🏆 *reviewed paper*

GEARViewer: A State of the Art Real-Time Geospatial Visualization Framework

Stephan Mantler, Gerd Hesina, Michael Greiner, Werner Purgathofer

(Dr. Stephan Mantler; VRVis Research Center; Donau-City-Strasse 1, 1220 Vienna, Austria; step@vrvis.at)
(Dr. Gerd Hesina; VRVis Research Center; Donau-City-Strasse 1, 1220 Vienna, Austria; hesina@vrvis.at)
(Michael Greiner; GEOCONSULT Wien, Hütteldorfer Strasse 85, 1150 Vienna, Austria; michael.greiner@vienna.geoconsult.at)
(Prof. Werner Purgathofer; VRVis Research Center; Donau-City-Strasse 1, 1220 Vienna, Austria; purgathofer@vrvis.at)

1 ABSTRACT

Geospatial visualization is playing an increasingly important part in the planning and public discussion of infrastructure projects. In addition to pre-rendered highly realistic imagery, interactive viewers have become an important tool for this task. Advances in rendering technology and performance have reduced the gap in visual quality between pre-rendered imagery and real-time applications, and the additional possibilities of a live visualization may become important tools in the decision making process.

Historically, the step from GIS data or a highly detailed architectural model to a representation that is suitable for real-time display has been complex and required a very finely tuned workflow. On the other side, the general public is relatively spoiled by the extremely high quality of computer games and CGI films. Naturally, a tremendous amount of work and time is typically spent to fully optimize computer games for the available hardware. This is usually not possible for geospatial visualization tasks, but nonetheless current interactive viewing applications must strive to achieve similar quality and performance to be successful.

In this paper, we present GEARViewer, a state of the art geospatial rendering framework developed at VRVis. Developed in close cooperation with major stakeholders in the Austrian road and railway infrastructure, it bridges the gap between GIS applications and real-time rendering, and achieves a high degree of realism and performance while supporting many of the tasks involved in geospatial visualizations.

2 INTRODUCTION

In the planning and public discussion of large infrastructure projects, many decisions need to be made that depend on a good understanding of how the project will fit into its environment. Traditionally, this has been done with 2D plans, architectural drawings or pre-rendered still and motion pictures. However, especially in the case of the general public, a more versatile, interactive approach may be helpful to provide a better understanding of the project.

The essential technology for interactive visualizations has been demonstrated by numerous games, where vast environments were displayed with formidable detail and realism, and it should certainly be possible to build a geospatial viewer application with similar capabilities. However, games are typically high budget productions with long development cycles, and the rendering system and 3D models are often finely tuned for the best possible quality and performance. This development style is not quite suitable for geovisualization projects, where changes need to be incorporated on short notice, and manpower is typically much more limited. Furthermore, data is often created in GIS or architectural modeling applications, and with different goals than real-time rendering. This typically means that the available models may not be directly suitable – they may be much too detailed, or only available as outlines, missing surface descriptions.

Therefore, preparing models for interactive visualization requires a different workflow and other tools than typical 3D game engineering. This also applies to the functionality available in the viewing application itself, which servers a different purpose than most game playing environments. From experience, especially in planning discussions, the main benefit does not lie solely in a photorealistic display. Much more essential is the capability of being able to explain complex interrelations, enrich common 2D planning data by embedding it in a spatial 3D context (possibly together with simulation input), and being able to answer complex questions live with a suitable toolset, which sometimes requires a certain degree of abstraction.

In the following text, we will present GEARViewer, an interactive geospatial viewer developed in recent years at VRVis in collaboration with large Austrian infrastructure providers and civil engineering consultants. After an overview of related work, we will discuss the basic architecture functionality of our system and provide detail on how various design decisions were made. Finally, we will present the workflow supported by our system, and show how it was used in two practical applications.

3 RELATED WORK

Basically all software currently in use in the field of planning, GIS and simulation has more or less feature rich and realistic visualization front ends in its portfolio, trying to serve the need for realistic 3D display. Depending on their respective evolutionary history, market demands and overall resources, the visualization capabilities vary from rudimentary (i.e. specialized simulation packages) to highly sophisticated as in some modern CAD, 3D modeling or GIS packages. Despite the impressive development in this area, existing systems may have their individual strengths but often also drastic shortcomings especially when it comes to real-time and interactive display. The main problems usually concern extensive data, either through high-resolution orthophotography, massive laser scans or highly detailed 3D models; smooth integration of numerous 3D data formats, render performance and/or additional output and interchange options.

As an example, ArcScene (the 3D front end to ESRI products with real-time capabilities [ARCSCENE]) already offers seamless GIS data integration and a high quality level of geospatial visualization, but has its limits when trying to enforce highest data resolutions, and the integration of data from external 3D modeling packages is laborious and sometimes even impossible. The resulting display quality (i.e. texturing, lack of shader effects) and performance lags behind state of the art real-time scenes (see Figure 1).

