Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

ADVANCED 3D VISUALIZATION FOR SIMULATION USING GAME TECHNOLOGY

Jonatan L. Bijl
Csaba A. Boer

TBA B.V.
Karrepad 2a
2623 AP
Delft, Netherlands

ABSTRACT

3D visualization is becoming increasingly popular for discrete event simulation. However, the 3D visualization of many of the commercial off the shelf simulation packages is not up to date with the quickly developing area of computer graphics. In this paper we present an advanced 3D visualization tool, which uses game technology. The tool is especially fit for discrete event simulation, it is easily configurable, and it can be kept up to date with modern computer graphics techniques. The tool has been used in several container terminal simulation projects. From a survey under simulation experts and our experience with the visualization tool, we concluded that realism is important for some of the purposes of visualization, and the use of game technology can help to achieve this goal.

1 INTRODUCTION

Simulation is the process of designing a model of a concrete system and conducting experiments with this model in order to understand the behavior of the concrete system and/or evaluate various strategies for the operation of the system (Shannon 1975). Although the simulation experiment can produce a large amount of data, the visualization of a simulated system provides a more complete understanding of its behavior. In the visualization, key elements of the system are represented on the screen by icons that dynamically change position, color and shape as the simulation evolves through time (Law and Kelton 2000).

There are several purposes for visualization, which can be categorized as *validation*, *analysis* and *marketing* (Bijl 2009a). Balci categorizes visualization as a dynamic validation technique in his taxonomy of validation techniques (Balci 1997). Displaying graphical images of dynamic behavior of a model during execution enables the user to discover errors by seeing. Following the animation of the model as it is running and comparing it with real life operation helps the user to identify discrepancies between the model and the system. Analysis refers to the process of gathering information and drawing conclusions from the simulation. Where statistics can show that a production line is inefficient, visualization can be used to locate the bottlenecks. The third term, marketing, refers to increasing the confidence of people in the provided solution. This means both convincing companies to buy the software, and convincing non-engineers of the validity of the result.

When simulation is used for training operational personnel, 3D visualization plays a very important role. For example, simulations are used in the military to train the soldiers without exposing them to any physical danger. Instead, the participants are seated next to a computer or inside tank or fighter simulators. Typically these training simulations are not intended to be a 100% solution for unit training but rather 70% solution with the remaining 30% requiring soldiers to get into real tanks in the field (Singhal and Zyda 1999). Simulation training is also applied in other fields than defense, such as the aviation industry or medical simulation training (Kincaid, Hamilton, Tarr, and Sangani 2003). Defense is a pioneer in the use of 3D videogame technology for training (Zyda 2005). Nowadays, the other fields also use videogame technology extensively for their training simulations.(Pulley 2007)

Although in the area of the COTS simulation packages the use of 3D visualization is increasing, these visualizations are not realistic. For validation and analysis, the provided 3D animation might suffice, but for the purposes of marketing and training, more realism might be required. In this paper the need for realistic 3D visualization is researched, and the role that the videogame technology can play in the implementation. This is illustrated using the 3D visualization component that TBA developed as example. The paper is organized as follows. Section 2 presents an overview of 3D visualization support of some major COTS simulation packages. In Section 3 the technology provided by game engines is described, and the expected usefulness of this technology is investigated through a survey. Section 4 describes how TBA developed an advanced 3D visualization component by making use of these opportunities. Then in Section 5 this component is evaluated in relation to effect of the use of a game engine. Finally in Section 6 the conclusions are drawn.

2 USE OF 3D VISUALIZATION IN COTS SIMULATION PACKAGES

In this section we present an overview of the 3D visualization support of some major COTS simulation packages in a nutshell. The selected simulation packages are taken from the Swain list (Swain 2009), which is the most complete list containing almost all the commercial simulation packages. A purposeful selection is made, focusing the packages that have the most support for 3D visualization.

All the selected packages are analyzed concerning two aspects: (1) the way in which the visualization is used and (2) the clarity and realism of the visualization. In order to describe how the visualization is used, several of the criteria from (Strassburger et al. 2005) were taken into account. The clarity and realism of the visualization are measured by classifying them with strong and weak points.

