1 Tage

2 Tage datafile description

This file describes tage input files.

3 Comments

The input file uses standard C++ comments:

```
// One line comment

/*
  Two or more line comment
*/
```

4 Properties

All properties are set by

```
property_name = value
```

a "value" can be strings, numbers (hexa, integer, float-point), colors, vectors and enumerated values.

4.1 Boolean

Boolean is a binary value (true/false) and it's used for switches or on/off properties. The false value is written as 0 and true any other number (typically 1).

```
// light is enabled
enable_ligth = 1

// shadows are disabled
enable_shadows = 0
```

4.2 Numbers

Numbers are standard numerical values and can have a decimal part.

```
size = 10
height = 1.1
```

4.3 Strings

Strings don't use commas and can't contain spaces. String values are typically used for modificator/generator names.

```
name = my_modificator_name
```

4.4 Colors

Colors can be defined by three ways - by separated RGB values (0-255), by one hexadecimal digit (HTML color) or as a vector (R,G,B). There is an example of color_center set to R:33, G:25, B:7

```
// by RGB:
color_center_r = 33
color_center_g = 25
color_center_b = 7

// by one hexa number (RRGGBB)
color_center = 211907

// by vector (R,G,B)
color_center = (33,25,7)
```

See the _r,_g and _b suffixes. They are 0 by default.

4.5 Vectors

Vectors are composed from two or three numbers and they can be integer or floating point numbers. For instance, we want to set light_position vector:

```
light_position_x = -1
light_position_y =  1
light_position_z = -1
```

See the _x,_y and _z suffixes. They are 0 by default. Another option is to use a vector format (x,y,z):

```
light_position = (-1,1,-1)
```

4.6 Angles

An angles are normal numbers (an angle in degrees), from 0 to 360. They are used in polar coordinates and so on.

```
some_angle = 20.6
```

4.7 Enumerated types

Enumerated types are values which can have some predefined values. They are typically used for blocks type descriptions, some types, targets, operations and so on.

```
// coordinate type
type = MODIFICATOR_COORDINATE

// set modificator_target to texture
modificator_target = TEXTURE

// set modificator_target to geometry
modificator_target = GEOMETRY
```

4.7.1 Aritmetic operation

It's one of frequently applied enumerated types and defines requested arithmetics operation. It's used for coordinates, color/height operations and many more. Aritmetic operation anumerator is used in this context:

```
result = destination OP source
```

where OP is defined as:

SET	result = source
ADD	result = destination + source
SUB	result = destination - source
MODULATE	result = destination * source
MODULATE2X	result = destination * source * 2

4.8 Intervals

Some values can be set as interval. If a value is an interval, it means it can get any value from the border values. The border values are marked as "_min" and "_max" suffixes. Intervals are always used with other types (number, angle, color, vector). Intervals can be set as a normal (non-interval) value, too.

```
/* Number intervals
*/
// Interval set by only one value so it's always 10
angle = 10

// Interval set by two border values,
// can be any value from 10 to 20
angle_min = 10
angle_max = 20

/* Vector intervals
*/
// As components
position_min_x = 10
position_min_y = 10
position_min_z = 10

position_max_x = 20
position_max_y = 20
position_max_z = 20

// As vectors
position_min = (10,10,10)
position_max = (20,20,20)

/* Color intervals
*/
// As components
color_min_r = 10
color_min_g = 10
color_min_b = 10

color_max_r = 20
```

```
color_max_g = 20
color_max_b = 20

// As vectors
color_min = (10,10,10)
color_max = (20,20,20)

// As hexadecimal (HTML) colors
color_min = 0a0a0a
color_max = 141414
```

4.9 Coordinates

Coordinates are 2D area which describes where a modificator is applied. The coordinate is a whole block with "type = MODIFICATOR_COORDINATE", index (will be described later) and start and size (or end) 2D vectors. There is an example of area which starts at (0,0) and is 40x40 pixels wide:

```
{
  type = MODIFICATOR_COORDINATE

  index = 0

  start_x = 0
  start_y = 0

  size_x = 40
  size_y = 40
}
```

5 Basic blocks

An atomic part of the file is a block inside compound braces. It describes one atomic unit inside generator or some generator values. Each block must contain its name and type.

```
{
  name = generator
  type = GENERATOR_MESH

  /* All generator params come here
  */
}
```

Blocks can be nested, like this one:

```
/* Describes pixel generator and its color definition
*/
{
  name = pixel_point
  type = MODIFICATOR_POINT_SINGLE

  {
    type = MODIFICATOR_POINT_SINGLE_COLOR
    color_center = 3b5528
  }
}
```

All block examples bellow uses this format:

```
{
  /* First part contains block name and type:
  */
  name = block_name
  type = block_type

  /*
    Second part is a list of all posible properties,
    descriptions and default values:

    [property_type]  property_name

    If the property_type is an enumerated type, all
    posibilies come here:

    VALUE_1
    VALUE_2
    VALUE_3
  */
  property_name = default_value_of_the_property
}
```

6 Generator architecture

Whole generator is designed as a modificator chain. There is one master (root) modificator and it passes results to slave modificators. A last modificator in the chain writes results (color pixel, heights) directly to a generator target (it can be mesh itself, mesh texture or something else).

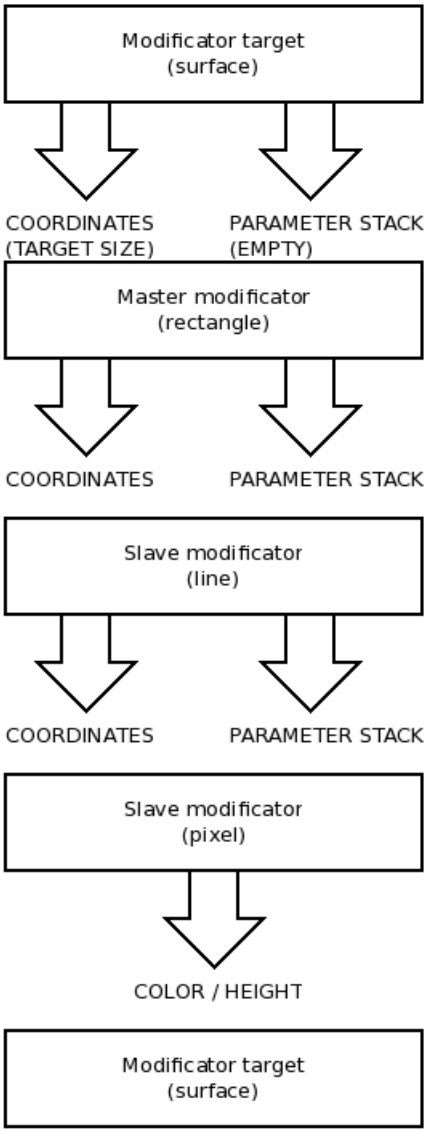

Figure 1: Modificator chain which draws a line composed from single pixels.

