nvision 08
THE WORLD OF VISUAL COMPUTING

Learning to write mental ray shaders

Andy Kopra
Senior Technical Writer
mental images, Inc.

NVIDIA

Learning to write mental ray shaders

1. Shaders and the structure of mental ray
2. Strategy of the new shader book
3. Cross-referencing in the shader book
4. Website support for the book’s software
5. The CDROM included with the book

Shaders and the structure of mental ray

1

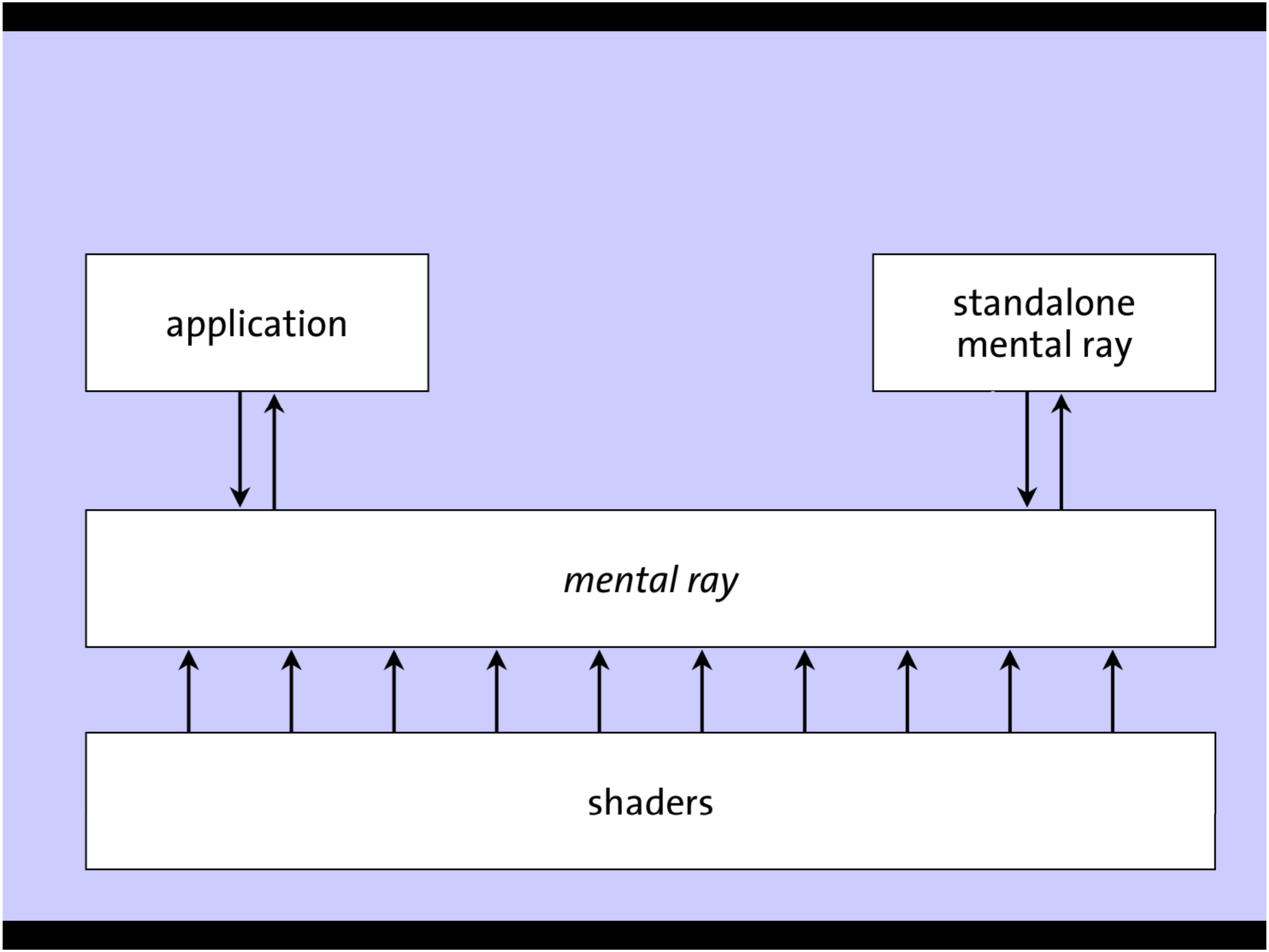
Shaders and the structure of mental ray

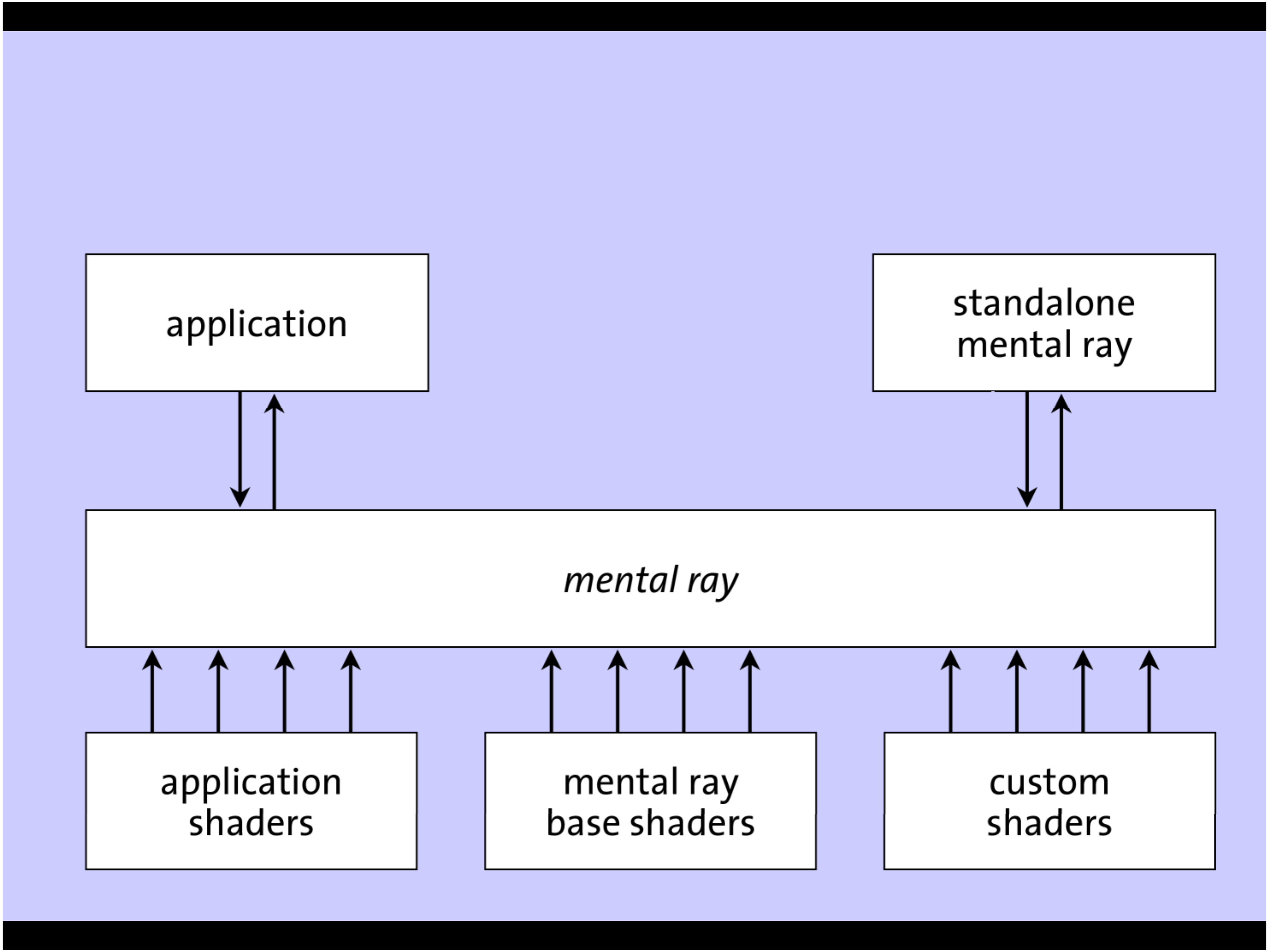
The mental ray renderer is a software library that is embedded in different interface applications.

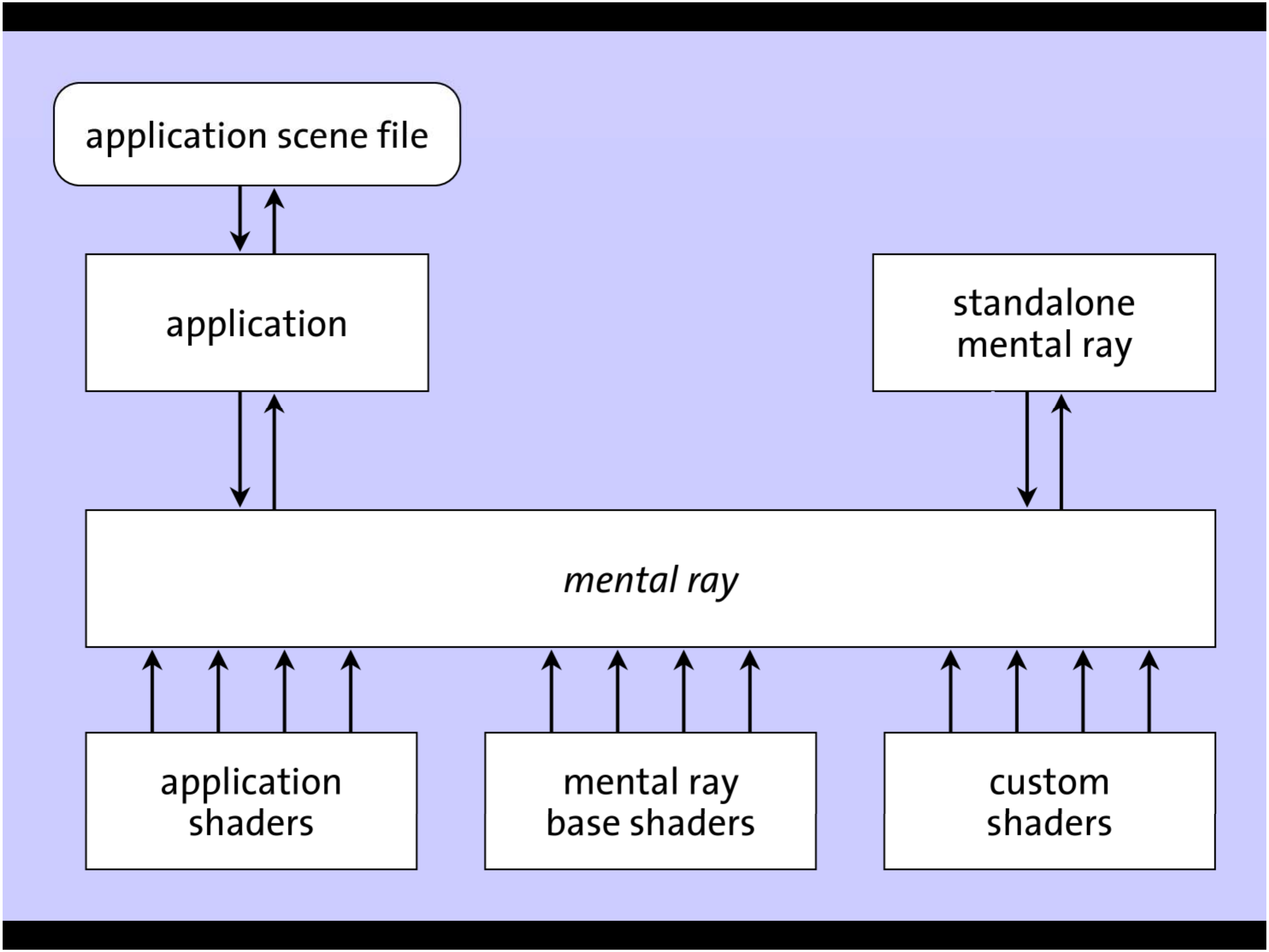
mental ray

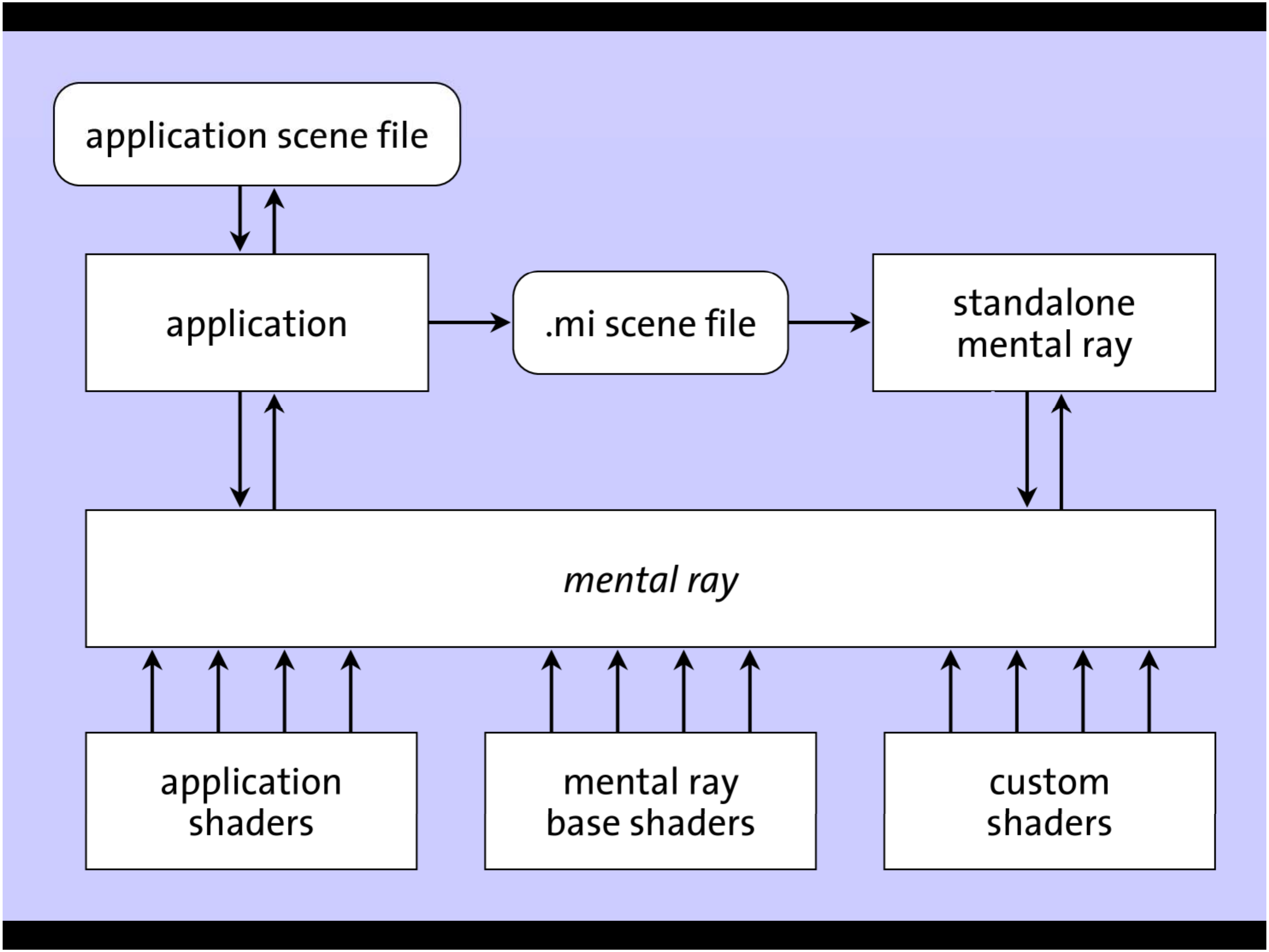

application
mental ray


application
standalone
mental ray
mental ray

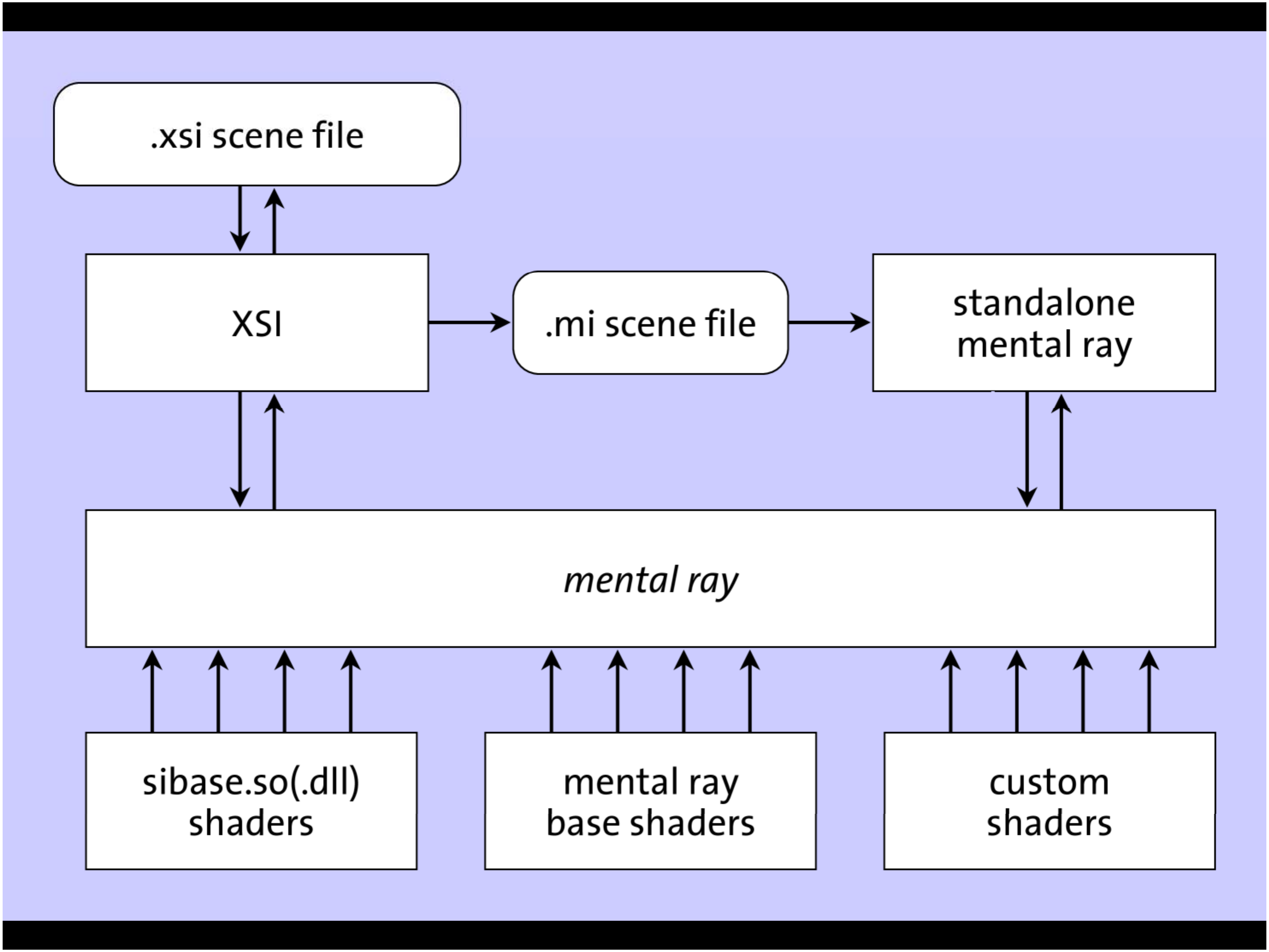

application
standalone
mental ray
mental ray
shaders

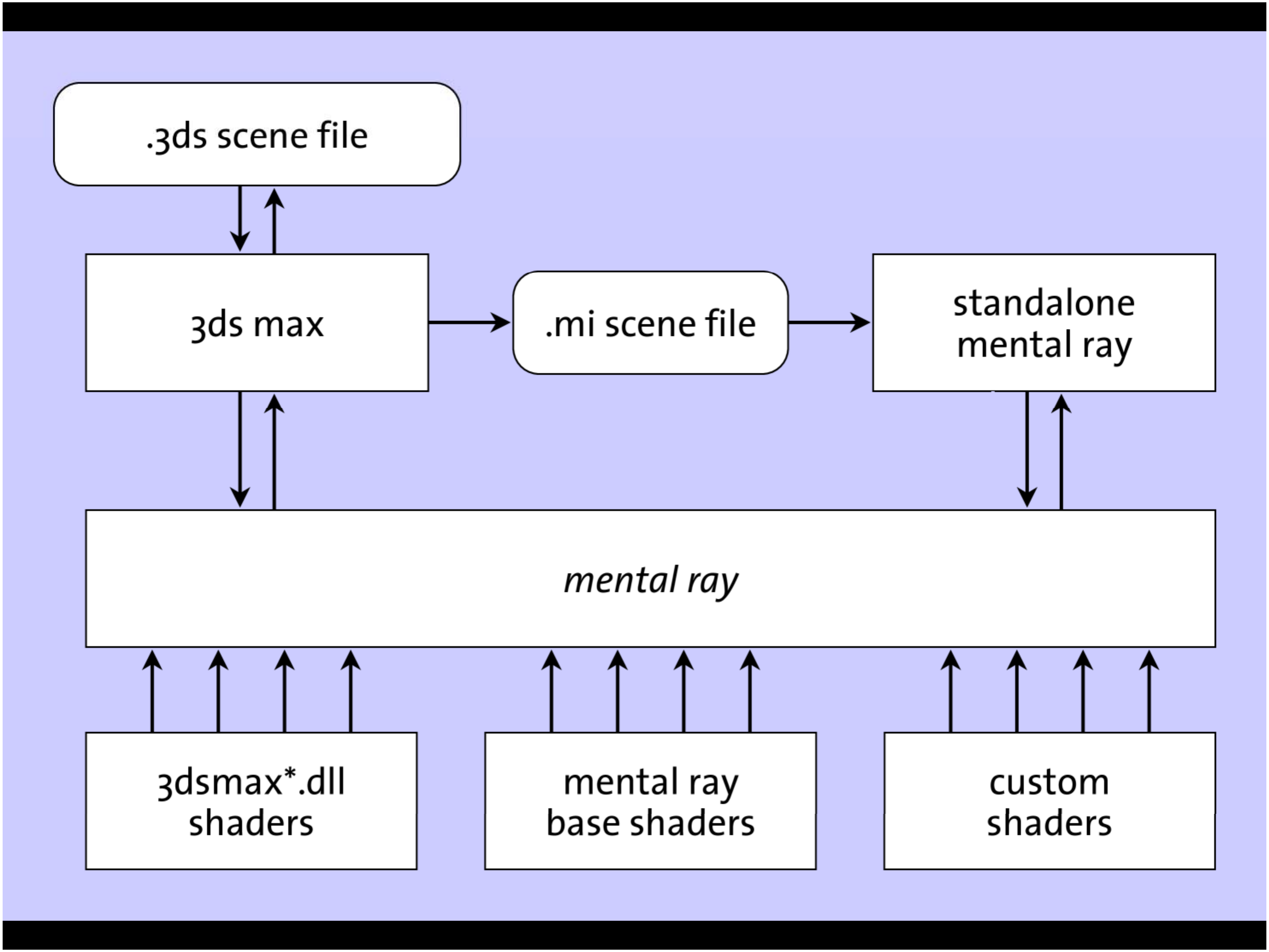

application
standalone mental ray
mental ray
application shaders
mental ray base shaders
custom shaders

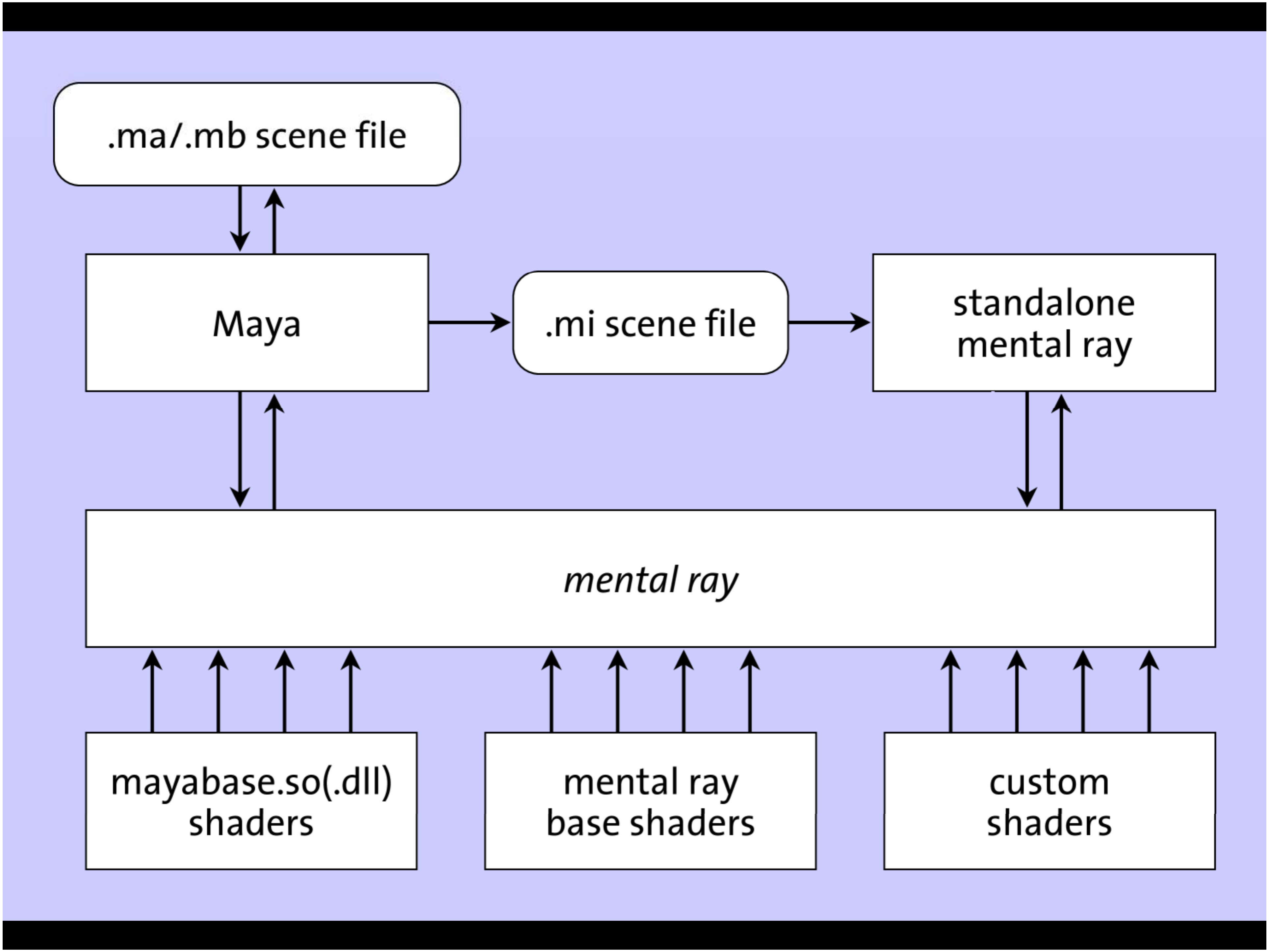

application scene file
application
standalone mental ray
mental ray
application shaders
mental ray base shaders
custom shaders

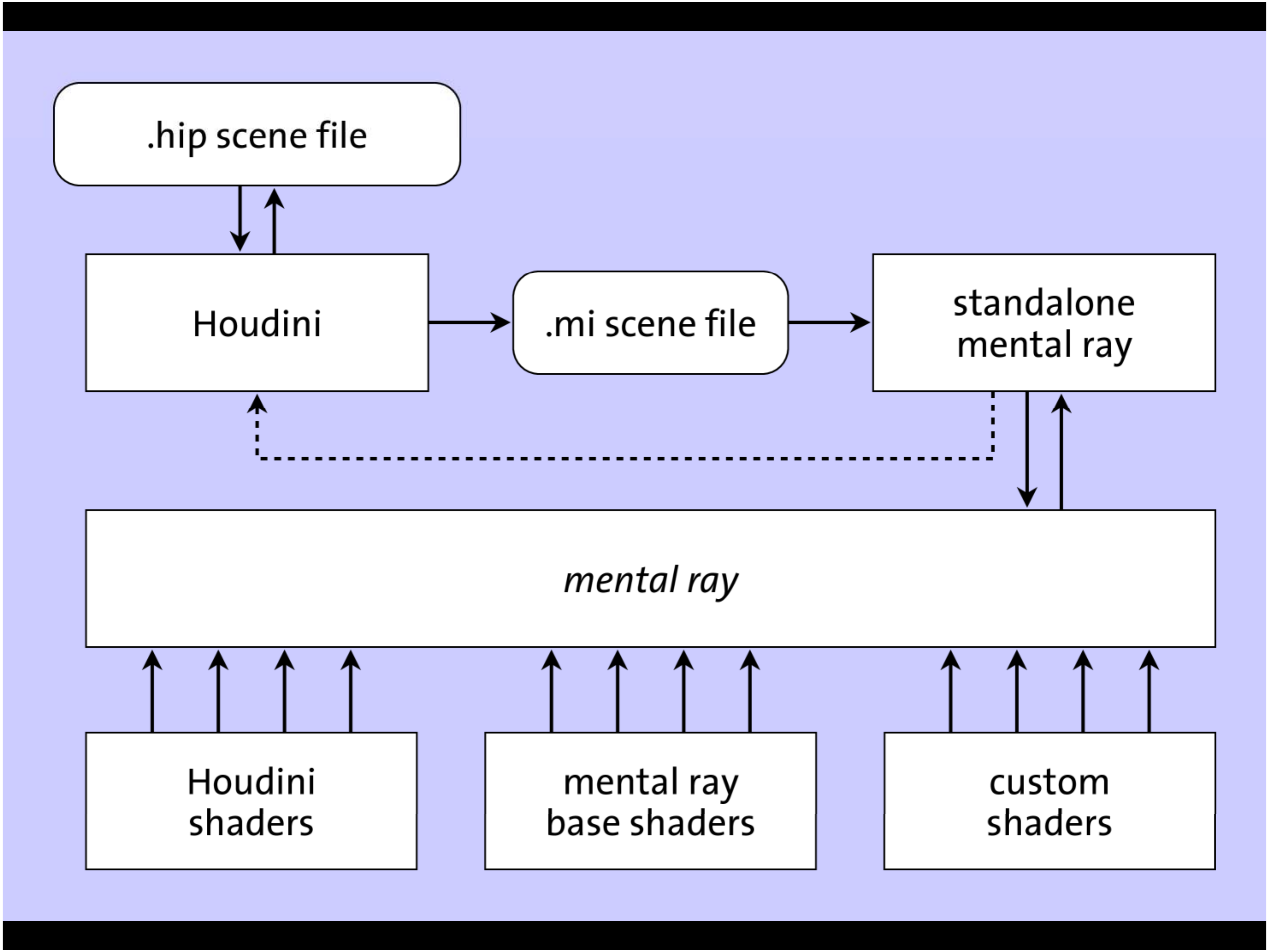


application scene file
application
.mi scene file
standalone mental ray
mental ray
application shaders
mental ray base shaders
custom shaders


.xsi scene file
XSI
.mi scene file
standalone mental ray
mental ray
sibase.so(.dll) shaders
mental ray base shaders
custom shaders

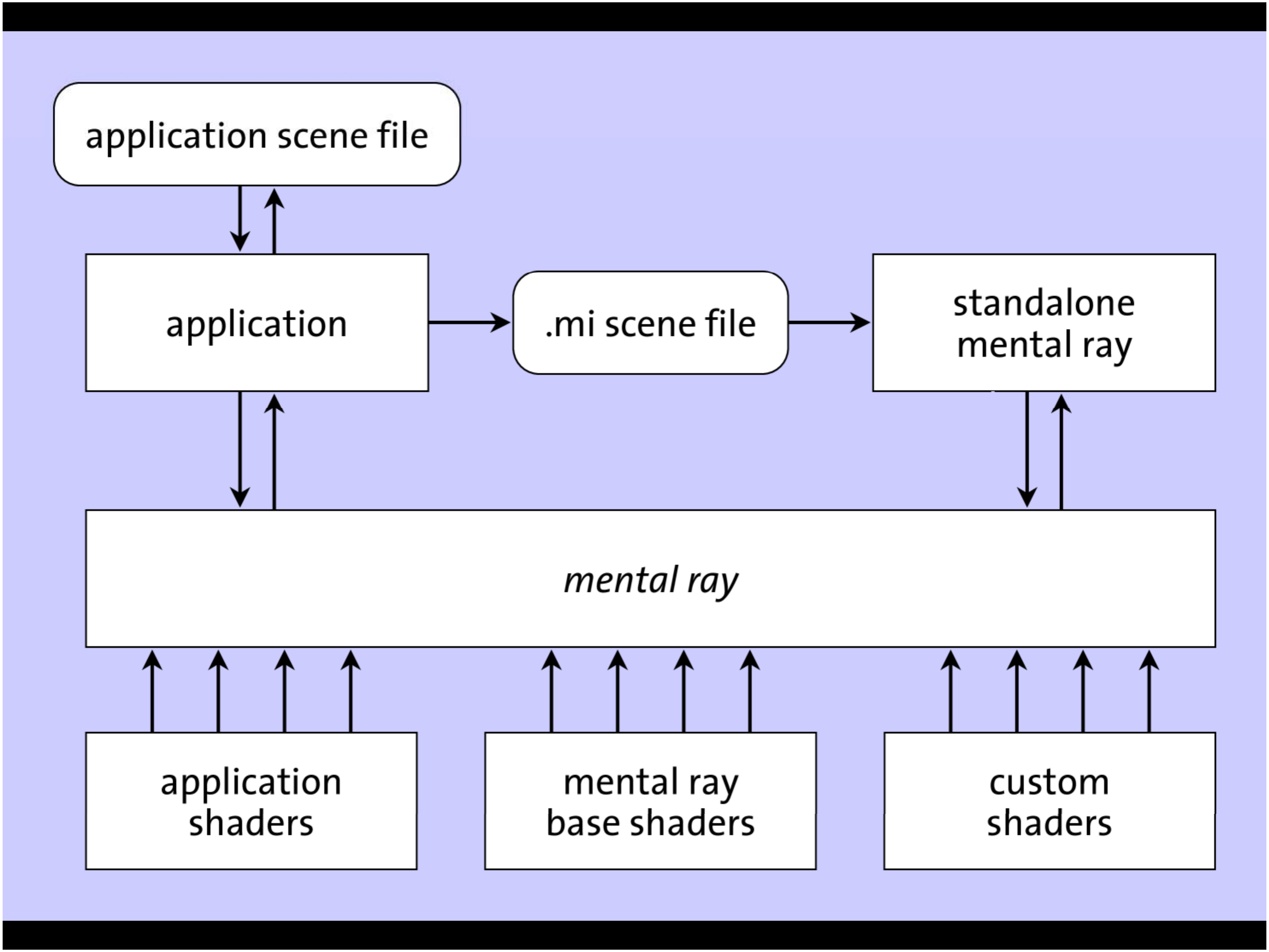

.3ds scene file
3ds max
.mi scene file
standalone mental ray
mental ray
3dsmax*.dll shaders
mental ray base shaders
custom shaders

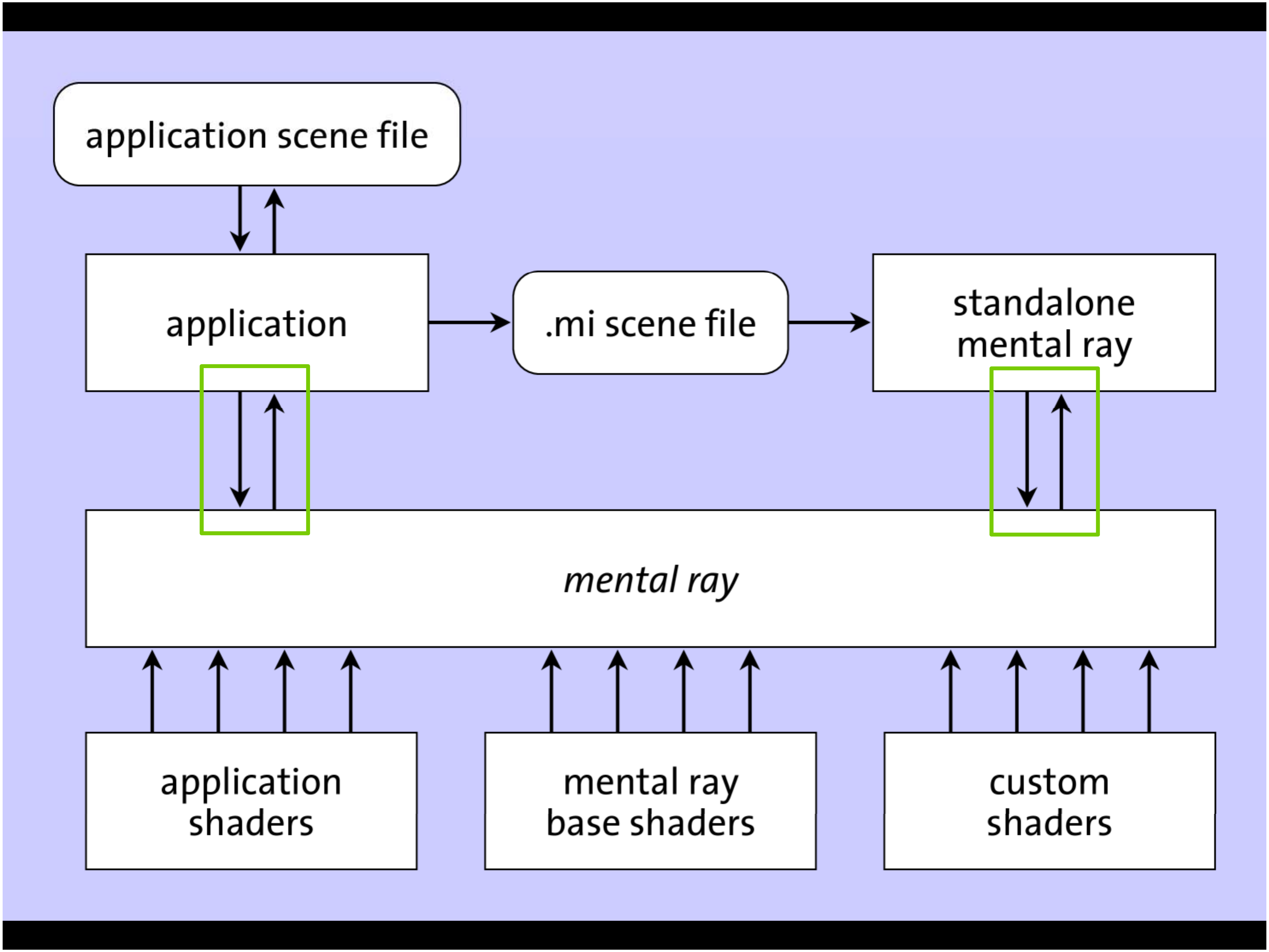

.ma/.mb scene file
Maya
.mi scene file
standalone mental ray
mental ray
mayabase.so(.dll) shaders
mental ray base shaders
custom shaders

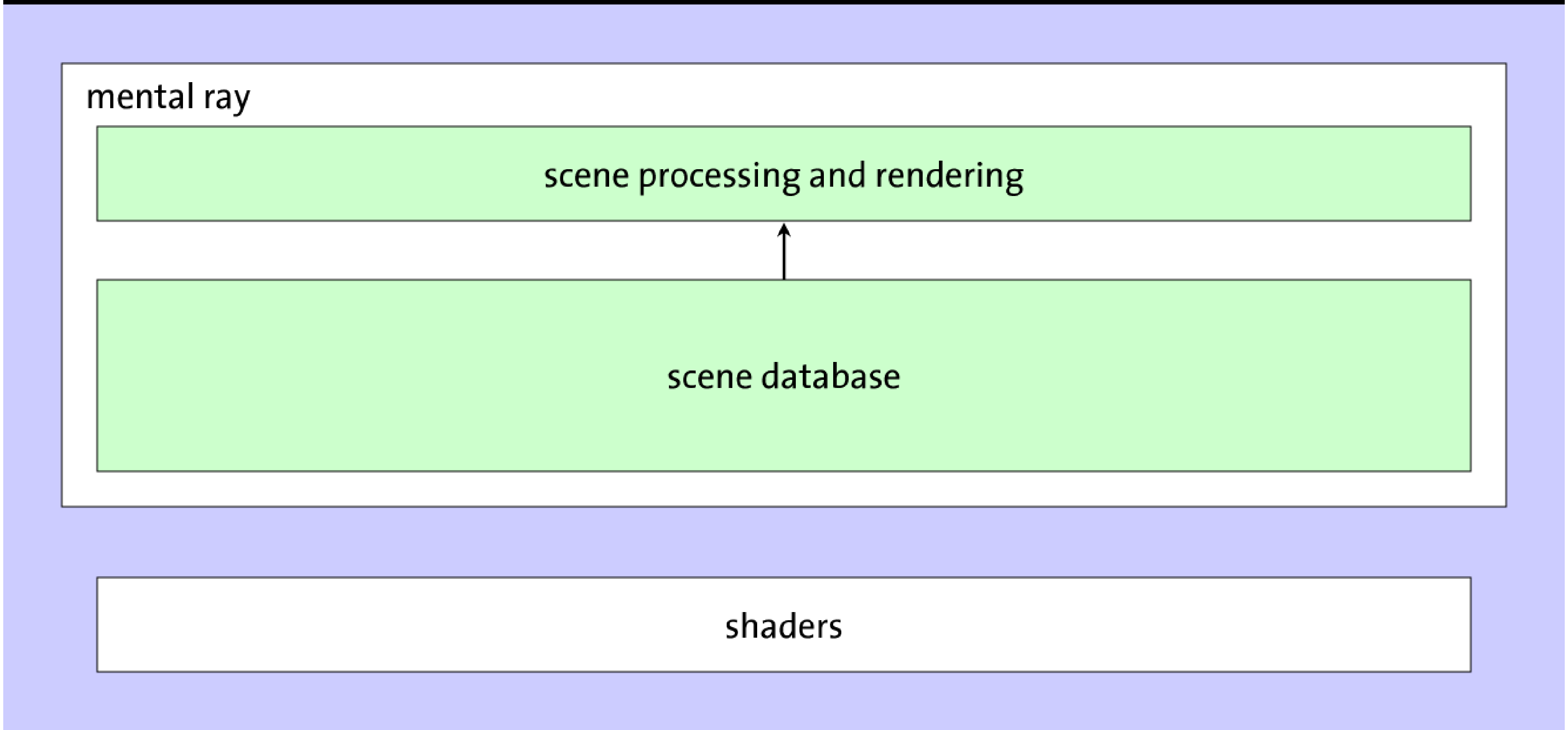

.hip scene file
Houdini
.mi scene file
standalone mental ray
mental ray
Houdini shaders
mental ray base shaders
custom shaders

