POV-Ray à la conquête du feu et de la fumée: le système de particules de Rune

Merci à Diamond Editions pour son aimable autorisation pour la mise en ligne de cet article, initialement publié dans Linux Magazine N°71

Olivier Saraja (olivier.saraja@linuxgraphic.org)

Un système de particules peut être utile pour simuler un grand nombre de choses: un banc de poissons dans des eaux tropicales, les forces rebelles se jettant à l'assaut de l'Etoile de la Mort ou des choses à la fois plus anodines et plus spectaculaires, comme des flammes et de la fumée, des étincelles ou tout simplement une cascade d'eau. Cet article est une introduction à l'usage des systèmes de particules. Il en existe plusieurs, mais nous nous attarderons sur celui de Rune, car le sien se révèle très générique et peu spécialisé, bien que facile d'emploi, facilitant la simulation d'un grand nombre de phénomènes possibles.

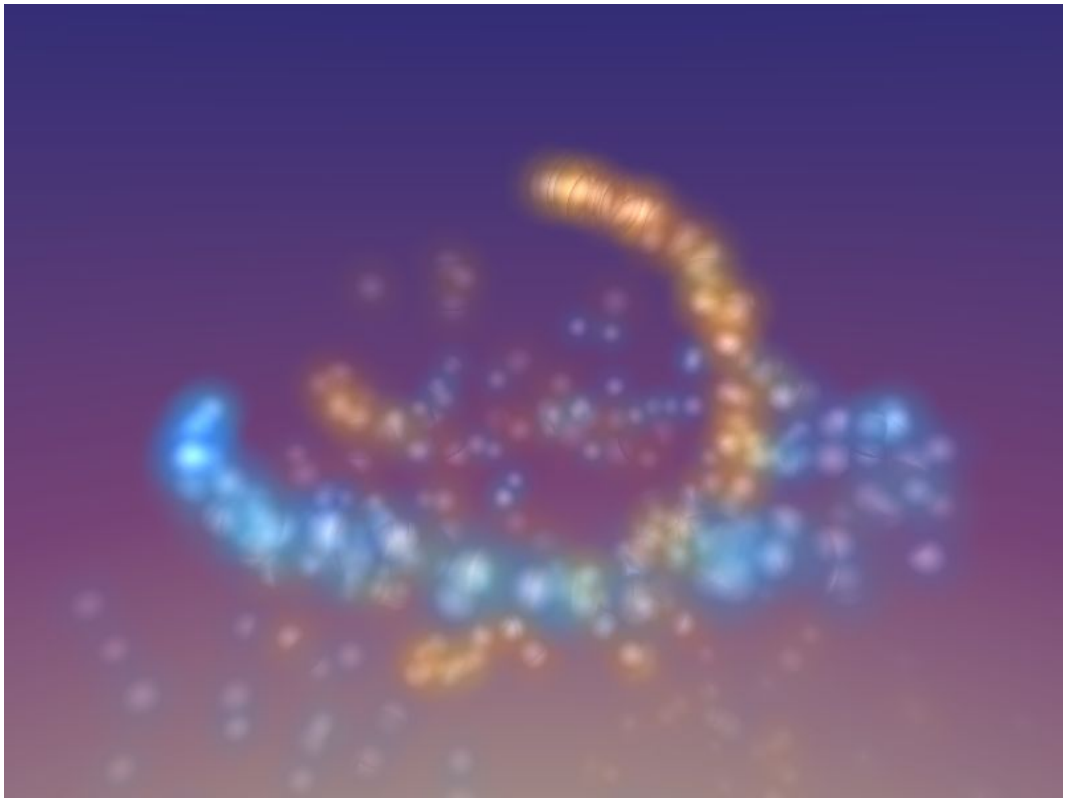

Figure 1: Un exemple très coloré de particules en action

Il en est peut-être parmi vous qui ne savent absolument pas ce qu'est un système de particule. Disons qu'une particule est un tout petit élément (assimilable à un point immatériel) auquel on va donner des attributs bien précis, dans le cadre d'une animation: une « date » de naissance ainsi qu'une durée de vie, une vitesse et une direction initiales. Lors de son « voyage » au-travers de votre scène, la particule sera affectée par le champ de gravité terrestre, la force du vent, la résistance de l'air, et rebondira contre les obstacles (soit à la façon d'un boule de billard, soit d'une façon totalement chaotique) ou les traversera en les ignorant superbement. La particule pourra être émise par un point totalement immatériel, ou par un objet apparaissant dans votre scène. Il ne vous restera plus qu'à déterminer le nombre de particules (parfois des centaines de

milliers!) et à leur attribuer formes et couleurs pour obtenir un flot immense de particules, chacune gérée individuellement! Grâce aux réglages que nous présenterons dans cette article, vous n'aurez pas grand mal à plier ce flot de particules à des lois physiques (simplifiées mais néanmoins bien assez réalistes pour nos besoins) pour aboutir à des comportements crédibles: fumée et flammes sont alors les premiers principes d'application des particules. C'est du moins ceux auxquels nous allons nous attacher dans cet article, à votre charge de découvrir les autres applications (quasiment infinies!) possibles.

1. L'utilisation du système, en résumé

Pour utiliser ce système, vous aurez un nombre minimal de lignes à inscrire en plus dans votre scène. En effet, il vous faudra d'abord inclure le fichier *include* **particle.inc**, puis spécifier toutes les options, poursuivre avec la macro **particle_element** et enfin appeler le système de particules. Si vous ne voulez pas vous perdre et être sûr de ne rien oublier, vous pourriez partir sur la base de la scène **quickref.pov**, livrée avec ce système de particules, qui condense l'intégralité des options et l'architecture minimale à garantir.

```
// inclusion du système de particules à votre scène
#include "particle.inc"

// définition des réglages
#declare particle_start  = 0.0;
#declare particle_end    = 1.0;
#declare particle_cyclic = off;
#declare particle_steps  = 100;
...

// mise en oeuvre de la macro particle_element
#macro particle_element ()
...
#end

// appel du système de particules
particle_system ("mon_système_de_particules")
```

De part son fonctionnement, ce système de particules va écrire sur votre disque dur (dans le même répertoire que votre scène courante) des fichiers présentant une extension de type ***.data**, qui contiendront l'état de votre système de particules entre chaque frame de l'animation. Lorsque l'animation est terminée d'être tracée, vous pourrez librement effacer ces fichiers, à moins que vous souhaitiez les modifier à la main et/ou les appeler ultérieurement avec l'option **load_system** (pour les utilisateurs avancés).

