A LARGE-AREA TERRAIN SIMULATION ON PC

Chaoqun Wang [a], Lingkui Meng [a], Zhiyong Lin [a]

[a] School of Remote Sensing Information Engineering Wuhan University, 129 Luoyu Road, Wuhan, 430079, China
wcq671@163.com, lkmeng@public.wh.hb.cn, zhy_lin@263.net

KEY WORDS: Large-area terrain, Simulation, Vega Prime, OpenGL, Grid stacks, Kalman filter

ABSTRACT:

This paper presents a general architecture of the large-area terrain simulation system based on a single processor PC. It provides methods of modeling, management and dynamic scheduling for large-area terrain, achieving a balance between image quality and real-time rendering. The Grid stacks methods are developed to manage the database. The AOI approach has been developed to dispatch the database dynamically in the browse. The kalman filter approach has been developed to forecast the observer's position to improve the fluency of the browse. The implementation is based on OpenGL, and the technology is based on MultiGen-Paradigm Vega Prime platform, which provide a developer interface for a special system.

1. FOREWORD

In the recent years, with the computer science's development, especially the PC, the application of the visual simulation is becoming more and more popular. As an important part of visual simulation, the large-area realtime 3D terrain simulation is the base of the computer simulation, such as city design and planning, GIS (geographic information system) and battlefield training. The paper introduces an approach of large-area terrain visual simulation based on PC.

2. OVERALL STRUCTURAL DESIGN

2.1 System Function

1. Complex large-area realtime 3D terrain databases.
2. A number of simulation entitles are included.
3. User can change the pan formats and the viewpoint freely.
4. A good user's interface.

2.2 System Hardware and Software

Computer configuration: Inter Pentium 4, 1.8GHz, 512 RAM, 60GB HD, GeForce 4 MX 440 Graphics Card.
Operating system: Windows 2000 Professional or Windows XP.
Software developing platform: Microsoft VC++ 6.0, Vega Prime and OpenGL
Modeling tool: CTS (Creator Terrain Studio).

2.3 System Structural

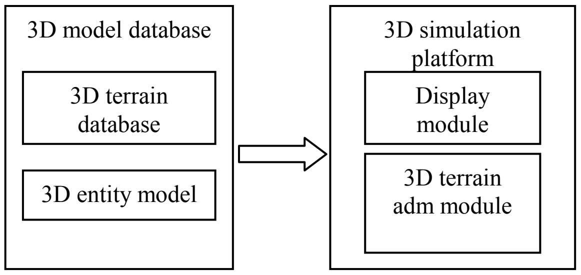


Figure 1. System structural

Figure 1 depicts the overall system structure. A description of the functionalities of its components follows:

(1) 3D terrain database: this is the most important data in the system. It is the base of the other function and application. It includes terrain data and scenery data. The terrain data come from the DEM and the scenery data come mainly from the culture feature data and attribute data.

(2) 3D entity model: simulation entities and geographical data. Those entities can be having various computational models conducting different types of activities, such as tanks firing, aircraft bombing craters.

(3) Display module: usually, displaying per-frame include three parts: Application, Cull and Draw. In the Application, after loading the data, the module accepts the all kind of parameters, such as view point, angle of view. In the Cull, the module traverses the data, and finds the visibility data. In the Draw, the module draws the visibility data, making it into a frame, and pulls the frame into the frame buffer.

(4) 3D terrain adm module: in order to improve the quality, efficiency and lower cost, the 3D terrain administering module is developed to administer and dispatch terrain patches. The main idea is to forecast the visibility terrain patches and only to load the patches.

3. MODELING, ADMINISTERING AND DISPATCHING THE 3D DATABASE

3.1 Modeling the 3D Database

We mainly use CTS to build the 3D database. They are maked up of three parts (Figure 2).
Figure 2 depicts the overall process of modeling the 3D database. A description of the functionalities of its components follows:
(1) CTS reads gridded elevation data that are in the following formats: DEM, DED. CTS tessellates the elevation data using either the Polymesh or Irregular Mesh algorithm that you specify. You can design the tiles and triangle spacing for your

needs. Since CTS is output oriented, you define the desired tiling and triangle sampling of the output that the CTS tessellators build. CTS creates an OpenFlight file for each terrain tile.

