Rendering

Erik Reinhard[1,2] Erum Arif Khan[2] Ahmet Oğuz Akyüz[2]

1 Introduction

In the real world, light sources emit photons which normally travel in straight lines until they interact with a surface or a volume. When a photon encounters a surface, it may either be absorbed, reflected or transmitted. Some of these photons may hit the retina of an observer where they are converted into a signal which is then processed by the brain, thus forming an image. Similarly, photons may be caught by the sensor of a camera. In either case, the image is a 2D representation of the environment.

The formation of an image as a result of photons interacting with a 3D environment may be simulated on the computer. The environment is then replaced by a 3D geometric model and the interaction of light with this model is simulated with one of a large number of algorithms. The process of image synthesis by simulating light behavior is called *rendering*.

As long as the environment is not altered, the interaction of light and surfaces gives rise to a distribution of light in a scene which is in equilibrium, i.e. the environment does not get lighter or darker.

Since all rendering algorithms model the same process, it is possible to summarize most rendering algorithms by a single equation, which is known as the *rendering equation*. The underlying principle is that each point $\mathbf{x}$ on a surface receives light from the environment. Some of this light is absorbed and some is reflected. The light that falls on a point on a surface may be coming directly from a light source, or may have been reflected one or more times by other surfaces.

Considering a point $\mathbf{x}$ on some surface that receives light from all directions, the material of the surface determines how much of this light is reflected, and in which directions. The reflective properties of a surface are also dependent on wavelength, which gives each surface its distinctive color. A material may therefore be modeled using a function which describes how much light incident on a point on a surface is reflected for each incoming and each outgoing direction. Such functions are generally known as *bi-directional reflectance distribution functions* (BRDFs), and are denoted here as $f_r(\mathbf{x}, \Theta_i, \Theta_o)$. This function is dependent on the position on the surface $\mathbf{x}$, as well as the angle of incidence Θ_i and the outgoing direction Θ_o.

To determine how much light a surface reflects into a particular direction, we can multiply the BRDF for each angle of incidence with the amount of incident light $L_i(\mathbf{x}, \Theta_i)$ and integrate these pairwise multiplications. This yields a quantity for one specific outgoing direction.

A point on a surface may also emit light, which is denoted with a non-zero term $L_e(\mathbf{x}, \Theta_o)$. This term is dependent on position on the surface (e.g. a television screen emits light which is spatially varying in intensity), and may also be directionally varying (e.g. spot lights emit more light in some directions than in others).

Thus, the amount of light that leaves a point $\mathbf{x}$ on a surface in a particular direction Θ_o may be modeled as follows:

$$L_o(\mathbf{x}, \Theta_o) = L_e(\mathbf{x}, \Theta_o) + \int_{\Omega_i} g(\mathbf{x}, \Theta_i, \Theta_o)\, d\,\omega_i \qquad (1)$$

$$g(\mathbf{x}, \Theta_i, \Theta_o) = f_r(\mathbf{x}, \Theta_i, \Theta_o)\; L_i(\mathbf{x}, \Theta_i)\; \cos\Theta_i \qquad (2)$$

This equation is known as the *rendering equation*. To compute how much light is reflected into a particular direction, we need to integrate over all incident directions (a hemisphere of directions Ω_i if we assume that the surface is not transparent). Thus, the above equation will have to be recursively evaluated for each point in the environment that is visible from $\mathbf{x}$.

To compute an image by simulating light in the above manner, we would have to evaluate the rendering equation for each pixel separately (multiple times if we were to apply *anti-aliasing* in the process). It should be clear that the number of computations required to evaluate this equation even once is astronomical. For practical problems, the computational cost of evaluating the rendering equation directly is too high. However, there are many ways to simplify this equation, for example by removing parts of the computation that do not contribute significantly to the final solution.

It is for instance possible to only account for the direct contribution of light sources, and ignore all reflected light. Such algorithms fall in the class of *local illumination* algorithms. If indirect illumination, i.e. illumination after one or more reflections or transmissions, is accounted for, then we speak of *global illumination* algorithms.

