ANIMATION TECHNIQUES

1. Computer-assistant animation introduces changes to the virtual world.
   - Animation covers all changes that have a visual effect.
     - *motion dynamics*: changes in positions.
     - *update dynamics*: changes in shape, color, transparency, structure, and texture.
     - *changes in lighting, camera position, orientation, focus, and maybe rendering technique.*
   - Controlling such a wide range of attributes has resulted in a plethora of techniques:
     - Develop algorithms to generate *curve*s and special techniques for forming one seamless curve from a set

of jointed short curve segments.

     - Study issues of *color* and *realism*, and tools for assigning objects with a wide range of *physical attributes* in the form of color and texture.
     - Spend considerable *time* on illuminating the imaginary world, e.g., to decide where each light source is to be positioned, its intensity of color.

2. Linear interpolation
   - Given the values, v_s and v_e, of some attribute in the starting and ending points, a value v_t in between is determined by

$$v_t = (1-t)\,v_s + t\,v_e$$

When t changes from 0 to 1, v_t varies smoothly from v_s

to v_e.

- This form of interpolation can be applied to any pair of quantities, whether they be colors, speeds, positions, angles, or coordinates.

3. Non-Linear interpolation

- The sine function can be used for blending one number into another, for example, to interpolate between v_s and v_e such that the initial rate is fast, but slows down until at the end of a sequence.

$$v_t = (1 - \sin(\theta))\, v_s + t\, v_e,\ 0° \leq \theta \leq 90°$$

- The square-law relationship does the inverse of the sine function, i.e., slow starting and fast ending.

$$\begin{aligned} v_t &= (1 - t^2)\, v_s + t^2\, v_e \\ &= v_s + (v_e - v_s)\, t^2 \end{aligned}$$

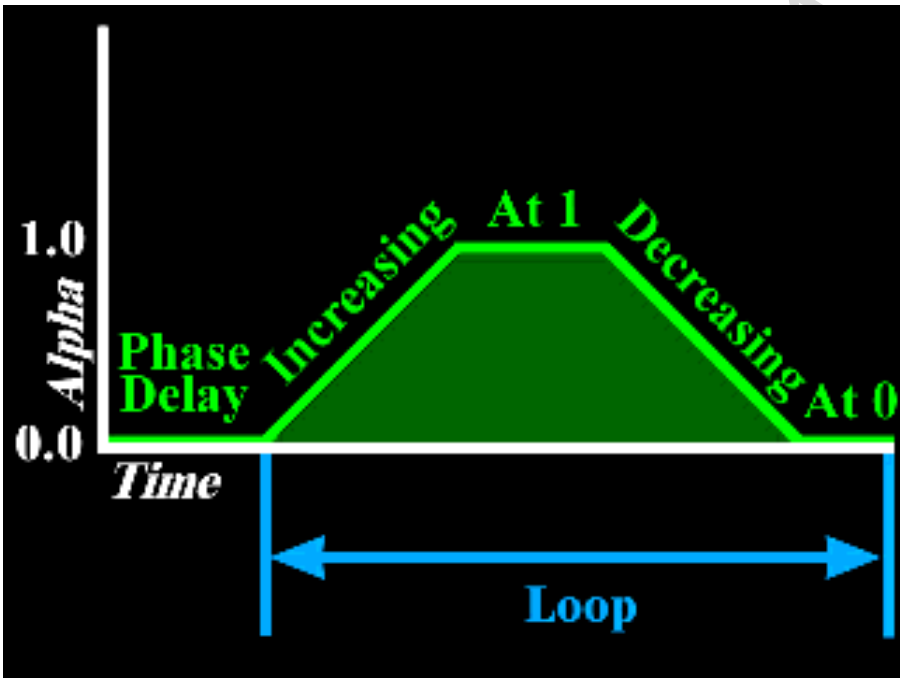
- Splines can be used for slow-in and slow-out.

