Plan 9 in Technicolor

Russ Cox

Harvard College
Bell Labs, Lucent Technologies
rsc@plan9.bell-labs.com

August 23, 1999

Bitblt

Invented in 1975 at Xerox PARC.

Used on the Blit and in released Plan 9.

`bitblt(dst, r, src, sp, fn)`

`Src` is translated to align the point `sp` with upper left corner of rectangle `r`.

For each point in the rectangle,

`d ← fn(d, s).`

`fn` can be any of the 16 binary boolean operators.

`bitblt(dst, src, S)`
— copy src

`bitblt(dst, src, D^S)`
— XOR dst with src (cursor, text drawing, highlights).

`bitblt(dst, src, D|S)`
— overlay a line drawing onto a bitmap.

`bitblt(dst, src, D&S)`
— white out dst where source is zero (white).

Bitblt Examples

bitblt(hello world , [blank | black] , D^S) = hello world

bitblt(pjw , [face] , D|S) = [face] pjw

Bitblt Implementation

For the most part, just a loop over machine words, complicated by

- overlapping bitmaps
- edge conditions
- hardware limitations

On the Blit, an on-the-fly compiler generated optimal code at each bitblt call.

All geometric drawing operators (point, line, circle, polygon, etc.) eventually translate into a sequence of calls to bitblt.

Porter-Duff Algebra

Described in SIGGRAPH '84 proceedings.

In addition to red, green, blue, images get an alpha channel that specifies how much the image covers each pixel.

0 is not at all

1 is fully covered.

Pixels are stored with red, green, blue premultiplied by alpha.

To draw one image (s) onto another (d),

$$d \leftarrow d \times (1 - \alpha_s) + s$$

One image (s) in another (m) means

$$s \times \alpha_m$$

Think of ‘‘picture in window’’.

Inferno Draw

Designed with the Porter-Duff compositing algebra in mind.

`draw(dst, r, src, sp, mask, mp)`

> `Src` and `mask` are translated to align the points `sp` and `mp`, respectively, with upper left corner of rectangle `r`.
>
> For each point in the rectangle, the `src` pixel replaces the `dst` pixel if the `mask` pixel is non-zero (non-white).
>
> In terms of Porter-Duff algebra, this is a boolean (`src` in `mask`) over `dst`.

Lost `bitblt`'s XOR, but still have block copy, ‘‘or’’, and selective erasure.

XOR isn't very useful on color images anyway.

Makes images ‘‘write only’’: `bitblt`'s implicit ‘‘read-modify-write’’ loop becomes ‘‘read-replace-write’’ in `draw`.

Inferno Images

`Draw` uses *ldepth* to specify image format: $\log_2$ *depth*.

supports ldepth 0, 1, 2, 3.

Operands need not be of same ldepth.

To convert down, discard lower bits.

To convert up, replicate the bits.

For ldepth 0, 1, 2 (all greyscale), this just works.

For 8-bit color, the color map is designed to preserve up-replication.

Inferno Images

Image shape is defined by three elements:

r — the rectangle for which we have actual bits.

clipr — the bounding rectangle of the image.

repl — whether r's data replicates to fill clipr.

A picture: r = (0,0), (100,100), clipr = r, repl = 0.

A solid color: r = (0,0), (1,1),
clipr = ($-\infty$, $-\infty$), ($+\infty$, $+\infty$), repl = 1.

Wallpaper: r = (0,0), (50,50),
clipr = ($-\infty$, $-\infty$), ($+\infty$, $+\infty$), repl = 1.

A wallpapered box: r = (0,0), (50, 50),
clipr = (0,0), (100,100), repl=1.

Inferno draw implementation

Almost a loop over machine words once everything is same depth, complicated by

- Overlapping bitmaps
- Edge conditions
- Hardware limitations

Converting all data to the same depth before processing makes things fast enough.

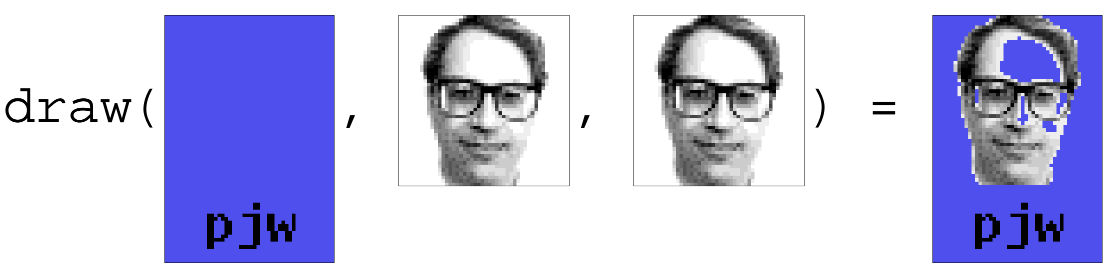
All geometric drawing operators (point, line, circle, polygon, etc.) eventually translate into a sequence of calls to draw.