Finally, the rendering equation is known as a Fredholm equation of the second kind, which implies that no analytical solutions are known. We therefore have to resort to numerical approximations to evaluate the rendering equation. In particular, this equation is routinely discretized, turning its evaluation into a sampling problem.

In summary, rendering involves the creation of images by simulating the behavior of light in artificial scenes. Such scenes consist of descriptions of surfaces and light sources (the *geometry*). In addition to having a position in space and a particular shape, surfaces are characterized by the manner in which they interact with light (*material properties*). In the following sections, geometry and materials are discussed in greater detail, followed by a brief explanation of the more prominent local and global illumination algorithms.

2 Geometry

The shape an object can be modeled with a collection of simple primitives, including polygon and triangle meshes, spline surfaces, and point based representations. Geometric representations can either be modeled by hand using modeling software such as Alias Wavefront, or objects can be scanned with a laser scanner.

A frequently used representation is a mesh (Figure 1). A mesh is made up of one or more simple polygonal shapes, for example triangles. Some polygons share boundaries with other polygons in the mesh and together produce the structure of the object. Of course, polygons will only approximate the shape of the actual object. The larger the number of polygons used, the closer the approximation will be to the actual shape of the object. The number of polygons

[1]University of Bristol, reinhard@cs.bris.ac.uk
[2]University of Central Florida

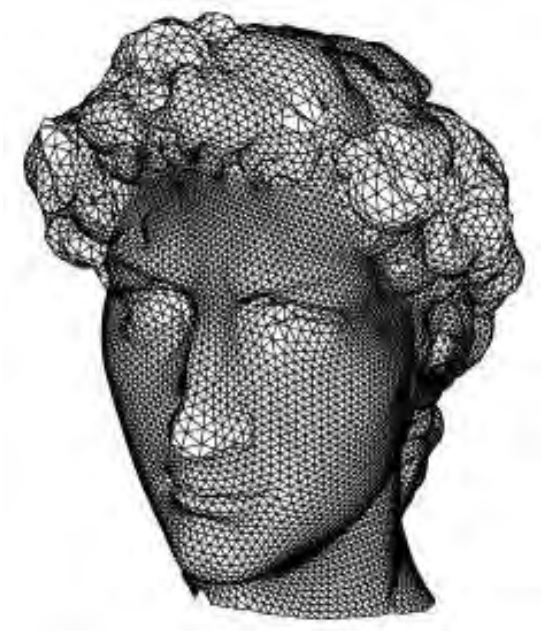

Figure 1: *Mesh representation of a head.*

also determines the time it takes to render the object, thereby affording a trade-off between quality and computation time.

For efficiency purposes, large meshes may be reduced in size. One technique to reduce the number of polygons representing an object's surface is *displacement mapping*. In this technique, the surface of the object is represented by fewer, larger polygons, and the small scale features are captured in a depth map. Points on the surface, which is represented by polygons are then displaced according to the displacement map. In this way, the object shape retains fine features, but the number of polygons used to represent the object is smaller.

Another way of reducing rendering time is to use *level of detail* algorithms. These algorithms ensure that the object is represented by as many primitives as necessary, dependent on distance to the viewer. If the object is far from the viewpoint, most of the fine details will not be visible and thus the object's shape may be represented by fewer polygons. If the viewpoint approaches the object, the object needs to be represented by a larger number of polygons so that the fine scale features, which are now visible, are adequately visualized.

The shape of an object may also be described by parametric equations such as Bézier curves and B-splines. The parametric surface has certain advantages over the simpler polygonal model. First, the representation is much more concise. If an object has fine features, the mesh will require many polygons to represent the object. However, patches of the same surface, when represented parametrically, will be fewer. This is due to the fact that each patch can represent a curved surface segment, whereas triangles and polygons are flat.

Scanners can be used to determine the shape of existing objects. The output from a scanner is a dense set of points. Typically these points define the vertices of a triangle mesh. Relatively recently, algorithms have been developed to render point clouds directly, obviating the need for triangulation. This approach also lends itself to simpler level of detail algorithms since altering the number of points is more straightforward than altering the number, size and shape of polygons or patches representing the object shape.

