Journal of System Simulation

Volume 32 | Issue 4 Article 12

4-16-2020

Lightmap-based GI Collaborative Rendering System for Web3D Application

Shao Wei
1. Tongji University, Shanghai 201814, China;;

Liu Chang
2. Nanchang Hangkong University, Nanchang 330063, China;

Jinyuan Jia
1. Tongji University, Shanghai 201814, China;;

Lightmap-based GI Collaborative Rendering System for Web3D Application

Abstract

Abstract: For the real-time dynamic lighting rendering in low-end web environment, *a view-independent* collaborative real-time rendering system is *presented*. The whole system consists of a cloud renderer which produces the global illumination, a web renderer which computes the local illumination, and a scalable collaboration solution which takes the *user behavior* into account to adjust the *range and frequency* of the lighting computation. The cloud renderer computes the lighting data into the Lightmap, streams the Lightmap to the web client. After computing the direct light, the web renderer achieves the global illumination information by Sampling the Lightmap, and shows the realistic scene. The results show that the system has a good performance both in the web and cloud renderer, and can achieve less computation in the lighting rendering and more realistic look in the final frame.

Keywords

lightmap, Web3D, collaborative rendering system, voxel based global illumination, view independent rendering

Recommended Citation

Shao Wei, Liu Chang, Jia Jinyuan. Lightmap-based GI Collaborative Rendering System for Web3D Application[J]. Journal of System Simulation, 2020, 32(4): 649-659.

第 32 卷第 4 期 系统仿真学报© Vol. 32 No. 4
2020 年 4 月 Journal of System Simulation Apr., 2020

基于光照贴图的 Web3D 全局光照协作式云渲染系统

邵威[1]，刘畅[2]，贾金原[1]
(1. 同济大学，上海 201814；2. 南昌航空大学，江西 南昌 330063)

摘要：提出一套*视点无关*的 Web3D 协作式全局光照渲染系统，以解决 Web3D 动态光照实时渲染问题。整套系统由负责全局光照的云渲染器和负责直接光照的 Web 渲染器构成，并把*用户行为*作为计算全局光照*范围和频率*重要参考依据。云渲染器在光照贴图中渲染光照数据，流式化传输到 Web 渲染器；Web 端在对场景进行直接光照渲染后，通过采样光照贴图信息以获得包含全局光照信息的强真实感场景渲染效果。经实验表明，本系统在云端和 Web 端均有优异的效率表现，并体现出渲染计算强度轻量化、渲染效果真实化。

关键词：光照贴图；Web3D；协作式渲染系统；体素化全局光照；视点无关渲染
中图分类号：TP391.9 文献标识码：A 文章编号：1004-731X (2020) 04-0649-11
DOI: 10.16182/j.issn1004731x.joss.19-VR0471

Lightmap-based GI Collaborative Rendering System for Web3D Application

Shao Wei[1], *Liu Chang*[2], *Jia Jinyuan*[1]
(1. Tongji University, Shanghai 201814, China; 2. Nanchang Hangkong University, Nanchang 330063, China)

Abstract: For the real-time dynamic lighting rendering in low-end web environment, *a view-independent* collaborative real-time rendering system is *presented*. The whole system consists of a cloud renderer which produces the global illumination, a web renderer which computes the local illumination, and a scalable collaboration solution which takes the *user behavior* into account to adjust the *range and frequency* of the lighting computation. The cloud renderer computes the lighting data into the Lightmap, streams the Lightmap to the web client. After computing the direct light, the web renderer achieves the global illumination information by Sampling the Lightmap, and shows the realistic scene. The results show that the system has a good performance both in the web and cloud renderer, and can achieve less computation in the lighting rendering and more realistic look in the final frame.

Keywords: lightmap; Web3D; collaborative rendering system; voxel based global illumination; view independent rendering

引言

随着 Web3D 技术的不断发展，越来越多的网站应用 Web3D 技术来展示商品、游戏娱乐，丰富了网站的观赏性和交互性。作为 Web3D 应用的核心 API 库，WebGL 被几乎所有的网页浏览器所支持。然而，为嵌入网页之中，WebGL 不得不牺牲一定的渲染能力，目前主流浏览器仍然只支持基于 OpenGL ES 3.0 的 WebGL 2.0 标准。这意味着一些三维渲染中经常用到的主流技术如计算着色器、几何着色器以及曲面细分等在 Web 中无法使用，这很大程度上限制了 Web3D 虚拟场景的渲染效果。

收稿日期：2019-08-30 修回日期：2019-12-05；
作者简介：邵威(1994-)，男，上海，硕士生，研究方向为计算机图形学；刘畅(1983-)，男，江西南昌，博士，研究方向为虚拟现实，计算机图形学；贾金原(1963-)，男，山东乐陵，博士，教授，研究方向为虚拟现实、计算机图形学。

云烘焙[1-2]是协同式渲染机制中的一种，它是为了弥补 Web3D 应用自身渲染能力不足而提出的一套专门针对 Web3D 应用的动态场景光影渲染机制。该机制对 Web3D 场景中的光影渲染进行前后端协同式渲染，其中计算复杂较高的光影渲染工作被放于云服务器后端，如：软阴影、全局光照等；而计算复杂度较低的光影渲染工作被放于 Web 前端，如：直接光照渲染、基于屏幕的环境遮挡等；最后使前后端的渲染结果在 Web 中进行“混合”后输出。然而包括云烘焙[1-2]和云游戏[3]在内的大部分协同式云渲染系统都是视点相关的。受传输延时的影响，前后端的渲染内容是不一致的，这导致视点相关的云渲染系统在处理前后端渲染帧“混合”时必然出现“孔”失真和重影的伪像，这就需要系统耗费一定资源予以处理。

