What's new in Psychtoolbox-3?

A free cross-platform toolkit for Psychophysics with Matlab & GNU/Octave

Mario Kleiner, David Brainard, Denis Pelli

Allen Ingling, Richard Murray, Christopher Broussard,
Frans Cornelissen & many others...

Cognitive & Computational Psychophysics
Max Planck Institute for Biological Cybernetics

Foreword ;-)

This PDF document contains the “cleaned” slides of a public presentation about Psychtoolbox-3, presented at the 30th European Conference on Visual Perception (ECVP 2007) in Arezzo, Italy.

Some (not too important) illustrations had to be removed, because we don't have written permission for distribution by the owners / copyright holders of that illustrations.

This set of slides also contains a few slides not shown in the presentation, so the order of slides may look a bit scrambled sometimes, sorry...

I tried my best to make this presentation accurate, but of course, we can not be held liable for any factual errors, omissions or inaccuracies and the possible damage done to your work. We are grateful for corrections if you find any errors.

Mario Kleiner

Trademarks...

- OpenGL and the OpenGL logo are registered trademarks of Silicon Graphics Inc.
- Quicktime and the Quicktime logo are registered trademarks of Apple Inc.
- ASIO and the ASIO logo are registered trademarks of Steinberg Media Technologies GmbH.
- Matlab and the Matlab logo are registered trademarks of The Mathworks Inc.
- All other trademarks and copyrights are property of their respective owners.

Motivation

- Requirements for typical experimental studies in cognitive science:
  - Accurate and flexible control of stimulus properties.
  - Robust timing for stimulus presentation and response collection.
- Low-level programming languages and API's:
  - Very flexible.
  - Difficult and time consuming to learn and use.
- High-level canned experiment packages:
  - Easy to learn & use.
  - Considerable lack of flexibility.
- Middle ground: Interpreted languages like Matlab or Python:
  - Easy to learn *and* flexible.
  - Insufficient level of control, timing precision and performance for psychophysics out of the box.

⇒ Psychtoolbox: Extensions for Matlab with functions for precise stimulus presentation and response collection.

Psychtoolbox-3

- A set of Matlab extensions to facilitate programming of psychophysics experiments in Matlab.
  - Compiled Plugins (written in C) for time critical operations and interfacing with graphics-, sound- and response collection hardware.
  - A couple of hundred Matlab M-Files to simplify higher level tasks like data analysis, logging of responses to files, monitor calibration, ...
- Free software, licensed under GNU General Public License (GPL-2)
- Community project: Please contribute!
- Available for Apple MacOS/X, Microsoft Windows (and GNU/Linux).
- Tested and developed with Matlab 7. Older Matlabs may mostly work.
- Some support for GNU/Octave 2.1.73 on Intel based Macintosh and GNU/Linux.

History

- Version 1: Released 1995 for Macintosh. Written by David Brainard.
- Version 2:
  - Released 1996 for MacOS by David Brainard and Denis Pelli.
  - Released 2000 for MS-Windows by Allen Ingling and others.
  - Hybrid of Pelli's popular VideoToolbox and Brainard's PTB-1.

⇒ 24,324 Downloads for Windows, 8,743 for MacOS. Popular :-)

- Release of Apple MacOS/X and doom of OS-9 causes problems:
  - C-Plugins not portable: Old MacOS9 code not well suited for MacOS/X.
  - C source code not modular: Difficult for new developers to contribute.
  - Doesn't take any advantage of modern graphics hardware.

⇒ Create PTB−3 as an improved redesign to solve those problems.

Psychtoolbox-3

- Released 2004 for MacOS/X by Allen Ingling, 2006 for Windows & Linux:
  - M-Functions (mostly) adapted from Version 2.
  - C-Plugins rewritten from scratch for increased modularity, better maintainability and portability. Uses platform independent API's and open standards / open source toolkits like OpenGL wherever possible.
  - Tries to take advantage of modern graphics- and sound hardware.
  - New license: GPL to facilitate more 3rd party code contributions, provide users with some well defined license, allow code-reuse and integration of other 3rd party toolkits.
  - Subversion as code management system:
    - All changes are archived and documented.
    - Simplifies collaborative development.
    - Provides convenient software update mechanism to users.
    - One new release about every 2 weeks, bugfixes sometimes faster.
- Approx. 3600 known installations (≈2300 Windows, ≈1300 OS/X)

Outline

- Visual stimulus presentation with "Screen": What's new?
    - Stimulus presentation: Multi-display, Stereo, Double buffering, Accurate onset timing, Timestamping, Standard paradigms...
    - Fast 2D stimulus creation: Batch drawing, Texture mapping, Alpha blending, Movie playback, Video capture, Image processing...
    - Automatic stimulus postprocessing: HDR, Calibration, ...
    - Fast 3D stimulus creation: OpenGL for Matlab
- Sound output: Low-latency, high precision Sound (and 3D Sound)
- Basic functions for timing and response collection.
- What else is in the box?
- Outlook & More Questions?

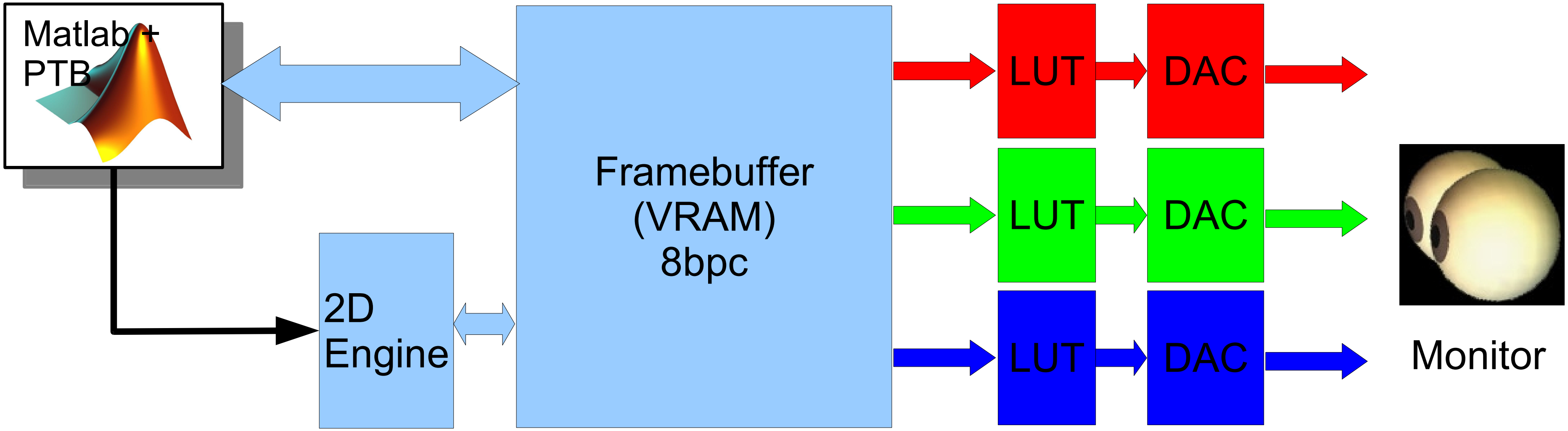
The Visuals: Screen()

- Controls all aspects of the graphics- and display-hardware.
- Performs all 2D drawing operations.
- Controls stimulus onset timing and provides timestamps.
- Allows for some high performance image processing.
- Performs on-demand stimulus post-processing.

⇒ This part has evolved most, compared to PTB-2.

⇒ Based on OpenGL, uses latest features of OpenGL-2 hardware if available.

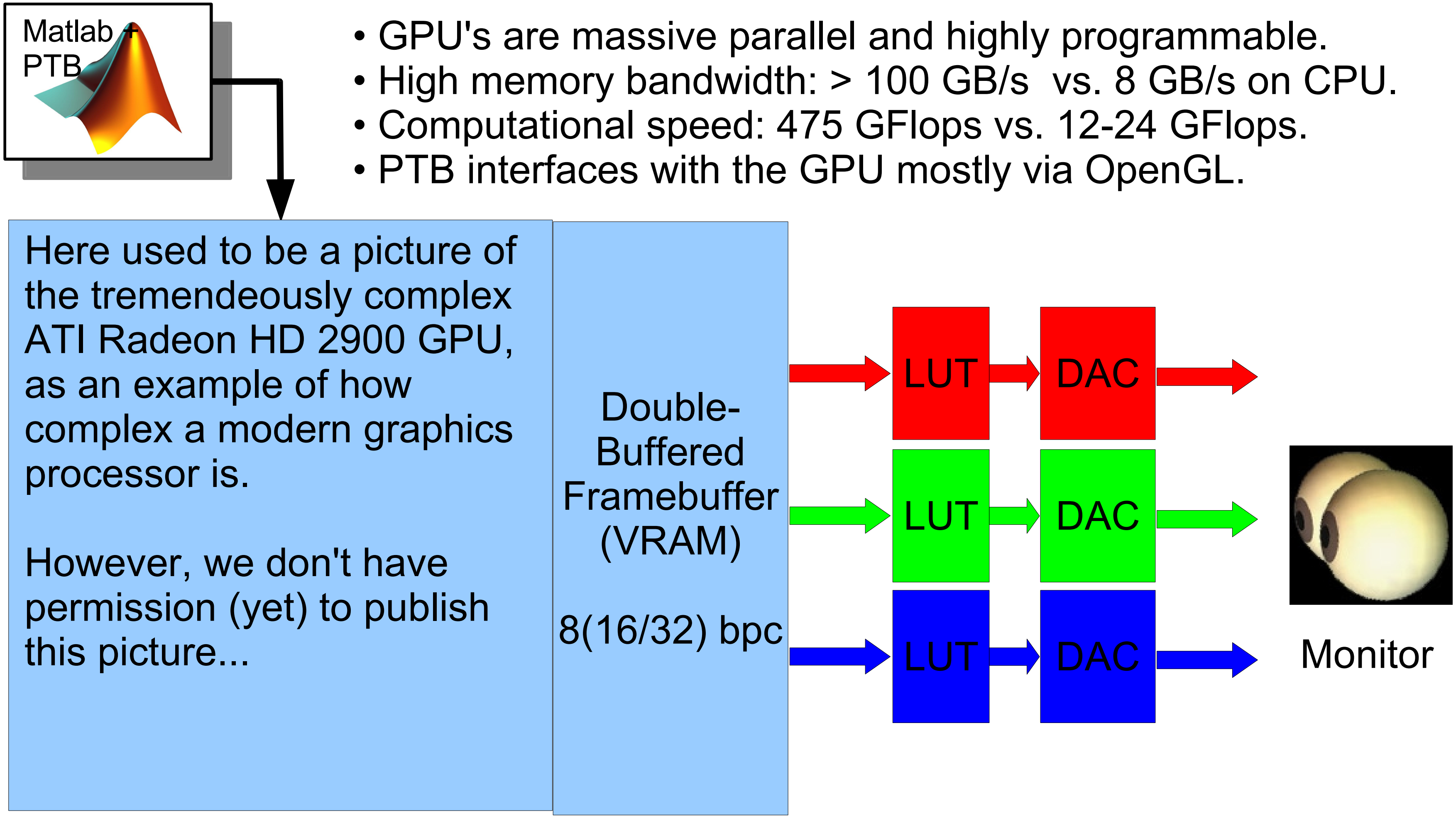
Graphics hardware anno domini 1995
The PTB-1/2 single-buffered drawing model



- Graphics card == Mostly passive image store.
- Psychtoolbox draws directly into Framebuffer while buffer is scanned out for display.
- Most drawing operations implemented/executed in software on slow CPU.
- Only very few operations are hardware-accelerated (Image copy, filled rectangles).

Graphics hardware today:

- GPU's are massive parallel and highly programmable.
- High memory bandwidth: > 100 GB/s vs. 8 GB/s on CPU.
- Computational speed: 475 GFlops vs. 12-24 GFlops.
- PTB interfaces with the GPU mostly via OpenGL.



Here used to be a picture of the tremendeously complex ATI Radeon HD 2900 GPU, as an example of how complex a modern graphics processor is.

However, we don't have permission (yet) to publish this picture...

Example: ATI Radeon X2900 GPU schematic (From www.xbitlabs.com)

Graphics hardware requirements:

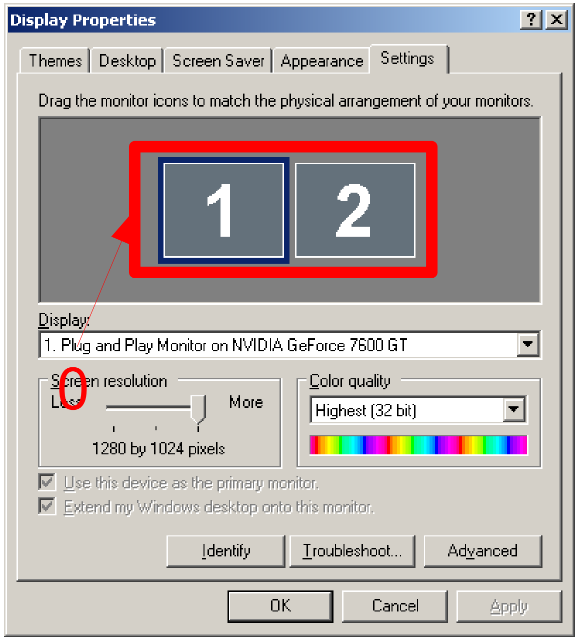
- Basic functionality available on any graphics hardware:
  - Basic OpenGL 1 support required.
  - 16 MB VRAM required.
- High speed and advanced features need recent graphics hardware:
  - OpenGL 1.5, better 2.0/2.1 capable.
  - > 128 MB VRAM, more is better.
  - Simplifies a lot of complex stimulus programming tasks.
  - Allows for high precision, e.g., 32 bpc floating point.
- PTB-3 regularly tested on: NVidia GeForce 7800, ATI Radeon X1600, GeForceFX-5200.
- If you can choose, go for a:
  - NVidia GeForce-8 series: GF-8600 or better...
  - ATI X-2000 series: X-2600 or better...
- Onboard chips are very limited in functionality and speed! (e.g. Intel GMA-950 of MacBooks)

Support for Multidisplay setups:

% Enumerate all connected displays:

myscreens = Screen('Screens');

- Multi-display setups now supported on all operating systems.
- Difference in display enumeration between operating systems:
    - OS/X: 0 = Main display (or internal laptop flat-panel), 1 = 2nd display (or external display on laptop), 2 = 3rd display...
    - Windows: 0 = "Virtual display" spanning the main display and its right neighbour. 1 = Real main display, 2 = Real display 2, ...


Display Properties

Drag the monitor icons to match the physical arrangement of your monitors.

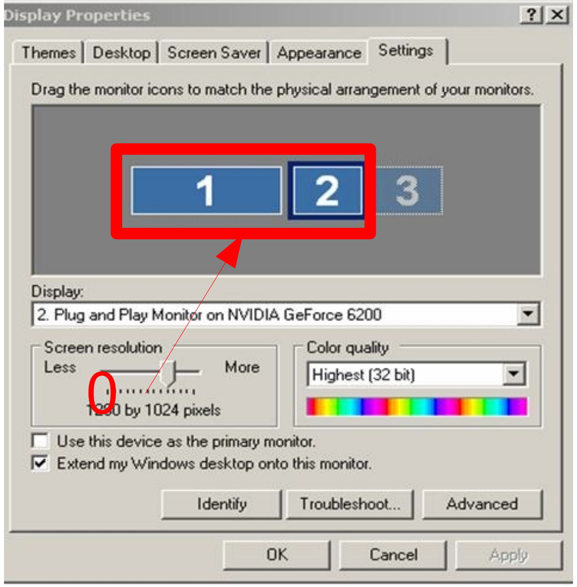
Display: 1. Plug and Play Monitor on NVIDIA GeForce 7600 GT

Screen resolution: 1280 by 1024 pixels

Color quality: Highest (32 bit)

☑ Use this device as the primary monitor.

☑ Extend my Windows desktop onto this monitor.


Display Properties

Drag the monitor icons to match the physical arrangement of your monitors.

Display: 2. Plug and Play Monitor on NVIDIA GeForce 6200

Screen resolution: 1280 by 1024 pixels

Color quality: Highest (32 bit)

☐ Use this device as the primary monitor.

☑ Extend my Windows desktop onto this monitor.

Support for Multidisplay setups:

- Only use multi-display if you really need it!
- Use 1 dual-head graphics card for binocular stimulation, not 2 single-head cards!
- If you need a separate “operator” display (Matlab window etc.), attach it to a separate graphics card!
- Buy *fast* graphics card with *generous amount* of VRAM.
- For binocular/stereo stimulation: Choose same resolution, color depth and monitor refresh rate for both physical displays!
- Interesting option for Laptops: Use a display-splitter to split the external display output into two separate displays.

