A wireless library for the Nintendo dual screen (DS) game console[1]

Mihai Pâslariu, Nacer Boudjlida
Lorraine University, LORIA, UMR 7503, France

Abstract

Due to the popularity of multiplayer games, all of today's gaming consoles have wireless capabilities. However, in order to write the network related part of such a game, the programmer is required to understand and use a low level and difficult API. We address this issue by creating a networking engine that needs to be both efficient to run on the limited resources offered by a gaming console, but also easy to use in order to shorten development time. In this paper we first explain the need for such an engine, then we continue with some theoretical background and finally we present and discuss our results.

Keywords: Computer Games, Video Game, Multiplayer, Wireless.

1. Introduction

A multiplayer video game is a game where more than one person can play in the same game environment at the same time. Adding a multiplayer component to a Video game involves a lot of work in designing the network model, in implementing communication between devices and in thoroughly testing the game. This leads to increase the development time. A possible solution to the problem is the use of an application independent network engine [5] that can be easily integrated in any game without requiring a lot of extra work. The work which is described hereafter takes exactly this approach, to design a reusable engine that will exploit the wireless capabilities of the Nintendo Dual Screen (DS) and also allows a rapid integration in any application. The end result of this work is a C++ code library.

[1] In Proceedings of the 2013 International Conference on Sport Science and Computer Science (CCCS'2013). December 24-25 2013, Hong Kong. To appear in the International Journal "WIT Transactions on Information and Communication Technologies", ISSN: 1743-3517.

In addition, the engine had to meet the following requirements: *(i)* be conform to the requirements imposed by Nintendo for its game titles, *(ii)* use minimal resources and *(iii)* have a user-friendly interface in order to allow fast development of multiplayer games. The paper is structured as follows: section 2 introduces the problem statement and the formal background of this work, section 3 details the structure and the role of every component in the library we propose, section 4 discusses our results especially from a performance point of view and finally, section 5 includes concluding remarks and further work.

2. Problem Statement and Theoretical Background

2.1 Video Game Consoles

There are a few elements common to all game consoles: controllers, power supply, core unit, game media and memory card. This work focuses on a game console called Nintendo DS. A detail that should be noticed, since it impacted this work, is that the ARM processor [4] in the Nintendo DS does not have floating-point support. So, we had to emulate the floating point operations. Further, the Nintendo DS has built-in 802.11 wireless network connections [1] compatible with the only Wireless Equivalent Privacy (WEP) encryption) using a proprietary protocol.

2.2 Wireless Networks

Wireless networks can be classified into: *managed networks* and *ad-hoc networks*. *Managed networks* always contain a special node (called the access point) that manages communication among other nodes. In managed networks, wireless access has special security considerations so most modern access points come with built-in encryption. An *ad-hoc network* [9] is a network connection method which is most often associated with wireless devices. The connection is established for the duration of one session and requires no base station. Instead, devices discover others within range to form a network. Devices may search for target nodes that are out of range by flooding the network with broadcasts that are forwarded by each node. Connections are possible over multiple nodes (*multihop ad hoc network*). Routing protocols then provide stable connections even if nodes are moving around.

2.3 Interpolation/Extrapolation

In a typical video game, there is one device (a server) that decides the state of the game. All the other participants periodically receive snapshots or updates with the state. When communicating through a protocol where packet loss can happen, the data packets that contain the state may get lost. Loosing packets can cause an incorrect movement of objects on the screen. Even if the packets reach their destination, the delay between sending and receiving them can cause the same effect if it is too high. To partly hide this problem from the user, many games use interpolation and extrapolation algorithms [8, 10].

Interpolation is a method of constructing new data points within the range of a discrete set of known data points [2, 7]. On the other hand, *extrapolation* is the process of constructing new data points outside a discrete set of known data points. It is similar to the process of interpolation but its results are often less meaningful, and are subject to greater uncertainty. The current engine implements interpolation algorithms whose results are discussed in section 4.

3. The Wireless Library

The library has four important components: *ad-hoc* wireless, resource sharing, Internet communication and interpolation (see Figure 1.) In the following, we will have a look at what each component does, the approach used to solve the problem and the reason for it. To simplify the development, many methods in the engine are synchronous, thus they only finish after the operation is completed. Whenever possible, an asynchronous alternative is also available. Furthermore, if a standard component already exists it is included in the engine so the resulting application will have the same look as other Nintendo DS titles. One of such an example is the Internet connection utility, a small application that can scan for an access point and configure the Internet connections.