Fig. 1: Comparison of GEARViewer and ArcScene at 2 levels of detail. Left: GEARViewer showing 5cm orthoimage overview and close up (with line of sight GUI); Right: ArcScene with same dataset and viewpoints at maximum capacity

Google Earth [GOOGLE] shows a similar situation with extremely powerful imagery handling, a decent 3D model interface and also good rendering capabilities, but a noticeable weak spot in terrain resolution and handling, and generally limited interactive analysis tools that mostly focus on current situation data which is usually not satisfactory for planning projects.

The very broad and complex software branch of 3D modeling packages offers almost unlimited possibilities in 3D model generation as is impressively shown in various CGI movies. Together with GIS, many CAD and modeling packages pose a major backbone when it comes to visualizing planning projects. Various standardized and to some degree convertible export formats are the base for using this huge pool of powerful tools also for real-time output. Their main strength and focus today however lies in streamlined pipelines for rendered images and film output, which usually has conflicting requirements to real-time scenes. Therefore most of the sophisticated effects and tools cannot be made available in real-time models.

A number of real-time rendering engines are available today that make use of cutting edge hardware development and GPU features with impressive display effects, detail and performance. Examples for such systems include NVIDIA SceniX [NVIDIA], OGRE [OGRE], OpenSceneGraph [OSG], and many game engines (which also typically provide infrastructure for input handling, game entities, and animation and gameplay scripting) [GAMEBRYO, QUEST3D, VALVE].

Typically, rendering and game engines are only used as the foundation for the actual development, and a sizeable team of programmers and content creators cooperate to produce the final product. Therefore, while rendering and game engines do not provide the functionality sought in this project, they could certainly be used as the starting point for development, and in fact the rendering framework developed at VRVis and used as the foundation for development has quite similar functionality to OpenSceneGraph, with the additional benefit of the development team being already familiar with its codebase, and the core developers being readily available to fine tune and adapt its capabilities for our purposes.

4 SYSTEM OVERVIEW

In a typical application, the system is used to present various alternatives of a given infrastructure project. The entire scene can be interactively explored, switching between project alternatives, adjusting daylight and illumination parameters, taking measurements to provide exact visibility and distance information, and using a number of other features.

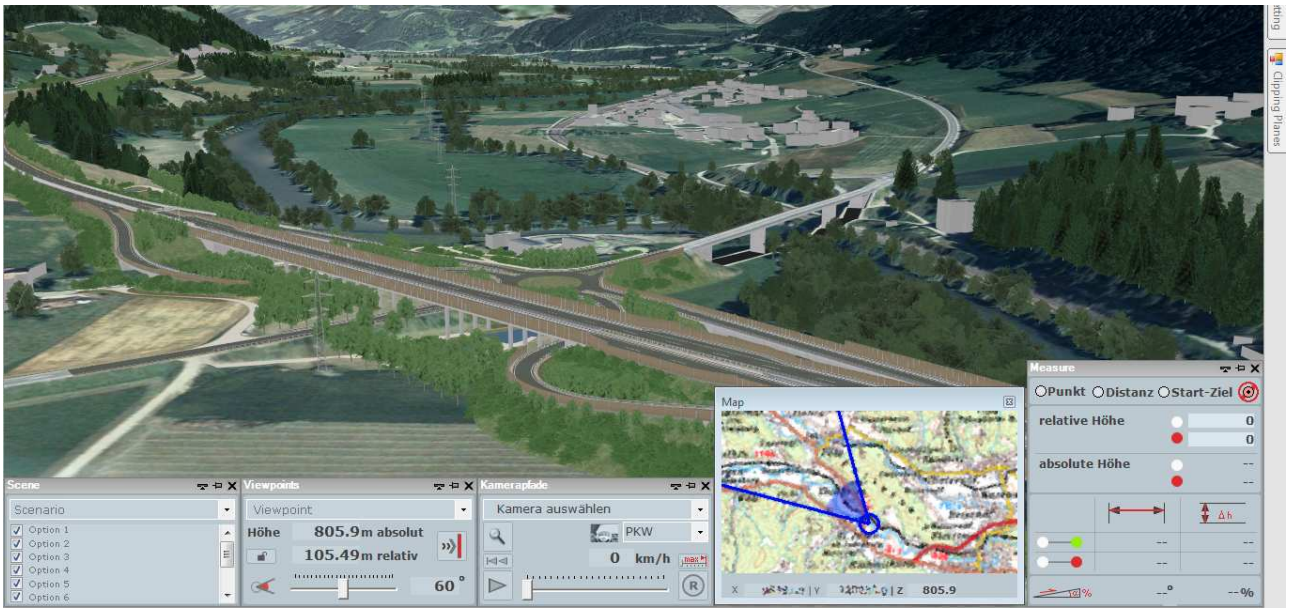

Fig. 2: GEARViewer main render window with various GUI elements.

