Notcurses
blingful TUIs and character graphics

nick black <nickblack@linux.com>
for FOSDEM 2021 (2021-02-06)
who are you, what's all [gestures] this?

- hacker from the southeastern us (atlanta)
- normally an hpc/compilers guy
- wrote two large NCURSES programs
  - didn’t care to write another

- Notcurses: an intellectual descendant of Curses...
  - ...but neither attempts nor claims compatibility

Goal: become the de facto TUI/character graphics library for new applications
text user interfaces / character graphics

Defined by two properties:

- **Text mode**: drawable unit is a glyph, not a pixel
- **Stream I/O**: we read and write to streams
  - (rather than a framebuffer)

- Linux/FreeBSD virtual console
- terminal emulator under X/Wayland
- pseudottys (e.g. ssh or a terminal multiplexor)
- and even hardware terminals.
character sets and control sequences

We draw with *glyphs*, styling them (and moving around the terminal) with control sequences. Glyphs correspond to EGCs—Extended Grapheme Clusters.

The glyphs available are a function of our (process-scope) encoding, our font (outside of our control, and often unknowable), and our terminal.

- Notcurses supports only the UTF-8 encoding of Unicode (UCS), or ANSI_X3.4-1968 (ASCII). Unicode is *strongly* preferred.

Control sequences are a function of our terminal, abstracted via terminfo.

- Make sure your **TERM** environment variable is accurate!
why not X/Open (now SUS4) Curses?

- Limited Unicode support
- Limited threading support
- PseudoColor and crufty colorpair system
- Control via global variables
- Identifier naming all over the place
- Basic functionality available only as extensions

NCURSES is a superb implementation (and extension) of Curses. The limitations of Curses are fundamental to, and embedded in, the Curses API.
what of other existing TUI libraries?

There is no shortage of alternative TUI/CG libraries (several dozen at least), at various levels of completeness / activity, in many different languages.

None of them met all my goals for a Curses successor.
design goals for a 21st century Curses—Notcurses (1/4)

- Written and usable in C, but intended for use in safer languages
  - C is the base language / lingua franca of UNIX and its system calls
  - We can wrap it in just about any other language
  - Notcurses was designed in conjunction with c++/rust wrappers
  - Very real performance results, as we'll see later

- Two modes to address two major application styles
  - Direct mode: works with standard I/O, for batch/line-driven CLIs
  - Rendered mode: renders and blits frames for fullscreen TUIs
  - Code extensively shared between the two, single shared library
design goals for a 21st century Curses—Notcurses (2/4)

- Designed for use with multiple threads, meaning:
  - Areas of safe concurrent use are clearly delineated
  - Designed to maximize the area of safe parallel intersections

- Generalize drawing surface support
  - Surfaces can be any size, and are free to be partially/totally offscreen
  - Z-axis (total ordering for presentation) within each pile
  - Binding (directed acyclic forest) within each pile, with resize cascades
  - Three independent channels per cell: glyph, fg color, bg color
design goals for a 21st century Curses—Notcurses (3/4)

- **TrueColor, color blending, and default colors**
  - Default colors are taken from the terminal configuration
  - Allows transparency to the desktop, if so configured
  - Use default colors when reasonable; they allow a degree of user configuration
  - Palette-indexed PseudoColor to minimize bandwidth when possible

- **Multimedia support**
  - Sits atop FFmpeg or OpenImageIO (GStreamer coming soon), or null implementation
  - State-of-the-art quad- and sexblitters
design goals for a 21st century Curses—Notcurses (4/4)

- **Widgets (in both rendered and direct mode)**
  - High-res progress bar (8 steps per cell)
  - Selector, multiselector, tree selector
  - Menus, plots
  - Freeform input boxes (libreadline in direct mode)
  - Several types of boxes, polyfills, rotations

- **Perf domination!**
  - O(1) translation, z-axis move, reparenting, destruction
  - Optimal rendering and rasterization
  - Extensive profiling, performance tracking as part of CI, 25 benchmarks in notcurses-demo
A Tour of Notcurses
direct mode: style your `printf()`s

- Best for scrolling, line-based UIs.
- Use regular stdio, plus `ncdirect_*()` functions to emit control sequences.
- The cursor can be moved/located, and screen coordinates determined
  - Useful for e.g. spinners and multiline widgets
- Images can be rendered, boxes and progress bars drawn, etc.

