基于 Cg 和 OpenGL 的实时水面环境模拟

马 骏[1]，朱衡君[1]，龚建华[2]
（1.北京交通大学机械与电子控制工程学院，北京 100044；2.中国科学院遥感应用研究所，北京 100101）

摘 要：实时水面效果可以大大增强虚拟现实系统的沉浸感。基于 Cg 图形硬件开发语言以及 OpenGL,综合利用 P-buffers 技术、凹凸纹理技术、投影纹理技术以及动态纹理技术模拟了一个实时的、包括反射、折射、水面波动、太阳波光以及 Caustics 等效果的水面环境。该方法仅需要两个三角形单元来构建水面。

关键词：虚拟现实；实时水面；Cg；P-buffers；凹凸纹理；投影纹理；OpenGL
中图分类号：TP391.9 **文献标识码：**A **文章编号：**1004-731X (2006) 02-0395-06

Simulation of Real-time Water Surface Based on Cg and OpenGL

MA Jun[1], *ZHU Heng-jun*[1], *GONG Jian-hua*[2]
(1.School of Mechanical, Electronic and Control Engineering, Beijing Jiaotong University, Beijing 100044, China;
2.Institue of Remote Sensing Application, CAS, Beijing 100101, China)

Abstract: The effect of real-time water surface can consumedly enhance the immersion characteristic of virtual reality system. Based on Cg and OpenGL, some technologies including P-buffers, Bump Mapping, Projective Texture Mapping and Dynamic Texture Mapping were used synthetically to simulate a real-time water surface. In this simulation, some effects, such as reflection, refraction, wave, sun light reflection and caustics, were realized and *the water surface was constructed only by two triangle*s.

Key words: virtual reality; real-time water surface; Cg; P-buffers; bump mapping; projective texture mapping; OpenGL

引 言

自然场景的描述是虚拟现实技术和计算机图形学的研究热点，其中水面的模拟是自然场景描述的重点，对增加虚拟自然环境的沉浸感起到了非常重要的作用。水面模拟主要分为实时性的和非实时性的两类。非实时性的水面模拟目前已经有了很多工具，例如Autodesk公司的3DS MAX™、Alias公司的MAYA以及小巧但是功能强大的自然景观制作软件TerraGen等，这些软件能够渲染出真实感非常强的水面效果。实时性的水面模拟主要集中应用在虚拟现实系统中。所谓实时性就是在软件系统中能够和用户达到实时交互的程度。试验表明，软件系统必须以不低于20帧/秒的刷新率刷新场景才能获得实时的感受。虚拟现实系统必须兼顾实时性和真实感，力争达到既有较逼真的效果又有较高的绘制刷新率。事实上，虚拟现实系统就是要在真实感和实时性上取得一个合理的折中。虽然计算机硬件和过去相比取得了非常大的进步，但是就水面模拟来讲，既要达到实时性又要有前述软件所能渲染的效果，目前还是无能为力的，因此必须采用一些算法和技术来模拟以达到实时效果。

收稿日期：2004-12-07 **修回日期：**2005-11-09
基金项目：国家自然科学基金项目（40341011）
作者简介：马骏(1975-)，男，河北沧州人，博士生，研究方向为虚拟现实、车辆运行仿真、虚拟地理环境等；朱衡君(1950-)，男，上海人，教授，博导，研究方向为虚拟现实、车辆运行仿真、检测与故障诊断等；龚建华(1965-)，男，浙江海盐人，研究员，博士，研究方向为地学可视化、虚拟地理环境等。

1 相关研究及开发工具

1.1 相关研究

在虚拟现实和计算机图形学领域，实时水面模拟始终受到众多学者的关注。Nick Foster 基于 Navier-Stokes 方程对三维水面进行了模拟[1]，该模型实现较为复杂。Kei Iwasaki 利用图形硬件实现了一种小面积水域的快速水面折射反射光影效果的模拟[2]。Kei Nakano 则利用粒子系统对小面积水域进行了模拟[3]。Jerry Tessendorf 采用快速傅立叶变换（Fast Fourier Transformation，FFT）产生平铺的高程图，然后在此基础上对海水进行了模拟[4]，该方法灵活性很强，既可以用于高质量的水面视频渲染，也可以用于实时的水面模拟。Vladimir Belyaev 在文献[4]的基础上对构建水面的网格进行层次细节（Levels of Detail，LOD）处理，并加入水面的反射和折射效果，使所模拟的水面真实感得到进一步提高[5]。Damien Hinsinger 建立了远离海岸的海浪可编程模型，该方法模拟的海面具有实时动画效果，并允许用户交互地“飞过”无边的大海[6]。上述方法描述的水域要么偏小（如文献[1-3]），要么远离海岸（如文献[6]），而且均是应用一定数量的网格来构建水面，然后采用不同的数学模型来控制网格点的高程，最终模拟近似平静（和波涛汹涌相比）的水面。随着网格的格点数增加，其效果越逼真，但是同时需要绘制的几何图元也越多，造成实时性下降。为了取得较好的实时性，减少场景绘制几何图元的数据量是最直接、最有效的办法，因此本文摒弃了前面文献所叙述的通过高分辨率网格模拟波

