Dynamic Collision Detection in Virtual Reality Applications

Jens Eckstein
Daimler-Benz AG
Research and Technology
Ulm, Germany
eckstein@cs.uni-sb.de

Elmar Schömer
Saarland University
Department of Computer Science
Saarbrücken, Germany
schoemer@cs.uni-sb.de

ABSTRACT

We present data structures and algorithms for *dynamic* collision detection in virtual reality (VR) applications. The methods are applicable to all general polygonal models. They combine the advantages of collision detection using bounding volume (BV) hierarchies with the ability to compute dynamic collision detection results. The results are used as input for further simulations, e.g. contact or dynamics simulation. First we present new methods to compute BV hierarchies using optimization goals which can also be used to improve known computation methods. Second we show how to integrate BV hierarchies into a process of dynamic collision detection, so that the bounding objects as well as the surface patches of the objects are tested for overlap during their motions. The performance of the techniques is shown by means of a fitting simulation in the automotive industry.

Keywords: dynamic collision detection, bounding volume hierarchies, fitting simulation, virtual reality

1 Introduction

Virtual reality (VR) is a leading-edge technology in the development process of technological challenging products as can be seen in current automotive industry. Virtual prototyping is an effictive means to shorten development times for new cars. One of the most interesting parts in the virtual prototyping of a car is the packaging which includes fitting simulations. VR offers the framework to simulate such processes in real-time within a most realistic environment. One of the key problems for the VR applications is the real-time simulation of realistic object behaviour in complex virtual environments. These environments have the following characteristics:

- *Model complexity:* The input models consist of thousands of polygons.
- *Unstructured representation:* The objects are collections of polygons with no topological information. They may have cracks, holes, T-joints or non-manifold geometry. Such objects are known as *polygon soups*.
- *Unspecified motions:* The object motions in VR applications are not specified in advance.
- *Need for dynamic collision detection:* [Hel95] distinguishes between three types of collision detection results:
  - *Static* collision detection checks the interference of objects at one particular configuration.
  - *Pseudo-dynamic* collision detection performs static interference tests at very short time intervals in order not to miss any collisions.

- *Dynamic* collision detection considers the swept volumes of objects during their motions and is able to report contact time and contacting object features.

The information on the contact geometry is necessary to implement further object simulations such as contact simulation [Buc98], dynamics simulation [Mir96, Sau98b, Sau98c] or virtual grasping [Sau98a]. Fast dynamic collision detection is a major bottleneck of multi-body simulation and thus a challenging problem.

Main Contribution

We present efficient algorithms for dynamic collision detection in VR applications. They base on BV hierarchies which have been successfully used in static and pseudo-dynamic collision detection so far. Our algorithms build BV hierarchies with different types of bounding objects (sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB)) and extend known techniques by simple but effective means. Our major contributions are:

- new algorithms for the computation of BV hierarchies with different types of bounding objects; extension of known hierarchy computation methods to improve collision detection performance.
- generalization of the *static* and *pseudo-dynamic* collision detection procedures to a full *dynamic* collision detection process. To do this we present efficient dynamic bounding object tests and basic tests between object features (vertices, edges, faces).
- comparison of our BV hierarchies with the best known hierarchies.

We have implemented the presented collision detection methods as basic part of the multi-body simulation module of **DBView**, the VR software platform of Daimler-Benz Research and Technology. The performance of our methods is demonstrated using complex virtual scenes taken from fitting simulation during the packaging of the car development process.

The rest of the paper is organized as follows. In section 2 we provide an overview of other hierarchical collision detection methods. Section 3 describes the computation of BV hierarchies. In Section 4 we explain the methods to integrate BV hierarchies into a process of dynamic collision detection. Section 5 presents the performance results of our methods and compares the different BV hierarchy types. Section 6 concludes by mentioning some applications which profit from a dynamic collision detection in a high degree.

