文章编号:1671-4598(2020)04-0205-06 DOI:10.16526/j.cnki.11—4762/tp.2020.04.042 中图分类号:TP311 文献标识码:A

基于 Web 的时变体数据的体绘制方法

张 桢[1],吴亚东[1,2]
(1. 西南科技大学 计算机科学与技术学院,四川 绵阳 621000;
2. 四川轻化工大学,四川 宜宾 644005)

摘要:传统 Web 体绘制方法主要集中在利用服务器端进行预处理和绘制任务,浏览器端仅用于呈现绘制结果,这样会造成服务器负载过高,同时,当绘制参数发生更改时,必须向服务器请求新的绘制结果,这样也易受网络延迟的影响;为了解决以上问题,实现在浏览器本地进行体绘制和交互,文章提出一种基于 WebGL 的体绘制方法,以时变体数据为例,在浏览器端实现光线投射体绘制算法;同时,为了提升绘制效率和减少内存占用,文章基于维度压缩方法,优化时变体数据的预处理过程;最后,文章设计了 Web 体绘制系统,引入暴风时变数据集以验证方法的有效性,结果表明,该方法能够在浏览器本地对时变体数据进行体绘制,绘制时间在 50 ms 以下,帧速率可达到 50 FPS 以上,同时支持实时交互,并且当绘制参数发生更改时,系统能够直接在浏览器端进行重新绘制。

关键词:体绘制;WebGL;时变体数据;维度压缩

Volume Rendering of Time—Varing Volume Data in Web

Zhang Zhen[1], Wu Yadong[1,2]
(1. School of Computer Science and Technology, Southwest University of Science and Technology, Mianyang 621000, China;
2. Sichuan University of Science & Engineering, Yibin 644005, China)

Abstract: Traditional web volume rendering methods mainly focus on use of server for preprocessing and rendering, the browers is only used to be display the rendering result. This will cause high server load. Moreover, request for new rendering result must be sent to server when rendering parameters change, which is also susceptible to network latency. To solve the above problems, realizing volume rendering and interaction in browsers, a WebGL—based volume rendering method is proposed in this paper, which implemented ray—casting algorithm, taking time—varing volume data as example, in browers. In addition, for reducing rendering time and memory usage, the preprocessing of volume data is optimized based on dimension compression method. Finally, this paper designed a web volume rendering system, and introduced Isabel Hurricane data sets to verify the effectiveness of the methods. The results show that this method can realize volume rendering of time—varing data in the browser. The rendering time is less than 50ms, and the frame rate can reach more than 50FPS. Furthermore, it supports real—time interaction and redraw directly in the browser when the rendering parameters change.

Keywords: volume rendering; WebGL; time—varing volume data; dimension compression

0 引言

时变体数据是随时间步长变化的单步态体数据的集合,也是科学数据的重要分支,分析时变体数据能够获取时变体数据的内在关联,并获悉数据随时间变化的发展趋势。而体绘制则是针对时变体数据的有力分析方法,体绘制是将三维数据集显示为二维投影的技术[1],相比于面绘制[2],体绘制能够直观地呈现出体数据的三维模型,并能够展示出数据的整体结构特点[3]。然而,体绘制算法计算量庞大,目前的体绘制成像系统多集中在本地运行,采用 C/S(客户端/服务器)架构,用户需要在计算机安装特定的应用软件,并利用本地的图形硬件实现体绘制算法,提升体绘制效率,因此,C/S 架构无法便捷的进行访问,也难以实现跨平台[4]。

近年来,基于 B/S(浏览器/服务器)架构的科学数据绘制成像系统逐渐兴起。B/S 架构具有易访问、跨平台等特点[5],同时 WebGL 的出现更使得浏览器能够利用本地 GPU 进行高性能图形渲染[6]。因此,基于 B/S 架构硬件依赖小,同时兼顾跨平台访问和高性能的特点,本文提出了一种基于 B/S 架构的时变体数据的体绘制系统,该系统通过 WebGL 标准实现了光线投射体绘制算法,在浏览器端实现对体数据的体绘制及显示,并支持实时交互。同时,通过优化体数据的预处理过程,缩短了单步态体数据的绘制时间,减少单步态体数据绘制所占用的内存空间,使得浏览器端能够将时变体数据集合全部载入,并逐步绘制显示,以动画的形式呈现出时变体数据的变化效果。最后,本文引入 Isabel 飓风数据集进行案例测试,验证本文方法的可行性。

