

# 1 Tag datafile description

This file describes tag input files which describes whole scene.

## 2 Comments

The input file uses standard C++ comments:

```
// One line comment

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

## 3 Properties

All properties are set by:

```
property_name = value
```

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

### 3.1 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
```

### 3.2 Numbers

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

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

### 3.3 Strings

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

```
name = my_modifier_name
```

### 3.4 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.

### 3.5 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)
```

### 3.6 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
```

### 3.7 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 modifier_target to texture
modifier_target = TEXTURE

// set modifier_target to geometry
modifier_target = GEOMETRY
```

#### 3.7.1 Arithmetic 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. Arithmetic 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

### 3.8 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
```

### 3.9 Coordinates

Coordinates are 2D area which describes where a modifier 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
}
```

## 4 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
    possibilities come here:

    VALUE_1
    VALUE_2
    VALUE_3
*/
property_name = default_value_of_the_property
}

```

## 5 Generator architecture

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

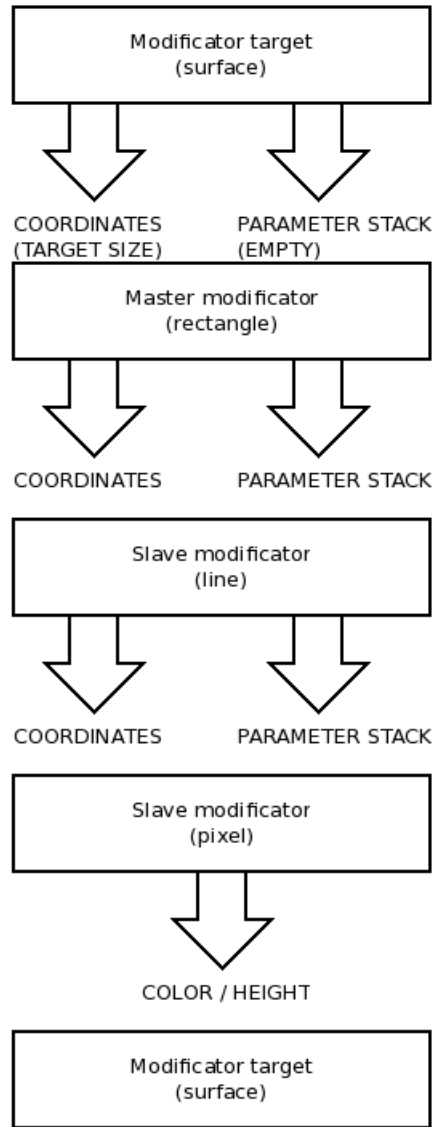


Figure 1: Modifier chain which draws a line composed from single pixels.

A picture 1 is a nice example of a modifier chain. The first modifier 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 modifier paints whole area by pixels. So it calls specified slave modifier (the line here) for each pixel inside the  $(0,0) -> (2048,1024)$  target surface area.

A next modifier here is the line and it's drawn for each pixel inside the target surface. We can define (by coordinate block inside the line modifier) 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 modifier here - pixel.

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

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

- **Targets** are "final" modifiers which transforms results into geometry (mesh) or texture.
- **Generator** launches one or more modifiers 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 modifier so if we take the line modifier from previous example, the line modifier –> pixel modifier –> texture target chain will generate single pixels to texture, but line modifier –> generator chain will generate complete 3D objects on given coordinates.

- **Generator launcher** launches generators.

## 6 Generators

### 6.1 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
}
```

### 6.2 Generator

Generator defines which modifiers 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

    /*
    Modifier name and its target:

    [string]          modifier
    [enumerated type] modifier_target

    TEXTURE
    GEOMETRY

```

```

        GENERATOR_MESH
        AUX
    */
}

```

Generator items:

- **modifier** - launches a generators with this name.
- **modifier\_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 modifier name and its target
    */
    modifier = first_modifier
    modifier_target = TEXTURE

    /* Second modifier name and its target
    */
    modifier = second_modifier
    modifier_target = GEOMETRY
}

```

## 6.3 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.

### 6.3.1 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_incremental
        [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_incremental = 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\_incremental**
- **bunch\_slice\_rotation\_range**
- **bunch\_slice\_rotation\_step**

### 6.3.2 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

### 6.3.3 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

## 7 Generator targets

### 7.1 GEOMETRY target

### 7.2 TEXTURE target

### 7.3 GENERATOR\_MESH target

### 7.4 AUX target

## 8 Generator modifiers

### 8.1 A generic modifier

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

