Writing mental ray shaders

by Andy Kopra

Part 1: Structure

The structure of the scene

The structure of the scene

Understanding scene structure

Understanding scene structure

Why?

Understanding scene structure

Why?

To translate the corresponding vocabulary between applications

Understanding scene structure

Why?

- To translate the corresponding vocabulary between applications
- To clarify how mental ray is implemented in an application

Understanding scene structure

Why?

- To translate the corresponding vocabulary between applications
- To clarify how mental ray is implemented in an application
- To identify the various ways in which shaders and Phenomena can be used throughout the scene

Understanding scene structure

Why?

- To translate the corresponding vocabulary between applications
- To clarify how mental ray is implemented in an application
- To identify the various ways in which shaders and Phenomena can be used throughout the scene
- To promote the efficient design of shaders and Phenomena

Data representation in mental ray

Data representation in mental ray

- Scene represented by a *database of entities*

Data representation in mental ray

- Scene represented by a *database of entities*
- Objects, lights, cameras are *elements*

Data representation in mental ray

- Scene represented by a *database of entities*
- Objects, lights, cameras are *elements*
- Elements to be rendered are *instanced*

Data representation in mental ray

- Scene represented by a *database of entities*
- Objects, lights, cameras are *elements*
- Elements to be rendered are *instanced*
- Rendering procedures are *shaders*

Data representation in mental ray

- Scene represented by a *database of entities*
- Objects, lights, cameras are *elements*
- Elements to be rendered are *instanced*
- Rendering procedures are *shaders*
- Algorithmic parameters are specified as *options*

instance + shaders + options = render

The .mi scene description format

The .mi scene description format

- ASCII representation of scene database

The .mi scene description format

- ASCII representation of scene database
- Input data file for mental ray

The .mi scene description format

- ASCII representation of scene database
- Input data file for mental ray
- Grammar defined by `yacc` and provided in printed and on-line documentation

The .mi scene description format

- ASCII representation of scene database
- Input data file for mental ray
- Grammar defined by `yacc` and provided in printed and on-line documentation
- Composed of scene elements and commands

The .mi scene description format

- ASCII representation of scene database
- Input data file for mental ray
- Grammar defined by `yacc` and provided in printed and on-line documentation
- Composed of scene elements and commands
- Example scenes can be constructed and modified using a text editor

Top-level elements in the scene file

Top-level elements in the scene file

Specification

Top-level elements in the scene file

Specification

How should the rendering be done?

Top-level elements in the scene file

- Specification
  - *How should the rendering be done?*
- Shading definitions

Top-level elements in the scene file

- Specification
  - *How should the rendering be done?*
- Shading definitions
  - *How should rendered things look?*

Top-level elements in the scene file

- Specification
  - *How should the rendering be done?*
- Shading definitions
  - *How should rendered things look?*
- Rendering entities

Top-level elements in the scene file

- Specification
  - *How should the rendering be done?*
- Shading definitions
  - *How should rendered things look?*
- Rendering entities
  - *What will be rendered in the scene?*

Top-level elements in the scene file

- Specification
  - *How should the rendering be done?*
- Shading definitions
  - *How should rendered things look?*
- Rendering entities
  - *What will be rendered in the scene?*
- Instance placement and grouping

Top-level elements in the scene file

- Specification
  - *How should the rendering be done?*
- Shading definitions
  - *How should rendered things look?*
- Rendering entities
  - *What will be rendered in the scene?*
- Instance placement and grouping
  - *Where should rendered things be placed?*

Top-level elements in the scene file

- Specification
  - **`options`**
- Shading definitions
  - **`shader texture material data`**
- Rendering entities
  - **`camera object light lightprofile`**
- Instance placement and grouping
  - **`instance instgroup`**

Top-level elements in the scene file

How (general)

`options`

How (shaders)

`shader texture material data`

What

`camera object light lightprofile`

Where

`instance instgroup`

A simple scene file

Set options
Define camera
Instance camera
Define material
Define object
Instance object
Define root group
Render

A simple scene file

- Set options
- Define camera
- Instance camera
- Define material
- Define object
- Instance object
- Define root group
- Render

```
options "opt"
    object space
    contrast .1 .1 .1
end options
```

A simple scene file

- Set options
- Define camera
- Instance camera
- Define material
- Define object
- Instance object
- Define root group
- Render

```
camera "cam"
    output "tif" "y.tif"
    focal 35
    aperture 22.05
    aspect 1.333
    resolution 400 300
end camera
```

A simple scene file

- Set options
- Define camera
- Instance camera
- Define material
- Define object
- Instance object
- Define root group
- Render

```
instance "cam-inst" "cam"
    transform
        1   0   0   0
        0   1   0   0
        0   0   1   0
        0   0  -9   1
end instance
```

A simple scene file

- Set options
- Define camera
- Instance camera
- Define material
- Define object
- Instance object
- Define root group
- Render

```
material "yellow"
    "mib_illum_lambert" (
        "ambient"  1 1 0,
        "ambience" 1 1 1
    )
end material
```

A simple scene file

Set options
Define camera
Instance camera
Define material
Define object
Instance object
Define root group
Render

```
object "square"
    visible
    group
        -.5 -.5  0
         .5 -.5  0
         .5  .5  0
        -.5  .5  0
         v 0 v 1 v 2 v 3
         p 0 1 2 3
    end group
end object
```

A simple scene file

Set options
Define camera
Instance camera
Define material
Define object
Instance object
Define root group
Render

```
instance "square-inst" "square"
    material "yellow"
    transform
          1   0   0   0
          0   1   0   0
          0   0   1   0
         -2  -1   0   1
end instance
```

A simple scene file

Set options
Define camera
Instance camera
Define material
Define object
Instance object
Define root group
Render

```
instgroup "root"
        "cam-inst" "square-inst"
end instgroup
```

A simple scene file

Set options
Define camera
Instance camera
Define material
Define object
Instance object
Define root group
Render

```
render "root" "cam-inst" "opt"
```

A simple scene file

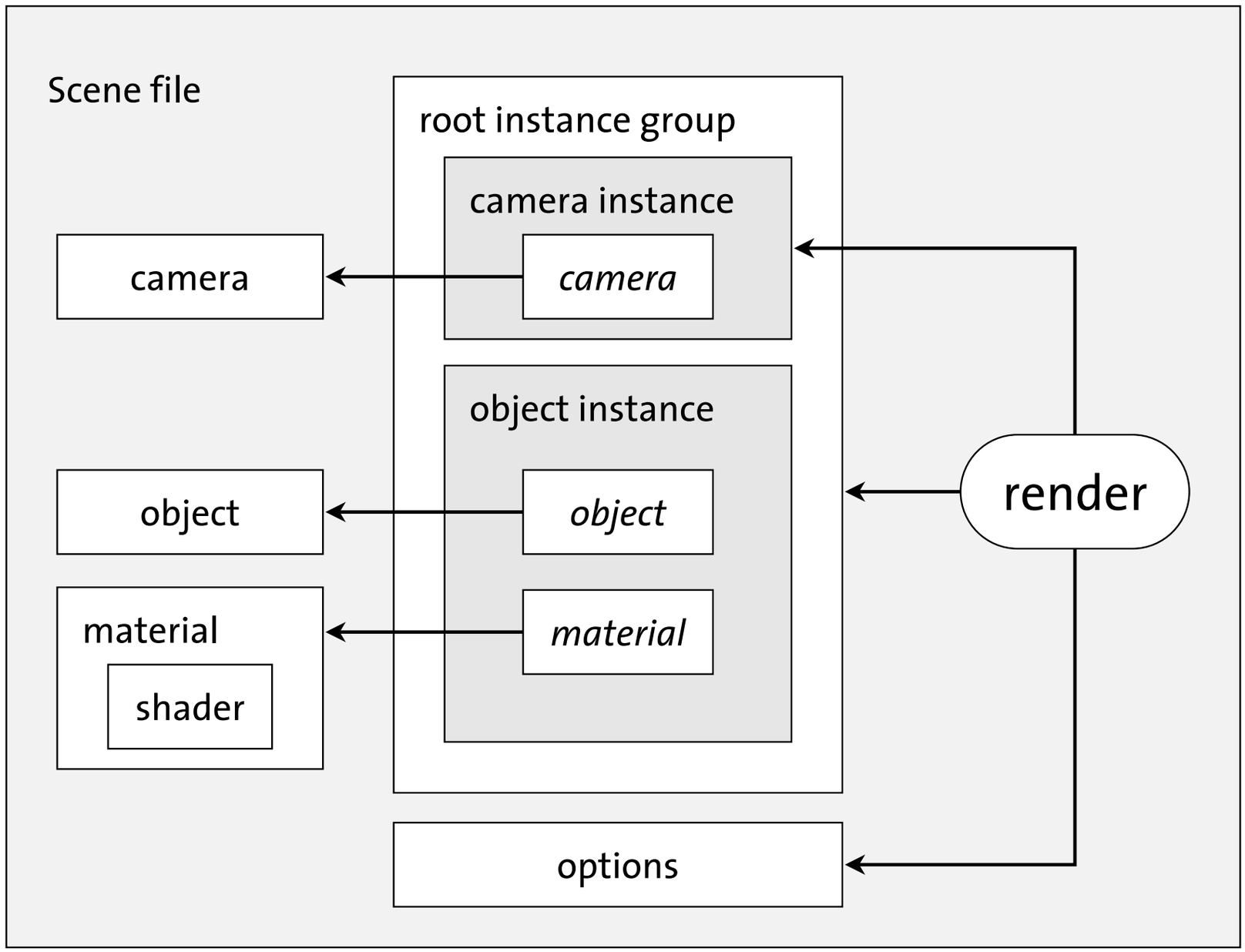
Set options
Define camera
Instance camera
Define material
Define object
Instance object
Define root group
Render

Exercise 3: Rendering and editing a scene

1. In a terminal window (Command Prompt in Windows, the Terminal app in OS X, or a shell in Linux), go the `MRT/ray/shaders` directory.
2. Compile `one_color.c`. In Windows, run `one_color.bat`. In Linux and OS X, enter `make one_color.so`. What new file(s) have been created?
3. In a terminal window, go to the `MRT/ray/scenes` directory.
4. Render: `ray yellow_square_1.mi`
5. View `yellow_square_1.tif` with `imf_disp`

Exercise 3: Rendering and editing a scene (part 2)

1. Change the current directory to `MRT/ray/scenes`
2. Copy `yellow_square_1.mi` to `yellow_square.mi`.
3. Open file `picture.mi` in a text editor (for example, WordPad in Windows, TextEdit in OS X, or Emacs in Linux).
4. Change the output filename to `yellow_square.tif` (search for the string ".tif") and re-render. Display resulting image.
5. Find the transformation matrix of the square's instance.
6. Move the square to the left, and re-render. Translation is defined by the last row of the matrix.
7. Find the material name of the instance of the square. Find where that material is defined.
8. Change the color of the square, and re-render.



The hierarchical structure of the scene

The structure of the scene

A complete scene file

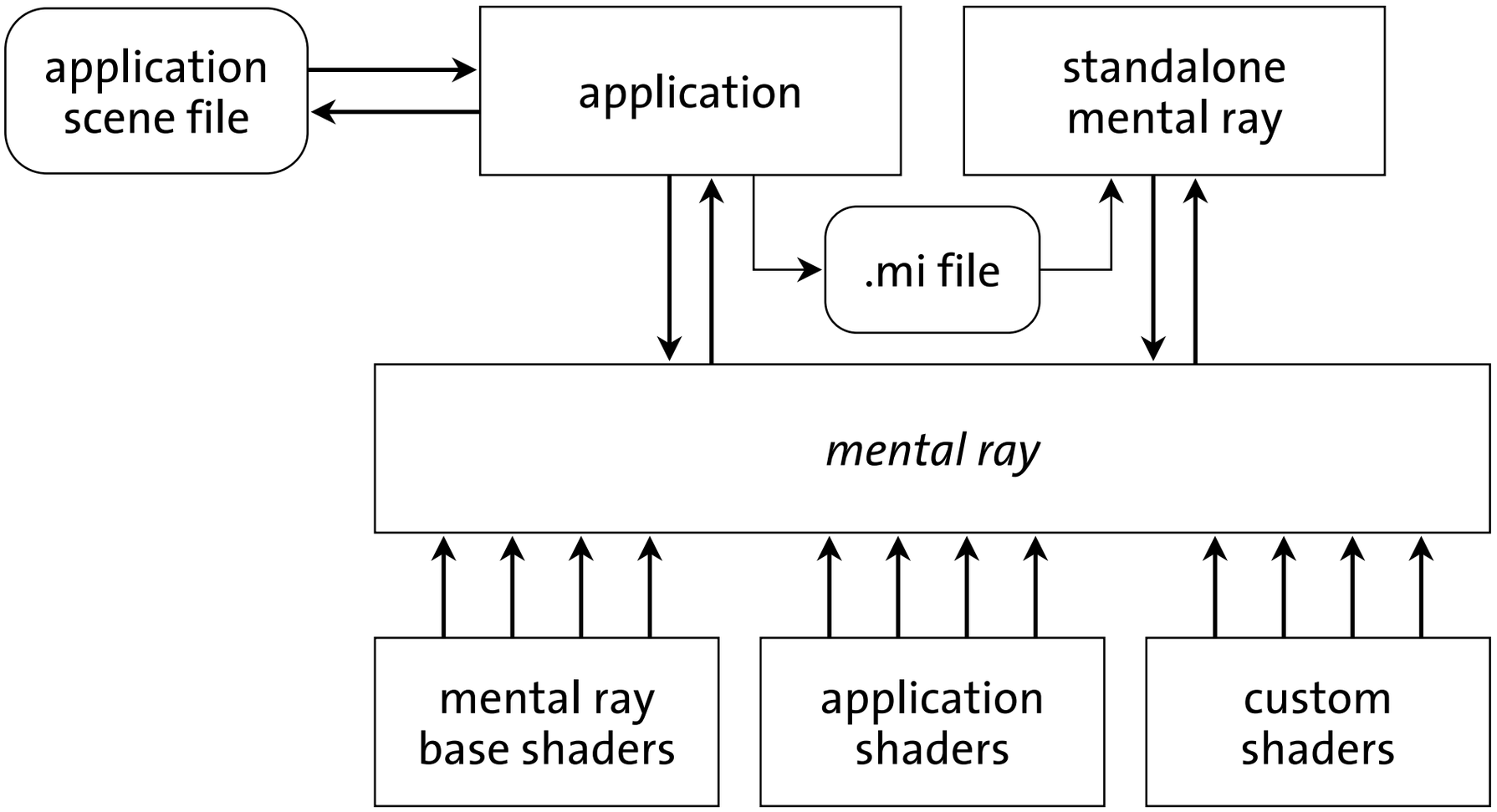
```
verbose on
link "one_color.so"
$include "one_color.mi"

options "options"
    object space
    contrast .1 .1 .1
end options

camera "camera"
    output     "rgba" "tif" "picture.tif"
    focal      35
    aperture   22.05
    aspect     1.3333
    resolution 200 150
end camera

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

material "yellow"
    "one_color" ( "color"  1 1 0 )
end material

object "square"
    visible
    group
        -.5 -.5  0
         .5 -.5  0
         .5  .5  0
        -.5  .5  0
        v 0 v 1 v 2 v 3
        p 0 1 2 3
    end group
end object

instance "yellow-square" "square"
    material "yellow"
    transform
         1  0  0  0
         0  1  0  0
         0  0  1  0
        -2 -1  0  1
end instance

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

render "root" "main-camera" "options"
```

