

CS 4621

Particle Systems

Particle Systems

- Small objects, approximated as point masses
- Rotational motion is ignored
- They can be used in great numbers, without bogging down the system
- Can be used to simulate smoke, fire, clouds, and even cloth
- Reeves '83: Star Trek II: Wrath of Khan





Demos

How do they work?

- Have forces
- Want to find positions
- Earlier we did first order equation
 - Now, second order equation

How do they work?

- Have forces
- Want to find positions
- Integrate the particle equations of motion
- Have a pair of ODEs
 - $\ddot{x} = a = F/m$
 - $\dot{x} = v$

System states

- Every particle has a state s
 - $s = (\text{position, velocity, mass, age, color, ...})$
 - p and v vary with time
 - Each p and v is a 3-vector
- The entire system state is S
 - $S = (p_1, v_1, p_2, v_2, p_3, v_3, ...)$
 - Can think of S as just a vector in $6n$ dimensions
- $P, V, A,$ and F are $3n$ -vectors

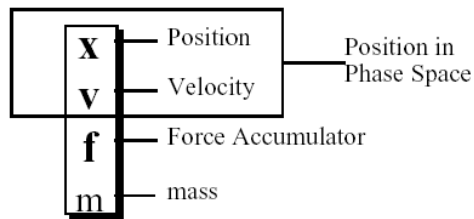
© Kavita Bala, Computer Science, Cornell University

Simulation Loop

- Particle loop
 - Initialize/Emit particles
 - Run integrator (evaluate derivatives)
 - Update particle states
 - Render
 - Repeat!
- Worry about memory
 - Don't allocate/deallocate; recycle!

© Kavita Bala, Computer Science, Cornell University

Particle Implementation

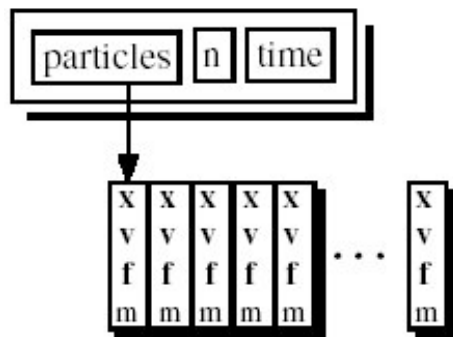


Each particle represented by a minimum of 10 values

Additional values are possible

- Electric charge
- Color
- Particle age

Particle Structure

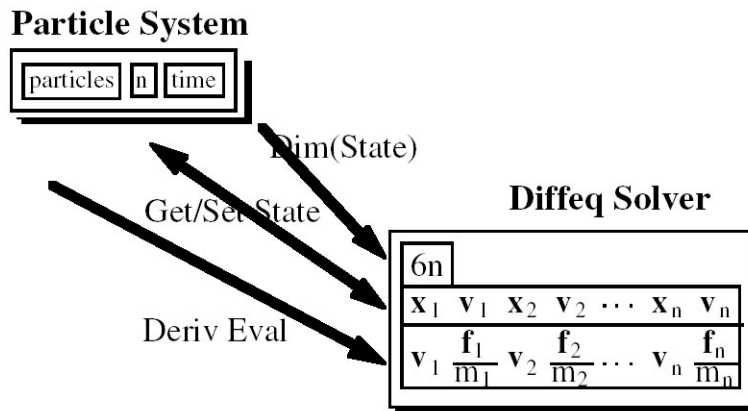


er Science, Cornell University

Integration

- How do we implement an integrator?
 - Write a black-box that works on any f function
 - Takes an initial value at time t , a function f' and timestep h . Returns $f(t+h)$
 - The integrator can be completely separate from the particle representations
 - If your system has complex forces, repeated f' evaluations become the bottleneck

Interface to Integrator



- Particle system must allow the solver to read and write state and call the derivative function

© Kavita Bala, Computer Science, Cornell University

```

/* length of state derivative, and force vectors */
int ParticleDims(ParticleSystem p){
    return(6 * p->n);
};

/* 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];
    }
}

```

© Kavita Bala, Computer Science, Cornell University

```

/* 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++);
    }
}

/* 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;
    }
}

```

© Kavita Bala, Computer Science, Cornell University

```

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 */
}

```

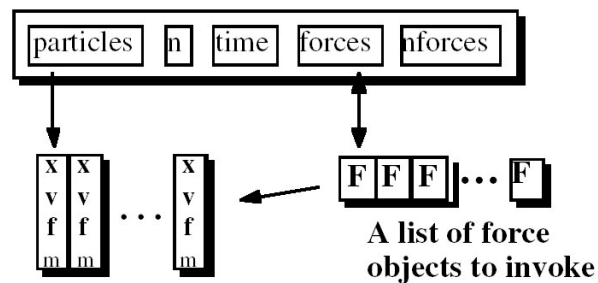
© Kavita Bala, Computer Science, Cornell University

Integration

- Euler Method
 - $S(t+h) = S(t) + \Delta T \cdot S'(t)$
 - What's S' ?
 - $S' = (P', V') = (V, A) = (V, F/m)$
 - Simple to implement
 - Requires only one evaluation of S'
 - Simple enough to be coded directly into the simulation loop

