Freie Universität Berlin

Bachelor's thesis at the Software Engineering Research Group of the Institute of Computer Science

Application Mode Setting Emulation In XWayland

Robert Mader
Student ID: 4676235
robert.mader@fu-berlin.de

First Examiner: Prof. Dr. Lutz Prechelt
Second Examiner: Prof. Dr. Volker Roth

Berlin, July 19, 2018

Abstract

The graphic stack of the Linux/BSD world is currently undergoing major changes. The X11 protocol, the central piece ever since its introduction in the 1980s, is being replaced by the modern Wayland protocol. In order to smooth the transition phase, there exists a emulation layer called XWayland. In this thesis I propose and try to implement support for applications requiring mode setting (read: changing display resolutions) in XWayland, using an optional and also to be implemented Wayland protocol extension.

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This paper was not previously presented to another examination board and has not been published.

July 19, 2018

Robert Mader

Contents

1 Introduction

Display mode setting is the act of setting a resolution and refresh rate of a display device. Many legacy fullscreen applications, especially games, rely on this ability as a convenient way to change their environment to their needs, thereby allowing optimal performance and reduced complexity of the application.

On modern operating systems, this practice is increasingly discouraged, mainly because of its complex technical implications. The common notion is to require applications to adopt to a given environment instead, arguing that technical complexity and performance implications have been dramatically reduced by modern rendering techniques.

In the Linux/BSD world, a new display protocol called Wayland recently emerged to overcome limitations of the aging X11 protocol and provide a future proof basis for graphical applications to communicate with compositors, the programs responsible to deliver the graphical output of applications on the actual display device. As part of its future-oriented design and in contrast to X11, Wayland doesn't allow mode setting by individual applications.

To smooth on the transition-period between the two protocols, the major X11 implementation Xorg was extended with a new back-end called XWayland. It allows X11-based applications to function just normally in Wayland based desktop environments. While great efforts have been made to ensure near perfect compatibility, applications requiring the ability to change display modes are not supported yet.

At the end of 2015 a bug entry concerning this issue was opened[1] and the possibility of reimplementing support for application mode setting was discussed. While this was quickly dismissed[2], the next likely solution was laid out: somehow emulate mode setting in a way so the desired outcome - make an fullscreen application fill the whole screen - is mirrored as closely as possible, independently of the actual mode of the display.

Moving from the X11 to the Wayland protocol is a highly desirable step for the Linux/BSD ecosystem[3][4][5], with strong implications concerning security, feature enablement, performance and maintainability of code. However, as a lesson from similar processes in the past like the introduction of pulseaudio[6], a strong resistance within the community and fragmentation within the ecosystem is to be expected if good backward compatibility is not archived. This thesis is an attempt to create a general solution for applications requiring mode setting.

[1] find a solution for XWayland games trying to set display resolution https://bugzilla.redhat.com/show_bug.cgi?id=1289714

[2] https://fedoraproject.org/wiki/Wayland_features#XRandR_control_of_Wayland_outputs

[3] https://wiki.gnome.org/Initiatives/Wayland

[4] https://wayland.freedesktop.org/faq.html

[5] https://www.phoronix.com/scan.php?page=article&item=x_wayland_situation&num=1

[6] http://0pointer.de/blog/projects/jeffrey-stedfast.html

2 Definitions and Fundamentals

Protocol Set of specified commands, allowing applications to communicate with each other. Sometimes referred to as API - Application Programming Interface.

Windowing System Part of an Operating System. Collection of applications that realize a concept how to manage graphical output of several applications on shared output device like a screen.

Display Server Application responsible for managing graphical output of a Windowing System. In order to display graphical output, applications need to connect to such a server as client, using a Display Server Protocol.

Client Application connected to a Display Server.

Window / Surface Slightly different concepts to represent graphical output of Clients within a Windowing System. In this context used mostly interchangeable.

Buffer Memory representation of the content of a Surface in the form of a texture or image. Raw form of the graphical output of a application.

Windowing Manager Application responsible for arranging Windows within a Windowing System.

X11 / Xorg A Display Server Protocol and its most widely used implementation. Originating in the 1980s and commonly used on Linux/UNIX/BSD Operating Systems. Evolved heavily from how it was intended to be used originally to how it is used on modern systems.

X11 Window Manager - XWM A Windowing Manager for X11 Clients.

Extended Window Manager Hints - EWMH Protocol for communication between X11 client and XWM.

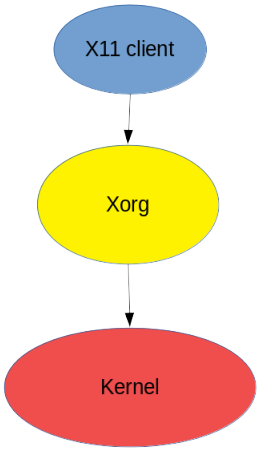
Randr and vidmode Protocols for mode setting.

Wayland A new Display Server and Window Managing Protocol.

While having a very minimal base protocol, it is designed to be extendable vie optional Protocol Extension. Being protocols themselves, I will occasionally simply refer to them as such.

Wayland Compositor Wayland Display Server and Windowing Manager at the same time. Many XWMs got extended to become Wayland Compositors.

Figure 1: Classic X11 design



Weston Reference implementation of Wayland and its Protocol Extensions. Very minimalist and not intended to be used by end users.

Mutter An XWM and Wayland Compositor.

Desktop Environment Combination of Display Server, Window Manager (or Wayland Compositor) and several to make them usable.

Session Instance of a Destop Environment

The Viewporter Protocol Extension `wp_viewporter`, a Protocol Extension that I will also refer to as `viewporter protocol` from here on. One of three `modifiers` that decouple the relationship between a buffer and its surface. Allows to apply scaling and cropping before the content of a buffer is `composited`.

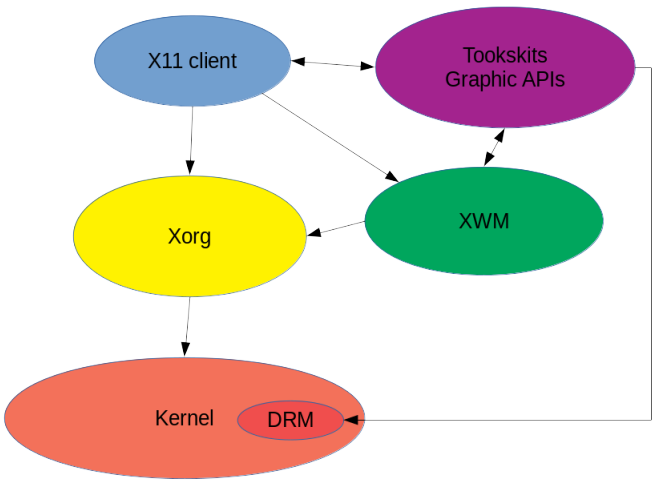
3 Technical approach

3.1 Mode setting emulation

When discussing possible solutions with members of the Xorg/Wayland community, there was a strong consensus to try to stick to the current XWayland design as closely as possible. Among others that means:

- XWayland should archive its goals by making use of Wayland protocols, as a Wayland Compositor should be its only dependency.

Figure 2: Modern X11 design



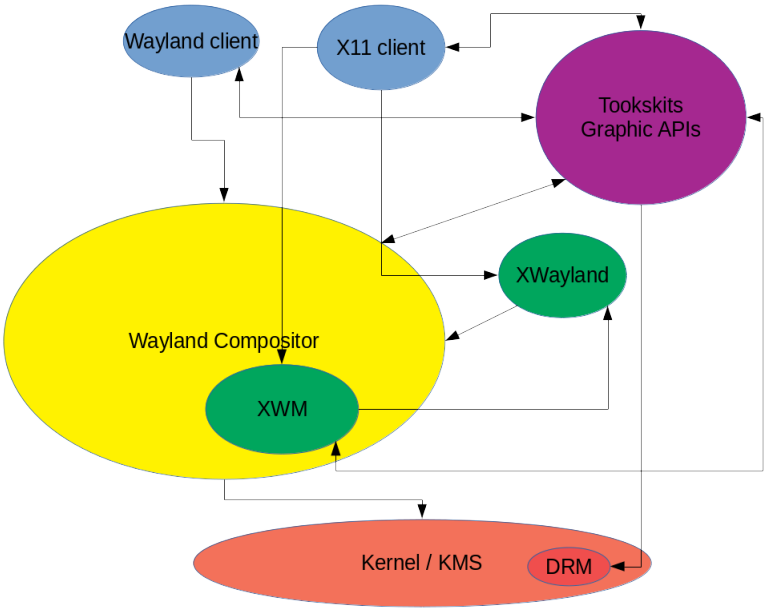
- Window management is the responsibility of the XWM. Therefore XWayland does not make use of Wayland Window Management Protocol Extensions such as XDG-Shell. Further, there is no common concept of a fullscreen mode in X11 - it is part of the Window Management
- XWayland implements the RandR and vidmode protocols. Those implementations will have to get extended to emulate mode setting.
- There needs to be a way to scale up graphical output. The viewporter protocol exposes this functionality to Wayland clients, so it should be used.

