Revisiting the Visibility Problem with a Hybrid Structure Paradigm

Icaro L. L. Cunha and Luiz M. G. Gonçalves
Universidade Federal do Rio Grande do Norte
DCA-CT-UFRN, Campus universitrio, Lagoa Nova, 59078-970, Natal, R, Brazil
Email: ivellius@gmail.com, lmarcos@dca.ufrn.br

***Abstract*—We revisit the problem known as the visibility problem, for computing a potentially visible set of primitives, proposing a solution that can be used to accelerate the processing in real-time 3D visualization applications. We come up with a resulting dry structure in the sense of data reduction that can be used for on-line, interactive applications. Our main goal is to load the minimum amount of primitives from the scene during the rendering stage, as possible. For this purpose, our algorithm executes the culling by using a hybrid paradigm based on viewing-frustum, back-face culling and occlusion models. Results have shown a substantial improvement over these traditional approaches if applied separately. This novel approach can be used in devices with no dedicated processors or with low processing power, as cell phones or embedded displays, or to visualize data through the internet, as in virtual museums applications.**

***Keywords*-Visibility Culling; Visualization Structure; Real-time 3D Visualization; Hidden Primitive Culling**

I. Introduction

In this work, we revisit the visibility problem, that is, the problem of computing a potentially visible set of primitives in a scene model, in the perspective of interactive and real time applications. In light of this study, we propose enhancements in the visualization pipeline model coming up with a visualization structure that minimizes the amount of useless data being loaded to generate the visualization of a given scene. We accomplish this by dividing the scene into a grid, then associating to each grid block the primitive(s) that belong to it and finally determining the visible set of primitives. As a novelty, our method makes use of the J_1^a triangulation structure [1] to create the subdivision grid. We choose this structure due to its algebraic and adaptive useful features.

In 3D graphics, hidden surface determination (also known as hidden surface removal, occlusion culling or determining the visible surfaces) is the process used to determine which parts of the surfaces are visible and, consequently, which ones are not visible from a certain point of view. An algorithm for hidden surface determination is a solution to the problem of visibility, which is one of the first major problems in the area of 3D computer graphics. The process of hidden surface determination is sometimes called hiding. The analog for lines is to render the hidden line removal. Determining the hidden surface is required to process an image properly, so that one cannot look through (or walk into) the walls in virtual reality applications for example.

Besides the advances in hardware technology and software experimented in the last decades, the visibility determination has yet being one of the main problems in computer graphics. Various algorithms capable of removing hidden surfaces have been developed over the years to solve this problem [2]. Basically, these algorithms determine which set of primitives that make up a particular scene are visible from a given viewing position. Cohen et al. [2] establishes that the basic problem is mostly solved, however, it is known that, due to the constant demand for higher amount of 3D data, algorithms such as Z-buffer and other classical approaches may have trouble while trying to guaranteeing the visualization of the scene in real time. In fact, agreeing to Cohen, one can find works in the literature such as Jones [3], Clark [4] and Meagher [5] that addressed this issue in the very beginning of CG. Nonetheless, we recall the attention to the actual importance of this problem during evolution of CG. Researchers as Airey et al. [6], Teller and Sèquin [7], [8], and Greene et al. [9], focuses on this issue and have revisited the problem of visibility to speed up visualization.

Visibility culling aims to quickly reject primitives that do not contribute to the generation of the final image of the scene. This step is done before the step of hidden surface removal is performed, and only the set of primitives that contribute at least to one pixel in the screen is rendered. Visibility culling is executed using the following strategies: back-face and view-frustum culling [10]. Back-face culling intends only to use primitives looking at (facing to) the camera display, and view-frustum culling rejects primitive located outside the visualization frustum. Over the years, efficient hierarchical techniques have been developed [4], [11], [12], as well as other optimizations [13], [14] in order to accelerate visibility culling. Besides these techniques, the occlusion culling process avoids rendering primitives that are hidden (occluded) by other primitives in the scene. This technique is more complex because it involves an analysis of the overall relationship between all primitives in the scene. Figure 1 illustrates the relation between the three known culling techniques.

