AniTMT Users Guide

Contents

1 Introduction

Films like Toy Story and A Bug's Life showed impressively, what quality may be achieved meanwhile by totally coputer generated films. To make such films – although in a more modest form – is a big dream for us since we know about Raytracing.

We used the Raytracer POV-Ray for our first attempts. It convinced us in static pictures but the integrated abilities for animation has been too complicated. We also didn't find any other software that offers an appropriate solution.

For this reason we decided to develop our own animation system. We chose the name AniTMT where TMT stands for the first letters of our last names.

2 How can I animate a scene ?

There are two component types that may be animated a 3D scene.

2.1 Animate objects

Objects can be moved on a flight path as a union. Additionally the direction of the object may be changed depending on the path.

To animate the whole object is always useful if you want to move it along a path of combined track elements.

2.2 Animate values

For an interpolation of values a variable is used that is already defined in the scene. This variable may be used in an expression or any where else.

Values may be used to animate properties (like the color) of an object or of the scene. It is also useful to change the size by using a scale expression or to rotate an Object only around one axis. The usage of a variable has to be defined explicitely in the scene file.

3 Preparing the scene description

A scene may contain different components. These can be either objects that may be moved and rotated or simple values. You have to specify the name of all components you want to animate in the scene file.

3.1 POV files

Here we discribe how to define components in POV-Ray scenes that may be animated.

3.1.1 Giving names to objects

In order to give names to objects that should able to be animated we use the following syntax:

```
sphere {                  // My_Sphere        <-- this is the name
  < 0, 0, 0 >, 0.45
  pigment { OldGold }
  rotate < 0, 0, 0 >
  translate < -4, 1, 1 >
}
```

The name of the object is inserted behind the object type (sphere), the left brace and the double slash that indicates a comment line. This causes POV-Ray to ignore the name. (see chapter 4.2)

3.1.2 Insert variables

Variables have a name in the POV-Ray syntax in any way. You have to specify a default value in the POV file as follows:

```
#declare MyVariable = 12345;
```

The declaration has to be terminated by a semicolon necessarily. `anitmt` will remove all `#declare` expressions that are animated and insert a new one. That is why you should use another variable to make a loop or something similar. This one may be initialized by the animated variable as shown in the following example:

```
#declare MyStartValue = 1;
#declare MyEndValue   = 9;
#declare i = MyStartValue;

#while (i <= MyEndValue)

  box {
    <0,0,0>, <1,1,1>
    pigment { color rgb < i * 0.1, 0, 0 > }
    translate i * y
  }

  #declare i = i + 1;
#end
```

If you would use `MyStartValue` instead of `i`, `anitmt` might remove the incrementation

```
#declare MyStartValue = MyStartValue + 1;
```

too.

4 Defining the animation in the ADL file

We developed a new file format for animations scripts "animation description language" (ADL).

4.1 Structure and syntax

The ADL file has a hierarchy on different levels. This represents the structure of scenes, components (objects/ variables), functions and subfunctions.

```
povscene MyScene {          // POV-Ray-Szene
  filename "myscene.pov";     // Szenendatei
  my_obj {                    // Name des Objektes
    move {                      // Bewegung auf Flugbahn
      straight {                  // Gerade zum Ursprung
        startpos    <-2,0,0>;       // Startposition
        startdir    x;              // Startrichtung
        length      3;              // Laenge in LE
        startspeed  2;              // Geschwindigkeit in LEs
      }
      circle {                    // 180 Grad Kurve
        normal      y;              // X-Z Ebene
        radius      2;              // Radius in LE
        angle       180;            // Winkel in Grad
      }
      circle {                    // 130 Grad Kurve schief im Raum
        center      <1,1,-1>;       // Rotationszentrum
        angle       130;            // Winkel in Grad
      }
      straight {}                 // Gerade in die Unendlichkeit
    }
  }
}
```

There are mainly two types of statements. You can open blocks whose body is enclosed in braces and you can define properties in blocks.

Blocks are introduced by a statement (for example `povscene`) and an optional name (ex: `MyScene`) which is necessary for explicite references.

