Towards a Structured Selection of Game Engines for Virtual Environments

Martin Westhoven(✉) and Thomas Alexander

Fraunhofer Institute for Communication, Information Processing and Ergonomics FKIE, Human Factors, Wachtberg, Germany
{martin.westhoven, thomas.alexander}@fkie.fraunhofer.de

Abstract. Development and maintenance of virtual reality engines are coupled with large effort. It is therefore common today, to use existing solutions originating from the entertainment sector. This is often a compromise, since they fulfill individual requirements only in parts, due to their different background. The decision for a specific engine can have a large effect on the effort required to implement own functionality. The number of existing engines further complicates decision making. To enable a comprehensible and replicable decision making, we propose a structured selection process. In a multi-step approach, first the requirements and criteria for comparison are identified and analyzed. A pre-filtering is then used to select a feasible number of engines which are then compared in detail.

Keywords: Game engines · Virtual environments · Serious gaming

1 Introduction

Virtual Reality (VR) is usually associated with realistic computer-generated worlds and natural interaction with them. It is characterized by close, often multimodal, sensory contact with the artificial environment. When consistent, this allows for experiencing so-called presence. The basic idea is nothing new and numerous studies were published since the 90's. VR was often connected to complex and cost-intensive technologies, e.g. head-mounted displays or projection rooms, as well as custom software. During the last decade, the availability of low-cost commercial components changed the field, so that today even professional simulation systems use them for generating virtual environments. Often, games optimized for realistic rendering or their underlying engines are employed. Compared to custom in-house solutions, the rendering quality can thus be raised while simultaneously lowering the development cost.

Of the many genres of video games, those from a first person view are especially suited, as they often already aim for realism in the depiction of and interaction with virtual environments. Of those, 3D-action games focus on fighting and highly dynamic-sometimes even emotionally demanding-missions. Games such as Quake,[1] America's Army,[2] Operation Flashpoint,[3] Unreal[4] and FarCry[5] revealed the potential for professional

[1] © id Software, Mesquite, TX, USA.
[2] © US Army, MOVES Institute, Monterey, CA, USA.
[3] © Codemasters, Warwickshire, UK.
[4] © Epic Games, Raleigh, NC, USA.
[5] © CryTek GmbH, Frankfurt a.M., GER.

R. Shumaker and S. Lackey (Eds.): VAMR 2015, LNCS 9179, pp. 142–152, 2015.
DOI: 10.1007/978-3-319-21067-4_16

applications, as the development and maintenance cost of state-of-the-art in-house solutions was continually growing [1, 2].

The integration into professional simulator systems is sometimes also called Serious Gaming. Serious Gaming stands for the use of technologies and concepts originating from the entertainment sector for serious purposes. Playful aspects can, but in most cases do not have priority [3].

At first, such software was and is unsuited to be employed for professional and "serious" purposes. Empirical studies demand objectivity, validity, reliability and practicability. Furthermore, there exist several other, more technical requirements, such as controllability, reproducibility and the ability to protocol events. Aiming primarily at entertaining its users, game software often does not take these into account. They therefore require modifications which are coupled with additional effort. This has to be considered when performing respective studies, leading to high demands regarding the software to be chosen.

A requirements-oriented choice for an engine can reduce the effort required to accommodate for missing features and can therefore also result in a more preferable cost-benefit ratio. However, due to the large number of existing engines, this is again coupled with a large effort. Lewis and Jacobsen reported over 600 commercial engines in 2002 [4], while the database of devmaster.net[6] currently counts 370 engines, 283 of whom are actively developed and maintained. An exhaustive comparison of every engine for some intended purpose is thus normally not possible. Solutions which were successfully used in previous projects are often used instead. This is not always optimal and especially not possible for novices to the field. To lower the effort for a comparison to a feasible amount, the number of engines has to be reduced previous to a detailed consideration. Methodologies for selecting an engine can help making appropriate decisions. To maximize the benefit, the requirements of the field of application should also be taken into consideration.

This work focuses on the criteria on which a selection can be based and the measure for comparison. The selection criteria are closely scrutinized to identify those, which can be used in a pre-filtering step with low effort. The selection criteria are evaluated by a measure for the effort required for modifications [5]. First, relevant work is shown to locate this work in the field of research. Second, the approach is presented and finally an example is given to illustrate and discuss it.