2.1 AnyLogic

AnyLogic is a Java-based simulation tool that supports several simulation formalisms. Simulations are created using state machines, action charts and events. For each component in the simulation a 3D object can be selected to represent it. A library with standard 3D objects is provided, but it is also possible to import them from other sources. It is also possible to add paths and other primitive shapes in the 2D and 3D animation. With AnyLogic, the graphs that define the logic of the simulation are separated from the specification of the layout of the visualization. In this way the graphs for the logic can be structured in a clear and orderly way, and at the same time the animation can be laid out as cluttered or organized as it is in reality. Apart from the 2D or 3D animation, the user can also design the layout of how the simulation should be shown. He can also add statistic graphs and customizable settings to this layout. In this way the user can make sure that during the simulation run, he has a good overview of the progress and results of the simulation.

2.2 Arena

Arena is a package in which 3D animation is mainly used for presentation purposes (Rockwell Automation 2010). When building or running the simulation there is no 3D animation, the user has to do with simple 2D animation. For the 3D animation there is a separate 3D player program, which comes with the enterprise edition of Arena. In order to produce 3D animation with Arena, there are several steps to take. First a simulation model must be created and run. Then, the user can set up a 3D environment in the 3D player, and configure how the different aspects of the simulation should be shown in 3D. Finally the user can view the 3D animation, and save it as a movie.

2.3 Enterprise Dynamics

For Enterprise Dynamics, the 3D animation is one of the main advertised features of the simulation software. The simulation itself is created by drawing diagrams in 2D. For all the simulation nodes in the simulation setup, a 3D counterpart can be defined. This can either be a 3D model, or a shape consisting of primitives. This appearance is defined using a simple scripting language.

2.4 FlexSim

FlexSim has integrated 3D animation into the core of its system. It has a 3D environment, which is used for constructing the simulation, laying out the animation and viewing the simulation run. The nodes of the simulation are represented by 3D models, which can be placed in the 3D environment. Then they can be connected to the other nodes, and their appearance can be modified if necessary. The 3D model that is used for an entity or a node can be selected from the 3D model library that is supplied by FlexSim, but custom 3D models can also be imported. When the simulation is started, this 3D environment comes to life and shows the progress of the simulation. During the simulation run it is still possible to select objects and edit their properties.

2.5 Simio

Creating a simulation model in Simio is quite similar to the process in Enterprise Dynamics, except that the user interface is easier to use. The functionality of the simulation is drawn with 2D nodes, the specific parameters are set in a dialog, and adding 3D animation is done by selecting a 3D model for each node in the 2D view. In addition, Simio comes with a very nice extra feature: it seamlessly integrates with Google warehouse, a huge online repository of 3D models. This makes it much easier to add custom 3D shapes.

2.6 Comparison

To classify the use of visualization three criteria from (Strassburger, Schulze, Lemessi, and Rehn 2005) are used. The first one is *Temporal Parallelism*. This tells whether the visualization runs concurrent with the simulation run, or the visualization is done afterwards. The second criterion, *Interaction*, specifies whether a model can be changed via its visualization, e.g. clicking on an object and then editing the properties. The third criterion is *Visualization Tool Autonomy*, in which it is specified to what extent the visualization tool is dependent on and integrated in the simulation tool.

Table 1 provides an overview of the classification of the three criteria for the discussed simulation packages. Concerning the appearance of the visualizations, the examined simulation packages are all quite similar. The appearance is appropriate to give an impression on the workings of the simulation. In some packages (Enterprise Dynamics, FlexSim, Simio) it is possible to add 3D text in the visualization, to show some figures like the throughput of a processing unit. However, this is done in 3D and therefore is only readable from a top view. AnyLogic provides a way of showing these figures in 2D instead, while simultaneously showing the 3D animation.

Table 1: Comparison of the capabilities of the visualization of several simulation packages

	AnyLogic	Arena	ED	FlexSim	Simio
Concurrent	$\surd$	-	$\surd$	$\surd$	$\surd$
Interactive	-	-	$\surd$	$\surd$	$\surd$
Integrated	-	-	-	$\surd$	-

In none of the tested packages the animation is realistic. However, it is realistic enough for validation and analysis and is also usable for presentation and marketing purposes. Nevertheless, it might be possible to raise the quality of the visualization to a higher level by looking at recent improvements in other areas of software development, like the game industry.

3 OPPORTUNITIES PROVIDED BY GAME ENGINES

There are many areas in software industry where real-time 3D visualization is being used. One of the most advanced is the 3D game industry. The game industry is a multi-million dollar industry (Vancouver Film School 2010), striving to use the full potential of home computers, game consoles and handheld computers. Due to the complexity of computer games, and the specialized knowledge that is required in several different domain areas, games are often created in a modular way (Blow 2004). The heart of most 3D games is the *game engine*, which provides the main functionality of the game. It consists of several modules that take care of 3D graphics, artificial intelligence, memory management, networking, audio, level loading and other tasks that are required in games. This engine is designed in such a way that the whole engine, or parts of it, can be re-used in other games. In this chapter we will investigate the cutting edge 3D animation technology that is being used in the 3D graphics modules - the rendering engines - of modern game engines. First, an overview of the features is given. Then, we evaluate these features on their usefulness in visualization.

