

Levels of Detail (LOD) Engineering of VR Objects

Jinseok Seo

Gerard Jounghyun Kim

Kyo Chul Kang

Dept. of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
San 31, Hyoja-dong, Pohang, Kyungbuk, Korea
82-562-279-5664

jsseo@postech.ac.kr

ABSTRACT

For real-time performance, virtual reality systems often employ various performance optimization techniques. One of the most popular methods is the use of levels of detail (LOD). In previous papers by this author [1], we have proposed to use software engineering principles such as the concept of hierarchical and incremental modeling, and simultaneous consideration of form, function and behavior for modeling VR objects. Each model refinement stage driven by such a philosophy produces step-by-step form and its fitting function and behavior. We can make good use of these by-products as LOD's for adaptive display and simulation by additionally specifying conditions for LOD switching. Such specifications can be simulated and analyzed in advance for estimation of performance for given simulation environment. Such an engineering process deals with behavior and geometry together; different geometric LOD's can have different behaviors which may in turn be dependent upon an existence of geometric feature that may not be preserved if created using simplification algorithms. Mixing geometric and behavior LOD's is particularly useful for populating the virtual environment with "live" objects (e.g. people, vehicles, creatures, particles, etc.) for improving the realism within the computational resource constraint. We demonstrate our approach by modeling an automobile object with three levels of geometric and behavior detail in a top-down manner, simulate their instances in a small virtual town, and based on the simulation result, make predictions to the maximum allowable number of vehicles that will maintain an acceptable frame rate if executed in a faster simulation environment. We believe that our approach combines the idea of hierarchical refinement of virtual objects and the use of LOD in a very natural and intuitive manner.

Keywords

Levels of Detail (LOD), Software Engineering, Specification, Real-time Rendering, Simulation, Top-down Design.

*LEAVE BLANK THE LAST 3.81 cm (1.5")
OF THE LEFT COLUMN ON THE FIRST PAGE
FOR THE COPYRIGHT NOTICE*

1. INTRODUCTION

Despite the rapidly improving processing power and graphics capabilities of today's computer, implementing realistic, yet fast and interactive VR system will continue to demand engineering of the "resource vs. realism" trade-offs. One of the most popular methods of such performance optimization is the use of levels of detail (LOD). The basic idea of LOD is of improving the display performance by rendering objects with less detail when they are visually less important in a frame. The use of LOD's in VR, in most cases, actually refer to the use of different levels of "geometric" detail; an idea of improving the display performance by rendering objects with less detail when they are visually less important in a frame. Exploitation of the similar idea for behavior or simulation has been suggested as well [1][2]. Techniques of generating different levels of geometric LOD are mostly "bottom-up" approaches [3], that is, the most detailed mesh model are "simplified" to several levels. No particular methods of producing behavior levels of detail seem to exist.

On the other hand, the idea of "top-down" design of software and geometric objects has been advocated for many years [4][5][6]. In [1], we have introduced a CASE tool called ASADAL/PROTO that supports the application of software engineering principles (such as the concept of hierarchical and incremental modeling, and simultaneous consideration of form, function and behavior) for modeling VR objects. At each model refinement driven by such a philosophy, we obtain geometric, functional and behavior models of intermediate level of detail. Naturally, we can make good use of these by-products as LOD's for adaptive display and simulation by also making it possible to specify conditions and methods of LOD switching. Put it differently, we should consider how each LOD might be simulated during the hierarchical modeling process. It also means that the development mostly proceeds from "abstract but affordable" models to "detailed, but bulky" models, and that abstract models need not be extracted from detailed ones.