A picture 1 is a nice example of a modificator chain. The first modificator here is the rectangle and it's applied to whole target surface because local coordinates are not defined. If the target surface is 2048 pixels wide and 1024 pixels high, the rectangle will get $(0,0) - > (2048,1024)$ coordinates.

By default the rectangle modificator paints whole area by pixels. So it calls specified slave modificator (the line here) for each pixel inside the $(0,0) - > (2048,1024)$ target surface area.

A next modificator here is the line and it's drawn for each pixel inside the target surface. We can define (by coordinate block inside the line modificator) the line size and direction so we can obtain a rectangle filled by single lines. The lines are composed from single pixels which are drawn by the last modificator here - pixel.

- **Modificators** are atomic generator parts specialized to one task. Each modificator is configured by properties (from the data file), coordinates (from the data file and/or previous modificator) and parameters (from previous modificator) and pastes its results (coordinates, properties, parameters) to another modificator.

For instance there is a modificator which generates a line and it pastes the results (coordinates for each single point which lies on the line) to another modificator which draws them.

- **Targets** are "final" modificators which transforms results into geometry (mesh) or texture.
- **Generator** launches one or more modificators and specify which targets are used. An output of generator is a complete 3D object with material and texture.

  Generator itself can be used as a modificator so if we take the line modificator from previous example, the line modificator $->$ pixel modificator $->$ texture target chain will generate single pixels to texture, but line modificator $->$ generator chain will generate complete 3D objects on given coordinates.
- **Generator launcher** launches generators.

7 Generators

7.1 Generator launcher

Generator launcher defines which generators are performed and their order. It can be only one in the whole data file.

```
{
  /* Launcher name and type
  */
  name = generator_launcher_name
  type = GENERATOR_LAUNCHER

  /* Performed generators.
  */
  generator_mesh = first_generator
  generator_mesh = second_generator
  generator_mesh = third_generator
}
```

7.2 Generator

Generator defines which modifiators are launched, their targets and order. There can be as many generators as you want in a data file and are distinguished by their names:

```
/* A simple generator
*/
{
  /* Generator name and type
  */
  name = generator_name
  type = GENERATOR_MESH

  /*
    Modificator name and its target:

    [string]          modificator
    [enumerated type] modificator_target

      TEXTURE
      GEOMETRY
```

```
      GENERATOR_MESH
      AUX
  */
}
```

Generator items

- **modificator** - launches a generators with this name.
- **modificator_target** - defines a target of the generator.
  - **TEXTURE** - texture target (color or height)
  - **GEOMETRY** - heights in mesh geometry
  - **GENERATOR_MESH** - target is another generator
  - **AUX** - an auxiliary surface (color or height)

There is an example of a generator there:

```
/* A simple generator
*/
{
  /* Generator name and type
  */
  name = generator_name
  type = GENERATOR_MESH

  /* First modificator name and its target
  */
  modificator = first_modificator
  modificator_target = TEXTURE

  /* Second modificator name and its target
  */
  modificator = second_modificator
  modificator_target = GEOMETRY
}
```

7.3 Generated object parameters

A 3D object generated by single generator is (for now) a flat mesh with one big texture. If the texture is too big, it's sliced to smaller parts. The object is described by mesh, material and texture block.

7.3.1 Mesh params

Describes generated mesh parameters like type, size and so on:

```
{
  name = mesh_name
  type = MESH_PARAMS

  /*
    [enumerated value]  mesh_type

      MESH_LAND
```

```
      MESH_BUNCH
      MESH_GRASS
      MESH_BUSH
  */
  mesh_type = MESH_LAND

  /*
    Mesh dimensions. All values are vectors.

    [vector] start
    [vector] diff
    [vector] size
  */
  start = (0,0,0)
  diff = (1,1,1)
  size = (1,1,1)

  /*
    Parameters related to bunch:

    [int, interval]   bunch_slice_num
    [int, interval]   bunch_slice_segments

    [float, interval] bunch_slice_x_offset
    [float, interval] bunch_slice_z_offset

    [angle, interval] bunch_slice_falling
    [angle, interval] bunch_segment_falling

    [int]             bunch_slice_rotation_incemental
    [angle, interval] bunch_slice_rotation_range
    [angle, interval] bunch_slice_rotation_step
  */
  bunch_slice_num = 6
  bunch_slice_segments = 1

  bunch_slice_x_offset = 0
  bunch_slice_z_offset = 0

  bunch_slice_falling = 0
  bunch_segment_falling = 0

  bunch_slice_rotation_incemental = 0
  bunch_slice_rotation_range = 180
  bunch_slice_rotation_step = 0
}
```

Mesh items

- **mesh_type** -
  - **MESH_LAND** - a flat land
  - **MESH_BUNCH** - a bunch of plates
  - **MESH_GRASS** - not implemented yet

  - **MESH_BUSH** - not implemented yet

- **start** - mesh location
- **diff** - a size of one segment
- **size** - number of segments
- **bunch_slice_num**
- **bunch_slice_segments**
- **bunch_slice_x_offset**
- **bunch_slice_z_offset**
- **bunch_slice_falling**
- **bunch_segment_falling**
- **bunch_slice_rotation_incemental**
- **bunch_slice_rotation_range**
- **bunch_slice_rotation_step**

7.3.2 Material params

Describes material of a generated mesh:

```
{
  name = test_material
  type = MATERIAL_PARAMS

  /*
    [boolean] transparent
    [boolean] double_side
  */
  transparent = 0
  double_side = 0
}
```

Material items

- **transparent** - transparent material are for bunches
- **double_side** - double sided material are used by bunches

7.3.3 Texture params

Describes texture for a generated mesh.

```
{
  name = test_texture
  type = TEXTURE_PARAMS

  /*
    [vector]  texture_size
    [int]     texture_height
    [color]   background_color
```

```
    [int]     texture_alpha
  */
  texture_size = (512,512)
  texture_height = 512
  background_color = (0,0,0)
  texture_alpha = 0
}
```