2 Relevant Work

Previous work already tried to identify criteria for comparison of game engines: One approach is to divide engines into so-called functional blocks, which comprise of rendering, audio, physics and AI [6]. The number of existing engines is recognized as a problem and addressed by leaving out engines still in early development or engines missing essential features like e.g. audio support. Also, the availability of editors and their quality is considered. Finally though, direct monetary cost and popularity are used

[6] http://www.devmaster.net/devdb/engines, last accessed February 9th, 2015.

to decide which engines to compare. The detailed comparison focuses on modifiability, content and gameplay of the engines.

The selection of engines for displaying content in a first-person view is described in [7]. Criteria for initial filtering are actuality and popularity, especially in scientific work. The selected engines are compared in the topics of rendering, APIs, documentation and support.

When generating dynamic 3D-environments, the underlying engine is an important factor [8]. The selection criteria for an open-source engine are therefore discussed. Community support, documentation and licensing are of primary concern, whereas modifiability aspects form a secondary criterion.

The comparison of engines regarding their suitability for architectural design is discussed in [9]. Code accessibility is informally used to evaluate the engines from a developer's view and the editing tools to evaluate from an end user's view.

A framework for comparison of engines is presented by [10]. It identifies criteria for comparing game engines for use in serious games. Technical elements are distinguished from non-technical ones. Also, requirements from a gaming perspective are considered, e.g. support for creating a narrative. The respective elements are compared in free text and the results are visualized in tables showing the supported features.

Another work focuses on support for consumer VR technologies in game engines [5]. They argue, that with enough effort every feature can be implemented in any engine and that therefore support for single features is not very meaningful on its own. A so-called Level of Support is determined to assess the amount of effort required to modify an engine respectively. Qualitative and general properties are also considered if they allow for better estimating the suitability of an engine.

The aim of this work is to combine the different approaches for selecting game engines to enable a methodic procedure. The results of a requirements analysis are used as a starting point. The Level of Support notion is then used for technical detail comparison on basis of the collected filter and selection criteria.

3 Selection Criteria and Requirements

The identification of important criteria is essential for the selection of a suitable engine. Expanding [10] 's distinction of technical and non-technical elements, the criteria are divided into software-related, development-related and acquisition-related criteria. Table 1 gives an exemplary overview of possible criteria. Software-related criteria are those features the engine provides. They encompass development-related criteria as a whole or part-wise. The latter describe all those criteria, which have an influence on the development with the engine and therefore on the effort to modify it as well.

An example for a criterion which is development-related, but not software-related, is the experience of one's own developers. Finally, acquisition-related criteria are those criteria which are directly coupled with the availability of the engine. As a basis for identifying the most important criteria, the framework of [10] is extended by relevant criteria from [6] and [9].

Since criteria for comparison are defined directly through the requirements of the intended application, they cannot entirely be specified a forehand. The presented

Table 1. Classification of relevant example criteria for engine selection

Software	Development	Acquisition
Audiovisual display	Accessibility	Accessibility
Rendering	Documentation	Licensing
Animation	Support	Cost
Sound	Code Access	System Requirements
Streaming	Introduction Effort	
Functional display		
Scripting		
Supported AI		
Physics engine		
Event handling		
Combinability		
Component export/import		
Development tools		
Networking		
Client-Server		
Peer-to-Peer		
Heterogeneity		
Multi-platform support		

criteria can be used as a starting point for requirements analysis, but also during the analysis can new criteria for comparing the engines be found. Analogous it can be found, that specific criteria do not have to be considered in regard to the intended application. As an overarching requirement stands the goal to minimize the effort to satisfy the other requirements. An important aspect is therefore the possibly existent experience of a development team with the respective engines, the used programming languages and the useable development kits, as this can significantly influence the modification effort.

4 Pre-filtering

Exclusion criteria can offer a fast way for an initial filtering. They are therefore denoted as filter criteria. Especially easily examinable criteria such as the acquisition cost are suited for this use. To the contrary, a detailed comparison of rendering techniques would be ill-suited. It is furthermore favorable to examine such criteria, which cannot be addressed by modification or if so, only with difficulties.

Through this procedure, mainly accessibility-related criteria are considered, acquisition-related criteria in particular. Obviously, only those engines should be examined, which can be obtained after all.

The relevance of development-related criteria stems from their influence on the modifiability of an engine. Requirements of software-related criteria can be met with sufficient modification effort, the latter strongly depending on the development-related criteria, such as editor tools, documentation and support.

