Making global illumination user-friendly

Gregory J. Ward

Lawrence Berkeley Laboratory
1 Cyclotron Rd., 90-3111
Berkeley, CA 94720 USA

ABSTRACT

Global illumination researchers tend to think in terms of mesh density and sampling frequency, and their software reflects this in its user interface. Advanced rendering systems are rife with long command lines and parameters for tuning the sample densities, thresholds and other algorithm-specific variables, and the novice user is quickly lost in a sea of possibilities. This paper details a successful effort of making one such global illumination system usable by people who understand their problems, even if they do not understand the methods needed to solve them, through an assisted oracle approach. A single program is introduced to map a small set of intuitive control variables to the rendering commands and parameter settings needed to produce the desired output in a reasonable time. This new executive program then serves as the basis for a graphical user interface that is both friendly in its appearance and reliable in its performance. Finally, we conclude with some future directions for improving this interface.

1. Introduction

As rendering and especially global illumination techniques advance, the number of user-settable rendering parameters tends to increase. This is because most algorithms have associated sampling rates that are not determined by any basic property of the rendering equation, but are rather a function of the modeled environment and user requirements for output quality. Therefore, the programmer provides the user with parameters to control the calculations so that the best trade-off between time and accuracy can be achieved for a given application.

Such flexibility may be perceived as unwanted complexity by the novice user, and setting the parameters correctly to obtain the best result often requires an intimate understanding of the underlying algorithms. It is little wonder that program authors and their close associates have the most success with advanced rendering software, since they are the only ones who can make it behave properly.

The real difficulty in global illumination is mapping a given set of algorithms to a given problem in an efficient way. Current rendering software is a lot like a box of tools, and if it is not accompanied by the requisite expertise, nothing good can be built from it. What we need to do as system designers is empower the user by supplying the needed expertise along with the tools so that they can build their own house, or museum, or space station, or whatever. Other researchers have suggested this in previous papers and the notion of an *oracle* has been introduced, which is a computational agent that decides what to sample and where in a global illumination calculation [Drettakis91].

Creating robust oracles is a very difficult problem, however, and requires that some limited subset of "common sense" be programmed into these agents. For example, the user may know the geometric detail contained in a model, but a simple count of polygons is usually a poor measure, since it can be thrown off by a relatively smooth surface that is finely tesselated or a small number of polygons all intersecting each other. If the geometric detail is important to the setting of calculation parameters, an oracle will have trouble deciding what to do if it does not have common sense enough to determine the real detail level.

An alternate approach is to analyze what is being computed, and find ways to adjust the calculation automatically to some goal. In the case of synthetic images, we must find an accuracy metric that tells us how far we have to go in our progressive calculation. This can perhaps be done for a Monte Carlo path-tracing simulation, but we still do not have a good handle on what does and does not matter to the user in the resulting image. The metrics we can compute, such as RMS error, have been shown to have little correspondence to quality or correctness as perceived by a human observer. Other metrics, such as those described in [Rushmeier95], are still in the experimental stages. Even if we find and accept a good metric for image quality, we are still left with the problem of mapping our algorithms to progressive approaches that work well with this metric. For example, we may find that geometric shape is very important to human observers. If we are using progressive refinement radiosity, the geometry displayed remains constant, while the illumination changes. No amount of iteration will improve our polygonal model, so we end up refining along the wrong axis for a metric that is sensitive to geometry.

In this paper, we demonstrate an assisted oracle approach, which leans on the user to find out certain common-sense things about the model, then employs a simple set of rules to arrive at the appropriate calculation parameters from this information. The user is allowed general control over things such as "image quality", which loosely translates to visual accuracy, and output resolution, but is freed from having to understand the details of algorithms employed in the calculation. And by relying on the user's common sense, the oracle is freed from having to understand basic things about the real world.

2. Basic Concepts

Although we restrict ourselves in this paper to the context of a specific rendering system, we introduce the following concepts that may be applied readily to other global illumination calculations.

Executive Control Program

Although toolbox systems provide the greatest flexibility, they are difficult to learn and can be difficult to run even for the experienced user. By introducing an executive program, we simplify the most frequently used operations by combining them into a single rendering command. Doing the job well, however, requires setting the many parameters of the constituent programs very carefully. Freeing the user of this burden is at least as important as simplifying the running of the software.

