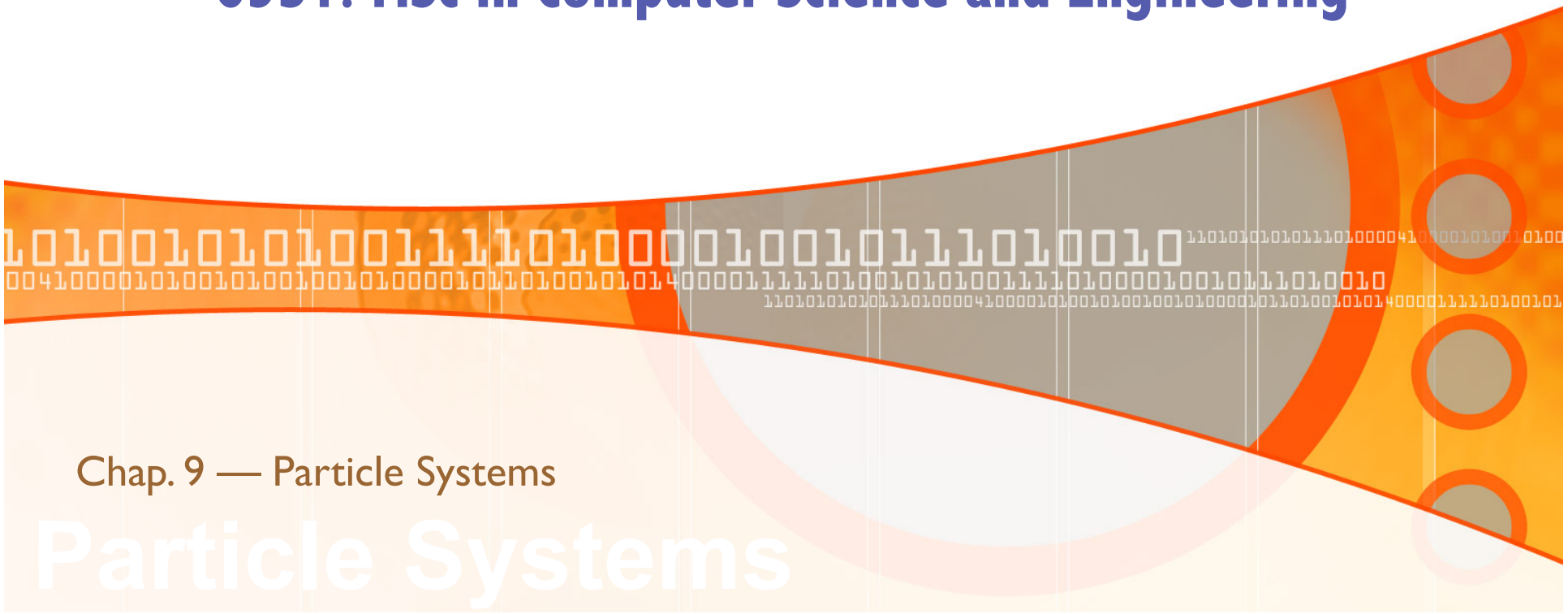
These notes were taken from a variety of sources including text book, Wikipedia, various Maya references, and SIGGRAPH articles

Video Game Technologies

6931: MSc in Computer Science and Engineering



Chap. 9 — Particle Systems

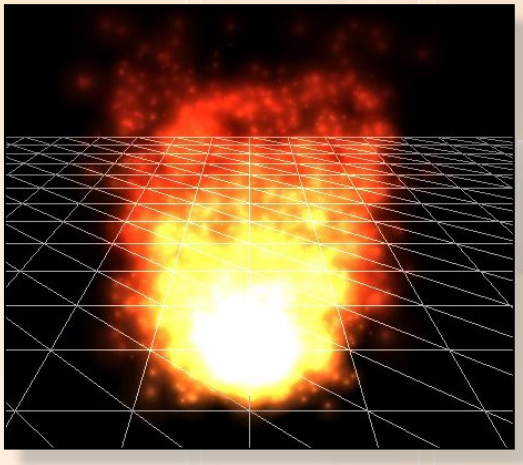
Particle Systems

Overview

- Introduction.
- A bit of history of particle systems.
- Definitions: particle and particle system.
- Attributes of particles and particle systems.
- Particle/particle system: data structures.
- Assumptions on particle system (W. Reeves).
- Collision avoidance.
- Particle emission, dynamics (Verlet integration), and rendering.
- Particle Life Cycle.
- Particle system: general algorithm.
- Final considerations.

