CSL (*sizzle*): The CREATE Signal Library

Stephen Travis Pope
Center for Research in Electronic Art Technology (CREATE)
Graduate Program in Media Arts and Technologies (MAT)
University of California, Santa Barbara (UCSB)
stp@{create,mat}.ucsb.edu

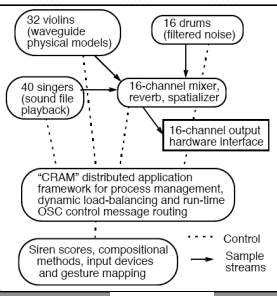
June, 2004

Outline: Intro. to CSL

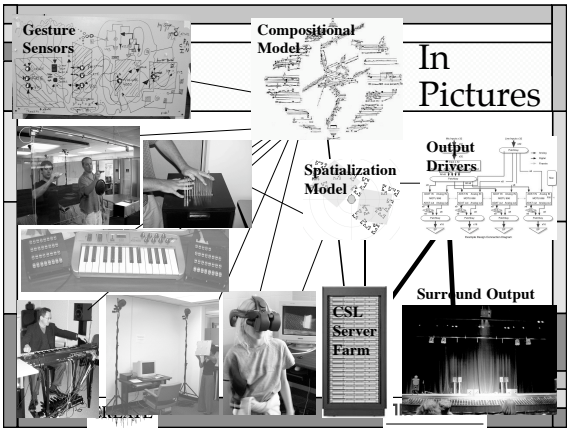
- CSL ("Sizzle"): The CREATE Signal Library for digital audio synthesis & processing
  - Context: CREATE R&D
  - Background and relatives
  - Technical overview
  - Code examples
  - Evaluation, next steps
  - Steps for getting started programming with CSL

CREATE Synthesis/Performance Group Goals

- Support reliable "orchestra-scale" sound synthesis, multi-modal gestural sensing and control, and pluriphonic projection (up to 128 channel output in the CNSI sphere)



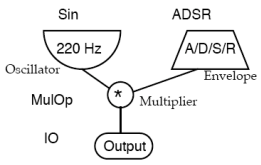
In Pictures



What's CSL ?

Demo

- General-purpose, portable C++ programming framework for real-time digital audio synthesis and processing
- Used for stand-alone applications, plug-ins, OSC/MIDI servers, etc.



CSL Relatives (Software Synthesis)

- Like Cmix, STK, Siren, JSyn, MxV, or CLM
  - Delivered as a fcn/class library in a general-purpose programming language
- Unlike SuperCollider, Csound, Max
  - Not its own language
  - No scheduler
  - Uses C++ development environment

Why on Earth another one???

- Cmix -- old, flaky
- SuperCollider -- different question, complex
- Csound, Music-N -- not languages, source clarity
- Jsyn -- closed DSP kernel
- STK -- PM-centric, tick model
- CLAM -- way complex
- CLM -- who knows LISP?
- Siren/Squeak -- who knows Smalltalk?

Our Requirements

- Simple, easy to learn
- Flexible, multi-purpose
- Portable
- Scalable
- Embeddable
- Distributable
- Network-oriented
- Debuggable

CREATE MAT Media Arts and Technology Graduate Program

CSL Background

- "CREATE Oscillator" -- 1998, CORBA_A/V-based sample streaming, CORBA IDL for instruments
- MAT 240D course (digital audio synth. techniques, Spring '01, '03)
  - CO1 (minimal 1 KLOC), CO2 (full-featured)
  - CSL_lean (redesign from scratch by one person)
  - CSL3 (2004, 25 KLOC, full-featured)
- Designs driven by immediate needs for concrete applications (pieces, theses, etc.)

CREATE MAT Media Arts and Technology Graduate Program

CSL3 Basics: Core Classes

- **Buffer** objects (1 class + helpers)
  - Multichannel non-interleaved sample storage
  - "Smart" object (not just a float**), ptr. mgmnt.
  - Handle malloc/free, filling statistics, etc.
- **FrameStream** classes (Ugens) (many)
  - Respond to the message next_buffer(input, output)
  - Processors have a FrameStream as input
- **Mix-in** classes (vs. wrapper classes)
  - Phased, Positionable, Writeable, Cacheable, etc.

