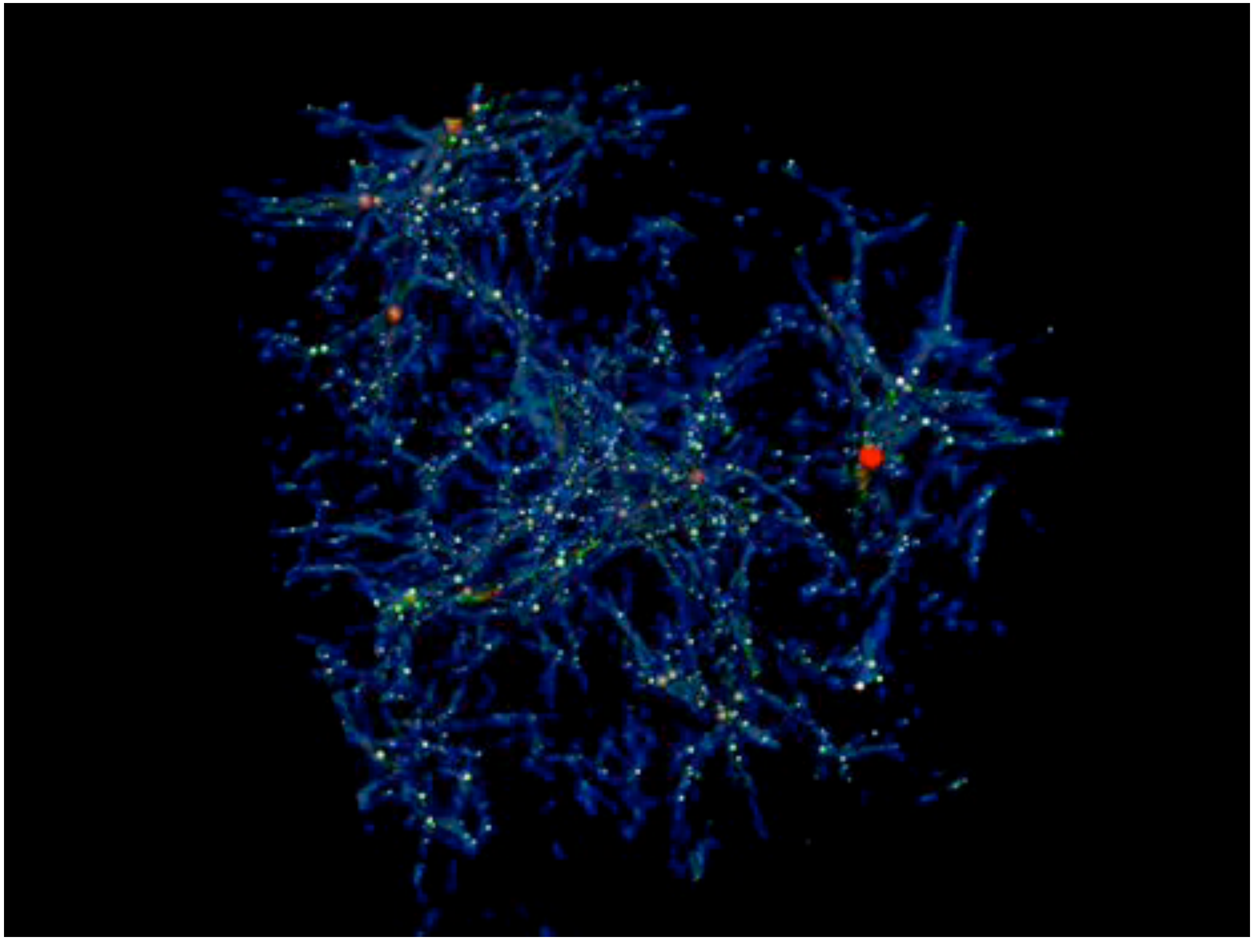
Supplement to Lecture 25

Animation I: Articulate Systems

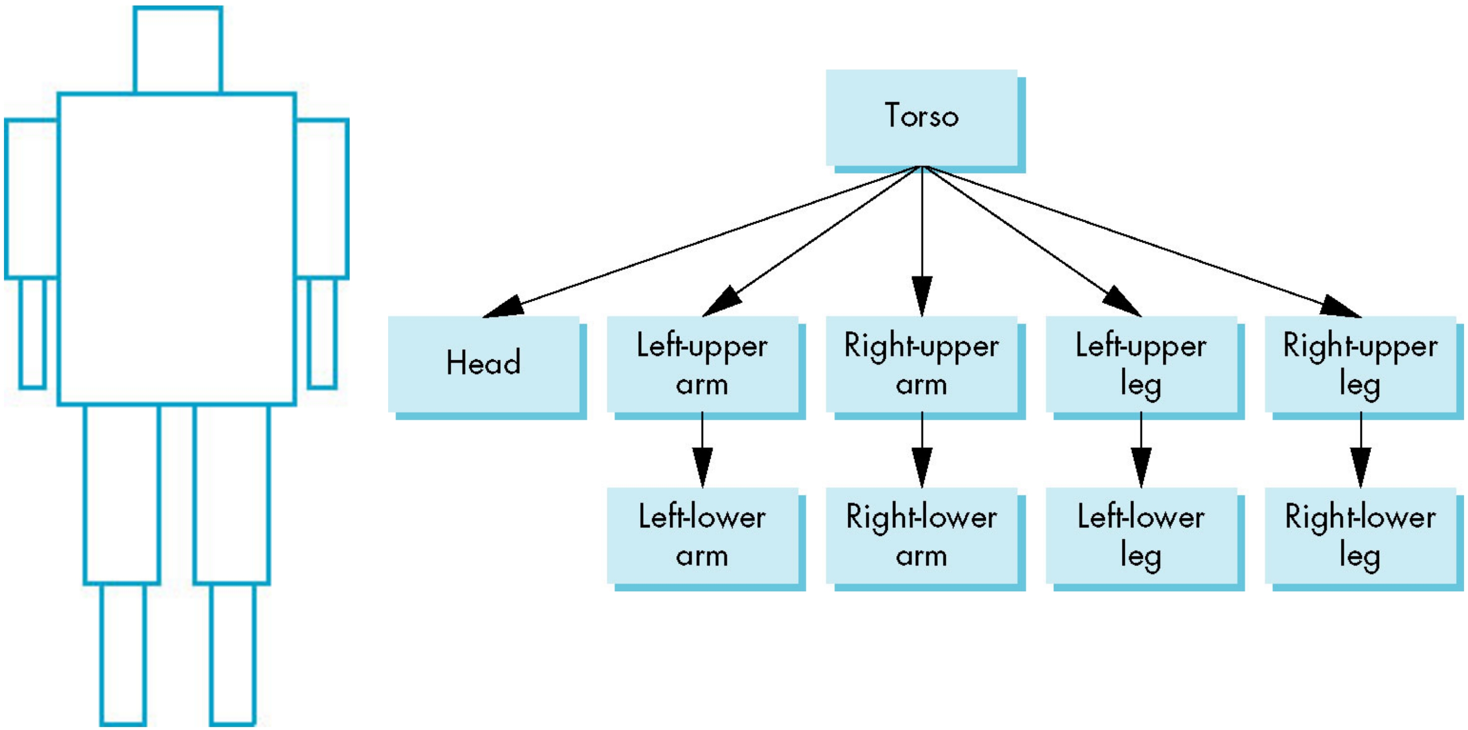
Animation II: Particle Systems



Animation I: Articulate Systems

