Interactive Realtime Terrain Visualization for Virtual Set Applications

Arnfried Griesert
Competence Center ITV
Fraunhofer Institut Medienkommunikation
Schloss Birlinghoven
D-53754 Sankt Augustin

Oliver Walczak, Jens Herder
Virtual Sets and Virtual Environments Laboratory
Department of Media
FH Duesseldorf, University of Applied Sciences
Josef-Gockeln-Str. 9
D-40474 Duesseldorf

Germany

voice/fax: [+49](2241) 14-2149/2449
e·mail: griesert@imk.fraunhofer.de
www: www.imk.fraunhofer.de

[+49](211) 4351-800/803
olly.walczak@gmx.de,
herder@fh-duesseldorf.de
vsvr.medien.fh-duesseldorf.de

Abstract

Virtual set environments for broadcasting become more sophisticated as well as the visual quality improves. Realtime interaction and production-specific visualization implemented through plugin mechanism enhance the existing systems like the virtual studio software 3DK.

This work presents an algorithm which can dynamically manage textures of high resolution by prefetching them depending on their requirement in memory and map them on a procedural mesh in realtime. The main goal application of this work is the virtual representation of a flight over a landscape as part of weather reports in virtual studios and the interaction by the moderator.

Keywords: terrain engines, level of detail, virtual set environments, virtual studios

1 Introduction

Virtual set environments (virtual studios) are used in modern broadcasting facilities. Live and preproduced content are mixed and become an unique program. Live production has its advantages in covering actual situations (news) and in keeping the costs down by limiting the production time.

Weather reports have shown perspective animations of flights over cities showing weather conditions [ifa97]. Those animations were preproduced and required several hours for production. We report about a realtime system based on terrain engines, allowing to have arbitrary flights under live control for virtual studios. Enabling up-to-date weather reports with interactive control of the visualization.

The task of terrain engines is to draw virtual environments in three dimensions in their whole wideness and complexity. They have to visualize datasets of environmental information in realtime as polygon structures on the display. Whether the deal is to visualize geographic datasets, military simulations or simply to create entertainment software like computer games, anywhere terrainengines with special algorithms optimized for their special tasks are applied. And the rapid progress of graphics hardware anytime leads to new ways of software development.

The terrain engine is embeded in a realtime renderer with interaction control of a virtual studio which has multiple interfaces to devices and applications as shown in Figure 1 [Her01]. Special content like weather flights can be incorporated using a plugin mechanism.

The terrain engine for virtual set environments as well as the necessary algorithm is described in depth in [Wal03]. An overview about virtual set environments and an introduction to the virtual studio software 3DK [3dk02], which were used in this project can be found in [GAB+98]. The broadcasting requirements for virtual set environment as well as a brief review of similar projects are given in Section 2. How terrain visualization can be integrated into a weather broadcast program and what additional steps are necessary are

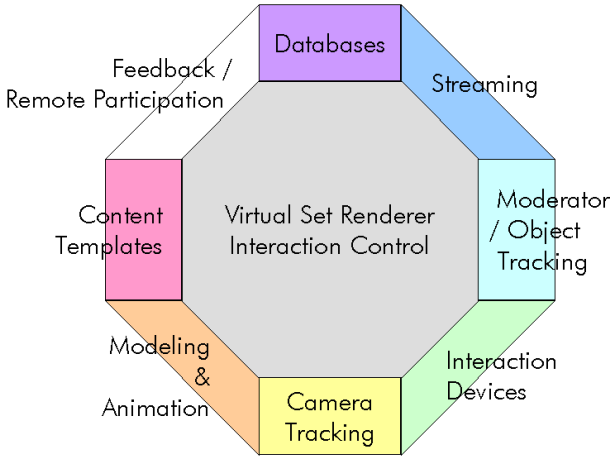

Figure 1: Multiple interfaces of a Virtual Set Renderer [Her01]

presented in Section 3. Achieving high visible quality requires specific focus on modeling and texturing which is given in Section 4. Interaction and working with the system is described in Section 5. The implementation itself and the system design is outlined in Section 6. Finally, we discuss our experiences with the system and give future directions in Section 7.

