Updated GameTools: Libraries for Easier Advanced Graphics in Serious Gaming

Rubén Jesús García, Jesús Gumbau, László Szirmay-Kalos and Mateu Sbert

Abstract Advanced graphics effects can be difficult to create and implement by nonexperts. We present here multiplatform, multigame engine libraries, designed to simplify the use of efficient state-of-the-art computer graphics algorithms in the fields of geometry and illumination. A summary of our experiences updating and porting the libraries is included, with recommendations possibly useful to other programmers. The main features of the updated libraries are described. Two serious games which use the geometry libraries are presented here as use cases, both dealing with teaching history to middle school students. Finally, we show an implementation of a board game played in Europe since roman times (of interest to museums specialized in medieval or roman times) which uses the illumination libraries.

Keywords Game engine · Serious game

R. J. García (✉)
Institute of Informatics and Applications, University of Girona, Girona, Spain
e-mail: rgarcia@ima.udg.edu

J. Gumbau
Universitat Jaume I, Castellón, Spain
e-mail: jgumbau@lsi.uji.es

L. Szirmay-Kalos
Budapest University of Technology and Economics, Budapest, Hungary
e-mail: szirmay@iit.bme.hu

M. Sbert
University of Girona, Girona, Spain
e-mail: mateu@ima.udg.edu

Y. Cai and S. L. Goei (eds.), *Simulations, Serious Games and Their Applications*, Gaming Media and Social Effects, DOI: 10.1007/978-981-4560-32-0_12,

1 Introduction

A serious game is a game designed for a primary purpose other than pure entertainment. As a consequence, the development team is often more centered on realistic modeling of the aspects of the virtual world which are required for the teaching of techniques which the user will later use in his professional life.

It is common that designers and programmers are experts in the application domain, rather than in the field of computer graphics (especially when comparing to firms creating entertainment games).

The creation of new graphic effects is becoming increasingly complex, due to two main reasons:

- New effects often require the use of advanced capabilities of Graphical Processing Units (GPUs), which use architectures quite different from that of CPUs, and require the knowledge of specialized languages (Cg, HLSL, GLSL, CUDA) and techniques (parallel architectures with distributed memory, specific geometry, vertex and pixel shaders, and different types of memory with tricky access modes).
- These effects are becoming increasingly supported by complex physics simulations, which require knowledge of methods to solve differential equations efficiently.

Therefore, the existence of a general library of graphic effects and techniques, which could be used in different projects by nonspecialists in the field of computer graphics, can be a great asset for researchers and programmers in the serious gaming community.

The next section describes previous work on which our library is based, while Sect. 3 describes the new libraries' porting procedure and implementation details. Section 4 contains three use cases: the Jaume I, Legends of Girona, and Nine Men's Morris games. Finally, Sect. 5 concludes the chapter.

2 GameTools

GameTools (2008b) was a European Union project which brought together six universities and four industrial partners during 2004–2008, and whose aim was to create a set of *graphic libraries* encompassing different aspects of computer graphics: geometry simplification, global illumination, and visibility culling.

The libraries were meant to aid game companies who did not have a large enough budget to maintain a specialized team in advanced computer graphics techniques. Companies were supposed to focus on the game story, ensuring playability, and user enjoyment, while the advanced graphic effects were provided by the GameTools libraries. GameTools libraries were provided for the windows platform, using visual studio 2005 and DirectX 8, with support for the Ogre3D (OGRE 2012) game engine, version 1.0.8, and the Shark3D (2012) engine.

2.1 Geometry Libraries

To create a virtual world, one of the tasks is modeling scenarios and characters using geometric primitives, usually triangles. These (usually complex) models require substantial processing power to render. In order to decrease computation time without sacrificing quality, the geometry of far-away objects can be transformed (simplified) so that the cost of rendering is significantly reduced. Since the simplified objects occupy a small area in the screen, imperfections are difficult to distinguish to the human eye. If the level of detail is changed continuously as the object moves, the change will not be noticed by users.

The Geometry libraries contain algorithms dealing with geometry processing in the Context of Continuous Level of Detail (CLOD). In particular, mesh simplification (González et al. 2007b), spherical light fields (Domingo et al. 2007), tree leaves (Rebollo et al. 2007a, b), and mesh management (Gumbau et al. 2007) are included. Error metrics taking textures into account are also used (González et al. 2007a).

An example of the software developed to manage simplification methods can be seen in Fig. 1.

2.2 Illumination Libraries

The illumination libraries contain an assortment of efficient algorithms for realistic rendering, based on the concepts of approximate raytracing (Szirmay-Kalos et al. 2005a, b; Umenhoffer et al. 2007; Lazányi and Szirmay–Kalos 2005) (Fig. 2), indirect illumination gathering (Méndez et al. 2005; Szécsi et al. 2006; Lazányi and Szirmay-Kalos 2006; Szirmay-Kalos and Lazányi 2006; Umenhoffer and Szirmay-Kalos 2007) (Fig. 3), scattering media simulation (Umenhoffer et al.

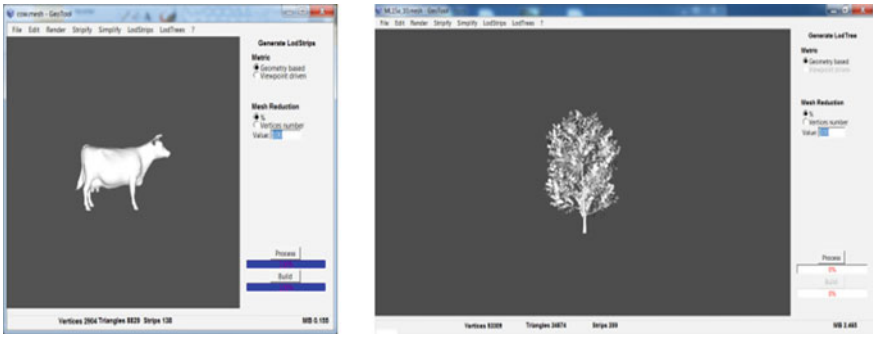

Fig. 1 Continuous level of detail: *triangle* strips for general models; *tailored* algorithms for vegetation