A worrying warrior is a worrior

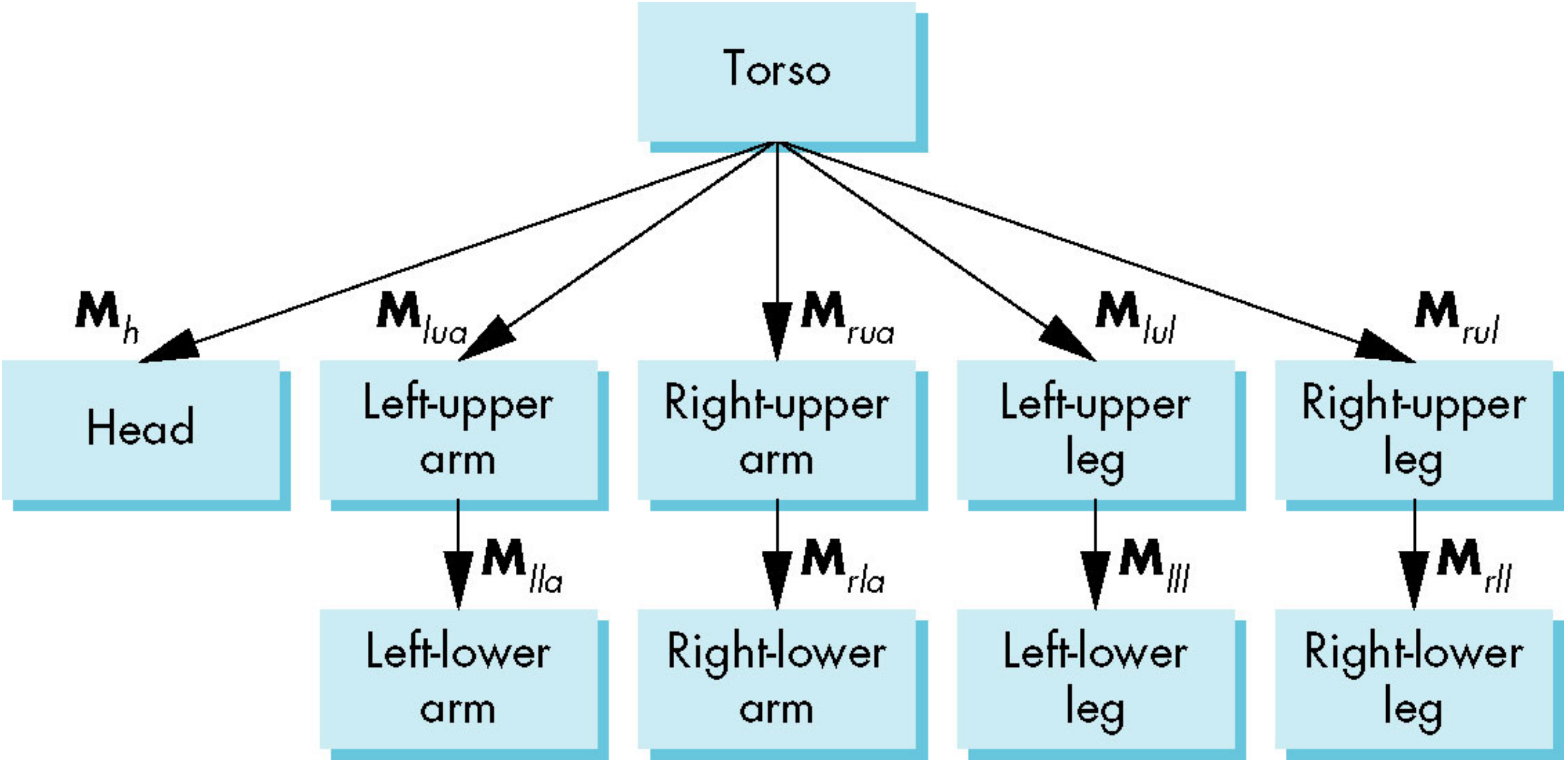
Humanoid Figure



Building the Model

- Can build a simple implementation using quadrics: ellipsoids and cylinders
- Access parts through functions
  - `torso()`
  - `left_upper_arm()`
- Matrices describe position of node with respect to its parent
  - $\mathbf{M}_{lla}$ positions left lower leg with respect to left upper arm