针对上述问题，提出一种在贴图空间进行全局光照计算的视点无关的协作式云渲染系统。本系统使用体素化全局光照算法，将光照数据直接存储到光照贴图中。然后通过 H.264 视频流的方式将光照信息传递给前端。该方法充分利用了光照贴图视点无关的特性，前端仅需要在 Shader 阶段采样贴图即可获取光照数据，根据 UV 坐标着色到物体表面。本系统解决了物体不重叠的问题。

1 相关工作

1.1 远程渲染

在渲染模式上，远程渲染指的是把复杂度较高的三维渲染任务放在远程服务器进行，充分利用服务器强大的硬件渲染能力，快速且高效的获取客户所需求的渲染效果。基于这种渲染模式的系统把渲染任务完全放在服务器，并以图像帧和视频流的形式向 Web 客户端传输数据，Web 客户端只是用来接收和显示结果而没有参与任何渲染任务，这使得 Web 客户端的硬件需求得以降低[3]。由于该系统把渲染任务完全部署在服务器端，模型数据只需要在服务器端存储并渲染，用户只能通过 Web 客户端浏览这些模型的渲染效果而无法直接访问这些模型数据，这非常有利于这些模型数据的保护[4]。然而，此类系统的服务器设备具有计算密集型特点，这严重的限制了整个系统的可扩展性；另外，无法避免的交互延时也严重影响了用户体验质 量[3]。Shu 等提出高质量低延迟的远程渲染可视化系统[5]以及 Maamar 提出的面向移动端的远程渲染可视化系统都属于此类系统[6]。

1.2 协作式渲染

协作式渲染是一种基于 Web 前端和云后端协同操作的新型渲染模式。该模式下的渲染系统基于前后端硬件设备的性能把系统中的渲染任务进行分摊，并让 Web 客户端深入参于渲染流程。因此，此类系统通常把计算强度高的渲染任务放在云服务器，而把计算强度较低的渲染任务放在 Web 客户端；这样既可以释放 Web 客户端的渲染能力也可以降低云服务器的渲染负担。此类系统在动态场景的光照渲染研究中被广泛应用，Crassin 提出的 Cloud Light 系统首次提出了光照渲染任务的分摊[1]，渲染光照任务 GPU 与最终帧着色 GPU 直接传输数据协作。Chang 提出的 Cloud Baking 系统在 Web 客户端实现了光照的实时协同式渲染[2,7]，其中 Web 客户端实现了直接光照渲染，而云服务器端实现了全局光照渲染，并将数据保存在了视点相关的光图(GIMap)中。之后为了优化和调度 GIMap，提出了光图树[8]机制。Keith Bugeja 提出的 Remote Asynchronous Indirect Lighting (RAIL)实现了视点无关的协同式光照渲染[9]。Shading Atlas Streaming(SAS)[10]将所有的着色计算全部放到云端完成，并使用一种类似 Virtual Texture 的 Shading Atlas 机制进行存储，前端仅需要光栅化模型及采样纹理即可，代价是云端需要花费大量资源计算每帧可见几何模型的 Shading Atlas。协作式渲染各方法对比见表 1。

表 1 协作式渲染相关方法对比表
Tab. 1 Collaborate Rendering Approaches Comparison

方法	云渲染器		网络传输			客户端	
	全局光照算法	计算量	传输数据	数据量	频度	云端光照合成算法	计算量
CloudBaking	Voxel Based	中	视点相关 GIMap	中	每帧	3D image wraping	中
光图树	N/A	小(预计算)	视点相关 GIMap	中	每帧	3D texture wraping	中
RAIL	VPL Based	中	视点无关的存储结构	中	当场景变化时	辐射度重建和估计	大
SAS	N/A	大	Streaming Atlas+可见几何信息	大	每帧	Streaming Atlas 采样	小
我们的方法	Voxel Based	中	视点无关 Lightmap	中	当场景变化时	Lightmap 采样	小

1.3 Lightmap 技术

虚拟场景中光照信息通常会以像素的形式被保存于图片之中，图片称之为 Lightmap。Lightmap 记录的数据通常以整个虚拟场景为对象，为了获得高度真实感的全局光照效果，往往需要花费数小时进行离线渲染，且当场景中的光照和模型等信息发生变化时，Lightmap 就需要全部或者部分重新计算。在这个过程中，路径跟踪[11]，辐射度[12]等离线渲染技术被深度使用。

为了让 Lightmap 实时产生以便使其支持动态光照渲染或动态场景渲染，学者们通过以下 2 个方面进行改进：

(1) 加速全局光照烘培速度：Keller 提出的 Instant Radiosity[11]通过预计算光源部分路径，生成虚拟点光源 Virtual Point Light (VPL)，使路径跟踪转换为一个多光源问题(Many Light Global Illumination)，使交互式全局光照成为可能。之后大量工作在此基础上进行改进。Dachsbacher 提出的 Reflective Shadow Mapping (RSM)[12]改进了 VPL 生成算法，通过在光源处对场景进行光栅化，一次计算可生成上千个 VPL。Luksch 使用 VPL 进行 KD 树聚类成 Virtual Polygon Lights[13]，大大降低了光源数量，使 Lightmap 计算可以在数分钟甚至数秒内完成。此外，Voxel Based GI[14]技术将场景表述为体素，通过圆锥体跟踪进行光照计算，同时简化了可见性计算，进一步降低了计算量，使全局光照可以实时计算。

(2) 拓宽 Lightmap 纹素存储内容：Sloan 提出的 Precomputed Radiance Transfer (PRT)[15]技术通过预计算，将辐射度生成为球谐函数的系数，存放于 Lightmap 纹素的 RGB 通道中，以获得低频漫反射和折射的全局光照。Ozturk 提出的 Semi-Dynamic Light Maps[16]从另外一个角度，通过单独预计算每个物体的光影贡献，整合出符合当前光照情况的的 Lightmap，以达到可以动态开关灯光，调整光照强度、颜色的目的。