```

{
  /*
    Basic modifier 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 modifiers:

    [string] modifier_slave
    [string] modifier_pre
    [string] modifier_post
*/

/*
    Local coordinates

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

```

#### Generic modifier 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 axis.
- **pixel\_step\_y** - distance between pixels in Y axis.
- **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**
- **modifier\_slave** - it's called for each coordinate generated by this master modifier.
- **modifier\_pre** - it's called before modifier start and with top coordinates only.
- **modifier\_post** - it's called when modifier finishes and with top coordinates only.

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

## 8.2 Coordinates

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

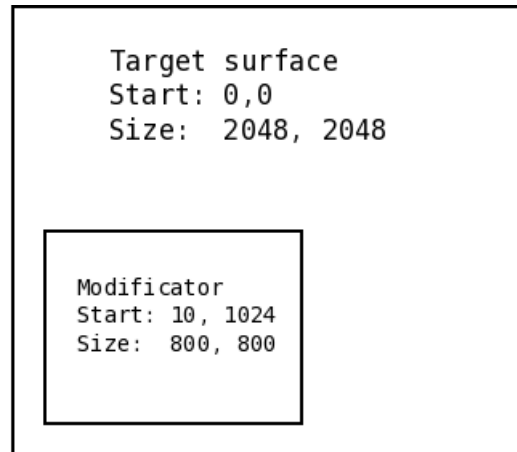


Figure 2: Target surface and one modifier.

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

## 8.3 Coordinate block

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

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 modifier is called for each of them.

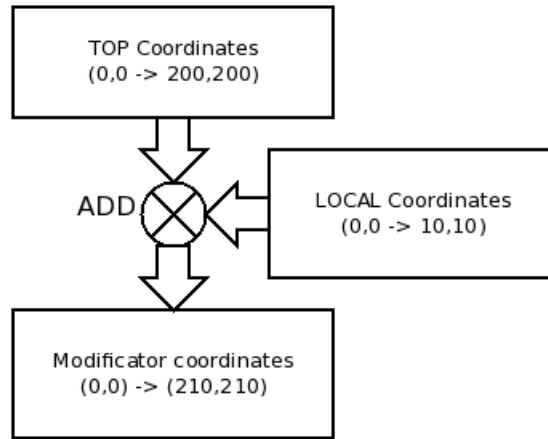


Figure 3: An example of top and local coordinates composition.

#### Coordinate setup properties:

```

{
  /*
    Local coordinates setup:

    [arithmetic operation] coordinates_operation
    [boolean]               coordinates_random
    [int]                   coordinates_random_num
    [enumerated type]       modifier_start
    [enumerated type]       modifier_size

    COORD_CURRENT
    COORD_LAST_START
    COORD_LAST_SIZE
    COORD_LAST_START_SIZE
  */
  coordinates_operation = OPERATION_SET
  coordinates_random = 0
  coordinates_random_num = 0
  modifier_start = COORD_CURRENT
  modifier_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.
- **modifier\_start, modifier\_size** - it defines parts of top coordinates (start and size parts) for current coordinates\_operation. It can be top coordinates from previous modifier (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 useful for generating objects which have to be connected (e.g. objects strips).

#### Modifier 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)

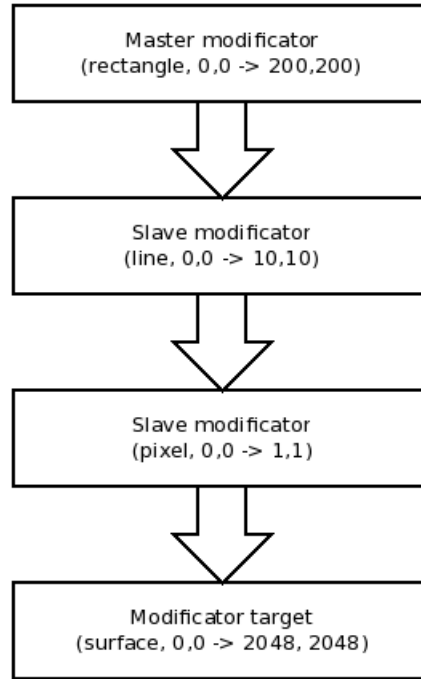


Figure 4: An example of modifier chain with local coordinate setup.

## 8.4 Modifier parameters

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

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

### Modifier parameters type

Modifier parameters are defined by modifier 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



## 8.5 Point modifiers

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

### 8.5.1 Single point modifier

Single point modifier writes to target (slave modifier 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 modifier, it writes only single pixel to (20,20) with (1,1) size. Pixel\_size property is ignored by this modifier.