Figure 2: *An example of an object rendered with a translucent material.*

3 Materials

The micro-structure of the object determines the way light interacts with it, and hence it determines the appearance of the object. This micro-structure is represented by a material description, such as the BRDF f_r introduced in Section 1.

If a surface scatters light equally in all directions, we call the material *diffuse* or *Lambertian*, leading to a BRDF which is a constant function, i.e. $f_r = \rho/\pi$, where ρ is a measure of how much light is reflected. Other materials may reflect light preferentially as function of the direction of incidence. For instance, a mirror reflects almost all light in the reflected direction. Inbetween lie glossy materials which scatter light into a cone centered around the direction of mirror reflection.

The angles Θ_i and Θ_o can each be decomposed into an angle ϕ in the plane of the surface, and an azimuthal angle θ, for instance $\Theta_i = (\theta_i, \phi_i)$. If the material's reflective properties depend only on θ_i, θ_o and $\phi_i - \phi_o$ then reflections are invariant to rotation around the surface normal, and the material is called isotropic. On the other hand, if f_r depends on θ_i, θ_o, ϕ_i, and ϕ_o, rotation around the surface normal will alter the reflection, and the material is called anisotropic (brushed aluminium is an example).

Real materials can be measured, or BRDFs may be modeled empirically. In the latter case, reciprocity and conservation of energy are considered important features of any plausible BRDF. Reciprocity refers to the fact that f_r should return the same result if Θ_i and Θ_o are reversed. Conservation of energy means that light is either reflected or absorbed, but not lost in any other way.

Extensions to basic BRDFs include models for transparency (e.g. glass), translucency, and spatial variance. Translucency stems from light scatter inside a surface, as shown in Figure 2. Wax, skin, fruit and milk, all display some degree of translucency. An example of a spatially varying material is woodgrain. Normally, *texture mapping* is used to account for this. A texture map can be created by taking a photograph of the desired material, and then this texture is mapped onto the surface of the object. Texture maps and BRDFs may be combined to yield spatially variant BRDFs, or *Bidirectional Texture Functions* (BTFs).

4 Local Illumination

Images may be rendered by projecting all the geometry onto a plane that represents the screen in 3D space, thus implementing a local illumination model. For each pixel, the nearest object may be tracked using a *z-buffer*. This buffer stores for each pixel the distance between the view point and the currently nearest object. When a new

object is projected, its distance is tested against the distances stored in the z-buffer. If the new object is closer, it is drawn and the z-buffer is updated. The color assigned to the pixel is then derived from the object's color using a simple shading algorithm.

The simplicity of projective algorithms makes them amenable to hardware implementation. As a result most graphics cards implement a graphics pipeline based on z-buffering. To maximize performance, geometry is typically limited to simple shapes such as triangle and polygonal meshes. Only simple materials are supported.

However, modern graphics cards incorporate two programmable stages which allow vertices and pixels to be manipulated respectively. This provides flexibility in an otherwise rigid hardware environment. Programming these two stages is achieved through APIs such as *OpenGL* or *DirectX*. The (limited) ability to program graphics cards has given rise to many extensions to the basic z-buffer algorithm, such as *shadow maps* which compute shadows.

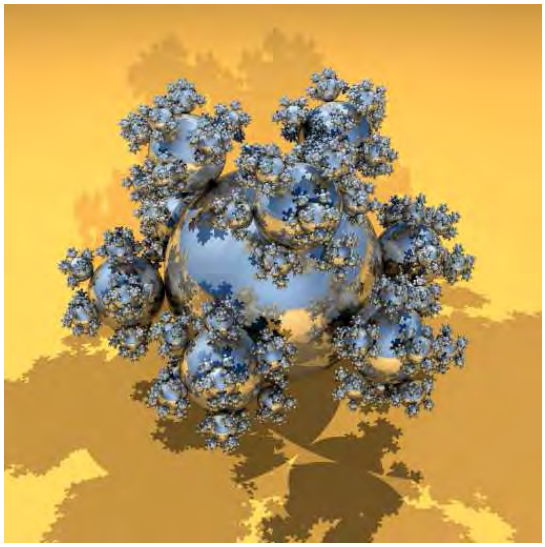

Figure 3: *A typical ray traced image, consisting of reflective spheres and sharp shadows.*

