גרפיקה ממוחשבת - שיעור 12

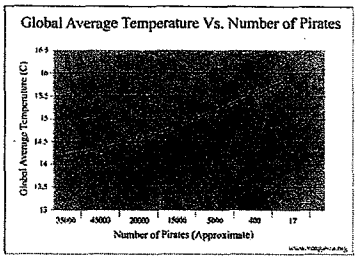
Animation

ליאור שפירא
גרפיקה ממוחשבת
סמסטר ב' תשס"ט

מבוסס על שקפים של תומס פנקהאוסר, פרינסטון

סקר הוראה

- סקר סקר סקר!
- נמשך 20.6 – 31.5
- חשוב!!
- מאוד!!!



2

הצצה לשבוע הבא...

- רן גל יספר לנו על iWires
- גיל הופר יספר לנו על 'Coordinates for Instant Image Cloning', אלטרנטיבה ל-Poisson

- ג'קי יספר לנו על אנימציה של דמויות אנושיות ותמצות תנועות

3

Syllabus

I. Image processing
II. Modeling
III. Rendering
IV. Animation


Image Processing
(Rusty Coleman, CS426, Fall99)


Rendering
(Michael Bostock, CS426, Fall99)


Modeling
(Dennis Zorin, CalTech)


Animation
(Angel, Plate 1)

Computer Animation

- What is animation?
  - Make objects change over time according to scripted actions


Pixar

- What is simulation?
  - Predict how objects change over time according to physical laws

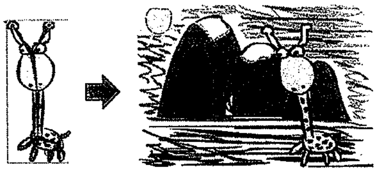

University of Illinois

Computer Animation


Pixar

Outline

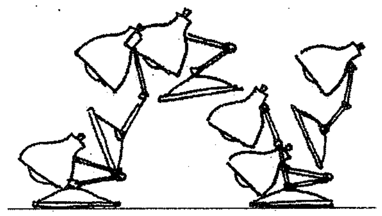
- ➢ Keyframe animation
- Adding inverse kinematics
- Adding dynamics



Lior 09

Keyframe Animation

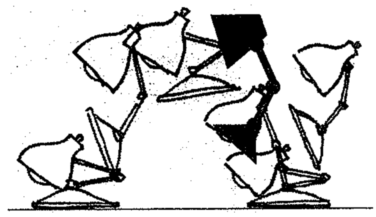
- Define character poses at specific time steps called "keyframes"



Lasseter '87

Keyframe Animation

- Interpolate variables describing keyframes to determine poses for character "in-between"

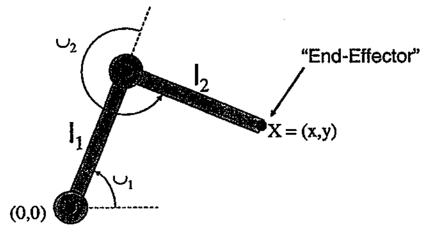


Lasseter '87

כאשר יש לנו מחשב, אנחנו יכולים לייצר
מעט פריימים ולתת למחשב להשלים את השאר.
ההשלמה צריכה להיות "חכמה" וזה לא תמיד פשוט.

Example: 2-Link Structure

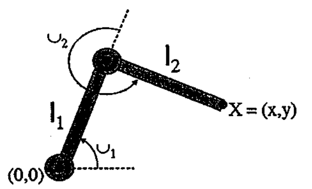
- Two links connected by rotational joints



ניתן להגדיר keyframe באמצעות הגדרת
הזוויות. ע"י כך ניתן למצוא את X (מיקום).

Forward Kinematics

- Animator specifies joint angles: Θ_1 and Θ_2
- Computer finds positions of end-effector: X

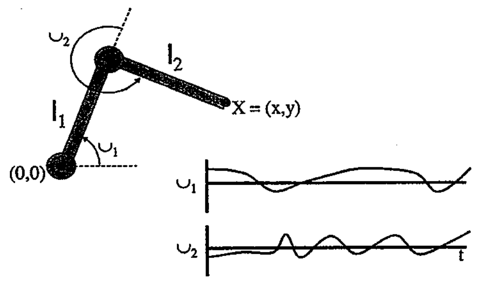


$$X = (l_1 \cos\Theta_1 + l_2 \cos(\Theta_1 + \Theta_2), l_1 \sin\Theta_1 + l_2 \sin(\Theta_1 + \Theta_2))$$

כאן חישוב גיאומטרי פשוט.

