A model for adapting 3D graphics based on scalable coding, real-time simplification and remote rendering

Marius Preda · Paulo Villegas · Francisco Morán · Gauthier Lafruit · Robert-Paul Berretty

Abstract Most current multiplayer 3D games can only be played on dedicated platforms, requiring specifically designed content and communication over a predefined network. To overcome these limitations, the OLGA (On-Line GAming) consortium has devised a framework to develop distributive, multiplayer 3D games. Scalability at the level of content, platforms and networks is exploited to achieve the best trade-offs between complexity and quality.

Keywords 3D graphics compression and adaptation · On-line games

1 Introduction

Producing synthetic images is in general a highly demanding process in terms of computation. It depends on the complexity of the 3D scene and on the requirements for realistic rendering. It can however be facilitated by hardware equipment for accelerated 3D rendering, or by rendering scalability (switching off some rendering features makes it possible to obtain synthetic images on less powerful platforms).

Current games are delivered with instructions about the minimal hardware configuration necessary to play them. During their initialization phase, the hardware is profiled and binary selection of rendering features is performed. The model is even more constrained for games designed for a priori known platforms (game consoles).

All the many existing representation formats for 3D graphics imply some balance between a more or less compact set of parameters and a (respectively) more or less complex software to interpret and render them. An object may be defined at low-level (e.g., a set of colored vertices and the connection between them) or by using predefined high-level primitives (e.g., teapot, humanoid). In general, the first variant needs a larger amount of parameters, but it is the one used by graphic cards for rendering. Within an isolated environment (e.g., games on DVD), when storage is not an issue, low-level representation of the data is preferred also for file format. However, in on-line scenarios, when 3D content has to be transmitted over a network (sometimes a low-bandwidth and high-latency one, as in the case of mobile devices) and to heterogeneous terminals, specific technologies such as compression, adaptation and high-level representation must be used. Such technologies are the object of the current paper, which describes a framework for developing scalable 4D (animated 3D) game content adaptively streamed to a variety of terminals over heterogeneous networks. Thanks to the tools implemented into this framework, developed within the OLGA research project, it is possible to render the same textured 4D content at wildly different qualities and frame rates, according to network and terminal profiles.

Section 2 introduces the mathematical model governing the framework, with configurations for adaptation, coding and rendering. Section 3 discusses the approaches supporting the three key techniques for the media composing a 3D object: geometry, texture and animation. The optimal network configuration needed for the complexity of the system is described in Sect. 4. Finally, Sect. 5 concludes our presentation.

2 Formal framework

The process we are modeling is that of the visualization on a terminal of a 3D graphical model, possibly coming from a server across a network channel. This can be represented as a workflow involving a set of transformations. As input, the 3D model is defined as a set of features $\{F_i\}$ of very heterogeneous nature: vertex locations and connectivity, texture bitmaps and coordinates, key frames for animation... The output is $\{P_i\}^{2D} = \{R_i, G_i, B_i, A_i\}$, a set of 2D pixels, with color (RGB) and transparency (A), ready to be displayed into, e.g., a frame-buffer; or $\{P_i\}^{3D} = \{R_i, G_i, B_i, A_i, D_i\}$, a set of 3D pixels containing a depth component computed at rendering time. The goal is then to produce either of the two sets $\{P_i\}^{2D}$ or $\{P_i\}^{3D}$ from the set $\{F_i\}$ in an optimized way, taking into account the constraints coming from the elements in the system.

2.1 Set of transformations: rendering, coding, simplification and modeling

$$\{P_i\} = \mathbf{R}(\{F_i\}), \tag{1}$$

$$\{F_i^C\} = \mathbf{C}(\{F_i\}), \tag{2}$$

$$\{F_i^S\} = \mathbf{S}(\{F_i\}), \tag{3}$$

$$\{F_i\} = \mathbf{M}(\{M_i\}). \tag{4}$$

The most straightforward transformation is that of *rendering* **R**, which projects the model onto planes suitable for display, thus directly converting $\{F_i\}$ into $\{P_i\}$ as stated by (1). The computation of **R** depends on the intelligence of terminal and on the nature of $\{F_i\}$, which may contain a higher or a lower level representation.