How to choose the stimulation display?

```
% Open full-screen stimulus presentation window on display 'screenid':
mywindow = Screen('OpenWindow', screenid);
```

Stereo display support for easy binocular stimulation:

- One optional flag in *Screen('OpenWindow', ..., stereoMode);* to select between different modes:
  - Monoscopic.
  - Quad-buffered frame sequential stereo for shutter glasses.
  - Dual-display stereo for stereo goggles, stereoscopes, polarized stereo...
  - Anaglyph Red/Green and Red/Blue stereo with controllable gains and mode of presentation: *SetAnaglyphStereoParameters(...);*
- One command in your stimulus drawing loop to switch between drawing of left- (*bufferid=0*) and right-eye (*bufferid=1*) view:

  Screen('SelectStereoDrawBuffer', myWindow, bufferid);

- Demos: *StereoDemo, ImagingStereoDemo*

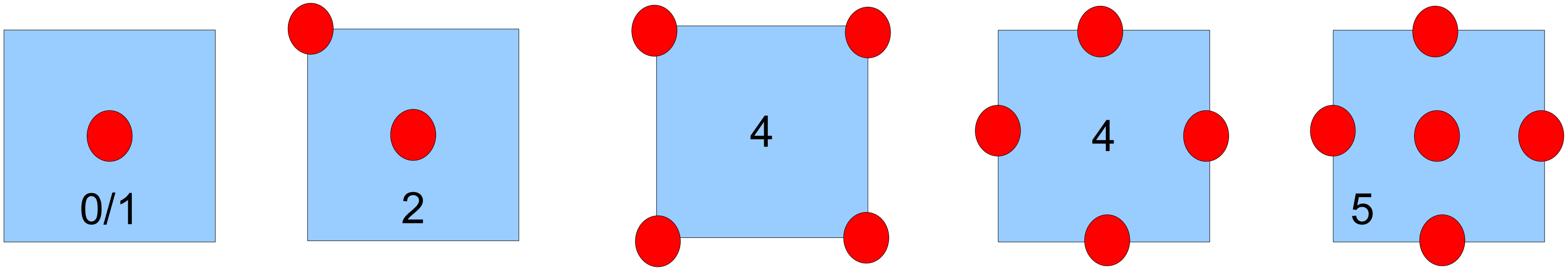
Stereo display for dual-display setups:

- Slightly different approach on Windows vs. MacOS/X:
  - Windows:
    ```
    % Open on screen 0, with stereomode 4:
    win = Screen('OpenWindow', 0, ...., 4);
    ```
  - MacOS/X:
    ```
    % Open left-eye window first on 'lefteyedisplay', with stereomode 10:
    win = Screen('OpenWindow', lefteyedisplay, ...., 10);
    % Open right-eye window 'righteyedisplay', with stereomode 10.
    % Don't care about window handle, this window is just a passive
    % receiver:
    Screen('OpenWindow', righteyedisplay, ...., 10);
    ```

    Use 'win' windowhandle for all further commands...
- Demos: *StereoDemo*, *ImagingStereoDemo*

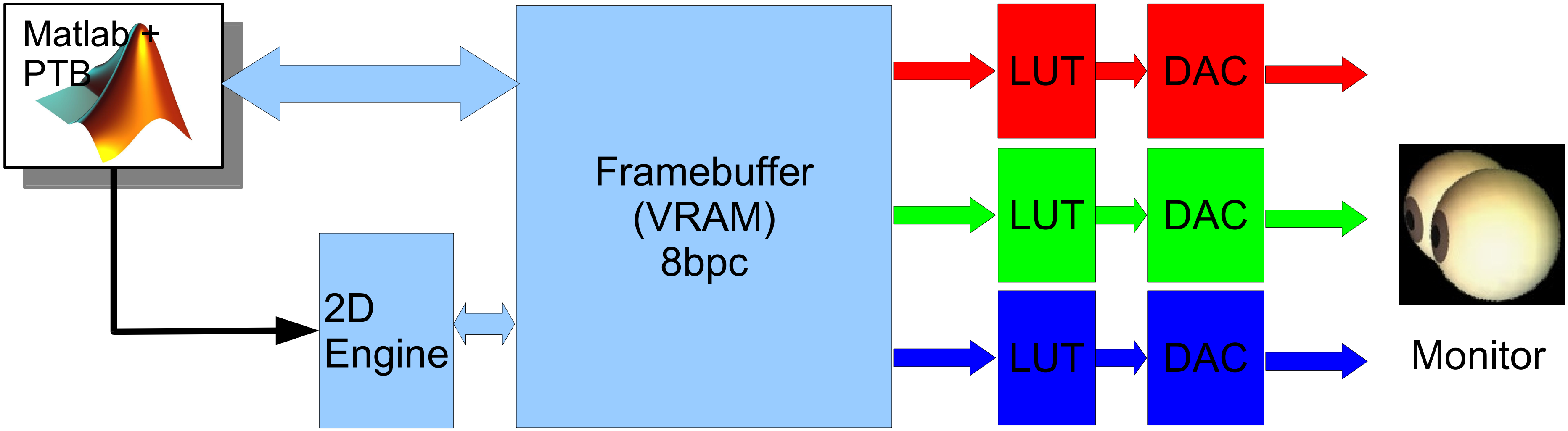
Multisampling for full scene Anti-Aliasing:

- Modern hardware can compute multiple color samples per output image pixel and resolve them into weighted average color.
- Provides anti-aliasing for images.
- Expensive in terms of memory usage and computation time!
- Implementation differs between graphics hardware models!
- *% Open window, request 'multiSample' samples per pixel:*
  win = Screen('OpenWindow', ..., multiSample);



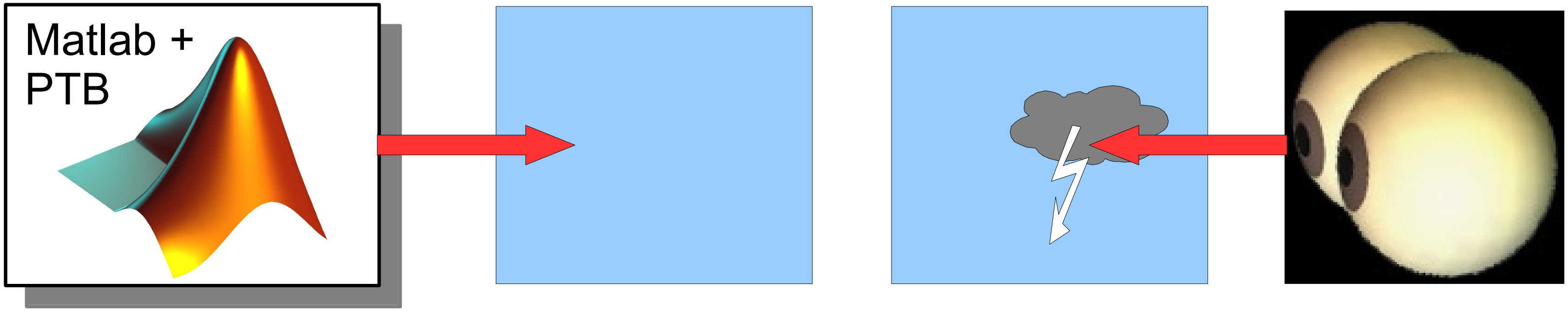
The PTB-1/2 single-buffered drawing model:

Drawing by PTB and scanout by display engine happen simultaneously on same framebuffer



- Problem: Need to synchronize drawing with vertical refresh of display. *Screen('WaitBlanking', window);*
- If drawing is too slow to stay ahead of display scanning beam, tearing will occur!
- Limits duration (==amount/complexity) of drawing operations to fractions of a video refresh duration!

PTB-3: Double buffered drawing model - Concept



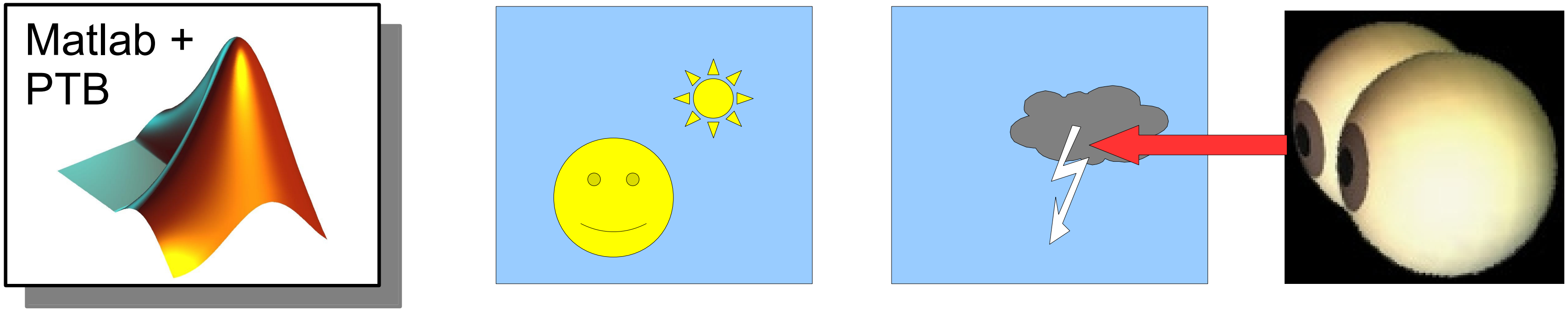
Matlab +
PTB

Invisible
Backbuffer
(VRAM)

User-Visible
Frontbuffer
(Display)

1) Subject sees cloudy picture on the display screen (==Frontbuffer).
2) Matlab issues Screen() drawing commands to Psychtoolbox, Psychtoolbox translates these commands into OpenGL rendering commands and submits them to the graphics accelerator hardware. Alternatively, one can submit low level OpenGL 3D drawing calls calls directly via the OpenGL for Matlab interface (see later), or via some compiled OpenGL C-Plugin.

PTB-3: Double buffered drawing model - Concept



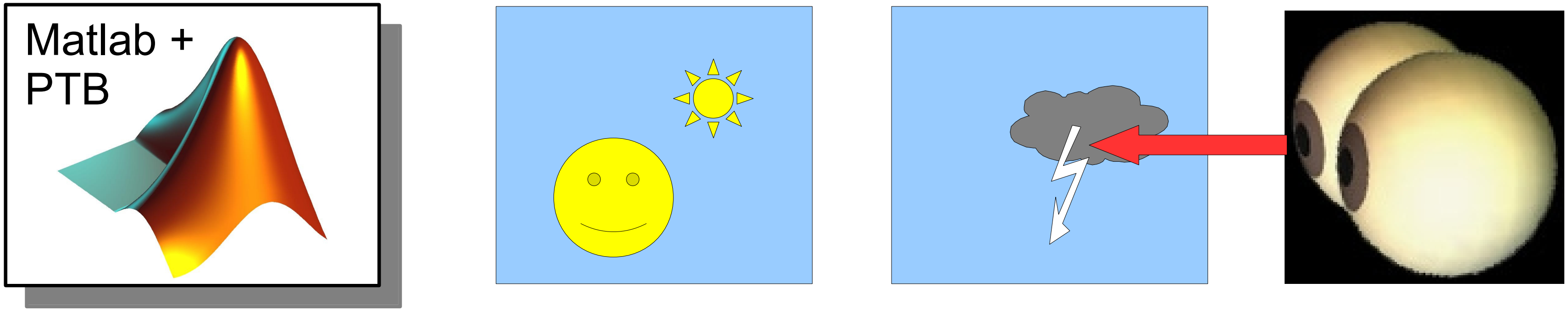
Matlab +
PTB

Invisible
Backbuffer
(VRAM)

User-Visible
Frontbuffer
(Display)

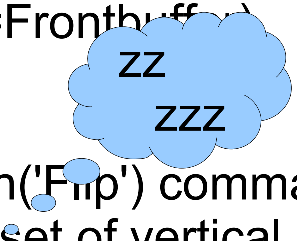
1) Subject sees cloudy picture on the display screen (==Frontbuffer).
2) Matlab issues Screen() drawing commands.
3) Graphics hardware draws into backbuffer, processing the OpenGL commands in the background while Matlab and Psychtoolbox are able to do other stuff in parallel, e.g., keyboard queries, sound output...

PTB-3: Double buffered drawing model - Concept



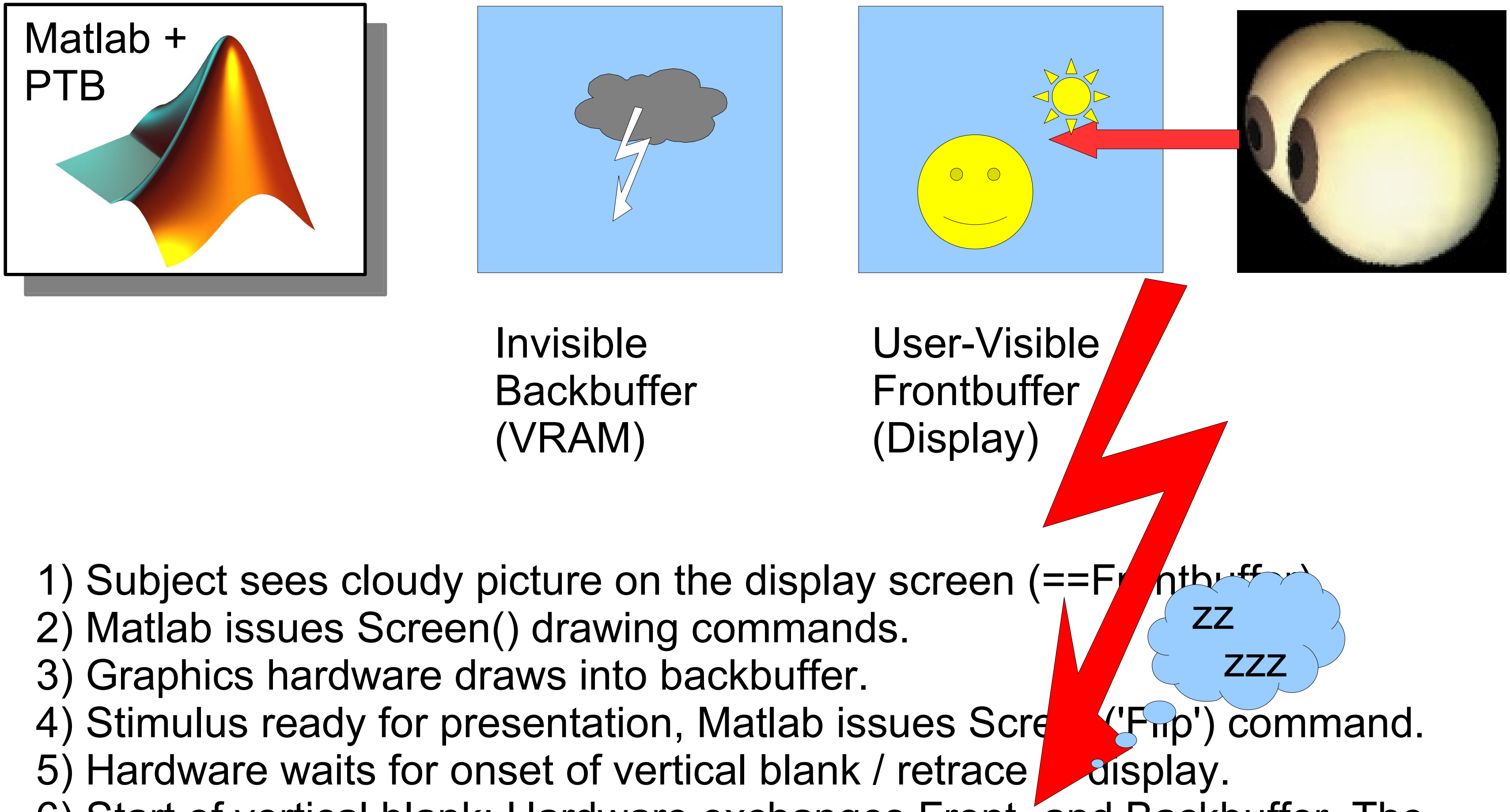
Invisible
Backbuffer
(VRAM)

User-Visible
Frontbuffer
(Display)



1) Subject sees cloudy picture on the display screen (==Frontbuffer)
2) Matlab issues Screen() drawing commands.
3) Graphics hardware draws into backbuffer.
4) Stimulus ready for presentation, Matlab issues Screen('Flip') command.
5) Matlab goes to sleep. Graphics hardware waits for onset of vertical blank / retrace of display. Waiting happens via some built-in low-level trigger circuitry in the hardware, unaffected by any timing noise on the main processor.

PTB-3: Double buffered drawing model - Concept



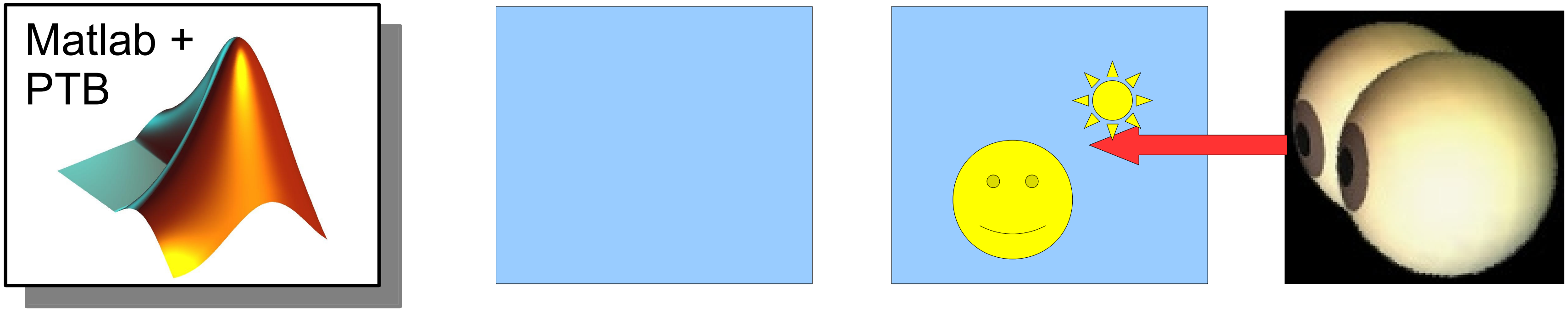
Matlab +
PTB

Invisible
Backbuffer
(VRAM)

User-Visible
Frontbuffer
(Display)

1) Subject sees cloudy picture on the display screen (==Frontbuffer).
2) Matlab issues Screen() drawing commands.
3) Graphics hardware draws into backbuffer.
4) Stimulus ready for presentation, Matlab issues Screen('Flip') command.
5) Hardware waits for onset of vertical blank / retrace of display.
6) Start of vertical blank: Hardware exchanges Front- and Backbuffer. The low-level circuitry of the graphics hardware ensures microsecond accurate sync of bufferswap with the vertical blank! Matlab gets woken up...

PTB-3: Double buffered drawing model - Concept



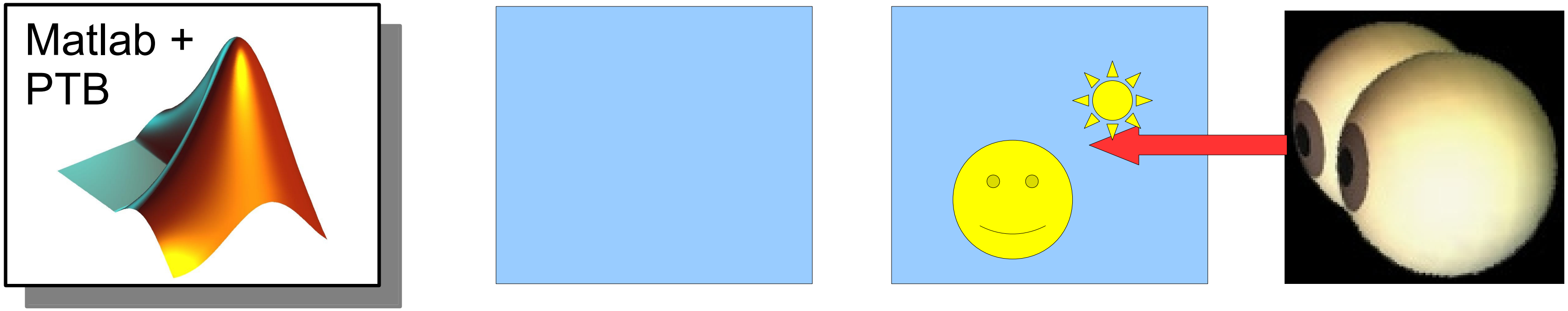
Matlab +
PTB

Invisible
Backbuffer
(VRAM)

User-Visible
Frontbuffer
(Display)

1) Subject sees cloudy picture on the display screen (==Frontbuffer).
2) Matlab issues Screen() drawing commands.
3) Graphics hardware draws into backbuffer.
4) Stimulus ready for presentation, Matlab issues Screen('Flip') command.
5) Hardware waits for onset of vertical blank / retrace of display.
6) Start of vertical blank: Hardware exchanges Front- and Backbuffer.
7) Subject perceives a tear-free stimulus update, PTB takes stimulus onset timestamp and clears the backbuffer for next stimulus drawing cycle.

