Interactive Exploration of Coastal Restoration Modeling in Virtual Environments

Andreas Gerndt*[a], Robert Miller[b], Simon Su[c], Ehab Meselhe[d], Carolina Cruz-Neira[c]
[a]German Aerospace Center (DLR), Simulation and Software Technology, Brunswick, Germany
[b]C. H. Fenstermaker & Associates, Inc., Lafayette, LA, USA
[c]Louisiana Immersive Technologies Enterprise (LITE), Lafayette, LA, USA
[d]Center for Louisiana Inland Water Studies, University of Louisiana at Lafayette, LA, USA

ABSTRACT

Over the last decades, Louisiana has lost a substantial part of its coastal region to the Gulf of Mexico. The goal of the project depicted in this paper is to investigate the complex ecological and geophysical system not only to find solutions to reverse this development but also to protect the southern landscape of Louisiana for disastrous impacts of natural hazards like hurricanes. This paper sets a focus on the interactive data handling of the Chenier Plain which is only one scenario of the overall project. The challenge addressed is the interactive exploration of large-scale time-depending 2D simulation results and of terrain data with a high resolution that is available for this region.

Besides data preparation, efficient visualization approaches optimized for the usage in virtual environments are presented. These are embedded in a complex framework for scientific visualization of time-dependent large-scale datasets. To provide a straightforward interface for rapid application development, a software layer called VRFlowVis has been developed. Several architectural aspects to encapsulate complex virtual reality aspects like multi-pipe vs. cluster-based rendering are discussed. Moreover, the distributed post-processing architecture is investigated to prove its efficiency for the geophysical domain. Runtime measurements conclude this paper.

Keywords: Coastal Restoration, Virtual Reality, Scientific Visualization, Large-scale Datasets, Time-dependent Visualization, Hybrid Data Visualization

1. INTRODUCTION

Louisiana's coast is one of the largest deltaic systems in the world and a crucial natural and economic resource for the State and the USA. As a result of 19th century river management and a complex interaction among natural subsidence, sea-level rise, canal dredging, hurricanes, and oil and gas extraction, the coastal landscape is disappearing with over 1500 sq miles lost to open water since 1956. The water and sediment of the Mississippi River (annually 600,000 cubic feet per second and 120 million tons of sediment) are vital land-building resources now funneled directly into the Gulf of Mexico. [10]

Comprehensive, large-scale solutions to retain these valuable resources to restore the Louisiana coast are essential. This project seeks to integrate ecological and geomorphological tools to forecast the feasibility and long-term impacts of landscape-scale restoration strategies. This includes large-scale diversion scenarios of the Mississippi River, the Gulf of Mexico, and the Chenier Plain wetland.

Visualizing dynamic morphological simulation data interactively in immersive environments is a tremendous challenge. Not only unsteady simulation datasets become large pretty quickly but also the terrain used to represent the landscape may show an enormous resolution depending on the accuracy of the measurement techniques. Additionally, high resolution satellite images are mapped on the terrain. For the virtual reality (VR) integration, powerful open source tools were incorporated. The basic post-processing functionalities are provided by the Visualization Toolkit (VTK) [13].

As the coastal restoration project is just one example of a series of scientific oriented applications with similar requirements, we furthermore developed an additional software layer called VRFlowVis which combines basic features to more comprehensive objects which eventually offers the application developer a neat application programming

*andreas.gerndt@dlr.de; phone +49 531 295-2782; fax +49 531 295-2767; www.dlr.de

interface for fast scientific application prototyping without caring about the underlying technology in use. This layer also covers the integration of multi-pipe environments as well as cluster-based visualization systems. This transparency makes it much more complicated for designing an appropriate architecture in VRFlowVis. But on the other hand, the beneficiary is again the application developer who almost only has to know how to develop scientific visualization applications in his or her specific engineering domain.

The remaining paper is structured as follows: The next section reflects related work in the field of large scale data in virtual environments. Then, a set of geomorphological models are presented important for the coastal restoration in Louisiana. With a focus on the Chenier Plain region (cf. Fig. 1), various sources of used data and the approaches to process these large datasets for interactive visualization are presented. The successive section describes the distributed VR framework which has been developed to manage massive models. Runtime results prove its efficiency and a discussion for future work concludes the paper.

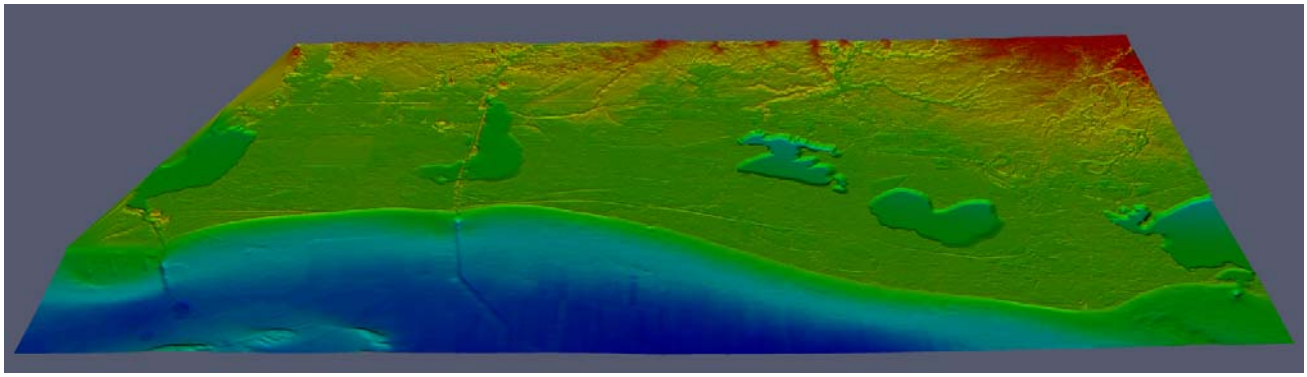

Fig. 1. Chenier Plain of Louisiana, one of the models considered for the coastal restoration project. Elevation is color-coded.

2. RELATED WORK

One of the first available systems for VR-based flow visualization was the Virtual Wind Tunnel presented by Bryson et al. An extension connected the virtual environment to a post-processing back-end which was responsible for the feature extraction [1]. The data flow and applications of distributed post-processing for interactive visualization is also investigated by Kuhlen et al [6]. Their framework, as most of the applications available nowadays, makes use of post-processing algorithms implemented in the Visualization Toolkit (VTK) [13]. The work presented in this paper also uses only VTK algorithms.