1 国内研究现状

体绘制是对科学数据进行三维重建的方法之一,它直

收稿日期:2020-01-03; **修回日期**:2020-03-02。
项目基金:国家自然科学基金(61802320,61872304);国防基础科研计划(JCKY2017404C004)。
作者简介:张 桢(1993-),男,四川阆中人,硕士研究生,主要从事科学可视化方向的研究。
通讯作者:吴亚东(1974-),男,河南周口人,博士,教授,主要从事可视化与可视分析、人际交互和虚拟现实方向的研究。

接由三维数据场生成在屏幕上显示的二维图像[7]。体绘制技术可以展示出科学数据的内部结构，在描述定形物体上具有明显的优势，因而成为了科学数据可视化最为广泛采用的方法之一。

近年来，在 Web 端进行体数据的体绘制得到了长足的发展，无需借助第三方的浏览器插件，纯面向 Web 的可视化系统逐渐成为未来的发展趋势[8]。但在早期工作中，体绘制计算量庞大，且硬件性能较低，B/S 架构的可视化系统主要集中在利用服务器端对数据进行处理并生成绘制结果，而浏览器端仅用于绘制结果的呈现[9]。高鹏[10]等利用 VTK（Visualization Toolkit）可视化工具包，将数据在服务器端进行处理渲染，并生成二维图片，浏览器通过加载二维图片从而实现“伪 3D”的图像显示，用户在浏览器端实时生成交互请求，这些请求包括视角变换，缩放等，服务器根据交互请求进行绘制并生成新的二维图片。雷辉等[11]实现了基于 HTML5 的医学可视化系统，服务器端利用 DCMTK 模块对医学数据进行解析，并绘制出结果，浏览器端采用图像载入模块加载绘制结果并呈现。侯晓帅等[12]提出的 B/S 架构解决方案采用在服务器端进行图像分析，并利用服务器的 GPU 加速技术绘制图像，并将结果发送至浏览器，浏览器端采用 HTML5 和 JavaScript 技术开发显示模块用于呈现绘制结果。而随着科学体数据的数据量日益庞大，单个服务器已无法满足渲染要求，Robin M. Weiss[13]等使用集群服务器进行数据渲染，集群服务器将多个渲染引擎连在一起，为体数据的绘制提供强大的渲染能力。

服务器端渲染出图像结果并发送至浏览器的解决方案可以利用服务器的高性能，但服务器负载会随着并发用户的增长而提升，并发用户过多会导致服务器延迟过高，运行缓慢，甚至崩溃。而 WebGL 的出现使得面向浏览器的体绘制方法发生了改变，它能够直接调用本地计算机的 GPU 在浏览器展示 3D 模型，借助 WebGL，可以直接在浏览器端对数据进行体绘制。Daniel Haehn 等[14]开发了 Slice：Drop 可视化系统，系统采用 WebGL 在 HTML5 的 Canvas 元素中进行 3D 图像的绘制，并可以进行部分的交互，但该系统没有实现直接体绘制的算法，比如光线投射算法[15]。

基于上述工作，目前基于 B/S 架构的科学体数据的体绘制主要面临着以下问题：

1）体绘制算法实现多在服务器端进行，服务器需承担数据预处理和绘制，绘制结果以二维图片的形式传至浏览器端，服务器的负载会随并发用户数量的上升而增长；

2）浏览器以二维图片显示绘制结果，制约了交互性能，浏览器需将交互请求发送至服务器并重新绘制，得到新的结果，交互性能易受网络带宽和服务器负载影响，造成延时过高，等待时间过长甚至无响应等问题；

3）在浏览器端利用 WebGL 进行体绘制时，需将数据载入 GPU，时变体数据具有多个单步态数据，全部载入 GPU 会导致内存占用过高，WebGL 程序会因此崩溃，无法有效的对时变体数据的所有步态进行实时的绘制和呈现。

