Arcan

Developer Introduction

Outline

- **Prelude**
  - building / setting up
  - Lua cheat sheet
- **Appl**
  - skeleton
  - images / transformations / …
  - resources
  - advanced example(1)
  - frameservers
  - advanced example(2)
- **Postlude**
  - information sources
  - moving forward

Building / Setting up

- **Basic Dependenices**: cmake > 2.8.12, clang > 3.1 or gcc > 4.6, sqlite3, openal-soft, lua5.1+ or luajit-2.0
- ***Conditional / Optional* Dependencies** (*video platform*, *frameserver support*):
  - (libvlc, ffmpeg, libsdl1.2, libvncserver, libapr, libtsm
- **Quick build:**
  - git clone https://github.com/letoram/arcan.git
    - optional (static luajit, openal, freetype)
      - cd arcan/external/git
      - bash clone.sh ; cd ../../../
  - cd arcan/src/ ; mkdir build ; cd build
  - cmake -DVIDEO_PLATFORM="sdl" ../src
  - make -j 12

Building / Setting up <cont>

- -DVIDEO_PLATFORM="sdl"
    - Video platform is crucial - determines input model, graphics acceleration model (can be overridden with -DAGP_PLATFORM=[gl21,gles2,gles3,vulcan,pixman])
- there are others:
    - egl-dri (native linux etc. graphics)
    - egl-gles (low powered arm boards e.g. raspberry pi)
    - x11, x11-headless (specialized legacy)
    - sdl (easiest to get to work, support X, OSX, BSD, Windows)
- Statically / Tightly coupled and tracked with arcan version due to the volatile/bug- prone downsides to dealing with graphics

Lua Cheat Sheet

Necessary

```
function myfun()
 note = 4; -- default scope is global
 print(_G[“note”]); -- gives 4
 local note = 5;
 print(_G[“note”]); -- gives 4 again!
  a = function(b)
   print(b, note); -- find note in outer
   return 1, 2; -- multiple returns
  end
 a(); -- gives nil, 5;
end

local a = {b = function(c,d)
  print(c,d); end };
  a:b(1); -- will print ref. to a, 1

-- use pairs not ipairs for a[“bah”]=1;
for i,v in ipairs({4,3,2,1}) do
 print(i,v); -- 1,4 then 2,3 etc.
end

print(type(1), type(1.0)); -- all nums
have same type
```

Gotchas

```
a = {1,2,3,4};
print(a[0]) -- nil, 1-indexed!

print(#a); -- 4
a[“test”] = true;
print(#a); -- 4

~= instead of !=
no += -= %= ++ -- etc.
no switch/case/continue

b = (a ? 1 : 2); -- doesn’t work
b = a and 1 or 2; -- does work
```

Appl

- *“a little more than an app, a lot less than an application”*
- pronounced like app- with a deep depressive sigh added at the end, or like app- and then ‘blowing raspberries’
- execution model (think node.js): *asynchronous* (primarily), *event-driven*, *imperative*

- pick a name here (e.g. myappl): restrictions = ([a-Z0-9] [_a-Z0-9])
1 *

- create a matching folder, a .lua script and a function + function_prefix:

myappl\

myappl.lua

contains at least:
function myappl()
end

arcan ./myappl or *arcan /path/to/myappl* or, if myappl exists in *ARCAN_APPLBASE* namespace (don’t worry about that now), just *arcan myappl*

Skeleton

myappl.lua

```
function myappl()
— prepare initial model
end

function myappl_clock_pulse(ts, nticks)
end

function myappl_input(iotbl)
— react to input (lots of info in iotbl)
end

function myappl_shutdown()
— store / save settings
end
```

1. engine sets things up, init.
2. loads / parses appl
3. injects api into lua- context
4. runs main entry point
5. main engine loop {
    1. process event loop
    2. update render model
    3. preframe hook
    4. synch to output
    5. postframe hook
    6. if (~monotonic) time: clock_pulse

}

(advanced) entry points: `_preframe_pulse postframe_pulse display_event`

Images, Transformations…

“fade in a 64x64 px red square”

```
function myappl()
 local vid = color_surface(64, 64, 255, 0, 0); <— starts out hidden!
 blend_image(vid, 1.0, 100, INTERP_SINE); <— reach 1.0 in 100 pulses
end
```

“move / pulse ‘logo.png’ around the edges of the screen for infinity”

```
function myappl()
 local vid = load_image(“logo.png”, 64, 64);
 if not valid_vid(vid) then return shutdown(“missing logo.png”); end
 blend_image(vid, 1.0, 40);
 blend_image(vid, 0.0, 40);
 move_image(vid, VRESW - 64, 0, 20);
 nudge_image(vid, 0, VRESH - 64, 40);
 move_image(vid, VRESH - 64, 0, 80);
 move_image(vid, 0, 0, 40);
 image_transform_cycle(vid, true);
end
```

wait, ‘logo.png’ comes from where?