纹的方法，而仅采用两个三角形图元组成的四边形来构建水域，通过映射动态凹凸纹理（Bump Mapping）来描述水面的波动、通过映射动态纹理和投影纹理（Projective Texture Mapping）来描述水面的 Caustics 效果[2]和太阳波光效果、通过映射 P-buffers 技术获得的反射纹理和折射纹理来描述水面的反射和折射效果、通过一定的数学模型来扰动这些纹理从而获得水面波动扭曲效果，所有这些均利用 OpenGL 和 Cg（C for graphics）高级图形硬件开发语言来实现。本文第 2 节介绍水面反射、折射及波动效果的实现方法，第 3 节介绍太阳波光和 Caustics 效果的模拟。

1.2 开发工具

本文的实时水面模拟是基于OpenGL和Cg来实现的。OpenGL是在SGI等多家世界闻名的计算机公司的倡导下，以SGI的GL三维图形库为基础制定的一个通用共享的、性能卓越的开放式三维图形标准[7]。

Cg是最早为可编程图形硬件设计的高级编程语言。在2001年至2002年，NVidia公司与Microsoft公司密切合作，继承了C语言的语法和语义、非实时着色语言（shade language）以及实时三维图形库OpenGL和Direct3D三大技术遗产而开发了最初的Cg语言。Cg程序需要可编程图形硬件(第四代图形处理器的支持，可以稳定地工作在所有主要图形硬件厂商的可编程图形处理器上。图形处理器（Graphics Processing Unit，GPU）不同于中央处理器（CPU），它能够执行复杂的图形计算，是专门为处理图形设计的，具有高效性和专用性，这也是Cg得以诞生的原因。GPU所执行的两个主要操作是顶点操作（vertex operations）和片元操作（fragment operations），Cg与二者对应的就是顶点程序(vertex program)和片元程序(fragment program)。顶点程序能在GPU中对模型网格中的每个顶点执行某种计算，而片元程序是在GPU中对模型网格的每个片元执行指定计算。Cg是基于数据流模型的，渲染三维场景时，每当一个顶点被处理或者光栅化产生一个片元，程序员编写的顶点或片元程序就会被执行。顶点程序总是在片元程序之前运行，并且其输出参数可以直接作为片元程序的输入参数。片元程序对每个片元进行操作并输出操作结果，通常表现为片元的颜色值。Cg的出现使得控制使用可编程图形硬件绘制的物体的形状、外观和运动成为可能，并为开发人员提供了一个新的、对图形硬件操作的抽象层，使得程序员不必再直接使用图形硬件汇编语言来进行程序编写[8]。由于Cg的片元程序可以对光栅化后的每个片元进行操作，因此虽然本文的水面仅用2个三角单元构建，仍然可以实现对其所映射的纹理进行波动扭曲处理，这在OpenGL中无法完成。

2 带有反射、折射及波动效果水面的实现

2.1 水面几何体的创建

前述文献对于水面的模拟是基于一定分辨率的网格来实现的，在该网格上加载某种数学模型来获得水面波动的频率和振幅，从而获得近似平静的波动效果（诸如文献[5,6]）。这对于单纯模拟一个广阔的水域效果较好，但是对于一个复杂的虚拟场景来讲，水面网格势必造成系统绘制压力的增加，降低实时性。因此本文采用1个四边形来构建水面，该四边形仅由2个三角形构成，可以大大减少系统需要绘制的几何图元，最终降低送入渲染管道的数据量。

