Gazebo

Some images removed :(

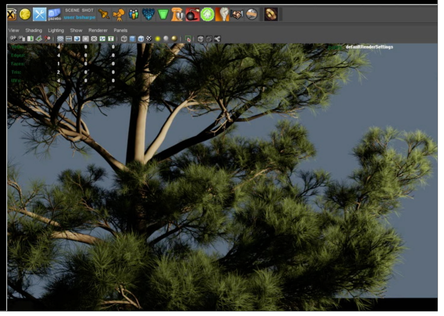
- Hello, my name is Robert Cannell and I'm the Lead Software Engineer on Gazebo.

Today I'll be talking about Gazebo, and how it fits in to the internal pipeline at Weta along with some use cases, design decisions and reasons for making some of the choices that we do when developing the software

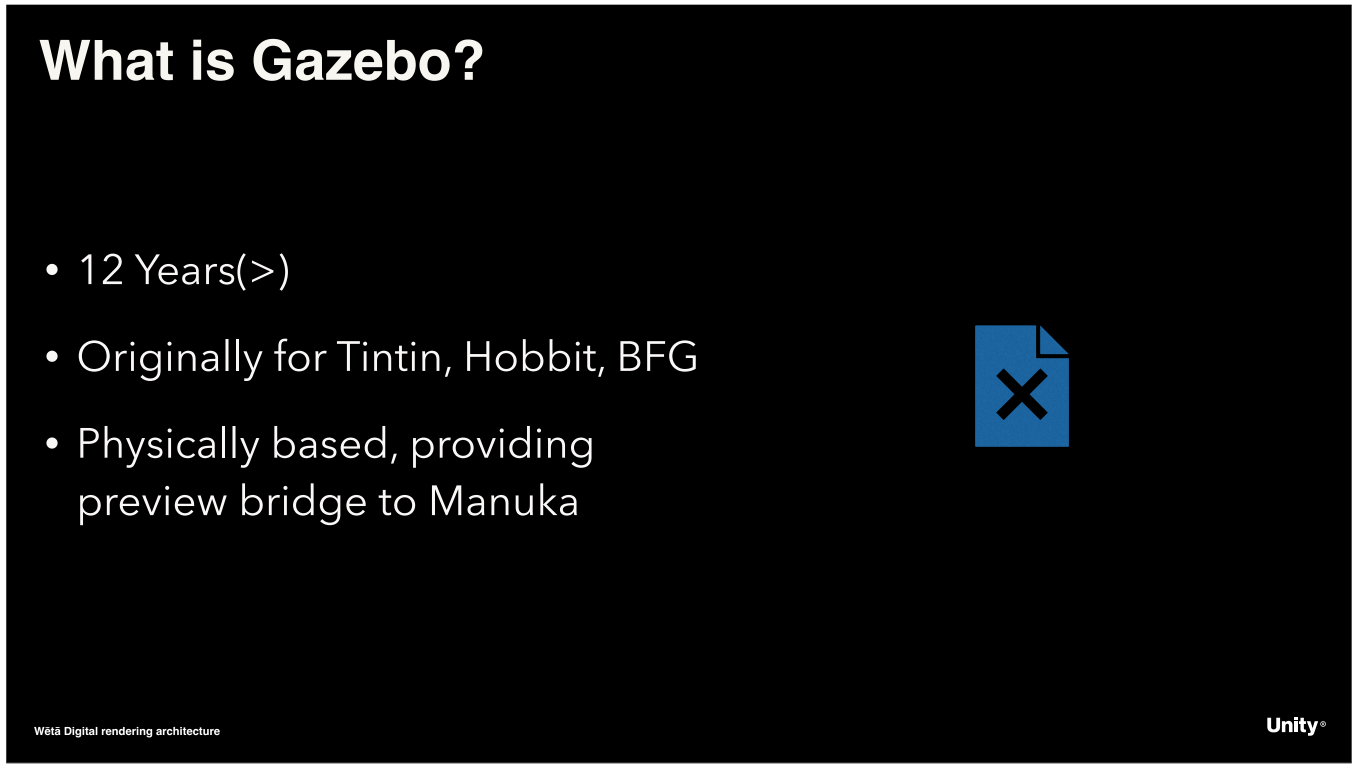
For this talk I'll discuss a bit about the history of Gazebo and some of the challenges that we face when developing for realtime VFX rendering.

What is Gazebo?

- Gazebo is Weta Digital's general-purpose real-time renderer
- Primary viewport renderer in multiple DCCs, across many departments
- Serves as a common way to view and edit the movie data
- Built as a solution from stage / virtual production to lighting / representation of final frame



Gazebo is a real-time renderer built to handle the complexity of visual effects needed for modern motion pictures and it services the breadth of the pipeline at Wētā. This starts with use on stage in the demanding virtual production environments Wētā creates, but also serves most of the artists with day-to-day usage, where Gazebo has been integrated as a viewport into the relevant Digital Content Creation (DCC) tools, and even the odd final render. This provides a common viewing framework that gives artists consistent, interactive views of their scenes across the pipeline in the viewport of the tool they are using. Gazebo is designed to interactively render scenes many of those original viewports could not display.

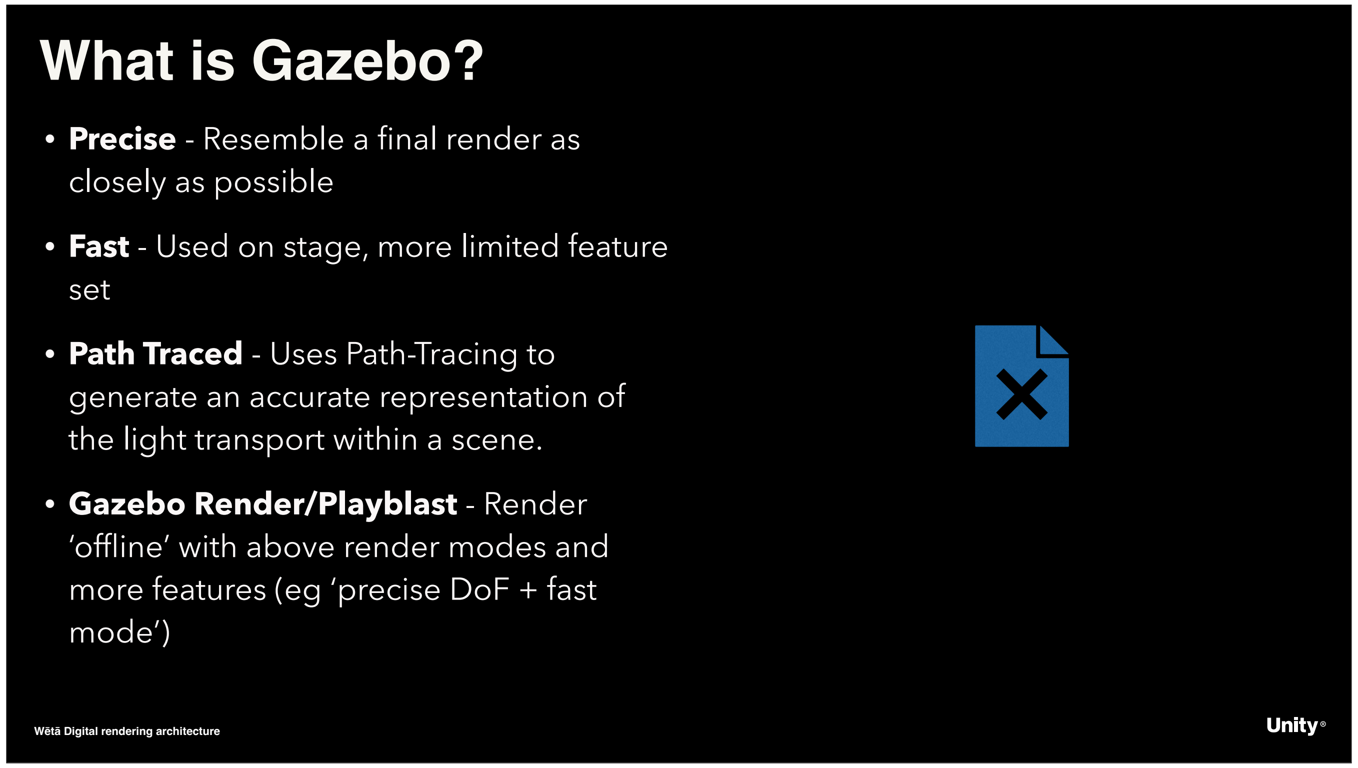


What is Gazebo?