Based on these constrains, the plan is to iteratively develop a solution centered around XWayland. Whenever feasible, implementation of functionality should happen there, in a as generic as possible manner. If no sufficient solutions can be found, the next step is to evaluate if the XWM or the WM compositor can help - in this order, as I want to avoid increasing complexity of Wayland implementations.

I anticipate the following steps to be necessary:

- Expose mode setting to XWayland-clients
  - only the corresponding application should be affected. Most importantly it should not be possible for one application to negatively affect another
  - Applications might have lots of implicit assumptions about mode setting. Try to emulate the expected behavior in as many scenarios as possible
- Scale the output of corresponding applications to screen size using the viewporter protocol
  - Do not change the proportions - add black bars where necessary

Figure 3: Wayland design with XWayland



- Make sure input events, such as mouse clicks, work accordingly
- Make sure this works across disruptive events like real mode changes, minimization and re-maximization and - probably most difficult - in multi-monitor environments

3.2 Viewporter implementation in Mutter

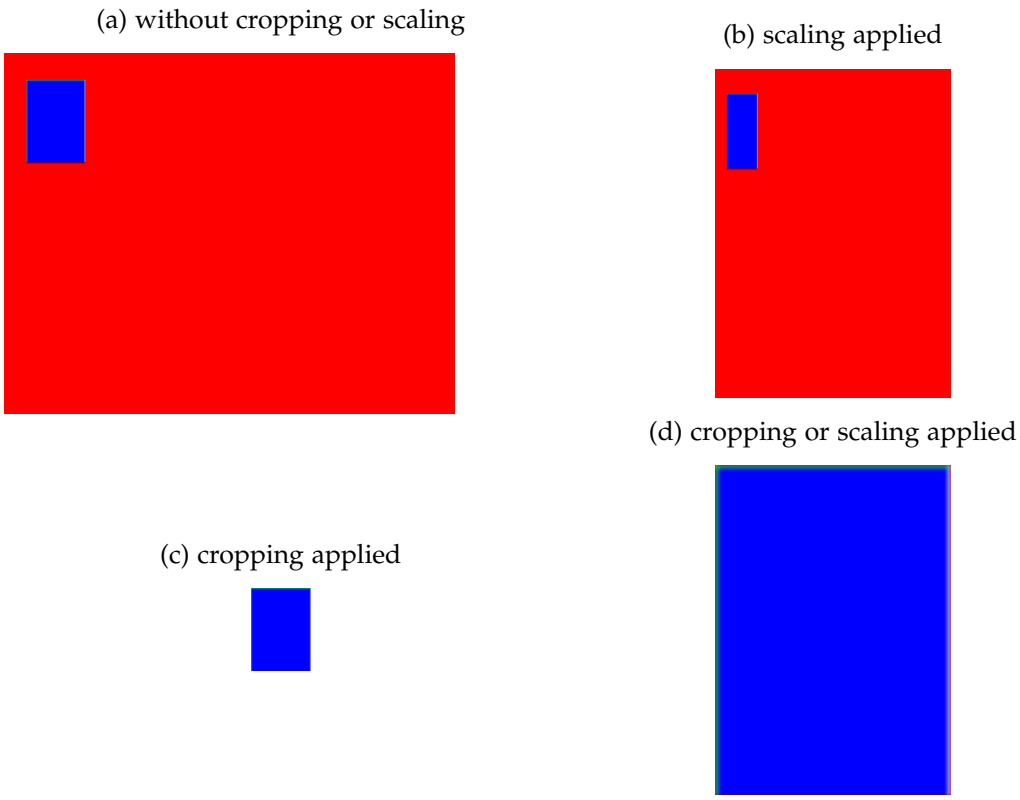
Weston, the Wayland reference implementation, is currently the only Wayland compositor implementing the viewporter protocol. Unfortunately it only implements very minimal set of XWM capabilities and many X11 applications do not even start when run in a Weston session with XWayland - especially those that use non-standard ways to archive fullscreen behavior, which happen to be the same that need mode setting in many cases. Further more the goal is to create a solution that can be used in real world scenarios as soon as possible.

I therefore conclude it to be a necessary prerequisite to implement the viewporter protocol in a compositor that:

- is commonly used by end users, implying it has relatively mature Wayland capabilities
- includes a XWM that is compatible to as many ways to archive fullscreen behavior as possible

This limited the options to the following three: Mutter of the Gnome project, KWin of the KDE project and Enlightenment. the decision is use Mutter, as its the one with the biggest user base and the most mature Wayland capabilities.

Figure 4: The test application weston-scaler for the viewporter protocol



The goal for the implementation is to be thorough and solid so the upstream project accepts it. It can also be assumed this will minimize distractions by unnecessary bugs in the mode setting emulation effort.

This intermediate step also orthogonally helps with the underlying motivation of accelerating Wayland adoption - the protocol provides a solid alternative for mode setting for projects that want to target Wayland directly.

4 Realization

In this part the most significant aspects of the work are to be highlighted and some insights are given into tricky problems.

4.1 Implementation of the viewporter protocol in Mutter

4.1.1 Porting old patches

The first step on the way for a proper implementation naturally was to look for previous attempts. It turned out there existed a patch-set [7] from 2014, which partly implemented a draft version of the viewporter protocol. It was based the Weston version from that time.

[7] https://gitlab.gnome.org/GNOME/mutter/tree/wip/viewport

I started porting it to the current Mutter code base and the stable release of the protocol. The basic underlying structure had partly changed quite fundamentally - for example protocol extensions now have to be implemented in a more formal way (compare [8] and [9]).

When finally building, the patches turned out to be fundamentally broken in many key aspects. Still, they provided a solid base in terms of general structure and there are several parts that remain nearly unchanged until now. I therefore started improving upon them.

4.1.2 Rendering

The most obvious aspect of the protocol is to modify the way how the content of a buffer is rendered on the screen. Without modifiers, the content of a buffer represent a 2D texture which only needs to be positioned on the screen and is then rendered pixel by pixel.

As of now there exist three buffer modifiers in the form of Wayland protocol extensions: viewports, buffer scales and transformations. Transformations are not yet implemented in Mutter, but they would allow to rotate surfaces in 90 degree steps. Buffer scales allow the dimensions of a buffer to be multiples of the output size and are used for high-dpi scenarios. They are already implemented.

These modifiers have to be applied in the right order to produce the desired outcome.

Additionally, the compositor[10] has the ability to reuse the last composed image and only repaint certain areas. This is referred to as partial drawing.

The preexisting patch set included an approach from before buffer scales where implemented, but even excluding that it was completely wrong and I ended up replacing it completely. This took me many iterations to come up with a clean solution and was subject for optimizations several times[11]

4.1.3 Surface damage

Similar to partial drawing within the compositor, there exists a similar optimization technique on protocol level. Instead of repainting the whole surface area, a client can signal which part of it changed. It can do so in surface coordinates or in buffer coordinates - with buffer scaling being the only implemented modifier, the difference was rather trivial before.

[8] https://gitlab.gnome.org/GNOME/mutter/commit/0829749049025c480d86bc122d02c97b5767a6b3#f0827862a44c095256a9234a20fb3e02a388a318

[9] https://gitlab.gnome.org/treba123/mutter/commit/e882e0be8ce6368ba803eb5186b30eecf7e2840d#706eb71e1fbf7f03bb236478492f46f25b3f8ccb

[10] in this context this refers to the component in Mutter which "composes" the final image how it is displayed on the screen by taking all outputs of all applications, arranging and then painting them accordingly

[11] compare: preexisting solution https://gitlab.gnome.org/GNOME/mutter/commit/725d0ad6804e1f49b404889509d3470aa93f6937#f0d3982b8d7215c98669a84b8570ff195c1e1bbb_216_235, new solution https://gitlab.gnome.org/treba123/mutter/commit/6d6cac4b4615fecb1d2209781718ad76b2167d74#f0d3982b8d7215c98669a84b8570ff195c1e1bbb_325_332

I therefore needed to implement a translation mechanism for the cases a client would choose surface coordinates. This part did not exist at all in the preexisting patch set and finding out about and understanding it was way harder than implementing the solution eventually.[12][13].

4.1.4 Bugs and incomplete implementations

A substantial amount of work turned out to be fixing existing bugs or completing previously sufficient but not complete implementation in the project. This includes:

Build system In order to start the implementation, I needed the build system to auto-generate a c-header file from the protocol specification, written in XML.

