Foundation ActionScript Animation: Making Things Move!
AdvancED ActionScript 3.0 Animation
NUMERICAL INTEGRATION
An object is launched directly upward at 64 feet per second from a platform 80 feet high. What will be the object’s maximum height? When will it attain this height? When will it hit the ground?
Quadratic formula

\* s(t) = -16t^2 + v_0t + h_0
EULER INTEGRATION
Leonhard Euler

Smart old guy

(Oiler not You-ler)
Euler Integration

* Add acceleration to velocity
* Add velocity to position
* Repeat
So what else is there?

- Euler
- Backwards Euler
- Semi-implicit Euler
- Verlet
- Velocity Verlet
- Beeman’s Algorithm
- Midpoint Method
- Heun’s Method
- Newmark-beta Method
- Leapfrog Integration
- Monte Carlo Integration
- Runge-Kutta Methods
So what else is there?

* Runge-Kutta Methods
RUNGE-KUTTA

Wicked Smart Old Guys

Carl Runge

Wilhelm Kutta
People who did NOT invent a numerical integration method

Mark Zuckerberg

Grant Skinner

Nicolas Cannasse (not yet)
RUNGE-KUTTA SECOND ORDER INTEGRATION
(or just RK2 if you are cool)
// pos1 is current position of object
// vell1 is current velocity of object
accel1 = acceleration(pos1, vell1)

pos2 = pos1 + vell1 * time
vel2 = vell1 + accel1 * time
accel2 = acceleration(pos2, vel2)
pos1 += (vell1 + vel2) / 2 * time
vell1 += (accel1 + accel2) / 2 * time
// pos1 is current position of object
// vel1 is current velocity of object
accell1 = acceleration(pos1, vel1)

pos2 = pos1 + vel1 * time
vel2 = vel1 + accell1 * time
accell2 = acceleration(pos2, vel2)
pos1 += (vel1 + vel2) / 2 * time
vel1 += (accell1 + accell2) / 2 * time
// pos1 is current position of object
// vel1 is current velocity of object
accel1 = acceleration(pos1, vel1)

pos2 = pos1 + vel1 * time
vel2 = vel1 + accel1 * time
accel2 = acceleration(pos2, vel2)
pos1 += (vel1 + vel2) / 2 * time
vel1 += (accel1 + accel2) / 2 * time
// pos1 is current position of object
// vel1 is current velocity of object
accel1 = acceleration(pos1, vel1)

pos2 = pos1 + vel1 * time
vel2 = vel1 + accel1 * time
accel2 = acceleration(pos2, vel2)

pos1 += (vel1 + vel2) / 2 * time
vel1 += (accel1 + accel2) / 2 * time
// pos1 is current position of object
// vel1 is current velocity of object
accel1 = acceleration(pos1, vel1)

pos2 = pos1 + vel1 * time
vel2 = vel1 + accel1 * time
accel2 = acceleration(pos2, vel2)
pos1 += (vel1 + vel2) / 2 * time
vel1 += (accel1 + accel2) / 2 * time
var time:int = getTimer();
var elapsed:Number = (time - _oldTime) / 1000;
_oldTime = time;

var accell1:Point = acceleration(_position, _velocity);

var position2:Point = new Point();
position2.x = _position.x + _velocity.x * elapsed;
position2.y = _position.y + _velocity.y * elapsed;

var velocity2:Point = new Point();
velocity2.x = _velocity.x + accell1.x * elapsed;
velocity2.y = _velocity.y + accell1.x * elapsed;

var accell2:Point = acceleration(position2, velocity2);

_position.x += (_velocity.x + velocity2.x) / 2 * elapsed;
_position.y += (_velocity.y + velocity2.y) / 2 * elapsed;

_velocity.x += (accell1.x + accell2.x) / 2 * elapsed;
_velocity.y += (accell1.y + accell2.y) / 2 * elapsed;
var time:int = getTimer();
var elapsed:Number = (time - _oldTime) / 1000;
_oldTime = time;

var accell1:Point = acceleration(_position, _velocity);

var position2:Point = new Point();
position2.x = _position.x + _velocity.x * elapsed;
position2.y = _position.y + _velocity.y * elapsed;

var velocity2:Point = new Point();
velocity2.x = _velocity.x + accell1.x * elapsed;
velocity2.y = _velocity.y + accell1.x * elapsed;

var accel2:Point = acceleration(position2, velocity2);

