AVTK - the UI Toolkit behind OpenAV Software

Harry VAN HAAREN
OpenAV Productions,
Co. Clare,
Ireland.
harryhaaren@gmail.com

Abstract

AVTK[1] is a small lightweight user interface toolkit designed for building custom graphical user interfaces interfaces. It was conceived particularly to build LV2 plugin GUIs, however it can be equally useful in standalone programs.

Focusing on user experience, AVTK promotes ease of use for novices yet affords power-users the most efficient interactivity as possible.

The author feels this is particularly important in live-performance software, where user-experience and creativity are in close proximity.

Keywords

User Experience, User Interface, LV2 Plugins.

1 Introduction

AVTK is a C++ user interface library. It is designed specifically for creating LV2 plugins[2], but can be used to build normal user interfaces too. OpenAV created AVTK out of the need for a flexible and powerful way of creating custom user interfaces for audio plugins.

In the background section, common issues when building user interfaces for plugins are presented. The implementation of the AVTK user interface library is then shown, and code samples are provided, providing an example of how to build a minimal interface. Finally we discuss how the common problems presented in the background are solved in this implementation.

2 Background

When creating an embeddable user interface for a plugin, issues arise that do not apply to writing normal user interfaces. These issues arise from the interaction between the user interface of the host program and plugin.

2.1 Embedding

Embedding is the process of showing a user interface created in one toolkit inside the window of another toolkit. This is a complex problem and many different solutions exist. Many solutions are platform dependant, ie: they only work on Linux, Mac OsX or Windows.

PUGL[3] is a cross platform library which "supports embedding and is suitable for use in plugins". AVTK uses PUGL to embed into host program's windows, and it can create a standalone window too if embedding isn't desired.

2.2 Modal Popup Windows

Many general purpose toolkits have the functionality to create modal popup windows. This type of window can cause problems when embedding a plugin UI in a host window.

The first problem is that the popup is set to be "above" its parent window, however the parent window is generally already set to be "above" the host program. This can cause the popup window to be below the parent window, however modal popup windows don't allow the user to interact with the parent. The user is force to manually bring the popup window to the front, and interact with it. This has a negative impact on the user experience (UX).

The second problem is one of stalling the UI thread of the host program. When a modal popup dialog is created, many user interface toolkits wait for the user to interact with it, before continuing the execution. This stalls the host programs UI thread, as that thread created the dialog. The end result is that as the popup window is shown, the host programs UI is frozen. Again, a negative impact on UX.

2.3 User Experience

When using a plugin user interface, user experience is of very high importance. In the context of AVTK, the user is most likely creating music, or involved in a creative process of some description. To ensure the best UX for both novice and power users, the normal user interaction concepts are augmented in AVTK. Care is taken to ensure that these augmented interaction possi-

bilities do not cause confusion to novice users.

Transparently augmenting the interaction with a widget provides power users with better UX, and ultimately in the user achieving their goal more efficiently. The scroll wheel on the mouse, and modifiers keys are used in order to augment the interface for efficency.

3 The Implementation

The implementation of AVTK draws from experience gained while developing custom interfaces using other toolkits.

The main window is treated as one large canvas, and widgets are drawn into this canvas. As widgets can be transparent, creating layered interfaces becomes easy.

3.1 Dependencies

During the design stage cross-platform libraries were chosen, in order to ensure portability. Cairo[4] is used for all drawing routines, while PUGL provides access the window and input events. PicoJSON[5] is used for parsing JSON, while tinydir[6] provides access to the filesystem.

3.2 Widget Class

The Widget class is the core of AVTK. The Widget class has virtual functions that a developer can override to customize behaviour. The following is an excerpt of the `Widget` class:

Listing 1: Widget Class

```
class Widget
{
public:
  Widget( Avtk::UI* ui,
          int x,
          int y,
          int w,
          int h,
          string label );

  // draw and handle events
  virtual void draw(cairo_t* cr);
  virtual int handle(PuglEvent* e);

  // set value on widget
  float value();
  void  value( float v );

  // change notification callback
  void (*callback)(Widget*, void*);
  void* callbackUD;
};
```

The most important function is draw(), which paints the widget to screen. The handle() function deals with user input using the cross-platform `PuglEvent` abstraction.