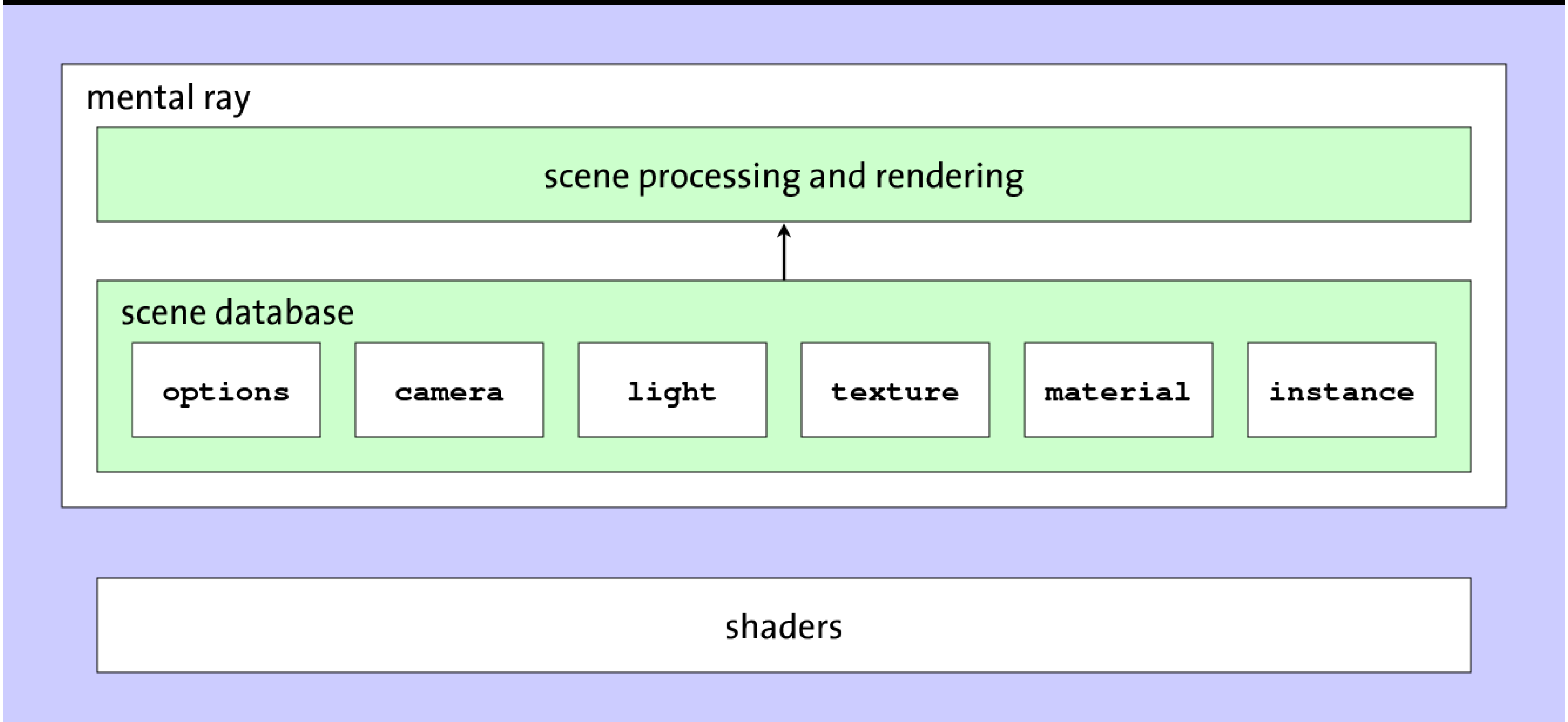
Shaders and the structure of mental ray

Scene data in mental ray is implemented as a database.


application scene file
application
.mi scene file
standalone
mental ray
mental ray
application
shaders
mental ray
base shaders
custom
shaders

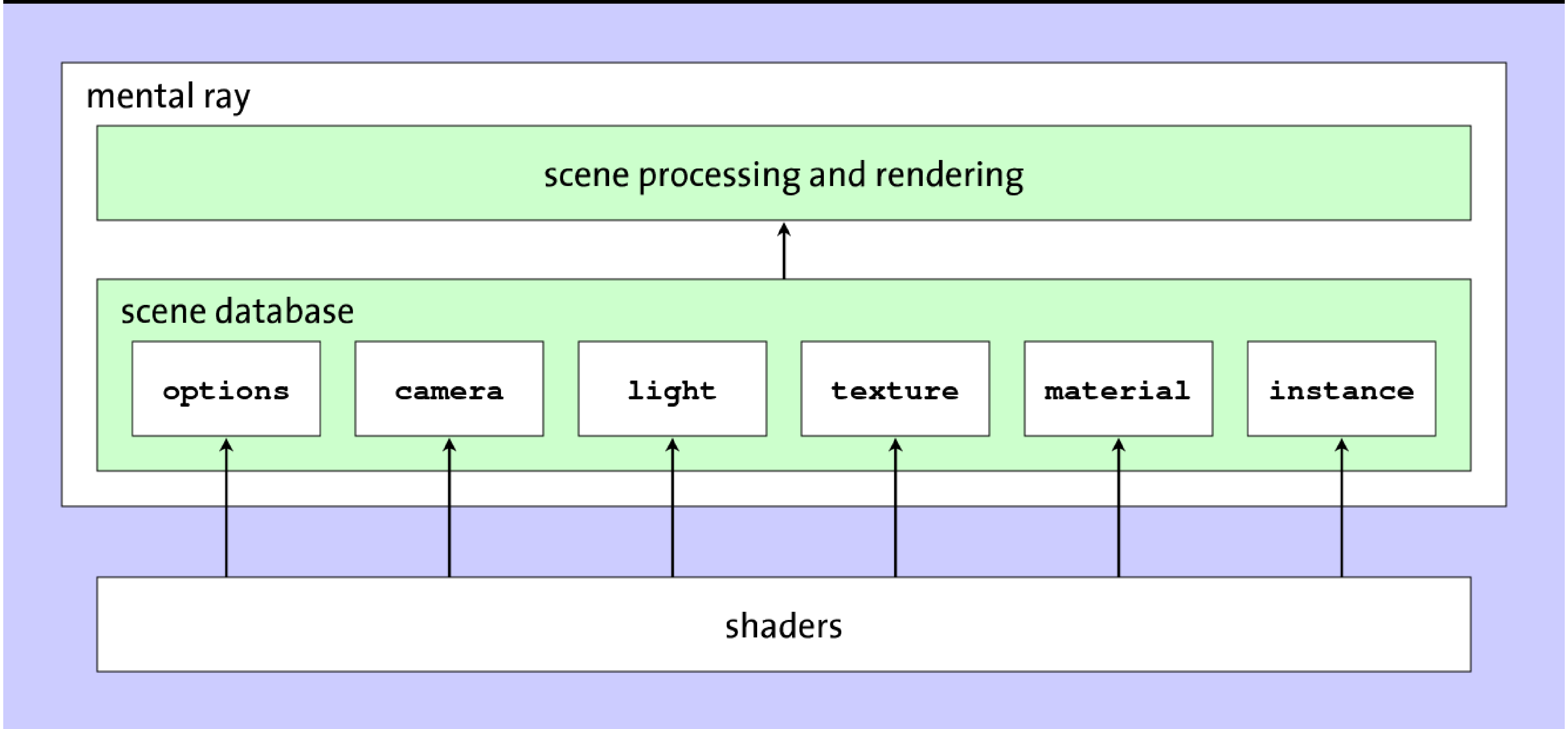

application scene file
application
.mi scene file
standalone
mental ray
mental ray
application
shaders
mental ray
base shaders
custom
shaders

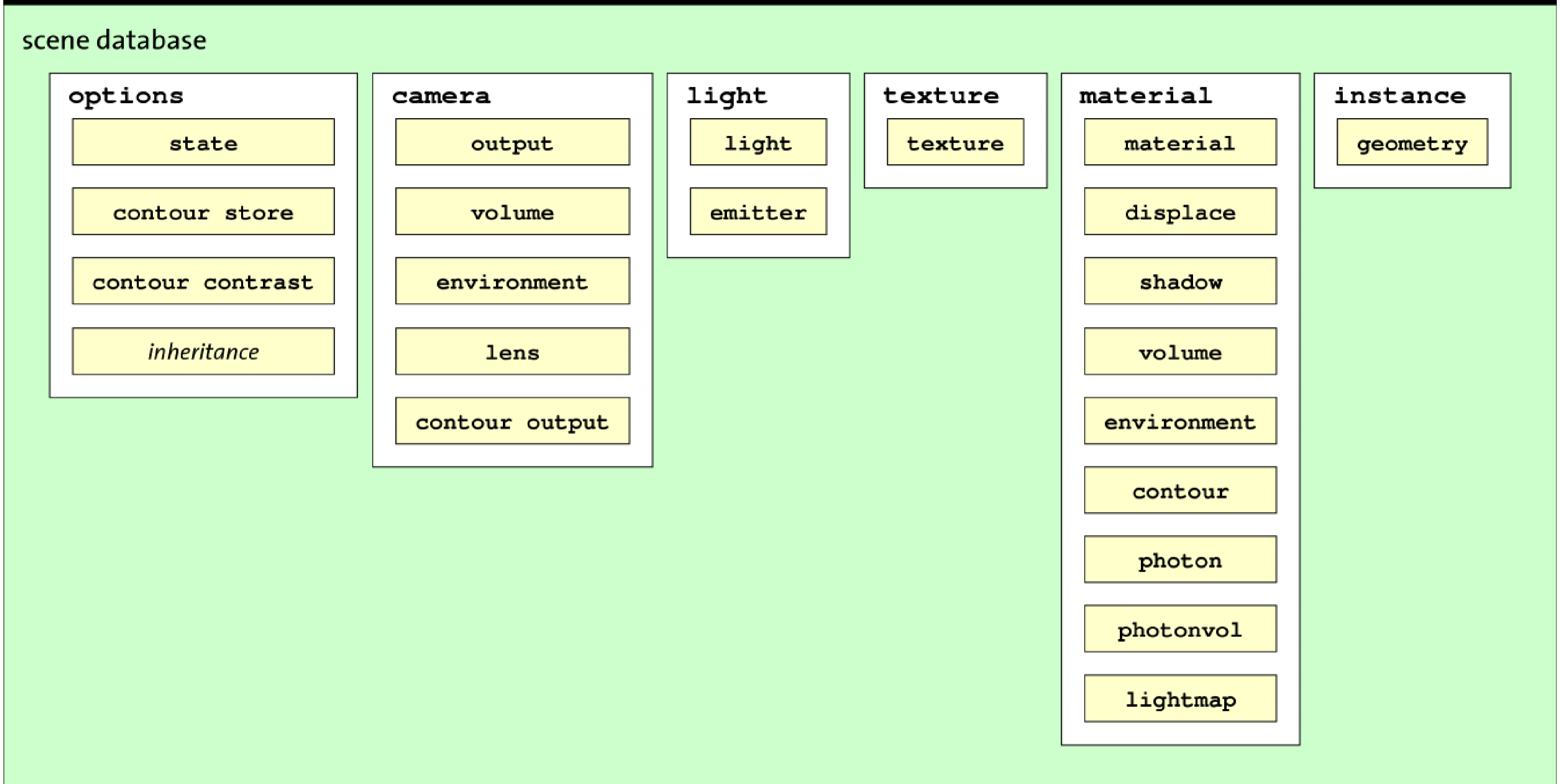

mental ray
scene processing and rendering
scene database
shaders

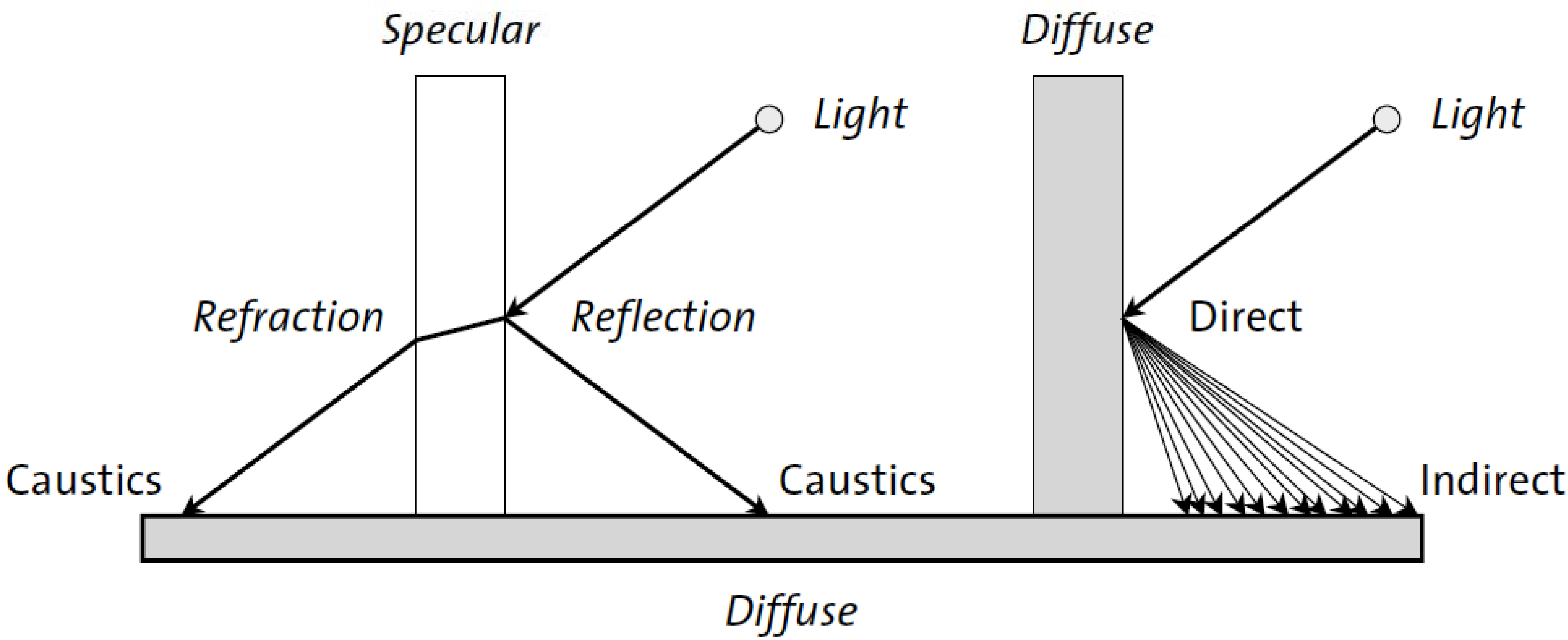

mental ray
scene processing and rendering
scene database
options
camera
light
texture
material
instance
shaders

Shaders and the structure of mental ray

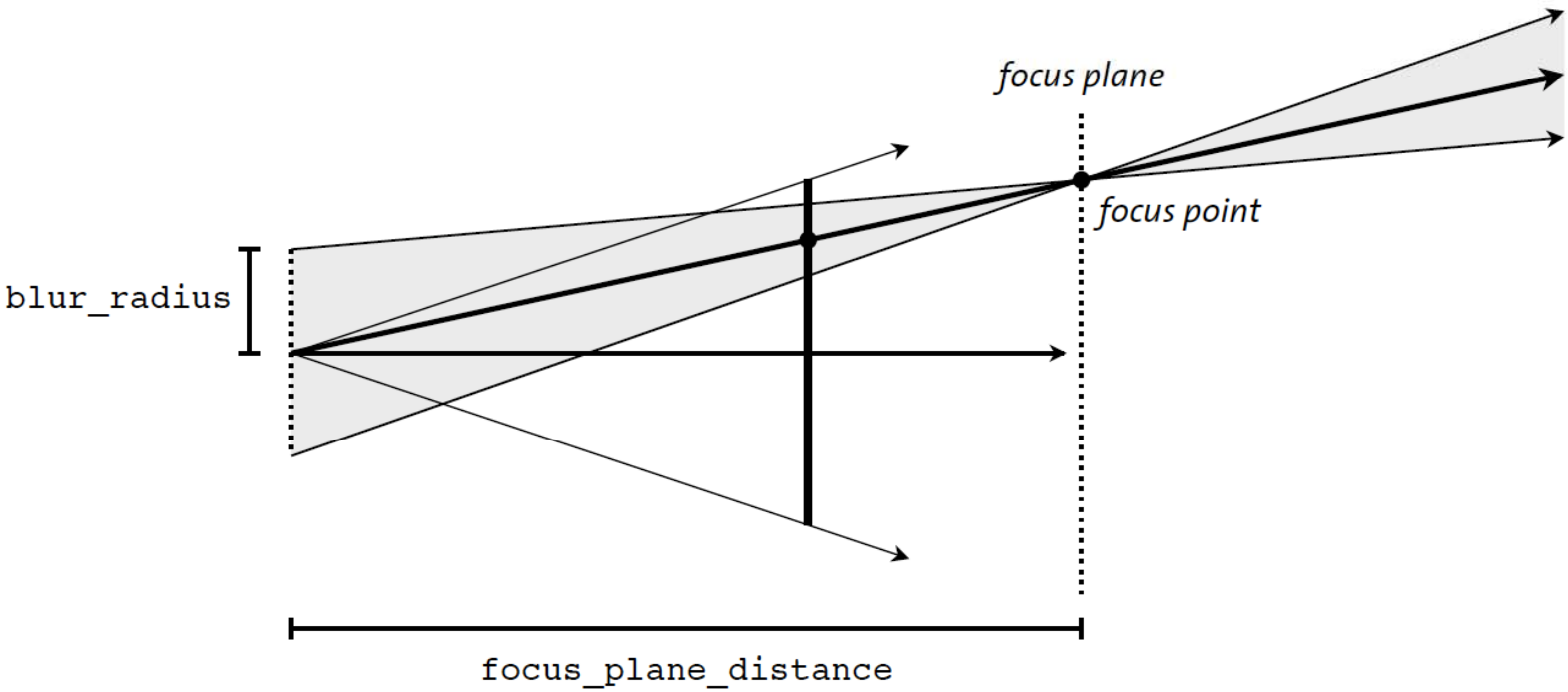
Shaders are represented as elements in the scene database that can modify the behavior of many phases of the rendering pipeline.

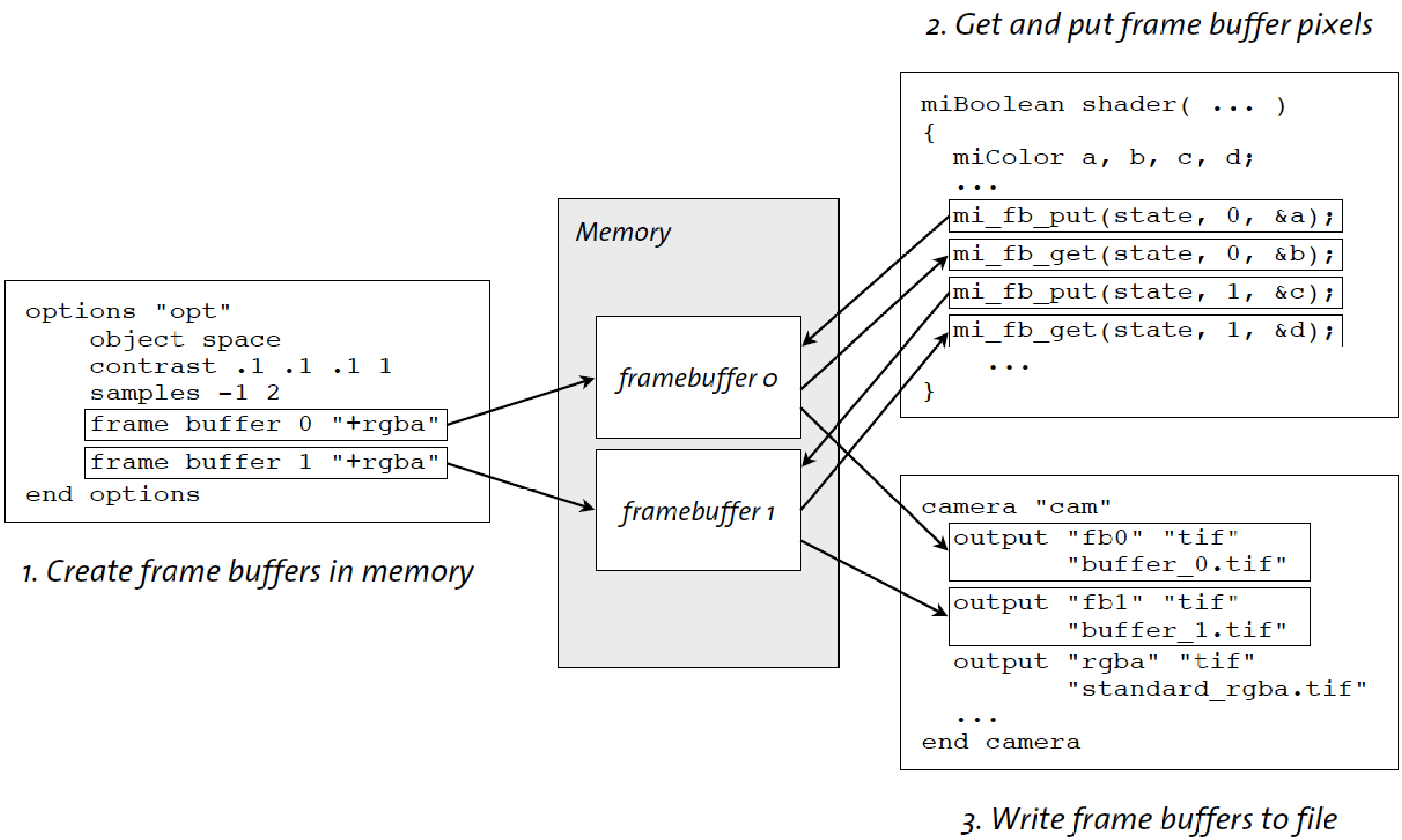

mental ray
scene processing and rendering
scene database
options
camera
light
texture
material
instance
shaders


scene database
options
state
contour store
contour contrast
inheritance
camera
output
volume
environment
lens
contour output
light
light
emitter
texture
texture
material
material
displace
shadow
volume
environment
contour
photon
photonvol
lightmap
instance
geometry



Specular
Diffuse
Light
Light
Refraction
Reflection
Direct
Caustics
Caustics
Indirect
Diffuse


focus plane
focus point
blur_radius
focus_plane_distance


2. Get and put frame buffer pixels
miBoolean shader(...)
{
miColor a, b, c, d;
...
mi_fb_put(state, 0, &a);
mi_fb_get(state, 0, &b);
mi_fb_put(state, 1, &c);
mi_fb_get(state, 1, &d);
...
}
Memory
framebuffer 0
framebuffer 1
options "opt"
object space
contrast .1 .1 .1 1
samples -1 2
frame buffer 0 "+rgba"
frame buffer 1 "+rgba"
end options
1. Create frame buffers in memory
camera "cam"
output "fb0" "tif"
"buffer_0.tif"
output "fb1" "tif"
"buffer_1.tif"
output "rgba" "tif"
"standard_rgba.tif"
...
end camera
3. Write frame buffers to file

Strategy of the new shader book

2

Strategy of the new shader book

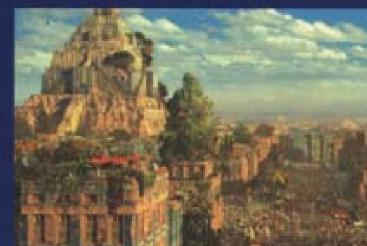
Beginning shader programmers needed a tutorial to complement the mental ray reference handbooks.

T. Driemeyer

mental ray Handbooks Vol. 1

Rendering with mental ray®

Third, completely revised edition



SpringerWienNewYork

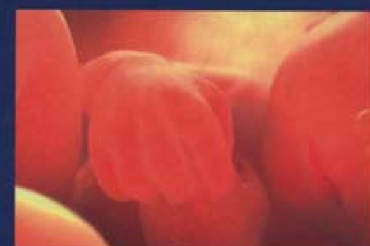
with cd-rom

T. Driemeyer
R. Herken (eds.)

mental ray Handbooks Vol. 2

Programming mental ray®

Third, completely revised edition

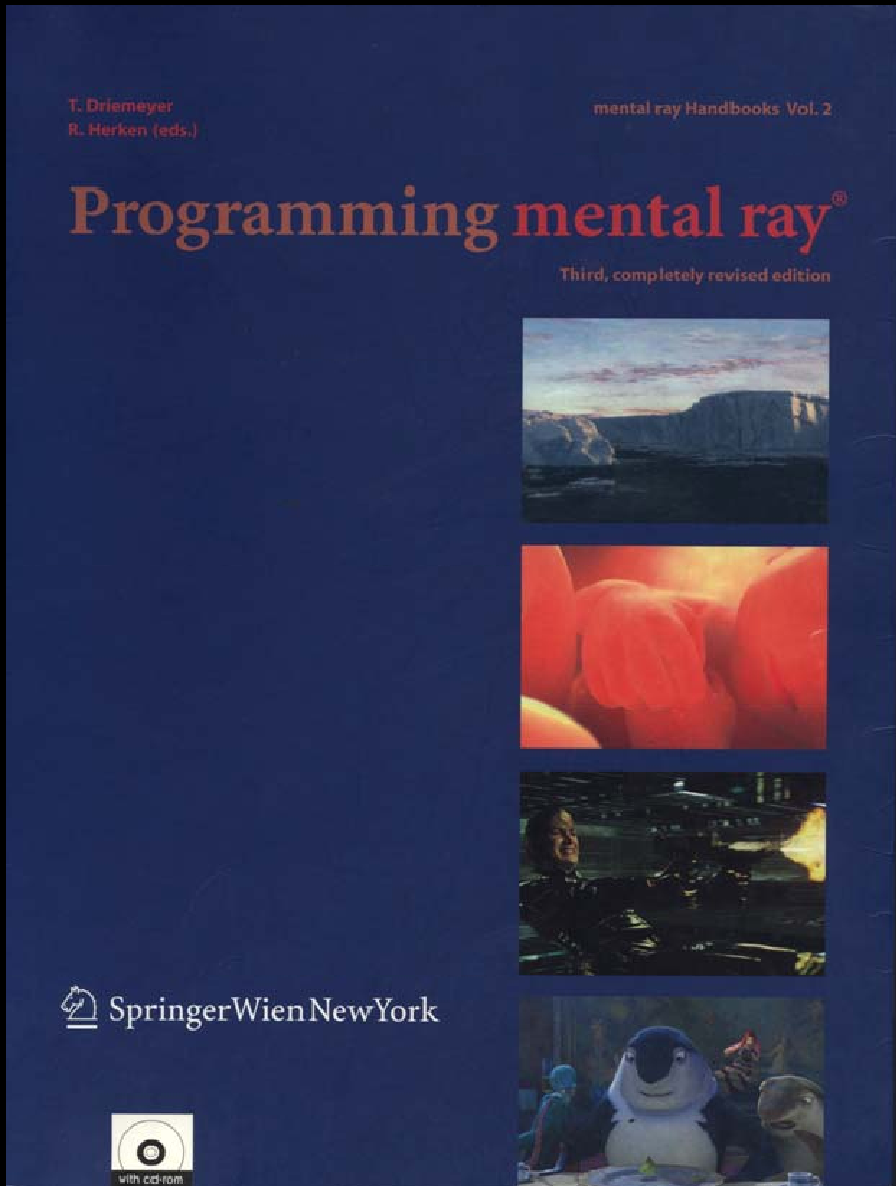


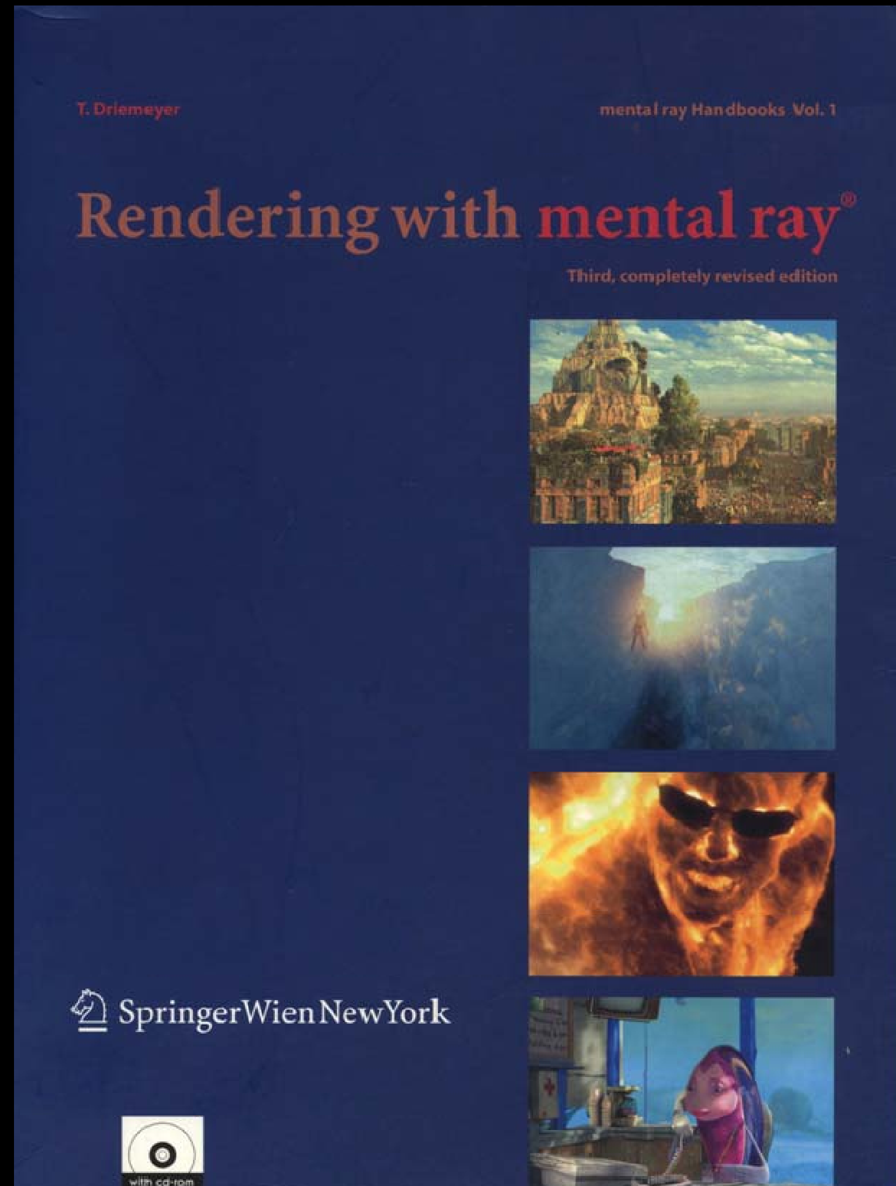
SpringerWienNewYork

with cd-rom

#3?

[redacted]

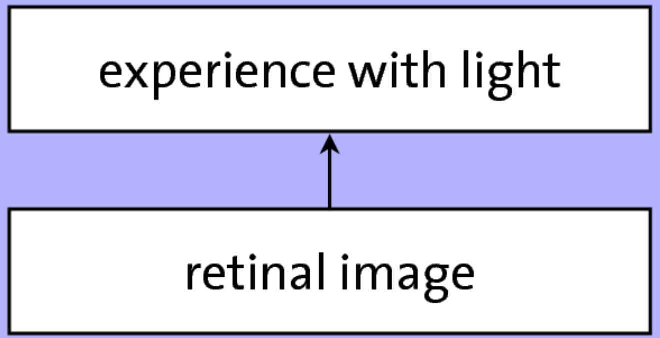

T. Driemeyer
R. Herken (eds.)
mental ray Handbooks Vol. 2
Programming mental ray®
Third, completely revised edition
SpringerWienNewYork
with cd-rom

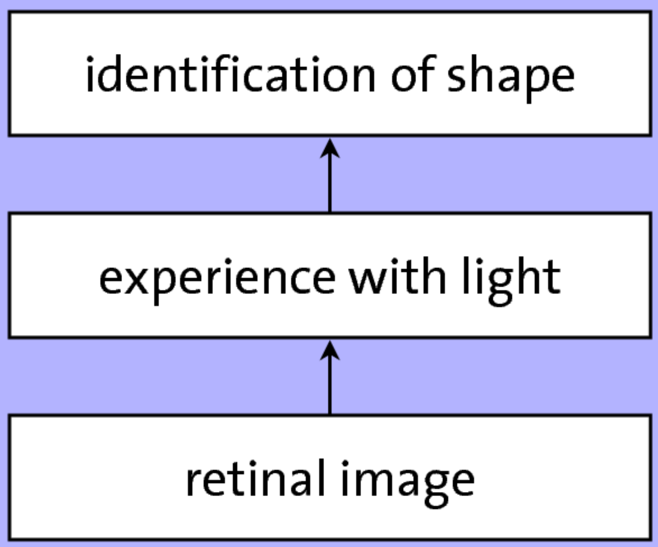

T. Driemeyer
mental ray Handbooks Vol. 1
Rendering with mental ray®
Third, completely revised edition
SpringerWienNewYork
with cd-rom

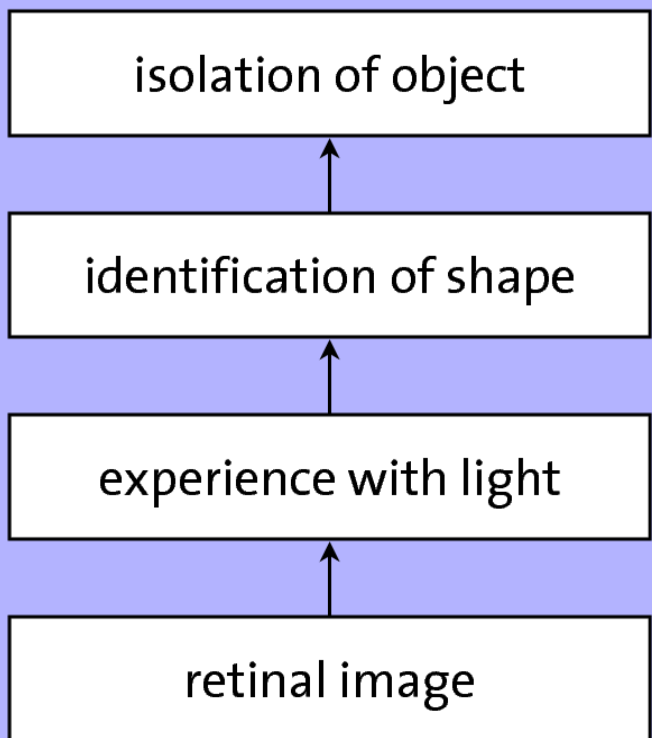
Strategy of the new shader book

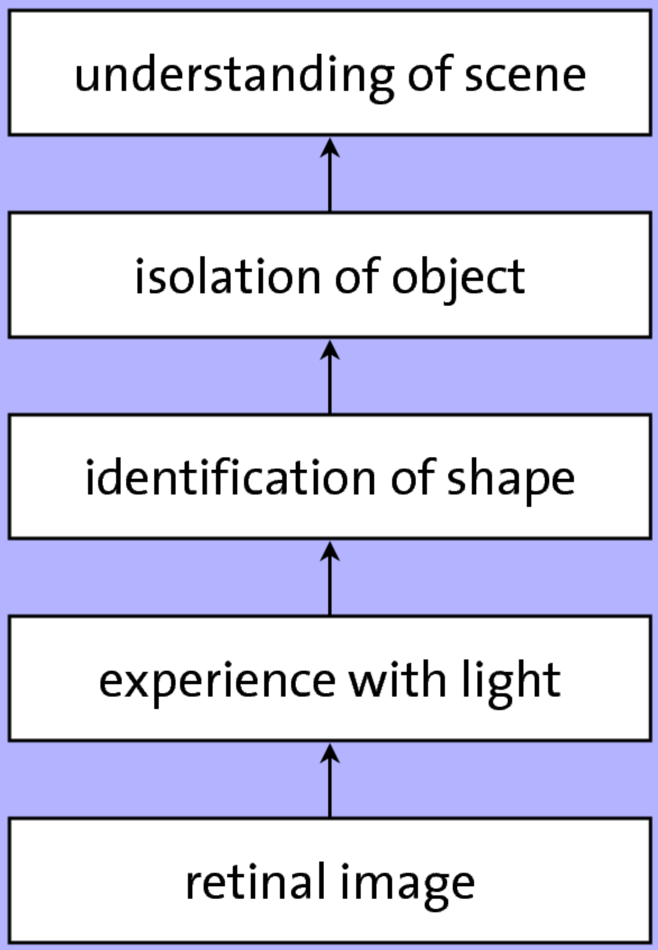
The order of shader presentation in the new book is based on how we see, not on how the underlying software is organized.

