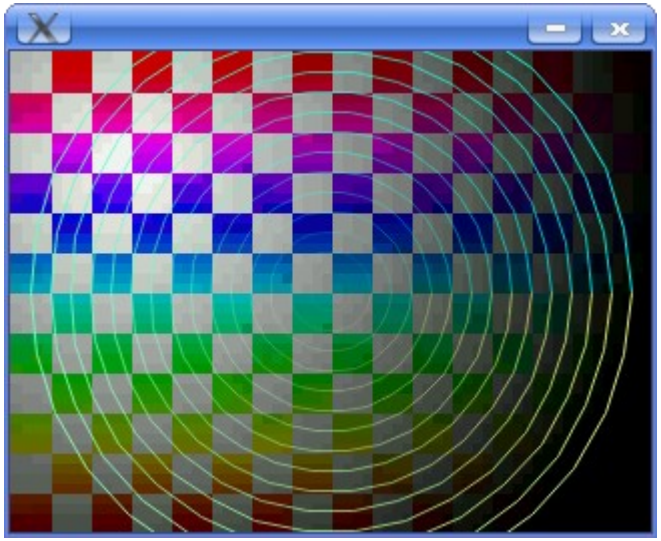
The CRT X-Y Library

Draw lines, make games.



What Is libcrtxy?

Specs:
Graphics library on top of libSDL
Draws lines
Doesn't do much more!
Meant to be scalable

Purpose:
Make it easy to (encourage, in fact) write classic arcade-style vector games

The name:
"X-Y" were a kind of CRT screen in arcade games. (Plain "libxy" was taken)

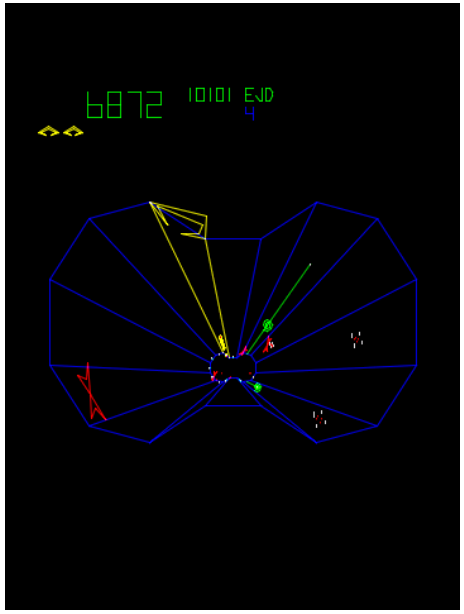
Example Classic Games

Star Wars

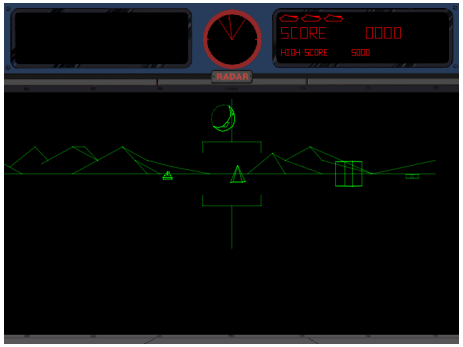
Asteroids

Lunar Lander

Tempest



Battlezone



*** Photos: klov.com**

Scalability

- Screen size (obviously)
  - Should be independent of gameplay
- Rendering quality
  - Alpha-blending
  - Anti-aliasing
  - etc.
- Encourage game logic portability
  - Fixed-point math for lines
  - FPS independence
- Backends
  - SDL bitmap surface
  - OpenGL *
  - OpenGL ES *

* Eventually?!

Scalability Screenshot

User-centricity

User decides backend, rendering quality, etc.

User even decides screen size!

* "User" in this case may also include 'packager' — as in the person who ports/packages your game for some particular environment, such as a handheld Linux PDA.

Via configuration files...

- libcrtxy - global (/etc/libcrtxy/libcrtxy.conf)
- libcrtxy - local (~/.libcrtxyrc)
- application - global (/etc/SOMEGAME.conf)
- application - local (~/.SOMEGAMErc)

Via libcrtxy environment variables...

