# Using Game Engines for Visualization in Scientific Applications

Karl-Ingo Friese, Marc Herrlich, and Franz-Erich Wolter

**Abstract** In recent years, the computer gaming industry has become a large and important market and impressive amounts of money are spent on the development of new game engines. In contrast to their development costs, the price for the final product is very low compared to a professional 3D visualization/animation program. The idea to use this potential for other purposes than gaming seems obvious. This work gives a review on three *Serious Gaming* projects, analyzes the encountered problems in a greater context and reflects the pros and cons of using game engines for scientific applications in general.

## 1 Introduction

In 2002 the University of Hannover held its yearly open house day and our department had a small presentation as well. One of the projects we showed was a diploma thesis, showing caves, i.e. former mines, reconstructed from laser scan data. The focus of the thesis was the reconstruction itself, not the visualization, aiming for engineering post processing. Still, it was possible to show the reconstructed cave walls in some way (figure 1).

At that open day we had a visitor who was an archaeologist. She was very interested in the caves, since they had just discovered a new cave with bones and wall paintings in a nearby mountain area, which was inaccessible for a larger public. When we told her that our programs could show only the *outside* of a cave, without the possibility to *walk through it* she was a bit disappointed.

Karl-Ingo Friese, e-mail: kif@gdv.uni-hannover.de
Franz-Erich Wolter, e-mail: few@gdv.uni-hannover.de
Institute of Man-Machine-Communication, Leibniz Universität Hannover, Germany

Marc Herrlich, e-mail: mh@tzi.de
Research Group Digital Media, Universität Bremen, Germany

The idea to visualize these caves also from the *inside* was born. The goal was to write a small application that would allow a more natural form of showing our caves. What we wanted was an application, which would allow to run in a first-person-perspective through a cave-like virtual dungeon and it was obvious that such applications already existed in the form of computer games or to be more precise: first-person-shooters.

Compared to professional scientific visualization software, computer games have to favor real-time rendering over physical correctness and data accuracy. We took the challenge to see if we could visualize our reconstructed cave with a 3D computer game, which, after solving some problems at the beginning, turned out to be possible.

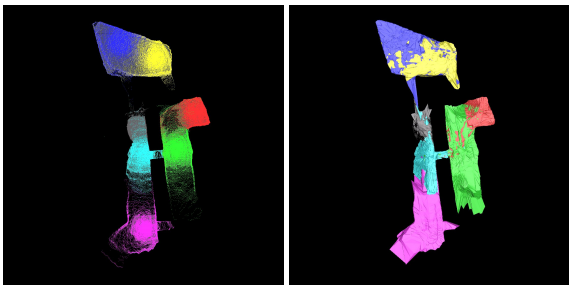

**Fig. 1** A (reconstructed) cave with three chambers, seen from the outside. The different colors represent the different laser measurements.

Since then, several projects using computer games to visualize scientific data followed. This paper gives an overview of three of them, classify them in the context of *Serious Gaming* and finally reflect if and when the use of game engines in scientific applications can be useful.

## 2 What has been done?

The terms *Serious Games* and *Serious Gaming* are used in the literature to describe very different application scenarios. While the concept of *Serious Games* is originally stemming from the area of game-based learning and education [1, 17], today it is more generally used to describe a whole spectrum of applications [14]. In this sense, the area of *Serious Games* incorporates all aspects of applying computer game technology to non-entertainment uses, including but not limited to simulation, visualization and VR. For this paper we are only considering 3D-games, not puzzles, 2D-shooters, etc. and will use the following definition of *Serious Games*:

**Definition 1.** Every application that makes significant use of game technology and is not primarily intended for pure entertainment is a *Serious Game*.

Historically, games were designed and implemented on a case-to-case basis, leaving only little room for easy modification and reuse. Therefore, the usefulness of

game technology for other application areas was somewhat limited. However, this changed with the appearance of the first modern game engines, e.g. the Quake or Unreal series, which provided better modularity. Today, developers can choose from a number of suitable commercial and non-commercial engines.