2 Broadcasting requirements

The broadcasting requirements for virtual studios differ from those for virtual environments. While virtual environments set their main focus onto the rendering of a very complex scene with many polygons, sophisticated animation scenarios and the ability of different kinds of user interaction, a constant frame rate and accurate image quality plays in most cases a minor role.

For virtual studio background visualization we must synchronize the rendering with an external sync source. This quarantees a constant frame rate of 50 half images for the PAL video format (60 for NTSC). To put it in other word, we only have 20 ms for rendering one image. So in most cases, the complexity of our virtual set must be reduced to fit this condition.

A stable high image quality has to be achieved at a constant framerate. Every rendering engine must runs synchronously to the external studio clock. Virtual environments as used in the current internet, for example, have the ”best effort” that results in a variable framerate because it depends on content and interaction.

The requirements for virtual studios can be described as:

- constant framerate
- high visual quality
- easy to use in a broadcasting production environment
- automatic or simple refresh of incoming data (e.g. satellite images)
- free selection of flight paths
- visualization of weather conditions

In 1997 we demonstrated in cooperation with a local TV station how a distributed weather forecast can be managed. The terrain weather flight simulation presented in this paper can be seen as an extension to this work, where now the main focus lies in the realtime rendering and the ability that the operator can adjust the graphic on the fly. Also the moderator should be able to interact with this kind of weather visualization.

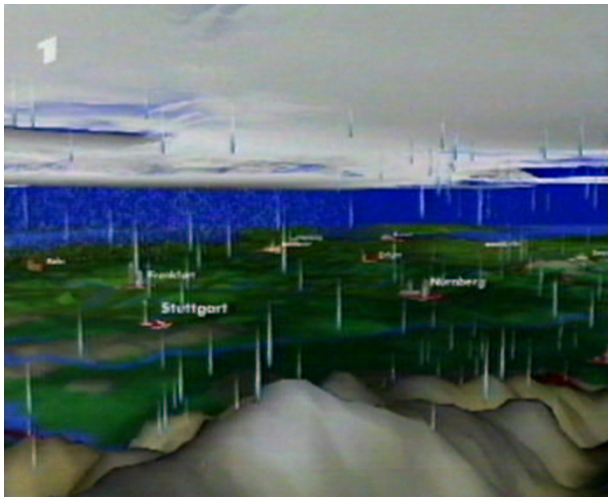

Figure 2: Screenshot from the preprocessed terrain flight for virtual studios shown 1997 in the first German television program ARD

3 Workflow

There were several approaches for weather terrain flights in the last years everyone could see on television. The main problem was the time necessary for rendering the flight over the terrain together with the actual weather condition.

The weather data comes normaly in form of simple image files with colored representation of clouds, rain,

snow, thunder strikes, temperature and so on. Each pixel shows the weather for the corresponding location. With this information the graphics system can calculate the animation for the flight movie. Often this work is done with common 3D animation tools with very good quality but although with poor rendering performance. In one case eight PCs were used up to three hours for rendering the animation for each daily weather forecast. The disadvantage is obvious. When you see the visualisation in form of a terrain flight, the data is more or less outdated.

The application described here renders a terrain flight in realtime by using weather data image files which will be loaded automaticly to the graphics system and by adding the current weather condition to our visualisation. Start and end point for the animation can be arbitrarily set by the operator.

It is also possible that the moderator can interact with the virtual terrain via external devices like a remote stylus. The moderator can show places of interest on demand to the TV viewers or he can switch to another scenario with additional weather information.

The integration of the terrain into the virtual set can be done in several ways. In 1997 it was placed on a virtual table in the middle of the set. By disconnecting the virtual camera from the real camera position, we get a smooth transition from the virtual set to the weather information given by our terrain engine. Because our new terrain engine is fully integrated in the virtual set application 3DK, we have further possibilities. Using an earth globe and flying directly through volume clouds is just one of them. This makes the daily weather forecast more interesting to the viewers and opens new graphical ways of transition between the virtual set and the the flight over the landscape.

4 Modeling and texturing

There are several approaches made for realtime terrain rendering which have been investigated in the past. As the most populars can be seen Geomipmapping [Geo00], Quadtrees and the ROAM [DWS+97] algorithm.

