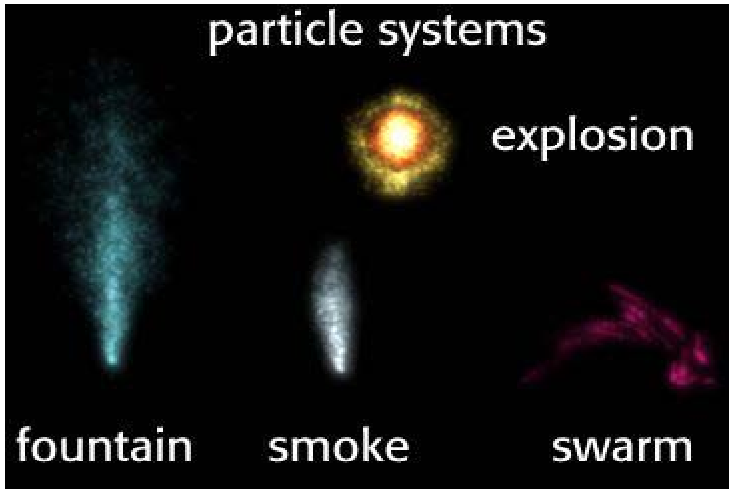
Animation & Rendering with Particle Systems

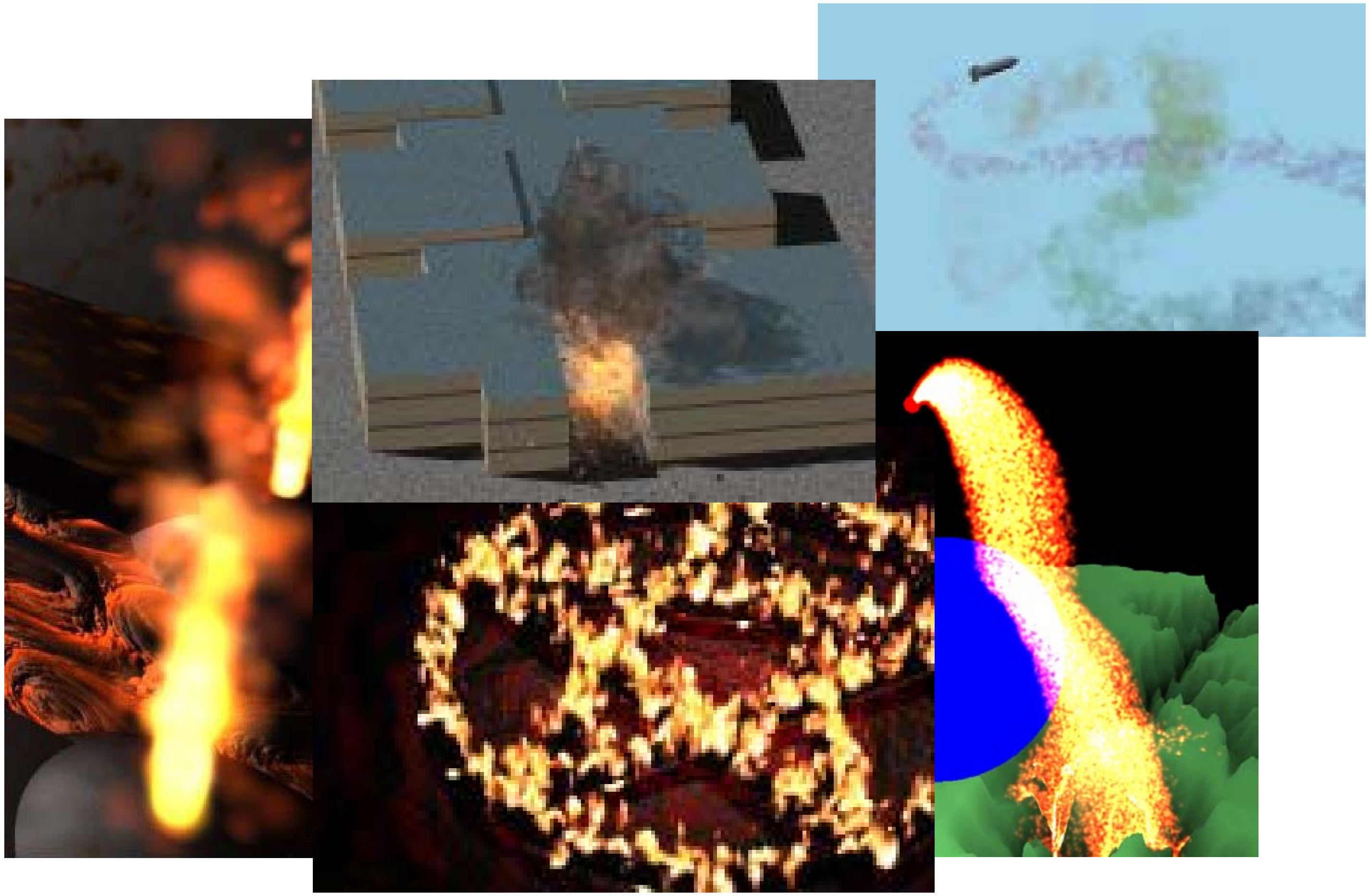
- Motivation
- Applications
- Formulation
- Dynamics
- Rendering
- Summary
- Readings

Particle Systems: Motivation

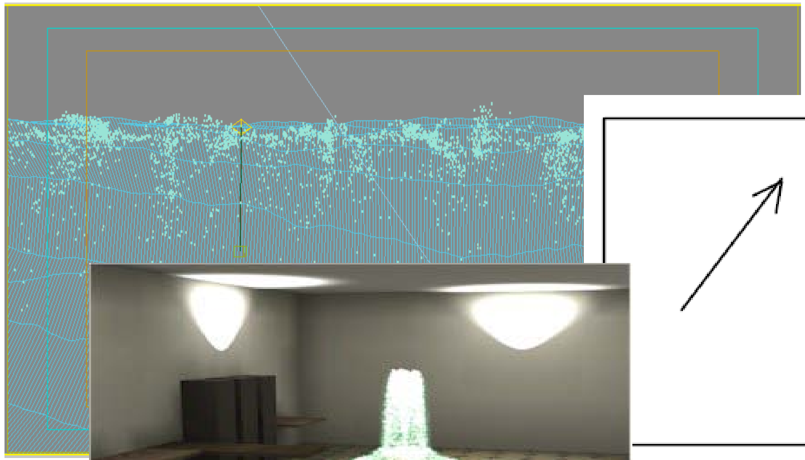
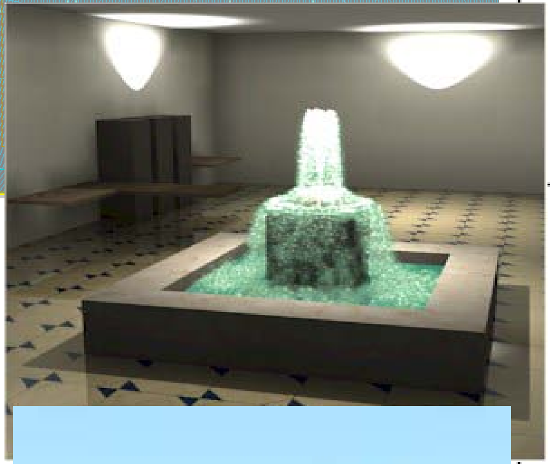
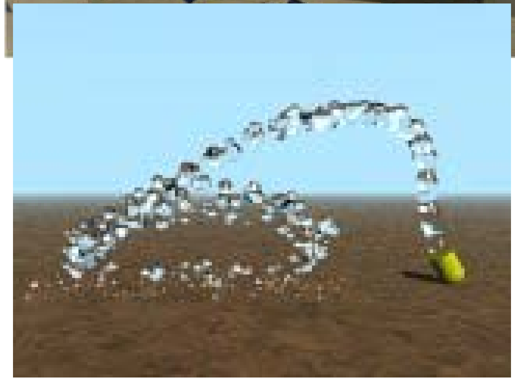
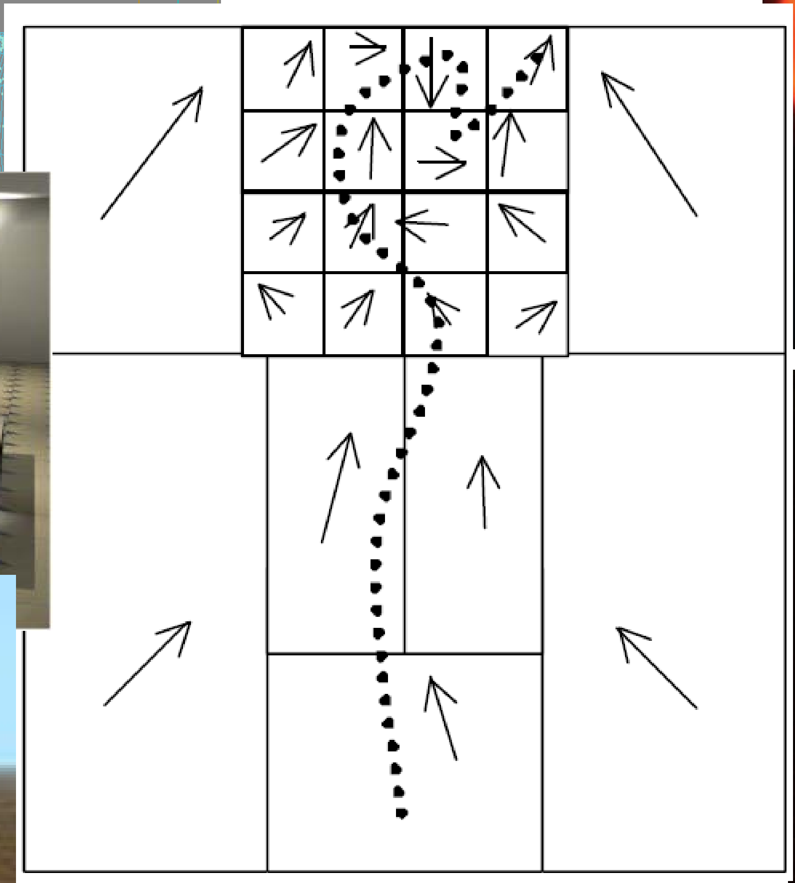
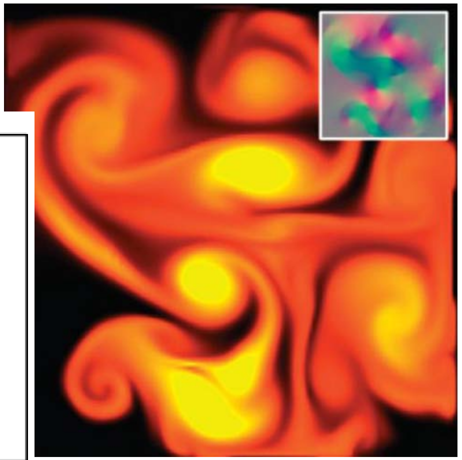