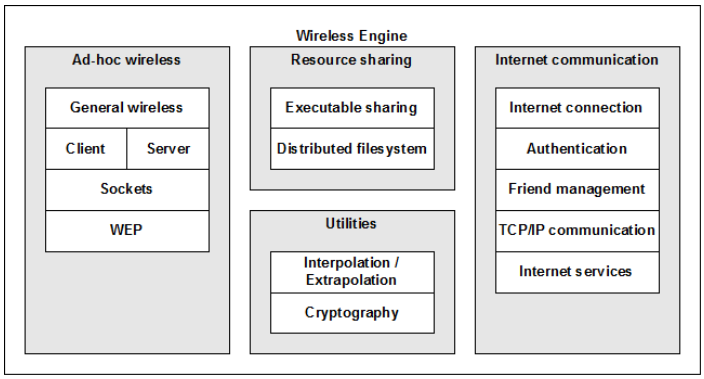

Fig.1: The Structure of the Network Engine.

3.1 The Ad-hoc Networks Component

This component provides routines to allow two or more Nintendo DS devices to connect to each other and exchange messages through the wireless connection. In most ad-hoc networks, all peers have the same functions but, due to restrictions on the DS, we will choose one console to be the server and the rest will be the clients. The server device will control the communication parameters and intermediate all communication between client devices.

The engine contains classes for both server and client. The server broadcasts a beacon message and waits for clients to connect to it. The clients scan for the servers beacon and then establish a connection to the server. After the communication is established, actual communication is performed thanks to

send/receive-like functions. This component also features support for WEP encryption. A client that is not configured to use WEP cannot connect to a server with a WEP encryption but a client configured to use encryption can connect to an unsecured server. The engine also has methods to get the connection strength and to configure wireless parameters like the channel to use, the amount of time to scan a specific channel or the time interval at which to send beacon information.

These communication methods allow a user to write multiplayer games much faster than programming directly with the low level API of the device. As an example, the code below is all what is needed to write a client program:

```
Client client;
ServerInfo server;     Socket socket;
// scan indefinitely for one server
client.scan(&server, 1);
// connect to the server
if ( ! client.connect(&server) )     { // treat error or retry }
// create a socket
socket.startOnClient(&client, SAFE_PORT, &listener);
while (game_is_running)
{   // … game code ..
    // send a message to the server
    socket.send(buffer, size);
    // receive 100bytes
    socket.receiveAll(buffer, 100);   }
// stop client
client.close();
```

3.2 The Resource Sharing Component

This component features the ability to share an executable game code on devices that do not have it, as well as a mechanism to use files through the network as if they were local.

The first part, sharing an executable, is called "Download Play [6] and it enables users to play multiplayer games with other Nintendo DS systems using only one game card. Players must have their systems within wireless range (up to approximately 30 meters) of each other and the guest system to download the necessary data from the host system. However, the "Download Play" feature can only share a binary executable file. To share resource files, we offer classes that allow a device to share its files with other devices. This is especially useful for transferring the rest of the game after the executable was downloaded by the guest device. An application that wants to use this feature typically does the following operations:

1) A Nintendo DS device with a game card starts a game and shares it.
2) Other devices connect to it and start downloading the game: the only executable is transferred on the client.
3) After the download, the game is started.
4) The server DS device can then share its resource files.

5) Client devices, that are now running the game, can retrieve files from the server's file system.

3.3 The Internet Connection Component

This component allows a Nintendo DS device to connect to the Internet and access a wide range of services. It provides the following services:

a) **Internet connection**: This component enables configuring the connection to the Internet, initializing the library and running the connection process.
b) **Authentication** is a component to authenticate the user on the server to facilitate the user's access to the Internet. This authentication is done internally and does not require the user to enter any user-name or password.
c) **Friend management** is a feature that keeps a list of friends. Friend lists are used for matchmaking, to create multiplayer games between players that know each other.
d) **The communication component** offers reliable and unreliable data transmission using protocols from the TCP/IP stack. Internet services} is a component that offers support for services like matchmaking, ranking, storage, etc.

3.4 The Utility Component

We implemented interpolation and extrapolation algorithms to help the application hide the effects of packet loss and of network lag. The interpolation/extrapolation classes make use of templates to let the user decide which data type to use for storing points. The example below shows how to use these algorithms in an application:

```
LinearInterpolator<int, 2> app(HISTORY_SIZE);
int data[10][2];
int interpolatedData[2];
int extrapolatedData[2];
int time;
// add points with timestamps between 0 and 40
for ( int i = 0; i < 10; i++ )
    app.addData( data[i], i * 4 );
// interpolate
time = 6;
app.interpolate( time, interpolatedData );
// extrapolate
time = 50;
app.extrapolate( time, extrapolatedData);
```

The major limitation in the use of interpolation algorithms on the Nintendo DS is the lack of floating point instructions on the processor. Although it is possible to emulate floating point instructions, this is not usually an option since it dramatically decreases performance. Using integer numbers, the performance is good but the disadvantage is that numeric errors can frequently occur. Most of

the algorithms work with both floating point and integers if the values of points and timestamps are in certain documented interval. Another offered feature is the physics interpolator. Given the fact that many games nowadays have a physics engine and most in-game objects move according to the laws of Newtonian physics, we created a new type of interpolation, that tries to guess the values for distance, speed and acceleration and then use them to interpolate or extrapolate the position at any given time.

