MPEG-4 Audio

- Motivation
- "objekt-basierte" Kodierung
- Übersicht über MPEG-4 Audio

- Structured Audio
- SAOL - orchestra language
- SASL - score language
- SASBF - sample bank format
- Tools

- Einordnung SA vs. MIDI vs. VST
- Demo

Literatur

MPEG Homepage, www.cselt.it/mpeg/
MPEG-4 Audio Homepage: sound.media.mit.edu/mpeg4

J. Lazarro, Structured Audio Book, www.cs.berkeley.edu/~lazarro/sa/book/

diverse Konferenz/Workshop-Beiträge und Tutorials auf dem MPEG-Server, u.a.:
Workshop-Vorträge auf dem 106th / 108th AES convention workshop

B. Grill, MPEG-4 scalable audio coding
J. Herre, MPEG-4 general audio coding
E. Schreier, MPEG-4 structured audio

Hendrich, Digitale Audioverarbeitung, tech-www.informatik.uni-hamburg.de/lehre/

MPEG-4 Audio:

ursprüngliches Ziel von MPEG-4:
- Audio-/Video-Kodierung bei sehr geringen Bitraten
- für mobile Systeme

Paradigmenwechsel während der Entwicklung:
- eigenständige Medien-"Objekte"
- mit entsprechenden Attributen und Methoden
- Framework mit einer Vielfalt von Codecs / Verfahren
- BIFS-Szenengraph zur Integration der Objekte (Raum, Zeit)

für Audio:
- unterschiedliche Codecs für Sprache / Musik / Synthese
- zusätzliche Effekte (Hall, Chorus, ...) und 3D-Audio
- Dekodierung und Mix erst während der Wiedergabe

MPEG-4 Audio: Übersicht

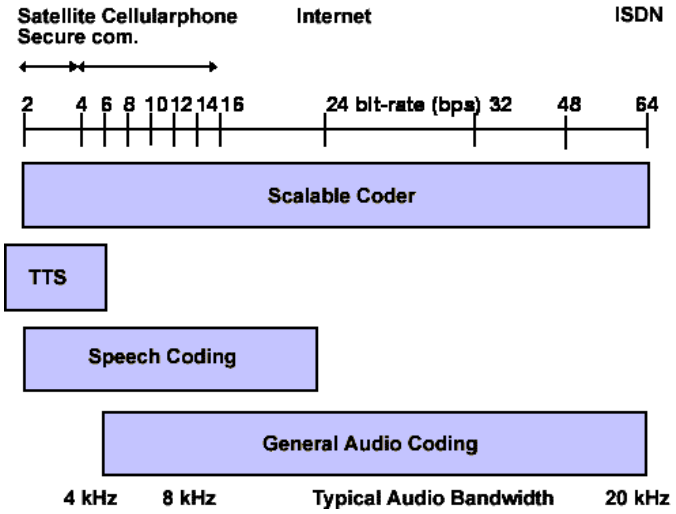
MPEG-4 definiert fünf Audio-Codecs:

- GA: general audio coder, verbesserte Version von MP3/AAC
- Speech: speech coder, sehr geringe Bitraten
- SC: scalable audio
- TTS: text-to-speech
- SA: structured audio, algorithmische Audiobeschreibung
- AudioBIFS: BInary Format for Scenes: Integration der Objekte, 3D-Positionierung, Mixer, Effekte

- Anwendungen können jeweils geeignetsten Codec auswählen
- GA-Codec basiert auf den MPEG-1/2 Codecs
- aber bessere Audioqualität oder geringere Bitraten

(siehe AES 106 und AES 108 workshops, www.cselt.it/mpeg/)

MPEG-4 Audio: Codec vs. Bitrate



MPEG-4 Audio: Perceptual Noise Substitution

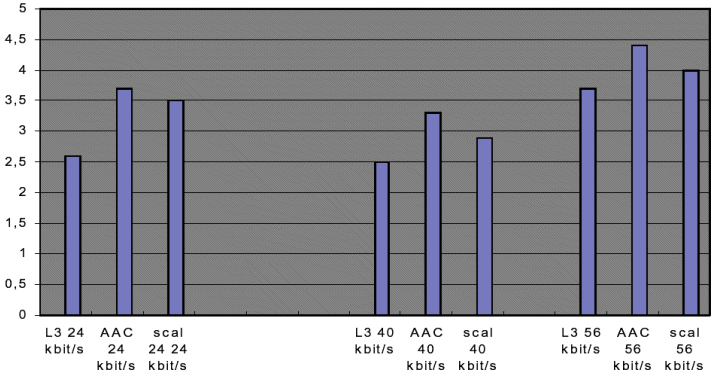
"perceptual coding" Prinzip in MPEG-1/2:

- Ausnutzen der Maskierungseffekte und der Hörschwelle
- Bitzuteilung für Frequenzbänder abhängig von MNR/SNR-Werten
- Unterscheidung tonale / atonale Schallanteile