Forward Kinematics

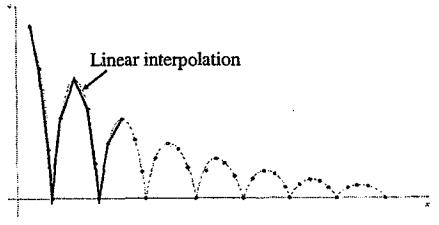
- Joint motions can be specified by spline curves



ניתן להגדיר את השינוי בזווית לאורך
זמן ע"י spline

Keyframe Animation

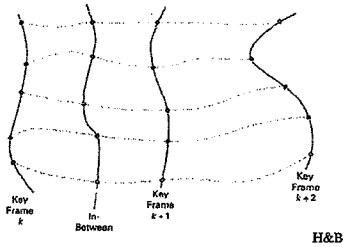
- Inbetweening:
  - Linear interpolation - usually not enough continuity



H&B Figure 16.16

Keyframe Animation

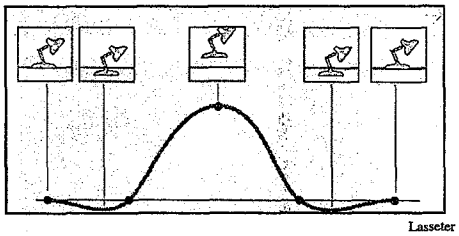
- Inbetweening:
  - Spline interpolation - maybe good enough



H&B Figure 16.11

Keyframe Animation

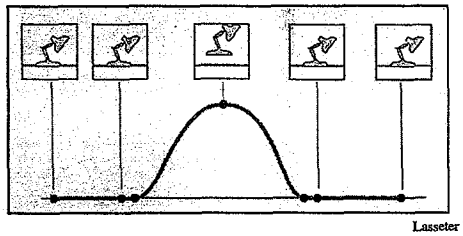
- Inbetweening:
  - Cubic spline interpolation - maybe good enough
    - » May not follow physical laws



Lasseter `87

Keyframe Animation

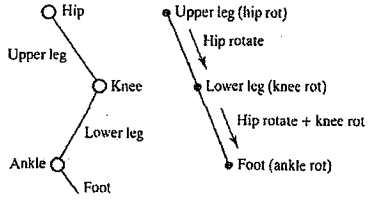
- Inbetweening:
  - Cubic spline interpolation - maybe good enough
    - » May not follow physical laws



Lasseter `87

Example: Walk Cycle

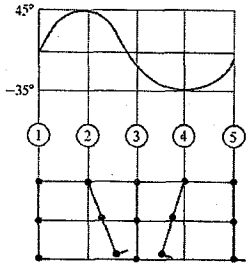
- Articulated figure:



Watt & Watt

Example: Walk Cycle

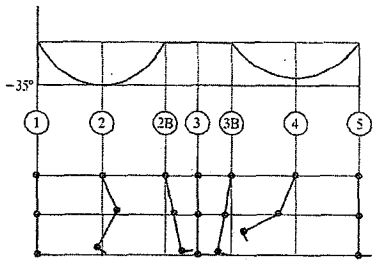
- Hip joint orientation:



Watt & Watt

Example: Walk Cycle

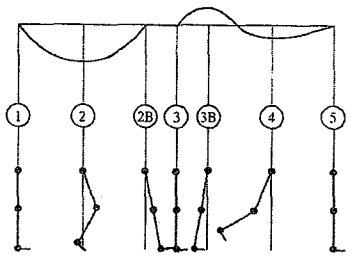
- Knee joint orientation:



Watt & Watt

Example: Walk Cycle

- Ankle joint orientation:



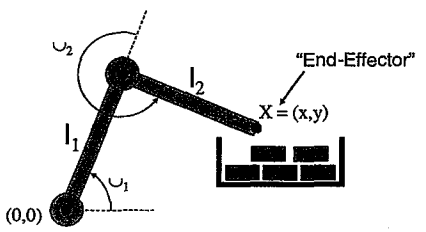
Watt & Watt

Outline

- Keyframe animation
- Adding inverse kinematics
- Adding dynamics

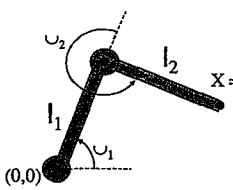
Example: 2-Link Structure

- What if animator knows position of "end-effector"



Inverse Kinematics

- Animator specifies end-effector positions: X
- Computer finds joint angles: υ_1 and υ_2:

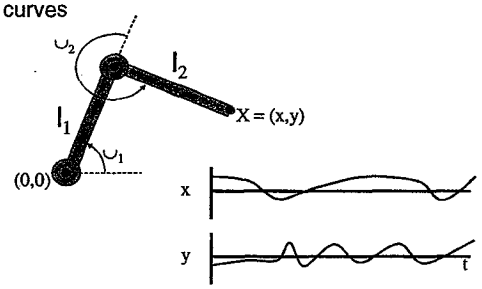


$$\Theta_2 = \cos^{-1}\left(\frac{x^2 + x^2 - l_1^2 - l_2^2}{2l_1 l_2}\right)^{\pm}$$

$$\Theta_1 = \frac{-(l_2 \sin(\Theta_2))x + (l_1 + l_2\cos(\Theta_2))y}{(l_2 \sin(\Theta_2))y + (l_1 + l_2\cos(\Theta_2))x}$$

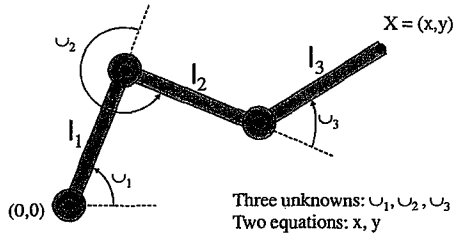
Inverse Kinematics

- End-effector postions can be specified by spline curves



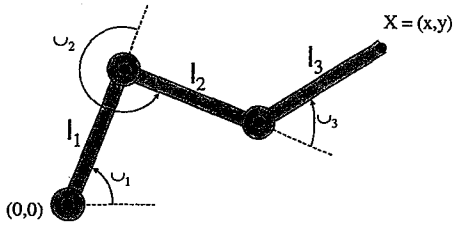
Inverse Kinematics

- Problem for more complex structures
  - System of equations is usually under-defined
  - Multiple solutions

