COLLISION MODULE INTEGRATION IN A SPECIFIC GRAPHIC ENGINE FOR TERRAIN VISUALIZATION

Susana López slopez@euve.org
Borja Fernández bfernandez@euve.org
Arkaitz Glz. de Arrieta agarrieta@euve.org
José Daniel Gómez de Segura jdgsegura@euve.org
Alex García-Alonso agalonso@si.ehu.es

Virtual Reality Department

European Virtual Engineering (EUVE)

Vitoria – Spain

Abstract

This paper presents the design and implementation of a collision management module. This module is part of a specific graphic engine that manages large size terrain data. This engine has been used for example in a Basque Country terrain demo with a database up to 2.5Gb and texture resolution of 1024x1024 (3mts/pixel). As data textures are increasing resolution it seems convenient to allow near-terrain navigations. This behavior makes more complex the collision problem while navigating a large database. The main purpose of this project is the integration of a new collision module without any remarkable decrease in the performance of the engine. This work comprises the following research topics: preprocessed collision data structure definition, collision data management algorithm in execution time, collision detection algorithms and collision response algorithm. It has provided more flexibility in the interaction of the engine with the user, and more realism overflying terrains.

Keywords: graphic engine, collision detection, terrain, visualization.

1. Introduction

The work presented in this paper is closely related to a specific terrain visualization engine in real time [1]. Some of the basic features of this engine are the following ones.

- Dynamic geometry and textures load.
- Different levels of texture (only the needed resolution is loaded).
- Multitexture.
- 2D information integrated.
- User interaction through mouse and joystick.

The new module developed in the engine emerged from the necessity of managing collisions with the terrain. Before, the minimum level the user could descend to was limited, and the implementation of a better collision system did not report any extra benefit. That was due to the lack of resolution in the textures. Now, the texture management problem solved, new possibilities have appeared. It is interesting to have nearly full mobility for the user, so collisions should be studied again. The way of managing them is not evident because of the database size.

The main object of this project was the integration of a collision module in a graphic engine with large geometrical complexity without performance reduction. This module should be designed and developed with this purpose.

Not to reduce the performance, there are two important factors. Firstly, the data structure used in the algorithms was recommended to be preprocessed before execution time and stored in a fixed format. Secondly, the collision algorithms, what means, how to implement the collision detection, the collision reaction and the way the data should be obtained in execution time.

2. Background

The collision problem is very common in graphical applications. The object of this paper is to show how the collision problem can be solved efficiently in certain types of environments with terrain data meshes, and how existent algorithms were modified and used.

In general, 3D detection algorithms can be divided in four approaches: space-time volume intersection, swept volume interference, multiple interference detection and trajectory description. [2]

Generally, the complexity of the collision problem is due to the data required in the algorithms and the number of times an algorithm should be called.

There are many data structures proposed. For big amount of data, the most useful ones are the hierarchical structures, like Octrees [3], Octree-like structures [4], BSP-trees [5], Brep-indices, tetrahedral meshes and Regular grids. The hierarchical structures may apply for terrains.

Dealing with terrains, other problems exist.

On one hand, terrains are formed by a collection of triangles that should be sorted into the hierarchical structure. When the mesh is not regular, some conditions are necessary for an efficient separation. Some approaches have been done related to terrain visualization [6].

On the other hand, the algorithm implementation is very important too. When collisions are calculated in interactive systems, algorithms should be very fast. In our case, the precision of the collision is not as important as the spent computing time.

There are many algorithms for polygon collision depending on the nature of the collision objects and the intersection test employed (spheres, bounding boxes, polygonal shapes…) [7].

The swept-sphere collision detection and collision response applied here are based in Fauerby work [8]. The clearly difference with our work is that we have built a two levels algorithm, and just one of the levels is based on his work. Besides, in our case the managed data are specifics for dealing with complex terrain data.

3. Proposed Solution

The developed collision module is explained in detail in this chapter. The following topics are developed.

