iOS Development

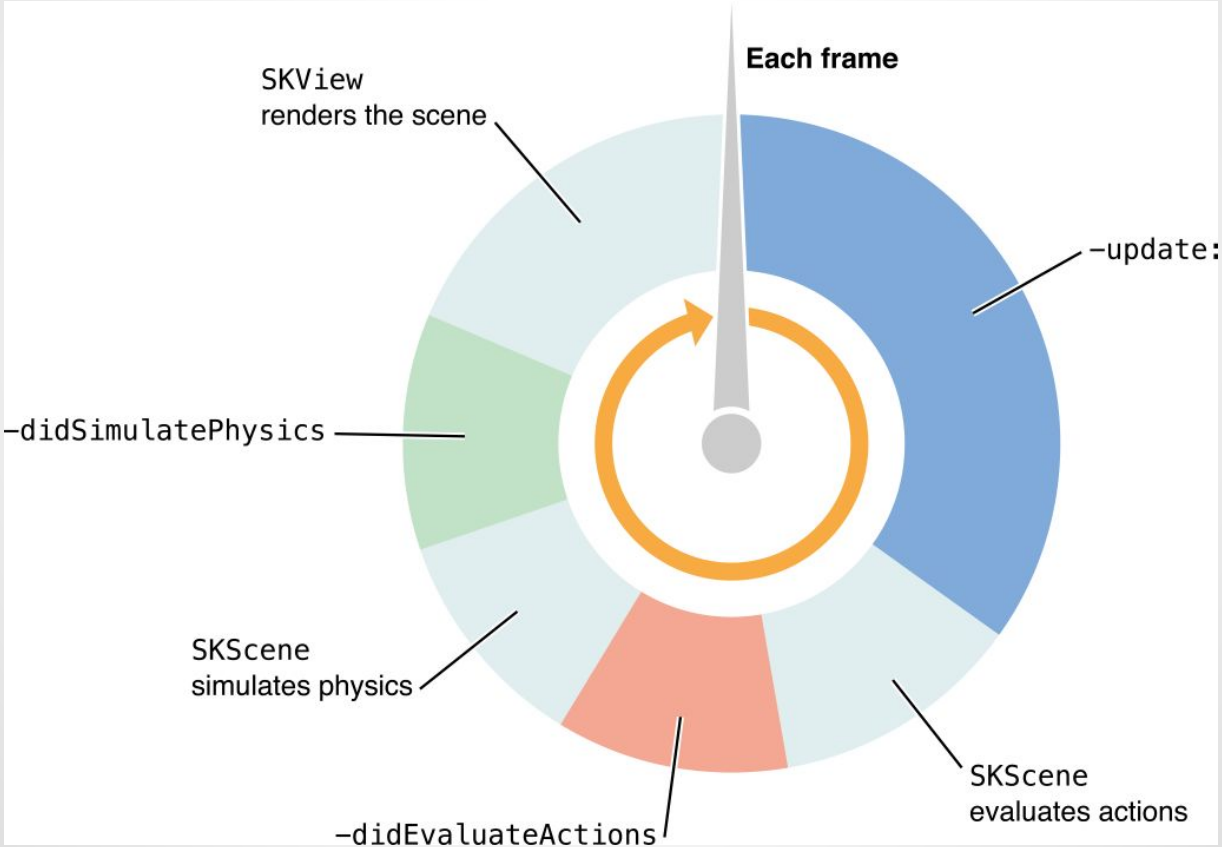
iOS Games with SpriteKit

Readings

SpriteKit - Apple Documention

SpriteKit Tutorials

SpriteKit Game Loop

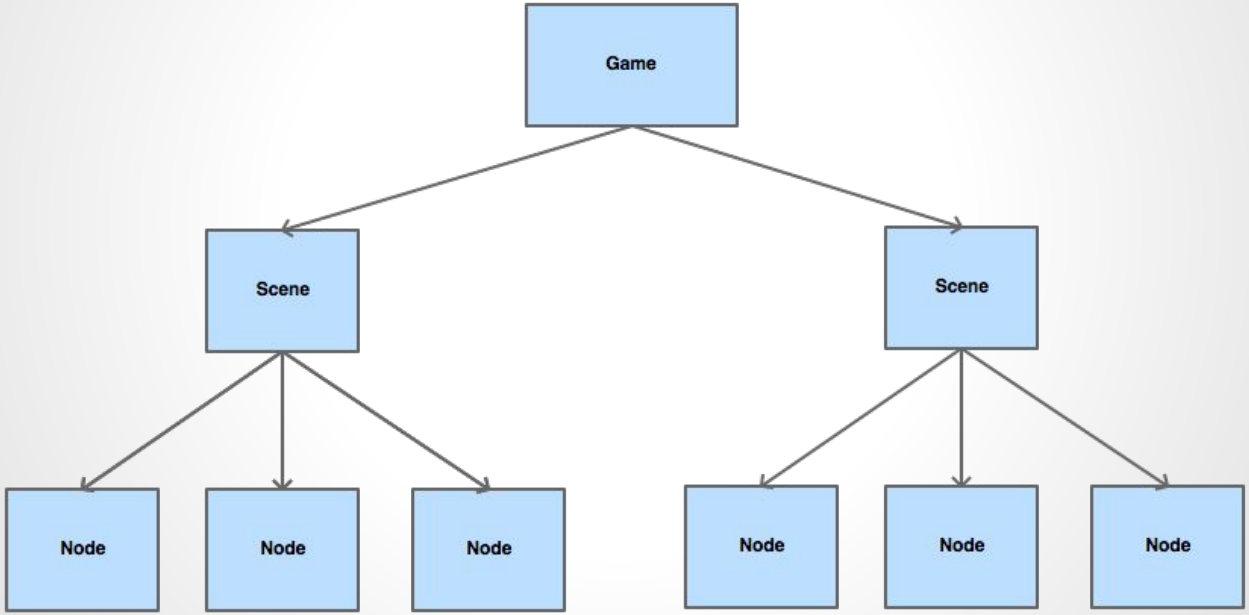


What is a Sprite?

Sprites are independent contexts that have their own content, size, position, attributes, etc.

