A Web platform for the interactive visualization and analysis of the 3D fractal dimension of MRI data

J. Jiménez a, *, A.M. López a, J. Cruz a, F.J. Esteban b, J. Navas c, P. Villoslada d, J. Ruiz de Miras a

a Department of Computer Science, University of Jaén, Jaén, Spain
b Department of Experimental Biology, University of Jaén, Jaén, Spain
c Department of Mathematics, University of Jaén, Jaén, Spain
d Service of Neurology, Hospital Clinic Barcelona, Barcelona, Spain

A R T I C L E   I N F O

Article history:
Received 21 June 2013
Accepted 24 May 2014
Available online 6 June 2014

Keywords:
3D fractal dimension
Box-counting
Skeleton
Magnetic-resonance imaging
WebGL
CUDA

A B S T R A C T

This study presents a Web platform (http://3dfd.ujaen.es) for computing and analyzing the 3D fractal dimension (3DFD) from volumetric data in an efficient, visual and interactive way. The Web platform is specially designed for working with magnetic resonance images (MRIs) of the brain. The program estimates the 3DFD by calculating the 3D box-counting of the entire volume of the brain, and also of its 3D skeleton. All of this is done in a graphical, fast and optimized way by using novel technologies like CUDA and WebGL. The usefulness of the Web platform presented is demonstrated by its application in a case study where an analysis and characterization of groups of 3D MR images is performed for three neurodegenerative diseases: Multiple Sclerosis, Intrauterine Growth Restriction and Alzheimer’s disease. To the best of our knowledge, this is the first Web platform that allows the users to calculate, visualize, analyze and compare the 3DFD from MRI images in the cloud.

© 2014 Elsevier Inc. All rights reserved.

1. Introduction

A fractal is a geometry object that is self-similar at different scales. Fractals are described by fractal geometry, which was first proposed by Benoît Mandelbrot [1]. The fractal dimension (FD) is a non-integer number that characterizes the morphometric variability of a complex and irregular shape. In recent years, several studies have shown the utility of the FD as a good descriptor of structures in a wide range of scientific fields such as biomedicine [2–9], materials analysis [10–12], environmental science [13,14], and computer graphics [15], among others.

When performing the calculation of the FD value, some authors choose to implement an algorithm that fits to their particular case study. This is the case of the papers presented in [7,8] or [16], where a MacOS application that reads directly from FreeSurfer files is developed [17]. But there also exist some general desktop applications that allow the user to calculate the FD. General and wide-use image analysis programs such as ImageJ [18], and also more specific programs for the FD calculation such as HarPA – Harmonic and Fractal Image Analysis [19]. There are also specific programs designed to calculate the FD. This is the case of the Visual Basic tool announced in [20] and later improved in [21] by Grossu et al., a tool that allows us to estimate the FD of both 2D and 3D images represented as CSV files. Sezer presents in [22] a Java-based software called FRACEK for calculating the fractal dimension of 2D images. In [23], the authors present a Visual C# software for estimating the FD of 2D vector data. Recently, a C++ program designed for calculating the 3DFD of volumes generated from Magnetic Resonance Images (MRI) was presented in [24]. Finally, a so-called fast MatLab script for calculating both 2D and 3DFD is presented in [11].

All of these desktop solutions are designed for a specific platform, and access to them could be difficult. Nowadays, web-based applications have become a very interesting solution and are overcoming the classic desktop software development [25]. The software that resides on the web engine presents some advantages over local applications. For example, when a software application is developed for the web, it could be directly accessed and executed by any user around the world, independently of his running OS (cross-platform character) and without installing any special software. Only an internet connection and a web browser for executing the application are needed. This is the main advantage of the web-based applications over the desktop-based ones. In addition, Web applications are centralized in the cloud, in a computer

* Corresponding author. Address: Department of Computer Science, University of Jaén, Campus Las Lagunillas s/n, 23071 Jaén, Spain. Fax: +34 953212472.
E-mail address: jjibanez@ujaen.es (J. Jiménez).

http://dx.doi.org/10.1016/j.jbi.2014.05.011
1532-0464/© 2014 Elsevier Inc. All rights reserved.
server, so it is guaranteed that the users will always access the newest version of the software, which is a very important feature. For these reasons, in recent years several web-based applications for resolving many kinds of problems have appeared. Focusing on the medical imaging research field, some platforms that allow visualizing and processing 2D and 3D medical images have been developed. For example, in [26] an online 3D visor of medical images was published. Other interesting web-based applications that allow processing 2D and 3D medical images have been published in [27,28].

But to the best of our knowledge there are no web-based applications designed to calculate the FD of a 2D/3D image yet. Therefore, we focus our efforts on the development of a Web platform that allows the user to calculate and analyze the FD value of magnetic resonance images (MRI), independently of the characteristics of his personal computer. We focus on this class of images because, nowadays, the FD analysis of medical images is the most established and widely used FD applied procedure, but our software could be extended to process other scientific images.

The rest of the paper is organized as follows. First, we describe the theoretical basis and the computational methods on which our Web platform is based. Then some concrete implementation details are highlighted due to their interest and novelty. Afterwards, the user interface and main functionalities of the Web platform are outlined. In Section 5 we show a case study in which our Web platform is used for analyzing and characterizing 3D MR brain images related to three neurodegenerative diseases. Finally, we summarize our work in the conclusions section.

2. 3D fractal dimension: theoretical base

The main feature of the web-based application presented in this paper is the estimation of the FD value of 3D volumes constructed from MR images. Thus, in this section we describe the basic algorithms and the theoretical basis for our Web application. In addition, the algorithm improvements that have been studied and implemented are outlined.