As it turned out, at that time `wp_viewporter` would have been the first stable protocol as all other protocols were marked unstable, implying a slightly different auto-generation process.

The existing code did not work for stable protocols and was incomplete for all protocols with multi-word names, requiring workarounds for the later. I therefore provided a solution sufficient to fix both issues [14], but it got replaced by an even cleaner approach from the Weston-project, which then got merged very early in the process.

Window geometry While the Wayland-base-protocol is highly generic and does not specify anything window related, there exists a sub-protocol called xdg-shell that is meant to provide functionality roughly similar to EWMH in X11-world. For example, there exists windows and they have a geometry:

> The window geometry of a window is its "visible bounds" from the user's perspective. Client-side decorations often have invisible portions like drop-shadows which should be ignored for the purposes of aligning, placing and constraining windows.[15]

As it turned out eventually, a critical part of the specification in one function - `set_window_geometry` - was ignored:

> When applied, the effective window geometry will be the set window geometry clamped to the bounding rectangle of the combined geometry of the surface of the xdg_surface and the associated subsurfaces.[16]

[12] https://gitlab.gnome.org/treba123/mutter/commit/9813d5179b032bb7ecfef98bfe1a82b30a9a1862#f0827862a44c095256a9234a20fb3e02a388a318_282_391

[13] https://gitlab.gnome.org/treba123/mutter/commit/9813d5179b032bb7ecfef98bfe1a82b30a9a1862#f0827862a44c095256a9234a20fb3e02a388a318_255_304

[14] https://bugzilla.gnome.org/show_bug.cgi?id=792203

[15] https://github.com/wayland-project/wayland-protocols/blob/3f282987d6e5cfd8d643886c5165d8a35141912a/stable/xdg-shell/xdg-shell.xml#L447

[16] https://github.com/wayland-project/wayland-protocols/blob/3f282987d6e5cfd8d643886c5165d8a35141912a/stable/xdg-shell/xdg-shell.xml#L472

This implies a simple check of the requested size against a (recursively combined) surface size. But it was left out, resulting in unwanted behavior as windows with very small surfaces could get treated as if they were much bigger by the WM. Ones figured, the fix was very straight forward[17]. This bug was especially hard for me to track down. There were several other patches in place that solved the problem partly before I finally uncovered the root problem. Therefore this solution made the implementation much smaller.

I further see this fix as especially valuable, as I suspect it to be necessary for the transformations protocol, too (apart from correctly handling exotic client applications).

Opaque regions A similarly tricky bug was a not existing check in the implementation of opaque regions. Opaque regions are another optional performance optimization technique that lets clients help compositors reducing work. Clients can define regions - sets of rectangles - on their surfaces that they do not plan to make transparent.

While this does not need to hold true in the future, current compositors often work in the way that surfaces get drawn bottom to top. That is very wasteful if the topmost surface hides big parts or even all of the underlying parts. But if a surface is transparent, the underlying part needs to be drawn first. On current generation hardware it is still cheaper to draw multiple times compared to first check every single surface on top for transparency, therefore it is useful for the compositor if clients simply let them know in advance where they can skip it.

The following part of the specification was left out:

> The compositor ignores the parts of the opaque region that fall outside of the surface.[18]

This resulted in broken rendering in one of the test applications[19]. As it is not possible to test it without viewporter support, it was not obvious the problem was not part of the implementation. After long research I finally found and fixed the issue[20].

4.2 Mode setting emulation

Dynamic mode-setting, that is changing the mode without restarting the X11-server and all attached programs, was not originally considered in the protocol. It got added via extensions later, first via the infamous vidmode, second in the early 2000s by RandR (Resize and Rotate). XWayland supports both of them and to make the mode setting emulation complete, vidmode will have to get considered at some point. For now I choose to only consider the much more common RandR.

[17] https://gitlab.gnome.org/GNOME/mutter/merge_requests/141

[18] wl_surface_interface Struct Reference https://people.freedesktop.org/~whot/wayland-doxygen/wayland/Server/structwl__surface__interface.html#a423a1dc45922dbf23d9acb7b42d118d5

[19] https://gitlab.gnome.org/GNOME/mutter/issues/132#note_169130

[20] https://gitlab.gnome.org/GNOME/mutter/merge_requests/148

> or xf86vidmode, because we've all made inescapable mistakes in life
> (Daniel Stone, Xorg and Wayland developer - on IRC)

4.2.1 Mode exposing

The first step was to make XWayland actually expose multiple modes over RandR. XWayland is designed not to use any hardware information by itself but only rely on the information it is given by the Wayland-compositor. Current practice is to have the compositor send exactly one mode - the currently used one. Furthermore XWayland currently ignores any further send modes. One X11 developer, Olivier Fourdan, kindly jumped in and - in a fairly hands-on manner - provided a patch to make XWayland expose all modes it receives [21]

With that done, it was easy to patch Mutter to expose some common modes that are required by several applications.[22]

4.2.2 Setting a mode - sorry it's all so awful

The next step was to expose the ability to actually set modes.

The RandR-protocol lets each application with the corresponding user-privileges change the mode, either by using the library libXRandR or the command-line tool xrandr. Crucially to understand forthcoming difficulties is the fact it is stateless. There is no daemon-like application keeping track which application sets a mode. Current implementations don not have any way relate a mode change to a window - and it gets worse.

At the time of writing, XWayland implements the RandR-protocol in version 1.0, using the function `rrSetConfig` to set a mode.

Initial attempts to modify it resulted in expected but undesirable behavior - if a mode was set, it would apply for all XWayland-clients, making them unusable or corrupt their output to a certain degree.

Discussing the issue with one of the main developers, he told me:

```
(17:24:13) ajax: robert_mader: i spent a bit of time looking at
    cleaning out the randr code the other day. basically ->
   rrSetConfig is a trap, don't use it.
(17:26:57) robert_mader: ah ok
(17:27:15) robert_mader: what would you propose? Implement
   randr 1.5 for XWayland?
(17:27:41) ajax: the 1.2 hooks should be all you really need
   for what you're doing i think
(17:28:44) robert_mader: Ok
(17:37:15) robert_mader: ajax: thanks btw. :)
(17:37:57) ajax: np, sorry it's all so awful
```

The proposed solution was pretty straight forward[23] and had a very pleasant result: due to the different API, applications are now able to call mode changes "successfully" without changing any state within XWayland. This is a very crucial step as

[21] https://patchwork.freedesktop.org/patch/191035/
[22] https://gitlab.gnome.org/treba123/mutter/commit/021e80717634bb225722678ff4c6d3e85e5ebf0e
[23] https://bugs.freedesktop.org/show_bug.cgi?id=104644

it already solves the most fundamental problem: applications requiring mode setting to start at all now do so. Their graphical output doesn not yet get scaled, therefore being small on high resolution displays. But they do not fail completely. As I see that as a major step already, I am hosting packages for the fedora distribution[24] containing the required patches.

A second important effect was that there seems to be no way in `rrSetConfig` to get a list of client windows without make very work intensive modifications all over XWayland. One reason for the quote above. The XRandR 1.2 hook in turn makes it easy - an important detail further down the road.

4.2.3 Scaling

Having a quite solid base - viewporter support in Mutter and non-disruptive mode setting in XWayland - I had all prerequisites to finally start with the actual experimentation: scaling the output of applications.

The main areas I started working on so far are:

Window attribution This is what I consider the most difficult step in the whole effort to succeed. There has to be made a connection between a mode change and the window of the application that requested it.

As the current APIs do not allow direct attributions, I am investigating possibilities for indirect ways. Test based on interpretation of X11 window events - e.g. window size change events - have proven to be promising, but the balance between false positives and false negatives remains hard.

Setting viewports Given a successful window attribution, the basic concept of setting a viewport works well. While occasionally facing rendering issues at the beginning, it helped me ironing out bugs in the viewporter implementation.

Keeping the ration As resolutions can have different width-to-height ratios than the real resolution, distortion of the image should avoided by adding black bars - like in movies. This happens to be quite easy using subsurfaces[25]. - especially if the scaled up resolution is not a power of two multiplied While this was successfully tested in Weston, in Mutter bars positioned on top or left of the application surface are displayed above it instead, hiding parts of of the content. This has not been fixed yet.

Hardware cursor correction From the perspective of this context, there are basically two groups of applications concerning cursor input: those that don not use hardware cursors and those that do.

The difference between them is basically that in the first scenario, the application is responsible for drawing the cursor, while in the second, the cursor is drawn

[24] https://copr.fedorainfracloud.org/coprs/treba/xwayland-list-modes/

[25] https://people.freedesktop.org/~whot/wayland-doxygen/wayland/Server/group__iface__wl__subsurface.html

independently, making it possible to freely move the cursor around even when the application is stuttering or frozen.

