DGPAS

Dave G’s Particle Animation System 0.95
for POV-Ray 3.0

David Govoni
david.govoni@sympatico.ca
www.geocities.com/SiliconValley/Vista/3378

A system for the creation of particles within POV-Ray without external programs or multiple source files. Particles can appear and disappear during animations, they can move, accelerate, and even bounce.

Table of Contents

Introduction and Overview

Welcome to Dave G’s Particle Animation System (DGPAS). This include file adds the ability to do basic ‘particle animation’ within POV-Ray. DGPAS is just a single include file you can use within your pov files. This should make it portable across all POV-Ray platforms.

The best way to understand the system is to walk through the tutorial section. This introduces the many features of the system one at a time. As you progress you will be introduced to all the types of particles.

What Does It Do?

DGPAS allows you to have ‘particles’ within a POV-Ray animation. You give each particle a time of birth, a time of death, and inform DGPAS what type of particle it is. For each frame POV-Ray renders in an animation DGPAS determines if the particle should exist during this frame. If the particle does exist DGPAS can place, move, or accelerate the particle. You can either have DGPAS place an object into the scene to represent the particle or you can place an object yourself.

How Does It Work?

DGPAS uses the POV-Ray parser to do all the work. In a pov file you define the particles to appear in the scene. After you define each particle you include the DGPAS file. This file makes POV-Ray process the information and do all the necessary math and logic. DGPAS does all its calculations as a pov file is being parsed just prior to rendering. There is no external program or executable. Since POV-Ray parses an pov file for each frame of an animation the DGPAS system processes all particles before each rendered frame.

To accomplish this DGPAS uses the POV-Ray clock variable (and it can use variables from Dave G’s TimeLine System as well). Since POV-Ray calculates the correct clock value for every frame DGPAS is independent of the number of frames rendered. Render an pov animation with 10 or 10,000 frames and the particles move the same way throughout the length of each resulting animation. If a particle exists for the first half of the animation it will exist for 5 frames in the 10 frame render and 5,000 frames in the 10,000 frame render.

Who's It For?

DGPAS was written for several groups. For my own use, it was written to be easily used as I create pov files by hand. Anybody who 'hand rolls' pov scenes should be able to use this file. However it was also intended for another group pov scene writers.

There are a number of POV-Ray modellers and scene builders around. Very few of them include animation and particles. This file, along with the other files in the Dave G's series are meant to give these programs the ability to create pov files with animation and other advanced features. I look at these modellers as nothing more than advanced automatic pov file generators. To create animations they must either create separate pov files for each frame or somehow convince the POV-Ray program to calculate all the animation itself. This system is an attempt to make POV-Ray do all the work for the program.

The modeller or scene builder simply has to insert the commands to animate a particle into a single pov file and let POV-Ray do all the work. This system automatically handle different numbers of frames so the generating program does not have to make a different pov file whenever the user wants to render a different number of frames. All the program has to do is insert the correct commands and call the include file.

What Is A Particle?

A particle is treated as a point object for the purposes of moving and accelerating it.

However, after DGPAS calculates the current position of a particle it will place any item you can declare into the scene automatically. If you want to place an object yourself DGPAS will just return all the information you need

What Types Of Motion Can DGPAS Calculate?

There are a number of different types of motion DGPAS handles internally.

1) No motion at all (Still_Particle) - DGPAS simply determines if the particle exists. And if it does exist DGPAS places the object into the scene.

2) Interpolated motion (Moving_Particle) - DGPAS interpolates the position of the particle between two points during its entire lifespan. The particle starts at the first point when it is created and moves to the second point where is dies.

3) Constant velocity (Velocity_Particle) - DGPAS takes a starting point and a velocity and moves the particle at a constant velocity until it dies.

4) Accelerated motion (Accel_Particle) - DGPAS takes a starting point, an initial velocity, and an acceleration. DGPAS then accelerates the particle until its death.

5) Bouncing particles (Bounce_Particle) - DGPAS takes an accelerating particle and makes it bounce back along one axis. This can be used to simulate bouncing balls or other motion.

Particles can also be constrained within boundaries on each axis. You can therefore restrict a particle within a volume of space and the particle will bounce around inside. This is called folding a particle.

How Many Particles Does DGPAS Handle?

DGPAS can handle any number of particles. Each time you define a particle in your file you include the DGPAS system. DGPAS then acts on the current parameters as a particle. If you then change the parameters and include DGPAS again DGPAS acts as if there is another unique particle. The only limitation is the amount of space in your pov file and the time you are willing to let the POV-Ray parser spend processing particles.

Tutorial

Before the tutorial starts, just two notes.

First, I am assuming you know how to render a single frame or an animation using POV-Ray. I don’t take the time the get into the details of the various command line and options.

Secondly, all animation are rendered with the +KC or Cyclical_Animation=true flag set.

Scene 1 - A Blue Sphere

To illustrate how easy the DGPAS system works start with a new pov file and enter the following (or use PAS01.POV):

```
#include "colors.inc"

camera { location <0, 3, -6> look_at <0, 0, 0> }

light_source { <20, 20, -20> color White }

plane { y, 0 pigment { checker color White color Black } }

#declare DGPASCommand = "RESET"
#include "DGPAS.INC"

#declare DGPASObject = sphere { <0, 1, 0>, 1 pigment { color Blue } }

#declare DGPASCommand      = "STILL_PARTICLE"
#include "DGPAS.INC"
```

If you render this as a single frame you will see a blue sphere floating over a checkerboard plane. Looking at the file you’ll notice that the variable DGPASCommand is set to a value of “RESET” then the DGPAS.INC file is included.

