ISSN 0361-7688, Programming and Computer Software, 2020, Vol. 46, No. 3, pp. 223–232. © Pleiades Publishing, Ltd., 2020.
Russian Text © The Author(s), 2020, published in Programmirovanie, 2020, Vol. 46, No. 3.

Visualization of Large Scenes with Deterministic Dynamics

V. A. Semenov[a,b,c,*], V. N. Shutkin[a,], V. A. Zolotov[a,***], S. V. Morozov[a,d,****], and V. I. Gonakhchyan[a,*****]**

[a] *Ivannikov Institute for System Programming, Russian Academy of Sciences, ul. Solzhenitsyna 25, Moscow, 109004 Russia*

[b] *Moscow Institute of Physics and Technology (National Research University), Institutskii per. 9, Dolgoprudnyi, Moscow oblast, 141701 Russia*

[c] *National Research University Higher School of Economics, ul. Myasnitskaya 20, Moscow, 101000 Russia*

[d] *Moscow State University, Moscow, 119991 Russia*

**e-mail: sem@ispras.ru*

***e-mail: v451ly@ispras.ru*

****e-mail: vladislav.zolotov@ispras.ru*

*****e-mail: serg@ispras.ru*

******e-mail: pusheax@ispras.ru*

Received December 20, 2019; revised January 9, 2020; accepted January 19, 2020

Abstract—Visualization of large dynamic scenes is a challenging computer graphics problem. There are many approaches to solving this problem: frustum culling, occlusion culling, geometry simplification, and rendering optimization. One of the effective methods is the levels of detail (LOD) for scene objects. For large scenes, the hierarchical LOD (HLOD), whereby the levels of detail are created for large groups of objects, has proven efficient. However, this method faces certain difficulties when processing dynamic scenes. In this paper, we propose a method for visualizing scenes with deterministic dynamics that is based on hierarchical dynamic levels of detail (HDLOD). We also describe methods for generating HDLOD clusters and their visualization. Results of computational experiments conducted confirm high efficiency and practical potential of the proposed method.

DOI: 10.1134/S036176882003007X

1. INTRODUCTION

Visualization of large 3D scenes remains a challenging computer graphics problem. Even though the capabilities of graphics hardware grow and new methods and algorithms are developed, the realistic or authentic visualization of complex scenes remains an unattainable goal for many computer graphics applications. Many industrial software packages, e.g., CAD/CAE/CAM, BIM, and GIS, are critical with respect to the complexity of scenes and detalization of individual objects, because these packages involve a certain interactive model of human–machine interaction and enable efficient scene rendering on user's hardware. Nowadays, scenes often include thousands or millions of polygon models created in 3D modeling applications or obtained by scanning real-world objects. Moreover, scene objects can exhibit dynamic behavior given by deterministic or random events of their appearance, disappearance, or movement in a scene.

There are many approaches to solving this problem: frustum culling, occlusion culling, geometry simplification, and rendering optimization. Currently, geometry simplification is one of the most efficient approaches. It has one goal: reducing the number of polygons while preserving, to the extent possible, the main features of the original model. Geometry simplification methods differ in their basic decimation operators (vertex removal, edge collapse, vertex clustering, etc.), error metrics, and topology requirements to a polygonal model. More information can be found in [1]. The method described in [2] should be especially noted. Owing to edge collapse and quadric error metric, this method demonstrated good performance and high quality of simplification. In addition, this method can be employed for models that are not simply connected 2D manifolds.

One of the promising methods for visualizing complex scenes is the levels of detail (LOD) for scene objects. This line of research in computer graphics has

a long history, which began with the introduction of the LOD concept by James Clark in 1976. This concept implies that, for each scene object, a number of its representations with different degrees of detail are created. When visualizing a scene, suitable levels of detail are selected in such a way that more accurate representations are used for the objects close to the viewer, while more rough ones are used for distant objects.

Presently, to visualize large scenes, hierarchical levels of detail (HLOD) are widely employed [3, 4]. In contrast to classic LOD, HLOD provide simplified representations not only for individual objects but also for groups of objects organized in multi-level hierarchies. Instead of selecting a suitable level of detail for each object, it becomes possible to process groups of objects. This provides a higher degree of simplification and enables a reduction in the time it takes to traverse the scene tree, as well as reduction in the number of draw calls.

However, HLOD is difficult to use for arbitrary dynamic scenes. Each time objects appear, disappear, or move in the scene, their representations must be recomputed. The time required for the recomputation generally exceeds the time required for rendering the scene, which makes HLOD useless for scenes with a large number of dynamic objects. Known attempts to optimize the recomputation process by performing incremental updates executed in parallel were not successful [3].

