Building an Open Source Game Studio: opportunities for small teams and developing countries

Cesar Mauricio Pachón Meneses

***Abstract*—The high quality and maturity of Open Source and free tools bring new opportunities to small teams to reduce startup costs without losing quality. This opportunity can be extended to developing countries as an accelerator for the consolidation of a video games' industry. This paper describes how to develop video games using only free and Open Source tools, how to choose and integrate them, in order to obtain a real Open Source Game Studio.**

***Index Terms*—Blender, Gimp, Open Source, video game development, video games studio.**

I. Introduction

Traditionally, video games studios had been characterized by having a large team of artists and programmers, and huge budgets with the capability to support all these people during development timelines between eighteen months to two years. If we add the hardware, software's licenses, office rent and others, and it will become clear that serious development projects were not a reachable goal for small teams. Nevertheless, new opportunities for small teams have emerged with the appeareance of new platforms, devices, markets and business models. As a direct effect of globalization, the opportunities have been there for all the countries equally, but, as experience shows, latin american countries seem to be out of the game.

There are many reasons, ranging from cultural and historical to educational issues, and their analysis is beyond the scope of this paper. But when a small team is put toghether and has a project in mind, and all the restrictions are overcome, in the end they will face the ultimate problem: very short budgets.

And here is where Open Source and free technologies can help. The kind of help is, in fact, larger than licensing costs reductions, as developed countries had discovered [1]. Section II will explain the ways in which a small team can take advantage of Open Source technologies, and section III will show the concept of production workflow and how it can be implemented with these tools..

Finally, the last section will show a case study on how Open Source and free technologies can bring a commercial project to life.

Manuscript received October 9, 2007.

C. M. Pachón was with the National University of Colombia as reseacher student, the UMNG as teacher and auxiliar reseacher and now as independent consultant and developer of videogames, computer graphics and virtual worlds. cesarpachon@gmail.com, http://www.geocities.com/cesarpachon

II. Advantages of open source for small teams

A. Open Standards

Open Source tools are closely related to open standards. Traditionally, closed software houses had designed their own specifications for everything: from file definition to network protocols. It means that it was almost impossible to change from one provider to another (wich was the end goal of these companies). But now, open standards have changed the landscape, and interchangeability between tools is a requirement that all modern tools must provide. In the core of the groups and consortiums which define the standards are always opensource implementations, also known as RI or Reference Implementations, that in many cases have very few differences with their commercial counterparts. Furthermore: in many cases the open source implementation uses the standards as core functionality, while commercial tools will offer the standard feature as secondary. An example is the SVG standard (Scalable Vector Graphics) designed for vectorial drawing. The Inkscape tool uses SVG as its principal format, while other commercial drawing tools would offer SVG integration through more or less functional export and import plugins, and sometimes as something optional. Summarizing, the use of open standards reduces the risk asociated with being tied up to a single software provider. It gives flexibility to the team, the risk and flexibility can be estimated in money by any project planning methodology.

B. Licensing costs

The most obvious of the advantages, reducing license costs would make the difference between allowing a project to go on or die prematurely. This is specially true in some very competitive markets, with low earning margins, like the games for mobile phones one. Even in the case where the team has the money, it can always be used in more critical costs such as developers' salaries. Lower costs can always be passed to the final client, and becoming itself more price-competitive in a global market is a clear advantage.

C. Educational Issues

Another risk that any development project must face is to

know each member of the team's knowledge of the tools. Poor knowledge is a recipe for disaster. Learning as you go, maybe too. Usually the final tools for the project are not well defined in early stages. In fact, in parallel to the design stage, technical team must make recommendations on the tools and technologies more appropriated to turn the design into a real product. These recommendations will be biased by personal expertise and constrained by budget's size. How can open source technologies reduce this risk? First, it is more likely for team members to have real experience with open source tools, either as part of personal exploration or for academical projects. Secondly, it is possible that some member of the team be an active participant in some open source initiative, which newly will be a big advantage. Finally, support from the community in forums, wikis, tutorials and mail lists will be even better that the support from commercial providers, as showed in the bazaar and cathedral model discussions[5].

D.Reducing technological risk

Technological landscapes changes quickly, but the market do it faster. For some kind of business models, be able to adapt to changing client requirements is mandatory. This is the case of advergaming projects, which are characterized for low budgets and even shorter timelines. Another feature is an almost complete indifference from the client regarding the technical features and tools used in the project. In advergaming projects the final product is different from the initial proposals, because the project will be adjusted all the time by the client, who will ask for new features. Some of this features will require the team to include new libraries and technologies not planned. For example, a game was planned with a very simple collision model, and suddenly, the client asks now that instead of walking, the character drive a vehicle that can crash with another objects. The team planned the game with very basic collision routines, and now it seems that they will need a physics engine. If they licensed a third party graphic engine for the game, maybe they will need to license the physic engine separately.. and that will hit their costs! In the open source side, they will have many options, like the inclusion of ODE or NEWTON libraries, or moving the project to other engine which offers integrated physic routines.