2.2 反射、折射纹理的获得

2.2.1 反射、折射原理

水面反射现象是指：将处于水面以上的可见物体视为光源，如图1a所示，其A点发出的光线有一部分到达水面（称为入射光线$\vec{I}$）会经过水面的反射（称为反射光线$\vec{R}$）进入人的眼睛，给人造成光线是由水面下A'发出的假象，从而形成可见物的倒影。根据光的反射定律，反射光线和入射光线对称的分布于反射表面的法向$\vec{N}$两侧，可以认为水面是纯镜面，因此，如果$\vec{I}$、$\vec{R}$、$\vec{N}$均为单位向量则满足下式

$$\vec{R}=2\vec{N}(\vec{N}\cdot\vec{I})+\vec{I} \tag{1}$$

折射现象是由于光在不同介质中的传播速度不同造成的。介质的密度越大光传播速度越小，因此当光由一种介质进入另一种不同密度的介质时，会发生光线改变传播方向的现象，即为折射现象。Fresnel定律对光的折射描述如下

$$h_1\sin q_I=h_2\sin q_T \tag{2}$$

其中h_1、h_2分别是入射光所在介质的折射系数和折射光所在介质的折射系数，q_I、q_T分别是入射角和折射角。当物体水下部分A点发出的光由水进入空气中时(如图1b，称为入射光线$\vec{I}$)，由于水的密度大于空气，光线会改变方向形成折射光线$\vec{T}$进入人眼，人眼会误认为A'点是原物体的A点，因此造成物体水下部分变短的错觉。图中，h_1、h_2分别是水和空气的折射系数且满足式(2)。

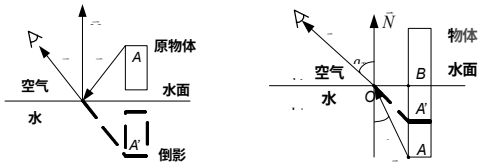


图 1 反射、折射原理图

2.2.2 反射纹理的获得

传统获得镜面反射成像的绘制方法有两种：一是采用OpenGL中的模板缓存(stencil buffer)进行绘制，二是绘制景物的镜像图，然后对反射面作透明处理。这两种传统方法存在一些弊病，即：使用范围较窄，并且不易实现水面波动造成倒影扭曲的效果。文献[9,10]分别对水下和水上的折反射光学效果进行了描述，其中文献[10]采用了投影纹理映射的办法来绘制反射倒影：即用一个镜像的视景体把整个场景（除了水面）渲染成纹理，该纹理作为反射纹理采用纹理投影的方法映射到水面，获得水面的倒影效果。这种贴反射纹

理的方法模拟水面更容易获得水面波动造成倒影扭曲的效果，因此本文基于此思想结合Cg实现另一种较为简单的方法：利用窗口坐标查询反射纹理的办法来获得倒影效果 见2.3节。反射纹理就是将需要反射的景物沿水面做镜像处理然后去除水面以上的部分进行绘制并渲染成的纹理 实际上需要镜像的部分仅仅是水面以上的所有场景 以水面为分界面来划分，一般存在三种情况：1.整个景物处于水面以上，例如房屋、建筑等；2.整个景物部分在水面以上，部分在水面以下，例如地形、船只等；3.整个景物完全在水面下面，例如水草等。对于第3种情况（如图2中物体B）不会形成反射倒影，只涉及折射问题，将在2.2.3节讨论。第1种情况处理比较简单，将整个景物进行镜像即可，如图2所示物体A。第2种情况处理相对麻烦，以地形为例，图中是地形某横断面。地形做镜像处理后（如2图中虚线），剔除水面以上的部分剩余便是需要渲染到反射纹理的部分 剔除方法采用OpenGL提供的附加裁减平面来实现。OpenGL中除了构成视景体的6个裁减面外还提供了至少6个附加的裁减平面。空间任一平面的方程是

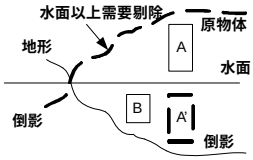


图2 反射纹理获得原理图

$$Ax+By+Cz+D = 0 \tag{3}$$

因此，附加裁减面可以由该方程来表示，并由4个系数A、B、C、D唯一确定。OpenGL中满足

$$(A，B，C，D)M^{-1}(X_e，Y_e，Z_e，W_e)>0 \tag{4}$$

的半空间内的点将保留，其余的将被裁减掉，其中$(X_e，Y_e，Z_e，W_e)$是视点坐标，M是当前模型视图变换矩阵。场景中获得水面的平面方程并作为裁减面，然后利用glClipPlane函数将水面以上的景物剔除、渲染，便获得了反射纹理。