The post-processing of large terrains based on LiDAR point data is the focus of the work by Keylos et al. [5]. Jimenez et al. additionally considers time-varying processes for coastal observations [7] whereas Yoon et. al. presented view-dependent approaches to enable interactive rendering of such massive models [14].

3. COASTAL RESTORATION MODELING

The coastal restoration project described in this paper studies different regions of southern Louisiana. Because of the diversity of the data of interest, different solvers are used to compute time-dependent data fields. Additionally, collected data from e.g. field measurements are available to be used for comparison purpose as well as for combined visualization approaches. Following three scenarios are selected:

- Scenario I: Chenier Plain. This is a regional model of the landscape located between Interstate 10 and the Gulf of Mexico coastline, and between Texas state line and Vermilion Bay. A commercial code produced a two-dimensional model of hydrodynamics and salinity results in channels and the marshland [8]. Also available are three-dimensional rainfall radar data and information how the system responds through increased runoff and marsh inundation.
- Scenario II: Northern Gulf of Mexico. This three-dimensional model of hydrodynamic, salinity, and suspended sediment concentration covers the region of the northern part of the Gulf of Mexico. Of particular interest are morphological changes like erosion and deposition of the near-shore Gulf bottom. The simulation is based on open source code.

- Scenario III: Lower Mississippi. Morphological and sediment transport modeling of an approximate 20-mile reach of the lower Mississippi river south of New Orleans have been employed by an open source solver. The main aspects of this simulation are suspended sediment concentration and erosion / deposition of the river bottom on response to the flow field.

Despite the fact that first preliminary results of scenario III are already available, the focus of this paper is set to the integration of the heterogeneous data produced for the Chenier Plain. Even the regular 3D simulation dataset for the Gulf of Mexico is not incorporated into the current version of the application although the bathymetry of the northern Gulf of Mexico is also used to create a high resolution terrain for scenario I.

3.1 Chenier Plane Simulation Data

The simulation dataset for the Chenier Plain consists of salinity, water level, and water surface velocity fields on a 544 x 145 rectilinear 2D grid. The solver produced 365 time steps, one step per day for the year 2003. The topology of the grid is defined implicitly by specifying the origin in NAD83 (North American Datum, 1983) coordinates (UTM zone 15) and the easting / northing space for the grid point distance in meters. Additionally, a regular terrain grid with a resolution of 560 x 160 grid points is available. It also contains a part of the bathymetry of the Gulf of Mexico, the lakes, and the marshland of the Chenier Plain. Height values outside the considered Chenier Plain region are not valid and set to 10 meters which is high enough to be distinguished from valid points of the generally flat region. Furthermore, the simulation data only contains valid points where water (lakes, marshland, Gulf of Mexico) is defined. Invalid points are set to -0.0.

Although both grids show different resolutions, grid points in the x-y plane are coincident. For a straightforward visualization, all simulation quantities were combined to one dataset based on a regular grid where each point is stored with water level as z-component. This yielded a 3D mesh moving in z-direction over time. Visualizing both grids simultaneously presents valid simulation data only at points above the terrain grid. Invalid points are always hidden below the terrain (cf. Fig. 2, right). With this approach, it is also possible to evaluate the appearance and disappearance of marshland over time. To avoid render artifact (e.g. z-fighting) and to emphasize the structure of the (plain) region, the z-direction was scaled by 100. Eventually, the terrain grid and the simulation grid were cropped where useful information is stored.

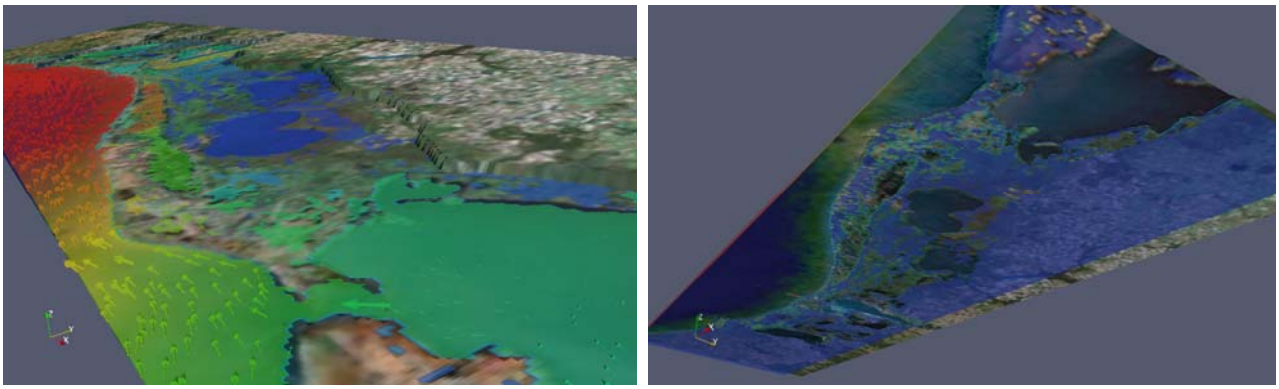

Fig. 2. Unsteady visualization of salinity surface (with changing water level over time) and water surface velocity as arrow glyphs over a coarse textured terrain (left), invalid simulation values (blue) below the terrain (bottom up view, right)

A common approach to visualize scalar fields on surfaces is to determine the color of each grid point by applying a user definable lookup table to the scalar value. Color values between grid points are interpolated. For the salinity, a semi-transparent surface rendering enables an improved impression of the underlying bathymetry. The velocity vectors are drawn as e.g. arrow glyphs using speed to define the length (and width) and salinity to specify the color of a glyph. To reduce cluttering and to gain interactive rendering, only a certain number of randomly masked grid points are selected still sufficient to present the overall behavior of the vector field. In a last step, a pretty small texture which was manually warped and rotated to roughly match the region can be mapped to the terrain. These visualization approaches offer many configuration parameters for a flexible post-processing and are compact enough to be performed with a high frame rate in immersive environments (cf. Fig. 2, left).

3.2 LiDAR Point Data

The resolution of the simulation terrain used is extremely coarse. To see more details of the region like canals, streets, etc., elevation data by U. S. Geological Survey (USGS) is available for the Chenier Plain as well. The original data came from a very large DEM and LiDAR data repository called Atlas, which is hosted by the Louisiana State University (LSU) [15]. Here, one can find panels for almost all regions of Louisiana. Beside the originally measured airborne-based LiDAR data, edited point datasets are also provided. In this case, noise introduced by plants, buildings, animals like birds etc. are eliminated.