Fig. 2 Approximate raytracing: realistic transparency and caustics

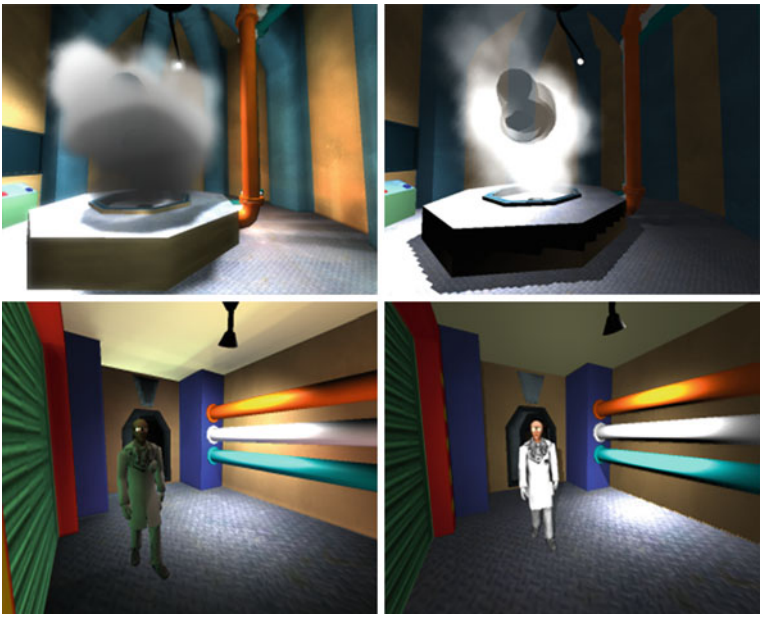

Fig. 3 Indirect illumination gathering (*left*) versus local illumination model (*right*)

2006; Umenhoffer and Szirmay-Kalos 2005, 2006) (Fig. 4), and postprocessing effects such as depth of field, tone mapping (Toth and Szirmay-Kalos 2007), glow (Fig. 5), and image distortion caused by heat shimmering (Fig. 4 left, and Fig. 6). The effects can be used together with no restrictions.

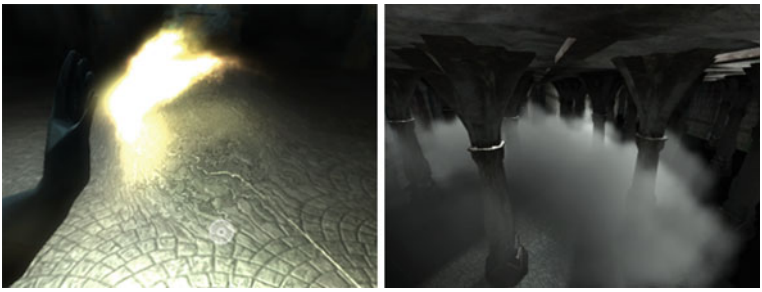

Fig. 4 Scattering media simulation: fire and smoke

Fig. 5 Postprocessing examples: depth of field, tone mapping

Fig. 6 Postprocessing examples: image distortion due to heat

Fig. 7 Potentially visible set (PVS) and coherent hierarchical culling (CHC)

2.3 Visibility Libraries

Given a scene that we wish to render, the visibility problem consists in calculating efficiently which geometry is (at least partially) visible from a viewpoint (Hadwiger and Varga 1999). If we have some information about the type of scene being rendered, we can sometimes apply specific algorithms to discard invisible geometry.

The GameTools visibility libraries contain state-of-the-art visibility culling algorithms, examples of which can be seen in Fig. 7. Figure 7 (left) shows an interactive walkthrough of the city of Vienna which uses a potentially visible set visibility culling algorithm. Figure 7 (right) shows the use of coherent hierarchical culling (Bittner et al. 2004) to reduce the number of rendered objects rendered from the initial 2000 to only 425.

3 Updated GameTools

A new project was started in 2009 to update the geometry and illumination routines of GameTools to support other operating systems and gaming platforms. Since the Ogre3D engine is multiplatform, running on windows, linux, and OSX, it was considered a worthy endeavor to rewrite the GameTools routines portably, using the Ogre3D engine as a test bed to check for regressions.

However, the rapid pace of development in free software projects, when compared to proprietary software (and the lesser emphasis on backwards compatibility) have forced us to follow the Ogre3D development closely and continue updating our libraries at a fast pace. Currently, our version uses Ogre3D 1.7.3 and Unity3D 3.4. A simplified structure diagram of the routines can be seen in Fig. 8.

The illumination routines are divided into render techniques (materials) and render runs (auxiliary techniques used by the render techniques). In Ogre3D and Unity3D, materials are built using render techniques. Unity3D scripts control some

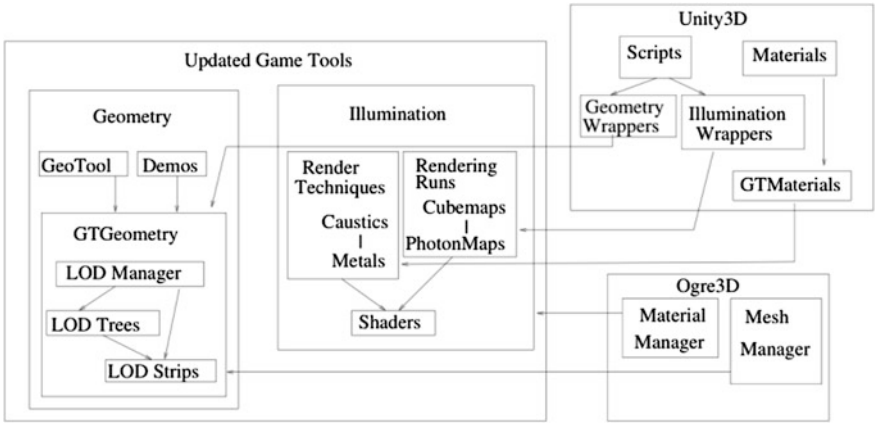

Fig. 8 Structure diagram of the updated GameTools libraries, and their integration in the Unity3D and Ogre3D engine

of the functionality. The geometry routines control the LOD of the models, and are called from the material manager in Ogre3D and from Unity3D scripts.

3.1 Porting Procedure and Lessons Learnt

The original GameTools libraries follow correct design, programming and testing procedures, and work quite well in their supported platform. However, while updating the code, we have found undocumented assumptions which introduce hidden bugs and make porting more difficult. This section describes some of these bugs and describes procedures which would have found most of these bugs in the early phases of development, when they could be fixed more easily.

3.1.1 Persistent Data Structures

The GameTools libraries create persistent data for different purposes. For example, the geometry library creates and stores triangle strips and additional data to manage the CLOD of both triangle strips and procedural trees. The illumination library creates and stores illumination samples and data for the precomputed path maps.