Since these traditional contributions revisiting the visibility culling, as cited above, and some few further contributions [15], [16], [17], [18], not recent, which will be better explored in Section II, we could not find any up to date literature on this subject. As there are many online and real-time 3D world navigation applications (virtual museums or buildings), and

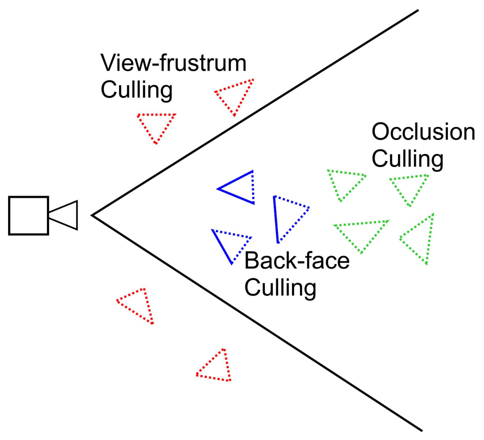

Fig. 1: Types of visibility culling techniques: view-frustum culling (in red), back-face culling (in blue), and occlusion culling (in green).

they require significant amount of primitives to be rendered in real-time, we decided to revisit the visibility problem to see if it can be even enhanced. In fact, we observe that the above techniques has some difficulties as the complexity of the occlusion culling to analyze the whole scene or the amount of computations in order to determine primitives facing sides or if primitives are inside the frustum in visibility culling methods. These are not as trivial mainly in massive data or where the scene is modeled at once. This may occur in a museum scene where there are too many sculptures for example or in outside scenarios such as a city landscape.

We overcome those difficulties by proposing a different approach introducing and adapting the above simple techniques to our context. We employ a visualization structure that is capable of obtaining a potentially visible set of primitives of the scene from the viewpoint of a user. As said above, this structure is constructed by subdividing the whole scene into a grid based on a triangulation. With this, we achieve significant improvements on the results. In particular, our method can handle this kind of data reducing the resource requirements. In the experiments, we verify a removal gain of 15% to 20% of primitives for single objects and could observe several interesting results that validate and delimit our approach. These will be better discussed during the paper and mainly in the results section.

Contributions: The main contribution of this work is the visualization structure itself that is capable of determining occlusion culling in two ways: internal block occlusion and adjacent block occlusion. The use of algebraic functions for fast access of block culling, block faces and adjacency is also another contribution that has helped enhancing the visualization process.

One other contribution of this structure is that it can also be used to determine an hierarquical order of graphics data transmition of online applications. In case of a Virtual Museum application, for example, it can be used to determine the visible set of primitives at the starting point of simulation. So, in this case, our structure can not only be used to minimise the ammount of data in the visualization stage, but also in transmition stage of online applications.

II. Related work

Since view frustum and back-face culling are mostly trivial to determine, the recent literature up to this point has mostly been focused on occlusion culling. One of the most valuable works that we found in this subject area is the one of Cohen et al. [2] that is a very useful survey. They subdivide and classify the various occlusion culling techniques developed proposing a classification of the various methods in accordance to their characteristics, as seen in the next subsections.

A. Techniques Classification

The occlusion culling methods can be classified as point based or region based methods. Point based methods execute their computations from the perspective of the view point. On the other hand, region based methods perform their computations from a global point of view that is valid from any region of the scene. Point based methods can be also classified as:

- Image Precision Methods - operate with the discrete representation of the objects when they are broken into fragments during the rasterization process. Among these are the Ray Tracing [19], [20], [21], [22], [23] and the Z-buffer [9], [24], [5], [25], [26] based methods that are the most common.
- Object Precision - use the raw objects for the visibility computation. Among these methods we can find the works by Luebke and Georges [27] and by Jones [3].

Also, region based methods can be classified as:

- Cell-and-Portal - Starts with an empty set of visible primitives and adds them through a series of portals. Among cell-and-portal based methods we cite works by [6], [8].
- Generic Scenes - Initially assumes that all primitives are visible and then eliminates if they are found to be hidden.

Cohen et al. [2] also devised other classification criteria such as: if a method overestimates the visible set (conservative) or approximates it; what is the degree of over-estimation for the conservative methods; if a method treats the occlusion caused by all objects in the scene or just a selected subset of occluders; if each occluder is treated individually or as group to be more precise; if the method is restricted to 2D floorplan or can they handle 3D scenes; if the method executes a precomputation stage; if it requires special hardware to do the precomputation stage or even the rendering stage; and if it can treat dynamic scenes.