Texture items

- **texture_size**
- **texture_height**
- **background_color**
- **texture_alpha**

8 Generator targets

8.1 GEOMETRY target

8.2 TEXTURE target

8.3 GENERATOR_MESH target

8.4 AUX target

9 Generator modificators

9.1 A generic modificator

There is a basic setup which is included in any modificator. All properties are available in all modificators, although they do not have to implement all of them and some properties can have a different meaning.

```
{
  /*
    Basic modificator properties:

    [boolean] area_inverted
    [int]     pixel_size

    [int]     pixel_step
    [int]     pixel_step_x
    [int]     pixel_step_y

    [boolean] pixel_step_random
    [int]     pixel_step_random_min
    [int]     pixel_step_random_max

    [float]   pixel_color_density

    [boolean] probability_fade
    [float]   probability_fade_start
    [float]   probability_fade_stop
```

```
    [boolean] color_fade
    [float]   color_fade_start
    [float]   color_fade_stop

    [boolean] erode_border
    [float]   erode_factor

    [float]   size_variator_theshold
    [float]   size_variator_factor
  */

  /*
    Mask properties:

    [string]  mask
  */

  /*
    Slave modificators:

    [string] modificator_slave
    [string] modificator_pre
    [string] modificator_post
  */

  /*
    Local coordinates

    Each basic setup may contain local coordinate setup. It's defined by nested
    MODIFICATOR_COORDINATE block and is described in next chaper.
  */
}
```

Generic modificator properties

- **area_inverted** - inverted rendering.
- **pixel_size** - size of single pixel. It's used only by MODIFICATOR_POINT_SINGLE.
- **pixel_step** - distance between pixels, draws grid instead of solid surface.
- **pixel_step_x** - distance between pixels in X asis.
- **pixel_step_y** - distance between pixels in Y asis.
- **pixel_step_random** - randomize pixel distances.
- **pixel_step_random_min, pixel_step_random_max** - pixel distance boundary.
- **pixel_color_density** - a probability of pixel emission, from $< 0, 1 >$ range.
- **probability_fade** - pixel probability fading, it's used by MODIFICATOR_POINT_EXTENDED only.
- **probability_fade_start**
- **probability_fade_stop**

- **color_fade** - pixel color fading, it's used by MODIFICATOR_POINT_EXTENDED only.
- **color_fade_start**
- **color_fade_stop**
- **erode_border** - pixel border erosion.
- **erode_factor**
- **size_variator_theshold** - obsolete.
- **size_variator_factor** - obsolete.
- **mask**
- **modificator_slave** - it's called for each coordinate generated by this master modificator.
- **modificator_pre** - it's called before modificator start and with top coordinates only.
- **modificator_post** - it's called when modificator finishes and with top coordinates only.

As for slave modificators - you can define up to five slave modificators for each class. Those modificators are called in order how is defined.

9.2 Coordinates

Each modificator is applied to an area which is restricted by "top" coordinates. Top coortinates are defined by master modificator or size of target surface for the first modificator.

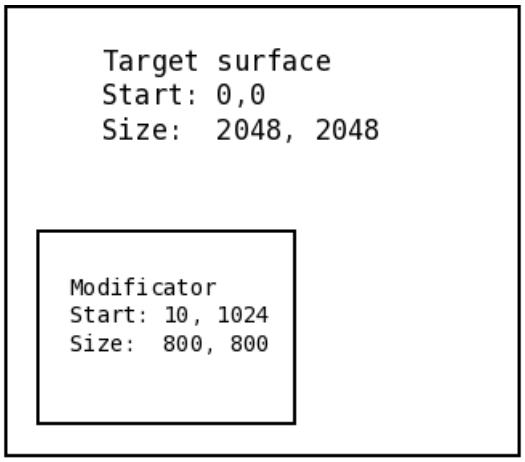

Figure 2: Target surface and one modificator.

Those "top" coordinates are further modified by local coordinate block (randomization, size extension and so on).

9.3 Coordinate block

Defines a block with **local** coordinate configuration. Top coordinates are defined by master modificator or modificator target and local coordinates are defined by coordinate block which can be included in any modificator.

Coordinate block defines operation between top and local coordinates, whether the local ones are generated (randomized) or not and so forth. If there are more than one coordinate block, the modificator is called for each of them.

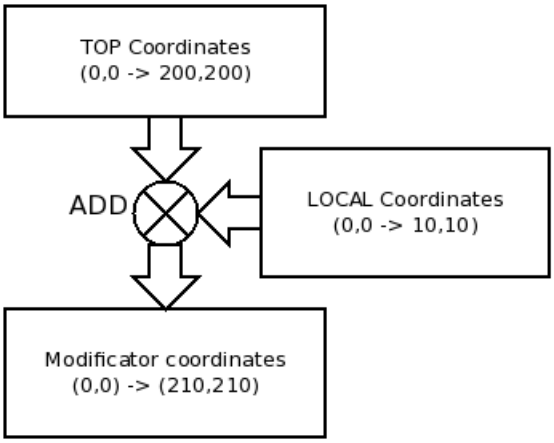

Figure 3: An example of top and local coordinates composition.

Coordinate setup properties

```
{
  /*
    Local coordinates setup:

    [aritmetic operation] coordinates_operation
    [boolean]             coordinates_random
    [int]                 coordinates_random_num
    [enumerated type]     modificator_start
    [enumerated type]     modificator_size

      COORD_CURRENT
      COORD_LAST_START
      COORD_LAST_SIZE
      COORD_LAST_START_SIZE
  */
  coordinates_operation = OPERATION_SET
  coordinates_random = 0
  coordinates_random_num = 0
  modificator_start = COORD_CURRENT
  modificator_size = COORD_CURRENT

  /*
    First coordinates blocks:
  */
  {
    type = MODIFICATOR_COORDINATE

    /*
      [vector] start
      [vector] size
      [int]    index
    */
  }
```

```
  /*
    Second coordinates blocks:
  */
  {
    type = MODIFICATOR_COORDINATE

    /*
      [vector] start
      [vector] size
      [int]    index
    */
  }

  /*
    Third one...
  */
  {
    [...]
  }
}
```

Coordinate block properties

- **coordinates_operation** - defines operation between top and local coordinates.
- **coordinates_random** - if it's set to 1, local coordinates are generated by random number generator in boundaries given by coordinates with index 0 and index 1 (see bellow).
- **coordinates_random_num** - number of generated local coordinates.
- **modificator_start, modificator_size** - it defines parts of top coordinates (start and size parts) for current coordinates_operation. It can be top coordinates from previous modificator (COORD_CURRENT) or result of last top and local coordinates composition:
  - **COORD_CURRENT** - current top coordinates
  - **COORD_LAST_START** - start of last coordinate composition (start part)
  - **COORD_LAST_SIZE** - size of last coordinate composition (size part)
  - **COORD_LAST_START_SIZE** - endpoint of last coordinate composition (start+size parts). It's userful for generating objects which have to be connected (e.g. objects strips).

