Introduction to Virtual Environments - Spring 2004 - Wernert/Arns

Lecture 4.2 – Animation & Interaction

Outline
1. Animation Techniques & Concepts
2. Input & Event Concepts
3. Implementation Details

0. Resources

1. Animation Techniques & Concepts

1.1 Concepts
- What can be animated? Since we're redrawing the scene every frame, just about anything!
  - objects: position, orientation, scale (uniform & non-uniform)
  - shapes: vertices, normals, colors, textures
  - cameras: position, orientation, FOV
  - lights: color, intensity, position, etc.
  - background
- Important principles from traditional animation (from John Lasseter)
  - squash and stretch
  - timing
  - anticipation
  - slow in and out
  - arcs
  - secondary action
  - scene staging (camera location)
  - appeal
  - exaggeration (cartoon physics!)

 FIGURE 2: Squash & stretch in bouncing ball. (image from Lasseter paper)
### 1.2 Animation Methods

- **Flipbook Method**
  - Keyframing (Implicit Motion specification)
  - Explicit Motion Specification
  - Procedural (including scripting)
  - Simulation (Constraint-based systems, physics engine)
  - Shape Morphing
  - Hierarchical Animation

**FlipBook Animation**
- cycle through a series of 3D models or representations
- useful for some visualizations and problems that defy other techniques
- VRML Switch node is useful for this method
- **pros**: simple; generalizes easily
- **cons**: tedious; large storage requirements; loading time and performance issues

**Keyframe Animation**
- grew out of traditional cell animation
- animation tied to a timer: local (triggered) or global timer
- specify animated parameters (frames) at specific (key) times
- interpolate values at intermediate times
  - linear interpolation
  - cubic interpolation: Catmull-Rom spline
  - cubic approximation
- VRML97 has explicit support for keyframe animation using combinations of TimeSensor and Interpolator nodes
  - **pros**: conceptually simple
  - **cons**: continuity and smoothness issues; unintended effects, especially with rotations; linear, pre-programmed behavior only

**Explicit Motion Specification**
- extension of keyframe animation concept
- explicitly edit the parameter curves in a 2D or 3D window
more explicit control of ease-in/ease-out parameters
for graphical VRML editors (e.g. Cosmoworlds), curves may be exported as piecewise approximations for the linear interpolators using lots of time samples.

**pros**: better continuity control
**cons**: still linear; limited user interaction

---

**Procedural Animation**

- control animated parameters with a piece of code or script description
- examples from Dave Pape's Ygdrasil scripting environment (see http://www.evl.uic.edu/yg/)

```java
mover mover1 (startPosition(0 0 0), endPosition(0 10 0), endOrientation(0 0 180))
pointFollower mover2 (file(motionPath.txt))
timer moverTimer (duration(5), when(changed, mover1.set($value)))
```

- VRML supports procedural animation implicitly through the general **Script** node
- **pros**: easier to modify at runtime; can incorporate user input on-the-fly
- **cons**: requires programming-like skills and careful attention to syntax; no good visual methods
  - possible exception: Alice 3D from CMU (www.alice.org)
Simulation-based systems

- **Physics Engine**
  - specify the physical properties of the object: shape, mass, velocity, etc.
  - system computes collisions and interactions between objects and updates positions accordingly

- **Constraint-based System**
  - specify constraints - e.g. fixed length or radius, fixed points, fixed angle, collocated points, point on a line, etc.
  - user (or system) updates an independent component, the system resolves the locations and orientations of the dependent components
  - **pros**: simple and powerful for developers
  - **cons**: difficult to develop initial system; constraint specification may be awkward; may not scale well (simulation or constraint satisfaction computation may overwhelm computational resources of the system)
  - example: Geometer's Sketchpad

Shape Morphing

- Interpolate vertex positions between two or more models
- create soft or deformable objects
- usually requires all models to have the same number of vertices and the same connectivity
- **pros**: effective (if it works for your model)
**cons**: may be difficult to maintain continuity (e.g., vertices, connectivity) between models.

Hierarchical Modeling & Animation

- Scene Graph concepts
Scene Graph in VRML Editor

Scene Graph (Simulation Tree) in EON
**Forward Kinematics**
- move parent node and children move with it

**Inverse Kinematics**
- move child node to final location, compute appropriate values for any/all ancestor nodes

**Pros**: powerful, logical way to work with mechanical systems

**Cons**: difficult to get convincing human motion

**Extensive example**: H-anim standard (http://www.h-anim.org/)
- only supports forward kinematics

2. Input and Event Concepts

Physical Input Devices

- mouse, keyboard, trackers, etc.