This utility component also features cryptography related functions. It can encrypt/decrypt sets of data with various algorithms, it can create digests and it can verify digital signatures.

4. Results

We have run several tests for performance evaluation. Due to the limited space in this paper and since some of the results are confidential, we only discuss two of them.

Result 1 - Client-Server transfer rate measurement: Many factors may influence transfer speed between two devices, among these:

a) the type of communication (reliable or unreliable),
b) the distance between devices,
c) the interference from other wireless devices,
d) other tasks executed on the processor of the device at the same time,
e) the size of the message that is sent,
f) the size of the buffer,
g) the number of client devices connected to a server, and so on.

In figure 2 we measured transfer rate from a client to the server for multiple message sizes and with different numbers of connected clients. Sending fewer larger data packets increases the performance over sending more smaller packets.

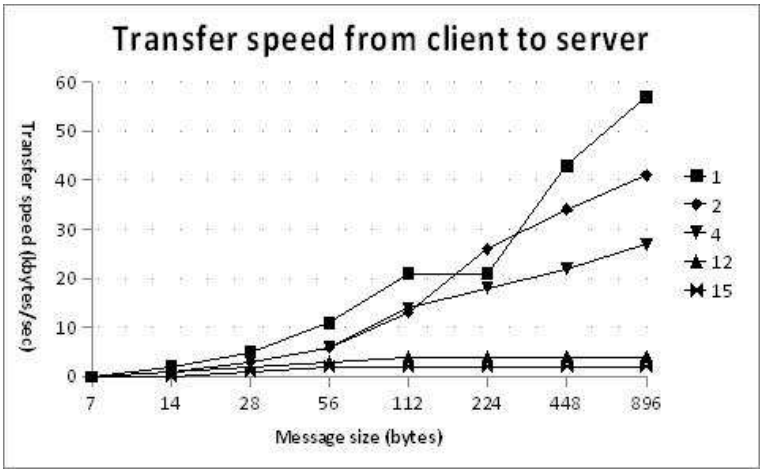

Fig.2: Transfer speed from a client to the server

For example, sending 896 bytes at once instead of 7 bytes 128 times can increase performance up to almost 60 times. The second conclusion is that the communication speed varies with the number of clients. This is because the buffer on the server is divided between the clients, so less data is served to them.

Result 2 - Interpolation: For the interpolation component there are two parameters to test: accuracy and speed. The main parameters that affect performance are the algorithm used for interpolation and extrapolation, the data type used for storing points (integer or floating point), the data structure used for points (vectors, lists, ...) and even compiler optimizations. But, as already stated, the processor on the Nintendo DS does not support floating point operations, so they are emulated in software. This induces a performance penalty. In figure 3, the difference between floating points and integers is clearly visible.

These measurements show the number of microseconds that each algorithm needs to execute the two important actions: adding a new data point and interpolating/extrapolating between two points. The two versions, debug and release, are presented here to show the amount of optimization done by the compiler. We can see that the amount of compiler optimizations is greater for those algorithms that do more mathematical computations: like physics, cubic, quadratic or tangent. The data structure used to stored points is not included in the test since it does not influence the results. Another conclusion is that some algorithms do more work when adding data while others spend more time in the interpolation method. We can see that the quadratic spline, cubic spline, tangent and physics algorithms do significant computations when adding a new point. That's where the equations of the spline or movement are calculated. We can also see that all the algorithms take approximately the same amount of time to interpolate because the interpolation procedure is a simple matter of applying a formula for a given point.