The features that are described in this section are selected from the feature documents (Unigine Corp. 2011) and (Epic Games 2011b). These features will be discussed in three categories: features that improve the *performance*, the *appearance* and the *usability*.

3.1 Improving the Rendering Performance

In order to increase the rendering performance, game developers identify bottlenecks in the execution of the game. These can be bottlenecks in processing power, memory usage, and data transmission. Then, the optimization consists of either reducing the load, or shifting it to a less burdened part of the system.

Level of Detail is a technique to reduce the amount of polygons that need to be drawn. For each object, a detailed 3D model and one or more simplified models are supplied. Objects that are close to the camera are shown using the detailed 3D model. Objects that are further away are represented by one of the simplified 3D models. However, it requires several versions of each 3D model to be supplied.

Clipping and Culling are also techniques to reduce the amount of polygons. They mostly consist of simple and fast checks which can decide whether polygons are out of view. These techniques are so simple and effective that they almost always give an improvement.

Batching is a technique to optimize the rendering of huge amounts of non-moving objects. For a graphics card it is faster to draw one complex object than thousands of simple objects, because switching between objects and materials takes time. Batching can be used to group objects together into one bigger object, so that fewer switches need to be done.

Instancing not only tries to reduce the amount of material- and object switches, but also tries to reduce the amount of data that needs to be sent to the graphics card. Traditionally, if a 3D model would appear hundreds of times in a scene, it would also be sent to the graphics card hundreds of times. If the amount of objects or the complexity of the 3D model is very high, this means a lot of data needs to be sent to the graphics card. Instancing re-uses the 3D model so that it only needs to be sent once, and also groups them by material to reduce the amount of material switches.

3.2 Improving the Appearance

In computer games, a lot of effort is made to create convincing graphics. This is often done by using tricks and relatively simple calculations to mimic the real world.

There are several ways in which the surfaces of the objects are made more realistic. The most basic of them is the use of textures. Textures are pictures or photos which are applied onto a 3D shape. In modern game engines, each shape uses several textures, specifying different properties of the surface. The shininess, color, relief can be set with *specular*, *diffuse* and *normal* maps. A *reflection* can be rendered by first drawing the environment of an object, and then projecting it on the object itself.

Shadows play an important role in the realism of graphics. Not only because they make the lighting of the scene more believable but also because they increase the depth perception. Through the years several approaches for rendering shadows have been developed. The more recent approaches heavily depend on graphics card functionality, and are able to draw shadows at realtime speeds.

One of the lighting-related issues that is less noticeable, but that still has an important contribution to the realism, is *ambient occlusion*. In the real world, small spaces and concave corners receive less light, and therefore appear darker. Ambient occlusion uses some quick algorithms to detect most of these areas, and darkens them accordingly.

There are also ways to emphasize light. Light beams - also referred to as *God rays* - can emphasize bundles of bright light shining into relatively darker places. *bloom* adds a glowing effect to neon signs, traffic lights and other light sources.

Computer games can also mimic some different kinds of weather, for example fog and rain, clouds moving across the sky, and the rising and setting of the sun.

Effects caused by the lens of a camera, or the lens of a human eye, can also be reproduced. With *depth of field* the objects that are further away can be shown out of focus, and *lens flares* can be drawn when the camera is looking into the sun.

Modern game engines also contain ways of animation. *Skeletal animation* can be used to make people walk and to make truck wheels turn. Small simulations can be used to achieve some effects. *Cloth simulation* allows capes and flags to move along with the movements of the objects they are attached to. *Particle simulations* apply several simple rules to define the behavior of lots of simple objects - the particles. Together these give the impression of a fire, smoke, explosions, clouds or snow. Advanced simulations are used to display flocks of creatures - bees, crabs, birds, fish. These simulations have no important function in the logic of the game, but are essential to give a more lively impression of the virtual world.

Even though the purpose of all these effects is to mimic the real world, that does not ensure a realistic visualization. As stated in (Wages, Grünvogel, and Grützmacher 2004), a visualization that is very realistic on some aspects, might actually draw the attention of the user to the less realistic aspects, making the visualization as a whole less believable. So, even with all these effects at our disposal, it is necessary to make wise decisions about their use.