The algorithm described here is an extension to the quadtree data structure used for realtime visualization of complex terrain meshes. Even though the ROAM approach is much faster, quadtrees are well suited when using large textures which must be divided in lower tiles as we will see later in this section.

To speed up rendering time, we used frustum culling to determine what branches of the quadtree can be ignored when drawing the geometry for the terrain. Only the leaves with bounding boxes that are completely or partly inside the visible area of the viewing frustum will be handled, all others are ignored. Figure 3 demonstrates the bounding boxes of the nodes seen from the point of the camera position. White frames show nodes which are completly inside the frustum while blue colored boxes are partly outside. Branches which do not intersect with the viewing frustum are not further subdivision. The right picture shows the considered boxes seen from a different camera position other than the one used for culling.

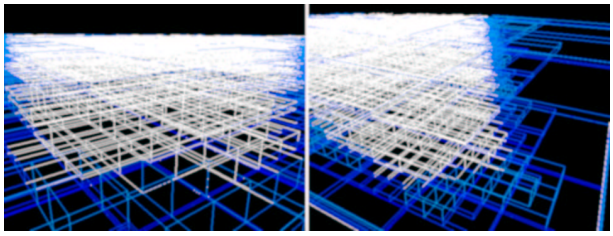

Figure 3: Screenshots demonstrating the quadtree culling mechanism

Given a grayscale heightmap image, we construct a regular mesh with vertices that are shifted in y-direction according to the pixel values. The mesh size does not need to have the same resolution as the heightmap when interpolating these height values. In our example we used a heightmap with 1833x1972 pixels where each pixel corresponds to a size of 75 meters in the real world. As mentioned before, we used a tree for storing the geometry. The root node defines the whole mesh, while each children divides this mesh into smaller clusters. Because we are using quadtrees, each node has exactly four child nodes that contain the bounding box information and textures to be used. There nodes are mainly used for culling and thus speeding up the rendering process. Only the leaf nodes describe the geometry and are used for rendering. Figure 4 illustrates the quadtree structure with its bounding boxes and one triangle fan inside.

The problem we focused is how to handle large textures for the surface with the limited amount of memory space on the graphics board. In our example we have a satellite image from one part of Germany with a resolution of 5 meters per pixel. The whole image has a size of 27474x27474 pixels. This allows us to fly with our terrain engine above a surface that has 138x148 kilometers dimensions. The problem is that

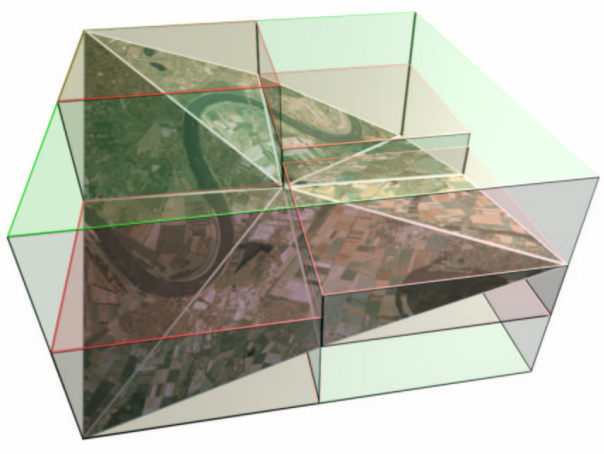

Figure 4: Illustration of the quadtree structure with its bounding boxes and one triangle fan drawn inside

we need about 2.3GB for the whole ground texture which is much more than current hardware can handle.

Tiling the surface image is influenced by the underlying hardware. Though most graphic cards can handle 2048x2048 textures or larger, we encountered a bottleneck at the AGB bus interface for transfering the data into memory. In our test environment we found out that 1024x1024 textures can be handled by the hardware without consuming too much time. However, we decided to tile our surface data into 256 smaller pieces. This is because each level of a quadtree has a number of child nodes which is a power of four. So when using 256 texture tiles each child on level 4 in our tree holds exactly one texture image. All the leaf nodes below will be rendered using this texture.

On the other hand, some branches are abandoned by culling. Therefore we implemented a texture prefetching routine that transfers only those textures into memory which will be used for the current rendering step. All textures outside the vicinity of our virtual camera can be dropped out of memory.