While the first scenario is not very desirable from the user perspective, it is not problematic in this context. The cursor is part of the surface that is scaled up, the application draws the cursor where it thinks it should be, everything is fine.

The second group is problematic: as the cursor is not part of the surface that is scaled up, it remains unscaled. That can be a problem on very high DPI displays, as the cursor could become to small. Secondly and much more important: as the application surface is scaled, the position of the cursor as calculated by the application and how its drawn on the screen diverge.

This becomes visible when click-events happen, as the application interprets the coordinates the way as if it was not scaled.

The naive solution for the second problem is to correct cursor event coordinates accordingly in XWayland. While this should be sufficient in theory, it turned out to be more complex than initially thought and remains bug prone at the moment.

XWM compatibility The XWM part of Mutter has a couple of sophisticated methods to determine when it should consider an application to be in fullscreen mode. The XWM does not actually "know" that itself is the WC. While being the same implementation, it is a client of XWayland, which in turn is a client of Mutter. Mutter ru

5 Status and outlook

5.1 Viewporter implementation

A number of prerequisite changes are in the process [26] or pending[27][28] review, as well as the implementation itself[29]. While an in-depth review by the maintainers did not happen, yet, and will most probably unveil certain shortcomings, I am confident they will be of simple nature.

All test applications I am currently aware of - the reference tests from Weston[30] and a demo-frontend for the gstreamer-multimedia-framework [31] - were tested successfully to render correctly.

[26] https://gitlab.gnome.org/GNOME/mutter/merge_requests/141
[27] https://gitlab.gnome.org/GNOME/mutter/merge_requests/148
[28] https://gitlab.gnome.org/GNOME/mutter/merge_requests/149
[29] https://gitlab.gnome.org/GNOME/mutter/merge_requests/121
[30] weston-scaler and weston-simple-damage, both part of the Weston project, https://github.com/wayland-project/weston/tree/master/clients
[31] https://cgit.freedesktop.org/gstreamer/gst-plugins-bad/tree/tests/examples/waylandsink

5.2 Mode setting emulation

The ability to expose and set modes is pending review[32][33]. This will probably not happen before the Xorg-project finishes its current migration from cgit and bugzilla to gitlab[34]

Most of the remaining goals were archived on a proof-of-concept level, partly from early on[35], but far from being feature-complete enough to get merged upstream. While relatively functional in simple environments, work on complex scenarios like multi-monitor environments was not even started yet and I will continue to work on it.

6 Evaluation

6.1 Approach

While it is not yet possible to evaluate if the approach will lead to a sufficient solution, I am very satisfied with the immediate steps. There have been no major road-blockers and the approach did not have be shifted in a fundamental way. I attribute this mostly to the careful design of the Wayland-protocol, Mutter and XWayland.

6.2 Result

I value the implementation of the viewporter protocol in Mutter as the main accomplishment. Ironically, that is the exact counter part to the original goal: instead of improving compatibility with legacy applications that will not get ported to use Wayland, the result helps making it easier to create or port Wayland applications in the first place - an odd but pleasant side effect.

Apart from that the results remains far below the initial goals - a clear sign I have underestimated the needed effort and/or overestimated my capabilities.

My conclusion is I should have limited the scope of the thesis to a more specific part, probably the implementation of the viewporter protocol

6.3 Personal experience

The process of understanding the internals of the projects was mostly auto-didactic - trial and error were significant parts of the process. While the community - mostly consisting of highly paid specialist, chatting on IRC while working[36][37][38] - was very helpful and welcoming, proper mentoring, let alone an in depth introduction into or real documentation of the existing code would probably have helped archiving the technical goals.

[32] https://bugs.freedesktop.org/show_bug.cgi?id=104644

[33] My copr-repository with patched Mutter and XWayland packages for the current Fedora-Version https://copr.fedorainfracloud.org/coprs/treba/xwayland-list-modes/

[34] https://lists.freedesktop.org/archives/wayland-devel/2018-May/038100.html

[35] https://bugs.freedesktop.org/show_bug.cgi?id=104643#c2

[36] irc://irc.gnome.org/gnome-shell

[37] irc://freenode.net/wayland

[38] irc://freenode.net/xorg-devel

I knew this from the beginning and to jump into this kind of project nevertheless was a conscious decision. For me, most aspects of this project were new - the size and development processes of the projects, the technical concepts and protocols, the community, the language ("self-explaining" - read: undocumented - low-level C-code). Therefore it has has been a dense learning experience and, as the process required me to dive deep into way more areas than materialized in this thesis, I now feel way more competent in the general area of low-level graphics development.

