Freescale Semiconductor
Application Note

Document Number: AN4045
Rev. 0, 01/2010

Building a 3D Graphic User Interface in Linux

Building Appealing, Eye-Catching, High-End 3D UIs with i.MX31

by *Multimedia Application Division*
Freescale Semiconductor, Inc.
Austin, TX

To compete in the market, apart from aesthetics, mobile devices are expected to provide simplicity, functionality, and elegance. Customers prefer attractive mobile devices and expect new models to be even more attractive. For embedded devices, a graphic user interface is essential as it enhances the ease of use. Customers expect the following qualities when they use a Graphical User Interface (GUI):

- Quick and responsive feedback for user actions that clarifies what the device is doing.
- Natural animations.
- Provide cues, whenever appropriate, instead of lengthy textual descriptions.
- Quick in resolving distractions when the system is loading or processing.
- Elegant and beautiful UI design.

This application note provides an overview and a guide for creating a complex 3D User Interface (UI) in Linux® for the embedded devices.

Contents

freescale semiconductor

1 X Window System

The X Window system (commonly X11 or X) is a computer software system and network protocol that implements X display protocol and provides windowing on bitmap displays. It also provides a standard toolkit and protocol stack to build GUIs in Linux. The X window system is a widely used graphical windowing system in Unix and Linux. It was developed in 1984. The X window system is viable and it is also the standard environment for the Unix windowing systems.

Figure 1 shows the screenshot of an X Window system.

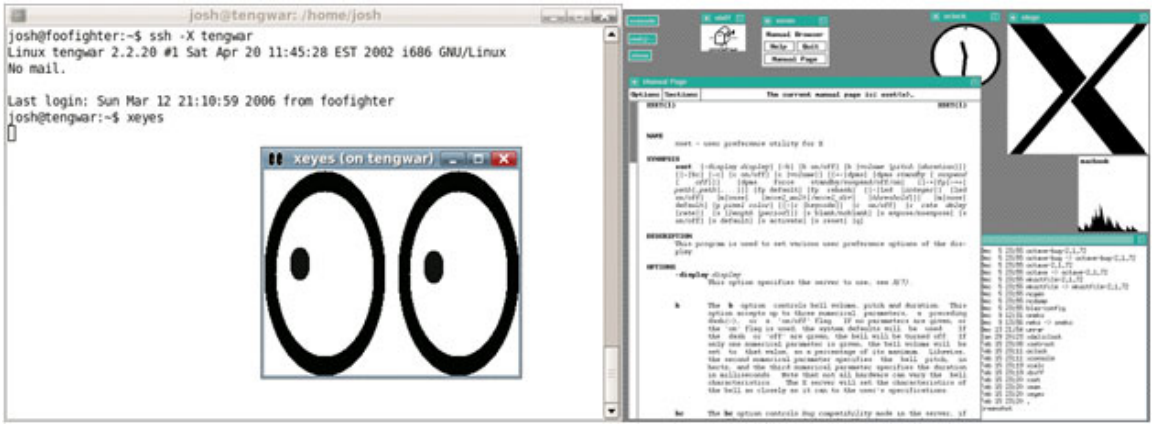

Figure 1. X Window System

The X Window system provides the basic framework or primitives for building GUI environments, such as drawing and moving windows on the screen and interacting with a mouse and/or keyboard. The X Window system does not authorize the UI, only the individual client programs handle the authorization. The visual styling of X-based environments varies and different programs provide different interfaces. The X Window system is not a part of the operating system on any of the systems it runs on. It is a user application which is built as an additional layer on top of the host operating system. So, X Window system is network transparent (ability to transmit graphical data over the network and integrate it with applications running and displaying locally) and operating system independent.

The current protocol version of the X Window system (X11) originated in September 1987 and is used in every UNIX system. The X Window system provides the following features:

- Basic framework or primitives.
- Input/Output communication with hardware.
- Network transparency specifically designed to be used over network connections.

The X Window system is often used through high-level abstraction libraries such as Graphics Toolkit (GTK), Qt, Simple Direct media Layer (SDL), Evas, Tk, Xaw, Motif, and so on.

1.1 UI Issues

X window system does not contain any specification for UI or most inter-application communications. This results in different interfaces and applications that do not work in synchronization. Generally, graphic programmers address the application’s look and feel and communication by coding a specific desktop

environment or a widget toolkit. This avoids dealing with the Inter-Client Communications Convention manual (ICCCM), which is a standard for inter operability between X Window system clients in the same X server.