Then, such specifications can be simulated and analyzed in advance for estimation of performance for given simulation environment. Such an engineering process deals with behavior and geometry together; different geometric LOD's can have different behaviors which may in turn be dependent upon an existence of geometric feature that may not be preserved if created using simplification algorithms. For instance, we may still want to render the motion of cow's legs at a distance (with all other geometric details of the cow simplified). Mesh simplification algorithms that can selectively preserve user-defined features, which will satisfy such a purpose, is yet to be devised. Alternatively, we may wish to simplify the simulation model

together with the simplified geometry at a distance. A scheme of handling dynamic LOD for both behavior and geometry at run time would be needed as well. Most simulation packages offer dynamic switching of geometric LOD's, but not for behavior or simulation models. Mixing geometric and behavior LOD's is particularly useful for populating the virtual environment with "live" objects (e.g. people, vehicles, creatures, particles, etc.) for improving the realism within the computational resource constraint.



Figure 1: A small virtual town populated with number of vehicles

In this paper, we outline a hierarchical engineering process of modeling virtual objects with consideration of their geometric and behavioral LOD. The objective of the LOD engineering is to maximize the realism and quality (e.g. low variation in frame rate) of the virtual environment for a given computational resource. We demonstrate our approach by modeling an automobile object with three degrees of geometric and behavior detail in a top-down manner, simulate their instances in a small virtual town (See Figure 1), and based on the simulation result, make predictions to the maximum number of allowable vehicles that will keep an acceptable frame rate if executed in a faster simulation environment. Based on the specification and simulation results, we have implemented the "town with a traffic", and compare the estimated and actual performance.

2. Related Work

Conventional approaches to preparing for (geometric) LOD's for virtual objects is through a process called the "Polygon Budgeting" in which objects in the scene are identified, and assigned the maximum number of polygons depending on the scene complexity according to the limitation of the target graphics hardware. Detailed geometric models are created using geometric modelers and CAD systems, and then simplified using mesh simplification algorithms. A good survey of simplification algorithms is given in [7]. One of the outstanding problems with simplification algorithms is the preservation of geometric features (e.g. topology, curvature, vertex position, etc.) There is an inherent trade-off between degrees of simplification and appearance preservation. Moreover, these algorithms are applied

to the overall model, and can not "selectively" simplify or preserve certain "semantic" features which may have to remain important visually.

The exploitation of simulation levels of detail has been suggested first by [2]. Carlson used three levels of simulation detail for one legged virtual robots (the simplest model being a point-mass model and the most complex one being a full rigid body dynamic model) and demonstrated how the overall performance varied by dynamically switching the simulation LOD's. However, their work did not elaborate on the engineering side of the approach. The models and LOD switching techniques were handcrafted for the particular example. In terms of display quality, the notion of "Temporal LOD" can be used to adjust the constancy of the frame rate, as its variation is known to cause performance degradation [8].

Generation of LOD's is based on either bottom-up or ad-hoc implementations. Both engineering design community and software engineering community have long advocated for a top-down approach to design [4][5][6]. Intermediate models obtained from hierarchical modeling processes serve as excellent source of specifications and documentation for both software and engineering design for later maintenance. In the context of virtual reality, they have an additional use as LOD's. In our previous paper by this author, we have introduced a CASE tool called ASADAL/PROTO that supports hierarchical and incremental modeling and simulation, and simultaneous consideration of form, function and behavior for modeling VR objects [1]. Work reported in this paper is in continuation of this work, to apply the method not only for modeling but also for performance optimization. In [9], a hierarchical grouping approach to modeling "crowds" in VE has been suggested. Depending on the viewpoint, the "resolution" of the crowd may differ and likewise their behavior; at close range, different behaviors of each constituents are visible and from a distance, crowds are shown as one texture. Similarly to the work of Carlson [2], geometry of each group could not have been generated using the conventional mesh simplification algorithms and the behaviors of each crowd group must have been coded separately. The purpose of this work is to formulate a structured approach to modeling such entities in VE as well.

3. LOD Engineering

In this section, we outline the overall process of LOD engineering of VR objects using ASADAL/PROTO, a hierarchical VR object prototyping tool. For more details of the capabilities of ASADAL/PROTO, refer to [1]. Before LOD engineering of a single virtual object can start, a scene-wide planning must precede with polygon budgeting. LOD engineering does not replace polygon budgeting but extends it. Figure 2 shows the overall process.

