Galactica, a Digital Planetarium that explores the Solar System and the Milky Way

Jorge d'Alpuim
ISCTE – Instituto Universitário de Lisboa
ISTAR-IUL, Lisbon
jorge_alpuim@iscte.pt

Miguel Sales Dias
Microsoft Language Development Center, Lisbon
ISCTE – Instituto Universitário de Lisboa
ISTAR-IUL, Lisbon
miguel.dias@microsoft.com

Abstract

This paper describes a new Digital Planetarium system that allows interactive visualization of astrophysical data and phenomena in an immersive virtual reality (VR) setting. Taking advantage of the Cave Hollowspace at Lousal infrastructure, we have created a large-scale immersive VR experience, by adopting its Openscenegraph (OSG) based VR middleware, as a basis for our development. Since our goal was to create an underlying system that could scale to arbitrary large astrophysical datasets, we have splitted our architecture in offline and runtime subsystems, where the former is responsible for parsing the available data sources into a SQL database, which will then be used by the runtime system to generate the entire VR scene graph environment, for the interactive user experience. Real-time computer graphics requirements lead us to adopt some visualization optimization techniques, namely, GPU calculation of textured billboards representing stars, view-frustum culling with octree organization of scene objects and object occlusion culling, to keep the user experience within the interactivity limits. We have built a storyboard (the "Galatica" storyboard), which describes and narrates a visual and aural user experience, while navigating through the Solar System and the Milky Way, and which was used to measure and evaluate the performance of our visualization acceleration algorithms. The system was tested with an available dataset of the complete Milky Way (including the solar system), featuring 100.639 textured billboards representing stars and additional 104.328 polygons, representing constellations and planets of the solar system. We have computed the frame rate, GPU traverse time, Cull traverse time and Draw traverse time for three visualization conditions: (A) using standard OSG view frustum culling technique; (B) using view frustum culling with and our octree organizing the scene's objects; (C) using view frustum culling with our octree organizing the scene's objects and our occlusion culling algorithm. We have generally concluded that our octree organization and octree plus object culling techniques out-performs the standard OSG view frustum culling, when around half or less than half of the dataset is in view of the virtual camera.

Keywords

Virtual reality, CAVE, Lousal, Immersive Virtual reality, octree, occlusion culling, astrophysical, Solar System, Milky Way, Hipparcos, Openscenegraph

1. INTRODUCTION

If we approach Immersive Virtual Reality (VR) from the perspective of its requirements for real-time 3D Computer Graphics, we certainly conclude that we need to put in place in our VR system, visualization acceleration techniques in order to provide the user with real-time interactive experiences, within an environment capable of depicting photorealistic scenes in stereoscopy, in higher resolution multi-display projection systems, such as the one available at the CAVE Hollowspace of Lousal (CaveH) [Soares10], in southern Portugal (near the city of Grândola) and used in this work. There are in fact available a quite wide range of different approaches to accelerate the processing of large and complex datasets, depending on the kind of dataset we are willing to represent. In this paper, we are especially interested in visualizing large datasets of astrophysical celestial objects, such as the ones included in the Digital Universe Atlas (DUA), assembled by the American Museum of Natural History (AMNH) [Abbott02]. This catalogue features a high-precision dataset with 100.639 stars, which was sensed by the Hipparcos scientific satellite of the European Space Agency (ESA), launched in 1989 and operated until 1993. Hipparcos, was targeted to precision astrometry and provided the community with accurate measurement of the positions of celestial objects. DUA is a multi-dimensional atlas intended to represent the known universe and used as a basis for further data analysis and exploration. In this framework, we present in this paper, a new interactive Digital Planetarium tool for the CaveH at Lousal, able to paint with the opportunity to experience the visualization and interactive navigation across arbitrary large astrophysical datasets, such as the DUA. Displaying an arbitrary large dataset of rich 3D content in an Immersive VR setting with high resolution and real-time stereoscopic visualization requirements, relying solely on the Core CPU power, without any specific 3D acceleration technique in mind, would naturally compromise the user experience. We could easily break the real-time interactivity performance, since the time taken to process the entire scene complexity would become noticeable. Visualization acceleration techniques are meant to help decreasing the CPU effort to process and draw the geometry in each frame, by using appropriate scene organization techniques, real-time visualization technique and leveraging the power of the available GPU Cores (in the order of hundreds) to unload the burden of the few existing CPU Cores, in the computing system. In this context, the identified real-time 3D Computer Graphics requirements, namely, a scene graph to describe the 3D environment, a volumetric organization technique of the scene, an object culling approach and GPU rendering of most of the celestial objects, where matched by the adoption of the Openscenegraph (OSG) [Osfield05] C++ development environment, specially targeted to the processing and visualization of large and complex 3D environments (in terms of polygon count), which has already some built-in acceleration techniques. In addition to OSG, we have used the CaveH middleware [Dias07] [Soares10], an in-house developed logic to produce and manage 3D content in the multi-display Cave Automatic Virtual Environment (CAVE) at Lousal, an immersive large-scale VR system much like the one proposed by [Cruz-Neira92] in 1992. Since the CaveH supports projection in many different computing architectures (from tablet and laptop displays to large-scale multi-projection planes) and it is based on the OSG graphics platform, which is a layer above the OpenGL library, we adopted the same development environment in order to avoid later unexpected compatibility issues. This paper presents also the results obtained with the Galactica application that provides the user with an experience of a space travel across the Solar System and its surrounding galaxy (Milky Way). We provide a comparison between the OSG standard pipeline and our customized 3D rendering pipeline, with our visualization acceleration techniques embedded in the OSG rendering traversal cycle, specifically developed to support the requirements of Galactica.

