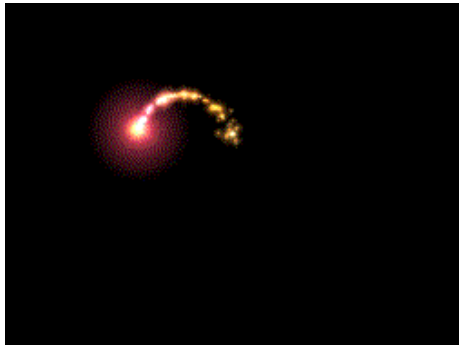
粒子系统(Particle Systems)



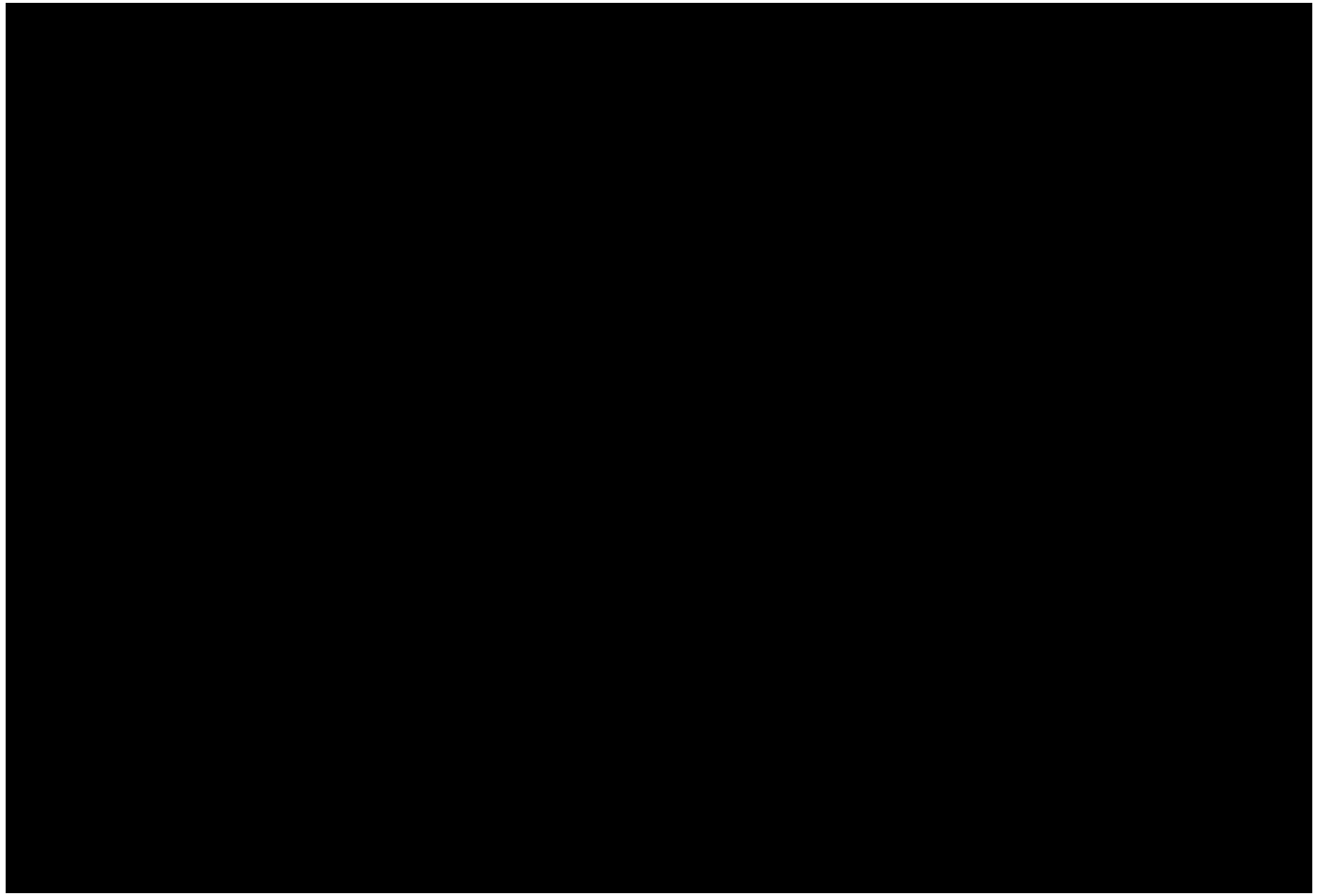
金小刚

Email: jin@cad.zju.edu.cn

浙江大学CAD&CG国家重点实验室

紫金港校区蒙民伟楼512

粒子系统演示

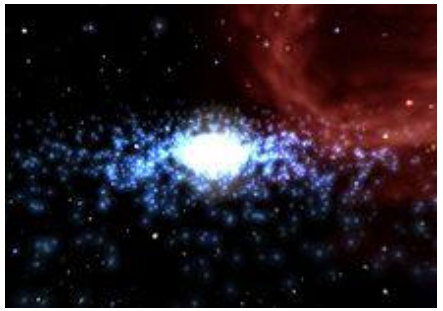


“Very simple procedural rules can create very deep visual effects.”

粒子系统(Particle systems)

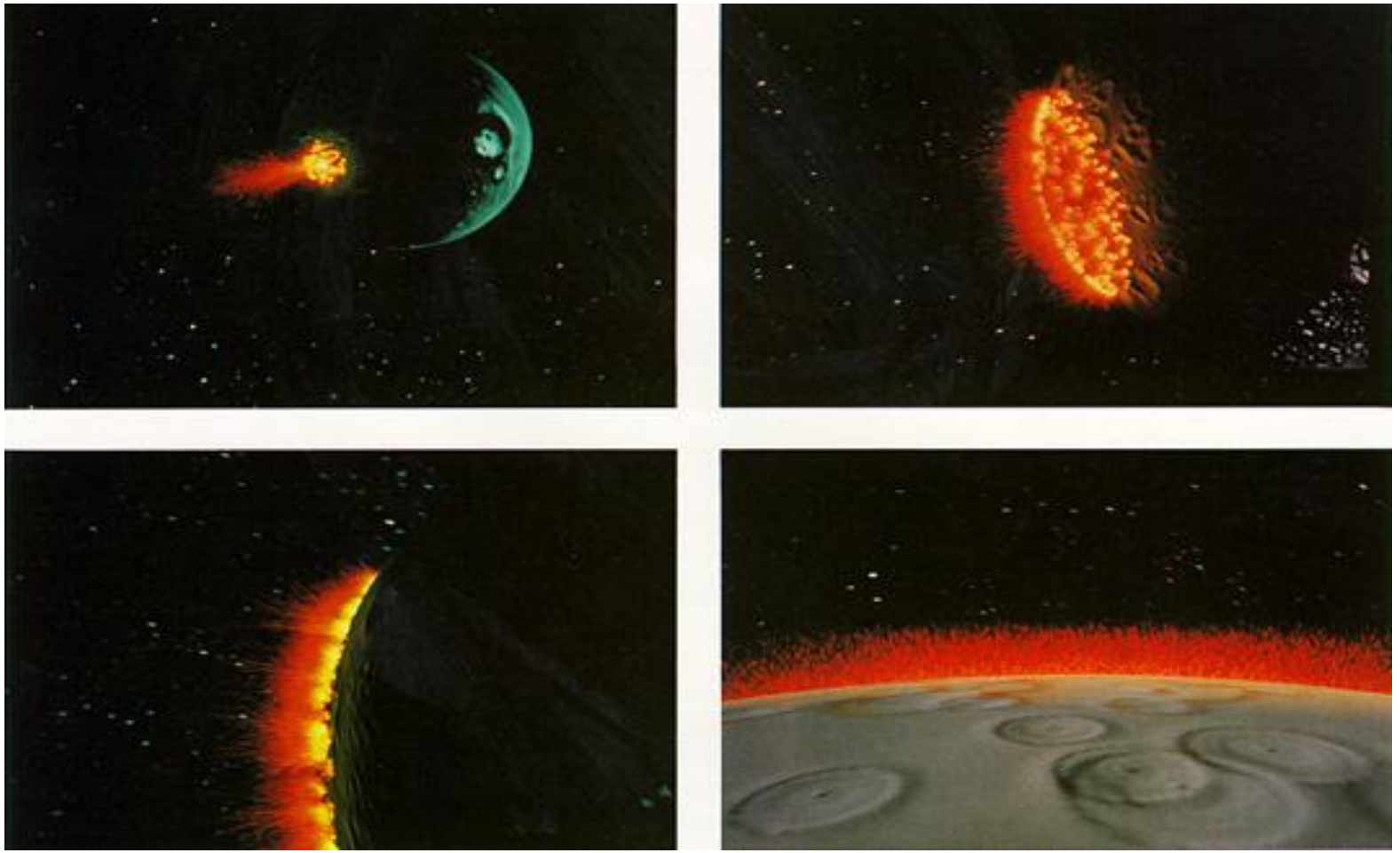
- 粒子系统是最实用的过程动画技术之一，是影视特技、游戏、图形学中生成**视觉特效**的一种**主要方法**。

- 这一方面的先驱是William T. Reeves，他在1983年所发表的Siggraph论文中成功地提出了一种模拟**不规则自然景物**("fuzzy" phenomena)生成和动画的系统，也就是所谓的粒子系统。

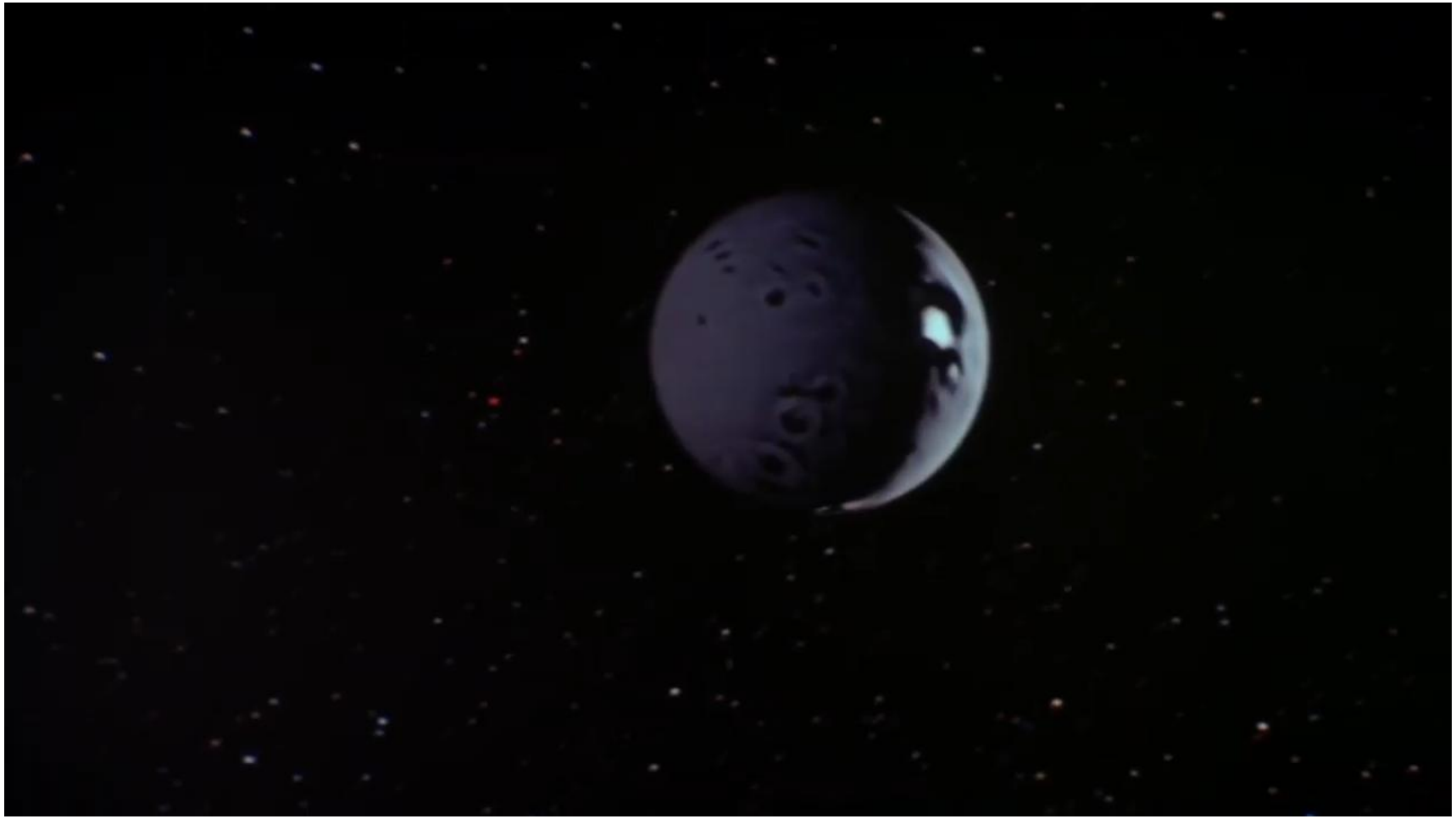


- 粒子系统采用了一套完全不同于以往的造型、绘制方法来构造、绘制景物，**造型和动画巧妙地连成一体**。景物被定义为由成千上万个不规则的、随机分布的粒子所组成，而每个粒子均有一定的生命周期，它们不断改变形状、不断运动。

历史——星际迷航2: 可汗之怒, 1983
(Star Trek II - The Wrath of Khan)



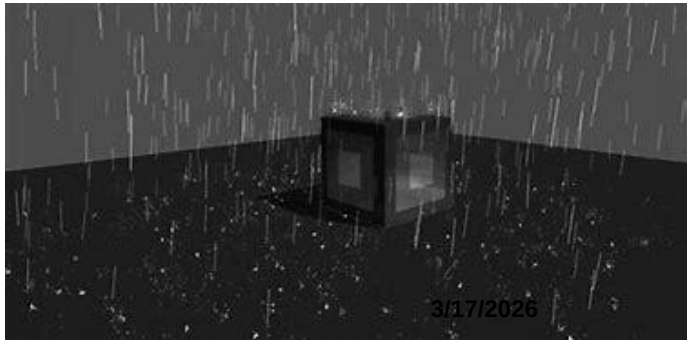
历史——星际迷航2: 可汗之怒, 1983
(Star Trek II - The Wrath of Khan)