2 Overview of GUI Options for Linux

This section describes the GUI options for Linux.

2.1 Graphics Toolkit

Graphics Toolkit (GTK) is a library for creating graphical UIs similar to the Motif's look and feel. It is one of the most popular toolkits for the X Window system. GTK is designed to be small and efficient, but flexible enough to allow programmers to create interfaces and uses a variety of standard UI widgets.GTK provides several container widgets which can be used to control the layout of the UI elements.

GTK is summarized as follows:

- Created to help development of the GNU Image Manipulation Program (GIMP) with X windows system.
- Focuses on visual widgets (buttons, text box, and so on).
- Allows non-X11 platforms usage.
- Basic free-drawing operations.
- Adopted by GNOME project due to its Lesser General Public License (LGPL).

Figure 2 shows an application developed using GTK, which has better capabilities and more flexibility than X Window system.

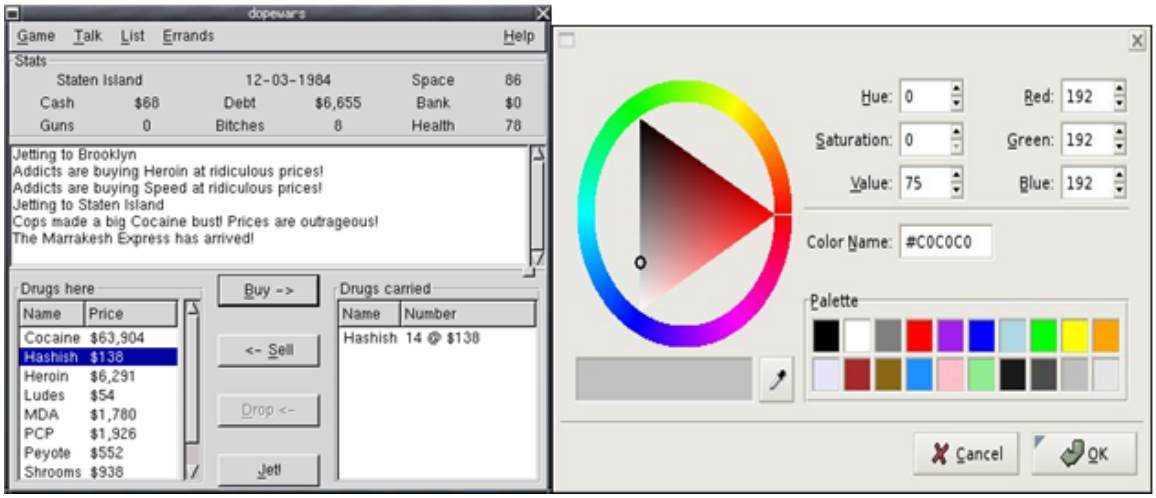

Figure 2. Application Developed Using GTK

GTK presents this flexibility in a uniform framework. Specifically, it supports object oriented programming that is well adapted to the purposes of a UI toolkit. It also provides a reasonable solution and

disciplined programming interface. These features make it easy and reliable to access GTK from languages other than C.

However, GTK is limited in many ways, such as nature of being multiplatform (and multi-flavor), and lead to a non native look and feel. The framework is pixel-level based so building a semi-complex application need a very significative effort from the developer, because animation and transitions have to be implemented from scratch. Finally, GTK does not support 3D in any way, so anything 3D related is not possible using GTK.

2.2 Open Graphics Library®

The Open Graphics Library (OpenGL) is the most widely adopted 2D and 3D graphics application programing interface (API) in the industry. It is used in thousands of applications to wide variety of computer platforms.

The Open GL is an industry-adopted standard for hardware and software, which is present in every device that has 3D acceleration. It is very powerful and flexible and is adopted by GNOME project due to its LGPL license. It can perform basic free-drawing operations.

The main features of Open GL are as follows:

- Focuses on 3D (2D is handled as consequence).
- Fast triangle/rectangle/polygon drawing.
- Fast matrix transformations (scale, rotate, shear, perspective).
- Fast alpha blend—a process of convex combination of two colors allowing for transparency effects in computer graphics.

Figure 3 shows the Open GL graphic images.

