Hair Simulation
(and Rendering)

Image from Final Fantasy
(Kai's hair)

Adrien Treuille

Overview

- Project
  - Solving Linear Systems
  - Questions About the Project
- Hair
  - Real Hair
  - Hair Dynamics
  - Hair Rendering
- Course Evaluations

Overview

- Project
  - **Solving Linear Systems**
  - Questions About the Project
- Hair
  - Real Hair
  - Hair Dynamics
  - Hair Rendering
- Course Evaluations

Solving Linear Systems

- **Want to solve system of the form:**

$$Ax = b$$

- ***A* is symmetric:**

$$A^T = A$$

- ***A* is positive-definite:**

$$x^T A x > 0 \quad \forall x$$

Interface

```
// Matrix class the solver will accept
class implicitMatrix
{
 public:
  virtual void matVecMult(double x[], double b[]) = 0;
};


// Solve Ax = b for a symmetric, positive definite matrix A
double ConjGrad(int n, implicitMatrix *A, double x[], double b[],
      double epsilon,   // how low should we go?
      int    *steps);
```

Implicit Matrix

```
// Matrix class the solver will accept
class implicitMatrix
{
 public:
  virtual void matVecMult(double x[], double b[]) = 0;
};
```

- `matVecMult`: **a method that performs matrix multiplication**
- `x`: **the input vector**
- `b`: **the output vector**

Implicit Matrix

```
// Solve Ax = b for a symmetric
// positive definite matrix A
double ConjGrad(int n, implicitMatrix *A,
  double x[], double b[],
  double epsilon,
  int    *steps);
```

- `n`: **number of dimensions**
- `implicitMatrix`: **matrix instance**
- `x`: **the *output* vector**
- `b`: **the *input* vector**
- `epsilon`: **how low should we go? (1.0^{-5})**
- `steps`: ***inputs* the max steps and *outputs the actual steps***

Example 1

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

```
#include "linearSolver.h"

class A1 : public implicitMatrix {
    public:
        virtual void matVecMult(double x[], double b[]) {
            b[0] = 2 * x[0];
            b[1] = 1 * x[1];
        }
};

int main(int argc, char **argv) {
    double x[2] = {0.0, 0.0};
    double b[2] = {1.0, 1.0};
    int steps = 100;

    implicitMatrix *a1 = new A1();
    double err = ConjGrad(2, a1, x, b, 1.0e-5, &steps);
    delete a1;

    printf("Solved in %i steps with error %f.\n", steps, err);
    printf("A1 * [%f %f]^T = [%f %f]^T.\n", x[0], x[1], b[0], b[1]);

    return 0;
}
```

```
linear-solver-example@CMU-274306$ ./solve1
Solved in 1 steps with error 0.000000.
A1 * [0.500000 1.000000]^T = [1.000000 1.000000]^T.
```

Example 2

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} x = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Example 2

```
#include "linearSolver.h"

class A2 : public implicitMatrix {
    public:
        virtual void matVecMult(double x[], double b[]) {
            b[0] = 2.0 * x[0] + 1.0 * x[1];
            b[1] = 1.0 * x[0] + 1.0 * x[1];
        }
};

int main(int argc, char **argv) {
    double x[2] = {0.0, 0.0};
    double b[2] = {3.0, 4.0};
    int steps = 100;

    implicitMatrix *a2 = new A2();
    double err = ConjGrad(2, a2, x, b, 1.0e-5, &steps);
    delete a2;

    printf("Solved in %i steps with error %f.\n", steps, err);
    printf("a2 * [%f %f]^T = [%f %f]^T.\n", x[0], x[1], b[0], b[1]);

    return 0;
}
```

Why implicitMatrix?

```
#include "linearSolver.h"

class A1 : public implicitMatrix {
    public:
        virtual void matVecMult(double x[], double b[]) {
            b[0] = 2 * x[0];
            b[1] = 1 * x[1];
        }
};
```

***O*(*n*)**

vs

```
#include "linearSolver.h"

class A2 : public implicitMatrix {
    public:
        virtual void matVecMult(double x[], double b[]) {
            b[0] = 2.0 * x[0] + 1.0 * x[1];
            b[1] = 1.0 * x[0] + 1.0 * x[1];
        }
};
```

$O(n^2)$

Example 3

$$\begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} x = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Not positive definite!

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -1$$

Example 3

- **What if *A* is not symmetric or not positive-definite?**

$$Ax = b$$

- **Then solve the *normal* equations:**

$$A^T A x = A^T b$$

Overview

- Project
  - Solving Linear Systems
  - **Questions About the Project**
- Hair
  - Real Hair
  - Hair Dynamics
  - Hair Rendering
- Course Evaluations

Questions?

Overview

- Project
  - Solving Linear Systems
  - Questions About the Project
- Hair
  - **Real Hair**
  - Hair Dynamics
  - Hair Rendering
- Course Evaluations

Real Hair: Curly

Short
curly hair

Real Hair: Straight

Long
smooth hair

Real Hair

- Typical human head has 150k-200k individual strands.
- Dynamics not well understood.
  - Subject still open to debate.

Overview

- Project
  - Solving Linear Systems
  - Questions About the Project
- Hair
  - Real Hair
  - **Hair Dynamics**
  - Hair Rendering
- Course Evaluations

Hair Dynamics

- Control Mesh
- Mass-Spring Systems
- Rigid Links
- Super Helices

Hair Dynamics

- **Control Mesh**
- Mass-Spring Systems
- Rigid Links
- Super Helices

Control Mesh

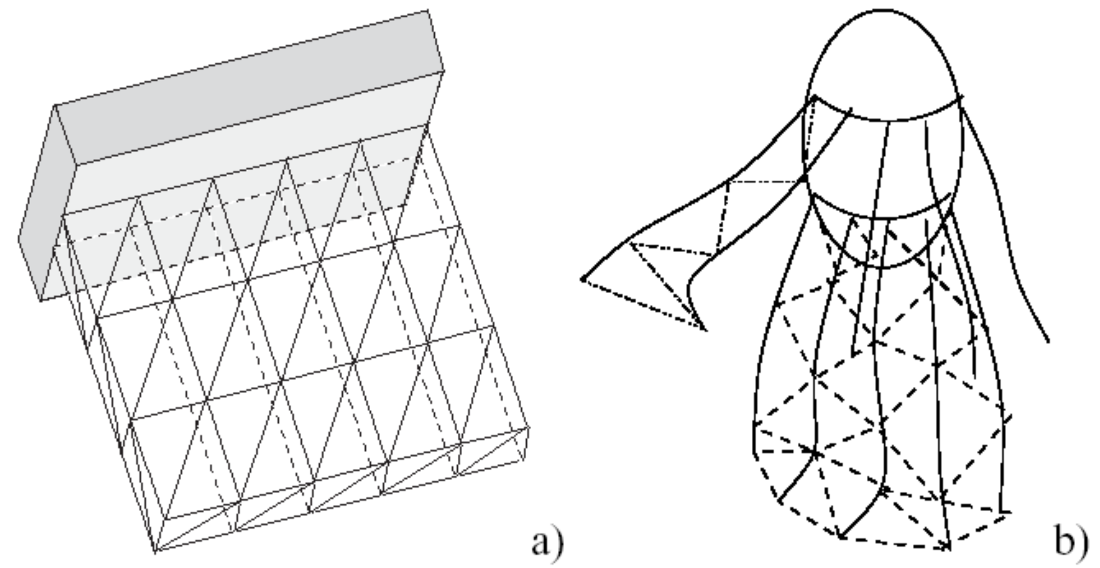


Figure 3: a) For a brush, triangle strips can be inserted between horizontally and vertically adjacent guide hairs. b) For a human scalp, triangle strips are inserted only between horizontally adjacent guide hairs

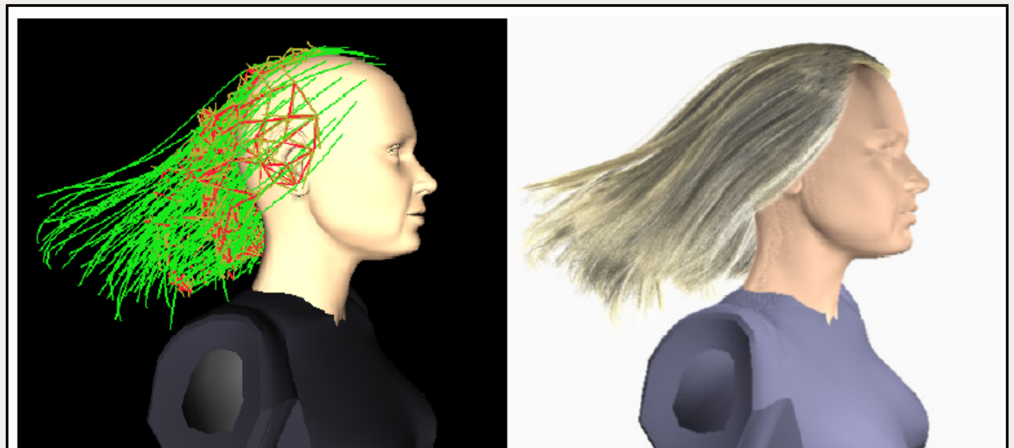


Figure 5: Left: a sparse hair model displayed with static links. Right: a rendered image of the interpolated dense model.

A Practical Model for Hair Mutual Interactions
Johnny T. Chang, Jingyi Jin, Yizhou Yu.
ACM SIGGRAPH Symp. on Computer Animation. pp. 73-80, 2002.