> **Note:**
>
> Vous pouvez mettre plusieurs système particules en oeuvre, en insérant à la suite des précédents de nouveaux blocs de définition des réglages, de mise en oeuvre de la macro **particle_element** et enfin de l'appel du système de particules, en donnant à cette étape un nom différent, cette fois! Par exemple:
>
> ```
> #include "particle.inc"
>
> // définition des réglages du premier système
> #declare particle_start  = 0.0;
> #declare particle_end    = 1.0;
> #declare particle_cyclic = off;
> #declare particle_steps  = 100;
> ...
> // mise en oeuvre de la macro particle_element
> #macro particle_element ()
> ...
> #end
>
> // appel du premier système de particules
> particle_system ("systeme_1")
> ```

```
// définition des réglages du deuxième système
#declare particle_start  = 0.0;
#declare particle_end    = 1.0;
#declare particle_cyclic = off;
#declare particle_steps  = 100;
...
// mise en oeuvre de la macro particle_element
#macro particle_element ()
...
#end

// appel du premier système de particules
particle_system ("systeme_2")
...
// etc avec autant de systèmes que souhaités
```

Bien sûr, vous noterez qu'il est inutile de répéter la ligne **#include "particle.inc"** à chaque fois.

2. Les paramètres affectant la simulation de particules

Tous les paramètres déclarés du système de particules doivent rester constants au cours d'une animation, vous n'avez donc pas le droit de les faire varier en fonction de la valeur d'horloge **clock**, comme nous avons pu le voir faire dans les deux articles *Vos premières animations avec POV-ray*, 1ère partie et 2ème partie, respectivement dans GNU/Linux Magazine 66 et 68 ou disponibles en ligne sur **http://www.linuxgraphic.org**. De même, tous les paramètres sont optionnels. Lorsqu'un paramètre n'est pas spécifié, le paramètre par défaut sera employé, s'il en existe un. Bien sûr, nous vous indiquons celui-ci le cas échéant, mais vous le retrouvez également dans la scène de départ **quickref.pov**.

2.1 Les paramètres d'horloge

Les paramètres d'horloge contrôlent le comportement du système de particules vis-à-vis de la variable clock de POV-ray:

```
// Clock settings
// **************

   #declare particle_start  = 0.0;
   #declare particle_end    = 1.0;
   #declare particle_cyclic = off;
   #declare particle_steps  = 100;
```

particle_start, particle_end

Ce sont les valeurs de **clock** définissant l'intervalle de temps pendant lequel le système de particules est actif. Par défaut:

```
#declare particle_start = 0.0;
#declare particle_end = 1.0;
```

particle_cyclic

Ce paramètre vous permet de régler l'état initial des particules, au début de **particle_start**, de façon à ce qu'elles concordent avec leur état final, en fin de **particle_end**, de sorte que l'émission de particules soit cyclique. Pour obtenir ce résultat, il vous suffit de régler ce paramètre sur **on**. Veuillez noter que la première frame de l'animation sera *parsée* très lentement, car le système devra considérablement « remonter le temps » afin de déterminer l'activité particulaire antérieure. Par défaut:

```
#declare particle_cyclic = off;
```

> **Attention:**
>
> Le temps de *parsing* de la première image de l'animation des particules sera beaucoup plus long que la normale; en effet, POV-ray va calculer le comportement du système de particules un grand nombre d'images en avance pour ajuster l'état du système initial afin qu'il corresponde à l'état final.

particle_steps

Cette valeur spécifie le nombre d'étapes de calcul réalisé au cours de l'animation particulaire. De faibles valeurs *parseront* rapidement, alors que des valeurs élevées conduiront à des résultats plus précis. Une bonne astuce pour déterminer si vous utilisez suffisamment d'étapes consiste à effectuer le tracé de l'animation avec seulement la moitié des étapes précédentes. Si le résultat paraît très différent, c'est que vous avez utilisé trop peu d'étapes. En revanche, si vous effectuez le tracé de l'animation avec deux fois plus d'étapes et que le résultat final est pratiquement identique, en ce cas vous utilisez suffisamment d'étapes, voire même peut-être plus que nécessaire. Par défaut:

```
#declare particle_steps = 100;
```

> **Remarque:**
>
> Si vos particules semblent être émises par petits groupes compactes, c'est certainement du au fait que vous avez spécifié une valeur de **particle_steps** insuffisante. Essayez simplement d'augmenter cette valeur pour résoudre ce problème.

