International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 3 (2011) pp. 514-521
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Interactive 3D Visualization of a Large University Campus over the Web

Eduardo Vendrell[1] and Carlos Sánchez[2]

[1] Instituto de Automática e Informática Industrial, Universitat Politècnica de València,
Avenida de los Narajnos, s/n, Valencia 46021, Spain
even@upv.es

[2] Instituto de Automática e Informática Industrial, Universitat Politècnica de València,
Avenida de los Narajnos, s/n, Valencia 46021, Spain
carsanb1@posgrado.upv.es

***Abstract*: Nowadays, with the rise and generalized use of web applications and graphical hardware evolution, one of the most interesting problems deals with realistic real-time visualization of virtual environments on web browsers. This paper shows an on-line application to dynamically visualize a large campus on the World Wide Web. The application focuses on a smooth walk through a large 3D environment in real-time as an alternative way to index geographically related information. This way, contents are continuously filtered based on viewpoint's position. This can be made thanks to the availability of different models corresponding to different levels of detail (LOD) for each modeled building. A server storage model has been purposed including all models, compound of meshes, textures and information. The technique is based on an algorithm that performs a progressive refining of the models, according to the distance from the viewpoint.**

***Keywords*:** Web3D, Visualization, Graphics, Real-time rendering, Interactive**.**

I. Introduction

Internet growing technology, together with the increasing performance of low-cost home computers, makes possible web-based applications including real-time visualization of large virtual environments. In fact, this is an interesting and current problem with different approaches ranging from simulation of natural phenomena [1][2], to visualization of medical images [3].

When talking about web visualization, there are two important factors to consider: the amount of information to be displayed, and the client-server architecture. The first one is important as, typically, geometric 3D models are large files to be stored, downloaded, and displayed. The second one is related both, to the computer system designed to support a web application, and the performance of the client computer. Some approaches considering these factors can be seen at [4][5][6].

Most visualization techniques of 3D data and virtual environments are based on level of detail (LOD) models, including pre-computed lightmaps and progressive refining. Current researches focus their goal in generating optimal geometric models [7][8] and fast rendering algorithms [9][10][11]. Most of these techniques are intended for off-line applications, where time delays for model and texture loadings are minimal.

There are some web-based applications to display 3D models of large environments as GIS developments. These approaches [12][13][14] tend to use VRML a high level objected-oriented language intended to describe scenes and behaviors of 3D objects. In this application, VRML has not been used due to lacks in current players to create a complex and realistic simulation. Instead, the application has been generated under Virtools [15], a software platform intended for game application.

This paper shows the implementation of a web application to display a large campus as a virtual environment, using on-line real-time rendering. The application is based on an algorithm that executes a progressive refining of 3D models of the campus‟ buildings, stored on a server. The algorithm optimizes resources by running an asynchronous request for information while the user moves through the virtual environment. In this way, we obtain 3D web visualization, considering restrictions of the user's computer, within a reasonable response time.

II. Terminology and data storage model

A. Terminology

The object of the application described in this paper is displaying 3D objects as a part of a complete University Campus model. This Campus has to be represented as a large environment on a web browser, including linked information to all graphical elements of the simulation.

We consider that objects compose a 3D environment. Each object represents a single building or urbanistic elements such as streets, urban furniture, plants… and stores the basic properties of them. Meshes define the 3D topology of objects. Every object has to be linked with one or more meshes.

Differentiating between objects and meshes allows the application to implement the LOD technique, originally

introduced in [16]. This method is widely used in virtual reality [17] and allows to control, in real-time, the visualization quality of the 3D models. It automatically changes the complexity of the mesh and the resolution of textures according to the distance between the viewpoint and the viewed objects. By employing this technique, a great amount of 3D data can be dealt in real-time over a common network environment, such as DSL.

A mesh is composed by polygons and vertices. A vertex represents a point in the space, and defines a position along with other information. A polygon is a set of three different vertices, defining a triangle in the space, and its appearance is described by a material, composed by a shader and textures.

A shader is a low-level computer program executed in the GPU that determines the final surface appearance of a polygon. It performs several calculations for each vertex and pixel rendered, and establishes the way textures are combined. A texture is a bi-dimensional image that represents the intensity distribution of a component over the polygons‟ surface. It is characterized by its resolution and the number of bits assigned to each sample. The relationship between a texture and the way it is applied on a polygon is called texture mapping. Texture mapping information is stored in vertices and, by allowing multiple mapping channels per vertex, different textures can be applied to the surface with different criteria.