In this paper we will focus on the area of visualization using modern computer game engines. Before discussing our own research and results in the following sections, we will present a current overview of the ongoing research and development, focusing on indoor and outdoor GIS and CAD data visualization and interaction.

In 2002, Rhyne [13] argued that scientists today have much to learn from the computer games industry regarding computer graphics, visualization, and interfaces, especially with the background of cluster computing. In this area it is an ongoing trend to build clusters from standard PC hardware instead of specialized workstations or mainframes. In Rhyne's opinion, scientific visualization applications can benefit from computer game technology, as most computer games are optimized for commodity hardware. On the other hand, she states some drawbacks in using computer game engines, especially concerning data accuracy and reliability. She also mentions the traditionally short release cycles in the games industry, which may lead to incomplete or unstable graphics drivers.

While this is certainly true, these short release cycles also have their advantages. Namely the availability and support for new hardware and software features. Furthermore, as production costs for modern games are exploding, the development processes have matured and there is a specialization taking place that is dividing game companies into technology developers and technology users, which will lead to more robust solutions.

In the same year, Herwig and Paar [10] discussed the suitability of game engines for landscape visualization and planning. They presented different usage scenarios and analyzed the requirements of landscape architects concerning supporting tools and to what extend game engines can solve these problems. They also showed preliminary results of tests conducted with a landscape visualization based on the *Unreal Engine*. Their findings fit in very well with our own research in landscape visualization based on the CryEngine presented in section 3.3.

In the following, we will report on a number of visualization projects, which approximately fall into the same time span as our own projects described thereafter.

In 2001, Freudenberg et al. [6] described a low-cost VR installation powered by the Shark3D game engine. Using commodity hardware, i.e. three standard PCs and beamers, they employed the game engine's built-in rendering and networking features to create a distributed rendering system capable of driving the VR projection in real-time. Their system had enough power reserves to render pre-distorted images to compensate for the spherical projection plane. Opposed to expensive off-the-shelf VR solutions, the game-based solution clearly demonstrated the advantage of being able to use standard commodity hardware.

In 2002, Shiratuddin and Thabet [15] described the implementation of a virtual office walkthrough system based on the Unreal Engine. They derived the geometry from 2D CAD data importing it into the engine. Furthermore they used cheap input devices like the Microsoft Sidewinder Freestyle Pro gamepad in conjunction with

the engine's real-time capabilities to allow 6-DOF real-time interaction in a photo-realistic environment. Most of the data conversion was either executed manually or using commercial tools. In our own research, presented in the following sections, we tried to automate the process as much as possible.

Germanchis et al. [8, 9] explored 2004 the potential of game technology for the visualization of geographical data in the context of human path finding and spatial cognition research.

The visual quality and the level of interaction provided by modern game engines had certainly reached a level making them suitable for research in the area of human cognition. The authors used a full set of professional commercial tools, e.g. ArcGIS, to prepare the data for the game engine. This again contrasts our own (semi-)automatic approaches.

Fritsch and Kada [7] discussed 2004 indoor as well as outdoor visualization of geographical data based on different game engines, among them the Quake3 Engine and the Unreal Engine 2. They also discussed the benefits of game engines compared to other software libraries and presented concepts for integrating them with other software packages for different purposes, e.g. Computer Aided Facility Management-Systems. They came to the conclusion that the conversion process of geographical data into the data format of the game engine is one of the major obstacles for every game engine based application.

Arendash [2] demonstrated 2004 how the Unreal editor could be exploited as an intuitive authoring tool for web-based virtual worlds, i.e. VRML or X3D based virtual worlds. He presented a tool to extract geometry, texturing, and lighting data from the Unreal data format into a valid VRML/X3D representation.

Also in 2004, Lepouras and Vassilakis [12] presented the concept of building virtual museums by using a game engine. This virtual exhibition space took advantage of the high visual quality of modern game engines. Lepouras and Vassilakis also conducted a user acceptance study of their virtual museum prototype, which showed very promising results.

