Writing mental ray shaders

by Andy Kopra

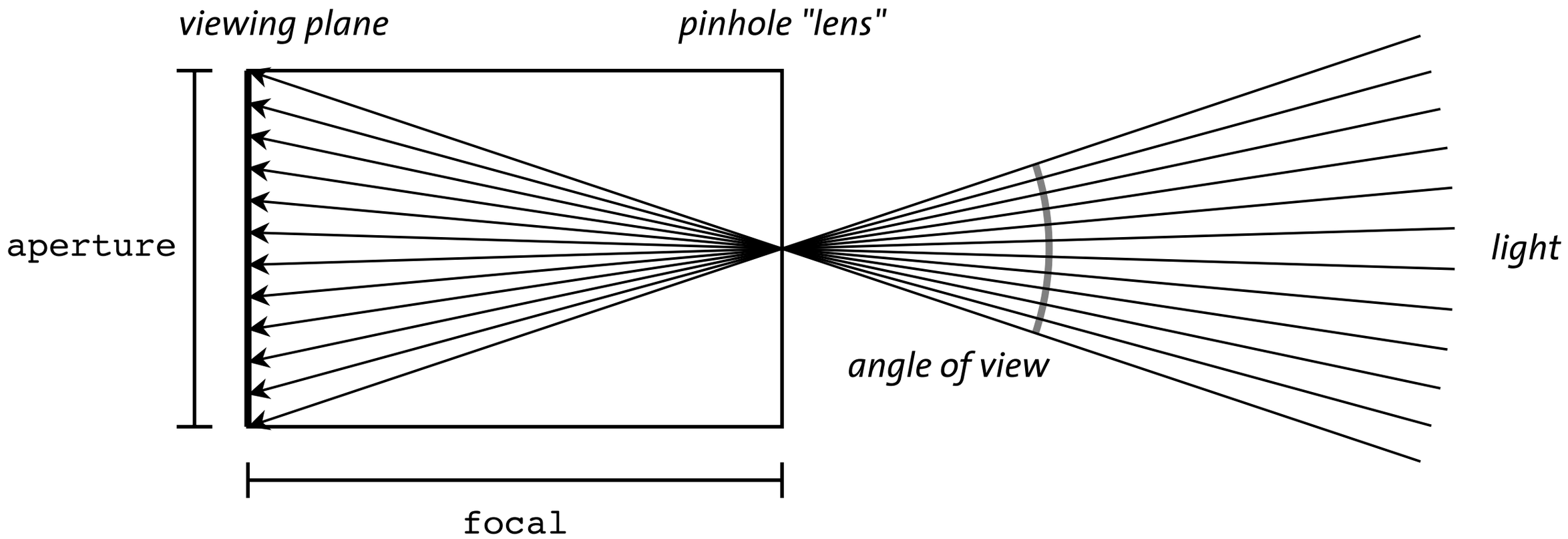
Part 6: Image

Changing the lens

Changing the lens

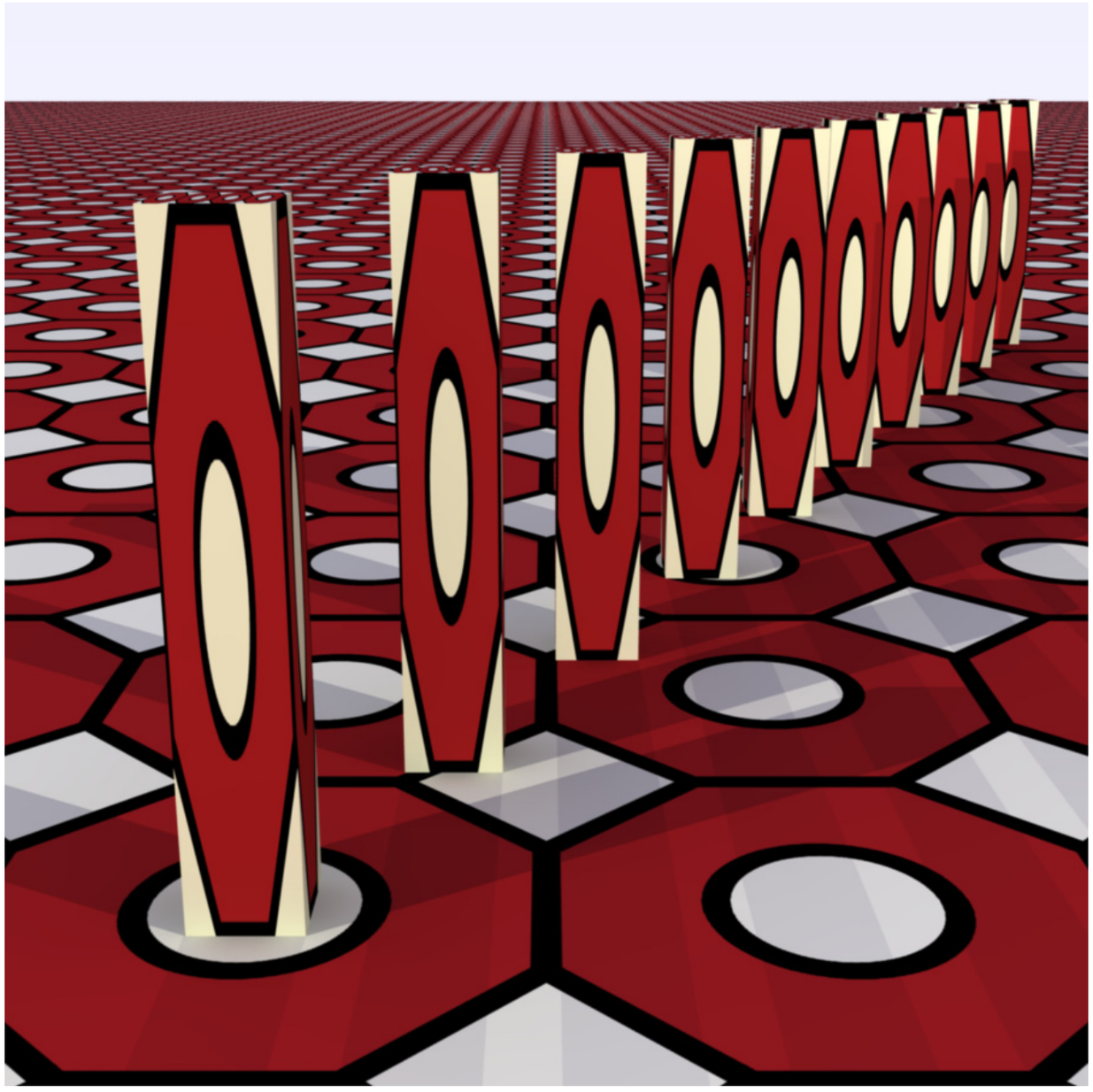
A conceptual model of a camera
Shifting the lens position
Mapping the scene around the camera to a rectangle
A fisheye lens
Depth of field
Multiple lens shaders

Changing the lens



Pinhole camera model as seen from above with aperture and focal attributes

Changing the lens

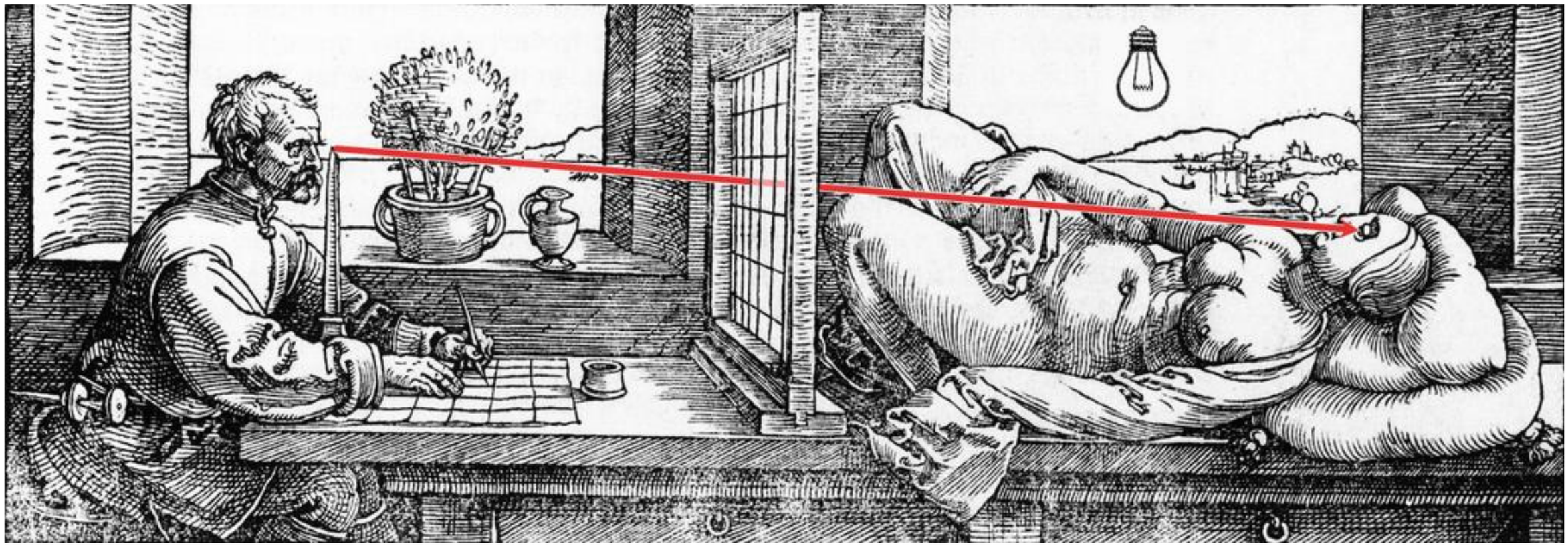


```
shader "sky"
    "color_ramp" (
        "colors" [ 0 0 0 0,
                   0 0 0 .47,
                  .95 .95 1 .48,
                  .1 .1 .6 1 ] )

camera "camera"
    output "rgba" "tif"
        "lens_1.tif"
    focal 1.5
    aperture 1
    aspect 1
    resolution 300 300
    environment = "sky"
end camera
```

Scene rendered with camera defined by its geometrical properties and an environment shader

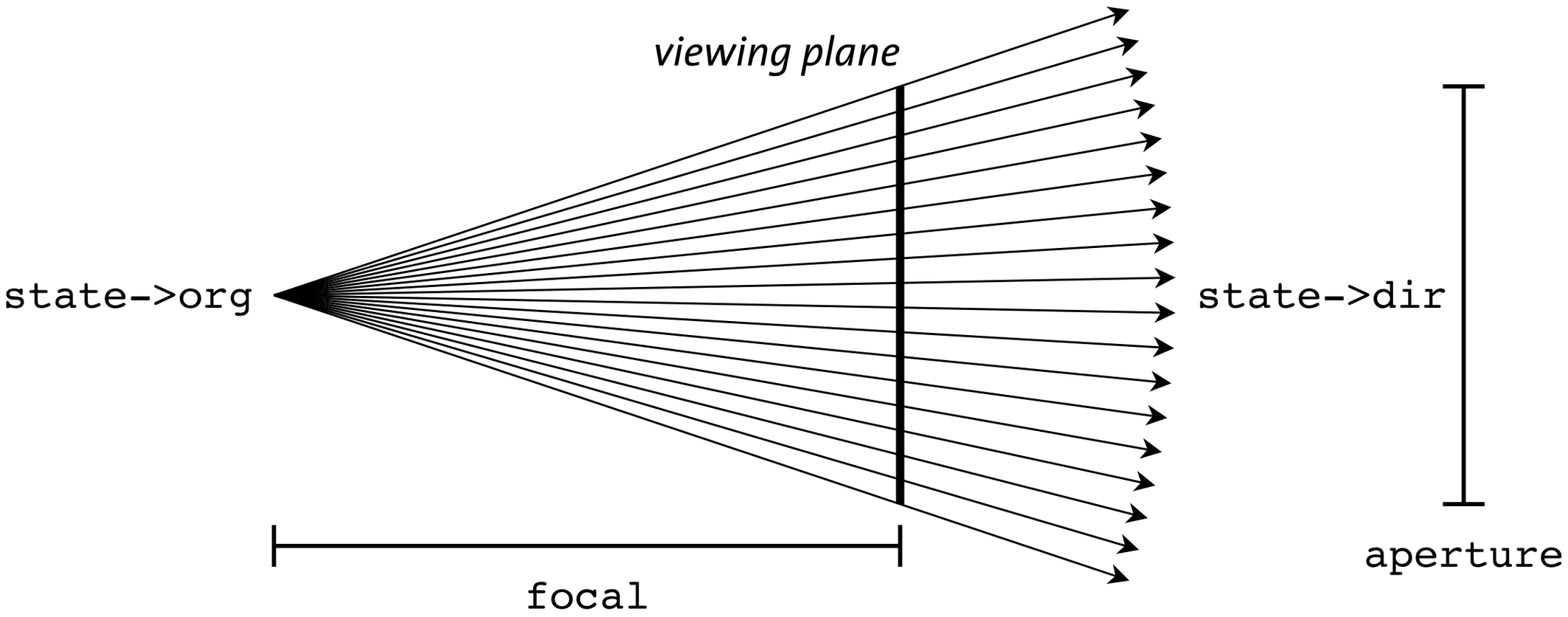
Changing the lens

A conceptual model of a camera



Calculating the color of a point projected on a screen in front of the observer

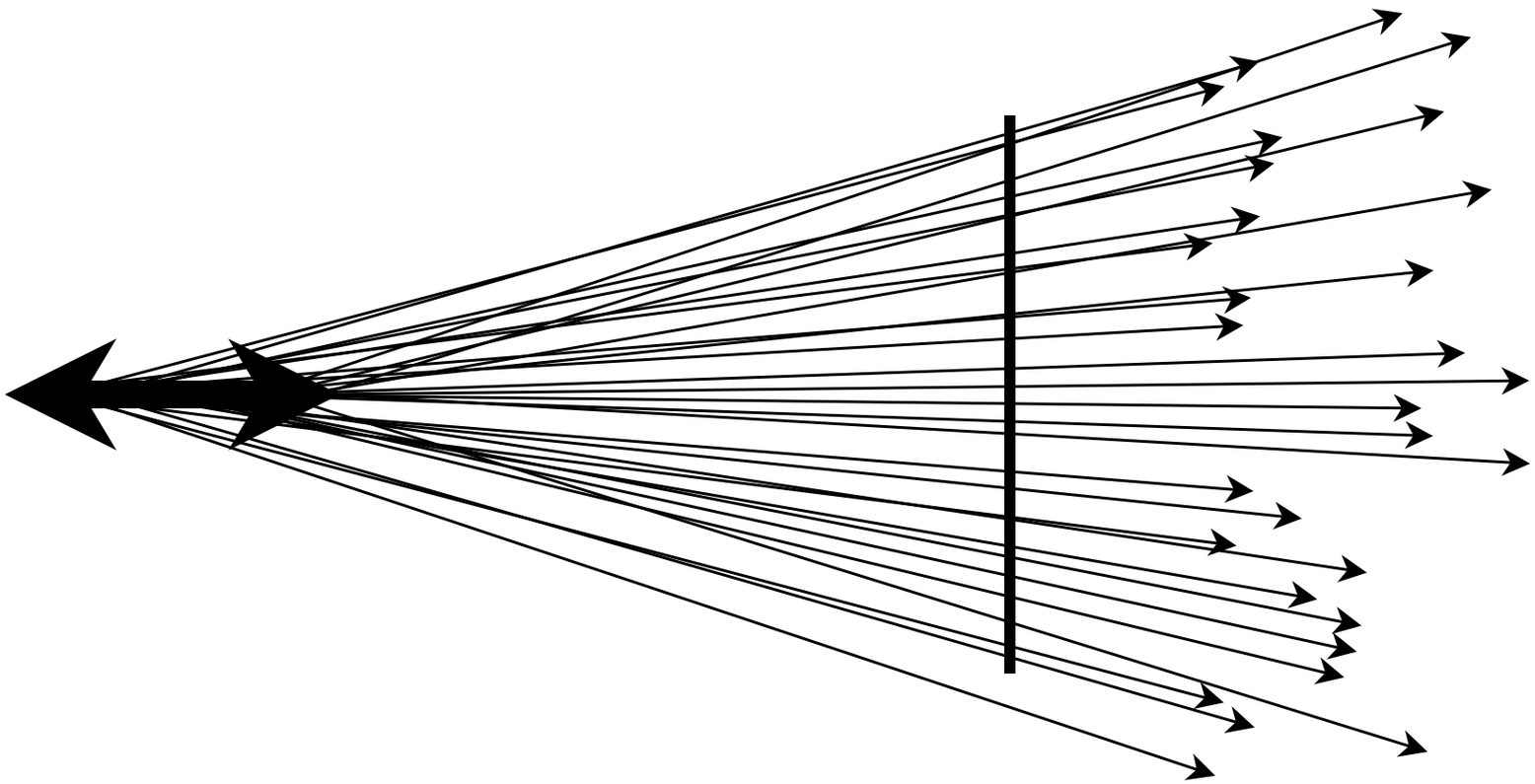
Changing the lens



Eye rays starting at code{state->org} and traveling in direction code{state->dir}, seen from above

Changing the lens

Shifting the lens position



Moving the origin of the eye ray by redefining its z component

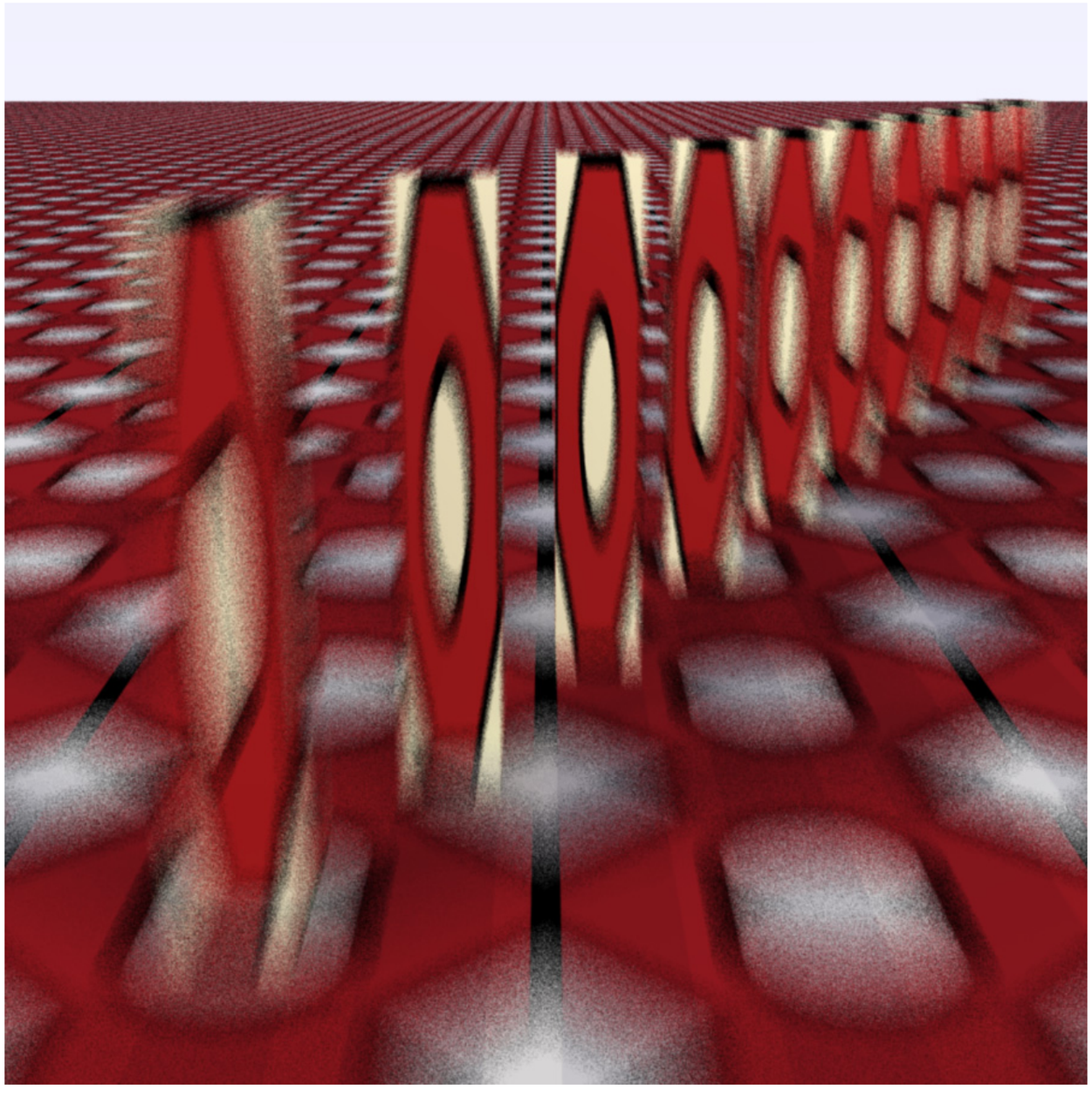
Changing the lens

Shifting the lens position

```
declare shader
    color "streak" (
        scalar "max_distance" default .1 )
    apply lens
    scanline off
    trace on
end declare
```

Scene file declaration of shader "streak"

Changing the lens



Shifting the lens position

```
camera "camera"
    output "rgba" "tif"
        "lens_2.tif"
    focal 1.5
    aperture 1
    aspect 1
    resolution 300 300
    environment = "sky"
    lens
        "streak" (
            "max_distance" .2 )
end camera
```

Moving the lens position in space

Changing the lens

Shifting the lens position

```
1  miBoolean default_lens (
2      miColor *result, miState *state, void *params )
3  {
4      return mi_trace_eye(result, state, &state->org, &state->dir);
5  }
```

Default lens behavior expressed as a shader

Changing the lens

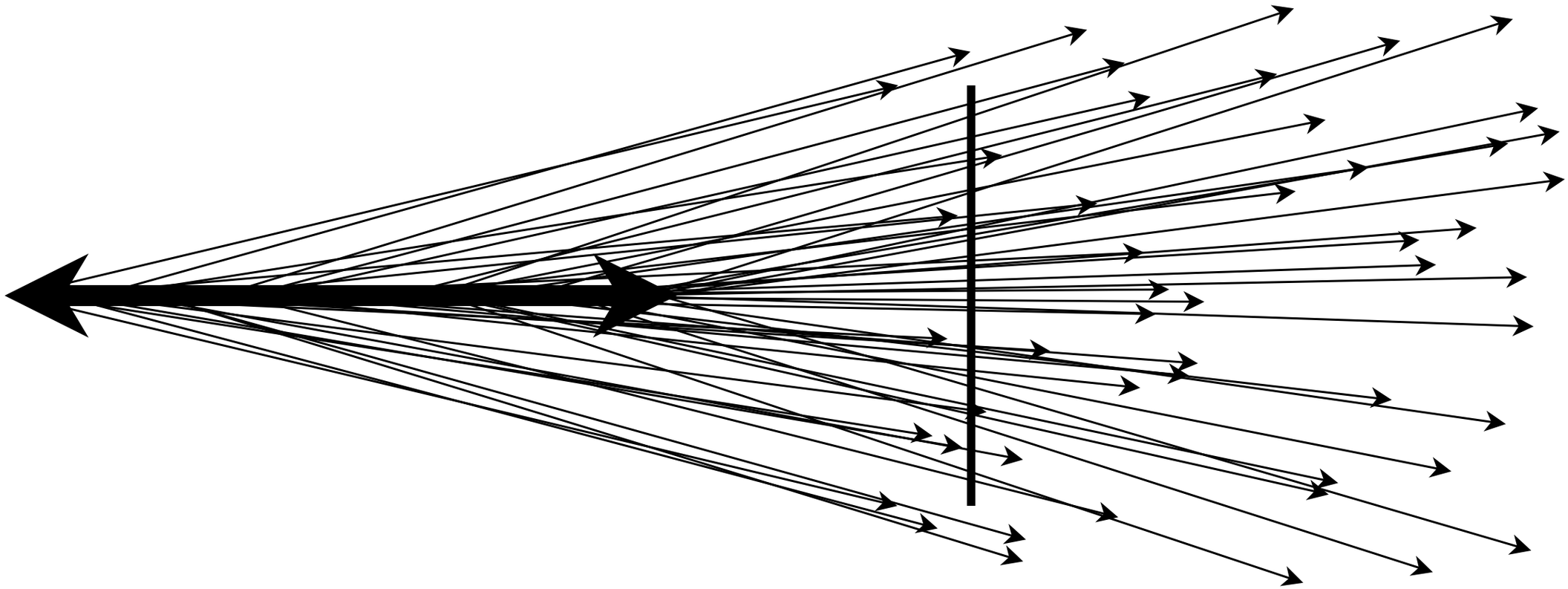
Shifting the lens position

```
1  struct streak {
2      miScalar max_distance;
3  };
4
5  miBoolean streak (
6      miColor *result, miState *state, struct streak *params )
7  {
8      miScalar max_distance = *mi_eval_scalar(&params->max_distance);
9      state->org.z += miaux_random_range(-max_distance, max_distance);
10     return mi_trace_eye(result, state, &state->org, &state->dir);
11 }
```

