An open-source framework for air guitar games

Lucas S. Figueiredo, João Marcelo X. N. Teixeira, Aline S. Cavalcanti, Veronica Teichrieb, Judith Kelner
Virtual Reality and Multimedia Research Group
Center of Informatics of UFPE
Recife-PE, Brazil
Email: {lsf, jmxnt, vt, jk} @cin.ufpe.br, aline@gprt.ufpe.br

***Abstract*—This paper presents an open-source framework for developing guitar-based games using gesture interaction. The goal of this work was to develop a robust platform capable of providing seamless real time interaction, intuitive playability and coherent sound output. Each part of the proposed architecture is detailed and a case study is performed to exemplify its easiness of use. Some tests are also performed in order to validate the proposed platform. The results showed to be successful: all tested subjects could reach the objective of playing a simple song during a small amount of time and the most important, they were satisfied with the experience.**

***Index Terms*—air guitar; music game interaction;**

I. Introduction

Over the past few years, game industry has added peripheral tools to games enhancing gameplay experience. These peripherals have also increased levels of active engagement with games and widened the appeal of games to audiences who may never have played them before. Examples of such tools include the Eye Toy [1], Dance Mat [2], the Wii motion controller [3], shown in Figure 1, among others. One of the most successful examples is the evolution of game interaction with the advent of music games.

Fig. 1. Console peripherals, from left to right: PS3 Eye Toy, Dance Mat and Wii basic controllers.

The concept of music games refers to video games in which the gameplay is meaningful and often almost entirely oriented toward the player's interactions with a musical score or individual songs. In the last decade, they have conquered a significant space on game market, and in consequence, game industry is now focusing its efforts on musical instruments and simplified devices that favor user interaction.

Video game interaction represents a research area in constant evolution. It has started with game-specific, few functional controllers, to more sophisticated ones, including features such as motion sensors and adaptable design formats, as shown in Figure 2. Recently, Microsoft presented Project Natal [4], a system containing depth-sensors, a camera and a microphone (shown in Figure 3), which claims to be capable of replacing conventional controllers. Such technology would allow a person to use his/her own arms and legs to directly interact with the game.

Fig. 2. Controller evolution example, from the Atari controller (left) to the sophisticated Wii Wheel and Wii Zapper ones (right).

Fig. 3. Project Natal sensors.

This "controller-less" approach may be applied to games similar to Guitar Hero or Rock Band. The idea of playing an "air guitar" refers to imitate rock guitar gestures along music without the actual physical instrument. It is more showmanship than musical performance and does not require real musical skills. Instead of just acting along with the imaginary guitar, the user could control an actually playable "virtual" instrument, capable of emitting sounds according to his/her movements.

Concerning the project of an "Air Guitar game", it is possible to emphasize some requirements in order to obtain a robust and usable application:

- Interaction has to occur in real time. Since the user is playing music, the time constraint is very strict and implies that the delays on visual and audio responses are minimum. This may be obtained by using simple image processing techniques, capable of executing on most computer configurations.
- User actions may mimic the act of playing a real guitar. There is no meaning to create an "air guitar" application if users must press buttons instead of holding an instrument and actually playing it.

978-0-7695-3963-8/09 $26.00 © 2009 IEEE
DOI 10.1109/SBGAMES.2009.17

- Realistic sound response is required. The sound output may reflect what should happen in case the user was playing a real guitar. Simplifications can be used, in order to make the interaction easier, but the sensation of immersion may only be reached by correct action-reaction correspondence.

In this context, an open-source framework for developing guitar-based games using gesture interaction is proposed. The goal is to develop a robust platform capable of satisfying the previously mentioned requirements that can be easily extended and usable in order to allow integration in interactive applications such as music games. The main contribution is the framework itself, since no similar open-source tool was found in the literature. Making available such tool should drive the gaming community to develop more and better music games using the "controller-less" approach.

This paper is organized as follows. The next section presents some related work regarding music games, specifically about guitar-based applications. Section III describes the proposed framework architecture, the main technologies and the techniques used. Section IV shows as case study an example application and the results obtained performing user tests. Finally, some final remarks are presented in Section V and future works are pointed in order to improve the proposed framework.

II. Related Work

Many works share their piece of contribution to the air guitar framework's idea. Among them, music games play their role as the main motivator of this work. Due to the increased popularity of this type of game in the last few years, it has been conquering space in the current game market [5]. Adagio [6], GuitarFreaks [7], FreQuency [8] and DJ Hero [9] are some games that can be cited, varying from simple flash-based web implementations to more sophisticated ones made for the latest console generations.

The games cited before served as a starting point for the evolution that was going to happen. Frets on Fire (an open-source music game for PC) [10] suggested a different use for the keyboard, in which the player should hold it using both hands, simulating an electric guitar. Games such as Guitar Hero [11] and Rock Band [12] improved user interaction by adopting simplified musical instruments for game input. Consequently, they remained for a long time on the top of sold games, and still are a fever among players scattered all over the world.