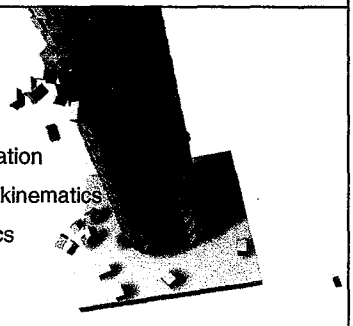


Inverse Kinematics

- Solution for more complex structures:
  - Find best solution (e.g., minimize energy in motion)
  - Non-linear optimization



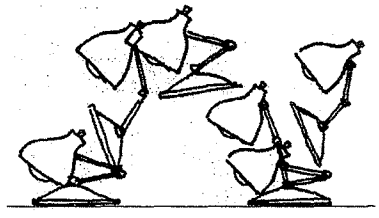
Outline



- Keyframe animation
- Adding inverse kinematics
- Adding dynamics

Dynamics

- Simulation of physics insures realism of motion



Lasseter `87

Spacetime Constraints

- Animator specifies constraints:
  - What the character's physical structure is
    - » e.g., articulated figure
  - What the character has to do (keyframes)
    - » e.g., jump from here to there within time t
  - What other physical structures are present
    - » e.g., floor to push off and land
  - How the motion should be performed
    - » e.g., minimize energy

Spacetime Constraints

- Computer finds the "best" physical motion satisfying constraints
- Example: particle with jet propulsion
  - **x**(t) is position of particle at time t
  - **f**(t) is force of jet propulsion at time t
  - Particle's equation of motion is:

$$mx'' - f - mg = 0$$

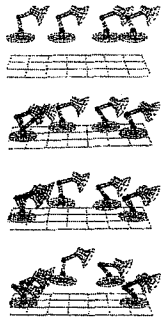
  - Suppose we want to move from a to b within t_0 to t_1 with minimum jet fuel:

Minimize $\int_{t_0}^{t_1} |f(t)^2| dt$ subject to $x(t_0)=a$ and $x(t_1)=b$

Witkin & Kass `88

Spacetime Constraints

- Solve with iterative optimization methods



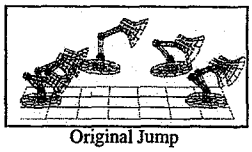
Witkin & Kass `88

Spacetime Constraints

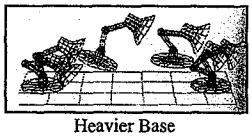
- Advantages:
  - Free animator from having to specify details of physically realistic motion with spline curves
  - Easy to vary motions due to new parameters and/or new constraints
- Challenges:
  - Specifying constraints and objective functions
  - Avoiding local minima during optimization

Spacetime Constraints

- Adapting motion:



Original Jump

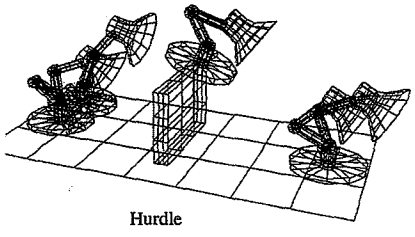


Heavier Base

Witkin & Kass `88

Spacetime Constraints

- Adapting motion:

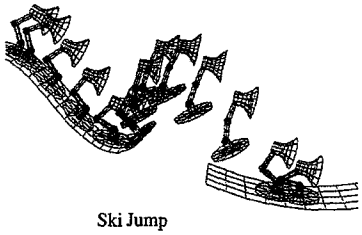


Hurdle

Witkin & Kass `88

Spacetime Constraints

- Adapting motion:



Ski Jump

Witkin & Kass `88

Spacetime Constraints

- Advantages:
  - Free animator from having to specify details of physically realistic motion with spline curves
  - Easy to vary motions due to new parameters and/or new constraints
- Challenges:
  - Specifying constraints and objective functions
  - Avoiding local minima during optimization

Example: Manipulation of Sims.

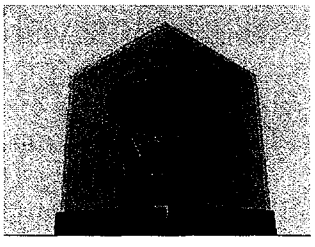
Interactive Manipulation of Rigid Body Simulations.
Popovic et al Siggraph 2000.

Popovic

Summary

- Keyframe animation
  - Poses specified at key times
  - In-betweening to fill in the rest
- Incorporating inverse kinematics
  - Makes keyframes easier to specify
- Incorporating dynamics
  - Makes animation easier to adapt

Simulation



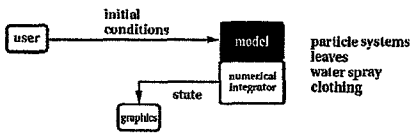
39

Simulation

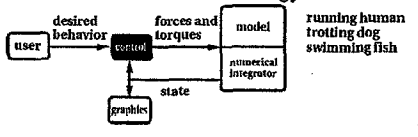
- Dynamics
  - Considers underlying forces
  - Compute motion from initial conditions and physics
- Kinematics
  - Considers only motion
  - Determined by positions, velocities, accelerations

