An Open Source Framework to Manage Kinect on the Web

Francisco Moreno, Esmitt Ramírez, Francisco Sans and Rhadamés Carmona
Computer Graphics Center, Computer School
Faculty of Sciences, Central University of Venezuela
Caracas, Venezuela, 1040
email: francisco.moreno@ciens.ucv.ve, esmitt.ramirez@ciens.ucv.ve,
francisco.sans@ciens.ucv.ve, rhadames.carmona@ciens.ucv.ve

***Abstract*—Human-computer interaction has had continuous changes in recent year, with a significant improvement in touch screens and motion sensors. New sensing technologies, like Microsoft Kinect, provide a low-cost way to add interactivity with gestures and postures. The current tendency is to develop software for the web. Interactions through the Kinect can be an additional benefit for these applications. While there is a solution to use the Kinect in the web, it is only supported on Windows platform by using Internet Explorer browser. In this paper, we propose a robust, interoperable, elegant and efficient server-client open source framework which allows interacting with the Kinect or similar capture device from browsers. The tests support our hypothesis, resulting in a low consumption of memory/time in different modern browsers. Also, an experimental test was performed to prove its usefulness, getting a rate of 30 fps successfully.**

Keywords*—kinect, open source, framework, image processing, interoperability.

I. Introduction

In recent years, several novel approaches to human-computer interaction (HCI) became popular for their wide spread accessibility and low costs. Devices based on touch screens for mobiles, large displays, motion sensors, and others are increasing their presence in the modern world. Motion sensors as Microsoft Kinect, Nintendo Wii and Sony PlayStation Move were originally developed to be accessories in video game consoles, but used as low-cost HCI devices in many research areas.

The Kinect sensor was originally intended to be a motion-sensing input device for the XBox 360, allowing the user to control games via gestures and spoken commands [1]. In this way, the Microsoft Kinect sensor allows natural gestures/postures without touching a game controller. Also, it offers functions as 3D scanner capable of capturing a color image, infrared, depth map, and skeleton-articulated structure of a user, see Fig. 1.

It is possible to use the Kinect to recognize gestures and skeleton tracking to control assorted control applications. All these features increase popularity of the Kinect in several applications such as touch screens, full body 3D scanners, face tracking, and other applications that changes the original use of the kinect [3].

Since the last decade, the current development is to create software for the web [4], and Kinect-based applications are not the exception. For instance, the ability to use Kinect data within a classic web application. Moreover, they involve new kinds of interactions on traditional websites and new kind of web applications, which were until now, exclusively, standalone. There is an official version to use the Kinect on the web [5], but it is limited to Microsoft Internet Explorer browser over a Windows operating system.

Fig. 1: Kinect allows capture data as a webcam, depth map and skeleton info of users in front of it. Image taken from Kinect for Windows website [2].

Due to this limitation, it is important to have an open source tool to capture and handle features offered by the Kinect, supporting most of the popular browsers for developing different applications. In this paper, we present an open source framework which offers an effective and robust server-client approach to manage a pipeline for data acquisition and rendering using the Kinect. This approach is able to work in any modern HTML5-support browser.

We can summarize our contribution as described below:

- Usage of Microsoft Kinect as an acquisition device without depending of a proprietary SDK, being platform independent.
- A robust client-server approach based on WebSocket to develop a base for future applications.
- Implement a standard format to exchange and display data of color image, depth map and skeleton.

This paper is organized as follows: section II presents the main previous research, which represents the basis of our study. Section III exposes the design and development of our tool, describing each module and data structures used. Section

978-1-4673-9143-6/15/$31.00

IV explains the experimental tests and shows the obtained results. Finally, section V presents the conclusions and future work.

II. Related Work

The Microsoft SDK Kinect has fully support on the desktop, where several researches have been using the Kinect as a lower-cost acquisition device to develop specialized applications [6]–[8], serious games [9]–[12], and social areas such as cognitive studies of groups [13]–[15], in culinary arts [16] or inclusive in dancing [17].

To clarify, all access to the Kinect via browser requires the user to install some kind of service in client machine. This service would then expose itself via the TCP networking stack (e.g. Flash or Silverlight) so a browser plug-in can call back to it on the localhost. In the same way, it is possible to use HTML5 to expose this service using WebSockets or an AJAX-style service returning JSON data (i.e. JSON is an open standard format to transmit data objects consisting of attribute-value pairs).

Our research shows us that there is a solution based in a JavaScript API provided by Microsoft Research [5] to develop an application using the Kinect in a web browser. Also, there are other solutions provided by the community of developers [18], [19] which use the proprietary SDK of Microsoft. These APIs allow the access to the data provided by the Kinect (i.e. color and depth images) and DOM events for joint activity. The API of Microsoft Research is an ActiveX plugin which runs in a browser. It requires the Microsoft Kinect drivers to be installed on the machine. This creates a limitation, since the operating system must be Windows with recent versions of IE (i.e. IE9 or newer).

