

LOW FLYING FOR TERRAIN VISUALIZATION

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

Virtual Reality Department
European Virtual Engineering (EUVE)
Vitoria – Spain

Abstract

This paper presents the design and implementation of low flights over virtual terrains using a specific graphic engine that manages large size terrain data. This engine has been used for example in a Basque Country (North of Spain) terrain demo with a database up to 2.5Gb and texture resolution of 1024x1024 (3mts/pixel). Every year, geographical visualization applications use textures of higher resolution, so, it seems convenient to allow near-terrain navigations. As a consequence, collision calculi are more complex, and collision modules must be redefined in order to maintain the performance of the navigation engine. This work presents a first approach to this problem with the following topics: preprocessed collision data structure definition, collision detection algorithms and collision response algorithm.

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

1. Introduction

This paper presents the new techniques employed for performing flights at low height in a specific graphic engine for terrains visualization.

Before, this engine had the flying height limited. The user could descend just to a fixed height, because the textures had not enough resolution. Now, the textures data are increasing, it is interesting allow the user nearly full mobility.

To perform low flights, some changes in the graphic engine are needed. As there is no height limitation, the user can now collide with the terrain. We need to introduce a new module charged of managing collisions.

The main object of this project is 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 is 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.

This paper is a first approach to the problem solution and is divided into the following topic:

- Specific engine overview[1], where it is explained the main features of the engine employed for terrains visualization.
- Proposed solution for the new collision module introduced.
- Results after introducing the new module into the engine.

2. Application Description

The motivation for developing this graphic engine started two years ago. At that time, we were involved in the development of a real time simulation of a civil engineering work over large terrain areas. We employed platforms which had no easy mobility so we realized there was a necessity for a PC platform engine for our purposes.

Not to interfere with our customers work's privacy, some requirements were being taking into account such as the following ones:

- use of standard Digital Elevation Models (DEM) and orthophotography available from the Internet or from public institutions
- fast enough on a standard PC so that most users can run the software at home

- optimised for vast terrain areas with high visual fidelity
- easy to fly and visually attractive

We developed a 3D graphic engine for high quality and visual fidelity terrain visualization. Here are some of the engine features:

- The terrain area is not limited by the graphic capabilities of a PC, but for the amount of secondary storage space.
- It can also show georeferenced information about singular points, such as cities, villages and mountains.
- The user can move, or better fly, over the terrain database using different input devices such as the mouse or the joystick.

The terrains database are obtained from DEM, and lately processed with specific algorithms in order to obtain an adaptive triangle mesh. This reduces the number of elements without introducing too much error from the original DEM.

Each tile has an associated texture created with orthoimagery from the same source as the DEMs.

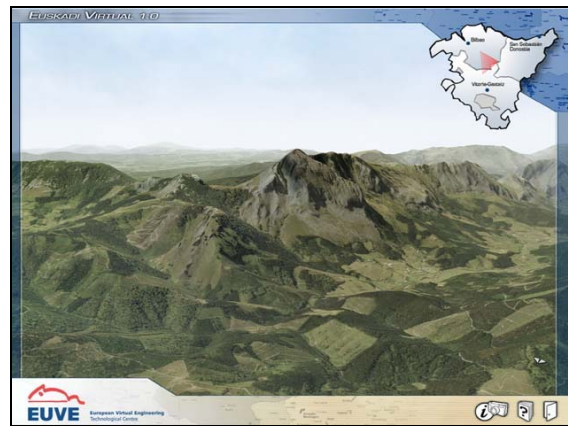
When the observer moves, tiles and textures outside that grid are unloaded from memory and replaced with new ones, so that there is always geometry in the camera's view frustum.

This engine uses multithreading, so that certain critical parts of code are not blocked by I/O operations. A multiscreen version has also been developed, such version can run distributed over several computers synchronised over TCP/IP links and can be used for larger displays or passive stereoscopic visualisation.

Other features are:

- OpenGL coding
- Culling with Bounding Boxes and Bounding Spheres
- Anisotropic texture filtering
- Circular fog
- Textures in 16 bit format to reduce memory utilization
- Vertex Buffer Object extension for OpenGL for a faster vertex transfer to the graphics card memory.
- Information in different selectable languages such as Spanish, English or Basque.

In figure 1 appears some shots.



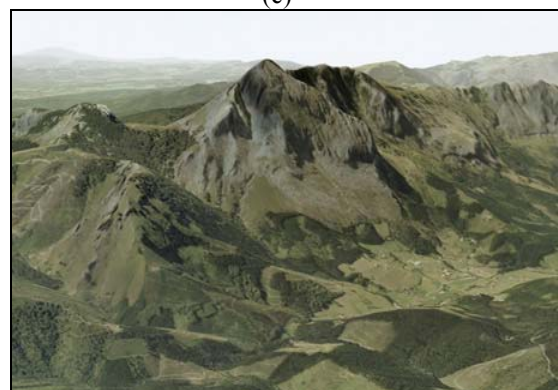
(a)



(b)



(c)



(d)

Fig1 (a)(b) Shots of the visualization engine, Basque Country Demo. (c)real photo (d) program shot of the same place as (c).