In order to optimize graphical memory usage, textures are supposed to be squared, and their resolution has always to be a power of two. Also, for the user to be able to decide the visualization quality based on its computer/network performance, and to improve simulation frame rate with the refining algorithm, textures will have 3 different levels of detail: 128x128 pixels, 256x256 pixels and 512x512 pixels.

Two main components have to be defined for each material: the diffuse channel and the lightmap channel. The first one indicates the surface color and the second one stores the lighting information for each polygon that uses the material. Additional components can be added in order to represent specific surface properties, like reflecting, shiny, or transparent zones. Figure 1 illustrates the effect of blending different texture channels in the same model.

Lightmap importance resides on the simulation's visual richness: lighting calculation is mandatory in order to create a realistic environment. However, the computational cost of realistic lighting simulation can compromise the application performance. To solve this problem, lighting information is pre-calculated for each model, and stored in a single texture [18] that defines how dark or bright the surface of each polygon will be. Lightning information can also be stored in mesh vertices, but this technique presents problems due to geometry subdivision [19].

Separating textures and shaders in materials is very important in order to optimize network and graphic memory usage: too many textures can produce memory overflows and seriously affect the performance. To minimize the number of textures requested to the server and loaded in memory, generic diffuse textures will be used. Thus, once a building and its textures are loaded into the application, the next building to be downloaded can share previously loaded textures, so it will not need to consume new resources. The larger the environment is, the higher number of textures will be shared. Notice that lightmap textures cannot be shared, since they store specific information for each polygon.

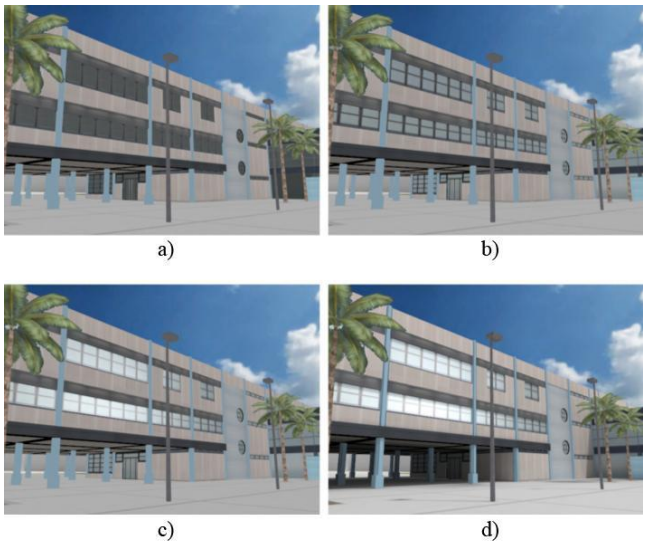

Figure 1. Incremental results when blending additional components. (a) Only diffuse component. (b) Addition of a pre-calculated sky reflection and reflection masks. (c) Addition of the specular component and specular masks. (d) Addition of a second channel pre-calculated lightmap.

Since the goal of the application is providing information related to the environment simulated, one last element has to be introduced: the 'Point Of Information' (POI). Points of information are intended to be visual entries of information, distributed along the environment, and hierarchically sorted in a navigation menu. By clicking them, users can have a quick access to detailed information about the place they are in. Basically, a point of information is defined by its position in the environment, a short text that identifies it, and a texture to represent it as an icon in the simulation.

Additionally, in order to filter information and differentiate the importance of each point of interest, their visibility and size in the simulation is determined by their distance to the viewpoint. This way, only information related to the place the user is visiting will be shown, and the most important POI will be bigger and more visible in long distances.

B. Data storage model

Once the main elements of the 3D simulation and their relationships have been introduced, it is necessary to define the way to store them, and how the information will be linked in order to create an information-oriented 3D simulation. Established relationships have to be considered when defining an architecture to store all models, and query responses have to be deterministic in the sense that results cannot be ambiguous. Moreover, data storage has to support on-demand streaming transfers, respecting proposed LOD technique, in which meshes and textures are independent from objects and materials respectively.

To achieve this, meshes (with vertex, polygon and material definitions) are stored in binary compressed files. In a simulation with n different levels of detail for each object, there will be n independent mesh files.