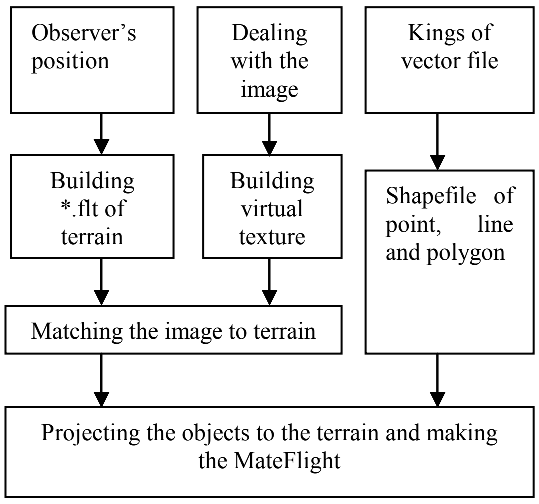

Observer's position
Dealing with the image
Kings of vector file
Building *.flt of terrain
Building virtual texture
Shapefile of point, line and polygon
Matching the image to terrain
Projecting the objects to the terrain and making the MateFlight

Figure 2. Flow chart of building the database

(2) A virtual texture is a tiled multi-layered (MIP-mapped) image, which can be geo-referenced. The highest resolution tiles are displayed when the observer is close to the terrain on which the virtual texture is mapped; lower-resolution tiles are displayed when the observer is farther away. The goal in generating a virtual texture is to make the transitions between the high resolution imagery and lower resolution imagery as visually seamless as possible. To support this goal, the Image Tools allow you to Blend and Feather the image.

(3) In CTS, Project Culture converts and tessellates point, linear, and areal features from source files into 3D models. The feature data can be either in MultiGen DFD or ESRI shapefile format and be included in a vector grid dataset. (To convert existing DFD data to shapefile format, use the Convert Vector tool.) The 3D models are then projected onto the terrain.

3.2 Data Administering

If you've used Levels of Detail (LOD) for models, you know the advantages of displaying the right amount of visual information to the observer based on the observer's distance from a model. We've used Grid stacks for the models. Grid Stacks apply the LOD concept to textures, elevation data, and vector feature data. Grid Stacks are created for a specific area block in a terrain. As the observer approaches the area block, higher LODs replace lower LODs until the highest level of detail is displayed.

With Grid Stacks, the terrain can consist of many layers, each with its own data. Each layer is divided into cells along lines of latitude and longitude. As the observer moves across the terrain during runtime, the cells are "paged" into memory as needed, which ensures efficient rendering times. This concept of Virtual Textures (which are called clip maps on some high-end SGI workstations) maps imagery data, such as a satellite image, across the entire terrain skin as a single "virtual" texture. The terrain is then divided into cells small enough to page into memory as needed during runtime.

Grid Stacks extend this idea of Virtual Textures to include not only Imagery data but all the data in a simulation. Each level contains all the vector features, imagery, and terrain data for a given amount of detail. There is usually more than one level for each cell, so that more detail is visible as you approach an Area of Interest. A landing strip, for example, would display its highest level of detail closest to the landing strip, where detail is visible to observers when the aircraft is taking off or landing. With a Grid Stack, it is easy to coordinate the textures, elevation data, and vector features for each level of the landing strip.

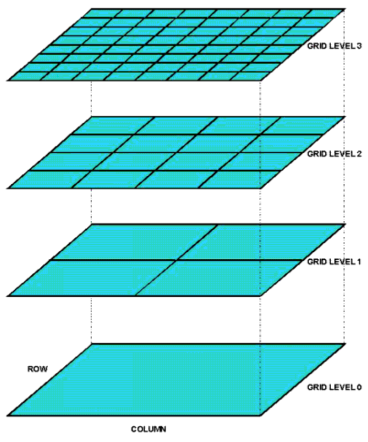


Figure 3. Grid stacks

Figure 3 depicts Grid stacks. The Grid stack is divided into levels. Each level has cells that have a 1 to 4 relationship with adjacent levels. For example, here Level 0 has one cell, Level 1 has four cells, Level 2 has 16 cells, and Level 3 has 64 cells. Each level (except Level 0) has "parent cells" and may have "child cells". Collectively, these levels and their relationship are referred to as a "Grid Structure".