These files contain different types of integers, floating point, and boolean data, and since the only supported platform was Visual Studio 2005 on Microsoft Windows, the files were created by storing the binary data directly onto disk, and read by transferring the data from disk into memory. No effort was spent documenting endianness, size of data types, alignment padding or otherwise. The move

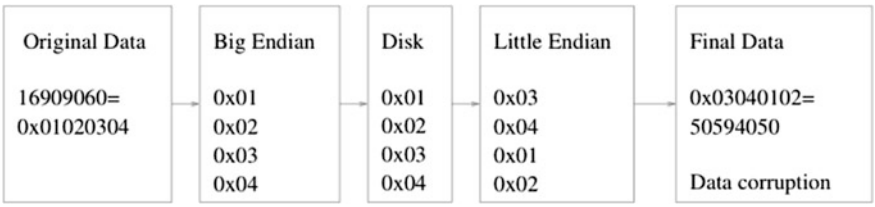

Fig. 9 Example of file transfer between machines with different endianness creating corrupted data

to Windows on 64 bit architectures was not foreseen. This means that the use of a different compiler (even on 32 bit Windows) would introduce bugs in file access, and complicates the porting to Mac OSX (where the size of booleans is different), 64 bit Windows and linux (where the size of long is different) and tablets (where endianness is different). See Fig. 9 for an example.

This difficulty in porting could have been prevented if data types had been standardized in the software design phase, and if proper methods to separate memory data structures from disk data structures had been added. We recommend adding a variety of hardware and software platforms to the testing procedures, including big and little endian architectures, different word sizes if possible, and different compilers, even if only one platform will be eventually supported. This ensures that nonportable code can be detected early, isolated if the effort to make it portable is deemed too high, and that file formats are properly documented.

3.1.2 Memory Access Problems

Memory access problems are the most common bugs in C++ development, and many applications have been created trying to reduce this problem, even going to the trouble of emulating the whole processor and memory access (Valgrind 2012; Seward and Nethercote 2005). Despite that, mistakes in memory access remain in most programs, and are a major fraction of crashing bugs and hacking/cracking attempts.

Our experience porting and updating the GameTools routines showed that these programming errors can remain hidden and be difficult to fix because of two reasons. First, the memory management code on Microsoft Windows does not reuse free memory very aggressively, so freed memory remains ‘valid’ for potentially a large amount of time. Second, the Visual Studio 2005 implementation of the standard template library collections in many cases does not take advantage of the fact that references to objects in collections are not valid after a change (this would allow the use of complex data structures to decrease computation time). Software testing does not find any of these errors, since to the operating system, the memory is actually valid. Again, testing on different architectures, compilers and operating systems permits early detection and fixing of these errors at a small cost.

3.1.3 External Libraries

Complex software is not created from scratch. In most cases, many already existing libraries are reused, and care must be taken to write efficient code to interface with them. This is especially problematic when libraries expect data in different formats, since in many cases more time is spent translating formats than doing actual work.

However, the main problem we have encountered is that many libraries which appeared to be thriving during the time frame of the original GameTools project have been abandoned, forcing us to either use old, unmaintained versions with unpatched security problems, or "inherit" these libraries and update them ourselves. The problem with this last approach, of course, is that the managing of large libraries is a big project in itself, and managing more than one can be overwhelming for small groups.

This problem can be mitigated by identifying the supporting libraries, finding other libraries solving similar problems, and designing an abstract interface which can deal with different back-ends (the model view controller architecture is one possible architecture). Although this is more costly than choosing only one library, the long-term benefits usually are worth the cost. Many large libraries, therefore, follow this approach, which we recommend. An example is OpenCV (2012), which is one dependency of our augmented reality demo. See Fig. 10 for an overview.

Interestingly, many well-maintained libraries presented the opposite problem: development was so fast that retaining compatibility with updated versions of the

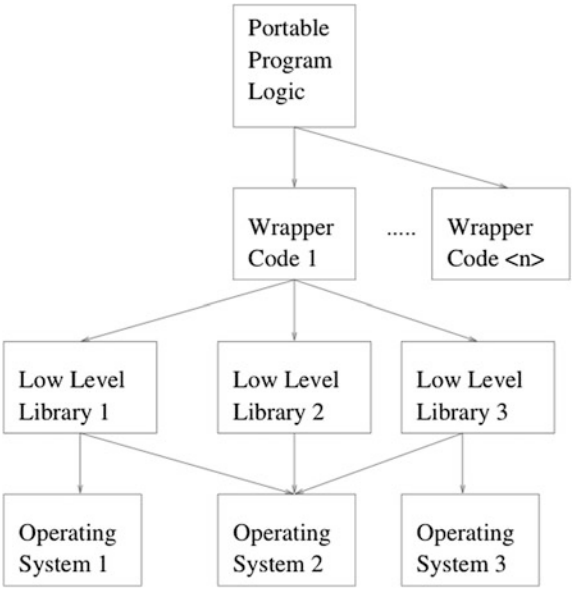

Fig. 10 Example of runtime or compile-time selection of libraries in portable code

libraries was much more costly than originally expected. In particular, the Ogre3D game engine released two major versions (and eight minor versions) during the time frame of our new project, which required updating of the interfaces between the GameTools libraries and Ogre3D.

This last problem is quite important when fusing free software libraries (which seem to have a quicker pace of development), since the support for older versions from the development team is small, and since having access to the source code encourages the changing of the APIs and the removal of older, less efficient APIs, which breaks binary compatibility.

Additionally, care should be taken when choosing external libraries to preserve the portability of the whole system. Not only should libraries advertise support for the operating systems targeted by the software being developed, but testing procedures should be added to test this support, since in many cases, support can be enough to create simple applications, but advanced features are not supported. For example, although Unity3D supports both Windows/DirectX and OSX/OpenGL platforms, the OpenGL support only supports second generation hardware, and does not permit the use of complex, third or fourth generation shaders, even though the underlying operating system and card supports it. As a consequence, our advanced illumination routines cannot run on OSX Unity3D.

Another caveat worth mentioning is the licensing of these libraries. While libraries are chosen taking their licensing into account so that they can be bundled in final products, it is sometimes the case that the licensing of the final product changes in order to respond to marketing or other pressures. In our case, the original GameTools libraries were created under european union funding, and original plans included both open source licenses (for initial libraries) and commercial licenses (for optimized code) (GameTools 2008a). The follow-up project was started under a Catalan grant, and encouraged proprietary licenses.

These changes in licensing require a reassessment of all the supporting libraries, and may require new libraries to be swapped in. Though the approach presented above will help the transition, we cannot encourage enough contacting the copyright holders of all used libraries beforehand and requesting their opinion on different types of licenses (including obtaining customized licenses).