Each object has an entry on a database, indicating the location of the files that contain the different LOD meshes, and some basic information to be displayed in the simulation, like the name of the building, the color of its zone, the location the viewpoint should be placed in case the user double-clicks the building…

Textures are stored in JPEG compressed images, as this format offers the best compression ratio, having a different file for each LOD and each channel. Each material has a base-name assigned in order to reference its texture files. Texture filenames are compound by a base-name and two suffixes, intended to differentiate LOD parameters and provide channel information. This way, a texture file called 'wall_512_d.jpg' is used by materials with a base-name 'wall' when the LOD algorithm decides that 512x512 pixel textures have to be used, and the texture represents the diffuse component, because of the '_d' suffix.

The shader to be used by the material to blend textures and compute the final appearance of objects is determined 'on the fly' based on the available channel textures matching the base-name, and current LOD. This way, there is one different shader for each possible allowed combination of channels: diffuse + lightmap, diffuse + lightmap + alpha, diffuse + lightmap + reflection mask…

Points of interest are stored in a database, indicating the location of the texture that represents them, their location on the 3D world, size and visibility parameters, and some basic information to be displayed during the simulation, as for instance their label text.

In order to hierarchically organize contents in a navigation menu, there is a table including one entry for each menu item. This entry stores information about its label text, the parent item in the hierarchical tree, a rank to establish the order on its level, and a link to the POI or building that the menu item represents. This way, a quick access to contents and locations in every simulation is provided.

Basic information of interactive objects is stored in their own database entries. However, extended information is treated differently: since HTML language capabilities exceeds 3D engine capabilities on displaying text/multimedia contents, detailed data is stored in an independent database in HTML format, and is shown on a separate frame in the application, rendered by the browser as a webpage.

Extended information and simulation interactive objects are connected through a reference in object and POI entries in the database. Each time a user asks for related information on some building or POI, this reference is requested to the database, and the result is displayed out of the simulation frame. This way, non-3D contents, can be added or edited by administrators in a back-end application through a HTML content editor. Menu items also provide access to extended information implicitly: when a user clicks a menu item, the associated POI or model is queried, and the link to the related information is returned.

Figure 2 illustrates the data storage model purposed that respects the content structure explained and the implemented LOD algorithm restrictions.

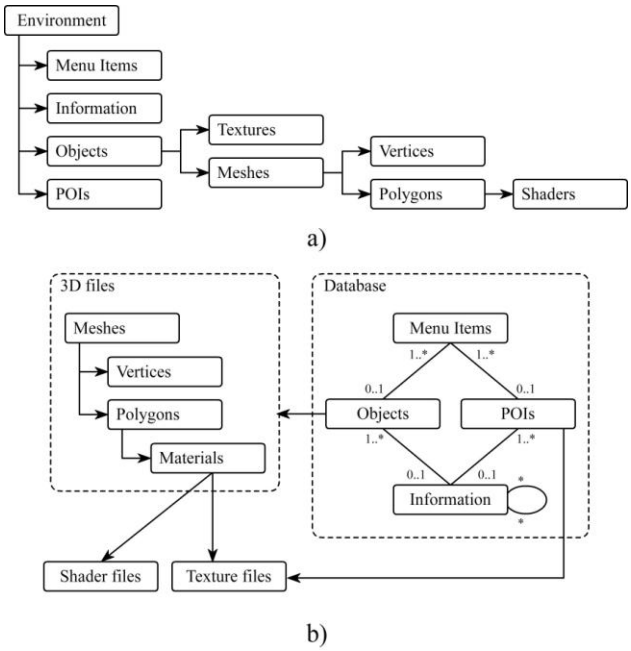

Figure 2. Data storage model purposed. (a) Conceptual model. (b) Purposed storage model.

III. Network architecture and communication process

Since we work in a networked environment, the architecture of the application is based on a client-server paradigm with a Software Delivery strategy [20]. There are four main components that play specific roles [21]:

- Application server: stores 3D data, information and the application code, and transfers them to the clients. Since it serves to several clients, it does not provide complex computations for individuals. Instead of this, the server sends the application code and the data to the client [20], in order to generate the real-time simulation. Therefore, the computational power is provided by the end-user.
- Player server: stores the binary files that perform the web player installation on client computers. Since this web-application requires direct access to graphic hardware in order to perform a fast and complex simulation, clients have to install a player in their browsers that loads the code stored in the application server, and performs the simulation. Differentiating the application code and the player allows that, with only one installation, clients can load lots of different applications. Also, multiplatform compatibility is implemented in the player, making the application code platform-independent. In addition, bug fixes and maintenance tasks are simplified with this architecture.
- Web client: in charge of the real-time rendering, interaction with the user, and all data requests to the server. First application run requires a player installation through the player server.
- Network: responsible for data transfers between client and server. Because of time delays, interactions between the user and the 3D environment have to be performed primarily on the client side, keeping this client-server communication only for data requests.

