Rendering of X3D Content on Mobile Devices with OpenGL ES

Daniele Nadalutti*
HCI Lab
Dept. of Math and Computer Science
University of Udine
via delle Scienze, 206
33100 Udine, Italy

Luca Chittaro†
HCI Lab
Dept. of Math and Computer Science
University of Udine
via delle Scienze, 206
33100 Udine, Italy

Fabio Buttussi‡
HCI Lab
Dept. of Math and Computer Science
University of Udine
via delle Scienze, 206
33100 Udine, Italy

Abstract

The availability of more powerful mobile devices, sometimes equipped with graphics accelerators, is making it easier to experiment with mobile 3D graphics. In this paper, we exploit the main emerging standard in 3D rendering on mobile devices (OpenGL ES) to build a mobile player (called MobiX3D) for X3D and H-Anim content. The rendering engine of the MobiX3D player supports classic lighting and shading algorithms. We discuss the performance of the player and we apply it to sign language visualization.

CR Categories: H.5.1 [Information Interface and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.3.8 [Computer Graphics]: Applications

Keywords: 3D rendering, mobile devices, OpenGL ES, X3D, sign language

1 Introduction

In recent years, the increasing performance of mobile computing devices such as Personal Digital Assistants (PDAs) or high-end mobile phones has allowed these devices to support more and more complex applications. However, rendering 3D graphics on mobile devices is still considered a difficult task. Mobile devices are indeed characterized by some serious limitations with respect to desktop systems:

- limited CPU and memory;
- absence or limited performance of graphics accelerators;
- absence or limited performance of FPUs;
- energy consumption issues that limit designers of hardware and software for mobile devices;
- lack of powerful development and debugging environments.

For these reasons, research on mobile 3D rendering is still limited, but the increasing capabilities of mobile devices are now making it easier to experiment with 3D. The proposed approaches fall in three categories:

- hardware architectures for 3D rendering on mobile devices;
- remote 3D rendering architectures (the rendering process is carried out on a powerful remote server and the results are sent as a video to the mobile device using a wireless network);
- software architectures which carry out the entire 3D rendering process on mobile devices.

The third category is very promising, but has received less attention than the others. The most interesting results [Chang and Ger 2002; Duguet and Drettakis 2004] exploit techniques that are alternative to the traditional polygon-based 3D rendering. The goal of these approaches is to obtain a lower complexity for the 3D rendering process without losing much quality in the rendered scene. However, in the latest years, OpenGL ES [Khronos Group 2003] has emerged as a standard for 3D rendering on mobile devices and its approach is polygon-based.

In this paper, we experiment with polygon-based rendering of X3D content on mobile devices using the OpenGL ES 1.1 API to build a rendering engine that implements classic shading, lighting and navigation algorithms. This rendering engine is integrated into our X3D mobile player, called MobiX3D. At present, the MobiX3D player fully supports the H-Anim [Humanoid Animation Working Group 2004] standard and supports a large subset of the X3D Interactive profile. Moreover, the MobiX3D player has been used for sign language visualization on mobile devices. Our final goal is to support the X3D Interactive profile.

At the moment of writing, there are no released solutions for displaying X3D content on mobile devices and there is only one commercial solution for displaying VRML content on mobile devices, i.e. Pocket Cortona [ParallelGraphics 2004].

This paper is organized as follows. Section 2 surveys related work. Section 3 describes in detail our work, analyzing the architecture, the most interesting implementation details of the MobiX3D player, and the subset of X3D it supports. Section 4 describes the use of this player in the context of sign language visualization. Section

*e-mail:nadalutti@dimi.uniud.it
†e-mail:chittaro@dimi.uniud.it
‡e-mail:buttussi@dimi.uniud.it

5 discusses the performance of our player. Section 6 provides conclusions and outlines future research directions.

2 Related Work

2.1 3D Rendering on Mobile Devices

Although the literature about 3D rendering is wide, 3D rendering on mobile devices is still a scarcely explored subject. Moreover, since mobile devices have only recently reached a performance that allows them to manage 3D graphics, most of the available papers about 3D rendering on mobile devices focus on hardware or remote rendering architectures rather than on-board software solutions.

Research about hardware architectures for 3D rendering on mobile devices [Woo et al. 2002; Sohn et al. 2004; Kameyama et al. 2003] aims at producing very small graphic accelerator chips with high performance and low power consumption.