- 12 Years(>)
- Originally for Tintin, Hobbit, BFG
- Physically based, providing preview bridge to Manuka

To provide the consistent view of data, Gazebo is physically based, handles physical lighting through our PhysLight integration, and has support for spectral rendering. The purpose is to provide an accurate real-time preview of the final frame which would be rendered in our off-line renderer, Manuka.

This ability to provide images and reference renders for virtual production and performance motion capture, but then be able to view data consistently all the way through subsequent look development, animation / motion edit, creature simulation, layout, set dressing, and all the way into lighting makes Gazebo a unique real-time renderer.

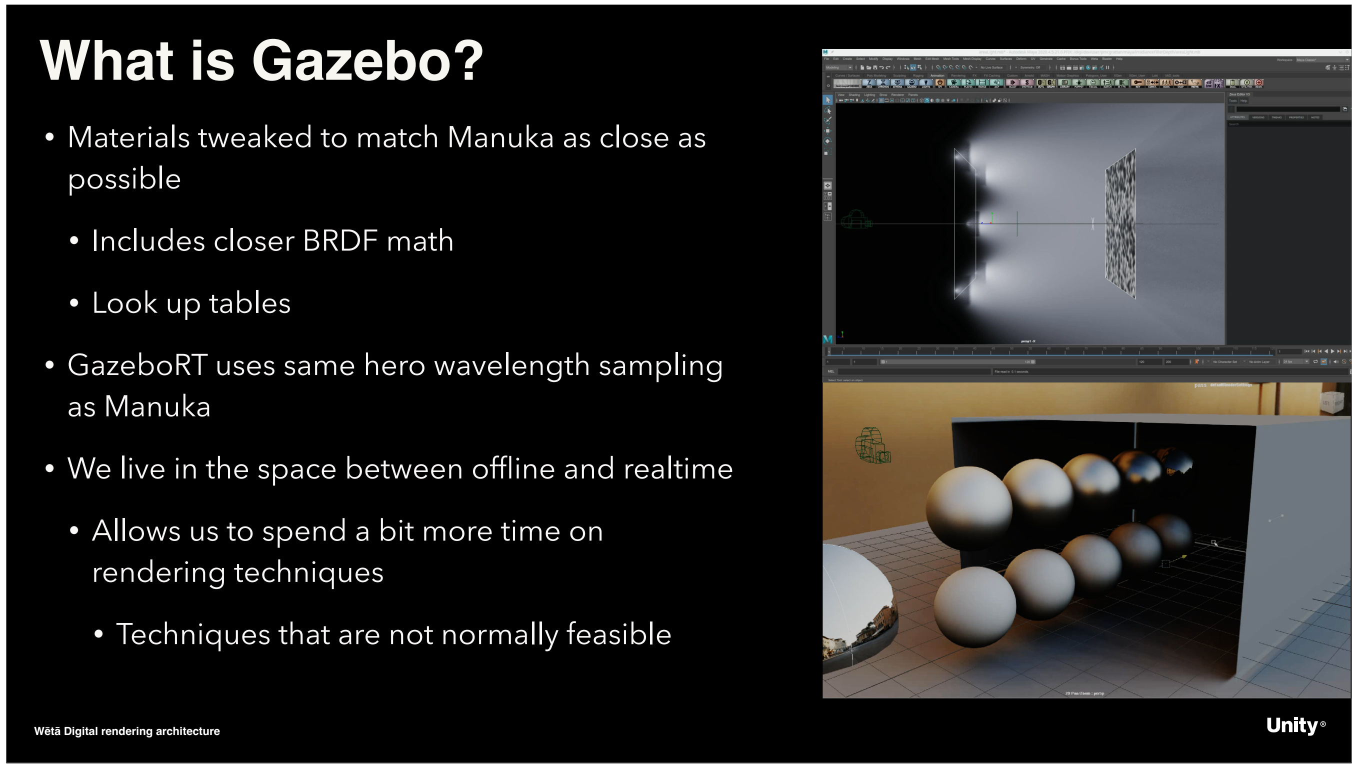


What is Gazebo?

- **Precise** - Resemble a final render as closely as possible
- **Fast** - Used on stage, more limited feature set
- **Path Traced** - Uses Path-Tracing to generate an accurate representation of the light transport within a scene.
- **Gazebo Render/Playblast** - Render ‘offline’ with above render modes and more features (eg ‘precise DoF + fast mode’)

Gazebo is a multi-use tool and for that reason it can be customised to the task at hand that the user is trying to achieve. There are many rendering options to change the performance vs fidelity aspects of Gazebo, and one of these settings is the render mode. This consists of Fast Mode, which is a pure “one frame” rasterizer which attempts to render the final output as fast as possible, given the current quality settings. For a more detailed render, a user may select Precise Mode which stochastically re-renders the frame and re-samples all lighting information (IBLs, Area Lights, Directional etc) for every sample and averages the samples together to produce a high-fidelity render. For even more realistic rendering (including better Global Illumination and Shadowing) the user can select our Path Traced mode which allows for a more truthful rendering whilst also having the benefits of being rendered in a DCC and all the other features that come with Gazebo integration such as Camera Plates, Compositing and Wireframes.

Renders can also be done via a Gazebo Render which allows users to add more features such as better Depth of Field, Motion Blur and Upscaling of the final image and these can be run on our internal render wall. This is at the cost of taking more time to render but still at a fraction of the time it would take to do a full offline render with our internal offline Path Tracer. The ability to render out Fisheye and LatLong for ingestion into other tools is possible also



What is Gazebo?

- Materials tweaked to match Manuka as close as possible
  - Includes closer BRDF math
  - Look up tables
- GazeboRT uses same hero wavelength sampling as Manuka
- We live in the space between offline and realtime
  - Allows us to spend a bit more time on rendering techniques
    - Techniques that are not normally feasible

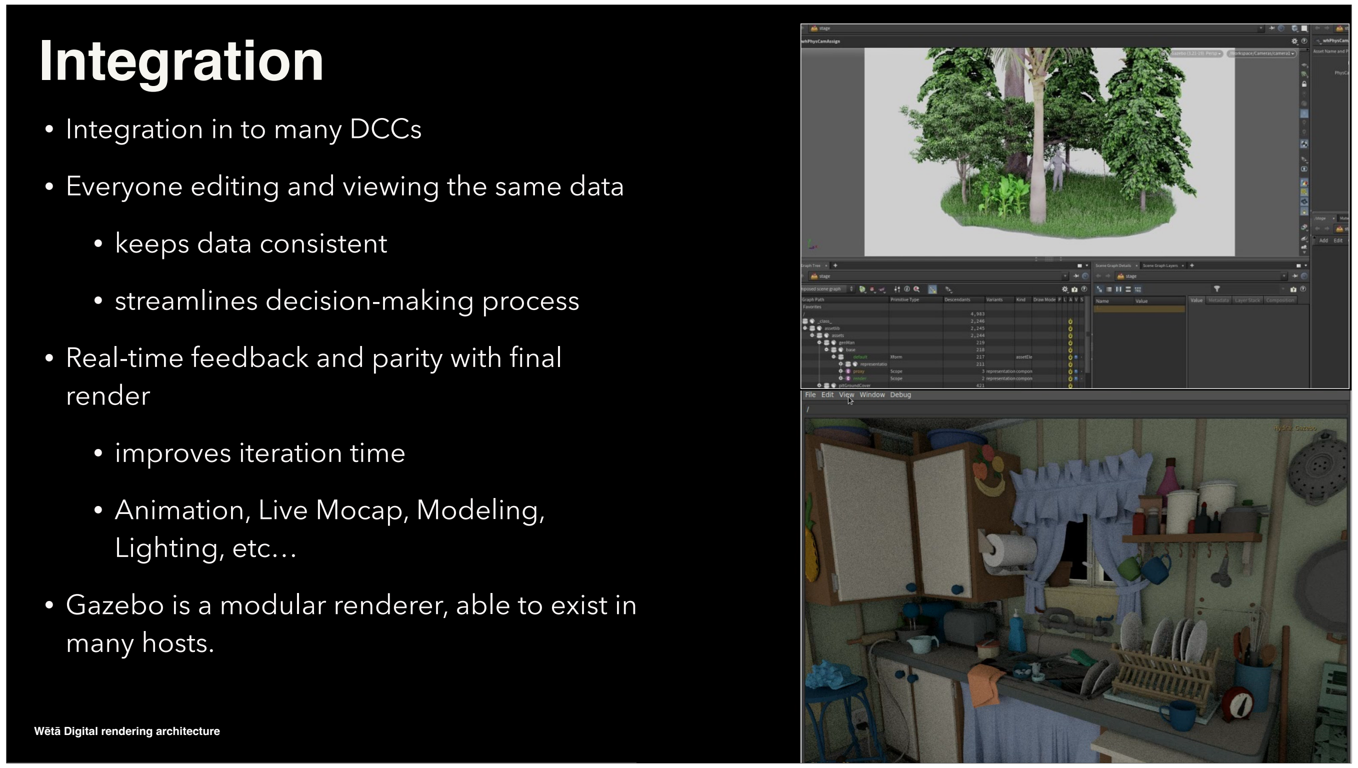
Wētā Digital rendering architecture

Unity®

I like to say that we live in the space ‘between’ offline and realtime and that this affords us some extra luxuries -
We are not constrained by constant 60fps or 30fs targets and this allows us to research algorithms or approaches that are not always possible in a fully realtime environment. Frame rate targets are variable and usually differ per department and use case - some departments may need 60fps where others might be fine with sub-30 as long as there is interactivity. Of course, dropping below this is not great and we aim to try to avoid this where we can.