Introduction

- *Particle systems* are a Monte-Carlo style technique which uses thousands (or millions) or tiny graphical artifacts to create large-scale visual effects.
- Particle systems are used for ***hair***, ***fire***, ***fireworks***, ***smoke***, ***water***, ***waterfalls***, ***spray***, ***foam***, ***clouds***, ***crowds***, ***herds***, ***explosions*** (*smoke*, *flame*, *chunks of debris*), ***energy glows***, in-game *special effects* and much more. Widely used in movies as well as games.
- The basic idea:
  "If lots of little dots all do something the same way, our brains will see the thing they do and not the dots doing it."



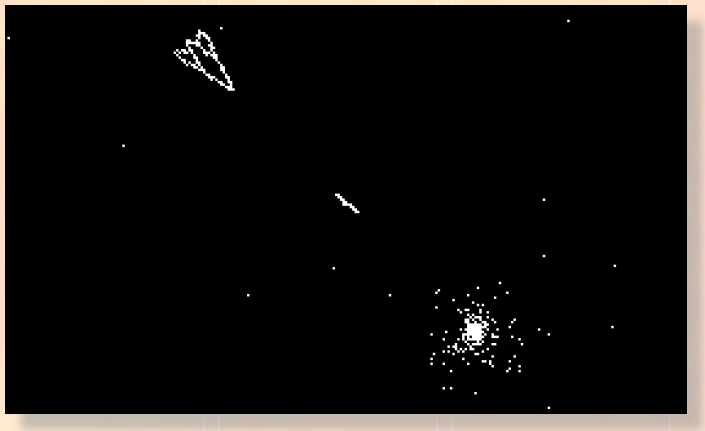
A particle system created with 3dengfx, from wikipedia.

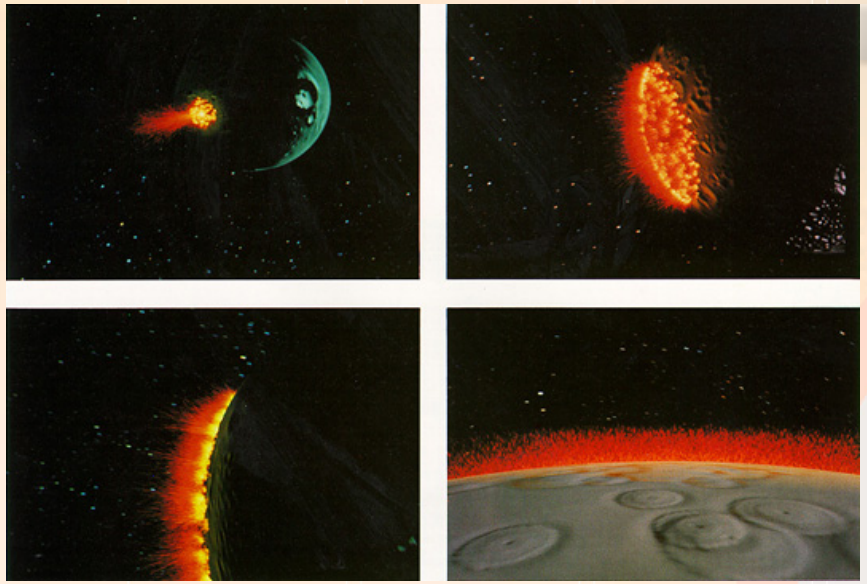
Screenshot from the game Command and Conquer 3 (2007) by Electronic Arts; the "lasers" are particle effects.

