A Novel 3D Game API for Symbian OS Smartphones

Fadi Chehimi
Infolab21
Lancaster University, Lancaster
LA1 4WA, UK
+44 (0)7731395946
f.chehimi@lancaster.ac.uk

Paul Coulton
Infolab21
Lancaster University, Lancaster
LA1 4WA, UK
+44 (0)1524 510393
p.coulton@lancaster.ac.uk

Reuben Edwards
Infolab21
Lancaster University, Lancaster
LA1 4WA, UK
+44 (0)1524 510392
r.edwards@lancaster.ac.uk

ABSTRACT

Mobile phones are becoming one of the major personal entertainment devices amongst the general public. We are currently seeing a paradigm shift from the traditional voice-centric applications to those that incorporate music, pictures, video, games, internet browsing, etc. Although mobile games represent a significant portion of the mobile entertainment market, they still have some way to go before they reach the revenues generated form the downloads of ringtones, music, and wallpapers. This is due to a number of factors such as the wide ranging demographic of mobile phone users and restrictions of the mobile platforms. Of these platform restrictions is the lack of adequate support for 3D games development which is one of the most problematic issues for many game developers. To alleviate this situation and provide a unified framework we introduce in this paper a novel 3D mobile games API that simplifies the development of 3D games on Symbian mobile phones and allows the production of more feature-rich titles.

Categories and Subject Descriptors

J. Computer Applications, J.7 Computers in Other Systems-Consumer.

General Terms

Design, Experimentation.

Keywords

Mobile 3D Games, Mobile Phones, Symbian OS, OpenGL ES, 3D Game Engines, 3D Game API, Game Design, Game Structure.

1. INTRODUCTION

The console and PC games industry is expected to reach a worldwide figure of $47 billion by 2010 [1] with an anticipated customer base of over 126 million gamers [2]. If this number is compared to the 3 billion expected mobile users in the same time frame [3], and the $11 billion to be generated from mobile games, we notice an obvious disparage. The minimal proliferation of mobile games is relied to a number of factors including audience demographic and technical challenges. Whilst a number of challenges can be identified in this paper we are concerned with those technical issues that affect 3D mobile games in particular. And in the following paragraphs we shall explore some that particularly affect games development.

One factor is the limited-resources nature of mobile phones compared to those experienced on PC's and game consoles. Amongst these restrictions the following are the most significant:

- Limited memory;
- Relatively slow processors speed;
- Lack of dedicated graphics processors;
- Small screen sizes and resolutions;
- Limited user input/output interfaces;
- Short-operation battery life.

These limitations have a direct correlation to why many current mobile games being merely 2D ports of old arcade games. This has led to many traditional gamers being disappointed with the perceived poor quality, features, and limited 3D experience available in these games.

A second contributing factor affecting development has been the absence of 3D-graphics-dedicated APIs that are tailored to address these limited devices. Some highly experienced developers and game houses have developed their own 3D APIs [4] but these have not been open or standardized for the general community of mobile game developers. This has recently and partially been resolved by the introduction of the following APIs [3]:

- OpenGL ES; for Symbian OS, Binary Runtime Environment for Wireless BREW, Mobile Linux and Windows Mobile
- Mobile 3D Graphics M3G; for Java 2 Micro Edition J2ME

A final factor to consider is that having more than 20 different mobile platforms and operating systems makes it very difficult for game publishers to port their games and engines/APIs across all platforms; thus, limiting the potential revenue for a game.

In this paper we introduce a novel 3D gaming API as a solution to this portability problem which will provide a common upper layer interface for the Symbian platforms and OpenGL ES 1.0 (Embedded Systems). The Symbian OS has been chosen for this games API as it is the leading mobile platform in the market [5] with more than 70 million devices shipped by March 2006 [6]. OpenGL ES 1.0 is the graphics API selected as it is native to the operating system and supported by most new Symbian phones. Whereas, the latter versions of this API, 1.1 and 2.0, are currently still limited to laboratory devices. Nokia has recently released its multimedia phone the N93 which is the first to support OpenGL ES 1.1 specification [7], and many forthcoming models will undoubtedly utilise this standard.