In a networked (or storage) scenario another transformation appears: the *coding* **C**, defined in (2), is the composition of encoding (compression), transmission and decoding, which adapts the set of features for efficient transmission. In general, it compacts the representation of $\{F_i\}$ and reduces the needed network bandwidth. It can also make the representation more robust against channel errors, or provide scalable formats or a standardized representation.

In many cases it is possible to simplify the set of features without affecting the final quality of the presented pix-map, because the display resolution, the viewing conditions or the scene allow the elimination of irrelevant information. This operation could be performed simultaneously with rendering, but it is advantageous to model it as a separate transformation, called *simplification* **S**, which provides adaptation to terminal and viewing constraints: see (3).

The fourth transformation considers a previous state, that of the description of the model, which can be made in several forms: human instructions, procedural primitives, formal semantic representations, or some form of metadata, possibly standardized, e.g., MPEG-7-compliant. When the description is rich enough it is possible to synthesize the content from it , so we define the *modeling* transformation **M** as the one that turns the metadata $\{M_i\}$ into a model, as indicated by (4).

2.2 Processing chain and constraints

We can now apply the defined transformations, **R**, **C** and **S** to define the processing chain that goes from the 3D model to the final pix-map. This chain has considerable flexibility in the order in which transformations are applied, as can be seen in the set of (5a–f).

$$\{P_i\} = \mathbf{R} \circ \mathbf{S} \circ \mathbf{C}(\{F_i\}),^2 \tag{5a}$$

$$\{P_i\} = \mathbf{R} \circ \mathbf{C} \circ \mathbf{S}(\{F_i\}), \tag{5b}$$

$$\{P_i\} = \mathbf{C}' \circ \mathbf{R} \circ \mathbf{S}(\{F_i\}), \tag{5c}$$

$$\{P_i\} = \mathbf{C}' \circ \mathbf{S}' \circ \mathbf{R}(\{F_i\}), \tag{5d}$$

$$\{P_i\} = \mathbf{S}' \circ \mathbf{C}' \circ \mathbf{R}(\{F_i\}), \tag{5e}$$

$$\{P_i\} = \mathbf{S}' \circ \mathbf{R} \circ \mathbf{C}(\{F_i\}). \tag{5f}$$

Equation (5a) states a processing chain where the model is first encoded at the server side and transferred to the client, in which it is simplified and then rendered. In (5b), simplification happens at the server side, and the model is then sent to the client, by which it is rendered. In (5c), the model is both simplified and rendered by the server, and the resulting pix-maps are then sent to the client for immediate display.

Equation (5d) describes the case when the model is first rendered at the server side, simplified in the image/video domain and then encoded. In (5e), rendering is performed remotely, and simplification in the image/video domain is done by the client. And finally, (5f) is similar to (5a) in that simplification and rendering are done at the client side, but simplification is done in the image/video domain. The coding transformation for (5c–e) is named $\mathbf{C}'$ to indicate its different nature, since it works in a different domain, encoding pix-maps instead of models (it is therefore an image/video codec); similarly for $\mathbf{S}'$ in (5d–f).

All these configurations are in principle capable of producing the same output, or at least outputs that are visually indistinguishable from each other. The decision between them will come from the constraints imposed by the environment: computational power at server side, required bandwidth between server and client, network latency and computational power at client side. The chains represented by (5a–f) reflect different trade-offs between those restrictions. Although the relationship between them is too complex to be easily tractable, some heuristics can be defined. Terminals with very low processing capabilities will be best served by (5c–e), in which the client essentially limits itself to decode and display the pix-maps delivered by the server. Additionally, if the network bandwidth is reduced, it is advisable to simplify the content before the transmissions (5c) and (5d). As the computing power of the client increases, the configuration may change to that of (5b), in which the rendering capabilities of the terminal allow sending data in the model domain, something that is (presumably) more bandwidth-efficient. Still more powerful terminals could take also the load of simplification, thereby freeing the server from that task, maybe with bandwidth costs but a gain in flexibility (5a). In the special case in which simplification in the image domain is preferred (e.g., because it is hardware-accelerated while simplification in the model domain is not) then (5f) could be used.

