The ExtReAM Library: Extensible Real-time Animations for Multiple Platforms

Pieter Jorissen, Jeroen Dierckx and Wim Lamotte

Hasselt University
Expertise Centre for Digital Media
and transnationale Universiteit Limburg
Wetenschapspark 2, BE-3590 Diepenbeek

Interdisciplinary institute for
BroadBand Technology (IBBT)
Expertise Centre for Digital Media
BE-3590 Diepenbeek, Belgium

Email: {pieter.jorissen, jeroen.dierckx, wim.lamotte}@uhasselt.be

***Abstract*— We introduce a dynamic, platform-independent framework for computer animation. The *ExtReAM* library is constructed around an object-oriented core that can easily be extended. The core system provides functionality for managing objects and plugins. The plugins on the other hand are used to perform more specific tasks such as loading and animating objects using different animation techniques. Different plugins can be used for similar tasks on different platforms and they need to be loaded only when their functionality is required. Easy integration of our system into applications was also one of the main design objectives. In this paper we present the main architecture and motivate the most important design decisions. Furthermore, we show results of two applications demonstrating the use of the *ExtReAM* library. The first application allows animated worlds to be controlled on desktop platforms while the second one allows animated scenes to be displayed on a PocketPC mobile device.**

***Index Terms*— 3D Computer animation, games, multiple platform development, system architectures.**

I. Introduction and Motivation

3D computer animation has been one of the key research areas in the movie, gaming and Virtual Reality (VR) communities for several years now. Despite these efforts, there are still several problem areas and the growing demand for more realism in animation still exists. As a result, a lot of research is focusing on improving very specific areas such as realistic human motion simulation, cloth and hair rendering, physical simulations and so on. Simultaneously, a lot of new gaming devices have entered the market over the last decades. Examples include Xbox, Playstation, Nintendo Game Cube and so on. These are all equipped with special 3D hardware and enough memory and processing power to show high resolution interactive 3D graphics. Also, due to recent advances in processing power and memory capacity, small portable or handheld devices such as PDAs and smart phones are currently also capable of supporting graphical user interfaces with audio and video playback. Some of these have already been equipped with special 3D hardware as well, making them suitable for more interactive 3D animated applications and games. So it is clear that the number of platforms capable of displaying interactive 3D graphics is growing rapidly.

Many games and software components have been, and are still being developed for specific platforms. Since development time for games is a critical factor, not much attention or money is usually spent on portability or extensibility of game engines or their parts. As a result, many platform specific SDKs, libraries, game and animation engines have been developed over the years. Some specific applications or games are being ported to other platforms such as PDAs or smart phones, but since little or no consideration for these platforms was given during the design phase, the porting process is usually difficult and time-consuming. However, since more and more game developers are creating games for more than one platform, there is a trend toward more multiple platform or *platform-independent* components. In research and other less commercial areas some open platform-independent animation libraries and SDKs have been studied and developed. However, these systems are usually limited to platforms such as MS Windows, Linux and in some cases MacOS, which we consider to be desktop platforms. Other platforms are usually considered to be too specific or non-relevant by the developers. On the other hand, most libraries or engines for 3D games or animation are not open source or extensible in any way. Consequently, application developers can only work with the provided functionality. Some open source projects do exist, but adding new animation techniques is often hard and unsupported.

The *ExtReAM* library, presented here, was developed with the goal of providing application developers with a 3D animation library that is easy to integrate, work with, and extend with new animation techniques, without recompiling the core. By providing a very lightweight core library and moving the animation techniques to separate plugins, the library can be used on many platforms and in all kinds of applications. Applications can even provide different plugins for different platforms if needed. In this way, application developers can create many reusable building blocks to construct an application with.

The remainder of this paper is structured as follows: section II presents the related work in the areas of 3D computer animation techniques and mentions the most important animation systems that exist today. In section III the main architecture of the *ExtReAM* library is explained. Special attention is given to the plugin and object management systems since these are the most important parts of the core library. Section IV describes several plugins that were created to test the system and provide basic animation functionality. Section V then shows some of the results that were achieved with our system. Two applications were developed and tested using the *ExtReAM* library, one for controlling animated scenes on a

desktop computer and another one for showing animations on a PocketPC device. The paper ends with some conclusions and an overview of the future work.

