Procedural Modeling

Constructs 3D models using algorithms
• for models that are too complex (or tedious) to create manually, e.g.,
  • landscapes, mountains, clouds, planets
  • trees, plants, ecosystems
  • buildings, cities
• usually defined by a small set of data, or rules, that describes the overall properties of the model, e.g.,
  • trees defined by branching properties and leaf shapes
• model is then constructed by an algorithm
• to add variety and realism,
  • often involves fractal geometry
  • often includes randomness
  • e.g., a single tree pattern can be used to model an entire forest

Self-Similarity

Self-similar:
• viewing scale not apparent from object appearance
• object features are statistically similar between object parts and the overall object
  • for example, we always see a jagged line no matter how close we look at a coastline
• infinite details: looks good and natural at every resolution
• achieved using random numbers and fractional dimension

Randomness

Makes models more interesting, natural, and less uniform and “clean”

Due to the discrete and finite nature of computers, we can only generate pseudo-random numbers, based on some initial seed value

• pseudo-random sequences are repeatable, simply by resetting the seed value
  • a different seed value generates a different sequence of pseudo-random numbers
• for repeatability, be careful of dynamic events effecting the use of the pseudo-random sequence
**Fractal Dimension**

Measures the “roughness” of an object.
- More jagged objects have a larger fractal dimension.

Use Hausdorff variant to approximate fractal dimension:
- Subdivide an object into self-similar pieces with a scaling factor $s$.
- Count the number of pieces ($n$) covered by the original object.
- Fractal dimension $d = \frac{\log(n)}{\log(s)}$.
  
  Example: Sierpinski Triangle.
  - Take a triangle.
  - Scale down to $\frac{1}{2}$.
  - Make 3 copies and arrange into a triangle of original size.
  - Repeat forever.
  - $d = \frac{\log(3)}{\log(2)} = 1.58$.

**Procedural Terrain Modeling**

Landslides are often constructed as height fields.

Want: a height function $y = h(x)$.

Random midpoint displacement:
- Start with an initial figure (e.g., line or triangle).
- Split at midpoints and add random displacement.
- Recurse, decreasing the magnitude of displacements by a factor $0 < f < 1$.

1D Example:

**L-Systems**

Developed by A. Lindenmayer, a biologist, in 1968 to model growth patterns of algae and plants.

Grammar-based fractal-like models, a.k.a., “graftals.”

Based on parallel string-rewriting rules:
- Describe an object by a string of symbols.
- And a set of production/rewriting rules.
  - Incorporate notions such as branching, pruning, …
L-Systems Extensions

**Bracketed**: save/restore state (for branches)

**Parametric**: production governs by parameter, e.g., change color at certain level of recursion

**Stochastic**: randomly choose one of several production rules provided for a symbol

**Context sensitive**: production based on neighboring symbols

---

L-Systems

Define a grammar: $G = \{V, S, \omega, P\}$

- $V$: alphabet, set of symbols that can be replaced (variables)
- $S$: set of symbols that remain fixed (constants)
- $\omega$: axiom, string of symbols defining initial state
- $P$: production/rewriting rules

Example:

- $V$: $A, B$
- $S$: $\cdot$
  - $+$: turn left
  - $-$: turn right
  - $\left[\right]$: push state onto stack
  - $\right\left[\right]$: pop state from stack
- $\omega$: $B$
- $P$: $B \rightarrow A [+B] [-B]$

---

Parametric L-System

**Turtle graphics**:

- $F$: draw forward
- $f$: move forward
- $+\cdot$: turn left by $\text{angle}$ // parameter
- $-\cdot$: turn right by $\text{angle}$ // parameter
- $\left[\right]$: push current state onto stack
- $\right\left[\right]$: pop current state from the stack

**Koch's Snowflake**

- $\text{angle}$ $(2\pi) / 6$ // parameter
- axiom $F$
- production rule: $F \rightarrow F + F - F + F$