Starting with plastic guitars, the use of specific input devices simulating musical instruments for game control has been popularized. Nowadays, it is possible to form an entire band by the use of drums, microphones and guitars (see Figure 4), all played by different users. As a way of improving or evolving the current user interaction available on games, especially on music games, a new type of interaction was necessary. The concept of the proposed framework was influenced by a great number of input devices and user interaction methods, such as Dance Mats, Wiimotes and the recently announced Project Natal (presented at the E3 conference in June 2009 and not yet available). From those three aforementioned, Project Natal is the one with closest relation to the proposed work, since it also deals with the recognition of player's movements without using wires or buttons for interaction. Another example of body motion capture is found in [13]. They proposed a system that recognizes some user intentions into a dancing performance and uses this information as input for some tasks, i.e. sound generation.

Fig. 4. Input devices for music games. Electric guitars joysticks made based on a Gibson SG on the left, and on the right a simplified drums joystick.

The Virtual Air Guitar [14], [15] project is the related work that mostly approximates to this work. In fact, it implements basically the same features that are available on the proposed framework, and also uses the same image processing methods. However, since The Virtual Air Guitar project happens to be a proprietary solution, there is no code neither an executable demo application available to the community. The same occurs with the Virtual Slide Guitar project [16], a similar work that differs basically by the capture method, using infra-red sensors. There is no downloadable source code or a demo application of it. So, some of the major contributions of this framework is the fact that it is open-source, makes use of a simple but robust color tracking algorithm (which favors application performance), interprets movements like a virtual guitar being played, and provides developers with a hands-on demo application, that can be modified and used as a start point for more complex simulations.

Besides applications similar to guitar games, the air guitar [17] practice was analyzed as well. It is important to notice that Air Guitar is a consolidated activity, even with championships disputed all over the world (the oldest one is in its 14th edition). The performance of Air Guitar players was observed and evaluated regarding the movements that could be incorporated in the framework. Some of them, such as fingered-ones, are not detected using the color tracking method proposed in this paper; in spite of that, the framework interaction is based on their hand movements. The idea of moving a hand to play the strings and the other one to choose the chords to be played has

been focused on this research. Details regarding framework's implementation, example applications and tests performed are detailed further.

III. The Framework

The proposed framework consists of a tool capable of enabling the player to use his/her hands as an "input device" for games and other entertainment applications. More specifically, it focuses on transforming the "air guitar" performance in a way of interaction. The idea is to provide the game developer with the basis for constructing the interaction with the "imaginary guitar", in a way that it is possible to adjust the framework's use according to the needs of the project being developed.

The complete framework is modularized in order to be easily used, altered and updated. The overall architecture is presented in Figure 5. It represents a pipeline of a generic application containing all five major steps involved in the air guitar processing: input image acquisition, image processing, guitar simulation, sound generation and rendering. It is important to notice that the framework makes use of other libraries that provide support for each processing phase. DSVideoLib [18] was used for capturing webcam's image, FMOD Ex [19] was responsible for playing the output sounds and all visual results were rendered with OpenGL [20]. The entire code is written using the C++ language. Each pipeline step is better described in the next subsections.

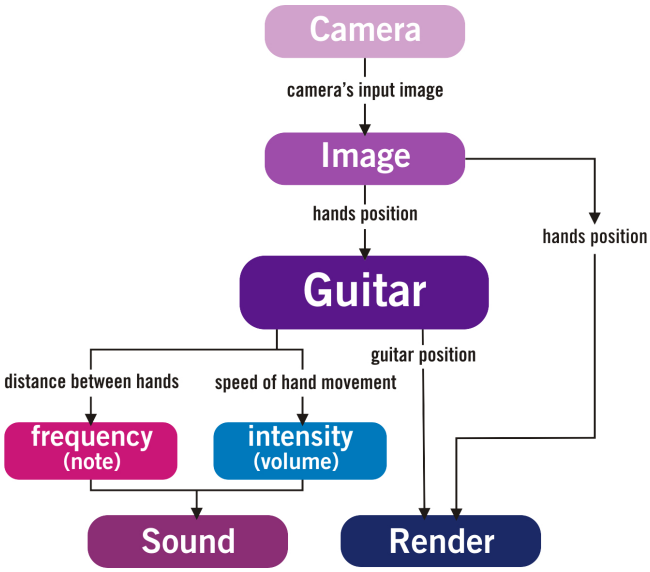

Fig. 5. Framework pipeline architecture.

A. Camera

Tracking the player's hands can be achieved by using many different techniques. Haptic gloves, Wiimotes or model-based hand tracking [3] are well-known alternatives for performing such task. The proposed framework adopts a color tracking approach as it proved to be the best cost-benefit method to acquire knowledge about the position of user's hands in real time. This type of tracking requires, besides the PC itself, a common webcam and a pair of gloves to be worn by the user, with a color that contrasts well with the surrounding environment.

This method presents many advantages, listed as follows. The first one is the cost of required equipment, which is lower when compared to the price of haptic gloves or even Wiimotes. The processing load demanded by the algorithm is low, which makes possible its real time execution. It is also robust to occlusions, rapid motions and blur, being capable of recovering the tracking immediately after such events.

The Camera module comprises all implementation regarding webcam configuration and image capture. It has functions that provide access to different webcam information, such as the acquisition of data from the last frame (for further rendering). This module communicates with the webcam attached to the computer using the DSVideoLib library.