For more complex graphics, including forms, plots, and just about anything that needs rapidly update the full screen, we instead use...
rendered mode

- Rendered mode operates like more of an OpenGL model
- Create surfaces (ncplanes), draw on them, order them
- Render a collection of surfaces (ncpile) to a frame
- Rasterize a frame to the terminal as a unit
- Only upon rasterization is the output area changed
  - Operations between rasterizations are wholly virtual
- Only updated cells are emitted
- Pile/plane state persists across rendering operations
- Can achieve thousands of FPS
- Can use the *alternate screen* where available
- Mixing with standard I/O will result in madness
rendered mode operation model

- A frame is rendered from a single pile
- Piles are entirely independent
  - Multiple threads can freely mutate multiple piles
- A pile is rendered using the Painter's Algorithm
  - Only cells overlapping the visual area are rendered
  - Painting proceeds from higher planes to lower ones
  - An output cell is solved when we have a defined glyph, fg, and bg
- No need to use multiple piles except for parallel performance
  - Can also be convenient for multimodal UIs
- Rasterizing a frame syncs the visual display to the frame
  - A frame, once rendered, can be held an arbitrary amount of time
  - Rasterization is optimized via damage maps
rendered mode data model—ncpiles (1/4)

- One or more piles in a rendered mode context, defined by:
  - Visual area (geometry can change at any time)
  - User doesn't work with piles explicitly
  - One or more planes
    - If all planes within a pile are destroyed/reparented, the pile is destroyed
  - Each pile has a z-axis (total order) and binding forest (DAF)
    - Subtree-wide moves, reparentings, destruction
    - Resize events cascade down to all children

- Piles (and all their planes) cannot be mutated while being rendered
- Only one thread may reorder/create/destroy planes within a pile at a time
- Multiple threads may mutate distinct piles concurrently
One or more planes per pile, defined by:
- A geometry, and an origin relative to the pile's visual area
- An active background color, foreground color, and style (used for output)
- A framebuffer of nccells and a backing EGCPool
- A user-managed opaque pointer, and a name (used for debugging)
- A resize callback function
- A virtual cursor location
- A base cell, used where cells are undefined
- Scrolling/z-axis/binding forest state

Planes are the fundamental drawing surface of Notcurses
Multiple threads may mutate multiple planes concurrently
A cell is a 16-byte structure, with possible spillover into an EGCPool.

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       gcluster (up to 4 UTF-8 bytes, OR a 24-bit index)       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    backstop   |     width     |           stylemask           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    foreground                                                 |
|                                                 background    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           channels                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           background                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
rendered mode data model—\texttt{nccells} (3b/4)

- EGCs of 4 or fewer UTF-8 bytes are inlined directly into \texttt{gcluster}
  - All currently-defined UCS characters are encoded in 4 or fewer UTF-8 bytes
- Larger EGCs are stored in the EGCPool
  - Indicated via initial byte of 0x01, followed by 24-bit offset
  - UTF-8 guarantees bytes < 128 are single ASCII characters
  - We don't allow control characters into a cell, thus 0x01 is unambiguous
- \texttt{backstop} is always 0, so that \texttt{gcluster} can be used as a C string
- \texttt{width} is the number of columns occupied by the EGC
  - Secondary cells of a multicolumn EGC have \texttt{width} \neq 0, \texttt{gcluster} == 0
- \texttt{stylemask} is a bitfield corresponding to italics, reverse video, blink etc.
rendered mode data model—channels (4/4)

- 64-bit structure composed of two 32-bit channels
- Each channel encodes either 24-bit RGB, 8-bit palette index, or default color
- 2 bits of "alpha" (OPAQUE, BLEND, TRANSPARENT, HIGHCONTRAST)
## Formatted Output, Parameterized by Type

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncplane_putc</td>
<td>Cell (including style/channels)</td>
<td>nccell</td>
</tr>
<tr>
<td>ncplane_putchar</td>
<td>7-bit ASCII as EGC</td>
<td>char</td>
</tr>
<tr>
<td>ncplane_putwc</td>
<td>Wide char as EGC</td>
<td>wchar_t</td>
</tr>
<tr>
<td>ncplane_putugc</td>
<td>UTF-8 EGC</td>
<td>const char*</td>
</tr>
<tr>
<td>ncplane_putwgc</td>
<td>Wide char EGC</td>
<td>const wchar_t*</td>
</tr>
<tr>
<td>ncplane_putstr</td>
<td>UTF-8 string</td>
<td>const char*</td>
</tr>
<tr>
<td>ncplane_putwstr</td>
<td>Wide string</td>
<td>const wchar_t*</td>
</tr>
<tr>
<td>ncplane_printf</td>
<td>Formatted output</td>
<td>const char*, ...</td>
</tr>
<tr>
<td>ncplane_vprintf</td>
<td>Formatted output</td>
<td>const char*, va_list</td>
</tr>
</tbody>
</table>
boxes and fills