Previous work often uses fuzzy filtering criteria like breadth of use, innovative features, modularity and actuality. The breadth of use or popularity typically relates to the suitability of the application fields of the engine. This can be a signifier for the fulfillment of general requirements and can be checked quickly. Modularity can also be checked quickly, at least on the surface. It can yield insights on the effort required for modifications, making it a possible filter criterion. Modularity can also be extended afterwards, as is the case with e.g. the CryENGINE[7]'s community-developed plugin system.[8] The implementation of innovative features however depends on their respective complexity and is as such not generally suitable. Actuality on its own does not provide much information, since there are still hundreds of actively developed engines. Answering the question if an engine uses recent technology or how difficult it is to implement the technology into less recent engines requires a thorough examination and should therefore be performed only after a first filtering step.

Rarely mentioned is the possible experience of developers as a filter criterion. It can be checked quickly and be used as a supporting filter criterion.

How many engines should be compared in detail is dependent of the resources available and the requirements given. Other work compares between three to six engines [5–7, 9, 10], which also to our own experience amounts to a manageable effort.

5 Detail Comparison

Having gathered the requirements and chosen the engines to compare, the final step is to determine which engine fulfills most of the former. This can be measured by the approximate effort for modifying the engine towards satisfying the requirements. The examination of the support of graphics engines for VR technologies in [5] uses the following scale of effort:

1. Re-Engineering
2. Source-code modification
3. In-engine programming (scripting, plugins)
4. Graphical in-engine programming (e.g. Node graphs)
5. Natively supported

The respective items describe entry points for modifying an engine. A higher score is achieved when an engine fulfills the requirements or is easily modifiable to do so. It is obvious though, that the step from modifying source code to reverse engineering an engine is a huge leap. The scale can be used with software- and development-related criteria. Acquisition-related criteria, like e.g. the licensing cost, could also be measured with the modification effort, but the details of this measurement will likely be located in jurists' and business economists' areas of expertise. They will therefore not be considered in this work.

[7] © CryTek GmbH, Frankfurt a.M., GER.

[8] http://hendrikp.github.io/Plugin_SDK/, last accessed: January 30th, 2015.

6 Case Example: Requirements

A short example illustrates the method's usage: The background is the selection of a game engine for a laboratory environment. Apart from a setup with a treadmill and a large screen projection (see Fig. 1), the engine is to be used for immersive VR with head-mounted displays (HMD) with a large field of view (FOV) as well.

Fig. 1. Laboratory setup with treadmill and a large screen projection

The most common approach to create a large FOV as of now is to use optical lenses, which require the software to compensate for induced distortion. An example is the well-known Oculus Rift.[9] To avoid breaking the immersion, the presentation should be as close to reality as possible. As this is a rather imprecise requirement, this is broken down to the support for rendering techniques. The techniques of Tesselation [11], Screen Space Sub-Surface Scattering (SSSSS) [12] as well as Screen Space Directional Occlusion (SSDO) [13] are used exemplarily, since a full examination of rendering techniques would be well beyond the scope of this work. As further requirements, a high modifiability to include experimental components and networking capabilities to allow for cooperative scenarios are given.

Until now, the CryENGINE has been used and it is to be checked if this is really the best choice for this application. The development team therefore already has experience with the engine and the development tools. To determine the suitability of other engines, documentation, support and the learning curve of the engines are thus essential criteria.

7 Case Example: Filtering

To reduce the effort required for the filtering step, the criterion of popularity was used. Popularity is easy to assess by querying so-called modding websites, centered on everything needed to modify existing games, but also encompassing the means to build

[9] Oculus VR, Inc., Irvine, CA, USA.

entirely new games. We used the site Mod DB,[10] which listed the Unity[11] engine as the top engine at that time. The detail comparison was therefore between CryENGINE and Unity. There exist several other sites which list engines in a more or less detailed manner. Especially for novices to the field, these can give a good overview of existing engines.

8 Case Example: Detail Comparison

The support for immersive VR-HMDs is examined first. To non-natively compensate for the previously mentioned distortions often caused by the incorporated optics, an access point to the rendering pipeline is most helpful. CryENGINE offers this in combination with a rather expensive license variant and else restricts access. In the latter case, there is no possibility to include custom-built shaders and the rendering engine cannot be accessed directly. A computationally expensive approach is to modify the backbuffer after the engine finishes the rendering step. This is a source-code modification (2), whereas Unity offers native support (5) for most common devices.

Both CryENGINE and Unity implement Tessellation, the division of complex polygons into primitive surfaces, as triangulations. Also, both support the roughly approximated and normal-based Phong Tesselation [14]. In conclusion, both natively implement this feature (5).

