A Generic Interface to Colors, Materials, and Textures

Wolfgang Stürzlinger

GUP, Johannes Kepler Universität, Altenbergerstr.69, A-4040 Linz, Austria/

ABSTRACT

Material models predict the reflection of light by surfaces in a scene. Many models have been proposed, all with different scope of effects and different parameters. The same holds for color models and texture definitions. Most computer graphics systems implement only one model with a minimal interface. If a different model is to be integrated this interface needs to be restructured which results in many code changes in the whole system.
This work presents a new approach to a generic interface to colors and materials. By enumerating all possible operations on these graphics objects an interface for reusable and portable code is defined. A seamless integration of textured surfaces into a computer graphics system is made possible by generic texture trees which implement the generic material interface. This portable definition of textures is realized via a tree of texture nodes and texture generators.

Key Words: Material Model, Color Model, Texturing, Generic Interface

INTRODUCTION

One of the main topics in computer graphics is the generation of images. The light reflected by the scene is computed and the user can see the resulting image on an output medium. Light is described by an electromagnetic spectrum, only the visible portion (perceived as color) is relevant for computer graphics. The most widely used color representation is the RGB model, although no standards exist for it. For an overview of color models see [Hall89]. Material models are used to simulate how light is reflected by a surface when it is submitted to illumination. Various material models have been proposed (e.g. [He et al.91], [Schlick94], [Strauss92], [Ward92]), for a recent overview see [Glassner95]. Texture is used to simulate the structure of non-homogeneous material such as wood. Texture functions define the color depending on surface position. Other methods map images onto surfaces. See [Watt-Watt92] for an introduction.
Most computer graphics system use only one color and material model internally and implement their own texturing system. Additionally almost all systems use different interfaces. This makes it hard to share code between different applications and makes the exchange of color and material data and texture definitions problematical. To achieve independence of a program working with graphical entities (such as scene objects, colors, materials, textures, images, and so on) from the underlying model, a generic interface including all possible operations on these entities has to be defined.

Related Work

An approach for a generic graphics kernel is presented in [Beier-Bozetti95] discussing generic object and image interfaces, but colors, materials, and textures are not included.
Shade trees were previously proposed [Cook84] allowing the user to specify how materials and textures react to illumination. The leaves of the tree are values pertinent to the shading process. The nodes operate on these values and the root returns a color as result. A different approach to the handling of multiple material models and textures was taken by the RenderMan System [Upstill89] using a special programming lan-

guage to create custom shaders. An implementation of a rendering system able to handle a few different material definitions was presented by the author in [Stürzlinger et al.93], but this approach was limited to the one color model and considered only diffuse and specular reflection. Adding textures was not discussed at all. Recently an open rendering kernel was presented by Slussalek and Seidel [Slussallek-Seidel96].

Motivation

Each color and material model uses different parameters and allows for different effects. This makes it hard to reuse previous implementations and share code between different applications. Consider the following simple example to illustrate the desirability of a generic interface to color. An old ray tracing system works internally with the RGB model and the code is littered with statements such as:

```
for (i = 0; i < 3; i++)
    color1[i] *= color2[i];
```

Imagine transforming this code to a color representation which uses four samples in the visible spectrum and uses natural splines to interpolate in between for better accuracy.

One of the biggest shortcomings of texture is that no standard definition exists and every graphics system implements its own version. Especially the area of texture functions is rich with different definitions for the same material. E.g. for wood textures a multitude of functions was proposed and even more variants have been implemented in rendering systems. Also no consensus exists on the correct combination of material models and texturing. Some realisations modify only the diffuse color of a material whereas others modify also the specular component.

The goal of this work is to define a generic interface to color and material models allowing also for textured surface. A generic interface defines all applicable functions which must be implemented by each different model independent of the underlying representation. Then a computer graphics system can use rely on this interface to access the graphics entities. The generic interface facilitates the reuse of code for color, material, and texture descriptions in different systems written by different people. Additionally it provides a basis for the definition of portable and reusable texture definitions.