III.PRODUCTION WORKFLOW

A.Asset pipeline

Unlike traditional software development, videogames development is an extremely multidisciplinary task. From the design document to the testing and delivery of the product, many interdependences between the sub teams will emerge. Ironically, these conflicts will arise even in the case of one person team, because some change in coding would require slightly modifications to art assets. Equally, some art assets would be constructed in a way that is not appropriated for the game.

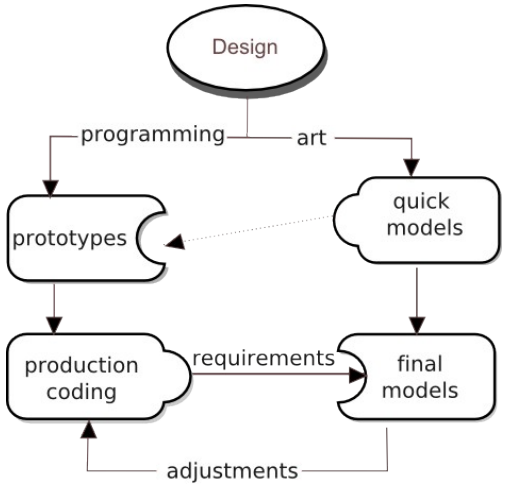

Fig 1: Programming and art task run in parallel, but are mutually dependent.

The formats used for loading art assets into the game are a common source of problems. The artist will use a modeling tool with its proprietary format, which will then run some export plugin to transform the model into a suitable format to be loaded into the game. Once the programmer do this, the artist will notice that the model in which he invested many hours looks really different in the game that in the modeling tool.

Fig. 1 Shows this situation. Programming and artistic activities are developed in parallel, but they are closely inter-related.

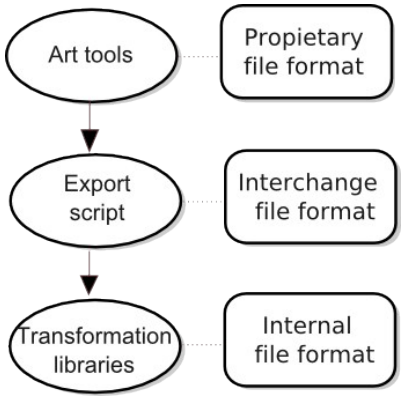

Fig. 2: asset pipeline, assets would change their format through the process.

The team must careful and make sure that all the tools used in the production workflow will work together in a seamless way. It is a priority to check that both the art tools and the developer libraries will use the same open standards. It is also good to have more than one option in both the art tool and the developer libraries, because in some cases not all the formats will be appropriate for each art asset. For example, OBJ is good for static geometry, but it can not handle skeleton deformation [2].

A careful selection of tools and formats must be done in

order to make the “asset pipeline” as smooth as possible, as Fig. 2 shows.

B.In house file formats

Sometimes the development team defines its own file formats. these formats are known as "in house file formats", and there are many reasons to do that:

1. To protect artwork: putting the final artwork of the game in public formats for final distribution makes them more vulnerable to be copied and modified. However, the practice of using obscure and undocumented in-house formats to protect the art assets is not recommended, since it is preferable to use digital signature and other well-tested encryption technologies that will save the time required to develop and maintain these formats.
2. To reduce size: Another reason for in-house formats are reduction in the size of the final distribution. This is a bad reason too, because it is always preferable to use other compression technologies.
3. To optimize loading times: This indeed is a valid reason. There is a big difference in the loading times between resources saved in text based file formats and binary file formats. In some cases, it is possible to swap complete binary structures from memory to disk, thus getting the maximum speed in load times.
4. Supporting additional features: There is a software design principle that encourages the separation of data and logic. This principle says that the source code must keep only logic driven by data stored in external repositories, not in the source code[4]. A simple example of this are the implementations of FSM's (Finite State Machines) used widely in videogames' projects. In a basic implementation, a FSM can be written as a long sequence of “if.. else.. if ..” sentences. A implementation more compliant with the data-driven principle would write generic classes to represent states and transitions, and would load the particular conditions from a file. Another example is the terrain information required for some AI algorithms. Precalculated graphs and indicators associated to the terrain would bring valuable information to the agents. In large worlds, visual editor tools are mandatory, and a way to build them is to extend the modeling tools with plugins to use them as specialized design tools.