针对上述问题，本文做了以下工作：

1）基于 WebGL，实现了 Web 下的光线投射体绘制算法，浏览器利用本地 GPU 对数据进行体绘制并呈现，并支持本地的实时交互；

2）基于维度压缩对原始体数据的预处理过程进行优化，缩小了原始体数据的体量，并减少了绘制时间以及 GPU 内存占用，使得本地 GPU 能够载入时变体数据的更多步态。

1.1 本文贡献

在 Web 端采用 WebGL 实现了光线投射体绘制算法，能够利用本地 GPU 对体数据进行体绘制，并且支持实时交互；

基于维度压缩优化了数据预处理过程，使得 Web 页面能够更快速、更流畅地绘制科学数据，并且能够同时载入时变体数据的更多步态，便于观察时变体数据的整体变化趋势。

2 时变体数据的 Web 体绘制

光线投射算法是体绘制常用算法之一。通过投射一条光线穿过原始 3D 数据体，并在穿入点和传出点之间进行等间隔均匀采样，通过传递函数计算采样点的颜色和不透明度，将采样点向前合成，合成结果即是屏幕上一个可见像素点。该过程由图 1 所示。通过投射多条光线，所得到的像素点最终在屏幕上呈现出绘制结果。

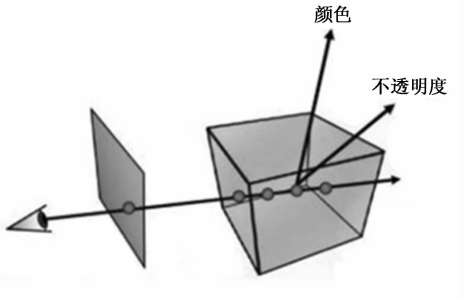


图 1 光线投射算法示意图

在 Web 下实现光线投射算法，需要先对原始数据进行预处理操作。

2.1 基于维度压缩的体数据预处理

科学体数据是 3D 形式，WebGL 仅支持 2D 纹理，因此需对原始的体数据进行预处理。本文系统将原始的 3D 体数据转换为 2D 纹理，转换示意图如图 2 所示，该数据为人脚部位的 CT 扫描。本文方法将 Z 维度拆分，并将 Z 切片依次平铺到二维图上，最终形成图 2 所示的二维图集。

生成 2D 图集的过程主要包括：原始数据归一化、转换原始数据类型、生成 2D 图集。为提升绘制效率及缩小内存占用，本文在预处理过程中引入维度压缩方法，以下将介绍基于维度压缩的原始体数据的预处理过程。

2.1.1 原始体数据的归一化

科学数据以采集的真实数值存储，在生成 2D 纹理过程中，需要将体数据的值赋给 RGBA 颜色分量（取值范围：0

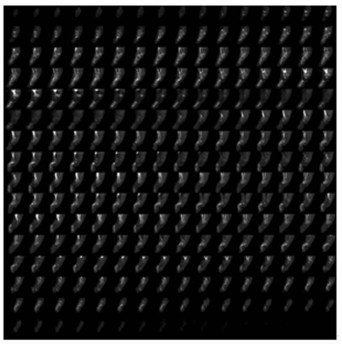


图 2　人脚 CT 扫描数据转换而成的二维纹理图集

～255），因此，首先需将体数据的值归一化到（0～255）区间。获取原始体数据中体素值的最大值 Max 和最小值 Min，则对于原始体数据中每一个体素值 V，归一化过程可由公式（1）～（2）表示：

$$K = (255 - 0)/(Max - Min) \tag{1}$$

$$V = 0 + K * (Array\_A - Min) \tag{2}$$

2.1.2　转换原始数据类型

科学体数据通常使用 float 类型存储，而 Unsigned Char 类型相比于 float，所占空间可节省四分之三，这将减轻服务器的数据存储负担，提升数据的载入速度，针对于大规模的时变体数据，通过转换数据类型，将降低存储用量，该过程伪代码如下所示：

```
begin:
#将原始数据读入数组 Array_source
read source data into Array_source;
#创建新数组 Array_new
create new char array Array_new;
for each i in Array_source
{
#将 Array_source 的值通过格式转换,存入 Array_new
Array_new[i] =
(unsigned char)Array_source;
}
End
```

