doi:10.21311/001.39.5.23

# I4Ocean: An Interactive Simulation and Scientific Visualization Platform for Marine Application

**Pengbo Ji, Fenglin Tian, Shuai Liu, Yuchi Jiang, Ge Chen**

*College of Information Science and Engineering, Ocean University of China, Qingdao 266100, China*

**Abstract**

As an excellent tool for dealing with ocean issues, the marine geographic information system (MGIS) has been attached increasing importance by people. This paper introduces a new MGIS platform of interactive simulation and scientific visualization for marine applications named i4Ocean. It provides an intuitive 3D visual analysis and display tool for marine researchers. The platform is not only able to simulate various elements of the marine environment, but also visualize multi-dimensional data and oceanic model data. We design the rendering engine and scene management framework for marine applications to meet the needs of describing and analyzing the complicated ocean environment. In this paper we detail how to implement marine application on i4Ocean platform through the simulation of ocean environment and 3D streamline visualization of irregular flow field data. Several examples of ocean simulation, vector field data and scalar field data visualizations based on i4Ocean are given to prove the feasibility of i4Ocean. The platform allows the users to write particular programs on it to study marine phenomenon easily and shows the results in the graphical form. It also helps the researchers and forecasters to investigate, monitor, analyze and forecast the changing marine resources and environmental conditions.

**Key words:** I4Ocean, Marine Simulation, 3D Streamline, Scientific Visualization.

## 1. INTRODUCTION

With the concept of "Digital Ocean" being put forward, many researchers have been focusing on constructing practical system for effective ocean management and conservation. By putting the ocean physical, chemical, biological and geological information into a super computing system, MGIS makes the ocean a virtual graphical system to help people develop and protect the ocean. At the same time, it supports for the intuitive visualization of the marine data and resources sharing. However, as the advanced technology applied in exploring the ocean, the marine data volumes are growing exponentially. Because of the spatiotemporal, three-dimensional (3D) and intrinsically dynamic features of the marine data, it is difficult to make a breakthrough in digital ocean domain. For many people, the ocean looks uniform or without what we think of as spatial pattern. However, in reality, everything in the ocean, physics, chemistry, biology and human activities are explicit to a time and place. Compared with traditional Geographic Information System (GIS), there are much more requirements for MGIS. It should be not only able to analyze multi-dimensional and multi-source isomerous data, but also visualize spatio-temporal data to take broad brush views of ocean processes and human impacts on the ocean and reflect their spatiotemporal structure. There are some powerful physical models of the oceans (E.g., many spatiotemporal models of the oceans and atmosphere make use of the concept of "sigma coordinates" in defining their vertical dimension.) at both the global and regional levels. There seems to be little that our current GIS tools can do to neither support their spatiotemporal analysis nor to assist in visualizing their results. There has not yet been a popular platform that specializes in marine application and be able to simultaneously meet the requirements of both the simulation and the visualization.

In this paper, we aim to present a practical solution aiming at making up for the shortage of traditional GIS. We present a new MGIS platform for better visualizing the marine information and simulating the ocean scene in real time. The platform contributes to the management of marine resource exploitation, the protection of marine ecological environment, and the warning decisions of marine disaster, the scientific research of marine atmospheric environment and the marine public service. The platform introduced in this paper, which is called "i4Ocean", meaning "eye for ocean", is originally developed to help people to explore, manage and exploit the ocean. The acronym "i4" represents the four features of our platform : immersion, interaction, imagination and intelligence. Our platform is able to effectively meet the urgent needs of integrating real-time rendering marine big data, information visualization, simulation of marine operations and marine GIS analysis. In the meantime, secondary development and script development based on the platform are also supported.

While simulating the real ocean environment, i4Ocean improves the rendering engine of the traditional GIS platform. Now, it provides better support for simulating and visualizing measured data, remote sensing data and model computing data. Meanwhile we add volume rendering for 3D scalar data and 2D or 3D streamline for 3D flow vector data to i4Ocean. In this paper, the innovation points are listed as the following three aspects: First, the spatio-temporal data models and application framework is established for marine data storage, data analysis and data visualization services. Then, a rendering engine which is used for efficiently analyzing data and managing special effects is designed to support GPU acceleration. Finally, based on the rendering engine and the scene management framework, the scientific visualization of marine scalar field data and vector field data are accomplished in this platform.

## 2. RELATED WORKS

The Google Earth, World Wind and Skyline were all good tools to show information on the earth in 3D view. MyOcean, the EU multilateral cooperation projects, provided open and free marine analysis and forecasting services. It combined with satellite and observation data. The open source project of OsgOcean was part of the VENUS, which was also jointly developed by EU. It used vivid simulation method to reconstruct the ocean and the real scene of the sea floor. Gertman et al. developed RSVP focused on the fusion of multi-source remote sensing data and visualization (Gertman, Olsoy, Glenn and Joshi, 2012). And they did a good job on color profile and volume rendering for LiDAR data. Rautji et al. submerged a remote sensing data visualization system using the OpenGL graphics library and Java 3D technology to immerse users in data browsing (Rautji, Gaur, and Khare, 2013). China digital ocean prototype system (CDOPS), which was designed and constructed tentatively, provided a visualization, presentation and spatial analysis platform for 3D ocean monitoring data and information(Zhang, Dong, Li, Luo and Chi, 2011).

For a MGIS platform, the integration of virtual reality scenes is indispensable. Experts and scholars have done a lot of research on marine scene rendering, especially on water rendering. Blinn (Blinn, 1978), who was an earlier experts working on simulating water, presented bump mapping method to obtain the real rough surface texture, through disturbance surface normal vector. Since Fishman proposed height field method, it had been widely used in the field of water waves simulation. Fractal noise, also known as perlin noise, was produced by Perlin. With this method, Johanson generated sea surface height field. He presented a LOD method called “projection grid” to create a grid mesh whose vertices were even-spaced, not in world-space (Johanson, 2004). Bruneton et al. presented a new algorithm for water modeling, animation, illumination and rendering in real-time, at all scales and for all viewing instances, based on a hierarchical representation, combining geometry, normal and BRDF (Bruneton, Neyret and Holzschuch, 2010).