Figure 3 illustrates the network architecture purposed.

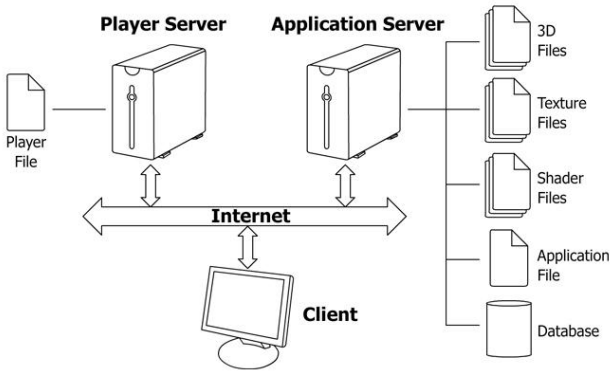

Figure 3. Network architecture

Communication process starts with a client request to the application server. First data transferred is the application code. However, if the application server detects that the client does not have a suitable player, or it needs to be updated, redirects the client to the player server. After the player has been correctly installed, the client downloads the application code.

Since communication is always started by the client, and data is downloaded on demand, first thing a client needs to know are the available resources stored in the application server. A query to the database has to be made in order to find out which objects conform the simulation, and texture filenames have to be parsed to be able to decide, in real time, which shaders to apply to different materials.

Once resources are located, simulation can start. In our approach, considering a large environment, first, client has to download the lowest LOD of all objects, or a reduced set compound by the closest ones to the viewpoint. Notice that *closest objects* can be estimated without additional downloads, since this information is stored in object's database entries. Depending on viewer's position, objects will be sorted according to the distance, in order to request additional data, to progressively refine the meshes and textures of the closest objects. Also, high detailed buildings that are too far from viewpoint have to be simplified with a lower detail LOD to prevent graphic memory overflows or low frame rates.

The need for getting continuously distances between the viewpoint and objects can seriously affect performance if the process is not simplified: exact point-object distance calculation is extremely inefficient, since it is necessary to intersect a ray with each polygon for each object loaded in the scene. An alternative would be to assume that the distance could be approximated as the Euclidean distance between the viewpoint and the barycenter of the studied object, obtaining a minimum computational cost in the sorting operation. The problem with this technique is that, due to disparity in size and shape of objects, big measuring errors will be committed that will lead to a bad visibility order.

As a compromise, trying to balance computational performance and quality of results, distances are approximated by using ray-cube intersections (assuming a cube as the bounding box that contains an object). Using this method, each time the distance between the viewpoint and an object is calculated, a ray from the barycenter of the object towards the viewpoint is created, and then intersected with the bounding box of the studied object. Point-object distance is approximated as the Euclidean distance between the computed intersection point and the viewpoint position. This technique considerably reduces computational costs from the first proposal, and minimizes the error committed in the second proposal. Figure 4 illustrates the purposed technique.

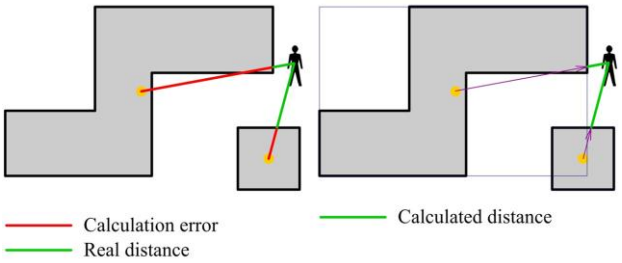

Figure 4. Distance calculation. (left) viewpoint/barycenter distance. (right) viewpoint/bounding-box distance.

Object re-sorting is not necessary every frame: it can be assumed that, in a single frame, the viewpoint will not change significantly. Thus, by introducing a new parameter to control the necessary displacement of the viewpoint in order to re-order the objects, processor's usage will be considerably reduced. Once objects are sorted, a downloading list is generated considering closest objects to the viewpoint with a greater priority. Because of low transfer rates of the network, all objects cannot be downloaded while sorting. Instead of this, *n* downloading threads are created when the application starts. These threads are in charge of downloading and processing in parallel only high priority objects from the sorted list. The maximum number of high-detail objects is established as a parameter, while the number of downloading threads can be estimated according to a compromise between models to be loaded and network time latency.