获得反射纹理采用P-buffers技术。P-buffers为离屏渲染提供了方便，它有很多属性与屏幕渲染的帧缓存属性相同例如：RGBA颜色、Z深度以及Stencil等，但是P-buffers与帧缓存的关键不同点是其尺寸以及位深属性是完全独立于当前显示模式的。创建P-buffers以后对其进行读写、绘制与帧缓存没有不同，另外NVidia以及ATI公司的显卡对P-buffers操作均进行了硬件加速，而OpenGL的两个扩展WGL_ ARB_pbuffer和WGL_ARB_pixel_format对其有很好的支持。利用P-buffers获得反射纹理的步骤如下：

(1) 配置P-buffers。配置P-buffers主要注意：OpenGL除2.0版本外，均要求纹理的大小为$2^m\times2^n$（m，n是正整数），而反射纹理应该是整个绘图区域，并不能保证这个条件，因此在创建纹理对象、设置像素格式、创建P-buffers时必须引入相应的OpenGL扩展来消除这种限制。

(2) 创建反射的动态纹理。一般由P-buffers获得动态纹理有三种方法：1.利用函数glReadPixels读出P-buffers中的颜色数据，然后利用纹理函数glTexImage2D来创建纹理；2.通过函数wglShareLists来共享P-buffers及帧缓存的纹理空间，然后调用函数glCopyTexSubImage2D来建立纹理；3.如果显示硬件支持，利用WGL_ARB_render_texture扩展可以直接将P-buffers渲染成纹理。这三种方法的效率是递增的，第1种最低，第3种最高。本文采用的是第3种方法。

(3) 在适当时候将渲染的反射纹理作为参数传给Cg程序。

2.2.3 折射纹理的获得

折射纹理的获得原理和反射纹理相同 但是需要渲染为折射纹理的部分是在水面以下场景。以水面为分界面划分场景的情况见2.2.2节，因此只需要考虑第2、3种情况。完全在水下的景物不需要特殊处理，对第2种情况需要剔除水面以上的部分，剔除方法同2.2.2节。由于介质的折射系数不同，因此光线折射的程度也不同，常见介质的折射系数见表1。

表1 常见介质的折射系数

介质	折射系数	介质	折射系数
真空	1.0	水	1.333
空气	1.0003	玻璃	1.5

场景中我们考虑的是空气与水的折射问题 由图1b可以获得下式

$$\overline{AB}\cdot tgq_I = \overline{A'B}\cdot tgq_T \tag{5}$$

因此

$$\frac{\overline{A'B}}{\overline{AB}} = \frac{\cos q_T}{\cos q_I}\cdot\frac{h_2}{h_1} \tag{6}$$

由表1可得$h_1=1.333$，$h_2\approx1.0$，最终式(6)变换为

$$k=\frac{\overline{A'B}}{\overline{AB}}=\frac{1}{1.333}\sqrt{\frac{(1+\sin q_T)(1-\sin q_T)}{(1+\frac{\sin q_T}{1.333})(1-\frac{\sin q_T}{1.333})}} \tag{7}$$

该函数的图形如图3中的虚线曲线，显然，该函数是递减函数，并且$q_T\in[0,p/2]$时，$k\in[0.75,0]$。实际上k是一个缩放系数，其作用是将场景模型在竖直方向上以水面为基准进行k倍缩放从而模拟折射效果。缩放后用水面作为附加裁减平面将水面以上部分进行裁减获得折射场景 最后将该场景渲染到P-buffers而获得折射纹理。由于系数k的计算包含开方、浮点数除法以及正弦函数，会影响绘制效率，因此将该函数简化为由两个线段组成的分段函数，如图3所示实线。分段函数的方程如下

$$\begin{cases}k=-0.16q_T+0.75;0\le q_T\le0.8\\ k=-0.8q_T+1.26;0.8<q_T\le1.57\end{cases} \tag{8}$$

这样，系数k的计算仅包括浮点数的乘法和加法，可以提高计算和绘制效率，而且由于人眼对折射的感知不精准，所以不会造成不真实感。

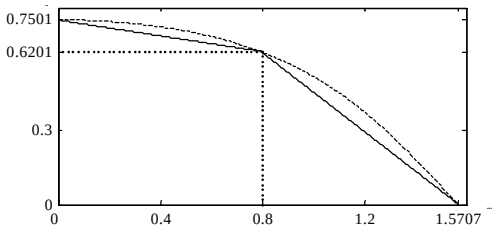


图3 缩放系数k的函数图

2.3 水面反射、折射及波动效果的实现

水面反射、折射及波动效果主要通过Cg程序来实现。

(1) 反射倒影的实现。获得反射纹理图后，在绘制水面时水面的颜色应该通过反射纹理图查找获取。由于该反射纹理图尺寸是当前绘制区窗口大小，因此在Cg程序的片元程序中，用当前窗口每个像素的坐标作为查找参数对反射纹理进行查询，所获取的颜色值即为水面倒影的颜色，将该颜色与水面的纹理颜色进行插值运算即可获得最终的倒影效果，如图4所示。Cg代码如下：

