

Introduction to Psychtoolbox-3

Mario Kleiner

(with some new demos contributed by Peter Scarfe)

ECVP 2013 edition - For Psychtoolbox v3.0.11 „Stadtmusikanten“ and later

With minor updates for Psychtoolbox v3.0.15

Trademarks...

- OpenGL and the OpenGL logo are registered trademarks of Silicon Graphics 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.

Menu:

- Intro: Motivation, supported platforms, license, etc.
- Visual stimulus presentation basics:
 - Basic drawing functions, coordinates, colors
 - Visual stimulus onset, timing and timestamping
 - Advanced drawing functions
 - Stimulus post processing and special display hardware
- Auditory stimulus presentation basics
- Response collection basics

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.
 - Considerable knowledge about underlying operating systems and (computer-)hardware needed to use effectively!
 - High-level canned experiment packages:
 - Easy to learn & use.
 - Considerable lack of flexibility for demanding tasks.
 - 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 and Octave with functions for precise and flexible stimulus presentation and response collection.

Psychtoolbox-3: What is it?

- A set of Matlab/Octave extensions to facilitate programming of psychophysics experiments in Mathworks proprietary Matlab(tm) or the free and open source GNU/Octave.
 - Compiled Plugins (written in C) for time critical operations and interfacing with graphics-, sound- and response collection hardware and operating system services.
 - A couple of hundred Matlab M-Files to simplify higher level tasks like data analysis, logging of responses to files, monitor calibration, ...
- Free open source software (FOSS), licensed mostly under the MIT license.
- Version 2 originally written for Apple MacOS by Denis Pelli and David Brainard, released around 1995. Later a more limited release for MS-Windows around 2000.
- Version 3, a complete rewrite of the low-level C layer and many high level functions:
 - Since 2004 for Apple OSX by Allen Ingling of the Pelli Lab at CNS - New York University
 - Since 2006 for MS-Windows & GNU/Linux by Mario Kleiner of the Buelthoff lab, MPI for Biological Cybernetics, Tuebingen, Germany.
- Git (hosted on GitHub) as code management system:
 - <https://github.com/Psychtoolbox-3/Psychtoolbox-3>
 - All changes are archived and documented.
 - Simplifies collaborative development.
 - Provides convenient software update mechanism to users via a Subversion frontend.
 - One new release about every 2 months, bugfixes usually faster.
- Over 43000 known unique installations and over 125000 downloads as of August 2013.

Supported computer and software platforms:

- Supported processors, operating systems and scripting environments:
 - GNU/Linux: (Highly recommended for demanding work!)
 - 32-Bit and 64-Bit Octave, 32-Bit and 64-Bit Matlab (\geq v7.4 - R2007a)
 - 32-Bit and 64-Bit Octave on non-Intel cpus, e.g., ARM or PowerPC
 - Linux packages provided by the NeuroDebian project (<http://neuro.debian.net>) for convenient installation on GNU/Debian based systems, e.g., Debian and Ubuntu Linux.
 - Apple/OSX:
 - 64-Bit Octave \geq v3.6 and 64-Bit Matlab (\geq R2010a) on Intel cpus with OSX version 10.6 „Snow Leopard“ or later.
 - MS/Windows:
 - 32-Bit and 64-Bit Matlab (\geq v7.4 - R2007a) on Intel cpus.
 - Windows XP or later, but Windows Vista or later not recommended for multi-display setups.

Graphics hardware requirements:

- Basic functionality available on any desktop 3D graphics hardware:
 - Basic OpenGL 1.2 support required. Needs hardware from 1998 or later.
 - 16 MB VRAM required.
- High speed and advanced features need recent graphics hardware!
 - OpenGL 2.1 or later. Needs hardware from around 2006 or later.
 - > 128 MB VRAM, more is better!
 - Simplifies a lot of complex stimulus programming tasks.
 - Allows for high precision framebuffers, e.g., 32 bpc floating point.
- If you can choose, go for the latest NVidia or AMD graphics cards.
- Latest generations of Intel HD graphics cards good enough for not too demanding tasks, at least on Linux and OSX.
- Avoid Hybrid graphics / „Optimus“ Laptops!

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 current Matlab.
 - Sometimes differences in support for toolboxes or their syntax.
 - Most PTB-Matlab scripts run unmodified, some need small tweaks.
- A bit less convenient to use for beginners:
 - No GUI, like Matlab in -nojvm mode. But useable GUI almost finished (ETA 2013).
 - „classdef“ Object oriented programming support not yet, but almost (ETA 2013-2014)
 - Limitations in GUI programming, Java support, plotting wrt. Matlab.
- Psychtoolbox can use Octave 3.2 and later as runtime system instead of Matlab:
 - On GNU/Linux. Get it via your distro package management.
 - 64-Bit Octave 3.6 On OSX for Intel based Apple Macs. From HomeBrew, Fink etc.
 - Currently not on MS-Windows, mostly due to lack of significant user interest.
- Stats as of September 2013: At least 20.3% of all Linux users and 2.9% of all OSX users use Octave instead of Matlab. So far no known issues related to use of Octave.
- More info on <http://www.octave.org>

Stuff that helps:

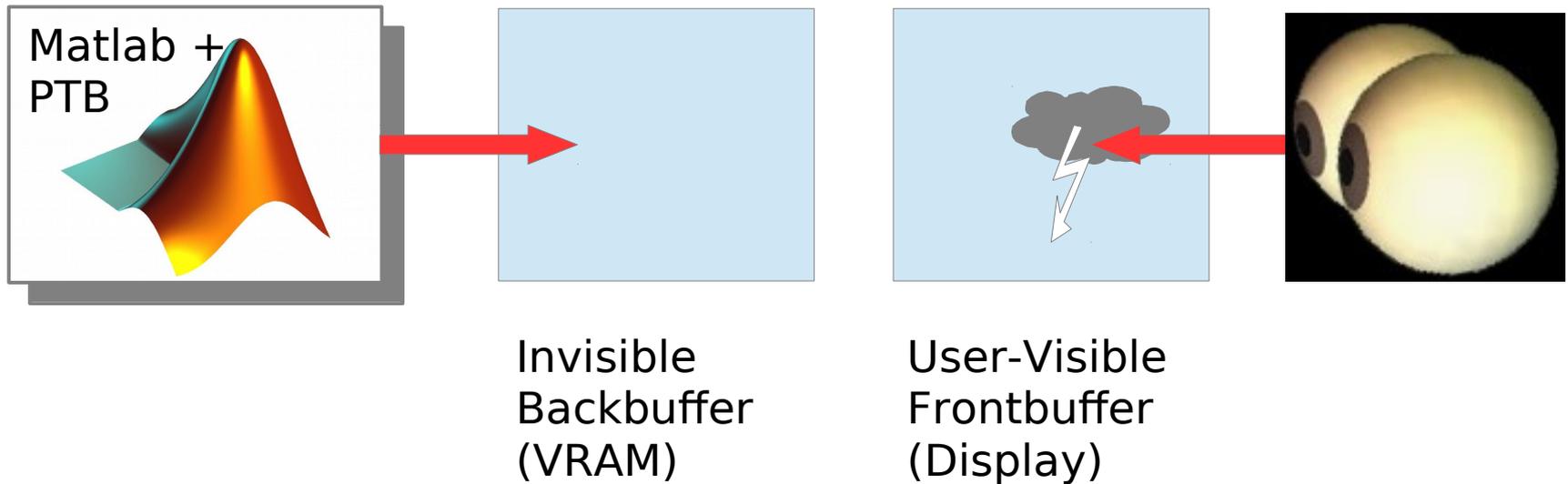
- *PsychDebugWindowConfiguration* is your friend on single display setups! Shows stimulus display half-transparent, so you can use the GUI and editor/debugger and see your stimulus at the same time.
- CTRL + C aborts a running script. Sometimes you need to ALT+TAB to the proper command window or click it with the mouse beforehand.
- help *functionname* for help on M-Files and high level help:
 - E.g., *help Psychtoolbox*, *help PsychDemos*, *help Screen*, ...
- For low-level functions which have many subfunctions:
 - *Function subfunctionname ?*
 - E.g., *Screen OpenWindow ?* - Prints help on Screen subfunction 'OpenWindow'.

Basic drawing and display:

HelloWorldDemo.m:

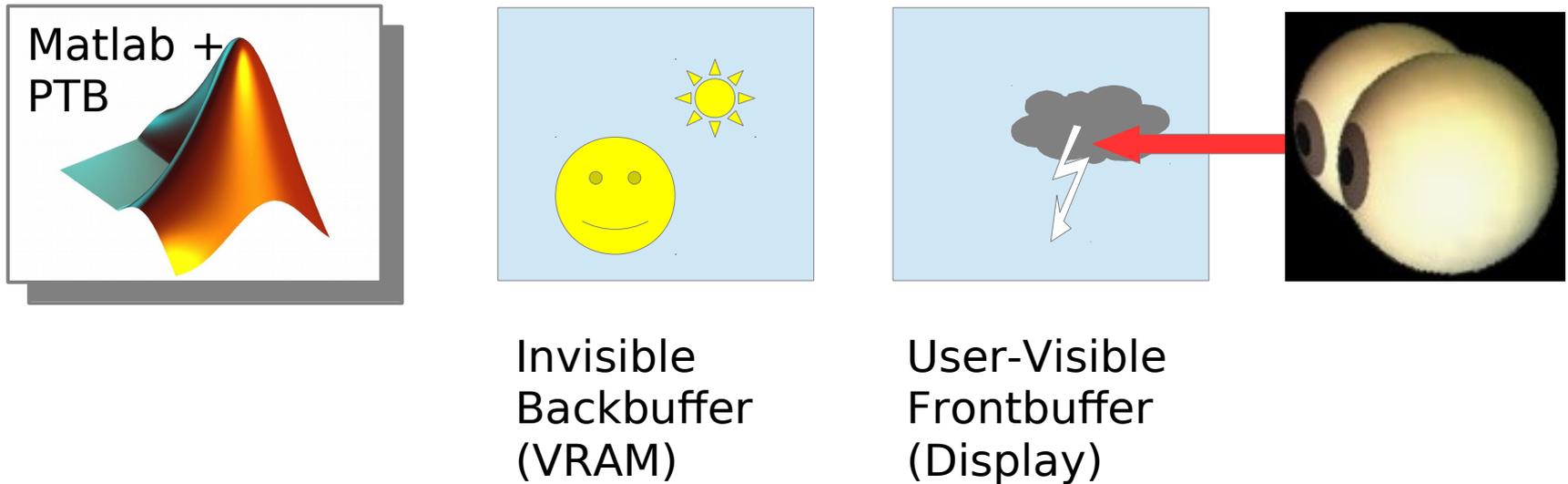
1. Open a window on the display.
2. Draw the text „Hello World!“
3. Show it.
4. Wait for a key press.
5. Close the window and clean up after yourself.

PTB-3: Double buffered drawing model - Concept



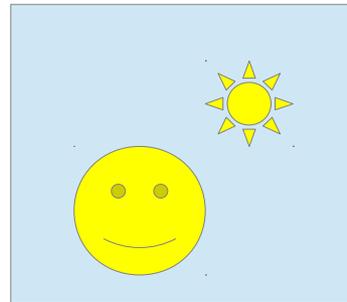
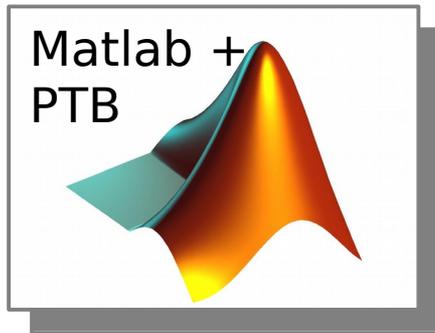
Subject sees cloudy picture on the display screen (==Frontbuffer). Matlab issues `Screen()` drawing commands to Psychtoolbox, Psychtoolbox translates these commands into OpenGL rendering commands and submits them to the graphics accelerator hardware. Alternatively, one can submit low level OpenGL 3D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

PTB-3: Double buffered drawing model - Concept

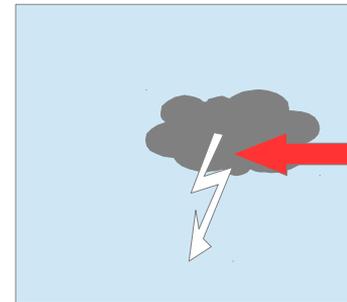


Subject sees cloudy picture on the display screen (==Frontbuffer).
Matlab issues `Screen()` drawing commands.
Graphics hardware draws into backbuffer, processing the OpenGL commands in the background while Matlab and Psychtoolbox are able to do other stuff in parallel, e.g., keyboard queries, sound output...

