Volumetric Rendering: Black Sea Shipwrecks

6.837 Introduction to Computer Graphics, Fall 1999

Bill Wallis (*bwallis@mit.edu*) **Michael Dewberry (*dewb@mit.edu*)**

Abstract: During the summer of 1999, Professor David Mindell of the MIT Science and Technology Department, used global positioning system (GPS) transponders and a combination of narrow and side beam sonar technologies to produce sub-bottom and bathymetry data sets for a vessel that sank in the Black Sea over 1,000 years ago. The goal of this project is to align the positional navigation system (PNS), depth, and density data into a unified data set and then to develop software to view and analyze the shipwreck's artifactual evidence in 2- and 3-Dimensions. The visualization aspect of this effort uses Silicon Graphics' (SGI) Volumizer API to implement a voxel-based sampling and polygonization technique in order to overcome limitations inherent to traditional surface rendering algorithms. The analysis part of the project provides a framework for applying density heuristics and filters in order to help isolate 'characteristic densities' and overcome structural ambiguities derived from sources such as natural decomposition, potential settling and structural damage, and the latitudinally sparse nature of the data set.

I. Overview

i. Sonar Technology and the Data Collection Process

Sonar data was collected using analog sensors attached to a robotic submersible named JASON over the wreck site within a rectangular area of about 40 by 30 meters. The arc length of the narrow beam's sampling field was approximately 2 degrees and so each pass or 'track line' over the wreck site represents a density strip of on the order of several centimeters wide.Density data was sampled at 3000 Hz in the vertical direction, using a frequency of 150 Khz for approximately 6-10 meters of meaningful subbottom penetration. Because of the financially expensive nature of the collection process, the track lines were sampling at a few meters apart and so the coverage of density data is quite sparse in directions orthogonal to JASON's path of motion. The start time for each vertical sample period is timestamped and, since the data was collected by directly sampling the sonar signal, the relationship between data points is similarly in terms of time. Multi-path distortions and sonar echo potentially exist, though the employed analysis does not take these effects into consideration.

The side beam sonar was sampled at the same rate and was reconstructed into a bathymetry mesh at the Woods Hole Oceanographic Institute using a sophistaced digital filtering and pre-processing. Because of the large arc length of the sampling array and the resulting overlap between subsequent passes, this data set represents a complete topological map of the sea floor. Unlike the narrow beam sonar data, however, this data has already been processed to convert the data of the sonar wavefront from the time to the space domain and so this data set is specified in timestamped, relative x,y, and z UTM coordinates. Because of the large arc length empolyed in wide beam sonar, corrections were also necessary to account for the spatial differentials that arise from wavefront propagation.

Concurrent with the above processes, a triangulated GPS tracking system with two transponders of known position, collected information about JASON's position, again timestamping each set of sampled data. Despite that the timestamps for each of the data sets is indepdent,the three data sets can be synchronized using a straight-forward nearest-neighbor approach in order to relate - and hence align - each data set in time. Throughout this paper, reference will be made to relative UTM coordinates rather than absolute coordinates in order to conceal the location of the submersed vessel.

ii. Fundamentals of Applied Volumetric Rendering

In general, volumetric rendering is a more general case than surface-based rendering in that makes fewer a priori assumptions about transparency and surface boundaries. For many applications, it is quite reasonable to assume that objects within a scene are opaque and that some mediums, such as air, are ideally transparent. In some cases, however, the object of interest is the primarily semi-transparent object and it may even lack a distinct surface boundaries. In these cases, procedures such as surface determination and computing normal vectors can become either unfeasible or computationally prohibitive. In these cases, it then seems best to focus ones computational attention on the transmittive rather than the predominately reflective aspects of the sampling procedure and this is precisely what volumetric rendering does.

Although the sampling path may be arbitarily defined, volumetric rendering techniques take series of 3-dimensional samples of the volume's interior. While volumetric ray casting techniques typically takes these samples along pathes orthogonal to the viewport, SGI volumizer takes samples parallel to the viewport in order to be able to leverage the target machine's texture mapping hardware for increased performance.With Volumizer, voxel data may be imported into the application using the IFL image library, where a volume may be described by a 3-D image file, such as a TIFF image, or by a series of 2-dimensional image plates.Since a planar image is essentially the degenerate case of a volume, both types are supported by Volumizer.