3.1 Polygon Budgeting

Hoffman[10] gives a good review of the polygon budgeting process. Just to summarize it, the first step in the polygon budgeting is to understand the limits of the target hardware where the application is to run on. Although not exact, it is possible to obtain a rough estimate of how much time it takes to process and shade a polygon from hardware specifications, and simple benchmark tests. In addition, one needs to also determine the desired frame rate, field-of-view, and viewing distance (far

clip) of the application (the type of application may influence values of these parameters; for instance, flight simulation requires a relatively larger far clip).

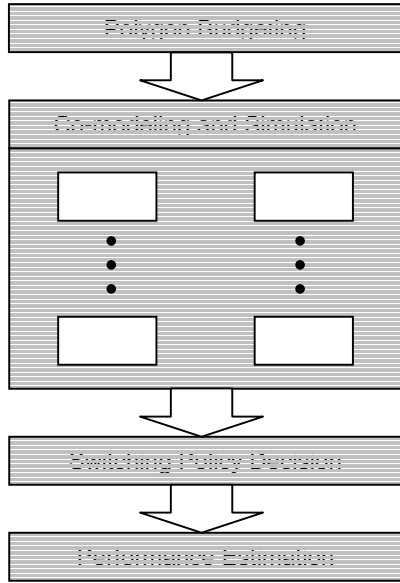


Figure 2: The Overall Process of LOD Engineering

	No. of Visible Objects at one time	Appx. No. of Polygons per Object	Total Avail. Polygons per frame (7K)
Far (LOD 0)	80	25	2,000
Intermediate (LOD 1)	50	40	2,000
Close (LOD 2)	15	200	3000

Table 1: A Polygon Budget Table

The next step is to analyze the approximate expected scene complexity. The scene complexity generally refers to the number of polygons visible for a given frame. Table 1 shows a polygon budget for a system that can process about 7,000 polygons for a given frame rate. We fill in the left most column first estimating the scene complexity at the object level. Based on the budget of 7,000 polygons, we make a decision as to the number of levels of detail and how many polygons to assign for each level (the rightmost column). Then, the approximate number of polygons assignable to each object at each level is computed (middle column). The more experienced the application developer is, the more accurate estimate this will be. Even so, this estimate is only very rough because certainly each object will vary in the number of required polygons to represent its geometry. However, these figures can be used as an initial guidance during more detailed object modeling.

3.2 Co-Modeling of Behavior and Geometry

According to the ASADAL/PROTO modeling process, once important system objects are identified, their form, function and behavior are modeled “concurrently”, possibly affecting one another. Each aspect of the model is refined hierarchically can be simulated at each refinement stage. The polygon budget brings a soft constraint on the degrees of refinement and number of refinement stages. The main objective and merit to using the hierarchical and incremental modeling strategy is to progressively achieve the most detailed model by focusing on the more important parts or features of the model one at a time. As the intermediate models now become the LOD, these high priority features to be added, simulated and verified at each stage are to be determined by what deems to be visually important for the given LOD level. The order on the priority of these features will determine the flow of the co-evolution of behavior and geometry LOD. We may think of this as refining the model in its “feature” space, and producing the corresponding behavioral and form model.

Addition of “features”¹ is manifested in the STATECHARTS and VOS² as two major operations: in the STATECHARTS, as (1) decomposition of an object state (and addition of associated functional decomposition and new events/transitions/actions), (2) addition of a new concurrent process, and (3) replacement of states (and associated specification data); and likewise, in the VOS, as (1) decomposition in the geometric configuration, (2) addition of a geometric feature and (3) replacement of geometry. These operations are illustrated in the Table 2.