B. Image

Given the current frame, obtained by accessing Camera functions, the color tracking is then initialized. The role of the Image module is to find the two groups of pixels that represent both player's hands (assuming they are already wearing the gloves). This information is used on the estimation of hands' screen position. The processing phase responsible for this task searches for all pixels inside the current frame that are within a pre-defined range of color values and then organizes the encountered groups as described next.

The search starts at the top left pixel and iterates over all subsequent ones. The first step attempts to create color groups based on the searched color along all image. In order to recognize if a pixel remains within the specified color range, the relationship between its RGB components is analyzed. For example, in case the application is looking for yellow gloves, the search takes into account that both Red and Green components must present similar values and that the Blue component value has to be considerably lower than the other ones. This approach works well since the relationship between RGB components is robust to most illumination changes.

A labeling algorithm [21] is a procedure for assigning a unique label to each object (a group of connected components) in an image. This type of algorithm is used for any subsequent analysis procedure and for distinguishing and referencing labeled objects. Labeling is an indispensable part of nearly all applications in pattern recognition and computer vision. In this work, a specific labeling algorithm was implemented in order to locate the regions corresponding to the selected glove colors. It is described as follows.

Initially, an image map is created in order to store the pixels already visited, and the ones marked with a valid label. For each pixel, a verification is performed to check if its color matched the range of the one searched. In case there is no match, the pixel is marked as visited in the image map and the search continues on the next adjacent pixel. If there is a match, then a pixel group object is created and the search

starts using the current pixel as origin. The search considers a 4-connectivity neighborhood scheme (vertical and horizontal - up, down, left and right directions) and takes as radius size **n** a customizable value (the default value is 2). This value guarantees that even if a pixel is separated from the origin pixel by a small distance, according to the search radius, it still can be included in the pixel group.

The labeling algorithm developed adopts a greedy approach, where each successfully neighbor found (a pixel that is within the specified color range) is added to the initial pixel group and a new search is initiated using this pixel as start point. It is important to highlight that the search for pixel groups always marks the visited pixels in the image map and such pixels are not considered on further searches. Since small image resolutions are used (320 x 240 pixels), this guarantees an acceptable processing time for real time applications. When a pixel group search ends (there are no more successful neighbors found), the algorithm continues to iterate across the image, ignoring already marked pixels and creating new pixel groups, until the entire image is visited.

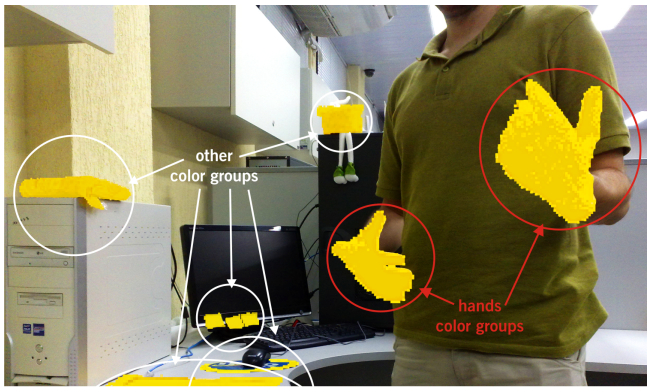

Fig. 6. Color groups composed by yellow sets of pixels. The greater groups almost ever represent the player hands.

In sequence, after all pixel groups are already constructed, the two with the majority of pixels are selected. In case the size of these two groups is lower than a pre-defined threshold, the processing stops and the current frame is considered as tracking failure. If this is not the case, these two greater groups will represent the hands of the player as shown in Figure 6. This is a guaranteed way of finding the player hands, considering that the gloves' color contrasts clearly with the background environment.

In case there are other elements that match the color requisite, they must be represented by smaller pixel groups, since the hands of the player are located near the camera and in consequence they occupy a larger area on the screen. For simplification reasons, only the center of each hand is considered on further processing. The center of a color group (also known as geometric center, or just centroid) is obtained by summing all pixel coordinates (horizontal and vertical) related to that pixel group and then dividing the result by the amount of pixels of the group.

Once both centers are calculated, the one located on the leftmost part of the screen refers to the left hand and the other to the right one. Considering that when playing the guitar a person rarely inverts the side of his/her hands, this approach presents a good result. Besides this functionality, the Image module provides developers with functions to access the encountered color groups in a way that it is possible for him/her to understand and alter some framework behaviors.

C. Guitar

Considering the hand points were obtained, the virtual guitar simulation is initialized. The Guitar module task is to detect player's intentions when performing movements similar to the act of playing a real guitar. The idea is to automatically interpret his/her movements and give as return both visual and audio cues. The guitar model available from the framework corresponds to a simplified version of common electric guitars, since it has less frets and a single string. This simplification is fundamental in order to make possible the player interaction. Playing with more strings would make the interaction impracticable, since the hand movements should be much more precise (as happens with a real guitar). The number of frets is customizable, but it is highly recommendable to use only a few (from 2 to 10 frets) instead of the number corresponding exactly to a real guitar (about 22/23 in most cases), due to the same problem described earlier (precision of interaction).