B. Recent approaches

Bittner et al. [15] propose a from-region visibility method were rays are cast to sample visibility and use the information from each ray for all view cells it intersects. It uses adaptive sampling strategies based on ray mutations that exploit the coherence of visibility. Chandak et al. [16] propose a method that is somewhat similar to ours, their approach uses a set of high number of frusta and computes blockers for each one using simple intersection tests. They use this method to accurately compute the reflection paths from a point sound source. The method of Tian et al. [17] integrates adaptive sampling-based simplification, visibility culling, out-of-core data management and level-of-detail. During the prepocessing stage the objects are subdivided and a bounding volume clustering hierarchiy is built. They make use of the Adaptive Voxels, which is a novel adaptive sampling method applied to generate LOD models. Antani et. Al. [18] introduce a fast occluder selection algorithm that combines small, connected triangles to form large occluders and perform conservative computations at object-space precision. The approach is applied to computation of sound propagation, improving the performance of edge diffraction algorithms by a factor of 2 to 4.

C. *Contextualization*

Our method can be classified as a region based method, similar to the cell-and-portal. The only difference is that instead of a set of cells we use the scene subdivision grid and instead of a portal we use the adjacency between grid blocks. This has proved experimentally to be useful as well, at least.

Following the other cited criteria of Cohen et al. [2], our method overestimates the potentially visible set. We discuss the degree of over-estimation in Section V. Though our occlusion is block based, our method treats the occlusion created by all the blocks in groups. Our method handles 3D scenes and, due to the J_1^a triangulation structure, we believe we can add a fourth dimension for animation applications. A pre-computation stage executes without the need of a special hardware, though the use of a dedicated graphics processor would speed-up the process. And so far we treat dynamic scenes by ignoring the occlusion made by the moving objects. Because of this we do not show results based on moving objects.

D. *The J_1^a Triangulation*

The J_1^a triangulation [1] is an algebraically defined structure that can be built in any size. To accommodate local aspects, the J_1^a triangulation handles refinements naturally. Two of its main characteristics are the existence of a mechanism to uniquely represent each simplex of the triangulation and the existence of algebraic rules to traverse the structure. Using these rules prevents the structure need to store connectivity information of simplices thus enabling more efficient storage.

The J_1^a triangulation consists of a computational grid formed by n-dimensional hypercubes (blocks). Each block is divided by $2^n n!$ n-simplices that can be described algebraically using the sixfold:

$$S = (g, r, \pi, s, t, h).$$

The first two elements of S defined in that block are contained in the simplex, being a vector g n-dimensional coordinates indicating the block in a particular level of refinement r grid. Figure 2, left, illustrates a two-dimensional grid of J_1^a and, right, a block of this outstanding level of refinement grid with $r = 0$ (0-block) and $g = (3; 2)$. Also in Figure 2 it can be noticed that the blocks are darker for grid level r = 1 (socalled 1-block) and hence form part of a region of the grid of higher resolution.

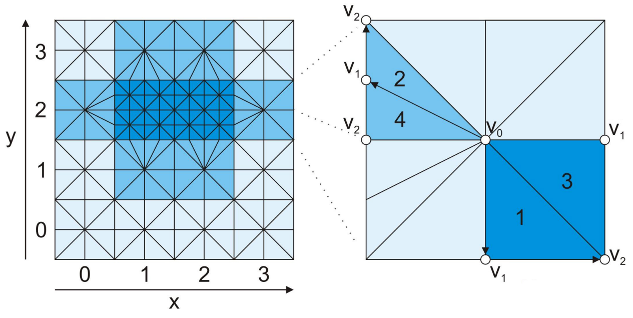

Fig. 2: 2D example for a triangulation grid J_1^a (left) and details of the block $g = (3; 2)$, $r = 0$ where it is shown two paths for tracing the block simplices.

We believe that since the J_1^a triangulation is naturally adaptive, this help in making this structure more precise when it comes to over-estimating the potentially visible set from the scene. This is because, with more refined (smaller) blocks in certain regions of the scene, more blocks are fully filled. Due to this, the adjacent block occlusion culling can be more common and thus more effective.

Since we are using J_1^a triangulation structure in this work for a purpose other than triangulating, from now on we will refer its basic grid as the J_1^a structure.

III. Technique Overview

We base our visualization structure on the algebraic structure of the J_1^a structure. The whole 3D scene is contained within the grid created by the initial J_1^a structure. From the point of view of the camera, the J_1^a structure calculates all elements visible on it. So every time that the camera is moved the visible set of primitive calculation is redone.