Support for switching between alternatives is provided by allowing multiple scenarios (which may be different design options, present and planned system, etc.) for the presented scene. Each scenario is made up of a number of layers that can be individually shown or hidden at runtime. In turn, each layer consists of at least one original model; individual models are combined and packed at load time for efficient rendering. Therefore, switching between layers or scenarios is very fast once data is retained in memory.

The interrelation between models, layers and scenarios is stored in an XML based scene description that is loaded at startup. This scene description also controls which parts of the scene are used for testing against collisions and which texture and shader attributes to use for each model, defines the animation networks used for dynamic content in the scene, and provides other details.

To support loading a variety of models, a flexible import mechanism supports loading a variety of sources; its modular structure is easily adaptable to new data structures and file formats. All data is saved in an optimized, binary cache file format. These cache files can be loaded directly and used instead of the original source data. Therefore, a closed data package can be supplied to third parties instead of having to give away the original source data. Cache files are also interchangeable between scenarios and visualization projects, which provides advantages when complex imports require significant processing time.

Various overrides are possible at load time without the need for re-importing source data. This includes offsets, scale and rotation transforms, and assigning shader programs and other rendering properties. Doing this externally to the original model and cache files allows one to use a base model that doesn't need to be touched for fine tuning or model reuse, and also works for file formats that do not support these properties.

The GEARViewer UI (Figure 2) consists mainly of a full screen sized main render window showing the real-time scene as large as possible. Navigation is supported using a mouse, keyboard, game pad or spacemouse. The small GUI elements displayed at the bottom of the screen contain the most often used settings and are grouped thematically into i.e. viewpoint controls, camera paths, scenario handling, output or 2D map and can be minimized to title-bar size or closed completely if unneeded. Hidden in a roll-over sidebar are additional expert settings to gain access over the whole rich feature and parameter settings such as lighting and advanced HDR, animation or clipping plane control just to name a few. The arrangement and availability of GUI elements is configurable according to respective requirements and some elements appear automatically when triggering certain features such as the measure and visibility toolset or video render-output. Optionally, a full-screen stereo rendering mode (without GUI elements) is also available.

4.1 System Features

4.1.1 Geospatial File Formats

Traditionally, the exchange format for providing data from GIS applications to real-time rendering system required a detour to one of the mainstream geometry file formats (VRML, 3DS, OBJ, COLLADA, etc.). This required additional steps in the workflow to prepare the geometric model and often resulted in models that were not ideally suitable for rendering. To support a streamlined workflow and to optimize rendering quality and performance, we decided to natively support a number of geospatial file formats. Specifically, our system handles georeferenced imagery (like orthophotos), ESRI shapefile data, and ArcGrid raster data natively, providing a range of import options for creating geometry from GIS input data. Of course traditional model formats like VRML are also supported.

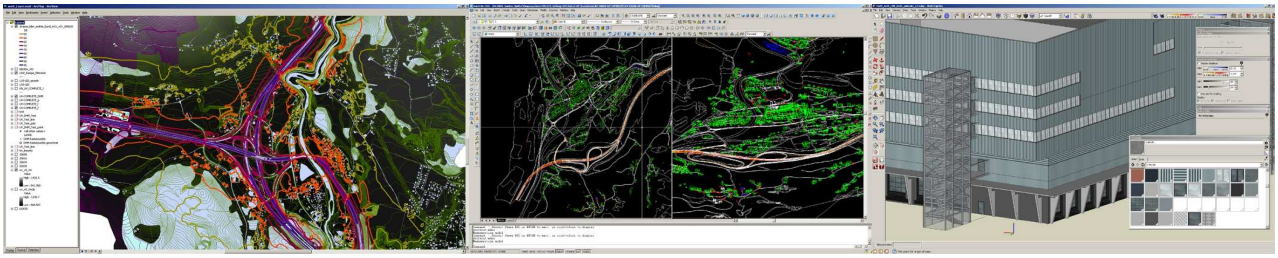

Fig. 3: The real-time scene combines source data from numerous GIS, CAD and 3D modeler applications at various levels of detail.

Typically, the working space of our system is the Austrian MGI projection, and the framework assumes that input data is all provided in the same georeferenced projection using cartesian model coordinates. Automatic reprojection from other reference systems is only partially supported, but we intend to extend this functionality and provide full functionality in other working spaces in the future.

Georeferenced imagery can be used for applying textures to arbitrary geometry (both created from shapefiles or raster data, and models loaded from VRML files or any other input file format); they can also be used as a data source for the 2D map view. Textures are reprojected to the working space of our system as required.

ESRI Shapefiles are supported using PointZ, (Poly)LineZ and (Multi)PolygonZ features. Automatic lifting of 2D data to the scene is currently not supported, since there is no provision for specifying which parts of the scene should be used for lifting. Upon import, shapefile coordinates and database attributes can be used to influence geometry creation. Our system supports arbitrary calculations through C# expressions that are compiled on the fly and evaluated for each data point. For example, this can be used to control extrusion parameters or scaling of individual instances based on database fields.