3.3 Improving the Usability

In a computer game, the story and the action are supposed to be the main focus. Therefore character and camera control are made as unobtrusive and intuitive as possible. Many games have an automatic camera control system, in which the camera movement is decided based on the actions of the player and the surroundings in the game (Christie, Olivier, and Normand 2008). When big objects block the view, like trees or buildings, the control system either automatically moves the camera, so that everything can be clearly seen, or makes the objects transparent, so that the camera can look through them. There are also semi-automatic camera control systems, in which the user can control the camera with simple mouse actions or keys.

The usability of a game engine does not only concern in-game usage, but also the development process. In well-designed game engines programmers are not the only ones able to configure or tweak the game. Almost nothing is hard-coded in the game engine. Instead, the settings can be configured using configuration files and scripts.

Editing tools are often provided with the game engine. There are drag-and-drop *level editors*, which can be used to easily build a game level, without any expertise in the area of 3D programming. Atmospheric effects and lighting can be configured, objects can be placed, a landscape can be drawn. It is possible to import 3D objects and characters from popular 3D drawing programs, and convert them to the game engine's format.

3.4 Usefulness of Game Engine Features in Simulations

Now that an overview has been given of the features of current game engines, their usefulness for simulations needs to be evaluated. The three categories are evaluated separately.

Whether improvements in the rendering *performance* are needed, mainly depends on the required complexity and detail of the visualization, and the hardware on which it is run. Optimization is needed when the frame rate of the visualization becomes noticeably slow, or the memory or processor usage is too high.

In order to quantify the usefulness of the 3D effects that improve the *appearance*, a survey has been held under nine simulation experts. They were selected both from simulation industry (TBA) and from academics (Delft University of Technology). For each of the three categories of use (validation, analysis and marketing) the participants were asked to grade the importance of every 3D effect. The survey is described in more detail in (Bijl 2009a). The results are shown in Table 2. From this survey it can be concluded

Table 2: Results of the survey on usefulness of 3D rendering features for visualization, ranging from 0 (useless) to 10 (must-have). The values are rounded for better readability of the table. The average is calculated based on the non-rounded values.

	Validation	Analysis	Marketing	Average
Textures	5	4	9	6.3
Normal maps	1	1	6	2.6
Reflection	1	1	4	1.9
Lights	3	3	8	4.6
Shadows	3	2	8	4.5
Amb. Occl.	1	1	4	1.9
Light beams	1	1	4	1.9
Bloom	0	0	2	0.7
Clouds	0	0	5	1.9
Fog	1	1	4	2.1
Rain	2	2	6	3.4
Sunrise, sunset	4	3	6	4.4
Lens flare	0	0	1	0.4
Depth of field	1	1	4	1.8
Skeletal animation	3	2	6	3.7
Particle effects	2	2	4	2.3
Cloth simulation	1	1	4	2.3
Flocks of creatures	1	1	2	1.3

that many effects that are used for realism are considered unnecessary but nice to have for validation and analysis, but for the purpose of marketing many of them are deemed more important. However, some of the features, like the lens flare and the flocks of creatures, are not even considered important for marketing.

The *usability* features inside the game engine, e.g. the camera control, can only be reused if a similar functionality is requested for the visualization. The configuration possibilities of the game engine are also likely to be usable, as far as it concerns the required rendering quality. The file converters or exporters can also easily be used. But the more game-specific tools, such as the level editors, might not be usable because they are specifically designed for a certain type of game. In that case it would be better to develop a level editor that is especially for simulations.

4 A GAME ENGINE BASED 3D VISUALIZATION COMPONENT

In this section a 3D visualization component is described, that uses a game engine to draw the graphics. In the development of this component, we took advantage of several of the opportunities of game engines, presented in the previous section. It can view DEVS simulation runs in real-time, and can also play them back after the simulation is finished. It is even possible to fast-forward and rewind through finished simulation runs, and to view the simulation runs over a network or Internet connection. Simulated objects can be inspected by clicking on them in the 3D animation. The state of the simulated object at that point in time will be displayed. Furthermore, when a new type of object is added in the simulation, its visualization can be configured without any programming. Finally, the component can be used to record movies of the simulation run, for use in presentations or on websites.

In this section we describe the choice of the rendering engine and the design of the component. Then we discuss how TBA uses this visualization component.