Remote rendering is the most common solution adopted in the literature for mobile devices. In this type of rendering, the process is carried out on a remote computer with powerful graphic acceleration and the results are sent (usually in video format) to the mobile device using a wireless network. As a result, the mobile device is just a simple client. Remote rendering has been used to: i) display complex objects, such as 3D anatomy models, on mobile devices [Grimstead et al. 2005; Lamberti et al. 2003; Sanna et al. 2004], and ii) implement augmented reality systems [Pasman and Woodward 2003].

The need for a wireless network is a major disadvantage of remote rendering solutions: wireless networks cannot be set up for every possible environment. Moreover, current wireless networks have limited bandwidth and these solutions need complex algorithms for the preparation of data to be sent to the client.

Interesting proposals in software architectures for 3D rendering on mobile devices concern alternative methods for 3D rendering such as image-based rendering and point-based rendering. Image-based rendering differs from polygon-based rendering in the input, i.e. a set of 2D color images containing depth information at each pixel (depth images). Each depth image also contains a matrix that describes the camera or viewing setup, i.e. its position and its direction. Because of its input, computational complexity of image-based rendering is generally lower than polygon-based rendering on mobile devices: it is linear in the number of pixels in the display, while polygon-based rendering is linear in the number of triangles (or other polygons) in the scene. This method has one main disadvantage: there can be gaps between neighboring pixels because every pixel value is computed independently. The rendering algorithm proposed in [Chang and Ger 2002] consists of two phases: i) a preprocessing phase that converts 3D models (provided as set of polygons) into depth images; ii) a phase that computes displayed images from depth images (warping).

Point-based rendering is a technique that renders the objects starting from their vertices. In [Duguet and Drettakis 2004], vertices are represented with hierarchical structures, called ρ-grids, i.e. three-dimensional recursive grids with a regular and uniform subdivision at each level. The subdivision factor is ρ x ρ x ρ for each cell. The position of each vertex can be implicitly determined by following the ρ-grid structure starting from the center of the screen. This structure is very compact and flexible and allows one to display the scene at different levels of detail. The experimental results obtained by Duguet and Drettakis [2004] proved that the optimal value for ρ is 3.

Other ideas for 3D rendering on mobile devices can be derived from architectures for 3D rendering that are aimed at low-power desktop computers. Tile-based rendering [Antochi et al. 2002] is a promising approach in this context. This technique decomposes a scene into smaller regions (tiles) and renders them one-by-one. The main advantage of this scheme is that a small local buffer can be associated to every tile, saving energy and improving efficiency. Tile size is a critical variable because the more accesses to local buffers are made the more energy is saved.

2.2 OpenGL ES

In the latest years, various libraries for 3D rendering on mobile devices have been proposed, e.g. miniGL [Digital Sandbox Inc. 2000] on Palm OS and PocketGL [PocketGL 2000] on Microsoft PocketPC were among the first and they were both subsets of OpenGL APIs.

In 2003, the Khronos Consortium [Khronos Group 2003] proposed OpenGL ES as a standard. After that, many libraries for 3D rendering on mobile devices are implementations of the OpenGL ES standard. The first API released by the Khronos Consortium was the OpenGL ES 1.0 API. It is a subset of OpenGL 1.3, eliminating redundancies and workstation functionalities that are not suitable for the context of mobile devices. The OpenGL ES 1.0 API uses fixed-point arithmetic that is more efficient than floating-point arithmetic in mobile devices without floating-point unit.

In 2004, the OpenGL ES 1.1 API was released. It improves version 1.0 by adding some OpenGL 1.5 functionalities, supporting hardware fixed-function accelerators and floating-point units, and improving power management.

In 2005, due to the increasing variety of mobile devices on the market, the Khronos Consortium organized the OpenGL ES APIs into two families: OpenGL ES 1.x for devices with fixed-function 3D graphics accelerators and OpenGL ES 2.x for devices with programmable 3D graphics accelerators. The OpenGL ES 2.0 API has been released in July 2005 and uses the GLSL language [Kessenich et al. 2004] for shading. The release of this API might stimulate the production of programmable 3D graphics accelerators for mobile devices, that could dramatically improve 3D graphics performance of mobile devices.

3 MobiX3D player

In this section we describe in detail the MobiX3D player. To develop our player, we used the Hybrid Rasteroid library [Hybrid Ltd. 2005], an implementation of the OpenGL ES 1.1 API, and the GlutES [Pouderoux and Marvie 2005] toolkit for the rendering operations. GlutES is the mobile version of the Glut toolkit for OpenGL and provides a set of high-level functions to manage 3D graphics with OpenGL ES.