The API developed, which has been termed Syga-PI3D (*Sy*mbian *Ga*me *API 3D*), is still under development and in this paper we present the first stage which is of a three-stage development cycle. In the next section we shall introduce Syga-PI3D and its features before section 3 discusses its design and external supporting tools. In section 4 we present the API's implementation whilst sections 5 and 6 will identify its features indicating those that are completed and those still to be implemented. In section 7 we draw our overall conclusion and discuss the future evolution of Syga-PI3D.

2. SYGA-PI3D IN A NUTSHELL

Syga-PI3D is an application programming interface (API) that enables mobile game developers to develop 3D games for Symbian smartphones without the need to consider:

- the particular coding conventions of the operating system,
- the different user interface frameworks,
- and the syntax of OpenGL ES.

It acts like an upper-layer interface on top of Symbian OS and OpenGL ES providing API facets and functions that will help in creating feature rich 3D games.

To aid developers in addressing phone limitation issues several offline supporting tools have been implemented for Syga-PI3D to help in constructing 3D game environments. These tools are specifically aimed at minimising the excessive processing on these power-hungry devices. The tools are responsible of tasks like building optimized data files, for game objects and characters, partitioning game world spaces, and enabling selective scene rendering.

Syga-PI3D implements many game-design requirements that would be expected for any 3D game. For instance, the API provides implementations for: collision detection algorithms, texturing and textures controlling, space partitioning procedures, rendering management systems, various 3D effects, models loading and optimization, and networking facilities for multiplayer games. The programmer need only instantiate objects of these systems in his/her game and the API will do their specified tasks with minimal input required from his/her side. For example, when a game environment is set up, a spaces-tree object of the game world, or level, is instantiated automatically. However, the object still needs to be constructed by the programmer as a lineartree, quadtree or octree and parameterized with its dimensions and representing data file(s).

3. THE DESIGN OF THE API

Syga-PI3D is still in its evolution and this paper presents the work completed for the first development stage. Two more stages will follow which will be highlighted in subsequent sections. This first stage includes the implementation of the core structure of the API with its main functional features alongside some external help tools. The other stages will include the artefacts and the performance optimisations in addition to an expansion of the building blocks. In this section we discuss the overall design of the API and explain how the external tools cooperate within its framework.

3.1 Design Overview

As mentioned previously, Syga-PI3D is designed to interface Symbian OS and OpenGL ES hiding their implementation and providing abstract layer for programmers independent from any UI framework, as shown in Figure 1. As we will expand upon in a latter section Syga-PI3D uses the Window Server (WSERV) of Symbian OS directly, without any specific UI-dependencies to build game environments (GameEnv). GameEnv sets the screen size, enables/disables full screen drawing, selects screen colour mode (maximum 64K including alpha channel), sets game pop-up menu options, and handles user input and events. It also instantiates under-the-hood OpenGL ES environment (OGLESEnv) and the game spaces-tree, associates manually added game objects and models, and manages scene rendering with collaboration between the programmer (manually) and GameEnv (Automatically).

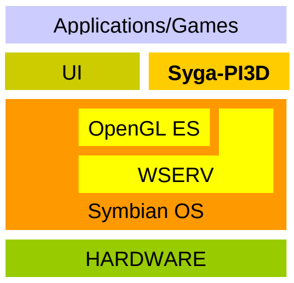

Figure 1. Integration of Syga-PI3D with Symbian OS Platform and OpenGL ES API

OGLESEnv initializes OpenGL states by the programmer calling GameEnv API functions to set up lighting properties, shading modes, material colours, perspective frustum, hidden face removal, clipping, culling and other 3D effects. Default settings are used if no values are specified by the programmer. An alternative way to set up these states is by coding OpenGL commands in one of the pure virtual functions that need to be implemented in any Syga-PI3D game, as we will elaborate further.

OGLESEnv always enables a camera which has a default position at origin (0,0,0) and orientation towards the negative z-axis (0,0,-100). Note that the right-hand coordinate system is used where the positive x-axis is the thumb, positive y-axis being the index finger and positive z-axis being the middle finger outwards the page. The camera API has been added to Syga-PI3D since cameras are required features in any 3D game. It is implemented to facilitate different viewing options for various game genres. It enables panning view in single plane, the third-person view, in Doom-like games, and a character-following view whereby the camera follows the main character in game, such as driving games.