The `value()` functions set and get the value of the widget. A callback function can be provided to be notified of activity on a particular widget instance.

3.3 Minimal UI

The code in listing 2 shows a minimal AVTK user interface with a single button. Notice that we override the `widgetValueCB()`function, and that the callback function of any child widgets is automatically routed to the `UI` instance.

Listing 2: Minimal Demo UI

```
class DemoUI : public Avtk::UI
{
  public:
  DemoUI(PuglNativeWindow parent = 0) :
    Avtk::UI( 400, 240, parent )
  {
    new Avtk::Button( this,
      50, 20, 300, 200, "Button" );
  }

  void widgetValueCB(Avtk::Widget* wid)
  {
    printf("Widget %s with value %f\n",
      wid->label(), wid->value() );
  }
};

int main()
{
  return DemoUI().run();
}
```

3.4 Custom Widget Creation

In order to customize a widget one simply derives from the `Widget` base class, and overrides the `draw()` method. The Cairo library is used to draw widgets, and a `cairo_t*` context is passed into the draw() function. Calling the desired Cairo functions will draw the widget.

The `value()` function of the `Widget` base class can be called to get the current value of the widget - allowing easy drawing of the widget in its current state.

The `handle()` function can be overriden in case the custom widget requires non-default user input handling.

3.5 Modal Widgets

Modal popups are implemented as a popup *widget*, instead of its own window in AVTK. This is to keep UX consistent, and avoid the pitfalls of modal windows as described in the background section.

As the calling thread should not be stalled while the popup is shown, AVTK treats a popup widget like a normal widget. The exception is

Figure 1: The Fabla2.0 interface, powered by AVTK. Edited for print.

event handling, which allows the modal widget to disable interaction with the other widgets while it is shown.

Figure 2 shows two user interfaces one of which has a modal dialog shown. Note the dialog is positioned close to the cursor for ease of interacting with it.

3.6 Theming

The theme engine in AVTK is geared towards providing a variety of themes to a widget, and being able to change them on the fly. Themes are loaded from JSON files at runtime, allowing easy modification.

Theme files provide colours for "use-cases". Some examples are background, foreground, background-dark etc. This approach allows using the same theme file on any widget, and the widget adapts the colours in the theme to what is being drawn. This lightweight approach to themeing allows fast prototyping, and scales to having many different themes available for each widget.

Fig. 1 shows an interface, and various widgets themed to show the user the effect of the widget in question. In this particular screenshot changing the "Bank" changes the primary colour of the interface, indicating the change to the user.

4 Special Features

This section describes special features of AVTK. The interaction between the power user and the interface is where the design of AVTK flourishes. Many widgets afford operation with hotkeys, right-clicks and drag-n-drop areas. The following sections discuss where the interaction between user and interface has been augmented.

4.1 Right Click and Default Values

User input from the mouse buttons is leveraged in almost every widget in order to allow efficient resetting of controls. Any `Widget` that uses its `value()` is enabled with a right-click feature to reset to the default value. Calling `defaultValue()` sets a new default value for the widget.

Right-click to reset to default value can be disabled, using the `Widget::rClickMode()` function.

4.2 Groups and Scroll events

`Groups` can be used to provide radio-button style selection of a particular widget in the group (see Fig 2. for widget examples).

This is particularly useful in scrollable areas, which are commonly used in applications to select a particular option from a range of predetermined options.

Using input from the scroll wheel is an intuitive mapping - but there is a conflict when a `Group` is in a `Scroll` widget as the Scroll widget uses the scroll event. In order to achieve an optimal workflow, a `Ctrl+Scroll` action is added to a `Group` allowing the user to navigate group.