2.1. 3D volume generation from 2D images

A 3D volume, also known as 3D image or 3D matrix in the literature, is usually generated by stacking several 2D images or slices. This way of stacking images implies that each final (x, y, z) position of the 3D volume corresponds to the value of the pixel with coordinates (x, y) obtained from the 2D image number z. Just as we refer with the word “pixel” to each position of a 2D image, each position of a 3D volume is generally known as a “voxel”. In our Web platform, biomedical images are the expected input data, as previously mentioned, so in this case each 2D image is an MRI slice, and each voxel of the 3D volume stores an integer value between 0 and 255, as previously seen, it is necessary to set a threshold value, u, to determine whether a pixel belongs to the object or whether it corresponds to the background. In other words, it is necessary to binarize the images. Therefore, a voxel with an edge size of l (so l + 1 consecutive pixels in the three dimensions from the new voxel) will be labeled as black if all the pixels contained in that voxel have a value equal to or greater than the threshold u. On the contrary, that voxel will be considered as white if all the covered pixels have a value less than u, and the voxel will be labeled as gray in all other cases.

A graphical example of the 3D box-counting process is shown in Fig. 1 for different edge sizes, where only the gray voxels are represented. Gray voxels correspond to the boundary voxels of the object, i.e. the surface cuts across them, while black voxels represent the inner sections. Logically, the higher the value of the edge box size is, the lower the resolution and the accuracy of the obtained representation are. So we can consider the box-counting algorithm as a Level of Detail (LOD) representation, since the original 3D volume has to be represented with different voxel resolutions, with the aim of counting how many boxes are necessary in each voxelization.

2.2. CUDA programming model

The use of the Graphic Processing Unit (GPU) for improving algorithms and processes with a general purpose (GPGPU) has grown exponentially in recent times. The most common GPGPU programming models are NVIDIA CUDA [31] and Kronos OpenCL [32], CUDA being the more established nowadays. We use CUDA in the Web platform to parallelize and optimize the more time-consuming algorithms involved in 3DFD computation.

The CUDA programming model allows the programmer to simultaneously launch thousands of GPU threads. Each thread executes the same function, called a kernel, on a dataset. All threads are organized into several levels. Individual threads are grouped in 1-, 2- or 3-dimensional thread-blocks. Thread-blocks are grouped in a mono- or bi-dimensional grid, and also, only on NVIDIA GF100 GPUs [33], in three-dimensional grids.

Each CUDA-capable GPU has several Multi Processors (MPs), each one consisting of some Streaming Processors (SPs) with SIMD (Simple Instruction – Multiple Data) architecture. These processors are responsible for executing all the threads in a parallel way. The CUDA memory architecture is explicitly managed by the programmer. Each device has a large amount of slow Global Memory and some small-size and fast-access memory modules, called Shared Memory. Since the arrival of the GF100 Fermi architecture [33], a real cache memory hierarchy has been introduced. The CUDA organization philosophy allows a high level of scalability because thread-blocks are automatically assigned to idle MPs, independently of the number of existing cores in the device. Therefore, the same compiled program can be executed on different and heterogeneous CUDA capable devices.

2.3. 3D fractal dimension computation

The FD of a 3D volume (3DFD) can be estimated by applying different methods [34]. One of the most widely used methods is the one known as box-counting calculation [35], a suitable method when working with statistically self-similar models [36]. In general terms, the box-counting algorithm consists of covering the object with a grid of square boxes with an edge size of l, then counting how many of these square boxes are totally or partially filled by the model represented in the 2D image or the 3D volume. Each one of these square boxes has a specific classification depending on whether it is completely, partially or not contained by the model. In the first case (full filled box), the box is labeled as black voxel; in the second case (partially filled box) it will be labeled as a gray voxel; and finally the rest of the boxes are labeled as white voxels. This process of counting and classifying boxes must be repeated for different box edge sizes, and also for different values of l.

Since each pixel of the 2D slices takes an integer value between 0 and 255, as previously seen, it is necessary to set a threshold value, u, to determine whether a pixel belongs to the object or whether it corresponds to the background. In other words, it is necessary to binarize the images. Therefore, a voxel with an edge size of l (so l + 1 consecutive pixels in the three dimensions from the new voxel) will be labeled as black if all the pixels contained in that voxel have a value equal to or greater than the threshold u. On the contrary, that voxel will be considered as white if all the covered pixels have a value less than u, and the voxel will be labeled as gray in all other cases.

A graphical example of the 3D box-counting process is shown in Fig. 1 for different edge sizes, where only the gray voxels are represented. Gray voxels correspond to the boundary voxels of the object, i.e. the surface cuts across them, while black voxels represent the inner sections. Logically, the higher the value of the edge box size is, the lower the resolution and the accuracy of the obtained representation are. So we can consider the box-counting algorithm as a Level of Detail (LOD) representation, since the original 3D volume has to be represented with different voxel resolutions, with the aim of counting how many boxes are necessary in each voxelization.
linear regression in which the $X$ axis represents the inverse of the edge size of the voxel, $l$, and the $Y$ axis represents the counting, $N(l)$, for the selected type of voxel. In other words, in the $X$ axis each one of the considered voxelization resolutions is represented, and the $Y$ axis shows the number of voxels that have been counted for its corresponding resolution. The final value for the 3DFD corresponds to the slope of the exposed linear regression:

$$3DFD = -\frac{\ln N(l)}{\ln l}$$

An example of this linear regression is shown in Fig. 2. Once the line is generated, it is necessary to adjust it to obtain the most reliable FD estimation. Thus, the selection of the voxel size range affects the value of the estimated FD. But this selection is still an open issue. The general method consists of discarding the points that fall outside of the linear part, usually the first and the last points of the line. Then the linear portion of the log–log plot that finally determines the FD value is obtained by selecting the point range that maximizes the correlation value [2,11].