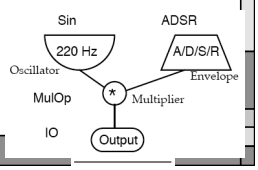
CREATE MAT Media Arts and Technology Graduate Program

Simplest CSL Program

Demo

Sine wave with envelope

```
// Create a sine oscillator -- this is a C++ comment
        Sine osc(220.0);          // frq = 220 Hz
// Create an ADSR envelope -- args are (dur, att, dec, sus, rel)
        ADSR env(3.0, 0.06, 0.2, 0.2, 1.5);
// Create a multiplier for osc & env
        MulOp mul(osc, env);
// Plug it into the (global) output
        globalIO.set_root(mul);
```



CREATE MAT Media Arts and Technology Graduate Program

Sine Osc Alternatives

Processor::set_input()
Ugen::set_scale()
Ugen::set_offset()

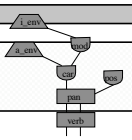
```
// Use the envelope object as a generator and processor (VCA)
    SumOfSines osc(220.0, 1, 5, 0.7…);   // make a sum-of-sines
    Triangle env(3.0);                   // triangle envelope
    env.set_input(osc);                  // send osc as input to env
    gIO.set_root(env);                   // env is root

// Use the osc's  scale (volume control or AM) input
    SquareBL osc(220.0);                 // make a band-lim square
    Gaussian env(3.0, 0.2);              // envelope with bell width
    osc.set_scale(env);                  // set osc scale to env
    gIO.set_root(osc);                   // osc is root
```

CREATE MAT Media Arts and Technology Graduate Program

Reverb'd Panning Dual-Env. FM

(7 Ugens, minimal, procedural style)




```
//// FM instrument with stereo panning and reverb ////
ADSR a_env(1, 0.01, 0.1, 0.1, 0.6);   // create ampl. env., ADSR(dur, a, d, s, r)
ADSR i_env(1, 0.001, 0.1, 0.5, 0.5);  // create FM mod. index env.
i_env.set_scale(110);                 // scale i_env by base freq.
Sine car, mod(110);                   // create 2 sine oscs: carrier & modulator
mod.set_scale(i_env);                 // scale the modulator by the i_env
mod.set_offset(220);                  // add in the base freq.
car.set_frequency(mod);               // set the carrier's freq. to the modulator
a_env.set_input(car)                  // plug the carrier into the a_env's input
Sine pos(0.25);                       // create an LFO for panning
Panner pan(a_env, pos);               // plug the a_env into a stereo panner
Stereoverb verb(pan);                 // plug the panner into a stereo reverb
gIO->set_root(verb);                  // plug the reverb into the output
//  gMixer->add_input(verb);          // or add it to a global mixer
```

Demo

CREATE MAT Media Arts and Technology Graduate Program

CSL FrameStream Details

- Core FrameStream methods
  - `next_buffer(inBuf, outBuf)` - fill in a buffer's worth of frames (input buffer is signal from ADC)
  - `next_sample(inBuf, outBuf)` - 1 sample; adjust phases
  - `is_fixed_over(in)` - is the receiver's value fixed over range?
  - `is_active()` - are a graph's envelopes on?
- Several policies for handling `next_buffer()` with multi-channel I/O buffers: call `mono_next_buffer()` and iterate (vs. copy - FanOut and Splitter/Joiner)

CSL Sources, Controls, and Processors

- **Sources**
  - Oscillators (perfect, BL), SumOfSines, Noise, SoundFiles, Chaotic/IteratedFS, IFFT, Physical Models, Granulators, Signal windows
- **Control**
  - Envelopes, LFOs, LFNoise, ProbDists, DynamicVariables, OSC, MIDI, GUI, CORBA, XML, note lists, Feature extractors, Input followers
- **Processors**
  - Operators, Mixers, Filters/banks, Reverbs, (N-M)Panners, DelayLines, FDN, WaveShape, Lo-latency Convolution, FFT/IFFT, LPC/FIR
- **Support**
  - RingBuffer, ThreadedFrameStream, BlockResizer, RateConvertor, Splitter/Joiner, FanOut (needed), Interleaver/Deint., Test `main()`s
  - Tools: FIR/Reverb IR Design, Spectrum DBs, Control-mapping