To cover the Chenier Plain, 282 LiDAR panels are needed. Each panel has a resolution of 5 x 5 meters and can consist of up to 5.1 million points. All panels together obtain a total number of 820 mio. scattered points. This means that the LiDAR data provides an information density which is approximately 25,000 times higher than that of the simulation grid. The sizes and the different resolutions can be compared in Fig. 3.

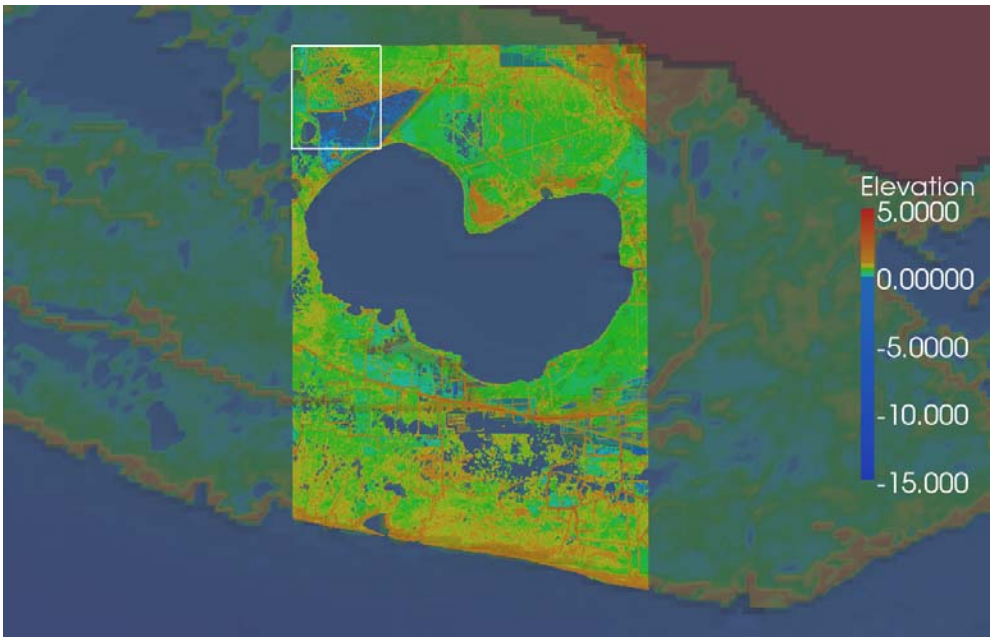

Fig. 3. Very fine details provided by LiDAR data around the White Lake, coarse Chenier Plain simulation terrain (background), and the size of one LiDAR panel (white box).

3.3 High Resolution Chenier Plain Terrain

To replace the bathymetry grid used for the simulation by a considerably higher topography model, the LiDAR data was used. Unfortunately, the Atlas repository does not include enough panels to yield a rectangular terrain. Therefore, the missing panels were taken from a further repository which offers 30 x 30 meters LiDAR data results with less accurate height information. As these were just included to complete boundary terrain information, this was not of importance.

More serious was the problem that the LiDAR measurement does not offer values where water, marsh, etc. is. To fill up those holes, bathymetry data for the northern part of the Gulf of Mexico could be used (cf. Fig. 4, left). The National Geophysical Data Center offers a Gulf bathymetry and coastal relief model with a resolution of 3 arc seconds (approx. 98 mio. scattered points). Using this database [17], not only the upper half seafloor of the Gulf of Mexico between Texas and Florida but also information about the bathymetry of lakes in the vicinity of the shoreline is available. However, depth information is not valid for all lakes. Whereas the Calcasieu Lake for instance shows a useful ground, other lakes as the White Lake (cf. Fig. 4, right) can just be used as contour definitions. Again, this was nevertheless sufficient enough to fill up the holes in the terrain.

A clip filter was applied to remove points outside of the region of interest and to produce an exact rectangular terrain applicable for a straightforward texture mapping later on if wanted. However, the result was far away to be applicable for interactive rendering. A simple random masking of points reduced the complexity to 1 million points. Delaunay triangulation (cf. [4]) eventually produced the terrain as shown at Fig. 1.

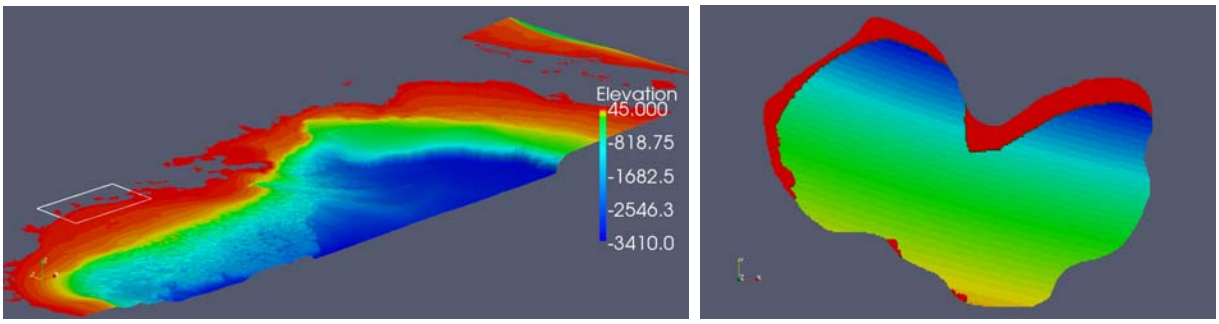

Fig. 4. Bathymetry of the Gulf of Mexico (left), colored according to elevation values (in meters). White box encompasses the Chenier Plain region. Lake bathymetry in coastal region does not offer always accurate depth information (White Lake as example, right).

3.4 Fast Feature-based Rendering

A very fast way to extract first information for the data exploration is the random masking of points. These can then be rendered as OpenGL points. As the size of such point objects is defined in screen space, they always cover the same amount of pixels independent of the distance between the view point and the actual 3D point coordinate. This means that one can see a closed image of the terrain if the amount of points reaches a certain point density threshold (cf. Fig. 5, middle). And it can be rendered with a much higher frame rate as a polygonal mesh (cf. Fig. 5, left) with the same number of points (cf. [11]). But for most cases, the number of points needed for good visual quality is still too high for interactive rendering. On the other hand, the amount can be decreased by increasing the size of the points (in pixels). But then, it may become blurry.