- Particle systems can model complex fuzzy shapes and dynamics
- Simplicity and strength in numbers
- Offer both local and global control

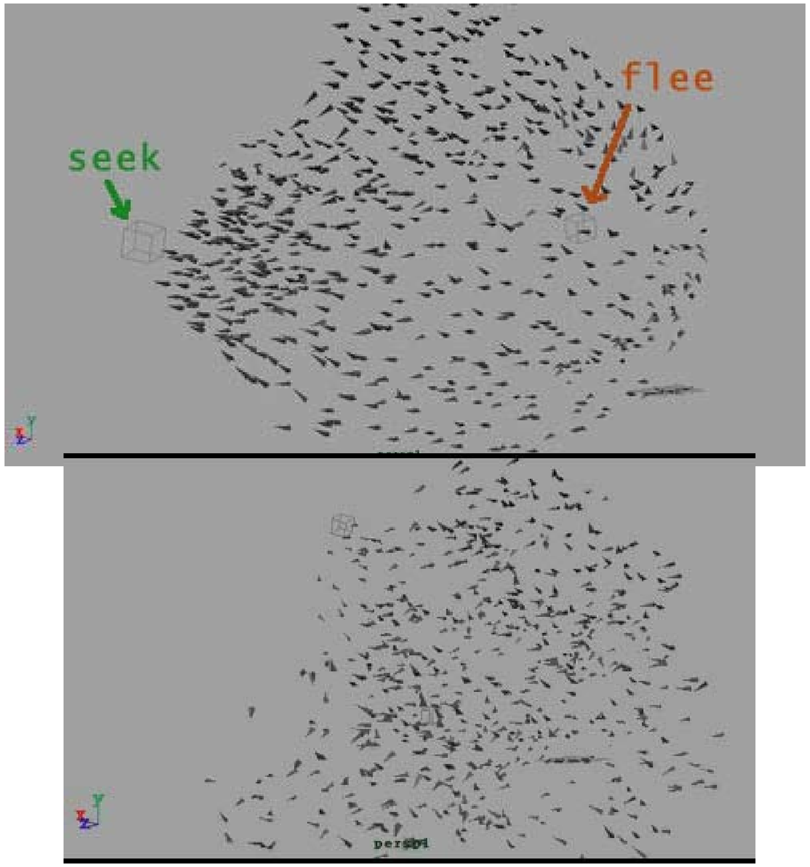
Applications: Fuzzy Phenomena

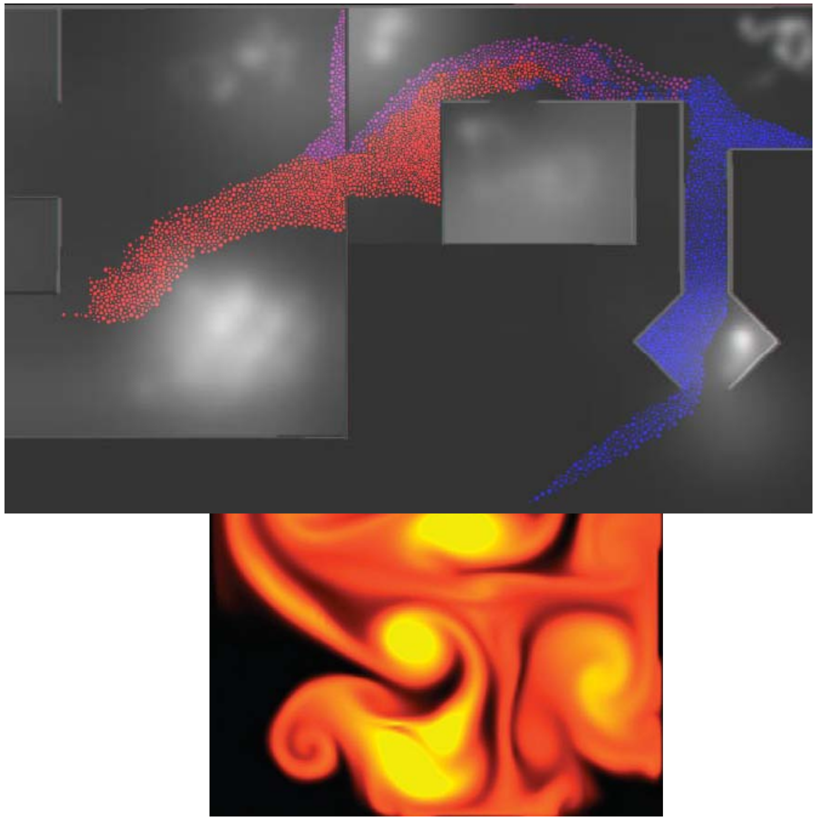


Applications: Fluids

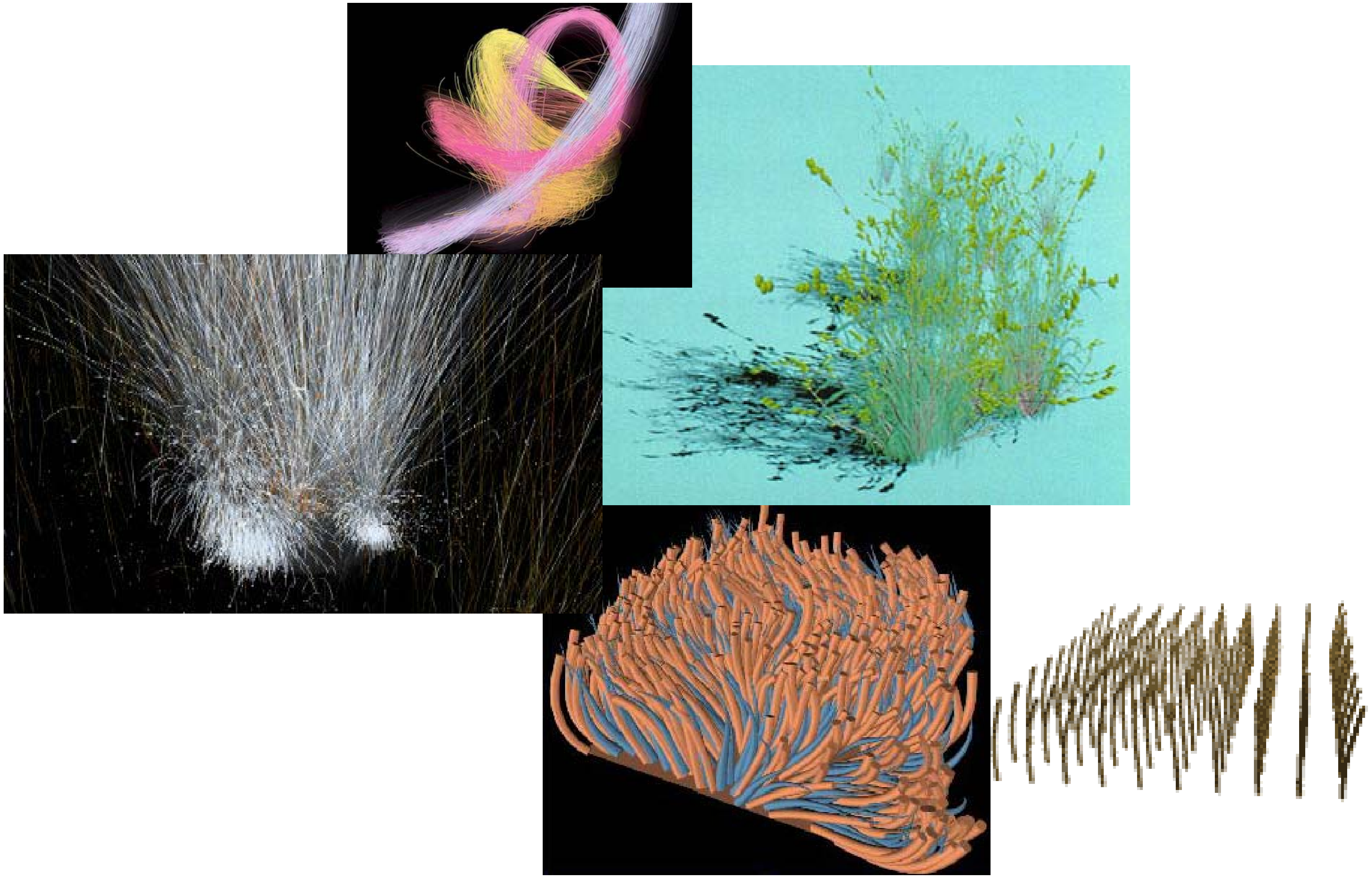






Applications: Flocks

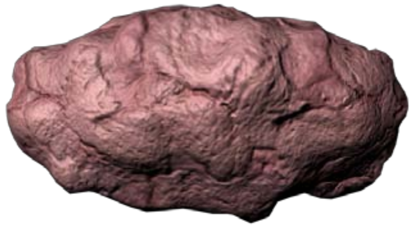
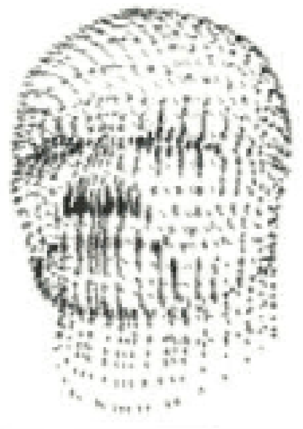
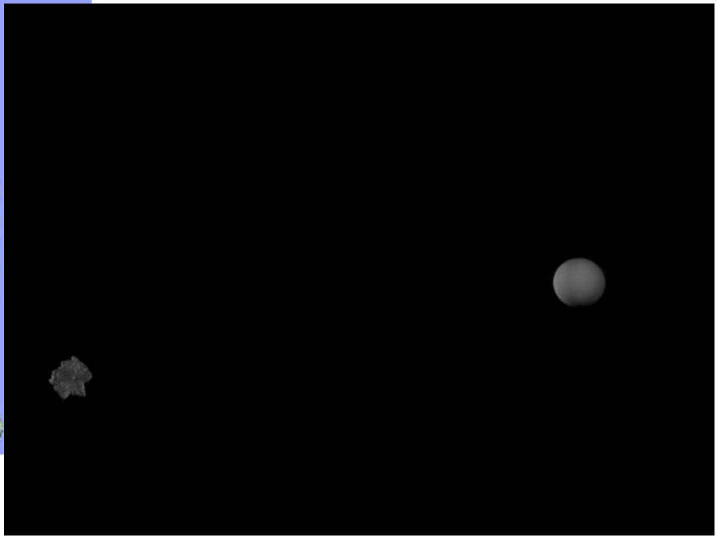