After importing the voxels, data is subdivided into 2-dimensional bricks, the size of which is determined by the amount of supported texture memory.This permits discrete sections of the volume to be loaded into texture memory, processed sequentially, and then super-imposed in order to render the entire geometry.Because loading textures in memory is an expensive process, their size is chosen by a just-fit parameter in order to make optimize use of the texture memory.If the volume size is such that a brick size is smaller than the texture memory, multiple textures may be loaded into memory and the OpenGL bind feature may be used to avoid unecessary loading and to facilitate texture reuse.

Once subdivided into bricks and aggegratedinto brick collections, sampling occurs starting with the near plane of the view frustum and terminating at the far plane. After each plane is sampled, the distance from viewport is incremented by distance determined by the sample rate and the volume is ultimately described by a series of these planar texture or volume slices.These textures, however, represent only the appearance of the volume and then must be mapped onto defined geometries in a process called polygonization, in which associated texture maps with semi-transparent geometries. This volume slicing technique is better than a ray casting approach, since the computations are performed by the underlying hardware, though the end result is the same: the volume is reduced to a set of semi-transparent polygons that can then be merged with any other polygon-based graphics system, such as OpenGL or SGI Optimizer or Performer.

iii. Description of the Analysis Framework

The analysis framework essentially acts as a processing intermediary in that it contains the data structures and routines required to import the synchronized data in a potentially piece-wise fashion and then render it into a Volumizer-compatible 3-dimensional TIFF file.The power behind the analysis framework is that it allows the user to apply customizable filters to the images and data prior to the rendering process, setting the stage for flexible annotation and visual analysis of the data. In addition, it allows a user to apply filters to small subsections of the track lines and view the results in 2-dimensions so that filter experimentation is both feasible and efficient.The analysis framework has been designed in this way, since the actual data of interest - the structural artifacts of the shipwreck - constitute relatively small ’signals’ within data set and so significant pre-processing will be necessary to achieve any kind of tangible results.

As a first take, the analysis engine supports some basic filter primitives, such as the ability to compute image differentials. Such an approach is motivated by the assumption that some areas of the site will contain only ’noise’ signals in terms of densities and so a differential computation may allow the user to subtract out the unwanted and dominate noise signals. Here, noise does not necessary refer to factors such as sonar echo, but rather to densities that make up the typical composition of the bottom and can be interpreted as silt or other naturally occuring materials. Some slightly more sophisticated techniques such as differential histograms have been implemented, but better and more precise filters are left for the user to develop and implement.

II. Project Goals

i. Aligning the PNS and Sonar Data

The original data files, the PNS files that contain positional information and the LOG files that contain density data, must be merged by finding timestamps in each file that are closest in time.This matching provides the first values of the density data with an x,y, and z coordinates. Since the density data is in terms of time, each the relative z-offset of each density value must be computed in order to put all axial orientations into the space domain for visualization and integration with the bathymetry mesh data.For the purpose of this project, a first order approximation of Newtonian round trips times is used to convert each time sample into a space sample. Because the density of the subbottom causes the speed of the sonar wave to change velocity, a second order approximation would entail performing a discretized intergral along the z-direction in order to provide more precise depth offsets.Once this conversion has been made, it is relatively straight-forward to integrate the models into a singe volume space and the process primarily involves data manipulation, conversion, and renormalization.n order to increase the efficiency and to account for sonar attenuation, the density values are also truncated at a certain depth, since the remainder of the data represents only sonar echo and other extraneous information that does not consitute minimal round trip times. The information necessary to construct the model is then contained in a single synchronized file that has a file size of approximately half that of the original PNS and LOG files.

ii. Color Modeling and Density Interpretation

The second stage of the pre-processing steps involved choosing a color model and interpreting the density values as RGB values, since cathode ray technologies ultimately require using this color model. In order to represent a full range of color values, the data is initially normalized and mapped to 300 degrees of the hue spectrum on the HSV color cone, since human perception is best at distinguishing differences in wavelength intensities and using pure colors maximizes this effect. From an analytic point of view, the final image product should also be optimized to display density differentials, since they potentially correlate to object boundaries. In addition to varying the hue, the color conversion process also performs gamma correction by changing the luminosity of each color value based on its density using a quadra-linear interpretation scheme. Since low densities are mapped to the blue end of the spectrum and high densities to the red end of the spectrum, this arrangement was intended to de-emphasize aqueous materials, but also to implicitly highlight mixed-composition materials that are adjacent in space. An amphora, for instance, could potentially exists as a high-density exterior shell with an aqeuous interior and a less pronounced external density gradient at the shell surface. The above scheme would make the interior appear much darker, but by virtue of being non-linear also emphasize the exterior gradient.