As the viewpoint is continuously changing its position, the sorted list is re-computed and some active downloading threads can lose its priority. In this case, the process is not interrupted because otherwise we could face a situation in which no object download will conclude. Instead of this, active downloading threads always finish their process, so non-used downloaded objects will remain in system cache memory.

The complete communication process is illustrated in figure 5 from client perspective.

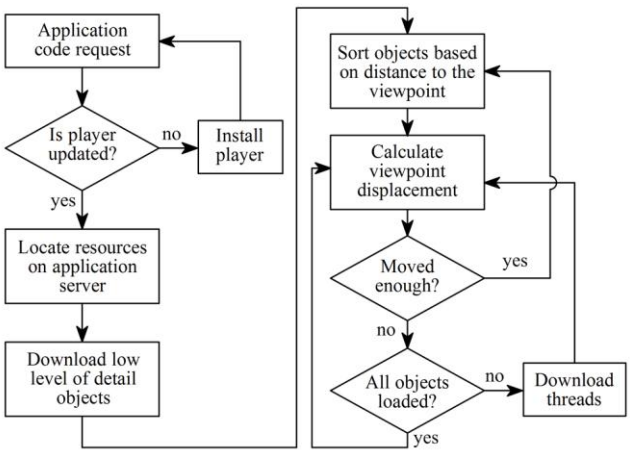

Figure 5. Network architecture

IV. Evaluation

In order to demonstrate the feasibility of the purposed technique, several traces have been made. Measures have been taken while the camera moves around the environment within a fixed trajectory, considering different quality settings. The main goal is getting information about the usage of hardware resources during the camera movement. As long as the complete model of the campus is not yet finished, a subset of 40 buildings has been chosen. As results will show, this does not invalidate the test, since new buildings will not affect significantly performance.

In order to provide realistic estimations, and considering the use of the application on domestic environments, a three year old desktop computer has been used: AMD Athlon 64 Dual Core 4200+ 2.21GHz, with 3 GB of RAM, and a GeForce 8600 GTS with 512 MB of VRAM under Microsoft Windows XP SP3. Network access is performed through a standard 4 Mbit cable connection.

There are two main system loads to measure: the initial preload of low-poly models, and the walkthrough. The first one is common to all quality settings, and is intended to provide a fast environment preview, while the second one depends strongly on quality settings and provides an interactive realistic simulation. Network usage during the preload of the environment is shown in Chart 1. Final video memory usage, after all models are loaded and displayed, is 28 MB (including GUI textures). No frame rate drop has been observed while loading the model. Notice that only 18 MB of VRAM are used to display low-poly buildings.

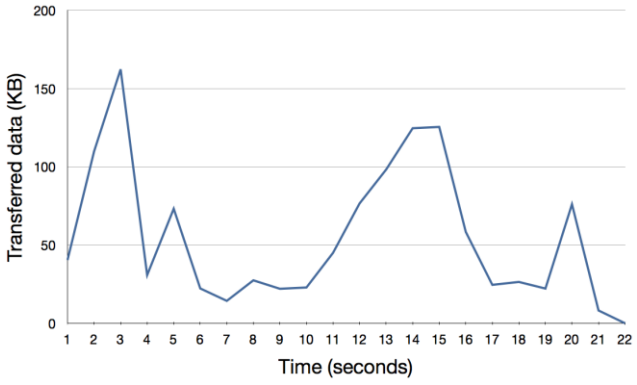

Chart 1. Transferred data during the preload.

To measure hardware usage during the walkthrough, four quality presets have been used depending on texture/lightmap resolution. Table 1 illustrates different texture sizes available, while quality presets are shown in Table 2.

Kind of texture	Resolution (pixels)	Average file size (KB)
Diffuse/Alpha/Mask	128x128	3.09
	256x256	10.63
	512x512	78.81
Lightmap	512x512	88.32
	1024x1024	258.56
	2048x2048	784.23

Table 1. Available textures, and average file size.

Name	Diffuse resolution (pixels)	Light resolution (pixels)
Low	128x128	512x512
Medium	256x256	1024x1024
High	256x256	2048x2048
Ultra	512x512	2048x2048
Ultra cached	512x512	2048x2048

Table 2. Quality presets.

The camera trajectory during the walkthrough has been defined in order to create a worst-case condition: camera moves continuously towards not loaded buildings, starts in a heterogeneous building zone where no textures can be reused, and with no object loaded. The moving speed is 11 km/h (while average running speed in a marathon in 10.32 km/h) and the trajectory length is 1 km. During the walkthrough, the eight closest buildings are always displayed with high-detailed meshes, and the rest of them are in low level of detail. This way, simulation looks realistic, and mesh refinement does not interfere with visualization.

