Particle Systems

- many things have been described as particle systems
- key characteristics
  - a *collection of particles*
  - a *random element* (within limits)
    - typically randomness in particle attributes such as position, velocity, color

Particle System Evolution

- repeat
  - generate new particles and add to system
    - initialize individual attributes for new particles
  - remove particles which have exceeded their lifetime
  - move/update current particles
  - render current particles
- until done

Particle Generation

- number of new particles introduced at a given frame is random, within limits
- examples
  - random number of new particles

    `# new particles = avg + rand()*var`
  - depends on the screen area covered by object

    `# new particles = ( avg`$_{sa}$ `+ rand()*var`$_{sa}$ `)*screen area`
  - depends on the frame number

    `# new particles =` $avg_{initial} + delta*(f-f_0) + rand()*var$

avg = average # of new particles
var = variability in # of new particles
rand() = random # between -1 and 1

avg_{sa} = average # of new particles per unit of screen area
var_{sa} = variability in # of new particles per unit of screen area

$avg_{initial}$ = average # of new particles at frame f_0
delta = change in average # of new particles per frame
f = frame #
f_0 = first frame at which particles are to be generated

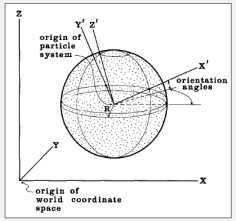
Particle Initialization

- initial values must be assigned for each attribute when particles are created
  - position and velocity (direction and speed)
  - size and shape
  - color and transparency
  - lifetime
  - path
  - …

Particle Initialization

- initial position and direction of movement
  - defined by *emitter* associated with particle system

example: spherical emitter
initial position is within sphere
initial direction is away from center



W. Reeves, "Particle Systems – A Technique for Modeling a Class of Fuzzy Objects", ACM Transactions on Graphics, April 1983

- initial speed, color, transparency, size determined randomly given average and variability

```
speed = avg + rand() * var
```


Particle Initialization

- particle shape is a parameter to particle system
  - spherical
  - rectangular
  - streaked spherical (for motion blur)
  - ...

Particle Extinction

- particles can be removed from the system for various reasons
  - lifetime is exceeded
  - particle no longer contributes to image (e.g. intensity is too low)
  - particle has moved too far from initial position
  - ...

Particle Dynamics

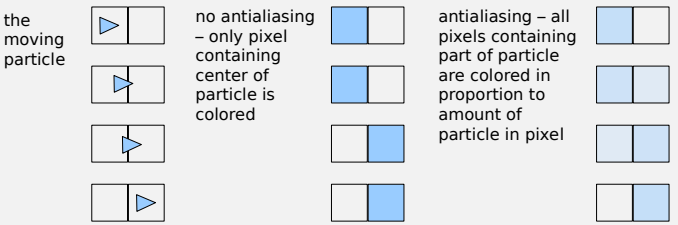
- specifies how values of particle attributes change over time
- for position, can script path or specify dynamics or behavioral rules
  - if dynamics, `position += dt*velocity`
  - ignore particle-particle collisions
- color, transparency, size change according to global rate-of-change parameters
  - `size += dt*( size rate of change )`

Particle Rendering

- simplifying assumptions
  - particles do not interact with non-particles
    - rendering algorithm only needs to consider particles
  - for fire effects, particles can be treated as point light sources
    - no need for collision detection
    - no need for visible surface determination
      - each particle contributes brightness to pixel(s) covered
    - no need to consider shadowing of particles
    - color of pixel determined only by color of particle – no direct or indirect illumination

Particle Rendering

- antialiasing is necessary
  - particles are small and overlapping – likely that multiple particles cover one pixel
  - temporal aliasing is significant for subpixel-sized objects



Temporal Aliasing

- artifacts due to evolution of scene through time
  - jerky motion of fast-moving objects near viewer
  - wagon wheels appearing to move backwards

- temporal antialiasing
  - can *supersample* and average the resulting images

Additional Complexity

- system described so far is a bare-bones particle system
- many extensions are possible
  - additional attributes
    - e.g. acceleration
    - vary rate-of-change parameters with each particle instead of being global
    - …
  - more complex emitters
  - hierarchical particle systems
  - solid particles
    - need to consider collisions – with other objects in the scene and/or other particles
  - …