In this paper, we discuss dynamic scenes with deterministic events, which determine the appearance, disappearance, and movement of objects in the scene. In this case, determinism is understood as the presence of a priori knowledge about when and with what objects the events occur. For effective visualization of these scenes, we propose a new method, which is called hierarchical dynamic levels of detail (HDLOD). This method creates hierarchical levels of detail that do not require recomputation when animating a dynamic scene. Our method differs fundamentally from the LOD methods used to render scenes with typical geometric models and predefined behavior patterns, e.g., scenes that simulate pedestrian flows [5].

The HDLOD method does not preclude the use of various rendering methods, including frustum culling and occlusion culling [6]. Having loaded a simplified representation of the scene in video memory, popular methods for visibility checks on the GPU can be employed [7–9]. This paper describes algorithms for the generation of HDLOD clusters and their visualization by using various rendering methods and presents results of our computational experiments, which confirm the efficiency and practical potential of the method proposed.

2. HIERARCHICAL DYNAMIC LEVELS OF DETAIL

Suppose that a scene $S(t)$ is defined in a 3D Euclidean space E^3 on a simulation time interval $t \in [0, T]$ and is represented as a linear list of objects $s(g_s, v_s, p_s) \in S$. Each object has an invariable geometric representation $g_s \subseteq E^3$. The presence of the objects in the scene is determined by their visibility functions $v_s(t) : [0,T] \to \{0,1\}$ in such a way that the function $v_s(t)$ is one if the object s appears in the scene at the instant t and it is zero if it does not. The position of the object is determined by its position function $p_s(t) : [0,T] \to M$, where M is a set of 4×4 matrices. It should be noted that the movement of objects can be defined in many different ways, e.g., by using a set of key points at which the position and orientation of an object are determined and the time when the object appears at a certain key point is specified. The position of the object at each instant can be determined by interpolating its positions at the nearest key points. However, for rendering, the position of the object is typically specified using a model transformation matrix. Moreover, any representation of the object's position can be expressed in terms of a matrix. Hence, without loss of generality, it can be assumed that, at each instant $t \in [0,T]$, the position of the object can be represented by a certain matrix.

Figure 1 shows an example of a dynamic scene that simulates the construction of a skyscraper in accordance with a predefined design project. As simulation time goes, its structural elements and construction machinery are installed at the construction site or removed from it. These appearances and disappearances can be rendered using the visibility functions shown in Fig. 2. The landscape elements do not change throughout the entire simulation period. In addition, during the construction, various machinery moves all over the construction site (see Fig. 3).

Hereinafter, we assume that the geometric representation of the scene objects is given by a set of triangles. We do not make any assumptions about the topology of the objects' boundary representations and do not require the boundary representations to be connected or manifolds, because, for rendering, only the so-called polygon soup is often provided without guarantees of any topological properties.

In terms of dynamic behavior, all objects in the scene can be divided into three classes.

The class of *static* objects includes the objects whose visibility and position do not change throughout the entire simulation period, i.e., $\forall t \in [0,T]\ v_s(t) = 1$, $p_s(t) = I$, where I is the identity matrix.

The class of *pseudo-dynamic* objects consists of the objects that appear and disappear without changing their positions: $\forall t \in [0,T] p_s(t) = I$, but $\exists t_1, t_2 \in [0,T]$ such that $v_s(t_1) \neq v_s(t_2)$.

Finally, the class of *dynamic* objects implies that both the visibility function and position function of the object vary with time: $\exists t_1, t_2, t_3, t_4 \in [0,T]$ such that $v_s(t_1) \neq v_s(t_2)$, $p_s(t_3) \neq p_s(t_4)$.

