Writing mental ray shaders

by Andy Kopra

Part 4: Shape

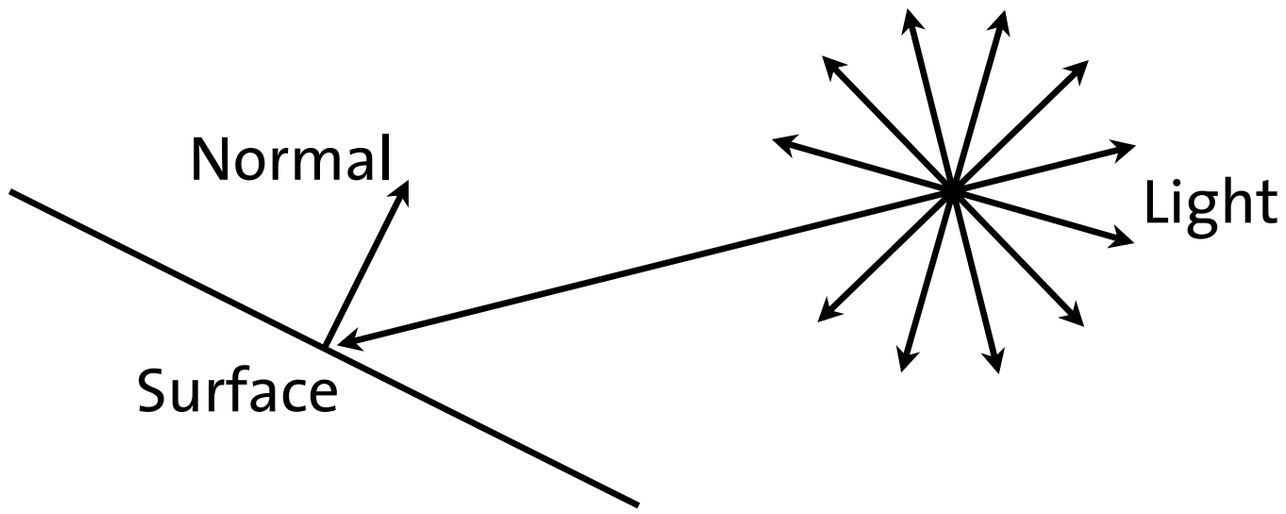
Modifying surface geometry

Modifying surface geometry

- Surface position and the normal vector
- A simple displacement shader
  - Using texture coordinates
- Shader lists and displacement mapping
- Using texture images to control displacement mapping
- Combining displacement mapping with color
- Using noise functions with displacement mapping

Modifying surface geometry

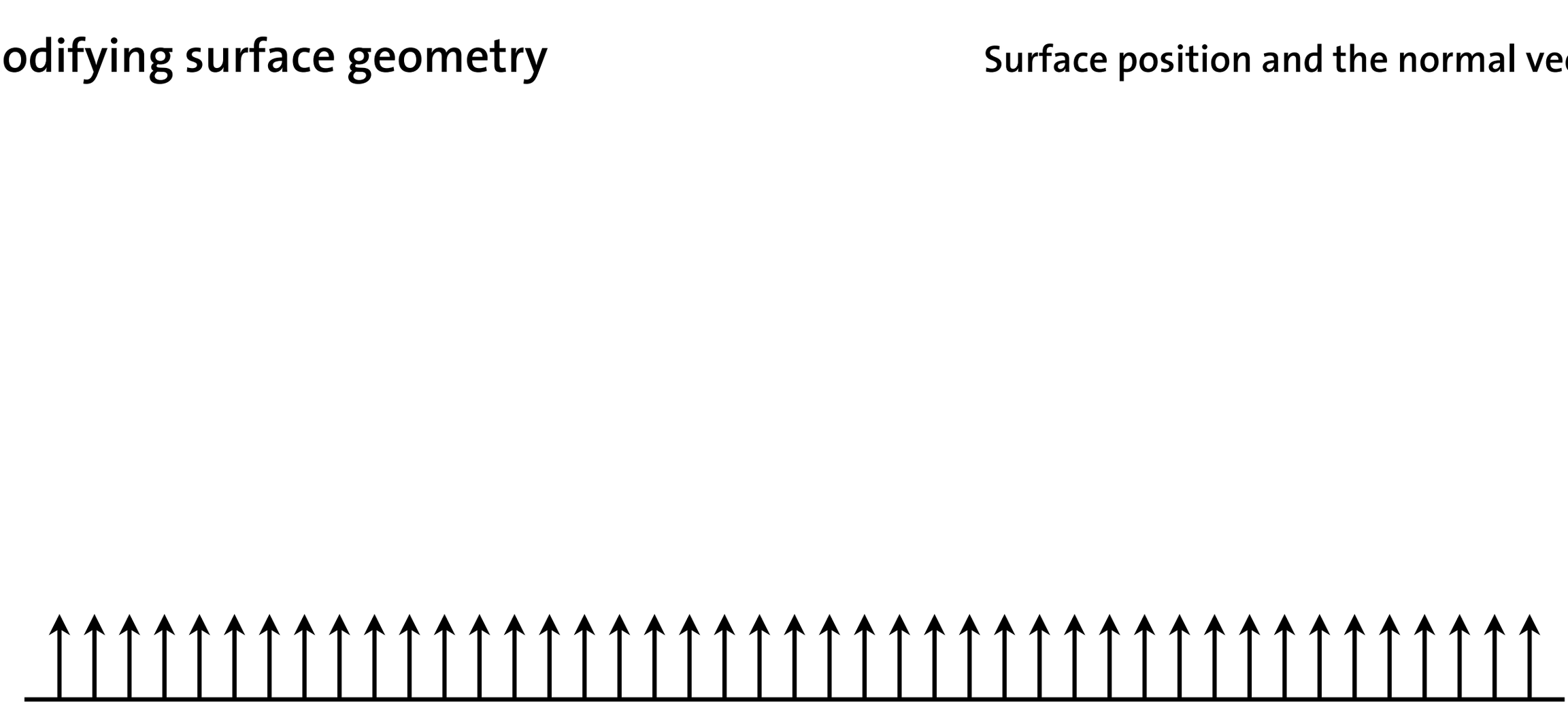
Surface position and the normal vector



The orientation of the light ray at a point on the surface

Modifying surface geometry

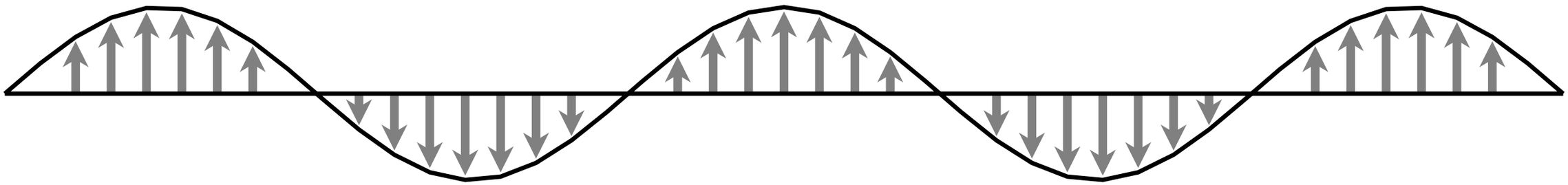
Surface position and the normal vector



The orientation of a flat surface

Modifying surface geometry

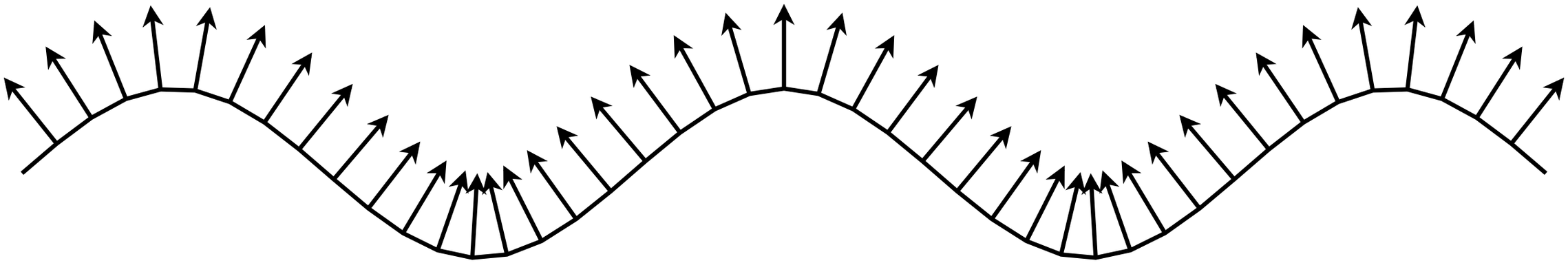
Surface position and the normal vector



Points on the surface are displaced to new positions

Modifying surface geometry

Surface position and the normal vector



The new surface with its normal vectors

Modifying surface geometry

A simple displacement shader

```
material "lambert"
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
end material
```

A polygon with lambert used as its material shader

```
1  double miaux_sinusoid(double v, double frequency, double amplitude)
2  {
3      return sin(v * frequency * M_PI * 2.0) * amplitude;
4  }
```

Auxiliary function: miaux_sinusoid

Modifying surface geometry

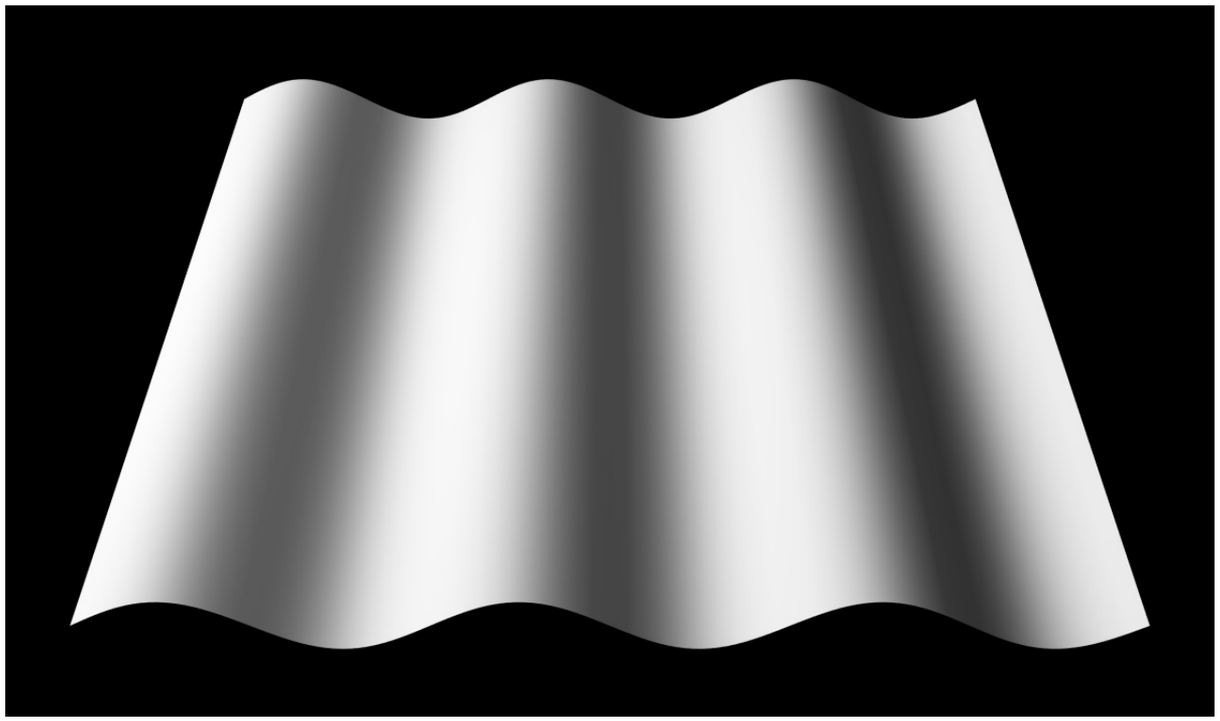
A simple displacement shader

```
declare shader
    scalar "displace_wave" (
        scalar "frequency" default 1,
        scalar "amplitude" default .5 )
end declare
```

Scene file declaration of shader "displace_wave"

Modifying surface geometry

A simple displacement shader



```
material "wave"
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
    displace "displace_wave" (
        "frequency" 3,
        "amplitude" .03 )
end material
```

Displacement based on the world space coordinates of a polygon

Modifying surface geometry

A simple displacement shader

```
 1  struct displace_wave {
 2      miScalar frequency;
 3      miScalar amplitude;
 4  };
 5
 6  miBoolean displace_wave (
 7      miScalar *result, miState *state, struct displace_wave *params )
 8  {
 9      *result += miaux_sinusoid(state->point.x,
10                                *mi_eval_scalar(&params->frequency),
11                                *mi_eval_scalar(&params->amplitude));
12      return miTRUE;
13  }
```

Source code of shader "displace_wave"

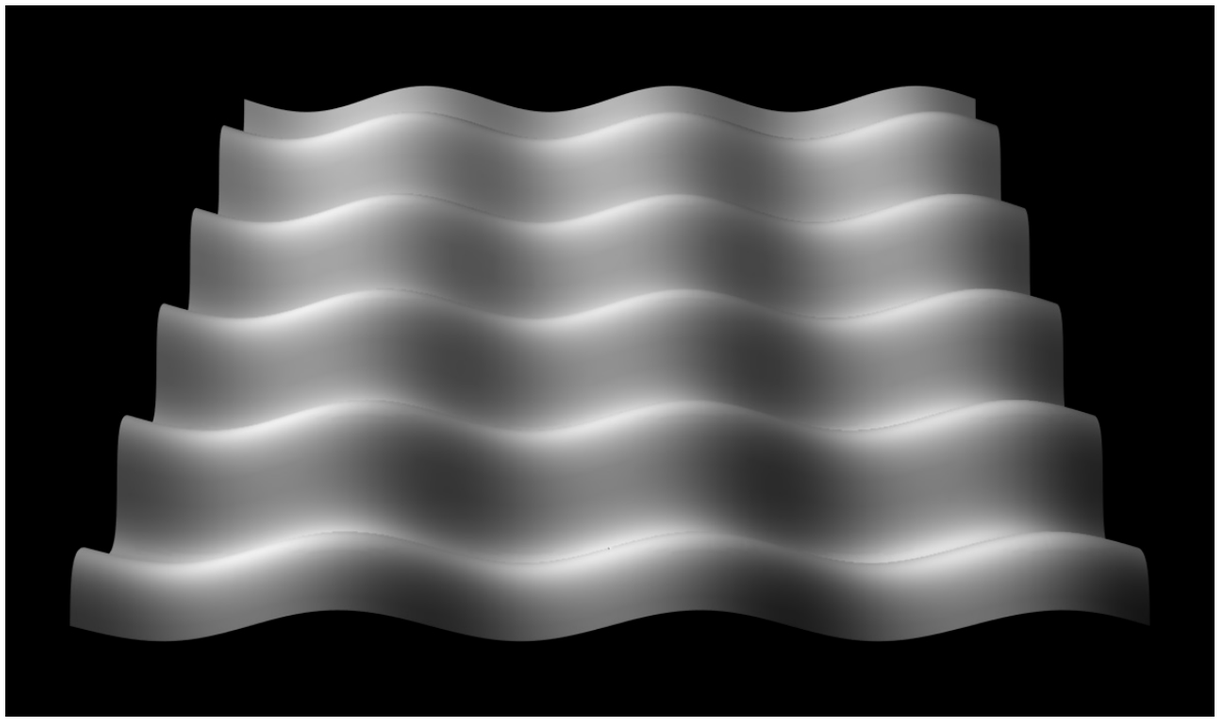
A simple displacement shader

```
declare shader
    scalar "displace_wave_uv" (
        scalar "frequency_u" default 1,
        scalar "amplitude_u" default .5,
        scalar "frequency_v" default 1,
        scalar "amplitude_v" default .5 )
end declare
```

Scene file declaration of shader "displace_wave_uv"

Modifying surface geometry

A simple displacement shader



```
material "wave_uv"
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
    displace "displace_wave_uv" (
        "frequency_u" 3,
        "amplitude_u" .02,
        "frequency_v" 5,
        "amplitude_v" .04 )
end material
```

Displacement based on the texture space coordinates of a polygon

Modifying surface geometry

A simple displacement shader

```
 1  struct displace_wave_uv {
 2      miScalar frequency_u;
 3      miScalar amplitude_u;
 4      miScalar frequency_v;
 5      miScalar amplitude_v;
 6  };
 7
 8  miBoolean displace_wave_uv (
 9      miScalar *result, miState *state, struct displace_wave_uv *params )
10  {
11      *result +=
12          miaux_sinusoid(state->tex_list[0].x,
13                         *mi_eval_scalar(&params->frequency_u),
14                         *mi_eval_scalar(&params->amplitude_u)) +
15          miaux_sinusoid(state->tex_list[0].y,
16                         *mi_eval_scalar(&params->frequency_v),
17                         *mi_eval_scalar(&params->amplitude_v));
18      return miTRUE;
19  }
```

Source code of shader "displace_wave_uv"

Modifying surface geometry

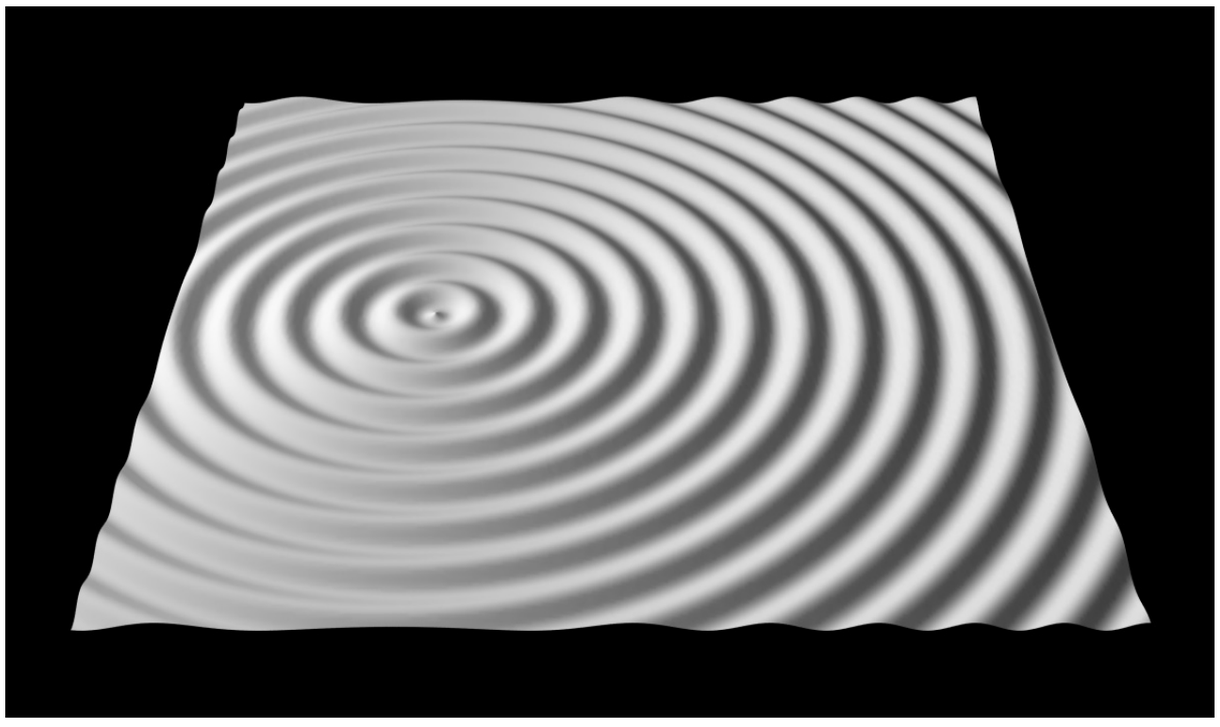
A simple displacement shader

```
declare shader
    scalar "displace_ripple" (
        vector "center"    default .5 .5 0,
        scalar "frequency" default 1,
        scalar "amplitude" default .1 )
end declare
```

Scene file declaration of shader "displace_ripple"

Modifying surface geometry

A simple displacement shader



```
material "ripple"
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
    displace
        "displace_ripple" (
            "center" .3 .5 0,
            "frequency" 16,
            "amplitude" .005 )
end material
```

Circular ripples defined by the distance from a texture coordinate position in a polygon

Modifying surface geometry

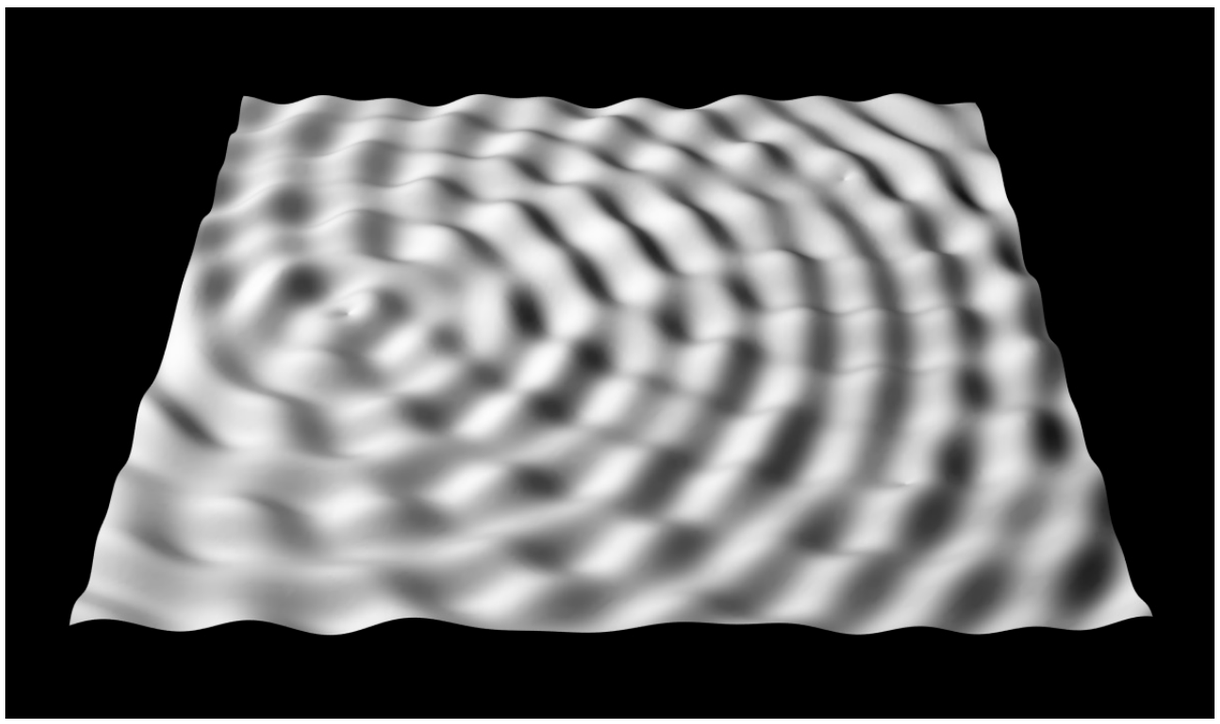
A simple displacement shader

```
 1  struct displace_ripple {
 2      miVector center;
 3      miScalar frequency;
 4      miScalar amplitude;
 5  };
 6
 7  miBoolean displace_ripple (
 8      miScalar *result, miState *state, struct displace_ripple *params )
 9  {
10      *result += miaux_sinusoid(mi_vector_dist(mi_eval_vector(&params->center),
11                                               &state->tex_list[0]),
12                                *mi_eval_scalar(&params->frequency),
13                                *mi_eval_scalar(&params->amplitude));
14      return miTRUE;
15  }
```

Source code of shader "displace_ripple"

Modifying surface geometry

Shader lists and displacement mapping

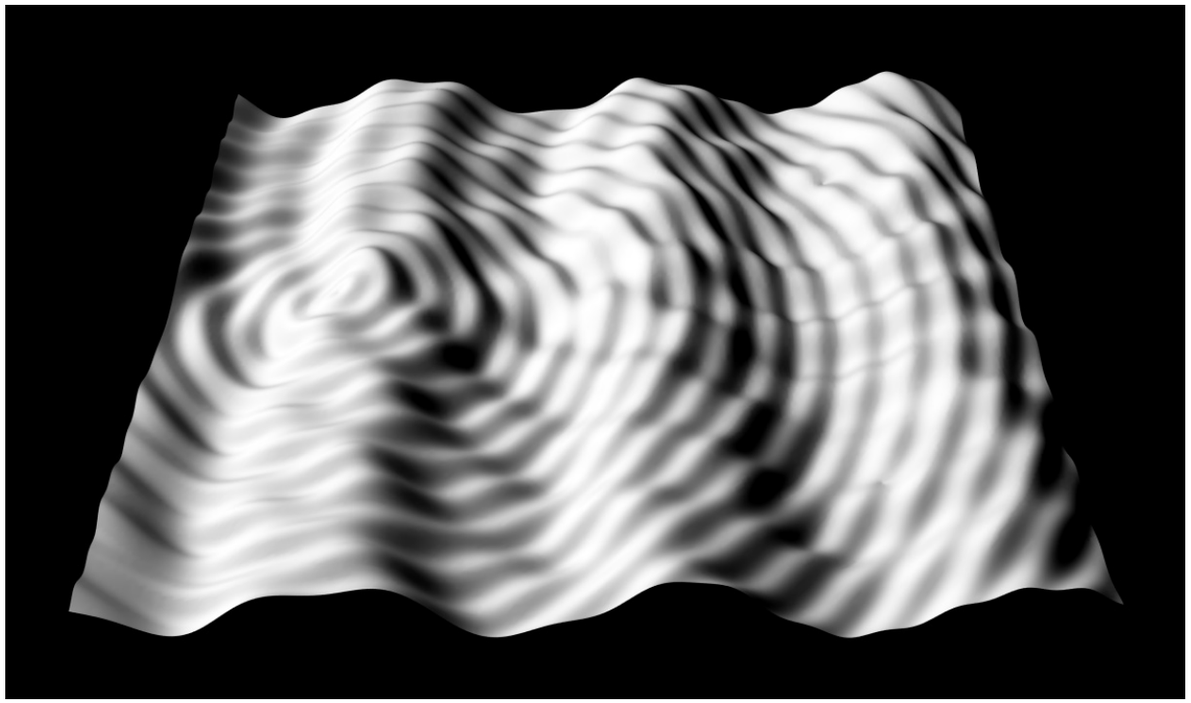


```
material "three_ripples"
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
    displace
        "displace_ripple" (
            "center" .2 .5 0,
            "frequency" 12,
            "amplitude" .005 )
        "displace_ripple" (
            "center" .8 .8 0,
            "frequency" 10,
            "amplitude" .005 )
        "displace_ripple" (
            "center" .8 .2 0,
            "frequency" 8,
            "amplitude" .005 )
end material
```

List of three displacement shaders

Modifying surface geometry

Shader lists and displacement mapping



```
material "waves_with_three_ripples"
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
    displace
        "displace_wave" (
            "frequency" 3,
            "amplitude" .03 )
        "displace_ripple" (
            "center" .2 .5 0,
            "frequency" 20,
            "amplitude" .003 )
        "displace_ripple" (
            "center" .8 .8 0,
            "frequency" 10,
            "amplitude" .005 )
        "displace_ripple" (
            "center" .8 .2 0,
            "frequency" 8,
            "amplitude" .005 )
end material
```

Displacement shaders with different levels of detail

Modifying surface geometry

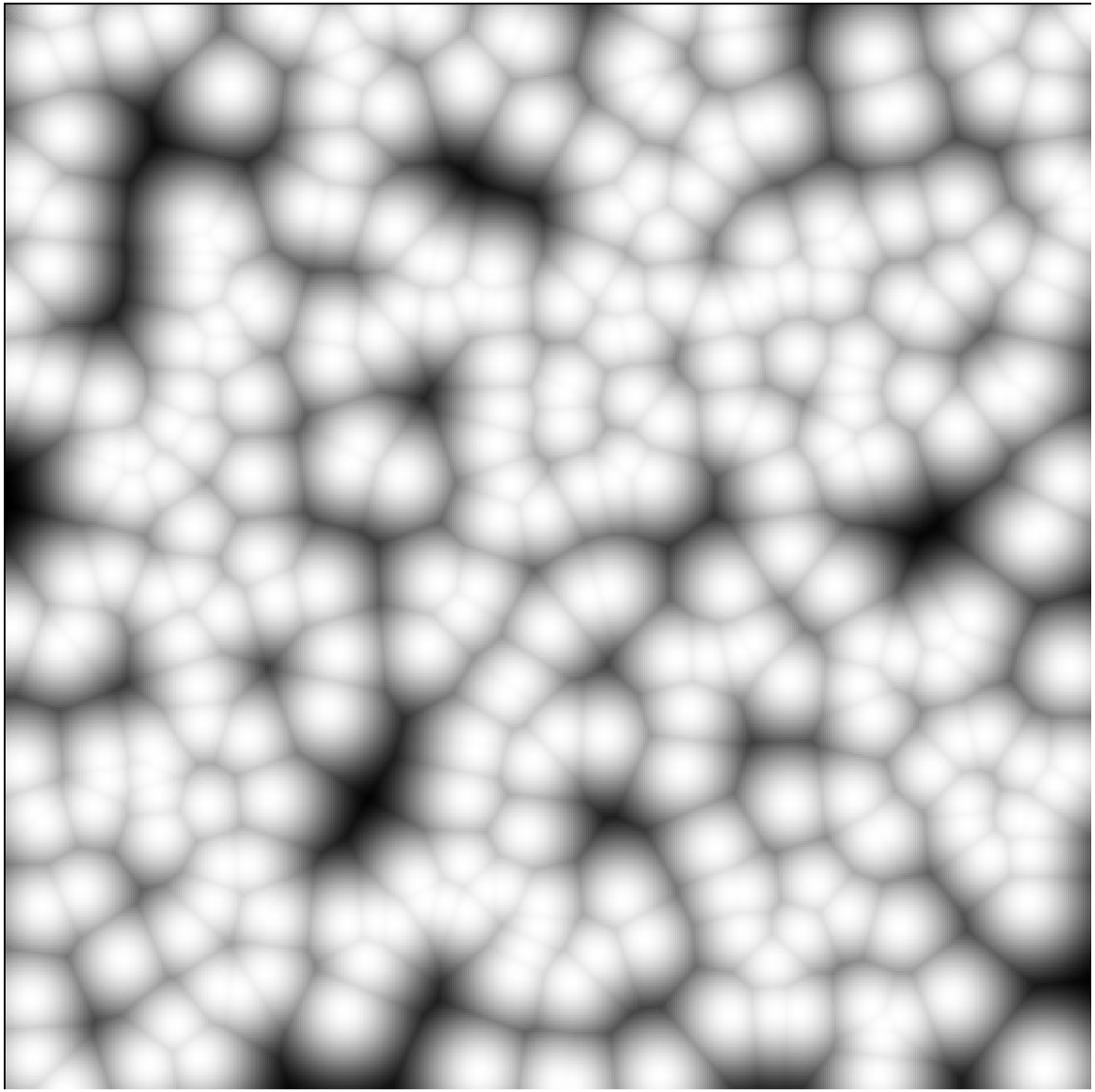
Using texture images to control displacement mapping

```
declare shader
    scalar "displace_texture" (
        color texture "texture",
        scalar "factor" default .1 )
end declare
```

Scene file declaration of shader "displace_texture"

Modifying surface geometry

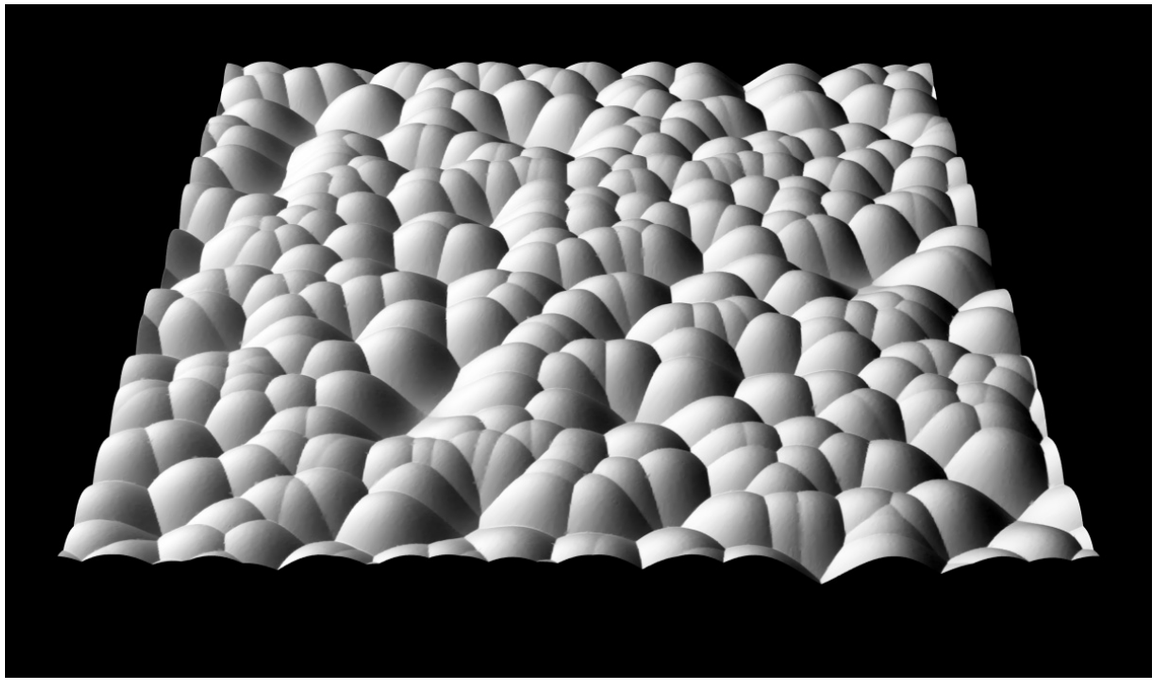
Using texture images to control displacement mapping



An image for use as a displacement map

Modifying surface geometry

Using texture images to control displacement mapping



```
color texture "bubbles" "bubbles.tif"

material "bubble"
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
    displace "displace_texture" (
        "texture" "bubbles",
        "factor" .05 )
end material
```