GENERIC COLOR AND MATERIAL INTERFACE

Material models typically include color specifications, at least to describe the color of the surface. Light interaction with materials must be defined, and all corresponding computations must be hidden by a generic interface. To achieve independence of the material model from the underlying color model a generic interface to color has to be defined.

Generic Color Interface

The light hitting a surface is described by an electromagnetic spectrum. The material of the surface reflects different wavelengths of the incoming light differently which creates the sensation of color to the human observer. How much a given material reflects for each wavelength is described by a reflectance spectrum. Light emitted by a light source is described by an emission spectrum. In computer graphics (and everyday life) only the visible part is of relevance and the representation of the reflectance/emission spectrum is described by a color model. The most widely used color representation is the RGB model, although it is not standardized. The most general form is the representation of the spectrum by samples at nanometer (nm) intervals. Other color models are used, for a recent overview see [Glassner95], [Hall89].

To achieve independence of a program working with materials representations from the underlying color model, a generic interface including all relevant operations on colors has to be defined. One of the most basic operations is the summation of light coming from multiple independent light sources realized by adding their emission spectra. The simulation of reflection by weighting the incoming light by the reflectance curve of the material, and a scaling operation to be able to account for the fall-off of the reflected light depending on the surface orientation are also important. A special operation is the calculation of the relative wavelength dependent refraction index of the interface between two materials. Another phenomenon is the exponential decay along the length of a light ray in a homogeneous medium. Many rendering systems optimize calculations when the contributions fall below a certain threshold. Therefore the brightness of a color is important. To allow the conversion of different color representations

the conversion from and to a spectrum sampled at nanometer (nm) intervals is needed.

Therefore the generic color interface defines the following operations:

- `+:` Adding two colors.
- `*:` Multiplication of two color representations to simulate reflection.
- `*:` Scaling a color by a scalar value.
- `/:` Division of two color representations to obtain the relative refraction index.
- `pow:` Exponential function with a scalar exponent.
- `brightness:` Converting the color to a brightness value with 0 defined as black.
- `ColorSpectrumNM:` Conversion from/to spectrum sampled at nm intervals.

As available in most object oriented languages operators are defined for easier usage. The following operations are either of organisational nature or are valuable in an actual implementation. The creation of a color object is needed. An explicit possibility to set the color to black is necessary in some rendering algorithms. For the conversion between simple color representations explicit formulas are often known, therefore more efficient conversions can be performed. If a conversion cannot be performed explicitly, it is always possible to convert via the `ColorSpectrumNM` type.

- Creation of a color with the default value of black.
- `black:` Setting the color to black.
- Conversion from/to other color representations (RGB,CIE, HLS, Spectrum9, etc.).

With the presented color interface it is now possible to describe light and surface reflectance independent of the actual internal color representation. The generic color interface is used in the following to implement a generic interface for local illumination models describing the behaviour of illuminated materials.

Generic Material Interface

A material model describes how a surface reflects light. Other equivalent names are local illumination or local shading model. The term local is used as it is here not of interest where the light is coming from. Later on page 7 global illumination models will be discussed. The scope is limited to time-independent material models.

The material model describes the reflection of incoming light by a surface dependent on the orientation of the directions of incoming and outgoing light. The geometry for light reflection is depicted in Fig. 1, where N is the normal vector of the surface at the point under consideration. A very simple material model was
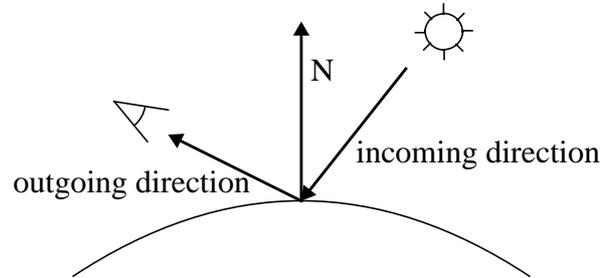

Figure 1. Geometry for Material Model.

defined by Lambert. It simulates a diffuse surface which reflects light independent of the incoming direction and depends only on the cosine of the surface normal vector and the outgoing direction. This predicts the fall-off of intensity if the surface is rotated away from the viewer.