Our visualization structure is able to determine occlusion culling in two ways: internal block occlusion and adjacent block occlusion. The internal block culling occlusion is identified using only the primitives inside each block and it is done for every block face as illustrated in Figure 3 for face F_1.

Figure 4 illustrates the overview of our technique. Basically, it is subdivided into stages: the preprocessing and the visualization stages. In the following we better detail these stages.

A. *Preprocessing Stage*

During the preprocessing stage, we use the J_1^a structure basic grid to generate the visualization blocks.

The preprocessing stage is done in four steps:

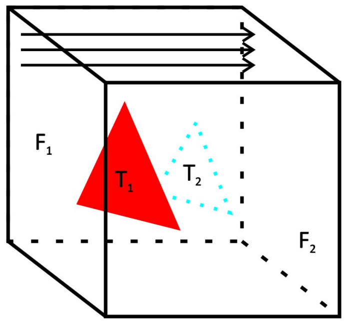

Fig. 3: Technique for determining internal block occlusion using an approach similar to Z-buffer. When verifying the visible set of Triangles for face F_1, the rays determine that triangle T_1 hides triangle T_2.

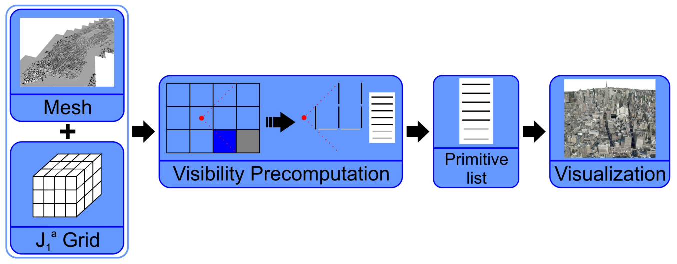

Fig. 4: Overview of the processing pipeline for visualization.

1) Given the user input where it is established the minimal dimension of the grid along one of its axis, we determine the size of the grids edges and then calculate the rest of the grids dimension.
2) Each triangle is firstly mapped to the grid block(s) where it is contained;
3) By using the triangles normals we identify what face of the block(s) the triangle is *looking at*. In this step we treat the back-face culling;
4) The internal block occlusion is then calculated using an approach similar to Z-buffer. The final list of primitives is assigned to each block faces. During this step we can find out if the block face is fully *filled*, this is needed to determine adjacent block occlusion in the visualization stage. This step is illustrated in Figure 3.

B. Visualization Stage

We use the grid to verify what elements are being directly looked at. Given the camera position ((i,j) illustrated in Figure 5) and its look-at vector, we use the J_1^a structure transition function to access the blocks in its line of sight. The look-at vector is also used to determine which block face(s) will be used to compose the final list of potentially visible of primitives.

Before each transition from block to block, the visible face of the current block is verified if it is fully filled in order to permit the operation. If the case is true, this means that the visible set from that face totally hides the visible set from its adjacent block face, so the hidden block and its subsequent adjacent blocks are not needed for the rendering stage. In Figure 5, the transition step illustrated. In this Figure, the after visiting the block $(i+1,j)$ the blocks $(i+2,j+1)$, $(i+2,j)$ and $(i+2,j-1)$ are queued to be visited next. After visiting block $(i+2,j-1)$ and determining that its visible face is fully filled, the next adjacent blocks are not directly added to the visitation cue. But there might be a case, in this example, where the block $(i+2,j)$ adds one or more of these adjacent blocks to the cue.

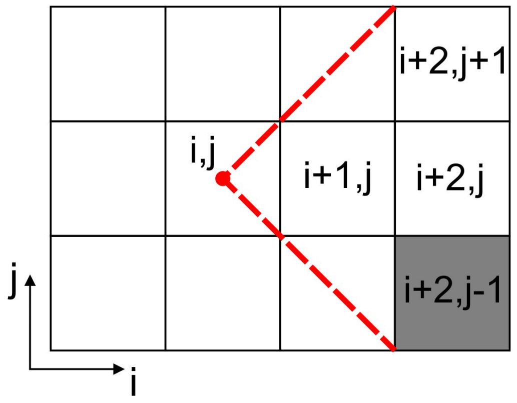

Fig. 5: Illustration of the trasition block operation, we illustrate a 2D case analogous to the 3D case. In this example each visible block is visited to obtain the visible set of primitives that belongs to them. In this illustration we indicate that the visible face of block $(i+2,j-1)$ is fully filled, so the next adjacent blocks are not directly added to the visitation queue due to it(they might be added due to another block).