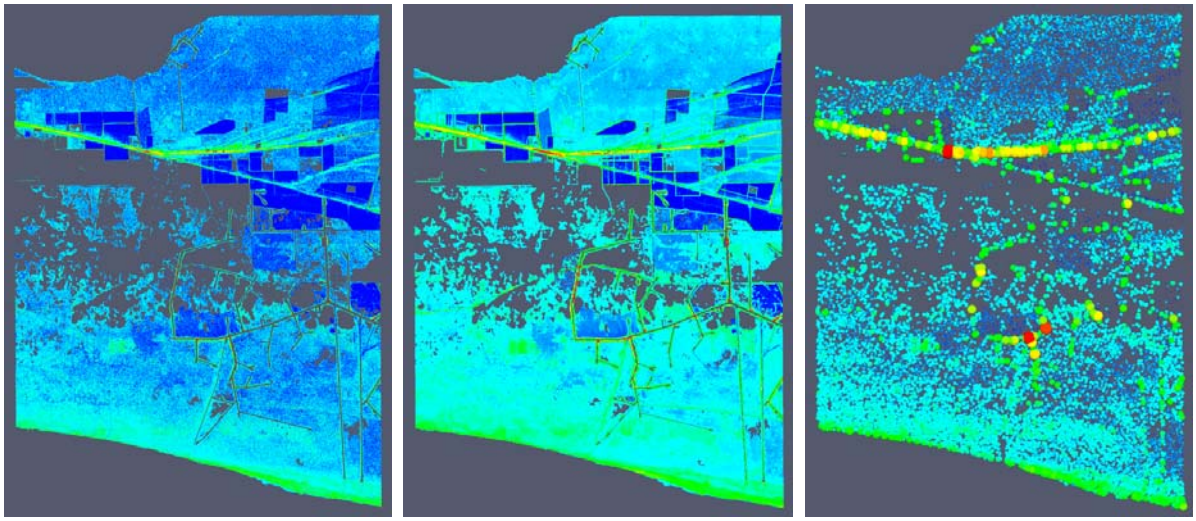

Fig. 5. The Pecan Island region between White Lake and the Gulf of Mexico, Delaunay triangulated and 90% decimated polygonal mesh (left), 12 mio. LiDAR points (mid.), and 2,5k glyphs (right). Elevation determines color and sizes (on the right image only) of the visualization objects.

There is a third way to reduce the amount a points but to emphasize features of the terrain. The height information can be clamped to an arbitrary range to describe elevation that can be taken to draw 3D glyphs instead of 2D OpenGL points. The elevation is not only used to color the glyph but to specify its size as well (cf. Fig. 5, right).

In the presented case, elevation is considered as a feature of the terrain. However, the position of the feature is determined randomly although a certain amount of masked points gives a good overall survey. To overcome this random behavior, a further post-processing step can be applied. The feature edge filter finds important edges of terrains depending on a preset feature angle. Combined with the elevation feature approach, less information can accomplish a very efficient enhancement of the simulation data presentation (cf. Fig. 6). Unfortunately, feature edges can not be detected on point clouds but on meshes so that the time-consuming Delaunay filter has to be employed again.

Nevertheless, the yielded post-processing pipelines contain many parameters for an optimum tailoring to different requirements for interactive data exploration. For instance, masking a certain amount of points randomly before applying the Delaunay filter decreases the accuracy but improves the processing time.

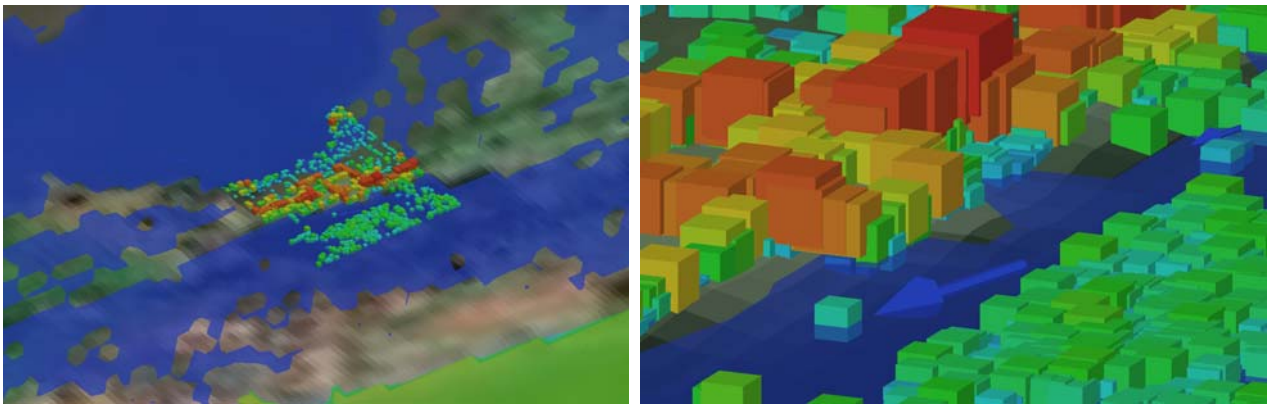

Fig. 6. Pecan Island LiDAR panel features depicted by cube glyphs. Point positions determined by randomly masked points on feature edges. Color and size of the cubes are determined by the computed elevation scalar field. Useful information provided from distant (left) and close (right) view points.

3.5 Mixed Rendering

As shown already on the figures above, the amount of data to visualize at once can be reduced by considering merely one or a few LiDAR panels. This leads to a multi-resolution approach considering the current view point of the user. The panel close to the user (in a first implementation, this is identical with the panel the user stands on) is tessellated and then the complexity is reduced by a mesh decimation step. All other panels are visualized as point clouds after reducing the complexity by the mask point filter. The hypothesis here is that only terrain details of close panels can be perceived by the user. Other panels just offer data to complete the terrain at the horizon. As points are visualized in screen space, the resolution of panels with a higher distance to the user might be additionally reduced. This yields a multi-resolution mixed rendering approach to achieve an adaptive, interactive rendering.

This strategy, however, has some shortcomings which have to be kept in mind. Illumination of terrains emphasizes details. This works only with computed surface normals at each point. But normals like feature edges can only be determined on meshes. In a mixed rendering mode, tessellated panels with normals rendered beside point cloud panels without normals show a substantial visual discontinuity. Therefore, for a high visual quality, all panels must be converted to meshes first before the mixed rendering approach incorporating normals can be employed (cf. Fig. 7). One looses the advantage of presenting surrounding point-based panels very quickly. On the other hand, normals can be computed offline and stored with the point clouds so that they are immediately available as soon as they are required.