More complex material models also consider other possibilities of light-surface interaction. An example are anisotropic materials (such as brushed aluminium) which reflect light differently when the surface is rotated around the axis defined by the normal vector. In general, the reflected light depends on the relative positions of all three vectors (normal, incoming, outgoing) and the surface orientation. The refraction (or transmission) of light at the surface of transparent media also needs to be addressed. Refraction again depends on the incoming and outgoing directions. To correctly model refracted light it is necessary to know the indices of refraction of both participating media (e.g. glass and liquid). As prisms can be used to split light into a rainbow spectrum this refraction index is actually wavelength dependent.

If a material reflects light the direction of perfect reflection is easily predicted from the geometry. Advanced material models take the roughness of the surface into account and modify the perfect reflection direction randomly in a small solid angle around the reflection vector. The same holds for the refraction vector. With transparent media it is also possible that no refraction takes place, in the case of total internal reflection.

A rendering system also needs a description of the light emitted by a light source or a luminous material in a given direction. Additionally some global illumination models use light coming from the ambient (i.e. no specific incoming direction) as an approximation to indirect illumination. Therefore a method for this approximation is needed. The last effect is the absorption of light travelling through a homogeneous medium such as air or water.

A generic interface for materials must include all above effects. The following table summarizes the functionality of the generic material interface implementing the local reflection model:

- `reflection:` Given the light coming from a certain direction and the incoming light calculate the light reflected into a given direction.
- `refraction:` Given an incoming direction and the incoming light predict the light refracted into a another direction taking the relative index of refraction of the material interface into account.
- `refractionIndex:` Return the wavelength dependent index of refraction of the material.
- `reflectionDir:` For an given incoming direction return the reflection vector.
- `refractionDir:` For an given incoming direction and a given relative refraction index return the refraction vector.
- `emission:` Return the amount of the light emitted in a given direction.
- `ambient:` For illumination coming from the ambient return how much light is reflected in a given direction.
- `absorption:` For a given amount of light travelling along a ray return how much is not absorbed after a certain distance.

Most material models only include part of this functionality. E.g. a perfect mirror reflects all light independent of the geometry and wavelength, does not refract light, reflects light only in the ideal direction, emits no light, reflects all ambient light, and does not absorb light as it is not transparent. For another example how to implement such functions consider the `reflection` method for the well known Phong model. The method computes the necessary dot products, sums the diffuse and specular term, and returns the result.

The following operations are either of organisational nature or used to implement optimizations and special effects for certain rendering systems. The creation of a material object is needed. The normal vector used in the shading calculations usually only depends on the geometry of the object. But bump-mapping [Blinn78] changes the geometric normal vector to simulate small structures in the surface such the rough skin of an orange. Therefore, a method is provided which allows the material model to change the normal vector. To facilitate the display of the modelled scenes with hardware acceleration a function is defined which returns the Lambertian diffuse color of the surface.

- Creation of a material object.
- `normal:` Normal vector for a given point on the surface.
- `opaque:` Returns if an material is opaque or transparent.
- `diffuse:` Returns the diffuse color of the material.

As presented the generic material interface is able to handle only homogeneous surfaces. Many natural surfaces exhibit a certain microstructure, called texture. In the following section the generic interface is applied to textured surface.

GENERIC TEXTURE TREES

Texture is customarily used to simulate the inherent structure of surfaces such as wood or marble. More precisely texture modifies the material for each point on the surface. For this a texture function is evaluated which depends either on the position on the surface, the position in space, the directions and magnitudes of the partial derivatives of the surface at the point, the normal vector, the color of a raster image at the surface point, the gradient of a scalar field, and so on.

Previously shade trees were proposed [Cook84] allowing the flexible definition of textures and material models. The leaves of the tree are values pertinent to the shading process. The nodes operate on these values and the root returns a color as result. Here the concept of shade trees is modified for the definition of generic texture trees. These texture trees extend the range of the generic material interface to textured surfaces. The functionality of the original shade trees is extended to allow the inclusion of recent developments in texture research.

The following problems need to be addressed:
- Textured surfaces should provide the generic material interface as presented above.