Texture map samples as displacement values

Modifying surface geometry

Using texture images to control displacement mapping

```
1  struct displace_texture {
2      miTag texture;
3      miScalar factor;
4  };
5
6  miBoolean displace_texture (
7      miScalar *result, miState *state, struct displace_texture *params)
8  {
9      miColor color;
10     mi_lookup_color_texture(&color, state,
11                             *mi_eval_tag(&params->texture),
12                             &state->tex_list[0]);
13     *result += *mi_eval_scalar(&params->factor) * color.r;
14     return miTRUE;
15 }
```

Source code of shader "displace_texture"

Modifying surface geometry

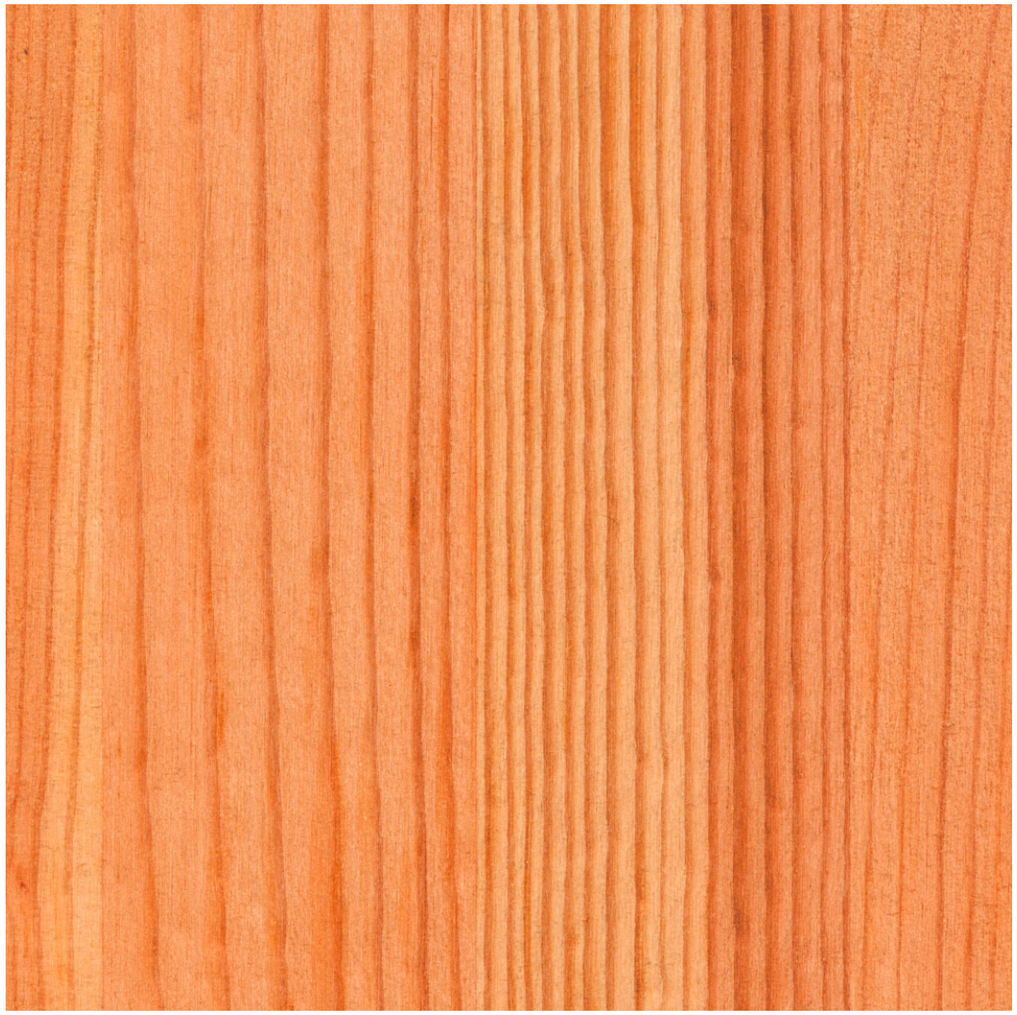
Combining displacement mapping with color

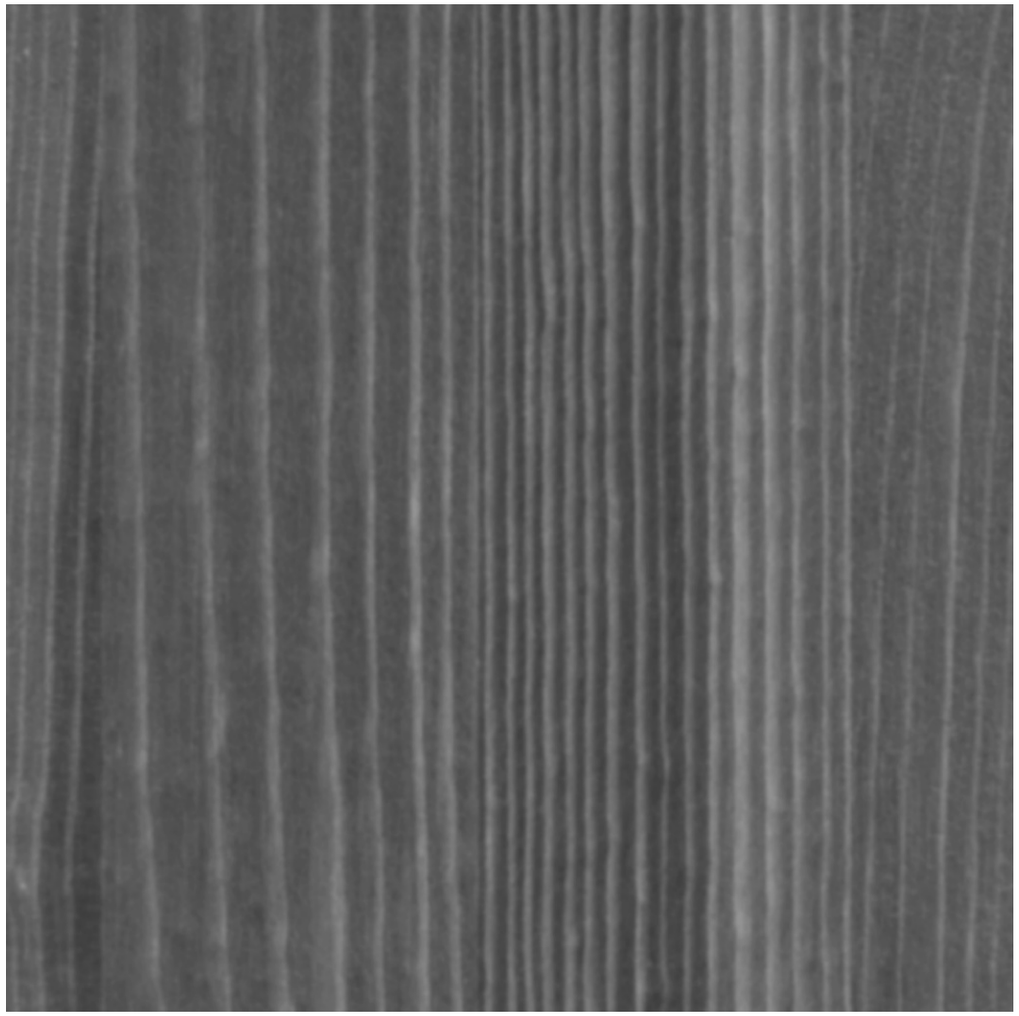
```
shader "wood_color"
    "texture_uv" (
        "tex" "wood_color_texture" )

material "wood"
    "lambert" (
        "diffuse" = "wood_color",
        "lights" ["light_inst"] )
end material
```

Wood texture map on a polygon

Modifying surface geometry

Combining displacement mapping with color





The original wood image for color and the processed version for use in displacement

Modifying surface geometry

Combining displacement mapping with color

```
color texture "wood_color_texture"
              "wood_color.tif"

color texture "wood_displace_texture"
              "wood_bump.tif"

shader "wood_color"
    "texture_uv" (
        "tex" "wood_color_texture" )

material "wood_displace"
    "lambert" (
        "diffuse" = "wood_color",
        "lights" ["light_inst"] )
    displace
        "displace_texture" (
            "texture" "wood_displace_texture",
            "factor" .02 )
end material
```

Texture maps defining both color and displacement for a polygon

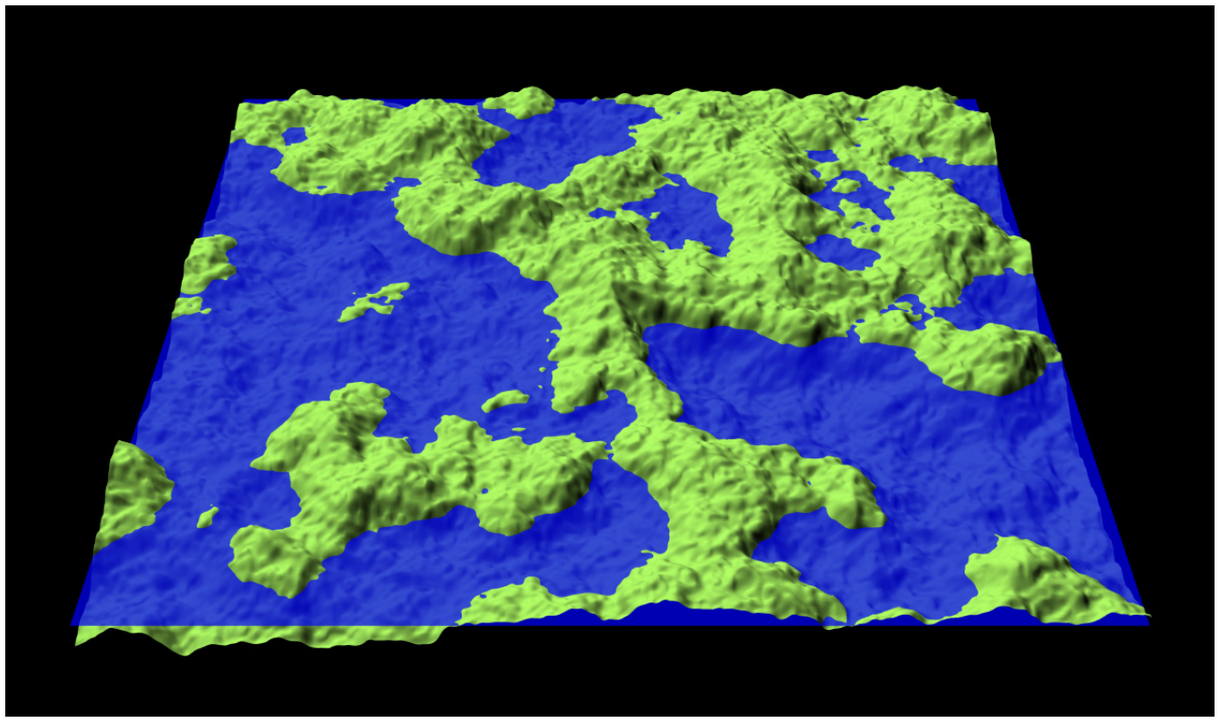
Using noise functions with displacement mapping

```
declare shader
    scalar "summed_noise_scalar" (
        scalar  "point_scale" default 1,
        scalar  "magnitude" default 1,
        scalar  "octave_scaling" default 2,
        scalar  "summing_weight" default 2,
        integer "number_of_octaves" default 5 )
end declare
```

Scene file declaration of shader "summed_noise_scalar"

Modifying surface geometry

Using noise functions with displacement mapping



```
shader "land_displacement"
    "summed_noise_scalar" (
        "point_scale" 10,
        "magnitude" .05 )

material "water"
    "transparent" (
        "color"  0 0 1,
        "transparency" .3 .3 .3 )
end material

material "land"
    "lambert" (
        "diffuse"  .7 1 .4,
        "lights" ["light_inst"] )
    displace
        = "land_displacement"
end material
```

Displacement around zero as shown by object interpenetration

Modifying surface geometry

Using noise functions with displacement mapping

```
 1  struct summed_noise_scalar {
 2      miScalar point_scale;
 3      miScalar magnitude;
 4      miScalar octave_scaling;
 5      miScalar summing_weight;
 6      miInteger number_of_octaves;
 7  };
 8
 9  miBoolean summed_noise_scalar (
10      miScalar *result, miState *state, struct summed_noise_scalar *params )
11  {
12      miVector object_point;
13      miScalar magnitude = *mi_eval_scalar(&params->magnitude), noise_sum;
14
15      mi_point_to_object(state, &object_point, &state->point);
16      mi_vector_mul(&object_point, *mi_eval_scalar(&params->point_scale));
17
18      noise_sum = miaux_summed_noise(&object_point,
19                                     *mi_eval_scalar(&params->summing_weight),
20                                     *mi_eval_scalar(&params->octave_scaling),
21                                     *mi_eval_integer(&params->number_of_octaves));
22
23      *result += miaux_fit(noise_sum, 0.0, 1.0, -magnitude, magnitude);
24
25      return miTRUE;
26  }
```

Source code of shader "summed_noise_scalar"

Exercise 17: Displacement shaders

1. Copy `displace_3.mi` to `displace.mi` and change camera output to `displace.tif`
2. Add more `displace_ripple` elements to shader list.
3. Change to use `displace_texture` with a picture from the Web — refer to `displace_5.mi`
4. Modify shader `displace_texture` to use the average of the texture channels.

Exercise 17: Displacement shaders (part 2)

Modify shader `displace_texture` to use the average of the texture channels.

Old:

```
miColor color;
mi_lookup_color_texture(&color, state,
                        *mi_eval_tag(&params->texture),
                        &state->tex_list[0]);
*result += *mi_eval_scalar(&params->factor) * color.r;
```

New:

```
miColor color;
miScalar average;
mi_lookup_color_texture(&color, state,
                        *mi_eval_tag(&params->texture),
                        &state->tex_list[0]);
average = (color.r + color.g + color.b) / 3.0;
*result += *mi_eval_scalar(&params->factor) * average;
```

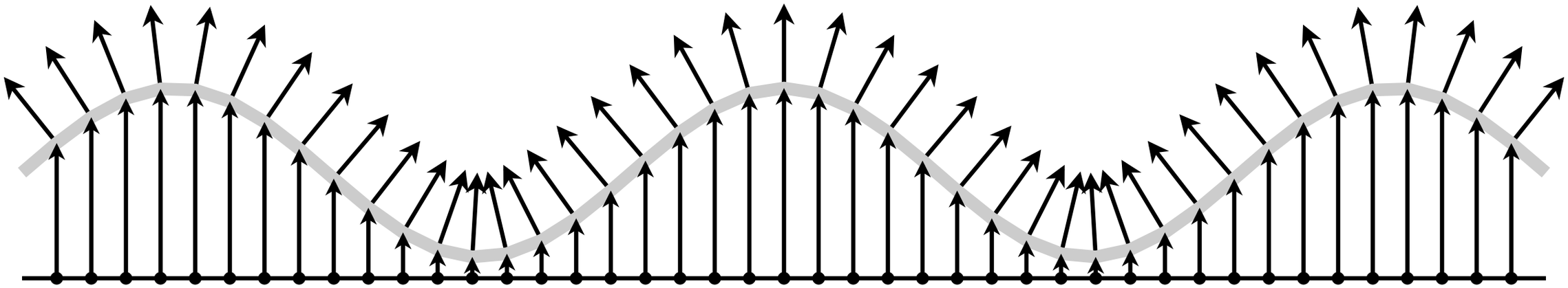
Modifying surface orientation

Modifying surface orientation

Simulating surface orientation
Modifying the normal in a shader
Changing the normal based on texture coordinates
Evaluating a function in the sample neighborhood
Shader lists and bump mapping
Using images to control bump mapping
Combining bump mapping with color

Modifying surface orientation

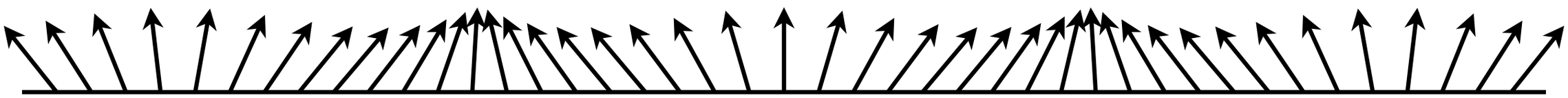
Simulating surface orientation



Associating points on a surface with a surface geometry to be simulated

Modifying surface orientation

Simulating surface orientation



Changing the normals of a flat surface to simulate geometric complexity

Modifying surface orientation

Modifying the normal in a shader

```
material "lambert"
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
end material
```

Simple scene for bump mapping shader examples

Modifying surface orientation

Modifying the normal in a shader

```
material "wave"
    "bump_wave" (
        "frequency" 3,
        "amplitude" .4 )
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
end material
```

Modifying the apparent surface orientation by changing the x component of its normal

Modifying surface orientation

Modifying the normal in a shader

```
1  double miaux_sinusoid(double v, double frequency, double amplitude)
2  {
3      return sin(v * frequency * M_PI * 2.0) * amplitude;
4  }
```

Auxiliary function: miaux_sinusoid

Modifying surface orientation

Modifying the normal in a shader

```
declare shader
    color "bump_wave" (
        scalar "frequency" default 1,
        scalar "amplitude" default .5 )
end declare
```

Scene file declaration of shader "bump_wave"

Modifying surface orientation

Modifying the normal in a shader

```
 1  struct bump_wave {
 2      miScalar frequency;
 3      miScalar amplitude;
 4  };
 5
 6  miBoolean bump_wave (
 7      miColor *result, miState *state, struct bump_wave *params )
 8  {
 9      state->normal.x += miaux_sinusoid(state->point.x,
10                                        *mi_eval_scalar(&params->frequency),
11                                        *mi_eval_scalar(&params->amplitude));
12      mi_vector_normalize(&state->normal);
13      return miTRUE;
14  }
```

Source code of shader "bump_wave"

Modifying surface orientation

Changing the normal based on texture coordinates

```
declare shader
    color "bump_wave_uv" (
        scalar "frequency_u" default 1,
        scalar "amplitude_u" default .5,
        scalar "frequency_v" default 1,
        scalar "amplitude_v" default .5 )
end declare
```

Scene file declaration of shader "bump_wave_uv"

Modifying surface orientation

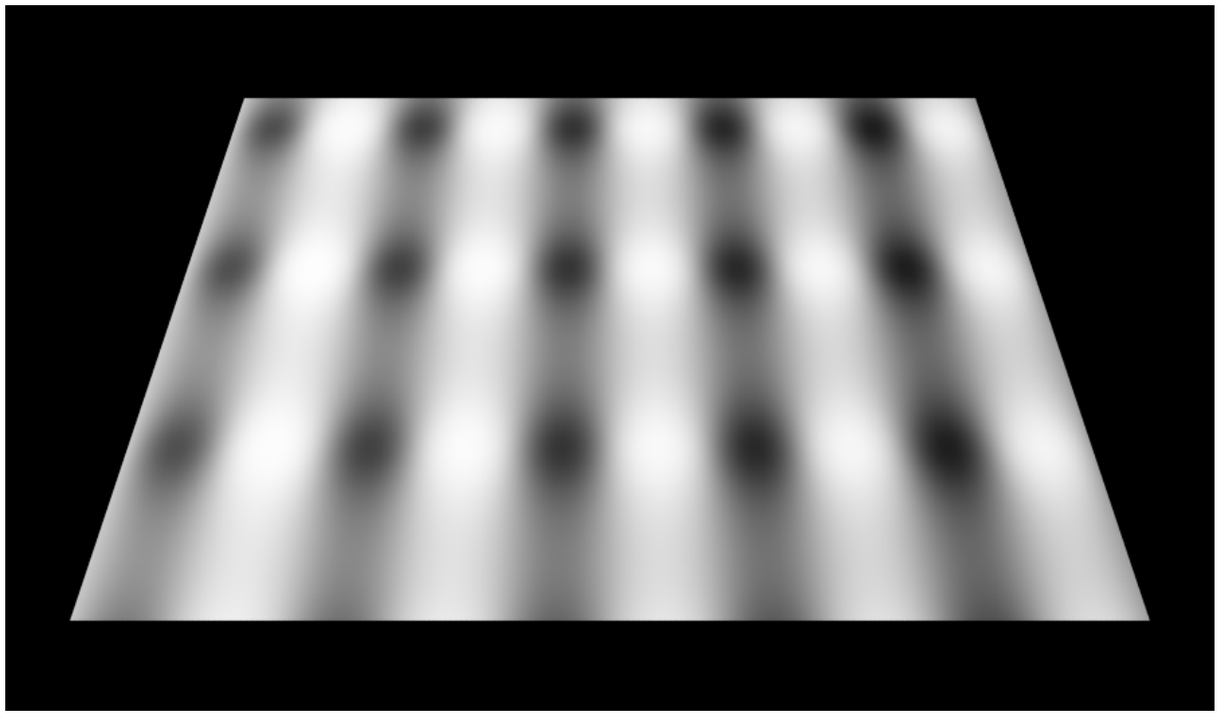
Changing the normal based on texture coordinates

```
 1  struct bump_wave_uv {
 2      miScalar frequency_u;
 3      miScalar amplitude_u;
 4      miScalar frequency_v;
 5      miScalar amplitude_v;
 6  };
 7
 8  miBoolean bump_wave_uv (
 9      miColor *result, miState *state, struct bump_wave_uv *params )
10  {
11      state->normal.x += miaux_sinusoid(state->tex_list[0].x,
12                                        *mi_eval_scalar(&params->frequency_u),
13                                        *mi_eval_scalar(&params->amplitude_u));
14      state->normal.y += miaux_sinusoid(state->tex_list[0].y,
15                                        *mi_eval_scalar(&params->frequency_v),
16                                        *mi_eval_scalar(&params->amplitude_v));
17      mi_vector_normalize(&state->normal);
18      return miTRUE;
19  }
```

Source code of shader "bump_wave_uv"

Modifying surface orientation

Changing the normal based on texture coordinates

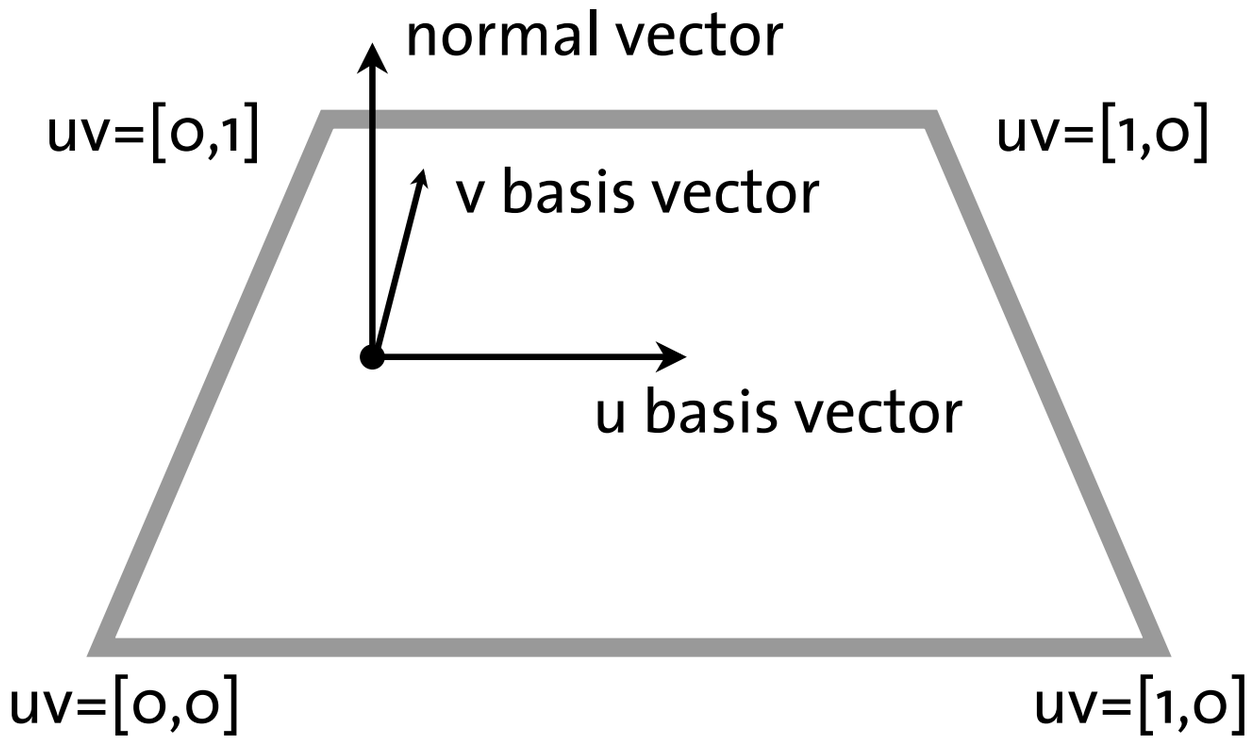


```
material "wave_uv"
    "bump_wave_uv" (
        "frequency_u" 5,
        "amplitude_u" .4,
        "frequency_v" 3,
        "amplitude_v" .4, )
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
end material
```

Modifying both the x and y components of the surface normal

Modifying surface orientation

Evaluating a function in the sample neighborhood



The normal vector and the two bump basis vectors

Modifying surface orientation

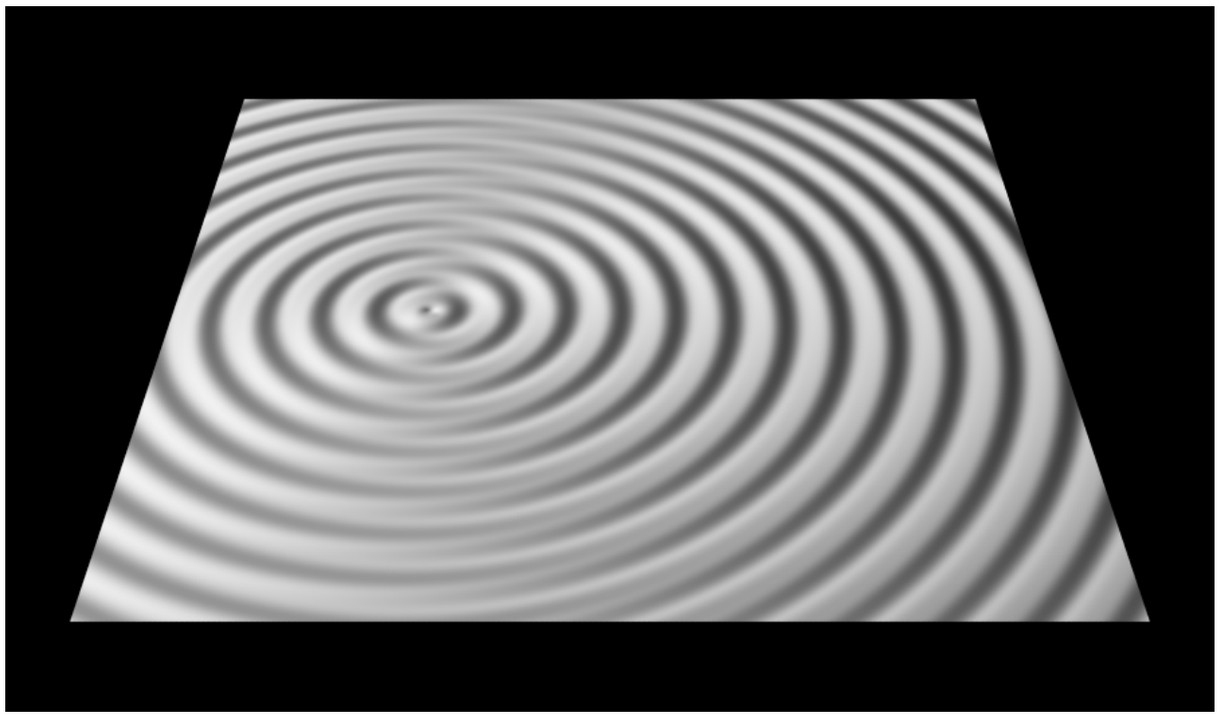
Evaluating a function in the sample neighborhood

```
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"

Modifying surface orientation

Evaluating a function in the sample neighborhood



```
material "ripple"
    "bump_ripple" (
        "center" .3 .5 0,
        "frequency" 16,
        "amplitude" .5, )
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
end material
```

Modifying the normal based on the distance from a point as input to a sinusoidal function

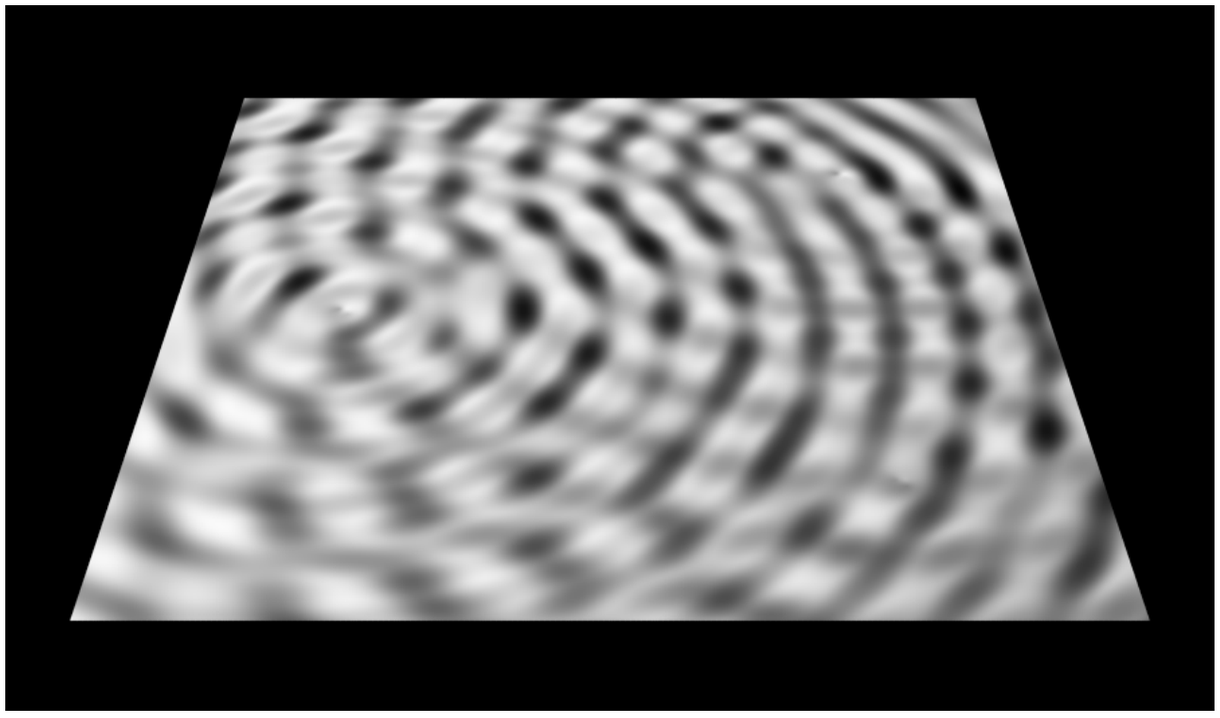
Modifying surface orientation

Evaluating a function in the sample neighborhood

```
 1  struct bump_ripple {
 2      miVector center;
 3      miScalar frequency;
 4      miScalar amplitude;
 5      miScalar nearby;
 6  };
 7
 8  miBoolean bump_ripple (
 9      miColor *result, miState *state, struct bump_ripple *params )
10  {
11      double ripple_here, ripple_over, ripple_up,
12          change_going_over, change_going_up;
13      miVector here, over, up;
14
15      miVector center = *mi_eval_vector(&params->center);
16      miScalar frequency = *mi_eval_scalar(&params->frequency);
17      miScalar amplitude = *mi_eval_scalar(&params->amplitude);
18      miScalar nearby    = *mi_eval_scalar(&params->nearby);
19
20      miVector bump_basis_u = state->bump_x_list[0];
21      miVector bump_basis_v = state->bump_y_list[0];
22
23      here = state->tex_list[0];
24      miaux_set_vector(&over, here.x + nearby, here.y, here.z);
25      miaux_set_vector(&up, here.x, here.y + nearby, here.z);
26
27      ripple_here = miaux_sinusoid(mi_vector_dist(&center, &here),
28                                   frequency, amplitude);
29      ripple_over =  miaux_sinusoid(mi_vector_dist(&center, &over),
30                                    frequency, amplitude);
31      ripple_up = miaux_sinusoid(mi_vector_dist(&center, &up),
32                                 frequency, amplitude);
33
34      change_going_over = ripple_over - ripple_here;
35      change_going_up = ripple_over - ripple_up;
36
37      mi_vector_mul(&bump_basis_u, -change_going_over);
38      mi_vector_mul(&bump_basis_v, -change_going_up);
39
40      mi_vector_to_object(state, &state->normal, &state->normal);
41      mi_vector_add(&state->normal, &state->normal, &bump_basis_u);
42      mi_vector_add(&state->normal, &state->normal, &bump_basis_v);
43      mi_vector_normalize(&state->normal);
44      mi_vector_from_object(state, &state->normal, &state->normal);
45
46      return miTRUE;
47  }
```

Source code of shader "bump_ripple"

Modifying surface orientation



