# Native D3D9 on Mesa Gallium Nine : the status

Axel Davy

FOSDEM 2015

# What is this talk about ?

- 2002: d3d9 release
- 2004: OpenGL 2.0 release
- 2004: d3d9 gets improved with shader model 3 support
- 2006: OpenGL 2.1 release
- 2006: d3d10 release
- 2008: OpenGL 3.0 release
- 2009: d3d11 release
- 2010: OpenGL 3.3 and 4.0 release
- 2012: first game with d3d11 support but no d3d9 support
- 2014: most new d3d games still released with d3d9 support

# What is this talk about ?

**Why do we want d3d9 ?**
⇒ If you want play all d3d9 games released. There's a lot of them

**But we have Steam on Linux ?**
⇒ That's cool, but what about this game *Put your game here* which is not ported ?

**Recent games are enough for me !**
⇒ Cool for you

**But wine already supports d3d9 ?**
⇒ Yes, but we can get better support with Gallium Nine.

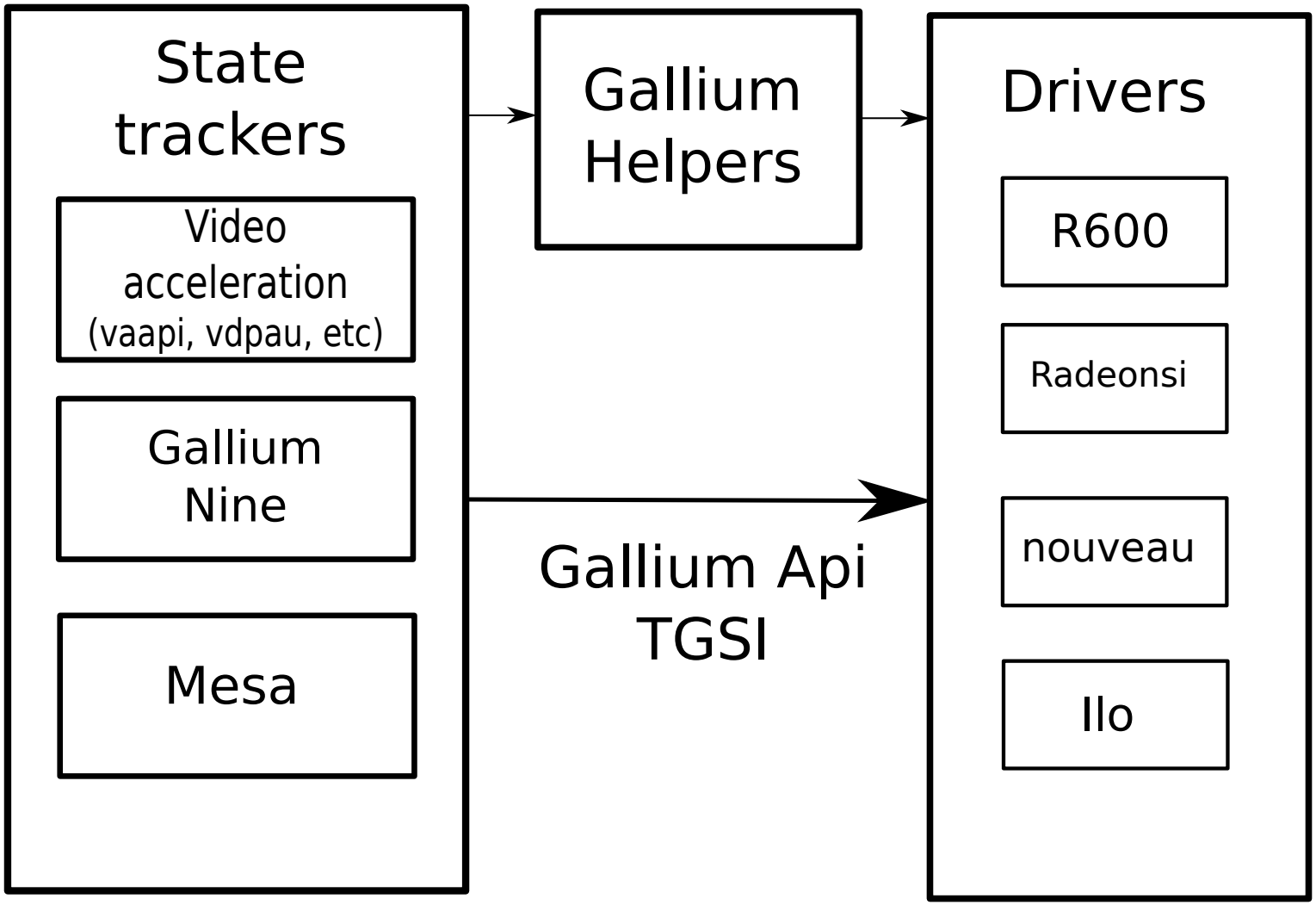
# What is Gallium Nine

Project started in **2010** by **Joakim Sindholt**.

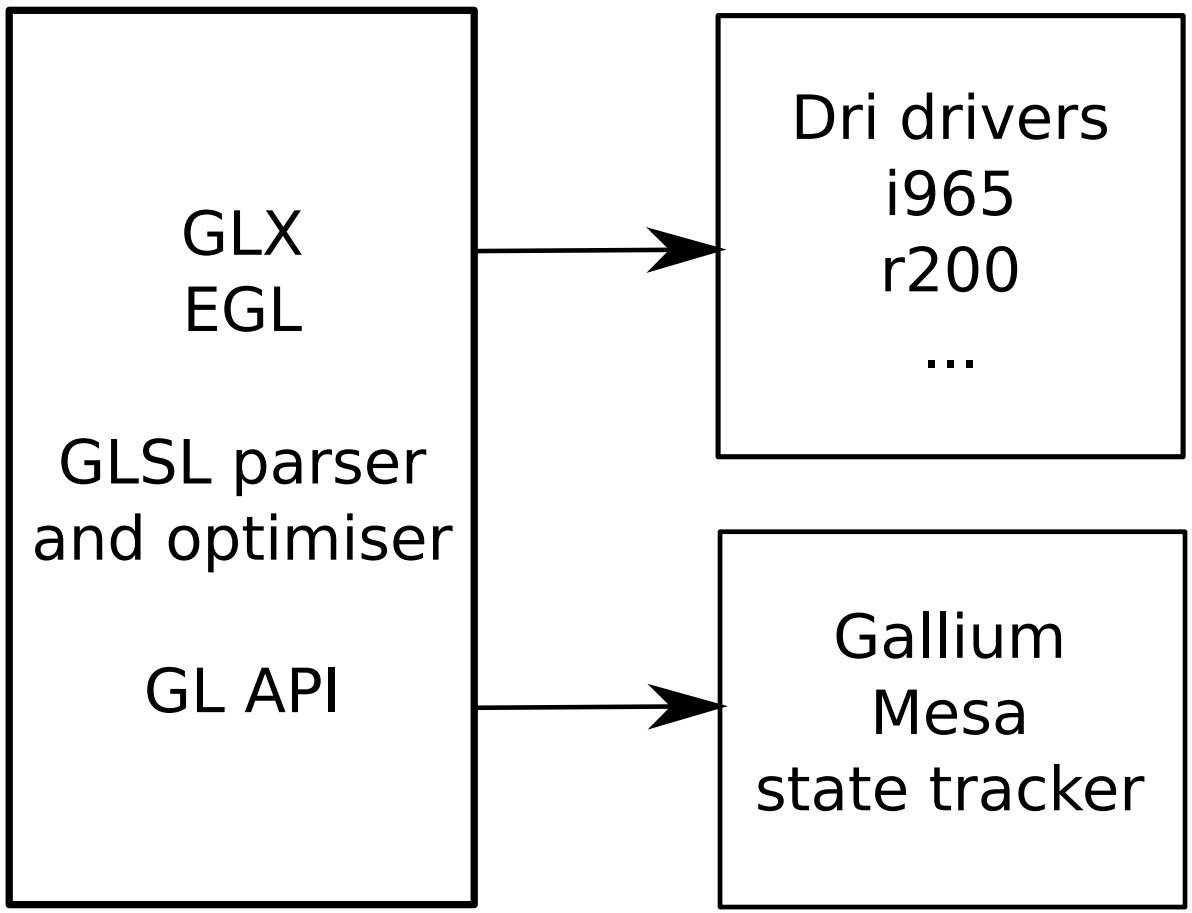
Boosted in **2013** by **Christoph Bumiller**

Project slowly improves over **2014** and get merged in Mesa.