In 2005, Jacobson and Lewis [11] presented an open source project derived from the Unreal Engine called CaveUT for immersive Cave-like virtual reality projection environments. In the same year, Stock et al. [16] demonstrated how the Torque Game Engine can be connected to a web-based map server to create an easy-to-use collaborative environment for landscape visualization and planning. They exploited not only the rendering capabilities, but especially the networking features present in most computer game engines.

In other works, game engines have also been used to provide visualization and interaction metaphors in completely different and more abstract areas, which are not listed here because they would go beyond the scope of this paper.

## 3 Projects at the Welfenlab

This section will report on three projects, realized with students of the Welfenlab. The first is a (very basic) visualization of caves with Quake3. A second approach with Unreal Tournament 2004 had the same goal. The third project focused on landscape visualization in planing processes with FarCry.

The base for the first two projects was a digital model of man-made caves, which we reconstructed before in a previous work. Its original goal was to receive a reconstruction as precise as possible, resulting in a highly detailed triangle surface, which could also be exported as a solid volume model for CAD applications like AutoCAD.

### *3.1 Cave Visualization with Quake3*

The first approach to visualize the reconstructed caves in a first person perspective used the (even at that time) rather old Quake3. The main reason for this was that Quake3 was well understood and available for Windows and Linux. Quake3 was produced and published by IdSoftware, released in December 1999 and supported shaders, curved surfaces, 32-bit color and of course hardware rendering. Only a single license (of the original game) was necessary for the presentation of the results.

The goal of this first project was not only to see if it is possible to visualize the cave, but also to analyze how difficult it would be to convert our high resolutional scientific data into the restricted surface representation of a computer game, without losing too many details. This work was done by Dominik Sarnow in his junior thesis. The choice of the game engine was based on the following criteria:

- the engine must support dynamic light computations
- it should have no license conflicts
- it should use an open file format that is easy to understand
- it should have at least some available documentation

Quake3 fulfilled all but the third criterion, because as almost all other game engines, the native Quake3 level format is binary, proprietary and far from being human readable. Fortunately, there was an easy solution for this: the existence of the level editor GtkRadiant.

GtkRadiant is a level editor for Quake3 and other games. It is free for non-commercial use and is available for several platforms. Its native map format is a sequence of numbered entities, the first entity always being the world which contains geometrical objects (brushes). An entity consists of a class name, an origin that defines the place of the entity in the map and texture/material properties.

Instead of writing a converter of our data into the binary format of Quake3, we decided to export into the text format of GtkRadiant. Still we had to deal with the specific restrictions of Quake3: a limited number of triangles (per level), a map

format that can only interpret convex objects and most obvious: all coordinates had to be of integer value.

Since the maximal number of triangles was limited, a prestep reducing the numbers of triangles was necessary. We used a simple shortest-edge removal procedure which was easy to implement and produced acceptable results.

The geometry supported by GtkRadiant consisted of planar and convex geometric entities, *cut out* of a plane. Since our original data was a triangulated surface, the natural approach seemed to be to turn each triangle into a map entity. However, this resulted in a huge number of entities, consequently we spent the extra effort of locating planar areas in our original surface and finding suitable compositions of convex polygons, reducing the number of entities significantly. This interesting geometrical problem was looked into more deeply in the bachelor's thesis of Daniela Lauer.

The problem of transferring vertex coordinates to integer was easy to solve: center the cave data, scale the coordinates (taking into account that the maximum supported integer value for a coordinate is 64k) and delete extra decimal places. Afterwards a correction step was necessary to remove degenerated triangles. Due to the nature of the internal geometry description, it was very important to create closed surfaces, because otherwise the number of polygons in the resulting level file exploded.