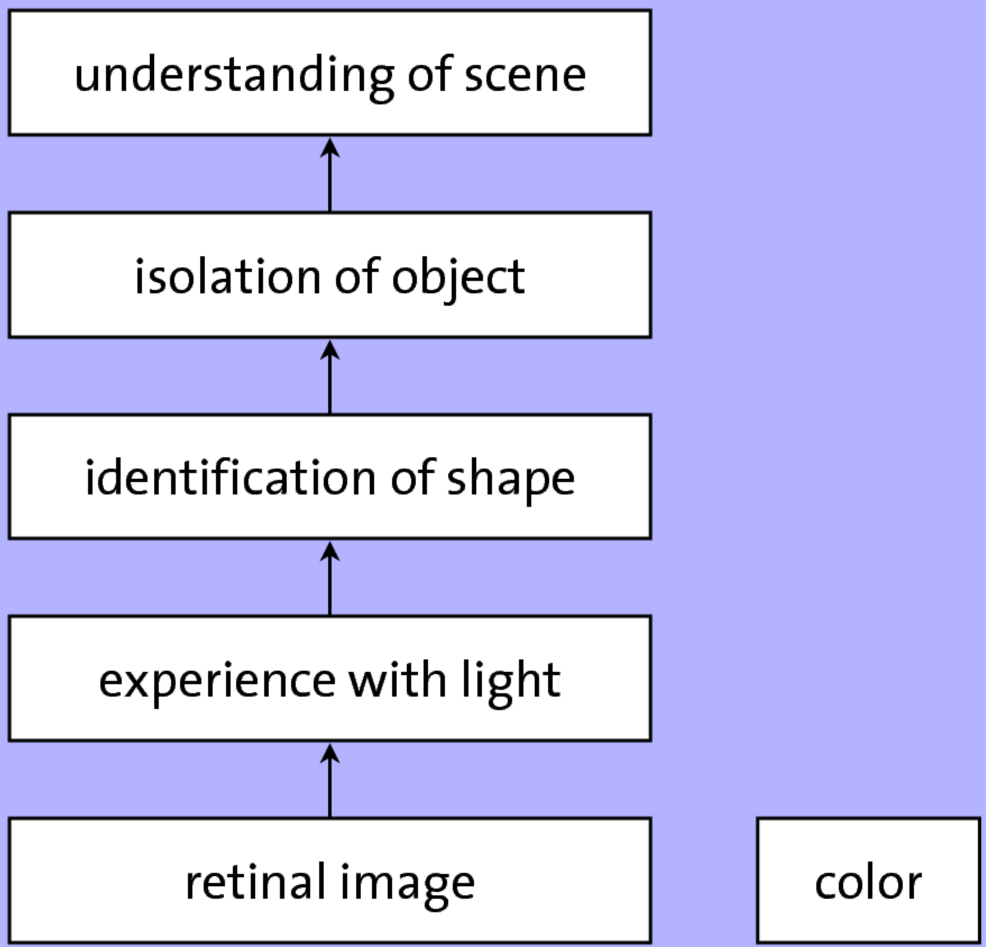
retinal image

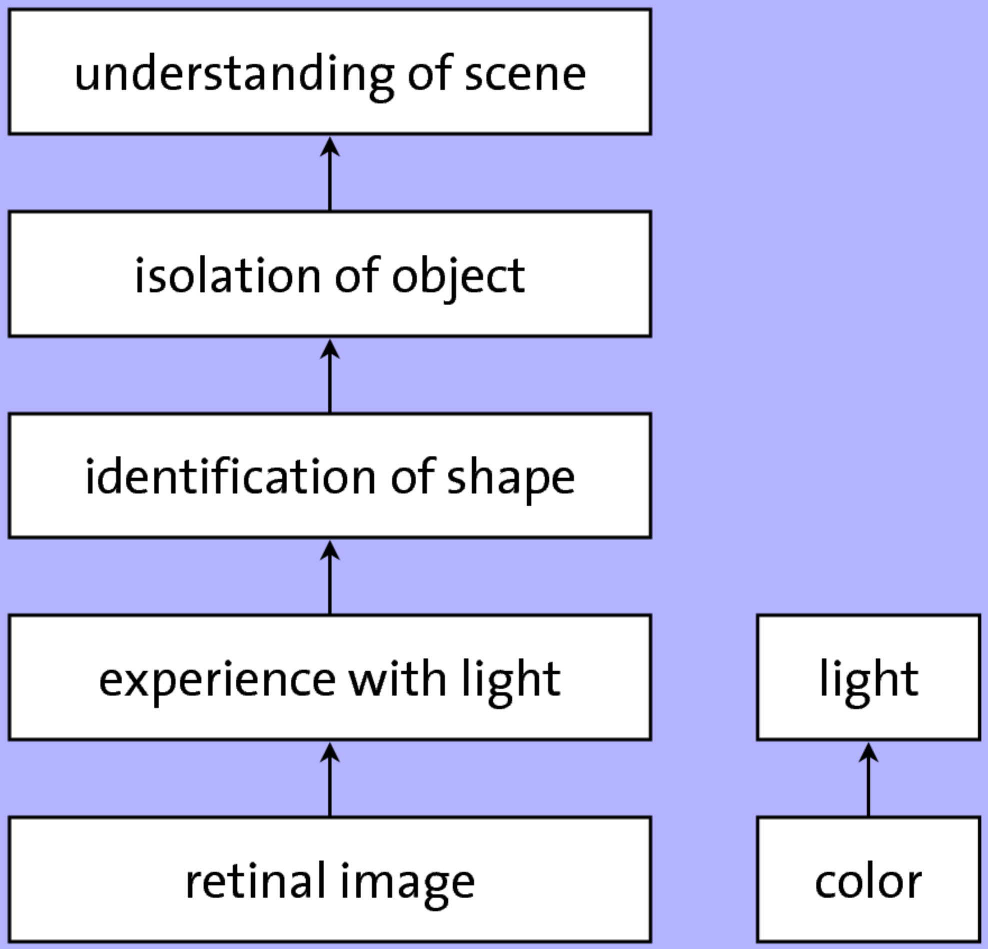

experience with light
retinal image

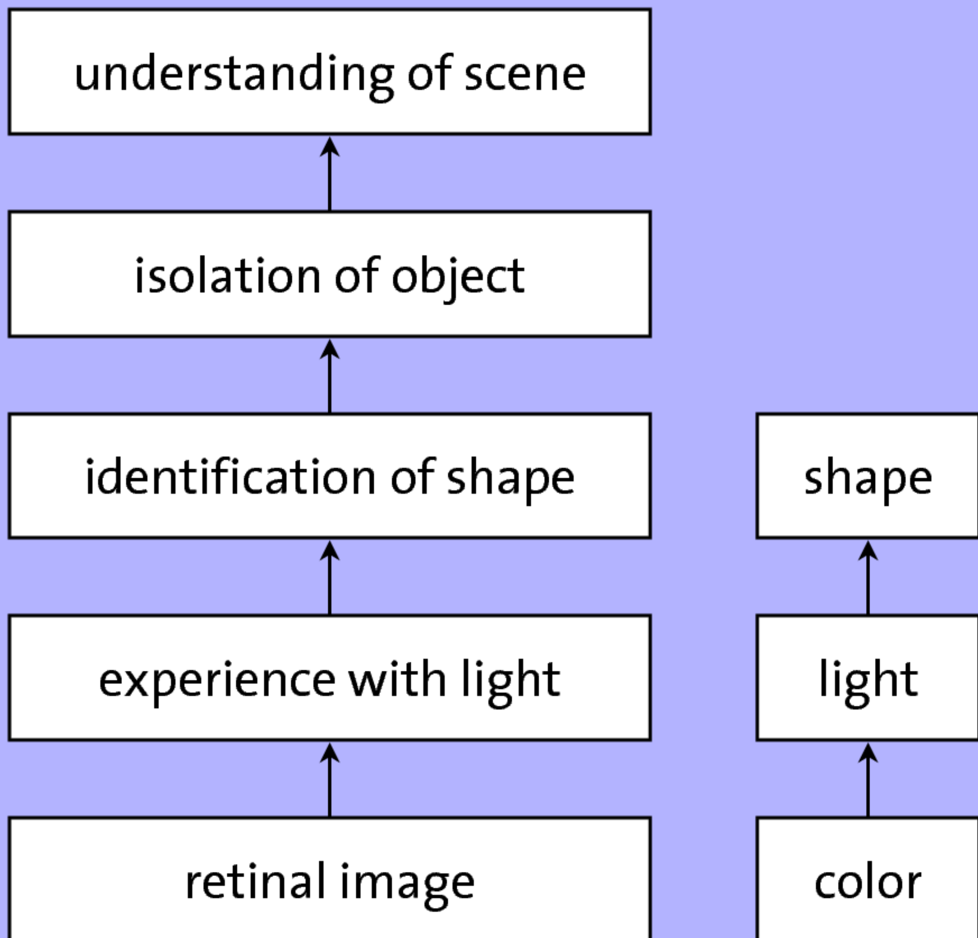

identification of shape
experience with light
retinal image

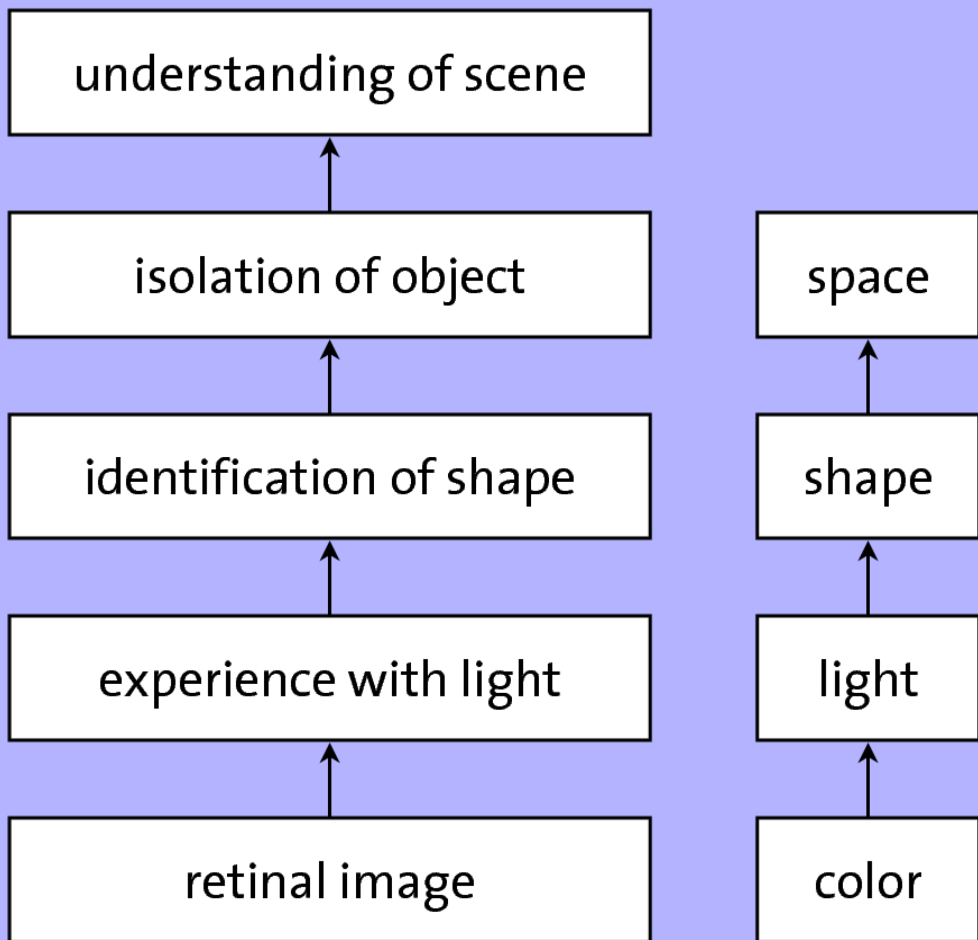

isolation of object
identification of shape
experience with light
retinal image

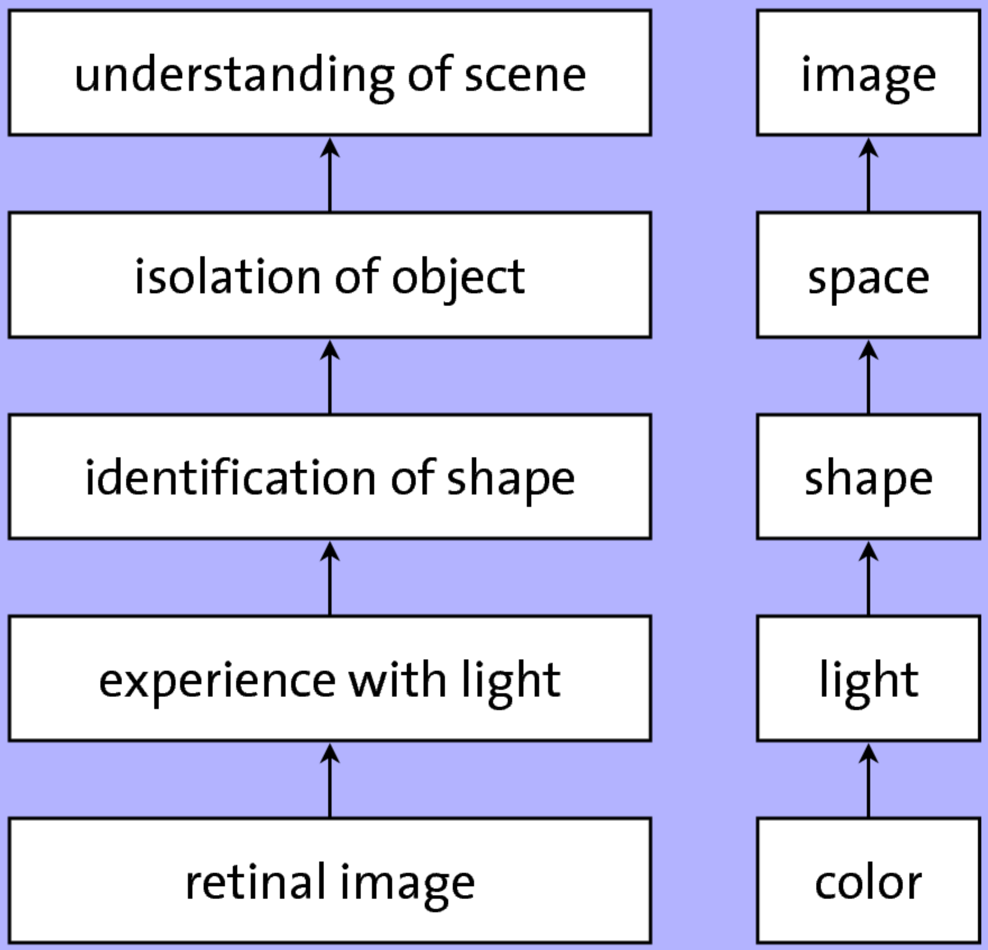

understanding of scene
isolation of object
identification of shape
experience with light
retinal image

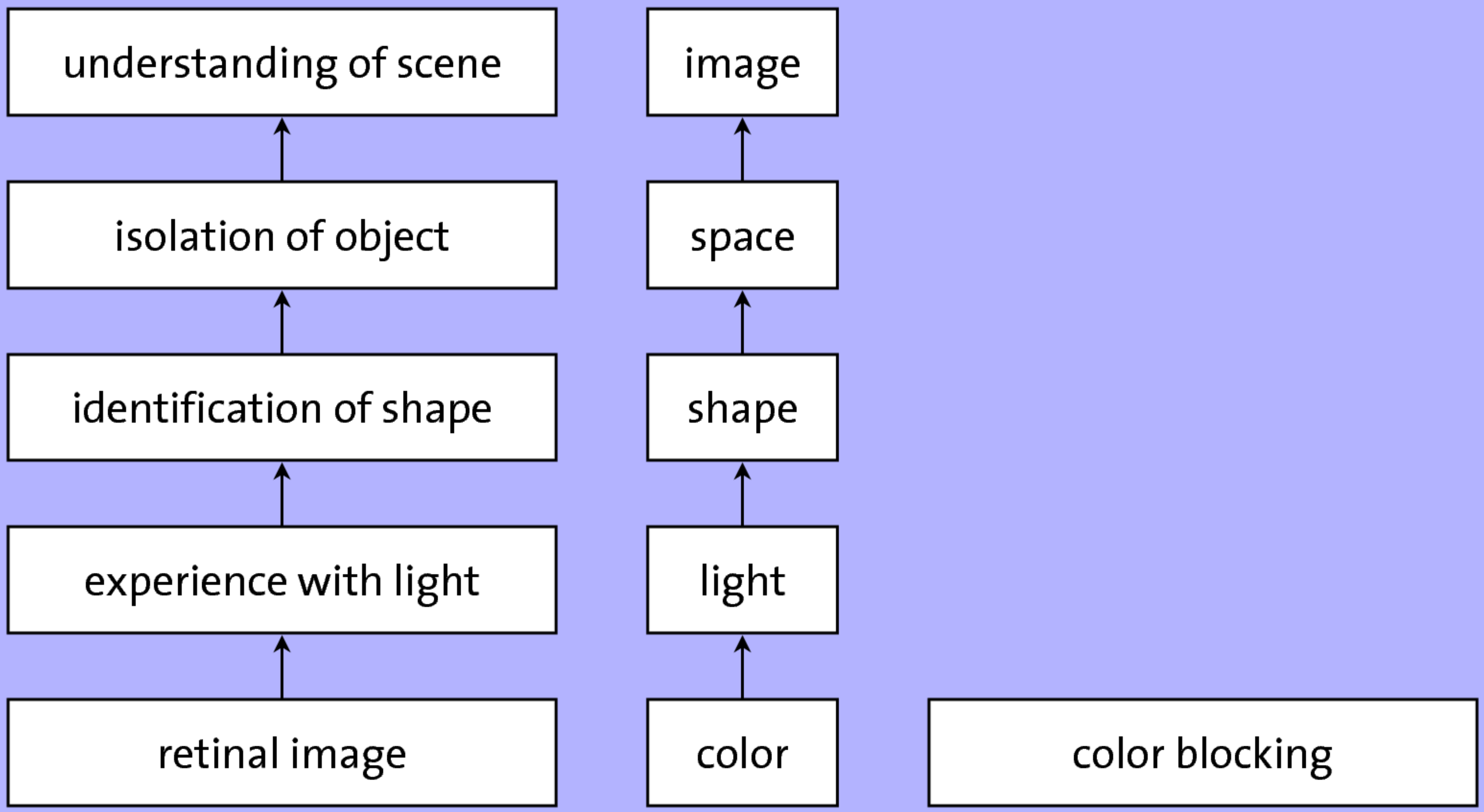

understanding of scene
isolation of object
identification of shape
experience with light
retinal image
color

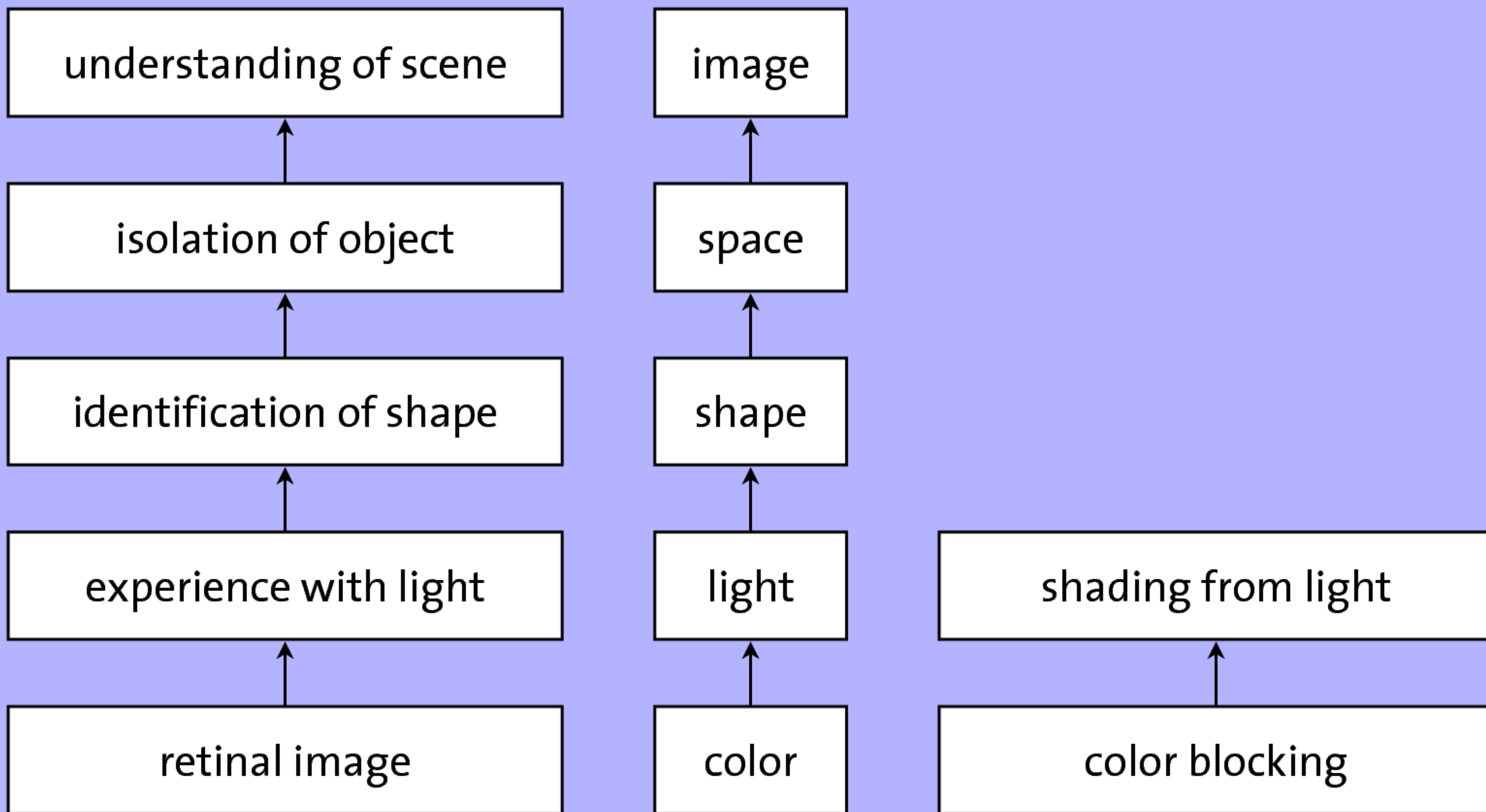

understanding of scene
isolation of object
identification of shape
experience with light
retinal image
light
color

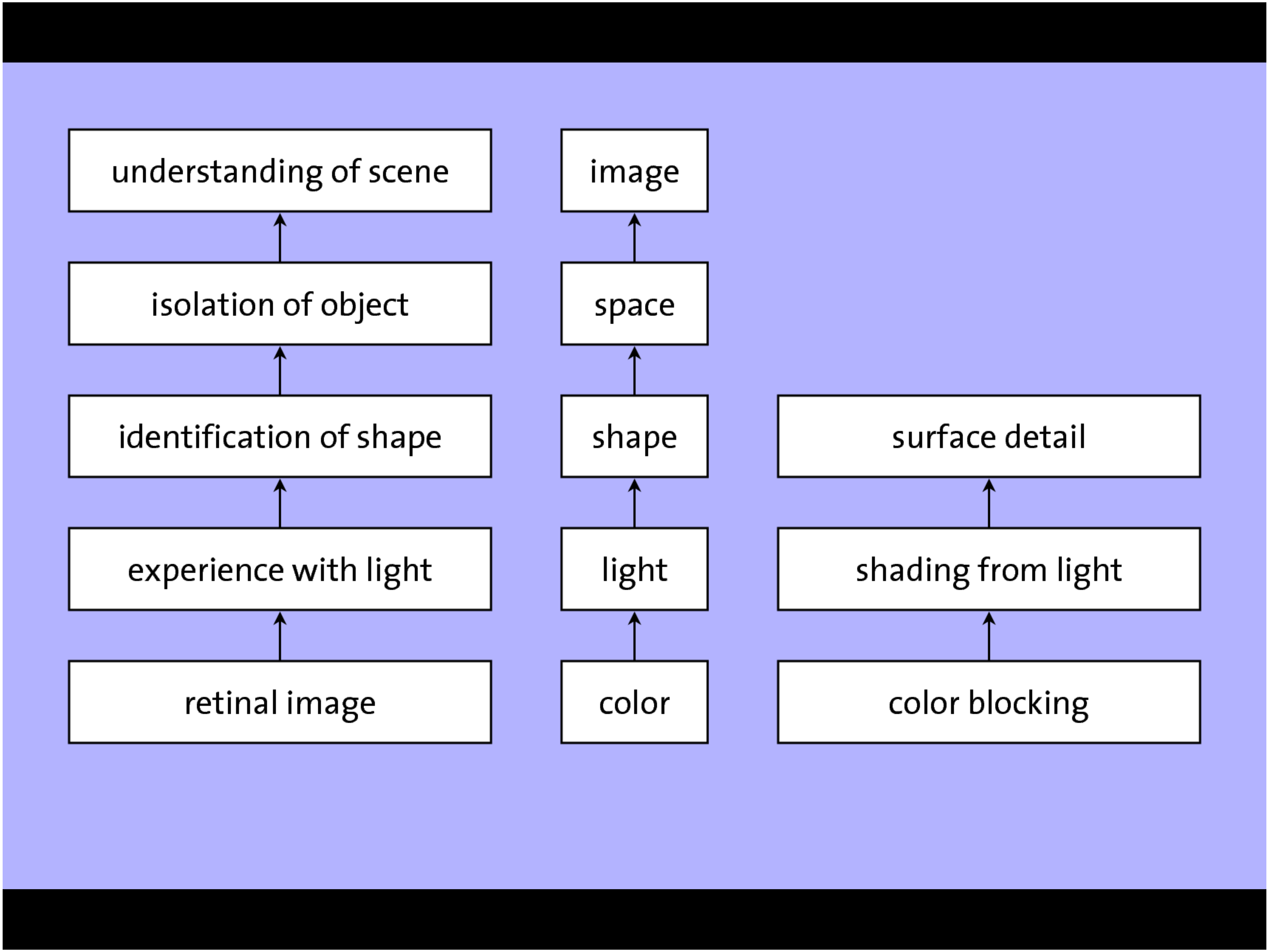

understanding of scene
isolation of object
identification of shape
experience with light
retinal image
shape
light
color

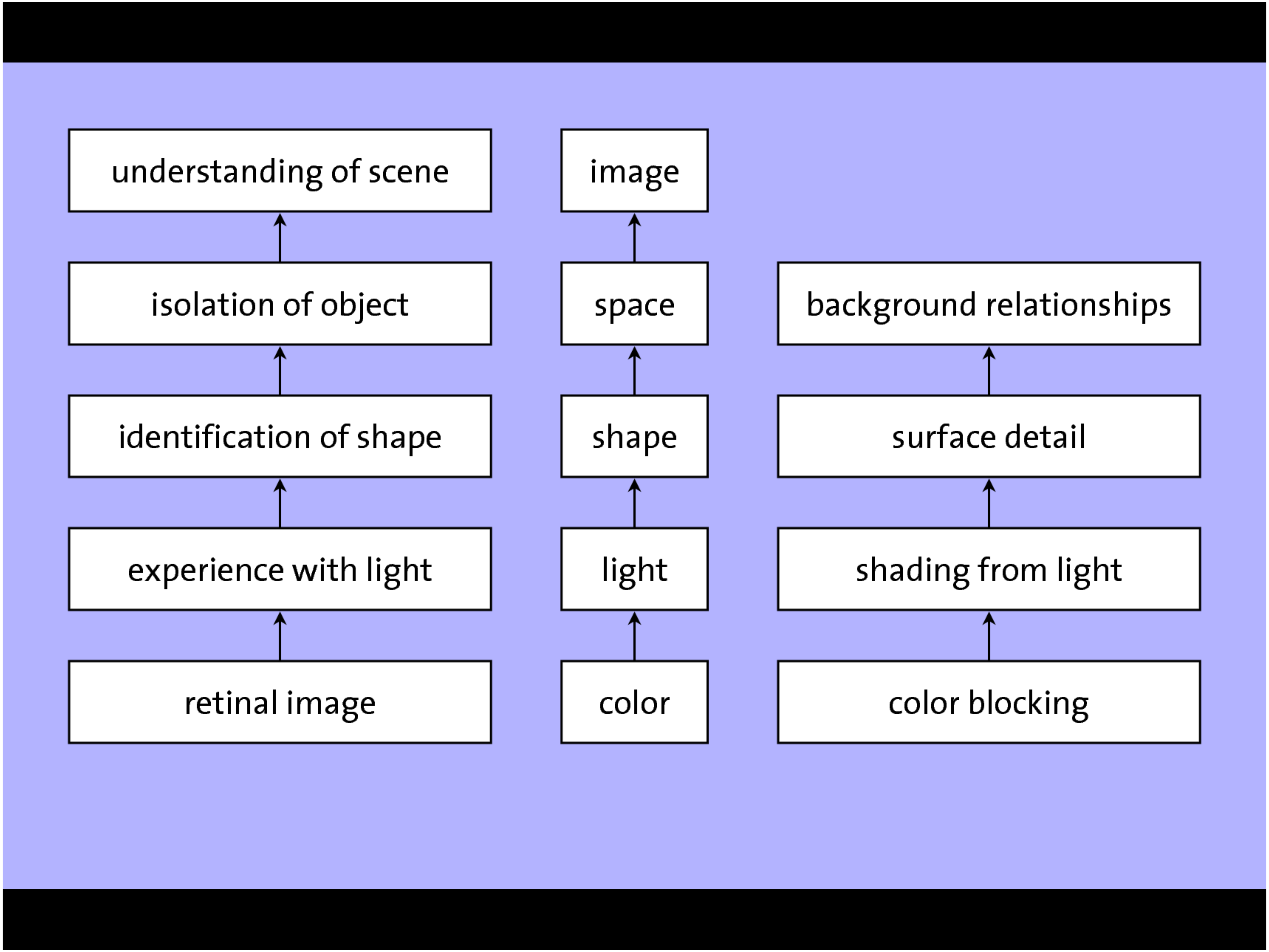

understanding of scene
isolation of object
identification of shape
experience with light
retinal image
space
shape
light
color

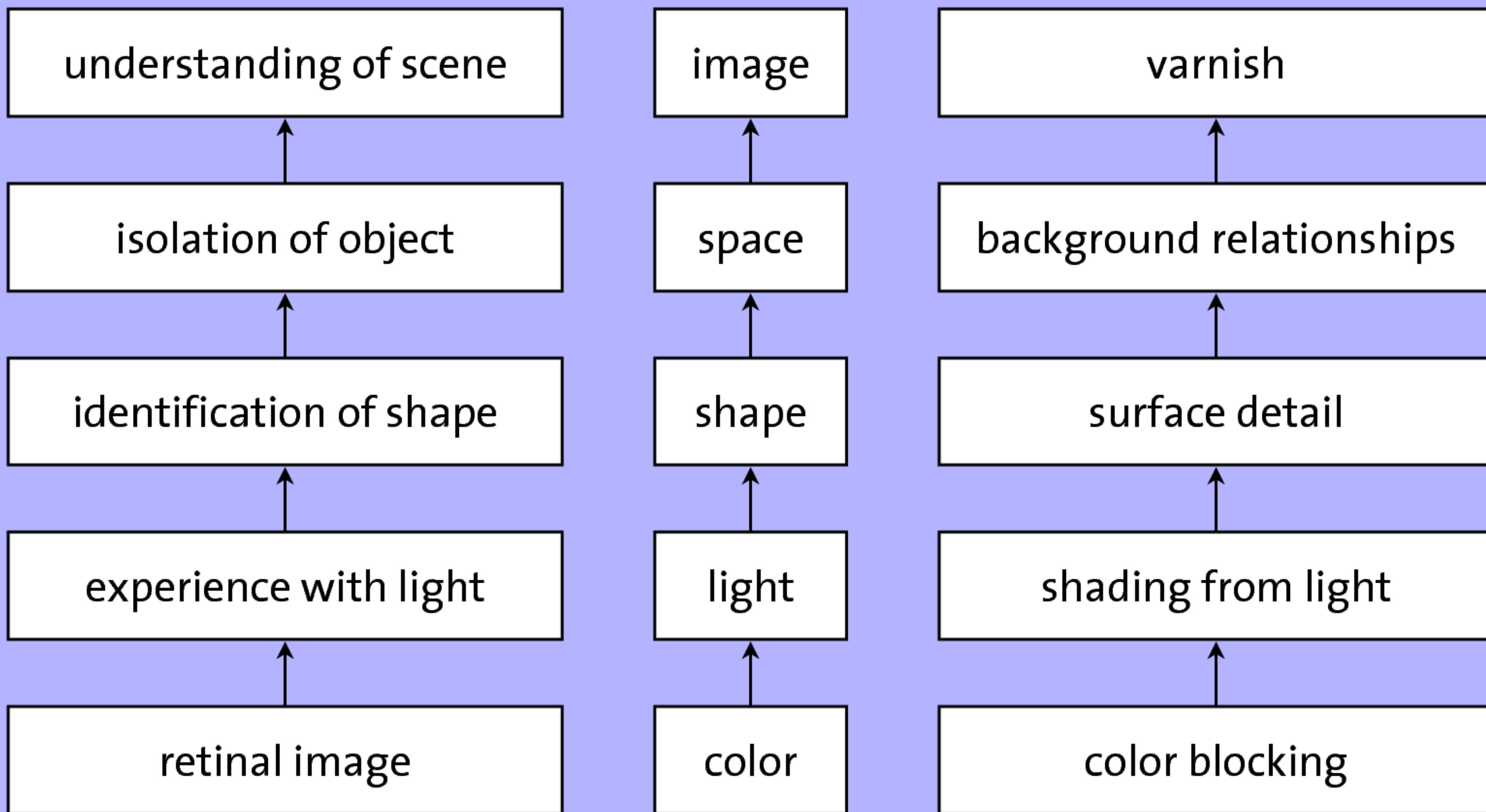

understanding of scene
isolation of object
identification of shape
experience with light
retinal image
image
space
shape
light
color


understanding of scene
isolation of object
identification of shape
experience with light
retinal image
image
space
shape
light
color
color blocking

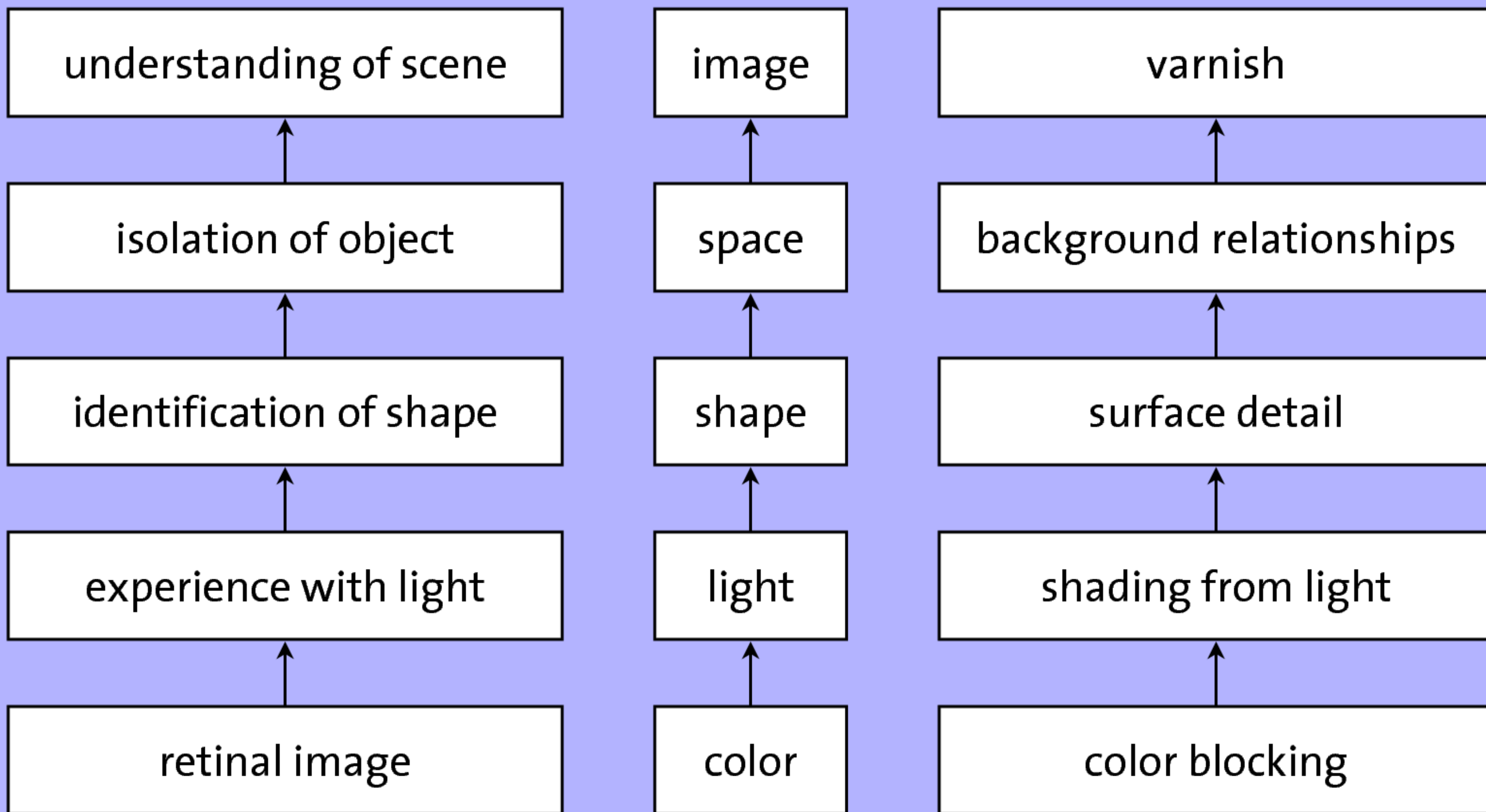

understanding of scene
isolation of object
identification of shape
experience with light
retinal image
image
space
shape
light
color
shading from light
color blocking


understanding of scene
isolation of object
identification of shape
experience with light
retinal image
image
space
shape
light
color
surface detail
shading from light
color blocking

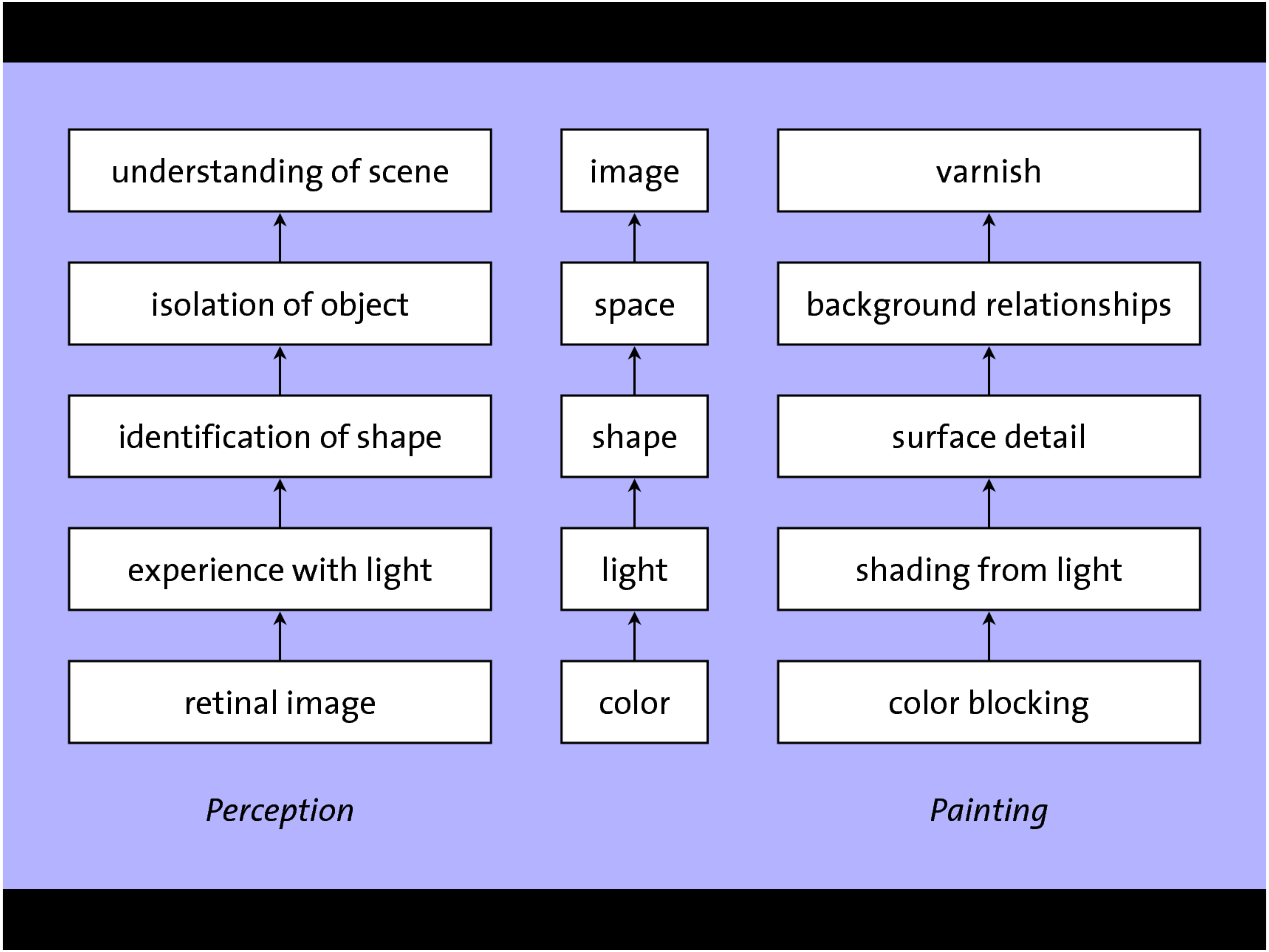

understanding of scene
isolation of object
identification of shape
experience with light
retinal image
image
space
shape
light
color
background relationships
surface detail
shading from light
color blocking

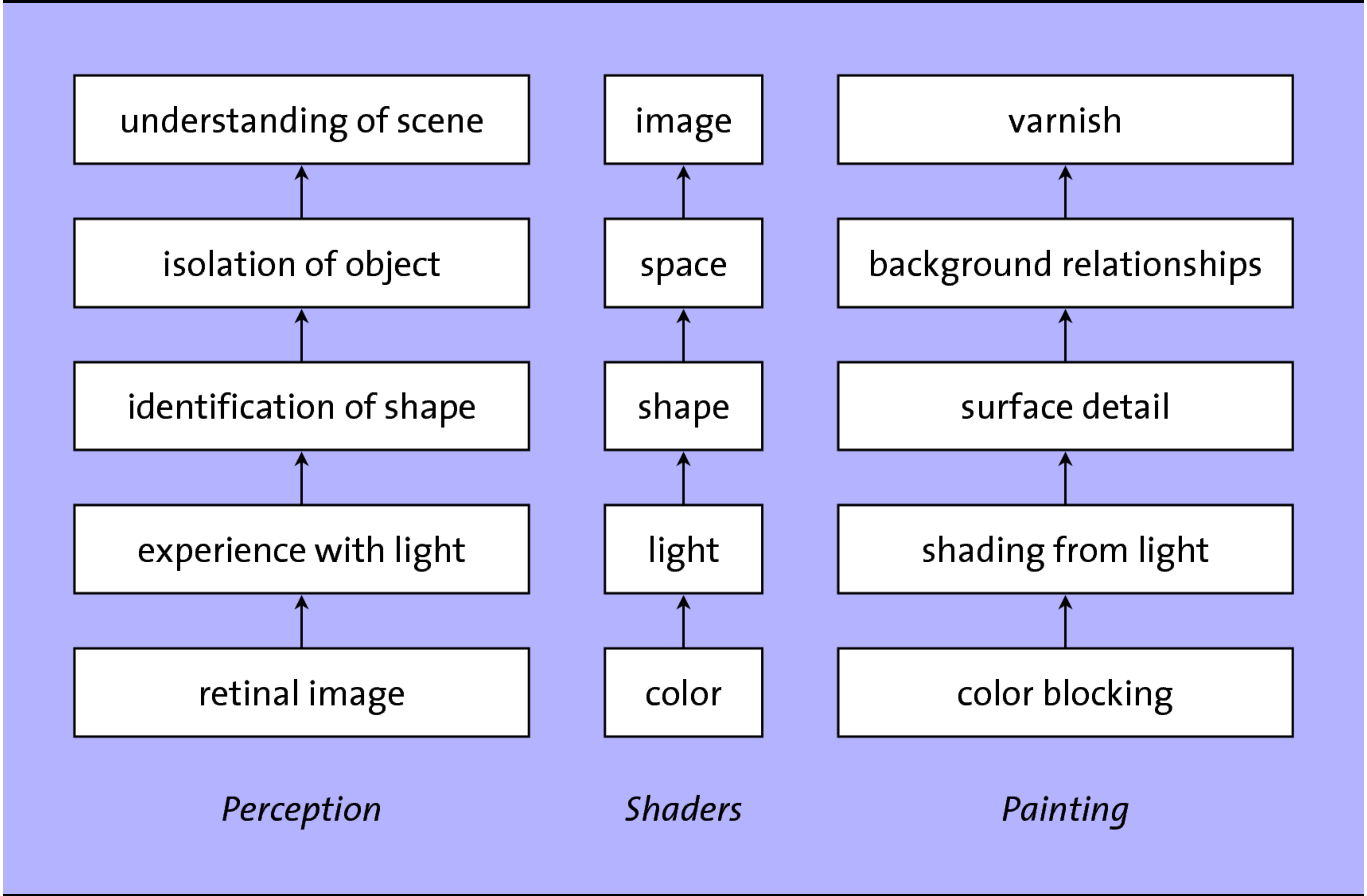


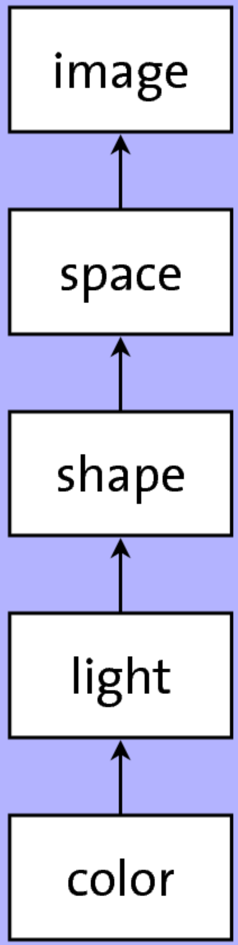
understanding of scene
isolation of object
identification of shape
experience with light
retinal image
image
space
shape
light
color
varnish
background relationships
surface detail
shading from light
color blocking


understanding of scene
isolation of object
identification of shape
experience with light
retinal image
image
space
shape
light
color
varnish
background relationships
surface detail
shading from light
color blocking