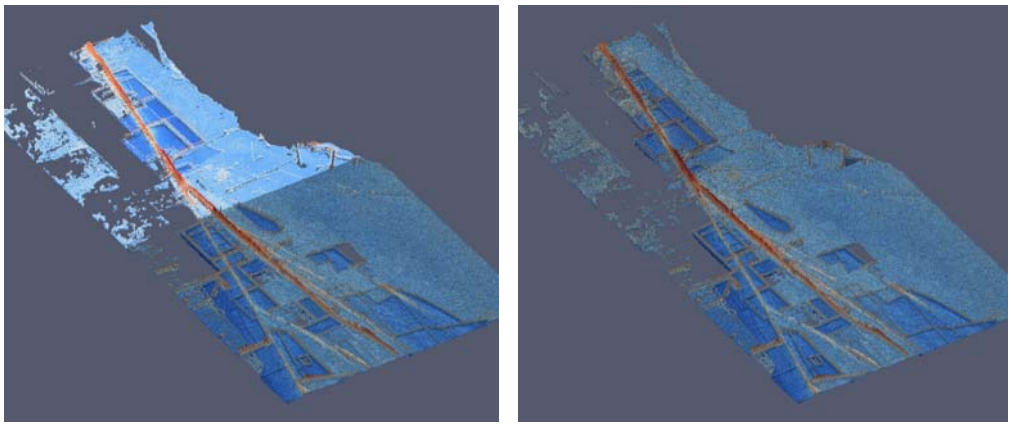

Fig. 7. Two LiDAR panels (decimated to 25% complexity) around Pecan Island (upper panel: point-based, lower panel: Delaunay triangulated). Surface normals only computed for the triangulated panel (left), and computed for the point-rendered and the triangulated panel (right).

3.6 Texturing

For the simulation bathymetry grid, a very simple satellite image was used to map some more landscape context information on the terrain. In this case, an image was captured from Google Maps, which was not aligned to the grid at all. But using an image processing tool with a bathymetry contour image as background, this captured Internet image could be roughly registered by warping and rotating. Finally, it was cropped to the simulation grid which makes texture mapping very easy as texture coordinates are identical to the x-y-coordinates of the normalized (values between 0.0 and 1.0) terrain bounding box. In this case, the result was very satisfactory because of the low resolution of the bathymetry grid (cf. Fig. 2).

To receive a more realistic terrain not only for the evaluation of the simulation result but for further aspects of coastal restoration, the high resolution terrain based on the LiDAR data (cf. Sec. 3.3) was used. The simple Internet image, however, is not sufficient enough anymore to correlate with all presented features in the terrain. As an alternative, LANDSAT7 satellite images are available on the National Map Seamless Server by USGS [18]. These are already registered and can be used directly without any image processing. The only pre-processing step needed is cropping simulation data and the satellite image to the terrain grid boundaries. Compared to the Google Maps image of the Chenier Plain region with a resolution of 975 x 282 pixels, the LANDSAT7-based texture shows a much higher resolution: 7245 x 3474 pixels (cf. Fig. 8). A low-resolution version of this texture was produced for virtual environment with less powerful graphics hardware.

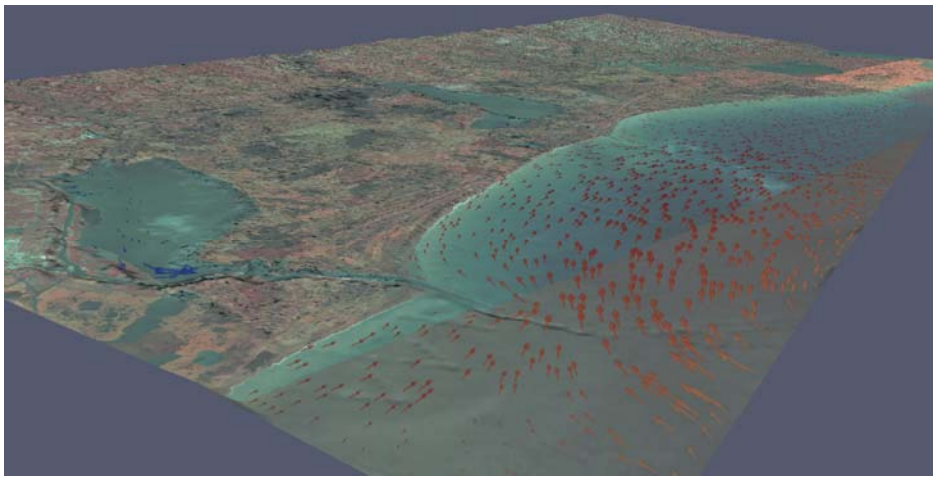

Fig. 8. Orthoimagery provided by LANDSAT7 mapped on the LiDAR-based terrain that was reduced to a 1 million point grid and scaled in z-direction. Additionally, vector glyphs of the simulation dataset are visualized.

For the panel-based multi-resolution terrain, an adaptable approach for texture mapping would also be preferable. From the same source where the LiDAR panels are from, separate satellite images for each panel are accessible [16]. Beside other formats, these digital orthophoto quarter quadrangle (DOQQ) images are also available in JPEG 2000 format with a resolution of around 6700 x 7580 pixels each. Each pixel of such a satellite tile represents 1 square meter. This means that the texture has approximately a 10 times higher resolution than the corresponding LiDAR panel. In contrast to LiDAR data, information is also available where water is measured. Furthermore, the textures are slightly larger than the LiDAR panel bounding boxes (in x- and y-direction). Therefore, texture coordinates cannot be determined straightforwardly. Fortunately, the satellite texture comes with information at which NED coordinate the upper left corner is located. In conjunction with texture size and panel bounding box, the texture coordinates can easily be computed for each LiDAR point (cf. Fig. 9).

4. VR-BASED APPLICATION

The presented coastal restoration scenarios comprise different scales, complexities, and exploration requirements. The VR application has to consider all presented scenarios. Moreover, the goal was to develop a framework hiding post-processing implementation details and the usage of VR technologies in order to offer a straightforward application programming interface. On the other hand, it should be open enough to be suitable for a large range of engineering problems. In this section, the developed framework used for the coastal restoration project is depicted.

Fig. 9. Correct texture mapping on two panels of Pecan Island for high resolution rendering (left: point rendering with normals; right: polygonal rendering with normals).