Another feature of i4Ocean platform is the merging of scientific data visualization methods. In the case of vector field visualization, many methods are practical, such as direct visualization, texture-based visualization, geometric-based visualization, feature-based visualization, partition-based visualization (Andrea, Robert, Robert, Ivan and Helwig, 2012). Geometric-based flow visualization shows the features of data by using streamlines, streak lines, time lines, and path lines (Rautji, Gaur and Khare, 2013). Take 3D streamline visualization as an example. Ye et al. (Ye, Kao and Pang, 2005) extended the flow guided approach proposed by Verma et al. (Verma, Kao and Pang, 2000)to visualize 3D flows. Marchesin et al. (Marchesin, Chen, Ho and Ma, 2010) described the complex structure of the flow focusing on streamline addition and removal algorithms by combining view-dependent and view-independent criteria to avoid visual clutter due to a potentially high density of streamlines. A different strategy to avoid this problem was introduced by Chen (Chen, Yan, Yu, Max and Ma, 2011) who combines the advantages of clustering methods and illustrative rendering techniques. Günther et al. presented a global line selection approach based on optimization process. He obtained view-dependent opacities of the line segments, allowing a real-time free navigation while minimizing the danger of missing important structures (Günther, Rössl and Theisel, 2013). In another case of scalar field visualization, volume rendering is a method of extracting meaningful information from volumetric data. Over the years many techniques have been developed to render volumetric data, Kruger et al. (Kruger and Westermann, 2003) first address the integration of early ray termination and empty-space skipping into texture based volume rendering on graphical processing units (GPU). Volume classification is a major issue in volume visualization. Transfer functions have been proved to be a powerful tool for classification (Haidacher, Patel, Bruckner, Kanitsar and Groller, 2010). Many researches (Rautji, Gaur and Khare, 2013; Sereda, Bartroli, Serlie and Gerritsen, 2006) focused on the transfer functions have been done. In our work we realize Ray-Casting volume rendering algorithm based on the rendering engine of i4ocean, which can help showing the feature of the ocean.

## 3. SYSTEM OVERVIEW

### *3.1. System Framework*

System framework design is a very complex work in software engineering, with the increasing demands of visual function. Thus, the number and scale of rendering modules in engine will be getting larger, which causes the more difficult management and extension. Comparing with the Model-View-Controller (MVC) architecture in network platform, the underlying architecture referred in this article is similar to MVC. But we change both the engine module partition and the subordinate function to meet the request of the ocean simulation and marine data visualization. As shown in Figure 1, the following section introduces MVAR (model-view-adapter-rendering engine) constructs used in i4Ocean.

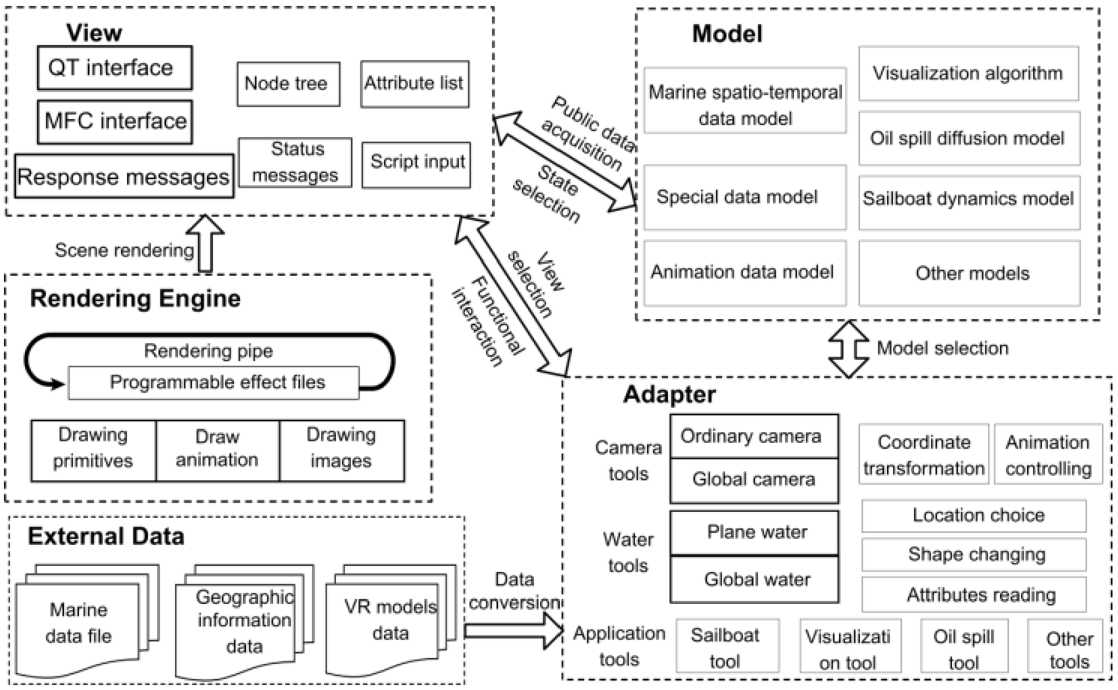

**Figure 1.** I4Ocean system architecture diagram

*View* is the operational user interface. The basic structure of i4Ocean consists of node tree, property list, output window and rendering window. These interfaces have been encapsulated into the engine core code. Given that construction of the complex ocean environment, we created a rendering engine using a variety of rendering effects to accomplish the marine data visualization. Every visual effect works with different drawing states accordingly. With state-management mechanism, we could effectively manage independent states(depth buffer, transparent channels, etc) to make them not influenced by each other. *Adapter* controls the realization and identification of application function, and executes corresponding code. *A*dapter gets input from the user, then calls *model* and *view* to complete the user's requirements. Finally, it determines on which *view* to display the returned data.

### *3.2. Rendering Engine*

The i4Ocean rendering engine is a programmable real-time rendering engine based on OpenGL 3D graphics library. Figure 2 shows the architecture of i4Ocean rendering engine and how we encapsulate the OpenGL API in the rendering engine. It takes data structure named *effect* as the smallest unit of rendering process to manage the special effects separately. It supervises and controls the OpenGL states, avoiding errors in states management. Each *effect* can contain one or more *techniques*. *Technique* is the approaches to accomplish the special effects, showing in different ways of implementation on the same *effect*. One *technique* can contain several *passes* while simple *technique* requires only one *pass*. *Pass* is rendering process with multiple render states. One *pass* contains vertex shader(VS), geometry shader(GS), tessellation shader(TS), pixel shader(PS) and some rendering states. Simultaneously, *pass* is a basic function unit. Geometric data, such as points and lines, gets into graphics card and runs according to a *pass*. It acts similar to Photoshop layers, adding layers one by one in order. Finally, the result can be generated. The most expensive operation is usually the code delivery processing from CPU to GPU (all the vertices in each pass are transferred from the CPU to the GPU). If the scene is very complex in the same case, it may result in poor performance.