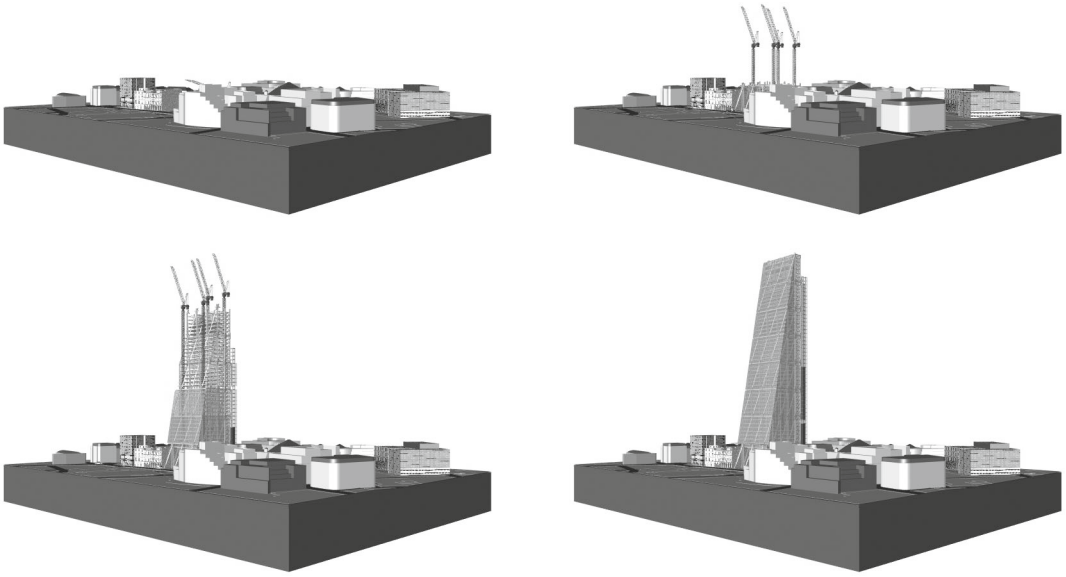

Fig. 1. Example of a dynamic scene that simulates the construction of a skyscraper.

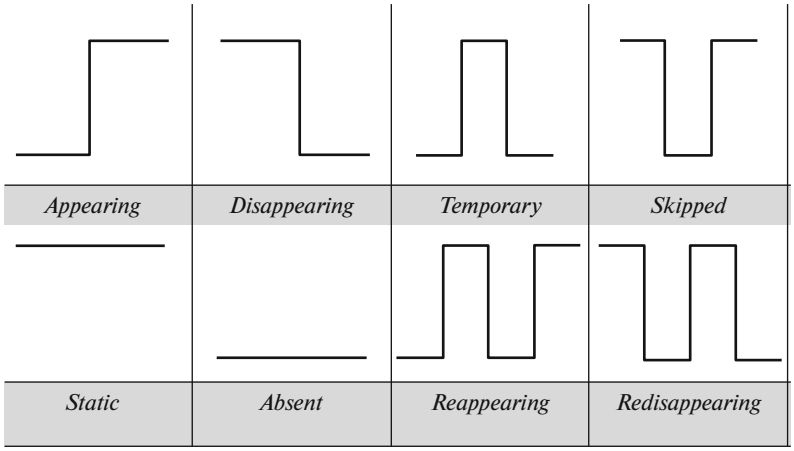

Fig. 2. Some typical visibility functions.

We refer to the hierarchical dynamic levels of detail (HDLOD) as a tree of clusters $C(G, V, P) = \{c(g_c, v_c, p_c), \prec\}$ represented by a set of clusters $c(g_c, v_c, p_c)$ with assigned geometric representations g_c, visibility functions $v_c(t) : [0,T] \to [0,1]$, and position functions $p_c(t) : [0,T] \to M$. In contrast to the visibility functions of objects $v_s(t)$, which take values 0 or 1, the visibility functions of clusters $v_c(t)$ take values on the interval from 0 to 1, thus using the concept of partial truth. Indeed, since some objects in a cluster can appear in the scene at some instant and other objects can disappear at the same instant, it is impossible to make an unambiguous decision about the appearance of the whole cluster. For the clusters, an agglomeration relation $\prec$ is also defined in such a way that $c' \prec c$ if and only if the cluster $c' \in C$ is a direct descendant (in the tree) of the cluster $c \in C$.

The first step of HDLOD generation is the classification of the scene objects into static, pseudo-dynamic, and dynamic. Each class uses its own cluster generation method.

Static and pseudo-dynamic cluster trees have similar structures: the leaves of the trees are strictly defined individual objects. Their internal nodes are clusters that aggregate the geometry (in the pseudo-dynamic case, also visibility) of the corresponding subtrees. Moreover, as the level in the tree rises, the geometry and visibility function of the clusters become simpler with the root of the tree being the simplest representation of a large group of objects.

Fig. 3. Example of dynamic objects in the scene (construction machinery).

Dynamic objects can have different motion patterns and different instants of appearance and disappearance. This significantly complicates the analysis and hinders their effective clustering. That is why, for each dynamic object, an individual cluster is created. In typical scenes, the same model can be used to represent different objects. For instance, several excavators that have the same geometric representation, being instances of the same general model of an excavator, can move all over the construction site. This is taken into account when generating clusters for dynamic objects: they can refer to the same geometric representation. It should be noted that, for clusters of dynamic objects, parent clusters can be additionally generated, which contain simplified geometry, simplified visibility function, and simplified position function.

Finally, all root nodes in the trees of static, pseudo-dynamic, and dynamic clusters are attached to a virtual node to form a single HDLOD tree. Figure 4 shows an example of this tree.