01:37开始

应用极为广泛

- 云彩
- 烟雾
- 流动的水(瀑布、水花)
- 爆炸、碎片、尘土
- 火(火花)
- 喷泉
- 烟花
- 下雪、下雨、落叶
- 大群体
- 草、发丝，等等



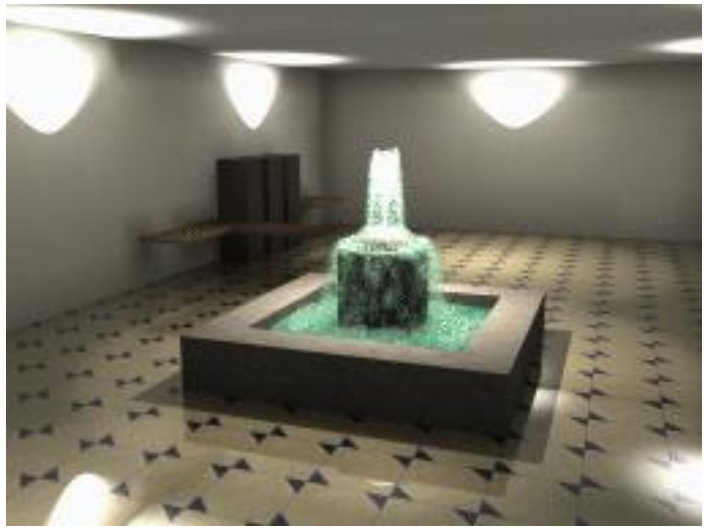
关于William T. Reeves(1959-)

- 参考文献
  - William T. Reeves: Particle Systems - a Technique for Modeling a Class of Fuzzy Objects. ACM Trans. Graph. 2(2): 91-108 (1983)
  - William T. Reeves, Ricki Balu: Approximate and probabilistic algorithms for shading and rendering structured particle systems. SIGGRAPH 1985: 313-322
- 截止到2026年3月17日，论文"Particle Systems”被引用次数：2767次。
- 因《锡铁小兵Tin Toy》1988年获得第61届奥斯卡金像奖最佳动画短片(与John Lasseter)。
  - Tin Toy (1988) (Producer/Technical Director/Modeler/Additional Animator)
  - Toy Story (1995) (Supervising Technical Director / Modeling / Animation System Development / Renderman Software Development)
  - A Bug's Life (1998) (Supervising Technical Director)
  - Finding Nemo (2003) (Lead Technical Development)

应用例子





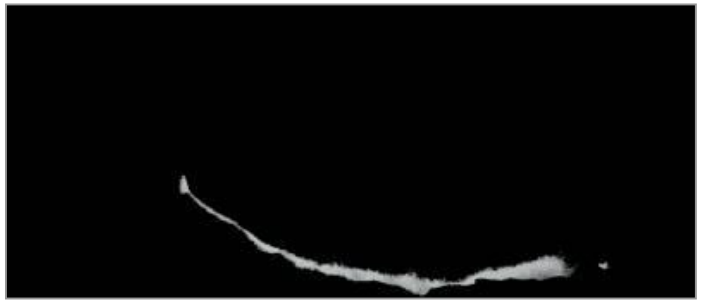


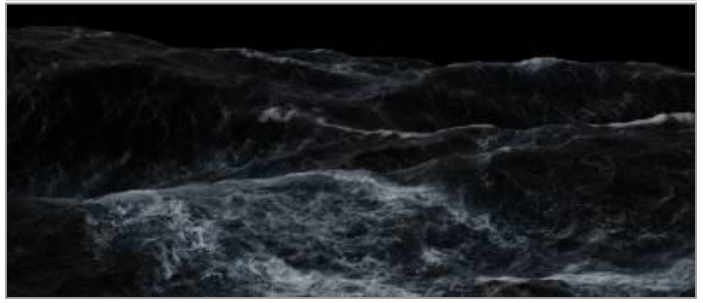

应用例子

泡沫、溅起的水花采用粒子系统生成









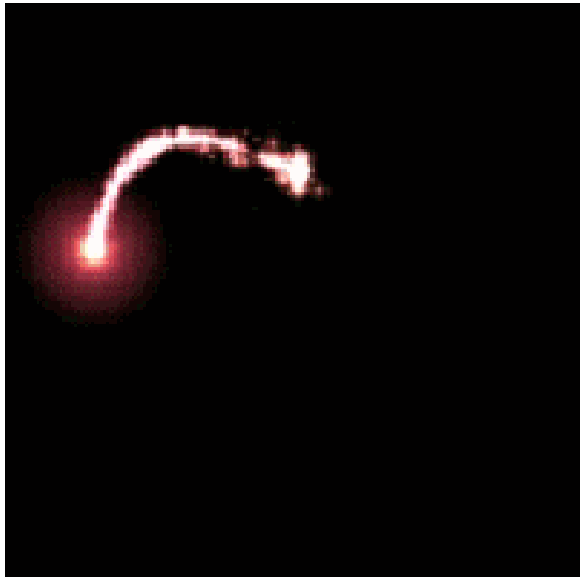

《完美风暴》剧照

粒子系统——假设

- 粒子一般与其它粒子不碰撞
- 除非处于聚集状态，粒子不向其它粒子投射阴影
- 粒子只向其它环境投射阴影
- 粒子不反射光
- 粒子通常有有限的生命周期

粒子系统——每一帧中的步骤

- 粒子系统的基本思想是将许多简单形状的微小粒子作为基本元素聚集起来形成一个**不规则的模糊物体**，每个粒子均经历出生、成长、衰老和死亡的过程，与粒子有关的每一个参数均将受到一个随机过程的控制。生成粒子系统**某一帧画面**的基本步骤是：
  - 生成新的粒子并加入系统中；
  - 赋予每一新粒子以一定的属性；
  - 删除那些已经超过其生命周期的粒子；
  - 根据粒子的动态属性对粒子进行移动和变换；
  - 绘制并显示由有生命的粒子组成的图形。



粒子生成

- 对于每一帧，根据一个**控制的随机过程**生成粒子：
  - 用户可控制每帧的平均粒子数和其概率分布
  - 粒子数可以是时间的函数

- 方法一：
  粒子数$n=m+v*\text{rand}()$
  其中，m为平均粒子数；
  rand()返回一个[-1,1]之间的随机数；
  v为变化幅度；

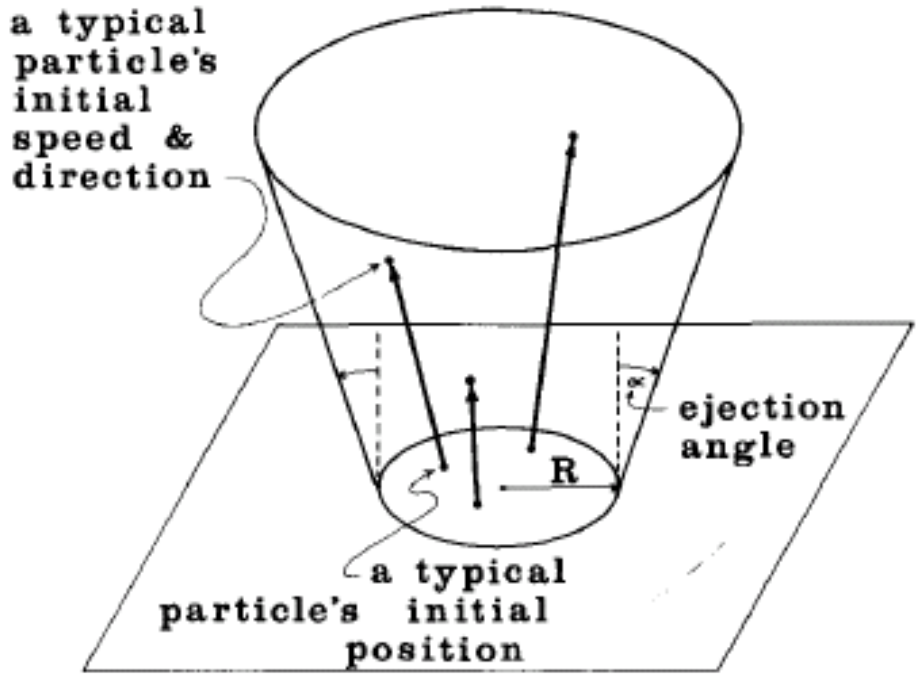
- 方法二：
  粒子数$n=(m+v*\text{rand}())*A$
  其中，m为平均粒子数；
  rand()返回一个[-1,1]之间的随机数；
  v为变化幅度；
  A为物体的屏幕面积（Screen Area）

粒子属性(Attributes of a Particle)

- 一个粒子(质点)的属性可以包括
  - 位置
  - 速度
  - 大小
  - 质量
  - 力累加器(force accumulator)
  - 生命周期
  - 绘制属性 (形状参数, 颜色, 透明度)
  - ...

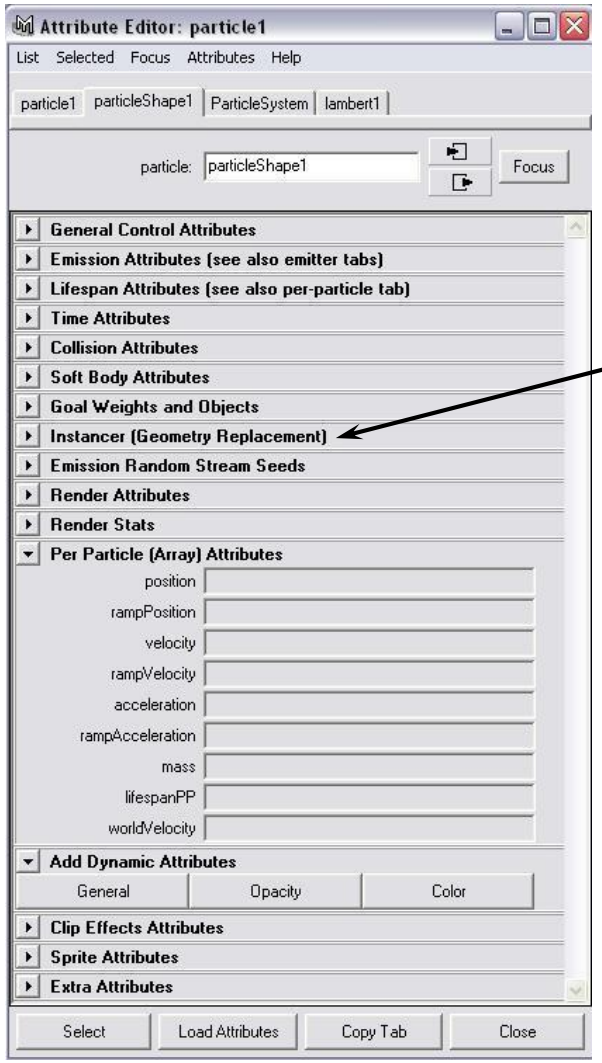
- 当粒子生成时，对其属性进行初始化

  Value = mean + Rand()*variance



粒子系统属性
(Attributes of a Particle System)

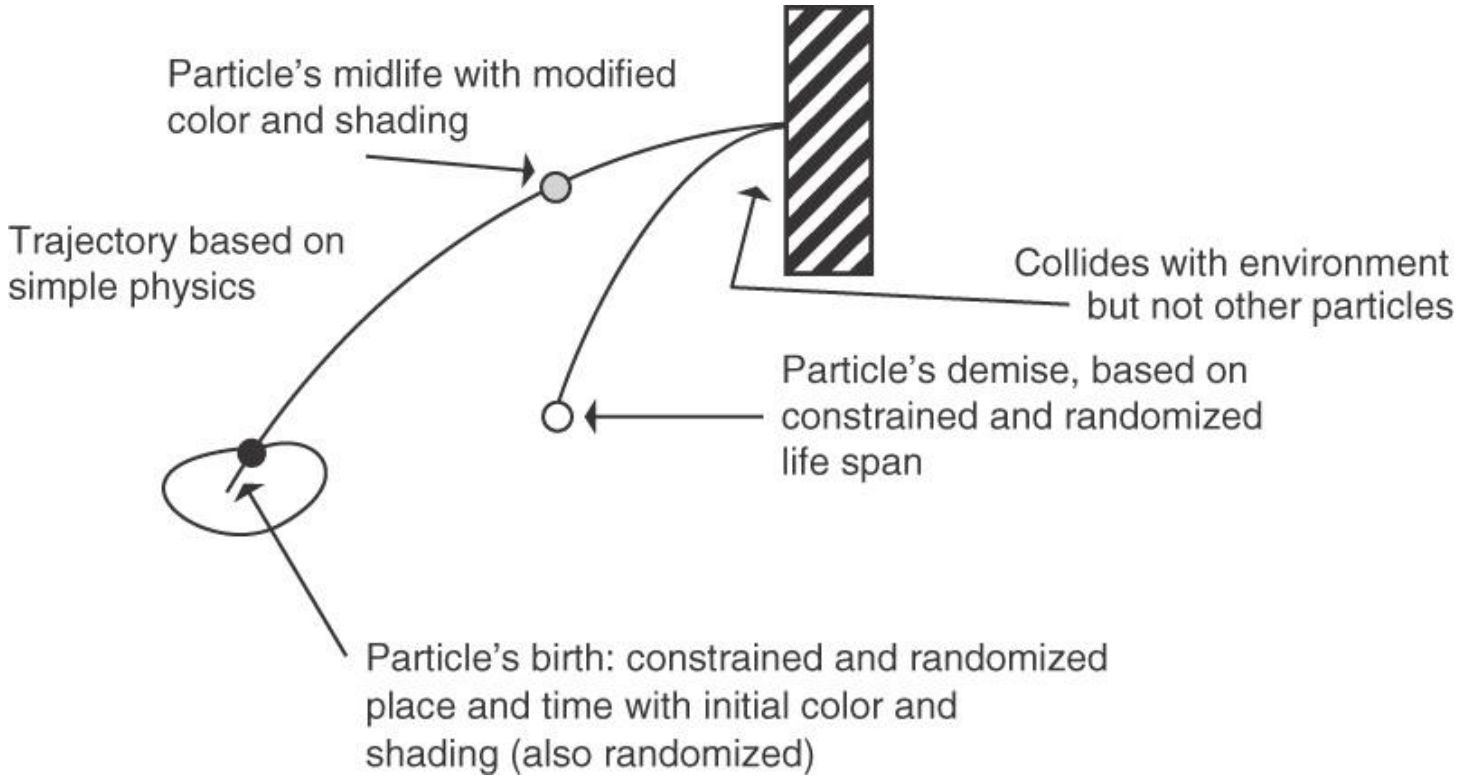
- Particle List：指向一个粒子系统的指针
- Position：粒子系统的位置
- Emission Rate：决定生成新粒子的创建方法
- Forces ：加入力可极大地增强物理真实程度
- Current State：用来记录系统的状态，例如在模拟爆炸时，粒子系统会很快停止发射
- Blending：用来设置粒子相互混合的方式
- Representation：例如，粒子的纹理
- …



Instancer (Geometry Replacement)

https://help.autodesk.com/view/MAYAUL/2025/ENU/?guid=GUID-E882C129-B93D-4394-931B-453026C9D374