4.1 Game Engine

TBA has been using 3D animation for its container terminal simulations since 2005. However, the used 3D library, Java3D, had some drawbacks. First, the library was not fast enough to render the hundreds of thousands of objects that were needed by TBA at an acceptable speed. Second, the development of the Java3D library has stopped in 2004, so the animation could not use the more recent animation techniques.

Given these reasons, it was decided to switch to a new 3D library. One of the criteria for choosing a new library was that it would eliminate these two issues. Another selection criterion was that it should be able to achieve as realistic graphics as possible for the huge and complex simulations. Because of the quick developments in the computer game industry, this was the first place to look.

Nowadays there is a quite broad choice of good open source and commercial game engines. The latter often are feature-rich, but they are also quite expensive. For most game engines a license has to be bought per developer, starting at $1500 (Epic Games 2011a), and for some engines also a fee has to be paid per sold product. In the first place, we decided to investigate the open source rendering engines instead.

As the main programming language used at TBA is Java, the first free option was jMonkeyEngine (jME), a pure Java rendering engine. This option was quite quickly discarded as the frame rate of the demos showed that jME did not have the performance that was required.

Four open source, C++ based game engines were found, namely Ogre3D, OpenSceneGraph, Crystalspace and Irrlicht. To find out whether these engines were up to the task, they were compared on the basis of their graphic capabilities, the quality of finished software using the engine, the quality of the code and the quality of the support by its community. A more in-depth documentation of this comparison can be found in (Bijl 2009b). Based on this comparison, it was concluded that Ogre3D was the engine that was most up to the task. There is however a huge difference between Ogre3D and the solutions provided by the commercial game engines. Ogre3D is more of a library than a ready-to-use tool. Quite some C++ programming is needed in order to use the features that the engine provides. Ogre3D itself does not supply a level editor, but a third party level editor for Ogre3D is available.

4.2 Design of the Visualization Component

4.2.1 Flexible Configuration

The 3D visualization component has been designed to be a flexible visualization tool. One of the main requirements was that the modeler should not need to program anything for the visualization. Adding a few lines of xml code and adding a 3D model should be enough. The configuration system is based on the concept that both simulation objects and 3D objects have properties. Thus, a relation can be defined between the two types of objects. In DEVS simulations the simulation objects have properties that describe the state of the object. For example, in a simulation an entity has a position, and a resource can have a position, a handling rate, and can be occupied or free. As far as the animation is concerned, there is no real difference between resources and entities. The only requirement is that there are properties that can be visualized. A 3D object also has properties, which define its appearance: a position, an orientation, a 3D model, a color or texture. By nature a DEVS simulation run consists of a series of state changes. When one of the properties of the simulation object changes, the corresponding change in appearance can be applied to the visualization.

The configuration file for the animation contains a section for every type of simulation object. Inside this section, three kinds of settings can be configured for this type of object. At first, there are properties that are constant during the run of the simulation. For example, a truck will always have the shape of a truck. Therefore it can be configured to use a 3D shape of a truck to visualize the entity truck. Second, there are properties that do not change during the simulation run, but are different in the initial state. A simulation might contain trucks for internal and external transportation. This can be configured based on the value of that property at the initialization of the truck. Because this property is not expected to change, there is no need to revisit the property. Third, there are properties that change during the simulation run, like the position of an entity. As soon as the position of a truck changes in the simulation, this change is also applied to the animation.

The second and third type of settings require a translation of simulation properties to animation properties. Due to the fact that there are many possible translations, this translation has to be done programmatically. Some more frequently used translations are already supplied as modules, but if a specific one is needed, a small module will need to be programmed to support that. For example, for the container terminal simulation a specific translation module was needed to deduce the container type and color from the container name and size.

4.2.2 Concurrency

In (Henriksen 1999) two kinds of 3D animation are shown: concurrent and post-run. There are several ways in which a 3D visualization component could be linked to a simulation in a concurrent way. It could be implemented, for example, according to one of the COTS Simulation Package interopability reference models (SISO 2010). However, these models are intended for the synchronization of simulations. An other approach could be using a TCP/IP connection, over which the state changes are transmitted from the simulation to the visualization (Fumarola, Seck, and Verbraeck 2010). However, we have chosen for a solution that not only provides concurrent visualization, but also post-run. This was achieved by using a database.

The simulator stores all the state changes, which were caused by events, in a database. In this way, a log of the simulation run is made, from which the data can be retrieved efficiently. The visualization component has its own clock, separate from the simulation clock. This is used to play the animation at a regular pace. The animation clock time is used to query the database for the newest updates, which are then applied. In this way it is also possible to play the animation at a slower speed, pause it, or even play it backwards.