In addition to geometric and behavioral representations, each cluster $c \in C$ stores the following derived attributes: bounding box b_c, size or power w_c, as well as spatial error ε_c and time error γ_c defined below. In fact, these attributes are evaluated when generating hierarchical dynamic levels of detail and are used to display them. Thus, each HDLOD cluster is represented as a tuple $c(g_c, v_c, p_c, b_c, w_c, \varepsilon_c, \gamma_c)$.

The spatial error ε_c can be defined as an absolute error that specifies the maximum permissible local deviation of the geometric representation of the cluster c from the aggregated representation of objects $o \prec .. \prec c' \prec c$:

$$\varepsilon_c = \max_{c' \prec c} (\varepsilon_{c'}) + D_H\left(g_c, \bigcup_{c' \prec c} g_{c'}\right).$$

Here, the error ε_c is determined recursively using the Hausdorff metric $D_H(A, B)$, which is the longest distance from a point of one set to the nearest point of another set:

$$D_H(A, B) = \max\{\max_{x \in A} \min_{y \in B} D(x, y), \max_{y \in B} \min_{x \in A} D(x, y)\},$$

where A and B are closed sets of points and $D(x, y)$ is the metric's function in the Euclidean space.

The time error γ_c is defined for pseudo-dynamic clusters as the maximum deviation of the cluster's visibility function from the visibility functions of the original objects:

$$\gamma_c = \max_{c' \prec c} (\gamma_{c'} + D_V(v_c, v_{c'})),$$

where the distance $D_V(v_A, v_B)$ is computed using a functional metric:

$$D_V(v_A(t), v_B(t)) = \frac{1}{T} \int_0^T |v_A(t) - v_B(t)| dt.$$

These parameters are used to estimate the deviation of geometry and proximity of temporal behaviors. The spatial errors are computed when simplifying the geometry of the cluster. The temporal errors can be evaluated together with the cluster visibility function. To evaluate the cluster visibility function, the following formula is used:

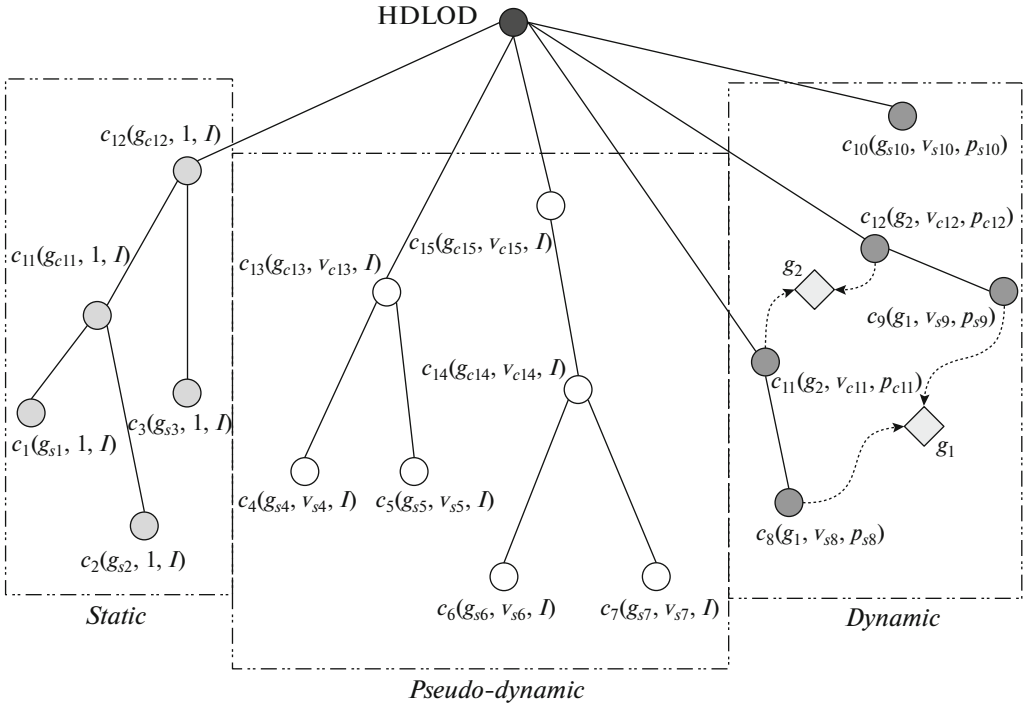

Fig. 4. Example of the HDLOD tree.

$$v_c(t) = \frac{\sum\limits_{c' \prec c} w_{c'} v_{c'}(t)}{\sum\limits_{c' \prec c} w_{c'}}.$$

The use of the weighted sums allows us to better take into account the behavior of significant objects and adequately render the behavior of the cluster. The function $v_c(t)$ expresses the weighted number of visible objects in the cluster at the instant t. If it takes a value of one, then all objects of the cluster are present in the scene at the instant t; if it takes a zero value, then they are absent in the scene at the instant t.