II. Related Work

Many techniques exist for animating 3D objects. The most well known are *kinematics* and *dynamics*. Kinematic techniques are concerned with explicitly transforming the different parts of the animated objects. Important kinematic techniques include keyframe animation and inverse kinematics [1], [2]. Motion capture data can also easily be used to control an animated figure using kinematics. Motion graphs [3] use a database of kinematic motions to automatically calculate transitions between keyframes. Dynamic, in contrast to kinematic, animation techniques use physical forces and laws to simulate the object movements. Since creating physical controllers for complex articulated 3D characters is not a trivial task [4], dynamic animation techniques in games are usually limited to ragdoll physics [5]. For example, when in a first-person shooter somebody is shot, the user looses control over his character, and the physics engine takes over, creating a realistic body response. Hybrid techniques have also been investigated in the past [6], [7], and are still being studied now. The main goal usually includes achieving more control over dynamically animated characters. However, attempts to adjust kinematic motions with dynamic effects to improve realism and variation have also been explored.

Over the years, many systems have been developed for creating animations using these techniques. The best known are the commercial modeling packages that are mainly concerned with kinematic techniques. The most important ones include 3D Studio Max [8], Maya [9] and Blender [10], an open source project. These systems can be used to create animated characters and movies, using their own built-in techniques. However, the main focus here is on the modeling task. The generated models can be integrated into applications and the predefined animation data can be used to animate the objects, but animations can not be changed unless the application has its own animation control. Endorphin [11] is another commercial package. It uses *dynamic motion synthesis* and adaptive behaviors to create very realistic animated 3D characters. Dance (Dynamic Animation and Control Environment) [12], on the other hand, is an open and extensible framework for computer animation focused on the development of physically based controllers for articulated figures. These systems provide the tools to create physically simulated animations but are not suited to be integrated into other applications.

Animation libraries, on the other hand, are specifically designed to be easily incorporated into other software. Cal3d [13], for example, is a skeleton based 3D character animation library written in C++ in a platform- and graphics API-independent way. It can easily be integrated into different applications and provides basic skeleton animation techniques such as forward kinematics, keyframed animations and animation blending. However, Cal3D is limited to these animation techniques since it was specifically designed for skeleton animation and provides little means to extend its functionality. Granny 3D [14] is another commercial animation system that can easily be integrated into applications. It has powerful support for all kinds of skeleton animations and is available for several platforms. Extensibility is however hardly provided.

In contrast to these previously mentioned systems, the *ExtReAM* library was developed to be an animation library that can easily be extended with new animation techniques and is easy to integrate in all kinds of applications, on all kinds of platforms. We realized this by implementing a very lightweight, platform-independent core system. Animation and simulation techniques are added through plugins that can be used by the system. Plugins can easily be created by developers due to the object oriented design.

III. System Overview

The *ExtReAM* library contains a very lightweight core that is primarily responsible for object management and registering plugins and their components. Furthermore, it has a built-in event and command system that can be used by the plugins. The more platform-independent and resource consuming tasks such as file loading, object creation, command handling and the animation techniques are left to be implemented in the plugins. This results in a very flexible system that can be used in all kinds of applications and on many different platforms.

Fig. 1 shows how the *ExtReAM* library is used in an application. The application chooses which plugins and objects to load. A Plugin can be loaded into memory for just as long as its functionality is required. As a result, it will only consume resources when necessary. Because the *ExtReAM* system is an animation library only, rendering is not included. All the necessary data for rendering can however easily be retrieved from the system. The application has to step the library every time frame and can execute commands through the *command handler*. Interaction with the actual plugins is completely transparent.

A. Plugin System

Plugins are used extensively in the *ExtReAM* library. A plugin is a "building block" on top of the core system. Plugins can perform common tasks like loading mesh files or specific tasks like animating a skeleton-based character. As plugins are one of the main components of the system, we assured that creating a new plugin is as simple as possible. As a result of the object-oriented design, in order to create a new plugin, the developer must only derive from a couple of classes and implement a few methods. More specifically, it involves deriving from the `Plugin` class and implementing a `start` and `stop` method.

The core system internally uses a *plugin manager* to start and stop plugins and to get information from certain plugins. It also ensures that plugin dependencies (other plugins that a certain plugin relies on) are started first and that plugins are not loaded more than once.

