虚拟全球地形环境中实时碰撞检测和响应

罗飞雄 黄跃峰 钟耳顺

摘要：许多虚拟现实程序需要模拟物体之间的各种交互。其中一个就是保证物体不被穿透。我们提出一个新颖的算法检测虚拟相机和地形的碰撞。碰撞被检测到后，我们给出一个良好的碰撞响应。这个算法不需要为物体提前计算包围盒，所以相对于使用球树（使用球作为包围盒）的算法，节省了内存。在一个虚拟全球地形环境程序中测试，这个算法能达到实时速度。经过试验比较这个算法比使用球树的碰撞检测算法大约快几十倍。

关键字：碰撞检测，碰撞响应，虚拟环境，地形

Real-time Collision Detection and Response in Virtual Global Terrain Environments

Feixiong Luo, Ershun Zhong, Yuefeng Huang, Hui Guo, Junlai Cheng
[1]*The State Key Lab of Resources and Environmental Information System, Institute of Geographic Sciences and Natural Resources Research, CAS, China*
[2]*Graduate University of Chinese Academy of Sciences, China*
luofx@supermap.com;zhonges@supermap.com;huangyuefeng@supermap.com;
guohui@supermap.com;chengjunlai@supermap.com

Abstract

In many virtual reality applications it is necessary to simulate the interaction among objects. One of the basic requirements is to ensure a non-penetration of bodies. We show a novel algorithm for detecting collisions between camera and terrain. After collisions are detected we propose a fancy collision response. Our collision detection algorithm doesn't need precomputation and reduces memory requirement comparing to the collision detection algorithm using sphere trees. In a virtual global terrain environment application, our collision detection and response algorithm easily runs at real-time rate. Our collision detection algorithm runs faster about dozens of times than the collision detection algorithm using sphere trees.

Keywords: Collision Detection, Collision Response, Virtual Environment, Terrain.

1. Introduction

Although the non-penetration of rigid bodies is quite common in the real world, in virtual environments the contrary is true [1]. In a virtual global terrain environment we not only pay attention to collisions between objects but also focus on collisions between camera and terrain. In this paper we concentrate on collisions between camera and terrain. It is necessary to conduct collision detection and response for avoiding penetration between camera and terrain in order to make our simulation more believable and realistic.

Collision detection and response problems have been investigated for more than three decades. Many object-space collision detection algorithms based on volume hierarchies and spatial partitioning work well on rigid objects [2, 3, 4, 5]. Recently, GPU-based algorithms are increasingly used to perform collision computations. These algorithms exploit the capabilities of GPUs which involve rasterization and visibility query to check for overlaps without precomputation [6, 7, 8, 9]. These algorithms work well on rigid or deformable objects and can check collisions or self-collisions. Some researchers even use hardware-accelerated ray-intersection testing for high-performance collision detection [10]. On the other hand, most Collision response algorithms are based on physical laws [11, 12, 13, 14, 15, 16]. These algorithms explore conservation laws or constraints or impulse dynamics to achieve reasonable collision responses. For visual global terrain environments object-space collision detection algorithms require costly precomputation and GPU-based collision detection algorithms don't work well under all circumstances as some computers have limited GPU resources and couldn't afford to expensive GPU computations.

We propose a novel collision detection algorithm, Quick Oriented Terrain Collision Detection (QOTCD), to detect collisions between camera and terrain. After

detecting the collision, we present a fancy collision response to make our simulation more realistic. Our collision detection and response algorithm is implemented on a 3.0 GHz PC with NVIDIA GeForce FX 8400 GS card and is applied to a virtual global terrain environment application. It is easily able to perform collision detection and response at real-time rate.

The rest of the paper is organized as follows. We give an overview of previous work on collision detection and response in Section 2. We present our collision detection and response algorithm in Section 3. In Section 4, we analyze our collision detection algorithm comparing to the algorithm using sphere trees.

2. Previous work

In this section, we give a brief survey of prior work on collision detection and response.

2.1. Collision detection

Many object-space algorithms use spatial structures to accelerate interference computations [2, 3, 4, 5]. Typically for a simulated environment consisting of many moving objects, these algorithms use spatial subdivision algorithms or checking whether the bounding boxes of the objects overlap to reduce the number of pairwise collision checks. Furthermore, based on spatial partitioning or bounding volume hierarchies algorithms are used to accelerate accurate checking collisions.