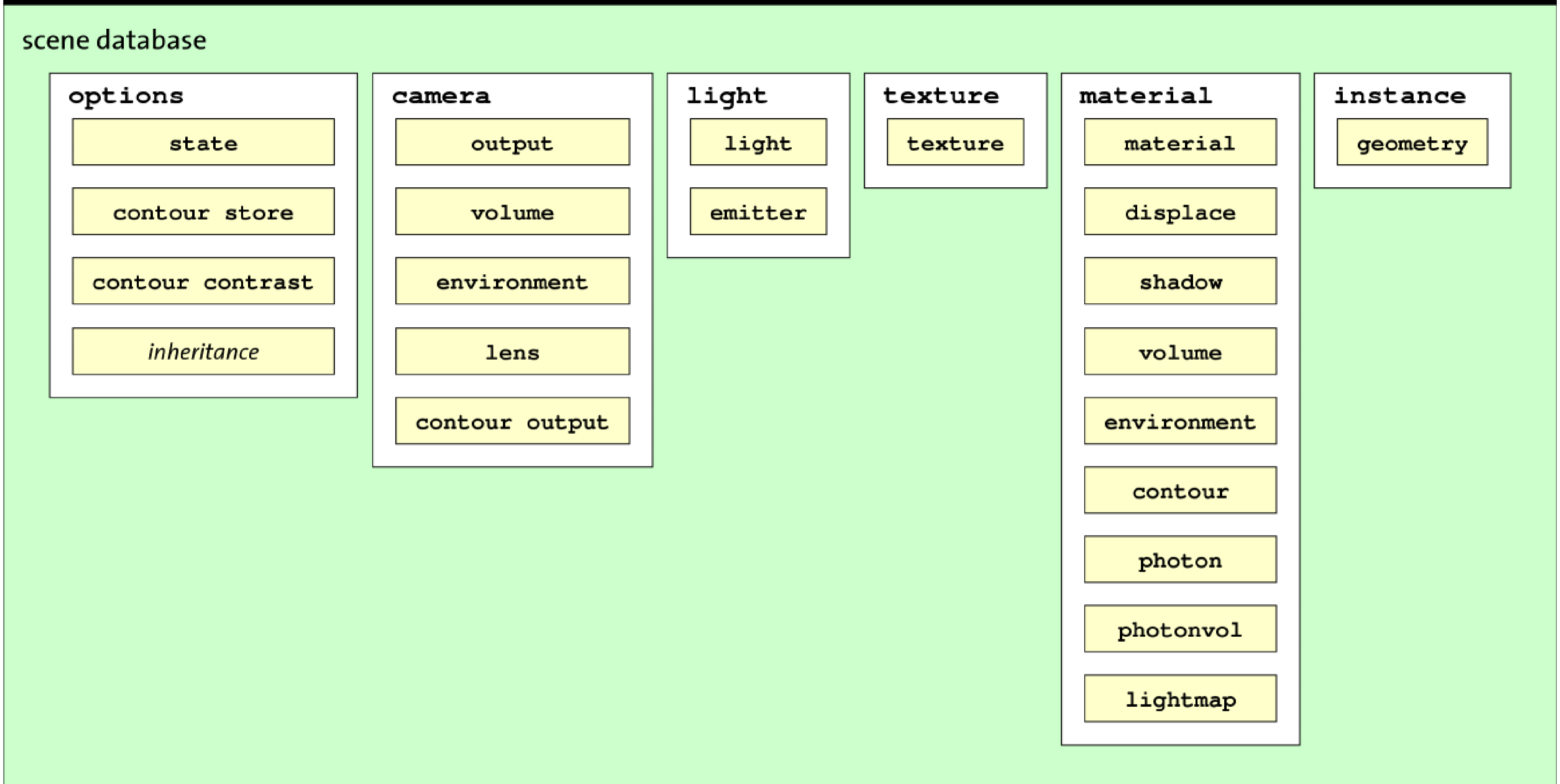
Perception


understanding of scene
isolation of object
identification of shape
experience with light
retinal image
image
space
shape
light
color
varnish
background relationships
surface detail
shading from light
color blocking
Perception
Painting


understanding of scene
isolation of object
identification of shape
experience with light
retinal image
image
space
shape
light
color
varnish
background relationships
surface detail
shading from light
color blocking
Perception
Shaders
Painting

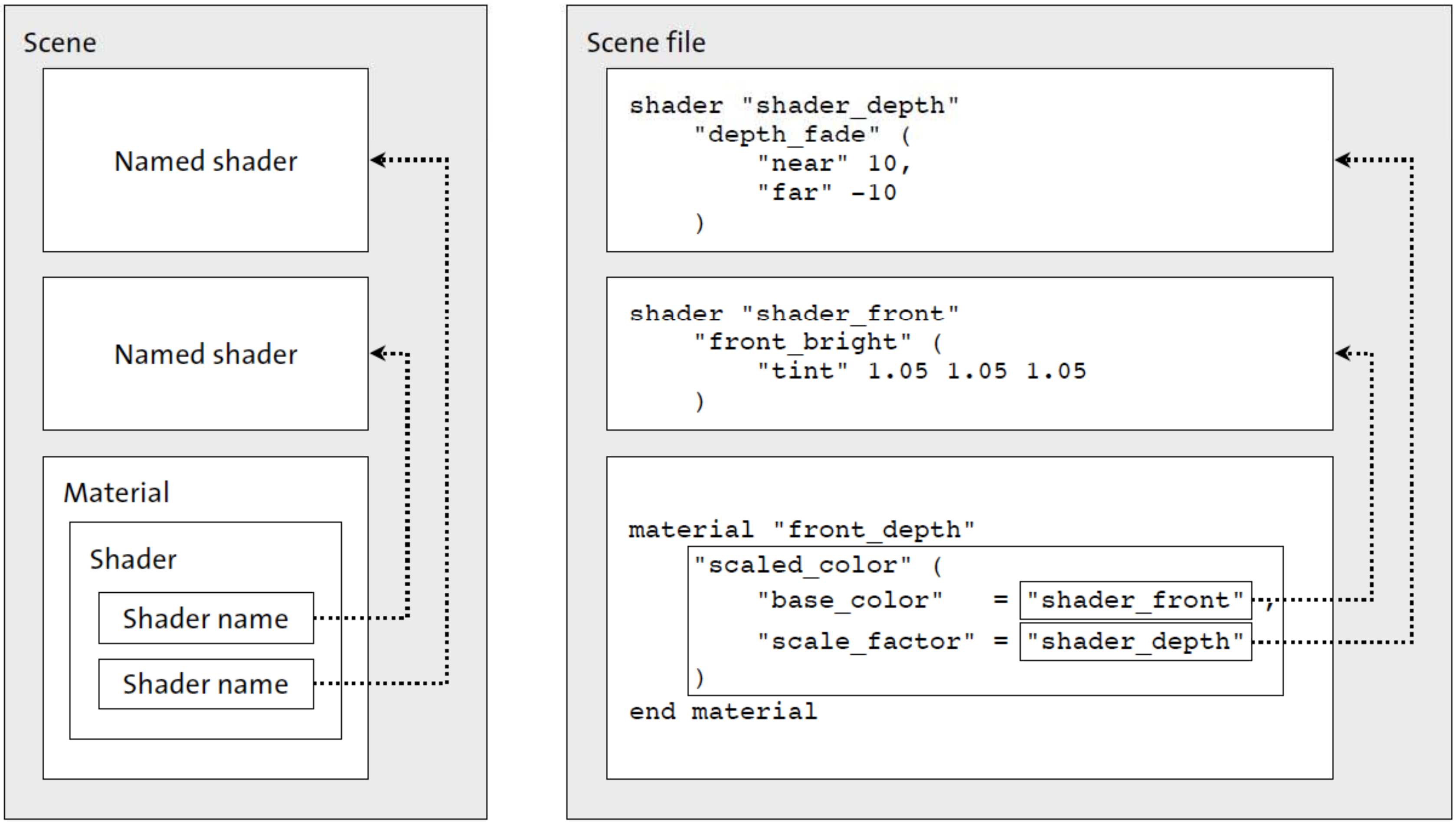


Shaders

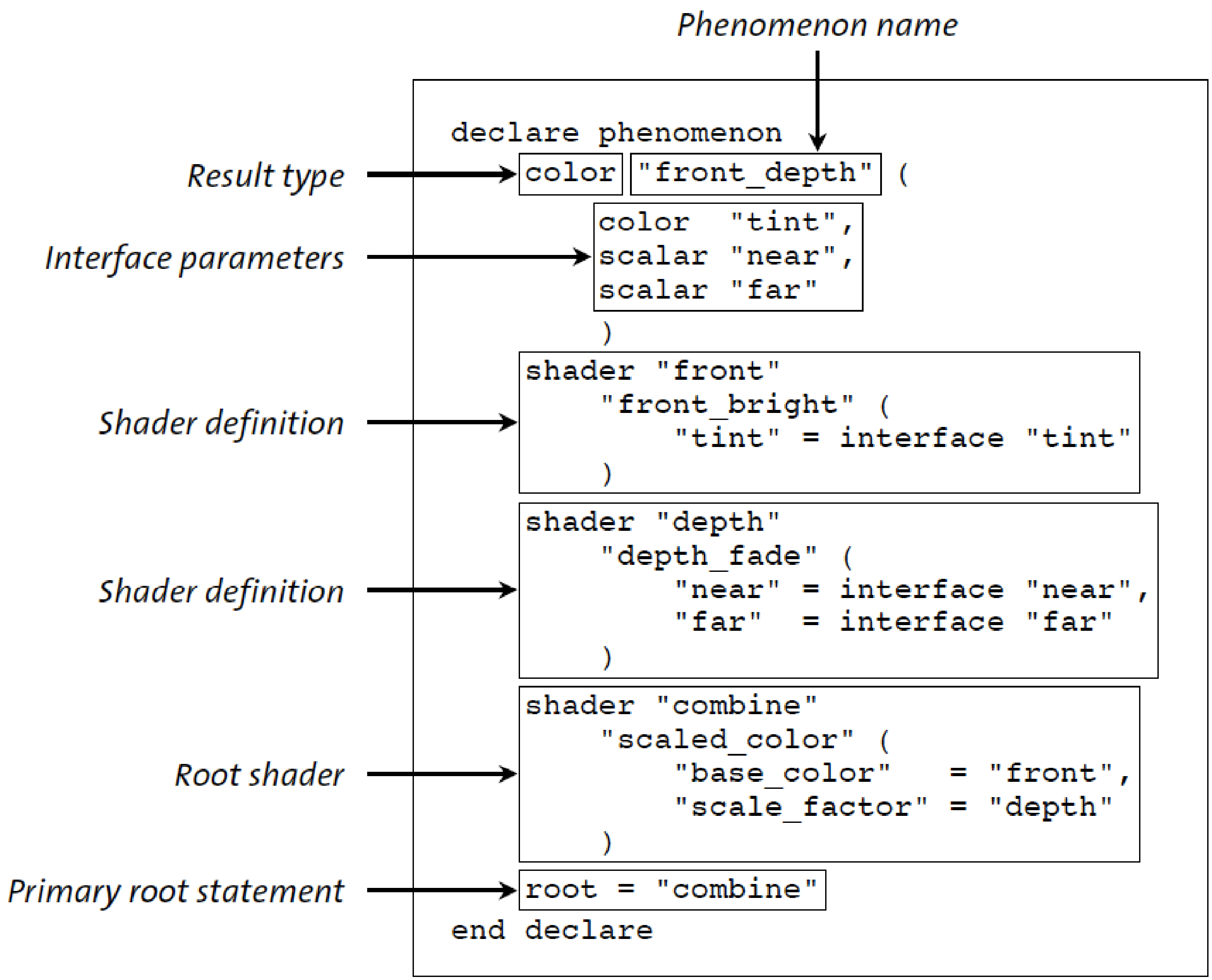


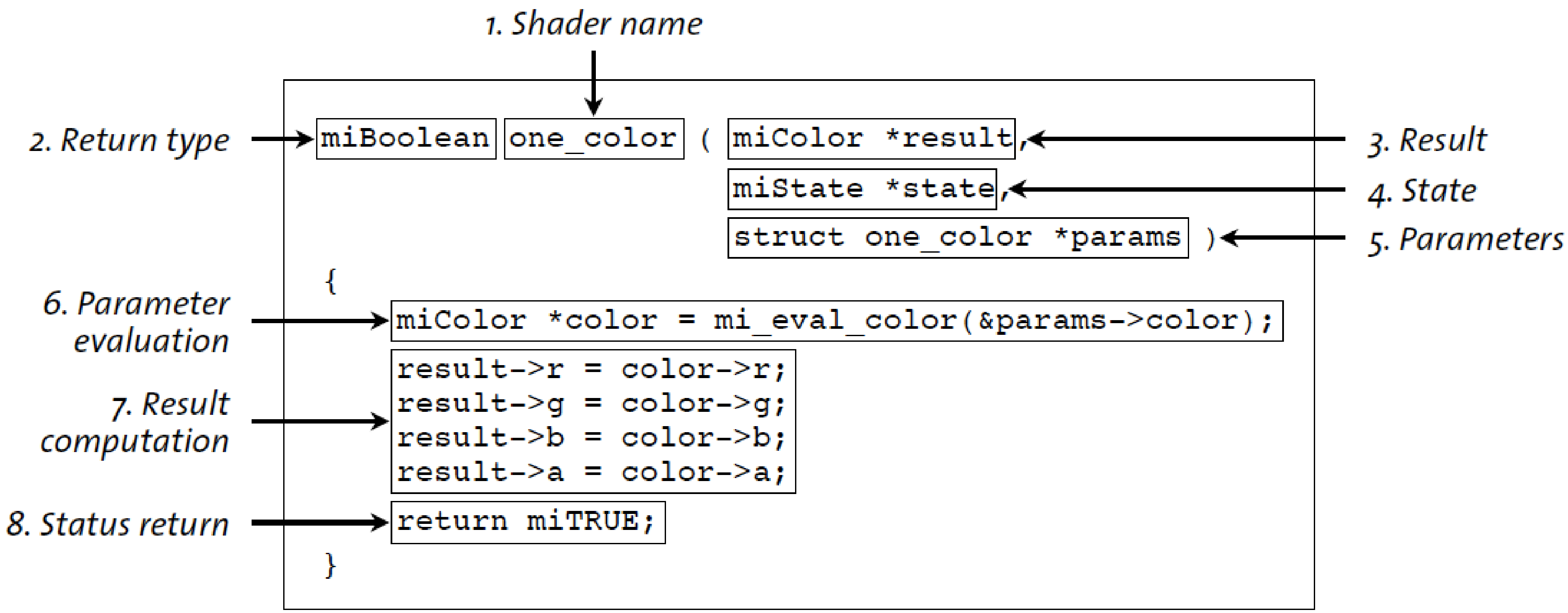
scene database
options
state
contour store
contour contrast
inheritance
camera
output
volume
environment
lens
contour output
light
light
emitter
texture
texture
material
material
displace
shadow
volume
environment
contour
photon
photonvol
lightmap
instance
geometry

Table of Contents



Scene
Named shader
Named shader
Material
Shader
Shader name
Shader name
Scene file
shader "shader_depth"
"depth_fade" (
"near" 10,
"far" -10
)
shader "shader_front"
"front_bright" (
"tint" 1.05 1.05 1.05
)
material "front_depth"
"scaled_color" (
"base_color" = "shader_front",
"scale_factor" = "shader_depth"
)
end material


Phenomenon name
Result type
Interface parameters
Shader definition
Shader definition
Root shader
Primary root statement
declare phenomenon
color "front_depth" (
color "tint",
scalar "near",
scalar "far"
)
shader "front"
"front_bright" (
"tint" = interface "tint"
)
shader "depth"
"depth_fade" (
"near" = interface "near",
"far" = interface "far"
)
shader "combine"
"scaled_color" (
"base_color" = "front",
"scale_factor" = "depth"
)
root = "combine"
end declare

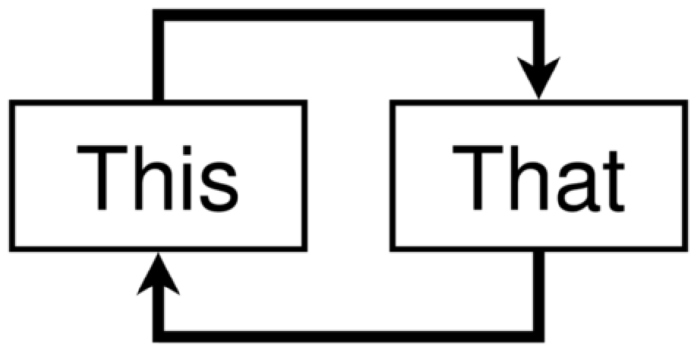

1. Shader name
2. Return type
miBoolean
one_color
(
miColor *result,
3. Result
miState *state,
4. State
struct one_color *params
)
5. Parameters
{
6. Parameter evaluation
miColor *color = mi_eval_color(&params->color);
7. Result computation
result->r = color->r;
result->g = color->g;
result->b = color->b;
result->a = color->a;
8. Status return
return miTRUE;
}

Cross-referencing in the shader book

3

Cross-referencing in the shader book

Marginal references point to the first two books.


This
That

```
instance "main-camera"
    "camera"
        transform
            1  0  0  0
            0  1  0  0
            0  0  1  0
            0  0 -9  1
end instance
```

Figure 2.11: A camera instance with an alternate text format

If you are writing an application that generates .mi scene files directly, you are free to format the scene file text in the manner that best expresses the structure implied by the application.

2.6 Grouping the elements in the scene

Rend 373, 14.2. Instance Groups

We've only defined a single instance, but there can be any number in the scene file. Instances can be organized into *hierarchies* of any complexity using *instance groups*. Before we begin to render the scene, we will need to specify the top-level group that will contain all the instances to be rendered. In our simple scene, we've only instanced a square, so the top-level instance group will only contain the square and camera instances. Defining the `instgroup` block follows the common pattern: a reserved word (`instgroup`), the name for the block, additional information appropriate for the type of block, and the final `end` statement.

```
instgroup "root"
    "main-camera" "yellow-square"
end instgroup
```

Figure 2.12: The root instance group for scene containing the elements to be used in rendering

Prog 173, 2.7.16. Instance Groups

To construct an object hierarchy, an `instgroup` block can contain other instance groups as well as other objects. In modeling applications, a hierarchy is often a natural way to represent large structures, and this organization in the application can be represented in the scene database with the `instgroup` element.

2.7 Scene file commands

Rend 36, 2.2. Anatomy of a Scene

All the previous examples showed the various elements that can be contained in the scene file. The scene file can also contain *commands* that do not define elements but tell mental ray to perform an action at the point in the scene file where the command occurs.

Prog 186, 3.1. Dynamic Linking of Shaders

For example, to use a shader in a scene file we need to do two things:

1. *Load* the compiled shader into memory when rendering begins; and

Rend 272, 11.1. Declarations

2. *Declare* the data types of the shader and its parameters so that its use later in the scene file can be correctly parsed.

Prog 79, 2.6.2. Shader Compilation and Linking
Prog 74, 2.6. Commands

In the scene file, shader loading and declaration are usually done using the `link` and `$include` commands, respectively. For example, to use the `one_color` shader in our simple scene, these commands are included at the beginning of the file:

```
instance "main-camera"
    "camera"
        transform
            1  0  0  0
            0  1  0  0
            0  0  1  0
            0  0 -9  1
end instance
```

Figure 2.11: A camera instance with an alternate text format

If you are writing an application that generates .mi scene files directly, you are free to format the scene file text in the manner that best expresses the structure implied by the application.

2.6 Grouping the elements in the scene

Rend 373, 14.2. Instance Groups

We've only defined a single instance, but there can be any number in the scene file. Instances can be organized into *hierarchies* of any complexity using *instance groups*. Before we begin to render the scene, we will need to specify the top-level group that will contain all the instances to be rendered. In our simple scene, we've only instanced a square, so the top-level instance group will only contain the square and camera instances. Defining the `instgroup` block follows the common pattern: a reserved word (`instgroup`), the name for the block, additional information appropriate for the type of block, and the final `end` statement.

```
instgroup "root"
    "main-camera" "yellow-square"
end instgroup
```

Figure 2.12: The root instance group for scene containing the elements to be used in rendering

Prog 173, 2.7.16. Instance Groups

To construct an object hierarchy, an `instgroup` block can contain other instance groups as well as other objects. In modeling applications, a hierarchy is often a natural way to represent large structures, and this organization in the application can be represented in the scene database with the `instgroup` element.

2.7 Scene file commands

Rend 36, 2.2. Anatomy of a Scene

All the previous examples showed the various elements that can be contained in the scene file. The scene file can also contain *commands* that do not define elements but tell mental ray to perform an action at the point in the scene file where the command occurs.

Prog 186, 3.1. Dynamic Linking of Shaders

For example, to use a shader in a scene file we need to do two things:

1. *Load* the compiled shader into memory when rendering begins; and

Rend 272, 11.1. Declarations

2. *Declare* the data types of the shader and its parameters so that its use later in the scene file can be correctly parsed.

Prog 79, 2.6.2. Shader Compilation and Linking
Prog 74, 2.6. Commands

In the scene file, shader loading and declaration are usually done using the `link` and `$include` commands, respectively. For example, to use the `one_color` shader in our simple scene, these commands are included at the beginning of the file:

2.7 Scene file commands

Rend 36, 2.2. Anatomy of a Scene

All the previous examples showed th
The scene file can also contain *comn*
perform an action at the point in the s

Prog 186, 3.1. Dynamic Linking of Shaders

For example, to use a shader in a scen

1. *Load* the compiled shader into r

Rend 272, 11.1. Declarations

2. *Declare* the data types of the sha
   can be correctly parsed.

Prog 79, 2.6.2. Shader Compilation and Linking
Prog 74, 2.6. Commands

In the scene file, shader loading and d
commands, respectively. For exampl
commands are included at the beginni

2.7 Scene file commands

Rend 36, 2.2. Anatomy of a Scene

All the previous examples showed th
The scene file can also contain *comm*
perform an action at the point in the s

Prog 186, 3.1. Dynamic Linking of Shaders

For example, to use a shader in a scen

1. *Load* the compiled shader into r

Rend 272, 11.1. Declarations

2. *Declare* the data types of the sha can be correctly parsed.

Prog 79, 2.6.2. Shader Compilation and Linking

Prog 74, 2.6. Commands

In the scene file, shader loading and d
commands, respectively. For exampl
commands are included at the beginni

2.7 Scene file commands

Rend 36, 2.2. Anatomy of a Scene

All the previous examples showed th
The scene file can also contain *comn*
perform an action at the point in the s

Prog 186, 3.1. Dynamic Linking of Shaders

For example, to use a shader in a scene

1. *Load* the compiled shader into r

Rend 272, 11.1. Declarations

2. *Declare* the data types of the sha
can be correctly parsed.

Prog 79, 2.6.2. Shader Compilation and Linking
Prog 74, 2.6. Commands

In the scene file, shader loading and d
commands, respectively. For exampl
commands are included at the beginni

Cross-referencing in the shader book

An index of topics in the first two books point to references in the new book.

References to Volumes I and II

Throughout this book, marginal references to the first two volumes of mental ray documentation, *Volume I: Rendering with mental ray* and *Volume II: Programming mental ray* point to further information about the current topic. This index lists the pages in this book that discuss the topics of Volumes I and II, sorted by section title.

References to Volumes I and II

Throughout this book, marginal references to the first two volumes of mental ray documentation, *Volume I: Rendering with mental ray* and *Volume II: Programming mental ray* point to further information about the current topic. This index lists the pages in this book that discuss the topics of Volumes I and II, sorted by section title.

[redacted]

Throughout this book, marginal references to the first two volumes of men
tion, *Volume I: Rendering with mental ray* and *Volume II: Programming*
further information about the current topic. This index lists the pages in th
the topics of Volumes I and II, sorted by section title.

Throughout this book, marginal references to the first two volumes of men
tion, *Volume I: Rendering with mental ray* and *Volume II: Programming*
further information about the current topic. This index lists the pages in th
the topics of Volumes I and II, sorted by section title.

Volume I: Rendering with mental ray

Throughout this book, marginal references to the first two volumes of men
tion, *Volume I: Rendering with mental ray* and *Volume II: Programming*
further information about the current topic. This index lists the pages in th
the topics of Volumes I and II, sorted by section title.

Cross-referencing in the shader book

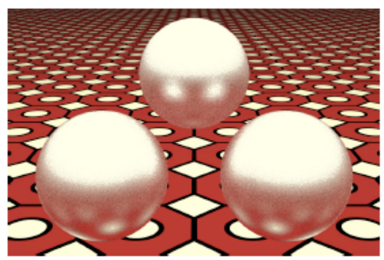
Shader code examples in the text point to the full source code listing in Appendix B.

```
1  struct glossy_reflection {
2      miScalar shiny;
3  };
4
5  miBoolean glossy_reflection (
6      miColor *result, miState *state, struct glossy_reflection *params )
7  {
8      miVector reflection_dir;
9      miScalar shiny = *mi_eval_scalar(&params->shiny);
10     mi_reflection_dir_glossy(&reflection_dir, state, shiny);
11
12     if (!mi_trace_reflection(result, state, &reflection_dir))
13         mi_trace_environment(result, state, &reflection_dir);
14
15     return miTRUE;
16 }
```

Figure 14.11: Shader source of `glossy_reflection` (p.513)

Once we've acquired the value of parameter `shiny` in **line 9**, the structure of main part of the shader in **lines 10–13** is the same as `specular_reflection`. In **line 10** we determine the glossy reflection direction. Or rather, we should say that we determine *one possible* glossy direction. The direction values determined by `mi_reflection_dir_glossy` are chosen somewhere within the cone determined by the `shiny` parameter. That "somewhere" is important; mental ray determines successive values of the glossy direction so that the rays are well distributed within the cone. (We'll talk more about this in the next section.)

As in the `specular_reflection` shader, if the reflected ray did not strike another object or the trace depth would be exceeded, we use the color of the environment in **line 13** for the result of our shader.



```
options "opt"
    object space
    samples 0 2
    contrast .1 .1 .1
    trace depth 5
end options

material "reflect"
    "glossy_reflection" (
        "shiny" 3 )
end material
```

Figure 14.12: Glossy reflection

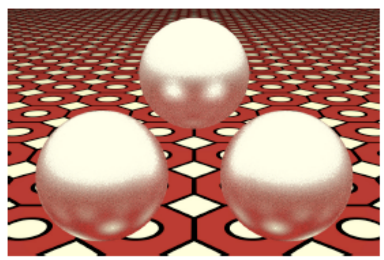
A single ray sent for each glossy reflection produces the grainy look in Figure 14.12. Our `contrast` value in the options block is `.1 .1 .1`. Will increasing the quality of the render by lowering the contrast value help with grainy look of our glossy reflection? We'll set the contrast to `.01 .01 .01` to find out.

```
1  struct glossy_reflection {
2      miScalar shiny;
3  };
4
5  miBoolean glossy_reflection (
6      miColor *result, miState *state, struct glossy_reflection *params )
7  {
8      miVector reflection_dir;
9      miScalar shiny = *mi_eval_scalar(&params->shiny);
10     mi_reflection_dir_glossy(&reflection_dir, state, shiny);
11
12     if (!mi_trace_reflection(result, state, &reflection_dir))
13         mi_trace_environment(result, state, &reflection_dir);
14
15     return miTRUE;
16 }
```

Figure 14.11: Shader source of `glossy_reflection` (p.513)

Once we've acquired the value of parameter `shiny` in **line 9**, the structure of main part of the shader in **lines 10–13** is the same as `specular_reflection`. In **line 10** we determine the glossy reflection direction. Or rather, we should say that we determine *one possible* glossy direction. The direction values determined by `mi_reflection_dir_glossy` are chosen somewhere within the cone determined by the `shiny` parameter. That "somewhere" is important; mental ray determines successive values of the glossy direction so that the rays are well distributed within the cone. (We'll talk more about this in the next section.)

As in the `specular_reflection` shader, if the reflected ray did not strike another object or the trace depth would be exceeded, we use the color of the environment in **line 13** for the result of our shader.



```
options "opt"
    object space
    samples 0 2
    contrast .1 .1 .1
    trace depth 5
end options

material "reflect"
    "glossy_reflection" (
        "shiny" 3 )
end material
```

Figure 14.12: Glossy reflection

A single ray sent for each glossy reflection produces the grainy look in Figure 14.12. Our `contrast` value in the options block is `.1 .1 .1`. Will increasing the quality of the render by lowering the contrast value help with grainy look of our glossy reflection? We'll set the contrast to `.01 .01 .01` to find out.

```
 1 struct glossy_reflection {
 2     miScalar shiny;
 3 };
 4
 5 miBoolean glossy_reflection (
 6     miColor *result, miState *state, struct glossy_reflection *params )
 7 {
 8     miVector reflection_dir;
 9     miScalar shiny = *mi_eval_scalar(&params->shiny);
10     mi_reflection_dir_glossy(&reflection_dir, state, shiny);
11
12     if (!mi_trace_reflection(result, state, &reflection_dir))
13         mi_trace_environment(result, state, &reflection_dir);
14
15     return miTRUE;
16 }
```

Figure 14.11: Shader source of `glossy_reflection` (p.513)

Once we've acquired the value of parameter `shiny` in **line 9**, the structure of main part of the shader in **lines 10–13** is the same as `specular_reflection`. In **line 10** we determine the glossy

```
1   struct glossy_reflection {
2       miScalar shiny;
3   };
4
5   miBoolean glossy_reflection (
6       miColor *result, miState *state, struct glossy_reflection *params )
7   {
8       miVector reflection_dir;
9       miScalar shiny = *mi_eval_scalar(&params->shiny);
10      mi_reflection_dir_glossy(&reflection_dir, state, shiny);
11
12      if (!mi_trace_reflection(result, state, &reflection_dir))
13          mi_trace_environment(result, state, &reflection_dir);
14
15      return miTRUE;
16  }
```

Figure 14.11: Shader source of glossy_reflection (p.513)

Once we've acquired the value of parameter shiny in **line 9**, the structure of main part of the shader in **lines 10–13** is the same as specular_reflection. In **line 10** we determine the glossy

specular_reflection

Page 184

```
declare shader
    color "specular_reflection" ()
    version 1
    apply material
end declare
```

```
#include "shader.h"

int specular_reflection_version(void) { return 1; }

miBoolean specular_reflection (
    miColor *result, miState *state, void *params  )
{
    miVector reflection_direction;
    mi_reflection_dir(&reflection_direction, state);

    if (!mi_trace_reflection(result, state, &reflection_direction))
        mi_trace_environment(result, state, &reflection_direction);

    return miTRUE;
}
```

glossy_reflection

Page 188

```
declare shader
    color "glossy_reflection" (
         scalar "shiny" default 5 )
    version 1
    apply material
end declare
```

```
#include "shader.h"

struct glossy_reflection {
    miScalar shiny;
};

int glossy_reflection_version(void) { return 1; }

miBoolean glossy_reflection (
    miColor *result, miState *state, struct glossy_reflection *params  )
{
    miVector reflection_dir;
    miScalar shiny = *mi_eval_scalar(&params->shiny);
    mi_reflection_dir_glossy(&reflection_dir, state, shiny);

    if (!mi_trace_reflection(result, state, &reflection_dir))
        mi_trace_environment(result, state, &reflection_dir);

    return miTRUE;
}
```

specular_reflection

Page 184

```
declare shader
    color "specular_reflection" ()
    version 1
    apply material
end declare
```

```
#include "shader.h"

int specular_reflection_version(void) { return 1; }

miBoolean specular_reflection (
    miColor *result, miState *state, void *params  )
{
    miVector reflection_direction;
    mi_reflection_dir(&reflection_direction, state);

    if (!mi_trace_reflection(result, state, &reflection_direction))
        mi_trace_environment(result, state, &reflection_direction);

    return miTRUE;
}
```

glossy_reflection

Page 188

```
declare shader
    color "glossy_reflection" (
         scalar "shiny" default 5 )
    version 1
    apply material
end declare
```

```
#include "shader.h"

struct glossy_reflection {
    miScalar shiny;
};

int glossy_reflection_version(void) { return 1; }

miBoolean glossy_reflection (
    miColor *result, miState *state, struct glossy_reflection *params  )
{
    miVector reflection_dir;
    miScalar shiny = *mi_eval_scalar(&params->shiny);
    mi_reflection_dir_glossy(&reflection_dir, state, shiny);

    if (!mi_trace_reflection(result, state, &reflection_dir))
        mi_trace_environment(result, state, &reflection_dir);

    return miTRUE;
}
```

specular_reflection

Page 184

```
declare shader
    color "specular_reflection" ()
    version 1
    apply material
end declare
```

```
#include "shader.h"

int specular_reflection_version(void) { return 1; }

miBoolean specular_reflection (
    miColor *result, miState *state, void *params  )
{
    miVector reflection_direction;
    mi_reflection_dir(&reflection_direction, state);

    if (!mi_trace_reflection(result, state, &reflection_direction))
        mi_trace_environment(result, state, &reflection_direction);

    return miTRUE;
}
```

glossy_reflection

Page 188

```
declare shader
    color "glossy_reflection" (
         scalar "shiny" default 5 )
    version 1
    apply material
end declare
```

```
#include "shader.h"

struct glossy_reflection {
    miScalar shiny;
};

int glossy_reflection_version(void) { return 1; }

miBoolean glossy_reflection (
    miColor *result, miState *state, struct glossy_reflection *params  )
{
    miVector reflection_dir;
    miScalar shiny = *mi_eval_scalar(&params->shiny);
    mi_reflection_dir_glossy(&reflection_dir, state, shiny);

    if (!mi_trace_reflection(result, state, &reflection_dir))
        mi_trace_environment(result, state, &reflection_dir);

    return miTRUE;
}
```

glossy_reflection

```
declare shader
    color "glossy_reflection" (
        scalar "shiny" default 5 )
    version 1
    apply material
end declare
```

```
#include "shader.h"

struct glossy_reflection {
    miScalar shiny;
};

int glossy_reflection_version(void) { return 1; }

miBoolean glossy_reflection (
    miColor *result, miState *state, struct glossy_reflection *params  )
{
    miVector reflection_dir;
    miScalar shiny = *mi_eval_scalar(&params->shiny);
    mi_reflection_dir_glossy(&reflection_dir, state, shiny);

    if (!mi_trace_reflection(result, state, &reflection_dir))
        mi_trace_environment(result, state, &reflection_dir);

    return miTRUE;
}
```

glossy_reflection

```
declare shader
    color "glossy_reflection" (
         scalar "shiny" default 5 )
    version 1
    apply material
end declare
```

```
#include "shader.h"
```

```
struct glossy_reflection {
    miScalar shiny;
};
```

```
int glossy_reflection_version(void) { return 1; }
```

```
miBoolean glossy_reflection (
    miColor *result, miState *state, struct glossy_reflection *params  )
{
    miVector reflection_dir;
    miScalar shiny = *mi_eval_scalar(&params->shiny);
    mi_reflection_dir_glossy(&reflection_dir, state, shiny);

    if (!mi_trace_reflection(result, state, &reflection_dir))
        mi_trace_environment(result, state, &reflection_dir);

    return miTRUE;
}
```

glossy_reflection


```
declare shader
    color "glossy_reflection" (
         scalar "shiny" default 5 )
    version 1
    apply material
end declare
```

```
#include "shader.h"

struct glossy_reflection {
    miScalar shiny;
};

int glossy_reflection_version(void) { return 1; }

miBoolean glossy_reflection (
    miColor *result, miState *state, struct glossy_reflection *params  )
{
    miVector reflection_dir;
    miScalar shiny = *mi_eval_scalar(&params->shiny);
    mi_reflection_dir_glossy(&reflection_dir, state, shiny);

    if (!mi_trace_reflection(result, state, &reflection_dir))
        mi_trace_environment(result, state, &reflection_dir);

    return miTRUE;
}
```

Cross-referencing in the shader book

Source code listings in Appendix B point back to the code's description in the text.

specular_reflection

Page 184

```
declare shader
    color "specular_reflection" ()
    version 1
    apply material
end declare
```

```
#include "shader.h"

int specular_reflection_version(void) { return 1; }

miBoolean specular_reflection (
    miColor *result, miState *state, void *params  )
{
    miVector reflection_direction;
    mi_reflection_dir(&reflection_direction, state);

    if (!mi_trace_reflection(result, state, &reflection_direction))
        mi_trace_environment(result, state, &reflection_direction);

    return miTRUE;
}
```

glossy_reflection

Page 188

```
declare shader
    color "glossy_reflection" (
        scalar "shiny" default 5 )
    version 1
    apply material
end declare
```

```
#include "shader.h"

struct glossy_reflection {
    miScalar shiny;
};

int glossy_reflection_version(void) { return 1; }

miBoolean glossy_reflection (
    miColor *result, miState *state, struct glossy_reflection *params  )
{
    miVector reflection_dir;
    miScalar shiny = *mi_eval_scalar(&params->shiny);
    mi_reflection_dir_glossy(&reflection_dir, state, shiny);

    if (!mi_trace_reflection(result, state, &reflection_dir))
        mi_trace_environment(result, state, &reflection_dir);

    return miTRUE;
}
```

specular_reflection

Page 184

```
declare shader
    color "specular_reflection" ()
    version 1
    apply material
end declare
```

```
#include "shader.h"

int specular_reflection_version(void) { return 1; }

miBoolean specular_reflection (
    miColor *result, miState *state, void *params  )
{
    miVector reflection_direction;
    mi_reflection_dir(&reflection_direction, state);

    if (!mi_trace_reflection(result, state, &reflection_direction))
        mi_trace_environment(result, state, &reflection_direction);

    return miTRUE;
}
```

glossy_reflection

Page 188

```
declare shader
    color "glossy_reflection" (
         scalar "shiny" default 5 )
    version 1
    apply material
end declare
```

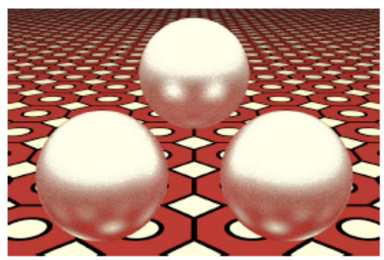
```
#include "shader.h"

struct glossy_reflection {
    miScalar shiny;
};

int glossy_reflection_version(void) { return 1; }

miBoolean glossy_reflection (
    miColor *result, miState *state, struct glossy_reflection *params  )
{
    miVector reflection_dir;
    miScalar shiny = *mi_eval_scalar(&params->shiny);
    mi_reflection_dir_glossy(&reflection_dir, state, shiny);

    if (!mi_trace_reflection(result, state, &reflection_dir))
        mi_trace_environment(result, state, &reflection_dir);

    return miTRUE;
}
```

```
1  struct glossy_reflection {
2      miScalar shiny;
3  };
4
5  miBoolean glossy_reflection (
6      miColor *result, miState *state, struct glossy_reflection *params )
7  {
8      miVector reflection_dir;
9      miScalar shiny = *mi_eval_scalar(&params->shiny);
10     mi_reflection_dir_glossy(&reflection_dir, state, shiny);
11
12     if (!mi_trace_reflection(result, state, &reflection_dir))
13         mi_trace_environment(result, state, &reflection_dir);
14
15     return miTRUE;
16 }
```

Figure 14.11: Shader source of `glossy_reflection` (p.513)

Once we've acquired the value of parameter `shiny` in **line 9**, the structure of main part of the shader in **lines 10–13** is the same as `specular_reflection`. In **line 10** we determine the glossy reflection direction. Or rather, we should say that we determine *one possible* glossy direction. The direction values determined by `mi_reflection_dir_glossy` are chosen somewhere within the cone determined by the `shiny` parameter. That "somewhere" is important; mental ray determines successive values of the glossy direction so that the rays are well distributed within the cone. (We'll talk more about this in the next section.)

As in the `specular_reflection` shader, if the reflected ray did not strike another object or the trace depth would be exceeded, we use the color of the environment in **line 13** for the result of our shader.



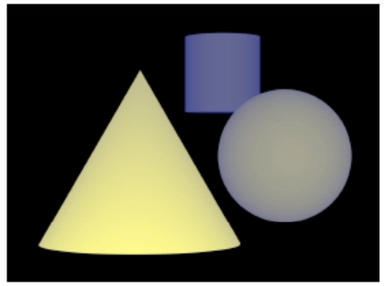
```
options "opt"
    object space
    samples 0 2
    contrast .1 .1 .1
    trace depth 5
end options

material "reflect"
    "glossy_reflection" (
        "shiny" 3 )
end material
```

Figure 14.12: Glossy reflection

A single ray sent for each glossy reflection produces the grainy look in Figure 14.12. Our `contrast` value in the options block is `.1 .1 .1`. Will increasing the quality of the render by lowering the contrast value help with grainy look of our glossy reflection? We'll set the contrast to `.01 .01 .01` to find out.

Cross-referencing in the shader book

Utility functions encapsulate lower-level details and make shader code clearer.



```
material "depth"
    "depth_fade_tint" (
        "near" 1.25,
        "near_color" 1 1 .5,
        "far" -1.15,
        "far_color" 0 0 .5 )
end material
```

Figure 7.7: Blending between two colors based on the *z* coordinate of a point on a surface

7.3 Clarifying the shader with auxiliary functions

Defining auxiliary functions can clarify the method being used in the shader. In this chapter we'll begin to develop a library of auxiliary functions that will make the strategies of the shaders clearer from their code. All of these functions will be named with a prefix of `miaux` ("mental images auxiliary," in the same spirit as the `mi_*` functions in the mental ray library), and will be consolidated in a library.

Prog 307, 3.26. Functions for Shaders

First we'll define a function to rescale a value from one range to another, so that the proportional relationships of the new values match those of the original:

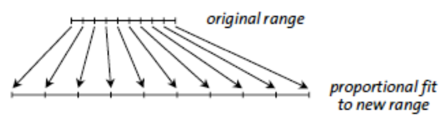


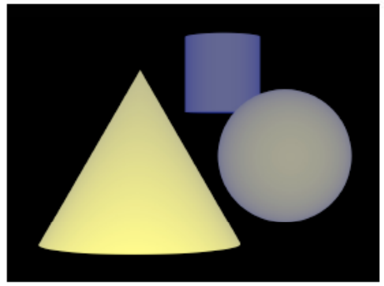
Figure 7.8: Converting from one scale to another

This basic function will be useful when we want define a relationship between scales of arbitrary type, like the mapping from the `near` and `far` parameter values to a normalized range of 0.0 to 1.0.

```
double miaux_fit(
    double v, double oldmin, double oldmax, double newmin, double newmax)
{
    return newmin + ((v - oldmin) / (oldmax - oldmin)) * (newmax - newmin);
}
```

Figure 7.9: Function `miaux_fit`

Since we're blending two colors based on a weighting factor, we'll define a function to perform this color blending:



```
material "depth"
    "depth_fade_tint" (
        "near" 1.25,
        "near_color" 1 1 .5,
        "far" -1.15,
        "far_color" 0 0 .5 )
end material
```

Figure 7.7: Blending between two colors based on the *z* coordinate of a point on a surface

7.3 Clarifying the shader with auxiliary functions

Defining auxiliary functions can clarify the method being used in the shader. In this chapter we'll begin to develop a library of auxiliary functions that will make the strategies of the shaders clearer from their code. All of these functions will be named with a prefix of `miaux` ("mental images auxiliary," in the same spirit as the `mi_*` functions in the mental ray library), and will be consolidated in a library.