Furthermore, there are applications which allow web browsers to have full control on the Kinect as gesture capture device. This is achieved using a second on-running application as shown in [20]. A remarkable research in this area is the work presented in [21], which consist in 2 subprojects: server-side and client-side. However, this solution employs the original SDK which makes the solution a C# application. Other applications which use the proprietary SDK can be found in [22] and [23].

There are solutions based on the OpenNI library [24], which find the interoperability of natural user interfaces for Natural Interaction (NI) devices. This library operates as a middleware between applications, for instance DepthJS [20] (which only works for Google Chrome). Applications to control drones [25], or web server that provides an HTTP interface for the Kinect [26], or to capture the skeleton [27], [28], are also solutions implemented using this open source approach.

A few commercial solutions employ the Kinect on any web browser to develop applications. An outstanding example is ZDK [29] which allows to access to the color image, depth map and skeleton data from the Kinect with a high-level abstraction for the components using Unity.

In this work, we introduce an interoperable open source framework to use the Kinect as capture device from popular web browsers in order to set up a new kind of applications.

III. Our Approach

In this section, we discuss about the proposed framework. First, we describe a global overview of the solution; then, each module is detailed in further subsections: image acquisition, service processing, networking, browser processing and, display.

A. Architecture

We implemented a framework to capture Kinect's data and display it on different web browsers. It is based on a client-server approach. Fig. 2 shows the structural architecture of our proposal.

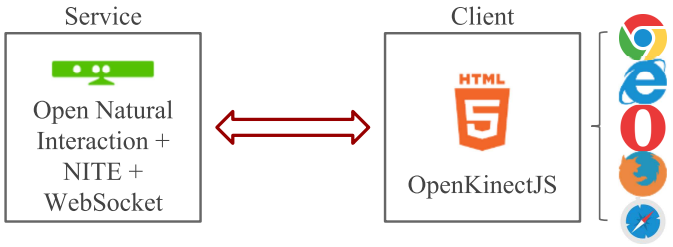


Fig. 2: Basic architecture of our approach showing the Service and Client functional structures.

Basically, our solution can be divided in two sections: A server-side application which uses OpenNI, NiTE and WebSocket, and a client-side application using JavaScript on a HTML5 canvas. Notice that the Kinect device is located in the server-side. However, instead of using the Microsoft SDK for Kinect, we use the drivers offered by the library OpenNI to control the device hardware. Also, we use the library NiTE for the user recognition, and WebSockets to achieve real-time communication with the client. The server-side is implemented as a Service. Thus, the client can request Kinect data to the server, and perform some local processing to manage such a data in the browser.

The client-side runs over a browser with HTML5 support, such as Google Chrome [30], Microsoft Internet Explorer [31], Opera [32], Mozilla Firefox [33] and Apple Safari [34]. The client-side is the responsible to do the required processing. We implemented the basic functionality, which includes: showing the color image, depth map, and skeleton.

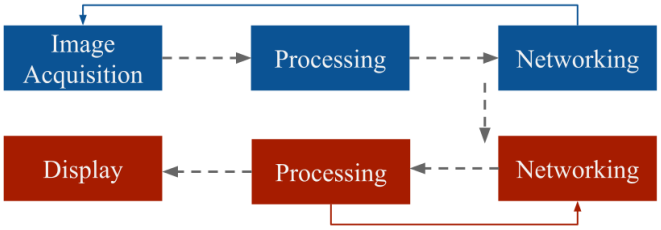


Fig. 3: Modules of our solution. Blue modules are located in server-side and red modules in client-side.

The solution is composed by a set of modules which are distributed in server and client side. The Fig. 3 represents an overview of the available modules. The upper modules

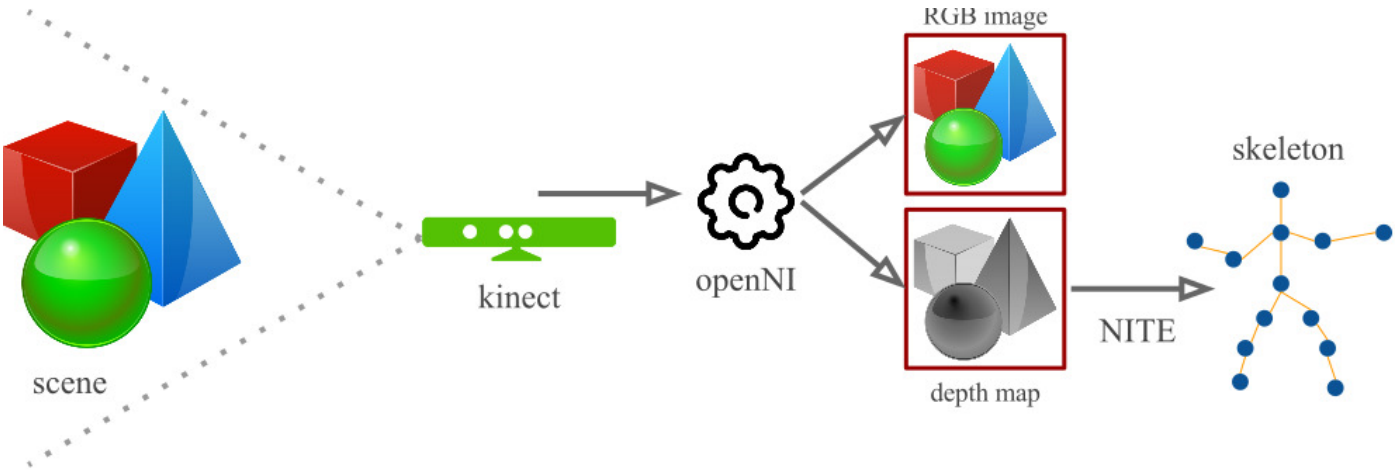


Fig. 4: Scheme of the image acquisition stage in our framework.