In Fig. 2 the red dots correspond to the voxel edge sizes that have been discarded, while the green dots represent the selected values when calculating the final regression line. In this example, the box-counting ($Y$ axis values) is the sum of both gray and black voxels.

Some studies have focused on improving the exposed box-counting algorithm in terms of efficiency. One of the more important and widely used algorithms for performing the box-counting calculation is the one developed by Liebotich and Toth [37] and later improved by Hou et al. [38]. This algorithm is especially indicated for binary models, like the segmented MR image and its skeleton representation, as will be seen later. It has a time complexity of $O(N \cdot \ln^2 N)$. In brief, Hou’s algorithm assigns a unique bit string code to each position of the object, and then these codes are sorted according to their values, low to high. In the last step, the algorithm iteratively applies a mask to the codes and checks the values that are equal, since they correspond to the same box, thus labeling the boxes and obtaining the box-counting.

In a previous study we further optimize Hou’s algorithm by developing an implementation designed for its execution on the Graphic Processing Unit (GPU) [39]. That optimized GPU implementation was developed by using NVIDIA CUDA. In this way the running time of the box-counting algorithm is decreased, the new GPU-based algorithm being 28 times faster than the classic mono threaded implementation on CPU, according to our tests. This improvement is especially interesting when performing the FD calculation on huge datasets of images, which is a usual situation. In other work we have also analyzed the performance obtained by using OpenCL in both GPU and multi-core CPUs [40].

However, Hou’s box-counting algorithm, both the original algorithm and the one optimized for the GPU, has a limitation: it computes the box-counting only considering those boxes that have a power-of-2 edge size. Therefore, the number of multi-resolution grids of boxes used to cover the voxelized object is lower than

![Fig. 1. Box-counting voxelization of a sample 3D image (in this case, the segmented gray matter of a brain composed of 155 MR Images of $155 \times 188$ pixels each), for different voxel edge sizes.](image1)

![Fig. 2. Linear regression for 3DFD computation from box-counting data.](image2)
the number of grids used with the classic box-counting algorithm, because of the limited number of values that I could take. Therefore, there are fewer points to adjust the regression line and then the FD estimation could not be as accurate as when all the points are available. This fact particularly affects the 3DFD estimation, since 3D models normally have a lower resolution in each axis than 2D images. So, for flexibility reasons and according to our successful previous studies [4–6,24], we include the classic algorithm in our Web platform, which offers sufficient values to estimate the FD in any case. Meanwhile, we are performing different tests and studies to ensure that, with the values obtained with Hou’s box-counting algorithm, the FD analysis provides similar results to our previous and contrasted studies on brain characterization [4–6,24].

2.4. Skeleton generation from a 3D volume

In general terms, a skeleton is a special representation of an object that captures its topological essence in a simple and very compact way. The skeleton known as curve skeleton [41] is a 1D representation for a 3D object, and it is the most compact and minimal object representation possible. So from here on, and throughout this paper, when we use the term skeleton we refer to the curve skeleton, unless otherwise stated. Skeletons have a wide variety of uses and applications [42]: virtual navigation, virtual endoscopy and animation, segmentation of anatomical structures, medical image registration, matching of 3D objects, morphing, surface reconstruction and mesh repair, among others.

In many biomedical studies it is interesting to perform the FD calculation both on the original images and also on their skeleton representation [3–5,43–46]. This is because, by performing the FD analysis on the skeleton a double indicator could be obtained, which is related to the hypothesis being analyzed, and it is even possible to obtain some results and conclusions that could not be obtained solely with the FD results associated with the original 3D volume.

There are several kinds of algorithms for calculating the 3D skeleton [42]: based on the thinning technique, on the distance fields theory, and also on the Voronoi diagram. In our Web platform we use a thinning algorithm because of its usefulness and properties when working with 3D volumes represented by voxels, not represented by a mesh. Thinning algorithms work by iteratively deleting voxels of the original 3D object until its minimal representation is obtained. There are several thinning algorithms in the literature, but we have implemented into our Web platform the one introduced in [47], one of the most widely referenced and used algorithms presented in the literature.

As in the case of the box-counting calculation, to generate the skeleton the original 3D image has to be binarized in order to determine the positions that belong to the object, and consequently the ones that belong to the background. Therefore, a threshold \( u \) has to be set and only the pixels of the images that have a value greater or equal to it are considered in the skeletonization process. Once the 3D skeleton is generated, the 3DFD calculation could be performed as outlined in Section 2.3, but starting from the 3D matrix defined by the skeleton itself and without applying a threshold cut, because the threshold is applied at the beginning, when the skeleton is generated.

An important problem presented by all the thinning algorithms is their high computational cost; they are very time-consuming processes. For example, according to our tests the 3D skeleton generation of a standard model with a resolution of 256 voxels in each axis always surpasses one minute of running time. In the case of a 3D model with higher resolution, i.e., 512 voxels in each axis, the execution takes around 15 min, so the running time has an almost exponential increase as the model resolution is higher. This fact is not acceptable in interactive and dynamic environments such as web applications, where a fast response of the system is necessary. For this reason, in a previous study we optimized Palágyi's thinning algorithm by developing an implementation designed for its fast and parallel execution on GPU [48,49]. This implementation was also developed by using NVIDIA CUDA, and it has been included in the Web application, ensuring running time rates that allow an interactive and satisfactory user experience.

3. Implementation details

In this section some implementations details regarding the development and optimization of the 3DFD Web platform are outlined. In addition to the hardware and the general structure of the platform, we highlight those aspects and novel technologies that are especially useful for its performance and usability, such as the ones related to multi-GPU programming, or the 3D graphics development for the web.