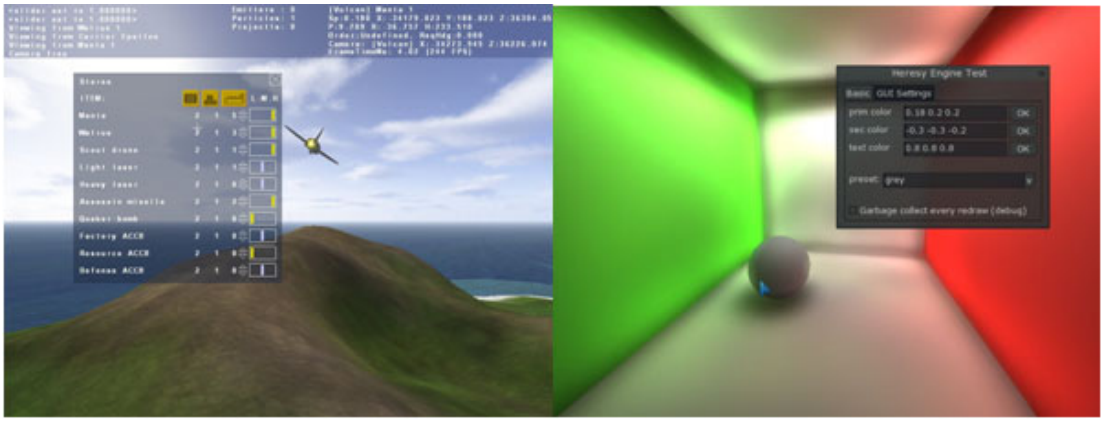

Figure 3. Open GL Graphic Images

Although OpenGL is an widely adopted standard for graphics, it is not very friendly for developing GUIs. Every frame or button should be a set of triangles projected in 2D on a plane (screen) and input has to be given by picking rays or by buffer-analysis. Currently, most Linux graphic card drivers (X servers) do not support hardware-accelerated OpenGL for remote applications. They support hardware acceleration for local applications only.

3 Clutter Toolkit - Solution for GUIs

This section describes the Clutter toolkit.

3.1 Features

Clutter is an open source software library for creating fast, visually rich and animated GUIs. It uses Open GL for Embedded Systems (OpenGL ES) for rendering with an API which hides the underlying GL complexity from the developers. So, the Clutter API is easy to use, efficient and flexible.

Clutter has the following features:

- Scene-graph of layered 2D interface elements manipulated in 3D space by positioning, grouping, transparency, scaling, clipping, and rotation.
- Frame based animation engine providing path interpolation, transitions, and other custom effects by behaviors and timelines, which are introduced to simplify 3D and 2D effects creation.
- Advanced input event handling.
- Custom Pango renderer providing efficient internationalized 8-bit unicode transformation format (UTF8) text rendering.
- Support for high end Open GL features such as shaders and fixed base operators (FBOs).
- Support for media playback with GStreamer, Cairo graphics rendering, GTK+ embedding, Bo.
- Object oriented design by GObject with a familiar GTK+ like API.
- Runs on Linux, with back end window system support for GLX (OpenGL Extension to the X Window System), EGL, WGL, simple direct media layer (SDL) and Cocoa.
- Support for mobile devices with fixed point internals and portability across Open GL, OpenGL ES 1.1 and OpenGL ES 2.0.

Clutter aims to be nonspecific. It implements no particular style, rather provides a rich generic foundation that facilitates rapid and easy creation of higher level tool kits tailored to specific needs.

Figure 4 and Figure 5 show Clutter images.

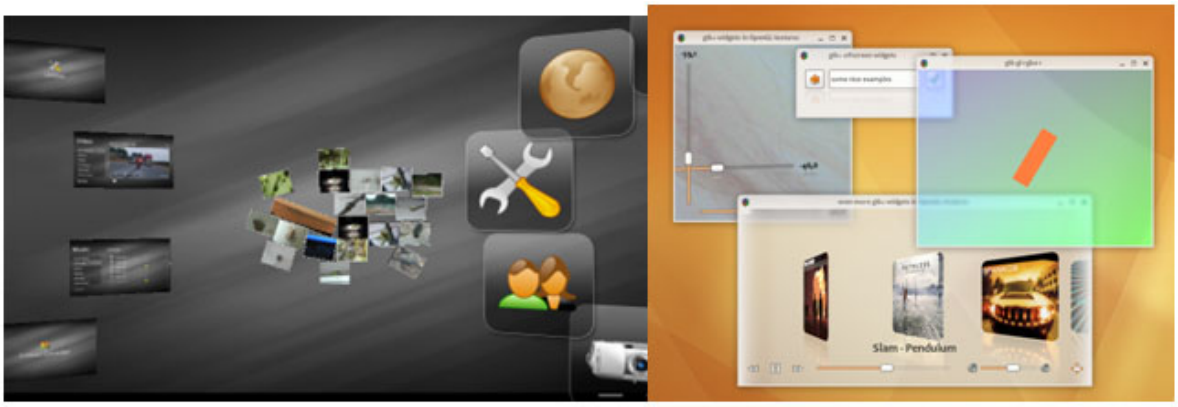

Figure 4. Clutter Images

Figure 5. Clutter Images

3.2 Clutter Overview

Clutter semantics work by having a stage (a window) and then adding actors (widgets) to the stage and manipulating through the actor API. Actors can contain child actors (for example, ClutterGroup) and they can be manipulated as a whole.

