Agenda

- About Me
- Handle and atom types
- Modeling interaction: Actions, Action Sets, and Interaction Profiles
- Dive into OpenXR app structure/API usage
- Time permitting: Question and Answer

Slides, with speaker notes and links, will be available at https://www.khronos.org/developers/library/2020-chinavr
About Me: Ryan Pavlik

- Open-source VR software developer since 2009
- OpenXR working group
  - participant since the first official meeting in January 2017
  - elected specification editor in April 2019
- Principal Software Engineer at Collabora
  - Focusing on XR client project
  - Leading our OpenXR contributions
  - Developer on Monado
OpenXR Handle Types

- Instance
  - ActionSet
    - Action
  - Session
    - Space
    - Swapchain
Atoms

- Instance
  - ActionSet
    - Action
  - Session
    - Space
    - Swapchain
  - Path
  - SystemId
Modeling Interaction with Actions

- Focus first on what users do, not the hardware they do it with
- Important for hardware-independence.
- **Action**: A semantic (meaningful) bit of interaction
  - Types: Boolean (button), Float (analog), Vec2, Pose (tracked object), Haptic
  - e.g. “grab_object”, “teleport”, “hand_pose”
- **ActionSet**: a group of related actions for a context, environment, etc.
  - e.g. “menu”, “gameplay”, “driving”
  - One or more active at any time
Suggested Bindings and Interaction Profiles

• How you customize for hardware you’ve tested, without excluding the rest
• For each controller type you’ve tested (“interaction profile”), suggest bindings for actions
  - With as many or few action-binding pairs as you like - OK if not all actions have a suggested binding
  - Can suggest multiple bindings per action in a call: e.g. both left and right hands can “grab_object”
  - Binding is an XrPath atom representing a path string like /user/hand/right/input/select/click
• If your application is used on different hardware, the runtime may re-map your actions to the available hardware
• Set up actions, action sets, and suggested bindings once, at startup

interaction profiles added by vendor extensions XR_MSFT_hand_interaction, XR_HUAWEI_controller_interaction, and multi-vendor extensions XR_EXT_eye_gaze_interaction, XR_EXT_hp_mixed_reality_controller, XR_EXT_samsung_odyssey_controller
One last Action setup step

- Set up Actions, Action Sets, and provide suggested bindings at application start.
- Before you can use them, one more call is required later:
  - \texttt{xrAttachSessionActionSets}
  - Associates them with the session
  - Makes them immutable
  - Editor authors: tear down session, actions, action sets and re-create to modify them
- Why is action setup done all up front and immutable?
  - Good rebinding experience needs maximum information on interaction early in execution
Sample of Actions

- These are the actions from “hello_xr” - see `OpenXrProgram::InitializeActions`
- All in one action set, “gameplay”, due to simplicity of the app
- All are specified for both left and right hand as “subaction paths” because we might want to know which hand did an action - which hand grabbed object, etc.

<table>
<thead>
<tr>
<th>actionName</th>
<th>localizedActionName</th>
<th>actionType</th>
<th>subaction path</th>
</tr>
</thead>
<tbody>
<tr>
<td>grab_object</td>
<td>Grab Object</td>
<td>Float Input</td>
<td>/user/hand/left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/user/hand/right</td>
</tr>
<tr>
<td>hand_pose</td>
<td>Hand Pose</td>
<td>Pose Input</td>
<td>/user/hand/left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/user/hand/right</td>
</tr>
<tr>
<td>quit_session</td>
<td>Quit Session</td>
<td>Boolean Input</td>
<td>/user/hand/left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/user/hand/right</td>
</tr>
<tr>
<td>vibrate_hand</td>
<td>Vibrate Hand</td>
<td>Vibration Output</td>
<td>/user/hand/left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>/user/hand/right</td>
</tr>
</tbody>
</table>
xrSuggestInteractionProfileBindings 1

- Standard defines “khr/simple_controller” as a minimal subset profile
- Note here that `grab_object` is float, but suggested to bind to “select/click” (boolean)
  - Runtime will automatically convert boolean to a 1 or 0.