In a real case, those heuristics should be tested by performing adequate validation of the chosen configuration, and through this also the determination of the right chain parameters (number of servers, bandwidth per client, etc.) could be obtained.

3 RCS technologies for on-line games

Creating compelling 4D objects and characters is a very time-consuming task. Several versions (from very simple to complex) of the same object have to be created to enable switching between them at run-time, according to playing conditions (e.g., viewpoint). Adaptation is here the discrete operation of switching between predefined versions. The switching conditions are obtained after a long and human-resources consuming procedure when the game is played in all possible combinations. On the contrary, our global approach aims at producing flexible solutions to create scalable 4D content from scratch, or to recycle already existing 4D content to have it be scalable. When appropriate, scalable (off-line) encoding is of the utmost importance to enable continuous adaptation (at run-time, under constrained system resources) of the 4D content parameters, handled in two manners:

- Directly in the compressed domain, thanks to progressive bit-streams that can be stripped through packet selection mechanisms, in which only the visible portions of a 3D object geometry, texture and animation are transmitted and decoded at the appropriate quality.
- In the spatial/temporal domain (3D model or image/video) by run-time trans-coding on servers. Instead of predefining discrete LODs (Levels Of Detail), by continuously monitoring the terminal and the network, a server can compute the LOD an object should have. Performing the adaptation through trans-coding may become a costly operation in terms of server loading and network bandwidth, thus it will be used only for objects for which scalable representation is sub-optimal.

In the first case, $\mathbf{C}$ and $\mathbf{S}$ are related (the simplification operation is done by discarding packets from the coded stream), whereas in the second they may be de-correlated.

3.1 Geometry

In 3D games there is a large variety of meshes: from objects with very few polygons used for décor to very large meshes modeling terrain or characters. The geometric properties of the objects are also highly heterogeneous: there are objects with corners (high frequencies) and smooth objects such as most organic shapes, animals, humans, etc. Treating all the objects with the same chain is clearly sub-optimal and sometimes even impossible. With respect to the geometric properties of the mesh, two processing paths are defined:

1. For non-organic objects, server-side simplification with fine granularity and continuous LOD control is performed. When content is to be sent from one player to another, it is first sent to a processing server that is informed on the capabilities of the receiver. Then it decodes the mesh, automatically simplifies it through vertex removal and re-encodes it.
2. For organic objects, wavelet decomposition with hierarchical LOD is applied. Several 3ds Max plug-ins and exporters were implemented to enable an artist to semi-automatically simplify an arbitrary connectivity 3D mesh, optionally re-mesh it to have subdivision connectivity, and code it in a plain or scalable manner. Our 3D mesh simplification, the transform $\mathbf{S}$ in Sect. 2, is based on Garland’s quadric error metrics technique

and yields significant improvements over 3ds Max's native solution. We developed two implementations of the simplification algorithm: a plug-in for 3ds Max and a standalone software module integrated in the server and allowing run-time vertex removal, in the trans-coding scenario.

For mesh coding, the transform **C** in Sect. 2, several techniques were developed. For processing path 1, we developed 3ds Max exporters able to encode it thanks to two MPEG-4 tools: BIFS (BInary Format for Scene) and 3DMC (3D Mesh Coding). BIFS defines a binary encoded version of an extended set of VRML, trying to balance the compression performances with the extensibility, ease of parsing and simple bit-stream syntax. However, BIFS does not fully exploit the spatial and temporal correlation of 3D objects. To overcome this limitation, MPEG defined specific tools like the ones contained in the 3DMC toolset The two corresponding decoders are integrated in the PC (Personal Computer) version of our player. As for the CP (Cell Phone) platform, only the BIFS decoder is implemented since it has less processing power requirements. A specific behavior of path 1, including remote rendering (i.e., the transform **R**) is considered for several cases: on the PC platform, for décor objects situated far from the camera; on the CP platform, for all décor objects not involved in the game play; and, on both platforms, for objects whose available triangle budget (computed by the scene manager) is less that the minimum required: see Fig. 1. The simplification operation performed by the server (Fig. 1, c) is replaced by a rendering operation with a view of the object mapped on a rectangle (Fig. 1, d).