The guitar simulation process works as described next. The result of this process is the correct placement of a virtual guitar based on the position of the hands. Assuming that the left hand is positioned over guitar's arm (it is possible to change this configuration), the following steps are executed in sequence:

1) The two points in red shown in Figure 7 are used as reference points, found based on the hands locations. According to them, it is possible to make an approximate estimative of guitar's arm and body positions. Both arm and body possess motion speeds, which are defined before the application is running. Such speeds are used to make the guitar follow the user hands, as if the player were holding it. Since the left hand movements follow the guitar axis direction and the right hand movements are perpendicular to this same axis, different motion speeds are applied to the reference points. Because of its perpendicular amplitude of movement, the right hand presents a higher motion speed than the left one.
2) Whenever the current frame represents the first successful frame tracked (i.e. all previous frames were considered tracking failures or this was the first frame captured from the webcam), the reference points will correspond exactly to the hands points. If this is not the case, both guitar's arm and body will follow respectively left and right hand points, as shown in Figure 7.
3) The guitar movement is described by the following algorithm. The position of reference points is updated based on a fixed percentage of the difference vector between reference and hand point. Such percentage

corresponds to the motion speed of guitar's arm or body. If the distance found is lower than a customizable jitter threshold, then the reference point keeps its position. This approach guarantees a soft displacement for the guitar's arm and body points, which favors visualization and interaction experience.

4) The player interaction is defined as follows. Whenever he/she crosses the "string line" (the same as the guitar axis), the framework captures the intention of playing. The fact that the guitar's body moves slower than the arm helps the user preventing non-intentional sounds. On the other hand, the speed of guitar's arm does not need to be small.

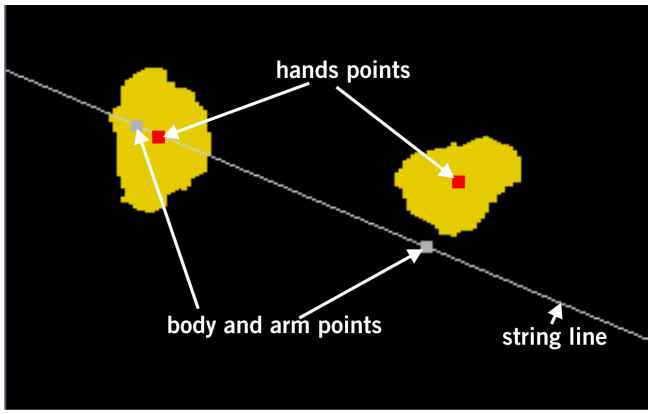

Fig. 7. Guitar points (arm and body) following the hands points. Note that the arm point is much closer to its corresponding string point than the body point. This occurs due to the motion speed of each one.

5) The framework makes use of other few mechanisms to guarantee that players's interaction with the air guitar was correctly captured. The jitter threshold is one of them. Because of it, the string line stops following the player's hands whenever they are too close to the line. This way, the string line will not cross the right hand point by its own. Another important policy is that only high intensity movements represent the intention of playing. These are measured by the distance between last and current right hand points. Finally, a single cross direction is taken into consideration, down or up. In case both were used, the sound would eventually be played even if it was not user's intention.
6) Simultaneously to the capture of player's intention, two values are stored for further use by the Sound module. Such values correspond to the captured intensity and distance between right and left hands at the moment the player crossed the string line.

D. Sound

After gathering both distance and intensity values, as described earlier, the sound processing starts. The module must be adequately configured before running in order to work properly. At first, the number of frets needs to be specified. Based on this information, the virtual guitar arm is divided into different frets that play distinguishable sounds. The closer the fret is to the guitar's body, the higher is the frequency of the corresponding sound.

Every sound generated by the virtual guitar is related to a previously chosen base sample, which is recommended to be a played single note stored in any format supported by FMOD Ex (.wav, .mp3 among others). Such base sample will be played whenever the user plays the guitar, what means that the intensity entry value has surpassed the intensity threshold (predefined value of the Sound module). The base sound will be played in a way that its frequency will be modified according to the fret pointed by the user's left hand. In other words, the Sound module takes into consideration the information of the loaded standard frequency audio sample (sound file) and increases its frequency whenever necessary. The first fret of the guitar (the farthest to the guitar's body) will correspond to the base sound, while the next ones will increment the frequency played as described next.

The Sound module uses the twelve-tone equal tempered scale as base to perform frequency increments. It was chosen because of its segmentation method, which is widely used in current music panorama. Almost all fret-divided instruments use it, and it represents the division that most approximates the just intonation (a segmentation which presents a not equal division but has perfect consonant intervals) without adding a lot of additional notes. The difference between the tempered scale and the reference case (the just intonation) is not higher than 1%, as shown in Table I. The just intonation could also be used, but at the same time it presents perfect intervals, in some combinations it does not perform well. Other equal tempered scales could also be used, since the number of divisions per octave (currently twelve) can be easily altered in real time (e.g., some Indian musicians use 31 divisions).

This way, the initial frequency of the base sample is altered by multiplying its value by $2^{1/12}$ (approximately 1.0595) as many times as the number indicated by the current fret. The base sound is considered the matrix for all other sounds, even in case it is not a well known note, only preserving the relative relationship between them; i.e., the absolute tuning considering A4 as 440Hz is not taken into account, since the objective is to guarantee that different frets perform coherent sounds, differing from each other.