Dynamics

Passive--no muscles or motors



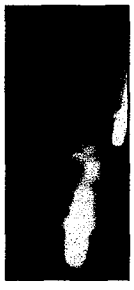
Active--internal source of energy



Hodgins

Passive Dynamics

- No muscles or motors
  - Smoke
  - Water
  - Cloth
  - Fire
  - Fireworks
  - Dice



McAllister

Passive Dynamics

- Physical laws
  - Newton's laws
  - Hook's law
  - Etc.
- Physical phenomena
  - Gravity
  - Momentum
  - Friction
  - Collisions
  - Elasticity
  - Fracture

McAllister

Fun with Bunny

Porous Flow in Particle-Based Fluid Simulations
Lenaerts et al, SIGGRAPH 2007

Fire with cellular patterns
Ron Fedkiw

A Finite Element Method for Animating Large Viscoplastic Flow
Bargteil et al SIGGRAPH 2007

44

Particle Systems

- A particle is a point mass
  - Mass
  - Position
  - Velocity
  - Forces
  - Color
  - Lifetime

- Use lots of particles to model complex phenomena
  - Keep array of particles
  - Newton's laws

Our Particle


```
enum ParticleType { Create,Update };

struct Vector2d{
   double x;
   double y;

};

struct Particle {
   Vector2d Pos; //Position of the particle
   Vector2d Vel; //Velocity of the particle
   int age; //Current age of the particle
   int LifeSpan; //Age after which the particle dies
   int color;
   int size;

};
```

46

Particle Systems

- For each frame:
  - Create new particles and assign attributes
  - Delete any expired particles
  - Update particles based on attributes and physics
  - Render particles

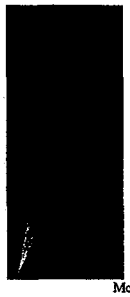
Our Particle

```
void Init(long num_part,long num_forces, Vector2d Forces[],
   ParticleType part_type,Vector2d vel, Vector2d pos1,Vector2d
   pos2, int lifespan, int color, int size){

Particles = new Particle[num_part];
   ParticleNum=num_part;

for(int i=0;i<num_forces;i++){
      TotForce.x+=Forces[i].x;
      TotForce.y+=Forces[i].y;

}
   Particle_Type=part_type;
   Vel=vel;
   Pos1=pos1;
   Pos2=pos2;
   LifeSpan=lifespan;
   Color=color;
   Size=size;
   randomize();
   InitParticles();

}
```

48

Creating Particles

- Where to create particles?
  - ➢ Predefined source
  - ◦ Surface of shape
  - ◦ Where particle density is low
  - ◦ etc.

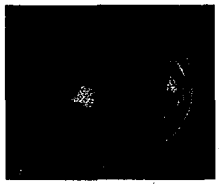


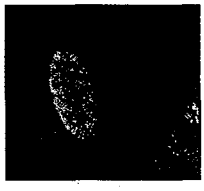
McAllister

Creating Particles

- Where to create particles?
  - ◦ Predefined source
  - ➢ Surface of shape
  - ◦ Where particle density is low
  - ◦ etc.

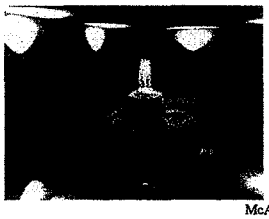
Reeves





Deleting Particles

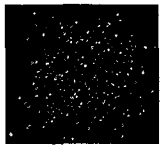
- When to delete particles?
  - ◦ Predefined sink
  - ◦ Surface of shape
  - ◦ Where density is high
  - ◦ Life span
  - ◦ Random



McAllister

Rendering Particles

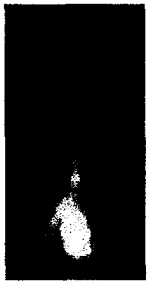
- Rendering styles
  - ➢ Points
  - ◦ Polygons
  - ◦ Shapes
  - ◦ Trails
  - ◦ etc.



McAllister

Rendering Particles

- Rendering styles
  - ◦ Points
  - ➢ Polygons
  - ◦ Shapes
  - ◦ Trails
  - ◦ etc.



McAllister

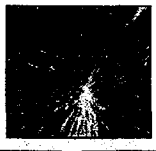
Rendering Particles

- Rendering styles
  - ◦ Points
  - ◦ Polygons
  - ➢ Shapes
  - ◦ Trails
  - ◦ etc.

McAllister

Rendering Particles

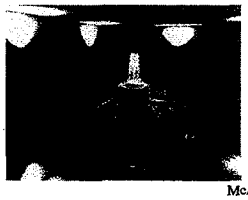
- Rendering styles
  - Points
  - Polygons
  - Shapes
  - ➢Trails
  - etc.