### *3.3. Scene ManaGement Framework*

With the explosive expansion in the volume of marine data, the rendering demand of visualization for big data is also growing. We propose a kind of optimal scheduling strategy between computer's main memory and auxiliary memory based on the memory file mapping technology(Vrolijk and Post, 2006). As a result, i4Ocean platform can support the real-time visualization rendering of big data. We use data file storage structure based on *Itemdata* in this paper, which is the minimum rendering unit. The *ItemData* in figure 3 mainly contains the description information including the data pointer (character pointer), the data length (64 - bit integer), the original position in data file (64 - bit integer), whether the data contains out-of-core information or not (boolean) etc. Geometric model *entities*(including the vertex information, index information, texture information, etc.) are all organized by the *Itemdata*.

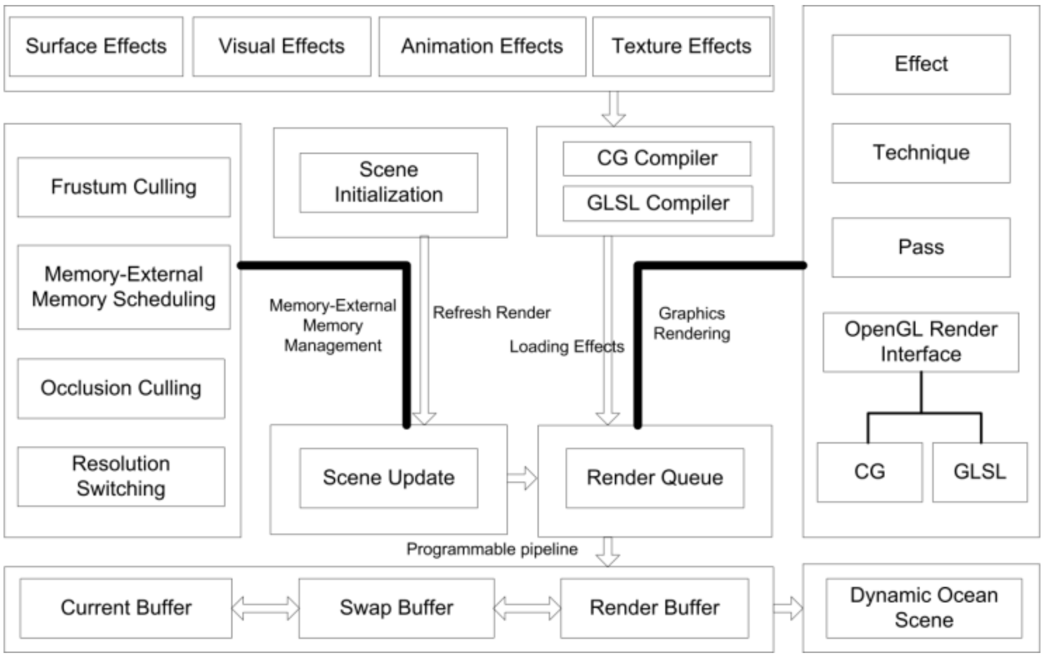

**Figure 2.** Rendering engine architecture diagram

The exported peripheral storage file contains both *index files* and *data files*. When loading the data, we load the information in the *index files* into memory first. According to the index information, using the windows memory mapping technology, we can map the corresponding data section in the data file into the memory for rendering. In the visualization process of big data, the data constantly exchanges between the inner and outer memory. During that, we should set a certain size of *memory buffer*. When the data size to be loaded is bigger than free *memory buffer* capacity, it is time to release some data have not been rendered in current memory, until the size of free memory buffer meet the requirements of the new data. The principle is “first in first out”. *Index file* stores the details of all the data in scene including index information of the data, the relationship between leaf *nodes* and *entities*, the starting position of each *Itemdata* in peripheral storage file, etc. Based on the data structure above, we achieved implementation to the operation of the data stream management.