- CRTXY_ANTIALIAS=OFF, CRTXY_WIDTH=640, CRTXY_HEIGHT=480, etc.

Via libcrtxy command-line options to application (a la standard Qt options to KDE apps)

- --crtxy-antialias off, --crtxy-width 640, --crtxy-height 480, etc.

Using libcrtxy: Overview

Compiling with libcrtxy:
gcc mygame.c -c `crtxy-config --cflags`
gcc mygame.o -o mygame `crtxy-config --cflags --libs`

```
#include "crtxy.h"

int main(int argc, char * argv)
{
    XY_fixed n;
    XY_options opts; // Struct to store options for
    init'ing

    XY_default_options(&opts); // Set hard-coded defaults
    XY_load_options(&opts); // Read libcrtxy config files
    XY_load_options("~/.WHATEVERrc", &opts); // Read our
    conf
    XY_parse_envvars(&opts); // Abide by env. vars
    XY_parse_options(argc, argv, &opts); // Read command-
    line

    n = 10 << XY_FIXED_SHIFT; // Canvas will be '10x10'
    XY_init(&opts, 10, 10); // Init libcrtxy
    …
```

Using libcrtxy: SDL Event Loop

```
do
{
    XY_start_frame(30); // Max out at ~30fps

    while (SDL_PollEvent(&event)) // You just use libSDL...
    {
        // Deal with all key, mouse, joystick & timer
        events.
        // (Funcs provided to convert canvas<->screen
        coords.)
    }

    // Move things... (your game logic)
    // Draw things... (using libcrtxy drawing funcs.)

    XY_end_frame(XY_true); // Max out at ~30fps (see above)
}
while (!done);
```

XY_end_frame() will delay to prevent going faster than max FPS if given a 'true' argument, otherwise will SDL_Delay(1) to give OS some time.

Return value of XY_end_frame() can be used (if not throttling) when calculating how things should move.

Using libcrtxy: Frame Rates

If **throttling FPS** via:

```
XY_start_frame(SOME_FPS);
...
XY_end_frame(XY_true);
```

then your math can remain simple:

```
ship_x = ship_x + ship_speed;
```

All movement may slow down if the system gets bogged down, though.

If running **frame-rate-independent** via:

```
XY_start_frame(0);
...
ticks_since = XY_end_frame(XY_false);
```

then math is affected by how many milliseconds it's been since the last frame ended:

```
ship_x = ship_x + (ship_speed * ticks_since) / 100;
```

In other words, if little time passed since the last frame, don't move things in as large a step as if more time passed.

Using libcrtxy: Fixed-point math

Possibly slower than floating-point on systems with FPUs... but I'm actually more worried about systems *without* FPUs (handhelds, mobile phones, internet tablets, etc.)

`1 << XY_FIXED_SHIFT` is "1.0" in XY_fixed terms.
`c = XY_mult(a, b)` is "c = a * b"
`c = XY_div(a, b)` is "c = a / b"

Also:
XY_fpart(3.6) — fractional part ... (0.6)
XY_ipart(3.6) — integer part ... (3.0)
XY_round(3.6) — round up to nearest integer ... (4.0)
XY_rfpart(3.6) — "1 - XY_fpart()" ... (0.4)

And:
XY_cos()
XY_sin()

Lines and points are given in "XY_fixed" fixed-point values, in terms of 'canvas' size (given to XY_init()). That is then scaled up/down to the actual screen size (set in the XY_opts by whatever means the user gave it to us — config file, env. vars, command-line).

Using libcrtxy: Drawing lines

XY_setcolor(R, G, B, A)
    sets color and alpha, returns an XY_color

XY_drawline(x1, y1, x2, y2, color, thickness)
    draws a line

XY_drawpoint(x, y, color, thickness)
    draws a point

Yeah, that's really all you can do! :^)

* Thickness not yet supported

Using libcrtxy: Line Groups

Getting a little like OpenGL...

XY_new_lines()
creates a new "XY_lines" and returns pointer to it

XY_add_line(lines, x1, y1, x2, y2, color, thickness)
adds a line to an XY_lines group

XY_draw_lines(lines)
draws them!