2 技术路线

提出了协作式全局光照渲染系统的完整构架。整个系统由 2 个渲染器构成，分别是运行在强大的 GPU 上的云渲染器和运行在浏览器上的 Web 渲染器，两者通过 WebSocket 协议进行通信。而在本系统中，运行在不同软硬件环境下的 2 种渲染器的运行流程都有一定独立性和完整性，如图 1 所示。

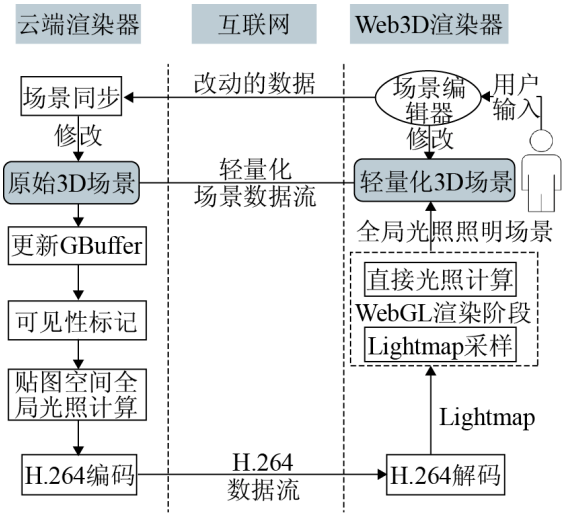


图 1 协作式全局光照渲染系统构架
Fig. 1 Collaborate global illumination rendering system structure

具体为：

(1) 云渲染器工作流程：

1) GBuffer 预计算：对虚拟场景光照计算中需要使用的模型相关信息，如：深度信息、法线信息或顶点信息进行预计算，并将其保存在相关贴图中。若场景中的模型位置发生变化，则该阶段需要不断的更新，否则只要生成一次即可。

2) Lightmap 生成：系统会据用户交互操作，确定 Lightmap 计算范围，进行标记，并通过贴图空间全局光照算法生成 Lightmap。

3) Lightmap 编码流式化传输：对 Lightmap 进行 H.264 编码，并将其流式化传往 Web 前端。

(2) Web 渲染器工作流程：

1) Lightmap 解码：对从云服务器端接收到的 H.264 数据流进行解码并还原成 Lightmap。

2) Web 前端直接光照计算：通过 Web 渲染器对 Web3D 场景的直接光照进行渲染。

3) Web 渲染器后处理：通过采样 Lightmap 获得全局光照漫反射和镜面反射数据以获取最终的渲染效果并输出。

在整个协同式云渲染系统的生成过程中，主要贡献如下：

(1) 提出一种贴图空间实时渲染的体素化全局光照算法。将全局光照信息存放于视点无关的 Lightmap 而非视点相关的数据结构中，本系统可以把场景中所有光照信息都存储在一张贴图中。由于网络环境延迟等问题导致云端无法及时传输新的光照数据，用户仍可以看到之前已经传输过的光照。并且通过预测机制，本系统可以保证在后几帧内，仍有充分的间接光照数据可以进行渲染。

(2) 提出一套根据用户的漫游行为动态调整全局光照计算范围和频度的协作策略。当用户在场景中进行漫游时，本系统仅计算当前视点下的可见表面的光照信息，一次渲染只改变 Lightmap 上一部分纹素的信息。而这些信息只有在场景中与光照相关的元素发生变化时，如场景中的模型数据或光源信息调整时，才需要更新。一旦 Lightmap 被完整的计算过一遍，且场景不发生改变时，就不需要重新计算漫反射光照。

(3) 提出一套实时云端光照协作渲染的解决方法。基于贴图空间全局光照计算与动态协作策略。本系统不必逐帧渲染、传输光照数据，且客户端也无需做过多的合成、处理工作，减轻负担的同时增强了画面的真实感。

3 Lightmap 的构建

与 Cloud Baking[2]渲染 GI map 不同，本系统将光照信息存储在贴图空间中，称为光照贴图(Lightmap)。光照贴图是一种记录物体表面光照信息的存储结构，贴图纹素的 UV 坐标与虚拟场景中 3 位模型的 UV 坐标一一对应，可以根据 UV 来索引光照信息。

传统的离线烘培器一般只计算全局漫反射或者漫反射与折射的混合信息，存放在 Lightmap 的 RGB 通道中。然而，全局漫反射是一种视点无关的信息，折射却会根据视线方向发生改变。若将二者分离存储，当用户在已经计算过全局光照的部分场景漫游时，完全可以仅计算折射部分信息，进一步降低了计算量。基于这一思路，根据渲染方程重新设计了 Lightmap 纹素的存储内容。

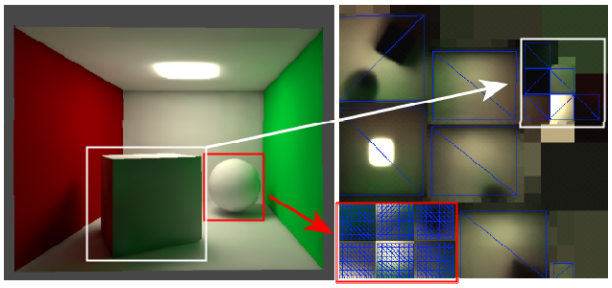
本章将介绍在云渲染系统中如何在 Lightmap 中构造和存储视点无关的全局光照信息。