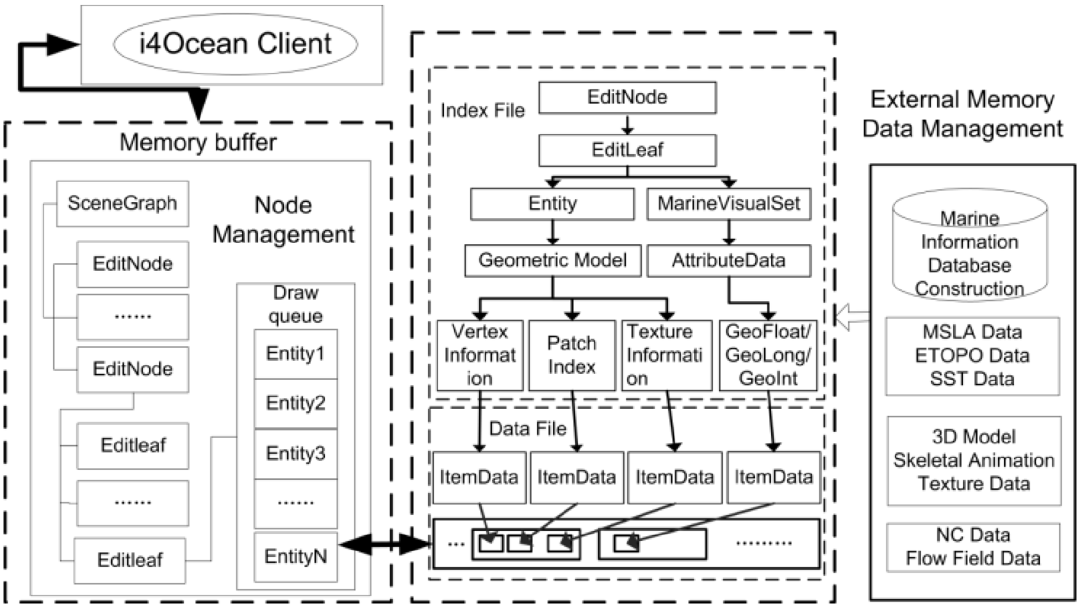

**Figure3.** Scene management and rendering structure diagram

There is only one *SceneGraph* node, which is the root node of all the scene nodes. A lot of functional nodes named *Editnode* are mounted under the root node. Each *Editnode* works as a parent node of one application function, such as one building node, one sailing node, one typhoon phenomenon visualization node. The *EditNode* contains many queues such as time list, depth list, latitude list, longitude list and attribute list. These lists are used to store the spatio-temporal and attribute data of geographic data. And the data is stored as custom *GeoFloat* or *GEOInt* formats according to its float or int type. A *Controller* used for controlling dynamic effects of time-varying data is bound to each parent *Editnode* or child *Editnode* in the scene tree. The leaf node named *Editleaf*, inheriting from *Editnode*, is the smallest authoring unit, providing a variety of matrix operation interfaces including rotation and translation. The *Entity* is the minimum geometry data unit for drawing. It is used to organize various geometric unit, to save description information, the geometric data and various property description.

## 4. VR APPLICATIONS

In this paper, we achieved high quality scene effect and the ocean simulation while guarantee well rendering efficiency. The sea-land virtual display system on our platform is developed based on ETOPO1 data published by U.S. National Geophysical Data Center. On one hand, the buildings, roads, vegetation on land are displayed in three dimensions. On the other hand, some virtual boats modeled with 3DMAX, virtual ocean surface and underwater bubble, light shafts, caustics, and fish underwater are vividly displayed in real time after the importing process in the last chapter.

We applied the method presented by Johanson (Johanson, 2004)easily to the engineered marine environment simulation on i4Ocean. We write GPU's vertex shader and pixel shader code in GLSL in this paper. First, with a periodic wave superposition, we get height field map and normal disturbance map of water surface grid. Then, we vividly simulate the phenomenon of the ocean surface reflection, refraction, etc. Finally we can get dynamical water surface real-time, with controllable parameter interface. We can change the surface appearance by these interaction parameters, such as water color, wave height, wave velocity, Fresnel coefficient etc. The implementation details of virtual interactive water is introduced in the following sections. The rendering process is shown in figure 4 below.

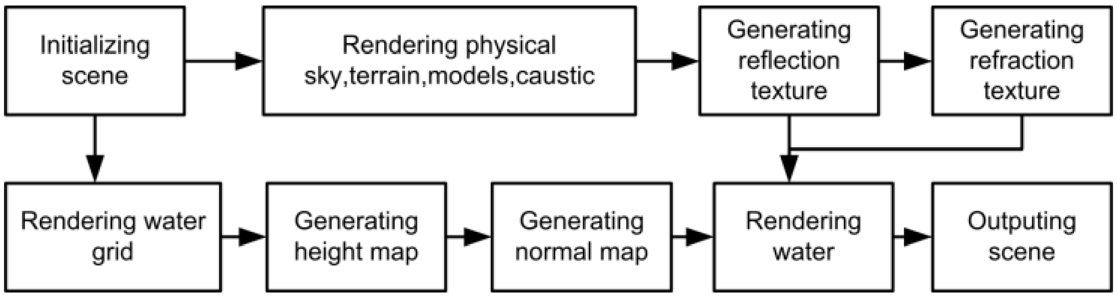

**Figure 4.** Rendering process of virtual water scene

### *4.1. Ocean Waves Modeling*

First, we create a 512 * 512 resolution fixed grid as shown in figure 5(b). One grid works as an *Entity* being mounted at one *Editleaf*. Thus we can create more than one water grid. All the information of vertex and index in the grid are respectively saved into two memory buffers which are named *VBO* and *IBO*. Then, we write effect shader files in GLSL for generating height map, normal map, superimposed water surface. Each render effects above is saved in the *Shape* data structure, which is separately applied to the three rendering passes. When the *Controller* is active, all the rendering processes illustrated in figure 4 are executed in real time after the *Initializing* process be finished.

There are many methods for the generation of wave height fields. Some methods are more concerned with the efficiency of real-time implementation than the detail information. Some other methods pay attention to the sense of reality but resulting in complex computing. We simulate the dynamic ocean by using perlin noise map as figure 5 (a). It can further increase efficiency by using noise texture directly. The height map is calculated according to the time parameter in real time as follows:

$$hight = saturate(1 - turbulence(pos / wavelength, octaves, ofset)) \tag{1}$$

$$offset = Time * (wavespeed * waveDir + wavegush) \tag{2}$$

Where *pos* is the original location of the vertices. The *wavelength* is the length of each wave form and *octaves* stands for an empirical value of 8.0. The function *turbulence (x, y, z)* in equation (1) is a custom disturbance function. The vertex offset in each rendered frame is defined as equation (2). Where *Time* is the system time, *wavespeed* is the user-defined flow velocity, *waveDir* is the user-defined flow direction, and *wavegush* is upwelling speed of waves. In the first pass of each frame, we generate a height map according to the above algorithm, which is shown in figure 5 (c). The normal map, shown in figure 5 (d), which is generated in the second pass, is calculated according to the height map in real time after pass 1. By Kriging interpolation algorithm, it is generated among every vertex height of the surface mesh. The normal map is used to guide the vertex variation rule and the optical calculation of water surface.