2.2 Paramètres généraux des particules

Les paramètres généraux des particules permettent de définir le nombre de particules en jeu, ainsi que leur longévité.

```
// General particle settings
// ************************

   #declare particle_frequency = 100;
   #declare particle_life      = 1.0;
   #declare particle_lifeturb  = 0.0;
   #declare particle_seed      = 123;
   //#declare particle_maxnumber = 100;
```

particle_frequency

Cette valeur détermine combien de particules seront émises par incrément de **clock**. Par défaut:

```
#declare particle_frequency = 100;
```

particle_life

Cette valeur détermine la durée d'une particule (en unité de **clock**) avant qu'elle ne disparaisse. Par défaut:

```
#declare particle_life = 1.0;
```

particle_lifeturb

Cette valeur contrôle la quantité de hasard prenant part à la durée de vie de la particule. 0.0 indique une durée de vie pas du tout aléatoire, tandis que 1.0 indique une durée de vie totalement aléatoire. Par défaut:

```
#declare particle_lifeturb = 0.0;
```

particle_seed

Cette valeur détermine la « graine aléatoire » utilisée dans la simulation de la particule. Par défaut:

```
#declare particle_seed = 123;
```

> **Note:**
>
> Le système faisant appel à une graine aléatoire plutôt qu'à un nombre déterminé au hasard, il devient donc pseudo-aléatoire plutôt que véritablement aléatoire; cela a une importance dans le sens où cela garantie qu'avec les mêmes paramètres, vous obteniez exactement la même animation lorsque vous en faites le tracé deux fois d'affilée.

particle_maxnumber

Si elle est spécifiée, cette valeur définit le nombre maximum de particules autorisées à coexister en même temps. Si elle n'est pas spécifiée, le système déterminera automatiquement ce nombre, et le fera efficacement, la plupart du temps. Mais si l'option **particle_emitting** (voir les *Réglages de l'émetteur*, en section suivante) n'est active qu'un court laps de temps, en ce cas le calcul automatique déterminera un nombre de particules beaucoup trop élevé, ce qui ralentira le *parsing*. En ce cas, vous voudrez sans doute définir **particle_maxnumber** manuellement lorsque **particle_emitting** est court.

2.3 Réglages de l'environnement

L'environnement permet de régir le frottement de l'air, la force du vent et sa direction, ainsi que l'effet de la pesanteur, sur les particules en jeu. Certaines options sont en fait des macros, fonctionnant avec une ou deux variables, **Clock** (avec un C majuscule, à ne pas confondre avec **clock**, la variable de POV-ray qui régit l'écoulement du temps de votre animation) et **Point**. Les vecteurs employés conjointement à ces macros doivent être exclusivement dépendants de ces deux variables, et en aucun cas d'autres valeurs externes au simulateur de particules, comme **clock**, par exemple.

```
// Environment settings
// ********************

   #declare particle_drag      = 0.0;
   #declare particle_transfer  = 0.0;

   #macro particle_gravity  (Clock,Point) <0,0,0> #end
   #macro particle_wind     (Clock,Point) <0,0,0> #end
```

particle_drag

Cette valeur définit la résistance de l'air. Plus grande est cette valeur, plus les particules seront freinées par l'air, ou seront affectées par la direction du vent si celui-ci est spécifié. Par défaut, l'air ne présente aucune résistance:

```
#declare particle_drag = 0.0;
```

particle_transfer

Si l'émetteur est lui-même en mouvement, cette option permet de déterminer quel part de sa propre vitesse il communiquera aux particules qu'il émettra. Une valeur de 0.0 correspond à aucune vitesse transmise, et 1.0 à 100% de vitesse transmise.

Par défaut, la valeur est égale à 0.0, mais si vous souhaitez une simulation réaliste, elle doit souvent être réglée à 1.0, et l'option **particle_drag** doit alors être utilisée pour freiner les particules, en bout de course. Par défaut:

```
#declare particle_transfer = 0.0;
```

> **Attention:**
>
> Cette fonctionnalité ne détecte que les mouvement et vitesses communiquées par les réglages du **particle_emitter** à l'exception de **particle_emitobj**. Pour plus d'informations, lisez les *Réglages de l'émetteur*, dans la section suivante.

particle_gravity

Cette macro contrôle la force de pesanteur qui doit affecter les particules. Elle est gouvernée par deux paramètres: **Clock** et **Point**, qui permettent de faire varier la gravité en fonction du temps, ou en fonction de la position dans l'espace. Cette macro fonctionne conjointement à un vecteur, typiquement **-y** ou n'importe quel multiple, comme **-5*y**, mais en fait n'importe quel vecteur peut-être employé (simulation de particules dans l'espace, par exemple!). Par défaut:

```
#macro particle_gravity (Clock,Point) <0,0,0> #end
```

particle_wind

Cette macro contrôle le vent qui affecte les particules. Elle est gouvernée par deux paramètres: **Clock** et **Point**, qui permettent de faire varier le vent en fonction du temps, ou en fonction de la position dans l'espace. Cette macro fonctionne conjointement à un vecteur. Le vent n'a aucune influence si l'option **particle_drag** a été déclarée égale à 0.0. Par défaut:

```
#macro particle_wind (Clock,Point) <0,0,0> #end
```

2.4 Réglages de l'émetteur

Les réglages de l'émetteur contrôlent l'émetteur de particules, depuis lequel les particules sont émises. Plusieurs réglages permettent de spécifier les directions dans lesquelles sont émises les particules. Quand plus d'une de ces options sont spécifiées, les vecteurs de direction sont simplement ajoutés les uns aux autres. Tous les paramètres sont optionnels. Lorsqu'un paramètre n'est pas spécifié, la valeur par défaut sera utilisée, s'il en existe une. De plus, comme précédemment, certaines options sont en fait des macros, fonctionnant avec une ou deux variables, **Clock** et **Point**, et les vecteurs associés doivent être exclusivement dépendants de ces deux variables.

```
// Emitter settings
// ****************

   #macro particle_emitter  (Clock) <0,0,0> #end
   #macro particle_emitting (Clock) on      #end
   #macro particle_emitvect (Clock) <0,0,0> #end
   #macro particle_emitturb (Clock) 0.0     #end
   //#macro particle_emitobj  (Clock) object {} #end
   #macro particle_emitobjn (Clock) 0.0     #end
```

particle_emitter

Cette macro contrôle la position de l'émetteur de particules. Elle n'a qu'un seul paramètre: **Clock**. qui permet de déplacer la position de l'émetteur en fonction du temps, et fonctionne conjointement à un vecteur. Par défaut:

```
#macro particle_emitter (Clock) <0,0,0> #end
```