Sprites handle their own drawing, collision detection and can even contain other sprites.

SpriteKit Hierarchy



SpriteKit Scenes

Scenes are collections of nodes. These can be for:

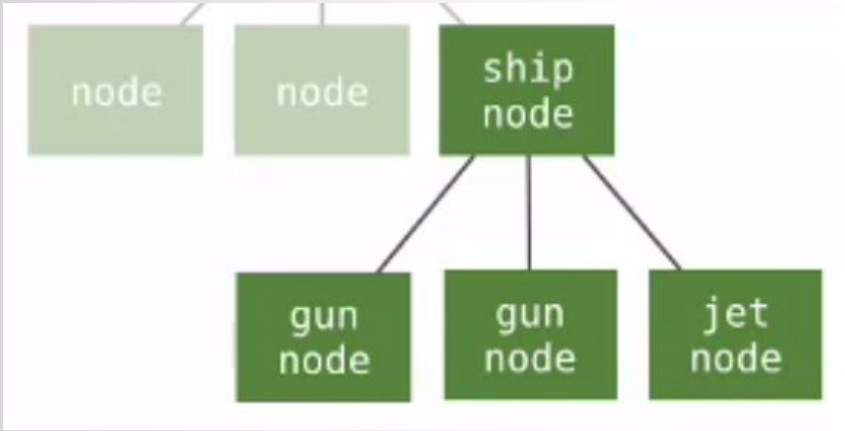
- Start screen
- Levels
- Game options
- Credits screen

SpriteKit Nodes

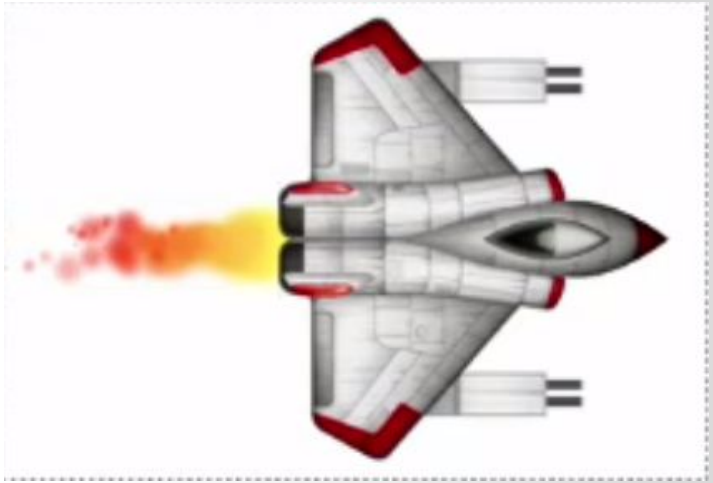
Nodes are objects in your scene and can represent:

- Characters
- Enemies
- Background
- Obstacles
- Debris

SpriteKit Nodes



=



SpriteKit Node Types

SKSpriteNode

SKLabelNode

SKEmitterNode

Sprite Node Implementation

In your scene:

```
SKSpriteNode *spaceship = [SKSpriteNode
spriteNodeWithImageNamed: @”spaceship”];

spaceship.position = CGPointMake(x, y);

[self addChild: spaceship];
```

CGPoint - X,Y Coordinates in Obj-C

C-Struct

```
struct CGPoint {
    CGFloat x;
    CGFloat y;
};
typedef struct CGPoint CGPoint;
```

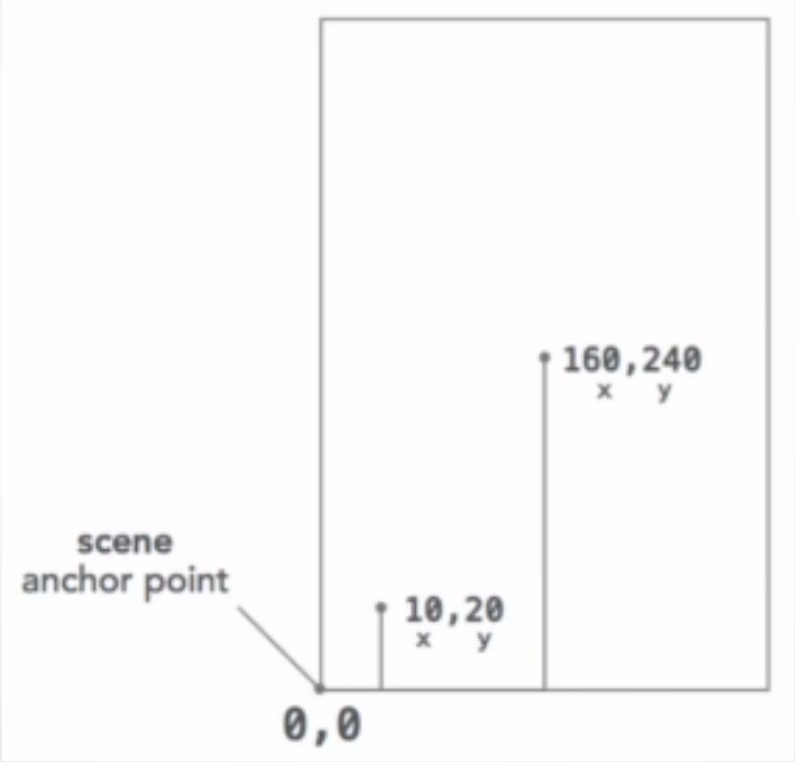
Creating a CGPoint

```
CGPointMake(CGFloat x, CGFloat y);
```

Example:

```
CGPoint position = CGPointMake(20.0f, 250.0f);
```

Coordinate Systems

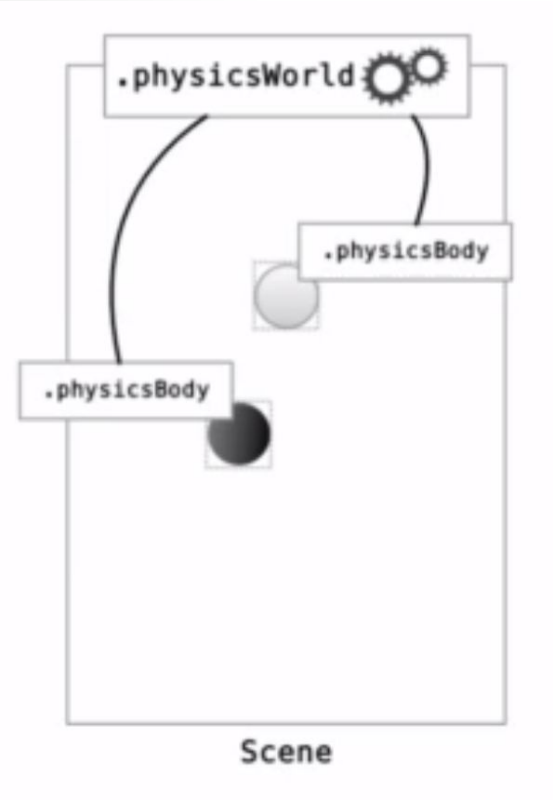


Physics Engine

The physics engine is on by default and controls:

- Gravity
- Mass and acceleration calculations
- Collisions
- And more

Physics Bodies



Physics Body Types

Volume - Most objects in your game

Edge - Usually boundaries, platforms or obstacles

Dynamic - Moved by simulator

Static - Not moved or affected by gravity

Physics Body Properties

- Friction - (0-1) Stickiness against collisions
- Linear Damping - (0-1) Medium of motion
- Restitution - (0-1) Bounciness
- Density
- Mass

Collisions and Contacts

Games aren’t useful unless you can detect the interaction between the autonomous entities within.

Collision vs. Contact

Collision - Keeps objects from intersecting

Contact - When two objects touch

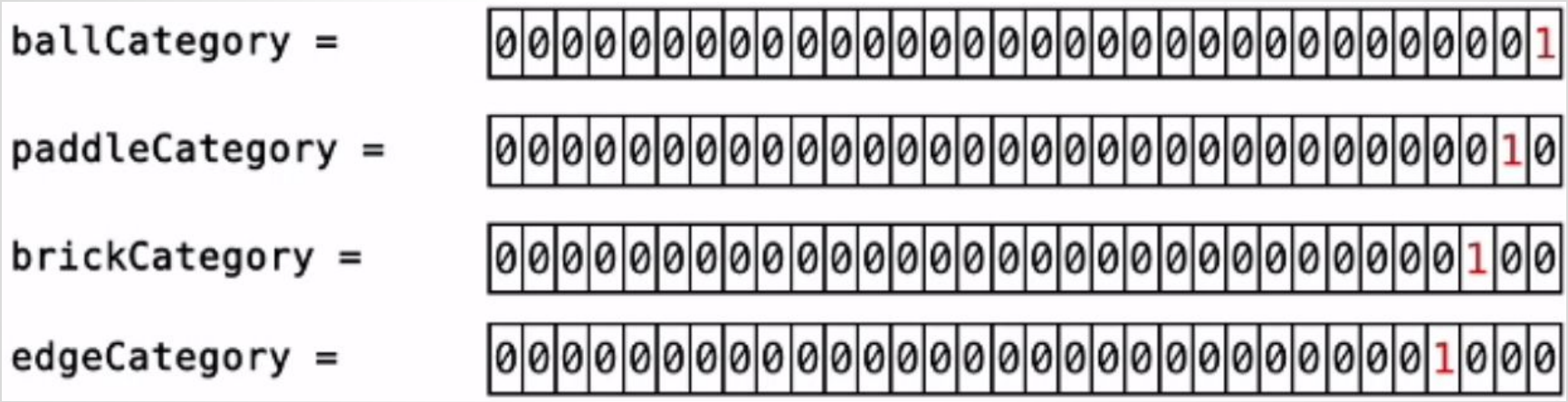
Collision and Contact BitMasks

Groups of sprites can be assigned categories for contact detection.

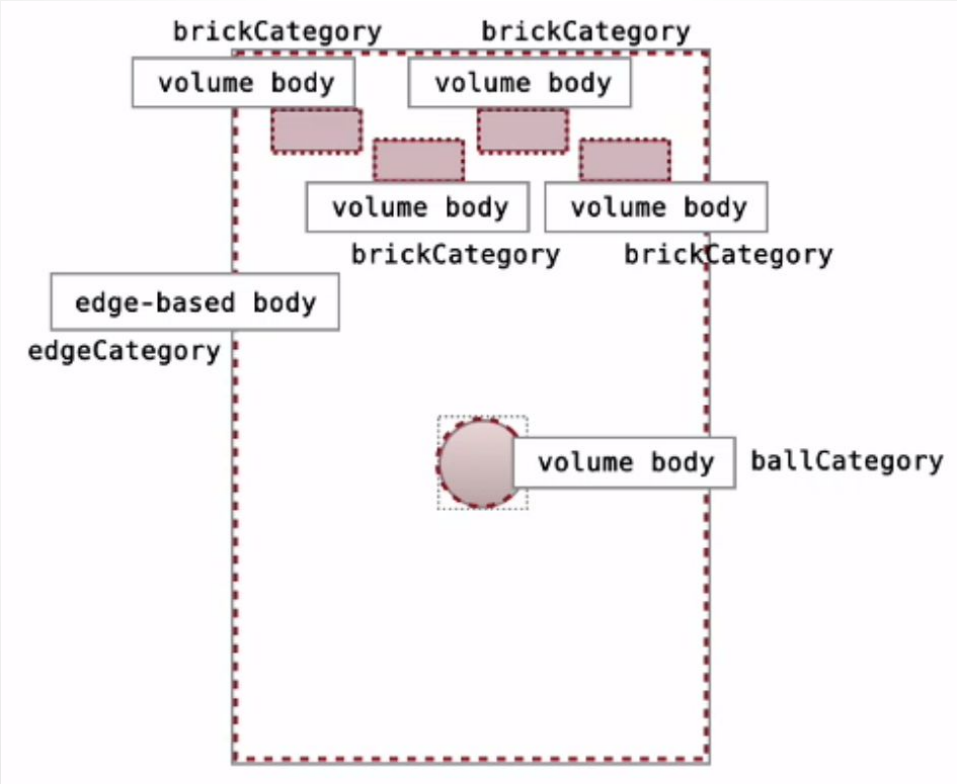
We put these into 32 bit integers, called bitmasks. Therefore, we can have up to 32 categories.