Tree of Matrices



Transformation Matrices

- There are 10 relevant matrices
  - $\mathbf{M}$ positions and orients entire figure through the torso which is the root node
  - $\mathbf{M}_h$ positions head with respect to torso
  - $\mathbf{M}_{lua}$, $\mathbf{M}_{rua}$, $\mathbf{M}_{lul}$, $\mathbf{M}_{rul}$ position arms and legs with respect to torso
  - $\mathbf{M}_{lla}$, $\mathbf{M}_{rla}$, $\mathbf{M}_{lll}$, $\mathbf{M}_{rll}$ position lower parts of limbs with respect to corresponding upper limbs

Traversal & Display

- The position of the figure is determined by 11 joint angles (two for the head and one for each other part)
- Display of the tree requires a *graph traversal*
  - Visit each node once
  - Display function at each node that describes the part associated with the node, applying the correct transformation matrix for position and orientation

NOTES

- The position of figure is determined by 11 joint angles stored in **`theta[11]`**
- Animate by changing the angles and redisplaying
- We form the required matrices using **`glRotate`** and **`glTranslate`**
  - More efficient than software
  - Because the matrix is formed in model-view matrix, we may want to first push original model-view matrix on matrix stack

Stack Based Traversal

- Set model-view matrix to $\mathbf{M}$ and draw torso
- Set model-view matrix to $\mathbf{MM}_{\mathrm{h}}$ and draw head
- For left-upper arm need $\mathbf{MM}_{\mathrm{lua}}$ and so on
- Rather than recomputing $\mathbf{MM}_{\mathrm{lua}}$ from scratch or using an inverse matrix, we can use the matrix stack to store $\mathbf{M}$ and other matrices as we traverse the tree

Traversal Code

```
figure() {
    glPushMatrix()
    torso();
    glRotate3f(…);
    head();
    glPopMatrix();
    glPushMatrix();
    glTranslate3f(…);
    glRotate3f(…);
    left_upper_arm();
    glPopMatrix();
    glPushMatrix();
```