Intuitive Control Variables

In order to free the user from having to think in terms of global illumination algorithms and procedures, we must distill a set of intuitive control variables that are both comprehensive and comprehensible. This actually turns out to be much easier than it sounds, though we can offer no foolproof method for mapping common-sense parameters to algorithmic ones. Such mapping requires an expert in the particular rendering software who can code this knowledge into a working program.

Simulation Zones

The simulation zone is a concept we introduce to isolate a specific set of rendering parameters. A zone is a set of simulation conditions that includes a complete scene description, a single lighting condition, and a region of interest (usually a room or an object). Within a zone there may be multiple views specified, but the illumination and other gross scene conditions remain constant. (Small objects may be animated, so long as it does not have a large effect on the lighting.) By focusing on one zone at a time, it is possible to derive a set of reliable rendering parameters.

Graphical User Interface for Rendering

The concept of a GUI for rendering is nothing new, but the design of such an interface usually requires much attention and many trials. We show in this paper how a GUI can be placed on top of an executive control program with very little thought or effort, yielding excellent results. This is because there is a very natural correspondence between the intuitive control variables and the controls of a GUI.

The specific example we explore in this paper is a program called rad, which is a recent addition to the *Radiance* lighting simulation and rendering system [Ward94]. Rad takes a small number of intuitive values and combines this information with some gleanings from the compiled scene description to assign all of the various parameters that control the simulation. We start by enumerating some of these algorithmic parameters and illustrating how difficult they are for the common user to set, then show how we extracted a more intuitive set of control variables. Next, we show how easily these variables can be attached to a GUI. Finally, we discuss the need for a diagnostic tool to troubleshoot problem images, which we leave for future exploration.

3. RADIANCE Calculation Parameters

Table 1 shows an abbreviated list of the program parameters that control the rendering process in *Radiance,* and their default values. The primary author of this software can look at these parameters and understand what they mean and why they are there, and by experience how to set them for various rendering situations. However, the average user looks at these with a very puzzled expression and says, "Well, I guess I could try changing this one or that one to see what happens," and off they go.

Table 1. Rendering algorithm parameters and default values.

Rendering Parameter	Interpretation	Default Value
-pj	pixel jitter	0.67
-ps	pixel sample	4
-pt	pixel threshold	0.05
-dt	direct threshold	0.05
-dc	direct certainty	0.5
-dj	direct jitter	0.0
-ds	direct sampling	0.25
-dr	direct relays	1
-dp	direct pretest density	512
-sj	specular jitter	1.0
-st	specular threshold	0.15
-av	ambient value	0.0 0.0 0.0
-ab	ambient bounces	0
-aa	ambient accuracy	0.2
-ar	ambient resolution	32
-ad	ambient divisions	128
-as	ambient super-samples	0
-lr	limit reflection	6
-lw	limit weight	0.005

Through no fault of their own, the users take what from the programmer's perspective seems like a carefully crafted set of tools, and they start banging on things with them. How could we expect anything different? No one has taught them what these things mean. The manual pages are long and difficult to follow, and the research papers explaining what is behind the command line are even worse. People have been given the tools, but no real guidance in their application. Even the author had to experiment at one time to learn what settings were reasonable and which ones produced good results under specific circumstances; this knowledge is very difficult to impart.

Another stumbling block for new *Radiance* users is that the software is broken into many independent modules, following the UNIX toolbox paradigm. There is no single user interface even on the command line, and users must learn about many different programs before seeing their first image. There are several CAD translators and object generator programs to assist in scene creation. There is a program to compile the scene description, and one to render the scene into an unfiltered image, and one to reduce the image and perform anti-aliasing, and conversion and display programs to put the image in a common format or display it on the monitor. Unless the user's mind works like the programmer's, mastering the intricacies of *Radiance* seems more tedious and painful than intuitive.

Our first goal in developing a user interface to this software is to replace the tedium and confusion with simplicity and clarity. We do this by unifying the operation of the software and replacing the nasty algorithm-derived parameters with more intuitive variables based on common sense. A single command taking a short input file (less than 1K, typically) can control the core rendering modules of *Radiance* and put them into a consistent, logical interface. Next, we can put a GUI on top of this new executive program so that the user does not even have to think about what to type, and we can hook in other functionality that is difficult to handle in a single command line program.

