

Snowfall: More Winter frivolity

Edward Dale

Animation Algorithms

4005-763

Professor Joe Geigel

<http://scompt.com/school/classes/animation-algorithms/snowfall>

Project Description

I plan to implement the technique for modeling falling snow described in [MMAL05]. I would like to apply it to a simple scene with the ultimate goal of an animation similar to the one that Moeslund et al. presented. I don't intend to model the accumulation of the snow at all at this point, although this may be one possible extension point for the project. The actual techniques followed involved creating a visually realistic model of a snowflake and then animating snowfall with it using a particle system.

Approach

The approach taken to model the snowflake is identical to that from [MMAL05]. A flake is considered to have a number of layers. At each layer, a constant number of triangles are created with one vertex randomly picked in the current layer and the other two vertices randomly picked from the previous layer. The number of triangles per layer is determined by whether the snow is "wet" or "dry". Wet snowflakes have a much higher density and, therefore, number of triangles per layer. The number of layers and total diameter of the snowflake are both functions of the temperature at which the flake was created. All three of these parameters (`trianglesPerLayer`, `diameter`, `numLayers`) are inputs to the flake class.

The snowfall was modeled using a particle system with Perlin noise perturbing the rotation and position of the flakes. In [MMAL05], Navier-Stokes equations were used to model a wind field, but this was determined to be too involved for this stage of the system. Using Perlin noise, each flake is moved according to noise based on its position and the current time. Also, the y-component of the position is decreased by a noise amount to simulate gravity. Once a flake reaches a certain vertical point in the scene, it is killed.

Implementation

There are only two major components of the system. There is the flake module that handles creating and rendering the flake and the particle module that animates a number of snowflakes in a snowfall simulation. A `flakemovie` module was created also to render a single flake rotating in space.

The flake module contains the Flake class, which contains a single constructor that takes the three parameters mentioned above. It also takes an additional argument of whether or not to render the layers as nested translucent spheres for illustration purposes. The constructor calls the `makeFlake` method which is the implementation of the algorithm from [MMAL05]. All vertices are generated as spherical coordinates and not converted to cartesian until the last step. As part of that last step, the vertices are assembled into a list of triangles and a triangle mesh is created using cgkit [cgk]. As the flake is represented by a single cgkit object, it has position and rotation slots that can be used to animate it.

The particle module is basically a particle system that renders flakes instead of points. It is based off of the code from the particle system assignment in class. Each flake starts out at a random point in the $y = 40$ plane and with a random rotation axis. At each timestep, a number of particles are born and the existing particles have their position and rotation increased by an amount generated from Perlin noise. Once a particle reaches the $y = -40$ plane, it is killed.

The flake movie module was created to create an animation that mimics one created by Moeslund et al.. It renders a single flake rotating in space. The flake can either rotate at a constant speed or at a speed determined using Perlin noise.

Results

The results of the program is either a real-time animation or a rendered one depending on how it is invoked using cgkit. To execute the snowfall, the `viewer.py` script from cgkit [cgk] must be executed, passing `particle.py` as a parameter. This will show an OpenGL window with the snowfall simulation. The `render.py` script from cgkit can be used to render the scene with a RenderMan renderer. The `flakemovie.py` script can also be passed to either one of those programs to see the rotating snowflake. Still images and movies can be viewed at [sco].

Future Enhancements

The particle system still needs to be improved. Ideally, the Navier-Stokes equations would be used to move the flakes around the system. Also, accumulating the snow on the ground would be nice and would tie into my Computer Graphics 2 project, which involves deforming ground materials like snow and sand.

References

[cgk] The python computer graphics kit. <http://cgkit.sourceforge.net/>.

[MMAL05] T.B. Moeslund, C.B. Madsen, M. Aagaard, and D. Lerche. Modeling falling and accumulating snow. *Vision, Video and Graphics*, 2005.

[sco] scompt.com - the website of edward dale. <http://www.scompt.com>.