GameEnv instantiates a spaces-tree structure to manage polygon distribution in a game for use by rendering and collision detection systems. The game space dimension is specified by the programmer and the partitioning process will be performed by the API automatically. There are three different types of space partitioning supported:

- lineartree where the space is partitioned into linearly linked spaces as an array of spaces,
- quadtree where the space is divided into four subspaces recursively until a specific depth is met,
- and octree where the space is split in the same manner as for quadtree but into eight subspaces instead.

Only lineartree and quadtree have been implemented in stage I. Octree structure will be made available in stage II. Note that programmers using linear partitioning have to construct each space node individually and add it to the linear tree. A node could be located anywhere in the game world without neighbouring specifications, as for quad and octree. The API will only construct the tree's root which will connect all floating nodes.

The data required to partition a space and distribute its polygons, for collision detection and scene rendering, is maintained from the `.sgd` (Symbian Game Data) data file generated by the Space Partitioning Collision Detection (SPCD) tool. This file is parsed in the construction phase of the spaces-tree. The file has to be passed to GameEnv in order to associate it with the global tree.

Another design feature implemented by Syga-PI3D, and managed by GameEnv, is the rendering management system. It includes occlusion and scene division and requires interaction with space partitioning. Feasible rendering is a crucial and intricate process that requires intensive research into possible solutions that are optimal for mobile phones. It has not been implemented in stage I and thorough research is being conducted to select an appropriate model.

Last but not least, any game must have characters and objects to create premise for the game play. Syga-PI3D's design provides support for importing *objects* and *models*. "Objects" refer to static objects in a game like elements of a level such as bridges, statues, weapons lying on the ground and power-ups scattered throughout the level. Whilst "models" refer to animated game characters such as the main game characters, competitive characters for other players or even non-player character(s).

Similar to space-partitioning, objects and models are represented with data files and textures generated by external API tools. Objects use the `.sod` (Symbian Object Data) data file format and models use `.smd` (Symbian Model Data). The developer needs only to instantiate an object, or a model, and pass its particular data file to its constructor, then associate it to GameEnv. This will allow the API to manage the object/model throughout the game. The loading will start behind the scene without the programmer needing to initiate parsing and data manipulation. All loading processes are performed asynchronously and optimised against failure.

Figure 2 summarizes this general description of the Syga-PI3D API and identifies how its components integrate with each other for any Syga-PI3D-based 3D mobile game.

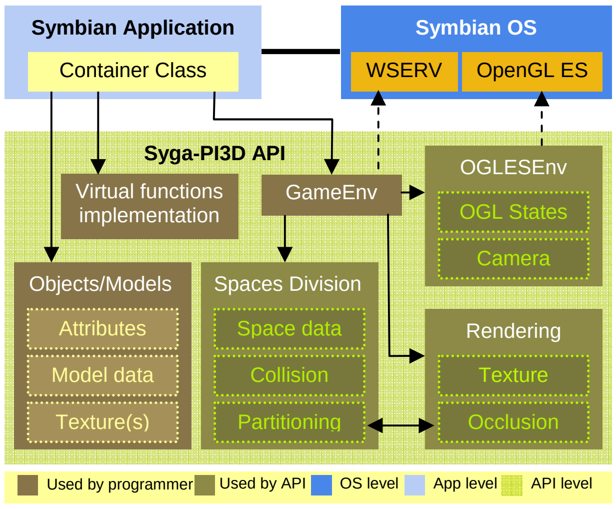

Figure 2. Syga-PI3D API Underlying Design.

3.2 External Tools

Syga-PI3D facilitates the management of game assets with the support of four offline help tools. The first tool, called Symbian Object Data (SOD), creates static objects' data files in the `.sod` format. This format is extracted from Wavefront OBJ format with optimized contents such as fitting an object's data arrays into byte or short types in order to minimize the memory required. It also adds header information such as the sizes of these arrays to eradicate the need to calculate them at runtime. Once created, a `.sod` file need only be associated with its appropriate object instance in the game in that object's constructor.