```
material "three_ripples"
    "bump_ripple" (
        "center" .2 .5 0,
        "frequency" 12 )
    "bump_ripple" (
        "center" .8 .8 0,
        "frequency" 10 )
    "bump_ripple" (
        "center" .8 .2 0,
        "frequency" 8 )
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
end material
```

Three modifications to the surface normal in a shader list

Modifying surface orientation

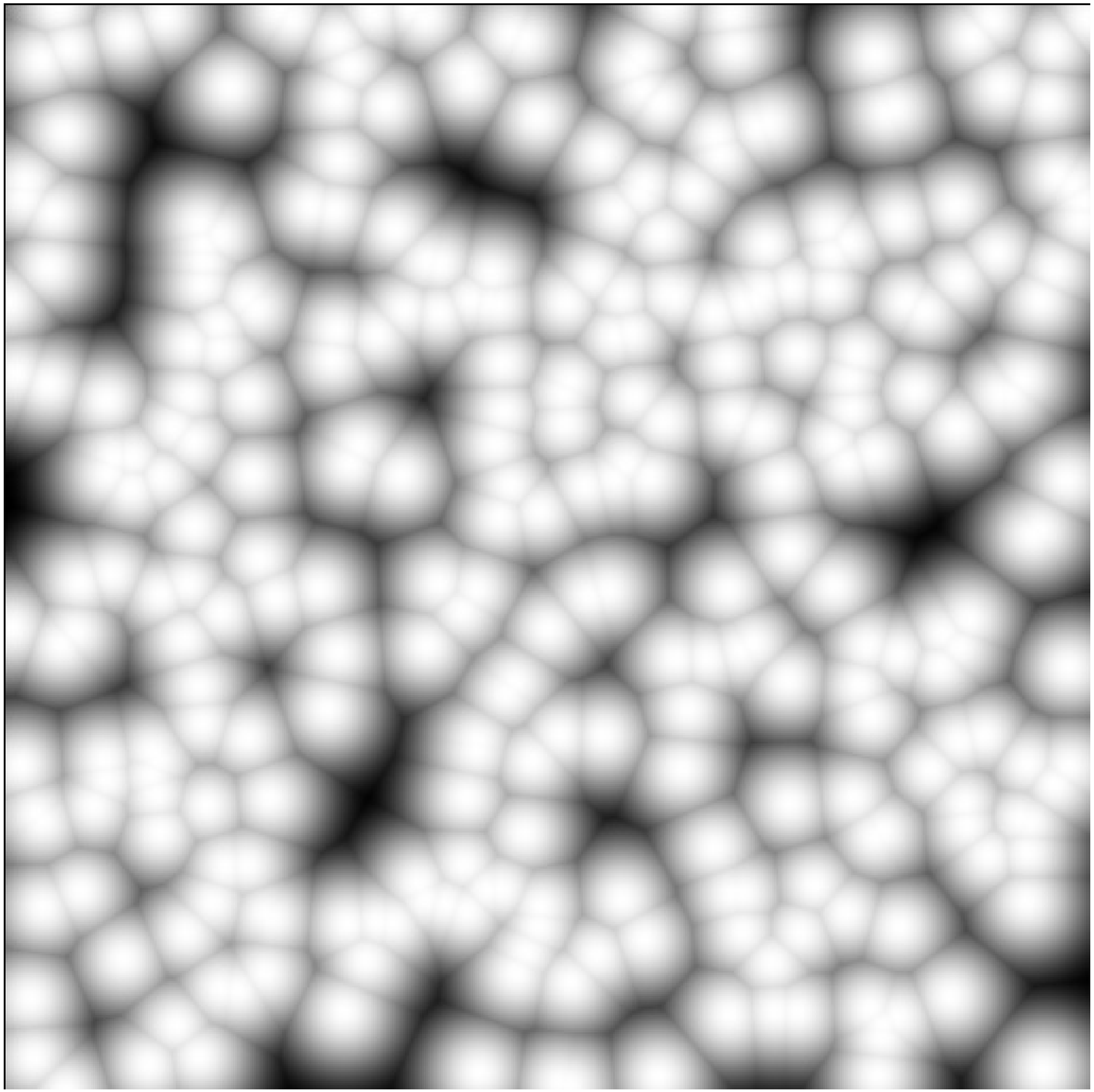
Using images to control bump mapping

```
declare shader
    color "bump_texture" (
        color texture "texture",
        scalar "factor" default 1,
        scalar "nearby" default .001 )
end declare
```

Scene file declaration of shader "bump_texture"

Modifying surface orientation

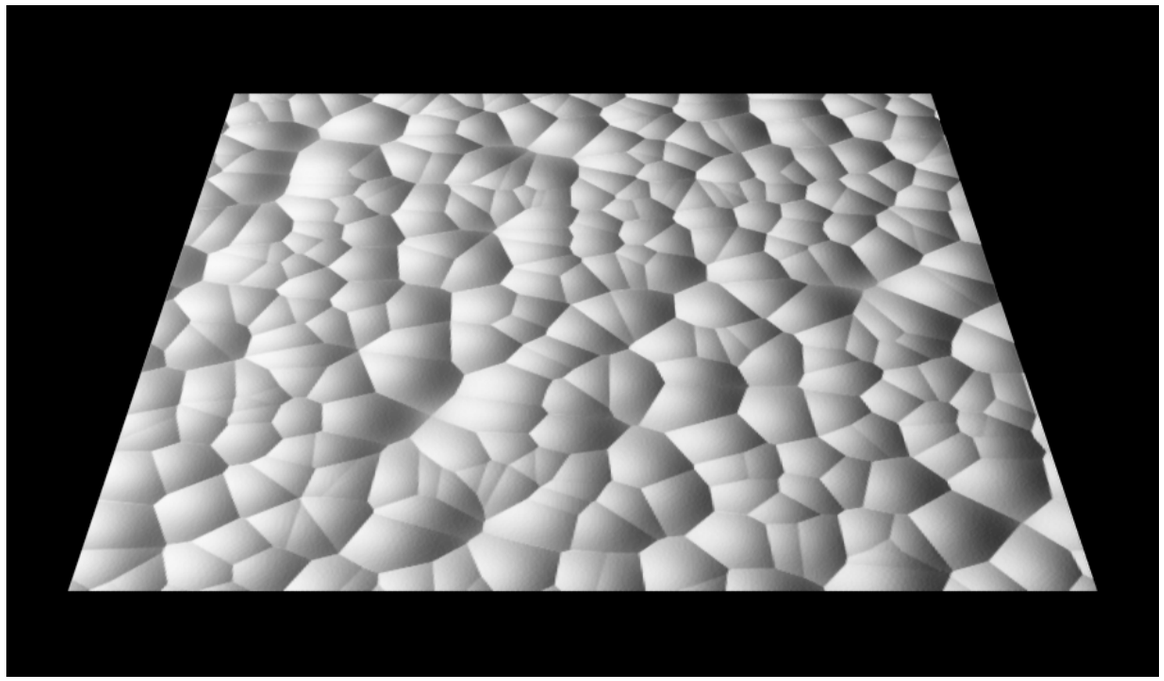
Using images to control bump mapping



An image for use as a bump map

Modifying surface orientation

Using images to control bump mapping



```
color texture "bubbles" "bubbles.tif"

material "bubble_texture"
    "bump_texture" (
        "texture" "bubbles",
        "factor" 30 )
    "lambert" (
        "diffuse" 1 1 1,
        "lights" ["light_inst"] )
end material
```

Using the color values of a texture map to define surface orientation

Modifying surface orientation

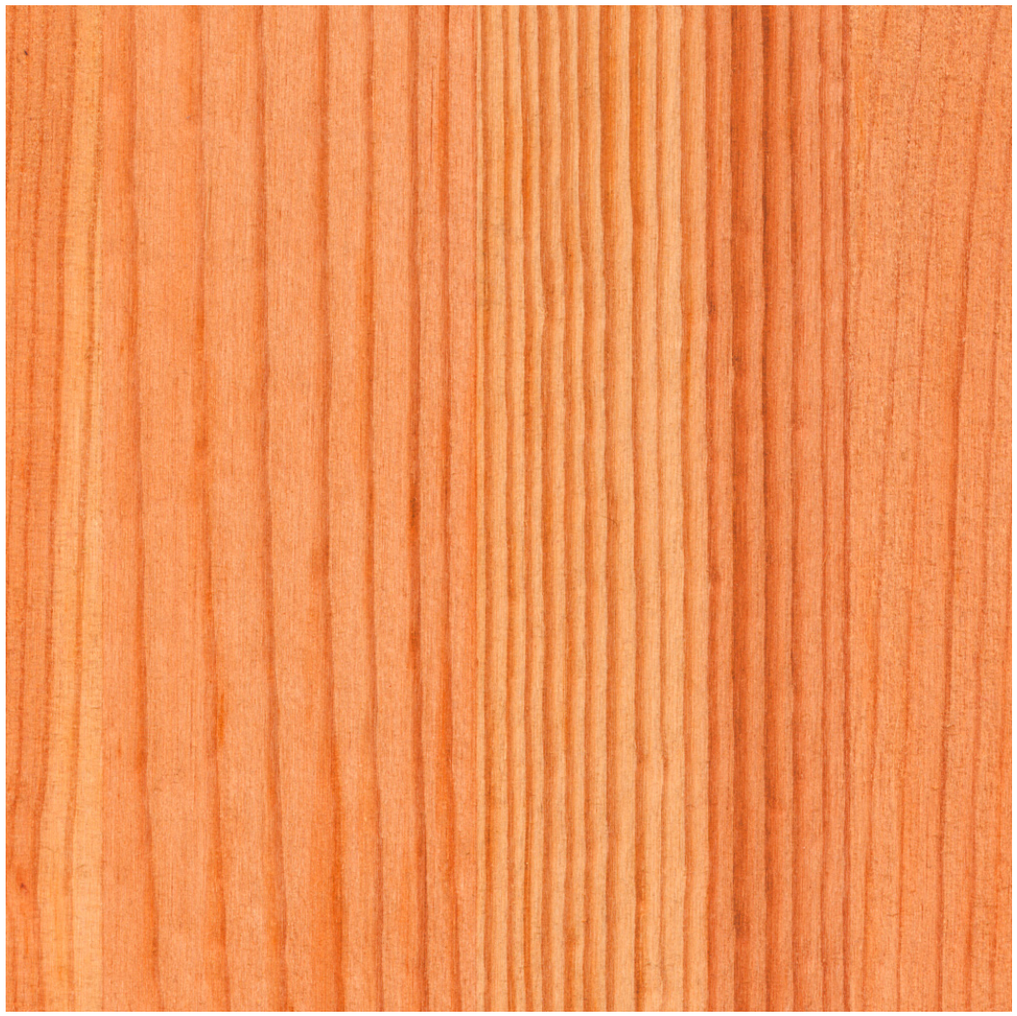
Using images to control bump mapping

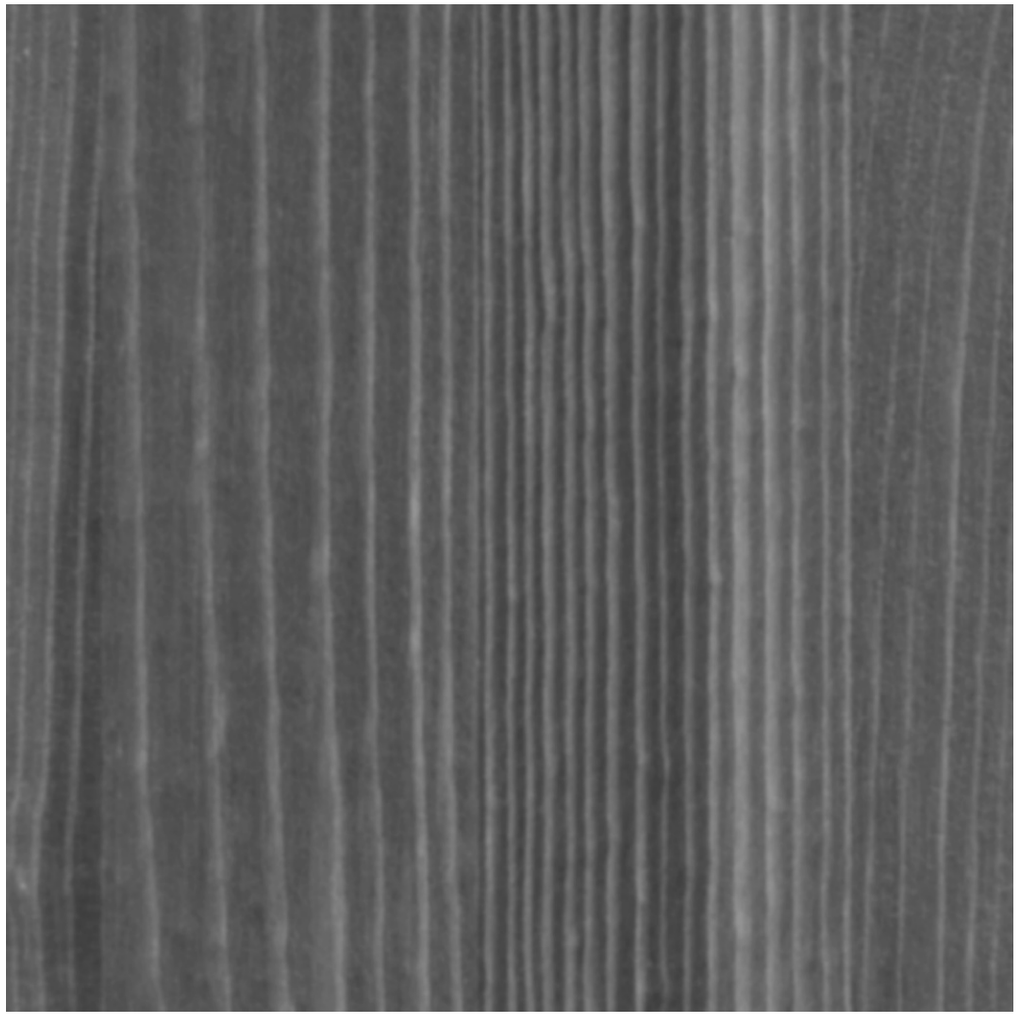
```
 1  struct bump_texture {
 2      miTag texture;
 3      miScalar factor;
 4      miScalar nearby;
 5  };
 6
 7  miBoolean bump_texture (
 8      miColor *result, miState *state, struct bump_texture *params )
 9  {
10      miColor color_here, color_over, color_up;
11      miScalar value_here, value_over, value_up,
12          change_going_over, change_going_up;
13      miVector here, over, up;
14
15      miTag texture = *mi_eval_tag(&params->texture);
16      miScalar factor = *mi_eval_scalar(&params->factor);
17      miScalar nearby = *mi_eval_scalar(&params->nearby);
18
19      miVector bump_basis_u = state->bump_x_list[0];
20      miVector bump_basis_v = state->bump_y_list[0];
21
22      here = state->tex_list[0];
23      miaux_set_vector(&over, here.x + nearby, here.y, here.z);
24      miaux_set_vector(&up, here.x, here.y + nearby, here.z);
25
26      if (!mi_lookup_color_texture(&color_here, state, texture, &here))
27          return miFALSE;
28      value_here = miaux_color_channel_average(&color_here);
29
30      mi_flush_cache(state);
31      value_over = mi_lookup_color_texture(&color_over, state, texture, &over) ?
32          miaux_color_channel_average(&color_over) : value_here;
33
34      mi_flush_cache(state);
35      value_up = mi_lookup_color_texture(&color_up, state, texture, &up) ?
36          miaux_color_channel_average(&color_up) : value_here;
37
38      change_going_over = factor * (value_over - value_here);
39      change_going_up = factor * (value_up - value_here);
40
41      mi_vector_mul(&bump_basis_u, -change_going_over);
42      mi_vector_mul(&bump_basis_v, -change_going_up);
43
44      mi_vector_to_object(state, &state->normal, &state->normal);
45      mi_vector_add(&state->normal, &state->normal, &bump_basis_u);
46      mi_vector_add(&state->normal, &state->normal, &bump_basis_v);
47      mi_vector_normalize(&state->normal);
48      mi_vector_from_object(state, &state->normal, &state->normal);
49
50      return miTRUE;
51  }
```

Source code of shader "bump_texture"

Modifying surface orientation

Combining bump mapping with color





The original wood image for color mapping and its negative for bump mapping

Modifying surface orientation

Combining bump mapping with color

```
color texture "wood_color"
    "wood_texture.tif"

material "wood_bump_texture"
    "bump_texture" (
        "texture" "wood_bump",
        "factor" 10 )
    "blinn" (
        "specular" .3 .3 .3,
        "diffuse" = "wood_shader",
        "lights" ["light_inst"] )
end material
```

Using a single image as the source for both color and bump-mapping information

Creating geometric objects

Creating geometric objects

- Creating a square polygon
- Procedural use of the geometry API
- Placeholder objects
    - Creating an object from a file specified in the scene
    - Creating an object from a file within a shader
    - Creating an object instance from file within a shader

Creating geometric objects

Creating a square polygon

```
# 1. Begin the object definition.
object "square-object"

    # 2. Set the object flags.
    visible on
    shadow 3

    # 3. Begin the object group definition.
    group
        # 4. Define the vectors to be used for vertices.
        -.5 -.5  0
         .5 -.5  0
         .5  .5  0
        -.5  .5  0

        # 5. Define the vectors to be used for normals.
        0 0 1

        # 6. Define the vectors to be used for UV coordinates.
        0 0 0
        1 0 0
        1 1 0
        0 1 0

        # 7. Specify the sets of vectors to be used for the position,
        #    normal and texture coordinates of each vertex.
        v 0 n 4 t 5
        v 1 n 4 t 6
        v 2 n 4 t 7
        v 3 n 4 t 8

        # 8. Specify the vertices to be used for each polygon.
        p 0 1 2 3

    # 9. End the object group definition.
    end group

    # 10. End the object definition.
end object
```

An object defined by a .mi file

Creating geometric objects

Creating a square polygon

```
object : OBJECT opt_symbol
             { curr_obj = mi_api_object_begin($2); }
         obj_flags
         object_body
         END OBJECT
             { mi_api_object_end(); }
       ;
```

Rule in the yacc grammar for the object element

Creating geometric objects

Creating a square polygon

```
 1  miBoolean square (
 2      miTag *result, miState *state, void *params)
 3  {
 4      int i;
 5      miVector v;
 6      miTag object_tag;
 7
 8      /* 1. Begin the object definition. */
 9      miObject *obj = mi_api_object_begin(mi_mem_strdup("::square"));
10
11      /* 2. Set the object flags. */
12      obj->visible = miTRUE;
13      obj->shadow = 3;
14
15      /* 3. Begin the object group definition. */
16      mi_api_object_group_begin(0.0);
17
18      /* 4. Define the vectors to be used for vertices. */
19      v.x = -.5; v.y = -.5; v.z = 0; mi_api_vector_xyz_add(&v);
20      v.x = .5;  mi_api_vector_xyz_add(&v);
21      v.y = .5;  mi_api_vector_xyz_add(&v);
22      v.x = -.5; mi_api_vector_xyz_add(&v);
23
24      /* 5. Define the vectors to be used for normals. */
25      v.x = 0; v.y = 0; v.z = 1; mi_api_vector_xyz_add(&v);
26
27      /* 6. Define the vectors to be used for texture coordinates. */
28      v.x = 0; v.y = 0; v.z = 0; mi_api_vector_xyz_add(&v);
29      v.x = 1; mi_api_vector_xyz_add(&v);
30      v.y = 1; mi_api_vector_xyz_add(&v);
31      v.x = 0; mi_api_vector_xyz_add(&v);
32
33      /* 7. Specify the sets of vectors to be used for the position,
34            normal and coordinates of each vertex. */
35      for (i = 0; i < 4; i++) {
36          mi_api_vertex_add(i);
37          mi_api_vertex_normal_add(4);
38          mi_api_vertex_tex_add(i + 5, -1, -1);
39      }
40
41      /* 8. Specify the vertices to be used for each polygon. */
42      mi_api_poly_begin_tag(1, 0);
43      for (i = 0; i < 4; i++)
44          mi_api_poly_index_add(i);
45      mi_api_poly_end();
46
47      /* 9. End the object group definition. */
48      mi_api_object_group_end();
49
50      /* 10. End the object definition. */
51      object_tag = mi_api_object_end();
52
53      /* 11. Add the resulting object to the database. */
54      return mi_geoshader_add_result(result, object_tag);
55  }
```

Source code of shader "square"

Creating geometric objects

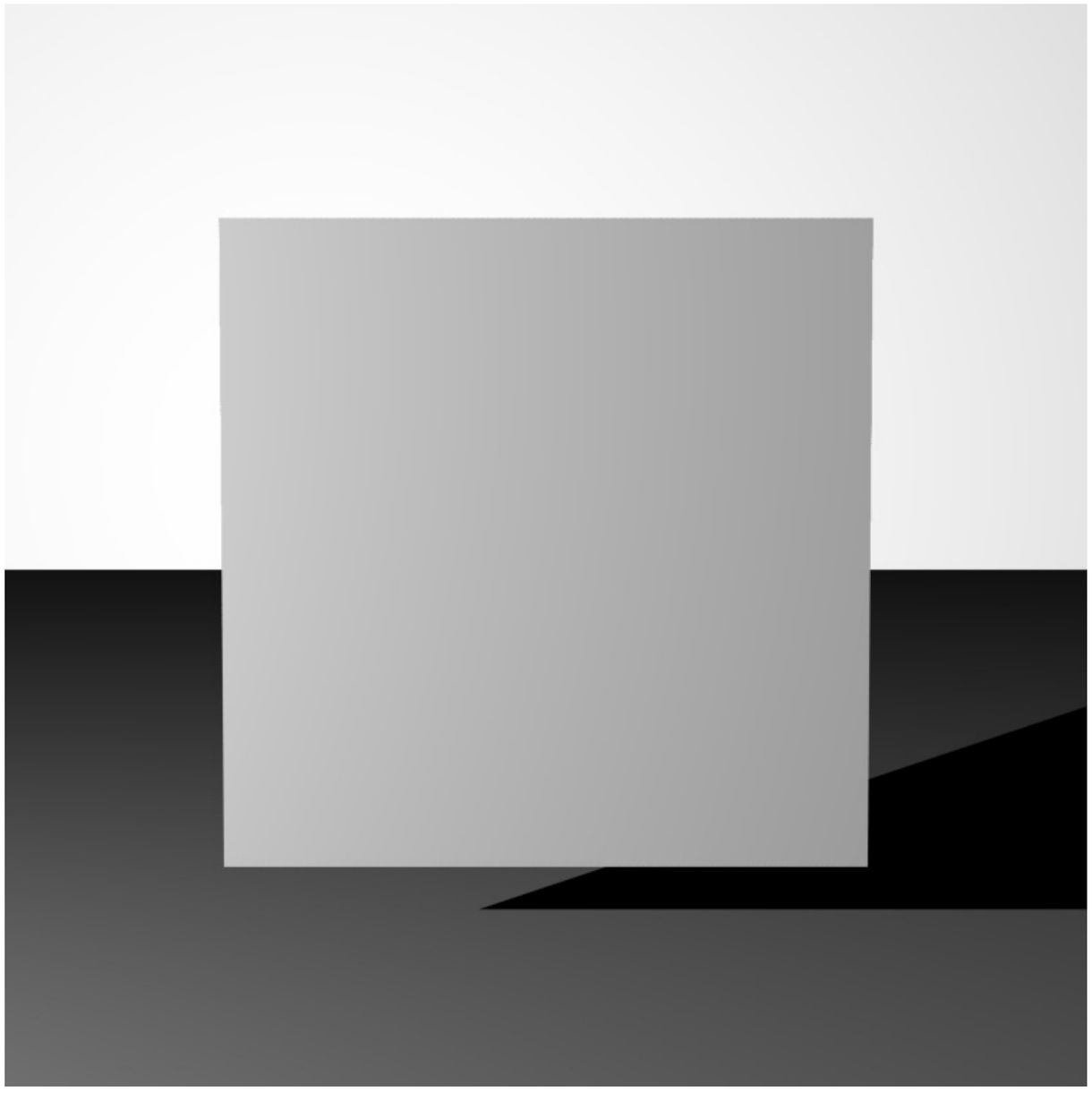
Creating a square polygon

```
declare shader
    geometry "square" ()
end declare
```

Scene file declaration of shader "square"

Creating geometric objects

Creating a square polygon

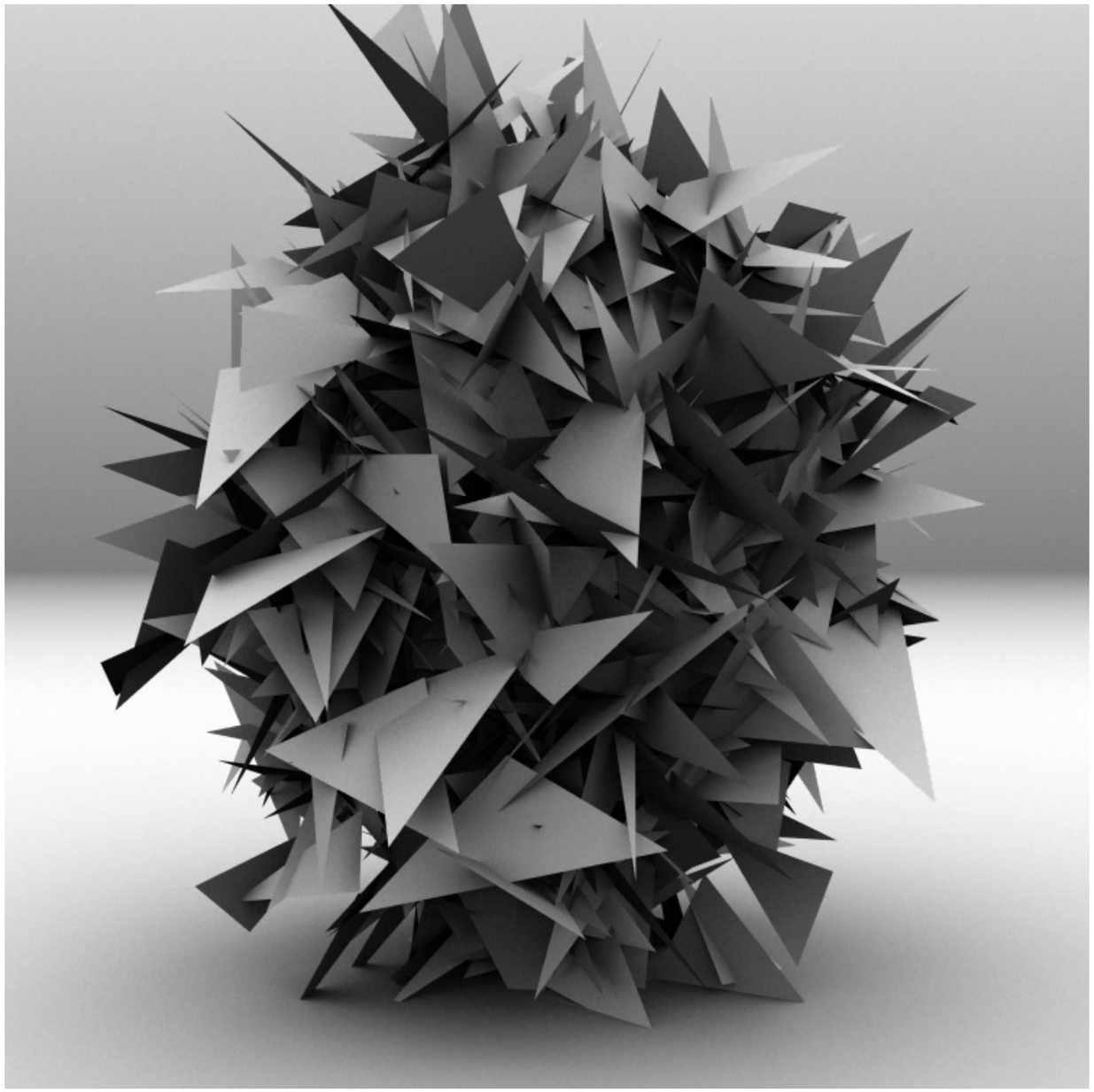


```
material "diffuse"
    "lambert" (
        "lights" ["light-inst"] )
end material

instance "square-instance"
    geometry "square" ()
    material "diffuse"
    transform
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 -.2 0 1
end instance
```

Geometry shader square used in object instance

Creating geometric objects

Procedural use of the geometry API



```
material "amb_occlude"
    "ambient_occlusion_cutoff" (
        "samples" 200,
        "cutoff_distance" 9 )
end material

instance "triangles-instance"
    geometry "triangles" (
        "count" 1000,
        "bbox_min" -.5 -.5 -.5,
        "bbox_max" .5 .9 .5,
        "edge_length_max" .5 )
    material "amb_occlude"
    transform
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 0 .2 1
end instance
```

Procedural geometric construction rendered with ambient occlusion

```
1  float miaux_random_range(float min_value, float max_value)
2  {
3      return miaux_fit(mi_random(), 0.0, 1.0, min_value, max_value);
4  }
```

Auxiliary function: miaux_random_range

```
1  void miaux_random_point_in_unit_sphere(float *x, float *y, float *z)
2  {
3      do {
4          *x = miaux_random_range(-.5, .5);
5          *y = miaux_random_range(-.5, .5);
6          *z = miaux_random_range(-.5, .5);
7      } while (sqrt((*x * *x) +(*y * *y) + (*z * *z)) >= 0.5);
8  }
```

Auxiliary function: miaux_random_point_in_unit_sphere

```
1  void miaux_add_vertex(int index, miScalar x, miScalar y, miScalar z)
2  {
3      miVector v;
4      v.x = x; v.y = y; v.z = z;
5      mi_api_vector_xyz_add(&v);
6      mi_api_vertex_add(index);
7  }
```

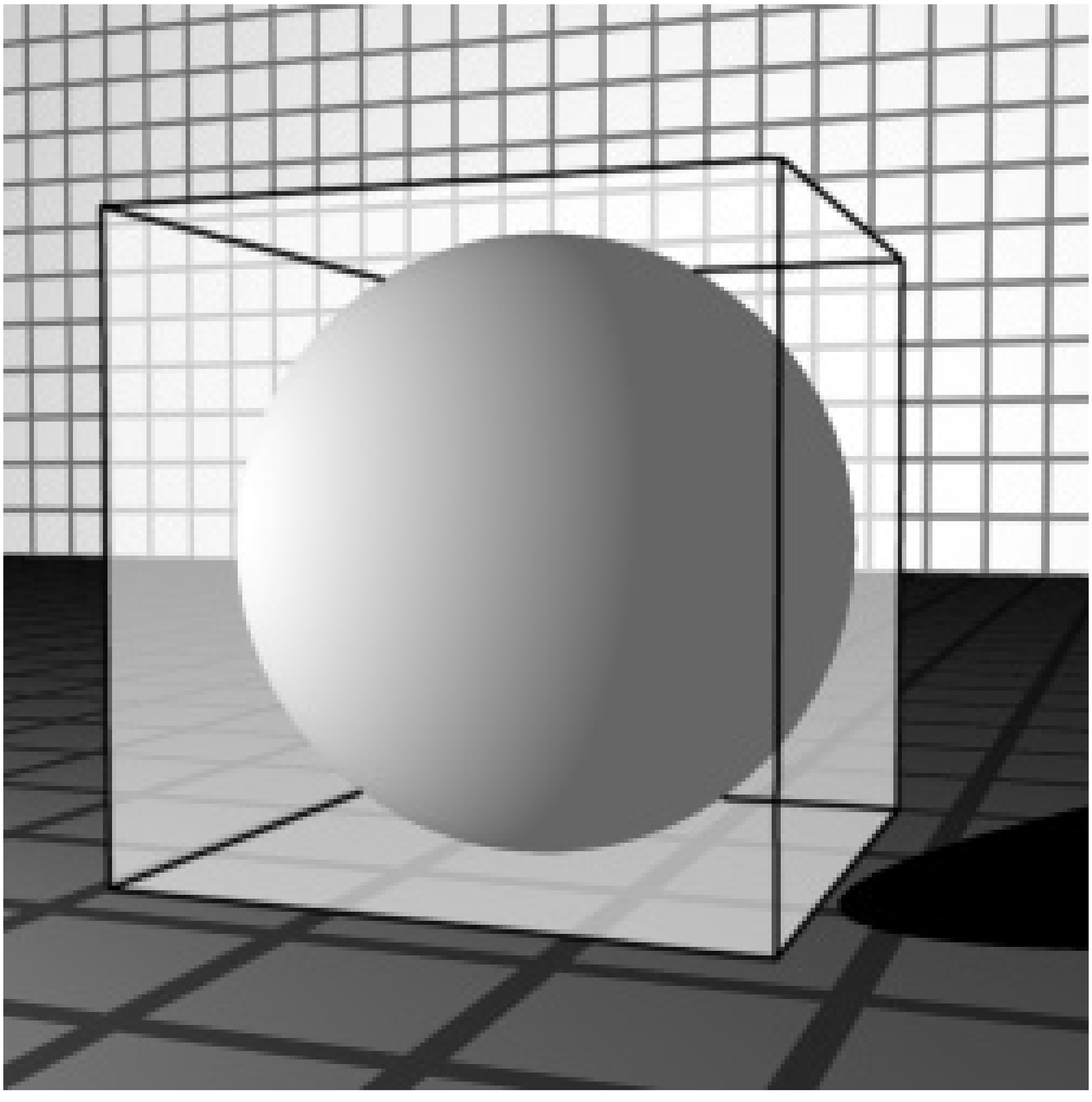
Auxiliary function: miaux_add_vertex

Creating geometric objects

Procedural use of the geometry API

```
 1  void miaux_add_random_triangle(int index, float edge_length_max,
 2                                 miVector *bbox_min, miVector *bbox_max)
 3  {
 4      int vertex_index;
 5      float offset_max = edge_length_max / 2.0;
 6      float offset_min = -offset_max;
 7      float x, y, z;
 8
 9      miaux_random_point_in_unit_sphere(&x, &y, &z);
10      x = miaux_fit(x, -.5, .5, bbox_min->x, bbox_max->x);
11      y = miaux_fit(y, -.5, .5, bbox_min->y, bbox_max->y);
12      z = miaux_fit(z, -.5, .5, bbox_min->z, bbox_max->z);
13
14      miaux_add_vertex(index, x, y, z);
15      miaux_add_vertex(index + 1,
16                       x + miaux_random_range(offset_min, offset_max),
17                       y + miaux_random_range(offset_min, offset_max),
18                       z + miaux_random_range(offset_min, offset_max));
19      miaux_add_vertex(index + 2,
20                       x + miaux_random_range(offset_min, offset_max),
21                       y + miaux_random_range(offset_min, offset_max),
22                       z + miaux_random_range(offset_min, offset_max));
23
24      mi_api_poly_begin_tag(1, 0);
25      for (vertex_index = 0; vertex_index < 3; vertex_index++)
26          mi_api_poly_index_add(index + vertex_index);
27      mi_api_poly_end();
28  }
```

Auxiliary function: miaux_add_random_triangle



A bounding box enclosing a sphere

Creating geometric objects

Procedural use of the geometry API

```
declare shader
    geometry "triangles" (
        string "name",
        integer "count" default 100,
        vector "bbox_min" default -.5 -.5 -.5,
        vector "bbox_max" default .5 .5 .5,
        scalar "edge_length_max" default 1,
        integer "random_seed" default 1955 )
end declare
```

Scene file declaration of shader "triangles"

Creating geometric objects

Procedural use of the geometry API

