A Real-Time 3D Rendering System with BRDF Materials and Natural Lighting

Alexei Ignatenko[*], Ildar Valiev[**], Kirill Dmitriev[**], Boris Barladian[**], Sergey Ershov[**], Alexei Voloboy[**], Vladimir Galaktionov[**]

[*] Department of Computational Mathematics and Cybernetics
Moscow State University, Moscow, Russia
ignatenko@graphics.cs.msu.su

** M.V. Keldysh Institute for Applied Mathematics
Russian Academy of Sciences

Abstract

We present a system for real-time realistic rendering of 3D scenes. Most of available on the market 3d visualization systems lack physical correctness of rendering, especially concerning complex materials and light sources. Our system is aimed to provide the physically correct visualization to the extent possible by modern graphics hardware. It supports natural sunlight illumination, complex BRDF materials, real-time specular reflections, integrated computation of illumination maps and lighting textures, tone mapping control. Areas of application are automotive and architectural design.

Keywords: *real-time rendering system, BRDF, sunlight illumination, real-time reflections, shadow generation.*

1. INTRODUCTION

This paper describes a real-time realistic rendering system, called FLY. The main requirement to the system was interactive rendering speed and physical realism to the extent possible by modern hardware.

Important elements of visually persuasive result of 3D rendering are presence of shadows, specular reflections (e.g. mirrors), and secondary illumination effects (color bleeding, caustics, etc). Besides this, the rendering with the natural sunlight illumination (specified by a geographic location of the scene and a preferred time) is often needed for architectural design. At the same time, evolution of local illumination models is required to support the visualization of materials given by an arbitrary BRDF (Bidirectional Reflection Distribution Function [1]). Such kind of materials is required for visualization of car paints and other complex metallic surfaces, e.g., in automotive design.

In the paper we present our solutions and tradeoffs made in order to integrate different global and local illumination techniques while keeping the rendering at interactive frame rates.

A lot of 3d rendering systems exist on the market, from game engines and VRML browsers ([2], [3]) to specialized products for automotive and architectural CAD design ([4], [5], [6]).

However, to authors' knowledge, none of them meet our goals. VRML browsers and game engines could produce visually attractive images, but operate in color space instead of physical luminance. Besides, none of VRML browsers have integrated raytracer to calculate global illumination. Another drawback of even the most advanced game engines is necessity for manual design of scene, because most of the visual effects will not function properly in automatic mode. On contrary, our system works with arbitrary geometry.

Specialized commercial applications for automotive and CAD design declare support for interactive shadows and physically based rendering. At the same time, there are no known systems that support visualization of materials with true BRDF and natural daylight.

Figure 1: Examples of FLY interactive rendering**.**

Our system is based on modern rasterization hardware. We considered two approaches: traditional polygonal rasterization and modern interactive raytracing ([7]). An advantage of rasterization is high speed due to extensive hardware support. Interactive raytracing features internal support for global illumination. We've chosen the former, in spite of the fact that raytracing techniques better fit our needs on the first sight. However, there is a tradeoff between quality and speed, because interactive raytracing is still several times slower on typical scenes, especially for high-resolution displays. We use OpenGL ([8]) as a low-level hardware access layer API in connection with

own Monte-Carlo raytracer ([9]) used for calculation of lighting maps and textures.

A distinctive feature is that FLY operates with physical illumination attributes until final transferring to hardware (which requires vertex colors and textures in low-range monitor RGB colors). This allows effective manual and automatic tone-mapping control.

Several rendering modes were implemented: simple rendering mode (based on OpenGL internal Phong lighting), lighting maps mode (with pre-computed vertex luminance and lighting textures converted to OpenGL vertex colors as necessary), and a special sunlight mode, designed for simulation of sky and sun illumination.

A user is allowed to specify a geographic location of the scene and a preferred time. The sun and sky illumination is calculated from these attributes. The sky is converted into a background illumination; the sun position is used to generate shadows using the modified shadow volume technique ([10]). The internal OpenGL shading mode is replaced with externally calculated vertex colors. See Section 3 for details.

A special algorithm for rendering of surfaces covered by complex materials with given BRDF, was elaborated. It is based on real-time general of a reflection texture. This texture is then applied to corresponding objects via OpenGL spherical mapping. BRDF rendering is described in Section 4.

Highly specular materials are rendered using real-time specular reflections techniques built over OpenGL (see Section 5).