2.1.3　生成 2D 纹理

将原始体数据的 Z 维度拆分为切片，将切片依次排列在一张图像上，即可生成 2D 切片图像集，2D 切片图像集上每一个像素点的 RGBA 分量值都由原始体数据对应体素值（已归一化到（0～255）区间）赋予。浏览器将此 2D 切片图像集作为纹理载入 GPU，便可进行光线投射体绘制。生成 2D 纹理的过程如图 3 所示。该过程伪代码如下所示，生成 2D 纹理后，即可进行体绘制和呈现。

```
begin:
#原始体数据的三个维度
define width, height, depth;
#读取原始体数据至数组
data = readSourceData(filename);
#定义 PNG 图像行列各分布多少张切片
define slicePerAxis = int(sqrt(depth));
#指定 PNG 图像的宽度及高度
define imageWidth = width * slicePerAxis;
defineimageHeight = height * slicePerAxis;
#创建一张指定大小的透明 PNG 图片
Image=Image.new(imageWidth, imageHeight);
#将图片存入像素映射表 colormap
#根据数组修改其像素值
colormap = Image.load();
for(int z = 0; z < depth; z++)
for(int y = 0; y < height; y++)
for(int x = 0; x < width; x++)
{
index = x + y * width + z * width * height;
rgba = data[index];
#获取 PNG 图像上的 X 坐标
realPos X = x + (z % slicePerAxis) * width;
#获取 PNG 图像上的 Y 坐标
realPos Y = y + ((int)(z / slicePerAxis)) * height
#将体素值写入对应点的 RGBA 分量
colormap[realPos X, realPos Y] =
(rgba, rgba, rgba, rgba);
}
End
```

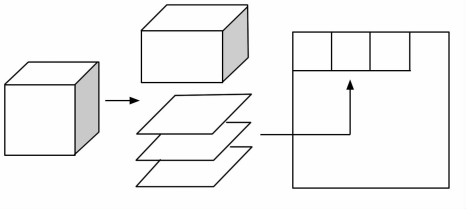


图 3　生成 2D 纹理，纹理上每一个像素点的 RGBA 像素值为（V，V，V，V），其中 V 是原始体数据中对应点的体素值

2.1.4　时变体数据维度压缩

WebGL 在绘制过程中将逐像素处理 2D 纹理，这是体绘制时间和内存的主要开销，而观察时变体数据整体变化过程，浏览器需载入全部单步态数据，要实现快速绘制和低内存占用，需考虑如下因素：1）缩短单步态数据的绘制时间；2）减少单步态数据的内存占用。因此，本文采用维度压缩方法，压缩原始体数据维度，减少 2D 纹理像素点数量。

本文采用跳跃取样方法进行维度压缩，设置步长 step，在（x，y，z）处取样后，跳跃至（x+step，y+step，z+step）处继续取样，并将取样点集合存储为新的体数据。通过跳跃取样，数据维度将缩小至原维度的 1/step，体素数量将减少至原数据的 $(1/step)^3$，极大提升了 GPU 处理纹理的速度，以及降低 GPU 内存的开销，维度压缩伪代码如下所示：

```
#原始体数据的三个维度
define width, height, depth;
#读取原始体数据至数组
source_data = readSourceData(filename);
```

```
    #创建新的体数据模板
    create new_data;
    #定义跳跃采样值的跳跃系数 step
    for(int z = 0; z < depth; z = z + step)
    for(int y = 0; y < height; y = z + step)
    for(int x = 0; x < width; x = z + step)
    {
    index = x + y * width + z * height * depth;
    #将采样值写入新数组
    write data[index] into new_data;
}
```

该方法会损失原始体数据的部分精度，但将显著提高绘制效率，经过案例测试表明，在维度压缩前后，绘制结果相差无几，因此进行维度压缩具有较好的可行性。

通过以上方法对原始体数据进行预处理操作，最终将得到 WebGL 体绘制所需的 2D 纹理，以下本文将介绍如何基于 WebGL，实现光线投射体绘制算法。

2.2 基于 WebGL 的光线投射体绘制

光线投射体绘制算法如图 2 所示，其过程可表述为生成包围盒、投射光线、取样、光线终止，以及采样点的合成。以下将分别介绍。

2.2.1 生成包围盒

包围盒是一个立方体，将 3D 体数据包围其中，采用包围盒取代体数据，则无需考虑体数据复杂形式，简化计算。设实际体数据维度为（$width$，$height$，$depth$），则据此创建一个维度同样为（$width$，$height$，$depth$）的长方体作为包围盒。