iii. Track Data Structure Basics

The Track class is intended to provide a flexible means for manipulating and analyzing either single or multiple passes over the wreck site. It is designed to allow the user to specify an offset into a synchronized file and a length of the number of sample points to read. Since the amount of data is often quite large, this allows the user to segment the loading process or to concentration on small areas of the data in isolation. Following a data read the user may then either render a two dimensional image or perform functions on the raw density values prior to the rendering. Here, the rendering primitive - an image - is closely coupled with the data structure, because the analysis is primarily visual, the image is highly dependent of JASON's current position, and the memory cost prior to rendering is simply an additional unintialized object pointer. The track data structure, however, is otherwise isolated from the graphical interface and so can be used either in conjunction with the supplied TrackViewer class application for 2-dimensional viewing or with the VoxelDraw class for 3-dimensional viewing. Future versions should also support a binary loader for speed considerations, since the StringTokenizer class is not particularly efficient in parser an ascii file. Since many of the density values are small denomination integers, however, the use of binary files may not actually provide much in terms of a file size reduction, unless the data items are narrowed to short integers. Additionally, Perl and Java make different assumptions with respect to binary layout for their type primitives, which will prove problematic in reading such values.

iv. 2-Dimensional Viewing and Analysis

The TrackViewer class allows the user to import a Track object from a synchronized data file, but it also allows the entire state of a Track object, including its associated image, to be stored and retrieved as a serialized object. The user can, therefore, save this object and take advantage of a binary loader by saving the object after its first use. Though this is not space efficient, it is processor efficient, since the time required to read an image is less than it is to calculate an image from the raw density values, paricularly if one or more image filters must be applied to the image to achieve the same image state.

TrackViewer also allows multiple tracks segments to be loaded simultaneously and for the user to switch back and forth between them using subclasses of the ActiveList class. By selecting a op from the ops menu, a user may directly apply image filters to pairs of tracks - such as to perform a differential histogram operation. If the track segments are of different sizes, then operation occurs over the domain defined by the width of the smallest track. During on operation, the first track argument is cloned, the operation is applied, and the resulting track appears as a new track object in the TrackList. Since the rendering time can become somewhat significant for drawing large images to the screen, only one track sample can be present on the active track canvas. If the user wishes to view two tracks at the same time, the user must run two instances of the application. Further use of the application should be straight-forward, as the application is packaged into a simple menu driven user interface.

v. Rendering Arbitrarily-defined Volumetric Subspaces

Using the VoxelDraw class, the user can write very simple code to render track data into a 3-dimensional TIFF image file. The ability to render objects, however is not limited track data, but is extensible to any object that is defined in real coordinates and has a color or derived-color value associated with each data point. The process essentially takens the real data coordinates, the dimensions of the TIFF image 'canvas', normalizes the data, and scales it to fit within the volume space. In this case, scaling occurs along the longests axial dimension for which data is present in order to maximize use of the canvas, but to also maintain proportionate scaling. Multiple objects can then be rendering to the same volume, using multiple passes of the draw procedure. Since data can be read as well as written various schemes, such as color averaging, could be employed into order to incoporate overlapping volumes.

In many cases, however, objects may constitute only small portions of the volume space. From an efficiency point of this is quite inefficient to render and so the reader may question the use of a single image file over the use of smaller individual image files. SGI Volumizer, however, supports a method called 'space leaping', which allows a volume to be subdivided and then tessellated in smaller separate volume units, thereby eliminating many extraneous pixel fill calculations.

vi. 3-Dimensional Volumetric Modeling and Extensibility

Once the model has been drawn to the voxel canvas, it can be loaded into a Volumizer application via an IFL brick loader procedure and rendered using the previously outlined volumetric procedure. For cube sizes of 64x64x64, the model should be fully interactively on an SGI O2; larger volume sizes, such as 256x256x64, however, saturate the texture memory and tend to have substantial performance lag due to repeated brick loading. Since the result of the Volumizer process produces a set of semi-transparent polygons, the model may be incoprated into a heteogeneous graphics scheme with other sufrace rendered objects, using such API's as Performa, Open Inventor, or OpenGL.