For further frame rate savings, render settings are able to be tweaked in order to improve performance if required, at the cost of rendering fidelity, and this in itself can bring challenges when dealing with show environments that need a consistent look over the entire show - if there is a performance improvement that is needed sometimes it has to not involve a change to the existing look of the shot as for example in Avatar the look of the shot needs to match a reference template throughout the entire pipeline and If this reference template changes in look it can introduce issues and/or extra work later in the pipeline when decisions are made on this template.

We tweak our materials to match Manuka in various ways including adopting look up tables or math changes to our BRDFs. This does come at a small cost, but the benefits are worth it.



Integration

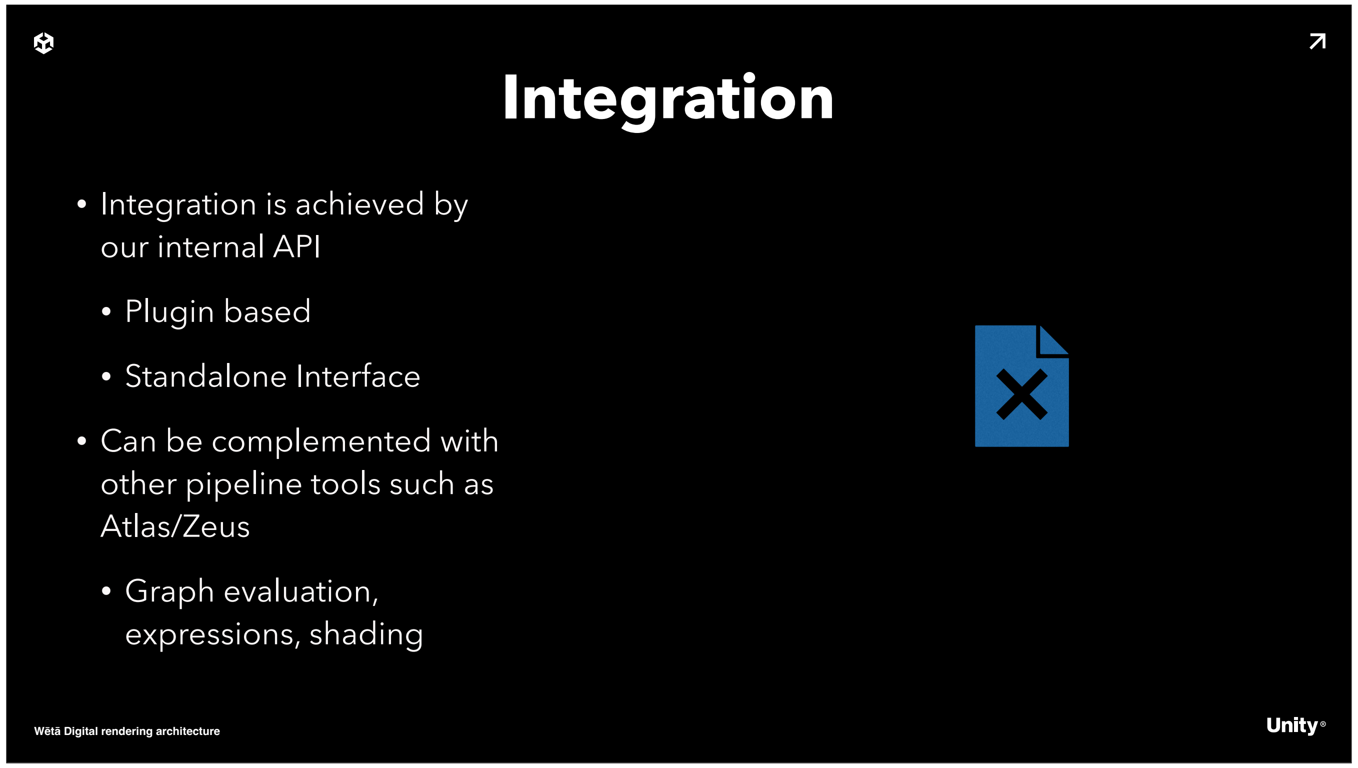
- Integration in to many DCCs
- Everyone editing and viewing the same data
    - keeps data consistent
    - streamlines decision-making process
- Real-time feedback and parity with final render
    - improves iteration time
    - Animation, Live Mocap, Modeling, Lighting, etc…
- Gazebo is a modular renderer, able to exist in many hosts.

Wētā Digital rendering architecture

The ability to view the movie data directly, without change, is a key feature of Gazebo. It helps keep data consistent, allowing for our many different departments to work closely together in a streamlined fashion.

Gazebo's realtime feedback keeps our artists efficient; and parity with the final render keeps turnaround times to a minimum.

Gazebo itself is a modular renderer, which means it can be plugged into different DCC host applications such as: Maya, MotionBuilder, Katana, Houdini (native viewport), and more recently as a USD hydra delegate.

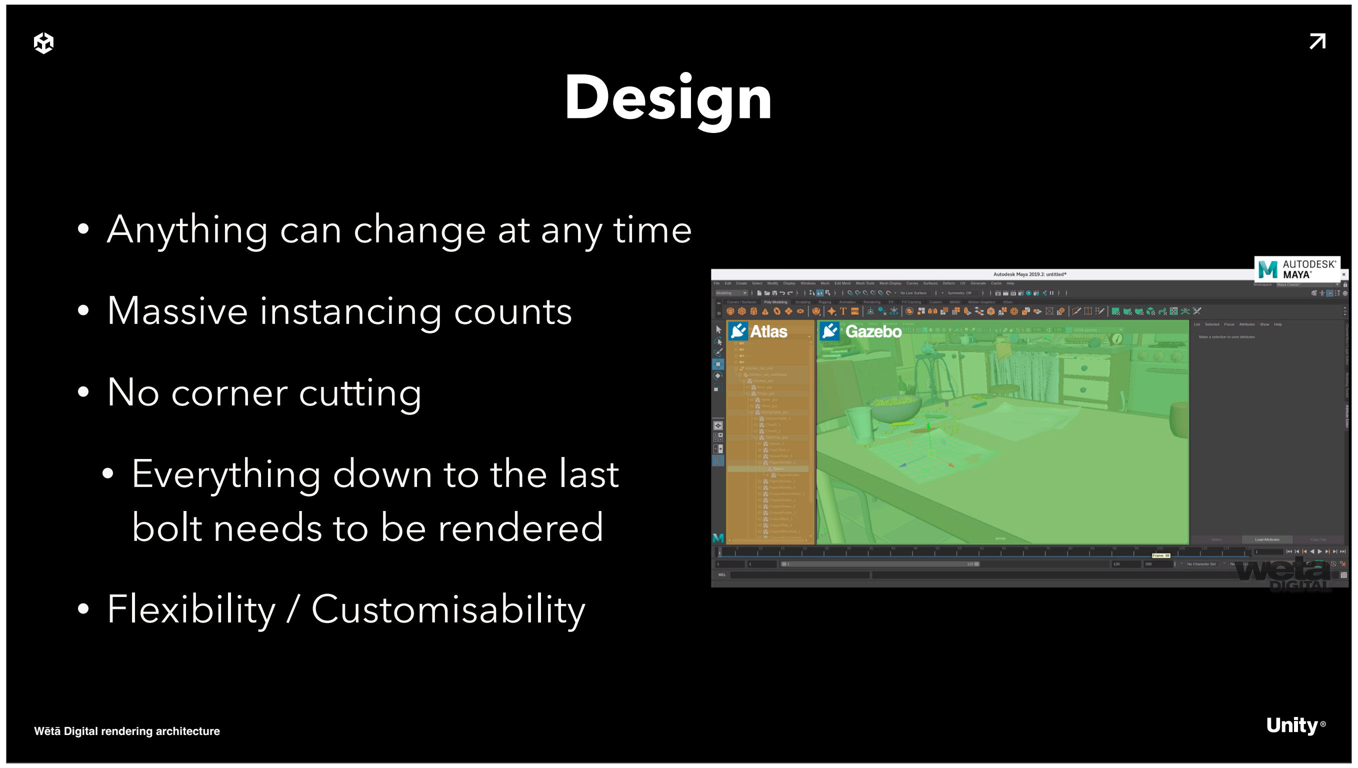


Integration

- Integration is achieved by our internal API
  - Plugin based
  - Standalone Interface
- Can be complemented with other pipeline tools such as Atlas/Zeus
  - Graph evaluation, expressions, shading