For each cluster, it must be decided if to display it, ignore it, or use its more accurate child representations. For this purpose, a time-dependent spatial error $\delta_c(t)$ is computed (spatial error caused by both geometric and temporal simplifications):

$$\delta_c(t) = \begin{cases} 0, & \text{for} \quad v_c(t) = 0 \\ \varepsilon_c, & \text{for} \quad v_c(t) = 1 \\ \varepsilon_c + (1 - v_c(t)) \sum\limits_{c' \prec c} w_{c'}, & \text{for} \quad \frac{1}{2} \le v_c(t) < 1 \\ \varepsilon_c + v_c(t) \sum\limits_{c' \prec c} w_{c'}, & \text{for} \quad 0 < v_c(t) < \frac{1}{2}. \end{cases}$$

When rendering the scene, the cluster tree is traversed. At each node, cluster attributes are analyzed to determine if the further traversal of the subtree is required. It is checked whether the cluster is present in the scene at a given simulation instant ($v_c(t) > 0.5$), whether the bounding box b_c of the cluster falls within the visibility cone, and whether the cluster error $\delta_c(t)$ is adequate to provide a desired quality. If all conditions are met, the representation of the cluster is immediately selected for displaying. In this case, the entire subtree can be excluded from the traversal. If the third condition is not satisfied, then the traversal of the cluster subtree continues and its child nodes also undergo the same checks until the leaf nodes with strictly defined scene objects are reached. The pseudocode of the HDLOD visualization algorithm is presented in Appendix A.

3. HDLOD GENERATION

The hierarchical dynamic levels of detail can be generated automatically using the proposed method. The first step of the HDLOD generation is the classification of the scene objects into static, pseudo-dynamic, and dynamic ones. Each class uses its own cluster generation method.

First, we consider the method for processing pseudo-dynamic objects as the most complex ones. This method uses hierarchical (bottom-up) clustering and multi-level accuracy control. It begins with individual objects and sequentially groups them into larger clusters until a desired number of levels is reached. The root clusters can be further simplified if the scene needs to be displayed as part of a more complex composition. The clustering should be carried out with

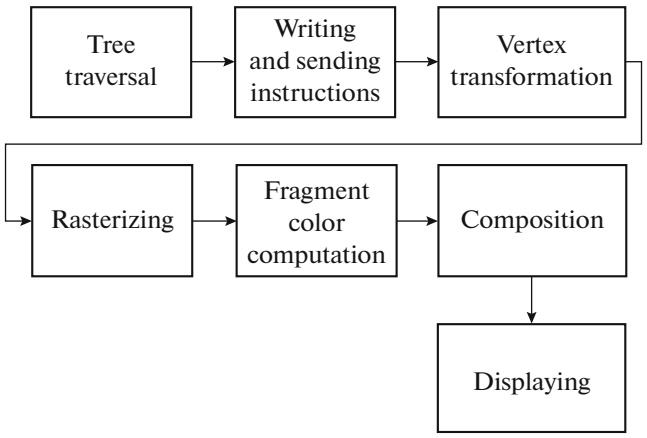

Fig. 5. Main steps of scene object rendering.

rigid accuracy control: the resulting tree of clusters must satisfy many requirements, including the expected number of levels of detail, degrees of nodes, spatial density and overlap of child clusters, their temporal proximity, and reduction in the complexity of clusters as the level of the tree rises.

Unfortunately, classical clustering methods cannot be directly applied to the problems of the HDLOD generation. The corresponding requirements are quite complex and may contradict each other. This complicates the mathematical formalization of the metric functions and connectivity criteria required for clustering methods [10]. The unacceptably high computational complexity of these methods is another factor preventing the adaptation of the classical results. For instance, a naive implementation of agglomerative clustering has the time complexity of $O(n^3)$ and memory consumption of $O(n^3)$, which makes it inapplicable even to fairly simple scenes. Faster hierarchical clustering with the time complexity of $O(n^2)$ and memory consumption of $O(n^2)$ is also not suitable for solving the problems under consideration [10].

Thus, we use another approach to generate the tree. The clustering process is divided into steps, at each of which the clusters generated must satisfy certain accuracy requirements. As new steps are made, these requirements are weakened in such a way as to guarantee the completion of the process after a certain number of steps equal to the number of levels of detail L.