The final list of blocks faces will then be used to compose the potentially visible set for the rendering stage. As said before, every time the camera is moved to another block or its look-at vector is changed, the calculation to obtain the visible set of primitives is redone.

Our method is capable of dealing well with primitive transparency in the case of both types of occlusion culling. It does so during the internal block occlusion stage. When the ray encounters a transparent/semi-transparent primitive, that primitive is tagged for transparency processing and the ray will continue to try and find another primitive along its path.

IV. Experiments

To assess the efficiency of our visualization structure, we have performed a series of experiments, where each model is visualized individually and we navigate along the scene. During the experiments, we basically verify the following result data.

- The mean frame rate (still (A) and navigation (B)) in comparison with using the whole data (C). This is a basic measure for any interactive 3D application and is measured in frames per second;
- Mean ratio (B) between the potentially visible set and the total number of primitives of the scene (#Pri). We

also analyze the mean ratio of only using view-frustrum and back-face culling (A) with the #Pri, we verify this to identify the proportion of elimination from each culling technique. The values range is $0-1$ and can be interpreted as for example: if ratio is equal to 0.10 then this means that the method eliminates 10% of the original primitives;

- Mean overestimation ratio (as proposed in [2]), where we compare the ratio between the size of the visible set (VS) and the size of the potentially visible set (PVS), in other words the ratio is equal to VS/PVS.

Just like the result data from 1, we calculate the mean value of the data in 2 and 3 after each scene navigation. For each result data we also present its standard deviation (σ). In order for providing a fair comparison between each ratio value, the same sequence of the scene navigation steps are made for every experiment executed in each model.

V. Results

We have performed a series of experiments to assess the visualization structures efficiency. These experiments were done on an Intel Core i7 2.00Ghz PC with 8GB Ram, with a Radeon HD 6770Ms Graphics Card and running on Windows 7 (64bits).

For these experiments, we use a series of simple mesh models obtained from the Aim-at-Shapes Repository[1]: Chinese Lion (identified as i in result tables), Vase (identified as ii), Armadillo (identified as iii), Hand (identified as iv) and Eros (identified as v), and also use the Manhattan model (identified as vi). This model was developed by Andrew Lock and is available for purchase at the 3D Cad Browser homepage [2], it is composed by a set of 306 meshes with a total of 3.6 million polygons, 5 million vertices and 296 texture images each one with the resolution of 4096x4096. This model is a great study case as it has all primitives forming a unique model of the scene. Figures 6 and 7 illustrate each model. The models shape characteristics and high level of delail makes them ideal to test our structures efficiency for visualizing individual meshes.

Figures 8 and 9 illustrates the series of navigation steps taken during the experiments. For the simple meshes we use the steps in Figure 8 and for the Manhattan model we use the steps illustrates in Figure 9.

Tables I and II present the result data detailed in the previous section obtained during our experiments. As we mentioned previously, the experiments executed in each model are done using the same sequence of navigation steps and same grid dimension.

As we can see from Table I the frame rate obtained when visualizing each model using the visualization structure is better than using the whole data of the model. This result only ascertains that the structure is working properly, the important results from this series of experiments is if the recalculation of the visible set affects the frame rate. By comparing columns 1.A and 1.B we can see that the recalculation step executed

[1]Homepage: http://aim-at-shape.net/

[2]Available at: http://www.3dcadbrowser.com

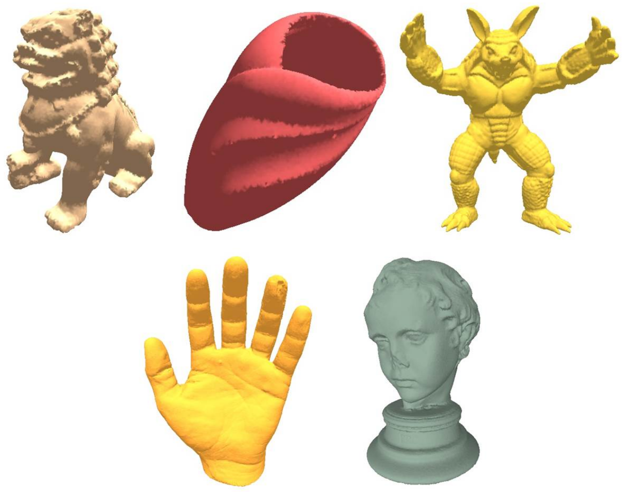

Fig. 6: Models used for testing the proposed approach. All models are very interesting test subjects due to their shape and high level of detail. Meshes Chinese Lion, Vase, Armadillo, Hand and Eros are respectively composed of 108k, 113k, 344k, 391k and 395k triangles.