Graphics hardware has been increasingly employed to accelerate collision detection. GPU-based collision detection algorithms involve image-space collision detection algorithms and other collision detection algorithms. Image-space algorithms are based on depth and stencil buffers and are limited to closed objects which include rigid objects and deformable objects. However, image-based algorithms require depth-buffer readbacks, which can be expensive on commodity graphics hardware [6, 7]. The GPU-based algorithms using visibility query are not restricted to closed objects and check for self-collisions [8, 9]. Some algorithms try to use hardware to improve the performance of collision detection [10].

2.2. Collision response

Based on conservation laws collision response algorithms consider physical equations and energy conversation [11, 12, 13]. These algorithms are based on conservative laws of momentum, conservative law of kinetic energy, the relative velocity at post-collision position, elasticity, friction and mass-spring. An experiment in which the sensitivity of subjective presence to varying collision response parameters in examined is described [11]. Based on constrains collision response algorithms are proposed for animated cloth simulation [14]. Some algorithms use impulse dynamics for collision response [15, 16]. A simple algorithm is well suited to modeling physical systems with large number of collisions, or with contact modes that change frequently and is faster than constraint-based methods [15]. However, it cannot be used for deformable object collisions. A few collision response algorithms for deformable objects in virtual surgery are presented [17, 18]. A hybrid approach in which the finite element model and a particle model are used to simulated flexible dynamics of the duct and catheter respectively [19]. Another response algorithm which adds and subtracts virtual particles as need is proposed [20].

2.3. Collision about terrain

An algorithm using sphere trees to check collisions between camera and terrain is presented [21]. It builds spheres for camera's motion and each terrain tile in a virtual terrain environment during preprocessing. A rapid algorithm, Quick Oriented Terrain Algorithm (QOTA), is proposed to support terrain following [22]. This algorithm gives our collision detection algorithm, QOTCD, an inspiration. A technique for providing automatic animation and collision avoidance of arbitrary objects is used to a cloud movement over terrain application [23].

3. Quick Oriented Terrain Collision Detection and Response

In this section, we present our novel collision detection and response algorithm. Our approach for collision detection and response aims at avoiding collisions before they occur.

3.1. Global terrain visualization

In order to balance the overload between visual fidelity and real-time rendering the global terrain, visualization usually relies on multi-resolution hierarchy technology. This technology divides the whole global terrain into different levels and tiles in terms of latitude and longitude. The concise scheme of global terrain visualization is shown in Fig 1. The whole terrain data is also divided and distributed in small blocks stored in files during preprocessing. Each tile whose structure is Regular Square Grid (RSG) or Triangulated Irregular Network (TIN) constructs meshes using the vertices of its corresponding terrain block stored in a file. Global terrain meshes are constructed by linking meshes of all tiles. When the camera moves, terrain tiles outside the frustum are unloaded from memory and replaced with new ones. When the camera gets close to the globe, precise terrain tiles replace coarse terrain tiles. In contrary, when the

camera is far away from the globe, coarse terrain tiles are rendered instead of precise terrain tiles. It is necessary to make sure that there are always geometries in the camera's frustum.

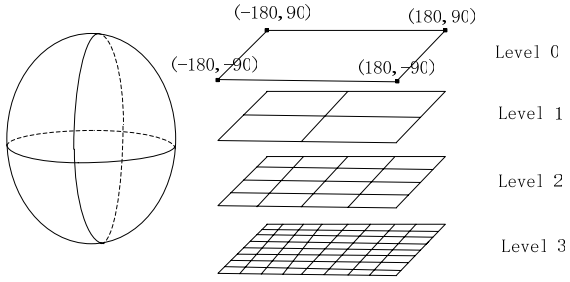


Fig 1. Global terrain visualization

3.2. Quick Oriented Terrain Collision Detection

While a terrain model specifies a surface in 3D space, it is essentially 2D in nature and can be viewed as 2D map marked with heights. More specifically, it is assumed that like other 3D models, a terrain model is composed of polygons [22]. While a terrain model may contain tens of thousands of polygons, a ray from the camera to the globe center, Oriented Ray, intersects a single polygon. Oriented Ray intersects the surface of globe at a point called Intersecting Point which is the closest point to the camera among points on the global surface. The overview of QOTCD is shown in fig 2.

