OpenGL-Plugins für das XMMS

Felix J. Ogris
felix_juergen.ogris@fh-bielefeld.de

11. Oktober 2006

X MultiMedia System

- **der** Audioplayer unter Linux & Unix
- 5 Plugintypen
  - Inputplugins
  - Outputplugins
  - Effektplugins
  - generische Plugins
  - Visualisierungsplugins

Visualisierungsplugins

- liegen in den Verzeichnissen
  - `/usr/lib/xmms/Visualization`
  - `$(HOME)/.xmms/Plugins/Visualization`
- sind *shared libraries*
- müssen eine Funktion `get_vplugin_info()` exportieren

get_vplugin_info

- Einsprungpunkt in das Plugin
- liefert einen Zeiger auf eine *statische* Struktur vom Typ `VisPlugin`

Also:

```
static VisPlugin mein_plugin = {
  /* ... kommt gleich ... */
};

VisPlugin* get_vplugin_info () {
  return &mein_plugin;
}
```

VisPlugin

```
typedef struct {
  void (*init)(void);
  void (*cleanup)(void);
  void (*playback_start)(void);
  void (*playback_stop)(void);
  void (*render_pcm)(gint16 pcm_data[2][512]);
  void (*render_freq)(gint16 freq_data[2][256]);
  char *description;
  int num_pcm_chs_wanted;
  int num_freq_chs_wanted;
  ...
} VisPlugin;
```

Arbeitstiere
render_pcm & render_freq

render_pcm & render_freq

- render_pcm bekommt decodierte Samples als Parameter übergeben
- render_freq bekommt per FFT das Frequenzspektrum übergeben
- dürfen keine aufwendigen Berechnungen vornehmen, sonst stockt der Abspielprozess

Deshalb:

- OpenGL-Darstellung in einem eigenen *Thread*
  - render_pcm & render_freq kopieren Daten nur in einen globalen Puffer
- Verzicht auf GLUT
  - Aufruf von `glutSwapBuffers()` aus einem anderen Thread bewirkt kein Neuzeichnen der OpenGL-Szenerie
  - Polling über GLUT-Timerfunktion „unsauber“
- direkte Programmierung des *X Window (X11)* Grafiksystems

Pluginablaufplan

1. `playback_start` startet die OpenGL-Zeichenroutine `display_func` in einem eigenen Thread

Pluginablaufplan

1. `playback_start` startet die OpenGL-Zeichenroutine `display_func` in einem eigenen Thread
2. `display_func`
   1. öffnet ein X11-Fenster
   2. wartet auf Sampledaten

Pluginablaufplan

1. `playback_start` startet die OpenGL-Zeichenroutine `display_func` in einem eigenen Thread
2. `display_func`
   1. öffnet ein X11-Fenster
   2. wartet auf Sampledaten
3. `render_pcm` bzw. `render_freq`
   1. kopieren Sample- bzw. Frequenzdaten in eine globale Variable
   2. signalisieren `display_func`, dass neue Daten bereit stehen

Pluginablaufplan

1. `playback_start` startet die OpenGL-Zeichenroutine `display_func` in einem eigenen Thread
2. `display_func`
   1. öffnet ein X11-Fenster
   2. wartet auf Sampledaten
3. `render_pcm` bzw. `render_freq`
   1. kopieren Sample- bzw. Frequenzdaten in eine globale Variable
   2. signalisieren `display_func`, dass neue Daten bereit stehen
4. `display_func` bereitet die Daten auf und stellt sie dar

Pluginablaufplan

1. `playback_start` startet die OpenGL-Zeichenroutine `display_func` in einem eigenen Thread
2. `display_func`
   1. öffnet ein X11-Fenster
   2. wartet auf Sampledaten
3. `render_pcm` bzw. `render_freq`
   1. kopieren Sample- bzw. Frequenzdaten in eine globale Variable
   2. signalisieren `display_func`, dass neue Daten bereit stehen
4. `display_func` bereitet die Daten auf und stellt sie dar
5. Rücksprung zu 2.2, bis `playback_stop` aufgerufen wird