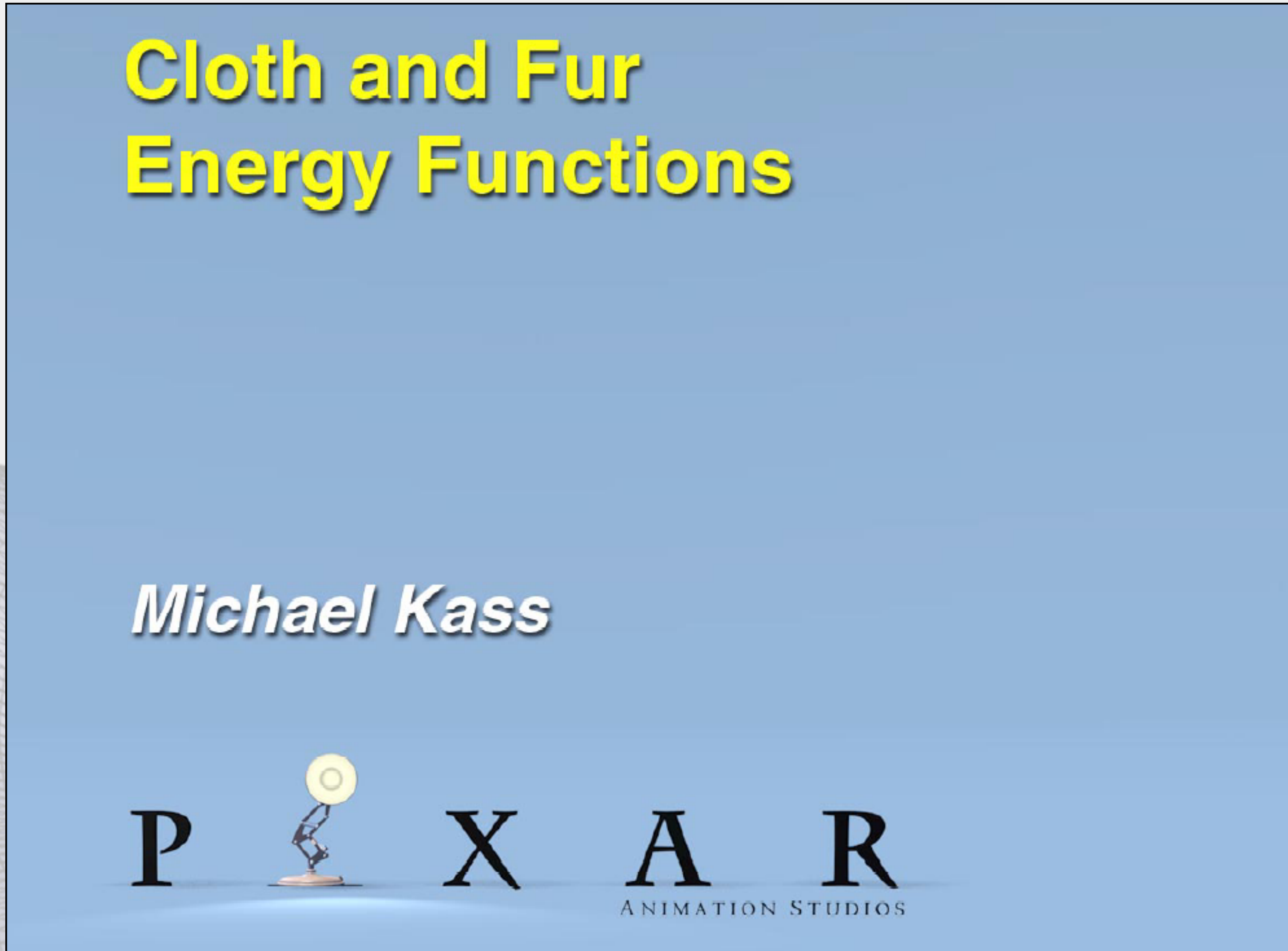
Control Mesh

ha_guide_hair.avi

Hair Dynamics

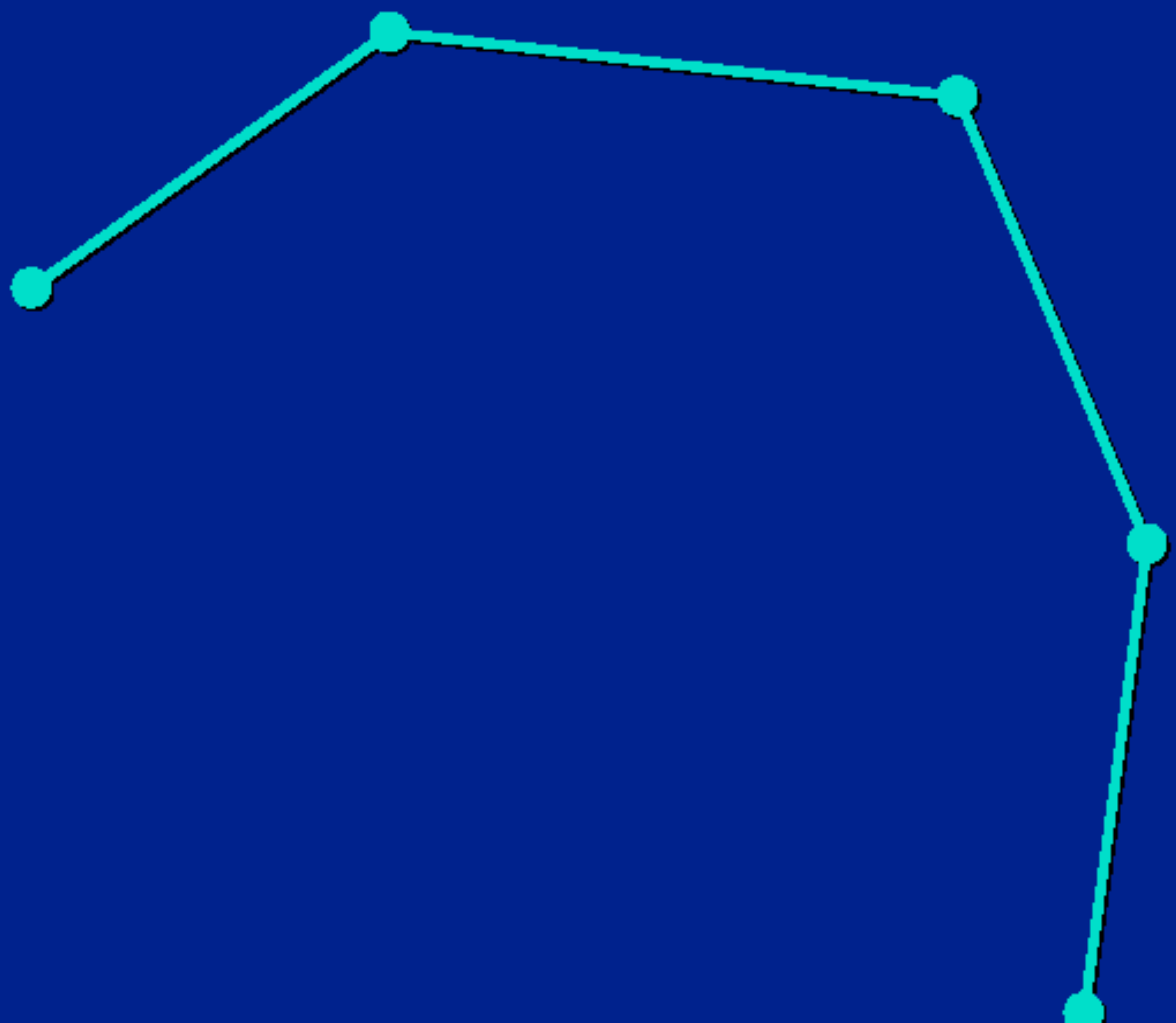
- Control Mesh
- **Mass-Spring Systems**
- Rigid Links
- Super Helices

Recall...


Cloth and Fur
Energy Functions
Michael Kass
PIXAR
ANIMATION STUDIOS

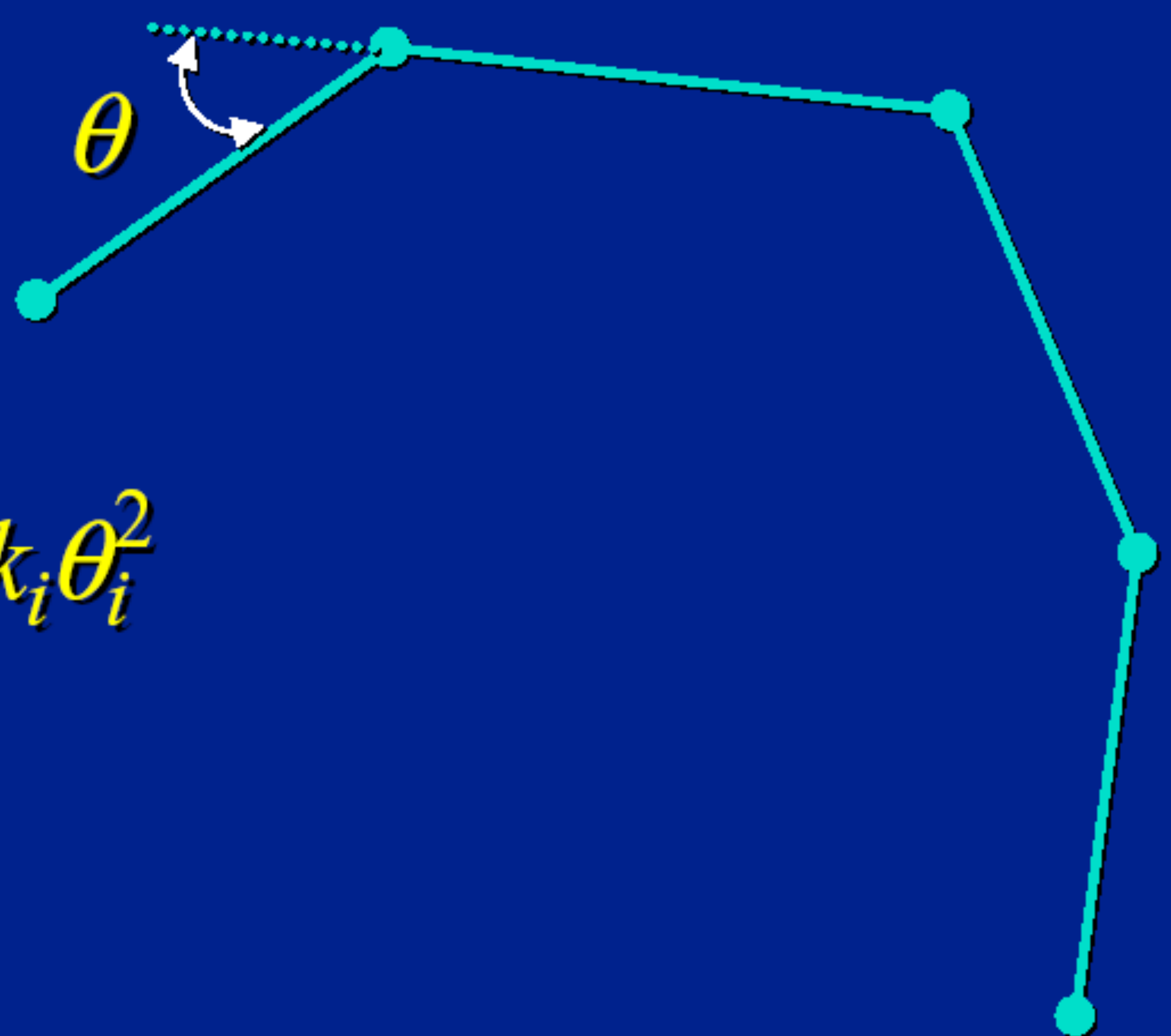
Hair Model

Limp hair: Just a set of springs.



Hair Model

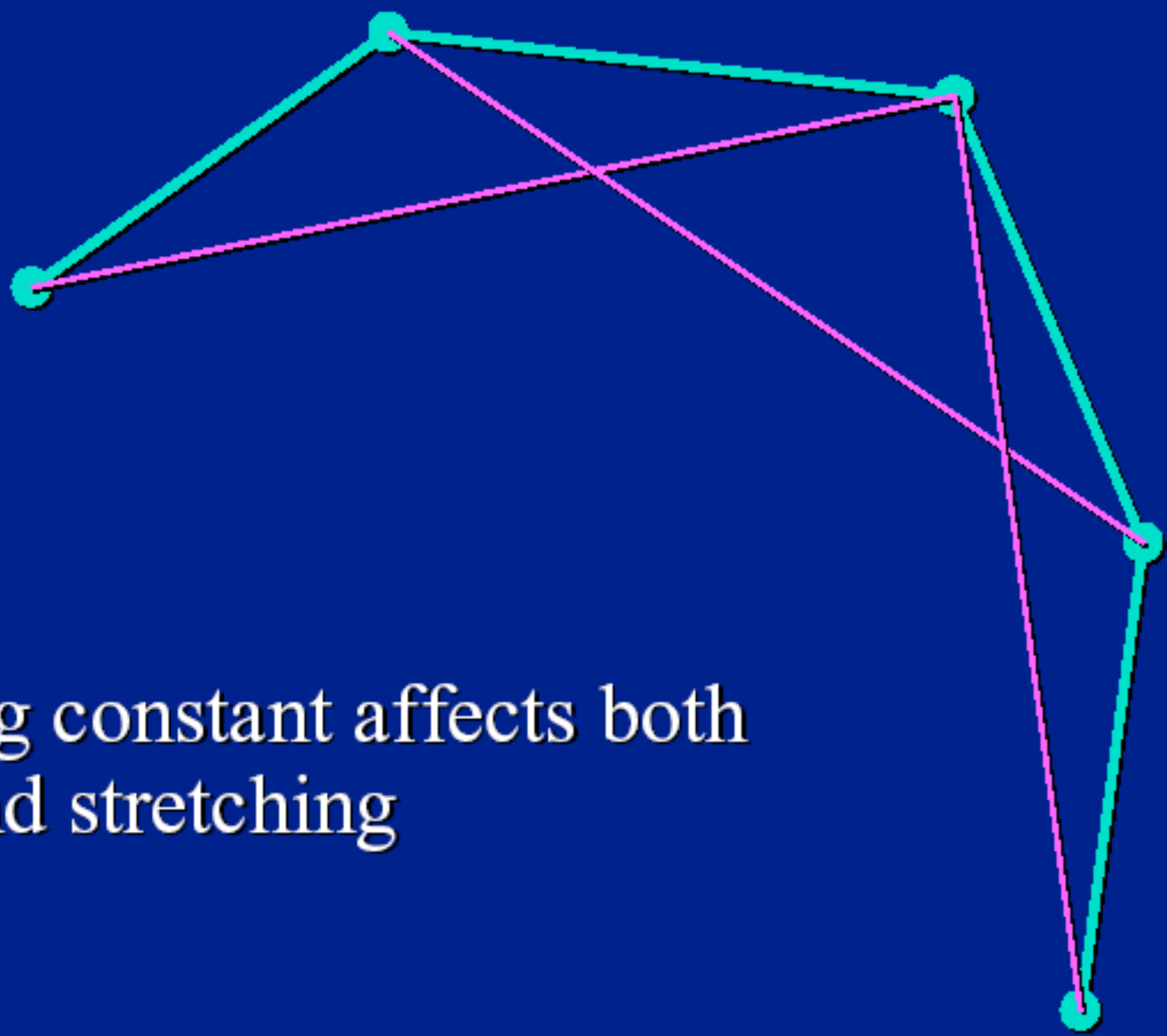
Add body: Angular Springs



$$E = \frac{1}{2}\sum_i k_i \theta_i^2$$

Hair Model

Alternative: More Linear Springs



Difficulty:

Each spring constant affects both bending and stretching

Discretization

Make sure energy independent of sampling.