Prog 307, 3.26. Functions for Shaders

First we'll define a function to rescale a value from one range to another, so that the proportional relationships of the new values match those of the original:

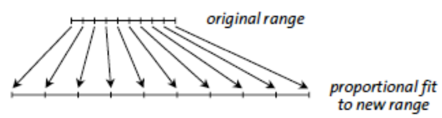


Figure 7.8: Converting from one scale to another

This basic function will be useful when we want define a relationship between scales of arbitrary type, like the mapping from the `near` and `far` parameter values to a normalized range of 0.0 to 1.0.

```
double miaux_fit(
    double v, double oldmin, double oldmax, double newmin, double newmax)
{
    return newmin + ((v - oldmin) / (oldmax - oldmin)) * (newmax - newmin);
}
```

Figure 7.9: Function `miaux_fit`

Since we're blending two colors based on a weighting factor, we'll define a function to perform this color blending:

First we'll define a function to rescale a value from one range to another, so that the proportional relationships of the new values match those of the original:

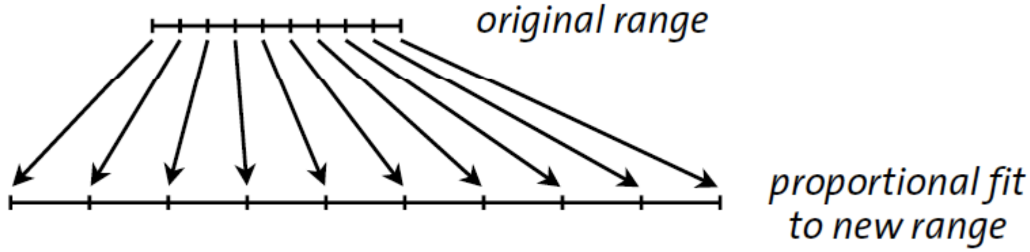


Figure 7.8: Converting from one scale to another

This basic function will be useful when we want define a relationship between scales of arbitrary type, like the mapping from the `near` and `far` parameter values to a normalized range of 0.0 to 1.0.

```
double miaux_fit(
    double v, double oldmin, double oldmax, double newmin, double newmax)
{
    return newmin + ((v - oldmin) / (oldmax - oldmin)) * (newmax - newmin);
}
```

Figure 7.9: Function `miaux_fit`

First we'll define a function to rescale a value from one range to another, so that the proportional relationships of the new values match those of the original:

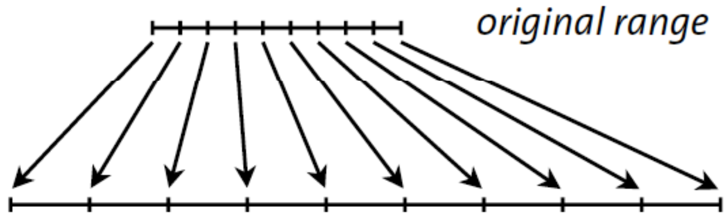


Figure 7.8: Converting from one scale to another

This basic function will be useful when we want define a relationship between scales of arbitrary type, like the mapping from the `near` and `far` parameter values to a normalized range of 0.0 to 1.0.

```
double miaux_fit(
    double v, double oldmin, double oldmax, double newmin, double newmax)
{
    return newmin + ((v - oldmin) / (oldmax - oldmin)) * (newmax - newmin);
}
```

Figure 7.9: Function `miaux_fit`

```
    p->v4            = *mi_eval_vector(&params->v4);
    p->approximation = *mi_eval_integer(&params->approximation);
    p->degree        = *mi_eval_integer(&params->degree);

    miaux_define_hair_object(
        p->name, hair_geo_4v_bbox, p, result, hair_geo_4v_callback);

    return miTRUE;
}

miBoolean hair_geo_4v_callback(miTag tag, void *params)
{
    miHair_list *hair_list;
    miScalar    *hair_scalars;
    miGeoIndex  *hair_indices;
    hair_geo_4v_t *p = (hair_geo_4v_t *)params;
    int hair_count = 1, hair_scalar_count = 4 * 3 + 1;

    mi_api_incremental(miTRUE);
    miaux_define_hair_object(p->name, hair_geo_4v_bbox, p, NULL, NULL);
    hair_list = mi_api_hair_begin();
    hair_list->approx = p->approximation;
    hair_list->degree = p->degree;
    mi_api_hair_info(0, 'r', 1);

    hair_scalars = mi_api_hair_scalars_begin(hair_scalar_count);
    *hair_scalars++ = p->radius;
    miaux_append_hair_vertex(&hair_scalars, &p->v1);
    miaux_append_hair_vertex(&hair_scalars, &p->v2);
    miaux_append_hair_vertex(&hair_scalars, &p->v3);
    miaux_append_hair_vertex(&hair_scalars, &p->v4);
    mi_api_hair_scalars_end(hair_scalar_count);

    hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
    hair_indices[0] = 0;
    hair_indices[1] = hair_scalar_count;
    mi_api_hair_hairs_end();

    mi_api_hair_end();
    mi_api_object_end();

    return miTRUE;
}
```

Function	*Page*
miaux_init_bbox	598
miaux_adjust_bbox	598
miaux_set_vector	590
miaux_describe_bbox	598
miaux_define_hair_object	598
miaux_tag_to_string	590
miaux_append_hair_vertex	598

```
    p->v4            = *mi_eval_vector(&params->v4);
    p->approximation = *mi_eval_integer(&params->approximation);
    p->degree        = *mi_eval_integer(&params->degree);

    miaux_define_hair_object(
        p->name, hair_geo_4v_bbox, p, result, hair_geo_4v_callback);

    return miTRUE;
}

miBoolean hair_geo_4v_callback(miTag tag, void *params)
{
    miHair_list *hair_list;
    miScalar    *hair_scalars;
    miGeoIndex  *hair_indices;
    hair_geo_4v_t *p = (hair_geo_4v_t *)params;
    int hair_count = 1, hair_scalar_count = 4 * 3 + 1;

    mi_api_incremental(miTRUE);
    miaux_define_hair_object(p->name, hair_geo_4v_bbox, p, NULL, NULL);
    hair_list = mi_api_hair_begin();
    hair_list->approx = p->approximation;
    hair_list->degree = p->degree;
    mi_api_hair_info(0, 'r', 1);

    hair_scalars = mi_api_hair_scalars_begin(hair_scalar_count);
    *hair_scalars++ = p->radius;
    miaux_append_hair_vertex(&hair_scalars, &p->v1);
    miaux_append_hair_vertex(&hair_scalars, &p->v2);
    miaux_append_hair_vertex(&hair_scalars, &p->v3);
    miaux_append_hair_vertex(&hair_scalars, &p->v4);
    mi_api_hair_scalars_end(hair_scalar_count);

    hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
    hair_indices[0] = 0;
    hair_indices[1] = hair_scalar_count;
    mi_api_hair_hairs_end();

    mi_api_hair_end();
    mi_api_object_end();

    return miTRUE;
}
```


Function	*Page*
miaux_init_bbox	598
miaux_adjust_bbox	598
miaux_set_vector	590
miaux_describe_bbox	598
miaux_define_hair_object	598
miaux_tag_to_string	590
miaux_append_hair_vertex	598

```
miBoolean hair_geo_4v_callback(miTag tag, void *params)
{
    miHair_list *hair_list;
    miScalar    *hair_scalars;
    miGeoIndex  *hair_indices;
    hair_geo_4v_t *p = (hair_geo_4v_t *)params;
    int hair_count = 1, hair_scalar_count = 4 * 3 + 1;

    mi_api_incremental(miTRUE);
    miaux_define_hair_object(p->name, hair_geo_4v_bbox, p, NULL, NULL);
    hair_list = mi_api_hair_begin();
    hair_list->approx = p->approximation;
    hair_list->degree = p->degree;
    mi_api_hair_info(0, 'r', 1);

    hair_scalars = mi_api_hair_scalars_begin(hair_scalar_count);
    *hair_scalars++ = p->radius;
    miaux_append_hair_vertex(&hair_scalars, &p->v1);
    miaux_append_hair_vertex(&hair_scalars, &p->v2);
    miaux_append_hair_vertex(&hair_scalars, &p->v3);
    miaux_append_hair_vertex(&hair_scalars, &p->v4);
    mi_api_hair_scalars_end(hair_scalar_count);

    hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
    hair_indices[0] = 0;
    hair_indices[1] = hair_scalar_count;
    mi_api_hair_hairs_end();

    mi_api_hair_end();
    mi_api_object_end();

    return miTRUE;
}
```

```
miBoolean hair_geo_4v_callback(miTag tag, void *params)
{
    miHair_list *hair_list;
    miScalar    *hair_scalars;
    miGeoIndex  *hair_indices;
    hair_geo_4v_t *p = (hair_geo_4v_t *)params;
    int hair_count = 1, hair_scalar_count = 4 * 3 + 1;

    mi_api_incremental(miTRUE);
    miaux_define_hair_object(p->name, hair_geo_4v_bbox, p, NULL, NULL);
    hair_list = mi_api_hair_begin();
    hair_list->approx = p->approximation;
    hair_list->degree = p->degree;
    mi_api_hair_info(0, 'r', 1);

    hair_scalars = mi_api_hair_scalars_begin(hair_scalar_count);
    *hair_scalars++ = p->radius;
    miaux_append_hair_vertex(&hair_scalars, &p->v1);
    miaux_append_hair_vertex(&hair_scalars, &p->v2);
    miaux_append_hair_vertex(&hair_scalars, &p->v3);
    miaux_append_hair_vertex(&hair_scalars, &p->v4);
    mi_api_hair_scalars_end(hair_scalar_count);

    hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
    hair_indices[0] = 0;
    hair_indices[1] = hair_scalar_count;
    mi_api_hair_hairs_end();

    mi_api_hair_end();
    mi_api_object_end();

    return miTRUE;
}
```

```
    p->v4            = *mi_eval_vector(&params->v4);
    p->approximation = *mi_eval_integer(&params->approximation);
    p->degree        = *mi_eval_integer(&params->degree);

    miaux_define_hair_object(
        p->name, hair_geo_4v_bbox, p, result, hair_geo_4v_callback);

    return miTRUE;
}

miBoolean hair_geo_4v_callback(miTag tag, void *params)
{
    miHair_list *hair_list;
    miScalar    *hair_scalars;
    miGeoIndex  *hair_indices;
    hair_geo_4v_t *p = (hair_geo_4v_t *)params;
    int hair_count = 1, hair_scalar_count = 4 * 3 + 1;

    mi_api_incremental(miTRUE);
    miaux_define_hair_object(p->name, hair_geo_4v_bbox, p, NULL, NULL);
    hair_list = mi_api_hair_begin();
    hair_list->approx = p->approximation;
    hair_list->degree = p->degree;
    mi_api_hair_info(0, 'r', 1);

    hair_scalars = mi_api_hair_scalars_begin(hair_scalar_count);
    *hair_scalars++ = p->radius;
    miaux_append_hair_vertex(&hair_scalars, &p->v1);
    miaux_append_hair_vertex(&hair_scalars, &p->v2);
    miaux_append_hair_vertex(&hair_scalars, &p->v3);
    miaux_append_hair_vertex(&hair_scalars, &p->v4);
    mi_api_hair_scalars_end(hair_scalar_count);

    hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
    hair_indices[0] = 0;
    hair_indices[1] = hair_scalar_count;
    mi_api_hair_hairs_end();

    mi_api_hair_end();
    mi_api_object_end();

    return miTRUE;
}
```

Function	*Page*
miaux_init_bbox	598
miaux_adjust_bbox	598
miaux_set_vector	590
miaux_describe_bbox	598
miaux_define_hair_object	598
miaux_tag_to_string	590
miaux_append_hair_vertex	598

```
    p->v4            = *mi_eval_vector(&params->v4);
    p->approximation = *mi_eval_integer(&params->approximation);
    p->degree        = *mi_eval_integer(&params->degree);

    miaux_define_hair_object(
        p->name, hair_geo_4v_bbox, p, result, hair_geo_4v_callback);

    return miTRUE;
}

miBoolean hair_geo_4v_callback(miTag tag, void *params)
{
    miHair_list *hair_list;
    miScalar    *hair_scalars;
    miGeoIndex  *hair_indices;
    hair_geo_4v_t *p = (hair_geo_4v_t *)params;
    int hair_count = 1, hair_scalar_count = 4 * 3 + 1;

    mi_api_incremental(miTRUE);
    miaux_define_hair_object(p->name, hair_geo_4v_bbox, p, NULL, NULL);
    hair_list = mi_api_hair_begin();
    hair_list->approx = p->approximation;
    hair_list->degree = p->degree;
    mi_api_hair_info(0, 'r', 1);

    hair_scalars = mi_api_hair_scalars_begin(hair_scalar_count);
    *hair_scalars++ = p->radius;
    miaux_append_hair_vertex(&hair_scalars, &p->v1);
    miaux_append_hair_vertex(&hair_scalars, &p->v2);
    miaux_append_hair_vertex(&hair_scalars, &p->v3);
    miaux_append_hair_vertex(&hair_scalars, &p->v4);
    mi_api_hair_scalars_end(hair_scalar_count);

    hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
    hair_indices[0] = 0;
    hair_indices[1] = hair_scalar_count;
    mi_api_hair_hairs_end();

    mi_api_hair_end();
    mi_api_object_end();

    return miTRUE;
}
```

Function	*Page*
miaux_init_bbox	598
miaux_adjust_bbox	598
miaux_set_vector	590
miaux_describe_bbox	598
miaux_define_hair_object	598
miaux_tag_to_string	590
miaux_append_hair_vertex	598

```
    hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
    hair_indices[0] = 0;
    hair_indices[1] = hair_scalar_count;
    mi_api_hair_hairs_end();

    mi_api_hair_end();
    mi_api_object_end();

    return miTRUE;
}
```

Function	*Page*
miaux_init_bbox	598
miaux_adjust_bbox	598
miaux_set_vector	590
miaux_describe_bbox	598
miaux_define_hair_object	598
miaux_tag_to_string	590
miaux_append_hair_vertex	598

```
    hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
    hair_indices[0] = 0;
    hair_indices[1] = hair_scalar_count;
    mi_api_hair_hairs_end();

    mi_api_hair_end();
    mi_api_object_end();

    return miTRUE;
}
```

Function	*Page*
miaux_init_bbox	598
miaux_adjust_bbox	598
miaux_set_vector	590
miaux_describe_bbox	598
miaux_define_hair_object	598
miaux_tag_to_string	590
miaux_append_hair_vertex	598

```
    return mi_api_object_end();
}
```

Chapter 20 – Modeling hair

```
void miaux_define_hair_object(
    miTag name_tag, miaux_bbox_function bbox_function, void *params,
    miTag *geoshader_result, miApi_object_callback callback)
{
    miTag tag;
    miObject *obj;
    char *name = miaux_tag_to_string(name_tag, "::hair");
    obj = mi_api_object_begin(mi_mem_strdup(name));
    obj->visible = miTRUE;
    obj->shadow = obj->reflection = obj->refraction = 3;
    bbox_function(obj, params);
    if (geoshader_result != NULL && callback != NULL) {
        mi_api_object_callback(callback, params);
        tag = mi_api_object_end();
        mi_geoshader_add_result(geoshader_result, tag);
        obj = (miObject *)mi_scene_edit(tag);
        obj->geo.placeholder_list.type = miOBJECT_HAIR;
        mi_scene_edit_end(tag);
    }
}

void miaux_describe_bbox(miObject *obj)
{
    mi_progress("Object bbox: %f,%f,%f  %f,%f,%f",
                obj->bbox_min.x, obj->bbox_min.y, obj->bbox_min.z,
                obj->bbox_max.x, obj->bbox_max.y, obj->bbox_max.z);
}

void miaux_adjust_bbox(miObject *obj, miVector *v, miScalar extra)
{
    miVector v_extra, vmin, vmax;
    miaux_set_vector(&v_extra, extra, extra, extra);
    mi_vector_sub(&vmin, v, &v_extra);
    mi_vector_add(&vmax, v, &v_extra);
    mi_vector_min(&obj->bbox_min, &obj->bbox_min, &vmin);
    mi_vector_max(&obj->bbox_max, &obj->bbox_max, &vmax);
}

void miaux_init_bbox(miObject *obj)
{
    obj->bbox_min.x = miHUGE_SCALAR;
    obj->bbox_min.y = miHUGE_SCALAR;
    obj->bbox_min.z = miHUGE_SCALAR;
    obj->bbox_max.x = -miHUGE_SCALAR;
    obj->bbox_max.y = -miHUGE_SCALAR;
    obj->bbox_max.z = -miHUGE_SCALAR;
}

void miaux_append_hair_vertex(miScalar **scalar_array, miVector *v)
{
    (*scalar_array)[0] = v->x;
    (*scalar_array)[1] = v->y;
    (*scalar_array)[2] = v->z;
    *scalar_array += 3;
}

void miaux_append_hair_data(
    miScalar **scalar_array, miVector *v, miScalar position,
    miScalar root_radius, miColor *root, miScalar tip_radius, miColor *tip )
{
    (*scalar_array)[0] = v->x;
    (*scalar_array)[1] = v->y;
```

```
    return mi_api_object_end();
}
```

Chapter 20 – Modeling hair

```
void miaux_define_hair_object(
    miTag name_tag, miaux_bbox_function bbox_function, void *params,
    miTag *geoshader_result, miApi_object_callback callback)
{
    miTag tag;
    miObject *obj;
    char *name = miaux_tag_to_string(name_tag, "::hair");
    obj = mi_api_object_begin(mi_mem_strdup(name));
    obj->visible = miTRUE;
    obj->shadow = obj->reflection = obj->refraction = 3;
    bbox_function(obj, params);
    if (geoshader_result != NULL && callback != NULL) {
        mi_api_object_callback(callback, params);
        tag = mi_api_object_end();
        mi_geoshader_add_result(geoshader_result, tag);
        obj = (miObject *)mi_scene_edit(tag);
        obj->geo.placeholder_list.type = miOBJECT_HAIR;
        mi_scene_edit_end(tag);
    }
}

void miaux_describe_bbox(miObject *obj)
{
    mi_progress("Object bbox: %f,%f,%f  %f,%f,%f",
                obj->bbox_min.x, obj->bbox_min.y, obj->bbox_min.z,
                obj->bbox_max.x, obj->bbox_max.y, obj->bbox_max.z);
}

void miaux_adjust_bbox(miObject *obj, miVector *v, miScalar extra)
{
    miVector v_extra, vmin, vmax;
    miaux_set_vector(&v_extra, extra, extra, extra);
    mi_vector_sub(&vmin, v, &v_extra);
    mi_vector_add(&vmax, v, &v_extra);
    mi_vector_min(&obj->bbox_min, &obj->bbox_min, &vmin);
    mi_vector_max(&obj->bbox_max, &obj->bbox_max, &vmax);
}

void miaux_init_bbox(miObject *obj)
{
    obj->bbox_min.x = miHUGE_SCALAR;
    obj->bbox_min.y = miHUGE_SCALAR;
    obj->bbox_min.z = miHUGE_SCALAR;
    obj->bbox_max.x = -miHUGE_SCALAR;
    obj->bbox_max.y = -miHUGE_SCALAR;
    obj->bbox_max.z = -miHUGE_SCALAR;
}

void miaux_append_hair_vertex(miScalar **scalar_array, miVector *v)
{
    (*scalar_array)[0] = v->x;
    (*scalar_array)[1] = v->y;
    (*scalar_array)[2] = v->z;
    *scalar_array += 3;
}

void miaux_append_hair_data(
    miScalar **scalar_array, miVector *v, miScalar position,
    miScalar root_radius, miColor *root, miScalar tip_radius, miColor *tip )
{
    (*scalar_array)[0] = v->x;
    (*scalar_array)[1] = v->y;
```

```
void miaux_adjust_bbox(miObject *obj, miVector *v, miScalar extra)
{
    miVector v_extra, vmin, vmax;
    miaux_set_vector(&v_extra, extra, extra, extra);
    mi_vector_sub(&vmin, v, &v_extra);
    mi_vector_add(&vmax, v, &v_extra);
    mi_vector_min(&obj->bbox_min, &obj->bbox_min, &vmin);
    mi_vector_max(&obj->bbox_max, &obj->bbox_max, &vmax);
}

void miaux_init_bbox(miObject *obj)
{
    obj->bbox_min.x = miHUGE_SCALAR;
    obj->bbox_min.y = miHUGE_SCALAR;
    obj->bbox_min.z = miHUGE_SCALAR;
    obj->bbox_max.x = -miHUGE_SCALAR;
    obj->bbox_max.y = -miHUGE_SCALAR;
    obj->bbox_max.z = -miHUGE_SCALAR;
}
```

Cross-referencing in the shader book

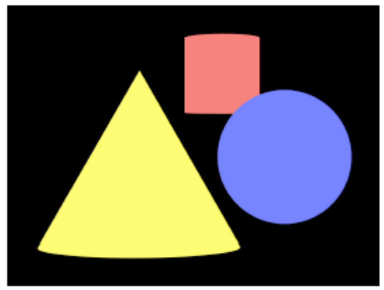
Example renderings are displayed with the relevant portion of the scene file that produced them.

Store contour shader returns multiple values. (See Figure 10.7.)

5.4 Using a shader in a material

Prog 114, 2.7.4. Materials

To use shader `one_color` to render a scene, we supply values for its parameters and include in it a material. In Figure 5.8, shader `one_color` is included anonymously in three different materials for the three object instances.



```
material "yellow"
    "one_color" (
        "color" 1 1 .4 )
end material

material "blue"
    "one_color" (
        "color" .4 .4 1 )
end material

material "red"
    "one_color" (
        "color" 1 .4 .4 )
end material
```

Figure 5.8: A single color for the entire object defined by shader `one_color` in the material

Throughout the book, we'll be rendering images like Figure 5.8 using the shaders developed in each chapter. Accompanying the image will be a fragment of the scene file that describes how the shader is used, or includes other parts of the scene file that will affect the final image. All the rendered images are collected in Appendix A beginning on page 457 and serve as a visual index to the various techniques we'll develop.

5.5 Shader programming style

To simplify the structural diagram of shader `one_color`, the three arguments to the shader function were placed on different lines. To shorten the source code examples throughout the book, most shaders will be shown with their arguments all on the same line.

```
miBoolean one_color (
    miColor *result, miState *state, struct one_color *params )
{
    miColor *color = mi_eval_color(&params->color);
    result->r = color->r;
    result->g = color->g;
    result->b = color->b;
    result->a = color->a;
    return miTRUE;
}
```

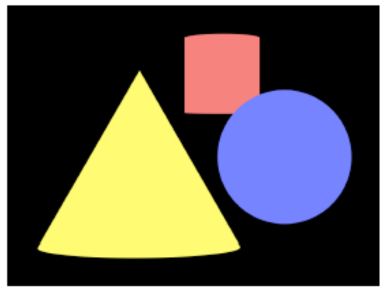
Figure 5.9: Putting the standard shader arguments on a single line

But we can go further than just rearranging the code to shorten it. The size of a variable of type `miColor` is known by the compiler in advance (a C struct containing a field for each of the red, Store contour shader returns multiple values. (See Figure 10.7.)

5.4 Using a shader in a material

Prog 114, 2.7.4. Materials

To use shader one_color to render a scene, we supply values for its parameters and include in it a material. In Figure 5.8, shader one_color is included anonymously in three different materials for the three object instances.



```
material "yellow"
    "one_color" (
        "color" 1 1 .4 )
end material

material "blue"
    "one_color" (
        "color" .4 .4 1 )
end material

material "red"
    "one_color" (
        "color" 1 .4 .4 )
end material
```

Figure 5.8: A single color for the entire object defined by shader one_color in the material

Throughout the book, we'll be rendering images like Figure 5.8 using the shaders developed in each chapter. Accompanying the image will be a fragment of the scene file that describes how the shader is used, or includes other parts of the scene file that will affect the final image. All the rendered images are collected in Appendix A beginning on page 457 and serve as a visual index to the various techniques we'll develop.

5.5 Shader programming style

To simplify the structural diagram of shader one_color, the three arguments to the shader function were placed on different lines. To shorten the source code examples throughout the book, most shaders will be shown with their arguments all on the same line.

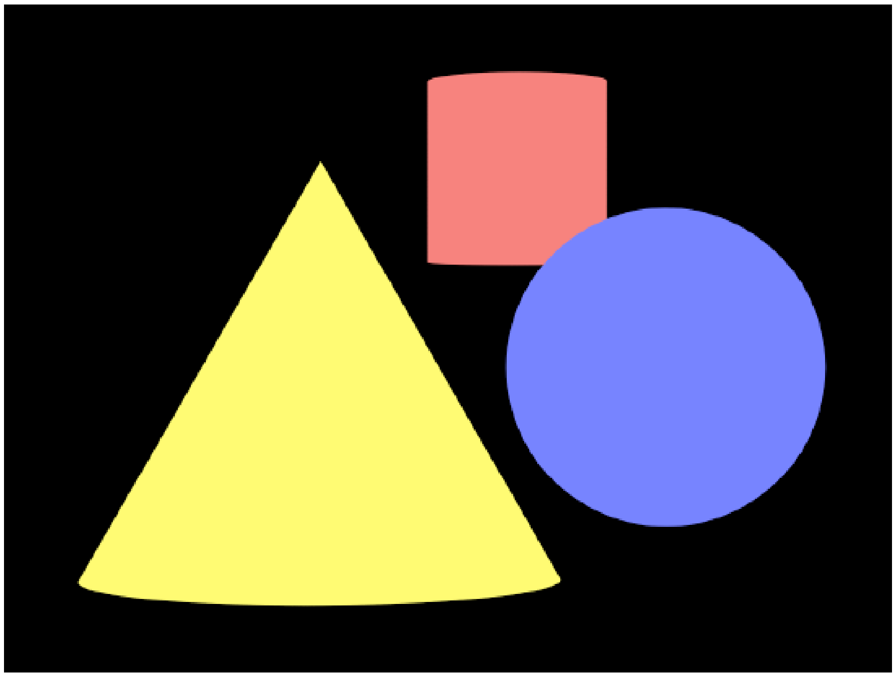
```
miBoolean one_color (
    miColor *result, miState *state, struct one_color *params )
{
    miColor *color = mi_eval_color(&params->color);
    result->r = color->r;
    result->g = color->g;
    result->b = color->b;
    result->a = color->a;
    return miTRUE;
}
```

Figure 5.9: Putting the standard shader arguments on a single line

But we can go further than just rearranging the code to shorten it. The size of a variable of type miColor is known by the compiler in advance (a C struct containing a field for each of the red,

[redacted]

a material. In Figure 5.8, shader `one_color` is included anonymously in three different materials for the three object instances.



```
material "yellow"
    "one_color" (
        "color" 1 1 .4 )
end material

material "blue"
    "one_color" (
        "color" .4 .4 1 )
end material

material "red"
    "one_color" (
        "color" 1 .4 .4 )
end material
```

Figure 5.8: A single color for the entire object defined by shader `one_color` in the material

Throughout the book, we'll be rendering images like Figure 5.8 using the shaders developed in each chapter. Accompanying the image will be a fragment of the scene file that describes how

[redacted]

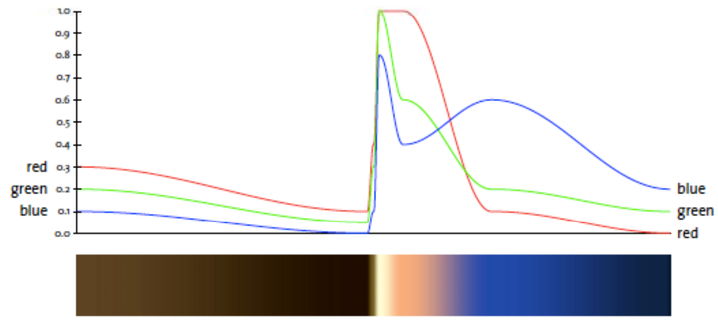


Figure 21.33: Color curves specified in the scene file for shader `color_ramp`

21.6 Environment shaders for cameras and objects

Different environment shaders can be used for the camera and object instances. For any object instance without an environment shader, the camera's environment shader will be used as the default.

```
shader "red_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.0  0.5,
                 1.0  1.0  0.0  0.51,
                 1.0  0.2  0.0  0.55,
                 0.1  0.2  0.6  0.7,
                 0.12 0.05 0.2  1.0 ] )

shader "pink_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.1  0.50,
                 1.0  1.0  0.8  0.51,
                 1.0  0.6  0.4  0.55,
                 0.1  0.2  0.6  0.7,
                 0.05 0.1  0.2  1.0 ] )

camera "cam"
    output "rgba" "tif" "environment_6.tif"
    focal 1.5
    aperture 1.5
    aspect 1
    resolution 300 300
    environment
        = "red_sunset"
end camera

material "corinthian_material"
    "specular_reflection" ()
    environment = "pink_sunset"
end material
```

Figure 21.34: Different environment shaders used for the camera and object

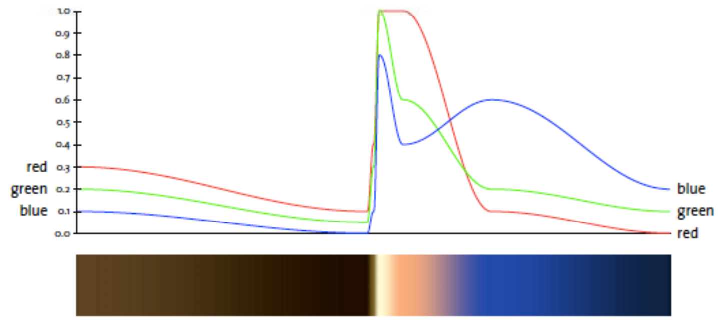


Figure 21.33: Color curves specified in the scene file for shader `color_ramp`

21.6 Environment shaders for cameras and objects

Different environment shaders can be used for the camera and object instances. For any object instance without an environment shader, the camera's environment shader will be used as the default.

```
shader "red_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.0  0.5,
                 1.0  1.0  0.0  0.51,
                 1.0  0.2  0.0  0.55,
                 0.1  0.2  0.6  0.7,
                 0.12 0.05 0.2  1.0 ] )

shader "pink_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.1  0.50,
                 1.0  1.0  0.8  0.51,
                 1.0  0.6  0.4  0.55,
                 0.1  0.2  0.6  0.7,
                 0.05 0.1  0.2  1.0 ] )

camera "cam"
    output "rgba" "tif" "environment_6.tif"
    focal 1.5
    aperture 1.5
    aspect 1
    resolution 300 300
    environment
        = "red_sunset"
end camera

material "corinthian_material"
    "specular_reflection" ()
    environment = "pink_sunset"
end material
```

Figure 21.34: Different environment shaders used for the camera and object

```
shader "red_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.0  0.5,
                 1.0  1.0  0.0  0.51,
                 1.0  0.2  0.0  0.55,
                 0.1  0.2  0.6  0.7,
                 0.12 0.05 0.2  1.0 ] )

shader "pink_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.1  0.50,
                 1.0  1.0  0.8  0.51,
                 1.0  0.6  0.4  0.55,
                 0.1  0.2  0.6  0.7,
                 0.05 0.1  0.2  1.0 ] )

camera "cam"
    output "rgba" "tif" "environment_6.tif"
    focal 1.5
    aperture 1.5
    aspect 1
    resolution 300 300
    environment
        = "red_sunset"
end camera

material "corinthian_material"
    "specular_reflection" ()
    environment = "pink_sunset"
end material
```

Figure 21.34: Different environment shaders used for the camera and object

```
shader "red_sunset"
    "color_ramp" (
        "colors" [ 0.3  0.2  0.1  0.0,
                   0.1  0.05 0.0  0.49,
                   0.4  0.3  0.0  0.5,
                   1.0  1.0  0.0  0.51,
                   1.0  0.2  0.0  0.55,
                   0.1  0.2  0.6  0.7,
                   0.12 0.05 0.2  1.0 ] )

shader "pink_sunset"
    "color_ramp" (
        "colors" [ 0.3  0.2  0.1  0.0,
                   0.1  0.05 0.0  0.49,
                   0.4  0.3  0.1  0.50,
                   1.0  1.0  0.8  0.51,
                   1.0  0.6  0.4  0.55,
                   0.1  0.2  0.6  0.7,
                   0.05 0.1  0.2  1.0 ] )

camera "cam"
     output "rgba" "tif" "environment_6.tif"
     focal 1.5
     aperture 1.5
     aspect 1
     resolution 300 300
     environment
         = "red_sunset"
end camera

material "corinthian_material"
     "specular_reflection" ()
     environment = "pink_sunset"
end material
```

Figure 21.34: Different environment shaders used for the camera and object

Cross-referencing in the shader book

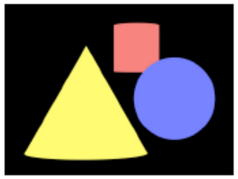
The picture index in Appendix A provides pointers back to the section in which the rendered picture was discussed.

Appendix A

Rendered scene files

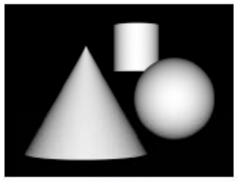
This appendix contains rendered images from the scene files used as examples throughout the book labeled with the page number on which the image appears.

Chapter 5 – A single color

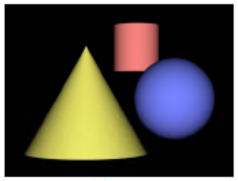


Page 54

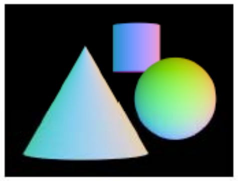
Chapter 6 – Color from orientation



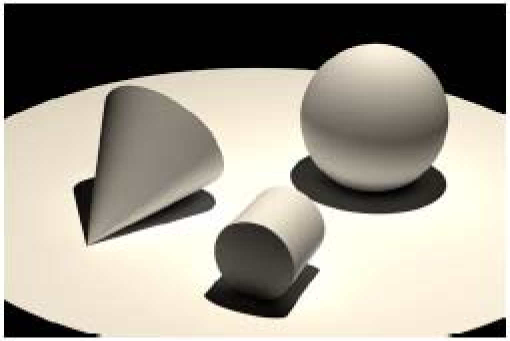
Page 59

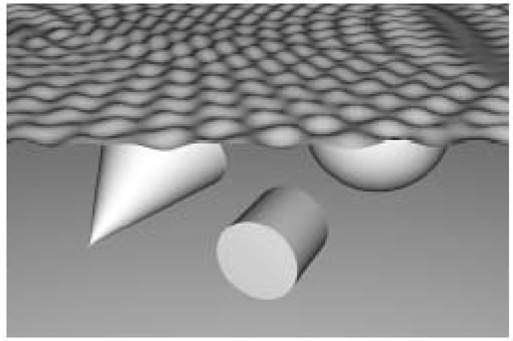


Page 59

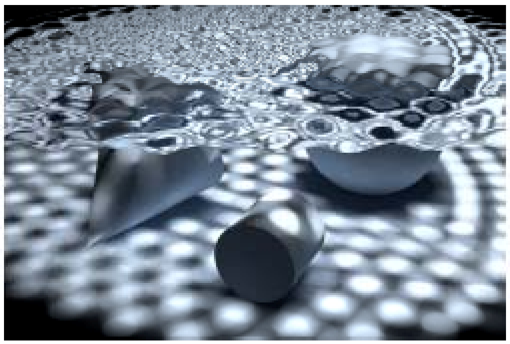


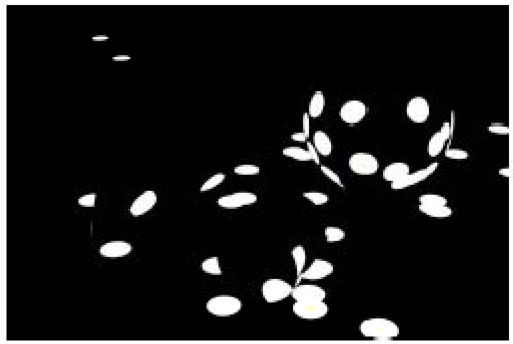
Page 62

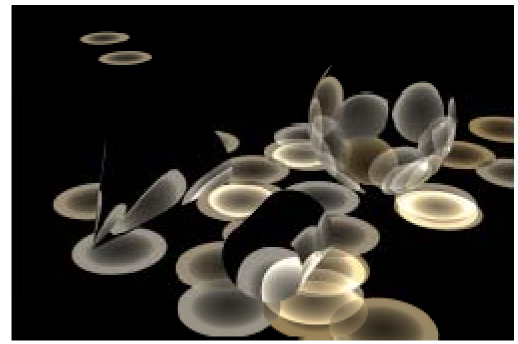

Page 232

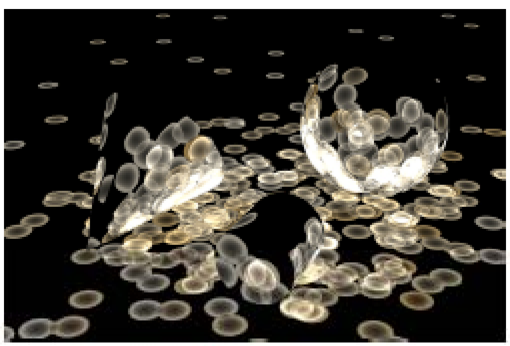

Page 233

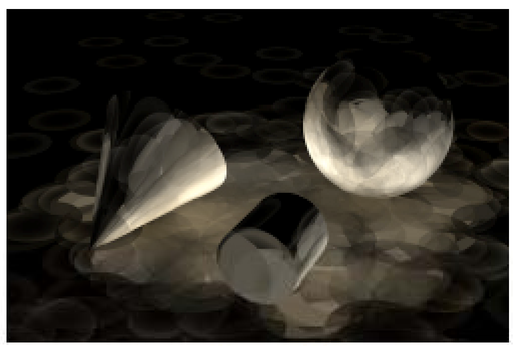

Page 234

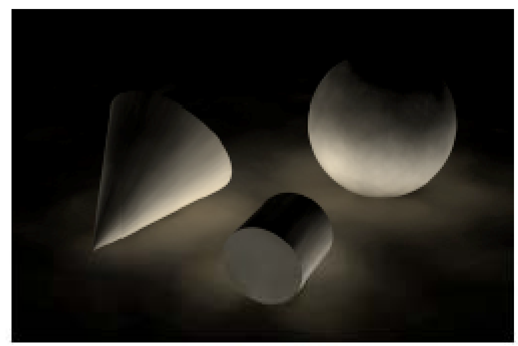

Page 235

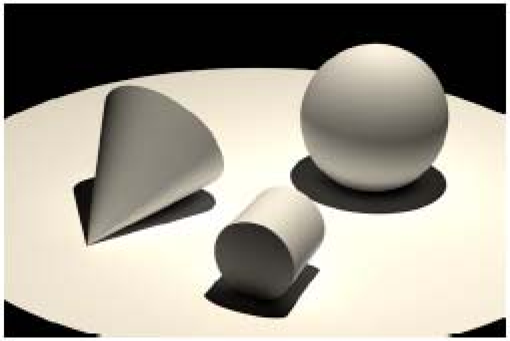

Page 236


Page 236

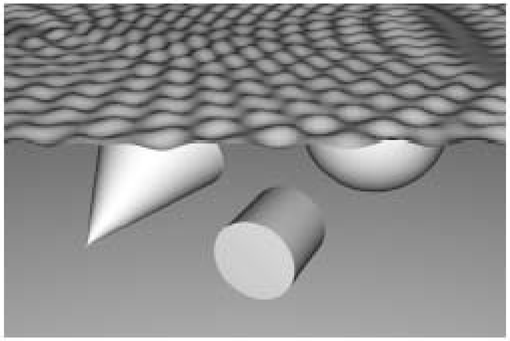

Page 237


Page 237


Page 238

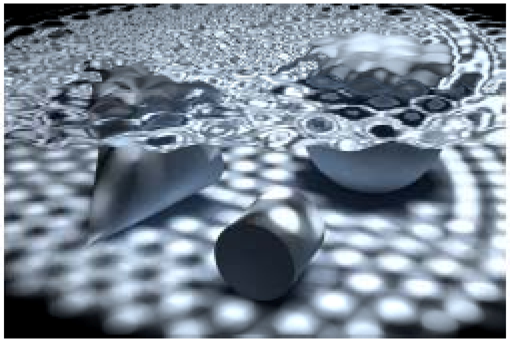


Page 232

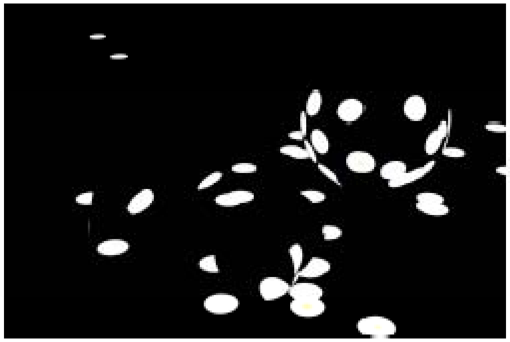


Page 233

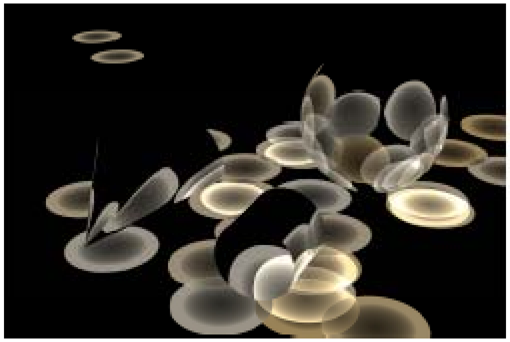
Page 234



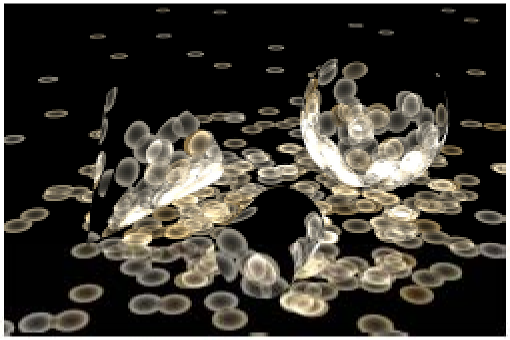
Page 235



Page 236

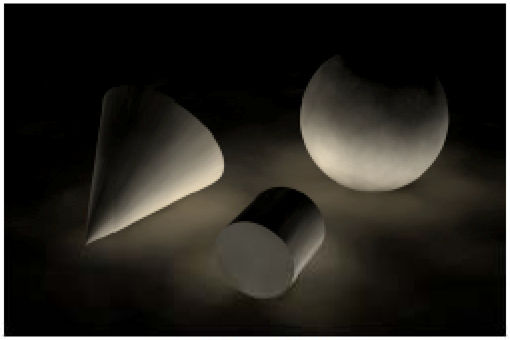


Page 236



Page 237

Page 237

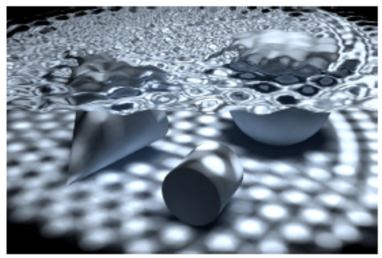


Page 238

is transmitted in **line 16**, the incoming energy is attenuated by the transparency parameter in **lines 12–13**. The refraction direction in **lines 14–15** is calculated by `mi_refraction_dir` in the same manner as the refraction shaders of Chapter 15. Here, however, we're not sampling the scene with a new ray, but sending a photon in the refraction direction to represent the transmission of light energy.