PTB-3: Double buffered drawing model - Concept



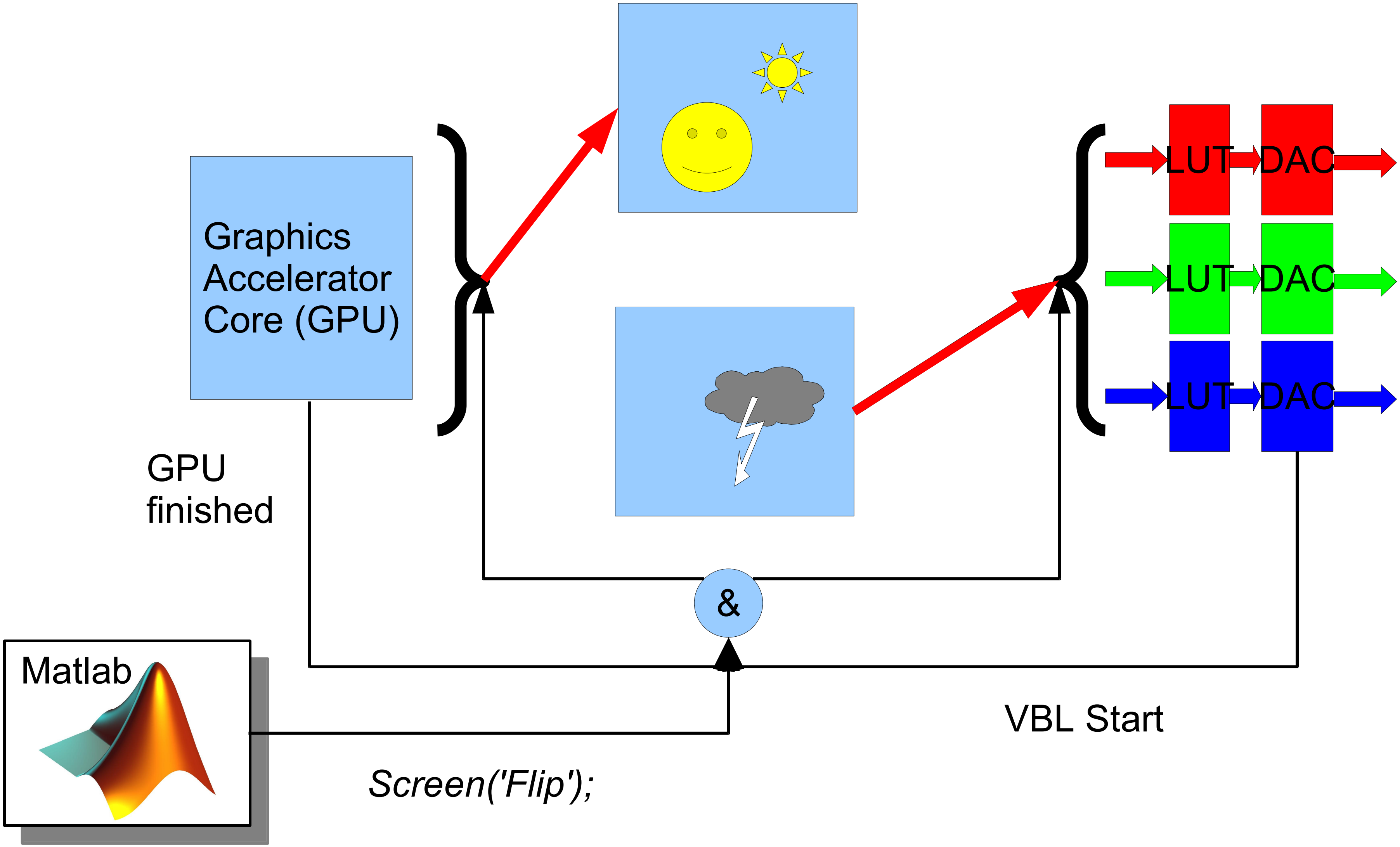
Matlab +
PTB

Invisible
Backbuffer
(VRAM)

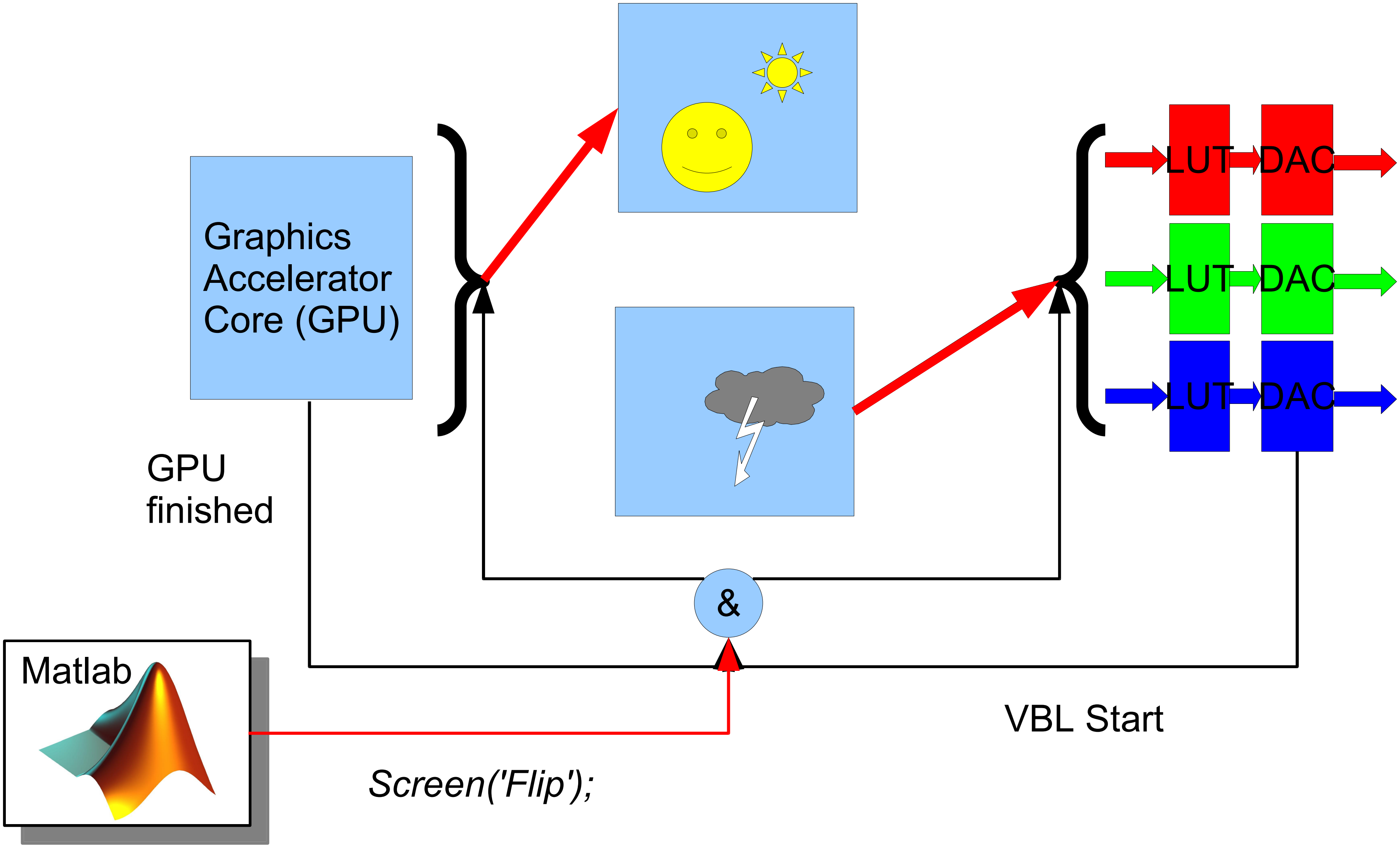
User-Visible
Frontbuffer
(Display)

1) Subject sees new sunny picture on the display screen (==Frontbuffer).
2) Backbuffer is cleared to background color for next stimulus.
3) Ready for next display redraw cycle...

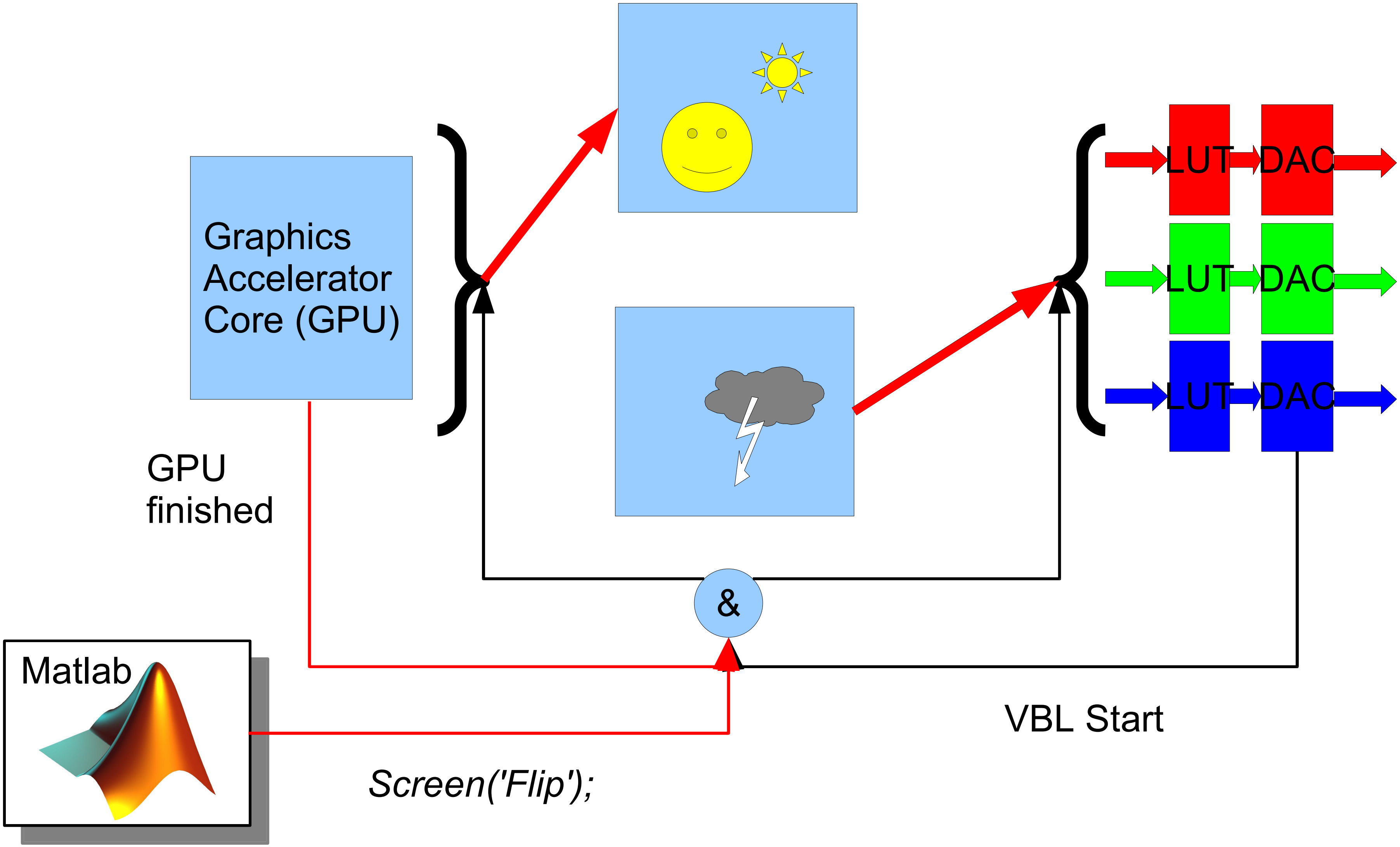
Double buffered drawing model - Implementation:



Double buffered drawing model - Implementation:



Double buffered drawing model - Implementation:



Graphics Accelerator Core (GPU)

GPU finished

Matlab

Screen('Flip');

&

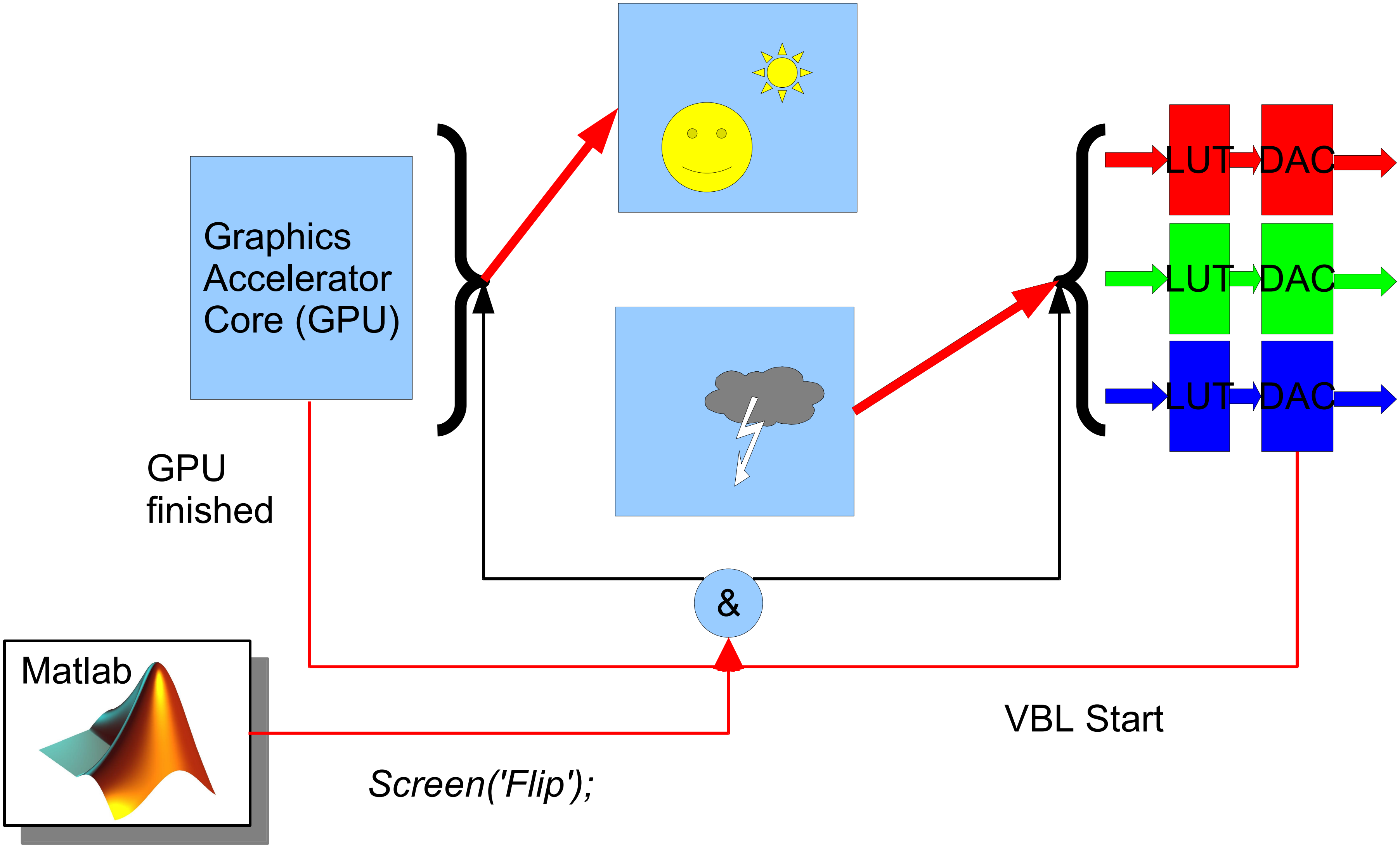
VBL Start

LUT DAC

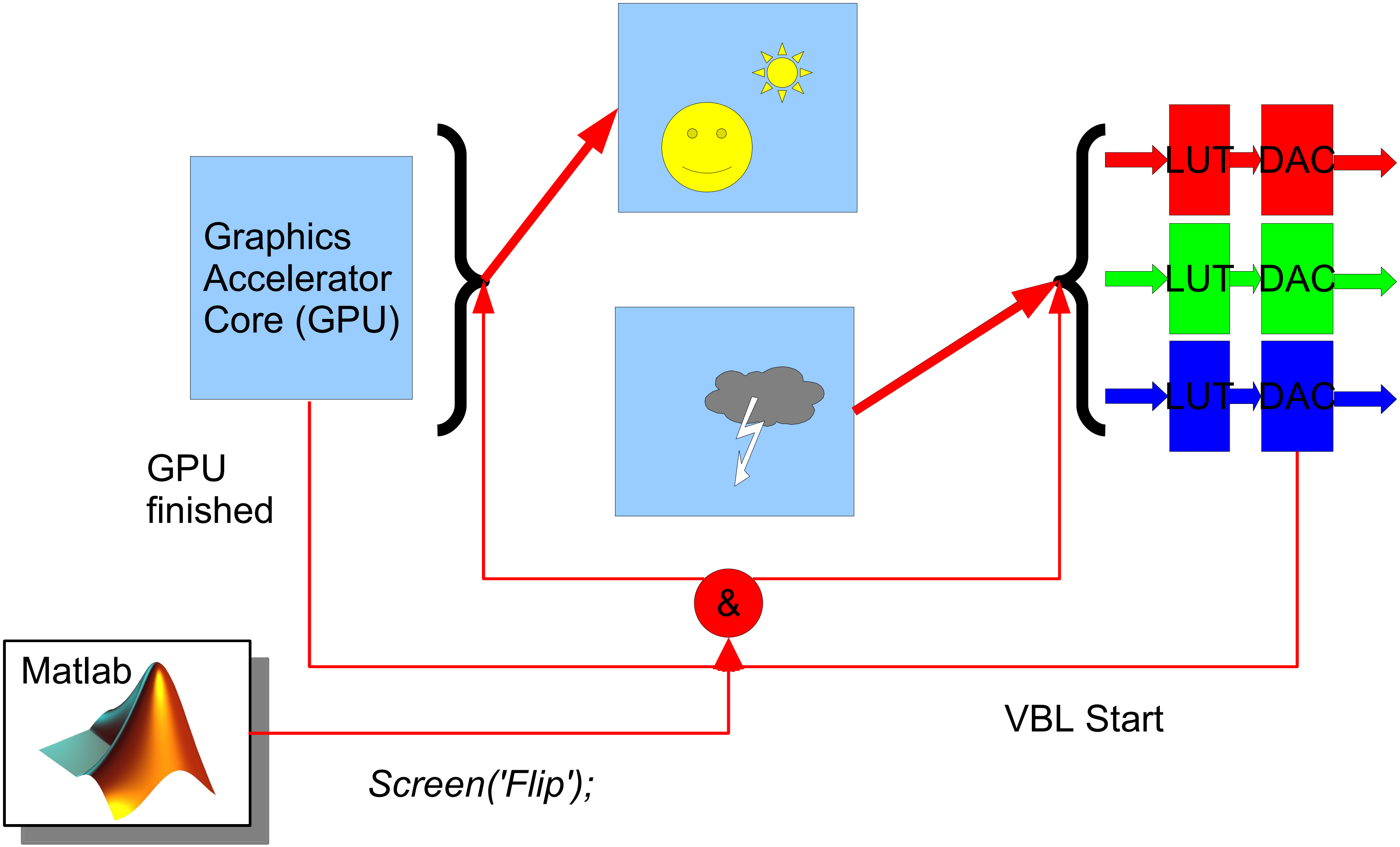
LUT DAC

LUT DAC

Double buffered drawing model - Implementation:



Double buffered drawing model - Implementation:



Graphics Accelerator Core (GPU)