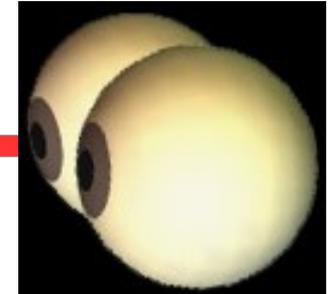
PTB-3: Double buffered drawing model - Concept



Invisible
Backbuffer
(VRAM)

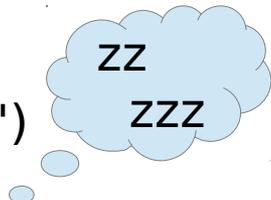


User-Visible
Frontbuffer
(Display)



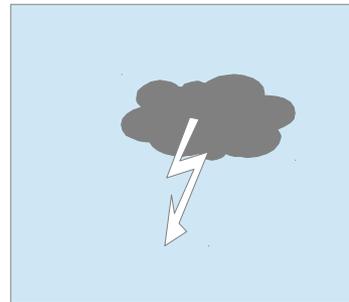
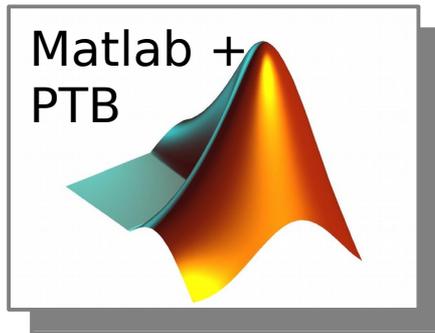
Stimulus ready for presentation, Matlab issues Screen('Flip') command.

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.

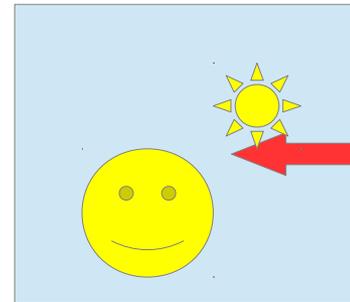


ZZ
ZZZ

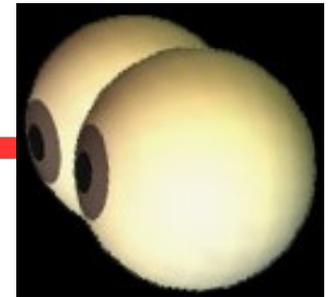
PTB-3: Double buffered drawing model - Concept



Invisible
Backbuffer
(VRAM)

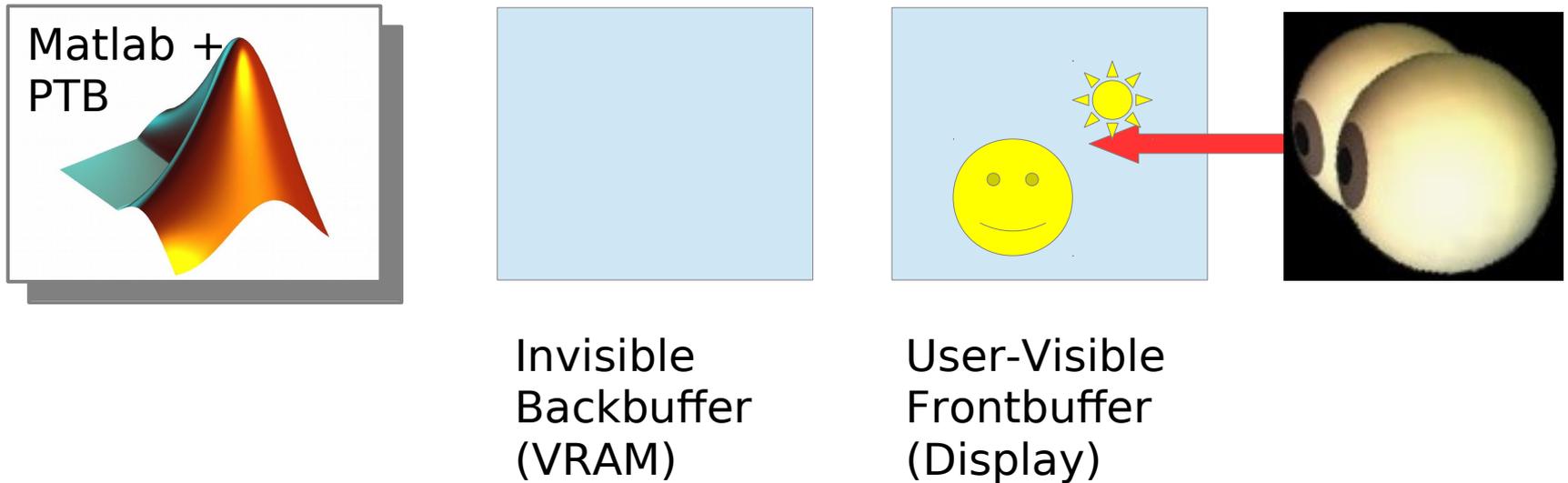


User-Visible
Frontbuffer
(Display)



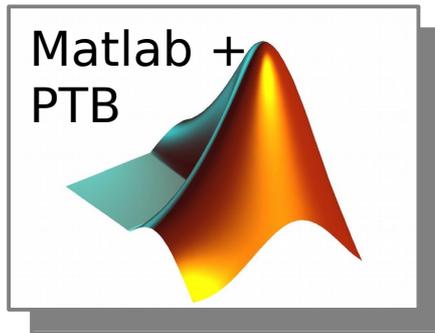
Start of vertical blank: Hardware exchanges Front- and Backbuffer.
The low-level circuitry of the graphics hardware ensures
microsecond accurate sync of bufferswap with the vertical blank!
Matlab gets woken up...

PTB-3: Double buffered drawing model - Concept

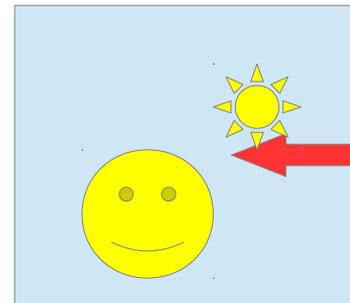


Subject perceives a tear-free stimulus update, PTB takes stimulus onset timestamp and clears the backbuffer for next stimulus drawing cycle.

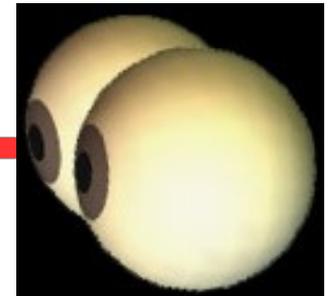
PTB-3: Double buffered drawing model - Concept



Invisible
Backbuffer
(VRAM)



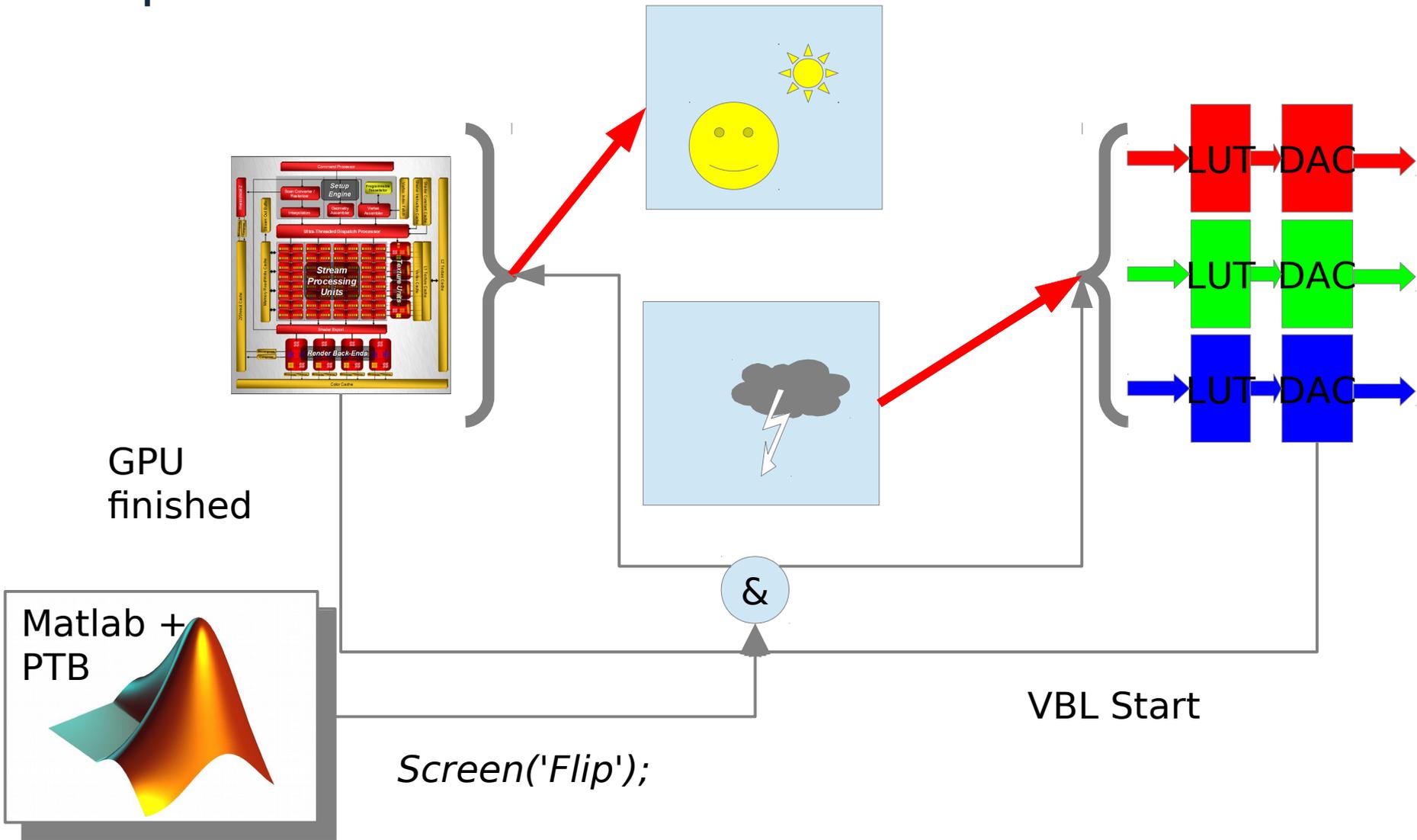
User-Visible
Frontbuffer
(Display)



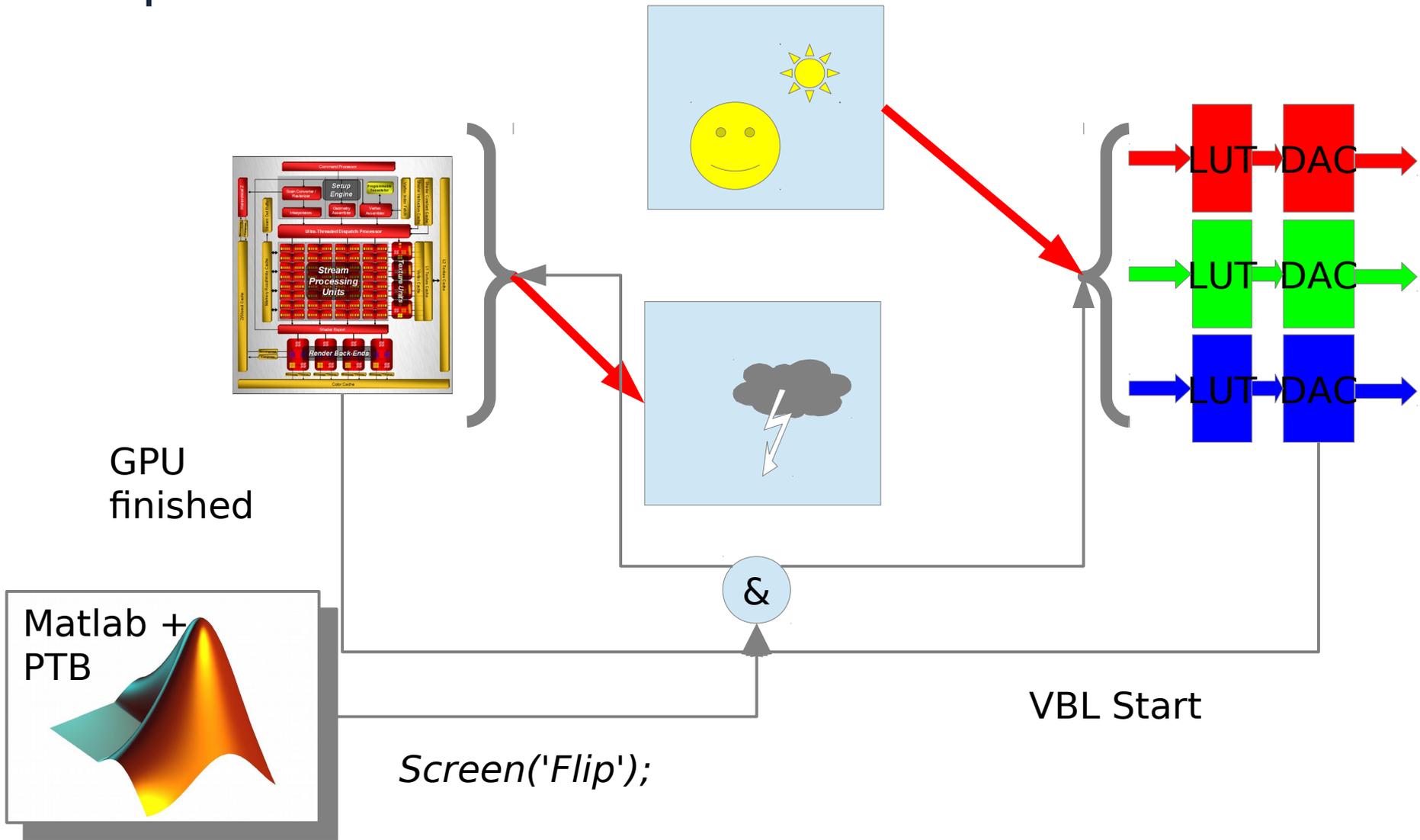
Subject sees new sunny picture on the display screen
(==Frontbuffer).