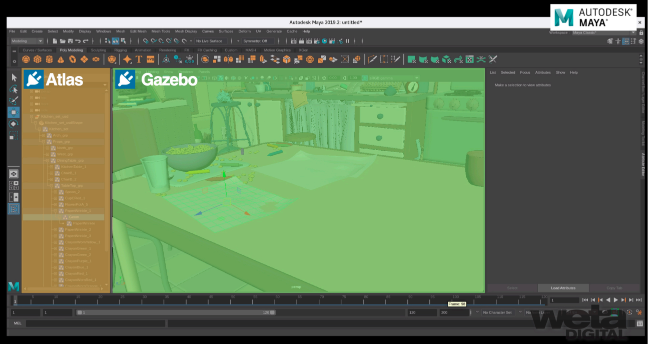
Gazebo provides an interface for interacting with the renderer via an in-house API. This API allows for the implementation of provider plugins that can then modify the internal representation of a scene by adding and removing objects and shading information and other data that is associated with the rendering of these objects. These plugin providers can be dynamically loaded, and Gazebo will render the scene without any knowledge of the providers themselves - if the provider does not exist or cannot be loaded, then simply any items that it provides are not rendered. This API is backwards compatible to allow for the many unique combinations of providers and versions of these providers that can potentially be in use at any one time.

This provider system uses a common rendering API and is tightly integrated with our in-house scene delegate system Atlas to provide the ability to render a level of instancing and complexity not possible with built-in viewports. Combined with our in-house UI system, Zeus, we now have a full solution to browsing and editing data in realtime, within many DCC viewports



Design

- Anything can change at any time
- Massive instancing counts
- No corner cutting
  - Everything down to the last bolt needs to be rendered
- Flexibility / Customisability



Before I talk a bit about some of the systems within Gazebo I wanted to outline some of the design principles that drive our day to day decisions in development of the software.

Anything can be changed; Directors might want a wall removed, a box moved or the camera location changed which requires responsiveness along with the ability to handle massive amounts of geometry and lighting due to the fact that anything could potentially be visible at any time and therefore everything that can potentially be seen at any point of time needs to be available to be rendered.

During dressing use, set dressings can be done that require millions of instances to be placed that require responsive feedback and contact data in order to make good decisions on where the items are to be placed.

Models and Lighting have much smaller data sets, but require accurate lighting to make accurate decisions.

Flexibility is key to this also so that our users have options to dynamically tailor their setup to the shot at hand, whether this been tweaking lighting features, compositing image plates or changing rendering modes, the goal is to support the varying workflows that exist here.

Wētā Digital rendering architecture

Unity

This is an example from <> rendered in GazeboRT that shows some of the complexity required in a single shot, at full hero resolution - everything here needs to be available to be rendered because it could be in the shot or the camera could be moved to any place at any time.

Every nut and bolt has been modelled and the interior of all these models are not empty inside

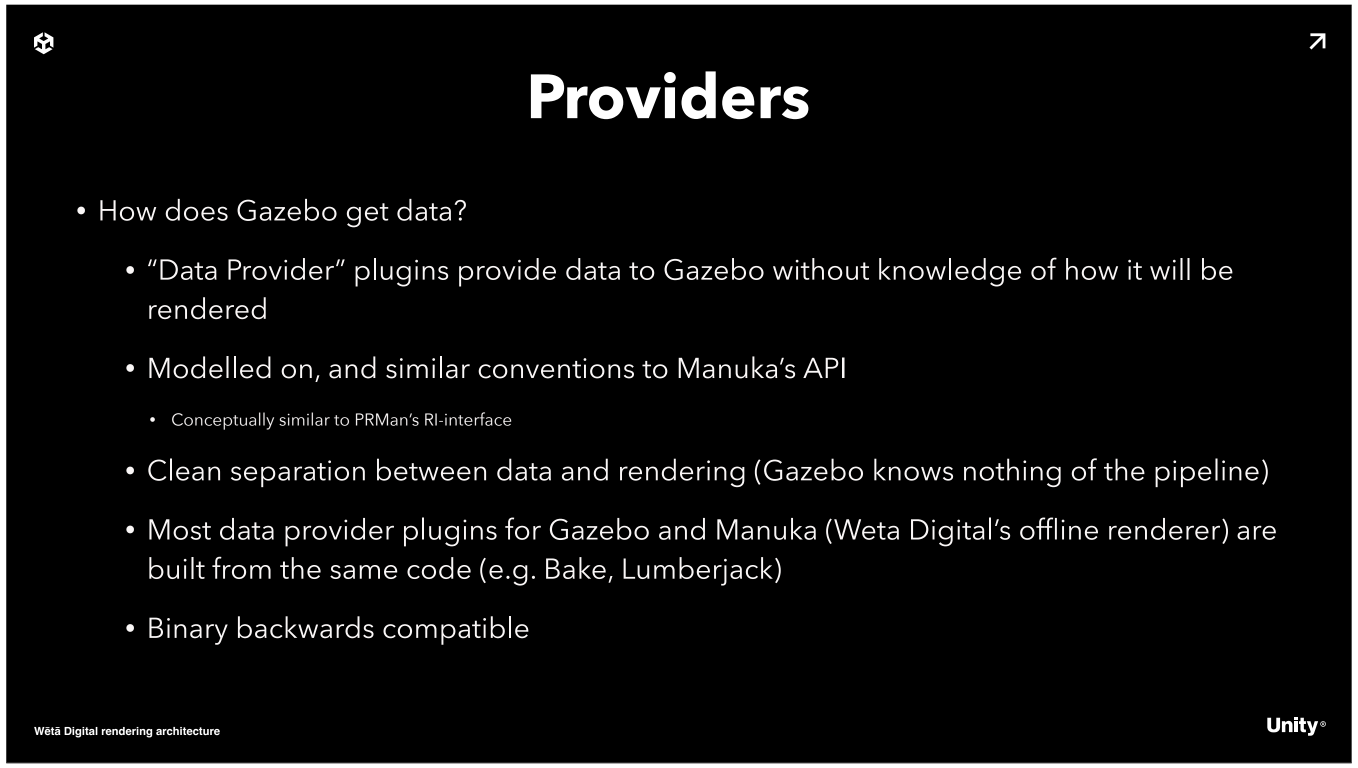
Rocks and pebbles are their own individual instances, each of these are meticulously placed by hand. This scene includes Thanos and for those of you who can't see him, <c>he's here

Wētā Digital rendering architecture

Unity

And here

This is just an example of some of the data that we are dealing with which comes with with its own set of issues ranging from memory usage, to streaming of data. These are all challenging issues that we continue to work on and refine all the time and are already looking at ways to improve our current technology