GPU finished

LUT

DAC

LUT

DAC

LUT

DAC

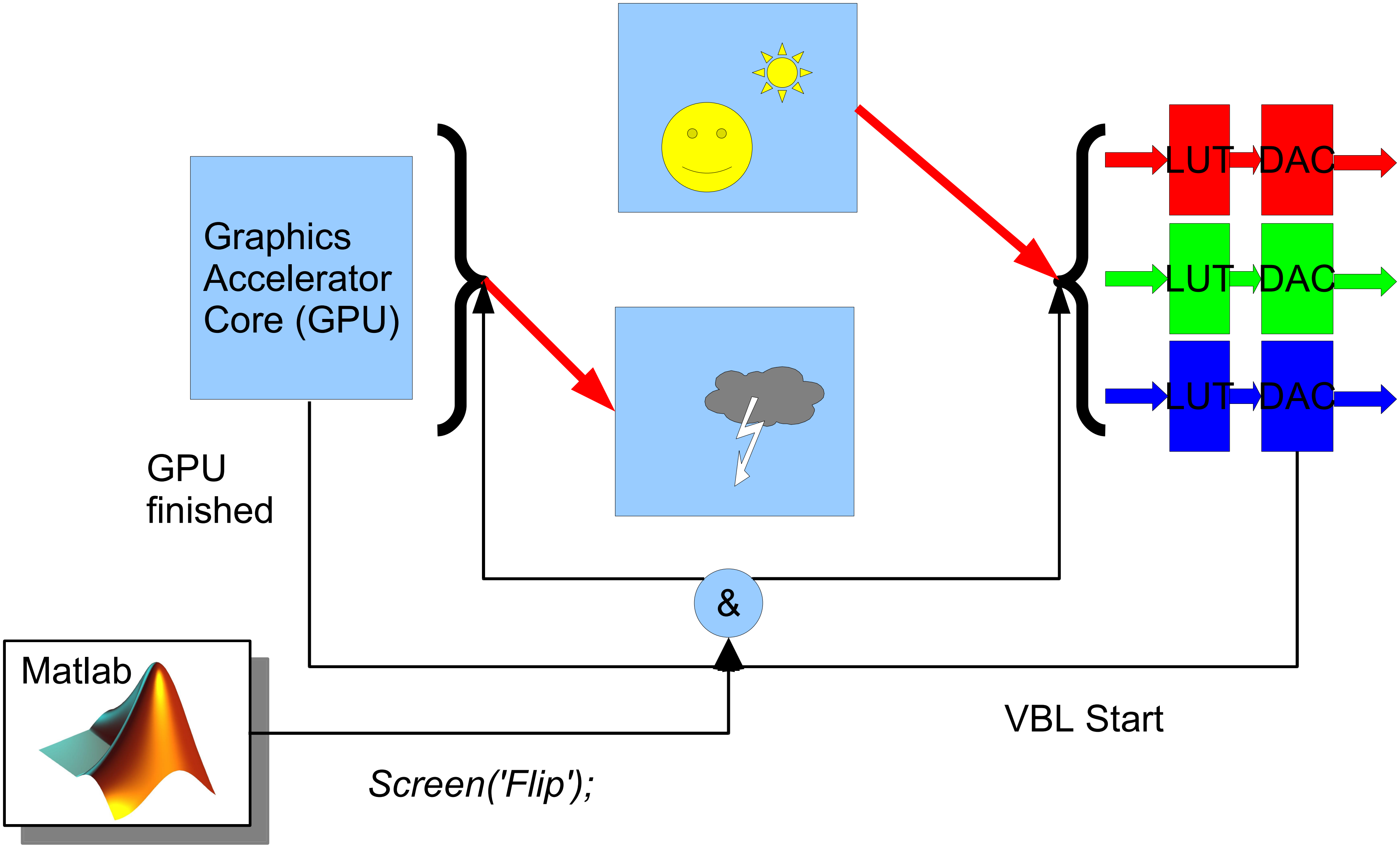
&

Matlab

VBL Start

Screen('Flip');

Double buffered drawing model - Implementation:



PTB-2 ==> PTB-3 : 'Flip' replaces 'Waitblanking'

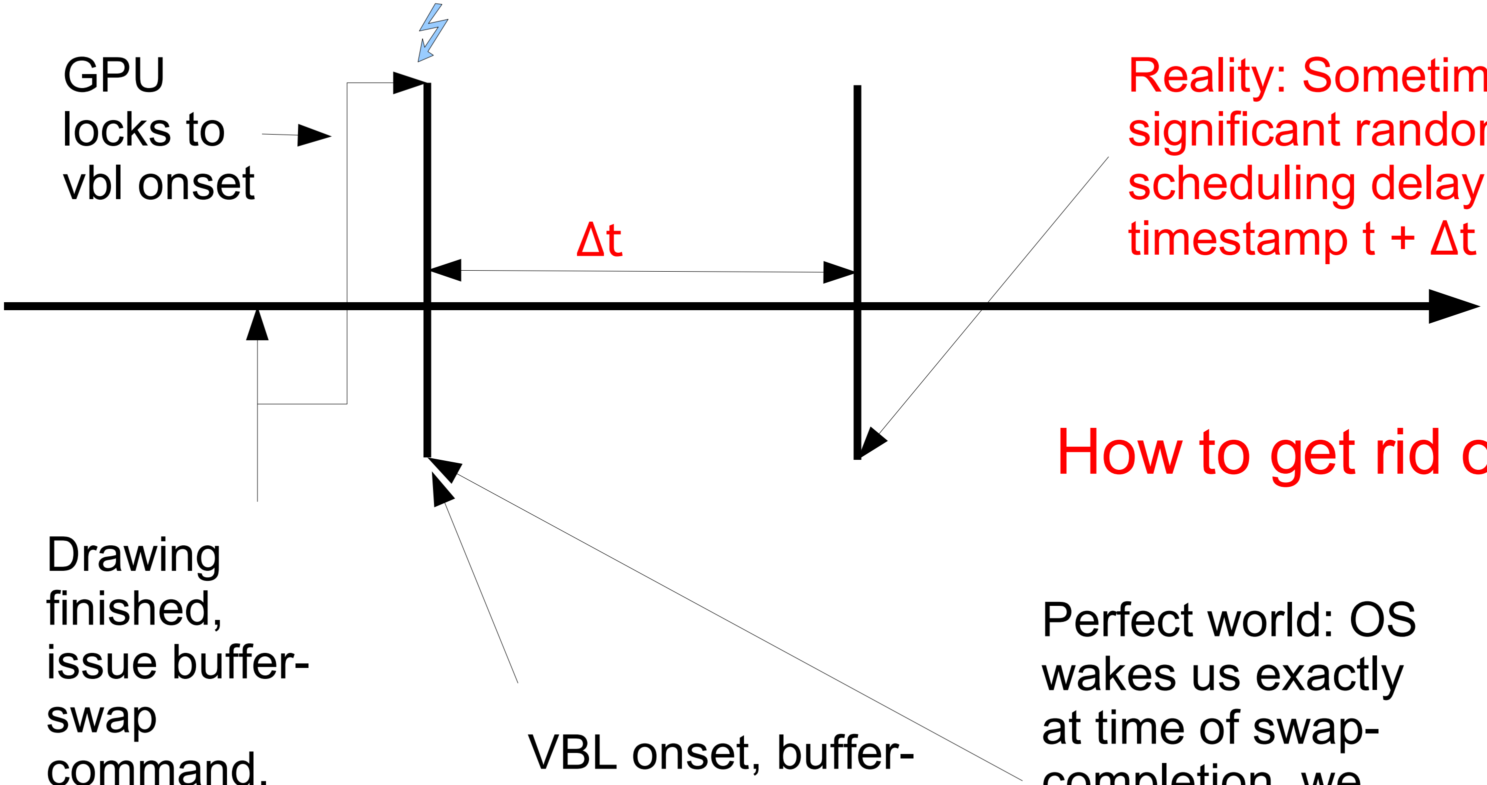
- Old style:

  ```
  Screen('WaitBlanking', window);
  ...very limited number of Screen() drawing commands... or
  Screen('CopyWindow', myoffscreen, window);
  ```
- New style:

  ```
  ...Arbitrary number of arbitrary complex drawing commands...
  tonset = Screen('Flip', window, myrequestedonsettime);
  ```

- Advantages:
  - Drawing commands can take much longer than a video refresh cycle.
  - No need for Offscreen windows, but still available if you want 'em.
  - Frame-accurate, tear-free stimulus onset at requested time.
  - Provides sub-millisecond accurate stimulus onset timestamps.

Screen: Stimulus onset timestamps



GPU locks to vbl onset

Δt

Reality: Sometimes significant random scheduling delay – Wrong timestamp t + Δt

How to get rid of Δt ?

Drawing finished, issue buffer-swap command, go to sleep, tell OS to wake us after swap-completion.

VBL onset, buffer-swap in sync with VBL, “Stimulus onset” at t

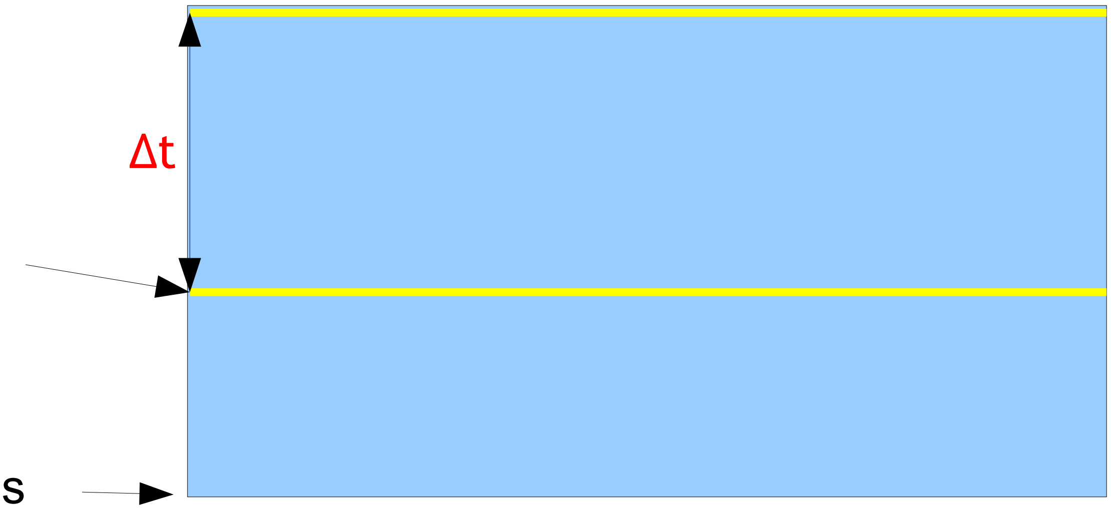
Perfect world: OS wakes us exactly at time of swap-completion, we take high-precision timestamp of stimulus onset.

Screen: Beamposition as phase-locked display clock

⇒ Δt = (beampos/height) * refresh

⇒ tonset = tmeasured − Δt

Display (Scanout engine) @ 100 Hz =
10 msecs per refresh cycle.



Current beam
position 627 ==
5.784 ms since
start of refresh.

Height = 1064 scanlines
(1024 visible + 40 VBL)

Reduces jitter / noise in timestamps to less than 0.1 ms on modern systems.

⇒ Accurate timestamps without special hardware.

⇒ Currently supported on MS-Windows and MacOS/X PowerPC.

⇒ MacOS/X IntelMac uses accurate VBL interrupt timestamps by interfacing with low-level kernel driver.

Flat panels and projectors: Trouble for precise timing!

- Flip timestamps tell pretty accurate when first pixel of stimulus image is "leaving" the graphics card.
- Equal to onset time for CRT's.
- Equal to *earliest possible* onset time for flat panels and projectors, but:
  - Long, non-linear switch latencies on liquid crystal displays. Per-pixel timecourse!
  - Long processing latencies on panels with OverDrive technology. Up to 50 ms.
  - Audio-Video sync?!?

Here used to be some example graphs of...

Timecourse for black -> gray level transition on overdrive display.

...from Toms-Hardware Guide. We don't have permission to publish them yet.

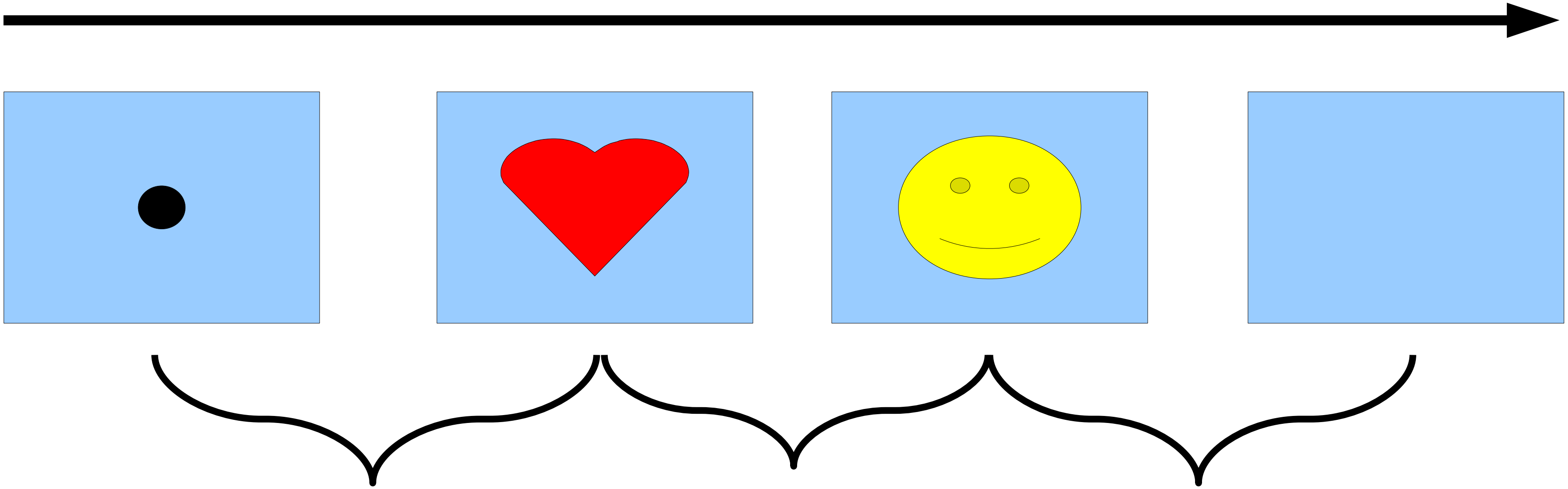
See the originals and background article on:

<http://www.tomshardware.co.uk/2005/11/10/the_secret_sauces_of_thguk/index.html>

Transition times for black -> gray level transition. Graphs taken from XBit-Labs:

<http://www.xbitlabs.com/articles/other/display/lcd-guide.html>

Screen('Flip') – Example 1: Fix->Prime->Target



500 ms

100 ms

200 ms

Screen('Flip') – Example 1: Fix->Prime->Target

1. % Draw fixation spot to backbuffer:
   ```
   winRect = Screen('Rect', win); slack = Screen('GetFlipInterval', win)/2;
   Screen('FillOval', win, 255, CenterRectInRect([0 0 20 20], winRect));
   ```
2. % Show fixation spot on next retrace, take onset timestamp:
   ```
   tfixation_onset = Screen('Flip', win);
   ```
3. % Draw prime stimulus image to backbuffer:
   ```
   Screen('DrawTexture', win, primeImage);
   ```
4. % Show prime exactly 500 msecs after onset of fixation spot:
   ```
   tprime_onset = Screen('Flip', win, tfixation_onset + 0.500 - slack);
   ```
5. % Draw target stimulus image to backbuffer:
   ```
   Screen('DrawTexture', win, targetImage);
   ```
6. % Show target exactly 100 msecs after onset of prime image:
   ```
   ttarget_onset = Screen('Flip', win, tprime_onset + 0.100 - slack);
   ```
7. % Show target exactly for 200 msecs, then blank screen.
   ```
   ttarget_offset = Screen('Flip', win, ttarget_onset + 0.200 - slack);
   ```

⇒ *Choose your presentation times as a multiple of the video refresh duration! 100 Hz is a good refresh setting for 10 ms timing granularity.*

Screen('Flip') – Example 2: Animation with const. fps

```
% Get basic timing info: Duration of a single video refresh interval:
refresh = Screen('GetFlipInterval', win);
% Synchronize to retrace at start of trial/animation loop:
vbl = Screen('Flip', win);
% Loop: Cycle through 300 images:
for i=1:300
   % Draw i'th image to backbuffer:
   Screen('DrawTexture', win, myImage(i));
   % Show images exactly 2 refresh cycles apart of each other:
   vbl = Screen('Flip', win, vbl + (2 – 0.5) * refresh);
   % Keyboard checks, whatever... Next loop iteration.
end;
% End of animation loop, blank screen, record offset time:
toffset = Screen('Flip', win, vbl + (2 – 0.5) * refresh);
```

⇒ *Choose your monitors video refresh rate as multiple of wanted animation framerate! For 25 fps -> 75 Hz or 100 Hz is a good refresh setting. For 30 fps -> 60 Hz, 90 Hz, 120 Hz...*

PTB1/2 “Compatibility mode”:

- You can get the old single-buffered drawing model back:

```
% Add this at top of your script:
Screen('Preference', 'EmulateOldPTB', 1);
```

- Reenables *Screen('WaitBlanking')* and immediate drawing.
- No timing guarantees, no support with problems!
- Most new interesting features disabled.
- Useful for “quick and dirty” porting of old code?

Basic 2D drawing commands

- Same 2D drawing primitives as in PTB-2, just *faster*:
  - (Filled) Circles and ellipses:
    Screen('FrameOval', window, color, boundingrect [, penWidth]);
    Screen('FillOval', window, color, boundingrect);
  - (Filled) Rectangles:
    Screen('FrameRect', window, color, boundingrect [, penWidth]);
    Screen('FillRect', window, color, boundingrect);
  - Lines of different thickness and stipple patterns:
    Screen('DrawLine', window, color, fromH, fromV, toH, toV[,penWidth]);
  - (Filled) Arcs:
    Screen('DrawArc', window, color, boundingrect, startAngle, arcAngle);
    Screen('FrameArc', window, color, boundingrect, startAngle, arcAngle);
    Screen('FillArc', window, color, boundingrect, startAngle, arcAngle);
  - (Filled) convex and concave Polygons:
    Screen('FillPoly', window, color, xy);
- Text drawing with detail improvements:
  - High quality anti-aliased text on MacOS/X.
  - Readable text on Windows (*We're probably working on it soon... ;-)*
  - Convenient text formatting: *DrawFormattedText()* ...