The MobiX3D player currently supports a subset of the X3D Interactive profile and the full H-Anim standard. The X3D nodes currently supported by MobiX3D are listed in table 1.

Type	Name
Scene Graph Structure Node	X3D (Root)
	Scene
	Shape
Scene Graph Statements	Route
	DEF/USE
Metadata Node	MetadataDouble
	MetadataFloat
	MetadataInteger
	MetadataSet
	MetadataString
Base Geometry Node	Sphere
	Cone
	Cylinder
	Box
Composed Geometry Node	PointSet
	IndexedLineSet
	IndexedFaceSet
Grouping Node	Transform
	Group
	Anchor
	Inline
Appearance Node	Appearance
	Material
	ImageTexture2D
	Color
	Coordinate
	Normal
	TextureCoordinate
	TextureTransform
Bindable Node	Viewpoint
	Background
	NavigationInfo
Interpolator Node	PositionInterpolator
	ColorInterpolator
	RotationInterpolator
	ScalarInterpolator
	CoordinateInterpolator
	NormalInterpolator
Light Node	DirectionalLight
	PointLight
	SpotLight
Time-dependent Node	TimeSensor
Filter Node	BooleanFilter
Trigger Node	TimeTrigger
H-Anim Node	HAnimHumanoid
	HAnimJoint
	HAnimDisplacer
	HAnimSegment
	HAnimSite
Descrption Node	WorldInfo

Table 1: Nodes currently supported by the MobiX3D player

With the MobiX3D player, the user can navigate through the scene by pressing the cursor keys on the mobile device. Our player currently supports two classic navigation modes:

- pan: the user navigates the scene by moving the camera position parallel to the view plane;
- walk: the user navigates the scene using a walk-like behavior.

The rendering engine of the MobiX3D player supports three classic shading and lighting algorithms:

- wireframe: displays only the edges of the polygons in the scene;
- flat: associates only one color to every polygon in the scene;
- gouraud: assigns colors to every pixel within a polygon using a linear interpolation of the colors of its vertices.

3.1 Architecture

Figure 1 illustrates a high-level architecture of the MobiX3D player, that is organized in four modules:

- Renderer: this module manages all the rendering process, calling the other modules when necessary. Its input is a X3D file and its output is the display of the 3D scene.
- Event Manager: this module manages the events. It contains the set of all Routes of the X3D scene. Its input is a set of events (supplied from the Renderer) and its output is a set of changes in the attributes of X3D nodes that define the scene. Two types of events are handled: (i) user events: generated when the user interacts with the player (e.g., shading model changes); (ii) animation events: generated to implement animations (e.g., timers update).
- Scene Representation: this module contains an internal representation of the X3D scene graph. Its input is a set of changes sent by the Scene Manager and its output is a set of calls to the Renderer module for visualizing these changes on the display.
- Parser: this module parses the X3D file. Its input is a X3D file (supplied from the Renderer) and its output is the set of X3D nodes that defines the 3D scene (sent to the Scene Representation for initializing its scene graph) and the set of Routes that defines the animations (sent to the Event Manager for initializing its Route list).

3.2 Implementation

MobiX3D has been developed in C++ for the PocketPC 2003 platform. Two versions have been implemented: one using eMbedded Visual C++ 4.0, the other using Visual Studio 2005.

3.2.1 X3D Scene Management

The most complex parts of the MobiX3D player are those that manage X3D animations and the representation of scene graph.

In X3D, animations are implemented by a set of Routes which establish event paths between nodes. The nodes involved in an animation are usually TimeSensor and Interpolator nodes to build a time-dependent animation, TimeTrigger and BooleanFilter nodes to start a part of an animation immediately after the previous part has finished, and geometry nodes whose attributes are modified during the animation.
In the MobiX3D player, Routes are managed by a function called by a timer that clocks every 50 milliseconds (20 times per second). In this function, all the Routes statements in the X3D scene are solved and, if the animation is active, the values of TimeSensor nodes are updated. Then, the related Interpolator values are calculated and the geometry nodes attributes are updated. Finally, the values of