图 4 反射效果获取过程

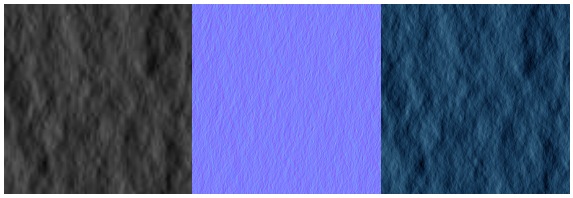
```
OUT.color = lerp(tex2D(waterTexture, IN.texcoord0 ),
                 texRECT(reflectTex, winPosition.xy),lerpScale);
```

其中waterTexture是水面自身纹理，reflectTex是反射纹理，winPosition是窗口坐标，lerpScale是插值权值。

(2) 折射效果的实现。由于折射场景获取进行了折射处理，因此绘制折射效果时的水面颜色通过折射纹理图查找即可，查找方法同反射。对水面纹理颜色和折射纹理颜色进行插值来获得输出的水面颜色。最终获得的折射效果如图5。注意：由于折射影响，图中白色参照物体产生了折弯现象，同时水深仿佛变浅了。

图 5 仅有折射效果的场景

(3) 水面凹凸纹理的实现。由于整个水面仅由2个三角图元构成，因此通过改变顶点高程来模拟水面的波动是不现实的，而凹凸纹理可以弥补这一缺陷。凹凸纹理贴图是Blinn于1978年提出的，该方法能以较低的几何细节来获得非常逼真的凹凸效果。对于实时凹凸贴图目前主要存在如下几种方法[11]：1)浮雕凹凸贴图；2)环境图凹凸贴图；3)点乘凹凸贴图。其中点乘凹凸帖图是通过高程纹理图计算每个顶点的法向量并单位化，然后将这些向量的x，y，z分量映射到[0，255]区间内，并用颜色值的r，g，b来代替，从而形成一个和高程图同样大小的位图，称为法向图[12]。利用每个顶点的法向与经过插值的光源向量进行点乘运算得到一个算子，然后利用该算子与模型真正的纹理（称为细节纹理）融合就会获得一个与光源位置相关的凹凸感表面。之所以有这样的结果是因为这种点乘运算正好和光照模型的漫反射分量计算方式相同。本文的凹凸纹理贴图是基于这种思想实现的。高程图实际上就是一个灰度图，每个像素的颜色值代表该像素对应顶点的高度值，如图6a所示。假设该图的像素个数是$P\times P$，其中$P=2^m$，m是正整数，用数组height[x][y]存储高程图(x，y)处的高程值，那么(x，y)处的两个正交切向量为



a b c

图 6 凹凸纹理的纹理图

$$\overrightarrow{T_1}=(1.0,\ 0.0,\ \text{height}[x+1,\ y]-\text{height}[x-1,\ y])$$

$$\overrightarrow{T_2}=(0.0,\ 1.0,\ \text{height}[x,\ y+1]-\text{height}[x,\ y-1])$$

则该点处的法向为$\overrightarrow{N_{xy}}=\overrightarrow{T_1}\times\overrightarrow{T_2}$，将该向量单位化并将各个分量映射到区间[0，255]，即为法向图在(x，y)处的颜色值。图6b是该方法获得的与图6a对应的法向图，图6c则是绘制水面时需要映射的细节纹理。Cg程序的顶点程序中需要一个uniform型参数，用于传入场景光源的位置。在该程序中，需要计算光源到顶点的向量——光源向量以备片元程序使用。片元程序有两个纹理参数，分别是细节纹理参数和法向图纹理参数。首先将传入的光源向量单位化，然后查询法向图纹理获得颜色并映射到[-1, 1]区间即为相应顶点的法向，把单位化的光源向量和顶点法向做点乘获得一个算子，用该算子和细节纹理查询获得的颜色乘积作为片元程序的输出颜色，便获得了水面的凹凸效果，如图7所示。

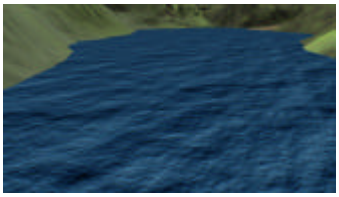


图 7 仅有凹凸效果的场景

(4) 水面波动效果的实现。水面波动由两部分组成：第一是动态凹凸纹理模拟波动的水面；第二是通过一定数学模型对反射折射纹理坐标进行扰动实现纹理扭曲，使得效果更贴近现实。

