3D UI development on Android

Austin Lai[#1], Patrick Tsai[*2]

WCP2/OSS2/AP1, Mediatek
No. 15, Lane 91, Sec. 1, Neihu Rd.,
Taipei City, Taiwan, R.O.C
[1]austin.lai@mediatek.com
[2]patrick.tsai@mediatek.com

***Abstract*—Google android is the most popular smart phone open source system in the world. With more powerful CPU and GPU, fantasy wonderful 3D vision and interaction can be realized in Android system. We construct a 3D UI development tool kit and use it to implement 3D UI apps on Android. We adopt "Irrlicht", an open source high performance real-time 3D engine written and usable in C++, as our native 3D engine. Base on Irrlicht, we development 'ngin3d', a Java 3D framework that provides high level 3D API for apps and communicates with Irrlicht by JNI. Based on ngin3d, we generate an apps call 'Media3D' that shows weather, photo, and video content in 3D world. This paper describes what we implement and how we design 3D UI in Android system. The 3D engine architecture we design is suitable for Android system. At the end of this document, we will show the 3D UI effect in Android UI.**

***Keywords*—Android, OpenGL ES, Irrlicht, 3D engine, 3D UI**

I. Introduction

With the popular touch screen and the mature 3D graphic technology, the demand of novel user interface for handheld devices increases stringently by consumers. In 3D user interfaces (3DUIs), users interact with a computer with an aspect of three-dimensional space. It provides more intuitive interface and interactive user experience, which brings user a fresh feel.

MT6573 platform incorporates the highly-integrated core chipset including applications processing system: a 650 MHz ARM® 11subsystem and an advanced 3D graphics: PowerVR Series5 SGX GPU from Imagination Technologies. This platform supports the latest versions of the Android operating system. 3DUI project starts on this innovative platform.

Until now Google doesn't provide any 3D UI framework for android app developers. In order to avoid using low-level graphics APIs such as OpenGL directly and increase 3D apps coding efficiency, we need to construct our own 3D UI Java framework and use it to write fantasy apps on Android with high performance.

II. 3D UI Engine Architecture

Basically the software architecture of 3D UI has three layers. The bottom layer is Irrlicht, a native high performance real-time 3D engine using OpenGL/OpenGL ES[1][2][6]. The middle layer is 3D UI SDK written by Java, which provides rich APIs for apps. The top layer is app logics that consist of Android[3] and ngin3d framework. Figure 1 illustrates the framework of the three layers. Let start with Irrlicht.

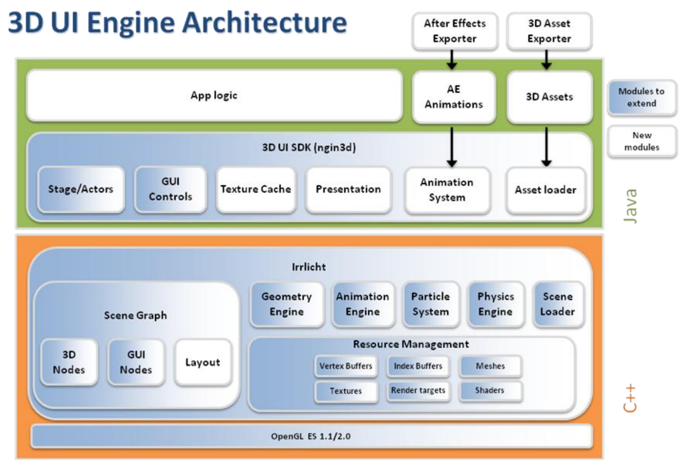

Fig. 1 3D UI Engine Architecture

A. *Irrlicht*

The Irrlicht Engine is a cross-platform high performance real-time 3D engine written in C++. It provides high level APIs for creating complete 3D and 2D scenes. It comes with an excellent documentation and integrates all state-of-the-art features for visual representation which can be found in commercial 3d engines, such as dynamic shadows, particle systems, character animation, indoor and outdoor technology, and collision detection. All of them are accessible through a well-designed C++ interface, which is extremely easy to use.