2.2.2 包围盒空间坐标与 2D 纹理的映射

实际体绘制中，WebGL 载入的是 2D 纹理，因此需要将包围盒空间坐标与 2D 纹理上的像素点坐标建立映射关系，以便投射光线即采样。根据表 2 生成 2D 纹理的伪代码，包围盒内部任意一点坐标（x_0，y_0，z_0）与 2D 纹理像素点实际坐标（X，Y）的对应关系为：

$$X = x_0 + (z_0 \% \ slicePerAxis) \times width \tag{3}$$

$$Y = y_0 + ((int)(z_0/slicePerAxis)) \times height \tag{4}$$

其中，$slicePerAxis$ 表示 2D 纹理上每一行所具有的切片数量。

2.2.3 投射光线并采样

设置视点，模拟观察者的眼睛，从视点出发投射一条光线，穿过包围盒，计算与包围盒穿入与穿出交点坐标，分别作为光线的起点和终点。自定义间隔值，并在光线上进行等间距的采样，该过程由图 4 所示。

设穿入点坐标为 $A(x,y,z)$，穿出点坐标为 $B(x',y',z')$，则可计算出光线向量 $\overrightarrow{AB}$，进一步可计算光线上各个采样点坐标，其过程如公式所示：

$$\overrightarrow{AB} = (x'-x, y'-y, z'-z) \tag{5}$$

$$(x_n, y_n, z_n) = (x,y,z) + \frac{n}{N}\overrightarrow{AB} \tag{6}$$

其中：N 表示该条光线上采样点的总个数，(x_n,y_n,z_n) 则表示第 n 个采样点。得到采样点坐标后，即可根据公式（3）～（4）采集 2D 纹理上实际对应的像素点。

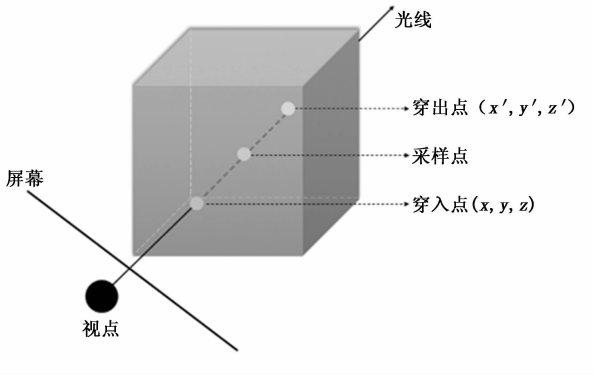


图 4 采样示意图

2.2.4 光线停止及采样点合成

通过传递函数，将采样点真实数值转换为颜色和不透明度，并由远及近（距离视点），开始合成同一光线上的采样点，并最终得到屏幕上的像素点。该过程如图 5 所示。这是一个迭代过程，当光线到达终点时，迭代过程便会停止，为缩减体绘制的时间，当其满足采样点合成过程中，累积不透明度到达 1 时，迭代过程可提前停止。由于当累积不透明度到达 1 时，后续的采样点将不会影响合成结果，因此迭代过程便可提前停止。由此，光线投射算法停止的条件为：

1）迭代过程到达光线终点；

2）迭代过程中，累积不透明度到达 1。

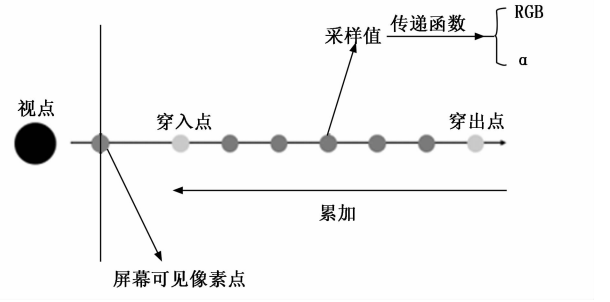


图 5 采样点的合成

当所有光线上的采样点向前合成为屏幕上的一个可见像素时，即可在屏幕上观察到最终的绘制结果，图 3 所示人脚部位的 2D 纹理图集在浏览器上体绘制的效果如图 6 所示。