Animations and visual effects can be created using the timelines and behaviors. Timelines provide accurate frame based animations. Behaviors further extend this by taking a timeline, a control function (ClutterAlpha) and then applying to actors as to modify a property as a function of time.

The following steps are used to show the basic structure (base actors and containers) of clutter workflow:

1. The following code declares the objects:

```
ClutterActor    *stage;
ClutterTimeline *main_timeline;
```

2. The following code explains initialization of objects (pseudo objects), after declaration:

```
clutter_init(&argc, &argv);
stage = clutter_stage_get_default();
ClutterColor     stage_color = { 0x00, 0x00, 0xff, 0xff };
clutter_stage_set_color(CLUTTER_STAGE (stage), &stage_color);
clutter_actor_set_name(stage,"stage");
clutter_actor_set_size (stage, SCREEN_WIDTH, SCREEN_HEIGHT);
```

`ClutterActor`—Base abstract class for all visual stage actors.

Example 1 shows a simple layout of the clutter base objects, which are the Actor, Box, Layout and container.

Example 1. Layout of the Clutter Base Objects

```
ClutterActor *desktops_main_root;
```

`ClutterContainer`—Interface for implementing container actors.

```
desktop_container = clutter_group_new();
desktop_root = clutter_group_new();
```

`ClutterLayout`—Interface for implementing layouts

`ClutterBox`—Base class for layout containers

3. The images are loaded to use them as textures in the program:

```
GdkPixbuf *temp_img;
temp_img = gdk_pixbuf_new_from_file("images/background.jpg", NULL);
desktop_center = clutter_texture_new_from_pixbuf(temp_img);
clutter_actor_set_name(desktop_center,"Center desktop");
```

3.3 Creating the Scenegraph

After base actors and containers are created, they must be ordered in a scenegraph (tree).

The following steps are used to order them in a scenegraph:

1. The code for declaring the main nodes is a follows:

```
clutter_container_add(CLUTTER_CONTAINER(stage),main_root);
clutter_container_add(CLUTTER_CONTAINER(desktops_main_root),desktop_container);
clutter_container_add(CLUTTER_CONTAINER(desktop_container),desktop_center);
```

   In this code, `main_root`, `desktops_main_root`, `desktop_container`, and `desktop_centers` are the main nodes of our data structure. Desktop backgrounds (images) and icons are added as children of these main nodes.

2. The code for adding desktop background and icons as children of the main node is as follows.

```
clutter_container_add(CLUTTER_CONTAINER(icons_root),back_icon);
clutter_container_add(CLUTTER_CONTAINER(icons_root),(main_icons[0]));
clutter_container_add(CLUTTER_CONTAINER(icons_root),main_icons[1]);
```

   After they are created and added to the scenegraph, the code is operated to perform its basic operation of actor to modifying its position using the code shown below:

```
clutter_actor_set_position(back_icon,10,640);
clutter_actor_set_position(main_icons[0],70,15);
clutter_actor_set_position(desktop_container,0,0);
```

Position is set in a 2D plane, however, it is also possible to change its depth with the following function:

```
clutter_actor_set_depth(desktop_container,-480);
```

Here the position of the `desktop_container` is changed, so that all its children are also affected. This desktop is 480 units far from the camera.

Figure 6 shows the clutter images having all the above works.

Figure 6. Image Representing the Clutter

3.4 Behaviors

A behavior is a controller object for `ClutterActor`, which is used to control one or more properties of an actor (such as its opacity, or its position). A Clutter behavior is controlled by an alpha function stored inside a ClutterAlpha object. An alpha function depends solely on time. The alpha function computes a value which is then applied to the properties of the actors controlled by a behavior.

The following are some of the predefined behaviors provided by Clutter:

- ClutterBehaviorPath, which controls the position of a set of actors making them walk along a set of nodes.
- ClutterBehaviorOpacity, which controls the opacity of a set of actors.
- ClutterBehaviorScale, which controls the width and height of a set of actors.