3.1. Hardware available on the web server

When developing a Web platform with intensive computation and high storage requirements, it is important to setup a powerful hardware configuration for successfully managing the simultaneous connections, and also for optimizing the algorithms in a parallel way, then achieving good performance rates and ensuring a satisfactory user experience.

The server computer which our web-based software runs on has two Intel Xeon E5620@2.40 GHz CPUs with 4 independent cores each and a capacity of simultaneously managing 8 threads. This was accompanied by 12 GB RAM and two hard disk drives of 1 TB each, enough for storing the received images. In addition, the system has two NVIDIA GPUs for executing the optimized processing algorithms in a fast way. These two GPUs are a NVIDIA Tesla C2050 (Tesla2050) and a NVIDIA GeForce GTX 670 (GTX670).

Table 1 shows a comparison of the main hardware specifications of both GPUs. In general terms, GTX670 stands out for its higher number of processing cores but the Tesla presents better memory-related specifications, i.e., global and cache memory amount and memory bus bandwidth. Currently, the NVIDIA driver version 306.97 is installed, together with the CUDA driver version 4.2.

3.2. Design of the Web application

In this section we explain the design and architecture of the Web platform. As is well-known, the client (a Web browser) transmits the orders to the server, unless the client can dynamically resolve the user’s request by using some technology such as JavaScript. The requests to the server can be dealt with in a synchronous or asynchronous way, by using AJAX calls in the asynchronous case. Then the server processes and performs the requested operations through a server-side programming language, PHP in our case, or by using local system procedures and software. Therefore, the server stores the database of the application, and also the system scripts, toolkits and components used to execute the processes. When the server has processed the client request, it returns the results by refreshing the Web page, in the synchronous-call case, or by sending an XML or JSON (JavaScript Object Notation) [50] document that would be asynchronously interpreted in a second plane procedure by the client.

Fig. 3 shows a brief scheme of the system structure and technologies used to develop the software presented in this paper.

The graphical interface should be user-friendly and has to allow the maximum online interaction capabilities. By using simple web
forms, the whole webpage has to be reloaded after each interaction of the user, a fact that in most cases does not provide a satisfactory and appropriate user experience [51]. Therefore, it is necessary to use some dynamic technologies that allow the user to perform assertions on the fly, such as JavaScript, or even to send second-plane requests to the server, such as AJAX.

In our Web application the server performs the calculations by using PHP, and a MySQL database to store the data. Some command-line binary software is also executed. This external software is related, in our case, with medical image managing, by using ImageJ scripts to encode/decode the uploaded images. In addition, C++/CUDA optimized algorithms are invoked when required. This binary software will return its results in a standard array that is interpreted by the PHP code, thus obtaining the results that will be interpreted and transferred to the client.

In the Web application presented throughout this paper we focus on MRI medical images because, as seen in the introduction section, the biomedical image study has become one of the most important research applied fields related to FD analysis. For this reason medical images are the main data input of our application, and the starting point for any later process. Thus, the users have to provide their previously segmented medical images converted to either the Analyze 7.5 file format [52] or to the DICOM file format [53].

Fig. 4 shows the main modules and the data flow of the Web application. It is organized in three main parts: one which corresponds with the image upload process, both in a single or a multiple way, another one related to the single or multiple 3DFD calculation on these images (or their skeleton representation) by applying the box-counting method, and a third part which is related to the analysis, the 2D/3D visualization and managing processes of the previously stored 3D images and 3DFD results.

### Table 1

<table>
<thead>
<tr>
<th></th>
<th>Tesla C2050</th>
<th>GeForce GTX 670</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA computing capability</td>
<td>2.0</td>
<td>3.0</td>
</tr>
<tr>
<td>NVIDIA architecture</td>
<td>Fermi</td>
<td>Kepler</td>
</tr>
<tr>
<td>GPU clock rate</td>
<td>1.15 GHz</td>
<td>1.06 GHz</td>
</tr>
<tr>
<td>Multiprocessors (MPs)</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>Cores per MP</td>
<td>32</td>
<td>192</td>
</tr>
<tr>
<td>CUDA shared mem.</td>
<td>44</td>
<td>1344</td>
</tr>
<tr>
<td>Simultaneous threads per MP</td>
<td>1536</td>
<td>2048</td>
</tr>
<tr>
<td>Global memory amount</td>
<td>3 GB</td>
<td>2 GB</td>
</tr>
<tr>
<td>Memory bus width</td>
<td>384 bits</td>
<td>256 bits</td>
</tr>
<tr>
<td>Registers (32 bits)</td>
<td>32 K</td>
<td>64 K</td>
</tr>
<tr>
<td>Max registers per thread for a full MP occupancy</td>
<td>21 registers</td>
<td>32 registers</td>
</tr>
<tr>
<td>Shared memory * L1 cache amount</td>
<td>64 KB</td>
<td>64 KB</td>
</tr>
<tr>
<td>Shared memory amount</td>
<td>Up to 48 KB</td>
<td>Up to 48 KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>768 KB</td>
<td>512 KB</td>
</tr>
</tbody>
</table>

## 3.3. 3D interactive visualization of MRI data on the Web

In this section we describe the technologies used and the architecture developed to allow an interactive visualization of the 3D structures required for the 3DFD analysis in our Web platform.

WebGL (Web-based Graphics Library) is a standard Web API developed by the Khronos Group. This API extends the capability of the JavaScript programming language, allowing it to generate interactive 3D graphics in any compatible Web browser, without the need for extra plug-ins. The WebGL API [54] is based on the OpenGL ES 2.0 standard [55]; it has a cross-platform character and is royalty-free. WebGL objects are shown on web browsers thanks to the HTML5 canvas element, and their related data is accessible through the Document Object Model (DOM) interface, so it could be considered as an alternative option to the HTML5 canvas.

