Bringing Large Scale Console Games to iOS:

A Technical Overview of

Adaptation

Stéphane Khalil Jacoby
CTO / Co-Founder @ SQUARE 1 GAMES

GDC 12
SMARTPHONE & TABLET GAMES SUMMIT
GAME DEVELOPERS CONFERENCE 2012
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9

The Bard's Tale: A little history

1985

The Bard's Tale I:
Tales of the Unknown

1987

The Bar'd Tale II:
The Destiny Knight

1988 1991

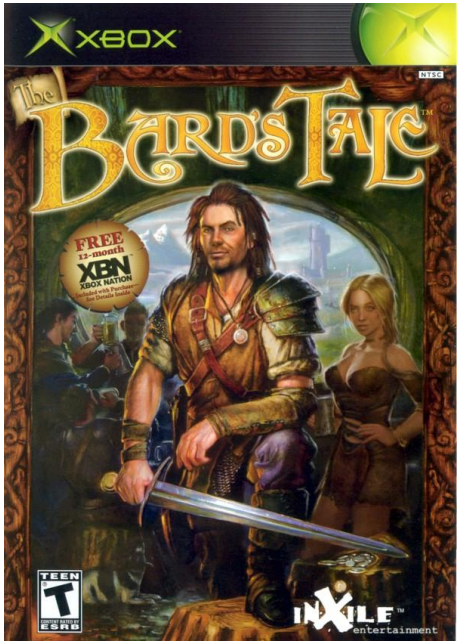
The Bard's Tale III:
Thief of Fate

2004

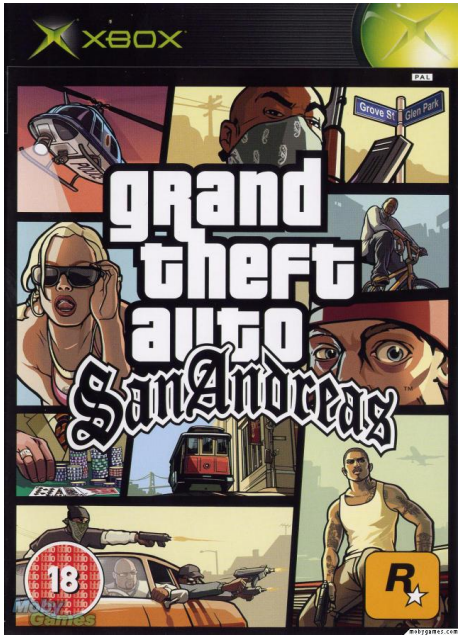
The Bard's Tale:
A Quest for Coin and Cleavage