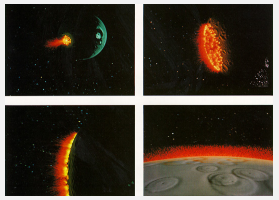
Particle Hierarchies

- each particle is itself an emitter
- parameters of parent particle are inherited (with variations) by child emitter
  - e.g. position of parent particle determines origin of child emitter
  - e.g. average color and variance of parent particle are used to pick random (within range) average color and variance for child emitter

Example: The Genesis Effect

https://www.youtube.com/watch?v=NM1r37zlBOQ

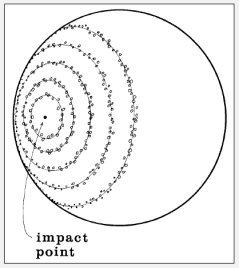
Star Trek II: The Wrath of Khan (1982)



- two-level particle system
  - first level distributes emitters around impact point on planet's surface
  - second level handles explosion out from surface

The Genesis Effect: Particles

- first level particle system distributes emitters around point on planet's surface

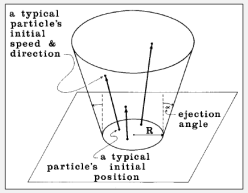


W. Reeves, "Particle Systems – A Technique for Modeling a Class of Fuzzy Objects", ACM Transactions on Graphics, April 1983

- radius of ring where emitters are produced depends on time (frame)
- number of new emitters in each ring is based on ring circumference and density parameter

The Genesis Effect: Particles

- second level particle systems handle explosion out from surface



W. Reeves, "Particle Systems – A Technique for Modeling a Class of Fuzzy Objects", ACM Transactions on Graphics, April 1983

- emitter is circle centered at first-level particle position
- number of new particles per frame depends on surface area covered by emitter
- initial velocity of particles is away from planet's surface, within cone defined by ejection angle
- parameters (average color, rate of color change, emitter radius, etc) based on parent particle system, but vary randomly

The Genesis Effect: Particle Extinction

- particles are removed when…
  - lifetime expires
  - intensity falls below minimum intensity
  - they fall below the surface of planet

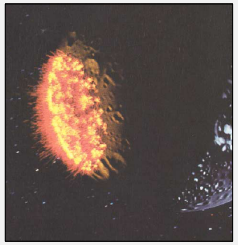
The Genesis Effect: Rendering

- color of second level particles is mostly red, with a small amount of green and blue
  - when only a few particles overlap a single pixel, pixel appears red
  - when many particles overlap a single pixel, green and blue contributions become significant
  - over time green and blue drop off faster than red

result is white hot in very center, cooling to yellow-orange and finally red towards edge

The Genesis Effect: Rendering

- motion blur
  - particles actually drawn as straight line between position at time *t* and at time *t*+*dt*/2 (halfway through frame)
- light from particles
  - planet is modeled/rendered separately
  - light source placed above impact point to simulate glow on planet's surface from rings of fire



W. Reeves, "Particle Systems – A Technique for Modeling a Class of Fuzzy Objects"

Example: A Growing Cloud

parent emitter defines where child emitters will be located and how they move

location and size of parent particles define origin and size of child emitters actual cloud particles are generated within child emitters

early on, parent particles are small and close together

child emitters are also small

newly-generated particles are within the shaded regions

how the cloud appears – particles have not moved much

- child emitters control billowing of each region of the cloud
- parent emitter/particles control relationship between cloud parts and overall cloud movement

as the system evolves, the parent particles grow and move apart

child emitters also grow larger

newly-generated particles are within the shaded regions

cloud appears filled in because older particles still exist (and are also moving)