Behavior Refinement Operations	Geometric Refinement Operations
Decomposition of a State (b_1)	Decomposition in Configuration (g_1)
Addition of Concurrent Process (b_2)	Addition of Geometric Object (g_2)
Replacement of States (b_2)	Replacement of Geometry (g_3)

Table 2: Possible Refinement Operators

Although there may be other refinement operations possible, for the purpose of this paper, we inductively define the Specification of a Geometric and Behavior Level of Detail each to be the VOS and STATECHARTS/DFD models created using the three refinement operators with respect to the one-level less detailed model respectively.

Definition 1: Geometric LOD, G_n , of a geometric model G is a VOS created by applying the three geometric refinement operators to a non null G_{n-1} , the one level less detailed geometric specification of G . The initial geometric LOD may be null. The first non null LOD is “created” by the user.

¹ We will not deal with deletion of features, that is, the levels of detail increase, the number of “features” increase monotonically.

² In ASADAL/PROTO, a behavior/function is specified using STATECHARTs and Data Flow Diagrams, and a geometric form is specified using a construct called VOS.

$$G_n = g_i(G_{n-1})$$

Definition 2: Behavior LOD, B_n , of a behavior model B is a STATECHARTS/DFD pair created by applying the three behavior refinement operators to a non null B_{n-1} , the one level less detailed behavior specification of B . The initial behavior LOD may be null. The first non null LOD is “created” by the user.

$$B_n = b_i(B_{n-1})$$

Definition 3: An LOD, L_n , of a model M is a tuple of G_i and B_i .

$$L_n = (G_i, B_i)$$

For an LOD to be valid, its components, G_i and B_i must be compatible, that is, can not refer to a non-existent geometric (e.g. component) or behavioral feature (e.g. state, event).

Note that the definition of the LOD does not impose any binary constraint other than compatibility. In our proposed model of co-evolution of behavior and form, the compatibility constraint is naturally established. An alternative approach is to consider form and behavior in isolation and their association determined later. In this scheme, while the generation of LOD’s may be easier, establishing the association may be difficult or even impossible (by neglecting to include a necessary behavioral or geometric feature). Note that the whole process is based on an assumption is that the finer the model is, for geometry, there will be more polygons to process, and likewise for behavior, there will be more states, events, and actions to process.

3.3 Specifying Switching Policy

The condition for “when” the LOD switching occurs is usually specified according to the distance of the object from the viewer. It would be desirable to specify more detailed conditions for LOD switches. For instance, within a fixed distance, different LODs may be used depending on the viewing angles. Ideally, these conditions must be specified during the co-modeling of geometry and behavior by envisioning and estimating the dynamic spatial distribution of the virtual objects. This is a very difficult process and it is inevitable to go through cycles of trial and error to tune the system just right. By using ASADAL/PROTO’s capability of specification simulation, some of the effort can be saved before the programming level

In most simulation environments that support geometric LOD’s, the model switch usually occurs abruptly and may cause “popping”, the effect referring to the models seen as suddenly changing their appearance. There is not an elegant solution to this problem other than using many levels of geometric levels of detail.

The same problem arises for behavior levels of detail. When refining the behavior model, one must also consider how the model switch between B_n and B_{n-1} will occur so that the switch can occur as smoothly as possible visually and correctly simulation-wise. This entails specifying “special” state transitions among the states of B_n and B_{n-1} . Currently, it is not possible to specify such state transitions between two refinement levels using the STATECHARTS of ASADAL in a very clean manner. We are considering to add new semantics for our implementation of the STATECHARTS to accommodate this feature.

3.4 Simulation and Performance Estimation

As indicated before, implementing a VR system often requires cycles of performance tuning and the purpose of carrying out an LOD engineering beforehand is to reduce this cycles of work as much as possible at the specification level.

While the primary purpose of specification simulation is to verify the logic and temporal properties of behavior specification, its expected performance can be estimated indirectly by collecting relative timing data. However, there is an inherent difficulty because the computational model of specification simulation can be quite different from the computational model of the platform where the actual VR implementation would be run (which can be several such as one-single-loop model, distributed model, etc). Moreover, if it would be possible to specify conditions and actions for LOD switches, its overhead must be accounted for during simulation and its performance estimation. This calls for a specification simulation tool whose computation model is modeled after typical VR simulation models.