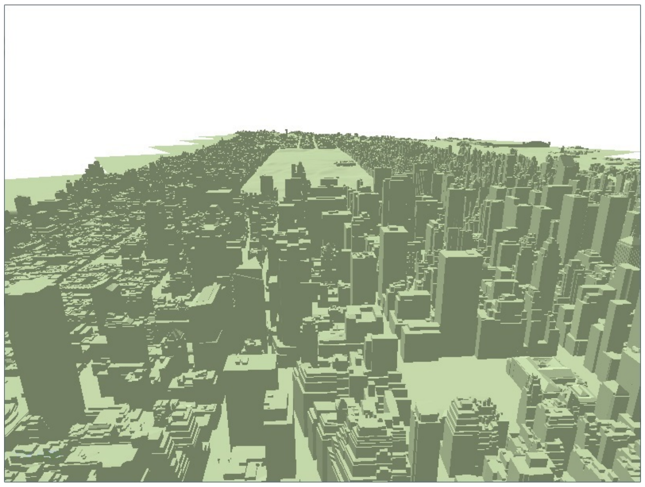

Fig. 7: Illustration of the Manhattan model. This model is composef of various objects which potentially makes it ideal for the occlusion detection experiments.

TABLE I: Result comparison of the frame rate experiments for each model. We analise the mean frame rate of static scene (1.A), dinamic scene (1.B) and the scene with the whole data visualization. We also analise the standard deviation of each experiment.

Model	1.A	$\sigma(1.A)$	1.B	$\sigma(1.B)$	1.C	$\sigma(1.C)$
i	28.569	0.967	25.773	2.591	18.883	2.766
ii	26.842	1.304	24.962	2.215	19.890	0.924
iii	11.367	0.557	9.756	0.860	6.221	0.245
iv	7.161	0.410	6.929	0.638	5.628	0.223
v	6.957	0.456	6.566	0.703	5.364	0.326
vi	10.75	0.884	9.85	0.788	0.6	0.043

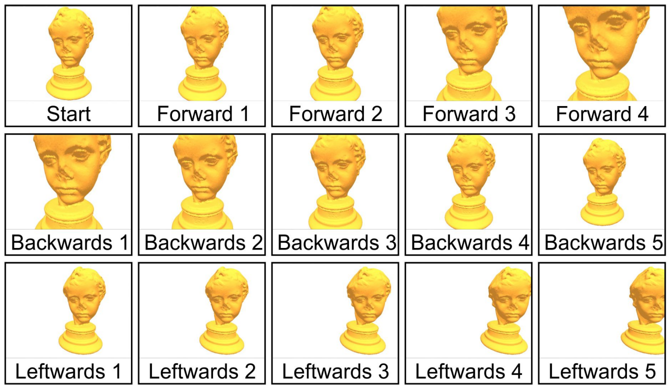

Fig. 8: Illustration of the navigation steps used during the experiments with the simple models illustrated in Figure 6.

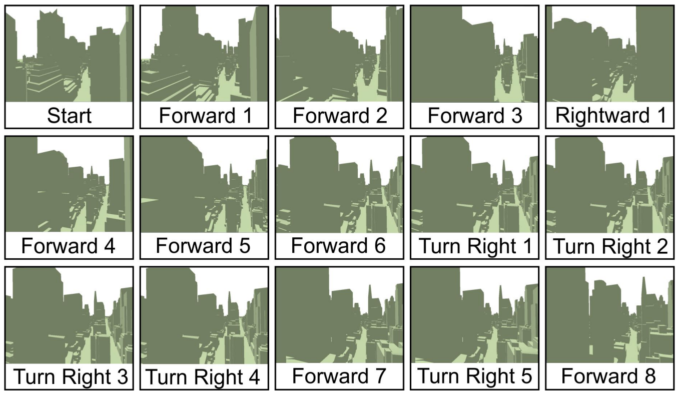

Fig. 9: Illustration of the navigation steps used during the experiments with the Manhattan model.

along the navigation sequence does not affect the frame rate too much, lowering it at most by 15%.