4.1 The Framework

The main aspects of the developed VR-based visualization framework are covered by toolkits freely available as public domain or open source projects. Almost all functionalities of the data processing are offered by the Visualization Toolkit (VTK). It is based on a pipe and filter concept that uses the output of one processing step as input for the successive filter. The end of such a pipeline yields a set of polygons and texture information needed for the rendering process (cf. Fig. 10 as an example). [13]

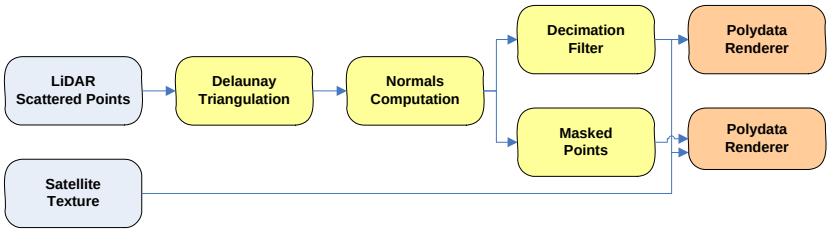

Fig. 10. Time-consuming but highly accurate VTK pipeline for interactive terrain visualization. Depending on the view point, the pipeline may switch to a polygon-based (upper branch) or a point-based rendering (lower branch).

Although VTK also offers rendering capabilities, this task is taken over by ViSTA FlowLib [12]. The main goal of this toolkit is the time-correct visualization of unsteady datasets. For this purpose, it creates data containers as simple arrays for each dataset used in an application. In general, a simulation time step of such an unsteady dataset is considered as input of a post-processing pipeline. The processed result is then stored in a bin of the data array. Furthermore, ViSTA FlowLib subdivides the visualization time for one simulation loop into discrete time levels (in our case: the year 2003 into 365 days). The application is responsible to map time levels to the corresponding bins of each data array. Thus, the current visualization time determines the current time level which eventually points to all bins used for the current render frame. This mapping allows a high flexibility in order to combine several datasets for a synchronized visualization. Also steady and unsteady datasets may be overlapped as needed in the coastal restoration project (cf. Fig. 11).

Actually, ViSTA FlowLib does not care about the post-processing. It only expects that the data arrays are filled with data which are defined in a format supported by the selected renderer. ViSTA FlowLib offers a set of renderers for various visualization approaches. In our case, only the VTK polydata renderer is used. Therefore, all post-processing pipelines have to produce data in VTK polydata format.

In general, the application is responsible to assign data to the array bins. On the other hand, the application can also decide that the data is produced by an external post-processing application. Here, Viracocha comes into play [3]. It is a parallelization framework with a communication protocol which allows receiving requests from ViSTA FlowLib to compute data and sending results back for rendering. Running on a supercomputer, Viracocha can also carry out the requests in parallel. The application configures the requests and determines the kind how data is transmitted, i.e., whether partial results by individual Viracocha processes are send in parallel to ViSTA FlowLib or they are bundled first to one large packet. But then, fortunately, communication aspects and data handling are automatically managed by the base

toolkits. The application developer has not to control the data flow. As soon as results arrive at the visualization frontend, they are rendered automatically.

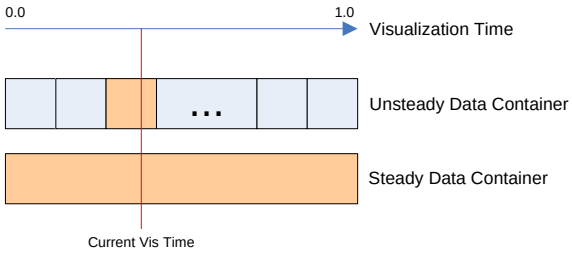

Fig. 11. The current visualization time (red line) determines the current bins (orange) of registered data containers which are used for the current rendering frame. The example shows two data containers, one for unsteady data and one for steady data, respectively.

The toolkit Viracocha itself only coordinates requests, the creation of worker groups responsible to compute data in parallel, and the communication between visualization frontend and computation backend. It does not define the type of data and how to compute the data, i.e., it does not specify the post-processing pipeline. As a library, it just defines the work flow and therefore may be used for arbitrary computation domains. Domain-specific algorithms have to be implemented as an executable application on top of it. For the post-processing of flow simulations, an application called Hobbes has been developed at Aachen University. This executable may be enhanced by own communication and computation approaches.

Once received at the frontend, VistaFlowLib renders the produced visualization objects immediately. As they are three-dimensional, the view point can be changed to obtain a different perspective of the scene. The view point determination and perspective definition is performed by VR Juggler [2]. In general, VistaFlowLib is based on ViSTA, a further VR toolkit, but this does not support some techniques like tracking servers and multi-pipe visualization systems. Therefore, VR Juggler was chosen to organize display configuration, tracking devices, and – very important – the application event loop.

4.2 VRFlowVis

The described framework is very powerful but also complicated to handle. But the goal was to have a framework at hand for the engineering domain, e.g., developers should concentrate their efforts on implementing discipline specific algorithms. VR configuration, process distribution and parallelization, data flow, rendering aspects, etc. should be hidden by user-friendly interfaces. Therefore, we have been developing a further toolkit called VRFlowVis as a layer of all these other libraries to organize and arrange various components of the overall framework which yielded an application programming interface for fast scientific application development. The overall framework with dependencies among each other is depicted in Fig. 12.

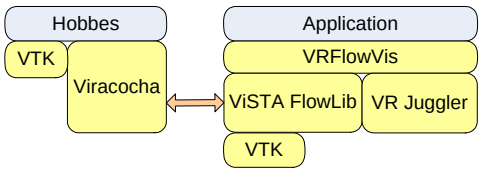

Fig. 12. Toolkit dependencies of the used distributed post-processing framework (yellow: libraries; blue: executables).

As a first step, the setup of post-processing algorithms was simplified by offering initialization files. The application just has to point to such an exploration initialization file where post-processing algorithms, the result data type, time information, parameters for the backend, rendering properties, etc. are specified. Here, one can also distinguish between a remote extraction mode and a local loader mode. The latter source mode loads already computed data on the frontend and stores it directly in the data containers. Based on the usage of initialization files, it is now possible to create an efficient application with only one main function containing approximately 5 lines of code to start the whole system. The remaining tasks are managed by VRFlowVis.

Especially for smaller post-processing tasks and if one does not want to enhance Hobbes, a third mode allows the local definition of VTK pipelines on the application layer. As one goal of the distributed system is to relieve the visualization

frontend from intensive I/O, time-consuming computations, and post-processing with large memory footprint, local operations should absolutely be restricted to smaller or to start-up operations.

The processed result can be added to a so-called visualization component (VisComp). A VisComp consists of a source, a data, and a sink. The source is an object managing how data is generated depending on the selected source mode. The data is the data container with a data type that fits to the source output. Finally, the sink is generally a renderer which is able to visualize the stored data type. The VisComp is the central object of VRFlowVis. The user interface allows switching through defined VisComps typically responsible for one extraction task. In the coastal restoration, the bathymetry, the salinity scalar surface, the velocity glyphs, and the LiDAR panels are accessible separately by such VisComps. The user can select a certain VisComp to start the extraction, to load the terrain, to toggle the visibility, etc.