PNS in MPEG-4 general audio:

- Encoder erkennt Rauschanteile im Signal
- aber keine direkte Kodierung dieses Rauschens
- sondern Parameterextraktion (Amplitude, Spektrum)
- "noise substitution flag" im Bitstream
- Decoder erzeugt passendes pseudozufälliges Rauschen
- extrem kompakte Repräsentation von Geräuschen

MPEG-4 Audio: Bitraten vs. Qualität



- Hörtests mit trainierten Testpersonen
- subjektive Bewertung, Skala 5.0 .. 1.0
- CD-nahe Qualität bei 64 kb/s (für stereo)
- im Notfall "akzeptabel" bei 16 Kb/s

Audio-Synthese:

- Wiedergabe kodierter / komprimierter Daten
- Reproduktion von Musikinstrumenten
  - leichter spielbar
  - billiger, portabel, flexibler, leiser/lauter, ...
  - Rekonstruktion defekter, verlorener Instrumente
- neuartige Klänge
  - Vielfalt der möglichen Algorithmen
  - Erinnerung: Wahrnehmung: Grundton plus Spektrum
- Sprachsynthese

Musiksynthese: Klassifikation

musikalische Gesichtspunkte:

- Eingabegeräte
- Spieltechniken und -hilfen
- Klangmöglichkeiten
- Echtzeitfähigkeit, Stimmenzahl

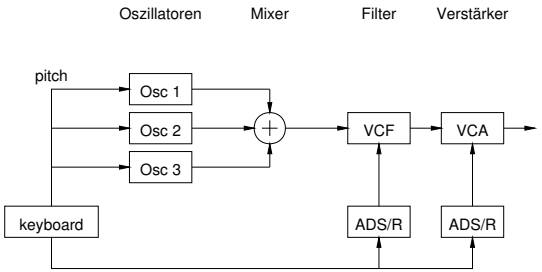
technische Gesichtspunkte:

- Syntheseverfahren
- Implementierung
- uva.

verschiedene etablierte Verfahren:

subtraktiv	("analoge" Synthese)
additiv	("Orgeln")
sampling	(meistens kombiniert mit Filtern)
Frequenzmodulation	
physical modelling	
...	

Minimoog: Blockschaltbild

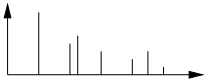


einfacher, aber ausreichend flexibler Signalfluß:

- Vorbild für praktisch alle späteren Synthesizer
- Erweiterungen über flexiblere Oszillatoren
- und weitere Modulationsmöglichkeiten

Subtraktive Synthese: Konzept

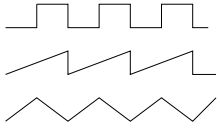
- additive Synthese wegen Kosten unmöglich



subtraktive Synthese:

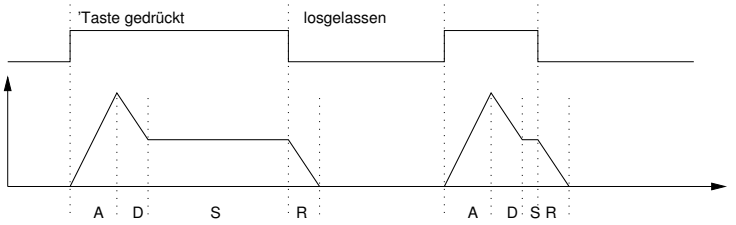
- obertonreiches Ausgangssignal
- gewünschten Klang durch Filterung erzeugen
- zusätzliche Modulation für "lebendigen" Klang

- Oszillatoren mit obertonreichen Klängen:
  - Rechteck, PWM Rechteck, Impulse
  - Sägezahn, Dreieck
  - Rauschen
  - oder unreiner Sinus



- musikalisch sinnvolle Filter, vor allem Tiefpaß
- Anschlagdynamik, Hüllkurven, Unisono, ...

Hüllkurven: ADSR



○ attack	Anschlagphase	Zeit
○ decay	Abfallzeit nach dem Anschlag	Zeit
○ sustain	eigentlicher Klang	Level
○ release	Ausklingen	Zeit

- bei Bedarf mehrstufige / komplexere Hüllkurven
- attack/decay-Phase besonders wichtig für den Klang

Sampling

- Aufzeichnen von natürlichen Klängen / Geräuschen
- getriggerte Wiedergabe dieser "samples"

in Verbindung mit einem subtraktiven Synthesizer:

- Verwendung der Samples als Ausgangsmaterial
- statt der "einfachen" Oszillatoren
- erfordert ein/mehrere Samples pro Ton
- oder Frequenzanpassung (pitch-shifting) der Samples

- Klangformung via Filter und Verstärker
- Hüllkurven, Anschlagdynamik, usw.
- fast immer digital realisiert (Ausnahme: Mellotron)
- zusätzlich Digitaleffekte

Additive Synthese

- Erzeugen eines Klangs durch Mischen von Sinussignalen
- zeitliche Modulation durch Amplitudenhüllkurve der Einzelsignale