Shapefile information can then be used in a variety of ways. Point data can be used for automatic placement of hardware instanced geometry like trees, using shapefile attributes to select individual models and transformation parameter. Linear features can be displayed as is (for example, color coded to show isophones from highway noise simulations) or used in combination with user defined cross section profiles to extrude simple features such as noise barriers or fences. Finally, polygonal information can also be extruded vertically; for example, this feature has been used extensively to create block models from building outlines, using a simple concrete texture for the sidewalls and georeferenced orthophotography for the top surface.

ArcGrid raster data is ideally suited for creating large terrain models. The original raster data is used to create a level of detail hierarchy so that distant parts of the terrain model can be rendered at lower resolutions to optimize resource and performance costs. Since ArcGrid data may contain values signifying missing data, these data points can be either left out (creating visible holes in the resulting terrain model) or mapped to a predefined value. For example, this functionality is used to create space in terrain models where excavations need to be displayed (and the original surface in the DEM would interfere), or where smaller, higher-resolution insets are available. Again, the georeferenced image data mentioned above is used to apply surface detail to the terrain model. Both the level of detail selection strategy and the texture resolution for each level of detail can be specified by the user for optimal results in each application case.

Due to the modular architecture of our framework, integrating further data formats in the future is straightforward, and does not impede any existing workflow.

4.1.2 Time System

'Real time' is a difficult concept for presenting large geospatial projects. For example, it may be desirable to present the looks of a new building over the course of a day compressed into a few minutes, or to adjust the animation time system to quickly populate animation networks on startup or to handle non-real-time output properly. Therefore, our system includes three different time systems: *animation time* is used to determine vehicle speed; *environment time* controls lighting effects such as sun position and shadows, and *output time* is typically real-time for interaction, but for rendering video output is set to a fixed rate per rendered frame. Animation and environmental time can be either stopped (frozen) or coupled to output time using varying factors. Therefore, vehicle animation may for example be slowed down to be less distracting, and environmental time may be simultaneously sped up by a large factor to present a full day/night cycle in a short period of time. Additionally, environment time can be freely set to arbitrary values to quickly view the scene at a specified date or time.

4.1.3 Path-based Vehicle Animation

In many cases, enriching the presented environment with vehicle traffic or other dynamic content gives the end user a better impression of the effects of the planned project. Two variants of such an animation subsystem are currently under development, and may coexist within any scenario. One is suitable for vehicles that are strictly bound to existing paths, like trains which must of course always remain on tracks, while the other is geared towards more autonomous vehicles, allowing cars to react to their environment and change lanes. Both variants also support vehicle chains to simulate trains or trailer trucks.

The basic structure is essentially the same: each animation network is defined by a number of polylines within a shapefile, with the line itself providing the central axis of an animation path, and attributes within the shapefile being available to define the intended vehicle direction, maximum speed, information how paths might form multiple logically connected lanes of a road) target traffic density, and other details.

For each network, the system then identifies possible source and drain nodes. To release a new vehicle, a random source is chosen and a random path to a drain node is found according to the targeted traffic density.

Vehicles are rendered using hardware instancing: the geometry and texture data of each vehicle model only exists once, and is repeated on the GPU hardware as often as required using appropriate transformations for each vehicle position. To improve diversity, colors can be adjusted for each instance through a custom pixel shader. A simple modulation of the present (texture) colors typically does not produce the desired result, so we use a separate texture color channel to modulate between the original texture and the per-instance color. In our system, this information is stored in an illumination map that accompanies most models to include self-luminance information for nighttime scenes. However it would be equally possible to use, for example, the alpha channel of an RGBA texture.

Vehicle position updates are constantly calculated in a separate thread, and are therefore independent of the current render frame rate. For strictly path bound vehicles, all path segments from source to drain node are pre-calculated and merged into a single path upon vehicle creation. This merged path is then used to animate the vehicle at constant speed. After each position update, front and rear axes of each vehicle are explicitly constrained to remain on the path; in the case of vehicle chains the training sub-vehicles are also aligned on the path behind the main vehicle. On the other hand, autonomous vehicles only receive their target endpoint, and are designed to automatically find a suitable route. They can exploit logical connections between paths forming multiple lanes of the same road, and therefore change lanes for more realistic behavior. These vehicles are also aware of other traffic and adjust their speed accordingly to avoid collisions. This increased realism also comes at a higher computational cost, however in our current prototype the main bottleneck is still displaying larger numbers of vehicles, and not the calculations required for updating their positions.

4.1.4 Environment

Our system supports a geospatial reference location to accurately depict the sun position and lighting on a given longitude and latitude. Currently the sky is rendered using a Preetham day sky model [PREETHAM] and optional cloud layers. In addition, the sun position is used for calculating dynamic illumination and shadow mapping of the scene as well as day/night changes. Many parameters of the system are user tunable, such as shadow map extents, resolution and biasing, illumination intensity and shadow contrast, and enabling high dynamic range rendering with support for advanced effects such as blooming and dynamic brightness

adaptation. This enables the user to fine tune the environment for specific visualization purposes, and store and recall all parameters to quickly switch between different setups.

