8 Physics Simulations

8.1 Billiard-Game Physics

8.2 Game Physics Engines

Literature:

cocos2d-x.org

R. Engelbert: Cocos2d-x Beginner's Guide, 2nd ed.,
Packt Publishing 2015

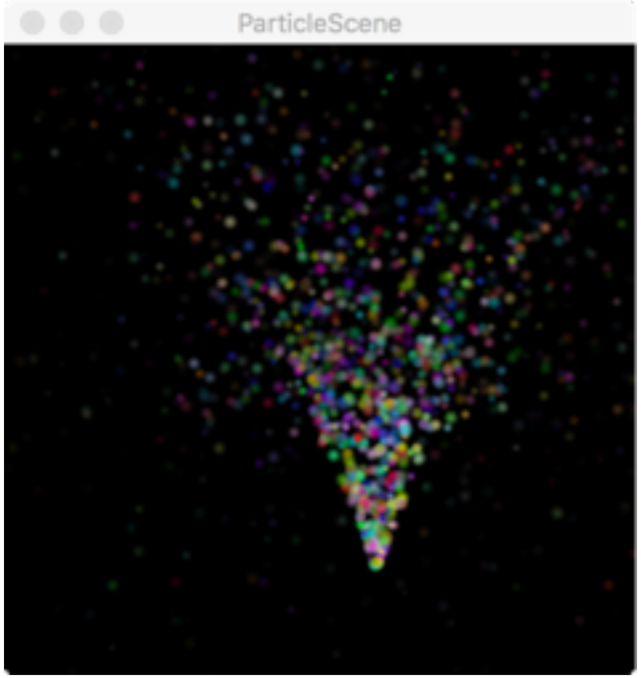
Particle Animations

> "A particle system is a collection of many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system."
>
> (W.T. Reeves)

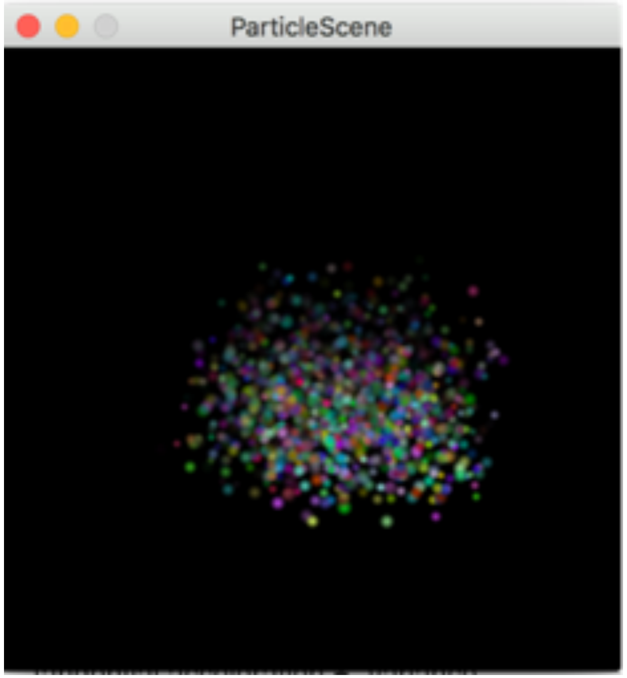
- Animation of complex physical phenomena:
  - Smoke, explosions, sparks, rain, snow, …
  - Irregular, complex, and ill-defined surfaces
- Creation process for many small objects (particles)
  - Using stochastic processes
- Typical implementation:
  - Particle *emitter*: Source of the particles, shape located on stage
  - Particle behavior parameters, e.g.:
    - Spawning rate (number of particles over time)
    - Particle direction and velocity
    - Particle lifetime
  - Parameters often specified in fuzzy way (central value plus deviation range)
- Seminal paper:
  William T. Reeves: Particle Systems A Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics* 17(3), 1983

Particles: Gravity Mode vs. Radius Mode

- ***Gravity Mode*** lets particles fly toward or away from a center point. Its strength is that it allows very dynamic, organic effects.
- ***Radius Mode*** causes particles to rotate in a circle. It also allows you to create spiral effects with particles either rushing inward or rotating outward.