**Start**: $F$

**Generation 1**: $F + F - F + F$

**Generation 2**: $F + F - F + F + F - F + F + F - F + F - F + F + F - F$

---

Turtle Graphics

**Koch’s Island**

- $\text{angle}$ $(2\pi) / 4$
- axiom $F + F + F + F$
- production rule: $F \rightarrow F + F - F + F + F - F$

**Tree**

- $\text{angle}$ $(2\pi) / 16$
- axiom $++++FS$
- production rule: $S \rightarrow + [FS] - [FS] - [FS]$
L-Systems for Plants

L-systems can capture a large number of plant species, though designing rules for specific species is not easy.

See algorithmicbotany.org/papers/
• a free 200 pages ebook
• covers many variant of L-systems and different plant types

L-System for Cities

Street system:
• start with a single street
• branch and extend with parametric L-system
• parameters: goals and constraints
  • goals control street direction, spacing
  • constraints allow for parks, bridges, road loops

Motion

Keyframing: interpolates motion from “key” positions
+ perfect control
  - can be tedious
  - no realism

Procedural: solves equations to compute motion
+ realistic motion
+ automatic generation
  • once you have the program, you can get lots of motion
  - difficult to control
  - hard to tell a story with purely procedural means
  • mostly used for supporting background/effects
Procedural Animation

Using a process to control or animate some attribute, including shape (modeling), of the object

Steps:
• program some rules for how the system will behave
• choose some initial conditions for the world
• run the program, maybe with user input to guide what happens
• program outputs the position/shape of the scene over time

Physically-based Animation

Kinematic (key framing): describes motion without considering causes leading to motion
• considers only poses

Dynamics (procedural): considers underlying forces
• inverse dynamics:
  • given prescribed motion, what are the forces and torques required?
• forward dynamics:
  • compute motion from initial conditions and physical laws:
    • given forces and torques, what is the motion?

Physically-based Animation

• dynamics: movements
• point mass particle systems
• spring-mass: animating the shape and reaction of cloth
• fluid flow: water waves

Behavioral animation
• bird or fish flocking
• crowd animation

Inverse Dynamics

Given prescribed motion, what are the forces and torques required?

Active character: character has internal source of energy

Animator specifies constraints:
• character’s physical structure
  • e.g., articulated figure
• character’s task
  • e.g., jump from here to there in time t
• other physical structures present
  • e.g., floor to push off and land
• motion requirements
  • e.g., minimize energy
Inverse Dynamics

Computer solves for the “best” physical motion satisfying constraints

Example: object with jet propulsion
- \( x(t) \): position of object at time \( t \)
- \( f(t) \): propulsion force at time \( t \)
- equation of motion: \( m \ddot{x} - f - mg = 0 \)
- task: move from \( a \) to \( b \) from \( t_0 \) to \( t_1 \) with minimum jet fuel,
  minimize \( \int_{t_0}^{t_1} |f(t)| \, dt \) with \( x(t_0) = a \) and \( x(t_1) = b \)
- solve with iterative optimization method

Controlling Motion

E.g., with state machines

Challenges:
- specifying constraints and objective functions
- avoiding local minima during optimization
- retargeting motion to new characters
- need a larger variety of motions (not just well-defined sport motions)
- real-time performance

Inverse Dynamics

Advantages:
- animators don’t have to specify details of physically realistic motion with spline curves
- easy to vary motions due to new parameters and/or new constraints

For example, adapting motion:

original jump
heavier base

Controlling Motion

E.g., with state machines

Challenges:
- specifying constraints and objective functions
- avoiding local minima during optimization
- retargeting motion to new characters
- need a larger variety of motions (not just well-defined sport motions)
- real-time performance
Forward Dynamics

Given initial conditions and physical laws, what is the motion?
• initial conditions: mass, forces, torques
• apply physical laws, e.g., Newton’s laws, Hook’s law, etc.
• simulate physical phenomena:
  • gravity, momentum (inertia), friction, collisions, elasticity, solidity, flexibility, fracture, explosion, fluid flow, aerodynamics (drag/viscosity, turbulence)
• motion computed using numerical simulation methods
  • particle systems
  • rigid bodies
  • soft-objects (spring-mass)
  • fluid

How to Render Fire?

