NATIVE GAME DEVELOPMENT WITH COCOS2D-X

NHN Entertainment
Juhyun Yun
CONTENTS

1. What is Game Engine?
   1.1. Famous Game Engines for Android
   1.2. Differences between each Game Engines
   1.3. Why Cocos2d-x?

2. Cocos2d-x Rendering Pipeline
   2.1. Game Loop and Rendering Pipeline
   2.2. Event Processing
   2.3. Rendering Optimization

3. UI Framework Comparison between Android and Cocos2d-x
   3.1. Rendering Architecture Comparison
   3.2. Rendering Components
   3.3. Conclusion
WHAT IS GAME ENGINE?

Cocos 2D

Unreal Engine

Unity

CryEngine
I. WHAT IS GAME ENGINE?

• Game Engine
  Software framework designed for the creation and development of video games.

• Game Engine has
  • rendering engine (“renderer”)
  • physics engine or collision detection(response)
  • sound
  • scripting
  • animation
  • AI (Artificial Intelligence)
  • Networking
  • Streaming
  • Memory management
  • Threading
  • Localization Support
  • Scene Graph
1.1. FAMOUS GAME ENGINES FOR ANDROID

- Unity3D
- CryEngine
- Unreal Engine
- Corona SDK
- Cocos2D
1.2. DIFFERENCES BETWEEN EACH GAME ENGINES

- Focus on whether 2D or 3D
- What languages for Scripting
- Support Cross-Platform?
- Support Integrated Development Environment?
- Support Profiler?
- Other Third-Party Libraries.
## 1.2. Differences Between Each Game Engines

Focus on Cocos2d-x & Unity3D

<table>
<thead>
<tr>
<th></th>
<th>Cocos2d-x</th>
<th>Unity3D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>2D or 3D</strong></td>
<td>2D</td>
<td>3D</td>
</tr>
<tr>
<td><strong>Language for Scripting</strong></td>
<td>C++</td>
<td>C# / Javascript / Boo</td>
</tr>
<tr>
<td><strong>Base Runtime</strong></td>
<td>Native (NDK)</td>
<td>Mono (.NET 2.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(convert to C++ by IL2CPP for iOS 64)</td>
</tr>
<tr>
<td><strong>IDE</strong></td>
<td>Cocos Studio</td>
<td>UnityEditor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MonoDevelop or Visual studio</td>
</tr>
<tr>
<td><strong>Profiler</strong></td>
<td>Not Support (ARM DS-5 Streamline)</td>
<td>Support UnityEditor</td>
</tr>
<tr>
<td><strong>Third-Party Libs</strong></td>
<td>C++ Libs</td>
<td>Asset Store</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Free or Paid)</td>
</tr>
</tbody>
</table>
1.2. DIFFERENCES BETWEEN EACH GAME ENGINES DEVELOPMENT ENVIRONMENTS
1.3. WHY COCOS2D-X?

- OpenSource (MIT License)
- Optimized for 2D Graphics
- Cross Platform
- Simple Native API (NDK, C++)
- Integration for Third-Party Project.
- Smaller Build Size than Other Engines.
- Fast Running.
COCOS2D-X
RENDERING
PIPELINE
2. COCOS2D-X RENDERING PIPELINE

• A Simple Game Loop

- Run
- Startup ➔ Update ➔ Stop ➔ Shutdown
- Loop ➔ Draw

• A Advanced Game Loop

- Run
- Startup ➔ Update ➔ Stop ➔ Shutdown
- Resync ➔ Wait
  - Yes ➔ Are we significantly behind?
  - No ➔ Draw
- Have we rendered a frame recently?
  - No ➔ Do we have extra time?
  - Yes ➔ Draw
- Yes ➔ Have we rendered a frame recently?
2.1. GAME LOOP & RENDERING PIPELINE

- **Director**
- **Scene**
- **EventDispatcher**
- **GLView**
- **Scheduler**
- **Renderer**
- **ActionManager**

**Cocos2dxRenderer**

1. `mainLoop()`
2. `update()`
3. `render()`
4. `visit(nodes)`
5. `render()`
6. `drawScene`
7. `render()`
8. `visit(nodes)`
9. `render()`
10. `visit() & draw()`

- `visit()` & `draw()`
- `pollEvents()`
- `swapBuffers()`
- `Add Command`
2.1. GAME LOOP & RENDERING PIPELINE

Renderer

visitRenderQueue

RenderQueue

3D Objects

2D Objects

std::function

Batch

without VBO

Batch

- global order (for Sorting RenderQueue)
- Bind Texture
- Set Blend Func & Shader & ModelView Transform