A complete scene file

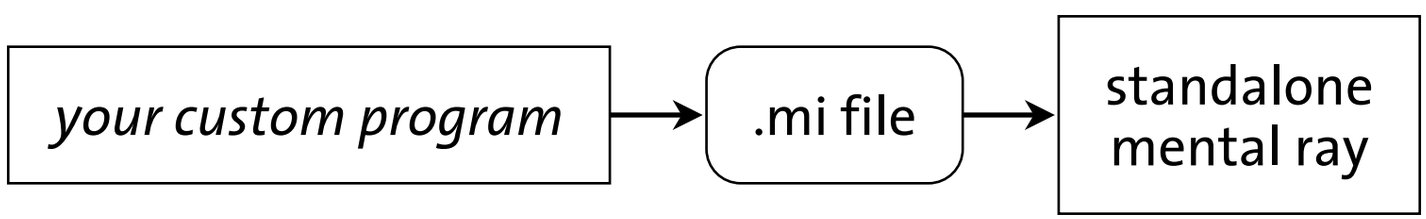
The structure of the scene

A complete scene file

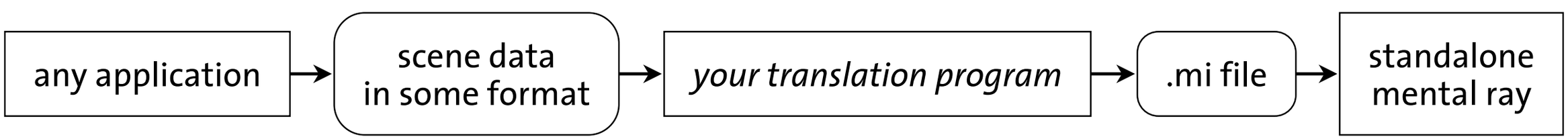
Rendering of scene file square_scene.mi



Shaders, 3D applications, standalone mental ray, and the mental ray core



The .mi scene file as the input to mental ray from your program



The .mi scene file as the output of your translation program

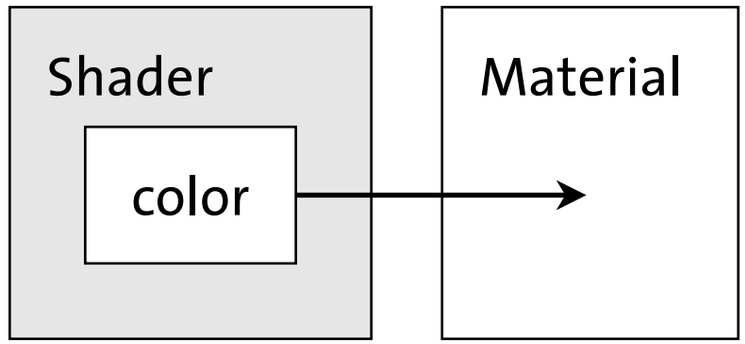
The structure of a shader

The structure of a shader

- Defining a color with a shader
- Providing input to a shader
- Telling the shader about its context
- Accumulating the effect of a series of shaders
- Combining the previous and current results

The structure of a shader

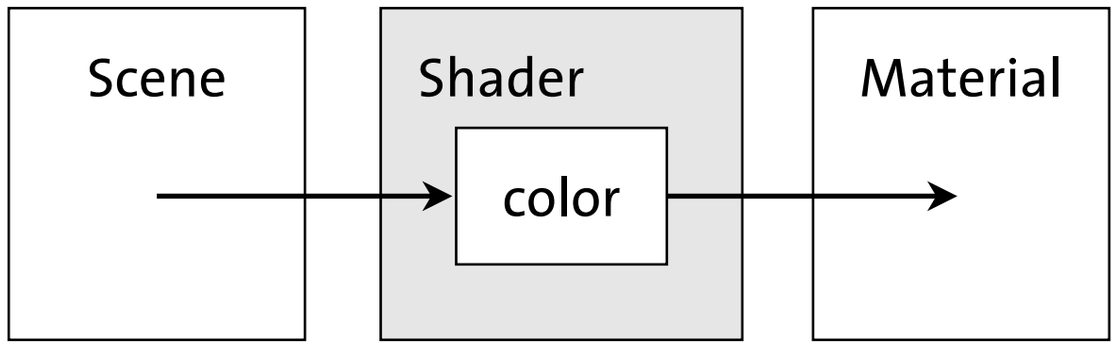
Defining a color with a shader



A shader that always produces the same color

The structure of a shader

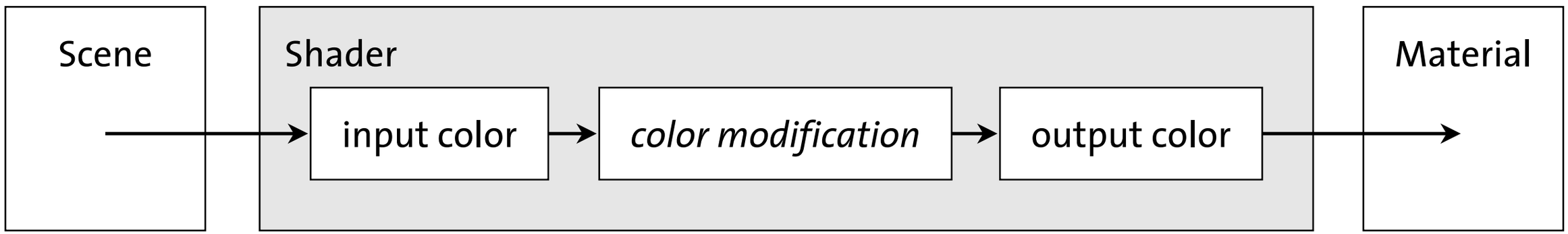
Defining a color with a shader



A shader that provides the color supplied as an input

The structure of a shader

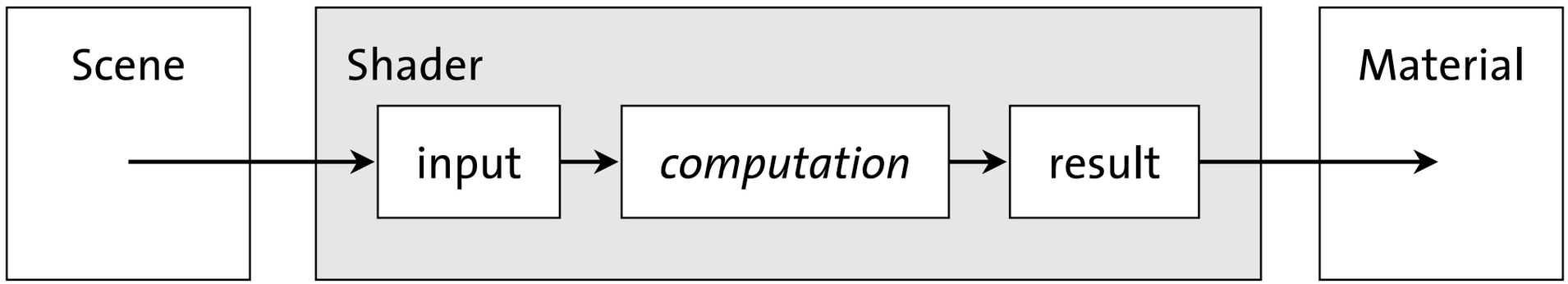
Defining a color with a shader



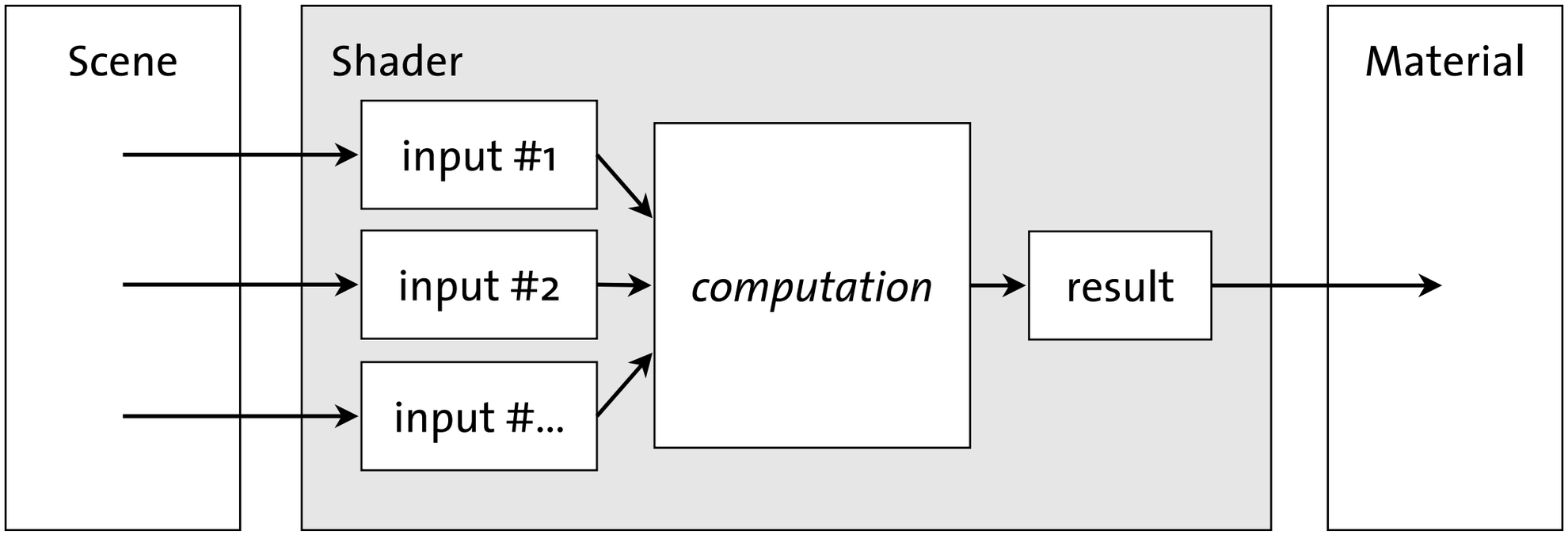
Modifying an input color to produce a new color

The structure of a shader

Providing input to a shader



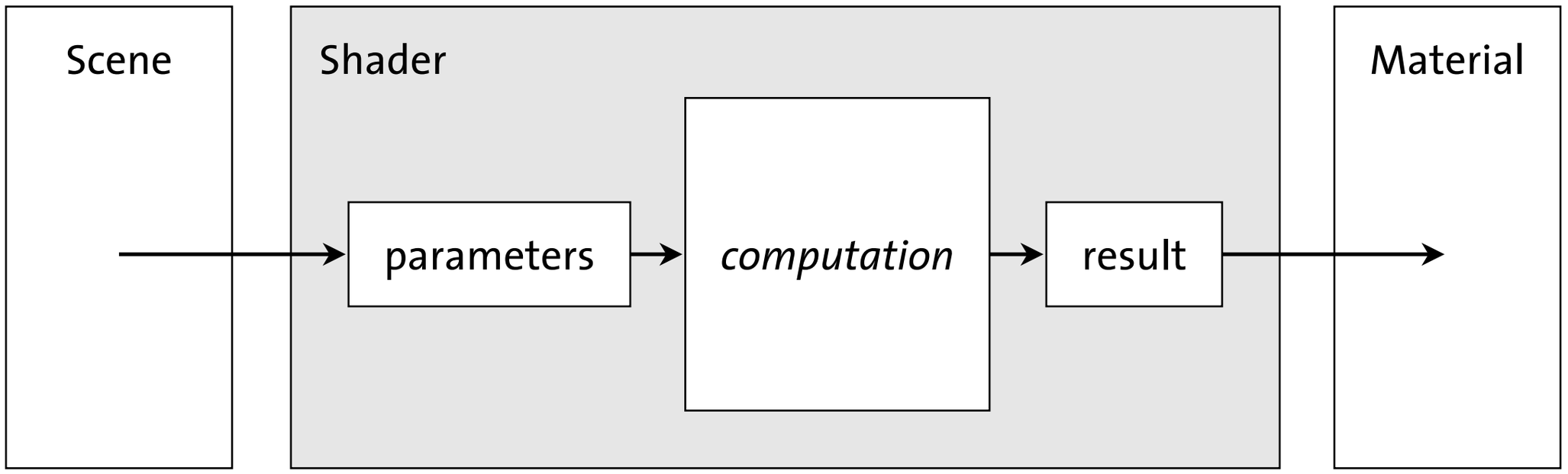
Modifying an arbitrary input value



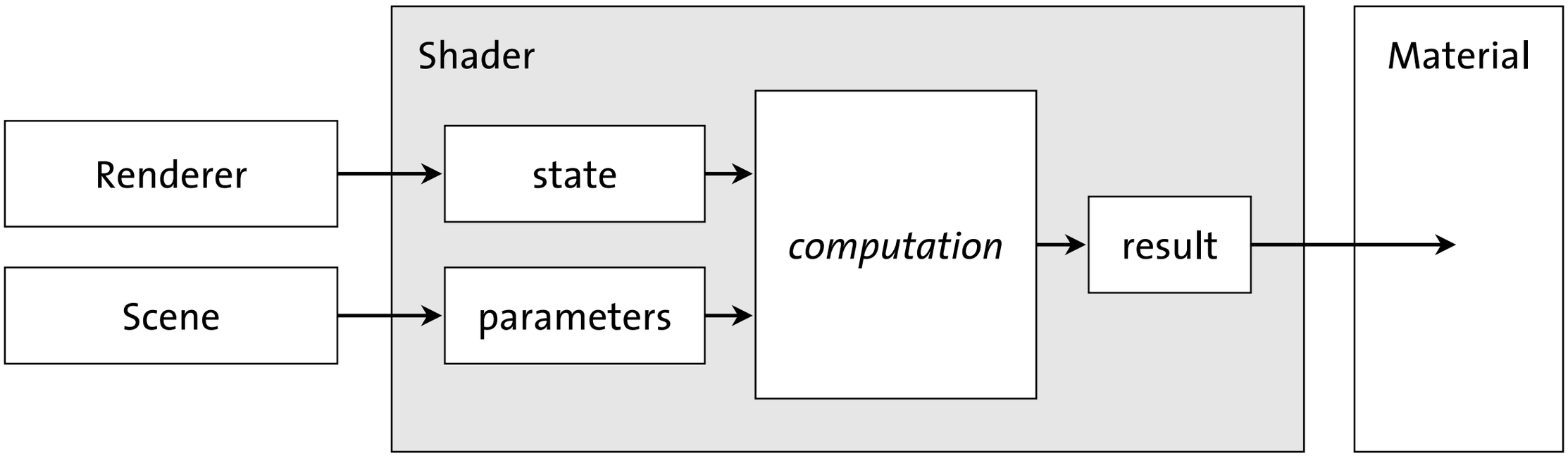
Supplying multiple inputs to a shader

The structure of a shader

Providing input to a shader



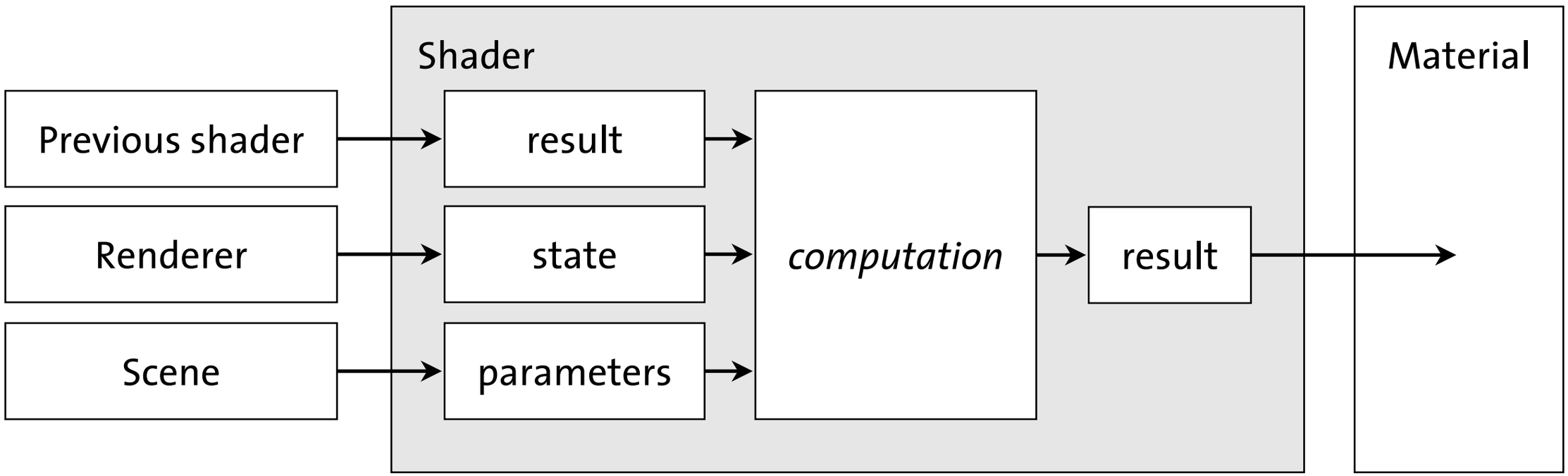
Using the parameter input as a container for arbitrary amounts of data