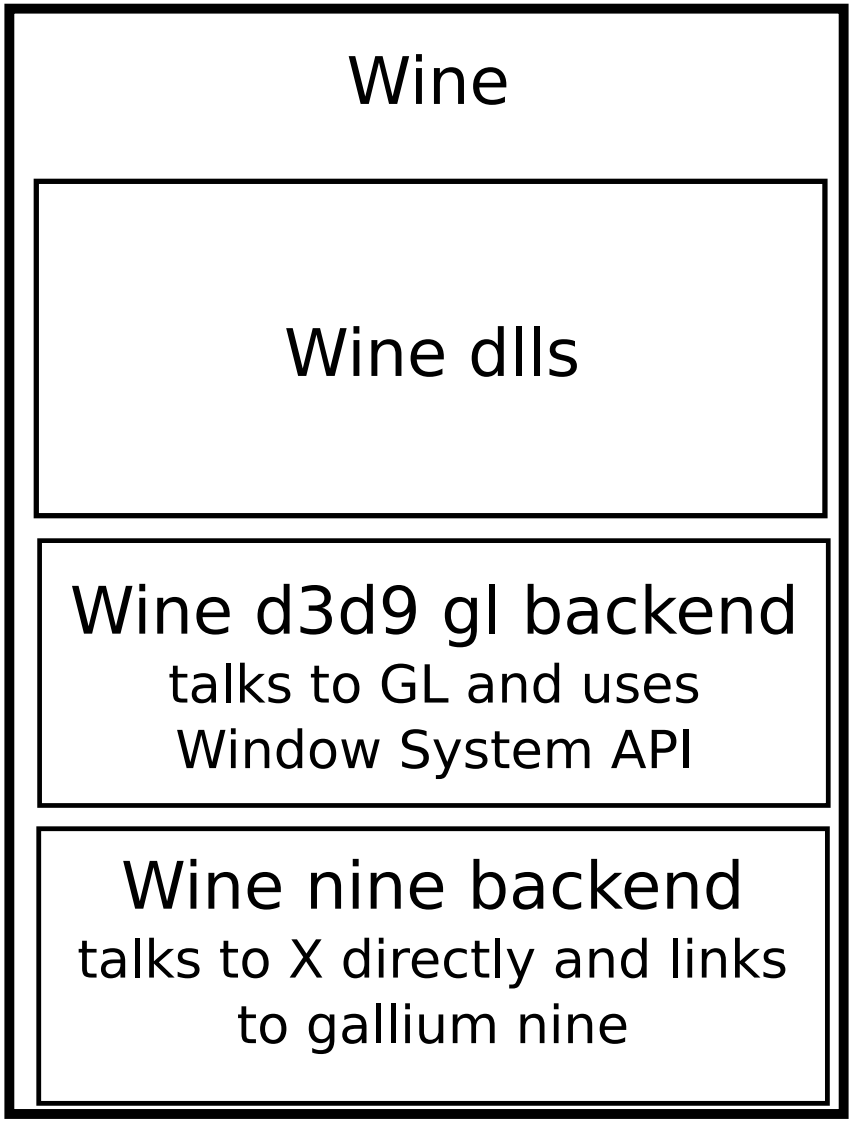
# What is Gallium Nine



# What is Mesa


GLX
EGL
GLSL parser
and optimiser
GL API
Dri drivers
i965
r200
...
Gallium
Mesa
state tracker

# How Gallium Nine and Wine are linked



# Plan

# Gallium Nine

**Gallium Nine is**:

- Mesa only. No proprietary drivers support !
- Gallium only. Poor intel support !

**It is composed of**:

- Gallium state tracker
- Wine d3d9.dll integration

# How integration works

Wine - Gallium
d3d9.dll → Direct3DCreate9 → IDirect3D9.
IDirect3D9 → IDirect3DDevice9.

IDirect3D9: Used to get supported formats, resolutions, multisampling modes and device info.
IDirect3D9: Uses D3DAdapter9 for the implementation.
IDirect3DDevice9: Used for everything related to rendering.
IDirect3DDevice9: Uses ID3DPresent to get window size and send buffers to the screen.

# How integration works

- Wine connects to Gallium Nine and implements all the Window system bits
- Gallium Nine does everything else

$\Rightarrow$ It is possible to use Gallium Nine without Wine (`Xnine`).

# Window system integration

Implementation goals:

- Client side buffer allocation
- Good multi-gpus laptop support
- Behaviour close to expected behaviour

Answer:

- `X DRI3` is about client side buffer allocation ($\neq$ `DRI2`)
- `X PRESENT` enables control with precision the buffer presentation

For better compatibility, we implemented `DRI2`/`PRESENT` fallback relying on `EGL_EXT_image_dma_buf_import` extension

# Plan

# Present extension

**D3D9** expects Render-ahead queue.

**OpenGL**: As Fast as possible OR synchronized with screen refresh.
synchronized with screen refresh: if at vblank n, two frames are presented, only last one will be shown (at vblank $n + 1$).
$\Rightarrow$ **Tripple buffering** possible.

**D3D9**: As Fast as possible OR synchronized with screen refresh.
synchronized with screen refresh: new presentation is a last vblank scheduled + 1.
All frames are presented. **NO Tripple buffering**.

# D3D9 Render-ahead queue

Apps define the number of back buffers and vblank synchronization.

At every presentation you get a free back buffer from the back buffer pool (order/behaviour defined by parameter). Wait is done when no back buffer is free.
$\Rightarrow$ In practice apps use 2 back buffers, so OpenGL behaviour is ok.
However some apps use 3 back buffers.

# multi-gpu

Some laptops have integrated gpu + dedicated gpu.

Under Mesa OpenGL you can use `DRI_PRIME` or `device_id` to choose the gpu.

# How GPU offloading works

Reminder on how `DRI_PRIME` works:

Get granted access to the device:

- `DRI2`: Special Flag for it
- `DRI3`: Use Render-nodes!

How devices talk to each other.

- Render to a tiled buffer in VRAM
- `DRI2`: Send it to `X` server, which will copy to linear buffer
- `DRI3`: Copy to a linear buffer and present it

# multi-gpu

Sorry !

# multi-gpu

Sorry !

`DRI_PRIME` under `DRI3` sucks. It wasn't intended to !

- dma-buf fences still not implemented for all gpus
- radeon driver doesn't use dma copy anymore for the presentation copy

$\Rightarrow$ GPU will sometimes display whole frames older than the previous one, or display one partially updated (triangle shaped tearing)

# multi-gpu

`DRI_PRIME` sucks because of synchronization.

`DRI2`: No synchronization expected. dgpu copies to one buffer, igpu reads from it.
`DRI3`: Synchronization expected one day. dgpu copies to several buffers, igpu reads from them.
`DRI2` always tears, `DRI3` has more potential but will show frames in wrong order or not rendered yet because of missing synchronization.

Note: we could workaround Mesa to have `DRI3` do the same than `DRI2` for now.

# multi-gpu

**Wait !**

# multi-gpu

**Wait !**

You expect synchronization done in the kernel.

# multi-gpu

**Wait !**
You expect synchronization done in the kernel.

Why not Mesa side ?

# multi-gpu

**Wait !**
You expect synchronization done in the kernel.

Why not Mesa side ?
⇒ That's the solution taken for Gallium Nine

# multi-gpu

Gallium Nine **`thread_submit=true`** parameter

Uses an additional thread to do the presentations.
Wait the buffer is rendered before presenting.

Result: Excellent. Same performance, but NO **`DRI_PRIME`** bugs.
Tear-free possible !

# Presentation of multisampled buffers

Apps can ask for a multisampled backbuffer/depth buffer.
But you want to present a single-sampled buffer.

Similar to the multi-gpu case, do a copy.
Rendering is done to multisampled buffer, and copied to non-multisampled buffer.

# Throttling

**Throttling**: Wait done when cpu submits too fast new frames and gpu cannot keep up.

⇒ Extremely important for **lag control**.

**Throttling queue**: Usually 2 buffers max for Mesa.

Controlled in Gallium Nine by `throttle_value` (default 2).
0 means "always wait" (equivalent to glFinish. Bad for performance. No lag).
−1 means "do not wait": Have fun.

# Plan

# How apps do render

Usually an app does every frame hundreds of:

- Change some Render states
- Change textures bound
- Update vertex buffer
- Switch to another Vertex/Pixel shader
- Update shader constants
- Draw
- Repeat until Presentation

Apps minimize the changes done at every draw call for better performance

# State changes

Render states in gallium are changed in groups

```
pipe_depth_stencil_alpha_state
pipe_rasterizer_state
pipe_blend_state
pipe_sampler_state
...
```

D3D9 states are changed individually

```
D3DRS_SHADEMODE
D3DRS_CULLMODE
D3DRS_FILLMODE
...
```

# State changes

States changes are commited before every new draw call.

# Vertex/Pixel shaders

Wine and Mesa state tracker both delay shader compilation at draw time.

$\Rightarrow$ there are stuttering during the first minutes of play.

This is because a vertex shader can be used with several pixel shaders and vice versa $\Rightarrow$ need to link the gl shader again for all new combinations, and because of coordinate handling (rendering to framebuffer or backbuffer doesn't have same coordinates).

# Vertex/Pixel shaders

Gallium Nine doesn't have to cope with coordinate system changes: Doesn't change !

Gallium Nine compiles shaders at the time they are expected to be compiled. Are compiled once for all (except for very special cases, but very few shaders will be concerned).