Backbuffer is cleared to background color for next stimulus.
Ready for next display redraw cycle...

Double buffered drawing model - Implementation:



Double buffered drawing model - Implementation:



Basic drawing: Coordinates

Coordinates:

Origin = top-left corner of a window at $[X,Y] = [0,0]$, x-Axis pointing right, y-Axis pointing down.

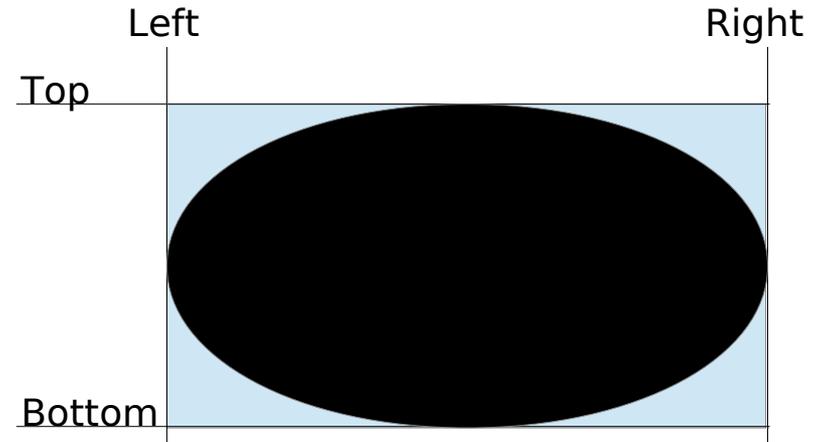
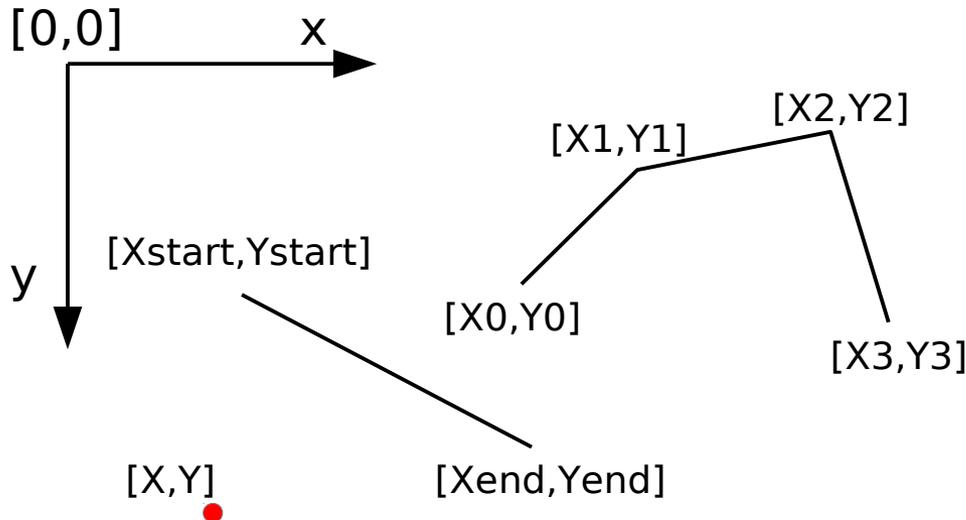
Position and size unit is pixels → Resolution dependent!

Point coordinates = $[X, Y]$

Line coordinates are point pairs = $[Xstart, Ystart], [Xend, Yend]$

Polygons are sequences of points $[X0,Y0], [X1,Y1], \dots, [Xn, Yn]$

Objects with some area, e.g., Rectangles, Ovals, Arcs, textures, are defined by their „bounding rectangle“ - the smallest rectangle which encloses the area occupied by the object. The bounding rect is defined by its $rect = [left, top, right, bottom]$ boundaries („LeTteRBox“)



Basic drawing: Color specification.

Colors:

Grayscale can be specified as a single number, e.g., 0.75

Real colors are usually specified as [Red, Green, Blue] triplets, defining the requested intensity of red, green and blue in a color.

Traditionally color and grayscale intensity values specified as integral numbers ranging from 0 to 255 in 256 steps. This allows for 256 grayscale intensity levels, or $256 * 256 * 256 = 16.8$ million shades of color.

E.g., [128, 128, 0] for a yellow of roughly 50% maximum intensity.

→ Most traditional Psychtoolbox scripts and (still) most of our demos use this.

Better: Use a normalized color range of floating point values from 0.0 to 1.0, with 0.0 == Minimum intensity, 1.0 == Maximum intensity.

E.g., 0.5 = 50% intensity gray, [0.75, 0.75, 0.1] = 75% red + 75% green + 10% blue for yellow with a sprinkle of blue.

→ Color specifications independent of color resolution of display device.

→ Thinking and coding in a normalized range usually more natural.

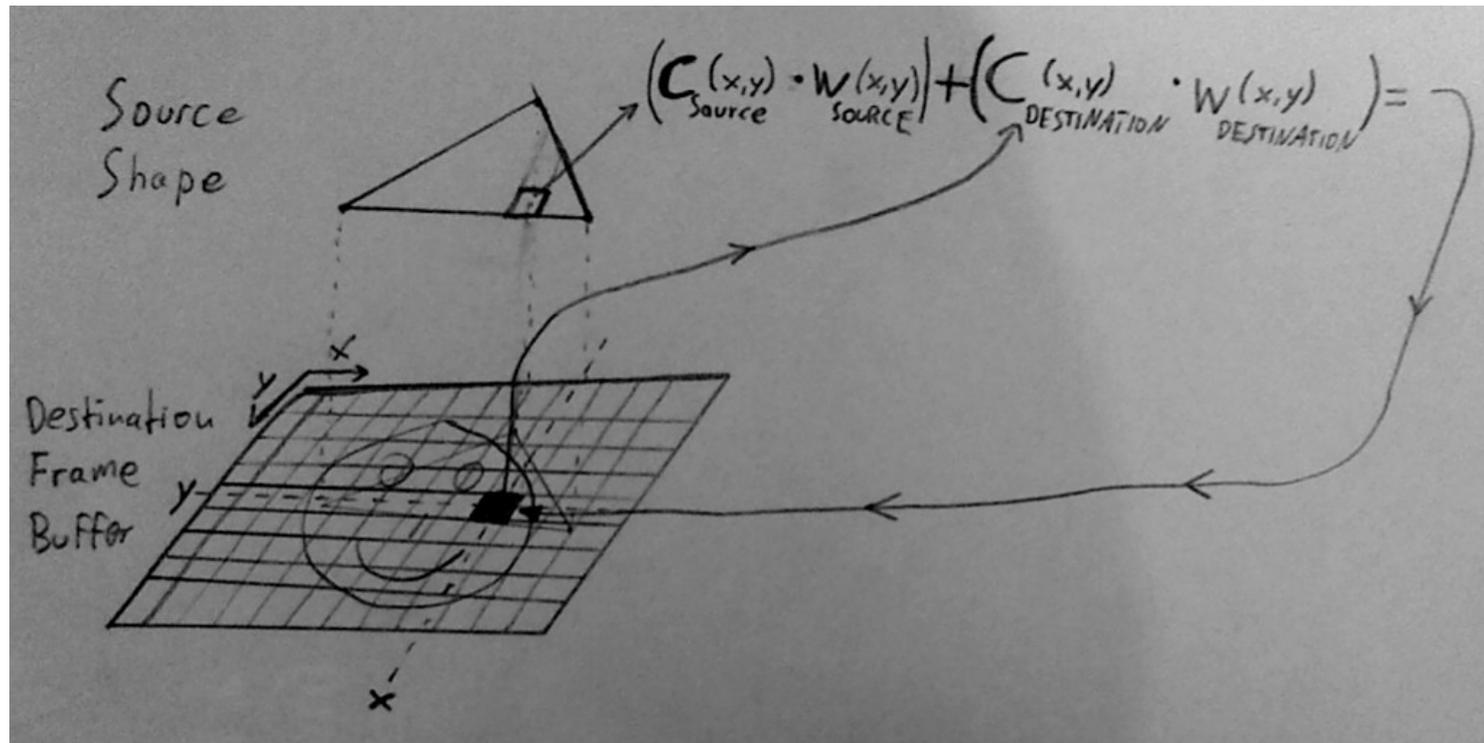
→ *PsychDefaultSetup(2);* at the top of your script enables this new and better style.

Basic drawing: Alpha blending

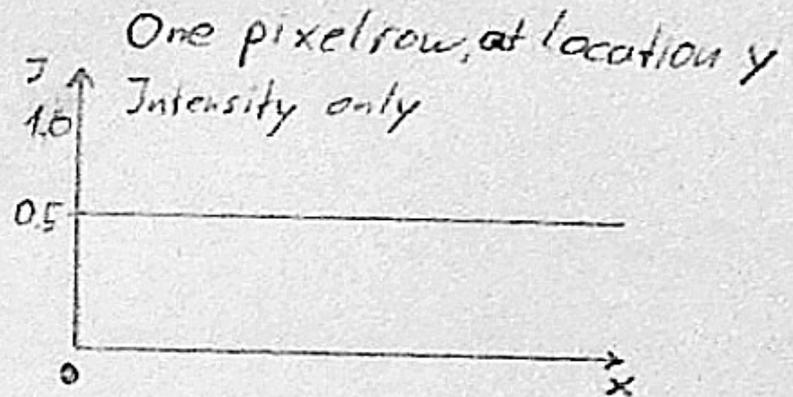
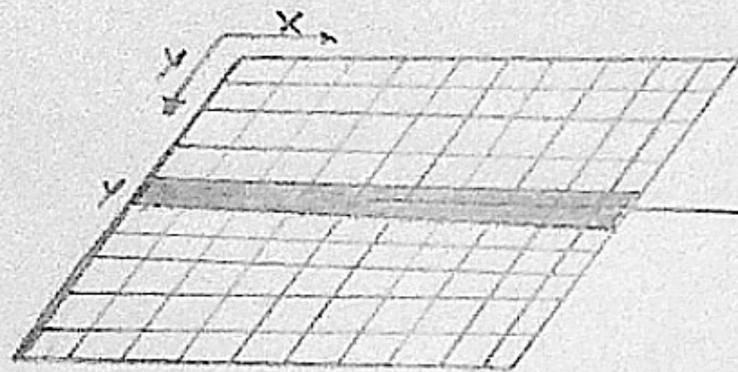
$$C_{destination} = (C_{source} \cdot W_{source}) + (C_{destination} \cdot W_{destination})$$

Whenever a pixel of a shape is drawn into a framebuffer location:

- New pixel color is a weighted sum of previous pixel color at that location and pixel color of “incoming” new fragment from drawn shape.
- *Screen('Blendfunction', window, Wsource, Wdestination)*; allows to select the way that these weights are selected for blending.

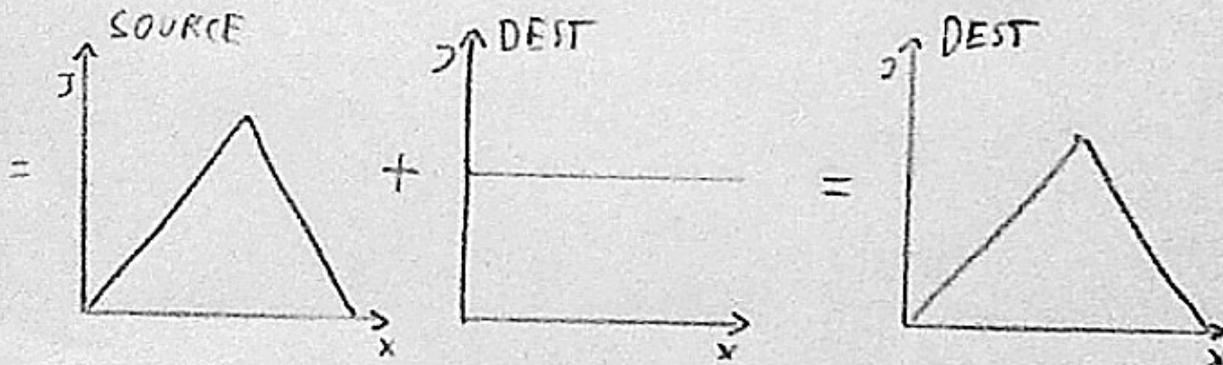


Basic drawing: Alpha blending



Screen ('Blend Function', with `GL_ONE`, `GL_ZERO`),

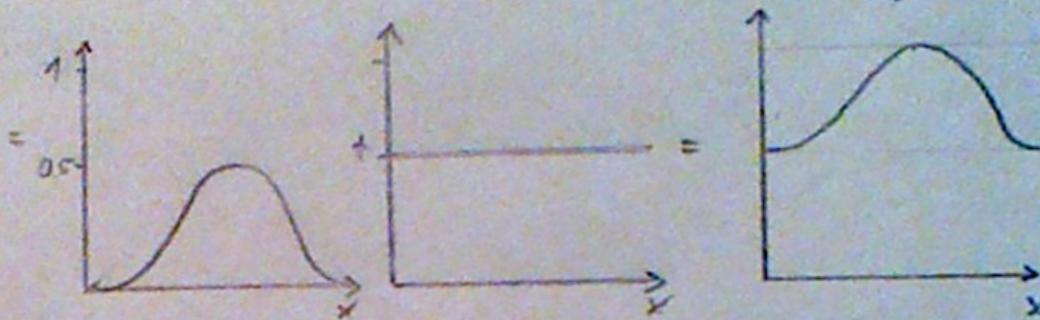
$$\Rightarrow C_{\text{DEST}} = (C_{\text{SOURCE}} \cdot 1) + (C_{\text{DEST}} \cdot 0) = C_{\text{SOURCE}}$$