> **Note:**
>
> L'usage de toutes ces macros avec le paramètre **Clock** peut vous poser quelques soucis si vous n'êtes pas trop familier de l'animation à l'aide de splines avec POV-ray. Par exemple, nous allons définir une spline qui correspondra au trajet de notre émetteur de particules. Supposons donc le code suivant:
>
> ```
> #declare Location_Spline =
> spline {
>    natural_spline
>    0.00, < -2,0,  2>,
>    0.10, < -2,0,  2>,
>    0.11, < -2,0,  2>,
>    0.20, < -2,5,  2>,
>    0.30, < -2,5,  2>,
>    0.40, < -4,5, -2>,
>    0.55, <  2,5, -6>,
>    0.65, < -2,6,-10>,
>    0.75, < -2,6,-10>,
>    1.00, <-32,9,-40>,
> ```

```
      1.25, <-62,9,-70>
   }
```

Il ne nous reste plus qu'à indiquer au système de particules que l'émetteur doit suivre cette spline au cours de l'animation, par exemple à l'aide de la ligne suivante:

#macro particle_emitter (Clock) Location_Spline(Clock) #end

Si à la place de Locatio_Spline(Clock) vous spécifier quelque chose comme <0, 3, 2.5>, vous déterminez alors une position constante de l'émetteur dans votre scène.

Pour être sûr de bien comprendre l'usage des splines au cours d'une animation, nous vous invitons à reconsulter la 2ème partie de l'article *Votre première animation avec POV-ray* dans GNU/Linux Magazine N°68, ou en ligne sur **http://www.linuxgraphic.org**.

particle_emitting

Cette macro sert à activer ou désactiver l'émission de particules en fonction du temps. Elle n'a qu'un seul paramètre: **Clock**, qui permet de renvoyer valeur booléenne qui sera **true** (vraie) ou **false** (fausse). Par exemple, l'expression **(Clock<0.5)** désactivera l'émetteur lorsque la valeur d'horloge du simulateur de particules atteindra 0.5. Par défaut:

```
#macro particle_emitting (Clock) on #end
```

particle_emitvect

Cette macro contrôle la direction dans laquelle les particules sont émises ainsi que la force avec laquelle elles le sont. Elle fonctionne conjointement à un vecteur. Par défaut:

```
#macro particle_emitvect (Clock) <0,0,0> #end
```

particle_emitturb

Cette macro permet de perturber l'émission de particules dans les trois directions principales. Elle n'a qu'un seul paramètre: **Clock**, qui permet donc de faire varier la perturbation en fonction du temps. Par défaut:

```
#macro particle_emitturb (Clock) 0.0 #end
```

particle_emitobj

Cette macro totalement optionnelle permet de forcer les particules à être émise depuis un objet plutôt que depuis un point. Dans ce cas, les particules sont émises par la surface de l'objet. Cette macro n'a qu'un seul paramètre: **Clock**, qui permet de déplacer l'objet émetteur en fonction du temps. L'objet **particle_emitobj** sera automatiquement translaté par le point spécifié dans le paramètre **particle_emitter**. Par conséquent, soit l'objet **particle_emitobj** doit être centré à l'origine en permanence, soit le **particle_emitter** devra être réglé sur <0,0,0> en permanence. Relisez également **particle_transfer** dans les *Réglages de l'environnement*, plus haut. Par défaut: néant. Si la macro n'est pas spécifiée, un simple point sera employé en guise d'émetteur.

particle_emitobjn

Lorsque la macro **particle_emitobj** est utilisée pour créer un objet émetteur, la macro **particle_emitobjn** peut être utilisée pour forcer les particules à être émises suivant les normales de la surface de l'objet émetteur. Cette macro n'a qu'un seul paramètre: **Clock**. Elle fonctionne conjointement à un nombre réel qui contrôle la force avec laquelle les particules sont émises par l'objet émetteur. Par défaut:

```
#macro particle_emitobjn (Clock) 0.0 #end
```

2.5 Paramètres de collision

Les paramètres de collision déterminent ce qu'il arrive lorsque les particules entrent en collision avec des objets. Rappelez-vous que tous les paramètres #déclarés du système de particules doivent demeurer constant durant toute l'animation. Comme précédemment, tous les paramètres sont optionnels, et lorsqu'un paramètre n'est pas spécifié, la valeur par défaut est utilisée, s'il en existe une.

```
// Collision settings
// ******************

   //#declare particle_blockobj     = object {}
   #declare particle_bounce       = 0.5;
   #declare particle_bounceturb   = 0.5;
   #declare particle_friction     = 0.0;
   #declare particle_bounceoffset = 0.01;
   //#macro particle_killobj    (Clock) object {}    #end
```

particle_blockobj

Il s'agit de l'objet contre lequel la simulation de particules va effectuer une détection de collision. Si cette option n'est pas spécifiée, la détection de collision est désactivée. Par défaut: néant. Si aucun objet n'est spécifié, la détection de collision est désactivée. Par exemple, si vous souhaitez que les particules rebondissent contre l'objet Mur (déjà spécifié, déclaré ou inclus en amont dans la scène), utilisez la ligne suivante (le **;** de fin de ligne n'est pas nécessaire):

```
#declare particle_blockobj = Mur
```

> **Attention**
>
> Vous ne pouvez pas définir un objet mobile comme étant un objet bloquant... La version actuelle du système de particules de Rune ne le permet pas pour l'instant, mais il le permettra peut-être un jour, dans une version ultérieure!