Source code of shader "streak"

Changing the lens

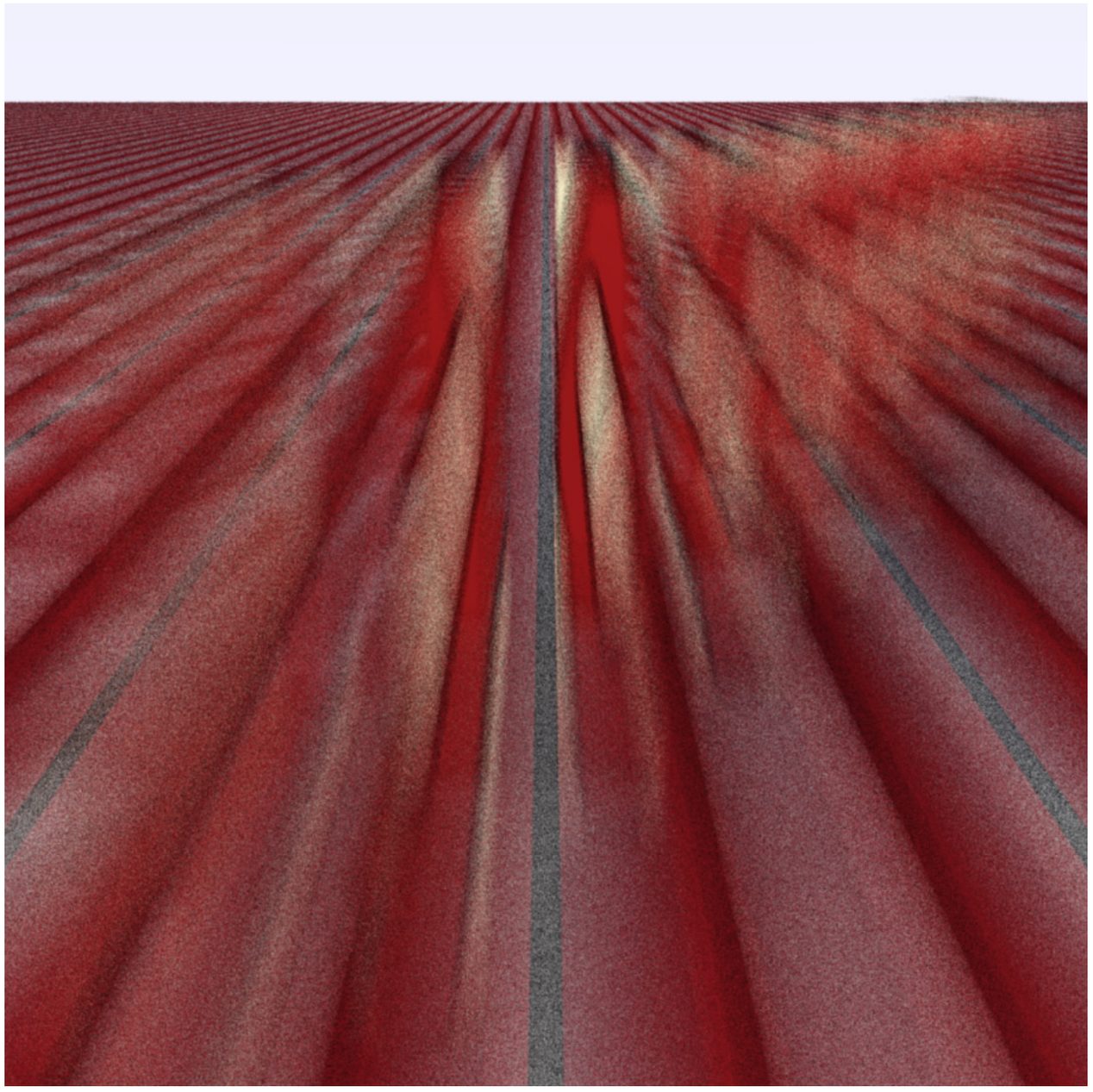
Shifting the lens position



Moving the origin of the eye ray a greater amount in z

Changing the lens

Shifting the lens position

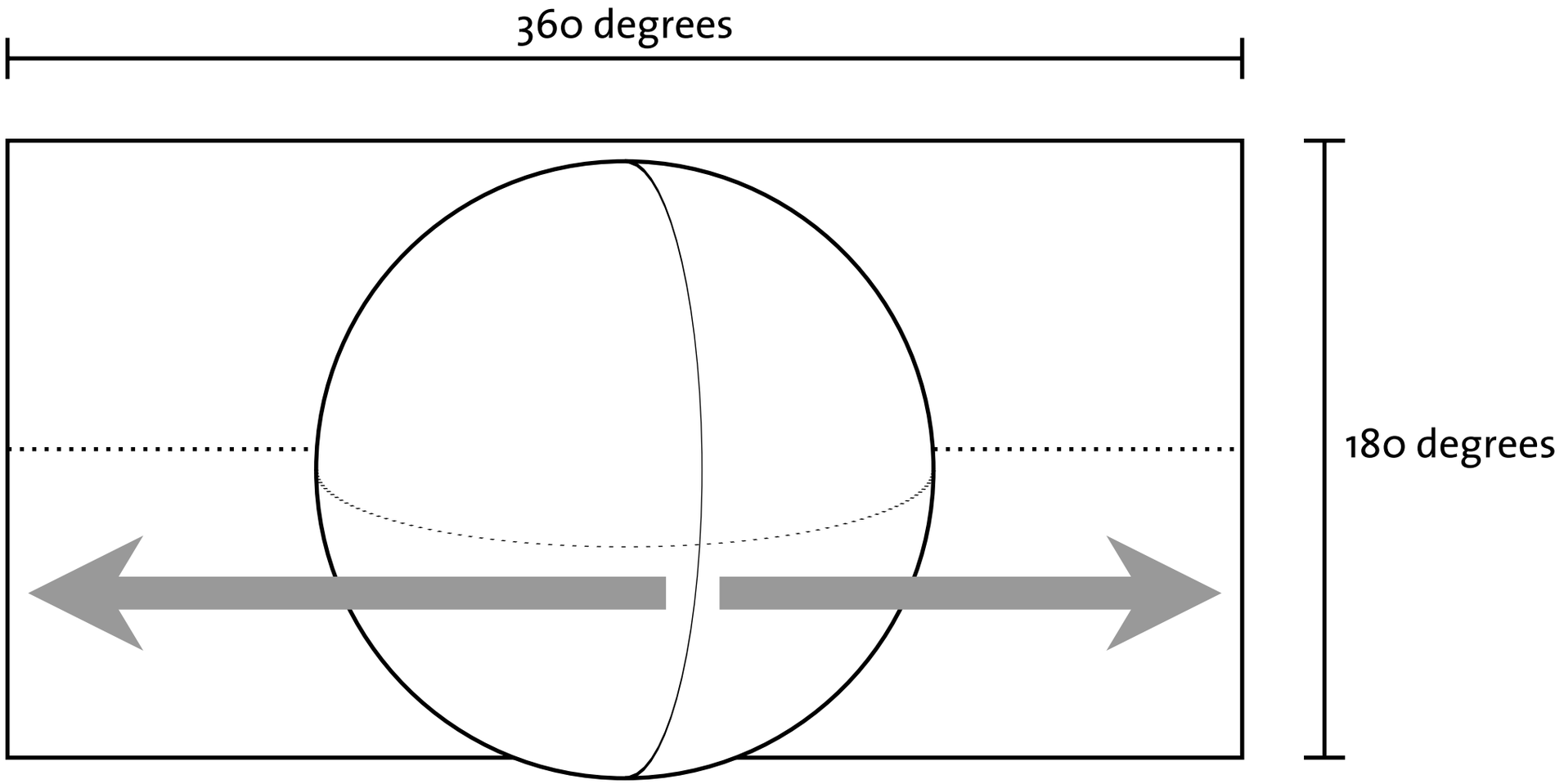


```
camera "camera"
    output "rgba" "tif"
        "lens_3.tif"
    focal 1.5
    aperture 1
    aspect 1
    resolution 300 300
    environment = "sky"
    lens
        "streak" (
            "max_distance" 1 )
end camera
```

A large value for the max_distance parameter to the streak lens shader

Changing the lens

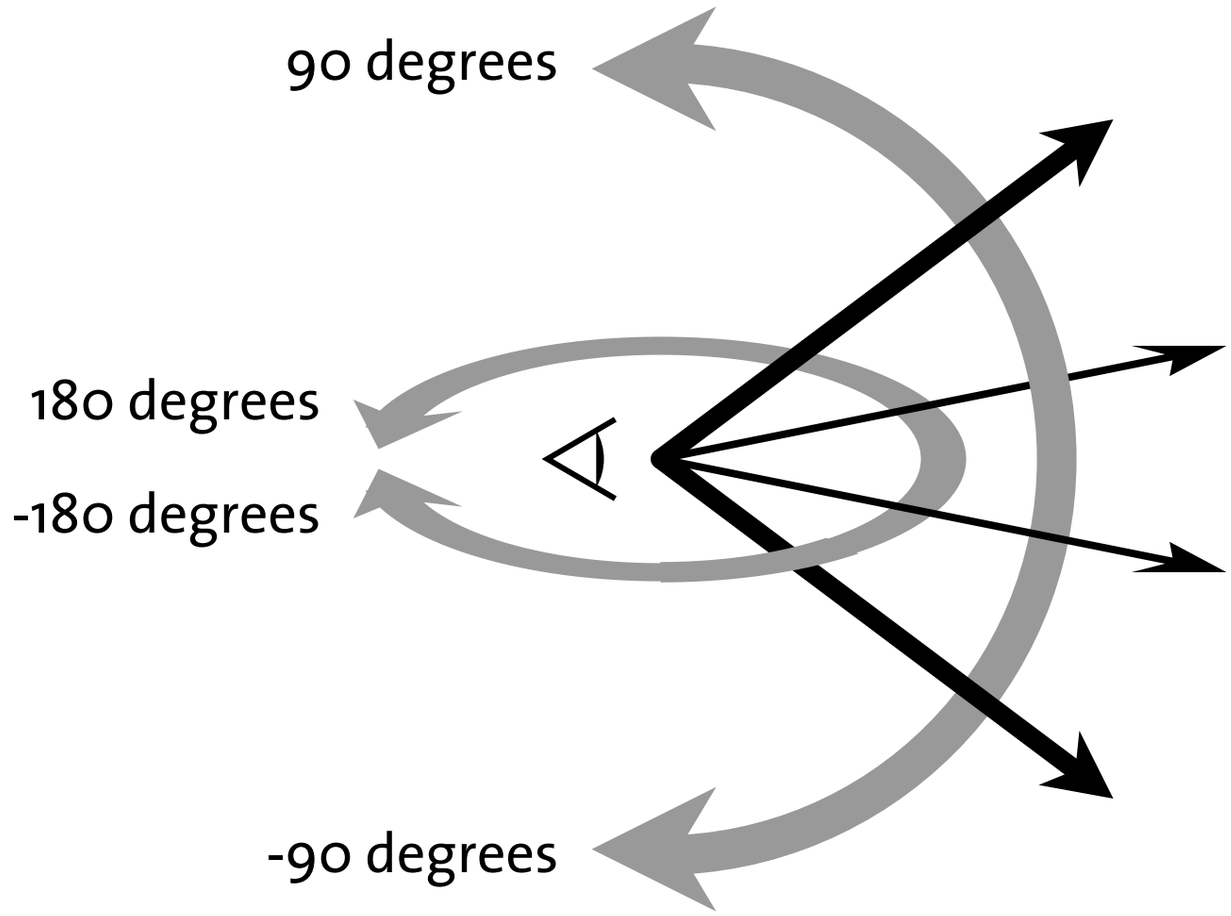
Mapping the scene around the camera to a rectangle



Representing ray directions as a sphere around the camera and their projection to a rectangle

Changing the lens

Mapping the scene around the camera to a rectangle



All possible ray directions from the camera defined by ray rotations in x and y

Changing the lens

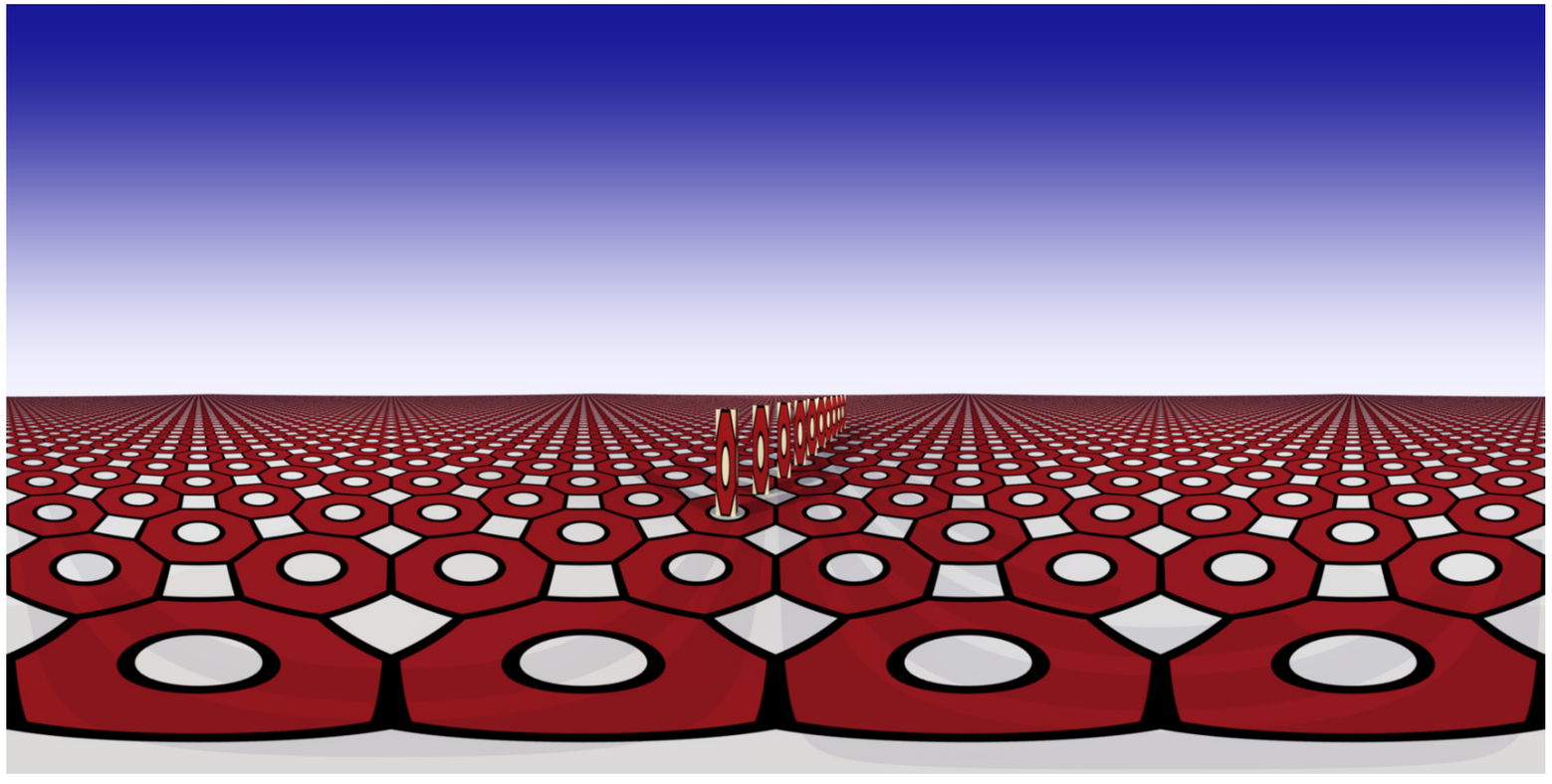
Mapping the scene around the camera to a rectangle

```
declare shader
    color "equirectangular" ()
    apply lens
    scanline off
    trace on
end declare
```

Scene file declaration of shader "equirectangular"

Changing the lens

Mapping the scene around the camera to a rectangle



```
camera "camera"
  output "rgba" "tif"
    "lens_4.tif"
  focal 1.5
  aperture 1
  aspect 1
  resolution 600 300
  environment = "sky"
  lens
    "equirectangular" ()
end camera
```

The equirectangular projection implemented in a lens shader

Changing the lens

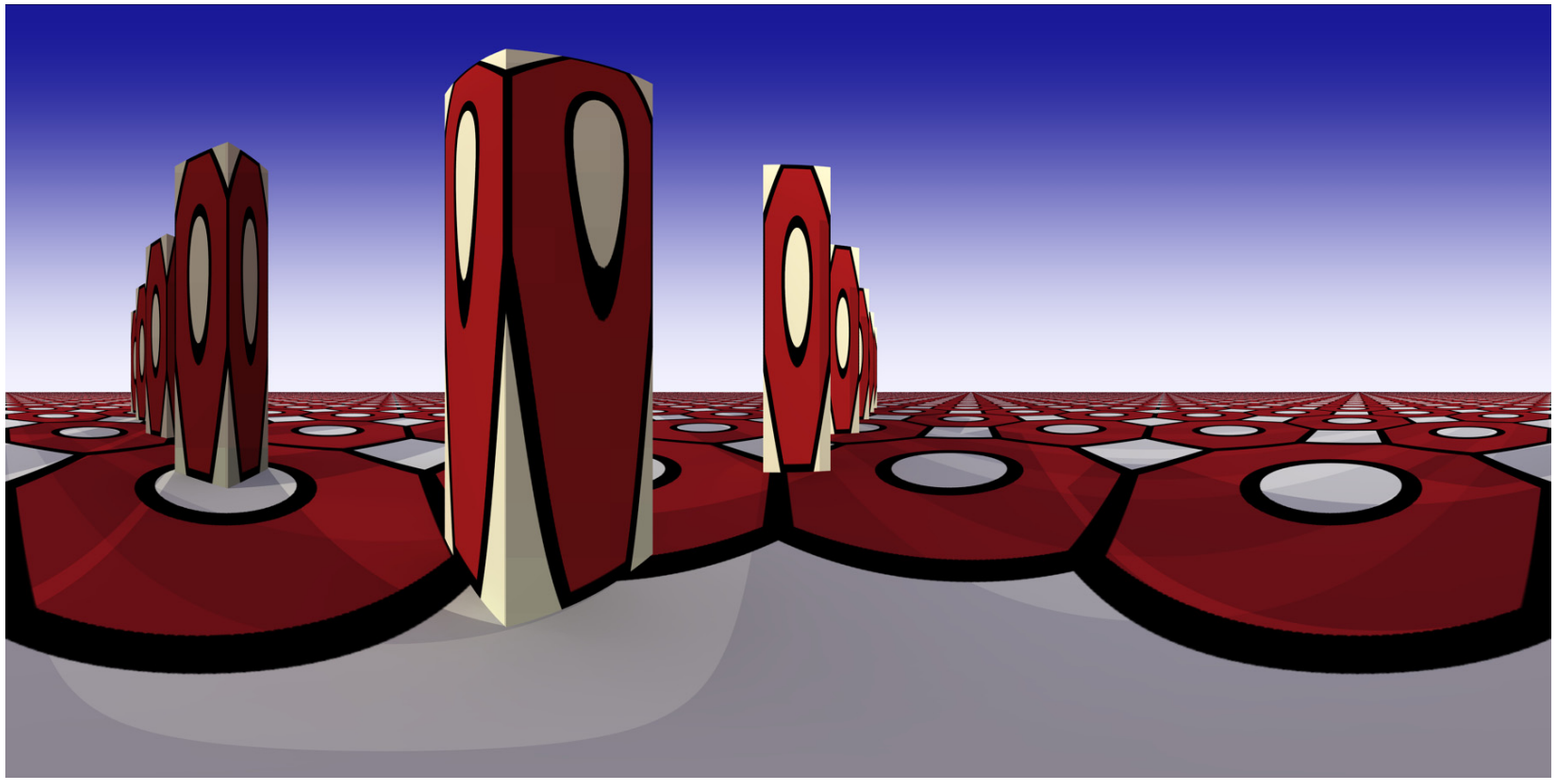
Mapping the scene around the camera to a rectangle

```
 1  miBoolean equirectangular (
 2      miColor *result, miState *state, void *params )
 3  {
 4      miMatrix matrix;
 5      miVector eye_ray_direction = {0, 0, -1};
 6      miScalar x_fractional_position =
 7          state->raster_x / state->camera->x_resolution;
 8      miScalar y_fractional_position =
 9          state->raster_y / state->camera->y_resolution;
10
11      mi_matrix_rotate(
12          matrix,
13          miaux_fit(y_fractional_position, 0, 1, -M_PI/2, M_PI/2),
14          miaux_fit(x_fractional_position, 0, 1, M_PI, -M_PI),
15          0);
16      mi_vector_transform(&eye_ray_direction, &eye_ray_direction, matrix);
17
18      return mi_trace_eye(result, state, &state->org, &eye_ray_direction);
19  }
```

Source code of shader "equirectangular"

Changing the lens

Mapping the scene around the camera to a rectangle

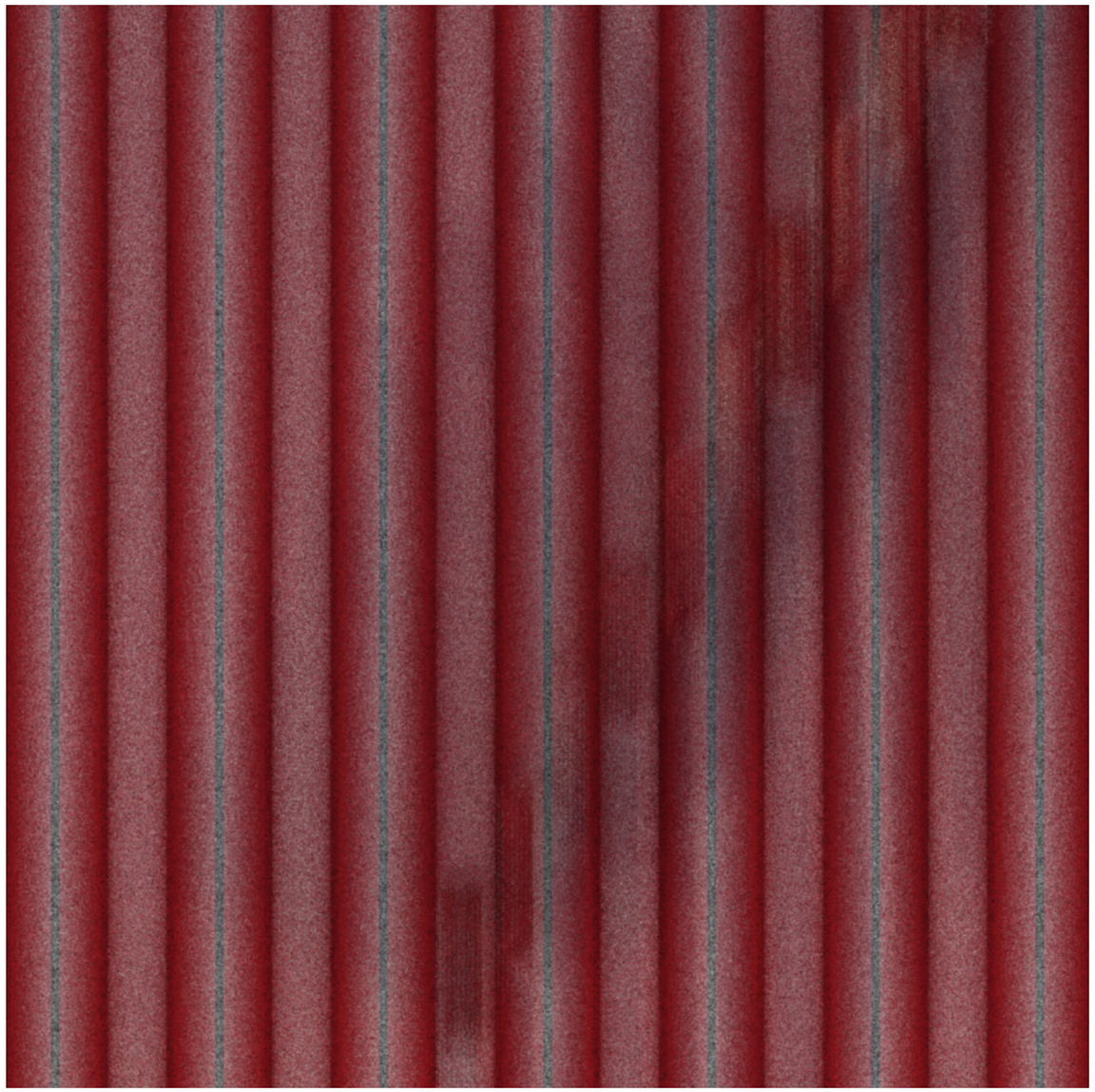


```
camera "camera"
  output "rgba" "tif"
    "lens_4A.tif"
  focal 1.5
  aperture 1
  aspect 2
  resolution 600 300
  environment = "sky"
  lens
    "equirectangular" ()
end camera
```