_position.x += (_velocity.x + velocity2.x) / 2 * elapsed;
_position.y += (_velocity.y + velocity2.y) / 2 * elapsed;

_velocity.x += (accel1.x + accel2.x) / 2 * elapsed;
_velocity.y += (accel1.y + accel2.y) / 2 * elapsed;
```javascript
var time:int = getTimer();
var elapsed:Number = (time - _oldTime) / 1000;
_oldTime = time;

var accel1:Point = acceleration(_position, _velocity);

var position2:Point = new Point();
position2.x = _position.x + _velocity.x * elapsed;
position2.y = _position.y + _velocity.y * elapsed;

var velocity2:Point = new Point();
velocity2.x = _velocity.x + accel1.x * elapsed;
velocity2.y = _velocity.y + accel1.x * elapsed;

var accel2:Point = acceleration(position2, velocity2);

_position.x += (_velocity.x + velocity2.x) / 2 * elapsed;
_position.y += (_velocity.y + velocity2.y) / 2 * elapsed;

_velocity.x += (accel1.x + accel2.x) / 2 * elapsed;
_velocity.y += (accel1.y + accel2.y) / 2 * elapsed;
```
var time:int = getTimer();
var elapsed:Number = (time - _oldTime) / 1000;
_oldTime = time;

var accell1:Point = acceleration(_position, _velocity);

var position2:Point = new Point();
position2.x = _position.x + _velocity.x * elapsed;
position2.y = _position.y + _velocity.y * elapsed;

var velocity2:Point = new Point();
velocity2.x = _velocity.x + accell1.x * elapsed;
velocity2.y = _velocity.y + accell1.x * elapsed;

var accel2:Point = acceleration(position2, velocity2);

_position.x += (_velocity.x + velocity2.x) / 2 * elapsed;
_position.y += (_velocity.y + velocity2.y) / 2 * elapsed;

_velocity.x += (acce11.x + accel2.x) / 2 * elapsed;
_velocity.y += (acce11.y + accel2.y) / 2 * elapsed;
var time:int = getTimer();
var elapsed:Number = (time - _oldTime) / 1000;
_oldTime = time;

var accel1:Point = acceleration(_position, _velocity);

var position2:Point = new Point();
position2.x = _position.x + _velocity.x * elapsed;
position2.y = _position.y + _velocity.y * elapsed;

var velocity2:Point = new Point();
velocity2.x = _velocity.x + accel1.x * elapsed;
velocity2.y = _velocity.y + accel1.x * elapsed;

var accel2:Point = acceleration(position2, velocity2);

_position.x += (_velocity.x + velocity2.x) / 2 * elapsed;
_position.y += (_velocity.y + velocity2.y) / 2 * elapsed;

_velocity.x += (accel1.x + accel2.x) / 2 * elapsed;
_velocity.y += (accel1.y + accel2.y) / 2 * elapsed;
var time:int = getTimer();
var elapsed:Number = (time - _oldTime) / 1000;
_oldTime = time;

var accel1:Point = acceleration(_position, _velocity);

var position2:Point = new Point();
position2.x = _position.x + _velocity.x * elapsed;
position2.y = _position.y + _velocity.y * elapsed;

var velocity2:Point = new Point();
velocity2.x = _velocity.x + accel1.x * elapsed;
velocity2.y = _velocity.y + accel1.x * elapsed;

var accel2:Point = acceleration(position2, velocity2);

_position.x += (_velocity.x + velocity2.x) / 2 * elapsed;
_position.y += (_velocity.y + velocity2.y) / 2 * elapsed;