For processing path 2, the coding can comply with the WSS tool already in MPEG-4 AFX , a.k.a. WaveSurf, or follow the PLTW (Progressive Lower Trees of Wavelet coefficients) technique Here the **C** and **S** transforms are performed in a congruent manner. The two corresponding decoders (MPEG-4-compliant and PLTW-based) are both integrated in our software framework for the PC platform. As for the CP one, only the PLTW-based decoder has been ported, since it has less memory requirements than WaveSurf. The main novelty of our PLTW technique is that the resulting bit-stream does not impose on the less powerful decoders the need of building detail trees as deep as required by the maximum LOD encoded, because it does not follow blindly Said's SPIHT algorithm [6]. PLTW sends the wavelet coefficients on a per-LOD basis, thus achieving "local SNR scalability" within "global spatial scalability": the set of coefficients is also hierarchically traversed, but they are scanned in LODs, which yields a spatially scalable bit-stream. The decoder first receives all the coefficients corresponding to a LOD and, only when it has finished reading them, and only if it still has enough resources, it proceeds with those from the next.

WSSs permit to code the shape of a 3D model in a multiresolution manner with very good compression, but require a large CPU overhead for a fine-grained, on-the-fly control of the content complexity in execution of time-regulated applications such as networked, interactive terrain fly-overs or 3D games. In fact, the CPU overhead to control the rendering time the WaveSurf tool may be as large as the rendering time itself. Moreover, typical implementations of WSSs multiply by four the number of triangles in every subdivision step, which enables only very discrete LOD management, and therefore yields abrupt and often disturbing quality changes while only supporting coarse-grained adaptation to a target execution time. We introduced some add-ons to enable a low-complexity, yet efficient fine-grained quality/runtime trade-off in execution time control. To achieve this target, the WSS mesh regions are progressively decoded in a continuous LOD fashion, by subdividing only the important regions of the geometry. The importance and order for subdividing the triangles are given by their impact on the error to the target mesh, i.e., the triangles that decrease this error mostly are subdivided first. With special subdivision platform mapping techniques using LOD-based moving windows the complexity of the subdivision control is largely reduced, resulting in an overhead of only a small percentage in the final decoding and rendering execution time for the two different platforms.

3.2 Textures

After carrying out a preliminary comparative study between JPEG, JPEG 2000, and MPEG-4's VTC [9] with respect to the considered criteria and desired functionalities, the JPEG 2000 technology was selected, and several tools developed. The system supports view-dependent texture streaming, optimized for both PC and CP thanks to JPEG 2000 and JPIP (JPEG 2000 Internet Protocol), in which a bit-stream packet selection mechanism takes the user's viewpoint information into account. Additionally, a JPEG 2000 bit-stream packet

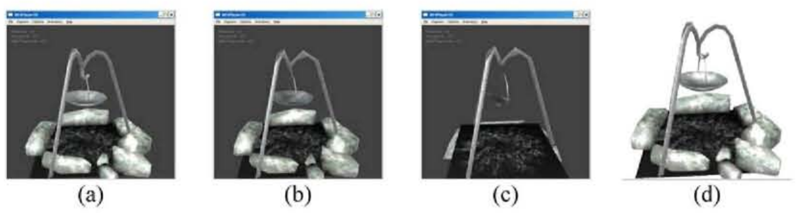

Fig. 1 A décor object simplified at different resolution levels: (**a**) 80%, (**b**) 60%, (**c**) 20%, visually unacceptable, so version (**d**), an image mapped on a billboard, is used instead

selector has been integrated in the Simplificator module running on the server, supporting resolution scaling and bit-plane removal. The LOD selection takes into account both the available bandwidth between server and terminal, and the terminal screen resolution.