4. Example: “Virtual Town with Traffic”

In this section, we demonstrate our idea of LOD engineering for modeling and determining the appropriate number of instances of LOD’s of a vehicle object. There are 30 vehicles (with a fixed path) to be populated in a small virtual town.

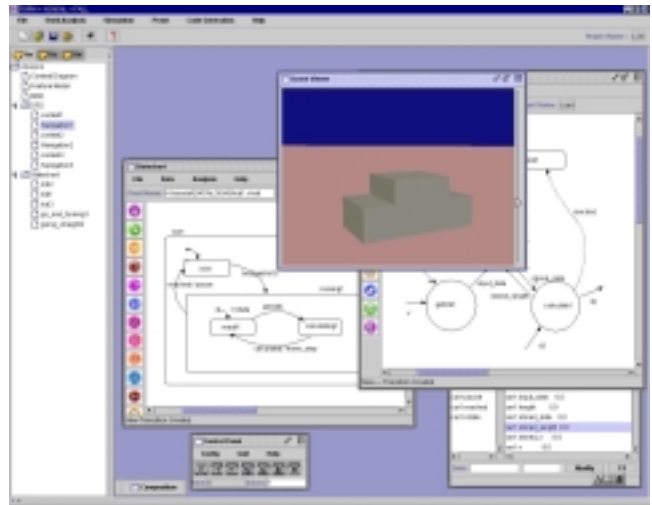


Figure 3: First level Geometric and Behavioral LOD

4.1 Behavior and Geometry

First, we “create the most abstract geometric LOD for the vehicle, with two rectangular cubes (24 polygons). Next, its corresponding behavior is specified. The vehicle is to run back and forth on a given path in constant velocity with “abrupt” turning at the corner. No detection of collision with other vehicles is used (See Figure 3).

At the next level, the behavior model is refined first. First, we add a behavior such that when a collision is detected, the vehicle stops for a fixed number of seconds and restarts. We also add the “smooth” turning behavior for more realistic traffic look. The vehicle still moves in same constant velocity inherited from its predecessor LOD. Although not necessary (since there is no new geometric feature required by the new behavior model), we go

ahead and refine the next level of vehicle geometry by decomposing the vehicle body and adding the wheels and windshields (about 242 polygons, See Figure 4).

In the next level, we decide to refine the geometry. We add the license plate, the hub caps, and the lights resulting in a model of about 432 polygons. With these geometric features, the final behavior LOD is designed that adds new behaviors for velocity control based on collision detection, animation of the running wheels, computation of the right steering angles, and creation of dynamic paths.

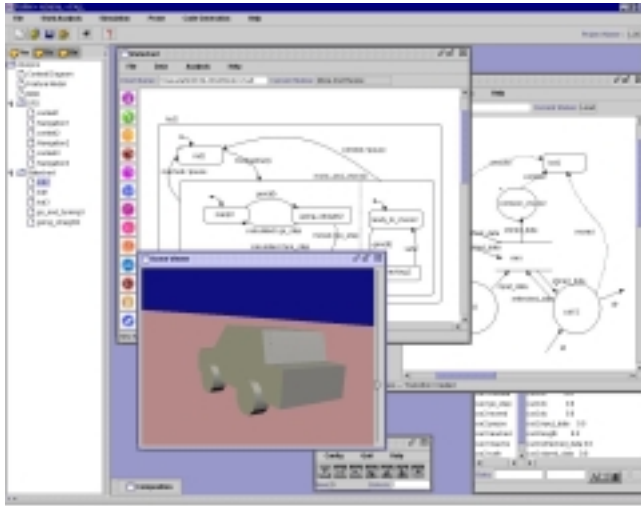


Figure 4: Second Level Geometric and Behavior LOD

4.2 Switching Policy

For both geometric and behavior LODs, we used the simple viewing distance criterion for their switching conditions. The switching of the behavior was handled in a special manner; at the time of the switch (e.g. when a vehicle crosses the viewing distance boundary), its position is recorded and handed to the next behavior LOD. That position was used as parameters to computing the next position to make the vehicle move as smoothly as possible between LOD switches.