The paper is structured as follows. In the section 2, we describe in detail the datasets used by Galactica. The third section details the underlying system architecture, responsible for managing, simulating and running the application. Section 4 explains the overall VR simulation and visualization pipeline, by showing how the data is represented and explaining in detail the real-time 3D algorithms developed in order to achieve real-time performance. Section 5 presents and discusses the results on the measured performance metrics obtained by simulating the Galatica application using the different acceleration algorithms. In Section 6 we extract some conclusions and provide some ideas for further research. We also acknowledge our thanks to all persons that directly or indirectly helped achieving the Galatica application results.

2. THE DIGITAL UNIVERSE ATLAS DATASET

As mentioned before we used the DUA dataset, loading only the data of the Milky Way subset (corresponding to the Hipparcos sensed data). We were willing to provide an application to navigate through the Solar System's astrophysical data and phenomena and the Milky Way. Since the DUA lacks on specific Solar System data, we gathered the complementary data from another publicly available dataset contained in the Celestia [Laurel01] software package. As a result, the data depicted in the Galatica application comprise planets, stars and polylines, this last ones describing the constellations and the planetary orbits. Using the above mentioned data sources we were able to paint the following objects:

Planets: The Solar System's planets are drawn as spheres with their surface textures mapped into it, simulating an approximation of the real planet's shapes. The used surface textures were taken from [Hastings-Trew00], an online resource providing free high-resolution images to the scientific community. The planets are represented by their appearance, dimensions, orbital lines and rotation's axis. The planet's orbits are also present in our VR simulation and will be drawn by polylines, with a color assigned, as a sequence of points around the Sun, forming the elliptical planet's orbits.

Stars: Representing the stars is a complex task due to the fact that, when a star is quite distant from the viewer, we only perceive it as a shiny small point in the sky and this effect is computationally difficult to reproduce. Therefore, our stars' representation, are an approximation of the visual effect our eye can perceive, which is a point with a colored halo around. That being said, we decided to use a texture of a hallo provided by the DUA (Figure 1), transform it into a hallo transparency texture and map it into a colored surface. The corresponding surface is a quadrangle 3D sprite, simulating the billboard effect, in order to ensure that the star's hallo is always facing the virtual camera that represents the viewer pose (position and orientation).

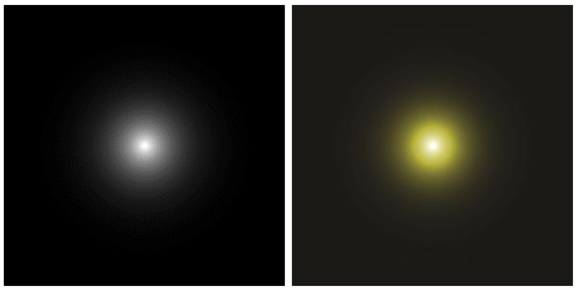

Figure 1: [Left]: Original halo texture given by DUA. [Right]: Simulation of a yellow star halo effect by mapping the halo transparent texture into a yellow surface.

The left image present in Figure 1 is a full opaque image of a white hallo, which was the basis to generate the transparent mask image created to allow representing colored hallows (Figure 1 – [Right]), where its black pixels represents the transparent pixels.

Figure 2: Landscape taken from our VR application depicting the Scorpius constellation, one of the Zodiac's constellation.

Drawing polylines joining the stars' position will generate the stars' constellations effect, as it can be observed on Figure 2. Since the constellations are 3D objects drawn in the VR scene, those constellations are only realistically represented when they are observed from a perspective closer to the Earth's surface. Other way, the constellations will be "deformed" as perceived by the viewer, once the viewer´s relative pose to the Earth is different.

3. VR SYSTEM ARCHITECTURE

The system built to simulate the required VR scene is divided in two: offline and runtime subsystems (Figure 3). The offline subsystem is responsible for selecting and parsing available astrophysical data and metadata, both from local and/or remote sources, into a relational database that will be read by the runtime subsystem. The runtime subsystem, builds the VR scene based on the data stored in the relational database, which was previously populated by the offline subsystem.