5 Ray Tracing and Ray Casting

One of the basic operations in rendering is to compute which (part of an) object is visible from a given point in space and a given direction. Such sampling of the scene is often accomplished with a technique called *ray casting*. A ray is a half-line starting at a specified point in space (its *origin*) and aimed at a particular *direction*. There may be many objects located along the line of sight of such a ray, and to compute which object is closest to the ray origin, the ray is intersected with each object. The point on the surface of the nearest object where the ray intersects, is called the *intersection point*. Functions for ray intersection calculations are available for a wide variety of geometric primitives, including triangles, polygons, implicit surfaces, and splines — a distinct advantage of any ray-casting based algorithm.

An image may be created of a scene by specifying a camera position, and casting (primary) rays starting at this position into different directions associated with the pixels that make up the image. This process computes for each pixel the nearest object. The color of the nearest object is then assigned to its corresponding pixel. Such a ray caster may be extended to a full ray tracer by also shooting secondary rays. These rays start at the intersection points of the primary rays and are aimed into specific directions based on which type of lighting effect is desired.

For instance, rays may be traced from an intersection point towards the light sources. Such shadow rays are useful for computing shadows, since the shading of the intersection point can be adjusted based on whether it was in shadow or not.

If an intersection point belongs to an object with a specular material, an additional ray may be shot into the reflected direction. This direction is computed by mirroring the incident ray into the *surface normal*, a vector which specifies the surface orientation at the intersection point. The reflected ray is then recursively traced and its returned color is assigned to the intersection point, which is in turn used to color a pixel. The same procedure is followed for transmitted rays in the case of transparent objects. A typical ray tracing example is shown in Figure 3.

Thus, ray tracing is a recursive algorithm based on casting rays. Starting from the view point, it is called *eye ray tracing*. It is a relatively straightforward way to evaluate a simplified version of the rendering equation (1), known as the ray tracing equation:

$$
\begin{aligned}
L_o(\mathbf{x}, \Theta_o) = {} & L_e(\mathbf{x}, \Theta_o) + \\
& \sum_{L} \int_{\mathbf{x}_l \in L} v(\mathbf{x}, \mathbf{x}_l)\; f_{r,d}(\mathbf{x})\; L_e(\mathbf{x}_l, \Theta'_o)\; cos\,\Theta_l\; d\omega_l + \\
& \int_{\Theta_s \in \Omega_s} f_{r,s}(\mathbf{x}, \Theta_s, \Theta_o)\; L(\mathbf{x}_s, \Theta_s)\; cos\,\Theta_s\; d\omega_s + \\
& \rho_d(\mathbf{x})\; L_a(\mathbf{x})
\end{aligned}
$$

The four terms on the right hand side are the emission term, followed by a summation of samples shot toward the light sources. The visibility term $v(\mathbf{x}, \mathbf{x}_l)$ is 1 if position $\mathbf{x}_l$ on the light source is visible from point $\mathbf{x}$ and 0 otherwise. The integration in the second term is over all possible positions on each light source. The third term accounts for specular reflections and the fourth term is the *ambient term* which is added to account for everything that is not sampled directly.

Thus, in ray tracing only the most important directions are sampled, namely the contributions of the light sources and mirror reflections. This represents a vast reduction in computational complexity over a full evaluation of the rendering equation, albeit at the cost of a modest loss of visual quality. Finally, ray tracing algorithms can now run at interactive rates, and under limited circumstances even in real-time.

6 Radiosity

Both ray tracing and the local illumination models discussed earlier are view-point dependent techniques. Thus, for each frame all illumination will be recomputed. This is desirable for view-point dependent effects, such as specular reflection. However, diffuse reflection does not visibly alter if a different viewpoint is chosen. It is therefore possible to preprocess the environment to compute the light interaction between diffuse surfaces. This may be achieved by employing a radiosity algorithm (Figure 4). The result can then be used to create an image, for instance by ray tracing or with a projective algorithm.