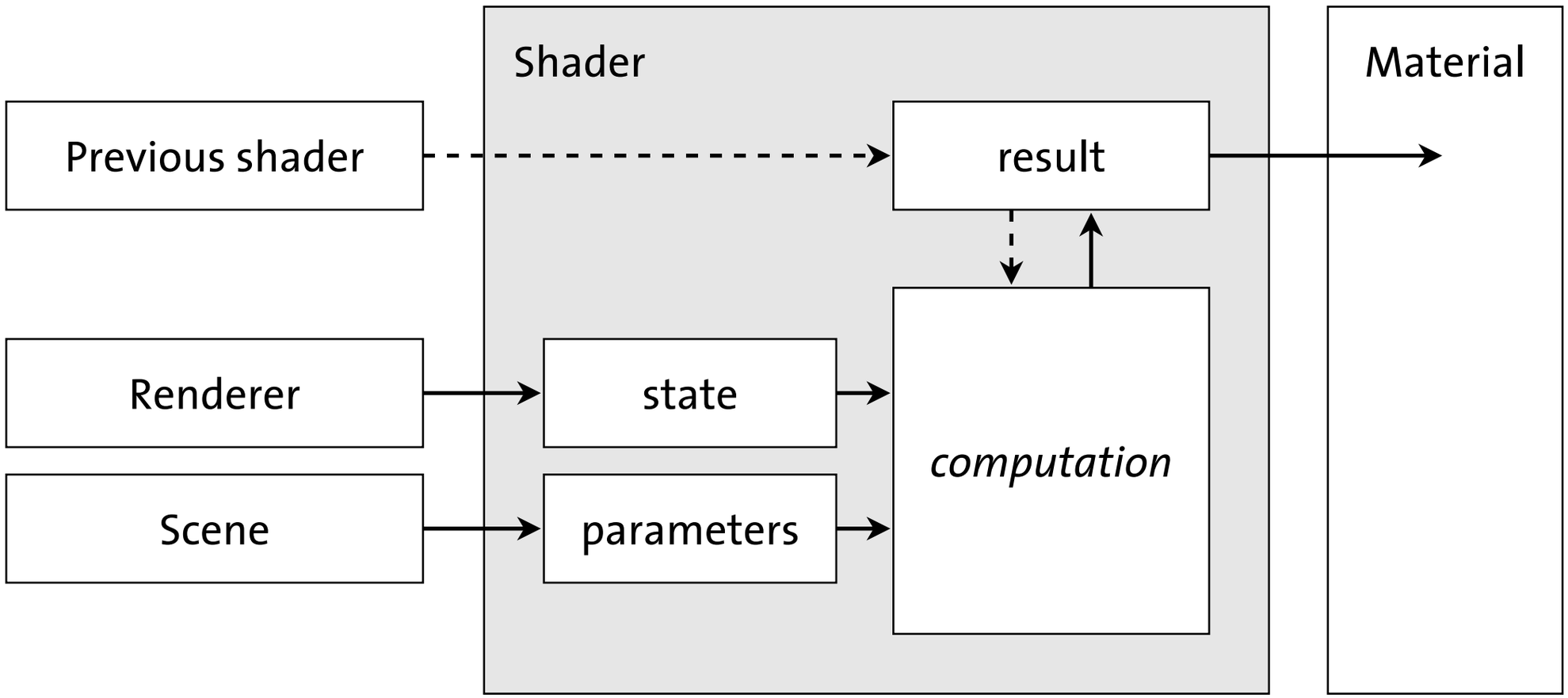
Adding the current rendering state as an input to the shader

The structure of a shader

Accumulating the effect of a series of shaders



Providing the result of the previous shader as an input



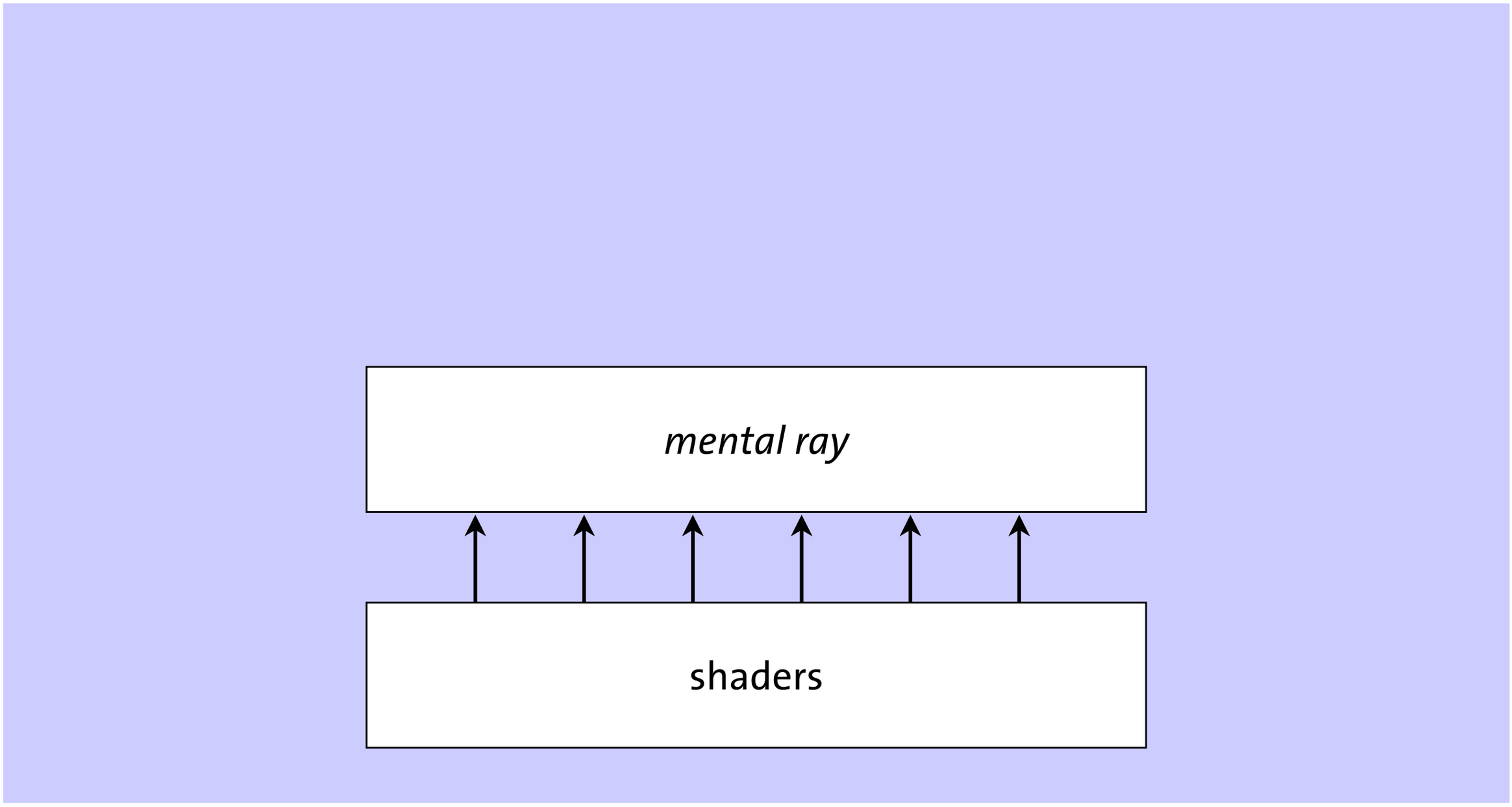
The standard shader input/output model

Shaders in the scene

Shaders in the scene

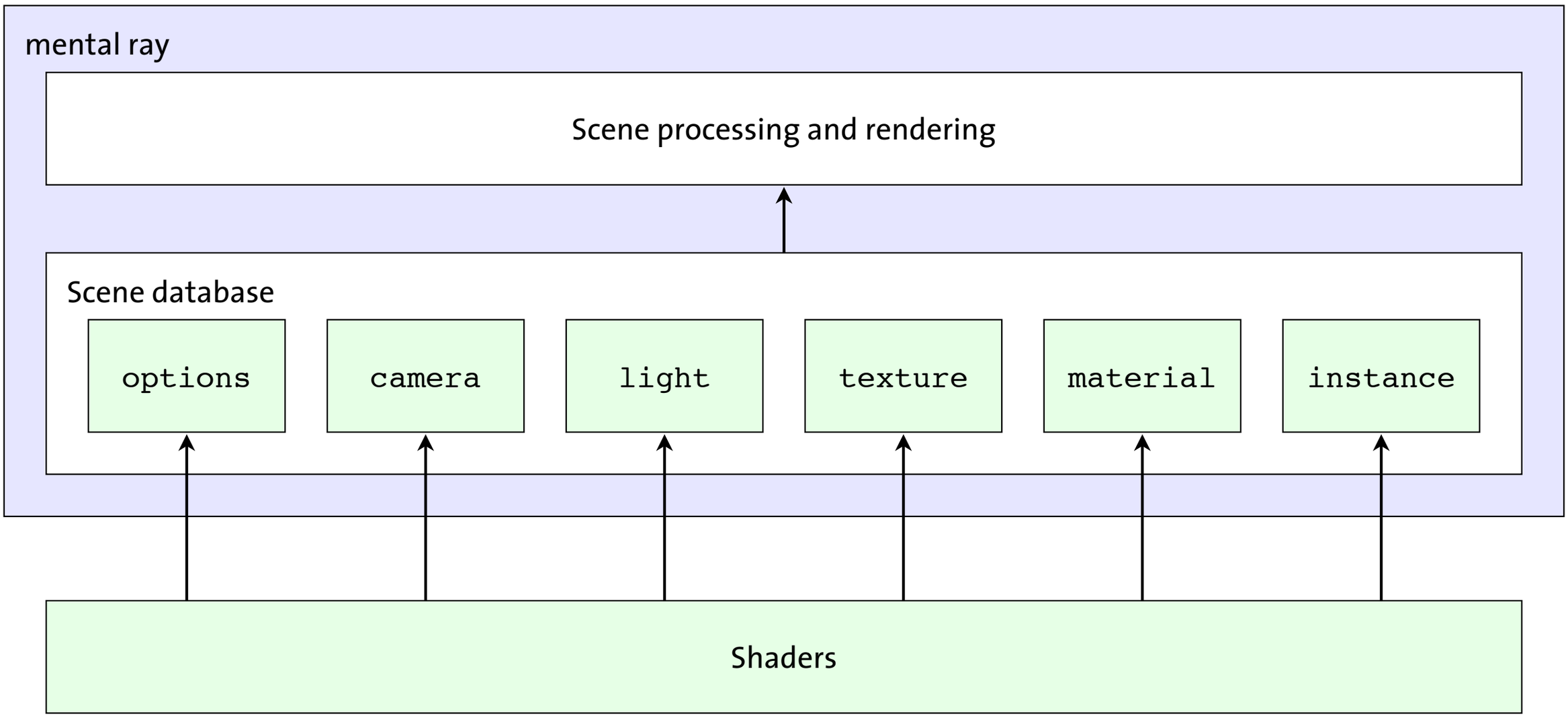
- Describing parameters: shader declarations
- Missing parameter values: defining defaults
- Defining the material: anonymous and named shaders
    - Anonymous shaders
    - Named shaders
- Chaining shaders of the same result type: shader lists
    - Using the result of the previous shader
    - Modifying the rendering state in a shader list
    - Named and anonymous shaders in the shader list
- Connecting shaders in a network: shader graphs
- Named shader graphs: Phenomena
    - The components of a Phenomenon
    - Defining a material as a Phenomenon
- The five shader usage types

Shaders in the scene



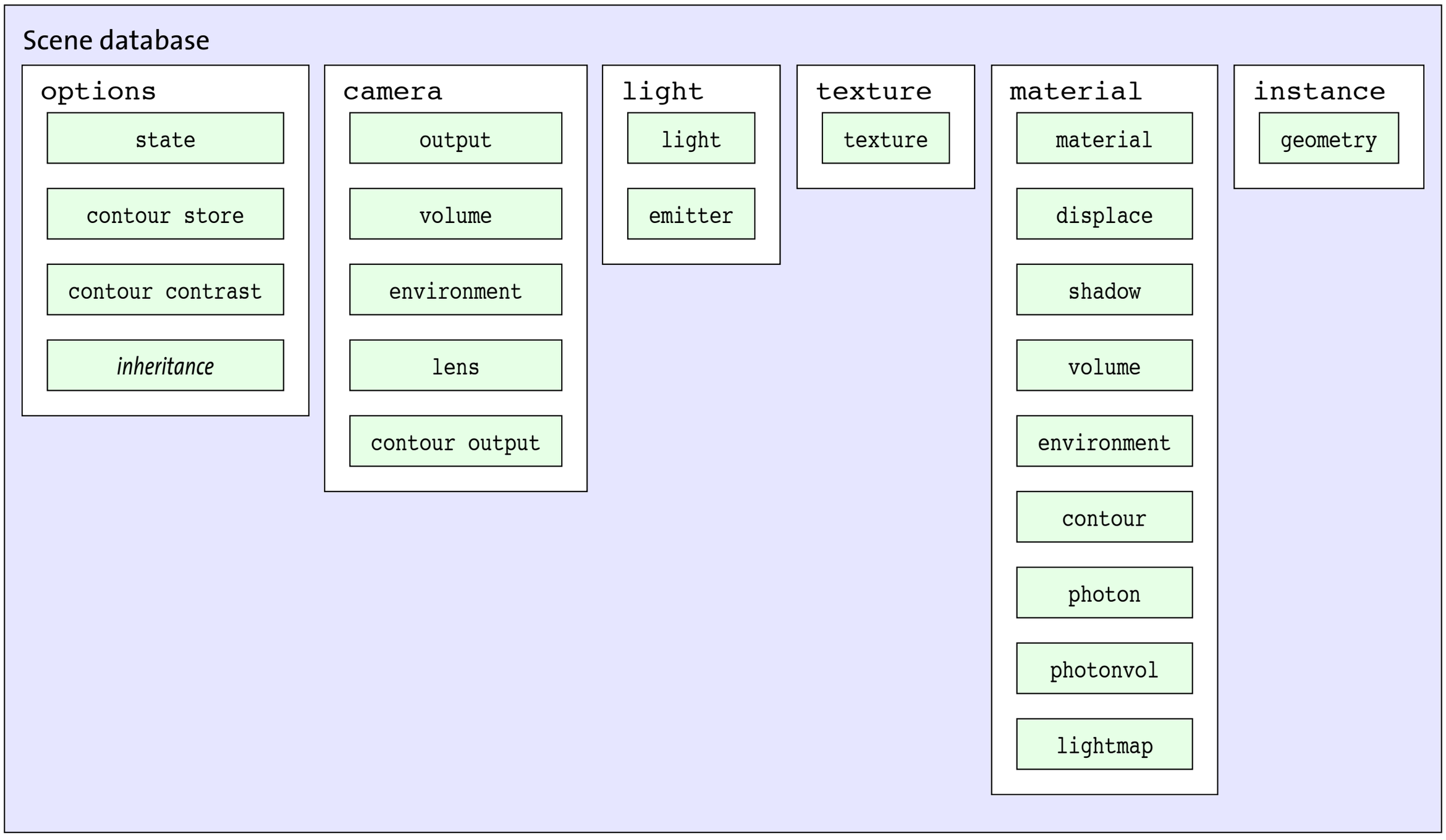
Shaders are components loaded by the mental ray core when rendering begins

Shaders in the scene



Shaders loaded at run-time are referenced throughout the scene database

Shaders in the scene



Different scene elements use different subsets of the set of shader types

Shader declaration and use

Shader declaration and use

Shader declaration

Shader declaration and use

Shader declaration

Specification of the shader data types

Shader declaration and use

Shader declaration

Specification of the shader data types

The shader parameters

Shader declaration and use

Shader declaration

Specification of the shader data types

The shader parameters

The shader's resulting value

Shader declaration and use

Shader declaration

Specification of the shader data types

The shader parameters

The shader's resulting value

Shader use ("calling the shader")

Shader declaration and use

Shader declaration

Specification of the shader data types

The shader parameters

The shader's resulting value

Shader use ("calling the shader")