WebGL is implemented right into the web browser, so each browser vendor is responsible for supporting the WebGL standard. Thus, the desktop Web browsers that currently offer support for representing the WebGL canvas are: Mozilla Firefox (since version 4.0), Google Chrome (since version 9), Safari (since version 5.1), and Opera (since version 12). Microsoft Internet Explorer does not offer support for WebGL yet. However, a plug-in that will allow rendering WebGL on Internet Explorer 6.0 and newer versions [56].

Some frameworks have appeared in recent dates to facilitate the WebGL programming. Each one of these frameworks introduces an extra API layer, coded with JavaScript, which offers a high-level communication point between the developer’s code and the low-level WebGL API. In our case, in order to code the 3D graphical representation in the web application, we choose a well-known and tested WebGL framework: Three.js [57].

We use WebGL in our Web platform to show a web-integrated 3D visualization of the original biomedical images uploaded to the system, together with its multi-resolution 3D voxelized representations associated with the box-counting calculation (i.e. Fig. 1). In addition the skeleton representation of each model is also depicted, to show the differences between this representation and its corresponding full voxelization.

When working with WebGL it must be taken into account that, in addition to the classical problems associated with the 3D graphics topic we have to deal with some additional determining aspects, such as the data transfer through the network between the server and the client, the computational load that the JavaScript language is able to support, or the memory allocation limits set by some web browsers. All these factors imply that we must study and develop a particular solution in order to achieve the required performance when working with the high volume of data our system has to manage.

In order to represent the original MR image (composed of several 2D grayscale slices distributed in three views) in a 3D space, the browser requests each one of these slices at once from the server, so a previous data load is necessary to allow the user to visualize and interact with the 3D representation. Once the MR
image is transferred to the client, the user could interact with it in real time by using the native navigation controls offered by Three.js/WebGL. Fig. 10D shows an example of this 3D interactive visualization in the Web platform of the original MR images through WebGL.

Regarding the visualization of the voxelized models associated with the box-counting calculation (the different LODs of the 3D model), the best option could be to operate in the same way as in the previous case, by transferring the MR image to the client and then performing the required operations on the browser. But this way the computational load associated to JavaScript would increase considerably, since it would have to identify the gray and black voxels of the model for the selected resolution. In addition, it is not efficient to transfer the full gray-scale MR image from the server to the client when only a binary voxelization, usually without having the full resolution, is going to be represented. Therefore, it is necessary to adopt another strategy for the process of representing these multi-resolution models.

Fig. 5 shows the whole process for calculating, transferring and interactive visualizing of the different voxelizations needed for a 3D box-counting calculation. When the client requests the visualization of a 3D representation, with a concrete voxel size resolution and a threshold, the server locally runs a C-coded program which is responsible for generating the 3D matrix which corresponds to the desired 3D voxelization, thus identifying the black, gray and white voxels. The size of this 3D matrix would vary depending on the resolution (the LOD level) of the voxelized model that is going to be represented. Once the box-counting calculation has been performed, a PHP script will interpret the 3D matrix returned to encode it by using JSON. The JSON object is transferred to the client browser in a second plane response, by using the AJAX communication protocol. Then, once the data has been received on the web-browser, the values of the 3D matrix are read in order to identify the voxels that are going to be represented in the WebGL canvas. Afterwards, while the 3D model is shown and while the user interacts with it, new asynchronous requests to the server are performed. In this way, the 3D models with a resolution just above and below that of the model that it is being represented are requested in advance, and stored in the client memory. Thus, when the user modifies the resolution parameter (the 3D model LOD), the web application needs only draw it in the WebGL canvas minimizing the execution time, since that 3D representation has been previously requested, calculated and saved, by using the AJAX second plane calls. If the user changes the value of the threshold parameter the resulting 3D model has to be recalculated, so the application cannot use the one stored in memory and has to perform a new request to the server. In addition, when a 3D model is finally displayed its final geometry is also stored in memory, thus avoiding recalculation for a new display.

Starting from the client computer and following the orange path in Fig. 5 (the continuous line), we show how the first 3D model, with a default LOD value, is requested to the server, and how this returns the encoded data as a JSON object. Once this object is processed in the client, new LODs of the same 3D model are requested in advance, by using second plane AJAX calls (black path – dashed line in Fig. 5). If the user wants to visualize an LOD of the model that has been previously requested to the server, the web application follows the blue path in Fig. 5 (dotted line) and directly obtains, without involving the server, the JSON object that represents the selected 3D model, and generates the geometry. In the case that the required 3D object has been previously displayed, the Web application needs only retrieve the WebGL geometry which corresponds to the required LOD (green path in Fig. 5) and draw it.

Thanks to the asynchronous and second plane requests, the communication time between client and server is partially hidden. In addition, by maintaining a cache memory with the objects that have been displayed previously, the running time is greatly decreased when representing these repeated models. Regarding the efficiency while displaying each voxelized 3D model, the main problem appears for high-resolution models. In this case it is necessary to represent around one million voxels, an amount of data that is not easily manageable, causing the crash of the web browser if we try to display it in a direct way. So it is necessary to find efficient solutions that allow us to display high-resolution voxelizations in the WebGL canvas.

With the aim of resolving this issue, we apply two optimizations. First, it is necessary to reduce the geometry of the object to be represented. To do this without affecting the visualization, the representation of the voxels that are not visible has to be avoided. When drawing a solid cube with the Three.js API, we have to select which sides of the cube are graphically represented.
Therefore, by analyzing the neighbourhood of each voxel we determine which cube sides are going to remain hidden, and the geometry of these sides are not rendered. In this way a voxel that is totally surrounded by other voxels will not be represented at all, a fact that is not visually perceptible, thus decreasing the scene geometry and improving the general performance.