Total energy: $E = \frac{1}{2} k \sum (l - l_{\text{rest}})^2$

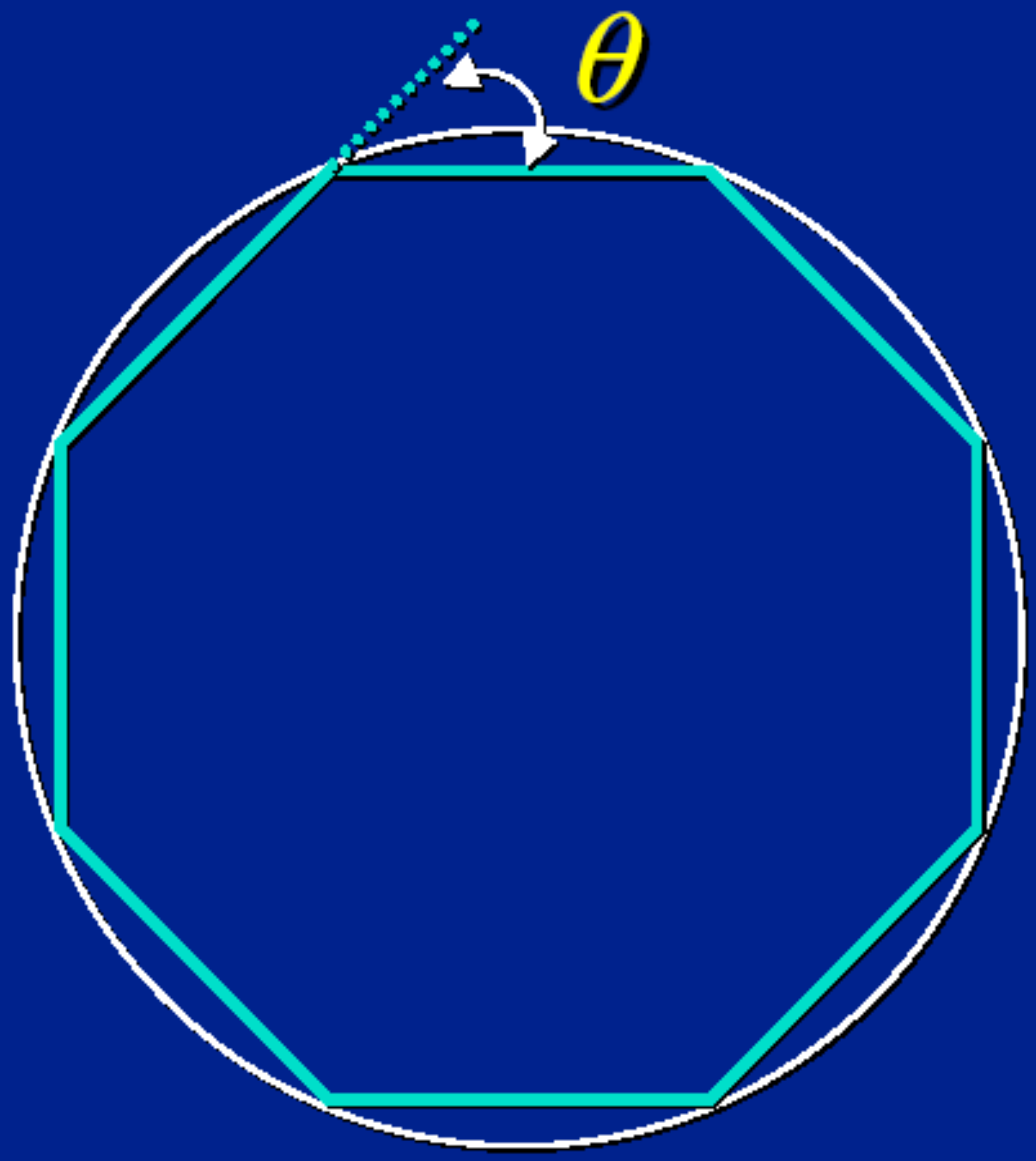
Stretch 100%: $E = \frac{1}{2} nk \left(\frac{L}{n} \right)^2$

Constant energy implies:

$k \propto n$ or $k_i \propto \frac{1}{l_i}$

Note: High sampling --> stiffness

Discretization



Consider a discretized circle.

$$E = \frac{1}{2} k \sum \theta^2$$

Again, constant energy implies:

$k \propto n$ or $k_i \propto \frac{1}{l_i}$

Disadvantages

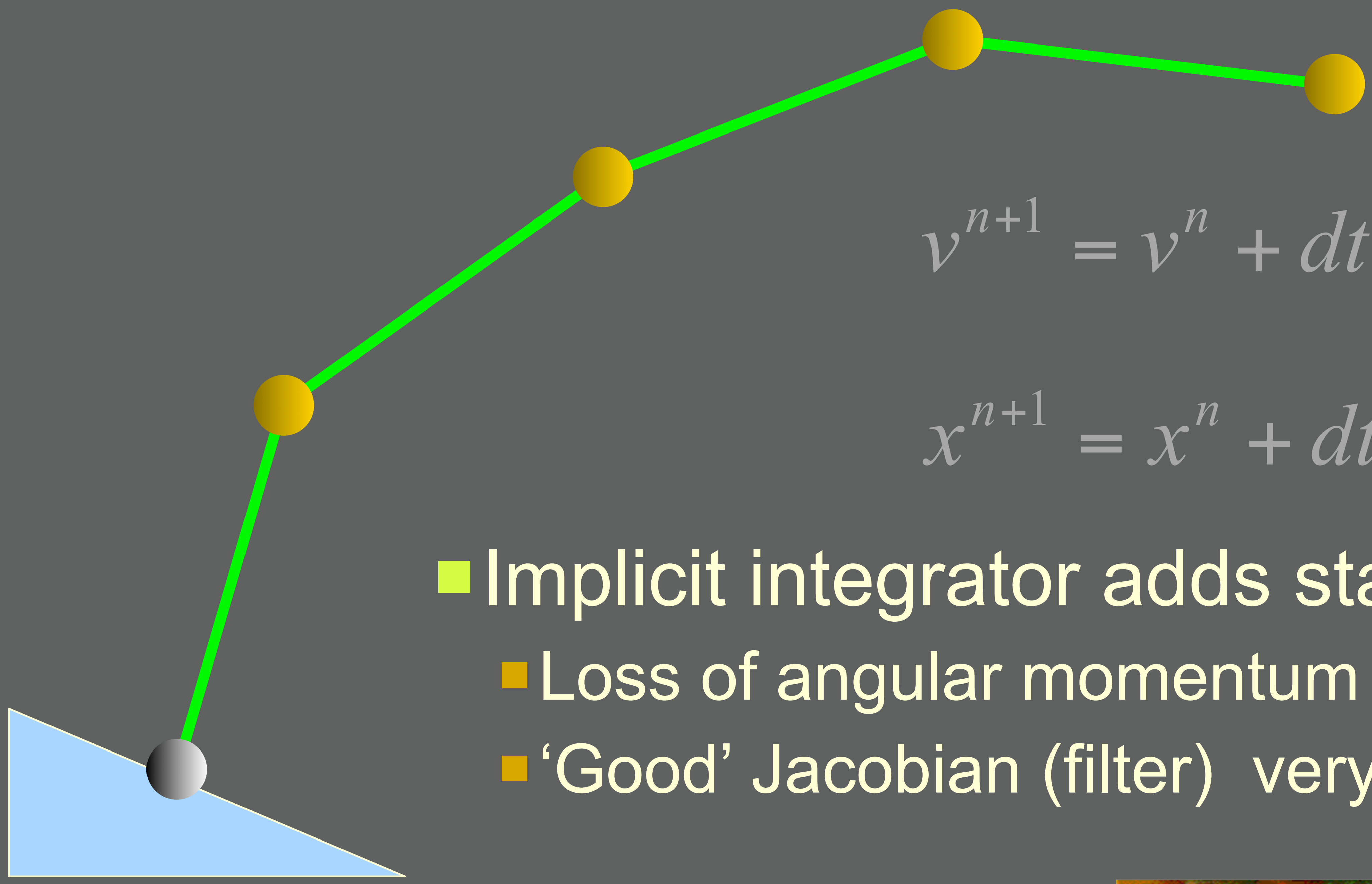
- Torsional Rigidity
- Non-stretching of the strands

Hair simulation in Rhythm and Hues - The Chronicles of Narnia

Tae-Yong Kim
Rhythm and Hues Studios

Rhythm + Hues Studios

$k = \infty$ → implicit integration?

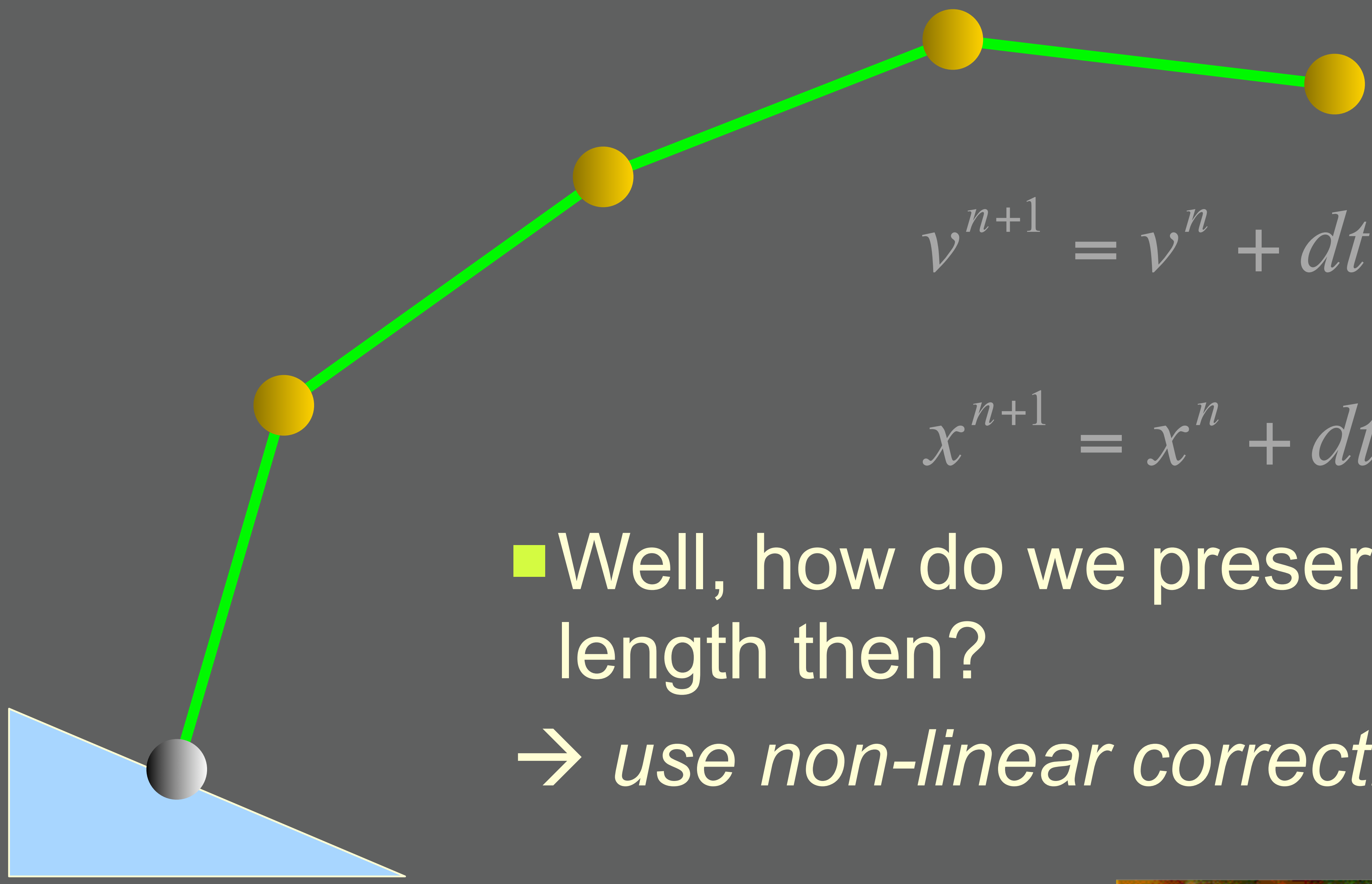


$$v^{n+1} = v^n + dt\frac{F^{n+1}}{m}$$

$$x^{n+1} = x^n + dt\,v^{n+1}$$

- Implicit integrator adds stability
  - Loss of angular momentum
  - ‘Good’ Jacobian (filter) very important

k Is infinity!