A bit of history of particle systems

- 1962: Ships explode into pixel clouds in "Spacewar!", the 2nd video game *ever*.
- 1978: Ships explode into broken lines in "Asteroid".
- 1982: The Genesis Effect in "*Star Trek II: The Wrath of Khan*", William Reeves

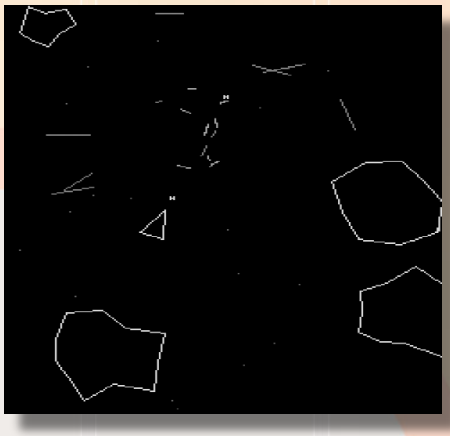
Spacewar





Star Trek II

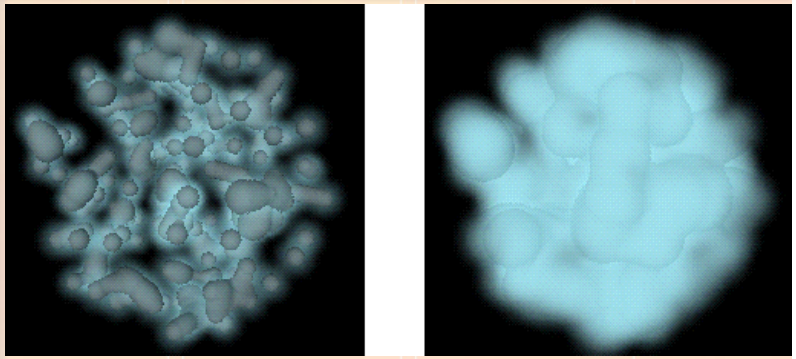
Asteroid



Fanboy note: OMG. You can play the original Spacewar! at http://spacewar.oversigma.com/-- the actual original game, running in a PDP-1 emulator inside a Java applet.

Definitions

- Metaballs are spheres that blend together to form surfaces.
- By representing particles as metaballs blobby surfaces can be created.



- ***Particle***. A particle is a point in 3D space.
- ***Other shape representations***: lines, polygons, metaballs, etc.
- ***Particle physics***:
  - Forces (e.g. gravity or wind) accelerate a particle.
  - Acceleration changes velocity.
  - Velocity changes position
- ***Particle system***. A particle system is a collection of a number of individual elements or *particles.*
  - Particle systems control a set of particles that act autonomously but share some common attributes.
  - A particle system is dynamic, particles changing form and moving with the passage of time.
  - A particle system is not deterministic, its shape and form are not completely specified.
  - The shape of a particle system changes over time.

Particles are generated in various shapes: points, lines, polygons, and even 3D models. Point shapes, for example, are used for generating smoke, lines shapes for a waterfall effect, and polygons for snow effect, or 3D models for a flock of birds.

Attributes

- Attributes of a Particle
  - Position
  - Velocity
  - Life Span (birth/dead)
  - Size (InitialSize = MeanSize + Rand() x VarSize)
  - Weight
  - Representation
  - Color
  - Owner

- Attributes of a Particle System
  - Particle List
  - Position
  - Emission Rate
  - Forces
  - Current State
  - Blending
  - Representation