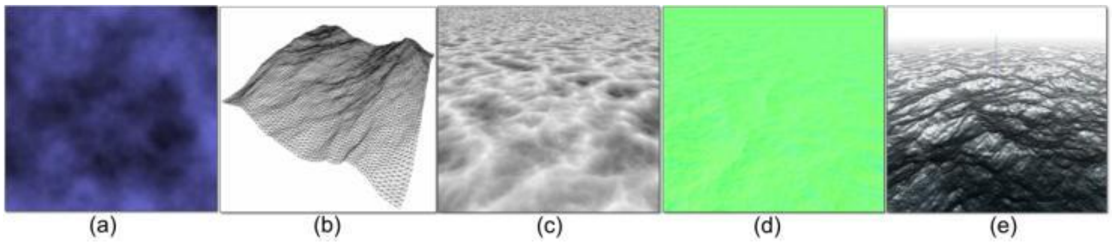

**Figure 5.** Some rendering results during creating water surface (a)Perlin noise map (b)Created grid (c)Generated height map (d)Generated normal map (e)Surface combined the height map and normal map

***4.2. The Optical Processing of Water Surface***

To build realistic virtual ocean scene, we also need to add various optical phenomena to the scene, such as water refraction, reflection, Fresnel phenomena and underwater caustics effect. Then, we add all the texture map to the water surface. And the height field map changes over time constantly. Coupled with optical effects, we can generate dynamic ocean surface vividly. A formula in GLSL to generate water surface as equation (3).

$$finalcolor = lerp(lerp(WaterColor, \mathrm{Re}\, frColor, DeepFactor), \mathrm{Re}\, flColor, fresnel) \quad (3)$$

Where *WaterColor* is the color of the water. *RefrColor* is the texture of refraction, and *ReflColor* is the reflection color map similarly. *DeepFactor* is grayscale image saving the water depth distribution information. And *fresnel* parameter works as weighting parameters between *RefrColor* and *ReflColor* to simulate Fresnel phenomenon. Through the calculation of parameters listed above, we have got all the pixel color data contributed to the screen. After that, we can produce the final pixel color on screen by integrating all the results.

In an integrated system, the number of objects to be rendered is so big that we need try our best to reduce the amount of calculation. Traditional reflection, refraction and caustic map generation require real-time computing, as a result, the calculations are inefficient from performance standpoint. With some simplifications and changes, we make the process adapt to the GPU rendering. In addition to the rendering camera, we create an extra camera to generate the reflection images. We need do some view-culling to reduces the amount of internal memory objects that will be rendered before paint. The extra camera locates in a symmetrical position with the rendering camera at the bottom of the water. The rendering result, which is the reflection map, is saved as a texture in frame buffer. Furthermore, refraction map can be generated by the rendering camera simultaneously, with the rendering result being saved as a texture in frame buffer too. Due to the effect of surface being focused and defocused, the beam exposure to the surface will be enhanced or attenuated when arrives at the bottom. The grain projected onto the underwater objects surface is so called caustic. In order to simplify calculation, we replace the calculating caustic phenomenon with transforming caustic textures.

We create a virtual physical sky scenery around to act as the actual sky. Thus it contributes to high-fidelity of water reflection. Then put the virtual scene in the sky ball and set camera position upon the surface of the water. Realistic simulation of sky, which plays an important role in VR scenes roaming and lighting. Atmospheric scattering is the main factor affecting the color of the sky and it is also the main reason why objects in the distance sank away from view. In this paper, referencing to atmospheric scattering physical model and GPU based atmospheric scattering calculation, we analyze the principle of atmospheric scattering. In this way, we realize the simulation of realistic sky in physical way. By controlling the direction of the sun light with GPU parameters, we accomplish the dynamic display of sunrise and sunset finally. More specifically, we construct sky model with uniform triangular mesh. In the pipeline of vertex shader, the outside integral and the inner integral of a number of sampling points are finished. While in the pixel shader pipeline, we make a fusion of the scattering integral results with the cloud image and accomplish the visual effects of sky. All the physical attributes, such as sun position and sky color, can be changed via the parameters in rendering effect files. The appearance of physical sky ball is shown as figure 6 (a).

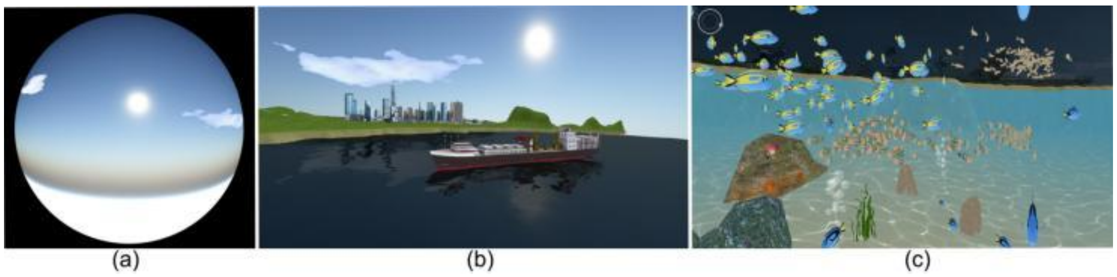

**Figure 6.** Some rendering results during creating virtual ocean scene (a)The appearance of physical sky ball (b)Land-sea landscape integration application (c)GPU-accelerated simulation of shoal and caustic

## 5. VISUALIZATION OF MARINE IRREGULAR FLOW FIELD DATA

The characteristics of marine dynamic process are mostly reflected by marine and atmospheric vector field data, which is of great significance for the research of large-scale ocean phenomena's formation, evolution and dynamic mechanism. In the visualization field for vector field data, 3D dynamic visualization of irregular data is a difficult issue(McLoughlin, Laramee, Peikert, Post and Chen, 2010). Taking Princeton Ocean Model(POM) computing data as the research object, we introduces a tetrahedron based 3D streamline generation approach. The approach puts forward a solution for this problem. POM model is a 3D baroclinic shelf shallow water numerical model for the ocean circulation. It uses sigma coordinate in vertical orientation, however, orthogonal curve coordinates is chosen in horizontal orientation. Therefore, it does make it difficult to visualize flow field data with streamline method. Combining the distribution-based seeds generation and tetrahedral mesh generation algorithms for irregular vector field data, we smoothly achieve streamline visualization on i4Ocean. A summary of our method is available as figure 7 below.