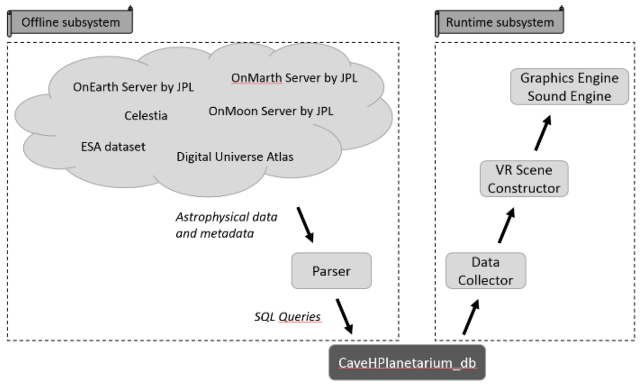

Figure 3: Overall VR system architecture, described with two subsystems.

This logical split made on the overall system brings scalability advantages to the application due to its ability to visually simulate different datasets of arbitrary sizes.

4. VR SIMULATION AND VISUALIZATION PIPELINE

As it was stated before, our 3D development environment was based on CaveH, which is a C++ middleware that builds on the OSG scene graph and 3D rendering SDK. An intrinsic feature regarding OSG, is that it will organize all scene objects in an acyclic graph, in order to efficiently traverse all scene's objects in each 3D visualization pipeline traversal, comprising 4 traversal passes per frame: an event, an update, a cull and a draw pass in each frame [Wang10]. This order should be respected in order to keep the data consistency and to prevent unexpected behaviors. The event traversal, is the one responsible for collecting user inputs, such as keyboard and mouse operations, which is extremely important in an interactive application, due to its ability to provide the application, the possibility of reacting to user inputs. The update traversal, will inspect all the scene's objects and will transform them as a callback response to the user's inputs and/or by their own geometric transformation parents, in case of dynamic objects. Cull traversal, will filter which objects are seen/not seen by a virtual camera and therefore, which ones should be processed and drawn, by different metrics, such as their visibility, their importance in final image, their translucency, etc, preventing irrelevant objects to keep being processed in further traversals. Object that are elicited to be drawn in the current frame, are added to a "Draw List". The last traversal, the draw traversal, will process objects in the internal "Draw List", to be sent to the display, directly creating the OpenGL calls that will create the rendered scene. The DUA's Milky Way subset, our main testing dataset, comprises 100.639 stars which will be represented by textured billboards, each one with unique characteristics (taken from the dataset metadata), hence it may lead to performance issues if we do not take any pipeline precaution, limiting the amount of data traversed and drawn, in every frame. As stated by [Akenine-Möller08] "Acceleration algorithms will always be needed" in any VR application. Our visualization acceleration tackled, in particular, the object's culling task. As a conclusion, our algorithms will operate during the pipeline's culling traversal.

OSG already provides by default some culling algorithms such as the "small feature culling", which clips away the objects that will contribute with only few pixels in the rendered image. Although, since we want to draw stars in the night sky we need to draw even the smaller objects, and so we disabled this small feature culling approach. Another important culling approach already built-in the OSG pipeline is the standard "view frustum culling" algorithm, which clips away all the objects not seen by the virtual camera, by intersecting each object's bounding volume against the virtual camera's view frustum (that is, the truncated visualization pyramid). We will be using it in our approach because it will prevent all the invisible objects from being drawn, reducing the number of OpenGL calls which would reduce the time taken to draw the scene. In our application we adopted the standard octree approach, as our spatial organization data structure. This technique divides the 3D space in a regular way, which allows rapid building of the entire tree and provides an efficient and fast way to search for all the visible objects, stored in such octree data structure. Another real-time acceleration algorithm we found useful to include in our approach is the occlusion culling algorithm, which clips away the occluded objects, this is, the algorithm will test if any object inside the view frustum is hidden behind another object or a set of objects and, in that case, will also clip it away.

4.1 Data representation

Choosing an optimized way to represent the data is an important part of an efficient real-time VR application. As it was stated previously, we represented each star as textured billboards and the planets as textured spheres. Drawing each star as an independent billboard, using the default billboard's OSG object, implies high computational demand, breaking the application's interactivity (see Table 1).

OSG billboard structure		
# Objects	# GPU calls	Frame rate (fps)
11	11	~ 195
100	100	~ 189
1 009	1 009	~ 150
10 221	10 221	~ 52
100 629	100 629	~ 8

Table 1: Preliminary performance results for a variable number of objects drawn in the scene, using standard OSG Billboard objects with increasing number of objects, running in mono projection mode.

Therefore, we adopted a different approach to simulate the billboard effect of each star, based on direct GPU drawing of such objects, by the mean of OpenGL's shading language programing, the GLSL. Our approach allows to draw all the stars in a single GPU call, while changing each star object pose to face the camera every frame. In order to keep each star facing the camera, while drawing each vertex of each star´s billboard (a quadrangle), we need to rotate such quadrangle by its corresponding star's pivot point.