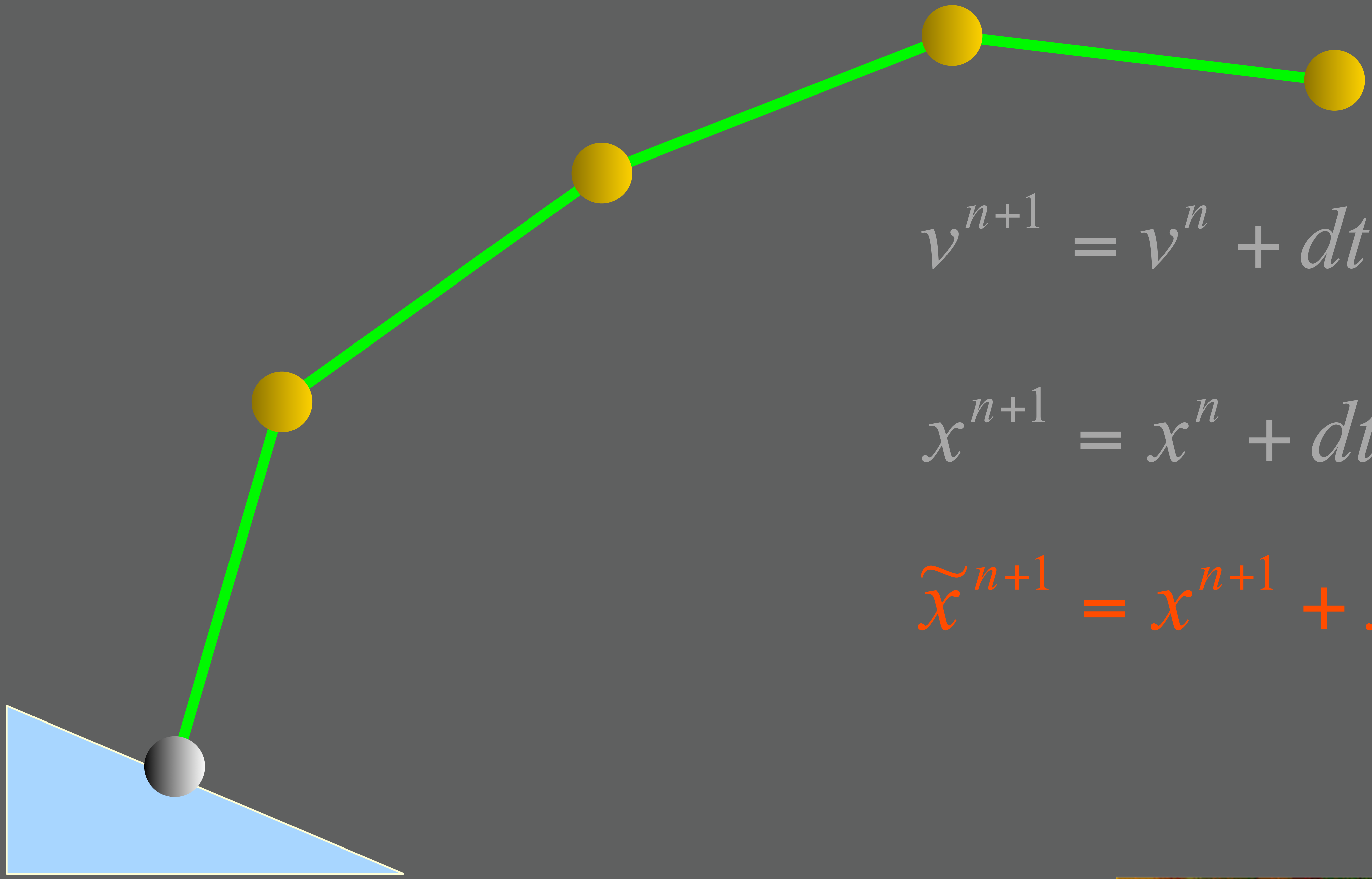
$$v^{n+1} = v^n + dt\frac{F}{m}$$

$$x^{n+1} = x^n + dtv^{n+1}$$

- Well, how do we preserve length then?