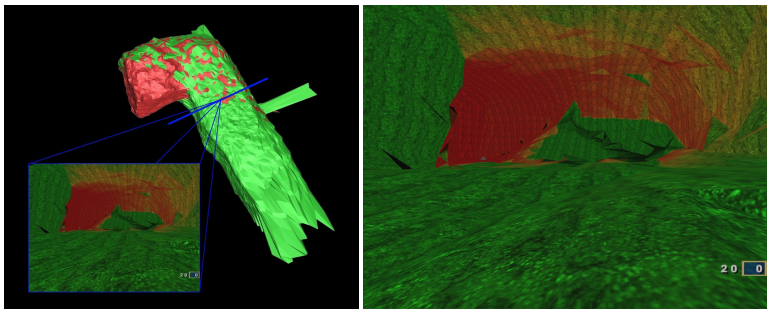

**Fig. 2** Walking through the reconstructed cave with Quake3

In the resulting *Quake3 level*, colored light sources were placed at the exact positions of the lasers during the measuring of the original caves, giving a very intuitive impression of the visibility region of each measurement, shown in figure 2.

## *3.2 Cave Visualization with UT 2004*

During the work with Quake3, we realized that the visual restrictions would be quite strong, resulting in a second project using the more modern game Unreal Tournament 2004 (UT 2004), leading to the bachelor's thesis of Michael Hanel.

In the UT 2004 project, we followed a similar approach: writing a converter that would not export to the (proprietary and binary) format of the game itself, but into the text format of its level editor, which is free for download.

The Unreal Engine 2 was designed for PC, Sony PlayStation2 and Microsoft XBox and runs with Microsoft Windows XP and Linux. It was used in Unreal Tournament 2004 and Unreal 2. It supports CSG (constructive solid geometry) and BSP (binary space partitioning) geometry, 12 steps of MIP-Mapping, static and dynamic light sources. The texture format is 32 bit with a resolution up to 2048 $\times$ 2048 pixels. The engine can show up to 150.000 triangles in view.

Contrary to UT 2004, which also runs on Linux, the map editor UnrealEd 3.0 is a Windows-based application. It can read and write two native data formats: *Unreal Tournament Map .ut2* (binary) and *Unreal Text Format .t3d* (plain). Within the editor, every object is represented by an *Actor*. *Actor* objects combine general and specific attributes, such as the object class, position, size, color, etc. The most important actor classes are *Brushes*, *TriggerLights* (Lights) and the *PlayerStart*.

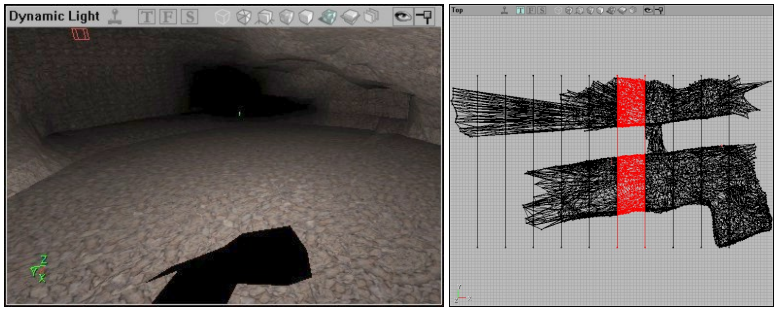


**Fig. 3** BSP Holes and their solution: segmenting the surface into several brushes

One of the biggest problems in this work was that the original data consisted of several thousand triangles for the cave surface, producing high computing costs for the engine. The first approach was to convert the whole cave as a single brush object, which turned out to be problematic, since the resulting level contained BSP holes. Therefore we had to reduce the number of triangles per brush. Experiments showed that 500 triangles per brush seemed to be an upper limit.

In the bachelor's thesis of Michael Hanel, this was solved by an equidistant segmentation of the original data into horizontal slices, illustrated in figure 3. These segments were converted into brushes that needed to be closed with side walls for the automated merging within the engine. The equidistant segmentation approach was not optimal, since it did not guarantee an upper limit of triangles per brush, but relatively easy to implement and chosen due to time constraints.