Fast 2D batch drawing and parallelism:

- Some commands are optimized for fast batch-drawing:
  - Distribute Matlab->PTB call overhead of approx. 20 µs per call onto many primitives. Distribute setup overhead of drawing calls, allow some internal optimizations.
- Instead of drawing one primitive at a time, define hundreds of similar primitives and draw them with 1 call:

Instead of:

```
Screen('FillRect', win, [red1 green1 blue1], [left1 top1 right1 bot1]);
Screen('FillRect', win, [red2 green2 blue2], [left2 top2 right2 bot2]);
...
Screen('FillRect', win, [redn greenn bluen], [leftn topn rightn botn]);
```

Write:

```
mycolors = [red1 green1 blue1; red2 green2 blue2; ... ; redn greenn bluen];
myrects = [left1 top1 right1 bot1; left2 top2 right2 bot2; ... ; leftn topn rightn botn];
Screen('FillRect', win, mycolors, myrects);
```

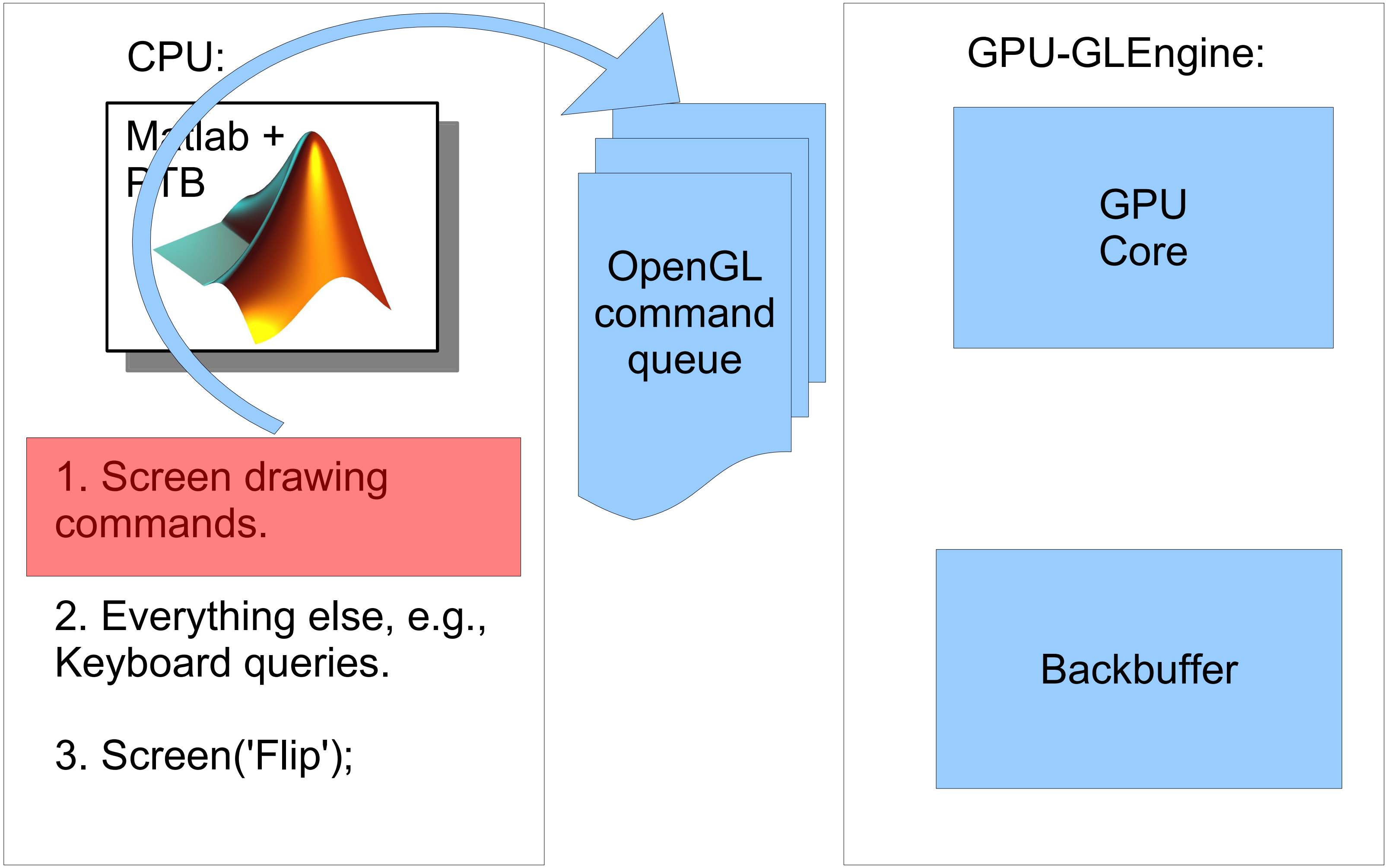
Fast 2D batch drawing and parallelism:

- Currently supported for fast batch drawing: Rectangles, Ellipses/Ovals, Anti-Aliased dots, Anti-Aliased lines, textures.
  - *Screen('DrawDots', ...);*
  - *Screen('DrawLines', ...);*
  - *Screen('DrawTextures', ...);*

- Typical speedup on Intel MacBookPro (n=500 items per frame):
  - Filled Rectangles of 80x80 pixels: 1.6 µs/item vs. 31 µs/item = 19.4x
  - Framed Rectangles of 80x80 pixels: 1.2 µs/item vs. 36 µs/item = 30.0x
  - Filled Ovals of 80x80 pixels diameter: 6.2 µs/item vs. 37 µs/item = 6.0x
  - Textures of 80x80 pixels: 12.4 µs/item vs. 36 µs/item = 2.9x
- Test it yourself: *DrawingSpeedTest*

- Another speedup measure: Parallelize!
  Screen('DrawingFinished', window);

Screen: Fast batch drawing, OpenGL parallelism



CPU:

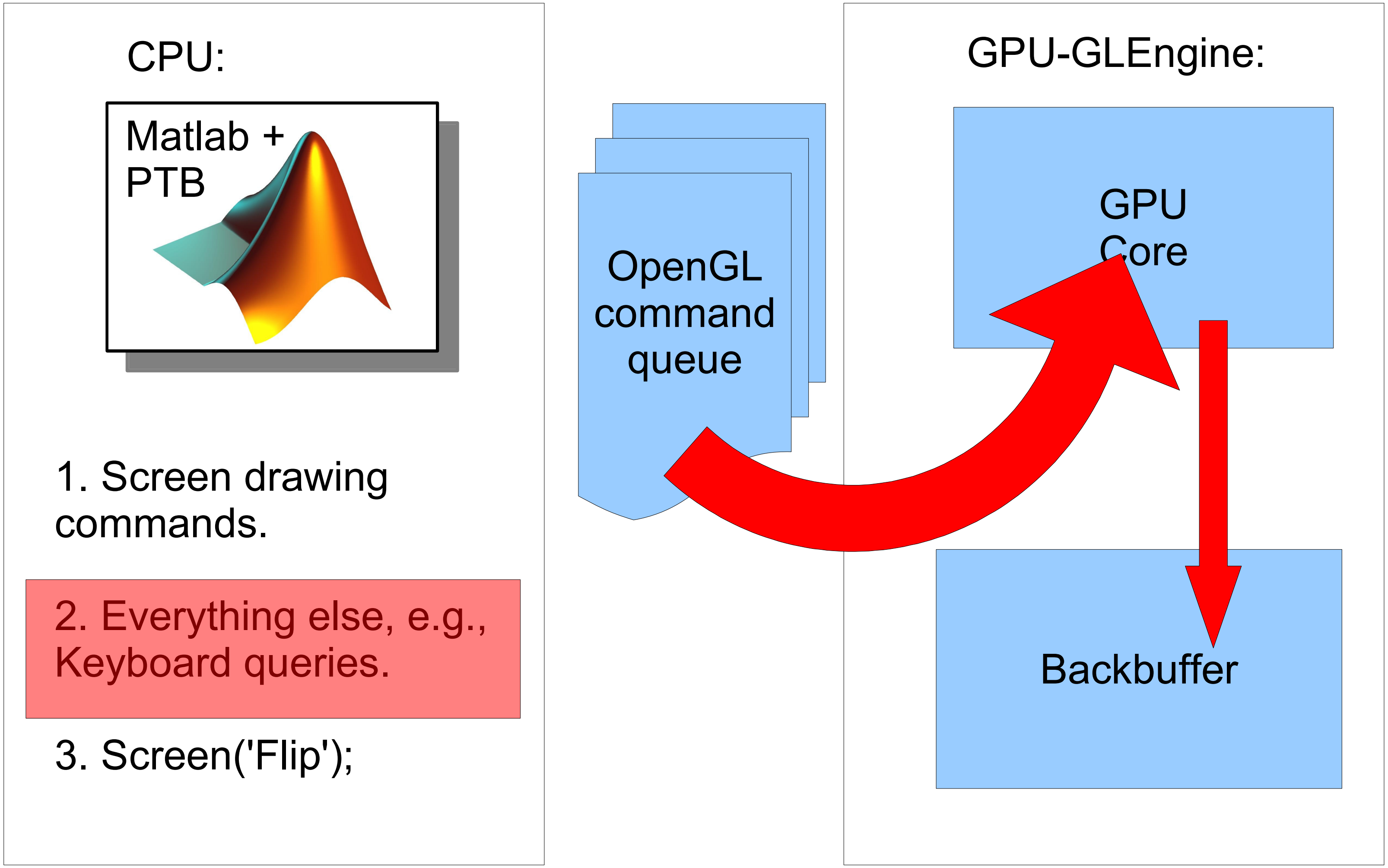
Matlab +
PTB

1. Screen drawing commands.

2. Everything else, e.g., Keyboard queries.

3. Screen('Flip');

Screen: Fast batch drawing, OpenGL parallelism



CPU:

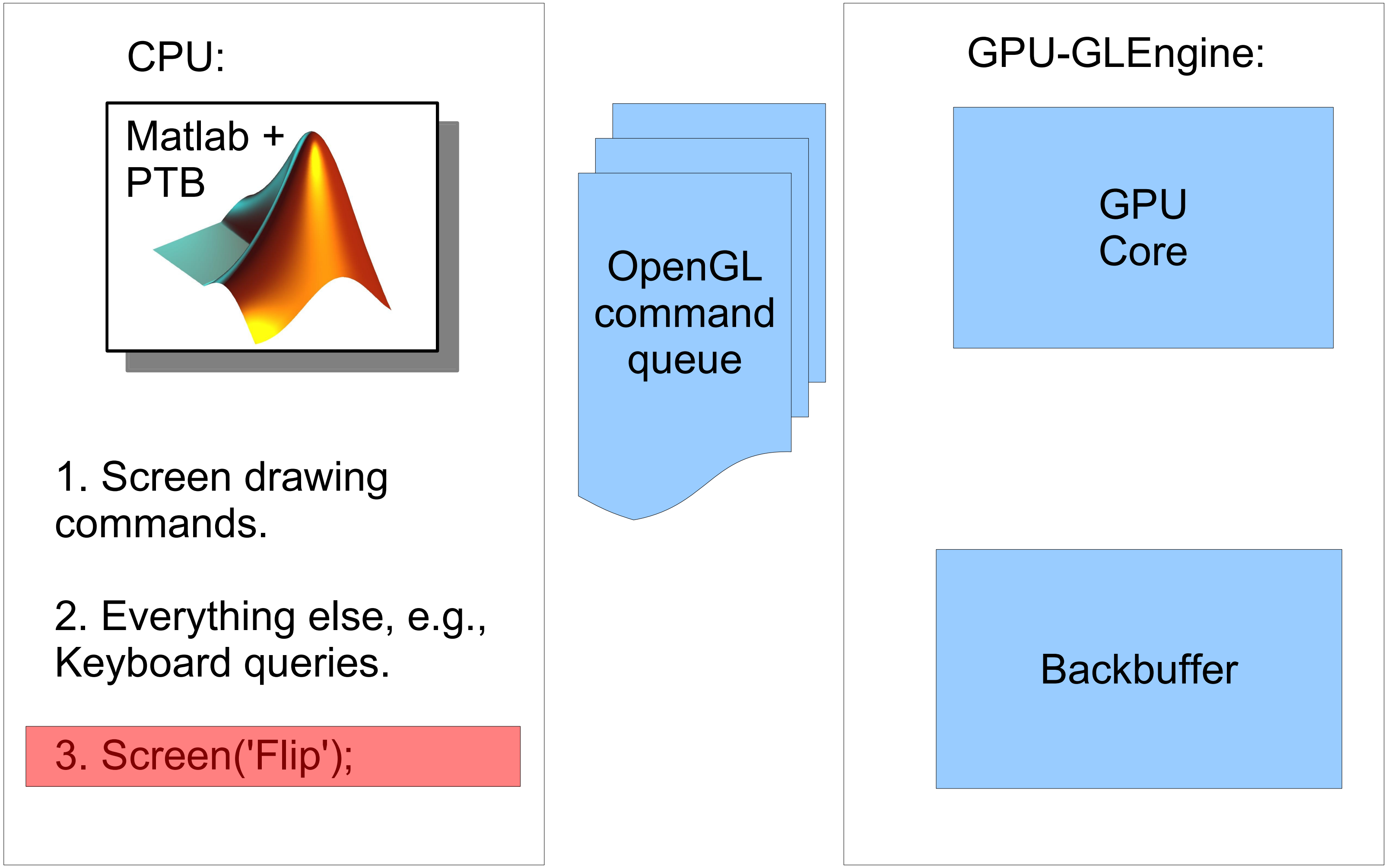
Matlab +
PTB

1. Screen drawing commands.

2. Everything else, e.g., Keyboard queries.

3. Screen('Flip');

Screen: Fast batch drawing, OpenGL parallelism



CPU:

Matlab + PTB

1. Screen drawing commands.

2. Everything else, e.g., Keyboard queries.

3. Screen('Flip');

OpenGL command queue

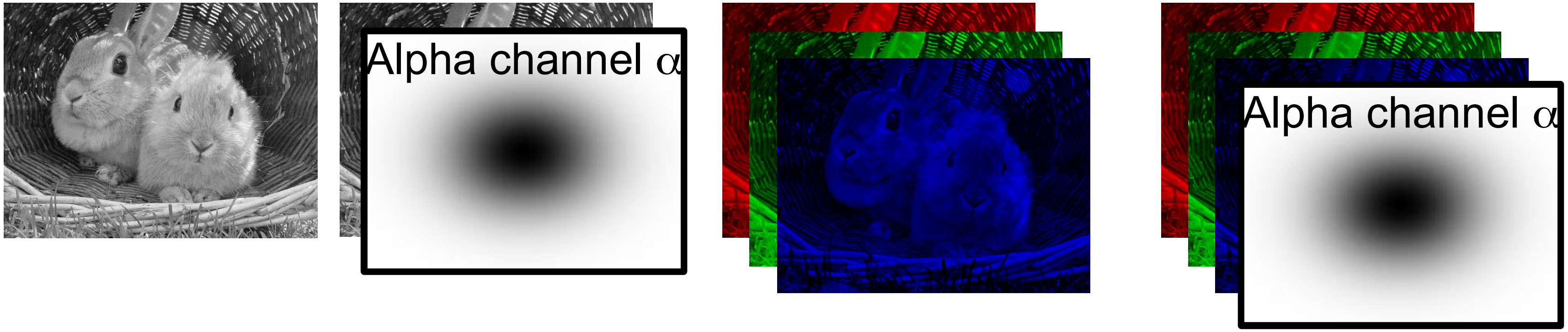
GPU-GLEngine:

GPU Core

Backbuffer

Textures as image representation

- "Enhanced" OpenGL textures as image representation:
  - Texture = *width x height* 2D matrix of pixels, each with up to 4 color components/values per pixel location *(x,y)*.
  - Texture image managed by graphics hardware for high performance, not directly accessible by Matlab. Referenced by a handle (a unique id).
  - Think of them as containers for image data or other 2D data.
- One to four channels are supported:



Textures as image representation

- Can have alpha channel for encoding of transparency or "weights".
- Can be represented with floating point precision and sign:
  - 16 bit floating point: 1 bit sign, 5 bit exponent, 10 bit mantissa Effective resolution for visible content: 3 decimal digits or 1024 levels.
  - 32 bit floating point: 1 bit sign, 8 bit exponent, 23 bit mantissa Effective resolution for visible content: 6.5 decimal digits or 8 million levels.
- Textures can originate from different sources:
  - From Matlab (Matrices and loaded image files):
    matlabMatrix = imread('MyCuteStimulusImage.jpg');
    mytex = Screen('MakeTexture', mywindow, matlabMatrix);
  - From Quicktime movie playback engine:
    mytex = Screen('GetMovieImage", mywindow, mymovie);
  - From builtin video capture engine:
    mytex = Screen('GetCapturedImage', mywindow, mycamera);
  - From OpenGL routines:
    mytex = Screen('SetOpenGLTexture', mywindow, ...);

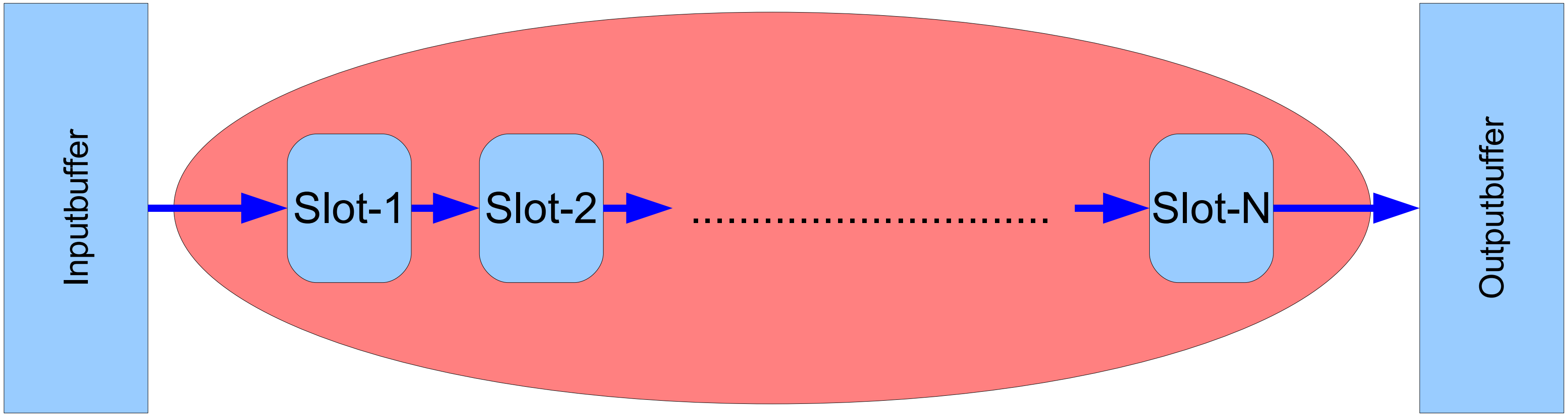
Textures: What you can do with them

- Drawing textures:
  ```
  Screen('DrawTexture', mywindow, mytex [,src][,dst][,rotAngle]);
  Screen('CopyWindow', mytex, mywindow, ...);
  ```
- Drawing *into* textures: Textures as Offscreen window buffers...
  ```
  mytex = Screen('OpenOffscreenWindow', mywindow [, color] [, rect])
  ```
- Drawing *with* textures: Textures as stencils / brushes...
- Transforming textures: GPU image processing:
  Example: Convolution with 13x13 gaussian blur operator:

  Creation:
  ```
  blurop = CreateGLOperator(window);
  Add2DConvolutionToGLOperator(blurop, fspecial('gaussian', 13, 5.5));
  ```

  Application:
  ```
  blurredtex = Screen('TransformTexture', intex, blurop);
  ```

GPU image processing operator:



Slot can contain:

- Builtin function
- Matlab M-Function
- C-Callback function
- GLSL GPU-Shader

Image buffers:

- RGBA color format (Color + Transparency)
- 8 bits per channel (Standard framebuffer)
- 16 bits fixed point (ATI graphics only)
- 16 bits floating point (≈10 bpc resolution + sign)
- 32 bits floating point (≈23 bpc resolution + sign)

GPU image processing

Example GLSL fragment shader:

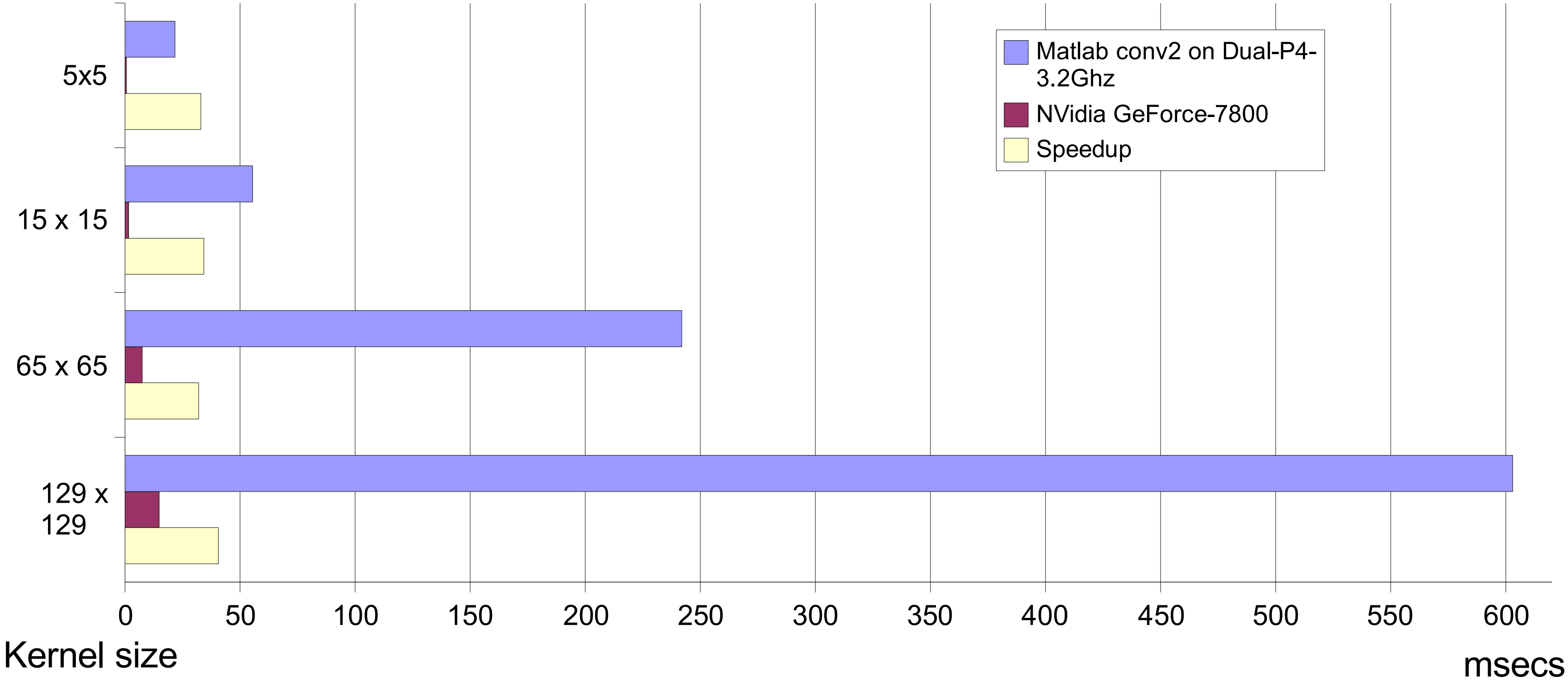
Conversion of color image into grayscale image

```
const vec4 ColorToGrayWeights = { 0.3, 0.59, 0.11, 0.0 };
uniform sampler2DRect Image;
void main()
{
   vec4 incolor = texture2DRect(Image, gl_TexCoord[0].st);
   float luminance = dot(incolor, ColorToGrayWeights);
   gl_FragColor.a = incolor.a;
   gl_FragColor.rgb = luminance;
}
```

- Executed once for each output pixel.
- Can read from many input pixels + 1D/2D/3D lookup tables + constants + builtin variables (e.g., position) to compute color of output pixel.
- Many standard operations supported in C-like language, for scalars, vectors and matrices: max, min, clamping, comparison, linear interpolation, sin, cos, exp, log, pow, add, multiply, dotproduct, crossproduct, norm, some screen space derivatives...
- Operates in IEEE single precision float.
- SIMD streaming computations *very* fast.

Example: Speedup for 2D image convolution

Execution time for 1 channel convolution with 512x512 image:



- Speedup for n channels = n * Speedup for 1 channel (Linear scaling)
- Typical speedup vs. Matlab ≈ n * 33 for separable convolution.
- Test it yourself: *ConvolutionKernelTest.m*
- Needs *recent* hardware!!!

Procedural textures:

- Only defined algorithmically (GLSL shader), not (only) by image matrix.
- No image storage requirement: "Infinite resolution and size"
- Useful for parametrically defined complex dynamic stimuli?
- See *ExpandingRingsDemo* and *MandelbrotDemo* for examples.

<u>Example GLSL fragment shader: "Expanding rings shader"</u>

```
uniform vec2  RingCenter;
uniform float RingWidth;
uniform float Radius;
uniform float Shift;
uniform vec4  secondColor;
void main()
{
    vec2 pos = gl_TexCoord[0].xy;
    float d = distance(pos, RingCenter);
    if (d > Radius) discard;
    d = floor((d - Shift) / RingWidth);
    gl_FragColor = mix(gl_Color, secondColor, mod(d,2.0));
}
```

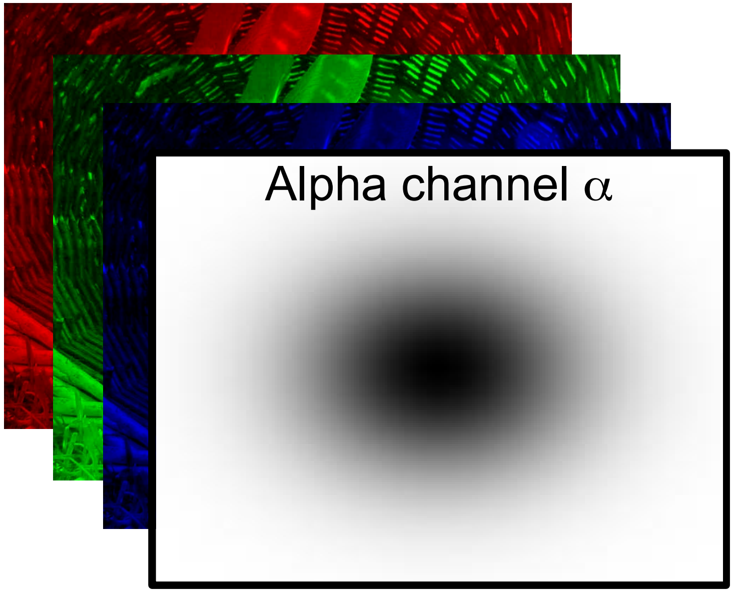
Screen – Textures: More stuff you can do...

- Scrolling of textures in subpixel steps for drifting gratings: *DriftDemo2* - Uses subpixel accurate bilinear interpolation. 1/64th pixels or 1/256th pixels (slower) on ATI, 1/256th pixels on NVidida.
- Rotation of textures with bilinear interpolation: *AlphaRotateDemo*
- Up-/ Downscaling with bilinear interpolation.
  - *Screen('DrawTexture', win, tex, srcRect, dstRect, Angle, filter, alpha)*

Screen: Textures and Alpha blending

- Textures can have an alpha channel (per pixel weight mask) and a global alpha value in addition to their luminance or color channels.
- Selected pen color for 2D/3D drawing can have an alpha component (applies to whole drawn object): *color = [red green blue alpha];*
- Framebuffer has an alpha channel as well, which stores alpha values of drawn textures and shapes.
- Alpha blending allows to combine color values of drawn shape or texture pixels with existing color values of destination pixels in the framebuffer:

$$C_{newdestination} = (C_{source} \cdot W_{source}) + (C_{destination} \cdot W_{destination})$$



Screen: Textures and Alpha blending

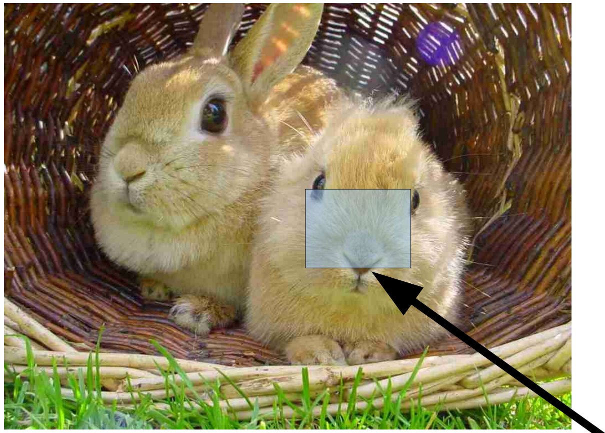
$$C_{newdestination} = (C_{source} \cdot W_{source}) + (C_{destination} \cdot W_{destination})$$

- Whenever a pixel is drawn into framebuffer: New pixel color is a weighted sum of previous pixel color and pixel color of "incoming" new fragment from drawn texture or shape.
- *Screen('Blendfunction', window, Wsource, Wdestination);* allows to select the way that the weights are selected for blending.
- Examples:
  - *Screen('Blendfunction',w,GL_ONE,GL_ZERO);* *% Overdraw*
    $\Rightarrow C_{newdestination} = (C_{source} \cdot 1) + (C_{destination} \cdot 0) = C_{source}$
  - *Screen('Blendfunction',w,GL_ONE,GL_ONE);* *% Superposition*
    $\Rightarrow C_{newdestination} = (C_{source} \cdot 1) + (C_{destination} \cdot 1) = C_{source} + C_{destination}$
  - *Screen('Blendfunction',w,GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);*
    $\Rightarrow C_{newdestination} = C_{source} \cdot \alpha_{source} + C_{destination} \cdot (1 - \alpha_{source})$ *% Masking*

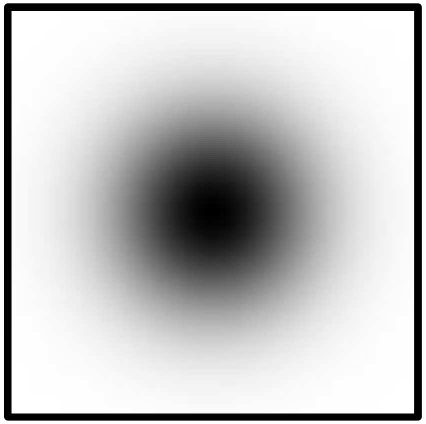
Screen – Textures: What you can do with them

- Masking textures with textures: Apertures *(AlphaImageDemo)*

imgtex=



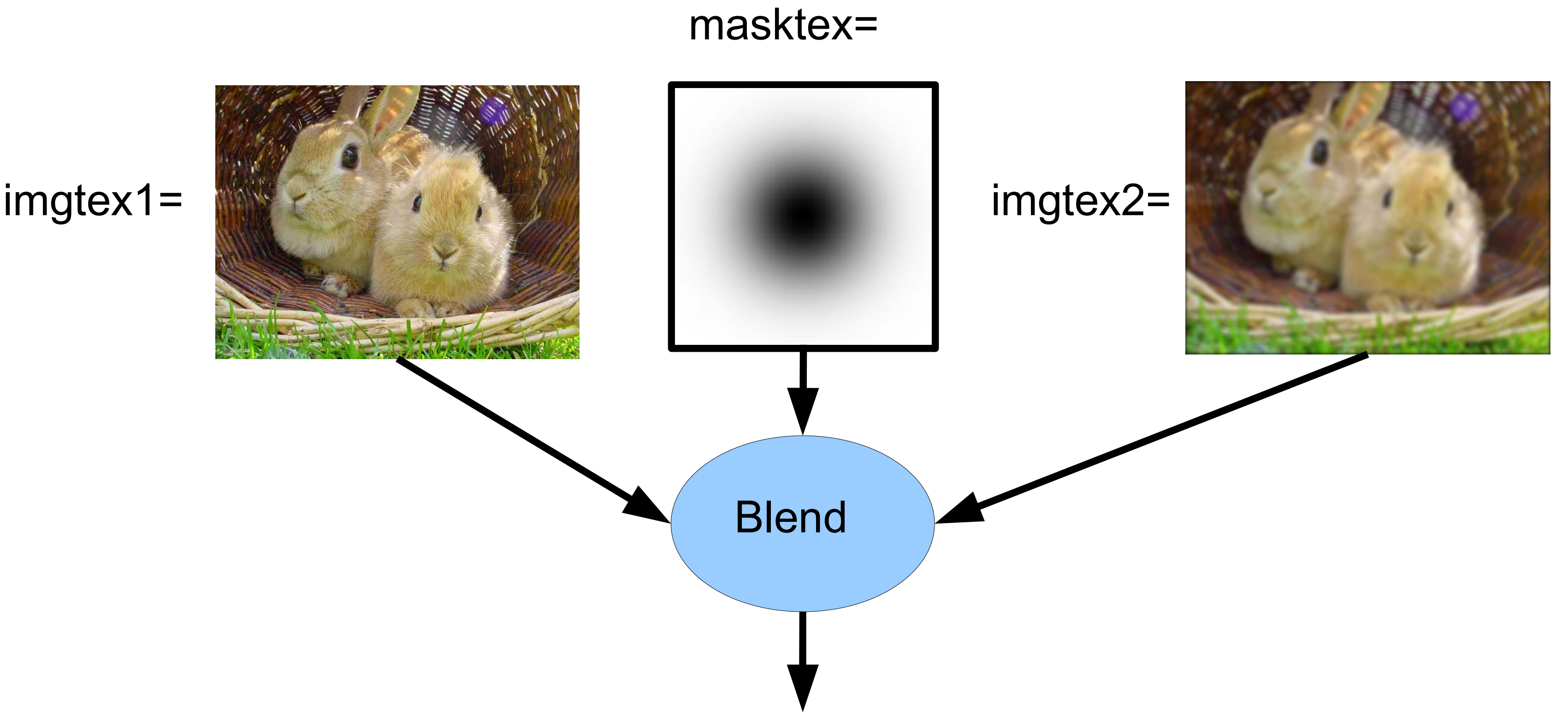
masktex=



```
% Enable alpha blending, suitable for apertures:
Screen('BlendFunction',mywindow, GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
% ... Query mouseposition or eyetracker to find area of interest apertureRect ...
% Draw image for current frame:
Screen('DrawTexture', mywindow, imagtex, apertureRect, apertureRect);
% Overdraw -- and therefore alpha-blend -- with gaussian alpha mask
Screen('DrawTexture', w, masktex, [], apertureRect);
```

Screen – Textures: What you can do with them

- Blending textures with textures: Compositing *(GazeContingentDemo)*



masktex=

imgtex1=

imgtex2=

Blend

Movie playback with Quicktime

- Quicktime movie playback supported on MacOS/X and Windows.
- With/without sound, looped/singleshot, at different speeds and backward. Frame accurate seeking is also supported.
- Simultaneous playback of multiple movies possible on fast hardware.
- Engine takes care of general timing and audio-video sync, you fetch the images as time-stamped textures and draw them at your will, e.g.:

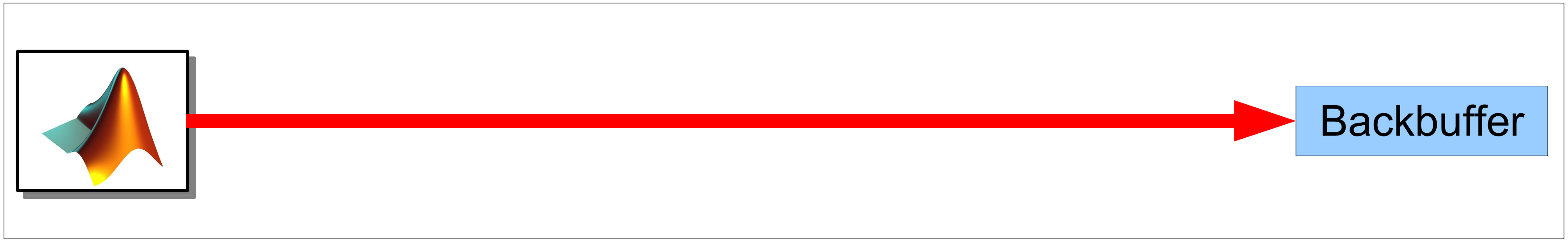
```
while ~KbCheck & tex ~= -1
        [tex pts] = Screen('GetMovieImage', window, movie);
        if tex > 0
                Screen('DrawTexture', window, tex);
                Screen('Close', tex);
                stimonset = Screen('Flip', window);
        end
end
```

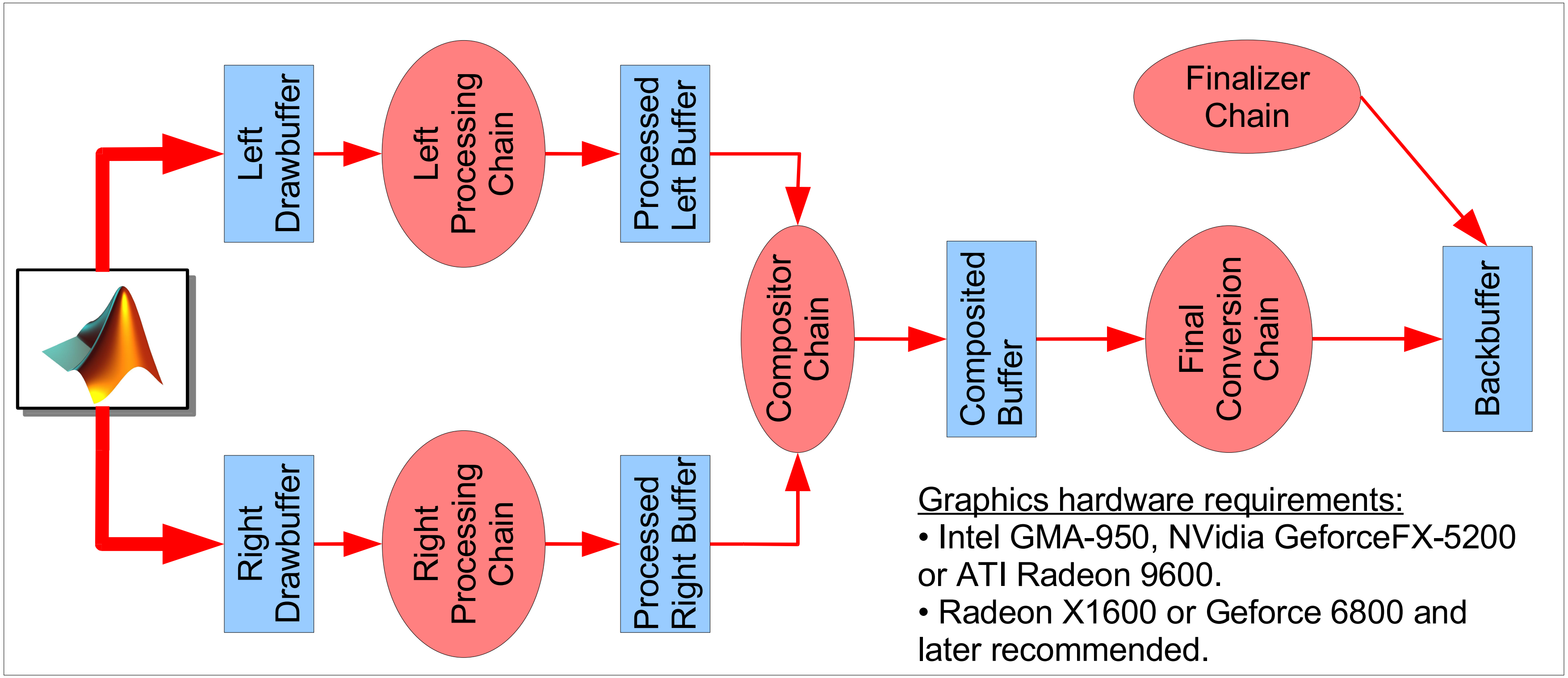
- Engine provides presentation timestamps *pts* to correlate stimulus onset or subjects responses with events in the video stream.