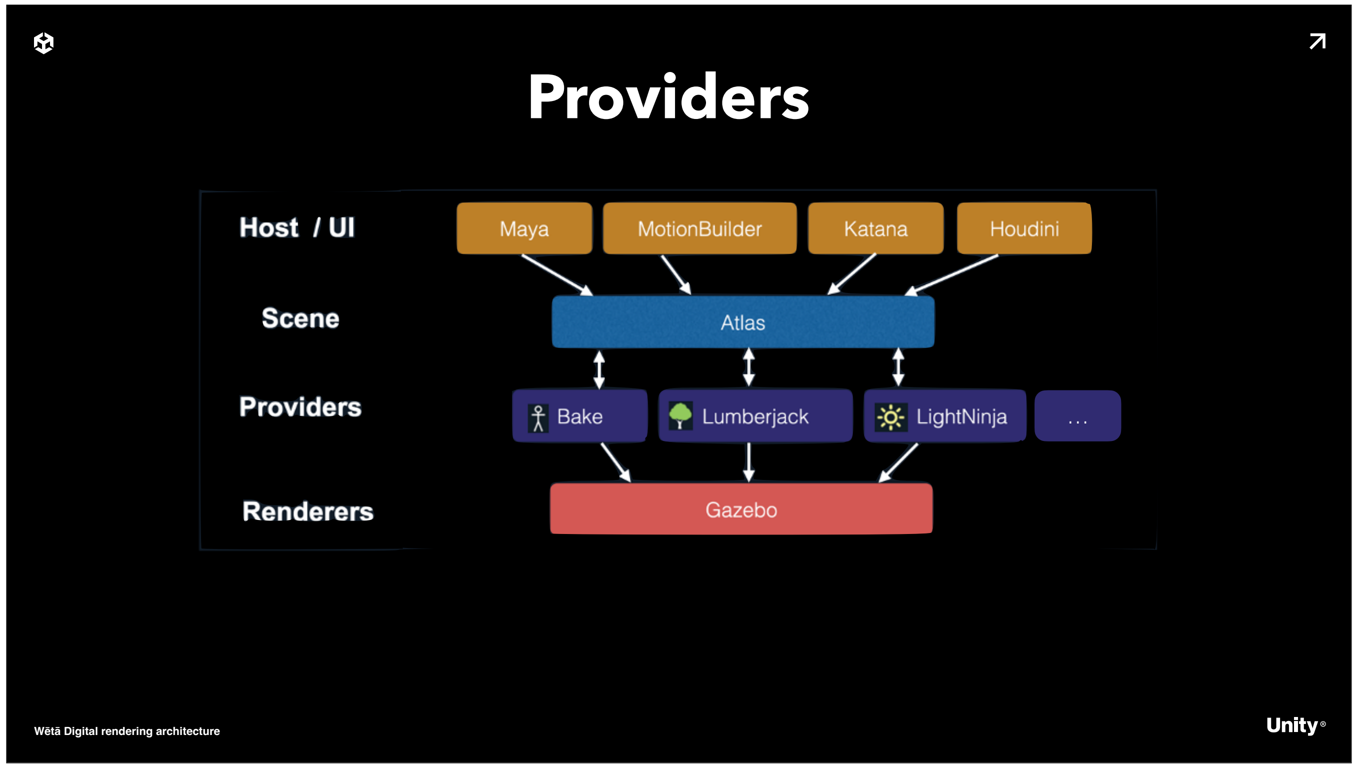
Providers

- How does Gazebo get data?
  - "Data Provider" plugins provide data to Gazebo without knowledge of how it will be rendered
  - Modelled on, and similar conventions to Manuka's API
    - Conceptually similar to PRMan's RI-interface
  - Clean separation between data and rendering (Gazebo knows nothing of the pipeline)
  - Most data provider plugins for Gazebo and Manuka (Weta Digital's offline renderer) are built from the same code (e.g. Bake, Lumberjack)
  - Binary backwards compatible

So how does Gazebo read the movie data?
We have "Data Provider" plugins for this.
These plugins have no knowledge of how Gazebo will render the data.
And because of these plugins, Gazebo has no knowledge of where the data comes from.
so this maintains a clean separation between data and rendering.

A similar concept is used in Manuka, meaning that most provider plugins, for each renderer, are built from the same code. This is how we keep things consistent.

Our providers are binary backwards compatible also in order to accomodate shows that can be locked-down at any point in time



Providers

Wētā Digital rendering architecture

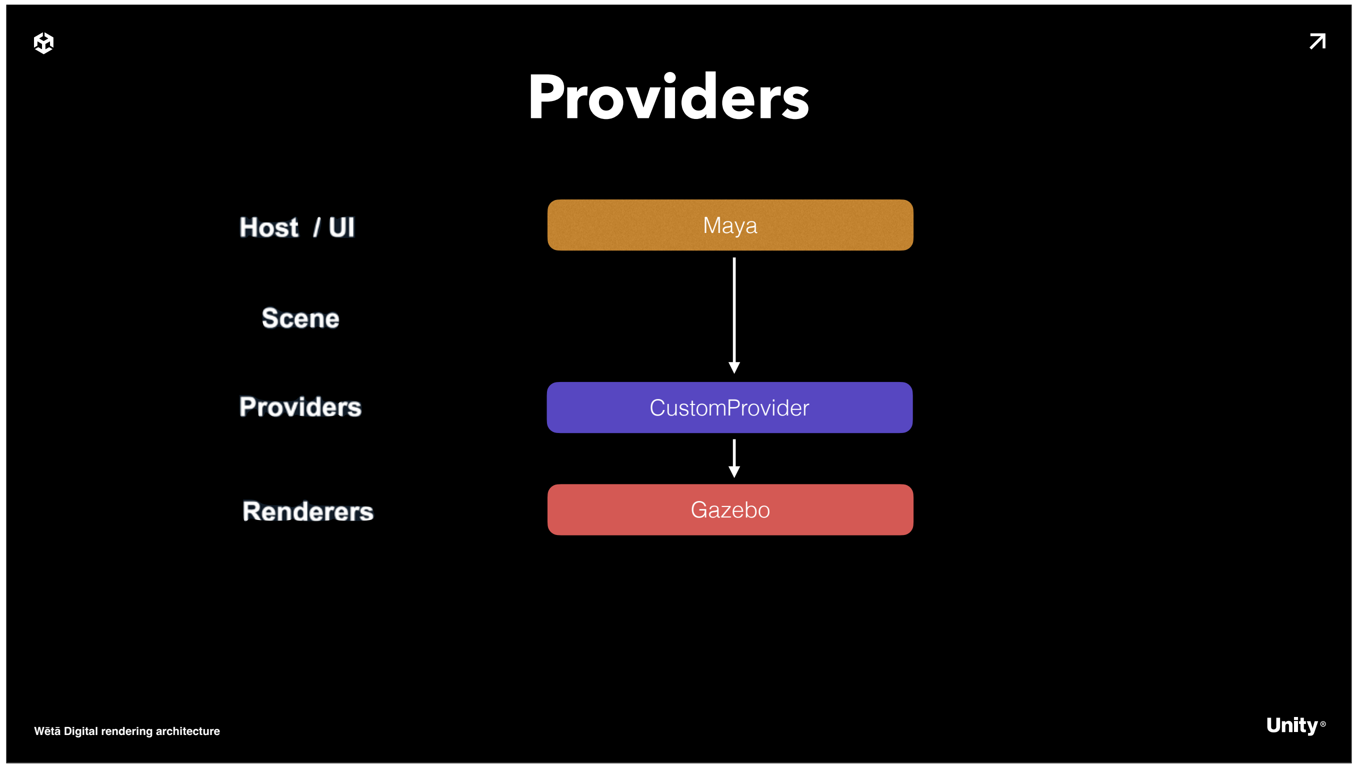
Unity

To illustrate this concept, we have a diagram.
At the top we have our host applications, which provides the user-interface layer.
Data is fed down to our scene abstraction layer, called Atlas, which is conceptually similar to USD.   We also have a similar setup for our USD hydra delegate.

Provider plugins then take over interpreting the scene and providing data to each renderer, in a way that suits the purpose best. For example here LightNinja provides lights and can create both physical and non physical lighting.