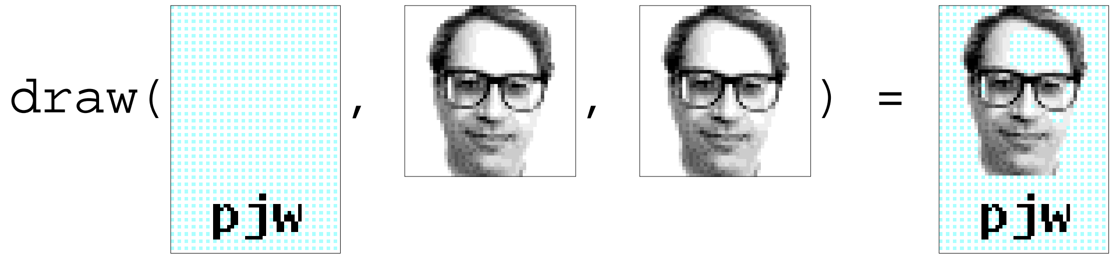
Inferno Draw Examples

draw(pjw , ,) = pjw

draw(, ,) =

Inferno Draw Examples (cont.)





True Color Draw

`draw(dst, r, src, sp, mask, mp)`

Perform the usual translation.

Replace `dst` with (`src` in `mask`) over `dst`, but this time with full due paid to alpha.

(Inferno draw was just a boolean alpha.)

True Color Images

Same rectangle descriptions as before, but can’t use ldepth anymore.

Decided it was impossible to unambiguously convey pixel format with depth.

Now use strings describing pixels: ‘‘r5g6b5’’ means 5 bits of red, then 6 bits of green, then 5 bits of blue.

- Also have ‘‘a’’ (alpha), ‘‘k’’ (grey), and ‘‘x’’ (don’t care).
- Depth is sum of numbers, and must be a divisor or multiple of 8.
- No channel can be more than 8 bits.

True Color Images (cont.)

Pixels are arranged in big endian order within bytes: `11223344`.

Bytes are arranged in little endian order within pixels: r5g6b5 is `gggbbbbb rrrrrggg`.

It's just what VGAs do.

Strings are the external representation: inside the library we use an encoding of the string as a ulong (unsigned 32-bit int) called a channel descriptor. `Chantostr` and `strtochan` convert between ulongs and strings.

True Colors

Bitblt and Inferno draw use all zeros as white and all ones as black.
(They're drawing with ink.)

RGB uses all zeros as black and all ones as white (drawing with light).