RenderCommand

TrianglesCommand

QuadCommand

GroupCommand

CustomCommand

BatchCommand

PrimitiveCommand

MeshCommand

Call OpenGL Functions
When called `Flush()` function
2.2. EVENT PROCESSING

**Touch** : `onTouchEvent()`

- **MotionEvent**
  - A. Down
  - B. Move
  - C. Up
  - D. Cancel

**Cocos2dxGLSurfaceView**

- **Runnable**
  - `handleActionXXXX()`
  - `Cocos2dxRenderer`
  - `handleTouchxxx()`

- **GL View**
  - `TouchEvent`
  - `TouchEventDispatcher`

- **GL Thread Queue**
  - Postfix
    - A. Down
    - B. Move
    - C. Up
    - D. Cancel
  - `Params`
    - A. `point count`
    - B. `touch id`
    - C. `point x`
    - D. `point y`
2.3. RENDERING OPTIMIZATION

- **Decouple the scene graph from the renderer**
  Visiting nodes issues graphics commands and put them on a queue, but doesn’t actually invoke any OpenGL rendering code.

- **Viewing frustum Geometry culling**
  Sprites (and geometries more in general) not visible from the camera’s point of view be automatically removed from the current frame and not rendered.

- **Rendering on a thread**
  The execution of all the rendering commands (i.e. OpenGL calls) will be moved to different thread than the main one (this will allow for better parallelism and usage of more than one CPUs cores where possible).

- **Automatic batching**
  Efficiently reduces the number of draw calls (automatically) batching them together when possible (i.e. sprites using the same material).

- **(Node based) Customizable rendering**
  As in the current version of Cocos, users will still be able (if needed) to customize rendering on a per node basis, calling OpenGL commands directly, disregarding the official renderer (but possibly incurring on worst performances).

- **Optimized for 2D, but suitable for 3D as well**
  The new renderer should be optimized for 2D games, but it should be capable of handling 3D objects as well.
2.3. RENDERING OPTIMIZATION

MULTI-THREAD (1)

Rendering nodes during visit, will not call into OpenGL anymore. Cocos graphics commands will be added to a CommandQueue instead de facto decoupling the scene graph from the rendering.
2.3. RENDERING OPTIMIZATION

MULTI-THREAD (2)

SingleThread

Frame
update | visit | render | update | visit | render

Frame

MultiThread

1. update | visit | render
2. update | visit | render
3. update | visit | render
4. update | visit | render
2.3. RENDERING OPTIMIZATION

AUTOMATIC BATCHING

• The Quads are not required to share the same Material. But if all the Quads share the same material, then only one draw call will be used (automatic batching).

• A very simplified version of automatic batching algorithm works like this:

1. A new Quad is added to the Buffer
2. If the new Quad shares the same Material as the previous Quad, nothing happens.
3. However, if the new Quad has a different Material than the previous Quad, then the previous Quads are drawn.
2.3. RENDERING OPTIMIZATION

AUTOMATIC CULLING

• If the AABB (Axis-Aligned Bounding Box) of the node is outside the frustum, then no Commands will be submitted to the RenderQueue.

• This will be performed in the cocos2d main thread.
UI FRAMEWORK COMPARISON BETWEEN ANDROID AND COCOS2D-X
3. UI FRAMEWORK COMPARISON

- **Looper**
  - dispatchMessage()

- **Message Queue**
  - Render Queue

- **Activity Thread**

- **Handler**
  - dispatchMessage()

- **Choreographer**
  - dispatchMessage()
  - Cocos2dxRenderer
    - onDrawFrame()

- **Activity**
  - Window

- **Views**
  - Director
    - drawScene()
  - Scene
## 3.1. Rendering Architecture Comparison

<table>
<thead>
<tr>
<th>Android</th>
<th>Cocos2d-x</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageQueue</td>
<td>RenderQueue</td>
</tr>
<tr>
<td>Looper</td>
<td>Renderer</td>
</tr>
<tr>
<td>Choreographer</td>
<td>Cocos2dxRenderer (implements GLSurfaceView.Renderer)</td>
</tr>
<tr>
<td>Activity</td>
<td>Director(?)</td>
</tr>
<tr>
<td>CustomView</td>
<td>Node(Sprite)</td>
</tr>
</tbody>
</table>
3.2. RENDERING COMPONENTS

(1) MESSAGEQUEUE ↔ RENDERQUEUE

• MessageQueue (Android)
  • Low-level class holding the list of messages to be dispatched by a Looper.
  • Messages are not added directly to a MessageQueue, but rather through Handler objects associated with the Looper.
3.2. RENDERING COMPONENTS

(2) MESSAGEQUEUE \leftrightarrow \textsc{RENDERQUEUE}

- RenderQueue (Cocos2dx Native)
  - A class holding the list of rendering commands to be executed by a renderer.
  - Sort rendering commands by GlobalOrder (that means depth)
    - A. If global orders have positive or negative value, that are sorted by global order.
    - B. If global orders have zero value, that aren’t sorted.
      (because queue index is the ordering)
3.2. RENDERING COMPONENTS

LOOPER ↔ RENDERER

- Looper

- Class used to run a message loop for a thread.