3.1.4 Portability in GPU Code

We have used Nvidia's Cg language to create portable algorithms running on GPUs. Cg can be used from both OpenGL and DirectX, and runs on windows, linux and OSX; the GPUs supported include nVidia, 3Dlabs, ATI and Matrox. Additionally, Cg and HLSL are quite similar languages, allowing us to reuse the original HLSL code created for the original GameTools libraries.

Nevertheless, we have encountered some small, unexpected portability problems, which we detail here. Although Cg allows us to write portable programs, in many cases some parts of the graphic pipeline environment implicitly affects the result. The default values of the projection and modeling matrices, and other

rendering state, are different in DirectX and OpenGL. As a result, even simple code might need to be either rewritten to take these values into account, or the expected value of all the rendering states affecting the GPU code must be documented. In particular, since the handedness of OpenGL and DirectX is different, checking the z-buffer might require changing the comparison operators. This is complicated by the fact that Ogre3D uses an OpenGL-like environment when rendering under DirectX. Additionally, the use of external rendering engines and libraries restricts the constructs which can be used in the Cg code, since rendering engines may have old versions of the Cg compiler, or use HLSL compilers with additional restrictions on the code they accept.

We again recommend testing all available architectures from early stages of development, when all the implementation details and decisions are still fresh in the developers' minds.

3.2 Supported Game Engines and Libraries

The updated GameTools libraries are portable, running on Linux, Windows, and Mac OSX. Supported game engines include Ogre3D and Unity3D (Unity 2012), but users may use them in conjunction with other engines or libraries. Example C-language wrappers and different demos showing the use of the API are provided. An example of an augmented reality demo using the ArUco (2012) and OpenCV libraries is shown in Fig. 11.

3.3 Geometry Library: GTGeometry

The GTGeometry library contains different modules dealing with CLOD algorithms:

Fig. 11 Example augmented reality application using our libraries, running on GNU/Linux and OSX, at different LODs

- Stripification: takes a list of triangles as input and produces triangle strips suitable for efficient rendering.
- Simplification: generate geometry or viewpoint-driven simplified meshes.
- Simplification of trees: specific algorithms for vegetation.
- Managing of LODs: Automatic calculation of optimal LODs.

The library is backwards compatible with the original GameTools geometry libraries, and has been ported to the Unity3D engine. The modules are described in more detail in the following sections.

3.3.1 Stripification Module: LODStrips

The stripification module contains code to create efficient triangle strips from general models. The first demo demonstrate the LODStrips multiresolution run-time library (Geometry: LODStrips Library). The application (Fig. 12) shows a group of models which are able to change their level of detail depending on the distance of the group of objects to the camera. The information panel on the bottom-left corner of the screen shows the current LOD factor, frames per second and the amount of geometry sent to the renderer. It can be seen how the frame rate increases as the LOD decreases.

The level of detail can be calculated in two ways: automatic LOD (based on the distance of the group to the camera) and manual LOD (the user changes the level of detail independently from the distance), which changes the level of detail of the objects manually. This last mode is useful to see the meshes in detail even when their level of detail is set to the minimum.

The demo also shows how LOD operations can be minimized by grouping some instances of the same model to be managed by a single LODStrips multiresolution model. This is useful when some models need similar levels of detail and will improve the overall performance.

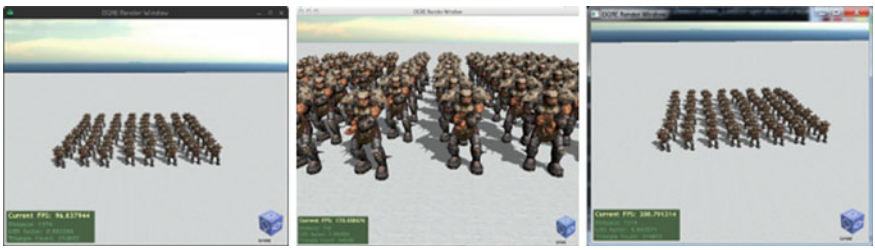

Fig. 12 LODStrips example, Ogre3D, Linux, Mac OSX, Windows

3.3.2 Continuous LOD Managing Module: LODManager

The LODManager demo features a massively populated scene composed of 1,200 models. Each one of them is attached to an independent multiresolution model instance that manages its level of detail. To manage the level of detail of such a vast scene, we introduce the use of the LODManager, which decides whether an object can change its level of detail freely or just has to borrow an already calculated LOD snapshot.

The demo allows the user to enable or disable the LODManager capabilities to show the difference in performance. The LODManager is able to keep the bottleneck of the application in the graphics engine and not in the LOD calculations. A screenshot can be seen in Fig. 13.

3.3.3 Continuous LOD for Vegetation: LODTrees

This demo presents Geometry: LODTreeLibrary, the LODTrees multiresolution runtime library. The demo is composed of some groups of multiresolution trees. The LOD of each group, composed by some trees of the same type, is managed by a single LODTree instance to optimize performance. The result is a forest of multiresolution trees which change its level of detail depending on the distance to the camera of each one of these groups.

The geometry of the trees in our demo is provided by the Xfrog software (2012), but the LODTrees module itself can use geometry from other sources. Figure 14 shows an image of the LODTrees demo.

3.3.4 Geometry Management Application: GeoTool

GeoTool is an application which eases the creation of the LOD models. It uses the FLTK (2012) toolkit to ensure portability among the three supported architectures. Its main features include simple operations such as triangle stripification, mesh

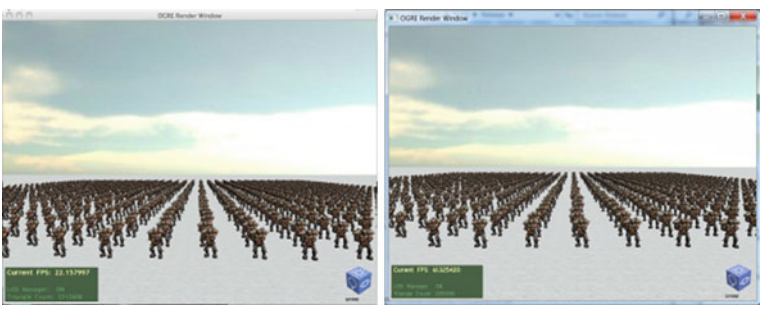

Fig. 13 LODManager example, Ogre3D, Mac OSX, Windows

Fig. 14 LODTrees example, Ogre3D, GNU/Linux and Mac OSX

simplification (geometry or viewpoint based), leaves simplification, and complex operations (e.g., LODStrip and LODTrees creation and visualization). A screenshot of the software can be seen in Fig. 15.