McAllister

Particle Systems

- For each frame:
  - Create new particles and assign attributes
  - Delete any expired particles
  - ➢Update particles based on attributes and physics
  - Render particles



McAllister

Equations of Motion

- Newton's Law for a point mass
  - f = ma
- Computing particle motion requires solving second-order differential equation

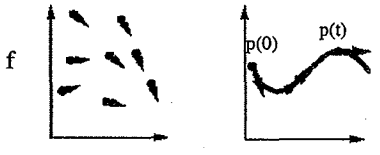
$$\ddot{x} = \frac{f(x,\dot{x},t)}{m}$$

- Add variable v to form coupled first-order differential equations

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

Solving the Equations of Motion

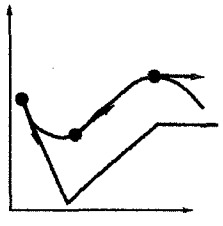
- Initial value problem
  - Know p(0), v(0), a(0)
  - Can compute force at any time and position
  - Compute p(t) by forward integration



Hodgins

Solving the Equations of Motion

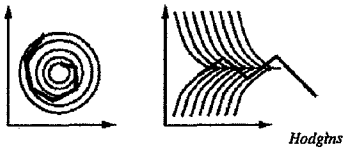
- Euler integration
  - p(t+⊗t)=p(t) + ⊗t v(t)
  - v(t+⊗t)=v(t) + ⊗t f(x,t)/m



Hodgins

Solving the Equations of Motion

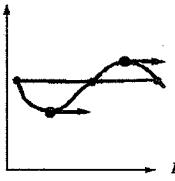
- Euler integration
  - p(t+⊗t)=p(t) + ⊗t v(t)
  - v(t+⊗t)=v(t) + ⊗t f(p(t),t)/m
- Problem:
  - Accuracy decreases as ⊗t gets bigger



Hodgins

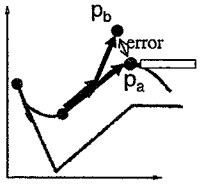
Solving the Equations of Motion

- Midpoint method (2nd order Runge-Kutta)
  - Compute an Euler step
  - Evalute f at the midpoint
  - Take an Euler step using midpoint force
    - » v(t+⊗t)=v(t) + ⊗t f(p(t) + 0.5*⊗t v(t),t)



Solving the Equations of Motion

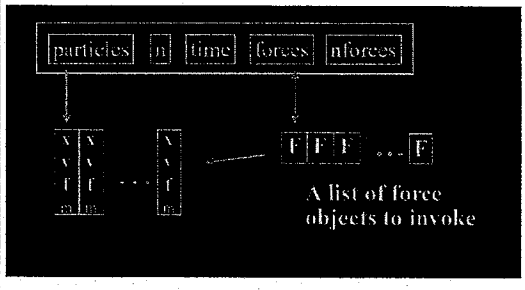
- Adapting step size
  - Compute p_a by taking one step of size h
  - Compute p_b by taking 2 steps of size h/2
  - Error = $| p_a - p_b |$
  - Multiply step size by factor (constant/error)



Particle System Forces

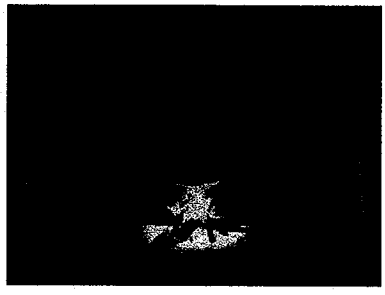
- Force fields
  - Gravity, wind, pressure
- Viscosity/damping
  - Liquids, drag
- Collisions
  - Environment
  - Other particles
- Other particles
  - Springs between neighboring particles (mesh)
  - Useful for cloth

Particle System Forces



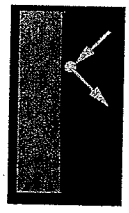
Example: Gravity

Example: Fire



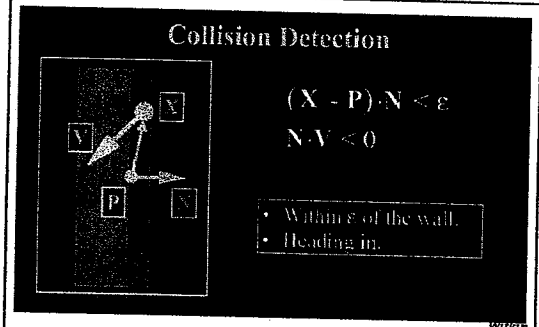
Example: Bouncing Off Wall

- Requires
  - Collision detection
  - Collision response (dynamic forces)