4.3 Performance Estimation and Comparison

Given the geometric and behavior LOD's, it is possible to define the LOD's by looking for the compatible pairs. Since we know, through the modeling process, which geometric model influenced which behavior model and vice versa, we already know that certain pairs are possible. Here, for simplicity, we just use the three LOD's, (G_0, B_0) , (G_1, B_1) , and (G_2, B_2) . In order to decide upon the appropriate number of instances of each LOD, we make a rough estimation of the dynamic spatial behavior of the vehicles using the scheme shown in Figure 5 (x, y, z are no. of LOD2, LOD1, LOD0 instances respectively).

ASADAL/PROTO's model of visual simulation is implemented in Java3D (the specification is semi-automatically translated into Java3D) and thus follows its computation model. This made very difficult for us to collect relative performance data from specification simulation, because Java3D renders the scene only

upon an update by other simulation threads, while in Performer³, the execution environment we chose, the rendering process never sits idle. The Figure 6 show relative performance data collected using different LOD assignments using the specification simulation (the graph) and using Performer/SGI Indigo (the table). The timing data in the graph of Figure 6 include both application and rendering, while the timing data in table reflect the application time only. While the relative performance mostly correlated between the two models, it was difficult, unfortunately, to distinguish between cases B, C, D and E (Figure 6) at the specification level for any performance assessment due the difficulty stated above. It is obvious that the performance will be the worst when using all detailed models and vice versa. The difference among the cases like B, C, D, and E may become clear if there were more virtual objects to be populated or by making the computational model of the specification simulation same as that of the execution, as stated before.

$$d1 : d1 + d2 : d1 + d2 + d3 = \sqrt{x} : \sqrt{x + y} : \sqrt{x + y + z}$$

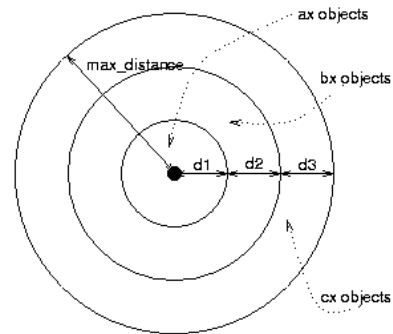


Figure 5: Assigning the number of LOD instances.

5. Conclusion and Future Work

In this paper, we have presented the concept of a structured LOD engineering processing using a top-down refinement strategy. We do not contend that this method is to replace the need for bottom-up approaches as design of systems often employs both paradigms. We have already proposed in a previous paper the use of structured specifications prior to code design for VR system development. Making use of the models from the intermediate refinement levels for LOD is a natural and intuitive method for performance conscious modeling. Before proceeding to the next modeling stage, we can decide whether certain part of the system could be refined or not on the basis of the intermediate simulation results. Hopefully, the resulting virtual world is a conglomeration of geometric objects and their behaviors that are appropriately refined in different parts, and switching policies that are tailored for the user needs.

ASADAL/PROTO, our specification making tool, is still short of supporting the requirements for VR software. We plan to, one by one, add specification tools and modify the computational model of the specification simulation of ASADAL/PROTO to make it a better planning tool for VR applications.

³ Performer is a registered trademark of Silicon Graphics, Inc.

6. ACKNOWLEDGMENTS

The work presented in this paper was supported in part by the Korea Institute of Science and Technology Planning and the Korea Institute of Science and Technology.