Texture mapping polygons is fast and acceptable for short-lived effects
• overlay a flame point with a series of textured polygons
• for enhanced realism, we could introduce secondary light sources at the fire

Problems:
• hard to sustain for long periods
• hard to spread or change shape
• no translucency

Particle Systems

Approximate flame, and other amorphous objects, by a discrete set of small particles

Single particles are very simple, just point mass with attributes:
• mass, position
• velocity, forces
• color, temperature, shape
• lifetime

Large groups can produce interesting effects:
• rockets: fireworks
• clouds: water drops

Particle Systems

For each frame:
• create new particles according to a probability distribution and assign attributes
• delete any expired particles
• update particles based on attributes and physics
  • numerical solutions to ODE
• render particles: motion blur, compositing

Where to create particles?
• predefined sources
• surface of shape
• where particle density is low, etc.
Particle Systems

When to delete particles?
• predefined sink
• surface of shape
• where density is high
• lifetime
• random

Example: water
• new particles created each frame
  • the number created per frame is normally distributed
• each particle has initial downward velocity
  • again, normally distributed
• on each successive frame, each particle is acted on by “wind” force to the right

Impact

Dynamic:
• position (x)
• velocity (v)
• rotation?

Newtonian particles: \( \mathbf{F} = m \mathbf{a} \)
• \( \mathbf{F} \) and \( \mathbf{a} \) are vectors
• given \( \mathbf{F} \) and position at time \( t \), \( x(t) \), how does the position change to \( x(t+1) \)?

Differential Equations

Differential equations describe the relation between an unknown function and its derivatives

Solving a differential equation means finding a function that satisfies the relation, plus some additional constraints

Ordinary Differential Equation (ODE):
• “ordinary”: function of one variable
• partial differential equation (PDE): more variables

Discrete time: \( x(t+1) = f(x(t),t) \)

Continuous time: \( \frac{dx(t)}{dt} = f(x(t),t) \)

Computing particle motion requires solving a 2nd-order differential equation: \( \ddot{x} = \frac{\mathbf{F}(x,x,t)}{m} \)

Instead, add variable \( v \) to form coupled 1st-order differential equations: \( \dot{v} = \dot{x} \), \( \dot{x} = \frac{\mathbf{F}(x,v,t)}{m} \)

For animation, want a series of values \( x(t_i), i = 0, 1, 2, \ldots \)
• samples of the continuous function \( x(t) \)
Path Through a Field

\( f(x, t) \) is a vector field defined everywhere
• e.g., a velocity field

- it may change based on \( t \)
- \( x(t) \) is a path through the field: \( x(t) = x_0 + \int_{t_0}^{t} f(x,t) dt \)

- usually no analytical solution

Solving the Integration

Since we’re working in fixed-frame intervals, we can use a simple approximation (Euler’s method):

- define step size \( h \)
- given \( x_0 = x(t_0) \), take step:
  \( t_1 = t_0 + h \)
  \( x(t_1) = x_0 + h f(x_0, t_0) \)

- piecewise-linear approximation of the curve
- step size controls accuracy: smaller, more accurate
- may need to take many small steps per frame

Solving the Integration

Euler’s method is inaccurate

To move along a circle:

\[
\begin{align*}
  x(t) &= \begin{pmatrix} r \cos(t + h) \\ r \sin(t + h) \end{pmatrix} \\
  \dot{x} &= f(x, t) = \begin{pmatrix} -y \\ x \end{pmatrix}
\end{align*}
\]

Moving along tangent, e.g.,

Euler’s method \( x(t_i) = x_{i-1} + h f(x_{i-1}, t_{i-1}) \) spirals outward, can leave curve, no matter how small \( h \) is
- smaller \( h \) will just diverge more slowly