The second tool termed SPCD (Space Partitioning-Collision Detection) deals with a more critical game structure feature: space partitioning. It divides the game world into spaces in a tree structure with four nodes (quadtree) or eight nodes (octree) in a `.sgd` file. Should this division be performed at runtime it would be computationally expensive requiring many levels of recursion, which would put a heavy burden on mobile phones' dynamic memory and processor cycles. Thus, off-loading this requirement from the device is a good practice since it lessens the power drain of the battery.

Each space node in the tree, linear, quad or octree, contains the objects, or models, that belong to it and a set of polygons of the world design that intersect with it. Adding polygons is performed by SPCD which utilises the game world model data file(s) generated by SOD. The SPCD then extracts the world's polygons and spreads them in the tree spaces (nodes) they optimally belong to. However, adding objects and models is done in their construction phase. The GameEnv instance in the game is passed as a parameter to the constructor of the object or model which then traverses the environment's spaces-tree to find where it should add itself. This division of the game world into spaces and distributing polygons and objects helps with collision detection. The player, or the main character in the game, will need to test collision only with polygons and objects/models that exist in its same game subspace.

The third and forth tools have yet to be implemented but their functionalities have been identified. The Symbian Model Data (SMD) tool will generate the `.smd` data files for in-game animated models. The particular format has not been finalized but most likely it will be an optimized version of Quake III MD3 format. The last tool, the OCC, will be used to manage occlusion, occluding objects, scene rendering and scene texturing in games. Both tools will be introduced in stage II of Syga-PI3D.

The external tools are all console-based at the moment and will be associated with graphical user interface (GUI) in stage II.

4. SYGA-PI3D IMPLEMENTATION

Syga-PI3D has been developed with object-orientation in mind. In order to write efficient and reliable games on Symbian platform using Syga-PI3D or any other framework, certain operating system dependent considerations have to be retained. In this section we cover the implementation structure of Syga-PI3D and introduce critical Symbian-dependent issues and how these are addressed in the API.

4.1 Implementation Structure

To utilise all features mentioned in Section 3.1, programmers have to instantiate objects of each API feature class. Figure 3 shows the structure and relations of all API classes that have been implemented in stage I. The figure identifies the classes in generic terms without reference to their data types. However, each class, except `TVertex3D`, has floating-point and fixed-point implementations.

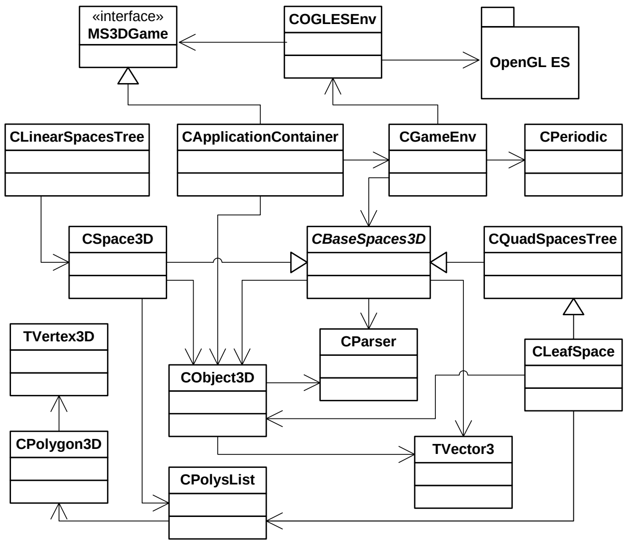

Figure 3. UML Model of Syga-PI3D Classes

The game entry point starts in `CApplicationContainer` class, for a particular application called "Application" thus the name `CApplicationContainer`, where `CGameEnv` should be instantiated by the programmer. With the aid of functions in `CGameEnv` he can set game timer, represented in `CPeriodic` which is a timer API of Symbian OS, set OpenGL states via `COGLESEnv` which communicates with OpenGL ES API in the operating system level (Figure 1), and selects which tree structure to use, either `CQuadSpacesTree` or `CLinearSpacesTree`. The appropriate game space data file created by SPCD will be used here and `CParser` will generate its data structure. As mentioned earlier, if `CLinearSpacesTree` is chosen the programmer has to instantiate as many `CSpace3D` objects as is required in the game.

