Easy Papervision3D space dust tutorial and source

by Benny Bottema - Friday, May 29, 2009

http://www.bennybottema.com/2009/05/29/easy-papervision3d-space-dust-tutorial-and-source/

Many space shooters use this concept where you are the pilot that looks out from the front window, lasering down enemies for bounty, right. Ever noticed spacedust? It's very subtle when it is used, but it adds to the realism of outer space, making the space feel less empty, less static.

Here's how you can do that in Papervision3D, using ParticleFields. I've used the trunk of the papervision code repository, revision 851, but it should work with the last release without to much hassle. Note that I'm no PV3D guru; I just started to learn this stuff and I'm just sharing what I've learned so far.

- download spacedust source

```
/wp-content/uploads/2016/02/Spacedusttutorial.swf, 600, 300
```

(note: space dust uses the skybox from the skybox tutorial)

Here's how it works:

- divide space into a 3d grid and parts of the spacedust cloud on 8 gridpoints directly around the camera (the ship), we'll call these dustpockets
- when the camera moves, dustpockets are removed and created to keep only the 8 gridpoints around the ship occupied

Divide space into a 3d grid and snap dust pockets on it

We're not actually creating a grid, but we're going to say things like: "Give me the closest 3d point behind the ship in the upper left corner on the invisible grid", so that we can create a dust pocket in that position. Sounds complicated? Luckily, the math is easy:

Let's say there's only the x-axis. Now from 0 to 100, I want the gridpoint on the left of 32 on grid with size 5. Here is how it would look:

As you can see, the x-value 30 is the value to the left of 32 on a grid with size 5. Here's the equation to get to that answer:

gridpoint left: Xgrid = Xvalue - (Xvalue % gridsize)
gridpoint right: Xgrid = Xvalue - (Xvalue % gridsize) + gridsize

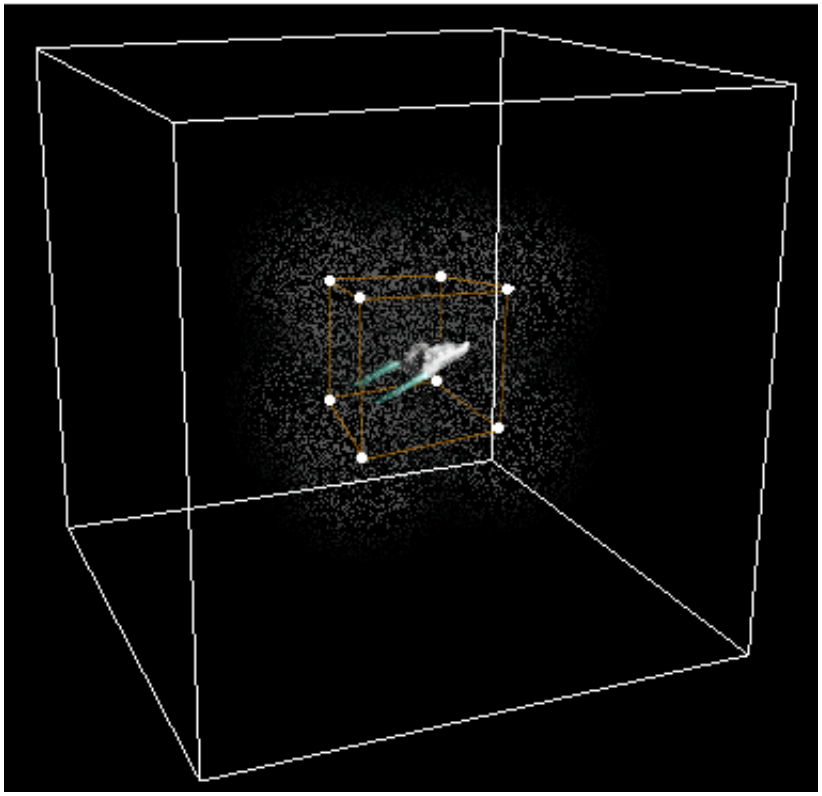
Example gridpoint left for value 32 and grid size 5:
gridpoint left: Xgrid = 32 - (32 % 5)
gridpoint left: Xgrid = 32 - (2)
gridpoint left: Xgrid = 30

Example gridpoint right for value 32 and grid size 5:
gridpoint left: Xgrid = 32 - (32 % 5) + 5
gridpoint left: Xgrid = 32 - (2) + 5
gridpoint left: Xgrid = 35

We're going to apply this kind of math to find all 8 gridpoints around the camera. So not just the x-axis, but y and z-axis as well.