DGPAS.INC is the file containing DGPAS. All variables that DGPAS understands begin with DGPAS and then a descriptive name. The DGPASCommand variable tells DGPAS which type of particle you want. When DGPASCommand is set to “RESET”, DGPAS simply resets all variables back to their defaults.

After the call to “RESET” all DGPAS’ variables to their defaults DGPASCommand is set to “STILL_PARTICLE” and DGPAS.INC is included again. This time DGPAS processes a particle. All the variables for the particle are set to their default values. By default a particle is on for the entire animation and the object DGPASObject will be shown when the particle is alive. Since the particle is alive for the entire animation the DGPASObject, or the blue sphere, is visible in the scene.

If you render this file as an animation you will see a blue sphere sitting on a checkboard plane on each rendered frame. Again by default DGPAS shows a particle for the entire length of the animation. Let’s make it more interesting.

Scene 2 - A Disappearing Blue Sphere

Change the last two lines to: (or use PAS02.POV):

```
#declare DGPASCommand      = "STILL_PARTICLE"
#declare DGPASInitial_Time = 0
#declare DGPASFinal_Time   = 0.5
#include "DGPAS.INC"
```

Rendered as a single frame the sphere appears. If you render the new file as an animation then the sphere will appear for the first half of the frames and disappear during the second half. If you watch POV-Ray’s object count there will only be one object, the plane, during the second half of the frames.

By default the initial and final times are based on POV-Ray’s clock variable which ranges from 0 to 1 throughout the animation. However decimal fractions aren’t exactly second nature to most of us so change the lines to:

```
#declare DGPASCommand      = "STILL_PARTICLE"
#declare DGPASInitial_Time = 0
#declare DGPASFinal_Time   = 50
#declare DGPASTime_Scale   = 100
#include "DGPAS.INC"
```

If you render this new version the sphere will still appear for the first half of the animation. The DGPASTime_Scale variable multiplies POV-Ray’s clock to a value that is easier to deal with.

Otherwise nothing has changed with the particle.

Scene 3 - Moving Right Along

From now on all the scenes will be animated in some way. To see the results you must render the file as an animation of multiple frames.

So far the particles have just stayed put in the scene. This is about to change. Remove all the lines after the first include of DGPAS.INC. Then type these into the pov file (PAS03.POV):

```
#declare DGPASCommand = “MOVING_PARTICLE”
#declare DGPASStart_Point = <-2, 1, 0>
#declare DGPASEnd_Point = < 2, 1, 0>
#declare DGPASInitial_Time = 25
#declare DGPASFinal_Time = 75
#declare DGPASTime_Scale = 100
#include “DGPAS.INC”
```

Then change the declaration of DGPASObject. Currently the sphere is centred around `< 0, 1, 0>`. Since STILL_PARTICLE only places the object directly into the scene this is fine. However MOVING_PARTICLE translates the object into the scene. DGPAS assumes that the object is centered around `< 0, 0, 0>` so enter this instead of `< 0, 1, 0>`. This should look like this:

```
#declare DGPASObject = sphere { <0, 0, 0>, 1 pigment { color Blue } }
```

Render this file and after a quarter of the frames have passed the blue sphere will shift across the scene and disappear with a quarter of the frames to go.

The MOVING_PARTICLE command shifts the particle between a given starting and ending point. The motion is linear and will finish just as the particle is due to disappear. Add the following lines to the bottom of the file: (PAS03A.POV)

```
#declare DGPASStart_Point  = <-2, 1, 3>
#declare DGPASEnd_Point    = < 2, 1, 3>
#declare DGPASInitial_Time = 0
#include "DGPAS.INC"
```

Rendering this new scene will result in two particles. One appearing and starting to move before the other. But they will both end up at the same point at the same time.

After the first particle is defined and DGPAS is called, the DGPAS variables retain the values you assigned. Therefore the DGPASCommand, DGPASEnd_Point, and DGPASTime_Scale do not have to be re-declared if you want to keep their previously set values.

Scene 4 - Moving with Velocity

So far we have just created a particle that moves along a fixed straight line path. Now for a more open ended particle. Enter the following into a new pov file (PAS04.POV)

```
#include "colors.inc"

camera { location <0, 3, -6> look_at <0, 0, 0> }

light_source { <20, 20, -20> color White }

plane { y, 0 pigment { checker color White color Black } }

#declare DGPASCommand = "RESET"
#include "DGPAS.INC"

#declare DGPASObject = sphere { <0, 0, 0>, .5 pigment { color Blue } }

#declare DGPASCommand          = “VELOCITY_PARTICLE”
#declare DGPASStart_Point      = <-3,1,1>
#declare DGPASInitial_Velocity = <3,1,1>
#include “DGPAS.INC”
```

When you see the results you’ll find a blue sphere moving towards the right, and upwards, and away from you. The particle is alive for 1 unit of time. Remember the default DGPASInitial_Time and DTPASFinal_Time values are 0 and 1. The DGPASTime_Scale is also set to 1 by default.

Therefore DGPAS calculates the particle’s position as if it moves for 1 unit of time with a velocity of <3,1,1>.