Modificator coordinate sub-block properties

- **start** - coordinate start
- **size** - coordinate size, endpoint is calculated as start+size
- **index** - coordinate index (used by randomized local coordinates, see bellow)

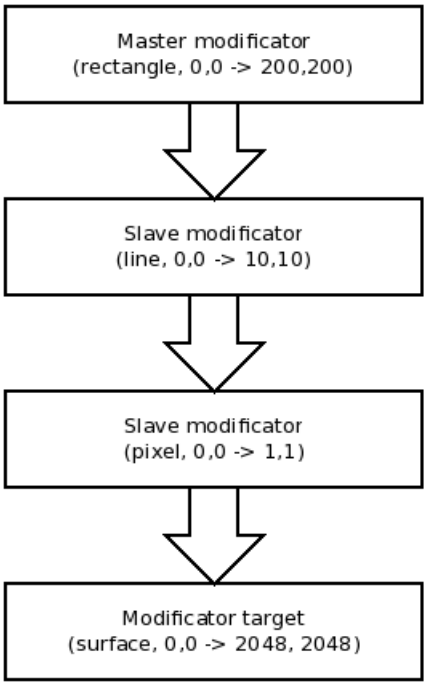

Figure 4: An example of modificator chain with local coordinate setup.

9.4 Modificator parameters

Parameters are float point values in $< 0, 1 >$ range which are passed between modificators on parameter stack. If a modificator generates any parameter, the parameter is added on top of the parameter stack. If a modificator does not emit any parameter the stack is passed without modification.

The parameters and are typically used by simple point modificator for color/height generation and so on. For instance, there's a fractal modificator which generates a height map. The fractal modificator calls a slave modificator (simple point modificator for instance) for each generated pixel and as parameter passes pixel height. So the slave pixel modificator can draw pixels by color adjusted by pixel height. Another example showns figure 1.

Modificator parameters type

Modificator parameters are defined by modificator parameter enum type:

PARAM_PREV_0	a parameter on top of the parameter stack
PARAM_PREV_1	a second one
PARAM_PREV_2	third...
PARAM_PREV_3	
PARAM_PREV_4	
PARAM_SCATTER	a random number from $< -1, 1 >$ range
PARAM_SCATTER_HALF	a random number from $< 0, 1 >$ range
PARAM_HEIGHT_MAP	not implemented yet
PARAM_HEIGHT_MESH	not implemented yet
DEFAULT	an alias for PARAM_SCATTER_HALF

9.5 Point modificators

Point modificators are designed as last modificators and usually write data directly to targets (height to mesh geometry or color/heights to texture).

9.5.1 Single point modificator

Single point modificator writes to target (slave modificator or texture target) only one single point. Its size is always 1x1 so for instance if it gets $(20, 20) -> (100, 100)$ coordinate from master modificator, it writes only single pixel to $(20, 20)$ with $(1, 1)$ size. Pixel_size property is ignored by this modificator.

The single point modificator consists from basic setup in main block and subblocks. The subblocks define particular color/height operations and are subsequently applied to a temporary color/height value. Number of color/height subblocks is not limited.

This temporary color is get from modificator target, goes through subblocks and is applied back to modificator target (as color/height to texture or height to mesh geometry).

Single point modificator block contains

```
{
  name = some_modificator_name
  type = MODIFICATOR_POINT_SINGLE

  /*
    Generator type:

    [enumerated type]  generator_type

      GENERATOR_GAUSS
      GENERATOR_RAND

    [boolean]          generator_separated
  */
  generator_type = GENERATOR_GAUSS
  generator_separated = 0

  /*
    Color operations:

    [aritmetic operation]  color_operation
    [bool]                 color_blend
  */
  color_operation = SET
  color_blend = 0

  /*
    Height operation:

    [aritmetic operation]  height_operation
   */
  height_operation = SET

  /*
    Generated colors can be crop:
```

```
    [boolean]               color_borders
    [color]                 color_border_min
    [color]                 color_border_max
   */
  color_borders = 0
  color_border_min = (0,0,0)
  color_border_max = (255,255,255)

  /*
    Color tables:

    [string]                color_table
    [string]                color_table_center
    [string]                color_table_delta
  */

  /*
    Color subblock describes single color operation.
  */
  {
    type = MODIFICATOR_POINT_SINGLE_COLOR
    [...]
  }

  /*
    Height subblock describes single height operation.
  */
  {
    type = MODIFICATOR_POINT_SINGLE_HEIGHT
    [...]
  }
}
```

Single point modificator properties

- **generator_type** - it's a generator type used for PARAM_SCATTER and PARAM_SCATTER_HALF randomisation.
  - **GENERATOR_GAUSS**
  - **GENERATOR_RAND**
- **generator_separated** - for each cycle in color/height box (see bellow) is generated a new random value
- **color_operation** - color operation between target and color pixels generated by this modificator
- **color_blend** - blend the generated pixels
- **height_operation** - height operation between target and heights generated by this modificator
- **color_borders** - are generated colors shrink to this range?
- **color_border_min** - minimal border color
- **color_border_max** - maximal border color

Color table can define a colors which can be used for color generation. For instance you can take a picture and generate pixels with colors from the image. If the color table is active, for each generated color is located the nearest color in the image (in RGB) and the nearest color is used as a result instead of the generated one.

- **color_table** - Image file (png, jpg,...) witch will be used for color table composition. Final generated colors are altered with colors from this table.
- **color_table_center** - Image file (png, jpg,...) witch will be used for color table composition. Center colors (from each color box) are altered with colors from this table.
- **color_table_delta** - Image file (png, jpg,...) witch will be used for color table composition. Delta colors (from each color box) are altered with colors from this table.

