XRay: Photorealistic Rendering System

Ioannis Tsiombikas*
City Liberal Studies
Affiliated Institute of the University of Sheffield

Panagiotis D. Bamidis†
Lab of Medical Informatics
Medical School
Aristotele University of Thessaloniki

Dimitris Dranidis‡
South East European Research Centre
Research Centre of the University of Sheffield
and CITY Liberal Studies

Abstract

This describes "*X-Ray*", a "*photorealistic rendering system*", or simply a "*renderer*". A program that facilitates the production of photorealistic visualizations, and time-varying animations, of mathematically defined 3D environments. This system provides an advanced feature-set, such as monte-carlo ray-tracing, programmable shaders and network rendering, often encountered in expensive commercial renderers, while being available as free software.

Keywords: rendering, computer graphics, ray tracing, photorealism

1 Introduction

Visual perception is one of our more refined senses. "A picture is worth a thousand words", goes the popular proverb, which is true due to the great efficiency of information retrieval in visual media.

1.1 Definitions

Computer graphics is the field of computer science that deals with algorithms for the presentation of visual information through a computer. This ranges

*e-mail: nuclear@siggraph.org
†e-mail: bamidis@med.auth.gr
‡e-mail: ddranidis@seerc.org

from mundane tasks such as drawing graphical user interfaces for human-computer interaction, to visualizing immersive 3-dimensional environments or presenting scientific data.

3D computer graphics, mostly deal with the mathematics and algorithms of converting 3-dimensional objects and environments into raster images. A raster image is a rectangle area of pixels, which can be presented as a continuous signal of varying intensities to humans, through regular computer displays.

The process of transforming a concise mathematical description of a virtual 3D environment into a raster image, is called "rendering", and the software that performs this task is called a "renderer".

The significance of the word "photorealistic" in this context refers to the attempt to facilitate the creation of visualizations which are more or less indistinguishable from reality. This is done, at the software level, by providing the ability to calculate the various subtle interactions of light with the virtual environment. And this is one of the major goals of this system.

1.2 Geometry Representation and Rendering Algorithms

There are many ways to represent the geometry of 3D objects in a computer. The most widely used are the following:

- Mathematical equations describing the object. For example the quadratic equation $x^2 + y^2 + z^2 = r^2$, perfectly describes a sphere.
- Polyhedral boundary approximation. Using a mesh of polygons to approximate the shape of the object's surface, for example 8 polygons can be connected to form a ring, thus approximating the surface of a cylinder.
- discrete 3D scalar fields, or "voxels" can be used to define the volume occupied by the object.

Of these, the polygon mesh representation is the most popular, due to the ease of manipulation and rendering; transforming the polygon vertices transforms the object itself, and projecting the polygons to a view plane is sufficient to visualize the object. But most importantly, the fact that given enough polygons any shape can be approximated to an arbitrary degree.

There are two basic algorithms used in 3D rendering: *polygon rasterization* and *ray tracing*[1]. The first one is almost always used in real-time rendering, where it is important to be able to render many times per second, because it is inherently very efficient. However, it has nothing to do with how light interacts with the environment, making it extremely hard or impossible to robustly produce effects such as reflection, refraction, or even

shadows. Thus, where quality and photorealism is more important than speed of execution (i.e. off-line rendering), ray tracing is the way to go.

With ray tracing, the basic algorithm for every pixel of the raster image, "shoots" rays backwards from the view point, into the environment and calculates intersections of each ray with the 3D objects. Wherever an intersection is found, the illumination contribution from each light source is calculated, and secondary reflection or refraction rays are shot recursively to the appropriate directions. The resulting illumination returned by that recursive tracing is accumulated with the local illumination to produce a color value for the pixel that spawned each primary ray.

2 X-Ray Features

The X-Ray renderer is a photorealistic off-line rendering system, providing advanced features commonly associated with expensive commercial renderers. Some of the key features of this renderer are:

- Advanced, monte–carlo ray tracer, able to simulate the subtle interactions of light with the virtual environment, required for photorealistic rendering.
- Extensibility through programmable shaders, which can be written to control the rendering process, and extend the renderer with custom shading models or rendering algorithms.
- Flexible client–server architecture.
- Reusability; the rendering core is written in the form of a cross–platform ISO C/C++ library, which can be linked with any application that requires 3D visualization.
- Fully specified XML scene description format, which is easy to parse, and easy to write converters and other supporting utilities, for maximum interoperability with 3D authoring programs, or other renderers.
- Free software; The whole system is released under the terms of the GNU General Public License (GPL).