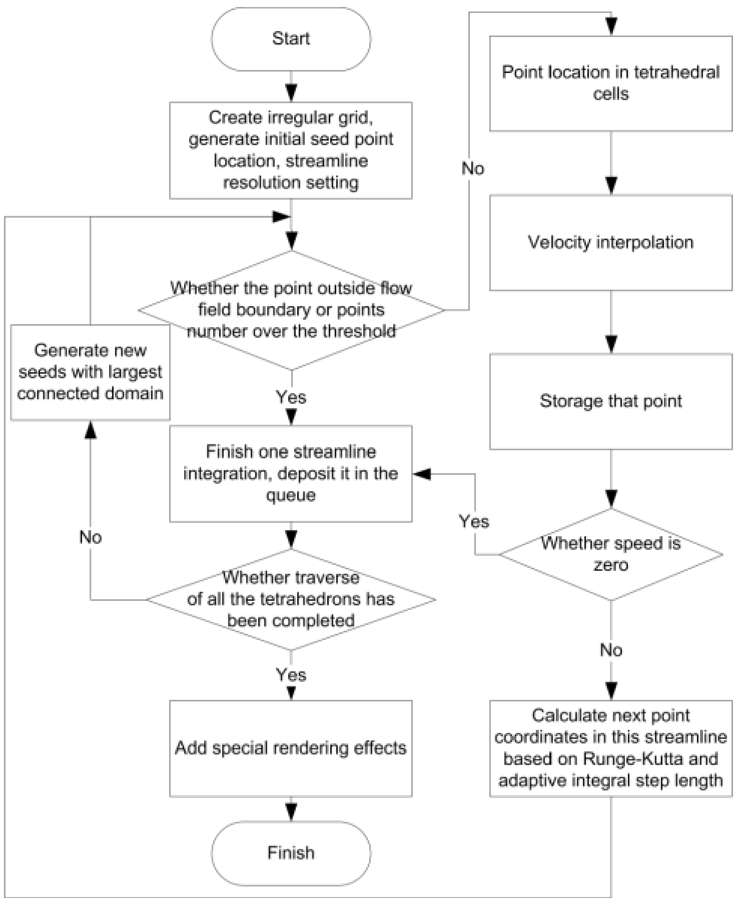
Start
Create irregular grid, generate initial seed point location, streamline resolution setting
Whether the point outside flow field boundary or points number over the threshold
No
Yes
Point location in tetrahedral cells
Velocity interpolation
Storage that point
Whether speed is zero
Yes
No
Calculate next point coordinates in this streamline based on Runge-Kutta and adaptive integral step length
Finish one streamline integration, deposit it in the queue
Whether traverse of all the tetrahedrons has been completed
No
Generate new seeds with largest connected domain
Yes
Add special rendering effects
Finish

**Figure 7.** The major steps for our tetrahedral-based 3D streamline generation framework

### *5.1. Streamline Placement Strategy*

Because the POM data comes in irregular patterns, we cannot apply the traditional method to streamline integrals easily. We need do some preprocessing on data before. In the initialization stage, constructing hexahedral grid for 3D irregular flow field data will be finished. Then, each hexahedral cell is decomposed into 6 tetrahedral cells. In this paper, the size of the grid may be configured according to the user's needs. To speed up the process of points locating and values interpolating, saving the topological relationship among the tetrahedral cells by a specific data structure is a good choice.

There is no clear judgment or definition about the quality of the streamline placement strategy at present. Generally, the strategy that can generate longer streamline is considered better than the one with shorter. Thus, by determining the seed point position in the flow field, we can determine the streamline placement strategy. In this paper, we propose a seeding strategy based on spatial distribution of the probability. The specific issue method mentality is: First, take the hexahedral grid generated in the initialization stage as control grid, initialize each cell of control grid with empty state. Then, set all the cells being crossed by streamline to be full state. Traverse all the grids and find the largest interconnecting region with empty state in control grid. Take the region center as the starting point position of next streamline and set the crossed grid cell to be full state also. Finally, repeat the process above until all the cells of control grid are filled.

Point location is very important in the flow field visualization. Generally, transforming the physical space of flow filed into computational space in curved grid by Jacobi matrix is the most common method to solve this problem. Our method adopts a kind of volume coordinates algorithm, which is similar to the normal vector algorithm. It helps us to understand the relationship between point and tetrahedron and to avoid repeating calculation.

### *5.2. Streamline Integral*

Among the constructing numerical integration methods for fluid, the familiar ones are Euler method, second-order runge-kutta method and fourth-order runge-kutta method. The calculation accuracy of Euler method is not high. The fourth-order runge-kutta method has high accuracy but high complexity. The second-order runge-kutta method is a compromise of the two methods above. It has both good precision and high computing efficiency, so we choose the second-order runge-kutta method for streamline integral operation in this article.

In the process of building streamline, proper integral step will greatly help improve the calculation precision, increasing calculation speed and saving considerable time. In our research, we adopt the method of dynamic streamline integral step. It considers two factors: 1) Control integral step size according to the change of the velocity vector direction. If the angle between two velocity vectors of adjacent points in streamline is too big, that means changing very rapidly. Therefore, we should make the integral step size shorter. 2) Determining the integral step size according to the radius of largest inscribed circle within the tetrahedral cell is another decide-consider option. When the step size is so small that generating one streamline in a tetrahedral cell costs too many steps, we take the radius as integral step size. That way, we make the integral process jump out of the loop in one tetrahedron.

### *5.3. High Perception of Streamline Visualization*

The result after integral is saved in lines as shown in figure 8(a) below. The attributes of position, color and opacity that vertices have are kept in the vertex buffer defined in memory. In order to obtain perfect dynamic expression effect, we make the best use of GPU to accelerate the algorithm process and to transform geometric shape of streamlines. Geometry shader, the third shader after the vertex shader and fragment shader, is formally introduced in shader model 4 (the fourth generation of graphics shader architecture). That has become the core in OpenGL3.X, giving programmers more freedom and flexibility. After the geometric deformation in geometry shader, we transform the input curves into stream tubes before add arrows. Accordingly, we can express the meaning of flow fields in streamline way. Furthermore, we also can transform it to other deformations, such as ribbon, wafer, strips, etc.