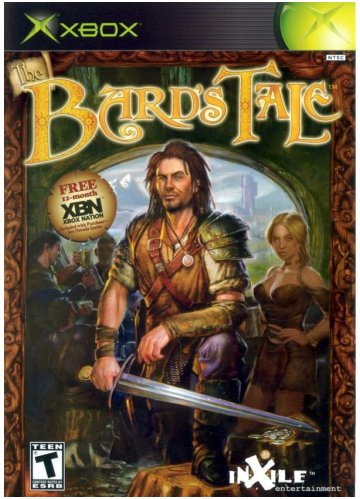
2011

The Bard's Tale: October 2004

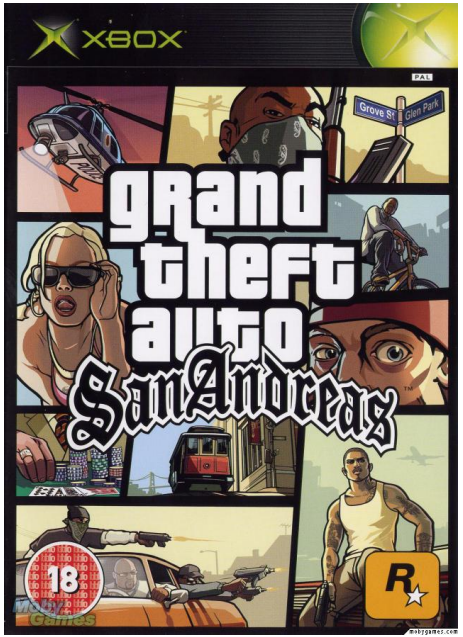


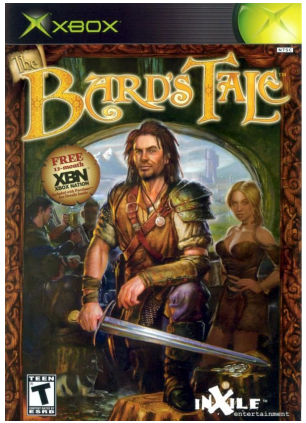
The Bard's Tale: October 2004

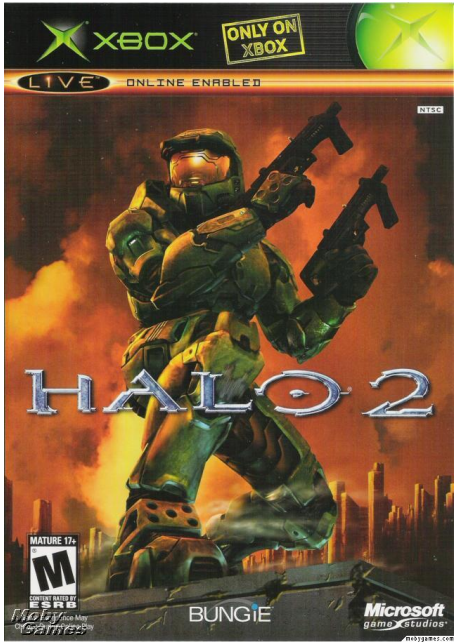




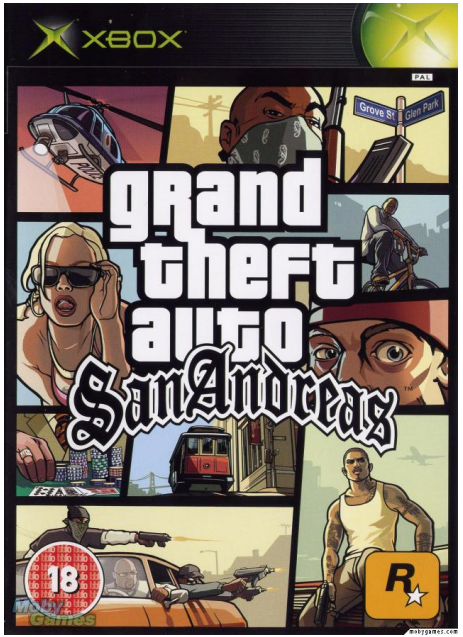
The Bard's Tale: October 2004



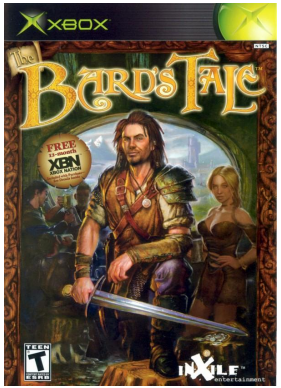


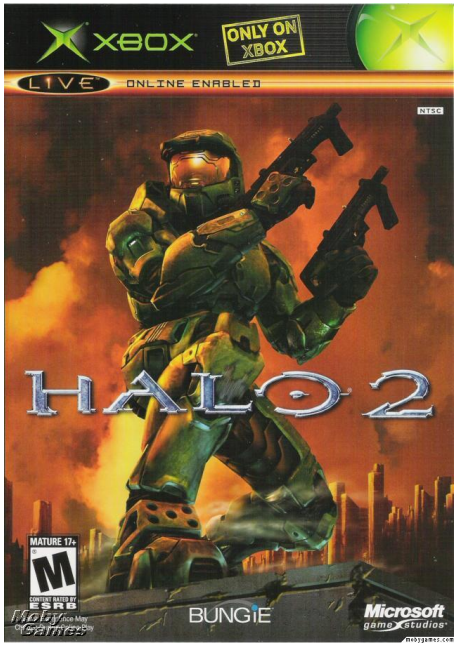


The Bard's Tale: October 2004



IGN.COM
2004 AWARDS
BEST GAME NO ONE PLAYED





The Bard's Tale: 2011

Console Games on iOS: Why?

- Devices are fast enough
- Market segment is not as crowded
- Huge potential user base
- Rich back catalog of content
- Potentially very low cost

Schedule & Team

- Dev: 1 Engineer
  - Initial Port: 13 weeks
  - Optimization: 6 weeks
  - iOS specific features: 5 weeks
- Art support: 1 week
- QA: 2 testers 2 weeks (x2)
- Design: 4 weeks
- Total Cost: **33 man weeks**

SQUARE 1 GAMES

Session Overview
Part I: Port Process

- Application framework
- Development workflow
- Rendering and Audio
- Conversion to touch controls

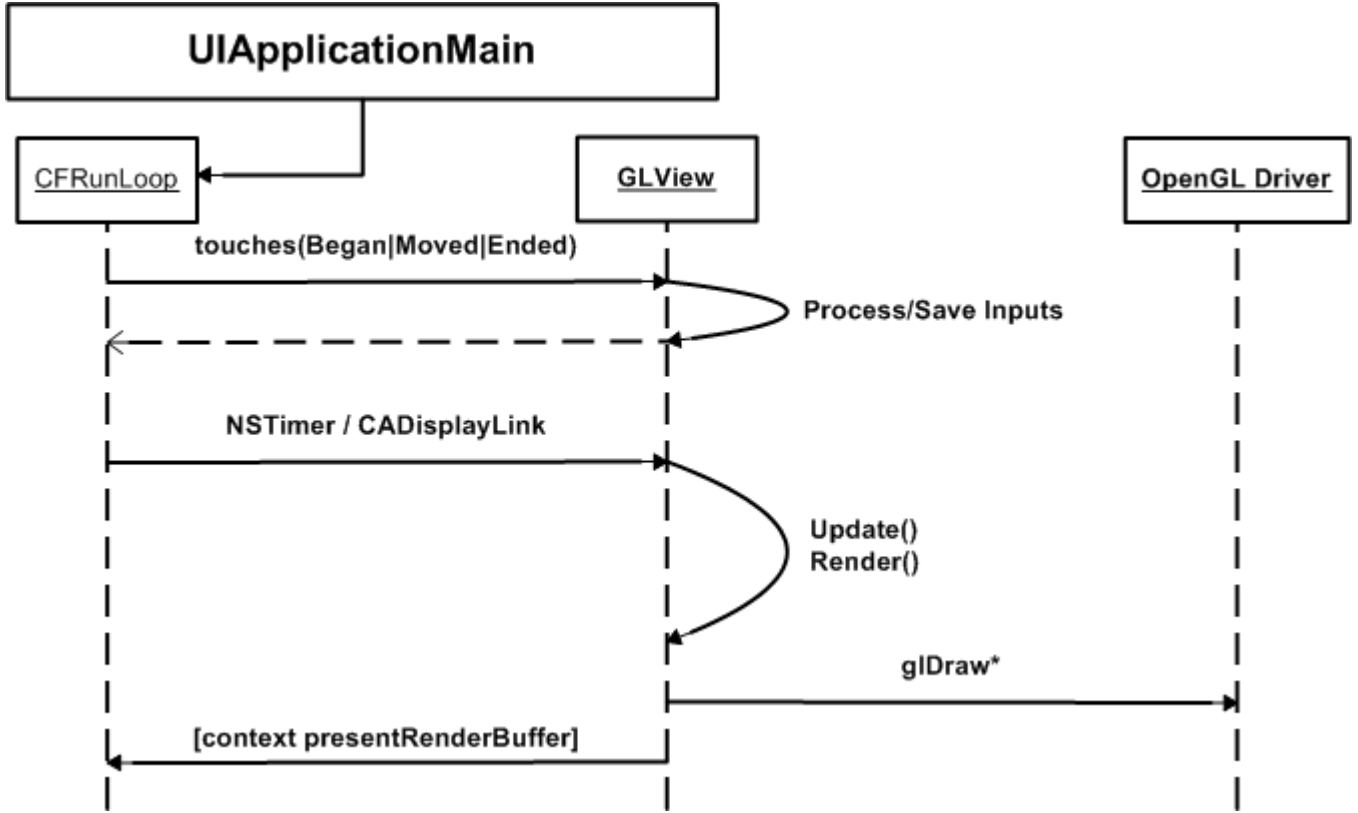
Session Overview (cont'd)
Part II: Post-Port Phase

- Performance and Memory Optimization
- App Bundle Size Reduction
- Adding iOS specific features after the fact
- iCloud integration

Application Framework

- iOS SDK imposes structure and paradigms
- Console Games used their own
- Where to begin?