Figure 4: *An early example of a scene preprocessed by a radiosity algorithm.*

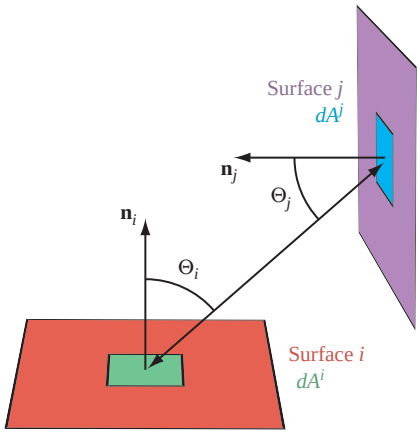

Figure 5: *Geometric relationship between two patches.*

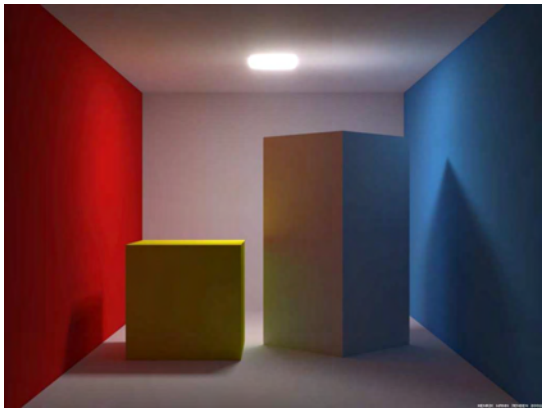

Figure 6: *Example of color bleeding. In particular the gray object on the right has a tinge of yellow and blue, caused by light that was first reflected off the yellow and blue surfaces.*

The surfaces in the environment are first subdivided into small patches. For computational efficiency it is normally assumed that the light distribution over each patch is constant. For small enough patches this is a fair assumption. Each patch can receive light from other patches, and then diffusely reflect it. A common way to implement radiosity is to select the patch with the most energy and distribute this energy over all other patches, which then gain some energy. This process is repeated until convergence is reached. Since all patches are assumed to be diffuse reflectors, the rendering equation (1) can be simplified to not include the dependence on outgoing direction Θ_o:

$$L(\mathbf{x}) = L_e(\mathbf{x}) + \rho_d(\mathbf{x}) \int_{\mathbf{x}} L(\mathbf{x}') \frac{\cos\,\Theta_i\,\cos\,\Theta'_o}{\pi\,\|\mathbf{x}'-\mathbf{x}\|^2}\, v(\mathbf{x},\mathbf{x}')\, dA'$$

where $v(\mathbf{x},\mathbf{x}')$ is the visibility term, as before. This equation models light interaction between pairs of points in the environment. Since radiosity operates on uniform patches rather than points, this equation can be rewritten to include a *form factor* F^{ij}, which approximates the fraction of energy leaving one patch and reaching another patch:

$$L^i = L^i_e + \rho^i_d \sum_j L^j F^{ij}$$

$$F^{ij} = \frac{1}{A^i} \int_{A^i}\int_{A^j} \frac{\cos\,\Theta_i\,\cos\,\Theta_j}{\pi\, r^2}\, \delta^{ij}\, dA^j\, dA^i$$

where the visibility term v between points is replaced with δ^{ij}, which denotes the visibility between patches i and j. The form factor depends on the distance between the two patches, as well as their spatial orientation with respect to one another (Figure 5). In practice the computation of form factors is achieved by ray casting.

The radiosity algorithm can be used to model diffuse interreflection, which accounts for visual effects such as *color bleeding* (the colored glow that a surface takes on when near a bright surface of a different color, as shown in Figure 6).