At each step of the method $l\,(1 \le l \le L)$, an attempt is made to form new clusters with sizes $w_c \le w(l)$ and errors $\varepsilon_c \le \varepsilon(l)$ and $\gamma_c \le \gamma(l)$. For this purpose, the threshold values $w(l)$, $\varepsilon(l)$, and $\gamma(l)$ are selected in such a way that they monotonically increase with rising level. For instance, we can suggest the following values:

Table 1. Количество объектов и треугольников в сцене

	Static	Pseudo-dynamic	Dynamic	Total
Objects	498	40 010	15 649	56 157
Triangles	66 784	2 879 506	1 313 767	4 260 057

$$w(l) = \frac{l}{L}W, \quad \varepsilon(l) = 0.01\frac{l}{L}W = 0.01w(l),$$
$$\gamma(l) = 0.25\frac{l}{L}T,$$

where W is the size of the entire scene.

At each step, the method compiles a list of active clusters to be used at the next step. Initially, the list of active clusters includes all original objects. At each step of the method, some candidates among the active clusters are selected using Hilbert space-filling curves [11]. This allows us, on the one hand, to select candidates in the entire volume of the scene and, on the other hand, to localize them in densely filled regions. Next, for each candidate, its neighbors are found. The neighbors must satisfy conditions for spatial and temporal proximity and also guarantee the generation of a parent cluster with a desired accuracy. A new cluster is formed from the candidate and its neighbors, which become its children. The geometric representation of the new cluster is obtained by combining the representations of the children. It should be simplified to achieve the desired error $\varepsilon_c \le \varepsilon(l)$, e.g., by using the algorithm mentioned above [2]. The visibility function of the new cluster is determined by evaluating the weighted visibility function $v_c(t)$ described above. The new cluster is added to the list of active clusters, while its children are removed from it. If, for the current candidate, no suitable neighbors are found, then it is excluded from the analysis at the current step; however, it participates at the following steps. The pseudocode of the algorithm described above is presented in Appendix B.

Using spatial indexing, the clustering process can be carried out in $O(n\log n)$, where n is the number of scene objects. As compared to the classical clustering methods mentioned above, this result is much more adequate for large scenes. To quickly find the neighbors, various index structures can be employed [11]; in particular, regular dynamic octrees [12] perform well for pseudo-dynamic scenes. In our implementation, we use ordering with respect to each coordinate.

For static objects, the same clustering method is employed; in this case, however, only the spatial aspect is taken into account. Thus, static objects can be regarded as a special case of pseudo-dynamic ones.

For each dynamic object, an individual cluster is generated. In this case, the geometric representations of the objects are analyzed. If there are objects with the same representation, then the representation of their

Table 2. Frame rendering time (in milliseconds) when visualizing the skyscraper construction scene

	1/1 of screen	1/4 of screen	1/16 of screen	Without HDLOD
Beginning	3.81	3.58	3.29	16.9
1/3 of period	17.47	17.31	15.59	35.2
2/3 of period	8.5	7.87	7.53	54.22
End	3.72	3.58	3.25	61.56
Animation	9.59	8.51	7.94	47.43

clusters is set by referring to this representation. This makes it possible to avoid data duplication.

4. RENDERING

The efficiency of the HDLOD cluster visualization depends on the rendering methods employed. In this paper, we propose two subsystems operating in parallel. The first one determines visible clusters (visible surface determination), while the second one uploads polygonal representations of the clusters into video memory and sends instructions to the GPU by using the OpenGL programming interface.

When visualizing the HDLOD tree, it is traversed with cluster visibility checks at the current simulation instant from a specified camera position, taking into account the accuracy of the cluster's polygonal representation. For the clusters that passed the checks, messages about their visibility are sent to the second subsystem. Having received the message, the second subsystem sends a triangulated representation of the cluster and its rendering instruction to the GPU. Once all necessary data are buffered in the GPU memory, a draw call is made to run the execution of the rendering instructions. The execution of each instruction consists of the following steps: transforming the vertices (vertex shader), rasterizing the triangles, computing the color of the fragments (fragment shader), composing, and displaying (see Fig. 5).

The HDLOD method reduces the time required to complete the whole process described above. First, the number of triangles in the geometric representations of the clusters is reduced, which accelerates data transfer to GPU memory and reduces the time of vertex transformation. Second, the total number of rendering instructions is reduced, thus reducing the cost of their processing.

When processing a large number of instructions, CPU resources are used to validate and store the current state of the graphics pipeline. To reduce the overheads, the OpenGL extension NV_Command_List can be employed, which makes it possible to pre-generate an array of rendering instructions together with the current state of the pipeline [13]. This extension is quite efficient but has limited support on modern graphics hardware.