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

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

**Single point modifier block contains:**

```
{
  name = some_modifier_name
  type = MODIFIER_POINT_SINGLE

  /*
    Generator type:

    [enumerated type]  generator_type

    GENERATOR_GAUSS
    GENERATOR_RAND

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

  /*
    Color operations:

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

  /*
    Height operation:

    [arithmetic 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 sub block describes single color operation.
*/
{
    type = MODIFICATOR_POINT_SINGLE_COLOR
    [...]
}

/*
    Height sub block describes single height operation.
*/
{
    type = MODIFICATOR_POINT_SINGLE_HEIGHT
    [...]
}
}

```

#### Single point modifier 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 modifier
- **color\_blend** - blend the generated pixels
- **height\_operation** - height operation between target and heights generated by this modifier
- **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.

#### 8.5.2 Single point modifier - color sub block

Color sub blocks 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.

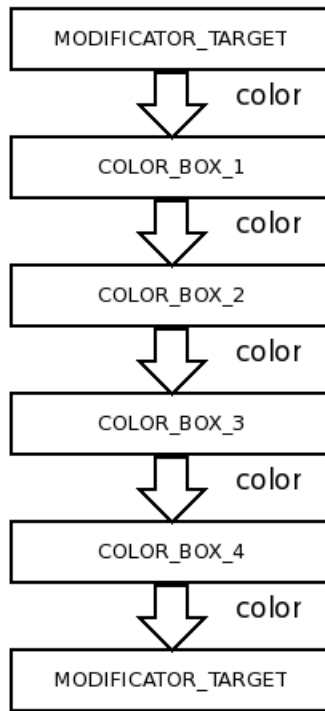


Figure 5: Color sub-block workflow.

#### Color sub block scheme

The color sub block is applied to input color and passes the result as an output color. The color operation applied to the input color is controled by another input - a modifier parameter.

This parameter can be a random number, a parameter from previous modifier and so on (see **Modifier parameters** chapter).

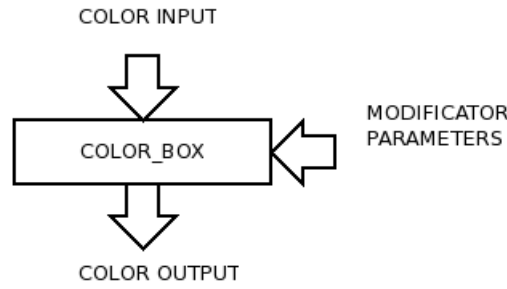


Figure 6: Color sub-block flow.

### Color output calculation

There is a figure 7 with detailed schema how the color sub block 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 modifier 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 modifier.

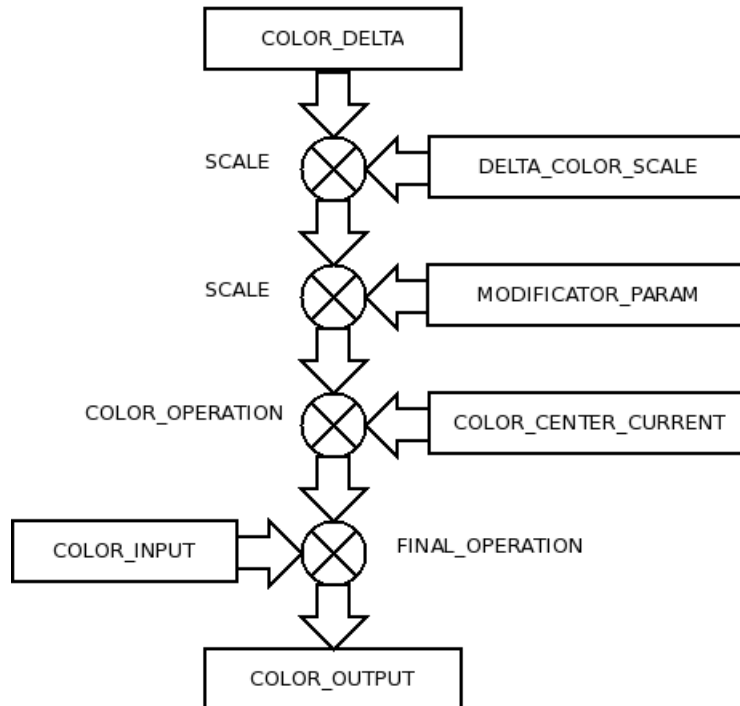


Figure 7: Color output calculation.