Gravity

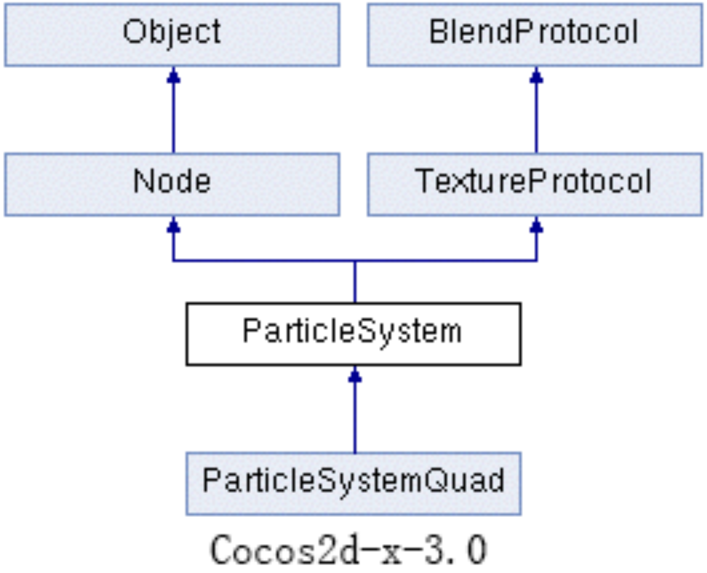


Radius

Example: Particles in Cocos2d-x

Attributes of a Particle System:

- emission rate of the particles
- Gravity Mode (Mode A):
- gravity
- direction
- speed +- variance
- tangential acceleration +- variance
- radial acceleration +- variance



Properties common to all modes:

- life +- life variance
- start spin +- variance
- end spin +- variance
- start size +- variance
- end size +- variance
- start color +- variance
- end color +- variance
- life +- variance
- blending function
- texture

Gravity Mode (Mode A):

- gravity
- direction
- speed +- variance
- tangential acceleration +- variance
- radial acceleration +- variance

Radius Mode (Mode B):

- startRadius +- variance
- endRadius +- variance
- rotate +- variance

http://www.cocos2d-x.org/wiki/Particle_System_Comparison_of_v2x_and_v3x

Cocos2d-x: Simple Particle Example (1)

- For predefined particle effects, extremely simple:

```
// Create particle emitter
auto emitter = ParticleFireworks::create();
this->addChild(emitter);
```

- For custom particle effects, not difficult:

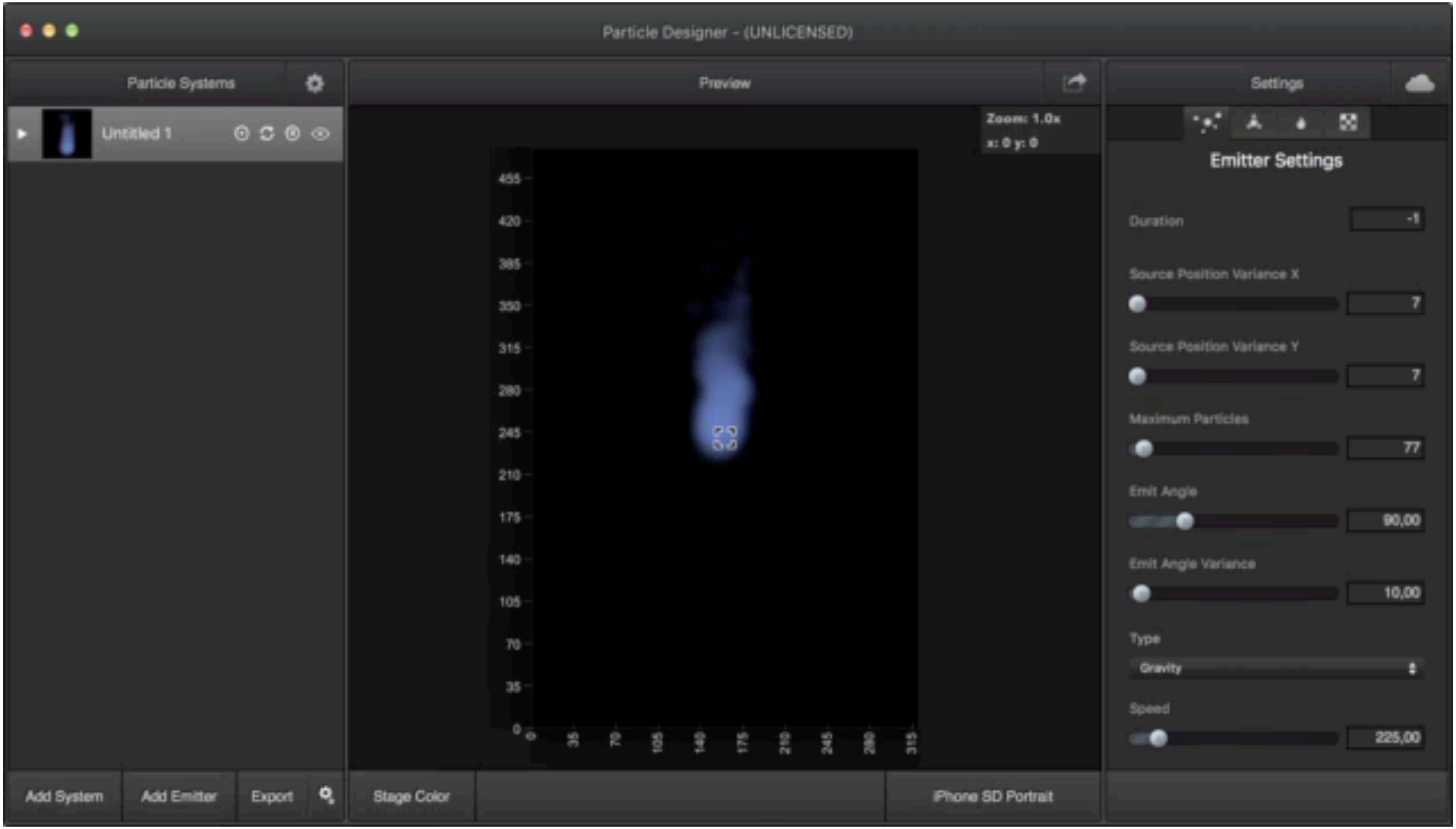
```
_jet = ParticleSystemQuad::create("jet.plist");
_jet->setSourcePosition(Vec2(-_rocket->getRadius() * 0.8f,0));
_jet->setAngle(180);
_jet->stopSystem();
this->addChild(_jet, kBackground);
```

- Note:
  - Animation starts immediately after creation of particle system
  - `stopSystem()` pauses animation until needed
  - `resetSystem()` starts animation again

Simple Particle Example (2)

```
kListener->onKeyPressed = [=](EventKeyboard::KeyCode keyCode, Event* event) {
    if (keyCode == EventKeyboard::KeyCode::KEY_SPACE) {
        CCLOG("Space pressed");
        if (animRunning)
            emitter->stopSystem();
        else
            emitter->resetSystem();
        animRunning = !animRunning;
    }
    if (keyCode == EventKeyboard::KeyCode::KEY_R) {
        CCLOG("R pressed");
        emitter->stopSystem();
        emitter->setEmitterMode(ParticleSystem::Mode::RADIUS);
        emitter->setStartRadius(150);
        emitter->setStartRadiusVar(30);
        emitter->setEndRadius(ParticleSystem::START_RADIUS_EQUAL_TO_END_RADIUS);
        emitter->resetSystem();
     }
    if (keyCode == EventKeyboard::KeyCode::KEY_G) {
        CCLOG("G pressed");
        emitter->stopSystem();
        emitter->setEmitterMode(ParticleSystem::Mode::GRAVITY);
        emitter->resetSystem();
    }
```