Applications: Rendered Trails



Applications: Soft bodies

Formulation: Representing Objects

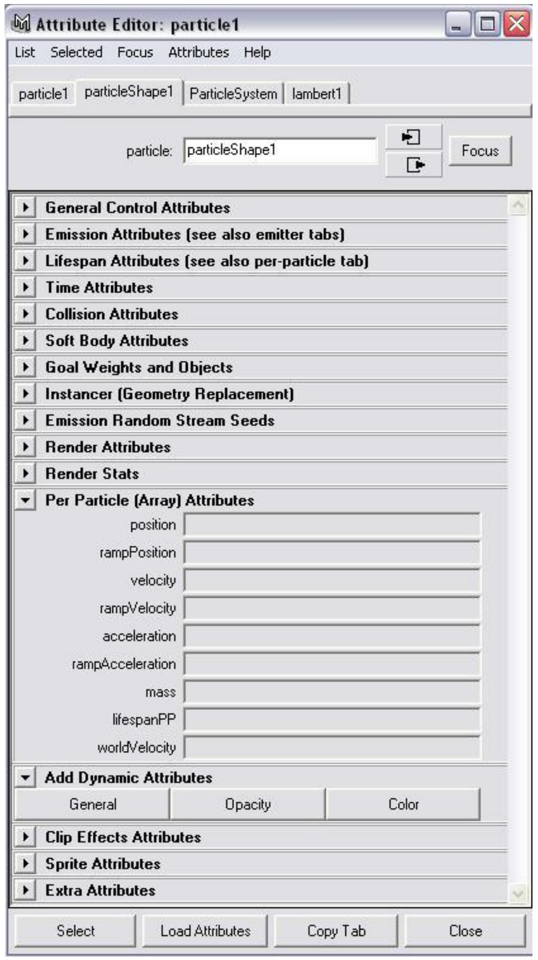
- An object is represented as clouds of primitive particles that define its volume rather than by polygons or patches that define its boundary.
- A particle system is dynamic, particles changing form and moving with the passage of time.
- Object is not deterministic, its shape and form are not completely specified. Instead

Formulation: Basic Model

1) New particles are generated into the system.
2) Each new particle is assigned its individual attributes.
3) Any particles that have existed past their prescribed lifetime are extinguished.
4) The remaining particles are moved and transformed according to their dynamic attributes.
5) An image of the particles is rendered in the frame buffer, often using special purpose algorithms.

Formulation: Particle Attributes

- Position
- Velocity
- Mass
- Size
- Color
- Transparency
- Lifetime
- Shape
- Texture
- Orientation
- Material Properties



Attribute Editor: particle1

List Selected Focus Attributes Help

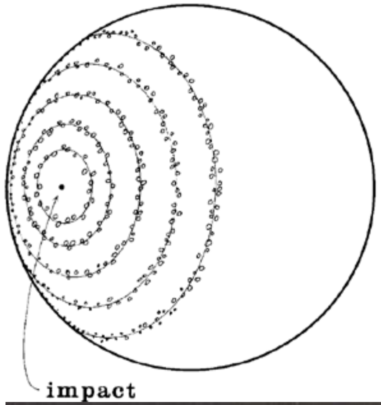
particle1 | particleShape1 | ParticleSystem | lambert1

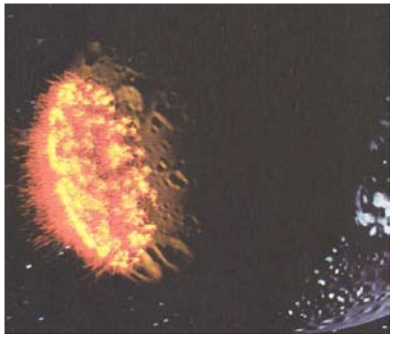
particle: particleShape1 Focus

- General Control Attributes
- Emission Attributes (see also emitter tabs)
- Lifespan Attributes (see also per-particle tab)
- Time Attributes
- Collision Attributes
- Soft Body Attributes
- Goal Weights and Objects
- Instancer (Geometry Replacement)
- Emission Random Stream Seeds
- Render Attributes
- Render Stats
- Per Particle (Array) Attributes
  - position
  - rampPosition
  - velocity
  - rampVelocity
  - acceleration
  - rampAcceleration
  - mass
  - lifespanPP
  - worldVelocity
- Add Dynamic Attributes
  - General | Opacity | Color
- Clip Effects Attributes
- Sprite Attributes
- Extra Attributes

Select | Load Attributes | Copy Tab | Close

Alias|Wavefront's Maya Particle System





Formulation: Particle Generation

Particles are generated using processes with an element of randomness.

One way to control the number of particles created is by the particles generated per frame:

$$Nparts_f = MeanParts_f + Rand() \times VarianceParts_f$$

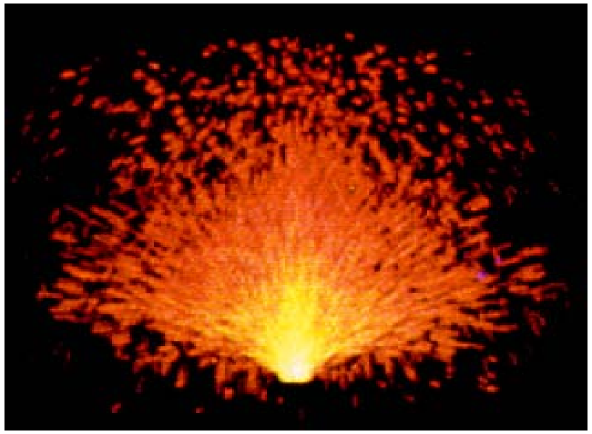
Another method generates a certain number of particles per screen area:

$$Nparts_f = (MeanParts_{SAf} + Rand() \times VarianceParts_{SAf}) \times ScreenArea$$

With this method the number of new particles depends on the screen size of the object.

Formulation: Particle Extinction

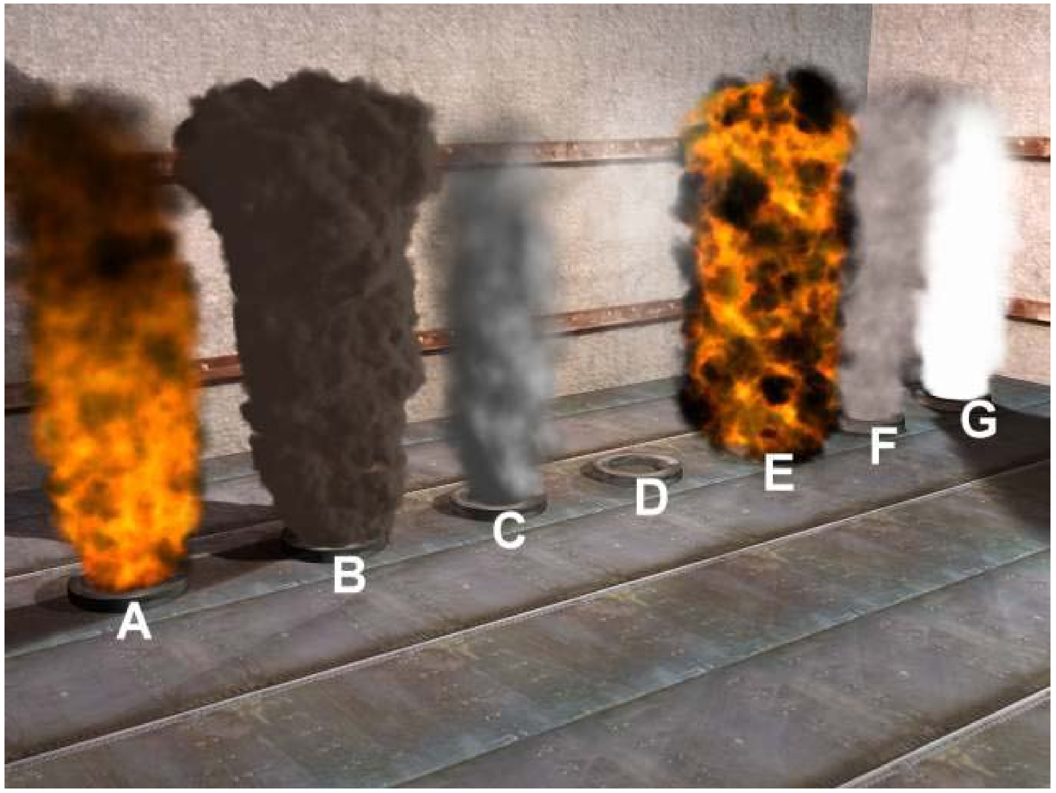
- When generated, given a lifetime in frames.
- Lifetime decremented each frame, particle is killed when it reaches zero.
- Kill particles that no longer contribute to image (transparency below a certain threshold, etc.).