# Mapping Vertex shader outputs to pixel shader inputs

## Vertex shader code

```
VS3.0
DEF c87 { 306.5 1.000000 0.000000 0.000000 }
DCL v0 POSITION0
DCL v1 TEXCOORD0
DCL v2 COLOR0
DCL v3 BLENDWEIGHT0
DCL v4 BLENDINDICES0
DCL o0 POSITION0
DCL o1.xy__ TEXCOORD0
DCL o2.xyz_ TEXCOORD1
DCL o3 COLOR0
DCL o4 COLOR1
...
```

# Mapping Vertex shader outputs to pixel shader inputs

## Vertex shader code

```
VERT
DCL IN[0]
DCL IN[1]
DCL IN[2]
DCL IN[3]
DCL IN[4]
DCL IN[5]
DCL IN[6]
DCL OUT[0], POSITION
DCL OUT[1].xy, GENERIC[0]
DCL OUT[2].xyz, GENERIC[1]
DCL OUT[3], COLOR
DCL OUT[4], COLOR[1]
...
```

# Mapping Vertex shader outputs to pixel shader inputs

## Pixel shader code

```
PS3.0
DEF c15 { 2.000000 -1.000000 0.000000 0.000000 }
DEF c16 { -0.000000 -1.000000 -2.000000 1.000000 }
DEFI iconst[0] { 3 0 0 0 }
DCL v0.xy__ TEXCOORD0
DCL v2.xyz_ TEXCOORD1
DCL v6 COLOR0
DCL v7 COLOR1
...
```

# Mapping Vertex shader outputs to pixel shader inputs

We associate a index to every usage/index possible, and fills the data into `GENERIC[index]`.

Index bijection fixed. `GENERIC[index]` can be sparse.
No need to recompile when using different pixel or vertex shader !

# Translate shader code

Code in binary format, already optimised.

Translation easy:

```
MUL r0._yzw r0.yyyy c10.xxyz
```

becomes

```
MUL TEMP[0].yzw, TEMP[0].yyyy, CONST[10].xxyz
```

# Translate shader code

But special cases to handle around 0, Inf and NaN

```
RSQ r0.x___ r0.xxxx
```

becomes

```
RSQ TEMP[0].x, TEMP[0].xxxx
MIN TEMP[0].x, IMM[0].wwww, TEMP[0].xxxx
```

With `IMM[0].wwww = FLT_MAX`

# Mapping formats

```
D3DFMT_A8R8G8B8 => PIPE_FORMAT_B8G8R8A8_UNORM
D3DFMT_D24S8    => PIPE_FORMAT_S8_UINT_Z24_UNORM
D3DFMT_D24X8    => PIPE_FORMAT_X8Z24_UNORM
D3DFMT_D16      => PIPE_FORMAT_Z16_UNORM,
```

We map to the equivalent gallium format.

# Conclusion

- State handling is easy
- Draw call mapping are easy
- Shader code to TGSI is easy
- Format conversion is easy

Great, but why are there still bugs ?

- fixed function code special undocumented behaviours
- How to handle cases supposed to be forbidden by the spec, but that apps do anyway ?
- Undocumented special behaviours
- Stateblocks are hard to implement right

# CPU overhead

- Gallium Nine has low CPU overhead because the conversion from d3d9 call to gallium API is easy.
- With Gallium API, we can assume API call succeed. No need to check driver error. Checks are done by Gallium Nine before submission.
- State change: Could do better
  - What we do: put flags on which gallium state groups need being updated. Update them at draw call.
  - What we could do: update the state groups structure right away, and put flag to submit it at draw call.

# Plan

laptop
Intel(R) Core(TM) i5-3210M CPU @ 2.50GHz
Amd HD 7730M (Slowest GCN card !)

OS:
Win 7
Ubuntu 14.10
Arch Linux, Mesa Ixit git + llvm SI scheduler + dma copy enabling patch

This is a GPU limited scenario.
Under Win, the Amd card is maximum 2x better than the Intel card, but it is only reached for heavy games (Skyrim, etc)

| OS | Intel card | Amd card |
|---|---|---|
| Win | 83 | 85 |
| Ubuntu Native | 55 | 60 |
| Arch nine with SI scheduler | NA | 89 |
| Arch nine without SI scheduler | NA | 80 |
| Arch wine with SI scheduler | 50 | 63 |
| Arch wine without SI scheduler | 50 | 56 |

Frames per Second (fps) on Portal
on the same scene with same settings (Mid)

Sorry, couldn't test more on this machine. But as additional info, Skyrim looks like 75% of win perf under Arch nine. (And more like 50% for Wine)

`Intel i5 3330`
`Amd HD 7790`

OS:
`opensuze factory, Mesa Ixit git`

Note:
tests with `WINEDEBUG=-all`, cpu on `performance` governor
This is a more CPU limited scenario.