```
 1  struct triangles {
 2      miTag name;
 3      miInteger count;
 4      miVector bbox_min;
 5      miVector bbox_max;
 6      miScalar edge_length_max;
 7      miInteger random_seed;
 8  };
 9
10  miBoolean triangles (
11      miTag *result, miState *state, struct triangles *params )
12  {
13      int vertex_index;
14      miObject *obj;
15      miInteger count = *mi_eval_integer(&params->count);
16      miVector *bbox_min = mi_eval_vector(&params->bbox_min);
17      miVector *bbox_max = mi_eval_vector(&params->bbox_max);
18      miScalar edge_length_max = *mi_eval_scalar(&params->edge_length_max);
19      char* name =
20          miaux_tag_to_string(*mi_eval_tag(&params->name), "::triangles");
21
22      mi_srandom(*mi_eval_integer(&params->random_seed));
23
24      obj = mi_api_object_begin(mi_mem_strdup(name));
25      obj->visible = miTRUE;
26      obj->shadow = 3;
27
28      mi_api_object_group_begin(0.0);
29      for (vertex_index = 0; vertex_index < count * 3; vertex_index += 3)
30          miaux_add_random_triangle(
31              vertex_index, edge_length_max, bbox_min, bbox_max);
32      mi_api_object_group_end();
33
34      return mi_geoshader_add_result(result, mi_api_object_end());
35  }
```

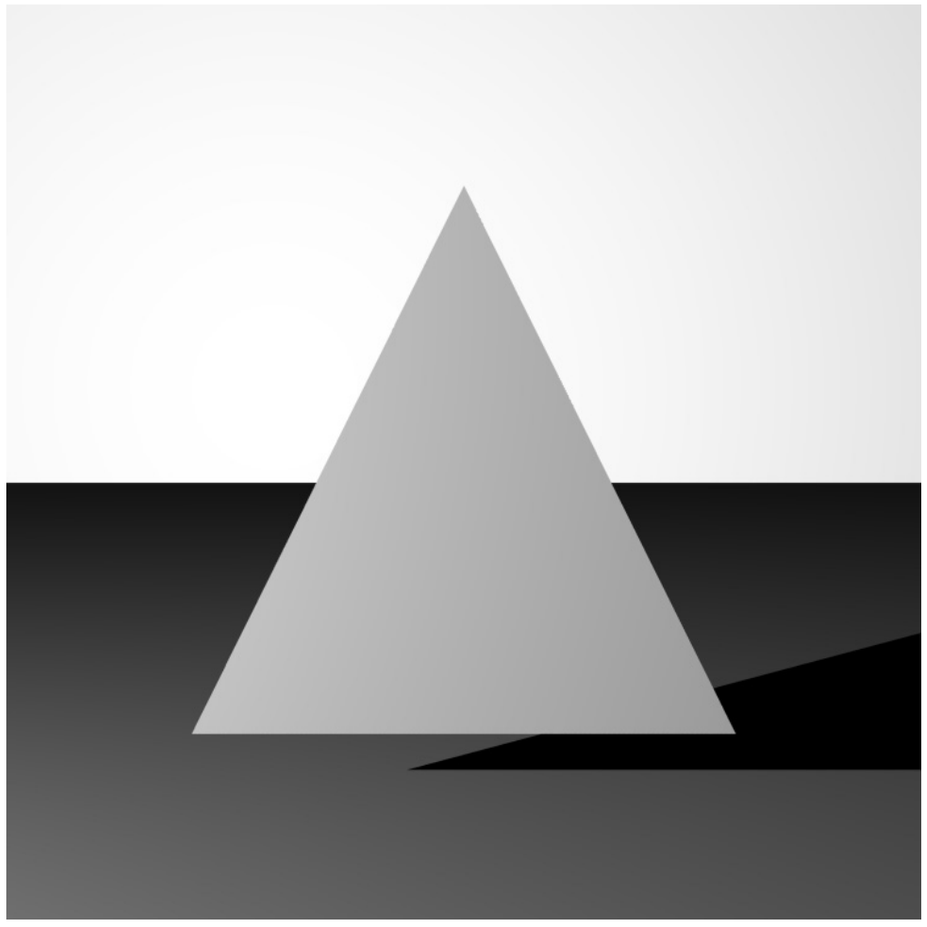
Source code of shader "triangles"

```
incremental
object "triangle-object"
    visible on
    shadow 3
    group
        -.5 -.5 0
        .5 -.5 0
        0 .5 0
        0 0 1
        0 0 0
        1 0 0
        0 1 0
        v 0 n 3 t 4
        v 1 n 3 t 5
        v 2 n 3 t 6
        p 0 1 2
    end group
end object
```

Object file triangle_object.mi with the incremental command

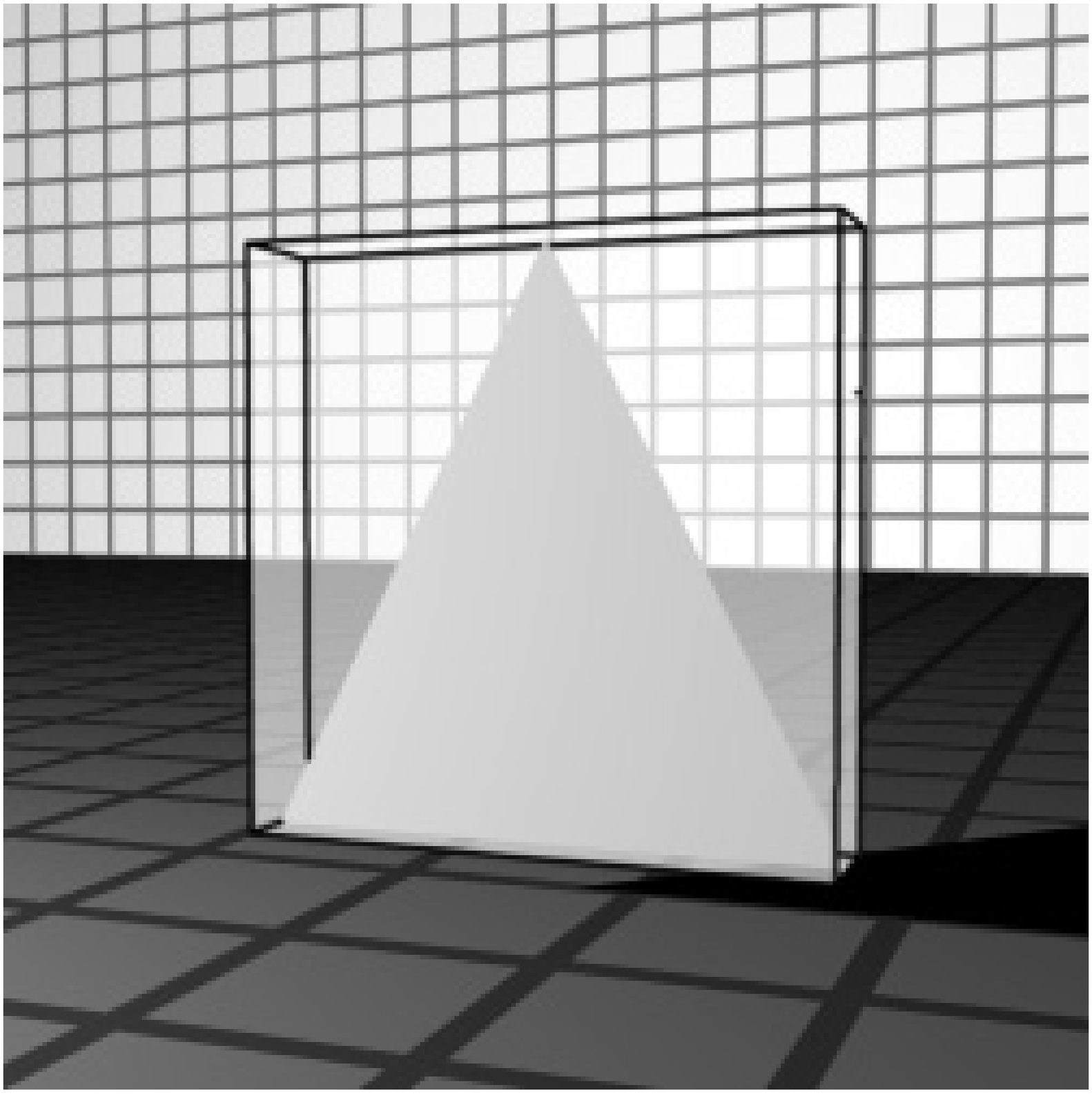
Creating geometric objects

Placeholder objects



```
material "diffuse"
    "lambert" (
        "lights" ["light-inst"] )
end material

object "triangle-object"
    visible
    shadow 3
    box -.5 -.5 -.1 .5 .5 .1
    file "triangle_object.mi"
end object

instance "triangle-instance" "triangle-object"
    material "diffuse"
    transform
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 -.2 0 1
end instance
```

Object data read from file using the file statement in the object block



A bounding box that encloses a polygon with a greater thickness than necessary

```
1  miTag miaux_object_from_file(
2      const char* name, const char* filename,
3      miVector bbox_min, miVector bbox_max)
4  {
5      miObject *obj = mi_api_object_begin(mi_mem_strdup(name));
6      obj->visible = miTRUE;
7      obj->shadow = 3;
8      obj->bbox_min = bbox_min;
9      obj->bbox_max = bbox_max;
10     mi_api_object_file(mi_mem_strdup(filename));
11     return mi_api_object_end();
12 }
```

Auxiliary function: miaux_object_from_file

```
declare shader
    geometry "object_file" (
        string "name",
        string "filename",
        vector "bbox_min" default -1 -1 -1,
        vector "bbox_max" default 1 1 1 )
end declare
```

Scene file declaration of shader "object_file"

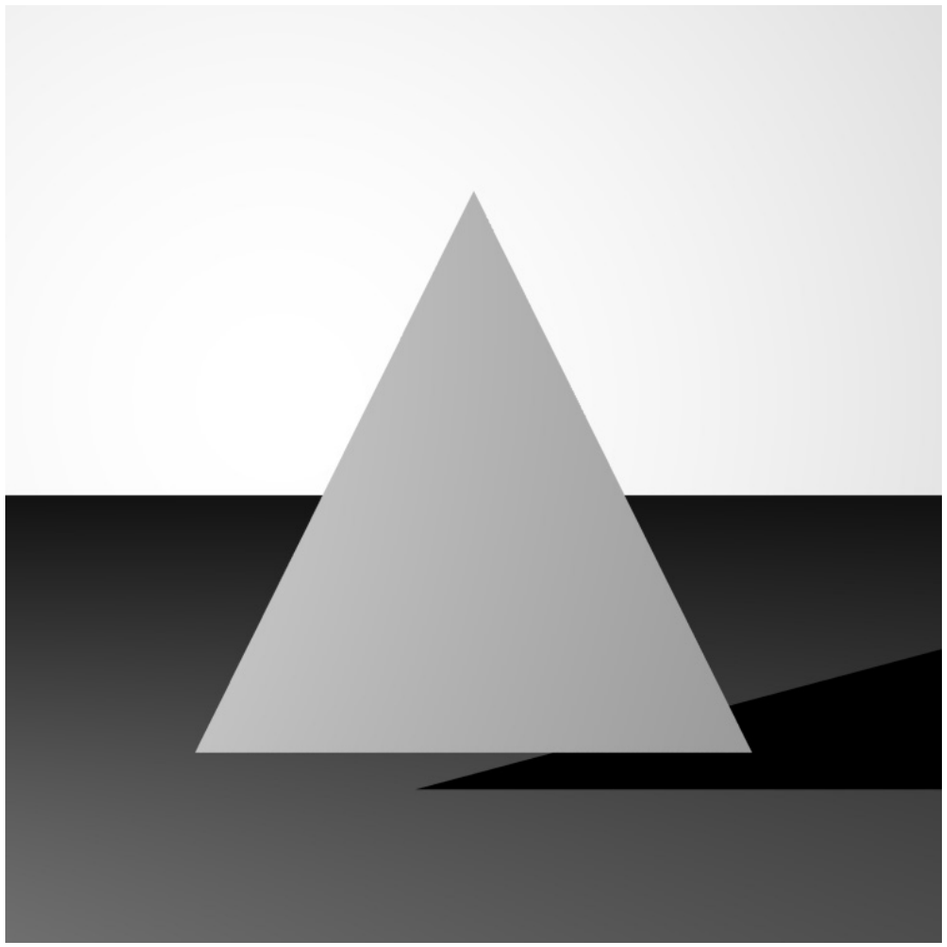
Creating geometric objects


```
 1  struct object_file {
 2      miTag name;
 3      miTag filename;
 4      miVector bbox_min;
 5      miVector bbox_max;
 6  };
 7
 8  miBoolean object_file (
 9      miTag *result, miState *state, struct object_file *params )
10  {
11      char *name, *filename;
12
13      if (!(name = miaux_tag_to_string(*mi_eval_tag(&params->name), NULL)) ||
14          !(filename = miaux_tag_to_string(*mi_eval_tag(&params->filename), NULL)))
15          return miFALSE;
16
17      return mi_geoshader_add_result(
18          result,
19          miaux_object_from_file(name, filename,
20                                 *mi_eval_vector(&params->bbox_min),
21                                 *mi_eval_vector(&params->bbox_max)));
22  }
```

Source code of shader "object_file"

Creating geometric objects

Placeholder objects



```
material "diffuse"
    "lambert" (
        "lights" ["light-inst"] )
end material

instance "triangle-instance"
    geometry
        "object_file" (
            "name" "::triangle-object",
            "filename" "triangle_object.mi",
            "bbox_min" -.5 -.5 -.01,
            "bbox_max" .5 .5 .01, )
    material "diffuse"
    transform
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 -.2 0 1
end instance
```

Object data read from file in the shader object_file

```
incremental
object "wedge-object"
    visible on
    shadow 3
    group
        -.5 -.5 .1
        .5 -.5 .1
        0 .5 .1
        -.5 -.5 -.1
        0 .5 -.1
        .5 -.5 -.1
        v 0 v 1 v 2 v 3 v 4 v 5
        p 0 1 2
        p 3 4 5
        p 0 2 4 3
        p 2 1 5 4
        p 1 0 3 5
    end group
end object
```

Object file wedge_object.mi

```
declare shader
    geometry "instanced_object_file" (
        string "name",
        string "filename",
        material "material",
        vector "bbox_min" default -1 -1 -1,
        vector "bbox_max" default 1 1 1,
        array vector "positions" )
end declare
```

Scene file declaration of shader "instanced_object_file"

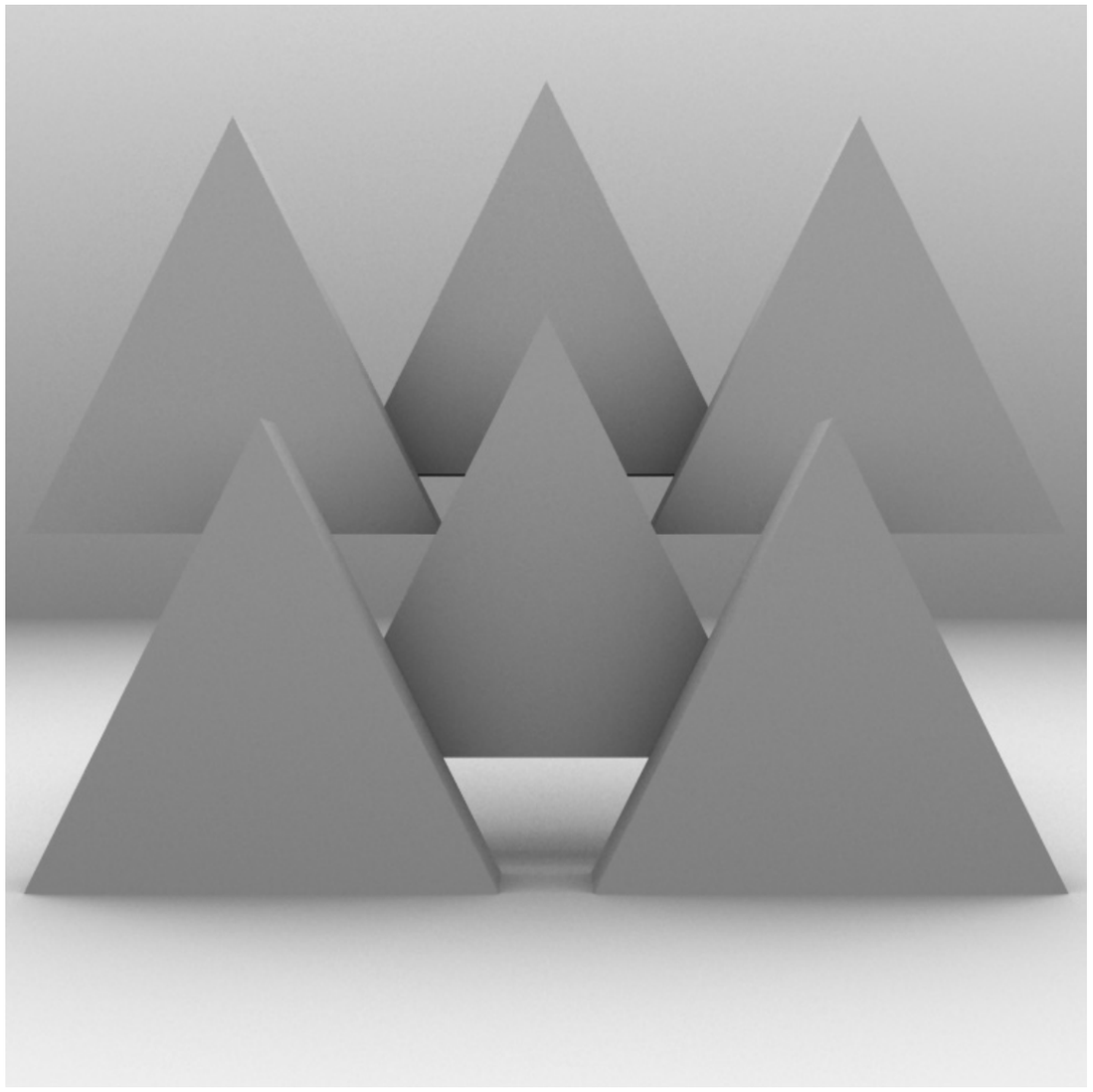
Creating geometric objects


```
 1  struct instanced_object_file {
 2      miTag name;
 3      miTag filename;
 4      miTag material;
 5      miVector bbox_min;
 6      miVector bbox_max;
 7      int i_positions;
 8      int n_positions;
 9      miVector positions[1];
10  };
11
12  miBoolean instanced_object_file (
13      miTag *result, miState *state, struct instanced_object_file *params )
14  {
15      int i;
16      static int instance_index;  /* Geometry shaders are single-threaded. */
17      char *filename, *name, instance_name[1024];
18      miInstance *instance;
19      miTag object_tag, instance_tag;
20      int position_count = *mi_eval_integer(&params->n_positions);
21      miVector *positions = mi_eval_vector(params->positions) +
22          *mi_eval_integer(&params->i_positions);
23
24      if (!(name = miaux_tag_to_string(*mi_eval_tag(&params->name), NULL)) ||
25          !(filename = miaux_tag_to_string(*mi_eval_tag(&params->filename), NULL)))
26          return miFALSE;
27
28      object_tag = miaux_object_from_file(name, filename,
29                                          *mi_eval_vector(&params->bbox_min),
30                                          *mi_eval_vector(&params->bbox_max));
31
32      for (i = 0; i < position_count; i++, positions++) {
33          miMatrix matrix;
34          miTag material_tag;
35          sprintf(instance_name, "%s-%d", name, instance_index++);
36          if (!(instance = mi_api_instance_begin(mi_mem_strdup(instance_name))))
37              return miFALSE;
38
39          mi_matrix_ident(matrix);
40          matrix[12] = positions->x;
41          matrix[13] = positions->y;
42          matrix[14] = positions->z;
43          mi_matrix_copy(instance->tf.local_to_global, matrix);
44          mi_matrix_invert(instance->tf.global_to_local,
45                           instance->tf.local_to_global);
46
47          material_tag = *mi_eval_tag(&params->material);
48          if (material_tag)
49              instance->material = mi_phen_clone(state, material_tag);
50
51          instance_tag = mi_api_instance_end(0, object_tag, 0);
52          if (instance_tag == miNULLTAG ||
53              mi_geoshader_add_result(result, instance_tag) == miFALSE)
54              return miFALSE;
55      }
56      return miTRUE;
57  }
```

Source code of shader "instanced_object_file"

Creating geometric objects

Placeholder objects



```
material "amb_occlude"
    "ambient_occlusion_cutoff" (
        "samples" 200,
        "cutoff_distance" 9 )
end material

instance "triangle-instance"
    geometry
        "instanced_object_file" (
             "name" "::wedge-object",
             "filename" "wedge_object.mi",
             "bbox_min" -.5 -.5 -.01,
             "bbox_max" .5 .5 .01,
             "material" "amb_occlude",
             "positions"
                 [ -0.6  0.0  -1.0,
                    0.6  0.0  -1.0,
                    0.0  0.25 -1.25,
                   -0.75 0.75 -1.5,
                    0.75 0.75 -1.5,
                    0.0  0.9  -1.75 ] )
end instance
```

Multiple instances created from a file in shader instanced_object_file

Modeling hair

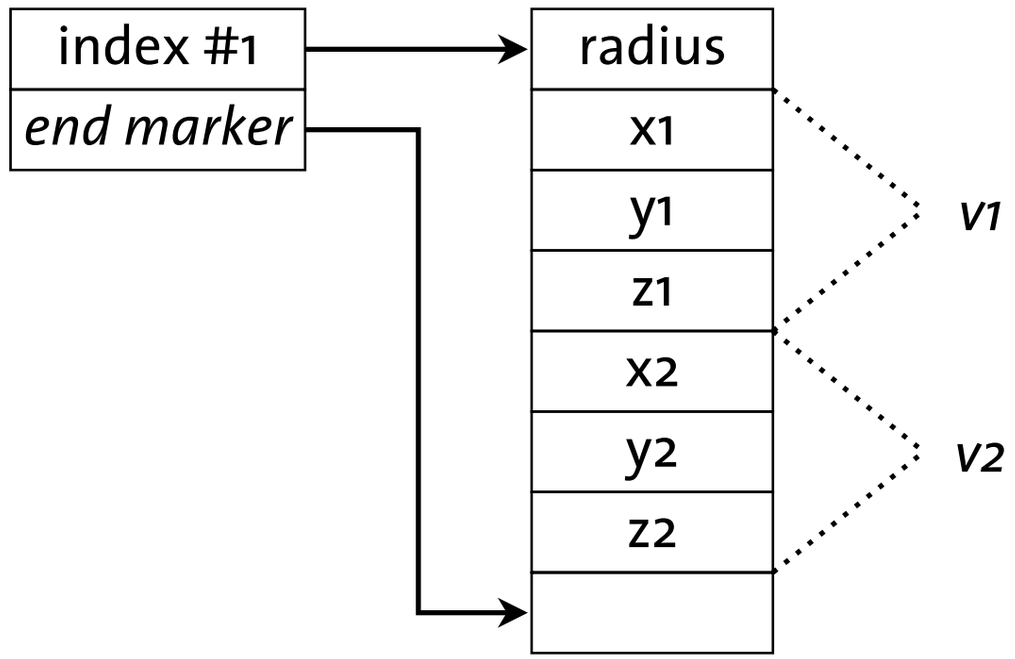
Modeling hair

Placeholder objects and callback functions
Structure of the shader and its callback
Basic principles of the hair geometric primitive
A geometry shader for a single hair
Visualizing the hair's barycentric coordinates
Higher order curves in the hair primitive
Additional data attached to the hair primitive
Multiple hairs
A basic lighting model for hair
The hair primitive as a general modeling tool
The hair primitive and particle systems
Particle system data and dynamic rendering effects

Single hair with two hair vertices

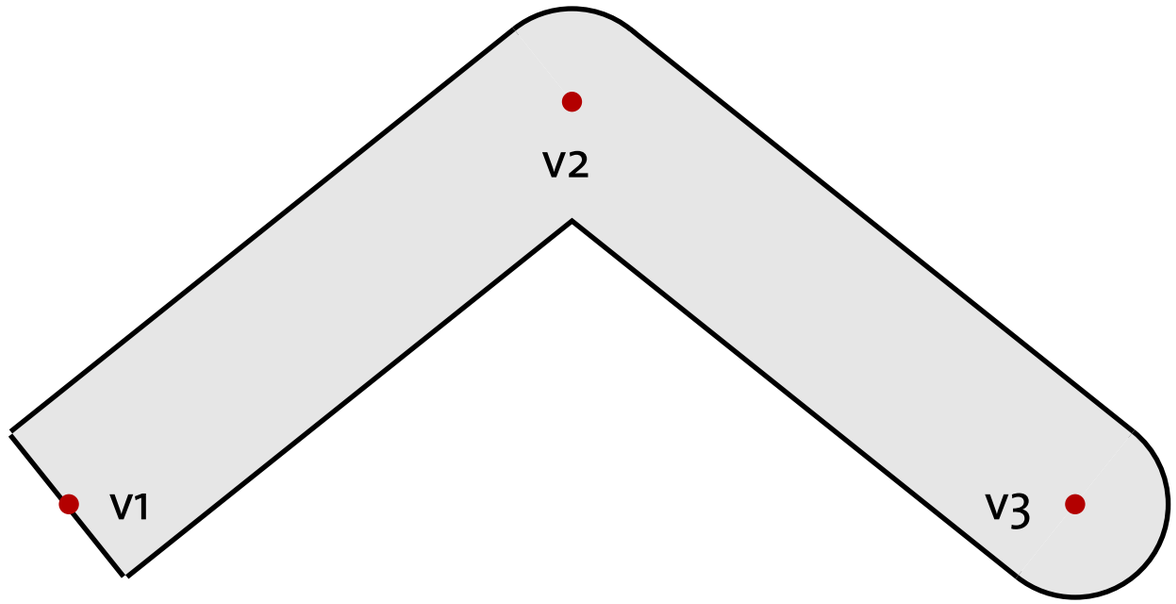
Modeling hair

Basic principles of the hair geometric primitive



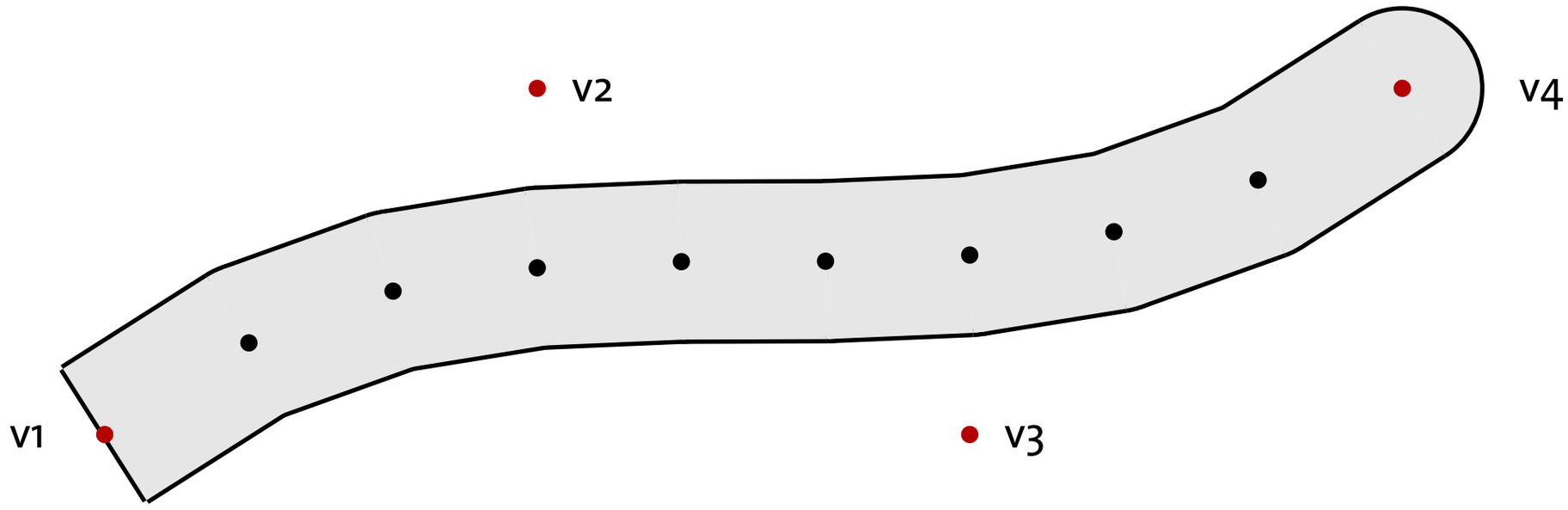
One hair with one segment (two vertices) and a radius for the entire hair

Basic principles of the hair geometric primitive



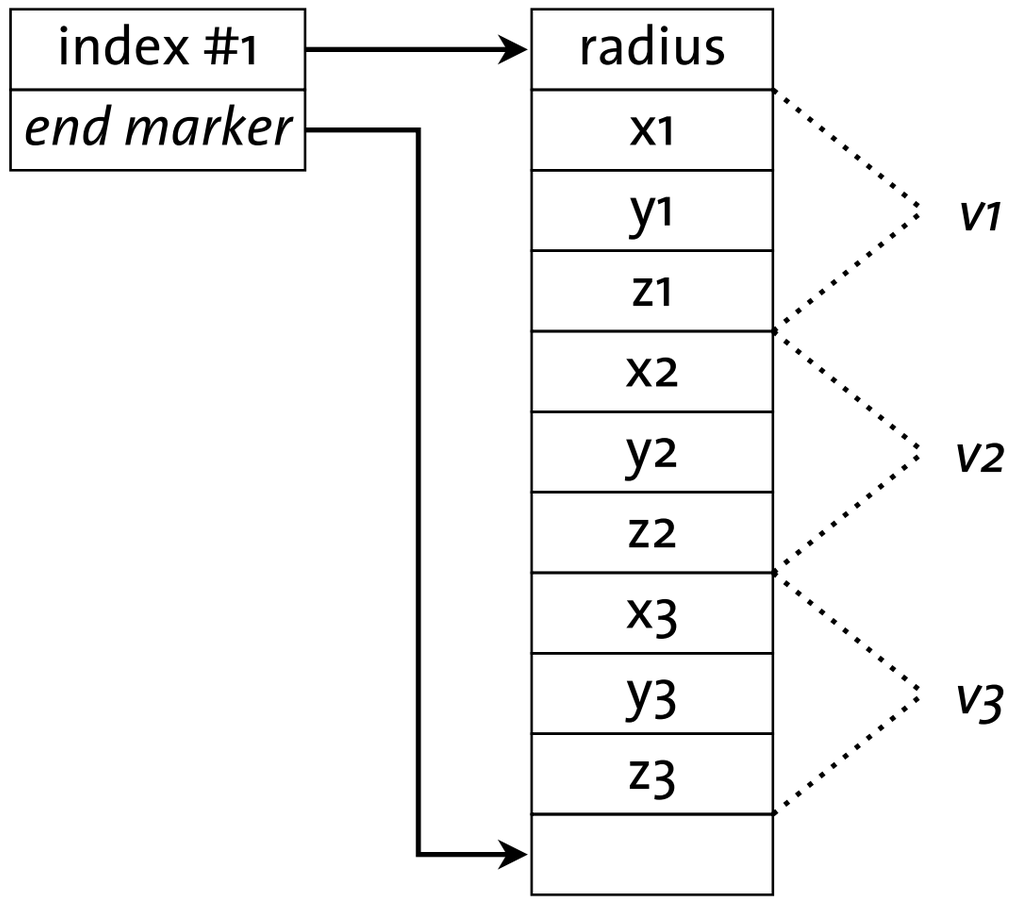
Single hair with three hair vertices

Basic principles of the hair geometric primitive



Single hair with four hair vertices and Bezier curve (degree = 3, approximation = 10)

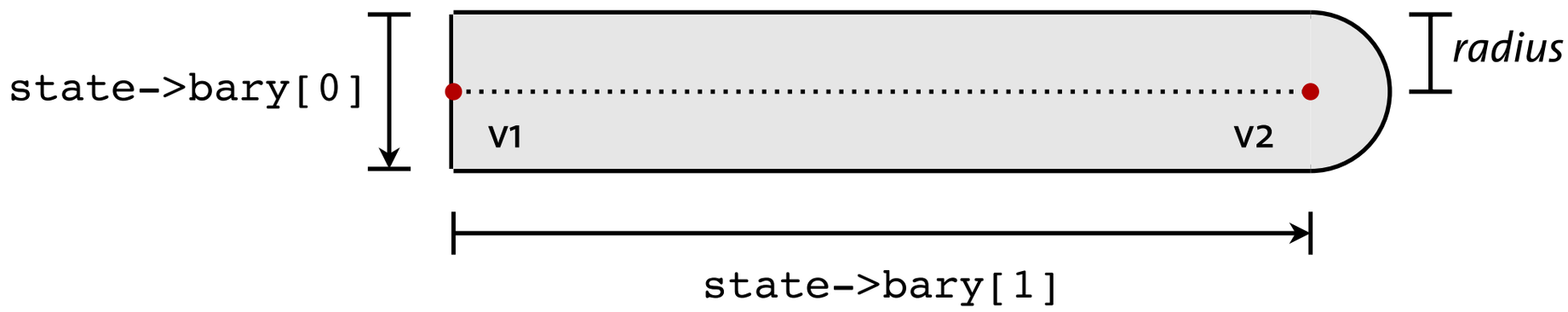
Basic principles of the hair geometric primitive



One hair with two segments (three vertices) and a radius for the entire hair

Modeling hair

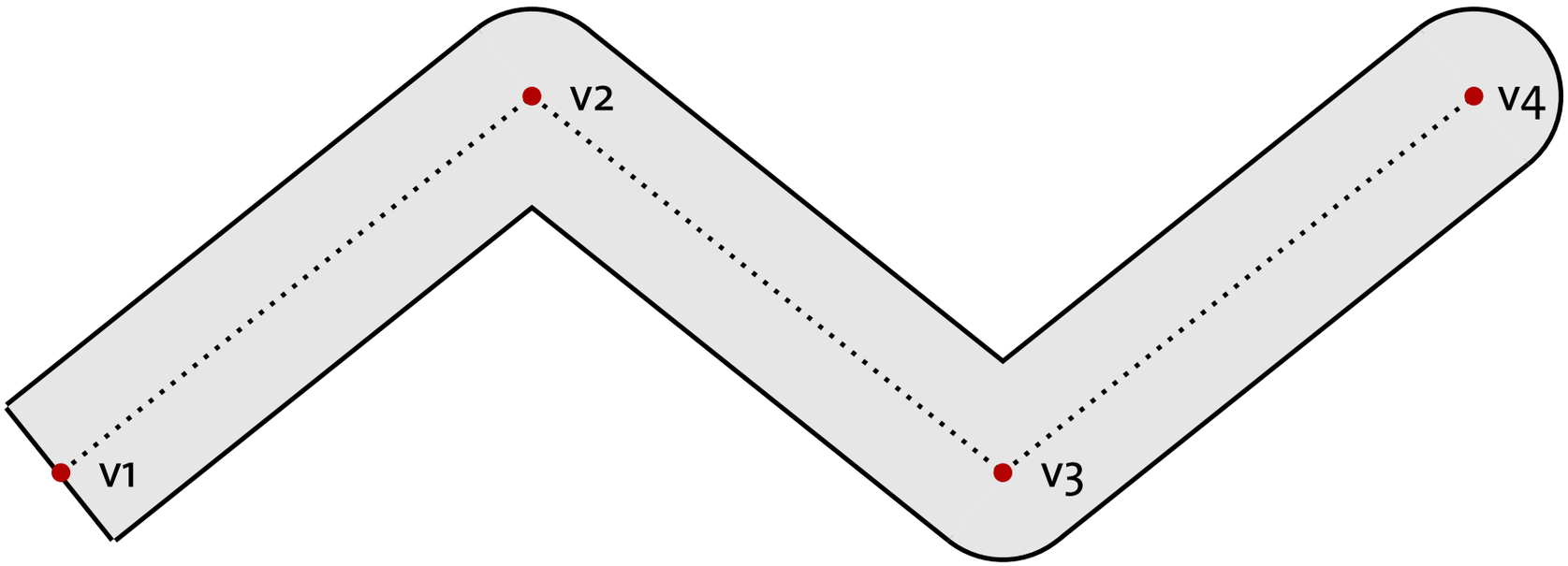
Basic principles of the hair geometric primitive



Barycentric coordinates describing the size of the hair

Modeling hair

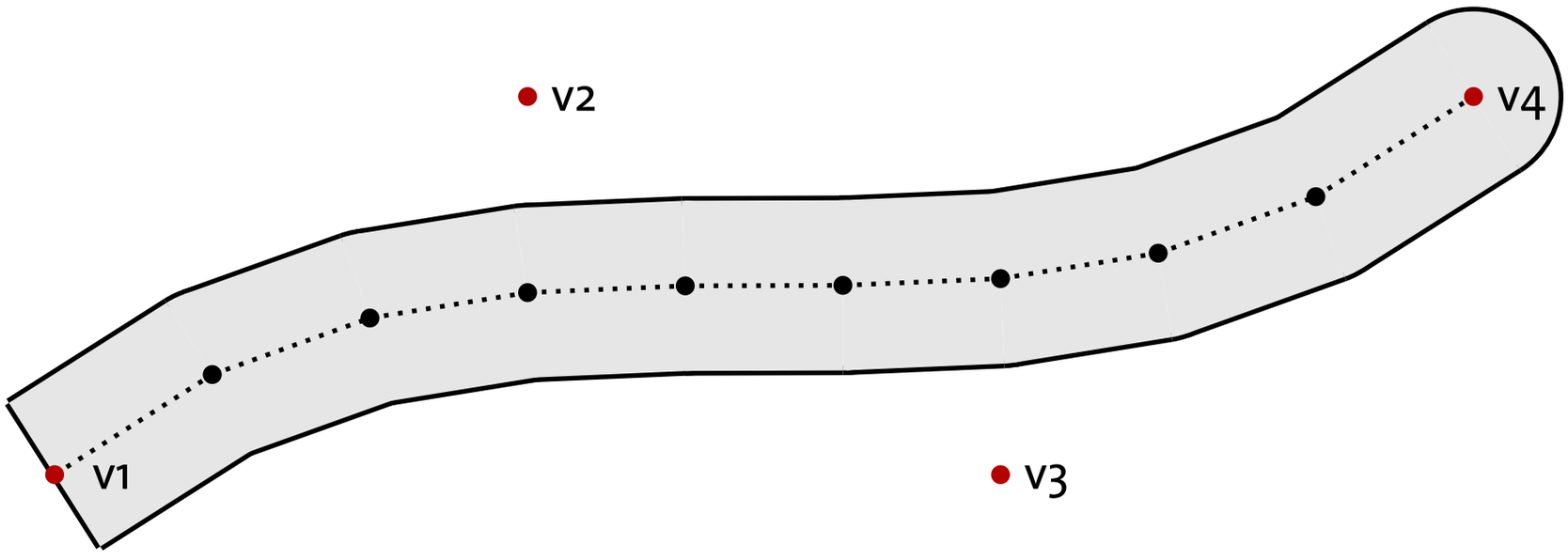
Basic principles of the hair geometric primitive



Length in code{state->bary[1]} defined by the length of line segments connecting vertices

Modeling hair

Basic principles of the hair geometric primitive



Bezier curve length defined by segments created from approximation points

Modeling hair

A geometry shader for a single hair

