1 Tage datafile description

This file describes tage 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 Numbers

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

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

3.2 Strings

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

```
name = my_modificator_name
```

3.3 Colors

Colors can be defined by who ways - by separated RGB values, (0-255) by one hexadecimal digit (HTML color) or as a vector (R,G,B). For instance, we want to set R:33, G:25, B:7 to color_center value:

```
// 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.

3.4 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)
```

3.5 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
```

3.6 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
```

3.6.1 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

and OP is defined as:

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

3.7 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
```

3.8 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.

```
{
  type = MODIFICATOR_COORDINATE

  index = 0

  start_x = 0
  start_y = 0

  size_x = 40
  size_y = 40
}
```

This example describes an area which begins at (0,0) and size 40x40 pixels.

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  - property description

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

    VALUE_1     - a description of VALUE_1
    VALUE_2     - a description of VALUE_1
    VALUE_3     - a description of VALUE_1
  */
  property_name = default_value_of_the_property
}
```

5 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).

Basic terms:

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 the results to 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.

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 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 - a name of performed modificator
    [enumerated type] modificator_target - its target

    Modificator targets can be:

    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 some 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
}
```

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 types can be:

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

  /*
    Mesh dimensions. All values are vectors.

    [vector] start - lesh location
    [vector] diff  - a size of one segment
    [vector] size  - segments num
  */
  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
}
```

6.3.2 Material params

Describes material of a generated mesh:

```
{
  name = test_material
  type = MATERIAL_PARAMS

  /*
    [int] transparent
    Transparent material are for bunches
  */
  transparent = 0

  /*
    [int] double_side
    Double sided material are used by bunches
  */
  double_side = 0
}
```

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
}
```

7 Generator targets

7.1 GEOMETRY target

7.2 TEXTURE target

7.3 GENERATOR_MESH target

7.4 AUX target

8 Generator modificators

8.1 Modificators and 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 (for a first modificator).

Those "top" coordinates are further modified by local (in modificator) coordinate configuration (for instance by randomization, size extension and so on).

8.2 A generic modificator

This 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 properties:

    [int]   area_inverted
    [int]   pixel_size

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

    [int]   pixel_step_random
```

```
    [int]   pixel_step_random_min
    [int]   pixel_step_random_max

    [float] pixel_color_density

    [int]   probability_fade
    [float] probability_fade_start
    [float] probability_fade_stop

    [int]   color_fade
    [float] color_fade_start
    [float] color_fade_stop

    [int]   erode_border
    [float] erode_factor

    [float] size_variator_theshold
    [float] size_variator_factor
  */

  /*
    Slave modificators:

    You can define up to five modificators for each class.

    [string] modificator_slave - It's called for each coordinate generated
                                 by this master modificator.
    [string] modificator_pre   - It's called before modificator start and
                                 with top coordinates only.
    [string] modificator_post  - It's called when modificator finishes and
                                 with top coordinates only.
  */

  /*
    Local coordinates

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

}
```

8.3 Coordinate specification

MODIFICATOR_COORDINATE

MODIFICATOR_COORDINATE defines a block with local coordinate configuration. pict.

Top coordinates are defined by master modificator or modificator target (for first modificator). Local coordinates are defined by MODIFICATOR_COORDINATE block. It 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 MODIFICATOR_COORDINATE block, the configured modificator is called for each local coordinate.

A part of coordinates setup is in basic modificator block and the rest is in MODIFICATOR_COORDINATE blocks:

```
{
  /*
    Basic modificator block
  */

  /*
    Local coordinates setup

    Defines how are the local coordinates combined with the top one.

    [aritmetic operation] coordinates_operation

      Defines operation between top and local coordinates.

    [int]                 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).

    [int]                 coordinates_random_num

      Number of generated local coordinates.

    [enumerated type]     modificator_start
    [enumerated type]     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        - result of last coordinate composition (start part)
      COORD_LAST_SIZE         - result of last coordinate composition (size part)
      COORD_LAST_START_SIZE   - result of last coordinate composition (start+size parts)

      It's userful for generating objects which
      have to be connected (e.g. objects strips).
  */
```

```
  coordinates_operation = OPERATION_SET;
  coordinates_random = FALSE;
  coordinates_random_num = 0;
  modificator_start = COORD_CURRENT;
  modificator_size = COORD_CURRENT;

  /*
    Local coordinates blocks

    There can be one or many of those blocks and each of them defines
    one local coordinate.
  */
  {
    type = MODIFICATOR_COORDINATE

    /*
      [vector] start  - coordinate start
      [vector] size   - coordinate size
      [int]    index  - coordinate index (used by randomized local coordinates)
    */
  }

}
```

8.4 Point modificators

8.4.1 Single point modificator

MODIFICATOR_POINT_SINGLE MODIFICATOR_POINT_SINGLE_COLOR MODIFICATOR_POINT_SINGLE_HEIGHT

8.4.2 Extended point modificator

MODIFICATOR_POINT_EXTENDED

8.5 Rectangle modificator

MODIFICATOR_RECT
TODO:
Modificator input Modificator output

8.6 Height modificators

8.6.1 Height map modificator

MODIFICATOR_HEIGHT_MAP

8.6.2 Mid-point modificator

MODIFICATOR_FRACTAL

8.6.3 Perlin noise modificator

MODIFICATOR_PERLIN

8.7 Line modificators

8.7.1 Single line modificator

MODIFICATOR_LINE

8.7.2 Leaf modificator

MODIFICATOR_LINE_LEAF

8.7.3 Crack modificator

MODIFICATOR_CRACK

8.7.4 Network modificator

MODIFICATOR_NET

8.8 Bunch modificator

MODIFICATOR_BUNCH

8.9 Mask modificator

MODIFICATOR_MASK
MODIFICATOR_BITMAP MODIFICATOR_LIGHT MODIFICATOR_FILTER MODIFICATOR_GENERATOR_MESH