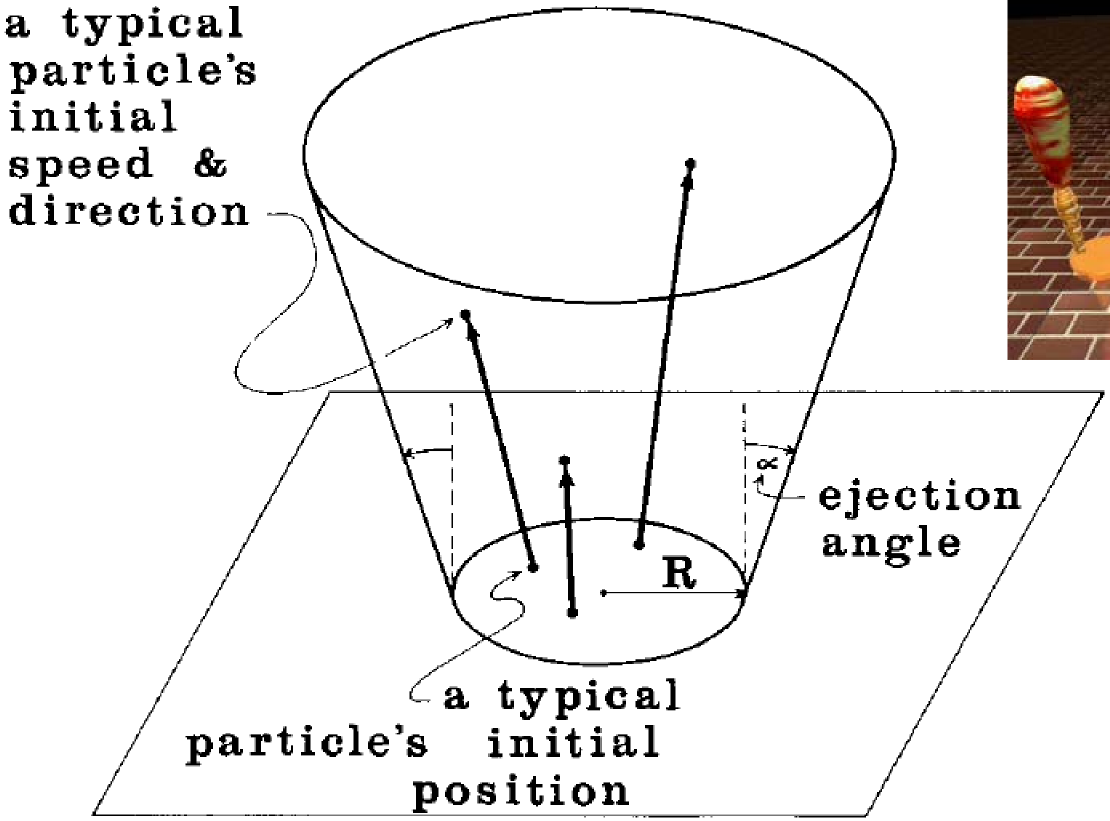
Formulation: Particle Animation

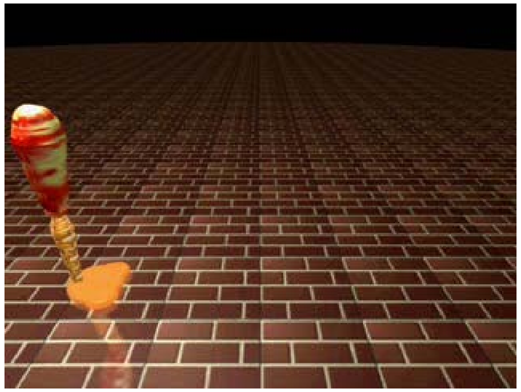
A particle's position is found by simply adding its velocity vector to its position vector. This can be modified by forces such as gravity.

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.



Formulation: Particle Animation


a typical particle's initial speed & direction
∝
ejection angle
R
a typical particle's initial position

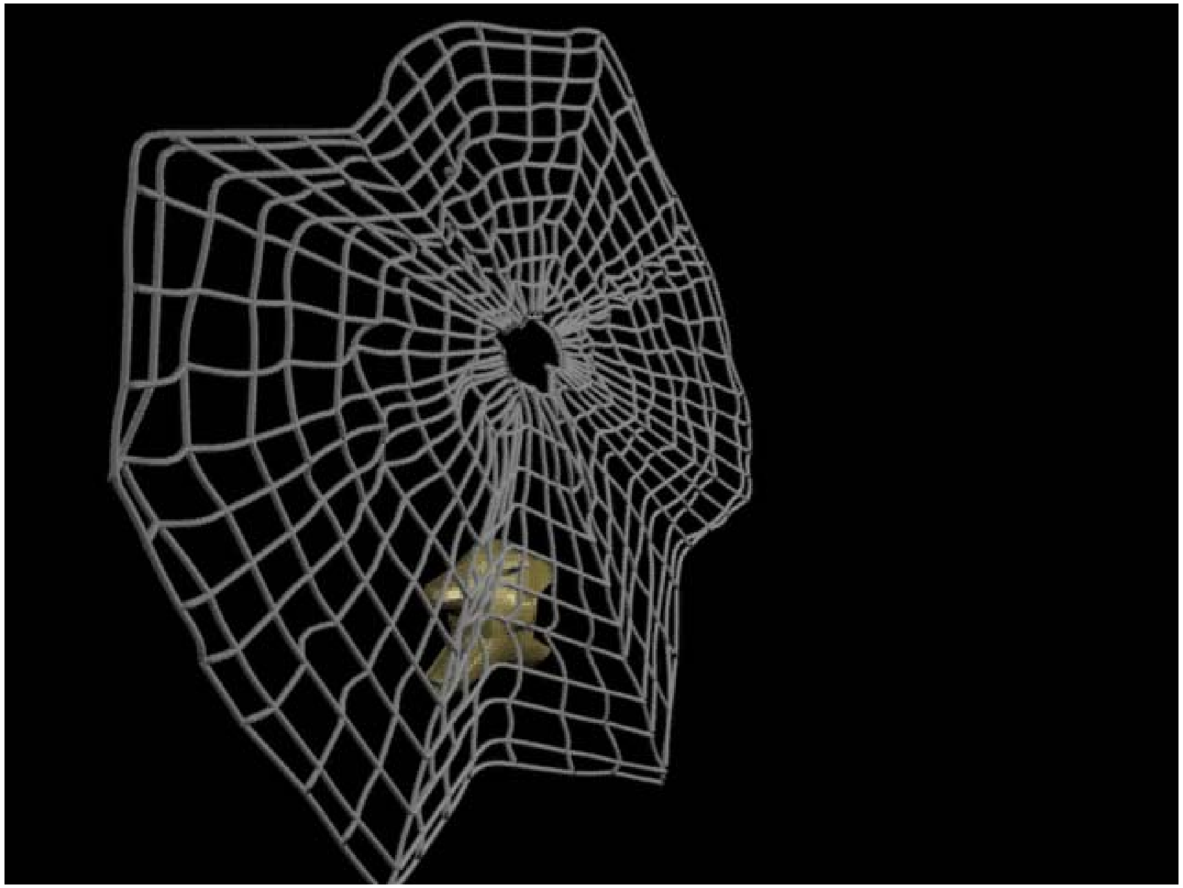


Formulation: Particle Hierarchy

Particle system such that particles can themselves be particle systems.

The child particle systems can inherit the properties of the parents.

Particle Dynamics



Particle Dynamics

- **Differential equation: f = ma**
- **Forces can depend on:**
  - **Position, Velocity, Time**

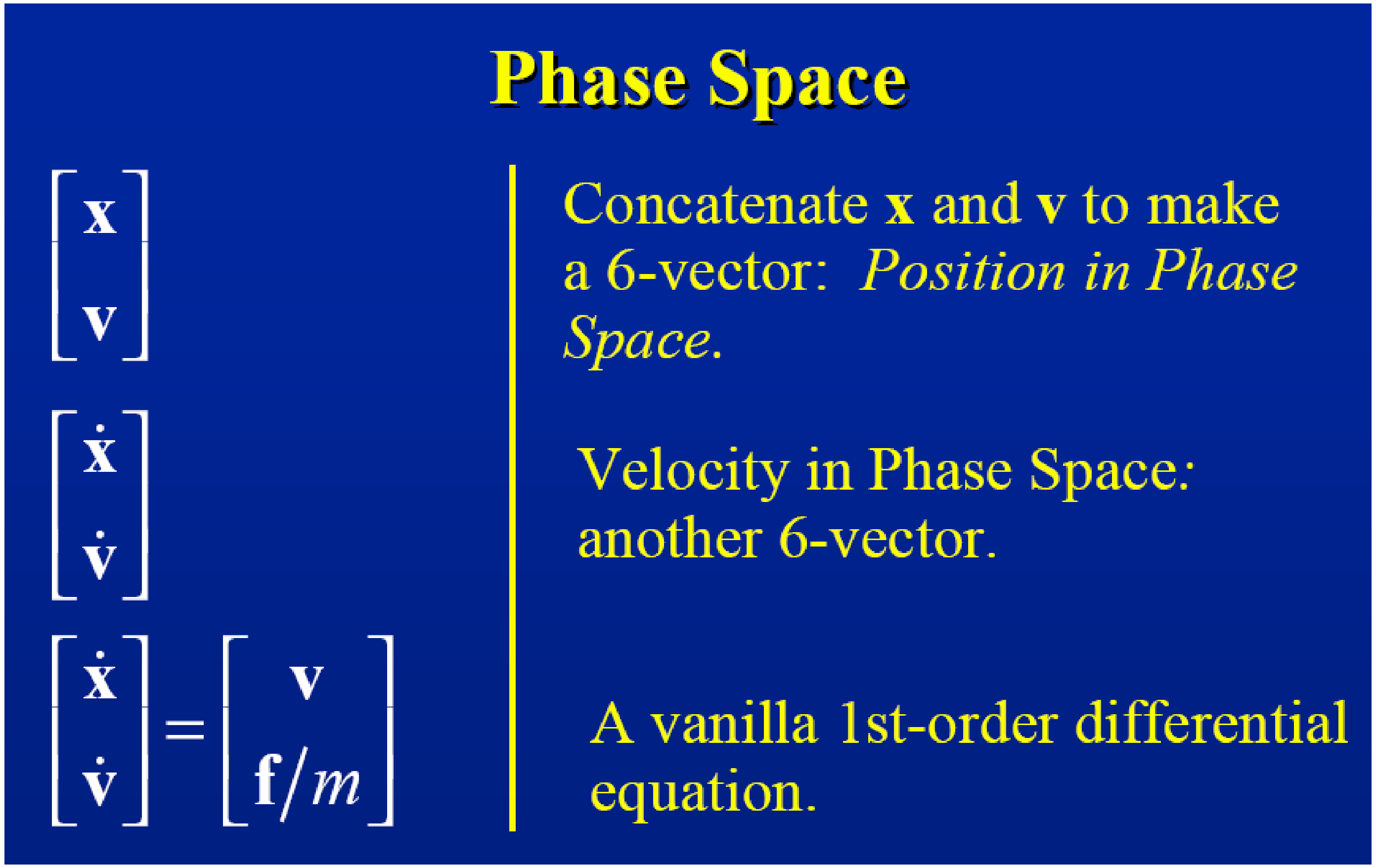
$$\ddot{\mathbf{x}} = \frac{\mathbf{f}(\mathbf{x}, \dot{\mathbf{x}}, t)}{m}$$

Not in our standard form because it has 2nd derivatives

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{v} \\ \dot{\mathbf{v}} = \mathbf{f}/m \end{cases}$$

Add a new variable, **v**, to get a pair of coupled 1st order equations.