B. Plugin components

To extend the core animation system, using the principles of object-oriented design, the *ExtReAM* library provides some

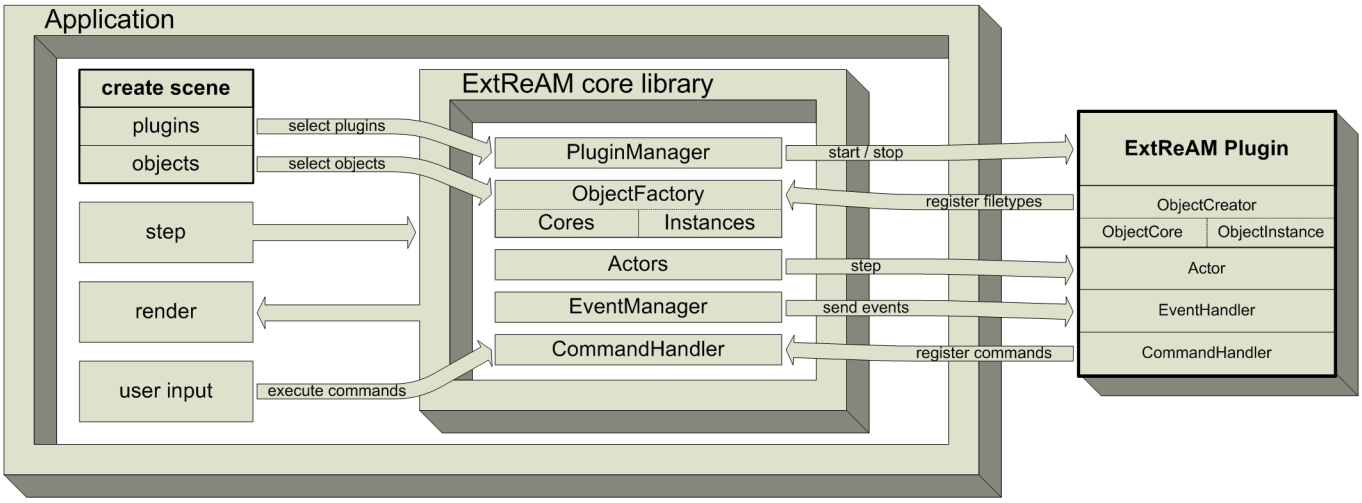


Fig. 1. This figure gives an overview of how the *ExtReAM* library can be used in an application.

abstract base classes that can be used in plugins in a standard object-oriented way. Here we give an overview of the different base classes and explain how to use them.

To provide a new *object type* in the *ExtReAM* system, three classes have to be implemented. How the actual objects are managed in the system is explained in section III-C.

- `ObjectCore`: an object core is generated from the data that gets read from file. From this core data, one or more *object instances* can be created. The instances can share the core data if desired, so redundant data can be minimized.
- `Objectinstance`: an object instance is an actual entity in the virtual world. It is created from an *object core* and can use that core's data.
- `ObjectCreator`: an object creator can register itself in the *object factory* (a part of the core) with certain file types, and is used to create object cores from file and object instances from these cores.

The rest of the plugin base classes are used to alter the behavior of the animation system:

- `Actor`: actors are used to alter the scene every timestep. The elapsed time is provided every frame when the `step` method is called. We have used actors, for example, to advance the physics system and blend skeleton-based animations (see sections IV-A and IV-B).
- `EventHandler`: this class has to be implemented if a plugin needs to react on certain events in the system, like objects being added or removed. An example is our *rigid body simulation* plugin (see section IV-A), where a physical shape is automatically created when an object instance is added, and deleted when the object instance is removed from the system.
- `CommandHandler`: through this class, a plugin can register the commands it can handle. By using commands, the coupling with plugins and the actual application can be very loose (the application does not have to know anything about the plugins). The core system has built-in commands to start or stop plugins, add or remove objects and alter their position or orientation. Examples of registered commands can be found in the *KeyFrameController* plugin (see section IV-B.1), where we register commands to start, stop and pause keyframed animations.

C. Object management

The *ExtReAM* system is designed to easily manage object creation and removal in the scene, as efficiently as possible. The library keeps track of *object core*s and *object instance*s. The *object core* contains all necessary data to create one or more *object instance*s. *Rigid objects* are a good example to use this structure: the core contains the geometry to be rendered and the bounding geometry to be used with collision detection. Everything that the actual instances need are a position and an orientation in the virtual world.