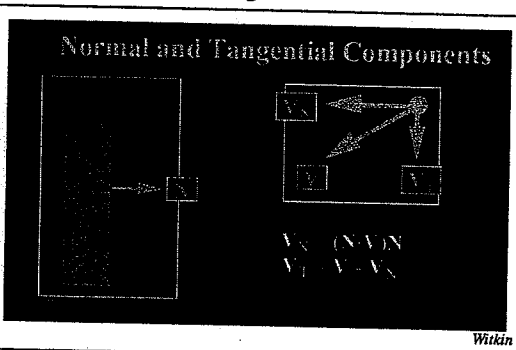


Witkin

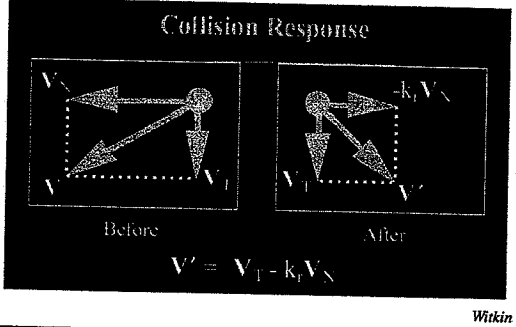
Example: Bouncing Off Wall



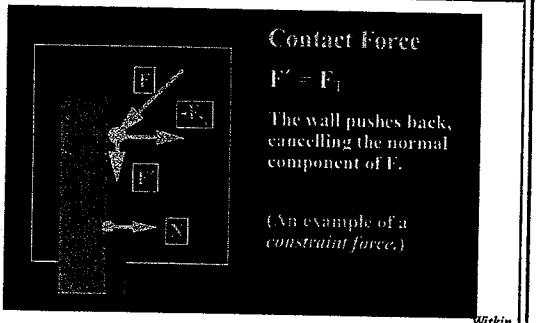
Example: Bouncing Off Wall



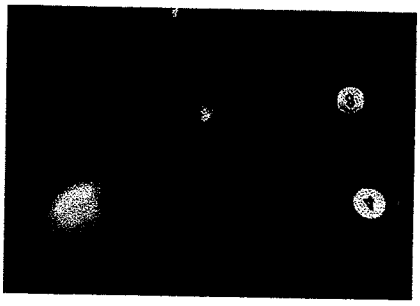
Example: Bouncing Off Wall



Example: Bouncing Off Wall



Example: Bouncing Off Particles



Advancing our Particles

```
void Run(){
    while(!kbhit()){
        delay(10);
        cleardevice();
        //delay(10);
        for(int i=0;i<ParticleNum;i++){
          if(Particles[i].age >= 0){
            setfillstyle(1,Particles[i].color);
            setcolor(Particles[i].color);
            //circle((int)Particles[i].Pos.x,(int)Particles[i].Pos.y,Particles[i].size);
            fillellipse(
            Par
            Par
            Par
            Par
            Par
```

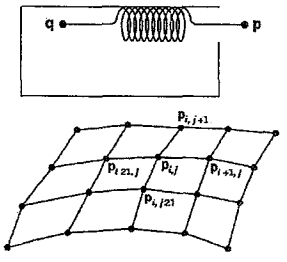
73

Example: Cloth

- Spring-mass mesh
- Hooke's law

$$f = -k_s(|d| - s)\frac{d}{|d|}$$

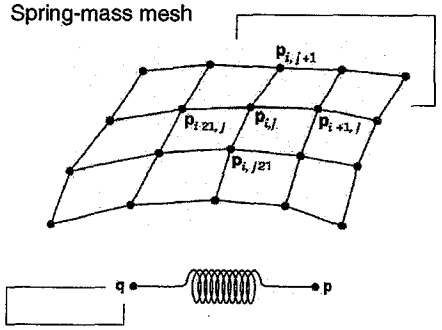
f = force
k_s = spring constant
d = p - q
s = resting length



Hodgins

Example: Cloth

- Spring-mass mesh



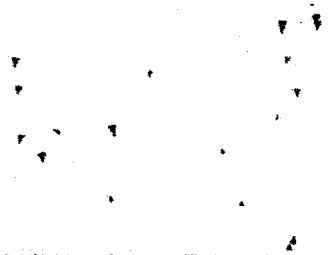
Hodgins

Example: Cloth

Animating Developable Surfaces

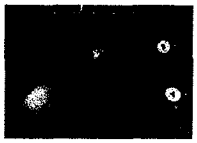
Simulating Knitted Cloth at the Yarn Level

Example: Flocks & Herds



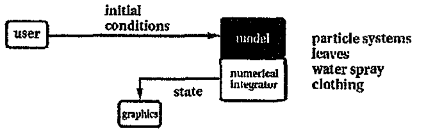
Summary

- Particle systems
  - Lots of particles
  - Simple physics
- Interesting behaviors
  - Waterfalls
  - Smoke
  - Cloth
  - Flocks
- Solving motion equations
  - Simplest method is Euler integration
  - Better to use adaptive step sizes

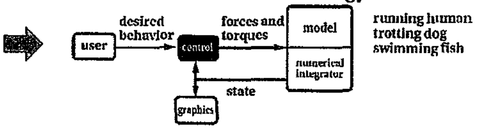


Passive vs. Active Dynamics

Passive—no muscles or motors



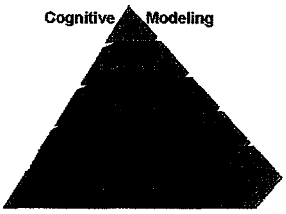
Active—internal source of energy



Hodgins

Active Dynamics

- Motions
  - Physics
  - Controllers
  - Learning
- Behaviors
  - States
- Cognition
  - Planning



Funge99

Motion

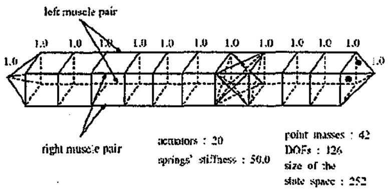
- Example 1: how do worms move?

Grzeszczuk95

Snake Motion

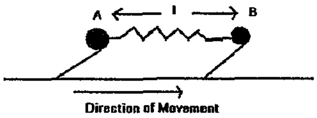
Grzeszczuk95

Worm Biomechanical Model



Grzeszczuk95

Worm Physics



$$f = k(L - I) - D\frac{dl}{dt}$$

$$a = f / m$$

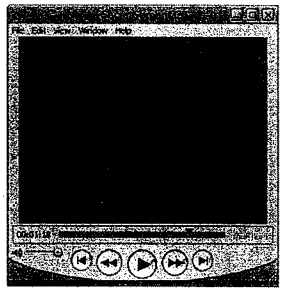
$$X = \frac{1}{m}\iint f\,dt\,dt$$

f = force along spring direction
k = spring force constant
D = damping force
I = current spring length
L = minimum energy spring length

... plus forces due to friction with ground. *Miller88*

מגדירים גם תנועה כמו מערכת קפיצים (נמתחים לכמה שחרורים וכו' לאיזון).