2.2.5 实时交互

本文通过调用 JavaScript 监测鼠标及滚轮的变化，更改视点的位置，并重新进行体绘制过程，得到新的绘制结果。由于是在本地利用 GPU 的高性能进行绘制，交互的延时不受网络带宽和服务器负载的影响，交互响应时间快，实时交互性能优于传统的体绘制系统。

3 Web 体绘制系统的实现与测试

本文采用 B/S 架构，搭建了 Web 体绘制系统，并引入

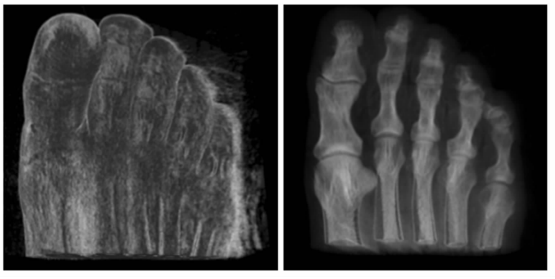


图 6　体绘制结果

Isabel 飓风数据集以验证本文方法的有效性，服务器端对原始数据进行预处理，生成 2D 纹理，浏览器端请求 2D 纹理，并载入 GPU，基于光线投射算法进行体绘制和呈现。系统整体架构如图 7 所示。系统设计了单文件查看模块和时变体数据观察模块。

1）单文件查看模块：单文件查看模块用于绘制和呈现时变体数据的单个步态等，该模块中，原始体数据未进行维度压缩，以观察体数据的全部细节，并支持观察者使用鼠标及滚轮对视图进行平移、旋转、缩放等交互操作。同时，系统还设计了传递函数面板，观察者可以使用不同的颜色编码和不透明度等控制选项，对体数据的绘制结果进行调节。

2）时变体数据观察模块：该模块需要将全部单步态数据载入 GPU，因此，该模块加载的原始体数据将进行维度压缩，以缩短单个步态数据的绘制时间和内存占用。通过设置间隔时间，WebGL 将依次动态的绘制并呈现每一个单步态数据，观察者由此能够观测到时变数据动态的变化过程，以分析时变体数据的整体变化趋势。

3.1　Isabel 飓风数据集

Isabel 飓风数据集是由美国国家大气研究中心所采集到的飓风信息，数据集的真实维度为 2 139 km * 2 004 km * 19.8 km，采集到的数据以 float 类型存入 500 * 500 * 100 的维度空间中，该数据集一共采集了 13 个变量，包括云水混合比、温度、压力等，每个变量共采集 48 个时间步长（每隔一小时采集一次，一共采集 48 个小时），单步态数据大小为 95 MB。

系统采用第一个变量（QCLOUD，云水混合比）作为案例测试，用以验证系统的有效性和可行性。

3.2　数据预处理

将 QCLOUD 的 48 个单步态体数据进行归一化操作和存储方式转换后，生成未经过维度压缩的 2D 纹理。然后对数据进行跳跃采样，本系统中设置跳跃步长 step 为 2，维度压缩后体数据体素数量从 2 500 万减少到了 3.125 万，再生成 2D 纹理。经过维度压缩和未经维度压缩生成的 2D 纹理对比如图 8 所示，图 8（a）未经维度压缩，图 8（b）经过维度压缩。