Particle/particle system: data structures

```
struct Particle{
  Vector3  pos;                  // current position of the particle
  Vector3  prevPos;              // last position of the particle
  Vector3  velocity;             // direction and speed
  Vector3  acceleration;         // acceleration

  float    energy;               // determines how long the particle is alive
  float    size;                 // size of particle
  float    sizeDelta;            // amount to change the size over time

  float    weight;               // determines how gravity affects the particle
  float    weightDelta;          // change over time
  float    color[4];             // current color of the particle
  float    colorDelta[4];        // how the color changes with time
};
```

```
class ParticleSystem{
public:
  ParticleSystem(int maxParticles, Vector3 origin);
  virtual void  Update(float elapsedTime)     = 0;
  virtual void  Render()                      = 0;
  virtual int   Emit(int numParticles);
  virtual void  InitializeSystem();
  virtual void  KillSystem();
protected:
  virtual void  InitializeParticle(int index) = 0;
  Particle     *particleList;   // particles for this emitter
  int           maxParticles;   // maximum number of particles in total
  int           numParticles;   // indicies of all free particles
  Vector3       origin;         // center of the particle system
  float         accumulatedTime;//track when waslast particle emitted
  Vector3       force;          // force (gravity, wind, etc.) acting on the system
};
```

W. Reeves: "*Particle Systems – A Technique for Modelling a Class of Fuzzy Objets*", Computer Graphics, 17(3), pp. 359-376, 1983

Assumptions on particle system (W. Reeves)

- 1st Assumption:
  - Particles do not collide with each other;
- 2nd Assumption:
  - Particles do not cast shadows on other particles, just on environment;
- 3rd Assumption:
  - Particles do not reflect light; modeled as point light sources.

- Example: ***Grass***
  - Entire trajectory of a particle over its lifespan is rendered to produce a static image.
  - Green and dark green colors assigned to the particles which are shaded on the basis of the scene's light sources.
  - Each particle becomes a blade of grass

white.sand by Alvy Ray Smith
(he was also working at Lucasfilm)

Collision avoidance

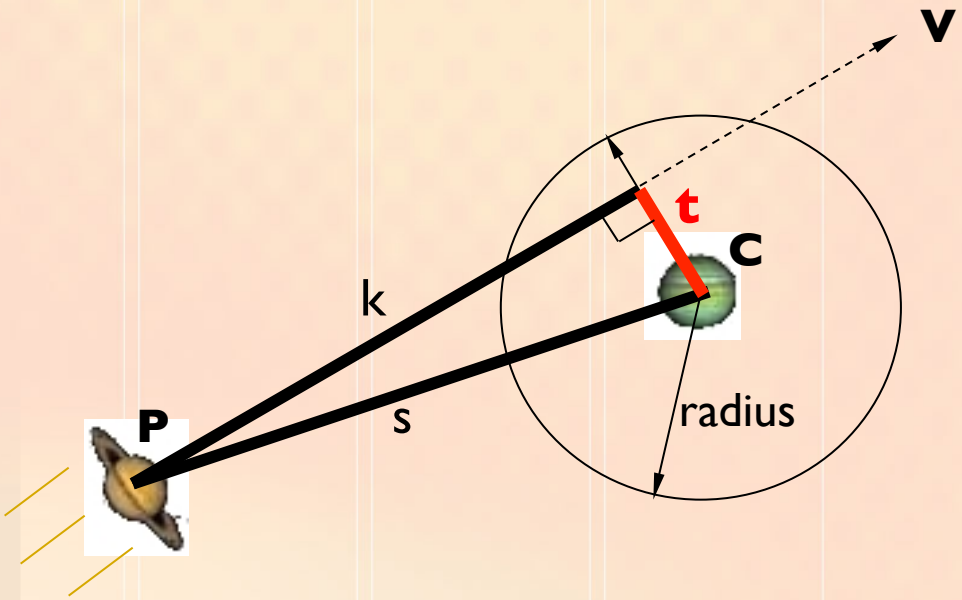
- Test for collision with bounding sphere:
  - s = | C – P |
  - k = (C – P) V/|V|
  - **t** = sqrt($s^2 - k^2$)
  - If (**t** < radius), penetration of bounding sphere occurs on current path
- Avoidance calculation:
  - Let $\begin{cases} r^2 = s^2 + t^2 \\ k^2 = s^2 + (|C-P| - t)^2 \end{cases}$
  - Solving for t