Some improvements can be adopted in order to make the interaction more attractive. To avoid undesired notes, it is possible to exclude some sounds, thus limiting the playable tone notes set to a few ones. Any of the twelve sounds and their corresponding doubled frequencies (octaves) can be activated/deactivated at any time. Consequently, in case the player wants to play a relative pentatonic scale with the base sound as matrix, he/she just needs to turn off the undesired notes as shown in Figure 8. The interface from the demo application developed for testing the framework shows at the top all enabled notes, from 1 to 12. In this case, for enabling/disabling any of them, the user should press any of the F_n keyboard keys, with n varying from 1 to 12. This feature also makes it possible to restrict the notes to a set of

TABLE I
RELATIONSHIP BETWEEN THE JUST INTONATION AND THE TWELVE-TONE EQUAL TEMPERED SCALE. NOTICE THAT THE DIFFERENCE BETWEEN THE CORRESPONDENT INCREMENTS IS NEVER LARGER THAN 1%.

Increments	Just Intonation	Equal Temp.	Difference
0	$1/1 = 1.000$	$2^{0/12} = 1.000$	0.0%
1	$16/15 = 1.067$	$2^{1/12} = 1.059$	0.7%
2	$9/8 = 1.125$	$2^{2/12} = 1.122$	0.2%
3	$6/5 = 1.200$	$2^{3/12} = 1.189$	0.9%
4	$5/4 = 1.250$	$2^{4/12} = 1.260$	0.8%
5	$4/3 = 1.333$	$2^{5/12} = 1.335$	0.1%
6	$7/5 = 1.400$	$2^{6/12} = 1.414$	1.0%
7	$3/2 = 1.500$	$2^{7/12} = 1.498$	0.1%
8	$8/5 = 1.600$	$2^{8/12} = 1.587$	0.8%
9	$5/3 = 1.667$	$2^{9/12} = 1.682$	0.9%
10	$9/5 = 1.800$	$2^{10/12} = 1.782$	1.0%
11	$15/8 = 1.875$	$2^{11/12} = 1.888$	0.7%
12	$2/1 = 2.000$	$2^{12/12} = 2.000$	0.0%

sounds necessary to play a specific song, as detailed later on the Case Study section. When a note is deactivated, the fret that would play its sound now plays the next activated one. Then all frets remain valid; what changes is the notes that are played.

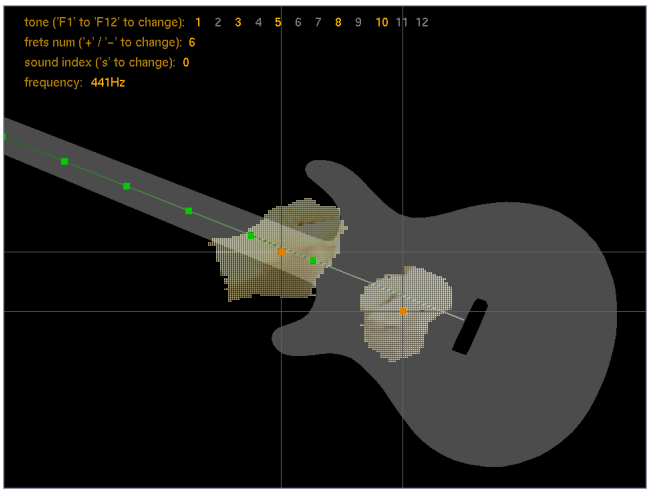

Fig. 8. Demo application interface. At the top of the image it is possible to see that the tone is limited to a pentatonic scale using the base sound as the first note. The current number of frets is six and can be set at any time during the playing. In the guitar, the frets division is represented by the green squares. Finally, the sound index represents the current sound archive that is being used as base sound and can also be dynamically altered.

Another implemented feature corresponds to the volume of the played sound according to the intensity of the input. The faster the right hand movement is, the higher will be the sound played. The envelope of the played sound (which determines the volume of the sound during a specific time) can also be altered by providing an increased sustain effect to the played note. A fret-based slide was also implemented, which makes the sound to change even in case the player does not move the right hand, by only sliding the left one after a played note, in a similar way to the slide technique on real guitars. Finally, more sound channels can be added in order to create more complex sound combinations, thus enhancing user's experience by simulating chords, for example. The Sound module, together with FMOD Ex, provides functions capable of managing all these parameters at any time, on the fly.

E. Render

This module is responsible for returning visual information to the player, helping him/her to achieve the application objectives. Although it is entirely possible to play without visual cues, interaction becomes more interesting with the viewing of the virtual guitar (and its frets divisions) according to the gloves' positions. Besides that, it can show other information such as text indications for activated/deactivated notes, current played frequency, among others, as shown in Figure 8.

The Render module also provides developers with some debugging functions, like rendering guitar points (both arm and body). This is useful for the application's test phase, since debugging is one of developers' most time consuming activities. This module uses OpenGL (and GLUT [22]) to perform such functions. If necessary, the render engine can be changed by constructing another render module (using OGRE3D [23], for example), similar to the one provided by the framework, without modifying the other framework modules.

IV. CASE STUDY

In this section will be described the case study, it is a simple example application using the framework. The user interface was a similar version of the interface shown in Figure 8. The framework evaluation process is described next.