At this point the game world is ready for action but the particular game objects and models have to be loaded to initiate this action. Each space has a reference to all objects/models that belong to it in a linked list structure. The list is obtained by the spaces base class `CBaseSpaces3D` which includes other space-specific information like dimension and position each represented with `TVector3D` objects. (The math in `TVector3D` has been optimized to meet the mobile platform requirements). From this list objects in any space can track which other objects to detect collision with efficiently. Loading animated models is not supported in this stage but should behave similar to loading static objects when addressed in stage II.

In `CApplicationContainer`, the programmer has to instantiate `CObject3D` for each object in the game. The `.sod` data files are used here for `CParser` to construct the objects. It has to be noted that objects must adhere to certain complexity limitations to retain performance of games on mobile phones. The maximum number of faces allowed for any object in a game is 2000, but 200 is recommended to maximise speed and memory benefits. Due to the small physical screen sizes on mobile phones this level of details should be sufficient without sacrificing any visual quality.

A distinctive feature in `CObject3D` API is the ability for its instances to detect collision automatically within games. There exist collision detection routines in this class that are managed by Syga-PI3D to prevent objects from colliding with each others in their space or for characters to walk through walls (polygons) in that space. Each space, linear, quad or octree, has a linked list structure (`CPolysList`) holding all the polygons (`CPolygon3D`) that belong to it. A `CPolygon3D` object is composed of three vertices of `TVertex3D` type. The reason for having a different class for vertices and not using `TVector3D` is relied to the size of a `TVector3D` object. `TVector3D` contains three `float` or fixed (32bit: s15.16) members giving a size of 12bytes for each vertex. This size is not convenient for the low-memory mobile phones since it will result a large size of geometry data arrays. Therefore, `TVertex3D` was introduced with `short` type members with a size of 6byte per vertex reducing by this the total size into the half. Add tot his that `short` types will always be used as SOD generates files with arrays of bytes or short integers.

As shown in Figure 3, `MS3DGame` is an interface that must be inherited by any Syga-PI3D game's container class. All events handling, game initializations, and game loop are manipulated via pure virtual functions of this interface and have to be implemented. The `GameInit()` function will initialise OpenGL states with default values. However the programmer can amend these values via the helping functions provided by `CGameEnv` or by hard-coding OpenGL ES commands in `GameInit()`. The later method is a flexible design feature of Syga-PI3D which enables experienced developers to set the states that are not addressed by the API, and which opens the door for future compatibility with new versions of OpenGL ES.

`GameLoop()` is another pure virtual function which controls the execution of game components and its `CPeriodic` timer. Here the programmer will set the number of frames sought after in the game and the API will synchronize its operations to meet that frame rate. If meeting the target is unsuccessful, the default of 25 frames/sec will be used.

The last virtual function has not been developed as yet but must be implemented in games for future use. `GameEventHandler()` handles user events and sends them to the API for processing. This functionality is already implemented elsewhere in the operating system level and the programmer must use it instead. Abstracting and directing this function to Syga-PI3D has to be resolved and it will require implementing a customised and optimized windowing system that is expected to improve game performance.

4.2 Symbian-Dependent Considerations

Just like any modern multitasking operating system, Symbian OS's architecture uses many advanced, but classical, constructs including pre-emptive multitasking threads, processes, asynchronous services, and internal servers for serializing access to shared resources [8]. However, Symbian OS has some particular features that have to be considered in order to write effective and robust mobile applications. In this section we highlight some of the major issues and explain how Syga-PI3D implements them

4.2.1 Error Handling

At the time of development of the Symbian OS, C++ conventional exception handling was not part of C++ standard. When it was introduced, it was found to add substantial overhead to the size of compiled code and to runtime RAM. Thus, Symbian has implemented a different handling approach represented in Traps and Leaves [8]. This error handling mechanism has been considered in the design of Syga-PI3D and implemented in all its classes. This means that programmers will be safe using the API functions without concern for code crashing. However, if programmer-defined, non-API functions cause an error, or "leave" in Symbian terms, programmers will have to handle these themselves.