Its main features are:

1) *Special effect*: There are lots of common special effects available in the Irrlicht Engine. They are not difficult to use, in most cases the programmer only need to switch them on. The engine is constantly extended with new effects, for example, animated water surfaces, dynamic lights, dynamic shadows using the stencil buffer and transparent objects, etc.

2) *Materials and Shaders*: To be able to create realistic environments quickly, there are lots of common built-in materials available in the engine. Some materials are based on the fixed function pipeline (light mapped geometry for example) and some are relying on the programmable pipeline (normal mapped/parallax per pixel lighted materials for example) which is offered by current 3d hardware. It is possible to mix these types of materials in a scene without interference. When certain features are required but not feasible in the hardware, the engine provides **fall back**

materials. However, if the built-in materials are not enough, it is possible to add new materials to Irrlicht at runtime without the need of modifying or recompiling the engine. Currently supported shader languages are: Pixel and Vertex shavers 1.1 to 3.0, HLSL, GLSL.

3) *Scene Management*: Rendering in the Irrlicht Engine is done using a hierarchical scene graph. Scene nodes are attached to each other and follow each other's movements, cull their children to the viewing frustum, and are able to do collision detection. A scene node can for example be a camera, an indoor or outdoor level, a skeletal animated character, animated water, a geometric mipmap terrain, or something completely different. In this way, the Irrlicht engine can seamlessly mix indoor and outdoor scenes together, gives the programmer full control over anything which is going on in the scene. It is very easily extensible because the programmer is able to add his own scene nodes, meshes, texture loaders, GUI elements, and so on. The geometry creator gives easy access to simple geometrical bodies, such as cylinder, cube, etc. Objects can be rendered as polygons, wireframe, or points, using triangles, lines, point and point sprite primitives.

4) *Others*: Lots of common file formats are supported, and are able to be loaded (and partially also written) directly from within the engine. Irrlicht supports all general render features needed for high quality rendering of materials and effects. Besides a few exceptions, all features are supported in all hardware accelerated APIs, and some are supported by the software renderers as well. Irrlicht has so much features and advantage, the most important is its completely free and open source. We can port and enhance it on Android as wish.[4]

B. *ngin3d*

The ngin3d is a Java layer 3D framework developed for Android platform. It provides a powerful high level API for creating complete 3D and 2D Android applications. It company with all state-of-the-art features such as animation system, stage/actor model, GUI control, texture atlas, texture cache, presentation architectures, and dragging animation, etc..

Following describes main features that ngin3d supports:

1) *Animation system*: There are two kinds of animation in ngin3d. One is property animation and the other is key-frame animation. A property animation changes a property's (a field in an object) value over a specified length of time. To animate something, you specify the object property that you want to animate, such as an object's position on the screen, how long you want to animate it for, and what values you want to animate between. A key-frame animation changes a property's value by couples of key value/time map. There is a Key-frame interpolator that helps to get correct value at specific time. Currently ngin3d supports loading key-frame data from JSON (JavaScript Object Notation) [12] format file that output by After Effect. As figure 1 show, the artists use After Effect to design any novel 3D animation, and then export the animation to JSON format files by After Effect plug-in that written by us, finally loading animations by animation loader.

2) *Stage / Actor model:* As figure 2 presents, in ngin3d, the contents it will display are stored as a tree of Actor objects. The most common actors are Image and Text that are used to display image and text as their names imply. A special actor, called Container, can contain other actors as its children. All non-leaf nodes in the tree are all Container objects. The root of Actor tree is a special container called Stage: Actors are organized in tree structure because some operations applied to the parent node can be applied to all its child nodes. For example, an actor can be scaled and its children will be scaled also.