Because we cannot make any prediction what textures to use next when moving the camera over the surface, we increase the vicinity by a small amount to upload the image before it reaches the inside of our viewing frustum. The following picture shows the determination of surrounding ground textures. Gray colored tiles represent textures that must be stored in texture hardware memory. A bounding sphere with a radius equal to the far clipping plane is used to determine what textures might be become important for the next rendering pass. All textures which are inside this sphere or hit the boundary line must be resident in hardware. The dotted circle shown in figure 5 increases this area and surrounds all textures which will be possibly used in the next rendering cycles when moving the camera.

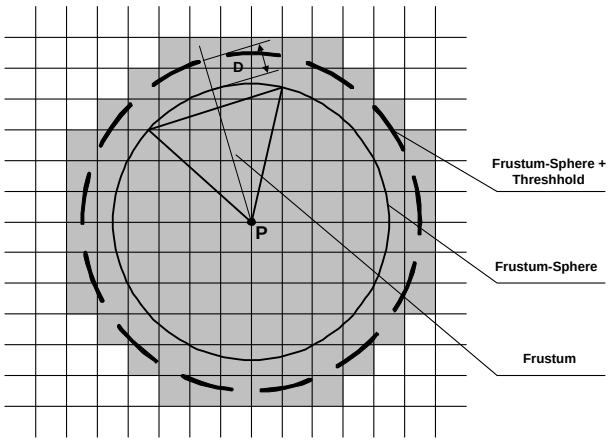

Figure 5: Determination of surrounding ground textures

The radius of the dotted circle (i.e. the increasement of the vicinity of textures) depends on the speed of the virtual camera.

$$Camera\ Speed \leq \frac{D}{Texture\ Loading\ Cycles} \quad (1)$$

Because a new texture load transfers a huge amount of data into texture memory (3MB for each texture tile in our case), it is necessary to split up the transfer of texture data over several rendering cycles. Therefore a second thread manages the texture uploads while the main thread renders the terrain and figures out what textures must be accessable.

To visualize the the sky we used a simple box geometry that is static in position to the camera. Six textures are mapped onto this box as shown in Figure 6.

Both in common give a very realistic impression of a real outdoor scene. Hemispheres have the disadvantage that they need a lot of polygons while sky box only consists of twelve triangles and can therefore be rendered in a very short time period. Our current work focuses on volume clouds that show the current weather for the area where the camera is located. Particle systems are used for rain, snow, or thunder strikes, respectively. Because the ground normally vanishes in the far haze and changes to the color of the sky some

Figure 6: Six textures for the sky box environment

kind of colored fog is used to simulate this effect when rendering the terrain. The only trick is that we must use the same color as the sky so the surface seems to fade away when looking in direction of the horizon.

The realtime rendered view in Figure 7 shows the river Rhine and the city Koblenz. The virtual sky as well as the fog in the background improve the visual impression.

4.1 Scalability and adaption of the terrain engine algorithm

The terrain engine was designed to run on different hardware platforms. It can be adjusted to constrains like frame rate or memory consumption in two ways by changing the quadtree recursion depth and the total mesh scale. Both factors influence the total costs of calculation time and memory usage. The terrain engine we developed consist of a 2 pass algorithm. While the first pass loads the heightfield data and builds the quadtree structure out of it, the second pass interacts with the texture manager and renders the graphic. The quadtree recursion depth has in impact on both the first pass and second pass of the algorithm. In the first pass this factor determines the memory demand to store all the geometry data as a quadtree structure into main memory. With the recursion level the amount of data increases exponentially.

$$N_{nodes} = \sum_{l=0}^{MaxLevel} 4^l \qquad (2)$$

It will certainly be a good idea to adjust this value according to the requirements of surface resolution and the applications needs. During the second runtime pass one can specify the depth to dive into the quadtree and to show all the geometry detail which has been prepared in the first pass. This is very important for framerate adaption. If the required framerate cannot be achieved the recursion depth can be decreased until the required performance is reached. Like anytime in computer graphics algorithms this is the common way to balance performance and detail quality. There is another kind of recursion depth to mention: the quadtree level which holds the textures. This level decides the principal texture impact of the running system. The texture tile size itself multiplied by the amount of tiles needed at the given texture output level of the quadtree count the total amount of texture data on the hard disc. But the amount of data which will be prefetched into system or texture memory depend mainly on the other factor we already talked about: the total mesh scale. Just to say it in first: this factor does not influence the quadtree data amount. It only comes into play during runtime as a main virtual adjustment for the texture management. By changing the total mesh scale the amount of prefetched textures and textures physically loaded into texture memory can be controlled. This is also where the frustum size becomes important. Frustum size and total mesh scale have a direct relationship in virtual world space. The more textures are found within the frustum surrounding sphere as well as within the frustum itself the higher the requirements on memory and data throughput on the target system will be.