"""

Edward Dale
2006-3-1
Animation Algorithms
Snowfall Project

Copyright (c) 2005, Edward Dale
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Edward Dale nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

"""

```
from cgkit.all import *
from random import choice,random,uniform
from math import pi,sin,cos,atan,acos
from OpenGL.GL import GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA
```

```
def sphericalToCartesian(spherical):
```

"""

Convert a point from spherical coordinates to cartesian coordinates.
<http://mathworld.wolfram.com/SphericalCoordinates.html>

"""

```
x=spherical[2]*cos(spherical[1])*sin(spherical[0])
y=spherical[2]*sin(spherical[1])*sin(spherical[0])
z=spherical[2]*cos(spherical[0])
return vec3(x,y,z)
```

```
class Flake:
```

"""

http://www.cvmt.dk/%7Etbm/Student_projects/04sne.pdf

"""

```
sphereMat=GLMaterial(ambient=(1,0,0,.1),
                    diffuse=(1,0,0,.1),
                    blend_factors=(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA))
flakeMat=GLMaterial(ambient=(1,1,1,.5),
                    diffuse=(1,1,1,.5),
                    blend_factors=(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA),
                    shininess=128.0,
                    specular=(1,1,1,.5),
                    emission=(1,1,1))
```

```

def __init__(self, numLayers=5,diameter=2.0, trianglesPerLayer=40, drawSpheres=False)
    """
    Executes the algorithm from the original paper and then
    creates a triangle mesh to be rendered.
    """
    self.numLayers=numLayers
    self.diameter=diameter
    self.trianglesPerLayer=trianglesPerLayer
    self.drawSpheres=drawSpheres
    self.layerwidth=self.diameter/self.numLayers
    self.mwidth=self.layerwidth*6
    self.group=Group()

    self.makeFlake()
    verts,faces=self.getVertFace(self.complete)
    mesh=TriMesh(name="Mesh", verts=verts,faces=faces,
                 auto_insert=False, material=Flake.flakeMat)
    self.group.addChild(mesh)

def makeFlake(self):
    """
    The algorithm from the original paper.
    """
    old=[]
    new=[]
    self.complete=[]
    for k in range(1,self.numLayers):
        if self.drawSpheres:
            s=Sphere(material=Flake.sphereMat,
                    radius=self.layerwidth*k,
                    name="Sphere%d"%k,
                    auto_insert=False)
            self.group.addChild(s)
        new,old=old,new
        new=[]

        A=atan(self.mwidth/k*self.layerwidth)

        for c in range(1,self.trianglesPerLayer):
            if k==1:
                # Pick a random point on the sphere for the first point
                # of first triangle
                # http://mathworld.wolfram.com/SpherePointPicking.html
                zenith=2*pi*random()
                azimuth=acos(2*random()-1)
            else:
                # Pick a random vertex from the previous layer
                a=choice(old)
                zenith=a[0]
                azimuth=a[1]

            r=(k-1)*self.layerwidth
            new.append((azimuth,zenith,r))

            # Create the other two vertices of the triangle
            for i in range(2):
                azimuth1=azimuth+uniform(-A,A)
                zenith1=zenith+uniform(-A,A)
                r1=uniform((k-1)*self.layerwidth, (k+1)*self.layerwidth)
                new.append((azimuth1,zenith1,r1))
            self.complete.append(new)

def getVertFace(self,flake):
    """

```

```
Calculates the verts and faces necessary for the TriMesh.
"""
verts=[]
for layer in flake:
    for vertex in layer:
        verts.append(sphericalToCartesian(vertex))

faces=[]
for i in range(len(verts)/3):
    faces.append((3*i,(3*i+1),(3*i+2)))

return verts,faces
```

```
"""
```

```
Edward Dale  
2006-3-1  
Animation Algorithms  
Snowfall Project
```

```
Copyright (c) 2005, Edward Dale  
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without modification,  
are permitted provided that the following conditions are met:
```

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Edward Dale nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND  
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,  
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF  
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE  
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS  
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT  
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR  
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF  
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS  
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
"""
```

```
from flake import Flake  
from random import random
```

```
def doMovie(randomSpin=True, camscale=3,  
            drawSpheres=False, trianglesPerLayer=20):  
    """  
    Draw one flake and rotate it indefinitely.  
    """  
    flake=Flake(drawSpheres=drawSpheres, trianglesPerLayer=trianglesPerLayer)  
  
    # Lights, camera  
    light=GLPointLight( diffuse=(1,1,1))  
    cam=TargetCamera(target=(0,0,0),pos=(camscale,camscale,camscale))  
  