7 Monte Carlo Sampling

The integral in the rendering equation (1) may be evaluated directly. However, since both the domain Ω_i and the integrand (2) are complex functions, a very large number of samples would be required to obtain an accurate estimate. To make this sampling process more efficient, a stochastic process called Monte Carlo sampling may be employed. The environment is then sampled randomly according to a *probability density function* (pdf) $p(\omega_i)$:

$$\int_{\Omega_i} g(\omega_i,\Theta_i,\Theta_o)\, d\omega_i \approx \frac{1}{N}\sum_{i=1}^{N} \frac{g(\omega_i,\Theta_i,\Theta_o)}{p(\omega_i)}$$

The number of sample points N can be set to trade speed for accuracy. Typically, to evaluate each $g(\omega_i,\Theta_i,\Theta_o)$, a ray is traced into the environment. For efficiency, the pdf should be chosen to follow the general shape of the integrand $g(\omega_i,\Theta_i,\Theta_o)$. There are many ways to choose the pdf, a process known as *importance sampling*.

In addition it is possible to split the integral into disjunct parts for which a simpler pdf may be known. This process is called *stratified sampling*. One could view ray tracing as a form of stratified sampling, since instead of sampling a full hemisphere around each intersection point, rays are only directed at the light sources and the reflected and transmitted directions. Both importance sampling and stratified sampling will help reduce the number of samples N required for an accurate evaluation of the rendering equation (1).

8 Photon Mapping

Certain complex types of illumination such as the caustic patterns created by light refracted through transparent objects are not effi-

Figure 7: *Light refracted through the transparent glass creates a caustic on the table.*

ciently sampled by Monte Carlo sampling alone. Rendering algorithms such as ray tracing, radiosity as well as local illumination models expressly omit the sampling that would be required to capture caustics.

To enable the rendering of caustics, shown in Figure 7, as well as make rendering of other light interactions such as diffuse interreflection more efficient, photons may be tracked starting at the light source (known as *photon ray tracing*), rather than tracing photons backwards starting at the viewpoint (as in eye ray tracing). They can then be deposited on diffuse surfaces after having undergone one or more refractions through dielectric (transparent) objects. Thus, photons are stored in a data structure called a *photon map*, which represents the distribution of light over the surfaces in an environment.

An image may then be created using conventional ray tracing. Whenever an intersection with a diffuse surface is detected, the photon map is used to determine how much light is present at the intersection point. The photon map may therefore be seen as a data-structure to connect the initial light pass with the subsequent rendering pass.

Regarding efficiency, photon maps need to be created only once as long as the scene is static. The rendering pass can be repeated for any desired view point.

9 Image-Based Rendering

To avoid the expense of modeling a complicated scene, it is sometimes more convenient to photograph a scene from different viewpoints. To create images for novel viewpoints that were not photographed, an interpolation scheme may be applied. Rendering using images as a modeling primitive is called *image based rendering*. Such techniques attempt to compute a continuous representation of the *plenoptic* function, given some discrete representation of it. The plenoptic function is defined as the intensity of light rays passing through the camera center at every camera location (V_x, V_y, V_z), orientation (θ, ϕ), and for every wavelength (λ) and time (t), that is:

$$P_7 = P(V_x, V_y, V_z, \theta, \phi, \lambda, t) \tag{3}$$

Thus, the plenoptic function may be considered a representation of the scene, such that, when input parameters like camera location and orientation are altered, the scene represented by the function changes accordingly. Simplified versions of the plenoptic function exist. For instance, if we assume that the environment is constant, we may remove the parameter t. The simplest plenoptic function is

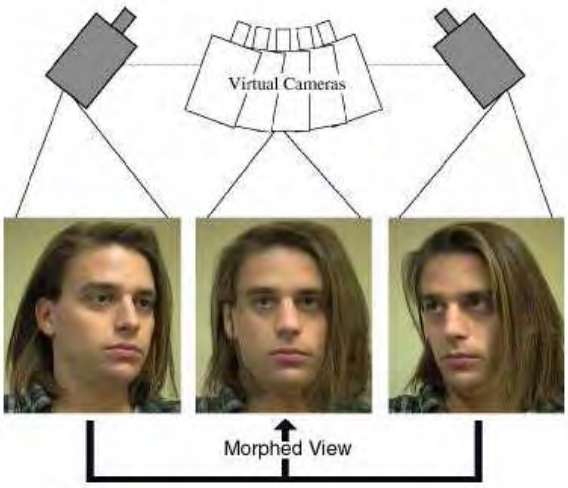

Figure 8: *Two images are used to produce a third using Image based rendering.*

a 2D panoramic view of the scene with a fixed viewpoint. θ and ϕ are the only two input parameters in this case.