The following program helps to start the behavior:

- The following code creates timeline and specifies number of frames per second and number of frames the animation lasts

```
ClutterTimeline *tl_selection;
tl_selection = clutter_timeline_new(15,30);
```

- The following code creates the alpha function:

```
ClutterAlpha *alpha = clutter_alpha_new_full (tl_selection,
CLUTTER_ALPHA_SMOOTHSTEP_DEC, NULL, NULL);
```

- The following code creates the behavior:

```
ClutterBehaviour *r_behave;
r_behave=   clutter_behaviour_rotate_new(alpha,CLUTTER_Y_AXIS,CLUTTER_ROTATE_CW,45,5);
```

- The following code executes the behavior:

```
clutter_behaviour_rotate_set_center(r_behave,SCREEN_WIDTH/2,0,
offset_z-SCREEN_HEIGHT/2);
clutter_behaviour_apply(r_behave,desktop_container);
```

- This last function is used to start the behavior that is previously created, see Section 3.5, “Animation by Frames,” for more information.

  The following function starts the behavior that was created previously:

```
clutter_timeline_start(tl_selection);
```

3.5 Animation by Frames

Clutter has a powerful and flexible framework for animating actors. The basis of it is the `ClutterTimeline` class which represents time period in frames. A `ClutterTimeline` takes two parameters—total number of frames and frame rate (in frames per second).

Once a signal (new-frame) is created, it can be attached and then on starting it (`clutter_timeline_start()`), the signal callback is called every time a new frame is reached. Using this callback and the current frame number, the information can be used to modify actor properties and produce an animation.

The following code shows how `g_signal_connect()` calls a callback function `on_new_frame()` in every frame, so the user can perform animations:

```
void desktop_transtion(ClutterContainer *container)
{
        ClutterTimeline *transition_timeline = clutter_timeline_new(30,30);
        g_signal_connect(transition_timeline,"newframe",G_CALLBACK(on_new_frame),container);
        clutter_timeline_start(transition_timeline);
}
```

The following code is called for every frame receiving a timeline, the actual frame number, and the data to be modified (Actor, Containers, so on.):

```
void on_new_frame(ClutterTimeline *timeline,gint frame_num, gpointer data
{
ClutterActor *group = CLUTTER_ACTOR(data);
clutter_actor_set_position(group,0,-transition_direction*frame_num*(32));
}
```

The actor's position is changed in every frame, achieving a smooth animation.

3.6 Event Handling

Clutter handles events in an efficient way. The events are as follows:

- Key event
- Mouse motion
- Mouse Scroll
- Touchscreen
- Buttons
- Additional devices

First the events are bind to the stage using `g_signal_connect()`. The following code binds the events to the stage:

```
g_signal_connect(stage,"event",G_CALLBACK(input_cb),NULL)
```

This means that when an input is registered, the function `input_cb()` is called. Example 2 is a sample code that explains the `input_cb()` function call.

Example 2. Sample Code to Call the input_cb()

```
void input_cb (ClutterStage*stage, ClutterEvent*event, gpointer data)
{
ClutterActor *current_actor;
current_actor = clutter_stage_get_actor_at_pos(stage,x,y);
switch (event->type)
{
         case CLUTTER_KEY_PRESS:
         len = g_unichar_to_utf8 (clutter_keysym_to_unicode (event->key.keyval),keybuf);
         keybuf[len] = '\0';
         printf ("- KEY PRESS '%s'\n", keybuf);
         break;
         case CLUTTER_KEY_RELEASE:
         len = g_unichar_to_utf8 (clutter_keysym_to_unicode (event->key.keyval),keybuf);
         keybuf[len] = '\0';
         printf ("- KEY RELEASE '%s'\n", keybuf);
         break;
         case CLUTTER_MOTION:
         // printf("- MOTION\n");
         break;
         case CLUTTER_BUTTON_PRESS:
         }
}
```

4 Conclusion

Clutter is a good solution for building GUIs and is easy to setup on th i.MX31. It reduces the development time and gives a fast and stylish implementation of a GUI build on OpenGL. Clutter is strongly recommend to build GUIs in Linux for the i.MX31 since, it is already used in the industry and does not use an API, which takes a longer development time.

5 Revision History

Table 1 provides a revision history for this application note.

Table 1. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	01/2010	Initial release.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN4045
Rev. 0
01/2010