Basic drawing: Alpha blending

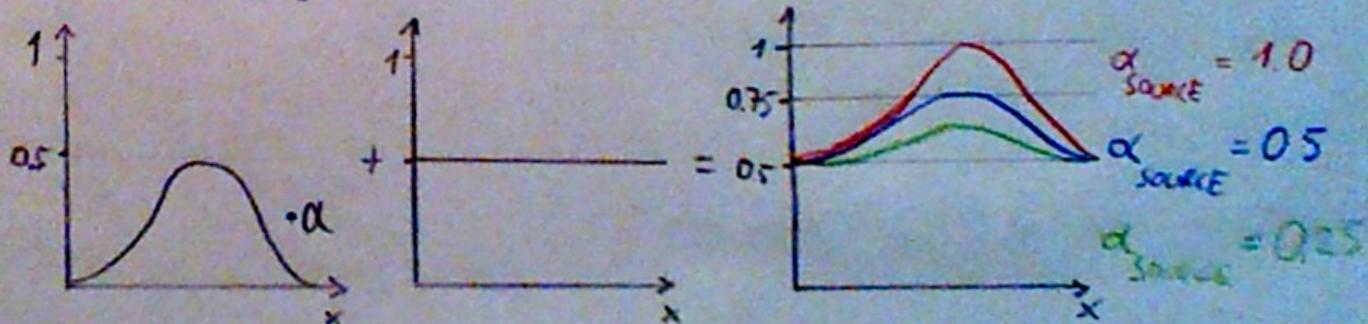
Screen ('Blendfunction', win, GL_ONE, GL_ONE)

$$\Rightarrow C_{DEST} = (C_{SOURCE} \cdot 1) + (C_{DEST} \cdot 1) = C_{SOURCE} + C_{DEST}$$



Screen ('Blendfunction', win, GL_SRC_ALPHA, GL_ONE);

$$\Rightarrow C_{DEST} = (C_{SOURCE} \cdot \alpha_{SOURCE}) + (C_{DEST} \cdot 1) = \alpha_{SOURCE} \cdot C_{SOURCE} + C_{DEST}$$

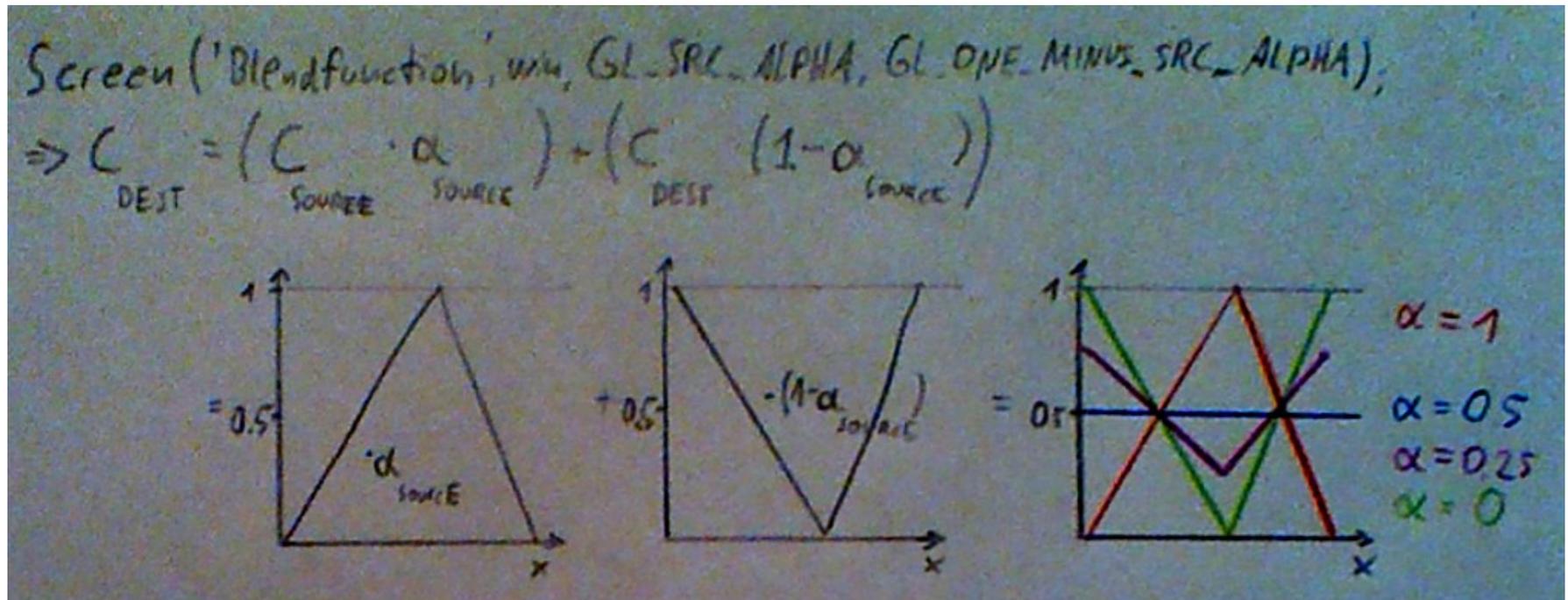


Basic drawing: Alpha blending

The blending weight W_{source} is usually called alpha value. It can be specified as a 4th „color“ component when specifying colors for drawing of shapes:

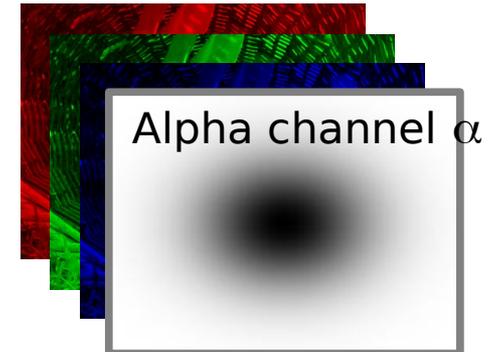
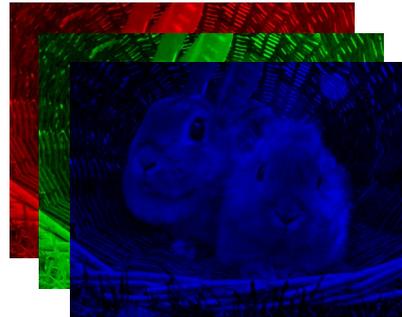
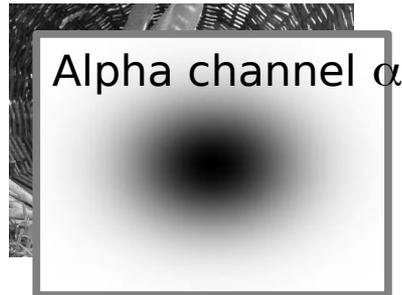
[red, green, blue] → [red, green, blue, alpha]

But: gray → [gray, gray, gray, alpha], not [gray, alpha] !!



OpenGL Textures as image representation

- Textures are containers for 2D pixel images
- Each texture is a *width x height* 2D matrix of image pixels, each with up to 4 color components or color channels per pixel location (x,y)
- Textures are created by Psychtoolbox from image matrices you pass, or from other sources, e.g., movie files or video cameras. They are managed by the graphics hardware for high performance. You only get a handle (= a unique name) for them, no direct access to their content.



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:

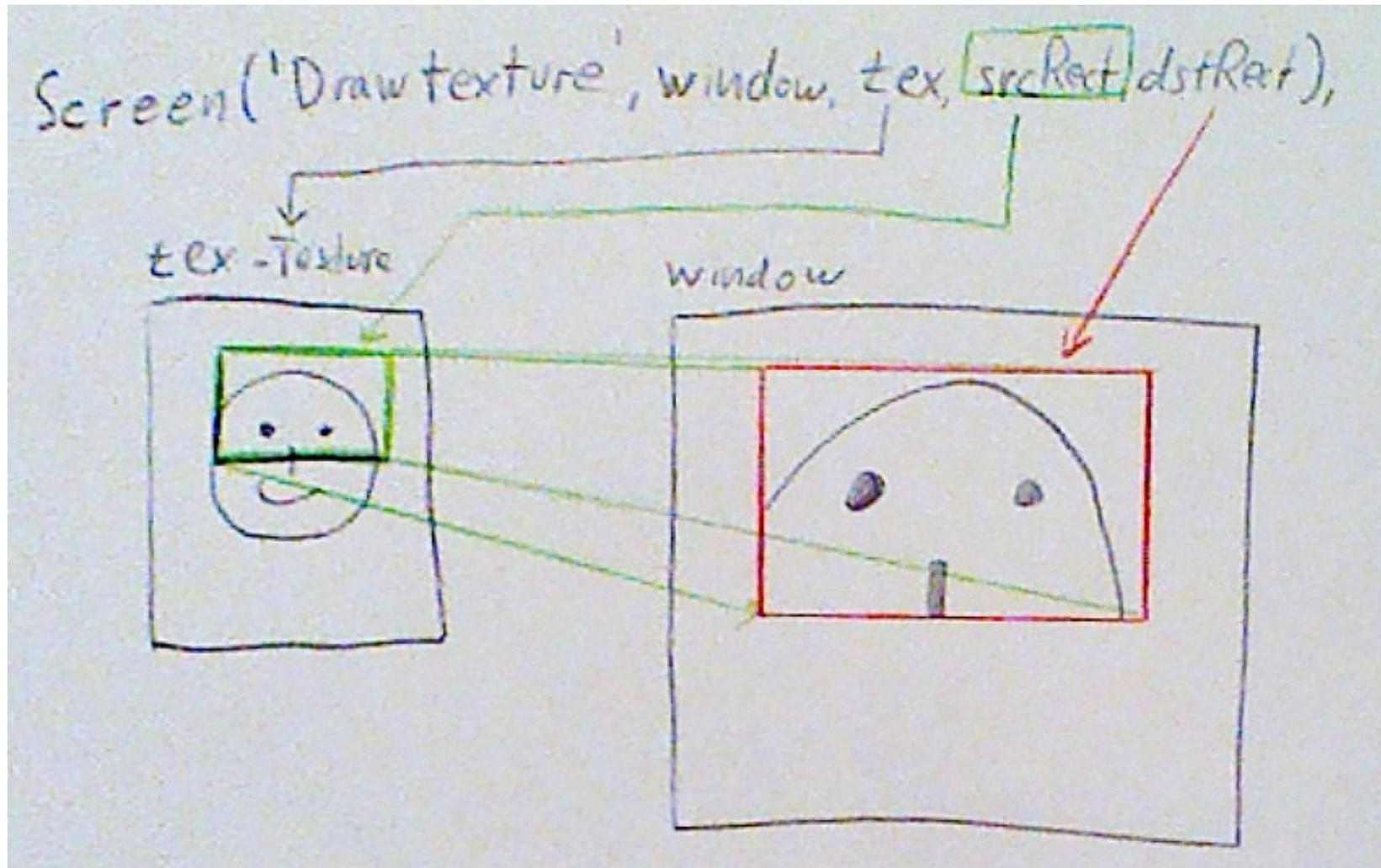
Creation:

```
bluop = CreateGLOperator(window);  
Add2DConvolutionToGLOperator(bluop, fspecial('gaussian', 13, 5.5));
```

