

# VR/XR: Big Picture Kickoff

With the groundwork for VR (or rather XR) support done during GSoC, it's time to think about the big picture for XR in Blender. This document is mainly to kick off the discussion, so we can start discussing solutions to some of the more fundamental tasks.

## OpenXR

It is important to mention that all VR support will be based on [OpenXR](#), the brand new specification for VR, AR, MR and all the other *Rs* out there. It is created by the Khronos group and has a huge number of supporters from all over the VR industry. It's likely going to be *the* standard for VR/AR/MR/... (XR).

OpenXR splits the workload between the application (Blender) and the OpenXR runtime. The runtime handles all the device specific stuff, provides many common features (e.g. time wrapping) and can add extensions to the OpenXR API. So the devices Blender will support are defined by the available OpenXR (compatible) runtimes. Currently:

- Windows Mixed Reality
- Oculus (though, not officially released)
- Monado (FOSS Linux OpenXR runtime)

For information on how to test our current OpenXR driven implementation, see [https://wiki.blender.org/wiki/User:Severin/GSoC-2019/How\\_to\\_Test](https://wiki.blender.org/wiki/User:Severin/GSoC-2019/How_to_Test)

## State of the Project

While there are multiple initiatives related to Blender and VR (e.g Marui's [Blender XR](#) and [MPX](#)), the first important milestone for XR support in main Blender is being finished with an ongoing Google Summer of Code project. It aims to bring *stable* and *well performing* OpenXR based VR rendering support into the core of Blender. It's not supposed to enable any interaction, it's solely focused on the foundation of HMD support. More concretely:

- OpenXR loader from the OpenXR SDK to connect to the System's active OpenXR runtime.
- OpenXR extension (and API-layer) management.
- VR session management.
- Well performing VR rendering - more performance improvements are possible, but we have a quite decent baseline.
- Carefully designed error handling strategy, cancelling the VR session with a useful user error message (e.g. "Failed to get device information. Is a device plugged in?") and no side-effects to the rest of Blender.

- Compatibility layer for DirectX-only runtimes.
- --debug-xr command line option enabling our own debug/information prints, OpenXR debug prints and the OpenXR core validation layer.
- --debug-xr-time command line option to print frame render times and FPS information.
- Abstraction (currently a GHOST\_Xr-API) for all OpenXR specific code. Makes higher level usage easier, but most importantly, improves maintenance (esp. when updating OpenXR versions).

There's also work being done to render the VR viewport in a separate thread, but that's not completed yet.

The one important part that's missing is controller support, or more precisely, input and haptic support. Although we can do some experiments, it would be helpful to first get a better idea of how VR interaction should work eventually, before much effort is put into supporting this.

This is the foundation that we can build on. It should allow people with experience in the field to join efforts to bring rich immersive experiences to Blender.

## Design Questions

The following lists a number of questions that have to be figured out. In a way they define some initial project requirements.

## Workflow

- **How will navigation in the virtual world happen?**  
Early on, we should decide how navigation in the VR view will work. It should probably be usable with and without handheld controllers.  
Other VR engines have already implemented their solutions, which may be useful as a reference.
- **How to define workflow specific VR UIs?**  
It would be nice if VR UIs could be integrated well with other workflow mechanics we have in Blender. For example there could be a simplified UI for 101-like templates, or specialized sculpting UIs. A version with focus on accessibility might also be interesting, VR offers new approaches for handicapped access.  
Basically we shouldn't set one VR UI in stone, but allow people to come up with better or alternative solutions for specified use-cases. We'd still deliver a polished default experience obviously.

This has big technical implications, as it influences how tools, operators, gizmos and other graphical elements have to be implemented.

## Operators, Tools & Gizmos

- **How will operators, tools and gizmos work together in VR?**

Will we only provide the active-tool concept, or can we allow non-modal operator execution as well? How will tools and operators be selected/invoked?

- **How can users customize keymap bindings of controllers?**

Note: The OpenXR specification “outsources” this part to the runtime. In Blender we would listen to abstract actions (e.g. “Teleport”) and the OpenXR runtime (Oculus, Windows Mixed Reality, etc.) binds this to a device button/gesture. This makes our job easier, we don’t have to care for device specific keymap bindings.

So assuming we don’t need to listen to device specific button events (which might still be needed), OpenXR already answered this question for us.

## Setup

- **How is the VR reference space defined?**

What’s needed is a point in space that acts as the origin of the VR experience, a *up* direction and an initial direction to look at (always perpendicular to the *up* direction?).

Note that the *up* direction and the origin basically define the floor plane.