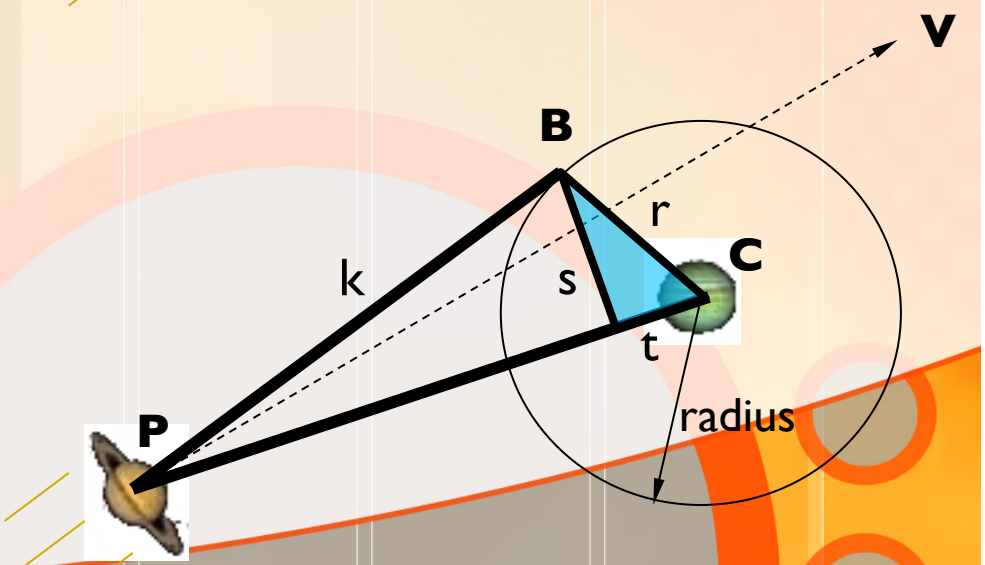
$$k^2 = r^2 - t^2 + (|C-P| - t)^2$$
$$= r^2 - t^2 + |C-P|^2 - 2|C-P|t + t^2$$

  - Hence

$$t = -\frac{k^2 - r^2 - |C-P|^2}{2|C-P|}$$

$$s = \sqrt{r^2 - t^2}$$





Collision avoidance: new trajectory

- Calculation of B:
  - Given

$$t = -\frac{k^2 - r^2 - |C - P|^2}{2|C - P|}$$

$$s = \sqrt{r^2 - t^2}$$

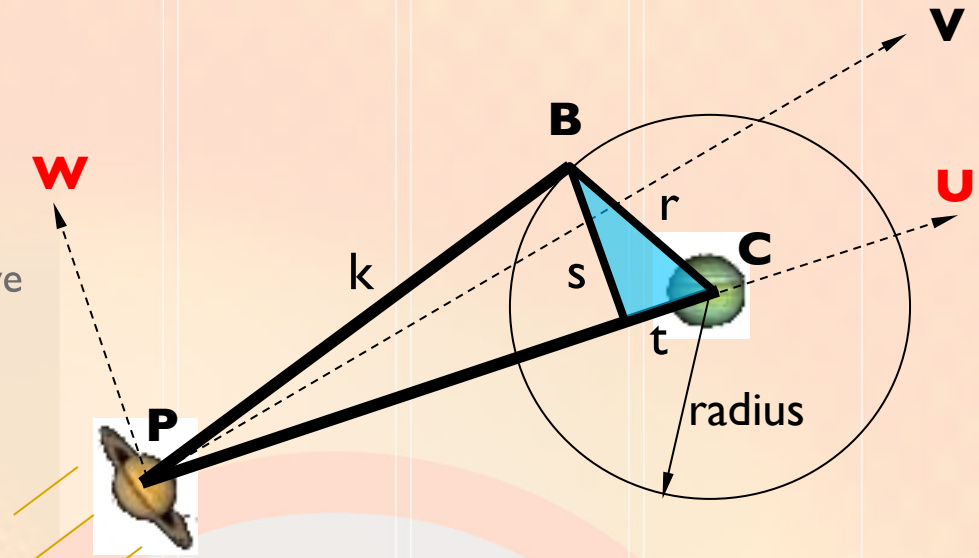
  - From the figure on the right hand-side, we see that

$$U = \frac{C - P}{|C - P|}$$

$$W = \frac{(U \times V) \times U}{|(U \times V) \times U|}$$

  - So

$$B = P + (|C - P| - t)U + sW$$



Particle system: how does it work?