	# Objects	# GPU calls	FPS Mono	FPS Stereo
OSG	3.934	3.934	~ 97	~ 27
OSG	100.629	100.629	~ 4	~ 2
GPU	3.934	1	~ 855	~ 462
GPU	100.629	1	~ 743	~ 396

Table 2 – Comparative table depicting the system's performance drawing all the stars by the two considered approaches (OSG or GPU Billboards), both in mono and stereo projection modes.

The vertex shader needs to receive the pivot points list because while drawing each star's billboard vertex, it will need to rotate according to its position regarding the corresponding pivot point.

As it may be observed from Table 2, drawing the stars with our GPU approach generates lower GPU calls and therefore higher performance, even when viewing all the objects present in the VR scene, which led us to realize that this was the best approach to create the star's billboard effect.

4.2 Octree spatial organization

In complex scenes with large number of objects to be processed, testing all object's visibility is a great struggle to the rendering pipeline, introducing a big delay in cull and following traversal phases. Organizing and sorting those objects in a spatial organization data structure is a common approach to reduce the number of visibility tests, boosting the time taken to process the entire scene. These data structures organize the scene objects in any N-dimensional space and typically are hierarchical, meaning that each level of the organization encompasses an arbitrary number of smaller levels which, during traversing time of the entire scene, can accelerate the quest of finding all the visible objects.

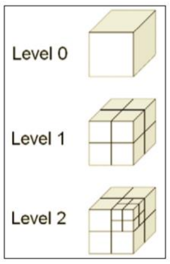

Figure 4: Octree sub-nodes division example [Wang10].

Even though these data structures may be dynamic, updating themselves while the contained objects change their state (position, size, orientation, etc…), would be time-consuming, introducing bigger delays and therefore, worse performance. In our Galactica storyboard (a travel through the Solar System), we do not consider at this stage the orbital motion of the planets and its satellites. Since in our case, the objects represented in the scene do not change position and size, we decided to adopt a static octree data structure. With this approach, traversing the entire scene during the cull phase, would reduce the number of visibility tests because when a level would not pass the test it would prevent any object, inside that level, to be tested.

The octree data structure construction [Akenine-Möller08] is performed in the beginning of the runtime subsystem cycle (see Figure 3). The octree is built based on the global scene bounding box, which is the octree's root node (see Figure 4). Starting in this node, the octree will be constructed recursively by splitting each node in two equal parts in each direction, generating eight new octants. To manage the octree's balance and its ability to be traversed rapidly, we set two rules to decide if a new octant should be created. Those rules are the maximum octree depth level and the other is the minimum number of objects per octant. All the stars stored in an octant will be drawn as part of the same OSG geometry object (with one vertex list and one polygon list), which means we will have as much OSG geometry objects, as the number of octants holding such objects.

Max. depth	**2**	**4**	**4**	**8**	**8**	**16**
Min. obj.	**128**	**64**	**128**	**64**	**128**	**128**
Pose 1	889	941	948	1128	1160	1175
Pose 2	910	926	918	1208	1190	1202
Pose 3	909	897	906	750	798	801
Max. dif.	21	44	42	458	392	401

Table 3: Frame rate measurements along three poses with incrementing number of objects built with different octree parameters. Data was captured in mono projection mode.

To create a balanced and efficient Octree, we present (in Table 3) the results achieved for different octree construction configuration. Pose 1 depicts a simulation of around 3.000 objects, Pose 2 depicts around 12.000 objects and Pose 3 depicts, the entire scene, around 100.600 objects. The table's last row represents the maximum frame rate difference between the best and worst performance pose. We set our goal to find the configuration that would bring the most balanced object's organization, in order to traverse the entire scene, with different number of visible objects, keeping the frame rate as high and uniform as possible. We may realize, observing the Table 3, that using a maximum depth higher than 4 brings lower frame rates with higher number of visible objects and bigger frame rate variations, and using a maximum depth lower than 4 brings lower frame rate performance in pose 1 because the scene is more unbalanced than with depth 4, providing slower scene traversing while observing fewer objects, which led us to choose the maximum depth of 4.

Having a maximum depth of 4 and minimum objects per octant of 64, we achieved a better performance in pose 2 than the results measured with 128 minimum objects per octant. Since this pose (pose 2) has a large amount of objects (but not the entire scene), which is the most common situation while simulating the VR application, we decided to use the following configuration to build our octree:

- Maximum depth: 4
- Minimum primitives per octant: 64

Embedding this octree organization structure with the standard view frustum culling algorithm, will prevent many visibility tests while traversing the entire scene on the quest of finding the visible nodes because, before testing each object inside an octant, the test is performed with the octant itself, and if the visibility test fails, it will prevent as many object's testing as the number of objects stored in that octant and its sub-octants children.