If instead of a panoramic view, we captured several images that are a part of this panoramic view, then these images would be a discrete representation of the plenoptic function. Image based rendering techniques take these discrete representations as input, and provide a continuous representation, for example the complete panoramic view in the above case. A technique might take two images with different viewpoints as input and produce a set of images that have viewpoints that lie in between the two original viewpoints.

There are many image based rendering techniques, and they may be broadly classified into three groups. The first group requires complete information of scene geometry, for example in the form of a depth map of the scene. This information along with one or more images is sufficient to render scenes from a viewpoint close to the viewpoint of the given image(s). 3D warping techniques belong to this category.

The second group of image based rendering techniques uses only input images of the scene to render another image of the same scene from a different view point. There is no reliance on any given information of the scene geometry. Examples include *light field rendering* and *lumigraph* systems.

The third group lies somewhere in between the previous groups. This group requires several input images as well as further geometric information in the form of correspondence features in the two images (for example points). Given this correspondence, the scene may be rendered from all viewpoints between the two viewpoints of the original input images. *View morphing* (Figure 8) and *interpolation* techniques fall under this category.

Images may also be used to represent the lighting of the scene alone, while geometry and materials represented directly. This process is called *Image based lighting* (IBL, see Figure 9. Here, the first step is to create the image that will represent the lighting of the scene. An image of a mirrored ball placed in the scene may be used to represent this lighting. Images typically have a limited range of pixel values (0 to 255), which cannot represent the lighting of an arbitrary scene. *High dynamic range* (HDR) images are used instead as their pixel values are not limited to 256 values and are proportional to the actual illumination of the scene. The captured image is then mapped to a sphere and the object is placed within it before rendering.

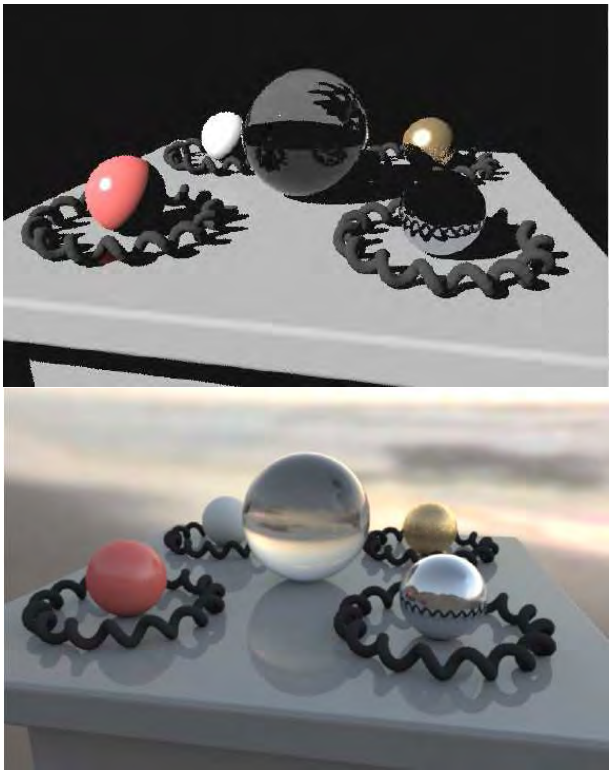

Figure 9: *Scene rendering without image based lighting (top), and with image based lighting (bottom).*

10 Further Reading

Rendering is an important part of the field of computer graphics. There are many excellent books, as well as a vast number of papers. General graphics books are [Shirley et al. 2005; Pharr and Humphreys 2004; Glassner 1995; Foley et al. 1990; Watt and Watt 1992]. Books specifically for ray tracing are [Glassner 1989; Ward Larson and Shakespeare 1998; Shirley and Morley 2003; Suffern 2007]. Global illumination is covered in [Dutré et al. 2003]. Radiosity is explained in detail in [Cohen and Wallace 1993; Sillion and Puech 1994; Ashdown 1994]. Photon mapping is described in [Jensen 2001]. For local illumination models as well as using the OpenGL API, see [Hearn and Baker 2004]. Image-based lighting is a relatively new rendering technique, described in [Reinhard et al. 2005]. For real-time rendering, see [Akenine-Möller and Haines 2002]. Parallel rendering is covered in [Chalmers et al. 2002]. The notation used for the equations in this article are based on Arjan Kok's thesis [Kok 1994].