<table>
<thead>
<tr>
<th>actionName</th>
<th>actionType</th>
<th>subaction path</th>
<th>/interaction_profiles/khr/simple_controller</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>grab_object</code></td>
<td>Float Input</td>
<td>/user/hand/left</td>
<td>/user/hand/left/input/select/click</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/input/select/click</td>
</tr>
<tr>
<td><code>hand_pose</code></td>
<td>Pose Input</td>
<td>/user/hand/left</td>
<td>/user/hand/left/input/grip/pose</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/input/grip/pose</td>
</tr>
<tr>
<td><code>quit_session</code></td>
<td>Boolean Input</td>
<td>/user/hand/left</td>
<td>/user/hand/left/input/menu/click</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/input/menu/click</td>
</tr>
<tr>
<td><code>vibrate_hand</code></td>
<td>Vibration Output</td>
<td>/user/hand/left</td>
<td>/user/hand/left/output/haptic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/output/haptic</td>
</tr>
</tbody>
</table>
xrSuggestInteractionProfileBindings 2

- HTC Vive controller
- The `grab_object` action is here suggested for the “trigger/value” input
  - trigger/value instead of select/click
  - float instead of boolean: no conversion required

<table>
<thead>
<tr>
<th>actionName</th>
<th>actionType</th>
<th>subaction path</th>
<th>/interaction_profiles/htc/vive_controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>grab_object</td>
<td>Float Input</td>
<td>/user/hand/left</td>
<td>/user/hand/left/input/trigger/value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/input/trigger/value</td>
</tr>
<tr>
<td>hand_pose</td>
<td>Pose Input</td>
<td>/user/hand/left</td>
<td>/user/hand/left/input/grip/pose</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/input/grip/pose</td>
</tr>
<tr>
<td>quit_session</td>
<td>Boolean Input</td>
<td>/user/hand/left</td>
<td>/user/hand/left/input/menu/click</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/input/menu/click</td>
</tr>
<tr>
<td>vibrate_hand</td>
<td>Vibration Output</td>
<td>/user/hand/left</td>
<td>/user/hand/left/output/haptic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/output/haptic</td>
</tr>
</tbody>
</table>
• Oculus Touch controller  
• Has a float input suitable for `grab_object` action - called “squeeze/value”  
• Only left controller has a menu button, so not suggesting a binding for `quit_session` on the right hand.

<table>
<thead>
<tr>
<th>actionName</th>
<th>actionType</th>
<th>subaction path</th>
<th>/interaction_profiles/oculus/touch_controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>grab_object</td>
<td>Float Input</td>
<td>/user/hand/left</td>
<td>/user/hand/left/input/squeeze/value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/input/squeeze/value</td>
</tr>
<tr>
<td>hand_pose</td>
<td>Pose Input</td>
<td>/user/hand/left</td>
<td>/user/hand/left/input/grip/pose</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/input/grip/pose</td>
</tr>
<tr>
<td>quit_session</td>
<td>Boolean Input</td>
<td>/user/hand/left</td>
<td>/user/hand/left/input/menu/click</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td></td>
</tr>
<tr>
<td>vibrate_hand</td>
<td>Vibration Output</td>
<td>/user/hand/left</td>
<td>/user/hand/left/output/haptic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/user/hand/right</td>
<td>/user/hand/right/output/haptic</td>
</tr>
</tbody>
</table>
Structure of an OpenXR App

- Get started
  - Instance
- Find out where/how to run
  - SystemId atom
  - ViewConfigurationType enum
- Set up your interaction/input handles
  - Create Action Sets, Actions
  - Suggest bindings
- Prepare your immersive experience
  - Create Session
  - Attach action sets
  - Create Reference and Action Spaces
  - Create Swapchain
- Participate in the frame loop and handle input
  - Poll for events too
Creating an Instance

- Choose which extensions you want
  - Need at least one graphics binding extension
  - Can identify available extensions: \texttt{xrEnumerateInstanceExtensionProperties}
- Choose which API layers you want, if any
  - Optional
  - \texttt{xrEnumerateApiLayerProperties}
- Set up application info
  - So runtimes can identify your app
- \texttt{xrCreateInstance}
System and Views

- **xrGetSystem**
  - with your desired form factor: HMD or handheld
  - may be temporarily unavailable

- **View configuration**
  - Mono, Stereo, ...
  - `xrEnumerateViewConfigurations` if you support more than one
  - `xrGetViewConfigurationProperties`
  - `xrEnumerateViewConfigurationViews`
    - mono has one view
    - stereo has two views

extended in vendor extension XR_VARJO_quad_views, multi-vendor extension XR_EXT_view_configuration_depth_range
Action Sets, Actions, and Suggested Bindings

