Chapter 21
Real-Time Ray Tracing Dynamic Scenes Based on WebGL

Sam At Chea and Fuyan Liu

Abstract WebGL is an immediate mode 3D rendering API designed for the Web. Ray tracing is a method that can generate photo-realistic image. For the past decades, ray tracing on the Internet browsers is an impossible task. WebGL's appearance gives this impossible task coming true. In this paper, we introduced a WebGL-based ray tracing for small dynamic scenes (about 3 K triangles) in real time.

Keywords Ray tracing · WebGL · Illumination model · Uniform grid

21.1 Introduction

Internet browser becomes more and more important in our life. In the past years, most of the contents which were shown on browsers are two dimensional. It is very difficult to draw three-dimensional (3D) contents on the browsers, still using less ray tracing method.

WebGL's appearance gives this impossible task coming true. WebGL is the abbreviation of Web Graphics Library [1]. It is a new technology and was released on December 10, 2009. WebGL is a low-level 3D graphic JavaScript API for developing 3D Web applications. It is based on OpenGL ES 2.0, but there are

S. A. Chea (✉) · F. Liu
School of Computer Engineering and Science, Shanghai University,
Shanghai 200072, China
e-mail: cheasamat@hotmail.com

F. Liu
e-mail: lfy@126.com

some differences. By using WebGL, we can construct 3D graphics on the webpage without any external plug-in.

Ray-tracing method is a hotspot of study in computer graphics area. It is an important method for generating realistic images. This method was proposed by Turner Whitted, in the paper of "An Improved Illumination Model for Shaded Display" in 1980 [2].

The concept of Turner Whitted ray-tracing algorithm is very simple. In brief, generated rays intersect with objects in the scene, rays that intersected with objects will have an intersection point, the color of the ray and the object at this point accumulates again and again until the last ray traced.

Ray-tracing method is simple, but there are also some barricades in rendering photo-realistic images on browsers. In the ray-tracing method, the most time consuming process is doing ray-triangle intersection. Even in some professional ray-tracing software, it is hard to achieve real time for some complex scenes. In this paper, we focus on the implementation of ray-tracing method on webpages, and achieve real time for simple scenes.

21.2 Improved Uniform Grid Construction on Webpage

Uniform grid is the best spatial subdivision structure on GPU. The main idea of this spatial subdivision structure is to subdivide the axis aligned bounding box of the whole scene into the same size units along each of the three axes X, Y, and Z. The critical factor for constructing the best uniform grid is to determine the number of units of the whole scenes. The number of units is not permanent for different scenes. If the number of units is too much, this means fewer triangles in each unit, and fewer intersection tests per unit. However, it will spend much more time on traversing the uniform grid [3, 4]. So it is hard to choose a once and for all method.

Another problem is that our ray-tracing program is based on webpages, there are more limitations compared to the traditional way. So, in our program, the number of triangles of the scene is confined to 3,000. We improved the construction method of the uniform grid for our particular scenes.

Wald 3 proposed a general equation to determine the number of units:

$$U_x = d_x\sqrt[3]{\frac{kT}{V}} \quad U_y = d_y\sqrt[3]{\frac{kT}{V}} \quad U_z = d_z\sqrt[3]{\frac{kT}{V}} \tag{21.1}$$

U_x, U_y, U_z	represent the number of units in X, Y, and Z dimension;
d_x, d_y, d_z	represent the diagonal of the axis aligned bounding box of the whole scene;
k	is a self-defined constant number, used to determine the sparse degree of the units in the box;

T	represents the total number of triangles in the scene;
V	represents the volume of the axis aligned bounding box of the whole scene;

This equation was widely used for determining the number of units in each dimension. But it is a little complex for our scenes. After the experimental study, we found it is inappropriate for the simple scenes (about 3 K triangles). In order to achieve real-time ray tracing on webpages, we simplified this equation. The improved equation as follow

$$U_x = L_x\sqrt[3]{T} \quad U_y = L_y\sqrt[3]{T} \quad U_z = L_z\sqrt[3]{T} \tag{21.2}$$

U_x, U_y, U_z	and T is identical in meaning to the equation above;
L_x, L_y, L_z	represent the length of the axis aligned bounding box of the whole scene along each axis.

This improved equation decreases the complexity of the uniform grid. Following are the steps for constructing the improved uniform grid:

Calculate the axis bounding box of the whole scene.
Computer the number of units on each axis, using Eq. (21.2);
Subdivide the axis bounding box of the scene; create a triangle list for each unit.

After the uniform grid construction completes, we should map the data structure to a texture layout that can use as input on the GPU. Because WebGL is derived from OpenGL ES 2.0, 3D texture is not supported. We use 2D texture to store the uniform grid structure, see Fig 21.1.

In the Fig. 21.1, left part is the constructed uniform grid of the scene; this structure is consisting of two chunks. Two chunks are mapped to a texture, look at the right part of Fig. 21.1. Each chunk contains a pointer to a list of triangles. Because the width and height of texture in WebGL must be the power of 2, unused space is used to pad the width and height to the power of 2.