Prog 335, 3.26.4. RC Photon Functions

Adding global illumination brightens the lower parts of the objects as light from the caustics are now included in the total light calculation.



```
options "opt"
    object space
    contrast .1 .1 .1 1
    samples -1 2
    shadow on
    displace on
    max displace .2
    globillum on
    globillum accuracy 500 2
    caustic on
    caustic accuracy 500 .1
end options

material "refract"
    "specular_refraction" (
        "index_of_refraction" 1.5 )
    displace
        "displace_ripple" (
            "center" .2 .5 0,
            "frequency" 20,
            "amplitude" .01 )
        "displace_ripple" (
            "center" .8 .8 0,
            "frequency" 20,
            "amplitude" .01 )
        "displace_ripple" (
            "center" .8 .2 0,
            "frequency" 20,
            "amplitude" .01 )
    photon
        "transmit_specular_photon" (
            "index_of_refraction" 1.5 )
end material
```

Figure 16.28: Caustics and global illumination used together

16.5 Visualizing the photon map

In Chapter 6, "Color from orientation," we made a shader that converted the surface normal into a color. A vector isn't a color, after all, but by treating it as one we are able to visualize the orientation of the surface. This could be a very useful technique when we are checking for possible problems in the construction of our geometric models.

In a similar vein, we can set the global illumination options to values that would not be useful for producing final imagery, but can help us understand the way that the photon tracing process distributes photons to construct the photon map for its use in rendering.

Rend 424, 19.5. Final Gathering, Global Illumination, and Caustics
Prog 215, 3.4.7. Options

In the photon tracing phase, the energy of a light emitting photons is divided up equally among

Cross-referencing in the shader book

The textual index includes references to shader and function names.

Index

Index references to shader source code

Index references to shader source code

Index references to shader source code

Index references to rendering state

Index references to API function calls

Index references to utility functions

Website support for the book's software

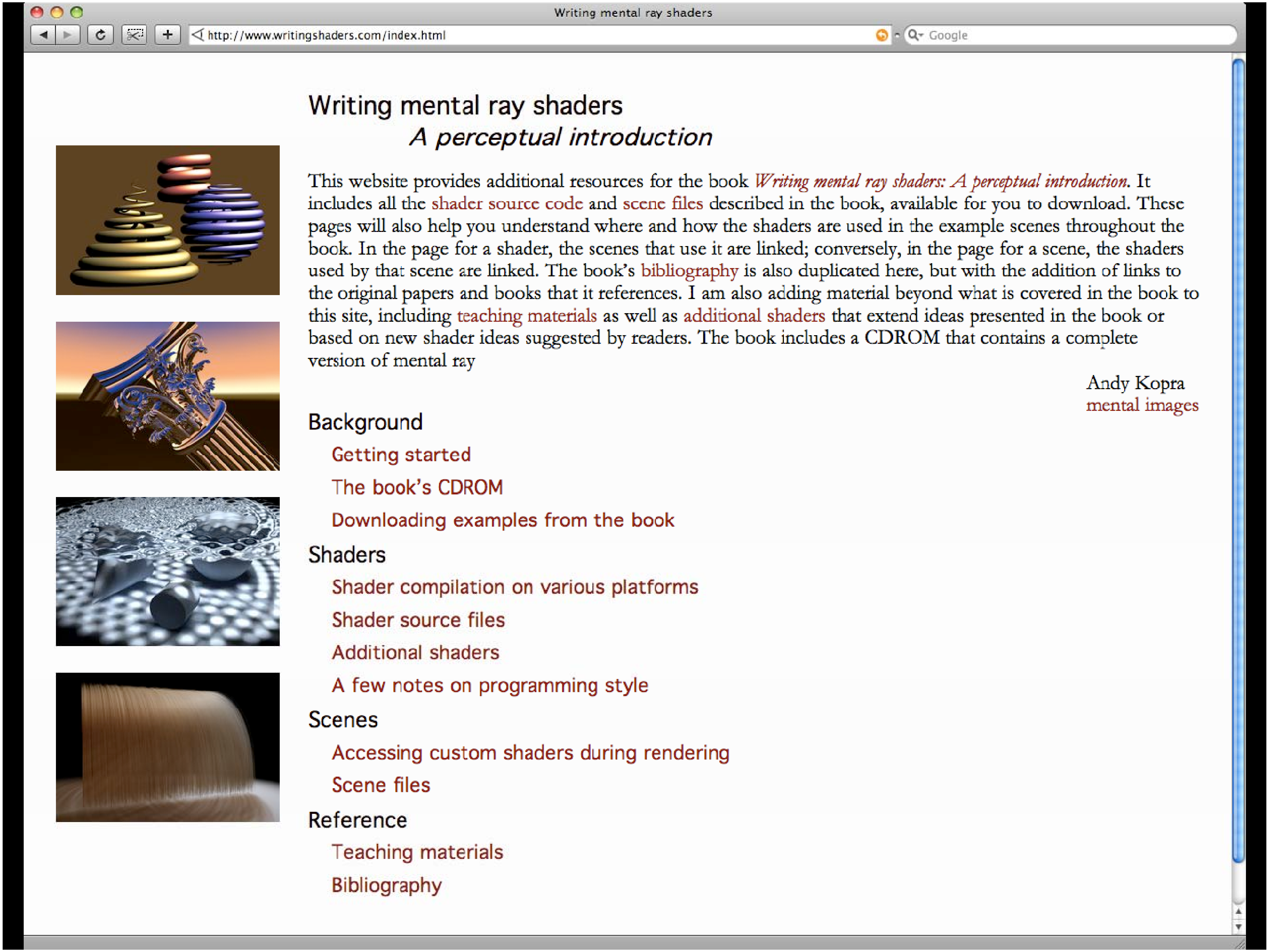
4

nvision 08
THE WORLD OF VISUAL COMPUTING

NVIDIA

Website support for the book's software

http://www.writingshaders.com/



Writing mental ray shaders
http://www.writingshaders.com/index.html
Google
Writing mental ray shaders
A perceptual introduction
This website provides additional resources for the book Writing mental ray shaders: A perceptual introduction. It includes all the shader source code and scene files described in the book, available for you to download. These pages will also help you understand where and how the shaders are used in the example scenes throughout the book. In the page for a shader, the scenes that use it are linked; conversely, in the page for a scene, the shaders used by that scene are linked. The book's bibliography is also duplicated here, but with the addition of links to the original papers and books that it references. I am also adding material beyond what is covered in the book to this site, including teaching materials as well as additional shaders that extend ideas presented in the book or based on new shader ideas suggested by readers. The book includes a CDROM that contains a complete version of mental ray
Andy Kopra
mental images
Background
Getting started
The book's CDROM
Downloading examples from the book
Shaders
Shader compilation on various platforms
Shader source files
Additional shaders
A few notes on programming style
Scenes
Accessing custom shaders during rendering
Scene files
Reference
Teaching materials
Bibliography

Website support for the book's software

The "Background" section is an overview of the resources provided by the website and how you can use them.

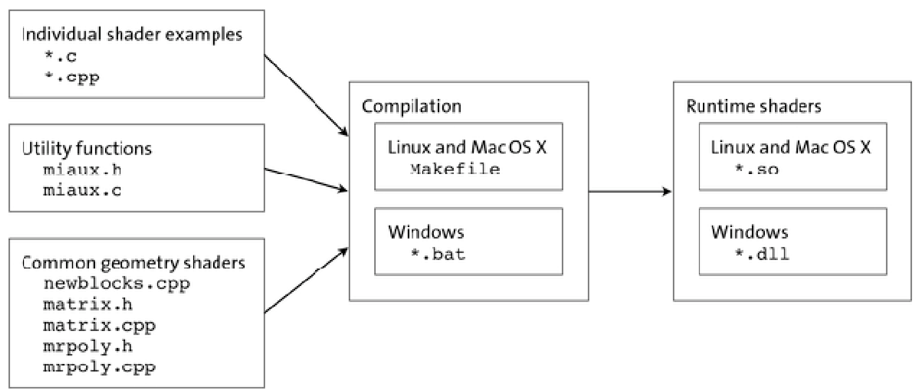
Getting started

Home

The good news is that mental ray shaders are written in the C or C++ programming languages. Unfortunately, the bad news is that … mental ray shaders are written in C or C++. But let's focus on the good part: Since the shaders are in C or C++, everything you know about programming in those languages, and all the libraries that you've written or have acquired, can potentially be useful in mental ray shader programming. However, the way that you compile C/C++ code varies in its details across different operating systems, and getting this right can be a frustrating hurdle right at the beginning. The installation of mental ray and your custom shaders are also dependent upon the structure of the file system. I've included a lot of information in this website to deal with these issues. I hope this page will help clarify the big picture so that you don't get lost in the details right off the bat.

Compiling shaders

The page "Shader compilation on various platforms" describes the various pieces required to compile the shaders in the book.



Compiling the shaders in the book from their source files

The book organizes the shader code in a simple way: each shader is almost always defined in its own file. (I say "almost always" because contour shading is divided into four processes, each represented by a separate shader. I've organized one set of related contours shaders in a single file in Chapter 10 of the c_tessellate shader.)

Many shaders also use utility functions from the "miaux" library, written for the book and designed as an auxiliary set to complement the "mi" library supplied with mental ray.

I also defined a few simple shapes as geometry shaders in file "newblocks.cpp". The "newblocks" shaders depend up a library of C++ classes called "mrpoly". These classes are an initial sketch of how you can approach geometry shaders at a higher level through class design. The low-level

Downloading examples from the book

http://www.writingshaders.com/download.html

Downloading examples from the book

Home

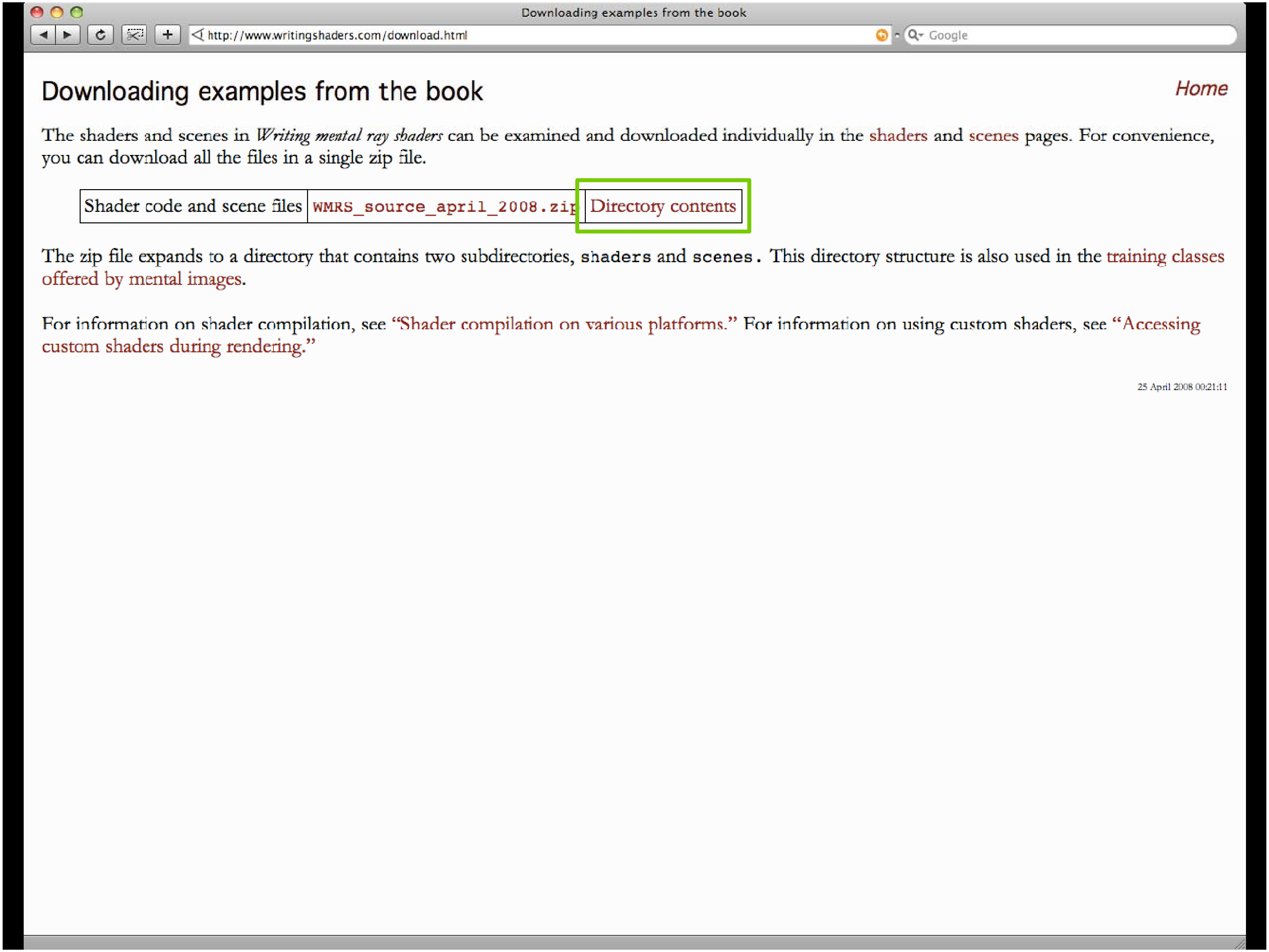
The shaders and scenes in *Writing mental ray shaders* can be examined and downloaded individually in the shaders and scenes pages. For convenience, you can download all the files in a single zip file.

Shader code and scene files	`WMRS_source_april_2008.zip`	Directory contents

The zip file expands to a directory that contains two subdirectories, `shaders` and `scenes`. This directory structure is also used in the training classes offered by mental images.

For information on shader compilation, see "Shader compilation on various platforms." For information on using custom shaders, see "Accessing custom shaders during rendering."

25 April 2008 00:21:11


Downloading examples from the book
http://www.writingshaders.com/download.html
Google
Downloading examples from the book
Home
The shaders and scenes in Writing mental ray shaders can be examined and downloaded individually in the shaders and scenes pages. For convenience, you can download all the files in a single zip file.
Shader code and scene files
WMRS_source_april_2008.zip
Directory contents
The zip file expands to a directory that contains two subdirectories, shaders and scenes. This directory structure is also used in the training classes offered by mental images.
For information on shader compilation, see "Shader compilation on various platforms." For information on using custom shaders, see "Accessing custom shaders during rendering."
25 April 2008 00:21:11

Contents of WMRS_source_april_2008.zip

http://www.writingshaders.com/WMRS_source_april_2008_contents.html

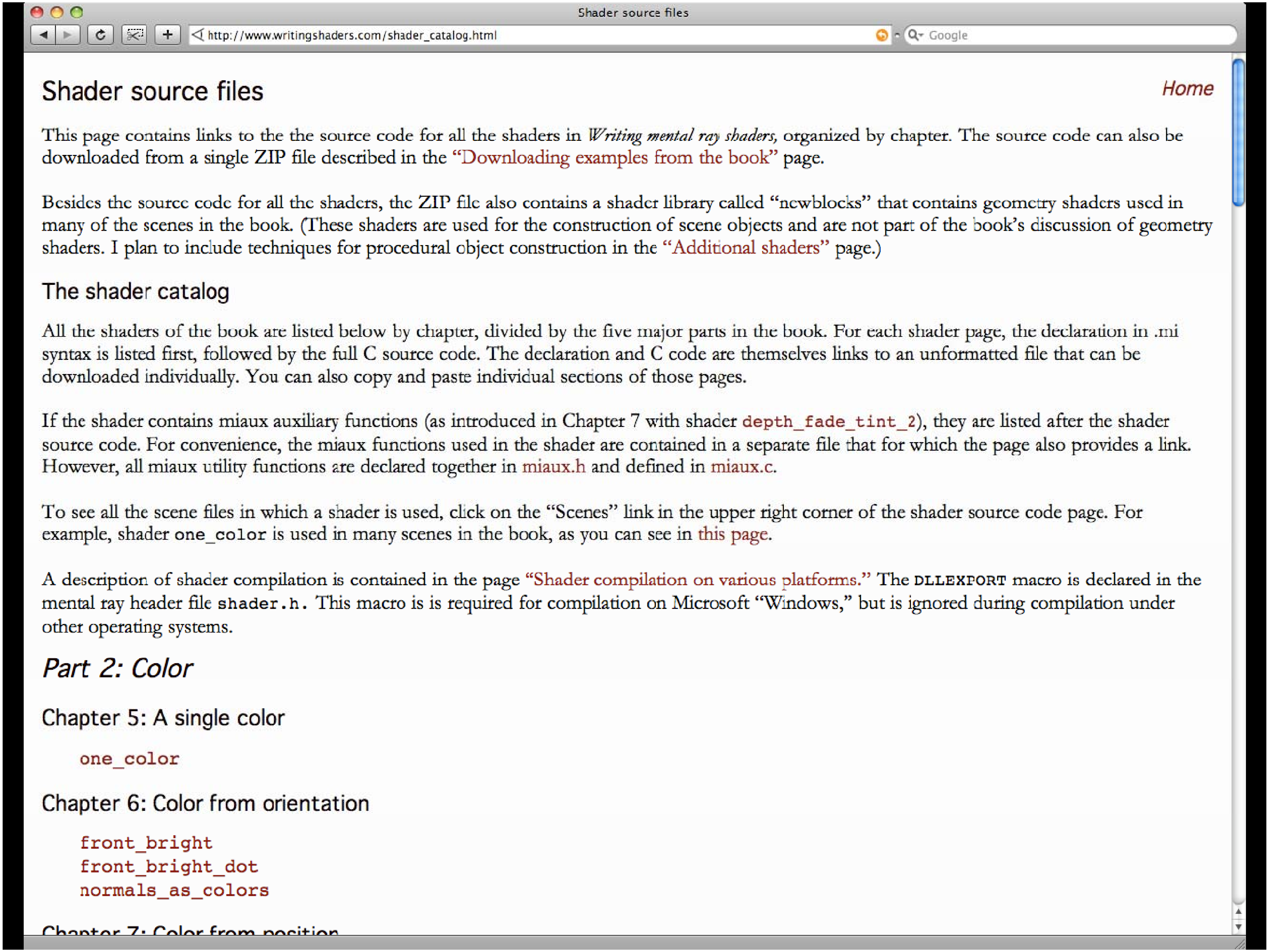
Contents of WMRS_source_april_2008.zip

Home

```
Archive:  WMRS_source_april_2008.zip
  Length     Date    Time    Name
 --------    ----    ----    ----
        0  04-22-08 23:52   WMRS_source_april_2008/
        0  04-22-08 22:45   WMRS_source_april_2008/scenes/
     2012  04-22-08 22:45   WMRS_source_april_2008/scenes/ambocclude_1.mi
     2013  04-22-08 22:45   WMRS_source_april_2008/scenes/ambocclude_2.mi
     2063  04-22-08 22:45   WMRS_source_april_2008/scenes/ambocclude_3.mi
     5135  04-22-08 22:45   WMRS_source_april_2008/scenes/atmosphere_1.mi
     5273  04-22-08 22:45   WMRS_source_april_2008/scenes/atmosphere_2.mi
     5272  04-22-08 22:45   WMRS_source_april_2008/scenes/atmosphere_3.mi
     5620  04-22-08 22:45   WMRS_source_april_2008/scenes/atmosphere_4.mi
     5672  04-22-08 22:45   WMRS_source_april_2008/scenes/atmosphere_5.mi
     5679  04-22-08 22:45   WMRS_source_april_2008/scenes/atmosphere_6.mi
     5682  04-22-08 22:45   WMRS_source_april_2008/scenes/atmosphere_7.mi
 12591274  04-22-08 22:45   WMRS_source_april_2008/scenes/bubbles.tif
     2901  04-22-08 22:45   WMRS_source_april_2008/scenes/buffer_1.mi
     3175  04-22-08 22:45   WMRS_source_april_2008/scenes/buffer_2.mi
     3710  04-22-08 22:45   WMRS_source_april_2008/scenes/buffer_3.mi
     4344  04-22-08 22:45   WMRS_source_april_2008/scenes/buffer_4.mi
     6080  04-22-08 22:45   WMRS_source_april_2008/scenes/buffer_5.mi
     1310  04-22-08 22:45   WMRS_source_april_2008/scenes/bump_1.mi
     1367  04-22-08 22:45   WMRS_source_april_2008/scenes/bump_2.mi
     1334  04-22-08 22:45   WMRS_source_april_2008/scenes/bump_3.mi
     1450  04-22-08 22:45   WMRS_source_april_2008/scenes/bump_4.mi
     1374  04-22-08 22:45   WMRS_source_april_2008/scenes/bump_5.mi
     1582  04-22-08 22:45   WMRS_source_april_2008/scenes/bump_7.mi
     3284  04-22-08 22:45   WMRS_source_april_2008/scenes/caustics_1.mi
     3882  04-22-08 22:45   WMRS_source_april_2008/scenes/caustics_2.mi
     3928  04-22-08 22:45   WMRS_source_april_2008/scenes/caustics_3.mi
   135464  04-22-08 22:45   WMRS_source_april_2008/scenes/colorstrip.tif
    19376  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_10.mi
    20316  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_11.mi
    19324  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_12.mi
    19750  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_13.mi
    20559  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_14.mi
    20580  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_15.mi
    20586  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_16.mi
    46560  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_2.mi
    46766  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_3.mi
    47048  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_4.mi
    50416  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_5.mi
    50410  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_6.mi
    50565  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_7.mi
    18942  04-22-08 22:45   WMRS_source_april_2008/scenes/contours_8.mi
```

Website support for the book's software

The "Shaders" section contains a catalog of all the shaders in the book, along with instructions for compilation on various platforms.



Shader source files

Home

This page contains links to the the source code for all the shaders in *Writing mental ray shaders,* organized by chapter. The source code can also be downloaded from a single ZIP file described in the "Downloading examples from the book" page.

Besides the source code for all the shaders, the ZIP file also contains a shader library called "newblocks" that contains geometry shaders used in many of the scenes in the book. (These shaders are used for the construction of scene objects and are not part of the book's discussion of geometry shaders. I plan to include techniques for procedural object construction in the "Additional shaders" page.)

The shader catalog

All the shaders of the book are listed below by chapter, divided by the five major parts in the book. For each shader page, the declaration in .mi syntax is listed first, followed by the full C source code. The declaration and C code are themselves links to an unformatted file that can be downloaded individually. You can also copy and paste individual sections of those pages.

If the shader contains miaux auxiliary functions (as introduced in Chapter 7 with shader `depth_fade_tint_2`), they are listed after the shader source code. For convenience, the miaux functions used in the shader are contained in a separate file that for which the page also provides a link. However, all miaux utility functions are declared together in miaux.h and defined in miaux.c.

To see all the scene files in which a shader is used, click on the "Scenes" link in the upper right corner of the shader source code page. For example, shader `one_color` is used in many scenes in the book, as you can see in this page.

A description of shader compilation is contained in the page "Shader compilation on various platforms." The **`DLLEXPORT`** macro is declared in the mental ray header file **`shader.h`**. This macro is is required for compilation on Microsoft "Windows," but is ignored during compilation under other operating systems.

Part 2: Color

Chapter 5: A single color

`one_color`

Chapter 6: Color from orientation

`front_bright`
`front_bright_dot`
`normals_as_colors`

Chapter 7: Color from position

Shader source files

http://www.writingshaders.com/shader_catalog.html

Chapter 8: The transparency of a surface

transparent
transparent_modularized

Chapter 9: Color from functions

show_uv
show_uv_steps
texture_uv_simple
texture_uv
vertex_color
summed_noise_color

Chapter 10: The color of edges

c_store
c_contrast
c_contour
c_output
c_tessellate
show_barycentric
front_bright_steps
c_toon
lambert_steps

Part 3: Light

Chapter 11: Lights

point_light
point_light_shadow
spotlight
soft_spotlight
sinusoid_soft_spotlight
point_light_falloff

Chapter 12: Light on a surface

Shader source files

http://www.writingshaders.com/shader_catalog.html

Chapter 8: The transparency of a surface

transparent
transparent_modularized

Chapter 9: Color from functions

show_uv
show_uv_steps
texture_uv_simple
texture_uv
vertex_color
summed_noise_color

Chapter 10: The color of edges

c_store
c_contrast
c_contour
c_output
c_tessellate
show_barycentric
front_bright_steps
c_toon
lambert_steps

Part 3: Light

Chapter 11: Lights

point_light
point_light_shadow
spotlight
soft_spotlight
sinusoid_soft_spotlight
point_light_falloff

Chapter 12: Light on a surface

Shader source for point_light_shadow

http://www.writingshaders.com/shader_pages/point_light_shadow.html

Shader point_light_shadow

Scenes Home

Click on the filename to display or download the file.

point_light_shadow.mi

```
declare shader
    color "point_light_shadow" (
        color "light_color" default 1 1 1 )
    version 1
    apply light
end declare
```

point_light_shadow.c

```
#include "shader.h"

struct point_light_shadow {
    miColor light_color;
};

DLLEXPORT
int point_light_shadow_version(void) { return 1; }

DLLEXPORT
miBoolean point_light_shadow (
    miColor *result, miState *state, struct point_light_shadow *params )
{
    *result = *mi_eval_color(&params->light_color);
    return mi_trace_shadow(result, state);
}
```

22 April 2008 23:40:03

Shader source for point_light_shadow

http://www.writingshaders.com/shader_pages/point_light_shadow.html

Shader point_light_shadow

Scenes Home

Click on the filename to display or download the file.

---

point_light_shadow.mi

```
declare shader
    color "point_light_shadow" (
        color "light_color" default 1 1 1 )
    version 1
    apply light
end declare
```

---

point_light_shadow.c

```
#include "shader.h"

struct point_light_shadow {
    miColor light_color;
};

DLLEXPORT
int point_light_shadow_version(void) { return 1; }

DLLEXPORT
miBoolean point_light_shadow (
    miColor *result, miState *state, struct point_light_shadow *params )
{
    *result = *mi_eval_color(&params->light_color);
    return mi_trace_shadow(result, state);
}
```

---

22 April 2008 23:40:03

Scenes using shader point_light_shadow

http://www.writingshaders.com/shader_scenes/point_light_shadow_usage.html

Scenes using shader `point_light_shadow`

Home

Click on the chapter title to see all the scenes in that chapter. Click on a filename to display or download that scene file.

Chapter 10: The color of edges

contours_4.mi
contours_15.mi
contours_16.mi

Chapter 11: Lights

lights_4.mi
lights_8.mi

Chapter 12: Light on a surface

shading_1.mi
shading_2.mi
shading_3.mi
shading_4.mi
shading_5.mi
shading_6.mi
shading_7.mi

Chapter 13: Shadows

shadows_1.mi
shadows_2.mi
shadows_A.mi
shadows_B.mi
shadows_C.mi
shadows_C1.mi
shadows_D.mi
shadows_E.mi
shadows_F.mi
shadows_G.mi

Chapter 19: Creating geometric objects

Shader source for lambert

http://www.writingshaders.com/shader_pages/lambert.html

Shader lambert

Scenes Home

Click on the filename to display or download the file.

---

lambert.mi

```
declare shader
    color "lambert" (
        color "ambient" default 0 0 0,
        color "diffuse" default 1 1 1,
        array light "lights" )
    version 1
    apply material
end declare
```

---

lambert.c

```
#include "shader.h"
#include "miaux.h"

struct lambert {
    miColor ambient;
    miColor diffuse;
    int     i_light;
    int     n_light;
    miTag   light[1];
};

DLLEXPORT
int lambert_version(void) { return 1; }

DLLEXPORT
miBoolean lambert (
    miColor *result, miState *state, struct lambert *params  )
{
    int i, light_count, light_sample_count;
    miColor sum, light_color;
    miScalar dot_nl;
    miTag *light;

    miColor *diffuse  = mi_eval_color(&params->diffuse);
    miaux_light_array(&light, &light_count, state,
                      &params->i_light, &params->n_light, params->light);
    *result = *mi_eval_color(&params->ambient);

    for (i = 0; i < light_count; i++, light++) {
```

Shader source for lambert

http://www.writingshaders.com/shader_pages/lambert.html

Shader lambert

Scenes *Home*

Click on the filename to display or download the file.

---

lambert.mi

```
declare shader
    color "lambert" (
        color "ambient" default 0 0 0,
        color "diffuse" default 1 1 1,
        array light "lights" )
    version 1
    apply material
end declare
```

---

lambert.c

```
#include "shader.h"
#include "miaux.h"

struct lambert {
    miColor ambient;
    miColor diffuse;
    int     i_light;
    int     n_light;
    miTag   light[1];
};

DLLEXPORT
int lambert_version(void) { return 1; }

DLLEXPORT
miBoolean lambert (
    miColor *result, miState *state, struct lambert *params  )
{
    int i, light_count, light_sample_count;
    miColor sum, light_color;
    miScalar dot_nl;
    miTag *light;

    miColor *diffuse  = mi_eval_color(&params->diffuse);
    miaux_light_array(&light, &light_count, state,
                      &params->i_light, &params->n_light, params->light);
    *result = *mi_eval_color(&params->ambient);

    for (i = 0; i < light_count; i++, light++) {
```

http://www.writingshaders.com/shader_sources/lambert.c

```
#include "shader.h"
#include "miaux.h"

struct lambert {
    miColor ambient;
    miColor diffuse;
    int     i_light;
    int     n_light;
    miTag   light[1];
};

DLLEXPORT
int lambert_version(void) { return 1; }

DLLEXPORT
miBoolean lambert (
    miColor *result, miState *state, struct lambert *params  )
{
    int i, light_count, light_sample_count;
    miColor sum, light_color;
    miScalar dot_nl;
    miTag *light;

    miColor *diffuse  = mi_eval_color(&params->diffuse);
    miaux_light_array(&light, &light_count, state,
                      &params->i_light, &params->n_light, params->light);
    *result = *mi_eval_color(&params->ambient);

    for (i = 0; i < light_count; i++, light++) {
        miaux_set_channels(&sum, 0);
        light_sample_count = 0;
        while (mi_sample_light(&light_color, NULL, &dot_nl,
                               state, *light, &light_sample_count))
            miaux_add_diffuse_component(&sum, dot_nl, diffuse, &light_color);
        if (light_sample_count)
            miaux_add_scaled_color(result, &sum, 1.0/light_sample_count);
    }
    return miTRUE;
}
```

Shader source for lambert

http://www.writingshaders.com/shader_pages/lambert.html

Shader lambert

Scenes Home

Click on the filename to display or download the file.

---

lambert.mi

```
declare shader
    color "lambert" (
        color "ambient" default 0 0 0,
        color "diffuse" default 1 1 1,
        array light "lights" )
    version 1
    apply material
end declare
```

---

lambert.c

```
#include "shader.h"
#include "miaux.h"

struct lambert {
    miColor ambient;
    miColor diffuse;
    int     i_light;
    int     n_light;
    miTag   light[1];
};

DLLEXPORT
int lambert_version(void) { return 1; }

DLLEXPORT
miBoolean lambert (
    miColor *result, miState *state, struct lambert *params  )
{
    int i, light_count, light_sample_count;
    miColor sum, light_color;
    miScalar dot_nl;
    miTag *light;

    miColor *diffuse  = mi_eval_color(&params->diffuse);
    miaux_light_array(&light, &light_count, state,
                      &params->i_light, &params->n_light, params->light);
    *result = *mi_eval_color(&params->ambient);

    for (i = 0; i < light_count; i++, light++) {
```

Shader source for lambert

http://www.writingshaders.com/shader_pages/lambert.html

Shader lambert

Scenes Home

Click on the filename to display or download the file.

---

lambert.mi

```
declare shader
    color "lambert" (
        color "ambient" default 0 0 0,
        color "diffuse" default 1 1 1,
        array light "lights" )
    version 1
    apply material
end declare
```

---

lambert.c

```
#include "shader.h"
#include "miaux.h"

struct lambert {
    miColor ambient;
    miColor diffuse;
    int     i_light;
    int     n_light;
    miTag   light[1];
};

DLLEXPORT
int lambert_version(void) { return 1; }

DLLEXPORT
miBoolean lambert (
    miColor *result, miState *state, struct lambert *params  )
{
    int i, light_count, light_sample_count;
    miColor sum, light_color;
    miScalar dot_nl;
    miTag *light;

    miColor *diffuse  = mi_eval_color(&params->diffuse);
    miaux_light_array(&light, &light_count, state,
                      &params->i_light, &params->n_light, params->light);
    *result = *mi_eval_color(&params->ambient);

    for (i = 0; i < light_count; i++, light++) {
```

Shader source for lambert

http://www.writingshaders.com/shader_pages/lambert.html

```
    *result = *mi_eval_color(&params->ambient);

    for (i = 0; i < light_count; i++, light++) {
        miaux_set_channels(&sum, 0);
        light_sample_count = 0;
        while (mi_sample_light(&light_color, NULL, &dot_nl,
                               state, *light, &light_sample_count))
            miaux_add_diffuse_component(&sum, dot_nl, diffuse, &light_color);
        if (light_sample_count)
            miaux_add_scaled_color(result, &sum, 1.0/light_sample_count);
    }
    return miTRUE;
}
```

---

lambert_util.c

```
void miaux_light_array(miTag **lights, int *light_count, miState *state,
                       int *offset_param, int *count_param, miTag *lights_param)
{
    int array_offset = *mi_eval_integer(offset_param);
    *light_count = *mi_eval_integer(count_param);
    *lights = mi_eval_tag(lights_param) + array_offset;
}

void miaux_set_channels(miColor *c, miScalar new_value)
{
    c->r = c->g = c->b = c->a = new_value;
}

void miaux_add_diffuse_component(
    miColor *result,
    miScalar light_and_surface_cosine,
    miColor *diffuse, miColor *light_color)
{
    result->r += light_and_surface_cosine * diffuse->r * light_color->r;
    result->g += light_and_surface_cosine * diffuse->g * light_color->g;
    result->b += light_and_surface_cosine * diffuse->b * light_color->b;
}

void miaux_add_scaled_color(miColor *result, miColor *color, miScalar scale)
{
    result->r += color->r * scale;
    result->g += color->g * scale;
    result->b += color->b * scale;
}
```

---

22 April 2008 23:40:07

Shader source for lambert

http://www.writingshaders.com/shader_pages/lambert.html

```
    *result = *mi_eval_color(&params->ambient);

    for (i = 0; i < light_count; i++, light++) {
        miaux_set_channels(&sum, 0);
        light_sample_count = 0;
        while (mi_sample_light(&light_color, NULL, &dot_nl,
                               state, *light, &light_sample_count))
            miaux_add_diffuse_component(&sum, dot_nl, diffuse, &light_color);
        if (light_sample_count)
            miaux_add_scaled_color(result, &sum, 1.0/light_sample_count);
    }
    return miTRUE;
}
```

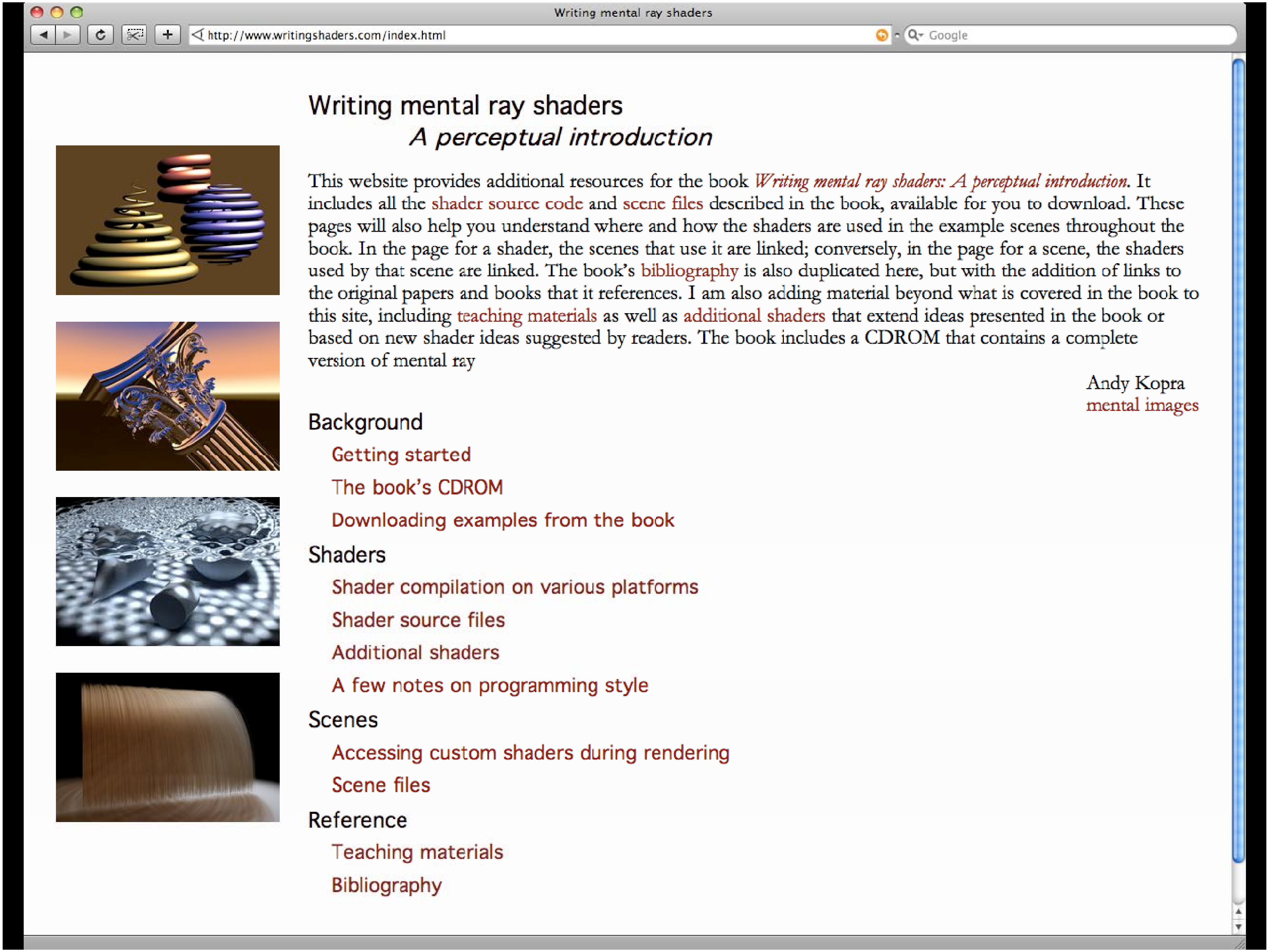
---

lambert_util.c

```
void miaux_light_array(miTag **lights, int *light_count, miState *state,
                       int *offset_param, int *count_param, miTag *lights_param)
{
    int array_offset = *mi_eval_integer(offset_param);
    *light_count = *mi_eval_integer(count_param);
    *lights = mi_eval_tag(lights_param) + array_offset;
}

void miaux_set_channels(miColor *c, miScalar new_value)
{
    c->r = c->g = c->b = c->a = new_value;
}

void miaux_add_diffuse_component(
    miColor *result,
    miScalar light_and_surface_cosine,
    miColor *diffuse, miColor *light_color)
{
    result->r += light_and_surface_cosine * diffuse->r * light_color->r;
    result->g += light_and_surface_cosine * diffuse->g * light_color->g;
    result->b += light_and_surface_cosine * diffuse->b * light_color->b;
}

void miaux_add_scaled_color(miColor *result, miColor *color, miScalar scale)
{
    result->r += color->r * scale;
    result->g += color->g * scale;
    result->b += color->b * scale;
}
```

---

22 April 2008 23:40:07

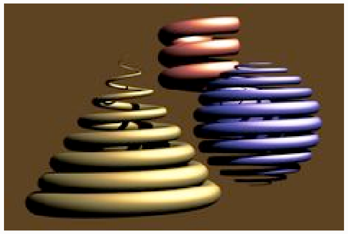

Writing mental ray shaders
http://www.writingshaders.com/index.html
Google
Writing mental ray shaders
A perceptual introduction
This website provides additional resources for the book Writing mental ray shaders: A perceptual introduction. It includes all the shader source code and scene files described in the book, available for you to download. These pages will also help you understand where and how the shaders are used in the example scenes throughout the book. In the page for a shader, the scenes that use it are linked; conversely, in the page for a scene, the shaders used by that scene are linked. The book's bibliography is also duplicated here, but with the addition of links to the original papers and books that it references. I am also adding material beyond what is covered in the book to this site, including teaching materials as well as additional shaders that extend ideas presented in the book or based on new shader ideas suggested by readers. The book includes a CDROM that contains a complete version of mental ray
Andy Kopra
mental images
Background
Getting started
The book's CDROM
Downloading examples from the book
Shaders
Shader compilation on various platforms
Shader source files
Additional shaders
A few notes on programming style
Scenes
Accessing custom shaders during rendering
Scene files
Reference
Teaching materials
Bibliography

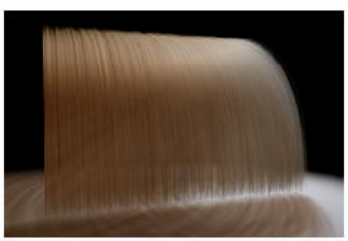
Writing mental ray shaders

http://www.writingshaders.com/index.html

Writing mental ray shaders

A perceptual introduction





This website provides additional resources for the book *Writing mental ray shaders: A perceptual introduction*. It includes all the shader source code and scene files described in the book, available for you to download. These pages will also help you understand where and how the shaders are used in the example scenes throughout the book. In the page for a shader, the scenes that use it are linked; conversely, in the page for a scene, the shaders used by that scene are linked. The book's bibliography is also duplicated here, but with the addition of links to the original papers and books that it references. I am also adding material beyond what is covered in the book to this site, including teaching materials as well as additional shaders that extend ideas presented in the book or based on new shader ideas suggested by readers. The book includes a CDROM that contains a complete version of mental ray

Andy Kopra
mental images

Background

- Getting started
- The book's CDROM
- Downloading examples from the book

Shaders

- Shader compilation on various platforms
- Shader source files
- Additional shaders
- A few notes on programming style

Scenes

- Accessing custom shaders during rendering
- Scene files

Reference

- Teaching materials
- Bibliography

Framebuffers — new strategies and syntax in mental ray 3.6

Home

Chapter 25 demonstrates how rendering components can be saved into separate files using framebuffers. This page describes the simplified scene file syntax for the definition of framebuffers introduced in mental ray version 3.6 as well as the use of geometry shaders in framebuffer definition.