粒子的生命周期

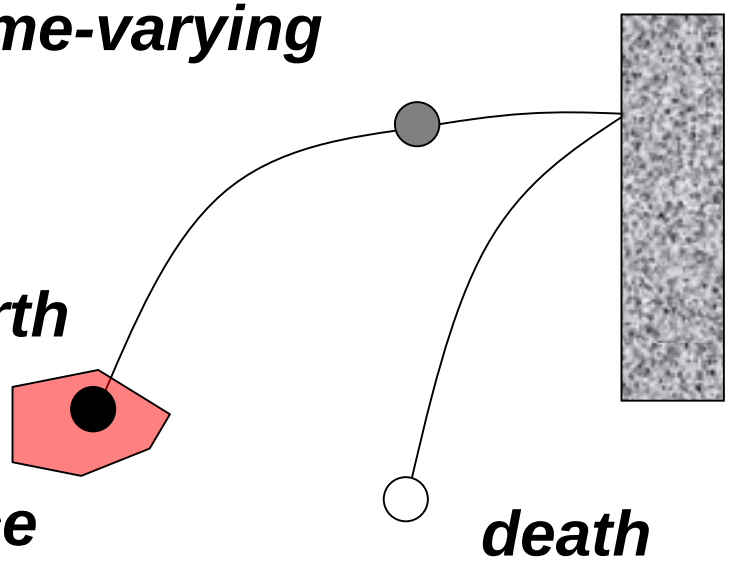


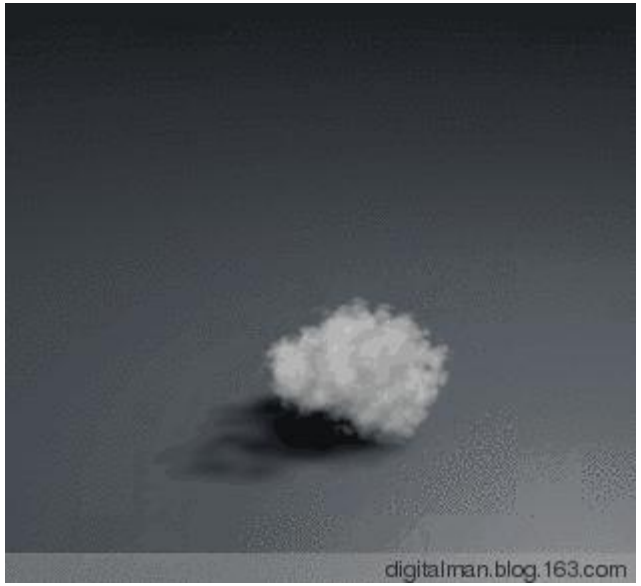
在每一新帧，每个粒子的生命周期减1，当该属性变成0时，把该粒子从系统中删除

粒子的生命周期

Reaction to environment

Aging: Time-varying attributes





粒子消亡

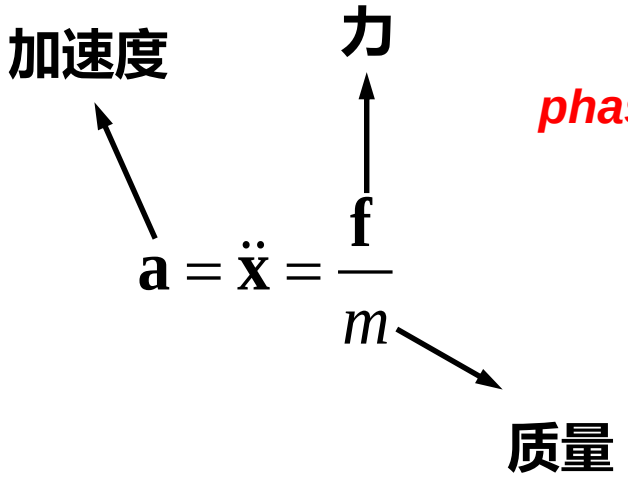
- 当如下情形发生时，粒子将销毁：
  - 生命周期结束
    - 当创建一个粒子时，将赋予它一生命周期。每模拟一帧，该值减1。当该值为0时，该粒子将被销毁。
  - 淡出(Fades out)
    - 当粒子的颜色/不透明度小于一用户给定的阈值时，该粒子不可见并将被销毁。
  - 移走(Moves away)
    - 不在视域。

粒子动画

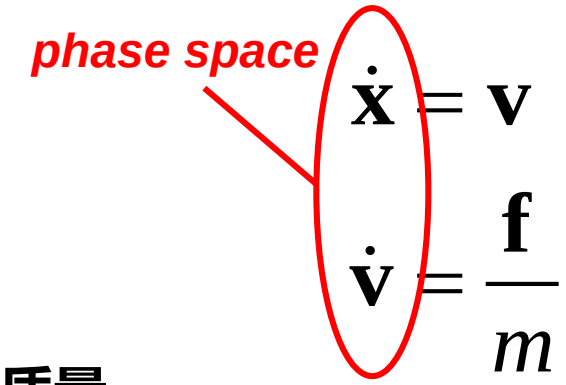
- 在生命周期内，对每个粒子设置动画
  - 包括位置、速度和绘制属性
- 位置和速度
  - 根据合力，计算粒子的加速度，进而更新粒子的速度和位置
- 颜色和透明度可以是时间（其剩余的生命周期、其高度等）的函数，形状可以是速度的函数

粒子上的力

- 粒子上的力可以是一元力(unary force)、粒子的偶力(particle pair force)或环境力
  - 一元力
    - 重力(Gravity), 粘性阻力(viscous drag)
  - 粒子的偶力
    - 可表示为弹簧阻尼器
  - 环境力
    - 根据粒子与环境的关系来得到
- 粒子对力做出反应
- 我们用微分方程来表示



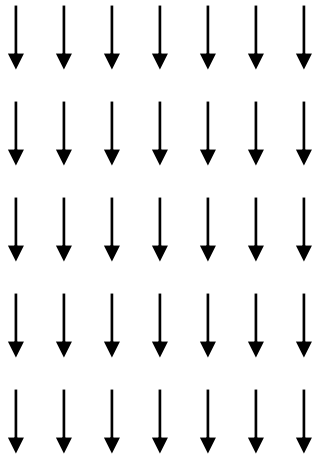
二阶偏微分方程



一阶偏微分方程

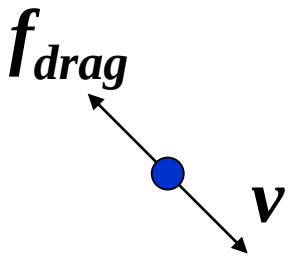
一元力(Unary Forces)

- 只与单个粒子相关的力



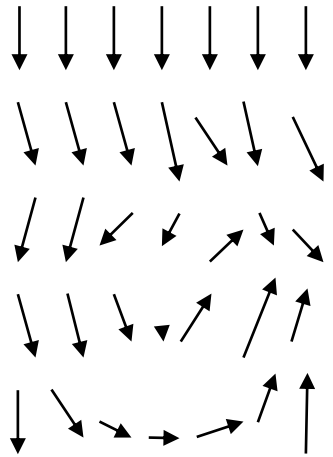
重力
Gravity

$$f = mg$$



粘性阻力
Viscous Drag

$$f = -k_d v$$

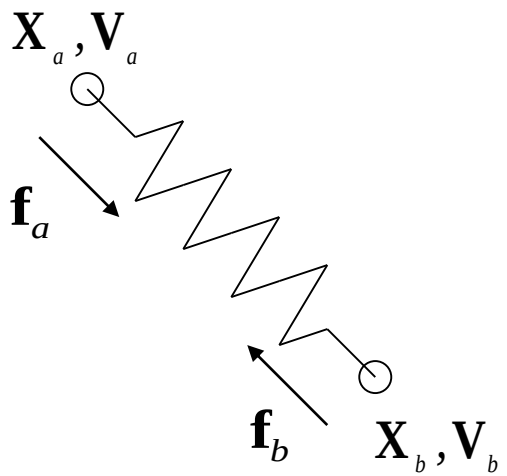


风场
Wind Fields

$$f = k v_{wind}$$

n元力(n-ary Forces)

- 依赖于*n*个粒子的力
  - 例如: 两个粒子之间的二元力——弹簧阻尼力



弹簧

$$\mathbf{f}_a = \underbrace{-k_s\left(\left|\mathbf{x}_a - \mathbf{x}_b\right| - l_0\right)\frac{\mathbf{x}_a - \mathbf{x}_b}{\left|\mathbf{x}_a - \mathbf{x}_b\right|}}_{\text{弹簧力}} \underbrace{-k_d\left(\frac{(\mathbf{v}_a - \mathbf{v}_b)\bullet(\mathbf{x}_a - \mathbf{x}_b)}{\left|\mathbf{x}_a - \mathbf{x}_b\right|}\right)\frac{\mathbf{x}_a - \mathbf{x}_b}{\left|\mathbf{x}_a - \mathbf{x}_b\right|}}_{\text{阻尼力}}$$

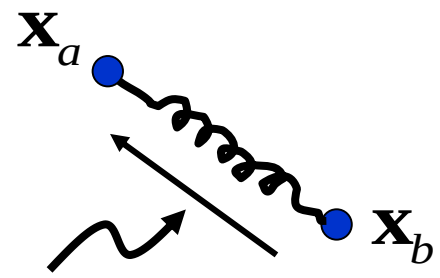
n元力: 弹簧力

- 如果粒子比其静止位置远，则弹簧力需要把它拉回:

$$l = |\mathbf{x}_a - \mathbf{x}_b| - l_0 > 0 \qquad \mathbf{f}_a = -k_s l \mathbf{d}, \quad k_s > 0$$

- 如果粒子比其静态位置近，则弹簧力需要把它拉开:

$$l = |\mathbf{x}_a - \mathbf{x}_b| - l_0 < 0 \qquad \mathbf{f}_a = -k_s l \mathbf{d}, \quad k_s > 0$$



$$\mathbf{d} = \frac{\mathbf{x}_a - \mathbf{x}_b}{|\mathbf{x}_a - \mathbf{x}_b|}$$

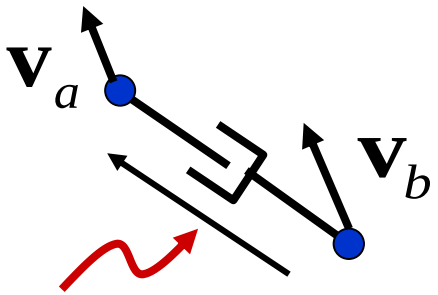
- 把上述两种情形相结合:

$$\mathbf{f}_a = -k_s(|\mathbf{x}_a - \mathbf{x}_b| - l_0)\mathbf{d}, \quad k_s > 0$$

$$= -k_s(|\mathbf{x}_a - \mathbf{x}_b| - l_0)\frac{\mathbf{x}_a - \mathbf{x}_b}{|\mathbf{x}_a - \mathbf{x}_b|}$$

n元力: 阻尼力

- 阻尼力/粘性阻力使粒子系统在外力消失后，渐渐回到静止状态
- **推荐**：每个粒子至少赋予一定的阻尼力
- 但是，过度的阻尼力会使得粒子像在糖浆上漂浮一样 (能量消失太快，不敏感)



$$\mathbf{d} = \frac{\mathbf{x}_a - \mathbf{x}_b}{\left|\mathbf{x}_a - \mathbf{x}_b\right|}$$

- 如果两个粒子正在**远离**，阻尼力会把他们拉回

$$(\mathbf{v}_a - \mathbf{v}_b) \bullet \mathbf{d} > 0$$

- 如果两个粒子正在**靠近**，阻尼力会把他们推开

$$(\mathbf{v}_a - \mathbf{v}_b) \bullet \mathbf{d} < 0$$

- 结合这两种情形:

$$\mathbf{f}_a = -k_d\left(\left|\mathbf{v}_a - \mathbf{v}_b\right| \bullet \mathbf{d}\right)\mathbf{d}, \quad k_d > 0$$

$$= -k_d\left(\frac{(\mathbf{v}_a - \mathbf{v}_b) \bullet (\mathbf{x}_a - \mathbf{x}_b)}{\left|\mathbf{x}_a - \mathbf{x}_b\right|}\right)\frac{(\mathbf{x}_a - \mathbf{x}_b)}{\left|\mathbf{x}_a - \mathbf{x}_b\right|}$$

软约束(Soft Constraints)

软约束方程

- 通常，指定一个**约束条件并使得它成立**比直接指定一个力来达到某种约束条件更加容易；
- **约束(或行为)函数**指定一个我们希望满足的条件

$$\mathbf{C}(\mathbf{x}_a, \mathbf{x}_b) = \mathbf{x}_a - \mathbf{x}_b = \mathbf{0}$$

或

$$C(\mathbf{x}_a, \mathbf{x}_b) = |\mathbf{x}_a - \mathbf{x}_b| - r = 0$$

约束条件

But how do we make it true?

能量方程(Energy Functions)

- 使得约束能量函数**最小化:**

$$E = \frac{k_s}{2}\mathbf{C} \bullet \mathbf{C}$$

- 从而得到一个作用在粒子上的**力**，使得满足约束条件:

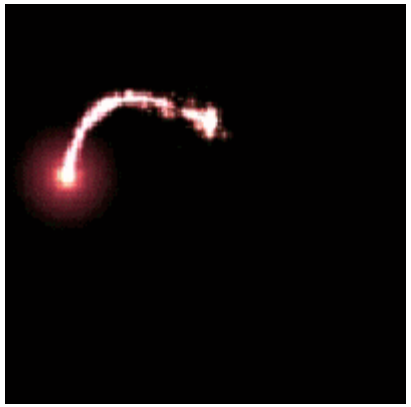
$$\mathbf{f}_i = -\frac{\partial E}{\partial \mathbf{x}_i} = -k_s \frac{\partial \mathbf{C}}{\partial \mathbf{x}_i}\mathbf{C}$$

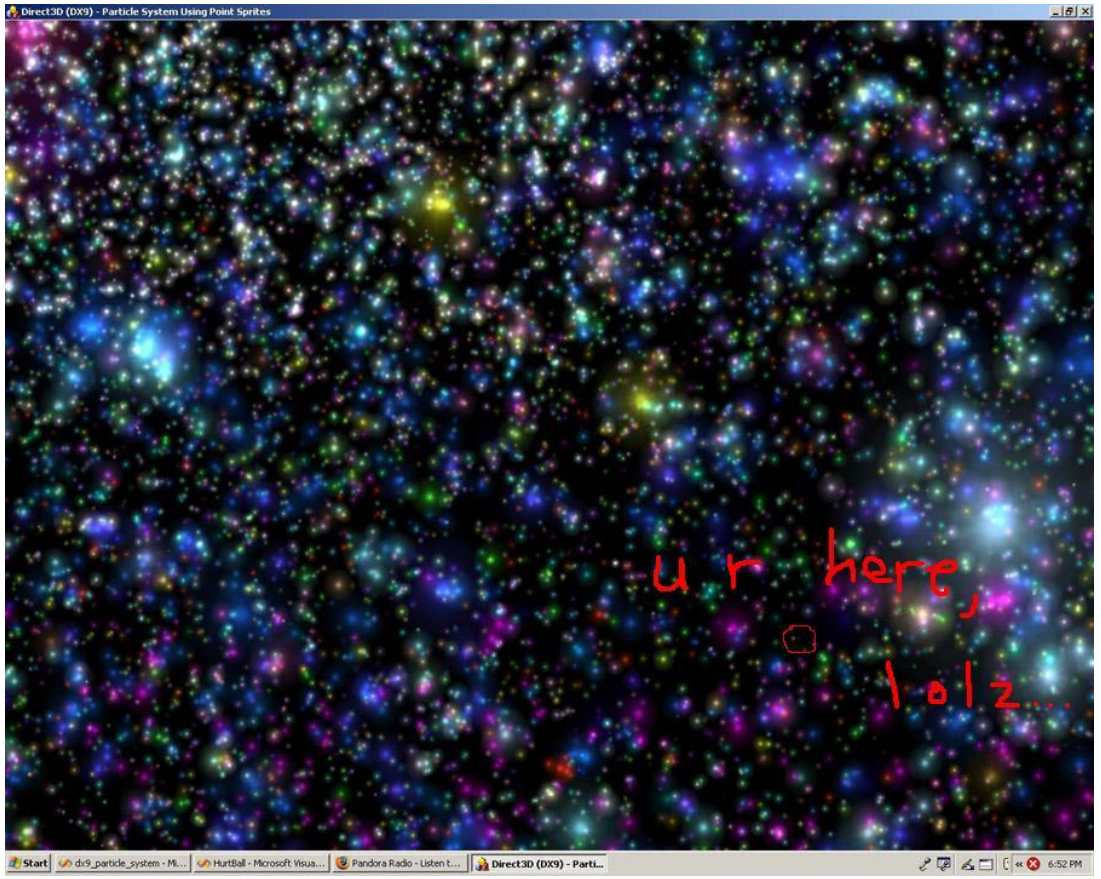
$$\mathbf{f}_i^{dampening} = -k_d \frac{\partial \mathbf{C}}{\partial \mathbf{x}_i}\mathbf{C}$$