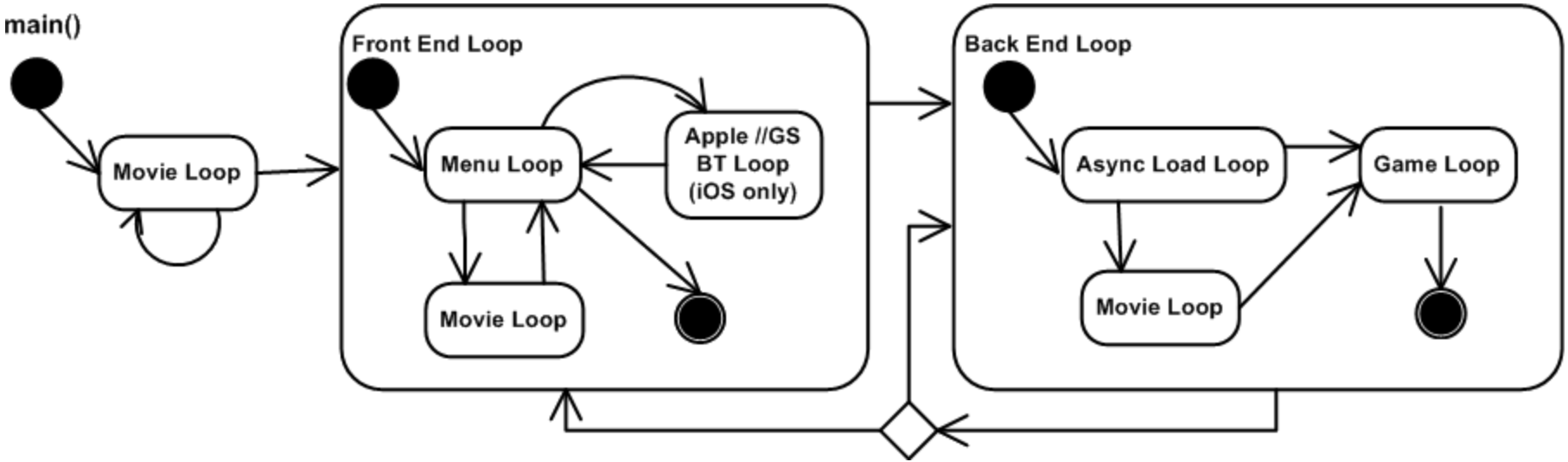
OpenGL Cocoa Touch App



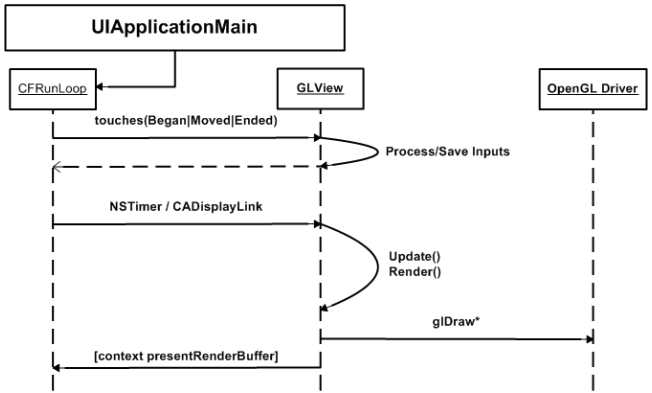
Typical Console Game Structure

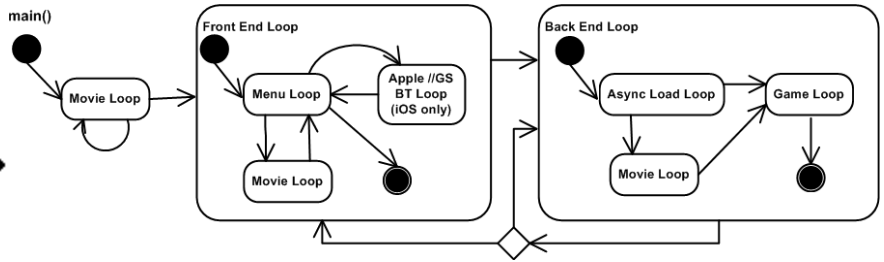
- Well, there isn't one
  - Different for each game
- At low level, comprises of some sort of monolithic loop (for(;;))
  - Good integration to the Cocoa Framework
  - BUT! May contain multiple loops
    - Nested
    - Sequential

The Bard's Tale Structure Simplified



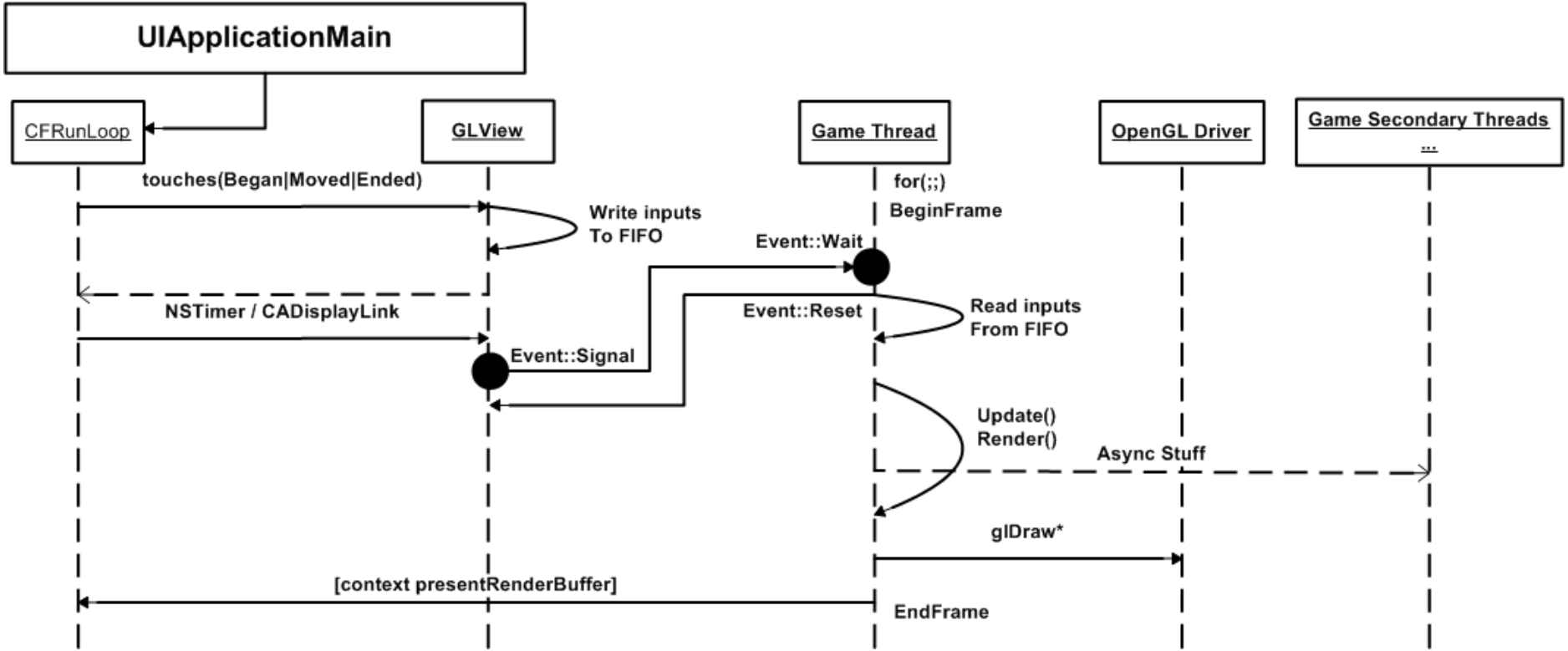
… doesn't quite fit the mold!