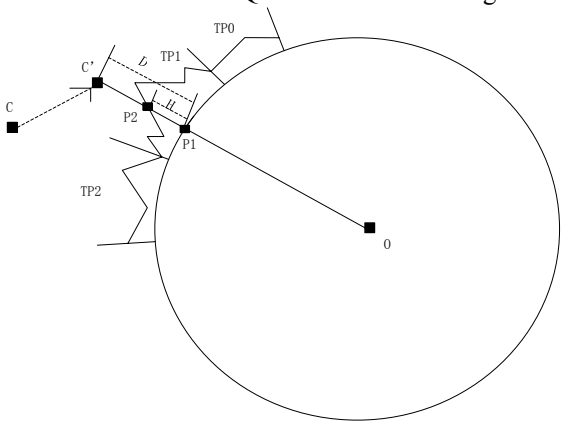


Fig 2. The overview of QOTCD: C denotes the current position of the camera; C' denotes the new position of the camera; O denotes the globe center; P1 denotes Intersecting Point and P2 denotes the point where Oriented Ray and the terrain intersect; TP0, TP1, TP2 denote three terrain tiles; H denotes the height of Intersecting Point. D denotes the distance from the camera to Intersecting Point. If D > H, there is no collision; if D <= H, there is a collision between the camera and the terrain.

When the camera reach a new position, it is required to check whether the camera penetrate the terrain. Determining whether a collision occurs is divided into three steps. First of all, we translate the new position from the standard three-dimensional space (which OpenGL/D3D operates in) into the sphere space which the global terrain data is in. We calculate the new value of the new position in the sphere space using formulary 1:

$$\begin{cases} L = a\tan(\dfrac{x}{z}) \\ B = a\sin(\dfrac{y}{\sqrt{x^2+y^2+z^2}}) \\ R = \sqrt{x^2+y^2+z^2} \end{cases} \quad (1)$$

Where (x, y, z) is in the standard three-dimensional space and (L, B, R) is in the sphere space. L, B, R denote latitude, longitude, radius. It is obvious that the longitude and latitude of Intersecting Point are L and B respectively. In addition the distance between the camera and Intersecting Point is calculated by subtracting the globe radius from R.

Next the terrain height of Intersecting Point is computed according to its longitude and latitude. We determine which terrain tiles contain Intersection Point. Every tile has a range composed of minimal (maximal) longitude, minimal (maximal) latitude. We quickly query the corresponding tiles by comparing the longitude and latitude of Intersecting Point with the range of terrain tiles in purely two dimensions. Additionally because Intersecting Point is the center of the camera's view, the corresponding terrain tiles must be in rendering terrain tiles array. We accelerate finding the corresponding terrain tiles through reducing the range from all terrain tiles to rendering terrain tiles. We select the max level terrain tile among terrain tiles which contain Intersecting Point as Intersecting Terrain Tile. Although the structure of Intersecting Terrain Tile may be TIN or RSG, the terrain block with respect to Intersecting Terrain Tile usually stores vertices in a good order. If Intersecting Point is one of vertices accidently (we give a enough small error) of the terrain block, we directly get the terrain height of this vertex; if Intersecting Point is in a quadrangle composed of four vertices we get maximal terrain height of four vertices; if Intersecting Point is in a column or a row, we get the maximal terrain height of two nearby vertices in the same column or row. The distribution of Intersecting Point in a 17×17 terrain block is shown in fig 3.

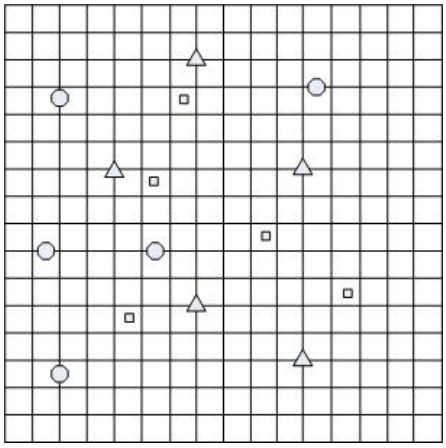


Fig 3. The distribution of Intersecting Point in a 17 x 17 terrain block: △ means Intersecting Point overlaps with a vertex completely; ○ means Intersecting Point is in a row or column; □ means Intersecting Point is in a quadrangle.