粒子的绘制（Rendering）

- 包括如下方法：
  - 把每个粒子当成一个**点光源**
    - 把每个粒子绘制成一个小的图元；
    - 把映射到同一像素的颜色相加；
  - 把每个粒子建模为一个**带纹理的广告牌**(textured billboard)
    - 纹理多边形面向视点
  - 把每个粒子看成**元球**，绘制整个元球系统(用于模拟水等液态效果)
  - …

粒子的绘制——把粒子当光源





粒子的绘制——把粒子当光源

Hanli Zhao, Ran Fan, Charlie C. L. Wang, Xiaogang Jin, Yuwei Meng: Fireworks controller. Computer Animation and Virtual Worlds, 20(2-3): 185-194 (2009)

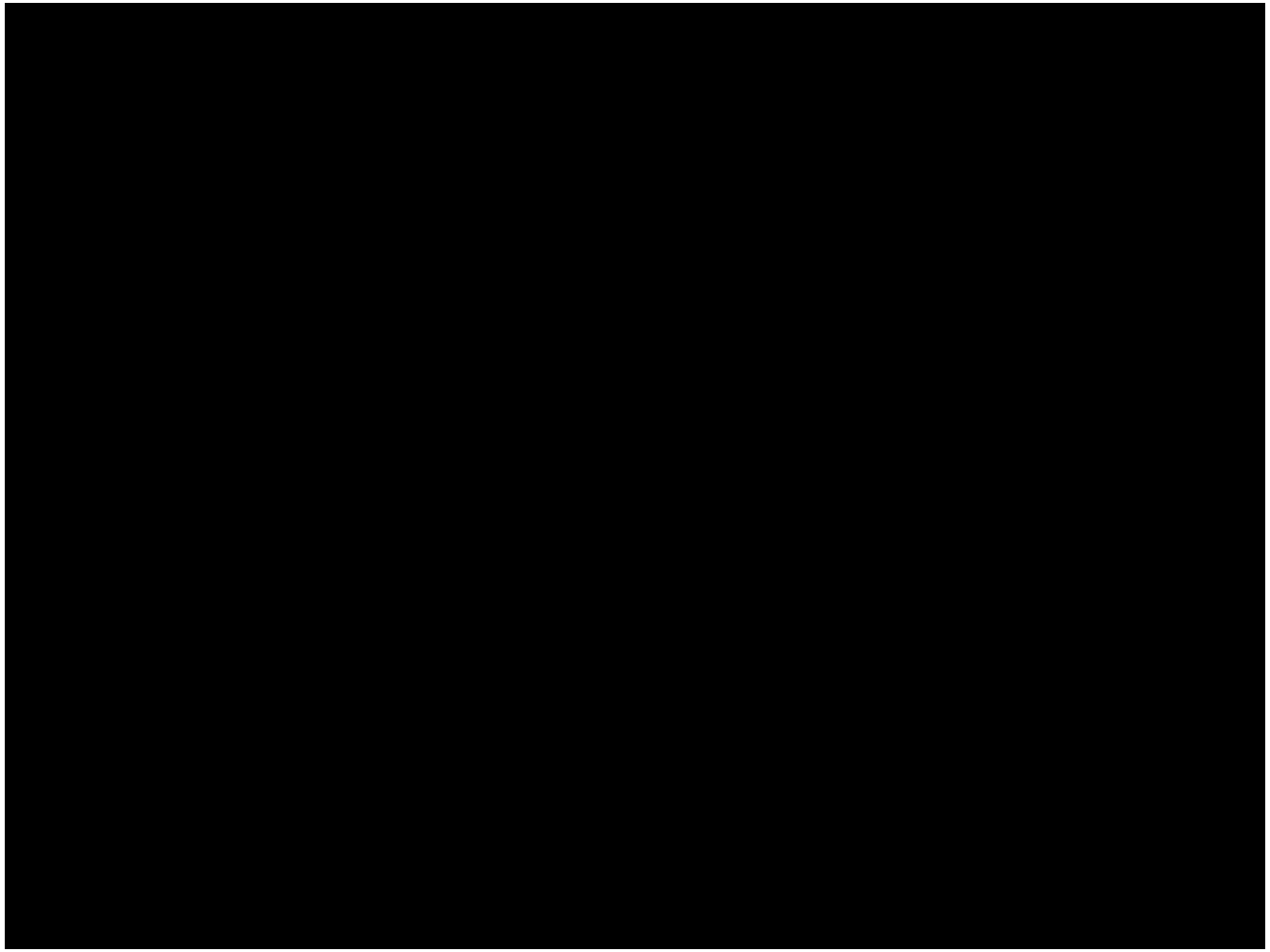
天梯：蔡国强的艺术

- 21年只为打造80秒烟花！他震撼张艺谋、感动5000万网友

杭州亚运会组委会

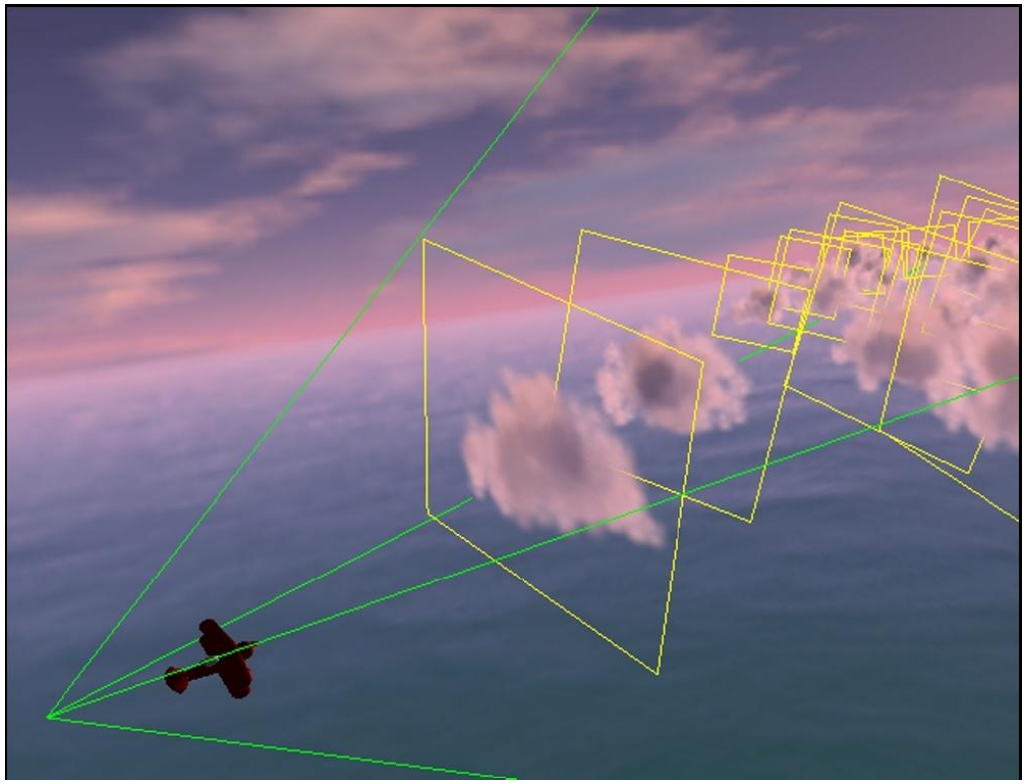
Xiaoyu Cui, Ruifan Cai, Xiangjun Tang, Zhigang Deng, Xiaogang Jin: Sketch-based shape-constrained fireworks simulation in head-mounted virtual reality. Comput. Animat. Virtual Worlds 31(2) (2020): e1920.

粒子绘制—把粒子作为带纹理的广告牌



粒子绘制—把粒子作为带纹理的广告牌

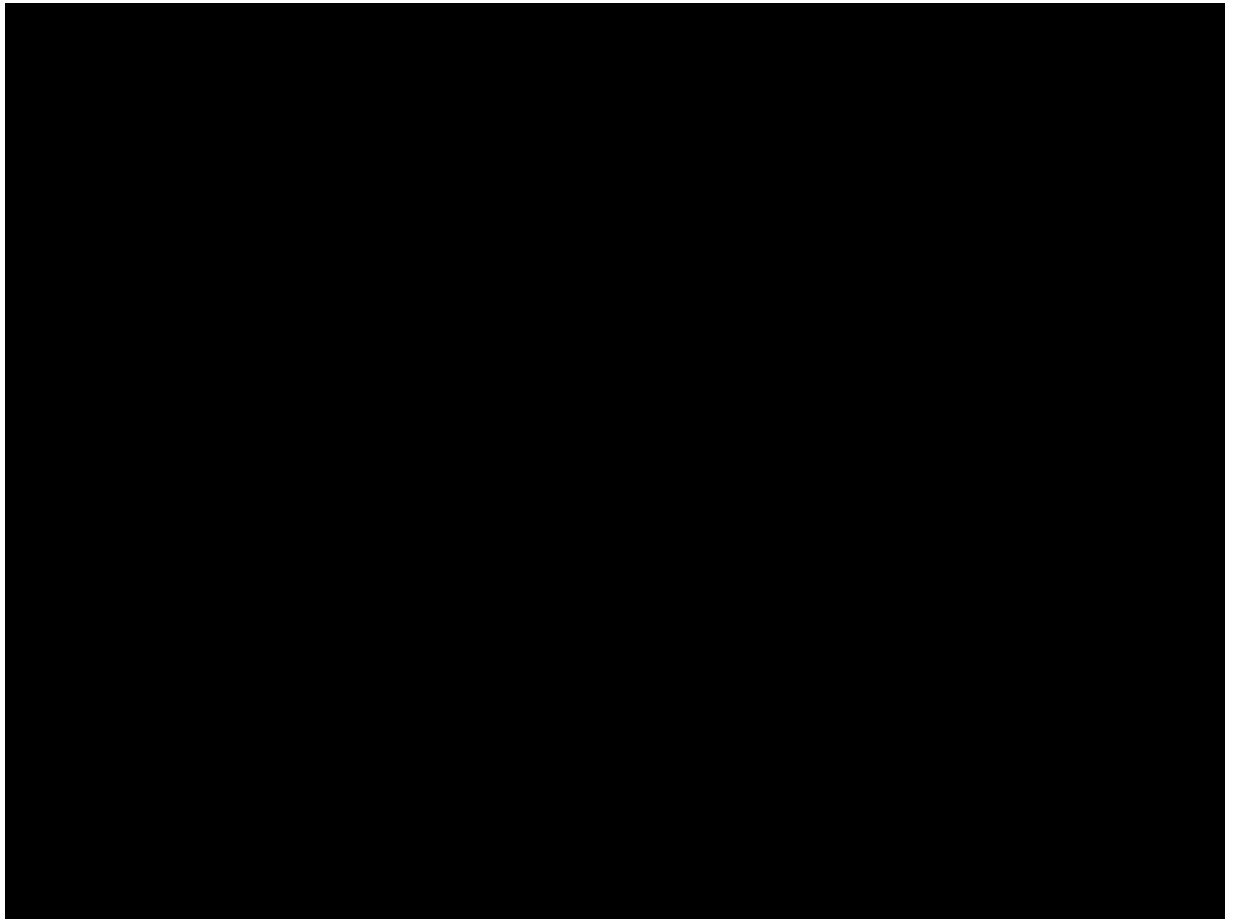
- 一个impostor：把一片云用一个带纹理的广告牌来代替(纹理图像为某一视点的云)
  - 如果视点位置的变化将带来较大的误差，则把纹理图像更新
    - Impostors对很多帧可重用
- 通过采用impostors, 我们可通过实时绘制成千上万的粒子来模拟云彩场景
- 参考：Niniane Wang, “Realistic and Fast Cloud Rendering”, Journal of Graphics, Gpu, and Game Tools, 2004, 9(3): 21-40.