save present model-view matrix

update model-view matrix for head

recover original model-view matrix

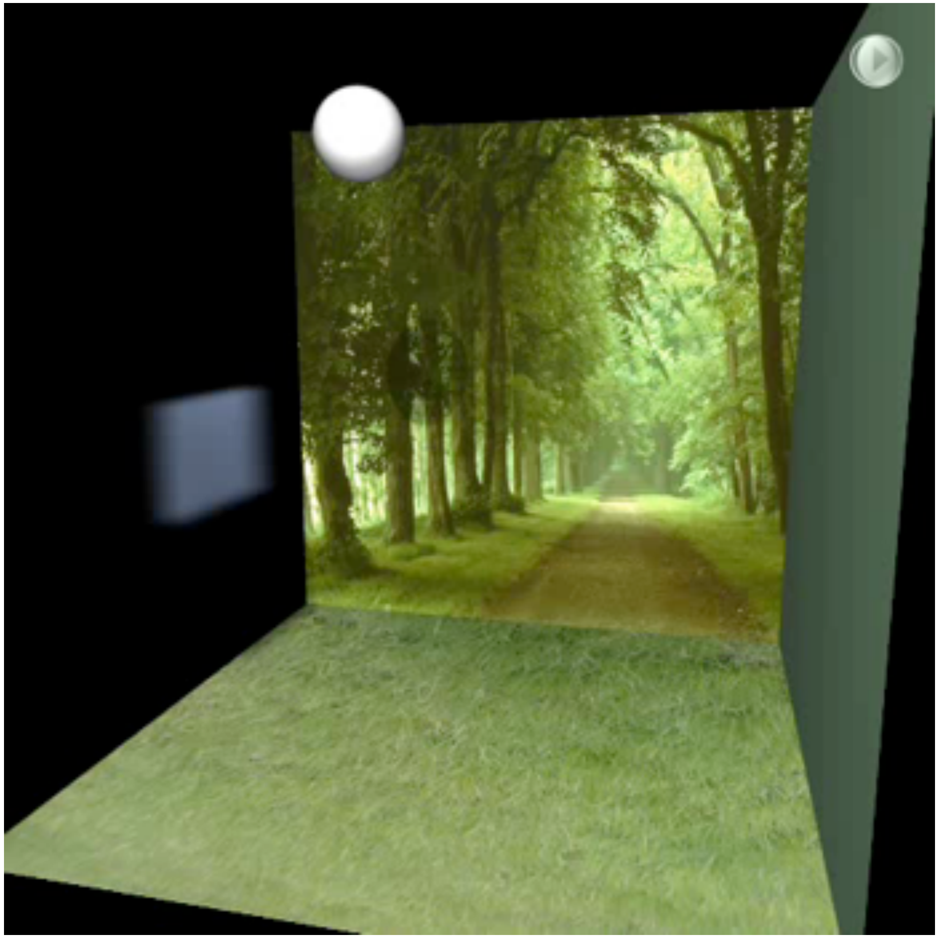
save it again

update model-view matrix for left upper arm

recover and save original model-view matrix again

rest of code

Animation II: Particle Systems



- Most important of procedural methods
- Used to model
  - Natural phenomena
    - Clouds
    - Terrain
    - Plants
  - Crowd Scenes
  - Real physical processes

Newtonian Particle

- Particle system is a set of particles
- Each particle is an ideal point mass
- Six degrees of freedom
  - Position
  - Velocity
- Each particle obeys Newtons' law

$$f = ma$$

Particle Equations

$\mathbf{p}_i = (x_i, y_i\ z_i)$

$\mathbf{v}_i = d\mathbf{p}_i /dt = \mathbf{p}_i' = (dx_i /dt, dy_i /dt\ , z_i /dt)$

$m\ \mathbf{v}_i' = \mathbf{f}_i$

Hard part is defining force vector

Force Vector

- Independent Particles
  - Gravity
  - Wind forces
  - O(n) calulation
- Coupled Particles O(n)
  - Meshes
  - Spring-Mass Systems
- Coupled Particles $O(n^2)$
  - Attractive and repulsive forces

Solution of Particle Systems

```
float time, delta state[6n],
 force[3n];
state = initial_state();
for(time = t0; time<final_time,
 time+=delta) {
force =  force_function(state,
 time);
state = ode(force, state, time,
 delta);
render(state, time)
}
```