TABLE II: Comparizon between the mean rate (and standard deviation) of the set where the primitive set are removedby the *view-frustum* and *backface culling* in relaiton with $\#Pri$ ($\bar{R}_{VF+BF/PRI}$) and the mean rate (and standard deviation) of PVS in relation to $\#Pri$ ($\bar{R}_{PVS/PRI}$). We also analyze the rate between the number of primitives that are really visible and the size of the potentially visible set of primitives for experiment ($\sigma(\bar{R}_{VS/PVS})$).

Model	2.A	$\sigma(2.A)$	2.B	$\sigma(2.B)$	3	$\sigma(3)$
i	0.459	0.005	0.111	0.001	0.913	0.058
ii	0.452	0.002	0.291	0.003	0.957	0.032
iii	0.411	0.004	0.165	0.002	0.894	0.075
iv	0.467	0.002	0.132	0.003	0.914	0.067
v	0.401	0.002	0.215	0.003	0.856	0.077
vi	0.726	0.015	0.204	0.031	0.821	0.061

In Table II, we can see that the reduction of data being used for the visualizations stage is quite significant. Although the back-face culling did most of the hidden primitive removal, the occlusion culling still removes a reasonable amount of hidden primitives. And in the case of the vase, due to its concave shape, we can see that the occlusion culling removes a higher proportion of primitives. It removes 29.1% in comparison to the other models where a rate lower than 22% is removed.

From these results, we can figure out that, since the models that we use in each experiment are single object meshes, the occlusion culling does not remove as much primitives because of the actual lack of occlusion that occurs in the respective scene. In the same manner, the overestimation ratio is low due to the use of the single object meshes. At most the possible visible set is higher than the visible set by 15%.

When we analyze the overestimation ratio of our method in comparison with previous works, we could find out that our methods visible set has a roughly similar ratio, ranging from $3-9$ For the three biggest models used during the experiments, the pre-processing stage took $60-80$ seconds to execute.

Though we are showing this as a result of the visualization structure, the execution of this stage does not affect the visualization stage itself and, in many cases, once the precomputation stage has been executed, its application can just store the data obtained.

VI. Conclusion

We introduce a new visibility paradigm based on the use of J_1^a triangulation structure that avoids the rendering of too much unnecessary primitives in a 3D scene. This structure is capable of executing culling operations in order to deal with the minimum amount of primitives in a scene during the rendering stage, as possible. To do that, we propose to execute the culling by combining the paradigms based on viewing-frustum, back-face culling and occlusion culling. To our knowledge, this approach is new in comparison to existing approaches and there is no way to do a comparison because the objective is different of those (real time and interactive visualization through the web, of massive and large scenarios). To our belief, this approach to occlusion culling is also different from previous known works. Results have shown a substantial improvement over the traditional approaches if applied separately. Regarding the applicability, this novel approach can be used in devices with no dedicated processors or with low processing power or to provide data for visualization through the internet. In our lab, it will be applied in virtual museums applications.

In the very short, we intend to use the J_1^a structures adaptive feature to improve the preprocessing stage. That is, by having smaller blocks in certain regions of the scene, there will be a higher amount of fully filled face blocks which makes the adjacent block occlusion culling works better. Thus, with the better occlusion culling, there will be less overestimation.

We believe that this visualization structure is not only useful in the context of real-time rendering, but also in other applications such as online virtual environment application. In this case, the data from the environment is obtained on-line, this can permit the application to send the right data for the user to view without the need of sending the whole scene. With this in mind, we would also like to apply our technique in this scenario.

References

[1] A. Castelo, L. G. Nonato, M. Siqueira, R. Minghim, and G. Tavares, "The j1a triangulation: An adaptive triangulation in any dimension," *Computer & Graphics*, vol. 30, no. 5, pp. 737–753, 2006.

[2] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand, "A survey of visibility for walkthrough applications," *IEEE Transactions On Visualization and Computer*, vol. 9, no. 3, 2003.

[3] C. B. Jones, "A new approach to the 'hidden line' problem," *The Computer Journal*, vol. 14, no. 3, pp. 232–237, 1971.

[4] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," *Commun. ACM*, vol. 19, no. 10, pp. 547–554, Oct. 1976.

[5] D. Meagher, "Efficient synthetic image generation of arbitrary 3-d objects," *IEEE Computer Society Conference on Pattern Recognition and Image Processing*, pp. 473–478, 1982.

