Collision problem: Characteristics for a Taxonomy

D. Borro, J. Hernantes
Ceit
Manuel de Lardizábal 15
20018 San Sebastián, Spain

A.García-Alonso
EHU
Manuel de Lardizábal 1
20018 San Sebastián, Spain

L. Matey
Tecnun (University of Navarra)
Manuel de Lardizábal 13
20018 San Sebastián, Spain

{dborro@ceit.es, jhernantes@ceit.es, alex.galonso@ehu.es, lmatey@tecnun.es}

Abstract
The Collision problem appears within many fields. The specific characteristics that can be identified in different problems lead to the broad set of specialized algorithms that appear in the literature. This paper deals with the first step needed to address the Collision Problem Taxonomy challenge: a survey that compiles and suggests a set of characteristics that could be used to discriminate collision problems, i.e. to generate the taxonomy.

1. Introduction
The Collision problem appears within many fields of interest like Geovisualization, Urban Walkthroughs, Robotics or Simulation. Surveys within this topic sometimes compile algorithms, others use applications as the compiling criterion. This survey has a different goal: it is the first step needed to address the Collision Problem Taxonomy challenge: a survey that compiles the characteristics that will be used to discriminate collision problems, i.e. to generate the taxonomy.

Although surveys and technical papers make reference to characteristics, there is a lack of systematization. Lin and Gottschalk [1] came near to this objective when they commented some characteristics: queries (see Section 2), pair vs. nbody (Section 3), static vs. dynamic (Section 4) and rigid vs. deformable.

A taxonomy is required because visualization systems are increasing the scope of problems they must deal with. For instance, geographical visualization is not limited to high altitude flights where the collision problem can nearly be neglected; low flying requires the integration of specific collision algorithms [2]. Even more, geographical visualization data is not restricted to height-maps and terrain textures (see Figures 1 and 2). Now urban models “emerge” from the terrain and each day more demanding links to data bases increase the range of models and information that must be displayed: even physical interaction through haptics. So, visualization systems, in this and in many different areas, must be able to distinguish the characteristics of the data set within the actual navigation range. Also they should identify the characteristics of the current task requested by the user. With this knowledge, applications should apply the most appropriated algorithms.

Surveys use different classification criteria [1, 3, 4, 5], for instance, some gather algorithms for non-polygonal models [1], while others consider simple primitives [3]. Although some technical papers are not general surveys, they provide a valuable compilation of previous work [6, 7, 8, 9].

Before entering the goal of the paper, applications will be summarized.

Robotics has studied the collision problem in detail [10]. Some problems can be pre-processed, so they are no real time systems, but in most cases the problem must be solved on-line. Physical Simulation covers many engineering analysis like automobile crash simulations [11] or mechanism analysis [12].

Collisions have been often considered by Virtual Reality applications. Some simulate physical environments so that operators can visualize, explore and interact with objects of the virtual environment [13, 14]. Simple VR-CAD applications can make measures and interference analysis among objects [15]. There are applications that deal with virtual prototypes to verify assemblies or carry out maintenance tasks [16]. Collision response is used to control force feedback devices, in order to provide realistic tactile sensations [17, 18, 19].

Animation often addresses collisions, as in cloth [20, 21] and herd-flock problems [22, 23]. Collisions also play a role for achieving realism in games [24].

Each one of the following sections will deal with one characteristic but before going on, one question should be considered: whether the geometric model used to describe the scenario is a characteristic of a given collision problem. In first place, one distinction must be proposed: which geometric model is the native “source” and which one has been derived from the native one as an algorithmic requirement to solve the collision problem. A derived model is more related to the algorithms, i.e. the ways to solve the problem, than to the characteristics that describe the problem. An analysis of
geometric models requires more space than that here available, and the topic itself deserves another study.

The Boolean characteristic "rigid vs. deformable" should be discussed after considering the geometric models, so it is not here considered. For similar reasons, the paper does not discuss geometric coherence, which is used to speed up computations when dealing with parametric surfaces or convex polyhedra. On the other hand, space and temporal coherence are commented in Sections 6 and 7.

Figure 3 shows a 3D high-density environment, expression that must not be confused with a high-density polygonal image, like those that may appear when projecting surfaces made up of a large number of polygons.

2. Type of query

Four types of queries are usually considered: collision, intersection, distance and prediction. The type of query is very important to choose the collision detection algorithm, since different applications need different queries [1].

The first type of query, collision, is used when it is only needed to know whether two models are colliding or not. Many applications need only this type of binary answers (TRUE/FALSE), to make a decision. So, they do not need more precise geometrical information about the event, for instance, when avatar movements must be restricted by physical obstacles [25], or when it is only needed to know if a tool touches any part of the scene [15].

Sometimes it is interesting to find the geometric intersection between two models in contact. This type of query is important for physical simulation and animation systems, where it is necessary to know contacts exactly to calculate a suitable collision response [26, 27]. Sometimes, rougher estimations that approximate intersections are enough [17, 28].