Arbitrary subsets of these features are in one way or another available in other rendering systems out there. For instance, there are some free software renderers available, like "POV-ray"[2], but they don't support advanced features like programmable shading, which is only available in big, commercial, non-free renderers like "mental ray"[3] or Pixar's "Renderman"[4]. Their intersection is what makes this system ideal to be used, freely, by the scientific community as a platform for further research into graphics algorithms and photorealistic rendering.

Now in order to demonstrate why are these features important for a renderer, let's examine the most important of those in more detail.

2.1 Advanced Ray-Tracing

The basic ray tracing algorithm as described in the introduction is minimal and elegant, but fails to capture by itself some important aspects of the behaviour of light.

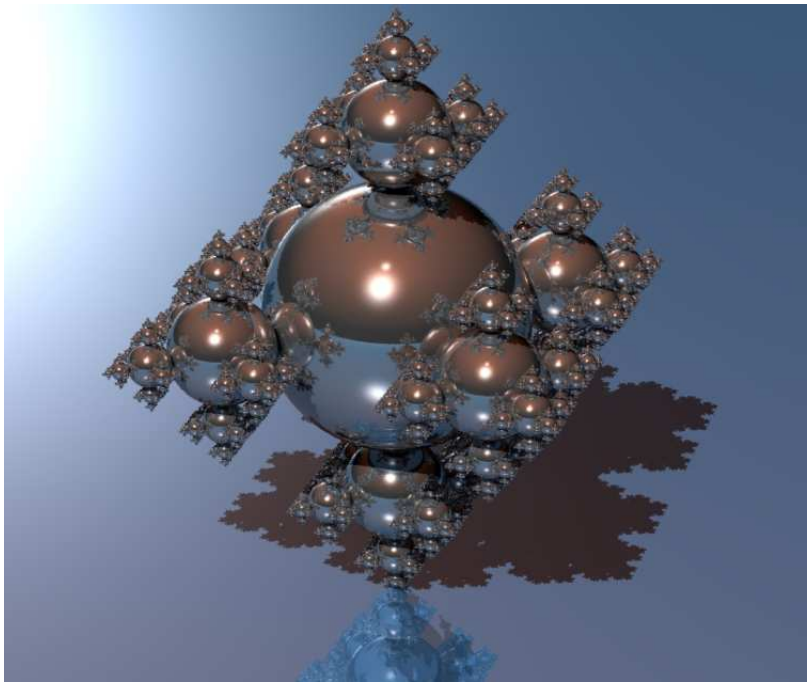

Figure 1: Simple Whitted ray tracing (image produced by x-ray)

In order to be able to describe the simulation of various paths of light throughout a 3D environment, it is important to quickly review a very convinient notation introduced by Heckbert, called: "light path notation."[5]

Light path notation consists of a string of characters from the set $\{L, S, D, E\}$, which stands for *light*, *specular interaction*, *diffuse interaction*, and *eye* respectively. By using combinations of these letters in a sequence we can describe all possible interactions of light in an environment. For example a diffuse sphere seen through the specular reflection of a mirror, corresponds to a `LDSE` path, while a caustic formed on a table by light concentrated by a lens, corresponds to an `LSDE` path.

Heckbert also used regular expressions to describe classes of possible light paths, for example returning to our original discussion, a basic ray tracer can simulate zero or one diffuse reflections, followed by zero or more

specular interactions before arriving at the eye; such a class of paths can be denoted by the expression `LD?S*E`, while all possible light paths in a scene are really described by `L(D|S)*E`.

So, it becomes apparent that in order to properly simulate the interactions of light with an environment, one must go beyond the simple ray tracing algorithm.

2.1.1 Monte–Carlo Integration in Ray Tracing

In order to achieve its stated goal of photorealism, X-Ray uses monte carlo methods enhancments to the basic ray tracing algorithm, also called "distribution ray-tracing."[6] This enhancment makes it possible to simulate such effects as *soft shadows* produced by area lights, *glossy reflections* from rought surfaces, and *spatial antialiasing*[1].