XY_start_lines(lines)
removes all lines from an XY_lines group (you can reuse)

Also:
XY_duplicate_lines(lines) — makes a copy, returns ptr. to new
XY_translate_lines(lines, x, y) — translates them by (x,y)
XY_scale_lines(lines, xscale, yscale) — scales them*
XY_rotate_lines(lines, angle) — rotates them*

* Around (0,0) origin

Doxygen for docs - Example

I'm learning doxygen... bear with me!

Add specially-formatted comments to code to describe types, functions, their args and their returns...

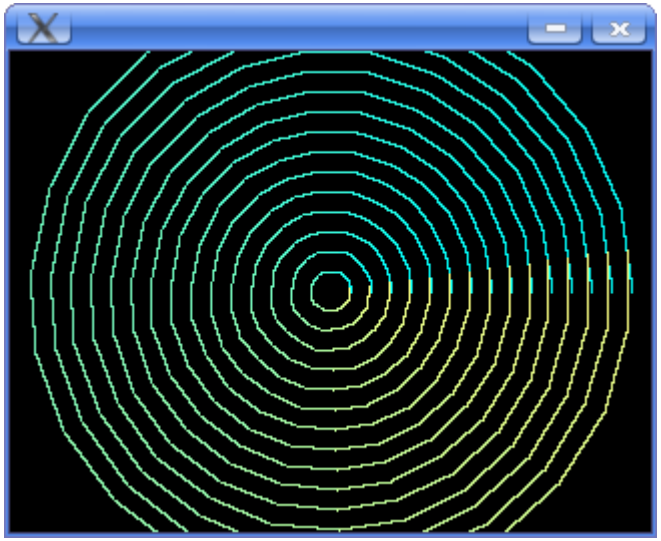
```
/**
 * Duplicates a collection.
 *
 * \param lines is an \ref XY_lines pointer from
 * which you want to copy.
 * \return a pointer to a new \ref XY_lines with all
 * lines from 'lines' copied
 * to it on success, or NULL on failure, and sets
 * error code to one of the
 * following:
 * \li \ref XY_ERR_MEM_CANT_ALLOC
 */
XY_lines * XY_duplicate_lines(XY_lines * lines);
```

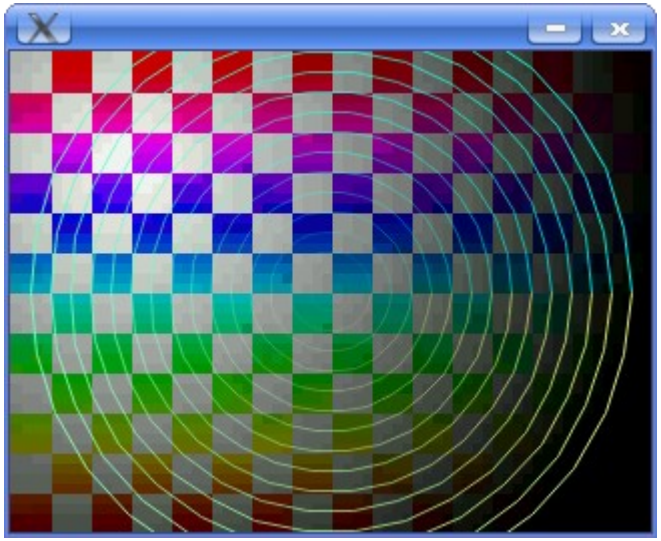
Doxygen for docs - Toil

Still figuring out best way to generate 100% of the HTML docs via doxygen.
(Not just API stuff, but discussion of purpose, how to compile and install lib, how to compile against lib, etc.)

Still figuring out best way to generate *sensible* man pages via doxygen.
(e.g. "man XY_init" should Do The Right Thing)

Q & A and Demos





Links

Home page:
libcrtxy.sourceforge.net

SourceForge project:
www.sourceforge.net/projects/libcrtxy

From the above, get to:
docs, CVS repository, mailing list, etc.

Bill Kendrick:
bill@newbreedsoftware.com

Thanks!