3.3 Dispatching the 3D Database

3.3.1 AOI: Generally, a terrain database has a lot of data, and so the computer can't load all the data to the memory. 3D terrain adm module must dynamically dispatch the patches of the database and only keep the AOI (Area of Interesting) in the memory. With the area of interest analysis, different patches of a terrain can be compared. The patches including the areas of interesting will load into the memory, and when the patches in memory not including areas of interesting, they will be thrown from memory.

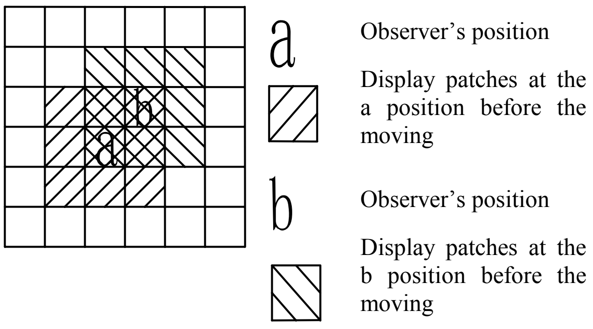

a
b
a
Observer's position
Display patches at the a position before the moving
b
Observer's position
Display patches at the b position before the moving

Figure 4. Display the patches

As can be seen from the Figure.4, when an observer is moving out of a patch it is currently on, some patches are swapped from the corresponding display database and then the new display patches will be loaded into the memory. Thus, the number of patches in the display database remains the same and the observer will always be on its center patch.

3.3.2 Forecast the Observer's Position: If knowing the observer's moving way or the other useful information, we can forecast the next position of observer. And so we find an approach to improve the implementation of dynamically dispatching. We can load the patches of AOI before the observer's moving into some patches, which can make the simulation system run more fluently and higher efficiently. In the system, we use the kalman filter to forecast the observer's position.

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback—i.e. for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate. The time update equations can also be thought of as predictor equations, while the measurement update equations can be thought of as corrector equations. Indeed the final estimation algorithm resembles that of a predictor-corrector algorithm for solving numerical problems as shown below in Figure 5.

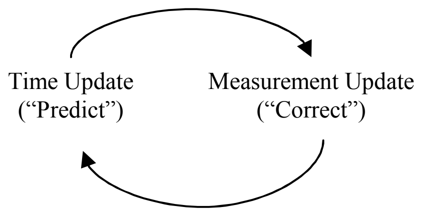


Figure 5. The ongoing discrete Kalman filter

Discrete Kalman filter time update equations.

$$\hat{x}_k^- = A\hat{x}_{k-1} + u_{k-1} \tag{1}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2}$$

where $\hat{x}_k$ = the state at the step k

u = the process noise

P_k^- = the a priori estimate error covariance

P_k = the a posteriori estimate error covariance

Q = the process noise covariance

The $n \times n$ matrix A in the difference equation (1) relates the state at the previous time step k-1 to the state at the current step k, in the absence of either a driving function or process noise. Please notice how the time update equations 1, 2 project the state and covariance estimates forward from time step k-1 to step k.

Discrete Kalman filter measurement update equations.

$$z_k = Hx_k + v \tag{3}$$

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{4}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{5}$$

$$P_k = (I - K_k H)P_k^- \tag{6}$$

where z_k = the measurement

v = the measurement noise

R = the measurement noise covariance

The $m \times n$ matrix H in the equation (4) relates the state to the measurement z_k. In practice H might change with each time step or measurement, but here we assume it is constant. The $m \times n$ matrix K in (4) is chosen to be the gain or blending factor that minimizes the a posteriori error covariance P_k. The process noise and measurement noise are assumed to be independent (of each other), white, and with normal probability distributions.

Above the five equation, x_k, z_k, A, H denote as fallows:

$$\hat{x}_k = \begin{bmatrix} x_{c(k)} & \dot{x}_{c(k)} & y_{c(k)} & \dot{y}_{c(k)} \end{bmatrix}^T$$

$$z_k = \begin{bmatrix} x_{c(k)} & y_{c(k)} \end{bmatrix}^T$$

$$A = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

where Δt = the interval between the step k-1 with the step k.