$$v_t = (1 - f(t))\, v_s + f(t)\, v_e$$

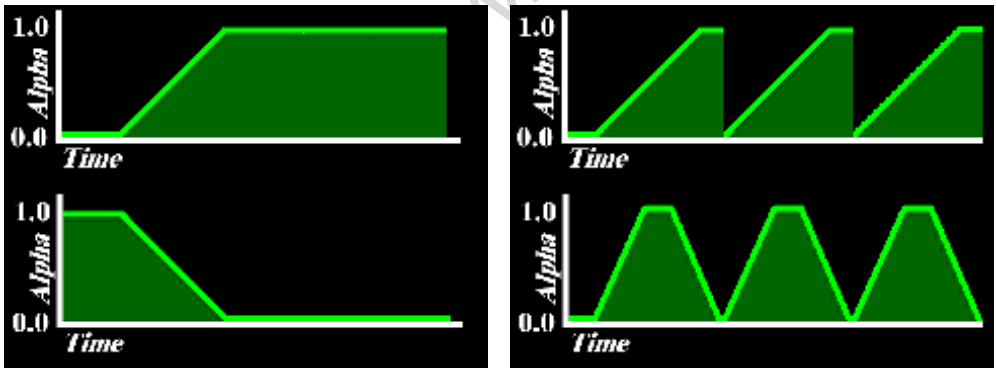
where $f(t)$ is a mathematical formulae that is essential for controlling changes that have a precise regularity or a well-defined form.

4. Since many simple behaviors can be expressed as interpolators, Java 3D provides *interpolator* behaviors
   - It allows that change of a parameter from a starting to an ending value during a time interval
     - Transforms, colors, switches
5. An interpolator uses two mappings:
   - Time-to-*Alpha*
     - *Alpha* is a generalized value that varies from 0.0 to 1.0 over a time interval
   - Alpha-to-*Value*
     - Different interpolator types map to different values, such as transforms, colors, switches

6. Mapping time to alpha
   - Class `Alpha` extends `Object`
     - `Alpha` methods construct and control alpha start and looping, or get the current value
     - `Alpha` methods also set stage parameters
   - An *Alpha generator* computes alpha using:
     - Trigger time
     - *Phase Delay* before initial alpha change
     - *Increasing time* for increasing alpha
     - *At-One* time for constant high alpha
     - *Decreasing* time for decreasing alpha
     - *At-Zero* time for constant low alpha



- Increasing and decreasing phases may be individually enabled or disabled and their acceleration controlled
  - *Increasing ramp* controls increasing acceleration
  - *Decreasing ramp* controls decreasing acceleration
- This model of alpha generalizes to several different types of one-shot and cyclic behaviors



7. Types of interpolators

- Simple interpolators map alpha to a value between start and end values
  - Single transforms
    - `PositionInterpolator`, `RotationInterpolator`, and `ScaleInterpolator`
  - Colors and transparency
    - `ColorInterpolator` and `TransparencyInterpolator`
  - Switch group values
    - `SwitchValueInterpolator`
- *Path* interpolators map alpha to a value along a path of two or more values
  - Single transforms

    - `PositionPathInterpolator` and `RotationPathInterpolator`
  - Combined transforms
    - `RotPosPathInterpolator` and `RotPosScalePathInterpolator`
- All interpolators specify a *target* into which to write new values
  - Transform interpolators use a `TransformGroup` target
  - A `ColorInterpolator` uses a `Material` target
  - A `TransparencyInterpolator` uses a `TransparencyAttributes` target
  - A `SwitchValueInterpolator` uses a `Switch` target
  - And so forth

- `Interpolator` class hierarchy

Class Hierarchy

```
java.lang.Object
└ javax.media.j3d.SceneGraphObject
  └ javax.media.j3d.Node
    └ javax.media.j3d.Leaf
      └ javax.media.j3d.Behavior
        └ javax.media.j3d.Interpolator
          ├ javax.media.j3d.ColorInterpolator
          ├ javax.media.j3d.PathInterpolator
          │ ├ javax.media.j3d.PositionPathInterpolator
          │ ├ javax.media.j3d.RotationPathInterpolator
          │ ├ javax.media.j3d.RotPosPathInterpolator
          │ └ javax.media.j3d.RotPosScalePathInterpolat
          ├ javax.media.j3d.PositionInterpolator
          ├ javax.media.j3d.RotationInterpolator
          ├ javax.media.j3d.ScaleInterpolator
          ├ javax.media.j3d.SwitchValueInterpolator
          └ javax.media.j3d.TransparencyInterpolator
```


8. RotationInterpolator example code

- Create a TransformGroup to animate

```
TransformGroup myGroup = new TransformGroup( );
```

- Create an alpha generator

```
Alpha upRamp = new Alpha( );
upRamp.setIncreasingAlphaDuration( 10000 );
upRamp.setLoopCount( -1 ); // loop forever
```

- Create and set up a rotation interpolator

```
RotationInterpolator mySpinner= new RotationInterpola();
mySpinner.setAxisOfRotation( new Transform3D( ) );
mySpinner.setMinimumAngle( 0.0f );
mySpinner.setMaximumAngle( (float)(Math.PI * 2.0) );
```

- Set the scheduling bounds and add it to the scene

```
mySpinner.setSchedulingBounds( bounds );
myGroup.addChild( spinner );
```