Screen – Video capture and recording

- Video capture and recording supported on MacOS/X and Windows by use of Quicktime Sequence Grabber API:
  - Needs VDIG plugin for specific source: Tested with DV cameras, USB Webcams and IIDC Firewire cameras. Works well on OS/X, works somewhat on Windows.
  - Provides moderate framerates and timing accuracy, e.g., 30 fps.
  - Limited control of attached cameras.
- Excellent video capture support for IIDC Firewire cameras on Linux:
  - Number of simultaneously controlled cameras only limited by bus bandwith.
  - Framerates > 300 fps and sub-millisecond accurate timestamping.
  - Very low latency. Low level control of all aspects of camera operation.
- Same programming interface as movie playback engine:
  - Subcommands to control the capture setup and operation.
  - Engine provides images as time-stamped textures, you fetch, process, draw them in a loop.

Screen: GPU image processing pipeline





Graphics hardware requirements:

- Intel GMA-950, NVidia GeforceFX-5200 or ATI Radeon 9600.
- Radeon X1600 or Geforce 6800 and later recommended.

GPU stimulus post - processing pipeline:

- Automatic application of image post-processing to stimuli at *Screen('Flip')* time.
- Simple setup at start of script:
  1. *PsychImaging('PrepareConfiguration');*
  2. *PsychImaging('AddTask', 'LeftView','GeometryCorrection','mon1.mat');*
  3. *PsychImaging('AddTask', 'RightView','GeometryCorrection', 'mon2.mat');*
  4. *PsychImaging('AddTask', 'AllViews','FlipVertical');*

  ...

  n. *PsychImaging('OpenWindow', screenid, backcolor, ...);*

- Currently implemented:
  - High precision framebuffers 16 and 32 bpc floating point.
  - Display geometry correction: *DisplayUndistortionBezier.m*
  - Horizontal-/Vertical mirroring of images.
  - High performance driver for BrightSide-Technologies 16 bpc HDR display.
  - High performance driver for Cambridge Research Systems 14 bpc Bits++.

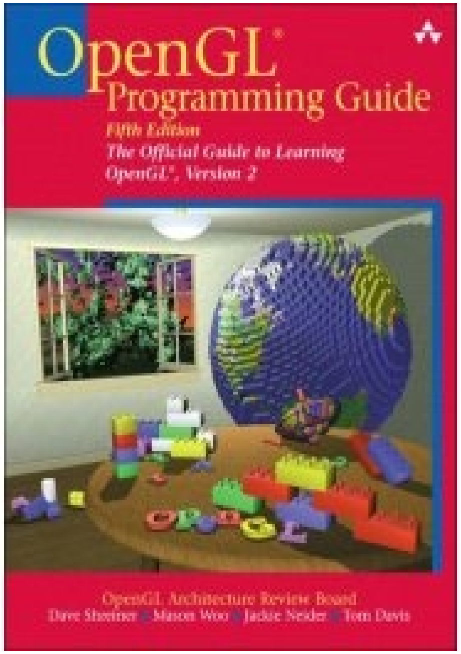
More stuff for GPU image processing pipeline

- Soon available as GLSL shader plugins and built-in functions:
  - Color to gray conversion, video deinterlacers.
  - CLUT based color conversion. Algorithmic conversions/mappings easy.
- Planned:
  - Per pixel Brightness/Gamma/Gain correction for displays?
- Future: Other things you'd like to see?

Screen: 3D graphics with OpenGL for Matlab

- OpenGL is *the* industry standard API for hardware-accelerated 3D computer graphics.
    - Cross-platform solution (MacOS/X, MS-Windows, Linux, Unixes, BeOS, Game-Consoles).
    - Easy to learn and use 3D state machine. Backwards compatibility and clean design is a top priority.
    - Provides standardized core functionality + vendor specific extensions.
- *MOGL* - Matlab interface to the OpenGL API (co-developed with Prof. Richard F. Murray, York University, Canada).
- *MOGL* Simple to use if you know how to use OpenGL in C!

Screen: 3D graphics with OpenGL for Matlab


OpenGL®
Programming Guide
Fifth Edition
The Official Guide to Learning
OpenGL®, Version 2
OpenGL Architecture Review Board
Dave Shreiner • Mason Woo • Jackie Neider • Tom Davis

- General information, pointers, tutorials: http://www.opengl.org

- Programming Guide for OpenGL 1.1 online: http://www.glprogramming.com/red/

- Later versions available in any good book store.

- The “Orange Book” about the GLSL GL Shading Language is also a good read for advanced users.

- Very nice tutorials with example code: http://nehe.gamedev.net

Screen: 3D graphics with OpenGL for Matlab

- Add a *InitializeMatlabOpenGL*; to top of your script to enable it.
- Wrap your 3D OpenGL Matlab code into pairs of:
  - *Screen('BeginOpenGL', window);* % Before issuing OpenGL commands...
  - *Screen('EndOpenGL', window);* % Before issuing Screen 2D commands...
- Use OpenGL commands nearly as in C:
  - *glEnd() --> glEnd;* % No need for empty brackets for void-functions.
  - *GL_ENABLE_LIGHTING --> GL.ENABLE_LIGHTING* % First _ replaced by .
  - *char mystr[100]; glGetString(GL_VERSION, (char*) mystr); --> mystr=glGetString(GL.VERSION);* % No memory pointer games.
  - Pass vectors and matrices directly as Matlab vectors or matrices.

Screen: 3D graphics with OpenGL for Matlab

- Helper functions provided for:
    - Loading and setup of textures. (see *MinimalisticOpenGLDemo*)
    - Interfacing with PTB's functions (see, e.g., *SpinningMovieCubeDemo*)
    - Loading of *simple* Alias-Wavefront OBJ 3D geometry files (Maya, 3DS-Max, ...)
    - Fast rendering of meshes and fast shape morphing. (see *MorphDemo*)
    - Loading/Setup of GLSL vertex- and fragment shaders. (see *GLSLDemo*).
- Also possible to intermix with compiled C OpenGL code:
    - Use Matlab's *mex* command to compile C OpenGL code into Matlab dynamic plugin. Your code contains pure C and OpenGL calls.
    - PTB performs display setup and management, bufferswaps, timing, response collection... == Acts as a GLUT replacement for Matlab.
- Get cross-platform OpenGL support without need for compilation + all the PTB extras like stereo display, accurate timing, image postprocessing and error checking.

Low latency, precisely timed sound with PsychPortAudio

- Based on PortAudio, a free, open-source, cross-platform audio library for realtime audio: <http://www.portaudio.com>
- Features:
  - Multi-channel playback, recording and full-duplex feedback operation.
  - Low-latency sound output on MacOS/X, and MS-Windows (ASIO required)
  - Precisely timed sound onset on MacOS/X, and MS-Windows (ASIO required).
- Successfully tested on:
  - MacOS/X, Intel MacBookPro with onboard sound.
  - MacOS/X, PowerPC-G5 with onboard sound.
  - Windows, M-Audio Delta multi-channel card with ASIO.
  - Windows, Creative Soundblaster X-Fi Xtreme Music (ASIO)(Erik Flister, UCSD).
  - Windows, Creative Soundblaster Audigy-2 ZS multi-channel (ASIO) (Virginie van Wassenhove, Caltech)
- Should work well on all OS/X Apple hardware and all Windows native ASIO hardware. For other machines: <http://www.asio4all.de>

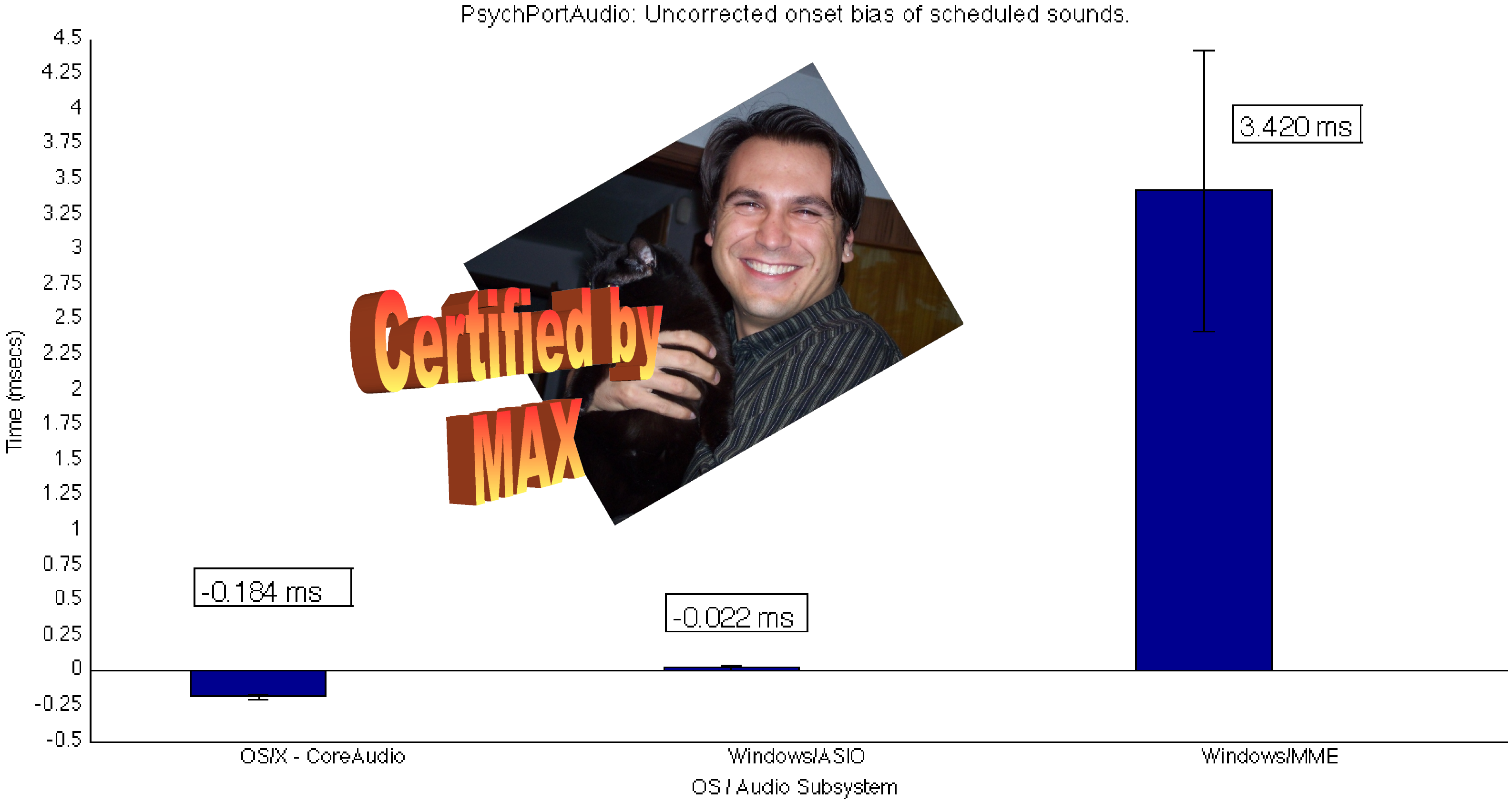
Low latency sound on Windows needs

- ASIO is a special low-latency, high timing precision driver architecture for Microsoft Windows, developed by Steinberg Media Technologies.
- "ASIO is a registered trademark and software of Steinberg Media Technologies GmbH": http://www.steinberg.de
- The standard sound subsystem of Microsoft Windows is not capable of reliably timed sound playback and capture with low latency!
- If you want low latency sound with high timing precision on Microsoft Windows, you'll need a sound card with native ASIO driver support and a special plugin for Psychtoolbox. Read "help InitializePsychSound" for instructions on how to get and install the plugin. Most better soundcards come with native ASIO support.
- If you don't have sound hardware with native ASIO driver support, you can try this: <http://www.asio4all.de> - Good luck! (May or may not work well, highly dependent on your sound hardware)

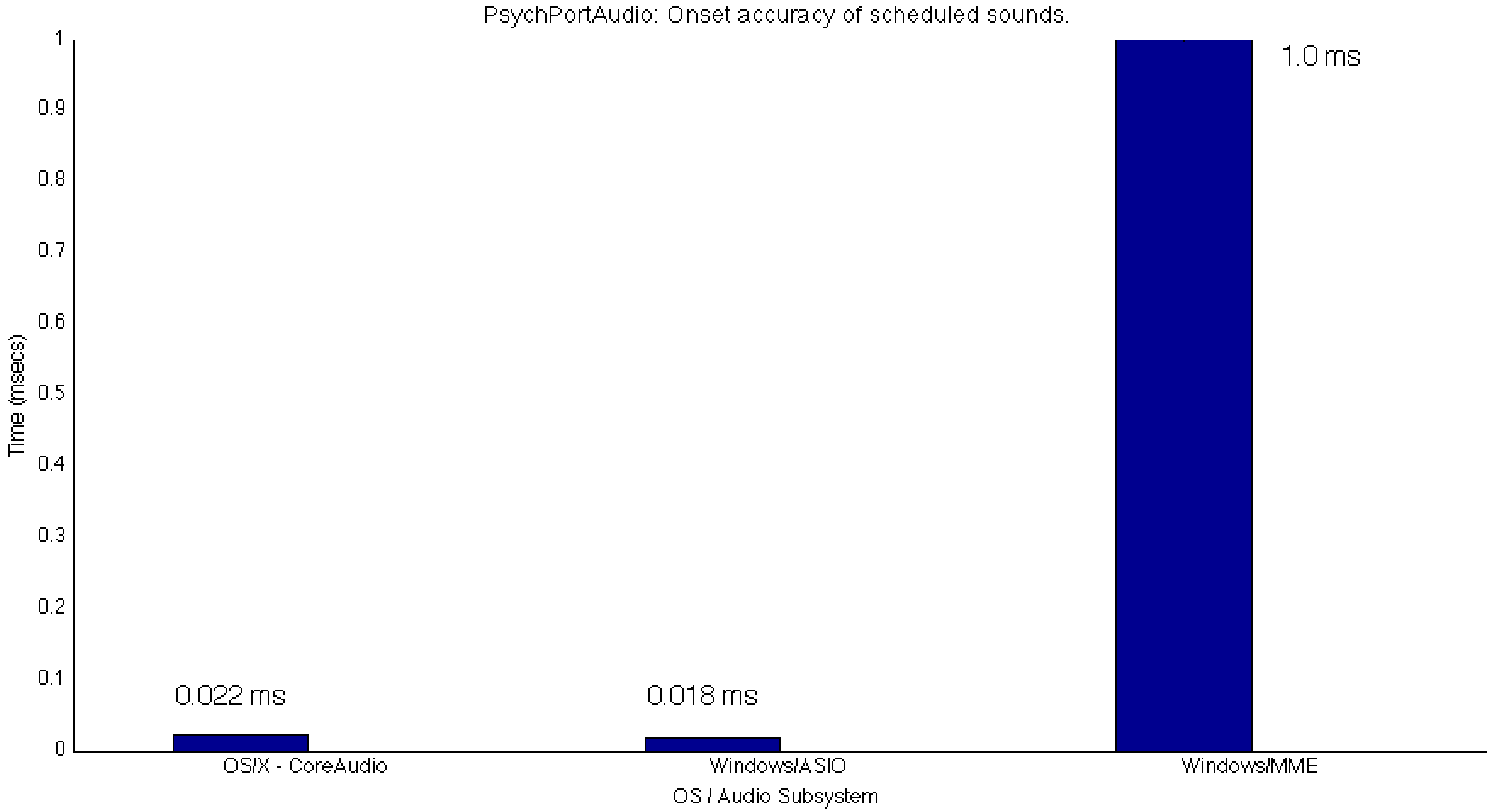
Low latency, timed sound with PsychPortAudio

- Simple interface, allows for asynchronous sound output at a scheduled system time, e.g., time *tvisualonset*:
  1. *Padevice = PsychPortAudio('Open', [deviceid], [mode], 2, 96000);*
  2. *Mysound = 0.9 * MakeBeep(1000, 0.1, 96000);*
  3. *PsychPortAudio('FillBuffer', Padevice, Mysound);*
  4. *PsychPortAudio('Start', Padevice, 5, tvisualonset);*
  5. *Visonset = Screen('Flip', window, tvisualonset – 0.004);*
  6. *...whatever...*
  7. *Audioonset = PsychPortAudio('Stop', Padevice);*
  8. *PsychPortAudio('Close' [,Padevice]);*
- Auto-selects settings for low latency, but overrides possible.
- High precision, low latency with standard hardware on OS/X.
- Special sound hardware with ASIO driver support required to get comparable performance on M$-Windows. E-mail me!

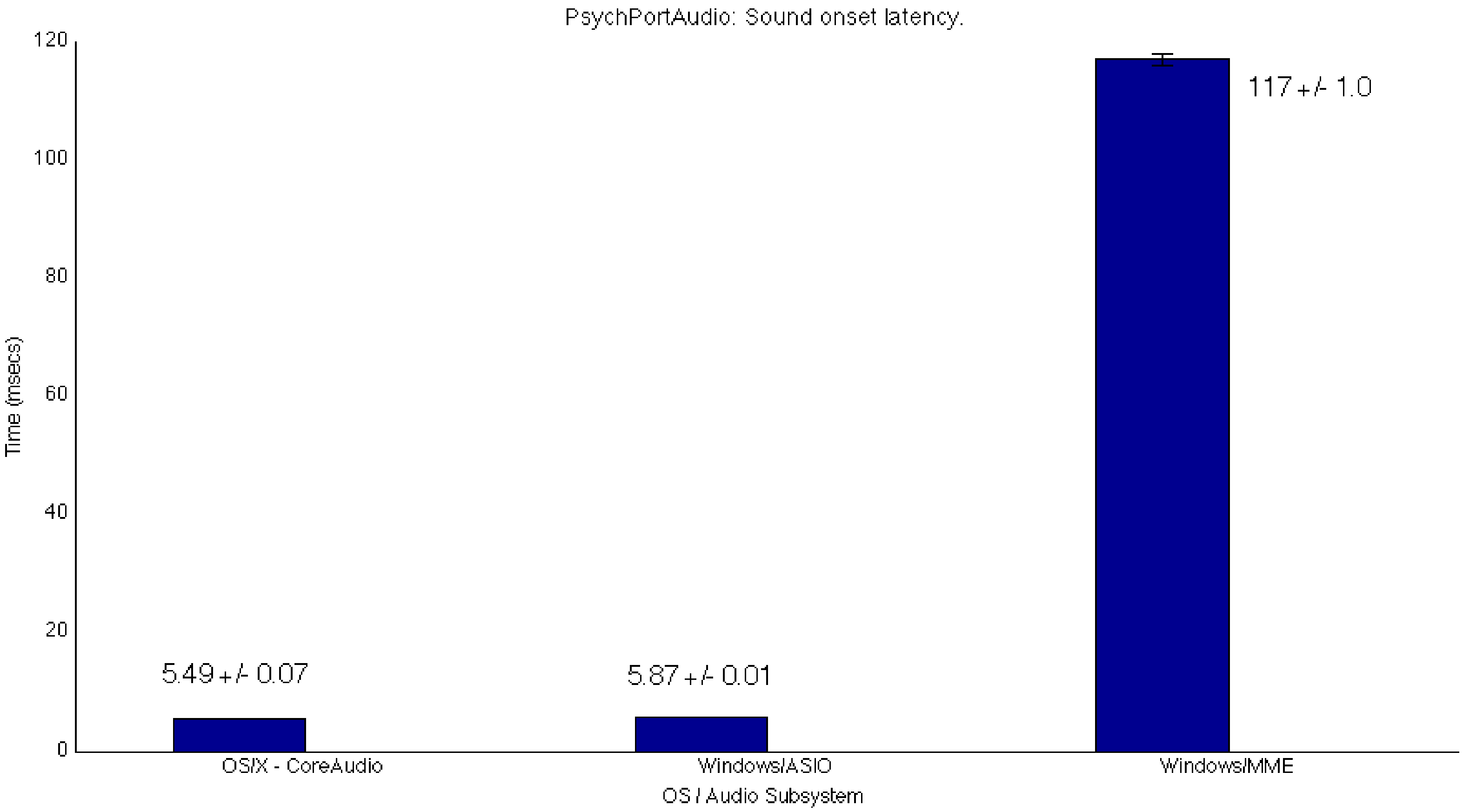
PsychPortAudio: Onset bias for scheduled sounds



PsychPortAudio: Onset accuracy for scheduled sounds



PsychPortAudio: Sound onset latency



PsychPortAudio - Demos!