Figure 7: Shot from a flight where the ground vanishes into the haze

4.2 Satellite images

The graphics in this paper was provided by the german company GAF [gaf02] which offers a variety of terrain data. Ground textures and heightfields are stored in geotiff format [Geo92], which is an extension to the ordinary tiff format with additional information like longitudes and latitudes stored as meta tags. This extra information is useful to map surface textures onto the mesh and to find the appropriate weather information so everything fits together even the data come from different providers. The heightfield in our demo scenario has a resolution of 1833x1972 which covers an area of 138x148 kilometers. The ground texture was an image of 27474x29572 pixels, where each pixel stands for approximately 5 meters in reality. This yields a very high detailed look of our terrain. Figure 8 shows the satellite image of a German section which was used in our demo application. The corresponding heightmap is shown in Figure 9.

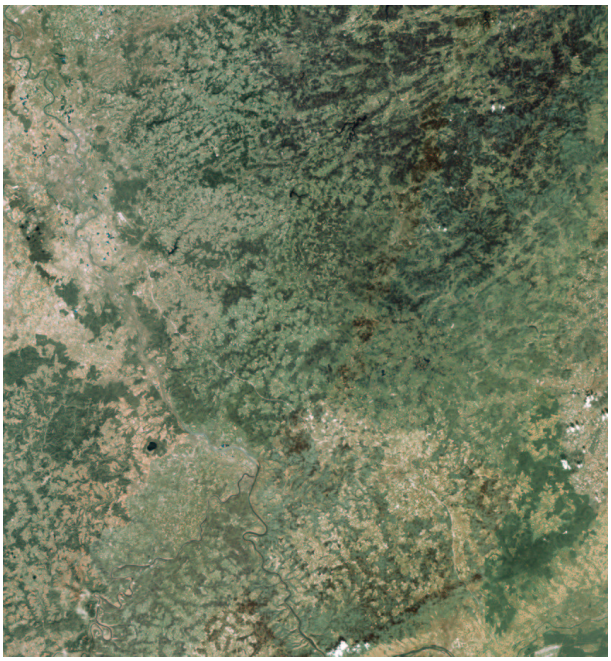

Figure 8: Section of the German ground texture provided by the GAF company

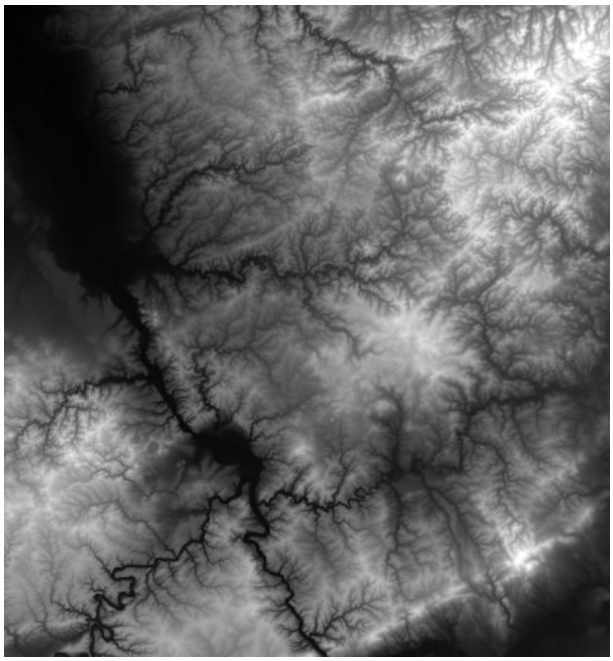

Figure 9: Heightmap fitting the area of Figure 8

5 Interaction and navigation

Rendering terrain flight for a weather forecast needs short term modifications of the animation. The operator gets the current weather data and the area of interest which should be visualized via the terrain flight. So he must be able to set the parameters for the virtual camera movement and some other settings for the flight very easily. Therefore we used a database with the locations for all the major cities of interest. The operator simply clicks on the locations which should appear during the flight and the system generates a simple spline out of it. Even though the animation can be fine-tuned afterwards, it should be sufficient to select the name of the cities to be shown and let the system do the rest. This realtime renderer can now produce the animation directly on air and is therefore a big improvement compared to conventional rendering packages which often need a lot of user interaction and often a huge amount of rendering time.