Particle Generation Software



Zoom: 1.0x
x: 0 y: 0

Emitter Settings

Duration -1

Source Position Variance X 7

Source Position Variance Y 7

Maximum Particles 77

Emit Angle 90,00

Emit Angle Variance 10,00

Type Gravity

Speed 225,00

https://71squared.com/

Transfer Format for Particles: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
       <key>angle</key>
       <integer>0</integer>
       <key>angleVariance</key>
       <integer>0</integer>
       <key>duration</key>
       <real>-1</real>
       <key>gravityx</key>
       <real>0</real>
       <key>gravityy</key>
       <real>0</real>
       <key>maxParticles</key>
       <real>479</real>
```

...

This is a VERY small excerpt of `jet.plist`!

Example: Advanced Particle Animations

https://www.youtube.com/watch?v=3qYQx5A40Tk

Physics Engines

- Physics engine = simulation of physics (covers collision detection)
  - Rigid bodies
  - Soft bodies
  - Fluid dynamics
- General problem with various applications
- Main focus here : Rigid body physics, mainly for games
- Simulation takes into account:
  - shapes
  - mass
  - all relevant forces

Box2D Physics Engine

- Written by Erin Catto (C++, many ports), 2006
  - Used in many games, including *Angry Birds*
- Bodies:
  - Convex polygons, circles, edge shapes
- Connections between bodies:
  - Joints
- Forces
- Includes simulation of:
  - Gravity
  - Friction
  - Collisions with elasticity (restitution)

See:
http://box2d.org
http://www.iforce2d.net

Body Properties

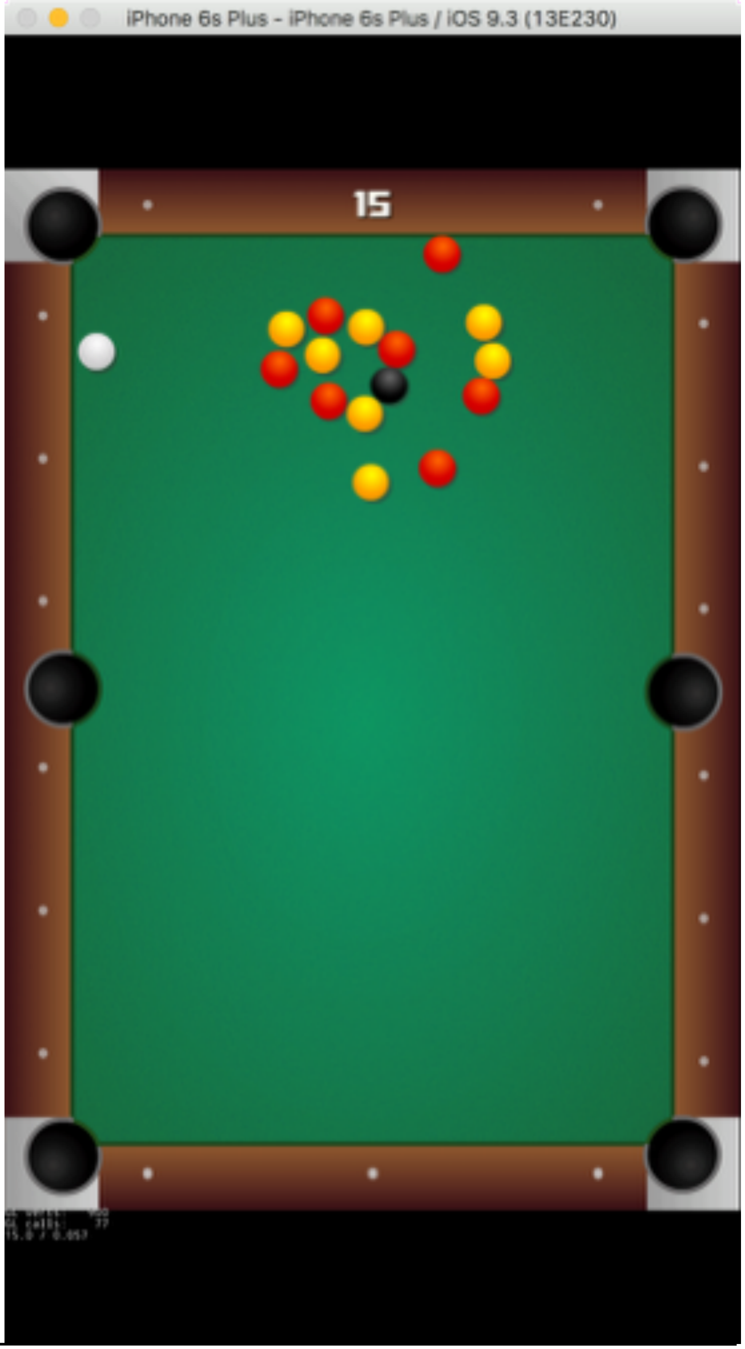
- More or less visible body properties:
  - Location (visible, in context)
  - Angle (often visible)
  - Mass/inertia
  - Velocity
  - Angular velocity
  - Rotational inertia (effort needed for spinning)
- Fixtures:
  - Shape of a body (polygon or circle)
  - Restitution (bounciness)
  - Friction (slipperiness)
  - Density (heaviness in relation to its area)
- Sensors:
  - Passive fixtures: Report contact

Integration of Box2D into Cocos2d-x

- Simulation root: `b2world` object
  - Filled with `b2body` objects
  - Usually provided with gravity vector
  - `AllowSleeping` parameter: Ignore objects which are not moving
  - `ContinuousPhysics` parameter:
- Simulation steps:
  - `b2world->Step(dt, v_iter, p_iter)`
- PTM (pixels to meters) ratio
  - Physics engine works based on meters, conversions are likely to be needed

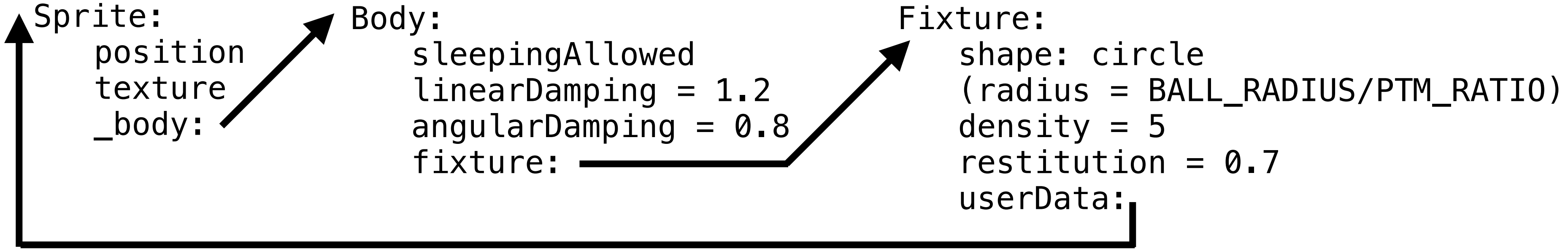
```
b2Vec2 gravity;
gravity.Set(0.0f, -10.0f);
world = new b2World(gravity);
world->SetAllowSleeping(true);
world->SetContinuousPhysics(true);
collisionListener = new CollisionListener();
world->SetContactListener(_collisionListener);
```

Example: MiniPool



Body and Sprite

- Body: Abstract representation of physical properties
- Sprite: Representation of movable graphics
- Example: Ball