Conceptual Input Devices:

- buttons: discrete (often binary) state
- valuators: continuous range of values
  - may be normalized (e.g., [-1, 1]) or bounded (e.g., rotation \([0, 2\pi])\)
  - 1 DOF: sliders & dials
  - 2 DOF: mouse position, tablet, thumbpad, wand joystick
  - 3 DOF: position only tracking, orientation only tracking
  - 6 DOF: position & orientation tracking

Capturing Input

- Event Model - register types of events you are interested in (and specify resulting action)
  - e.g., mouseButton event in GLUT programming
  - pros: events may be buffered, won’t be missed; more object-oriented; mimics many desktop programming paradigms
  - cons: buffered events may overload system; may only care about the most recent event

- Sampling/Querying Model - explicitly sample the state of the device
  - e.g., VR library will read head position/orientation at the start of each frame to update OpenGL view frustum
  - pros: explicit control over when & how device is read; better for real-time interaction in VR
  - cons: events can be missed (e.g., button clicks); more programming work
Event Types & Triggering

- **Time-based Events** - linked to a global clock
  - Timers & Alarms
  - Background Audio
  - MovieTextures
- **Viewpoint-based (Environmental) Events** - triggered by changes in the viewpoint
  - Collision Detection
  - Proximity Sensor - when user is within certain distance
  - Visibility Sensor - when object falls within the view frustum
  - Billboards
  - Positional Sound
- **Pointing Events** - triggered by touching/pointing/clicking with mouse or wand
  - Touch Sensor
  - Anchor (WWW Link)
- **Interaction Events (Drag Sensors)** - triggered by point/click/hold/drag with mouse or wand
  - Cylinder, Sphere, Plane sensors in VRML
  - "Grabber" sensor in VR APIs

Event Execution & Propogation Issues

- **timing** - when will result of the event happen?
  - once per frame
  - before the start of the next frame
  - as soon as possible
- **order of execution**
  - is any order implied or explicitly specified?
  - VRML does not guarantee any order of execution
- **fan-out and fan-in**
  - fan-out: one event can trigger multiple actions
  - fan-in: multiple events can trigger the same action
  - both are permitted in VRML
- **event loops**
  - may be permitted, but require some termination rule to avoid infinite loops
    - e.g., each node responds to event once per time stamp

3. Implementation Details

Node Networks

- Some nodes have pre-defined behaviors
  - Anchor
  - Billboard
  - Collision

- most other "action" nodes must be explicitly linked (routed) to create desired behavior
  - routing may be accomplished visually or textually
Node Names

- nodes in a scene graph or network can be given names to uniquely identify them
- In VRML, the DEF mechanism provides naming for instancing and other referrals.
  ```
  DEF Unit_Sphere Shape {
    appearance Appearance {
      geometry Sphere {radius 1.0}
    }
  } Transform {
    translation 10 10 10
    children
      USE Unit_Sphere
    }
  Transform {
    translation 20 20 20
    children
      USE Unit_Sphere
  }
  ```

Node Fields

- **Field Types**
  - systems usually enforce strict type checking
  - common types for VR:
    - Vec3f - position, vertex, normal
    - Color
    - Rotation
    - Image
    - Node
    - String
    - Time
    - Float, Int, Boolean,
  - In VRML, SF/MF distinction for single-valued field and multi-valued field
    - in a Transform node: translation is an SFVec3f
    - in a Coordinate3 node: point is an MFVec3f
- **Field Access**
  - categories:
private to node  
public to other nodes  
change notification  

In VRML  
field = private  
eventOut = if field value changes, it propagates values to any connected nodes  
eventIn = field whose value may be set by incoming connection  
exposedField = public field; may be accessed through eventIns/Outs  

Routing  
method of linking outputs to inputs to achieve desired behavior  
types should match  
connections may fan-in or fan-out  
loops may be permitted (but require some termination rule)  
In VRML  
referring to fields: <node name>.<field name>  
node name defined by DEF  
route syntax: ROUTE <eventOut> TO <eventIn>  
E.g.: ROUTE MyTimer.fraction_changed TO MyPositionInterp.set_fraction  

Interpolation  
general syntax for interpolation Nodes in VRML  
eventIn SFFloat set_fraction  
exposedField MFFloat key [...]  
exposedField MF<type> keyValue [...]  
eventOut [S|M]F<type> value_changed  

example: a rotor for a cube...  
DEF MyRotor OrientationInterpolator {  
  key [0.0, 0.5, 1.0]  
  keyValue [ 0 1 0 0.0,  
               0 1 0 3.14,  
               0 1 0 6.28 ]  
}  
ROUTE MyTimer.fraction_changed TO MyRotor.set_fraction  
ROUTE MyRotor.value_changed TO MyCubeXform.set_rotation  

VRML supports interpolators for:  
  Color  
  Coordinate  
  Normal  
  Orientation  
  Position  
  Scalar  

Scripts  
generalized programming nodes  
In VRML, may script in Java, JavaScript, or ECMAScript/VRMLScript  
In EON, may script in Visual Basic  
Script nodes may define an arbitrary number and type of input (eventIn), output (eventOut) fields  
It's the only way to create a global variable in VRML  
scripts may create and break ROUTEs at runtime  
you must have some method for triggering the script  
lots of details on how to initialize scripts, etc.
The additions of scripting and node prototyping features make VRML a powerful and expressive tool (however, it's not terribly scalable)