3.1 Lightmap 的 UV 坐标的获取

场景模型通常表述为三角网格。为了将场景表面的光照像素保存到贴图中，需要将三角网络参数化到二维 UV 空间中去，并将 UV 缩放、平移，打包到一个[0,1]的范围中，形成 Lightmap 图集。区别于贴图 UV，Lightmap 的 UV 需要保证场景中每一个表面都于贴图的一块一一对应，不与其他物体有重叠。如图 2 所示，以 Cornell Box 场景为例，图 2(a)框中的 3D 模型通过 UV 参数化后映射到图 2(b)对应颜色框内。UV 的参数化及打包操作可以通过 UVAtlas[17]、或者使用 RizomUV 等软件完成。在文献[13,18]中，也提到相应的

DOI: 10.16182/j.issn1004731x.joss.19-VR0471

Lightmap 预处理操作。最后将生成后的 UV 坐标数据填入模型顶点的第二套 UV 通道中。



(a) Cornell Box 场景　　(b) Lightmap

图 2　UV 参数化
Fig. 2　UV parameterization

3.2 Lightmap 的纹素生成与存储

在基于物理渲染中，当物体被 n 个非面光源照明时，给定视线向量 $\boldsymbol{v}$，着色方程如下：

$$L_0(\boldsymbol{v}) = \sum_{(k=1)}^{n} f_{shade}(E_{L_k}, \boldsymbol{l_k}, v, \boldsymbol{n}, c_{diff}, c_{spec}, m) \quad (1)$$

式中：L_0 为出射的辐射度，看到的物体表面的颜色，由 rgb 表示；E_{L_k} 为第 k 个光源的辐射照度；$\boldsymbol{l_k}$ 为入射光方向向量；v 为视线；方向 $\boldsymbol{n}$ 为该点表面法线向量；c_{diff} 与 c_{spec} 为物体表面漫反射折射率与高光反射折射率；m 为高光模型(如 Blinn-Phong 模型)中高光扩散系数或粗糙度。

在实时渲染中，着色方程通常会分解为多个部分，如在 unity 中，分为直接光照部分与间接光照部分：

$$f_{shade}(v,k) = f_{direct}(k) + f_{indirect}(v,k) \quad (2)$$

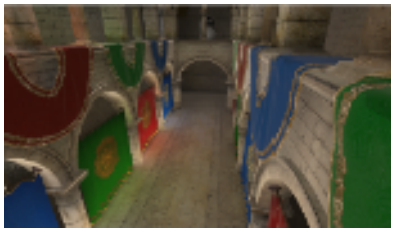
在本渲染系统中，直接光照部分由前端 Web 应用程序直接渲染，在服务器端我们只关注间接光照部分，间接光照可以通过实时全局光照算法(PRT，VoxelBasedGI)、记录 SH 的 Light Probe 或者环境贴图提供，最终合并成漫反射部分和高光部分，方程如下：

$$f_{indirect}(v) = c_{diff} \times diff(L_{Ind}) + R_F(\theta) \times spec(L_{Ind}) \quad (3)$$

式中：E_{Ind_k} 为间接辐照度；$diff$ 为取 E_{Ind} 的漫反射部分；$spec$ 为高光部分；R_F 为 Fresnel Term。

由于本系统在预处理是将场景中所有物体的表面都参数化、打包到了一张 Lightmap 中，这意味着 Lightmap 中只能携带低频的信息，所以漫反射贴图或者折射贴图这类高频信息贴图不适合存放在其中，见图 3。

(a) Lightmap 中存储 Albedo 和光照信息



(b) Lightmap 中仅存储光照信息

图 3　Lightmap 存储不同信息效果对比
Fig. 3　Effects comparison of storing different information in lightmap

而低频的全局光照信息却可以完美存放其中，所以我们也没有考虑 SAS 机制[10]。此外，仍需要考虑到在系统后续处理中，Lightmap 必须满足相关性的特征，以便对 Lightmap 进行图像上的压缩、编码成适用于网络传输的 H.264 格式。这使得我们不可以使用“打包”的方法，直接将 6 个低精度(RGB565)颜色值直接存到 RGBA8888 的 32 位通道中。在系统中，我们将镜面反射辐照度简化为亮度，这样就可以保存在透明通道中，RGB 存储漫反射辐照度，见图 4。

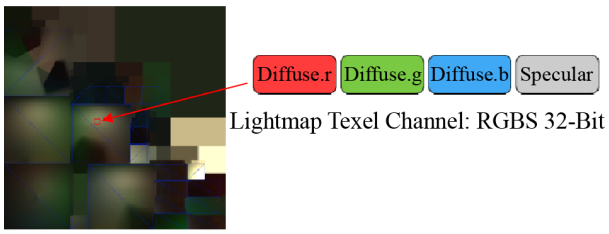


图 4　Lightmap 贴图纹素存储信息
Fig. 4　Lightmap texel channel information

4 贴图空间全局光照计算

传统的全局光照烘培器只能一次对整个场景烘培出一张或多张完整的 Lightmap，并且由于他们为了效果和质量，往往会选用一些无偏的全局光照算法，如光子映射，路径跟踪等，且大都使用 CPU 计算，这导致了这些烘培器往往需要几十分钟甚至几小时来烘培出 Lightmap，这不符合前端 Web3D 应用轻、快的特性。所以我们需要选择一些可以快速、甚至接近实时的计算出全局光照的算法，这里使用体素化全局光照算法。