7. REFERENCES

- [1] G. Jounghyun Kim, K. C. Kang, H. Kim and J. Lee, Software Engineering of Virtual Worlds, Proceedings of the ACM Symposium on Virtual Reality Software and Technology, pages 131-138, 1998
- [2] Deborah A. Carlson and Jessica K. Hodgins, Simulation Levels of Detail for Real-time Animation, Proceedings of Graphics Interface, pages 1-8, 1997
- [3] Hoppe, H. Progressive Meshes. Computer Graphics (SIGGRAPH '96 Proceedings), pages 99-108, 1996
- [4] SE – top down
- [5] Simon, H. The Sciences of the Artificial, Second Edition, MIT Press, 1981
- [6] Pahl, G. (Tr. by K. Wallace), Engineering Design, The Design Council, Springer Verlag, 1984
- [7] Luebke, D. A Developer's Survey of Polygonal Simplification, IEEE VR Conference Tutorial 7 Course Notes, 1999
- [8] Watson, B On Temporal Level of Detail: System Responsiveness, Feedback and User Performance, IEEE VR Conference Tutorial 7 Course Notes, 1999
- [9] Musse, S Crowd Modeling in Collaborative Virtual Environments, Proceeding of the ACM VRST Conference, pp. 115-124, 1998
- [10] Hoffman, W Database Design for Visual Simulation and Entertainment, SIGGRAPH 97 Course Notes on Designing Real Time Graphics for Entertainment, 1997

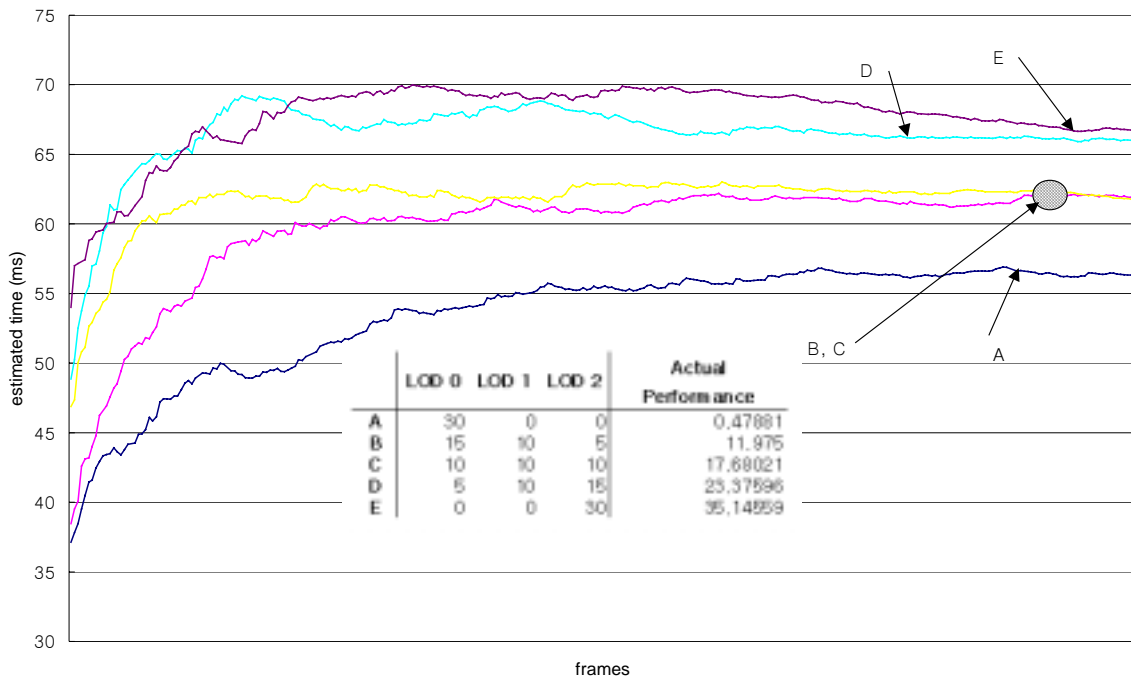


Figure 6: Performance of different LOD combinations at the specification level.