- direkte Konstruktion des Obertonspektrums
- im Prinzip jeder Klang erzeugbar
- aber sehr hoher Aufwand (je ein Osz./ADSR/Amp pro Oberton)
- entsprechend hoher Kontrollaufwand

(Pfeifen-) Orgel:

- Grundklänge ("Register") nicht nur Sinussignale
- keine Parameter (Lautstärke / Hüllkurven) einstellbar

Frequenzmodulation

- Ausgang eines Oszillators moduliert die Frequenz eines anderen
- "modulator" and "carrier" signals
- Ausgangssignal enthält Summen- und Differenzfrequenzen:

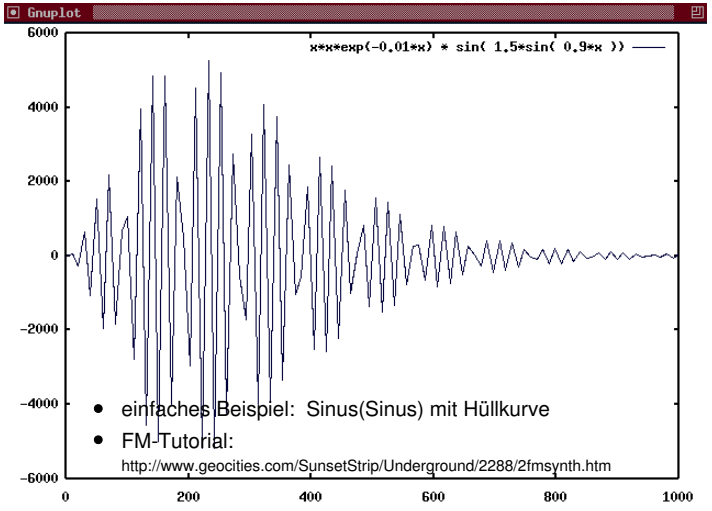
. . . C-3M C-2M C-M C C+M C+2M C+3M . . .

=> eigenständige Syntheseform: FM-Synthese

- Hüllkurven zur Regelung der Oszillatorintensität
- völlig neuartige, z.B. "glockige" Klänge
- populär durch Yamaha DX7, erster rein digitaler Synthesizer

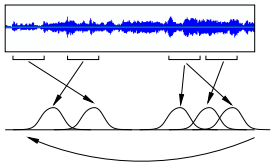
- kein direkter Zusammenhang zw. Parametern und Klang
- kaum zur Reproduktion "natürlicher" Klänge geeignet

Frequenzmodulation: Beispiel



- einfaches Beispiel: Sinus(Sinus) mit Hüllkurve
- FM-Tutorial:
  http://www.geocities.com/SunsetStrip/Underground/2288/2fmsynth.htm

Granularsynthese



- Zusammensetzen eines Klanges aus vielen kleinen "Körnchen"

- Ausgangsmaterial sind Samples / Sinuswellen
- Multiplikation mit Fensterfunktionen
- anschließend Aneinandersetzen und Looping
- "Körner": wenn sich einzelne Stückchen nicht mehr überlappen

- sehr viele Parameter zum "Herumspielen"
- daher auch komplex und wenig intuitiv

- Einführung und Literatur: keyboards 01/01 S.74ff
- GranuLab: http://hem.passagen.se/rasmuse/Granny.htm

Physical Modeling

- warum überhaupt vereinfachte Syntheseverfahren?

"physical modeling synthesis"

- Modellierung realer / imaginärer Schallquellen
- als physikalische Systeme: Saiten, Pfeifen, Resonanzkörper, ...
- Aufstellen der Bewegungsgleichungen

- numerische Lösung der Bewegungsgleichungen
- möglichst in Echtzeit

- im Prinzip beliebig genaue Simulation jedes Instruments
- aber extremer Rechenaufwand
- derzeit meistens auf 1D-Modelle beschränkt (Blasinstrumente)

Structured Audio: Konzept

- algorithmische Beschreibung von Klängen
- mit einer dazu angepassten Programmiersprache
- u.a. durch entsprechendes Zeitmodell (arate / krate)

- Übertragung der Algorithmen anstelle der Klangdaten
- jedes Syntheseverfahren, jeder Effekt möglich
- Decoder führt die Algorithmen aus
- erlaubt perfekte Rekonstruktion der Daten
- aber hoher Rechenaufwand

- bisher keine anwenderfreunlichen Tools verfügbar
- entsprechend wenig Material (Instrumente/Effekte/Musikstücke)
- keine Schutzmechanismen: Instrumente können kopiert werden
- derzeit virtuelle Instrumente fast nur im VST-Format

(Vercoe et.al.,Structured Audio, Proc. IEEE 86-5, 1998)