This paper is organized as follows. In the following section we present a bird's eye view of the system architecture. The next sections describe various features in details. In section 3 we describe implementation of the realistic sunlight illumination. Fourth section deals with the visualization of materials defined by BRDF. Section 5 describes an implementation of real-time specular reflections. Section 6 provides information about the support for illumination maps. Results are given in Section 7.

2. SYSTEM ARCHITECHURE OVERVIEW

The design of FLY system architecture is motivated by necessity to preserve physic attributes of the scene.

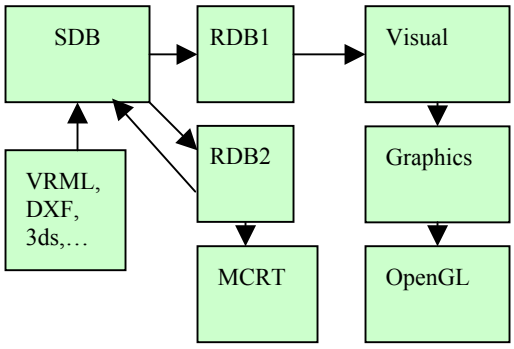

Figure 2: Overview of the system architecture. Generic SDB with physical attributes converted to two RDBs – one for OpenGL engine (Visual) and another for Monte-Carlo raytracing engine (MCRT).

The scene is kept in so-called SDB (Scene Description Database, see Figure 2). SDB is a hierarchical object-based structure, similar to scene interfaces used in VRML ([11]), OGLO ([12]), Java 3D ([13]), etc. Our representation can keep complex materials (with BRDF), lights (including ones with goniograms), etc., not limited to OpenGL-supported attributes. So it allows not only interactive rendering, but also complex offline photorealistic lighting simulation.

FLY supports import of different scene formats: VRML, 3DS, DXF, IOF (own internal format). If an imported format does not have physical attributes, an importing module creates attributes approximating the original ones.

FLY has two rendering engines: hardware and raytracing. Because SDB contains generic attributes, which cannot be at once passed to a rendering engine, for each type of a rendering engine specific data structures are prepared. We call them RDB (Rendering Description Database). Generic attributes are converted to more simple ones, understandable by given rendering engine. The user can control the transformation. For example, vertex luminancies are replaced by colors for OpenGL engine, by applying a tone-mapping operator. For raytracing engine, special optimization RDB, based on spatial subdivision, is also prepared. Each time SDB is changed by user actions or animation events, all RDBs are updated.

The interactive visualization module operates at RDB level. Its main purpose is different shading and optimization algorithms, such as view-frustum culling, back-to-front rendering for semi-transparent objects, multi-pass rendering, etc. Shadows, BRDF rendering, sunlight illumination are implemented in this module.

The final stage of the data flow is a graphics module, which is only a wrapper over the system graphics API. Currently we have implemented a wrapper over OpenGL, though other APIs are also possible.

3. SUNLIGHT ILLUMINATION AND SHADOWS

Figure 3: Natural sunlight rendering. Left – dawn (8:00), right – morning sun (11:00).

Rendering of scenes illuminated with sunlight is an important special case, used in architectural CAD systems. Visually persuasive and physically accurate rendering of sunlight involves

the following algorithms: generation of corresponding light sources for the sun and sky, generation of the scene background that represents sky, generation of shadows from the sun, and vertex light maps and light textures according to the illumination.

User is allowed to add sunlight to the scene by entering either geographic location of the scene or by direct specification of sun position by latitude and longitude. After this, the sunlight is added to the scene and the visualization mode is changed.

When sunlight is present in the scene, a semi-spherical background representing sky with the sun, is generated. It takes into account the location of the sun, so colors are different for the noon and the morning sun; see Figure 3 for an example.

The next step is the vertex colors calculation. Usually scenes with sun have high level of secondary illumination; so standard OpenGL internal Phong shading will not give attractive results.

Good results can be archived by calculation of lighting maps using Monte-Carlo raytracer. But it can take significant time. Therefore, we use different strategies depending on lighting map availability. If a lighting map is not available (has not been previously calculated), vertex colors are calculated using modified Phong model. Our modification adds empirical "directional ambient" term, which simulates secondary diffuse illumination by raising the luminance of corner vertices. Though this method is not physically justified, it gives good results. Certainly, if the user calculates lighting maps, they are used instead of the local illumination model.