Changing the position of the camera in the equirectangular projection

Changing the lens

A fisheye lens

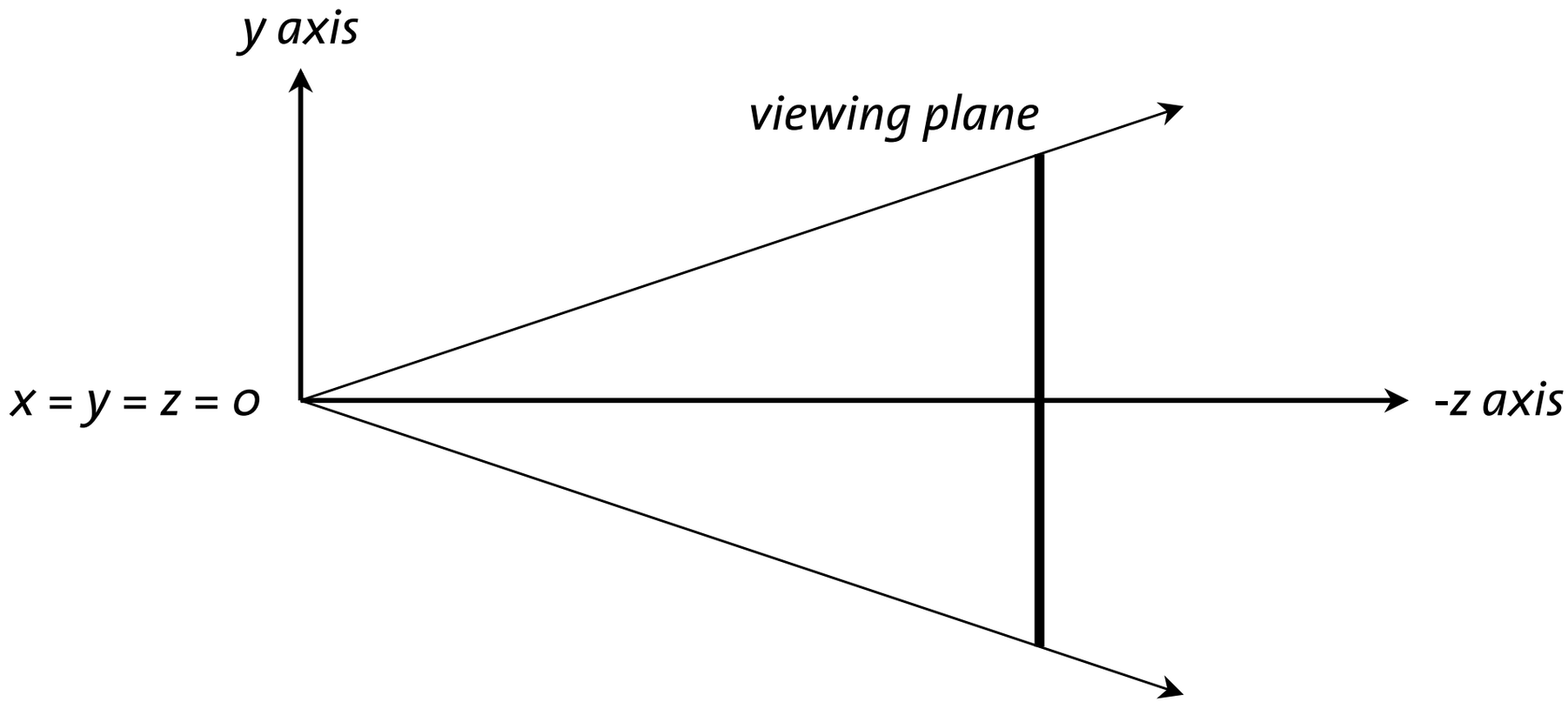


```
camera "camera"
    output "rgba" "tif" "lens_5.tif"
    focal 1.5
    aperture 1
    aspect 1
    resolution 300 300
    environment = "sky"
    lens
        "streak" ( "max_distance" .5 )
end camera

instance "camera-instance" "camera"
    transform
        1  0   0  0
        0  0   1  0
        0 -1   0  0
        0  0 -10  1
end instance
```

Streaking the camera in z with the camera pointed straight down

Changing the lens

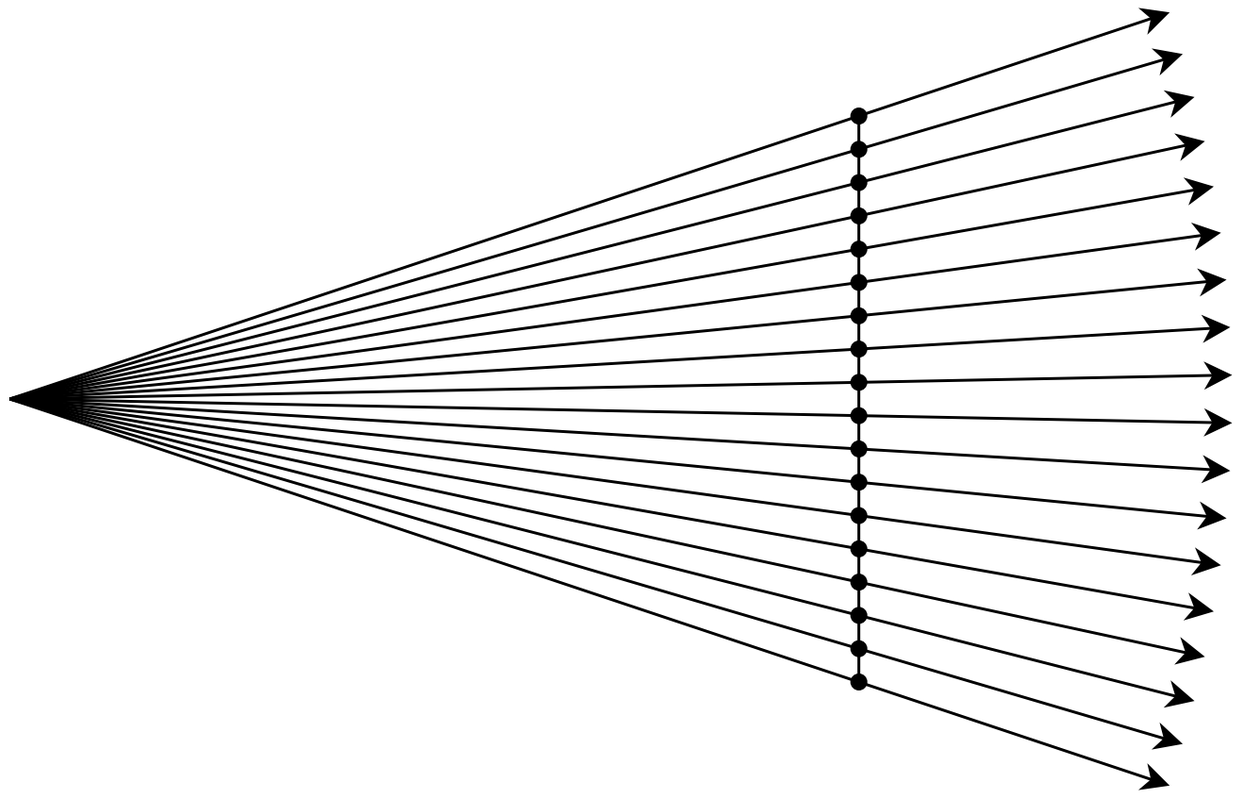
A fisheye lens



The result of transforming the camera to camera space, looking down the positive x axis

Changing the lens

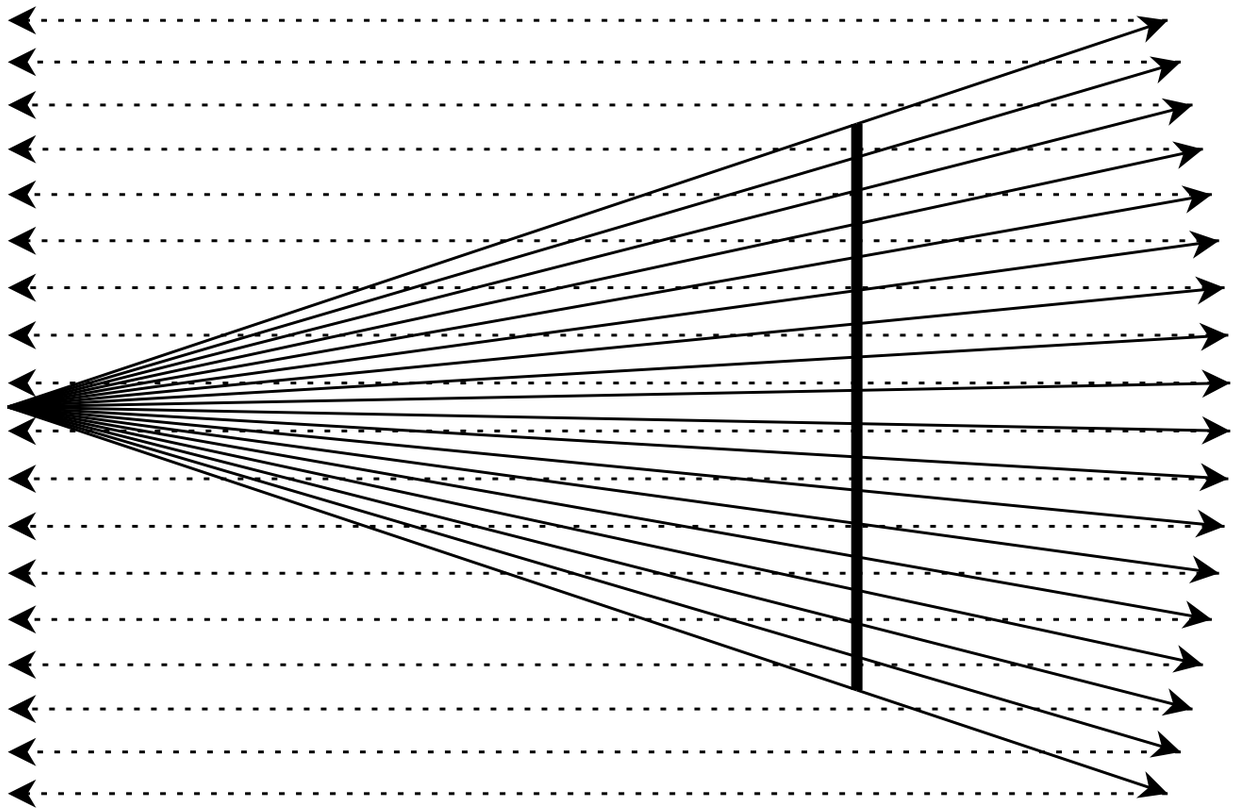
A fisheye lens



Ray intersections in the viewing plane

Changing the lens

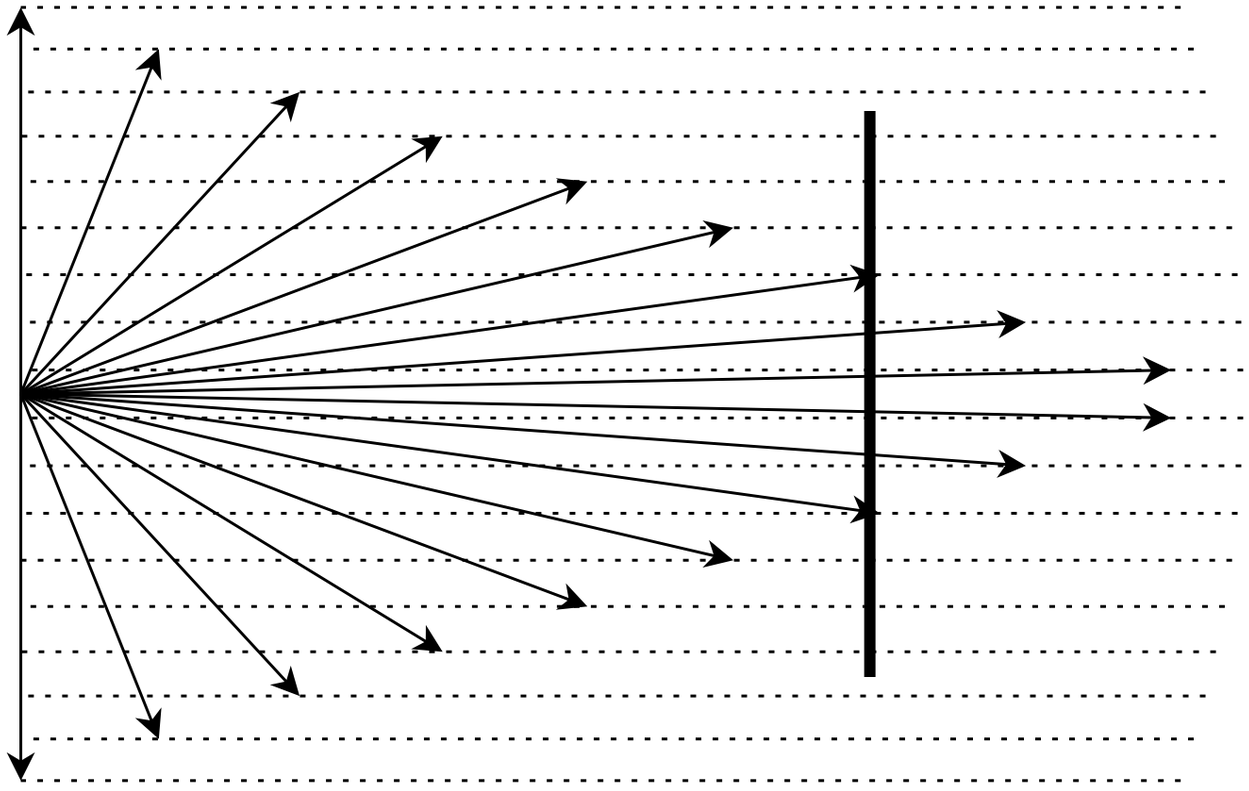
A fisheye lens



Projecting the eye ray back toward the normal plane of the camera direction at its origin

Changing the lens

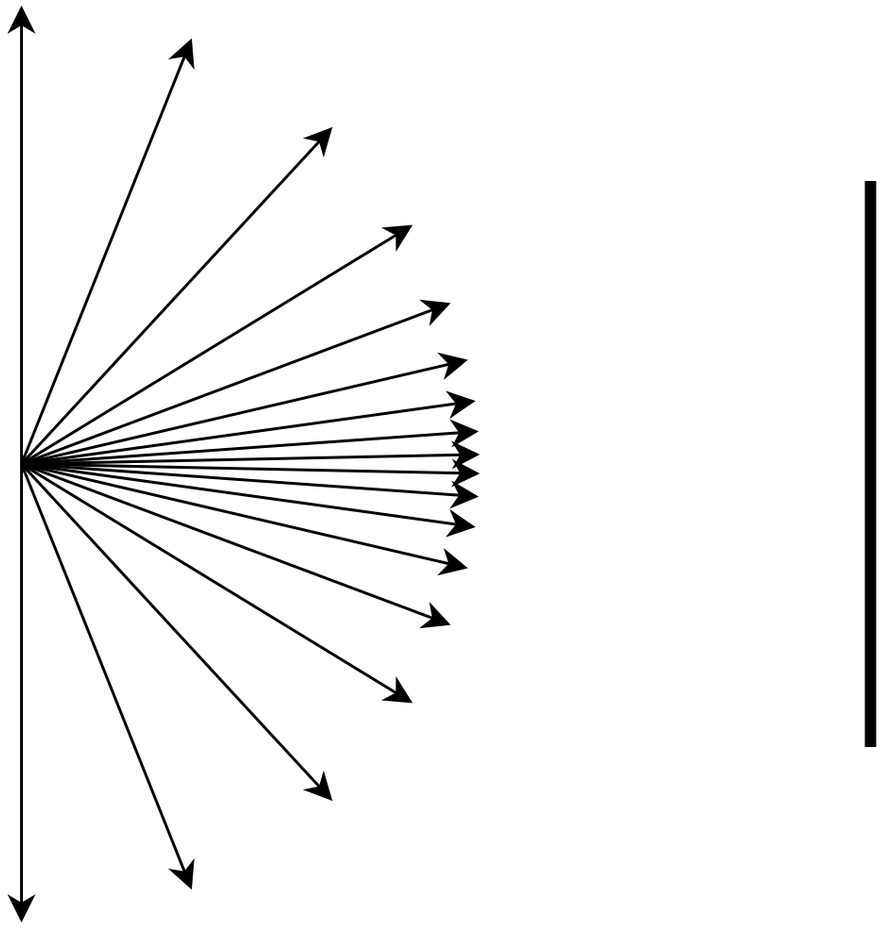
A fisheye lens



Scaling the forward component of the eye ray based on distance from the image center

Changing the lens

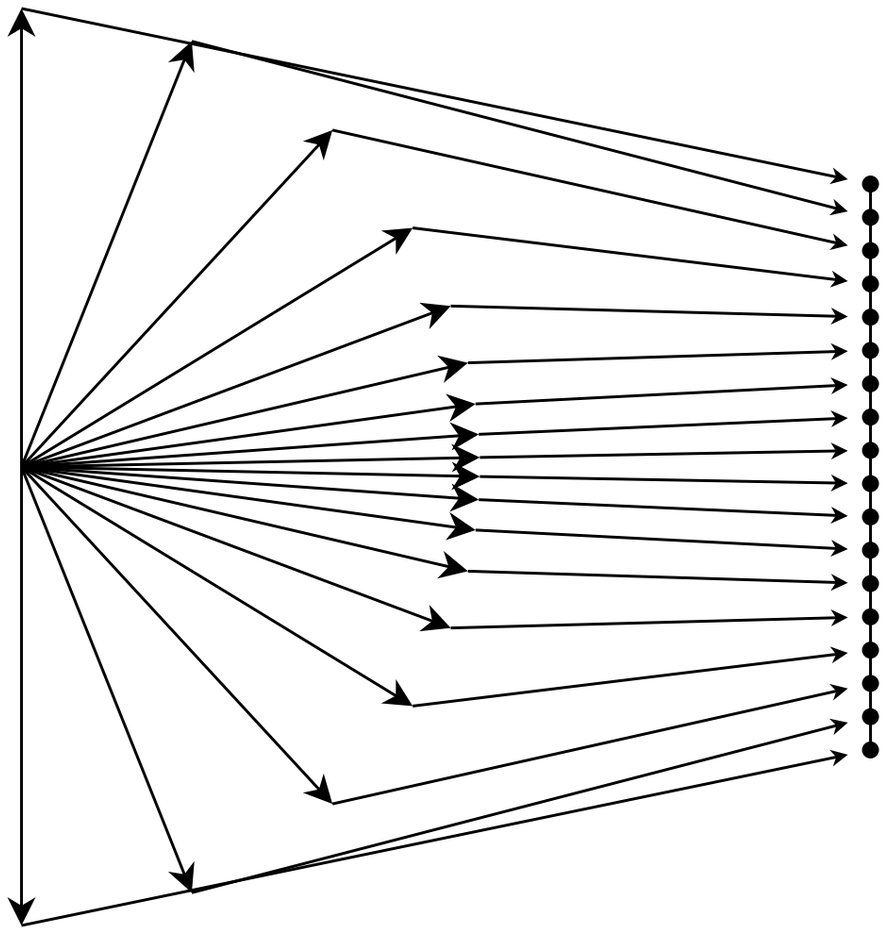
A fisheye lens



Normalizing the bent eye rays

Changing the lens

A fisheye lens



The association of the bent eye rays with the original intersection point in the viewing plane

Changing the lens

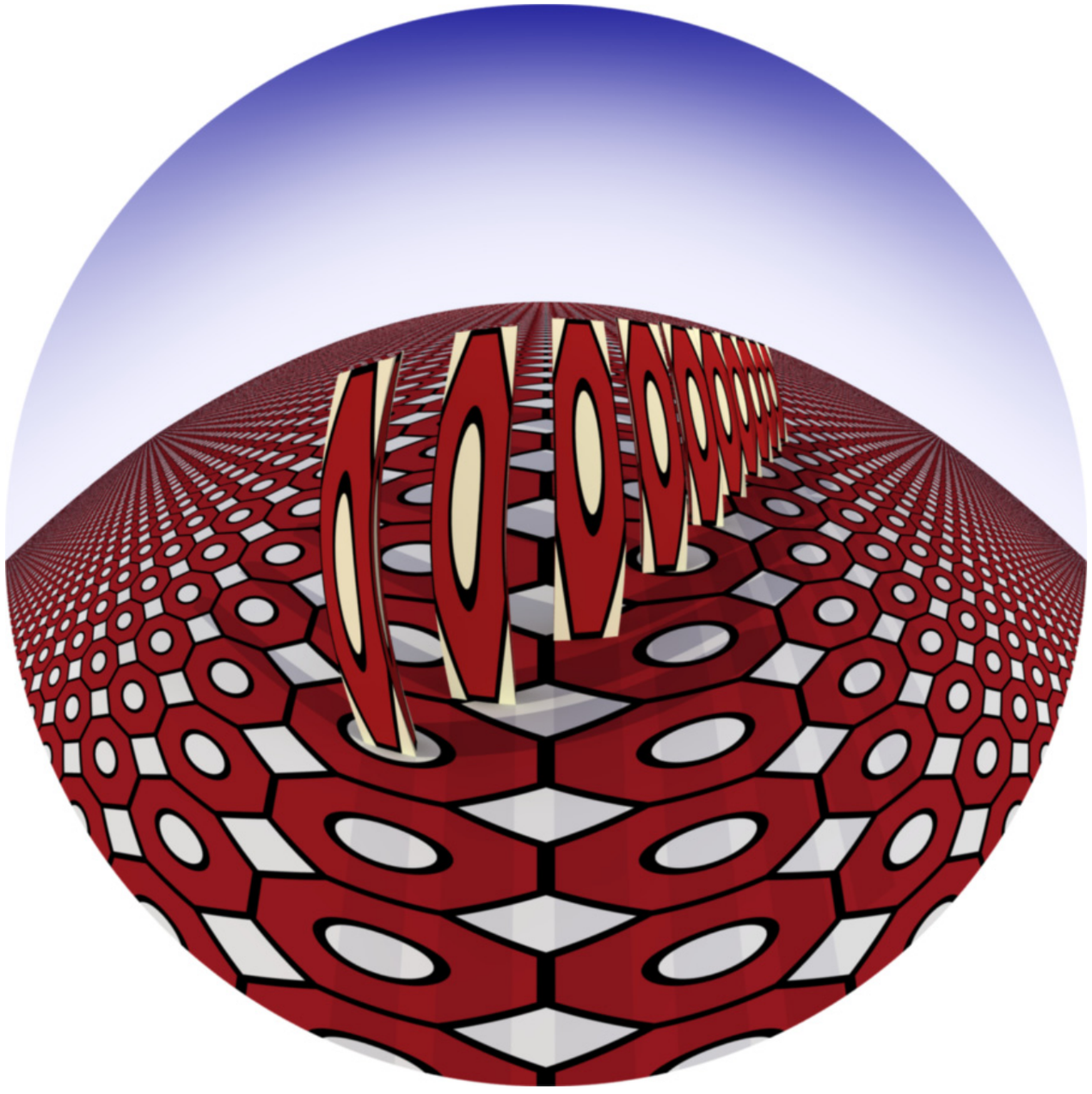
A fisheye lens

```
declare shader
    color "fisheye" (
        color "outside_color" default 1 1 1 )
    apply lens
    scanline off
    trace on
end declare
```