- BasicSoundOutputDemo
- BasicSoundInputDemo
- BasicSoundFeedbackDemo
- PsychPortAudioTimingTest

Never ever trust audio output blindly!!!
Measure! Measure! Measure!
(At least once for your setup)

3D spatialized Sound with OpenAL

- OpenAL is a free, open-source, cross-platform library for 3D sound.
- Used in many computer games and virtual reality applications.
- Psychtoolbox provides a one-to-one interface to OpenAL, following the same logic as the OpenGL interface:
  - *InitializeMatlabOpenAL;* % At the top of your script to initialize it.
  - Then use AL calls nearly as in C – with the same slight differences as with OpenGL for Matlab.
  - Not yet supported: Sound capture and some of the extensions.
- OpenAL:
  - Create individual sound sources in 3D space, each with its own position, speed, orientation, sound emission properties and sounds attached.
  - Create a "Listener" with selectable position, orientation and speed.
  - Computes spatialized sound for headphones, stereo speakers, 5.1 surround setups, multi-speaker setups.
  - Computes attenuation, reverb, occlusion, echo, interreflection, doppler effects.

3D spatialized Sound with OpenAL

- Quality and performance depends on underlying sound system and capabilities of sound hardware:
    - On low-end sound hardware w/o 3D sound support: Built-in algorithms implemented in software: High CPU load, high latency, only positional stereo panning and simulation of inter-aural latency.
    - On MacOS/X: Use of head related transfer functions (HRTF) and simulation of reverb and occlusion on any sound hardware.
    - On higher-end sound hardware with onboard 3D sound processing: Complex spatialization algorithms, dozens of sound sources, customizable libraries of HRTF's:
      E.g., "SensAura" based cards: Select between 1111 pairs of HRTFs, depending on direction of sound source wrt. Listener. Provide software to customize set of HRTF's to individual listeners.
      Cfe. <http://www.sensaura.com>
- More info, tutorials, docs on <http://www.openal.org>

Basic commands for system control & timing

- *T = GetSecs*
  - Query time with microsecond resolution.
  - Uses highest resolution system clock for measurement of time.
- *WaitSecs(duration)*
  - Wait for a specified amount of time 'duration'.
  - On MacOS/X: Accurate to < 0.3 milliseconds on average.
  - On MS-Windows: Accurate to < 2 milliseconds on average on a modern machine.
- *Priority()* - Switch Matlab process to realtime-scheduling mode.

Basic commands for system control & timing

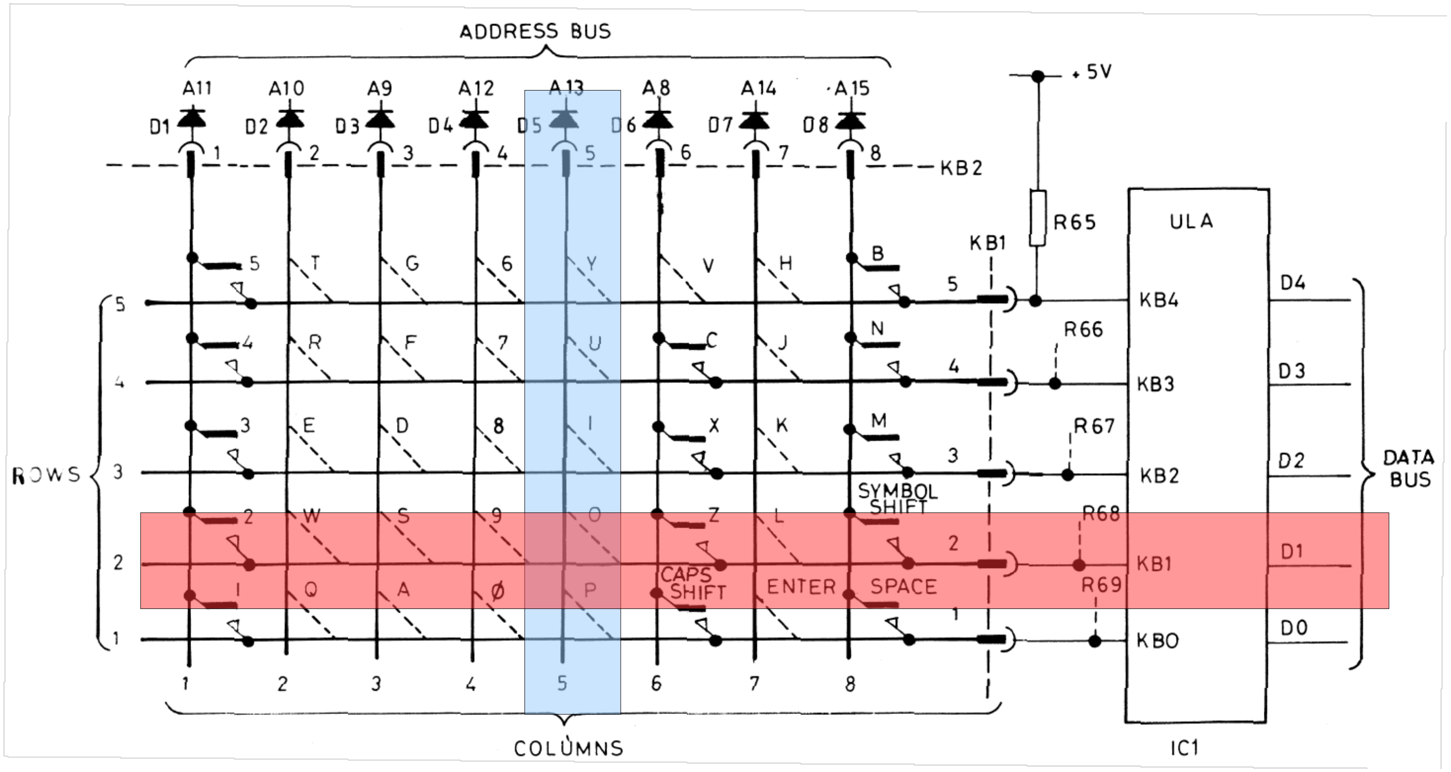
- Unmodified general purpose operating systems are not hard realtime-systems!
- For guaranteed timing accuracy of scheduling or for sub-millisecond accurate timing ⇒ Pick realtime OS, e.g., RealtimeLinux (<< 40 μs).
- Carefully configured system setups + well written scripts allow for millisecond accuracy most of the time:
  - Calm down your system: No virus checkers, software updates, ...
  - Obey some rules when writing your experiment scripts: See the FAQ section on Performance tuning on the Wiki.
  - PTB's *Priority()* command to switch Matlab into realtime scheduling mode. Significant reduction of timing noise.
  - PTB's optimized functions for timing, Matlabs builtin functions not good enough.
  - Detect and control for remaining infrequent timing glitches.

Basic response collection – Keyboard & Mouse

- *[x,y,buttons]=GetMouse(window);*
  Query current mouse position and mouse button state:
- *[down, secs, keycode]=KbCheck;*
  Query current state of *all* keys on a keyboard:
  - Can detect and report multiple simultaneous keypresses.
  - Can query multiple keyboards on Linux and OS/X, one on Windows.
  - Can mask out dead or stuck keys, often a problem on Laptops.
    DisableKeysForKbCheck()

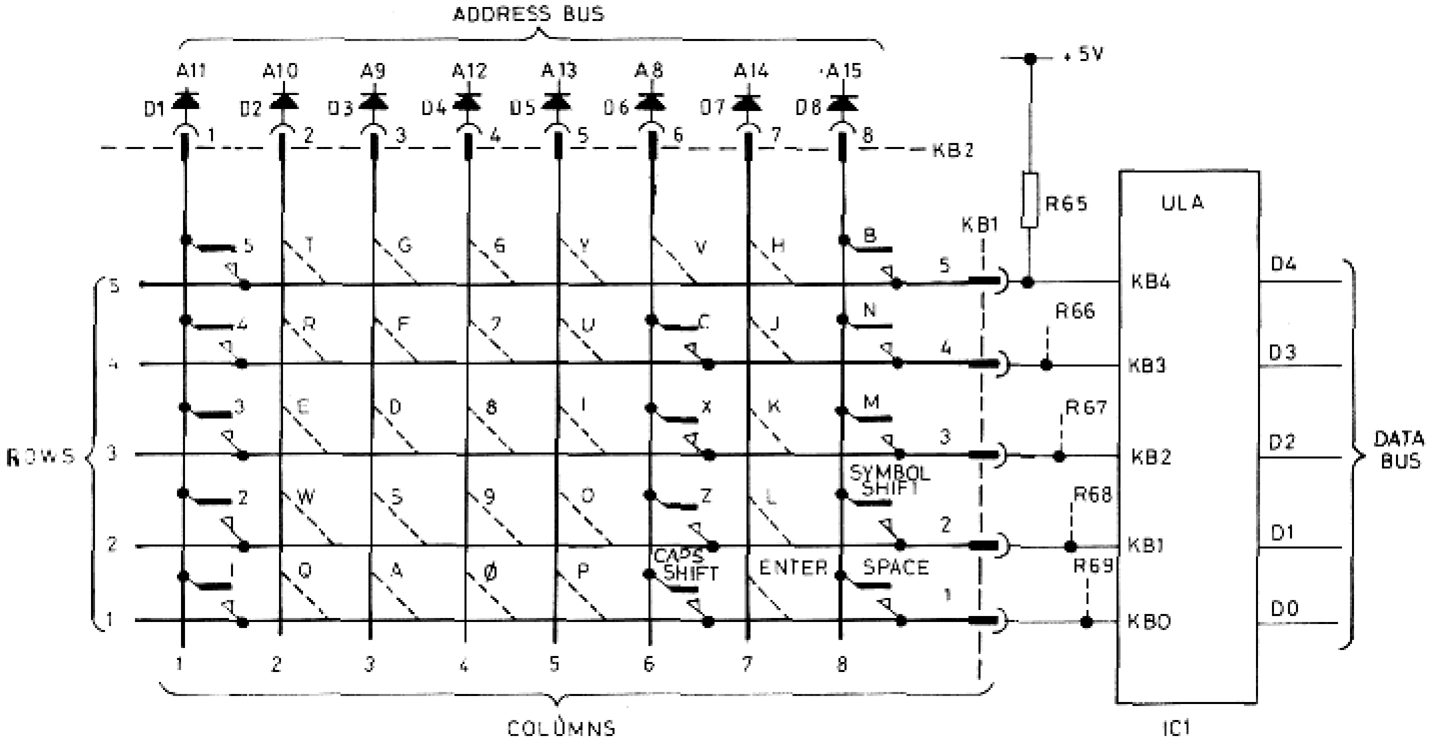
- Queries are fast, bypassing OS queues and application event queues.
- Despite that: Standard keyboards problematic for RT measurements!

Keyboard latencies: Keyboard encoder scan cycle



Keyboard organized as a matrix of columns (powered address lines) and rows (voltage sense lines). Keys attached to micro-switches that close a connection between a column and a row when key is depressed.

Keyboard latencies: Keyboard encoder scan cycle



Safes a lot of cables and connections (16+8 pins instead of 128 pins).
Keys in different columns get checked at different times ⇒ Timing bias. Scan cycle takes typically 8 – 20 msecs, depending on keyboard encoder.

Keyboard latencies: Summed up

Kb-Encoder: 5-20 ms

Debouncer: Up to20 ms

USB: Up to 10 ms

- Standard USB keyboard can add 30 - 50 ms latency, some of it as systematic bias!
- If you want to use a keyboard for RT measurements, use a special keyboard, e.g. DirectIN from Empirisoft: http://www.empirisoft.com/directinkb.aspx
- Another option: Response boxes that act as keyboards,e.g., fORP from http://www.curdes.com/usbforp.htm
- Mouse is more well-behaved if it can't generate movements.
- Still 10 ms timing noise due to USB polling.

What else is in the box? Getting an overview

- *help Psychtoolbox* – Introduction, overview of all categories of functions.
- *help <CategoryName>* - Help for a specific category of functions, list of all functions in that category with a one-line summary.
- *help <FunctionName>* - Detailed help for a specific function.
- The demos in the PsychDemos subfolder: *help PsychDemos*

- Need more info?
  - *help PsychDocumentation* – Miscellaneous documentation.
  - The Wiki: http://www.psychtoolbox.org
  - The user forum: http://tech.groups.yahoo.com/group/psychtoolbox

Useful Matlab helper functions by Category

- *PsychSignal* - Bandpass filtering, minima and maxima of matrices, scaling, RMSE, ...
- *Psychometric* – Psychometric function fitting: Logistic functions, Weibull functions, Normal functions, ROC curves...
- *PsychOptics* - Routines for calculating optical parameters...
- *PsychProbability* – Generate random numbers according to different distributions, fit distributions to data, random shuffling of vectors and matrices.
- *Quest* - Staircase procedure for optimal threshold estimation in a bayesian framework, (Watson & Pelli, 1983) and variants of it.
- *PsychFiles* - Convenient parsers and helper routines for reading and writing stimulus spec files or logging subject responses.
- *PsychColorimetric* - Routines for color space conversions, and for research on color perception...

Useful Matlab helper functions by Category

- *PsychGamma* - Determine gamma response curves of displays from measurements, generate gamma correction lookup tables...
- *PsychRects* - Helper functions for optimal arrangement of stimuli on the screen, e.g., centering images, scaling, arranging images along each other...
- *PsychOneliners* - Miscellaneous useful little helper routines, e.g., text formatting, text input from subject, information about your system...
- *PsychTests* - Test routines to assess correct operation of Psychtoolbox and of your system.

Lots more useful 3rd party stuff on the web, e.g., PsigniFit by Felix Wichmann and Jeremy Hill for Psychometric function fitting.

The Links-Section of the Wiki has some interesting pointers...

Support for Eyetrackers

- Eyelink-Toolbox: Support for the Eyelink gazetrackers Eyelink-1, Eylink-2 & Eyelink-1000 from SR-Research. Written and maintained by Frans Cornelissen, Enno Peters & John Palmer with contributions by Christopher Burns.
- iViewX-Toolbox: Support for the video based iView series of gazetrackers and the MRI compatible MeyeTrack tracker, both from SMI. Written and maintained by Frans Cornelissen.
  This toolbox is alpha-quality, but in a useable state.

Other interesting 3rd party Matlab toolkits

- The OptoTrak toolbox: Control the OptoTrak optic motion tracker via Matlab: (Written by Volker Franz, Uni Giessen) http://www.allpsych.unigiessen.de/vf/OptotrakToolbox/index.php

- HapticLibrary: Control diverse haptic force feedback devices from C++, C, Java, and Matlab. Supports a variety of devices from different vendors, e.g., all Phantom and Omni devices from Sensable, Delta and Omega from ForceDimension. Open Source (GPL), easily extendable via new driver plugins, OpenGL compatible coordinate systems: http://www.haptiklibrary.org/

- MatlabCentral is a large repository for free public domain Matlab code and Matlab extensions hosted by The MathWorks: http://www.matlabcentral.com

Input/Output support and communication

- Network communication via TCP/UDP/IP toolbox, written by Peter Rydesäter & Mitthögskolan Östersund.Included from Matlab-Central, modified for lower latency:
  - Transmit commands or network triggers with sub-millisecond delay on a local network.
  - Synchronize clocks of different computers in a multi-machine setup, e.g., use a LabView state system to control a stimulus computer which runs Matlab+PTB, or vice versa. (cfe. *Andreas Tolias)*
- Communication with arbitrary USB-HID devices on OS/X via PsychHID interface. Currently supported are Joysticks, mice, keyboards, fORP and the USB-1208FS box (8 A/D channels, 2 D/A channels, 16 DIO).
- Support for PR-650 colorimeter and EGI Netstation EEG system.
- Serial link control via PTB commands or Matlabs integrated serial control commands.
- Parallel port support not included, but multiple free solutions exist.

GNU/Octave as alternative runtime environment

- GNU/Octave is a free, open-source replacement for Matlab.
- Mostly - but not fully - compatible with Matlab:
  - Basic functionality comparable to Matlab 6.
  - Sometimes differences in support for toolboxes or their syntax.
  - Most PTB-Matlab scripts run unmodified, some need small tweaks.
- Less convenient to use:
  - No GUI, like Matlab in -nojvm mode.
  - Installation a bit harder.
- Psychtoolbox can use Octave 2.1.73 as runtime system instead of Matlab:
  - On GNU/Linux. (Often recompile required)
  - On MacOS/X for IntelMacs.
  - Support for MacOS/X on PowerPC possible on demand.
  - Not yet for Windows.
- Acceptance? More info on http://www.octave.org

Summary & Credits

- Current PTB-3 is a pretty useable construction site ;-)
- Especially strong points:
  - Visual stimulation.
  - Auditory stimulation.
  - Large user community.
- Biggest limitation: Good documentation & canned experiments.
- Other stuff you'd like to see? (Request page on Wiki or forum)
- Help if you can -> Contribute code or documentation.

Credits: (Waaaay incomplete)

Allen Ingling, Denis Pelli, David Brainard, Richard Murray, Christopher Broussard, Frans Cornelissen, Christopher Burns, Christopher Taylor, Finnegan Calabro, Keith Schneider, Maria McKinley, Michael Shadlen, Michael Silver, Florian Stendel, Kerstin Preuschoff, Cambridge Research Systems, Gergely Csibra, Quoc Vuong, Daniel Berger, Oguz Ahmet, Massimilliano di Luca, Patrick Minneault... ...You?

Questions?

The Wiki:

http://www.psychtoolbox.org

From there -> User forum...