(blue rectangles) are located in the server-side, and the lower modules (red rectangles) are located in the client-side.

The workflow starts in the Image Acquisition module. It receives the data captured from the Kinect device. The Processing module (server-side) validates the received data. Then, the Networking module is responsible to send and receive information. Notice that networking module is presented in both sides: server and client. Once data is received by the client-side, the Processing module (client-side) is able to require more data from the server-side, while the Display module is used to show the visual results.

The following subsections explain more deeply each implemented module in our solution.

B. Image Acquisition

The first stage in the pipeline consists in the acquisition of the image, captured by a sensor device (i.e. Kinect). This module is server-side which has access to basic built-in features of hardware such as color image (in RGB pixel format) and depth map. The Microsoft Kinect has other relevant features: information of microphones array or a quad rotor to move the angle of cameras up and down. The Kinect SDK provided by Microsoft handles all these device features. However, this SDK is not an open source solution.

Our approach supports any sensor acquisition device with the basic features used in this research: color image, depth map, and skeleton. This process is achieved using the OpenNI library, which is used, specifically for this work, to control the Kinect.

The process of image acquisition is shown in Fig. 4, where the Kinect captures the 3D scene to be process for OpenNI library. After, a RGB image and a depth map are generated. From the depth map, the skeleton of the user is computed (using NiTE library).

NiTE processes the information provided by OpenNI, in such a way that it is able to identify and discriminate common objects within the field of view of the device (e.g. humans), allowing a coarse detection of up to seven people in the scene and a finer detection of up to two users within the scene.

One of the users is selected as the direct/main participant to interact with the application. Using NiTE, the position of each joint can be detected, obtaining a virtual user skeleton located within the 3D space. The skeleton provides a set of joints located over the skeleton in strategic places. For each individual joint, the reliability of its location can be expressed according to three possible states: captured, inferred and/or unknown.

The data acquisition is only performed when the client requests such a data. We set the resolution of the images to 640×480 pixels, representing the maximum resolution supported by the Kinect with a rate of 30 frames per seconds (i.e. 30 fps).

C. Service Processing

The acquired data has to be validated before been processed for delivery via the communication channels. If any data value is invalid, a string is generated in JSON format containing an error code and a description associated with it. The purpose is to provide the client a feedback when his request cannot be processed.

As an example, the process that involves the capture of the user skeleton can fail if there is not a user to track or if the user has not been tracked correctly. In this case the service processing module detects this situation and notifies it to the client through an error message.

Notice that when a user is not tracked, it is consider as an error. In many other applications, this is treated only as a warning message or maybe another non-severe notification.

D. Networking

As mention before, the networking stage is placed in both server an client side. Then, before sending the information through the communication channels, a set of steps must be performed by the server:

1) Establish a successful connection with the client.
2) Check for any error in the data to notify to the client.
3) Prepare the data to be sent.

4) Calculate the data size.
5) Send the data through the WebSocket.

The data preparation varies depending on the data to send. For the skeleton, the coordinates (x, y, z) of each joint may be mapped (if required) into image space of the depth map. NiTE provides a set of instructions to perform such a mapping.

With the information of the 2D or 3D points, a string is generated in JSON format containing all skeletal joints and their positions. This string is then transformed into an array of bytes with UTF8 (8 bits Unicode Transformation Format) coding.

For the depth map, each of the depth distances (a 16 bits integer) is divided in two bytes. These distances are successively stored within an array of bytes. The size in bytes of this array is twice the resolution of the depth map.

For the color image, the color components (R, G, B) of the pixels are stored consecutively within an array of bytes. Because the colors are already represented in 8 bits, no extra processing is required. The array size is three times the resolution of the color image. Fig. 5 shows an example of this image decomposition.

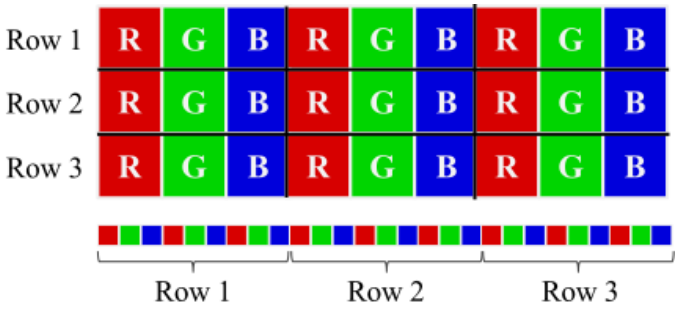


Fig. 5: Structure of the RGB image as a sequence of bytes to be store.