5 SOFTWARE ARCHITECTURE

Our viewer is based on a general purpose rendering framework that is under active development at VRVis and used in a large number of applications. The presented application is currently one of the primary users of this library, and its specific requirements are therefore actively driving its development, with integrated support for geospatial file formats, special functions for georeferenced imagery, and the capability of handling very large data sets. It was developed in C#, a language that greatly facilitated the multithreaded design of our framework and provides automatic memory management which minimizes the effort required for tracking resource usage and avoiding memory leaks.

Internally, the scene description explained in the system overview is converted into a hierarchical scene graph data structure, which is then traversed at runtime for rendering. This hierarchy maps very well to the scene/scenario/layer/model concept, and switching between scenarios or enabling or disabling layers simply activates or deactivates parts of the graph at render time. The hierarchical nature of the graph also facilitates geometric queries into the graph, since large parts can be quickly excluded when testing for possible intersections or visibility.

5.1 Render Precision

Particular care must be taken to avoid numerical precision problems given the enormous extents of typical geospatial visualization projects and the fact that current graphics hardware typically only supports floating point calculations with limited precision, which causes rendering artifacts due to numerical rounding errors when using global coordinates or even a geospatial coordinate system that uses large offsets (such as UTM).

Instead, our system uses a floating origin approach and local coordinate frames for all models. Local coordinate frames have the advantage of cache files that are independent of the actual placement of the model within the scene, and which are therefore easily reusable across projects. At runtime, the placement offsets and scene origin are resolved internally relative to the current camera position, such that offsets sent to the GPU for objects close to the viewpoint are guaranteed to be small, and therefore rendered at the best possible precision.

A similar problem exists with the z-buffer algorithm used by graphics hardware to determine visibility [CATMULL]. Sub-optimal parameters either cause scene elements that should be visible to disappear because they are outside the clipping range, or cause visual artifacts (z-fighting) when the limited numerical resolution of the buffer causes scene elements to be resolved incorrectly [HEDENUS]. The latter problem is particularly problematic: for example, a typical model of overhead signs requires a resolution of 5cm to display correctly, and is visible at large distances. We have found that no single strategy was suitable in all cases, so we have implemented a number of user selectable strategies that provide various tradeoffs between scene complexity, display quality and render performance.

One approach that requires very little computational cost is adjusting the near and far clipping distance based on the absolute height (relative to the scene origin) of the viewpoint. This provides a good approximation to viewing objects at close range when near the ground, and being able to view distant objects when high above ground. We have found that this approach works quite well for relatively flat terrain.

If the environment is significantly more profiled and intersection calculations to the ground surface are available, this approach can be modified to use above ground level (AGL) altitude instead of absolute height. This requires only one ray cast, and works well if the ground model is watertight. If it is not, and the ray cast misses an intersection, possible strategies may be to re-use the previous ground level, to perform a second, slightly offset ray cast, or to fall back to the reference-z approach. Furthermore, these ray casts are often performed anyway to display AGL elevation to the user, and are therefore essentially available for free.

However, this approach only take into account the geometry that is directly below the viewer, and therefore cannot incorporate structures that stand high above ground. One example would be viewing a bridge spanning a deep valley, where the system would estimate a high altitude above the ground below, while in actuality the bridge may be very close to the viewer. Our system can use kd-Trees for efficient intersection calculation, and this data structure also supports closest-point queries, such that it is quite easy to find the

closest triangle vertex; this can then be used for adjusting the near clipping plane appropriately. The far clipping distance can then be adjusted using either of the aforementioned strategies.

In summary, none of these solutions provide the best result in all possible scenarios, especially since significant parts of the environment are often excluded from intersection calculations for performance and memory efficiency. However, we find that one of these approaches typically works well for a given scene.

5.2 Texture and Shader Management

Texture maps are crucial for adding detail to objects, and are heavily used in typical geospatial visualization scenes – often adding up to hundreds of megabytes of texture memory. In addition to this massive resource consumption, frequent switching between textures has a strong negative impact on render performance. Therefore, optimized texture handling is crucial for the overall performance. In our system, textures can be specified in many common image formats to facilitate source input. Textures are then internally converted to the compressed DDS1, 3 or 5 formats [IOURCHA] depending on the source texture and input settings. In addition to the common diffuse color textures, additional textures (for example, for illumination or normal mapping) are supported by looking for specially named texture files next to the original data. This allows even models created in formats that do not support multi-texturing to be created using standard tools and subsequently augmented with additional texture information.