Scene file declaration of shader "fisheye"

Changing the lens

A fisheye lens



```
camera "camera"
    output "rgba" "tif"
        "lens_6.tif"
    focal 1.5
    aperture 1
    aspect 1
    resolution 300 300
    environment = "sky"
    lens
        "fisheye" ()
end camera
```

Shader fisheye added as a lens shader in the camera

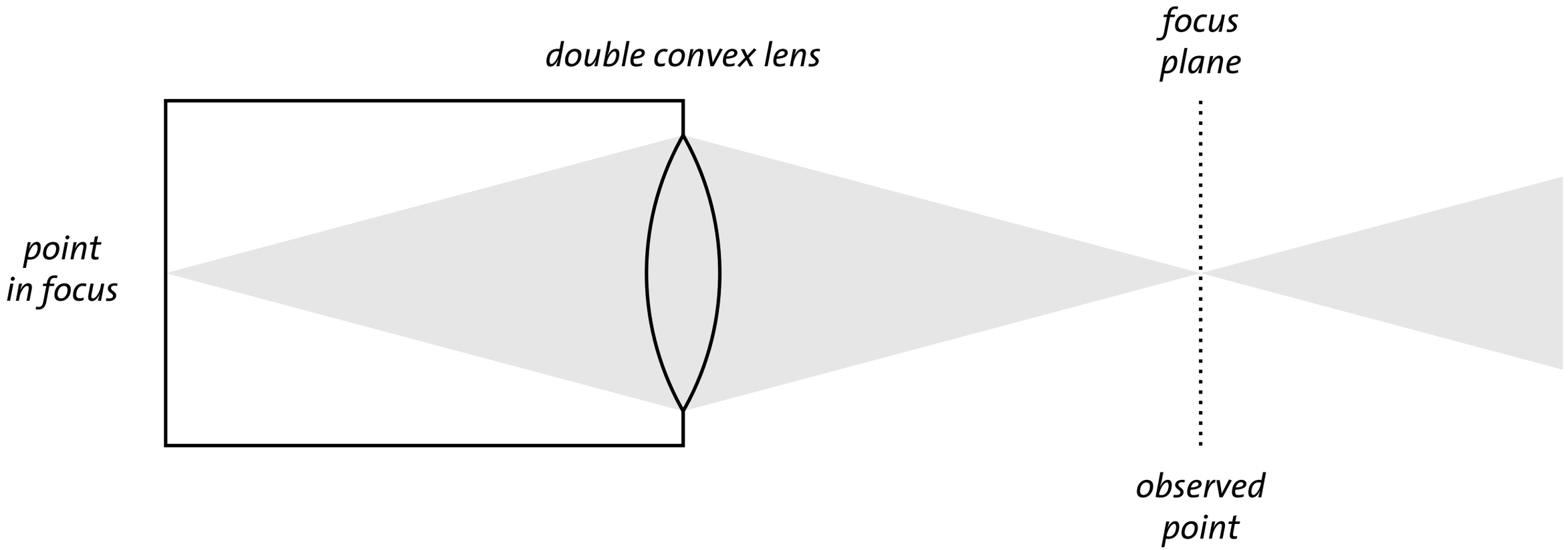
Changing the lens

A fisheye lens

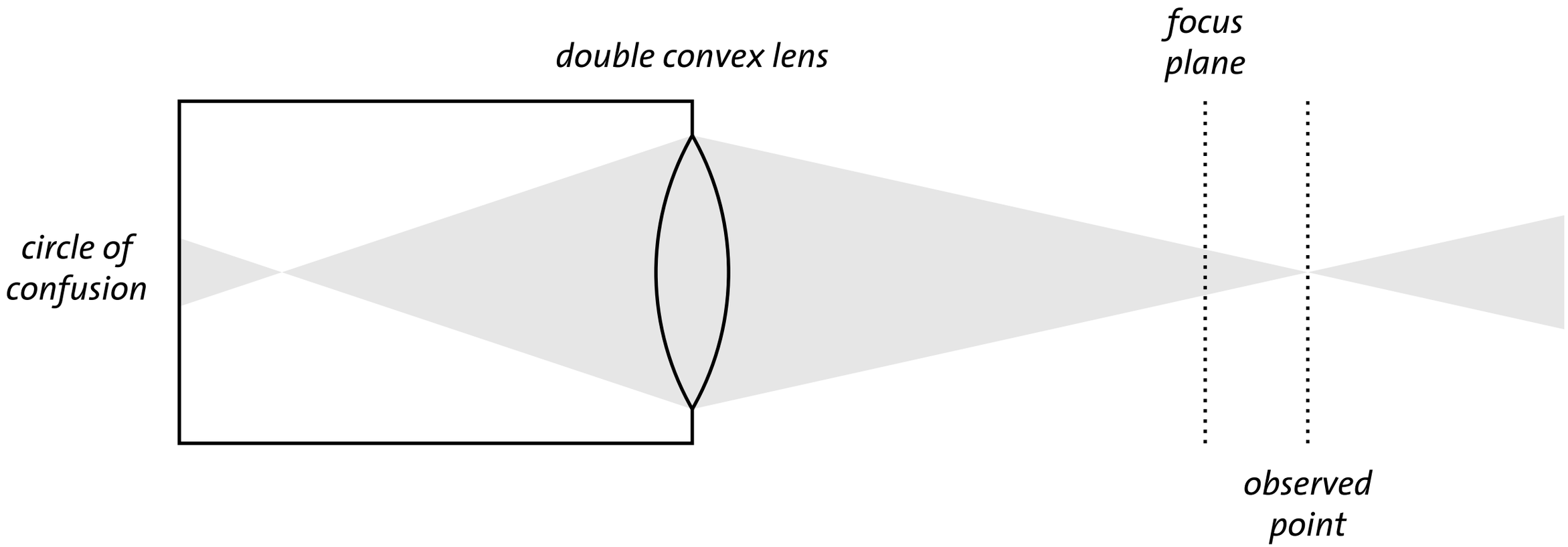
```
 1  struct fisheye {
 2      miColor outside_color;
 3  };
 4
 5  miBoolean fisheye (
 6      miColor *result, miState *state, struct fisheye *params )
 7  {
 8      miVector camera_direction;
 9      miScalar center_x = state->camera->x_resolution / 2.0;
10      miScalar center_y = state->camera->y_resolution / 2.0;
11      miScalar radius = center_x < center_y ? center_x : center_y;
12      miScalar distance_from_center =
13          miaux_distance(center_x, center_y, state->raster_x, state->raster_y);
14
15      if (distance_from_center < radius) {
16          mi_vector_to_camera(state, &camera_direction, &state->dir);
17          camera_direction.z *= miaux_fit(distance_from_center, 0, radius, 1, 0);
18          mi_vector_normalize(&camera_direction);
19          mi_vector_from_camera(state, &camera_direction, &camera_direction);
20          return mi_trace_eye(result, state, &state->org, &camera_direction);
21      } else {
22          *result = *mi_eval_color(&params->outside_color);
23          return miTRUE;
24      }
25  }
```

Source code of shader "fisheye"

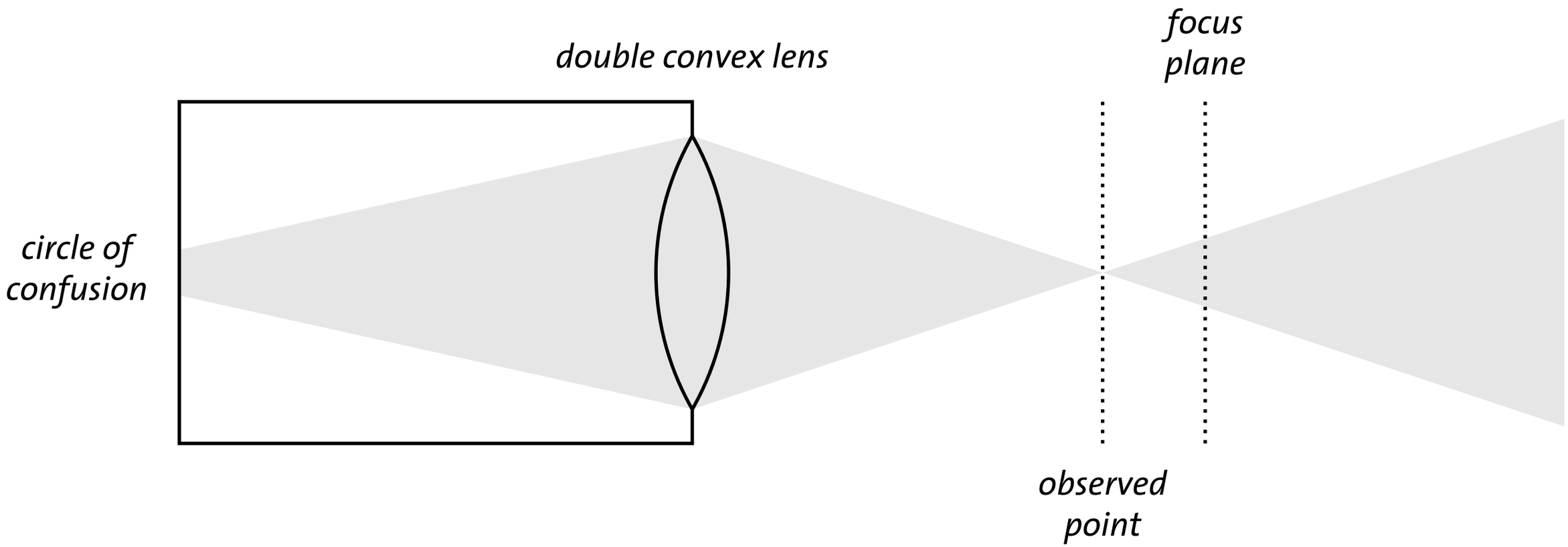
Changing the lens



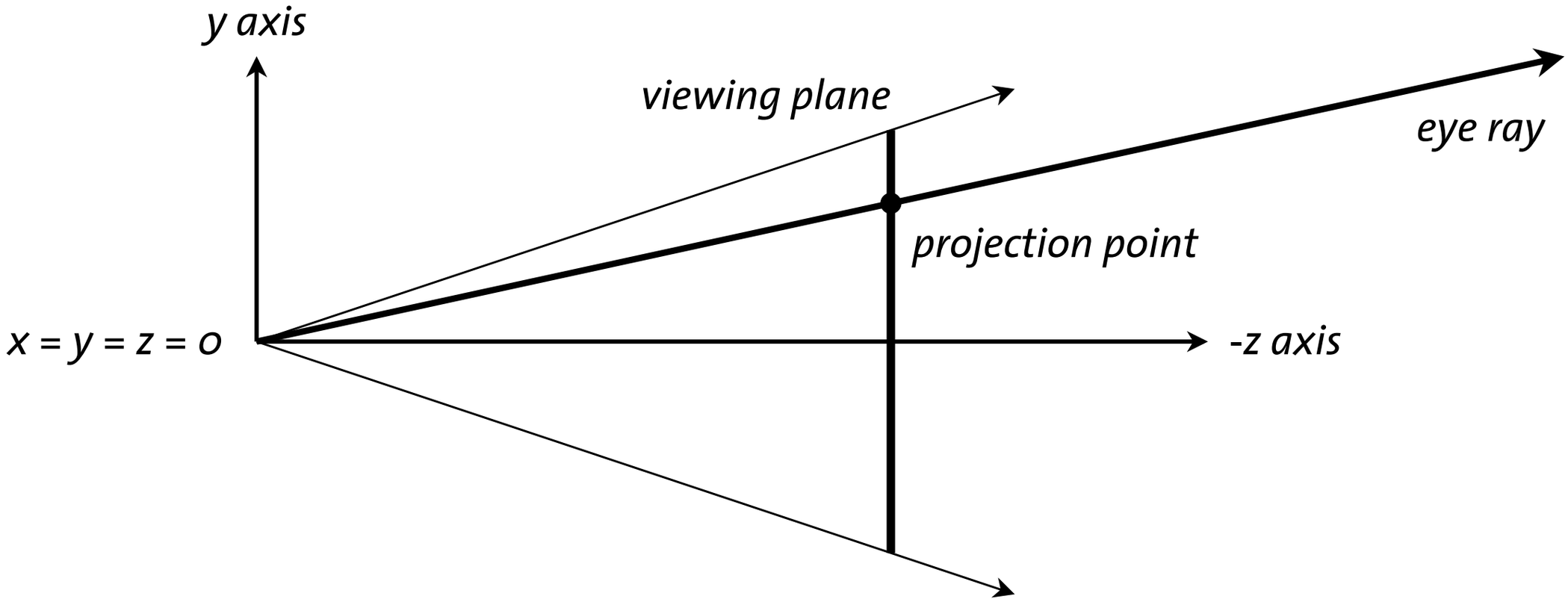
Observed point at the focus plane in focus on the camera plane



Observed point beyond the focus plane

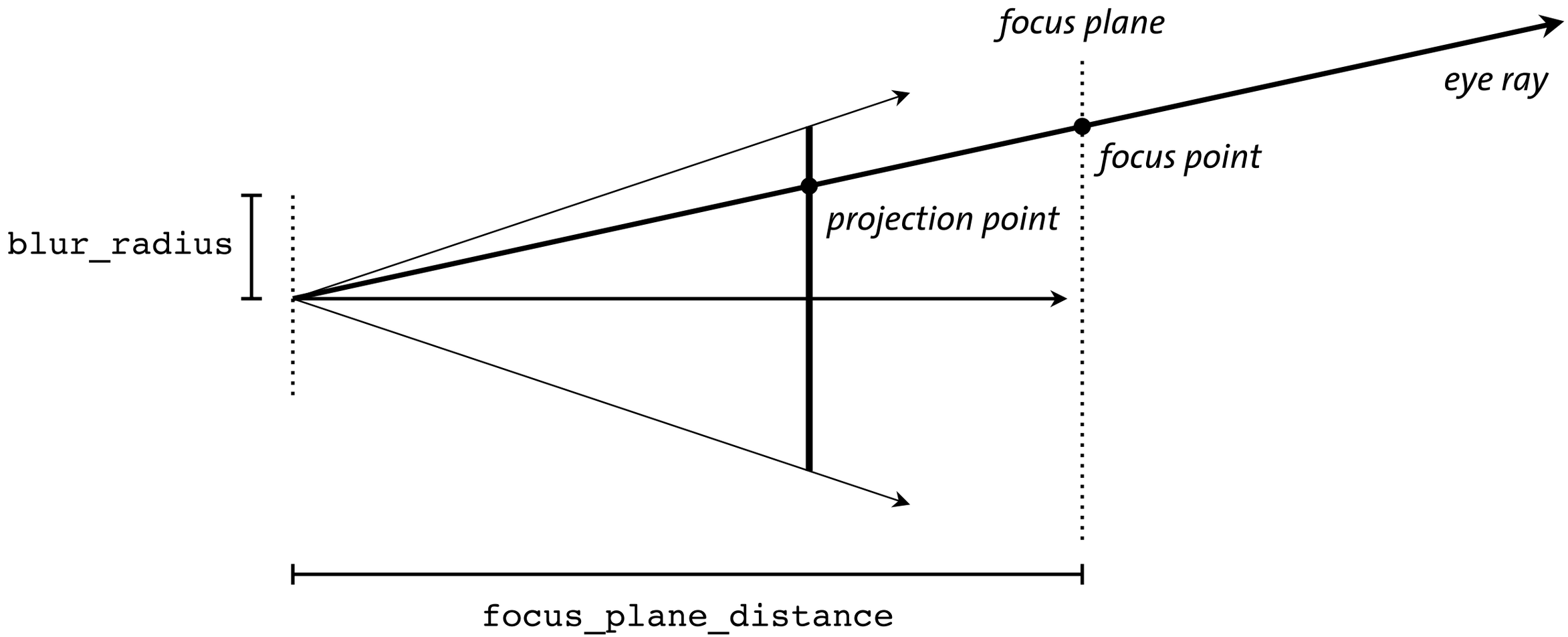


Observed point in front of the focus plane



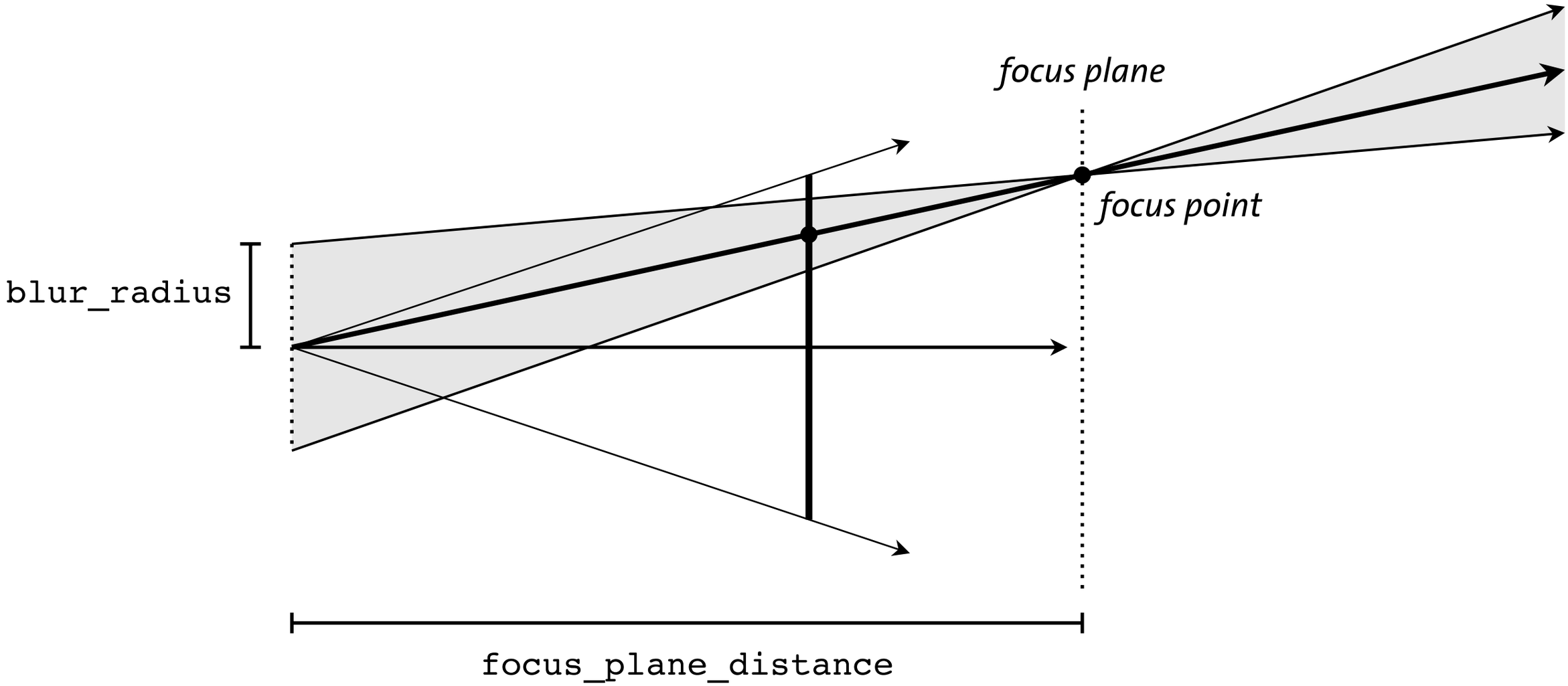
Defining a single eye ray in camera space

Changing the lens



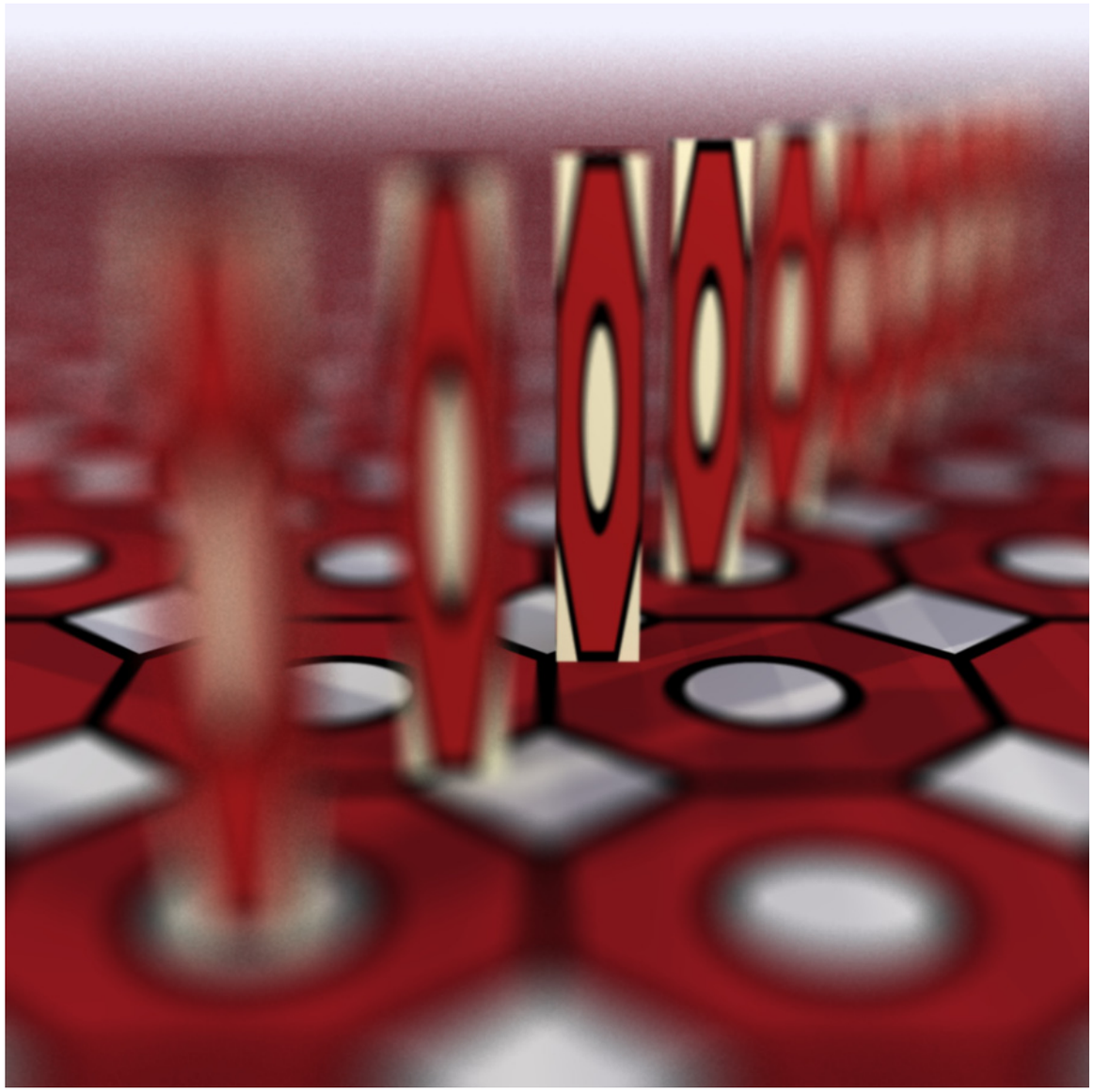
Defining a radius around the camera in its normal plane and a focus point on a focus plane

Changing the lens



The cone (in gray) in which rays are traced and averaged to produce the depth of field image