```
Sprite:             Body:                             Fixture:
   position            sleepingAllowed                   shape: circle
   texture             linearDamping = 1.2               (radius = BALL_RADIUS/PTM_RATIO)
   _body:              angularDamping = 0.8              density = 5
                       fixture:                          restitution = 0.7
                                                         userData:
```

```
b2BodyDef bodyDef;
bodyDef.type = b2_dynamicBody;
_body = _game->getWorld()->CreateBody(&bodyDef);
```

Managing Collisions

- Collision filters
  - Fixtures carry categoryBits (in Cocos2d-x)
    Objects collide only if their filters coincide
  - Bit level manipulations
- Contact listener
  - Register for being informed about begin/end of contact
- Pre-solve resolution:
  - Listen and react to a collision before reactions are calculated
  - `PreSolve` attribute of `CollisionListener`
  - E.g. to adapt friction or to cancel collision
- Post-solve resolution:
  - Interpretation of results of physics computation
  - E.g. to determine breaking, sticking of objects

Example: High-Level Program Code

```
//create circle shape
b2CircleShape  circle;
circle.m_radius = BALL_RADIUS/PTM_RATIO;

//define fixture
b2FixtureDef fixtureDef;
fixtureDef.shape = &circle;
fixtureDef.density = 5;
fixtureDef.restitution = 0.7;

//add collision filters so only white ball can be hit by cue
if (_type == kSpriteBall) {
    fixtureDef.filter.categoryBits = 0x0010;
} else if (_type == kSpritePlayer) {
    //white ball is tracked as bullet by simulation
    _body->SetBullet(true);
    fixtureDef.filter.categoryBits = 0x0100;
```