The OpenXR runtime calculates the head position/rotation based on that, including eye level above the ground. It can also define the room/movement boundaries.

Other engines probably already figured this out and we may be able to copy them.

- **How can settings be accessed from within the VR session?**

There are different settings to think about: Viewport settings, VR-specific settings (e.g. toggle positional tracking), operator settings, tool settings, etc. We’d probably not expose properties editor or user preference settings for now, although we could make it possible to show any editor in a 3D floating popup.

- **How can users change settings for the VR session, but outside of it? Do we allow this at all?**

For example, users may want to change viewport settings from the regular 2D UI, because the VR session renders too slow. This also matters for users that don’t have controllers.

- **How are session defaults and startup settings defined?**

This is mainly about viewport settings (display mode, lighting, shading, etc.) and VR specific settings. You may want to set these up even before the session starts. And you may want to set your own default settings, too.

# Implementation

- **How to implement support for operators, tools and gizmos for VR interaction?**

Currently, operators, tools and gizmos assume a 2D based interaction. Adding a third dimension is not trivial and we need to think about how to do this best.

Possible solutions:

- Add an `invoke_3d()` and/or `modal_3d()` callback to operators. Downside is that C-defined operators need this callback to be implemented in C too, it can't just be implemented in Python.
- Add wrapper operators (possibly in Python) which only call the wrapped operators to apply the operation. May not work well with modal operators and Python wrappers are limited by the Python API capabilities.

- **How will picking with controllers work?**

When users point into the VR-space with a controller, we need to figure out what it is pointed at.

Note: We can probably fake some picking feedback while no operations are invoked, i.e. we can draw a little dot on surfaces the controller points at with help of the depth buffer of the current frame.

- **How should the abstract VR actions (like "Teleport") be implemented in our event system?**

There's a good chance we'll offload VR rendering onto a separate thread with its own main-loop. Should we query and handle actions there, separate from the usual event system?

Also, some information we'd want to update for every frame, e.g. the controller position and rotation. So this is not event driven like the rest of Blender. How can this information be dispatched to modal operators (e.g. so that the transform tool keeps updating the object position as the controller moves).

- **How to implement drawing of 2D elements for 3D?**

For VR GUIs, we'd want to display quite a few 2D elements in the VR space.

For example pie menus, popups and all kinds of regular widgets (checkboxes, number buttons, etc.).

It may also be useful to allow displaying any editor of Blender as floating plane in the 3D environment and allowing interaction with controllers. Similar to how Windows Mixed Reality allows opening the regular desktop in a floating plane:



Note that this should not be the prominent way to interact with the Blender UI in VR. It's just something that can be implemented without too much trouble while being useful to workaround limitations of the VR UI, without having to take off the HMD. The exception, not the rule.

## Besides VR

Thanks to the OpenXR backend, it should be relatively easy to also support augmented and mixed reality experiences in Blender. So when we talk about VR, AR and MR are usually affected too.

What we'd like to get before too long is some basic MR support (e.g. modeling on a real world desk) and support for using frontal cameras of HMDs to display real objects (like keyboard and mouse) in the Blender VR session.

It seems that the OpenXR runtime again does all the difficult work for us. What we'd need is a way to enable these modes, and a few changes and extensions to our VR drawing to support mixed reality experiences. Some experiments should be done soon to check this.

## Goals

XR enables an entirely new way to work with computer generated 3D worlds. Instead of trying to take the UIs we've already implemented for traditional monitor use and make them work in XR, we should establish an experience on entirely new grounds. Otherwise, what we'll end up with is just a different, more difficult way to use Blender.

So to enter the new immersive world, we should think about which workflows could benefit the most from XR, and carefully craft a new, improved experience for them. Don't try to enable huge amounts of features for XR, risking bad outcome - or even total failure - but find out which features matter the most and focus on them first.

Long term, the experience can then be enriched further, with more features added as smaller, more concise projects. Eventually, XR would allow creating 3D content with unseen interactivity. The entirety of Blender's content creation capabilities available in an immersive 3D experience. However, this better be the long term vision for XR, not the goal for this initial project.