Changing the lens



```
camera "camera"
    output "rgba" "tif" "lens_7.tif"
    focal 1.5
    aperture 1
    aspect 1
    resolution 300 300
    environment = "sky"
    lens
        "depth_of_field" (
            "focus_plane_distance" 9,
            "number_of_samples" 100,
            "lens_radius" .2 )
end camera
```

Lens shader depth_of_field attached to the camera

```
declare shader
    color "depth_of_field" (
        scalar "focus_plane_distance" default 1,
        scalar "blur_radius" default .1,
        integer "number_of_samples" default 1 )
    apply lens
    scanline off
    trace on
end declare
```

Scene file declaration of shader "depth_of_field"

```
1  void miaux_to_camera_space(
2      miState *state, miVector *origin, miVector *direction)
3  {
4      mi_point_to_camera(state, origin, &state->org);
5      mi_vector_to_camera(state, direction, &state->dir);
6  }
```

Auxiliary function: miaux_to_camera_space

```
1  void miaux_from_camera_space(
2          miState *state, miVector *origin, miVector *direction)
3  {
4      mi_point_from_camera(state, origin, origin);
5      mi_vector_from_camera(state, direction, direction);
6  }
```

Auxiliary function: miaux_from_camera_space

```
1  void miaux_square_to_circle(
2      float *result_x, float *result_y, float x, float y, float max_radius)
3  {
4      float angle = M_PI * 2 * x;
5      float radius = max_radius * sqrt(y);
6      *result_x = radius * cos(angle);
7      *result_y = radius * sin(angle);
8  }
```

Auxiliary function: miaux_square_to_circle

```
1  void miaux_sample_point_within_radius(
2      miVector *result, miVector *center, float x, float y, float max_radius)
3  {
4      float x_offset, y_offset;
5      miaux_square_to_circle(&x_offset, &y_offset, x, y, max_radius);
6      result->x = center->x + x_offset;
7      result->y = center->y + y_offset;
8      result->z = center->z;
9  }
```

Auxiliary function: miaux_sample_point_within_radius

```
1  void miaux_z_plane_intersect(
2      miVector *result, miVector *origin, miVector *direction, miScalar z_plane)
3  {
4      miScalar z_delta = (z_plane - origin->z) / direction->z;
5      result->x = origin->x + z_delta * direction->x;
6      result->y = origin->y + z_delta * direction->y;
7      result->z = z_plane;
8  }
```

Auxiliary function: miaux_z_plane_intersect

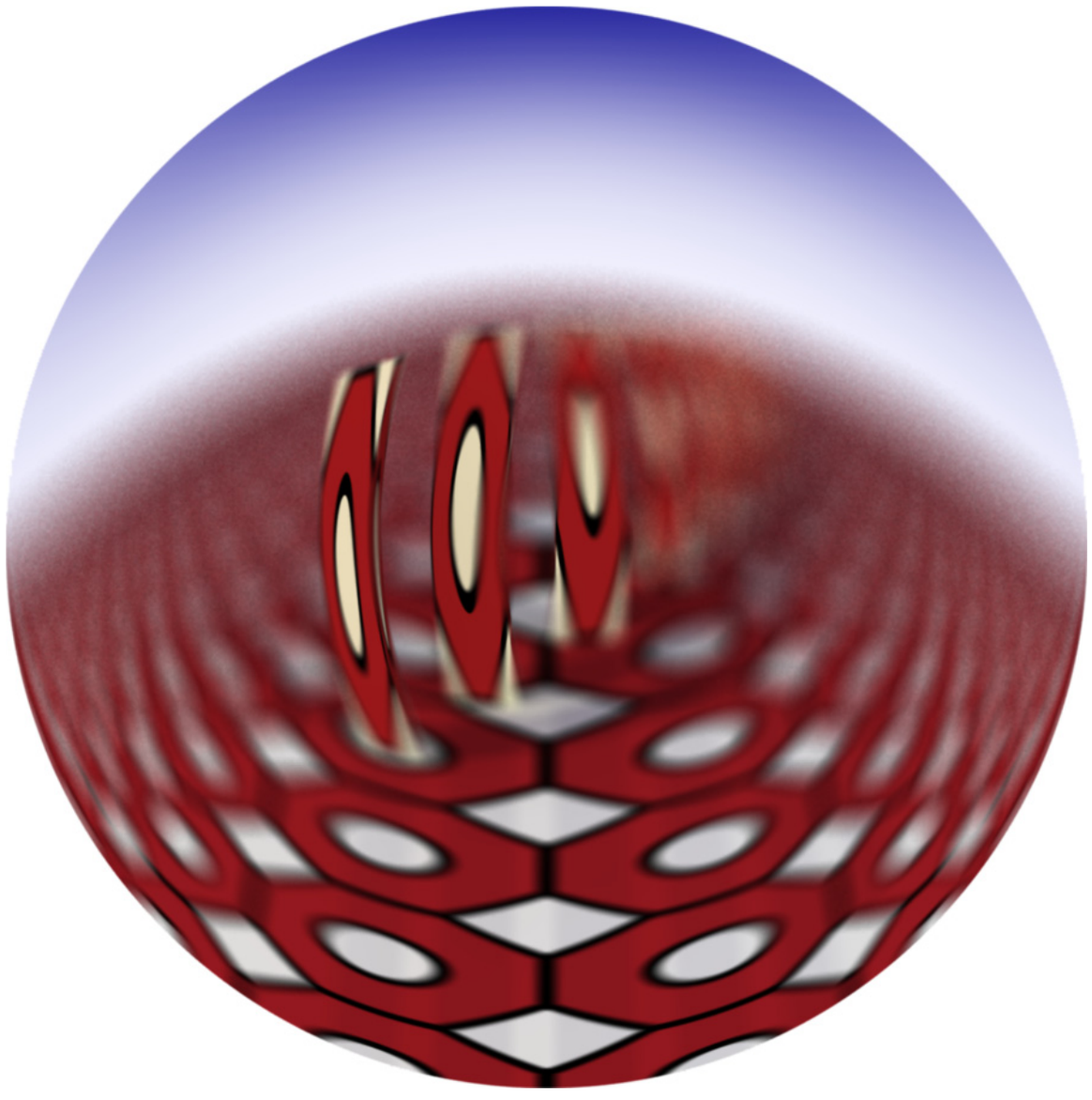
Changing the lens


```
 1  struct depth_of_field {
 2      miScalar focus_plane_distance;
 3      miScalar blur_radius;
 4      miInteger number_of_samples;
 5  };
 6
 7  miBoolean depth_of_field (
 8      miColor *result, miState *state, struct depth_of_field *params )
 9  {
10      miScalar focus_plane_distance =
11          *mi_eval_scalar(&params->focus_plane_distance);
12      miScalar blur_radius  =
13          *mi_eval_scalar(&params->blur_radius);
14      miUint number_of_samples =
15          *mi_eval_integer(&params->number_of_samples);
16
17      miVector camera_origin, camera_direction, origin, direction, focus_point;
18      double samples[2], focus_plane_z;
19      int sample_number = 0;
20      miColor sum = {0,0,0,0}, single_trace;
21
22      miaux_to_camera_space(state, &camera_origin, &camera_direction);
23
24      focus_plane_z = state->org.z - focus_plane_distance;
25      miaux_z_plane_intersect(
26          &focus_point, &camera_origin, &camera_direction, focus_plane_z);
27
28      while (mi_sample(samples, &sample_number, state, 2, &number_of_samples)) {
29          miaux_sample_point_within_radius(
30              &origin, &camera_origin, samples[0], samples[1], blur_radius);
31          mi_vector_sub(&direction, &focus_point, &origin);
32          mi_vector_normalize(&direction);
33          miaux_from_camera_space(state, &origin, &direction);
34          mi_trace_eye(&single_trace, state, &origin, &direction);
35          miaux_add_color(&sum, &single_trace);
36      }
37      miaux_divide_color(result, &sum, number_of_samples);
38      return miTRUE;
39  }
```

Source code of shader "depth_of_field"

Changing the lens

Multiple lens shaders

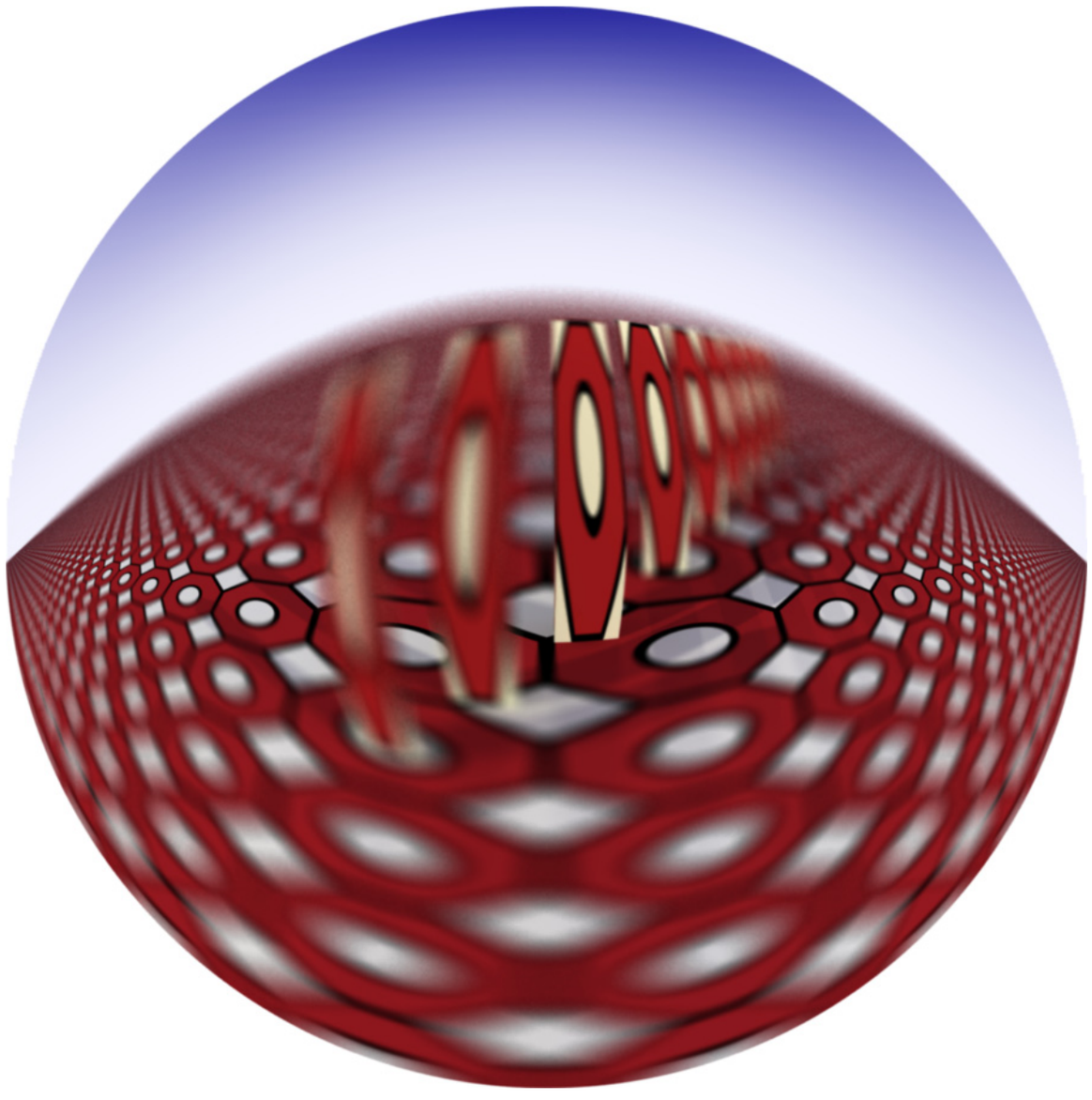


```
camera "camera"
    output "rgba" "tif" "lens_8.tif"
    focal 1.5
    aperture 1
    aspect 1
    resolution 300 300
    environment = "sky"
    lens
        "depth_of_field" (
            "focus_plane_distance" 9,
            "number_of_samples" 100,
            "lens_radius" .2 )
        "fisheye" ()
end camera
```

Calling two lens shaders in a shader list in the camera block

Changing the lens

Multiple lens shaders



```
camera "camera"
    output "rgba" "tif" "lens_9.tif"
    focal 1.5
    aperture 1
    aspect 1
    resolution 300 300
    environment = "sky"
    lens
        "fisheye" ()
        "depth_of_field" (
            "focus_plane_distance" 9,
            "number_of_samples" 100,
            "lens_radius" .2 )
end camera
```

A different rendering produced by reordering the lens shaders

Exercise 22: Use lens shaders

1. Copy `lens_1.mi` to lens.mi, changing the output filename.
2. Render `lens.mi`. (This will create a finalgather map file.)
3. Attach the `fisheye` lens shader to the camera in `lens.mi`.
4. Attach the `depth_of_field` lens shader to the camera in `lens.mi`.
5. Put the lens shaders in a list.

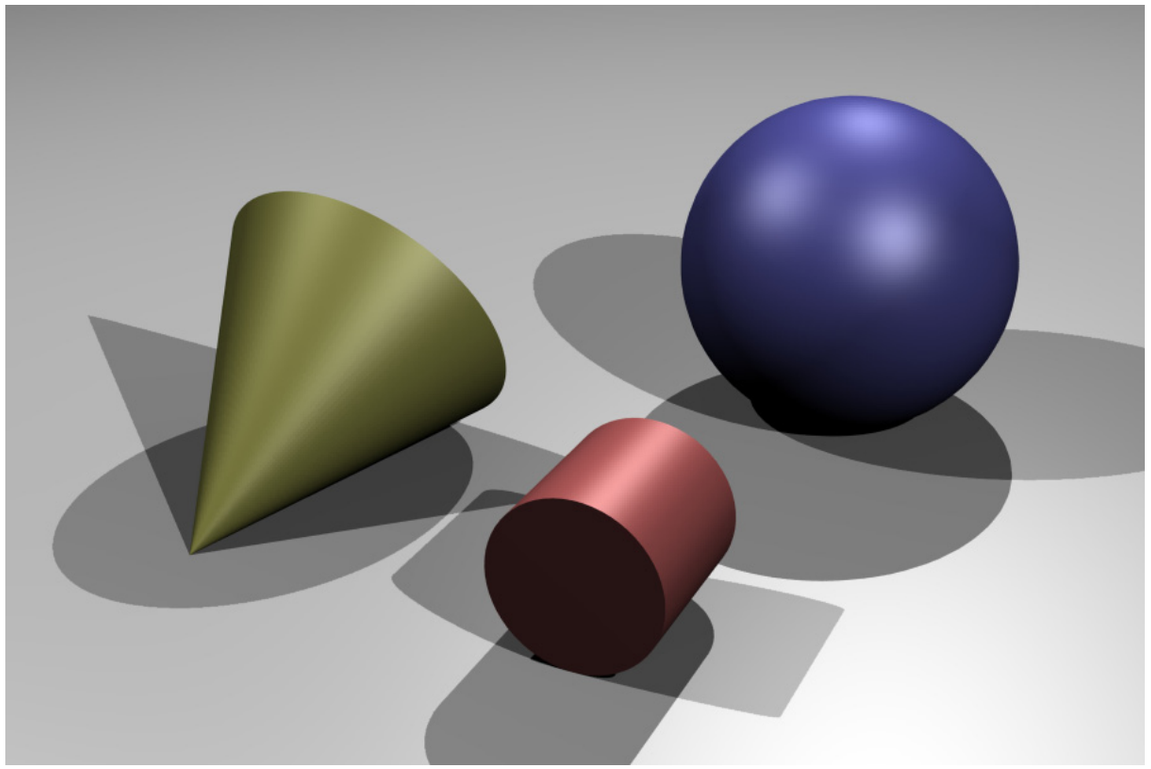
Rendering image components

Rendering image components

- Definition and use of frame buffers
- Separating illumination components into frame buffers
- Compositing illumination components
- Rendering shadows separately
- Saving components from standard shaders
- Using a material Phenomenon with framebuffer components

Rendering image components

Definition and use of frame buffers



```
camera "cam"
    output "rgba" "tif" "buffer_1.tif"
    focal 1.5
    aperture 1
    aspect 1.5
    resolution 300 200
end camera
```

A typical camera block in a scene file

Rendering image components

Definition and use of frame buffers

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

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

Rendering image components

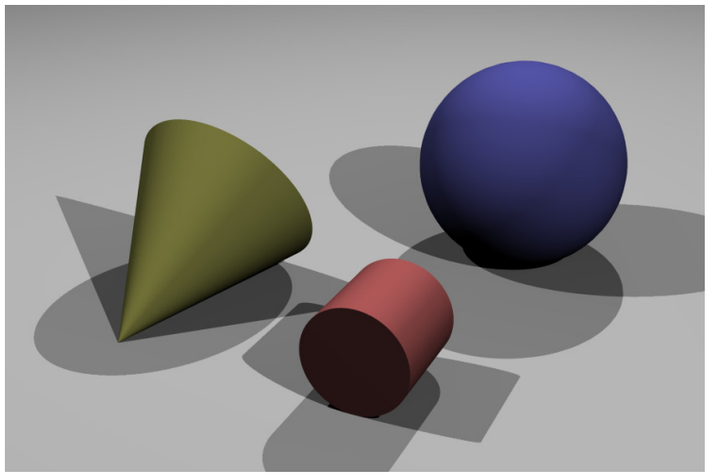
Separating illumination components into frame buffers

```
declare shader
    color "phong_framebuffer" (
        color  "ambient"   default 0 0 0,
        color  "diffuse"   default 1 1 1,
        color  "specular"  default 0 0 0,
        scalar "exponent"  default 30,
        array light "lights" )
end declare
```

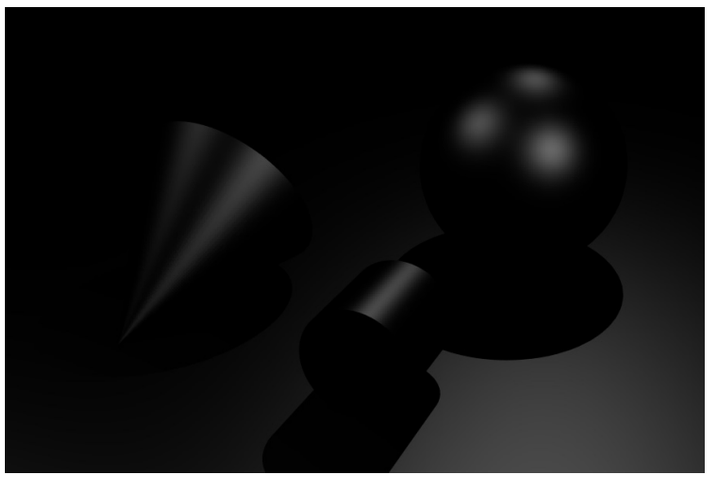
Scene file declaration of shader "phong_framebuffer"

Rendering image components

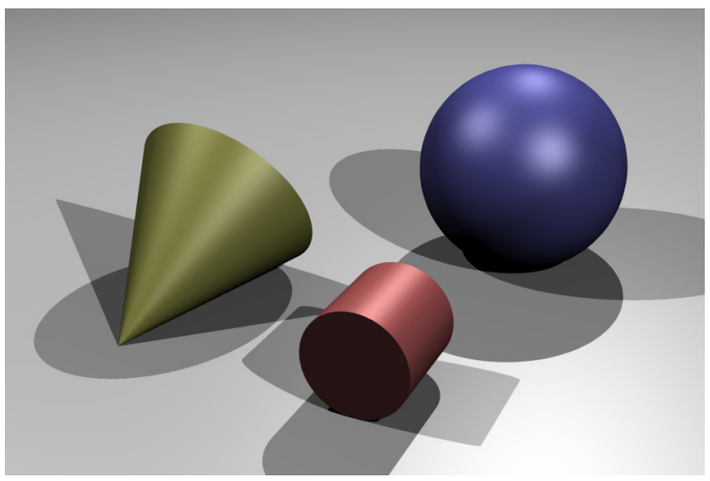
Separating illumination components into frame buffers



buffer_2_diffuse.tif



buffer_2_specular.tif



buffer_2.tif

```
options "opt"
    object space
    contrast .1 .1 .1 1
    samples -1 2
    frame buffer 0 "+rgba"
    frame buffer 1 "+rgba"
end options

camera "cam"
    output "fb0" "tif"
        "buffer_2_diffuse.tif"
    output "fb1" "tif"
        "buffer_2_specular.tif"
    output "rgba" "tif"
        "buffer_2.tif"
    focal 1.5
    aperture 1
    aspect 1.5
    resolution 300 200
end camera

material "yellow"
    "phong_framebuffer" (
        "diffuse" 1 1 .5,
        "specular" 1 1 1,
        "lights" ["L1", "L2", "L3"] )
end material

material "red"
    "phong_framebuffer" (
        "diffuse" 1 .5 .5,
        "specular" 1 1 1,
        "lights" ["L1", "L2", "L3"] )
end material

material "blue"
    "phong_framebuffer" (
        "diffuse" .5 .5 1,
        "specular" 1 1 1,
        "lights" ["L1", "L2", "L3"] )
end material
```

Scene with standard RGBA output and two frame buffers

Rendering image components

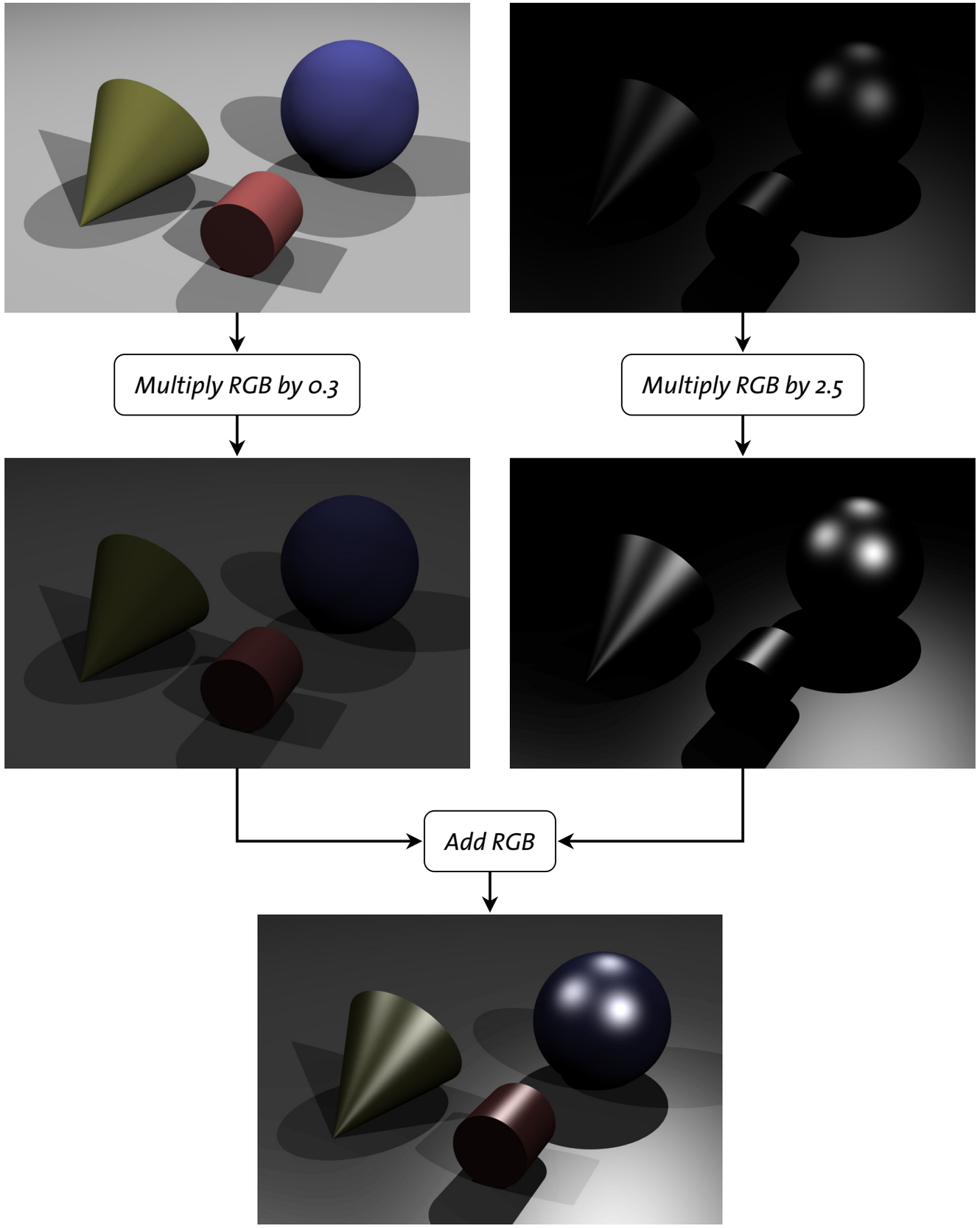
Separating illumination components into frame buffers

```
 1  miBoolean phong_framebuffer(
 2      miColor *result, miState *state, struct phong_framebuffer *params)
 3  {
 4      miColor light_color, diffuse_from_light, specular_from_light,
 5          diffuse_component, specular_component;
 6      int i, light_count, light_sample_count;
 7      miVector direction_toward_light;
 8      miScalar dot_nl;
 9      miTag *light;
10
11      miColor *diffuse = mi_eval_color(&params->diffuse);
12      miColor *specular = mi_eval_color(&params->specular);
13      miScalar exponent = *mi_eval_scalar(&params->exponent);
14      miaux_light_array(&light, &light_count, state,
15                        &params->i_light, &params->n_light, params->light);
16
17      *result = *mi_eval_color(&params->ambient);
18      miaux_set_channels(&diffuse_component, 0.0);
19      miaux_set_channels(&specular_component, 0.0);
20
21      for (i = 0; i < light_count; i++, light++) {
22          miaux_set_channels(&diffuse_from_light, 0.0);
23          miaux_set_channels(&specular_from_light, 0.0);
24          light_sample_count = 0;
25
26          while (mi_sample_light(&light_color, &direction_toward_light,
27                                 &dot_nl, state, *light, &light_sample_count)) {
28              miaux_add_diffuse_component(
29                  &diffuse_from_light, dot_nl, diffuse, &light_color);
30              miaux_add_phong_specular_component(
31                  &specular_from_light, state, exponent,
32                  &direction_toward_light, specular, &light_color);
33          }
34
35          if (light_sample_count > 0) {
36              miScalar scale_factor = 1.0 / light_sample_count;
37              miaux_add_scaled_color(
38                  &diffuse_component, &diffuse_from_light, scale_factor);
39              miaux_add_scaled_color(
40                  &specular_component, &specular_from_light, scale_factor);
41          }
42      }
43      mi_fb_put(state, 0, &diffuse_component);
44      mi_fb_put(state, 1, &specular_component);
45
46      miaux_add_color(result, &diffuse_component);
47      miaux_add_color(result, &specular_component);
48
49      return miTRUE;
50  }
```