Her Majesty's Secret Serpent



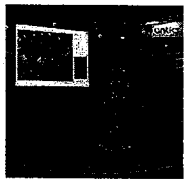
Miller89

Time permitting...

OTHER TOPICS

Other topics in Animation

- Motion Capture

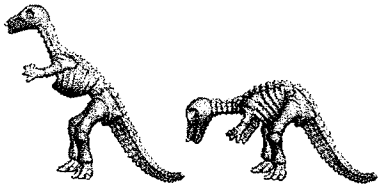


- Skinning

- Deformations
  - Differential representation
  - Cage deformations
  - Deformation transfer

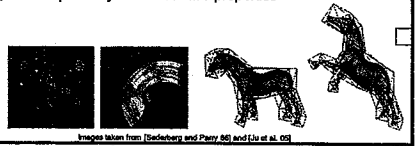
What do we expect from surface deformation?

- Smooth effect on the large scale
- As-rigid-as-possible effect on the small scale (preserves details)



Several approaches

- FFD (space deformation)
  - Lattice-based (Sederberg & Parry 86, Coquillart 90, ...)
  - Curve-/handle-based (Singh & Fiume 98, Botsch et al. 05, ...)
  - Cage-based (Ju et al. 05, Joshi et al. 07, Kopf et al. 07)
- Pros:
  - efficiency almost independent of the surface resolution
  - possible reuse
- Cons:
  - space warp, so can't precisely control surface properties



Images taken from [Sederberg and Parry 86] and [Ju et al. 05]

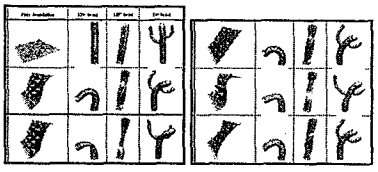
Several approaches

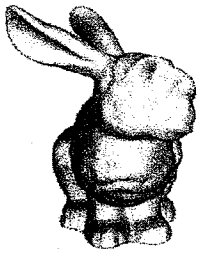
- Surface-based approaches
  - Multiresolution modeling
    Zorin et al. 97, Kobbelt et al. 98, Lee 98, Guskov et al. 99, Botsch and Kobbelt 04, ...
  - Differential coordinates – linear optimization
    Lipman et al. 04, Sorkine et al. 04, Yu et al. 04, Lipman et al. 05, Zayer et al. 05, Botsch et al. 06, Fu et al. 06, ...
  - Non-linear global optimization approaches
    Kraevoy & Sheffer 04, Sumner et al. 05, Hunag et al. 06, Au et al. 06, Botsch et al. 06, Shi et al. 07, ...

Images taken from Pirkle, Botsch et al. 06

Surface-based approaches

- Pros:
  - direct interaction with the surface
  - control over surface properties
- Cons:
  - linear optimization suffers from artifacts (e.g. translation insensitivity)
  - non-linear optimization is more expensive and non-trivial to implement





Sorkine et al 2004

LAPLACIAN SURFACE EDITING

http://www.cs.nyu.edu/~sorkine/ProjectPages/Editing/le.html

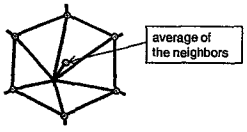
92

Differential Coordinates

- **Differential coordinates** are defined by the discrete Laplacian operator:

$$\delta_i = L(\mathbf{v}_i) = \mathbf{v}_i - \frac{1}{d_i}\sum_{j \in N(i)} \mathbf{v}_j$$

- For highly irregular meshes: cotangent weights [Desbrun et al. 99]



Why differential coordinates?

- They represent the **local** detail / local shape description
  - The direction approximates the normal
  - The size approximates the mean curvature

$$\delta_i = \frac{1}{d_i}\sum_{v \in N(i)} (\mathbf{v}_i - \mathbf{v})$$

$$\frac{1}{len(\gamma)}\int_{v \in \gamma} (\mathbf{v}_i - \mathbf{v})\,ds$$

$$\lim_{len(\gamma)\to 0} \frac{1}{len(\gamma)}\int_{v \in \gamma} (\mathbf{v}_i - \mathbf{v})\,ds = H(\mathbf{v}_i)\mathbf{n}_i$$

Why differential coordinates?