Generated parameters

None.

9.5.2 Single point modificator - color subblock

Color subblocks defines a single color operation and a result is a single color which is applied to a temporary color. The temporary color is loaded from target surface and when all color blocks are processed it's written back to target surface.

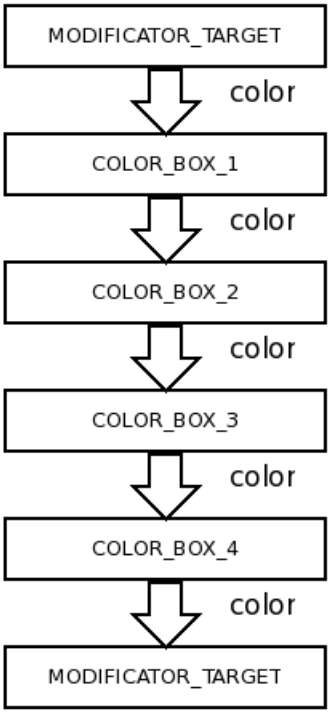

Figure 5: Color sub-block workflow.

Color subblock scheme

The color subblock is applied to input color and passes the result as an output color. The color operation applied to the input color is contoled by another input - a modifiator parameter. This

parameter can be a random number, a parameter from previous modifiator and so on (see **Modificator parameters** chaper).

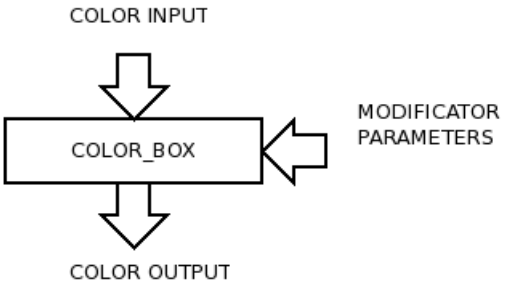

Figure 6: Color sub-block flow.

Color output calculation

There is a figure 7 with detailed schema how the color subblock output is calculated. There are two main components - **COLOR_DELTA** which is directly defined in the block and **COLOR_CENTER_CURRENT** which will be described later.

The **COLOR_DELTA** parameter is scaled by **COLOR_DELTA_SCALE** and then by modifiator parameter defined by **color_delta_parameter**. A color operation defined by **color_operation** is calculated and a result of this is combined with input color (by **final_operation**).

This operation is executed for every pixel which is processed by this modificator.

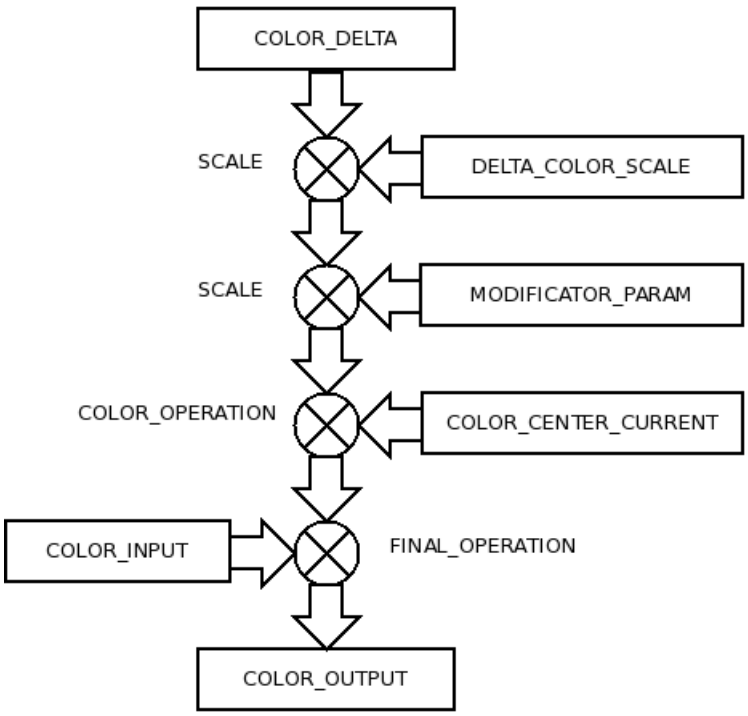

Figure 7: Color output calculation.

Color center current calculation

COLOR_CENTER_CURRENT which is referenced in previous paragraph is calculated only once before the pixels are emitted and then remains constant. It's useful when you want to set a background color whith some variation.

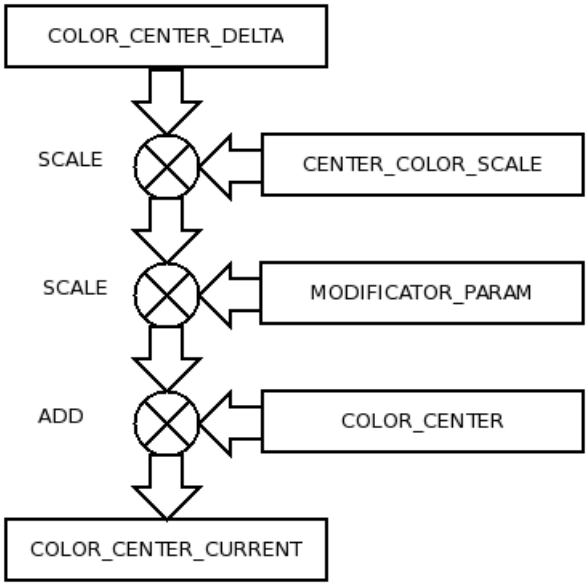

Figure 8: Color center current calculation.

Color subblock format

```
{
  type = MODIFICATOR_POINT_SINGLE_COLOR

  /*
    [aritmetic operation]   final_operation
    [boolean]               final_blend
    [modificator parameter] final_blend_parameter
  */
  final_operation = SET
  final_blend = 0
  final_blend_parameter = DEFAULT

  /*
    [aritmetic operation]   color_operation
    [boolean]               color_blend
    [modificator parameter] color_blend_parameter
  */
  color_operation = ADD
  color_blend = 0
  color_blend_parameter = DEFAULT

  /*
    [color]                 color_center
    [float]                 color_center_scale
    [float]                 color_center_delta
    [modificator parameter] color_center_parameter
  */
  color_center = (0,0,0,0)
  color_center_scale = 1
  color_center_delta = (0,0,0,0)
  color_center_parameter = PARAM_SCATTER_HALF;

  /*
    [color]                 color_delta
    [float]                 color_delta_scale
    [modificator parameter] color_delta_parameter
  */
  color_delta = (0,0,0,0)
  color_delta_scale = 1
  color_delta_parameter = PARAM_SCATTER_HALF

  /*
    Shortcuts for color definition:

    [color]                 color_min
    [color]                 color_max

    [color]                 color_center_min
    [color]                 color_center_max
  */
}
```

Color subblock properties

Color operations:

- **final_operation** - a color operation between color input and current color block result.
- **final_blend** - enable color blending for **final_operation**.
- **final_blend_parameter** - a parameter for **final_operation** blending.
- **color_operation** - a color operation between current color center and color delta.
- **color_blend** - enable color blending for this operation.
- **color_blend_parameter** - a parameter used for this color blending.

