net: Application Layer

Bausteine für einen Browser

Bernd Paysan

Forth–Tagung 2013, Garmisch–Partenkirchen

Übersicht

Motivation

Anforderungen

Ein paar kleine Demos
- Slideshow
- Videos
- Text

Ausblick

Wofür?

Die Leute möchten Informationen teilen (*teilen* bedeutet *kopieren)*

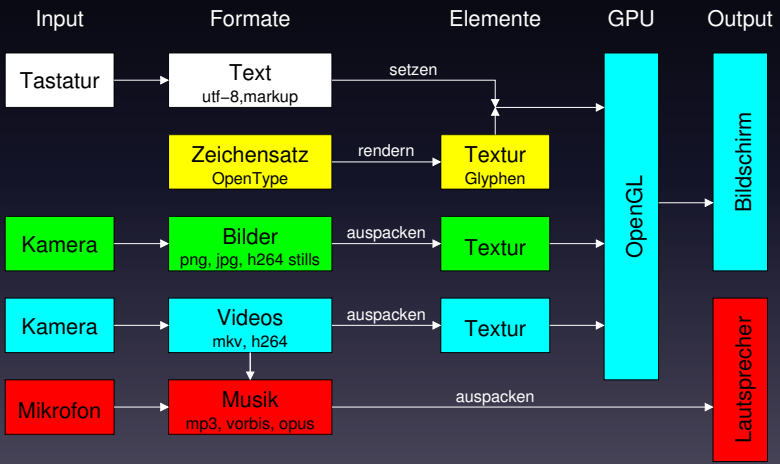
- Texte, Fotos, Videos, Musik
- Längere, strukturierte Dokumente
- Echtzeit–Medien (Chat, Telefonie, Video–Konferez)
- Zusammen spielen

Was möchte ich zeigen?

- Einen Vortrag des Titels habe ich schon 2011 in Wien gehalten — da war das noch komplett Vaporware: Das war der Plan.
- Inzwischen gibt's was, nämlich funktionierende Teile, die zusammengebaut werden müssen.
- Und ein Konzept, wie das Zusammenbauen aussieht.
- Muss neben PCs auch auf mobilen Plattformen wie Android laufen, deren Umgebung „etwas komisch“ ist.

Formate und Ausgabe

Wie stellt man etwas dar?



Warum OpenGL?

OpenGL kann alles

OpenGL rendert:

1. Dreiecke, Linien und Punkte — simple Grundformen
2. Texturen und Farbverläufe
3. und nutzt dazu Shader–Programme — und das ist das eigentlich Mächtige an OpenGL ab 2.0

Die eigentliche Anforderung ist: Visualisierung von *irgendwelchen* Daten. OpenGL kann das.

Wie verbindet man die Medien?

Lemma: Jede Glue–Logik wird Turing–complete

- Bisher benutzer Glue: HTML+CSS+JavaScript
- Und als Container für Flash, Java, ActiveX, PDF, Google's NaCl...
- Schussfolgerung: Von vornherein auf ein mächtiges Werkzeug setzen
- Der Browser ist eine Laufzeit- und Entwicklungsumgebung für Anwendungen

Sicherheit

Lemma: Jedes hinreichend komplizierte Format lässt sich für einen Exploit ausnutzen

Javas Ansatz, eine Sprache „von innen“ heraus zu sichern, kann als gescheitert betrachtet werden. Java ist heute Einfallstor Nummer 1 für Schädlinge.

Sandbox

- Den Prozess, der Netzwerkdaten interpretiert, in eine Sandbox sperren
- Netzwerkverbindungen über einen Proxy leiten — da fehlt noch ein shared–memory–Modul für net2o
- Der Zugriff auf Schlüssel muss außerhalb der Sandbox stattfinden
- „Same–Origin“–Policies funktionieren in der Cloud nicht.

Slideshow

Die Slide–Show verwende ich für diese Präsentation

Fader

```
: fade { n1 n2 f: delta-time -- } n1 n2 = ?EXIT
    ftime { f: startt }
    BEGIN  ftime startt f- delta-time f/ fdup 1e f<  WHILE
            <draw-slide
            1e blend n1 draw-slide
            ( time ) blend n2 draw-slide
            draw-slide>  REPEAT
    <draw-slide 1e blend n2 draw-slide draw-slide>
    fdrop ;
```

Slideshow 2

Noch mehr Effekte

Hslide

```
: hslide { n1 n2 f: delta-time -- } n1 n2 = ?EXIT
    ftime { f: startt }
    BEGIN  ftime startt f- delta-time f/ fdup 1e f<  WHILE
            <draw-slide
            pi f* fcos 1e f-
            [ pi f2/ fnegate ] FLiteral f* fcos 1e f-
            fdup n1 n2 > IF fnegate  THEN xshift n1 draw-slide
            2e f+ n1 n2 > IF fnegate  THEN xshift n2 draw-slide
            draw-slide>  REPEAT
    <draw-slide n2 draw-slide draw-slide>
    fdrop ;
```