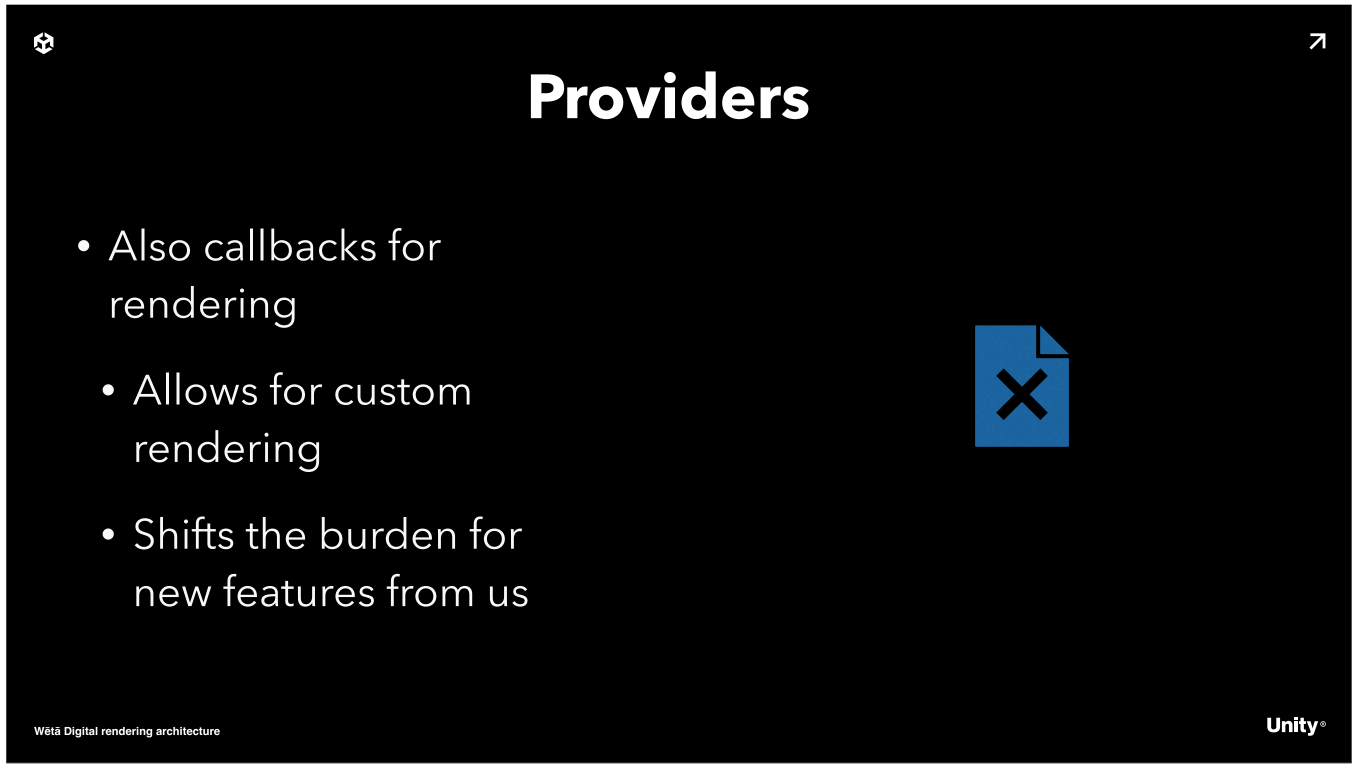
Low-poly geometry can be provided to Gazebo if need be, or if the user requires hero level geometry that is no problem either.

~~These ellipses on the edge here indicate many more plugins, and we do have many. Fur, Hair, Feathers, Simulation data, the list is long~~



Providers

Plugins can bypass our scene description and also talk to and interface with Gazebo directly. This is handy if a user just requires a light-weight plugin that doesn't need to interact with any other of Wētā's other internal tools or plugins or just wants to be able to operate in a set-up that does not have any other integrations enabled. This flexibility allows users to create providers that suit their purpose and needs for the current show or shot that they are working on.

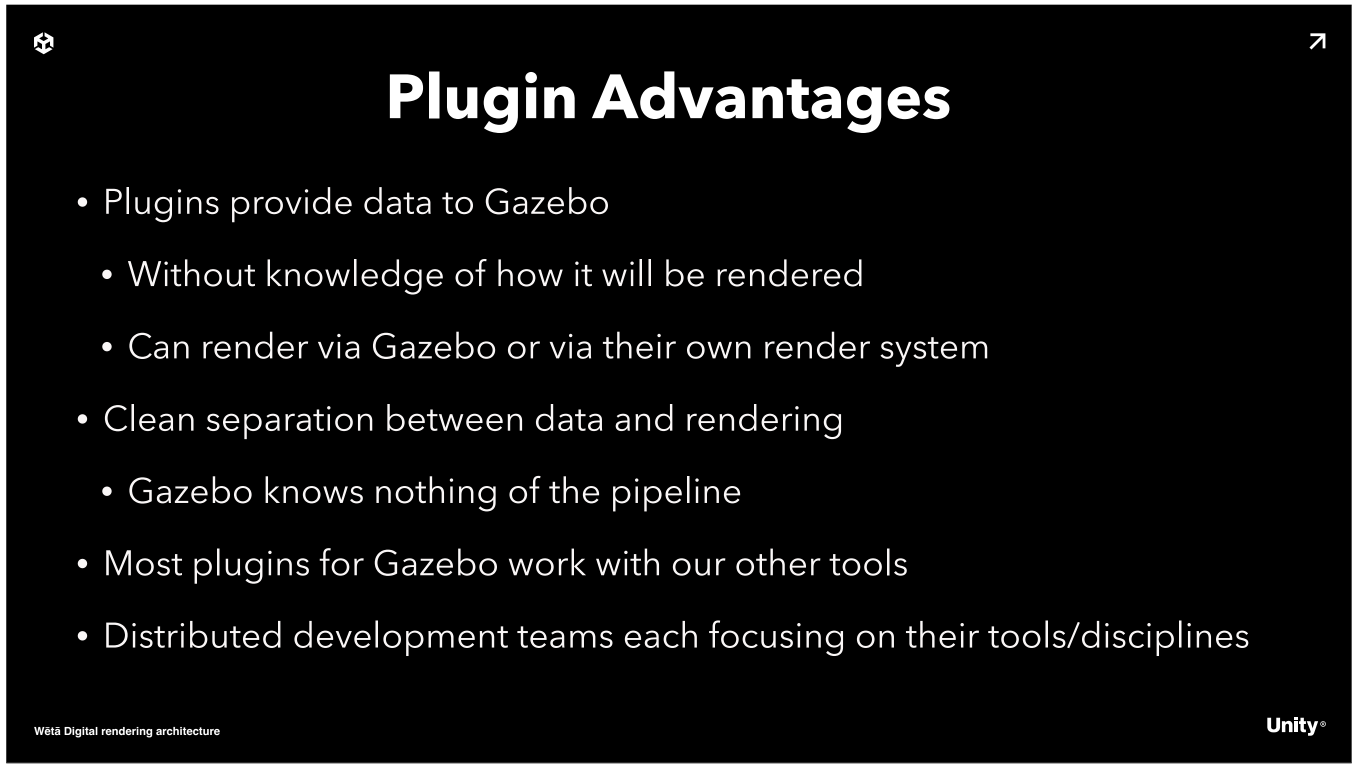


Providers

- Also callbacks for rendering
  - Allows for custom rendering
  - Shifts the burden for new features from us

Providers can also ‘add’ to the final render in callbacks, which allows for rendering of custom data or locators or other UI additions

This is a relatively basic render callback, and it therefore does have its downsides and can be quite restrictive but having it does allow for provider implementors to overlay new rendering effects without us needing to do the work. This is something we look to improve in the future to provide more customisation and flexibility to our users.