Let us note that textures are usually the largest part (in terms of data size) of a 3D model. The regular structure of an image allows easy control of the hierarchical representation. Thus, in our framework, textures are following the processing path 2, the adaptation being performed directly in the compressed domain.

3.3 Animation

Virtual characters are the most complex objects in a 3D game and, in order to represent compactly their animation, either temporal or spatial redundancy must be exploited. In the first case, linear or higher order interpolation is used to compute the value of an attribute based on key values. In the second, vertices are clustered and a unique value or geometric transform is assigned to each cluster. We used as a basis the MPEG-4 AFX BBA (Bone-Based Animation) toolset, which follows a traditional signal compression scheme by including both predictive and transformational encoding techniques. We considered the adaptation of animated content at two levels: geometry simplification constrained by dynamic behavior and animation frame reduction. The dynamic behavior was expressed as constraints used to parameterize the quadric error metrics technique by introducing a weighting factor to specify how the bones structure influences the simplification procedure.

An optimized implementation of the BBA encoder obtains up to a 70:1 compression factor with respect to a textual representation. Decoding animation data on low memory devices such as CPs requires server-side adaptation. Our approach was to reduce the number of the animation key frames so that the CP must only store a small quantity of information and use temporal interpolation. Animation simplification based on frame reduction was achieved by considering a progressive approach.

3.4 Simplification control

As explained in the previous sections, the key component of the processing path 1 is the Simplificator module executed on the server prior to the transmission of the content to a specific terminal on a specific connection. For an animated object the Simplificator adapts the number of vertices, the resolution of the texture and the animation. Together with the improvements introduced by the geometry, texture and animation coding tools with the main objective of reducing the data size, a global quality/bit-rate/execution time control can be obtained over all objects. We developed an intelligent global adaptation tool consisting in finding heuristics for approximately solving an NP-hard knapsack problem. To simplify the problem we searched for the nature of the function between the cost and the different parameters that are controlled by the Simplificator. With a regression coefficient of 93% measured over all the more than 80 objects in the game, the original MPEG-4 file size S decreases roughly tri-linearly with the LODs of geometry (G), texture (T) and animation (A):

$$S = a_G G + a_T T + a_A A + b. \tag{6}$$

Since parameters a_G, a_T, a_A and b are model-dependent, we extract them in the production phase and embed them in the in the “meta” atom of the MP4 file, during the exporting phase. To control the minimum quality that is visually acceptable, we also embed in the MP4 file the following three parameters: minimum number of vertices, minimum texture resolution and minimum number of animation frames. Below these thresholds, the 3D model will be remotely rendered and transformed in an image which will be mapped on a billboard.

Results of the adaptation algorithms for geometry, texture and animation can be visualized at www.MyMultimedia World.com, an on-line repository for MPEG-4 objects.

4 System components: terminals, servers and network

We developed the system to support a variety of terminals, to test and validate the scalability of game content: mobile terminals and PCs, from high-end gaming ones to laptops, and including 3D displays, for which the rendering transformation **R** yields $\{P_i\}^{3D}$. For the latter, lenticular sheets were used to turn a flat, 2D matrix display into an autostereoscopic multiview display able to present the viewer with different images from various slightly different viewing angles. To efficiently address 3D displays that require a high number of views (typically 9 or more), configuration described in (5e) is used, consisting in pre-render the 3D mesh data to an intermediate image-plus-depth format $\{P_i\}^{3D}$. The transformation **R** renders only one viewpoint and provides the depths of the pixels in the computed view to the display; a dedicated processor in the display can render the desired viewpoints at high quality in multiview 3D, as suggested in Fig. 2. $\{P_i\}^{3D}$ format is used for data transmission and, on the terminal, the scene is generated for multiple viewpoints.