4.3 Occlusion culling

Our approach considered the development of an occlusion culling algorithm based in the literature review [Hansong98] [Papaioannou06], which has the benefit of bringing higher frame rate performance, by discarding occluded objects in the culling traversal phase. This algorithm has the goal of avoiding drawing and processing objects that passed the octree intersection test, after the "view frustum culling algorithm", but are hidden by other objects or groups of objects inside the view frustum, therefore we would say it could also be understood as the hidden objects culling algorithm. The occlusion culling algorithm comprises two phases. The first one, corresponds to a standard Z-Buffer technique provided by OSG, where a two-dimensional array with one element per each pixel in the display, holds a list of depth values of objects that will be mapped into that Z-Buffer element. In the second phase, we compute the distance from each object to the camera eye, based on a ray casting technique, where we trace a ray from the eye point (of the virtual camera) to each pixel in the display intersecting all the objects on the ray path, storing in the Z-Buffer only the depth of the closest object intersected.

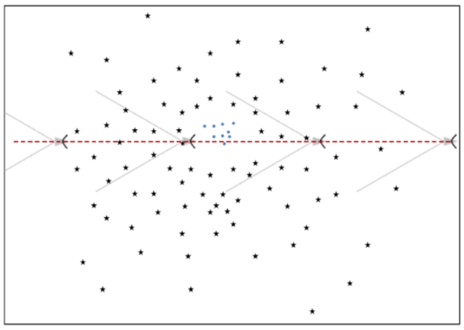

Figure 5: Top projection of the camera path across the Galatica scene, during the data collection for the performance evaluation experiment. The black stars represent the density of stars scattered all over the spatial scene, the blue circles represent the Solar System's planets and the red dashed line represents the projection of the path the virtual camera, during the simulation. In the final pose the camera includes the full dataset in its view frustum.

This way, while traversing the objects' scene we will be drawing only the objects with a Z-value lower than the value on the Z-Buffer, meaning that they are closer and therefore they are not occluded. Since our objects are organized in an octree structure, the ray casting method will test the depth value against the octants before testing the objects themselves, which will bring fewer tests by assuming that if an octant is occluded, all the objects contained within it are occluded too.

5. GALACTICA'S RESULTS AND DISCUSSION

To evaluate our system algorithms in different conditions we have created a camera animation path, crossing the whole VR scene from one side to the other passing through the scene center, where the Solar system is located. The animation lasts 30 seconds. Using this measurement approach allows us to evaluate and compare the algorithms performance in different conditions: we will start with a pose where there are no objects inside the view frustum, then the camera will be travelling backwards inside the scene, increasing the number of visible objects, until the camera reaches the opposite side, where the entire scene is inside the camera´s view frustum.

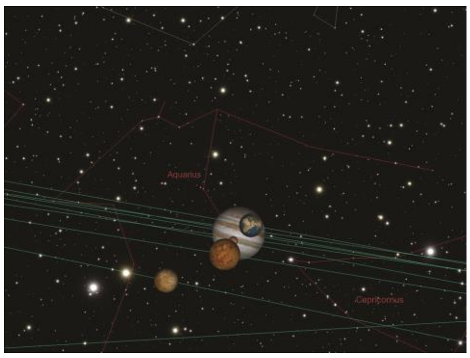

Figure 6: Screenshot taken from our Galatica application with the camera at the center of the scene viewing Solar System's planets, part of the planet's orbits and stars constellations (3D polylines linking stars).

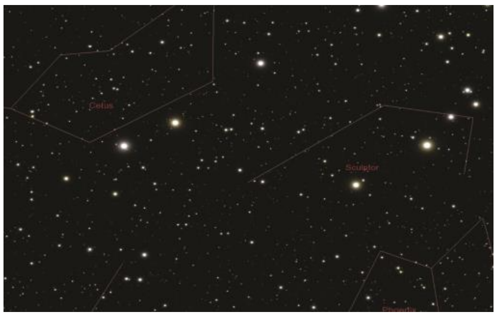

Figure 7: Screenshot taken from our Galatica application of a Milky Way's landscape.

A picture depicting the camera's path during the animation is presented in Figure 5 were we illustrate the different types and amounts of objects inside the view frustum along the camera's animation path. We may observe that this amount is increasing during the camera animation until all objects are visible. This approach will provide us a good way to compare the behavior from when there are no visible objects to when all objects are visible, and consequently need to be drawn by the graphics pipeline. Figure 6 and Figure 7 depict two different camera's poses that are available during the camera animation while the data is being collected.

5.1 Evaluation metrics

To evaluate our application and its performance we defined a set of metrics to measure the application's performance, by reading the continuous frame rate and times taken in the Cull, Draw and GPU traversals during the camera animation, which are the most relevant traversals affected by our custom algorithms. In order to validate and compare our algorithms, we simulated our application under different conditions:

1. OSG standard visualization techniques (per object view frustum culling);
2. OSG standard visualization techniques (per object view frustum), plus our Octree view frustum approach;
3. OSG standard visualization techniques (per object view frustum), plus our Octree view frustum approach and our occlusion culling algorithm.

The measurements were taken simulating the application in a PC with the following configuration:

- Intel Core i7-3970X CPU @3.5GHz
- 16 GB of RAM DDR3
- NVIDIA Quadro 5000 (2.5 GB GDDR5)
- Data set of 100.639 star objects and 104.328 polygons, representing planets of the solar system and constellations.