Parameters provided with actual values

Shader declaration and use

Shader declaration

Specification of the shader data types

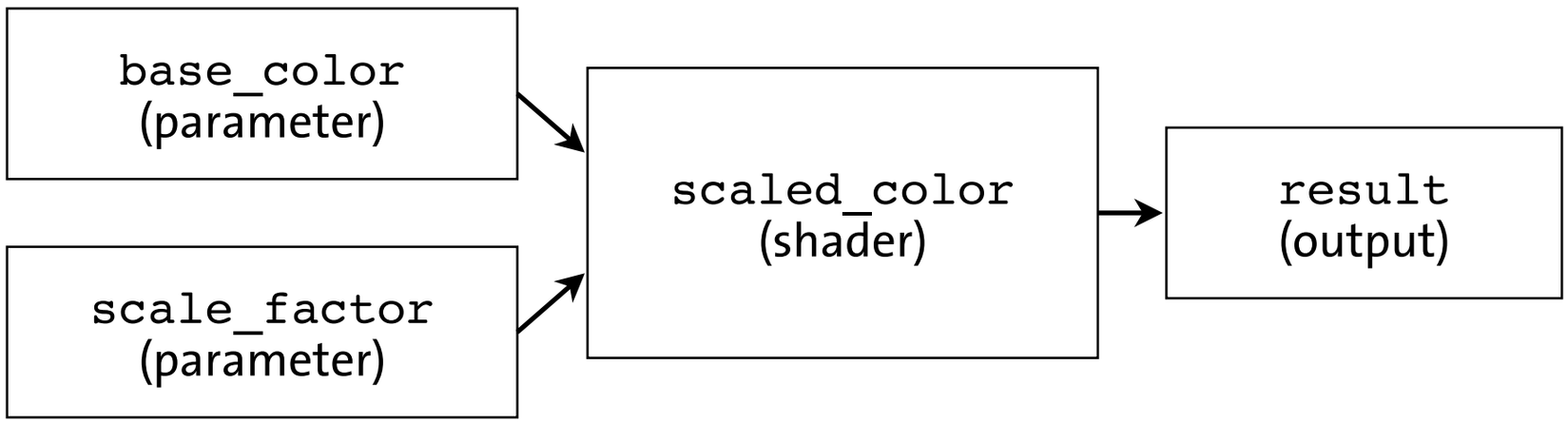
The shader parameters

The shader's resulting value

Shader use ("calling the shader")

Parameters provided with actual values

Shaders contained in materials



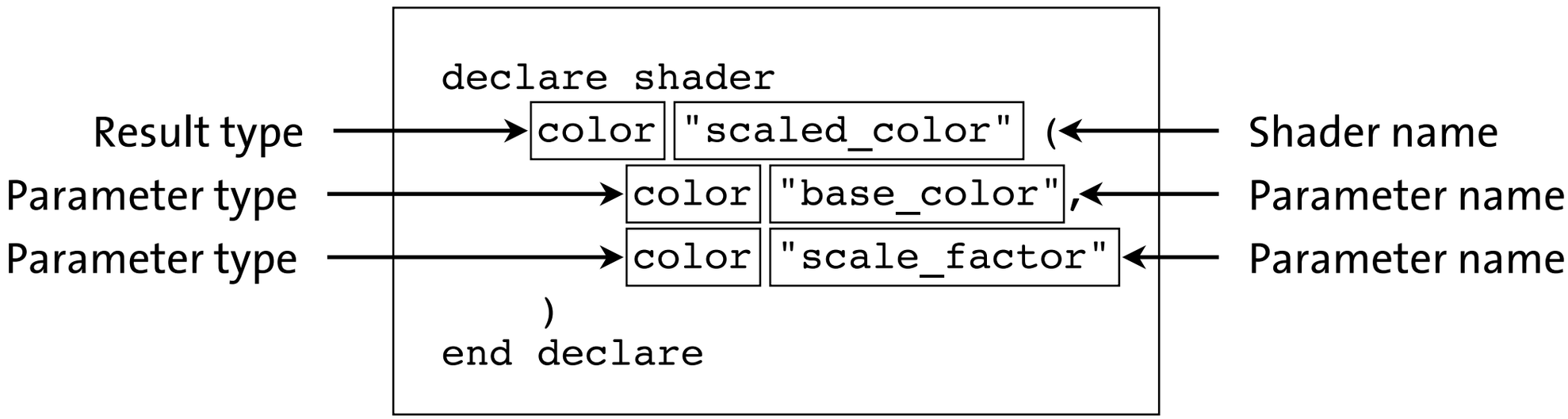
A shader to multiply two colors

```
result->r = base_color->r * scale_factor->r;
result->g = base_color->g * scale_factor->g;
result->b = base_color->b * scale_factor->b;
```

Multiplying color components in C

```
declare shader
    color "scaled_color" (
        color "base_color",
        color "scale_factor"
    )
end declare
```

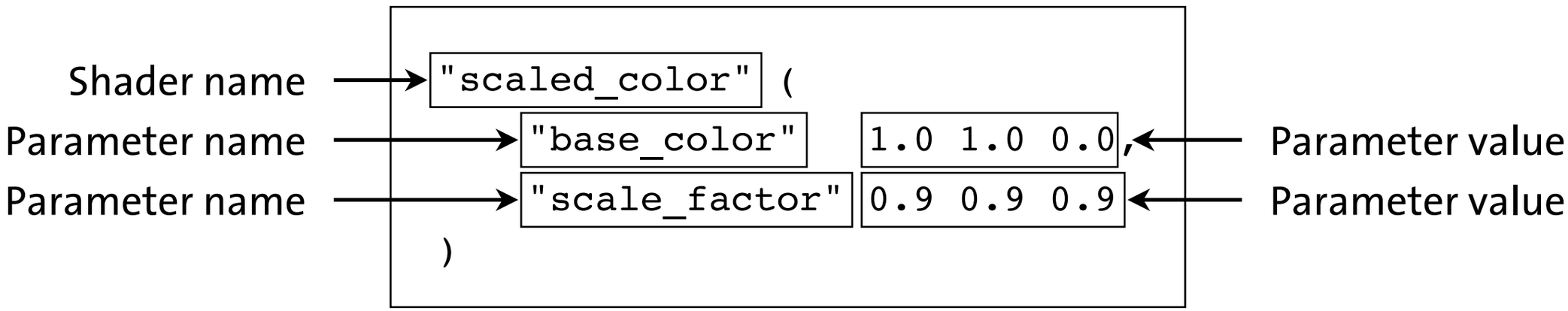
A shader declaration



Parameter names and data types in a shader declaration

```
"scaled_color" (
    "base_color"   1.0 1.0 0.0,
    "scale_factor" 0.9 0.9 0.9
)
```

A shader definition



Parameter names and values in a shader definition

Missing parameter values: defining defaults

```
declare shader
    color "scaled_color" (
        color "base_color"   default 1 1 1,
        color "scale_factor" default 1 1 1
    )
end declare
```

A shader declaration with default values

Missing parameter values: defining defaults

```
"scaled_color" ("base_color" 1 1 0)
```

A definition of scaled_color using a default value for scale_factor

Missing parameter values: defining defaults

```
declare shader
    color "mib_amb_occlusion" (
        integer "samples"             default 16,
        color   "bright"              default 1 1 1 1,
        color   "dark"                default 0 0 0 0,
        scalar  "spread"              default 0.8,
        scalar  "max_distance"        default 0,
        boolean "reflective"          default off,
        integer "output_mode"         default 0,
        boolean "occlusion_in_alpha" default off,
        # Version 2 parameters
        scalar  "falloff"             default 1.0,
        integer "id_inclexcl"         default 0,
        integer "id_nonself"          default 0,
    )
    version 2
    apply texture, light
end declare
```

Declaration of the ambient occlusion shader in the standard mental ray shader library

Using shaders in the scene file

Using shaders in the scene file

Anonymous shaders

Using shaders in the scene file

Anonymous shaders

Shader called directly in the material

Using shaders in the scene file

Anonymous shaders

Shader called directly in the material

Named shaders

Using shaders in the scene file

Anonymous shaders

Shader called directly in the material

Named shaders

Shader defined for later reference

Using shaders in the scene file

Anonymous shaders

Shader called directly in the material

Named shaders

Shader defined for later reference

Shader lists

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

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

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

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

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

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

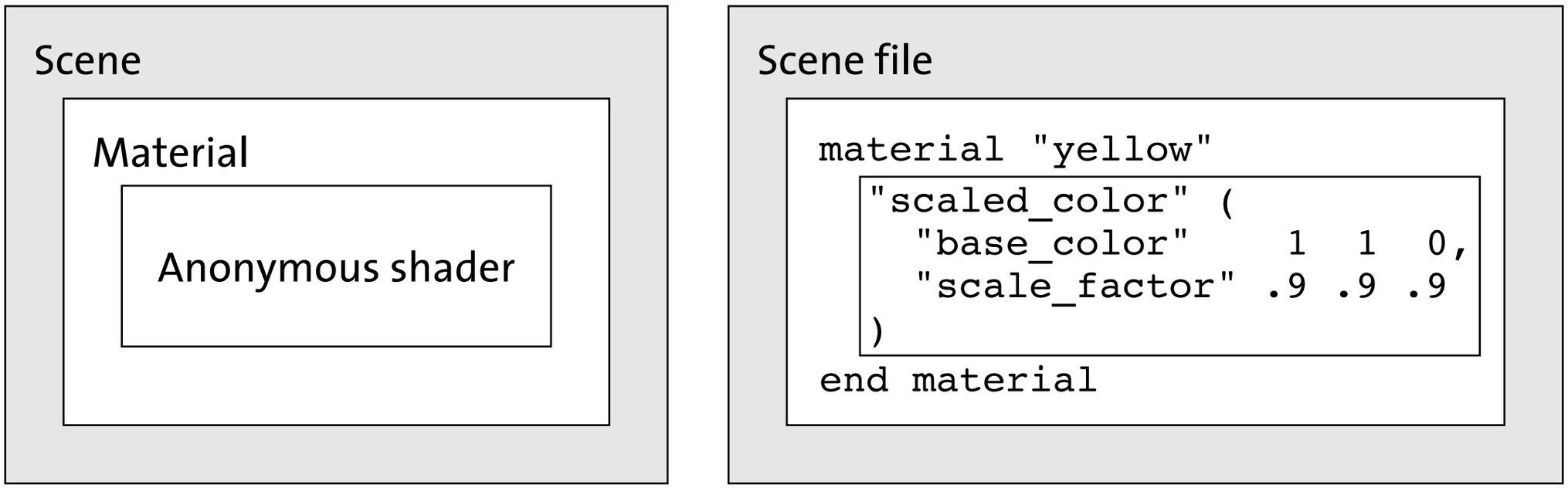
Shaders used as input to other shaders

Phenomena

Formalized shader graphs

Shaders in the scene

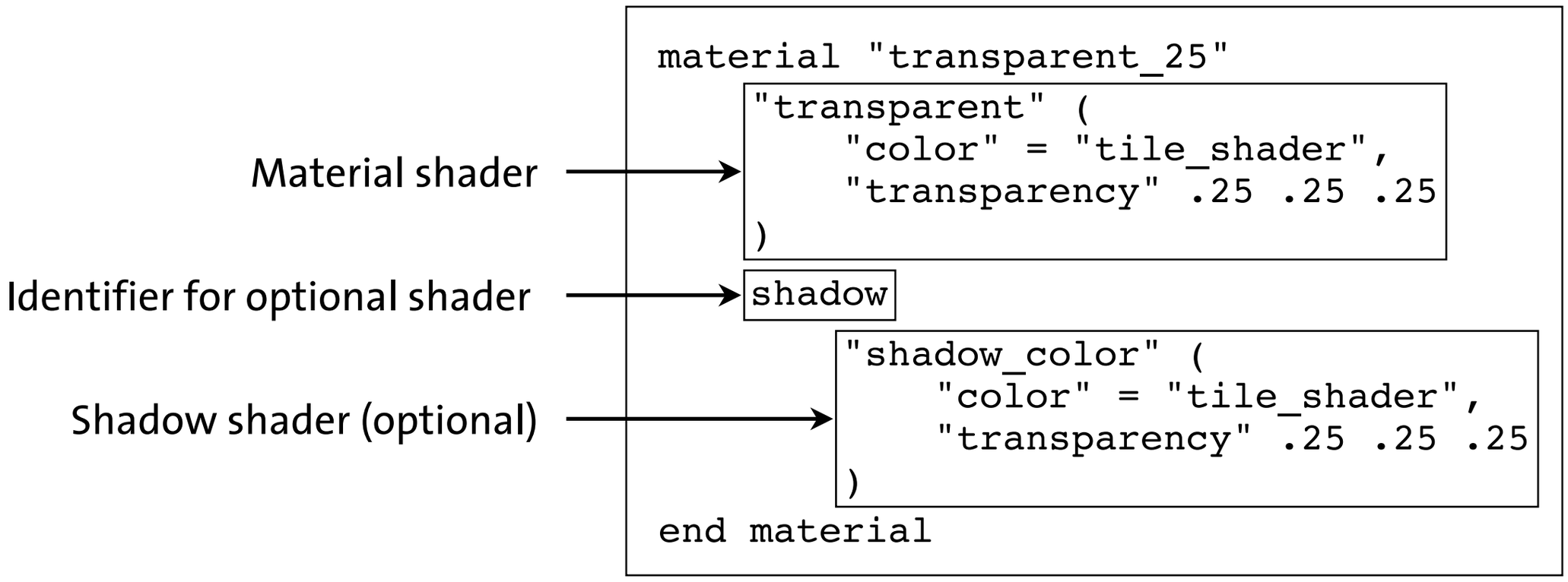
Defining the material: anonymous and named shaders



Anonymous shader in a material

Shaders in the scene

Defining the material: anonymous and named shaders



Material with a shadow statement and shadow shader

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

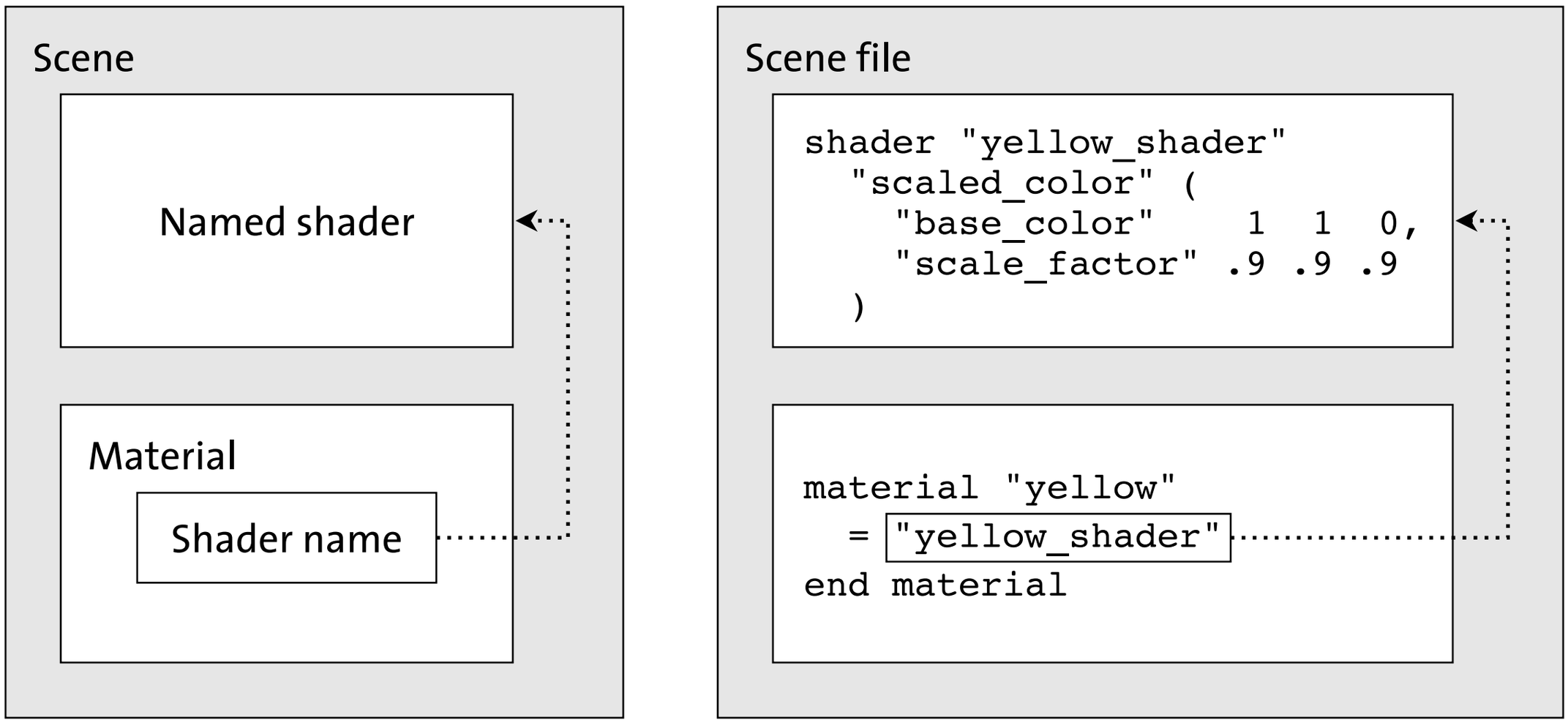
Shaders used as input to other shaders

Phenomena

Formalized shader graphs

Shaders in the scene

Defining the material: anonymous and named shaders



Named shader in a material referring to a previous shader definition

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

Chaining shaders of the same result type: shader lists

```
result->r += color_to_add->r;
result->g += color_to_add->g;
result->b += color_to_add->b;
```