In addition, it is important to reduce the number of WebGL calls as much as possible. Three.js offers a function that allows us to reduce the calls to the WebGL API and then also the data interchange between the CPU and the GPU. This function is THREE.GeometryUtils.merge(). This process binds a set of meshes, forming a single geometry. This process also allows us to reuse an existing mesh, change its coordinates or link together other meshes, thus reducing the creation of new objects and improving the performance. If this technique of geometry merging is not used, for each voxel a completely new cube mesh would be generated and added to the 3D scene, increasing the complexity. This technique can be used only for representing static geometry, e.g. a brain voxelization in which it is not necessary to individually translate or rotate each one of the voxels that form the whole model.

Fig. 10 shows several examples of the obtained Web-GL visualization of an LOD for a voxelized brain, its skeleton, a 2D slice of this LOD and the original MR images in a 3D representation.

3.4. Optimization of the skeleton generation algorithm: a multi-device approach

As outlined in Section 2.4, in order to obtain the skeleton of the 3D voxelized models uploaded to our system we use a fast thinning algorithm presented in [48], designed for its execution on a GPU. This algorithm is efficient and shows impressive speedups compared with the standard single-threaded CPU algorithm. However, it was designed for its execution on a single GPU, while in our computer server there are two GPUs, as seen in Section 3.1. Taking into account that the skeleton generation algorithm is the most time consuming process in our system, we decided to enter into the multi-device algorithm optimization to benefit from the whole hardware platform.

This multi-device optimization has followed these steps:

- The input dataset has to be split into two parts, so the algorithm is executed on each device in a parallel way over its own section of the dataset. Fig. 6 shows how when the original dataset is divided the value X is placed in the GPU A while its neighbors Y3, Y4, and Y5 are placed in GPU B. Therefore, it is necessary to duplicate some data (a slice, shaded squares in Fig. 6) in order to make available all the neighbors necessary for checking the deletion condition of each voxel (the thinning algorithm determines whether a voxel has to be deleted or not by reading its neighborhood). Then the boundary slice of the GPU B is replicated in A and vice versa, each neighbor being locally readable. The thinning algorithm is an iterative process, so at the end of each one of the iterations the GPU A has to update in the GPU B the data that has been just calculated, and vice versa.

- We measure the isolated performance of both GPUs to quantify the performance difference between them, then determining how the computational load has to be balanced. We used as input data the anatomical models of twenty generated normal brains presented in [58–60] and we selected five brain tissue classes with different topologies: cerebrospinal fluid (CSF), gray matter, white matter, skull and bone marrow. The performance test results in Table 2 show that the Tesla2050 GPU is 1.22 times faster than the GTX670. This performance difference implies that a theoretical and static computational load balance of 61–39% would be ideal for reaching the maximum multi-GPU algorithm performance.

- When splitting the voxelized model to send each part to a different GPU, we divide it at the slice level. Each one of the slices contains a different and unpredictable number of voxels.
belonging to the object. Therefore, when we apply the previously estimated optimal computational load distribution, the high performance peaks do not appear when expected. Fig. 7A shows how maximum performance is achieved with different data distribution depending on how the model is partitioned. For a human brain model, the maximum level of symmetry is found in the sagittal axis, and therefore we did the partition of the brain by its cerebral hemispheres. The performance peak was reached when we sent to the Tesla2050 GPU approximately 55% of the slices, as can be seen in Fig. 7B. Finally, Fig. 7 shows how an average speedup of 1.6× was achieved with respect the single-GPU algorithm executed on the Tesla2050 GPU (the faster one). The ideal 2× speedup is not reached due to data swapping between GPUs and since the two GPUs are not identical.

This procedure refers to the case in which the model has to be partitioned between two GPUs. If another hardware configuration is available, a new study to search for the optimal dataset distribution between the devices is required, but this performance analysis only has to be performed one time.

4. User interface and functionality

The user interface (UI) is a key element in any software application, because it is the communication point between the user and the system. In general terms, our aim was to develop a Web application that allows the user to easily calculate and analyze the FD of a 3D image, as described in Section 3.3. A 3D voxelized LOD representation of the whole 3D image is shown in Fig. 10A. We can also visualize the 3D skeleton representation of the original model (Fig. 10B). As seen in previous sections, this algorithm is developed and well optimized by using CUDA in a multi-GPU implementation, thus obtaining the interactivity needed in any web application. By interacting with the UI sliders, a slice by slice visualization of either the voxelized 3D model or the 3D skeleton is obtained (Fig. 10C). The “Images” visualization mode shows the original MR images combined and represented in a 3D space (Fig. 10D).

- Fig. 8 shows a snapshot of the application related to the upload process of a new 3D image to the server. Currently, the Web platform does not include any segmentation procedure, so the user must provide images already segmented. It is usually necessary to analyze the FD of a set of several images, e.g. the gray matter of several patients. For this reason the Web platform also offers the option of uploading a set of images into a packed file. In addition, each uploaded image can be included into a group, in order to later studies and comparative analysis purposes.

- Fig. 9 shows how the platform displays the data and the 2D slices associated with each 3D image.

- In addition to the 2D slice visualization, the application allows us to visualize a WebGL based 3D representation of the 3D image, as described in Section 3.3. A 3D voxelized LOD representation of the whole 3D image is shown in Fig. 10A. We can also visualize the 3D skeleton representation of the original model (Fig. 10B). As seen in previous sections, this algorithm is developed and well optimized by using CUDA in a multi-GPU implementation, thus obtaining the interactivity needed in any web application. By interacting with the UI sliders, a slice by slice visualization of either the voxelized 3D model or the 3D skeleton is obtained (Fig. 10C). The “Images” visualization mode shows the original MR images combined and represented in a 3D space (Fig. 10D).