The restriction of 500 triangles per brush cannot be taken for granted, it just appeared to reduce the BSP holes (almost) to zero, which we found out experimentally. In all three projects, similar engine limitations turned out during development

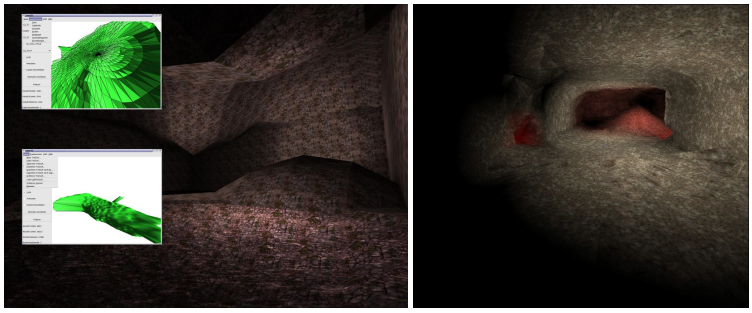

**Fig. 4** A screen shot showing the UT 2004 version of the cave in the level editor and the game itself

and had to be analyzed with trial-and-error methods. This seems to be a common problem for *Serious Gaming* projects, as described in section 4.

After this was done, the last remaining problem was the lack of a texture mapping method, that would provide the cave walls with a sufficiently realistic appearance. Together with dynamic lighting, the result was very convincing as seen in figure 4, giving people not involved in the project the immediate impression of a natural cave, much more than in the former Quake3 visualization.

### *3.3 Landscape Visualization with FarCry*

The third project we want to describe in this paper focuses on landscape visualization and planning (cf. [10]). Its main goal is to apply the visualization and interaction capabilities of modern game engines, in this case the CryEngine, to build the prototype of a visualization tool for landscape architects. Our goal was to provide a tool that generates a quick but photo-realistic visualization of an area based on real geographical data, allowing interactive movement through the landscape and real-time interactive modification of the terrain layout and vegetation placement.

One possible usage scenario could be a landscape architect, having a meeting with customers and trying to present his ideas and how they will transfer into reality. Many people have difficulties to imagine how the look of a landscape might change, e.g. with different arrangements of trees and other plants, therefore it is often crucial to provide images or models of the target outcome. This can be a very time consuming and expensive process. Under these circumstances the use of game engines might help to reduce the cost, when compared to professional solutions, while at the same time enhancing the visual quality of the final images. Game Engines also provide the opportunity for collaborative, interactive walkthroughs with no or very low additional cost.

A crucial requirement for a landscape planning tool is the possibility of fast modifications of the terrain and of the placement and arrangement of the vegetation and

other objects. This poses an additional challenge in comparison to 'simple' visualization tasks. In section 1 we argued that one major benefit of computer game technology is the optimization for commodity hardware. This was also a major concern in this project because in our scenario a landscape architect would have to be able to use his standard desktop or notebook computer to run our tool in front of his customers.

The CryEngine [4] is a commercial game engine developed by Crytek. It was first employed in the game Farcry, which we used for this project. The engine itself is accompanied by an editor tool called CryEngine Sandbox [3].

Our overall approach can be described as follows. In the first step, very similar to the Quake 2 and UT 2004 project, the geographical data is converted into a format that can be read by the CryEngine Sandbox. In the second step, the real-time editing features of the Sandbox are then exploited to perform any necessary modifications or rearrangements or to try out different landscape scenarios. Finally, a map is generated from within the Sandbox that can be used directly by the game, e.g. for collaborative exploration. It is important to note, that we try to automate the conversion process as much as possible to provide landscape architects with an easy to use tool. The key idea is, that the landscape architect only provides the basic data files and the visualization is then boot-strapped by the conversion tool.