Pluginablaufplan

1. `playback_start` startet die OpenGL-Zeichenroutine `display_func` in einem eigenen Thread
2. `display_func`
   1. öffnet ein X11-Fenster
   2. wartet auf Sampledaten
3. `render_pcm` bzw. `render_freq`
   1. kopieren Sample- bzw. Frequenzdaten in eine globale Variable
   2. signalisieren `display_func`, dass neue Daten bereit stehen
4. `display_func` bereitet die Daten auf und stellt sie dar
5. Rücksprung zu 2.2, bis `playback_stop` aufgerufen wird
6. `playback_stop` signalisiert `display_func` sich zu beenden und wartet auf das Ende des Threads

Pluginablaufplan

1. `playback_start` startet die OpenGL-Zeichenroutine `display_func` in einem eigenen Thread
2. `display_func`
   1. öffnet ein X11-Fenster
   2. wartet auf Sampledaten
3. `render_pcm` bzw. `render_freq`
   1. kopieren Sample- bzw. Frequenzdaten in eine globale Variable
   2. signalisieren `display_func`, dass neue Daten bereit stehen
4. `display_func` bereitet die Daten auf und stellt sie dar
5. Rücksprung zu 2.2, bis `playback_stop` aufgerufen wird
6. `playback_stop` signalisiert `display_func` sich zu beenden und wartet auf das Ende des Threads
7. `display_func` schliesst das X11-Fenster und beendet den Thread

Erstellte Plugins
Endlich etwas Animiertes

visual_pcm

- kopiert Stereo-Audiosamples an das Ende eines FIFO-Buffers
- stellt die einzelnen Werte dieses Buffers mit dünnen Linien dar
- erweckt so den Eindruck als „fließe“ das Musikstück durch das Fenster

Erstellte Plugins
Endlich etwas Animiertes

visual_pcm

- kopiert Stereo-Audiosamples an das Ende eines FIFO-Buffers
- stellt die einzelnen Werte dieses Buffers mit dünnen Linien dar
- erweckt so den Eindruck als „fließe“ das Musikstück durch das Fenster

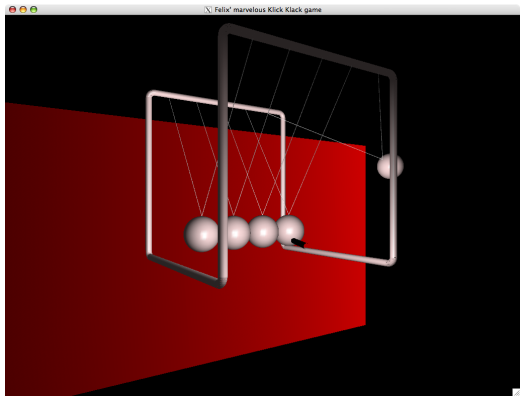
visual_freq

- stellt das aktuelle Stereo-Frequenzspektrum dar
- jede Linie ist dabei $\frac{Samplingfrequenz}{2*256} Hz$ „breit“
  (2 * 256 = Nyquist * Arraygrösse)

Erstellte Plugins
Endlich etwas Animiertes

klick_klack

- Fortsetzung des im Praktikum erstellten Modells eines Kügelchen- oder Klick-Klack-Spiels
- Entfernung aller GLUT-Funktionen (bis auf `glutSolidSphere`)
- Auslenkung der beiden äusseren Kugeln abhängig von der Energie der aktuellen Samples
- Ambientlight ebenfalls abhänging von der Energie
- $\mathit{Energie} := \mathit{Intensität} * (22050 - \mathit{Frequenz})$



Ende
Die letzte Folie

Danke für Ihre Aufmerksamkeit.
Fragen?