Finally the distance between the camera and Intersecting Point and the terrain height of Intersecting Point are known, we compare two values to determine whether a collision occurs. If the distance is greater than the terrain height, a collision doesn't occur. In contrary, if the distance is less than the terrain height, a collision occurs.

3.3. Camera-terrain collision response

After a collision has been detected, a fancy collision response is required to make our simulation more realistic. Collision response usually considers physical laws including conservative laws of momentum, conservative law of kinetic energy, elasticity and friction and so on. In global terrain environments, the terrain is a static object without velocity as well as acceleration while the camera may or may not have velocity or acceleration. Since only the camera is motive, it is not necessary to consider conservative laws for collision response. As a result the terrain and the camera are rigid bodies, it is not necessary to take account of elasticity for collision response. As our response aims at avoiding collisions, it is not meaningful to consider friction between the camera and the terrain for collision response. According to the above analysis, we have to deal with two following situations respectively:

1) Camera is static.
2) Camera is motive.

3.3.1. Camera is static. When the camera has no velocity and acceleration, our collision response is to simply stop and not allow the move when a collision is detected.

3.3.2. Camera is motive. The movement of the camera can be composed to a linear velocity and an angular velocity relative to the globe center. We discuss two kinds of movement separately. If the camera has a linear velocity, the method presented by [24], sliding along the terrain, is used. It is shown in Fig 4.

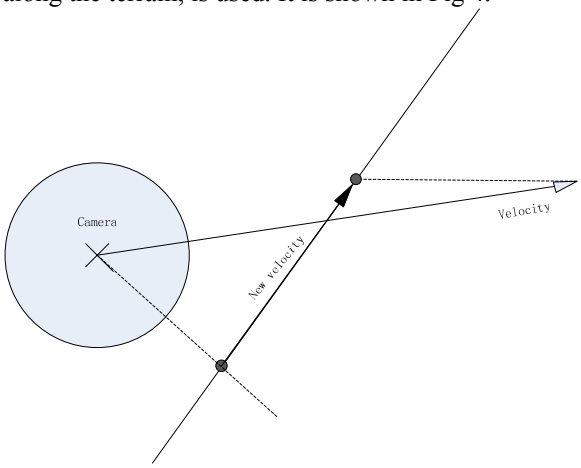


Fig 4. Camera sliding along the terrain

For the camera has an angular velocity, we only heighten the camera without changing the angular velocity when a collision is detected as shown in Fig 5. Since Acceleration is similar to velocity, we treat acceleration in the same way with velocity.

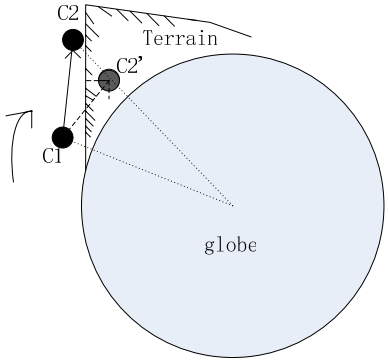
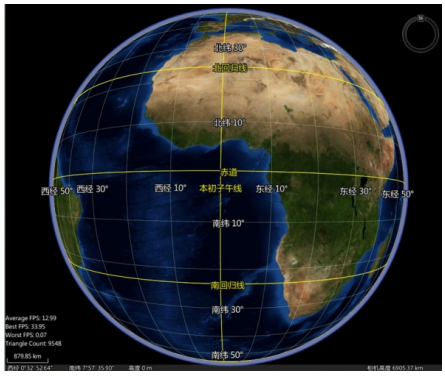


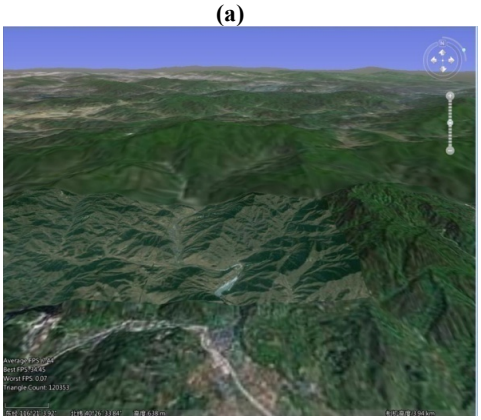
Fig 5. Heightening the camera without changing angular velocity: ω denotes the camera's angular velocity; C1 denotes the current position of the camera; C2' denotes the position of the camera without collision response; C2 denotes the position of the camera with collision response avoiding penetrating the terrain.