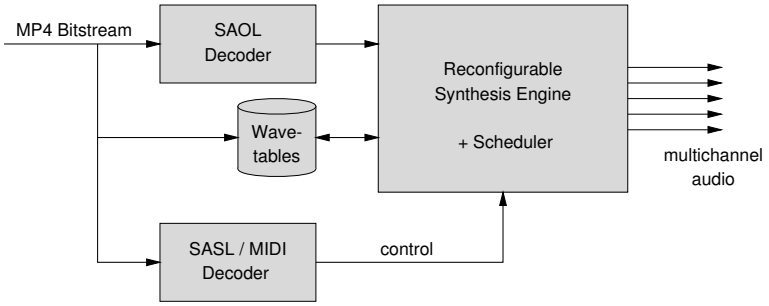
Structured Audio: Begriffe

"instrument"	ein einzelner, klangerzeugender Algorithmus
"orchestra"	Gesamtheit aller Instrumente in einem Bitstream
"score"	("Partitur"): die Kontrollparameter für die Instrumente
"score time"	Zeitpunkt eines Kontrollevents (gemessen in Takten)
"control rate"	Aktualisierungsfrequenz der Kontrollparameter
"audio rate"	Abtast- und Aktualisierungsrate der Instrumente
"sample"	PCM-kodierte Audiodaten (z.B. für Wavetable-Synthese)
"opcode"	parametrisierte DSP-Funktion
"bus"	Speicherbereich zum Datenaustausch zw. Instrumenten
"route"	globale Einstellung des Signalflusses

. . .

(ISO/IEC 14496-3, 1999)

SA: Blockschaltbild



Bitstream:

- enthält DSP-Algorithmen und Initialisierungen
- Sampledaten, soweit notwendig
- Steuersignale und zugehörige Zeitinformation

(Scheirer, MPEG-4 SA, 106th AES, Munich)

SA: Bitstream Header

```
class StructuredAudioSpecificConfig {  // bitstream header
  bit(1) more_data = 1;
  while (more_data) {  // shall have at least one chunk
    bit(3) chunk_type;
    switch (chunk_type) {
      case 0b000: orc_file orc; break;
      case 0b001: score_file score; break;
      case 0b010: midi_file SMF; break;
      case 0b011: sample single_sample; break;
      case 0b100: sbf sample_banK; break;
      case 0b101: symbtable sym; break;
    }
    bit(1) more_data;   // 1 = more data, 0 = end-of-stream
  }
}

class SA_access_unit {  // bitstream "event data"
  bit(1) more_data = 1;
  while (more_data) {
    bit(2) event_type;
    switch (event_type) {
       case 0b00: score_line score_evt; break;
       case 0b01: midi_event midi_evt; break;
       case 0b10: sample single_sample; break;
    }
    bit(1) more_data;
  }
}  ...
```


SA: Bitstream

MPEG-4 SA definiert das zugehörige Binärformat

- "symtable" Symboltabelle
- "orchestra" SAOL-Instrumentendefinitionen
- "score" SASL-Kontrolldaten für das orchestra
- "MIDI" MIDI-Kontrolldaten
- "sample data" einzelne Audiosamples oder SASBF-Bänke

- einzelne Einträge gekennzeichnet durch "tokens"
- direkte Navigation ("Vorspulen" usw.) nur eingeschränkt möglich

Auswertung des Bitstroms über globalen Scheduler:

- Intialisierung der globalen Daten (z.B. Abtastrate, Symboltabelle)
- Vorbereitung der "orchestra"-Algorithmen
- Parsen der "score" und "MIDI"-Steuerdaten
- Abspielen

SAOL

"structured audio orchestra languange":

- Sprache zur Beschreibung von Audio-DSP Algorithmen
- basiert auf CSOUND und Netsound

- Zeitmodell unterscheidet "arate" und "krate"
- arate: Audio-Abtastrate, global einstellbar
- krate: "Kontroll-Rate" für Parameteränderungen
- Audio-Algorithmen werden einmal pro Abtastperiode berechnet
- Kontroll-Algorithmen einmal pro K-Periode

diverse "built-in" Funktionen, u.a.

- stückweise lineare/exp. Interpolation
- elementare Synthesefunktionen (Sinus, Rauschen, ...)
- elementare Filter, FFT/iFFT,

SAOL: Codebeispiel

```
instr beep (pitch, amp) {
  ksig  vibfreq, env;                 // control signals
  asig  sound, f1, f2;                // audio signals

  table vibshape(harm,128,1);         // sine wave table

  vibfreq = cpsmidi(pitch)            // convert MIDI pitch to freq.
        * (1 + koscil(vibshape,5)/40); // add vibrato

  sound = buzz( vibfreq, 0.1, 0.9 );  // predefined noise source
  f1    = bandpass( sound, 500, 100 ); // predefined filter:  formants
  f2    = bandpass( sound, 700 + vibfreq*2, 100 ) / 10;

  env   = kline( 0, 0.05, 1, dur, 0 ); // linear interpolated envelope

  output( (sound+f1+f2) * amp * env ); // mix together
}
```

(aus Scheirer, 106th AES)

SAOL: Operatoren, Funktionen

- Datentypen int, float, table, ...
- Arithmetik, mathematische Funktionen (sin,cos,exp,log,...)
- universelle FIR-/IIR-Filter, diverse vordefinierte Filter
- FFT-/iFFT-Transformation, Fensterfunktionen
- Wavetablesynthese, Unterstützung für DLS-2 (SASBF)
- Granularsynthese, ...