A Appendix

A.1 Viewporter

```
From 39aff4bb0cc537f34bf84dbe2c5e9f64d2e36aa5 Mon Sep 17
   00:00:00 2001
From: Robert Mader <robert.mader@posteo.de>
Date: Mon, 2 Jul 2018 11:40:17 +0200
Subject: [PATCH 1/4] implement wp_viewporter: add meta-wayland-
   viewporter.c/h

---
 src/Makefile.am                          |   4 +
 src/wayland/meta-wayland-viewporter.c | 213
    ++++++++++++++++++++++++
 src/wayland/meta-wayland-viewporter.h |  28 ++++
 3 files changed, 245 insertions(+)
 create mode 100644 src/wayland/meta-wayland-viewporter.c
 create mode 100644 src/wayland/meta-wayland-viewporter.h

diff --git a/src/Makefile.am b/src/Makefile.am
index 82240ef..c82e8a5 100644
--- a/src/Makefile.am
+++ b/src/Makefile.am
@@ -90,6 +90,8 @@ mutter_built_sources += \
  xwayland-keyboard-grab-unstable-v1-server-protocol.h    \
  gtk-text-input-protocol.c         \
  gtk-text-input-server-protocol.h        \
+ viewporter-protocol.c           \
+ viewporter-server-protocol.h          \
  $(NULL)

 if HAVE_WAYLAND_EGLSTREAM
@@ -481,6 +483,8 @@
   libmutter_@LIBMUTTER_API_VERSION@_la_SOURCES += \
  wayland/meta-wayland-inhibit-shortcuts-dialog.h \
  wayland/meta-xwayland-grab-keyboard.c   \
  wayland/meta-xwayland-grab-keyboard.h   \
+ wayland/meta-wayland-viewporter.c \
+ wayland/meta-wayland-viewporter.h \
  $(NULL)
 endif

diff --git a/src/wayland/meta-wayland-viewporter.c b/src/
   wayland/meta-wayland-viewporter.c
new file mode 100644
index 0000000..9c1f2d1
--- /dev/null
+++ b/src/wayland/meta-wayland-viewporter.c
@@ -0,0 +1,213 @@
+/*
+ * Wayland Support
```

```
+ *
+ * This program is free software; you can redistribute it and/
    or
+ * modify it under the terms of the GNU General Public License
     as
+ * published by the Free Software Foundation; either version 2
     of the
+ * License, or (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be
    useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
    the GNU
+ * General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public
    License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
+ * 02111-1307, USA.
+ *
+ */
+
+#include <glib.h>
+#include "meta-wayland-viewporter.h"
+#include "meta-wayland-versions.h"
+#include "meta-wayland-surface.h"
+#include "meta-wayland-subsurface.h"
+#include "meta-wayland-private.h"
+#include "viewporter-server-protocol.h"
+
+static void
+destroy_wl_viewport (struct wl_resource *resource)
+{
+  MetaWaylandSurface *surface = wl_resource_get_user_data (
    resource);
+
+  if(!surface)
+    return;
+
+  surface->viewport_resource = NULL;
+
+  if(!surface->pending)
+    return;
+  surface->pending->buffer_viewport.buffer.transform =
    WL_OUTPUT_TRANSFORM_NORMAL;
+  surface->pending->buffer_viewport.buffer.scale = 1;
+  surface->pending->buffer_viewport.buffer.src_rect.x = 0;
+  surface->pending->buffer_viewport.buffer.src_rect.y = 0;
+  surface->pending->buffer_viewport.buffer.src_rect.width = 0;
```

```
+  surface->pending->buffer_viewport.buffer.src_rect.height =
   0;
+  surface->pending->buffer_viewport.surface.width = 0;
+  surface->pending->buffer_viewport.surface.height = 0;
+  surface->pending->buffer_viewport.changed = TRUE;
+}
+
+static void
+viewport_destroy (struct wl_client *client,
+                  struct wl_resource *resource)
+{
+  wl_resource_destroy (resource);
+}
+
+static void
+viewport_set_source (struct wl_client *client,
+                     struct wl_resource *resource,
+                     wl_fixed_t src_x,
+                     wl_fixed_t src_y,
+                     wl_fixed_t src_width,
+                     wl_fixed_t src_height)
+{
+  MetaWaylandSurface *surface = wl_resource_get_user_data (
   resource);
+  if(!surface)
+    {
+      wl_resource_post_error (resource,
+                              WP_VIEWPORT_ERROR_NO_SURFACE,
+                              "wl_surface for this viewport is
    no longer exists");
+      return;
+    }
+
+  int x = floor(wl_fixed_to_double (src_x));
+  int y = floor(wl_fixed_to_double (src_y));
+  int width = ceil(wl_fixed_to_double (src_width));
+  int height = ceil(wl_fixed_to_double (src_height));
+
+  if(x >= 0 && y >= 0 && width > 0 && height > 0)
+    {
+      surface->pending->buffer_viewport.buffer.src_rect.x = x;
+      surface->pending->buffer_viewport.buffer.src_rect.y = y;
+      surface->pending->buffer_viewport.buffer.src_rect.width
   = width;
+      surface->pending->buffer_viewport.buffer.src_rect.height
    = height;
+      surface->pending->buffer_viewport.changed = TRUE;
+    }
+  else if(x == -1 && y == -1 && width == -1 && height == -1)
+    {
+      surface->pending->buffer_viewport.buffer.src_rect.x = 0;
+      surface->pending->buffer_viewport.buffer.src_rect.y = 0;
```

```
+       surface->pending->buffer_viewport.buffer.src_rect.width
    = 0;
+       surface->pending->buffer_viewport.buffer.src_rect.height
     = 0;
+       surface->pending->buffer_viewport.changed = TRUE;
+     }
+   else
+     {
+       wl_resource_post_error (resource,
+                               WP_VIEWPORT_ERROR_BAD_VALUE,
+                               "all values must be either
    positive or -1");
+     }
+}
+
+static void
+viewport_set_destination (struct wl_client *client,
+                          struct wl_resource *resource,
+                          int dst_width,
+                          int dst_height)
+{
+  MetaWaylandSurface *surface = wl_resource_get_user_data (
    resource);
+  if(!surface)
+    {
+      wl_resource_post_error (resource,
+                              WP_VIEWPORT_ERROR_NO_SURFACE,
+                              "wl_surface for this viewport is
     no longer exists");
+      return;
+    }
+
+  if(dst_width > 0 && dst_height > 0)
+    {
+      surface->pending->buffer_viewport.surface.width =
    dst_width;
+      surface->pending->buffer_viewport.surface.height =
    dst_height;
+      surface->pending->buffer_viewport.changed = TRUE;
+    }
+  else if(dst_width == -1 && dst_height == -1)
+    {
+      surface->pending->buffer_viewport.surface.width = 0;
+      surface->pending->buffer_viewport.surface.height = 0;
+      surface->pending->buffer_viewport.changed = TRUE;
+    }
+  else
+    {
+      wl_resource_post_error (resource,
+                              WP_VIEWPORT_ERROR_BAD_VALUE,
+                              "all values must be either
    positive or -1");
```

```
+    }
+}
+
+static const struct wp_viewport_interface
   meta_wayland_viewport_interface = {
+  viewport_destroy,
+  viewport_set_source,
+  viewport_set_destination,
+};
+
+static void
+viewporter_destroy (struct wl_client *client,
+                    struct wl_resource *resource)
+{
+  wl_resource_destroy (resource);
+}
+
+static void
+viewporter_get_viewport (struct wl_client *client,
+                         struct wl_resource *master_resource,
+                         uint32_t viewport_id,
+                         struct wl_resource *surface_resource)
+{
+  struct wl_resource *resource;
+  MetaWaylandSurface *surface = wl_resource_get_user_data (
   surface_resource);
+
+  if (surface->viewport_resource)
+    {
+      wl_resource_post_error (master_resource,
+
   WP_VIEWPORTER_ERROR_VIEWPORT_EXISTS,
+                              "viewport already exists on
   surface");
+      return;
+    }
+
+  resource = wl_resource_create (client,
+                                 &wp_viewport_interface,
+                                 wl_resource_get_version (
   master_resource),
+                                 viewport_id);
+  wl_resource_set_implementation (resource,
+                                  &
   meta_wayland_viewport_interface,
+                                  surface,
+                                  destroy_wl_viewport);
+
+  surface->viewport_resource = resource;
+}
+
```

```
+static const struct wp_viewporter_interface
    meta_wayland_viewporter_interface = {
+  viewporter_destroy,
+  viewporter_get_viewport,
+};
+
+static void
+bind_viewporter (struct wl_client *client,
+                 void             *data,
+                 guint32           version,
+                 guint32           id)
+{
+  struct wl_resource *resource;
+
+  resource = wl_resource_create (client,
+                                 &wp_viewporter_interface,
+                                 version,
+                                 id);
+  wl_resource_set_implementation (resource,
+                                  &
    meta_wayland_viewporter_interface,
+                                  data,
+                                  NULL);
+}
+
+void
+meta_wayland_viewporter_init (MetaWaylandCompositor *
    compositor)
+{
+  if (wl_global_create (compositor->wayland_display,
+                        &wp_viewporter_interface,
+                        META_WP_VIEWPORTER_VERSION,
+                        compositor, bind_viewporter) == NULL)
+  g_error ("Failed to register a global wl-subcompositor
    object");
+}
diff --git a/src/wayland/meta-wayland-viewporter.h b/src/
    wayland/meta-wayland-viewporter.h
new file mode 100644
index 0000000..6927f09
--- /dev/null
+++ b/src/wayland/meta-wayland-viewporter.h
@@ -0,0 +1,28 @@
+/*
+ * Wayland Support
+ *
+ * This program is free software; you can redistribute it and/
    or
+ * modify it under the terms of the GNU General Public License
     as
+ * published by the Free Software Foundation; either version 2
     of the
```

```
+ * License, or (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be
    useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
    the GNU
+ * General Public License for more details.
+ *
+ * You should have received a copy of the GNU General Public
    License
+ * along with this program; if not, write to the Free Software
+ * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA
+ * 02111-1307, USA.
+ *
+ */
+
+#ifndef META_WAYLAND_VIEWPORTER_H
+#define META_WAYLAND_VIEWPORTER_H
+
+#include "wayland/meta-wayland-types.h"
+
+void meta_wayland_viewporter_init (MetaWaylandCompositor *
    compositor);
+
+#endif /* META_WAYLAND_VIEWPORTER_H */
--
2.17.1
```

```
From 0942b2f427dbad67e9cd180c820bbd71b8c0cf01 Mon Sep 17
    00:00:00 2001
From: Robert Mader <robert.mader@posteo.de>
Date: Mon, 2 Jul 2018 12:06:56 +0200
Subject: [PATCH 2/4] implement wp_viewporter: compositor
    changes

---
 src/compositor/meta-shaped-texture-private.h |   5 +
 src/compositor/meta-shaped-texture.c         | 134
     ++++++++++++++++----
 src/compositor/meta-surface-actor.c          |  11 ++
 src/compositor/meta-surface-actor.h          |   5 +
 4 files changed, 132 insertions(+), 23 deletions(-)

diff --git a/src/compositor/meta-shaped-texture-private.h b/src