Figure 2: Distributed or monte carlo ray tracing used for glossy reflection and soft shadows (image produced by x-ray)

Distribution ray-tracing works by numerically integrating various properties over their proper ranges, instead of just using a single sample. To

[1]temporal antialiasing is also possible with monte carlo ray tracing, but not currently implemented by the system

make this clear, consider the basic raytracing process of calculating reflections. A single ray is considered, whose direction is the reflected vector of the incoming ray. This produces only perfect-mirror reflections. Now if we are to instead sample a range of directions around the perfect reflection direction, as a function of surface roughness, essentially integrating over a solid angle of directions. We can obtain a more realistic "glossy" reflection.

2.1.2 Photon Mapping

Furthermore, X-Ray uses a cutting-edge technique called "photon mapping"[7] for capturing an otherwise very hard to simulate behaviour of light, caustics. Caustics are formed by light concentrated in a small area due to reflection or refraction from a highly specular object. A familiar example of caustics is the bright spot of light concentrated by passing through a lens, or a glass filled with liquid.

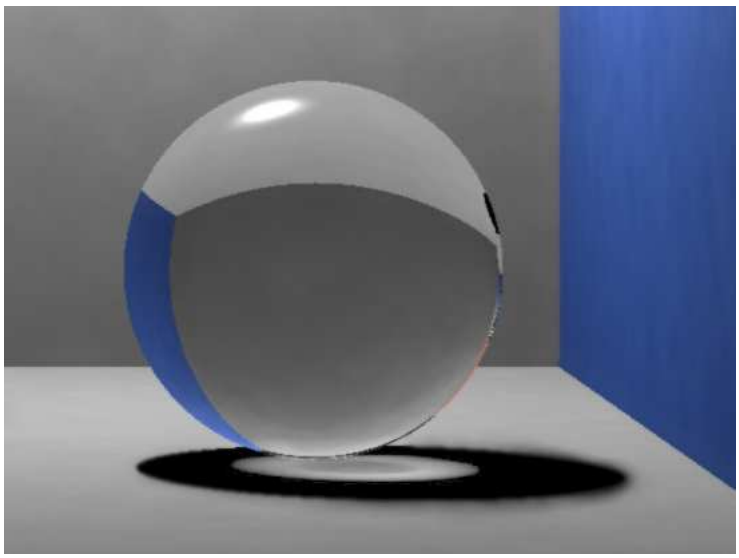

Figure 3: Caustics formed by concentrated light passing through a glass sphere (image produced by x-ray)

Photon mapping works by first tracing the paths of a big number of light particles (photons) througout the scene, and storing them at the end of their path as a 3D point cloud. It is extremely important to choose a data structure that provides very fast location of nearest-neighbor photons. In X-Ray we use a kd-tree[8] for that purpose, as proposed by Jensen[7].

Thus, through the use of advanced rendering techniques, X-Ray manages to compute almost all light paths in a virtual environment. More concisely, in light path notation, it can simulate all `LS*D?S*E` paths. The only thing it

can't handle properly right now is multiple diffuse reflections, which is been worked on currently (see future work section).

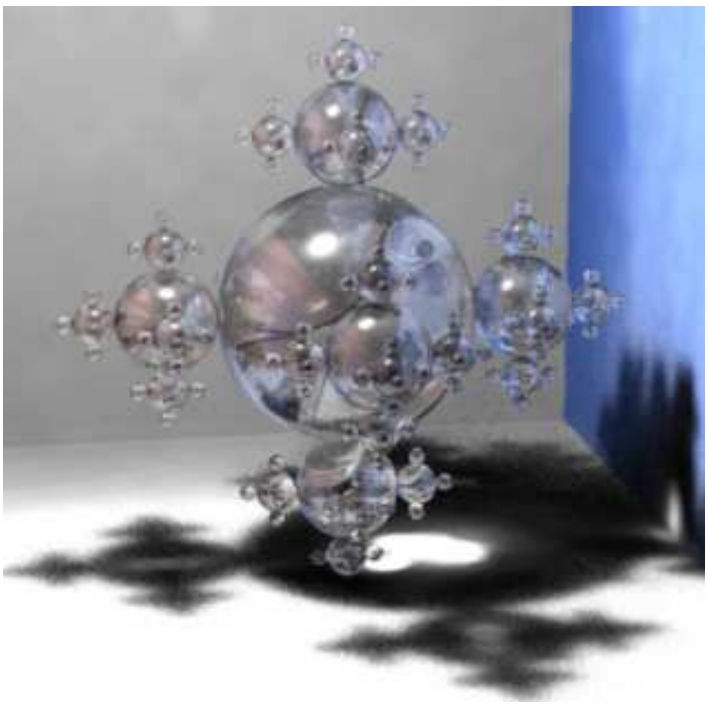

Figure 4: More complicated caustics produced by a glass sphereflake fractal (image produced by x-ray)