Consequences: Euler’s method is inaccurate and unstable
Euler’s Method: Unstable

Consider the following system: \( \dot{x} = -kx, \; x(0) = 1 \)
Exact solution is decaying exponential: \( x = x_0 e^{-kt} \)
Limited step size: \( x_i = x_{i-1} - h(kx_{i-1}) = (1-hk)x_{i-1} \)
If \( k \) is big, \( h \) must be small

\[
\begin{align*}
 h \leq 1/k, \text{ ok} \\
h > 1/k, \text{ oscillates } \pm \\
h > 2/k, \text{ explodes }
\end{align*}
\]

Solving the Integration

How can we improve upon Euler’s method?

By adding another term from the Taylor’s expansion of \( x(t) \):

\[
x(t + h) = x(t) + h \frac{dx}{dt} + \frac{h^2}{2} \frac{d^2x}{dt^2} + \cdots
\]

Consider two alternatives:
- the Trapezoid method
- the Midpoint method
both have errors on the order of \( O(h^3) \)

Trapezoid Method

\[
x(t_i) = x_{i-1} + h f(x_{i-1}, t_{i-1})
\]

Problem: approximated \( f \) varies from actual \( f \)
Idea: consider \( f \) at the arrival of the step and compensate for variation

Let \( f_0 = f(x_0, t_0) \)
\( f_1 = f(x_0 + hf_0, t_0 + h) \)

Then \( a = hf_0, \; b = hf_1, \)
\[
x(t_0 + h) = x_0 + (a + b)/2 + O(h^3)
\]

This is the trapezoid method, a.k.a. improved Euler’s method

Midpoint Method

\[
x(t_i) = x_{i-1} + h f(x_{i-1}, t_{i-1})
\]

Problem: approximated \( f \) varies from actual \( f \)
Idea: consider \( f \) at half step and compensate for variation

Choose \( \Delta x = h/2 f(x_0, t_0) \)
then rearrange as before,
let \( f_0 = f(x_0, t_0) \)
\( f_{mid} = f(x_0 + h/2 f_0, t_0, t_0 + h/2) \)
then \( x(t_0 + h) = x_0 + hf_{mid} + O(h^3) \)

This is the midpoint method, a.k.a., 2nd-order Runge-Kutta
Comparison

Midpoint:
• ½ Euler step
• evaluate \( f_{mid} \)
• compute step using \( f_{mid} \)

Trapezoid:
• Euler step (a)
• evaluate \( f_1 \)
• compute step using \( f_1 \)
• average (a) and (b)

Not exactly the same result, but same order of accuracy

Higher-Order Runge-Kutta

4\(^{th}\) order:
\[
k_1 = hf(x_0, t_0)
\]
\[
k_2 = hf(x_0 + \frac{k_1}{2}, t_0 + \frac{h}{2})
\]
\[
k_3 = hf(x_0 + \frac{k_2}{2}, t_0 + \frac{h}{2})
\]
\[
k_4 = hf(x_0 + k_3, t_0 + h)
\]
\[
x(t_0 + h) = x_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 + O(h^5)
\]
popular because orders higher than 4\(^{th}\) need more stages than orders to compute

<table>
<thead>
<tr>
<th>order</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>stages</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>11</td>
</tr>
</tbody>
</table>

Collisions

Collisions with environment, other particles
• detection: potential large number needed, use hierarchical bounding volumes
• response: particle shape and size needed
• due to temporal aliasing, sub-frame calculation required so as not to miss collision time

\[
v_n = (v \cdot n)n
\]
\[
v_i = v - v_n
\]
\[
v^{'} = v_i - k_r v_n
\]
k\(_r\): coefficient of restitution

May the Force . . .