4. RAD Control Variables

The key to designing a good user-assisted oracle is finding a minimal set of intuitive control variables that tells us exactly what we need to know from the user in order to run the software efficiently. The purpose is to minimize the number of decisions the user has to make for the majority of cases. The number of control variables must be small, and the possible values for these variables must also be small or arbitrary. (I.e. if a value is irrelevant to the calculation, such as the output file name, the user is given freedom to make arbitrary assignments. If a value affects the calculation, however, the user may be restricted to a choice of "high," "medium" or "low" rather than some numeric value.) We may keep the idea of default values around, but we want the user to feel comfortable changing these settings at will. To arrive at appropriate variables for a good user-assisted oracle, we must ask ourselves two questions:

1. What does the user wish to control?
2. What additional information do we require from the user to perform the simulation?

In answer to the first question, most users want to control the time versus quality tradeoff. They also want to control the views from which the renderings take place, and probably the size and resolution of the output. After that, the typical user is just not very interested in the rendering process.

Unfortunately, the oracle in rad is not smart enough to figure out the rest on its own, so we still need the user to give us some additional clues to enable us to do a good job rendering the scene in a reasonable time. Obviously, we need to know input file name(s). Rad also asks for intermediate and output file names, though these will be assigned default values if none are given. Since we are going to maintain a compiled version of the scene, we also need to know what other files the scene depends on, similar to the information required by the UNIX make facility.

Once the file information is settled, rad needs to know a little bit about the scene geometry and lighting for the specific area being rendered, which we call a *zone*. It was decided early on that a given set of variable settings should apply to just one zone, since the geometry and lighting could vary too much from inside to outside or even one room to another in a large model. Dividing the rendering task into zones greatly simplifies the job of setting rendering parameters in *Radiance*, which depend on geometric complexity and lighting variability.

A zone is specified by type, interior or exterior, and dimensions. Rad uses a 3-dimensional, axis-aligned box in its zone specification, though the precise boundaries are of little importance. The primary information derived from the zone is a relative "scale" for rendering parameters. The scale simply tells us the distance at which lighting ceases much to matter. This information is extracted from the overall size of the zone plus the geometric detail as specified in a separate variable. Secondarily, a zone provides a convenient mechanism for establishing default viewpoints to get the user started looking at their environment.

Table 2. Rad control variables and default values.

Variable	Interpretation	Type	Default Value
materials	materials file(s)	string	-
scene	scene file(s)	string	-
illum	illum object file(s)	string	-
objects	requisite file(s)	string	-
view	image view(s)	string	X (from maximum X)
UP	view up vector	string	- (effectively +Z)
QUALITY	target image quality	qualitative	Low
RESOLUTION	output image resolution	integer	512
PICTURE	output file root	string	input file root
OCTREE	octree (compiled scene) file	string	input file root + .oct
AMBFILE	ambient value file	string	-
OPTFILE	options file	string	-
REPORT	report interval and file	string	-
ZONE	region of interest	string	bounding cube, ext.
EXPOSURE	image exposure	real	- (automatic)
PENUMBRAS	penumbras important?	boolean	False
DETAIL	geometric detail	qualitative	Medium
INDIRECT	# important interreflections	integer	0
VARIABILITY	variation in illumination	qualitative	Low
oconv	oconv options	string	-
mkillum	mkillum options	string	-
render	rendering options	string	-
pfilt	pfilt options	string	-

Four additional variables are used to assess the difficulty of the lighting calculation for the given zone. These are the lighting variability, the number of indirect reflections, and the image exposure. Variability is a qualitative setting that indicates how much light levels vary in the zone, i.e. the dynamic range of light landing on surfaces. This is important for controlling the number of rays used in sampling indirect lighting. A second variable tells rad how many diffuse reflections are critical to the lighting of a space. If a direct lighting system is used, this is set to 0. For an indirect fluorescent system, it is set to 1. For some daylight shelf systems, the setting may be 2. A third variable tells rad what multiplier to use for exposing the final images. This can be done automatically, but the results are usually not as good, and rad uses this value also to set the "ambient level" for the zone, which is a function of the final light levels and therefore cannot be determined in advance except for very simple, closed environments. Finally, a fourth variable indicates whether or not rendering penumbras is important in this zone. Turning penumbras off does not mean that area sources are treated as points; it only means that the quality of soft shadows is less important than rendering time in this model.

