SPATIAL ANALYSIS BASED ON 3D RENDERING TECHNIQUES

Rolando Quintero, Giovanni Guzmán, Miguel Torres, Marco Moreno
Centre for Computing Research (CIC), National Polytechnic Institute (IPN). Juan de Dios Bátiz s/n UP “Adolfo López Mateos”, 07738, México City, MEXICO

1. INTRODUCTION

To improve the visualization mechanisms and the decisions making, we developed a new algorithm for rendering spatial raster data in real-time. The application has been designed to build scenes from spatial raster data originated by Digital Elevation Model (DEM). This application is focused on final users, because nowadays the Geographical Information Systems (GIS) with 3D spatial data represent a great challenge in the new trends of Geocomputation area [1].

The main objective of using 3D spatial data is to emphasize the development of solutions for different fields, such as Telecommunications, Urban Planning, Tourism, Architecture, Simulation of Natural Phenomena, Natural Disaster Preventions, etc. These areas present a lot of problems related to mechanisms of 3D visualization. With our application, one can solve the problem of 3D data representation and offer virtual scenes, which are ready to navigate, using either simulations or defined trajectories [2].

The proposed tool contains a module for converting the DEM format to specific raster format, to reduce the complexity and improve the visualization mechanism in other perspective, according to the elevation attribute. In the algorithm, we incorporated virtual programming flights and spatial overlays.

The present work is focused on spatial analysis, using overlays with vector data, Euclidean distance measurements and slope evaluation. This environment allows obtaining a general perspective of many study area. More details can be discovered with this algorithm, because the visualization parameters can be modified by the users.

2. SPATIAL DATA RETRIEVAL

Initially, it is necessary to acquire a satellite image of the scene, which should be used to obtain the texture of the data (RGB values). After this, the DEM type is used to obtain the elevation to represent the 3D environment.

Nowadays, there are some types of DEM such as: 7.5-minute, 15-minute, 2-arcsecond, and 1-degree units [3]. For implementation purposes, we proposed to choose the 1-degree DEM variant. A DEM consists of a sampled array of elevations for ground positions that are normally at regularly spaced intervals. The basic elevation model is produced by the Defense Mapping Agency (DMA), and distributed by the USGS, EROS Data Center in the DEM data record format. In reformatting product, USGS does not change the basic elevation information [4].

Basically, a DEM file is integrated by three types of records, usually called A, B and C. The structures of these records are the following [5]:

- record A contains information defining the general characteristics of DEM, including descriptive header information relating to the DEM’s name, boundaries, units of measurement, minimum and maximum data values, number of type B records and projection parameters. There is only one type A record for each DEM file, and it appears as the first record in the data file,
• record B contains elevation data and associated header information. All type B records of the DEM files are made up of data from one-dimensional bands, called points. Therefore, the number of complete points covering the DEM area is the same as the number of type B records in the DEM,
• record C contains statistics on the accuracy of the data in the file.

As first step, it is indispensable to read the minimum and maximum elevation values, which can be found in inches, or meters units. These values will be used to make the conversion process at gray levels.

The following step consists of reading and converting each elevation samples, which integrate the DEM scene. The results are stored into a new digital image that is represented by floating point numbers.

The elevation values are represented by meters or inches: it is necessary to make a conversion process at gray scale, because the scene generator requires the intensity between 0 and 255 ranges to represent the elevation of a sample.

To achieve this process, it is important to use the minimum and maximum altitude values, which previously are read. The algorithm must assign a 0 gray level value to the minimum altitude and 255 to the maximum altitude. With this assignation, the gray levels of the scene for any elevation are computed using the equation (1).

\[
\text{gray\_level} = \frac{\text{elevation} - \text{elevation}_{\text{min}}}{\text{elevation}_{\text{max}} - \text{elevation}_{\text{min}}} \times 255
\]  

The last step involves building two digital images to store all information about the texture and elevation data. These images are represented by means of 3-bytes into a bi-dimensional matrix.

(Fig. 1) shows the new images that are obtained. The (a) image represents gray levels according to the elevation. The (b) image represents the same scene according to the RGB texture values.

![Fig. 1 Obtained images.](image)

3. Visualization Stage

The visualization stage has two main tasks. First is the setup of loading and visualization data parameters to load the raster data and process them. While this process is completed, the information can be used by the render engine. The second task is related to the render engine. This render computes the data in a graphic way.
3.1 Data Loading and Visualization Parameters Setup

In this process, we take the data in the mentioned bitmaps and obtain a set of vertices, which will be used by the render engine. The process involves three tasks for each vertex:

Spatial coordinates computation

To obtain the spatial coordinates, we must use the bitmap containing the elevation data to define a vertex grid with the elevation data. This is called Elevation Bitmap (EBM). In (fig. 2) we illustrate this process.