Change the final lines to:

```
#declare DGPASCommand          = “VELOCITY_PARTICLE”
#declare DGPASStart_Point      = <-3,1,1>
#declare DGPASInitial_Velocity = <3,1,1>
#declare DGPASTime_Scale       = 2
#include “DGPAS.INC”
```

The particle will only appear for the first half of the animation. And it will travel the same distance before it disappears. The particle is still alive from time 0 to time 1, but with the change in the time scale that happens during the first half of the animation.

Now make change those lines to:

```
#declare DGPASCommand          = “VELOCITY_PARTICLE”
#declare DGPASStart_Point      = <-3,1,1>
#declare DGPASInitial_Velocity = <3,1,1>
#declare DGPASTime_Scale       = 3
#declare DGPASFinal_Time       = 2
#include “DGPAS.INC”
```

Now, the particle will travel twice as far. Since it is alive for a duration of 2 (from 0 to 2) it will move twice the distance of its initial velocity. Since the time scale for this particle is 0 to 3, the particle appears for the first two thirds of the rendered frames.

Scene 5 - Lets Speed Up The Action

Now let’s create a particle that is accelerated over time. Create a pov file that contains the following (PAS05.POV)

```
#include "colors.inc"

camera { location <0, 3, -6> look_at <0, 0, 0> }

light_source { <20, 20, -20> color White }

plane { y, 0 pigment { checker color White color Black } }

#declare DGPASCommand = "RESET"
#include "DGPAS.INC"

#declare DGPASObject = sphere { <0, 0, 0>, .5 pigment { color Blue } }

#declare DGPASCommand          = "ACCEL_PARTICLE"
#declare DGPASStart_Point      = <-3,.5,0>
#declare DGPASInitial_Velocity = <0,0,0>
#declare DGPASAcceleration     = <1,0,0>
#declare DGPASTime_Scale       = 3
#declare DGPASFinal_Time       = 3
#include "DGPAS.INC"
```

If you render this you will see my standard blue sphere sitting, at first, at the left of the image. Then, as you move through the animation, the sphere will start to move towards the right. As the animation progresses the sphere travels faster and faster.

This is you would expect. The particle starts with no initial velocity and is accelerated over time. In this case the particle lives for 3 units of time. During the first unit (or the first third of its life) the particle will accelerate to a speed of <1, 0, 0> and will have travelled <0.5, 0, 0>. By the end of the animation the particle will have reached a velocity of <3, 0, 0>.

Just like velocity_particles, accel_particles depend on their lifespans and the time scale to determine how far they will travel. In the case of the accel_particle it will not only move over time, but accelerate.

Scene 6 - Folding Up A Particle

Create the following pov file and render it. (PAS06.POV)

```
#include "colors.inc"

camera { location <0, 3, -6> look_at <0, 2, 0> }

light_source { <20, 20, -20> color White }

plane { y, 0 pigment { checker color White color Black } }

#declare DGPASCommand = "RESET"
#include "DGPAS.INC"

#declare DGPASObject = sphere { <0, 0, 0>, .5 pigment { color Blue } }

#declare DGPASCommand          = "VELOCITY_PARTICLE"
#declare DGPASStart_Point      = < 0,0.5, 0>
#declare DGPASInitial_Velocity = < 2,  0, 0>
#declare DGPASTime_Scale       = 5
#declare DGPASFinal_Time       = 5
#declare DGPASFold_X           = true
#declare DGPASFold_X_Min       = -3
#declare DGPASFold_X_Max       = 3
#include "DGPAS.INC"

#declare DGPASCommand          = "ACCEL_PARTICLE"
#declare DGPASStart_Point      = < 0,2.5, 0>
#declare DGPASInitial_Velocity = < 1,  0, 0>
#declare DGPASAcceleration     = < 1,  0, 0>
#include "DGPAS.INC"
```

Folding is a method to constrain the movement of a particle. A particle can be folded along each axis. By default particles are not folded unless you specify that they should be.

Folding requires two variables to be set. The axis minimum and the axis maximum. The particle will be kept within these limits.

Folding occurs after a particle is moved and accelerated normally. In this scene the two particles are moved if folding was turned off. Then DGPAS checks if a particle has to be folded.

In this case the particles are both folded along the X axis between -3 and 3.

If a particle ends up outside those limits it is folded repeatedly until it ends up within those limits.

What is folding? Well follow along for a moment.

Take the velocity_particle. It slowly moves along the X axis until it passes 3. For an example let’s say the particle has moved to a position of 3.2 along the X axis. Then DGPAS determines how far past 3 it has travelled. In this case it has travelled 0.2 to far along. So DGPAS folds the particle so that it appears 0.2 BELOW the maximum point. The particle ends up at 2.8 along the X axis.

In a later frame the particle may reach a point of 9.5 along the X axis. DGPAS determines the particle is 6.5 units passed the upper limit of 3 and folds it 6.5 units before the limit. The particle ends up at -3.5 (3 - 6.5). Then DGPAS determines that the particle is below the minimum value of -3. DGPAS determines that the particle has travelled 0.5 below the minimum and folds it so it is 0.5 ABOVE the minimum or -2.5. A particle is folded as many times as necessary to appear within the two limits.

Therefore the two particles seem to bounce between -3 and 3 along the X axis.

However, the top particle acts very strangely. It is being accelerated along the X axis. It is being accelerated towards the ‘left’ or positive direction of the X axis. But, since the folding occurs after the particle is accelerated and moved, the particle seems to continue to accelerate, but after the fold it seems to be accelerating in the opposite direction.

When the particle is moving towards the left (after the fold) it seems to be accelerating towards the left. When the particle is moving towards the right (after the fold) it seems to be accelerating towards the right.