2 Previous Work

Interference and collision detection problems have been extensively studied in the literature. The most powerful algorithms use BV hierarchies with various types of bounding objects. They include sphere trees [Qui94, Hub95, Pal95], AABB trees [Zac95, Zac97], OBB trees [Got96], k-DOP trees [Klo98, Zac98] or trees with other kinds of bounding objects [Bar96]. The advantage of BV hierarchies is that they can handle general polyhedral models with high model complexity but their use is currently restricted to static and pseudo-dynamic collision detection.

Different types of collision detection algorithms are based on the GJK-algorithm [Cam97, Leu96, Hec94, Gil88] or the closest-feature tracking algorithm [Lin93, Pon95, Mir97]. These algorithms are mainly restricted to convex polyhedral objects. E.g. [Mir96] uses the closest-feature tracking as collision detection for his impulse-based dynamics simulation.

Only few research has been performed to realize dynamic collision detection for bounding objects such as spheres, AABBs or OBBs. The main effort has been made to speed up static interference detection between those bounding object types. E.g. [Got96] proposed an efficient static collision test for OBBs.

3 Computation of BV Hierarchies

In this section we describe how to build BV hierarchies. The use of BV hierarchies is currently the fastest known collision detection

method for unstructured object representations. The computation of the hierarchies differs in various aspects:

- *Types of bounding objects:* Most of them use a single type of bounding objects.
- *Degree:* Most BV hierarchies have degree 2, higher degrees are rarely used, e.g. 8 [Hub95].
- *Type of computation:* There are generally two different kinds of computation: *top-down* and *bottom-up*.

We present two new *top-down* methods to compute BV hierarchies (see section 3.1 and 3.2) and improve the well known GOTTSCHALK-method. *Top-down* computation methods of BV hierarchies consist of two parts:

- compute a bounding object for a set of faces
- split a set of faces

The main difference of our methods from the methods known so far is the *splitting* step. The bounding objects enclose the faces at a certain quality which can be expressed by different measures, e.g. *volume*, *surface* of the bounding object or *directed Hausdorff-distance* of the bounding object to the enclosed set of faces. Splitting a set of faces means to bound the set of faces with more than one bounding object in order to get a tighter bounding. Common *top-down* computation methods for BV hierarchies split sets of faces with a single, fixed degree. Our new methods are able to split a set of faces with any degree given from 2 to 6. This flexibility enables us to control the splitting process so that each splitting improves the quality of the bounding objects by a given amount. Given a set of faces F and a bounding object E_F, the splitting of F in k sets of faces results in $F_1, \ldots, F_k$ with bounding objects $E_{F_1}, \ldots, E_{F_k}$. The quality of the splitting is the quality of the worst bounding object. So $q(E_F)$ is the quality of E_F and $\max_{i=1,\ldots,k} q(E_{F_i})$ is that of the splitting. The splitting fulfills the quality requirements if $\alpha \cdot q(E_F) \geq \max_{i=1,\ldots,k} q(E_{F_i})$ with a predefined *improvement factor* $\alpha \in (0,1)$. The goal of a splitting is to reach the quality improvement with a minimal splitting degree. Thus we start with degree 2, compute the splitting and increase the splitting degree until the improvement requirement is fulfilled. The splitting of a set of faces stops if the number of faces falls below a certain threshold. In our experiments 4 has been a good choice for this threshold. If the improvement cannot be reached with degree 6 the set of faces will be split with degree 6 to overcome local splitting problems.
It remains to explain the splitting mechanisms.

3.1 Heuristic without Optimization

The center of a face is the mean value of the 3D-coordinates of its vertices and the center of a face set is the mean value of the centers of its faces. The distance between faces and the distance of faces from points is computed using the centers of the faces. In the first step all faces belong to F_1, the center $\boldsymbol{c}_{F_1}$ of F_1 is the center of F. In step i ($i = 2, \ldots, k$) we select the face f_{max} which maximizes the distance from one of the current centers $\boldsymbol{c}_{F_1}, \ldots \boldsymbol{c}_{F_{i-1}}$. The center $\boldsymbol{c}_{f_{max}}$ of f_{max} is the temporary center of F_i. All faces are rearranged so that each face belongs to the subset $F_1, \ldots, F_i$ whose current center minimizes the distance to the face. Now all centers of $F_1, \ldots, F_i$ are recomputed as the mean values of its faces. The step of rearranging the faces and recomputing the centers of the subsets is iteratively done until no changes occur. At the end of the process all faces crystallize into the subsets $F_1, \ldots, F_k$ and for each set of faces a bounding object is computed.