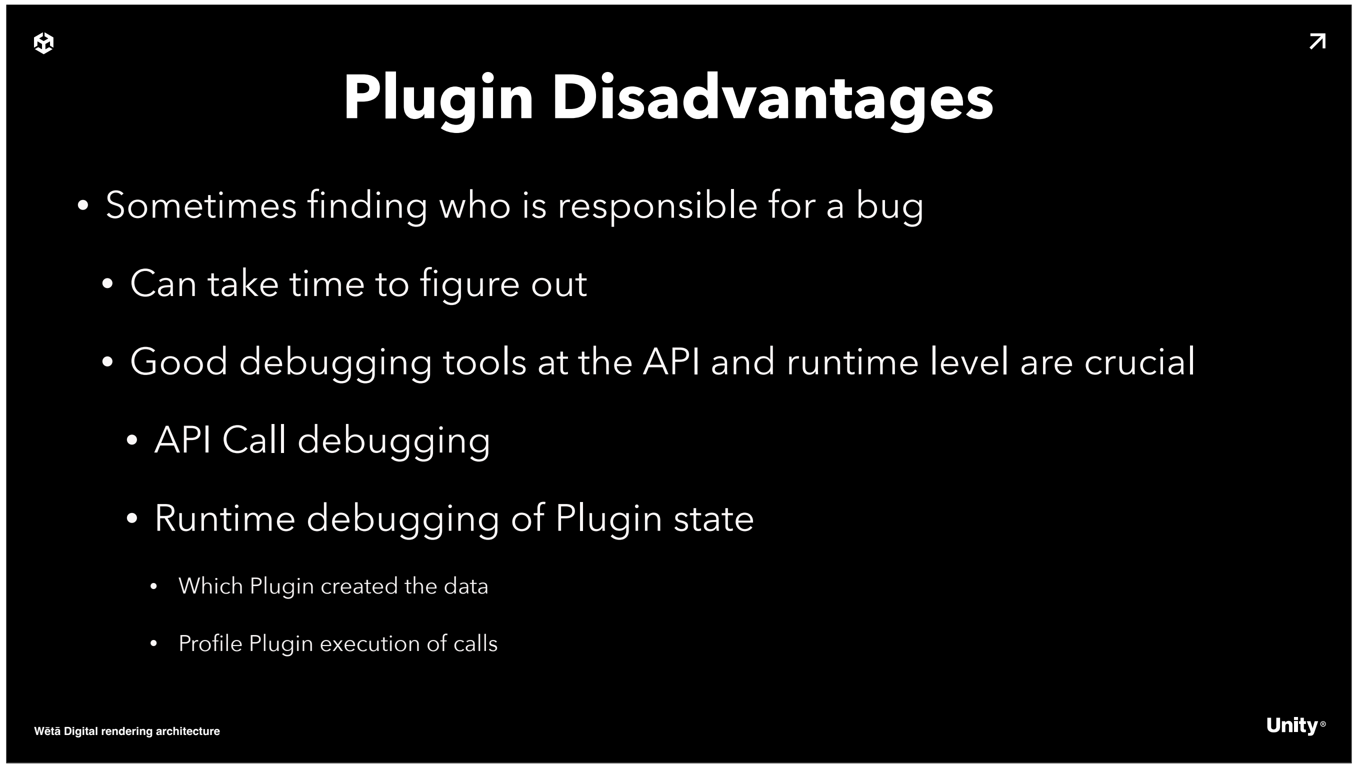
Plugin Advantages

- Plugins provide data to Gazebo
  - Without knowledge of how it will be rendered
  - Can render via Gazebo or via their own render system
- Clean separation between data and rendering
  - Gazebo knows nothing of the pipeline
- Most plugins for Gazebo work with our other tools
- Distributed development teams each focusing on their tools/disciplines

Unity®

Taking a step back, plugins give us a clear separation between the code providing data to Gazebo, and how that data is to be read. Plugins don't need to worry too much about how their content will be rendered, they just give us the data with a bit of shading information, set up some properties and that is it. Conversely, Gazebo isn't aware of the rest of the pipeline and can work in its own bubble -

We don't have to worry about generating new tree algorithms, or fixing skinning problems, or building sky models - this allows us to concentrate on what we need to do, which is attempting to push pretty pixels to the screen as fast as possible.



Plugin Disadvantages

- Sometimes finding who is responsible for a bug
  - Can take time to figure out
  - Good debugging tools at the API and runtime level are crucial
    - API Call debugging
    - Runtime debugging of Plugin state
      - Which Plugin created the data
      - Profile Plugin execution of calls

Having a plugin system is incredibly useful for us and helps us manage, design and debug Gazebo in a place where many multiple shows are being worked on, each with their own set of tools and software at varying versions. This however is not a magical answer to all our problems and there can be quite a few disadvantages to a system like this, although with enough time and patience; a lot of those could be solved. Having a separation between providers and Gazebo means that if there is a rendering issue or crash, there is some extra ground work required to track down where the issue has stemmed from - but at the same time having an explicit API layer means we know every location in which the provider is modifying Gazebo's state and issues can be tracked down relatively quickly with enough tracking and debugging information.

Due to a separation of responsibilities, regression testing can be a bit tricky as you need to test that your API operates as intended with the API features that are used by plugins and and the same time you want to make sure that you are not breaking the plugins also that use these features.

Overall, this system works pretty well for us due to the locking down nature of all the shows we work on that are on various different versions of different plugins and enables us to quickly iterate back over older versions to find where regressions have been introduced.

Wētā Digital rendering architecture

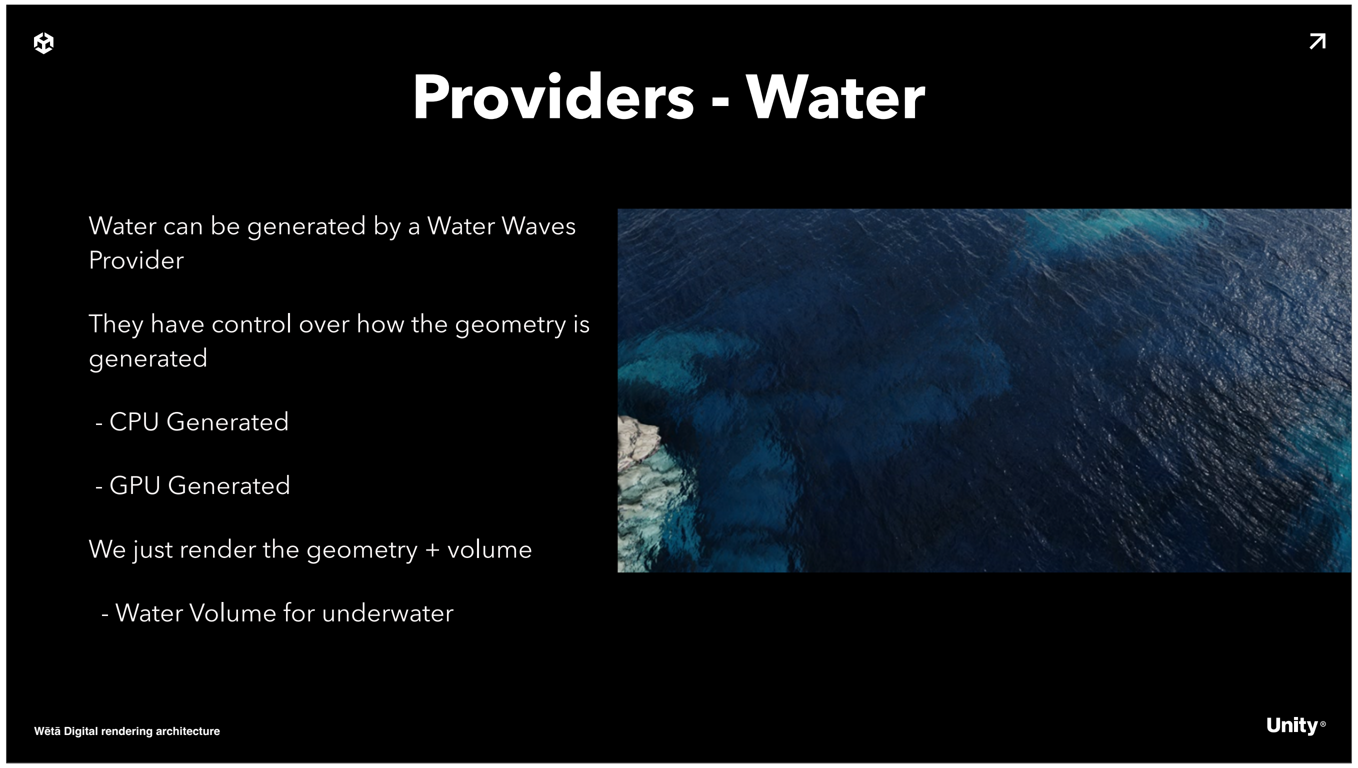
Unity®

In this example we have a point cloud distribution that can be used to visualize node output from our set dressing tool, Scenic Designer

Providers - Water

For another example of a plugin we have water…. -

A unique water rendering solution was developed for Avatar: The Way of Water that allows for more accurate rendering of light and its interaction within the volume and the water waves. This is integrated with all of Gazebo's lighting and allows for runtime visualisation of underwater rendering on stage which was crucial in the rendering of this movie given how much water exists in it



Providers - Water

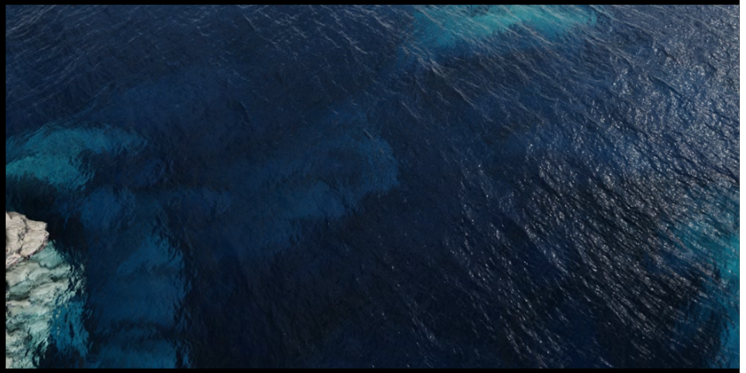
Water can be generated by a Water Waves Provider