To streamline the model creation workflow, special support was added to automatically apply georeferenced images as textures to arbitrary input models. Therefore, models can be supplied in formats that do not support texturing, or created in applications where applying area-wide images as textures is cumbersome. For example, this approach is used to automatically load and texture polygonal features from shapefiles and raster digital elevation maps without the need for manual interaction (see also Section 6 for details).

Where possible, multiple small textures are packed into larger atlases to reduce the number of context switches while rendering. Special care must be taken if textures are used as repeating patterns, and to avoid bleeding into adjacent textures when using filtered texture lookups or mipmapping; in our system textures receive a predefined padding that is sufficient for filtering and a limited number of mipmap levels. Textures that are found to be sensitive to such artifacts can be specifically excluded from packing.

As has been mentioned earlier, shader programs and other rendering parameters are decoupled from the actual geometry, and are assigned at special nodes during the traversal of the scene graph (and therefore apply to entire subgraphs and the models represented therein). They are therefore easy to change or replace, and in fact can be reloaded at runtime to facilitate debugging and fine tuning shader parameters. However, adding new shaders to the system at runtime is currently not supported.

The shader infrastructure is partially exposed to the end user, with much detail abstracted away but sufficient flexibility to allow users to essentially create arbitrary shaders. This functionality has been used for hardware instancing, per-instance model variation, special lighting effects (reflection mapping, animated textures, and others), and simple vertex animation in cases where a full-blown animation network would be overkill.

6 WORKFLOW

The workflow from raw project data to the final interactive scene varies in detail according to available input data, project need and of course the aspired complexity, display quality and level of detail which is directly constrained by the overall budget. Exceeding basic GIS data visualization, further refinement for more detailed planning projects or special questions (as shown later on) is always possible and usually realized by integrating more or less detailed additional vrml models into the scene. The model creation itself is not elaborated further in this text, as the range of software products capable of producing detailed 3D models is vast, as are the available export formats with their individual strengths and weaknesses. Vrml is only used as a container format and was chosen due to its wide propagation and sufficient features.

Care was taken to keep the necessary manual preprocessing steps as low as possible. The choice for ArcGrid as format for the digital terrain model (DTM) was made because it is fairly efficient, well documented and therefore also established in a large range of commercial products. The shapefile format is also thoroughly documented and widely available. Essentially, the manual preprocessing steps are the selection and is thematic filtering of source data according to their intended use, and converting 2D vector-geometry (i.e.

isophones) to 3D by conforming to terrain elevations for use as thematic overlays (since this step is not currently available in our framework).