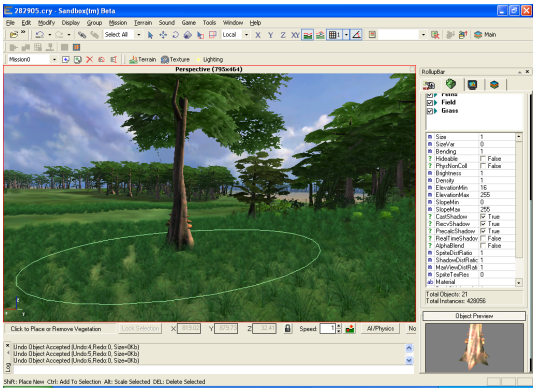

**Fig. 5** The CryEngine Sandbox [3] is automatically installed together with the game and free for non-commercial use. It provides real-time interactive tools for terrain shaping and vegetation placement.

We decided to employ the CryEngine because it supports very large outdoor terrains, naturally an important point in our application. The CryEngine Sandbox has also been a key factor. In contrast to many other editing tools, the Sandbox provides real-time interactive editing and a very comprehensive set of tools for terrain shaping and vegetation placement. It also supports seamless switching between in-game and editor modes (figure 5).

Our terrain visualization and the automatic placement of vegetation is based on two types of data. First, we need a digital elevation model (DEM), which describes the general shape of the landscape to a degree limited by the resolution of the available DEM. Second, we need a segmentation of the terrain according to types of vegetation present in the respective areas to be visualized. This segmentation usu-

ally decomposes the landscape into areas like forest, field, or meadow. For the purpose of storing the segmentation we used ESRI shape files [5], which is a standard data format commonly used in geographic information systems.

**Fig. 6** Ground Texture vs. Aerial Photography. The generated ground texture and the aerial photography of the same area match very well even though we used a coarse segmentation.

We have tested out prototype with sample data from certain areas in Lower Saxony and have received some very convincing results concerning realism and visual quality, shown in figure 6.

The biggest difficulties we encountered during the project were connected to the data conversion process. The resolution of the DEMs is in general very different compared to the internal heightmap resolution used in the game engine. Therefore this data needs to be resampled to be used by the Sandbox. The shape file data has to be matched and positioned correctly onto the terrain and of course it has to be clipped accordingly. Finally, the file formats used by the CryEngine are not publicly documented.

## 4 Reflection

In the last section, we presented three approaches to use existing computer game technology in scientific applications. The question that remains is: Was it a good idea or would it have been more suitable to use professional visualization software? The answer is as usual: *It depends*.

Using commercial computer games for a non-gaming context has huge advantages. First of all, they usually bring state-of-the-art graphics, often supporting a client/server concept which can be used for multi-user applications. Their most important advantage is of course the price: a single license usually does not cost more than 100 US$, while professional visualization tools easily cross 10,000 US$ per copy. Also, the professional software usually requires professional hardware, while computer games are designed to run on last year's low budget PC as well.

However, every advantage comes with a trade-off. The problems of the three projects from the last section seem to be exemplary for the field of *Serious Gaming* and can be divided into four categories:

1. **Lack of documentation**
   No matter how good a game engine and its editors are documented and how large its community is, it seems to be impossible to find out concrete numbers, e.g. the maximum number of polygons per object or the maximum file size of a level. The process of writing tools that convert scientific data into the file format of a game engine (or for its level editor) is usually very experimental.
2. **Engine-Dependent Restrictions**
   As seen in all three projects, every game engine had very specific restrictions, for example the 'integer-coordinates-only' drawback of Quake3 (section 3.1) or the maximum number of triangles per brushes in UT 2004 (section 3.2. These restrictions are usually not obvious before the development starts and result in time-expensive workarounds.
3. **Short Life-Span**
   Computer hardware evolves fast and a modern computer game usually lasts only a few years. As long as it is new, it is usually supported well, but compared to professional animation software, it is very unlikely that it will run on future operating systems or on hardware that will come out 3-4 years after its launch.
4. **Not Extendable**
   The application can do what the game can do. Nothing more, nothing less. Future customer requests might be expensive or impossible to implement.