- Assignment of materials to the different parts of the texture pattern.
- The separation/abstraction of the texture function from the material properties which are modified by it.
- The transformation of texture coordinate systems to allow for slanted wood textures, etc.

In this work textures are described with the means of a generic texture tree. Each node of the tree realizes the generic material interface, each material interface method traverses the tree to determine the result. Material descriptions form the leaves of the tree. Interior nodes are either interpolation nodes, bump mapping nodes, scaling nodes or transformation nodes (as described in the remainder of this section). The simplest texture tree is just a single material node. See Fig. 1 for an example of a texture tree defining an image inside a contour overlaid onto a wood texture.
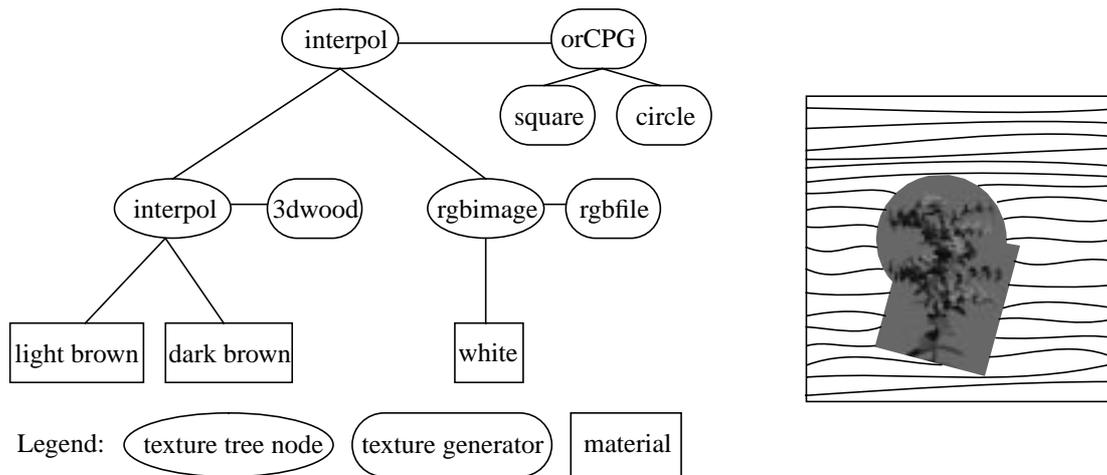
Figure 2. Sample Texture Tree and Resulting Image.

Interpolation nodes are used to decide between different materials or alternatively to interpolate between materials. This functionality covers a broad class of textures as many textures are based on a blend of materials. The interpolation node realizes a clean separation between the texture function generating the pattern and the assignment of materials to the parts of the pattern. This is done by evaluating a texture generator which returns a scalar interpolation parameter (usually in the range [0,1]). If the texture generator returns only 0 and 1 then the interpolation node realizes a decision between the two children. To simulate a blend between two different materials the texture generator returns values in the interval [0,1]. Each method uses linear interpolation between the results returned by recursive calls to the child nodes. Returned vectors (such as reflection vectors) are interpolated linearly as well. A special kind of interpolation node performs the blend for only a single method of the generic material interface, for all remaining methods the left child is evaluated. One possible use is the realization of spatially varying refraction indices as found in hand crafted glass.

A simple *texture generator* such as two dimensional checker uses the position of the intersection point on the surface to return either 0 or 1. Similarly CSG-tree and CPG-tree (constructive planar geometry = two dimensional CSG) texture generators use a tree of shapes in three or two dimensions to determine an inside/outside classification (again represented as return value of 0 or 1). Many natural surfaces such as wood can be simulated by blending materials dependent on a texture generator defined by a noise function applied to the surface position. A more specialized texture generator returns different values depending on which surface of the object was hit. Together with an interpolation node with multiple children it allows to map different materials onto the surfaces of e.g. a cube. For advanced textures such as marble multiple child materials are blended together by a spline interpolation node using again a noise texture generator.

A *bump mapping node* modifies the normal vector at a surface point. This is done by computing the *u*- and *v*-tangent vectors at the point (methods are provided by each object) and then using a texture generator based on gradients of a field to weight these vectors for normal vector calculation.