4.2.2 Memory Management

Memory leakage is a serious problem on mobiles phones with the limited memory available as it is not possible to simply reboot the mobile phone to mop up any memory leaks [8]. The Symbian OS is practically designed to manage its memory resources if a failure or leave occurs [8]. This is done by implementing a Clean-up Stack which keeps a reference to heap memory used in order to delete it in the case of leaves and prevent it being orphaned. As for error handling, memory management is systematically implemented in Syga-PI3D to provide the appropriate and secure ground for building high quality mobile games.

4.2.3 Event Handling and Windowing System

One of the most critical learning curves in developing applications for mobile phones is the effectiveness and accuracy in handling a large number of events. The presence of radio, shared recourses, asynchronous operations, operating system management and third-party applications all create a rich pool of events to handle. This in not limited to Symbian OS only but it is a common denominator for all mobile platforms.

In Symbian OS, event handling is part of the windowing system, which Syga-PI3D uses natively. This means the operating system already implements functions to handle user events. Such functions have not been abstracted by our API meaning that programmers have to use them and know where to place event-prone code. However, they would appear in stage II.

4.2.4 Synchronous vs. Asynchronous Operations

Symbian OS allows the usual context switching between threads and its distinctive cooperative multitasking technology: Active Objects. Explaining what and how Active Objects work is beyond the scope of this paper but in a nutshell, Active Objects provide a multi-threading-like behaviour in a process but within a single thread in that process instead of many. Active Objects will share the time-slice of the processor assigned to their thread, or process, by cooperating between themselves in performing a single or set of tasks. They cooperate asynchronously in an event-driven manner where the operating system searches through the outstanding Active Objects until it finds a completed one and runs its handler function, while the others keep running in the background. This reduces the overhead incurred by threads context switching on the kernel scheduler, memory management unit (MMU) and hardware cache. This is required as the state of the running thread and its memory in use at the time of pre-emption has to be saved in order to start from that point once the thread's execution is resumed. Last but not least, the context switching mechanism drains a lot of power and this should be avoided in battery-powered devices like mobile phones [8].

Active objects are used in all loading and parsing functions in the API. For instance, `LoadOBJ()` function in `CObject3Df` class is implemented using asynchronous Active Objects where the game would not block waiting object data to be loaded from `.sod` data files.

5. FEATURES IMPLEMENTED

As stressed throughout the paper, Syga-PI3D is in its first stage of development. The features of the API are scattered in its three phases allowing the opportunity of testing and bullet-proofing each stage components individually in order to build a solid and robust game API. The features that have been completed in this stage are as follows:

I. Space partitioning, where a game's world is subdivided into smaller spaces recursively to enable more advanced features.

II. Collision detection, which is empowered by space partitioning.

III. Environment set-up, including Symbian-related and OpenGL ES properties.

IV. API core implementation, with integration between it and Symbian OS.

V. Basic 3D effects, like shading, lighting, skybox, camera, transparency, blending, fog, etc.

VI. External support tools; which will offload memory and processing overhead from mobile phones.

VII. Materials and texture mapping; limited to 2D texture of JPEG format only. More formats may be supported in the latter stages.

VIII. Loading objects; OBJ-format used at the current time.

IX. Integrated abstract interfaces, allowing easier use of Symbian's and OpenGL ES' functionalities.

6. FUTURE FEATURES

The next two development phases of Syga-PI3D will focus on the visual artifacts in a game and performance optimizations of the API. Stage II will implement the following eye-candy effects:

I. Mip-map texturing, but without the ability to control textures' level of detail (LOD) as it is excluded from the specification of OpenGL ES 1.0 [9].

II. Animated models loading, which adds realism and interactivity to games.

III. Lens-flare, blur and reflection effects; the last two will implement stencil buffers.

IV. Rendering APIs, for scene management including occlusion systems, occluders detection, height mapping and scene partitioning.

V. SMD and OCC tools.

VI. Octree structure, the last space partitioning criteria.

VII. GUI for the external tools.

VIII. 2D and text display, for sprites and game counters.

IX. Playing audio.

X. Networking facilities for multiplayer games, including APIs for Bluetooth, General Packet Radio Service (GPRS), and probably Session Initiation Protocol (SIP) when it becomes available on Symbian phones.