Creation of object cores and instances in the system is handled by the *object factory*. The factory automatically chooses the right creator if the user asks to create an object. It makes sure a file is not read twice if that is unnecessary. Plugins provide the actual creators from which the factory can choose from.

IV. Plugin examples

A. Rigid Objects, Bodies and Physical Simulation

As a proof of concept, one of the first plugins that we implemented was one for *rigid body simulation*. To result in the most flexible solution, we actually created several plugins that can be used separately.

First of all, we implemented the *RigidObject* plugin, which holds a very simple object type: "RigidObject". The core has a geometry and a bounding box, and the instances have a position and an orientation. We also made several plugins to load different file formats such as ply, 3ds, Ogre [15], etc. These file loaders form different *object creators*. Then, we created a *Physics* plugin, that encapsulates the *Ageia PhysX* [16] engine. The plugin has an actor that advances the physical scene every timestep, and some physics specific functions,

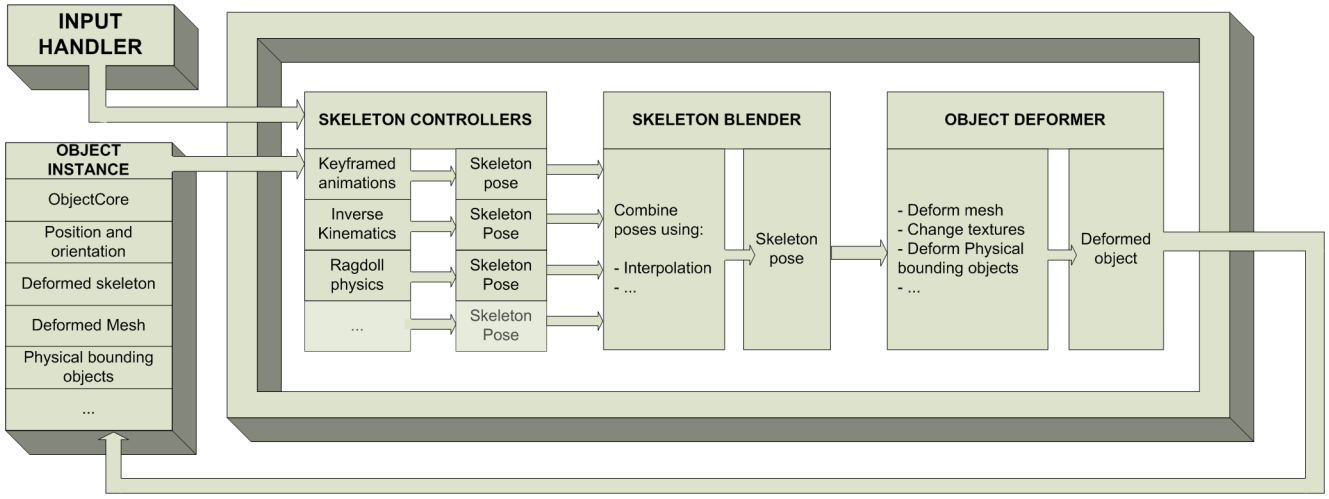


Fig. 2. In this figure an overview of the skeleton animation plugin is shown.

such as creating physical bodies from triangle meshes. Because the *Ageia PhysX* engine is only available for desktop computers, no physics plugin is implemented for PocketPC at the moment. This is a perfect example of plugins that would be different on several platforms: a simple physics plugin could be created for *PocketPC*, that could handle collision detection and other simulation aspects in a much simpler and memory-efficient way.

The rigid body plugin automatically couples physical shapes to rigid objects added to the scene to be used for collision detection and rigid body dynamics. After every time step, the actual positions and orientations are set according to the physical shapes that are simulated.

B. Skeleton Based Animation

Since skeleton animation is one of the most popular character animation techniques, we also created a *skeleton animation* plugin for the *ExtReAM* library. The "SkeletonObject" type has a core that contains a geometry and a core skeleton structure. It also contains a link between the skeleton and the geometry to do *vertex blending* [17]. The instances contain not only a position and an orientation in the virtual world, but also their own deformed skeleton and, if desired, a deformed geometry. Fig. 2 shows how the skeleton plugin is composed.