Scaling nodes weight the results of the methods with the value returned from a texture generator. The most prominent node in this class is the *image node,* which scales the color obtained from the recursive reflection function call to its child by the corresponding color in the image depending on the surface position. To avoid aliasing effects anti-aliasing methods (such as mip-mapping) are used. For the normal mapping of images a white material with the desired characteristics is used as child node. Other scaling nodes use noise functions as texture generators to simulate e.g. dirt on a surface.

Each texture generator calculates its results based on geometric properties at the surface point. *Transformation nodes* transform these values during the recursive evaluation of the sub-tree. This is useful to implement transformations on the patterns generated by the texture function. One example is a two dimensional rotation to change the direction of the stripes for a wood texture generator.

IMPLEMENTATION

The generic color and material interface and the texture trees were implemented in C++ in the framework of the Generic System [Beier-Bozetti95], a generic graphics kernel. The following definitions of the Generic kernel are used in the presented code: the letter G precedes all types (e.g. `GColor`), the letter m precedes member variables (e.g. `mDiffuse`).

Generic Color and Material Interface

See Fig. 3 for the definition of the generic color and material interface as implemented in the C++ language.

```
class GColor {                              // Generic Interface
    GColor();                               // constructor (set to black)
    GColor(const GColorSpectrumNM);         // conversion from spectrum[nm]
    operator GColorSpectrumNM();            // conversion to spectrum[nm]
    void black();                           // set explicitly to black
    GColor operator+(const GColor c) const; // addition
    GColor operator*(const GColor c) const; // for reflection
    GColor operator*(GFloat c) const;       // scaling
    GColor operator/(const GColor c) const; // division of spectra
    GColor pow(GFloat exp);                 // for absorption
    GFloat brightness();                    // convert to brightness in[0..1]
};
class GMaterial {                           // Generic Interface
    GMaterial();                            // constructor
    GMatColor reflection(GSurfPoint& sPoint,GVector3 inV,
        GMatColor illumination,GVector3 outV);
    GMatColor refraction(GSurfPoint& sPoint,GVector3 inV,
        GMatColor illumination,GMatColor refractIndex,GVector3 outV);
    GMatColor refractionIndex();
    GVector3 reflectionDir(GSurfPoint& sPoint,GVector3 inV,bool& valid);
    GVector3 refractionDir(GSurfPoint& sPoint,GVector3 inV,
        GMatColor refractIndex,bool& valid);
    GMatColor emission(GSurfPoint& sPoint,GVector3 outV);
    GMatColor ambient(GSurfPoint& sPoint,GMatColor illumination);
    GMatColor absorption(GMatColor illumination,GFloat length);
    GVector3 normal(GSurfPoint& sPoint);
    GMatColor diffuse(GSurfPoint& sPoint); // diffuse color
    bool opaque();                          // non transparent
};
```

Figure 3. Generic Interface to Colors and Materials

To make the calls to the generic material interface efficient and easy to use an object is introduced (called `GSurfPoint`) which holds all necessary geometric information for the point for which the material model (or the generic texture tree) is to be evaluated. The methods applicable to object are:

- Normal vector for shading, potentially modified by the material model.
- Point in three dimensions (local and world coordinates) for texture mapping.
- Two-dimensional coordinates on the surface (u, v) for texture mapping,
- The number of the surface of the object for surface dependent texture mapping
- u- and v-tangent vector as obtained from the geometric object for bump mapping.
- The length, the ray travelled inside the current material (i.e. the t-parameter of the ray-object intersection), used in the calculation of light absorption in transparent media.
- References to the material the ray travels from and into allows the calculation of the correct relative refraction index.
- A description of the apparent size of the pixel on the surface for anti-aliasing.

For an efficient implementation of repeated method calls (e.g. normal vector) all computed values for the current surface point are cached once they have been computed.

An important case to consider is the implementation of a material model which handles perfect specular reflection differently from diffuse and diffuse-specular reflection. Here the reflection method has to check internally if the incoming and outgoing directions align for perfect reflection (mathematically $\vec{O} = 2(\vec{I} \cdot \vec{N})\vec{N} - \vec{I}$). If they do, the formula for perfect specular reflection is to be used otherwise the normal reflection model is applied.