4.1 Lightmap 计算范围的确定

首先需要确定 Lightmap 中有哪些纹素需要加入计算。传统的全局光照烘培器在光源或者物体参数发生改变时，会对 Lightmap 上所有纹素进行重新计算，这样计算量大，耗时长。当光照、物体动态变化时，系统没必要计算除了屏幕之外的物体。当且仅当相机发生变化时，没必要计算与视点无关的漫反射部分光照。前端后端的计算、传输必然存在延迟，如果后端只计算当前帧画面，前端接受到 Lightmap 必然是前几帧的光照。基于上述问题，提出了一种视点限制的 Lightmap 标记方法(View-Restricted Lightmap Marking，VRLM)。对于每一个 Lightmap 纹素，其对应的位置要首先在本帧或后两帧的可视范围内，本系统才会对其进行计算。基于该方法，本系统使用 3 种 Lightmap 的计算策略：

(1) 在客户端加载模型场景阶段对整张 Lightmap 进行计算。用户在场景中进行漫游，仅相机发生变化时，只对那些针对相机变化的部分使用 VRLM 方法计算与视点相关的折射部分(Alpha 通道)全局光照。

(2) 若光源动态变化，系统使用 VRLM 计算漫反射和折射部分全局光照。

(3) 若光源经过变化以后长时间保持不动，系统则在 2/3 的基础上对没有被 VRLM 标记的部分进行计算。

4.2 基于视点预测的可见性标记

当用户在客户端 3D 场景中进行漫游的时候，每一帧相机的移动幅度是连续，微小的。根据上上帧前端接收到数据的时间和上一帧接收到数据的时间之差(作为值传到服务器中)，使用 DeadReckoning[19]算法预测从当前帧开始的后两帧相机位置数据，见图 5。

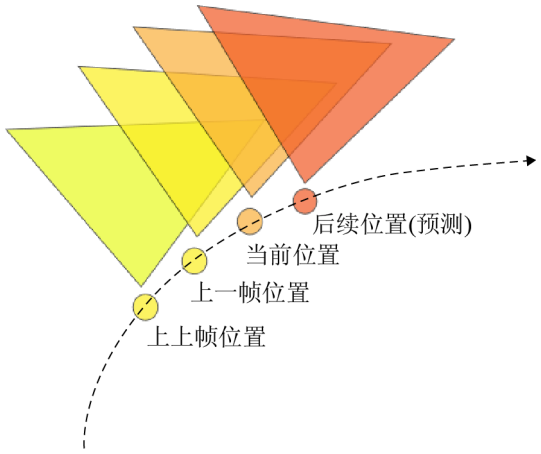


图 5 相机可见范围预测
Fig. 5 Camera visibility area prediction

有了当前相机和之后两帧相机参数后，本系统从 3 个相机视角渲染 3 张深度贴图(Depthmap)。深度贴图上记录着每个可见面片的深度信息，这时就可以对 Lightmap 进行标记可见信息。将 Lightmap 上每个合法纹素的世界位置通过相机透视矩阵变换转换到相机空间中，然后比较变换后的坐标 z 通道值与深度贴图对应位置的值的大小，若当前纹素的值与深度贴图的值的差在一定阈值里，就认为这个纹素是可见的。如图 5 所示，三角形表示相机视锥体，黄色三角形表示当前位置下的相机可见范围，橙色和橘黄色三角形表示经过 DeadReckoning 预测以后的相机可见范围。

另外，当光源不变时，之前已经存储了光照信息的纹素无需重复计算漫反射部分的光照，只计算视点相关镜面反射信息即可。

所以，对于每一个像素都需要存储 2 个布尔值以判断其是否需要进行漫反射，镜面反射的计算。

故本系统使用一张 8 位整型格式(R8)的贴图记录标记信息，上述两个布尔值通过位运算的方式打包到一个 8 位整型数中，称之为 Markmap。

算法 1 深度贴图可见性标记算法

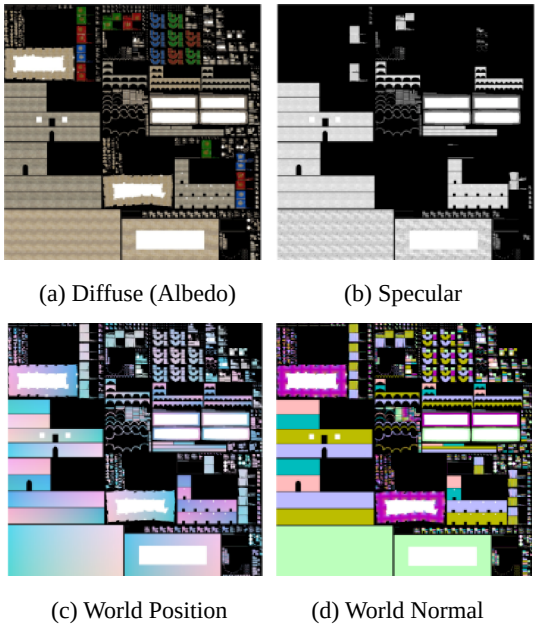
```
foreach texel in Lightmap
    for i=1 to 3
        P` = Transform P(Texel) to Camera 1 Space
        if (abs(P` - Depthmap(Texel) < Threshold)
            Specular = true
            if (Last_Lightmap(Texel) > 0 and
                Camera Not Move)
                Diffuse=false
            end
            Flag = Pack Specular and Diffuse
            Atomically set Flag to Markmap(texel)
        end
        Break;
```

4.3 贴图空间几何缓冲机制

在延迟渲染[20]管线中，由于本系统仅对屏幕上的可见像素进行着色，所以需要存储屏幕空间每个像素对应的材质属性和几何信息，称为几何缓冲(GBuffer)。我们的方法使用光照贴图空间计算光照，对于每一个纹素，本系统在计算的时候也需要知道当前像素对应表面的几何(法线、UV)和材质(漫反射颜色、漫反射系数、折射系数)，存储在多张贴图中，如图 6 所示。我们称其为贴图空间几何缓冲(Texture Space GBuffer，TSG)。这样，对于每一个贴图的纹素，系统可以从贴图的 UV 坐标值中获取各种着色需要的信息，计算出光照贴图，最后在前端着色阶段根据物体的 UV 坐标采样光照。