Forces acting on the particle system is a sum of a number of things

Force fields:
• gravity: constant downward force proportional to mass
  \( f = -mg \),
  \( g \): gravitational constant, 9.78 m/sec\(^2\) on Earth
• other particles: particles mutually attract/repell
  • attractive force: \( f = Gm_1m_2 \frac{d}{||d||} \)
  • repulsive force: \( f = -k_1 \frac{d}{||d||} \)
  • beware \( O(n^2) \) complexity!
Missed Collisions

Want to detect collision when \((x-p) \cdot n < \varepsilon, \varepsilon \geq 0\)

Often collision detected when \(v \cdot n < 0\)

Solution: backtrack
- compute intersection point
- ray-object intersection!
- compute response there
- reconstruct for remaining fractional time step

Other solution/hack:
- project to surface point closest to object

Particle Systems Rendering

Simple rendering:
- project particles to view frame
- blend particle projection with framebuffer content (translucency)

Particle color:
- make color a function of age
- make color a function of temperature
- requires other ODEs to govern these properties

Smoke Particle System

Constantly create particles

Particles move upwards, with Perlin turbulence added

Draw them as partially transparent circles that fade over time

Quake

Pre-render bitmap of fireball
Real-time rendered glow
Animated glowing particles
Trees in *Andre and Wally B.*

Lots of trees

Each tree branches recursively

**Leaves as particles**
- millions of particles
- need a simple model for shading each

---

**Spring-Mass Systems**

Model objects as systems of springs and masses

The springs exert forces, controlled by changing their rest length

A reasonable, but simple, physical model for muscles

Advantage: good looking motion when it works

Disadvantage: expensive and hard to control

---

**Shading Model for Trees**

- **Ambient**
  - dependent on how deep in the tree $d_a$
  - independent of light position

- **Diffuse**
  - dependent on distance $d_d$ inside the tree, in the direction of light

- **Shadowed**
  - if below plane, only ambient used

---

**Cloth**

Many types of cloth
- very different properties
- not a simple elastic surface
- woven fabrics tend to be very stiff
- anisotropic

Resolution of mesh is critical

Computation of collisions is expensive

---

*Chenney*
Challenges for Physically-Based Animation

Expensive and not necessarily realistic

What pieces of physics are necessary for appearance?

How to give animator control?
  • how to give artists and directors the results they want?

Perceptual Hacks

Viewers can be pretty oblivious to things like incorrect bounces and can’t exactly predict how things break

Just make sure objects don’t go through walls

Shift emphasis from physical accuracy to fast-and-looks-good

Behavioral Animation

Define rules for the way an object behaves and interacts
  • programs implement the rules
  • objects respond to their changing environment

Classic example: “boids” (Craig Reynolds)
  • emergent behavior: flocking
  • really just a particle system with a bit of perception and a bit of brain power

Flocking

Each boid perceives neighbors in a neighborhood

Each boid’s behavior is a simple function of nearby boids:
  • separation force
    • steer to avoid crowding local team mates, keep minimum distance
  • alignment force
    • align velocities
  • cohesion force
    • move towards average position of neighbors

http://www.red3d.com/cwr/boids/
http://www.riversoftavg.com/downloads.htm
Animation Summary (brief)

<table>
<thead>
<tr>
<th>Technique</th>
<th>Control</th>
<th>Time to Create</th>
<th>Computation Cost</th>
<th>Interactivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-Framed</td>
<td>Excellent</td>
<td>Poor</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Motion Capture</td>
<td>Good at time of creation, after that poor</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Procedural</td>
<td>Poor</td>
<td>Poor to create program</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

How Are Movies Animated?

Keyframing mostly
Articulated figures, inverse kinematics

Skinning
• complex deformable skin, muscle, skin motion

Hierarchical controls
• smile control, eye blinking, etc.
• keyframes for these higher-level controls
A huge time is spent building the 3D models, its skeleton and its controls

Mixing Techniques

Apply physical simulation of secondary motion on top of key-framed primary motion
• particularly appropriate for cloth, hair, water
• use particle systems for “fuzzy” objects

Mix motion-capture and physics:
• motion-captured person kicks a ball which is then physically simulated for trajectory