Swept Band-pass Filtered Noise

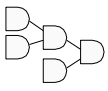
```
void test_filter_sweep() {
    ADSR a_env(3, 0.1, 0.1, 0.3, 1);          // ampl env = std ADSR
    WhiteNoise wnoise;                        // noise generator
    Sine centerSweep(0.5, 500.0, 1500.0);     // args = frq, ampl, offset
    Sine BWSweep(0.3, 50.0, 200.0);
    Butter lpfilter(wnoise,                   // BPF: in, type, ctr, bw
                Butter::BAND_PASS, centerSweep, BWSweep);
    a_env.set_input(lpfilter);
    log_msg("playing filter_sweep...");
    gMixer->add_input(a_env);                 // add to global mixer
}
```

SoundFile Playback Loop in a Thread

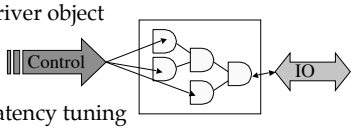
```
// Function that plays random samples from the given sound library
// Sample Libraries look like   vector <SoundCue *> snap;
void * play_wood(void * ignored) {            // Signature for forking as a thread
    SoundCue * voice;                         // Sound cue pointer
    StaticVariable pos(0);                    // Panner position value
    Panner pan( * voice, pos);                // Create panner
    gMix.add_input(pan);                      // Plug panner into the mixer
    while (true) {                            // Loop playing sounds
        voice = snap[rand() % snap.size()];   // Get a sound cue from the library
        pan.set_input( * voice);              // Send it to panner
        pos.set_value(f_rand() * 2.0 - 1.0);  // Set a new position
        voice->trigger();                     // Now trigger the sample
        sleep_sec((voice->duration() / 44100) + (f_rand() * 12.0));  // Sleep a bit
}}                                            // end of loop and function
```

The Big Picture of CSL

- Your basic DSP graph



- Now connect it to control input (OSC, MIDI, GUI, CORBA, XML), and audio IO driver object



- Buffering and latency tuning

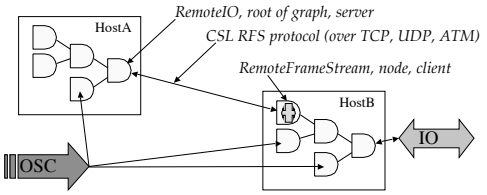
CSL DSP Graph Flexibility

- Sub-graphs can run at different:
  - Sample rates (for control),
  - Buffer sizes (for transforms),
  - Numbers of channels (for efficiency),
  - Buffer formats (interleaved or not),
  - In different threads, etc.
- These can be changed (within reason) at run-time (e.g., for load- or traffic-balancing)

Multi-host CSL Graphs

Demo

- Distributed sub-graph processing with RemoteIO (server) and RemoteFrameStream (RFS, client)
- RFS protocol, (optional) client buffering



RemoteStream/RemoteIO Details

- Uses simple protocol, LAN-oriented (we use switched 1000BaseT & TCP)
- Relatively careful (packet header/trailer, sequence numbers, format packets)
- Double-send optional with UDP/ATM
- RFS client uses ThreadedFrameStream with variable-sized (zero-possible) RingBuffer

Using RemoteFrameStreams

- Server (sample source) side: IO is RemoteIO

```
gIO = new RemoteIO(the_port); // Socket-based IO object
gIO->open();                  // open client socket
gIO->start();                 // start server read thread
                              // server CSL patch follows
```

- Client (sample reader) side

```
RemoteStream rfs("host_name", the_port, 2, buf_size);
Stereoverb verb(rfs);         // reverberate the RFS (e.g.)
gIO ->set_root(verb);         // plug reverb into the (real) output
```


Control, Latency, Scheduling

- All CSL processing is triggered by output requests (pull model, buffer size determines control rate)
- Slow computations should use ThreadedFrame-Streams or transform/convolver threads
- Control may change asynchronously; query `is_processing()` optional (semantics of control)
- Latency determined by buffer size, amount of cacheing in graphs, and RFS remote links (few msec for small buffers, < 1 msec doable [?])
- Dynamic graphs are rare; no time or event models

Instruments and OSC/MIDI/XML

- Instrument object
  - Holds onto a DSP graph; adds "reflective" accessors
  - Server `main()` function loads an instrument library, generates OSC address space or MIDI map (from accessors), and starts a listener thread on a socket
  - Example:

```
                                  // C++ instrument accessor decl.
list[0] = new Accessor("du", set_duration_f, CSL_FLOAT_TYPE);
list[1] = new Accessor("am", set_amplitude_f, CSL_FLOAT_TYPE);
    ... results in OSC address space
        /i1/              instrument 1 OSC commands
          /i1/du:         set-duration command
          /i1/am:         set-amplitude command
```