- ***Emission***: Particles are generated from an *emitter*.
    - Emitter position and orientation are specified discretely;
    - Emitter rate, direction, flow, etc are often specified as a bounded random range (Monte Carlo)
- ***Move***: Time ticks; at each tick, particles move.
    - New particles are generated; expired particles are deleted
    - Forces (gravity, wind, etc) accelerate each particle
    - Acceleration changes velocity
    - Velocity changes position
- ***Rendering***: Particles are rendered.

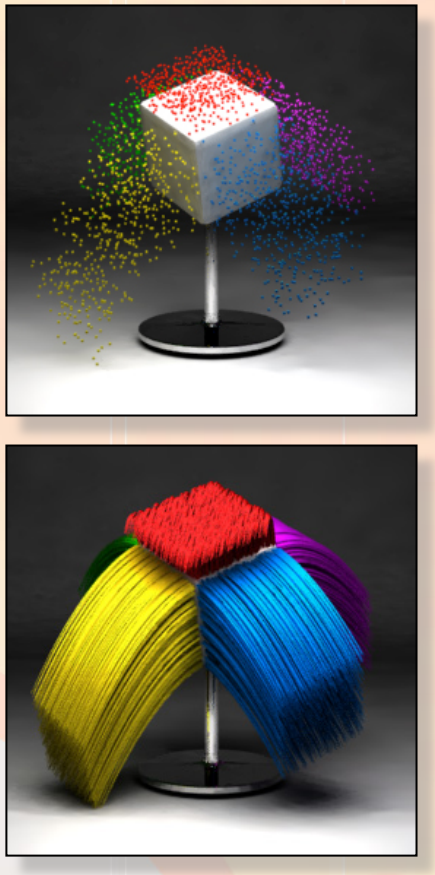
Particle emission

- Particles are generated using processes with an element of randomness.
- Two usual emission techniques:
  - emission per frame
  - emission per screen area
- ***Emission per frame***. Constrain the average number of particles generated per frame:

  # new particles = average # particles per frame
  + rand() . variance

- ***Emission per screen area***. Constrain the average number of particles per screen area:

  # new particles = average # particles per area
  + rand() . variance . screen area

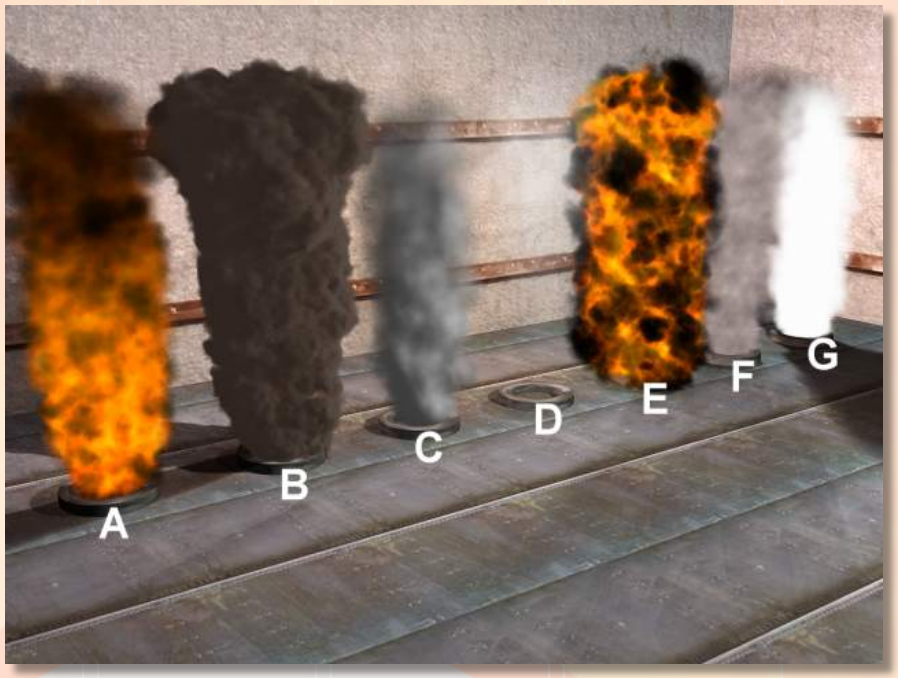


Transient vs persistent particles emitted to create a 'hair' effect (source:Wikipedia)

Other attributes can vary over time as well, such as color, transparency and size. These rates of change can be global or they can be stochastic for each particle.

Particle dynamics (move)

- A particle's position is found by simply adding its velocity vector (speed and direction) to its position vector. This can be modified by forces such as gravity.
- You now have a choice of integration technique:
  - Stateless integration
  - Iterative integration
- ***Stateless integration***. Evaluate the particles at arbitrary time *t* as a closed-form equation for a stateless system.
- ***Iterative (numerical) integration***:
  - Euler integration
  - Verlet integration
  - Runge-Kutta integration



Particle dynamics: two integration shortcuts

This is called Verlet integration and is used intensely when simulating molecular dynamics, or to emulate air friction in computer games.

http://en.wikipedia.org/wiki/Verlet_integration

- ***Closed-form function***
  - Represent every particle as a parametric equation; store only the initial position p_0, initial velocity v_0, and some fixed acceleration (such as gravity g.)
  - $p(t)=p_0+v_0t+½gt^2$
- ***No storage of state***
  - *Very* limited possibility of interaction
  - Best for water, projectiles, etc—non-responsive particles.
- **Discrete integration**
  - Remember your physics—integrate acceleration to get velocity):

    $$v'=v + a \,.\, \Delta t$$
  - Integrate velocity to get position:

    $$p'=p + v \,.\, \Delta t$$
  - *Collapse the two, integrate acceleration to position:*

    $$p''=2p'-p + a \,.\, \Delta t^2$$
- ***Timestep***
  - It must be nigh-constant; collisions are hard.

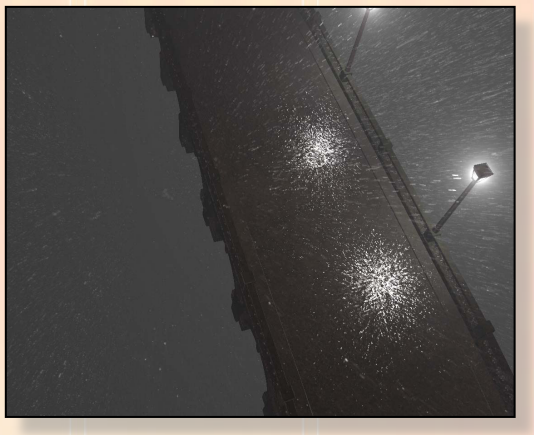
– Acceleration

  - It is computed using Newton's law f=ma (where f is the accumulated force acting on the particle).

Particle rendering

- A particle system can render particles as points, textured polygons, or primitive geometry
  - Minimize the data sent down the pipe!
  - Polygons with alpha-blended images make pretty good fire, smoke, etc
- Transitioning one particle type to another creates realistic interactive effects
  - Ex: a 'rain' particle becomes an emitter for 'splash' particles on impact
- Particles can be the force sources for a blobby model implicit surface
  - This is an effective way to simulate liquids
- Particles can obscure other objects behind them, can be transparent, and can cast shadows on other objects.

Image by nvidia



All of these 4 parameters can also be defined by applying a texture to them (typically a fractal texture). Many systems offer the user some "predefined" particle library settings for obtaining a particular look and feel.