COLOR_CENTER_CURRENT calculation:

- **color_center**
- **color_center_scale**
- **color_center_delta**
- **color_center_parameter**

Parameters related to **color_operation**:

- **color_delta**
- **color_delta_scale**
- **color_delta_parameter**

Shortcuts for color definition:

- **color_min, color_max** - a shortcut for color_center and color_delta definition:

  ```
  color_center = color_min
  color_delta = (color_max-color_min)
  ```

- **color_center_min, color_center_max** - a shortcut for color_center and color_center_delta definition:

  ```
  color_center = color_center_min
  color_center_delta = (color_center_max-color_center_min)
  ```

9.5.3 Single point modificator - height subblock

Height subblock is similar to color subblock but with float point numbers (from 0 to 1). Final height is calculated as:

```
height_delta_tmp = height_delta * height_parameter
height_current = height_center (height_operation) height_delta_tmp;
height_output = height_input (final_operation) height_current;
```

Where **height_input** is a height loaded from target surface or previous block and **height_output** is a final result.

Height subblock structure

```
{
  type = MODIFICATOR_POINT_SINGLE_HEIGHT

  /*
    [float]                 height_center
    [float]                 height_delta
  */
  height_center = 0
  height_delta = 0

  /*
    [modificator parameter] height_parameter
  */
  height_parameter = PARAM_SCATTER_HALF

  /*
    [aritmetic operation]   height_operation
  */
  height_operation = ADD

  /*
    [aritmetic operation]   final_operation
  */
  final_operation = SET

  /*
    [float]                 height_min
    [float]                 height_max
  */
}
```

Height subblock properties

- **height_center**
- **height_delta**
- **height_parameter**
- **height_operation**
- **final_operation**

Shortcuts for height definition:

- **height_min, height_max** - it's a shortcut for **height_center** and **height_delta** definition.

  ```
  height_center = height_min
  height_delta = (height_max - height_min)
  ```

9.5.4 Extended point modificator

Extended point modificator draws a point (circle) from single pixels. The pixel generation is controlled by properties of generic (basic) modificator.

```
{
  name = some_modificator_name
  type = MODIFICATOR_POINT_EXTENDED
}
```

Extended point modificator properties

Extended point modificator does not have any specific properties.

Generated parameters

None.

9.6 Rectangle modificator

Rectangle modificator generates points in whole area defined by coordinates.

```
{
  name = some_modificator_name
  type = MODIFICATOR_RECT
}
```

Rectangle modificator properties

Extended point modificator does not have any specific properties.

Generated parameters

None.

9.7 Height modificators

Heightmap modifiators are used for height or parameter generation.

9.7.1 Height map modificator

Heightmap modifiator is a simple heigtmap which can be loaded from a bitmap file, a target surface or generated by a fractal generator. Its typically useful as a master modificator for MODIFICATOR_POINT_SINGLE, where pixel height is inserted to parameter stack, passed to MODIFICATOR_POINT_SINGLE modifiator and used for color shift there.

```
{
  name = some_modificator_name
  type = MODIFICATOR_HEIGHT_MAP

  /*
    [string]    height_bitmap
    [string]    height_source
  */

  /*
    [boolean]   heightmap_intensity
  */
  heightmap_intensity = 1

  /*
```

```
    [float]     height_multiplier
    [float]     height_shift
  */
  height_multiplier = 1
  height_shift = 0

  /*
    [float]     height_range_min
    [float]     height_range_max
  */
  height_range_min = 0
  height_range_max = 1

  /*
    [boolean]   scale_target
    [int]       scale_width
    [int]       scale_height
  */
  heightmap_scale = 0
  height_scale_width = 0
  height_scale_height = 0
}
```

Height map modificator properties

- **height_bitmap** - Image file (png, jpg,...) witch will be used as source for heigtmap, where height is computed from color intensity.
- **height_source** - a name of modifiator where the heightmap is obtained from.
- **heightmap_intensity** - calculate illumination for each pixel an pass it to parameter stack.
- **height_multiplier, height_shift** - pixel height modificators:

  ```
  height_final = height_pixel * height_multiplier + height_shift
  ```

- **height_range_min, height_range_max** - height range filter. Pixels outside this range are ignored.
- **scale_target**
- **scale_width**
- **scale_height**

Generated parameters

Parameter	Meaning
0	relative pixel height
1	pixel height
2	pixel intensity

Relative pixel height means that pixel height is clamped to $<$ **height_range_min**, **height_range_max** $>$ ranges and adjusted by **height_multiplier** and **height_shift**. The formula is:

```
height_translated = (height_pixel - height_range_min) / (height_range_max - height_range_min)
height_output = height_translated*height_multiplier + height_shift
```

Pixel height is an absolute pixel height. If **height_range_min = 0.5** and **height_range_max = 0.8**, all pixels are at this range $< 0.5, 0.8 >$.

Pixel intensity means that a normal vector is calculated and its dot-product with light vector is passed here. The light vector is (0,1,0) by default.