BitMasking

BitMasking Categories



Example Categories



Defining Categories

```
static const uint32_t jetCategory     = 1;
static const uint32_t wallCategory    = 2;
static const uint32_t barrierCategory = 4;
static const uint32_t laserCategory   = 8;
static const uint32_t enemyCategory   = 16;
```

Binary

```
jetCategory      = b00000000000000000000000000000001; // 1
wallCategory     = b00000000000000000000000000000010; // 2
barrierCategory = b00000000000000000000000000000100; // 4
laserCategory    = b00000000000000000000000000001000; // 8
enemyCategory    = b00000000000000000000000000010000; // 16
```

Shifting Bits

```
let jetCategory:UInt8      = 1
let wallCategory:UInt8     = 1 << 1
let barrierCategory:UInt8 = 1 << 2
let laserCategory:UInt8    = 1 << 3
let enemyCategory:UInt8    = 1 << 4
```

Contact Test Bit Mask

```
spaceship.physicsBody.categoryBitMask =
laserCategory;


spaceship.physicsBody.contactTestBitMask =
laserCategory | enemyCategory;
```

Detecting Contacts

Make our Scene the contact delegate:

```
<SKPhysicsContactDelegate>


-(void)didBeginContact:(SKPhysicsContact *)contact
{
  …
}


self.physicsWorld.delegate = self;
```

Collisions and Contacts

Demo

Sound Effects

```
SKAction *soundFx = [SKAction
playSoundFileNamed:@"Tink.caf" waitForCompletion: NO];


[self runAction: soundFx];
```

SKAction Sound Effects

Demo

SKEmitterNode

```
SKEmitterNode *jetFlames =
[NSKeyedUnarchiver unarchiveObjectWithFile:
 [[NSBundle mainBundle] pathForResource:@"Fire" ofType:@"sks"]];


jetFlames.position =
CGPointMake(0,0-_spaceship.frame.size.height*2);


[_spaceship addChild: jetFlames];
```

SKEmitterNodes

Demo

Accelerometer Data in Your Game

Add CoreMotion.framework

Add acceleromteter protocol:
`<UIAccelerometerDelegate>`

Receive Accelerometer Data

```
//CoreMotion
self.motionManager = [[CMMotionManager alloc] init];
self.motionManager.accelerometerUpdateInterval = .2;


[self.motionManager startAccelerometerUpdatesToQueue:
    [NSOperationQueue currentQueue]
    withHandler:^(CMAccelerometerData  *accelerometerData, NSError *error)
    {
      [self outputAccelertionData:accelerometerData.acceleration];
      if(error){
          NSLog(@"%@", error);
      }
}];
```

Helper Function

```
-(void)outputAccelertionData:(CMAcceleration)acceleration
{
    currentMaxAccelX = 0;
    currentMaxAccelY = 0;


    if(fabs(acceleration.x) > fabs(currentMaxAccelX))
    {
        currentMaxAccelX = acceleration.x;
    }
    if(fabs(acceleration.y) > fabs(currentMaxAccelY))
    {
        currentMaxAccelY = acceleration.y;
    }
}
```