Vorgehen und Probleme

libSOIL: Einfache API zum Bilder–Laden

Die APIs von libjpeg und libpng sind ausgesprochen kompliziert.
Alternative: libSOIL:

libSOIL Textur laden

```
: load-texture ( addr u -- )
    open-fpath-file throw rot close-file throw tilde_cstr
    SOIL_LOAD_AUTO current-tex SOIL_FLAG_TEXTURE_REPEATS
    SOIL_load_OGL_texture drop ;
```

Zwiebel–Programmierung

Von außen ziemlich groß

Zwiebel–Programmierung

Martialisches Werkzeug wird empfohlen

Zwiebel–Programmierung

Zwiebel „all the way down“

Video

OpenMAX AL

- Auf Android ist OpenMAX AL das Video–Framework — ähnlich wie gstreamer, aber doch anders.
- Kann Videos in eine Textur rendern, aber auch aufzeichnen (von der Kamera)
- Input: MPEG Transport Stream
- C–API ist C++–artig (Struktur von Funktionspointern, also eine vtable)
- Die Native–API ist nur halb implementiert, zum Starten muss man Java via JNI bemühen
- Videoplayer verwendet dann vier Sprachen: Forth, C, Java, OpenGL Shader Language

JNI–Deklarationen

MediaPlayer

```
jni-class: android/media/MediaPlayer

jni-new: new-MediaPlayer ()V
jni-method: prepare prepare ()V
jni-method: start start ()V
jni-method: setSurface setSurface (Landroid/view/Surface;)V
jni-method: setVolume setVolume (FF)V
```

JNI–Deklarationen II

SurfaceTexture

```
jni-class: android/graphics/SurfaceTexture

jni-new: new-SurfaceTexture (I)V
jni-method: updateTexImage updateTexImage ()V
jni-method: getTimestamp getTimestamp ()J
jni-method: setDefaultBufferSize setDefaultBufferSize (II)V
jni-method: getTransformMatrix getTransformMatrix ([F)V
```

JNI–Aufruf

Timestamp abholen

```
: get-deltat ( -- f )
    media-sft >o getTimestamp o> d>f 1e-9 f*
    first-timestamp f@ f- ;
```

Java–Calls fügen sich nahtlos in Mini–OOF2 ein (Mini–OOF mit aktuellem Objekt)

MTS? Alle Videos sind heute MKV!

„Matroska“ klingt schon auch nach Zwiebel–Programmierung...

Wozu überhaupt einen Container?

- Übliche Begründung: Mehrere Dateien zu „unhandlich.“ Ich finde Verzeichnisse mit mehreren Dateien eigentlich handlicher als noch eine Möglichkeit, Container zu bilden.
- Container bricht Video und Audio in einzelne Bilder und kurze Pakete auf
- Zeitstempel für synchronisierte Wiedergabe
- Index für das schnelle „Spulen“

Matroska–Interpreter

Binäres XML–Format

Lösung: MKV einlesen, MTS ausspucken.

- Matroska–Parser hat eine Hash–Table für die binären Tags
- Jedem Tag ist noch eine Methode (mit Klarnamen) eines Mini–OOF2–Objekts zugeordnet
- Konkrete Parser können unterschiedliche Methoden haben, Beispiel: Dump zur Inspektion, und MTS–Konverter–Klasse.

Fonts rendern

Freetype–GL rendert OpenType–Fonts in OpenGL–Texturen

- Stand der Technik für Vektorfonts ist das OpenType–Format
- Gerendert werden Texturen, also muss der Vektorfont in die Textur
- FreeType–GL packt Zeichen in eine Textur (Cache), und macht aus jedem Zeichen einen Glyph
- Diese Glyphen verwandelt man in zwei Dreiecke und rendert sie

Text rendern Demo–Code

Fonts und Texte

```
48e FConstant fontsize#
atlas "/system/fonts/DroidSans.ttf\0" drop
fontsize# texture_font_new Value font1
atlas "/system/fonts/DroidSansFallback.ttf\0" drop
fontsize# texture_font_new Value font2
Variable text1$ "Dös isch a Tägscht." text1$ $!
Variable text2$ "这是一个文本：我爱彭秀清。" text2$ $!
```

Text rendern Demo–Code II

Fonts und Texte

```
: glyph-demo ( -- )  hidekb
    1 level# +!  BEGIN
        <render
        0. penxy 2!
        font1 to font  text1$ $@ render-string
        -100e penxy sf! -60e penxy sfloat+ sf!
        font2 to font  text2$ $@ render-string
        render>
        sync >looper
    level# @ 0= UNTIL ;
```

Ausblick

Diese Präsentation ist mit LaTeX Beamer gerendert...

- Die nächste Präsentation muss komplett in MINOΣ2 gerendert sein
- Texte und Videos müssen über net2o geholt werden, und nicht schon vorbereitet auf dem Gerät sein
- Dazu fehlt noch eine Typesetting–Engine mit Boxes und Glues, Absatzumbruch und Silbentrennung
- Viel weniger Klassen als in MINOΣ — dafür dann mehr Objekte
- Neben der hbox und vbox noch eine zbox für übereinander gestapeltes
- Animationen integriert
- Die einzelnen GLSL–Programme müssen in einem Programm sein, mit Switch–Statement

Literatur&Links

Bernd Paysan
net2o fossil repository
http://fossil.net2o.de/net2o/

Bernd Paysan
minos2 fossil repository
http://fossil.net2o.de/minos2/