9.7.2 Mid-point modificator

Mid point fractal generator is derived from heightmap modificator so its result is a heighmap. An output of fractal generator is written to temporary heighmap modificator, normalized to $< 0, 1 >$ range and processed as a normal heighmap, so all properties of heightmap modificator can be used.

```
{
  name = some_modificator_name
  type = MODIFICATOR_FRACTAL

  /*
    [float]             fractal_hurst
  */
  generator_hurst = 0.6

  /*
    [float]             fractal_delta
    [float]             fractal_center
  */
  fractal_delta = 1
  fractal_center = 0

  /*
    [float]             correction_center
    [float]             correction_border

    [float]             border_start
    [float]             perturbation

    [int]               pixel_distance
    [int]               pixel_fill

    [int]               pixel_filter
    [int]               pixel_filter_num

    [int]               generation_border
  */
  border_start = 0
  pixel_distance = 1
  pixel_fill = TRUE
  pixel_filter = FALSE
  pixel_filter_num = 0
  generation_border = 0

  /*
    [enumerated type]   interpolation
      MID_POINT
```

```
      LINE_MIN
      LINE_MAX
      LINE_CENTER
      LINE_PRIORITY_HIGH
      LINE_PRIORITY_LOW
      LINE_RANGE_HIGH
      LINE_RANGE_LOW
      LINE_RANDOM

    [enumerated type]   interpolation_first
      MID_POINT
      LINE_MIN
      LINE_MAX
      LINE_CENTER
      LINE_PRIORITY_HIGH
      LINE_PRIORITY_LOW
      LINE_RANGE_HIGH
      LINE_RANGE_LOW
      LINE_RANDOM

    [enumerated type]   interpolation_second
      MID_POINT
      LINE_MIN
      LINE_MAX
      LINE_CENTER
      LINE_PRIORITY_HIGH
      LINE_PRIORITY_LOW
      LINE_RANGE_HIGH
      LINE_RANGE_LOW
      LINE_RANDOM

    [int]               interpolation_border
  */
  interpolation = LINE_MAX
  interpolation_border = 0
}
```

Mid-point modificator properties

- **fractal_hurst** - A hurst coeficient for fractal generator. Low value (around 0.2) means great height variation, high values (around 0.8) produces soft shapes.
- **fractal_delta, fractal_center** - An output height is produced as

  ```
  height_output = fractal_delta * height_generated + fractal_center;
  ```

- **correction_center** - Height value used for center pixel. It's off by default.
- **correction_border** - Height value used for border pixels. It's off by default.
- **border_start** - Height values of first four corner points. It's 0 by default.
- **pixel_distance** - If a distance among the four pixels is smaller than this value, pixels are not generated. It's 0 by default so the whole area is filled pixel by pixel by default.
- **generation_border** - If a distance among the four pixels is smaller than this value, pixels are only interpolated and no new height is added by fractal generator.

- **pixel_fill** - If it's TRUE, all unset pixels (for instance when **pixel_distance** > 0) are interpolated for their neighbours.
- **pixel_filter, pixel_filter_num** - Filter generated pixels. Off by default.
- **interpolation** - A new height values are derived from their four neighbor points and you can define how.
  - **MID_POINT** - An averange value is calculated from all four points.
  - **LINE_*** - Only two opposite points are interpolated so we have two height values (h1,h2) and the parameter defines which one is used.
    * **LINE_MIN** - Minimal value from (h1,h2) is choosen.
    * **LINE_MAX** - Maximal value from (h1,h2) is choosen
    * **LINE_CENTER** - A value from (h1,h2) which is closer to **MID_POINT** is choosen.
    * **LINE_PRIORITY_HIGH** - A newer value from (h1,h2) is used.
    * **LINE_PRIORITY_LOW** - An older value from (h1,h2) is used.
    * **LINE_RANGE_HIGH** - It's used a value from (h1,h2) which has been calculated from wider range.
    * **LINE_RANGE_LOW** - It's used a value from (h1,h2) which has been calculated from shorten range.
    * **LINE_RANDOM** - A random value from (h1,h2) is picked.
- **interpolation_first, interpolation_second** - You can choose different interpolation methods for different degree of interpolation. **Interpolation_first** is used for initial interpolation steps and **interpolation_second** for the final ones.
- **interpolation_border** defines a boundary when **interpolation_first** is switched to **interpolation_second** and is backward oriented, so if **interpolation_border** $= 3$ it means the interpolation style defined by **interpolation_second** is used by last three interpolations.
- **perturbation** - A random value is added to interpolated values. The formula is:

  ```
  range = MAX(v1, v2, v3, v4) - MIN(v1, v2, v3, v4)
  new_height = new_height + rand() * range * perturbation;
  ```

  v11, v12, v21, v22 are four points used for new pixel generation, **rand()** is a random number generator with $< -1, 1 >$ range. It's off by default.

Generated parameters

Are the same as for heightmap modificator.

9.7.3 Perlin noise modificator

Perlin noise generator. It's derived from heightmap modificator so its result is a heighmap. It includes all MODIFICATOR_HEIGHT_MAP modificator parameters plus some extra.

```
{
  name = some_modificator_name
  type = MODIFICATOR_PERLIN

  /*
    [float]             perlin_persistence
    [int]               perlin_octaves_start
```

```
    [int]               perlin_octaves
  */
  perlin_persistence = 0.8
  perlin_octaves_start = 0
  perlin_octaves = 30
}
```

Perlin noise modificator properties

- **perlin_persistence** - the most inportant parameter, it defines how frequently is the pattern changed.
- **perlin_octaves_start, perlin_octaves** - First octave which is generated is **perlin_octaves_start** and only **perlin_octaves** are generated totaly.

Generated parameters

Are the same as for heightmap modificator.

9.8 Line modificators

9.8.1 Single line modificator

Draws line from start point to start+size point.

```
{
  name = some_modificator_name
  type = MODIFICATOR_LINE

  /*
    [int]       line_size
  */
  line_size = 1
}
```

Line modificator properties

- **line_size** - line width in pixels.

Generated parameters

Parameter	Meaning
0	pixel distance from start

Pixel distance from start is a distance from start coordinate. The parameter is 0 for pixel at start and 1 for pixel at start+size.

9.8.2 Leaf modificator

```
{
  name = some_modificator_name
  type = MODIFICATOR_LINE_LEAF

  /*
    [float, interval] leaf_start
    [float, interval] leaf_stop
```

```
    [float, interval] leaf_width
    [angle]           leaf_thread_angle
  */
}
```

Leaf modificator properties

- **leaf_start**
- **leaf_stop**
- **leaf_width**
- **leaf_thread_angle**

Generated parameters

Parameter	Meaning
0	pixel distance from leaf center
1	pixel distance from start

9.8.3 Crack modificator

```
{
  name = some_modificator_name
  type = MODIFICATOR_CRACK

  /*
    [enumerated type]   crack_type
      DEFAULT
      CENTER

    [float]             direction_treshold
    [float]             direction_angle_range

    [int]               crack_branches
    [boolean]           crack_angle_random
  */
  crack_type = DEFAULT

  direction_angle_range = 0.1
  direction_treshold = 0.1

  crack_branches = 1
  crack_angle_random = FALSE
}
```

Crack modificator properties

- **crack_type**
  - **DEFAULT** - starts at coordinates start, crack direction is size
  - **CENTER** - starts at coordinates center (start+size/2), crack direction is randomized
- **direction_treshold** - probability that crack direction is changed.