Adding color components in C to the previous shader's result

Chaining shaders of the same result type: shader lists

```
declare shader
    color "add_color" (
        color "color_to_add"
    )
end declare
```

Declaration of shader add_color

```
result->a = new_alpha_value;
```

Setting a new alpha value in C

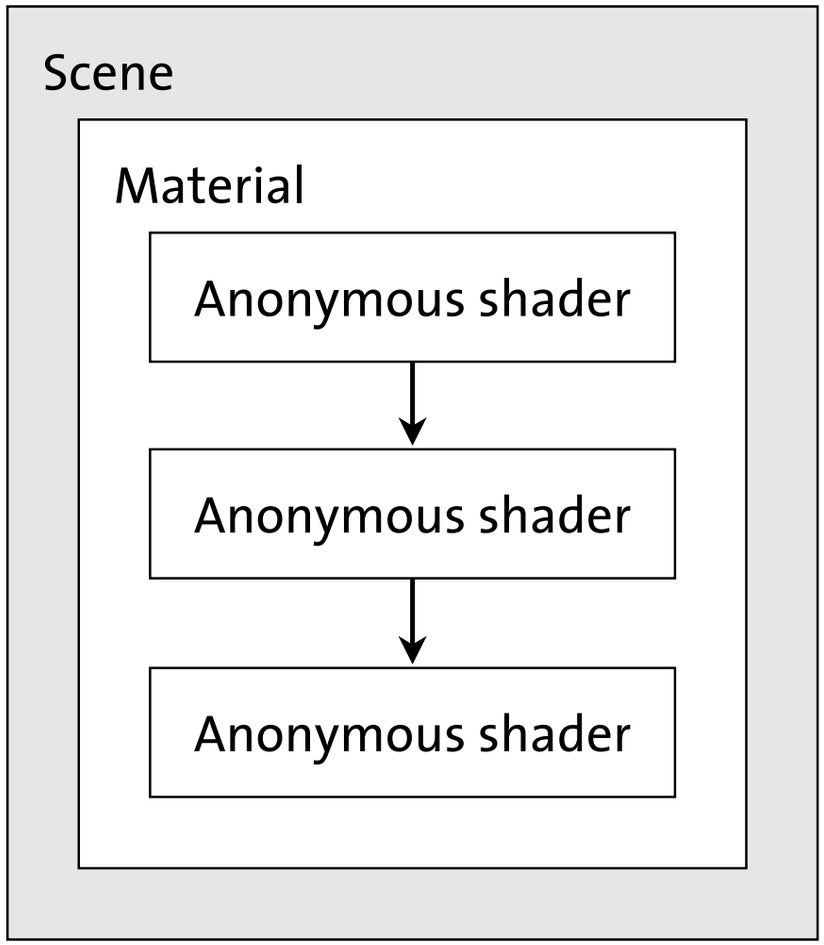
Chaining shaders of the same result type: shader lists

```
declare shader
    color "set_alpha" (
        scalar "new_alpha"
    )
end declare
```

Declaration of shader set_alpha

Shaders in the scene

Chaining shaders of the same result type: shader lists



Scene file

```
material "color_correct"
    "scaled_color" (
        "base_color"    1  1  0,
        "scale_factor" .9 .9 .9
    )
    "add_color" ( "color_to_add" 0 0 .2 )
    "set_alpha" ( "new_alpha" 1 )
end material
```

A list of anonymous shaders in a material

Chaining shaders of the same result type: shader lists

```
light "light"
    "point_light" (
        "light_color"  1 1 1
    )
    origin 10 10 20
end light

instance "light_instance"
    "light"
end instance
```

Use of standard shader mib_point_light to create light instance sidelight

Shaders in the scene

Chaining shaders of the same result type: shader lists

```
material "waves-1"
    "mib_illum_phong" (
        "diffuse" .6 .6 .6,
        "specular" .4 .4 .4,
        "exponent" 50,
        "lights" ["light_instance"]
    )
end material
```

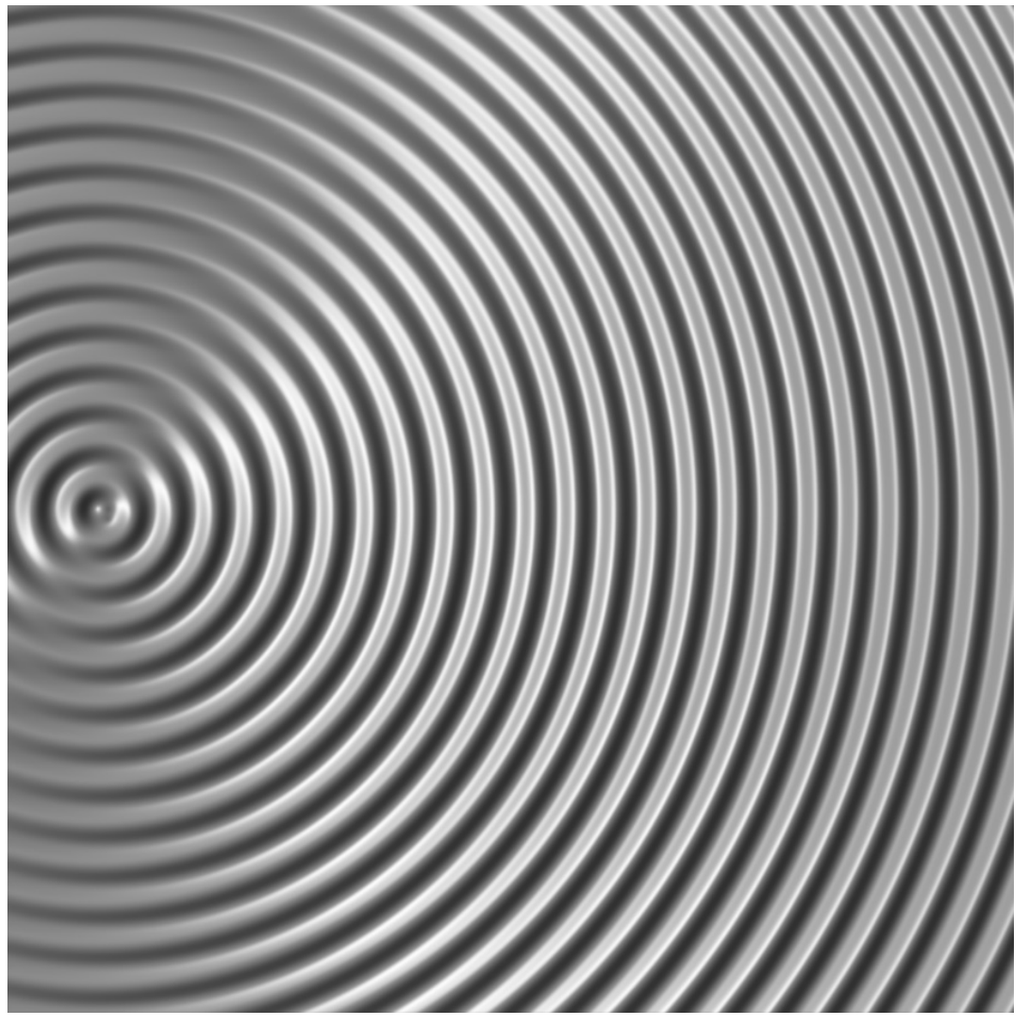
Rendering using mib_illum_phong only

Chaining shaders of the same result type: shader lists

```
declare shader
    color "bump_ripple" (
        vector "center"    default .5 .5 0,
        scalar "frequency" default 1,
        scalar "amplitude" default .5,
        scalar "nearby"    default .01 )
end declare
```

Scene file declaration of shader "bump_ripple"

Shaders in the scene



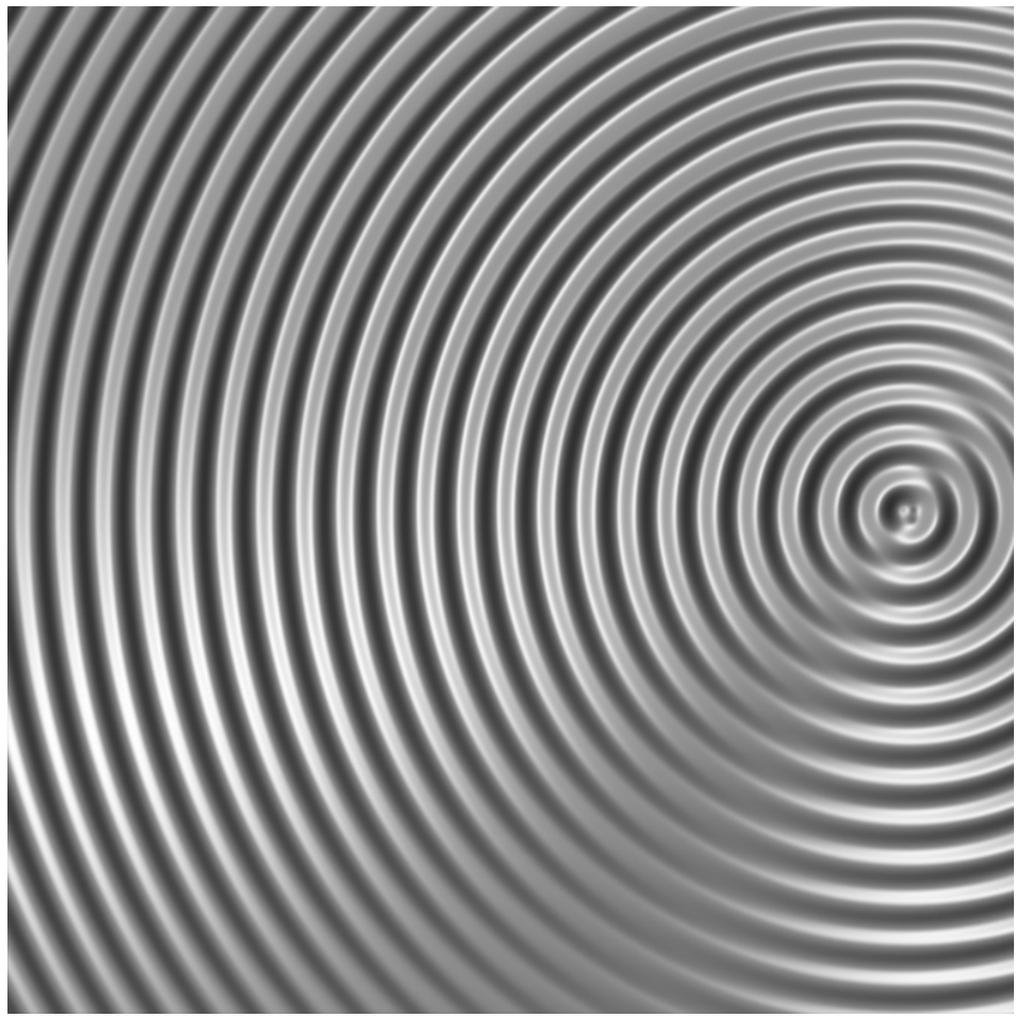
Chaining shaders of the same result type: shader lists

```
material "waves-2"
    "bump_ripple" (
        "center" .1 .5 0,
        "frequency" 25,
        "amplitude" .35
    )
    "mib_illum_phong" (
        "diffuse" .6 .6 .6,
        "specular" .4 .4 .4,
        "exponent" 35,
        "lights" ["light_instance"]
     )
end material
```

Rendering with modification of surface normal by bump_ripple

Shaders in the scene

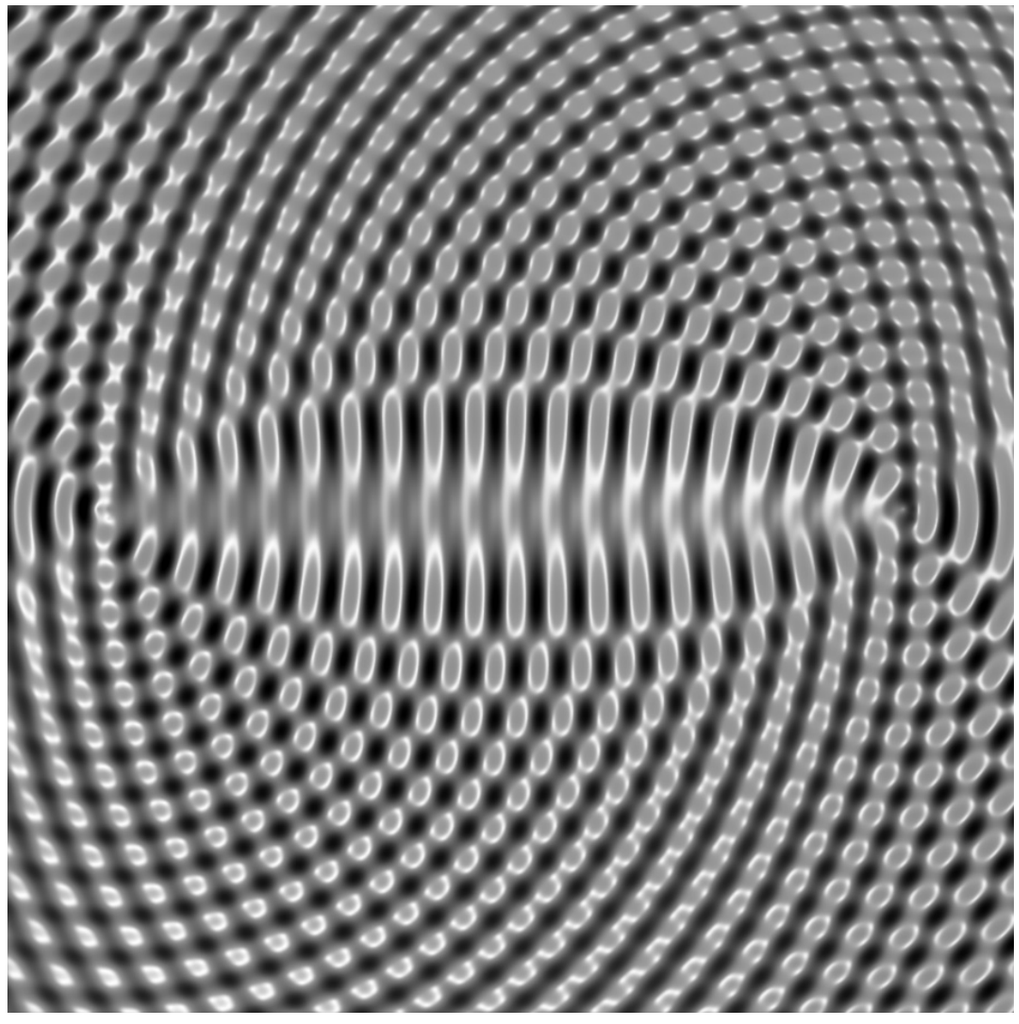
Chaining shaders of the same result type: shader lists



```
material "waves-3"
    "bump_ripple" (
        "center" .9 .5 0,
        "frequency" 25,
        "amplitude" .35
    )
    "mib_illum_phong" (
        "diffuse" .6 .6 .6,
        "specular" .4 .4 .4,
        "exponent" 35,
        "lights" ["light_instance"]
    )
end material
```