particle_bounce

Cette valeur détermine la force du rebond d'une particule après avoir frappé une surface bloquante, définie par l'option **particle_blockobj**. Avec une valeur de 0.0, il n'y a aucun rebond et la particule s'immobilise à la surface de l'objet, à moins d'être affecté par d'autres paramètres d'environnement. Avec une valeur de 1.0, toutefois, la particule rebondit à exactement la même vitesse qu'elle avait au moment du contact. Par défaut:

```
#declare particle_bounce = 0.5;
```

particle_bounceturb

Cette valeur détermine la part de perturbation ajoutée à la direction de rebond lorsqu'une particule rebondit contre une surface bloquante, en raison de l'usage des options **particle_blockobj** et **particle_bounce**. Avec une valeur de 0.0, les particules rebondissent de façon parfaite, à la façon de boules de billard, tandis qu'avec une valeur de 1.0, elles rebondissent de façon totalement aléatoier et inattendue. Par défaut:

```
#declare particle_bounceturb = 0.5;
```

particle_friction

Lorsque **particle_friction** est réglée à 0.0, lors d'un contact, la vitesse de glissement de la particule sur la surface est totalement inaltérée. En augmentant cette valeur, le « frottement » de la particule sur la surface va diminuer sa vitesse dans la direction tangentielle à la surface. Lorsque cette valeur est égale à 1.0, la vitesse dans la direction tangentielle est diminuée d'autant que l'est la vitesse normale à la surface, contrôlée par la variable **particle_bounce**. Généralement, une faible valeur de **particle_friction** permet aux particules de « couler » plus facilement le long des surfaces, tandis qu'une valeur élevée de **particle_friction** fait s'immobiliser plus ou moins rapidement les particules. Par défaut:

```
#declare particle_friction = 0.0;
```

particle_bounceoffset

Cette valeur est utilisée pour éviter les erreurs visuelles lors de la collision de particules. Elle permet de décaler légèrement la particule de la surface au moment de la collision, évitant ainsi que la particule ne « traverse » accidentellement la surface bloquante. Cette valeur devrait être suffisamment petite pour ne pas être remarquée au cours de l'animation, mais si vous constatez que des particules passent trop facilement au travers des surfaces, vous pourriez l'augmenter un peu. Par défaut:

```
#declare particle_bounceoffset = 0.01;
```

particle_killobj

Cette macro définie par l'utilisateur peut être utilisée pour faire tuer prématurément (avant qu'elles n'arrivent à la fin de leur durée de vie) les particules qui entrent dans l'espace occupé par un objet spécifique. Elle n'a qu'un paramètre: **`Clock`**. Par défaut: néant. Si la macro n'est pas spécifiée, il n'y a pas d'objet tueur de particules. Par exemple, si vous souhaitez que les particules symbolisant des bulles de savon éclatent au contact de l'objet Plafond (spécifié, inclus ou déclaré en amont de la scène), vous pouvez utiliser la ligne de code suivante:

```
//#macro particle_killobj (Clock) object { Plafond } #end
```

> **Note**
>
> La détection de collision n'est possible qu'avec l'environnement du système de particules, pas entre les particules elles-mêmes. Cela explique pourquoi il n'est pas possible d'utiliser ce système pour la simulation des fluides, comme par exemple remplir un verre de particules d'eau et faire tomber un glaçon dedans... Pour ce type d'usage, il convient de se tourner vers d'autres systèmes que celui de Rune.

2.6 La macro particle_element

La macro **`particle_element`** sert à déterminer l'apparence visuelle des particules. Que vous souhaitiez de l'eau, de la fumée, des étincelles ou quoi que ce soit d'autre, tout n'est qu'une affaire de bon réglage de la macro particle_element. A la base, on vous donne des variables qui vont vous servir à définir les particules. Contrairement aux autres macro définies par l'utilisateur, vous pouvez vous référer à la variable clock au sein de la macro **`particle_element`**. Les macros **`particle_element`** n'ont habituellement aucun paramètre, et c'est vous qui décidez ce qu'elles retournent, habituellement un objet. Les variables disponibles sont décrites ci-après.

> **Note**
>
> Rune fournit avec son système de particules quelques *include files* déjà complètement décrits pour simuler plus facilement et plus rapidement les types de particules suivants:
>
> **`expl.inc`**: ce fichier crée des particules de feu et de fumée basées sur des sphères texturées plutôt que sur l'usage des médias. L'option à spécifier est **`expl_element()`**.
>
> **`fire.inc`**: ce fichier crée des particules de feu en plaçant à l'endroit de chaque particule une sphère remplie par un média. La façon de spécifier ce type d'élément présente des particularités qui seront reprises dans un encadré de la partie *Appel du système de particules*.
>
> **`glitter.inc`**: ce fichier crée des phénomènes de halo lumineux à l'endroit de chaque particule, au moyen d'une astuce basée sur des disque texturés. L'option à spécifier est **`glitter_element()`**.
>
> **`glow.inc`**: ce fichier crée des étincelles luminescentes en plaçant à l'endroit de chaque particule une sphère remplie par un média. L'option à spécifier est **`glow_element()`**.
>
> **`smoke.inc`**: ce fichier crée des particules de fumée en plaçant à l'endroit de chaque particule une sphère remplie par un média. L'option à spécifier est **`smoke_element()`**.
>
> **`water.inc`**: ce fichier crée des gouttes d'eau en plaçant à l'endroit de chaque particule une sphère ou un blob. L'option à spécifier est **`water_element()`**.
>
> Chacun de ces fichiers comporte des paramètres propres à l'effet recherché, que vous pouvez conserver tels quels ou chercher à les modifier (en particulier ceux qui caractérisent la taille de la particule au regard de la scène). Pour les modifier, deux options: soit vous reprenez les paramètres du fichier **`*.inc`** dans votre propre scène (en ce cas, assurez-vous de déclarer les variables en question après avoir #inclu le

fichier **`*.inc`** concerné, sinon vos valeurs seront remplacés par les valeurs par défaut du fichier **`*.inc`**), soit vous les modifiez directement dans le fichier **`*.inc`**. Pour des raisons de convivialité, je vous déconseille fortement la seconde méthode, surtout si vous en savez pas trop ce que vous faîtes.

p_id

Un numéro individuel d'identité pour la particule, il s'agit donc d'un nombre entier.