The bone structure we use is the standard tree structure used for skeleton-based animation: the skeleton has one root bone and every other bone has a parent and a list of children. Every bone has a position and an orientation relative to its parent. Different *skeleton controllers* (which can be implemented as different plugins) can animate the skeleton. Every controller can animate a skeleton pose, and a *skeleton blender* combines the poses of all the controllers into the final skeleton instance. This blender uses interpolation between the final skeleton poses of the different active controllers. The skeleton blender can also apply weights to the different controllers so that one controller can have more influence on the entire or a specific part of the skeleton. An example of when this might be used is when we manipulate an avatar's hand for grasping something using IK while it is performing a keyframed walk animation.

To deform the geometry, an *object deformer* was implemented. This deformer uses the final *skeleton pose* generated by the *skeleton blender* to generate a deformed skin. It uses the standard weighted vertex blending technique [17]. If the deformed geometry is not needed in software, a graphics hardware method could be used to speed up the deformation. Our current system only provides a software implementation of vertex blending, so the same technique can be used on all platforms. Separate plugins could be created for the different platforms, wherein more platform-specific techniques could be used. On desktop platforms for example, the vertex blending could be implemented using *GPU* hardware.

Several skeleton controllers were already implemented for the *ExtReAM* library and we will discuss them here briefly.

1) The KeyFrame Controller: to be able to use standard keyframed skeleton animations, the first skeleton controller we implemented was the keyframe controller. For each skeleton-object core, we hold a list of available animations. An animation consists of a list of *keyframes* (positions and orientations at a specific time) for certain bones of the skeleton. In this way, animations can be defined on a specific part of the skeleton. When an animation is started for a certain skeleton-object instance, a new *animation state* is created that holds the current time and the weight of the animation. The controller calculates a pose for the bones by interpolating between the keyframes of the running animations. A *command handler* is created, that can be used to start and stop animations and to apply weights.

2) The Ragdoll Controller: using the physics plugin (see section IV-A), we implemented a simple ragdoll controller. When an object instance is added to the system, the ragdoll event handler builds an articulated rigid body structure for the physics engine. After every timestep, the skeleton pose is calculated from the movement of the physical objects.

3) The IK Controller: this controller uses the Cyclic Coordinate Descent (CCD [1]) technique to control a part of, or the entire skeleton of an animated character. A goal position can be specified along with the joints that form the IK chain. The joints in that chain will then be positioned and oriented so that the end of the IK chain approaches the goal position