3.3.5 GTGeometry in Unity3D

Unity3D Pro allows native C code to be called from within its scripts. To use GTGeometry within Unity3D, C wrapper routines can be used. An example of a LODTree controlled within Unity3d can be seen in Fig. 16.

3.4 Illumination Library: GTIllumination

The GTIllumination library contains different global illumination effects, implemented efficiently using GPU programming. Effects include realistic metals and glass, caustics, particle systems, hierarchical systems, tone mapping, precomputed radiance maps, multiple reflections and refractions, and various screen space

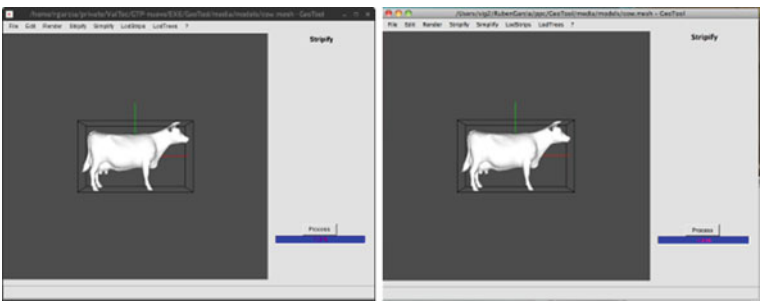

Fig. 15 GeoTool application, GNU/Linux and Mac OSX

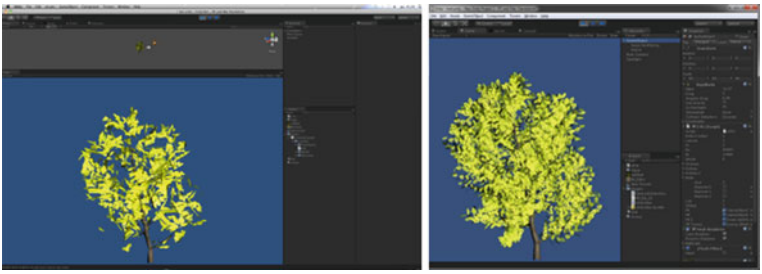

Fig. 16 Continuous LOD Tree in Unity3D, Mac OSX and Windows, at a LOD of 30 % (*left*) and 100 % (*right*)

ambient occlusion algorithms. Figures 17, 18, 19 and 20 show screenshots of the different effects.

The software package contains GPU shaders used to implement the different materials and effects. Interfacing packages have been created for Ogre3D and Unity3D in order to make the effects easy to use in these engines. The work flow is similar to using predefined materials and effects. Ogre3D's materials and techniques can be defined in material scripts with the use of keywords, and a Unity3D package is provided to add the new functionality, so that drag-and-drop can be used to indicate techniques and materials.

Scripts and prefabs are provided to link the different techniques to the relevant Unity3D objects; the only caveat is that since the (color and distance) cubemap textures required to calculate accurate reflections are included in the material, each

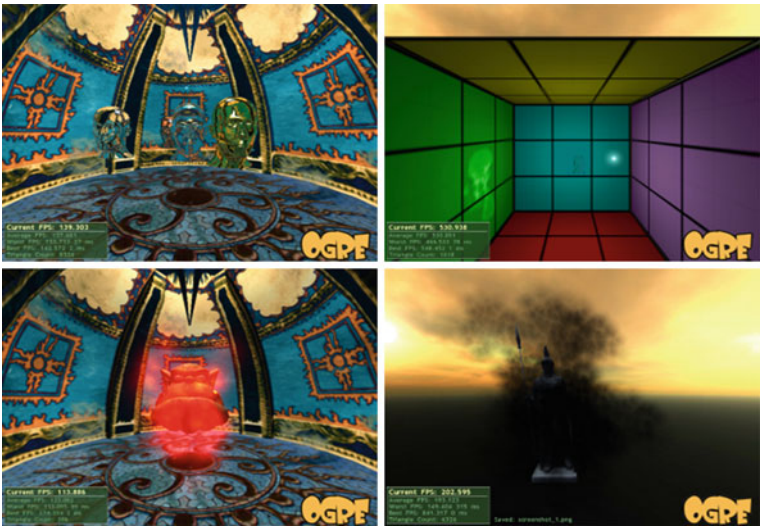

Fig. 17 Realistic metals and glass, caustics, particle systems, hierarchical systems

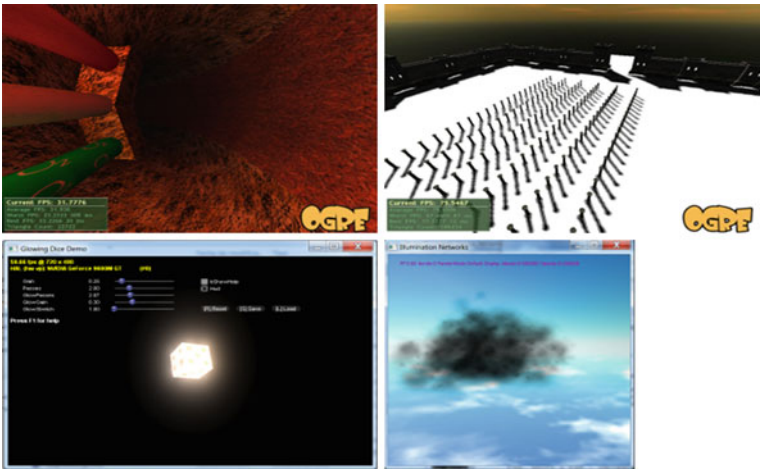

Fig. 18 Precomputed radiance maps, soft shadows, glowing effect, and illumination networks

Fig. 19 Tone mapping and eye adaptation

object should have its own instance of the material (the material can be shared among nearby objects but the quality of the picture will be affected, since objects sharing materials will not be reflected on each other). Figure 21 shows an example of the Unity3D package in action. The material for the skull object is a realistic gold, and its parameters and textures are updated in real time by a child CCM prefab containing the scripts and cameras required to update the color and distance cubemaps and other parameters. Additional scripts (not shown) are attached to the camera to ensure correct image generation.

Figure 22 shows examples of metal materials on the windows platform. The effects have been confirmed to work in Linux and Mac OSX using the windows emulator Wine (2012) (see Figs. 23 and 24), so we are confident that our routines will be available in Unity3D on the Mac platform as soon as Unity3D updates their OpenGL support, and on the linux platform when Unity3D adds support for the platform in the near future.