- At their most generic, boxes are drawn:
  - At some starting location, with some geometry
  - With six specified nccell(s) (including style and channels)
  - With the ability to leave out arbitrary edges/corners
  - With the ability to interpolate between corner channels
  - Simple, double, and rounded prepared variants
  - ncplane_perimeter() prepared geometry
- ncplane_gradient() fills a rectangular area with a gradient and EGC
  - Interpolation from four sets of corner channels
- ncplane_polyfill() replaces a region of the same EGC with an nccell
multimedia

- currently supports FFmpeg and OpenImageIO backends
- the multimedia backend (chosen at compile time) is in libnotcurses
- linking against only libnotcurses-core requires no multimedia backend
- **ncvisual objects created with**
  - `ncvisual_from_file()`: open and decode arbitrary files
  - `ncvisual_from_rgba()`: loaded from decoded RGBA in memory
  - `ncvisual_from_plane()`: loaded from `ncplane` contents
- **even when linking against libnotcurses, check for runtime support**
  - `notcurses_canopen_images()` / `notcurses_canopen_videos()`
### image blitters

<table>
<thead>
<tr>
<th>NAME</th>
<th>GEOMETRY</th>
<th>ASPECT</th>
<th>FIDELITY</th>
<th>GLYPHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>1x1</td>
<td>2:1</td>
<td>100%</td>
<td>1</td>
</tr>
<tr>
<td>Half</td>
<td>2x1</td>
<td>1:1</td>
<td>100%</td>
<td>3</td>
</tr>
<tr>
<td>Quad</td>
<td>2x2</td>
<td>2:1</td>
<td>50%</td>
<td>15</td>
</tr>
<tr>
<td>Sex</td>
<td>3x2</td>
<td>1.5:1</td>
<td>33%</td>
<td>63</td>
</tr>
<tr>
<td>Braille</td>
<td>4x2</td>
<td>1:1</td>
<td>12.5%</td>
<td>255</td>
</tr>
</tbody>
</table>
## plot blitters

<table>
<thead>
<tr>
<th>NAME</th>
<th>GEOMETRY</th>
<th>GLYPHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space</td>
<td>1x1</td>
<td>1</td>
</tr>
<tr>
<td>Half</td>
<td>2x1</td>
<td>2</td>
</tr>
<tr>
<td>Fourths</td>
<td>4x1</td>
<td>4</td>
</tr>
<tr>
<td>Quad</td>
<td>2x2</td>
<td>8</td>
</tr>
<tr>
<td>Eighths</td>
<td>8x1</td>
<td>8</td>
</tr>
<tr>
<td>Sex</td>
<td>3x2</td>
<td>11</td>
</tr>
<tr>
<td>Braille</td>
<td>4x2</td>
<td>14</td>
</tr>
</tbody>
</table>
selector and multiselector
results (growlight disk manager)

<table>
<thead>
<tr>
<th>Growlight</th>
<th>Blockdevs</th>
<th>Partitions</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>sdm</td>
<td>STORAGE DEVICE</td>
<td>1203 0.00 S12B none n/a</td>
<td>PATA</td>
</tr>
<tr>
<td>sstj</td>
<td>STORAGE DEVICE</td>
<td>1203 0.00 S12B none n/a</td>
<td>PATA</td>
</tr>
<tr>
<td>sdp</td>
<td>STORAGE DEVICE</td>
<td>1203 0.00 S12B none n/a</td>
<td>PATA</td>
</tr>
<tr>
<td>[PCI Express 0000:46.00.3 (x16, gen 4.0)]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[nvme-2 [0] (160bps to chip, 320bps (20%) demanded)]

[solidstate]

[nvme2n1]

[nvmealn]

[mp3masos-0 (160bps to chip, 42.130bps (263%) demanded)]
results (omphalos network explorer)
thanks, FOSDEM 2021!

go watch the demo