Source code of main shader of "phong_framebuffer"

Rendering image components

Compositing illumination components



Modifying the diffuse and specular components in a compositing operation

Rendering image components

Rendering shadows separately

```
 1  void miaux_add_light_color(
 2      miColor *result, miState *state, miTag light, char shadow_state)
 3  {
 4      int light_sample_count = 0;
 5      miScalar dot_nl;
 6      miVector direction_toward_light;
 7      miColor sample_color, single_light_color;
 8
 9      const miOptions *original_options = state->options;
10      miOptions options_copy = *original_options;
11      options_copy.shadow = shadow_state;
12      state->options = &options_copy;
13
14      miaux_set_channels(&single_light_color, 0.0);
15      while (mi_sample_light(&sample_color, &direction_toward_light,
16                             &dot_nl, state, light, &light_sample_count))
17          miaux_add_scaled_color(&single_light_color, &sample_color, dot_nl);
18      if (light_sample_count)
19          miaux_add_scaled_color(result, &single_light_color,
20                                 1.0/light_sample_count);
21
22      state->options = (miOptions*)original_options;
23  }
```

Auxiliary function: miaux_add_light_color

Rendering image components

Rendering shadows separately

```
declare shader
    color "shadowpass" (
        color "base_color",
        array light "lights" )
end declare
```

Scene file declaration of shader "shadowpass"

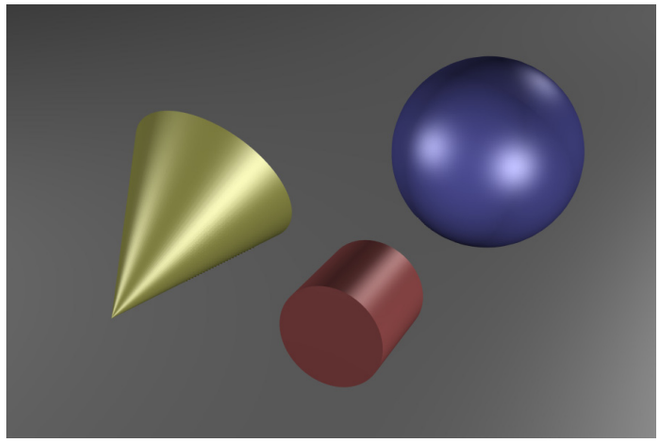
Rendering image components

Rendering shadows separately

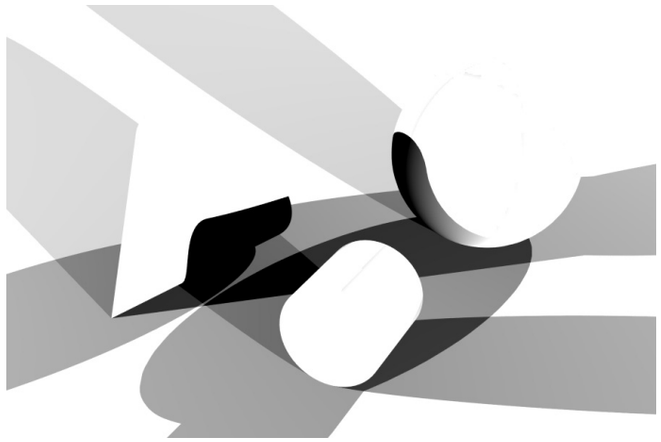
```
 1  struct shadowpass {
 2      miColor base_color;
 3      int     i_light;
 4      int     n_light;
 5      miTag   light[1];
 6  };
 7
 8  miBoolean shadowpass (
 9      miColor *result, miState *state, struct shadowpass *params )
10  {
11      int i, light_count;
12      miTag *light;
13      miColor without_shadow, with_shadow, shadow;
14      miColor *base_color = mi_eval_color(&params->base_color);
15      miaux_light_array(&light, &light_count, state,
16                        &params->i_light, &params->n_light, params->light);
17
18      miaux_set_channels(&without_shadow, 0.0);
19      miaux_set_channels(&with_shadow, 0.0);
20
21      for (i = 0; i < light_count; i++, light++) {
22          miaux_add_light_color(&without_shadow, state, *light, miFALSE);
23          miaux_add_light_color(&with_shadow, state, *light, miTRUE);
24      }
25      miaux_divide_colors(&shadow, &with_shadow, &without_shadow);
26
27      miaux_multiply_colors(result, base_color, &shadow);
28      mi_fb_put(state, 0, base_color);
29      mi_fb_put(state, 1, &shadow);
30
31      return miTRUE;
32  }
```

Source code of shader "shadowpass"

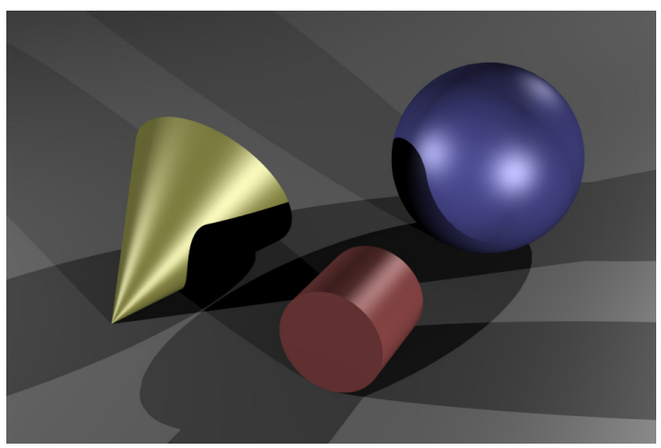
Rendering image components



buffer_3_without_shadows.tif



buffer_3_shadows.tif



buffer_3.tif

Rendering shadows separately

```
options "opt"
    object space
   contrast .05 .05 .05 1
    samples -1 2
    shadow off
    frame buffer 0 "+rgba"
    frame buffer 1 "+rgba"
end options

camera "cam"
    output "fb0" "tif"
        "buffer_3_without_shadows.tif"
    output "fb1" "tif"
        "buffer_3_shadows.tif"
    output "rgba" "tif"
        "buffer_3.tif"
    focal 1.5
    aperture 1
    aspect 1.5
    resolution 300 200
end camera

shader "yellow_phong"
    "phong" (
        "diffuse" 1 1 .5,
        "specular" 1 1 1,
        "lights" ["L1", "L2", "L3"] )

shader "red_phong"
    "phong" (
        "diffuse" 1 .5 .5,
        "specular" 1 1 1,
        "lights" ["L1", "L2", "L3"] )

shader "blue_phong"
    "phong" (
        "diffuse" .5 .5 1,
        "specular" 1 1 1,
        "lights" ["L1", "L2", "L3"] )

material "yellow"
    "shadowpass" (
        "base_color" = "yellow_phong",
        "lights" ["L1", "L2", "L3"] )
end material

material "red"
    "shadowpass" (
        "base_color" = "red_phong",
        "lights" ["L1", "L2", "L3"] )
end material

material "blue"
    "shadowpass" (
        "base_color" = "blue_phong",
        "lights" ["L1", "L2", "L3"] )
end material
```

Separating shadows into a separate frame buffer

```
declare shader
    color "framebuffer_put" (
        color "color",
        integer "index" )
end declare
```

Scene file declaration of shader "framebuffer_put"

Rendering image components

Saving components from standard shaders

```
1  struct framebuffer_put {
2      miColor color;
3      miInteger index;
4  };
5
6  miBoolean framebuffer_put(
7      miColor *result, miState *state, struct framebuffer_put *params)
8  {
9      *result = *mi_eval_color(&params->color);
10
11     if (state->type == miRAY_EYE)
12         mi_fb_put(state, *mi_eval_integer(&params->index), result);
13
14     return miTRUE;
15 }
```

Source code of shader "framebuffer_put"

```
declare shader
    color "add_colors" (
        color "x",
        color "y" )
end declare
```

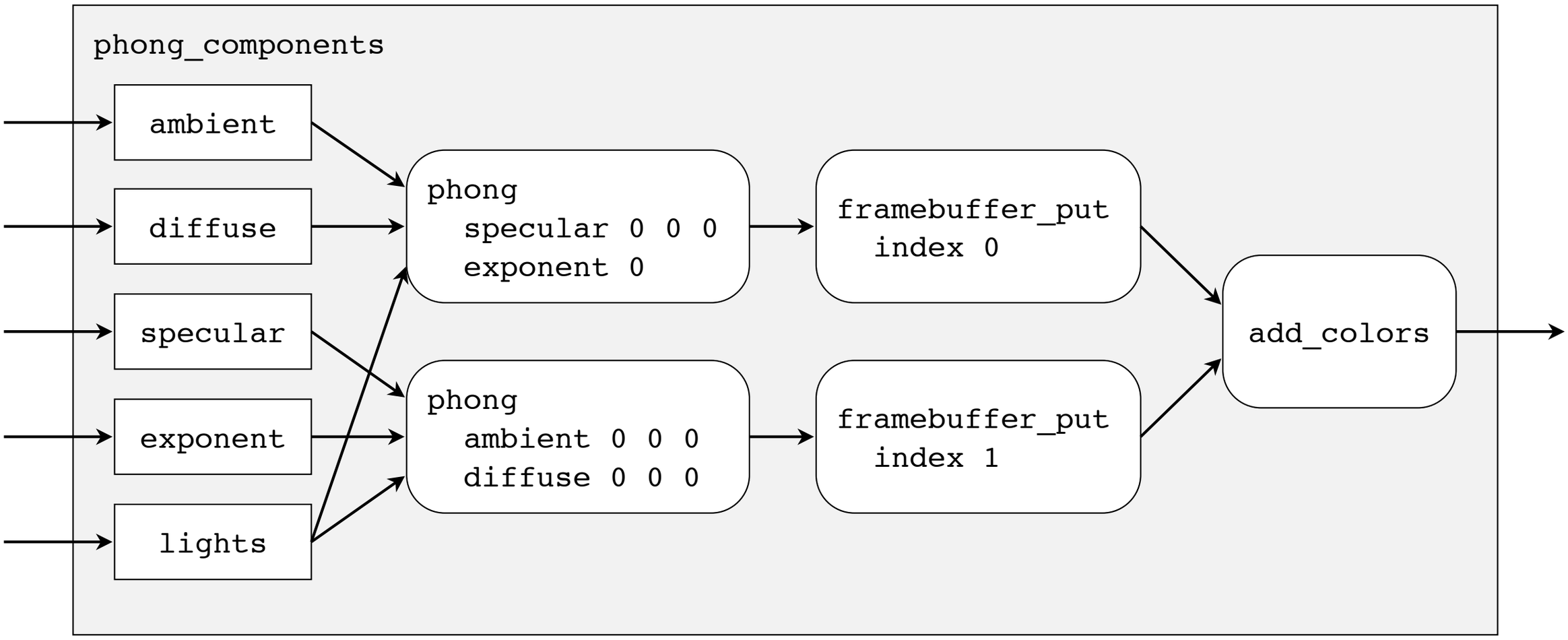
Scene file declaration of shader "add_colors"

```
 1  struct add_colors {
 2      miColor x;
 3      miColor y;
 4  };
 5
 6  miBoolean add_colors(
 7      miColor *result, miState *state, struct add_colors *params)
 8  {
 9      miColor *x = mi_eval_color(&params->x);
10      miColor *y = mi_eval_color(&params->y);
11
12      result->r = x->r + y->r;
13      result->g = x->g + y->g;
14      result->b = x->b + y->b;
15
16      return miTRUE;
17  }
```

Source code of shader "add_colors"

Rendering image components

Saving components from standard shaders



Design of a Phenomenon to calculate components by using shader phong twice

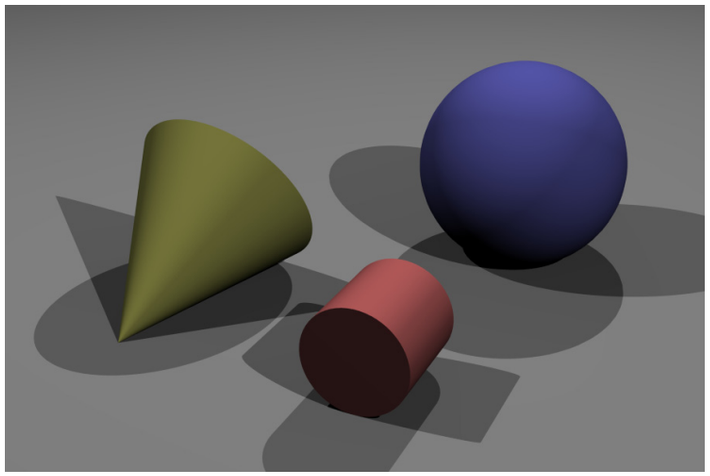
Rendering image components

Saving components from standard shaders

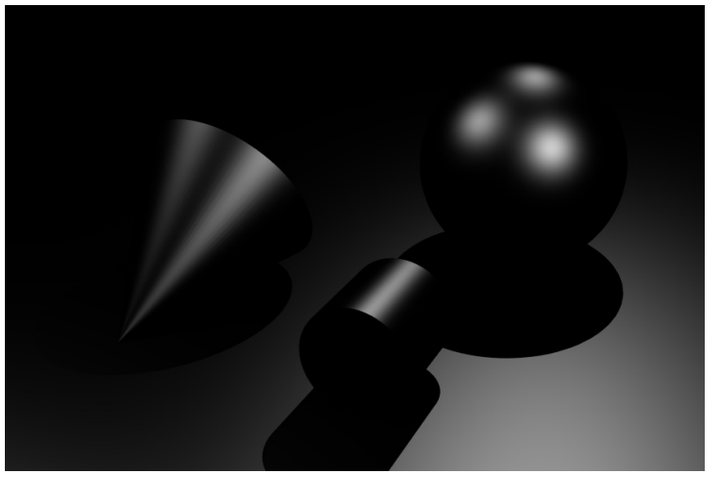
```
declare phenomenon
    color "phong_components" (
        color  "ambient"  default 0 0 0,
        color  "diffuse"  default 1 1 1,
        color  "specular" default 0 0 0,
        scalar "exponent" default 30,
        array light "lights" )

    shader "diffuse_component"
        "phong" (
            "ambient"  = interface "ambient",
            "diffuse"  = interface "diffuse",
            "specular" 0 0 0,
            "exponent" 0,
            "lights"   = interface "lights" )

    shader "specular_component"
        "phong" (
            "ambient"  0 0 0,
            "diffuse"  0 0 0,
            "specular" = interface "specular",
            "exponent" = interface "exponent",
            "lights"   = interface "lights" )

    shader "fb_diffuse"
        "framebuffer_put" (
            "color" = "diffuse_component",
            "index" 0 )

    shader "fb_specular"
        "framebuffer_put" (
            "color" = "specular_component",
            "index" 1 )

    shader "add_components"
        "add_colors" (
            "x" = "fb_diffuse",
            "y" = "fb_specular" )

    root = "add_components"

end declare
```

A reimplementation of shader phong_framebuffer as a Phenomenon

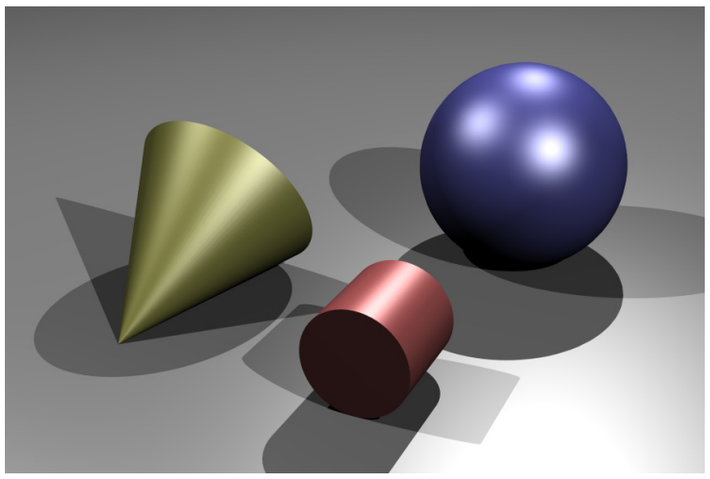
Rendering image components



buffer_4_diffuse.tif



buffer_4_specular.tif



buffer_4.tif

Saving components from standard shaders

```
options "opt"
    object space
    contrast .1 .1 .1 1
    samples 2 2
    frame buffer 0 "+rgba"
    frame buffer 1 "+rgba"
end options

camera "cam"
    output "fb0" "tif"
        "buffer_4_diffuse.tif"
    output "fb1" "tif"
        "buffer_4_specular.tif"
    output "rgba" "tif"
        "buffer_4.tif"
    focal 1.5
    aperture 1
    aspect 1.5
    resolution 300 200
end camera

material "yellow"
    "phong_components" (
        "diffuse" 1 1 .5,
        "specular" 2 2 2,
        "lights" ["L1", "L2", "L3"] )
end material

material "red"
    "phong_components" (
        "diffuse" 1 .5 .5,
        "specular" 2 2 2,
        "lights" ["L1", "L2", "L3"] )
end material

material "blue"
    "phong_components" (
        "diffuse" .5 .5 1,
        "specular" 2 2 2,
        "lights" ["L1", "L2", "L3"] )
end material
```

Using the phong_components Phenomenon to create frame buffer image files

Rendering image components

Using a material Phenomenon with framebuffer components

```
declare phenomenon
    "global_phong_components" (
        color "diffuse" default .5 .5 .5,
        color "specular" default .3 .3 .3,
        scalar "exponent" default 30,
        array light "lights" )

    shader "diffuse"
        "phong" (
            "diffuse" = interface "diffuse",
            "lights" = interface "lights",
            "specular" 0 0 0 )

    shader "specular"
        "phong" (
            "specular" = interface "specular",
            "exponent" = interface "exponent",
            "lights" = interface "lights",
            "diffuse" 0 0 0 )

    shader "indirect"
        "average_radiance" ()

    shader "indirect_diffuse"
        "op_mul_cc" (
            "A" = "indirect",
            "B" = interface "diffuse" )

    shader "write_diffuse"
        "framebuffer_put" (
            "color" = "diffuse",
            "index" 0 )

    shader "write_specular"
        "framebuffer_put" (
            "color" = "specular",
            "index" 1 )

    shader "write_indirect"
        "framebuffer_put" (
            "color" = "indirect_diffuse",
            "index" 2 )

    shader "add_direct"
        "add_colors" (
            "x" = "write_diffuse",
            "y" = "write_specular" )

    shader "add_indirect"
        "add_colors" (
            "x" = "add_direct",
            "y" = "write_indirect" )

    root = "add_indirect"
end declare
```

A Phenomenon with frame buffers containing the diffuse, specular and indirect components

Rendering image components Using a material Phenomenon with framebuffer components

```
declare phenomenon
    material "global_phong" (
        color "diffuse" default 1 1 1,
        array light "lights" )

    material "phong_buffers"
        "global_phong_components" (
            "diffuse" = interface "diffuse",
            "specular"  1 1 1,
            "lights" = interface "lights" )
        photon
            "store_diffuse_photon" (
                "diffuse_color" = interface "diffuse" )
    end material

    root material "phong_buffers"
end declare
```

A material Phenomenon including a photon shader sharing the diffuse color parameter

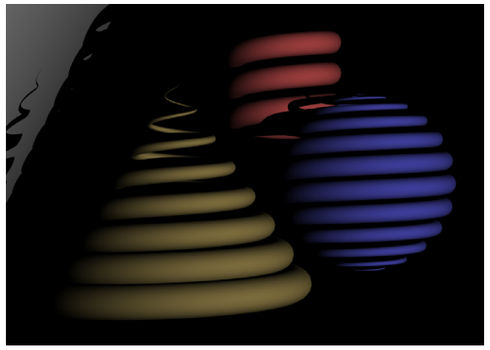
Rendering image components Using a material Phenomenon with framebuffer components

```
light "light"
    "spotlight" (
        "light_color" .6 .6 .6 )
    origin 4 -.5 7
    direction  -4 .5 -7
    spread .9
end light

instance "light_inst" "light" end instance

declare phenomenon
    material "global_phong_with_light" (
        color "diffuse" default 1 1 1 )

    material "phong_buffers"
        "global_phong_components" (
            "diffuse" = interface "diffuse",
            "specular"  1 1 1,
            "lights" [ "::light_inst" ] )
        photon
            "store_diffuse_photon" (
                "diffuse_color" = interface "diffuse" )
    end material

    root material "phong_buffers"
end declare
```

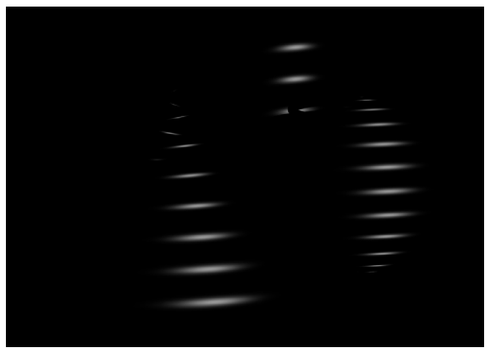
A material Phenomenon that includes a photon shader and a light instance reference

Rendering image components

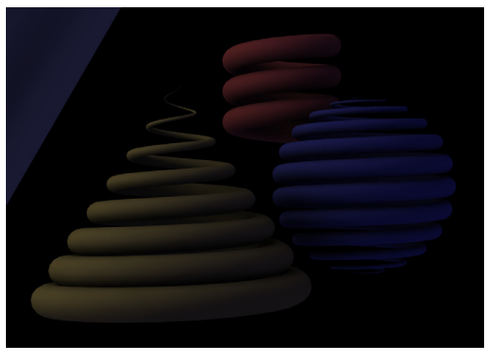
Using a material Phenomenon with framebuffer components



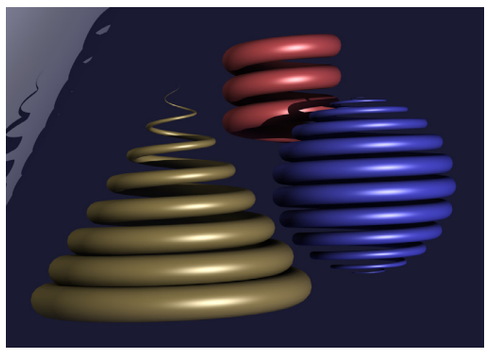
buffer_5_diffuse.tif



buffer_5_specular.tif



buffer_5_indirect.tif



buffer_5.tif

```
options "opt"
    object space
    contrast .1 .1 .1 1
    samples 0 2
    finalgather on
    finalgather accuracy 50 2 .5
    frame buffer 0 "+rgba"
    frame buffer 1 "+rgba"
    frame buffer 2 "+rgba"
end options

shader "red"
    "global_phong_with_light" (
        "diffuse" 1 .4 .4 )

shader "yellow"
    "global_phong_with_light" (
        "diffuse" .8 .7 .4 )

shader "blue"
    "global_phong_with_light" (
        "diffuse" .4 .4 1 )

shader "darkgray"
    "global_phong_with_light" (
        "diffuse" .1 .1 .1 )

shader "white"
    "global_phong_with_light" (
        "diffuse" 1 1 1 )

camera "cam"
    output "fb0" "tif"
        "buffer_5_diffuse.tif"
    output "fb1" "tif"
        "buffer_5_specular.tif"
    output "fb2" "tif"
        "buffer_5_indirect.tif"
    output "rgba" "tif"
        "buffer_5.tif"
    focal 90
    aperture 33.3
    aspect 1.4
    resolution 420 300
    environment
        "one_color" (
            "color" .1 .1 .2 )
end camera

instance "square-1"
    geometry "bw_square" (
        "name" "bw-square" )
    material "darkgray"
    transform
        0.1 0 0 0
        0 0.1 0 0
        0 0 0.1 0
        0 0 0.2 1
    globillum 3
end instance
```

Using a material Phenomenon to save the indirect illumination component

Exercise 23: Writing separate illumimation component files

1. Find the three places in `phenomena_E.mi` where:
    1. Frame buffers are created in the `options` block.
    2. Samples are put in the frame buffer by a shader.
    3. The frame buffer is written to a file.
2. Render `phenomena_E.mi` and display the four image files that are created. The filenames are defined in the camera object.
3. Phenomena can be used for scene-specific definition of shader parameters that should be the same whenever the Phenomena is used. Where is the specular color defined for all the objects?