A. Evaluation Methodology

In this work, the proposed framework was evaluated using a demo implementation of a simplified air guitar game. The tests were performed on an AMD Athlon X2 4800+ CPU, 1 GB of RAM equipped with an NVidia 8800 GTX GPU. The time necessary in order to correctly play a pre-defined sequence of notes was recorded. To assure the results were consistent, it was tried to expand the tested subjects to the highest number possible.

A total of 23 people were subject to the tests performed, varying on their age (from 17 to 47 years old) and gaming experience. After getting used to the demo application (playing freely during some minutes), they were asked to play a version of the introduction of the Smoke on the Water song (by Deep Purple), which is mainly composed by 4 different notes, as shown in Figure 9.

The virtual guitar was previously configured to allow this specific sequence to be performed. An electric guitar A3 played note with some distortion was used as base sound, being the first note of the introduction (differently from the B2 based power chord of the original version of the song). Only four different sounds were enabled, being the base sound (sound number 1) and the sounds 4, 6 and 7. The number of

frets were set to four, each one representing one of the enabled notes, by this way, facilitating the players task.

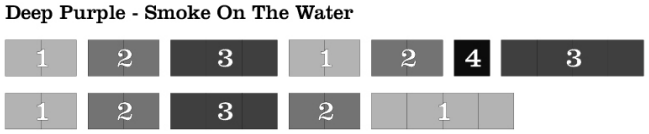

Fig. 9. Initial sequence from the "Smoke on the Water" song.

The test environment was organized as follows. An A4Tech webcam was placed at 1.30 meters above ground and about 1.5 meters in front of the user. Due to the camera characteristics, this placement showed to be a good choice for comfortably capturing the user's hands, allowing a good amplitude of movements. The demo visualization was projected on the wall, about 2.5 meters away from the spot where the subjects should stand. This should improve user's view and his/her immersion in the demo.

Preparation: the subjects put on the yellow rubber gloves and took place standing in front of the webcam. After that, they were told how to correctly position body and hands in order to enable the viewing of the entire virtual guitar, as well as their hands, on screen, as shown in Figure 10.

Familiarity with the application: before performing the test itself, the subjects had up to 3 minutes to understand the interface and check their performance by playing random notes.

Test: after understanding how the virtual guitar works, the subjects were presented to the sequence of notes that should be played (shown in Figure 9). They were told about the correspondence between the numbers written on the paper and the notes on the guitar, which varies according to the left hand's position. The total amount of time considered was the one spent performing correctly the entire sequence, from the moment the subject played the first note until the last correct one. In case of making some mistakes while playing the notes, the subject was told to restart the sequence from the beginning, without resetting or stopping the time count.

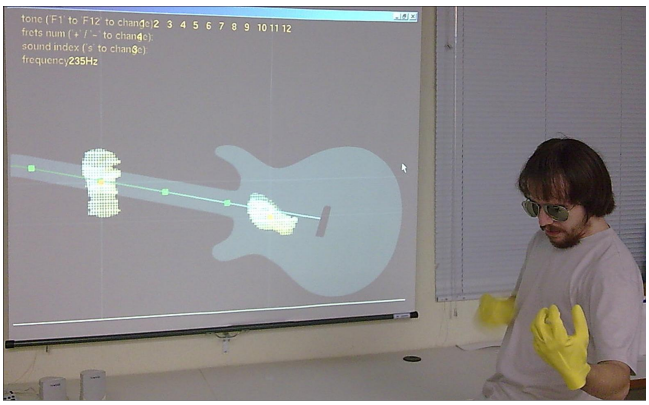

Fig. 10. Example application being controlled with two yellow gloves.

After performing the test, the subjects were oriented to answer a small free-form questionnaire about their experience with the application. The subjects were asked about difficulty in hands positioning, the application's visual and audio feedback and their impressions about the experience.

B. Results

The tests with the virtual guitar framework demo showed that the application is functional, intuitive and interesting. All 23 subjects submitted to the test were capable of performing the musical sequence correctly, without any problems. The mean execution time was 33.6 seconds, varying from 11 to 80 seconds.

Figure 11 shows the age of subjects and the time spent doing the test, indicating more density between the 20-30 year-old range, which is a potential target for a product using the framework technology. Most of the subjects in that age range spent between 11 and 41 seconds. This time was considered very satisfactory, since these people had never interacted with the application before.

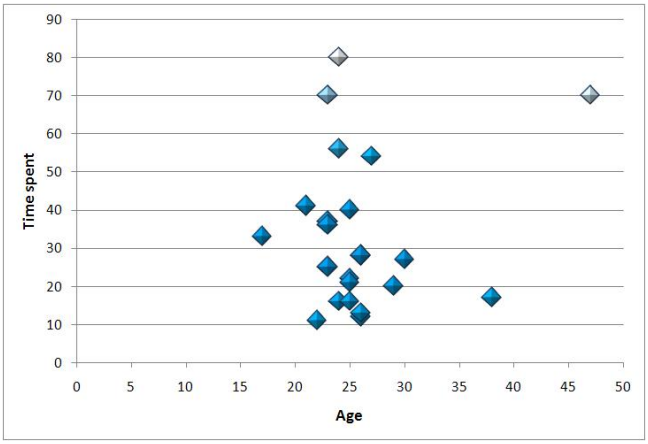

Fig. 11. Time spent × Age results.