If we are using the particle to simulate an object that can accelerate itself (A spaceship caught in a force field) then this is the intended effect. But if we are trying to simulate an external acceleration (such as gravity) then this doesn’t work. If gravity is accelerating a particle, then even when the particle folds the acceleration should be in the same direction.

But folding a particle doesn’t take this into account. But there is a way of doing this.

Scene 7 - Bouncing A Particle

Enter the following scene into a pov file (PAS07.POV)

```
#include "colors.inc"
camera { location <0, 6, -8> look_at <0, 4, 0> }
light_source { <20, 20, -20> color White }

plane { y, 0 pigment { checker color White color Black } }

#declare DGPASCommand = "RESET"
#include "DGPAS.INC"

#declare DGPASObject = sphere { <0, 0, 0>, .5 pigment { color Blue } }

#declare DGPASCommand          = "BOUNCE_PARTICLE"
#declare DGPASBounce_Plane     = "Y"
#declare DGPASBounce_Point     = 0
#declare DGPASStart_Point      = < 0, 7, 0>
#declare DGPASInitial_Velocity = < 0, 0, 0>
#declare DGPASAcceleration     = < 0,-1, 0>
#declare DGPASTime_Scale       = 12
#declare DGPASFinal_Time       = 12
#include "DGPAS.INC"
```

One of my many trained blue spheres appears over the standard checkerboard and starts to drop towards it. It accelerates as it drops towards the ‘floor’. When it reaches the checkerboard it bounces upwards.

However as it bounces upwards the particle is still accelerated ‘downwards’. It slows in midair exactly where it began and it starts accelerating towards the checkerboard.

Unlike the folding of a moving or accelerating particle, bouncing occurs as a particle is being moved and not afterwards. A particle is folded after its new position in space has been calculated. A particle is bounced by DGPAS as many times as necessary. And each time it is bounced the particle is still accelerated in the correct direction. The particle changes the direction of its motion (in this case from down to up) but the acceleration still pulls it in the same direction.

To make the math bearable particles can only be bounced off of one axis. This axis is chosen using the DGPASBounce_Plane variable. In this case it has been set to “Y”. This particle will bounce back and forth along the Y axis.

The DGPASBounce_Point tells DGPAS where the ‘ground’ is. If you render this animation you will notice that the sphere goes halfway into the checkerboard before it bounces. This is because the particle will bounce when its Y value reaches 0 (DGPASBounce_Point), but that value represents the middle of the sphere being rendered. To make the bounce more realistic we have two choices.

1) We can set DGPASBounce_Point to 0.5 (or the radius of the sphere). The particle will now bounce off a plane 0.5 units above the checkerboard.

2) Or we can set the plane down 0.5 units so it intersects the bottom of the sphere as the sphere bounces.

In this simplified example the particle has no initial velocity and no acceleration along any other axis. There is no limit to the initial velocity or acceleration we can place on the particle. For example change the values to (PAS07A.POV)

```
#declare DGPASBounce_Point     = 0.5
#declare DGPASStart_Point      = < -3,  7, 0>
#declare DGPASInitial_Velocity = <  0,  0, 0.3>
#declare DGPASAcceleration     = <0.1, -1, 0>
```

Now the particle is being accelerated along the X and Y axis and the particle has an initial velocity along the Z axis. (It could have velocities and accelerations along them all).

And the particle performs as you might expect. It bounce off of a point 0.5 units above the checkerboard, so the bottom of the sphere seems to bounce off of the checkerboard plane. The particle slowly moves away from the viewer. It also is accelerated towards the left and its speed along this axis slowly increases.

So far all is well. Except that our particle is perfect. Each bounce returns it to the same height it started. In reality a ball does not bounce as high as it was when it was dropped.

To simulate this loss of bounce height there is another variable you can set. It is called DGPASResilience. When the particle bounces its velocity is reversed along the bounce axis. In this case the particle is reversed along the Y axis. Instead of moving downwards the particle moves upwards. The particles velocity along the X and Z axis is unchanged. After the velocity is bounced the particles total velocity is multiplied by DGPASResilience. This value defaults in as 1.0.

Therefore the particle keeps moving at the same speed after it bounces. Just its direction changes.

If you add a line just before the second #include “DGPAS.INC” that reads: (PAS07B.POV)

```
#declare DGPASResilience = 0.8
```

And render the animation again you will see the particle does not bounce as high as it did previously.

However DGPAS does not try and simulate all the variances possible in reality. Instead it has to simplify the world somewhat. The velocity along all axis (X, Y, and Z) is multiplied by this factor.

So when the particle bounces its Y velocity is smaller, and since the velocity along the Y axis is still the same, the particle bounces a smaller bounce. The acceleration along the Y axis is not diminished. DGPASResilience only alters the velocity, not the acceleration.

Along the Z axis its velocity is also lessened. Instead of moving with a velocity of 0.3 it moves at a slower rate.

Along the X axis the particle is also slowed. After the slowdown the particle still is accelerated along the X axis at the same rate it was previously. DGPAS makes no attempt to simulate rolling friction (try bouncing a super ball, you can throw it away from yourself and it will return towards you thanks to the spin. DGPAS can not do this). DGPAS does not simulate variances in the resilience factor. (If a particle is moving very fast along the X axis and only at a minute velocity along the Y axis, it will slow down the same proportion along both axis. This may not simulate reality too well, but I had to draw the line somewhere!).