Properties consist of a statement (ex: `filename`) an a value (ex: `"myscene.pov"`) seperated by spaces and terminated by a semicolon. The statement defines the type of the value that may be one of scalar, vector, string or a complete expression with these types. Strings are enclosed in quotes and vectors look like `<x,y,z>`. Each block allows only a predefined set of properties.

On the top level you can open scenes. The statement for POV-Ray scenes is called `povscene`. All scenes need the property `filename`. In the scene you can add component blocks that are introduced slightly different. They don't need a statement but you have to specify the name as defined in the scene file (see chapter 3).

In the component blocks you can open function blocks. The function `change` for example interpolates scalar values and the function `move` is used to animate objects by combining track elements. The function defines the type of the component.

Subfuction blocks may be defined in the functions. They define the real behaviour of the component during the animation.

4.2 Rules for naming

In scene files like POV-Rayfiles you can specify names for objects. You can also give a name to any block in the ADL file in order to enable explicite references.

You have to follow some rules for these names:

- it may consist of alphanumeric characters or underscore ('_')
- it has to start with a alphabetic character
- it may have up to 100 characters
- it is case sensitive

5 Funktions

In order to animate a component you have to specify a function that defines how it is animated. Each function only works with one type of components. The function `move` for example tells the component that it is an object. It is useful to move an object along a flight path defined through track elements.

5.1 General

As the definition of an animation should be komfortable there are the following possibilities to define your animtaion:

5.1.1 Define values of properties

In the function there is a set of subfunctions that represent successive segments in time. The behaviour of these segments is defined by properties.

In common the properties get a static value:

```
startpos    < 5, 2, 3.141 >;
```

But it is also possible to define it with an expression:

```
startpos    (5 * x) + (2 * <0,1,0>) + <0,0,pi>;
```

AniTMT has mainly the same operations like POV-Ray. A detailed reference follows soon.

5.1.2 Definition by chosen properties

Every subfunction allows different properties like `starttime`, `endtime`, `startpos`, `endpos`, `duration`, ...

If the value of any property is needed, all possibilities to calculate that property are known. By a straight track for example the endposition may be calculated with the startposition, the startdirection and the length. For all of these properties it is also known how they might be calculated.

Though it is possible to determine only as much properties so that all other properties may be calculated.

5.1.3 Value determination with neighbour elements

The determination of all necessary properties works on several levels. At first it is tried to solve all properties as described before. If it isn't possible to calculate all of them some values are given by neighbour elements.

To get a process without time jumps the start- and endtime of two neighboured elements should be the same. If you define a flight path for example the positions should be equal to avoid any jumps in space. The directions should match to avoid sharp bends and the speeds should the same to make a good looking movement. On each level one property is enabled to be passed to neighbour elements in the direction discribed befor.

With this it is possible to define an animation with relative changes like the duration. Absolute values like the startime has to be defined on one point only.

5.1.4 Setting default values

As we want to provide both powerful and konfortable functions some properties are defined by default values if they couldn't be solved. For example a konstant movement is assumed instead of an accelerated one. This might be done by setting the acceleration to zero. The endtime can also be determined with the total duration of the animation.

5.1.5 Explicite references to other elements

Furthermore, it is possible to set up relationships between several movements. In such way, you can arrange for example that a missile hits an airplane at the correct time and position. Therefor the missile defines it `endtime` and `endpos` as reference to the same properties of the airplane.

It is always possible to use this reference in an expression. Like this it is very easy to let the missile miss the plain by reaching the position some seconds later.

5.2 List of Funktions

5.2.1 Scalar interpolation (change)

This function is used to animate a variable that the you may use in any way in the scene. That is useful for textures or rotations around one axis like doors.