The rendering of the diffusion of light in semi-transparent bodies, like skin or gems, can be computed in screen-space for real-time applications and is known as Screen-Space Sub-Surface Scattering. CryENGINE natively supports it (5), whereas in Unity a custom shader has to be implemented or bought (3).

Screen-Space Directional Occlusion extends the technique known as Screen-Space Ambient Occlusion (SSAO) [15], which allows to approximate light spread on non-reflexive surfaces. SSDO offers the possibility to include all light sources into the computation of Ambient Occlusion for generating shadows. Again CryENGINE natively supports the feature, while for Unity a shader has to be implemented (3).

Both engines offer ways to use third-party software libraries programmatically, thus making it easy to modify the engines. CryENGINE requires a source-code modification (2) and Unity allows adding libraries via the editor and using accessing them with scripts (3).

Networking support is offered by both engines natively for up to 32 clients (5).

Table 2 sums up the scores for the software-related criteria.

Documentation and support of the CryENGINE are organized together with tutorials and a support forum on a community website.[12] In addition there exist many unofficial video tutorials on both novice and expert topics. The documentation encompasses the development tools as well as tips on generating content like textures

[10] http://www.moddb.com/engines/top, last accessed: October 21st, 2014.

[11] © Unity Technologies, San Francisco, CA, USA.

[12] http://www.crydev.net, last accessed February 5th, 2015.

Table 2. Summary of the scores for software-related criteria

Criterion	CryENGINE	Unity
VR-HMD Support	2	5
Tesselation	5	5
Screen Space Sub-Surface Scattering	5	3
Screen Space Directional Occlusion	5	3
Modifiability	5	5
Networking	5	5
Sum	24	24

or 3D-models with external software. Against additional payment, licensees can also book courses and can contract the support team for development work.

Unity offers textual and video-based tutorials on their site,[13] organized both using keywords and topics. There is a separate documentation on scripting and unofficial books and tutorials exist as well. Technical support is generally free and the free version of the engine attracted a relatively large community which can be reached through a forum in addition. Sample projects help getting started quickly and the used components are available for free use.

To assess the learning curves, two simple scenarios were generated without prior knowledge on any of the two engines by a new colleague (see Fig. 2).

Fig. 2. Scenes in CryENGINE (left) and Unity (right)

Getting started with Unity is easy. Modelling terrain with satellite images and height maps only takes several minutes and adjusting height and textures or adding vegetation is intuitive. Placing objects is equally easy and imported assets are problematic only in occasions where e.g. automatic texturing fails due to non-standardized texture files. Since animations and scripts can be added to objects via drag and drop and

[13] http://unity3d.com, last accessed February 5th, 2015.

the parameters are added in editor, the learning curve only gets steeper when more complex custom scripts or shaders have to be implemented.

Terrain generation is equally easy in CryENGINE. Only importing satellite images is more complicated, since textures for different levels of detail have to be generated which requires a manual rasterization of the original image beforehand. Customizing height and texture as well as adding vegetation is also easy. The placement of objects is comparable to Unity, but importing requires native CryENGINE formats which always results in a detour via external modelling tools. Analogous to Unity, adding existing scripts or animations is rather easy and the learning curve steepens only when implementation work for custom scripts or source-code is required.

9 Case Example: Summary

Software-related criteria can be assessed easily. Both engines have the same score for the chosen criteria, since CryENGINE fulfills more requirements, but in Unity the respective modifications can be achieved more easily with in-engine tools. Concerning documentation, there exist comparable resources. The free support and more active community of Unity however are a notable advantage.

Taking the examined criteria into account, Unity would thus be suitable for a fresh start. Existing experience with CryENGINE however influences the ability to modify the engine as well as the ability to help new colleagues becoming acquainted with it. Switching the engine from CryENGINE to Unity would therefore rather result in more cost than benefit.

10 Discussion and Future Work

To enable a more structured and comprehensible process for selecting game engines, a multi-step approach was presented. It consists of identification of criteria for comparison and requirements, a pre-filtering step and a detail comparison.

The identification of comparison criteria is based on other work and is only exemplary, since requirements analysis for the specific application is always needed. Nevertheless it is a starting point which can be verified or modified by a follow-up requirements analysis. Categorizing the criteria helps in the decision which criteria can be used to filter the initial set of engines and which should be used in detail comparison. The class of development-related criteria overlaps with software-related criteria. It is planned to further examine this class to differentiate which of the criteria can actually be modified by software means.