3.2 Heuristic

During the splitting step of the *heuristic without optimization* we do not know anything about the quality of the bounding objects of the current subsets. The goal of this *heuristic* is to control the partitioning of the face set F by the quality of the bounding objects.
In the first step we select k faces which belong to different subsets in the final partition.

For this purpose we compute the smallest enclosing box B_F (with respect to the selected quality measure) for the whole set of faces F. We partition this box B_F into k subregions and select for each region the face of F which minimizes the distance to the center of the region. The partitioning of the box B_F is illustrated in figure 1 projected into two dimensions. For degree 2 the longest axis is halved, for degree 3 the longest axis is split twice, for degree 4 the longest and the second longest axis are halved, for degree 5 an additional center is put into the middle of B_F and for degree 6 the longest axis is split twice and the second is halved.

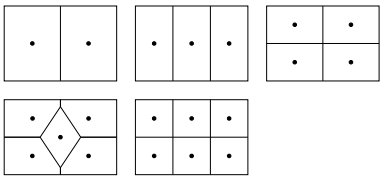

Figure 1: Partitioning of an OBB into regions

For each of these faces we compute the smallest bounding object (sphere, AABB, OBB). Now we insert in a random order the remaining faces of F. To do this for face f we insert f in each current subset F_i $(i = 1, \ldots, k)$ and modify the bounding object of F_i. We add f to a subset so that the worst quality of all bounding objects of $F_1, \ldots, F_k$ is minimal.

3.3 Modified Gottschalk-method

We simply use the algorithm sketched above to find the first k faces of the partition and assign each face f to the subregion whose center is closest to the center of f. In this way we obtain the subsets $F_1, \ldots, F_k$ and compute for each subset F_i $(i = 1, \ldots, k)$ a bounding object.

4 Dynamic collision detection

In Section 3 we described methods to compute BV hierarchies. These hierarchies are very efficient for static and pseudo-dynamic collision detection in complex virtual environments. They are currently not used in dynamic collision detection. We present methods to use BV hierarchies using spheres, AABBs and OBBs in dynamic collision detection, regardless which method is used to compute the hierarchies.

Taking a look at the process of static collision detection with BV hierarchies it can be seen that there are two different types of collision tests:

- bounding object tests
- basic tests between object features (vertices, edges, faces)

For both kinds of tests there exist efficient static collision tests. In the following we present these efficient dynamic collision tests which enable real-time collision detection in virtual environments of moderate size.

4.1 Bounding Object Tests

Efficient static collision tests between simple bounding objects are well known (see [Eck99] for a survey). For dynamic collision detection we need an extension of those methods to enable efficient dynamic collision detection for bounding objects. In order to keep most of the effectiveness of known static collision tests between bounding objects we approximate the swept volume of each bounding object during its motions by a new one which is used in an efficient static collision test.

Since we generally do not know the exact trajectories of the moving objects but only their positions and orientations at discrete points in time, we simply construct a sphere or a box which encloses the bounding objects at the beginning and at the end of a motion step and if desired at intermediate points in time. The smaller a single step is the more reliably the swept volume of the bounding objects is enclosed by the sphere or box. The smallest enclosing sphere for a set of spheres or boxes can be determined very efficiently. If we want to construct a small oriented enclosing box for a set of OBBs we choose the orientation of an arbitrary given OBB and proceed as in the case of axis aligned boxes. If the motion steps are small enough it suffices to enclose two bounding objects, namely those at the beginning and the end of the motion step.