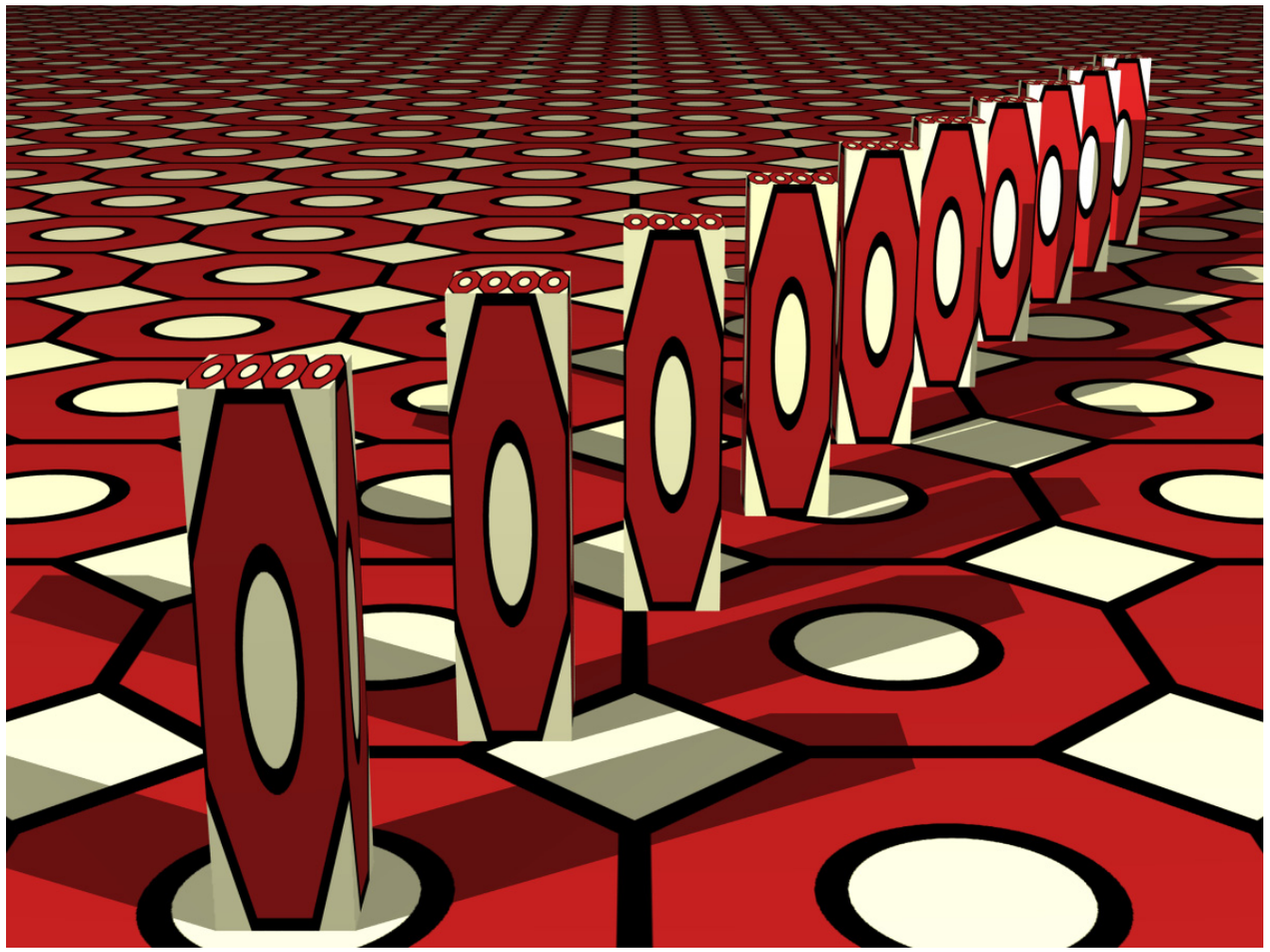
Modifying the final image

Modifying the final image

An image processing vocabulary
Adding a letterbox mask
A median filter in a shader
Compositing text over a rendered image
Multiple output shaders
Late binding and beyond

Modifying the final image

Adding a letterbox mask



```
camera "cam"
    output "rgba" "tif"
        "output_1.tif"
    focal 1.5
    aperture 1
    aspect 1.333
    resolution 400 300
end camera
```

Camera with standard file output and no output shaders

Modifying the final image

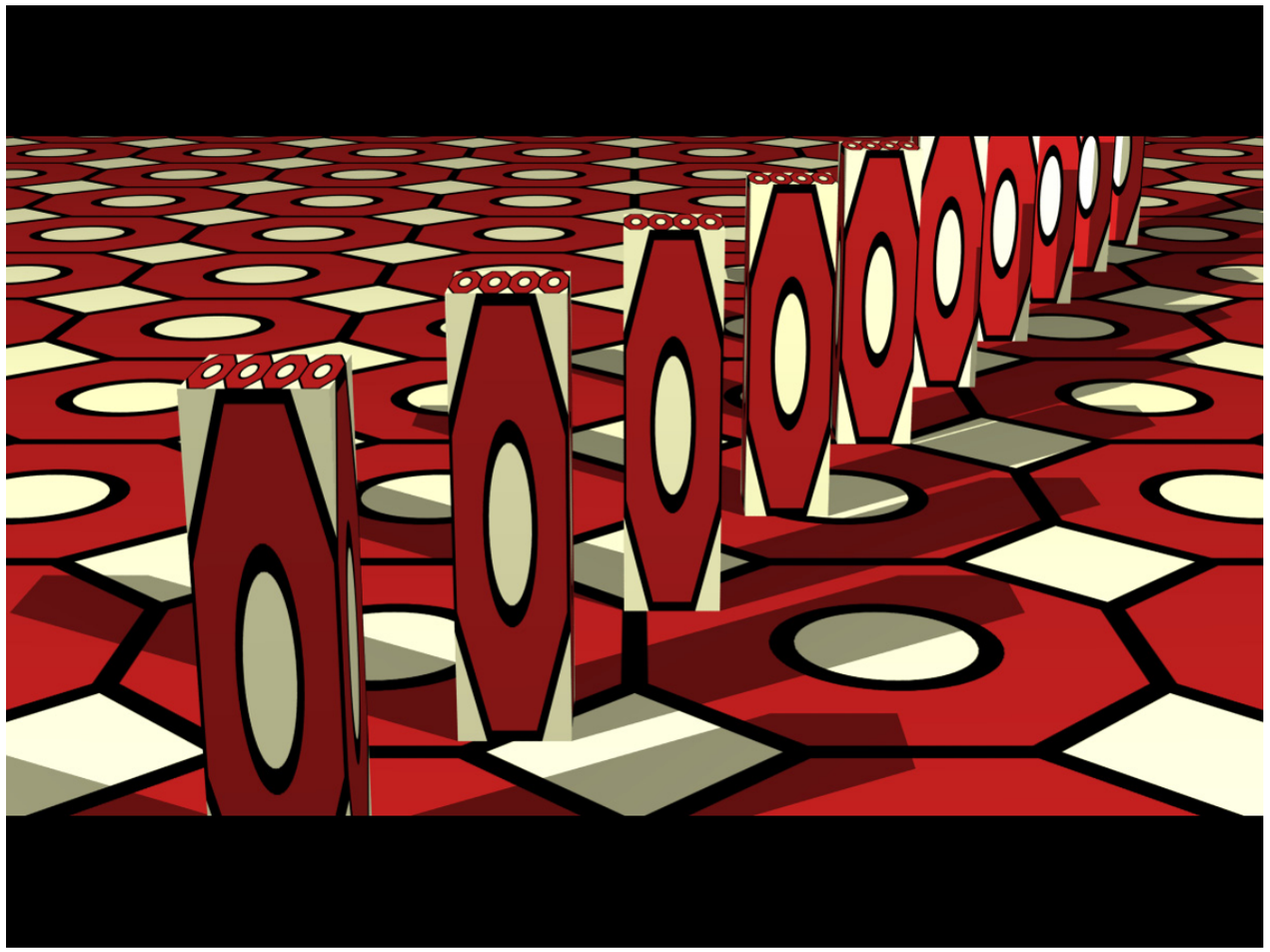
Adding a letterbox mask

```
declare shader
    color "letterbox" (
        scalar "aspect_ratio" default 1.85,
        color "outside_scale" default  0 0 0 )
end declare
```

Scene file declaration of shader "letterbox"

Modifying the final image

Adding a letterbox mask



```
camera "cam"
    output "rgba"
        "letterbox" ()
    output "rgba" "tif"
        "output_2.tif"
    focal 1.5
    aperture 1
    aspect 1.333
    resolution 400 300
end camera
```

Camera with output shader letterbox added

Modifying the final image

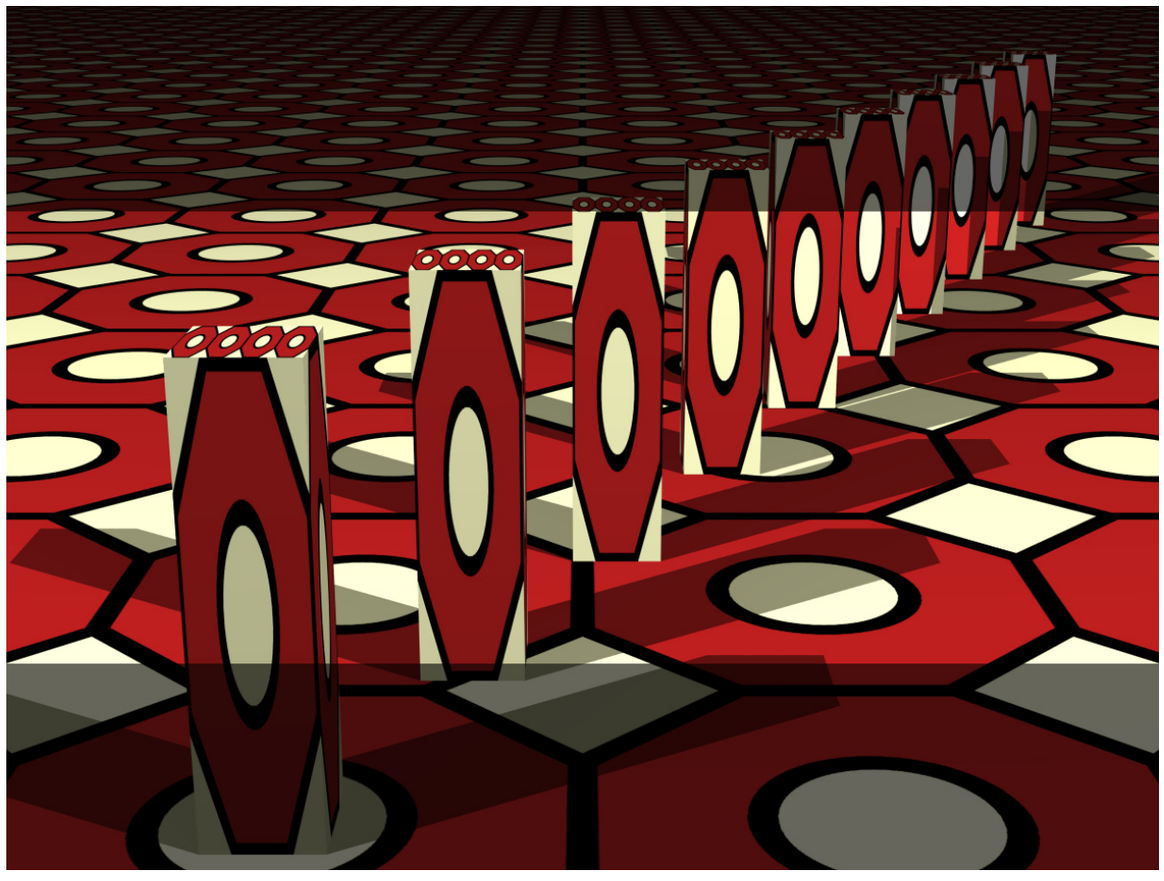
Adding a letterbox mask

```
 1  struct letterbox {
 2      miScalar aspect_ratio;
 3      miColor outside_scale;
 4  };
 5
 6  miBoolean letterbox(
 7      void *result, miState *state, struct letterbox  *params)
 8  {
 9      int x, y;
10      miColor pixel;
11      miScalar aspect_ratio = *mi_eval_scalar(&params->aspect_ratio);
12      miColor *outside_scale = mi_eval_color(&params->outside_scale);
13      miScalar image_width = state->camera->x_resolution;
14      miScalar image_height = state->camera->y_resolution;
15      miScalar letterbox_height = image_width / aspect_ratio;
16      miScalar y_min = (image_height - letterbox_height) / 2.0;
17      miScalar y_max = image_height - y_min;
18
19      miImg_image *fb = mi_output_image_open(state, miRC_IMAGE_RGBA);
20
21      for (y = 0; y < state->camera->y_resolution; y++) {
22          if (mi_par_aborted())
23              break;
24          for (x = 0; x < state->camera->x_resolution; x++) {
25              mi_img_get_color(fb, &pixel, x, y);
26              if (y < y_min || y > y_max) {
27                  miaux_multiply_colors(&pixel, &pixel, outside_scale);
28                  mi_img_put_color(fb, &pixel, x, y);
29              }
30          }
31      }
32      mi_output_image_close(state, miRC_IMAGE_RGBA);
33
34      return miTRUE;
35  }
```

Source code of shader "letterbox"

Modifying the final image

Adding a letterbox mask

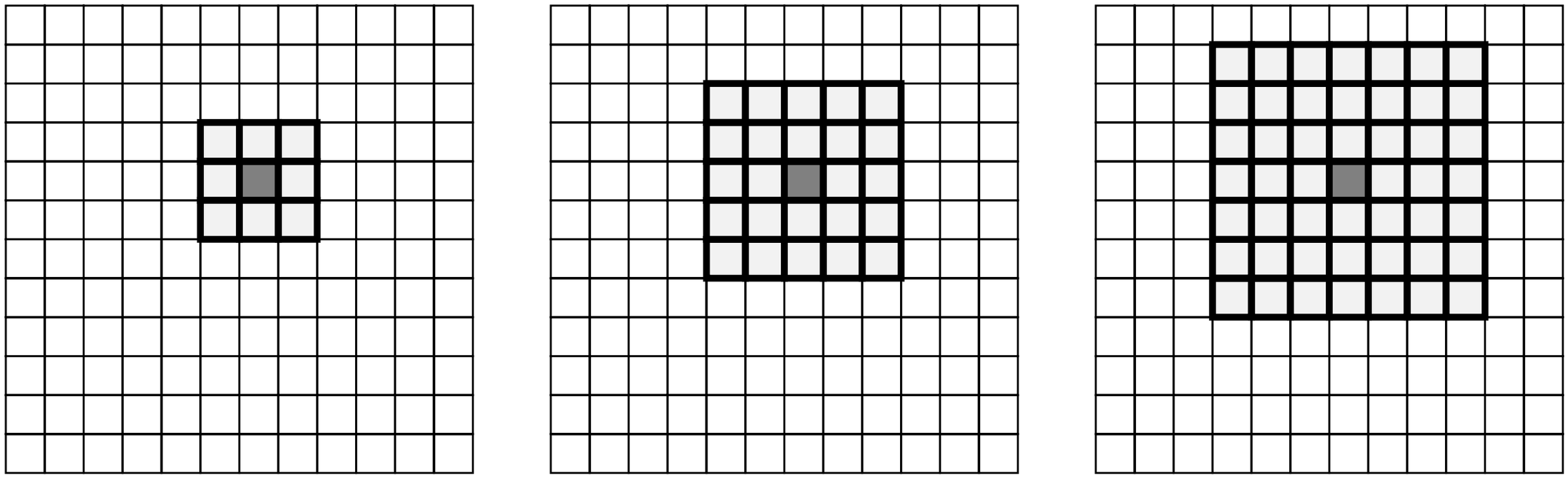


```
camera "cam"
   output "rgba"
      "letterbox" (
         "aspect_ratio" 2.55,
         "outside_scale" .4 .4 .4 )
   output "rgba" "tif" "output_3.tif"
   focal 1.5
   aperture 1
   aspect 1.333
   resolution 400 300
end camera
```

Camera with letterbox output shader with the Cinemascope aspect ratio of 2.55

Modifying the final image

A median filter in a shader



radius = 1

radius = 2

radius = 3

A radius defining a square region of pixels around a central pixel

Modifying the final image

A median filter in a shader

```
declare shader
    color "median_filter" (
        integer "radius" default 1 )
end declare
```

Scene file declaration of shader "median_filter"

Modifying the final image

A median filter in a shader

```
1  void miaux_pixel_neighborhood(
2      miColor *neighbors,
3      miImg_image *buffer, int x, int y, int radius)
4  {
5      int xi, yi, xp = 0, yp = 0, i = 0,
6          max_x = buffer->width - 1, max_y = buffer->height - 1;
7      miColor current_pixel;
8
9      for (yi = y - radius; yi <= y + radius; yi++) {
10         yp = yi > max_y ? max_x : yi < 0 ? 0 : yi;
11         for (xi = x - radius; xi <= x + radius; xi++) {
12             xp = xi > max_x ? max_x : xi < 0 ? 0 : xi;
13             mi_img_get_color(buffer, &current_pixel, xp, yp);
14             neighbors[i++] = current_pixel;
15         }
16     }
17 }
```

Auxiliary function: miaux_pixel_neighborhood

Modifying the final image


```
1  int miaux_color_compare(const void *vx, const void *vy)
2  {
3      miColor const *x = vx, *y = vy;
4      float sum_x = x->r + x->g + x->b;
5      float sum_y = y->r + y->g + y->b;
6      return sum_x < sum_y ? -1 : sum_x > sum_y ? 1 : 0;
7  }
```

Auxiliary function: miaux_color_compare

Modifying the final image

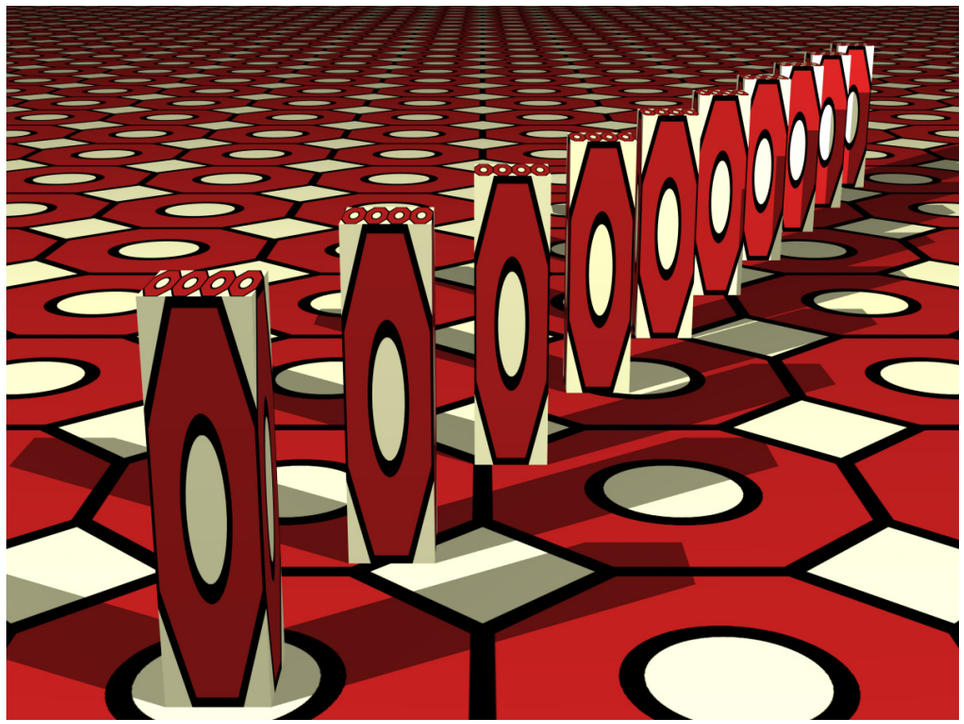
A median filter in a shader

```
 1  struct median_filter {
 2      miInteger radius;
 3  };
 4
 5  miBoolean median_filter(
 6      void *result, miState *state, struct median_filter *params)
 7  {
 8      miColor *neighbors;
 9      int radius = *mi_eval_integer(&params->radius), x, y,
10          kernel_size = (radius * 2 + 1) * (radius * 2 + 1),
11          middle = kernel_size / 2;
12
13      miImg_image *fb_input = mi_output_image_open(state, miRC_IMAGE_RGBA);
14      miImg_image *fb_output = mi_output_image_open(state, miRC_IMAGE_USER);
15
16      neighbors = (miColor*) mi_mem_allocate(kernel_size * sizeof(miColor));
17      for (y = 0; y < state->camera->y_resolution; y++) {
18          if (mi_par_aborted())
19              break;
20          for (x = 0; x < state->camera->x_resolution; x++) {
21              miaux_pixel_neighborhood(neighbors, fb_input, x, y, radius);
22              qsort(neighbors, kernel_size, sizeof(miColor),
23                    miaux_color_compare);
24              mi_img_put_color(fb_output, &neighbors[middle], x, y);
25          }
26      }
27      mi_mem_release(neighbors);
28
29      mi_output_image_close(state, miRC_IMAGE_RGBA);
30      mi_output_image_close(state, miRC_IMAGE_USER);
31
32      return miTRUE;
33  }
```