p_random

Un nombre au hasard entre 0 et 1 assigné à la particule courante. Si vous avez besoin d'un nombre aléatoire, utilisez celui-ci, car au sein d'une animation, vous ne pouvez pas faire appel à de véritables nombres aléatoires. Si vous avez besoin de plus d'un nombre aléatoire, basez-les sur celui-ci, ou sur le numéro individuel d'identité.

p_location

Le vecteur décrivant la position actuelle de la particule.

p_direction

Le vecteur décrivant la direction actuelle de la particule.

p_life

La durée de vie de la particule en unités d'horloge **`clock`**. En d'autres termes, le moment où elle va mourir à partir de l'instant de son émission.

p_age

L'âge courant de la particule en unités d'horloge **`clock`**.

p_birth

La valeur de l'horloge au moment de la naissance de la particule. Si l'option **`particle_cyclic`** est active, la valeur **`p_birth`** peut devenir plus grande que la valeur actuelle de l'horloge.

p_state

L'âge de la particule, exprimée par une valeur comprise entre 0.0 et 1.0, entre les moments où elle naît et où elle meurt.

p_rotate

Un vecteur rotation qui va obliger la direction **`z`** de la particule à pointer dans la direction vers laquelle se dirige la particule.

Bien évidemment, la macro **`particle_element`** peut être placée dans un fichier *include* et être appelée depuis vos scènes courantes. Rune fournit pas moins de six systèmes de particules prédéfinies prêtes à l'emploi (voir la Note, ci-avant). Par exemple, pour faire usage du **`smoke.inc`** prédéfini, vous utiliserez le code suivant:

```
#macro particle_element ()
smoke_element()
#end
```

3. Appel du système de particules

Lorsque tous les paramètres ont été spécifiés, le système de particules peut être chargé en appelant simplement la macro **`particle_system`**:

```
particle_system ("systeme_1")
```

Attention

Le système faisant appel à **fire.inc**, pour créer des flammes, en revanche, possède un code sensiblement différent, puisque vous laissez le bloc **particle_element** vierge, pour appeler la création de la flamme avec la ligne **fire_create ("flammes")**. Ainsi, vous aurez quelque chose comme:

```
#macro particle_element ()
smoke_element()
#end
particle_system ("flammes")
fire_create("flammes")
```

La chaîne de caractères transmise à la macro peut être absolument n'importe quoi, et ne sert qu'à identifier le système et à nommer les fichiers de données dans lesquels les données de particules sont stockés pendant le tracé des images de l'animation. Ce peut être particulièrement utile si vous avez plus d'un système de particules dans votre scène, auquel cas vous devriez leur donner des noms différents afin de ne pas les mélanger. Cela veut également dire que dans la console où s'affichent les messages de débogage, vous pourrez surveiller les statistiques de chaque système. La macro **particle_system** réalisera tous les calculs, et appellera les macros **particle_element** pour chaque particule active. Cela veut dire que si vous souhaitez que toutes vos particules soient placées à l'intérieur d'une **union**, la macro **particle_system** devra également être appelée du sein de cette **union**:

```
union {
   particle_system ("mon_systeme_de_particules")
}
```

De la même façon, si votre macro **particle_element** sert à créer un élément **blob**, vous devriez placer le système de particule à l'intérieur de l'objet **blob**:

```
blob {
   threshold 1.0
   particle_system ("mon_systeme_de_particules")
   texture {My_Water_Texture}
}
```

4. Tout feu tout flammes, un exemple pratique

Nous voici arrivés au bout de toute la partie technique relative à ce système de particules. Comme vous avez pu le constater, la partie la plus complexes consiste à définir l'aspect visuel des particules, mais Rune a eu la gentillesse d'en prédéfinir certains de sorte à simplifier l'usage de son système. Nous allons donc mettre à profit son travail pour réaliser une très simple animation: une bougie sur une étagère à proximité du plafond. Nous ne chercherons pas à obtenir une simulation très esthétique du feu et de la fumée, mais simplement quelque chose de visuel, de fonctionnel et de facile à comprendre. Le reste (ajustement des paramètres pour l'obtention du meilleur effet visuel) est affaire de patience et de recherche personnelle.

Nous allons commencer quelques éléments de base: les paramètres généraux, les fichiers #inclus, la caméra et la lumière.

```
global_settings {
   assumed_gamma 1.5
   max_trace_level 100
   radiosity {
      brightness 0.5
   }
}
```

Il est ici important d'augmenter la valeur de **max_trace_level**, car un grand nombre de sphères contenant des médias pouvant se superposer ou se chevaucher, des traces noires opaques pourraient apparaître dans les surfaces transparentes coincidantes, et totalement fausser le résultat visuel.

```
#include "colors.inc"
#include "woods.inc"
#include "smoke.inc"
```

```
#include "fire.inc"
```

Nous déclarons ici non seulement de quoi créer des textures pour les objets de notre scène, mais également les deux fichiers de formes et de couleur des particules prédéfinies que nous souhaitons utiliser pour notre simulation de bougie: **smoke.inc** pour la fumée, et **fire.inc** pour la flamme.

```
light_source {
   <0, 9, 0>, color White*0.2
}

light_source {
   <0,8.8,8.2>, color White*0.7
}

camera {
   perspective
   location <3, 7.5, 4>
   look_at <0, 8, 8>
```

La première source de lumière correspond à l'éclairage ambiant de la pièce. La seconde correspond à l'éclairage produit par la flamme de la bougie. Nous n'avons pas chercher à donner ici de coloration particulière à aucune des deux lumières. La caméra ne présente aucune particularité.

```
#declare Piece = box {
   <-10, -10, -10>, <10, 10, 10>
   texture {
      finish {
         diffuse 0.6
      }
      normal {
         crackle
      }
      pigment {
         color rgb <1, 1, 0.752941>
      }
      scale 0.15
   }
   translate y*0.5
   hollow
}

#declare Bougie = union {
   cylinder {
      <0, 1.5, 0>, <0, 0, 0>, 0.15
      texture {
         pigment {color White}
         finish {
            diffuse 1.0
            phong 1
            phong_size 20
         }
      }
      translate <0, 7, 9>
   }
   cylinder {
      <0, 1.7, 0>, <0, 0, 0>, 0.01
      pigment {color Black}
      translate <0, 7, 9>
   }
   translate <0, -0.1, -0.8>
}

#declare Etagere = box {
   <1.5, 0.15, 1.5>, <6.5, 0.5, 4>
   texture {
      T_Wood1
      rotate z*90
   }
   translate <-4.5, 6.4, 5.6>
}
```

```
object {Piece}
object {Bougie}
object {Etagere}
```