Particle Dynamics



Phase Space

$$\begin{bmatrix} \mathbf{x} \\ \mathbf{v} \end{bmatrix}$$

Concatenate **x** and **v** to make a 6-vector: *Position in Phase Space.*

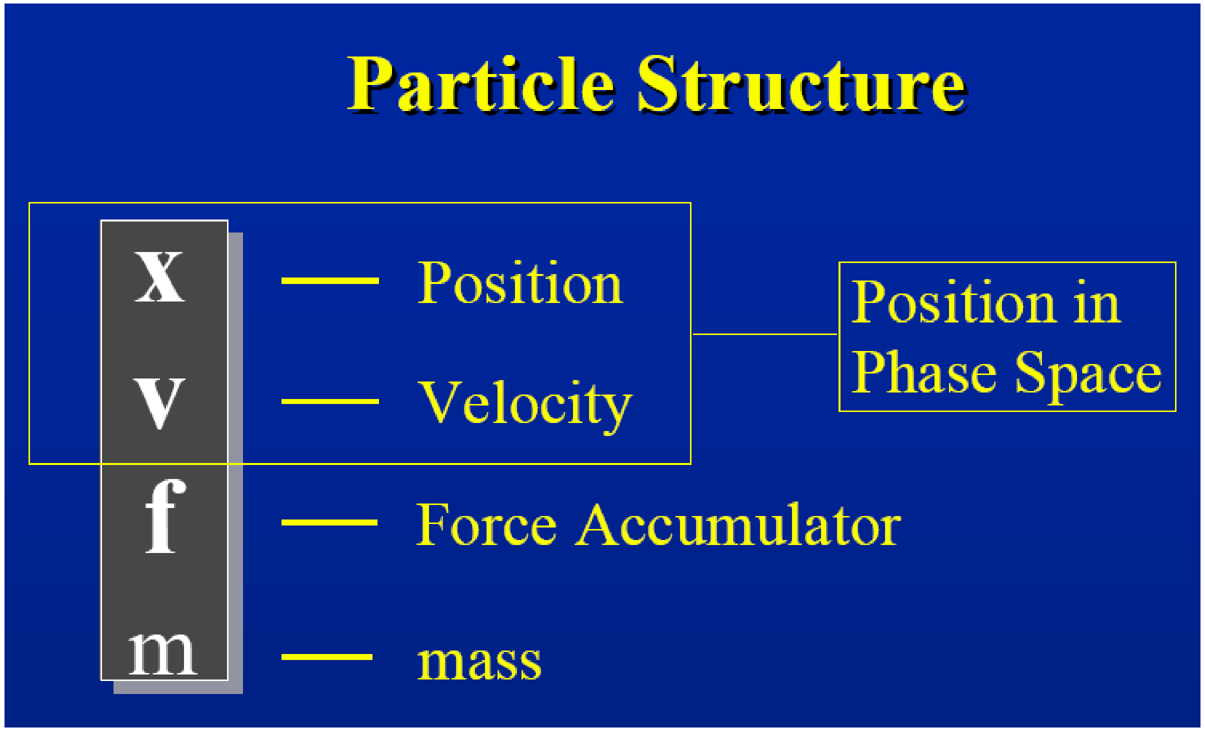
$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{v}} \end{bmatrix}$$

Velocity in Phase Space: another 6-vector.

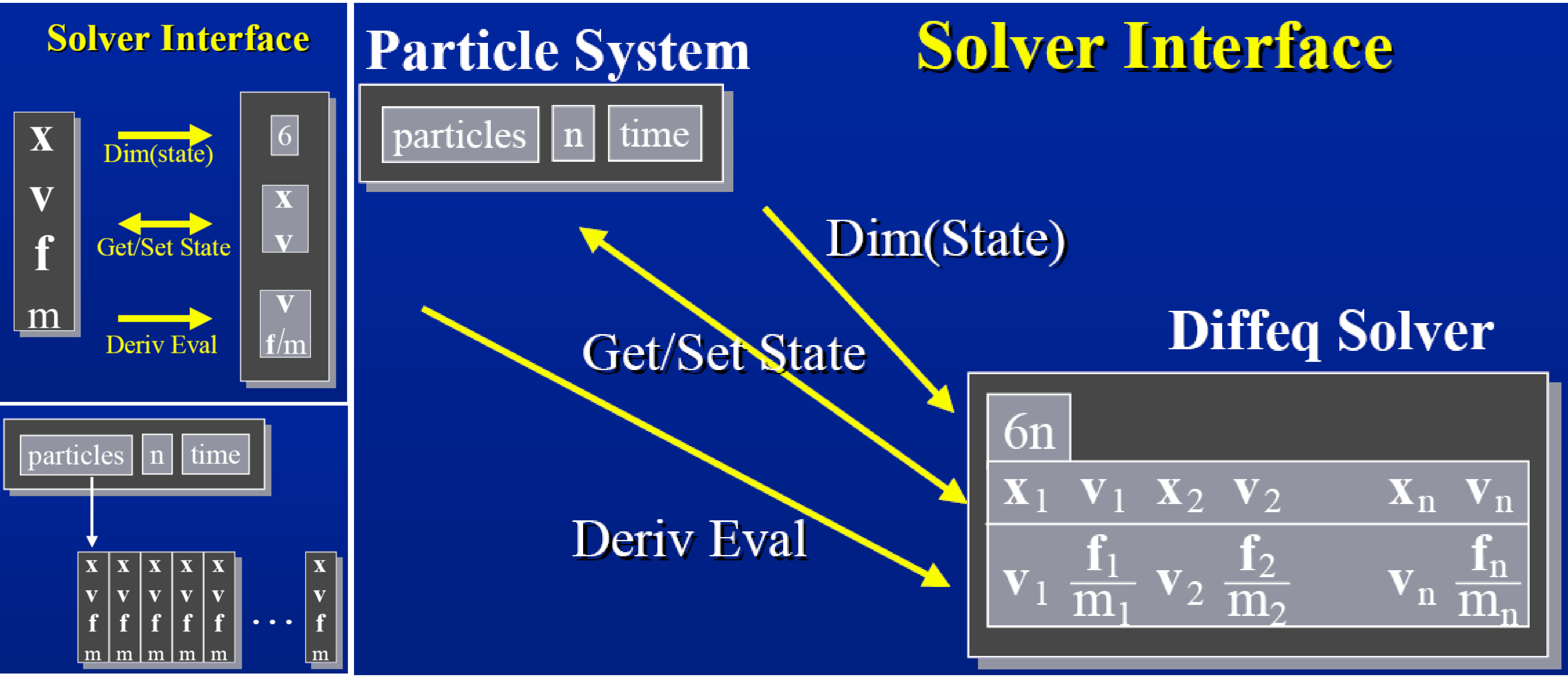
$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{v}} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{f}/m \end{bmatrix}$$

A vanilla 1st-order differential equation.

Particle Dynamics



Particle Dynamics



Particle Dynamics

Derivative Evaluation Loop

• **Clear forces**

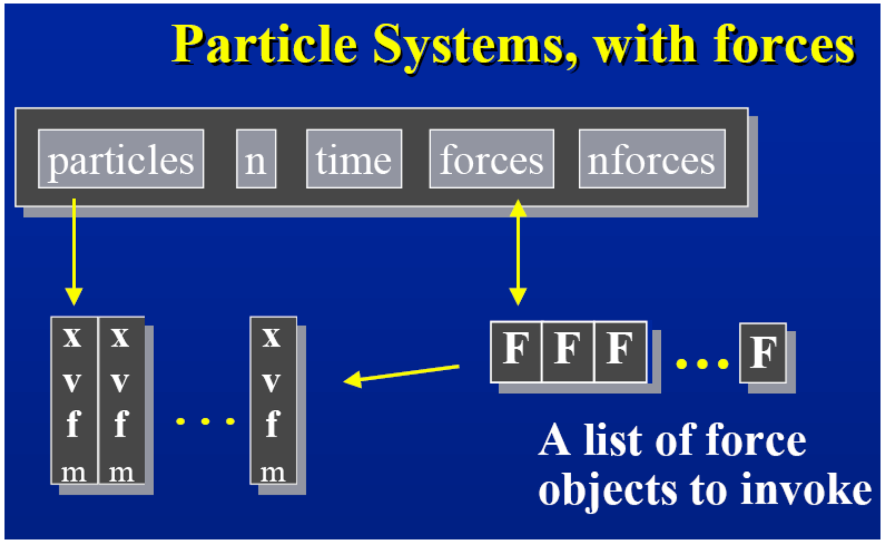
– Loop over particles, zero force accumulators.