Impostors: textured, view-oriented polygons

粒子绘制—把粒子作为带纹理的广告牌

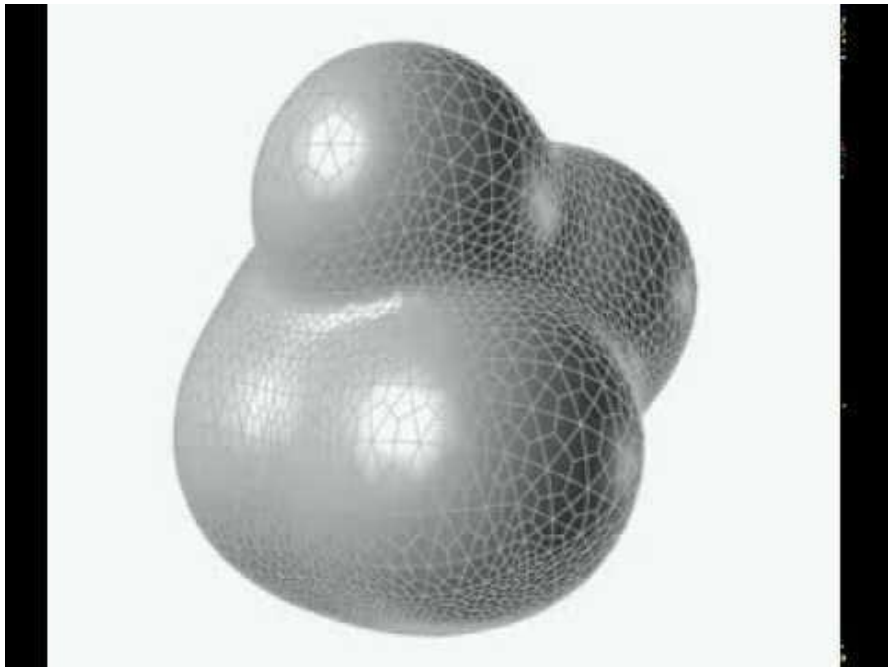
粒子绘制—把粒子作为带纹理的广告牌

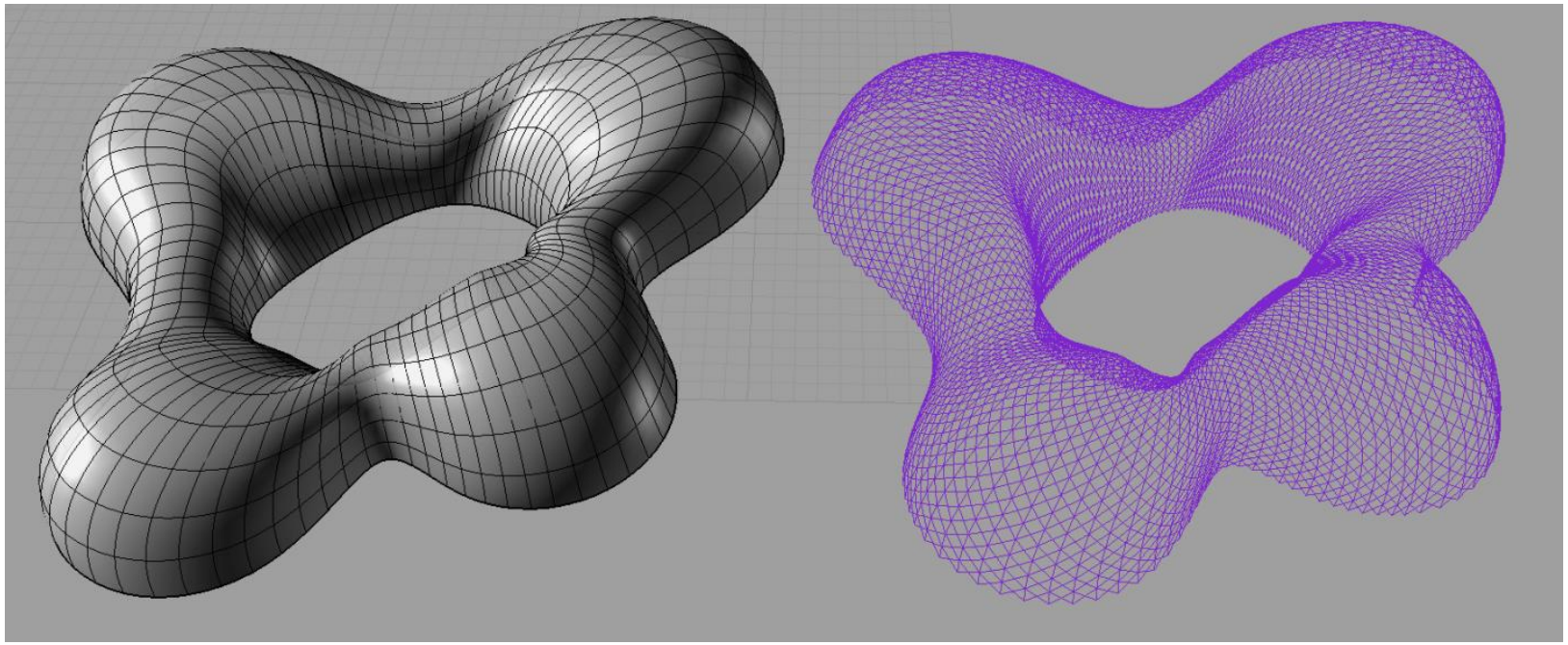


Video: Cloud rendering by Niniane Wang http://www.ofb.net/~niniane/clouds/

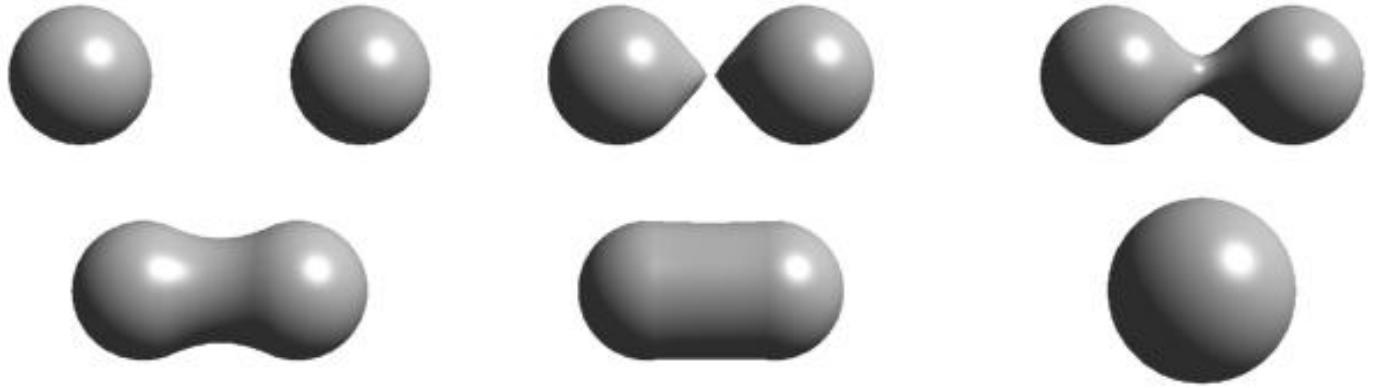
粒子绘制——把粒子作为元球

- 元球造型是一种隐式曲面造型技术，该技术采用具有等势场值的点集来定义曲面。因此，元球生成的面实际上是一张等势面。





粒子绘制——把粒子作为元球



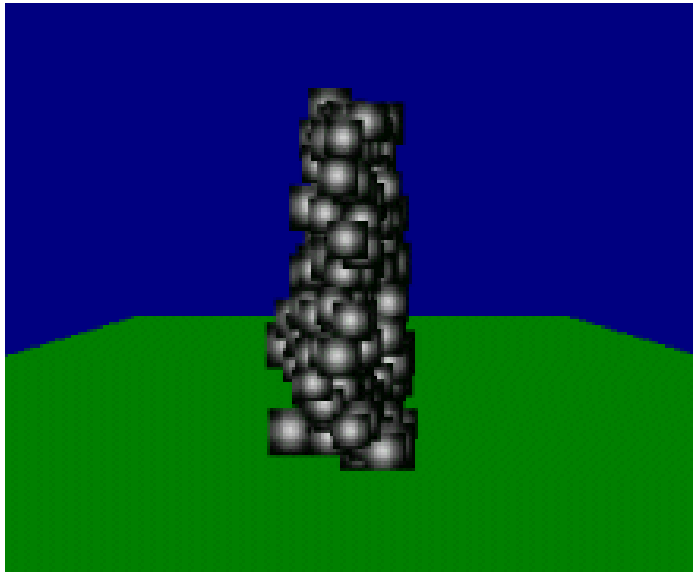
- 元球相互靠近到一定距离产生变形，再进一步靠近时则融合成光滑表面。
- 以两个元球为例，元球靠近时的变形过程如上图所示。最初是两个独立的球，彼此接近时，相对的面开始隆起变形，接近到 一定程度就会象水滴（水银）一样融合成一个面。然后变成花生形状、胶囊形状、最后变成一个球。
- 上述过程实际上提供了一种模拟两滴水 (水银)融合的动画过程，而用参数曲面和多边形网格造型方法来模拟此类动画过程是很困难的。
- 隐式曲面很适合表示可变形和可变拓扑的物体，因而对动画非常有用(如morphing)。

粒子绘制——把粒子作为元球

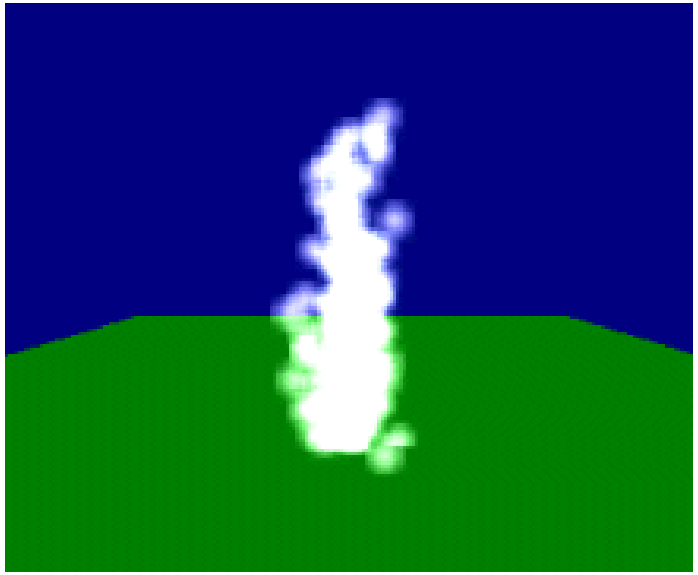
粒子绘制——把粒子作为元球

Terminator 3 T-X gets Magnetized

采用Alpha Blending进行粒子的绘制

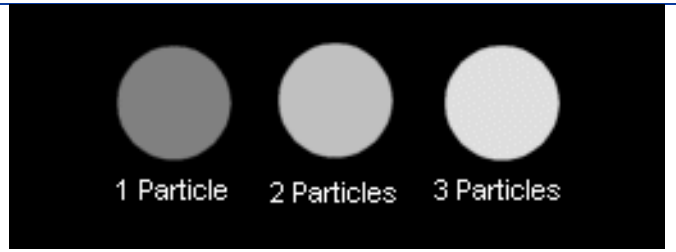


Without alpha-blending : particles are simply textured rectangles.



Alpha-blending is enabled

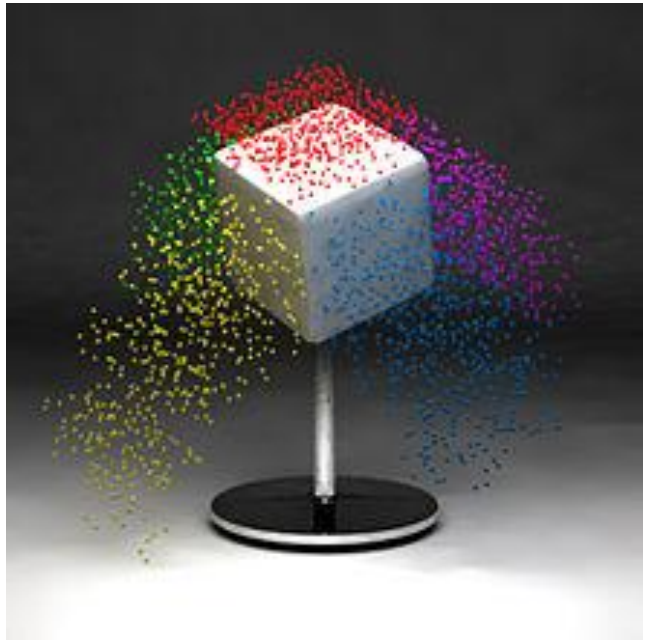
$$FinalColor = DestColor \times DestBlendFactor + SrcColor \times SrcBlendFactor$$



The more particles are on top of each other, the brighter the color in that area should be

“Snowflakes” versus “Hair”

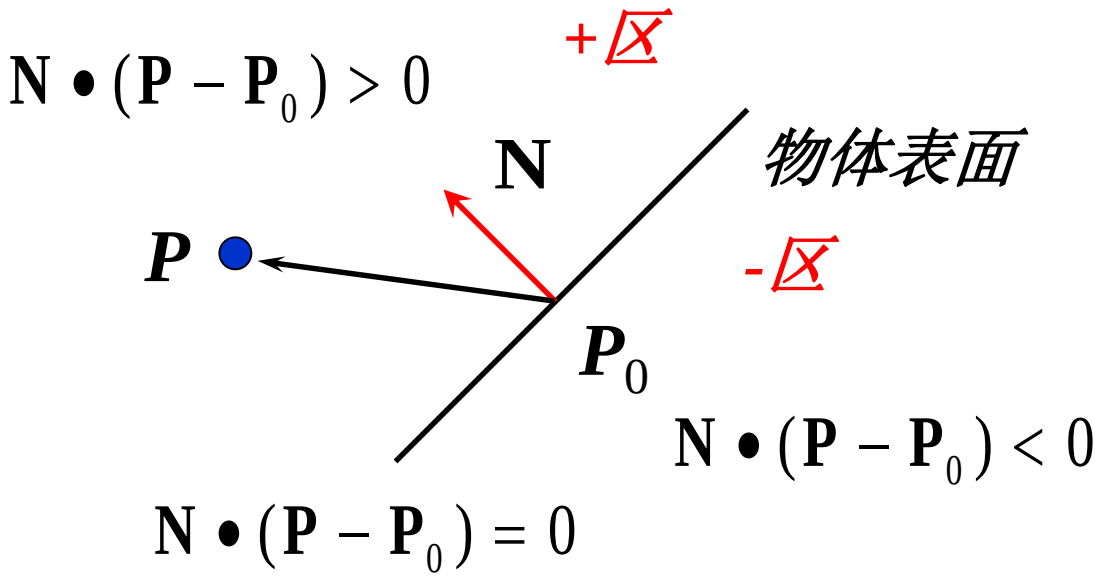
- 粒子系统可以是带动画的或者静止的。也就是说，每个粒子可在其生命周期中逐帧绘制，也可只绘制一次。
- 只绘制一次的丝状物体：头发、毛、草等。可用速度、力场、发射速度、偏移参数等控制。



一个立方体源发射5000个带动画的粒子，受重力影响　　***同样的立方体用静止的轨迹粒子(像发丝)绘制***

粒子的碰撞检测

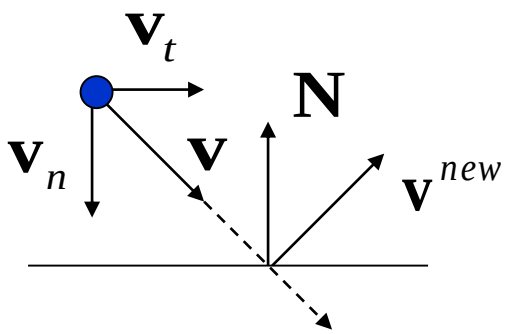
- 判断一个粒子是否与物体发生碰撞



- 当且仅当$\mathbf{N}\bullet(\mathbf{P}-\mathbf{P}_0)<0$时，粒子与平面发生碰撞

碰撞响应(Collision Response)

- 当一个粒子已经发生碰撞时，该做什么?
- 正确的做法是：把模拟**回退到接触点**(the (point of contact))
- 只需修改粒子的**位置**和**速度**



碰撞后，新的速度：

$$\mathbf{v}^{new} = -\varepsilon \mathbf{v}_n + \mathbf{v}_t$$

回弹系数
(coefficient of restitution)