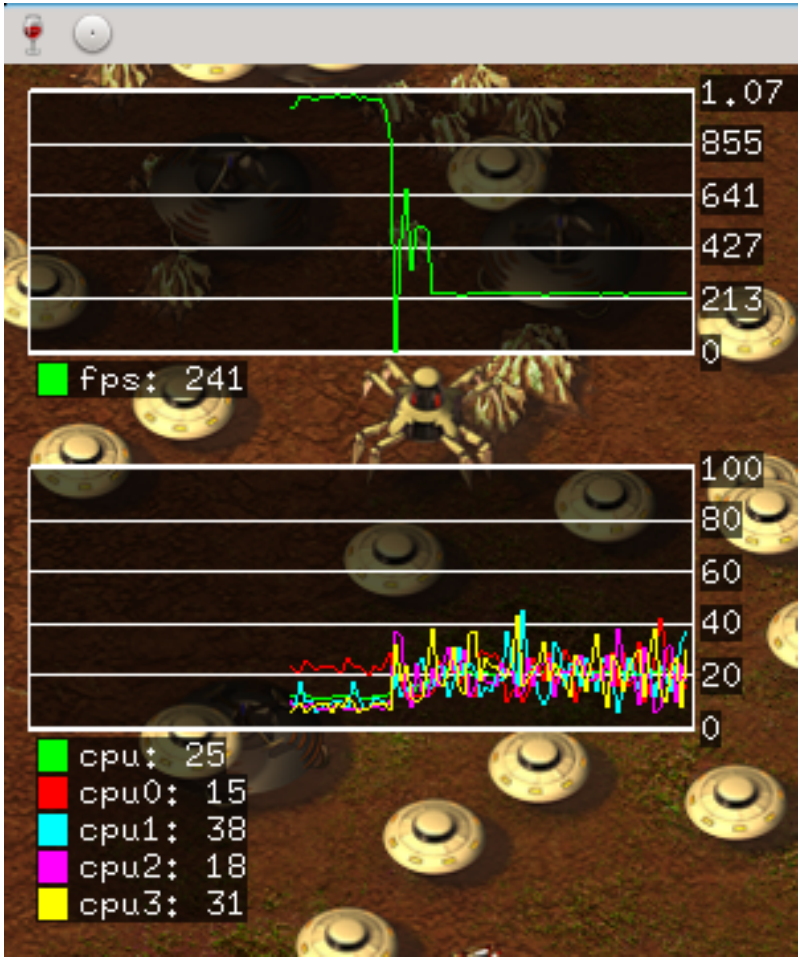



Figure: Gallium Hud under nine. Harvest Massive Encounter

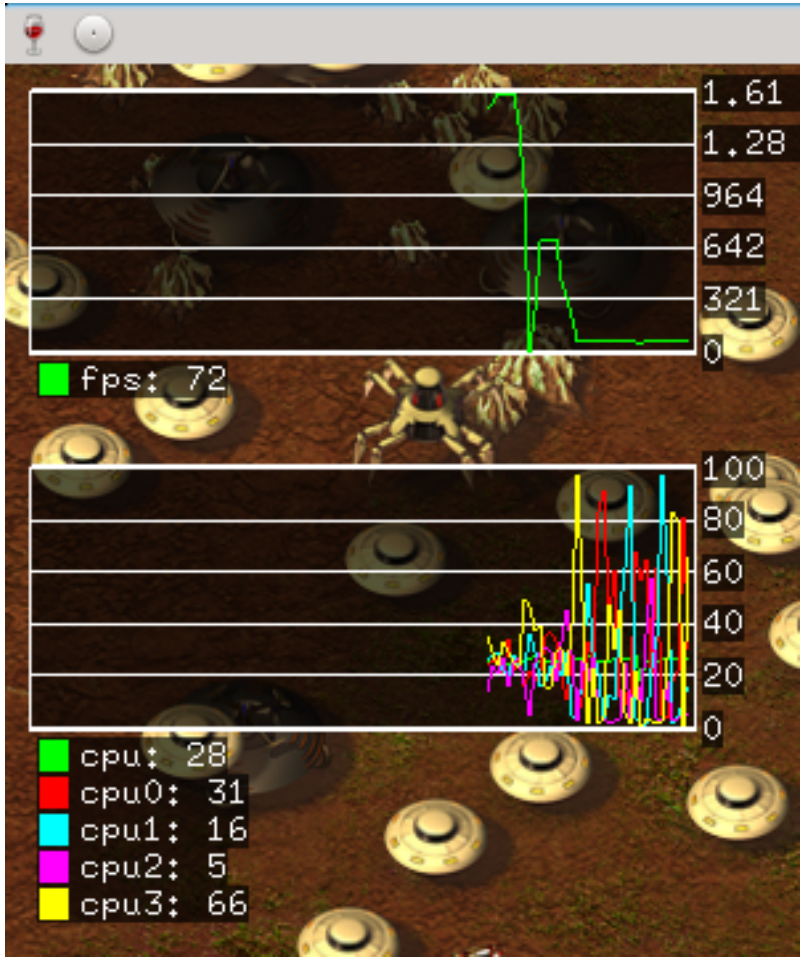



Figure: Gallium Hud under wine. Harvest Massive Encounter

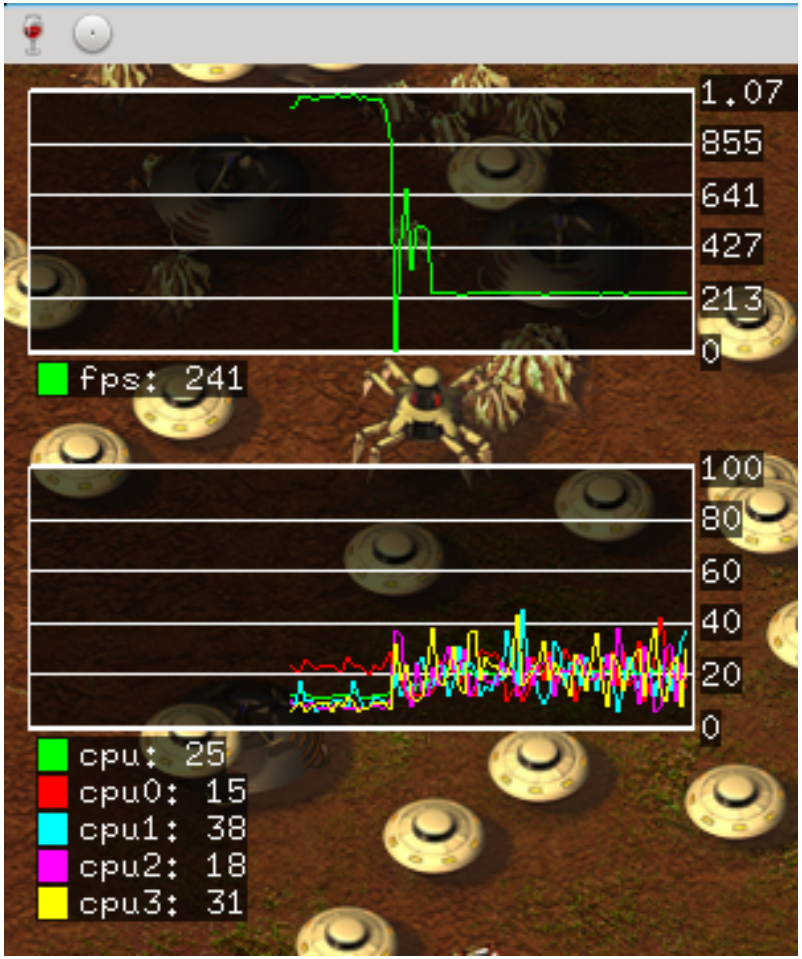



Figure: Gallium Hud under nine.
Harvest Massive Encounter

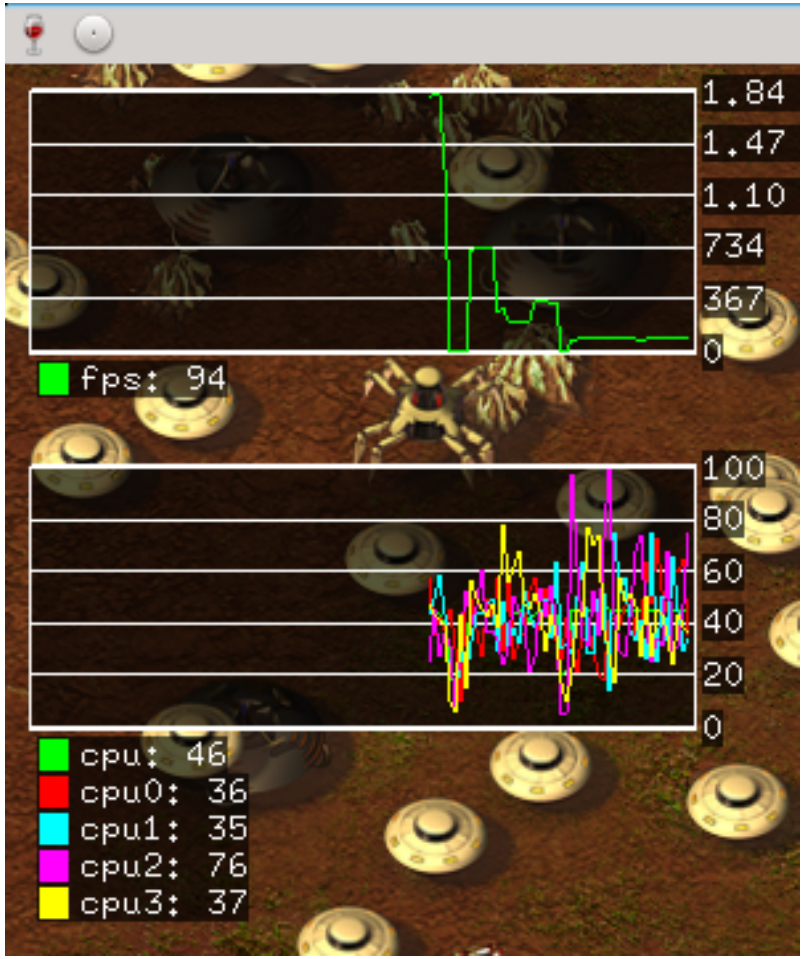



Figure: Gallium Hud under wine csmt.
Harvest Massive Encounter

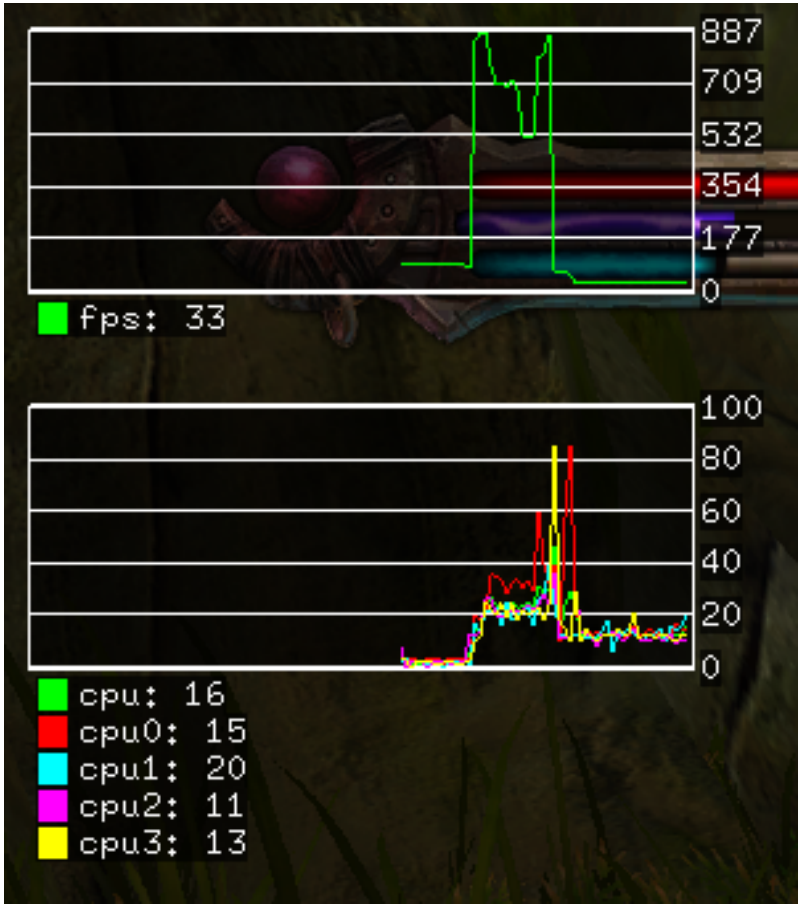



Figure: Gallium Hud under nine. Kingdoms of Amalur Reckoning

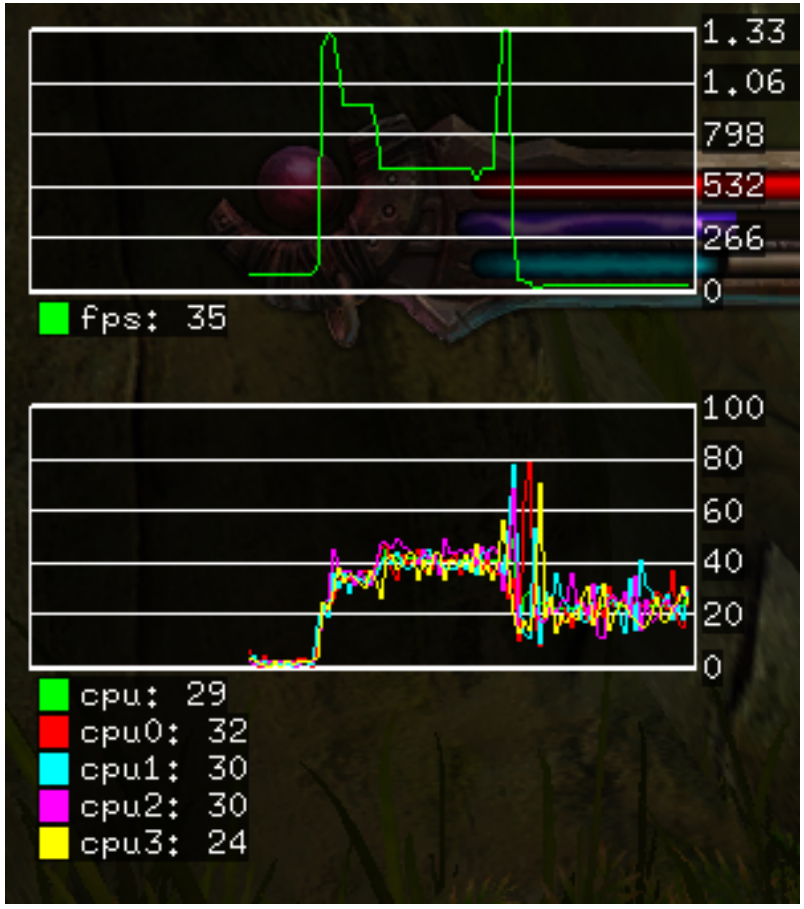