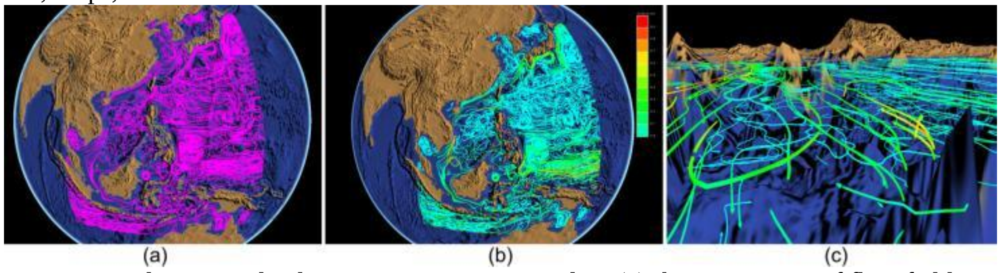

**Figure 8.** Some rendering results during creating 3D streamline (a)The appearance of flow fields with lines (b)Mapping color to velocity in streamline (c)3D flow distribution in underwater observation

However, the qualitative expression of streamline distribution can't meet the demand of the marine science research. Considering integrated velocity distribution, we map the velocity in each point on streamline to color distribution. By that analogy, the information of sea temperature and sea salinity all can be mapped to a color bar with quantitative expression of marine information. Besides color mapping, we give another attribute, opacity, to the streamlines. Using update mechanism in our application platform, combined with time control, we add dynamic flowing effect to streamline. The method is best applied at time-varying path line and time line visualization.

## 6. SOME VISUALIZATION CASES

### *6.1. 2D Streamline Visualization of Wind Field Data*

Realistic 2D evenly-spaced streamline for 2D wind field adopts four-order Runge-Kutta integration and cartesian orthogonal grid to keep vector field evenly-spaced. Simultaneously, it combines topology-based feature-extracted technology and high perceptual texture mapping technology to make streamlines express the right direction as well as look authentic. The left picture in figure 9 below is 2D streamline visualization of sea surface wind field data in the South China Sea and the right one is 2D streamline visualization of global wind field data.

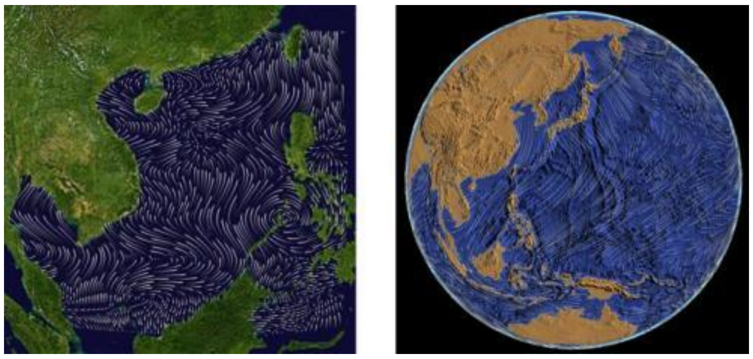

**Figure 9.** Rendering results of 2D static streamline visualization for wind field data

### *6.2. 2D Streamline Visualization of Flow Field Data*

In 2D dynamic visualization, we put massive Lagrangian advection particles to the domain. Streamlines are generated through multi-pass primitive distortion on GPU and numerical integration, preserving spatial and temporal continuity. Probabilistic density control based on radial basis function ensures uniform distribution of particles and full expression of the flow field characteristics. The pictures in figure 10 below are high-precision dynamic 2D streamline visualization of sea surface flow field data, which is inverted from the gridded maps of sea level anomaly data. The rendering result based on the Mercator projection is shown on the left and the right one is in global view.

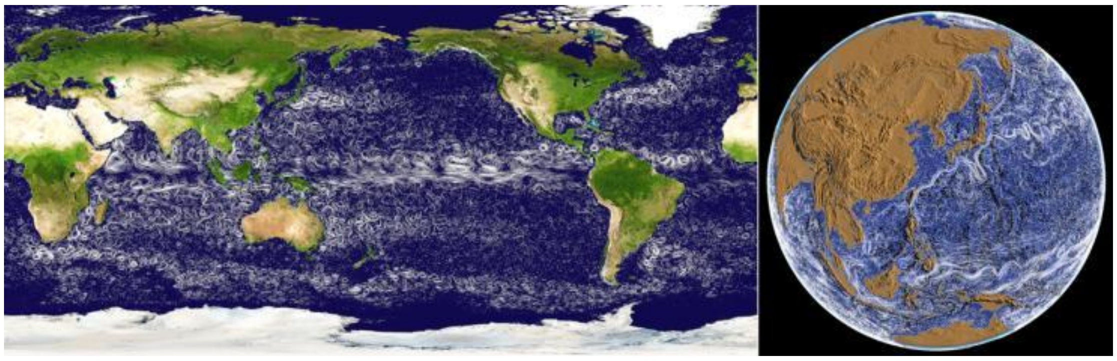

**Figure 10.** Rendering results of 2D dynamic streamline visualization for flow field data

### *6.3. Visualization of Marine 3D Scalar Field Data*

As an important index of physical process, chemical process, biological process and geological process, the scalar data in marine environment (temperature, salinity, density, etc.) embodies the multi-dimensional dynamic change characteristics of various marine parameters. Further analysis based on Ray-Casting volume rendering algorithm and extracting the features using transfer function can help showing the inner evolution rules of the ocean. In figure 11, the picture on the top left is the Western Pacific Warm Pool feature effect. The blue, yellow, red areas represent the temperature areas of 11℃-13℃, 20℃-23℃, 28℃-30℃. The picture on the bottom left corresponds to the Atlantic high-salt area. The blue, yellow, red areas represent the salt of 34,35,36. The right side is the enlarged detail effect.

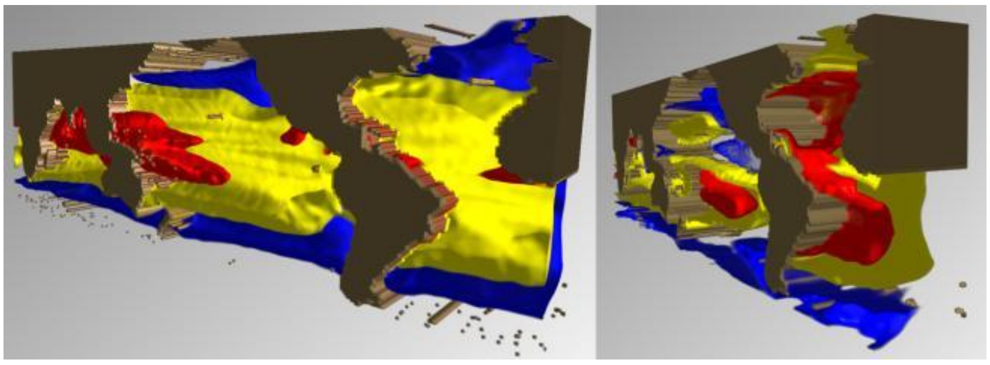

**Figure 11.** Feature extraction of the western pacific warm pool and the atlantic high-salt area based on global argo data