3.3　时变体数据的 Web 体绘制

浏览器端利用 WebGL 将数据载入 GPU 进行光线投射体绘制，本系统进行案例测试的硬件环境信息如表 1 所示。

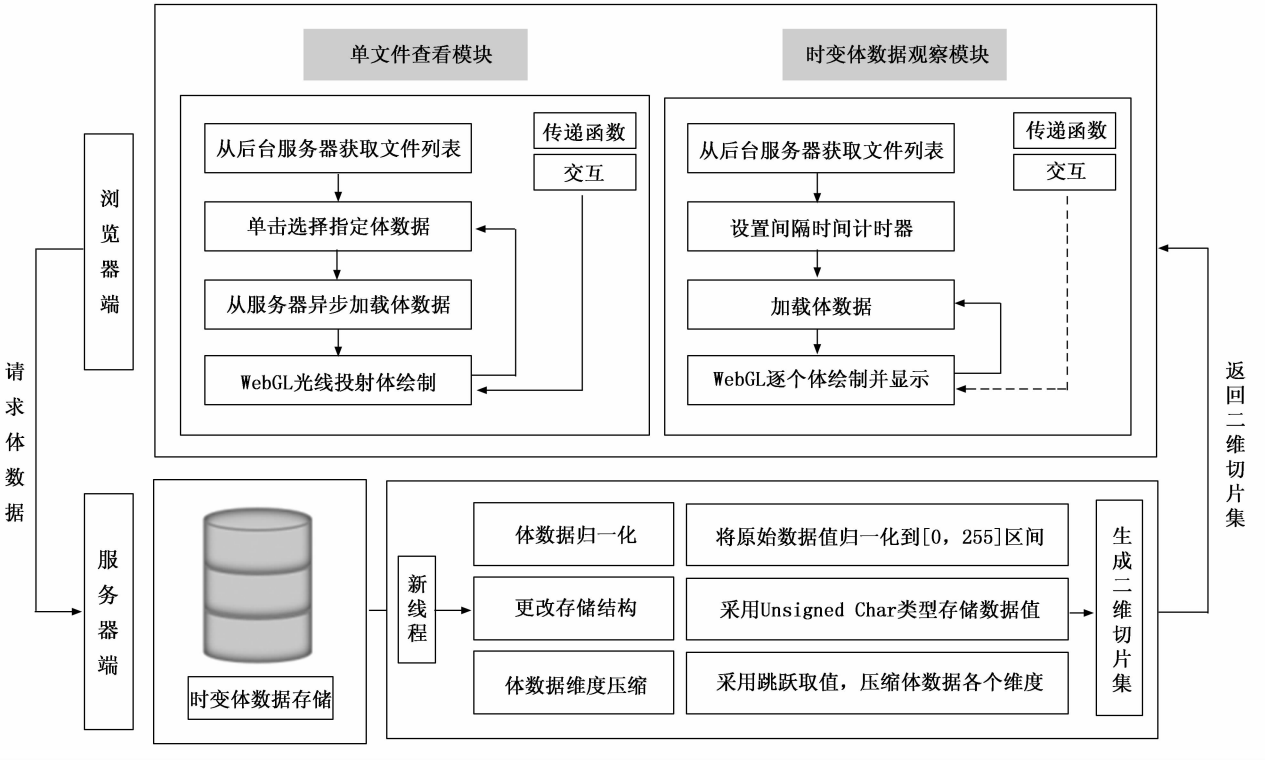
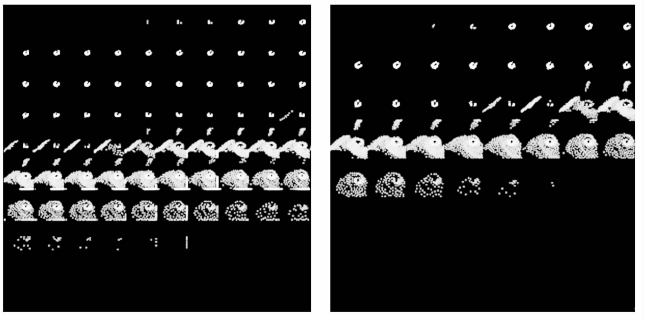


图 7　基于 B/S 架构的时变体数据体绘制系统框架图



(a) (b)

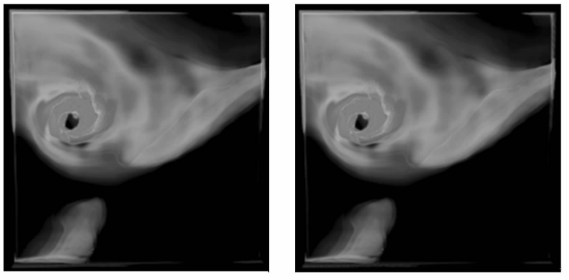
图 8 第一单步态体数据生成的二维图集

表 1 案例测试硬件环境信息

名称	型号
操作系统	Windows 10
显卡	NVIDIAGeForceGTX 970
显存	4G
CPU	Intel(R) Core(TM) i5 750@2.67GHz
内存	Kingston 16.0G
浏览器	GoogleChrome 76.0.3809.132