粒子运动方程的求解

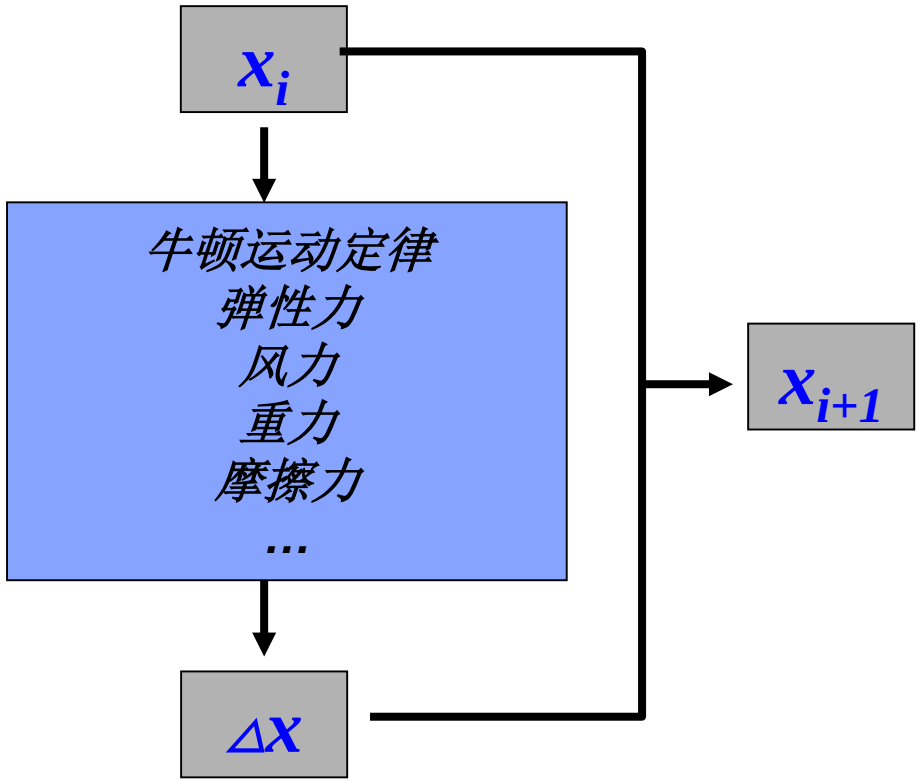
$$\mathbf{a} = \ddot{\mathbf{x}} = \frac{\mathbf{f}}{m}$$

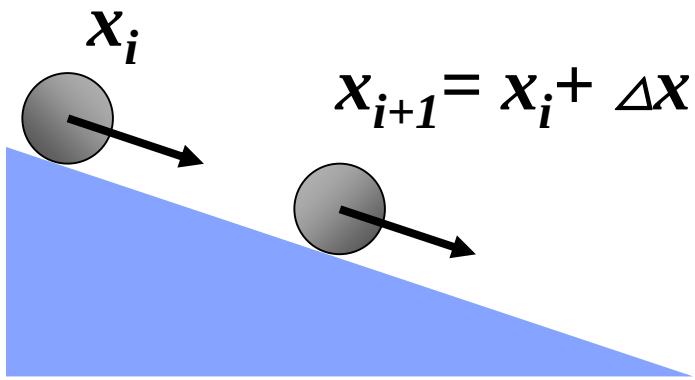
- 连续处理 **相对于** 离散时间模拟
  - 假设在单位时间步长内是常数，
    - 单位时间步长内对**加速度**积分，得到**速度**的变化，
    - 单位时间步长内对**速度**积分，得到物体**位置**的变化
  - 如何更新?
    - 欧拉积分法
    - Verlet法
    - …

粒子运动方程的求解

- 基于物理规律，生成一个系统的状态系列

$$\mathbf{a} = \ddot{\mathbf{x}} = \frac{\mathbf{f}}{m}$$





微分方程

微分方程

- 微分方程描述了一个未知函数与其导数之间的关系
- 解微分方程是为了找到满足该关系的函数
- 通常用数值求解方法!

常微分方程

- Ordinary differential equation (ODE)
  - 所有导数都关于一个独立变量，通常为时间t

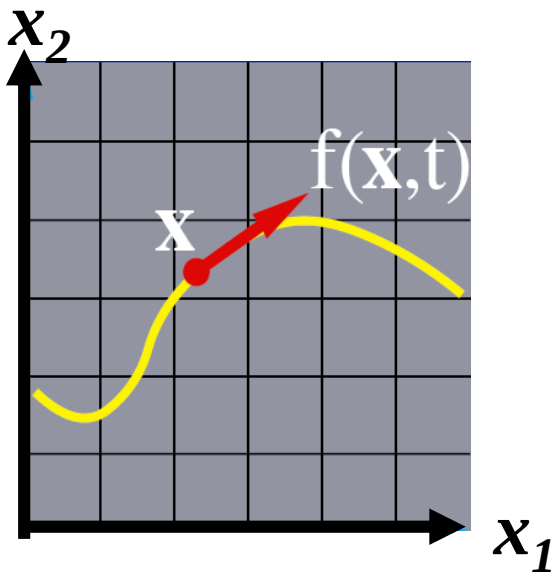
已知函数

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}(t)) = f(\mathbf{x}, t)$$

未知函数对时间的导数

$\mathbf{x}(t_0)$: 在 t_0时刻的状态

常微分方程解的可视化



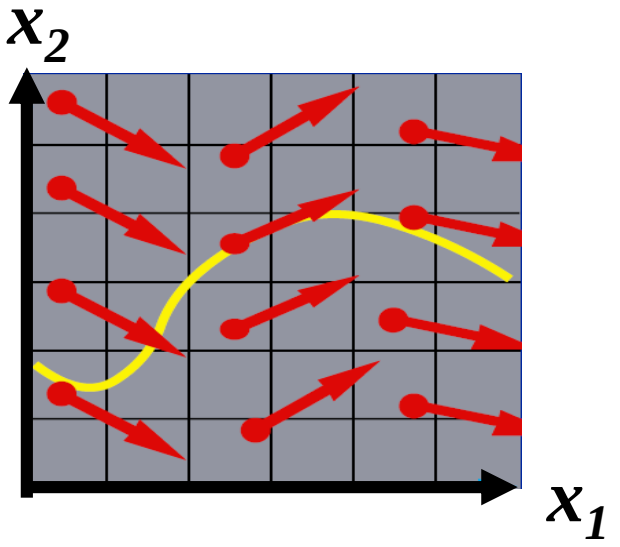
$$\dot{\mathbf{x}} = \frac{d\mathbf{x}}{dt} = f(\mathbf{x},t)$$

$\mathbf{x}(t)$：一个移动的点

$f(\mathbf{x},t)$：$\mathbf{x}$的速度

微分方程 $\dot{\mathbf{x}} = f(\mathbf{x},t)$ 的所有解定义了一个在平面 $(\mathbf{x},t)$ 上的向量场

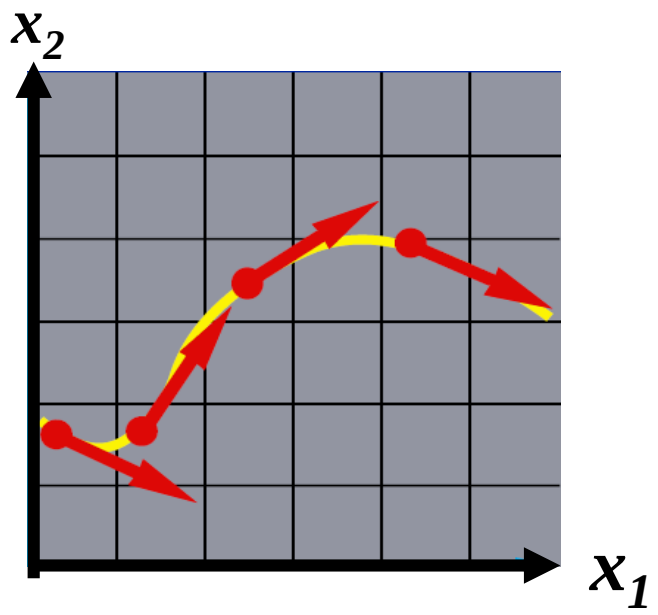
Think of this vector field as the sea, and the velocity of current at different places and time is defined by f(x,t)



初值问题

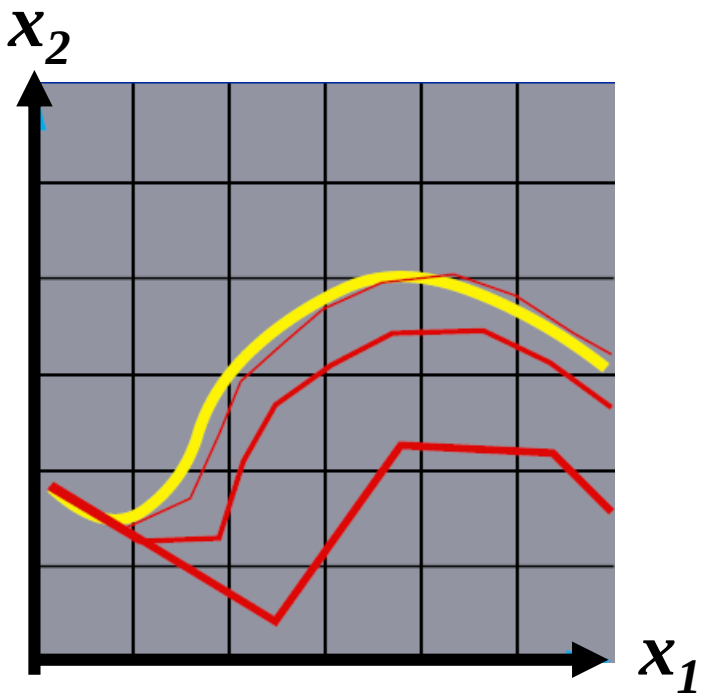
给定 $\dot{\mathbf{x}} = f(\mathbf{x}, t)$ 和 $\mathbf{x}_0 = \mathbf{x}(t_0)$, 求 $\mathbf{x}(t)$

- 对于一个特定的初值，其解是一条曲线
  - 给定初始点，顺着这条积分曲线



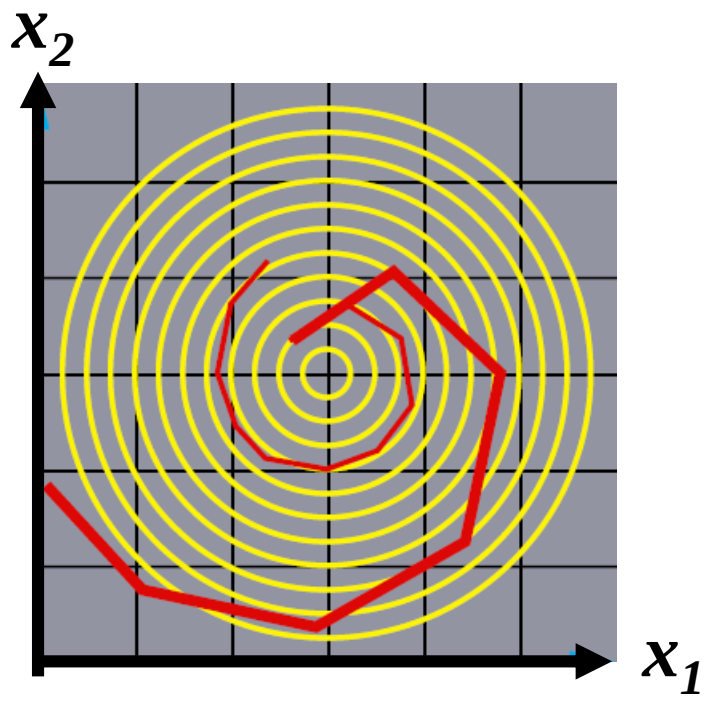
常微分方程解的数值解

- 与真实的积分曲线不同，数值解通常为一条多边形折线
- 步长越大，误差越大

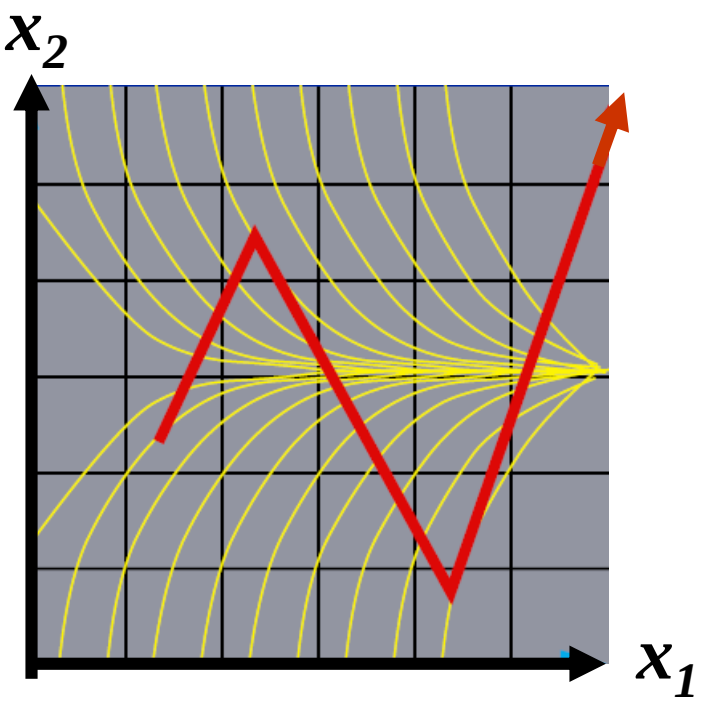


求解中可能产生的问题

- **问题I: 精度**(Inaccuracy)
  - 误差把$\mathbf{x}(t)$从一个圆变成了螺旋线!
  - 可能跳到其它圆



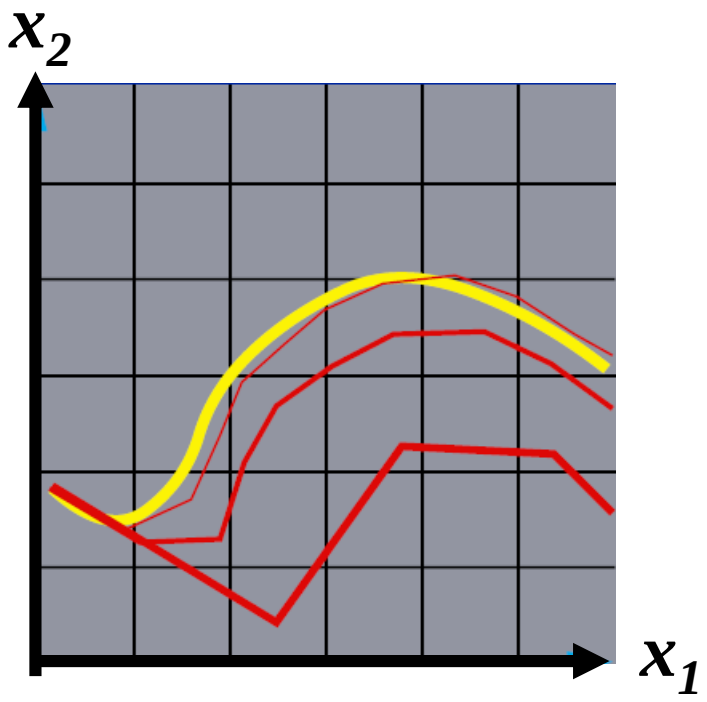
- **问题II: 不稳定**
  - 有可能发散!