4.2.3 Remote Animation

The database approach also makes it possible to view the animation from a different computer than the one on which the simulation runs. The simulation can run on a server, and several instances of the visualization component can be started on different computers, connecting to the server's database. This setup unfortunately has a problem for longer distances, because the delay between query and response causes a serious lag. This could be solved by implementing an asynchronous connection between the visualization component and the server, but this has not yet been necessary.

4.3 Application in Container Logistics

The visualization component has been applied in both simulation and emulation projects. To emulate a container terminal, TBA has developed CONTROLS (CONtainer TeRminal Optimized Logistics Simulation). This is a software application which aims to create a simulated virtual container terminal (Boer and Saanen 2008). It emulates the physical processes at a terminal (equipment behavior, driver behavior, operational scenarios - gate arrivals, train arrivals, vessel arrivals) and acts as a real terminal. It can be connected to a real terminal operating system, which treats the model as if it were the ”real world”. In this way, the model can be used to run operational scenarios, either just as they occurred in the past, or as configured by the user. Thus, CONTROLS can be used for:

- Realistic, comprehensive, safe and inexpensive *testing* of the terminal operating system for both new and existing sites
- Off-line *tuning* of terminal operating parameters, by forecasting simulation, replay animation, and long term projection simulation.
- Realistic, real-time *training*, in a completely operational setting, for individual operational staff or complete teams (the so-called ”human in the loop”).

The animation component is used in all projects and has been providing support in all three types of projects (testing, tuning and training). The realistic 3D animation, as seen in Figure 1, provided a great support, especially for training. Next to CONTROLS emulation projects, the visualization component has been used in several container simulation models created in the COTS simulation package eM-Plant (Tecnomatix). In these cases, the visualization component played an important role for validation and marketing purposes.

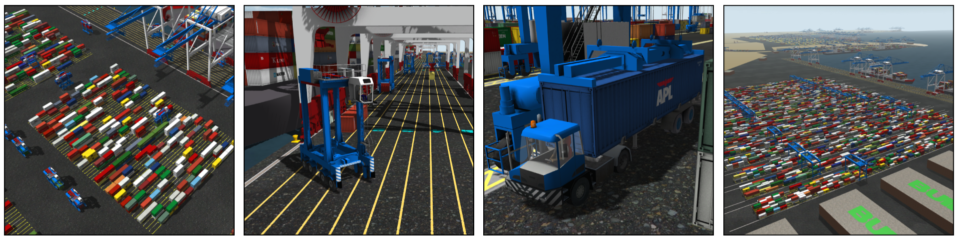

Figure 1: Screenshots of the 3D visualization provided by the new visualization component.

5 EVALUATION OF THE ADVANTAGES OF GAME TECHNOLOGY

The visualization component has been used in practice for more than a year. Therefore it is now possible to determine how useful the chosen approach was, especially the use of a game engine, and whether the approach can be recommended to the simulation industry.

One aspect of this evaluation is the comparison of the visualization component with the visualization in the COTS packages. To make a fair comparison, the 3D visualization component is categorized according to the criteria of (Strassburger et al. 2005), just as the other COTS packages. This is shown in Table 3.

Table 3: Categorization of the visualization of several simulation packages, and of the 3D visualization developed by TBA.

	AnyLogic	Arena	ED	FlexSim	Simio	3D visualization component
Concurrent/Post-process	√ -	- √	√ -	√ -	√ -	√ √
Interactive	-	-	√	√	√	√
Integrated	-	-	-	√	-	-

From the table we can conclude that the visualization component we built is not similar to any of the visualizations provided by the COTS packages. It differs from FlexSim as the 3D environment is not integrated at all with the creation of simulations. Rather, it is a separate application that can be used to view the simulation. It does allow a form of interaction, but only for the purpose of inspecting objects, not for disabling or editing objects. That is because unlike Simio, FlexSim and Enterprise Dynamics there is no feedback from the animation back into the simulation. On the other hand, the visualization component is more flexible in the viewing of the simulations, as it allows the replaying of a finished simulation run. Because the functionality of the visualization component is so different from the visualization that is provided with the COTS packages, they can only be compared with respect to the quality of the animation. In the other areas, the usefulness of the game engine features can be evaluated by itself, or by comparison with its Java3D-based predecessor.