4. Implement and performance

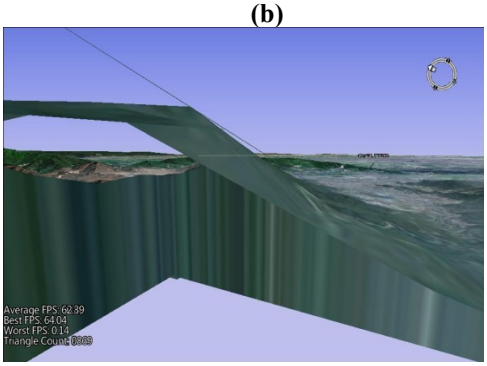
We have implemented our collision detection and response algorithm using OpenGL on a PC with a 3.0 GHz Intel Pentium CPU, an NVIDIA GeForce FX 8400 GS GPU, and 2 GB of main memory. The PC is running on the Windows Server 2003 operating system. We develop a virtual global terrain environment application with GTOP30. GTOP30 is a global digital elevation mode (DEM) with a horizontal spacing of 30 arc seconds (approximately 1 kilometer) and can be downloaded from the website (http://www1.gsi.go.jp/Geowww/globalmap-gsi/gtopo30/gtopo30.htm). We add some images to cover the terrain in order to make the terrain more true to nature. Fig. 6 shows several snapshots of our application.



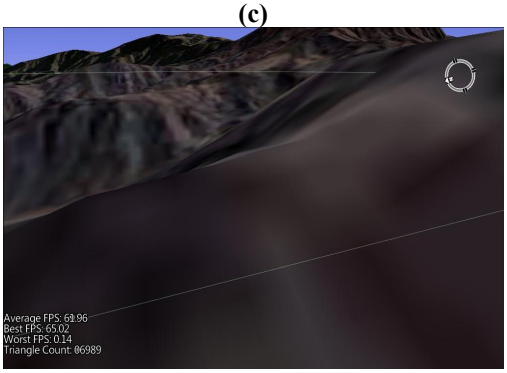
(a)



(b)



(c)



(d)

Fig 6. Shots from our virtual global terrain environment application: Fig (a) and Fig (b) are two snapshots of our application. In Fig (c) there is no collision detection and response, the camera penetrates the terrain, the terrain seems to be cracked; In Fig (d) there is collision detection and response, the camera couldn't penetrate the terrain, the terrain is more realistic.

In order to compare to other collision detection algorithms, we implement the algorithms using sphere trees used by Susana [21] in our application. However, there is a little change. We neglect collisions during each frame, so we build a sphere for the camera instead of the motion of the camera. The total cost function for interference detection can be formulated as the following equation [23]:

$$T = Nv \times Cv + Np \times Cp \tag{2}$$

where

T : total function for interference detection

Nv : number of bounding volume pair overlap tests

Cv : cost of testing a pair of bounding volumes for overlap

Np : is the number primitive pairs tested for interference

Cp : cost of testing a pair of primitives for interference

In our application, suppose there are n terrain tiles and each terrain tile has $m \times m$ vertices. It is obvious that the time complexity of the algorithm using sphere trees is $O(n, m^2)$. Table 1 summarizes the time complexity of QOTCD is $O(n)$. As the comparison operation is very fast, the time complexity of QOTCD $O(n)$ is nearly $O(1)$. An experiment (we set m =17 and the radius of the sphere covering the camera is 50 meters) proves our prediction as shown in Fig 7.: the collision time of the algorithm using sphere trees increases with the number of terrain tiles in a nearly linear relationship, the collision time of QOTCD is nearly constant with the increase of the number of terrain tiles. The performance of QOTCD is about dozens of times faster than the algorithm using sphere trees. That the maximal collision time of QOTCD is less than 1 millisecond contributes to run our application at real-time rate.