- **direction_angle_range** - defines maximal change (an angle) if crack direction is changed (because of **direction_treshold** condition).
- **crack_branches** - crack branch num (for **crack_type = CENTER**).
- **crack_angle_random** - randomize crack angles? (for **crack_type = CENTER**)

Generated parameters

Parameter	Meaning
0	pixel distance from start

9.8.4 Brick modificator

```
{
  name = some_modificator_name
  type = MODIFICATOR_BRICK

  /*
    [int] brick_corners

    [int] brick_width
    [int] brick_height

    [float] brick_width_scatter
    [float] brick_height_scatter

    [int] brick_width_max
    [int] brick_height_max

    [int] brick_width_zip
    [int] brick_height_zip

    [float] brick_width_join_pobability
    [float] brick_width_join_pobability_multiplier

    [float] brick_height_join_pobability
    [float] brick_height_join_pobability_multiplier
  */
}
```

Brick modificator properties

- **brick_corners**
- **brick_width**
- **brick_height**
- **brick_width_scatter**
- **brick_height_scatter**
- **brick_width_max**
- **brick_height_max**
- **brick_width_zip**

- **brick_height_zip**
- **brick_width_join_pobability**
- **brick_width_join_pobability_multiplier**
- **brick_height_join_pobability**
- **brick_height_join_pobability_multiplier**

Generated parameters

None.

9.9 Bunch modificator

Bunch modificator emits points which are connected by splines and filled with some pattern.

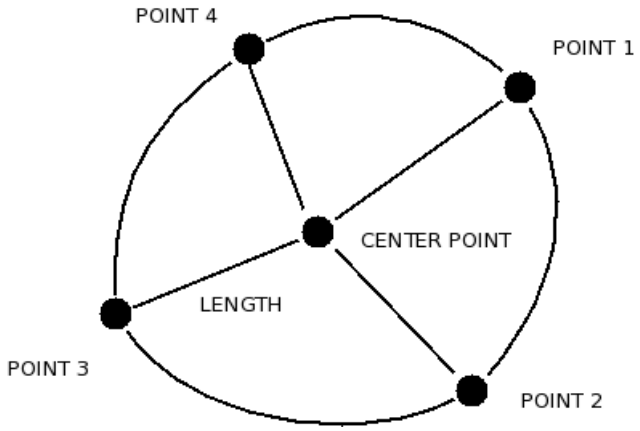

Figure 9: Bunch construction scheme.

A picture 9 describes the construction scheme. The whole object is constructed around center point which also has maximal height.

```
{
  name = some_modificator_name
  type = MODIFICATOR_BUNCH

  /*
    [float]           height

    [float]           height_correction_center
    [float]           height_correction_left
    [float]           height_correction_top
  */
  height = 1

  /*
    [float, interval] corner_curvature
  */
  corner_curvature = 0.4
```

```
  /*
    [int, interval]   points
    [float, interval] lenght
  */
  points = 4
  lenght = 0.75

  /*
    [float]           angle
    [int]             border
  */
  angle = 0
  border = 0
}
```

Bunch modificator properties

- **height** - maximal height of the emitted bunch (in the center point).
- **height_correction_center**
- **height_correction_left**
- **height_correction_top**
- **corner_curvature** - curvature parameter in points. It's a number from $< 0, 1 >$ range where 0 is totaly sharp and 1 is a circle.
- **points** - number of generated points (corners) of the bunch.
- **lenght** - a lenght between center point and the generated ones.
- **angle** - if the angle is set the angle between points is variated.
- **border**

Generated parameters

Parameter	Meaning
0	pixel height

9.10 Mask modificator

```
{
  name = some_modificator_name
  type = MODIFICATOR_MASK

  /*
    [string]          bitmap
    [enumerated type] mask_type

      MASK_BOOL
      MASK_COLOR
      MASK_HEIGHT
  */
}
```

Mask modificator properties

- **bitmap**
- **mask_type**
  - **MASK_BOOL**
  - **MASK_COLOR**
  - **MASK_HEIGHT**

Generated parameters

None.

9.11 Light modificator

Calculates light intensity for each pixel. Light values are written to target unless a slave modifiator is defined. If there is a slave modifiator defined, the light modificator does not emit any pixels to target but passes light intensity as a parameter to slave modificator.

```
{
  name = some_modificator_name
  type = MODIFICATOR_LIGHT

  /*
    [bool]    planar

    [vector]  light_position

    [float]   angle_min
    [float]   angle_scale

    [float]   specular_shine

    [color]   color_ambient

    [bool]    color_diffuse_active
    [color]   color_diffuse

    [bool]    color_specular_active
    [color]   color_specular
  */
  planar = 1
  light_position = (0,1,0)

  angle_min = 0
  angle_scale = 1

  color_ambient = (0,0,0,0)

  color_diffuse_active = 1
  color_diffuse = (1,1,1,1)

  color_specular_active = 0
  color_specular = (1,1,1,1)
```

```
  specular_shine = 1
}
```

Mask modificator properties

- **planar** - Is the light planar or spot?
- **light_position** - It's defined relative to target surface. $< -1, y, -1 >$ means upper left corner of target surface and $< 1, y, 1 >$ lower right corner of target surface. The y value defines light height and it's from $< 0, 1 >$ range, which is translated to $< 0, texture\_height >$.
- **angle_min, angle_scale** - An angle between surface normal vector and light vector is modidified as:

  ```
  angle_final = angle_min + angle_calculated * angle_scale
  ```

- **specular_shine** - $< 0, 1 >$ shine range.

Standard components of phong illumination model:

- **color_ambient**
- **color_diffuse_active**
- **color_diffuse**
- **color_specular_active**
- **color_specular**

Generated parameters

Parameter	Meaning
0	pixel light intensity

9.12 TODO

MODIFICATOR_BITMAP MODIFICATOR_FILTER MODIFICATOR_GENERATOR_MESH