The measurements presented regard to a simulation in stereo projection since our application is intended to be operating in the CaveH at Lousal, which is a large-scale immersive VR system with stereo projection.

5.2 Measurements

The following diagrams depict the results measured with the metrics and conditions stated in the previous sub-section, where the vertical axis represents the frame rate and the horizontal axis, the camera's animation normalized key frames. As it may be clearly observed in Diagram 1, both Octree and Octree plus Occlusion culling achieve higher frame rates from the beginning until almost half of the entire animation, and will become inverted from that point on, which is the expected behavior.

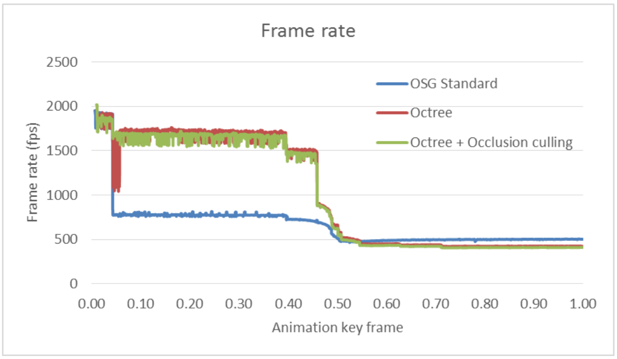

Diagram 1: Camera's animation frame rate variation in stereo projection mode.

In a first analysis, when any object hasn´t yet appeared inside the view frustum, all algorithms behave quite similarly, with frame rates around 1800 fps. This similar behavior happens due to the fact that with an octree spatial organization, the octree's root node won't pass the culling test and no other octant or nested object will be tested, resulting in only one test. Only one culling test is performed with the same visualization conditions, in a scene with the standard OSG scene graph, which won't pass the culling test as well and will prevent testing all the scene's objects. By the time the root node will pass the test, all objects will have to become tested in order to decide if they need to be drawn, and this moment happens around the 0,05 key frame, where we observe a big drop of the frame rate of the OSG standard algorithm. This big drop, in our case, has another explanation besides the delay introduced by the object visibility search tasks. As we have stated previously, all our stars are stored in only one geometry (a single OSG node) which is drawn during the GPU phase, which means that if there is any visible star inside an octant, the system will draw all the stars encapsulated in that OSG node, which is very inefficient. This phenomenon can be observed, by the large increase in the duration of the GPU pipeline's phase at the 0,05 key frame, in Diagram 2.

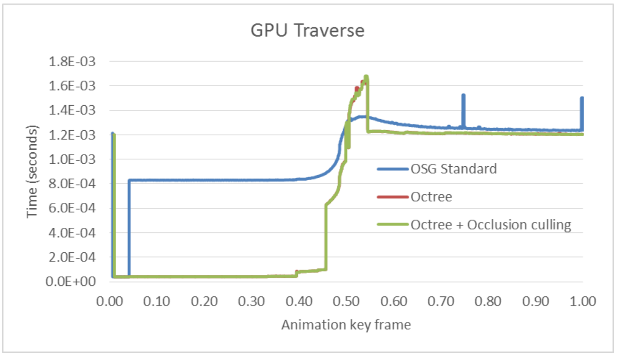

Diagram 2: Camera's animation GPU pipeline's traverse duration in stereo projection mode.

Since all the stars are embedded in the same scene graph object, with the OSG standard view frustum culling technique, the system performs only one visibility test to decide if the stars are visible or not. When the first star passes the test, all of them will be selected to be included in the cull list and therefore drawn in the respective phase, and this is the reason why this OSG standard algorithm, is the one who spends less time in the cull phase throughout the entire camera's animation (see Diagram 3).

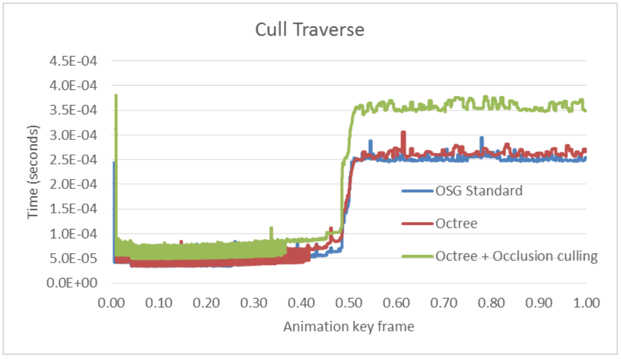

Diagram 3: Camera's animation cull pipeline's traverse duration in stereo projection mode.