- Threads by default do not have a message loop associated with them.

- Most interaction with a message loop is through the Handler class.

```java
class LooperThread extends Thread {
    public Handler mHandler;

    public void run() {
        Looper.prepare();

        mHandler = new Handler() {
            public void handleMessage(Message msg) {
                // process incoming messages here
            }
        };
       Looper.loop();
    }
}
```
3.2. RENDERING COMPONENTS

LOOPER ↔ RENDERER

- Renderer
  - Add Render Commands into the RenderQueue
  - Get Render Commands from RenderQueue and render the nodes.
3.2. RENDERING COMPONENTS

**CHOREOGRAPHER ↔ COCOS2DXRENDERER**

- Choreographer
  - Coordinate the timing of animations, input and drawing
  - The choreographer receives timing pulses (such as vertical synchronization) from the display subsystem then schedules work to occur as part of rendering the next display frame.
  - Each Looper thread has its own choreographer. Other threads can post callbacks to run on the choreographer but they will run on the Looper to which the choreographer belongs.
3.2. RENDERING COMPONENTS

CHOREOGRAPHER ↔ COCOS2DXRENDERER

- Cocos2dxRenderer
  - Create GLView object & Setting frame size
  - Call native functions from SurfaceView like kind of touch & activity event.
  - Call `onDrawFrame()` method (sleep by interval)

```java
@Override
public void onDrawFrame(final GL10 gl) {
    /*
    * No need to use algorithm in default(60 FPS) situation,
    * since onDrawFrame() was called by system 60 times per second by default.
    */
    if (sAnimationInterval <= 1.0 / 60 * Cocos2dxRenderer.NANOSECONDSPERSECOND) {
        Cocos2dxRenderer.nativeRender();
    } else {
        final long now = System.nanoTime();
        final long interval = now - this.mLastTickInNanoSeconds;

        if (interval < Cocos2dxRenderer.sAnimationInterval) {
            try {
                Thread.sleep((Cocos2dxRenderer.sAnimationInterval - interval) / Cocos2dxRenderer.NANOSECONDSPERMICROSECOND);
            } catch (Final Exception e) {
                }
            } /*
            * Render time MUST be counted in, or the FPS will slower than appointed.
            */
            
            this.mLastTickInNanoSeconds = System.nanoTime();
            Cocos2dxRenderer.nativeRender();
```
3.2. RENDERING COMPONENTS

**ACTIVITY? ↔ DIRECTOR**

- Activity
  - An activity is a single, focused thing that the user can do.
  - Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
}
```
3.2. RENDERING COMPONENTS

ACTIVITY? ↔ DIRECTOR

- Director
  - Cocos2dx Process has only one Director instance (Singleton)
  - Initializing the OpenGL.
  - Setting the orientation.
  - Director has a Scene Stack (Getting a running scene is available)
  - Draw Scene that is running. (in mainLoop())
  - Manage Scenes.
    - transition
    - start & stop animation
3.2. RENDERING COMPONENTS

**CUSTOMVIEW** ↔ **NODE(SPRITE)**

- Custom View
  - Custom View is extended from View
  - Refresh an view area when invalidate situations.
  - Call dispatchDraw(Canvas) method
3.2. RENDERING COMPONENTS

CUSTOMVIEW ↔ NODE(SPRITE)

- Node (Sprite)
  - Node is the base element of the Scene Graph.
  - Elements of the Scene Graph must be Node objects or subclasses of it.
  - Nodes can contain other Node objects
  - Nodes can schedule periodic callback
  - Nodes can execute actions
  - Node has lots of graphic properties. (z-order, scale, position, skew, anchor-point, rotation, opacity)
- Limitation
  If you want to draw something on the screen, you should use a Sprite instead.
  Or subclass Node and override ‘draw()’ function.
3.3. CONCLUSION

- Cocos2dx utilize Android GLSurfaceView.
  - for Rendering a Scene (Frame-Rate Rendering)
  - for TouchEvent
- Cocos2dx is a good android practice for native rendering using OpenGL. (If you want to implement some apps by using OpenGL and NDK, You can reference the structure of cocos2dx)
- If you want to draw a simple frame-rate animation in a View, It is better using SurfaceView with Java
REFERENCES

- Multi-Thread Rendering
  http://www.panda3d.org/blog/triple-your-frame-rate/

- The Game Loop

- Auto-Batching

- cocos2dx reference
  http://www.cocos2d-x.org/programmersguide

- Performance improvements in cocos2d-x v3.0
  http://malideveloper.arm.com/downloads/GDC14/Weds/12pmCocos2d-x.pdf