Finally, we construct shadows from the sun. This component is especially important, because on a sunny day shadows are very distinctive and sharp. There are two existing approaches for real-time shadow generation: shadow mapping and shadow volumes ([10]). Shadow mapping is a bit easier to integrate, because it smoothly works with any geometry. On the other hand, shadow volumes produce better quality sharp shadows.

We have implemented a modified shadow volume technique. Our modifications are aimed to decrease amount of shadow volumes, needed by the algorithm. Known solutions create shadow volumes geometry by finding all edges possibly belonging to object's silhouette. For complex object with rich internal topology (e.g., houses, or car with interior) this will create a lot of excess shadow volume geometry, significantly decreasing rendering speed.

We improve this algorithm by rendering a scene in an orthographic projection from a sunlight view and building a visibility map, which allows us to determine lit triangles (i.e. visible from the sun). This is done by encoding index of each triangle by its color. In case of naïve implementation of this algorithm, too little triangles (smaller than one pixel) can be missed in the visibility map. This will be noticeable as holes in shadows. We avoid this situation by splitting geometry into clusters, estimating minimal triangle size in the cluster and applying iterative algorithm that guarantee visibility of all significant triangles.

After this stage, we apply a silhouette-finding algorithm only for triangle visible from the sun.

4. BRDF-BASED MATERIALS RENDERING

FLY uses measured representation for materials with defined BRDF ([1]). Qualitative interactive visualization of surfaces with such materials is a challenging task, because it requires significant amount of computational work for accessing and interpolating values in BRDF tables. Moreover, per-vertex calculations are often not appropriate because they can miss unique features of particular BRDF, such as highlights.

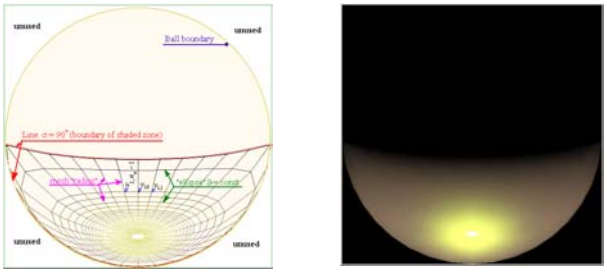

Figure 4: Ball image mesh for BRDF visualization

We use a novel technique for per-pixel calculation of BRDF-based lighting. This technique is based on real-time generation of a special texture, applied to the corresponding object by the reflection mapping. This is similar to the algorithm used in [14], however, the difference is in the way this texture is created. Cabral et al use an image-based technique based on weighted blending of several pre-computed spherical textures. Our approach is based on real-time generation of the texture from measured BRDF representation.

The spherical texture is an image of a unit ball made of a given material under given illumination conditions. In our approach, the image is generated by calculation of the colors of vertices of some mesh, and then interpolating inside its cells with OpenGL shading. This mesh is created for each light source and is optimized to have more dense cells around light reflection direction thus approximating highlight shape (see Figure 4). Ball images for different light sources are summed up to generate a final texture that is applied to the object using OpenGL reflection mapping.

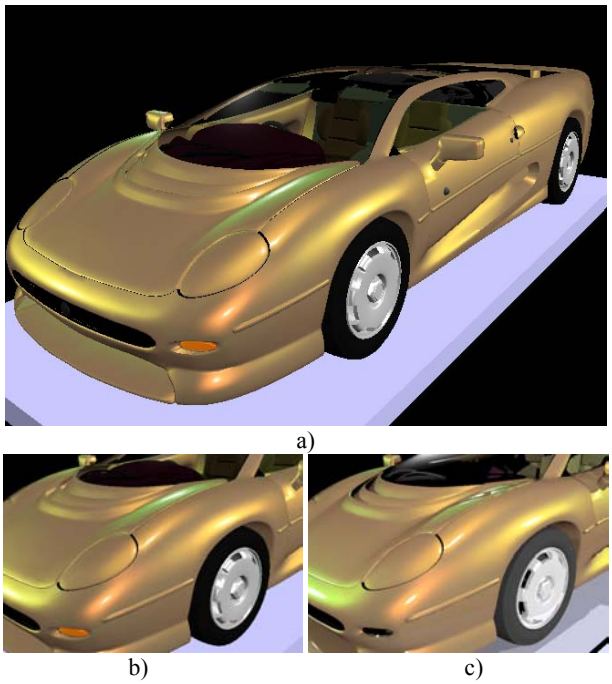

Figure 5: BRDF rendering examples. a) A car painted with BRDF material, lit by 5 point light sources b) A close-up rendering of the paint c) The same part rendered by a raytracer

This technique is suitable for 3D BRDFs and distant light sources. However, it shows attractive results for car paints. Comparison with raytraced BRDF materials shows little difference (see Figure 5).