In the area of performance, the techniques provided by Ogre3D were very important. In the old Java3D-based animation, which did not look realistic at all, some tricks had to be done to make it possible to be able to show tens of thousands of containers (Bijl and Fresen 2006). But due to the batching and instancing that were supplied by Ogre3D, it has not been necessary yet to implement such a trick, even though the amount of containers has increased to more than 100 000.

When comparing screenshots of the new visualization with the old Java3D visualization (Bijl 2009b) it is clear that the realism has improved in many aspects. A similar conclusion can be drawn when comparing it with the visualization of the COTS packages. In order to show more clearly how this improvement is achieved, the feature list from Table 2 is used again. In Table 4 these features are listed, showing to what extent they are supported by the different COTS packages and by the visualization component.

Table 4: Comparison of the 3D animation features supplied by the COTS packages, and by the visualization component. In the last column, features marked with ± are not yet supported in the visualization, but are supported in Ogre3D. Features marked with - are not supported by Ogre at the time of writing.

	Avg. Usefulness	AnyLogic	Arena	Ent. Dyn.	FlexSim	Simio	Controls
Textures	6.3	√	√	√	√	√	√
Normal maps	2.6	-	-	-	-	-	√
Reflection	1.9	-	-	-	-	-	±
Lights	4.6	?	?	√	-	-	√
Shadows	4.5	-	-	-	-	-	√
Amb. Occl.	1.9	-	-	-	-	-	√
Light beams	1.9	-	-	-	-	-	-
Bloom	0.7	-	-	-	-	-	√
Clouds	1.9	-	-	-	-	-	±
Fog	2.1	-	-	-	-	-	±
Rain	3.4	-	-	-	-	-	-
Sunrise, sunset	4.4	-	-	-	-	-	±
Lens flare	0.4	-	-	-	-	-	-
Depth of field	1.8	-	-	-	-	-	-
Skeletal animation	3.7	?	?	√	√	-	±
Particle effects	2.3	?	-	-	-	-	±
Cloth simulation	2.3	-	-	-	-	-	-
Flocks of creatures	1.3	-	-	-	-	-	-

It can be concluded that due to the use of Ogre3D, the amount of graphical features in the visualization component is much higher. Several of the features that were considered important in the survey are now supported. However, there are some features are supported by Ogre3D, but they are not yet supported by the visualization. This is because Ogre3D is designed more as a 3D library than as a finished product, and some extra programming work is required to use the feature in the visualization.

To make the system easy to use, we learned lessons from the game industry. For instance, in order to control the camera it was not possible to use a standard camera control mechanism from Ogre3D. However, a camera controller was developed that was inspired on the camera control in realtime strategy games. Also in the area of setting up the visualization, the approach of the game industry served more as an example than that Ogre3D's solution could be used directly. The use of configuration files and the file format for the terminal layout are based on Ogre3D's approach, but have been adjusted for the specific visualization needs. TBA's container terminal layout editor has been adjusted so that it also exports maps for Ogre3D.

6 CONCLUSION

The questions we investigate in this paper are whether and how the realistic aspect of 3D visualization in simulation packages can be improved. While 10 years ago almost all simulation packages had 2D visualization, nowadays most of the packages include 3D visualization features. However, the applied visualization techniques do not provide a *realistic* representation of the simulated world. In order to improve the quality of 3D visualizations, we propose to investigate solutions from the video game industry. The game industry is a multi-million dollar industry, striving to achieve the highest possible level of realism in a virtual environment. The modern computer graphics techniques used by these games are implemented in several open source and commercial game engines therefore they can be reused in other areas. In this paper we provided an approach for a generic 3D visualization component that uses such a game engine and has been applied in different container terminal simulation projects. As we illustrated with our approach, the technology is available and the realization is possible. Simulation package vendors are, however, still confronted with the question: *Is such a high level realism in 3D visualization really necessary?* The level of reality of the currently applied 3D visualization techniques probably suffices for many of the simulation purposes for which these packages are used, such as statistical or cost analysis. However, as illustrated in this paper, there are certain purposes, such as training emulations and marketing, where the level of realism is important. For instance, we have learned from our emulation projects that realistic 3D visualization was indispensable when the simulation model was used for training purposes. As in the coming years simulation packages will continue to compete, they will be improved in several aspects. One of the aspects that simulation package vendors can use to distinguish themselves from their competitors certainly is going to be the quality of 3D visualization. When moving towards a realistic 3D visualization we highly recommend the simulation package vendors to take advantage of the quickly developing game industry, instead of reinventing the wheel of 3D visualization.