Asked about the guitar feedback according to their hand interaction, 100% of the subjects said not to have felt any delay or misinterpretation of their movements. That real time feedback capacity is one of the framework's main attributes, due to the fact that any delay, even the smallest one, would compromise the usability of the virtual guitar, making hard for the user to move his/her hands spontaneously and intuitively.

Some of the subjects found difficulty while positioning their hands, in part due to the complete unknowing of the interface, in part due to the position of the webcam, which was fixed at the same height in all tests. Variations in height and scale of the subjects were not considered, and as a consequence some of them had to position their hands too high, too low, too open or too closed. However, most people managed to overcome this initial problem by finding an adequate way to position their hands in order to play the music sequence.

All tested subjects provided positive results regarding their experience. They seemed excited about the possibility of applying the framework capabilities into a real scenario.

Many of the answers suggested developing a game based on the demo application. Besides that, they pointed some improvements and adaptations, such as: adding more strings to the virtual guitar, in order to get a more realistically simulation; applying the technology to other instruments, like bass and drums; possibility of changing the guitar settings using the gloves themselves instead of the computer keyboard; implementation of other types of user interaction, like slide (by moving the left hand continuously along the frets, the sound would change without moving the right hand).

C. Lessons Learned

The tests described before were performed using a simplified set of the framework features. However, it proved to be possible to play entire songs, using all the frets and notes possibilities. The framework has several use possibilities, such as game development (as shown on the right of Figure 12), or an educational artifact, which could teach musical concepts, like notes, tones and rhythm to the users.

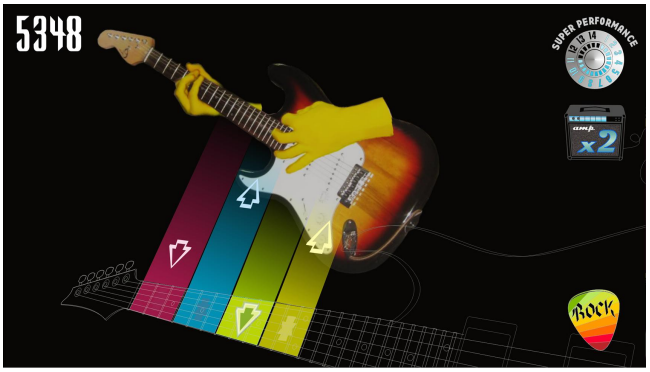

Fig. 12. Scratch of a game using the platform.

Due to its features, it was possible to notice that the framework is capable of providing both developers and users with all three requirements previously listed: real time interaction, actions mimicking a real guitar control and a realistic sound response. The satisfaction of tested subjects showed that performing a known type of interaction increases the application's playability. It is possible to have a simple but functional application only by using the framework, while complex functionalities could be added taking as base the demonstrated example application.

Taking a game as an application example, some features could be added to make it more interesting, such as: player's possibility to choose between several guitar models to play (as shown in Figure 13); monetary rewards system, in exchange for the correct execution of the songs; possibility to play new songs, progressively unlocked during the game; navigation through the game menus using only the hands wearing gloves, instead of a mouse or keyboard, using the same color-tracking technology applied in the guitar framework; support to guitar effects such as vibrato and slide (as shown in Figure 14 and Figure 15); more attractive design to the game interface, showing information of score and performance of the player (besides the sequence to be played, of course).

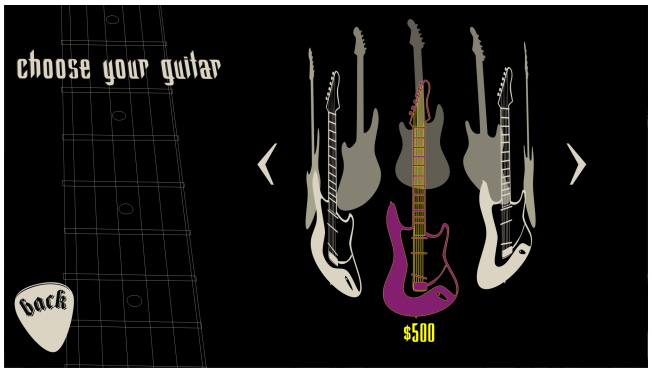

Fig. 13. "Choose your guitar" screen.

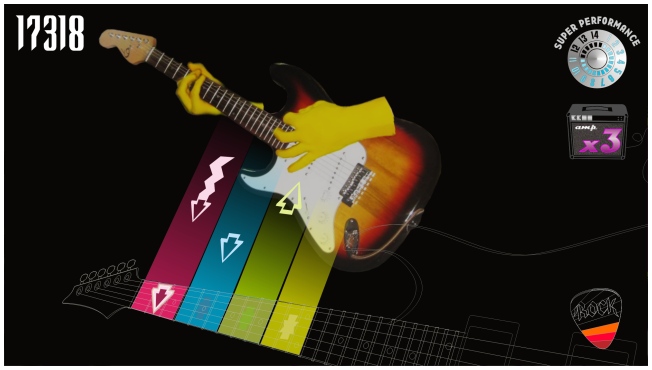

Fig. 14. Gameplay screen showing a "vibrato" indication.