Therefore, in extreme cases, or with objects that would modify the bounce, DGPAS will not do a good job of simulating the particle’s bounce. But for simpler particles and less extreme cases DGPAS works fine.

You can’t fold a particle along its bounce axis. The two forms of position modification are mutually exclusive. You can fold a particle along the two axis other than the bounce axis. You can bounce a particle along the Y axis and fold in along its X and Z. (Simulating a particle dropped into a rectangular box, for instance). However the folds off of the X and Z axis will be perfect ‘bounces’ with no loss of velocity. The particle will only slow down when it bounces along Y axis.

One more note about bouncing a particle. There is one more variable that can change how a particle acts. It is DGPASBounce_Min. Each time the particle is bounced DGPAS multiplies the velocity by the resilience factor. It then compares the resulting velocity along the bounce axis to

DGPASBounce_Min. If the velocity is less than this minimum then DGPAS does not bounce the particle any more. Instead it decides that the particle has stopped bouncing and will no longer move along the bounce axis.

DGPASBounce_Min defaults to 0.01. DGPAS determines that if the particle is this slowly after a bounce it might as well stay put at the bounce point. This provides a way for a particle to come to a complete stop, and provides a way for DGPAS to avoid calculating an incredible number of very small bounces.

Keep in mind that this only works along the bounce axis. If DGPAS determines that a particle will have stopped bouncing in the time before the current frame, DGPAS will continue to move and accelerate the particle along the other axis even after it has stopped bouncing.

Bouncing isn’t perfect. But, if used properly, it can add a sense of realism to a particle’s motion.

Scene 8 - My Particle or Yours?

DGPAS has one last major option concerning a particle. So far DGPAS has been placing an object into the scene for us. All the demonstrations have an object declared as DGPASObject. If DGPAS determines the particle should survive then DGPAS places this declared object into the scene at the appropriate spot.

However there are times in which you may not want DGPAS to place an object for you. You can, for example, use a particle to give a position for the camera. Or you may want to use a particle’s position as part of a more complex object.

In these cases you can have DGPAS calculate the existence and position of a particle without having it placed in the scene.

```
#include "colors.inc"

camera { location <0, 3, -6> look_at <0, 0, 0> }

light_source { <20, 20, -20> color White }

plane { y, 0 pigment { checker color White color Black } }

#declare DGPASCommand = "RESET"
#include "DGPAS.INC"

#declare DGPASCommand = "MOVING_PARTICLE"
#declare DGPASObject_Type = "USER"
#declare DGPASStart_Point = <-2, 1, 0>
#declare DGPASEnd_Point = < 2, 1, 0>
#declare DGPASInitial_Time = 25
#declare DGPASFinal_Time = 75
#declare DGPASTime_Scale = 100
#include "DGPAS.INC"
#if (DGPASParticle = true)
      sphere { DGPASCurrent_Point, 1 pigment { color Blue } }
#end
```

This is the moving particle tutorial scene 3 reworked. (PAS08.POV).

There are several changes. First, the declaration of DGPASObject has been removed. Second, the variable DGPASObject_Type has been set to “USER”. And lastly, if the DGPASParticle variable is true, a sphere has been placed into the scene at the location DGPASCurrent_Point.

The DGPASObject_Type variable determines if DGPAS places an object into the scene. If it is set to “USER” then no object is placed. If it is set to “DECLARED” then the DGPASObject declaration is placed into the scene and translated to the current position of the particle.

In both cases, whether or not an object is placed, there are a few variables returned by DGPAS that you can use. DGPASParticle is a boolean that returns true if the particle is alive. DGPASCurrent_Position is a vector that returns the position of the particle. (It is usually set to <0,0,0> if the particle is not alive. There are a few more listed in the reference section of this manual.

You can use these values in any way you wish. You are not forced to let DGPAS place a particle in the scene. Therefore you can use these values as part of an object, a look_at vector, a light_source position, or any other idea you can come up with.

And there, in a nutshell, is DGPAS. Hopefully you have an understanding of how DGPAS works and most of its features. From now on all that is left is to try it in your own scenes.

Goo luck rendering!

Reference

A little note about the variables used by DGPAS. All start with DGPAS and then have properly capitalized names. This may seem like extra typing but it was done to avoid conflict with other POV-Ray files and objects.

All variable names in the Dave G’s series start with “DG” and then more letters to indicate which part of the Dave G’s System it is part of. All internal variables in these files follow this format as well.

Unless you happened to name your variables with these letters (which I hope is highly unlikely) there should be no conflict between the Dave G’s series and any other POV-Ray files.

Particle Types and Commands

All these commands are declared into the DGPASCommand variable before the DGPAS.INC file is included. DGPAS reads the command and performs the appropriate action. In all but one case the command instructs DGPAS what type of particle it should process.

The command can be set to a different value before each time you include DGPAS. This way each particle can be of a different type.

Reset

This command instructs DGPAS not to calculate a particle. Instead DGPAS sets all its variables back to their defaults. See the section on Default Values for a reference on what all variables are set to.

It is usually used in two places.

First, at the beginning of a scene file that will use DGPAS. Technically the reset command does not have to be sent to DGPAS before the first particle is created. If you have not declared a variable, and DGPAS has not been called before in a particular file, then the remaining variables are set to their defaults before the particle is processed.

However, since the reset command sets all the variables to their defaults, it also set all the variables to the correct type. (Floats are setup as floats, vectors are setup as vectors). As a result if you try to set a float variable with a vector arguement afterwards POV-Ray will stop processing and flag you with an error.