Application:

```
blurredtex = Screen('TransformTexture', intex, bluop);
```

Texture drawing: srcRect and dstRect



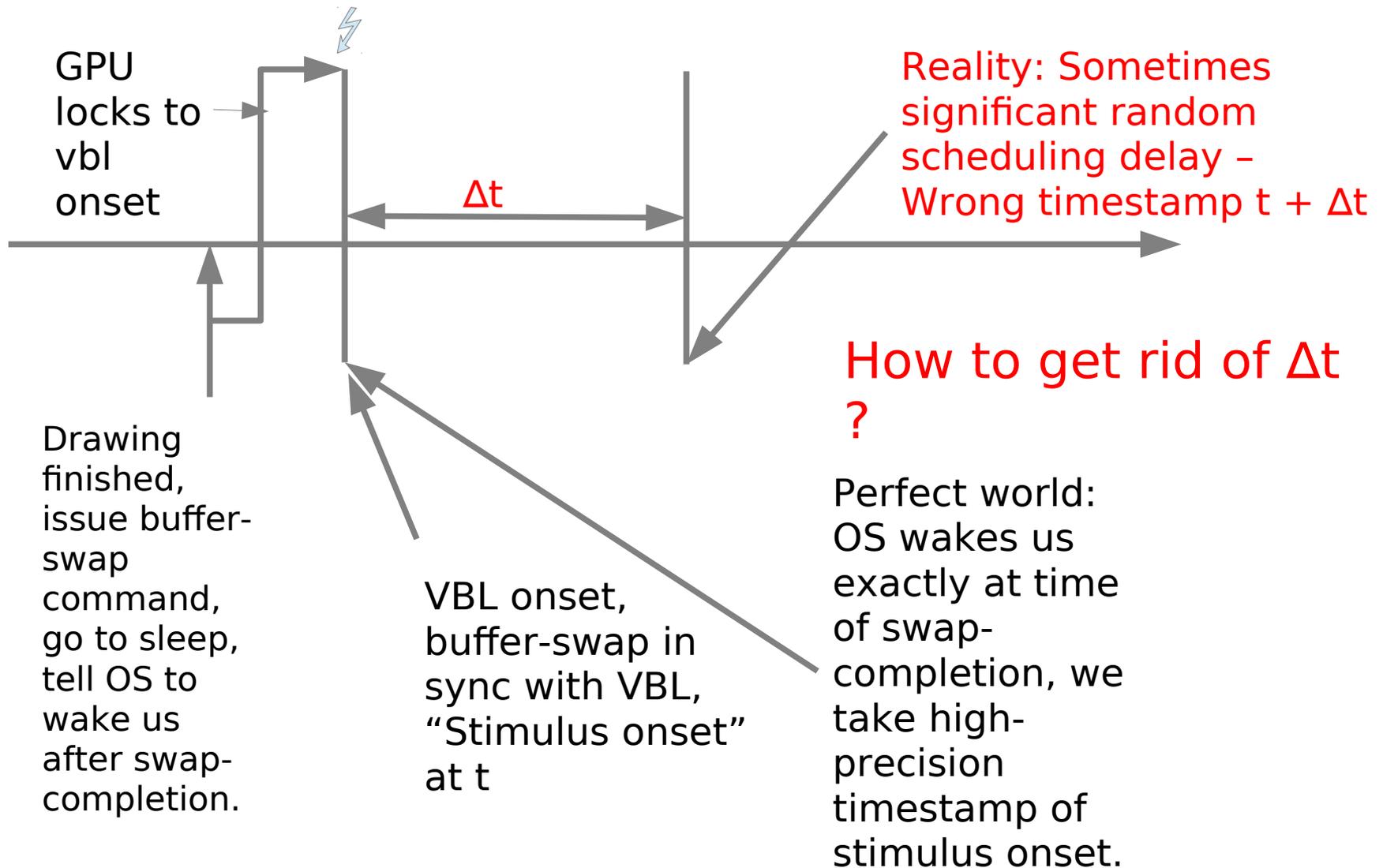
Procedural textures:

- 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.
- Expanding rings, Gabor patches, Sine gratings, Perlin noise patches.

Example GLSL fragment shader: “Expanding rings shader”

```
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: Stimulus onset timestamps



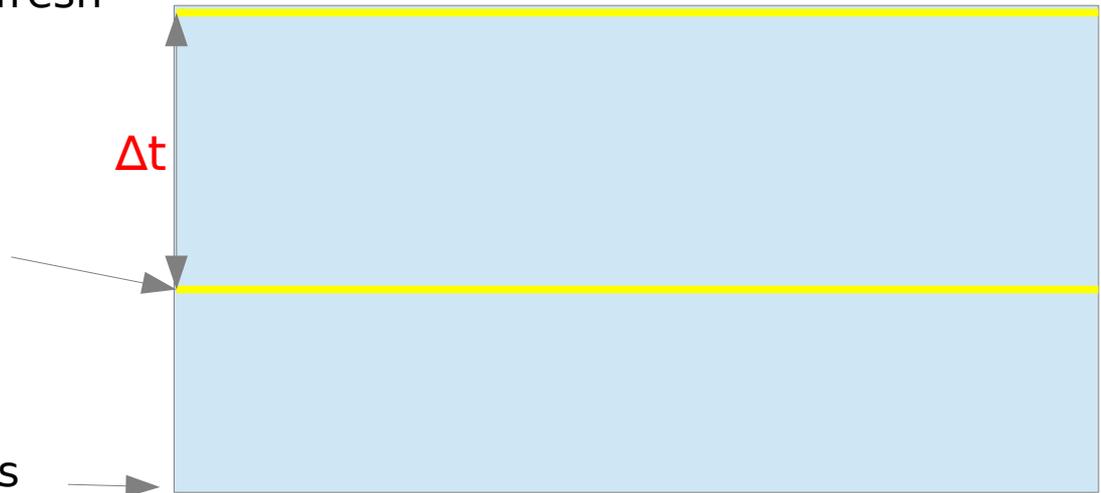
Screen: Beamposition as phase-locked display clock

- ⇒ $\Delta t = (\text{beampos}/\text{height}) * \text{refresh}$
- ⇒ $\text{tonset} = \text{tmeasured} - \Delta t$

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

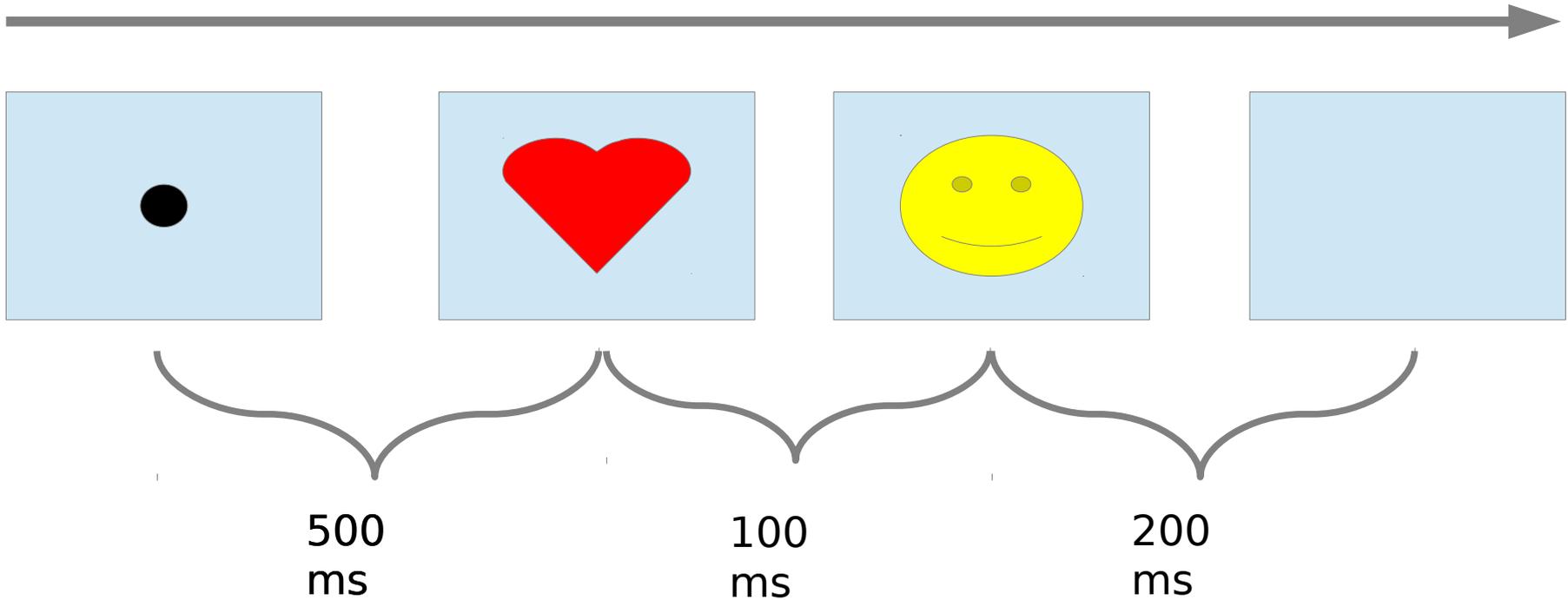
Height = 1064 scanlines (1024 visible + 40 VBL)

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



- Reduces jitter / noise in timestamps to less than 0.1 ms on modern systems.
- Accurate timestamps without special hardware!
- Supported on MS-Windows, MacOS/X and Linux.
- MacOS/X increasingly buggy. Needs *PsychtoolboxKernelDriver* installed.
- See ECVP 2010 poster in file *ECVP2010Poster_VisualTimingPrecision.pdf* in the Psychtoolbox/PsychDocumentation subfolder for benchmarks and more info.

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



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, primedImage);
 4. % Show prime exactly 500 msec 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 msec after onset of prime image:
ttarget_onset = Screen('Flip', win, tprime_onset + 0.100 - slack);
 7. % Show target exactly for 200 msec, 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...

Screen('AsyncFlipBegin') / Screen('AsyncFlipEnd')

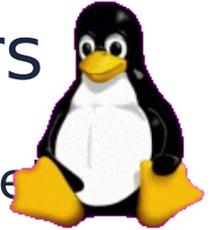
1. Schedule bufferswap in the background.
2. Execute and timestamps asynchronously on a separate thread.

Allows to overlap stimulus onset with other activities.

Allows to overlap onset of current stimulus and drawing of next (future) stimulus.

See AsyncFlipTest.m and MultiWindowLockStepTest.m

Linux as a future solution for power-users



- New Linux DRI2 graphics stack quickly evolving into a well designed system for precise stimulus display.
- E.g., <http://dri.freedesktop.org/wiki/CompositeSwap>
- Only operating system with built-in OS level support of OpenML for exact scheduling and time-stamping of graphics and video for some graphics cards.
- Ongoing plan: Integrate Psychtoolbox high-precision time-stamping into the OS kernel. Don't worry about timing woes anymore :-)
Worry in a more „interesting“ way ;-)
- Done for all Intel and AMD GPU's with open-source graphics drivers.
- In progress for NVidia desktop GPU's with open-source nouveau graphics driver.
- Implementations for some embedded GPU's with open-source graphics drivers.

Fast 2D batch drawing and parallelism:

Some commands are optimized for fast batch-drawing:

Distribute Matlab->PTB call overhead of approx. 10 μ s 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);
```

GPU image processing

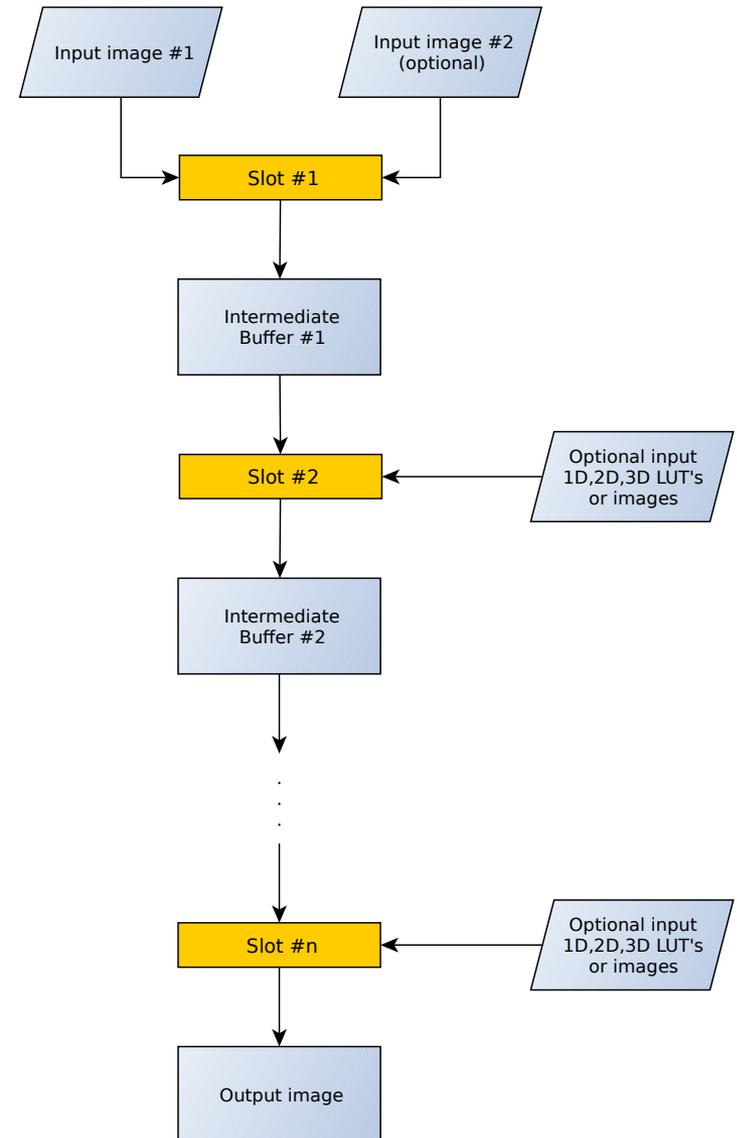
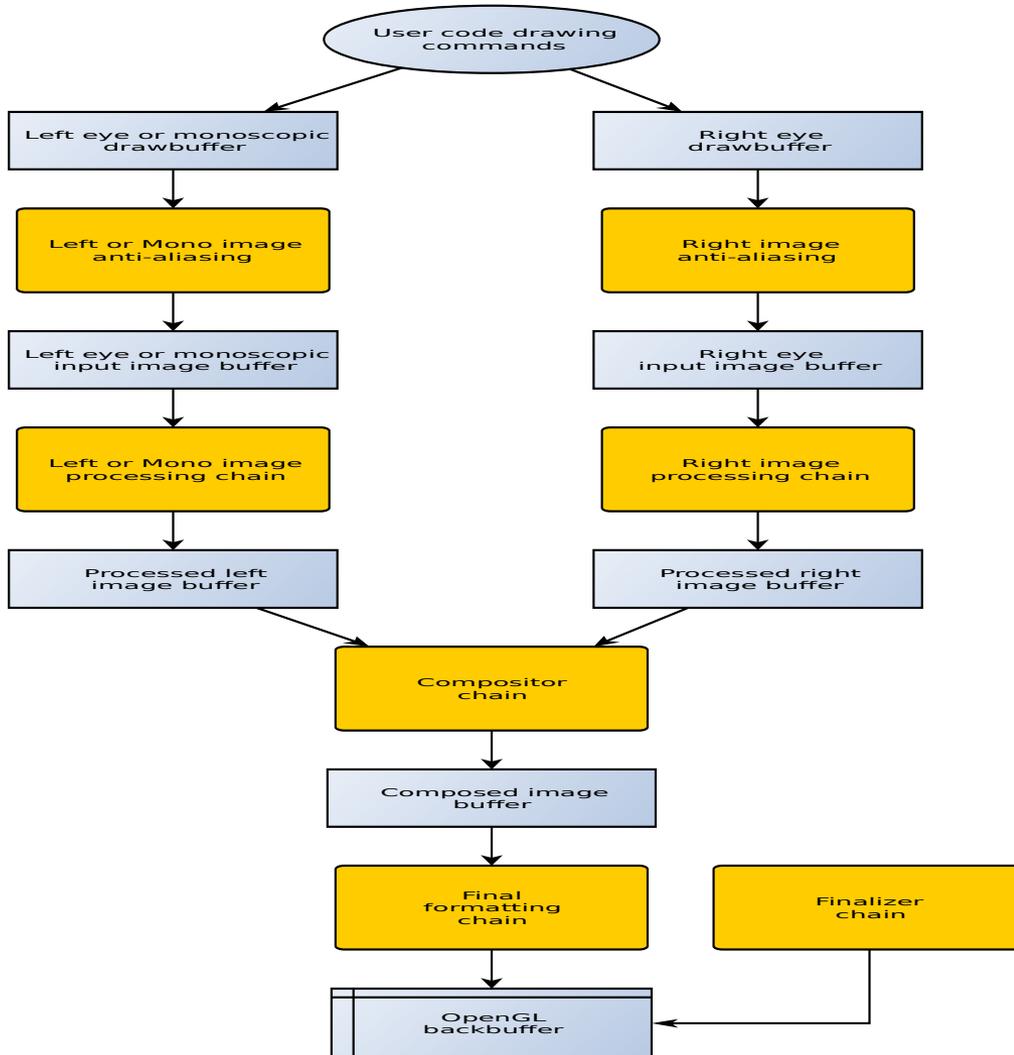
Example GLSL fragment shader:

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;
}
```

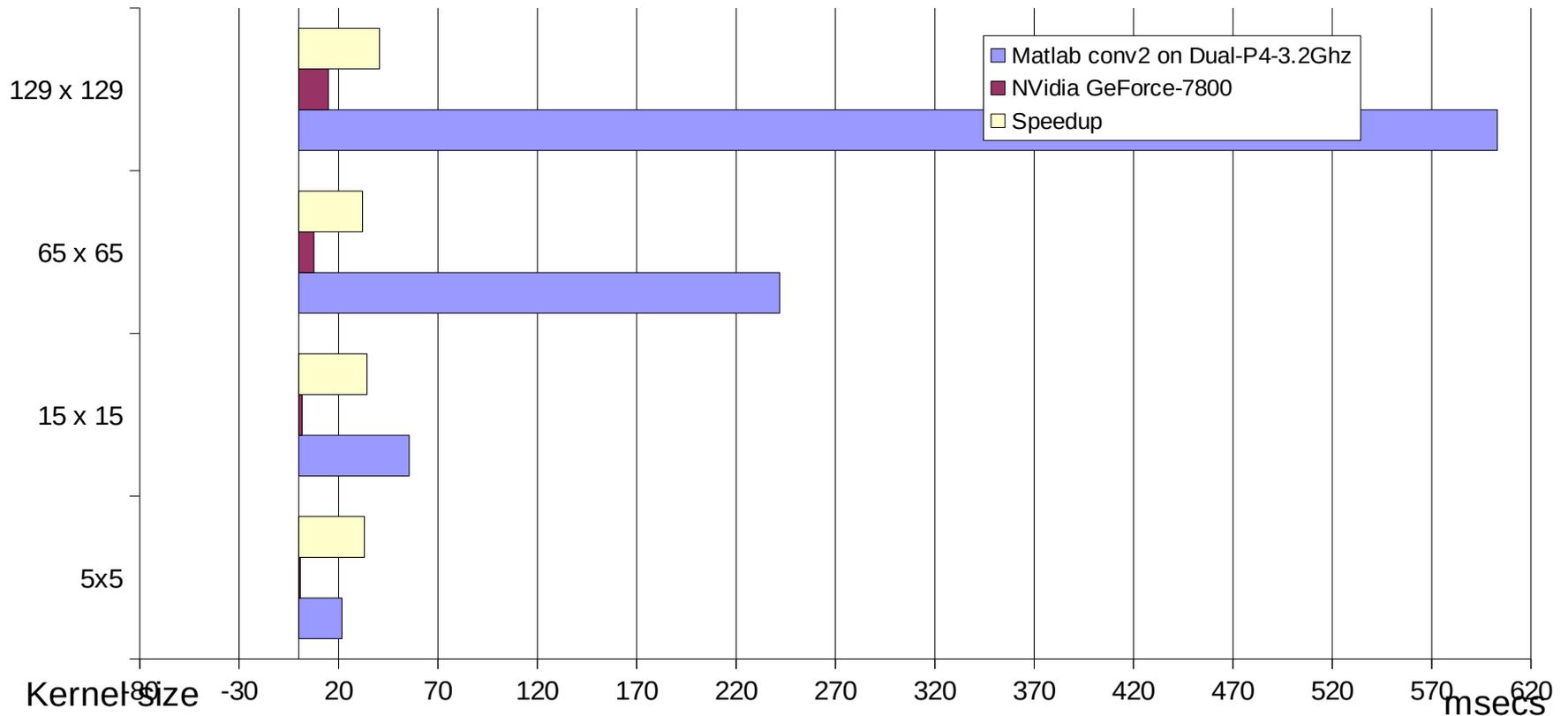
- Executed once for each output pixel.
- Can read from many input pixels + 1D/2D/3D lookup tables + constants + builtin variables (e.g., position) to compute color of output pixel.
- Many standard operations supported in C-like language, for scalars, vectors and matrices: max, min, clamping, comparison, 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.

GPU image processing:



Example: Speedup for 2D image convolution

Execution time for 1 channel convolution with 512x512 image:



Typical speedup vs. Matlab $\approx n * 33$ for separable convolution.

Test it yourself: *ConvolutionKernelTest.m*

Needs recent hardware!!!

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, bgcolor,);*
- Currently implemented:
 - High precision framebuffers 16 bpc and 32 bpc floating point.
 - Display geometry correction: Cfe. *DisplayUndistortionBezier*, *DisplayUndistortionBVL*, *DisplayUndistortionSphere/Cylinder*
 - Rescaling of stimulus images for „panel fitting“.
 - Horizontal-/Vertical mirroring, cloning of images.
 - Stereoscopic display modes for different methods and devices.
 - Drivers for BrightSide-Technologies 16 bpc HDR display, Cambridge Research Systems 14 bpc Bits++ and Bits#, Vpixmap Inc. 16 bpc DataPixx/ViewPixx/ProPixx, different Video attenuators, PseudoGrey / Bit-Stealing and native 10 bpc framebuffers of latest graphics cards.

Stereo display support for easy binocular stimulation:

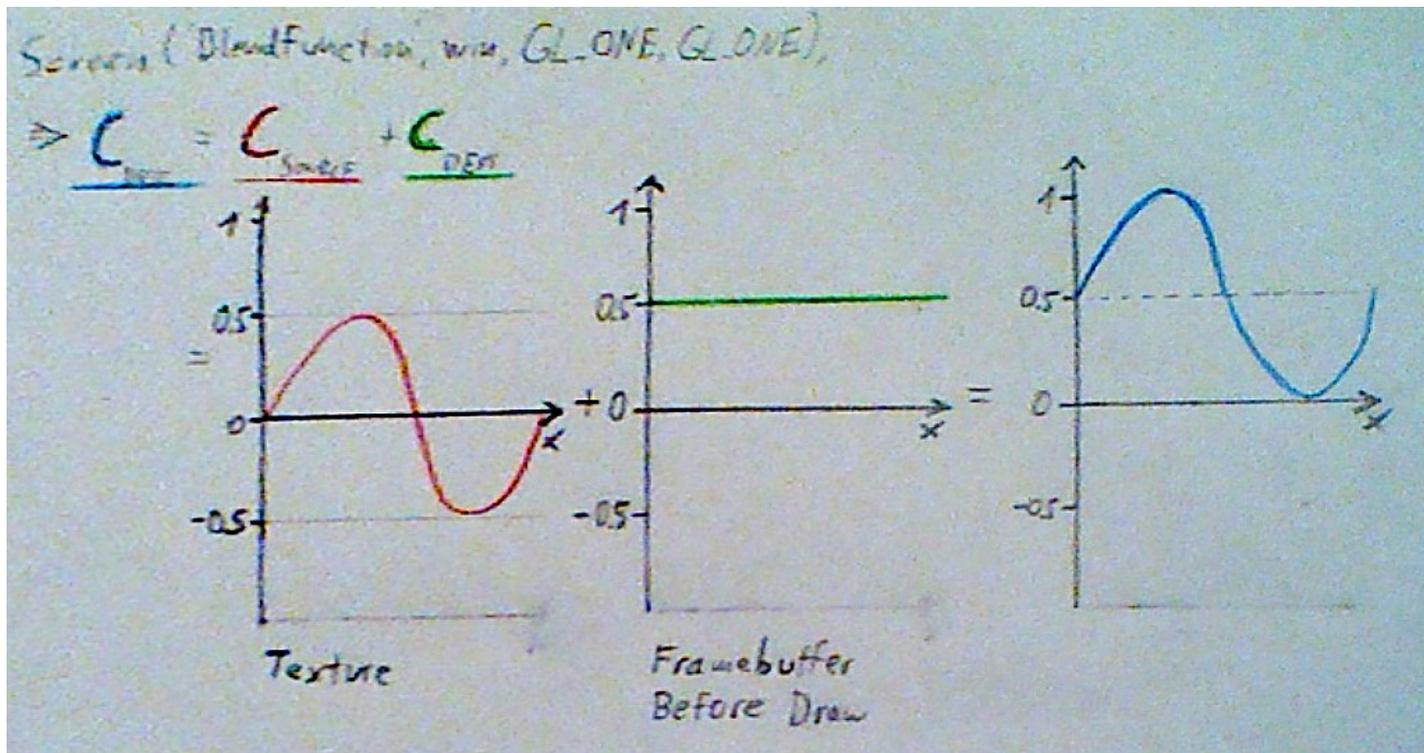
- One optional flag in *PsychImaging('OpenWindow', ..., stereoMode)*; to select between different modes:
 - Monoscopic.
 - Quad-buffered frame sequential stereo for shutter glasses on professional gfx-hardware.
 - Frame-Sequential stereo now also on fast consumer cards. YMMV...
 - Dual-display stereo for stereo goggles, stereoscopes, polarized stereo...
 - Anaglyph Red/Green and Red/Blue stereo with controllable gains and mode of presentation: *SetAnaglyphStereoParameters(...)*;
 - Line- and Column-interleaved stereo modes.
 - Adding new custom stereo presentation modes for new display hardware is easy.
- 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: *ImagingStereoDemo*, *StereoDemo*, *DrawDots3DDemo*, *MorphDemo*, *StereoViewer*, *ImagingStereoMoviePlayer*, ...

Floating point textures and framebuffers:

- Use of floating point framebuffers and textures allows to store pixel data with 32 bit non-linear precision ~ effective 23 bits of linear precision in typical displayable range.
- Allow use of negative color values for definition of stimulus shapes and intermediate results.
- `PsychImaging('AddTask','General','FloatingPoint32Bit');`
- Useful example: Linear superposition of contrast stimuli. Cfe. *AdditiveBlendingForLinearSuperpositionTutorial, GarboriumDemo.*



More stuff for GPU image processing pipeline

- Available as GLSL shader plugins and built-in functions:
 - Color to gray conversion, video deinterlacers.
 - 2D/3D CLUT based color conversion. Algorithmic conversions/mappings easy.
 - Gamma correction.
 - Per pixel Brightness/Gamma/Gain correction for displays.
 - Image pyramids for fast blurring and similar operations, e.g., for gaze-contingent displays. Cfe. *BlurredMipmapDemo*.
 - In progress: GPUmat CUDA interface. Cfe. *GPUFFTMoviePlaybackDemo*, „*help GPGPUDemos*“
- Look at „*help PsychImaging*“ for all available functionality!
- Future:
 - Other things you'd like to see?
 - Improved OpenCL / CUDA interface.

Movie playback with GStreamer



- With/without sound, looped/singleshot, at different speeds and backward. Frame accurate seeking is also supported for some movie formats.
- Simultaneous playback of multiple movies possible on fast hardware.
- Multi-Threaded decoding for higher performance. More supported video formats.
- 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.
- Cfe. *SimpleMovieDemo, PlayMoviesDemo, PlayDualMoviesDemo, ... help MovieDemos*

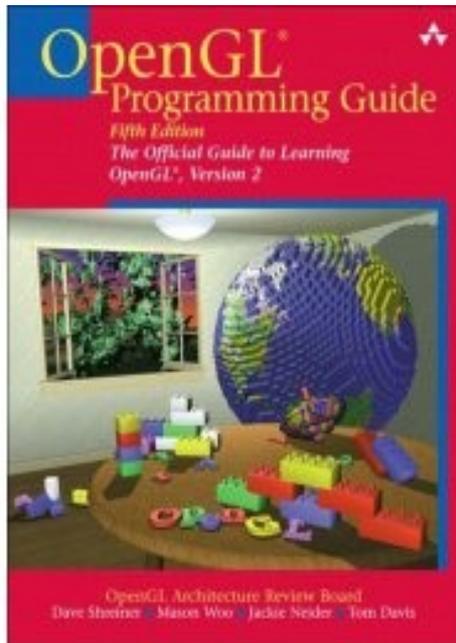
Screen - Video capture and recording

- Video capture and recording supported on standard hardware by use of GStreamer:
 - Tested with DV cameras, USB Webcams and IIDC Firewire cameras. Works very well on Linux, well on Windows, and somewhat ok on OSX.
 - Provides moderate framerates and timing accuracy, e.g., 30 - 120 fps.
 - Limited control of attached cameras, highly dependent on camera and operating system!
- Excellent video capture support for IIDC Firewire machine vision cameras on Linux and good support on OSX by internal use of FOSS libdc1394 library:
 - Number of simultaneously controlled cameras only limited by bus bandwidth.
 - 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.
- *help PsychVideoCapture, VideoCaptureDemo, ImagingVideoCaptureDemo, GPUFFTVideoCaptureDemo, ...*

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, Smartphones and tablets).
 - Easy to learn and use 3D state machine.
 - Provides standardized core functionality + vendor specific extensions.
 - New features of gfx-hardware usually supported *way ahead* in time of Direct3D!
- *MOGL* - Matlab interface to OpenGL (co-developed with Prof. Richard F. Murray, York University, Canada).