You might at this point be wondering why *rad* could not somehow figure out all this additional stuff and not bother the user about it? In principle, it could. This is the idea behind an "oracle" as recalled in the introduction. Unfortunately, writing omniscient oracles requires the kind of common sense reasoning that is easy for people familiar with a given environment, but difficult for computers. By the nature of the situation being modeled, the user *knows* whether to expect a high degree of variability in the lighting or a low one. Figuring this out automatically requires actually rendering the space at some level of detail, and thus solving the problem in order to begin. It may be possible to do this iteratively, but it is more complicated, takes longer and the result is no more reliable than asking the user. The question we have to ask when designing an interface is, "What can be done for the user in a way that is more pleasant than it is aggravating?" An interface that takes twenty minutes to come up with a default parameter setting is just plain annoying. As it is, rad already derives some information from the scene files, which may take several seconds to process if the scene has not already been compiled.

Table 2 shows a list of the rad variables, interpretation, type and default values. Lower case variables may have multiple settings, which are usually just concatenated together. Upper case variables may have only one valid setting.

Variables above the horizontal division are considered user control settings. In addition to the control of input files, views, image quality and output resolution, the user may set intermediate and output file names.

Variables below the horizontal division are considered program help settings, i.e. things that rad asks the user in order to determine what rendering parameters will work for this environment. These are the zone, detail, variability, indirect, exposure and penumbra settings we spoke of earlier. As a back door for expert users, variables are also provided for adding or overriding options to specific *Radiance* modules.

5. What Does RAD Do?

Once rad gets all this information, what does it do, exactly? Well, it depends on how it is invoked, but the usual action is as follows:

1. Compute default values for unspecified variables.
2. Derive rendering and filtering module parameters.
3. Recompile scene if necessary.
4. Render and filter each view.

Steps 1 and 2 are always carried out. Step 3 usually follows, unless "no action" is specified. Step 4 includes such subtleties as recovering aborted renderings, and may be replaced by an interactive or batch rendering of a single view if desired. Other options control what is printed on the standard output, if anything. (The default is to print each command as it is executed.) There is also a "touch" option for bringing file times up to date without actually doing anything, which is sometimes handy for avoiding overreactions to small changes.

To give an example of using rad on a real rendering problem, Listing 1a shows a typical rad input file, and Listing 1b shows the default values for unassigned variables as computed in Step 1. Listing 2 shows the commands executed by rad with their parameters.

How was the transformation of Listing 1 into Listing 2 accomplished by rad? Without going into detail, there are three separate procedures for Step 2, corresponding to the three possible settings of the *QUALITY* variable, Low, Medium and High. The Low procedure makes every possible compromise in

```
mat= iesroom.mat
scene= iesroom.rad extras.rad
scene= ceilingA.rad taskC.rad windows.rad
obj= terminal.rad typeA.rad typeA_cross.rad
ZONE= I  0 15  0 20  0 10
AMB= ver1.amb
VAR= Low
EXP= 1
QUA= Med
PEN= True
RES= 640 480
view= west -vf west.vp
view= efish -vf efish.vp
```

Listing 1a. Example rad input file, "ver1.rif".

```
OCTREE= ver1.oct
PICTURE= ver1
INDIRECT= 0
DETAIL= Medium
```

Listing 1b. Computed default values for unassigned rad variables.

```
oconv iesroom.mat iesroom.rad extras.rad ceilingA.rad taskC.rad
        windows.rad > ver1.oct
rpict -vf west.vp -x 1280 -y 960 -ps 3 -pt .08 -dp 512 -ar 22 -ds .2 -dj .5
        -dt .1 -dc .5 -dr 1 -sj .7 -st .1 -af ver1.amb -aa .25 -ad 196
        -as 0 -av 0.5 0.5 0.5 -lr 6 -lw .002 ver1.oct > ver1_west.raw
pfilt -1 -e 1 -r 1 -x /2 -y /2 ver1_west.raw > ver1_west.pic
rm -f ver1_west.raw
rpict -vf efish.vp -x 1280 -y 960 -ps 3 -pt .08 -dp 512 -ar 22 -ds .2 -dj .5
        -dt .1 -dc .5 -dr 1 -sj .7 -st .1 -af ver1.amb -aa .25 -ad 196
        -as 0 -av 0.5 0.5 0.5 -lr 6 -lw .002 ver1.oct > ver1_efish.raw
pfilt -1 -e 1 -r 1 -x /2 -y /2 ver1_efish.raw > ver1_efish.pic
rm -f ver1_efish.raw
```

Listing 2. *Radiance* commands executed by rad.

quality to achieve the fastest rendering times. This setting is best used for looking at geometry and picking views. The Medium procedure tries to do a reasonable job without taking too long. This setting is OK for work in progress where accuracy is not critical. The High procedure turns up parameters as necessary to achieve good quality results, even if the calculation will be slow. This setting is appropriate for presentation or publication images.