A review of framebuffer use prior to version 3.6

In releases of mental ray prior to 3.6, framebuffers defined by the user are identified by integer indices, and named simply by preceding the framebuffer index with the string "fb". These names are defined in the options block of the scene, for example, in lines 7-9 in this set of options statements from scene file `buffer_5.mi` from Chapter 25.

```
 1  options "opt"
 2      object space
 3      contrast .1 .1 .1 1
 4      samples 0 2
 5      finalgather on
 6      finalgather accuracy 50 2 .5
 7      frame buffer 0 "+rgba"
 8      frame buffer 1 "+rgba"
 9      frame buffer 2 "+rgba"
10  end options
```

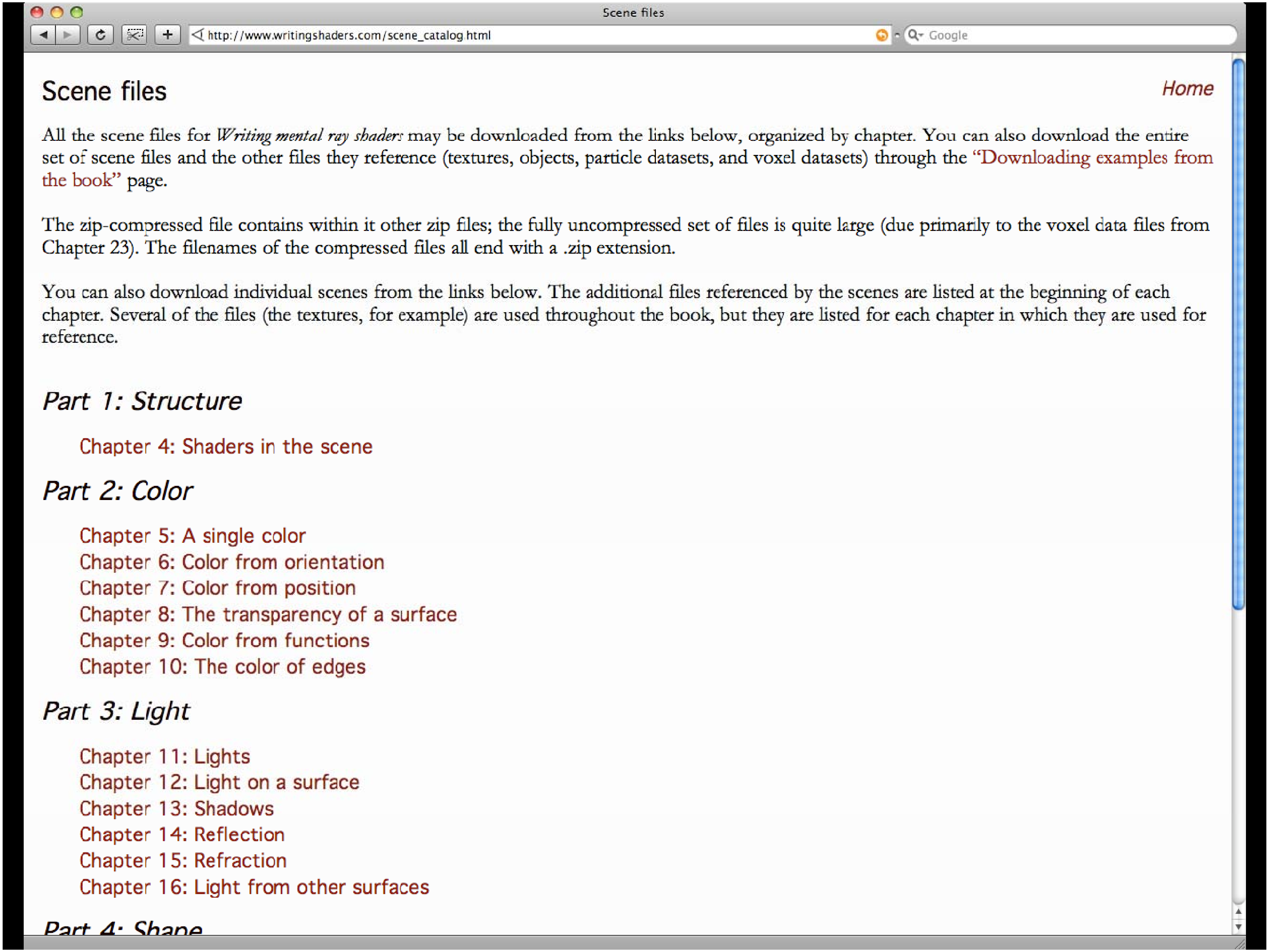
During rendering, shaders can access framebuffers using the API functions `mi_fb_put()` and `mi_fb_get()`. For example, Chapter 25 defines the shader `framebuffer_put` that simply passes along the color in the `result` pointer, but with the side effect of storing the color in a framebuffer with `mi_fb_put()` in line 7:

```
 1  miBoolean framebuffer_put(
 2      miColor *result, miState *state, struct framebuffer_put *params)
 3  {
 4      *result = *mi_eval_color(&params->color);
 5
 6      if (state->type == miRAY_EYE)
 7          mi_fb_put(state, *mi_eval_integer(&params->index), result);
 8
 9      return miTRUE;
10  }
```

When rendering is done, framebuffers are written to the files defined by "output statements" in the camera:

Website support for the book’s software

The “Scenes” section contains a rendered scene for all the examples in the book.



Scene files

Home

All the scene files for *Writing mental ray shaders* may be downloaded from the links below, organized by chapter. You can also download the entire set of scene files and the other files they reference (textures, objects, particle datasets, and voxel datasets) through the "Downloading examples from the book" page.

The zip-compressed file contains within it other zip files; the fully uncompressed set of files is quite large (due primarily to the voxel data files from Chapter 23). The filenames of the compressed files all end with a .zip extension.

You can also download individual scenes from the links below. The additional files referenced by the scenes are listed at the beginning of each chapter. Several of the files (the textures, for example) are used throughout the book, but they are listed for each chapter in which they are used for reference.

Part 1: Structure

Chapter 4: Shaders in the scene

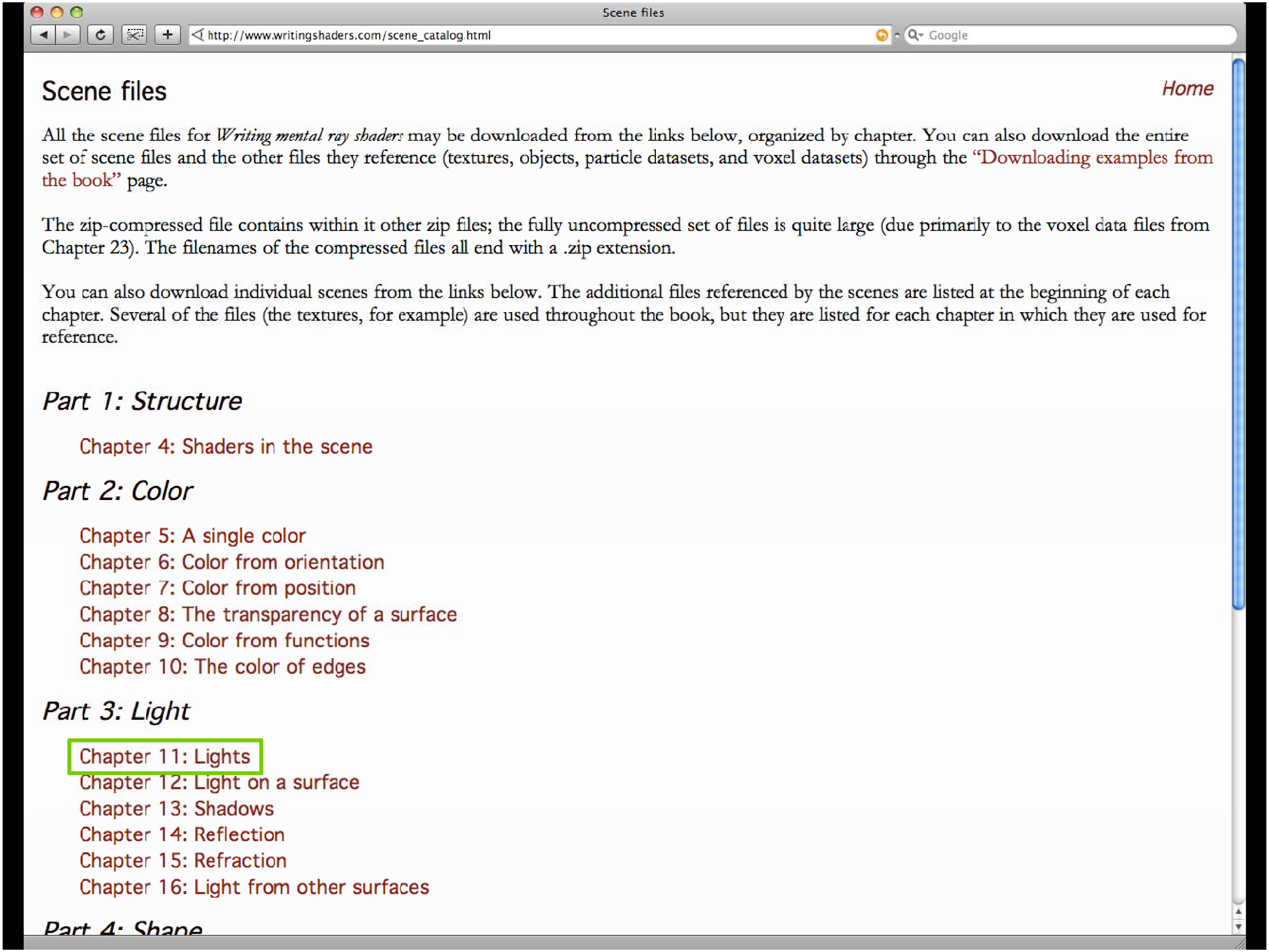
Part 2: Color

Chapter 5: A single color
Chapter 6: Color from orientation
Chapter 7: Color from position
Chapter 8: The transparency of a surface
Chapter 9: Color from functions
Chapter 10: The color of edges

Part 3: Light

Chapter 11: Lights
Chapter 12: Light on a surface
Chapter 13: Shadows
Chapter 14: Reflection
Chapter 15: Refraction
Chapter 16: Light from other surfaces

Part 4: Shape



Scene files

http://www.writingshaders.com/scene_catalog.html

Scene files

Home

All the scene files for *Writing mental ray shaders* may be downloaded from the links below, organized by chapter. You can also download the entire set of scene files and the other files they reference (textures, objects, particle datasets, and voxel datasets) through the "Downloading examples from the book" page.

The zip-compressed file contains within it other zip files; the fully uncompressed set of files is quite large (due primarily to the voxel data files from Chapter 23). The filenames of the compressed files all end with a .zip extension.

You can also download individual scenes from the links below. The additional files referenced by the scenes are listed at the beginning of each chapter. Several of the files (the textures, for example) are used throughout the book, but they are listed for each chapter in which they are used for reference.

Part 1: Structure

Chapter 4: Shaders in the scene

Part 2: Color

Chapter 5: A single color
Chapter 6: Color from orientation
Chapter 7: Color from position
Chapter 8: The transparency of a surface
Chapter 9: Color from functions
Chapter 10: The color of edges

Part 3: Light

Chapter 11: Lights
Chapter 12: Light on a surface
Chapter 13: Shadows
Chapter 14: Reflection
Chapter 15: Refraction
Chapter 16: Light from other surfaces

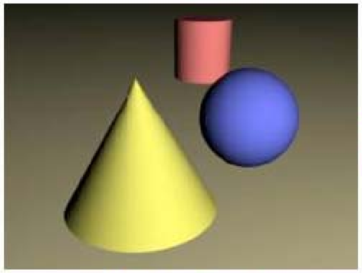
Part 4: Shape

Chapter 11: Lights

http://www.writingshaders.com/scene_catalog_files/chapter_11.html

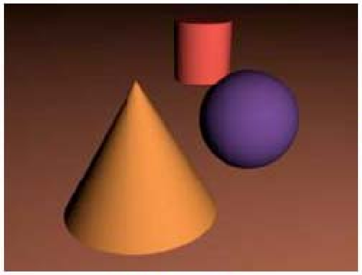
Chapter 11: Lights

Previous *Next* *Chapters* *Home*



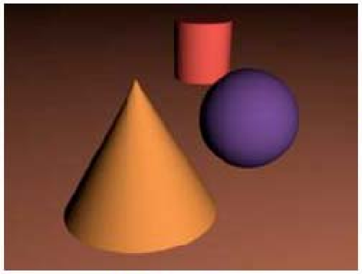
lambert
point_light

lights_1.mi



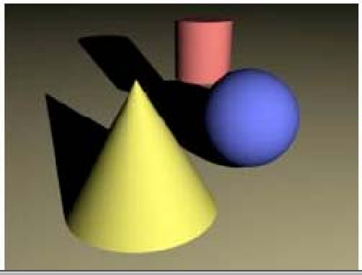
lambert
point_light

lights_2.mi

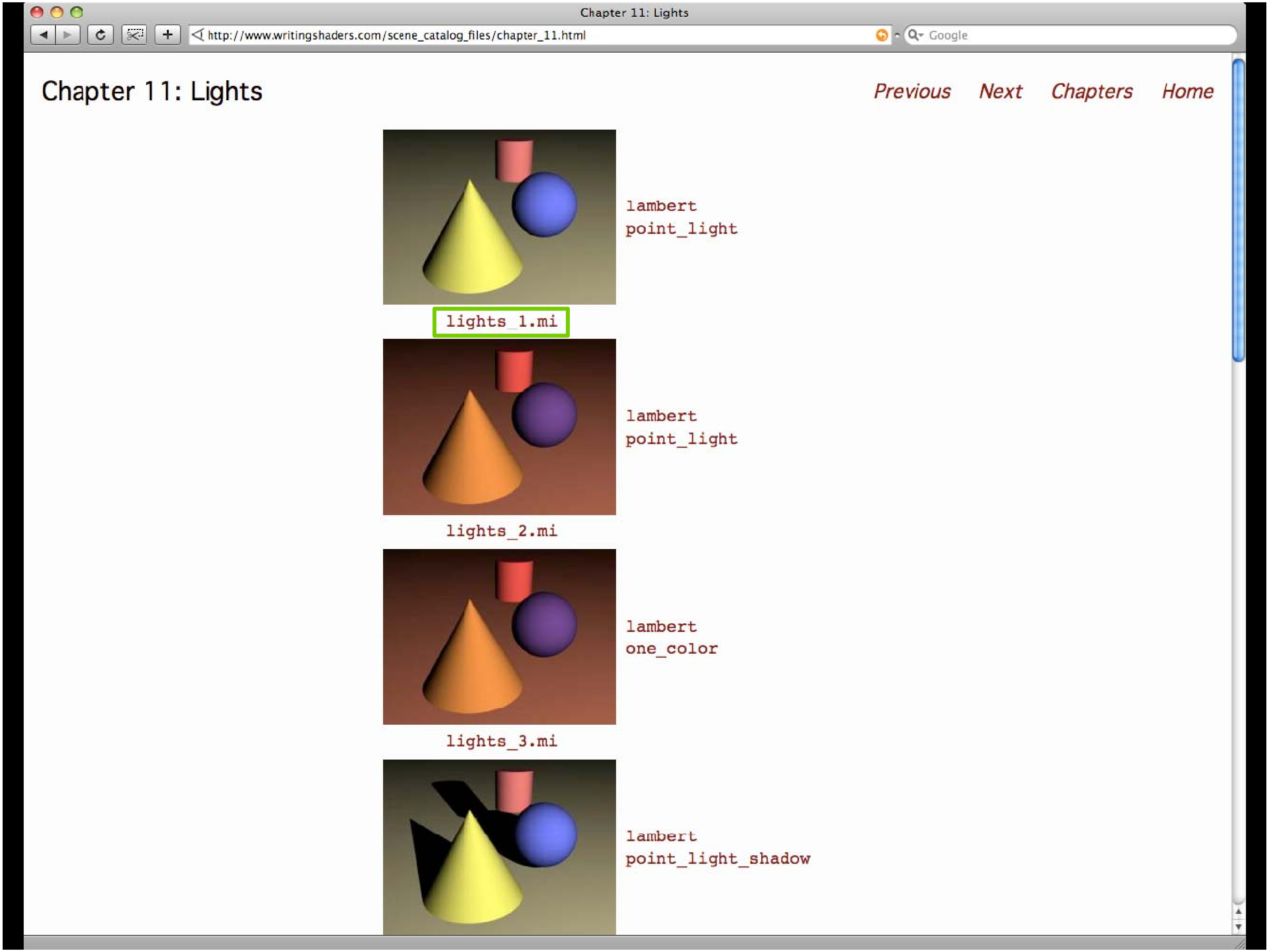


lambert
one_color

lights_3.mi



lambert
point_light_shadow


Chapter 11: Lights
http://www.writingshaders.com/scene_catalog_files/chapter_11.html
Google
Chapter 11: Lights
Previous
Next
Chapters
Home
lambert
point_light
lights_1.mi
lambert
point_light
lights_2.mi
lambert
one_color
lights_3.mi
lambert
point_light_shadow

http://www.writingshaders.com/scene_catalog_files/scene_catalog_mifiles/lights_1.mi

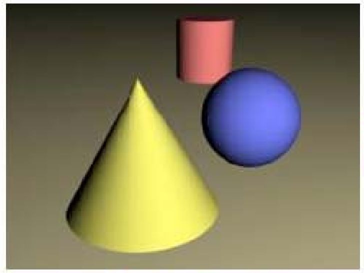
```
# mental ray scene file from "Writing mental ray shaders"
# http://www.writingshaders.com/scene_catalog.html

verbose on

link "lambert.so"
$include "lambert.mi"
link "point_light.so"
$include "point_light.mi"

options "opt"
    object space
    contrast .1 .1 .1 1
    samples 0 2
end options


light "white_light"
    "point_light" ()
    origin 2 2 5
end light

instance "light_instance"
    "white_light"
end instance

material "brown" opaque
    "lambert" (
        "diffuse" 1 .95 .7,
        "lights" ["light_instance"]
    )
end material

material "red" opaque
    "lambert" (
        "diffuse" 1 .4 .4,
        "lights" ["light_instance"]
    )
end material

material "yellow" opaque
    "lambert" (
        "diffuse" 1 1 .4,
        "lights" ["light_instance"]
    )
end material

material "blue" opaque
    "lambert" (
        "diffuse" .4 .4 1,
        "lights" ["light_instance"]
    )
```

Chapter 11: Lights

http://www.writingshaders.com/scene_catalog_files/chapter_11.html

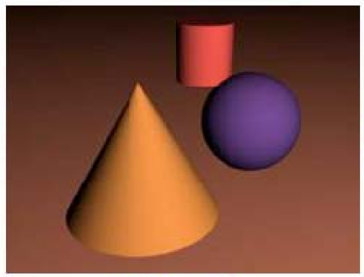
Chapter 11: Lights

Previous *Next* *Chapters* *Home*



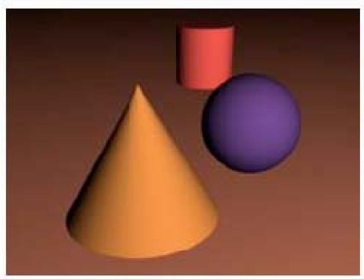
lambert
point_light

lights_1.mi



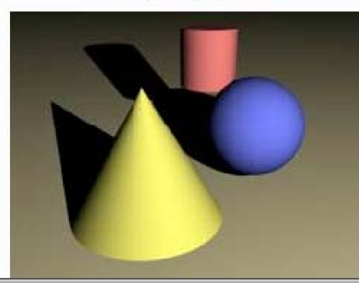
lambert
point_light

lights_2.mi

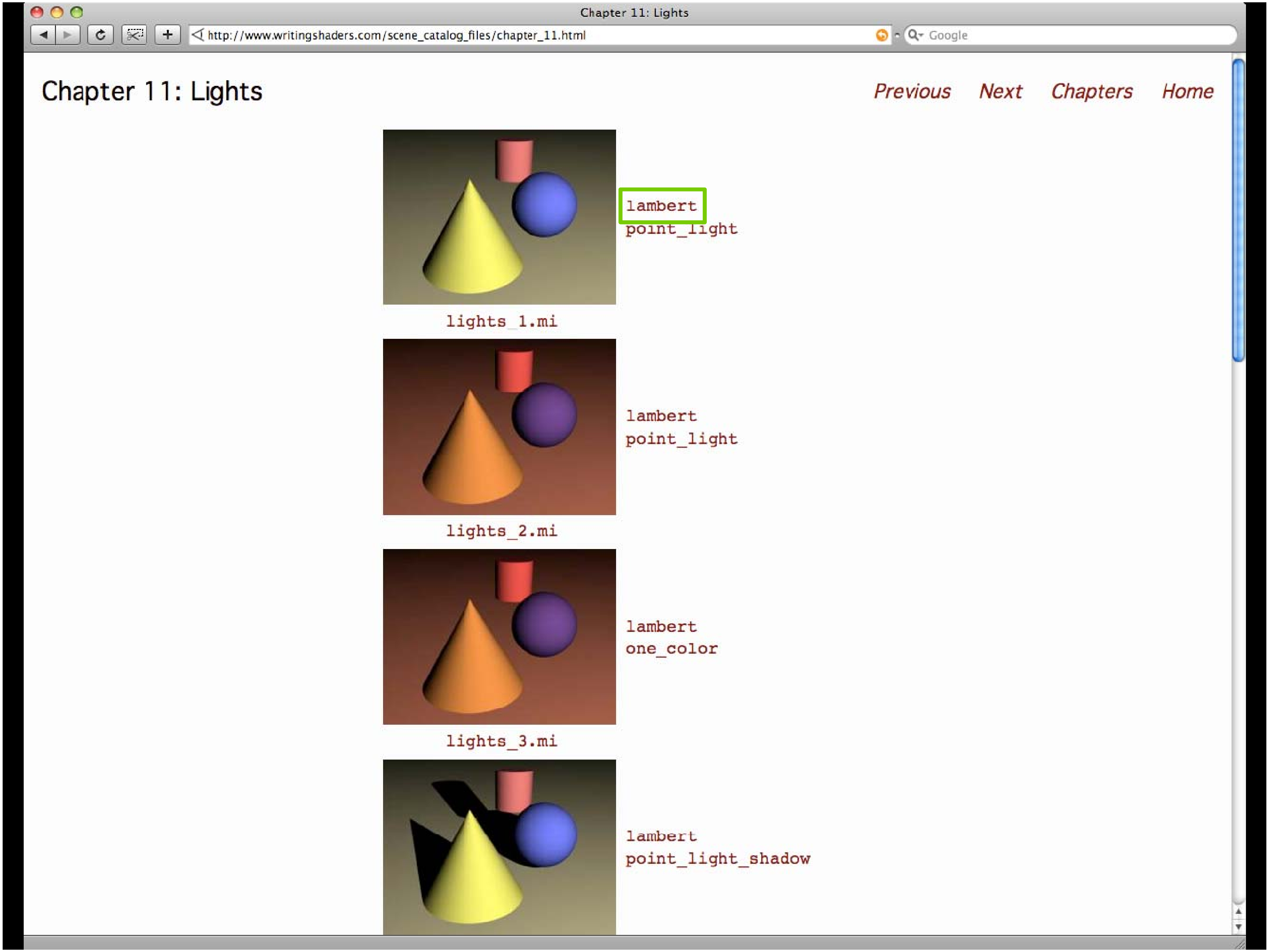


lambert
one_color

lights_3.mi



lambert
point_light_shadow


Chapter 11: Lights
http://www.writingshaders.com/scene_catalog_files/chapter_11.html
Google
Chapter 11: Lights
Previous
Next
Chapters
Home
lambert
point_light
lights_1.mi
lambert
point_light
lights_2.mi
lambert
one_color
lights_3.mi
lambert
point_light_shadow

Shader source for lambert

http://www.writingshaders.com/shader_pages/lambert.html

Shader lambert

Scenes Home

Click on the filename to display or download the file.

---

lambert.mi

```
declare shader
    color "lambert" (
        color "ambient" default 0 0 0,
        color "diffuse" default 1 1 1,
        array light "lights" )
    version 1
    apply material
end declare
```

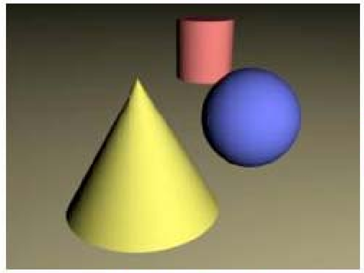
---

lambert.c

```
#include "shader.h"
#include "miaux.h"

struct lambert {
    miColor ambient;
    miColor diffuse;
    int     i_light;
    int     n_light;
    miTag   light[1];
};

DLLEXPORT
int lambert_version(void) { return 1; }

DLLEXPORT
miBoolean lambert (
    miColor *result, miState *state, struct lambert *params  )
{
    int i, light_count, light_sample_count;
    miColor sum, light_color;
    miScalar dot_nl;
    miTag *light;

    miColor *diffuse  = mi_eval_color(&params->diffuse);
    miaux_light_array(&light, &light_count, state,
                      &params->i_light, &params->n_light, params->light);
    *result = *mi_eval_color(&params->ambient);

    for (i = 0; i < light_count; i++, light++) {
```

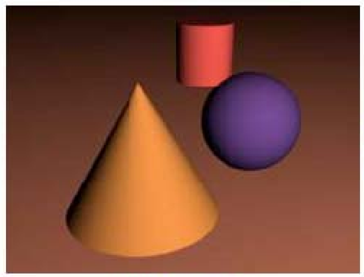
Chapter 11: Lights

Previous *Next* *Chapters* *Home*



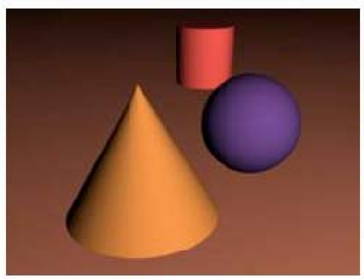
lambert
point_light

lights_1.mi



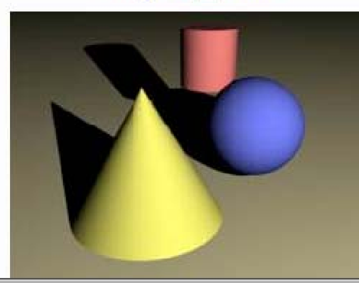
lambert
point_light

lights_2.mi

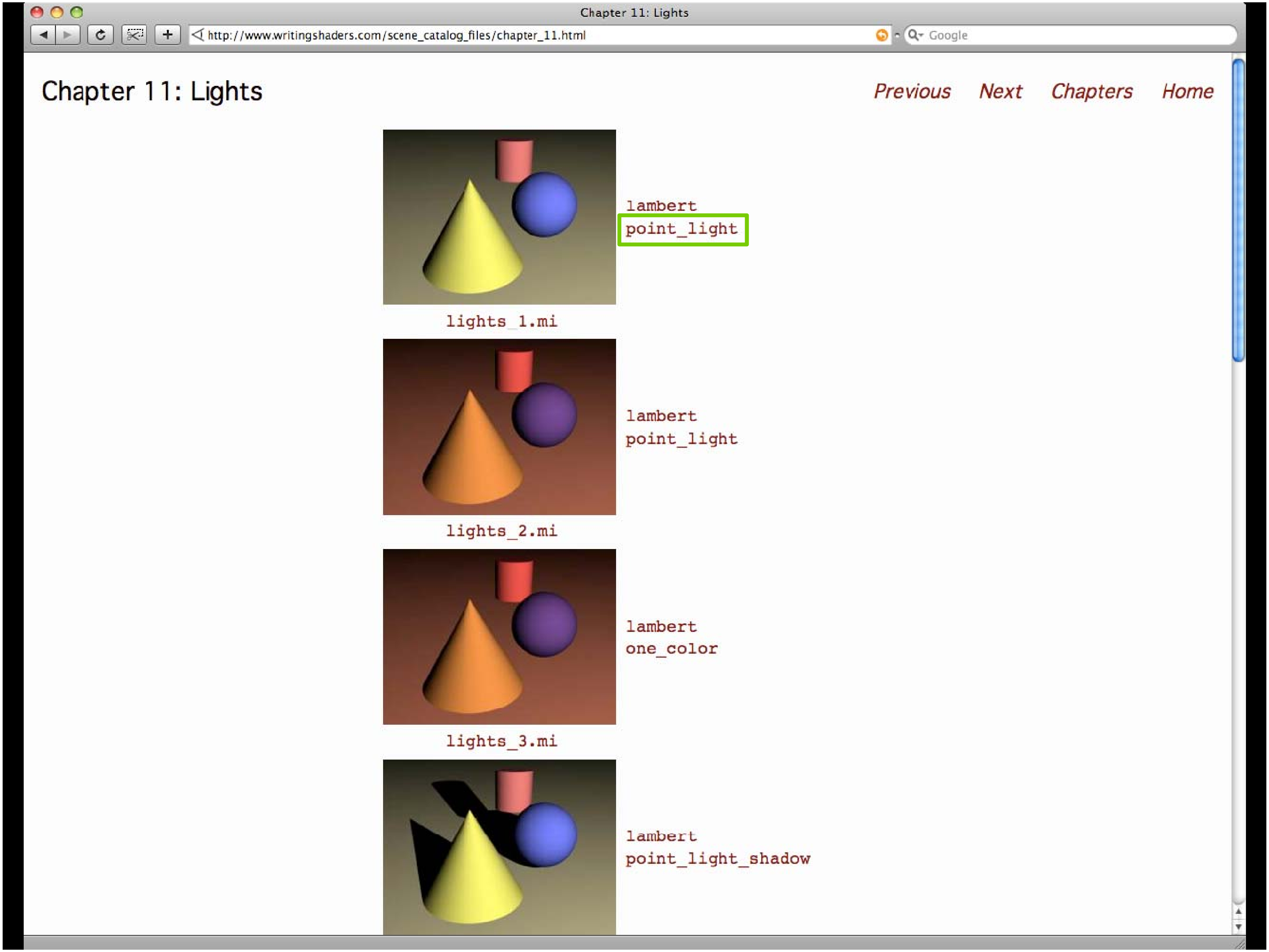


lambert
one_color

lights_3.mi



lambert
point_light_shadow


Chapter 11: Lights
http://www.writingshaders.com/scene_catalog_files/chapter_11.html
Google
Chapter 11: Lights
Previous
Next
Chapters
Home
lambert
point_light
lights_1.mi
lambert
point_light
lights_2.mi
lambert
one_color
lights_3.mi
lambert
point_light_shadow

Shader source for point_light

http://www.writingshaders.com/shader_pages/point_light.html

Google

Shader point_light

Scenes Home

Click on the filename to display or download the file.

---

point_light.mi

```
declare shader
    color "point_light" (
        color "light_color" default 1 1 1 )
    version 1
    apply light
end declare
```

---

point_light.c

```
#include "shader.h"

struct point_light {
    miColor light_color;
};

DLLEXPORT
int point_light_version(void) { return 1; }

DLLEXPORT
miBoolean point_light (
    miColor *result, miState *state, struct point_light *params )
{
    *result = *mi_eval_color(&params->light_color);
    return miTRUE;
}
```

---

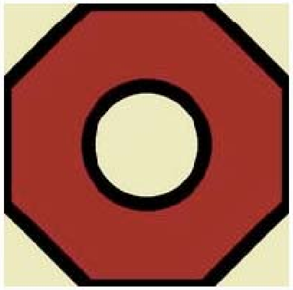
22 April 2008 23:40:03

Chapter 14: Reflection

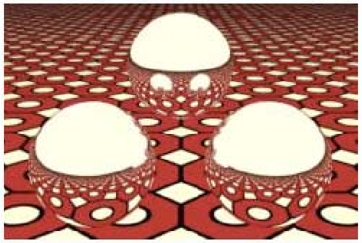
http://www.writingshaders.com/scene_catalog_files/chapter_14.html

Chapter 14: Reflection

Previous Next Chapters Home

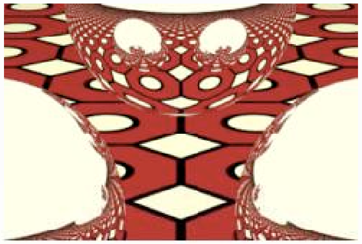


Texture: octatile.tif



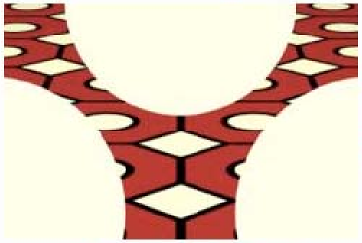
lambert
one_color
specular_reflection
texture_uv
point_light

reflection_1.mi



lambert
one_color
specular_reflection
texture_uv
point_light

reflection_2.mi



lambert
one_color
specular_reflection
texture_uv
point_light

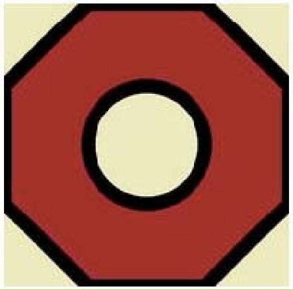
reflection_3.mi

Chapter 14: Reflection

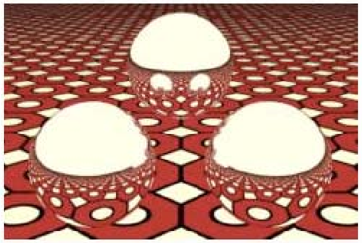
http://www.writingshaders.com/scene_catalog_files/chapter_14.html

Chapter 14: Reflection

Previous *Next* *Chapters* *Home*

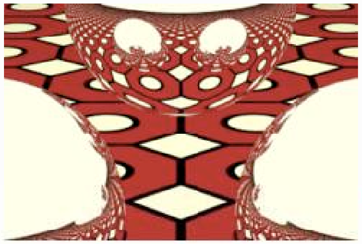


Texture: octatile.tif



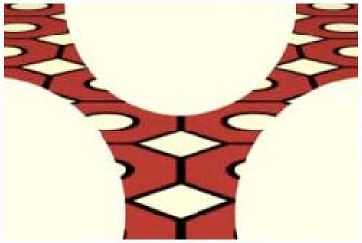
lambert
one_color
specular_reflection
texture_uv
point_light

reflection_1.mi



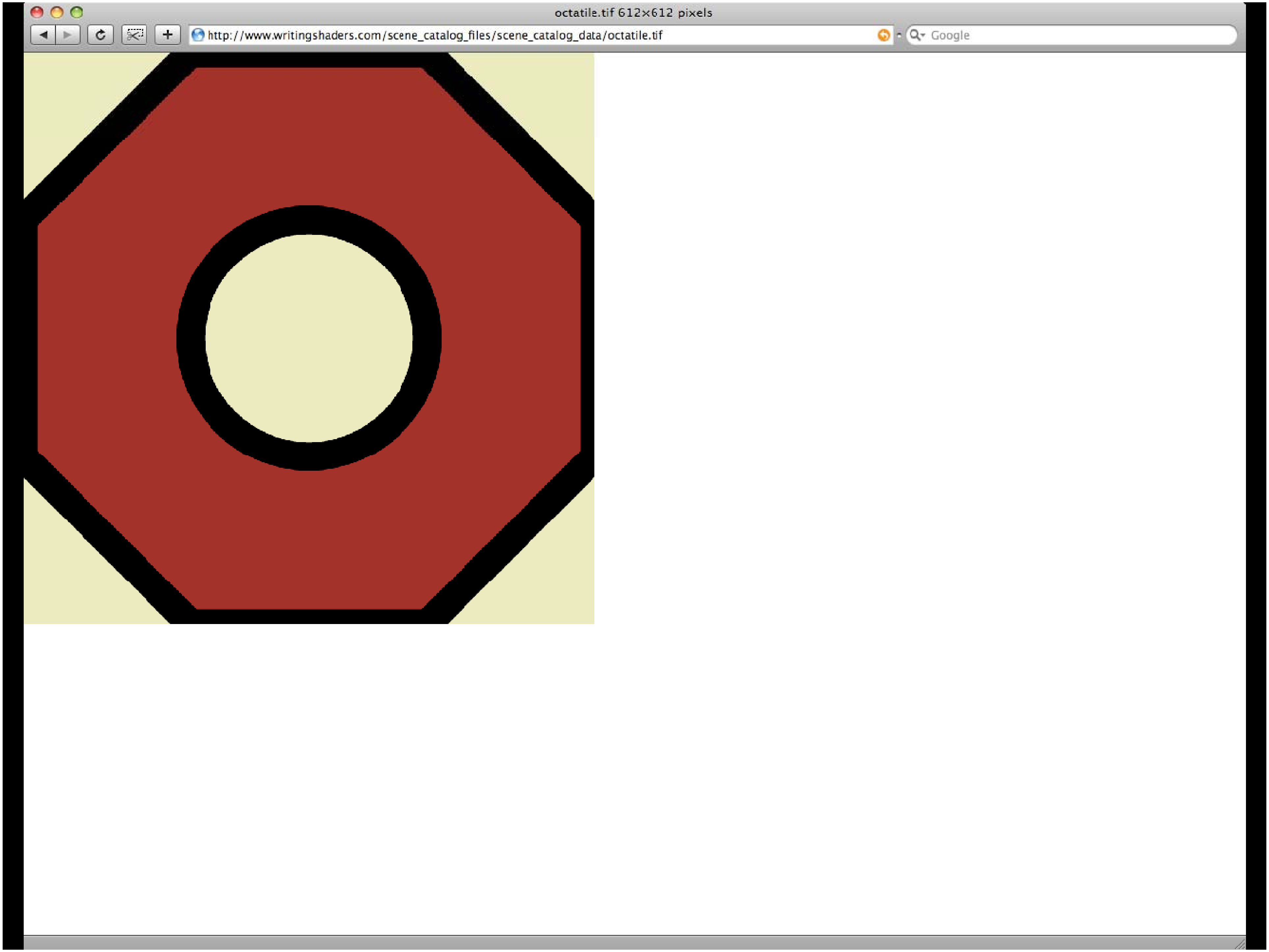
lambert
one_color
specular_reflection
texture_uv
point_light

reflection_2.mi



lambert
one_color
specular_reflection
texture_uv
point_light

reflection_3.mi

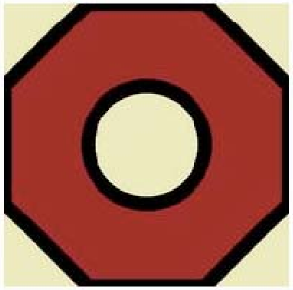

octatile.tif 612×612 pixels
http://www.writingshaders.com/scene_catalog_files/scene_catalog_data/octatile.tif
Google

Chapter 14: Reflection

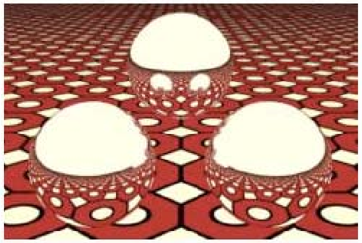
http://www.writingshaders.com/scene_catalog_files/chapter_14.html

Chapter 14: Reflection

Previous *Next* *Chapters* *Home*

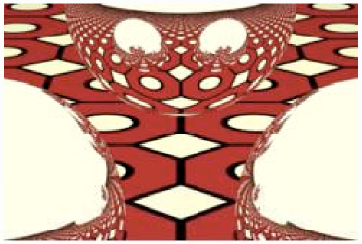


Texture: octatile.tif



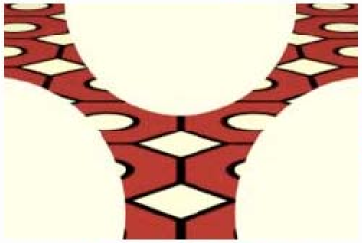
lambert
one_color
specular_reflection
texture_uv
point_light

reflection_1.mi



lambert
one_color
specular_reflection
texture_uv
point_light

reflection_2.mi



lambert
one_color
specular_reflection
texture_uv
point_light

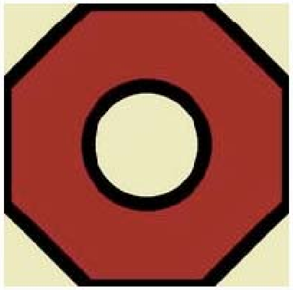
reflection_3.mi

Chapter 14: Reflection

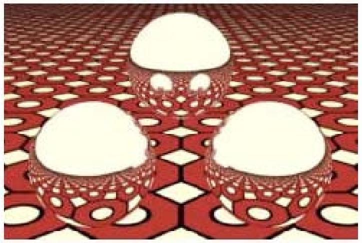
http://www.writingshaders.com/scene_catalog_files/chapter_14.html

Chapter 14: Reflection

Previous *Next* *Chapters* *Home*

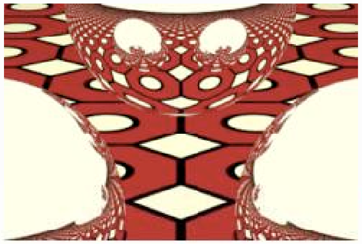


Texture: octatile.tif



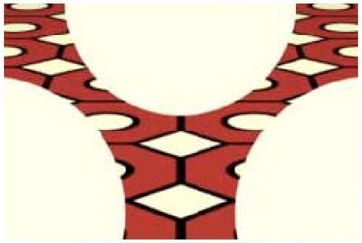
lambert
one_color
specular_reflection
texture_uv
point_light

reflection_1.mi



lambert
one_color
specular_reflection
texture_uv
point_light

reflection_2.mi



lambert
one_color
specular_reflection
texture_uv
point_light

reflection_3.mi

Website support for the book's software

The "Reference" section contains slides for use in teaching and links for the book's bibliography.

