Kenny Chao, Fotolia

Multimedia applications with OpenML

DOWN THE PATH

Programming multimedia applications in Linux used to be a tedious process, demanding specialized libraries and even custom code for addressing hardware directly. The OpenML library offers a simpler approach.

BY TIM SCHÜRMANN

Shortly after the millennium, a number of major league multimedia corporations joined forces to found the Khronos Group. 3Dlabs, ATI, Discreet, Intel, Nvidia, SGI, and Sun Microsystems all wanted to create a cross-platform multimedia library using the lean and successful OpenGL as a role model. Thanks to this standardized interface, programmers would be able to concentrate on their work rather than wasting time thinking about hardware and how to control it.

Just a year later, specification 1.0 of the Open Media Library [1] (or OpenML – not related or even good friends with OpenMP, a library for shared memory parallelism, or OpenAL, a library for sound output) was released. The whole enchilada was done – the interfaces, the internal functions – and the Khronos Group (see the "Khronos Group" box) stood back, celebrated, promised support at all kinds of conferences, and waited for something to happen. Developers all over the world now knew what the interface looked like, but there were no signs of a tangible implementation of the library, not to mention matching drivers or hardware.

Knight in Shining Armor

Luckily, major components of OpenML are based on work done by graphics specialists SGI. SGI already had the digital media-handling SDK (dmSDK) in its portfolio before dmSDK was integrated with the new multimedia library. This probably also explains the fairly rapid agreement on the new standard. SGI grabbed this opportunity by the scruff of its neck and developed the OpenML SDK from dmSDK. At this point, the Khronos Group took over the helm and nominated the results as its reference implementation. To promote more widespread use, SGI Free Software License C, or FreeC for short, was added to additionally support free commercial use [2]. This was in 2004. Since then, the reference implementation has led the life of a wallflower on SourceForge. The only program to fly the OpenML flag is Jahshaka (Figure 1), the semi-professional video editing and compositing program, although it is not generally considered the best of role models because of its quirky controls and tendency to crash [3].

Benefits

Today, the OpenML idea is still firmly in the doldrums, although it doesn't deserve it; in fact, the

library is definitely worth looking at. It not only standardizes and simplifies access to a computer's set of multimedia devices, it also helps to synchronize media and simplifies buffer handling. All of this is independent of the operating system and free of charge: Anyone can drop the specifications into a library and use the existing reference implementation in their products, whether commercial or open source.

In a similar style to OpenGL, OpenML just lays a thin abstraction layer over the existing multimedia hardware. Programmers can use a lean C API to talk to the hardware; the API is not only easier to handle but much easier on systems in which resources are scarce – embedded systems is just one example. Just as 3D engines rely on OpenGL, a special high-level library could rely on OpenML, and this, in turn, guarantees versatility and flexibility.