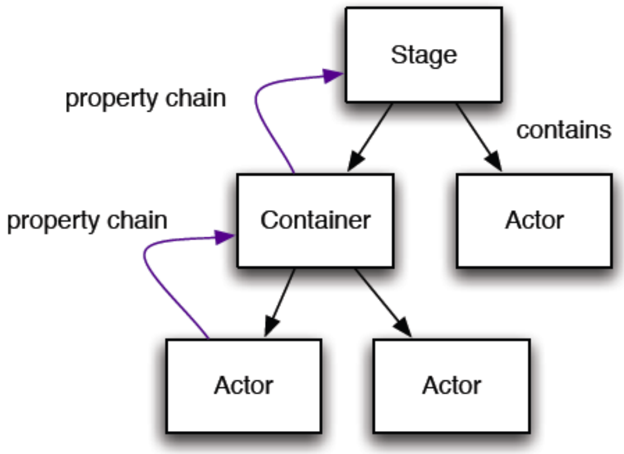

Fig. 2 Actor tree

Actor is the base class of all actor classes. Various derived classes are added for displaying different contents such as image and text. Some classes, such as Canvas2D and Canvas3D, are designed to provide extensibility for custom 2D and 3D content. Actor has various *properties* that can be modified to change the visual presentation and the content it's displayed. The modifications of properties are typically done in UI thread and they typically won't be applied immediately because the rendering is done in render thread. In section "*C. Rendering Architecture"* we will explain this concept more detail.

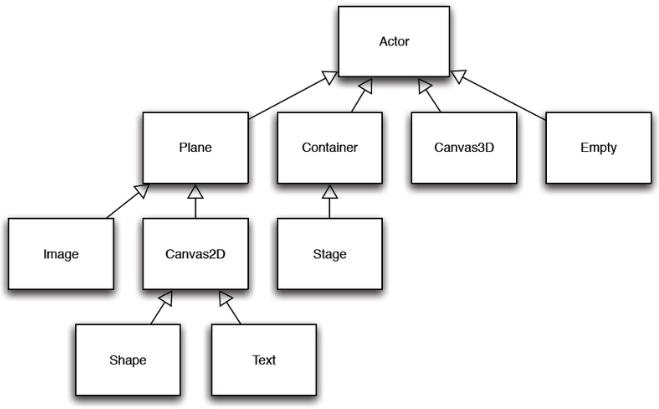

Fig. 3 Actor class hierarchy

3) *Transaction:* Every modification to a property is a basic transaction. Ngin3d provides *Transaction* class that is

responsible for comprising multiple property modifications into atomic updates to the render tree.

4) *GUI controls:* The user interaction with 3D object is done by collision system in Irrlicht. When user hitting on the screen, ngin3d will convert the hit point to the 3D world coordinates and send the information to Irrlicht to determine which object be hit.

5) *Texture Atlas:* A texture atlas is a large image, or "atlas" which contains many smaller sub-images, each of which is a texture for some part of a 3D object. The sub-textures can be rendered by modifying the texture coordinates of the object's uv map on the atlas, essentially telling it which part of the image its texture is in. In an application where many small textures are used frequently, it is often more efficient to store the textures in a texture atlas which is treated as a unit by the graphics hardware. In particular, because there are less rendering state changes by binding once, it can be faster to bind one large texture once than to bind many smaller textures as they are drawn.[7]

6) *Texture Cache:* A texture cache that maps texture source (file, resource, or anything else) to their texture object in memory. This feature can prevent loading multiple textures for the same 'source'.

7) *Presentation:* Presentation is used to separate stage layer and Irrlicht render scene tree. It makes stage/actor model independence with native Irrlicht hierarchical scene node graph. If one day we need to replace Irrlicht by other native 3D engine, we can keep the implementation of stage/actor model and just update the presentation implementation. Please reference figure 4, you can find Presentation is the bridge of Stage/Actor and Irrlicht scene graph, it play a major role in ngin3d render architecture.

C. *Rendering Architecture*

Figure 4 illustrates the ngin3d render architecture. The stage tree contains the property chains values for each actor. These are the values you set when you assign a value to an actor property.

The presentation tree contains the values that are currently being presented to the user. For example, setting a new value for the *Position* of an actor immediately changes the value in the stage tree. However, the *Position* value in the corresponding actor in the presentation tree will be updated later when rendering the actor.