- elementare Signalgeneratoren (Sinus, Rauschen, ...)
- delaylines, Reverb, Chorus, div. Effekte
- universelles Signal-Routing ("bus routing")

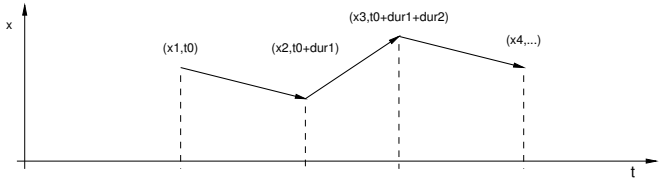
- MIDI-Integration und Konvertierungsfunktionen
- stückweise lineare/exp. Interpolation für Audio-/Kontrollsignale
- benutzerdefinierte Opcodes

SAOL: Beispiel "piano"

```
instr piano(pitch, vel) preset 1 92 0 4 {  // pitch, vel in MIDI
  ivar cps,scale,t,l,b;  ksig off;
  asig samp,env;

  table pitchmap(step,-1,0,0,28,1,38,2,46,3,55,4,64,5,69,6,200);
  table loop(data,-1,197585,151849,137125,132916,104286,59649,50774);
  table base(data,-1,21,36,41,51,61,66,71);

  imports table A0,C2,F2,Ds3,Cs4,Fs4,B4;   // global definierte Samples
  tablemap pno(A0,C2,F2,Ds3,Cs4,Fs4,B4);
  ...
  if (released) {    extend(0.01);
    env = aexpon(1,0.01,0);               // exp. release envelope
  }
  else { env = 1; }                       // play sample during attack

  pitch = pitch -12;
  t     = tableread(pitchmap,pitch);  // which sample for this note?
  l     = tableread(loop,t);          // read loop settings
  b     = cpsmidi(tableread(base,t)); // get sample base freq.
  cps   = cpsmidi(pitch);             // convert MIDI pitch to freq.
  samp  = loscil(pno[t],cps,b,l);     // wavetable synth

  scale = (vel/100)*(vel/100);        // simple velocity mapping
  env   = env * scale;
  output(samp * env * (1-(pitch/100)),samp*env*pitch/100);
}
```

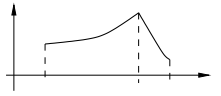
(elpelele.saol)

SAOL: aline, kline




```
kline( ivar x1, ivar dur1, ivar x2 [, ivar dur2, ivar x3, ...]);
aline( ivar x1, ivar dur1, ivar x2 [, ivar dur2, ivar x3, ...]);
```

- built-in Funktion für stückweise lineare Interpolation
- sowohl für Audio- als Controllerdaten
- einfache Beschreibung von Zeitsignalen
- aexpon für Interpolation mit exp()-Funktion
- arand, alinrand, ...: Zufallssignale



SASL

"structured audio score languange":

- separate "minimale" Sprache
- einfaches Textformat, effizientes Binärformat

- Beschreibung von "Kontrollereignissen":

  - globale Tempoangabe (bzw. -änderungen)
  - Erzeugen, Aktivieren, Löschen von Instrumenten
  - Aktivieren von "Noten"
  - Kontrollparameter für Instrumente

- Konzept wie MIDI / SMF
- aber explizite Zeitangaben / keine vordefinierten Controller

SASL: Beispiel

```
0.000 control ls_freq 100
0.000 control ls_gain 1
0.000 control bf_freq 500
0.000 control bf_gain 1
0.000 control bf_Q 1
0.000 control fad_value 0.4
0.000 control pp_value 0.5
0.000 oscillator 3 440
0.200 control fad_index 0
0.200 control fad_value 0.4
0.300 control fad_index 1
0.300 control fad_value 0.45
1.200 control fad_index 0
1.200 control fad_value 0.3
...
0.200 control pp_index 0
0.200 control pp_value 0.4
0.300 control pp_index 1
0.300 control pp_value 0.6
...
6.000 oscillator 3 220
7.000 control ls_index 0
7.000 control ls_freq 250
7.000 control ls_gain 1.6
9.100 end
```

für jedes einzelne Event:

<Zeitangabe>

<Instrument>

<Kontrollparameter>

<neuer Wert des Parameters>

MIDI: Motivation und Aufgabe

MIDI := "Musical Instruments Digital Interface"

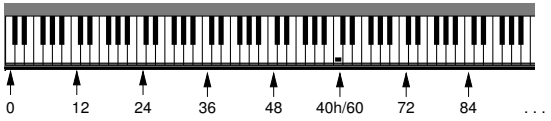
Hardware und Protokoll:

- Spielen von Synthesizern
- Gerätesteuerung / Parametereinstellung / Automation
- "Sequencer" zur MIDI-Aufzeichnung / Editieren / Wiedergabe

- derzeit praktisch alle Musik-Geräte mit MIDI-Schnittstelle
- z.B. Kombination beste Tastatur / bester Klangerzeuger
- auch für weitere Geräte, z.B. Lichtanalagen

- billig und robust (Hardware: RS232 8N1 31250baud optisch)
- extrem kompaktes Dateiformat: Musik mit 1 kbps (!)