The software implementation of the HDLOD tree visualization method uses the OpenGL procedure MultiDrawElementsIndirect, which allows the array of buffered instructions to be executed in a single call [14]. However, this requires arrays of instructions, transformation matrices, and materials. To reduce the cost of buffering and removing invisible surfaces, the spatial decomposition method based on single reference octrees is employed [15]. Each octant is associated with the corresponding arrays that are supported in the state consistent with the HDLOD tree and are updated as clusters appear, disappear, or move. If the cluster moves within an octant, then its matrix is updated in the corresponding array. If the cluster moves to another octant, then the arrays of both the octants are updated.

5. COMPUTATIONAL EXPERIMENTS

To test the introduced concept of HDLOD and our method for automatic HDLOD generation and visualization, we carried out a series of computational experiments. The times of frame rendering when navigating the scene at fixed simulation instants and the average times of frame rendering for animation throughout the entire simulation period were measured. For testing, we selected the dynamic scene of skyscraper construction (Fig. 1). This scene is an example of a typical project in the construction industry: it has a static environment, a pseudo-dynamic model of a skyscraper under construction (the structural elements of which are installed in accordance with a certain design project), and dynamic machinery used for construction. All scene objects are represented by polygonal meshes. Table 1 shows parameters of the scene.

To estimate the effectiveness of HDLOD, the scene was visualized for various camera positions. In the first position, the camera was placed at the closest range possible to make the scene fully visible. In the other positions, the camera was placed at such distances that the scene occupied one fourth and one sixteenth of the screen. The time stamps were selected at the beginning, at the end, at one third, and at two thirds of the simulation period. The camera positions at which the scene was fully visible on the screen were selected so as to exclude the effect of frustum culling. The computational experiments were conducted on a computer with Intel Core i7-4790 (3.6 GHz), 16 GB RAM, and GeForce GTX 750 Ti (2 GB).

Table 2 shows the performance measurements taken during the experiments. The last column contains results obtained when rendering the scene without using simplified representations, while the first three columns contain results obtained using the HDLOD tree. It can be seen that the use of HDLOD significantly improves performance. As the camera zooms out, the desired effect grows because increasingly large (and simple) clusters are selected to visualize the scene. This dependence is observed both when navigating the static scenes fixed at the selected instants and when animating the scene. In this experiment, the levels of detail for dynamic objects were not used. However, when simplifications for dynamic objects are applied, the growth in performance as the distance to the scene increases becomes more significant. With increasing complexity of the scene, by the end of the simulation period (the skyscraper under construction), we can observe an increase in frame-time when visualizing the scene without HDLOD. However, it can be seen that HDLOD shows the worst performance for the time stamp of 1/3 of the simulation period. This is due to the fact that, at this point, a large number of changes occur in the scene and, for many clusters, their visibility functions $f_c(t)$ are close to 0.5, while their time-dependent spatial errors $\delta_c(t)$ are very large, which forces the use of child representations. However, the performance remains higher than that without HDLOD.

A similar series of experiments were carried out for some other problems associated with visual simulation of construction projects, urban infrastructure programs, and engineering processes. The experimental results also confirmed the high efficiency and practical potential of the proposed method.

6. CONCLUSIONS

In this paper, we have proposed a method for visualizing scenes with deterministic dynamics that is based on hierarchical dynamic levels of detail (HDLOD). In contrast to popular LOD methods and their hierarchical extensions (HLOD), the proposed method is applicable to a wide class of large dynamic scenes. We have also described algorithms for the HDLOD tree generation and visualization at a specified accuracy. The results of the computational experiments have confirmed the high efficiency and practical potential of the proposed approach. Our further research will be devoted to analyzing algorithmic variants of the proposed method, as well as its industrial application.

APPENDIX A:

PSEUDOCODE OF THE HDLOD VISUALIZATION ALGORITHM

```
PROCEDURE DISPLAY_CLUSTER(CLUSTER n, VIEW v, TIME t, RESOLUTION r)
{
        IF (VALUE_OF(BEHAVIOR_FUNCTION(n), t) EQUAL 0)
                RETURN
        ELSE IF (IS_OUTSIDE_FRUSTUM(BOUNDING_BOX(n), v))
                RETURN
        ELSE IF (VALUE_OF(DELTA_FUNCTION(n), t) / DISTANCE(n, v) < r)
        {
                IF (VALUE_OF(BEHAVIOR_FUNCTION(n), t) >= 0.5)
                        RENDER(GEOMETRY(n), v))
                ELSE
                        RETURN
        }
        ELSE
        {
                SET_OF_CLUSTER children = CHILDREN_NODES(n)
                FOR_EACH (CLUSTER child IN children)
                        DISPLAY_CLUSTER(child, v, t, r)
        }
}
```

APPENDIX B:

PSEUDOCODE OF THE ALGORITHM FOR CLUSTERING PSEUDO-DYNAMIC OBJECTS

```
PROCEDURE GENERATE_HDLOD(SCENE scene, INTEGER levels, HDLOD tree)
{
        SET_OF_CLUSTER active = NULL, next = NULL
        FOR_EACH (OBJECT object IN OBJECTS(scene))
        {
                CLUSTER cluster = FORM_CLUSTER(object)
                 ADD_TO(cluster, tree)
                 ADD_TO(cluster, active)
         }
        FOR_EACH (INTEGER step = 1 TO levels)
        {
                REAL epsilon, gamma, w
                 COMPUTE_LEVEL_THRESHOLDS(scene, levels, step, epsilon, gamma, w)
                WHILE (NOT_EMPTY(active))
                 {
                        CLUSTER representative = SELECT_REPRESENTATIVE(active)
                        SET_OF_CLUSTER neighbors = FIND_NEIGHBORS(active,
                                representative, gamma, w)
                         IF (IS_EMPTY(neighbors))
                        {
                                ADD_TO(representative, next)
                                REMOVE_FROM(representative, active)
                        }
                        ELSE
                        {
                            SET_OF_CLUSTER children
                            ADD_TO(representative, children)
                            FOR_EACH(CLUSTER neighbor IN neighbors)
                                    ADD_TO(neighbor, children)
                           CLUSTER cluster = CREATE_CLUSTER(children)
                           SIMPLIFY (cluster, epsilon)
                           ADD_TO(cluster, tree)
                           ADD_TO(cluster, next)
                           REMOVE_FROM(representative, active)
                           FOR_EACH (CLUSTER neighbor IN neighbors)
                                    REMOVE_FROM(neighbor, active)
                        }
                 COPY(next, active)
                 EMPTY(next)
        }
}
```

REFERENCES

1. Luebke, D. et al., *Level of Detail for 3D Graphics,* San Francisco: Morgan Kaufmann, 2003.
2. Garland, M. and Heckbert, P.S., Surface simplification using quadric error metrics, *Proc. 24th Annu. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH),* 1997, pp. 209–216.
3. Erikson, C. et al., HLODs for faster display of large static and dynamic environments, *Proc. Symp. Interactive 3D Graphics (I3D),* 2001, pp. 111–120.
4. Lilley, S. and Cozzi, P., Cesium 3D tiles: Beyond 2D tiling, 2016. https://cesium.com/presentations/files/FOSS4GNA2016/3DTiles.pdf.

5. Toledo, L. et al., Hierarchical level of detail for varied animated crowds, *Visual Comput.,* 2014, vol. **30**, nos. 6–8, pp. 949–961.

6. Cohen-Or, D., Chrysanthou, Y.L., Silva, C.T., and Durand, F., A survey of visibility for walkthrough applications, *IEEE Trans. Visualization Comput. Graphics,* 2003, vol. 9, no. 3, pp. 412–431.

7. Bittner, J. et al., Coherent hierarchical culling: Hardware occlusion queries made useful, *Comput. Graphics Forum,* 2004, vol. 23, no. 3, pp. 615–624.

8. Guthe, M. et al., Near optimal hierarchical culling: Performance driven use of hardware occlusion queries, *Proc. Eurographics Symp. Rendering,* 2006, pp. 207–214.

9. Mattausch, O. et al., CHC++: Coherent hierarchical culling revisited, *Comput. Graphics Forum,* 2008, vol. 27, pp. 221–230.

10. Xu, D. and Tian, Y., A comprehensive survey of clustering algorithms, *Ann. Data Sci.,* 2015, vol. 2, pp. 165–193.

11. Samet, H., *Foundations of Multidimensional and Metric Data Structures,* San Francisco: Morgan Kaufmann, 2006.

12. Morozov, S. et al., Indexing of hierarchically organized spatial-temporal data using dynamic regular octrees, *Lect. Notes Comput. Sci.,* 2018, vol. 10742, pp. 276–290.

13. Lorach, T., OpenGL NVIDIA command-list: Approaching zero driver overhead, 2014. https://on-demand.gputechconf.com/siggraph/2015/presentation/SIG1512-Tristan-Lorach.pdf.

14. Bennett, J. and Carter, M., Performance gains achieved through modern OpenGL in the Siemens DirectModel rendering engine, 2015. http://on-demand.gputechconf.com/gtc/2015/presentation/S5387-Jeremy-Bennett.pdf.

15. Gonakhchyan, V., Efficient command buffer recording for accelerated rendering of large 3D scenes, *Proc. 12th Int. Conf. Computer Graphics, Visualization, Computer Vision, and Image Processing,* 2018, pp. 397–402.

Translated by Yu. Kornienko