```
declare shader
    geometry "hair_geo_2v" (
        string "name",
        scalar "radius" default .1,
        vector "start" default -0.5 0 0,
        vector "end" default 0.5 0 0 )
end declare
```

Scene file declaration of shader "hair_geo_2v"

```
1  void miaux_init_bbox(miObject *obj)
2  {
3      obj->bbox_min.x = miHUGE_SCALAR;
4      obj->bbox_min.y = miHUGE_SCALAR;
5      obj->bbox_min.z = miHUGE_SCALAR;
6      obj->bbox_max.x = -miHUGE_SCALAR;
7      obj->bbox_max.y = -miHUGE_SCALAR;
8      obj->bbox_max.z = -miHUGE_SCALAR;
9  }
```

Auxiliary function: miaux_init_bbox

```
1  void miaux_adjust_bbox(miObject *obj, miVector *v, miScalar extra)
2  {
3      miVector v_extra, vmin, vmax;
4      miaux_set_vector(&v_extra, extra, extra, extra);
5      mi_vector_sub(&vmin, v, &v_extra);
6      mi_vector_add(&vmax, v, &v_extra);
7      mi_vector_min(&obj->bbox_min, &obj->bbox_min, &vmin);
8      mi_vector_max(&obj->bbox_max, &obj->bbox_max, &vmax);
9  }
```

Auxiliary function: miaux_adjust_bbox

```
1  void miaux_describe_bbox(miObject *obj)
2  {
3      mi_progress("Object bbox: %f,%f,%f  %f,%f,%f",
4                  obj->bbox_min.x, obj->bbox_min.y, obj->bbox_min.z,
5                  obj->bbox_max.x, obj->bbox_max.y, obj->bbox_max.z);
6  }
```

Auxiliary function: miaux_describe_bbox

```
typedef void (*miaux_bbox_function)(miObject*, void*);
```

Bounding box creation function type miaux_bbox_function

Modeling hair

A geometry shader for a single hair

```
1  typedef struct {
2      miTag name;
3      miScalar radius;
4      miVector start;
5      miVector end;
6  } hair_geo_2v_t;
```

Source code of struct "hair_geo_2v_t"

Modeling hair

A geometry shader for a single hair

```
1  void hair_geo_2v_bbox(miObject *obj, void* params)
2  {
3      hair_geo_2v_t *p = (hair_geo_2v_t*)params;
4      miScalar bbox_increase = p->radius + .001;
5      miaux_init_bbox(obj);
6      miaux_adjust_bbox(obj, &p->start, bbox_increase);
7      miaux_adjust_bbox(obj, &p->end, bbox_increase);
8      miaux_describe_bbox(obj);
9  }
```

Source code of shader "hair_geo_2v"

Modeling hair

A geometry shader for a single hair

```
 1  void miaux_define_hair_object(
 2      miTag name_tag, miaux_bbox_function bbox_function, void *params,
 3      miTag *geoshader_result, miApi_object_callback callback)
 4  {
 5      miTag tag;
 6      miObject *obj;
 7      char *name = miaux_tag_to_string(name_tag, "::hair");
 8      obj = mi_api_object_begin(mi_mem_strdup(name));
 9      obj->visible = miTRUE;
10      obj->shadow = obj->reflection = obj->refraction = 3;
11      bbox_function(obj, params);
12      if (geoshader_result != NULL && callback != NULL) {
13          mi_api_object_callback(callback, params);
14          tag = mi_api_object_end();
15          mi_geoshader_add_result(geoshader_result, tag);
16          obj = (miObject *)mi_scene_edit(tag);
17          obj->geo.placeholder_list.type = miOBJECT_HAIR;
18          mi_scene_edit_end(tag);
19      }
20  }
```

Auxiliary function: miaux_define_hair_object

Modeling hair

A geometry shader for a single hair

```
 1  miBoolean hair_geo_2v (
 2      miTag *result, miState *state, hair_geo_2v_t *params )
 3  {
 4      hair_geo_2v_t *p = (hair_geo_2v_t*)mi_mem_allocate(sizeof(hair_geo_2v_t));
 5      p->name   = *mi_eval_tag(&params->name);
 6      p->radius = *mi_eval_scalar(&params->radius);
 7      p->start  = *mi_eval_vector(&params->start);
 8      p->end    = *mi_eval_vector(&params->end);
 9
10      miaux_define_hair_object(
11          p->name, hair_geo_2v_bbox, p, result, hair_geo_2v_callback);
12
13      return miTRUE;
14  }
```

Source code of main shader of "hair_geo_2v"

Modeling hair

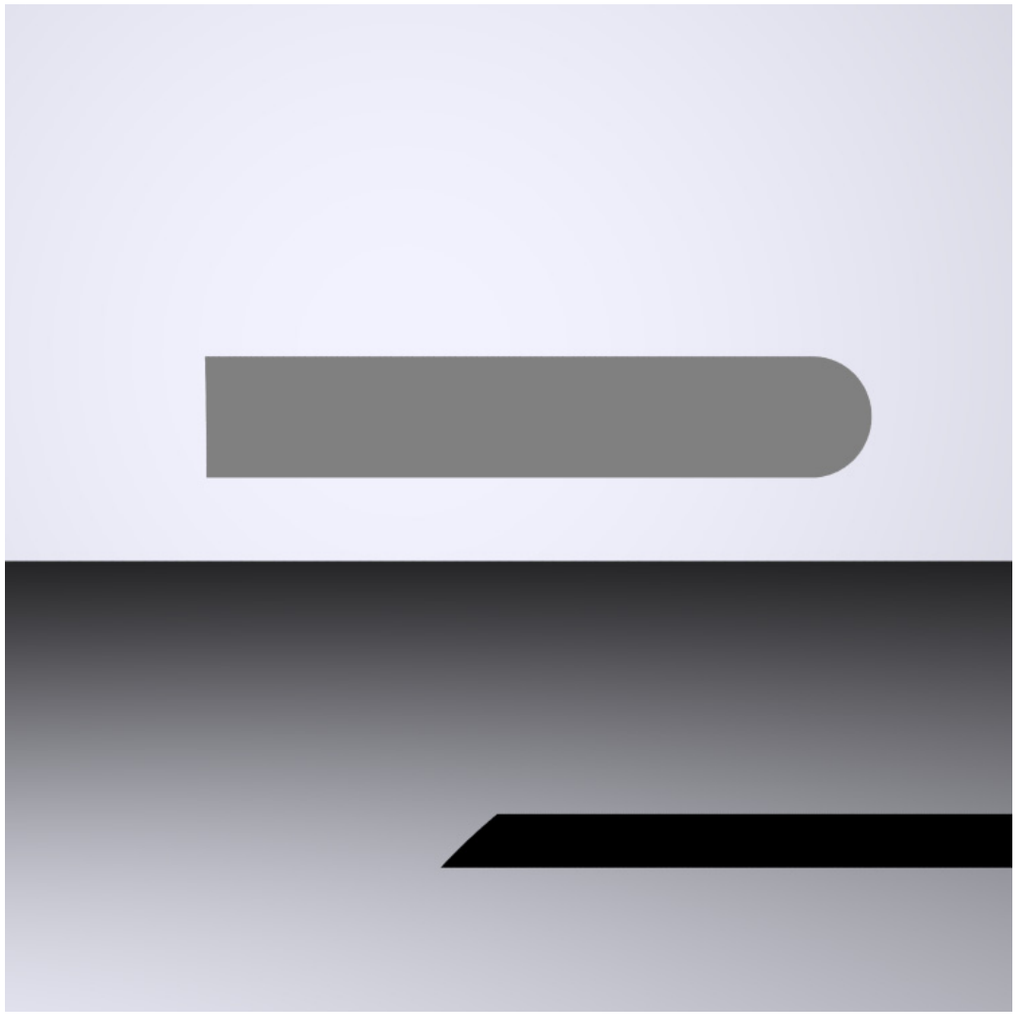
A geometry shader for a single hair

```
 1  miBoolean hair_geo_2v_callback(miTag tag, void *params)
 2  {
 3      miHair_list *hair_list;
 4      miScalar    *hair_scalars;
 5      miGeoIndex  *hair_indices;
 6      hair_geo_2v_t *p = (hair_geo_2v_t*)params;
 7      int hair_scalar_count = 7;
 8
 9      mi_api_incremental(miTRUE);
10      miaux_define_hair_object(p->name, hair_geo_2v_bbox, p, NULL, NULL);
11
12      hair_list = mi_api_hair_begin();
13      hair_list->degree = 1;
14      mi_api_hair_info(0, 'r', 1);
15
16      hair_scalars = mi_api_hair_scalars_begin(hair_scalar_count);
17      *hair_scalars++ = p->radius;
18      *hair_scalars++ = p->start.x;
19      *hair_scalars++ = p->start.y;
20      *hair_scalars++ = p->start.z;
21      *hair_scalars++ = p->end.x;
22      *hair_scalars++ = p->end.y;
23      *hair_scalars++ = p->end.z;
24      mi_api_hair_scalars_end(hair_scalar_count);
25
26      hair_indices = mi_api_hair_hairs_begin(2);
27      hair_indices[0] = 0;
28      hair_indices[1] = 7;
29      mi_api_hair_hairs_end();
30
31      mi_api_hair_end();
32      mi_api_object_end();
33
34      return miTRUE;
35  }
```

Source code of shader "hair_geo_2v"

Modeling hair

A geometry shader for a single hair



```
material "gray"
    "one_color" (
        "color" .5 .5 .5 )
end material

instance "hair-instance"
    geometry
        "hair_geo_2v" (
            "radius" .1,
            "start" -.5 .4 0,
            "end" .5 .4 0 )
    material "gray"
end instance
```

One hair of two vertices rendered with constant color

Visualizing the hair's barycentric coordinates

```
declare shader
    color "hair_color_bary" ()
end declare
```

Scene file declaration of shader "hair_color_bary"

Modeling hair

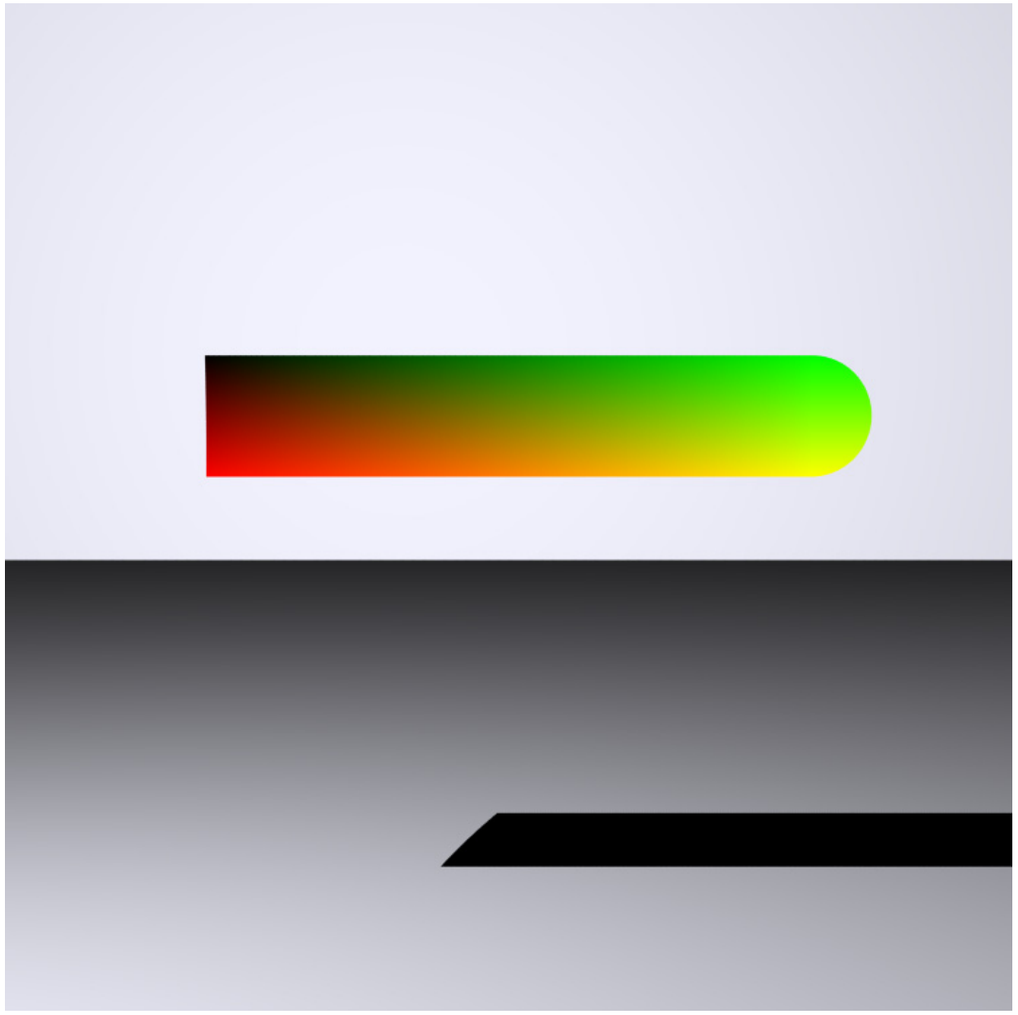
Visualizing the hair's barycentric coordinates

```
1  miBoolean hair_color_bary (
2      miColor *result, miState *state, void *params )
3  {
4      result->r = state->bary[0];
5      result->g = state->bary[1];
6      result->b = 0;
7      return miTRUE;
8  }
```

Source code of shader "hair_color_bary"

Modeling hair

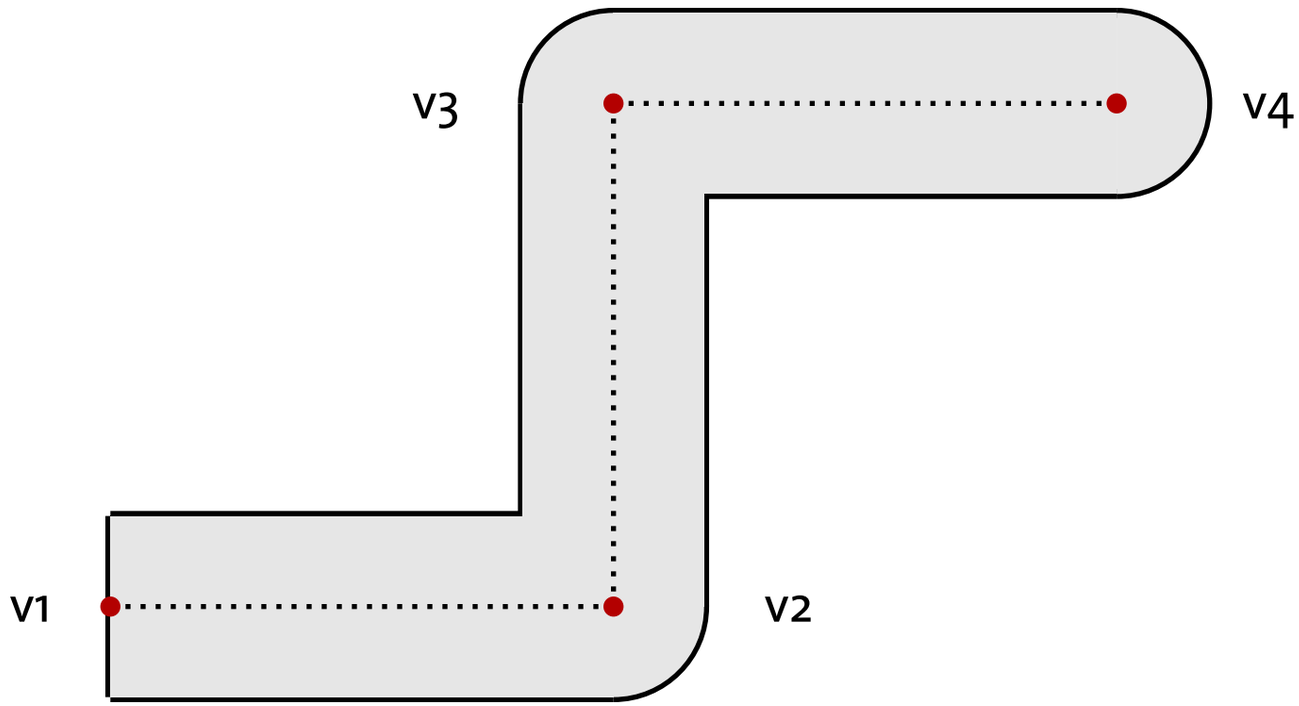
Visualizing the hair's barycentric coordinates



```
material "bary"
    "hair_color_bary" ()
end material

instance "hair-instance"
    geometry
        "hair_geo_2v" (
            "radius" .1,
            "start" -.5 .4 0,
            "end" .5 .4 0 )
    material "bary"
end instance
```

Two-vertex hair rendered with shader hair_color_bary

Higher order curves in the hair primitive



Degree parameter of 1 with vertices connected by straight lines

Modeling hair

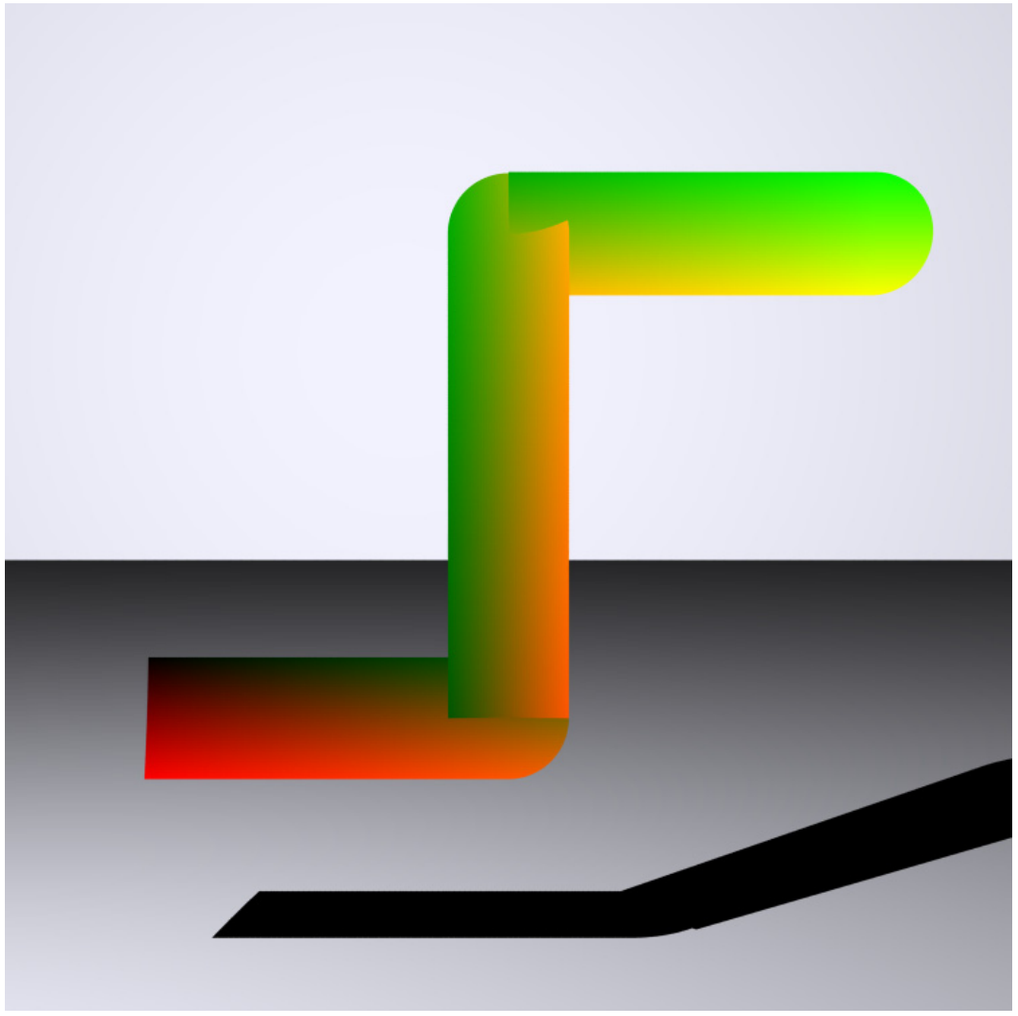
Higher order curves in the hair primitive

```
declare shader
    geometry "hair_geo_4v" (
        string "name",
        scalar "radius" default .1,
        vector "v1" default -1.5 0 0,
        vector "v2" default  -.5 0 0,
        vector "v3" default   .5 0 0,
        vector "v4" default  1.5 0 0,
        integer "approximation" default 2,
        integer "degree" default 1 )
end declare
```

Scene file declaration of shader "hair_geo_4v"

Modeling hair

Higher order curves in the hair primitive

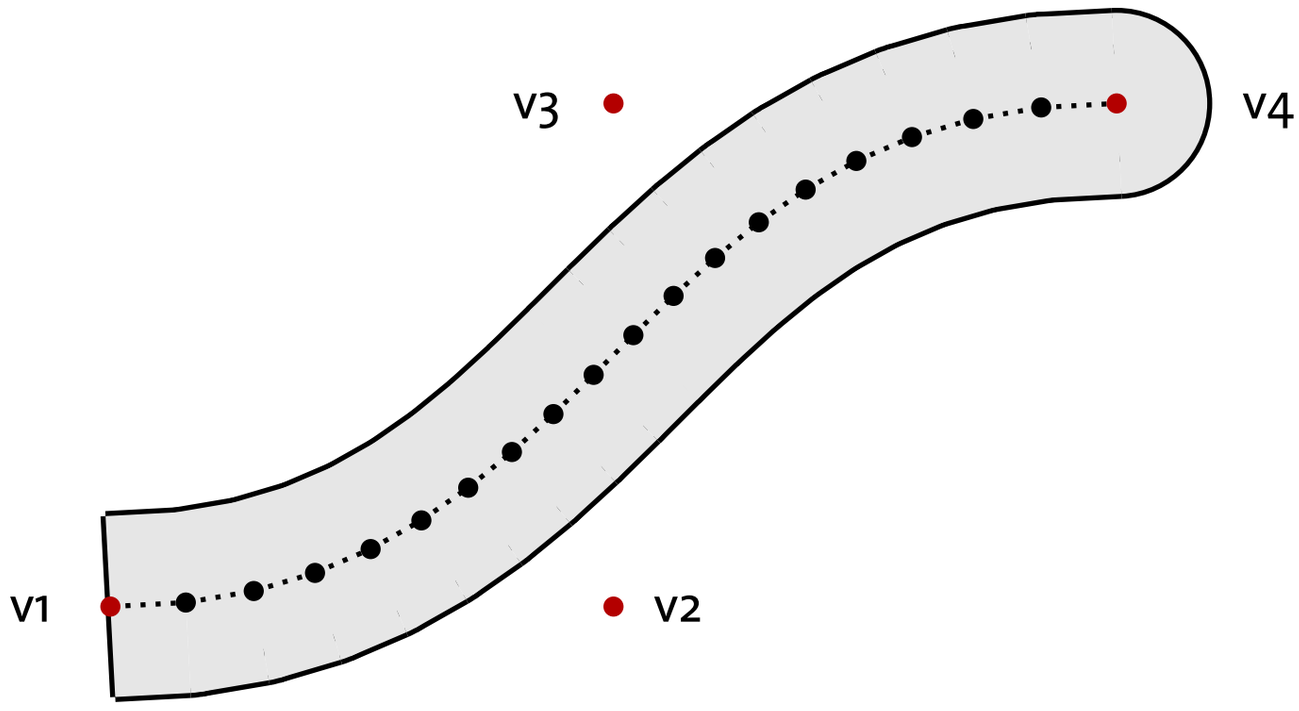


```
material "bary"
    "hair_color_bary" ()
end material

instance "hair-instance"
    geometry
        "hair_geo_4v" (
            "name" "::hair-1",
            "radius" .1,
            "v1" -.6 -.1 0,
            "v2" 0 -.1 0,
            "v3" 0 .7 0,
            "v4" .6 .7 0,
            "degree" 1 )
    material "bary"
end instance
```

Hair curve with a degree value of 1 rendered with shader hair_color_bary.

Modeling hair

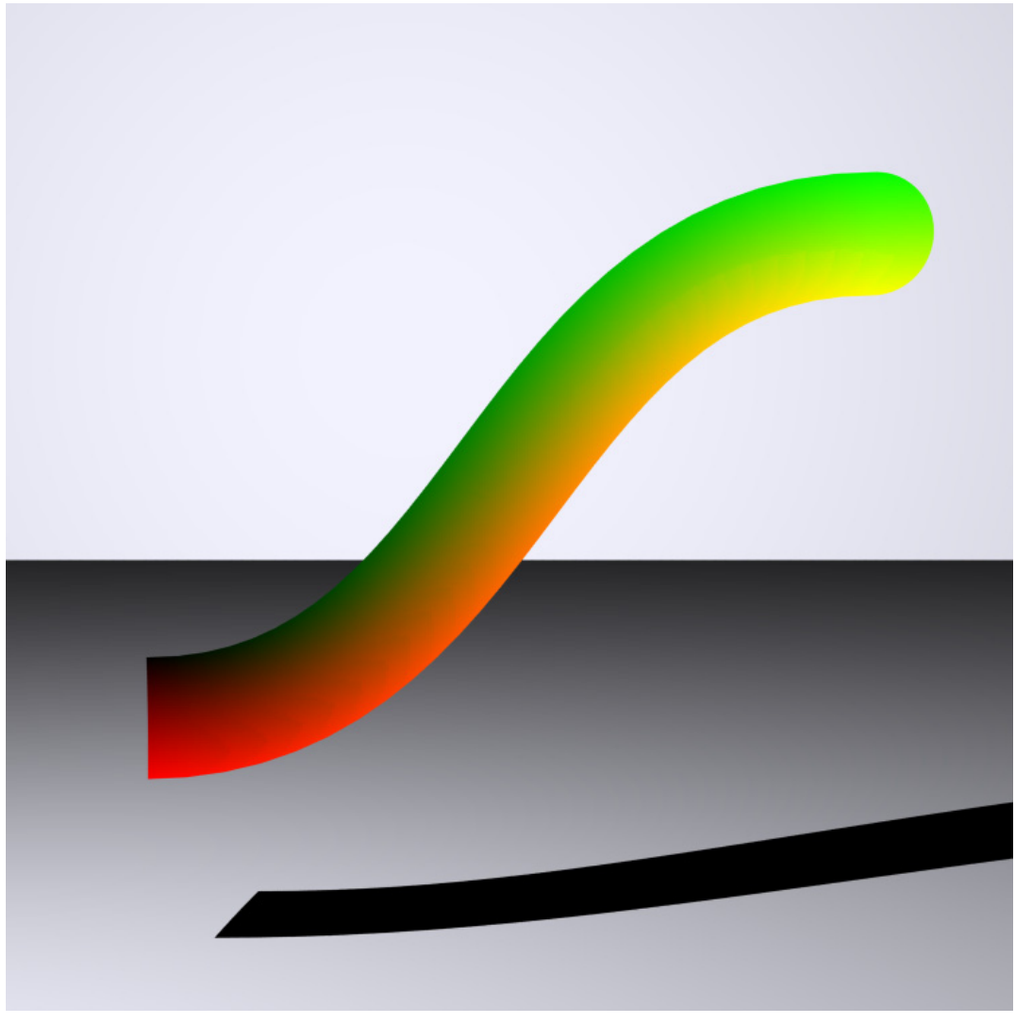
Higher order curves in the hair primitive



Bezier curve (degree=3, approximation=20) defined by four vertices

Modeling hair

Higher order curves in the hair primitive



```
material "bary"
    "hair_color_bary" ()
end material

instance "hair-instance"
    geometry
        "hair_geo_4v" (
            "name" "::hair-1",
            "radius" .1,
            "v1" -.6 -.1 0,
            "v2" 0 -.1 0,
            "v3" 0 .7 0,
            "v4" .6 .7 0,
            "approximation" 30,
            "degree" 3 )
    material "bary"
end instance
```

Hair curve with a degree value of 3 rendered with shader hair_color_bary

Modeling hair

Higher order curves in the hair primitive

```
1  void hair_geo_4v_bbox(miObject *obj, void* params)
2  {
3      hair_geo_4v_t *p = (hair_geo_4v_t*)params;
4      miScalar bbox_increase = p->radius + .001;
5      miaux_init_bbox(obj);
6      miaux_adjust_bbox(obj, &p->v1, bbox_increase);
7      miaux_adjust_bbox(obj, &p->v2, bbox_increase);
8      miaux_adjust_bbox(obj, &p->v3, bbox_increase);
9      miaux_adjust_bbox(obj, &p->v4, bbox_increase);
10     miaux_describe_bbox(obj);
11 }
```

Source code of shader "hair_geo_4v"

Modeling hair

Higher order curves in the hair primitive

```
 1  typedef struct {
 2      miTag name;
 3      miScalar radius;
 4      miVector v1;
 5      miVector v2;
 6      miVector v3;
 7      miVector v4;
 8      miInteger approximation;
 9      miInteger degree;
10  } hair_geo_4v_t;
11
12  miBoolean hair_geo_4v (
13      miTag *result, miState *state, hair_geo_4v_t *params )
14  {
15      hair_geo_4v_t *p = (hair_geo_4v_t*)mi_mem_allocate(sizeof(hair_geo_4v_t));
16      p->radius       = *mi_eval_scalar(&params->radius);
17      p->v1           = *mi_eval_vector(&params->v1);
18      p->v2           = *mi_eval_vector(&params->v2);
19      p->v3           = *mi_eval_vector(&params->v3);
20      p->v4           = *mi_eval_vector(&params->v4);
21      p->approximation = *mi_eval_integer(&params->approximation);
22      p->degree       = *mi_eval_integer(&params->degree);
23
24      miaux_define_hair_object(
25          p->name, hair_geo_4v_bbox, p, result, hair_geo_4v_callback);
26
27      return miTRUE;
28  }
```

Source code of main shader of "hair_geo_4v"