The efficiency of navigating a widget list is improved for power users familiar with the `Ctrl+Scroll` hotkey, while consistency is maintained in how widgets behave regardless of if they are in a scrollable area or not.

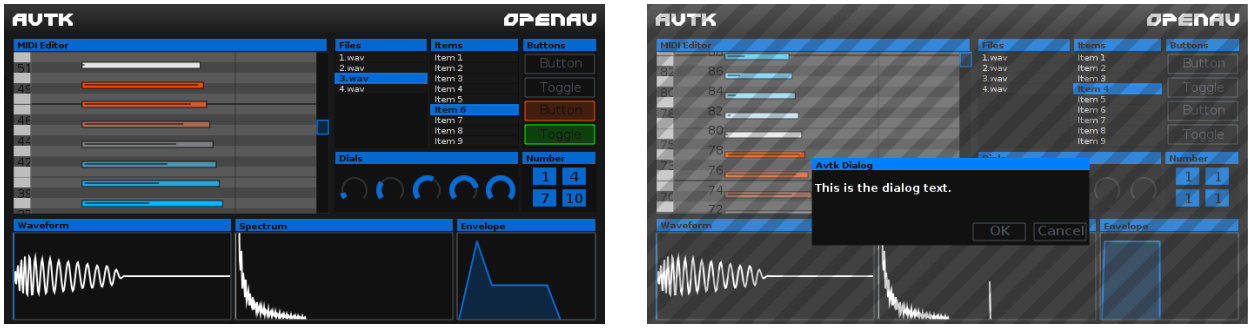

Figure 2: Showcase of AVTK widgets, a popup Dialog "stripes out" the background.

4.3 Filenames

When the user is searching for a particular audio file in a list of similar files, the filenames often have an identical start causing long filenames. This causes a workflow where the user must scroll horizontally and vertically in order to select the desired file.

In an attempt to solve the workflow issue as above, the AVTK file-browser includes an option to hide the common start of a filename. And example to illustrate this is as follows:

```
// Actual Filenames
samplepack_kick_heavy.wav
samplepack_kick_click.wav
samplepack_snare_snappy.wav

// AVTK Filenames
kick_heavy
kick_click
snare_snappy
```

The example above shows three filenames with the common prefix `samplepack_` followed by the information user requires, followed by a filetype extension.

In this example, the common prefix `samplepack_` has been removed, and the extension is removed for readability (It is noted that many general purpose toolkits already hide the file extension).

4.4 Testing of User Interfaces

When an issue is found in a program by a user, often the first step a developer takes is to attempt to reproduce the issue. The user is asked to describe what steps will reproduce the issue, and the developer mimics them in order to find the cause of the issue.

To improve the workflow in finding UI issues, user input is recorded, and then replayed on the developer's computer.

AVTK serializes the input events from the user to a JSON file, which is uploaded to the developer. They then replay the events, automatically mimicing the users input.

It should be noted that the developer must have the same version of the software as the user as pixel co-ordinates are stored in the event stream. It follows that if the user interface is re-arranged, the users actions may no longer achieve the same result.

5 Conclusion

AVTK is a small lightweight user interface toolkit, targeting developers who wish to build custom user interfaces. It solves common issues other toolkits have when embedding as a plugin UI by utilizing a more appropriate design for the use-case.

It has some special features geared towards power-users, and has a theme engine that allows developers create prototypes quickly. An event recording and playback mechanism is included to aid finding issues users are having with the software in question.

6 Future Work

OpenAV intends to continue using AVTK to build interfaces for plugins and standalone software. Currently the Fabla 2.0 sampler is the only complex project using AVTK.

Future work includes expanding the available widgets as necessary for multi-media centric software, and testing on all major platforms.

7 References

[1] http://openavproductions.com/avtk
[2] http://lv2plug.in
[3] http://drobilla.net/software/pugl
[4] http://cairographics.org
[5] http://github.com/kazuho/picojson
[6] http://github.com/cxong/tinydir