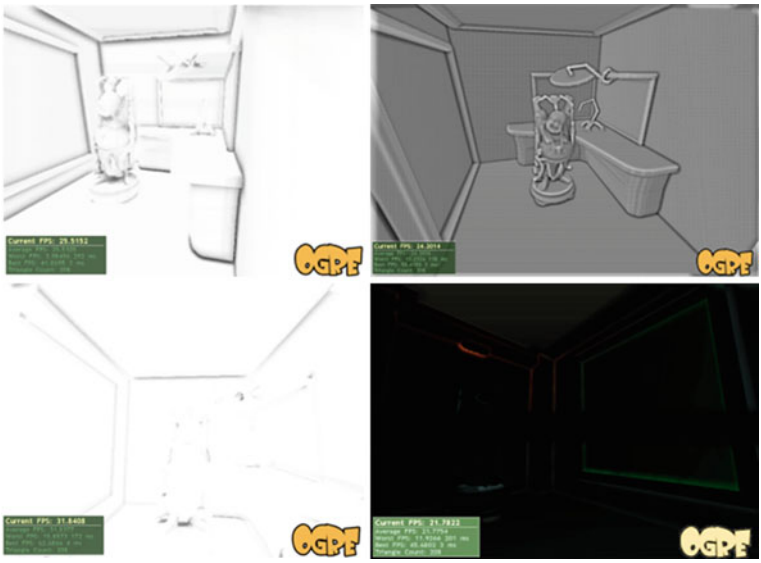

Fig. 20 Screen space ambient occlusion: classic, Crytek, volumetric, and volumetric with color bleeding

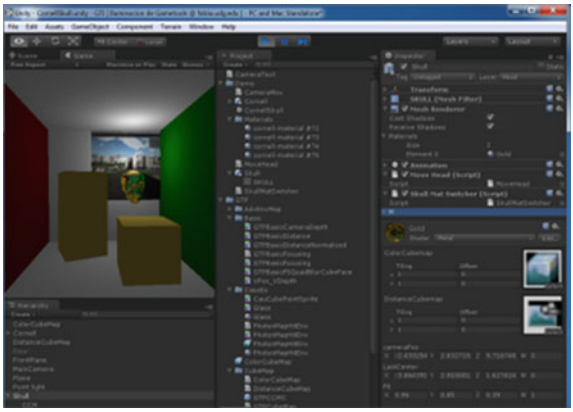

Fig. 21 GameTools illumination effects in Unity3D

4 Use Cases

The updated GameTools libraries have been used in two serious games created with the Unity3D engine. These games have been designed in collaboration with the faculty of arts and the faculty of education of the University of Girona, and they are to be used to help teaching local history to middle school students. An agile software development methodology has been followed to create these games.

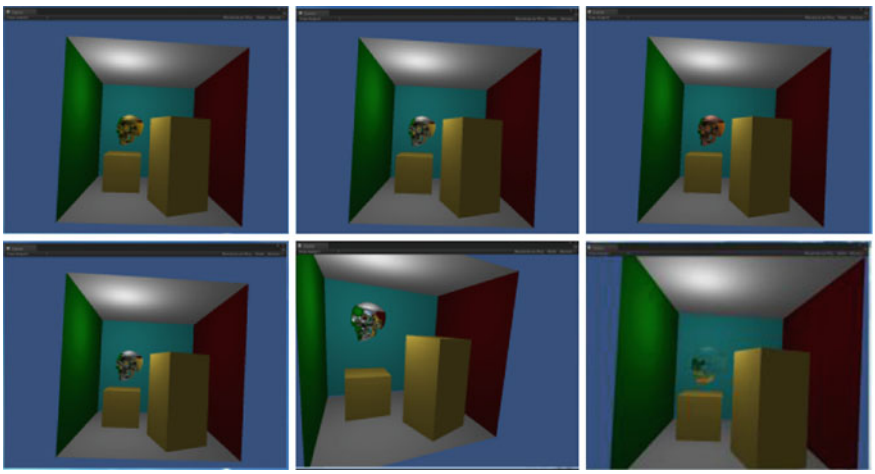

Fig. 22 Realistic gold, silver, copper, and aluminum metals, ideal metal and glass in the Unity3D engine (Windows)

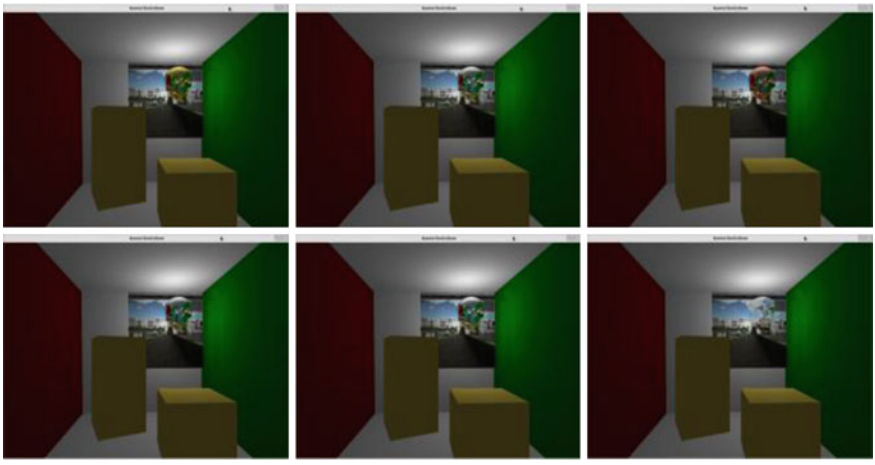

Fig. 23 Realistic gold, silver, copper, and aluminum metals; ideal metal and glass in the Unity3D engine (Linux + Wine)

Preliminary testing with end users has been carried out, and more widespread tests are planned for the near future.