4.4 贴图空间体素化全局光照算法

整个全局光照系统中的全局光照算法主要基于 Sparse-Octree Global Illumination[21]算法。具体步骤如下：(1) 将场景表述为体素；(2) 将光源“传播”到体素中；(3) 对标记的纹素，根据 GBuffer 及体素信息进行圆锥体追踪，将辐照度写入对应 Lightmap 纹素中。



(a) Diffuse (Albedo) (b) Specular

(c) World Position (d) World Normal

图 6 贴图空间 GBuffer
Fig. 6 Texture Space GBuffer

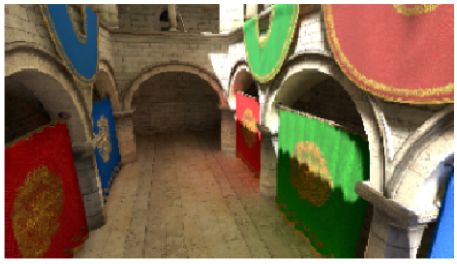
本系统使用 Sparse-octree Global Illumination[21]算法对场景做体素化操作。该操作会将场景模型由面片表述转换成体素块形式表述。系统将体素信息保存在 3D 纹理中，3D 纹理中的每一个体素存放一个光照值，表示改体素块反射的辐照度大小，由 RGB 表示。为了能在之后的环节支持加速碰撞检测、步进 ConeTracing 等操作，系统中使用多层级的 Mipmap 实现八叉树来管理体素。在 MipMap 的纹理金字塔中，下一层的 8 个节点与本层相同位置的父节点对应。在这里，我们创建了 6 层的 MipMap。

在创建好体素后，算法在每一个光源视角生成光源视图贴图。光源视图贴图用来描述一个光源所看见的所有可见表面的世界坐标位置。对于平行光源来说，光源视图贴图是用正交相机渲染获得，点光源则对光源周围六个面进行渲染，生成立方体贴图。

在最终渲染阶段，本系统使用 GPU 对所有 Lightmap 标记过的纹素，通过 UV 坐标查询 GBuffer 上的贴图、世界坐标等数据，在体素空间中进行圆锥体跟踪，将最终计算出来的全局光照漫反射和折射亮度分别存入该纹素的RGB和A通道中。

5 实验

在实验阶段，我们的云渲染器使用 C#和 unity 实现，运行在 e3 1231 v3，16 G 内存，GPU 为 GTX1070ti 的服务器上。客户端 Web3D 程序使用 ThreeJS 编写，运行在 Chrome 浏览器上，电脑配置为 i5-8400，16 G，GTX1050。我们使用 Sponza，Conference，Sibenik 作为测试场景，见图 7，使用体素化全局光照算法计算到一张 1 024*1 024 的光照贴图，客户端屏幕分辨率为 1 920*1 080。



(a) Sponza

(b) Conference

(c) Sibenik

图 7 实验测试场景
Fig. 7 Test scenes

实验 1 测试了 Web 渲染器在进行协作式渲染时的帧率变化。我们截取了 Web 客户端加载场景，并于服务器建立连接以后的 20 s 帧率变化。图 8 中可以看出，前端帧率始终维持在 30 帧以上，在 30~45 帧之间浮动。

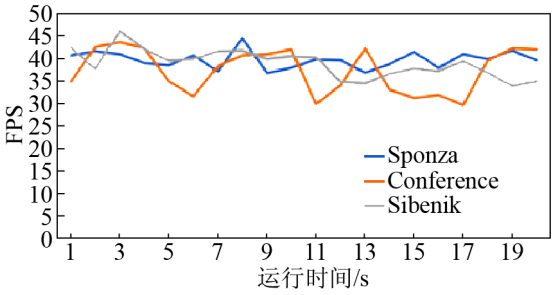


图 8 Web 客户端帧率变化
Fig. 8 Web Client FPS

由于 Lightmap 在有限的空间内存储了场景中所有的光照信息，每一个面片分配到的空间有限，导致光照信息必然受到一定的损失。实验 2 测试了云端计算的光照数据在 Lightmap 存储后在经过采样能够还原多少光照信息。为了消除不同渲染器采用不同的渲染算法导致最终渲染画面不同的影响，在本次实验中我们使用 Unity 作为渲染平台。在同一个场景、同一视角下，我们选取使用传统体素化全局光照算法渲染出来的全局光照画面作为参考图。模拟前端还原全局光照方法，通过采样 Lightmap 图还原出来的全局光照画面作为待评价图，使用 SSIM 进行全图像参考评价。由于参考图和带评价图都是屏幕空间下的全局光照信息，也是一种 GIMap[2,8]。故我们采用相同的评价体系，SSIM 在 0.88 及以上是，主观映像评测(Mean Opinion Score，MOS)分数达到 3~5 分之间，评价较好。

图 9 可以看到，在 3 个场景中，使用 Lightmap 还原的光照都有较高的 SSIM 数值(均大于 0.93)，说明使用我们的方法可以达到一个令人满意的渲染效果。在系统的设计上，我们的动态 Lightmap 更新机制可以低失真、快速的计算，并通过网络传输光照信息。为了评估云渲染器每一个阶段的 CPU/GPU 耗时情况，我们收集了运行阶段连续的