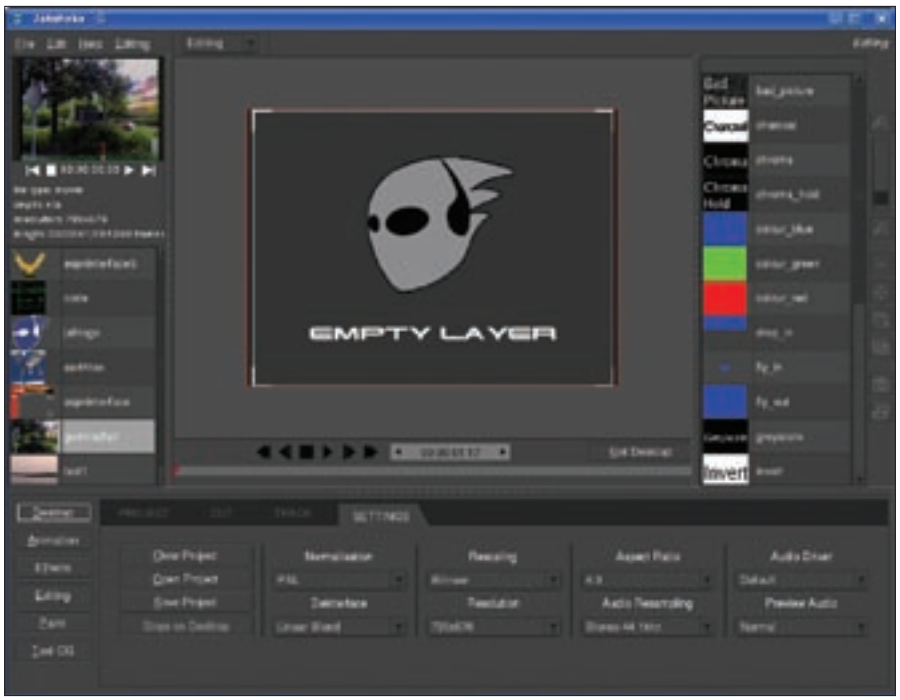

Figure 1: The Jahshaka video effects tool relies on OpenML.

Core Issues

As I mentioned earlier, the founding fathers of OpenML have simply combined standards to create a new one (see Figure 2). It only follows that OpenML comprises several components or parts, each of which handles a specific task (see Figure 3).

The core of OpenML is the Media Library (ML). It is the same as SGI's dmSDK for the main part. Its functions retrieve video and audio material from attached media devices and process the multimedia data, synchronizing the data where needed and displaying it on an output device.

The core library is supported by the OpenML Display Control Library (MLdc). It controls one or more output devices, such as the monitor, a TV, or even a wall full of displays. In addition to this, the MLdc handles fine tuning by, say, modifying the video format or implementing gamma correction. This allows an application to determine what to display on which monitor. MLdc also originated with SGI, or with the XSGIvc extension to X11 to be more precise.

The third known component that the Khronos Group assimilated was the popular OpenGL, to which a couple of extensions were added. This component handles 3D graphics and video and must be implemented by any OpenML-compliant graphics card. This means you can use graphics or videos as textures, perform fast color correction, enable image enhancement, or convert color spaces. In addition, users gain the ability to use interlaced material (such as video material that uses the European TV standard, PAL [4]).

The last OpenML component helps to synchronize data streams. To do so, it runs a number of timers in the background. The time source is always unadjusted system time (UST), or in plain English, The system time. OpenML additionally assigns a sequence number to each sample in each data stream. This would mean enumerating each frame in a video or each sample in audio material. OpenML dubs these labels "media stream counters" (MSCs). The relationship between the MSC and UST is the refresh rate. This would be 25 frames per second for video material. This provides a simple approach to synchronization or correction.

Sound Shifting

After all those acronyms, an example that illustrates how to use OpenML is probably a good idea. The approach is similar to that in the OpenML Media Library Software Development Kit Beginner's Guide [5] on how to output an

Khronos Group

The Khronos Group was founded in January 2000. Ever since, it has developed various multimedia interfaces in the style of OpenGL. The best known of these is probably OpenGL ES, a customized OpenGL for embedded systems. At the end of last year, the consortium finally took over maintainership and ongoing development of the OpenGL standard.

The members of the Khronos Group split up into three groups, which have more or less to say in the standardization process, depending on the amount of money they have invested.

Application developers can use the OpenML specifications and the reference implementation (SDK) free of charge. They are also allowed to implement the specification free of charge and to distribute it in combination with other products. If you like, you can advertise with the OpenML logo on your homepage for free.

For an annual fee, you can become an open member and contributor. This rank allows you to participate actively in various working groups.

Promoters (the steering group) mainly comprise the founders of the Khronos Group, the guys with the big money. They have a final veto right and define the organization's aims and policies.