Le décor est rapidement mise en place: une **box** pour la pièce, une autre pour l'étagère, et deux **cylinders** pour constituer le corps blanc de la bougie, et sa mèche noire. La seule subtilité concerne la **box** constituant la pièce: pour que les médias des particules soient visibles, ils ne doivent pas être enfermés dans des objets pleins. La **box** constituant la pièce se voit donc attribuer le paramètre supplémentaire **hollow**. Nous aurions pu spécifier immédiatement ces objets grâce à des lignes **object {...}**, mais nous avons préféré les déclarer auparavant afin de les nommer. Ainsi, chaque objet nommé peut ensuite être pris en compte avec le paramètre **particle_blockobj** lors de la détection de collision.

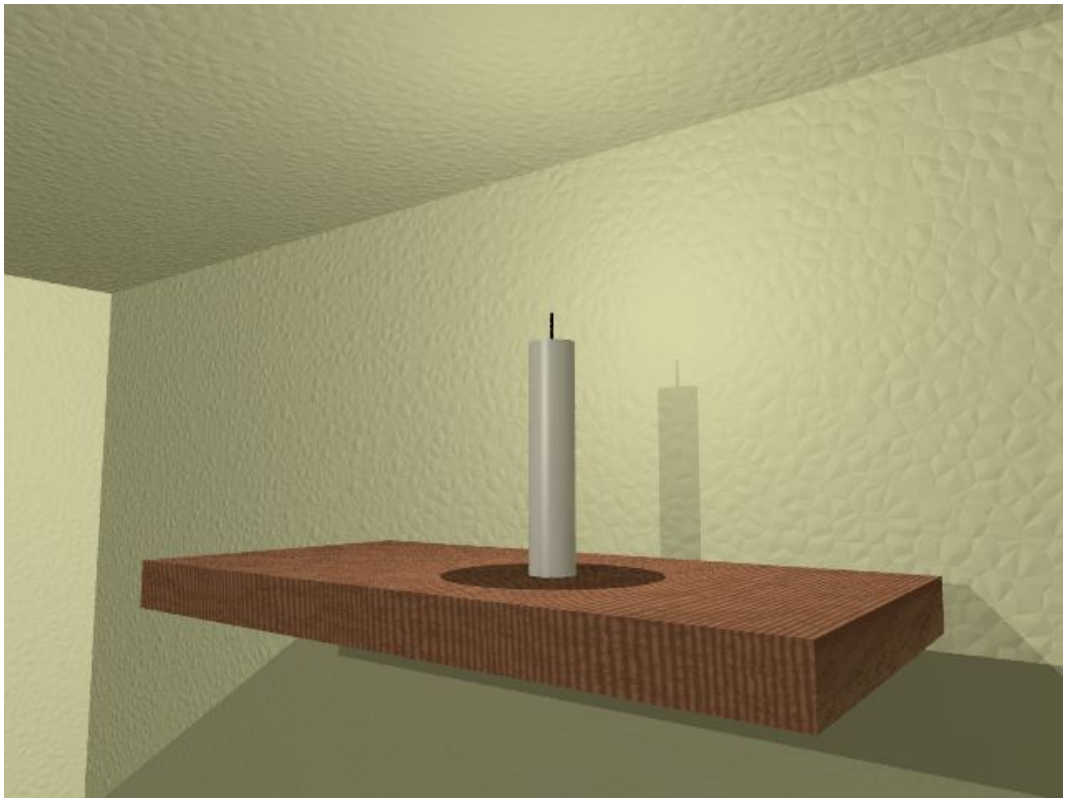

Figure 2: Le décor est planté. Seul l'éclairage trahit notre futur système de particules!

```
#include "particle.inc"


// Premier système de particules: la fumée
// ***************************************

// Paramètres d'horloge

   #declare particle_start  = -0.1;
   #declare particle_end    = 1.0;
   #declare particle_cyclic = off;
   #declare particle_steps  = 100;
```

Nous commençons la simulation du système de particules un peu avant le début de l'animation (particle_start = -0.1 pour un début d'animation à clock = 0.0.

```
// Paramètres généraux

   #declare particle_frequency = 100;
   #declare particle_life      = 1.0;
   #declare particle_lifeturb  = 0.3;
   #declare particle_seed      = 123;
```

Nous décidons que le durée de vie d'une particule peut varier de 30% (**particle_lifeturb =**

0.3) de l'une à l'autre.

```
// Paramètres de l'environnement

   #declare particle_drag      = 0.05;
   #declare particle_transfer  = 0.0;

   #macro particle_gravity  (Clock,Point) <0,0,0> #end
   #macro particle_wind     (Clock,Point) <0,0,0> #end
```

Nous ne souhaitons si vent ni gravité dans notre simulation, mais l'air oppose une légère résistance à la fumée montante: **particle_drag = 0.05**.

```
// Paramètres de l'émetteur

   #macro particle_emitter  (Clock) <0,9.0,8.2> #end
   #macro particle_emitting (Clock) on      #end
   #macro particle_emitvect (Clock) <0,15,0> #end
   #macro particle_emitturb (Clock) 0.1     #end
   #macro particle_emitobjn (Clock) 0.0     #end
```