600 帧耗时数据。分别对可见性标记、体素信息更新、漫反射计算、镜面反射计算 4 个阶段的耗时求平均值从而得出平均耗时，见图 10。在 CPU 端，系统主要耗时在体素信息更新上，占比 93.51%。其余可见性标记、漫反射计算、镜面反射计算分别占比 5.20%，1.11%和 0.19%。在 GPU 端，系统主要耗时在漫反射计算上，占比 57.30%，其余可见性标记、体素信息更新、镜面反射计算分别占比 0.41%，37.97%，4.32%。性能消耗占比最大的体素信息更新模块在场景物体、光源不发生变化时无需每帧计算更新信息。GPU 耗时占比最大的漫反射计算模块则可以根据 4.1 节提到的计算策略在光源不变时，不重复计算漫反射光照，只计算与视点相关的折射部分光照，从而降低该模块对系统的负载。

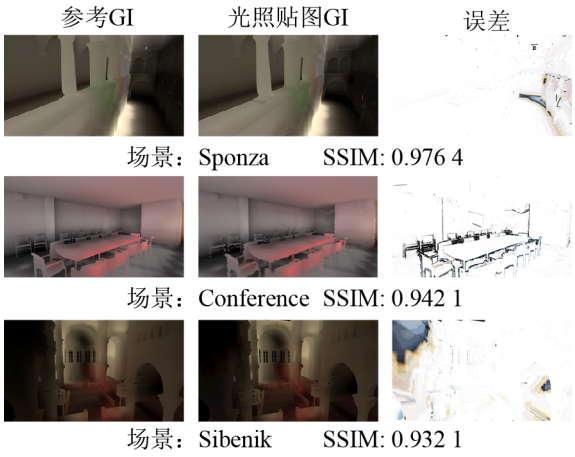


图 9 Lightmap 误差
Fig. 9 Lightmap error

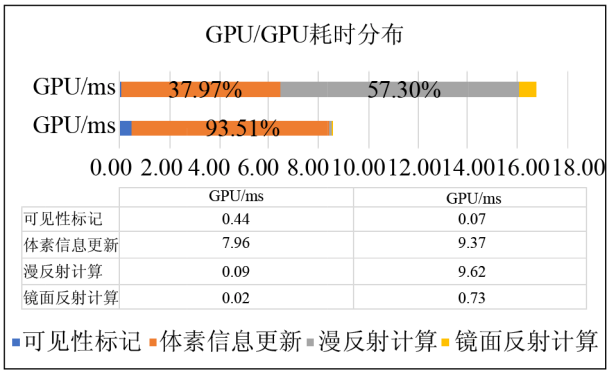
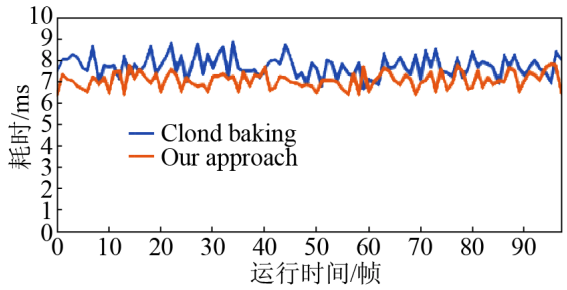
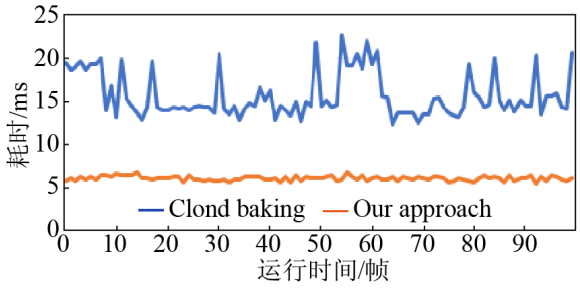


图 10 CPU/GPU 耗时分布
Fig. 10 CPU/GPU time consuming of each part

在 Cloud Baking[2]中，作者使用传统的屏幕空间体素全局光照算法，传输屏幕空间光照图 GIMap 的形式，每次只传输当前屏幕下的光照信息。所以这种方法必须实时地，逐帧传输 GIMap 至前端，无论光源有没有发生改变。为了对比 Cloud Baking 与本系统的服务器负载，在本次实验中，两种方法皆使用体素化全局光照算法，截取当相机发生变化过后 100 帧数据，见图 11。当光源发生变化后，由于本系统与视点无关，所以不需要重新计算漫反射数据，只需要计算镜面反射数据即可。而 Cloud Baking 由于输出的是视点相关的 GIMap，所以需要重新计算完整的全局光照数据。在 CPU 侧，本系统与 Cloud Baking 耗时相当，可以看出本系统虽然在云服务器端增加了可见性标记等步骤，但是不会造成过多的运算负担。在 GPU 侧，由于本系统仅需计算漫反射，耗时相对于 Cloud Baking 有显著的降低。因此我们的方法在服务器负载方面更有优势。



(a) CPU 耗时对比



(b) GPU 耗时对比

图 11 云渲染器使用本系统与 Cloud Baking 的 CPU/GPU 耗时对比
Fig. 11 CPU/GPU time consuming comparison of cloud renderer by using our approach and Cloud Baking

6 结论

本文提出了一个在贴图空间下计算的基于体素化全局光照算法的 Web3D 协作式渲染系统，算法的主要思想是使用 Lightmap 来存储和传输场景中所有的光照信息，这样就可以完美地解决使用 GIMap 传输会造成的“孔洞”伪像。我们使用体素化全局光照算法来实时计算全局光照。与传统的在屏幕空间计算全局光照不同，我们的方法直接在 Lightmap 贴图空间计算光照，并通过存储贴图空间的 GBuffer 来保存全局光照计算时需要使用到的物体贴图、光泽度、世界坐标和发现信息。针对用户不用的交互操作，我们提出了不同的光照计算策略，使 Lightmap 计算量降低，服务器端负载降低，同时能保证 Web 渲染时不会造成明显的伪像，极大的提高 Web3D 的画面质量。本文分别使用 Unity3D，ThreeJS 来实现客户端与服务器端的系统，通过 Sponza，Conference 两个场景验证了系统的有效性，并于当前流行的协同式渲染系统进行了对比，实验结果说明我们的方法可以有效的解决之前的方法所产生的问题，在性能和效果上都有显著的提高。