- *Camera -terrain collision detection* method is divided in two parts.
  - ***Global motion test*** → This method fulfils first approaches to the camera-terrain collision problem. This collision test is performed between the sphere that covers a terrain patch and a sphere that contains the camera motion (it covers the volume swept by the camera from current position to future position). It is shown in figure1.
  - ***Swept test*** → This second method is executed in terrain patches where first test suspects collision may happen. It consists of verifying if the camera trajectory (now modelled as a swept sphere between the current position and future position) collides to any triangle inside given terrain patch.
- *Collision data structure* and *Data access algorithm* for collision tests are also important factors for this module efficiency.
- And finally, *Collision reaction method.*

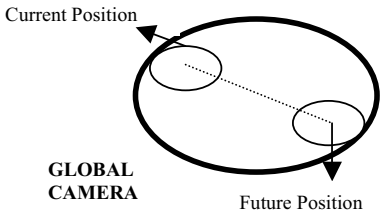

Fig.1 Sphere camera composition for *global motion test*

3.1. Collision data Structure

In this module, Collision data structure is the first subject to take care of. A large information database is employed, so, preprocessed and fixed information is very important. It is done to obtain the needed data quickly in execution time.

In this engine, the whole terrain database is distributed in small pieces. Each piece is stored in a particular file structure (.GEO). Some information was added to those files for managing collisions.

Although the collision module has just to deal with the terrain data inside the viewing frustum, and the complexity is reduced thanks to the division of the geometrical data in different files, special collision data is necessary. It prevents the system from testing all the triangles from all the files inside the viewing frustum. (The complexity is already high enough so as to reduce the performance of the engine)

The structure proposed and developed to deal with collisions is the following one.

Collision data within a file is divided into two different structures. Each one contains a different kind of data. If the test applied to the first one finds no collision, the second structure data, which is more complex, is not necessary.

These are the proposed data structures.

- Structure based on bounding spheres.
- Structure based on triangle information.

3.1.1. Structure based on Bounding Spheres

Each piece of terrain (each file) is approximated by a bounding sphere, with the following information.

- Sphere center
- Sphere radius

This sphere, called level 0 Sphere or main sphere of the file, contains the whole terrain inside.

This sphere can be subdivided into smaller spheres in lower levels. With this approximation a hierarchical bounding sphere tree is built. The terminal spheres of each branch have a pointer to the triangle structure data they contain.

There is not a fixed shape for the tree. Neither the levels nor the number of triangles contained in the last levels spheres are fixed, but a stop condition exists for preventing from weird and inefficient shapes. This condition is determined by two parameters as it will be explained: the maximum number of levels permitted in the tree (depth of the tree), and the maximum number of triangles referenced by a low level sphere.

An example of one file data structure is shown in the following pictures (figure 2).

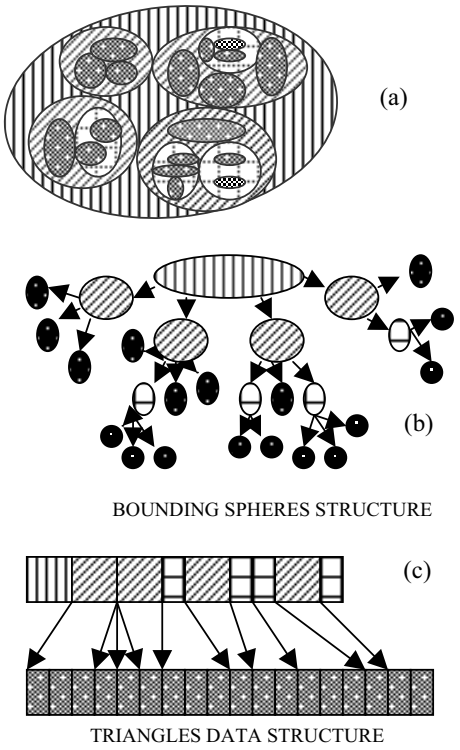

Fig.2 (a) Scheme containing spheres; (b) Hierarchical tree; (c) Memory structure.

One additional property of this structure is the cache memory optimization. The whole tree is serialized. That means the way of traversing is always in one direction.

Each sphere node has the following information.

- *Bounding Volume information* (Sphere center, Sphere radius).
- *Number of subvolumes* (children).
- *Offset to the collision data* where triangles are placed. Only leaf nodes have none null pointer here. The other nodes do not need it.
- *Offset to the next node of the same level*.