3.3.1 原始体数据维度压缩前后绘制对比

经过维度压缩之后，原始体数据会损失部分精度，但是将极大地提高体绘制的速度，如图 9 所示，图（a）为当跳跃步长 step 设置为 2 时进行维度压缩所得到的绘制结果，图（b）为不进行维度压缩时得到的绘制结果，两种方式的绘制时间和内存占用如表 2 所示。



(a)进行维度压缩 (b)不进行维度压缩

图 9 两种处理方式的绘制结果对比

表 2 维度压缩前后绘制时间对比

体数据	数据维度	绘制时间/ms	GPU 占用/MB
经过压缩	250 * 250 * 50	43	55
未经压缩	500 * 500 * 100	351	226

该案例可证明，进行维度压缩将显著提升绘制速度，同时对于绘制结果影响较小。

3.3.2 单文件查看模块

系统设计了单文件查看模块用以验证单个体数据的实时绘制和交互的性能，如图 10 所示，可在列表中选择需要查看的体数据。

通过鼠标，对数据进行旋转、缩放、平移等交互操作，以及修改传递函数参数等操作，视图能够实时地进行重绘，并显示新的渲染结果，修改传递函数的示例如图 11 所示。

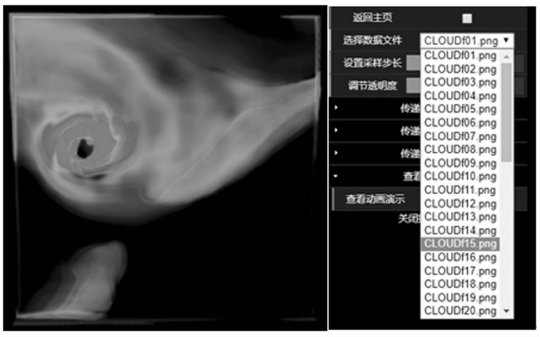


图 10 单文件查看模式的绘制效果

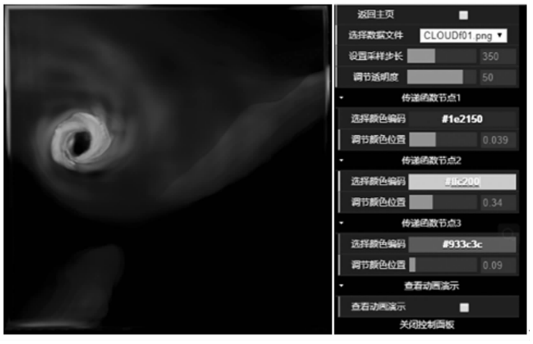


图 11 改传递函数的绘制效果

测试结果表明无论是交互，还是修改传递函数值，数据均能够实时地进行重新渲染，得到新的结果，且帧速率能够维持在 58～60 FPS。

3.3.3 时变体数据观察模块测试

该测试采用 QCLOUD 变量的 48 个步态数据，每个体数据都经过归一化，存储格式转换，以及采用步长为 2 的跳跃采样进行维度压缩，试验测试将 48 个步态的数据全部载入 GPU 进行预渲染，并设置显示间隔时间为 150 ms，测试结果表明，在浏览器端能够进行流畅的动画展示。测试相关结果如表 3 所示。

表 3 时变体数据观察模块测试参数

特征	测试结果
时间间隔	150 ms
GPU 内存占用	2.88 G
帧速率	52～58 FPS

4 结束语

本文提出了一种基于 B/S 架构的时变体数据的体绘制系统，该系统通过 WebGL 标准实现了光线投射体绘制算法，并支持实时交互。同时，基于维度压缩优化了体数据的预处理过程，减少了单步态数据的绘制时间和内存占用，使得浏览器端能够将时变体数据全部载入，并逐步绘制显示，以动画的形式呈现时变体数据的变化效果。最后系统引

（下转第 216 页）