4.2 Basic Tests

Suppose that two objects O_1 and O_2 are moving from their current configuration at time $t = 0$ to an intended future configuration at time $t = 1$. We want more than just determine, whether O_1 and O_2 intersect at time $t = 1$. We want to find out, whether (and when) O_1 and O_2 collide (for the first time) during their simultaneous motion, even if the configuration at time $t = 1$ is collision–free. Since our objects are modelled as sets of polygons, a collision between O_1 and O_2 can be detected by looking for contacts occurring between a vertex of O_1 and a face of O_2, or vice versa, or an edge of O_1 and one of O_2. For both contact types we define a function $d : [0, 1] \longrightarrow I\!R$, which changes its sign in the neighbourhood of a point in time at which a collision occurs. In this way the problem of calculating potential collision times is reduced to finding the roots of the function $d(t)$ in the interval $[0, 1]$.
In the case of a vertex–face collision we simply choose $d = \boldsymbol{n}^T\boldsymbol{v} - n_0$ to be the signed distance between the vertex $\boldsymbol{v}$ and the infinite plane $\{\boldsymbol{x} \in I\!R^3 | \boldsymbol{n}^T\boldsymbol{x} = n_0\}$, in which the face lies.
In the case of an edge–edge collision we consider the infinite lines defined by the edges. Let $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, and $\boldsymbol{w}_1$, $\boldsymbol{w}_2$ be the endpoints of both edges, then we set $d = \boldsymbol{n}^T(\boldsymbol{w}_1 - \boldsymbol{v}_1)$, where $\boldsymbol{n} = (\boldsymbol{v}_2 - \boldsymbol{v}_1) \times (\boldsymbol{w}_2 - \boldsymbol{w}_1)$. $d(t)$ becomes zero iff all four endpoints lie in a common plane, i. e. the lines intersect or are parallel.
In general we can only evaluate the function $d(t)$ at discrete points in time, so the roots have to be approximated. We made best experience with a simple interpolation scheme and the method regula falsi.
So far the condition $d(t) = 0$ is only a necessary condition for the occurrence of collisions between a vertex and a face or between two edges. That is the reason why we have to check every root τ of $d(t)$ in the interval $[0, 1]$ whether it corresponds to a real collision:

Vertex – Face

In the following we assume that $f = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_{k-1}, \boldsymbol{v}_k \equiv \boldsymbol{v}_0)$ is a convex face which is no restriction because non-convex faces can be triangulated. The vertices of f are enumerated clockwise when looking in direction of the face normal.
Provided that the vertex $\boldsymbol{v}$ lies in the plane of the face f at time $t = \tau$, we can examine whether $\boldsymbol{v}$ lies within the boundaries of f in the following way:

$$\forall\, i \in [0 : k\text{–}1] : ((\boldsymbol{v}_{i+1} - \boldsymbol{v}_i) \times \boldsymbol{n})^T(\boldsymbol{v} - \boldsymbol{v}_i) \leq 0.$$

Edge – Edge

Provided that the two infinite lines defined by the edges intersect in a common point at time $t = \tau$, we can easily decide whether the intersection point lies within both edges:

$$0 \leq (\boldsymbol{v}_2 - \boldsymbol{v}_1)^T(\boldsymbol{n} \times (\boldsymbol{w}_1 - \boldsymbol{v}_1)) \leq \boldsymbol{n}^2$$
$$0 \leq (\boldsymbol{w}_2 - \boldsymbol{w}_1)^T(\boldsymbol{n} \times (\boldsymbol{w}_1 - \boldsymbol{v}_1)) \leq \boldsymbol{n}^2$$

5 Implementation and Performance

We have implemented the algorithms as basic part of the simulation environment of the VR software platform **DBView** of Daimler-Benz Research and Technology.