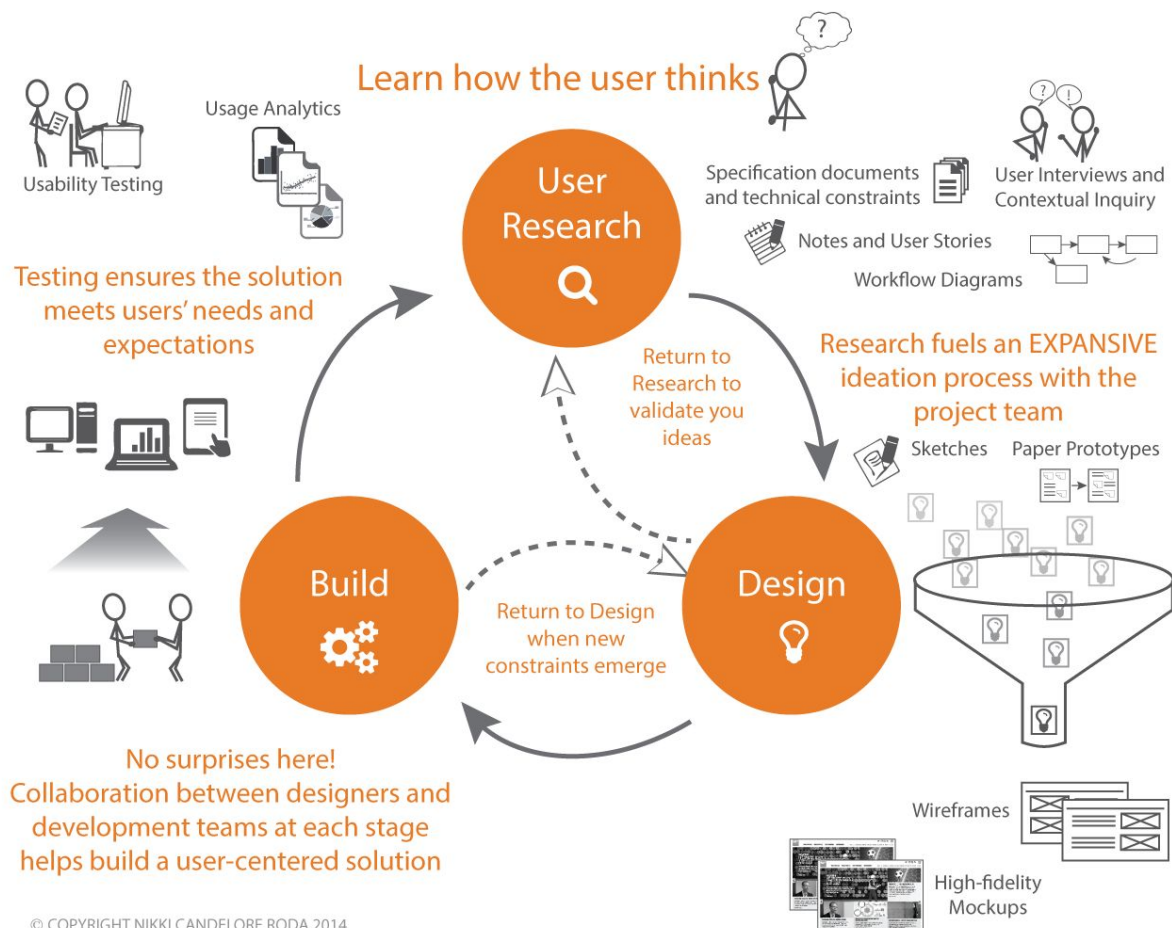
## Development Process

I would propose that we do a use-case driven development process. That means, we define a number (say 10-20) of specific use cases. These should be chosen wisely, based on what we think has most potential with XR.

First, just vaguely describe what requirements the system has for a use-case, a bit later on, define what tools and options are needed to get the use case done (possibly by a specific [persona](#)).

On top of that, the process should be done iteratively. That means, we define short term goals for an iteration, work on them and evaluate afterwards before we get into the next iteration. The iterations may include implementation work. So we don't do the waterfall-like "design first", but there should be enough design work done before first implementations are made.

Any implementation should be driven by a use-case (or multiple) that the team agrees is drafted out well enough. This is a vague interpretation of user-centered design processes, as commonly suggested in the field of human-computer interaction:



Source: <http://nikkiroda.com/user-centered-design-process/>

## The Team

Dalai Felinto is available for the role as coordination lead, Julian Eisel can lead development on the Blender source code side. The [MPX](#) team cares a lot about usability for drawing related workflows, they are interested in joining as well. Then there are people from [MARUI](#) who have shown much interest in helping tools and general UI development.

We further have quite a few candidates to be artist stakeholders. It's important that we don't only have artists that care for the usability, but also for the production environment aspects of the project outcome.

This seems to be a good mix of people with complementing roles. We should soon get together to define the overall project goals, the general vision for XR in Blender and to get work started.