### 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 with some variation.

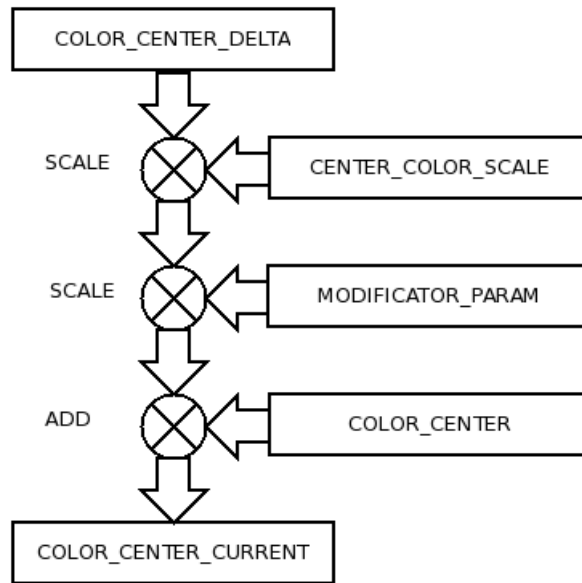


Figure 8: Color center current calculation.

### Color sub block format:

```
{
  type = MODIFICATOR_POINT_SINGLE_COLOR

  /*
    [arithmetic operation]   final_operation
    [boolean]                final_blend
    [modifier parameter]    final_blend_parameter
  */
  final_operation = SET
  final_blend = 0
  final_blend_parameter = DEFAULT

  /*
    [arithmetic operation]   color_operation
    [boolean]                color_blend
    [modifier 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
    [modifier 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
    [modifier 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 sub block 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)
```

### 8.5.3 Single point modificator - height sub block

Height sub block is similar to color sub block.

```
{  
  type = MODIFICATOR_POINT_SINGLE_HEIGHT  
  
  /*  
    [float]          height_center  
    [float]          height_delta  
  */  
  height_center = 0  
  height_delta = 0
```

```

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

/*
    [aritmetic operation]    height_operation
*/
height_op.op = ADD

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

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

```

**Height sub block 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)

```

It's similiar to color block but works with height (float point number from 0 to 1).

#### 8.5.4 Extended point modifier

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

```

{
    name = some_modifier_name
    type = MODIFICATOR_POINT_EXTENDED
}

```

**Extended point modifier properties:**

Extended point modifier does not have any specific properties.



**Generated parameters:**

None.

**8.6 Rectangle modifier**

Rectangle modifier generates points in whole area defined by coordinates.

```
{
    name = some_modifier_name
    type = MODIFICATOR_RECT
}
```

**Rectangle modifier properties:**

Extended point modifier does not have any specific properties.

**Generated parameters:**

None.

**8.7 Height modifiers**

Heightmap modifiers are used for height or parameter generation.

**8.7.1 Height map modifier**

Heightmap modifier is a simple heightmap which can be loaded from a bitmap file, a target surface or generated by a fractal generator. Its typically useful as a master modifier for MODIFICATOR\_POINT\_SINGLE, where pixel height is inserted to parameter stack, passed to MODIFICATOR\_POINT\_SINGLE modifier and used for color shift there.

```
{
    name = some_modifier_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 modifier properties:

- **height\_bitmap** - Image file (png, jpg,...) which will be used as source for heightmap, where height is computed from color intensity.
- **height\_source** - a name of modifier where the heightmap is obtained from.
- **heightmap\_intensity** - calculate illumination for each pixel and pass it to parameter stack.
- **height\_multiplier, height\_shift** - pixel height modifiers:

$$\text{height\_final} = \text{height\_pixel} * \text{height\_multiplier} + \text{height\_shift}$$

- **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  $\langle \text{height\_range\_min}, \text{height\_range\_max} \rangle$  ranges and adjusted by **height\_multiplier** and **height\_shift**. The formula is:

$$\begin{aligned} \text{height\_translated} &= (\text{height\_pixel} - \text{height\_range\_min}) / (\text{height\_range\_max} - \text{height\_range\_min}) \\ \text{height\_output} &= \text{height\_translated} * \text{height\_multiplier} + \text{height\_shift} \end{aligned}$$