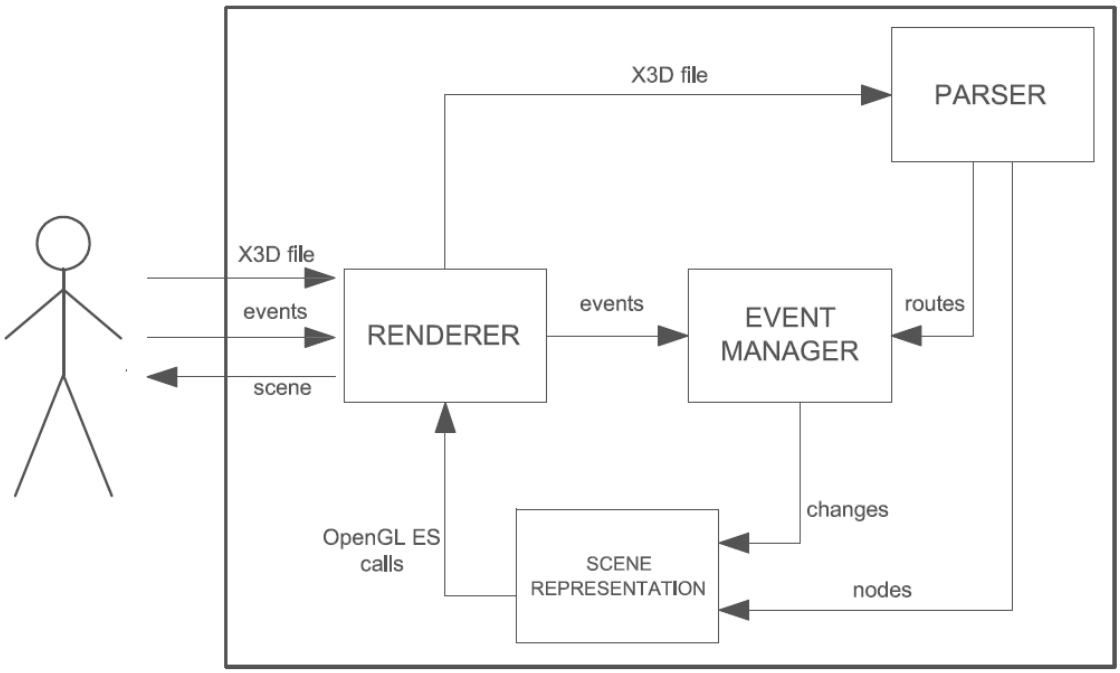

Figure 1: The architecture of MobiX3D.

TimeTrigger and BooleanFilter nodes are computed to start the next part of the animation.

The scene graph is represented by dedicated classes in the Scene Representation module. Every X3D node supported by our rendering engine is represented by its related class. Every class related to a X3D node contains two methods: draw and parse. The draw methods contain the OpenGL ES calls required to render the X3D nodes which they belong to. The parse methods contain the parsing instructions for the X3D nodes which they belong to. These methods have a fundamental role in our rendering engine: the initial values of the attributes of all nodes are retrieved and stored in memory, while parsing the X3D file, by sequentially calling the parse methods of the nodes. X3D scene drawing is carried out by sequentially calling the draw methods of the X3D nodes in the scene.

3.2.2 Some implementation issues

The issues encountered while developing our rendering engine are related to the limitations of OpenGL ES that implements only a subset of OpenGL functionalities.

Firstly, OpenGL ES supports only the rendering of triangles, lines and points, while OpenGL supports the rendering of all (convex) polygons, lines and points. The IndexedFaceSet node in X3D specifies a 3D shape composed by a structured set of polygons (with any number of edges). Each polygon is specified by a list of indexes into the vertex coordinates. These indexes can be specified indifferently in clockwise or counterclockwise order. So, to support this node, we had to develop an algorithm that converts every convex polygon into triangles.

Given a polygon P, with n edges, our triangulation algorithm can be formalized as follows: (i) polygon vertices are labeled with numbers from 1 to n, starting from an arbitrary vertex. The first vertex is labeled with the number 1, the second with number 2 and the last vertex with number n; (ii) the polygon is divided into the following set of triangles: $\{(1,2,3),\ldots,(1,\ n-1,\ n)\}$. Figure 2 illustrates an example of a polygon triangulated by this algorithm.

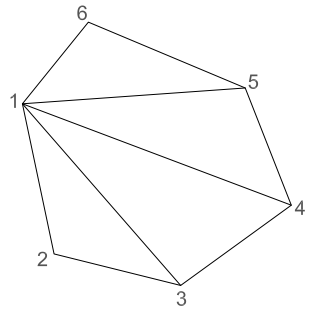

Figure 2: An example of triangulation of a convex polygon with the adopted algorithm.