Teaching materials

http://www.writingshaders.com/teaching.html

Teaching materials

Home

Slides for lectures

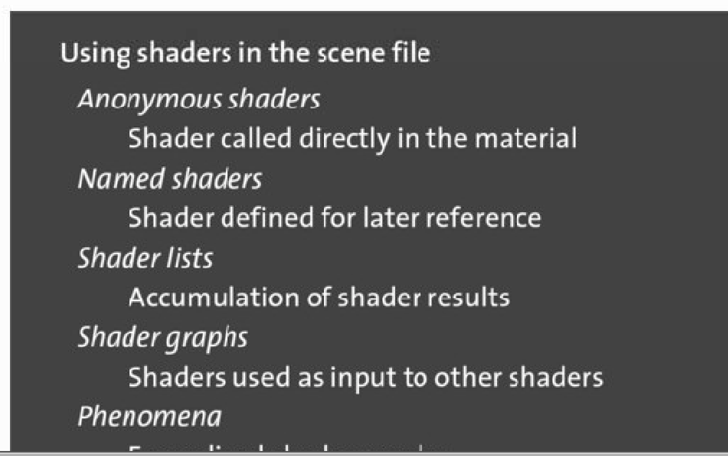
If you are a teacher and would like to use *Writing mental ray shaders* in the classroom, I have reformatted the book's diagrams and code as a set of PDF files that can be used in classroom lectures.

Introduction	`WMRS_part0_introduction_april_2008.pdf`
Structure	`WMRS_part1_structure_april_2008.pdf`
Color	`WMRS_part2_color_april_2008.pdf`
Light	`WMRS_part3_light_april_2008.pdf`
Shape	`WMRS_part4_shape_april_2008.pdf`
Space	`WMRS_part5_space_april_2008.pdf`
Image	`WMRS_part6_image_april_2008.pdf`

The lectures use the shaders and scene files from the book as well as some additional material you can download.

Additional lecture resources	`WMRS_lectures_extra_april_2008.zip`

Examples of the lecture slides



Teaching materials

http://www.writingshaders.com/teaching.html

Examples of the lecture slides

Using shaders in the scene file

Anonymous shaders
- Shader called directly in the material

Named shaders
- Shader defined for later reference

Shader lists
- Accumulation of shader results

Shader graphs
- Shaders used as input to other shaders

Phenomena
- Formalized shader graphs

The environment of the scene

Environment shaders for cameras and objects

```
shader "red_sunset"
   "color_ramp" (
      "colors" [ 0.1  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.0  0.5,
                 1.0  1.0  0.0  0.51,
                 1.0  0.2  0.0  0.55,
                 0.1  0.2  0.6  0.7,
                 0.12 0.05 0.2  1.0 ] )

shader "pink_sunset"
   "color_ramp" (
      "colors" [ 0.1  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.1  0.50,
                 1.0  1.0  0.8  0.51,
                 1.0  0.6  0.4  0.55,
                 0.1  0.2  0.6  0.7,
                 0.05 0.1  0.2  1.0 ] )

camera "cam"
    output "rgba" "tif" "environment_6.tif"
    focal 1.5
    aperture 1.5
    aspect 1
    resolution 300 300
    environment
        = "red_sunset"
end camera

material "corinthian_material"
    "specular_reflection" ()
    environment = "pink_sunset"
end material
```

Different environment shaders used for the camera and object

Rendering image components

Definition and use of frame buffers

Teaching materials

http://www.writingshaders.com/teaching.html

Formalized shader graphs

The environment of the scene

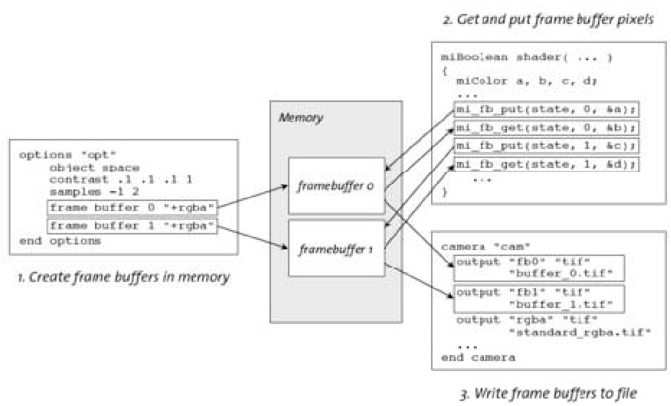
Environment shaders for cameras and objects

```
shader "red_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.0  0.5,
                 1.0  1.0  0.0  0.51,
                 1.0  0.2  0.0  0.55,
                 0.1  0.2  0.6  0.7,
                 0.12 0.05 0.2  1.0 ] )

shader "pink_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.1  0.50,
                 1.0  1.0  0.8  0.51,
                 1.0  0.6  0.4  0.55,
                 0.1  0.2  0.6  0.7,
                 0.05 0.1  0.2  1.0 ] )

camera "cam"
    output "rgba" "tif" "environment_6.tif"
    focal 1.5
    aperture 1.5
    aspect 1
    resolution 300 300
    environment
        = "red_sunset"
end camera

material "corinthian_material"
    "specular_reflection" ()
    environment = "pink_sunset"
end material
```

Different environment shaders used for the camera and object

Rendering image components

Definition and use of frame buffers



Interaction of the scene file and shaders in the use of frame buffers

Teaching materials

http://www.writingshaders.com/teaching.html

Formalized shader graphs

The environment of the scene

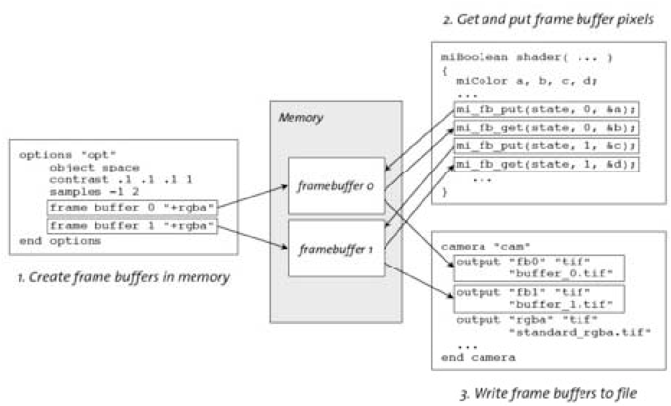
Environment shaders for cameras and objects

```
shader "red_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.0  0.5,
                 1.0  1.0  0.0  0.51,
                 1.0  0.2  0.0  0.55,
                 0.1  0.2  0.6  0.7,
                 0.12 0.05 0.2  1.0 ] )

shader "pink_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.1  0.50,
                 1.0  1.0  0.8  0.51,
                 1.0  0.6  0.4  0.55,
                 0.1  0.2  0.6  0.7,
                 0.05 0.1  0.2  1.0 ] )

camera "cam"
    output "rgba" "tif" "environment_6.tif"
    focal 1.5
    aperture 1.5
    aspect 1
    resolution 300 300
    environment
        = "red_sunset"
end camera

material "corinthian_material"
    "specular_reflection" ()
    environment = "pink_sunset"
end material
```

Different environment shaders used for the camera and object

Rendering image components

Definition and use of frame buffers



Interaction of the scene file and shaders in the use of frame buffers

Teaching materials

http://www.writingshaders.com/teaching.html

Formalized shader graphs

The environment of the scene

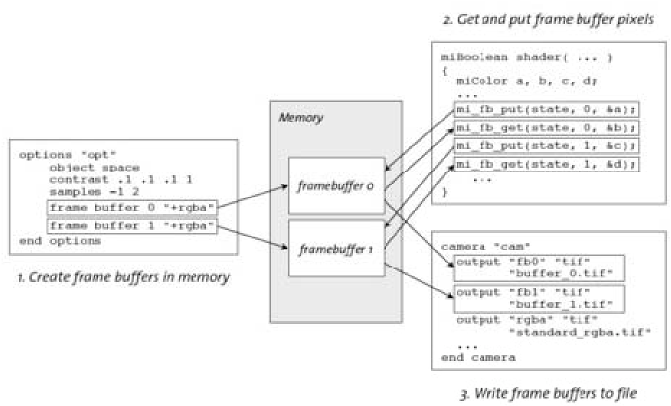
Environment shaders for cameras and objects

```
shader "red_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.0  0.5,
                 1.0  1.0  0.0  0.51,
                 1.0  0.2  0.0  0.55,
                 0.1  0.2  0.6  0.7,
                 0.12 0.05 0.2  1.0 ] )

shader "pink_sunset"
   "color_ramp" (
      "colors" [ 0.3  0.2  0.1  0.0,
                 0.1  0.05 0.0  0.49,
                 0.4  0.3  0.1  0.50,
                 1.0  1.0  0.8  0.51,
                 1.0  0.6  0.4  0.55,
                 0.1  0.2  0.6  0.7,
                 0.05 0.1  0.2  1.0 ] )

camera "cam"
    output "rgba" "tif" "environment_6.tif"
    focal 1.5
    aperture 1.5
    aspect 1
    resolution 300 300
    environment
       = "red_sunset"
end camera

material "corinthian_material"
    "specular_reflection" ()
    environment = "pink_sunset"
end material
```

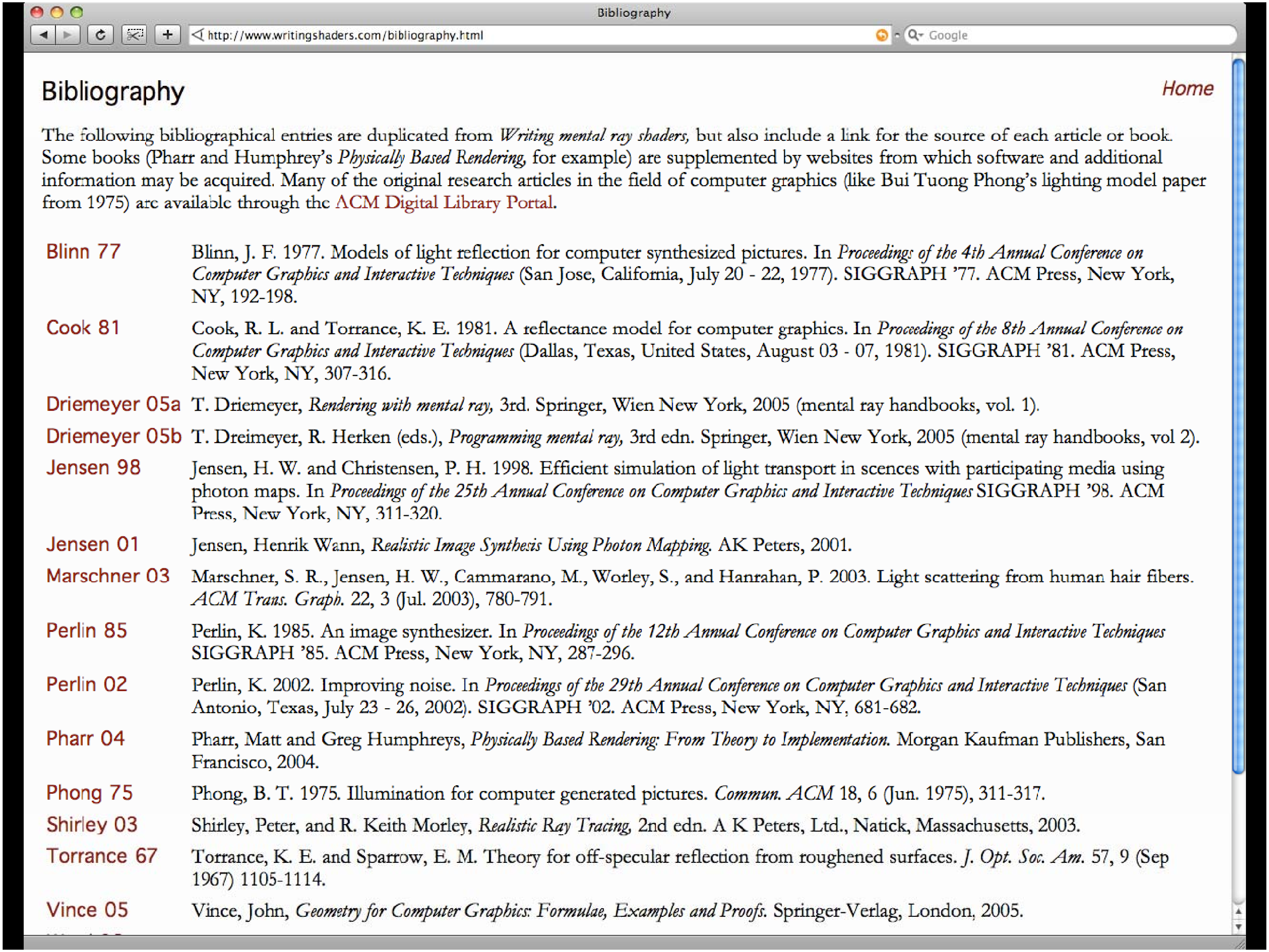
Different environment shaders used for the camera and object

Rendering image components

Definition and use of frame buffers



Interaction of the scene file and shaders in the use of frame buffers



Bibliography

http://www.writingshaders.com/bibliography.html

Google

Bibliography

Home

The following bibliographical entries are duplicated from *Writing mental ray shaders,* but also include a link for the source of each article or book. Some books (Pharr and Humphrey's *Physically Based Rendering,* for example) are supplemented by websites from which software and additional information may be acquired. Many of the original research articles in the field of computer graphics (like Bui Tuong Phong's lighting model paper from 1975) are available through the ACM Digital Library Portal.

Blinn 77	Blinn, J. F. 1977. Models of light reflection for computer synthesized pictures. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques* (San Jose, California, July 20 - 22, 1977). SIGGRAPH '77. ACM Press, New York, NY, 192-198.
Cook 81	Cook, R. L. and Torrance, K. E. 1981. A reflectance model for computer graphics. In *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques* (Dallas, Texas, United States, August 03 - 07, 1981). SIGGRAPH '81. ACM Press, New York, NY, 307-316.
Driemeyer 05a	T. Driemeyer, *Rendering with mental ray,* 3rd. Springer, Wien New York, 2005 (mental ray handbooks, vol. 1).
Driemeyer 05b	T. Dreimeyer, R. Herken (eds.), *Programming mental ray,* 3rd edn. Springer, Wien New York, 2005 (mental ray handbooks, vol 2).
Jensen 98	Jensen, H. W. and Christensen, P. H. 1998. Efficient simulation of light transport in scences with participating media using photon maps. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* SIGGRAPH '98. ACM Press, New York, NY, 311-320.
Jensen 01	Jensen, Henrik Wann, *Realistic Image Synthesis Using Photon Mapping.* AK Peters, 2001.
Marschner 03	Marschner, S. R., Jensen, H. W., Cammarano, M., Worley, S., and Hanrahan, P. 2003. Light scattering from human hair fibers. *ACM Trans. Graph.* 22, 3 (Jul. 2003), 780-791.
Perlin 85	Perlin, K. 1985. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques* SIGGRAPH '85. ACM Press, New York, NY, 287-296.
Perlin 02	Perlin, K. 2002. Improving noise. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques* (San Antonio, Texas, July 23 - 26, 2002). SIGGRAPH '02. ACM Press, New York, NY, 681-682.
Pharr 04	Pharr, Matt and Greg Humphreys, *Physically Based Rendering: From Theory to Implementation.* Morgan Kaufman Publishers, San Francisco, 2004.
Phong 75	Phong, B. T. 1975. Illumination for computer generated pictures. *Commun. ACM* 18, 6 (Jun. 1975), 311-317.
Shirley 03	Shirley, Peter, and R. Keith Morley, *Realistic Ray Tracing,* 2nd edn. A K Peters, Ltd., Natick, Massachusetts, 2003.
Torrance 67	Torrance, K. E. and Sparrow, E. M. Theory for off-specular reflection from roughened surfaces. *J. Opt. Soc. Am.* 57, 9 (Sep 1967) 1105-1114.
Vince 05	Vince, John, *Geometry for Computer Graphics: Formulae, Examples and Proofs.* Springer-Verlag, London, 2005.

Illumination for computer generated pictures

http://portal.acm.org/citation.cfm?id=360839&coll=Portal&dl=ACM&CFID=11726875&CFTOKEN=33287191

Illumination for computer generated pictures

Full text Pdf (1.65 MB)

Source Communications of the ACM archive
Volume 18 , Issue 6 (June 1975) table of contents
Pages: 311 - 317
Year of Publication: 1975
ISSN:0001-0782

Author Bui Tuong Phong Stanford Univ., Stanford, CA

Publisher ACM New York, NY, USA

DOI Bookmark: Use this link to bookmark this Article: http://doi.acm.org/10.1145/360825.360839
What is a DOI?

ABSTRACT

The quality of computer generated images of three-dimensional scenes depends on the shading technique used to paint the objects on the cathode-ray tube screen. The shading algorithm itself depends in part on the method for modeling the object, which also determines the hidden surface algorithm. The various methods of object modeling, shading, and hidden surface removal are thus strongly interconnected. Several shading techniques corresponding to different methods of object modeling and the related hidden surface algorithms are presented here. Human visual perception and the fundamental laws of optics are considered in the development of a shading rule that provides better quality and increased realism in generated images.

REFERENCES

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

A K Peters, Ltd. – Realistic Ray Tracing

http://www.akpeters.com/product.asp?ProdCode=1985

SHOPPING CART

Your cart is empty.

View Cart

QUICK SEARCH

Advanced Search

Search Tips

Full-text book search

powered by Google Books

CATALOG

- New & Upcoming
- Computer Science
- Popular Science
- Mathematics
- Robotics
- Featured
- Professional
- Research
- Textbooks
- Journals
- Videos

VeriSign Secured VERIFY

A K PETERS LTD

Publishers of Science and Technology

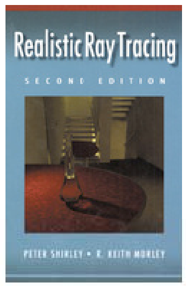
Home | Contact Us | Help
My Account | My Wish List
Newsletter Archive

NEWS & REVIEWS | ORDERING | ABOUT
FOR AUTHORS | FOR INT'L CUSTOMERS | FOR EDUCATORS

If you are a returning customer, please sign in. If you are a new customer, please register.

REALISTIC RAY TRACING

SECOND EDITION



by Peter Shirley, R. Keith Morley

Price: $49.00

Availability: Temporarily out of stock.

You may backorder this item. We will ship it when it becomes available.

Add to Shopping Cart

Summary

Concentrating on the "nuts and bolts" of writing ray tracing programs, this new and revised edition emphasizes practical and implementation issues and takes the reader through all the details needed to write a modern rendering system.

Most importantly, the book adds many C++ code segments, and adds new details to provide the reader with a better intuitive understanding of ray tracing algorithms.

Details

ISBN: 978-1-56881-198-7
Year: 2003
Format: Hardcover
Pages: 235
Web Site:
http://www.cs.utah.edu/~shirley/

Search Inside

Search within this book:

go

Browse the text of this book

powered by Google

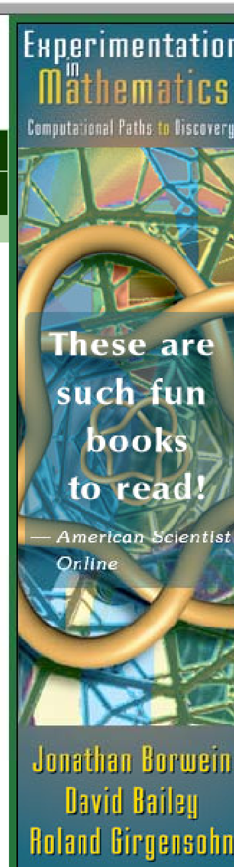
Recommendations

Customers who bought this item also bought:

- COLLADA: Sailing the Gulf of 3D Digital Content Creation
- Dungeons and Desktops: The History of Computer Role-Playing Games
- The Incompleteness Phenomenon
- The Game Programmer's Guide to Torque: Under the Hood of the Torque Game Engine
- Morphs, Mallards, and Montages: Computer-Aided Imagination

Browse Catalog

- Computer Science > Computer Graphics
- Textbooks > Computer Science
- Computer Science > Game Programming and Design



Physically Based Rendering: From Theory to Implementation

http://www.pbrt.org/

MATT PHARR • GREG HUMPHREYS

PHYSICALLY BASED RENDERING

FROM THEORY TO IMPLEMENTATION

From movies to video games, computer-rendered images are pervasive today. **Physically Based Rendering** introduces the concepts and theory of photorealistic rendering hand in hand with the source code for a sophisticated renderer. By coupling the discussion of rendering algorithms with their implementations, Matt Pharr and Greg Humphreys are able to reveal many of the details and subtleties of these algorithms. But this book goes further; it also describes the design strategies involved with building real systems—there is much more to writing a good renderer than stringing together a set of fast algorithms. For example, techniques for high-quality antialiasing must be considered from the start, as they have implications throughout the system. The rendering system described in this book is itself highly readable, written in a style called *literate programming* that mixes text describing the system with the code that implements it. Literate programming gives a gentle introduction to working with programs of this size. This lucid pairing of text and code offers the most complete and in-depth book available for understanding, designing, and building physically realistic rendering systems.

For a preview, download Chapter 7, "Sampling and Reconstruction".

News

June 28, 2008
: luxrender, a GPL Open Source fork of pbrt, has released version 0.5 with many new features, including full spectral rendering, bidirectional path tracing, a hierarchical material and texture system, displacement mapping, and much more.

November 18, 2007
: Check out luxrender, a GPL Open Source fork of pbrt. The project seems to be off to a strong start and there are some amazing images in the gallery.

July 4, 2007
: The long-awaited 1.03 patch release of pbrt has been released. See the downloads page.

August 12, 2006
: A number of cool new renderings have been added to the gallery page.

January 9, 2006
: Mark Colbert has updated his Maya plugin that exports Maya scenes to pbrt and renders them inside Maya to both support Maya under Windows and Maya under OS X.

May 17, 2005
: A Mathematica-to-PBRT exporter has been released. See the downloads page.

April 25, 2005
: The long-delayed first patch release of the pbrt source code has been released. This release fixes a number of bugs found by readers and the authors in the first 6 months after the book's publication. See the downloads page for links to the source code and information about the fixes in this release.

April 25, 2005
: A new photon mapping plugin, implmenting many improvements to the photon mapping implementation described in the book, has been added to the plugins page.

February 14, 2005
: *Physically Based Rendering* has won an Honorable Mention in in the Computer and Information Science category from the The Professional and Scholarly Publishing Division of the Association of American Publishers Awards. The award winners in each category were selected for their unique contribution to scholarly publishing and are considered by the panel of judges to be the best of the best for 2004.

The CDROM included with the book

5

The CDROM included with the book

A fully functional version of mental ray is included on the CDROM.

Writing mental ray Shaders

Proof of Purchase

The enclosed CD-ROM contains a fully functional version of mental ray®. To claim your license, please enter the following information into the SPM® license manager form after software installation.

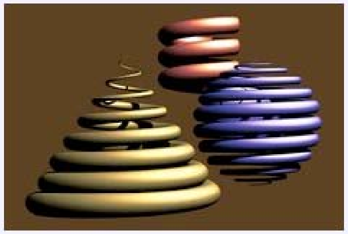
[redacted]

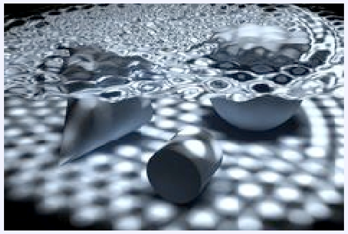
For further information please visit

http://www.mentalimages.com/licenses/

Writing mental ray shaders

file:///Volumes/mentalray/index.html





Writing mental ray shaders

A perceptual introduction

Install the fully functional mental ray® demo version

This CDROM contains a fully functional version of mental ray and a license that will allow you to run it on your computer. The installation process will copy mental ray and the SPM licensing software to your file system. If you are currently attached to the Web, a license will be requested from mental images and sent back to your computer. You will then be able to render with mental ray.

Enter the CDROM version of the book's website

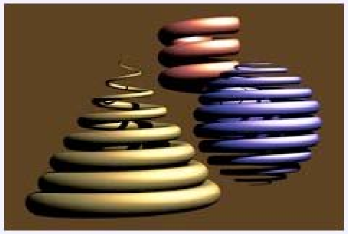
The shader source code, scene files, other data, and additional explanations are available on the book's website at http://www.writingshaders.com/. This CDROM also contains a copy of the on-line website made when the book was published. Once you've installed the software and license, you can compile the example shaders and start rendering the scene files with mental ray.

http://www.writingshaders.com/

Writing mental ray shaders

file:///Volumes/mentalray/index.html

Google

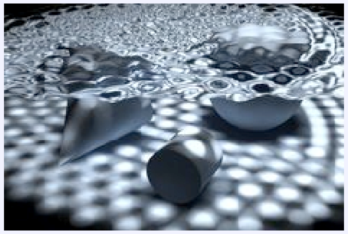


Writing mental ray shaders

A perceptual introduction

Install the fully functional mental ray® demo version

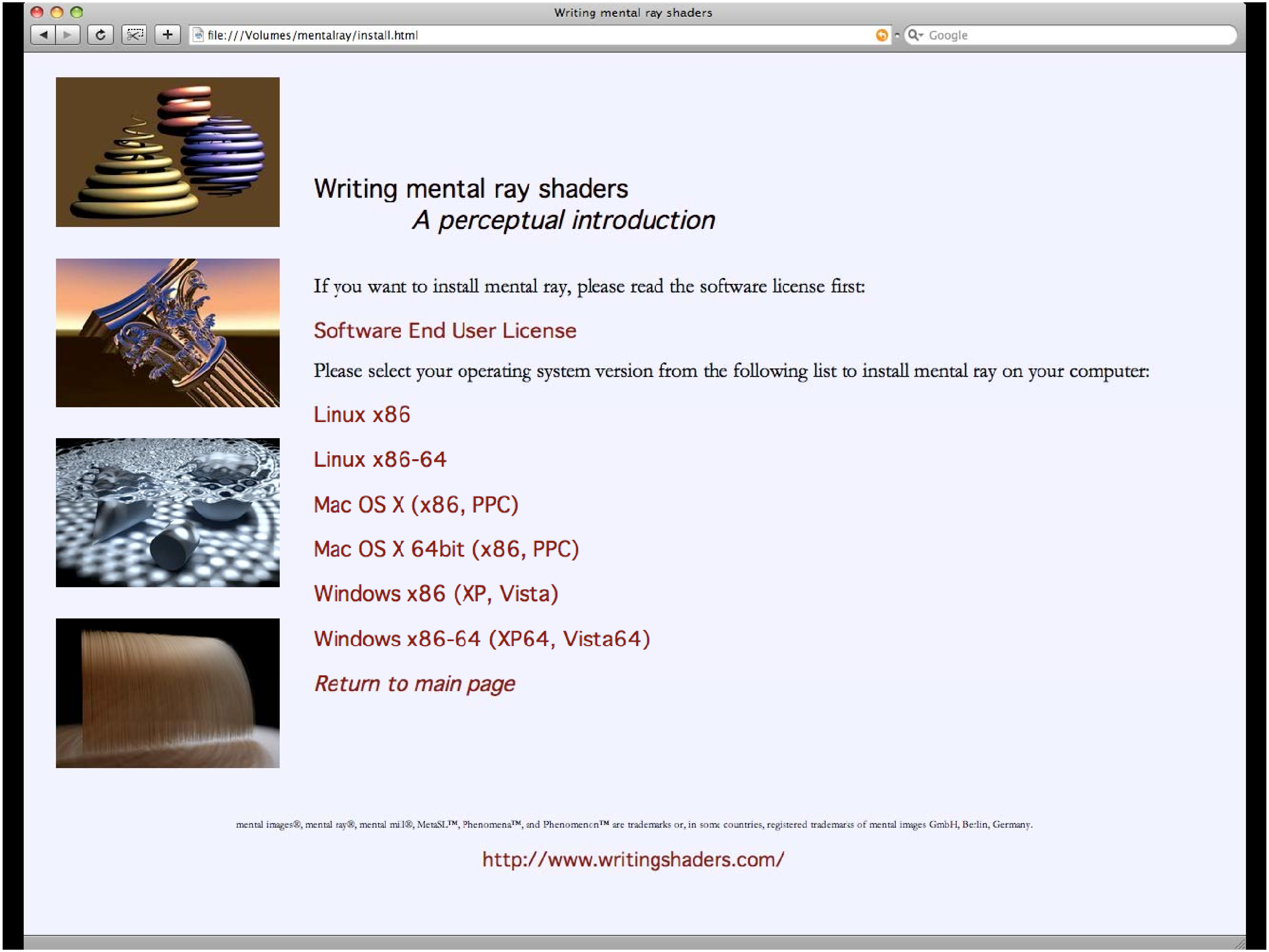
This CDROM contains a fully functional version of mental ray and a license that will allow you to run it on your computer. The installation process will copy mental ray and the SPM licensing software to your file system. If you are currently attached to the Web, a license will be requested from mental images and sent back to your computer. You will then be able to render with mental ray.



Enter the CDROM version of the book's website

The shader source code, scene files, other data, and additional explanations are available on the book's website at http://www.writingshaders.com/. This CDROM also contains a copy of the on-line website made when the book was published. Once you've installed the software and license, you can compile the example shaders and start rendering the scene files with mental ray.

http://www.writingshaders.com/


Writing mental ray shaders
file:///Volumes/mentalray/install.html
Google
Writing mental ray shaders
A perceptual introduction
If you want to install mental ray, please read the software license first:
Software End User License
Please select your operating system version from the following list to install mental ray on your computer:
Linux x86
Linux x86-64
Mac OS X (x86, PPC)
Mac OS X 64bit (x86, PPC)
Windows x86 (XP, Vista)
Windows x86-64 (XP64, Vista64)
Return to main page
mental images®, mental ray®, mental mill®, MetaSL™, Phenomena™, and Phenomenon™ are trademarks or, in some countries, registered trademarks of mental images GmbH, Berlin, Germany.
http://www.writingshaders.com/

#3?

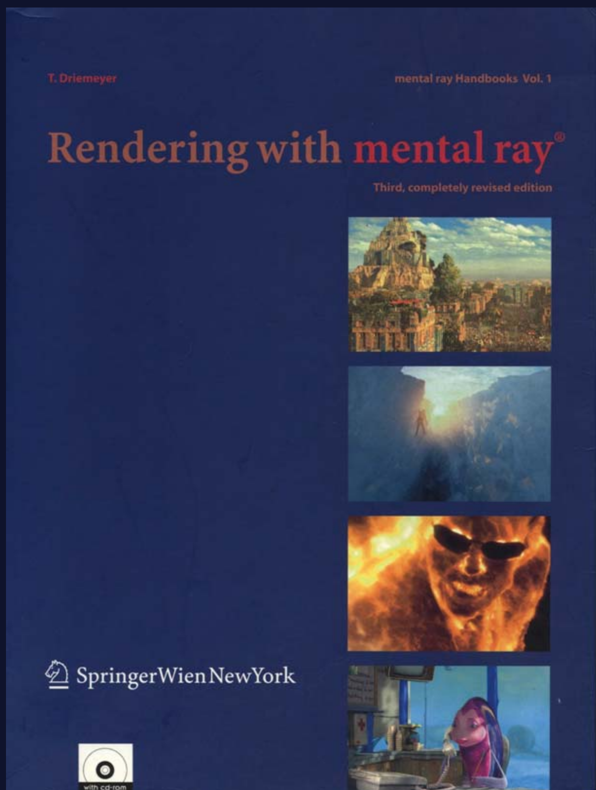
Andy Kopra

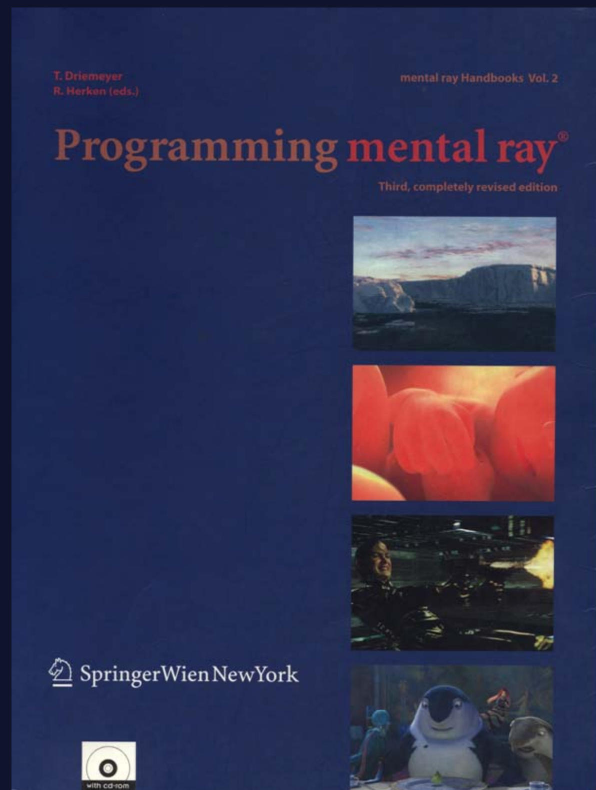
mental ray Handbooks Vol. 3

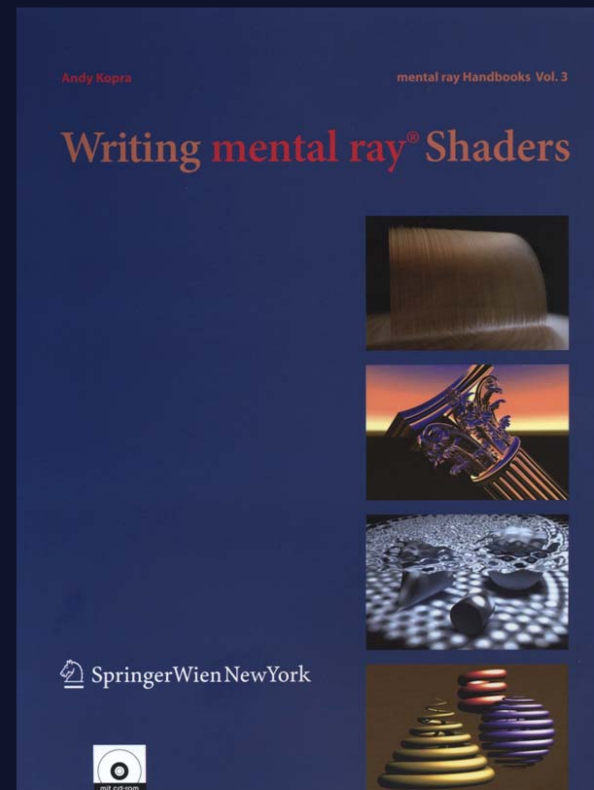
Writing mental ray® Shaders

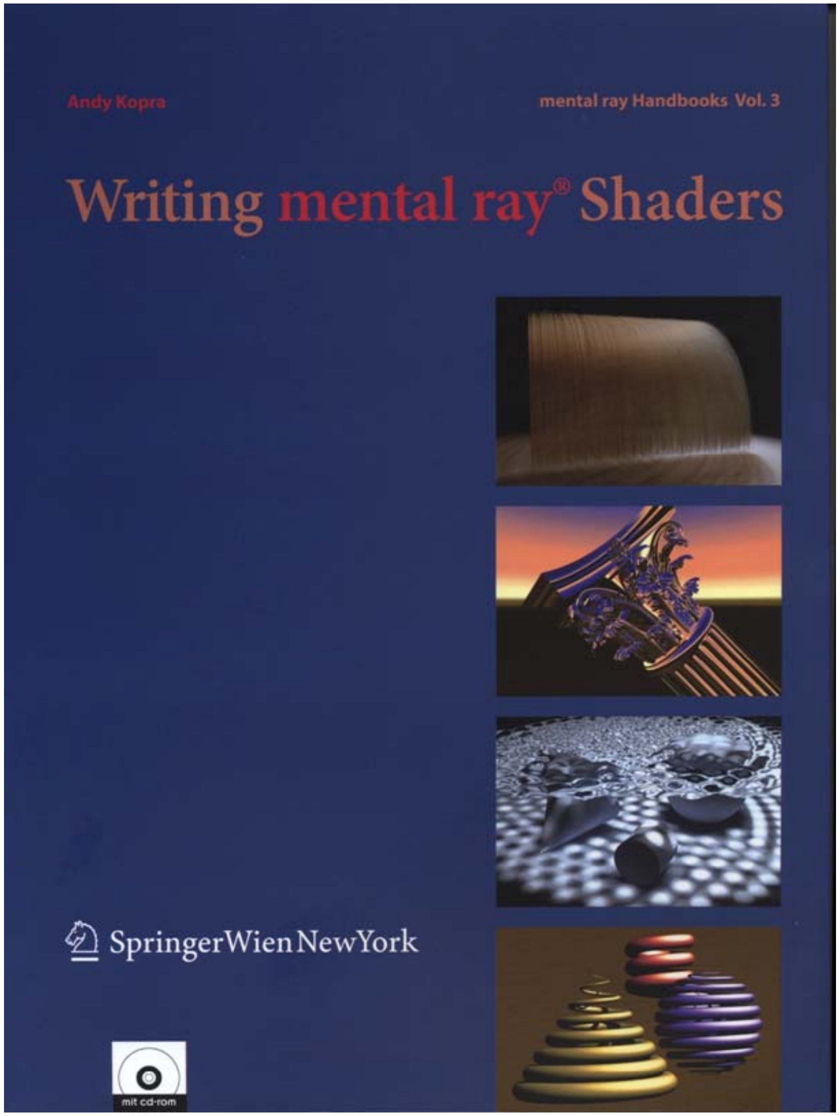
SpringerWienNewYork

mit cd-rom

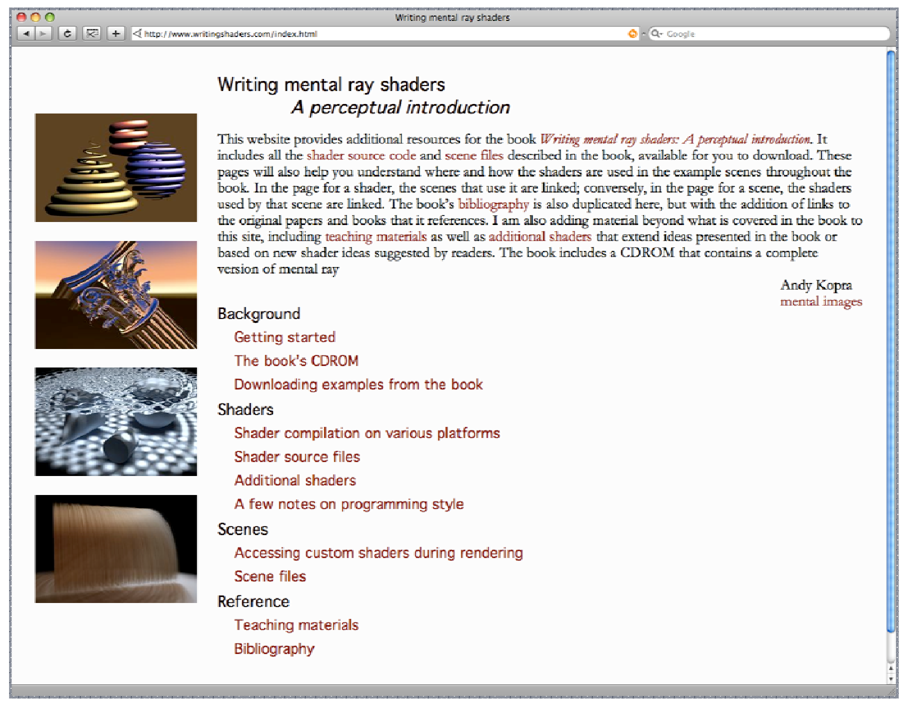

T. Driemeyer
mental ray Handbooks Vol. 1
Rendering with mental ray®
Third, completely revised edition
SpringerWienNewYork
with cd-rom


T. Driemeyer
R. Herken (eds.)
mental ray Handbooks Vol. 2
Programming mental ray®
Third, completely revised edition
SpringerWienNewYork
with cd-rom


Andy Kopra
mental ray Handbooks Vol. 3
Writing mental ray® Shaders
SpringerWienNewYork
mit cd-rom


Andy Kopra
mental ray Handbooks Vol. 3
Writing mental ray® Shaders
SpringerWienNewYork
mit cd-rom


mental ray® Handbooks, Vol. 3
Kopra: Writing mental ray® Shaders
© 2008 Springer-Verlag/Wien
ISSN 1438-9835
ISBN 978-3-211-48964-2
SpringerWienNewYork
mental ray® version 3.6.54
shader sources and scenes
© 1986–2008 mental images GmbH


Writing mental ray shaders
http://www.writingshaders.com/index.html
Writing mental ray shaders
A perceptual introduction
This website provides additional resources for the book Writing mental ray shaders: A perceptual introduction. It includes all the shader source code and scene files described in the book, available for you to download. These pages will also help you understand where and how the shaders are used in the example scenes throughout the book. In the page for a shader, the scenes that use it are linked; conversely, in the page for a scene, the shaders used by that scene are linked. The book's bibliography is also duplicated here, but with the addition of links to the original papers and books that it references. I am also adding material beyond what is covered in the book to this site, including teaching materials as well as additional shaders that extend ideas presented in the book or based on new shader ideas suggested by readers. The book includes a CDROM that contains a complete version of mental ray
Andy Kopra
mental images
Background
Getting started
The book's CDROM
Downloading examples from the book
Shaders
Shader compilation on various platforms
Shader source files
Additional shaders
A few notes on programming style
Scenes
Accessing custom shaders during rendering
Scene files
Reference
Teaching materials
Bibliography