[6] J. M. Airey, J. H. Rohlf, and F. P. Brooks, Jr., "Towards image realism with interactive update rates in complex virtual building environments," *SIGGRAPH Comput. Graph.*, vol. 24, no. 2, pp. 41–50, Feb. 1990.

[7] S. J. Teller, "Visibility computations in densely occluded polyhedral environments," Berkeley, CA, USA, Tech. Rep., 1992.

[8] S. J. Teller and C. H. Séquin, "Visibility preprocessing for interactive walkthroughs," *SIGGRAPH Comput. Graph.*, vol. 25, no. 4, pp. 61–70, Jul. 1991.

[9] N. Greene, M. Kass, and G. Miller, "Hierarchical z-buffer visibility," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '93. New York, NY, USA: ACM, 1993, pp. 231–238.

[10] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer graphics: principles and practice (2nd ed.).* Addison-Wesley Longman Publishing Co., Inc., 1990.

[11] B. Garlick, D. D. Baum, and J. Winget, "Interactive viewing of large geometric data bases using multiprocessor graphics workstations," pp. 239–245, 1990.

[12] S. Kumar, D. Manocha, W. Garrett, and M. Lin, "Hierarchical back-face computation," in *Proceedings of the eurographics workshop on Rendering techniques '96.* London, UK: Springer-Verlag, 1996, pp. 235–244.

[13] U. Assarsson and T. Mller, "Optimized view frustum culling algorithms for bounding boxes," *Journal of Graphics Tools*, vol. 5, pp. 9–22, 2000.

[14] M. Slater and Y. Chrysanthou, "View volume culling using a probabilistic cashing scheme," *ACM Virtual Reality Software and Technology VRST'97*, pp. 71–78, 1997.

[15] J. Bittner, O. Mattausch, P. Wonka, V. Havran, and M. Wimmer, "Adaptive global visibility sampling," in *ACM SIGGRAPH 2009 Papers*, ser. SIGGRAPH '09, 2009, pp. 94:1–94:10.

[16] A. Chandak, L. Antani, M. Taylor, and D. Manocha, "Fastv: From-point visibility culling on complex models," in *Proceedings of the Twentieth Eurographics Conference on Rendering*, ser. EGSR'09, 2009, pp. 1237–1246.

[17] F. Tian, W. Hua, Z. Dong, and H. Bao, "Adaptive voxels: Interactive rendering of massive 3d models," *Vis. Comput.*, vol. 26, no. 6-8, pp. 409–419, 2010.

[18] L. Antani, A. Chandak, M. Taylor, and D. Manocha, "Fast geometric sound propagation with finite edge diffraction," Technical Report TR10-011, University of North Carolina at Chapel Hill, 2010., Tech. Rep., 2010.

[19] K. Bala, J. Dorsey, and S. Teller, "Radiance interpolants for accelerated bounded-error ray tracing," *ACM Trans. Graph.*, vol. 18, no. 3, pp. 213–256, Jul. 1999.

[20] ——, "Interactive ray-traced scene editing using ray segment trees," in *Proceedings of the 10th Eurographics conference on Rendering*, ser. EGWR'99, 1999, pp. 31–44.

[21] D. Cohen-or and A. Shaked, "Visibility and dead-zones in digital terrain maps," *Computer Graphics Forum*, vol. 14, pp. 171–180, 1995.

[22] D. Cohen-Or, E. Rich, U. Lerner, and V. Shenkar, "A real-time photo-realistic visual flythrough," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, pp. 255–265, 1996.

[23] S. Parker, W. Martin, P. pike J. Sloan, P. Shirley, B. Smits, and C. Hansen, "Interactive ray tracing," in *In Symposium on interactive 3D graphics*, 1999, pp. 119–126.

[24] N. Greene and M. Kass, "Error-bounded antialiased rendering of complex environments," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '94. ACM, 1994, pp. 59–66.

[25] N. Greene, "Occlusion culling with optimized hierachical z-buffering," in *In ACM SIGGRAPH Visual Proceedings*, 1999.

[26] ——, "A quality knob for non-conservative culling with hierarchical z-buffering," 2001.

[27] D. Luebke and C. Georges, "Portals and mirrors: simple, fast evaluation of potentially visible sets," in *Proceedings of the 1995 symposium on Interactive 3D graphics*, ser. I3D '95. ACM, 1995, pp. 105–106.