Shader bump_ripple with a different value for the center parameter

Shaders in the scene



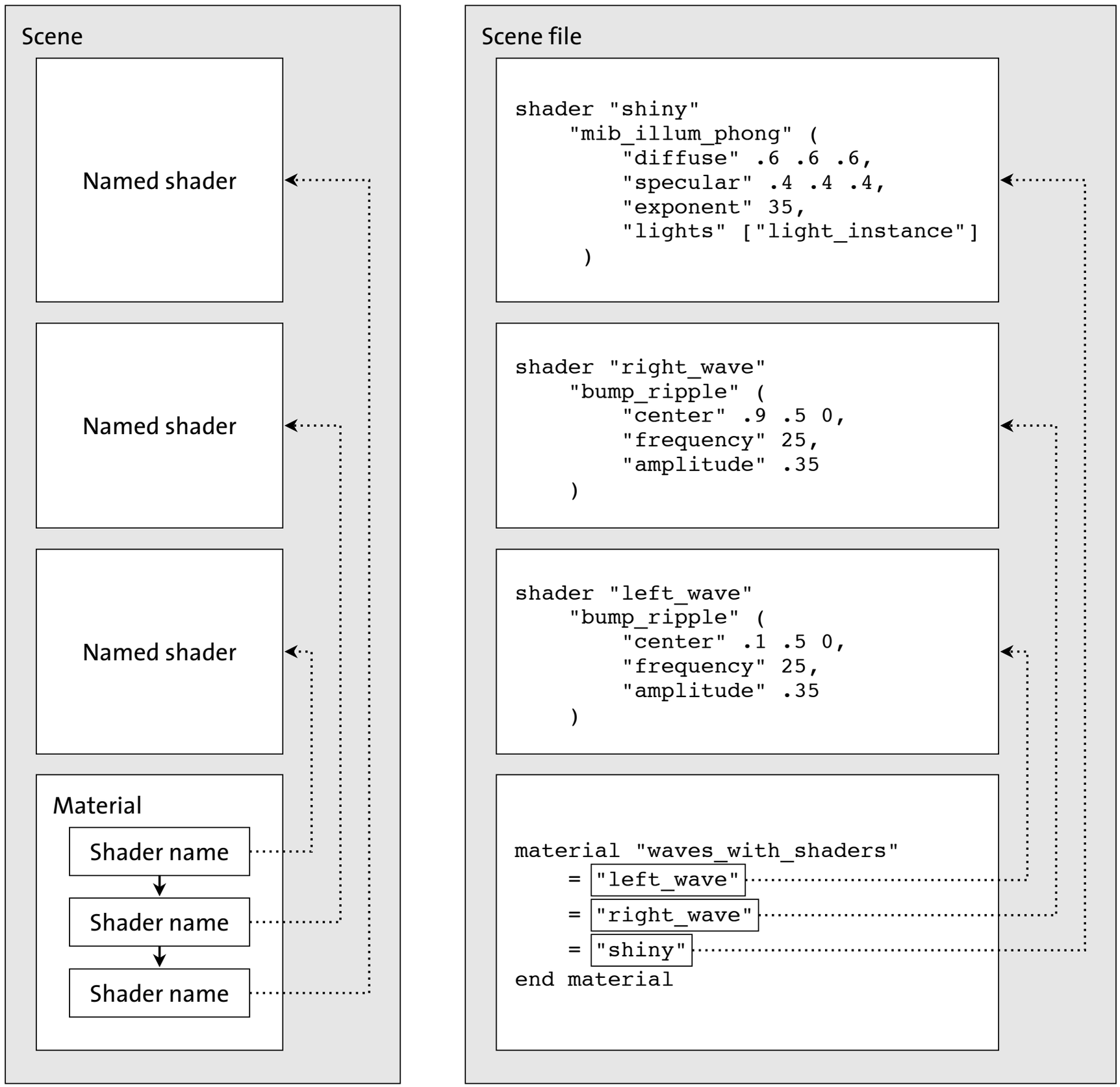
Chaining shaders of the same result type: shader lists

```
material "waves-4"
    "bump_ripple" (
        "center" .1 .5 0,
        "frequency" 25,
        "amplitude" .35
    )
    "bump_ripple" (
        "center" .9 .5 0,
        "frequency" 25,
        "amplitude" .35
    )
    "mib_illum_phong" (
        "diffuse" .6 .6 .6,
        "specular" .4 .4 .4,
        "exponent" 35,
        "lights" ["light_instance"]
     )
end material
```

Using a bump mapping shader twice with different parameters

Shaders in the scene

Chaining shaders of the same result type: shader lists



A list of named shaders in a material

Shaders in the scene

Scene

Named shader

Material

Anonymous shader

Anonymous shader

Shader name

Chaining shaders of the same result type: shader lists

Scene file

```
shader "shiny"
    "mib_illum_phong" (
        "diffuse" .6 .6 .6,
        "specular" .4 .4 .4,
        "exponent" 35,
        "lights" ["light_instance"]
     )
```

```
material "waves_mixed"
    "bump_ripple" (
        "center" .1 .5 0,
        "frequency" 25,
        "amplitude" .35
    )
    "bump_ripple" (
        "center" .9 .5 0,
        "frequency" 25,
        "amplitude" .35
    )
    = "shiny"
end material
```

A shader list in a material containing both anonymous and named shaders

Using shaders in the scene file

Anonymous shaders

Shader called directly in the material

Named shaders

Shader defined for later reference

Shader lists

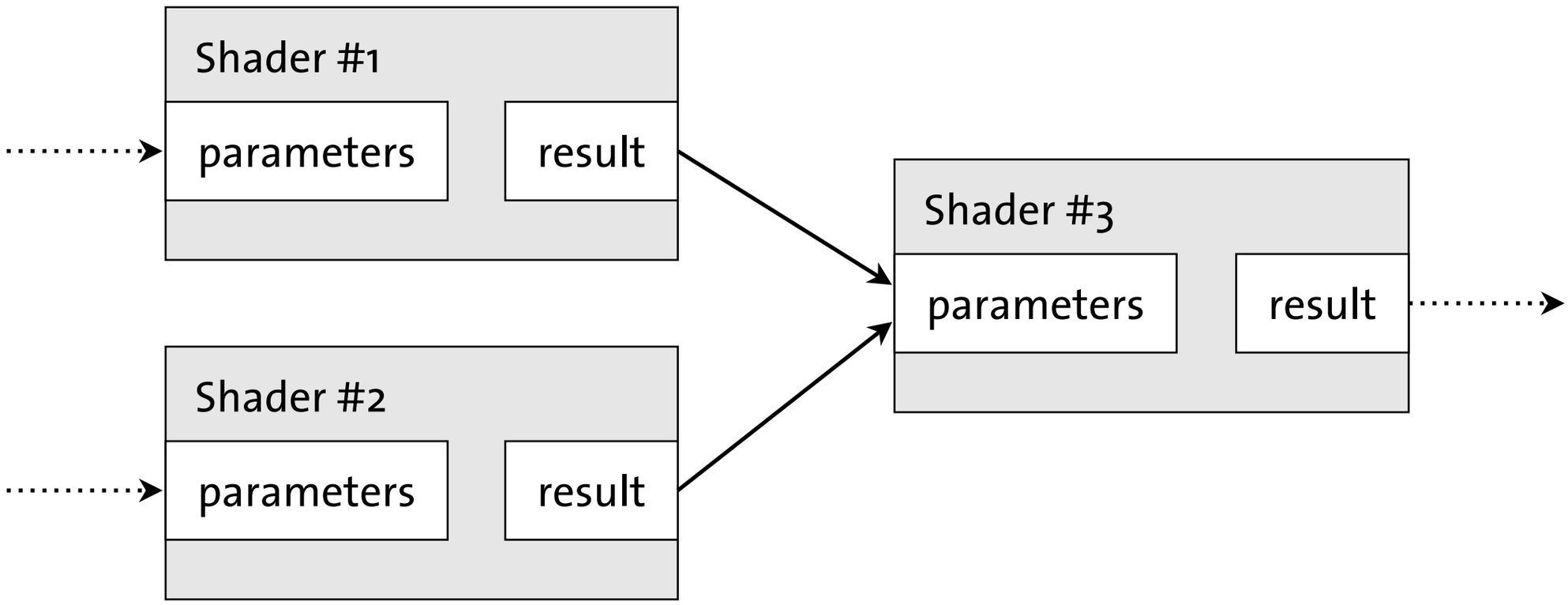
Accumulation of shader results

Shader graphs

Shaders used as input to other shaders

Phenomena

Formalized shader graphs



Shaders represented as connected nodes in a network

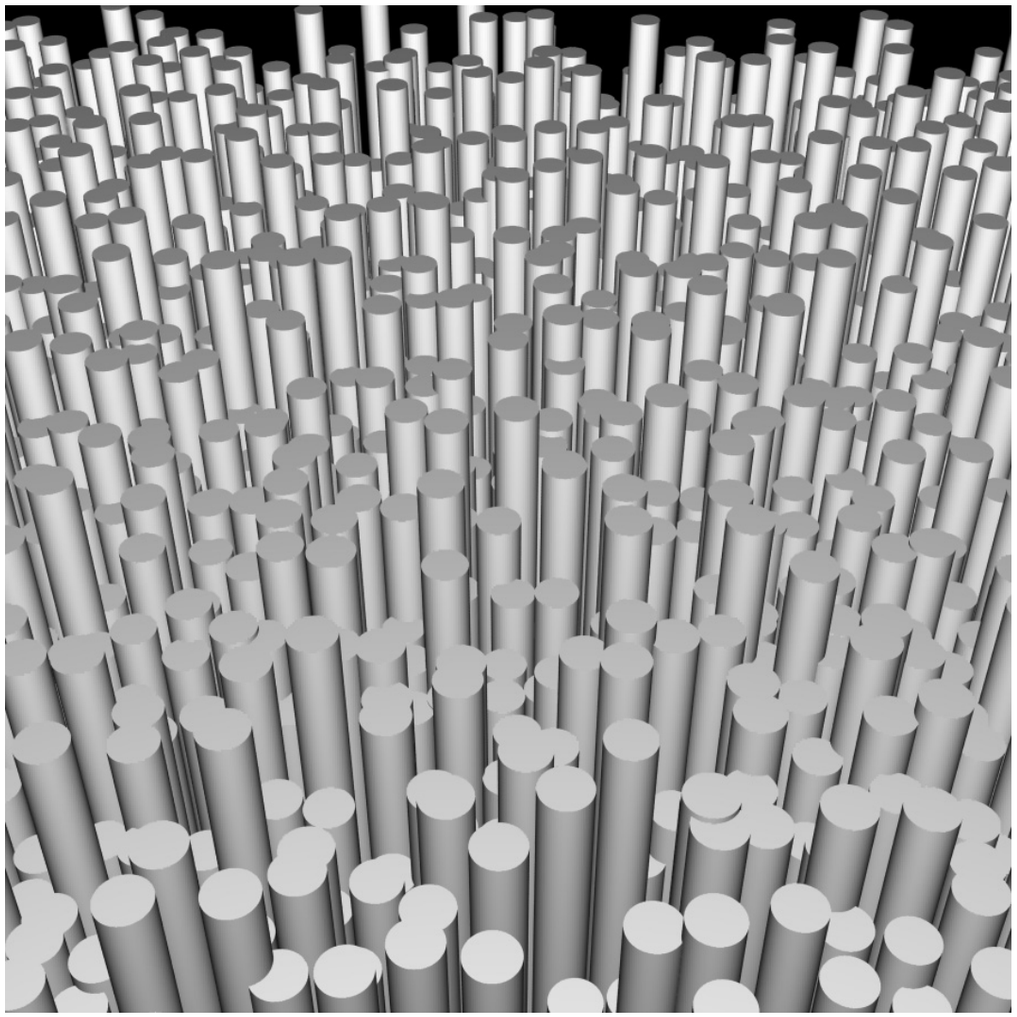
Shaders in the scene

Connecting shaders in a network: shader graphs

```
declare shader
    color "front_bright" (
        color "tint" default 1 1 1 )
end declare
```

Scene file declaration of shader "front_bright"

Shaders in the scene



Connecting shaders in a network: shader graphs

```
material "front"
    "front_bright" (
        "tint" 1.05 1.05 1.05
    )
end material
```

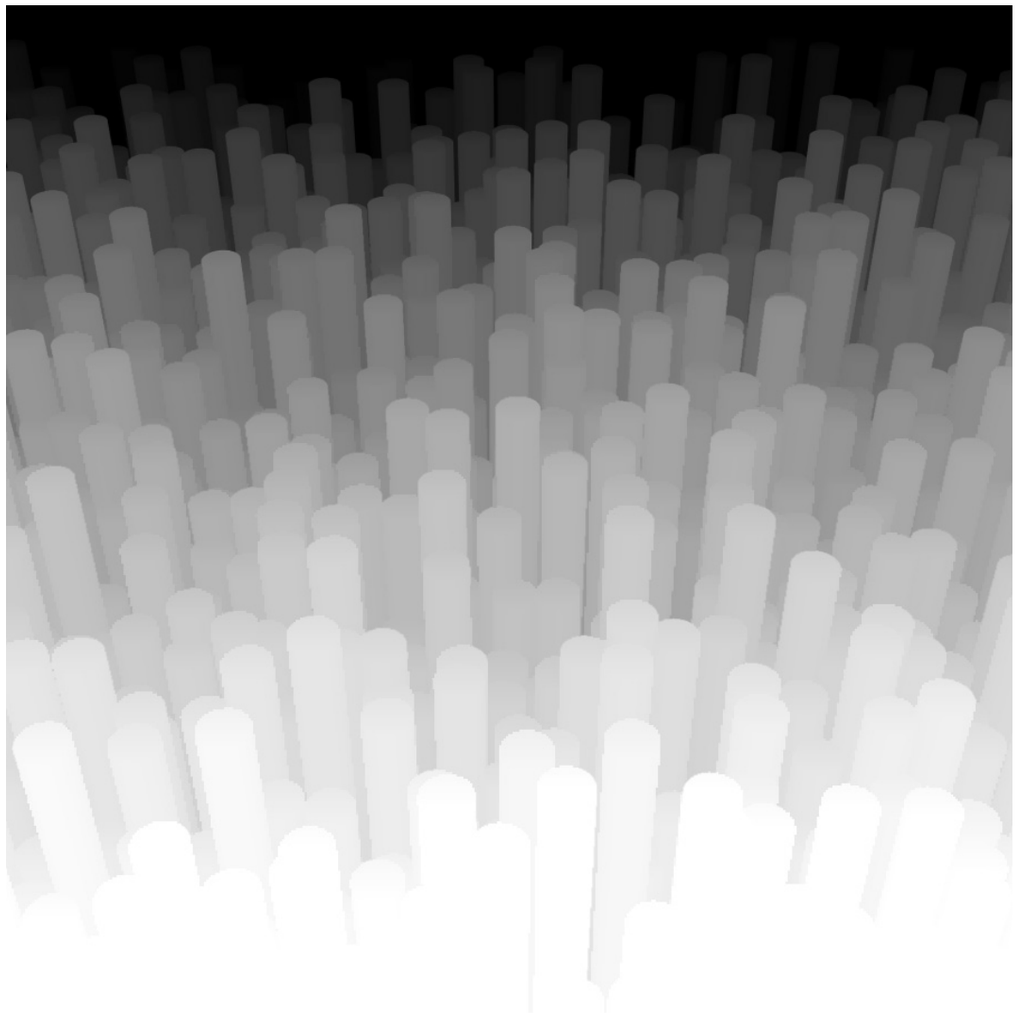
Using the front_bright shader in a material

```
declare shader
    color "depth_fade" (
        scalar "near",
        scalar "far" )
end declare
```

Scene file declaration of shader "depth_fade"