E. Browser Processing

When the client receives the data, the first step is to verify if the received information corresponds to an error message or not. If it does not correspond with an error, we check if the data is a valid JSON format or a binary format.

If the information received is in JSON format, we check if it corresponds with the skeleton information in 2D or 3D space. Then, the information is stored into a JavaScript object. In each case, a callback function is invoked in order to process the specific skeleton. The developer can optionally set a user-defined callback to process the skeleton. In any case, the user may indicate if the skeleton must be displayed or not. In our framework, we provide a function to display the skeleton into the 2D canvas.

If the information comes in binary form, its size must be evaluated. If the size of the coming information doubles the known size of the expected image, it is assumed that the image corresponds to a depth map. In this case, the received information is stored into an array of integers of 16 bits in JavaScript. As in the case of the skeleton, the user may define a callback function to be invoked, and the depth map may be automatically displayed if required.

If the size of the received information is three times the size of the expected image, it would correspond to a color image. This image is stored into an array of integers of 8 bits, where 3 consecutive bytes are the RGB channels of a pixel. The programmer may define a callback function to manage this image, and decide if the color image must be displayed or not.

When the received data has been processed, the library automatically sends to the server a request for a new frame of data, according to the last set of received data.

F. Display

The process of data rendering can be requested by the user in a specific time, or it can be activated automatically. This process will work differently depending of the kind of information requested to render (i.e. color, depth or skeleton).

If the information that is going to be rendered is the color image, only the byte array is given to the HTML5 canvas. In the case of the depth map, this must be normalized. Each distance of the image is transformed to the space $[0, 1]$ first, dividing each value within the maximum possible distance. With this new data, a new color array is created with 3 channels for every color. Each one will have the normalized distance multiplied by the maximum value in the RGB color space, the 255. This creates a grey scale image, where the darkest color indicates the nearest distance from the device, and the brightest the farthest.

To render the skeleton, the structure describing the skeleton must be transverse, drawing a fixed size circle in the position of each joint. However, the drawing of the joint can be selected by the user. For our approach, we used a simple circle, but it represents the center of a drawing structure.

An example of these three kind of rendering is observed in Fig. 6, where a color image (see Fig. 6(a)), a depth map (see Fig. 6(b)), and a skeleton are shown; red dots represent the joints while blue lines connect the dots to illustrate the figure (see Fig. 6(c)).

It is important to note that the different rendering processes described above, were designed to work in the HTML5 canvas of modern browsers. Thus, it is possible to show a combination of data such as color + skeleton or depth + skeleton in a same image. Fig. 7 shows an example of this characteristic implemented in our framework.

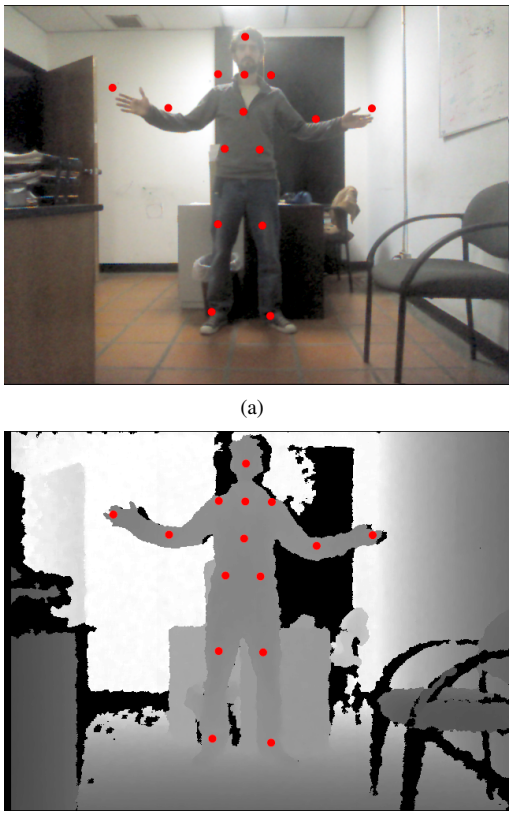
We have explained all the modules for our open source framework to manage the Kinect, from the image acquisition to the display modules. Now, the results of our research will be presented.

IV. Tests and Results

To test the effectiveness of our solution, we performed a set of experiments. These tests are related to the interoperability between different browsers, time and memory consumption. They are performed in two main operating systems: Windows 7 and Ubuntu. Also, we checked device interoperability, using

(a) (b) (c)

Fig. 6: Illustration of the application showing: (a) only the color image, (b) only the depth map, and (c) only the with joints.



(a) (b)

Fig. 7: A rendering example using our framework where it shows (a) the color image and the skeleton, (b) the depth map and the skeleton.

different devices at the same time. Finally, the tool was tested in a particle system application.