Our algorithm has several limitations. Firstly our algorithm directly gets the maximal terrain height of nearby vertices at cost of certain accuracy. Secondly our algorithm does not return any overlap information. Thirdly our algorithm is only applicable to virtual global terrain environments up to now.

Table 1. Number of operations per calculation for QOTCD

Step 1: Translating the camera position from the standard 3D space into the sphere space and computing the distance from the camera to Intersecting Point	3 squares, 1 square root, 2 divisions, 1 arc sine, 1 arc tangent, 1 subtraction.
Step 2-1: finding Intersecting Terrain Tile	4 comparisons
Step 2-2: Reading the terrain height of Terrain Point from Intersecting Terrain Tile	2 divisions, 0～4 comparisons
Step 3: Comparing the distance with the terrain height	1 comparison
Total operations	4*n+5～4*n+9 comparisons, 3 squares, 4 divisions, 1 square root, 1 arc sine, 1 arc tangent, 1 subtraction

5. Conclusions and future work

We have proposed an efficient collision detection and response algorithm for virtual global terrain environments. Our collision detection algorithm, Quick Oriented Terrain Collision Detection, comparing to the algorithm using sphere trees indicates up to an order of magnitude improvement in detecting efficiency for our application. Our collision detection algorithm neither requires the precomputation of bounding volume hierarchies nor the extra memory to save bounding volume hierarchies. Additionally we give a fancy collision response considering two situations. Our collision detection and response is readily to make our application run at real-time rate.

There are many avenues for future research. We would like to refine our algorithm for greater efficiency and more accuracy. We would like to do research on collisions between objects besides collisions between camera and terrain in our application. With the development of GPUs, we would like to explore the new programmability features of GPUs to further improve the performance of collision detection and response algorithms.

References

[1] L. Kavan, "Rigid Body Collision Response", *In proceedings of the 7th Central European Seminar on Computer Graphics*, 2003.

[2] B. Naylor, J. Amanatides, W. Thibault, "Merging bsp trees yield polyhedral modeling results", *In Proc. of ACM Siggraph*, 1990, pp. 115-224.

[3] S. Gottschalk, M. Lin, "OBB-Tree: A hierarchical structure for rapid interference detection", *Proc. of ACM Siggraph'96*, 1996, pp. 171-180.

[4] P. M. Hubbard, "Interactive collision detection", *In Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, 1993.

[5] J. Klosowski, M. Held, "Effcient collision detection using bounding volume hierarchies of k-dops", *IEEE Trans. on Visualization and Computer Graphics*, 1998, 21–37.

[6] G. Baciu, S. K. Wong, H. Sun, "Recode: An image-based collision detection algorithm", *Proc. of Pacific Graphics*, 1998, pp. 497-512.

[7] M. Teschner, S. Kimmerle, "Collision detection for deformable objects", *Computer Graphics Forum*, 2005, 24(1), 61-81.

[8] N. K. Govindaraju, M. Lin, "Quick-cullide: Fast inter- and intra-object collision culling using graphics hardware", *In IEEE Conf. on Virtual Reality*, 2005.

[9] N. K. Govindaraju, I. Kabul, M. Lin, "Fast continuous collision detection among deformable models using graphics processors", *Computers and Graphics*, 2007, 31(1), 5-14.

[10] Sung-Soo. Kim, Seung-Woo. Nam, In-Ho. Lee, "Hardware-Accelerated Ray-Triangle Intersection Testing for High-Performance Collision Detection", *Proceedings of Third International MIRAGE Conference*, 2007, pp. 70-81.

[11] S. Uno, M. Slater, "The Sensitivity of Presence to Collision Response", *Proc. Of IEEE Virtual Reality Annual International Symposium*, 1997.

[12] B. Geiger, "Real-Time Collision Detection and Response for Complex Environments", *Computer Graphics International 2000 (CGI'00)*, 2000.

[13] M. Moore, J. Wilhelms, "Collision Detection and Response for Computer animation", *Computer Graphics*, 1988, 22(4), 289-298.

[14] D. Baraff and A. Witkin, "Large steps in cloth simulation", *SIGGRAPH 98 Conference Proceedings*, 1998, pp. 43-54.

[15] B. Mirtich, J. Canny, "Impulse-based simulation of rigid bodies", *Symp. On Interactive 3D Graphics*, 1995.