Shaders in the scene

Connecting shaders in a network: shader graphs

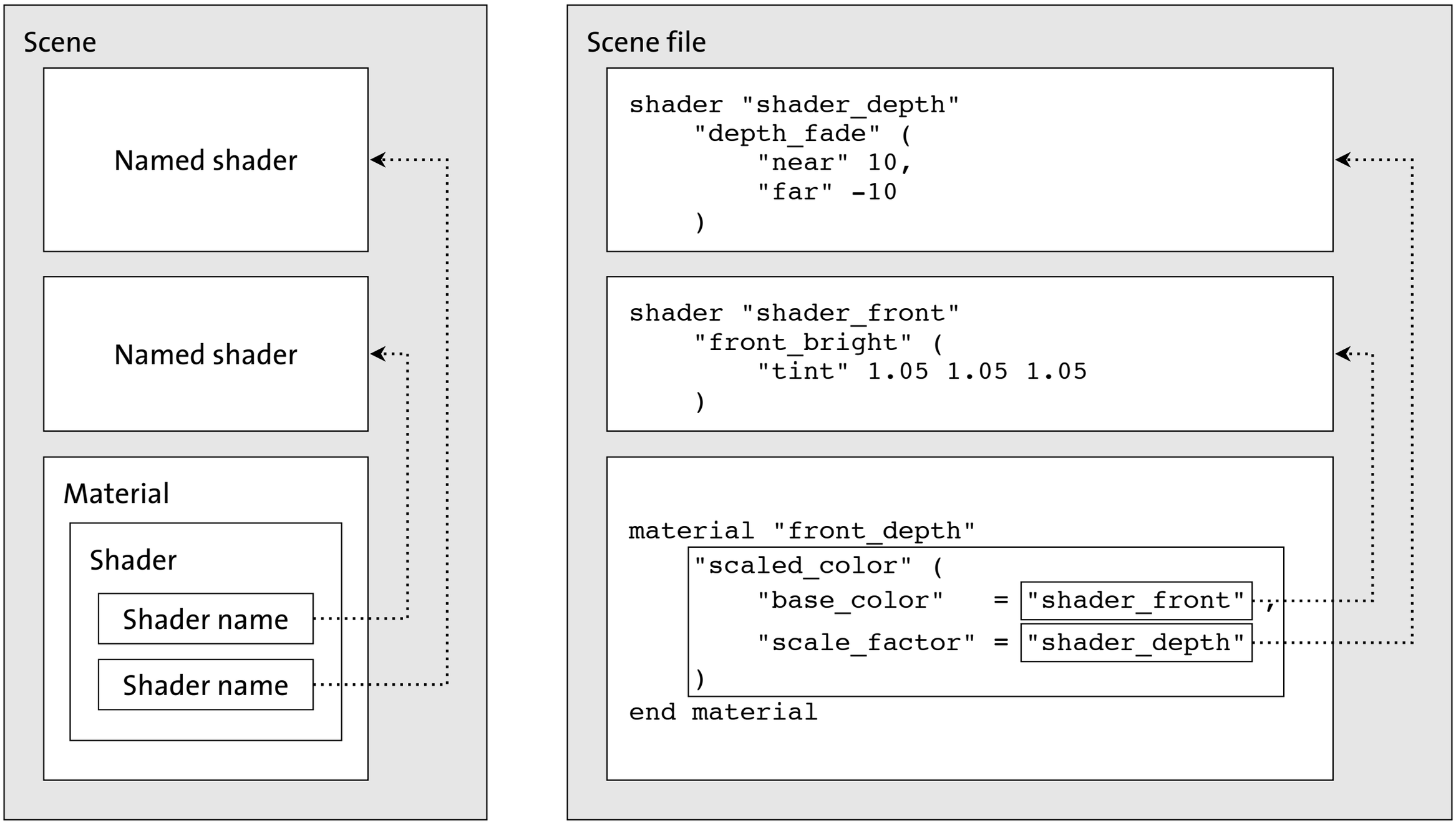


```
material "depth"
    "depth_fade" (
        "near" 10,
        "far" -10
    )
end material
```

Using the depth_fade shader in a material

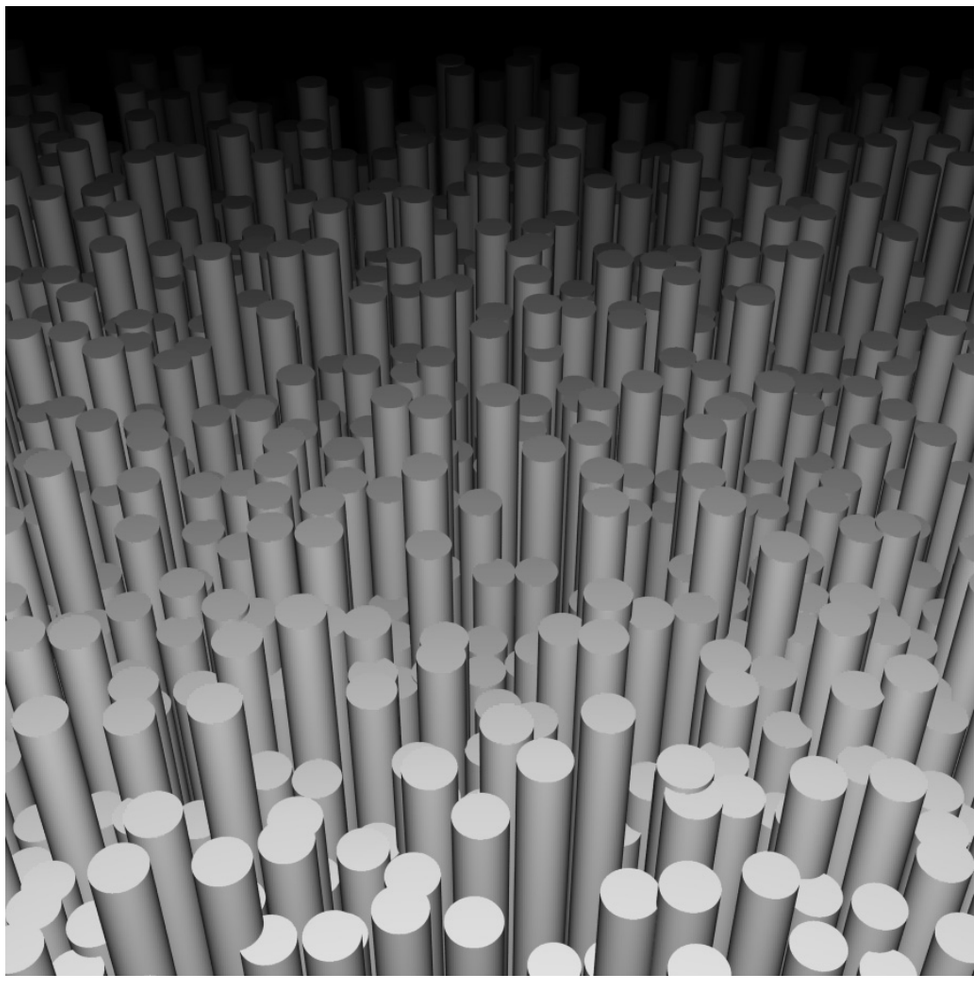
Shaders in the scene

Connecting shaders in a network: shader graphs



Shaders front_bright and depth_fade as parameter values for scaled_color

Shaders in the scene

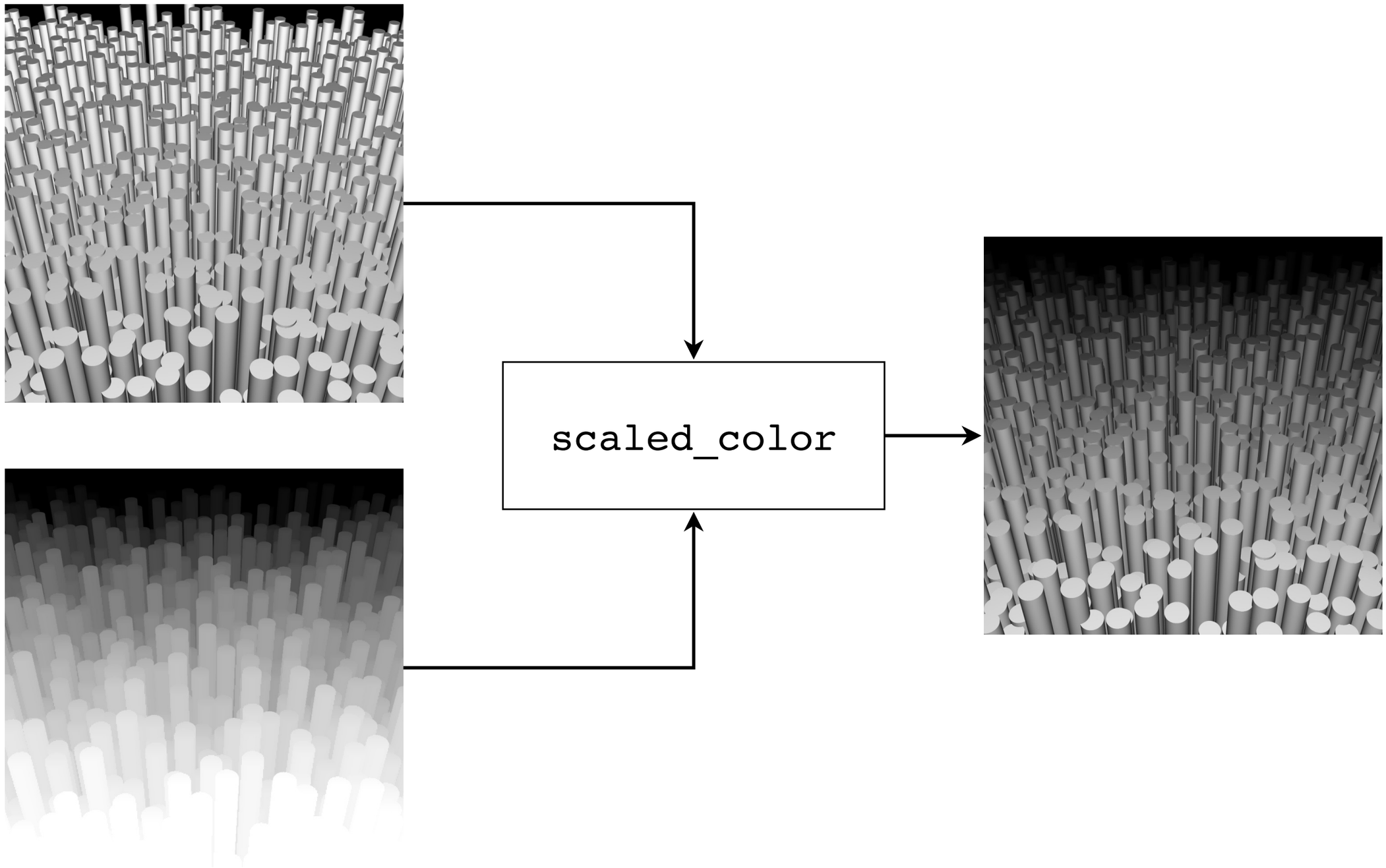


Connecting shaders in a network: shader graphs

```
material "front_depth"
    "scaled_color" (
        "base_color"   = "shader_front",
        "scale_factor" = "shader_depth"
    )
end material
```

Rendered image using shaders as parameter values

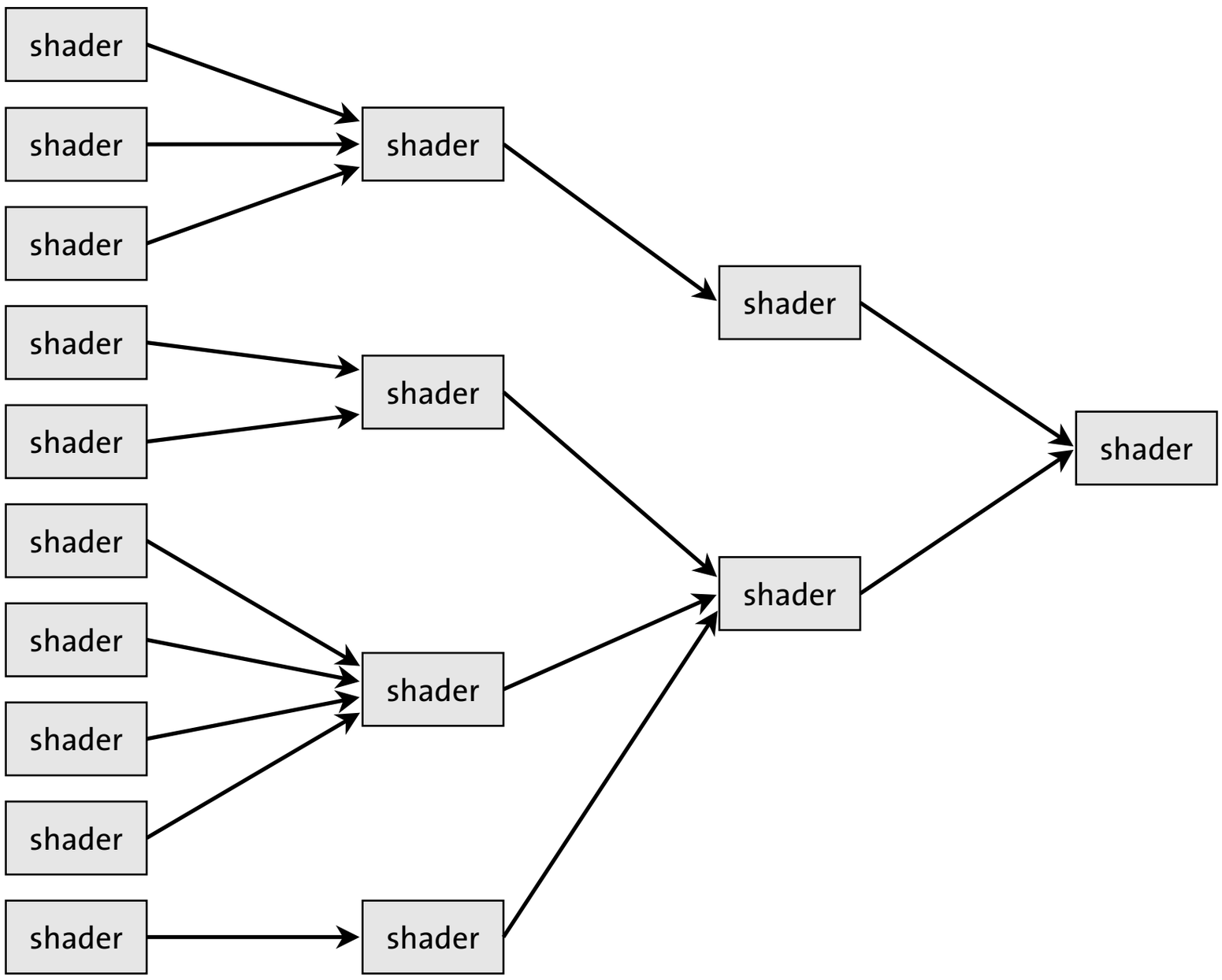
Connecting shaders in a network: shader graphs