GestureSensor Drivers & Servers

- Reusable sensor driver framework
  - Serial in, cacheing/differencing/throttling, OSC out
- GestureSensors: receive OSC or MIDI

```
void * mData;          // data array (typically a float *)
char * mCmd;           // OSC command (without the '/')
char * mTypeString;    // OSC type string, e.g., "ffff"
```

  - Event input thread mgmnt
  - Parsing and differencing
  - Map to static or global data or messages
- Subclasses
  - Gloves, Ebeam, Matrix, FOBirds, AdC_Panner, etc.

CSL main() for OSC Processing

```
        // Set up OSC address space root
init_OSC_addr_space();
        // EITHER: add the instrument library OSC addr. space
setup_OSC_instr_library(instrLibrary, numInstruments);
        // OR: create a background thread for a GestureSensor
Thread * aThread = ThreadPthread::MakeThread();
aThread->fork_thread(sensorThreadFcn, & someArgument);
        // Start the I/O callback thread for the global IO
gIO->start();
        // Run the OSC I/O loop function (never returns)
main_OSC_loop(theUDPPort);
```


OSC with a Shell Script

```
    # Shell script to test sending OSC messages to a simple CSL server
    # Create a convenient shell command alias
alias ssoo "sendOSC -h localhost 54321"
    # Play a note ("p" command) on instrument 1 (fm) and sleep
ssoo /i1/p;          sleep 3
    # Set a new "cf" value and play a note on instr 2
ssoo /i2/cf,50.0;    ssoo /i2/p;          sleep 3
    # play an FM note with parameters: dur/amp/car/mod/ind
ssoo /i4/pn,4.0,0.3,220.0,357.4,3.0;    sleep 4
    # load a sound file in instr 8
ssoo /i8/fi,"$CSL_DATA/shine.snd"
    # play a sampled sound with it
ssoo /i8/p;          sleep 1
```


CSL Cross-platform Portability

- MacOSX/Xcode, *nix, Linux/KDevelop, MS-Windows/VisualStudio
- Cross-platform APIs
  - PortAudio for RT sound IO†
  - LibSndFile for sound file IO
  - PortMIDI for MIDI†
  - LibNewRan for probability distributions
  - FFTW for FFT†
  - CyberX3D for VRML, OpenGL†
- Issues
  - C++ compiler, socket/thread code, GUI
  - Base sample data type (float vs int)

† = may use platform-specific APIs (CoreAudio, DSP_FFT, etc.)

Using CSL

- As a library
  - Link a graph and IO into your application, game, GUI, etc.
- For plug-ins
  - AudioUnits or VST with GUIs; call-back to next_buffer()
- For OSC, MIDI, CORBA, XML-RPC, etc. servers
  - Stand-alone instrument groups as soft-synths; RemoteIO
- With CRAM
  - Multi-host control/server/output configurations
- The main() function creates graph or mixer, may spawn threads, then registers an IO call-back object

CSL "beep" main (all of it!)

Demo

```
// Beep_main.cpp -- the simplest CSL "main" program -- a 3-second beep
#include "CSL_All.h"                // CSL "kitchen sink" include
using namespace csl;                // Use C++ CSL namespace
                                    // MAIN -- plays a 3-second beep
int main (int argc, const char * argv[]) {
    PAIO gIO;                       // PortAudio IO object
//  FileIO gIO("beep.aiff");        // OR: use a File IO object
    Sine osc(220);                  // create a sine oscillator at 220 Hz
    gIO.set_root(osc);              // plug it in to the IO
    gIO.open(); gIO.start();        // open/start the IO
    sleep_sec(3);                   // sleep 3 seconds (CSL blt-in fcn)
    gIO.stop(); gIO.close();        // stop/close the IO
    return(0);                      // exit
}                                   // link this with CSL libs, PortAudio, etc.
```


CSL Example: Se/Sp_Sp (2002)

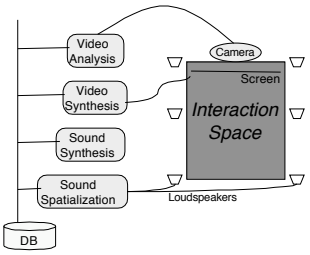
Sensing, computation, multi-presentation (MVC)