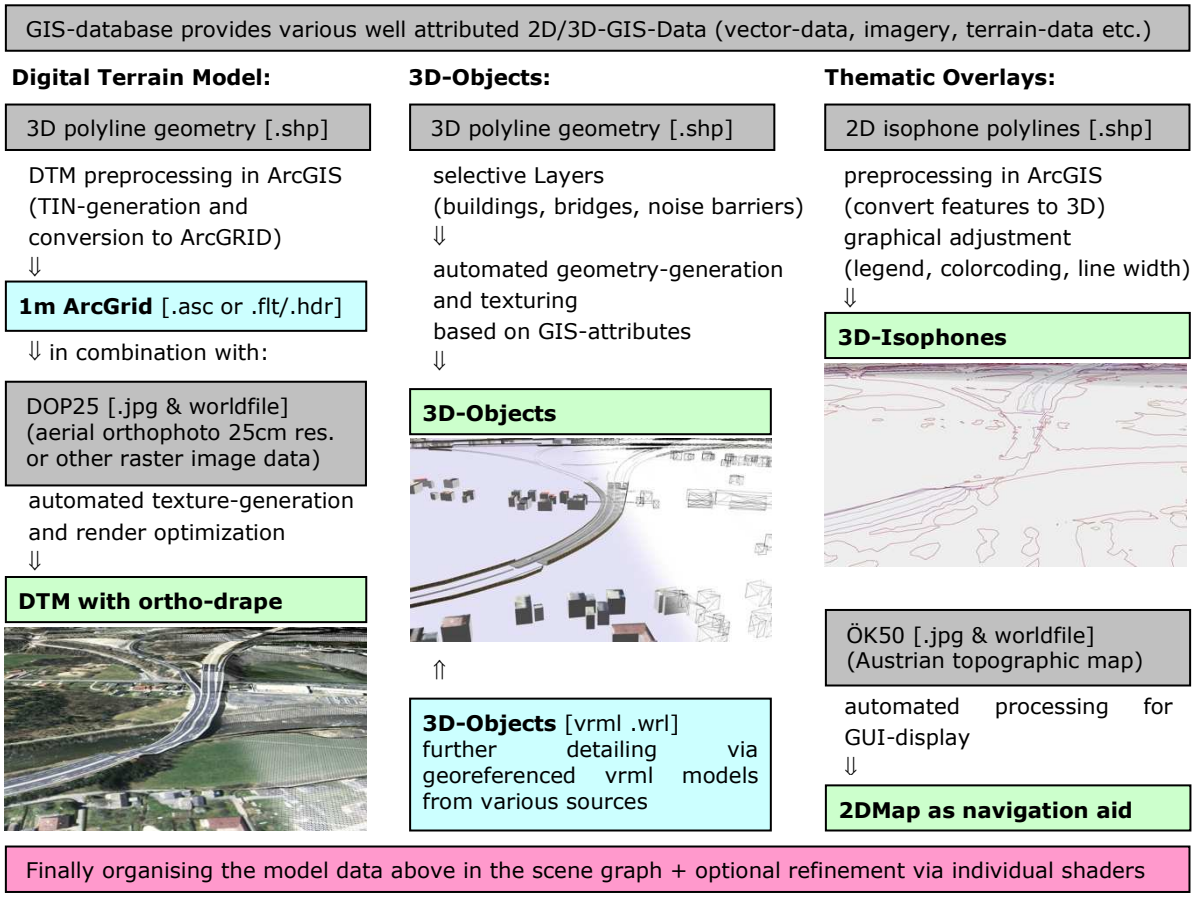

Fig. 4: Workflow from raw data to combined interactive real-time scene in GEARViewer.
This example is focused on fully automated scene generation based on readily available data.

6.1 Example: ASFINAG – Dynamic Noise Map

The goal of the "Dynamic Noise Map" project, developed in close collaboration with ASFINAG (primary federal road network provider in Austria) was to enhance the substantial AFINAG GIS information by displaying terrain data together with thematic overlays in an interactive real-time environment with additional usability features. A major demand was to completely eliminate manual processing steps for the scene, in order to provide similar models in the future for any other given area covered by the database "at the push of a button". This was to be demonstrated on a previously defined sample area.

Input data from the ASFINAG GIS database provided all necessary information to build a high-resolution landscape model of the chosen motorway segment and its surroundings. The data itself was initially

generated mostly by an area-wide aerial survey and is serviced by ASFINAG GIS-staff and its survey subcontractors who also provide the same data daily in a 2D intranet web service application.

The extent of the chosen project area was ~12 x 15km resulting in an area of 192 km² at a DTM raster resolution of 1m² covered with aerial imagery at a resolution of 25cm per pixel. A close-up range with the same DTM resolution was defined with 5 x 7km (35km²) but at a much higher orthoimage resolution of 5cm.

The terrain grid was generated from contour lines, break lines and points. 3D object information was available as 3D line/polygon features, either being a baseline or in case of buildings and bridges the top edge with the corresponding attribute information such as building- or noise barrier-height. Depending on geometry and object type, different extrusions and textures were applied to generate suitable scene elements.

After sifting through the data structure and available geometry and attributes, the vector-data was split into individual content related shapefiles (i.e. buildings, bridges, noise barriers, guard rails, etc.). For each shape, suitable import parameters were configured, managing all necessary parameters such as extrusion height, texturing, capping and transformations. Similarly, the parameters for 2D navigation map creation from orthophotos and for the terrain generation were also tested and defined.

These parameters were then used to create the central scene description, using predefined file names and locations for input data. Therefore, subsequent exports from the GIS database using the same structure could then be used to automatically create preprocessed cache files and visualizations for arbitrary areas, without manual interaction. The resulting scene can then be used as is, or further refined manually using specific shaders, custom textures or additional models as required. Figure 4 illustrates the presented workflow.

The application enables the user to move freely in an interactive real-time environment, viewing & analyzing the GIS data at its highest original resolution (1m² DTM with 5cm Ortho!). Also being able to modularly enable and disable data layers and switch between predefined sets and scenarios (i.e. day/night noise isophones), together with measure and line of sight analysis tools.

6.2 Example: Usage in various traffic infrastructure and flood protection projects

The functionality and performance of GEARViewer has also led to utilization in several other planning projects with large extents, very high detail and specialized requirements, where the provided technology resulted in a very positive response throughout the planning process with public presentation and discussion. In many of these cases, the projects required the integration of models and data from a wide range of planning companies and consultants. Typically, GIS data was used to provide large-scale landscape features, while the detailed object level was dominated by CAD data. Depending on the capabilities of the planning team, software and output formats, detailed objects such as buildings or bridges were built from scratch out of 2D plans and cross-sections, or directly integrated via textured 3D mesh exchange formats such as 3DS, OBJ, VRML, etc. However even if a 3D model already existed in various design programs, some manual refinement was often required in order to make it suitable for real-time rendering. These tasks included proper georeferencing, retexturing, reduction of overboarding detail and geometry integrity checks to avoid coplanar geometry and other artifacts. Some of these issues can now be corrected automatically during import, or can be modified later on via shaders or other render parameters.