© Kavita Bala, Computer Science, Cornell University

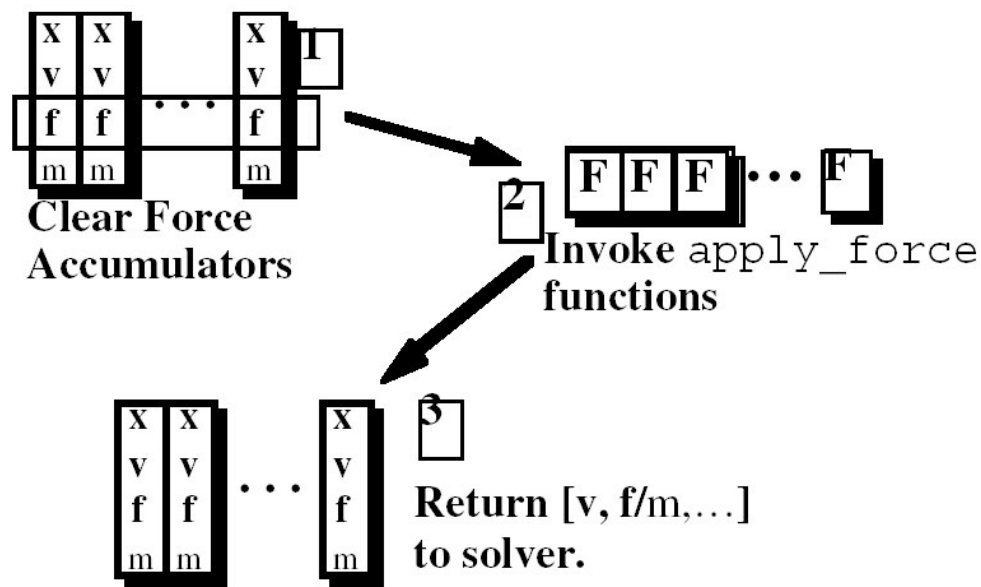
Forces



- Forces are stored at the system level
- They are invoked during the derivative evaluation loop

© Kavita Bala, Computer Science, Cornell University

Derivative Evaluation



© Kavita Bala, Computer Science, Cornell University

Forces

- Typically, have multiple independent forces
 - For each force, add its contribution to each particle
 - Need a force accumulator variable per particle
 - Or accumulate force in the acceleration variable, and divide by m after all forces are accumulated
 - Need to evaluate F at every time step
 - The force on one particle may depend on the positions of all the others

© Kavita Bala, Computer Science, Cornell University

Forces

- Example forces
 - Earth gravity, air resistance
 - Force fields
 - Wind
 - Attractors/Repulsors
 - Vortices
 - Springs, mutual gravitation

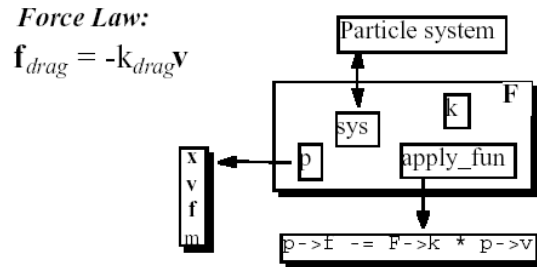
© Kavita Bala, Computer Science, Cornell University

Forces

- Earth Gravity
 - $f = -9.81 * (\text{particle mass in Kg}) * Y$
- Drag
 - $f = -k * v$
- Uniform Wind
 - $f = k$

© Kavita Bala, Computer Science, Cornell University

Applying a force



A Force Object: Viscous Drag

© Kavita Bala, Computer Science, Cornell University

Spring Mass system

- Force between 2 particles a and b
 - R is rest length, k_s is spring constant, k_d is damping constant, $l = (a-b)$



$$\mathbf{f}_a = - \left[k_s (|\mathbf{l}| - r) + k_d \frac{\dot{\mathbf{l}} \cdot \mathbf{l}}{|\mathbf{l}|} \right] \frac{\mathbf{l}}{|\mathbf{l}|}$$

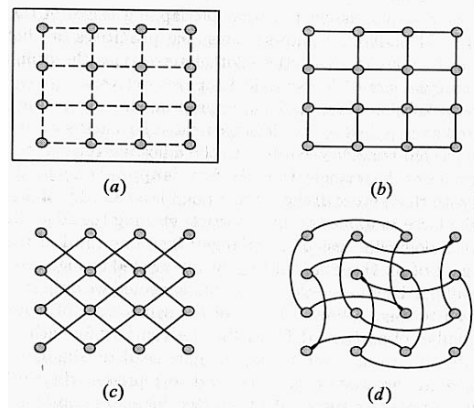
$$\mathbf{f}_b = -\mathbf{f}_a$$

$$\dot{\mathbf{l}} = \mathbf{v}_a - \mathbf{v}_b$$

© Kavita Bala, Computer Science, Cornell University

Interconnected Particles

- Can be used to model cloth or other deformable objects
 - Springs used to model stretch, shear, and bend forces.



© Kavita Bala, Computer Science, Cornell University

Spring Mass System

- Doing a simple spring mass system
 - Very doable
- Complex, robust systems with stiff equations
 - Not that easy to implement

© Kavita Bala, Computer Science, Cornell University

Simulation Loop Recap

- A recap of the loop:
 - Initialize/Emit particles
 - Run integrator (evaluate derivatives)
 - Update particle states
 - Render
 - Repeat!

© Kavita Bala, Computer Science, Cornell University

Emitters

- Usually described as a surface from which particles appear
 - Object with position, orientation
 - Regulates particle “birth” and “death”
 - Usually 1 per particle system
- Many user definable parameters:
 - size, mass, age, emitter size, initial velocity and direction, emission rate, collision detection (internal and external), friction coefficients, global forces, particle split times, delays, and velocities, color evolution, etc.

© Kavita Bala, Computer Science, Cornell University

Particle Systems

- New particles are born, old die
- At each time step
 - Update attributes of all particles
 - Delete old
 - Create new (recycle space)
 - Display current state
- To anti-alias draw line or trajectory from old position to new position