    /compositor/meta-shaped-texture-private.h
index 5b3f283..7ceac55 100644
--- a/src/compositor/meta-shaped-texture-private.h
+++ b/src/compositor/meta-shaped-texture-private.h
@@ -41,5 +41,10 @@ void meta_shaped_texture_set_fallback_size (
    MetaShapedTexture *stex,
                                            guint
```

```
                                                    fallback_height
                                                    );
 gboolean meta_shaped_texture_is_obscured (MetaShapedTexture *
    self);
 cairo_region_t * meta_shaped_texture_get_opaque_region (
    MetaShapedTexture *stex);
+void meta_shaped_texture_set_viewport (MetaShapedTexture
        *stex,
+                                       cairo_rectangle_int_t
     *src_rect,
+                                       int
                        dest_width,
+                                       int
                        dest_height,
+                                       int
                        scale);

 #endif
diff --git a/src/compositor/meta-shaped-texture.c b/src/
   compositor/meta-shaped-texture.c
index 0240c06..6f20707 100644
--- a/src/compositor/meta-shaped-texture.c
+++ b/src/compositor/meta-shaped-texture.c
@@ -106,8 +106,14 @@ struct _MetaShapedTexturePrivate
   cairo_region_t *clip_region;
   cairo_region_t *unobscured_region;

+  /* Viewport stuff */
+  cairo_rectangle_int_t viewport_src_rect;
+  guint viewport_dest_width, viewport_dest_height;
+  guint viewport_scale;
+
   guint tex_width, tex_height;
   guint fallback_width, fallback_height;
+  guint dest_width, dest_height;

   gint64 prev_invalidation, last_invalidation;
   guint fast_updates;
@@ -153,6 +159,7 @@ meta_shaped_texture_init (MetaShapedTexture
    *self)
   priv->mask_texture = NULL;
   priv->create_mipmaps = TRUE;
   priv->is_y_inverted = TRUE;
+  priv->viewport_scale = 1;
 }

 static void
@@ -168,8 +175,8 @@ set_unobscured_region (MetaShapedTexture *
   self,

       if (priv->texture)
```

```
         {
-          width = priv->tex_width;
-          height = priv->tex_height;
+          width = priv->dest_width;
+          height = priv->dest_height;
         }
       else
         {
@@ -325,20 +332,43 @@ static void
 paint_clipped_rectangle (CoglFramebuffer       *fb,
                          CoglPipeline          *pipeline,
                          cairo_rectangle_int_t *rect,
-                         ClutterActorBox       *alloc)
+                         ClutterActorBox       *alloc,
+                         MetaShapedTexture     *stex)
 {
+  MetaShapedTexturePrivate *priv = stex->priv;
   float coords[8];
   float x1, y1, x2, y2;
+  float src_x, src_y, src_width, src_height;

   x1 = rect->x;
   y1 = rect->y;
   x2 = rect->x + rect->width;
   y2 = rect->y + rect->height;

-  coords[0] = rect->x / (alloc->x2 - alloc->x1);
-  coords[1] = rect->y / (alloc->y2 - alloc->y1);
-  coords[2] = (rect->x + rect->width) / (alloc->x2 - alloc->x1
   );
-  coords[3] = (rect->y + rect->height) / (alloc->y2 - alloc->
   y1);
+  if(priv->viewport_src_rect.width > 0)
+    {
+      src_x      = priv->viewport_src_rect.x      * priv->
   viewport_scale;
+      src_y      = priv->viewport_src_rect.y      * priv->
   viewport_scale;
+      src_width  = priv->viewport_src_rect.width  * priv->
   viewport_scale;
+      src_height = priv->viewport_src_rect.height * priv->
   viewport_scale;
+    }
+  else
+    {
+      src_x      = 0;
+      src_y      = 0;
+      src_width  = priv->tex_width;
+      src_height = priv->tex_height;
+    }
+
+  coords[0] = rect->x      * src_width  / (alloc->x2 - alloc->
```

```
    x1) + src_x;
+   coords[1] = rect->y      * src_height / (alloc->y2 - alloc->
    y1) + src_y;
+   coords[2] = rect->width  * src_width  / (alloc->x2 - alloc->
    x1) + coords[0];
+   coords[3] = rect->height * src_height / (alloc->y2 - alloc->
    y1) + coords[1];
+
+   coords[0] /= priv->tex_width;
+   coords[1] /= priv->tex_height;
+   coords[2] /= priv->tex_width;
+   coords[3] /= priv->tex_height;

    coords[4] = coords[0];
    coords[5] = coords[1];
@@ -350,6 +380,38 @@ paint_clipped_rectangle (CoglFramebuffer
          *fb,
                                                  &coords[0],
                                                     8);
 }

+static void
+update_size (MetaShapedTexture *stex)
+{
+   MetaShapedTexturePrivate *priv = stex->priv;
+   guint dest_width, dest_height;
+
+   if (priv->viewport_dest_width > 0)
+     {
+       dest_width = priv->viewport_dest_width * priv->
    viewport_scale;
+       dest_height = priv->viewport_dest_height * priv->
    viewport_scale;
+     }
+   else if(priv->viewport_src_rect.width > 0)
+     {
+       dest_width = priv->viewport_src_rect.width * priv->
    viewport_scale;
+       dest_height = priv->viewport_src_rect.height * priv->
    viewport_scale;
+     }
+   else
+     {
+       dest_width = priv->tex_width;
+       dest_height = priv->tex_height;
+     }
+
+   if (priv->dest_width != dest_width ||
+       priv->dest_height != dest_height)
+     {
+       priv->dest_width = dest_width;
+       priv->dest_height = dest_height;
```

```
+      clutter_actor_queue_relayout (CLUTTER_ACTOR (stex));
+      g_signal_emit (stex, signals[SIZE_CHANGED], 0);
+    }
+}
+
 static void
 set_cogl_texture (MetaShapedTexture *stex,
                   CoglTexture       *cogl_tex)
@@ -384,8 +446,7 @@ set_cogl_texture (MetaShapedTexture *stex,
       priv->tex_width = width;
       priv->tex_height = height;
       meta_shaped_texture_set_mask_texture (stex, NULL);
-      clutter_actor_queue_relayout (CLUTTER_ACTOR (stex));
-      g_signal_emit (stex, signals[SIZE_CHANGED], 0);
+      update_size (stex);
     }

   /* NB: We don't queue a redraw of the actor here because we
      don't
@@ -417,7 +478,7 @@ meta_shaped_texture_paint (ClutterActor *
   actor)
 {
   MetaShapedTexture *stex = (MetaShapedTexture *) actor;
   MetaShapedTexturePrivate *priv = stex->priv;
-  guint tex_width, tex_height;
+  guint dest_width, dest_height;
   guchar opacity;
   CoglContext *ctx;
   CoglFramebuffer *fb;
@@ -476,13 +537,13 @@ meta_shaped_texture_paint (ClutterActor *
       }
     }

-  tex_width = priv->tex_width;
-  tex_height = priv->tex_height;
+  dest_width = priv->dest_width;
+  dest_height = priv->dest_height;

-  if (tex_width == 0 || tex_height == 0) /* no contents yet */
+  if (dest_width == 0 || dest_height == 0) /* no contents yet
   */
     return;

-  cairo_rectangle_int_t tex_rect = { 0, 0, tex_width,
   tex_height };
+  cairo_rectangle_int_t tex_rect = { 0, 0, dest_width,
   dest_height };

   /* Use nearest-pixel interpolation if the texture is
      unscaled. This
    * improves performance, especially with software rendering.
```

```
@@ -490,7 +551,7 @@ meta_shaped_texture_paint (ClutterActor *
    actor)

    filter = COGL_PIPELINE_FILTER_LINEAR;

-   if (meta_actor_painting_untransformed (tex_width, tex_height
    , NULL, NULL))
+   if (meta_actor_painting_untransformed (dest_width,
    dest_height, NULL, NULL))
      filter = COGL_PIPELINE_FILTER_NEAREST;

    ctx = clutter_backend_get_cogl_context (
       clutter_get_default_backend ());
@@ -565,7 +626,11 @@ meta_shaped_texture_paint (ClutterActor *
    actor)
              {
                cairo_rectangle_int_t rect;
                cairo_region_get_rectangle (region, i, &rect);
-               paint_clipped_rectangle (fb, opaque_pipeline, &
    rect, &alloc);
+               paint_clipped_rectangle (fb,
+                                        opaque_pipeline,
+                                        &rect,
+                                        &alloc,
+                                        stex);
              }
          }

@@ -618,16 +683,21 @@ meta_shaped_texture_paint (ClutterActor *
    actor)
                if (!gdk_rectangle_intersect (&tex_rect, &rect,
                   &rect))
                  continue;

-               paint_clipped_rectangle (fb, blended_pipeline, &
    rect, &alloc);
+               paint_clipped_rectangle (fb,
+                                        blended_pipeline,
+                                        &rect,
+                                        &alloc,
+                                        stex);
              }
          }
        else
          {
            /* 3) blended_region is NULL. Do a full paint. */
-           cogl_framebuffer_draw_rectangle (fb,
    blended_pipeline,
-                                            0, 0,
-                                            alloc.x2 - alloc.x1
    ,
-                                            alloc.y2 - alloc.y1
```

```
   );
+          paint_clipped_rectangle (fb,
+                                   blended_pipeline,
+                                   &tex_rect,
+                                   &alloc,
+                                   stex);
         }
     }

@@ -645,7 +715,7 @@ meta_shaped_texture_get_preferred_width (
   ClutterActor *self,
   guint width;

   if (priv->texture)
-    width = priv->tex_width;
+    width = priv->dest_width;
   else
     width = priv->fallback_width;

@@ -665,7 +735,7 @@ meta_shaped_texture_get_preferred_height (
   ClutterActor *self,
   guint height;

   if (priv->texture)
-    height = priv->tex_height;
+    height = priv->dest_height;
   else
     height = priv->fallback_height;

@@ -964,6 +1034,24 @@ meta_shaped_texture_get_opaque_region (
   MetaShapedTexture *stex)
   return priv->opaque_region;
 }

+void
+meta_shaped_texture_set_viewport (MetaShapedTexture
   *stex,
+                                  cairo_rectangle_int_t
   *src_rect,
+                                  int
   dest_width,
+                                  int
   dest_height,
+                                  int
   scale)
+{
+  MetaShapedTexturePrivate *priv = stex->priv;
+
+  priv->viewport_src_rect = *src_rect;
+  priv->viewport_scale = scale;
+
+  priv->viewport_dest_width = dest_width;
```

```
+  priv->viewport_dest_height = dest_height;
+  update_size (stex);
+  clutter_actor_queue_redraw (CLUTTER_ACTOR (stex));
+}
+
 /**
  * meta_shaped_texture_get_image:
  * @stex: A #MetaShapedTexture
diff --git a/src/compositor/meta-surface-actor.c b/src/
    compositor/meta-surface-actor.c
index bf8c76f..e7bcce0 100644
--- a/src/compositor/meta-surface-actor.c
+++ b/src/compositor/meta-surface-actor.c
@@ -242,6 +242,17 @@ meta_surface_actor_get_opaque_region (
    MetaSurfaceActor *actor)
    return meta_shaped_texture_get_opaque_region (priv->texture)
       ;
 }

+void
+meta_surface_actor_set_viewport (MetaSurfaceActor          *
    self,
+                                 cairo_rectangle_int_t    *
    src_rect,