Shader combinations as a compositing operation

Shaders in the scene

Connecting shaders in a network: shader graphs



A graph consisting of many connected shaders

Using shaders in the scene file

Anonymous shaders

Shader called directly in the material

Named shaders

Shader defined for later reference

Shader lists

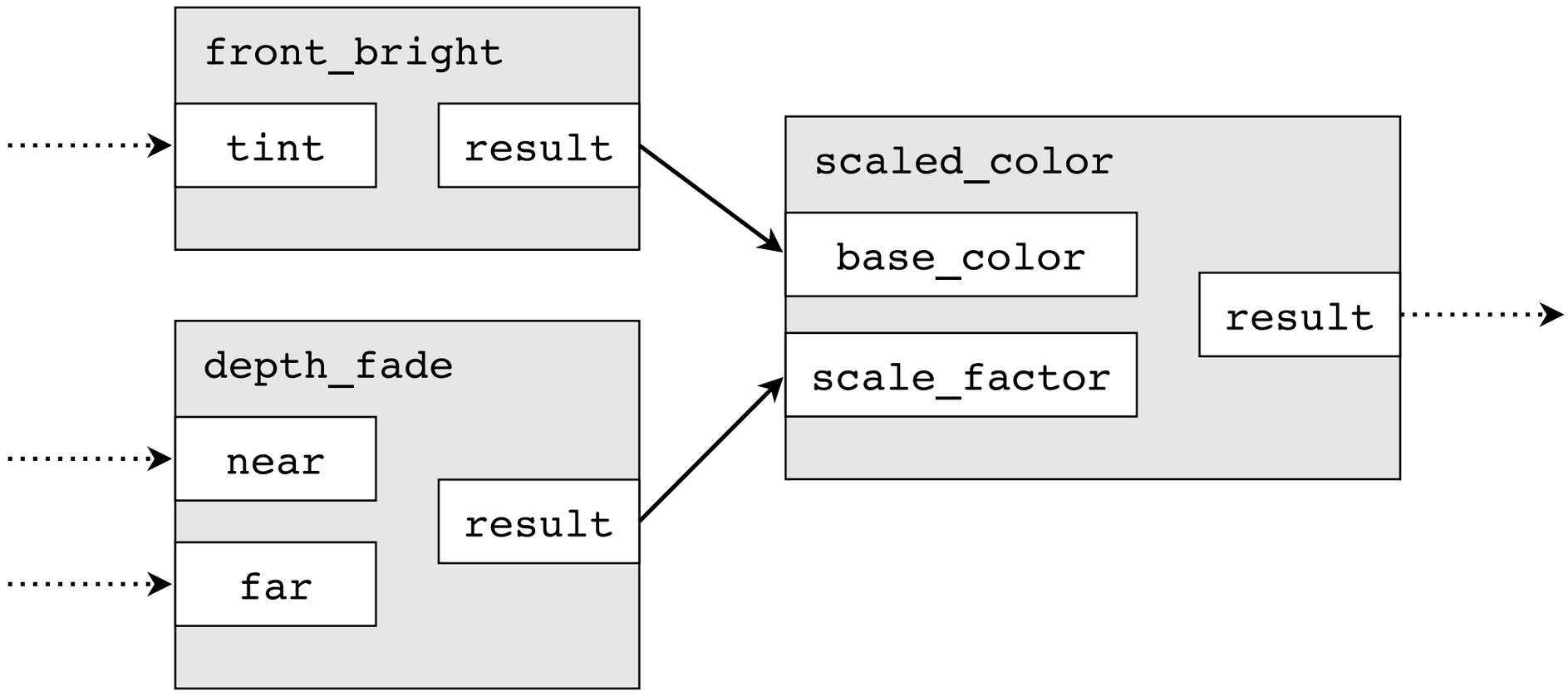
Accumulation of shader results

Shader graphs

Shaders used as input to other shaders

Phenomena

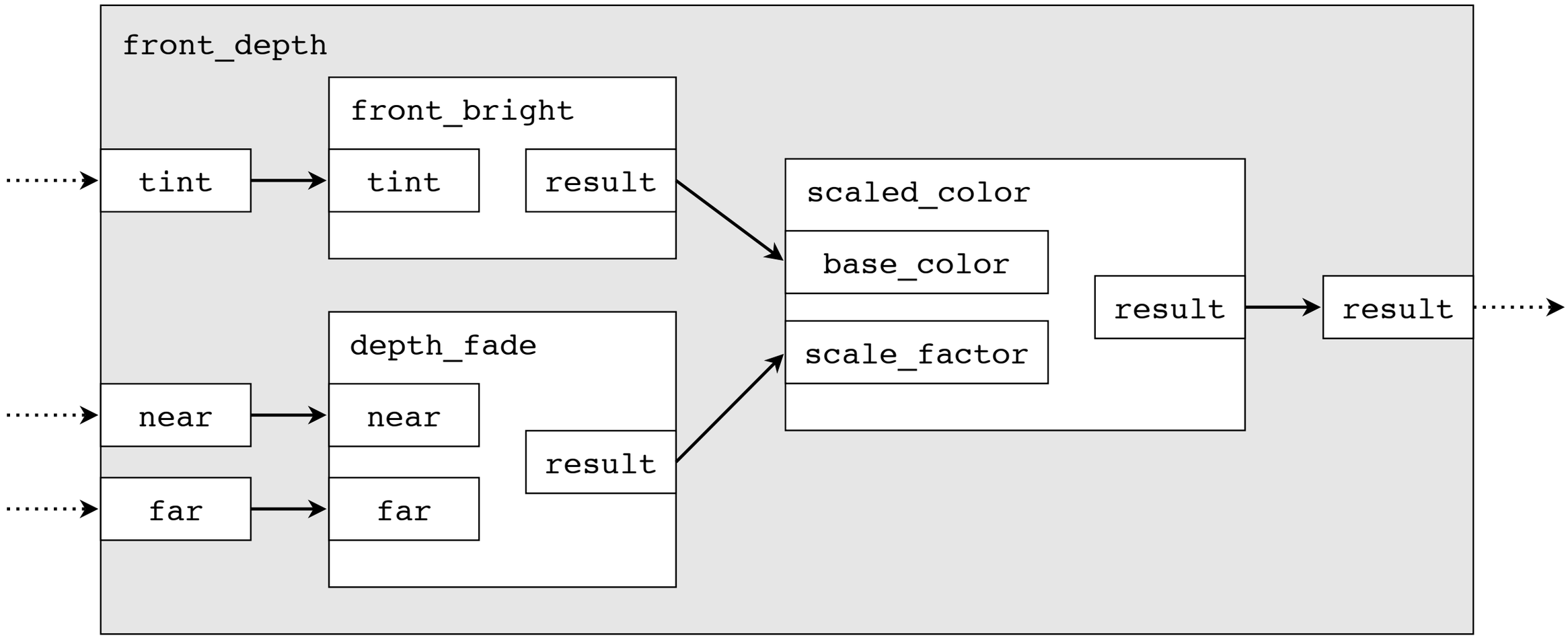
Formalized shader graphs



Shaders connected in a graph

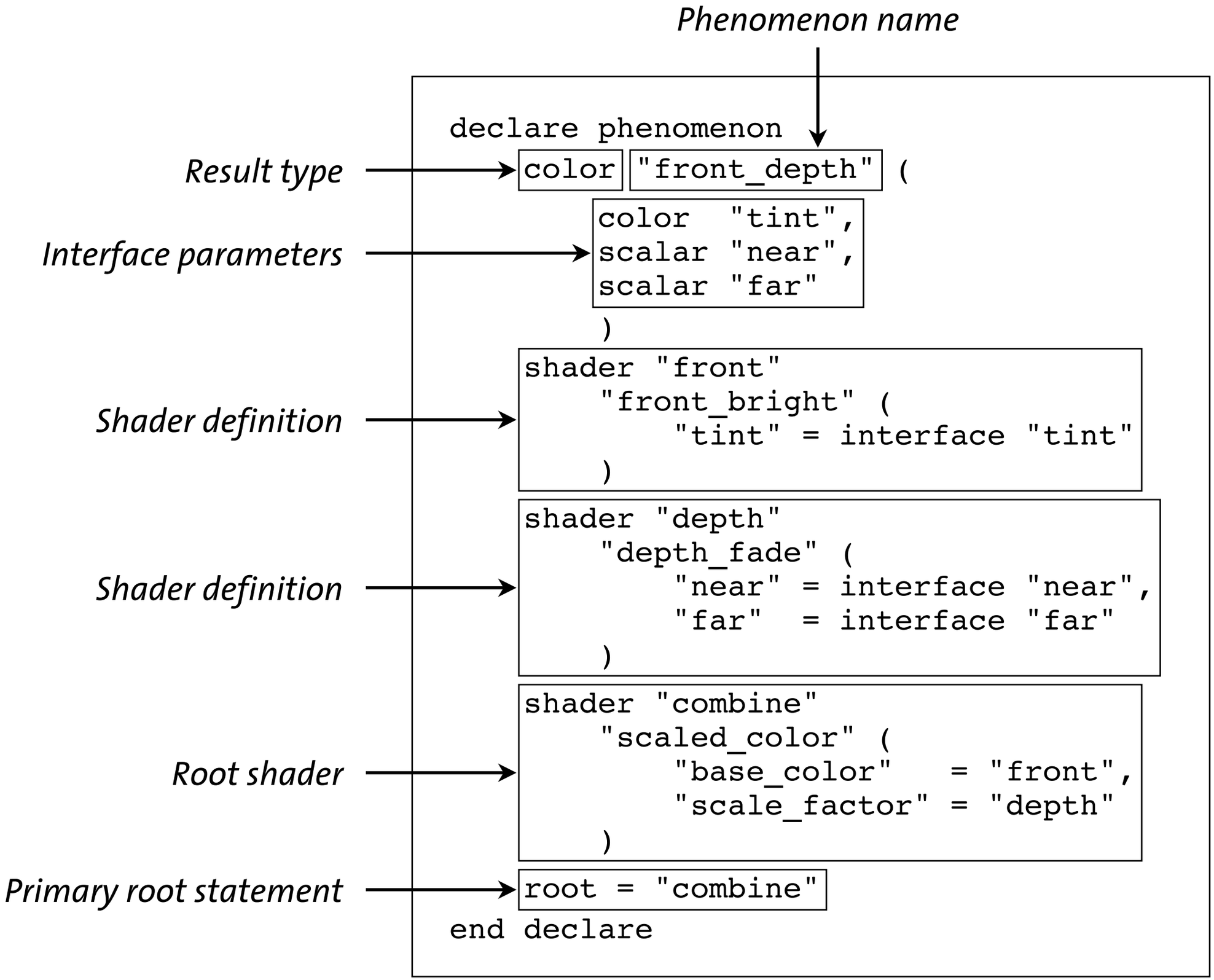
Shaders in the scene

Named shader graphs: Phenomena



Shaders connected in a graph as part of a Phenomenon called front_depth

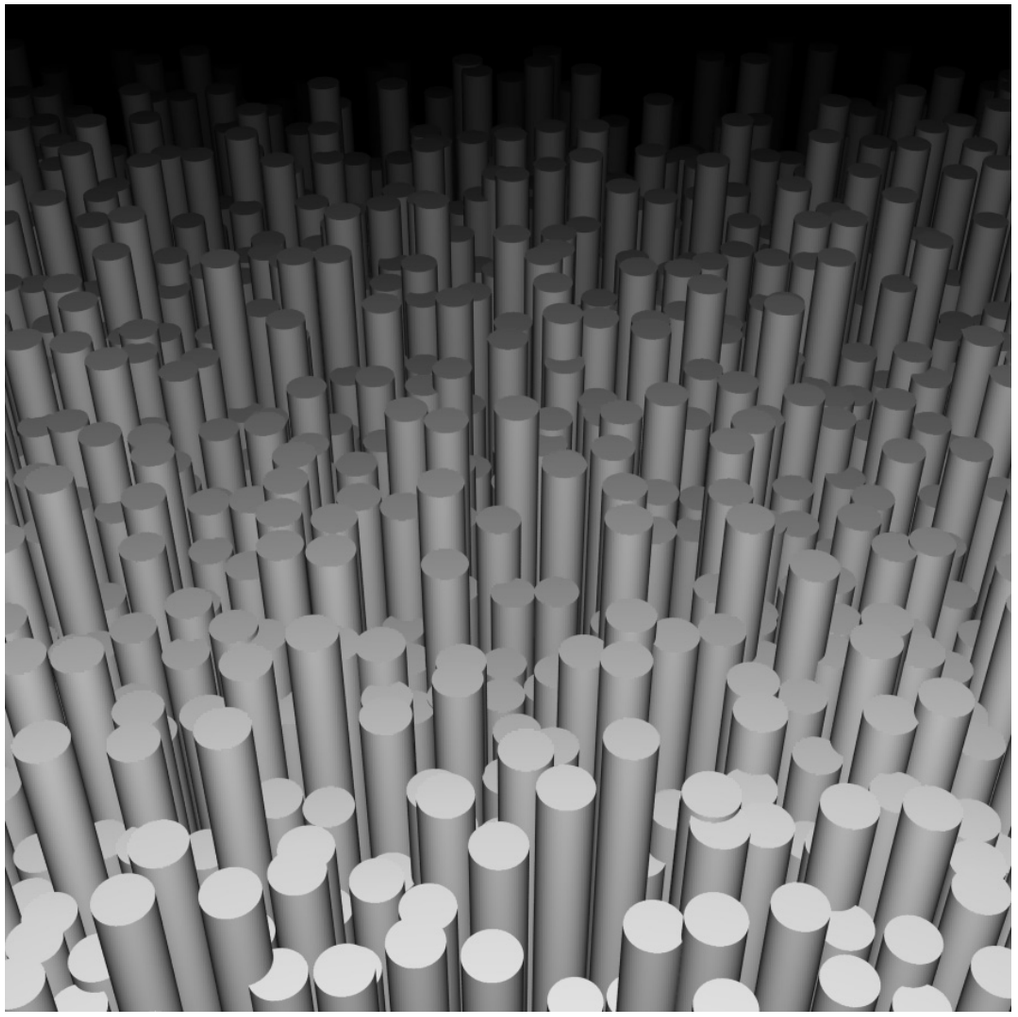
Named shader graphs: Phenomena



Declaration of a Phenomenon

Shaders in the scene

Named shader graphs: Phenomena



```
material "column_fade"
    "front_depth" (
        "tint" 1.05 1.05 1.05,
        "near" 10,
        "far" -10
    )
end material
```

Using the Phenomenon in a material

Creating a material with a Phenomenon

Creating a material with a Phenomenon

- Phenomena so far have functioned as shaders

Creating a material with a Phenomenon

- Phenomena so far have functioned as shaders
- A *material Phenomenon* defines a procedural material

Creating a material with a Phenomenon

- Phenomena so far have functioned as shaders
- A *material Phenomenon* defines a procedural material
- Can include all the shader types of a material

Creating a material with a Phenomenon

- Phenomena so far have functioned as shaders
- A *material Phenomenon* defines a procedural material
- Can include all the shader types of a material
- Useful for *encapsulating* interrelated shaders

Creating a material with a Phenomenon

- Phenomena so far have functioned as shaders
- A *material Phenomenon* defines a procedural material
- Can include all the shader types of a material
- Useful for *encapsulating* interrelated shaders
  - Material, shadow and photon shaders in one unit

Shaders in the scene

Named shader graphs: Phenomena

Material type declaration →

Material definition →

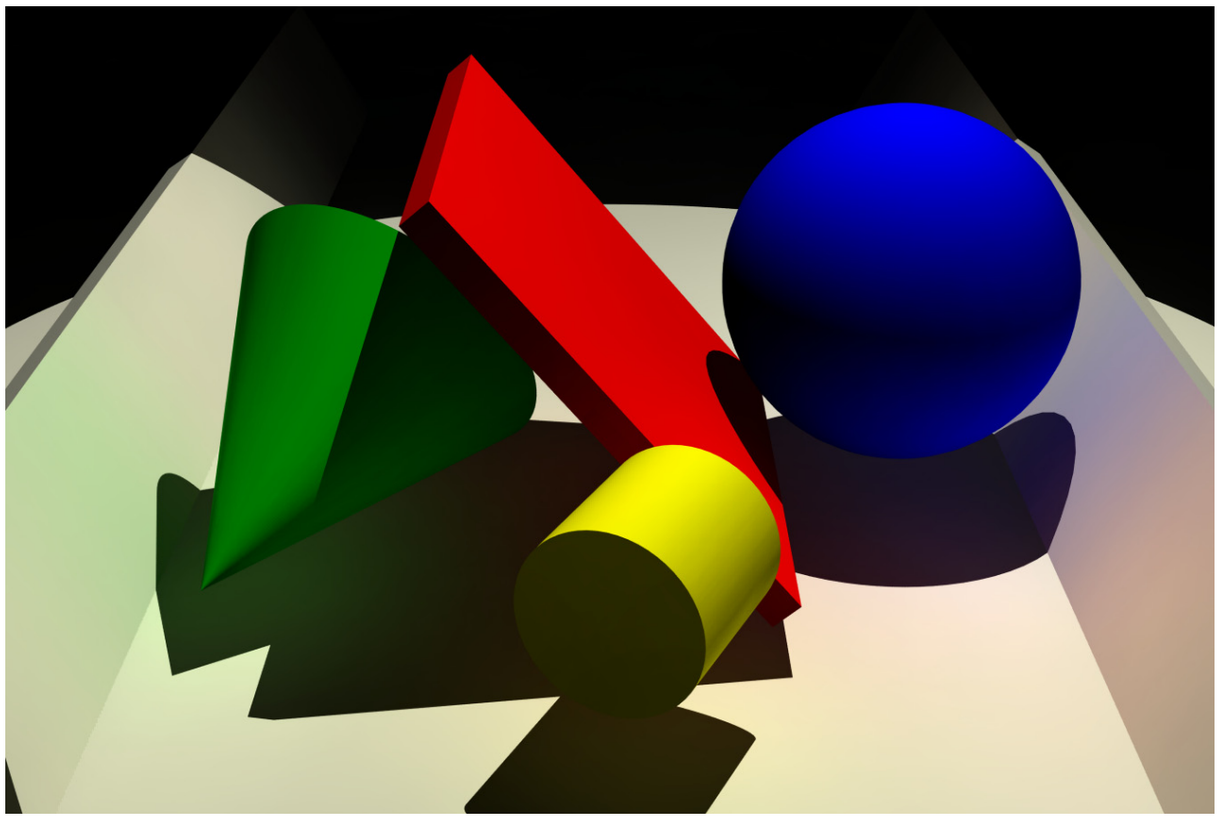
Root material →

```
declare phenomenon
    material "global_diffuse" (
        color "diffuse" default 1 1 1,
        array light "lights"
    )
    shader "indirect"
        "average_radiance" ()
    shader "indirect_diffuse"
        "scaled_color" (
            "base_color" = interface "diffuse",
            "scale_factor" = "indirect"
        )
    material "lambert_with_photons"
        "lambert" (
            "diffuse" = interface "diffuse",
            "ambient" = "indirect_diffuse",
            "lights" = interface "lights"
        )
        photon
            "store_diffuse_photon" (
                "diffuse_color" = interface "diffuse"
            )
    end material
    root material "lambert_with_photons"
end declare
```

Declaration of material Phenomenon global_diffuse that includes a photon shader

Shaders in the scene

Named shader graphs: Phenomena



```
shader "red"
    "global_diffuse" (
        "diffuse" 1 0 0,
        "lights" ["light_inst"] )

instance "block_inst" "block"
    material "red"
    transform
        1.81818 0 0 0
        0 0 -1.81818 0
        0 1.81818 0 0
        -1.63636 0.545455 0 1
end instance
```

Using a shader created from a material Phenomenon for the material of an instance