**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  $\langle 0.5, 0.8 \rangle$ .

**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.

### 8.7.2 Mid-point modifier

Mid point fractal generator. It's derived from heightmap modifier so its result is a heighmap. It includes all MODIFICATOR\_HEIGHT\_MAP modifier parameters plus some extra.

```
{
    name = some_modifier_name
    type = MODIFICATOR_FRACTAL

    /*
        [float]          fractal_hurst

        [float]          fractal_delta
        [float]          fractal_base

        [int]            limited_iteration
        [float]          limited_iteration_value

        [float]          correction_center
        [float]          correction_border

        [int]            filter_back
        [float]          border_start
        [float]          perturbation

        [int]            pixel_fill
        [int]            pixel_distance

        [int]            pixel_filter
        [int]            pixel_filter_num

        [enumerated type] interpolation
        [enumerated type] interpolation_first
        [enumerated type] interpolation_second

        [int]            interpolation_border
        [int]            generation_border
    */
}
```

Mid-point modifier properties:

- fractal\_hurst
- fractal\_delta
- fractal\_base
- limited\_iteration
- limited\_iteration\_value
- correction\_center
- correction\_border
- filter\_back

- `border_start`
- `perturbation`
- `pixel_fill`
- `pixel_distance`
- `pixel_filter`
- `pixel_filter_num`
- `interpolation`
- `interpolation_first`
- `interpolation_second`
- `interpolation_border`
- `generation_border`

**Generated parameters:**

Are the same as for heightmap modifier.

### 8.7.3 Perlin noise modifier

Perlin noise generator. It's derived from heightmap modifier so its result is a heightmap. It includes all MODIFICATOR\_HEIGHT\_MAP modifier parameters plus some extra.

```
{
  name = some_modifier_name
  type = MODIFICATOR_PERLIN

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

**Perlin noise modifier properties:**

- `perlin_octaves`
- `perlin_octaves_start`
- `perlin_persistence`

**Generated parameters:**

Are the same as for heightmap modifier.

## 8.8 Line modifiers

### 8.8.1 Single line modifier

```
{
  name = some_modifier_name
  type = MODIFICATOR_LINE

  /*
    [int]          line_size

    Draws line from start point to start+size point.
  */
  line_size = 1
}
```

**Line modifier properties:**

- **line\_size**

**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.

### 8.8.2 Leaf modifier

```
{
  name = some_modifier_name
  type = MODIFICATOR_LINE_LEAF

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

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

**Leaf modifier 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

### 8.8.3 Crack modifier

```
{
  name = some_modifier_name
  type = MODIFICATOR_CRACK

  /*
    [enumerated type]    crack_type

    DEFAULT
    CENTER

    [int]                crack_branches
    [int]                crack_angle_random
    [float]              direction_angle_range
    [float]              direction_treshold
  */
}
```

#### Crack modifier properties:

- **crack\_type**
  - **DEFAULT** - starts at coordinates start, crack direction is size
  - **CENTER** - starts at coordinates center (start+size/2), crack direction is randomized
- **crack\_branches**
- **crack\_angle\_random**
- **direction\_angle\_range**
- **direction\_treshold**

#### Generated parameters:

Parameter	Meaning
0	pixel distance from start

### 8.8.4 Network modifier

```
{
  name = some_modifier_name
  type = MODIFICATOR_NET

  /*
    [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
*/
}

```

Network modifier 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.

## 8.9 Bunch modifier

```

{
  name = some_modifier_name
  type = MODIFIER_BUNCH

  /*
    [float]          height

    [float]          height_correction_center
    [float]          height_correction_left
    [float]          height_correction_top

    [float, interval] corner_curvature

    [int, interval]  points
  */
}

```

```

        [float, interval] lenght

        [float]          angle
        [int]            border
    */
}

```

**Bunch modifier properties:**

- brick\_corners
- height
- height\_correction\_center
- height\_correction\_left
- height\_correction\_top
- corner\_curvature
- points
- lenght
- angle
- border

**Generated parameters:**

None.

## 8.10 Mask modifier

```

{
    name = some_modifier_name
    type = MODIFIER_MASK

    /*
        [string]          bitmap
        [enumerated type] mask_type

        MASK_BOOL
        MASK_COLOR
        MASK_HEIGHT
    */
}

```

**Mask modifier properties:**

- bitmap
- mask\_type
  - MASK\_BOOL
  - MASK\_COLOR
  - MASK\_HEIGHT



**Generated parameters:**

None.

## **8.11    TODO**

MODIFICATOR\_BITMAP

MODIFICATOR\_LIGHT

MODIFICATOR\_FILTER

MODIFICATOR\_GENERATOR\_MESH