The reason we need these 8 gridpoints is because we can't just create a single cloud. Since the ship can move in any direction and look in any direction, but we can't fill all of space with dust particles due to memory restrictions, we need to dynamically 'move' the cloud with the ship, while keeping the dust particles on the same place. The only way to do this is by dividing the single dust cloud into several small ones and create them as the ship goes. At the same time we need to keep the number of particles down, so we need the minimum dust pockets and still cover all sides by create 8 small clouds that together form a big cloud and can grow in any direction without draining too much memory and cpu power.

Here's how we're going to place the clouds:



Here’s the code to do this:

```
public function update(fieldsize:Number = 2500):void {
 // calculate segment (dustpocket) size
 const FIELDSIZE_POCKET:Number = fieldsize / 5;

 // determine the 8 gridpositions around the ship (eg. 1: front bottom
 left, 2: behind top right)
 const pocketPositions:Array = new Array();
 pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_POCK
ET, false, false, false));
 pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_POCK
ET, true, false, false));
 pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_POCK
ET, false, true, false));
 pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_POCK
ET, false, false, true));
 pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_POCK
ET, true, true, false));
 pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_POCK
ET, true, false, true));
 pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_POCK
ET, false, true, true));
 pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_POCK
ET, true, true, true));
```

```
}

/**
 * Calculates a gridpoint around the ship.
 */
private function calcGridPosition(position:Number3D, gridSize:Number,
xMin:Boolean, yMin:Boolean, zMin:Boolean):Number3D {
 const gridPosition:Number3D = new Number3D();
 gridPosition.x = (position.x - (position.x % gridSize)) + ((xMin) ? 0
 : gridSize);
 gridPosition.y = (position.y - (position.y % gridSize)) + ((yMin) ? 0
 : gridSize);
 gridPosition.z = (position.z - (position.z % gridSize)) + ((zMin) ? 0
 : gridSize);
 return gridPosition;
}
```