Figure: Gallium Hud under wine. Kingdoms of Amalur Reckoning

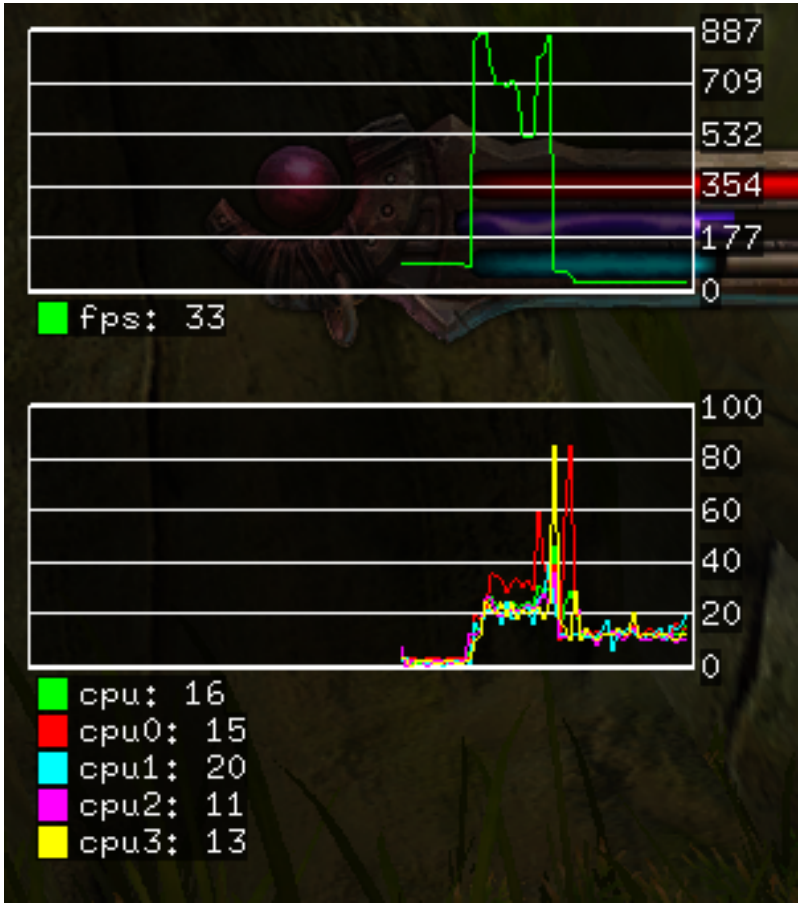



Figure: Gallium Hud under nine.
Kingdoms of Amalur Reckoning

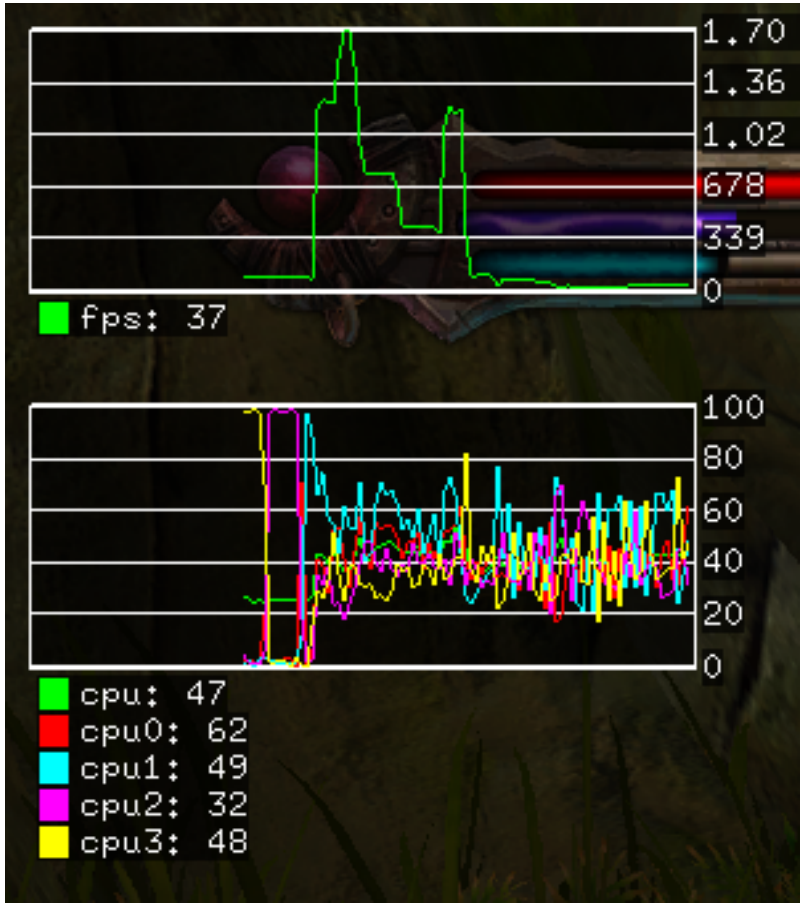



Figure: Gallium Hud under wine csmt.
Kingdoms of Amalur Reckoning

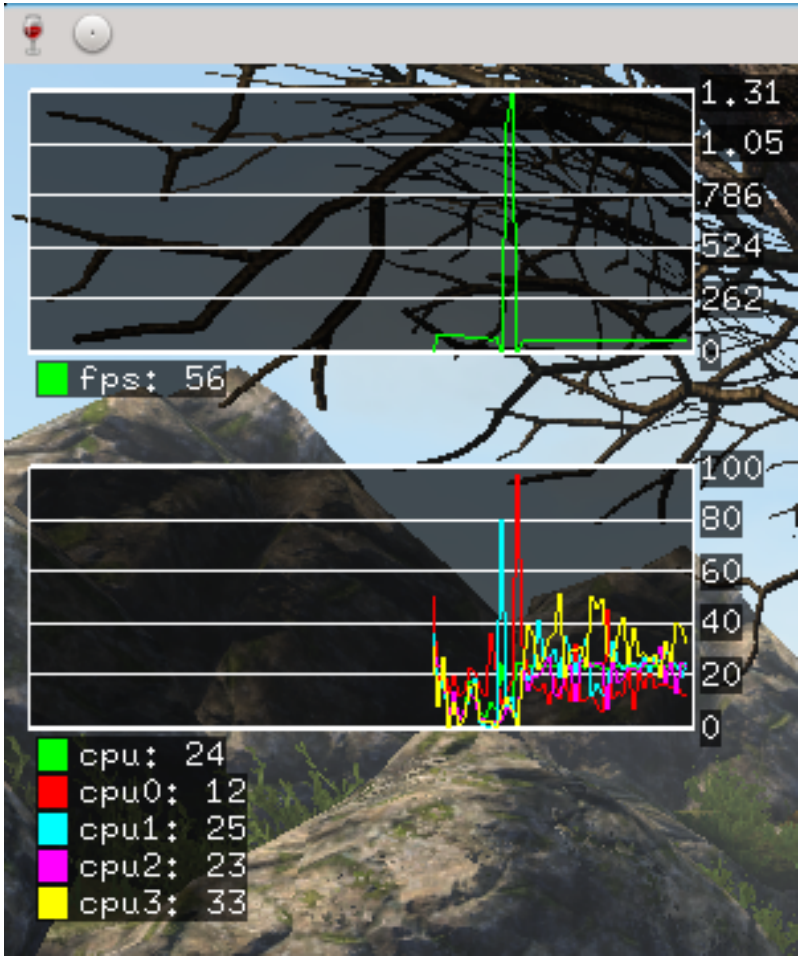



Figure: Gallium Hud under nine. Legend Of Grimrock 2

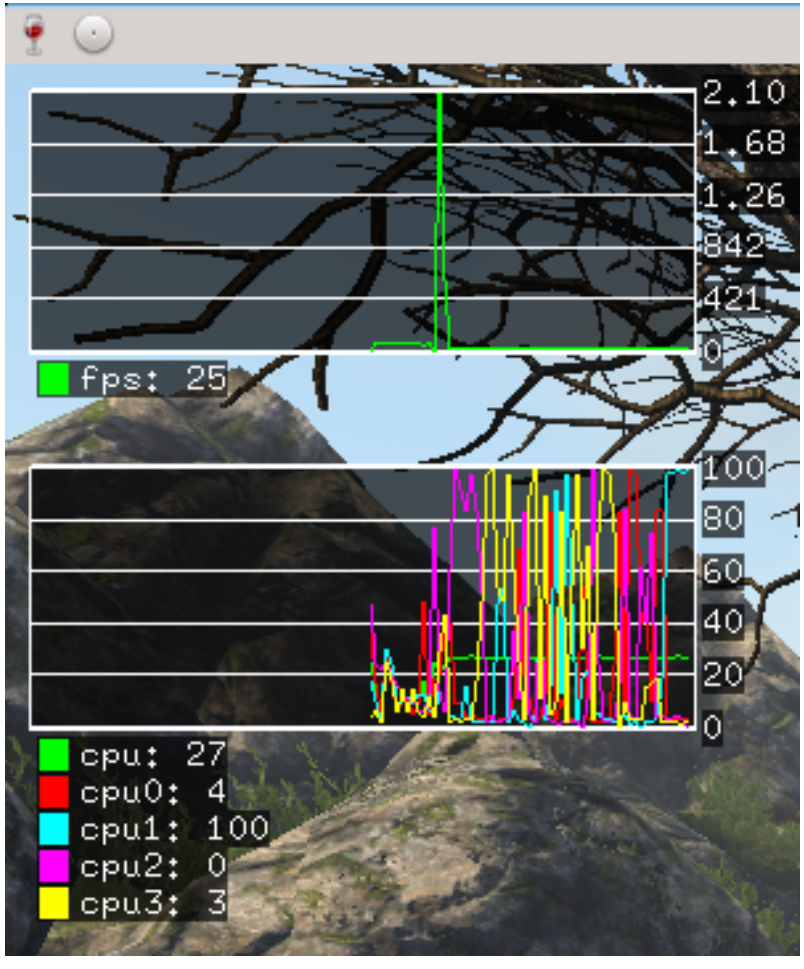



Figure: Gallium Hud under wine. Legend Of Grimrock 2

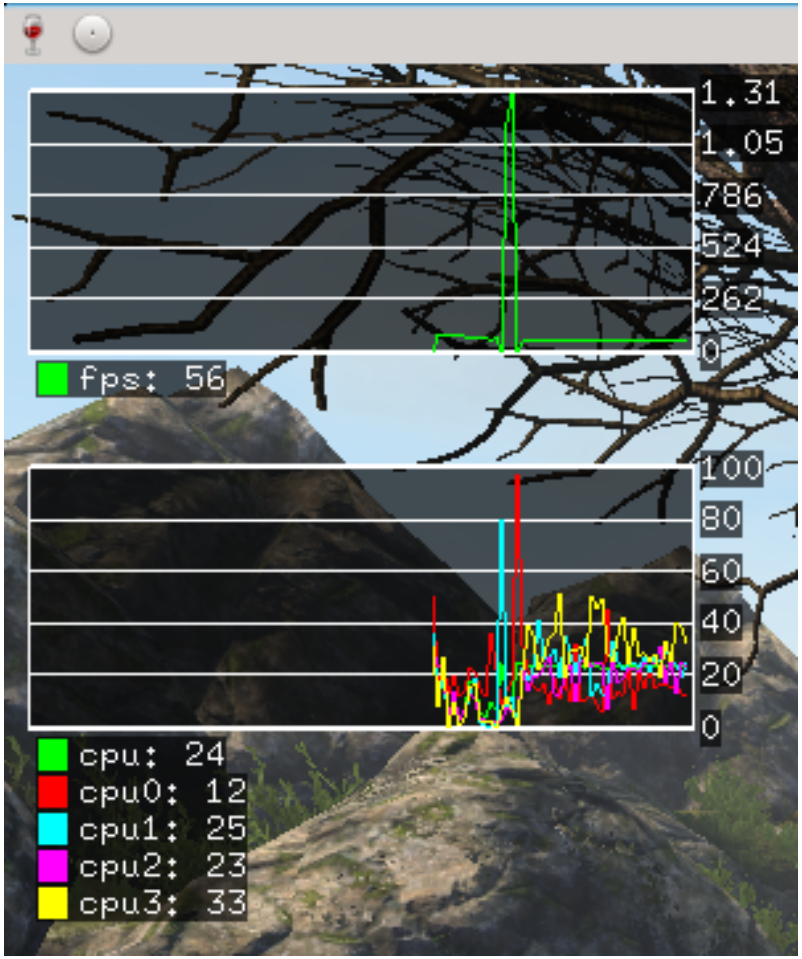



Figure: Gallium Hud under nine.
Legend Of Grimrock 2