A topic not addressed previously in this work is the initalization. The colors and materials are read from a file where each entry is preceded by the type of the following object. Depending on this type the appropriate parser routine is called to read the parameters and the object is created by this routine.

Generic Texture Trees

Texture trees are traversed repeatedly for each generic material interface method. Therefore, efficient of the tree is important. The interpolation node caches the result of the texture generator to provide efficient handling of a sequence of calls to the texture tree. Additionally the cases integer values of the interpolation parameter are optimized to avoid unnecessary function calls.

The CSG-tree texture generator uses a tree with the three dimensional solids provided by the Generic kernel to define shapes. The current implementation of the CPG-tree texture generator uses a shape tree with the following leaf objects: square, circle, closed polygon, closed polygon with free form curves (Bézier, B-Spline, NURBS) as edges. All mentioned primitives can be transformed by affine two dimensional transformations.

One interesting use of the image node is the implementation of special effects for images. Using a non-white material as child node gives the image a colored tint. Even more effects can be created by using whole texture trees as child.

Example Renderer

The generic material interface implements the local shading model for a surface. To generate an image of a scene global illumination effects such as multiple light sources and visibility have to be considered. One well known global illumination method is ray tracing. For each visible surface point in an image it adds the contributions of all light sources visible to this point. Additionally the light coming from the direction of perfect reflection and refraction is added by recursively generating rays. In the end the returned illumination towards the viewer is subject to the absorption of the medium the viewer is in. For a more in depth discussion see [Glassner89]. In Fig. 4 an example implementation of the core of a ray tracer using the generic interfaces is shown. Note that this code is independent of the material model used and textures are handled implicitly by this code. The `shade` method of the illumination model is independent of the light source type, therefore area light sources are easy to integrate by changing only the light source shading routine.

More general illumination models are based on the simulation of particles of light and use caches of radiance values in the scene (e.g. [Jensen96],[Ward94]). Again all needed operations can be realized with the presented interface.

```
GLightColor GPointLight::shade(GSurfPoint& sPoint,GMaterial& air,GVector3 outV) {
    GLightColor tColor();                    // init with black
    GVector3 inV((mPosition - sPoint.get3dPoint()).normalize);// direction to light
    GRay3 tLightRay(sPoint.get3dPoint(), -inV);
    if (raytracer.traceShadowRay(tLightRay)) { // light source visible?
        tColor = sPoint.getMaterial().reflection(sPoint,inV,mColor,outV);
        tColor = air.absorbe(tColor,distance(sPoint.get3dPoint(),mPosition));
    }
    return tColor;
}
GIllumColor GIllumModel::shade(GRay& pRay,GSurfPoint& sPoint,GMaterial& air){
    GMaterial& material(sPoint.getMaterial()); // get material of hit point
    GVector3 tView(pRay.getVector());    // get viewing vector
    bool valid = FALSE;
    // compute contribution of direct illumination by all light sources
    GIllumColor tColor = material.emission(sPoint,-tView); // Init with emission
    for (GLightItr _Itr(raytracer.mListLights); _Itr.more(); _Itr.next()) {
        tColor += _Itr.cur().shade(sPoint,air,-tView);
    }
    // compute specular reflection
    GVector3 tReflectDir = material.reflectionDir(sPoint,tView,valid);
    if (valid) {
        GRay3 tReflRay(sPoint.get3dPoint(), tReflectDir);
        GIllumColor tIllum = raytracer.traceShadingRay(tReflRay,air);
        tColor += material.reflection(sPoint,-tReflectDir,tIllum,-tView);
    }
    // compute specular refraction
    GIllumColor ri(air.refractionIndex() / material.refractionIndex());
    GVector3 tTransDir = material.refractionDir(sPoint,tView,ri,valid);
    if (valid) {                              // is also check for TIR!
        GRay3 tTransRay(sPoint.get3dPoint(), tTransDir);
        GIllumColor tIllum = raytracer.traceShadingRay(tTransRay,material);
        tColor += material.refraction(sPoint,-tTransDir,tIllum,ri,-tView);
    }
    // absorption until light reaches viewer
    return air.absorbe(tColor,sPoint.getTParameter());
}
```

Figure 4. Example Ray Tracer Implementation

CONCLUSION AND FUTURE WORK

A new generic interface to color and material models was presented. Portable and reusable color and material models can be implemented by providing functions conforming to this interface. Generic texture trees were proposed which allow the application of the generic material interface to textured surfaces. With these trees the reuse of code for textured surfaces is possible. Also this work can be considered a first step towards portable texture definitions.