The render-tree uses the value in the presentation-tree to render objects to the user. The render-tree is responsible for performing the compositing operations independent of application activity; rendering is done in a separate thread (GL thread) so that it has minimal impact on the application's run loop. Google advices developer using *GLSurfaceView* for OpenGL rendering. Since it's easy to use, has dedicated renderer thread and it's well-tested [10]. Ngin3d adopt it.

You can query an instance of actor for its corresponding presentation while an animation transaction is in process. This is most useful if you intend to change the current animation and want to begin the new animation from the currently displayed state. [5]

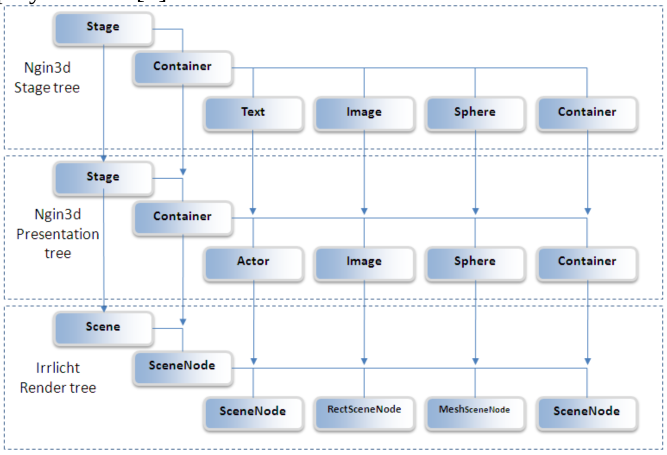

Fig. 4 ngin3d Rendering Architecture

Ngin3d has well communication with Irrlicht throughs JNI. In order to keep flexibility, we are trying to implement most features in ngin3d and leave performance-critical operations in Irrlicht.

D. *Media3D*

The Media 3D application is to demonstrate our 3D graphics capability in MT6573, and it includes 3D weather, 3D photo, and 3D video, all present in landscape mode (As shown in figure 5). This application used to show the capability of ngin3d [11].

1) *Main Page*: When launching Media3D, it will show Media 3D main screen in landscape mode. There are three panels stand for weather, photo and video respectively. At the bottom of panels, there is a water surface that combines ripple and reflection effect.

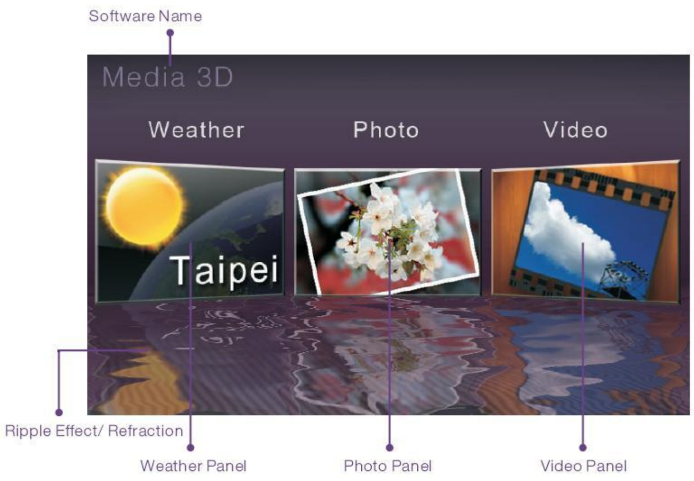

Fig. 5 Media3D main page

2) *Weather Page*: When you press the weather panel in main page, it will jump to 3D weather page. Weather page shows the weather of city in 3D stage with 3D animation. When dragging on the touch screen, as figure 6 show, old time-panel

and weather icon fly out and new ones fly in. Also the earth rotates to the right position of the city.