    # Action!  
    axis=vec3(random(),random(),random()).normalize()  
    if randomSpin:  
        e1=Expression("mat3().rotation(t+noise.snoise(t),axis)", axis=axis)  
    else:  
        e1=Expression("mat3().rotation(t,axis)", axis=axis)  
    e1.output_slot.connect(flake.group.rot_slot)  
  
doMovie()
```

```
"""
```

```
Edward Dale  
2006-3-1  
Animation Algorithms  
Snowfall Project
```

```
Copyright (c) 2005, Edward Dale  
All rights reserved.
```

```
Redistribution and use in source and binary forms, with or without modification,  
are permitted provided that the following conditions are met:
```

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Edward Dale nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND  
CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,  
INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF  
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE  
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS  
BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT  
OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR  
BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF  
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS  
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
"""
```

```
from cgkit.all import *  
from random import *  
from flake import Flake
```

```
class Particle:
```

```
    """
```

```
    Represents a particle in the system.
```

```
    """
```

```
    def __init__(self, deathCrit=-40):
```

```
        """
```

```
        Creates the snowflake and gives it a random rotation axis and a  
        noise-related rotation.
```

```
        """
```

```
        self.deathCrit=deathCrit  
        self.flake=Flake(trianglesPerLayer=3)  
        ax=vec3(random(),random(),random()).normalize()  
        e1=Expression("mat3().rotation(t+noise.snoise(t),axis)", axis=ax)  
        e1.output_slot.connect(self.flake.group.rot_slot)
```

```
    def step(self):
```

```
        """
```

```
        Steps a particle's position using noise.  
        Also determines whether it's dead yet.
```

```
        """
```

```
        pos=self.flake.group.pos  
        pos += cgkit.noise.vsnoise(self.flake.group.pos, getScene().timer().time )  
        pos.y -= cgkit.noise.noise(self.flake.group.pos)  
        self.flake.group.pos = pos
```

```
    return pos.y > self.deathCrit
```

```
class ParticleSystem:
```

```
    """  
    The particle system representing the snowfall.  
    """
```

```
def __init__(self, rate=0, dev=2, extent=40, initParticles=10):
```

```
    """  
    Creates a bunch of initial snowflakes.  
    """
```

```
    self.avggenrate=rate  
    self.genratedev=dev  
    self.particles=[]  
    self.extent=extent  
    for i in range(initParticles):  
        part=Particle()  
        self.initializeParticle(part)  
        self.particles.append(part)
```

```
def initializeParticle(self, p):
```

```
    """  
    Initializes a new particle.  
    All it does is set the the location to a random point in the  
    y=extent plane.  
    """
```

```
    p.flake.group.pos=vec3(uniform(-self.extent,self.extent),  
                           self.extent,  
                           uniform(-self.extent,self.extent))
```

```
def step(self):
```

```
    """  
    Steps the entire particle system.  
    Births new particles, kills old ones.  
    """
```

```
    newparticles=[]
```

```
    # Birth some new particles  
    numbirths=int(normalvariate(self.avggenrate, self.genratedev))  
    for x in range(numbirths):  
        i = Particle()  
        self.initializeParticle(i)  
        self.particles.append(i)
```

```
    # Kill some one ones.
```

```
    dead=0
```

```
    for i in self.particles:  
        life=i.step()  
        if not life:  
            # The particle is dead, so remove it and record another death  
            getScene().worldRoot().removeChild(i.flake.group)  
            dead += 1  
        else:  
            newparticles.append(i)
```

```
    self.particles=newparticles
```

```
# Draw the "ground"
```

```
Plane( lx=80, ly =80, pos=(0, -40, 0), material=GLMaterial(ambient=(0,1,0)),  
       rot=mat3().rotation(pi/2, vec3(1,0,0)))
```

```
seed(0)
```

```
system = ParticleSystem()
```

```
scale=50
# lights, camera, action
light=GLPointLight( diffuse=(1,1,1))
cam=TargetCamera(pos=vec3(scale,-15,scale),up=vec3(0,1,0), target=(0,-30,0))
eventManager().connect(STEP_FRAME, lambda:system.step())
```