The initial configuration of the Kinect resolution was 640×480 pixels for the RGB color image and depth maps. We use this resolution instead of 1280×960 pixels, because with the latter resolution the hardware performance is lower than 30 fps and it does not reach a real time performance.

All tests were executed on a PC with Windows 7 (64 bits) and Ubuntu 14.04 LTS (64bits), with an Intel(R) Core(TM) i7-3770 CPU of 3.40 GHz, and 8.00 GB of RAM. Additionally, the Kinect version used was the Kinect 1 for PC instead of the Kinect One to capture the performance in less capable devices.

A. *Windows*

To test on the Windows 7 PC operating system the environment was configured with the Microsoft SDK for Kinect, OpenNI 2.0 and NiTE 2.0. We used the proper Windows version of these software.

1) Browser Interoperability: A JavaScript library called stat.js [35] was used as a benchmark tool to measure the time in milliseconds between a client request, and the reception of the data from the server in the client-side. The server can only produce new information at 30 frames per seconds, but the client can request more frames.

In the case of Windows 7, the browsers used for the test were: Google Chrome version 42.0.23, Internet Explorer version 11, Opera version 29.0, and Mozilla Firefox version 37.0.1. All these versions were the latest to June 2015. Table I summarize the time of information retrieval from the server using different browser.

TABLE I: Time in milliseconds for information retrieval from the server by different browser in Windows.

	Browser			
Information Retrieved	Chrome	Explorer	Opera	Firefox
Color Image	30.00	32.00	31.00	34.00
Depth Map	33.00	32.00	33.00	30.00
Skeleton	30.00	33.00	32.00	31.00
Color + Depth	67.00	65.00	68.50	67.50
Color + Skeleton	65.50	65.50	65.00	67.00
Depth + Skeleton	66.00	68.00	64.50	66.50
All Buffers	99.67	99.33	99.67	99.00

The Table I shows the time between the client request and the data reception, on different browsers. As can be observed in the table, if the client requests one buffer of information (color image, depth map, or skeleton), the server lasts in average 32 milliseconds to send the requested information.

If the client requests a pair of buffers (color image and depth map, color image and skeleton, or depth map and skeleton), the server takes in average twice the time than with one buffer. This is expected, as the server receives the petitions

of the client one by one, stores them in a queue, and resolves the petitions one by one.

As the Kinect can only generate 30 fps, half of the frames resolve a petition of a specific buffer and the other half of the frames resolves the other petition. The same behavior is observed in the case of requesting three buffers. In this case, the server takes three times longer than with a single buffer to send the requested information.

Nevertheless, the tool allows the client to display every time a piece of information is received, even though all not information is available. In this case, the display is generated with the new available information, and with the previous frame information for those buffers needed. That allows the application to run at interactive times.

2) Memory Consumption: The Windows task manager tool was used for this test to measure the total RAM that the application consumes. The server memory occupancy is independent of the browser used and the number of buffers required by the client, occupying around 221 Mb of memory on average.

In the case of the client-side, the Table II shows the average of the memory occupancy for different browsers in Kb. We subtracted the memory consumption of the browser running the application, versus the memory consumption of the browser with a blank page.

TABLE II: Memory Occupancy of the application in the client browsers in Windows.

	Memory occupancy (Kb)
Chrome	171,063
Explorer	72,739
Opera	185,530
Firefox	266,568

As can be observed, Explorer has the lowest memory consumption, and Firefox has the highest memory consumption for all the browsers. It is interesting to notice how different this measure is for every browser, remarking the difference of implementation of each of them.

B. Ubuntu

In the case of Ubuntu, the Microsoft SDK for Kinect cannot be used. For the Kinect driver, the OpenKinect was necessary. To retrieve the information the OpenNI 2.2 and NiTE 2.0.0 was required. Again, we used the proper Linux-based machines version of these software.

1) Browser Interoperability: The browsers used for the test were: Google Chrome version 42.0.23, Opera version 31.0, and Mozilla Firefox version 31.0. Internet Explorer was not taken into account because there is not an official version available for Ubuntu.

For this test, we used the same benchmark tool and test methodology as in the case of the Windows operating system. We present in Table III the time results between the client request and the data reception from the server using different browser in Ubuntu.

TABLE III: Time in milliseconds for information retrieval from the server from different browser in Ubuntu.

	Browser		
Information Retrieved	Chrome	Opera	Firefox
Color Image	31.00	33.00	33.00
Depth Map	31.00	33.00	31.00
Skeleton	32.00	30.00	32.00
Color + Depth	65.00	66.00	66.00
Color + Skeleton	65.00	66.00	68.00
Depth + Skeleton	68.00	64.00	67.00
All Buffers	99.00	100.67	108.00

As shown in the previous table, the behavior is as expected: very similar to the Windows version. As before, the request of a pair of buffers takes twice the time than a request for a single buffer, working each petition from the client once at a time.

It is noteworthy that both obtained times (in Windows and Ubuntu) are the average of several execution of the benchmark. The results were taken from an average of 50 executions of each test.