_velocity.x += (accel1.x + accel2.x) / 2 * elapsed;
_velocity.y += (accel1.y + accel2.y) / 2 * elapsed;
RUNGE-KUTTA FOURTH ORDER INTEGRATION (RK4)
RK4: LIKE RK2, BUT WORSE
// pos1 is current position of object
// vel1 is current velocity of object
acc1 = acceleration(pos1, vel1)

pos2 = pos1 + vel1 / 2 * time
vel2 = vel1 + acc1 / 2 * time
acc2 = acceleration(pos2, vel2)

pos3 = pos1 + vel2 / 2 * time
vel3 = vel1 + acc2 / 2 * time
acc3 = acceleration(pos3, vel3)

pos4 = pos1 + vel3 * time
vel4 = vel1 + acc3 * time
acc3 = acceleration(pos4, vel4)

pos1 += (vel1 + vel2 * 2 + vel3 * 2 + vel4) / 6 * time
vel1 += (acc1 + acc2 * 2 + acc3 * 2 + acc4) / 6 * time
// pos1 is current position of object
// vel1 is current velocity of object
acc1 = acceleration(pos1, vel1)

pos2 = pos1 + vel1 / 2 * time
vel2 = vel1 + acc1 / 2 * time
acc2 = acceleration(pos2, vel2)

pos3 = pos1 + vel2 / 2 * time
vel3 = vel1 + acc2 / 2 * time
acc3 = acceleration(pos3, vel3)

pos4 = pos1 + vel3 * time
vel4 = vel1 + acc3 * time
acc3 = acceleration(pos4, vel4)

pos1 += (vel1 + vel2 * 2 + vel3 * 2 + vel4) / 6 * time
vel1 += (acc1 + acc2 * 2 + acc3 * 2 + acc4) / 6 * time
// pos1 is current position of object
// vel1 is current velocity of object
accl1 = acceleration(pos1, vel1)

pos2 = pos1 + vel1 / 2 * time
vel2 = vel1 + acc1 / 2 * time
acc2 = acceleration(pos2, vel2)

pos3 = pos1 + vel2 / 2 * time
vel3 = vel1 + acc2 / 2 * time
acc3 = acceleration(pos3, vel3)

pos4 = pos1 + vel3 * time
vel4 = vel1 + acc3 * time
acc3 = acceleration(pos4, vel4)

pos1 += (vel1 + vel2 * 2 + vel3 * 2 + vel4) / 6 * time
vel1 += (acc1 + acc2 * 2 + acc3 * 2 + acc4) / 6 * time
// pos1 is current position of object
// vel1 is current velocity of object
acc1 = acceleration(pos1, vel1)

pos2 = pos1 + vel1 / 2 * time
vel2 = vel1 + acc1 / 2 * time
acc2 = acceleration(pos2, vel2)

pos3 = pos1 + vel2 / 2 * time
vel3 = vel1 + acc2 / 2 * time
acc3 = acceleration(pos3, vel3)

pos4 = pos1 + vel3 * time
vel4 = vel1 + acc3 * time
acc3 = acceleration(pos4, vel4)

pos1 += (vel1 + vel2 * 2 + vel3 * 2 + vel4) / 6 * time
vel1 += (acc1 + acc2 * 2 + acc3 * 2 + acc4) / 6 * time
// pos1 is current position of object
// vel1 is current velocity of object
accl1 = acceleration(pos1, vel1)

pos2 = pos1 + vel1 / 2 * time
vel2 = vel1 + acc1 / 2 * time
acc2 = acceleration(pos2, vel2)

pos3 = pos1 + vel2 / 2 * time
vel3 = vel1 + acc2 / 2 * time
acc3 = acceleration(pos3, vel3)

pos4 = pos1 + vel3 * time
vel4 = vel1 + acc3 * time
acc3 = acceleration(pos4, vel4)

pos1 += (vel1 + vel2 * 2 + vel3 * 2 + vel4) / 6 * time
vel1 += (acc1 + acc2 * 2 + acc3 * 2 + acc4) / 6 * time
var time:int = getTimer();
var elapsed:Number = (time - _oldTime) / 1000;
_oldTime = time;

var accel1:Point = acceleration(_position, _velocity);

var position2:Point = new Point();
position2.x = _position.x + _velocity.x / 2 * elapsed;
position2.y = _position.y + _velocity.y / 2 * elapsed;

var velocity2:Point = new Point();
velocity2.x = _velocity.x + accel1.x / 2 * elapsed;
velocity2.y = _velocity.y + accel1.x / 2 * elapsed;

var accel2:Point = acceleration(position2, velocity2);