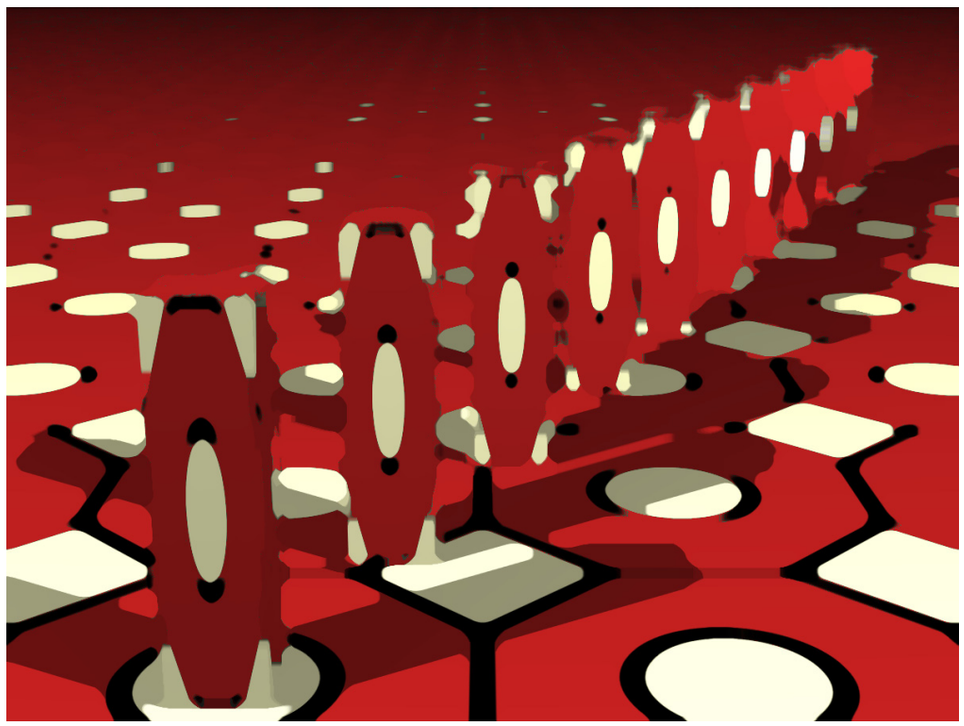
Source code of shader "median_filter"

Modifying the final image

A median filter in a shader



output_4.tif



output_4_median.tif

```
options "opt"
    object space
    samples 0 2
    contrast .1 .1 .1 1
    trace depth 2 2 4
    frame buffer 0 "+rgba"
end options

camera "cam"
   output "rgba"
      "median_filter" (
         "radius" 4 )
   output "rgba" "tif"
      "output_4.tif"
   output "fb0" "tif"
      "output_4_median.tif"
   focal 1.5
   aperture 1
   aspect 1.333
   resolution 400 300
end camera
```

Using the first user frame buffer as the output of an image processing operation

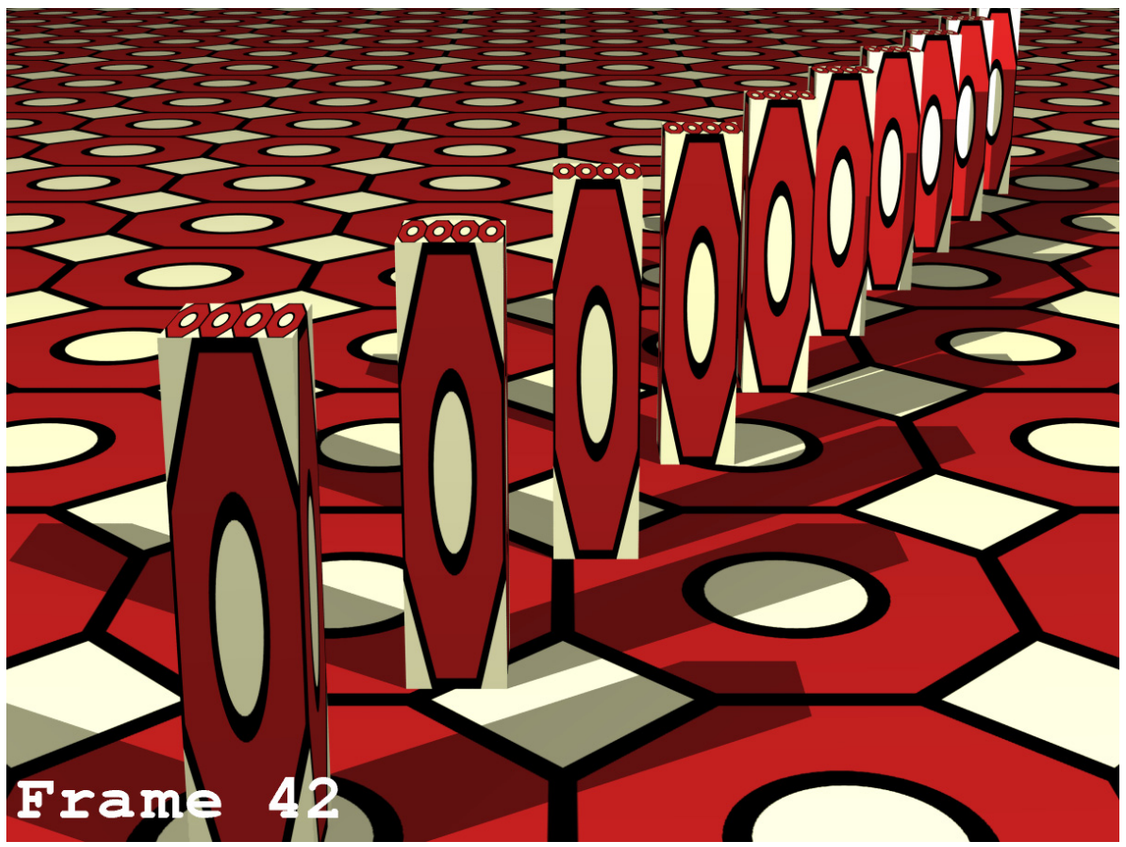
Modifying the final image


```
1  void miaux_copy_frame_buffer(
2      miImg_image *source, miImg_image *destination)
3  {
4      int x, y;
5      miColor pixel;
6      for (y = 0; y < source->height; y++)
7          for (x = 0; x < source->width; x++) {
8              mi_img_get_color(source, &pixel, x, y);
9              mi_img_put_color(destination, &pixel, x, y);
10         }
11 }
```

Auxiliary function: miaux_copy_frame_buffer

Modifying the final image

Compositing text over a rendered image




```
camera "cam"
   output "rgba"
      "annotate" (
         "fontimage_filename"
            "Courier-Bold_24.fontimage",
         "text" "Frame 42" )
   output "rgba" "tif"
      "output_5.tif"
   focal 50
   aperture 33.3
   aspect 1.5
   resolution 400 300
end camera
```

Adding text to an image using the annotate output shader in the camera

Modifying the final image

Compositing text over a rendered image

```
declare shader
    color "annotate" (
        string "text",
        string "fontimage_filename",
        integer "x" default 10,
        integer "y" default 10,
        color "color" default 1 1 1 )
end declare
```

Scene file declaration of shader "annotate"

Modifying the final image

Compositing text over a rendered image

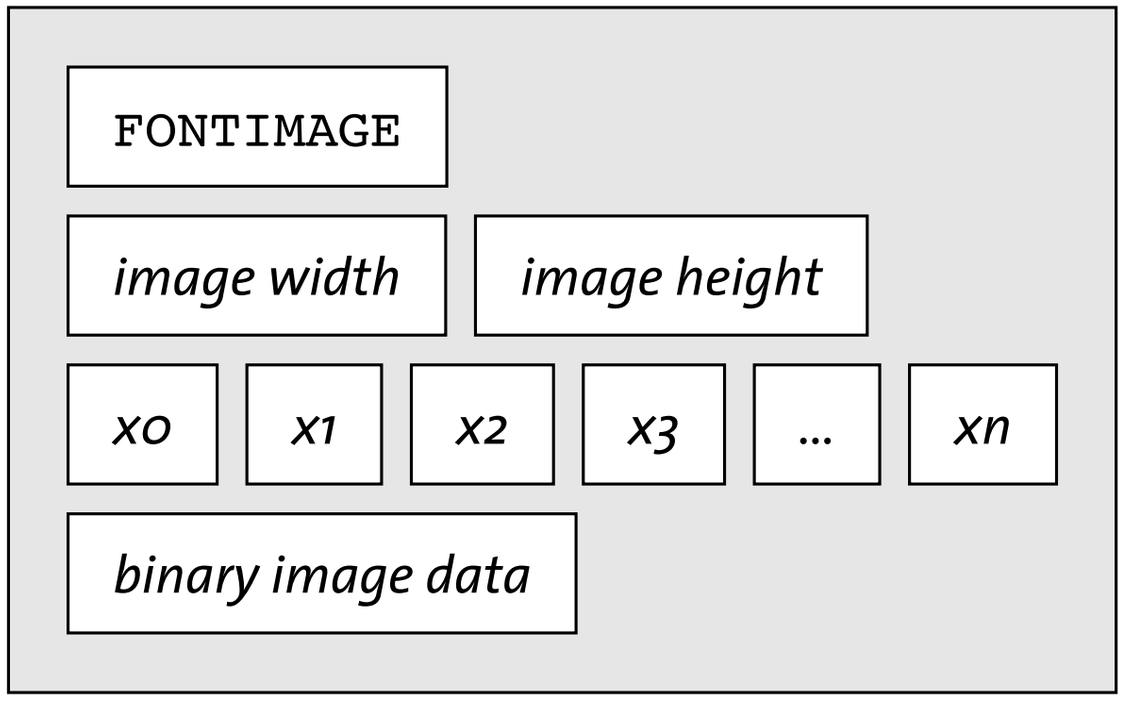
Part of the fontimage grayscale data for 24 point Helvetica

Identification string

Width and height of font image

Horizontal offset for each character

Image data, one byte per pixel, row major order



Data format of ``fontimage'' file

Modifying the final image

Compositing text over a rendered image

```
typedef struct {
    int width;
    int height;
    int *x_offsets;
    unsigned char *image;
} fontimage;
```

The C struct for a fontimage

Modifying the final image

Compositing text over a rendered image

```
 1  void miaux_load_fontimage(fontimage *fimage, char* filename)
 2  {
 3      int printable_ascii_size = 126 - 32 + 1;
 4      int i, image_data_size;
 5      char identifier[33];
 6
 7      FILE* fp = fopen(filename, "r");
 8      if (fp == NULL)
 9          mi_fatal("Could not open font image file: %s", filename);
10      fscanf(fp, "%s ", identifier);
11      if (strcmp(identifier, "FONTIMAGE") != 0)
12          mi_fatal("File '%s' does not look like a fontimage file", filename);
13
14      fscanf(fp, "%d %d ", &fimage->width, &fimage->height);
15      image_data_size = fimage->width * fimage->height;
16      fimage->x_offsets =
17          (int*)mi_mem_allocate(sizeof(int) * printable_ascii_size);
18      for (i = 0; i <= printable_ascii_size; i++)
19          fscanf(fp, "%d ", &fimage->x_offsets[i]);
20      fimage->image = (unsigned char*)mi_mem_allocate(image_data_size);
21      fread(fimage->image, image_data_size, 1, fp);
22
23      fclose(fp);
24  }
```

Auxiliary function: miaux_load_fontimage

```
1  void miaux_release_fontimage(fontimage *fimage)
2  {
3      mi_mem_release(fimage->x_offsets);
4      mi_mem_release(fimage->image);
5  }
```

Auxiliary function: miaux_release_fontimage

Modifying the final image

Compositing text over a rendered image

```
 1  void miaux_text_image(
 2      float **text_image, int *width, int *height,
 3      fontimage *fimage, char* text)
 4  {
 5      char *c;
 6      int i, total_width, xpos, first_printable = 32, image_size,
 7          index, start, end, font_index, fx, fy, tx, ty, text_index;
 8
 9      for (i = 0, total_width = 0, c = text; i < strlen(text); i++, c++) {
10          index = *c - first_printable;
11          start = fimage->x_offsets[index];
12          end = fimage->x_offsets[index + 1];
13          total_width += end - start + 1;
14      }
15      *width = total_width;
16      *height = fimage->height;
17      image_size = *width * *height;
18      (*text_image) = (float*)mi_mem_allocate(image_size * sizeof(float));
19
20      for (i = 0, c = text, xpos = 0; i < strlen(text); i++, c++) {
21          int index = *c - first_printable;
22          start = fimage->x_offsets[index];
23          end = fimage->x_offsets[index + 1];
24          for (fy = fimage->height - 1, ty = 0; fy >= 0; fy--, ty++) {
25              for (fx = start, tx = xpos; fx < end-1; fx++, tx++) {
26                  text_index = ty * *width + tx;
27                  font_index = fy * fimage->width + fx;
28                  (*text_image)[text_index] =
29                      (float)fimage->image[font_index] / 255.0;
30              }
31          }
32          xpos += end - start + 1;
33      }
34  }
```

Auxiliary function: miaux_text_image

Modifying the final image

Compositing text over a rendered image

```
 1  struct annotate {
 2      miTag text;
 3      miTag fontimage_filename;
 4      miInteger x;
 5      miInteger y;
 6      miColor color;
 7  };
 8
 9  miBoolean annotate(
10      void *result, miState *state, struct annotate *params)
11  {
12      if (params->text != miNULLTAG) {
13          int x, y, t_x, t_y, t_width, t_height;
14          float *text_image;
15          miImg_image *fb;
16          fontimage fimage;
17          miColor *text_color = mi_eval_color(&params->color), pixel_color;
18          char* text = miaux_tag_to_string(
19              *mi_eval_tag(&params->text), NULL);
20          int p_x = *mi_eval_integer(&params->x),
21              p_y = *mi_eval_integer(&params->y);
22          char* fontimage_filename =
23              miaux_tag_to_string(*mi_eval_tag(&params->fontimage_filename),
24                                  "Courier-Bold_24.fontimage");
25
26          miaux_load_fontimage(&fimage, fontimage_filename);
27          miaux_text_image(&text_image, &t_width, &t_height, &fimage, text);
28
29          fb = mi_output_image_open(state, miRC_IMAGE_RGBA);
30          for (y = p_y, t_y = 0; t_y < t_height; y++, t_y++) {
31              if (mi_par_aborted()) {
32                  mi_progress("Abort");
33                  break;
34              }
35              for (x = p_x, t_x = 0; t_x < t_width; x++, t_x++) {
36                  float alpha = text_image[t_y * t_width + t_x];
37                  mi_img_get_color(fb, &pixel_color, x, y);
38                  miaux_alpha_blend(&pixel_color, text_color, alpha);
39                  mi_img_put_color(fb, &pixel_color, x, y);
40              }
41          }
42          miaux_release_fontimage(&fimage);
43          mi_output_image_close(state, miRC_IMAGE_RGBA);
44      }
45      return miTRUE;
46  }
```

Source code of shader "annotate"

Modifying the final image

Multiple output shaders


```
camera "cam"
   output "rgba"
      "letterbox" (
         "aspect_ratio" 1.778,
         "outside_scale" .3 .3 .3 )
   output "rgba"
      "annotate" (
         "text" "HDTV aspect ratio: 16:9",
         "fontimage_filename"
            "Helvetica-Bold_20.fontimage",
         "color" 1 1 .5,
         "x" 20, "y" 270 )
   output "rgba"
      "annotate" (
         "fontimage_filename"
            "Courier-Bold_24.fontimage",
         "text" "00:55:05:15",
         "color" .8 .8 1,
         "x" 20, "y" 7 )
   output "rgba" "tif" "output_6.tif"
   focal 1.5
   aperture 1
   aspect 1.333
   resolution 400 300
end camera
```

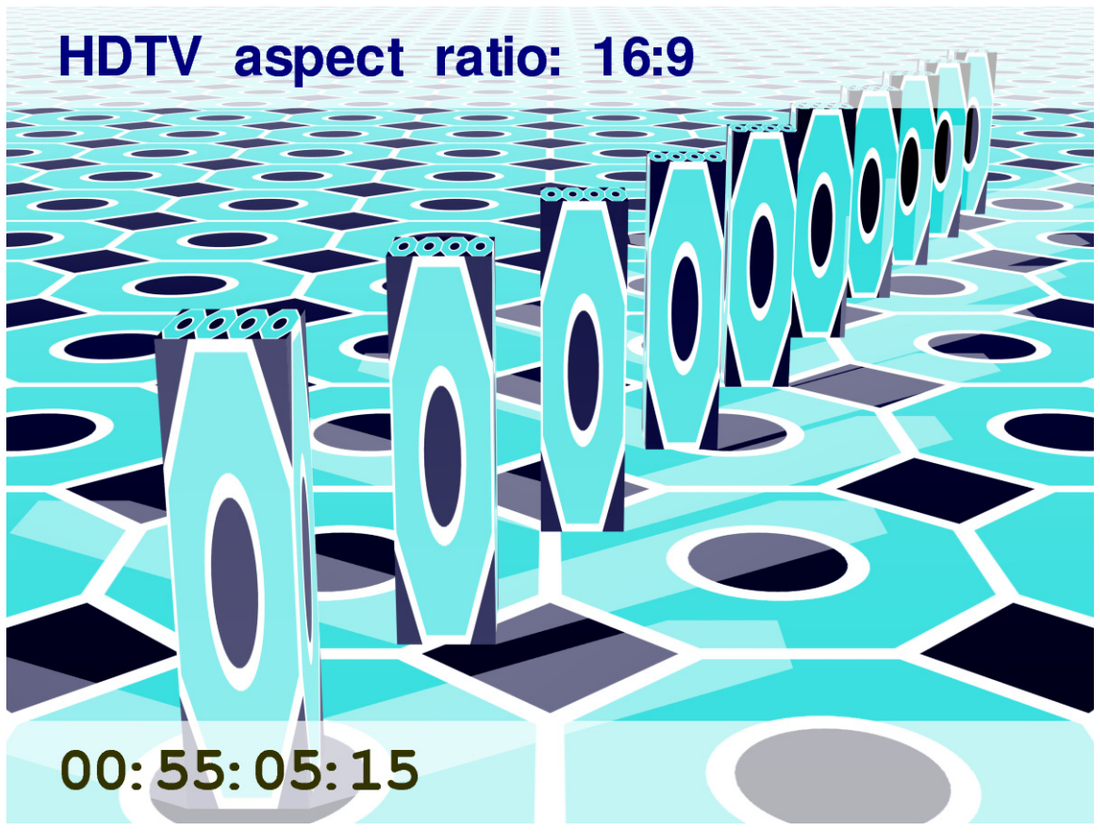
Using multiple output shaders

Modifying the final image


```
// C++

#include "shader.h"

extern "C"
int negate_version(void) { return 1; }

extern "C"
miBoolean negate(void *result, miState *state, void *params)
{
    miImg_image *fb = mi_output_image_open(state, miRC_IMAGE_RGBA);
    for (int y = 0; y < state->camera->y_resolution; y++) {
        for (int x = 0; x < state->camera->x_resolution; x++) {
            miColor pixel;
            mi_img_get_color(fb, &pixel, x, y);
            pixel.r = 1.0 - pixel.r;
            pixel.g = 1.0 - pixel.g;
            pixel.b = 1.0 - pixel.b;
            mi_img_put_color(fb, &pixel, x, y);
        }
    }
    mi_output_image_close(state, miRC_IMAGE_RGBA);
    return miTRUE;
}
```

Source code for shader negate with linkage defined by the extern "C" qualifier

Modifying the final image

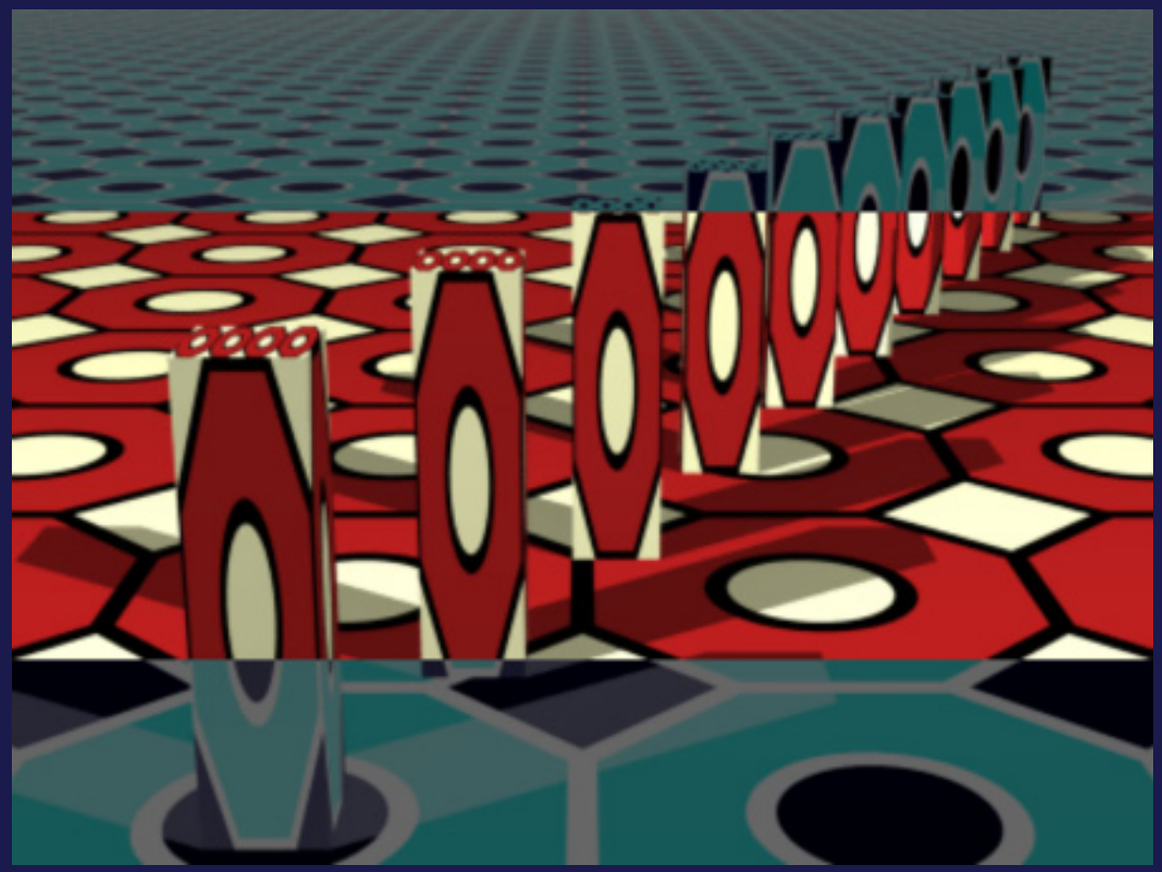



```
camera "cam"
   output "rgba"
      "letterbox" (
         "aspect_ratio" 1.778,
         "outside_scale" .3 .3 .3 )
   output "rgba"
      "annotate" (
         "text" "HDTV aspect ratio: 16:9",
         "fontimage_filename"
            "Helvetica-Bold_20.fontimage",
         "color" 1 1 .5,
         "x" 20, "y" 270 )
   output "rgba"
      "annotate" (
         "fontimage_filename"
            "Courier-Bold_24.fontimage",
         "text" "00:55:05:15",
         "color" .8 .8 1,
         "x" 20, "y" 7 )
   output "rgba"
      "negate" ()
   output "rgba" "tif" "output_7.tif"
   focal 1.5
   aperture 1
   aspect 1.333
   resolution 400 300
end camera
```

Mixing shaders written in C and CPP

Exercise 24: Output shaders

1. Copy `output_3.mi` to `output.mi`, change output file to `output.tif`, render and view.
2. Change the color scaling and aspect ratio parameters of shader `letterbox`
3. Modify shader `letterbox` to negate the outside area before scaling the color.



Exercise 24: Output shaders (part 2)

Modify shader `letterbox` to negate the outside area before scaling the color.

Old:

```
mi_img_get_color(fb, &pixel, x, y);
if (y < y_min || y > y_max) {
    miaux_multiply_colors(&pixel, &pixel, outside_scale);
    mi_img_put_color(fb, &pixel, x, y);
}
```

New:

```
mi_img_get_color(fb, &pixel, x, y);
if (y < y_min || y > y_max) {
   pixel.r = 1.0 - pixel.r;
   pixel.g = 1.0 - pixel.g;
   pixel.b = 1.0 - pixel.b;
   miaux_multiply_colors(&pixel, &pixel, outside_scale);
   mi_img_put_color(fb, &pixel, x, y);
}
```