From the 0,05 to 0,4 key frames we do not observe much differences in the various algorithms behaviors, regarding both frame rate or traverse duration, because the amount of visible objects is increasing at a low pace. During this period, the OSG standard algorithm average frame rate is around 750 fps. The Octree's frame rate is the highest, with around 1700 fps and the Octree plus Occlusion algorithm is 1600 fps, in average. This last algorithm spends a bit more time to search for occluded objects and this is the reason why its frame rate is a little bit lower than the Octree approach. The 0,4 key frame corresponds to the moment when the camera is approaching to the scene's center, where the smallest octants with planets are located, bringing a higher effort to compute what needs to be drawn, as it can be observed in Diagram 4. Between 0,4 and 0,55 key frames the camera is experiencing the visualization of a new kind of astrophysical objects, the Solar System's planets and their orbits, which will bring different phenomena to the different graphics pipeline's traversal phases, for all algorithms. Since the OSG standard algorithm only uses a single OSG node to represent all the stars, their only visible objects are one stars object, the planets and their orbits, which is not verified in the other two algorithms. These have the scene organized in an Octree and by this time they have a lot more data to traverse and all the new octants as well as the planets and their orbits, intersected by the view frustum. Consequently, this phase is the point where the algorithms' performances will suffer a change. In the cull phase, with OSG standard algorithm, the system only needs to traverse an object for all the stars and another per planet and orbit, whereas with the Octree approach, it needs to traverse each planet and orbit, as well as all the visible octants, which are nearly half of the total because we're in the center of the entire scene. The Octree plus Occlusion culling algorithm, will have to traverse the same as the Octree approach and will have also to test each planet for their occluded visibility, which means that this algorithm will be the one to take more time to compute the objects visibility (a small set, corresponding to just the Solar System planets), followed by the Octree approach, making the OSG standard approach the faster to process this pipeline's phase.

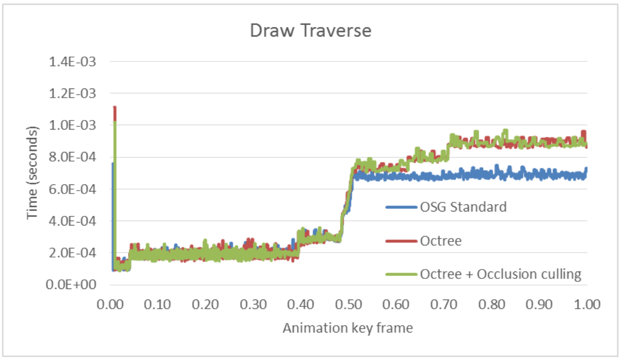

Diagram 4: Camera's animation draw pipeline's traverse duration in stereo projection mode. Regarding the draw pipeline's phase (Diagram 4), we may realize that in this phase the OSG standard approach takes less time, because it has to process less number of objects, followed by the Octree and Octree and Occlusion approaches, which behave almost the same way, since they have to process almost the same number of objects. This result shows that the occlusion calculation brings no relevant effect when the planets (occluders) are occluding each other, because they are only a few of such objects in the dataset, which is an expected result. In the GPU phase (Diagram 2), we can observe the same behavior as in the other pipeline's steps, because there are more objects to process than in the previous camera's animation phase. In this pipeline's traversal step, we may observe that there is a moment when the OSG standard algorithm take a little lower time to process the scene than the other approaches, which happens in the moment when all planets and orbits are visible, and will be followed by an inversion again, with the OSG standard to take more time to perform the processing, as the planets will become so small that they will occupy no pixels in the final image and won't be drawn. After this animation's phase and until the end of the experiment, the frame rates decrease a lot and bring better efficiency for the OSG standard algorithm. At this point the system is about to draw the entire scene and this standard technique is the faster approach given the small size of the dataset. In the worst case scenario, when everything is visible, the frame rate is in average 500 fps for the OSG standard algorithm. Using the Octree approach the frame rates are a little bit lower, around 420 fps in average. Finally, with Octree plus Occlusion culling, the frame rate is in average 400 fps. This happens because the OSG standard approach is, in this case, the one who has less objects to compute in all the pipeline's traversals.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have described the requirements, architectural elements, development and performance evaluation of a Digital Planetarium appropriate for an immersive VR environment, such as the CaveH at Lousal. Via the support and improvement of an existing VR system