Network usage during the walkthrough can be seen in Chart 2. Notice how *Low* and *Medium* presets have a lot of idle periods, while *High* preset has only a couple of them. *Ultra* preset is downloading data continuously. This means that building refinement will be delayed in a continuous worst-case camera movement. To improve this result, user movement speed could be limited, models could be simplified, or *Ultra* preset could be disabled. However, the commented situation will happen just once: *Ultra Cached* preset shows the network usages with *Ultra* preset, when models have been visited previously. Each time a model and textures are downloaded, web browser stores downloaded files in cache memory. While user does not delete cache files, transferences will be only to check if files have changed in the server. This way, future executions will no longer need such a high bandwidth usage.

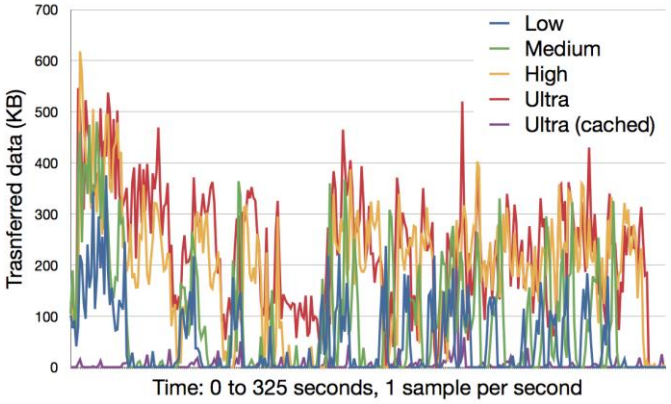

Chart 2. Network transferred data during the walkthrough.

To evaluate the quality settings effect on graphical hardware, video memory usage (VRAM) and frame rate have been measured during the purposed walkthrough. Chart 3 illustrates VRAM usage, while Chart 4 illustrates the average number of frames per second.

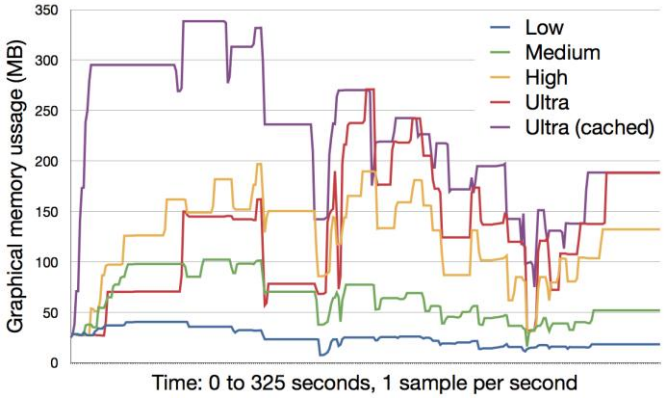

Chart 3. Memory usage during the walkthrough.

Notice how memory is released after object's visibility test fails. This way it can be assured that memory usage will be limited: if all high-detailed objects were hidden, only 28 MB of VRAM will be used. It is also important to point out that *Ultra* preset memory usage is lower than *Ultra Cached*, because not all objects could be loaded during walkthrough due to the network bottleneck.

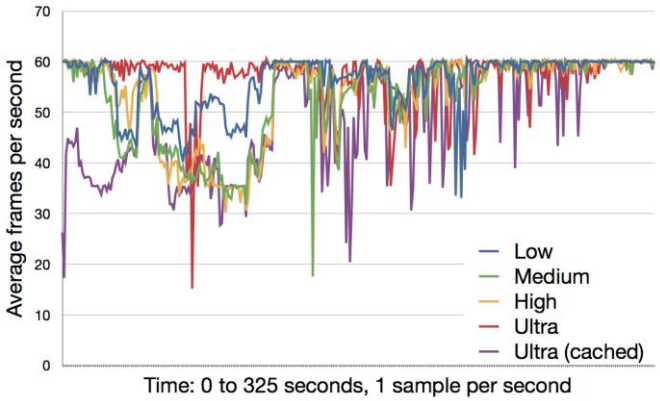

Chart 4. Framerate during the walkthrough.

Observing Chart 4 it can be appreciated that frame-rate keeps almost all the time in closer values to its hardware limit (60 frames per second), and only a few times is below 25 frames per second. This means that the GPU load is reasonable, and can be handled by the processor.