3. Proposed Solution

The swept-sphere collision detection and collision response applied here are based in Fauerby work [2].

The developed collision module is explained 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 figure 2.
 - **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*.

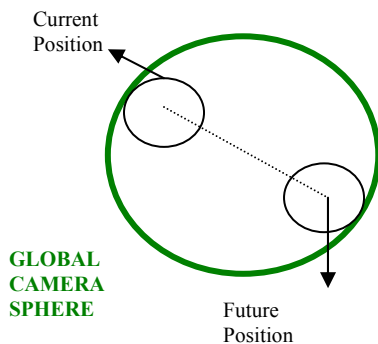


Fig.2 Sphere camera composition for *global motion test*

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 just has 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 base 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.

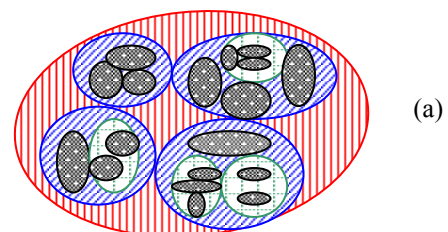
- Sphere center
- Sphere radius

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 3).



(a)

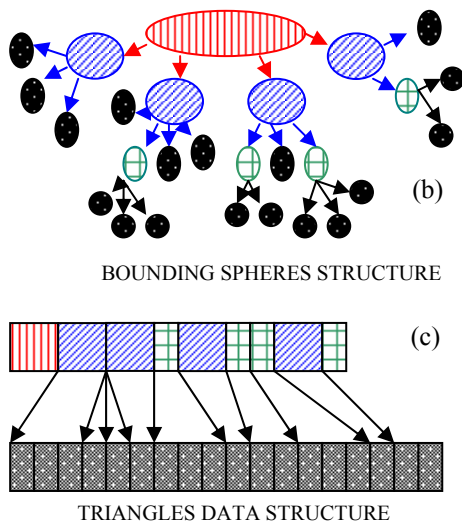


Fig.3 (a) Spheres contained scheme; (b) Hierarchical tree; (c) Memory structure.

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:
 - o The coordinates of the three vertices of the triangle.
 - o 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 correspondent algorithm.

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

✓ Global motion test for the **main level** of each tree.

- Failed → Discard that tree and pass to the following terrain piece until the last one. If no one is success there is no collision against the terrain.
- Success → A collision might happen. The lower levels of the tree should be tested.

✓ Collision detection test for **lower levels of each tree.**

- **Execute collision test**
 - o If terminal level → Access triangle structure and **execute Swept test.**
 - o Else → access Sphere node and execute **Global motion test**
- **Swept test collision?**
 - Success →
 - Update camera new position.
 - Go to more nodes = level.
 - Fail →
 - Go to more nodes = level.
- **Motion test collision?**
 - Success →
 - Down level.
 - Back to execute test.
 - Fail →
 - Go to more nodes = level.
- **More nodes = level?**
 - Yes →
 - Take next node.
 - Back to execute test
 - No → Up level.
 - o *First level?*
 - Yes → Finish. Go to next tree.
 - No →
 - Up level
 - Back to more nodes of the = level?

Once the collision is found, the algorithm carries on testing nodes until 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.

3.3. Global Motion Test

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 2.

3.4. Swept Test

This test is used with the terminal levels of the tree. If the data access algorithm arrives at this point, that means that 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 figure 2, here another approach is used. Camera current position and camera next position are modelled as two spheres of given radius

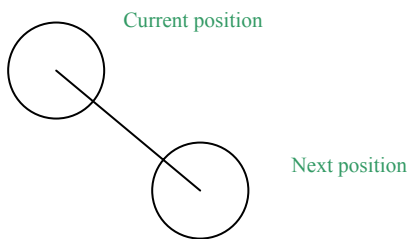


Fig.4 Camera model for swept test

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.

3.5. 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 for terrains the most suitable one 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.

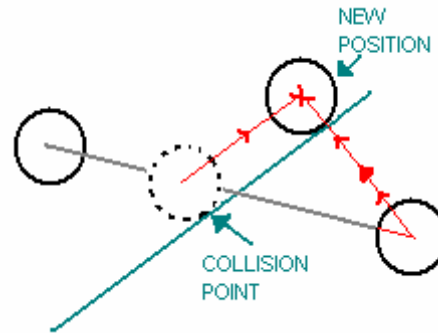


Fig.5 Collision reaction scheme

4. 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.



(a)



(b)

Fig. 6 Shots from Basque Country demo program.
Without collisions (a), and with collisions and not limited
altitude (b)

The main objective 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.

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

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.

5. Bibliography

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

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