IV.CASE STUDY

This section focuses in one case study, an advergaming videogame developed for an internet campaign. This is a real project with a one person team, and shows the risks faced by a small team and the help provided for open source and free tools in order to avoid them.

A.Podium project: requirements

“Podium” was the cornerstone of a advergaming campaign launched in 2005 and sponsored by a Colombian broadcasting agency. At the beginning, the client wanted a car racing game that could be played through internet, in a massive way (not multiplayer). It was important to have an authentication and timing server, because the client wanted to give a prize to the faster drivers each week.

In order to reach the largerest amount of people, they established some technical restrictions:

1. It had to run in older machines, even with on-board graphics chips, without 3D acceleration hardware.
2. The installer had to always have a size under 10 MB, even with all the audio, video and art assets.
3. Users should not need any other configuration besides former installation. They did not have to download and install any dependencies, like DirectX runtimes.
4. Each week the sponsor would free new levels for the game. These levels could be downloaded and installed easily.
5. The game had to allow changes in the UI areas each week, where the client wanted to expose third-party advertising.

B. Technological choices

With these requirements in mind, the next step was to choose the tools that were to be used in the project. Requirement 1 ,2, and 3 caused the rejection of some robust game engines, specially Ogre3D, because the engine's core itself had a size higher than the 10 MB limit. Others engines were rejected because they have dependencies with third party libraries, like DirectX runtimes or Java virtual machine, like Jmonkey.

It was decided to use a in-house 3D engine developed for previous projects. The engine, called OBPlus, was really simple, based in OpenGL and SDL, all of them free technologies without licensing restrictions for commercial usage.

OBPlus had very basic features, and some limitations that were not critical for the project. One of them, was the lack of support for Bone Mesh deformation, but that was not an issue because it was a cars game with rigid models.

OBPlus had also an interesting feature: it had many in-house file formats, both in binary and text forms, and export plugins for Blender, a 3D Open Source modeling and animation tool.

Then, the asset pipeline was defined as:

1. Modeling and animating in Blender.
2. Exporting to OBPlus text format using Python scripts.
3. Transforming from OBPlus text format to binary format, using the tools provided with the engine.

This was done both in geometry assets and in texture assets, satisfying a new requirement from the client: users would not be able to modify, edit or replace advertising assets from the game.

Other uses of propietary formats where the definition of levels, thanks to the integration with Blender through a Python script that allowed the use of Blender as a scene editor.

C.Project specific file formats

Some project specific file formats were developed. One of them was a path file format, used to describe the road topology for the AI agents. Thus, while the user controlled his own vehicle, artificial agents were driving others cars, using the path info for navigation. Here, Blender was used again as a "Path editor", thanks to a script written in Phyton.

D.Server side

In the server side, a web server was implemented using Java (J2EE) with a basic servlet implementation. The application server was JBOSS, a very robust and flexible Open Source alternative. And finally, a MySQL database was used for persistence of user information.

E. Free and OpenSource tools used

This is a relation of all the tools and libraries used in the project:
C++: programming language of the OBPlus engine.
OpenGL: 3D graphic library.
SDL: basic window and IO support for the client.
ODE: physic library, written in C.
Bloodsheed: free C++ editor environment, compiling under Cygwing.
SetupMaker: free tool for making installers.
Gimp: Image Manipulator Program. Used for edition of textures.
Blender: 3D modeling and animation package.
Python: scripting language used for write plugins for Blender.
Java: programming language used in the server side.
Eclipse: Open Source IDE for java development.
JBoss: Open Source web server used for the authentication and timing server.
MySQL: Free Data Base Manager used for account management.

V.Conclusion

Open Source and Free tools and technologies can be of great help for small teams who needs professional tools but do not have adequate budgets. Even when having the money, the use of free technologies can help to reduce production costs in order to offer more competitive prices.

References

[1] "Study on the Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU", Rishab Aiyer Ghosh, MERIT, september 2006, http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf, pp 9

[2] E. Pipho, Focus on 3D Models, Premiere Press, 2003, ISBN: 1-59200-033-9, pp 63-71

[3] N. Lin, Linux 3D Graphics Programming, ISBN 1-55622-723-X, pp 497-552.

[4] F. D. Laramee, A Game Entity Factory, Game Programming Gems II, pp 51,61

[5] E. Raymond, the Cathedral and the bazaar, O'Reilly Media, 1999, ISBN 1-56592-724-9