```
1  void miaux_append_hair_vertex(miScalar **scalar_array, miVector *v)
2  {
3       (*scalar_array)[0] = v->x;
4       (*scalar_array)[1] = v->y;
5       (*scalar_array)[2] = v->z;
6       *scalar_array += 3;
7  }
```

Auxiliary function: miaux_append_hair_vertex

Modeling hair

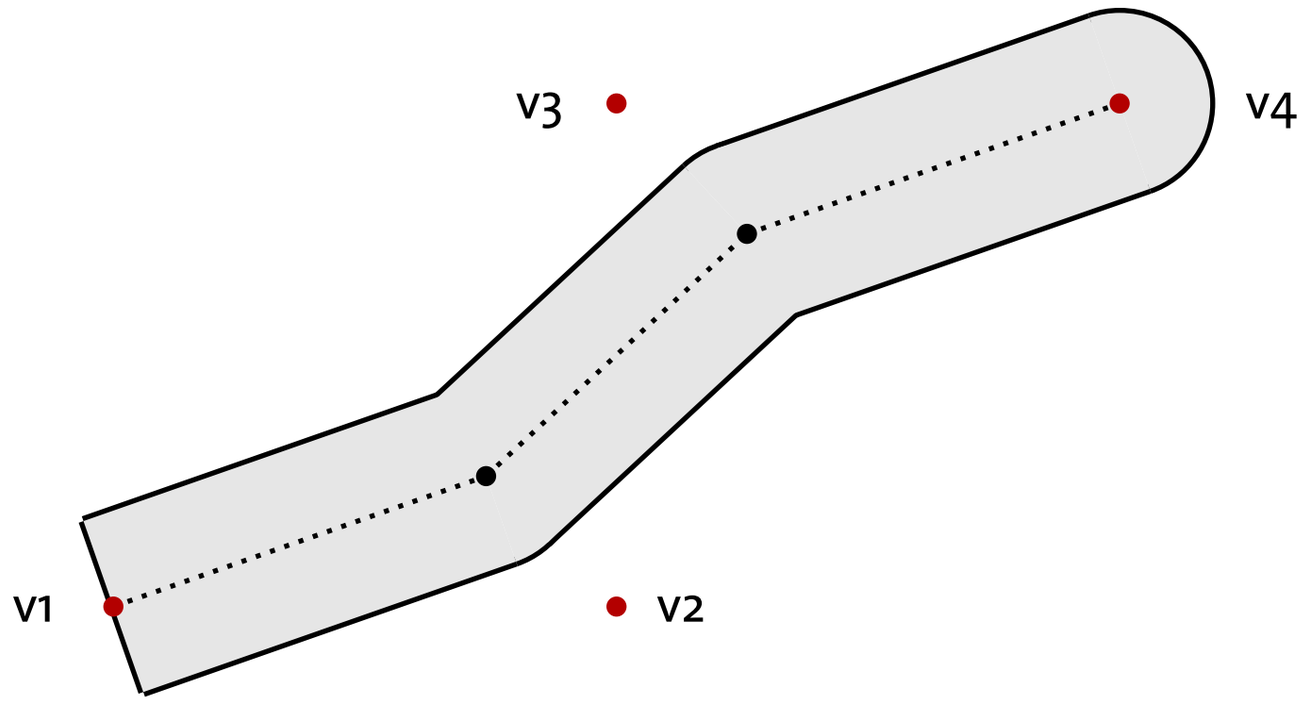
Higher order curves in the hair primitive

```
 1  miBoolean hair_geo_4v_callback(miTag tag, void *params)
 2  {
 3      miHair_list *hair_list;
 4      miScalar    *hair_scalars;
 5      miGeoIndex  *hair_indices;
 6      hair_geo_4v_t *p = (hair_geo_4v_t *)params;
 7      int hair_count = 1, hair_scalar_count = 4 * 3 + 1;
 8
 9      mi_api_incremental(miTRUE);
10      miaux_define_hair_object(p->name, hair_geo_4v_bbox, p, NULL, NULL);
11      hair_list = mi_api_hair_begin();
12      hair_list->approx = p->approximation;
13      hair_list->degree = p->degree;
14      mi_api_hair_info(0, 'r', 1);
15
16      hair_scalars = mi_api_hair_scalars_begin(hair_scalar_count);
17      *hair_scalars++ = p->radius;
18      miaux_append_hair_vertex(&hair_scalars, &p->v1);
19      miaux_append_hair_vertex(&hair_scalars, &p->v2);
20      miaux_append_hair_vertex(&hair_scalars, &p->v3);
21      miaux_append_hair_vertex(&hair_scalars, &p->v4);
22      mi_api_hair_scalars_end(hair_scalar_count);
23
24      hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
25      hair_indices[0] = 0;
26      hair_indices[1] = hair_scalar_count;
27      mi_api_hair_hairs_end();
28
29      mi_api_hair_end();
30      mi_api_object_end();
31
32      return miTRUE;
33  }
```

Source code of shader "hair_geo_4v"

Modeling hair

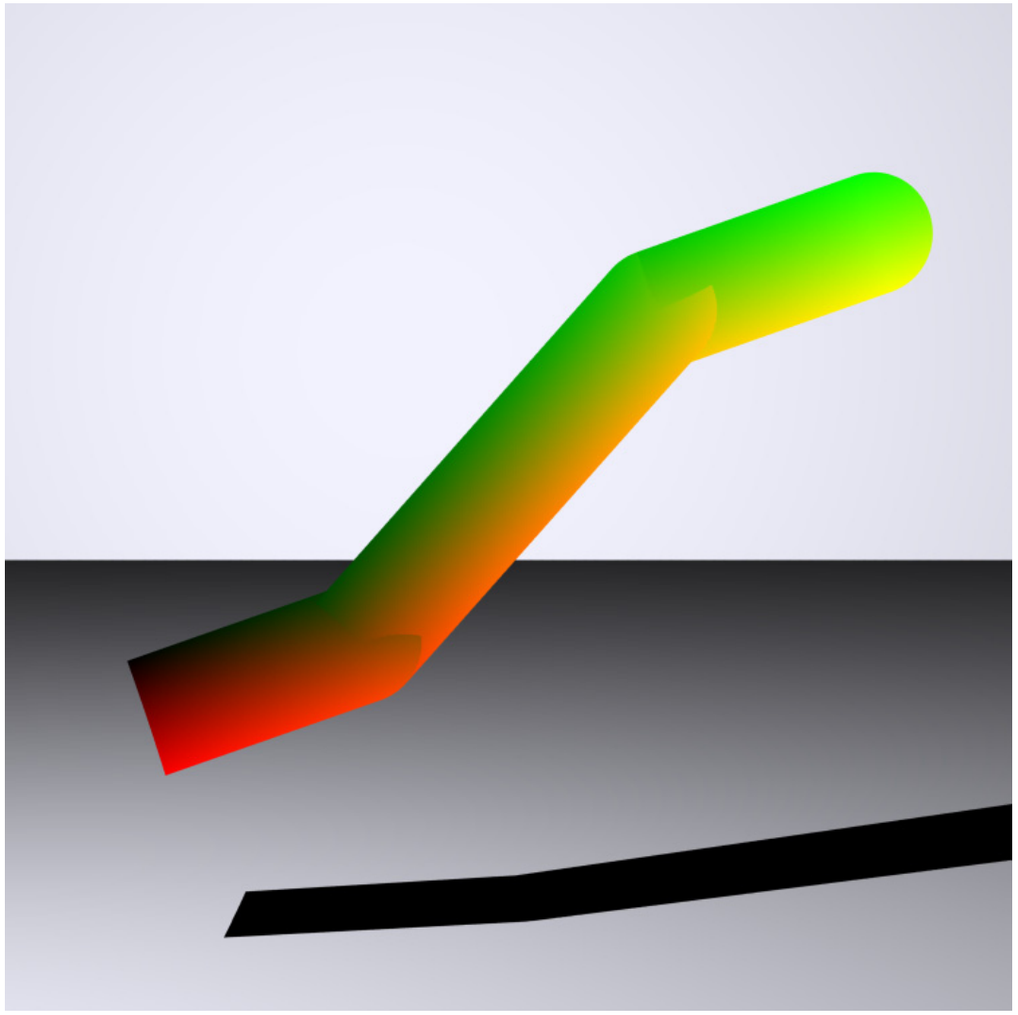
Higher order curves in the hair primitive



Bezier curve with approximation parameter of 4

Modeling hair

Higher order curves in the hair primitive



```
material "bary"
    "hair_color_bary" ()
end material

instance "hair-instance"
    geometry
        "hair_geo_4v" (
            "name" "::hair-1",
            "radius" .1,
            "v1" -.6 -.1 0,
            "v2" 0 -.1 0,
            "v3" 0 .7 0,
            "v4" .6 .7 0,
            "approximation" 4,
            "degree" 3 )
    material "bary"
end instance
```

Hair curve of four vertices with a degree value of 3

Modeling hair

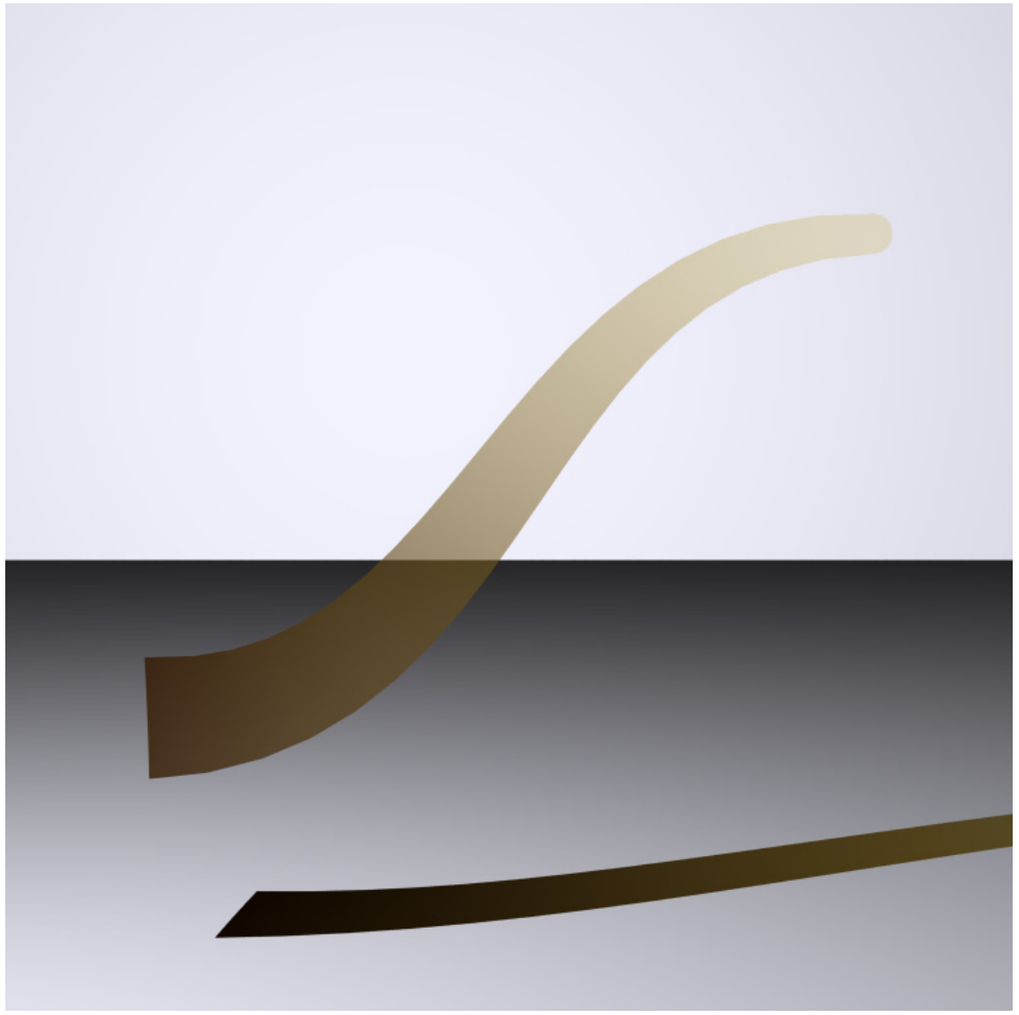
Additional data attached to the hair primitive

```
declare shader
    geometry "hair_geo_4v_texture" (
        string "name",
        vector "v1" default -1.5 0 0,
        vector "v2" default  -.5 0 0,
        vector "v3" default   .5 0 0,
        vector "v4" default  1.5 0 0,
        scalar "root_radius" default .1,
        color  "root_color" default 0 0 0 1,
        scalar "tip_radius" default .01,
        color  "tip_color"  default 1 1 1 0,
        integer "approximation" default 2,
        integer "degree" default 1 )
end declare
```

Scene file declaration of shader "hair_geo_4v_texture"

Modeling hair

Additional data attached to the hair primitive

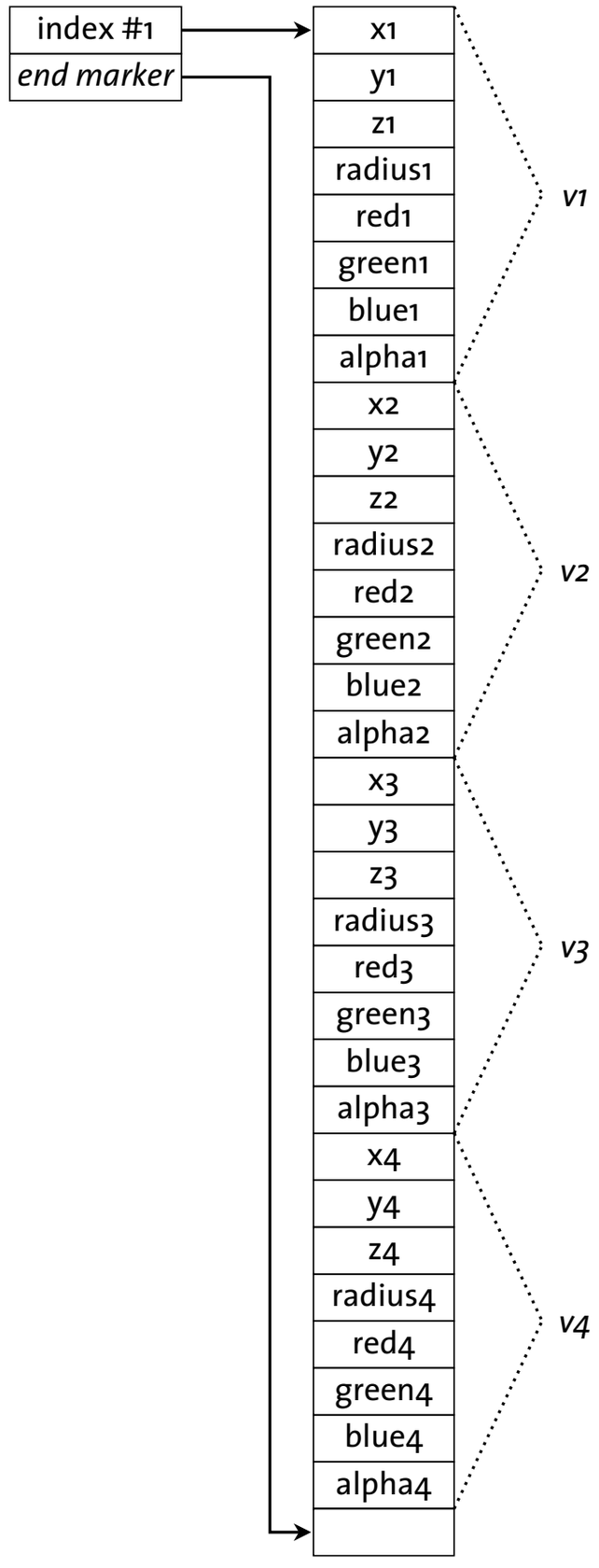


```
material "hair_color"
    "hair_color_texture" ()
     shadow
        "hair_color_texture" ()
end material

instance "hair-instance"
    geometry
        "hair_geo_4v_texture" (
            "name" "::hair-1",
            "v1" -.6 -.1  0,
            "v2"  0  -.1  0,
            "v3"  0   .7  0,
            "v4" .6   .7  0,
            "root_radius" .1,
            "root_color" .2 .1 0 .8,
            "tip_radius" .01,
            "tip_color"  1 .9 .4 .1,
            "approximation" 20,
            "degree" 3 )
    material "hair_color"
end instance
```

Hair curve with radius and color control at each vertex

Modeling hair

Additional data attached to the hair primitive



One hair with four segments, with a radius and texture (color) for each segment

Modeling hair

Additional data attached to the hair primitive

```
1  void hair_geo_4v_texture_bbox(miObject *obj, void* params)
2  {
3      hair_geo_4v_texture_t *p = (hair_geo_4v_texture_t*)params;
4      double max_radius = miaux_max(p->root_radius, p->tip_radius) + .001;
5
6      miaux_init_bbox(obj);
7      miaux_adjust_bbox(obj, &p->p1, max_radius);
8      miaux_adjust_bbox(obj, &p->p2, max_radius);
9      miaux_adjust_bbox(obj, &p->p3, max_radius);
10     miaux_adjust_bbox(obj, &p->p4, max_radius);
11     miaux_describe_bbox(obj);
12 }
```

Source code of shader "hair_geo_4v_texture"

Modeling hair

Additional data attached to the hair primitive

```
 1  typedef struct {
 2      miTag name;
 3      miVector p1;
 4      miVector p2;
 5      miVector p3;
 6      miVector p4;
 7      miScalar root_radius;
 8      miColor root_color;
 9      miScalar tip_radius;
10      miColor tip_color;
11      miInteger approximation;
12      miInteger degree;
13  } hair_geo_4v_texture_t;
14
15  miBoolean hair_geo_4v_texture (
16      miTag *result, miState *state, hair_geo_4v_texture_t *params )
17  {
18      hair_geo_4v_texture_t *p =
19          (hair_geo_4v_texture_t*) mi_mem_allocate(sizeof(hair_geo_4v_texture_t));
20
21      p->p1 = *mi_eval_vector(&params->p1);
22      p->p2 = *mi_eval_vector(&params->p2);
23      p->p3 = *mi_eval_vector(&params->p3);
24      p->p4 = *mi_eval_vector(&params->p4);
25      p->root_radius = *mi_eval_scalar(&params->root_radius);
26      p->root_color = *mi_eval_color(&params->root_color);
27      p->tip_radius = *mi_eval_scalar(&params->tip_radius);
28      p->tip_color  = *mi_eval_color(&params->tip_color);
29      p->approximation = *mi_eval_integer(&params->approximation);
30      p->degree = *mi_eval_integer(&params->degree);
31
32      miaux_define_hair_object(
33          p->name, hair_geo_4v_texture_bbox, p, result,
34          hair_geo_4v_texture_callback);
35
36      return miTRUE;
37  }
```

Source code of main shader of "hair_geo_4v_texture"

Modeling hair

Additional data attached to the hair primitive

```
 1  void miaux_append_hair_data(
 2      miScalar **scalar_array, miVector *v, miScalar position,
 3      miScalar root_radius, miColor *root, miScalar tip_radius, miColor *tip )
 4  {
 5      (*scalar_array)[0] = v->x;
 6      (*scalar_array)[1] = v->y;
 7      (*scalar_array)[2] = v->z;
 8      (*scalar_array)[3] = miaux_fit(position, 0, 1, root_radius, tip_radius);
 9      (*scalar_array)[4] = miaux_fit(position, 0, 1, root->r, tip->r);
10      (*scalar_array)[5] = miaux_fit(position, 0, 1, root->g, tip->g);
11      (*scalar_array)[6] = miaux_fit(position, 0, 1, root->b, tip->b);
12      (*scalar_array)[7] = miaux_fit(position, 0, 1, root->a, tip->a);
13      *scalar_array += 8;
14  }
```

Auxiliary function: miaux_append_hair_data

Modeling hair

Additional data attached to the hair primitive

```
 1  miBoolean hair_geo_4v_texture_callback(miTag tag, void *ptr)
 2  {
 3      miHair_list *hair_list;
 4      miScalar    *hair_scalars;
 5      miGeoIndex  *hair_indices;
 6      hair_geo_4v_texture_t *p = (hair_geo_4v_texture_t *)ptr;
 7      int i;
 8
 9      int hair_count = 1;
10      int vertices_per_hair = 4;
11      int scalars_per_vertex = 8;
12      int hair_scalar_count = vertices_per_hair * scalars_per_vertex;
13
14      miVector vertices[4];
15      vertices[0] = p->p1;
16      vertices[1] = p->p2;
17      vertices[2] = p->p3;
18      vertices[3] = p->p4;
19
20      mi_api_incremental(miTRUE);
21
22      miaux_define_hair_object(
23          p->name, hair_geo_4v_texture_bbox, p, NULL, NULL);
24
25      hair_list = mi_api_hair_begin();
26      hair_list->approx = p->approximation;
27      hair_list->degree = p->degree;
28      mi_api_hair_info(1, 'r', 1);
29      mi_api_hair_info(1, 't', 4);
30
31      hair_scalars = mi_api_hair_scalars_begin(hair_scalar_count);
32      for (i = 0; i < vertices_per_hair; i++)
33          miaux_append_hair_data(
34              &hair_scalars, &vertices[i],
35              miaux_fit(i, 0, vertices_per_hair, 0, 1),
36              p->root_radius, &p->root_color, p->tip_radius, &p->tip_color);
37      mi_api_hair_scalars_end(hair_scalar_count);
38
39      hair_indices = mi_api_hair_hairs_begin(hair_count + 1);
40      hair_indices[0] = 0;
41      hair_indices[1] = hair_scalar_count;
42      mi_api_hair_hairs_end();
43
44      mi_api_hair_end();
45      mi_api_object_end();
46
47      return miTRUE;
48  }
```

Source code of shader "hair_geo_4v_texture"

Modeling hair

Additional data attached to the hair primitive

```
declare shader
    color "hair_color_texture" ()
end declare
```

Scene file declaration of shader "hair_color_texture"

Modeling hair

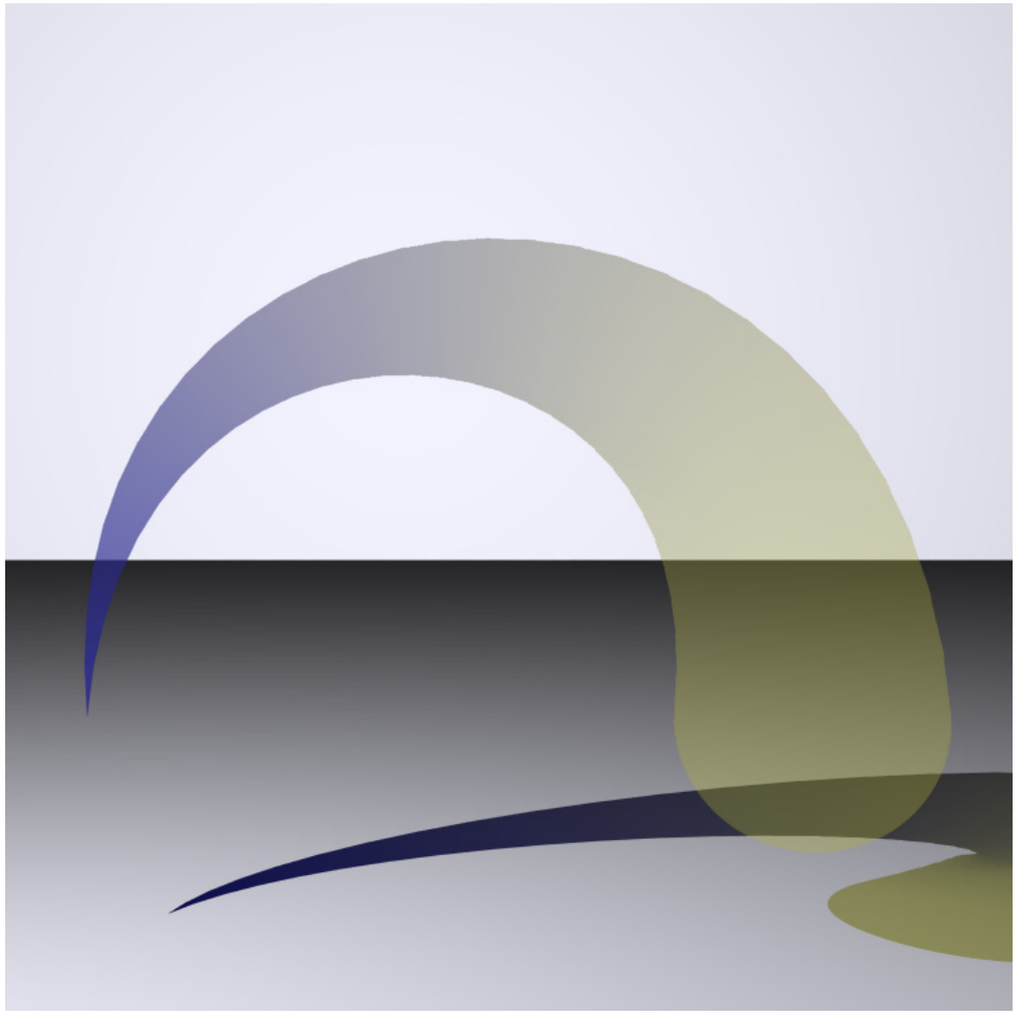
Additional data attached to the hair primitive

```
 1  miBoolean hair_color_texture (
 2      miColor *result, miState *state, void *params )
 3  {
 4      miColor hair_color, opacity, background;
 5      miaux_copy_color(&hair_color, (miColor*)&state->tex_list[0]);
 6
 7      if (state->type == miRAY_SHADOW) {
 8          miScalar transparency = 1.0 - hair_color.a;
 9          result->r *= miaux_shadow_breakpoint(hair_color.r, transparency, 0.5);
10          result->g *= miaux_shadow_breakpoint(hair_color.g, transparency, 0.5);
11          result->b *= miaux_shadow_breakpoint(hair_color.b, transparency, 0.5);
12          return miaux_all_channels_equal(result, 0.0) ? miFALSE : miTRUE;
13      }
14      miaux_copy_color(result, &hair_color);
15      miaux_set_channels(&opacity, result->a);
16      mi_opacity_set(state, &opacity);
17      if (result->a < 1.0) {
18          mi_trace_transparent(&background, state);
19          miaux_blend_channels(result, &background, &opacity);
20      }
21      return miTRUE;
22  }
```

Source code of shader "hair_color_texture"

Modeling hair

Additional data attached to the hair primitive

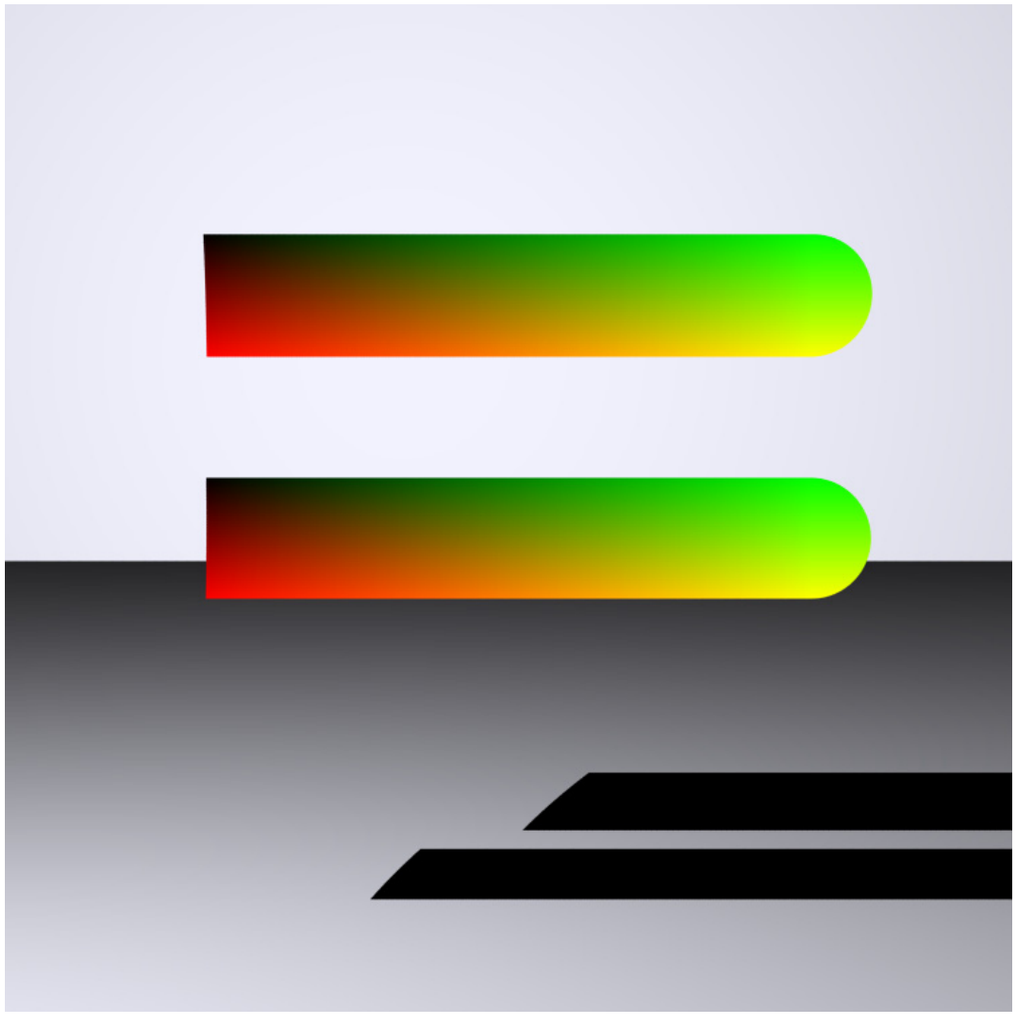


```
material "hair_color"
    "hair_color_texture" ()
    shadow
        "hair_color_texture" ()
end material

instance "hair-instance"
    geometry
        "hair_geo_4v_texture" (
            "name" "::hair-1",
            "v1" -.7 -.1  0,
            "v2" -.7  .8  0,
            "v3" .5   .8  0,
            "v4" .5   -.1  0,
            "root_radius" 0,
            "root_color" .1 .1 .6 .7,
            "tip_radius" .3,
            "tip_color"  .9 .9 .1 .2,
            "approximation" 30,
            "degree" 3 )
    material "hair_color"
end instance
```

Hair curve with larger radius at tip

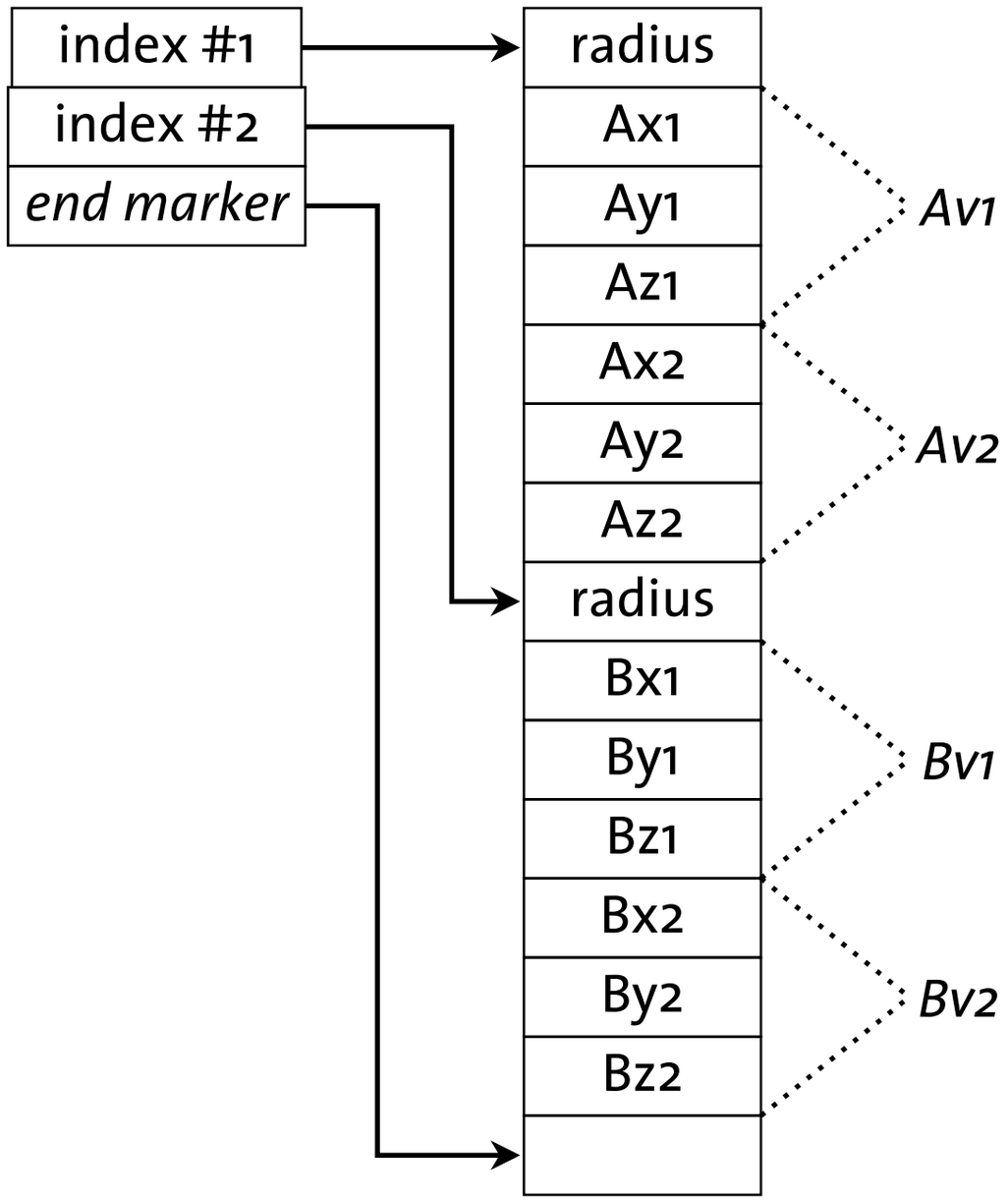
Modeling hair

Multiple hairs