Secondly it is used to clear the DGPAS system between particles. In some of my example files I set the DGPAS variables to the values I want and then include DGPAS. Then I change only the variables that need to be changed from the previous include. I can be sure that the other variables are not changed by DGPAS or by myself.

However, if I'm in a complicated pov scene with multiple inc files I tend to call the reset command before the first particle in an include file. Since I'm never sure what I may have set the DGPAS variables too in another file this is an easy way to set them all to known values. Then I can simply change the variables that must be different than the default.

NOTE: For computer generated pov scenes it is very easy to use the reset command before each particle. Processing the DGPAS file for a reset command is very quick. And it makes the programs job easier.

Still_Particle

This is the simplest form of particle. DGPAS determines whether the particle exists for the current frame but does not move it. DGPAS will place a DGPASObject in the scene if the DGPASObject_Type variable tells it to. But it will not move the particle from its current position.

After you create the particle by including DGPAS.INC you can use all the standard returned variables. See the section below. Note that the velocity will be <0,0,0> and the positition will be <0,0,0>.

Moving_Particle

The next simplest particle type. DGPAS first determines if the particle exists for the current frame and then moves the particle linearly between a starting point and an ending point.

For example:

```
#declare DGPASCommand = “MOVING_PARTICLE”
#declare DGPASStart_Point = <0,0,0>
#declare DGPASEnd_Point = <10,10,10>
```

The particle will move directly between the two points at a fixed velocity during its entire lifespan. If the particle is one quarter through its life span for the current frame the particle will be one quarter along the path between the starting point and the ending point.

Velocity_Particle

The velocity particle is given an initial position and an initial velocity. It then moves away from its start point. How far it travels depends on the time scale used. The particle moves one value of its initial velocity vector for each unit of time passed in its lifetime.

For example

Particle 1:
```
#declare DGPASInitial_Time     = 0
#declare DGPASFinal_Time       = 50
#declare DGPASTime_Scale       = 100
#declare DGPASInitial_Velocity = <1,0,0>
```

Particle 2:
```
#declare DGPASInitial_Time     = 0
#declare DGPASFinal_Time       = 0.5
#declare DGPASTime_Scale       = 1
#declare DGPASInitial_Velocity = <1,0,0>
```

Particle 3:
```
#declare DGPASInitial_Time     = 0
#declare DGPASFinal_Time       = 0.25
#declare DGPASTime_Scale       = 0.5
#declare DGPASInitial_Velocity = <1,0,0>
```

If each particle is particle is halfway through its lifespan then the first particle will move 25 units along the X axis, the second 0.25 units, and the third 0.125. The details for all three particles are:

Particle 1:
```
Lifespan           = 0 through 50 units of time
Half of Lifespan   = 25 units of time
Distance travelled = 25 units of time * velocity <1,0,0>
Distance travelled = <25,0,0>
```

Particle 2:
```
Lifespan           = 0 through 0.5 units of time
Half of Lifespan   = 0.25 units of time
Distance travelled = 0.25 units of time * velocity <1,0,0>
Distance travelled = <0.25,0,0>
```

Particle 3:
```
Lifespan           = 0 through 0.25 units of time
Half of Lifespan   = 0.125 units of time
Distance travelled = 0.125 units of time * velocity <1,0,0>
Distance travelled = <0.125,0,0>
```

Notice that in each case the particles will live on screen for half of the frames of the animation. Each particle is alive for the first half of its time scale. So while these particles will appear onscreen for the same length of time they will move vastly different amounts. The time scale affects all the moving particles this way (except the MOVING_PARTICLE ans STILL_PARTICLE).

Accel_Particle

The Accelerated particle is given an initial position, an initial velocity, and an initial acceleration. During its lifespan it moves from its position to a new spot based on its velocity, acceleration, and the amount of the time scale that has passed (see the section on VELOCITY_PARTICLEs).

For example:

```
#declare DGPASCommand          = “ACCEL_PARTICLE”
#declare DGPASStart_Point      = <5,5,5>
#declare DGPASInitial_Velocity = <1,1,1>
#declare DGPASAcceleration     = <-1,-1,-1>
```

If you declare the acceleration to be <0,0,0> then the particle will act as a VELOCITY_PARTICLE.

Bounce_Particle

Bouncing particles bounce off of one axis. You have to define which axis, and which point along that axis, the particle will bounce off of. See the tutorial section for a detailed explanation.

For example:

```
#declare DGPASCommand      = “BOUNCE_PARTICLE”
#declare DGPASBounce_Plane = “Y”
#declare DGPASBounce_Point = 0
#declare DGPASStart_Point  = <0,5,0>
#declare DGPASAcceleration = <0,-1,0>
```

DGPAS will check to see that the particle is accelerating towards the bounce point. If it isn’t then the particle will be treated as an ACCEL_PARTICLE to avoid the extra calculations.

Time Types

DGPAS determines whether a particle should be present by using the DGPASInitial_Time and DGPASFinal_Time variables. What it compares these times to depends on the DGPASTime_Type variable.

By default DGPAS bases its time calculations on the POV-Ray internal clock variable. But it can use some of the variables supplied by Dave G's TimeLine system (DGTL). You can have particles appear only during the course of certain TimeLine segments and move and accelerate depending on the duration of the segment. If you try to use the time types that need DGTL without using the DGTL system the particles will simply never appear.

All time types use the DGPAS time scale to modify the appropriate clock variable.