This is really all there is to it. The only thing we need to do now is create new dust pockets for gridpoints without one and remove dust pockets for points that are not within range anymore. Over and over again. Here's the entire Spacedust class:

```
package org.codemonkey.spacedusttutorial.papervision3d {
 import flash.utils.Dictionary;
 import org.papervision3d.cameras.Camera3D;
 import org.papervision3d.core.math.Number3D;
 import org.papervision3d.materials.special.ParticleMaterial;
 import org.papervision3d.objects.special.ParticleField;
 import org.papervision3d.scenes.Scene3D;

 /**
  * Spacedust simulator. Creates 8 segments of the dustcloud around th
e given camera at any point in space.
  * When the camera moves, new dustpockets are generated and old ones
(out of sight) are removed.
  *
  * @author Benny Bottema
  */
 public class Spacedust {

  private static const DEFAULT_PARTICLEMATERIAL:ParticleMaterial = new
 ParticleMaterial(0xffffff, .5, ParticleMaterial.SHAPE_CIRCLE);

  private var basic3DSetup:Basic3DSetup;
```

```
  private var camera:Camera3D;
  private var spacedustCloud:Dictionary;

  private var fieldsize:Number;
  private var particleCount:Number;
  private var dustMaterial:ParticleMaterial = new ParticleMaterial(0xf
fffff, .5, ParticleMaterial.SHAPE_CIRCLE);

  public function Spacedust(basic3DSetup:Basic3DSetup, camera:Camera3D
 = null, fieldsize:Number = 2500, particleCount:Number = 400, dustMate
rial:ParticleMaterial = null) {
   spacedustCloud = new Dictionary();
   this.basic3DSetup = basic3DSetup;
   this.camera = (camera != null) ? camera : basic3DSetup.camera;
   this.fieldsize = fieldsize;
   this.particleCount = particleCount;
   this.dustMaterial = (dustMaterial != null) ? dustMaterial : DEFAULT
_PARTICLEMATERIAL;
  }

  public function update():void {
   // calculate segment (dustpocket) size
   const FIELDSIZE_POCKET:Number = fieldsize / 5;

   // determine the 8 gridpositions around the ship (eg. 1: front bott
om left, 2: behind top right)
   var pocketPositions:Array = new Array();
   pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_PO
CKET, false, false, false));
   pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_PO
CKET, true, false, false));
   pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_PO
CKET, false, true, false));
   pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_PO
CKET, false, false, true));
   pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_PO
CKET, true, true, false));
   pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_PO
CKET, true, false, true));
   pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_PO
CKET, false, true, true));
   pocketPositions.push(calcGridPosition(camera.position, FIELDSIZE_PO
CKET, true, true, true));

   var newSpacedustCloud:Dictionary = new Dictionary();
```

```
   // create new dustfields or transfer existing ones (and remove exis
ting ones the scene)
   const particlesPerPocket:Number = particleCount / pocketPositions.l
ength;
   for each (var pocketPosition:Number3D in pocketPositions) {
    var pocketKey:String = pocketPosition.toString();
    var reusableDustPocket:ParticleField = spacedustCloud[pocketKey];
    // remove so we can delete the particles from the remaining partic
lefields (which means they are out of sight)
    delete spacedustCloud[pocketKey];
    // add the removed particlefield to the new dustcloud or create on
e if dustpocket is new for te current gridposition around the ship
    if (reusableDustPocket != null) {
     basic3DSetup.removeFromScene(reusableDustPocket);
     newSpacedustCloud[pocketKey] = reusableDustPocket;
    } else {
     newSpacedustCloud[pocketKey] = new ParticleField(dustMaterial, pa
rticlesPerPocket, 2, fieldsize, fieldsize, fieldsize);
     newSpacedustCloud[pocketKey].x = pocketPosition.x;
     newSpacedustCloud[pocketKey].y = pocketPosition.y;
     newSpacedustCloud[pocketKey].z = pocketPosition.z;
    }
   }

   // manually remove all the remaining obsolete particles from the ol
d cloud (they are out of sight)
   for each (var oldDustPocket:ParticleField in spacedustCloud) {
    oldDustPocket.removeAllParticles();
   }

   // now add the old dustpockets which are still within sight, plus t
he new dustpockets to the scene
   for each (var newDustPocket:ParticleField in newSpacedustCloud) {
    basic3DSetup.addToScene(newDustPocket);
   }

   // replace the old cloud with the new cloud
   spacedustCloud = newSpacedustCloud;
  }

  /**
   * Calculates a gridpoint around the ship.
   */
  private function calcGridPosition(position:Number3D, gridSize:Number
, xMin:Boolean, yMin:Boolean, zMin:Boolean):Number3D {
   const gridPosition:Number3D = new Number3D();
```

```
    gridPosition.x = (position.x - (position.x % gridSize)) + ((xMin) ?
 0 : gridSize);
    gridPosition.y = (position.y - (position.y % gridSize)) + ((yMin) ?
 0 : gridSize);
    gridPosition.z = (position.z - (position.z % gridSize)) + ((zMin) ?
 0 : gridSize);
    return gridPosition;
   }
  }
}
```

- Download Spacedust.as
- download entire spacedust source

Notice the particel fields we're using to create the eight dust pockets with.

new ParticleField(dustMaterial, particlesPerPocket, 2, fieldsize, fieldsize, fieldsize);

Also, the *Basic3DSetup* class I've used is simply a convenience class with default camera, scene, viewport etc. It's much like Papervision's own BasicView -which I didn't know about at the time- except even cleaner.

The Spacedust class accepts an optional camera to position the dust cloud around. Since this approach is meant to create the illusion of infinite space dust, it doesn't make sense to create clouds on non-camera's. Also, you can specify your own particle count and dust material in the constructor of the Spacedust class. The example at the top of the page uses only the defaults.

PDF generated by Kalin's PDF Creation Station