Camera-based multi-user sensing (aware space)

Computer vision SW follows mvmnt & grouping among attendees; sends OSC msgs to 3 (Video, CSL) servers

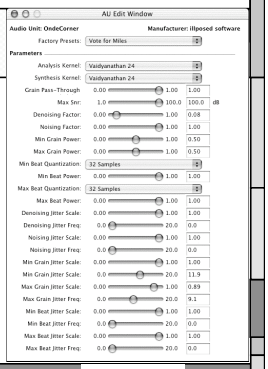
Synchronized multi-camera projection and 6-ch. surround sound

Port from SC2 to CSL2



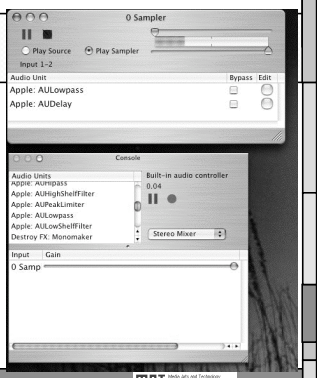
Example: OnDeCorner

- CR's AudioUnit plug-in for experimenting with wavelet transforms
- Pluggable FWT code
- Play to DAC or file



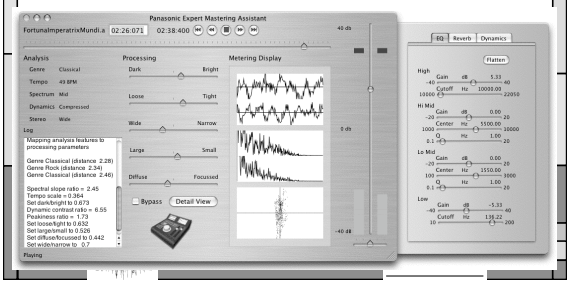
Example: Ouroboros

- CR's AudioUnit host application for processing sound files and live input
- Extensions planned for remote AudioUnits



Example: Expert Mastering Assistant Demo

Process: Analysis, GenreDB, Mapper, DSP, Interact



Generating CSL Graphs/Events

- Using scripting languages
- Smalltalk Slang translator
- From XML
- DragNDrop "patcher" GUIs
- Storing signals and graphs in an OODB
- Instrument libraries and event stores
- Auto-gen of flat namespace for C RMI

Example: LUA Patcher (worked, but failed)

CSL as a library for a scripting language

```
-- Lua program for a panning chaotic oscillator
panning_chaos = function {}
      lorenz = Lorenz{};
      envargs = {0.5, 0.0, 0.0, 0.003, 0.5, 0.5, 0.0};
      envelope = Envelope{envargs};
      panner = Panner2{lorenz, envelope};
      audio_out{panner};
 end
```


So we know it all, right?

- NOT!
- Many open architecture, design, modeling, implementation, deployment, issues
- Some basic choices we're still debating
- Some real dilemmas, limitations, principles
- Tensions between our design bias towards simplicity and "creeping featurism"

Open CSL Design Issues

- Basic models: buffer-based, event-based, signal-based
  - Current pull-model driven by PortAudio and CoreAudi APIs; granularity of events
  - Need a unification of types (semantics) of buffers (samples, FFT frames, FWT frames, IRs, etc,)
  - Signal semantics: operators on buffers vs. procedural ugens?
- How to support dynamic graphs in a simple system (punt)
- That latency thing, polynomial ctrl interpolation, clock sync.

Speed Hacks & Optimizations

- User-visible optimizations
  - is_fixed_over(), is_active() -- used
  - is_linear_over(), is_polynomial_over() -- ?
- Several kinds of buffers (cache optim.)
- Control interpolation?
- DSP graph-to-SMP allocation
- Managed sample-rate conversion
- Better C++ compiler (IBM or Intel/AMD)
- Many interesting optimizations would greatly complicate the system (our guess)

Conclusions

- For our requirements, we really had to start from scratch for most of the components.
- The KISS principle (or XP) paid off in simplicity, flexibility, and ease of use.
- There are many things we could have done other ways (we're still debating; that's the whole fun of it!).
- See create.ucsb.edu/{Siren, CSL, CRAM}

Getting Started Using CSL

- Download zip file or tarball (or subversion/cvs tree)
- Read the README and on-line docs
- Install support libraries (PortAudio, PortMIDI, libSndFile, OSC, FFTW, libnewran, etc.)
- Open C++ project tool for your platform (Xcode, KDevelop, VisualStudio, VI/Makefile)
- Select target main() file
  - Basic demo (start here to make certain you can link & run)
  - Test_mains (edit end of file and run)
  - OSC server, MIDI softsynth, other main()s
- Build CSL kernel libraries and demo target
- Start debugger and run!