→ *use non-linear correction*

non-linear post correction

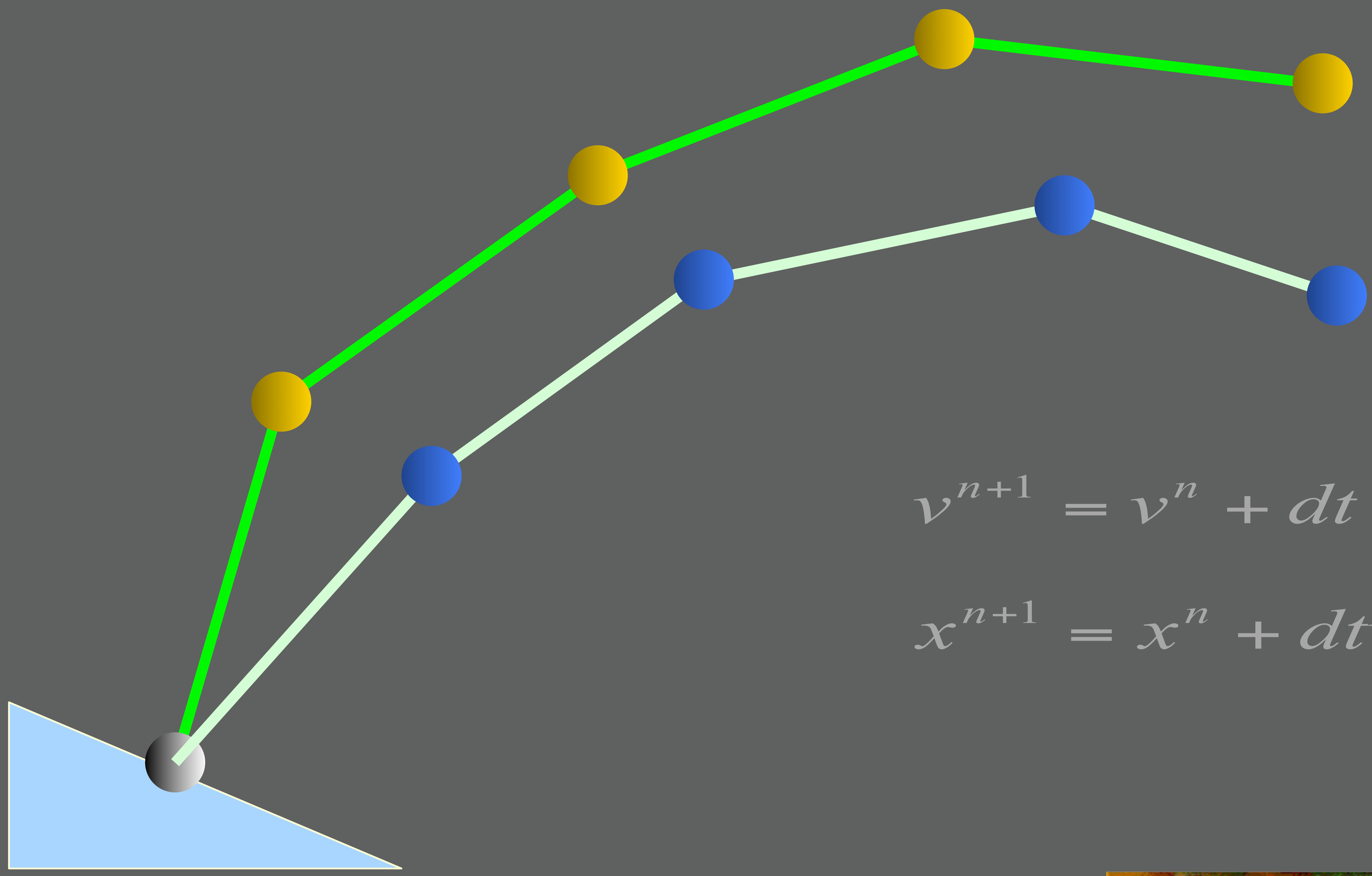


$$v^{n+1} = v^n + dt\frac{F}{m}$$

$$x^{n+1} = x^n + dtv^{n+1}$$

$$\tilde{x}^{n+1} = x^{n+1} + x^{corr}$$

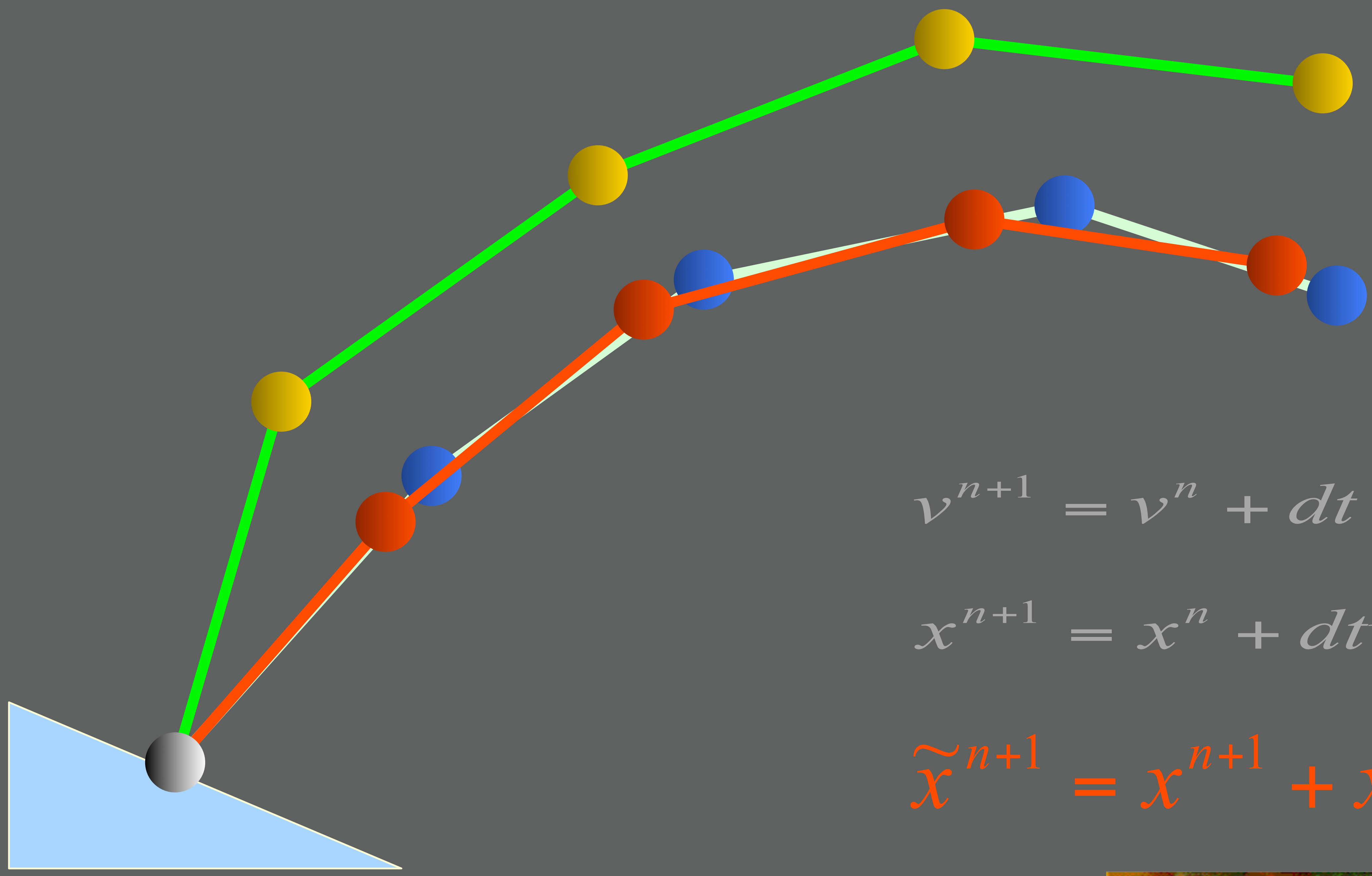
non-linear post correction



$$v^{n+1} = v^n + dt\frac{F}{m}$$

$$x^{n+1} = x^n + dt v^{n+1}$$

non-linear post correction

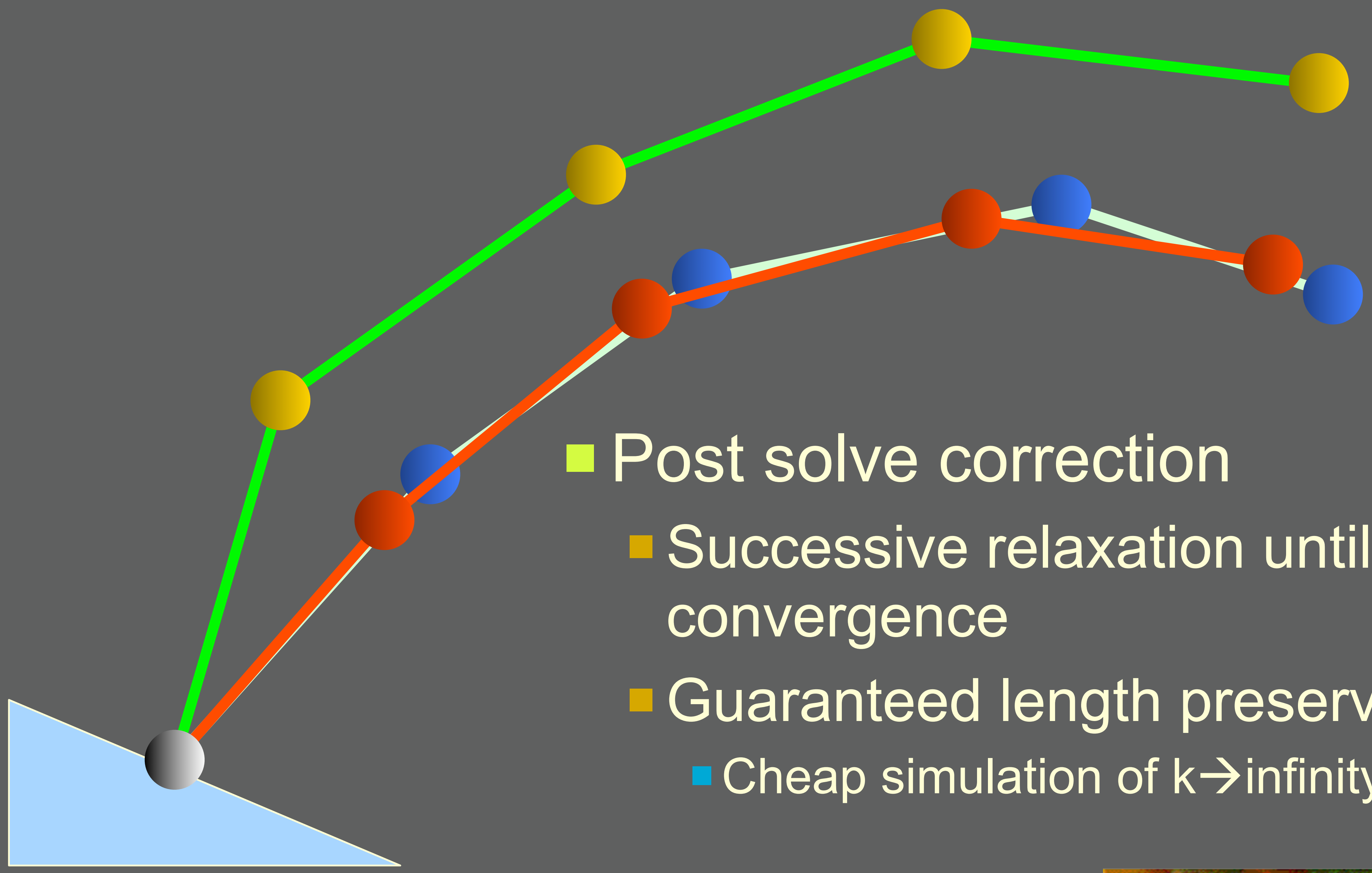


$$v^{n+1} = v^n + dt\frac{F}{m}$$

$$x^{n+1} = x^n + dt v^{n+1}$$

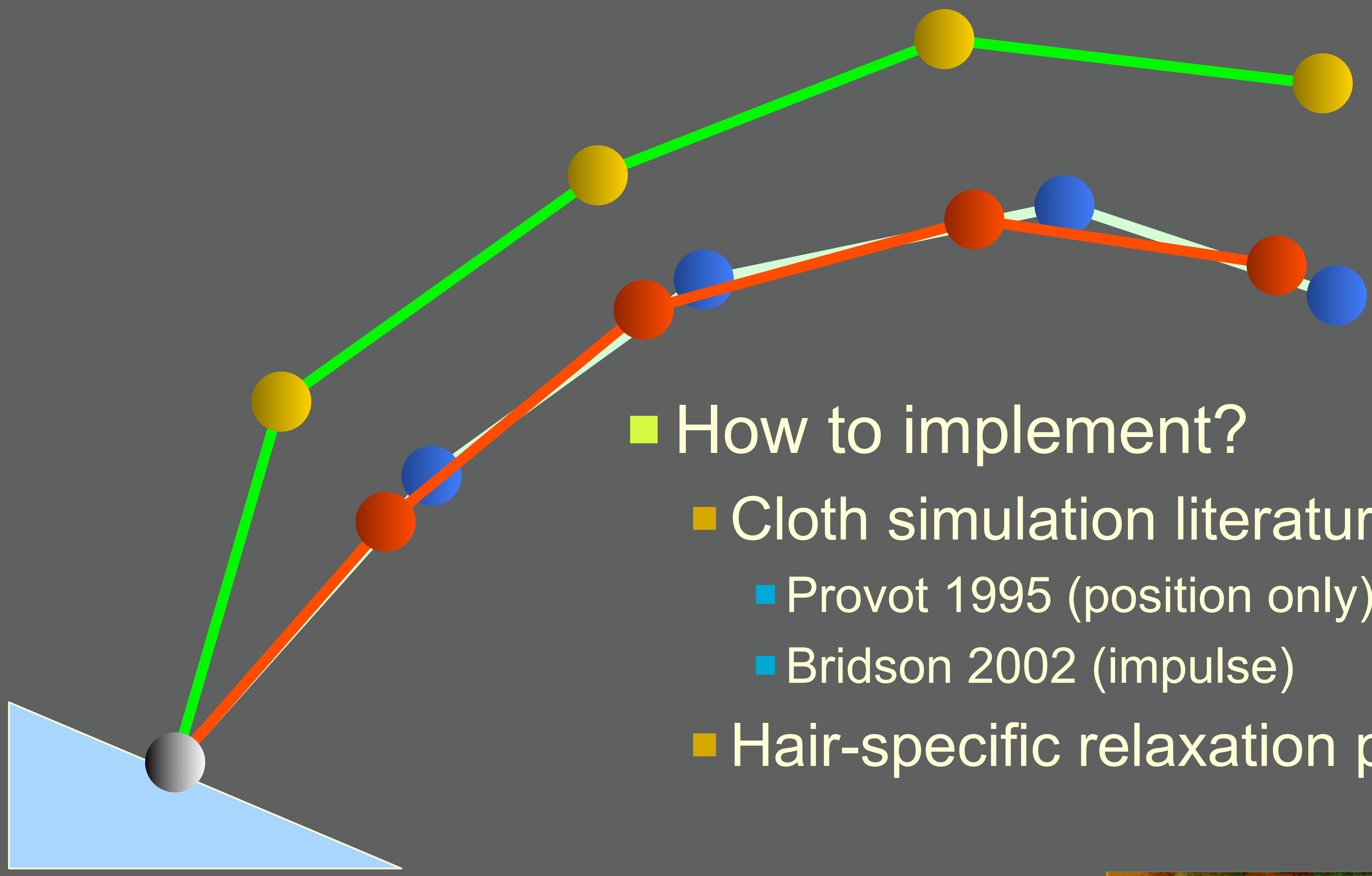
$$\tilde{x}^{n+1} = x^{n+1} + x^{corr}$$

non-linear post correction



- Post solve correction
  - Successive relaxation until convergence
  - Guaranteed length preservation
    - Cheap simulation of k→infinity

non-linear post correction



- How to implement?
  - Cloth simulation literatures
    - Provot 1995 (position only)
    - Bridson 2002 (impulse)
  - Hair-specific relaxation possible

Predictor-corrector scheme

- Implicit Filter (Predictor)

$$v^{n+\frac{1}{2}} = v^{n} + \frac{dt}{2}\frac{F^{n+1}}{m}$$

$$x^{n+1} = x^{n} + dt\,v^{n+\frac{1}{2}}$$

- Sharpener (Corrector)

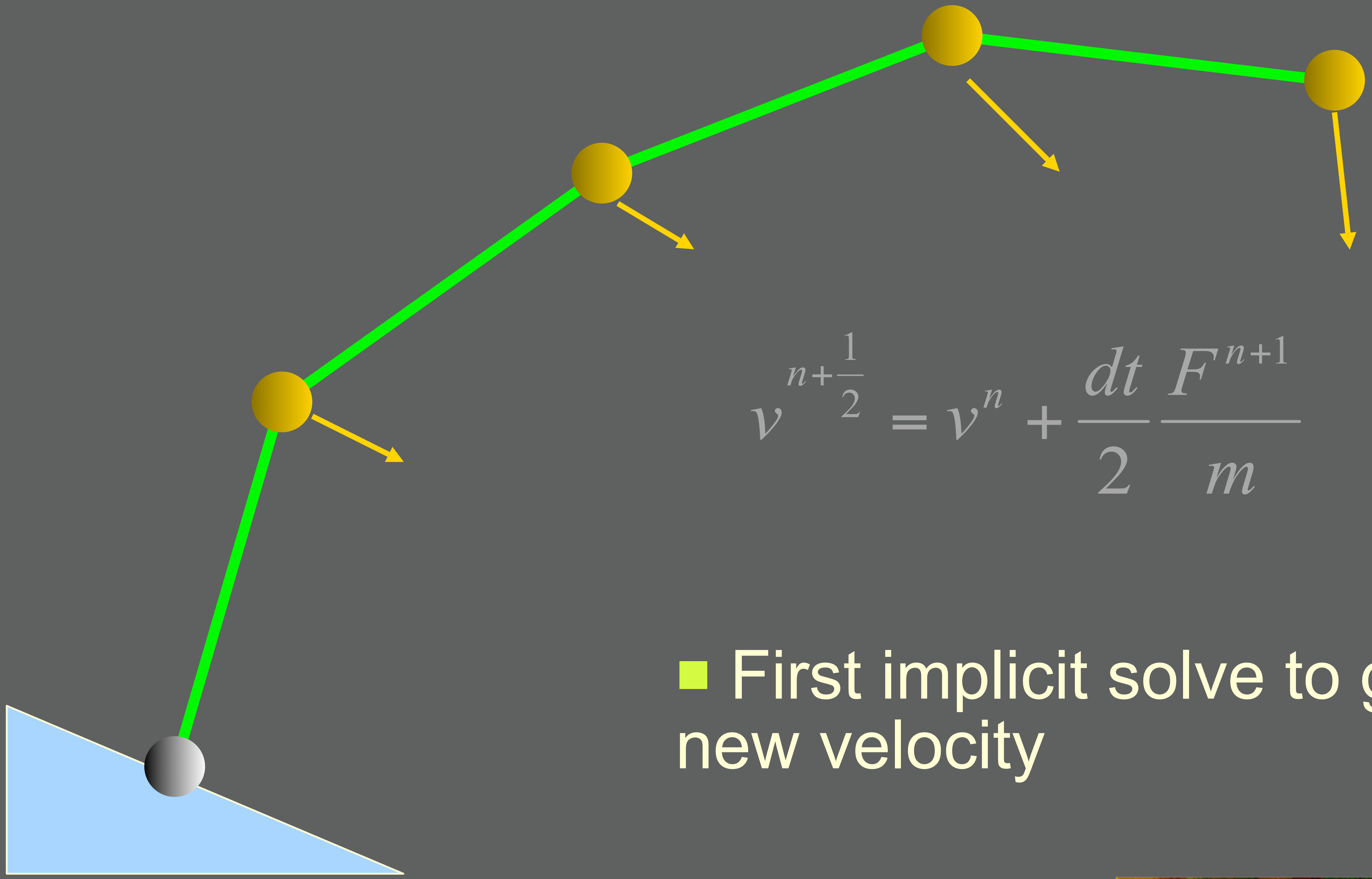
$$\tilde{x}^{n+1} = x^{n+1} + x^{corr}$$