The Adventures of André and Wally B. (1984)

https://www.youtube.com/watch?v=a_9Tsbduk9E

- *structured* particle systems used to model trees and grass

W. Reeves and R. Blau, "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems", SIGGRAPH '85

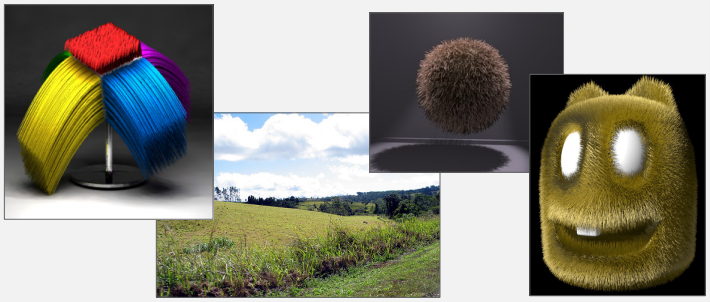
- also first CGI animation with motion blur and squash-and-stretch

Structured Particle Systems

- differences
  - particle system used primarily to describe the *shape* of the object, rather than its motion
    - render whole evolution of system at once
  - particles don't necessarily move (but they can)
  - particles are not independent
- similarities
  - hierarchical particle system (properties inherited from parent to child)
  - specific parameters chosen randomly (within limits)

Strands

- trajectory of particle defines the strand



http://en.wikipedia.org/wiki/Image:Strand_Emitter.jpg
http://www.blender.org/development/release-logs/blender-240/hair-strand-rendering/
http://www.becausewecan.org/blog/1?page=5

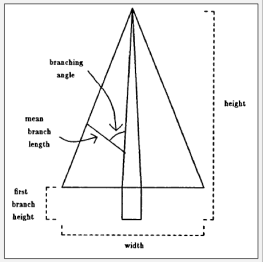
Example: Generating Trees

The Adventures of André and Wally B. (1984)

- recursive formulation…
  - trunk has branches attached
  - branches have sub-branches attached
  - sub-branches have sub-sub-branches
  - ...
- ...with hierarchical particle systems
  - top level particle system specifies parameters for whole tree, emitter for trunk, location of branches
  - next level systems specify parameters for individual branches, emitters for branches, location of sub-branches
  - …
- drawn with line segments (branches) and small circles (leaves)

Example: Generating Trees

- initial values for particle system properties are assigned randomly within range appropriate for tree type
- properties include…
  - those shown
  - distance between sub-branches
  - branching pattern



Reeves and Blau, 1985

Example: Generating Trees

- after tree particles have been generated according to type of tree, post-processing steps add realism
  - trees aren't perfect – branches missing, damaged, warped
  - branches bent according to gravity
  - prevailing winds
  - …

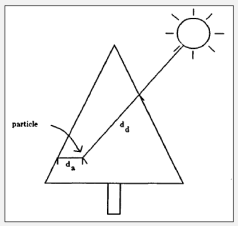
http://coastal.er.usgs.gov/navassa/hike/ep1.html

Example: Shading Trees

- tree particles reflect light instead of generating light
- need to handle
  - direct illumination (diffuse and specular)
  - self-shadowing
  - external shadowing
- the difficulties
  - too many particles (may have millions) to use normal illumination model
  - very small particles means shadow ray is unlikely to actually hit a particle

Example: Shading Trees

- solution: use probabilistic model to determine illumination and shadowing
- particles are most likely to be illuminated when near outer edge of tree



Reeves and Blau, 1985

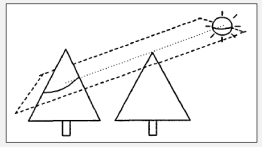
weight of diffuse component is based on d_d – the distance into the tree on the side of the light

specular highlights added randomly whenever d_d is small and branch/leaf is favorably oriented

weight of ambient component is based on d_a – the distance to the nearest edge of the tree's bounding volume (measured parallel to the ground)

Example: Shading Trees

- self-shadowing handled by adjustment of ambient component weight
- external shadowing
  - define plane containing light source and tip of neighboring tree



Reeves and Blau, 1985

particles above the plane are not shadowed – use full diffuse, specular, ambient components

probability of sunlight reaching particle (and thus if diffuse and specular components are included) decreases with distance below the plane

for particles well below plane, only ambient is used

Example: Rendering Trees

- sheer number of particles makes any scheme which involves keeping all particles in memory impractical
- strategy
  - sort trees from back-to-front (requires tree bounding volumes to not intersect)
  - for each tree from back-to-front
    - create buckets based on depth (distance from viewer)
    - as each particle is generated, sort into correct bucket
    - traverse buckets from back-to-front, painting particles in each bucket