For a typical weather forecast it is also uselfull that the moderator can interact with the content in the virtual set. Pointing, highlighting or changing aspects can increase the possibilities for such TV shows.

While our prototype uses a standard PC keyboard for interaction control, devices like a stylus can become a sophisticated natural input device for a moderator. The control moves from behind the scene to the front of the camera. So the moderator can live interact with the information shown for the viewers.

6 Implementation and system design

The virtual studio software 3DK, developed at the Fraunhofer Institut Medienkommunikation, is used for rendering the weather flight over Germany where the terrain engine was integrated by the plugin interface directly into the scenegraph for the virtual set. Stylus and user control is managed by the user interface that controls the rendering engines over IP network.

Because our terrain engine is a graphical primitive like a sphere, even though it is more complex, we can place it together with all the other primitives into the hierarchy of the scene graph used for rendering the virtual set. The plugin technology of 3DK allows us to extend the application in an easy way with this new kind of visualization. The operator can specify terrain parameters like the source for our weather data via the user interface. The terrain engine integrates into to virtual studio software as it is yet another graphical object. Features like spline animations could therefore be used for moving the camera over the terrain. The plugin also controls the access to an external database where we stored the locations for the several cities that we can pass in our flight.

7 Conclusion

The implementation on PC hardware with consumer graphics accelerators gives sufficient performance for realtime terrain visualization.

Realtime interaction in live virtual studio production changes the way how data can be presented. Up-to-date visualization of complex data sets with direct interaction control is another step forward in the broadcasting business.

Currently, we work on a generic interface for 3D input devices like a stylus or wand for the 3DK software, which will allow new interactions for the actors.

8 Acknowledgments

This work were performed in the framework of collaboration agreement of the Fraunhofer Institute of Media Communication and the FH Düsseldorf, University of Applied Sciences, fostering the development of virtual studio technology with joint projects and providing an educational environment. The authors thank Wolfgang Vonolfen for supporting the integration into the 3DK virtual studio software.

References

[3dk02] *3DK*, `www.3dk.de`, 2002, 3DK Virtual Studio Solutions.

[DWS+97] Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein, *ROAMing Terrain, Real-Time Optimally Adapting Meshes*, Tech. report, Los Alamos National Laboratory, Lawrence Livermore National Laboratory, 1997.

[GAB+98] Simon Gibbs, Costas Arapis, Christian Breiteneder, Valio Lalioti, Sina Mostafawy, and Josef Speier, *Virtual studios: An overview*, IEEE Multimedia **5** (1998), no. 1, 18–35.

[gaf02] *GAF AG*, `www.gaf.de`, 2002, distributer for satelite images, München.

[Geo92] *GeoTiff Homepage*, `www.remotesensing.org/geotiff/geotiff.html`, 1992.

[Geo00] *Fast Terrain Rendering Using Geometrical MipMapping*, `www.flipcode.com/tutorials/geomipmaps.pdf`, 2000.

[Her01] Jens Herder, *Interactive content creation with virtual set environments*, Journal of the 3D-Forum Society, Japan **15** (2001), no. 4, 53–56.

[ifa97] *Wheather flight demonstration including a frog avatar in virtual studio demonstrated at the* IFA, `www.khm.de/~actor/productions/frosch/frosch.html`, 1997.

[Wal03] Oliver Walczak, *Entwicklung eines Algorithmus zur Visualisierung von Geländedaten mit hochauflösenden Texturen in Echtzeit*, Master's thesis, Fachhochschule Düsseldorf und Fraunhofer Institut für Medienkommunikation Sankt Augustin, February 2003, Diplomarbeit (master thesis in German).