$$\tilde{v}^{n+\frac{1}{2}} = v^{n+\frac{1}{2}} + \frac{x^{corr}}{dt}$$

- Implicit Filter (Predictor)

$$v^{n+1} = \tilde{v}^{n+\frac{1}{2}} + \frac{dt}{2}\frac{F^{n+1}}{m}$$

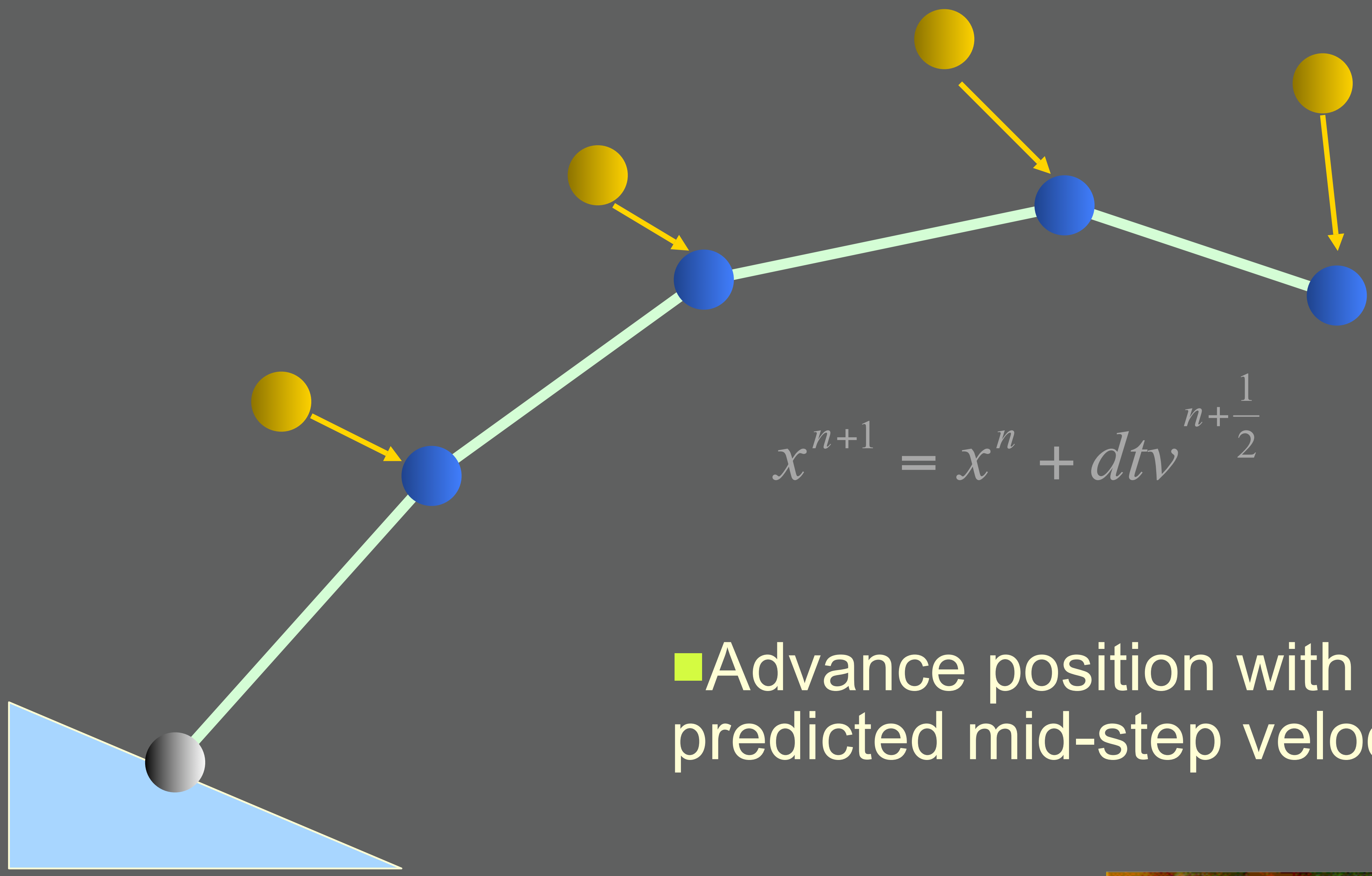
1.First pass-implicit integration



$$v^{n+\frac{1}{2}} = v^n + \frac{dt}{2}\frac{F^{n+1}}{m}$$

- First implicit solve to get new velocity

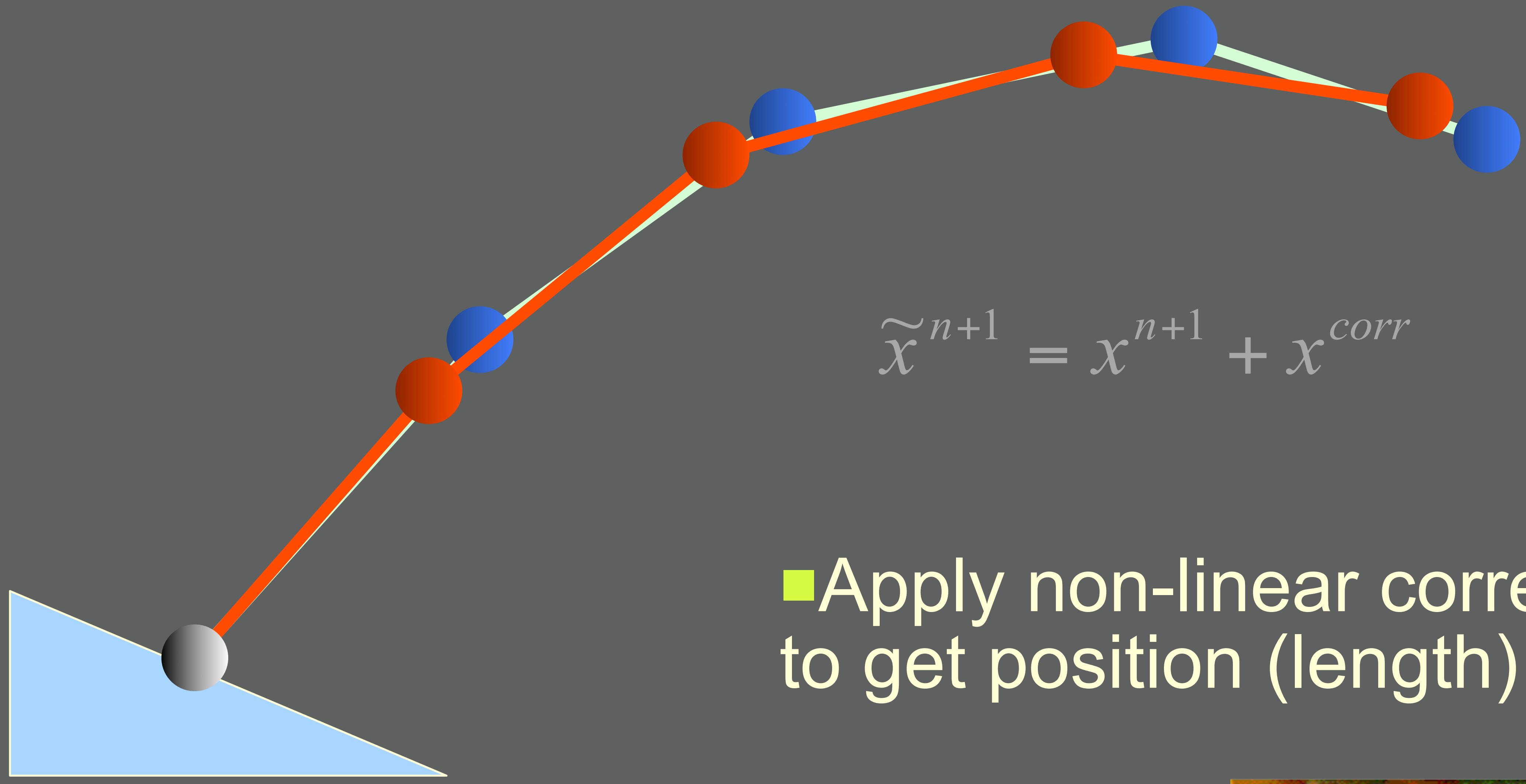
2.First pass-implicit integration



$$x^{n+1} = x^n + dt v^{n+\frac{1}{2}}$$

- Advance position with the predicted mid-step velocity

3.Non-linear Correction

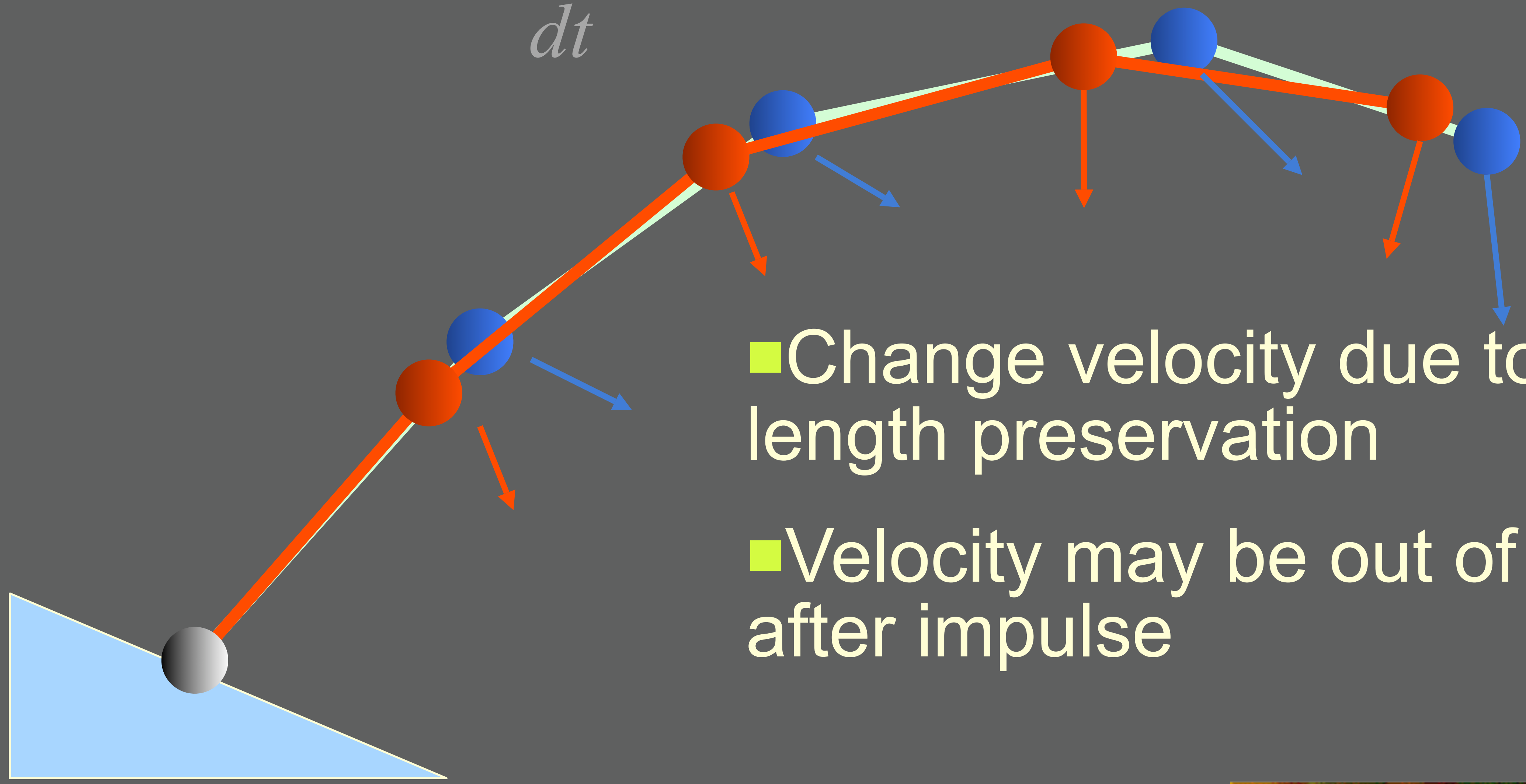


$$\tilde{x}^{n+1} = x^{n+1} + x^{corr}$$

- Apply non-linear corrector to get position (length) right

4.Impulse

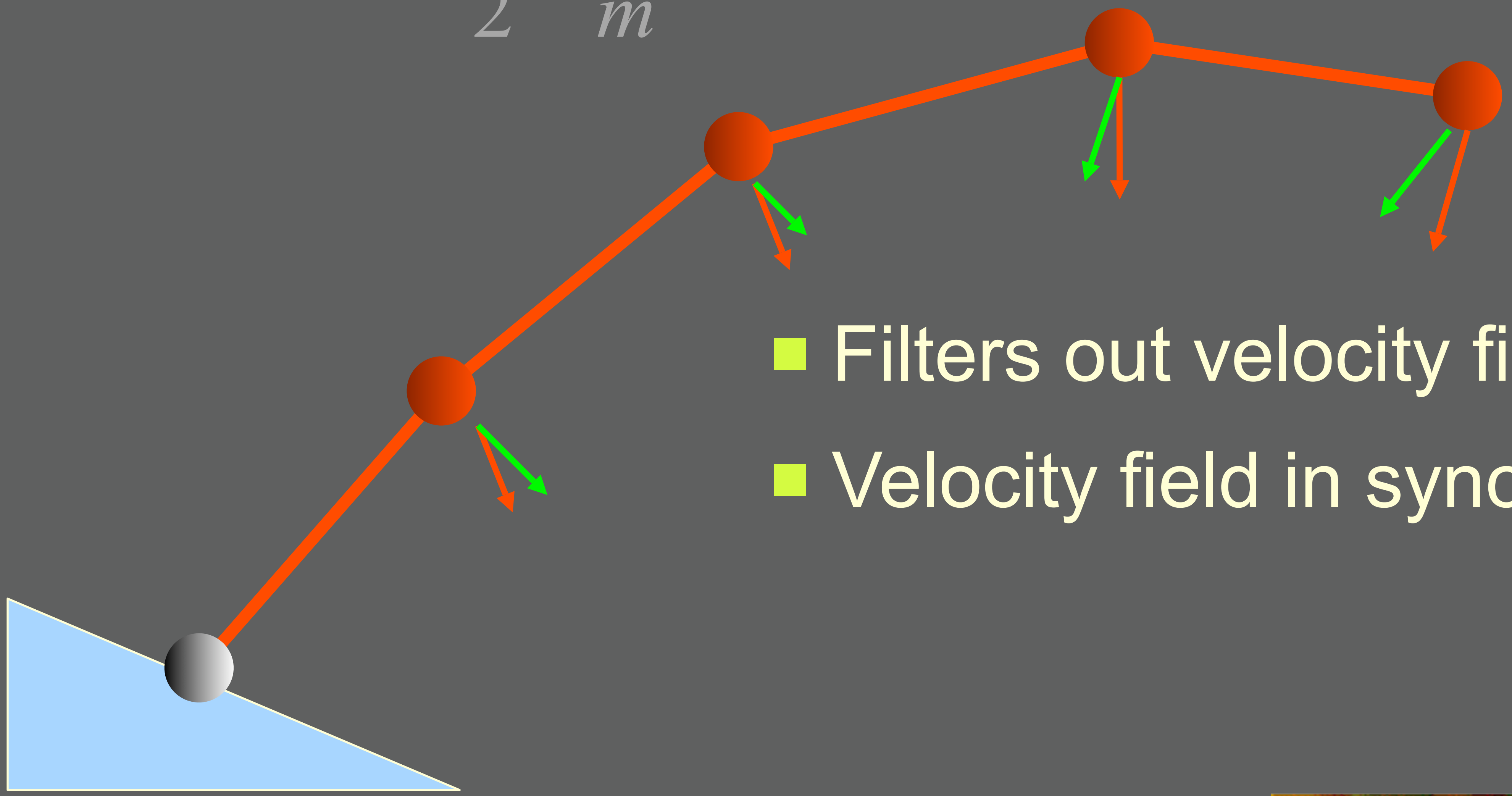
$$\tilde{v}^{n+\frac{1}{2}} = v^{n+\frac{1}{2}} + \frac{x^{corr}}{dt}$$



- Change velocity due to length preservation
- Velocity may be out of sync after impulse

5.Second implicit integration

$$v^{n+1} = \tilde{v}^{n+\frac{1}{2}} + \frac{dt}{2}\frac{F^{n+1}}{m}$$

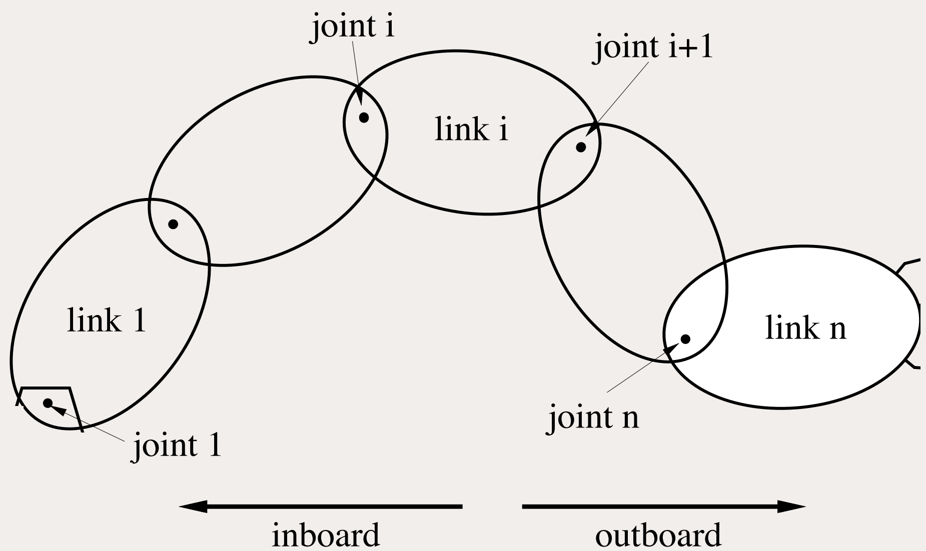


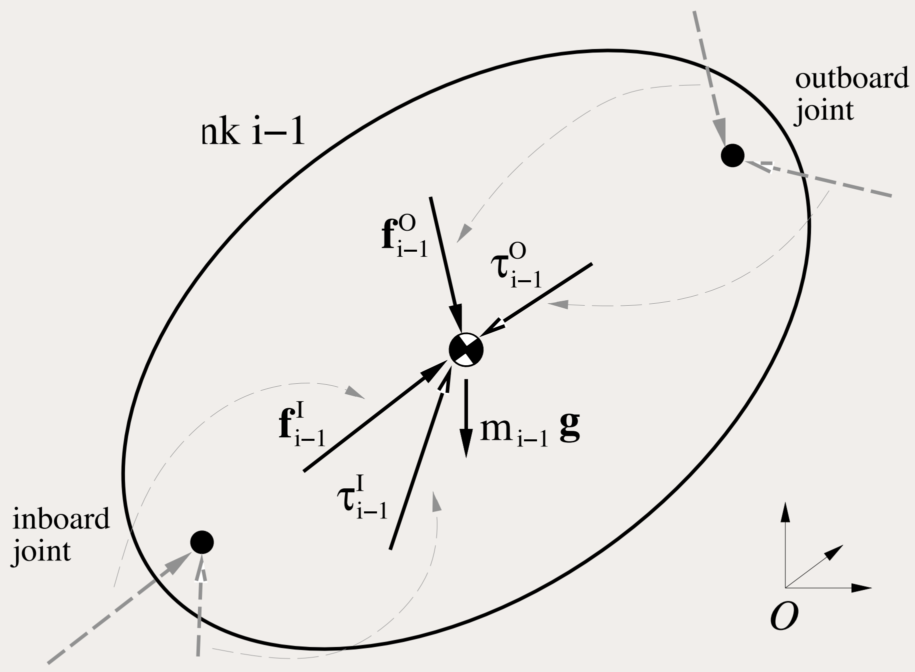
- Filters out velocity field
- Velocity field in sync again

Hair Dynamics

- Control Mesh
- Mass-Spring Systems
- **Rigid Links**
- Super Helices

Featherstone Algorithm





Impulse-based Dynamic Simulation of Rigid Body Systems

by

Brian Vincent Mirtich

If joint i is prismatic,

$$\hat{\mathbf{s}}_i'\hat{\mathbf{f}}_i^I = \begin{bmatrix} \mathbf{0} \\ \mathbf{u}_i \end{bmatrix}' \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} - \mathbf{d}_i \times \mathbf{f} \end{bmatrix} = \mathbf{f} \cdot \mathbf{u}_i.$$

The right hand side is the component of the applied force along the joint axis. This force must be supported by the actuator, hence, it is Q_i. If joint i is revolute,

$$\hat{\mathbf{s}}_i'\hat{\mathbf{f}}_i^I = \begin{bmatrix} \mathbf{u}_i \\ \mathbf{u} \times \mathbf{d}_i \end{bmatrix}' \begin{bmatrix} \mathbf{f} \\ \boldsymbol{\tau} - \mathbf{d}_i \times \mathbf{f} \end{bmatrix} = \mathbf{f} \cdot (\mathbf{u}_i \times \mathbf{d}_i) + (\boldsymbol{\tau} - \mathbf{d}_i \times \mathbf{f}) \cdot \mathbf{u}_i.$$

The right hand side reduces to $\boldsymbol{\tau} \cdot \mathbf{u}_i$, the component of the applied torque along the joint axis. This torque must be supported by the actuator, hence, it is Q_i. □

Substituting equation (4.23) for link i's spatial acceleration into (4.24) yields