2.2 Extensibility – Programmable Shading

Another very important feature of X-Ray is the extensibility through programmable shading. It means that the user may provide a small program associated with any object of the environment, to be used for calculating illumination and driving the ray tracing process, after an intersection is encountered with that object.

This gives the power to the user to achieve very complex custom shading effects, implement novel reflectance models, or change the way rays are traced through the scene.

X-Ray achieves a unique combination of ease of use and shader execution performance by the use of on-demand shader compilation from C++ source files (see implementation section for more details).

Of course a number of built-in shaders are available for use with X-Ray, implementing the most commonly used reflectance models: Phong[9], Cook-Torrance[10], and Oren-Nayar[11].

3 X-Ray Design & Implementation

Figure 5 illustrates the high-level architecture of the renderer.

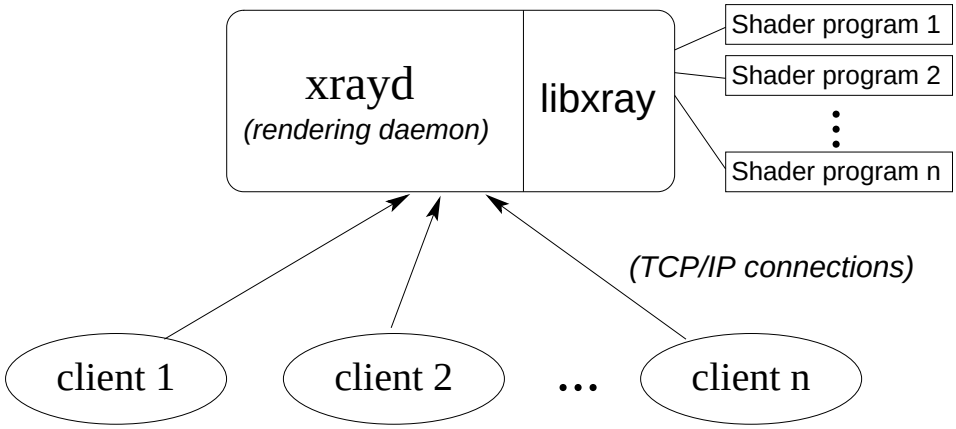

Figure 5: high level rendering system design

3.1 Libxray – The Core Library

The core of the renderer is implemented in the form of an easy to use reusable library, called "*libxray*" which may be linked with any program requiring high quality rendering and visualization. This aspect alone makes the system extremely flexible.

The library, provides all the necessary algorithms needed to perform photorealistic rendering, and it is really a manner of calling a few high-level functions for any program to be able to visualize a high quality 3D environment. Essentially, a program using libxray, needs only to instruct the library to read rendering options from a configuration file, call a function to read the scene definition, another function to perform the rendering, and two final calls to get the resulting pixels, and save them in an image file.

Even a more complicated program that would be able to utilize the full advanced data output capabilities of libxray to perform compositing, relighting, exposure control and other advanced processes, will only need a couple more calls to the library itself, and of course a lot of code to interface with the window system for presentation and user interaction, tasks which are out of the scope of our code.

The scene description understood by libxray is based on XML (a DTD is available provided for validation). XML was chosen in order to make it very easy to interoperate with other programs, such as 3D authoring tools, or renderer front-ends. Converters have been written for 3D Studio, Maya, PLY, and Collada data formats. And export plugins for various 3D

authoring tools can easily be written. Also, a realtime OpenGL preview program that reads the same scene descriptions is available.

Finally, the libxray, provides a C++ API to write user-defined shader programs, that drive the whole rendering process as described in the previous section.

The shader source files must contain an entry function "`shade`", which is called by the libxray core whenever during ray tracing, an intersection is encountered with an object whose material contains a reference to that shader.