The first game is called Jaume I and simulates the conquest of Mallorca by James I the Conqueror in 1229. The second game is legends of Girona. The legends of Girona game explores the different legends concerning the city of Girona. In particular, the legend of the miracle of Saint Narcis, who repelled the siege of Girona in 1285, has been implemented. The objective of these two games

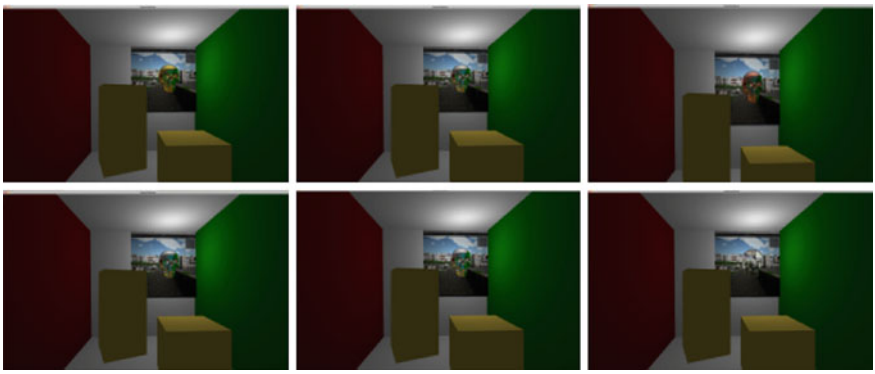

Fig. 24 Realistic gold, silver, copper, and aluminum metals in the Unity3D engine (Mac OSX + Wine)

is to provide an enjoyable way to study the history of the kingdom of Aragon in the middle ages (Jaume I) and specifically the history of the city of Girona (Legends).

The third use case is not strictly a serious game, but a computer version of a board game called Nine Men's Morris (and its more common variants). However, since this game has been played since roman times and was very popular in medieval times across most of Europe, it is of interest for interactive museum exhibits dealing with roman or medieval times. Because of this, we may consider it a serious game in the context of cultural heritage. The objective of this game is to familiarize museum visitors with the different variants of this milenary board game, and the materials commonly used during roman and medieval times. This game has been designed using a classical iterative development model, and informal testing has been carried out during the development of the game.

No changes in the design of the games were required to integrate the updated GameTools effects. Drag-and-drop was used to indicate the position of the CLOD trees and the realistic materials.

4.1 Jaume I: The Battle of Portopí

Jaume I, a strategy game, reproduces a historical battle in thirteenth century, with archery, infantry and cavalry, won by king of Aragon Jaume I which led him to conquer Mallorca. The game has been developed in collaboration with the faculty of arts of the University of Girona, and tries to reproduce the historical characters and skirmish situations. Successful players will develop mastery of the history of the Aragon kingdom before unification with the Castilian kingdom to form Spain.

Figure 25 shows a continuous level of detail tree created using the GTGeometry libraries at the beginning of the game.

Fig. 25 Screenshots of the Jaume I serious game, including a procedural tree from GTGeometry at different LODs

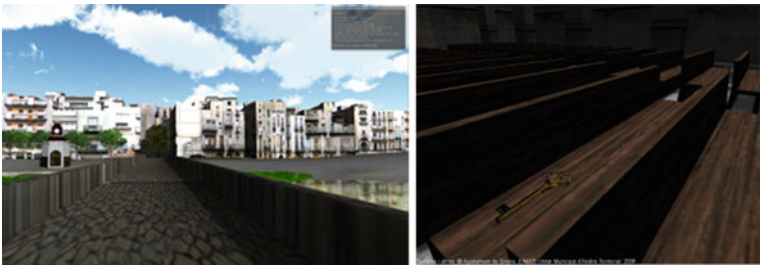

Fig. 26 Screenshots of the Legends of Girona serious game, including a procedural tree from GTGeometry and a realistic gold key from GTIllumination, running on Mac OSX and Windows

4.2 Legends of Girona

Legends of Girona is a first/third person educational game developed together in collaboration with the faculty of education of the University of Girona. The game simulates a legendary happening in the town during a foreign invasion in the thirteenth century. The successful player develops mastery of the history of the town of Girona and its ancient urban and geographical disposition through navigating and exploring game artifacts without being captured. Eventually the player needs to find a way out of a labyrinth in an old church crypt.

Figure 26 (left) shows a screenshot of the game with a tree showing simplified geometry. When the user crosses the bridge, the tree geometry is refined smoothly in real time using the continuous level of detail routines from our libraries. Figure 26 (right) shows another screenshot, which displays a gold key with realistic reflections in real time.

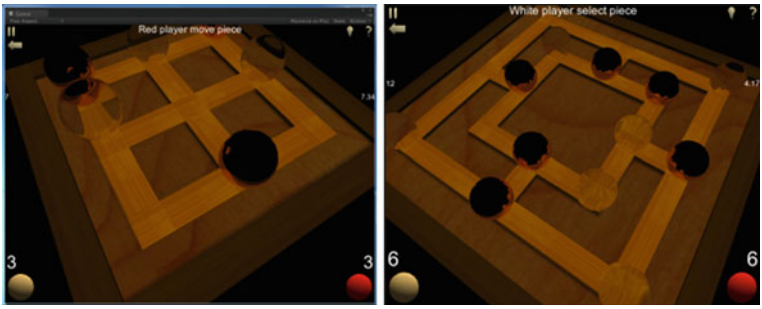

Fig. 27 Screenshots of the Nine Men's Morris game, including realistic cooper and glass materials from GTIllumination, running in Windows15.5 conclusions and future work

4.3 Nine Men's Morris

Nine Men's Morris is a traditional board game similar to Tic-Tac-Toe. While not strictly a serious game, we are interested in including this game (and its many variants) in interactive museum displays in the United Kingdom and Northern Europe, because this game was very popular there in medieval times.

We have implemented different variants of the Nine Men's Morris strategy game, and included an artificial intelligence opponent. We have used the realistic copper and glass materials from the GTIllumination libraries to distinguish the two players, and a wood material for the board. These materials have been used since ancient times for board games, and we expect the increased realism provided by our routines to help set the scenario in museum exhibits. Figure 27 shows a screenshot of the game; notice the multiple inter-reflections and refractions visible in the marbles' surfaces.

5 Conclusions and future work

We have presented in this paper a description of the updated GameTools libraries, showing how they can be used to create advanced graphic effects in popular game engines. Users of the routines are not expected to have in-depth knowledge of graphic algorithms, and this makes the routines especially suited for inclusion in serious games created by programmers with backgrounds other than computer graphics. The main advantage of the routines is that they can be used to increase the performance or realism of serious games without requiring changes in the design of the games. Three use cases are provided: the games Jaume I, Legends of Girona and Nine Men's Morris.

Our plans for future work include adding a visibility culling module to our routines, adding more game engines and platforms (including popular consoles

such as Microsoft Xbox, Nintendo Wii, and Sony PlayStation), and adding more advanced effects (other simplification and LOD techniques, more realistic shadows, other screen space ambient occlusion, and/or global illumination algorithms), in collaboration with academia.