3.1.2. Triangles information

The lowest bounding volumes of each branch have the offset to the triangle structure contained in them. If it is necessary to access the triangles, it means a collision could happen in that piece of the terrain. So, the additional information needed is the following one.

- Number of triangles contained in the sphere volume.
- For each triangle:
  - The coordinates of the three vertices of the triangle.
  - Coefficients of the plane that the triangle is contained in (A,B,C,D)

3.2. Collision data access algorithm in execution time

The following algorithm accesses structured data explained before and executes corresponding algorithm depend on the hierarchical level tested (figure 3).

It is applied to all hierarchical trees built from the data files of the pieces of terrain inside the viewing frustum.

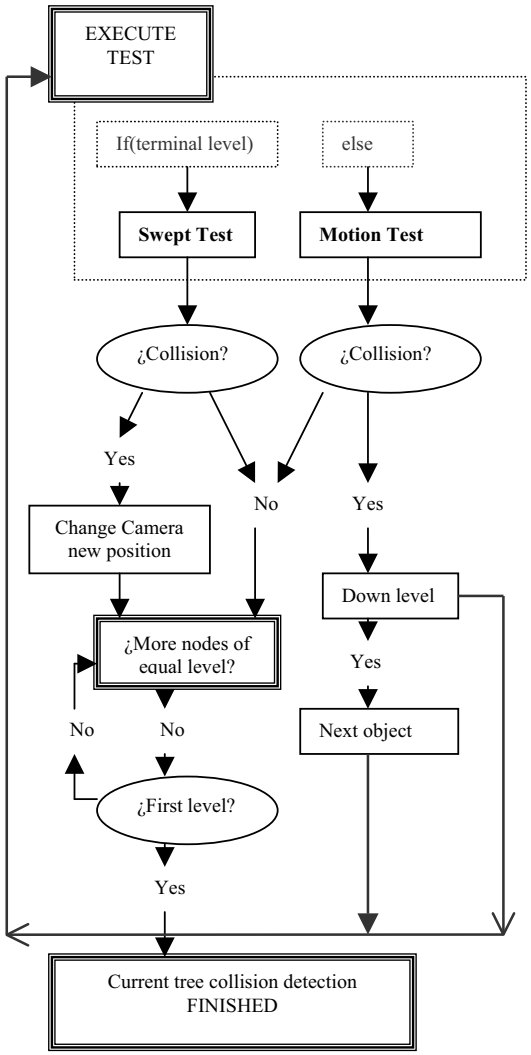

Fig.3 Collision detection algorithm

Continuing with the tree example of the figure 2, to determine if a collision occurs in a part of the terrain piece, the tested nodes are shown in figure 4 (a), and the access to the data in figure 4(b).

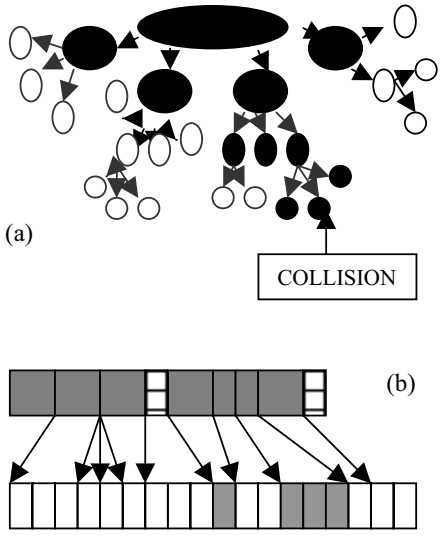

Fig.4 (a)(b) Scheme for a concrete collisionGlobal Motion Test

Once the collision is found, the algorithm carries on testing nodes till the last one of trees list. Just one value changes after each collision, the new camera future position which is obtained in the collision reaction algorithm.

The classical sphere-sphere collision test is used for all the levels except the terminal ones of each branch of the tree.

The two spheres involved in the test are formed in the following manner.

- One sphere is obtained from the bounding sphere tree. The sphere employed depends on the level of the tree.
- The other sphere is formed with some camera data. It is shown in Figure 1.