动态凹凸纹理的实现。水面凹凸纹理的实现过程中有两个uiniform型纹理参数——细节纹理和法向图纹理，这两个纹理均是在Cg程序外、OpenGL程序中设定的，设定纹理参数的Cg语言核心函数是void cgGLSetTextureParameter (CGparameter param, GLuint texobj)，该函数第一个参数用于指定将要被设定的纹理，第二个参数为该纹理指定具体的OpenGL纹理目标。动态凹凸纹理就是在每一帧开始绘制之初指定变化的OpenGL纹理目标，当OpenGL应用程序连续绘制时就会出现水面的动态效果。程序中需要指定的是一系列的水面凹凸纹理的细节纹理和法向图纹理。细节纹理可以通过诸如3DS MAX等动画软件渲染获得，需要的纹理幅数不宜过少，否则动画衔接不自然。

纹理扭曲的实现。实现动态凹凸纹理后，水面的波动较为逼真，但是反射和折射纹理是静止的，致使整个场景观察起来比较呆板。现实中由于水面波动会造成水面倒影扭曲现象。要产生这种效果最直接的办法是用某种方法扰动纹理映射坐标，由于水面仅由两个三角单元构成，共有4个纹理坐

标对，因此这种方法不可能在OpenGL中获得，Cg的出现为其提供了可能。Cg的片元程序是对光栅化了的每个片元进行操作，因此可以通过一定数学模型对传入的反射、折射纹理坐标进行扰动，从而获得波动扭曲效果。如果获得的某种数学模型不与动态凹凸纹理关联，则实现的效果非常不协调，为此对反射折射纹理坐标与凹凸纹理法向图的查询值进行如下关联

$$\begin{cases} T_{x1} = T_{x0} + C_o \cdot T_{x0} \cdot N_x \\ T_{y1} = T_{y0} + C_o \cdot T_{y0} \cdot N_y \end{cases} \tag{9}$$

其中T_{x1}、T_{y1}是扰动后的反射折射纹理坐标分量，T_{x0}、T_{y0}是未扰动的反射折射纹理坐标分量，N_x、N_y是对法向图查询获得的顶点法向分量，C_o是扩展系数，用于控制扭曲程度。由于较远的场景在屏幕上所占的像素数较少，如果这些场景和近处景物以同样的比例进行扭曲，则远处的扭曲不易察觉，式(9)中T_{x0}、T_{y0}作为因子与水面顶点法向相乘很好地消除了这种现象。最终的波动扭曲效果如图8所示。

图 8　经过波动扭曲处理的场景

3　太阳波光及Caustics效果的实现

现实中，由于自然光线的反复折射、反射使得水面光彩斑斓，如阳光在水面上造成的波光粼粼的效果以及水面的Caustics效果。本文对这两种现象的模拟方法如下所述。

(1) 波光效果模拟。波光效果模拟的实现利用了投影纹理技术[13]。普通的纹理映射必须为需要映射纹理的表面的每个顶点指定纹理坐标，图形硬件根据指定的纹理坐标对纹理进行查询、插值获得其它点的颜色值。投影纹理则不然，它并不需要为映射纹理的表面指定纹理坐标，纹理坐标是在程序中运算出来的，可以直观的认为需要映射的纹理是幻灯片，经过幻灯机光线照射而将其投影到景物表面。波光效果是预先制作一个需要的纹理，将这个纹理作为“幻灯片”，场景中的太阳作为“幻灯机”并将幻灯片投射到水面。同样应用2.3节的原理对投影纹理坐标进行扭曲，从而实现波光效果。实现投影纹理的步骤：

① 在OpenGL应用程序中设置投影纹理矩阵并传入Cg程序中。

② 为Cg的片元程序指定投影纹理。

③ 在顶点程序中计算投影纹理坐标。该顶点程序有一个uniform型变量接受①中传入的纹理矩阵。

④ 在片元程序中对投影波光纹理进行查询，查询参数使用顶点程序计算获得的投影纹理坐标的扰动值，扰动方程如下

$$\begin{cases} T_{px1} = T_{px0} + C_{po} \cdot N_x \\ T_{py1} = T_{py0} + C_{po} \cdot N_y \end{cases} \tag{10}$$