Clock

When DGPASTime_Type is set to "CLOCK" the existence of the particle is checked against POV-Ray's clock variable. This variable ranges from 0 to 1 as an animation is being calculated. DGPASTime_Scale can be used to increase this to a manageable set of numbers.

A particle's existence is completely dependant on the POV-Ray clock variable. Even if there is a TimeLine in the animation the particle will ignore it.

Segment_Clock

This option allows you to specify a particles life against the DGTLSegment_Clock variable. Within a TimeLine DGPASSegment_Clock ranges from 0 to 1 during each segment of the TimeLine. DGPAS uses the DGPASTime_Scale variable to make the range more manageable.

DGPAS also uses the DGPASSegment variable to limit the segment(s) in which the particle will appear. If DGPASSegment is set to 12 then the particle will only appear during segment 12. If DGPASSegment is set to 0 then the particle will appear in every segments of the animation (as long as the initial and final times allow).

Segment_Time

Within a TimeLine each segment is given a duration. During the rendering of a frame DGTL calculates which segment is currently active. It also determines how far through the segment the frame is. This amount is returned in DGTLSegment_Clock, which ranges from 0 to 1. Since

DGTL also returns the duration of the current segment DGPAS can also calculate how long into the current segment the current frame is.

If the current segment is has a duration of 10 then using the Segment_Time time type will give a time that ranges from 0 to 10 instead of 0 to 1. (Before the DGPASTime_Scale variable is multiplied in. If the time scale is 2 then the result will be a Segment_Time that ranges from 0 to 20, not 0 to 10).

Take the following pov file fragment

```
#declare DGTLSegment04 = 20
#include “DGTL.INC”
      ...
#declare DGPASTime_Type    = “SEGMENT_TIME”
#declare DGPASSegment      = 4
#declare DGPASTime_Scale   = 3
#declare DGPASInitial_Time = 10
#declare DGPASFinal_Time   = 50
#include “DGPAS.INC”
```

The particle above will only appear based on the DGTL TimeLine. This is because DGPASTime_Type is set to “SEGMENT_TIME”.

It will only appear during segment 4 of the animation since DGPASSegment is set to 4.

DGPAS will base the particle’s existence on the duration of segment 4 which is 20 (DGPASSegment04) and the time scale of 3 (DGPASTime_Scale). During the course of segment 4 DGPAS will compare the particles initial and final time to the range 0 through 60. 60 being a duration of 20 multiplied by a time scale of 3.

This particle will appear one sixth of the way into segment 4 and will disappear after five sixths of segment 4. (10 through 50 of 0 through 60).

Master_Clock

You can also have a particle appear based on the total duration of a TimeLine. With the time type of Master_Clock DGPAS checks the particle’s existence against the DGTLMaster_Clock variable multiplied by the time scale. DGPASMaster_Clock ranges from 0 to the total duration of the animation. This allows particle’s to appear and disappear regardless of which segment is currently being animated.

Other Particle Options

Folding Particle

Particles can be constrained along any number of axis. After a particle’s position is calculated the particle is ‘folded’ back and forth between a maximum and minimum value until it is within the constrained range. A detailed explanation is in the tutorial section.

Here is an example of a particle constrained within a box. (See PAS09.POV for another example).

```
#declare DGPASFold_X     = true
#declare DGPASFold_X_Min = -5
#declare DGPASFold_X_Max =  5
#declare DGPASFold_Y     = true
#declare DGPASFold_Y_Min = -5
#declare DGPASFold_Y_Max =  5
#declare DGPASFold_Z     = true
#declare DGPASFold_Z_Min = -5
#declare DGPASFold_Z_Max =  5
```

The DGPASFold_n variables are simple booleans that determine whether DGPAS should fold a particle along that axis. The _MIN and _MAX variables for each axis determine the limits of the contraint.

User Assigned Variables

These are the variables you set to define when a particle lives, how a particle moves, and if DGPAS places the particle into the scene automatically.

DGPASCommand - string

See the section on particle types for a listing of all possible values.

DGPASObject - object

If DGPAS has been instructed to place an object into the scene if a particle exists, DGPASObject is the particle it places. You can define any object as DGPASObject and it will be placed automatically.

For moving particles DGPAS assumes the object starts at <0,0,0> and translates DGPASObject to its correct location.

DGPASInitial_Time - float

When DGPAS checks for the particles existence it uses DGPASInitial_Time as the point in time which the particle will appear. What this value is compared to depends on the time type (see section above).

DGPASFinal_Time - float

When DGPAS checks for the particles existence it uses DGPASFinal_Time as the point in time which the particle will disappear. What this value is compared to depends on the time type (see section above).

DGPASTime_Scale - float

When DGPAS compares the initial and final times for a particle against other variables it multiplies those other variables by DGPASTime_Scale.

For example, if you are comparing times against the POV-Ray clock variable (which ranges from 0 to 1) you would have to use decimal values times within that range. To have a particle appear after the first quarter and disappear after the third quarter of an animation you would have to set initial and final times to 0.25 and 0.75.

If you set DGPASTime_Scale to 100 DGPAS will multiply the clock variable by 100. This gives a range of 0 through 100 which is more manageable.

DGPASTime_Type - string

See the section above on time types for details on possible settings for DGPASTime_Type.

DGPASSegment - float

If you are using a time type dependant on Dave G’s TimeLine System (DGTL) you can specify which segment the particle will appear during.