3.3. Swept Test

This test is used with the terminal levels of the tree. If the data access algorithm arrives at this point, it means that the other collision tests with the upper level were successful.

For this test the employed data changes. Now, it is necessary

- Triangle data structure information.
- New sphere model for camera. Instead of using the sphere model shown in picture 1, here another approach is used. Camera current position and camera next position are modelled as two spheres of given radius.

The principle of this test is to find if there is a sphere whose centre, placed in the line formed between the initial and final spheres collides to any given triangle.

In this case two steps are employed.

- First step → Find if there is a sphere that collides with the plane one triangle is contained in. The collision point is obtained.
- Second step → Verify if the obtained collision point is inside the given triangle.

If no collision point is obtained in the first step, the second one is not necessary.

3.4. Collision reaction

Once the collision point is obtained, the collision detection finishes. The following phase is the collision reaction. There are many ways to implement it but the most suitable one for terrains is sliding over them.

The camera direction changes and a new next position is obtained.

There is nothing special in this part of the module, so a brief graphical explanation in the figure 5 is enough.

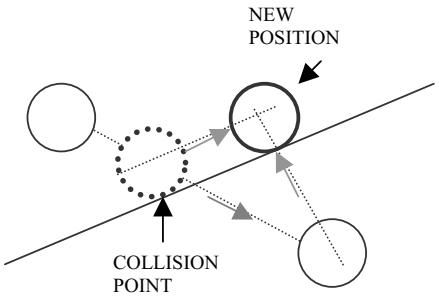

Fig.5 Collision reaction scheme

4. Discussion

Developed approaches and modified algorithms mentioned in the previous chapters are explained here.

4.1. Data structure

- ***Terrain approach as Bounding Spheres.***

  According with .GEO file data structure, hierarchical trees could be built by bounding spheres or by bounding boxes. (There are more possibilities but the most appropriates are these two). Depending on the shape of the intersection objects the algorithm efficiency changes.

  A *Bounding Sphere* was chosen because:

  - Camera is also modeled like a sphere and the sphere-sphere intersection algorithm takes less time than a box-sphere intersection test.

- Aspects related to axis were not necessary.

o ***Two separated data structures.***

The algorithm efficiency is due to the required calculus and the data used to perform them.

It was possible, as some referenced papers before explain, to achieve the triangles information once for each level of the tree, storing all the data in the same place. It is not right for collisions problem.

Structure division was made because:

- For collision problem all triangle information is not necessary for each level. Just the terminal level of each branch needs it.
- If all information is stored together and algorithms fail, jumps through the memory are needed. In this case a better management of the memory cache is obtained.

o ***Stop Condition building a hierarchical tree.***

These terrain meshes are irregular grids, and it is not easy to find an efficient division.

A fixed number of levels are not practical because the mesh can be composed by a big area with just one triangle and a small one with a high number of triangles. Redundant information will increase operations and memory accesses.

The stop condition designed and implemented here is based in two parameters:

- *The number of triangles referenced by each terminal sphere of each tree branch* (N). When the number of triangles in a bounding sphere is less than a fixed N, stop building down levels in that part of the tree.
- *The maximum number of global levels of the tree* (M).

With these two values the system arrives to a compromise. The worst case happens when algorithm arrives to the maximum level with a number of triangles higher than N. In general cases, with the tested terrains, the first parameter is enough.

In the demo explained before, the fixed values adopted are N = 20 and M = 20.

4.2. Data obtaining algorithm

Efficient cache memory use is an important achievement in our algorithm.

o *Cache memory use*

The algorithm has been designed to access data in a sequential increase mode. Being data structured in this manner pre-fetch can be used. (Ask to cache memory data blocks before they are needed). In this way, cache errors are minimized. (Cache errors = asked blocks that are not in cache memory and that should be asked to the secondary memory)

4.3. Global motion test

As it was commented, the algorithm used for the first collisions approach is the classical sphere-sphere intersection test.

o *Spheres definition*

One of the spheres contains the terrain patch in study. It depends on the node of the bounding sphere tree.