$$\hat{\mathbf{f}}_i^I = \hat{\mathbf{I}}_i^A ({}_i\hat{\mathbf{X}}_{i-1}\hat{\mathbf{a}}_{i-1} + \ddot{q}_i\hat{\mathbf{s}}_i + \hat{\mathbf{c}}_i) + \hat{\mathbf{Z}}_i^A.$$

Premultiplying both sides by $\hat{\mathbf{s}}_i'$ and applying Lemma 7 gives

$$Q_i = \hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A ({}_i\hat{\mathbf{X}}_{i-1}\hat{\mathbf{a}}_{i-1} + \ddot{q}_i\hat{\mathbf{s}}_i + \hat{\mathbf{c}}_i) + \hat{\mathbf{s}}_i'\hat{\mathbf{Z}}_i^A,$$

from which $\ddot{q}_i$ may be determined:

$$\ddot{q}_i = \frac{Q_i - \hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\,{}_i\hat{\mathbf{X}}_{i-1}\hat{\mathbf{a}}_{i-1} - \hat{\mathbf{s}}_i'\left(\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i\right)}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}. \quad (4.27)$$

Substituting this expression for $\ddot{q}_i$ into (4.26) and rearranging gives

$$\begin{aligned} \hat{\mathbf{f}}_{i-1}^I \;=\; & \left[\hat{\mathbf{I}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \left(\hat{\mathbf{I}}_i^A - \frac{\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}\right) {}_i\hat{\mathbf{X}}_{i-1}\right]\hat{\mathbf{a}}_{i-1} \\ & + \hat{\mathbf{Z}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \left[\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i + \frac{\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i\left[Q_i - \hat{\mathbf{s}}_i'\left(\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i\right)\right]}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}\right]. \end{aligned}$$

Comparing this to the desired form (4.24),

$$\hat{\mathbf{I}}_{i-1}^A = \hat{\mathbf{I}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \left(\hat{\mathbf{I}}_i^A - \frac{\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}\right) {}_i\hat{\mathbf{X}}_{i-1} \quad (4.28)$$

$$\hat{\mathbf{Z}}_{i-1}^A = \hat{\mathbf{Z}}_{i-1} + {}_{i-1}\hat{\mathbf{X}}_i \left[\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i + \frac{\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i\left[Q_i - \hat{\mathbf{s}}_i'\left(\hat{\mathbf{Z}}_i^A + \hat{\mathbf{I}}_i^A\hat{\mathbf{c}}_i\right)\right]}{\hat{\mathbf{s}}_i'\hat{\mathbf{I}}_i^A\hat{\mathbf{s}}_i}\right]. \quad (4.29)$$

Rigid Links

- Fewer degrees of freedom.
- Torsional forces.
- Difficult Implementation.
- Constraints Difficult.