## 7. CONCLUSION AND FUTURE WORKS

In this paper, we use various methods, such as management and scheduling of big marine data and marine virtual reality simulation technology, to accomplish dynamic and multi-dimensional rendering of global marine data. We have proposed and realized a prototype system. The system can simulate visible elements in marine environment, for example, weather, sea surface, marine creatures, submarine model and seabed terrain. It can also dynamically visualize important marine information from multi-angles in multi-modes, including rainfall, water color, water vapor, sea level anomaly, sea surface wind, sea surface pressure, significant wave height and sea surface temperature. Based on GPU programming, the prototype system uses self-developed rendering engine to calculate various visual effect and puts it under standardized effect management modules. Thus it leads to a bigger role in marine virtual reality technology and marine scientific visualization technology. For marine scientific researchers, it provides effective, convenient and direct-viewing 3D virtual environment and acts as a useful tool for visual analysis and display. We also designed data structure that is suitable for distributed storage and parallel analysis. This data structure can be used in interactive and 3D visualization for atmospheric and oceanographic spatio-temporal data. It is suitable for big marine data mining, efficient marine data managing and information extracting as well.

The future research direction:

(1) For the openness of the platform, it is not enough for a global ocean application platform of multi-type data visualization and virtual reality, which provide and collect data solely by developers. The platform should provide open data interface and visualization methods to enable users or scientific organizations upload their own research data or visual analyzing algorithms online.

(2) For the visualization of the platform, users want to do more than just visualize marine data, but data mining, feature extraction, analysis and prediction on marine data. The next step in the research for improving the platform is to study analysis method which can be used to accelerate graphic operations and data mining theory based on GPU computing.

(3) For the simulation of the platform, taking the curvature factors of the spherical platform into account and creating global ocean environment with optical effects are the work we need to do. By making use of tessellation shade(TS) and compute shader(CS) in OpenGL programmable rendering pipeline, we will get view-dependent adaptive subdivision mesh. This work will help for accelerating graphics operations and computational efficiency in rendering.

**Acknowledgements**

This work has been sponsored in part by the Funds for International Cooperation and Exchange of the National Natural Science Foundation of China (Grant No. 613111035), and by the National High Technology Research and Development Program of China (Grant No. 2013AA09A506-4). I would like to personally thank The First Institute of Oceanography of SOA who provides the POM data for me. I would also express my heartfelt gratitude to Sun Miao, who helps me during this thesis.

## REFERENCES

Andrea, B., Robert, C., Robert, P., Ivan, V. and Helwig, H. (2012) "Illustrative Flow Visualization: State of the Art, Trends and Challenges", *Proceedings of Eurographics*, pp. 075-094.

Blinn, J.F. (1978) "Simulation of wrinkled surfaces", *Computer Graphics*, 12(3), pp.245-251.

Bruneton, E., Neyret, F. and Holzschuch, N. (2010) "Real-time Realistic Ocean Lighting using Seamless Transitions from Geometry to BRDF", *Computer Graphics Forum*, 29(2), pp.487-496.

Chen, C.K., Yan, S., Yu, H.F., Max, N. and Ma, K.L. (2011) "An Illustrative Visualization Framework for 3D Vector Fields", *Computer Graphics Forum*, 30(7), pp.1941-1951.

Gertman, V., Olsoy, P., Glenn, N. and Joshi, A. (2012) "RSVP: Remote Sensing Visualization Platform for Data Fusion", *IEEE Virtual Reality Workshop on Immersive Visualization*, pp. 161-168.

Günther, T., Rössl, C. and Theisel, H. (2013) "Opacity Optimization for 3D Line Fields", *Acm Transactions on Graphics*, 32(4), pp.120:1-120:8.

Haidacher, M., Patel, D., Bruckner, S., Kanitsar, A. and Groller, M.E. (2010) "Volume Visualization Based on Statistical Transfer-function Spaces", *The 3rd IEEE Pacific Visualization Symposium*, pp. 17-24.

Johanson, C. (2004) "Real-time Water Rendering", M.S. diss., Lund University.

Kruger, J. and Westermann, R. (2003) "Acceleration Techniques for GPU-based Volume Rendering", *Proceedings of the 14th IEEE Visualization*, pp. 38-43.

Marchesin, S., Chen, C.K., Ho, C. and Ma, K.L. (2010) "View-dependent Streamlines for 3D Vector Fields", *IEEE Transactions on Visualization and Computer Graphics*, 16(6), pp.1578-1586.

McLoughlin, T., Laramee, R.S., Peikert, R., Post, F.H. and Chen, M. (2010) "Over Two Decades of Integration-Based, Geometric Flow Visualization", *Computer Graphics Forum*, 29(6), pp.1807-1829.

Post, F.H., Vrolijk, B., Hauser, H., Laramee, R.S. and Doleisch, H. (2002) "Feature Extraction and Visualization of Flow Fields", *The Eurographics Association*, pp. 69–100.

Rautji, S., Gaur, D. and Khare, K. (2013) "Immersive 3D Visualization of remote sensing data", *Signal & Image Processing: An International Journal*, 4(5), pp.61-73.

Sereda, P., Bartroli, A.V., Serlie, I.W. and Gerritsen, F.A. (2006) "Visualization of Boundaries in Volumetric Data Sets Using LH Histograms", *IEEE Transactions on Visualization and Computer Graphics*, 12(2), pp.208-218.

Verma, V., Kao, D. and Pang, A. (2000) "A Flow-guided Streamline Seeding Strategy", *IEEE Proceedings of the conference on Visualization*, pp. 163-170.

Vrolijk, B. and Post, F.H. (2006) "Interactive Out-of-core Isosurface Visualisation in Time-varying Data Sets", *Computers & Graphics*, 30(2), pp.265-276.

Ye, X., Kao, D. and Pang, A. (2005) "Strategy for Seeding 3D Streamlines", *16th IEEE Visualization Conference*, pp. 471-478.

Zhang, X., Dong, W., Li, S.H., Luo, J.C. and Chi, T.H. (2011) "China Digital Ocean Prototype System", *International Journal of Digital Earth*, 4(3), pp.211-222.