2) Memory Consumption: In the case of Ubuntu, we used the Gnome System Monitor to measure the total consumption of memory RAM. The server occupies 230 Mb to allocate our solution.

The memory consumption of the client-side is shown in Table IV, where is the values in Kb for Google Chrome, Opera and Firefox browser clients. Notice that the versions of the browser are different than the ones used in the Windows test. However, that aspect does not cause a strong impact in the results.

TABLE IV: Memory Occupancy of the application in the client browsers in Ubuntu.

	Memory occupancy (Kb)
Chrome	55,800
Opera	146,000
Firefox	240,300

As can be observed, Firefox was the browser with the most memory consumption and Chrome was the browser with less consumption. The behavior is almost the same as Windows in this test. Nevertheless, this test is not conclusive as all browsers have different implementations and ways to manage memory in each operating system.

Another aspect to remark is the better memory management of the browser tested in Linux against Windows. In all cases, the consumption is less (excluding Internet Explorer in the Windows case). However, this characteristic is not directly related with our solution but it is important to consider when selecting a base operating system to develop web-applications.

C. Device Interoperability

The application was tested in different devices at the same time, connecting several devices with the same remote server. Fig. 8 shows a PC displaying the color image and the skeleton (top-center), a cellphone displaying only the depth map

(bottom-left) , and another cellphone (bottom-right) displaying only the color image. As long as a device has an Internet connection and a browser with HTML5, it can connect to the server.

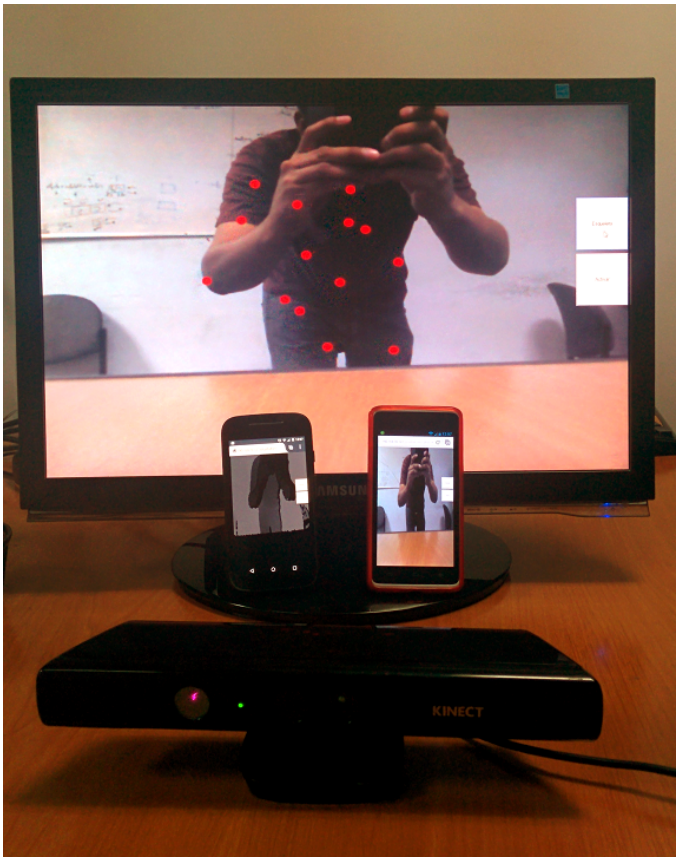


Fig. 8: An example of three devices connected to the same server showing the skeleton points, the depth map and the color image.

D. Integrating with an Application

An application using WebGL[1] was developed to test the complexity of integrating our tool with a basic program. The implemented program is a set of particles systems that are controlled with the Kinect. The main idea is to measure the performance and ease of integration of our framework with any public library available on Internet.

Firstly, notice that WebGL (Web Graphics Library) is a JavaScript API for rendering interactive 3D computer graphics within any compatible web browser without the usage of plug-ins. It is maintained by the Khronos Group, which also maintain the OpenGL graphics library. It is integrated with the standards of the browsers, allowing the use of the GPU with JavaScript code and shaders. With this graphics library and the information provided with our tool, Kinect interactivity can be added to several type of graphic applications.

To develop the application, the WebGL based engine BabylonJS [36] was used. Nowadays, BabylonJS is a widely known Javascript library to easily render 3D primitives in a browser. With this engine, a scene can be established with 15 basic particle system (i.e. associated to the joint in the skeleton). In this way, the particle system can be displayed without lacks at a frame rate of 60 fps.

Then, the interaction with Kinect was added, in which one particle system is mapped to each of the skeleton joints provided by our tool, as is presented in Fig. 9. Notice in the figure that the test created is interactive and captures several movements in real time (from left to right): standing with legs apart (see Fig. 9(a)), with hands on hips (see Fig. 9(b)), and standing on one foot (see Fig. 9(c)).

The particle system was chosen to exploit the high computing on displaying over a browser. For this study case, our open source solution shows a frame rate at 60 fps; keeping the processing overhead generated by the Kinect as a non-significant impact over an application.