(CaveH), we have developed and tested an application, referred to as Galatica, which is able to run a realistic and rich visual and aural astrophysical user experience. We have developed advanced visualization acceleration techniques to ensure that our application is efficient, runs at interactive rates in stereoscopy and is scalable to datasets of arbitrary dimensions,, as can be seen by the performance results (see Diagram 1). The application is based on a storyboard of a thematic travel through the Solar System, with visual and aural elements. The application is developed such that it can support other storyboards that might be defined for the DUA or other datasets. In fact, we have developed features to remove, update and add new datasets to the database that will feed the VR application, allowing to bring continuously diversified and up-to-date astronomical experiences to our immersive VR experience. We have computed the frame rate, GPU traverse time, Cull traverse time and Draw traverse time for three visualization conditions: (A) using standard OSG view frustum culling technique; (B) using view frustum culling with and our octree organizing the scene's objects; (C) using view frustum culling with our octree organizing the scene's objects and our occlusion culling algorithm. In our experimental analysis we observed the entire dataset of 100.639 star objects and 104.328 polygons. We have generally concluded that our octree organization and octree plus object culling techniques outperforms the standard OSG view frustum culling, when around half or less than half of the dataset is in view. This is due to the use of a balanced spatial organization structure to store the tested objects (a set of around 100K stars with their constellations and some planets with their orbital lines), preventing testing each object in the scene for its visibility. We have realized that when the virtual camera is navigating through the scene's objects, almost all the time the view frustum will hold many objects, but not the whole set contained in the entire scene, which is when our octree brings more value in efficiency, rather than the standard OSG approach. Our occlusion culling approach, in the case of our test VR scene (with the DUA dataset), didn't brought improvements on the frame rate because the only occluders present in the scene are the planets (the object occlusion culling algorithm does not apply to stars billboards) and they are only a small number of them, being only visible in a small part of the whole scene. We have taken this conclusions for the case of Galatica with a quite small planet dataset. The cases when our visualization optimization techniques aren´t more efficient than the standard OSG approach, are when the virtual camera is observing almost the entire VR scene. We expect that for larger datasets the advantage of our visualization and optimization techniques based in octree spatial organization and object culling, become more noticeable. As for future work, we expect to include more and larger data sources available in the community, which can provide more diversified types of data, like specific phenomena in an arbitrary galaxy, or the exploration of a nebulae or a black hole. We plan also to bring even more realistic and impressive landscapes to the user experience, by creating new or remodel the existing geometric models of the celestial bodies, with increasing level of detail and resolution. Although not tackled in the paper, we plan to improve the virtual navigation model by adding motion blur on the camera's movement, since it traverses many different ranges of speed. Creating new sonification cues for different types of astronomical bodies and managing how each sound would influence the user's experience, is an additional interesting direction for future research. As it can be expected, increasing the quality and resolution of the visual and aural content will bring a struggle to the performance, so a promising option is try new approaches to manage the storage of the objects. Performing the object testing at the polygon level rather than at the bounding box level, would bring a more reliable visual representation. Using the view-dependent level-of-detail approach to draw the objects might be a good direction to pursue, since it will make the draw phase to spend more effort on parts of the objects that are relevant to each view. The occlusion culling algorithm could be explored deeper, now that there are some new and faster GPU-based alternative ways of calculating the objects' visibility through occluders.

7. ACKNOWLEDGEMENTS

The authors would like to pay special appreciation to Lara Alegre and Alexandre Loureiro, that helped designing and writing the Galatica storyboard. To Mafalda Abrunhosa, a special thanks for her availability to record the narration of Galatica. To Filipe Gaspar and Vando Pereira a special appreciation for their guidance in the topics of GPU computing.

8. REFERENCES

[Abbott02] Abbott, B., Emmart, C., Marx, S. & Wyatt, R., "Digital Universe Atlas", http://www.haydenplanetarium.org/universe/, 2002.

[Osfield05] Osfield, R., "OpenSceneGraph", http://www.openscenegraph.org, 2005.

[Dias07] Dias, M. S., Soares, L. P., Varela, R., Pires, F., Bastos, R., Carvalho, N. & Costa, V., "CAVE-HOLLOWSPACE do Lousal - Princípios Teóricos e Desenvolvimento, Curso Curto", *15º Encontro Português de Computação Gráfica*, Microsoft Portugal, Oeiras, 17th October 2007.

[Cruz-Neira92] Cruz-Neira, C., Sandin, D., DeFanti, T., Kenyon, R. & Hart, J., "The CAVE: Audio Visual Experience Automatic Virtual Environment", *Communications of the ACM 35*, 1992, pp. 65–72.

[Laurel01] Laurel, C., "Celestia", http://www.shatters.net/celestia/, 2001.

[Hastings-Trew00] Hastings-Trew, J., "JHT's Planetary Pixel Emporium", http://planetpixelemporium.com/, 2000.

[Wang10] Wang, R. & Qian, X., "OpenSceneGraph 3.0: Beginner's Guide", 2010, Packt Publishing Ltd.

[Akenine-Möller08] Akenine-Möller, T., Haines, E. & Hoffman, N., "Real-Time Rendering", Third Edition, 2008, Wellesley, Massachusetts: A K Peters, Ltd.

[Hansong98] Hansong, Z., "Effective Occlusion Culling for the Interactive Display of Arbitrary Models", Ph.D. Thesis, 1998, University of North Carolina: USA.

[Papaioannou06] Papaioannou, G., Gaitatzes, A. & Christopoulos, D., "Efficient Occlusion Culling using Solid Occluders", *Proceedings of the 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, January 30th - February 3rd 2006, Plzen: Czech Republic.

[Soares10] Soares, L. P., Pires, F., Varela, R., Bastos, R., Carvalho, N., Gaspar, F. and Dias, M. S., "Designing a Highly Immersive Interactive Environment: The Virtual Mine", Computer Graphics Forum, The Eurographics Association and Blackwell Publishing Ltd., Volume 29, Issue 6, pages 1756–1769, September 2010.