On the network side, the architecture is decoupled between the game network and the content delivery network,

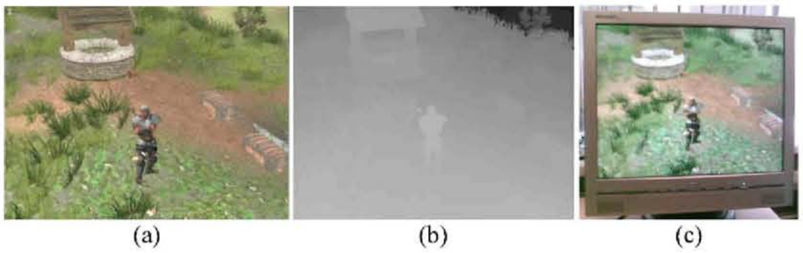

Fig. 2 2D image (**a**) and depth buffers (**b**), and final image on 3D display (**c**)

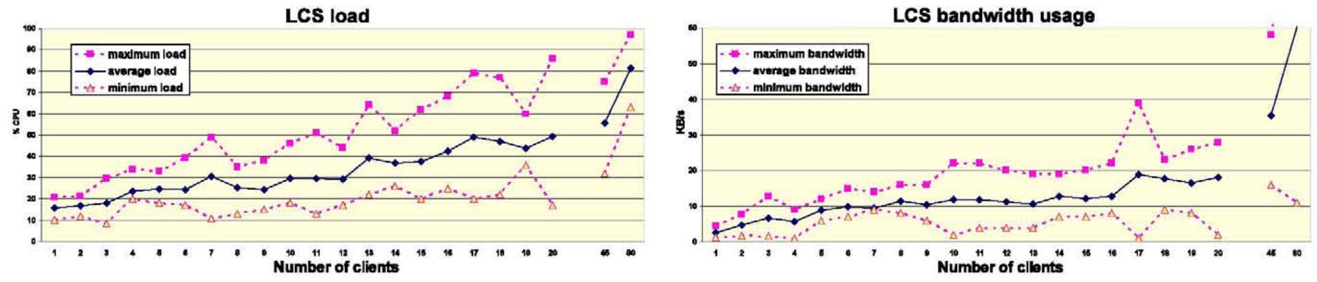

Fig. 3 LCS load and bandwidth as a function of the number of clients

coordinated through a central lobby server. The game network manages a distributed network of game logic servers, called ZGSs (Zone Game Servers). The content delivery network contains a pool of content adaptation and delivery servers, called LCSs (Local Content Servers).

Many of the transformation operations mentioned in Sect. 3 require extensive CPU power and memory. A pure client–server architecture for serving dynamically rendered content does not scale well, and that is why the peer-to-peer model was chosen: the network can contain many heterogeneous ZGSs and LCSs, potentially hosted at the most powerful PCs of the players themselves (a content adaptation server is installed on every client PC, and may be called upon dynamically to start acting as an LCS, depending on overall system conditions).

A deployed networked implementation of the **RCS** model in Sect. 2 enables live update, distribution and adaptation of content. Any game client can create its own content (in standard formats) and add it to the game in real time, by using the ZGS network (for referencing) and the LCS one (for the content itself). All terminals, from high-end 3D graphic PCs to humble CPs, are simultaneously active in the same game, and interacting with each other. The LCSs actively adapt the content to the client characteristics before delivery, using residual capacity available in the node. This adaptation corresponds mostly to the simplification operation $\mathbf{S}$, and is done through the set of simplification tools described in Sect. 3. An LCS, therefore, is an intermediate node, performing an operation akin to $\mathbf{C} \circ \mathbf{S} \circ \mathbf{C}^{-1}$. To improve performance, each LCSs caches the result of simplifications, so that future requests can be served immediately. An automatic load balancing mechanism was also added: the number of clients served by an LCS can change along time as a response to LCS load (result of adaptation tasks) and used bandwidth (result of content delivery). The load produced by adaptation tasks depends on various factors: the content to adapt, the adaptation parameters, or the number of concurrent adaptations. The resulting serve time is in general the sum of the time needed for adaptation plus the time for delivery (except for cached adapted content). Some tests were made under controlled conditions; the values for LCS load and bandwidth usage are represented in Fig. 3, against the number of simultaneous adaptation works being served.