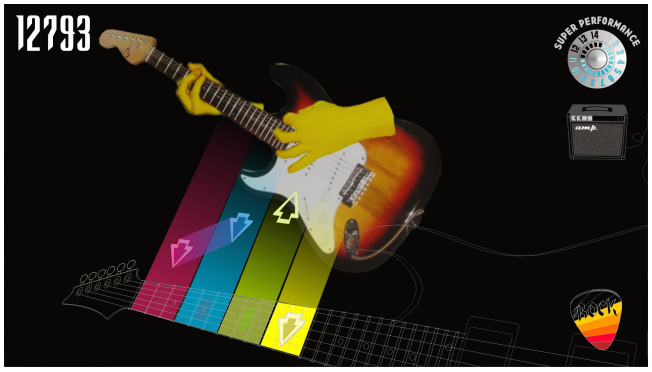

Fig. 15. Gameplay screen showing a "slide" indication.

V. Conclusion

In this work, an open-source framework for air guitar based applications has been detailed, showing that with a pair of colored gloves and a webcam it is possible to recognize user's intention of playing a virtual guitar. It was also described a case study, where an example application was used to validate the framework. This application was tested and the results

showed to be successful: all tested subjects could reach the objective of playing a simple song during a small amount of time and the most important, they were satisfied with the experience.

As future work, a number of new features can be added to the framework. Some effects such as guitar bend and vibrato simulation can be implemented to make the player interaction more interesting. A multi-player mode should also be possibly present in the next versions, with the only restriction that the players must use gloves with different colors.

The air guitar framework is available for download at http://sourceforge.net/projects/airguitarframew.

REFERENCES

[1] S. C. Entertainment, "Ps3 eyetoy," http://en.wikipedia.org/wiki/PlayStation_Eye, October 2007.

[2] M. Byrne, "Console players win on their points," http://www.fyrne.com/article_pages/console_players_win.html, 2004.

[3] Nintendo, "Wii remote controller," http://en.wikipedia.org/wiki/Wii_Remote, November 2006.

[4] Microsoft, "Xbox360 project natal," http://www.xbox.com/en-US/live/projectnatal/, 2009.

[5] D. Wixon, "Guitar hero: the inspirational story of an "overnight" success," *interactions*, vol. 14, no. 3, pp. 16–17, 2007.

[6] A. Games, "Adagio," http://www.newgrounds.com/portal/view/388085, 2007.

[7] K. D. E. Konami, "Guitarfreaks," http://en.wikipedia.org/wiki/GuitarFreaks, 1998.

[8] H. M. Systems, "Frequency," http://www.harmonixmusic.com/#games, November 2001.

[9] X. H. I. Europe, "Dj hero coming this year," http://xbox.hdtvinfo.eu/component/content/article/21-dj-hero-coming-this-year.html, March 2009.

[10] U. Voodoo, "Frets on fire," http://fretsonfire.sourceforge.net/, August 2006.

[11] A. RedOctane, "Guitar hero," http://hub.guitarhero.com/index_uk.html, November 2005.

[12] E. A. MTV Games, "Rock band," http://www.rockband.com/, November 2007.

[13] A. Camurri, M. Ricchetti, and R. Trocca, "Eyesweb - toward gesture and affect recognition in dance/music interactive systems," in *ICMCS '99: Proceedings of the IEEE International Conference on Multimedia Computing and Systems*. Washington, DC, USA: IEEE Computer Society, 1999, p. 9643.

[14] T. Mäki-Patola, J. Laitinen, A. Kanerva, and T. Takala, "Experiments with virtual reality instruments," in *NIME '05: Proceedings of the 2005 conference on New interfaces for musical expression*. Singapore, Singapore: National University of Singapore, 2005, pp. 11–16.

[15] T. K. A. H. A. Karjalainen, Matti; Mki-patola, "Experiments with virtual reality instruments," in *JAES Volume 54*. San Francisco, USA: Audio Engineering Society, 2006, pp. 964–980.

[16] J. Pakarinen, T. Puputti, and V. Välimäki, "Virtual slide guitar," *Comput. Music J.*, vol. 32, no. 3, pp. 42–54, 2008.

[17] D. Crane, *To Air is Human: One Man's Quest to Become the World's Greatest Air Guitarist*. Riverhead Trade, 2006.

[18] T. Pintaric, "Dsvideolib," http://www.ims.tuwien.ac.at/ thomas/dsvideolib.php, October 2005.

[19] F. Technologies, "Fmod," http://www.fmod.org/, December 2005.

[20] S. Graphics, "Open graphics library (opengl)," http://www.opengl.org/, 1992.

[21] K. Suzuki, I. Horiba, and N. Sugie, "Linear-time connected-component labeling based on sequential local operations," *Comput. Vis. Image Underst.*, vol. 89, no. 1, pp. 1–23, 2003.

[22] S. Graphics, "The opengl utility toolkit (glut)," http://www.opengl.org/resources/libraries/glut/.

[23] T. O. Team, "Object-oriented graphics rendering engine (ogre3d)," http://www.ogre3d.org/, March 2005.