This algorithm is not the best for triangulating polygons because it assumes that the polygon to triangulate is convex and forces the first vertex to belong to all the resulting triangles. We chose to use this algorithm because its complexity is linear and it does not require additional data structures to work: we have embedded this algorithm in the Parser and, when the Parser recognizes a new polygon, the algorithm automatically triangulates it using the Parser data structures. In literature there are some good algorithms (e.g. [Narkhede and Manocha 1995]) that support also non-convex polygons, but they are more complex ($O(n\ log\ n)$) and they would not be easily integrated in the Parser because they require additional data structures.
The chosen algorithm could be implemented with the OpenGL ES instruction glDrawArrays(GL_TRIANGLE_FAN), but this

instruction has different behaviors between clockwise and counterclockwise vertex orders (a counterclockwise order indicates front facing triangles, a clockwise order indicates back facing triangles). To avoid back-face culling of solid objects, we do not use the glDrawArrays instruction to triangulate polygons: our algorithm always orders the vertices of the obtained triangles in counterclockwise order.

Moreover, in OpenGL ES there is only one way for rendering a set of primitives. For example, to render a set of triangles one has to proceed in three steps: (i) declare an array for the coordinates of the vertices of all the triangles which will be rendered; (ii) use the glVertexPointer instruction to create a pointer to the vertices defined in the array; (iii) use the glDrawArrays(GL_TRIANGLES) instruction to render simultaneously the triangles.
In OpenGL, other approaches are available to render a set of triangles. The most used approach goes through the following steps: (i) execute the glBegin(GL_TRIANGLES) instruction to start the list of vertices; (ii) use the glVertex instruction to specify the coordinates of each vertex (a glVertex call can specify only a single vertex); (iii) use the glEnd instruction to end the list of vertices. This approach is more flexible because vertex coordinates can be calculated while drawing the primitives (between the glStart and glEnd instructions) and it is not necessary to specify the number of primitives that will be rendered.

4 Sign Language Application

The MobiX3D player has been used for displaying sign language sentences on mobile devices. Sign language visualization can be useful for teaching sign language to deaf children, for defining sign language visual dictionaries or, coupled with a speech recognition engine, for translating natural language into sign language. However, displaying sign language is a difficult task: if animations are not very precise the user can misunderstand the meaning of the gestures. Moreover, when there is no gesture associated to a word, it is necessary to fingerspell the word. Fingerspelling consists in spelling a word letter-by-letter with fingers movement and it is very difficult to display due to the little size of fingers and the similarity of the movements associated to the letters.

For these reasons, although sign language visualization has been studied in the literature, no proposals concern mobile devices because of their limited performance.
Sign language visualization is being used for teaching sign language to deaf children [Karpouzis et al. 2006; Geitz et al. 1996; Sagawa and Takeuchi 2002] or for defining sign language visual dictionaries [Wilcox et al. 1994]. The ViSiCAST project [Elliott et al. 2000] aims at improving the accessibility of public services by using monitors that display humanoids performing sign language translation of what employees say. Displaying a humanoid is a better solution than using text subtitles for two reasons: (i) most deaf people consider sign language as their mother language; (ii) humanoids can attract user's focus of attention and can convey additional conversational and emotional cues.

We have used the MobiX3D player to display a H-Anim humanoid that performs sign language on mobile devices. To improve the performance, in this application:

- we use a simplified, but realistic humanoid (about 6000 triangles);
- we disable navigation through the 3D scene to avoid possible loss of gestures due to incorrect viewpoint;
- there is one fixed lighting source and gouraud is used as the only possible shading model.

Figure 3 shows the H-Anim humanoid used in our tests while performing sign language. A phrase in sign language is implemented by an X3D file that contains a H-Anim humanoid and the animation, implemented by a structured series of TimeSensors, TimeTriggers, BooleanFilters, OrientationInterpolators and Routes. Sign language sentences used in our tests were built with the H-Animator system [Buttussi et al. 2006]. We used it to model sign language gestures and to concatenate them into sign language sentences.