Resources
(simplified)

Action

input_function
(system_load, load_image, …)

not found? fallback.

Datastore

Myappl

Resources

Database

Properties

unique to each appl and is (default) read/write, but can be restricted to read-only

key/value
store/load

1. for appl- specific settings
2. read-only execution model for loading other programs (separate tool, arcan_db to modify).

fonts, state snapshots, debug logs and other sensitive data
all have separate namespaces that can be remapped before starting
but defaults to being mapped to subpaths in resources)

Advanced Example (1)

(allow one active external connection)

```
function myappl()
 ext = target_alloc(“example”, external_event);
  show_image(ext);
end

function external_event(source, status)
  if (status.kind == “resized”) then
    resize_image(source, status.width, status.height);
  elseif (status.kind == “terminated”) then
    delete_image(source);
    ext = target_alloc(“example”, external_event);
    show_image(ext);
  end
end

function myappl_input(iotbl)
  if (valid_vid(ext, FRAMESERVER)) then
    target_input(ext, iotbl);
  end
end
```

1. set up an external listening connection

2. make sure video object matches the size of the connected client

3. forward all input to any connection (if alive)

to test: start, set ARCAN_CONNPATH=“example” (env.) and afsrv_terminal (if built)

Frameservers

(named in honor of the virtual dub project)

- Semi-trusted separate processes most commonly managed through related functions (launch_avfeed, launch_target, …)

- Also used for controlling external connections (previous example)

- Build-time probed configuration of available *archetypes* (terminal, game, avfeed, decode, encode, removing, …)

  - relates to event handling semantics, sandboxing profile, firewall rules etc.

  - available ones are shown in the global FRAMESERVER_MODES

- Can be replaced with custom set of other implementations: in-house / custom / even proprietary (blergh!)

  - default ones are ‘simple references’

Advanced Example (2)

(offscreen render video input)

```
function myappl()
 if not string.find(FRAMESERVER_MODES, “decode”) then
  return shutdown(“built without decode support”, EXIT_FAILURE);
 end
 ext = launch_decode(“test.avi”,
  function(source, status)
  — don’t care
  end
 );
 if not valid_vid(ext) then
  return shutdown(“missing test.avi”, EXIT_FAILURE);
 end
 square = color_surface(64, 64, 0, 255, 0);
 rotate_image(square, 45);
 show_image({ext, square});
 buf = alloc_surface(VRESW, VRESH);
 define_rendertarget(buf, {ext, square}, RENDERTARGET_DETACH);
 blend_image(buf, 1.0, 50);
 blend_image(buf, 0.0, 50);
 image_transform_cycle(buf, true);
end
```

(and a pulsating square even if decoder or video is broken)

Information Sources

- **Doc/ folder**
  - All exposed Lua API functions have a corresponding function name in doc/*.lua
  - These can be converted to man-pages (cd doc; ruby docgen.rb mangen; -> doc/mantmp)
    - Installed with normal build make install to man- accessible destinations (man 3 load_image), though might not want installed for namespace reasons
- **Wiki sources** (https://github.com/letoram/arcan/wiki)
  - Overview of functions, terminologi, detailed design descriptions, …
- **arcan -g -g** <- increase debug level to get more verbose execution output
  - if respath (e.g. arcan -p res) has a subdirectory ‘logs’, will be populated with both _warning.txt, _error.txt, crash states and frame server output.
  - system_snapshot(“dstfile.lua”); <— explicitly generate a snapshot of existing data-model, helpful to understand internal representation

Doc example

```
-- load_image
-- @short: synchronous load supported images
-- @inargs: resource, *startzv, *desw, *desh
-- @outargs: VID, fail:BADID
-- @longdescr: lots of text goes here
-- @note: use- comments, special cases etc.
-- @group: image
-- @cfunction: loadimage ( see engine/arcan_lua.c )
-- @related: load_image_asynch

function main()
#ifdef MAIN
  vid = load_image(“demoimg.png”);
  show_image(vid);
#endif

-- C preprocessor (cpp) used to generate good and bad examples for
-- automated testing and for manpages
#ifdef ERROR
  vid = load_image();
#endif
end
```

Moving Forward

- IRC, #arcan on freenode (chat.freenode.net)
- Exercises on wiki (github.com/letoram/arcan/wiki/Exercises)
  - Solutions appear in tests/exercises
- Design Slides @ https://speakerdeck.com/letoram/arcan-design