求解：欧拉法

- 是最简单的数值求解方法
- 时间步长越大，误差越大

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h \times f(\mathbf{x}, t)$$



给定 $\dot{\mathbf{x}} = f(\mathbf{x}, t)$ 和 $\mathbf{x}_0 = \mathbf{x}(t_0)$, 求 $\mathbf{x}(t)$

通过一阶泰勒展开来求解微分方程:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + h\dot{\mathbf{x}}(t) + \frac{h^2}{2}\ddot{\mathbf{x}}(\xi)$$

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

$$\mathbf{x}_0 = \mathbf{x}(t_0)$$

$O(h^2)$: *二阶精度*

$$\mathbf{x}(t+h) \approx \mathbf{x}(t) + h \cdot f(\mathbf{x}, t)$$

$$\longrightarrow \quad \mathbf{x}_{n+1} = \mathbf{x}_n + h\dot{\mathbf{x}}_n$$

欧拉法的缺点

- 精度差，难以实用
- 效率低
  - 需要用很小的时间步长以避免发散

Verlet积分方法

- The Verlet algorithm uses the positions and accelerations at time *t*, and the positions from the previous step x(t-h), to calculate the new positions x(t+h):

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \mathbf{v}(t)h + \frac{1}{2}\mathbf{a}(t)h^2 + \ldots$$

$$+$$

$$\mathbf{x}(t-h) = \mathbf{x}(t) - \mathbf{v}(t)h + \frac{1}{2}\mathbf{a}(t)h^2 - \ldots$$

$$\Rightarrow \quad \mathbf{x}(t+h) = 2\mathbf{x}(t) - \mathbf{x}(t-h) + \mathbf{a}(t)h^2$$

$$\mathbf{a}(t) = \frac{\mathbf{f}(t)}{m}$$

- The velocities do not appear in the Verlet integration scheme: *velocities are not necessary for generating particle trajectories*.

Verlet积分方法

To obtain the **new velocities**, we can calculate them from the difference between the positions at two different times:

$$\mathbf{v}(t) = \left[\mathbf{x}(t+h) - \mathbf{x}(t-h)\right]/2h$$

or

$$\mathbf{v}(t+\frac{1}{2}h) = \left[\mathbf{x}(t+h) - \mathbf{x}(t)\right]/h$$

Advantages:

Implementation is straightforward.

Storage requirements are modest:

two sets of positions and one set of accelerations.

9N stored numbers

Disadvantages:

Positions are calculated by adding a small term of order h^2 to the difference of two much larger terms, which may lead to a loss in precision（精度）:

$$\mathbf{x}(t+h) = 2\mathbf{x}(t) - \mathbf{x}(t-h) + \mathbf{a}(t)h^2$$

Velocities always lag behind positions.

Poor stability for large h.

Leap-Frog(蛙跳) Verlet积分方法

- Several variations on the Verlet algorithm have been developed, including the *leap-frog algorithm*:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \mathbf{v}(t+\frac{1}{2}h)h$$

$$\mathbf{v}(t+\frac{1}{2}h) = \mathbf{v}(t-\frac{1}{2}h) + \mathbf{a}(t)h$$

- The velocities at time t can be calculated from:

$$\mathbf{v}(t) = \frac{1}{2}\left[\mathbf{v}(t+\frac{1}{2}h) + \mathbf{v}(t-\frac{1}{2}h)\right]$$

- The velocities leap-frog over the positions to give their values at t+h/2. The positions then leapfrog over the velocities to give their new values at t+h, then the velocities at t +3h/2, and so on.

Leap-Frog(蛙跳) Verlet积分方法

- Advantages over standard Verlet:
  - Explicitly includes velocity (needed for kinetic energy).
  - Does not require calculation of differences of large numbers, so more accurate.
  - Same memory requirements (9N) as Verlet.

- Disadvantages over standard Verlet:
  - Positions and velocities are not synchronized.
    - Cannot calculate kinetic energy (from velocities) and potential energy (from positions) at the same time.
    - Not a problem - can always estimate v(t) from v(t+h/2) and v(t-h/2) if you need it.

Almost always used over Verlet.

Velocity Verlet积分方法

- The *velocity Verlet method* gives positions, velocities, and **accelerations** at the same time without compromising precision:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \mathbf{v}(t)h + \frac{1}{2}\mathbf{a}(t)h^2$$

$$\mathbf{v}(t+h) = \mathbf{v}(t) + \frac{1}{2}\left[\mathbf{a}(t) + \mathbf{a}(t+h)\right]h$$

- Implemented in three stages since calculating velocities requires accelerations at both t and t+h.

Velocity Verlet积分方法

- Actual implementation trick:
  1. Velocities at t+1/2h are calculated using information at t.

$$\mathbf{v}(t+\frac{1}{2}h) = \mathbf{v}(t) + \mathbf{a}(t)\frac{1}{2}h$$

  2. With these terms in memory, calculate:

$$\mathbf{x}(t+h) = \mathbf{x}(t) + \mathbf{v}(t)h + \frac{1}{2}\mathbf{a}(t)h^2 = \mathbf{x}(t) + \mathbf{v}\left(t+\frac{1}{2}h\right)h$$

  3. Calculate **a**(t+h) from forces using positions at t+h, and from this and **v**(t), calculate **v**(t+h).

$$\mathbf{v}(t+h) = \mathbf{v}(t) + \frac{1}{2}\left[\mathbf{a}(t) + \mathbf{a}(t+h)\right]h$$

Implementation

Particle

```
struct Particle
{
  Vector3  m_pos;          // current position of the particle
  Vector3  m_prevPos;      // last position of the particle
  Vector3  m_velocity;     // direction and speed
  Vector3  m_acceleration; // acceleration

  float   m_energy;         // determines how long the particle is alive

  float   m_size;           // size of particle
  float   m_sizeDelta;      // amount to change the size over time

  float   m_weight;         // determines how gravity affects the particle
  float   m_weightDelta;    // change over time

  float   m_color[4];       // current color of the particle
  float   m_colorDelta[4];  // how the color changes with time
};
```

Particle System

```
class ParticleSystem{
public:
      ParticleSystem(int maxParticles, Vector3 origin);
       // abstract  functions
      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 *m_particleList;  // particles for this emitter
  int       m_maxParticles;  // maximum number of particles in total
  int       m_numParticles;  // indicies of all free particles
  Vector3   m_origin;        // center of the particle system
  float     m_accumulatedTime;  //track when was last particle emitted
  Vector3   m_force;         // force (gravity, wind, etc.) acting on the particle system
};
```

Particle System Source Codes

- Particle System on Github

  网址： https://github.com/mshooter/ParticleSystem

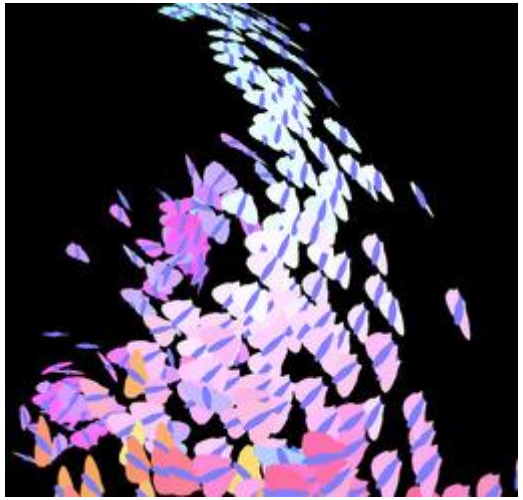
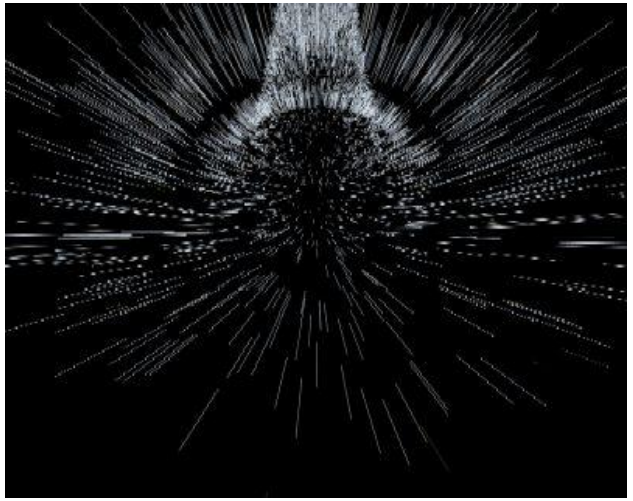
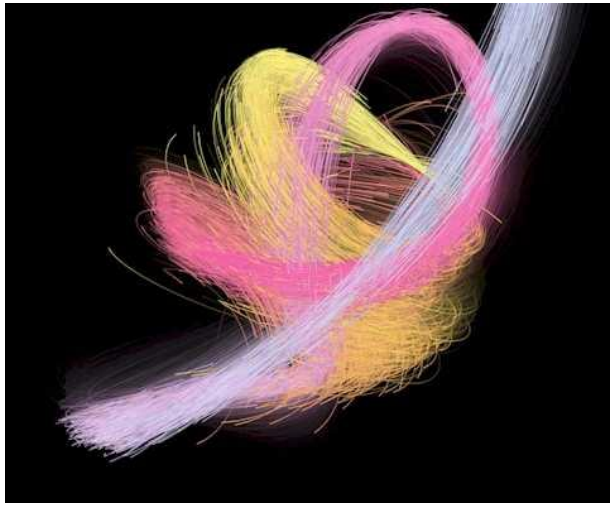
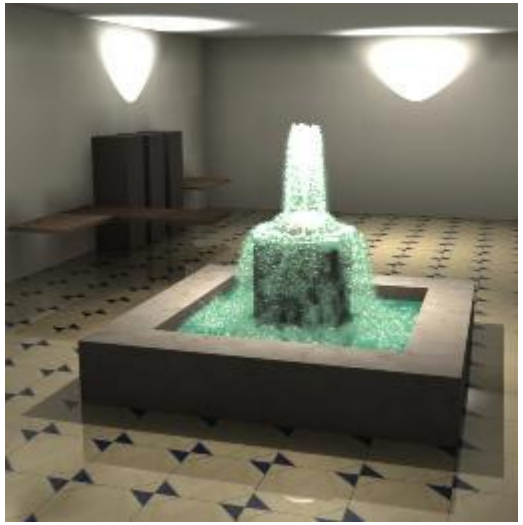
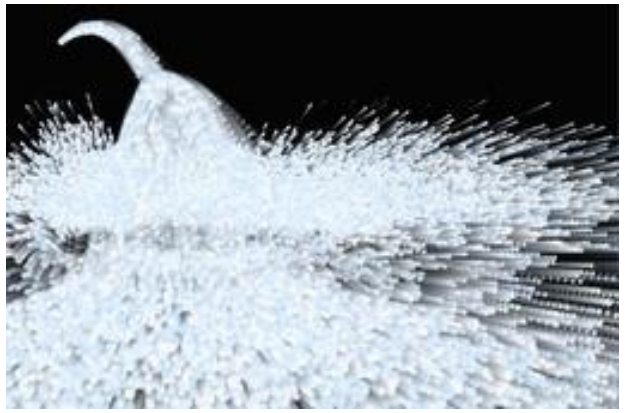
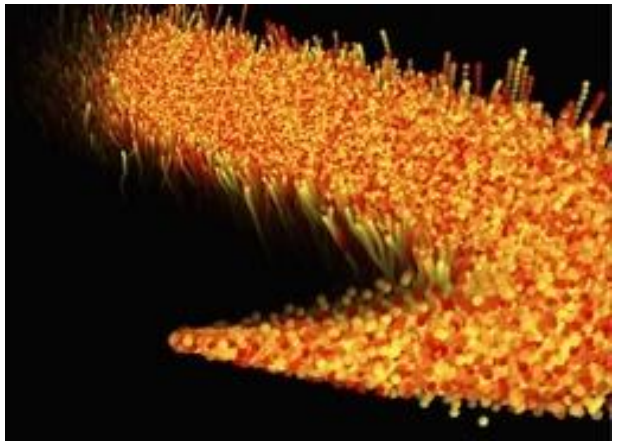
- A Basic Particles System