It is important to remark that the first third of the walkthrough is a very special case, since no object is already downloaded and eight models have to be requested. Also, as we intended to simulate the worst case execution scheme, the camera starts moving in the most complex zone of the campus, where buildings have more polygons and no textures can be reused (as illustrated in Chart 3). On the other hand, final third of the walkthrough is also in a complex polygon environment, but buildings are more homogeneous, lots of textures have been downloaded, and so lots of them can be shared. This way, memory usage and simulation speed show a remarkable improvement. The larger the environment, and the longer the user navigates through it, the better the results achieved.

As happened with VRAM usage, *Ultra* preset results are not representative, since not all objects could be loaded during the walkthrough. *Ultra Cached* preset provides more accurate results.

Chart 5 shows how textures are reused while the camera moves around the environment. Notice how first buildings cannot reuse previously downloaded textures, but how the next ones gradually increase the percentage of shared textures.

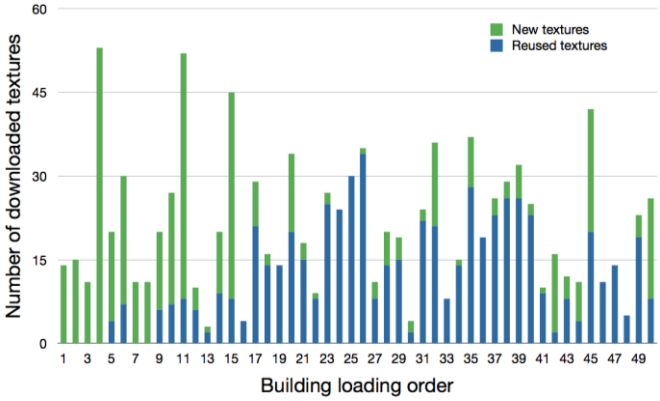

Chart 5. Texture sharing between models.

Figure 6 shows achieved results using *Medium* quality settings. Notice the highly detailed models and textures, and the realism achieved.

Figure 6. Achieved results using *Medium* quality preset.

The use of resources when generating these images is minimum, and can be handled by almost every home computer: maximum video memory never reaches 100 MB, average frame-rate values are between 60 and 35 frames per second, and bandwidth usage is acceptable (with lots of idle periods)

even in first run, where all models and textures have to be downloaded. Lower hardware requirements can be achieved by using *Low* quality settings or by displaying less high-quality LOD buildings. On the other hand, higher quality results can be obtained by using *High* or *Ultra* quality presets, allowing faster computers to create a more realistic simulation.

V. Conclusion

In this paper we have presented a feasible technique to visualize large environments in a web browser application embedded. From server storage to communications, all the process is oriented to a progressive refinement of the models based on the viewpoint position. A low cost algorithm to sort objects and demand data to the server has been introduced in order to reduce performance impact of intermediate calculations.

Evaluation results show the scalability of the technique due to a limited resource usage, independent of the complexity and size of the environment. Even in the worst case, the results are significantly acceptable.

Also, quality customization options introduced, allow the application to adapt simulation requirements to hardware restrictions by increasing/decreasing texture resolution and amount of high LOD buildings displayed. This way, powerful computers will provide an extremely detailed simulation, whilst low-power computers will provide proper results, allowing users to browse the entire campus.

The main bottleneck has been located in data transfers but, by using browser's cache memories, performance has been considerably improved, reducing transfers to a simple check of file changes in the server, after first execution.

Because of time restrictions, only two levels of detail have been used for buildings. Adding a third one will reduce considerably resources usage, and more accurate simulations could be performed. The next improvement of the technique will be focused on getting a multi-level LOD.

Acknowledgment

The authors want to thank the Vice-Rector for Information and Communications Technologies, the Information Area, and the Information Systems and Communications Area (ASIC) of the UPV, and, in particular, to Carlos Turró, for their kindly support.

References

[1] P. Heinzlreiter, G. Kurka, J. Volkert, "Real-time Visualization of Clouds", *Proc. WSCG'2002,* pp. 43-51, 2002.

[2] W. Lifeng, L. Zhouchen, F. Tian, Y. Xu, Y. Xuan, K. Sing Bing, "Real-time Rendering of Realistic Rain", *ACM SIGGRAPH 2006 Sketches*, Article No. 156, 2006.

[3] A. Poliakovet al, "Server-based Approach to Web Visualization of Integrated Three-dimensional Brain Imaging Data", *Journal of the American Medical Informatics Association,* Vol. 12, 2, 2005.