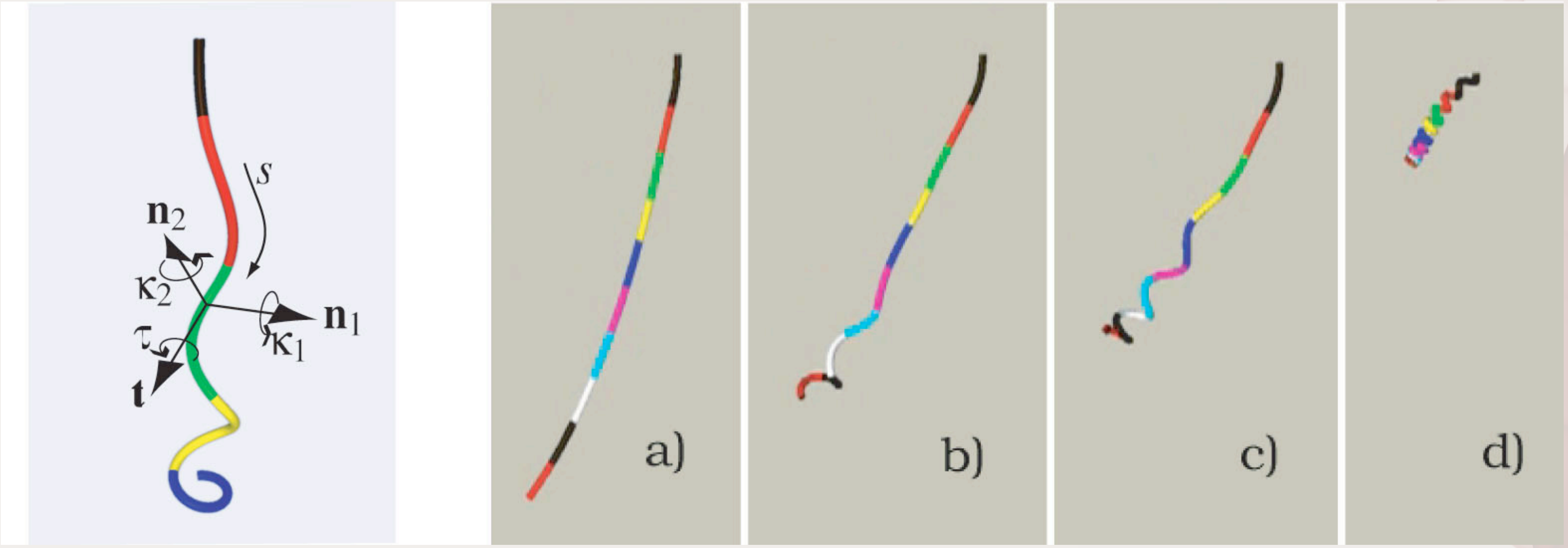
Hair Dynamics

- Control Mesh
- Mass-Spring Systems
- Rigid Links
- **Super Helices**

Super Helices

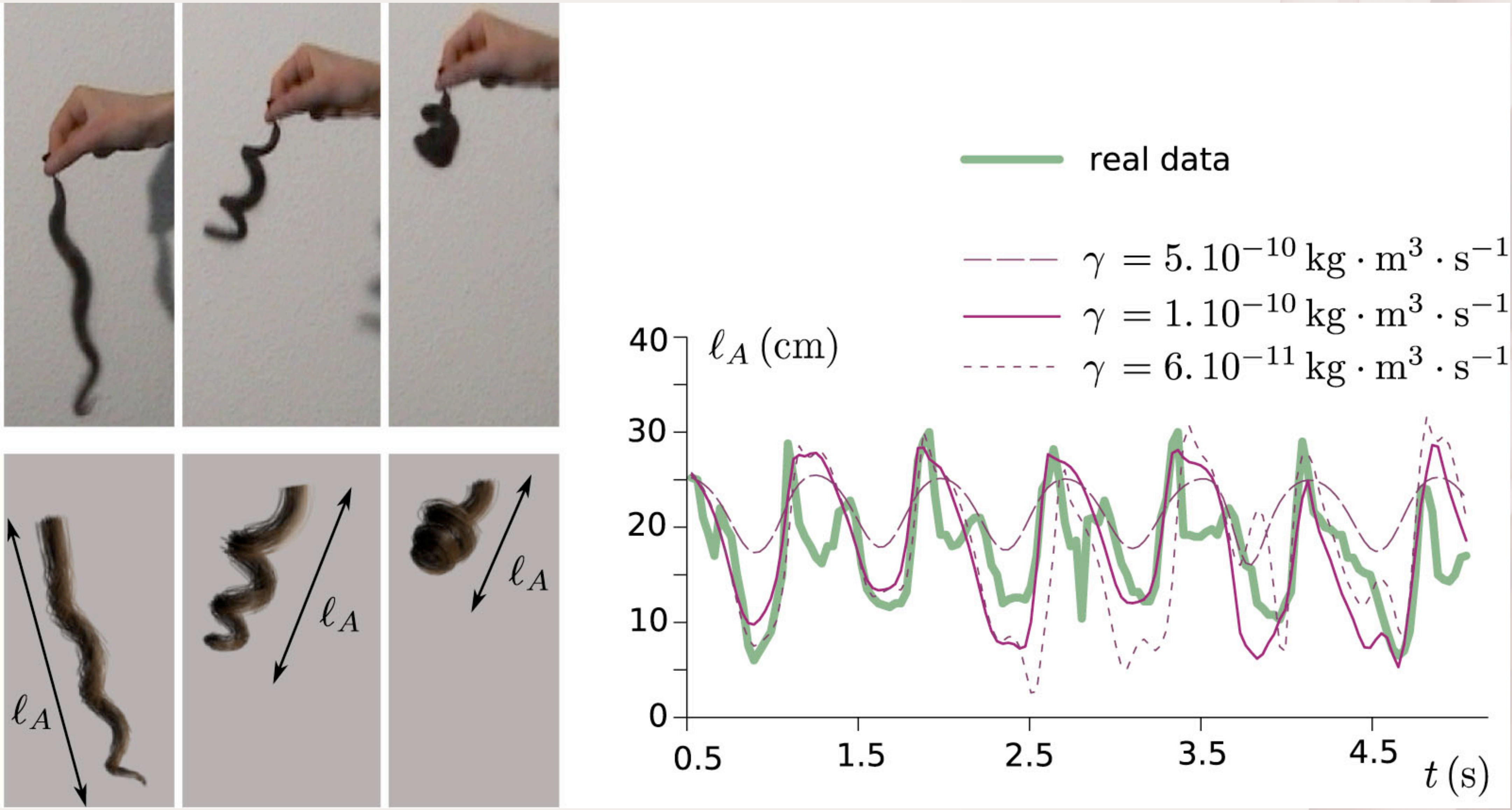
Why just use straight rods?

Super Helices



$$\mathbb{M}[s,\mathbf{q}]\cdot\ddot{\mathbf{q}}+\mathbb{K}\cdot(\mathbf{q}-\mathbf{q}^{\mathrm{n}})=\mathbf{A}[t,\mathbf{q},\dot{\mathbf{q}}]+\int_0^L \mathbf{J}_{iQ}[s,\mathbf{q},t]\cdot\mathbf{F}^{\mathrm{i}}(s,t)\,\mathrm{d}s.$$

Super Helices



Super Helices

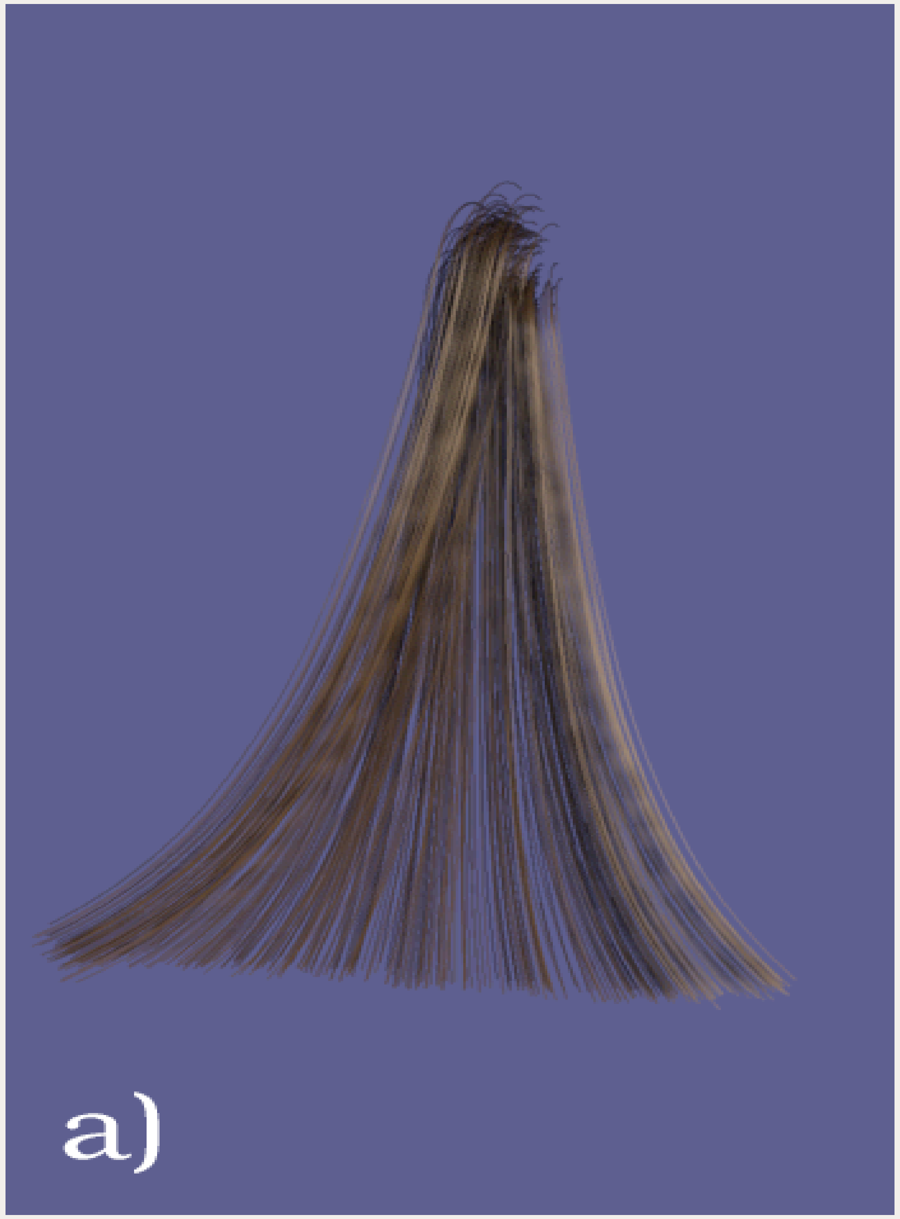
Part 3

Animation of a full head of hair

Overview

- Project
  - Solving Linear Systems
  - Questions About the Project
- Hair
  - Real Hair
  - Hair Dynamics
  - **Hair Rendering**
- Course Evaluations

Rendering



Interpolation

Extrapolation

Next Wednesday

- **Why not simulate every single strand?**
  - ***Jee Lee***

Conclusion

Video

Overview

- Project
  - Solving Linear Systems
  - Questions About the Project
- Hair
  - Real Hair
  - Hair Dynamics
  - Hair Rendering
- **Course Evaluations**

Question

- What are the salient parts of cloth that we want to simulate?
- How could we simulate cloth?
- What are the difficulties / problems with your approach?