To speed up the rendering, the renderers generally work with display list. These are compiled OpenGL directives which are submitted once to the graphics processing unit (GPU) and are used then directly from the graphics card memory for every render frame. Just after the application has modified properties, a new display list must be assembled and submitted. This is the same with textures. Once submitted, they are used directly on the graphics cards. If only one workstation with one graphics card (one graphics pipe) is used, nothing special has to be considered. As the concept of ViSTA FlowLib for the cluster of visualization workstations follows the replication of the whole application and just sharing input events, each node of such a cluster works like an isolated workstation with one render pipe. The disadvantage here is that data must be replicated as well. This can become a problem when using a distributed post-processing system. But fortunately, this is handled automatically by ViSTA FlowLib. Data received by the master is directly forwarded to the clients as well (cf. Fig. 13, right).

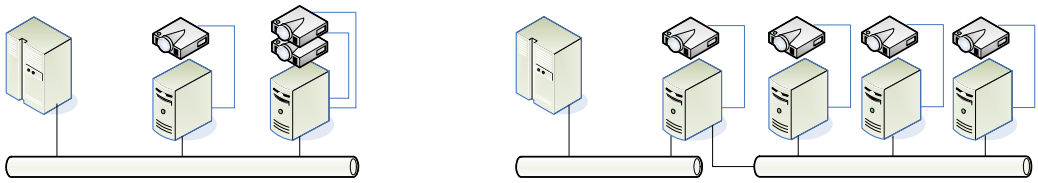

Fig. 13. Remotely extracted data and internally managed render data has to be handled differently in applications for single workstations with one GPU and with a multi-pipe system (left), and for visualization clusters (right).

However, if a workstation has multiple render pipes to drive multi-display systems needed for larger immersive environments (cf. Fig. 13, left), ViSTA FlowLib does not offer mechanisms for that anymore. But VR-Juggler does. In its application loop, the application can prepare visualization objects and evaluate VR devices which are processed sequentially before all render pipes are performed in parallel as multi-threads. In such an application loop, only one shared data container for all pipes would yield two problems: Display lists and texture images would only be submitted to the graphics card that belongs to the pipe which has access to that data first. The second problem is that in a parallel environment shared data must be locked for exclusive access. To overcome the depicted problems, VRFlowVis registers one ViSTA FlowLib instance for each render pipe so that the local data for each pipe is independent and can be rendered without locking the access. This maximizes the render frame rate, which is always the main goal for real-time systems like VR environments. Display lists and textures are handled correctly yet as well.

To hide also the multi-pipe behavior from the application developer, the VisComp does not fill the data containers for each render pipe directly but is designed as proxy objects. Whenever the application adds, modifies, or deletes visualization components, VRFlowVis forwards the changes internally to the pipe-assigned visualization objects. Furthermore, a VisComp just stores renderer properties which are then passed to all renderer instances assigned to the pipes. The source objects manage their produced data themselves and replicate them for multi-pipe data containers. Thus, the system can be used for all common visualization systems on market without complicating the programming interface for the domain-specific application development.

5. RESULTS

5.1 Hardware

The preferred system for the coastal restoration project is a 6-sided CAVE-like display system driven by an HP SVA (Scalable Visualization Array) V.1.1.1 architecture. This visualization cluster consists of 6 nodes with Fast Ethernet and Infiniband interconnects. Each node is equipped with two Opteron 246 processors (2 GHz), 2 GByte main memory, and

an NVidia Quadro FX 4500 graphics card. Further VR systems available are based on multi-channel SGI Prism systems with 14 graphics pipes in total. These Itanium-2 based visualization computers have up to 16 processors and 16 GByte shared main memory. Finally, a Dell visualization workstation with an NVidia Quadro FX 5600 graphics card, 8 Intel E5440 Xeon cores (2.8 GHz), and 32 GByte main memory was used to evaluate the runtime on a single-pipe desktop system. The measurements which are presented in the following paragraphs are performed on this workstation (Suse Linux v10.2, 64 bit) in an 800 x 800 pixels mono render window.

In virtual environments, a render frame rate down to 30 frames per second (fps) is still sufficient enough. But as soon as it goes below 10 fps, interactive work is not possible anymore [3]. When rendering the Google Maps texture mapped on the simulation bathymetry together with the salinity scalar surface and 4000 randomly masked velocity glyphs, a frame rate of 210 fps could be measured. Without simulation data, 340 fps could be yielded. As soon as 28,687 randomly masked LiDAR points are additionally visualized as 3D cube glyphs, the frame rate goes down to 150 fps. Masking 397,193 glyphs allows a frame rate of 42 fps, which is still sufficient for virtual environments. The complexity can be increased even more when using 2D OpenGL points instead of polygonal glyphs. With the same amount of points, the frame rate goes up again to 198 fps. 1,584,035 sampled OpenGL points shows 41.5 fps.

Replacing the simulation terrain and the Google Maps texture by the 1 mio. LiDAR-based terrain with the lower resolution LANDSTA7 texture, the frame rate drops to the lower frame rate threshold of 10.15 fps. This is not the fault of the texture. The original bathymetry with the high resolution LANDSAT7 texture rendered together with simulation data still reaches 220 fps on this high-end visualization workstation. This proves that mesh rendering is still a challenge for current GPUs. On the other hand, this alos shows the importance of textures as these can replace missing information lost by coarse meshes. Combined with mixed rendering (point-based rendering mixed with polygonal mesh rendering), a higher mesh and texture resolution can be used close to the user and points elsewhere without loosing interactivity.

The memory footprint on the visualization frontend also plays an important role. The bathymetry with the low resolution Google Maps texture and the simulation data consume 4.5 GByte main memory. Creating and submitting display lists for all 365 time steps allocates additional 5.9 GByte. This proves the importance of large shared memory on cluster-based as well as on multi-pipe VR-systems. The 8.9 GByte LiDAR data was processed sequentially and intermediate data as well as the original raw panel dataset were removed as soon as one panel was processed. Thus, 397.193 LiDAR cube glyphs do not increase the total memory footprint substantially anymore.