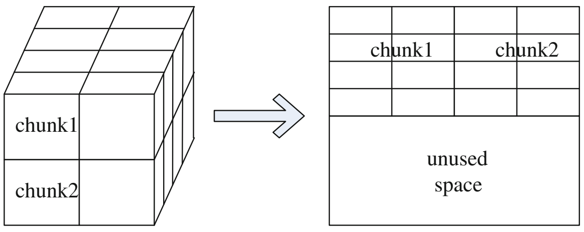

Fig. 21.1 Storing the uniform grid structure in texture

21.3 Headers Ray-Tracing Implementation by Using WebGL

WebGL is different to OpenGL in rendering pipeline. It is designed as a rendering context for the HTML Canvas element. Look at Fig. 21.2, it shows the simple process of rendering 3D scene by using WebGL [2] and [5].

First, browsers (such as Google Chrome, Mozilla Firefox, and Apple Safari) render the scene by using the graphics card on GPU, and then render the HTML. At last, images are combined on the CPU.

After the description of the background knowledge of uniform grid and simple rendering process, next is the implementation steps of our ray-tracing program. There are four basic steps for the whole process of our ray-tracing program:

1. Initial WebGL context and loading scene. WebGL serves as a binding between high-level JavaScript and low-level GPU operations.
2. Primary rays generate and traversal. Rays generate and traversal are completely on GPU. In fragment shader, ray-triangle intersection test is in parallel running. We use the 3D-DDA [6] traversal algorithm to find the closet intersection point.
3. Second rays traversal and shadow test. Second ray mainly includes reflection ray and refraction ray. These rays are similar with primary rays. Shadow test is used to determine if an intersection point is in shadow. Shadow rays are cast from the intersection point toward a light source, this ray also need to do intersection tests, if the closer intersection point is found, then this intersection point must be in shadow.
4. Computer shader. In steps 2 and 3, the intersection points are stored in a buffer. These points have some attributes, such as color, normal. In this step, we will accumulate the color of the intersection point, and then render the color to the screen.

The termination of one ray tracing decides by the tracing depth or when the ray hits a light source at last.

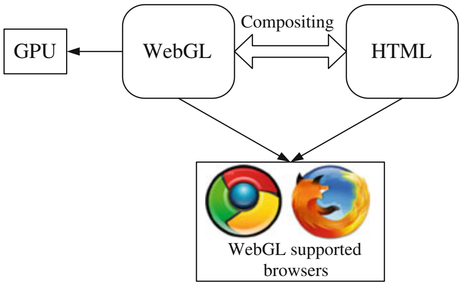

Fig. 21.2 Rendering process on webpages

21.4 Result

In our experiment, we use Mozilla Firefox to render the scene. The rendering result can be seen from Fig. 21.3. The scene consists of 2.3 K triangles and the rendering performance is 21.3FPS. Our image rendered at image resolution 640 × 320.

21.5 Future Work

We have successfully implemented a ray-tracing algorithm based on WebGL and generated some realistic images, but it is far from the end. WebGL is a new technology, only GLSL ES is supported, there are a lot of high-efficiency API cannot be used on WebGL. JavaScript also has some problems in execution speed. In our future work, we will find other ways to depress these problems.

Another future work is to improve the ray-tracing algorithm and implement some special effects. Ray tracing on webpages has some difference from traditional way, there is no way to designate the same for ray tracing on webpages. We will concentrate on the algorithm improvement in the future. Some special effects, such as refraction, depth of field, motion blur, and caustics are the important parts of ray tracing, we are missing a lot in this aspect right now. We will increase efforts to expand algorithms for implementing these special effects.

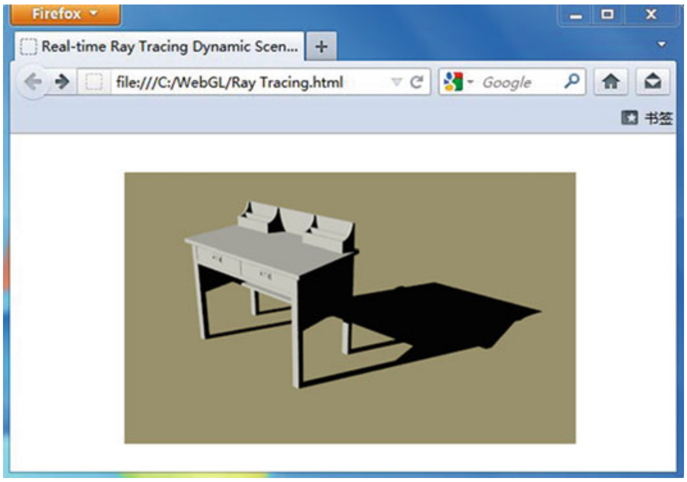
Firefox
Real-time Ray Tracing Dynamic Scen...
file:///C:/WebGL/Ray Tracing.html
Google
书签

Fig. 21.3 Rendering result image

References

1. Khronos Group WebGL Specification. https://www.khronos.org/registry/webgl/specs/1.0/
2. Whitted T (1980) An improved illumination model for shaded display. Commun ACM 98:343–349
3. Wald I, Ize T, Kensler A, Knoll A, Parker SG (2006) Ray tracing animated scenes using coherent grid traversal. ACM Trans Graph 47:434–436
4. Fernando R (2004) GPU gems: programming techniques, tip, and tricks for real-time graphics. Univ Appl Sci Basel 72:757–759
5. Kessenich J, Baldwin D, Rost R (2009) The OpenGL shading language. Version 46:1–5
6. Havran V (2001) Heuristic ray shooting algorithms, Ph.D. dissertation, Czech Technical University Praha, Czech Repubic 47:58–62