+                                 int
    dest_width,
+                                 int
    dest_height,
+                                 int
    scale)
+{
+  MetaSurfaceActorPrivate *priv = self->priv;
+  meta_shaped_texture_set_viewport (priv->texture, src_rect,
    dest_width, dest_height, scale);
+}
+
 static gboolean
 is_frozen (MetaSurfaceActor *self)
 {
diff --git a/src/compositor/meta-surface-actor.h b/src/
    compositor/meta-surface-actor.h
index 8c6dda2..2acbdc2 100644
--- a/src/compositor/meta-surface-actor.h
+++ b/src/compositor/meta-surface-actor.h
@@ -62,6 +62,11 @@ void meta_surface_actor_set_opaque_region (
    MetaSurfaceActor *self,
                                           cairo_region_t   *
                                              region);
 cairo_region_t * meta_surface_actor_get_opaque_region (
     MetaSurfaceActor *self);

+void meta_surface_actor_set_viewport (MetaSurfaceActor
```

```
    *self,
+                                       cairo_rectangle_int_t
    *src_rect,
+                                       int
                          dest_width,
+                                       int
                          dest_height,
+                                       int
                          scale);
 void meta_surface_actor_process_damage (MetaSurfaceActor *
     actor,
                                         int x, int y, int
                                            width, int height);
 void meta_surface_actor_pre_paint (MetaSurfaceActor *actor);
--
2.17.1
```

```
From bb0aab8139eebb514744b9b3030a797195f3eded Mon Sep 17
    00:00:00 2001
From: Robert Mader <robert.mader@posteo.de>
Date: Mon, 2 Jul 2018 15:01:36 +0200
Subject: [PATCH 3/4] implement wp_viewporter: meta-wayland-
    surface

---
 src/wayland/meta-wayland-surface.c | 144
     ++++++++++++++++++++++++++++-
 src/wayland/meta-wayland-surface.h |   1 +
 2 files changed, 142 insertions(+), 3 deletions(-)

diff --git a/src/wayland/meta-wayland-surface.c b/src/wayland/
    meta-wayland-surface.c
index 2c7b597..e85df33 100644
--- a/src/wayland/meta-wayland-surface.c
+++ b/src/wayland/meta-wayland-surface.c
@@ -46,6 +46,7 @@
 #include "meta-wayland-legacy-xdg-shell.h"
 #include "meta-wayland-wl-shell.h"
 #include "meta-wayland-gtk-shell.h"
+#include "meta-wayland-viewporter.h"

 #include "meta-cursor-tracker-private.h"
 #include "display-private.h"
@@ -251,6 +252,114 @@ meta_wayland_surface_assign_role (
    MetaWaylandSurface *surface,
     }
 }

+static void
+meta_wayland_surface_surface_to_buffer_coordinate (
    MetaWaylandSurface *surface,
+                                                   float sx,
```

```
    float sy,
+                                                   float *bx,
    float *by)
+{
+  MetaWaylandBuffer *buffer = surface->buffer_ref.buffer;
+  MetaWaylandBufferViewport *vp = &surface->buffer_viewport;
+  double src_width, src_height;
+  double src_x, src_y;
+  double surface_width, surface_height;
+
+  if (vp->buffer.src_rect.width == 0)
+    {
+      if (vp->surface.width == 0)
+        {
+          *bx = sx;
+          *by = sy;
+          return;
+        }
+
+      src_x = 0.0;
+      src_y = 0.0;
+      src_width  = (double)cogl_texture_get_width (buffer->
    texture);
+      src_height = (double)cogl_texture_get_height (buffer->
    texture);
+      surface_width  = src_width  / vp->buffer.scale;
+      surface_height = src_height / vp->buffer.scale;
+    }
+  else {
+    if (vp->surface.width == 0)
+      {
+        surface_width  = (double)cogl_texture_get_width (
    buffer->texture)  / vp->buffer.scale;
+        surface_height = (double)cogl_texture_get_height (
    buffer->texture) / vp->buffer.scale;
+      }
+    else
+      {
+        surface_width  = (double)vp->surface.width;
+        surface_height = (double)vp->surface.height;
+      }
+    src_x = (double)vp->buffer.src_rect.x;
+    src_y = (double)vp->buffer.src_rect.y;
+    src_width  = (double)vp->buffer.src_rect.width;
+    src_height = (double)vp->buffer.src_rect.height;
+  }
+
+  *bx = sx * src_width  / surface_width  + src_x;
+  *by = sy * src_height / surface_height + src_y;
+}
+
+static cairo_region_t *
```

```
+meta_wayland_surface_surface_to_buffer_region (
   MetaWaylandSurface *surface,
+                                               cairo_region_t
   *region)
+{
+  MetaWaylandBuffer *buffer = surface->buffer_ref.buffer;
+  MetaWaylandBufferViewport *vp = &surface->buffer_viewport;
+  int n_rects, i;
+  cairo_rectangle_int_t *rects;
+  cairo_rectangle_int_t surface_rect;
+  cairo_region_t *scaled_region;
+  float xf, yf;
+
+  if (vp->buffer.scale == 1 &&
+      vp->buffer.src_rect.width == 0 &&
+      vp->surface.width == 0)
+    return cairo_region_copy (region);
+
+  n_rects = cairo_region_num_rectangles (region);
+
+  rects = g_malloc (sizeof(cairo_rectangle_int_t) * n_rects);
+
+  for (i = 0; i < n_rects; i++)
+    {
+      cairo_region_get_rectangle (region, i, &rects[i]);
+
+      // use coordinates instead of width/height
+      rects[i].width  += rects[i].x;
+      rects[i].height += rects[i].y;
+
+      meta_wayland_surface_surface_to_buffer_coordinate (
   surface, rects[i].x,
+                                                         rects
   [i].y, &xf, &yf);
+      rects[i].x = floorf(xf * vp->buffer.scale);
+      rects[i].y = floorf(yf * vp->buffer.scale);
+
+      meta_wayland_surface_surface_to_buffer_coordinate (
   surface, rects[i].width,
+                                                         rects
   [i].height, &xf, &yf);
+      rects[i].width  = ceilf(xf * vp->buffer.scale);
+      rects[i].height = ceilf(yf * vp->buffer.scale);
+
+      // use width/height again instead of coordinates
+      rects[i].width  -= rects[i].x;
+      rects[i].height -= rects[i].y;
+    }
+
+  scaled_region = cairo_region_create_rectangles (rects,
   n_rects);
+
```

```
+  /* Intersect the scaled region to make sure no rounding
   errors made
+   * it to big */
+  surface_rect = (cairo_rectangle_int_t) {
+    .width  = cogl_texture_get_width  (buffer->texture),
+    .height = cogl_texture_get_height (buffer->texture),
+  };
+  cairo_region_intersect_rectangle (scaled_region, &
   surface_rect);
+
+  g_free (rects);
+
+
+  return scaled_region;
+}
+
 static void
 surface_process_damage (MetaWaylandSurface *surface,
                         cairo_region_t     *surface_region,
@@ -279,7 +388,8 @@ surface_process_damage (MetaWaylandSurface
   *surface,

   /* The damage region must be in the same coordinate space as
       the buffer,
    * i.e. scaled with surface buffer scale. */
-  scaled_region = meta_region_scale (surface_region,
   meta_wayland_surface_get_scale (surface));
+  scaled_region =
   meta_wayland_surface_surface_to_buffer_region (surface,
+

   surface_region);

   /* Now add the buffer damage on top of the scaled damage
      region, as buffer
    * damage is already in that scale. */
@@ -696,6 +806,15 @@ meta_wayland_surface_apply_pending_state (
   MetaWaylandSurface      *surface,
       surface->buffer_viewport.buffer.src_rect.height =
          pending->buffer_viewport.buffer.src_rect.height;
       surface->buffer_viewport.surface.width = pending->
          buffer_viewport.surface.width;
       surface->buffer_viewport.surface.height = pending->
          buffer_viewport.surface.height;
+
+      if(meta_wayland_surface_get_actor (surface))
+        {
+          meta_surface_actor_set_viewport (
   meta_wayland_surface_get_actor (surface),
+                                           &surface->
   buffer_viewport.buffer.src_rect,
+                                           surface->
```

```
    buffer_viewport.surface.width,
+                                          surface->
    buffer_viewport.surface.height,
+                                          surface->
    buffer_viewport.buffer.scale);
+        }
      }

    if (meta_wayland_surface_get_actor (surface) &&
@@ -1319,6 +1438,7 @@ meta_wayland_shell_init (
    MetaWaylandCompositor *compositor)
    meta_wayland_legacy_xdg_shell_init (compositor);
    meta_wayland_wl_shell_init (compositor);
    meta_wayland_gtk_shell_init (compositor);
+   meta_wayland_viewporter_init (compositor);
 }

 void
@@ -1785,9 +1905,18 @@ int
 meta_wayland_surface_get_width (MetaWaylandSurface *surface)
 {
    MetaWaylandBuffer *buffer;
+   MetaWaylandBufferViewport *vp = &surface->buffer_viewport;
    CoglTexture *texture;

-   if (surface->buffer_ref.buffer)
+   if(vp->surface.width > 0)
+     {
+       return vp->surface.width;
+     }
+   else if (vp->buffer.src_rect.width > 0)
+     {
+       return vp->buffer.src_rect.width;
+     }
+   else if (surface->buffer_ref.buffer)
      {
        buffer = surface->buffer_ref.buffer;
        texture = meta_wayland_buffer_get_texture (buffer);
@@ -1803,9 +1932,18 @@ int
 meta_wayland_surface_get_height (MetaWaylandSurface *surface)
 {
    MetaWaylandBuffer *buffer;
+   MetaWaylandBufferViewport *vp = &surface->buffer_viewport;
    CoglTexture *texture;

-   if (surface->buffer_ref.buffer)
+   if(vp->surface.height > 0)
+     {
+       return vp->surface.height;
+     }
+   else if (vp->buffer.src_rect.height > 0)
+     {
```

```
+       return vp->buffer.src_rect.height;
+     }
+   else if (surface->buffer_ref.buffer)
      {
        buffer = surface->buffer_ref.buffer;
        texture = meta_wayland_buffer_get_texture (buffer);
diff --git a/src/wayland/meta-wayland-surface.h b/src/wayland/
    meta-wayland-surface.h
index 50e4e0c..2a795f2 100644
--- a/src/wayland/meta-wayland-surface.h
+++ b/src/wayland/meta-wayland-surface.h
@@ -160,6 +160,7 @@ struct _MetaWaylandSurface
    GList *subsurfaces;
    GHashTable *outputs_to_destroy_notify_id;
    MetaWaylandBufferViewport buffer_viewport;
+   struct wl_resource *viewport_resource;

    /* Buffer reference state. */
    struct {
--
2.17.1
```

```
From 23b1a69e95a4e70615f43119c0c2c2865e5ba1e9 Mon Sep 17
    00:00:00 2001
From: Robert Mader <robert.mader@posteo.de>
Date: Thu, 26 Apr 2018 12:15:36 +0200
Subject: [PATCH 4/4] implement wp_viewporter: add add
 META_WP_VIEWPORTER_VERSION to meta-wayland-versions.h

---
 src/wayland/meta-wayland-versions.h | 1 +
 1 file changed, 1 insertion(+)

diff --git a/src/wayland/meta-wayland-versions.h b/src/wayland/
    meta-wayland-versions.h
index 81d37ef..4b20d04 100644
--- a/src/wayland/meta-wayland-versions.h
+++ b/src/wayland/meta-wayland-versions.h
@@ -53,5 +53,6 @@
 #define META_ZXDG_OUTPUT_V1_VERSION         1
 #define META_ZWP_XWAYLAND_KEYBOARD_GRAB_V1_VERSION 1
 #define META_GTK_TEXT_INPUT_VERSION         1
+#define META_WP_VIEWPORTER_VERSION          1

 #endif
--
2.17.1
```