```
material "bary"
    "hair_color_bary" ()
end material

instance "hair-instance"
    geometry
        "hair_geo_2v" (
            "name" "::hair-1",
            "radius" .1,
            "start" -.5 .2 0,
            "end" .5 .2 0 )
        "hair_geo_2v" (
            "name" "::hair-2",
            "radius" .1,
            "start" -.5 .6 0,
            "end" .5 .6 0 )
    material "bary"
end instance
```

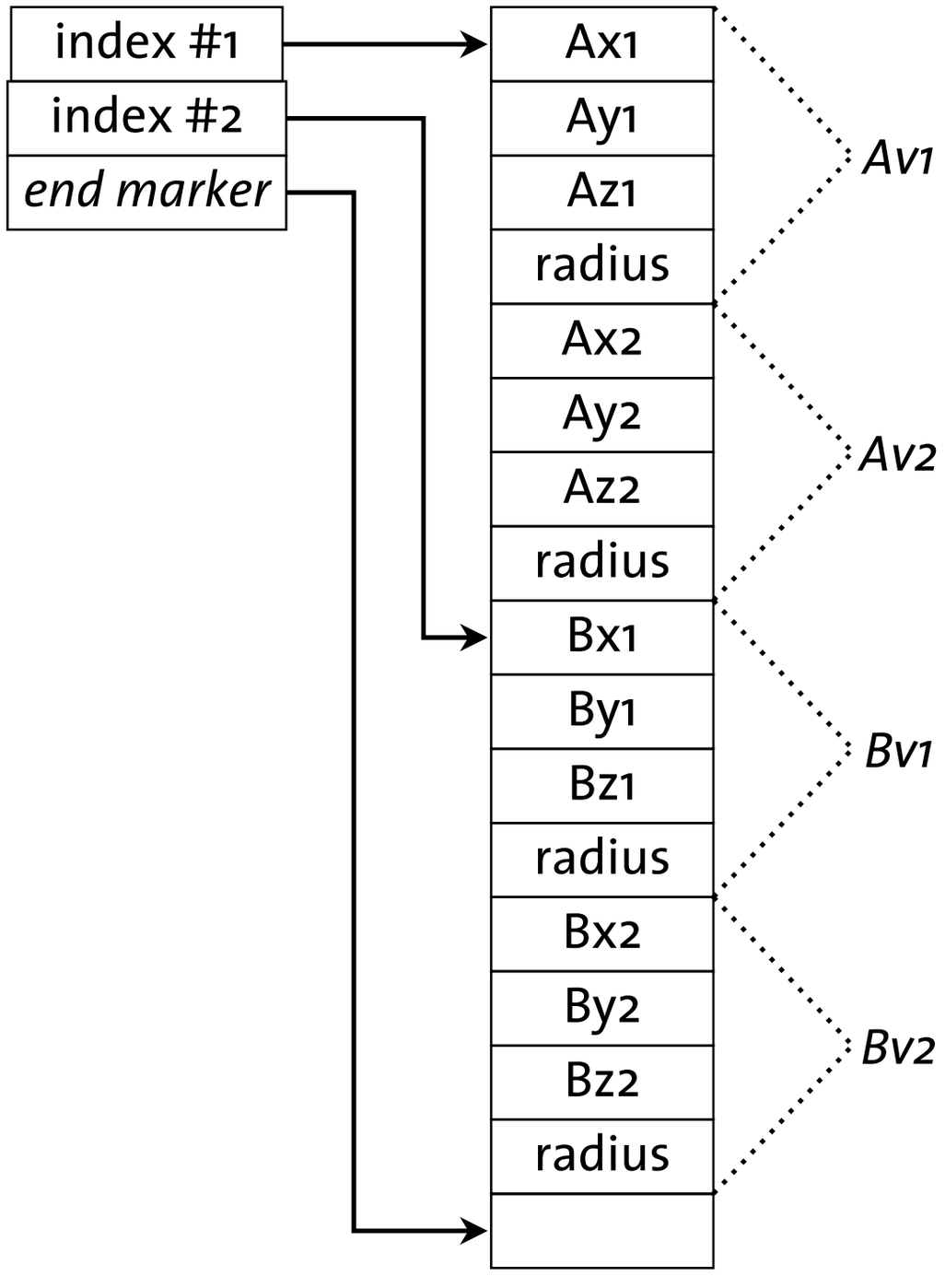
Two hairs created with a list of geometry shaders



Two hairs (A and B) with one segment each and a single radius for each hair

Modeling hair

Multiple hairs



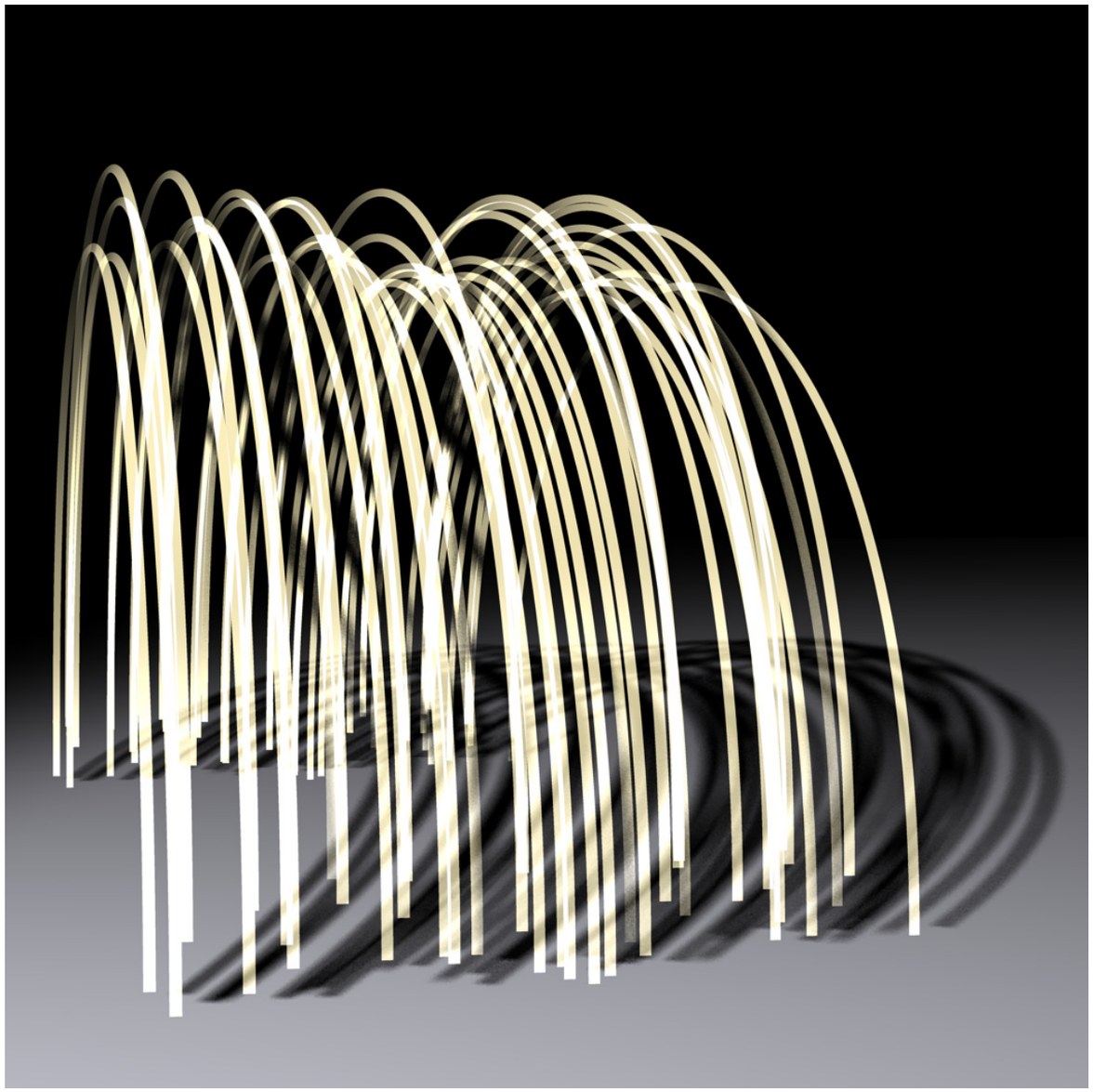
Two hairs (A and B) with two segments each and a radius for each segment

```
declare shader
    geometry "hair_geo_row" (
        string "name",
        integer "count" default 1,
        scalar "radius" default .01,
        vector "bbox_min" default -.5 -.5 -.5,
        vector "bbox_max" default .5 .5 .5,
        scalar "y_offset_max" default .01,
        scalar "z_offset_max" default .01,
        integer "approximation" default 100,
        integer "degree" default 3,
        integer "random_seed" default 1955 )
end declare
```

Scene file declaration of shader "hair_geo_row"

Modeling hair

Multiple hairs

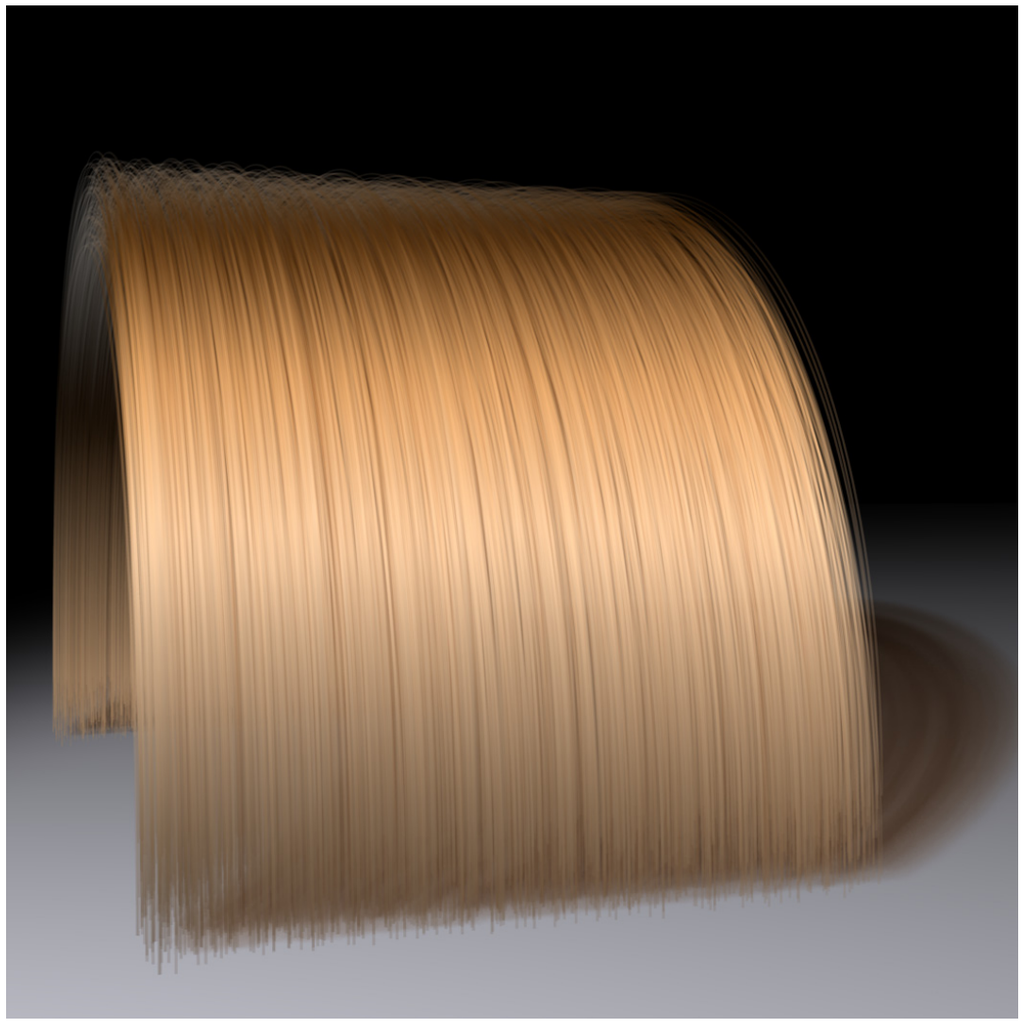


```
material "hair_lambert"
    "lambert" (
        "diffuse" 1 .95 .75,
        "lights" ["light-inst"] )
end material

instance "hair-instance"
    geometry
        "hair_geo_row" (
            "name" "::hair-1",
            "radius" .01,
            "count" 40,
            "bbox_min" -.7 -.5 -2.5,
            "bbox_max"  .7 1.5 0,
            "y_offset_max" .1,
            "z_offset_max" .1, )
    material "hair_lambert"
end instance
```

Bezier curve hairs distributed throughout a bounding box defined by shader parameters

Modeling hair

Multiple hairs



```
options "opt"
    object space
    contrast .1 .1 .1 1
    scanline rapid
    shadow on
    shadowmap detail
    samples collect 5
    filter mitchell
end options

material "hair_light"
    "hair_color_light" (
        "lights" ["light-inst"],
        "diffuse" .55 .3 .05,
        "specular" .65 .65 .65,
        "root_opacity" .2,
        "tip_opacity" .2,
        "exponent" 80 )
    shadow "hair_color_light" (
        "diffuse" .55 .3 .05,
        "root_opacity" .05,
        "tip_opacity" .05, )
end material

instance "hair-instance"
    geometry
        "hair_geo_row" (
            "name" "::hair-1",
            "radius" .0025,
            "count" 4000,
            "bbox_min" -.7 -.5 -2.5,
            "bbox_max"  .7 1.5 0,
            "y_offset_max" .1,
            "z_offset_max" .1, )
    material "hair_light"
end instance
```

Shader hair_geo_row with 4,000 hairs and the material shader developed in the next section

```
1  void hair_geo_row_bbox(miObject *obj, void* params)
2  {
3      hair_geo_row_t *p = (hair_geo_row_t*)params;
4      obj->bbox_min = p->bbox_min;
5      obj->bbox_max = p->bbox_max;
6  }
```

Source code of shader "hair_geo_row"

Modeling hair

Multiple hairs

```
 1  typedef struct {
 2      miTag name;
 3      miInteger count;
 4      miScalar radius;
 5      miVector bbox_min;
 6      miVector bbox_max;
 7      miScalar y_offset_max;
 8      miScalar z_offset_max;
 9      miInteger approximation;
10      miInteger degree;
11      miInteger random_seed;
12  } hair_geo_row_t;
13
14  miBoolean hair_geo_row (
15      miTag *result, miState *state, hair_geo_row_t *params )
16  {
17      hair_geo_row_t *p =
18          (hair_geo_row_t*) mi_mem_allocate(sizeof(hair_geo_row_t));
19      p->name          = *mi_eval_tag(&params->name);
20      p->count         = *mi_eval_integer(&params->count);
21      p->radius        = *mi_eval_scalar(&params->radius);
22      p->bbox_min      = *mi_eval_vector(&params->bbox_min);
23      p->bbox_max      = *mi_eval_vector(&params->bbox_max);
24      p->y_offset_max  = *mi_eval_scalar(&params->y_offset_max);
25      p->z_offset_max  = *mi_eval_scalar(&params->z_offset_max);
26      p->approximation = *mi_eval_integer(&params->approximation);
27      p->degree        = *mi_eval_integer(&params->degree);
28      p->random_seed   = *mi_eval_integer(&params->random_seed);
29
30      miaux_define_hair_object(
31          p->name, hair_geo_row_bbox, p, result, hair_geo_row_callback);
32
33      return miTRUE;
34  }
```

Source code of main shader of "hair_geo_row"

Modeling hair

Multiple hairs

```
 1  miBoolean hair_geo_row_callback(miTag tag, void *ptr)
 2  {
 3      miHair_list *hair_list;
 4      miGeoIndex *harray;
 5      miScalar *hair_scalars;
 6      int i, h, per_hair_data_count, vertex_count, vertex_size,
 7          per_hair_scalar_count, total_scalar_count;
 8      hair_geo_row_t *p = (hair_geo_row_t *)ptr;
 9      miScalar x, y_offset, z_offset,
10          yoff_max = p->y_offset_max, zoff_max = p->z_offset_max;
11
12      mi_api_incremental(miTRUE);
13      miaux_define_hair_object(p->name, hair_geo_row_bbox, p, NULL, NULL);
14
15      hair_list = mi_api_hair_begin();
16      hair_list->approx = p->approximation;
17      hair_list->degree = p->degree;
18      mi_api_hair_info(0, 'r', 1);
19      mi_api_hair_info(0, 't', 1);
20
21      per_hair_data_count = 2;  /* Radius and random value */
22      vertex_count = 4;
23      vertex_size = 3;
24      per_hair_scalar_count =
25          vertex_count * vertex_size + per_hair_data_count;
26      total_scalar_count = p->count * per_hair_scalar_count;
27
28      hair_scalars = mi_api_hair_scalars_begin(total_scalar_count);
29      mi_srandom(p->random_seed);
30
31      for (i = 0; i < p->count; i++) {
32          x = miaux_random_range(p->bbox_min.x, p->bbox_max.x);
33          y_offset = miaux_random_range(-yoff_max, yoff_max);
34          z_offset = miaux_random_range(-zoff_max, zoff_max);
35
36          *hair_scalars++ = p->radius;
37          *hair_scalars++ = miaux_random_range(0.0, 1.0);
38          *hair_scalars++ = x;
39          *hair_scalars++ = p->bbox_min.y + yoff_max + y_offset;
40          *hair_scalars++ = p->bbox_min.z + zoff_max + z_offset;
41          *hair_scalars++ = x;
42          *hair_scalars++ = p->bbox_max.y - yoff_max + y_offset;
43          *hair_scalars++ = p->bbox_min.z + zoff_max + z_offset;
44          *hair_scalars++ = x;
45          *hair_scalars++ = p->bbox_max.y - yoff_max + y_offset;
46          *hair_scalars++ = p->bbox_max.z - zoff_max + z_offset;
47          *hair_scalars++ = x;
48          *hair_scalars++ = p->bbox_min.y + yoff_max + y_offset;
49          *hair_scalars++ = p->bbox_max.z - zoff_max + z_offset;
50      }
51      mi_api_hair_scalars_end(total_scalar_count);
52
53      harray = mi_api_hair_hairs_begin(p->count + 1);
54      for (h=0; h < p->count + 1; h++)
55          harray[h] = h * per_hair_scalar_count;
56      mi_api_hair_hairs_end();
57
58      mi_api_hair_end();
59      mi_api_object_end();
60      return miTRUE;
61  }
```

Source code of shader "hair_geo_row"

Modeling hair

A basic lighting model for hair

```
declare shader
    color "hair_color_light" (
        color "ambient"   default 0 0 0,
        color "diffuse"   default 1 1 1,
        color "specular"  default 0 0 0,
        scalar "exponent" default 30,
        scalar "root_opacity" default 1,
        scalar "tip_opacity"  default 0,
        scalar "color_variance" default .1,
        array light "lights" )
end declare
```

Scene file declaration of shader "hair_color_light"

Modeling hair

A basic lighting model for hair

```
1  void hair_color_variance(
2      miColor *result, miState *state, struct hair_color_light* params)
3  {
4      miScalar maximum_variance = *mi_eval_scalar(&params->color_variance);
5      miScalar color_variance =
6          miaux_fit(state->tex_list[0].x, 0.0, 1.0,
7                    1.0 - maximum_variance, 1.0 + maximum_variance);
8      *result = *mi_eval_color(&params->diffuse);
9      miaux_scale_color(result, color_variance);
10     miaux_clamp_color(result);
11 }
```

Source code of shader "hair_color_light"

Modeling hair

A basic lighting model for hair

```
1  miScalar hair_alpha(miState *state, struct hair_color_light* params)
2  {
3      return miaux_fit(state->bary[1], 0.0, 1.0,
4                       *mi_eval_scalar(&params->root_opacity),
5                       *mi_eval_scalar(&params->tip_opacity));
6  }
```

Source code of shader "hair_color_light"

Modeling hair

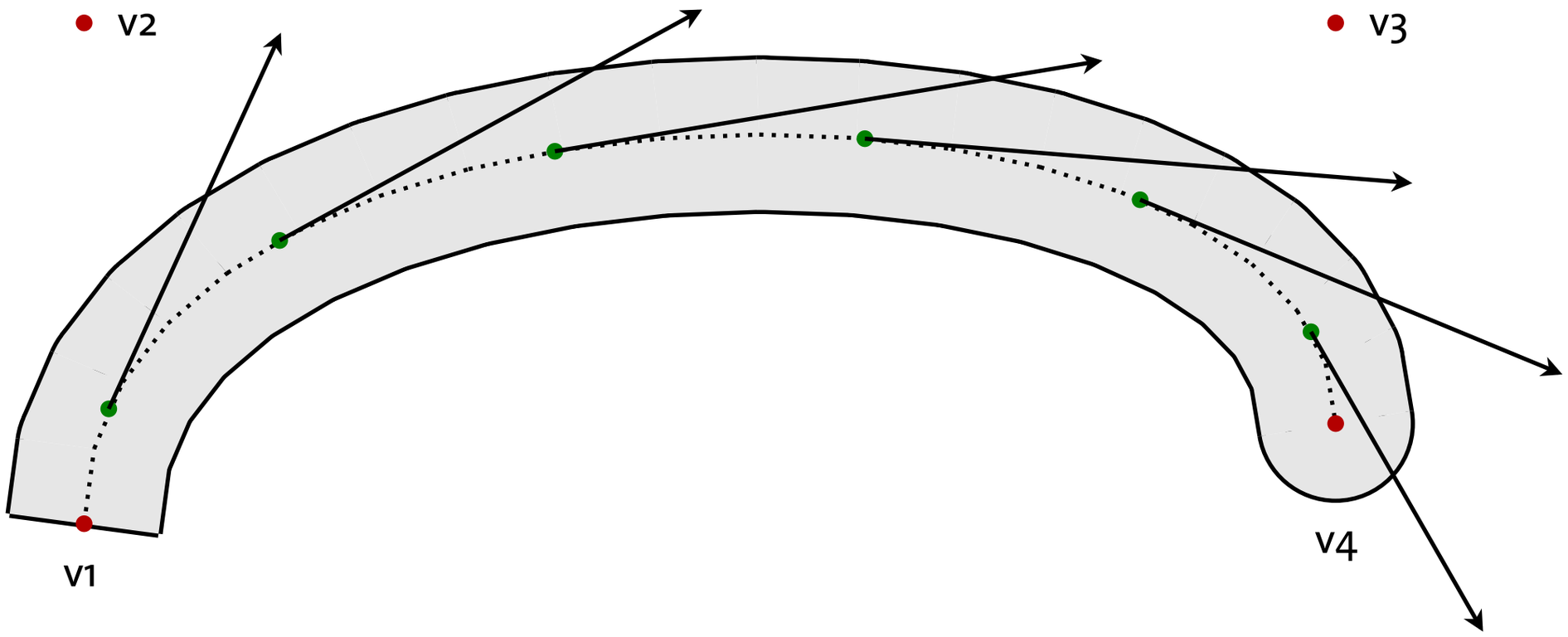
A basic lighting model for hair

```
 1  miBoolean hair_shadow(miColor *result, miColor *diffuse, miScalar alpha)
 2  {
 3      miScalar threshold = 0.001;
 4      miScalar transparency = 1.0 - alpha;
 5      result->r *= miaux_shadow_breakpoint(diffuse->r, transparency, 0.2);
 6      result->g *= miaux_shadow_breakpoint(diffuse->g, transparency, 0.2);
 7      result->b *= miaux_shadow_breakpoint(diffuse->b, transparency, 0.2);
 8      if (result->r < threshold &&
 9          result->g < threshold &&
10          result->b < threshold)
11          return miFALSE;
12      else
13          return miTRUE;
14  }
```

Source code of shader "hair_color_light"

Modeling hair

A basic lighting model for hair



Tangent to the hair curve at each point available as code{state->derivs[o]}

Modeling hair

A basic lighting model for hair

```
1  void miaux_add_diffuse_hair_component(
2      miColor *result, miVector *hair_tangent, miVector *to_light,
3      miColor *diffuse, miColor *light_color)
4  {
5      miScalar diffuse_factor = 1.0 - fabs(mi_vector_dot(hair_tangent, to_light));
6      miaux_add_diffuse_component(result, diffuse_factor, diffuse, light_color);
7  }
```

Auxiliary function: miaux_add_diffuse_hair_component

```
1  void miaux_add_specular_hair_component(
2      miColor *result, miVector *hair_tangent, miVector *to_light,
3      miVector *to_camera,
4      miColor *specular, miColor *light_color)
5  {
6      miScalar light_angle = acos(mi_vector_dot(hair_tangent, to_light));
7      miScalar view_angle = acos(mi_vector_dot(hair_tangent, to_camera));
8      miScalar sum = light_angle + view_angle;
9      miScalar specular_factor = fabs(M_PI_2 - fmod(sum, M_PI)) / M_PI_2;
10
11     result->r += specular_factor * specular->r * light_color->r;
12     result->g += specular_factor * specular->g * light_color->g;
13     result->b += specular_factor * specular->b * light_color->b;
14 }
```

Auxiliary function: miaux_add_specular_hair_component

```
1  void miaux_add_transparent_hair_component(miColor *result, miState *state)
2  {
3      miColor background_color;
4      mi_trace_transparent(&background_color, state);
5      if (result->a == 0) {
6          miaux_opacity_set_channels(state, 0.0);
7          miaux_copy_color(result, &background_color);
8      } else {
9          miaux_opacity_set_channels(state, result->a);
10         miaux_blend_colors(result, result, &background_color, result->a);
11     }
12 }
```

Auxiliary function: miaux_add_transparent_hair_component

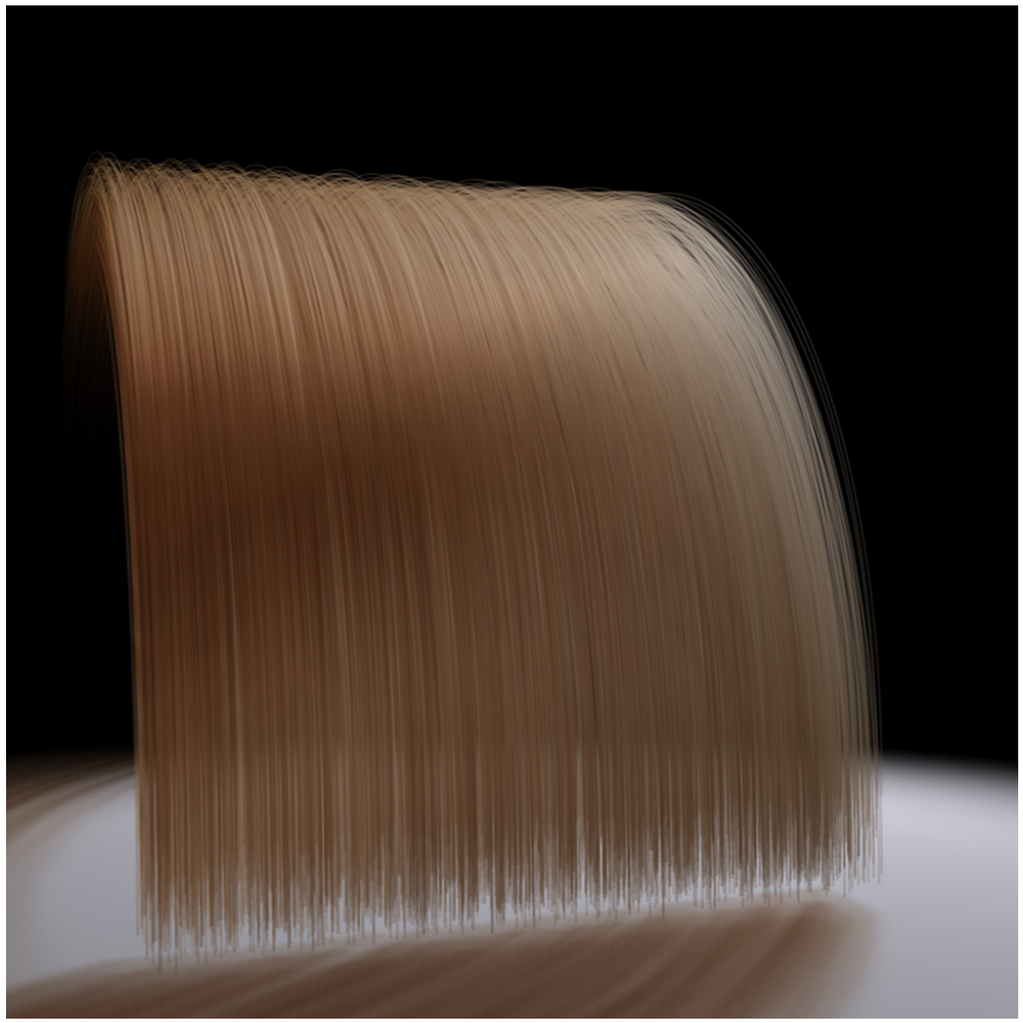
Modeling hair

A basic lighting model for hair

```
 1  struct hair_color_light {
 2      miColor  ambient;
 3      miColor  diffuse;
 4      miColor  specular;
 5      miScalar exponent;
 6      miScalar root_opacity;
 7      miScalar tip_opacity;
 8      miScalar color_variance;
 9      int      i_light;
10      int      n_light;
11      miTag    light[1];
12  };
13
14  miBoolean hair_color_light (
15      miColor *result, miState *state, struct hair_color_light *params )
16  {
17      int i, light_count, light_sample_count;
18      miColor diffuse, sum, light_color, *specular;
19      miVector direction_toward_light, to_camera;
20      miScalar dot_nl, alpha;
21      miTag *light;
22      miVector hair_tangent = state->derivs[0];
23      miVector original_normal = state->normal;
24
25      hair_color_variance(&diffuse, state, params);
26      alpha = hair_alpha(state, params);
27
28      if (state->type == miRAY_SHADOW)
29          return hair_shadow(result, &diffuse, alpha);
30
31      state->normal.x = state->normal.y = state->normal.z = 0.0f;
32      to_camera = state->dir;
33      mi_vector_neg(&to_camera);
34      mi_vector_normalize(&hair_tangent);
35
36      specular = mi_eval_color(&params->specular);
37      *result = *mi_eval_color(&params->ambient);
38      result->a = alpha;
39
40      miaux_light_array(&light, &light_count, state,
41                        &params->i_light, &params->n_light, params->light);
42
43      for (i = 0; i < light_count; i++, light++) {
44          miaux_set_channels(&sum, 0);
45          light_sample_count = 0;
46          while (mi_sample_light(&light_color, &direction_toward_light, &dot_nl,
47                                 state, *light, &light_sample_count)) {
48              miaux_add_diffuse_hair_component(
49                  &sum, &hair_tangent, &direction_toward_light,
50                  &diffuse, &light_color);
51              miaux_add_specular_hair_component(
52                  &sum, &hair_tangent,  &direction_toward_light, &to_camera,
53                  specular, &light_color);
54          }
55          if (light_sample_count)
56              miaux_add_scaled_color(result, &sum, 1.0/light_sample_count);
57      }
58      state->normal = original_normal;
59      if (result->a < 0.999)
60          miaux_add_transparent_hair_component(result, state);
61      return miTRUE;
62  }
```

Source code of main shader of "hair_color_light"

Modeling hair



A basic lighting model for hair

```
options "opt"
    object space
    contrast .1 .1 .1 1
    scanline rapid
    shadow on
    shadowmap detail
    samples collect 5
    filter mitchell
end options

material "hair_light"
    "hair_color_light" (
        "lights" ["light-inst"],
        "diffuse" .55 .3 .05,
        "specular" .65 .65 .65,
        "root_opacity" .2,
        "tip_opacity" .2,
        "exponent" 80 )
    shadow "hair_color_light" (
        "diffuse" .55 .3 .05,
        "root_opacity" .05,
        "tip_opacity" .05, )
end material

instance "hair-instance"
    geometry
        "hair_geo_row" (
            "name" "::hair-1",
            "radius" .0025,
            "count" 4000,
            "bbox_min" -.7 -.5 -2.5,
            "bbox_max"  .7 1.5 0,
            "y_offset_max" .1,
            "z_offset_max" .1, )
    material "hair_light"
end instance
```

Defining other rendering modes in the options block for hair rendering

Modeling hair

The hair primitive as a general modeling tool

```
material "hair_light"
    "hair_color_light" (
        "lights" ["light-inst"],
        "diffuse" .4 .5 .3,
        "specular" .6 .6 .3,
        "root_opacity" 1,
        "tip_opacity" .9, )
    shadow "hair_color_light" (
        "diffuse" .45 .5 .4,
        "root_opacity" 1,
        "tip_opacity" .5 )
end material

instance "hair-instance"
    geometry
        "hair_geo_curl" (
            "name" "::hair",
            "count" 200,
            "root_radius" .04,
            "tip_radius" .0001,
            "center" 0 .3 0,
            "length_min" .6,
            "length_max"  .85,
            "spiral_turns_min" .5,
            "spiral_turns_max" 1.5,
            "spiral_radius_min" .02,
            "spiral_radius_max" .03 )
    material "hair_light"
end instance
```

Plant-like shapes using the hair primitive

```
declare shader
    geometry "hair_geo_curl" (
        string "name",
        integer "count" default 1,
        scalar "root_radius" default .01,
        scalar "tip_radius" default .0001,
        vector "center" default 0 0 0,
        scalar "length_min" default .5,
        scalar "length_max" default 1,
        scalar "spiral_turns_min" default 2,
        scalar "spiral_turns_max" default 3,
        scalar "spiral_radius_min" default .01,
        scalar "spiral_radius_max" default .03,
        scalar "segment_length" default .05,
        integer "random_seed" default 1955 )
end declare
```

Scene file declaration of shader "hair_geo_curl"

```
 1  void hair_geo_curl_bbox(miObject *obj, void* params)
 2  {
 3     hair_geo_curl_t *p = (hair_geo_curl_t*)params;
 4     obj->bbox_min.x = p->center.x - p->length_max;
 5     obj->bbox_min.y = p->center.y - p->length_max;
 6     obj->bbox_min.z = p->center.z - p->length_max;
 7     obj->bbox_max.x = p->center.x + p->length_max;
 8     obj->bbox_max.y = p->center.y + p->length_max;
 9     obj->bbox_max.z = p->center.z + p->length_max;
10  }
```

Source code of shader "hair_geo_curl"

Modeling hair

The hair primitive as a general modeling tool