The graphs in the figure show a worst-case situation, in which clients are continuously demanding adapted content (client load is 100%). In the real world, a game client requests content only intermittently, as dictated by the game. To get some insight, we can make some simplifying assumptions, such as requests randomly scattered in time (modeled as the Poisson distribution). Setting then a maximum LCS load of 30%, we would need between 5 and 10 LCSs to be able to guarantee a server availability of 99.9% to a pool of 100 clients requesting adaptation loads between 10% and 40%.

5 Conclusions

Today's multiplayer 3D games often rely on dedicated/proprietary technological solutions for their servers (e.g., massively parallel, brute-force grid computing), and scale down content a priori, according to the bandwidth or rendering power of the "weakest" node in the infrastructure. We opted instead for a completely different paradigm: exploiting the scalability at the level of content, platforms and networks, possibly adapting the content, network and processing load to the distributive resources available over the end-to-end delivery chain. Therefore 4D (animated 3D) content is not stored locally on one single server or local storage medium (e.g., DVD), but is rather distributed over a multitude of servers spread all over the network with adequate load-balancing and fault-tolerance policies, and possibly hosted at the most powerful PCs of the players themselves!

We managed to integrate a chain of content conversion, transmission and rendering technologies into a heterogeneous infrastructure and terminal set, demonstrating real-time interactive 4D content adaptation. We developed a distributive multiplayer 4D game but, more importantly, we developed a framework to develop distributive multiplayer 4D games, or other multimedia applications with heavy and highly variable bandwidth and rendering requirements. And our framework hooks to a complete toolkit of standardized content representation/compression formats (MPEG-4, JPEG 2000), enabling easy deployment over existing infrastructure, while not impeding well-established practices in the game development industry.

References

Martínez, J.M., Morán, F.: Authoring 744: Writing descriptions to create content. IEEE Multimed. **10**(4), 94–99 (2003)

Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: Proc. ACM SIGGRAPH, pp. 209–216 (August 1997)

Taubin, G., Rossignac, J.: Geometric compression through topological surgery. ACM Trans. Graph. **17**(2), 84–115 (1998)

ISO/IEC JTC1/SC29/WG11: a.k.a. MPEG: IS 14496-16, a.k.a. MPEG-4 Part 16: Animation Framework eXtension (AFX) (February 2004)

Avilés, M., Morán, F., García, N.: Progressive lower trees of wavelet coefficients: efficient spatial and SNR scalable coding of 3D models. In: Proc. PCM, Pacific-rim Conf. Multimedia, November 2005. LNCS, vol. 3767, pp. 61–72. Springer, New York (2005)

Said, A., Pearlman, A.: A new, fast and efficient image codec based on set partitioning in hierarchical trees. IEEE TCSVT **6**(3), 243–250 (1996)

Gioia, P., Aubault, O., Bouville, C.: Real-time reconstruction of wavelet-encoded meshes for view-dependent transmission and visualization. IEEE TCSVT (Trans. Circ. Syst. Video Technol. **14**(7), 1009–1020 (2004)

Tack, K., Lafruit, G., Catthoor, F., Lauwereins, R.: Eliminating CPU overhead for on-the-fly content adaptation with MPEG-4 wavelet subdivision surfaces. IEEE Trans. Consum. Electron. **52**(2), 559–565 (2006)

ISO/IEC JTC1/SC29/WG11: a.k.a. MPEG (Moving Picture Experts Group): IS (International Standard) 14496-2, a.k.a. MPEG-4 Part 2: Visual (February 1999)

Preda, M., Tran, S., Prêteux, F.: Adaptation of quadric metric simplification to MPEG-4 animated object. In: Proc. PCM, November 2005. LNCS, vol. 3767, pp. 49–60. Springer, New York (2005)

Preda, M., Jovanova, B., Arsov, I., Prêteux, F.: Optimized MPEG-4 animation encoder for motion capture data. In: Proc. Web3D Symposium, pp. 181–190 (April 2007)

12. Berretty, R.-P.M., Peters, F.J., Volleberg, G.T.G.: Real time rendering for multiview autostereoscopic displays. In: Proc. Stereoscopic Displays and Applications Conf. SPIE, vol. 6055, pp. 208–219 (January 2006)