MIDI: Tonhöhe



Konzept der Tonhöhe (pitch):

- Tasteninstrument mit festen Tonhöhen
- wohltemperierte Stimmung
- kodiert als ein Byte, 0 .. 127
- "Schlüssel-C" = 60, C# = 61, D = 62 usw.

- andere Skalen möglich: "tuning"-Meldungen
- Feintuning via "pitch wheel" Meldungen (14 bit Auflösung)

MIDI: Channel messages

8K	note	dyn		note off
9K	note	dyn		note on
AK	note	dyn		poly after touch
BK	ID	value		control change
BK	00	BN	CK	PN bank select
CK	PN			program change
DK	value			channel pressure
EK	val	val		pitch wheel change, 14 bit
F0	Hersteller ID	7 bit data	parity	F7 sysex data

K	channel 0 .. F
note	pitch 0 .. 127
dyn.	attack / release
ID	controller-ID
BN	bank number

MIDI: note on, note off

90	40	7F		80	40	7f		90	41	52		80	41	06	

90h	note on	80h	note off	90h	note on	80h	note off
40h	middle C (=64)	40h	middle C	41h	c#1	41h	c#1
7Fh	very hard (forte)	7Fh	very hard	52h	medium	06h	very soft

(hier: alle Meldungen für Kanal 0)

"keyboard"-orientiert:

- feste Tonhöhen
- Anschlagstärke (attack/release)
- Modulation via "controllers"

- "all notes off" für Paniksituationen

BK	78	00

SASL: vs. MIDI

SASL:

- keine Beschränkung der max. Datenrate
- gleichzeitige Events möglich
- kontinuierliche, beliebig feine Kontrollmöglichkeiten
- "lesbarer" Quelltext und kompakter Bitstrom
- bisher nicht in Musikinstrumenten implementiert

MIDI:

- der de-facto Standard, riesiger Fundus an MIDI-Dateien
- in allen Geräten/Instrumenten implementiert
- zur Echtzeit-Kontrolle
- aber schlechte Zeitauflösung, geringe Datenrate

=> SA nutzt SASL und/oder MIDI zur Steuerung

SA: Tools

bisher nur wenige SA-Programme verfügbar:

saolc	SAOL/SASL-Compiler, MIT Media Lab Referenzsoftware für den MPEG-Standard
sfront	SAOL/SASL-Compiler, UC Berkeley nutzt C als Zwischenformat und gcc als Compiler
saint	SA INTerpreter, EPFL Lausanne direktes Ausführen von SAOL/SASL-Code

- interaktive Steuerung jeweils nur stark eingeschränkt möglich
- meistens nur WAV/AIFF-Ausgabe, Direktausgabe problematisch
- für Musiker bisher kaum brauchbar
- SA Webseite mit Tools: www.saol.net

(sound.media.mit.edu/mpeg4, www.cs.berkeley.edu/~lazarro/sa, lsiwww.epfl.ch/saog)

SA: SAOLC

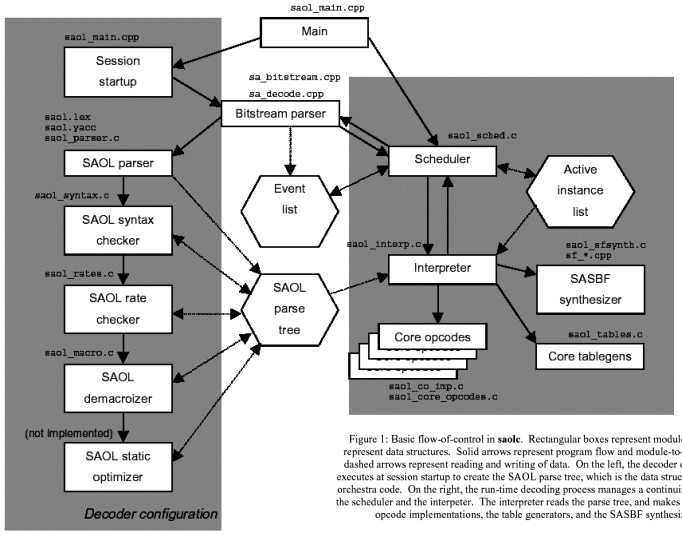


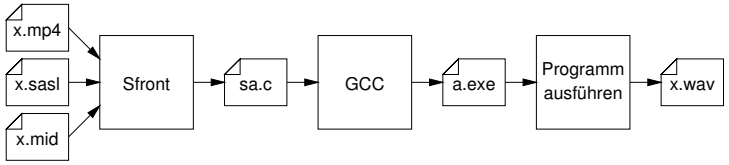
Figure 1: Basic flow-of-control in **saolc**. Rectangular boxes represent modules, hexagonal boxes represent data structures. Solid arrows represent program flow and module-to-module connections; dashed arrows represent reading and writing of data. On the left, the decoder configuration process executes at session startup to create the SAOL parse tree, which is the data structure that represents the orchestra code. On the right, the run-time decoding process manages a continuing interaction between the scheduler and the interpeter. The interpreter reads the parse tree, and makes use of the various core opcode implementations, the table generators, and the SASBF synthesizer as needed.