A.2 Xwayland RandR 1.2

```
From 3dc9fa28e743744e1031105b0d68eeda3d3f5fb9 Mon Sep 17
   00:00:00 2001
From: Robert Mader <robert.mader@posteo.de>
Date: Mon, 22 Jan 2018 17:57:38 +0100
Subject: [PATCH] xwayland: use RandR 1.2 interface (rev 2)

This adds the RandR 1.2 interface to xwayland and allows modes
advertised by the compositor to be set in an undistructive
   manner.

With this patch, applications that try to set the resolution
   will usually
succeed and work while other apps using the same xwayland
instance are not affected at all.

The RandR 1.2 interface will be needed to implement fake-mode-
   setting and
already makes applications work much cleaner and predictive
   when a mode
was set.

This depends on https://patchwork.freedesktop.org/series/34628/
---
 hw/xwayland/xwayland-output.c | 74
    ++++++++++++++++++++++++++++++++++
 1 file changed, 74 insertions(+)

diff --git a/hw/xwayland/xwayland-output.c b/hw/xwayland/
   xwayland-output.c
index f754b76..74778be 100644
--- a/hw/xwayland/xwayland-output.c
+++ b/hw/xwayland/xwayland-output.c
@@ -487,12 +487,73 @@ xwl_randr_get_info(ScreenPtr pScreen,
   Rotation * rotations)
     return TRUE;
 }

+#ifdef RANDR_10_INTERFACE
 static Bool
 xwl_randr_set_config(ScreenPtr pScreen,
                      Rotation rotation, int rate,
                         RRScreenSizePtr pSize)
 {
     return FALSE;
 }
+#endif
+
+#if RANDR_12_INTERFACE
+static Bool
+xwl_randr_screen_set_size(ScreenPtr pScreen,
+                          CARD16 width,
+                          CARD16 height,
```

```
+                           CARD32 mmWidth, CARD32 mmHeight)
+{
+    return TRUE;
+}
+
+static Bool
+xwl_randr_crtc_set(ScreenPtr pScreen,
+                   RRCrtcPtr crtc,
+                   RRModePtr mode,
+                   int x,
+                   int y,
+                   Rotation rotation,
+                   int numOutputs, RROutputPtr * outputs)
+{
+    RRCrtcChanged(crtc, TRUE);
+    return TRUE;
+}
+
+static Bool
+xwl_randr_crtc_set_gamma(ScreenPtr pScreen, RRCrtcPtr crtc)
+{
+    return TRUE;
+}
+
+static Bool
+xwl_randr_Crtc_get_gamma(ScreenPtr pScreen, RRCrtcPtr crtc)
+{
+    return TRUE;
+}
+
+static Bool
+xwl_randr_output_set_property(ScreenPtr pScreen,
+                              RROutputPtr output,
+                              Atom property,
+                              RRPropertyValuePtr value)
+{
+    return TRUE;
+}
+
+static Bool
+xwl_output_validate_mode(ScreenPtr pScreen,
+                         RROutputPtr output,
+                         RRModePtr mode)
+{
+    return TRUE;
+}
+
+static void
+xwl_randr_mode_destroy(ScreenPtr pScreen, RRModePtr mode)
+{
+    return;
+}
```

```
+#endif

 Bool
 xwl_screen_init_output(struct xwl_screen *xwl_screen)
@@ -506,7 +567,20 @@ xwl_screen_init_output(struct xwl_screen *
   xwl_screen)

     rp = rrGetScrPriv(xwl_screen->screen);
     rp->rrGetInfo = xwl_randr_get_info;
+
+#if RANDR_10_INTERFACE
     rp->rrSetConfig = xwl_randr_set_config;
+#endif
+
+#if RANDR_12_INTERFACE
+    rp->rrScreenSetSize = xwl_randr_screen_set_size;
+    rp->rrCrtcSet = xwl_randr_crtc_set;
+    rp->rrCrtcSetGamma = xwl_randr_crtc_set_gamma;
+    rp->rrCrtcGetGamma = xwl_randr_Crtc_get_gamma;
+    rp->rrOutputSetProperty = xwl_randr_output_set_property;
+    rp->rrOutputValidateMode = xwl_output_validate_mode;
+    rp->rrModeDestroy = xwl_randr_mode_destroy;
+#endif

     return TRUE;
 }
--
2.17.1
```