- The result of the box-counting algorithm is displayed in a log-log scattered plot, as can be seen in Fig. 11. The user can discard those plot points that do not follow a linear trend, by observing the regression line itself and the correlation value, thus adjusting the regression line and then approximating the FD value in a better way. The interactivity with the graph and the final FD results is performed dynamically, without refreshing the webpage, thanks to the JavaScript/AJAX implementation. Thus a satisfactory user experience is achieved. Once the user has adjusted the plot line, the 3DFD value and its associated values of error and correlation can be stored in the database. The Web platform can also simultaneously execute the FD calculation on multiple images with multiple threshold values, both on their original and skeleton representation, and also with different ranges of points for the regression line adjustment. The results are grouped into a formatted text file that can be downloaded and further processed to search for the desired parameters.

- Fig. 12 shows the user interface to calculate the 3DFD on one or more image groups with multiple images. This is organized into three steps; firstly, the user has to select the input parameters. To determine these parameters it is necessary to experiment with them in the single 3DFD calculation section, as seen in Fig. 11, looking for the greatest correlation value. Secondly,
the user has to select the image groups on which the 3DFD is going to be calculated. Finally, the 3DFD values (each value associated to one of the selected images) are automatically calculated in a second plane procedure. The user interface dynamically communicates to the user at which point the process is, and also which image is being analyzed at the moment.

- The numeric FD calculations could be queried and displayed on the Web application with the aim of analyzing the results, as Fig. 13 shows.

The Web platform offers three types of plots for representing previously calculated 3DFD results, together with the raw values, thus allowing an easy, graphical and statistical analysis of the data. The user can represent a scattered, a box and whiskers or a SEM (Standard Error of the Mean) plot. This is shown in more detail in the next section.

5. Case study: computation and comparative analysis of MRI data in neurodegenerative diseases

In the previous sections we have described the Web platform, its functionality and the specific techniques and algorithms employed to improve its performance in obtaining the interactive times needed in any Web application. In this section we show how we can use the Web platform to perform a comparative study based on the 3DFD analysis of 3D images that belong to different and related groups. This test aims to show how typical 3DFD studies can be performed entirely in the Web platform as the biomedical experts currently perform them using a diversity of independent software. As our case study we show three examples of studies of three neurodegenerative diseases (Multiple Sclerosis, Alzheimer’s disease and Intrauterine growth restriction) in which 3DFD has been revealed as a very promising descriptor in our previous work.
In the first study [4], 3DFD analysis was applied to the gray matter of 52 patients with MS and 20 healthy controls. The results of this work indicated that gray matter (GM) morphology is abnormal in patients with MS and that this alteration appears early in the course of the disease. This study found that patients with MS had an increase in the 3DFD of the GM compared to controls. Such differences in the 3DFD of the GM were found for either CIS (clinically isolated syndrome) or RRMS (relapsing-remitting MS) compared to controls. Finally, a significant difference in the GM 3DFD between CIS and RRMS patients was also found.

Fig. 10 shows the plot of the standard error of the mean obtained in the Web platform from the data used in [4]. Three groups of images were created in the platform: HC_MS_STUDY group for healthy controls, MS_CIS group for CIS patients and MS_RR group for RRMS patients. The 3DFD for the 3D images of each group was calculated using the parameters shown in Fig. 14: a threshold of 70 and a range of voxel sizes from 6 to 16 from the Gray + Black data of the box-counting. The selection of these values of the parameters comes from [4], where a previous manual segmentation of the images showed 70 as the most appro-
Appropriate threshold value (in a gray scale from 0 to 255) and, for this threshold, the range of voxel sizes from 6 to 16 maximized the correlation. This fact was also asserted in this case by using the single 3DFD calculation tool of the Web platform, presented in the previous section.

In the second study [5], a 3DFD analysis was applied to a sample of 18 singleton premature infants with a prenatal diagnosis of severe IUGR (Intrauterine growth restriction, a major cause of preterm delivery, neonatal morbidities, neonatal death, and stillbirth) diagnosed before 34 weeks of gestation, 15 preterm infants matched one-to-one for gestational age (GA, ±2 weeks) at delivery, and 15 neonates born at term. The results indicated a significant decrease of the 3DFD of the brain GM and white matter (WM) in the IUGR group when compared to the preterm or at term controls.

Fig. 11. Graphical user interface of the UJA-3DFD Web platform. 3DFD calculation: input parameters, the box-counting and log-log linear regression computation with the final 3DFD value.

Fig. 12. Graphical user interface of the UJA-3DFD Web platform. 3DFD calculation on multiple images.
group for intrauterine growth restriction patients. In this case, we show the results for the 3DFD of the skeleton of the WM of the 3D images in each group, calculated using the following parameters: a threshold of 70 and a range of voxel sizes from 9 to 27 from the Gray + Black data of the box-counting. Similar to the previous example, the selection of the values for these parameters is the same as that used in [5].

Finally, T1-weighted magnetic resonance imaging scans of 242 subjects (116 healthy controls under sixty years old, 63 equal to or over sixty years old and 63 subjects with mild to moderate Alzheimer’s disease) were included in the analysis in [6]. The results showed a significant decrease in the 3DFD of the GM both in normal aging and in Alzheimer’s Disease (AD). The results obtained showed a significant correlation between the GM 3DFD and age. In addition, and when compared to healthy controls, this study found a significant difference between healthy controls under 60 years old and AD group for Alzheimer’s disease patients. The 3DFD for the 3D images of each group was calculated using the parameters shown in Fig. 16: a threshold of 70 and a range of voxel sizes from 2 to 10 from the Gray + Black data of the box-counting (the same values used in [6]).