• **Calculate forces**

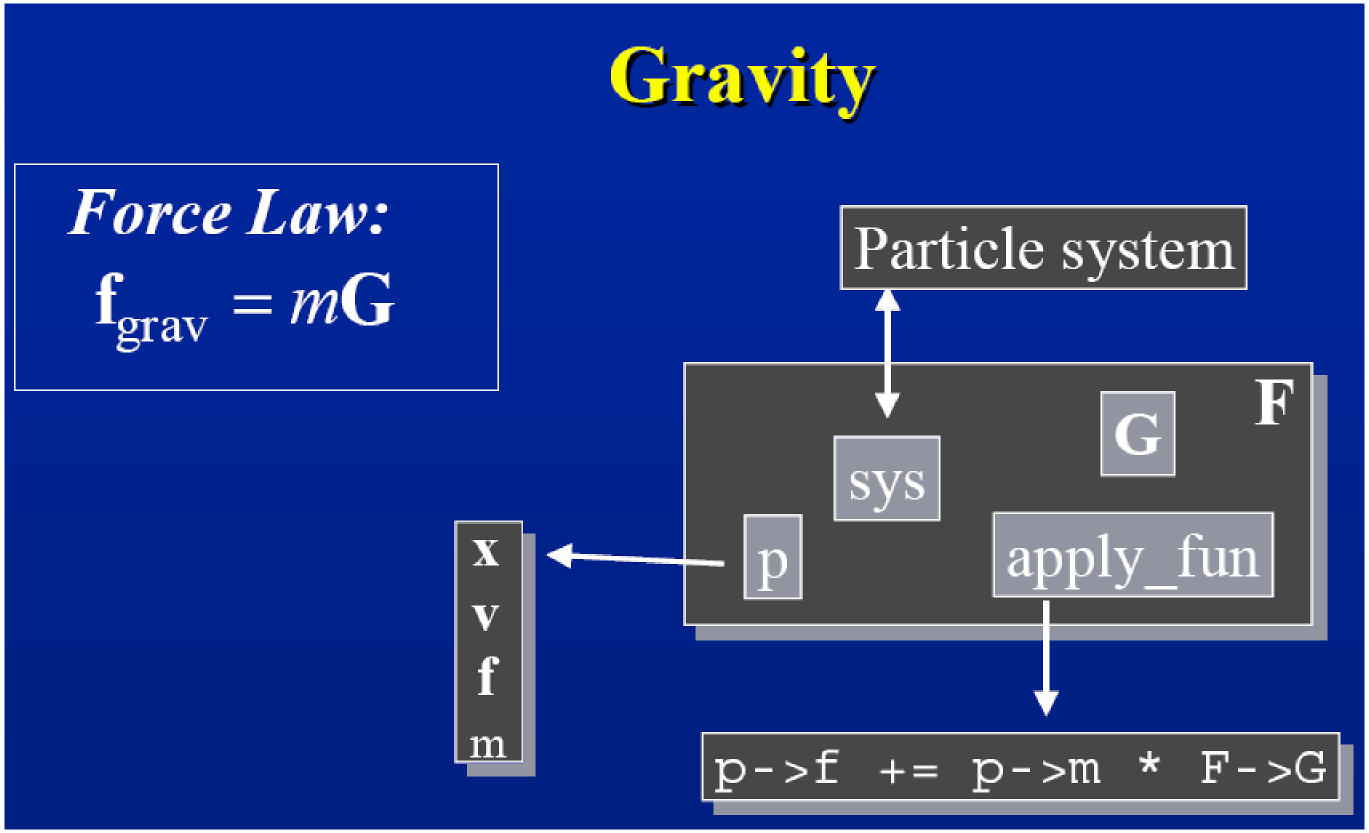
– Sum all forces into accumulators.

• **Gather**

– Loop over particles, copying **v** and **f**/m into destination array.


Particle Systems, with forces
particles
n
time
forces
nforces
x
v
f
m
x
v
f
m
x
v
f
m
F
F
F
F
A list of force objects to invoke

Particle Dynamics



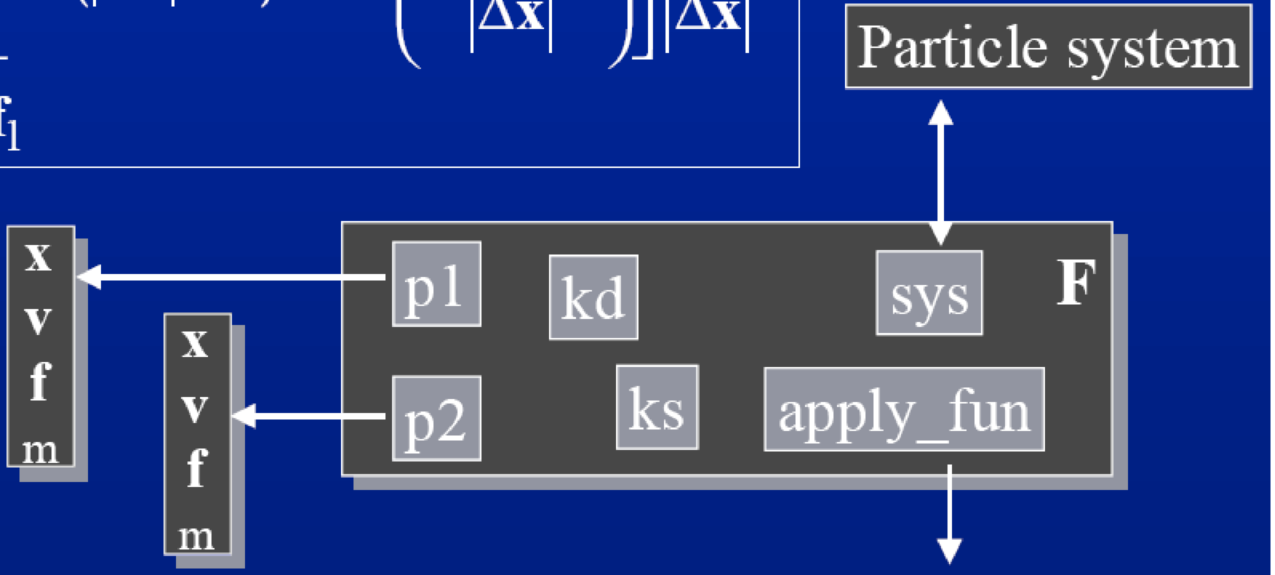
Particle Dynamics

Damped Spring

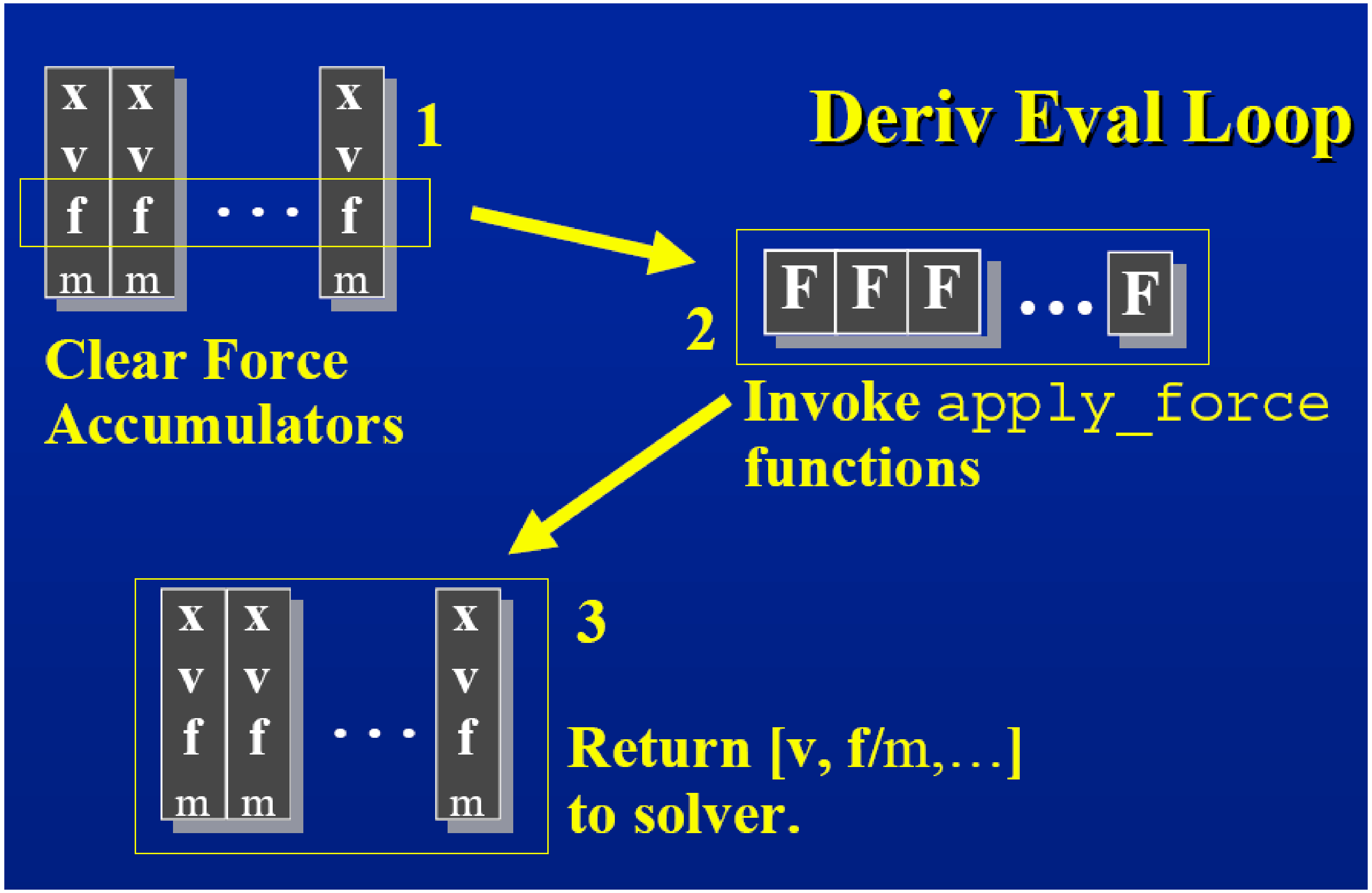
Force Law:

$$\mathbf{f}_1 = -\left[k_s\left(|\Delta\mathbf{x}| - r\right) + k_d\left(\frac{\Delta\mathbf{v}\cdot\Delta\mathbf{x}}{|\Delta\mathbf{x}|}\right)\right]\frac{\Delta\mathbf{x}}{|\Delta\mathbf{x}|}$$

$$\mathbf{f}_2 = -\mathbf{f}_1$$



Particle Dynamics



Particle Dynamics: Data Structure

```
typedef struct{
  float m;          /* mass */
  float *x;         /* position vector */
  float *v;         /* velocity vector */
  float *f;         /* force accumulator */
} *Particle;
```

```
typedef struct{
  Particle *p;      /* array of pointers to particles */
  int n;            /* number of particles */
  float t;         /* simulation clock */
} *ParticleSystem;
```