There is very little point in describing these routines in any detail, but pseudocode for the medium quality procedure is included in the Appendix for those who are curious. What is significant is that these three short routines embody much of the author’s knowledge about *Radiance* rendering and what works best in a given situation. As awkward as they may look, they are a form of expert knowledge, and it would have been impossible to write them without years of experience using this software. Initially, it seemed that coding this knowledge would be impossible, but it turned out instead to be cathartic.

```
rpict -vf west.vp -x 1280 -y 960 -ps 2 -pt .08 -dp 1024 -ar 45 -ds .2 -dj .5
      -dt .1 -dc .5 -dr 1 -sj .7 -st .1 -af ver1.amb -aa .25 -ad 196
      -as 0 -av 0.5 0.5 0.5 -lr 6 -lw .002 ver1.oct > ver1_west.raw
```

Listing 3. Rendering command after increasing *DETAIL* to "High".

It is fun to change one of the rad variables to see how it affects the rendering parameters. Since entirely different procedures are used for the three quality settings, changing this variable obviously has the biggest effect. Let us look instead at what happens when we change a minor variable such as the *DETAIL* setting. We will take it from the default setting of "Medium" to "High". Listing 3 shows the first rpict command with its new options. Note how only a few of the parameters change: the pixel sampling density (-ps), the direct presampling density (-dp), and the ambient resolution (-ar). It would have taken a *Radiance* expert to figure out which options to change and which to leave alone, but with rad, we only had to know that our scene is now more detailed.

Through this kind of experimentation, it is even possible for the user to gain some knowledge about the *Radiance* parameters without having to waste hours on bad renderings.

6. Putting a Graphical User Interface on RAD

Now that we have a user-friendly command line interface, we would like to take it one step further and provide a GUI for *Radiance* rendering. This turns out to be both easy and hard. It is easy in the sense that there are nice tools for building GUI's such as Tcl/Tk [Ousterhout94], which do most of the work for you. It is hard in the sense that takes about 50 Kbytes of interface code and another 50 Kbytes of help screens for an interface that manipulates about 1 Kbyte of rad control data. Fortunately, developing a GUI also allows us to add some functionality that we could not include in a single command line, such as image display and conversion. In the future, we may hook other tools to the interface as well; thus it may serve as a central point for running the entire software suite.

The current rad interface, called trad, is broken into seven interactive screens, which group functions into convenient categories. Table 3 lists the screen names, their functions, and which rad variables they may modify. Figure 1 shows a typical trad screen. The mode buttons are arranged in a constant area of the interface along the right hand side, together with HELP and QUIT buttons. A second constant area along the bottom is used for messages. The rest of the interface will change depending on which mode (screen) is selected. In the screen shown, the user has the option of changing the *ZONE* type and limits, the *DETAIL*, *INDIRECT*, and *VARIABILITY* settings, and the *EXPOSURE* value. All of these variables give details needed by rad to efficiently render a particular zone, thus they are logically grouped together. The "Copy" and "Revert" buttons in the lower right of the Zone screen may be used to selectively load the variables on this screen from another rad input file, or to return to the original settings from this file, respectively. These buttons are quite useful, and they appear on all of the trad screens that affect rad variables.

Table 3. Trad screens and functions.

Screen	Function	Modifies
File	Load/save Rad Input Files	All
Scene	Specify input files	OCTREE, materials, illum, scene, objects
Zone	Edit zone-related variables	ZONE, DETAIL, INDIRECT, VARIABILITY, EXPOSURE
Views	Edit views	view, UP, PICTURE, RESOLUTION
Options	Edit rendering options	QUALITY, PENUMBRAS, AMBFILE, OPTFILE, REPORT, oconv, mkillum, render, pfilt
Action	Start interactive or batch rendering	None
Results	Display/convert/print images	None

Figure 1. Trad Zone screen, one of seven such screens determined by the mode button selected on the right.

Context-sensitive help is provided through a help facility by control-clicking on any of the trad buttons or windows. It is assumed that the user has a working knowledge of *Radiance* and especially rad, but the GUI itself can be learned very quickly by calling on help whenever something is not understood.