With these three examples we showed how the Web platform allows a comparative, interactive and on-line analysis of different groups of 3D images based on the 3DFD. In summary, this analysis starts by uploading the 3D images to the Web platform and associating each image to a particular group of the case study. Next, the 3DFD is calculated for every 3D image or its skeleton. The specific parameters to be used to calculate the 3DFD (threshold and range of voxel size) can be explored and tested in the single 3DFD calculation section of the application, by an interactive manipulation of the regression line associated with the box-counting data. Finally, we can graphically analyze and extract conclusions on the results obtained thanks to three different plots available in the Web platform: standard error of the mean, box and whiskers and scatter. In addition, the Web platform gives the user the numeric 3DFD values, in case it is necessary to perform additional studies external to our application.

A comparative analysis of new images uploaded to the Web platform can be performed by including these images in any of the existing groups, or by creating a new group for these images and adding this new group to the comparative plot. In any case, the values of the parameters used to calculate the 3DFD for these...
new images must be the same as those used for the images in the previous existing groups.

Regarding the computational cost of a 3DFD analysis session on our Web platform, we measure the running times associated with the IUGR study [5], the most complex and costly case among the previously described ones, since the skeleton representation of the images is needed. In Table 3 we detail the times obtained with the Web platform compared to the equivalent times but computed with the non-optimized desktop software [24]. For a fair comparison, we install the desktop application on the server where the Web application resides.

First we can see how in the Desktop application the skeleton generation process takes up almost 99% of the running time of the execution. For this reason we decided to optimize that process by using the GPU, as described in Section 3.4, thus decreasing its running time to a half of the whole box-counting process. The results presented show that an acceleration of up to $\times 40$ is achieved when comparing the running times of both desktop and

---

**Table 3**

Performance comparison between desktop and web application, both installed on the same computer server. The performance results correspond to the IUGR analysis case study. Average values of “skeleton” and “total” running times for the images contained in each group. Speedup when comparing the accumulative running time of both applications.

<table>
<thead>
<tr>
<th>Group</th>
<th>#Images</th>
<th>Desktop application</th>
<th>Optimized web application</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Sk. running time (s)</td>
<td>Total running time (s)</td>
<td>% Skeleton</td>
</tr>
<tr>
<td>HC_PTM</td>
<td>15</td>
<td>74.382</td>
<td>75.248</td>
<td>98.85</td>
</tr>
<tr>
<td>HC_TERM</td>
<td>15</td>
<td>75.435</td>
<td>76.309</td>
<td>98.85</td>
</tr>
<tr>
<td>IUGR</td>
<td>18</td>
<td>70.188</td>
<td>71.039</td>
<td>98.80</td>
</tr>
</tbody>
</table>
Web application. Thus in the desktop application the analysis of the 3 groups involved on the IUGR study takes around one hour, while in the Web platform it takes only around 86 s. These results, together with the developed UI, show that the Web platform actually allows an interactive 3DFD analysis of non-simple sets of 3D images, especially in the computations where the skeleton of the image is needed, and offer an easy and intuitive way of performing 3DFD studies in an on-line multi-platform context.

6. Conclusions

In this paper we have presented a novel Web platform for computing and analyzing the 3DFD of magnetic resonance images. We have summarized the computational methods on which our software is based, such as the box-counting computation and the skeleton generation algorithm. The design of the Web platform has been outlined; both the internal layers and the user interface design. In addition, we have shown how the available hardware, in this case two powerful GPUs, has been exploited in order to achieve the maximum performance possible. We have also focused on the interactive 3D visualization of the volumetric data involved in the 3DFD analysis with WebGL, a recent technology that offers several advantages for representing 3D models on the web.

As examples of the usefulness and validity of the Web platform, we have presented a case study in which the 3DFD values obtained from several groups of segmented MR images of human brains belonging to healthy controls and Multiple Sclerosis, IUGR and Alzheimer’s disease patients are graphically and numerically compared in an efficient way. In this way we have shown how the Web platform allows us to easily analyze and characterize groups of MR images based on an on-line and interactive computation of the 3DFD.

The Web platform is accessible through the URL http://3dfd.ujaen.es.

Nowadays, new web-based technologies are being developed with the aim of improving and increasing the computing capabilities of the web browsers. The main example of this is WebGL [61], a new standard under development by the Khronos Group. In general terms, WebGL is a wrapper of the OpenCL computing platform. The final objective with the development of WebGL is to reach a high-performance computing level in web browsers by using the client devices, i.e. multi-core CPUs or GPU. As future work, it would be very interesting to use WebGL to improve the box-counting calculation and its visualization, by joining both WebGL and OpenCL technologies in the client machine [62].

From a biomedical point of view, our aim is to extend the Web platform by including efficient and interactive automatic and semi-automatic segmentation techniques of MR images and their normalization. In this way, the Web platform will offer the users the whole lifecycle of the 3DFD analysis, from the original MR image, as obtained from the MR scanner, to the graphical analysis of the 3DFD results.

A pending task is the formal evaluation of the usability of the Web platform. We plan to use our tool in clinical studies about the application of the 3DFD of MRI in several neurodegenerative diseases. In this context, the feedback of many clinical staff will allow us to evaluate and improve the Web platform usability.

Acknowledgments

This work has been partially supported by the University of Jaén, the Caja Rural de Jaén, the Andalusian Government and the European Union (via ERDF funds) through the research projects UJA2009/13/04, UJA2013/12/04, UJA2013/08/35 and PI10-TIC-5807.

References