其中T_{px1}、T_{py1}是扰动后的投影波光纹理坐标分量，T_{px0}、T_{py0}是未扰动的投影波光纹理坐标分量，N_x、N_y是对法向图查询获得的法向分量，C_{po}是扩展系数，用于控制投影波光纹理的扭曲程度。按照扰动后的纹理坐标对投影波光纹理进行查询获得输出颜色值以备与其它效果颜色值求和运算。

(2) Caustics效果的模拟。该效果也是通过动态纹理来实现的，其原理同2.3节动态凹凸纹理，同样需要预先渲染一定帧数的Caustics纹理，图9是其中的1幅。为了使得效果更为贴近现实，同样要对Caustics纹理的查询坐标进行扰动，具体扰动模型如下

$$\begin{cases} T_{cx1} = T_{cx0} + C_{co} \cdot N_x \\ T_{cy1} = T_{cy0} + C_{co} \cdot N_y \end{cases} \tag{11}$$

其中T_{cx1}、T_{cy1}是扰动后的Caustics纹理坐标分量，T_{cx0}、T_{cy0}是未扰动的Caustics纹理坐标分量，N_x、N_y是对法向图查询获得的法向分量，C_{co}是扩展系数，用于控制Caustics纹理的扭曲程度。

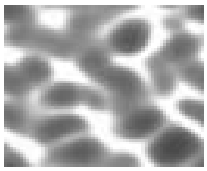


图 9　Caustics 效果的纹理

4　实验结果及结论

利用本文介绍的方法实现了对水面环境的实时模拟。实验硬件平台为CPU Athlon 2200+、内存512MB、NVidia3D图形加速卡（Geforce FX5600 ,128MB）、操作系统Windows 2000。开发工具为Visual C ++ 6.0、三维图形标准库OpenGL1.4、Cg库为Cg Toolkit 1.3。图10是最终实现的综合效果，实验结果数据见表2。由表可知：在该场景中，水面仅由2个三角形图元构成；随着场景复杂度的增加，水面对整个场景绘制刷新率的影响减小；场景复杂度增加对于绘制刷新率影响较大：以第一、二行数据为例，如果场景三角图元个数由13912增至27824，所增加的一万多图元用于构建水面（构建较好效果的水面这些图元远远不够），即使不考虑计算水面网格高程的影响，刷新率降为46帧/秒，而采用本文方法绘制水面，刷新率为55帧/秒，可见效果非常明显，而且随着场景复杂度的增加其优势更为突出。前述相关文献对水面的构建均采用了多格点网格，虽然有文献利用了LOD技术进行数据简化，但是三角图元个数仍十分可观，为了保证模拟效果的真实感，其数目不能过分减少，因此在复杂场景（包括地形、人物、建筑、植物等）中使用时有一定弊病，而本文的水面仅由2个三角形单元构建，综合利用各种纹理技术获得的水面效果亦非常逼真，因此对于复杂场景绘制十分有利并显示优势。

表 2　试验结果数据

未进行水面绘制		进行水面绘制	
场景三角图元个数（个）	刷新率（帧/秒）	场景三角图元个数（个）	刷新率（帧/秒）
13912	61	13914	55
27824	46	27826	45
41736	39	41738	39

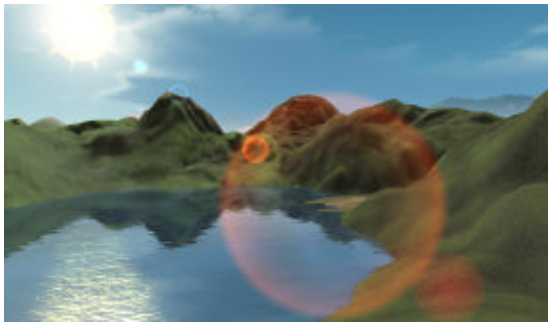


图 10 综合效果的场景图

5 结论

实时水面的模拟始终是计算机图形学和虚拟现实技术领域研究热点之一。本文摒弃了传统的利用高分辨率网格构建水面，并加载一定数学模型控制网格点高度来模拟水面的方法，而采用了纯纹理的方式，并结合最新的Cg图形硬件开发语言，在OpenGL中实现了一个实时的水面环境，该水面虽然仅有2个三角单元，但是其效果能大大增强虚拟现实系统的沉浸感。今后拟在水面以下通过雾的加入来模拟水下的环境，从而形成一个水环境的整体效果。

参考文献:

[1] Nick Foster, Dimitri Metaxas. Realistic Animation of Liquids [J]. Graphical Models and Image Processing (S1077-3169), 1996, 58(5): 471-483.