Convex objects have unique intersections between them but, often, objects are non-convex and multiple intersections may happen.

A very common query is distance. For instance, path planners calculate the distance from mobile objects to the nearer objects in their environment. Mechanism simulation uses this type of query as an aid to achieve safe designs, using the minimum distance between parts along all possible movements as a tolerance [29]. Some authors [30] have modified the distance computation algorithm and they also compute the penetration between two objects. It is described as the minimum translation distance needed to separate them.

The last type of query is collision prediction. If object motion can be known or bounded, the next collision can be determined or predicted. This prediction allows controlling, for instance, the time step in robotic simulation [31]. The system must decide if there is, or could happen, a collision within a given time step. A new prediction of safe time step must be ready, when the current safe motion time step finishes. Flight simulators calculate distances from mobile objects to the nearer objects in their environment to predict time bounds of safe movement.

3. Number of static and mobile objects

Applications manage a diverse number of objects: from tens of objects [12] to systems with hundreds or thousands of objects [13, 14]. The problem of collision detection is different depending on the number of objects. Of course, how many move and how many remain static also makes a great difference.

If the number of mobile objects, n, is bigger than two, the problem is called n-body or multi-body [32]. There are many applications that manage this kind of problems like dynamic simulations [33] or mechanism simulations [34].

The one pair case appears in systems with a mobile object and several static objects, which is very common in geographical navigation, virtual prototyping and maintenance applications. This problem branches into two types.

In some cases, as static objects do not change relative position, they can be considered as a unique global-static object [9, 17]. On the other hand, there are applications that use thousands of static objects and one mobile object, but consider each static object independently [14]. Although there are n objects in the system, we will call this kind of applications static n-body to avoid confusions with the n-body problem (n objects in movement).

The one pair problem is a subset of the n-body problem. In this one, it is necessary to calculate the possible collision of n(n-1)/2 possible pairs. The problem, in principle, is quadratic O(n^2) with the number of objects in the scene.

Hubbard [35] defined for the n-body a broad phase and a narrow phase. The n-body problem has a combinatorial nature. There are numerous techniques, based specially on spatial partition or bounding volumes, which reject pairs of objects quickly and achieve logarithm complexity O(nlogn), or even linear complexity in the broad phase [13]. In spite of the chosen method, any approach may have a worst-case time O(n^2) [36], but these cases are not very common and only appear in very special geometric configurations.

4. Static vs. dynamic

In most cases, the Collision problem is solved using static algorithms. This means that the analysis is performed at a given instant of time -frame- and the position of each object is known at that moment. Dynamic algorithms try to find if there is a collision within a finite time interval, and to provide the precise instant when it occurred.

Some authors [37] call pseudo-dynamic problem to a subset of the dynamic algorithms. They solve the dynamic problem by means of a sequence of static problems.
So, a problem will be dynamic if only dynamic algorithms can provide the required solution. If pseudo-dynamic algorithms are required, the problem will be dynamic. Anyway, both static and pseudo-dynamic problems may share many features, like precision control. In both cases a collision could go undetected if it occurs between to discrete non-collision states.

Problems are often classified as static, because it is assumed that objects changes in position from frame to frame are very small [38]. Dynamic problems do not appear in the bibliography as often as the static ones, but they have been broadly studied [3, 39, 40, 41].

A synchronization problem may appear when collision computation is part of a control process. When collision analyses cannot achieve the sampling rate required by other modules, the problem can be alleviated using a multiprocessor architecture [18, 42, 43].

It must be taken into account that in both cases, dynamic and static, there is a close relationship with the characteristic considered in next section: accuracy.

5. Accuracy

Some applications need more computation precision than others, and a trade off accuracy against computation speed must be considered to solve the collision problem. For instance, in Robotics often is an important thing to provide a real time feedback to avoid robot accidents [10]. In these cases security is more important than accuracy.

Sometimes accuracy in collision detection depends on the available time to detect the collisions. A computation without the maximal accuracy can be allowed if the system needs to provide a quick response because the collision module rate has finished [6]. The
algorithms used for solving this type of problems are called interruptible algorithms [4] or time-critical algorithms [6, 35, 44, 45]. Their advantage is that the application has a whole control over the time dedicated to the collision detection. As soon as the maximal time is exceeded, a collision response is returned obtaining real time and a constant frame rate [15].

The consequence of providing inaccurate collision response is the possibility of producing unrealistic object behaviors. Some applications assign different priorities based on different criteria: so some collision events would have more priority than others [46]. Applications have to deal with more and more complex and large environments, so problem scale has a great influence when accuracy must be traded. However, in some applications computation accuracy cannot be surrendered and collision computations must be carried out as a post-process, i.e., they cannot be computed in “real time” [33]. Examples appear in mechanism design or maintenance simulations. In these cases, models cannot be simplified, since the geometric mistakes introduced would invalidate the digital mock-ups.