- Local detail representation – enables detail preservation through various modeling tasks
- Representation with sparse matrices
- Efficient linear surface reconstruction

Overall framework

- Compute differential representation

$$\Delta = L(V)$$

- Pose modeling constraints

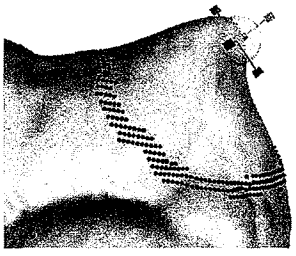
$$\mathbf{v}'_i = \mathbf{u}_i, \quad i \in C$$

- Reconstruct the surface – in least-squares sense

$$\tilde{V}' = \arg\min_{V'} \left(\|L(V') - \Delta\|^2 + \sum_{i \in C} \|\mathbf{v}'_i - \mathbf{u}_i\|^2 \right)$$

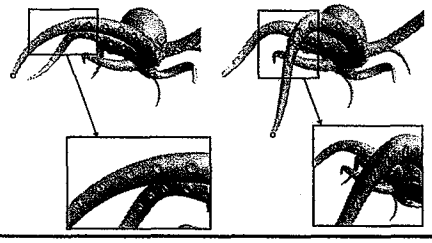
Overall framework

- ROI is bounded by a belt (static anchors)
- Manipulation through handle(s)



Problem: invariance to transformations

- The basic Laplacian operator is translation-invariant, but not rotation- and scale-invariant
- Reconstruction attempts to preserve the original global orientation of the details



Implicit definition of transformations

- The idea: solve for local transformations AND the edited surface simultaneously!

$$\tilde{V}' = \arg\min_{V'} \left(\sum_{i=1}^{n} \left\| L(\mathbf{v}_i') - T_i(\boldsymbol{\delta}_i) \right\|^2 + \sum_{j \in C} \left\| \mathbf{v}_j' - \mathbf{u}_j \right\|^2 \right)$$

Transformation of the local frame

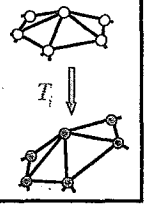
Defining the transformations T_i

$$\tilde{V}' = \arg\min_{V'} \left(\sum_{i=1}^{n} \left\| L(\mathbf{v}_i') - T_i(\boldsymbol{\delta}_i) \right\|^2 + \sum_{j \in C} \left\| \mathbf{v}_j' - \mathbf{u}_j \right\|^2 \right)$$

- How to formulate T_i?
  - Based on the local (1-ring) neighborhood
  - Linear dependence on the unknown $\mathbf{v}_i'$

Members of the 1-ring of i-th vertex

$$\begin{cases} \mathbf{v}'_{i_1} = T_i \mathbf{v}_{i_1} \\ \mathbf{v}'_{i_2} = T_i \mathbf{v}_{i_2} \\ \vdots \\ \mathbf{v}'_{i_k} = T_i \mathbf{v}_{i_k} \end{cases}$$



Defining the transformations T_i

- First attempt: define T_i simply by solving

$$T_i = \arg\min_{T_i} \sum_{j=1}^{k} \left\| \mathbf{v}'_{i_j} - T_i \mathbf{v}_{i_j} \right\|^2$$

$$\left(\quad T_i \quad \right) = \begin{pmatrix} | & | & & | \\ \mathbf{v}'_{i_1} & \mathbf{v}'_{i_2} & \cdots & \mathbf{v}'_{i_k} \\ | & | & & | \end{pmatrix} \begin{pmatrix} | & | & & | \\ \mathbf{v}_{i_1} & \mathbf{v}_{i_2} & \cdots & \mathbf{v}_{i_k} \\ | & | & & | \end{pmatrix}^*$$

Defining the transformations T_i

- Plug the expressions for T_i into the least-squares reconstruction formula:

$$\tilde{V}' = \arg\min_{V'} \left(\sum_{i=1}^{n} \left\| L(\mathbf{v}_i') - T_i \boldsymbol{\delta}_i \right\|^2 + \sum_{j \in C} \left\| \mathbf{v}_j' - \mathbf{u}_j \right\|^2 \right)$$

Linear combination of the unknown $\mathbf{v}_i'$

Constraining T_i

- Trivial solution for T_i will result in membrane surface reconstruction
- To preserve the shape of the details we constrain T_i to **rotations, uniform scales and translations**

$$T_i = \begin{pmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{pmatrix}$$

Linear constraints on t_{lm} so that T_i is rotation+scale+translation
??

Constraining T_i – 3D case

- Not linear in 3D:

$$\begin{pmatrix} \text{rotation} + \\ \text{uniform scale} \end{pmatrix} = s \exp H = s\left(\alpha I + \beta H + \mathbf{h}^T\mathbf{h}\right)$$

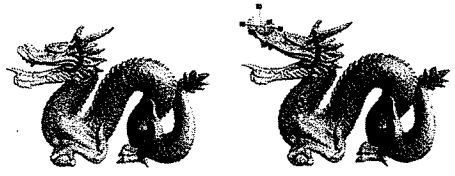
H is 3×3 skew-symmetric, $H\mathbf{x} = \mathbf{h}\times\mathbf{x}$

- Linearize by dropping the quadratic term

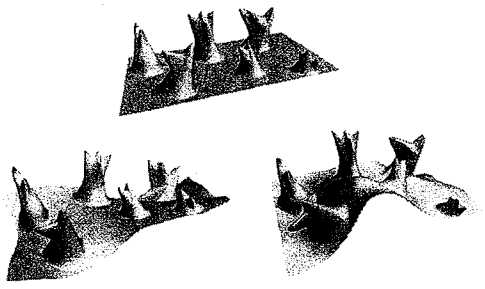
Adjusting T_i

- Due to linearization, T_i scale the space along the $\mathbf{h}$ axis by $cos\theta$
- When θ is large, this causes anisotropy
- Possible correction:
  - Compute T_i, remove the scaling component and reconstruct the surface again from the corrected δ_i
  - Apply our technique from [Lipman et al. 04] first, and then the current technique – with small θ.

Some results



Some results



Some results