Rendering parameters

Images by nvidia

- Color
  - After choosing the color of the particles, some software will allow you to animate it as well.
  - Example: Color of a fire particle changes as it moves away from the fire; flames near a log are orange near the flame source, darker red at the top of the flame, and black as they move farther away.
- Transparency
  - Mist – very transparent. Smoke – more opaque
- Blur
  - Sparks – distinct; no blur. Smoke – blur to a point that the viewer can't see individual particles The amount of blur is related to a particles **trail life**. As a particle moves through space, it may leave a visible trail behind. The longer the life of the trail, the longer the trail will become and the more visible it will be. This is normally measured in seconds.
- Glow
  - Makes particles look incandescent by increasing their brightness and color saturation.

Particle Life Cycle

- ***Generation***. Particles are generated randomly within a predetermined location of the fuzzy object (i.e., particle system). This generator is termed the *generation shape* of the fuzzy object, and this generation shape may change over time. Each particle's attribute is given an initial value, which may be fixed or may be determined by a stochastic process.
- ***Dynamics***. The attributes of each particle may vary over time. For example, the color of a particle in an explosion may get darker as it gets further from the center of the explosion, indicating that it is cooling off. In general, each particle's attribute can be specified by a parametric equation with time as the parameter. Particle attributes can be functions of both time and other particle attributes. For example, particle position is going to be dependent on previous particle position and velocity as well as time.
- ***Death***. Each particle has two attributes dealing with length of existence: age and lifetime. Age is the time that the particle has been alive (measured in frames), this value is always initialized to 0 when the particle is created. Lifetime is the maximum amount of time that the particle can live (measured in frames). When the particle age matches it's lifetime it is destroyed. In addition there may be other criteria for terminating a particle prematurely:
  - Running out of bounds - If a particle moves out of the viewing area and will not reenter it, then there is no reason to keep the particle active.
  - Hitting the ground - It may be assumed that particles that run into the ground burn out and can no longer be seen.
  - Some attribute reaches a threshold - For example, if the particle color is so close to black that it will not contribute any color to the final image, then it can be safely destroyed.

Particle system: general algorithm

1. Inject any new particles into the system and assign them their individual attributes
   - There may be one or more sources
   - Particles might be generated at random (clouds), in a constant stream (waterfall), or according to a script (fireworks)
2. Remove any particles that have exceeded their lifetime
   - May have a fixed lifetime, or die on some condition
3. Move all the current particles according to their script
   - Script typically refers to the neighboring particles and the environment
4. Render all the current particles
   - Many options for rendering

Further reading

- Alex Benton, University of Cambridge – A.Benton@damtp.cam.ac.uk, "Advanced Graphics", Lecture 7 - *Implicit Surfaces, Voxels, and Particle Systems*.
- Rahul Malhotra, Lecture on *Particle Systems*, CS536.
- Peter Capelluto, "Advanced Graphics", Lecture on *Particle Systems*, CS551.
- http://www.particlesystems.org/
- http://www.2ld.de/gdc2007/EverythingAboutParticleEffectsSlides.pdf
- www.evl.uic.edu/aej/527/lecture05.html
- mit.edu/groups/el/projects/spacewar/
- http://www.mirwin.net/
- http://www.javaworld.com/javaworld/jw-05-2006/jw-0529-funandgames.html?page=1
- William T. Reeves, "*Particle Systems - A Technique for Modeling a Class of Fuzzy Objects*", Computer Graphics 17:3 pp. 359-376, 1983 (SIGGRAPH 83).
- Lutz Latta, *Building a Million Particle System*, http://www.2ld.de/gdc2004/MegaParticlesPaper.pdf, 2004
- http://en.wikipedia.org/wiki/Particle_system

Summary

- Introduction.
- A bit of history of particle systems.
- Definitions: particle and particle system.
- Attributes of particles and particle systems.
- Particle/particle system: data structures.
- Assumptions on particle system (W. Reeves).
- Collision avoidance.
- Particle emission, dynamics (Verlet integration), and rendering.
- Particle Life Cycle.
- Particle system: general algorithm.
- Final considerations.