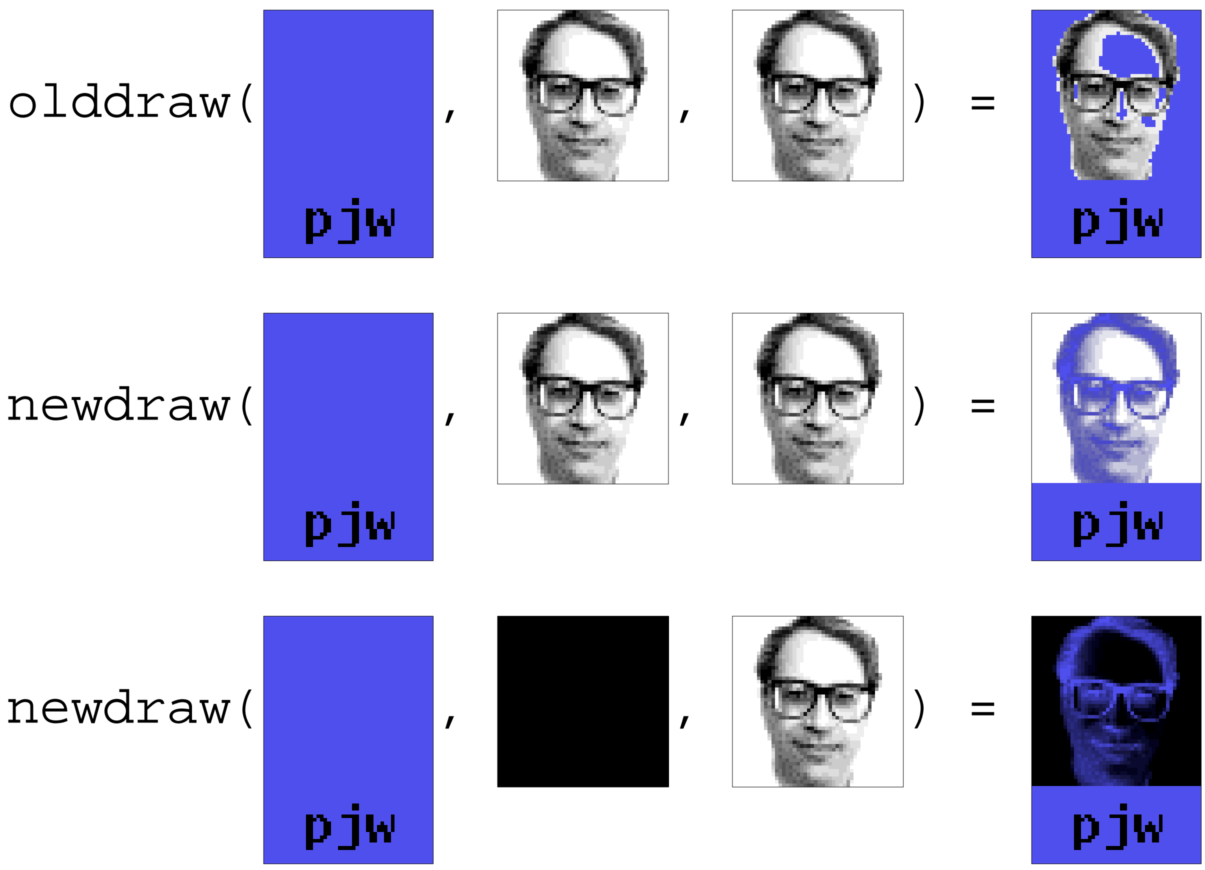
We must do what the VGAs do for RGB.

For consistency, even $\leq$ 8-bit images now use zeros as black and ones as white.

So now black and white have switched roles as masks.

True Colors (cont.)

So we lose the OR behavior for black-on-white bitmaps:



True Colors (cont.)

In the new draw, we can recreate it by stenciling.

First make a negative from the image:

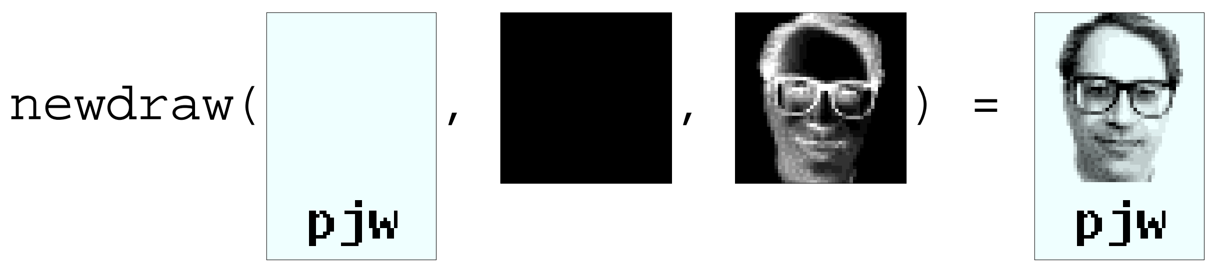
Start with all ones (white).

Where there are ones in the image, draw zeros (black).

newdraw(, ,) =

Now draw a solid color somewhere using the stencil as the mask.

With an appropriate background, it looks fine:



True Color Draw Implementation

Loop over pixels (not machine words), that must

Convert `src`, `dst` to 32-bit RGBA.

If mask has alpha channel use it as the integer mask value. Otherwise use the greyscale equivalent of the mask color as the mask value.

`dst` ← (`src` in `mask`) over `dst`.

Convert `dst` from RGBA back to pixel format.

Conversion is done by selecting the appropriate converter at the beginning of the loop and calling through a function pointer.

To reduce call overhead, convert a scan line at a time.

Special cases (move, copy, fill, boolean mask), are handled separately for speed.

When drawing on screen, hardware picks off what it can handle (move, copy, fill, boolean mask).

All geometric drawing operators (point, line, circle, polygon, etc.) eventually translate into a sequence of calls to draw.

Library Changes

Ldepth is gone, replaced by channel descriptors.

Colors are always passed around as 32-bit RGBA numbers rather than color map indices.