- 2 solutions
  - Change the object
  - Change the mold

Bringing it all together



Workflow: Data Deployment

- Console Games were large (DVD-ROM)
- Xcode signs and deploys entire App Bundle
- Slow iteration
- Post iOS 4.0, app data persists
  - Deploy entire data set with dedicated target
  - Debug with executable only targets

Rendering Core Runtime

- OpenGL ES 2.0 maps quasi fully to Direct3D

OP	XBOX (D3D DirectX 8): D3DDevice::*	iOS (Open GL ES 2.0) : gl*
Render States	SetRenderState	Enable, PolygonOffset, DepthMask, DepthFunc, BlendEquation, BlendFunc…
Textures	Create(Texture|Palette), Set(Texture|Palette) + Lock, SetTextureStageState	GenTextures, BindTexture, TexImage*, TexImageParameter*
Vertex/Index buffers	Create(Vertex|Index)Buffer, SetIndices, SetStreamSource + Lock	GenBuffers, BindBuffer, BufferData
Drawing	DrawPrimitive(UP), DrawIndexedPrimitive	DrawArrays, DrawElements
Vertex Declaration	CreateVertexShader(DECL)	VertexAttribPointer, EnableVertexAttribArray BindVertexArrayOES (iOS 4.0+ extension)

Rendering Core Runtime (cont'd)

- Differences in terms of shader dependent runtime

OP	XBOX (D3D DirectX 8): D3DDevice::*	iOS (Open GL ES 2.0) : gl*
Shader creation	ReadFile (pixel), (Create|Load)VertexShader	CreateShader, ShaderSource, CompileShader, AttachShader, LinkProgram
Shader activation	SetPixelShaderProgram, SelectVertexShader	UseProgram
Constant registers	Set(Vertex|Pixel)ShaderConstant	Uniform(1-4)(f+)(v+)

Shader Conversion

- Rewrite as OpenGL ES 2.0?

Microcode

```
// Constant OFFSETS - Common Header
#define VS_Bones 0
#define VS_ProjScale 140

vs_1_1

dcl_position v0
dcl_color v1
#define inPos v0       // XYZ pos
#define boneData v1 // weights & indices

// Copy Bone Indices
mov a0.xy, boneData.xy

// Build weighted matrix bone 1
mul r3, c[a0.x+0+VS_Bones], boneData.z
mul r4, c[a0.x+1+VS_Bones], boneData.z
mul r5, c[a0.x+2+VS_Bones], boneData.z
mul r6, c[a0.x+3+VS_Bones], boneData.z
```

```
// Add weighting for bone 2
mad r3, c[a0.y+0+VS_Bones], boneData.w,r3
mad r4, c[a0.y+1+VS_Bones], boneData.w,r4
mad r5, c[a0.y+2+VS_Bones], boneData.w,r5
mad r6, c[a0.y+3+VS_Bones], boneData.w,r6

// Transform vertex
mul r2, inPos.x, r3
mad r2, inPos.y, r4, r2
mad r2, inPos.z, r5, r2
add r2, r6, r2

mul oPos, r2, c[VS_ProjScale]
```

```
D3DDevice::SetVertexShaderConstant( VS_Bones,  &bones, 140 );
D3DDevice::SetVertexShaderConstant( VS_ProjScale, &proj, 1 );
```

GLSL

```
attribute vec3 position;
attribute vec4 boneData;

uniform highp mat4 bones[35];
uniform highp vec4 projScale;

#define BONE_1 bones[boneData.x]
#define BONE_2 bones[boneData.y]
#define WEIGHT_1 boneData.z
#define WEIGHT_2 boneData.w

void main()
{
    highp vec4 inputPos = vec4(position, 1.0);
    mat4  combinedSkin = ( WEIGHT_1 * BONE_1 ) +
                         ( WEIGHT_2 * BONE_2 );
    highp vec4 oPos = combinedSkin * inputPos;
    gl_Position = ( oPos * projScale ).yxzw
}
```

```
glUniformMatrix4fv(locationOf(bones), 35, FALSE, &bones );
glUniform4fv( locationOf(projScale), 1, &proj );
```

Shader Conversion (cont'd)

- Microcode emulation

```
// Constant OFFSETS – Common Header
#define VS_Bones 0
#define VS_ProjScale 140

vs_1_1



dcl_position            v0
dcl_color               v1
#define inPos v0       // XYZ pos
#define boneData v1 // weights & indices




// Copy Bone Indices
mov a0.xy,      boneData.xy
```

```
// Build weighted matrix bone 1
mul r3 , c[a0.x+0+VS_Bones] , boneData.z
mul r4 , c[a0.x+1+VS_Bones] , boneData.z
mul r5 , c[a0.x+2+VS_Bones] , boneData.z
mul r6 , c[a0.x+3+VS_Bones] , boneData.z

// Add weighting for bone 2
mad r3 ,  c[a0.y+0+VS_Bones] , boneData.w, r3
mad r4 ,  c[a0.y+1+VS_Bones] , boneData.w, r4
mad r5 ,  c[a0.y+2+VS_Bones] , boneData.w, r5
mad r6 ,  c[a0.y+3+VS_Bones] , boneData.w, r6

// Transform vertex
mul r2 , inPos.x , r3
mad r2 , inPos.y , r4, r2
mad r2 , inPos.z , r5, r2
add r2 , r6 , r2

mul oPos , r2 , c[VS_ProjScale]
```

```
D3DDevice::SetVertexShaderConstant( VS_Bones, &bones, 140 );
D3DDevice::SetVertexShaderConstant( VS_ProjScale, &proj, 1 );
```