It is important to keep in mind that the user skeleton is refreshing at a rate of 30 fps, which means that half of the rendered frames are using updated skeleton information. This outcome is expected because the web application renders the last information received by the client, even though a new request might have been sent by the client.

V. Conclusion and Future Work

This paper introduces an open source framework using open source libraries to use the Kinect as acquisition device to manage the data such as color image, depth map and skeleton. The framework tackles the limitation of using standalone applications when a proprietary SDK is used for the Kinect. It offers a server-client approach to develop applications on it.

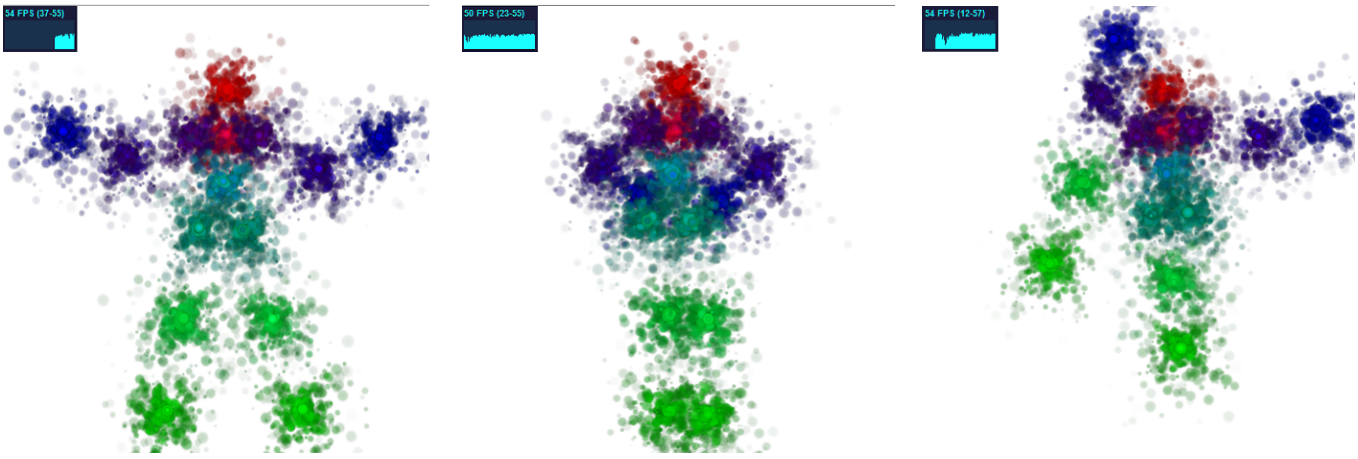
This work opens several possibilities regarding Kinect and other acquisition devices usage. We have tested the framework in a PC with Windows and Ubuntu, using different browsers. About 30 frames per second can be delivered to the client-side, including color image, depth map and skeleton. This frame rate is limited by the Kinect rate, which is also 30 fps.

Our proposal is focuses in the development in web applications which use the Kinect as acquisition device. Also, as we stated, its usage over different devices that have a modern browser (i.e. with support for HTML5) might be considered for applications in specialized conferences such as medical teleconferences with patients, school dancing sessions, and others real-time software.

Similarly, our tests proved its ease integration on existing libraries available using Javascript. In our case study, we labour our framework with a 3D particle system which demands graphic resources in the browser with ease. This test demonstrates that our solution does not affect drastically a real-time application on the web. We hypothesized that our solution might occupy an aspect to be considered in traffic/network high demanding applications in a client-server scheme.

For future work, we propose tests in other devices in order to get more information about the performance of our framework. Also, incorporate different networking scenarios to measure the performance of the framework when the server and client are in different physical machines. Furthermore, the

[1] https://www.khronos.org/webgl/



(a) (b) (c)

Fig. 9: Particle system developed using BabylonJS together with our solution capturing points using the Kinect in different positions: (a) standing with leg apart, (b) with hands on hips and, (c) standing on one foot.

server responses can be improved by stacking different buffer solicitations from the same client, and responding to them with only one call to the Kinect. This will allow a quicker receive of information by the client.

ACKNOWLEDGMENT

The authors would like to thank reviewers for their insightful comments on the paper, as these comments allow us to improve this work.

REFERENCES

[1] A. Davinson, *Kinect Open Source Programming Secrets: Hacking the Kinect with OpenNI, NITE, and Java*, 1st ed. McGraw-Hill Education TAB, 2012.

[2] Microsoft, "Kinect for Windows," http://goo.gl/imiEIK, 2013.

[3] J. St. Jean, *Kinect Hacks: Tips and Tools for Motion and Pattern Detection*, 1st ed. O'Reilly Media, 2012.

[4] T. O'Reilly, "What is web 2.0 - design patterns and business models for the next generation of software," September 2005, [Online; 30-September-2005]. [Online]. Available: http://www.oreilly.com/pub/a/web2/archive/what-is-web-20.html

[5] Microsoft Research, "Kinect in the Browser," http://goo.gl/sGW7E1, 2013.