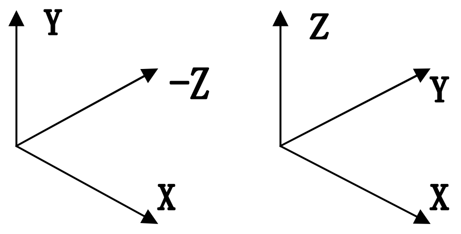
4. COMBINING VEGA PRIME AND OPENGL

As a current simulation platform, Vega Prime doesn't support many special functions, such as drawing pipe-thread. In order to achieve those kinds of functions, we use OpenGL in Vega Prime. Four key points in it follows:

(1) How to use the OpenGL in Vega Prime? In Vega Prime, vsChannel defines the draw function as callback. Default function is provided which perform basic functionality, however, the user is free to override the function as desired. And so we can draw the special scene by OpenGL in the function. The signature for the function is defined below: **DrawFunc —uint (*)(const vsChannel *, vrDrawContext *)**.

(2) How to adjust the coordinate of OpenGL with the coordinate of Vega Prime? We can see the coordinates of OpenGL and Vega Prime in the Figure.6.

Notification for the projective transformation, we must make the viewing volume of OpenGL equal with that of Vega Prime. For instance, in Vega Prime, we use the function of makeSymmetricProjection to set the projection matrix to a matrix that describes the given symmetric viewing frustum. So we can get the corresponding parameters (hfov, vfov, near, far) by the two functions of getFOVSymmetric and getNearFar and then apply the parameters to OpenGL.



Coordinate of OpenGL Coordinate of Vega Prime

Figure 6. Coordinates of OpenGL and Vega Prime

(3) In whichever OpenGL or Vega Prime, the observers must have the same positions. That is, the two viewpoints overlap. How to convert the viewpoint of OpenGL to that of Vega Prime? In Vega Prime, we can use the function of getPosition to get the observer's position. And in OpenGL, we can place the observer to the same position by the function of gluLookAt, which takes an eye position, a position to look at, and an up vector.

(4) In order to use OpenGL in Vega Prime, the observers not only have the same viewpoints but also have the same viewports. How to convert the viewport of OpenGL with that of Vega Prime? Resembling viewpoint, in Vega Prime, we can use the function of getViewport to get the rectangular region of the window to which to the image will be mapped. And in OpenGL, we can transform from OpenGL coordinate to Vega Prime coordinate system by calling the gluViewport function.

5. CONCLUSION AND FUTURE WORK

Based on the current software platform and tools, we have built a large-area terrain visualization system on PC. Grid stacks model has been developed to use the computer resource at the most important fields in order that the simulation system has better quality, higher efficiency and lower cost. A dynamic dispatch model has been developed to administrate the database, by which the performance of simulation system has no relation with the area of terrain. A dynamic dispatch forecast approach has been dreamed up to forecast the observer's position, which can improve the dispatch speed and make the browse more fluently. Additionally, Combining Vega Prime and OpenGL is the entrance to building a complex simulation system.

REFERENCES

Greg Welch. and Gary Bishop (1995)., *An Introduction to the Kalman Filter,* University of North Carolina at Chapel Hill, Department of Computer Science, Chapel Hill, NC, USA. TR95-041.

Kalman R. E., 1960. *A New Approach to Linear Filtering and Prediction Problems*, Transaction of the ASME—Journal of Basic Engineering, pp. 35-45 (March 1960).

Larry S. Bonura., 2001. *Vega Prime Programmer's Guide* [M]MultiGen-Paradigm Inc

Larry S. Bonura., 2001. *Vega Prime Desktop Tutor for Windows 2000, Windows XP Professional Edition, Solaris 8.0, and Red Hat 8.0 Linux* [M]MultiGen-Paradigm Inc

Larry S. Bonura., 2003. *Creator Terrain Studio Help* [M]MultiGen-Paradigm Inc

Larry S. Bonura., 2003. *Creator Terrain Studio User's Guide* [M]MultiGen-Paradigm Inc

Richard S. Wright. and Jr Michael Sweet., 2001. pp. 13-19, 52-63.