Shader Conversion (cont'd)

- Microcode emulation

```
// Constant OFFSETS - Common Header
#define VS_Bones 0
#define VS_ProjScale 140

#define CONST_ARRAY_SIZE VS_ProjScale + 1
uniform highp vec4 c[CONST_ARRAY_SIZE];
#define oPos gl_Position

attribute mediump vec3 v0;
attribute mediump vec4 v1;
#define inPos v0      // XYZ pos
#define boneData v1 // weights & indices

void main()
{
    lowp ivec2 a0;
    mediump vec4 r2, r3, r4, r5, r6;

// Copy Bone Indices
mov a0.=yivec2( boneData.xy );
```

```
// Build weighted matrix bone 1
mul r3 = c[a0.x+0+VS_Bones] * boneData.z;
mul r4 = c[a0.x+1+VS_Bones] * boneData.z;
mul r5 = c[a0.x+2+VS_Bones] * boneData.z;
mul r6 = c[a0.x+3+VS_Bones] * boneData.z;

// Add weighting for bone 2
mad r3 += c[a0.y+0+VS_Bones] * boneData.w; r3
mad r4 += c[a0.y+1+VS_Bones] * boneData.w; r4
mad r5 += c[a0.y+2+VS_Bones] * boneData.w; r5
mad r6 += c[a0.y+3+VS_Bones] * boneData.w; r6

// Transform vertex
mul r2 = inPos.x * r3 +
mad r2 , inPos.y * r4,+r2
mad r2 , inPos.z * r5,+r2
add r2 , r6;, r2

mul oPos = r2 * c[VS_ProjScale];
}
```

```
D3DDevice::SetVertexSh glUniform4fv( VS_Bones, &bones, 140 );
D3DDevice::SetVertexSh glUniform4fv( VS_ProjScale, 1, &proj );
```

Shader Conversion (cont'd)

- Microcode emulation

```
// Constant OFFSETS – Common Header
#define VS_Bones 0
#define VS_ProjScale 140

#define CONST_ARRAY_SIZE VS_ProjScale + 1
uniform highp vec4 c[CONST_ARRAY_SIZE];
#define oPos gl_Position

attribute mediump vec3 v0;
attribute mediump vec4 v1;
#define inPos v0      // XYZ pos
#define boneData v1 // weights & indices

void main()
{
    lowp ivec2 a0;
    mediump vec4 r2, r3, r4, r5, r6;

// Copy Bone Indices
    a0 = ivec2( boneData.xy );
```

```
// Build weighted matrix bone 1
    r3 = c[a0.x+0+VS_Bones] * boneData.z;
    r4 = c[a0.x+1+VS_Bones] * boneData.z;
    r5 = c[a0.x+2+VS_Bones] * boneData.z;
    r6 = c[a0.x+3+VS_Bones] * boneData.z;

// Add weighting for bone 2
    r3 += c[a0.y+0+VS_Bones] * boneData.w;
    r4 += c[a0.y+1+VS_Bones] * boneData.w;
    r5 += c[a0.y+2+VS_Bones] * boneData.w;
    r6 += c[a0.y+3+VS_Bones] * boneData.w;

// Transform vertex
    r2 = inPos.x * r3 +
         inPos.y * r4 +
         inPos.z * r5 +
         r6;

    oPos = r2 * c[VS_ProjScale];
}
```

```
glUniform4fv( VS_Bones, &bones, 140 );
glUniform4fv( VS_ProjScale, 1, &proj );
```

Audio Runtime

- Originally developed using XACT on XBOX
- OpenAL maps to DirectSound (well… almost)

OP	DirectSound	OpenAL
Buffer Creation	DirectSound*::CreateSoundBuffer + Lock	alGenBuffers + alBufferData(Static+)
Playback	DirectSoundBuffer::(Play|Pause|Stop)	alSource(Play|Pause|Stop)
Volume/Pitch	DirectSoundBuffer::Set(Volume|Pitch)	alSourcef (AL_GAIN | AL_PITCH)
Status	DirectSoundBuffer::GetStatus DirectSoundBuffer::GetCurrentPosition	alGetSource(i|f) (AL_SOURCE_STATE | AL_*_OFFSET)
Spatial Parameters	DirectSound3DBuffer::SetAllParameters	alSource(i|f)
Reverb	IDirectSoundFXI3DL2Reverb::SetParameters	**Not directly supported**
Streaming	Directly to looping buffer memory	Multi buffer queue alSource(Queue|Unqueue)Buffers
Notifications	DirectSoundNotify::SetNotificationPositions	**Not implemented but status can be polled**
Limits	**256 channels on XBOX**	**32 active sources**

Touch Controls

- Handle touches directly?
  - Lots of game dependencies on input
  - New/modified game code, more testing
- Context sensitive controls?
  - Translate touches to original gamepad state
  - Game code remains largely the same

Game runs: Now What?

- Sluggish (especially on older devices)
- Too LARGE:
5.8GB > 2GB
(unzipped App Size limit for App Store)

Target Frame Rate

- 60 Hz?
  - Terrible for battery life
- 30 Hz?
  - Game may depend on running at 60Hz
- Hybrid
  - 60Hz update rate: functionality intact
  - 30Hz render rate: low GPU and battery usage
  - 60Hz optionally on 5th gen+ devices

CPU Optimization

- Misaligned vertex attributes
- App is ARMv7 specific
- NEON support in floating point co-processor
- Many times faster than VFP
  - True SIMD vs. sequential scalar
  - Dual (Cortex A8) or Quad (Apple A4/A5)
  - Beware of pipeline stalls