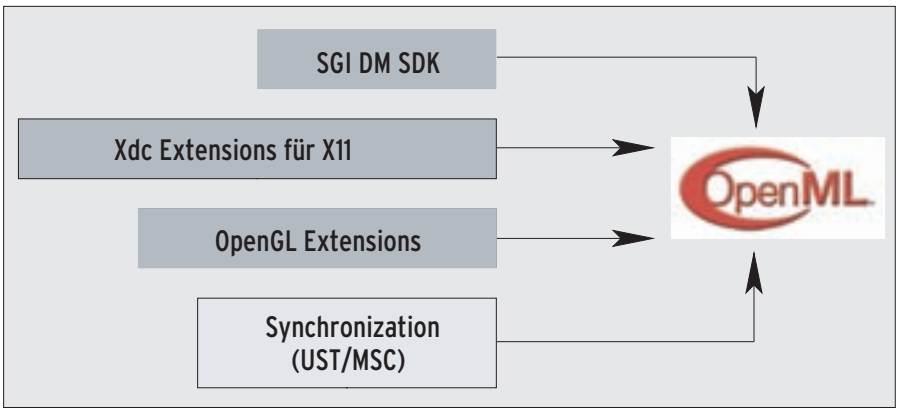

Figure 2: OpenML is based on three existing libraries, all of which originated with SGI. The synchronization components are new.

audio signal via the sound card. The audio data have already been read from disk and stored in main memory – OpenML does not help with this. Following standard programming practice, I will use a reserved memory block as a buffer.

Headers

As with any C program, this program must have the right headers to start. For an OpenML program, these headers include *ml.h*, for access to the core library, and *mlu.h*, which provides functions that rely on the core library.

Climbing the Tree

OpenML neatly organizes any multimedia devices that an application can use into what is known as a capability tree. The tree not only includes the devices themselves, but also some additional information on their capabilities. The *mlquery* tool included with the OpenML SDK [6] lists the usable devices on the current machine. Figure 4 shows an example.

To locate a device in the capability tree to match the planned operation, the application climbs around the tree until it finds a suitable candidate. The entry point is always the top layer (i.e., the root of the tree), which represents the whole computer system. My sample code looks for an audio playback device:

```
MLint64 devId=0;
mluFindDeviceByName➥
(ML_SYSTEM_LOCALHOST, ➥
"audio device", &devId);
```

After this call, the matching physical device is returned in the variable *devId*; *ML_SYSTEM_LOCALHOST* is the label for the whole system and, thus, the capability tree entry point.

Jacks

Unfortunately, developers can't just pass audio to the sound card without worrying about the consequences. What happens if the sound card has multiple outputs, for example? To resolve possible conflict, each physical device has one or more logical devices, known as jacks. They represent the places you could send video or audio data, or where they could enter the system. An example of a jack for a sound card could be a microphone or audio output. Thus, the jack represents the interface the application will use for system I/O.

Incidentally, a jack need not be mapped one-to-one with a physical interface: Because it is a logical device, a jack can represent multiple physical devices (as in multi-channel sound); conversely, multiple jacks can be mapped to a single physical interface. In my example, I need a jack to output the audio material. The following code simply finds the first available output that fulfills this condition.

```
MLint64 jackId=0;
mluFindFirstOutputJack➥
(devId, &jackId);
```

The next thing I need is a way of feeding the audio (or video) material to the jack and, from there, out of the system. In OpenML-speak this is known as a path, and it connects the buffer with the audio data in main memory to the jack (Figure 5). OpenML distinguishes between