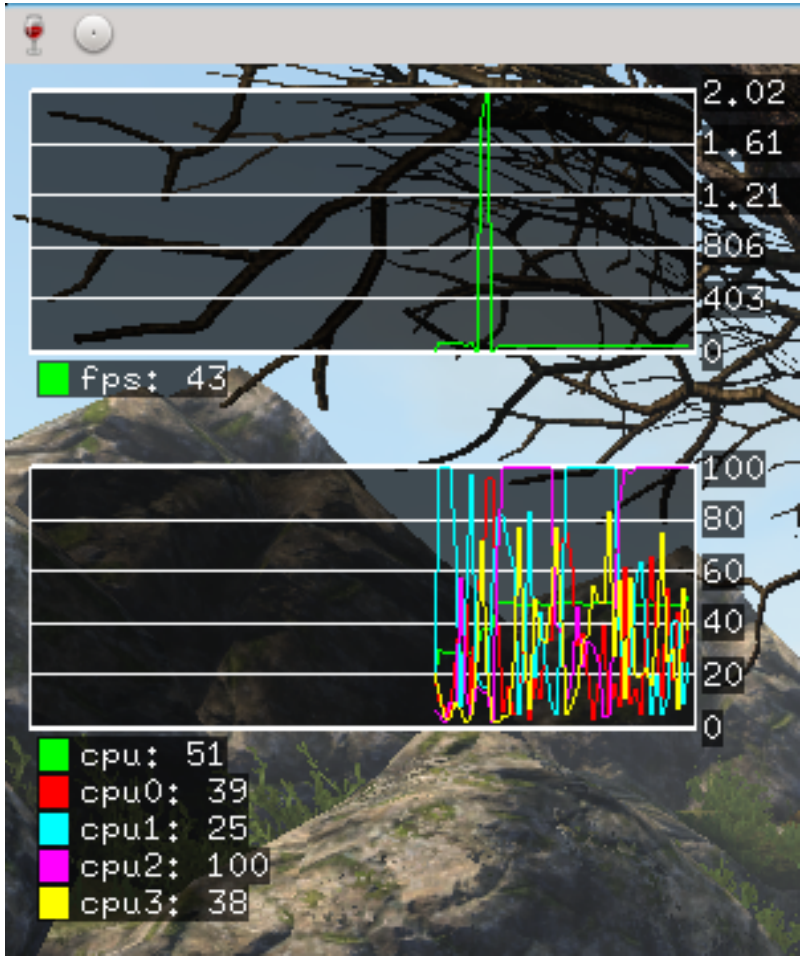



Figure: Gallium Hud under wine csmt.
Legend Of Grimrock 2

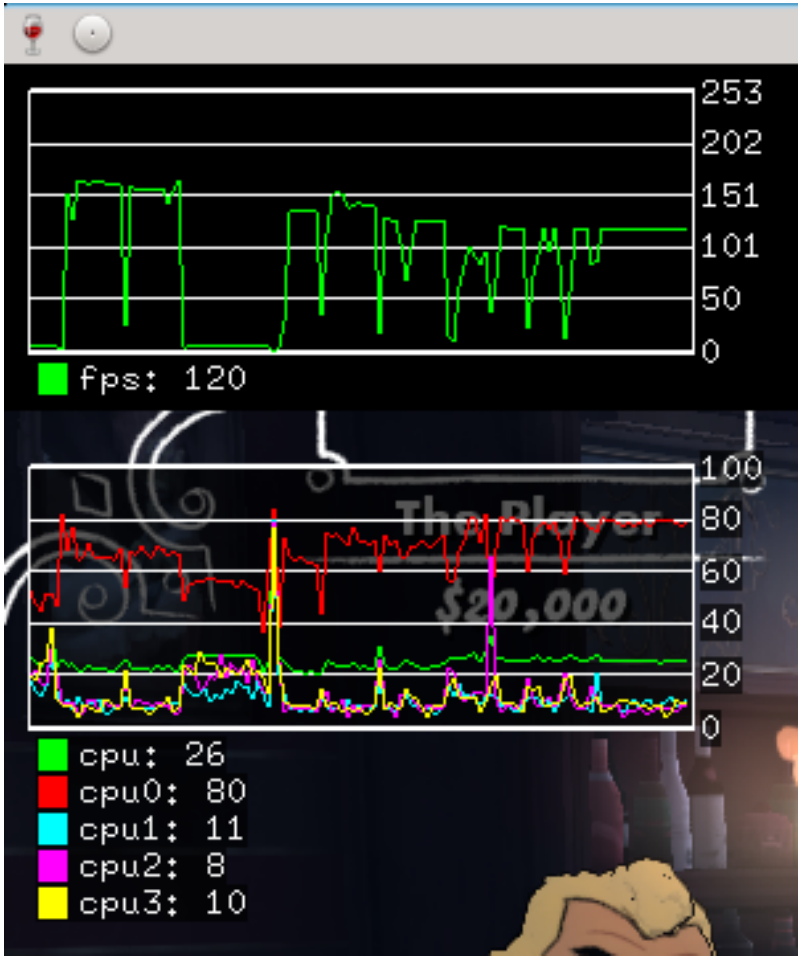



Figure: Gallium Hud under nine. Poker Night 2

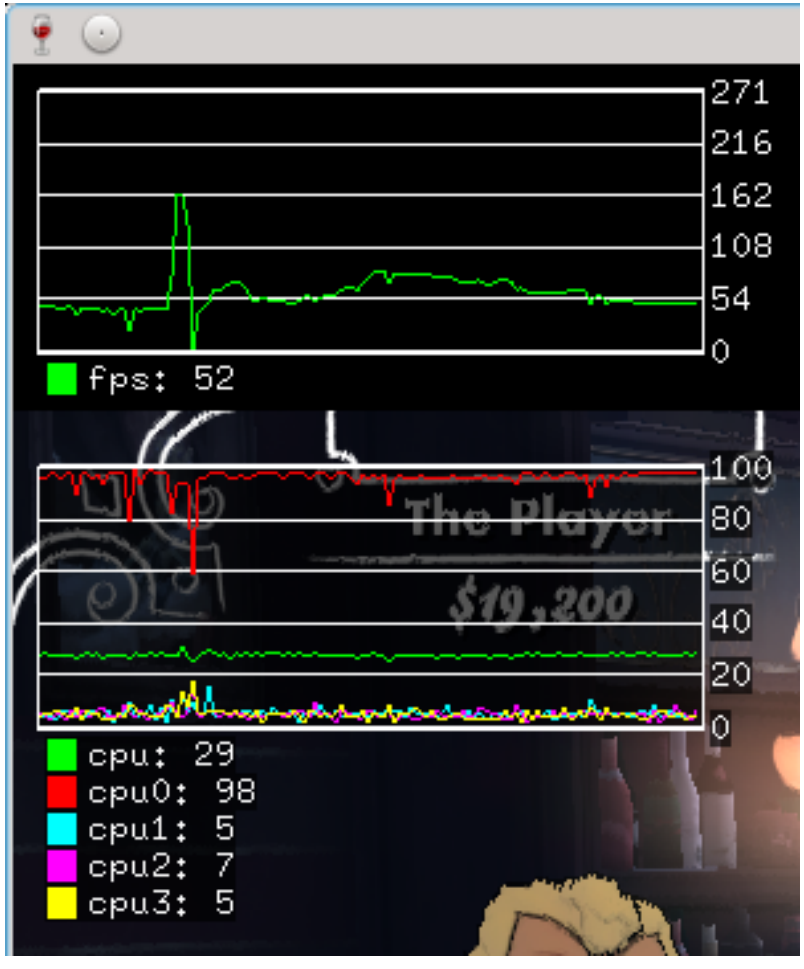



Figure: Gallium Hud under wine. Poker Night 2

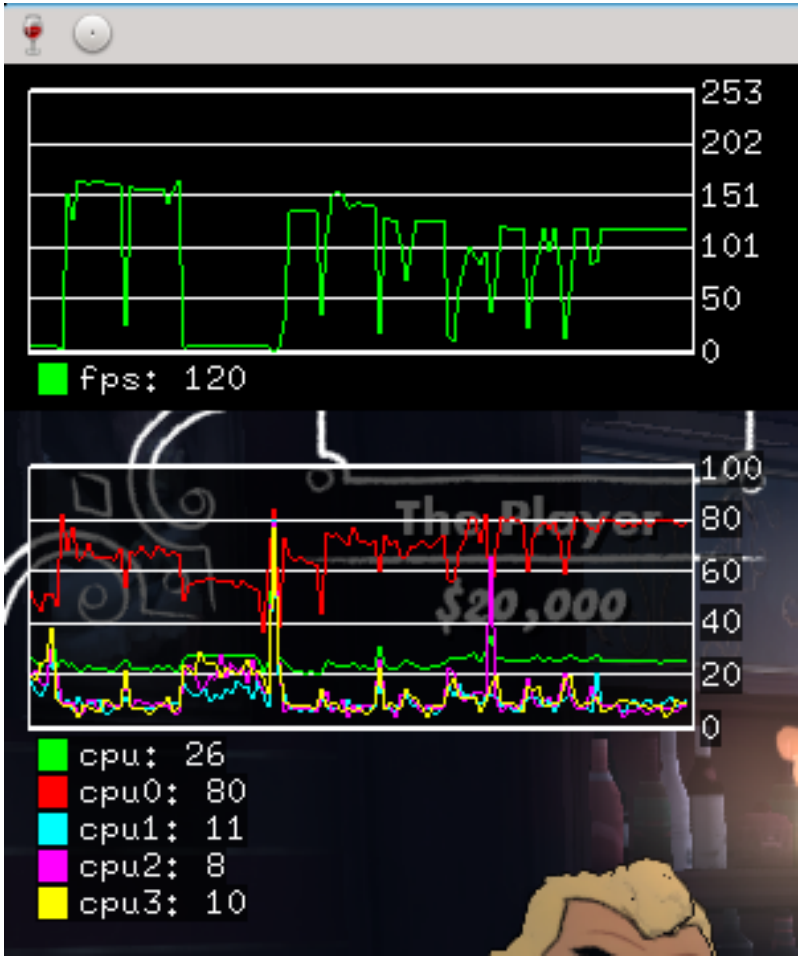



Figure: Gallium Hud under nine.
Poker Night 2

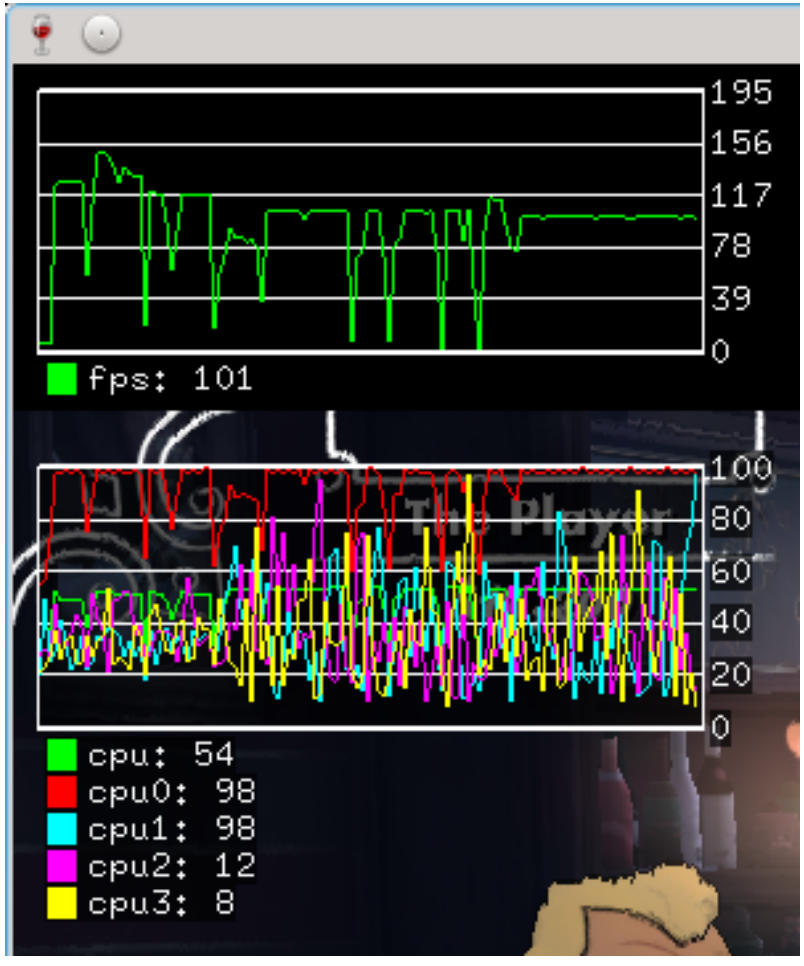



Figure: Gallium Hud under wine csmt.
Poker Night 2

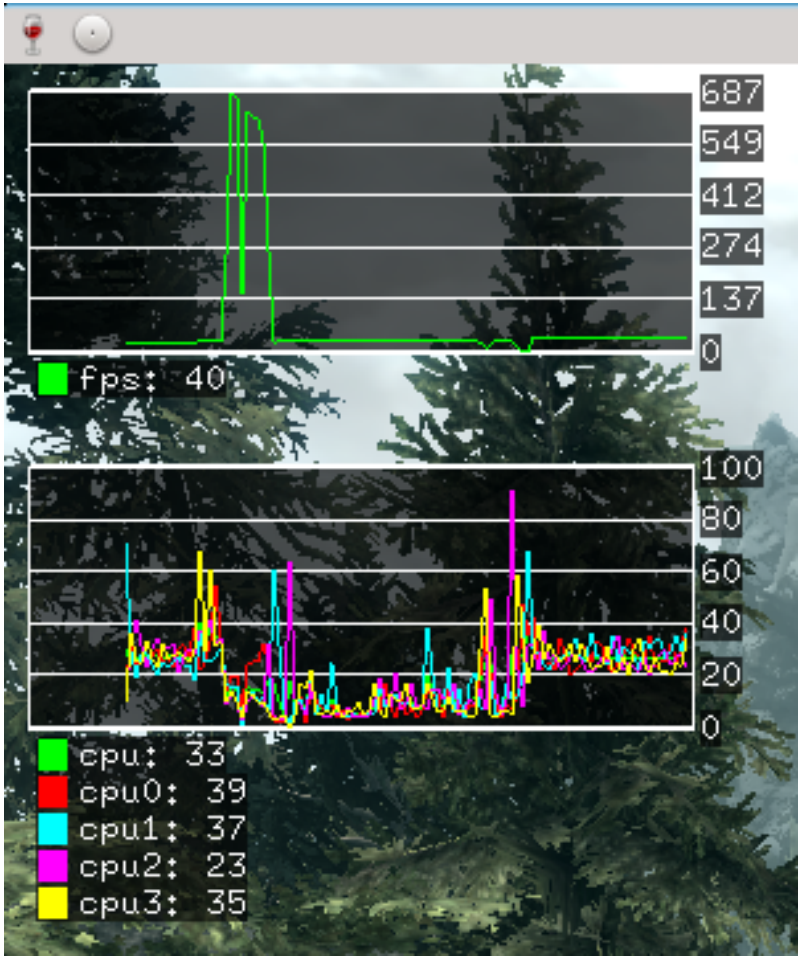



Figure: Gallium Hud under nine. Skyrim