CSL Source Organization (Categories)

- Main - Test/demo main() driver functions
- Kernel - Bufers and FrameStreams
- Sources - Oscillators, noise, envelopes, PhysMod
- Processors - Operators, filters, mixers, panners
- IO - IO drivers and LAN streaming
- Utilities - Thread and buffer support classes
- Instruments - OSC/MIDI instrument wrappers
- QT_GUI - Signal view GUI support for QT widgets
- OSC - CNMAT OSC library
- Auralizer - N-channel convolution-based spatializer
- Documentation - README, etc.

Central CSL Header Files

- CSL_Types.h -- the main include file for CSL3: data typedefs and cross-platform macros
- CSL_Core.h -- the CSL Kernel: Buffer, FrameStream, SampleStream, UnitGenerator, MixIn classes
- Gestalt.h -- class CGestalt (system constants)
- Variable.h -- abstract external variable (plug) class
- Oscillator.h -- specification of the base oscillator class and standard waveform generators
- Envelope.h -- The breakpoint envelope classes

Writing a CSL FrameStream Class

```
// Sawtooth oscillator class specification (.h file)
class Sawtooth : public Oscillator {                    // declare class
protected:                      // work-horse method
    status mono_next_buffer(Buffer & inputB, Buffer & outputB,
                            unsigned inBNum, unsigned outBNum);
public:                         // constructors
    Sawtooth();
    Sawtooth(float frequency);
};
```

- Writing the next_buffer() method
- Class Hierarchy

FrameStream - SampleStream - UnitGenerator → Phased Oscillator - Sawtooth

Sawtooth mono_next_buffer() Example

```
status Sawtooth :: mono_next_buffer(Buffer & inB, Buffer & outB,
                                    unsigned inNum, unsigned outBum) {
    sample * bufptr = outB._monoBuffers[outNum];    // samp ptr of out
    unsigned numFr = outputB._numFrames;            // # of frames requested
    float rateRecip = 1.0 / _sampleRate;            // phase increment scale
    for (unsigned i = 0;  i < numFr;  i++) {        // main sample loop
        *bufptr ++ = (_phaseAcc * _scaleC) + _offsetC;// store value to buffer
        _phaseAcc += _freqC * rateRecip;            // incr phase
        if (_phaseAcc  >=  1.0)                     // reset phase
            _phaseAcc  -=  2.0;
    }
    return cslOk;                                   // return OK status
}
```


CSL Processors

- MixIn class *Processor* adds an input SampleStream
- next_buffer method calls Processor::pull_input (inB, outB), possibly using a temp buffer
- This calls input's next_buffer method
- Now the processor operates on source's input buffer into its output buffer
- Filters, panners, etc.

CSL Add-on Packages

- "Advanced" sources
  - SHARC/IFFT additive synthesis, physical models/FDN, granulators, waveshapers
- GestureSensor drivers and OSC mapping
- OSC and CSL instruments
- Auralizer
  - VRML-based geometer, late-reverb modeling, and low-latency distributed many-channel convolution
- HRTF FIRs and HRTF databases (used with OSC head trackers)
- QT GUIs: signal display, control monitoring
- CRAM Interface: CRAM manager service class for CSL servers
- Wavelet code: wave++ discrete wavlet transform

CSL Resources

- CSL Home Page
  http://create.ucsb.edu/CSL
- CSL Downloads (doc, source tarball)
  http://wcreate.ucsb.edu/CSL/CSL_Overview.pdf
  http://create.ucsb.edu/CSL/CSL_ICMC_2003.pdf
  http://wcreate.ucsb.edu/CSL/CSL.tgz
- CSL Mailing List
  http://create.ucsb.edu/mailman/listinfo/CSL
  Send to CSL@create.ucsb.edu

Related Projects at CREATE

- Auralizer & VRML
- Pulsar Generator
- Creatovox
- MusicVisualization
- FMAK DB
- TimeMachine
- InteractEMGroup
- Creatophone
- Time-D Decomp
- SC_3 Work