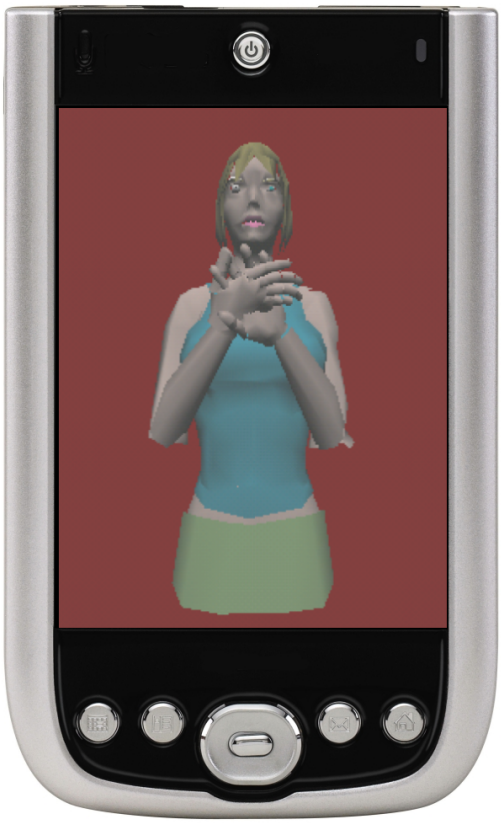

Figure 3: Humanoid performing sign language.

5 Performance

We tested the MobiX3D player on PocketPC devices, such as the Acer n10 and the Dell Axim X50V Pocket. The X50V has a 624 MHz processor with 64 MB of main memory and an Intel 2700G graphics processor with 16 MB video RAM.

We tested performance of our player using three different implementations of the OpenGL ES API:

- Rasteroid [Hybrid Ltd. 2005]: a proprietary implementation of OpenGL ES 1.1 API;
- Vincent [Will 2004]: an open-source implementation of OpenGL ES 1.1 API;
- Intel 2700G [Intel 2004]: implementation of OpenGL ES 1.0 API provided as interface by Intel 2700G graphics processors.

The 2700G has fixed-point arithmetic, so, unlike Rasteroid and Vincent, the Intel 2700G implementation does not support floating-point OpenGL ES instructions. Therefore, to test our player with the Intel 2700G library we had to substitute all floating-point instructions of the rendering engine with the corresponding fixed-point ones. Thus, to compare the performance of the three different implementations, we used a version of the MobiX3D player with fixed point OpenGL ES instructions, with a loss of accuracy in the representation of scenes.

While Rasteroid and Vincent carry out the entire rendering process via software, the Intel 2700G exploits some hardware acceleration functions.

Figure 4: Visualization of the Simple World.

To test the performance of the MobiX3D player, we have chosen a benchmark composed by three models:

- Simple World: a model of a very simple world with textures (Figure 4). It consists of four huts and four trees, built by using simple geometry nodes (cylinders, spheres, cones and boxes);
- Humanoid: the humanoid described in Section 4, animated and without textures (Figure 3);
- Udine3D: a simplified version of a model of a square in the city of Udine [Udine3D 2004] (Figure 5). This model was originally developed in VRML for the Web, then simplified for its visualization on mobile devices [Burigat and Chittaro 2005] and now converted into X3D for testing our player. It is textured and it needs transparency support to be correctly visualized.

Figure 5: Visualization of the Udine3D model.

Table 2 illustrates the main features of the three models. This benchmark tests the most complex features of the MobiX3D player: animations, textures and transparency. Moreover, to put under stress the player, we chose to have a high resolution for the X3D Sphere (20 meridians and 20 parallels), Cylinder and Cone (20 slices and 5 loops for base circles) primitives. So, these primitives are always composed respectively by 800, 440, and 240 triangles. This explains the number of triangles obtained in the Simple World and Udine3D benchmarks. The Humanoid benchmark is instead composed by only IndexedFaceSet primitives with triangular faces: its size in triangles does not vary among different X3D primitive implementations.

Benchmark	Simple World	Humanoid	Udine3D
Triangles	6584	6134	30666
Textures	0.87 MB	No	5.54 MB
Animations	No	Yes	No
Transparency	No	No	Yes

Table 2: Models used as benchmarks

Table 3 lists the performance obtained with the MobiX3D player.

Frame rates have been measured while navigating the scene for the Udine3D and Simple World benchmarks and while the animation was performed for the Humanoid benchmark.

Benchmark	Simple World	Humanoid	Udine3D
Rasteroid Version	6.2	4.4	1.4
Vincent Version	5.6	3.9	1.1
Intel 2700G Version	15.9	10.2	3.9

Table 3: Performance of MobiX3D in frames per second

Performance obtained by the Intel 2700G version is about 2-3 times better than that obtained by the Vincent and Rasteroid versions. This difference in performance is due mainly to hardware acceleration that is exploited by the Intel 2700G library. Moreover, the quality of the rendered image is better in the Intel 2700G version because it uses the texture compression algorithms provided by the graphics accelerator. Vincent and Rasteroid performance are comparable, with Rasteroid being slightly more efficient.