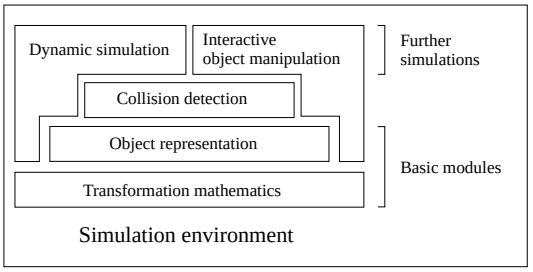

Figure 2: Structure of the simulation environment

One of the application areas for simulation of realistic object behaviour with a challenging part for collision detection is in the packaging process of the construction of cars. An example is shown in figure 3. The task is to remove the two covers for the light bulbs at the headlight. The full task additionally includes the removal of the two light bulbs after the removal of the covers. The scene contains only the relevant parts for the removal (circa 60000 faces). Even if more parts are included in the scene the performance of

the algorithms would not deteriorate because of the space partitioning system used to handle many objects in a scene (for details see [Eck99]).
We have interactively generated motion sequences for the removal of the two covers. The resulting motions consist of colliding and collision-free parts so that the performance in both cases can be seen. We have carried out extensive experiments with different types of bounding objects and pseudo-dynamic collision detection. In order to compare the different hierarchy types we have measured the real running time of the motion sequences. All experiments are performed with a Silicon Graphics MIPS R10000 194Mhz. In a first step we examined the influence of the parameters for degree and improvement on the running time. We omitted the diagrams for these tests (for details see [Eck99]). For each presented hierarchy type improvement factor $\alpha = 0.75$ has been best. Although improvement factor 0.75 is best for all three computation methods of section 3, it cannot be said that this is generally true. But experiments have shown that the computation methods generating hierarchies with flexible degrees tend to outperform the hierarchies with fixed degree.
For the three computation methods of sections 3.1-3.3 we compare the results of the simulation. The results are illustrated in figure 4. As we can see all three methods show a good overall performance. Common techniques to summarize the performance of collision detection algorithms during a motion sequence use average collision time. Taking a look at the illustrations in figure 4 we learn that the collision detection time varies very fast depending on the proximity of the objects in the scene. Therefore we take a closer look at the worst-case performance. The 40 msecs needed by the modified GOTTSCHALK-method is the slowest time measured but a the same time guarantees interactive rates.
In a second step we compare the performance of pseudo-dynamic and dynamic collision detection. It is obvious that dynamic collision detection is more time-consuming than pseudo-dynamic collision detection. But as can be seen from figure 5 dynamic collision detection using linear interpolation or linear interpolation with regula falsi results in computation times which are at most 100 msecs using linear distance interpolation with regula falsi. Thus our methods for dynamic collision detection facilitate the direct handling of objects in complex virtual environments at interactive rates.

6 Conclusions

In this paper we have presented new methods to compute BV hierarchies and to enable dynamic collision detection with BV hierarchies in VR applications. We have implemented the algorithms as part of the VR platform **DBView** of Daimler-Benz Research and Technology and demonstrated the performance by means of a complex example of a fitting simulation from automotive industry.
Further results concerning the interplay of our dynamic collision detection with contact and dynamics simulation are presented in [Buc98, Sau98b].

REFERENCES

[Bar96] G. Barequet, B. Chazelle, L.J. Guibas, J.S.B. Mitchell, and A. Tal. Boxtree: A hierarchical representation for surfaces in 3*d*. *Eurographics*, 15(3):387–396, 1996.

[Buc98] M. Buck and E. Schömer. Interactive rigid body manipulation with obstacle contacts. In 6^{th} *Int. Conf. in Central Europe on Computer Graphics and Visualization, WSCG'98*, pages 49–56, 1998.

[Cam97] S. Cameron. Enhancing GJK: Computing minimum and penetration distances between convex polyhedra. In *IEEE Int. Conf. Robotics and Automation*, 1997.

[Eck99] J. Eckstein. *Echtzeitfähige Kollisionserkennung für Virtual Reality Anwendungen.* Ph.D. thesis, Department of Computer Science, Saarland University, 1999.

[Gil88] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

[Got96] S. Gottschalk, M.C. Lin, and D. Manocha. OBB-tree: A hierarchical structure for rapid interference detection. *Computer Graphics*, pages 171–180, August 1996. Proc. SIGGRAPH'96.

[Hec94] P.S. Heckbert. *Graphics Gems IV*. Academic Press, 1994.