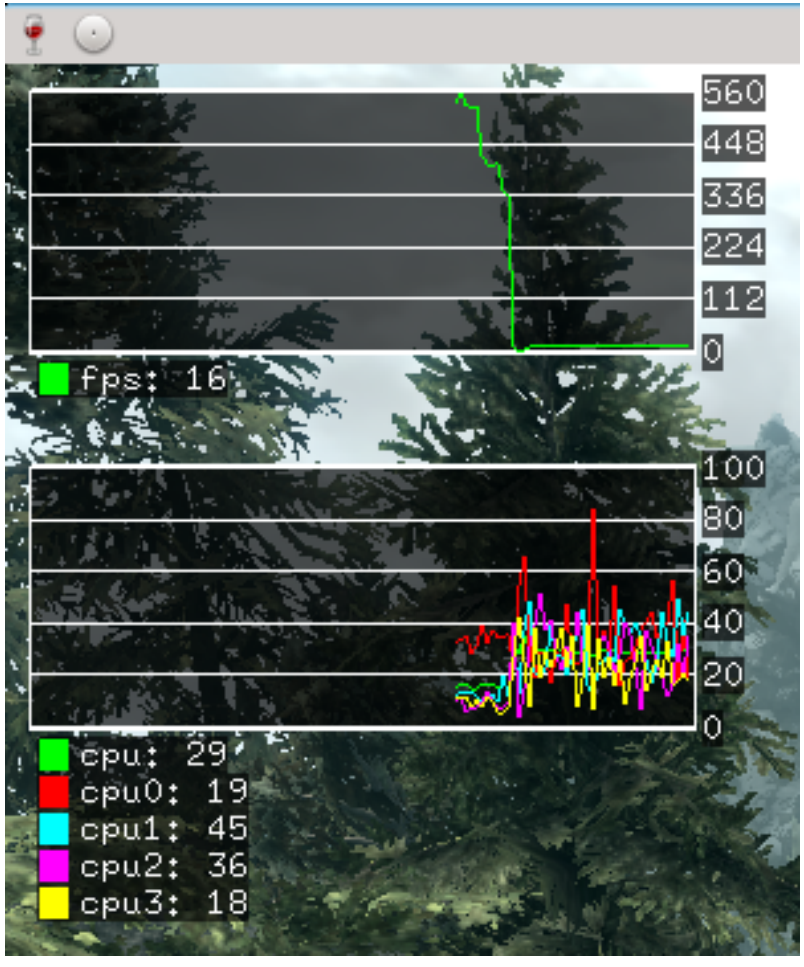



Figure: Gallium Hud under wine. Skyrim

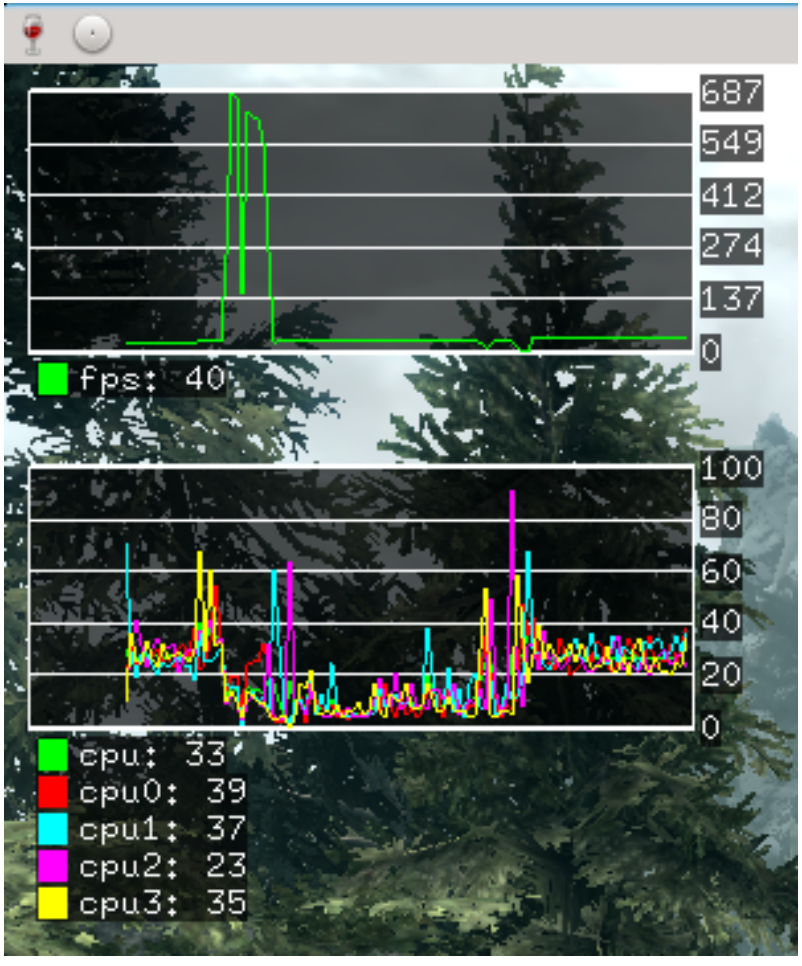



Figure: Gallium Hud under nine.
Skyrim

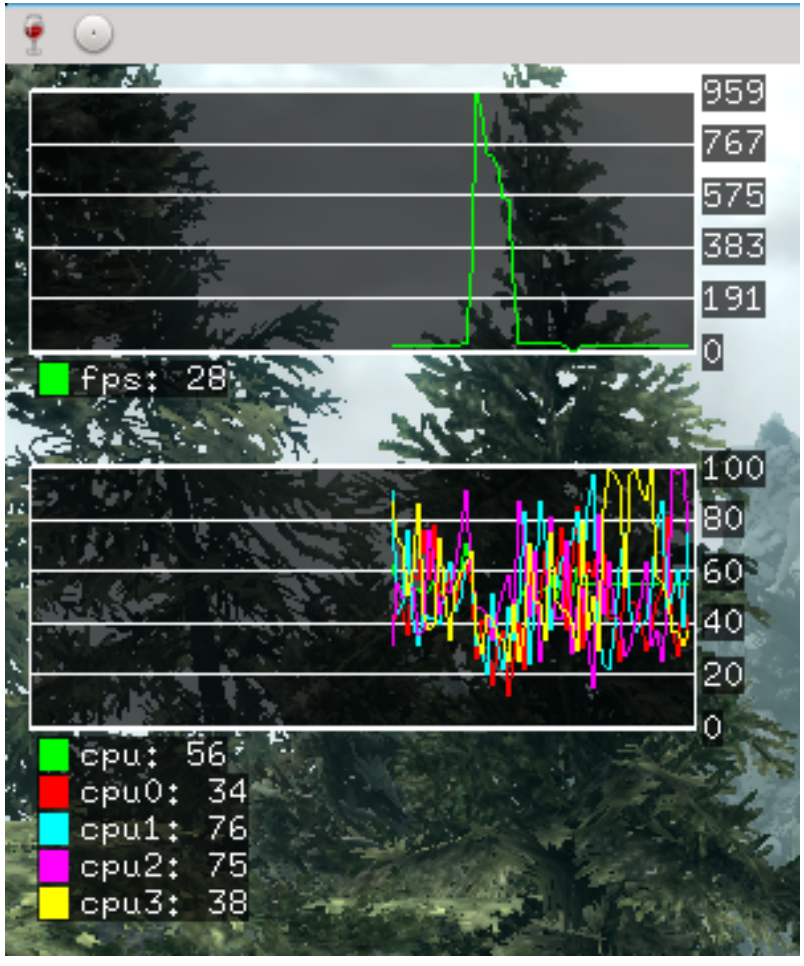



Figure: Gallium Hud under wine csmt.
Skyrim

- When Nine works, it's usually faster than Wine.
- Lower cpu usage

# Plan

# Wine vs Nine

Both Wine and Nine have bugs on some games (graphical bugs, games not launching, etc)
Currently Wine gets more games to work (but Nine manages to run games wine cannot run properly)

A fast, well-working Wine is better than everything else. But hard!
Better than working on d3d1x state trackers, it would be better help wine with GL extensions.
$\Rightarrow$ But in the next few years, we expect Gallium Nine to still beat Wine.

# Merging Nine support into Wine

Currently Mesa >= 10.4 have Gallium Nine support. But it needs special code Wine side.

One needs to compile a special branch of Wine → not easy for users!

We have now PlayOnLinux support, and we could be integrated to wine staging in the near future.

# This is the end...

Thanks for your attention.

Questions ?