7. Conclusion and Future Work

We have presented a user-friendly approach to rendering with advanced global illumination algorithms, and demonstrated the concept of a user-assisted oracle for setting calculation parameters. The system described works well and is accepted by even the most skeptical users once they give it a try. Even the author of the *Radiance* package prefers the new control program to the old manual method of rendering via pipes and monster command lines.

Future work shall continue in two areas. First, the GUI shall be linked to additional tools, such as CAD programs and translators on the input side and analysis tools on the output side. Second, a picture diagnostic tool shall be created to provide additional expertise in correcting problem renderings.

Even with the user-assisted oracle in rad, there are occasions when the rendering output is less than satisfactory, and the average user may have difficulty correcting such problems without deeper understanding of what can go wrong. A diagnostic tool would help the user to identify the nature of the problem with comparisons to other pictures with the same artifacts. The tool would then suggest or implement changes to the rad input file to correct these problems. This returns us to the iterative, trial and error approach we sought to avoid with our interface in the first place, but it should only be needed in exceptional cases, and rerendering with some intelligent changes is better than giving up or living with bad output.

Designing a good user interface to advanced rendering algorithms is not as simple as deciding what color buttons look best. It really requires an expert to sit down and codify the knowledge that permits him or her to create beautiful output with a given set of tools, so that less experienced users might do the same. We have shown that there is at least one path towards this goal. We believe there are many others, and encourage our fellow researchers to find them.

References

[Drettakis91]
Drettakis, George, Eugene Fiume, ‘‘Structure-Directed Sampling, Reconstruction, and Data Representation for Global Illumination,’’ Proceedings of the Second Eurographics Workshop on Rendering, Barcelona, 13-15 May 1991.

[Ousterhout94]
Ousterhout, John, *Tcl and the Tk Toolkit*, Addison-Wesley Professional Computing Series, 1994.

[Rushmeier95]
Rushmeier, Holly, G. Ward, C. Piatko, P. Sanders, B. Rust, ‘‘Comparing Real and Synthetic Images: Some Ideas About Metrics,’’ submitted to the Sixth Eurographics Workshop on Rendering, Dublin, Ireland, June 1995.

[Ward94]
Ward, Gregory, ‘‘The RADIANCE Lighting Simulation and Rendering System,’’ *Computer Graphics*, July 1994.

Appendix

```
procedure SET_MEDIUM_QUALITY_OPTIONS begin
                                        /* set pixel sampling, direct presampling, and ambient resolution */
D = size of scene bounding cube / average dimension of ZONE
switch (DETAIL)
case LOW:
    if (PENUMBRAS) then
        option("-ps 4")
    else
        option("-ps 8")
    endif
    option("-dp 256 -ar %d", 8*D)
    break
case MEDIUM:
    if (PENUMBRAS) then
        option("-ps 3")
    else
        option("-ps 6")
    endif
    option("-dp 512 -ar %d", 16*D)
    break
case HIGH:
    if (PENUMBRAS) then
        option("-ps 2")
    else
        option("-ps 4")
    endif
    option("-dp 1024 -ar %d", 32*D)
    break
endswitch
option("-pt .08")                       /* pixel threshold for medium quality */
if (PENUMBRAS) then                          /* set direct subsampling and jitter */
    option("-ds .2 -dj .5")
else
    option("-ds .3")
endif
option("-dt .1 -dc .5 -dr 1 -sj .7 -st .1")  /*set direct and specular sampling for medium quality */
if (INDIRECT > 0) then                            /* set indirect bounces */
    option("-ab %d", INDIRECT)
endif
if (defined(AMBFILE)) then                   /* set ambient file */
    option("-af %s", AMBFILE)
endif
switch (VARIABILITY)                              /* set indirect sampling */
case LOW:
    option("-aa .25 -ad 196 -as 0")
    break
case MEDIUM:
    option("-aa .2 -ad 400 -as 64")
    break
```

```
case HIGH:
        option("-aa .15 -ad 768 -as 196")
        break
endswitch
A = 0.5/EXPOSURE                        /* set ambient value */
option("-av %f %f %f", A, A, A);
option("-lr 6 -lw .002")
if (defined(RENDER)) then               /* add user-specified rendering options */
        option(RENDER)
endif

end SET_MEDIUM_QUALITY_OPTIONS
```