In our tests, performance obtained with the Intel 2700G version is slightly worse than that obtained with the Pocket Cortona VRML browser. However, Pocket Cortona does not use texture compression algorithms provided by Intel 2700G graphics accelerator and its texture accuracy is slightly worse than the Intel 2700G version of MobiX3D.

6 Conclusions and Future Work

To the best of our knowledge, MobiX3D is the first publicly available X3D player for mobile devices. The final goal of our project is to support the full X3D Interactive profile and the H-Anim standard. MobiX3D can be downloaded at http://hcilab.uniud.it/MobiX3D. With the MobiX3D player, we currently obtain a 3.9 fps navigation through a X3D world with about 30000 triangles and 5.54 MB of textures.

Our research is now proceeding in several directions. First, we will use more efficient algorithms for shading and texturing, like those described in [Ström and Akenine-Möller 2005].
Second, we will consider to port MobiX3D to other platforms (e.g. Linux, Symbian, or Palm).
Third, we will improve the sign language visualizer, modeling more accurate gestures and improving the humanoid. Then we will test our sign language visualizer with deaf users on sign language translation applications.
Finally, we will study the possibility of implementing seamless shape deformation algorithms (e.g. [Babski and Thalmann 1999]) on mobile devices. Seamless shape deformation algorithms map the humanoid into a single mesh and the movements into deformations of this mesh. These deformations are parametric and simulate the behavior of skin, muscles and bones. These algorithms tackle the discontinuity problems in the movement of traditional humanoids, implemented with a set of independent meshes.

7 Acknowledgements

Roberto Ranon and Stefano Burigat provided precious advice during the development of the described work.
Our research has been partially supported by the Italian Ministry of Education, University and Research (MIUR) under the PRIN 2005 project “Adaptive, Context-aware, Multimedia Guides on Mobile Devices”.

References

ANTOCHI, I., JUURLINK, B., AND VASSILIADIS, S. 2002. Selecting the Optimal Tile Size for Low-Power Tile-Based Rendering. In *ProRISC 2002: Proceedings of the thirteenth Annual Workshop on Circuits, Systems and Signal Processing*, 1–6.

BABSKI, C., AND THALMANN, D. 1999. A Seamless Shape for HANIM Compliant Bodies. In *VRML '99: Proceedings of the fourth symposium on Virtual reality modeling language*, ACM Press, New York, NY, USA, 21–28.

BURIGAT, S., AND CHITTARO, L. 2005. Location-aware Visualization of VRML Models in GPS-based Mobile Guides. In *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*, ACM Press, New York, NY, USA, 57–64.

BUTTUSSI, F., CHITTARO, L., AND NADALUTTI, D. 2006. H-Animator: A Visual Tool for Modeling, Reuse and Sharing of X3D Humanoid Animations. In *Web3D '06: Proceedings of the eleventh international conference on 3D Web technology*, ACM Press, New York, NY, USA.

CHANG, C.-F., AND GER, S.-H. 2002. Enhancing 3D Graphics on Mobile Devices by Image-Based Rendering. In *PCM '02: Proceedings of the Third IEEE Pacific Rim Conference on Multimedia*, Springer-Verlag, Berlin, Germany, 1105–1111.

DIGITAL SANDBOX INC. 2000. MiniGL. http://www.dsbox.com/minigl.html.

DUGUET, F., AND DRETTAKIS, G. 2004. Flexible Point-Based Rendering on Mobile Devices. *IEEE Computer Graphics and Applications 24*, 4, 57–63.

ELLIOTT, R., GLAUERT, J. R. W., KENNAWAY, J. R., AND MARSHALL, I. 2000. The Development of Language Processing Support for the ViSiCAST Project. In *Assets '00: Proceedings of the fourth international ACM conference on Assistive technologies*, ACM Press, New York, NY, USA, 101–108.

GEITZ, S., HANSON, T., AND MAHER, S. 1996. Computer Generated 3-Dimensional Models of Manual Alphabet Handshapes for the World Wide Web. In *Assets '96: Proceedings of the second annual ACM conference on Assistive technologies*, ACM Press, New York, NY, USA, 27–31.