  网址：**http://www.codeproject.com/Articles/10003/A-basic-Particles-System**

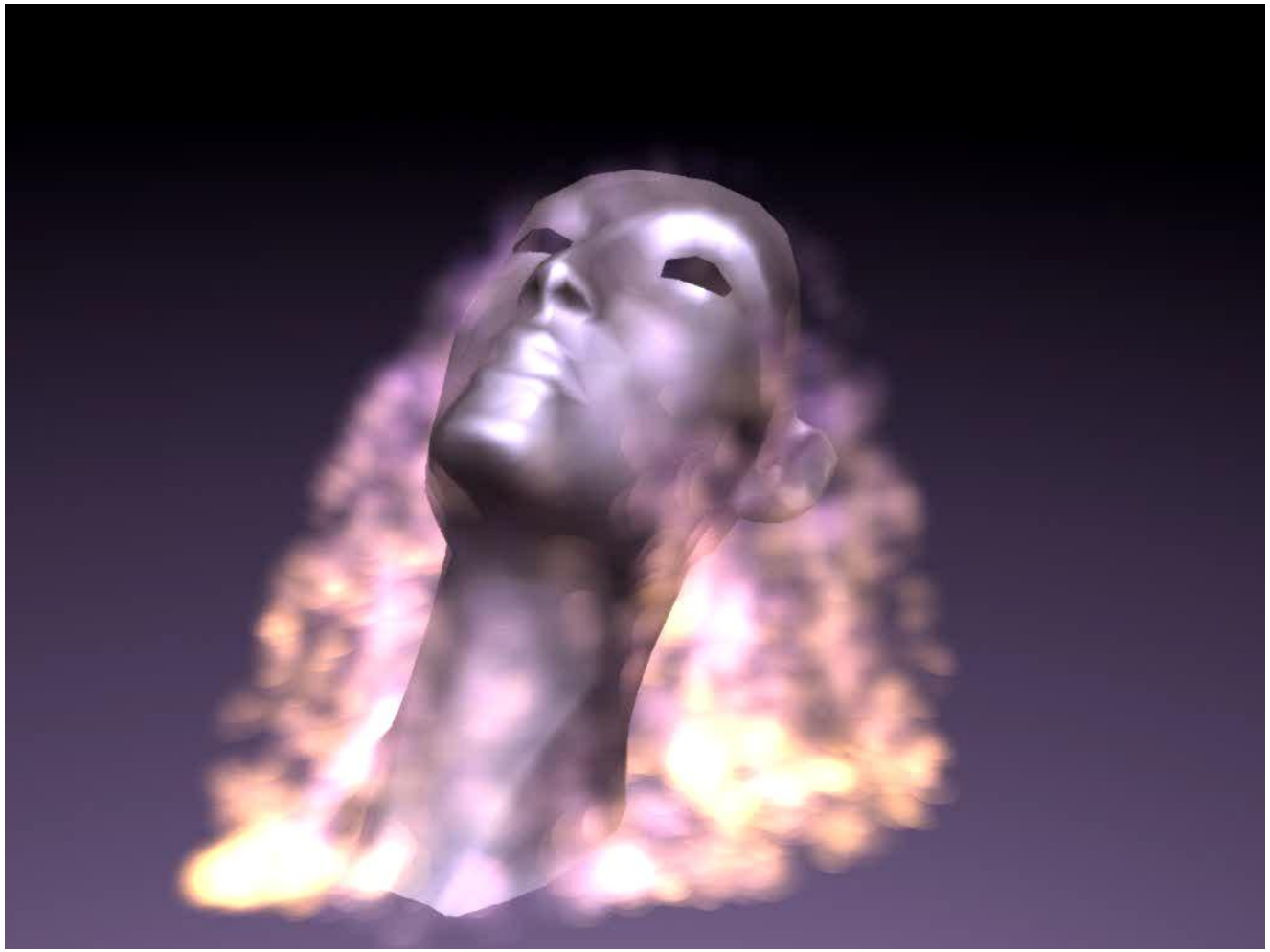
- Particle Systems

  网址： https://natureofcode.com/particles/

Screenshot

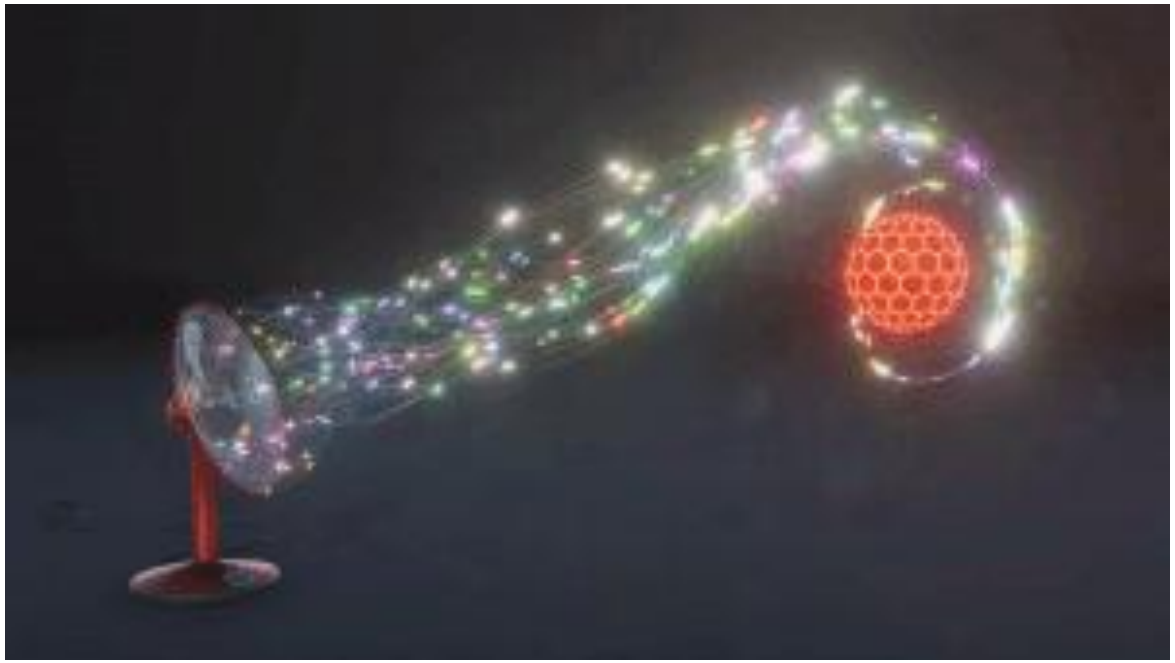







DEMO



总结

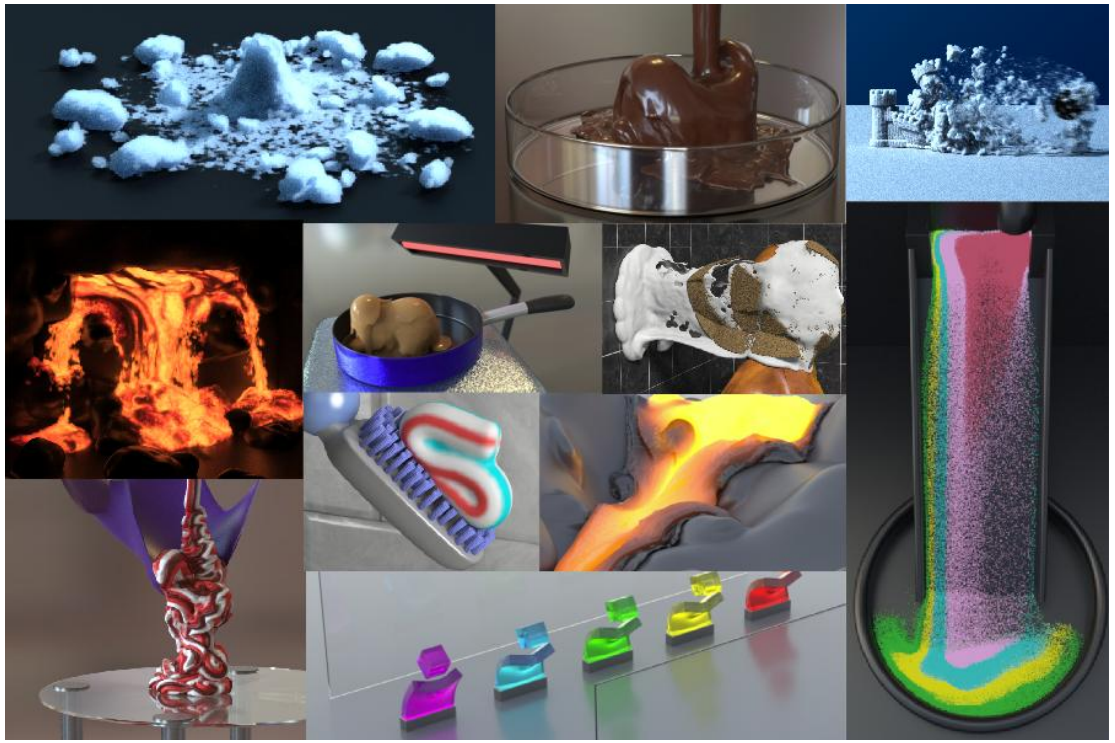
- 粒子系统应用非常广泛，可模拟火、爆炸、雨雪等很多很多动画特效；
- 是电影电视中常用的特效模拟方法；
- 对于高端应用，有专门的美工设计师；



进一步拓展

- The **Material Point Method** for Simulating Continuum Materials

物质点法采用质点来离散材料区域，用网格计算空间导数和求解动量方程，兼具拉格朗日和欧拉算法的优势， 非常适合模拟涉及材料特大变形和断裂破碎的问题。

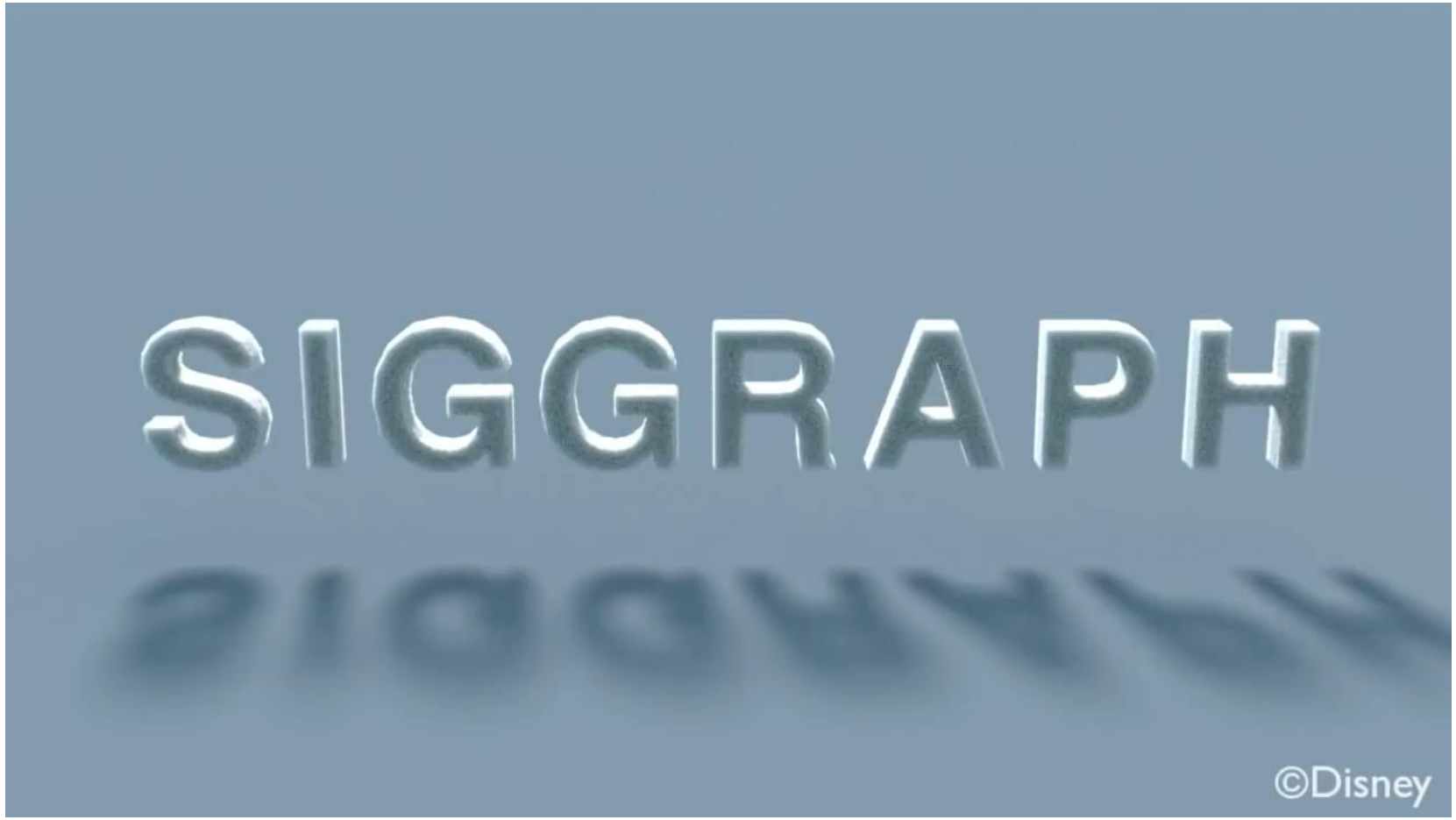


Particle (material point) p holds

- Position $\mathbf{x}p$
- Velocity $\mathbf{v}p$
- Mass mp
- Deformation gradient Fp （**变形梯度**，用于描述物体在发生形变时的局部应变和旋转情况）

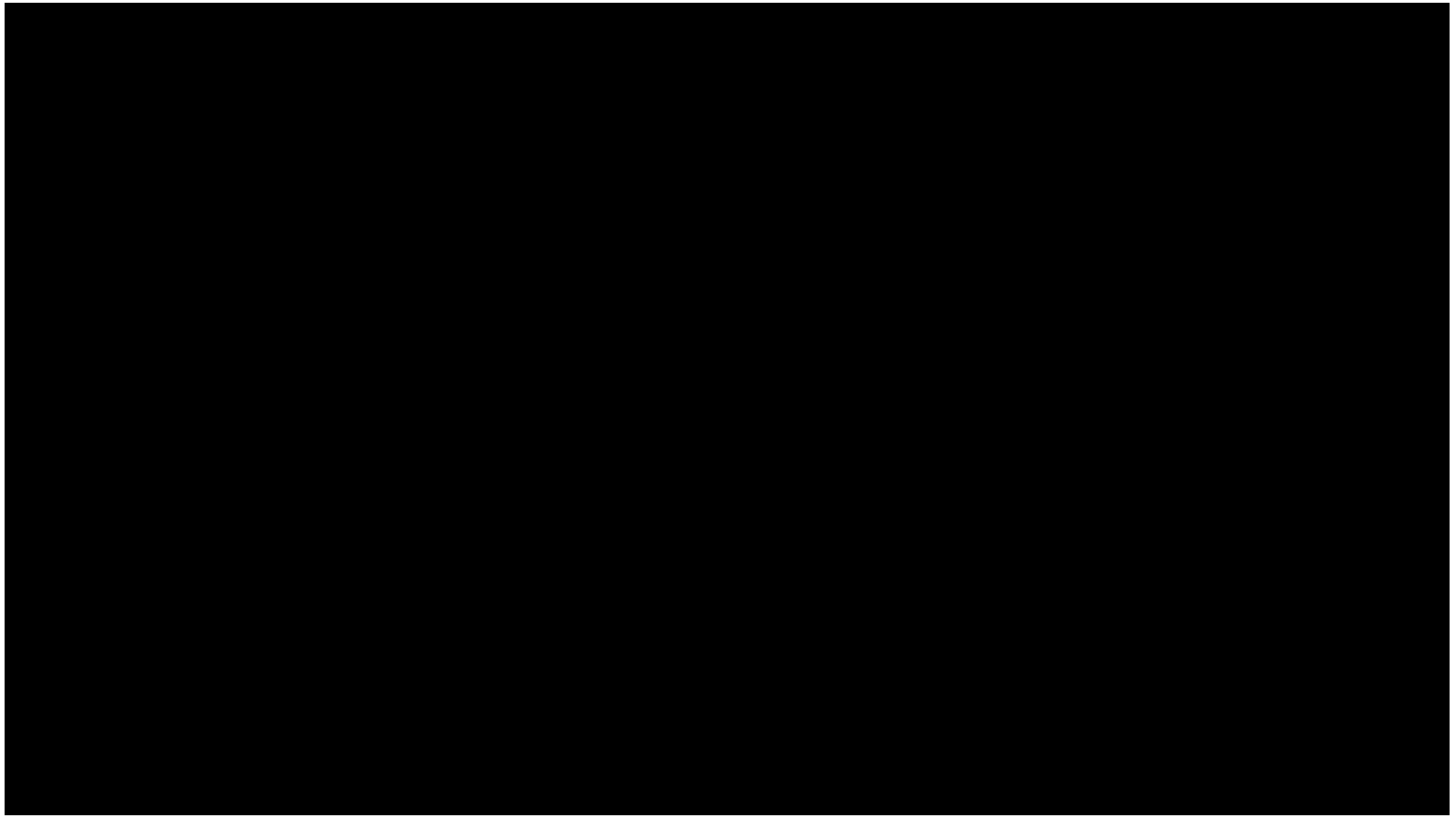
Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., & Selle, A. (2016). The material point method for simulating continuum materials. In ACM SIGGRAPH 2016 Courses (pp. 1-52).

进一步拓展



Stomakhin, A., Schroeder, C., Chai, L., Teran, J. and Selle, A., 2013. A material point method for snow simulation. ACM Transactions on Graphics (TOG), 32(4), pp.1-10.

DEMO



Demo (AE)

Demo

Making Snow with Particles - Blender 3.0 Tutorial

The End