After the base scenes had been assembled, custom shaders were then used to further improve the overall visual quality in a number of ways (see also Figures 5 and 6):

- Illumination maps: simulating lights/ night scenarios (skyline, cars, tunnel safety equipment, etc.)
- Reflections: realistic architectural materials (glass facades or reflecting surfaces, etc.)
- Animated textures and texture movement: simulating water flow and all kinds of time based material changes also in combination with Illumination maps (clouds, signal beacons, billboards, etc.)
- Geometry position interpolation: simple geometry animation according to various functions (rise and fall of water levels, moving tower cranes, circling helicopter, etc.)

Scenes were populated with multiple animation networks for car and train traffic as well as other moving object. Count and movement of traffic are roughly based on externally simulated data to give an approximate impression of the actual traffic density.

Fig. 5: Animation networks and rendering effects with real-time shadow maps and HDR rendering

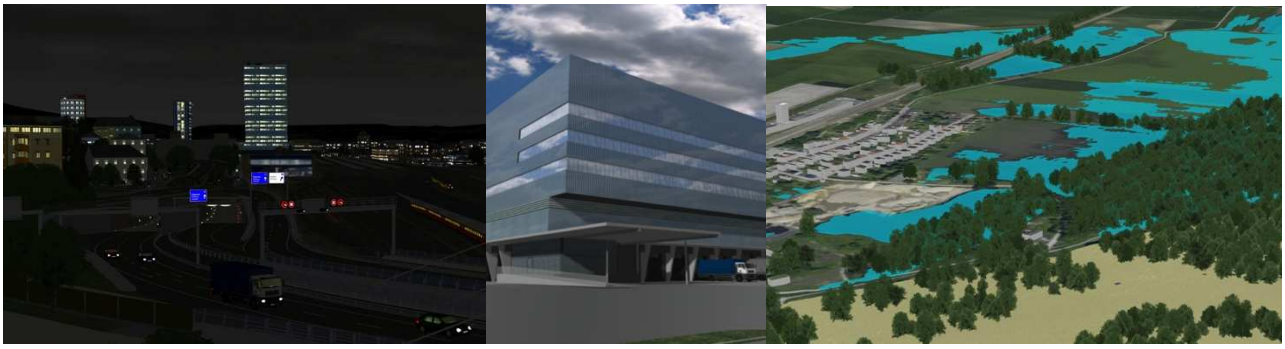

Fig. 6: Real-time shader usage: illumination maps, reflection maps, texture & geometry animation

7 CONCLUSION AND FUTURE WORK

The goal of GEARViewer and its preceding developments has been to create a versatile framework for displaying very large geospatial data in a real-time environment, with suitable handling and UI features for presentation purposes. The system supports a streamlined content creation workflow that integrates geospatial file formats for optimal performance and customizability.

In the future, we will continue to improve the handling and rendering performance for geospatial data. For dynamic content, we will investigate how high-level traffic analysis results for day or night traffic or changed project scenarios can be used to realistically populate the scenery with individual vehicles in a way that integrates well with the different time systems and usage scenarios. Additionally, a more capable scripting environment may be useful to control intersection lights, and other dynamic content.

Finally, safety simulations (for example, flood, water flow, or fire and smoke spread) also overlap and affect the infrastructural design process, and we are investigating the integration of such effects in the visualization.

8 ACKNOWLEDGEMENTS

VRVis is a COMET K2-Centre, supported by the Austrian Research Promotion Agency (FFG) project no. 279647. Development of the application was performed in collaboration with ASFINAG and GeoConsult.

9 REFERENCES

ARCSCENE: http://www.esri.com/software/arcgis/extensions/3danalyst/index.html, retrieved Feb 28, 2011.
CATMULL, Ed: A Subdivision Algorithm for Computer Display of Curved Surfaces.Ph.D. Thesis, Report UTEC-CSc-74-133, Computer Science Department, University of Utah, Salt Lake City, UT, 1974.
GAMEBRYO: http://www.gamebryo.com/, retrieved Feb 28, 2011.
GOOGLE: http://www.google.com/earth/index.html, retrieved Feb 28, 2011.
IOURCHA, Konstantine I., NAYAK, Krishna S., HONG, Zhou: System and method for fixed-rate block-based image compression with inferred pixel values, US Patent No. 5956431, Santa Clara (USA), 1999.
HEDENUS, Michael: Z-Buffer Precision and Frustum Planes. http://www.hedenus.de/zbuffer.pdf, retrieved Feb 28, 2011.
NVIDIA: http://developer.nvidia.com/object/scenix-home.html, retrieved Feb 28, 2011.
OGRE: http://www.ogre3d.org/, retrieved Feb 28, 2011.
OSG: http://www.openscenegraph.org/projects/osg, retrieved Feb 28, 2011.
PREETHAM, A.J., SHIRLEY, P., SMITS, B.: A Practical Analytical Model for Daylight. Published in Proceedings of SIGGRAPH '99, pp. 91-100, New York (USA), 1999.
QUEST3D: http://quest3d.com/, retrieved Feb 28, 2011.
VALVE: http://source.valvesoftware.com/, retrieved Feb 28, 2011.