If DGPASSegment is set to a value of 10 then DGPAS will only check to see if the particle exists during segment 10 of the TimeLine. If DGPASSegment is set to 0 then DGPAS will check for the particle's existence during each segment.

DGPASObject_Type - string

There are two values for DGPASObject_Type:

1) "DECLARED" - the default value

With this option selected DGPAS will place an object into the scene automatically. It places the object declared as DGPASObject. And before it places it into the scene it moves it to the point where the particle has moved to.

2) "USER"

DGPAS will not place an object into the scene when this option is selected. Instead it is up to the user to place an object at the appropriate point. (See Returned Variables below).

DGPASStart_Point - vector

This is the initial position of a particle. This is ignored by the Still_Particle type which always returns a position of <0,0,0>.

DGPASEnd_Point - vector

This is only used by the Moving_Particle type. A Moving_Particle will be linearly translated between DGPASStart_Point and DGPASEnd_Point during its lifespan.

DGPASInitial_Velocity - vector

This is the velocity given to a particle before it appears in the scene. This is the initial motion before a particle is accelerated or bounced.

It is used by the Velocity_Particle, Accel_Particle, and the Bounce_Particle types.

DGPASAcceleration - vector

This value is the acceleration placed on a particle over time. It is used by the Accel_Particle and the Bounce_Particle types.

DGPASBounce_Plane - string

For a Bounce_Particle this is the axis on which the bouncing will occur. For example, set to “Y” a particle will be bounced up and down the Y axis.

The name refers to the type of plane the particle will bounce off of. Since the particle can be moving all over the scene it will appear to bounce off an infinite plane, not a single axis.

DGPASBounce_Point - float

For a Bounce Particle this is the point on its bounce axis it will bounce off of.

DGPASBounce_Min - float

This is used to determine when a Bounce_Particle stops bouncing. After each bounce DGPAS checks to see if the particle’s velocity along the bounce axis is greater than DGPASBounce_Min.

If the velocity is higher than this minimum then DGPAS will continue to bounce the particle. If the velocity is less than this minimum then DGPAS assumes the bounces are too small to see (and would cease in the real world due to friction) and lets the particle sit at the bounce point along its bounce axis.

The particle will continue to move and accelerate along the other axis as necessary. But after it stops bouncing it will sit at the bounce point along the bounce axis.

DGPASResilience - float

This variable is used by DGPAS to determine how high a Bounce_Particle should bounce. DGPAS simply multiplies the particles velocity by this resilience factor at each bounce.

DGPASResilience defaults as 1, In which case the particle bounces to the same height after every bounce. Setting it to a value less than 1 causing decaying bounces. Setting the value to greater than one gives a super bounce that goes higher than the previous one.

DGPASResilience multiplies the velocity of the particle. If a particle's velocity is cut in half (DGPASResilience = 0.5) it will bounce to a height much less than 50% of the previous bounce. I find values in the 0.75-0.95 range work well.

DGPASFold_X, DGPASFold_Y, DGASFold_Z - boolean

You can constrain a particle along any axis (see the tutorial section, scene 6, for more details). These three variables determine if a particle should be folded along each axis. They default to false.

DGPASFold_n_Min, DGPASFold_n_Max

This variables tell DGPAS which points to fold particles along. There is a _Max and a _Min for each axis. (For the X axis there is DGPASFold_X_Min and DGPASFold_Y_Min for example). The tutorial section above has a much more detailed explanation of folding a particle.

Returned Variables

After DGPAS is included in a pov file it returns its calculations in these variables.

DGPASParticle - boolean

If the particle was determined to exist DGPASParticle is set to **true**. Otherwise it is set to **false**.

DGPASCurrent_Point - vector

If a particle exists DGPAS returns its position in this variable.

DGPASParticle_Life - float

If a particle is found to exist DGPAS returns the proportion of the particle's life that has passed (from 0 to 1) in DGPASParticle_Life.

DGPASCurrent_Velocity - vector

If a particle exists DGPAS calculates its velocity and returns it in DGPASCurrent_Velocity. This velocity is calculated based how far through the particles lifespan the current frame is and how that particle has been moved, accelerated, or bounced.

Default Values for Variables

Since there are so many DGPAS variables that can be set, here is a list of their default values. When DGPAS is first used, or whenever the “RESET” command is sent to DGPAS, these are the values assigned.

You only have to declare variables that differ from these defaults. If the initial time is correct you don’t have to place it in your pov file.

This list is also meant for the use of programmers creating programs that generate pov source code such as scene builders and modellers. This is a definitive list of the values that will be set after each “RESET” command.

```
DGPASInital_Time      = 0.0
DGPASFinal_Time       = 1.0
DGPASTime_Scale       = 1.0
DGPASTime_Type        = "CLOCK"
DGPASObject_Type      = "DECLARED"
DGPASSegment          = 0
DGPASStart_Point      = <-1, 0, 0>
DGPASEnd_Point        = < 1, 0, 0>
DGPASInitial_Velocity = < 0, 0, 0>
DGPASAcceleration     = < 0, 0, 0>
DGPASBounce_Plane     = "X"
DGPASBounce_Point     = 0
DGPASResilience       = 1
DGPASBounce_Min       = 0.01
DGPASFold_X           = false
DGPASFold_Y           = false
DGPASFold_Z           = false
DGPASFold_X_Max       =  5
DGPASFold_X_Min       = -5
DGPASFold_Y_Max       =  5
DGPASFold_Y_Min       = -5
DGPASFold_Z_Max       =  5
DGPASFold_Z_Min       = -5
```