考虑到实时性，本方法使用基于体素化 GI 作为全局光照算法，但是该算法不能计算光线多次反弹的信息，同时该方法使用的圆锥体追踪也是一个粗略的光线追踪方法。随着 RTX 等硬件光线追踪不断发展，未来我们可以考虑将一些离线的光线追踪方法整合到实时渲染管线中，使其可以在云渲染系统中实时的渲染更加真实的全局光照效果。

参考文献:

[1] Crassin C, LuebkE D, Mara M, et al. CloudLight: A System for Amortizing Indirect Lighting in Real-Time Rendering[J]. Journal of Computer Graphics Techniques (S2331-7418), 2015, 4(4): 1-27.

[2] Liu C, Ooi W T, Jia J, et al. Cloud baking: Collaborative scene illumination for dynamic Web3D scenes[J]. ACM Transactions on Multimedia Computing, Communications and Applications (S1551-6857), 2018, 14(3s): 1-20.

[3] Huang C Y, Chen D Y, Hsu C H, et al. Gaminganywhere: An open-source cloud gaming testbed[C]// MM 2013-Proceedings of the 2013 ACM Multimedia Conference. New York, NY, USA: Association for Computing Machinery, 2013: 827-830.

[4] Koller D, Turitzin M, Levoy M, et al. Protected interactive 3D graphics via remote rendering[J]. ACM Transactions on Graphics (S0730-0301), 2004, 23(3): 695.

[5] Shi S, Nahrstedt K, Campbell R. A real-time remote rendering system for interactive mobile graphics[J]. ACM Transactions on Multimedia Computing, Communications, and Applications (S1551-6857), 2012, 8(3s): 1-20.

[6] Maamar H R, Boukerche A, Petriu E. Streaming 3D meshes over thin mobile devices[J]. IEEE Wireless Communications (S1536-1284), 2013, 20(3): 136-142.

[7] Liu C, Jia J, Zhang Q, et al. Lightweight WebSIM Rendering Framework Based on Cloud-Baking[C]// New York, NY, USA: Association for Computing Machinery, 2017: 221-229.

[8] 刘畅, 贾金原, 陆一凡, 等. 光图树: Web3D 动态场景全局光照的协同实时渲染[J]. 系统仿真学报, 2019, 31(8): 1591-1604.
Liu Chang, Jia Jinyuan, Lu Yifan, et al. GI-Map Tree: Global Illumination Collaborative Real-Time Rendering in Web3D Dynamic Scene[J]. Journal of System Simulation, 2019, 31(8): 1591-1604.

[9] Bugeja K, Debattista K, Spina S. An asynchronous method for cloud-based rendering[J]. Visual Computer, Springer Berlin Heidelberg (S0178-2789), 2018: 1-14.

[10] Voglreiter P, Dokter M, Neff T. Shading atlas streaming[C]//SIGGRAPH Asia 2018 Technical Papers. New York, NY, USA: SIGGRAPH Asia 2018. 2018, 37(6).

[11] Keller A. Instant radiosity[C]// Proceedings of the 24th annual conference on Computer graphics and interactive techniques SIGGRAPH 97, 1997, 31(3): 49-56.

[12] Dachsbacher C, Stamminger M. Reflective Shadow Maps[C]. Proceedings of the 14th Eurographics workshop on Rendering. New York, NY, USA: Association for Computing Machinery, 2003(1): 197-201.

[13] Luksch C, Tobler R F, Habel R, et al. Fast light-map computation with virtual polygon lights[C]. Proceedings of the Symposium on Interactive 3D Graphics. New York, NY, USA: Association for Computing Machinery, 2013:

https://dc-china-simulation.researchcommons.org/journal/vol32/iss4/12
DOI: 10.16182/j.issn1004731x.joss.19-VR0471

87-94.

[14] Thiedemann S, Henrich N, Grosch T, et al. Voxel-based global illumination[C]//Proceedings of the Symposium on Interactive 3D Graphics. New York, NY, USA: ACM, 2011: 103-110.

[15] Sloan P P, Kautz J, Snyder J. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments[C]//Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '02. New York, NY, USA: Association for Computing Machinery, 2002: 527-536.

[16] Öztürk B, Akyüz A O. Semi-dynamic light maps[C]. ACM SIGGRAPH 2017 Posters, SIGGRAPH 2017. New York, NY, USA: Association for Computing Machinery, 2017: 1-2.

[17] Zhou K, Synder J, Guo B, et al. Iso-charts: Stretch-driven mesh parameterization using spectral analysis[C]//ACM International Conference Proceeding Series. New York, NY, USA: Association for Computing Machinery, 2004, 71: 45-54.

[18] Apers D, Edblom P, Rousiers C de, et al. Interactive Light Map and Irradiance Volume Preview in Frostbite[G]//HAINES E, AKENINE-MÖLLER T. Ray Tracing Gems. Berkeley, CA: Apress, 2019: 377-407.

[19] Pantel L, Wolf L C. On the suitability of dead reckoning schemes for games[C]. New York, NY, USA: Association for Computing Machinery, 2004: 79-84.

[20] Akenine-Moller T, Haines E, Hoffman N. Deferred Shading[G]//Fourth Edition. Real-time rendering. AK Peters/CRC Press, 2018: 883.

[21] Crassin C, Neyret F, Sainz M, et al. Interactive indirect illumination using voxel cone tracing[J]. Computer Graphics Forum (S1467-8659), 2011, 30(7): 1921-1930.