[16] F. Policarpo, A. Conci, "Real-Time Collision Detection and Response", *XIV Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'01)*, 2001.

[17] V. Vuskovic, M. Kauer, G. Székely, M. Reidy, "Realistic force feedback for virtual reality based diagnostic surgery simulators", *Proc. ICRA '00, IEEE International Conference on Robotics and Automation*, 2000, pp. 1592-1598.

[18] B. Lee, D. Popescu, and S. Ourselin, "Contact modelling based on displacement field redistribution for surgical simulation", *Medical Imaging and Augmented Reality: proceedings of the 2nd International Workshop (MIAR 2004)*, 2004, pp. 337-345.

[19] C. Basdogan, C. Ho, M. A. Srinivasan, "Virtual environments for medical training: graphical and haptic simulation of laparoscopic common bile duct exploration", *IEEE/ASME Transactions on Mechatronics*, 2001.

[20] O. Etzmus, B. Eberhardt, M. Hauth, W. Straser, "Collision Adaptive Particle Systems", *The Eighth Pacific Conference on Computer Graphics and Applications*, 2000.

[21] Susana López, Borja Fernández, "COLLISION MODULE INTEGRATION IN A SPECIFIC GRAPHIC ENGINE FOR TERRAIN VISUALIZATION", *Proceedings of the Eighth International Conference on Information Visualisation (IV'04)*, 2004.

[22] J. W. Barrus, R. C. Water, "QOTA: A Fast, Multi-Purpose Algorithm for Terrain Following in Virtual Environments", *Proceedings of the second Symposium on Virtual Reality Modeling*, 1997, pp. 59-64.

[23] P. K. Egbert, S. H. Winkler, "Collision Free Object Movement Using Vector Fields", *IEEE Computer Graphics and Applications*, 1996, 16(4), 18-24.

[24] K. Fauerby, "Improved Collision detection and Response", http://www.peroxide.dk, 2003.

[25] H. Weghorst, G. Hooper, D. Greenberg, "Improved computational methods for ray tracing", ACM Transactions on Graphics, 1984, pp. 52–69.

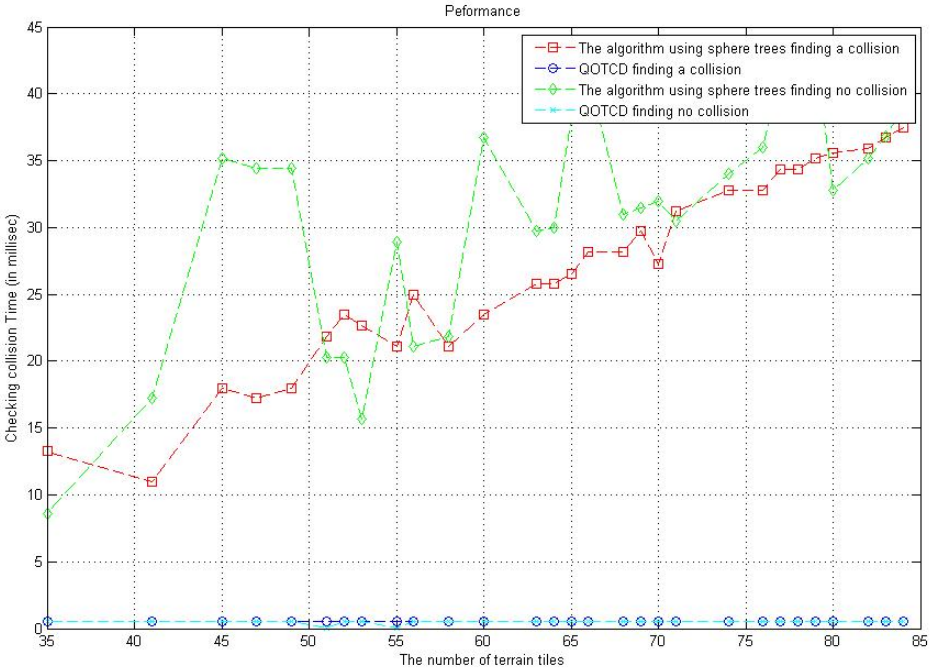


Fig 7. Performance comparison between QOTCD and the algorithm using sphere trees