GRIMSTEAD, I. J., AVIS, N. J., AND WALKER, D. W. 2005. Visualization Across the Pond: How a Wireless PDA can Collaborate with Million-Polygon Datasets via 9,000km of Cable. In *Web3D '05: Proceedings of the tenth international conference on 3D Web technology*, ACM Press, New York, NY, USA, 47–56.

HUMANOID ANIMATION WORKING GROUP. 2004. H-Anim. http://h-anim.org.

HYBRID LTD. 2005. Rasteroid. http://www.hybrid.fi/main/esframework/index.php.

INTEL. 2004. Intel 2700G Graphics Accelerator. http://www.intel.com/design/pca/prodbref/300571.htm.

KAMEYAMA, M., KATO, Y., FUJIMOTO, H., NEGISHI, H., KODAMA, Y., INOUE, Y., AND KAWAI, H. 2003. 3D Graphics LSI Core for Mobile Phone "Z3D". In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 60–67.

KARPOUZIS, K., CARIDAKIS, G., FOTINEA, S.-E., AND EFTHIMIOU, E. 2006. Educational Resources and Implementation of a Greek Sign Language Synthesis Architecture. *Computers & Education, Special Issue in Web3D Technologies in Learning, Education and Training, Elsevier*. (In press).

KESSENICH, J., BALDWIN, D., AND ROST, R. 2004. The OpenGL Shading Language. http://www.opengl.org/documentation/oglsl.html.

KHRONOS GROUP. 2003. OpenGL ES. http://www.khronos.org/opengles/.

LAMBERTI, F., ZUNINO, C., SANNA, A., FIUME, A., AND MANIEZZO, M. 2003. An Accelerated Remote Graphics Architecture for PDAs. In *Web3D '03: Proceeding of the eighth international conference on 3D Web technology*, ACM Press, New York, NY, USA, 55–61.

NARKHEDE, A., AND MANOCHA, D. 1995. Fast Polygon Triangulation Based on Seidel's Algorithm. In *Graphics Gems V*, A. W. Paeth, Ed. AP Professional, Boston, MA, USA, 394–397.

PARALLELGRAPHICS. 2004. Pocket Cortona. http://www.parallelgraphics.com/products/cortonace/.

PASMAN, W., AND WOODWARD, C. 2003. Implementation of an Augmented Reality System on a PDA. In *ISMAR '03: Proceedings of the second IEEE and ACM International Symposium on Mixed and Augmented Reality*, IEEE Computer Society, Washington, DC, USA, 276.

POCKETGL. 2000. http://www.sundialsoft.freeserve.co.uk/pgl.htm.

POUDEROUX, J., AND MARVIE, J. E. 2005. GlutES. http://iparla.labri.fr/softwares/glutes/.

SAGAWA, H., AND TAKEUCHI, M. 2002. A Teaching System of Japanese Sign Language Using Sign Language Recognition and Generation. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, ACM Press, New York, NY, USA, 137–145.

SANNA, A., ZUNINO, C., AND LAMBERTI, F. 2004. A Distributed Architecture for Searching, Retrieving and Visualizing Complex 3D Models on Personal Digital Assistants. *International Journal of Human Computer Studies 60*, 701–716.

SOHN, J.-H., WOO, R., AND YOO, H.-J. 2004. A Programmable Vertex Shader with Fixed-point SIMD Datapath for Low Power Wireless Applications. In *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM Press, New York, NY, USA, 107–114.

STRÖM, J., AND AKENINE-MÖLLER, T. 2005. iPACKMAN: High-quality, Low-complexity Texture Compression for Mobile Phones. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, ACM Press, New York, NY, USA, 63–70.

UDINE3D. 2004. http://udine3d.uniud.it.

WILCOX, S., SCHEIBMAN, J., WOOD, D., COKELY, D., AND STOKOE, W. C. 1994. Multimedia dictionary of American Sign Language. In *Assets '94: Proceedings of the first annual ACM conference on Assistive technologies*, ACM Press, New York, NY, USA, 9–16.

WILL, H. M. 2004. Vincent. http://sourceforge.net/projects/ogl-es.

WOO, R., YOON, C. W., KOOK, J., LEE, S. J., AND YOO, H. J. 2002. A 120mW 3D Rendering Engine with 6Mb Embedded DRAM and 3.2Gbyte/s Runtime Reconfigurable Bus for PDA-Chip. *IEEE Journal of Solid-State Circuits 37* (Oct.), 1352–1355.