The latest research on rendering is published in a variety of forums. The most relevant conferences are ACM SIGGRAPH, the Eurographics Symposium on Rendering and the Eurographics main conference. In addition, several journals publish rendering papers, such as ACM Transactions on Graphics, IEEE Transactions on Visualization and Computer Graphics, Eurographics Forum and the Journal of Graphics Tools.

Acknowledgments

We thank Matt Pharr, Henrik Wann Jensen, Steve Seitz, Paul Debevec, and Andrei Khodakovsky for kindly allowing us to reproduce some of their images.

References

AKENINE-MÖLLER, T., AND HAINES, E. 2002. *Real-time Rendering*, 2nd ed. AK Peters, Natick, MA.

ASHDOWN, I. 1994. *Radiosity: A Programmer's Perspective*. John Wiley & Sons, Oct.

CHALMERS, A., DAVIS, T., AND REINHARD, E., Eds. 2002. *Practical Parallel Rendering*. AK Peters, Natick, MA.

COHEN, M. F., AND WALLACE, J. R. 1993. *Radiosity and Realistic Image Synthesis*. Academic Press, Inc., Cambridge, MA.

DUTRÉ, P., BEKAERT, P., AND BALA, K. 2003. *Advanced global illumination*. A K Peters, Natick, MA.

FOLEY, J., VAN DAM, A., FEINER, S., AND HUGHES, J. 1990. *Computer graphics, principles and practice*, 2^{nd} ed. ed. Addison-Wesley.

GLASSNER, A. S., Ed. 1989. *An Introduction to Ray Tracing*. Academic Press, San Diego.

GLASSNER, A. S. 1995. *Principles of digital image synthesis*. Morgan Kaufmann, San Fransisco, CA.

HEARN, D., AND BAKER, M. P. 2004. *Computer graphics with OpenGL*, 3rd ed. Pearson Prentice Hall, Upper Sadle River, NJ.

JENSEN, H. W. 2001. *Realistic Image Synthesis using Photon Mapping*. A K Peters, Natick, MA.

KOK, A. J. F. 1994. *Ray Tracing and Radiosity Algorithms for Photorealistic Image Synthesis*. PhD thesis, Delft University of Technology, The Netherlands. Delft University Press, ISBN 90-6275-981-5.

PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering*. Morgan Kaufmann Publishers, San Fransisco.

REINHARD, E., WARD, G., PATTANAIK, S., AND DEBEVEC, P. 2005. *High Dynamic Range Imaging: Acquisition, Display and Image-Based Lighting*. Morgan Kaufmann Publishers, San Francisco.

SHIRLEY, P., AND MORLEY, R. K. 2003. *Realistic Ray Tracing*, 2nd ed. A K Peters, Natick, Massachusetts.

SHIRLEY, P., ASHIKHMIN, M., MARSCHNER, S. R., REINHARD, E., SUNG, K., THOMPSON, W. B., AND WILLEMSEN, P. 2005. *Fundamentals of Computer Graphics*, 2nd ed. A K Peters, Natick, Massachusetts.

SILLION, F. X., AND PUECH, C. 1994. *Radiosity and Global Illumination*. Morgan Kaufmann Publishers, Inc., San Francisco, California.

SUFFERN, K. 2007. *Ray Tracing from the Ground Up*. A K Peters, Natick, MA.

WARD LARSON, G., AND SHAKESPEARE, R. A. 1998. *Rendering with Radiance*. Morgan Kaufmann Publishers.

WATT, A., AND WATT, M. 1992. *Advanced Animation and Rendering Techniques, Theory and Practice*. Addison-Wesley, Wokingham, UK.