Calculation time for 256x256 ball image for 5 point light sources is less than 15ms.

5. REAL-TIME SPECULAR REFLECTIONS

OpenGL shading produces only approximate reflections of light sources. In order to improve realism of general images, we generate reflection of scene geometry by additional real-time algorithms.

We apply different methods for flat and curved reflectors. For the latter we use a well-known stencil-buffer technique ([15]). It is, however, not suitable for curved surfaces. For such surfaces we use reflection mapping with cubic environment map ([16]). Algorithms are selected automatically, depending on object's shape and material.

Figure 6: Real-time specular reflections with light maps rendering mode.

6. ILLUMINATION MAPS

An important part of FLY's rendering engine is a raytracing component. Our Monte-Carlo raytracing engine supplements OpenGL-based visualization. The raytracer is used to calculate vertex light maps together with light textures. Light maps and textures are stored in floating point luminance format and converted to vertex colors by applying user-defined tone mapping operator ([17]).

A special care is taken to process two-sided (open) geometry. For such surfaces we create light maps and textures for both sides. At the rendering time, a special vertex shader is used to select proper color and texture.

Figure 7: Illumination maps and shaded textures. Left – raytracing, right - OpenGL rendering (a lighting texture was applied to the floor)

7. RESULTS

Tests show that FLY is able to visualize scenes with sun illumination, BRDF and real-time specular reflections at interactive frame rates. The following Table 1 summaries performance tests that were done on an ordinary PC with Pentium4 2,8Ghz 1Gb RAM and Radeon 9700 Pro video accelerator.

Scene	Triangles number	Simple mode	Light maps	Sunlight mode
Mercedes (BRDF)	137551	33fps	35 fps	12 fps
Jaguar (BRDF)	107222	33 fps	38 fps	14 fps
House	312923	24 fps	26 fps	7 fps

Table 1: FLY performance in different rendering modes (frames per second).

8. ACKNOWLEGMENTS

This work was supported by INTEGRA Inc. (Tokyo, Japan).

The version of the paper with color illustrations can be found on http://www.keldysh.ru/pages/cgraph/publications/cgd_publ.htm

9. REFERENCES

[1] B. K.P Horn, "Robot Vision", MIT Press and McGraw-Hill, Cambridge, MA, 1986. ISBN: 0070303495 (Russian translation: Хорн Б.К.П. Зрение роботов: Пер. с англ.-М.:Мир, 1989.-487 с.,ил.).

[2] Cortona VRML Client. http://www.parallelgraphics.com.

[3] Blaxxun VRML Client. www.blaxxun.com.

[4] Opus Realizer. http://www.opticore.com

[5] Outline 3D. http://www.outline3d.com

[6] RTT. http://www.realtime-technology.com

[7] Cyrill Damez, Kirill Dmitriev, Karol Myszkowski "Global Illumination for Interactive Applications and High-Quality Animations", STAR - State of The Art Report, Eurographics, 2002.

[8] http://www.opengl.org

[9] J.T. Kajiya, The Rendering Equation, Computer Graphics, Annual Conference Series, ACM SIGGRAPH, pp. 143-150, 1986.

[10] Andrew Woo, Pierre Poulin, Alain Fournie, "A Survey of Shadow Algorithms", IEEE Computer Graphics & Applications, 1990.

[11] http://www.web3d.org

[12] http://www.sgi.com/software/optimizer/

[13] http://www.j3d.org

[14] B. Cabral, M. Olano, and P. Nemec. Reflection Space Image Based Rendering. SIGGRAPH, pp. 165–170, 1999.

[15] M. J. Kilgard, Real-time Environment Reflections with OpenGL, Slides, nVidia Corp.,1999.

[16] M. J Kilgard: Perfect Reflections and Specular Lighting Effects With Cube Environment Mapping, Technical Brief, nVidia Corp., 1999.

[17] Boris Barladian, Robust Parameter Estimation for Tone Mapping Operator, 13-th International Conference on Computer Graphics and Vision GraphiCon-2003, Moscow, September 5 -10, Conference Proceedings, pp.106-108, 2003