![Image](Image)

**Fig. 2** From the EBM to the spatial coordinates.

The conversion from the EBM to the vertex grid is made in the following way: Let us define the image \( l(i,j) \) of \( M \times N \) pixels and \( \beta \) bits per pixel, with \( 1 \leq i \leq M \) and \( 1 \leq j \leq N \). Also, we have \( F(x,y) \rightarrow z \) as the elevation function. \( V_x \) and \( V_y \) are the vectors defining the spatial section for the data. Now, we can get the mapping factors from the image to the elevation function. Equation (2) describes this method.

\[
\Delta_x = \frac{V_x - V_{x_{\min}}}{M}, \Delta_y = \frac{V_y - V_{y_{\min}}}{N}, \Delta_z = \frac{V_z - V_{z_{\min}}}{2^\beta}
\]  

(2)

By using these factors, it is possible to obtain the mapping equation (3). \( l(i,j) \) to \( F(x,y) \):

\[
F(V_x + \Delta_x, i, V_y + \Delta_y, j) = V_z + \Delta_z l(i, j)
\]

(3)

1 \leq i \leq M, 1 \leq j \leq N

Then, we can obtain the vertex grid as the function \( G(i,j) \rightarrow (x,y,z) \), which is defined in equation (4):

\[
G(i, j) = (V_x + \Delta_x, i, V_y + \Delta_y, j, V_z + \Delta_z l(i, j))
\]

(4)

---

1 The vertices contain not only quantitative information, but also qualitative information for a spatial point.
Texture coordinates computation
The process described in the previous section gives us a set of vertices that can be processed to obtain a 3D image. Nevertheless, it is indispensable to know not only these data, but also the texture information of the landscape.

To make this, we need to generate a set of texture coordinates for each vertex. The process is the following.

Let \( T(i,j) \rightarrow (u,v) \) be the function that represents the texture coordinates. This function is defined in equation (5):

\[
T(i,j) = \left( \frac{i}{M}, \frac{j}{N} \right) \quad \text{for} \quad 1 \leq i \leq M, 1 \leq j \leq N
\]  

(5)

Normal vectors computation
We must now compute the normal vector to each vertex (in the case that uses illumination schemas). These vectors are denoted as \( \mathbf{N}(i,j) \rightarrow (x', y', z') \), and are defined in equation (6):

\[
\mathbf{N}(i,j) = \sum_{k=1}^{K} \mathbf{N}_k(i,j)
\]

\[k = 1, K, 1 \leq i \leq M, 1 \leq j \leq N\]

\[
\mathbf{N}(i,j) = \frac{\mathbf{N}(i,j)}{\| \mathbf{N}(i,j) \|}
\]

Where \( K \) is the number of polygons met the vertex \( G(i,j) \). In (fig. 3) we illustrate these polygons.

![Diagram](image)

**Fig. 3** The normal vertex \( \mathbf{N}_v \) is the average of the normals \( \mathbf{N}_1, \mathbf{N}_2, \mathbf{N}_3 \) and \( \mathbf{N}_4 \). The normals of the polygons that meet the vertex.

3.2 Render Engine
With the parameters computed, we can render a landscape of the spatial data. A trivial algorithm to make this is the following:

```
RENDER(o)
  1 | for i = 1 to M-1
  2 |   for j = 1 to N-1
  3 |     RENDER-VERTEX(G[i,j])
  4 |     RENDER-VERTEX(G[i+1,j])
  5 |     RENDER-VERTEX(G[i+1,j+1])
  6 |     RENDER-VERTEX(G[i,j+1])
```

Notice that this process could produce a huge quantity of data to process. In the tests that we made, we have used a DEM that produces a vertex grid of 2048x2048 elements. It is more than 4 million of polygons. Appyling space partitioning algorithms [6] and hide surface
removal techniques [7], we must process a set of 500 thousands of polygons\(^2\) approximately. Then, processing such huge volume of data, it is necessary to decrease much more the number of polygons to process. This can be done by means of Level of Detail (LOD) algorithms.

**Some LOD algorithms**

[8], [9] present some algorithms to decrease the LOD in complex scenes. These algorithms present three main drawbacks:

- they are complex and increase the workload of the processor. They make changes to the terrain data (spatial data), because they are focus on the final visual appearance of the scene,
- they modify the terrain data depending on the observer’s viewpoint. Due to this, the spatial data analysis is not possible,
- the number of polygons rendered is variable, and then the frame per second (fps) rate is not constant during the simulation.

In this paper, we develop an algorithm to reduce the terrain LOD to speed up the data visualization. The algorithm faces the problems mentioned above.

**Real-time rendering**

The goals of the algorithm are: 1) be simple, 2) not to affect the terrain data and 3) run in real-time. The algorithm must warrant a maximum number of polygons to render (a constant fps rate).

To describe the algorithm, we must define some parameters first,

- a matrix \( L \) of \( H \times H \) defines the discrete LOD’s to use. \( H \) is an odd number greater than 1, and \( L[i,j] \neq 0 \) for \( 1 \leq i,j \leq H \),
- a number \( S \) defines the optimization unit size; it means that it is necessary to optimize regions of \( S \times S \) polygons,
- a vector \( o \) represents the observer position.