Acknowledgments This work has been supported by the research projects coded TIN2010-21089-C03-01 and IPT-2011-0885-430000 (Spanish Commission for Science and Technology) and by grants VALTEC09-2-0118 and 2009SGR643 (Catalan Government).

References

ArUco (2012) ArUco: a minimal library for augmented reality applications based on OpenCv. http://www.uco.es/investiga/grupos/ava/node/26. Accessed 23 January 2012

Bittner J, Wimmer M, Piringer H, Purgathofer W (2004) Coherent hierarchical culling: hardware occlusion queries made useful. url http://www.cg.tuwien.ac.at/research/publications/2004/Bittner-2004-CHC/. Proceedings of the Eurographics 2004. Grenoble, France

Domingo A, Escriva M, Abad F Lluch J, Camahort E, Vivo R (2007) Continuous LODs and adaptive frame-rate control for spherical light fields. In: Geometric modeling and imaging–new trends pp 73–78 doi:http://doi.ieeecomputersociety.org/10.1109/GMAI.2007.14

Fast light toolkit (FLTK) (2012) http://www.fltk.org/. Accessed 23 January 2012

GameTools (2008a) GameTools project newsletter. http://www.gametools.org/downloads/GTP_newsletter3.pdf. Accessed 20 January 2012

GameTools (2008b) GTP. http://www.gametools.org/. Accessed 20 January 2012

González C, Castelló P, Chover M (2007a) A texture-based metric extensión for simplification methods. In: Braz J, V′azquez PP, Pereira JM (eds) GRAPP (GM/R), Institute for systems and technologies of information, control and communication (INSTICC). Barcelona, Spain pp 69–76

González C, Gumbau J, Chover M, Castell P (2007b) Simplificación de mallas para juegos, vol 1. In: Congreso Español de Informática Gráfica (CEIG 2007), Eurographics Association, Zaragoza, Spain

Gumbau J, Ripolles O, Chover M (2007) Lodmanager: a framework for rendering multiresolution models in real-time applications. Proceedings of the WSCG'2007 short communications papers. Plzen, Czech Republic pp 39–46

Hadwiger M, Varga A (1999) Visibility culling. In: Proseminar Wissenschaftliches Arbeiten, 1998–1999 url: http://www.cg.tuwien.ac.at/∼msh/viscull.pdf

Lazányi I, Szirmay-Kalos L (2005) Fresnel term approximations for metals. Proceedings of the WSCG 2005, short papers. Plzen, Czech Republic pp 77–80

Lazányi I, Szirmay-Kalos L (2006) Indirect diffuse and glossy illumination on the GPU. In: Engel W (ed) ShaderX5: advanced rendering techniques. Charles river media, Chilton pp 345–358

Méndez A, Sbert M, Cata J, Sunyer N, Funtane S (2005) Real-time obscurances with color bleeding. In: Engel W (ed) ShaderX4: advanced rendering techniques. Charles river media, Chilton

OGRE (2012) OGRE—Open Source 3D Graphics Engine. http://www.ogre3d.org/. Accessed 20 January 2012

OpenCV (2012) Welcome–OpenCV wiki. http://opencv.willowgarage.com/wiki/. Accessed 23 January 2012

Rebollo C, Gumbau J, Ripolles O, Chover M, Remolar I (2007a) Fast rendering of leaves. Proceedings of the 9th international conference on computer graphics and imaging (IASTED) CGIM 2007. ACTA Press, Anaheim, pp 46–53 url http://dl.acm.org/citation.cfm?id=1710707.1710717

Rebollo C, Remolar I, Chover M, Gumbau J, Ripolles O (2007b) A clustering framework for real-time rendering of tree foliage. J Comp 2(4):57–67

Seward J, Nethercote N (2005) Using valgrind to detect undefined value errors with bit-precision. Proceedings of the annual conference on USENIX annual technical conference. USENIX Association, Berkeley, p 2–2 url http://dl.acm.org/citation.cfm?id=1247360.1247362

Shark3D (2012) Shark 3DTM by Spinor GmbH. http://www.spinor.com/. Accessed 20 January 2012

Szécsi L, Szirmay-Kalos L, Sbert M (2006) Light animation with precomputed light paths on the GPU. Proceedings of graphics interface 2006 (GI2006), Canadian Information Processing Society, Toronto pp 187–194

Szirmay-Kalos L, Aszódi B, Lazányi (2005a) Ray-tracing effects without tracing rays. In: Engel W (ed) ShaderX4: lighting and rendering. Charles River Media, Chilton

Szirmay-Kalos L, Aszódi B, Lazányi I, Mátyás P (2005b) Approximate raytracing on the GPU with distance impostors. Compu Grap Forum 24(3):695–704

Szirmay-Kalos L, Lazányi I (2006) Indirect diffuse and glossy illumination on the GPU. Proceedings of the SCCG 2006. Smolenice castle, Slovakia pp 29–35

Tóth B, Szirmay-Kalos L (2007) Fast filtering and tone mapping using importance sampling. Proceedings of the WSCG 2007 short papers. Plzen, Czech Republic pp 47–52

Umenhoffer T, Szirmay-Kalos L (2005) Real-time rendering of cloudy natural phenomena with hierarchical depth impostors. In: Eurographics short paper. Dublin, Ireland pp 65–68

Umenhoffer T, Szirmay-Kalos L, Szíjártó G (2006) Spherical billboards andtheir application for rendering volumetric data. In: Engel W (ed) ShaderX5: advanced rendering techniques. Charles River Media, Chilton pp 275–286

Umenhoffer T, Szirmay-Kalos L (2007) Robust diffuse final gathering on the GPU. Proceedings of the WSCG 2007 full papers. Plzen, Czech Republic pp 121–128

Umenhoffer T, Szirmay-Kalos L (2006) Spherical billboards for rendering Explosions.In: Graphics Interface. Quebec, Canada pp 57–64

Umenhoffer T, Patow G, Szirmay-Kalos L (2007) Robust multiple specular reflections and refractions. In: Nguyen H (ed) GPU Gems 3. Addison, Wesley

Unity (2012) Unity: game development tool. http://unity3d.com/unity/. Accessed 20 January 2012

Valgrind (2012) Valgrind home. http://valgrind.org/. Accessed 23 January 2012

Wine (2012) WineHQ—Run Windows applications on Linux, BSD, Solaris and Mac OS X. http://www.winehq.org/. Accessed 29 February 2012

Xfrog (2012) Xfrog—news. http://xfrog.com/. Accessed 23 January 2012