var position3:Point = new Point();
position3.x = _position.x + velocity2.x / 2 * elapsed;
position3.y = _position.y + velocity2.y / 2 * elapsed;

var velocity3:Point = new Point();
velocity3.x = _velocity.x + accel2.x / 2 * elapsed;
velocity3.y = _velocity.y + accel2.x / 2 * elapsed;

var accel3:Point = acceleration(position3, velocity3);

var position4:Point = new Point();
position4.x = _position.x + velocity3.x * elapsed;
position4.y = _position.y + velocity3.y * elapsed;

var velocity4:Point = new Point();
velocity4.x = _velocity.x + accel3.x * elapsed;
velocity4.y = _velocity.y + accel3.x * elapsed;

var accel4:Point = acceleration(position4, velocity4);

_position.x += (_velocity.x + 2 * velocity2.x + 2 * velocity3.x + velocity4.x) / 6 * elapsed;
_position.y += (_velocity.y + 2 * velocity2.y + 2 * velocity3.y + velocity4.y) / 6 * elapsed;

_velocity.x += (accel1.x + 2 * accel2.x + 2 * accel3.x + accel4.x) / 6 * elapsed;
_velocity.y += (accel1.y + 2 * accel2.y + 2 * accel3.y + accel4.y) / 6 * elapsed;
BEHOLD THE AMAZING RESULTS!!!
Runge-Kutta Summary

* Not 100% accurate, but probably more accurate than you’ll ever need.

* You probably don’t need it.

* If you do need it, you probably already know you need it.

* If you are wondering whether you need it or not, you probably don’t.

* If you do need it, you need it.
VERLET INTEGRATION
Loup Verlet

Old guys FTW!
Verlet Integration

- Developed for molecular interactions.
- Extremely stable.
- Velocity is not explicitly stored.
Advanced Character Physics

by Thomas Jakobsen

available at

www.gamasutra.com
Verlet System

* Points
* Constraints
* Sticks
* Structures
* Hinges
VERLET POINTS
It moved this far in the last frame.
So we’ll move it that far on the next frame.
Thus, any change in position becomes a change in velocity.
temp = currentPos
velocity = currentPos - oldPos
currentPos += velocity
oldPos = temp
VerletPoint
CLASS
package
{
    import flash.display.Graphics;
    import flash.geom.Rectangle;

    public class VerletPoint
    {
        public var x:Number;
        public var y:Number;

        private var _oldX:Number;
        private var _oldY:Number;

        public function VerletPoint(x:Number, y:Number)
        {
            setPosition(x, y);
        }

        public function update():void
        {
            var tempX:Number = x;
            var tempY:Number = y;
            x += vx;
            y += vy;
            _oldX = tempX;
            _oldY = tempY;
        }

        public function setPosition(x:Number, y:Number):void
        {
            this.x = _oldX = x;
            this.y = _oldY = y;
        }

        public function constrain(rect:Rectangle):void
        {
            x = Math.max(rect.left, Math.min(rect.right, x));
            y = Math.max(rect.top, Math.min(rect.bottom, y));
        }

        public function set vx(value:Number):void
        {
            _oldX = x - value;
        }

        public function get vx():Number
        {
            return x - _oldX;
        }

        public function set vy(value:Number):void
        {
            _oldY = y - value;
        }

        public function get vy():Number
        {
            return y - _oldY;
        }

        public function render(g:Graphics):void
        {
            g.beginFill(0);
            g.drawCircle(x, y, 4);
            g.endFill();
        }
    }
}
public function update():void
{
    var tempX:Number = x;
    var tempY:Number = y;
    x += vx;
    y += vy;
    _oldX = tempX;
    _oldY = tempY;
}
Verlet Constraints
Verlet Constraints
package
{
import flash.display.Graphics;
import flash.geom.Rectangle;

public class VerletPoint
{
    public var x:Number;
    public var y:Number;
    private var _oldX:Number;
    private var _oldY:Number;

    public function VerletPoint(x:Number, y:Number)
    {
        setPosition(x, y);
    }

    public function update():void
    {
        var tempX:Number = x;
        var tempY:Number = y;
        x += vx;
        y += vy;
        _oldX = tempX;
        _oldY = tempY;
    }