So why is it still reasonable to continue the work with game engines? Because of their potential. Computer games are highly specialized but also highly optimized, with development costs matching those of Hollywood movies. These games are sold at a very reasonable price as they are produced for a mass market. The situation can be compared with the current run on GPU-Programming. It can be safely said that (ab)using the GPU for non-rendering purposes is a very unpleasant if not questionable way of writing programs. Yet the impressive speedup that is gained with additional hardware costs of zero (almost every computer already has a fast GPU) made it very popular.

If the scientific application matches the potential of a game engine close enough, as in the use of FarCry for visualizing landscape planning processes, the costs of developing software with similar capabilities would by far go beyond the costs of finding solutions for the engine restrictions or buying a professional software. As long as the original problem does not exceed the capabilities too much, it might always be worth a closer look. However, one should always keep the drawbacks mentioned above in mind.

## 5 Summary and Outlook

In this work we reported on three different visualization projects making use of 3D computer games and tried to classify them in the context of other *Serious Gaming* projects (sections 2 and 3). We have shown that the visualization of scientific data

with game engines is possible and leads to promising results. We also discussed its drawbacks (section 4) and tried to extract common problems of all three projects.

In the future we would like to extend the presented work. Firstly, we would like to incorporate the latest developments in the area of game technology, i.e. enhanced rendering methods and the like. Secondly, we would like to explore the full potential of game engines not only in graphical terms. We think there is a great potential in using the available artificial intelligence and networking capabilties of modern engines.

## References

1. Abt, C.C.: Serious Games. University Press of America (1987)
2. Arendash, D.: The unreal editor as a web 3d authoring environment. In: Proceedings of the ninth international conference on 3D Web technology, pp. 119–126. ACM, Monterey, California (2004)
3. Crytek GmbH: CryEngine Sandbox Far Cry Edition User Manual, 1.1 edn. (2004)
4. Crytek GmbH: Far Cry Engine Overview, 1.0 edn. (2005)
5. Environmental Systems Research Institute: ESRI Shapefile Technical Description (1998). White Paper
6. Freudenberg, B., Masuch, M., Röber, N., Strothotte, T.: The computer-visualistik-raum: Veritable and inexpensive presentation of a virtual reconstruction. VAST2001: Virtual Reality, Archaelogy, and Cultural Heritage (2001)
7. Fritsch, D., Kada, M.: Visualisation using game engines. Archiwum ISPRS **35** (2004)
8. Germanchis, T., Cartwright, W.: The potential to use games engines and games software to develop interactive, three-dimensional visualisations of geography. ICC Proceedings, Durban pp. 352–357 (2003)
9. Germanchis, T., Pettit, C., Cartwright, W.: Building a three-dimensional geospatial virtual environment on computer gaming technology: Geographic visualization. Journal of spatial science **49**, 89–95 (2004)
10. Herwig, A., Paar, P.: Game engines: Tools for landscape visualization and planning? Trends in GIS and Virtualization in Environmental Planning and Design (2002)
11. Jacobson, J., Lewis, M.: Game engine virtual reality with caveut. Computer **38**, 79–82 (2005)
12. Lepouras, G., Vassilakis, C.: Virtual museums for all: employing game technology for edutainment. Virtual Reality **8**, 96–106 (2004)
13. Rhyne, T.M.: Computer games and scientific visualization. Commun. ACM **45**, 40–44 (2002)
14. Sawyer, B.: Serious games: Broadening games impact beyond entertainment. Computer Graphics Forum **26**, xviii (2007)
15. Shiratuddin, M.F., Thabet, W.: Virtual office walkthrough using a 3d game engine. International Journal of Design Computing **4**, 1329–7147 (2002)
16. Stock, C., Bishop, I.D., O'Connor, A.: Generating virtual environments by linking spatial data processing with a gaming engine. Trends in Real-time Landscape Visualization and Participation, Proceedings at Anhalt University of Applied Sciences, Wichmann pp. 324–329 (2005)
17. Zyda, M.: From visual simulation to virtual reality to games. Computer **38**, 25–32 (2005)