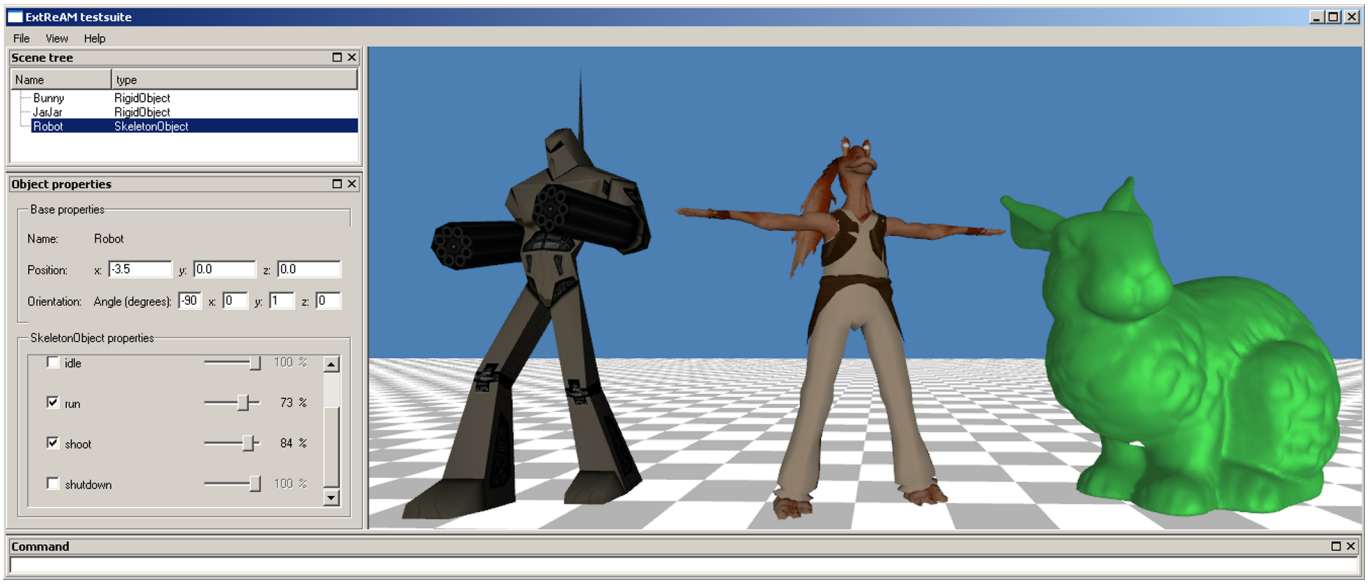


Fig. 3. This shows a screenshot of the desktop test application, using QT and OpenGL. The skeleton object properties widget (bottom left) allows the user to activate animations and set weight factors.

as close as possible. Also, joints can have some orientation constraints specified.

The skeleton animation actor, implemented in the plugin, uses the skeleton blender to animate the skeleton instances every timestep and deforms the geometry afterwards. The resulting deformed object instance can then be rendered and/or used for further simulation.

V. Test results

A. Desktop application

To demonstrate the *ExtReAM* system, a test application was developed for desktop platforms and was tested under MS Windows and Linux. The application provides functionality to load scenes described in a simple *XML*-based scene format that contains the necessary plugins and object positioning. Furthermore, the user can load extra plugins and add, remove and reposition objects at runtime. The interface was developed with the *QT 4.0* library [18] and shows a tree structure of the objects and object specific properties. These properties are specific for every *object type*. The system also supports command handling, so users can send commands to the system for example to invoke core or plugin functionality. OpenGL is used to render the scene.

The application was used to test *rigid body simulation*, *keyframed skeleton animation* and *ragdoll physics* using the plugins described in the previous sections. Using these plugins in the application was straightforward using our library and the plugin system did not introduce any noticeable slowdown. Fig. 3 shows a screenshot of the test application containing a scene with one skeleton-based object and two rigid object models. The user is controlling the keyframed animations of the skeleton object using the properties widget for skeleton objects. We did not optimize the rendering, since this was not part of our research. However, our test application is able to animate, simulate and render several hundreds of objects with interactive framerates.

B. PocketPC application

To test the *ExtReAM* library on PocketPC, we developed a native C++ application using *OpenGL ES* as the rendering backend. We used the Dell Axim X50v PDA, that has hardware-accelerated 3D graphics, so we could test advanced animation techniques without the delays caused by software rendering. The PocketPC application can load the same scenes as the desktop application, but can't load the desktop-specific plugins such as the *rigid body simulation*.

The plugin-based system is an efficient solution for use on devices with limited memory resources such as the PocketPC, because plugins (along with their linked libraries and memory usage) can be unloaded when their functionality is no longer required. The *object creator* plugins for example, that are used to generate object cores and instances from file, are loaded only when necessary, and are immediately unloaded after the objects have been created.

Fig. 4 shows a scene consisting of 18 *rigid objects* (each about 1800 polygons) and one animated *skeleton object* (with 400 polygons and 18 animated skeleton joints). The scene is rendered in real-time (approximately 20 frames per second). The plugins used are the same as in the desktop application. *Vertex blending* causes the most noticeable delays, since the algorithm is completely running in software and therefore not optimized for the device.

VI. Conclusions and Future Work

In this work we presented the main architecture and design considerations of the *ExtReAM* library. The system is built around an object-oriented, platform-independent core that can easily be extended by plugins. While the core system provides functionality for managing plugins and objects, the plugins are

Fig. 4. This figure shows the embedded test application running on a Dell PocketPC device. The robot is an animated *skeleton object* performing the "Walk" animation. The other objects in the scene are all *rigid objects.*

responsible for more specific tasks such as object loading and various animation techniques. Different plugins can be used for different platforms when necessary and plugins need to be loaded only when their functionality is required. This work also presents how easy it is to incorporate the library into applications and create new plugins for the system. Several developed plugins were described and the results of two applications using them, one for desktop and one for *PocketPC*, were presented. The overhead generated by the plugin system does not result in a noticeable slowdown, moreover it is very memory efficient, which is especially interesting use on mobile devices.

