math(s) – animation

math – animation 3

convert radians to degrees

```
radians = degrees * Math.PI / 180
degrees = radians * 180 / Math.PI
```

rotate to mouse

```
dx = mouseX - sprite.x
dy = mouseY - sprite.y
sprite.rotation = Math.atan2(dy,dx) * 180 / Math.PI
```

create oval(s)

```
public function frame(event:Event) {
x = centerX + Math.cos(angle) * radiusX;
y = centerY + Math.sin(angle) * radiusY;
angle += speed;
}
```

distance between point(s)

```
dx = x2 - x1;
dy = y2 - y1;
distance = Math.sqrt(dx*dx + dy*dy);
```

math – animation 4

combine color(s)

```
color24 = red « 16 | green « 8 | blue;
color32 = alpha « 24 | red « 16 | green « 8 | blue;
```

extract color(s)

```
red = color24 » 16;
green = color24 » 8 & 0xFF;
blue = color24 & 0xFF;
```

draw curve(s)

```
  // draw through (xt,yt) with endpoints (x0,y0) and (x2,y2)
x1 = xt * 2 - (x0 + x2)/2;
y1 = xt * 2 - (y0 + y2)/2;
moveTo(x0,y0);
curveTo(x1,y1,x2,y2);
```

math – animation 5

convert angular velocity(s)

```
vx = speed * Math.cos(angle);
vy = speed * Math.sin(angle);
```

convert angular acceleration(s)

```
ax = force * Math.cos(angle);
ay = force * Math.sin(angle);
```

math – animation 06

out-of-bound(s)

```
if ( sprite.x - sprite.width/2 > right ||
     sprite.x + sprite.width/2 < left ||
     sprite.y - sprite.height/2 > bottom ||
     sprite.y + sprite.height/2 < top )
     {
     ...
     }
```

correct friction(s)

```
speed = Math.sqrt(vx*vx + vy*vy);
angle = Math.atan2(vy,vx);
if (speed > friction) { speed -= friction; }
else { speed = 0; }
```

easy friction(s)

```
vx *= friction;
vy *= friction;
```

math – animation 08

easing(s)

```
dx = targetX - sprite.x;
dy = targetY - sprite.y;
vx = dx * easing;
vy = dy * easing;
sprite.x += vx;
sprite.y += vy;
```

spring(s)

```
ax = (targetX - sprite.x) * spring;
ay = (targetY - sprite.y) * spring;
vx += ax;
vy += ay;
vx *= friction;
vy *= friction;
sprite.x += vx;
sprite.y += vy;
```

offset(s)

```
dx = sprite.x - fixedX;
dy = sprite.y - fixedY;
angle = Math.atan2(dy,dx);
targetX = fixedX + Math.cos(angle) * springLength;
targetY = fixedY + Math.sin(angle) * springLength;
// spring to target
```

math – animation 09

collision(s)

```
dx = spriteB.x - spriteA.x;
dy = spriteB.y - spriteB.y;
distance = Math.sqrt(dx*dx + dy*dy);
if (distance < spriteA.radius + spriteB.radius) {
 ...
}
```

math – animation 10

coordinate rotation(s)

```
x1 = Math.cos(angle) * x - Math.sin(angle) * y;
y1 = Math.cos(angle) * y + Math.sin(angle) * x;
```

math – animation 11

momentum conservation(s)

```
vxTotal = vx0 - vx1;
vx0 = ((ball0.mass - ball1.mass) * vx0 + 2 * ball1.mass * vx1) /
      (ball0.mass + ball1.mass);
vx1 = vxTotal + vx0;
```

math – animation 12

gravitation(s)

```
dx = partB.x - partA.x;
dy = partB.y - partA.y;
distSQ = dx*dx + dy*dy;
distance = Math.sqrt(distSQ);
force = partA.mass * partB.mass / distSQ;
ax = force * dx / distance;
ay = force * dy / distance;
partA.vx += ax / partA.mass;
partA.vy += ay / partA.mass;
partB.vx += ax / partB.mass;
partB.vy += ay / partB.mass;
```

math – animation 14

law of cosine(s)

```
angleA = Math.acos((b*b + c*c - a*a) / (2 * b * c));
angleB = Math.acos((a*a + c*c - b*b) / (2 * a * c));
angleC = Math.acos((a*a + b*b - c*c) / (2 * a * b));
```

math – animation 15

basic perspective(s)

```
scale = fl / (fl + zpos);
sprite.scaleX = sprite.scaleY = scale;
sprite.alpha = scale;  // optional
sprite.x = vanishingPointX + xpos * scale;
sprite.y = vanishingPointY + ypos * scale;
```

Z sorting(s)

```
// array of 3D object with zpos property;
objectArray.sortOn("zpos", Array.DESCENDING | Array.NUMERIC);
for (var i:unint; i < numObjects; i++) {
  setChildIndex(objectArray[i].i);
}
```

3D distance(s)

```
distance = Math.sqrt(dx*dx + dy*dy + dz*dz);
```