- input paths, which supply a data stream and are stored in a buffer, and

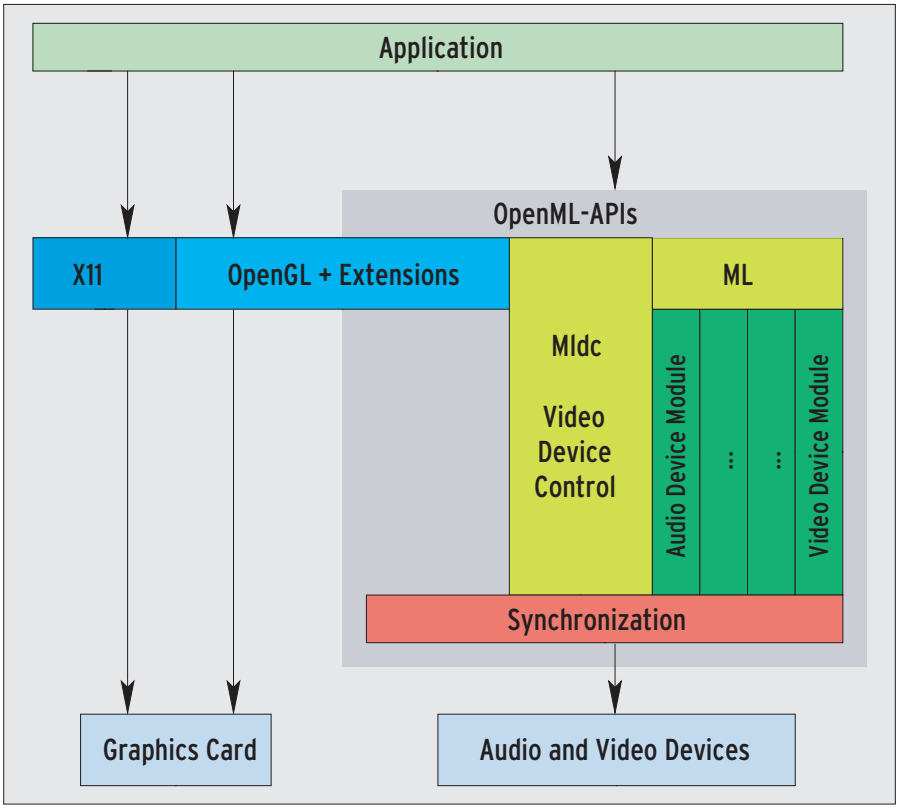

Figure 3: OpenML comprises four components, including the extended OpenGL, the MLdc device control, and a synchronization layer. The core component, ML, uses device modules to access the physical hardware.

- output paths, which accept a memory buffer from the application and pass the content on to the jack.

In my example, the following code uses a path to connect to the output jack:

```
MLint64 pathId=0;
mluFindPathToJack➥
(jackId, &pathId);
```

After this, the next two lines open the output path:

```
MLopenid openPath;
mlOpen➥
(pathId, NULL, &openPath);
```

This gives the application direct access to the hardware. At the same time, the function allocates the system resources required for subsequent operations. The second parameter in *mlOpen()* is used to pass in configuration parameters; in this case, I will use the defaults.

Mail Service

After establishing a connection to the output device, I have to take care of a couple of settings. The audio data I want to output were recorded in mono at a frequency of 44,100Hz. The developer needs to pass these settings on to the output device.

In OpenML, applications use messages to communicate with devices. Messages comprise a list of parameters and pairs of values stored in a type *MLpv* array. The header files define the parameters and a couple of values in the form of macros. The end of the list is designated by the *ML_END* tag.

For my example, I first need a new message with the settings shown in Listing 1 (16 bit, 44,100Hz, mono, with a gain of -12dB).

Now I'll send the message down the path to the jack:

```
mlSetControls➥
(openPath,➥
controls);
```

which is a blocking call. The function will not return until the settings have been delivered or an error occurs.

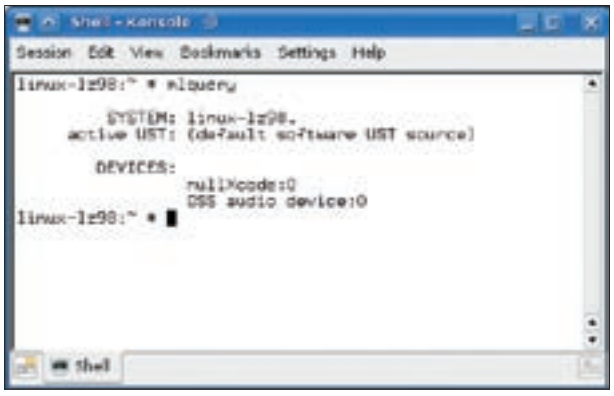

Figure 4: The mlquery tool outputs a list of multimedia devices on the system. It has found an OSS-compatible sound card in this example.

Mass Migration

Now, I'll send the audio data to the device. To do this, again, I need to bundle the data into a message and send the message down the path to the device. In my example, the *ourAudioBuffer* pointer marks the start of the buffer with the audio data. The SDK reference [6] tells how the data, which I will reference as *ourAudioBuffer*, has to be organized and formatted in memory. For stereo, the sample for the left channel is always followed by the sample for the right channel (interleaved).