[Hel95] M. Held, J.T. Klosowski, and J.S.B. Mitchell. Evaluation of collision detection methods for virtual reality flythroughs. In *Proc.* 7^{th} *Canadian Conf. Compt. Geometry*, pages 205–210, 1995.

[Hub95] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Trans. on Visual. and Comput. Graph.*, 1(3):218–230, September 1995.

[Klo98] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Trans. Visualization and Computer Graphics*, 4(1):21–36, January 1998.

[Leu96] K. Chung Tat Leung. An efficient collision detection algorithm for polytopes in virtual environments. Master's thesis, University of Hong Kong, Department of Computer Science, 1996.

[Lin93] M. C. Lin. *Efficient Collision Detection for Animation and Robotics*. PhD thesis, University of California, Berkeley, CA, December 1993.

[Mir96] B. Mirtich. *Impulse-based dynamic simulation of rigid body systems*. PhD thesis, University of California, Berkeley, 1996.

[Mir97] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, MERL, July 1997.

[Pal95] I.J. Palmer and R.L. Grimsdale. Collision detection for animation using sphere-trees. *Proc. Eurographics*, 14(2):105–116, 1995.

[Pon95] K. Ponamgi, D. Manocha, and M. Lin. Incremental algorithms for collision detection between solid models. In *Proc. ACM SIGGRAPH Symp. on Solid Modelling*, pages 293–304, 1995.

[Qui94] S. Quinlan. Efficient distance computation between non-convex objects. In *Proc. Int. Conf. on Robotics and Automation*, pages 3324–3329, 1994.

[Sau98a] J. Sauer, F. Lakssiouar, and T. Ullmann. Virtual calibrated grasping with data-gloves. In *VSMM, Gifu, Japan*, volume 1, pages 158–163, 1998.

[Sau98b] J. Sauer and E. Schömer. A constraint-based approach to rigid body dynamics for virtual reality applications. In *Proc. ACM Symposium on Virtual Reality Software and Technology*, pages 153–161, 1998.

[Sau98c] J. Sauer, E. Schömer, and C. Lennerz. Real-time rigid body simulations of some classical mechanics toys. In 10^{th} *European Simulation Symposium and Exhibition, ESS'98*, pages 93–98, 1998.

[Zac95] G. Zachmann and W. Felger. The BoxTree: Enabling real time and exact collision detection of arbitrary polyhedra. In 1^{st} *Workshop on Simulation and Interaction in Virtual Environments*, pages 104–113, 1995.

[Zac97] G. Zachmann. Real-time and exact collision detection for interactive virtual prototyping. In *Proc. ASME Design Engineering Technical Conferences*, 1997.

[Zac98] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proc. of IEEE, VRAIS'98 Atlanta*, March 1998.