Deciding which criteria to use for filtering depends on the requirements. It is thus difficult to offer more than general hints for their selection. Several criteria from other work are summarized which are mostly rather imprecise, but can be checked quickly. An example would be the popularity of engines. When time-wise possible, more precise criteria should be used to avoid dismissing e.g. less popular but perfectly suitable engines at this stage.

For detail comparison, we propose extending the score system for modification effort from [5] to all criteria which are software-dependent. This would allow for comprehensible scores for feature support. Some criteria however are ill-suited for these scores. This encompasses part of the development-related criteria as well as the acquisition. It would be desirable to also use a score system here, but for all practical purposes this will probably not be practicable. For example, the question arises as to how one could compare the documentations of engines. It is possible that work regarding document analysis can help in this case, which should be further investigated. Until then, a qualitative assessment remains the most practical solution, even if the results are somewhat diffuse.

The example illustrates the desirability of score systems throughout the comparison, since it would allow for weighting single criteria or even entire engines. Single criteria could be those of special importance for the intended application, while weighting an entire engine could be used to take the experience of developers with it into account. It would also be possible to use the weighting a posteriori, e.g. to examine which engine would be most suitable when leaving one or more aspects out.

References

1. Robillard, G., Bouchard, S., Fournier, T., Renaud, P.: Anxiety and presence during VR immersion: a comparative study of the reactions of phobic and non-phobic participants in therapeutic virtual environments derived from computer games. CyberPsychol. Behav. **6**(5), 467–476 (2003)
2. Lepouras, G., Vassilakis, C.: Virtual museums for all: employing game technology for edutainment. Virtual Reality **8**(2), 96–106 (2004)
3. Susi, T., Johannesson, M., Backlund, P.: Serious Games: an Overview. IKI Technical reports, HS-IKI-TR07-001, p. 28. Institutionen för Kommunikation och Information, Skövde, Sweden (2007)
4. Lewis, M., Jacobson, J.: Game engines. Commun. ACM **45**(1), 27 (2002)
5. Peek, E., Wünsche, B., Lutteroth, C.: Virtual reality capabilities of graphics engines. In: Skala, V. (ed.) WCSG 2013: Communication Papers Proceedings: 21st International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision in co-operation with EUROGRAPHICS Association, pp. 40–48. Václav Skala – Union Agency, Pilsen (2013)
6. Marks, S., Windsor, J., Wünsche, B.: Evaluation of Game Engines for Simulated Surgical Training. In: Proceedings of the 5th International Conference of Computer Graphics and Interactive Techniques in Australia and Southeast Asia, pp. 273–280. ACM, New York (2007)
7. Trenholme, D., Smith, S.P.: Computer game engines for developing 1st-person virtual environments. Virtual Reality **12**(3), 181–187 (2008)
8. Catanese, S.A., Ferrara, E., Fiumara, G., Pagano, F.: Rendering of 3D dynamic virtual environments. In: Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques, SIMUTools 2011, pp. 351–358, ICST, Brussels, Belgium (2011)
9. Sarhan, A.: The Utilisation of Games Technology for Environmental Design Education. Ph. D. thesis, University of Nottingham, UK (2012)

10. Petridis, P., Dunwell, I., Panzoli, D., Arnab, S., Protopsaltis, A., Hendrix, M., de Freitas, S.: Game engines selection framework for high-fidelity serious applications. Int. J. Interact. Worlds **2012**, 1–19 (2012)
11. Rockwood, A.: A generalized scanning technique for display of parametrically defined surfaces. Comput. Graph. Appl. IEEE **7**(8), 15–26 (1987)
12. Jimenez, J., Sundstedt, V., Gutierrez, D.: Screen-space perceptual rendering of human skin. ACM Trans. Appl. Percept. (TAP) **6**(4), 23 (2009)
13. Ritschel, T., Grosch, T., Seidel, H.-P.: Approximating dynamic global illumination in image space. In: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games, pp. 75–82, ACM, New York (2009)
14. Boubekeur, T., Alexa, M.: Phong tessellation. In: ACM SIGGRAPH Asia 2008 Papers, p. 141. ACM, New York (2008)
15. Bavoil, L., Sainz, M., Dimitrov, R.: Image-space Horizon-based ambient occlusion. In: ACM SIGGRAPH 2008 talks, p. 22. ACM, New York (2008)