[4] S. Rusinkiewicz, M. Levoy, "Streaming QSplat: a viewer for networked visualization of large, dense models", *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, pp. 63-68, 2001.

[5] P. Morillo, M. Fernández, N. Pelechano, "A Grid Representation for Distributed Virtual Environments", *Grid Computing*, Lecture Notes in Computer Science Vol. 2970/2004, Springer-Verlag, pp. 182-189, 2004.

[6] L. Chittaro, R. Ranon, "Adaptive 3D Web Sites", *P. Brusilovsky, A. Kobsa, W. Nejdl (eds.), The Adaptive Web - Methods and Strategies of Web Personalization, Lecture Notes in Computer Science*, Vol. 4321, pp. 433-464, 2007.

[7] O. Devillers, P.M. Gandoin, "Geometric compression for interactive transmission", *Proc. of the Conference on Visualization'00*, pp. 319-326, 2000.

[8] R. Estkowski, J.S.B. Mitchell, X. Xinyu, "Optimal Decomposition of Polygonal Models into Triangle Strips", *Proc. of the 18th annual symposium on Computational Geometry (SoCG''02)*, pp. 254-263, 2002.

[9] S. Guthe, et al, "Interactive Rendering of Large Volume Data Sets", *Proceedings of the Conference on Visualization'02*, pp. 53-60, 2002.

[10] L. Ming et al, "Online Accelerate Rendering of Visual Hulls in Real Scenes", *Journal of WSCG,*Vol 11, pp. 290-297, 2003.

[11] N. Tamuraet al, "A Practical and Fast Rendering Algorithm for Dynamic Scenes Using Adaptive Shadow Fields", *The Visual Computer: International Journal of Computer Graphics,* Vol. 22, Issue 9, pp. 702-712, 2006.

[12] J. Zara, "Virtual Reality and Cultural Heritage on the Web", *Proceedings of the 7th International Conference on computer Graphics and Artificial Inteligence*. Limognes, France, pp. 101-112, 2004.

[13] J. Zara, "A Scaleable Approach to Visualization of Large Virtual Cities", *Fifth International Conference on Information Visualization*. London, UK, 2001.

[14] L. Thomas, "Spacing Out: Web 3D and the Reconstruction of Archaeological Sites", *Ancient Studies – New Technology and Scholarly Research,* Newport, 2000.

[15] DASSAULT SYSTEMES. *3DVIA Virtools*. http://a2.media.3ds.com/products/3dvia/3dvia-virtools.

[16] C. RIKK, B. GAVIN, M. CHRIS, "The Virtual Reality Modeling Language Specification", *The VRML Consortium Incorporated*, 1997.

[17] Y. Takase et al, "A Development of 3D Urban Information System On Web", *Proceedings of the International Workshop on Processing and Visualization using High-Resolution Images,* ISPRS, 2004.

[18] R. Bastos, M. Goslin, H. Zhang, "Efficient Radiosity Rendering using Textures and Bicubic Reconstruction", *Proc. of the 1997 symposium on Interactive 3D Graphics*, pp. 77-ff, 1997.

[19] N. Ray et al, "Generation of Radiosity Texture Atlas for Realistic Real-Time Rendering", *Eurographics 2003,* 2003.

[20] K. Brodlie, “Visualization over the World Wide Web”, *Scientific Visualization (Dagstuhl ’97 proc.)*, IEEE Computer Society Press, pp.23-29, 2000.

[21] J. Zara, “Web-Based Historical City Walks: Advances and Bottlenecks”, *PRESENCE: Teleoperators and Virtual Environments.* Vol. 15, No. 3, pp. 262-277, 2006.

Author Biographies

Eduardo Vendrell was born in Carlet (Spain), 1967. He has a Ph.D on Computer Science from the Universitat Politècnica de València, UPV (2001). From 1991, he is teaching at UPV in the field of Robotics and CAD/CAM. Currently, he is associate professor in the School of Computer Science. His research interest is related to CAD/CAM and geometric modeling, having published different papers on international conferences and journals. He is member of the Institute of Control Systems and Industrial Computing (ai2), UPV. From 2009 he is also the Dean of the School of Computer Science.

Carlos Sánchez was born in Valencia (Spain) on February 22nd, 1981. He was graduated as I.T. engineer with “Industrial Computing” specialization in the Universitat Politècnica de València (UPV), Spain in 2009. He completed his Master’s degree in robotics in the University Institute of Control Systems and Industrial Computing (ai2) in the same university in 2010. Currently, he is a Ph.D student, working in graphics programming research.