```
 1  typedef struct {
 2      miTag name;
 3      miInteger count;
 4      miScalar root_radius;
 5      miScalar tip_radius;
 6      miVector center;
 7      miScalar length_min;
 8      miScalar length_max;
 9      miScalar spiral_turns_min;
10      miScalar spiral_turns_max;
11      miScalar spiral_radius_min;
12      miScalar spiral_radius_max;
13      miScalar segment_length;
14      miInteger random_seed;
15  } hair_geo_curl_t;
16
17  miBoolean hair_geo_curl (
18      miTag *result, miState *state, hair_geo_curl_t *params )
19  {
20      hair_geo_curl_t *p =
21          (hair_geo_curl_t*)mi_mem_allocate(sizeof(hair_geo_curl_t));
22      p->name               = *mi_eval_tag(&params->name);
23      p->count              = *mi_eval_integer(&params->count);
24      p->root_radius        = *mi_eval_scalar(&params->root_radius);
25      p->tip_radius         = *mi_eval_scalar(&params->tip_radius);
26      p->center             = *mi_eval_vector(&params->center);
27      p->length_min         = *mi_eval_scalar(&params->length_min);
28      p->length_max         = *mi_eval_scalar(&params->length_max);
29      p->spiral_turns_min   = *mi_eval_scalar(&params->spiral_turns_min);
30      p->spiral_turns_max   = *mi_eval_scalar(&params->spiral_turns_max);
31      p->spiral_radius_min = *mi_eval_scalar(&params->spiral_radius_min);
32      p->spiral_radius_max = *mi_eval_scalar(&params->spiral_radius_max);
33      p->segment_length     = *mi_eval_scalar(&params->segment_length);
34      p->random_seed        = *mi_eval_integer(&params->random_seed);
35
36      miaux_define_hair_object(
37          p->name, hair_geo_curl_bbox, p, result, hair_geo_curl_callback);
38
39      return miTRUE;
40  }
```

Source code of main shader of "hair_geo_curl"

```
1  void miaux_random_point_on_sphere(
2      miVector *result, miVector *center, miScalar radius)
3  {
4      miMatrix transform;
5      result->x = radius;
6      result->y = result->z = 0.0;
7      mi_matrix_rotate(transform, 0,
8                       miaux_random_range(0, M_PI * 2),
9                       miaux_random_range(0, M_PI * 2));
10     mi_vector_transform(result, result, transform);
11     mi_vector_add(result, result, center);
12 }
```

Auxiliary function: miaux_random_point_on_sphere

```
1  typedef struct {
2      miVector hair_end;
3      int vertex_count;
4      miScalar turns;
5      miScalar spiral_radius;
6  } hair_spec_t;
```

Source code of struct "hair_spec_t"

```
 1  int create_hair_specifications(
 2      hair_spec_t **hair_specs, hair_geo_curl_t *p)
 3  {
 4      float pi_2 = M_PI * 2.0;
 5      int scalars_per_vertex = 4, total_scalar_count = 0, i;
 6      *hair_specs =
 7          (hair_spec_t*)mi_mem_allocate(p->count * sizeof(hair_spec_t));
 8      for (i = 0; i < p->count; i++) {
 9          miScalar distance, length;
10          hair_spec_t *spec = &((*hair_specs)[i]);
11          spec->turns =
12              miaux_random_range(p->spiral_turns_min, p->spiral_turns_max);
13          spec->spiral_radius =
14              miaux_random_range(p->spiral_radius_min, p->spiral_radius_max);
15          distance = miaux_random_range(p->length_min, p->length_max);
16          length = distance + pi_2 * spec->spiral_radius * spec->turns;
17          miaux_random_point_on_sphere(
18              &(spec->hair_end), &p->center, distance);
19          spec->vertex_count = (int)(ceil(length / p->segment_length));
20          total_scalar_count += spec->vertex_count * scalars_per_vertex;
21      }
22      return total_scalar_count;
23  }
```

Source code of shader "hair_geo_curl"

```
1  void miaux_perpendicular_point(miVector *result, miVector *v, float distance)
2  {
3      result->x = -v->y;
4      result->y = v->x;
5      result->z = 0;
6      mi_vector_normalize(result);
7      mi_vector_mul(result, distance);
8  }
```

Auxiliary function: miaux_perpendicular_point

```
1  void miaux_point_between(
2      miVector *result, miVector *u, miVector *v, float fraction)
3  {
4      result->x = miaux_fit(fraction, 0, 1, u->x, v->x);
5      result->y = miaux_fit(fraction, 0, 1, u->y, v->y);
6      result->z = miaux_fit(fraction, 0, 1, u->z, v->z);
7  }
```

Auxiliary function: miaux_point_between

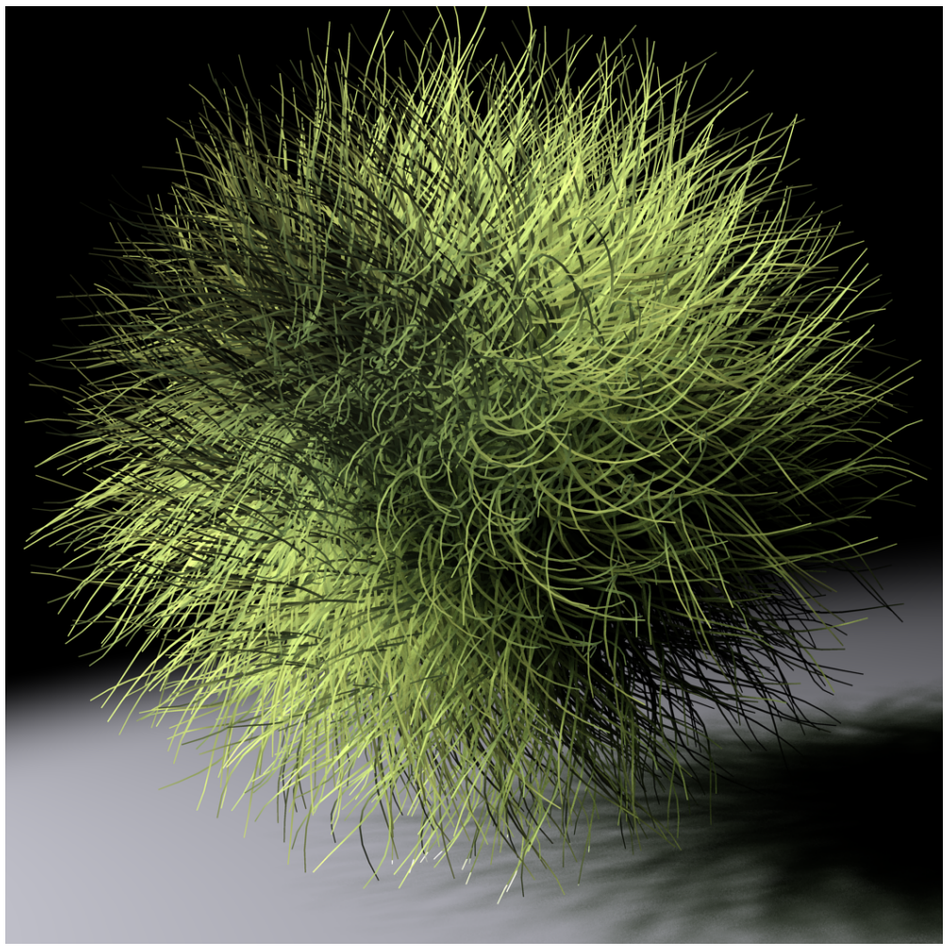
```
 1  void miaux_hair_spiral(
 2      miScalar** scalar_array, miVector *start_point, miVector *end_point,
 3      float turns, int point_count, float angle_offset, float spiral_radius,
 4      miScalar root_radius, miScalar tip_radius)
 5  {
 6      miVector base_point, spiral_axis, spiral_point;
 7      miVector axis_point;
 8      miMatrix matrix;
 9      float angle, pi_2 = 2 * M_PI, max_index = point_count - 1;
10      int i;
11
12      mi_vector_sub(&spiral_axis, end_point, start_point);
13      mi_vector_normalize(&spiral_axis);
14      miaux_perpendicular_point(&axis_point, &spiral_axis, spiral_radius);
15
16      for (i = 0; i < point_count; i++) {
17          float fraction = i / max_index;
18          miaux_point_between(&base_point, start_point, end_point, fraction);
19          angle = angle_offset + fraction * turns * pi_2;
20          mi_matrix_rotate_axis(matrix, &spiral_axis, angle);
21          mi_point_transform(&spiral_point, &axis_point, matrix);
22          mi_vector_add(&spiral_point, &spiral_point, &base_point);
23
24          *(*scalar_array)++ = spiral_point.x;
25          *(*scalar_array)++ = spiral_point.y;
26          *(*scalar_array)++ = spiral_point.z;
27          *(*scalar_array)++ =
28              miaux_fit_clamp(i, 0, max_index, root_radius, tip_radius);
29      }
30  }
```

Auxiliary function: miaux_hair_spiral

```
 1  miBoolean hair_geo_curl_callback(miTag tag, void *ptr)
 2  {
 3      miHair_list *hair;
 4      miGeoIndex  *harray;
 5      hair_geo_curl_t *p = (hair_geo_curl_t*)ptr;
 6      hair_spec_t *hair_specs;
 7      int i, total_scalar_count, hair_array_position;
 8      miScalar *hair_scalars;
 9
10      mi_srandom(p->random_seed);
11      mi_api_incremental(miTRUE);
12      miaux_define_hair_object(p->name, hair_geo_curl_bbox, p, NULL, NULL);
13      hair = mi_api_hair_begin();
14      hair->approx = hair->degree = 1;
15      mi_api_hair_info(1, 'r', 1);
16
17      total_scalar_count = create_hair_specifications(&hair_specs, p);
18      hair_scalars = mi_api_hair_scalars_begin(total_scalar_count);
19
20      for (i = 0; i < p->count; i++) {
21          float angle_offset = miaux_random_range(0, M_PI * 2);
22          miaux_hair_spiral(
23              &hair_scalars, &p->center, &hair_specs[i].hair_end,
24              hair_specs[i].turns, hair_specs[i].vertex_count, angle_offset,
25              hair_specs[i].spiral_radius, p->root_radius, p->tip_radius);
26      }
27      mi_api_hair_scalars_end(total_scalar_count);
28
29      harray = mi_api_hair_hairs_begin(p->count + 1);
30      hair_array_position = 0;
31      for (i = 0; i < p->count + 1; i++) {
32          harray[i] = hair_array_position;
33          if (i < p->count)
34              hair_array_position += hair_specs[i].vertex_count * 4;
35      }
36      mi_api_hair_hairs_end();
37
38      mi_api_hair_end();
39      mi_api_object_end();
40      mi_mem_release(hair_specs);
41
42      return miTRUE;
43  }
```

Source code of shader "hair_geo_curl"

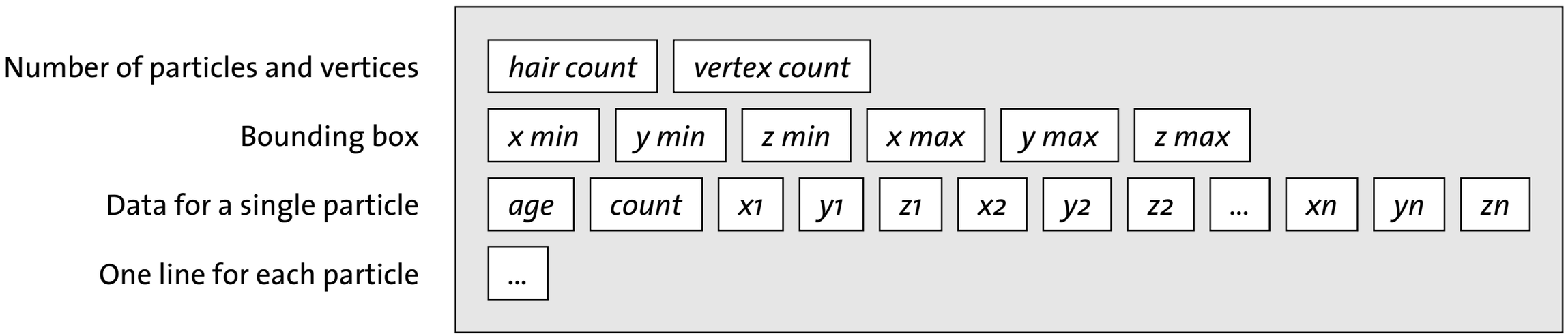
Modeling hair



The hair primitive as a general modeling tool

```
options "opt"
    object space
    contrast .1 .1 .1 1
    scanline rapid
    shadow on
    shadowmap detail
    samples collect 5
    filter mitchell 4
end options

material "hair_light"
    "hair_color_light" (
        "lights" ["light-inst"],
        "diffuse" .4 .5 .3,
        "specular" .6 .6 .3,
        "root_opacity" 1,
        "tip_opacity" .9, )
    shadow "hair_color_light" (
        "diffuse" .45 .5 .4,
        "root_opacity" 1,
        "tip_opacity" .5 )
end material

instance "hair-instance"
    geometry
        "hair_geo_curl" (
            "name" "::hair",
            "count" 4000,
            "center" 0 .3 0,
            "root_radius" .006,
            "tip_radius" .001,
            "length_min" .6,
            "length_max"  .85,
            "spiral_turns_min" 1,
            "spiral_turns_max" 2,
            "spiral_radius_min" .04,
            "spiral_radius_max" .06 )
    material "hair_light"
end instance
```

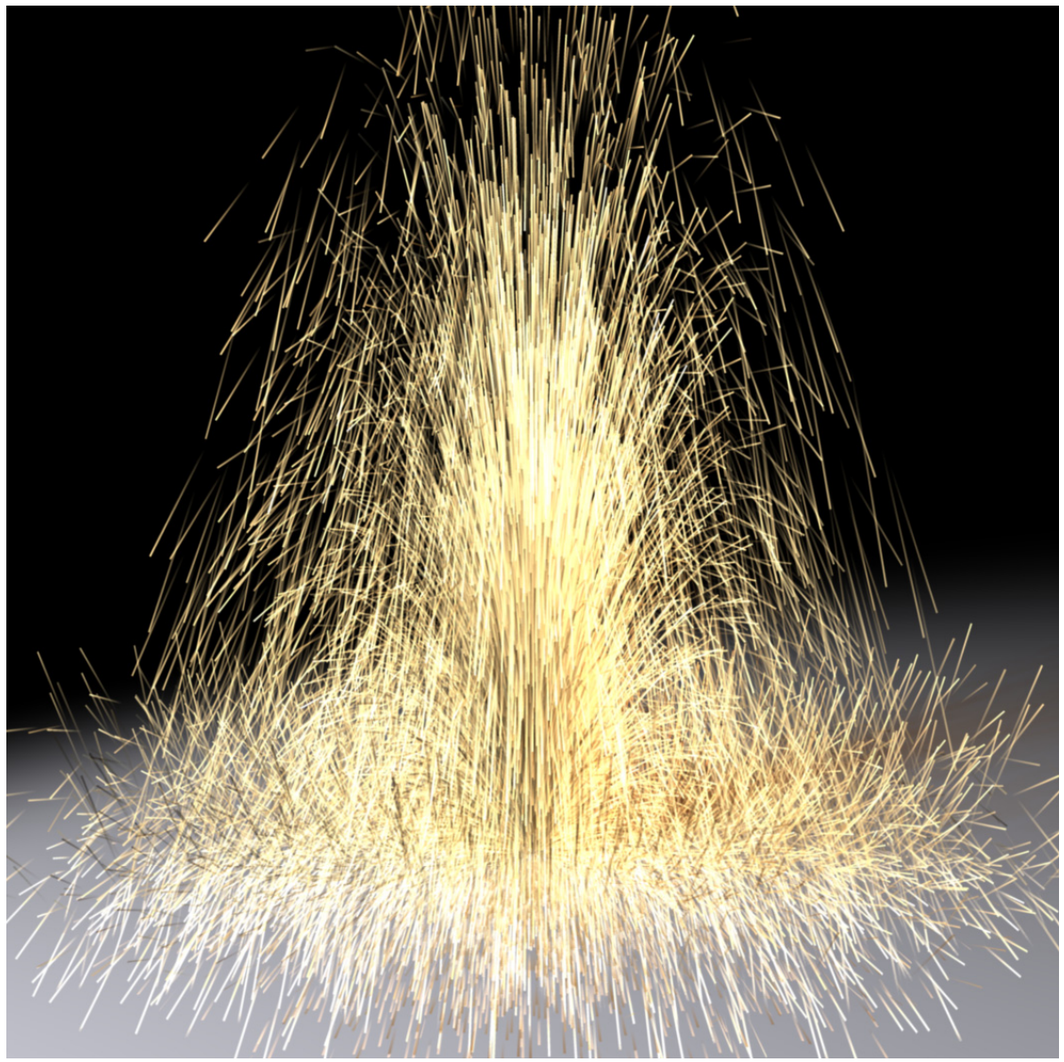
Increasing the number of hairs in shader hair_geo_curl



File format for particle system data as input to hair geometry shader

Modeling hair

The hair primitive and particle systems



```
instance "explosion-instance"
    geometry "hair_geo_datafile" (
        "radius" .001,
        "particle_filename" "explosion.hpd" )
    material "hair_light"
    transform
        .5 0 0 0
        0 .5 0 0
        0 0 .5 0
        0 .25 .4 1
end instance
```

Hair used as the geometric primitive to render a particle system

```
declare shader
    geometry "hair_geo_datafile" (
        string "name",
        scalar "radius" default .1,
        string "particle_filename" )
end declare
```

Scene file declaration of shader "hair_geo_datafile"

```
1  void miaux_hair_data_file_bounding_box(
2      char* filename,
3      float *xmin, float *ymin, float *zmin,
4      float *xmax, float *ymax, float *zmax)
5  {
6      int hair_count, data_count;
7      FILE* fp = fopen(filename, "r");
8      fscanf(fp, "%d %d ", &hair_count, &data_count); /* Ignore. */
9      fscanf(fp, "%f %f %f %f %f %f ", xmin, ymin, zmin, xmax, ymax, zmax);
10     fclose(fp);
11 }
```

Auxiliary function: miaux_hair_data_file_bounding_box

```
 1  void hair_geo_datafile_bbox(miObject *obj, void* params)
 2  {
 3     hair_geo_datafile_t *hairdata = (hair_geo_datafile_t*)params;
 4     char* particle_filename;
 5     particle_filename =
 6         miaux_tag_to_string(hairdata->particle_filename, NULL);
 7     if (particle_filename == NULL)
 8         mi_fatal("Particle filename required for hair_geo_datafile.");
 9     miaux_hair_data_file_bounding_box(
10         particle_filename,
11         &obj->bbox_min.x, &obj->bbox_min.y, &obj->bbox_min.z,
12         &obj->bbox_max.x, &obj->bbox_max.y, &obj->bbox_max.z);
13  }
```

Source code of shader "hair_geo_datafile"

```
 1  typedef struct {
 2      miTag name;
 3      miScalar radius;
 4      miTag particle_filename;
 5  } hair_geo_datafile_t;
 6
 7  miBoolean hair_geo_datafile (
 8      miTag *result, miState *state, hair_geo_datafile_t *params )
 9  {
10      hair_geo_datafile_t *hairdata =
11          (hair_geo_datafile_t*)mi_mem_allocate(sizeof(hair_geo_datafile_t));
12      hairdata->radius = *mi_eval_scalar(&params->radius);
13      hairdata->particle_filename = *mi_eval_tag(&params->particle_filename);
14
15      miaux_define_hair_object(
16          hairdata->name, hair_geo_datafile_bbox, hairdata, result,
17          hair_geo_datafile_callback);
18
19      return miTRUE;
20  }
```

Source code of main shader of "hair_geo_datafile"

```
 1  void miaux_read_hair_data_file(char* filename, miScalar radius)
 2  {
 3      int vertex_count, total_vertex_count, hair_scalar_size, vertex_total = 0,
 4          index_array_size, v, *hair_indices, *hi, hair_count, per_hair_scalars;
 5      float xmin, ymin, zmin, xmax, ymax, zmax, age;
 6      miScalar coord, *hair_scalars;
 7      miGeoIndex *harray;
 8      FILE *fp;
 9
10      fp = fopen(filename, "r");
11      fscanf(fp, "%d %d ", &hair_count, &total_vertex_count);
12      fscanf(fp, "%f %f %f %f %f %f ",
13             &xmin, &ymin, &zmin, &xmax, &ymax, &zmax);
14      mi_progress("particle bounding box: %f %f %f %f %f %f ",
15                  xmin, ymin, zmin, xmax, ymax, zmax);
16
17      per_hair_scalars = 2;
18      mi_api_hair_info(0, 'r', 1);
19      mi_api_hair_info(0, 't', 1);
20
21      hair_scalar_size = hair_count * per_hair_scalars + total_vertex_count * 3;
22      hair_scalars = mi_api_hair_scalars_begin(hair_scalar_size);
23
24      index_array_size = 1 + hair_count;
25      hi = hair_indices = (int*)mi_mem_allocate(sizeof(int) * index_array_size);
26      *hi++ = 0;
27      vertex_total = 0;
28
29      while (!feof(fp)) {
30          *hair_scalars++ = radius;
31          fscanf(fp, "%f ", &age);
32          *hair_scalars++ = age;
33          fscanf(fp, "%d ", &vertex_count);
34          for (v = 0; v < vertex_count * 3; v++) {
35              fscanf(fp, "%f ", &coord);
36              *hair_scalars++ = coord;
37          }
38          vertex_total += vertex_count * 3 + per_hair_scalars;
39          *hi++ = vertex_total;
40      }
41      mi_api_hair_scalars_end(hair_scalar_size);
42      harray = mi_api_hair_hairs_begin(index_array_size);
43      memcpy(harray, hair_indices, index_array_size * sizeof(int));
44      mi_api_hair_hairs_end();
45  }
```

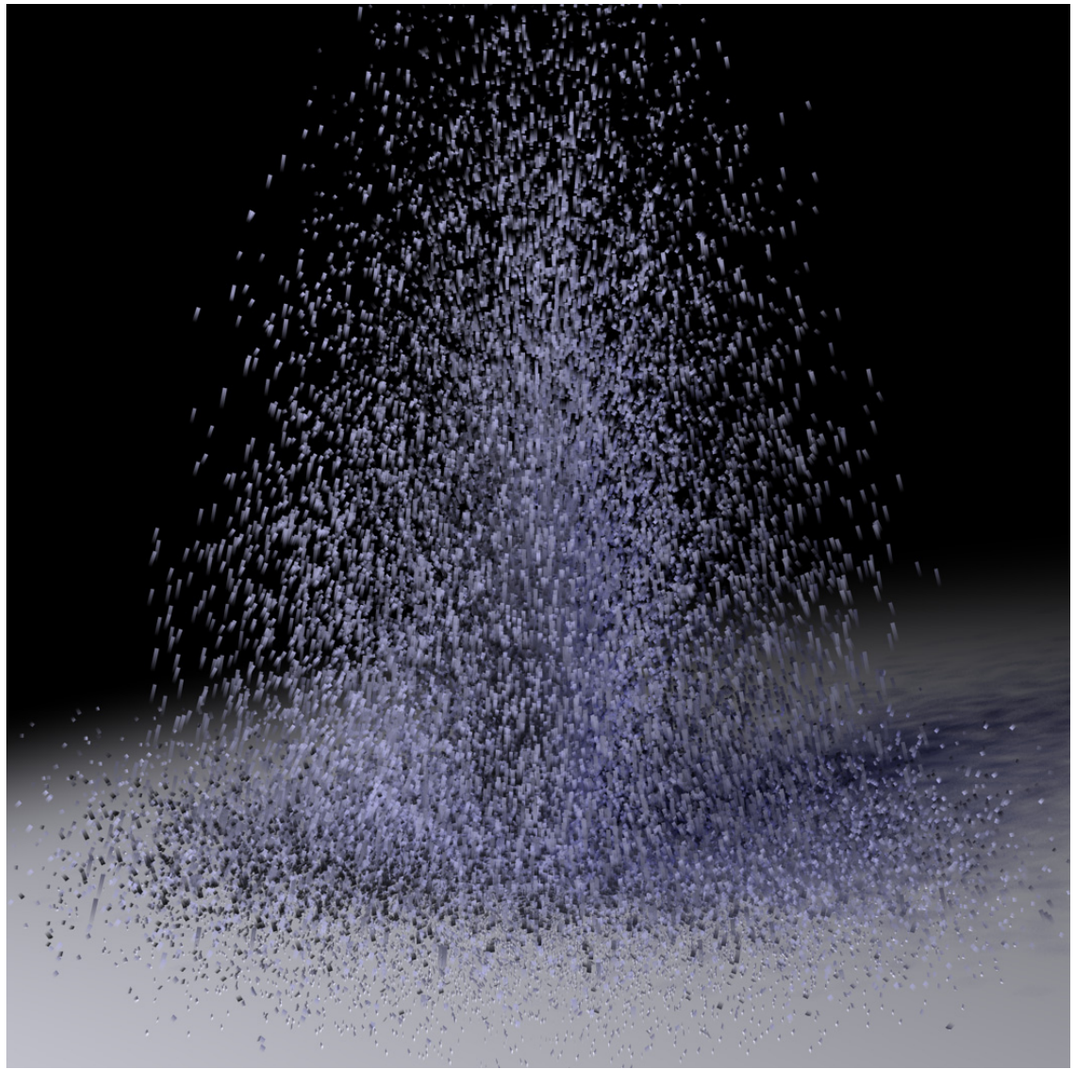
Auxiliary function: miaux_read_hair_data_file

```
 1  miBoolean hair_geo_datafile_callback(miTag tag, void *ptr)
 2  {
 3      miHair_list *hair;
 4      hair_geo_datafile_t *hairdata = (hair_geo_datafile_t *)ptr;
 5
 6      mi_api_incremental(miTRUE);
 7      miaux_define_hair_object(
 8          hairdata->name, hair_geo_datafile_bbox, hairdata, NULL, NULL);
 9      hair = mi_api_hair_begin();
10      hair->approx = hair->degree = 1;
11
12      miaux_read_hair_data_file(
13          miaux_tag_to_string(hairdata->particle_filename, NULL),
14          hairdata->radius);
15      mi_api_hair_end();
16      mi_api_object_end();
17      return miTRUE;
18  }
```

Source code of shader "hair_geo_datafile"

Modeling hair

The hair primitive and particle systems



```
options "opt"
    object space
    contrast .1 .1 .1 1
    scanline rapid
    shadow on
    shadowmap detail
    samples collect 5
    filter mitchell 4
end options

material "hair_light"
    "hair_color_light" (
        "lights" ["light-inst"],
        "diffuse"  .6 .6 .8,
        "specular" .2 .2 .2,
        "root_opacity" 1,
        "tip_opacity" 0 )
    shadow "hair_color_light" (
        "diffuse"  .6 .6 .8,
        "root_opacity" 1,
        "tip_opacity" 0 )
end material

instance "fountain-instance"
    geometry "hair_geo_datafile" (
        "radius" .002,
        "particle_filename" "fountain.hpd" )
    material "hair_light"
    transform
        .5 0 0 0
        0 .5 0 0
        0 0 .5 0
        0 .25 .4 1
end instance
```

Simulating water droplets with shorter hair lengths

```
declare shader
    color "hair_color_fire" (
        color "transparency" default 0.5 0.5 0.5 )
end declare
```

Scene file declaration of shader "hair_color_fire"

```
1  void miaux_color_fit(miColor *result,
2                       miScalar f, miScalar start, miScalar end,
3                       miColor *start_color, miColor *end_color)
4  {
5      result->r = miaux_fit(f, start, end, start_color->r, end_color->r);
6      result->g = miaux_fit(f, start, end, start_color->g, end_color->g);
7      result->b = miaux_fit(f, start, end, start_color->b, end_color->b);
8  }
```

Auxiliary function: miaux_color_fit

```
1  void miaux_blend_transparency(miColor *result,
2                                miState *state, miColor *transparency)
3  {
4      miColor opacity, background;
5      miaux_invert_channels(&opacity, transparency);
6      mi_opacity_set(state, &opacity);
7      if (!miaux_all_channels_equal(transparency, 1.0)) {
8          mi_trace_transparent(&background, state);
9          miaux_blend_channels(result, &background, &opacity);
10     }
11 }
```

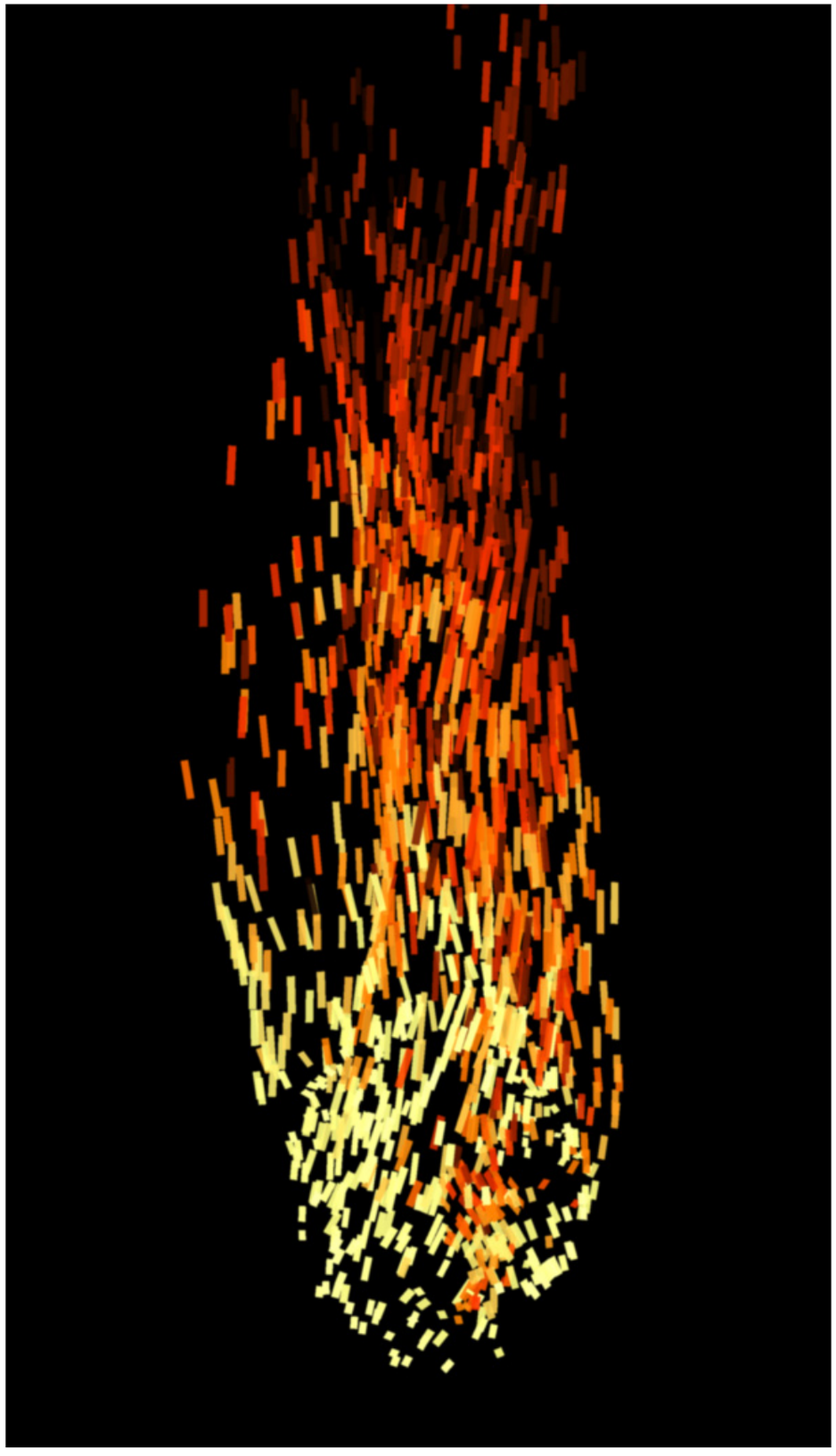
Auxiliary function: miaux_blend_transparency

```
 1  struct hair_color_fire {
 2      miColor transparency;
 3  };
 4
 5  miBoolean hair_color_fire (
 6      miColor *result, miState *state, struct hair_color_fire *params )
 7  {
 8      miScalar keys[6] = {0.0, .2, .4, .6, .9, 1.0};
 9      miColor colors[6] = {{1.2, 1.2, .7},
10                           {1, 1, .5},
11                           {1, .5, 0},
12                           {1, .2, 0},
13                           {.4, .1, 0},
14                           {0, 0, 0}};
15      miColor *transparency = mi_eval_color(&params->transparency);
16      miScalar age = state->tex_list[0].x;
17      int i;
18      for (i = 4; i >=0; i--) {
19          if (age >= keys[i]) {
20              miaux_color_fit(result, age,
21                              keys[i], keys[i+1], &colors[i], &colors[i+1]);
22              break;
23          }
24      }
25      miaux_blend_transparency(result, state, transparency);
26
27      return miTRUE;
28  }
```

Source code of shader "hair_color_fire"

Modeling hair

Particle system data and dynamic rendering effects



```
options "opt"
    object space
    contrast .1 .1 .1 1
    scanline rapid
    samples collect 3
    filter gauss 3 3
    dither off
end options

material "spark"
    "hair_color_fire" (
        "transparency" .1 .1 .1 )
end material

instance "fire-instance"
    geometry "hair_geo_datafile" (
        "radius" .03,
        "particle_filename" "fire.0055.hpd" )
    material "spark"
end instance
```

Modifying hair particle colors based on the age of the particle

Exercise 18: Geometry shaders

1. Copy `geometry_5.mi` to `geometry.mi` and change the output file to `geometry.tif`
2. In `geometry.mi`, change the ambient occlusion samples to 10 and its cutoff to 5, render and view.
3. Change the location and number of the object instances by changing the `positions` parameter.
4. Modify shader `instanced_object_file` to add random scaling of the instances.

Exercise 18: Geometry shaders (part 2)

Modify shader `instanced_object_file` to add random scaling of the instances.

Old:

```
mi_matrix_ident(matrix);
matrix[12] = positions->x;
matrix[13] = positions->y;
matrix[14] = positions->z;
```

New:

```
matrix[12] = positions->x;
matrix[13] = positions->y;
matrix[14] = positions->z;
matrix[0]  = .2 + mi_random() * 2;
matrix[5]  = .2 + mi_random() * 2;
matrix[10] = .2 + mi_random() * 2;
```

(Well...)