To avoid (or at least minimize) confusion, `display->ones` and `display->zeros` are replaced with `->black`, `->white`, `->transparent`, and `->opaque`.

Bitmap format changes

Bitmaps are as before, except:

ldepth is replaced by the channel descriptor string.

Relative to old bitmaps, every bit is inverted.

`fb/pcp` doesn't know about the new format. However, `fb/dump2bit` and `fb/bit2dump` convert between new bitmaps and images of type `dump` (which `pcp` does know about).

Compression, as before, is via the modified LZ77 (Storer and Szymanski).

Bitmap format changes

Old bitmaps are grandfathered in: `readimage` and `creadimage` still know how to sniff the headers and perform the correct transformation on the bits.

Applications that generate bitmaps (rio, page, etc.) now generate the new ones.

`p9bitpost`, which `lp` uses to convert Plan 9 bitmaps into Postscript, knows about the new format and will generate true color Postscript when handed true color data.

So `lp -dbinney /dev/screen` will produce meaningful results.

Benefits

We have grey fonts again, simplifying things like proof.

We can manipulate true color frame buffers.

So we can use MPEG cards that write to true color frame buffer memory.

2×2 pastel textures are gone in true color mode, replaced with solid colors.

Hardware acceleration gets used.

`cat /lib/words` in a full screen (1800x1350) window takes 15 seconds in the old draw on Rob's office PC.

It takes 2 seconds in the new draw because of hardware fills. Implementing character drawing would make it even faster.

Drawbacks

Every graphics app needs rewriting.

In `/sys/src`, everything is converted except some games.

Adding support for a VGA card is even more work now if you want true color on that card.

Drawing is still done by the kernel:

Image memory cannot be swapped.

Kernel binaries are slightly bigger (about 50k).

Future Work

VGAs at high resolutions and 16, 24, 32 bit depths hate CPU writes to video memory.

- It usually results in tearing and flickering during updates.

For each draw operation on VGA memory that cannot be handled by sending an RPC with the card, we should:

- Copy the relevant data from the VGA to its backing store.
- Perform the draw operation on the backing store.
- Write the data back using a block copy RPC.
- Since the controller is touching the memory, it can take care of avoiding tearing and flickering.

Future Work II

It’s easy to write a simple, correct draw implementation (275 lines including comments).

```
uchar m, M, sr, sg, sb, sa, sk,
    dr, dg, db, da, dk;

for each point
    d[rgba] = getpixel(dst, dp);
    s[rgba] = getpixel(src, sp);
    m = getmask(mask, mp);
    M = 255-(sa*m)/255;

    for c in r,g,b,a
        d[c] = (s[c]*m+d[c]*M)/255;

    putpixel(dst, dp, dr, dg, db, da);
```

Future Work II (cont.)

It’s hard to write a simple, correct, and fast draw implementation.

Effectively, the draw implementation is the above with enough conditionals and function calls pushed outside enough loops to make the overhead bearable.

I’d like to see a (non-optimal) on-the-fly compiler.

Interesting more for simplicity of implementation than speed, although should be faster in general case.

Probably still want to pick off special cases, but perhaps fewer.

Compiling is a definite loss for machines on which we need to flush the instruction cache (MIPS).

The Pentium and later Intel processors promise to keep instruction and data caches consistent.

Don’t know about Alpha, Power PC, but as VMs become more common, this should be the norm.

The Big Switch

Currently scheduled for early October, when everything has had time to soak.

Almost all applications in `/sys/src` are now converted; old draw applications are deprecated and won't be updated.

You can (and are encouraged to) run the new programs:

Put ``setupdraw'' in your Plan 9 profile before the `switch($service)` line. (A no-op if running an old kernel.)

Boot `nkernel` instead of `kernel` (`/mips/nbcarrera`, `/386/nbpc`, `/386/nbpcdisk`).

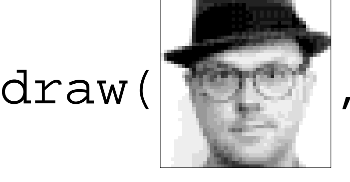
Or use `/usr/rsc/nvga/ndrawterm.exe` instead of `drawterm`.

To convert your own programs,

see `/usr/rsc/nvga/convert.ps`.
updated man pages in `/usr/rsc/nvga/man`.

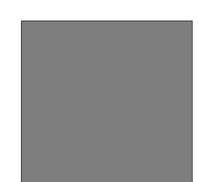
There will be news explaining this.

True color draw as simple morpher



,

,



) =