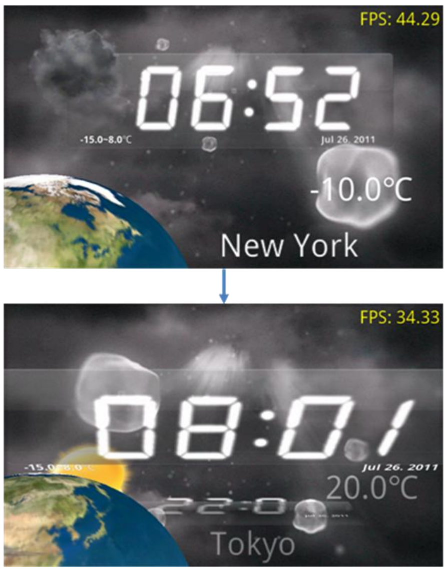

Fig. 6 Weather 3D transition

3) *Photo Page*: Photo page shows four photos with thumbnail views at one time. With dragging, photos on the screen fly out and new ones fly in with 3D transition. Please reference figure 7.

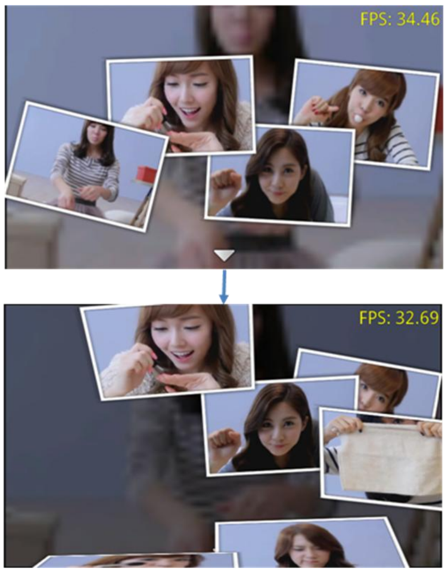

Fig. 7 Photo 3D transition

4) *Video Page*: Video page shows the first frame of videos. The 3D transition in video page is similar with photo page. (As shown in figure 8)

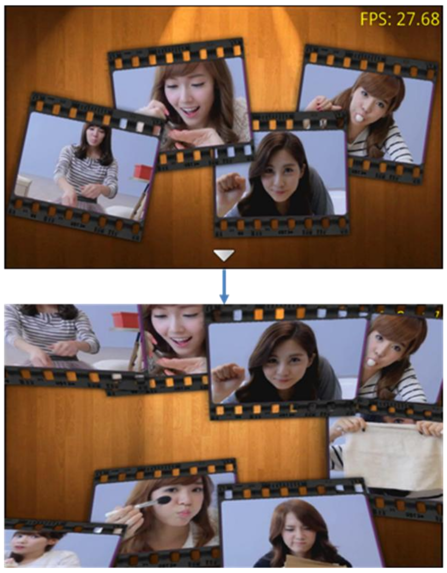

Fig. 8 Video 3D transition

III. CONCLUSIONS

Ngin3d is a collection of Java classes for 3D graphics rendering, projection, and animation. It follows Google's advice and use the best way to construct OpenGL ES 1.1/2.0 develop environment and embeds to Android System very well. It provides a flexible animations system that you can create fluid animation by using Property animation or loading Key-frame animation from the output of After Effect.

Developer can develop a 3D android application easily by using ngin3d. There are still many features can be supported by ngin3d. Include bitmap font, sprite/ sprite animation [7][9], mesh animation and stereoscopic 3D etc. In the future we will develop these features and integrate them into ngin3d.

REFERENCES

[1] Aaftab Munshi, Dan Ginsburg, and Dave Shreiner, "OpenGL ES 2.0 programming Guide".
[2] Richard S. Wright, Jr., Nicholas Haemel, Graham Sellers, and Benjamin Lipchak, "OpenGL superBible fifth Edition"
[3] http://developer.android.com/sdk/index.html
[4] http://irrlicht.sourceforge.net/
[5] http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CoreAnimation_guide/Articles/CoreAnimationArchitecture.html
[6] http://www.khronos.org/opengles/
[7] http://www.texturepacker.com/
[8] http://www.imgtec.com/
[9] http://www.cocos2d-iphone.org/
[10] http://www.google.com/events/io/2011/index-live.html
[11] Du Chun, "Media 3D 1.0 for version 0.92 Functional Specification"
[12] http://www.json.org/