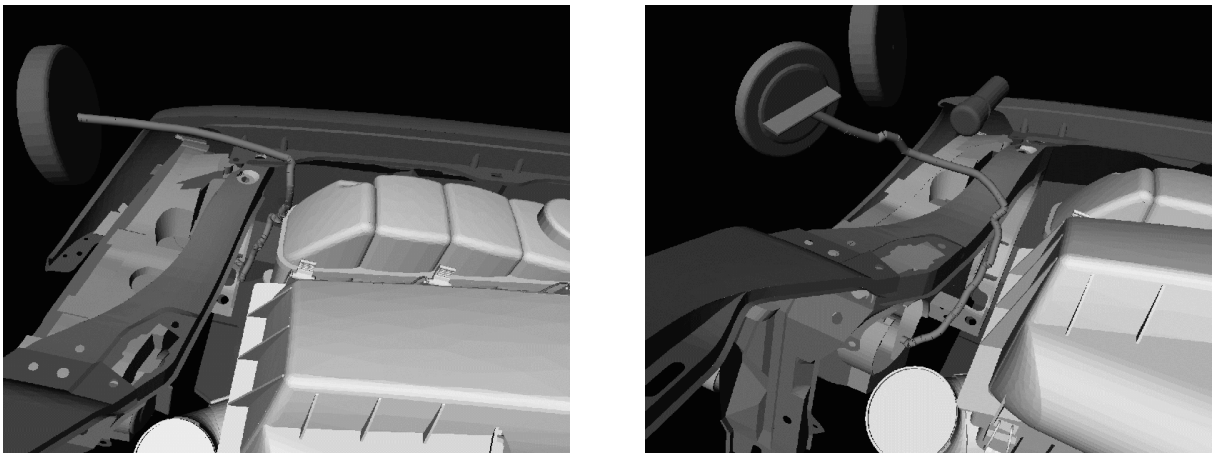

Figure 3: Removal of cover for the headlight and the additional headlight

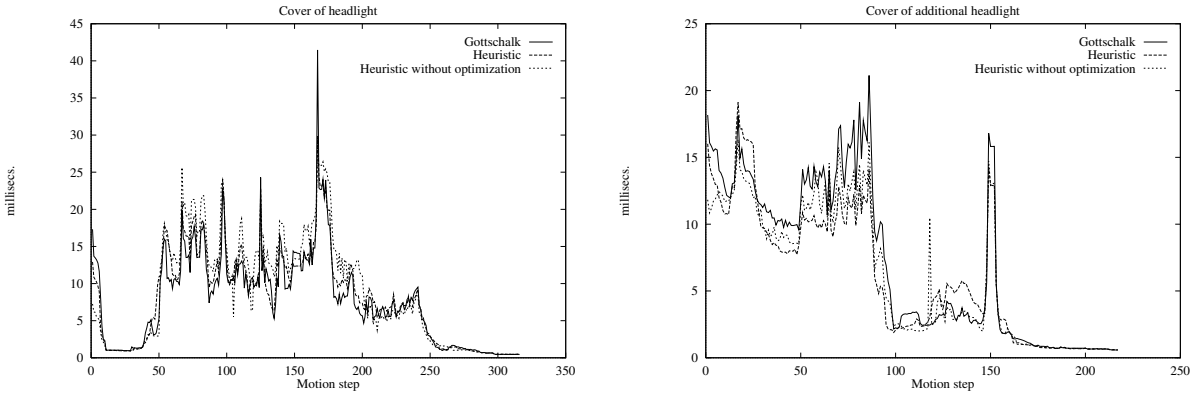

Figure 4: Comparison of query times for the BV hierarchy types

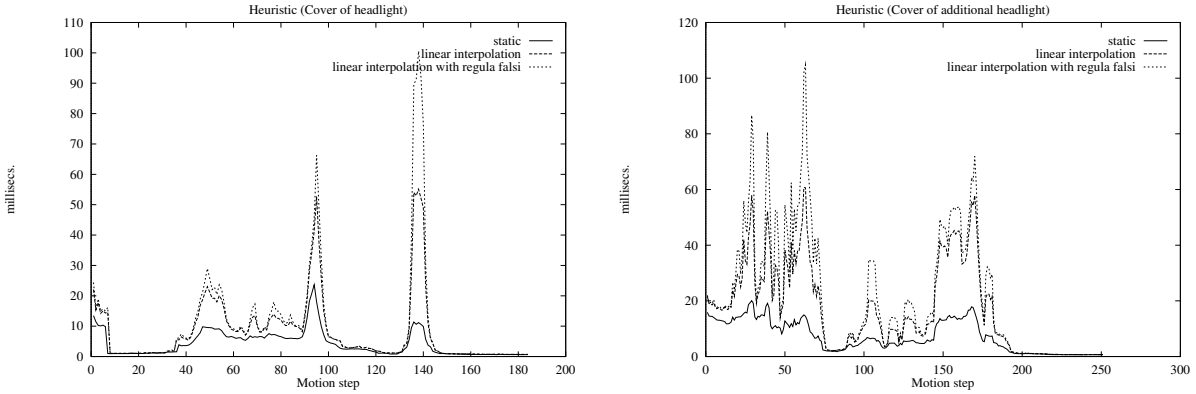

Figure 5: Comparison of query times for static and dynamic collision detection