The worst case is when both, precision and computing rate, must be reached to achieve a valid application [7, 13, 37, 38]. For instance, in virtual reality, the collision detection must be in real time to keep both frame rate and virtual immersion. Sometimes this goal can be achieved trading accuracy against computing frequency, but there are problematic cases, for instance, haptics require at least a 1 kHz refresh rate.

6. Spatial coherence

Many applications take advantage of spatial coherence when object distribution in the scene is known. Algorithms based on spatial partition can be particularly efficient if objects keep away from each other. Each object will be placed in a space region and it will have a small number of neighbor objects [36]. The intersection tests will be performed only in each one of those space regions.

7. Temporal coherence

In some applications or algorithms it is possible to make use of temporal coherence. This happens when computations performed in a previous frame or time interval can be used to speed up the computations in the next one, for instance, to achieve a faster convergence in an iteration process, or to save time when sorting data.

Several areas report the use of this characteristic, like Robotics [31]. Cloth animation [21] uses temporal coherence to speed up the collision detection among cloths and human models, because the movement of each geometric element is very small along each time step.

Temporal coherence also appears in problems that deal with distance calculation among closest features (vertices, edges or faces) [13, 47, 48]. In some of these works, both, temporal and geometric coherence, are exploited to accelerate computations and to reduce the number of pairs of objects that should be tested for collision [13, 47].

8. Object/polygon density

Applications differ a lot in the number of objects and polygons that built the scenario that is visited and visualized. One parameter often depends on the other one, but many different combinations can be found in the literature: many objects each one described by a similar number of polygons (a few units, dozens, thousands, …); few objects with a large number of polygons combined with many simpler objects; etc. Geometric density is an important concept to characterize collision and other navigation problems. It is more objective than the number of geometric elements: vertices, faces, surfaces, objects, etc. Scenes with similar number of elements may have a very different spatial distribution. So, given two scenes that have the same number of objects or polygons, one scene may be a very low-density scene, while the other one may pack elements tightly and achieve a very large density.

So, algorithm fitness for a given collision application depends not only on the number of geometric elements present in the scene, but also on scene geometrical density. In general, systems with low object density are called sparse systems and the other ones, compact systems [34].

Examples of sparse systems are simulation environments, which contain thousands of objects, but whose object density is very low, because the volume filled by the objects is very low compared with the workspace volume. Some applications [13, 49] work with environments of thousands of objects and simulate their behavior setting up different rules that specify their velocities. Object density can be parameterized as the percentage of the scene volume occupied by the objects.

Scenes with a large number of objects are not necessarily a compact system. Wilson et al. [14] deal with massive scenes like energy factory models containing about 15 millions of polygons. Though this number of polygons is very large, they constitute a sparse system, since the model covers an area of hundreds of square meters. On the other hand, Gottschalk et al. [7] work with models of hundreds of thousands of polygons simulating a gravitational field directed toward the centre of a large cube of pipes, and permitting a smaller cube to fall inward, tumbling and bouncing.

Nevertheless, the clearest examples of compact systems are the virtual reality systems that simulate accessibility, maintainability, assembly and analysis of machinery models [8, 9, 17, 18, 50]. In these cases the environment may contain millions of polygons in a few cubic meters.

The difference between sparse and compact systems is the proximity among the different objects, and as a consequence they differ in the probability of occurring (or not occurring) collision events. This characteristic
also has a large influence in the efficiency of the broad phase algorithms cited in Section 3.

Collision detection methods often use bounding volumes that enclose objects or other geometric sets. In sparse systems the probability of no collision between pairs of bounding volumes is very large, compared with compact systems: think about the relationships among envelope shape objects and all the objects that are in their skin or interior (see Figure 3).

9. Movement

The method employed to detect the collision also depends on the cogniscibility or predictability of the motion: sometimes the whole motion of the objects is known before starting the analysis; sometimes it can be predicted in advance with some degree of certainty; and often it can change brusquely at any moment.

If object movement is previously specified, it can be represented like a time parametric function, as in some path planning applications [51, 32].

Some authors [45] say that in an interactive application, a collision detection algorithm must carry out two requirements. The first one, the algorithm must achieve real time, i.e., a constant frame rate as high and scalable as possible. The second one, the algorithm must allow guided movements of objects by the user, i.e., the movement is not predefined. In these cases, it cannot depend on time parameter functions, since the movement will be subject to user forces [17, 18], or dynamic laws [26, 27, 52].

Conclusions

The characteristics that this paper compiles come from a survey that has an original perspective because it addresses a new goal: to provide the foundation for a collision problem taxonomy. Each characteristic is described in one specific section. Other surveys compile algorithms; this one defines the characteristics that will make possible a thorough classification of applications and algorithms. This classification will be useful to make decisions when navigation applications must deal with different situations. They will be able to characterize a given situation and accordingly apply the best-suited algorithms.

Dr. García-Alonso research was partially supported by the Spanish Ministry of Science and Technology MCYT (Project EDULAN, TIC-2002-03141).

References

March 24-25, 2002.