Simple Forces

- Consider force on particle i

$$\mathbf{f}_i = \mathbf{f}_i(\mathbf{p}_i, \mathbf{v}_i)$$

- Gravity $\mathbf{f}_i = \mathbf{g}$

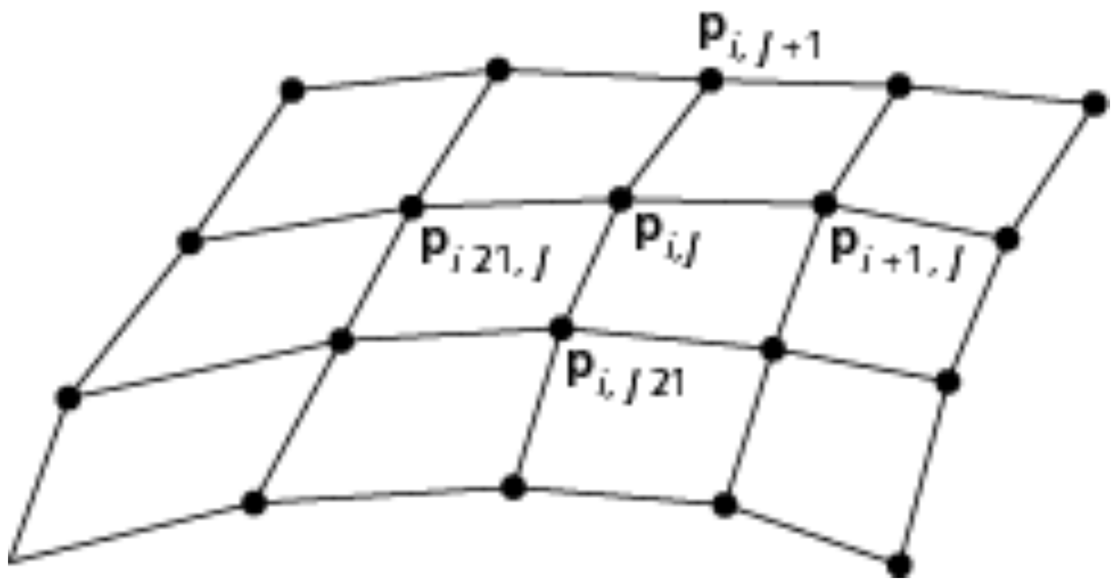
$$\mathbf{g}_i = (0, -g, 0)$$

- Wind forces
- Drag

$\mathbf{p}_i(t_0), \mathbf{v}_i(t_0)$

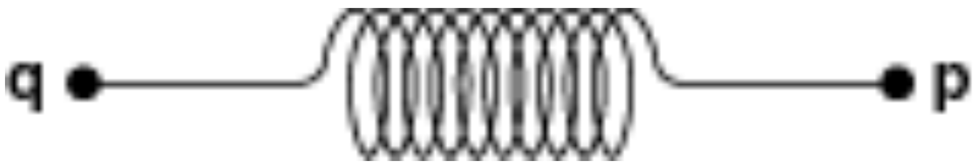
Neighborhood Graph

- Connect each particle to its closest neighbors
  - O(n) force calculation
- Use spring-mass system



Spring Forces

- Assume each particle has unit mass and is connected to its neighbor(s) by a spring
- Hooke's law: force proportional to distance ($d = \|\mathbf{p} - \mathbf{q}\|$) between the points



Hooke's Law

- Let s be the distance when there is no force

$$\mathbf{f} = -k_s(|\mathbf{d}| - s)\, \mathbf{d}/|\mathbf{d}|$$

k_s is the spring constant

$\mathbf{d}/|\mathbf{d}|$ is a unit vector pointed from $\mathbf{p}$ to $\mathbf{q}$

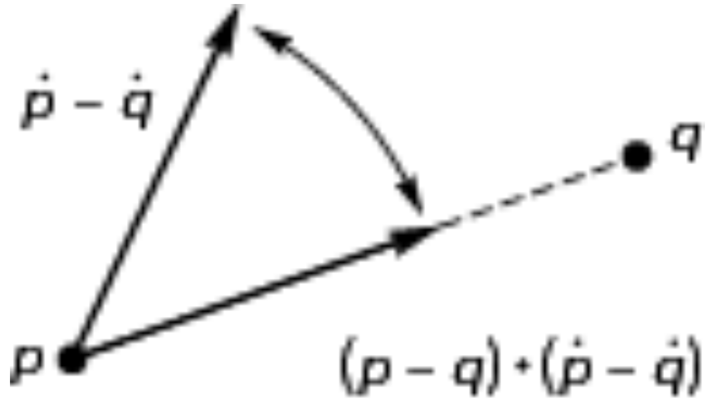
- Each interior point in mesh has four forces applied to it

Spring Damping

- A pure spring-mass will oscillate forever
- Must add a damping term

$$\mathbf{f} = -(k_s(|\mathbf{d}| - s) + k_d\, \mathbf{d}\cdot\mathbf{d}/|\mathbf{d}|)\mathbf{d}/|\mathbf{d}|$$

- Must project velocity



Attraction/Repulsion

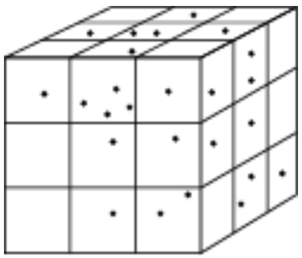
- Inverse square law

$$\mathbf{f} = -k_r \mathbf{d}/|\mathbf{d}|^3$$

- General case requires $O(n^2)$ calculation
- In most problems, the drop off is such that not many particles contribute to the forces on any given particle
- Fast Multipole Algorithm: O(n log n)