Future work includes the extension of the generic material interface to include effects like polarization, fluorescence, and phosphorescence. Also the extension to inhomogeneous transparent media (e.g. smoke, clouds) is planned.

References

[Beier-Bozetti95] Beier, E. and U. Bozetti, *"A Generic Graphics Kernel and a Customized Derivative"*, available from ftp://metallica.prakinf.tu-ilmenau/pub/PROJECTS/ GENERIC/dublin.ps.gz, 1995.

[Blinn78] Blinn, J. F., *"Simulation of Wrinkled Surfaces"*, Computer Graphics (SIGGRAPH `78 Proceedings), vol. 12, no. 3, pp 286-292, 1978.

[Cook84] Cook, R. L, *"Shade Trees"*, Computer Graphics (SIGGRAPH `84 Proceedings), vol. 18, no. 3, pp 223-231, 1984.

[Freudenthaler96] Freudenthaler, G., *"Erweiterung eines Generischen Graphischen Kernels"*, Diplomarbeit, Inst. für Informatik, Johannes Kepler University Linz, Austria, 1996.

[Glassner89] Glassner, A. S., *"An Introduction to Ray Tracing"*, Academic Press, 1989.

[Glassner95] Glassner, A. S., *"Principles of Digital Image Synthesis"*, Morgan Kaufman Publishers, 1995.

[Hall89] Hall, R., *"Illumination and Color in Computer Generated Imagery"*, Springer Verlag, 1989.

[He *et al.*91] He, X. D., K. E. Torrance, F. X. Sillion, and D. P. Greenberg, *"A Comprehensive Physical Model for Light Reflection"*, Computer Graphics (SIGGRAPH `91 Proceedings), vol. 25, no. 4, pp 175-186, 1991.

[Jensen96] Jensen, H. W., *"Global Illumination using Photon Maps"*, 7th EUROGRAPHICS Workshop on Rendering (Porto), pp 22-32, 1996.

[Schlick94] Schlick, C., *"An Inexpensive BRDF Model for Physically-based Rendering"*, Computer graphics forum (EUROGRAPHICS `94 Proceedings), vol. 13, no. 3, pp 233-246, 1994.

[Slussallek-Seidel96] Slussallek, P., H.-P. Seidel, *"Towards an Open Rendering Kernel for Image Synthesis"*, Proceeding of EG Workshop on Rendering 1996, pp 52-61, 1996.

[Strauss92] Strauss, P. S., *"A Realistic Lighting Model for Computer Animators"*, IEEE Computer Graphics and Applications, vol. 10, no. 11, pp 56-64, 1992.

[Stürzlinger *et al.*93] Stürzlinger, W., R. Tobler, and M. Schindler, *"FLIRT - Faster than LIght Ray Tracer"*, Technical Report, Institut für Computergraphik, Technical University of Vienna, Aug. 1993.

[Upstill89] Upstill, S., *"The RenderMan Companion"*, Addison-Wesley, 1989.

[Ward92] Ward, G. J., *"Measuring and Modeling Anisotropic Reflection"*, Computer Graphics (SIGGRAPH `92 Proceedings), vol. 26, no. 2, pp 265-272, 1992.

[Ward94] Ward, G. J., *"The RADIANCE lighting simulation and rendering system"*, Computer Graphics (SIGGRAPH `94 Proceedings), vol. 28, no. 2, pp 459-472, 1994.

[Watt-Watt92] Watt, A., and M. Watt, *"Advanced Animation and Rendering Techniques: Theory and Practice"*, Addison-Wesley, 1992.