The other sphere contains the camera. Even camera is defined in a point (x,y,z), it also has some values associated: *near plane* and *far plane*. Before the first one, and after the second one, nothing is drawn. Not to allow objects between near plane and camera form part of the collision method (which will disturb the application letting see the user how those polygons intersect against the near plane). A sphere with a radius bigger than the distance between the camera and near plane should be defined. One approximation is to consider that sphere encloses the current camera position and the camera next position.

o *Distance employed*

Instead of using distances, squared distances were used to reduce the calculus. With this approach, some expensive instructions are saved.

4.4. Swept test

This is the second algorithm to detect collisions. Generally four steps are necessary to verify if a sphere collides with a triangle. First, if the camera swept sphere collides to the plane which contains the triangle is tested. If that is successful, then it is tested if obtained collision point is inside the triangle, on any edge of it, or on any vertex. In our method, collision points are considered only if they are inside the triangle. That reduces the complexity in execution time. It works properly because of the terrain triangle mesh structure.

There is an important feature in this algorithm. When a collision point is detected, the collision trajectory is modified, but the algorithm still tests the rest of triangles of the mesh. Another new collision point allows us to correct those possible precision errors of the previous one.

5. Results

The most significant result obtained from the collision module integration in the engine was the given flexibility and visual aspects advance. Now terrains are accessible at heights not allowed before, as can be seen in figure 6. Previously, flights were not allowed bellow the highest geographical point (Figure 6-a), now the system allows ground following flights.

The collision management module requires adding the data structures we have described. However, it must be noted that the overall memory increment is negligible compared with the amount of memory required by the other data structures loaded in the system at any given instant of time through any simulation, i.e. textures. Even more, being able to fly at lower altitudes opens new researching goals, which would have no interest if navigation were restricted to high altitude flights.

(a)

(b)

Fig. 6 Shots from Basque Country demo program. Without collisions (a), and with collision control(b).

The main integration object was not to reduce the engine performance. In a machine with 2 processors Pentium 4, 1.7 GHz, and 1 GB RAM, there was not a single frame per second rate reduction (230fps) . That is due to the area managed with collisions is smaller, because when a real collision may happen the camera is close to the terrain and the area inside the frustum is smaller.

6. Conclusions

The new module was integrated in the system successfully. The performance of the engine did not decrease.

It provides the user more movement flexibility overflying terrains (It is very useful now the texture manager allows multitextures and different levels of detail).

As collisions are a classical problem in graphics, some methods were adapted to this particular case, explaining why some assumptions were adopted, and why some typical methods were modified. The other important development explained in this paper is the employed collision data structure and data flow in execution time algorithm.

The integration of surface geometry (buildings,..) with terrains in the engine is one of the future works of this engine. These collision algorithms were designed to manage also those future particularities.

7. Bibliography

[1] FERNANDEZ, B. "Interactive demo: OpenGLfly, versión Euskadi", EUROGRAPHICS 2003, ISSN1017-4656.

[2] P. JIMENEZ, F. THOMAS AND C. TORRAS "3D Collision Detection: A Survey". Computer & Graphics vol. 25, num2, pp 269-285. April 2001.

[3] HAMADA, K., AND HORI, Y. "Octree-based approach to real-time collision-free path planning for robot manipulator". ACM96-MIE (1996), pp 705-710.

[4] BANDI, S., AND THALMANN, D. "An adaptive spatial subdivision of the object space for fast collision detection of animating rigid bodies". Eurographics´95 (Aug 1995). Maastricht, pp 259-270.

[5] NAYLOR,B. F., AMATODES, J.A., AND THIBAULT,W.C. "Merging bsp trees yields polyhedral set operations". Computer Graphics (SiGGRAPH90 Proc) (Dallas (TX), May 1990), vol.24, pp. 115-124.

[6] VISGRAF Laboratory, "QLOD: A Data Structure for Interactive Terrain Visualization". TR-01-13, IMPA, 2001.

[7] TOMAS MÖLLER,ERIC HAINES ,"Real-Time Rendering" ,Ed Sales and Customer Service Office, 1999.

[8] FAUERBY, K. Improved Collision detection and Response. 2003. http://www.peroxide.dk