In the future, we will use the *ExtReAM* library as a base for researching *physically based character animation*. Furthermore, we have several ideas for using a new, more intuitive and memory efficient bone structure, as well as an optimized vertex blending algorithm for *PocketPC*. The system will also be used in an existing *networked virtual environment* setting. Using the system in several big projects will reveal more of its advantages and weaknesses. Naturally, we will continue to improve the main system. Finally, we will also use *ExtReAM* to investigate interaction in VR environments. Controlling virtual characters in a physically simulated world will be the main subject. Providing realistic force feedback to the user will also be looked at.

ACKNOWLEDGMENT

Part of the research at Expertise Centre for Digital Media (EDM) is funded by the ERDF (European Regional Development Fund), the Flemish Government and the Flemish Interdisciplinary institute for BroadBand Technology (IBBT). We would also like to thank the CVE group of the EDM for their help and support.

REFERENCES

[1] C. Welman, "Inverse kinematics and geometric constraints for articulated figure manipulation," Ph.D. dissertation, School of Computer Science, Simon Fraser University, 1989.

[2] T. Giang, R. Mooney, C. Peters, and C. O'Sullivan, "Real-time character animation techniques," Image Synthesis Group Trinity College Dublin, Tech. Rep. TCD-CS-2000-06, Feb. 2000.

[3] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," in *In Proceedings of ACM SIGGRAPH 02*, 2002, pp. 473–482.

[4] P. Faloutsos, M. van de Panne, and D. Terzopoulos, "Composable controllers for physics-based character animation," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2001, pp. 251–260.

[5] K. Pallister, *Game Programming Gems 5*. Charles River Media, Inc., 2005.

[6] A. Shapiro, F. Pighin, and P. Faloutsos, "Hybrid control for interactive character animation," in *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*. Washington, DC, USA: IEEE Computer Society, 2003, p. 455.

[7] V. B. Zordan and J. K. Hodgins, "Motion capture-driven simulations that hit and react," in *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*. New York, NY, USA: ACM Press, 2002, pp. 89–96.

[8] (2005) Discreet website. [Online]. Available: http://www.discreet.com/

[9] (2005) Alias website. [Online]. Available: http://www.alias.com/

[10] (2005) Blender website. [Online]. Available: http://www.blender.org/

[11] (2005) Natural motion website. [Online]. Available: http://www.naturalmotion.com/

[12] P. Faloutsos, M. van de Panne, and D. Terzopoulos, "Composable controllers for physics-based character animation," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 2001, pp. 251–260.

[13] (2005) The Cal3D project website. [Online]. Available: http://cal3d.sourceforge.net/

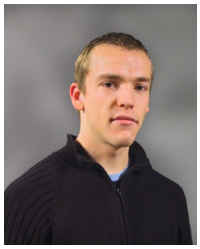
[14] (2005) The granny 3d website. [Online]. Available: http://www.radgametools.com/gramain.htm

[15] (2005) Ogre 3d website. [Online]. Available: http://www.ogre3d.org/

[16] (2005) Ageia physx. [Online]. Available: http://www.ageia.com/novodex.html

[17] J. Lander, "Skin them bones: Game programming for the web generation," in *Game Developer Magazine*, May 1998, pp. 11–16.

[18] Qt. [Online]. Available: http://www.trolltech.com/products/qt/



Pieter Jorissen is a PhD candidate in the Department of Computer Science at the Hasselt University, Belgium. He received his BS and MS degree in computer science from the Limburgs Universitair Centrum in 1999 and 2001 respectively. He is now working as a research assistant at the Expertise Centre for Digital Media (EDM), a research institute of the Hasselt University. His current research interests are in the fields of virtual interaction, collaborative virtual environments, animation and physical simulation.

Jeroen Dierckx graduated in computer science in 2004 at the Limburgs Universitair Centrum, Belgium. He is currently working as a researcher at the Expertise Centre for Digital Media (EDM), a research institute of the Hasselt University. His main research interest is computer animation, with specific interest in real-time autonomous character animation and physically based modeling.

Wim Lamotte received his MS degree in 1988 at the Free University of Brussels and his PhD degree in 1994 at the Limburgs Universitair Centrum. He is currently an assistant professor at the Hasselt University where he teaches computer networks, telecommunications, and multimedia technology courses. His research interests include networked virtual environments, networked multimedia and photo-realistic image synthesis. In these fields, he lead and participated in numerous research projects, both in local and European contexts. Wim Lamotte is member of the ACM (SIGCOMM) and IEEE (Computer Society and Communications Society).