L'émetteur de fumée est un simple point (la macro **particle_emitobj** n'est pas spécifiée) situé un peu au-dessus de la bougie; la fumée monte avec une vitesse importante (**particle_emitvect**) et avec une faible perturbation.

```
// Paramètres de collision

   #declare particle_blockobj     = object {Piece}
   #declare particle_bounce       = 0.2;
   #declare particle_bounceturb   = 0.5;
   #declare particle_friction     = 0.0;
   #declare particle_bounceoffset = 0.01;
```

Le seul objet à prendre en compte lors de la détection de collision est la **Piece** que nous avons pris soin de nommer auparavant. Ne souhaitant pas que la fumée rebondisse particulièrement contre le plafond, nous avons diminué la valeur de **particle_bounce**.

```
// Paramètres issus de smoke.inc
   #declare smoke_color      = <0.0,0.0,0.0>;
   #declare smoke_absorption = 2.0;
   #declare smoke_emission   = 1.0;
   #declare smoke_scattering = 0.0;
   #declare smoke_startsize  = 0.1;
   #declare smoke_endsize    = 0.2;
   #declare smoke_samples    = 1;
   #declare smoke_intervals  = 10;
```

Ces paramètres sont ceux qui régissent la forme et l'apparence de nos particules; ils proviennent du fichier **smoke.inc**, et non pas des paramètres du système de particules en lui-même. Dans ce cas particulier, nous avons souhaité avoir une fumée totalement noire (**smoke_color**) et que le média absorbe plus la lumière que ce qui est spécifié par défaut, afin de rendre la fumée un peu plus visible au cours de l'animation (**smoke_absorption**). Notez aussi que nous avons donner des tailles (**smoke_startsize** et **smoke_endsize**) compatibles des dimensions de notre scène.

```
// Macro particle_element
   #macro particle_element ()
   smoke_element()
   #end
```

Conformément aux instructions du fichier **smoke.inc**, nous demandons à ce que le système de particules génère la fumée grâce à l'ajout de la ligne **smoke_element()**.

```
// Appel du système de particules
   particle_system ("fumee")
```

```
// Premier système de particules: la fumée
// ***************************************
```

Nous allons passer un peu plus rapidement sur les paramètres de ce second système de particules, pour ne nous arrêter que sur les principales différences.

```
// Paramètres d'horloge

   #declare particle_start  = -0.1;
   #declare particle_end    = 1.0;
   #declare particle_cyclic = off;
   #declare particle_steps  = 100;

// Paramètres généraux

   #declare particle_frequency = 100;
   #declare particle_life      = 0.3;
   #declare particle_lifeturb  = 0.3;
   #declare particle_seed      = 123;

// Paramètres d'environnement

   #declare particle_drag      = 0.05;
   #declare particle_transfer  = 0.0;

   #macro particle_gravity  (Clock,Point) <0,0,0> #end
   #macro particle_wind     (Clock,Point) <0,0,0> #end

// Paramètres de l'émetteur

   #macro particle_emitter  (Clock) <0,8.6,8.2> #end
   #macro particle_emitting (Clock) on      #end
   #macro particle_emitvect (Clock) <0,1,0> #end
   #macro particle_emitturb (Clock) 0.1     #end
   #macro particle_emitobjn (Clock) 0.0     #end

// Paramètres de collision
// Néant, nous ne souhaitons pas de détection de collision pour la flamme

// Macro particle_element macro

#macro particle_element ()
#end
```

Nous laissons ici vide la macro particle_element (). En effet, le générateur de flammes est le seul système prédéfini à être appelé différemment des autres.

```
// Appel du système de particules

   particle_system ("flammes")

   #declare fire_method     = 1;
   #declare fire_color      = <0.85,0.45,0.15>;
   #declare fire_intensity  = 1.5;
   #declare fire_highlight  = 2.0;
   #declare fire_samples    = 1; // very low, but sometimes enough.
   #declare fire_turbulence = 0.1;
   #declare fire_size       = 0.1;
   #declare fire_stretch    = 0.05; // important to adjust to the individual scene!
```

Ces paramètres sont ceux qui régissent la forme et l'apparence de nos particules; ils proviennent du fichier **fire.inc**, et non pas des paramètres du système de particules en lui-même. A novueau, nous avons donné à nos particules une taille compatible de celle de notre scène (**fire_size** et **fire_stretch**).

```
   fire_create("flammes")
```

Et voici enfin la ligne qui permet de créer les flammes.

Il ne nous reste plus qu'à enregistrer cette scène sous un nom explicite, comme **bougie.pov**, par exemple. Et en guise de fichier **ini**, vous pouvez utilisez quelque chose comme celui-ci pour obtenir une animation très courte et très rapide, mais de dimensions et de qualité suffisante pour observer efficacement le jeu des particules.

```
Antialias=On
Antialias_Threshold=0.1
Antialias_Depth=2
Input_File_Name=particules-exemple.pov
Initial_Frame=1
Final_Frame=12
Initial_Clock=0
Final_Clock=1
Cyclic_Animation=off
Pause_when_Done=off
Width=640
Height=480
```

Et voilà, nous en avons fini de notre apprentissage des systèmes de particules!

Figure 3: La flamme est en place, la fumée également. Remarquez comme elle rebondit contre le plafond!

5. Conclusion

Nous venons de voir que la mise en place d'un système de particules n'a rien d'insurmontable. En revanche, il est évident que de solides bases avec le système d'animation de POV-ray sont quasi indispensables, de même que la connaissance du fonctionnement des médias. Il va donc vous falloir réviser un peu vos classiques avant d'entreprendre le voyage dans le monde fascinant des particules, mais au moins devriez-vous y prendre un grand plaisir.

6. Liens

La page du Système de Particules de Rune: **http://runevision.com/3d/include/particles/**

La homepage de POV-ray (version courante: v3.6): **http://www.povray.org**

La documentation officielle en français de POV-ray:
http://users.skynet.be/bs936509/povfr/index.htm

La section POV-Ray de linuxgraphic.org:
http://www.linuxgraphic.org/section3d/povray/index.html