[2] Kei Iwasaki, Yoshinori Dobashi, Tomoyuki Nishita. A Fast Rendering Method for Refractive and Reflective Caustics Due to Water Surfaces [J]. Computer Graphics Forum (S0167-7055), 2003, 22(3): 601-609.

[3] Kei Nakano. Water Surface Using Particle System [EB/OL]. http://graphics.sfc.keio.ac.jp/papers/2002a/nakano2002.pdf, 2002.

[4] Jerry Tessendorf. Simulating Ocean Water [Z]. SIGGRAPH 2001 Course notes, 2001.

[5] Vladimir Belyaev. Real-time simulation of water surface [EB/OL]. www.graphicon.ru/2003/Proceedings/Technical/paper316.pdf, 2003.

[6] Damien Hinsinger, Fabrice Neyret, Marie-Paule Cani. Interactive Animation of Ocean Waves [C]// Symposium on Computer Animation, 2002: 161-166.

[7] 吴斌, 段海波, 薛凤武. OpenGL编程权威指南[M]. 北京: 中国电力出版社, 2001.

[8] 洪伟, 刘亚妮, 李骑, 等. Cg教程——可编程实时图形权威指南[M]. 北京: 人民邮电出版社, 2004.

[9] K Iwasaki, Y Dobashi, T Nishita. Efficient Rendering of Optical Effects within Water Using Graphics Hardware [C]//Pacific Graphics 2001, 374-383.

[10] Jensen L. Deep-Water Animation and Rendering [EB/OL]. http://www.gamasutra.com/gdce/jensen/jensen_01.htm, 2001.

[11] 普建涛. 实时计算机图形学（第二版）[M]. 北京: 北京大学出版社, 2004.

[12] Wolfgang Heidrich, Hans-Peter Seidel. Realistic, hardware accelerated shading and lighting [C]//SIGGRAPH99 Proceedings, 1999: 171-178.

[13] Mark Segal, Carl Korobkin, Rolf van Widenfelt, *etal*. Fast shadows and lighting effects using texture mapping [J]. Computer Graphics (ACM) (S0097-8930), 1992, 26(2): 249-252.

(上接第 337 页)

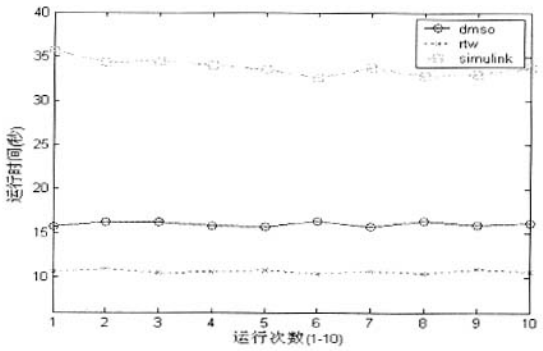


图 7 效率对比

联邦由于在图形环境下运行，负载较重，因此运行时间最长。而 rtw 联邦比 dmso 联邦运行速度快，是因为前者生成的代码经过了优化，后者采用面向对象的编程方法，增加了额外的负载。

4 结论

使用 RTI 接口模块开发应用于 HLA 分布式交互仿真的 Simulink 模型，从开发效率和运行效率上都有较大优势。随着 MATLAB Release14 的推出，用 M 语言编写的函数也可以用在 Simulink 模型中，拓展了 RTI 接口模块的适用范围。本文介绍的 RTI 接口模块是在 MATLAB6.5 下开发的，能与 KD-RTI，pRTI 互连，已成功应用于某通信仿真项目。

参考文献:

[1] 田新华, 冯润明, 翁干飞, 等. 一种将 Simulink/Stateflow 模型改造成 HLA 成员的方法[J]. 系统仿真学报, 2002, 14(7): 883-886.

[2] Sven Pawletta, Wolfgang Drewelow, Thorsten Pawletta. HLA-Based Simulation within an Interactive Engineering Environment [C]// Fourth IEEE International Workshop on Distributed Simulation and Real- Time Applications, 2000, 97-102.

[3] 郭斌, 熊光楞, 陈晓波, 等. MATLAB 与 HLA/RTI 通用适配器研究与实现[J]. 系统仿真学报, 2004, 16(6): 1275-1279.

[4] The Mathworks, Inc. Writing S-Functions, 2003 [Z].

[5] The Mathworks, Inc. Using Simulink, 2003 [Z].

[6] The Mathworks, Inc. Real-Time Workshop User's Guide, 2003 [Z].

[7] U.S. Department of Defense. High-Level Architecture Object Model Template Specification, Version 1.3 [S]. 1998.