    public function setPosition(x:Number, y:Number):void
    {
        this.x = _oldX = x;
        this.y = _oldY = y;
    }

    public function constrain(rect:Rectangle):void
    {
        x = Math.max(rect.left, Math.min(rect.right, x));
        y = Math.max(rect.top, Math.min(rect.bottom, y));
    }

    public function set vx(value:Number):void
    {
        _oldX = x - value;
    }

    public function get vx():Number
    {
        return x - _oldX;
    }

    public function set vy(value:Number):void
    {
        _oldY = y - value;
    }

    public function get vy():Number
    {
        return y - _oldY;
    }

    public function render(g:Graphics):void
    {
        g.beginFill(0);
        g.drawCircle(x, y, 4);
        g.endFill();
    }
}
public function constrain(rect:Rectangle):void {
    x = Math.max(rect.left, Math.min(rect.right, x));
    y = Math.max(rect.top, Math.min(rect.bottom, y));
}
Verlet Sticks
Verlet Sticks
VerletStick
CLASS
package
{
    import flash.display.Graphics;

    public class VerletStick
    {
        private var _pointA:VerletPoint;
        private var _pointB:VerletPoint;
        private var _length:Number;

        public function VerletStick(pointA:VerletPoint,
                                  pointB:VerletPoint,
                                  length:Number = -1)
        {
            _pointA = pointA;
            _pointB = pointB;
            if(length == -1)
            {
                var dx:Number = _pointA.x - _pointB.x;
                var dy:Number = _pointA.y - _pointB.y;
                _length = Math.sqrt(dx * dx + dy * dy);
            }
            else
            {
                _length = length;
            }
        }

        public function update():void
        {
            var dx:Number = _pointB.x - _pointA.x;
            var dy:Number = _pointB.y - _pointA.y;
            var dist:Number = Math.sqrt(dx * dx + dy * dy);
            var diff:Number = _length - dist;
            var offsetX:Number = (diff * dx / dist) / 2;
            var offsetY:Number = (diff * dy / dist) / 2;
            _pointA.x -= offsetX;
            _pointA.y -= offsetY;
            _pointB.x += offsetX;
            _pointB.y += offsetY;
        }

        public function render(g:Graphics):void
        {
            g.lineStyle(0);
            g.moveTo(_pointA.x, _pointA.y);
            g.lineTo(_pointB.x, _pointB.y);
        }
    }
}
public function update():void
{
    var dx:Number = _pointB.x - _pointA.x;
    var dy:Number = _pointB.y - _pointA.y;
    var dist:Number = Math.sqrt(dx * dx + dy * dy);

    var diff:Number = _length - dist;

    var offsetX:Number = (diff * dx / dist) / 2;
    var offsetY:Number = (diff * dy / dist) / 2;

    _pointA.x -= offsetX;
    _pointA.y -= offsetY;

    _pointB.x += offsetX;
    _pointB.y += offsetY;
}
public function update():void {
    var dx:Number = _pointB.x - _pointA.x;
    var dy:Number = _pointB.y - _pointA.y;
    var dist:Number = Math.sqrt(dx * dx + dy * dy);

    var diff:Number = _length - dist;

    var offsetX:Number = (diff * dx / dist) / 2;
    var offsetY:Number = (diff * dy / dist) / 2;

    _pointA.x -= offsetX;
    _pointA.y -= offsetY;

    _pointB.x += offsetX;
    _pointB.y += offsetY;
}
public function update():void
{
    var dx:Number = _pointB.x - _pointA.x;
    var dy:Number = _pointB.y - _pointA.y;
    var dist:Number = Math.sqrt(dx * dx + dy * dy);

    var diff:Number = _length - dist;

    var offsetX:Number = (diff * dx / dist) / 2;
    var offsetY:Number = (diff * dy / dist) / 2;

    _pointA.x -= offsetX;
    _pointA.y -= offsetY;

    _pointB.x += offsetX;
    _pointB.y += offsetY;
}
VerletStick
Demo
Verlet Structures
Triangle
Demo
ITERATION

Dealing With Multiple Constraints
updatePoints()
for(var i = 0; i < iter; i++)
{
    handleConstraints()
    updateSticks()
}
Verlet Hinges
Verlet Builder