REFERENCES

Balci, O. 1997. *Handbook of Simulation. Principles, Methodology, Advances, Application and Practice*, Chapter Verification, Validation and Testing, 335–393. John Wiley and Sons.

Bijl, J. L. 2009a. “How Game Technology can be used to improve Simulations”. literature survey, Delft University of Technology.

Bijl, J. L. 2009b. “Improving the visualization of 3D simulations using Computer Game technology”. Master's thesis, Delft University of Technology.

Bijl, J. L., and J. Fresen. 2006. “Improving 3D animation for Controls2”. Bachelor's thesis, Delft University of Technology.

Blow, J. 2004, February. “Game Development: Harder Than You Think”. *Queue* 1:28–37.

Boer, C. A., and Y. Saanen. 2008, December. “Controls: emulation to improve the performance of container terminals”. In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Moench, O. Rose, T. Jefferson, and J. W. Fowler, WSC '08, 2639–2647. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Christie, M., P. Olivier, and J.-M. Normand. 2008. “Camera Control in Computer Graphics”. In *Computer Graphics Forum*.

Epic Games 2011a. “Unreal Development Kit licensing”. Accessed Mar. 1, 2011. http://www.udk.com/licensing.

Epic Games 2011b. “The Unreal Engine renderer”. Accessed Mar. 1, 2011. http://www.unreal.com/features.php?ref=rendering.

Fumarola, M., M. Seck, and A. Verbraeck. 2010, December. “An approach for loosely coupled discrete event simulation models and animation components.”. In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 2161–2170. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Henriksen, J. O. 1999, December. “General-purpose concurrent and post-processed animation with Proof”. In *Proceedings of the 1999 Winter Simulation Conference*, edited by P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. Evans, WSC ’99, 176–181. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Kincaid, J. P., R. Hamilton, R. W. Tarr, and H. Sangani. 2003. *Simulation in education and training*, Chapter 19, 437–456. Norwell, MA, USA: Kluwer Academic Publishers.

Law, A. M., and W. D. Kelton. 2000. *Simulation Modeling and Analysis*. 3 ed. McGraw-Hill.

Pulley, J. 2007, April. “Serious Games”. In *Government Health IT*, Volume 2. 1105 Media.

Rockwell Automation 2010, March. “Arena 3D Player Product Profile”.

Shannon, R. 1975. *Systems simulation: the art and science*. Prentice Hall.

Singhal, S., and M. Zyda. 1999. *Networked virtual environments: design and implementation*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

SISO 2010, 9. March. “SISO-STD-006-2010: Standard for Commercial-off-the-shelf Simulation Package Interopability Reference Models”. Technical report, Simulation Interoperability Standards Organization.

Strassburger, S., T. Schulze, M. Lemessi, and G. D. Rehn. 2005, December. “Temporally parallel coupling of discrete simulation systems with virtual reality systems”. In *Proceedings of the 2005 Winter Simulation Conference*, edited by M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, WSC ’05, 1949–1957. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Swain, J. J. 2009, October. “Simulation Software Survey”. *OR/MS Today* 36 (5).

Unigine Corp. 2011. “Feature List of Unigine”. Accessed Mar. 1, 2011. http://unigine.com/products/unigine/features/.

Vancouver Film School 2010. “The Game Industry, Now and in the Future 2010”.

Wages, R., S. M. Grünvogel, and B. Grützmacher. 2004. “How Realistic is Realism? Considerations on the Aesthetics of Computer Games”. In *ICEC 2004, 3rd international conference on Entertainment Computing*, edited by M. Rautenberg, 216–225.

Zyda, M. 2005, September. “From visual simulation to virtual reality to games”. *Computer* 38 (9): 25–32.

AUTHOR BIOGRAPHIES

JONATAN L. BIJL is a software developer at TBA, where he has developed the Ogre3D based visualization tool. He received his BSc in Computer Science in 2006, and his MSc in Computer Graphics in 2009, both at the Delft University of Technology. His research specializes on the use of 3D rendering techniques for non-entertainment purposes, like simulation visualization and augmented reality. His email address is jonatan.bijl@tba.nl.

CSABA A. BOER is a senior product manager within TBA. He is responsible for several products within the organization, one of which is CONTROLS. He holds a Ph.D. in Computer Science and Logistics from Erasmus University Rotterdam. His research interests include distributed simulation, distributed virtual environments, port logistics, and port simulation and emulation. His email address is csaba.boer@tba.nl.