The last aspect considered in this section is processing runtime. Loading and processing the simulation data is not really an issue (14.4 seconds for salinity and 30.6 seconds for 4000 velocity glyphs). Loading e.g. 23 panels east of White Lake and masking 286,461 points to map glyphs on them also needs just 4.8 seconds. This is not real-time but fast enough to be carried out at start up. But as soon as triangulation is part of the post-processing, the computation time increases extremely. The 2D Delaunay filter needs 91 seconds for the right panel (3.76 mio. points) and 41 seconds on the left panel depicted on Fig. 9. When tessellating 23 panels east of the White Lake, extracting feature edges, and finally visualizing 446,994 masked points, one has to wait 26:45 minutes. Therefore, processing all 282 panels sequentially requires some hours before one can start the evaluation in interactive environments.

6. CONCLUSION AND FUTURE WORK

This paper has presented the ongoing work how to visualize efficiently large geophysical simulation and field data in virtual environments. This work is based on post-processing algorithms offered by VTK. One shortcoming especially of glyph-based strategies is that they produce a bunch of polygons for one data position which eventually reduces considerably the complexity of a scene that can rendered interactively. ViSTA FlowLib, however, offers already GPU-based renderers in order to visualize a huge amount of particles in real-time. Make use of these approaches could help to show many more terrain details in the coastal restoration project. In general, improved data formats and contemporary algorithms not provided yet by VTK could increase the complexity even more.

The distributed post-processing framework helps to relieve the visualization system of time-consuming tasks. Especially the feature-based approach, which is a very successful method to extract important structures stored in point clouds, is very time intensive. Data parallelism on the backend has the capability to reduce the processing time extremely. Mixed rendering (with precomputed normals) as a first multi-resolution approach can improve the runtime even more. With the option to keep a VTK pipeline open at the backend to compute updates, a permanent data streaming triggered by the current view point can be integrated. When the user moves over the terrain and crosses a border to a new LiDAR panel, the position is submitted to the backend which computes the mesh for the closest panel and delivers just a set of points

for surrounding panels depending on the distance to the view point. First data streaming approaches have already been implemented and evaluated [9].

7. ACKNOWLEDGEMENTS

The authors wish to thank the Virtual Reality Group, RWTH Aachen University, Germany who kindly made available the VistaFlowLib and Viracocha toolkits and supported the integration effort into the coastal restoration project application.

REFERENCES

[1] Bryson, S. and Gerald-Yamasaki, M. J., "The Distributed Virtual Windtunnel", Proceedings, Supercomputing, Minneapolis, Minnesota, USA, 275 – 284, (1992).

[2] Cruz-Neira, C., Bierbaum, A., Hartling, P., Just, C. and Meinert, K., "VR Juggler - An Open Source Platform for Virtual Reality Applications", Proceedings, 40th AIAA Aerospace Sciences Meeting and Exhibit 2002, Reno, NV, USA, (2002).

[3] Gerndt, A., Hentschel, B., Wolter, M., Kuhlen, T. and Bischof, C, "VIRACOCHA: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments", Proceedings, Supercomputing 2004, Pittsburgh, PA, USA, (2004).

[4] Isenburg, M, Liu, Y., Shewchuk, J., and Snoeyink, J., "Streaming Computation of Delaunay Triangulations", Proceedings, ACM SIGGRAPH, 1049 – 1056, (2006).

[5] Kreylos, O., Bawden, G., Bernardin, T., Billen, M. I., Cowgill, E. S. and Gold, R. D., "Enabling Scientific Workflows in Virtual Reality", Proceedings, VRCIA, Hong Kong, June 14 – 17, (2006).

[6] Kuhlen, T., Gerndt, A., Assenmacher, I., Hentschel, B., Schirski, M., Wolter, M. and Bischof, C., "Analysis of Flow Phenomena in Virtual Environments – Benefits, Challenges and Solutions", Proceedings, 11. International Conference on Human Computer Interaction, HCII 2005, Las Vegas, NV, USA, (2005).

[7] Jimenez, W. H., Correa, W. T., Silva, C. T. and Baptista, A. M., "Visualizing Spatial and Temporal Variability in Coastal Observatories", Proceedings, IEEE Visualization, 569 – 574, (2003).

[8] Miller, R. L., Arceneaux, J. C. and Meselhe, E. A., "Louisiana Chenier Plain Regional Hydrodynamic, Salinity and Hydrologic Numerical Models", 10th International Workshop on Wave Hindcasting and Forecasting and Coastal Hazard Symposium, North Shore, Oahu, Hawaii, Nov. 11 – 16, (2007).

[9] Quicker, A., "Out-of-Core Remote Data Streaming of Large-Scale LiDAR Datasets for Interactive Post-Processing", Diploma Thesis, Institute of Imaging and Computer Vision, RWTH Aachen University, (2008).

[10] Reed, D. J. and Wilson, L., "Coast 2050: A new approach to restoration of Louisiana's coastal wetlands". Physical Geography 25: 4-21, (2004).

[11] Sainz, M., Pajarola, R., "Point-based Rendering Techniques", Computer & Graphics, 28: 869-879, Elsevier Ltd., (2004).

[12] Schirski, M., Gerndt, A., van Reimersdahl, T., Kuhlen, T., Adomeit, P., Lang, O., Pischinger, S. and Bischof, C., "ViSTA FlowLib - A Framework for Interactive Visualization and Exploration of Unsteady Flows in Virtual Environments", Proceedings, 7th International Immersive Projection Technologies Workshop, and 9th Eurographics Workshop on Virtual Environments, ACM Siggraph, Zurich, Switzerland, 77 – 85, (2003).

[13] Schroeder, W., Martin, K. and Lorensen, B., "The Visualization Toolkit – An Object-Oriented Approach to 3D Graphics", 4th edition, Kitware Inc., (2006).

[14] Yoon, S., Salomon, B., Gayle, R. and Manocha, D., "Quick-VDR: Interactive View-Dependent Rendering of Massive Models", Proceedings, IEEE Visualization, 131 – 138, (2004).

[15] Atlas: The Louisiana Statewide GIS, Elevation Data Repository, Louisiana State University, http://atlas.lsu.edu/lidar/, accessed: 2008-12-07.

[16] Atlas: The Louisiana Statewide GIS, Digital Orthophoto Quarter Quadrangle Repository, http://atlas.lsu.edu/doqq2004/, accessed: 2008-12-07.

[17] National Geophysical Data Center, US Coastal Relief and Bathymetry (GEODAS, NGDC, NOAA), http://www.ngdc.noaa.gov/mgg/gdas/gd_designagrid.html, accessed: 2008-12-07.

[18] The National Map Seamless Server, U. S. Geographical Survey, http://seamless.usgs.gov/website/seamless/viewer.htm, accessed: 2008-12-07.