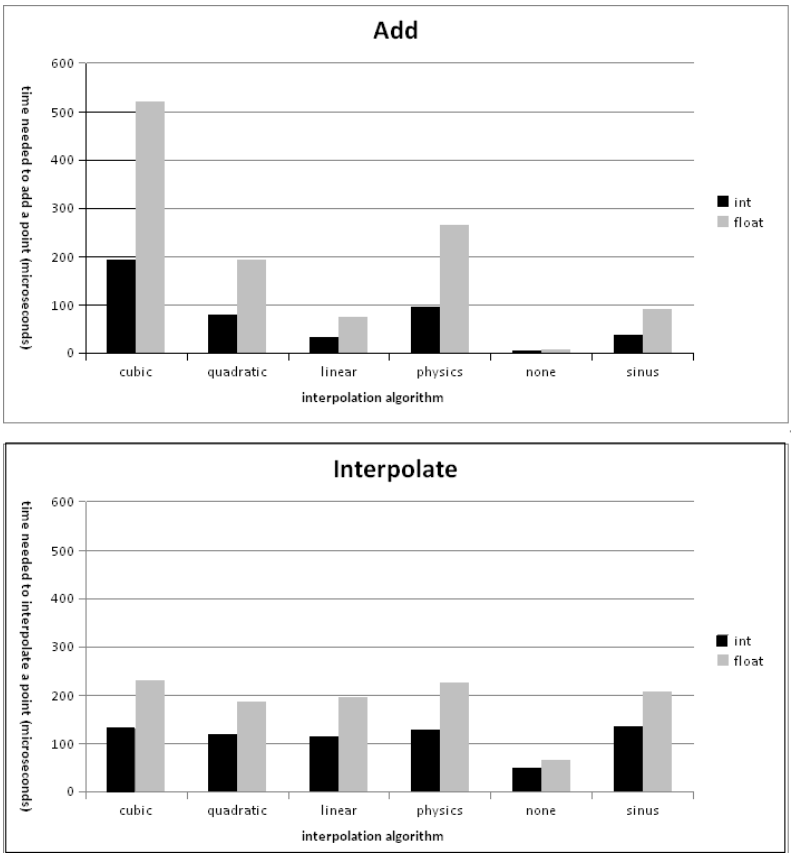

Fig.3: Interpolation results.

Many additional conclusions can be drawn: *(i)* Linear interpolation algorithm is the fastest and its results are good enough for many applications, (*ii)* Cubic interpolation produces the best results but it is slow and it may produce numeric errors, *(iii)* Quadratic interpolation produces severe oscillations and it is provided only for completeness, *(iv)* Other algorithms, like the physics ones, are very effective in particular situations.

5. Concluding Remarks

We have designed a network engine for the Nintendo DS gaming console that makes developing multiplayer games very easy. It offers to the programmers a simple API and allows them to write the networking part of the application very fast which, in turn, significantly reduces the development time of the game.

Regarding future work, the engine may be expanded to work with other game consoles or even with mobile devices such as smartphones. And finally, the ultimate goal is to develop a commercial Nintendo DS game using the developed engine.

Acknowledgements

M. Pâslariu is indebted to the members of the studio Le Caillou Information Multi-Services (Paris, France) for the very welcoming working environment. He is also indebted to the "Lorraine's Universities' Friends" Association (AUL) for its awards of this work.

References

[1] Goldsmith Andrea. Wireless Communications. Chapter: Overview of Wireless Communications. Stanford University, California. Hardback, August 2005. ISBN-13: 9780521837163 — ISBN-10: 0521837162.

[2] Kaw Autar K. Spline Interpolation. Holistic Numerical Methods Institute, College of Engineering, University of South Florida, Tampa. June 2004. http://numericalmethods.eng.usf.edu/ebooks /spline 05inp ebook.htm.

[3] Lusch Adam C., Fleury Adele V. and Chandra Surendar. Do Nintendo handles play nice? An analysis of its wireless behavior. September 2007. NetGames'07, Melbourne, Australia.

[4] McTernan Michael. ARM Stack Unwinding. http://mcternan.me.uk/ArmStackUnwinding/

[5] Pâslariu Mihai, Wireless Library for the Nintendo DS. Master thesis, Université de Lorraine (former Université Henri Poincaré Nancy 1), Computer Science Department. June 2008. Nancy, France.

[6] Nintendo. Nintendo DS, Nintendo DS Lite, Nintendo DSi-Wireless Router Information, http://www.nintendo.com/consumer/wfc/en_na/routers/

[7] O'Neill Charles. Cubic Spline Interpolation. May 2002. MAE 5093

[8] Tobias Fritsch, Hartmut Ritter, Lochen Schiller. The Effect of Latency and Network Limitations on MMORPGs (A Field Study of Everquest2). October 2005. NetGames'05, Hawthorne, New York, USA.

[9] Toh Chai K., Ad Hoc Mobile Wireless Networks: Protocols and Systems. Prentice Hall, December 2001. ISBN-10: 0-13-007817-4 — ISBN-13: 978-0-13-007817-9.

[10] Toronto Neil "Haste". Lag Compensation Technique and Code that does it. http://www.ra.is/unlagged/.