SASBF:

"SA sample bank format"

- Beschreibung von Sampledaten für Wavetable-Synthese
- entspricht dem DLS-2 Format (Microsoft DLS, Creative SoundFont2)
- "opake" Einbettung in MPEG-4 Bitstrom:
  - RIFF-Dateiformat, diverse Chunk-Typen definiert, u.a.:
  - einzelne Audiosamples, zugehörige Loop-Einstellungen
  - Kombination zu Multisamples / Velocity-Layers
  - Filter und Amp-Hüllkurven
  - diverse Effekteinstellungen
- erlaubt Zugriff auf DLS2/SF2 Samplelibraries
- Parametersteuerung über SASL oder MIDI

SA: Sfront



- parst SA-Binärformat, erzeugt SAOL/SASL-Quelltext
- übersetzt SASL/SAOL Quellcode nach C
- anschliessend Aufruf des C-Compilers (gcc)
- fertiges Programm berechnet die originalen DSP-Algorithmen
- portabel, Versionen für Unix/Linux/Windows verfügbar
- aber: kein interaktives Arbeiten
- `sfront -orc piano.saol -sco prelude.sasl -bitout prelude.mp4`

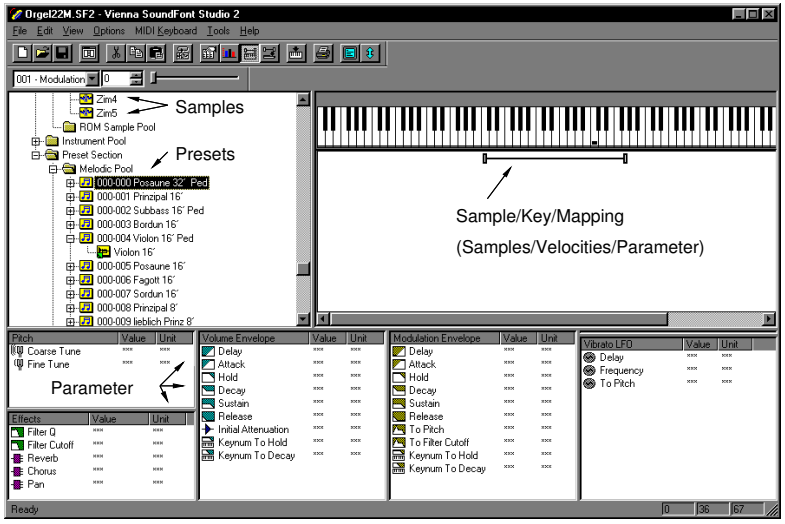
(www.cs.berkeley.edu/~lazarro/sa/)

SoundFont:

- Dateiformat für Samplebänke, Creative Labs 1997
- ursprünglich für Soundblaster 64 AWE
- RIFF-Struktur, insgesamt 22 Chunk-Typen:

RIFF
- SFBK
  - INFO-list
    - header chunks: name, version, ...
  - sdta-list
    - sample data
  - pdta-list
    - presets (headers, list)
    - instruments (headers, list)
    - sample headers

SoundFont: Vienna Editor



VST: Beispiel AGgain-Plugin

```
#include "AGain.hh"

AGain::AGain( audioMasterCallback audioMaster )
        : AudioEffectX( audioMaster, 1, 1 )  // 1 program, 1 parameter
{
  fGain = 1.0;              // default gain 0 dB
  setNumInputs( 2 );        // stereo in
  setNumOutputs( 2 );       // stereo out
  setUniqueID( "AGain" );   // unique name for this plugin
  canMono();                // ok to feed with input with same values
  canProcessReplacing();    // accumulate / overwrite
  strcpy( programName, "default" );
}

void AGain::setParameter( long index, float value )
{
  fGain = value;
}

...

void AGain::process( float **inputs, float **outputs, long n_samples )
{
  float *in1 = inputs[0];  float *out1 = outputs[0];
  float *in2 = inputs[1];  float *out2 = outputs[1];

  while( --n_samples >= 0 ) {
    (*out1++) += (*in1++) * fGain;   // accumulating: Mixer
    (*out2++) += (*in2++) * fGain;   // should use ISSE/3Dnow
  }
}

...
```


Steinberg: VST

"virtuelle Studio-Technologie":