Candidates for NEON SIMD

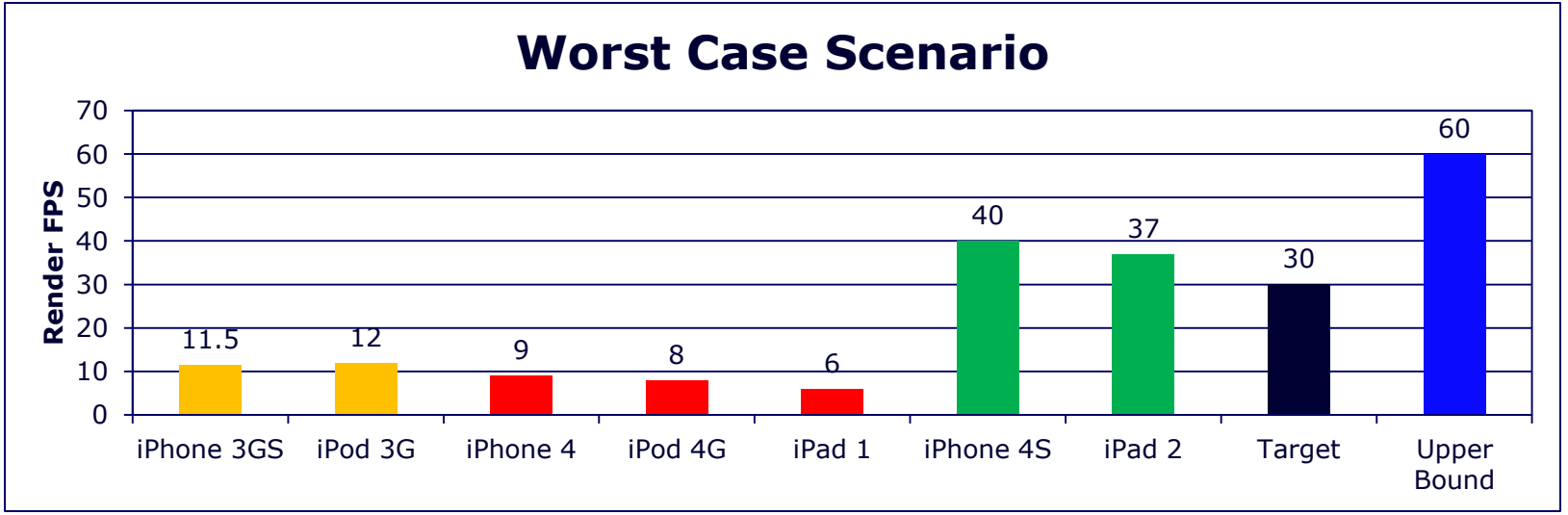
- Math library
- Frustum Culling
- Codecs
- CPU vertex shading
  - Skinning
  - Procedural vertex animation and effects

SGX GPUs: a Few Stats

- 3rd gen devices: SGX 535
- 4th gen devices: SGX 535 (faster clock)
  - x2 resolution or x4 pixels (x5 on iPad1)
- 5th gen devices: SGX 543 MP2
  - “x7 faster”, no problem

The Bard's Tale worst case metrics

- 120k polygons, 72k skinned, 2 weights
- All features, full screen resolution



Rendering Optimization

- Basics covered?
  - Minimal vertex format size
  - Texture size and mipmapping
- Alpha Test and the SGX
  - Fragment Discard is possibly most expensive operation on SGX
  - Huge impact on fill rate
  - Use alpha blend at all costs

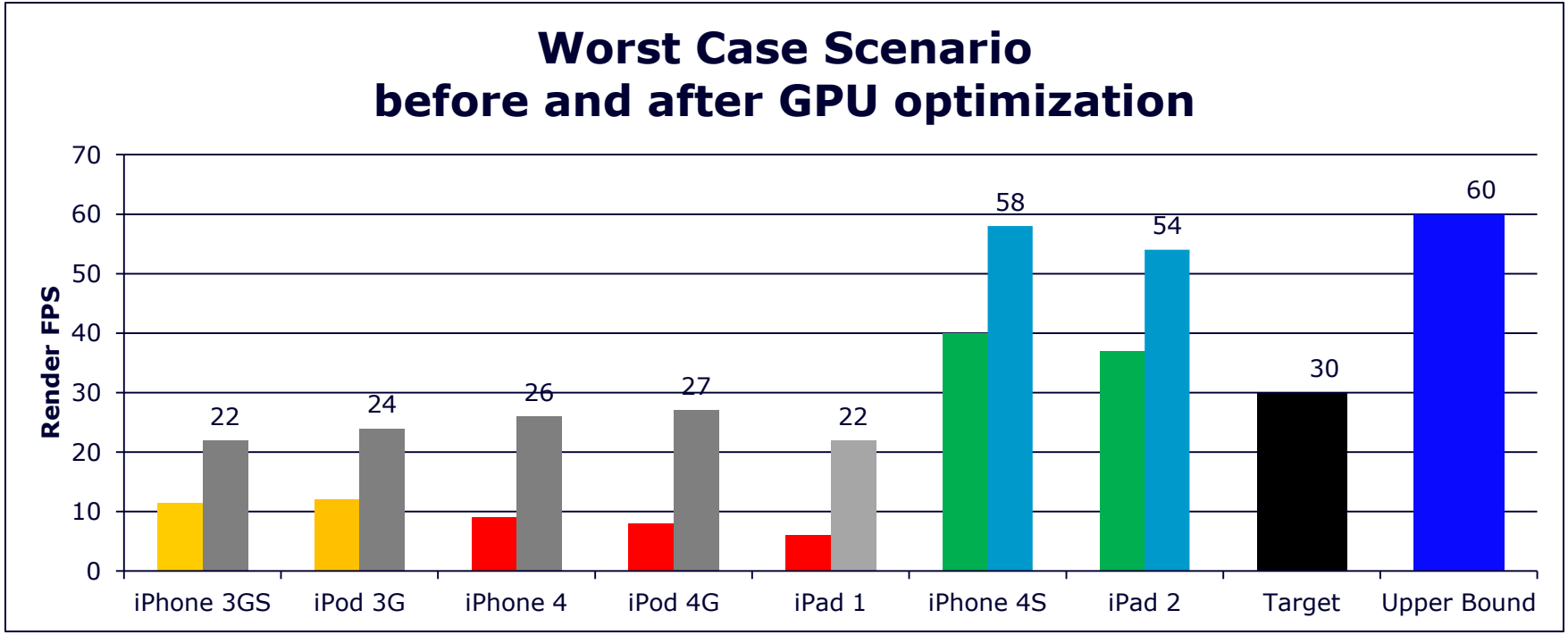
Eliminating Alpha Testing

- Replace with Alpha Blend – sorting issues
- Automated material separation
  - Evaluate each polygon's pixel coverage
  - Separate into 2 sets: opaque and translucent
- Load time triangle sorting
  - Resort all triangles based on view heuristics
  - Render mesh with alpha blend and depth write