- **ActionSet**: a group of related actions for a context, environment, etc.
  - `xrCreateActionSet`
- **Action**: A semantic (meaningful) bit of interaction
  - `xrCreateAction`
- For each controller type you’ve tested (“interaction profile”) call `xrSuggestInteractionProfileProfileBindings` once
Creating your **Session**

- **Graphics binding**
  - Do your graphics binding’s “GetGraphicsRequirements” call
  - Create your graphics binding struct
  - Chain it via next on `XrSessionCreateInfo`

- **xrCreateSession**
  - Requires a `SystemId`

- **Attach your action sets to the session**
  - `xrAttachSessionActionSets`
  - Makes them immutable
Create **Spaces**

- Multiple ways to get XrSpace handles
  - Reference space: from Session and enum
    - local space
    - view space
    - stage space
    - `xrCreateReferenceSpace`
  - Action space: from Session and pose Action
    - `xrCreateActionSpace`
  - For both reference and action spaces
    - Session is the parent handle
    - Can specify an additional, fixed transform at handle creation time
    - `xrLocateSpace`

extended in vendor extensions XR_MSFT.spatial_anchor, XR_MSFT.unbounded_reference_space
Create your Swapchain

- Get graphics API-specific formats via `xrEnumerateSwapchainFormats`
- `xrCreateSwapchain`
- Get access to graphics API-specific handles/references to the swapchain images
  - `xrEnumerateSwapchainImages`
  - Pass array of *extension-defined* structures
  - Save this information to use every frame
Frame loop

- Frame functions: called on the Session
  - `xrWaitFrame` to block until head-pose-dependent sim and rendering
  - `xrBeginFrame` to mark start of render
  - `xrEndFrame` to submit the image
  - Populate XrFrameEndInfo::displayTime using output of `xrWaitFrame`
Pipelined rendering

- Frame function synchronization
  - `xrBeginFrame/xrEndFrame` calls must be ordered “as if” single-threaded
  - At most one simultaneous `xrWaitFrame` call at a time
  - Each `xrWaitFrame` must eventually be matched with a unique `xrBeginFrame`
  - Any `xrWaitFrame` call must block until the previous frame’s `xrBeginFrame`
Swapchain and view management

- **Swapchain management**
  - [xrAcquireSwapchainImage](#) to get index
  - To look up/create your command buffers
  - [xrWaitSwapchainImage](#) before writing
    - Typically immediately after acquire
    - Do not submit command buffers until this returns
  - [xrReleaseSwapchainImage](#) before [xrEndFrame](#): implicitly uses most recently released image

- **xrLocateViews**
Getting input

- **xrSyncActions**
  - Specify which ActionSets should be active at this time
  - This is the only time non-pose input data updates

- **Get the data**
  - All ActionSets attached but not specified in xrSyncActions will have their actions return “not active”
  - Actions might not get data if your session is not focused, for privacy/security
  - [xrGetActionState*](#) calls

- **Poses: use** [xrLocateSpace](#)
Events

- **xrPollEvents**
  - Requires an Instance
  - Many events only happen during a Session

- **Describes changes to**
  - Active interaction profile
  - Continuity of reference spaces/tracking
  - Session state

- **Provide an XrEventDataBuffer for the runtime to populate with an event of some other type**
Wrap-up

• Outline
  - About Me
  - Introduction to OpenXR
  - OpenXR in context
  - OpenXR app structure/API usage
  - Time permitting: Question/Answer

• Resources
  - Landing page with news: khrionos.org/openxr
  - API registry (links to the spec, ref pages, all the repos, etc) khrionos.org/registry/openxr

• Community
  - Source, issue trackers, etc github.com/KhronosGroup?q=openxr
  - Chat khr.io/slack
  - Forum community.khronos.org/c/openxr

• Open-Source Runtime for Linux: Monado
  - Community project founded by Collabora, not a Khronos/OpenXR WG project
  - Repos, including additional (cross-platform) OpenXR-related projects gitlab.freedesktop.org/monado

Thank you!
OpenXR Master Class

Ryan A. Pavlik, Ph.D.
Principal Software Engineer, Collabora, Ltd.
OpenXR Working Group Spec. Editor