The advantage of using compiled C++ shader programs as opposed to using an interpreted language for that purpose, is of course that the execution of the shaders is very fast (something extremely important as they will be called thousands of times for each rendered frame).

The disadvantage of using compiled shaders would be the difficulty of having to go through compilation and linking for every shader, plus the issue of the inherent non-portability of the compiled object code. X-Ray solves both these problems with on-demand shader compilation server-side. Whenever a shader is needed, the source file is retrieved, compiled, linked and cached automatically by the renderer. Thus lifting all unnecessary effort from the user, who only provides the portable source file to the renderer.

3.2 The xrayd Rendering Daemon

The actual rendering program, which uses libxray, is implemented in the form of a networked rendering daemon.

This choice is significant as it allows extremely simple rendering front-end programs to be written, tailored to the needs of a specific user / class of users, or organization. But most importantly, because it allows the actual work (which is very processor-intensive) to be performed on powerful dedicated rendering servers, accessible by the users who may be working on inexpensive workstations.

The communication between client and server is performed through TCP/IP with a custom mostly text-oriented protocol.

4 Future Work

X-Ray is always under development, and efforts are made to improve it in many ways.

The most important aspect where the original work was lacking, is runtime efficiency. Currently multiple ideas are been worked on for optimizing the rendering process such as:

- The use of various space partitioning schemes to accelerate intersection tests.

- Adaptive supersampling, or even adaptive subsampling for fust preview renderings, to avoid shooting an excessive amount of rays where it wouldn't make much difference due to low frequency detail.
- Fast hardware rasterization pass using OpenGL, to identify primary ray hits.

Appart from performance optimizations, work is being done to improve the photorealistic rendering capabilities of X-Ray. The major goal is to simulate all possible light paths through the environment efficiently. Photon mapping can be used to achieve this, in conjunction with "path tracing"[12], which is the approach currently been worked on. Another method to achieve this result is the use of "radiosity"[13].

5 Conclusion

To conclude, X-Ray is a multiplatform, advanced photorealistic renderer, providing a wide range of features comparable to big commercial renderers, while being licensed under free software terms (GNU General Public License), in order to be of maximum use to the scientific community. It is extremely flexible and can be used as a platform to support further research in the field of computer graphics, as well as the means to produce high quality animations and special effects.

Figure 6: Part of the core rendering code, running on the Nintendo GameBoy Advance, demonstrating its high degree of portability.

References

[1] W. Turner, "An improved illumination model for shaded display," *CACM, 1980*, vol. 23, no. 6, pp. 343–349, 1980.

[2] "Pov-ray web site." [Online document], [cited 2005 Dec 14], Available HTTP: http://www.povray.org.

[3] "Mental ray web site." [Online document], [cited 2005 Dec 14], Available HTTP: http://www.mentalimages.com.

[4] T. Apodaca, "The RenderMan interface," *j-BYTE*, vol. 14, pp. 167–176, Apr. 1989.

[5] P. S. Heckbert, "Adaptive radiosity textures for bidirectional ray tracing," in *Proceedings of SIGGRAPH 1990*, vol. 24, pp. 145–154, Aug. 1990.

[6] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *Proceedings of SIGGRAPH 1984*, pp. 137–145, July 1984.

[7] H. W. Jensen, *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, 2001.

[8] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *CACM, 1975*, vol. 18, no. 9, pp. 509–517, 1975.

[9] B. T. Phong, *Illumination for computer-generated images.* PhD thesis, Dept. of Electrical Engineering, University of Utah, 1973.

[10] R. L. Cook and K. E. Torrance, "A reflectance model for computer graphics," in *Proceedings of SIGGRAPH 1981*, Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, 1981.

[11] M. Oren and S. K. Nayar, "Generalization of lambert's reflectance model," in *Proceedings of SIGGRAPH 1994*, Computer Graphics Proceedings, Annual Conference Series, ACM, ACM Press / ACM SIGGRAPH, 1994.

[12] J. T. Kajiya, "The rendering equation," in *Proceedings of SIGGRAPH 1986*, pp. 143–150, ACM Press / ACM SIGGRAPH, 1986.

[13] R. Siegel and J. R. Howel, *Thermal Radiation Heat Transfer*. Hemisphere Publishing Corp., 1981.