Particle Dynamics: Code

```
/* gather state from the particles into dst */
int ParticleGetState(ParticleSystem p, float *dst){
  int i;
  for(i=0; i < p->n; i++){
    *(dst++) = p->p[i]->x[0];
    *(dst++) = p->p[i]->x[1];
    *(dst++) = p->p[i]->x[2];
    *(dst++) = p->p[i]->v[0];
    *(dst++) = p->p[i]->v[1];
    *(dst++) = p->p[i]->v[2];
  }
}

/* scatter state from src into the particles */
int ParticleSetState(ParticleSystem p, float *src){
  int i;
  for(i=0; i < p->n; i++){
  p->p[i]->x[0] = *(src++);
  p->p[i]->x[1] = *(src++);
  p->p[i]->x[2] = *(src++);
  p->p[i]->v[0] = *(src++);
  p->p[i]->v[1] = *(src++);
  p->p[i]->v[2] = *(src++);
  }
}
```

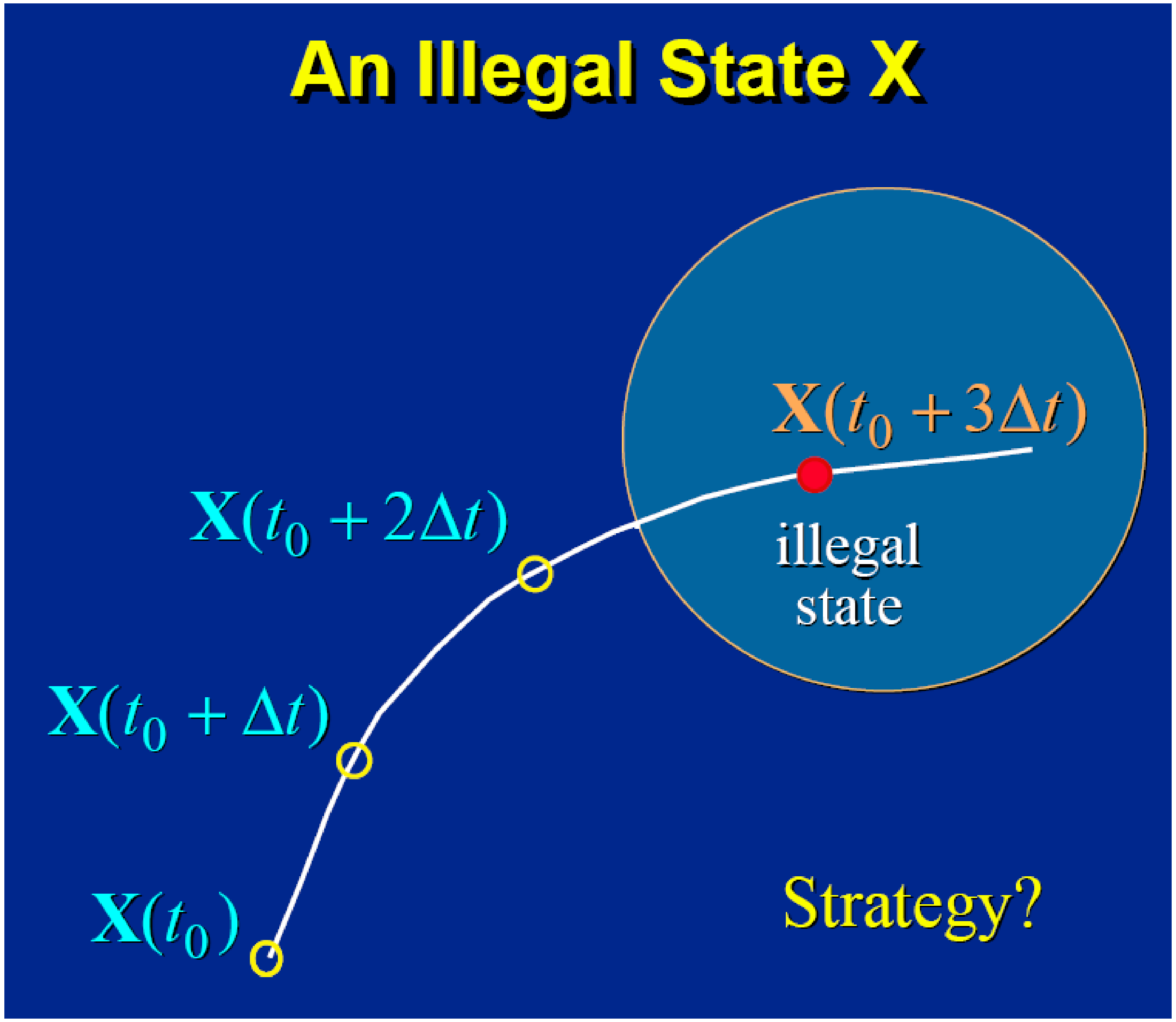
Particle Dynamics: Code

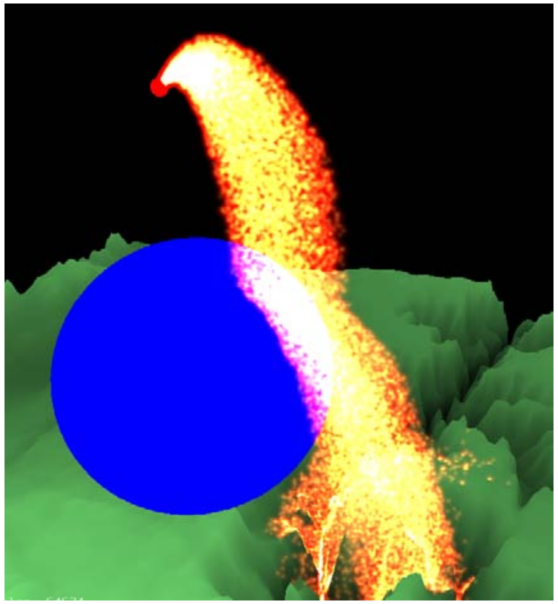
```
/* calculate derivative, place in dst */
int ParticleDerivative(ParticleSystem p, float *dst){
 int i;
 Clear_Forces(p);   /* zero the force accumulators */
 Compute_Forces(p); /* magic force function */
  for(i=0; i < p->n; i++){
    *(dst++) = p->p[i]->v[0];     /* xdot = v */
    *(dst++) = p->p[i]->v[1];
    *(dst++) = p->p[i]->v[2];
    *(dst++) = p->p[i]->f[0]/m; /* vdot = f/m */
    *(dst++) = p->p[i]->f[1]/m;
    *(dst++) = p->p[i]->f[2]/m;
  }
}
```

Particle Dynamics: Code

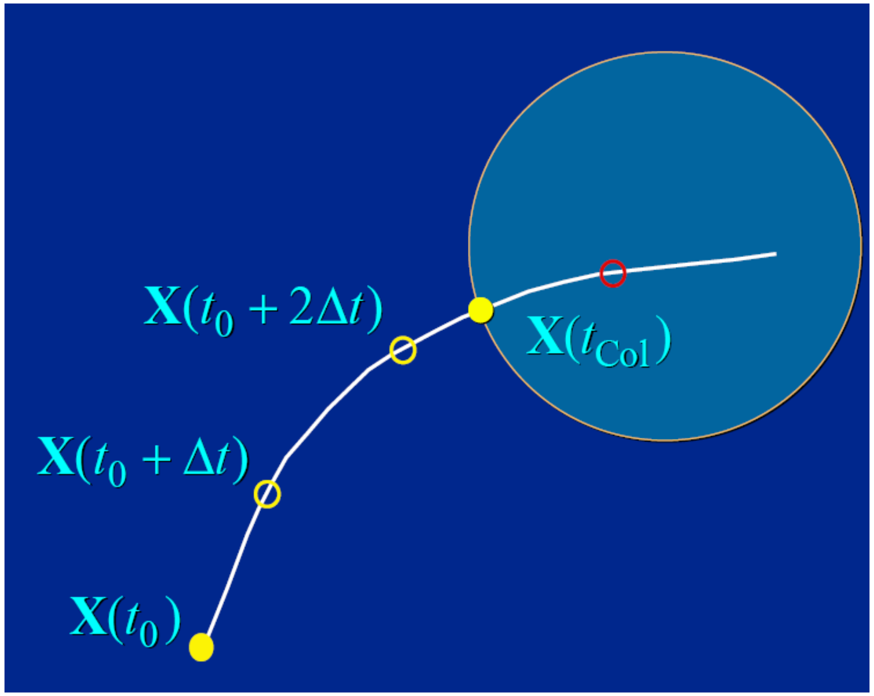
```
void EulerStep(ParticleSystem p, float DeltaT){
  ParticleDeriv(p,temp1);    /* get deriv */
  ScaleVector(temp1,DeltaT)        /* scale it */
  ParticleGetState(p,temp2);       /* get state */
  AddVectors(temp1,temp2,temp2);   /* add -> temp2 */
  ParticleSetState(p,temp2);       /* update state */
  p->t += DeltaT;                  /* update time */
}
```

Particle Collisions

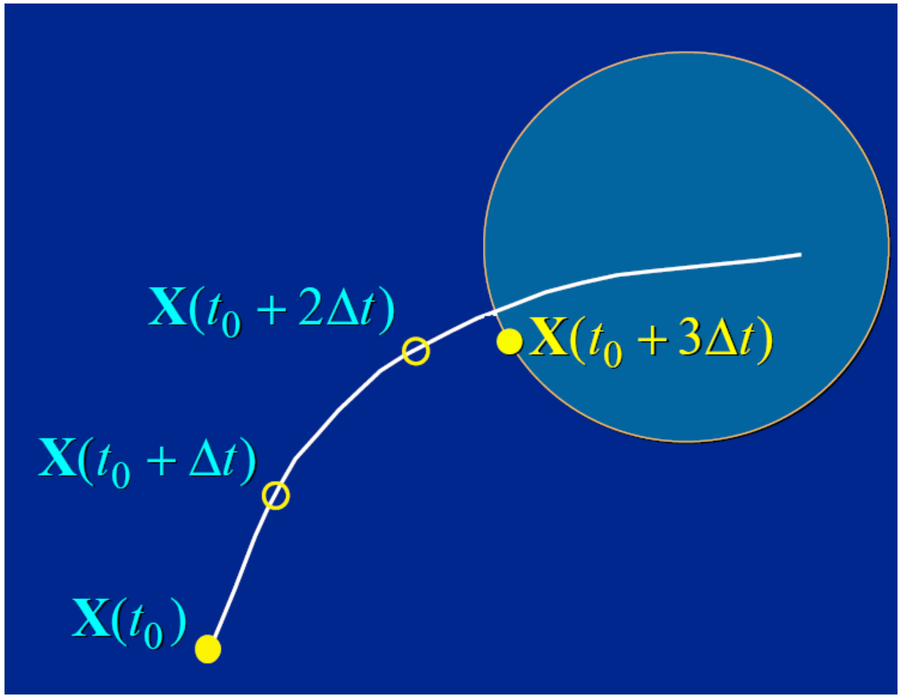

An Illegal State X
$\mathbf{X}(t_0 + 3\Delta t)$
illegal state
$\mathbf{X}(t_0 + 2\Delta t)$
$\mathbf{X}(t_0 + \Delta t)$
$\mathbf{X}(t_0)$
Strategy?