All subfunctions have some standard properties and some additional ones. The following may be defined for all subfunctions of change:

`starttime`	Starttime in seconds
`endtime`	Endtime in seconds
`duration`	duration in seconds
`startvalue`	Startvalue
`endvalue`	Endvalue
`difference`	Differance between start- and endvalue

For any scalar interpolation you may use 2 subfunctions:

- `linear`
- `accelerated`

linear The function `linear` is used for a konstant change of the value. The additional properties of `linear` are:

`slope`	change of the value in units per second

accelerated The function `accelerated` may be used for an accelerated change of a value. The additional properties are:

`startslope`	change of the value at the beginning in units per second
`endslope`	change of the value at the end in units per second
`acceleration`	acceleration of the change in units per second square

5.2.2 Movement of objects on a flight path (move)

All objects have a front and up vector depending on their initial location in the scene. These and the rotation center of an object may be specified in the object like that:

```
povscene{
  my_object{
    center      <1,1,1>;
    front       <1,0,1>;
    up          y;

    move{...}
  }
}
```

The default value for center is `<0,0,0>`, the one for front is `x` and the one for up is `z`;

In order to determine the exact positon of an object we use a system that specifies the line of vision by two perpendicular angles and additionally an angle that defines the rotaion around that direction.

An airplane for example may be rotated around the flight path. The resulting `up` vector of the object is used as normal vector of the horizontal plain. With this a horizontal and vertical rotation depending on the location of the airplane may be specified. By finally moving the object to the appropriate position, all possible locations may be reached.

The function `move` is used to define a flight path with track segments like a straight stretch or an arc of a circle. The line of vision is set tangatial to the path by default. The initial `up` direction of the path may be set by default and is passed from each track segment to the next one.

It is also possible to specify that an object isn't rotated dependant on the path (like an ufo) by defining the following property in the `move` block:

```
autorotate  false;
```

All subfunctions of `move` support accelerated movement and rotation around the three axes. That is why all of them accept the following properties:

General:

`starttime`	Starttime in seconds
`endtime`	Endtime in seconds
`duration`	Duration in seconds
`startup`	Up vector at the beginning
`endup`	Up vector at the end
`up_roll`	Scalar that specifies the rotation of the up vector around the track in degrees

Movement:

`startpos`	Startposition as vector
`endpos`	Endposition as vector
`startdir`	Startdirection as vector
`enddir`	Enddirection as vector
`startspeed`	Startspeed in units per second
`endspeed`	Endspeed in units per second
`acceleration`	Acceleration in units per second square

Rotation around the line of vision:

`startrot_roll`	Startvalue in degrees
`endrot_roll`	Endvalue in degrees
`diffrot_roll`	Differance in in degrees
`startrotspeed_roll`	Rotation speed at the beginning in degrees per second
`endrotspeed_roll`	Rotation speed at the end in degrees per second
`rotacceleration_roll`	Beschleunigung der Rotation

Rotation in the horizontal plain:

`startrotspeed_horizontal`	Startvalue in degrees
`endrotspeed_horizontal`	Endvalue in degrees
`diffrot_horizontal`	Unterschied in degrees
`rotacceleration_horizontal`	Rotation speed at the beginning in degrees per second
`startrot_horizontal`	Rotation speed at the end in degrees per second
`endrot_horizontal`	Beschleunigung der Rotation

Rotation in the vertical direction:

`startrot_vertical`	Startvalue in degrees
`endrot_vertical`	Endvalue in degrees
`diffrot_vertical`	Unterschied in degrees
`startrotspeed_vertical`	Rotation speed at the beginning in degrees per second
`endrotspeed_vertical`	Rotation speed at the end in degrees per second
`rotacceleration_vertical`	Beschleunigung der Rotation

Of corse you have to specify only few properties of all these. If you don't want a rotation you don't have to specify any. Then the angles will be set to zero by default and the object only rotates depending on the flight path. (see chapter 5.1.4)

The function `move` accepts two subfunctions at the moment:

- `straight`
- `circle`

straight The subfunction `straight` moves the object on a straight stretch. The only additional property that may be specified is:

`length`	Stretch in units

circle The subfunction `circle` moves an object on an arc of a circle. This may be in any plain. The follwing properties may additionally be specified:

`length`	Stretch in units
`normal`	Normal vector of the plain of the orbit
`radius`	Radius of the circle
`center`	Center of the circle
`angle`	Angle of the arc