As the programmer, I am responsible for creating and filling the buffer and, thus, equally responsible for honoring the conventions. If I get everything right, I can bundle the audio data into a message behind the *ourAudioBuffer* pointer (Listing 2).

Finally, I just send the message down the path:

```
mlSendBuffers(openPath, msg);
```

This sends the message to the path's mailbox; another step is required to empty the box:

```
mlBeginTransfer➥
(openPath);
```

This line finally tells OpenML to open the box and send the messages in order of arrival to the device. The use of a queue might seem redundant at first, but the presence of a queue does give me a handle on compensating for latency or drop-outs, such as the ones the operating system regularly causes.

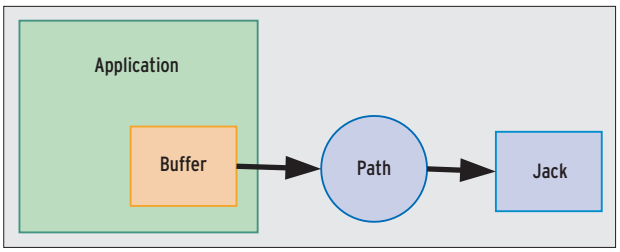

Figure 5: The multimedia application stores the media data in a buffer, which is passed to the jack along the path. The jack is the interface into or out of the computer system.

Restful Slumber

Because *mlBeginTransfer()* returns without waiting for output to complete, the application can do a couple of other useful things in the meantime. But beware: OpenML does not use locking, so you will not want to access any buffers you have sent. To make sure this doesn't happen, my sample application just goes to *sleep(5)*.

After processing the data, OpenML sends a success message to the application's mailbox, and the application opens the message by calling *mlReceiveMessage()*:

Listing 1: Audio Output

```
01 MLpv controls[5];    /*
   Message containing 5 values */
02 MLreal64 gain = -12; /* Gain
   */
03
04 controls[0].param = ML_AUDIO_
   FORMAT_INT32;
05 controls[0].value.int32 = ML_
   FORMAT_S16;
06 controls[1].param = ML_AUDIO_
   CHANNELS_INT32;
07 controls[1].value.int32 = 1;
08 controls[2].param = ML_AUDIO_
   GAINS_REAL64_ARRAY;
09 controls[2].value.pReal64 =
   &gain;
10 controls[2].length = 1;
11 controls[3].param = ML_AUDIO_
   SAMPLE_RATE_REAL64;
12 controls[3].value.real64 =
   44100.0;
13 controls[4].param = ML_END;
```

```
mlReceiveMessage(openPath, ➥
&messageType, replyMessage);
if(messageType==➥
ML_BUFFERS_COMPLETE)
  printf("Data transmitted!\n");
```

This mechanism also gives programmers access to any error messages. Additionally, every function returns an *MLstatus* type value, although this might not work for asynchronous transmissions, as in my example.

Freedom

To clean up, I need to close the path and release the jack:

```
mlClose(openPath);
```

The box titled "Stuffed Sausage" gives tips on neat handling for larger volumes of data.

Pipes and Transcoders

Besides jacks, OpenML also uses transcoders. A transcoder is a logical device that accepts data from a buffer, then performs an operation with the data and pushes the results out into another buffer (Figure 6).

Transcoders are mainly used to push media data back and forth between jacks. For example, a jack can supply data from a video camera; then feed the data through a matching transcoder, where it is unpacked; and finally feed the data to another jack to be displayed on a preview monitor.

Transcoders can be implemented as hardware (e.g., to calculate real-time video effects) or as software (e.g., MPEG2 decoders).

Whereas a jack is fed with data and messages by a path, a pipe handles the task of feeding data to a transcoder. A transcoder will always need at least one input pipe to feed it with input data, as well as one output pipe through which it delivers the results.