III. Project Accomplishments

i. Alignment Scripts in PERL

The alignment script, synch.pl, can batch PNS and LOG files using conventional file pattern matching as long as the file patterns implicitly sort to correct pairings. If the number of file pairings is greater than one, the user is prompted for confirmation in order make sure that the files are paired correctly. Performance monitoring has been integrating into the script, using the timer.pl utilities, to show CPU time and other statistics for each file pair that is processed, as well as a simple progress indicator that advances every 25 records. The resulting file that contains the synchronized data is placed in a file by with the same base filename as the input log file, but with the extension .syn. For the tracklines in the sample data, containing on average 2000 lines, this process takes under 10 minutes for all 6 pairs of data files. The resulting file is approximately one-half the size of the original file and may be further condensed by encoding the density values as shorts and switching to binary encoding.

ii. 2-Dimensional Visualization and Analysis Toolkit in Java

The original imaging code was written in perl,in a script called image.pl, though it was ported to Java in order to be integrating in the analysis toolkit.Since java code is typically compiled at runtime using JIT compilers, the performance of this code was also enhanced.The current state of the toolkit offers a flexible means of importing track data and for applying analytic filters to the raw data in a generic, serialized fashion.The intention here, however, is to provide a framework for analysis so that more advanced methods may be integrated in the framework with the ultimate goal of provide full scale object detection. The project leaves this exercise for the more sonarly inclined.

iii. 3-Dimensional Rendering using Volumizer in C++

The provided system allows users to easily import new track data and visualize it using Volumizer. Loaders have been provided for any synch file that was generated using the perl script mentioned above and the user needed only specify the bounding parameters for the input track. In addition, it supplies a special loader for the mesh data, so that future bathymetry data sets may be incoporated as well. Once the user has created a blank voxel canvas, using blank.pl. and drawn the objects to the canvas, the user has access to any one of many example Volumizer applications or may write their own, should the need for additional functionaility arise. One potentially interested pursuit would be to use voxel picking as means for 3-dimensional analysis: that is, find a location on track that contains a structural object, and use voxel picking - a procedure similar to 3-dimensional ray-casting - to find items in other tracks with the same density and highlight them as potential structural continuities. Hopefully, the accomplished work will provide information to relevant to locating such potentially interesting densities.

IV. Suggestions and Conclusions

i. Suggestions for Further Use as an Analytic Tool

The mechanics of sonar are quite complicated, and the authors do not pretend to fully understand them, since their backgrounds are in Computer Science rather than Ocean Engineering. It seems likely that there are many avenues and filters that can be designed and implemented that would provide quite useful information and would lead to actual object detection. Writing a short filter to the TrackFilter class and inserting a callback into the graphical interface and event handler would allow the user to implement such additional filters.

ii. Extensibility Arguments for Rendering Other Models

Since the VoxelDraw class has been written to generically take a object defined in real coordinates and to draw it into a discete volume space, this class could be used to prepare practically any volumetrwic model for rendering. One need only write a loader for the new data structure and use the provided source code template as an example.

iii. Implementation Problems

Their were many problems in implementing this project, many having to do with conceptually straightforward procedures that became technological frustrating. One example of a major stumbling blocks would be bridging the raw image data into a format that would be compatible with Volumizer - some seemingly legitimate TIFF images are simply not compatible. In addition, unfamiliarity with many of the technologies involved proved to be time-consuming and led to many pursuits that do not appear in the final product.

iv. Notes on Pursuing Further Optimizations

The code provided is reasonably efficient, but there are undoubtly areas that could be improved for performance. Some tile sizes for the images may be more optimal than others, as may be some volume sizes. Although space leaping may help performance, it is not clear how many of the textures are being cached and it is possible that a performance boost may be achieved by modifing code to eliminate some texure loads and implement better caching. Some resolutions and volume sizes may also improve appearance, since both contribute to volumetric sampling rates and to how the texels are mapped onto polygons during polygonization.

V. Appendix

i. Source Code Listings

Perl source files:	*Java source files:*	*Sample C++ source files:*
synch.pl	TrackViewer.java	voglMain.cxx
blank.pl	Track.java	voglArgsParser.cxx
mz.pl	Mesh.java	voglCache.cxx
logranges.pl	VoxelDraw.java	voglSpaceLeap.cxx
pnsranges.pl	MeshCanvas.java	*Project makefile:*
velocity.pl	TrackCanvas.java	Makefile
histogram.pl	TrackFilter.java	
histodif.pl	TrackList.java	
image.pl	TrackPicker.java	
image.pl	TrackMessageDialog.java	

image.pl	ActiveList.java
timer.pl	MenuItemList.java
utils.pl	

ii. Execution order

To create a simple 3-D model:

Run *synch.pl, blank.pl, VoxelDraw.java, and voglBasic (requires Volumizer).*

1. Synch.pl will synchronize the PNS and LOG files.

2. Blank.pl will create an empty voxel canvas.

3. VoxelDraw will create segmented track data and mesh objects and render them to the canvas.

4. VoglBasic will render the canvas using the Volumizer API.