Rendering "Optimization"

- Per device configurations
  - Less effects, less polygons
  - Reduce render targets
  - Shader LODs on lower end devices
    - Rigid/Smooth skinning
    - Less lights
  - Half native resolution

Results


Worst Case Scenario
before and after GPU optimization
Render FPS
70
60
50
40
30
20
10
0
22
24
26
27
22
58
54
30
60
iPhone 3GS
iPod 3G
iPhone 4
iPod 4G
iPad 1
iPhone 4S
iPad 2
Target
Upper Bound

Memory usage reduction

- 3rd & 4th Gen devices: 40-50 MB
- 5th Gen devices: 80-100 MB
- Fast load times
  - Hot load rarely used permanent assets
  - Flush texture caches/audio on memory warnings
- Downsampling and NPOT textures

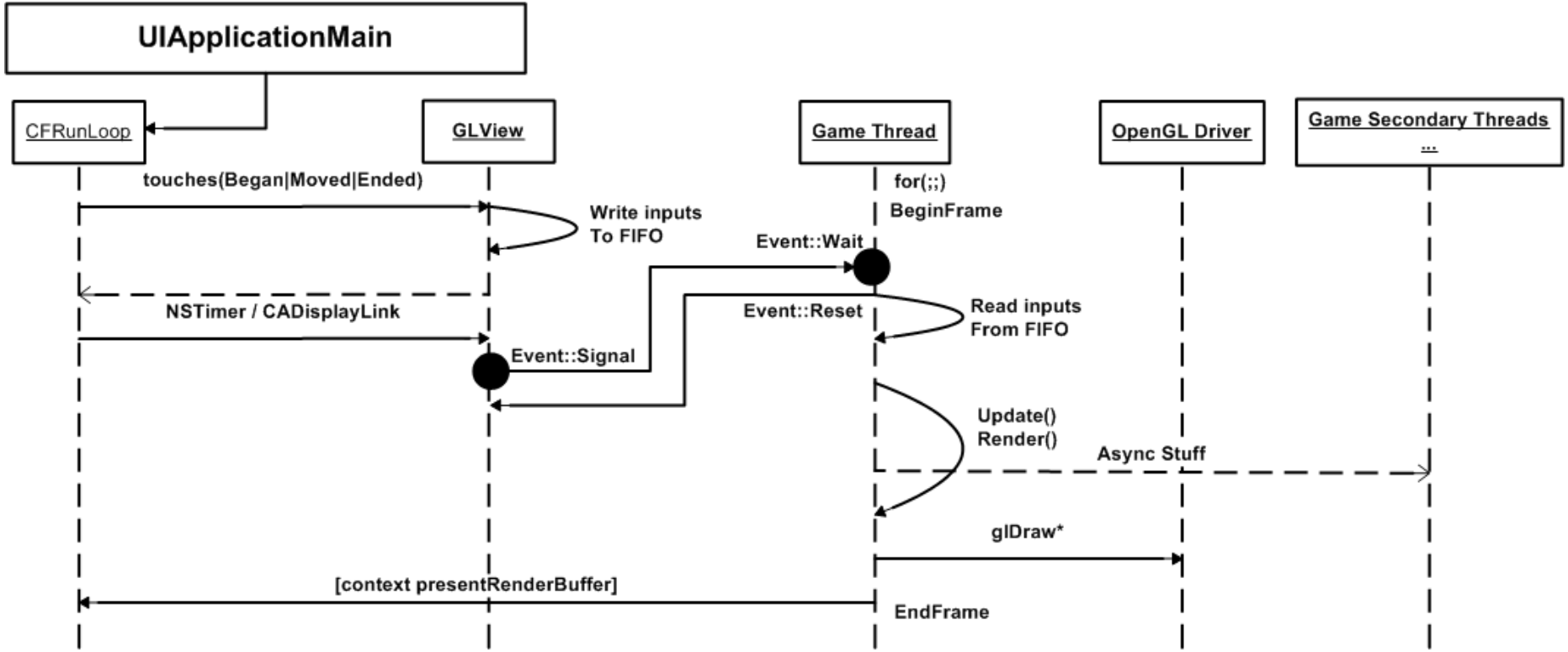
App Bundle Size Reduction

- Remove content?
  - Rather not if possible
- Reduction strategies
  - OGG compression
  - Texture Compression and reduction
  - Mipmap chain removal
  - Remove DVD-ROM redundant data