Particle Collision Detection

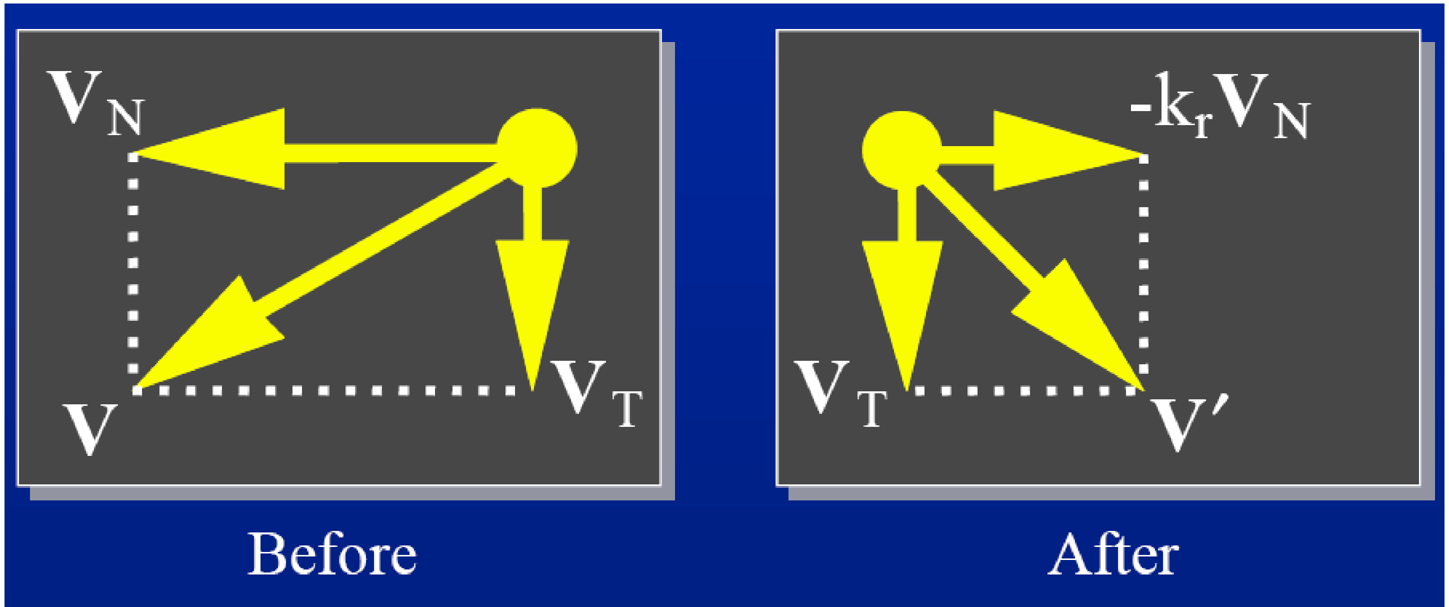


Back-step by adjusting Δt


$\mathbf{X}(t_0 + 2\Delta t)$
$\mathbf{X}(t_0 + 3\Delta t)$
$\mathbf{X}(t_0 + \Delta t)$
$\mathbf{X}(t_0)$

Choose nearest vertex on surface

Particle Collision Response



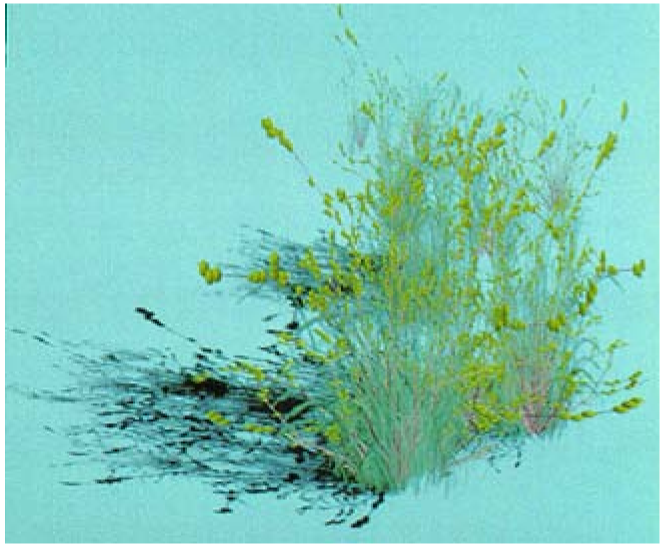
$$V' = V_T - k_r V_N$$

Particle Rendering

- For Star Trek II, Reeves made two assumptions which made things easier for him:
  - assumed particles wouldn’t intersect other geometry (handled occlusion by compositing) Particles can obscure other objects behind them, can be transparent, and can cast shadows on other objects. The objects may be polygons, curved surfaces, or other particles.
  - treated particles as point light sources, each particle contributing a bit to the brightness of a pixel (avoiding hidden surface detection and shadows). However, the particles did not actually cast light on the geometry in the scene; lights had to be added.

Rendering: Streaks & Swept Paths

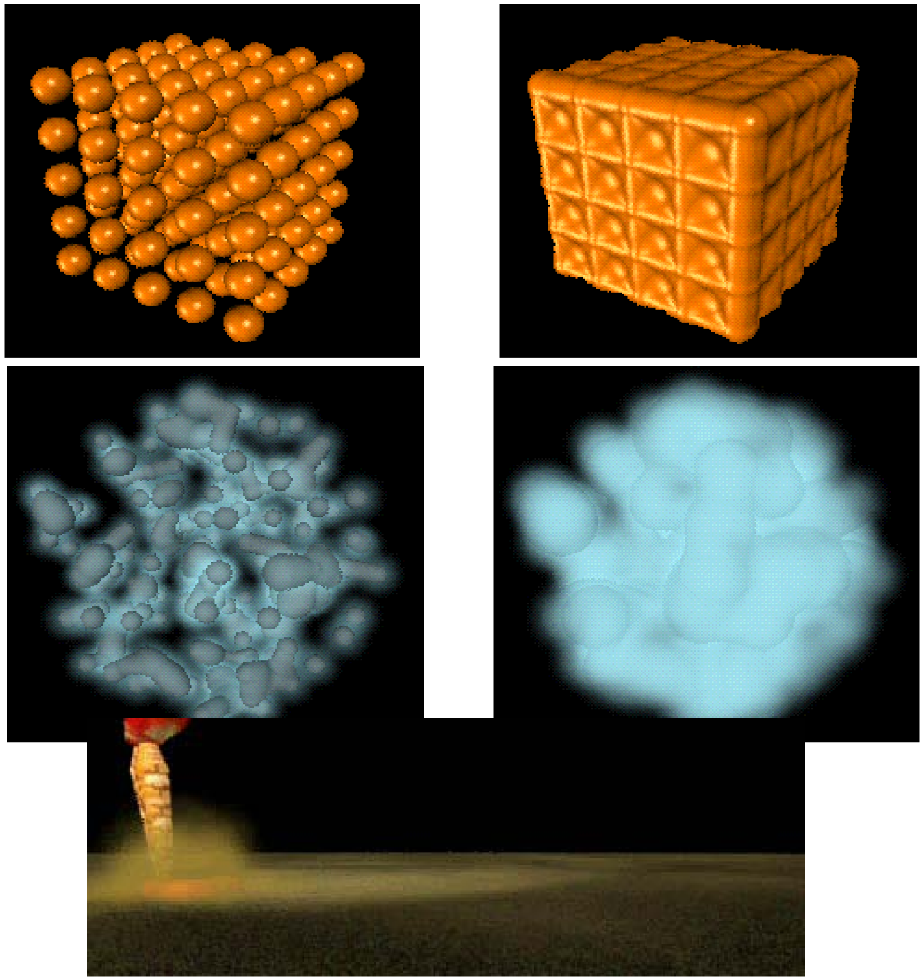
- Entire trajectory of a particle over its lifespan is rendered to produce a static image.
- Green and dark green colors assigned to the particles which are shaded on the basis of the scene's light sources.
- Each particle becomes a blade of grass.



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

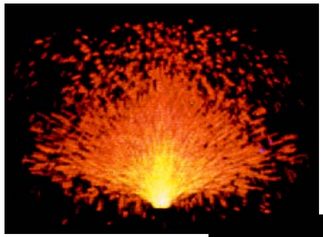
Rendering: Metaballs

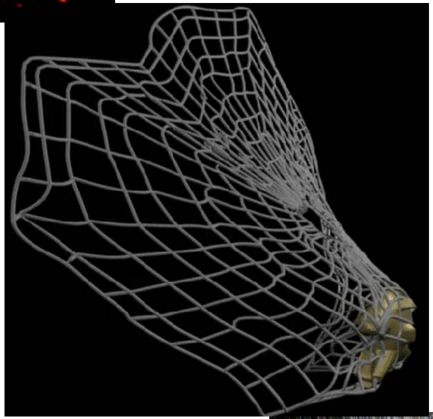
- Metaballs are spheres that blend together to form surfaces.
- By representing particles as metaballs blobby surfaces can be created.



Summary

- Motivation
- Applications
- Formulation
- Dynamics
- Rendering





References

- Reeves W.: *Particle Systems -- A Technique for Modelling a Class of Fuzzy Objects*, Computer Graphics, 17(3), pp. 359-376, 1983
- Witkin A.: *Particle Systems,* Siggraph 2001 Course Notes on Physically Based Modeling
- Baraff D.: *Collision and Contact ,* Siggraph 2001 Course Notes on Physically Based Modeling
- http://www.pixar.com/companyinfo/research/pbm2001/
- http://www.cs.cmu.edu/~djames/15-462/Fall03/assts/assignment5.html

Acknowledgements

- William T. Reeves @ Lucas Films
- Andrew Witkin & David Baraff @ Pixar Animation Studios
- P. Capelluto @ Virginia Univ. (CS)