Octree Enhancement

- Spatial subdivision technique
- Divide space into boxes
- Particle can only interact with particles in its box or the neighboring boxes
- Must update which box a particle belongs to after each time step

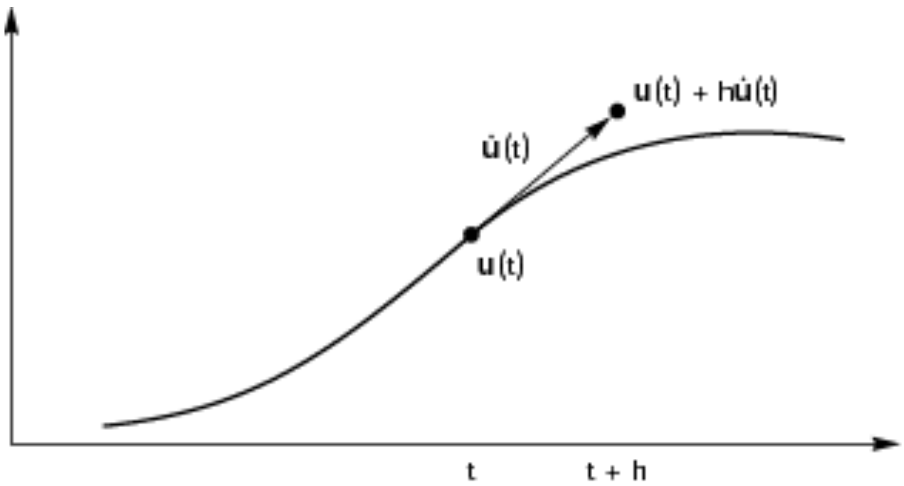


Field Calculations

- Consider simple gravity
- We don't compute forces due to sun, moon, and other large bodies
- Rather we use the gravitational field
- Usually we can group particles into equivalent point masses

Solution of ODEs

- Particle system has 6n ordinary differential equations
- Write set as $d\mathbf{u}/dt = g(\mathbf{u},t)$
- Solve by approximations using Taylor's Thm



Euler's Method

$$\mathbf{u}(t+h) \approx \mathbf{u}(t) + h\, d\mathbf{u}/dt = \mathbf{u}(t) + h\mathbf{g}(\mathbf{u}, t)$$

Per step error is $O(h^2)$

Require one force evaluation per time step

Problem is numerical instability

depends on step size

RUNGE KUTTA

$$\mathbf{u}(t + h) \approx \mathbf{u}(t) + h/2(\mathbf{g}(\mathbf{u}, t) + \mathbf{g}(\mathbf{u}, t{+}h))$$

Per step error is $O(h^3)$

Also allows for larger step sizes

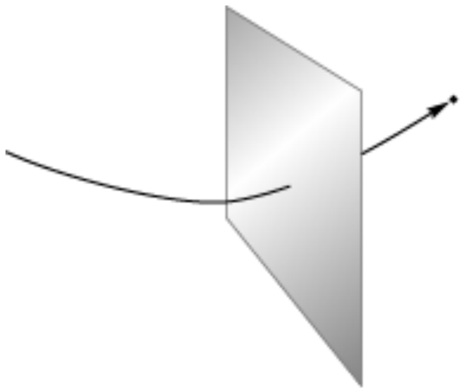
But requires two function evaluations per step

Also known as Runge-Kutta method of order 2

Constraints

- Easy in computer graphics to ignore physical reality
- Surfaces are virtual
- Must detect collisions separately if we want exact solution
- Can approximate with

repulsive forces



Collisions

Once we detect a collision, we can calculate new path

Use coefficient of resititution

Reflect vertical component

May have to use partial time step