[6] N. Villaroman, D. Rowe, and B. Swan, "Teaching Natural User Interaction Using OpenNI and the Microsoft Kinect Sensor," in *Proceedings of the 2011 Conference on Information Technology Education*, ser. SIGITE '11. ACM, 2011, pp. 227–232.

[7] E. Stone and M. Skubic, "Evaluation of an inexpensive depth camera for passive in-home fall risk assessment," in *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2011 5th International Conference on*, May 2011, pp. 71–77.

[8] Y. Cui and D. Stricker, "3D Shape Scanning with a Kinect," in *ACM SIGGRAPH 2011 Posters*, ser. SIGGRAPH '11. New York, NY, USA: ACM, 2011, pp. 57:1–57:1.

[9] S. Saha, M. Pal, A. Konar, and R. Janarthanan, "Neural network based gesture recognition for elderly health care using kinect sensor," in *Proceedings of the 4th International Conference on Swarm, Evolutionary, and Memetic Computing - Volume 8298*, ser. SEMCCO 2013. New York, NY, USA: Springer-Verlag New York, Inc., 2013, pp. 376–386.

[10] F. Moreno, J. Ojeda, E. Ramírez, C. Mena, O. Rodríguez, J. Rangel, and S. Álvarez, "Un framework para la rehabilitación física en miembros superiores con realidad virtual," in *Proceedings of the I Congreso Nacional de Computación, Informática y Sistemas (CoNCISa 2013)*, 2013, pp. 77–84.

[11] J. E. Muñoz, R. Chavarriaga, and D. S. Lopez, "Application of hybrid bci and exergames for balance rehabilitation after stroke," in *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*, ser. ACE '14. ACM, 2014, pp. 67:1–67:4.

[12] J. Ojeda, F. Moreno, E. Ramírez, and O. Rodríguez, "Gesture-gross recognition of upper limbs to physical rehabilitation," in *Proceedings of the International Congress of Numerical Methods in Enginnering and Applied Sciences (CIMENICS))*, 2014, pp. PI 7–12.

[13] A. L. S. Kawamoto and F. S. C. da Silva, "Using low-cost technologies in the development of people-monitoring applications," in *Proceedings of the 2013 XV Symposium on Virtual and Augmented Reality*, ser. SVR '13. IEEE Computer Society, pp. 204–207.

[14] R. Harper and H. Mentis, "The mocking gaze: The social organization of kinect use," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, ser. CSCW '13. ACM, 2013, pp. 167–180.

[15] B. Nansen, F. Vetere, T. Robertson, J. Downs, M. Brereton, and J. Durick, "Reciprocal habituation: A study of older people and the kinect," *ACM Trans. Comput.-Hum. Interact.*, vol. 21, no. 3, pp. 18:1–18:20, Jun. 2014.

[16] G. Panger, "Kinect in the kitchen: Testing depth camera interactions in practical home environments," in *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '12. ACM, 2012, pp. 1985–1990.

[17] Z. Marquardt, J. a. Beira, N. Em, I. Paiva, and S. Kox, "Super mirror: A kinect interface for ballet dancers," in *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '12. ACM, 2012, pp. 1619–1624.

[18] T. Anderson, "KinectServer," https://kinectserver.codeplex.com/, 2011.

[19] P. Kalogiros. (2012) KinectJS. http://kinect.childnodes.com/.

[20] MIT Media Lab. (2013, April) Depthjs. http://depthjs.media.mit.edu/.

[21] V. Pterneas. (2013, December) Kinect and HTML5 using WebSockets and Canvas. http://goo.gl/KLFwuD.

[22] Microsoft Open Technologies, Inc., "Kinect Common Bridge," https://github.com/MSOpenTech/KinectCommonBridge, 2014.

[23] W. Verweirder, R. Gerbasi, and J. Imhof. (2014) AIRKinect Extension. http://as3nui.github.io/airkinect-2-core/.

[24] Occipital, Inc., "OpenNI," http://structure.io/openni, 2015.

[25] P. Teixeira, "Kinect in the Browser using Node.js," http://metaduck.com/09-kinect-browser-node.html, 2015.

[26] Intrael, "Google Code," https://code.google.com/p/intrael/, 2011.

[27] npm Inc., "openni-browser," https://goo.gl/OB34cm, 2015.

[28] Octo Technology, “jKinect,” http://jkinect.com/, 2014.

[29] Motion Arcade Inc., “Zigfu Development Kit - ZDK,” http://zigfu.com/en/zdk, 2015.

[30] Google, “Chrome,” https://www.google.com/chrome, 2015.

[31] Microsoft, “Internet Explorer,” https://microsoft.com/ie, 2015.

[32] Opera Software, “Opera Browser,” https://www.opera.com, 2015.

[33] Mozilla, “Firefox,” https://mozilla.org, 2015.

[34] Apple, “Safari,” https://www.apple.com/safari/, 2015.

[35] Mr. Doob. (2015, August) stats.js. https://github.com/mrdoob/stats.js/.

[36] David Catuhe. (2015, August) babylon.js. http://www.babylonjs.com/.