Using the defined parameters, it is possible to outline the algorithm. The proposed algorithm is the following:

```plaintext
RENDER (o)
1 \( (o_x, o_y) \leftarrow \text{RELATIVE-POSITION}(0, G) \)
2 \( \text{for } i = -\frac{1}{2}H \text{ to } \frac{1}{2}H \)
3 \( x \leftarrow o_x + S(i - \frac{1}{2}) \)
4 \( \text{for } j = -\frac{1}{2}H \text{ to } \frac{1}{2}H \)
5 \( y \leftarrow o_y + S(j - \frac{1}{2}) \)
6 \( \text{RENDER-BLOCK}(x, y, L[i+\frac{1}{2}H, j+\frac{1}{2}H]) \)

RENDER-BLOCK(x, y, lod)
1 \( \text{if } lod > 0 \)
2 \( \text{for } i = x \text{ to } x + S \text{ step } lod \)
3 \( \text{for } j = y \text{ to } y + S \text{ step } lod \)
4 \( \text{RENDER-QUAD}(i, j, lod) \)

RENDER-QUAD(i, j, lod)
1 \( \text{RENDER-VERTEX}(G[i,j]) \)
2 \( \text{RENDER-VERTEX}(G[i+\text{lod},j]) \)
3 \( \text{RENDER-VERTEX}(G[i,j+\text{lod}]) \)
4 \( \text{RENDER-VERTEX}(G[i,j+\text{lod}]) \)
```

\(^2\) Quads, therefore a million of triangles.
By the use of the algorithm, we can easily compute the number of polygons to be processed \((N_P)\):

\[
N_P = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{S_{i,j}}{L_{i,j}}
\]

(7)

4. MAKING 3D SPATIAL ANALYSIS

In this paper, we developed a rendering technique to improve the algorithm performance and visualization. This methodology proposes a set of steps to make 3D spatial analysis. These steps are the following:

- define the goals of the analysis,
- establish the scale of altitude coordinates,
- define the characteristics of spatial vector layers,
- overlay the vector layers,
- make visual inspection,
- identify Euclidean distance and slope measurements,
- define automatic trajectories.

The steps 2, 6 and 7th are alternatives, because they depend on the particular analysis, according to the case of study. (Fig. 4) shows the analysis process for 3D spatial data.

5. TESTS AND RESULTS

There have been made some performance tests with different data sets. As we have mentioned, the performance of the algorithm is constant, no matter the volume of elevation data involved.

The tests made on a PC Pentium III @ 800MHz, 256Mb of RAM and a 3D card NVIDIA G-force 2MX. In each test case, we have obtained rates near 10fps, meanwhile with the trivial algorithm; it have been obtained rates of 0.1fps. The test cases are presented in (table 1)
Table 1 The test cases and its results.

The results of the application are shown in (fig. 5), using the proposed algorithm. The
(fig. 5a) shows the result with the trivial algorithm. (fig. 5b) presents the result using our
algorithm with the parameters in equation (8).

\[
L = \begin{bmatrix}
64 & 32 & 16 & 8 & 16 & 32 & 64 \\
32 & 16 & 8 & 4 & 8 & 16 & 32 \\
16 & 8 & 4 & 2 & 4 & 8 & 16 \\
8 & 4 & 2 & 1 & 2 & 4 & 8 \\
16 & 8 & 4 & 2 & 4 & 8 & 16 \\
32 & 16 & 8 & 4 & 8 & 16 & 32 \\
64 & 32 & 16 & 8 & 16 & 32 & 64
\end{bmatrix}
\]

\[S = 128\]

\[\Rightarrow N_p = 968\]

(a) 

(b) 

Fig. 5 . a) Result with the trivial algorithm. b) Result with the proposed algorithm.

6. CONCLUSIONS

As described in this work, the application has been designed and developed to work
with spatial data (raster and vector format). The implemented algorithm requires a low time
to process all data, which are stored both in the DEM and Landsat images. The only
restriction is that the images must correspond to the same scene. Two new digital images
that are generated can be easily accessed in a faster way by the proposed rendering method.

The developed algorithm fully fit the stated goals. It does not overload the processor, because it is very simple. Also, the rendering algorithm warrants a maximum number of elements to be rendered. In this way, we can manipulate this number \(N_0\) by varying the parameters of the matrix \(L\). Additionally the algorithm does not modify the spatial data. This characteristic is necessary to make the spatial analysis.

The application integrates a module to make automatic virtual flights through VFA. VFA consists of programming of the C++ node structures with the generated virtual scenes, providing the trajectories, calculated as Bezier curves, to the virtual scenes. All algorithms of the application and GUI have been programmed in C++.

The research areas – 3D-GIS and Spatial Analysis – take benefits from each other. 3D-GIS technology plays an important role in the analysis of the problems on visual mechanism and decision making, because it offers a unique capacities for automatic managing and analysis of spatial data. On the other hand, 3D-GIS achieve an improved decision model, which can be incorporated into GIS, because GIS + Database + Flight Path are equal to Interactive Virtual Environments.

7. ACKNOWLEDGMENTS

The authors of this paper wish to thank the Centre for Computing Research (CIC), General Coordination of Postgraduate Study and Research (CGEPi), National Polytechnic Institute (IPN) for their support. Additionally the authors wish to thank the reviewers for their pertinent comments.

8. REFERENCES