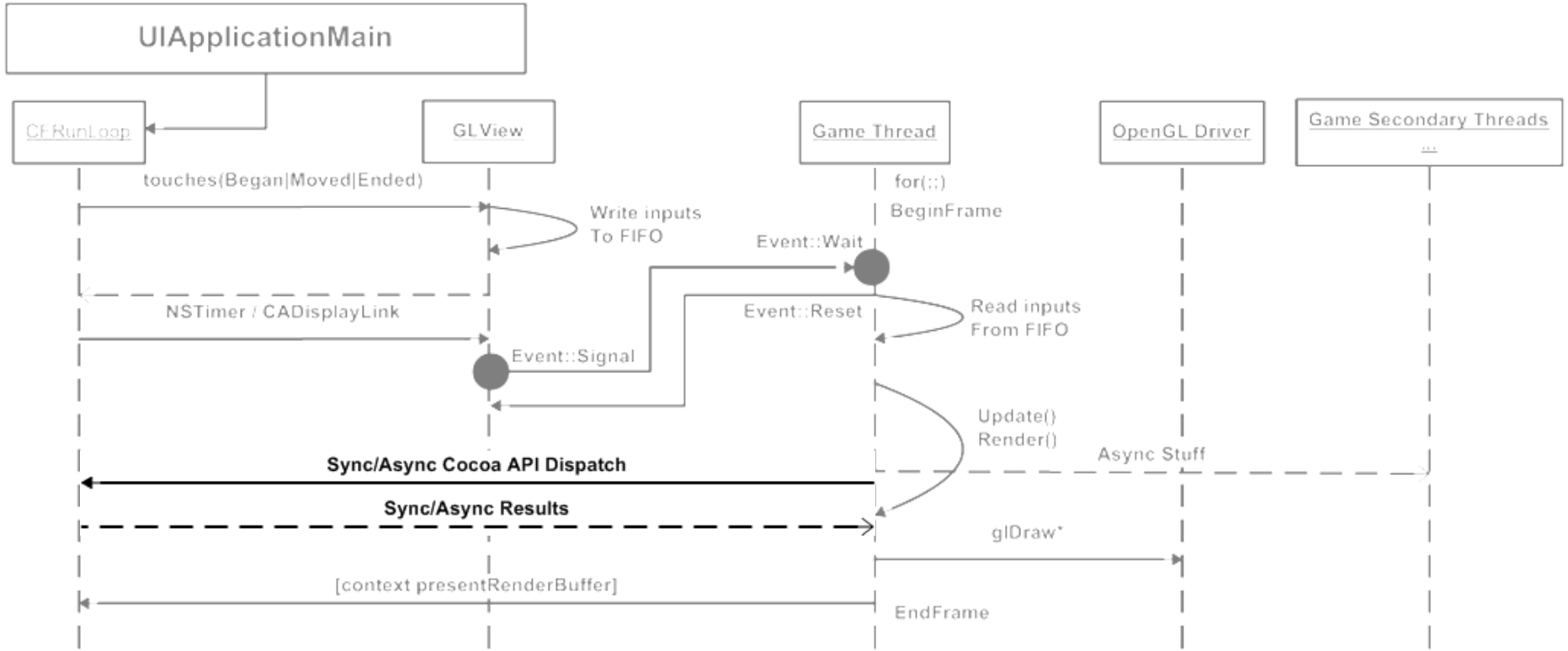
Adding iOS specific features

- Requires Objective C libraries
- Requires calls to be made on main thread
  - GameCenter/OpenFeint
  - In-App Purchase
  - TapJoy
  - DLC using NS* networking
  - iCloud

Re-integrating Cocoa technologies

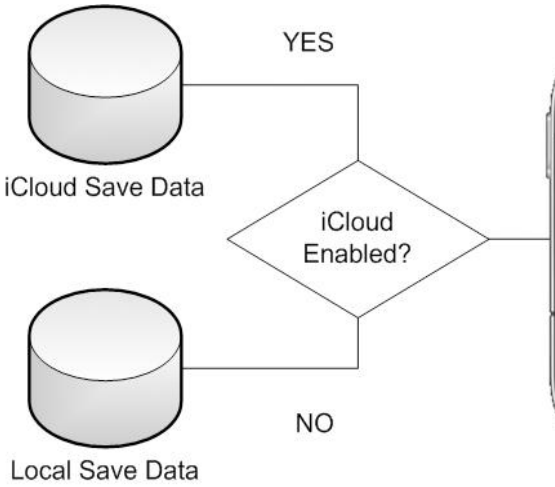


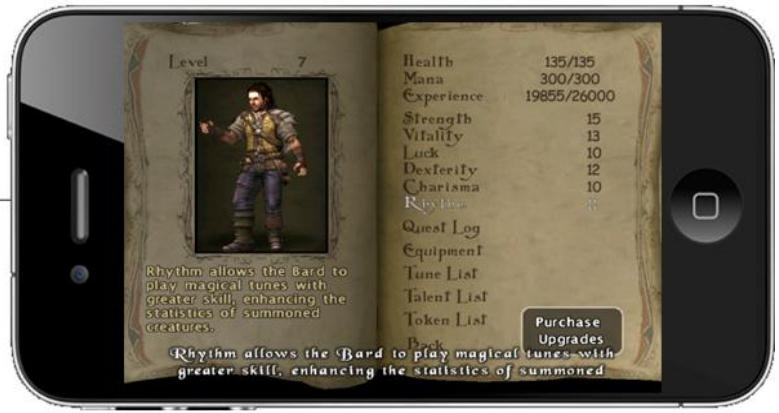
Re-integrating Cocoa technologies



iCloud

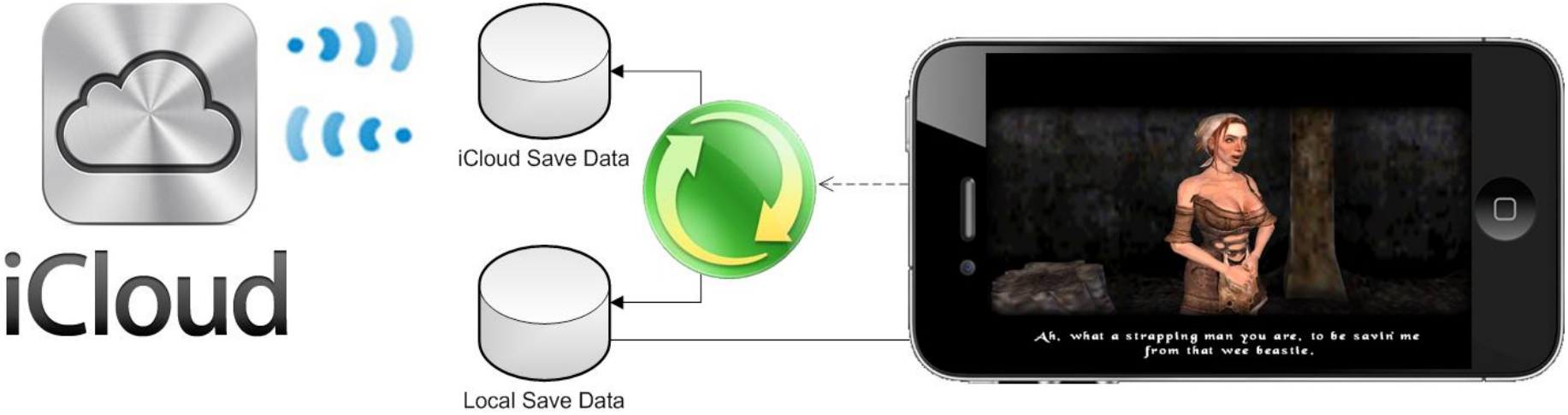
- Using iCloud File System Directly
  - New complications
  - Hybrid solution to support non-iCloud devices





iCloud (cont'd)

- Using iCloud as a dropbox
  - Independent of game's save data system
  - Can be culled at high level



Stéphane Khalil Jacoby
CTO / Co-Founder @ SQUARE 1 GAMES
stephane@s1games.com