- Schnittstelle / API für virtuelle Audiogeräte
- Audio- und MIDI-Funktionen, samplegenau
- Integration in Cubase / andere Hostapplikationen

- realisiert als C++ Basisklasse, implementiert für PC/Mac/SGI
- 32-bit Gleitkomma für alle Datenoperationen
- ISSE-Optimierung

- zusätzliche GUI-Wrapper für Oberfläche der Plugins
- minimaler Overhead, optimale Performance
- als Standard etabliert, Dutzende Effekte/Synths/Tools erhältlich

siehe Beispiel:

(www.steinberg.de)

SA vs. VST

SA:

- Sprache für DSP-Algorithmen, viele Operationen vordefiniert
- standardisiertes, portables Binärformat
- bisher keine (für Musiker) brauchbaren Tools verfügbar
- entsprechend geringe Verbreitung

VST:

- sehr einfache Schnittstelle, nur eine C++ Klasse
- "nackte" Schnittstelle: alle Algorithmen, Synchronisation "per Hand"
- direkter Datentransfer, geringer Overhead, geringe Latenz
- direkte Integration in wichtige Tools (Cubase, Logic, ...)
- volle Echtzeitkontrolle
- Dutzende von Freeware und kommerziellen Synths

=> zukünftige Rolle von SA?

AudioBIFS:

"Binary Format for Scene (descriptions)":

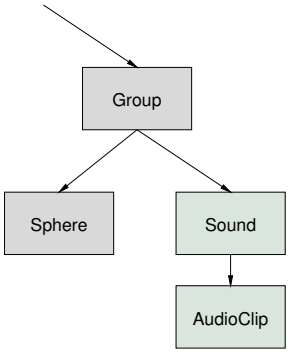
- hierarchische Beschreibung von Objekten im 3D-Raum
- AudioBIFS für Schallquellen
- mit dem "Sound" Knoten
- 3D-Position und Ausrichtung

- basiert auf VRML "Sound"-Objekt
- aber deutlich erweiterte Möglichkeiten

- zusätzliche Knoten für AudioMix, AudioFX, usw.
- Beschreibung von Sound-Effekten und Mischpult
- Schwerpunkt auf Musikproduktion, nicht nur für 3D-Audio

(Huopaniemi, 106th AES workshop, Munich, 1999)

AudioBIFS: VRML Codebeispiel




```
Group {
  children [
    Sphere {
      radius 3
      position 0,0,0
    }
    Sound {
      position 0,0,0
      source
        AudioClip {
          url "http://.../demo.wav"
          loop = 1
        }
    }
  ]
}
```

(siehe Huopaniemi, AES 106, Munich)

AudioBIFS: Features

• Sound	Schallquelle im 3D-Raum, wie VRML
• Sound2D	Schallquelle in 2D
• ListeningPoint	Position der Hörers
• TermCap	Anpassung an vorhandene Geräte (z.B. 5.1 vs. stereo)
• AudioSource	Datenquelle für Sound(2D), inkl. Streaming Anschluss an jeden MPEG-4 Decoder möglich
• AudioDelay	Verzögerung eines Audiosignals
• AudioBuffer	Zwischenspeichern von Audiodaten
• AudioMix	universelles "Mischpult", M-auf-N Kanäle
• AudioFX	Download beliebiger Effekte Beschreibung in SAOL

AudioBIFS: Codebeispiel

```
instr low_shelving(chan) { // instrument low-pass shelving filter
  ivar pi = 3.1415;
  ksig kk, rgain, freq, dgain, a0, a1, a2, b0, b1, b2;
  ksig first, ki, curr_gain, curr_freq;

  imports ivar ls_freq_init, ls_gain_init;        // initial values
  imports ksig ls_freq, ls_gain, ls_index;        // control signals
  asig ls_out[inchannels], this_chan[inchannels]; // audio ports
  oparray port[2];                                // controller inputs

  ... // initialization code

  // get controller input values, update gain and frequency
  dgain = port[0](curr_gain, 0.02);
  freq  = port[1](curr_freq, 0.02);

  // update filter parameters
  kk = sin(pi * freq / s_rate) / cos(pi * freq / s_rate);
  if(dgain >= 1) {  // boost case
    a0 = 1 + sqrt(2 * dgain) * kk + (kk * kk * dgain);
    a1 = 2*(kk * kk * dgain - 1);
    a2 = 1 - sqrt(2 * dgain) * kk + (kk * kk * dgain);
    b0 = 1 + sqrt(2) * kk  + (kk * kk);
    b1 = 2 * (kk * kk -1);
    b2 = 1 - sqrt(2) * kk + (kk * kk);
  } else { ... }  // cut case

  // call IIR filter with current parameters, mix
  ls_out[chan] = iir(input[chan], dgain*a0/b0, b1/b0, dgain*a1/b0, b2/b0, dgain*a2/b0);
  this_chan[chan] = input[chan];
  output(this_chan + ls_out);
}
```