Stage III will concentrate on performance issues, testing and optimizations. We have not included in this paper any testing or performance figures due to the absence of facilitating API. These and the following points are to be covered in stage III:

I. Testing APIs; will be added to help programmers debug their games and monitor the used/abused resources. These APIs will have on-phone and off-phone testing modules to enable testing games' real-time performance and assets production/integration.

II. Benchmarking APIs for performance monitoring.

III. Math optimizations, by implementing look-up tables for sin, cos and tan math operations used in games. Their software-only implementations by the operating system are slow. Look-up tables may enhance performance when used instead. Also, the square root operation for fixed point will be implemented to speed up calculations.

IV. LOD for models and objects; this is very crucial enhancement since it will reduce the number of polygons to render on each frame resulting on less possible flickering displays.

V. Customized windowing system; rather than using robust Symbian's windowing framework which contains plenty of useful and effective operations but unneeded in Syga-PI3D, we will implement our customized version with support to the only used operations.

VI. Texture caching, to limit the load of data transfer through the limited bus bandwidth.

It is worth noting here that "Particle Systems" which provide effects for fire, rain and water have been excluded from the current design of Syga-PI3D because of their relative limitations and complexities, although they are vital graphics systems to any modern 3D game. Supporting such features on mobile phones would drain battery, memory and processing resources quickly since they will be dealt with only in the software level. Some graphics processors like the new PowerVR MBX from Imagination Technologies provide hardware support for these effects [10] but they are not yet widely available on commercial phones. An implementation for a particle system will appear in a future version of Syga-PI3D once it proves feasibility and optimality for mobile phones.

7. CONCLUSION

Revenues in the mobile sector are very high in mature markets like in Japan and South Korea where dedicated platforms and sophisticated solutions are widely available. Both mobile service aggregators and mobile manufacturers work together to create the right environment and tools for mobile entertainment services to explode. We have studied the lessons and learnt from these leading markets and thus produced Syga-PI3D to be the adequate solution for developing feature mobile 3D games in the European market. It will enable developing 3D games more quickly and efficiently, and will facilitate embedding commercial messages and content in games for marketing purposes [11]; thus, opening new doors in front of not only game publishers but also innovative marketers to use the most innovative medium: Mobile Phones.

8. REFERENCES

[1] Hatfield D., "Game Industry Set to Explode", IGN, News, June 21, 2006, http://uk.pc.ign.com/articles/713/713784p1.html, accessed September 2, 2006

[2] Wegert T., "Gaming 101", ClickZ News, September 22, 2005, http://www.clickz.com/showPage.html?page=3550216 accessed September 1, 2006

[3] Chehimi F., Coulton P., and Edwards R., Advances in 3D graphics for mobile phones, Proceeding of *the 2nd IEEE International Conference on Information & Communication Technologies from Theory to Application*, Damascus, Syria, April 2006

[4] Chehimi F., Coulton P., and Edwards R., Evolution of 3-D games on mobile phones, Proceeding of *the IEEE Fourth International Conference on Mobile Business*, Sydney, Australia, 11-13 July 2005

[5] Evers J., “Symbian Threats Multiply”, ZDNet UK, January 23, 2006, http://news.zdnet.co.uk/internet/security/0,39020375,39248514,00.htm, accessed September 3, 2006

[6] Symbian Ltd, http://www.symbian.com, accessed August 20, 2006

[7] Nokia, http://www.nokia.com, accessed August 23, 2006

[8] Stichbury J., *Symbian OS Explained*, John Wiley & Suns Ltd, West Sussex, 2005, pp: XII, 13, 29, 111-126

[9] Astle D. and Durnil D., *OpenGL ES Game Development*, Thomson Course Technology, Boston MA, 2004, pp: 45-49

[10] PowerVR MBX Demos, Imagination technologies, http://www.imgtec.com, accessed September 1, 2006

[11] Chehimi F., Coulton P., and Edwards R., Mobile advertising: practices, technologies and future potential, Proceeding of *the IEEE Fifth International Conference on Mobile Business*, Copenhagen, Denmark, June 2006