- *MOGL* Simple to use if you know how to do OpenGL in C!

Screen: 3D graphics with OpenGL for Matlab



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 in C:
<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 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*).
 - Certain structure from motion point-light stimuli (see *FDFDemo*).
- Also possible to intermix with compiled C OpenGL code:
 - Use Matlab's / Octave's *mex* command to compile C OpenGL code into mex file 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.

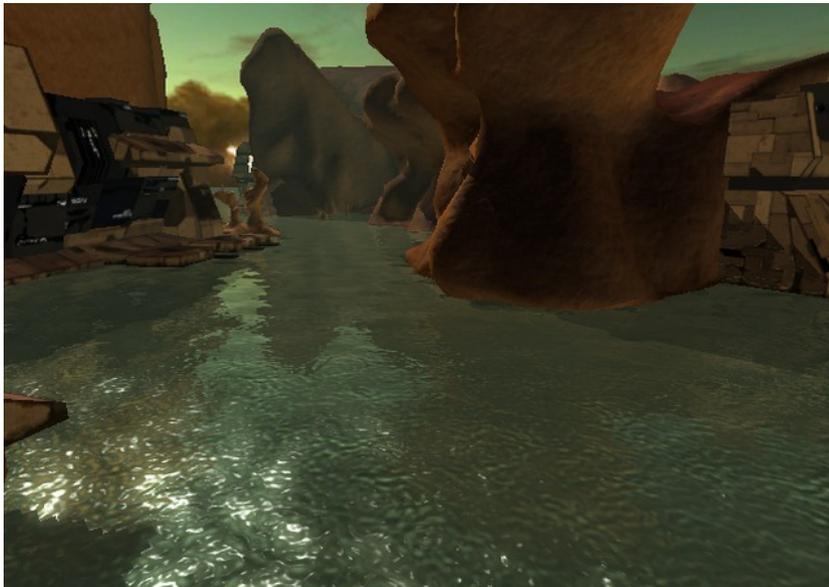
Horde3D

Overview

„Horde3D is a small open source 3D rendering engine. It is written in an effort to create a graphics engine that offers the stunning visual effects expected in next-generation games while at the same time being as lightweight and conceptually clean as possible. Horde3D has a simple and intuitive interface accessible from virtually any programming language and is particularly suitable for rendering large crowds of animated characters in next-generation quality.“

- <http://www.horde3d.org/>
- Horde3D is developed and funded by University of Augsburg. License EPL (Eclipse Public License).
- Psychtoolbox offers a simple wrapper around Horde3D.
- Wrapper not in official standard distribution, beta status, but works fairly well. Downloadable under MIT/BSD license from:
<https://github.com/kleinerM/Horde3DForPsychtoolbox>
- Happy customers so far:
 - Andreas Schindler & Andreas Bartels: VR-Environment for spatial cognition experiments.
 - Martin Giese et al.: Human character and crowd animation.
 - Stephan de la Rosa: Moven → Avatar animation for social interaction experiments.
 - You?

Horde3D

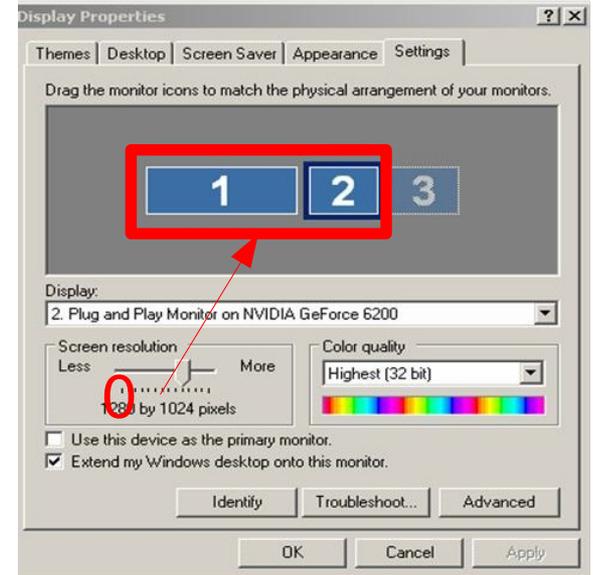
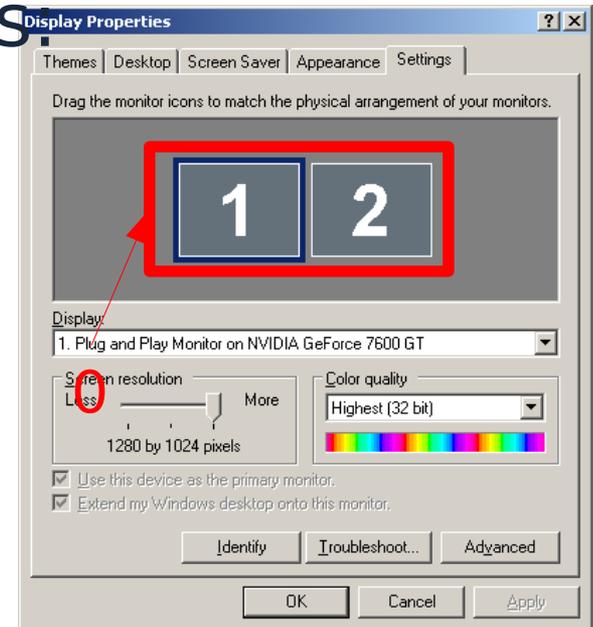


Support for Multidisplay setups

% 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, ...
 - Linux: 0 = X-Screen 0, 1 = X-Screen 1...



Support for Multidisplay setups:

- Only use multi-display if you really need it. (Performance impact)
- Use 1 dual-head graphics card for binocular stimulation, not 2 single-head cards. Ditto for multiple stimulation displays.
- 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.
- AVOID Windows Vista, Windows-7, Windows-8, ... for multi-display stimulation. Presentation timing and performance can be unreliable.
- Windows-8: No way do disable DWM desktop compositor. Multi-Display performance unclear atm., but potentially even worse than Windows-7.

Low latency, precisely timed sound with PsychPortAudio

- Utilizing 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 Linux, MacOS/X, and on MS-Windows.
 - Precisely timed sound on Linux, MacOS/X, and on MS-Windows.
 - Sound schedules ~ Flexible and precise Playlists for sound.
 - Sound mixing of multiple virtual sound cards into a physical sound card.
- Successfully tested on many systems, e.g.:
 - Linux & MacOS/X & Windows-10, Intel MacBookPro / MacPro with onboard sound.
 - Linux on different PC's and Laptops with onboard sound.
 - Windows, M-Audio Delta multi-channel card.
 - Windows, Creative Soundblaster X-Fi Xtreme Music (Erik Flister, UCSD).
 - Windows, Creative Soundblaster Audigy-2 ZS multi-channel (Virginie van Wassenhove, Caltech)
- Should work well on most Linux hardware, all OS/X Apple hardware and some Windows hardware.

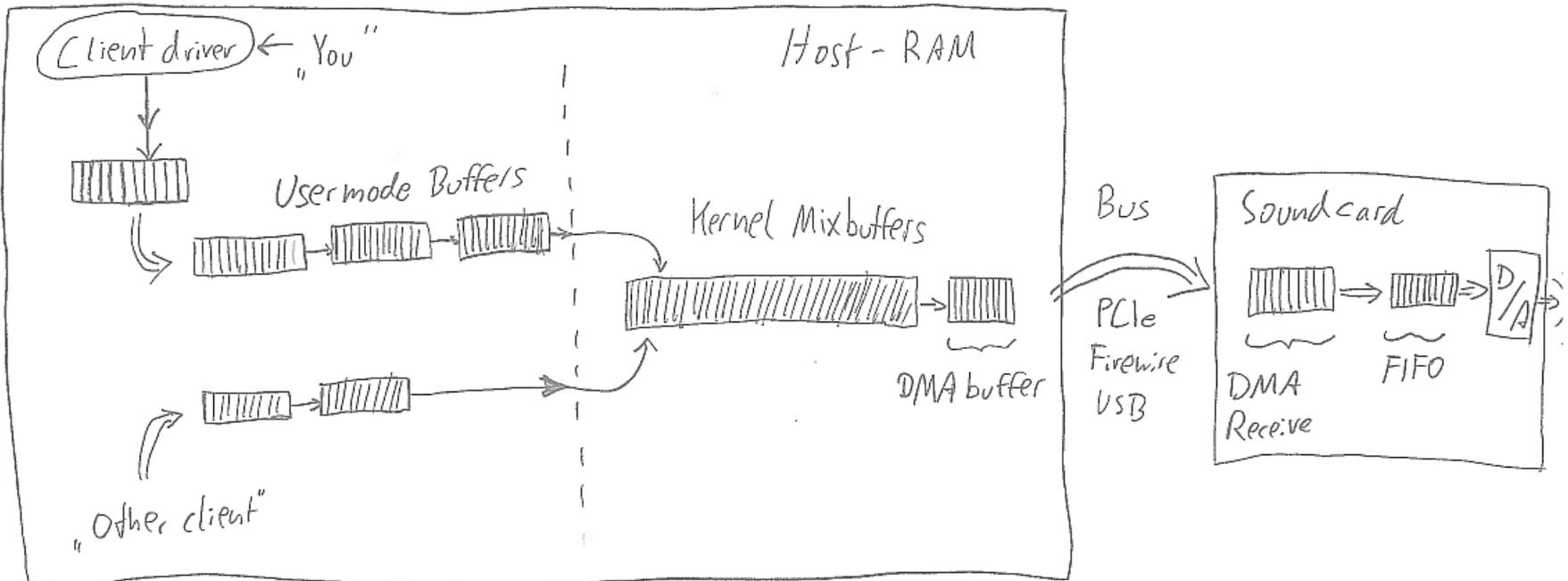
Audio buffering → Latency

Various FIFO (First in first out) buffers between your code and actual sound hardware, all causing audio latency.

Variable buffer size and buffer configuration → Variable latency!

Dependent on many possible factors: Sound hardware & device driver, operating system, software used, sampling rate, active audio effects, other running sound clients (system notification sounds, media players, web browsers etc.), system processor and bus speed, system load.

Latency can change from session to session, or within trials!

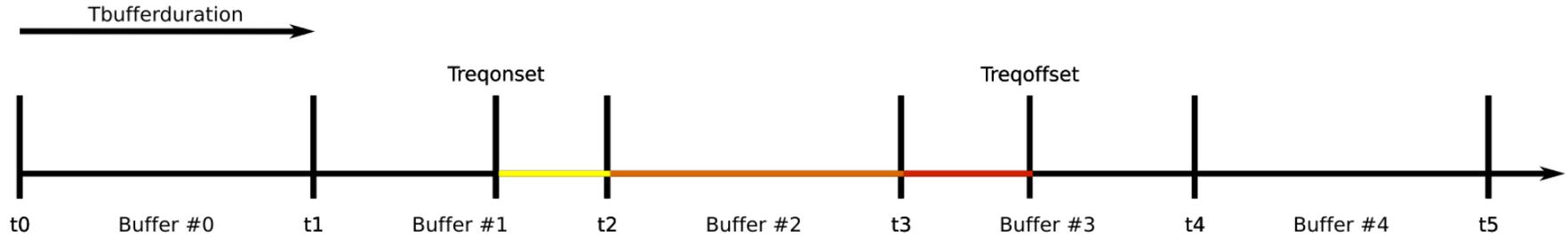


Low latency, timed sound with PsychPortAudio

- PsychPortAudio can automatically compensate for variable system latencies to provide high timing precision with off the shelf sound hardware.
- 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 Linux and OSX and Windows-10.

Audio scheduling: „A peek under the hood“

Timeline of audio output at sound card connector, split by hardware into segments covered by audio fragment buffers,



Sequence of PsychPortAudio's successive callback function invocations. Callback # i computes content of Buffer # i , according to output timeline:
 (Note: The i 'th callback gets executed many milliseconds before the predicted onset time " t_i " of the i 'th buffer to account for system latency)

- | | | | | |
|--|--|--|---|---|
| Callback #0:
$t_0 + T_{bufferduration} < T_{reqonset}$
=> Too early
=> Playback not yet started
=> Zero fill with silence | Callback #1:
$[t_1 ; t_1 + T_{bufferduration}]$
intersected by $T_{reqonset}$
=> Zero-fill until $SnReqOnset$
=> Start playback
=> Fill remainder with audio | Callback #2:
$T_{reqonset} < t_2$ and
$t_2 + T_{bufferduration} < T_{reqoffset}$
=> Continued Playback
=> Fill with audio samples | Callback #3:
$[t_3 ; t_3 + T_{bufferduration}]$
intersects $T_{reqoffset}$
=> Finish Playback
=> Audio until $SnReqOffset$
=> Zero fill remainder | Callback #4:
$T_{reqoffset} < t_4$
=> Too late
=> Playback stopped
=> Zero fill with silence |
|--|--|--|---|---|



█ = Zero values output = Silence
█ = Audio data output = Sound

Usercode playback soundbuffer: Read out by successive callback invocations.



PsychPortAudio – Demos:

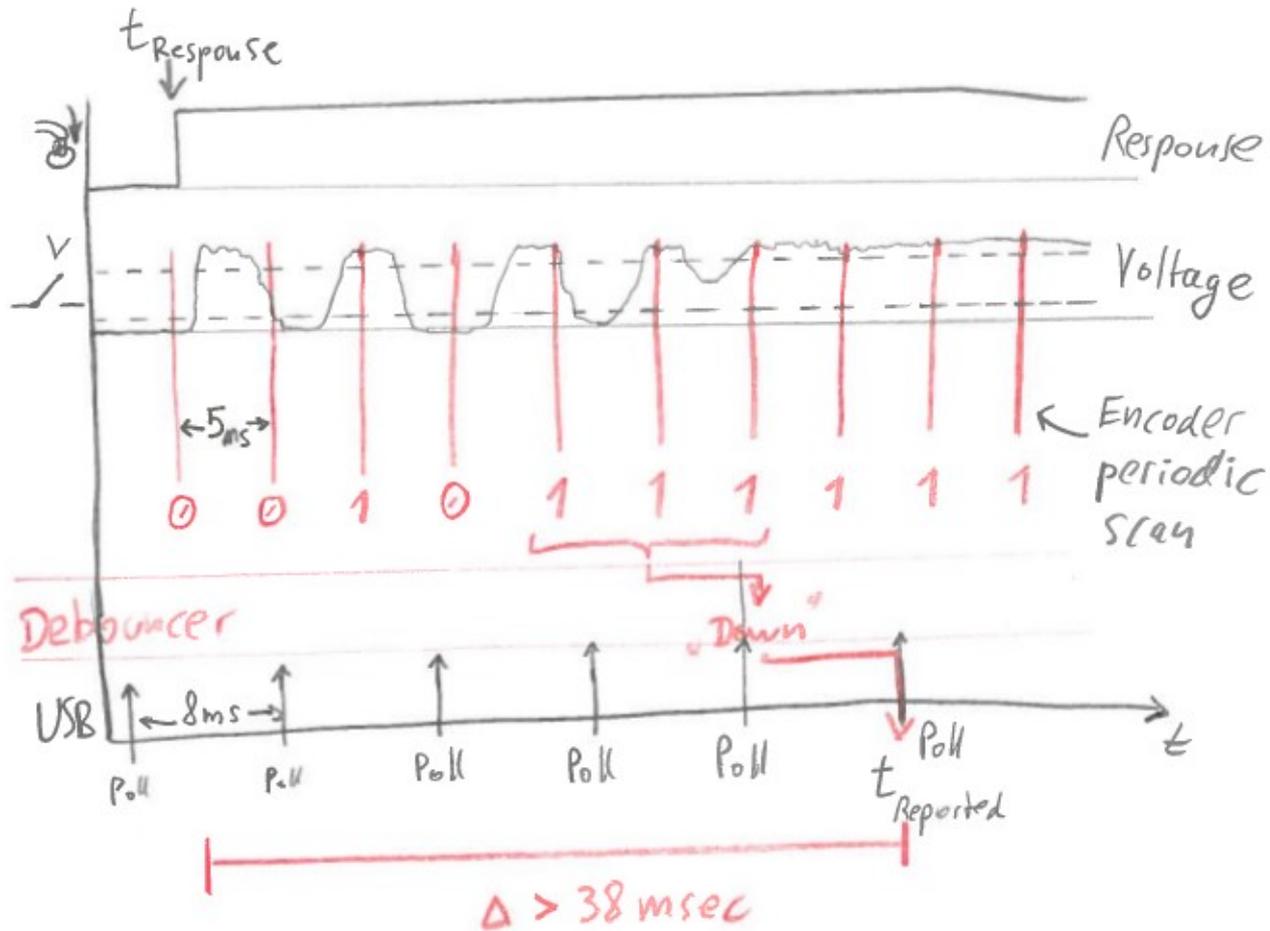
- BasicSoundOutputDemo
- BasicSoundInputDemo
- BasicAMAndMixScheduleDemo
- BasicSoundScheduleDemo
- DelayedSoundFeedbackDemo
- PsychPortAudioTimingTest

*Never ever trust audio output blindly!!!
Measure! Measure! Measure!*

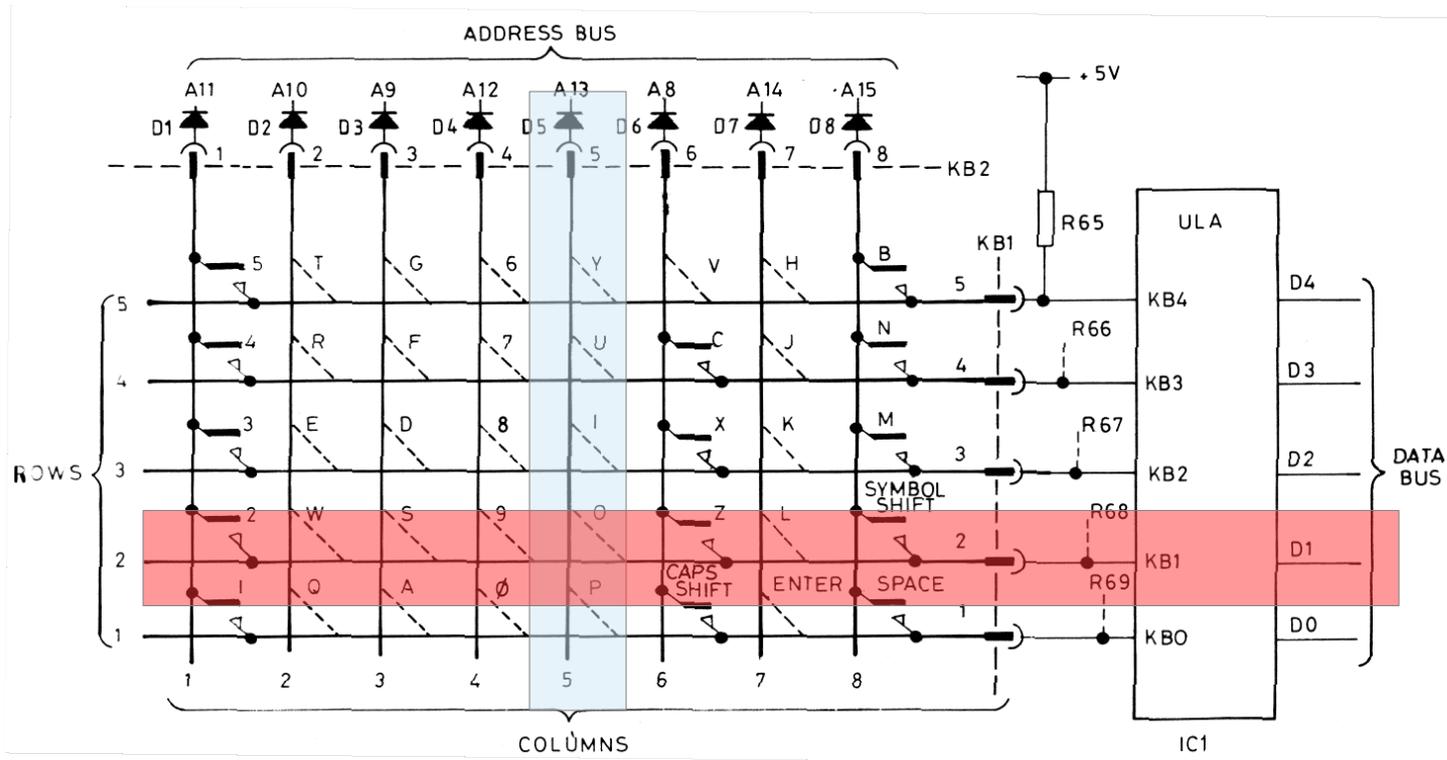
Basic response collection – Keyboard & Mouse

- `[x,y,buttons]=GetMouse(window [, mouseDevice]);`
Query current mouse position and mouse button state.
 - GetMouse on Linux can query multiple mice or touch input devices independently.
 - `[down, secs, keycode]=KbCheck([deviceNumber]);`
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.
See `DisableKeysForKbCheck()` and `RestrictKeysForKbCheck()`
 - Can query mouse-buttons and joystick buttons as if they are keyboard buttons on some systems.
 - See `KbDemo` for usage examples.
-
- Queries are fast, bypassing OS queues and application event queues.
 - Despite that: Standard keyboards and other standard input devices problematic for RT measurements due to the way the standard hardware as well as the USB bus works! No way for software to compensate for these problems.

Example: Scan + Debounce + USB sampling



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: Summed up

Encoder: 8-20 ms

Debouncer: > 20 ms

USB: 9 ms

- Standard USB keyboard can add over 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>
- Response boxes that act as keyboards, e.g., fORP from
<http://www.curdes.com/usbforp.htm>
- RTBox, if you can get it: <http://lobes.usc.edu/RTbox/>
- Bitwhacker for “do it yourself” adventures.
- Mouse can be sometimes more well-behaved if it can't generate movements. Highly model dependent.

Background response collection

- Keyboard queues can collect and timestamp button responses automatically in the background. Works from keyboards, keypads, mouse, joystick and some other standard HID button devices.
KbQueueCreate, KbQueueStart to start collection on a device. *KbEventGet, KbEventAvail, KbEventFlush* to retrieve button events. *KbQueueCheck, KbTriggerWait* for KbCheck style queries.
→ See *KbQueueDemo* for examples of use.
- Support for dedicated response box devices via *CMUBox()* for PST-Box, CMU-Box, fORP, Bitwhacker, Cedrus and similar devices.
- Support for ResponsePixx, RtBox and Cedrus boxes via dedicated functions *ResponsePixx, PsychRTBox, CedrusResponseBox*.
→ See *PsychRTBoxDemo, CedrusResponseBoxTest*, dedicated ResponsePixx demos from Vpixx.
- *KeyboardLatencyTest*, in combination with a recommended sound card for PsychPortAudio and a microphone, allows to determine the response timing precision of various supported response devices, e.g., most of the devices listed on this slide. → *help KeyboardLatencyTest*

Basic commands for system control & timing

- *T = GetSecs*
 - Query time with microsecond resolution.
 - Uses highest resolution system clock for measurement of time.
 - Compensate as much as possible for clock hardware flaws (especially needed on MS-Windows)
- *WaitSecs(duration)*
 - Wait for a specified amount of time 'duration'.
 - Accurate to < 0.1 milliseconds on average.
 - MS-Windows: Problematic in many scenarios. By far worst realtime behaviour of any operating system.
 - Linux: Especially suitable for precise timing, even under high load.
- *Priority()* - Switch process to realtime-scheduling mode, perform other realtime performance optimizations.

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 a realtime OS, e.g., RealtimeLinux ($\ll 40 \mu\text{s}$).
- 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.

Input/Output support and communication

- Network communication via TCP/UDP/IP toolbox, written by Peter Rydesäter & Mitthögskolan Östersund. Included from Matlab-Central, somewhat 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 Talias*)
- Communication with arbitrary USB-HID devices 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-650/655/... colorimeter, CRS ColorCal and OptiCal devices, and EGI Netstation EEG system.
- Serial link control via PTB IOPort command. More precise, robust and convenient for typical neuro-science use than Matlabs builtin commands.
- Parallel port support not included, but multiple free solutions exist.
- Significant improvements for digital/analog I/O expected in 2013! Stay tuned...

Questions?

The help system:

- Offline: *help Psychtoolbox*
- Online: <http://docs.psychtoolbox.org/>

The demos:

- *help PsychDemos* - demos in Psychtoolbox/PsychDemos subfolder.

The Website & Wiki:

- <http://www.psychtoolbox.org>
- If everything fails, from the Wiki -> User forum...
- Coming soon... - New beginner friendly tutorial-style demos by Peter Scarfe. Sneak preview of his nice work in progress (subject to change!) at <http://peterscarfe.com/ptbtutorials.html>