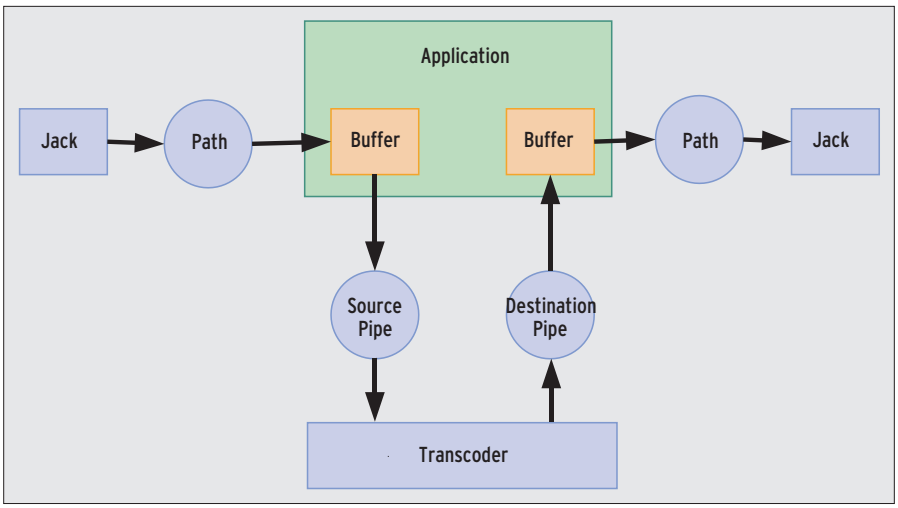

Figure 6: A path through OpenML: The data stream flows into a buffer. A pipe picks up the data and passes it through a transcoder, which uses a pipe to pump the results into a buffer in main memory.

Conclusions

In a style worthy of OpenGL, OpenML has what it takes to become a cross-platform standard for multimedia applications. The OpenML library has a reputation of being a competitor to Microsoft's DirectX, and this reputation is justified. What OpenML lacks is more support from hardware and software vendors.

Despite its promising start, OpenML can't be regarded as a panacea. The OpenML library lacks functions for important tasks such as streaming over networks, and OpenML still doesn't come with support for special input devices. The list of unsupported devices includes more than just the gamepads and wheels that gamers rely on. Also unsupported are the kinds of controls that multimedia terminals typically use. As OpenML gains attention in the development community, we'll hope that hardware vendors will step up to provide better support. ■

Listing 2: Bundling the Audio Data

```
01 MLpv msg[2];
02 msg[0].param = ML_AUDIO_
   BUFFER_POINTER;
03 msg[0].value.pByte =
   ourAudioBuffer;
04 msg[0].length =
   sizeof(ourAudioBuffer);
05 msg[1].param = ML_AUDIO_UST_
   INT64;   /* timestamp */
06 msg[2].param = ML_END;
```

Stuffed Sausage

If you need to send a larger volume of data or a large number of messages to a jack, your best approach is to first create a small buffer, then play back the buffer and wait for the wait handle to be returned:

```
MLwaitable pathWaitHandle;
mlGetReceiveWaitHandle§§
(openPath, &pathWaitHandle);
```

Then you just wait for your number to be called:

```
fd_set fdset;
FD_ZERO(&fdset);
FD_SET(pathWaitHandle, &fdset);
select(pathWaitHandle+1, §§
&fdset, NULL, NULL, NULL);
```

After this, you wait again for the corresponding acknowledge message to arrive. Then, you fill up the buffer with the next blob of data – images or the next audio track, for example – and re-use the old envelope to send it back:

```
mlSendBuffers§§
(openPath, replyMessage);
```

INFO

[1] OpenML: *http://www.khronos.org/openml*

[2] SGI Free Software License C: *http://oss.sgi.com/projects/FreeC*

[3] Jahshaka, an OpenML-based video effects program: *http://www.jahshaka.org/*

[4] Information on PAL: *http://en.wikipedia.org/wiki/PAL*

[5] OpenML Media Library Software Development Kit Beginner's Guide: *http://techpubs.sgi.com/library/tpl/cgi-bin/download.cgi?coll=0650&db=bks&docnumber=007-4376-001*

[6] OpenML SDK: *http://sourceforge.net/projects/oml-ri*