They have control over how the geometry is generated

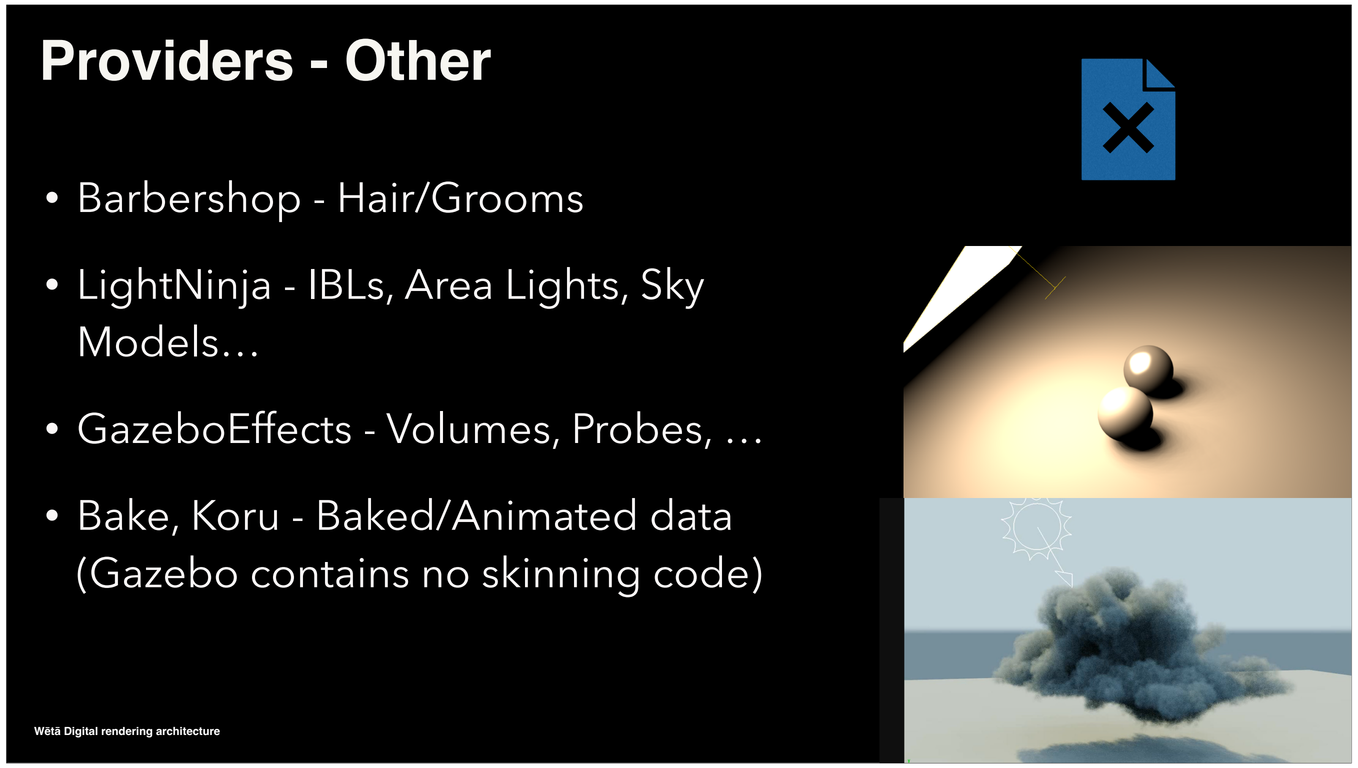
- CPU Generated
- GPU Generated

We just render the geometry + volume

- Water Volume for underwater

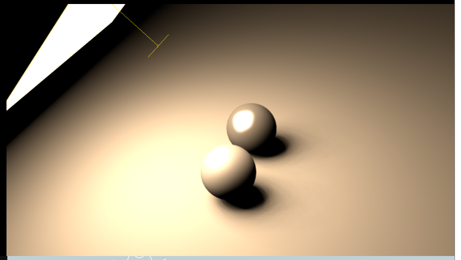


Water is mostly handled by our Water Waves provider which is responsible for generating the geometry that gazebo will render with and applying any shaders/settings to the water and water volume. This gives flexibility to the provider in that they can change how they generate the source data for the water geometry without having to wait for us to make the change. Another example is that the water wave data that is passed to Gazebo could be used by other providers to do effects such as boat wake or perfect contact of boats on the water.



Providers - Other

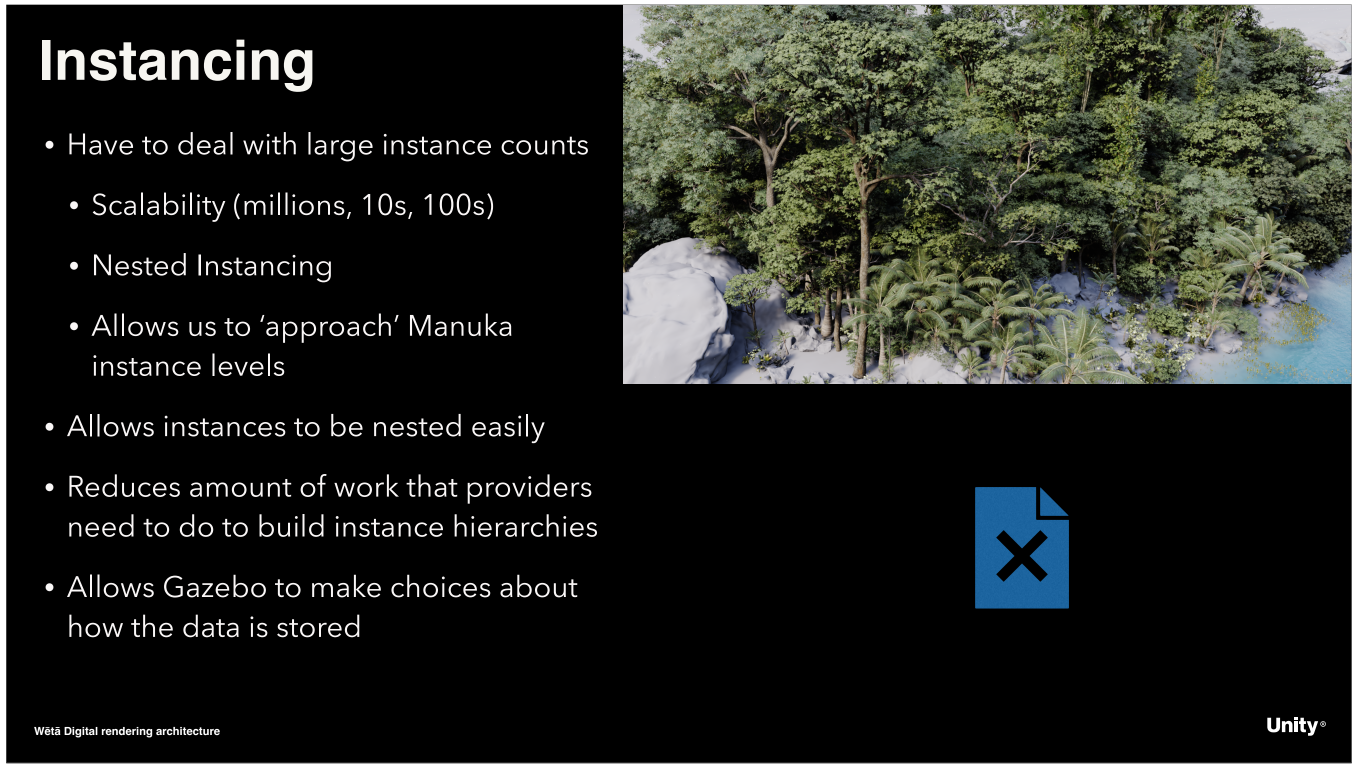
- Barbershop - Hair/Grooms
- LightNinja - IBLs, Area Lights, Sky Models…
- GazeboEffects - Volumes, Probes, …
- Bake, Koru - Baked/Animated data (Gazebo contains no skinning code)



There are many more providers available and used to augment our scenes - some examples include providers that can calculate all the grooming for the hair and provides the geometry to us in a form that can be curves or whatever geometry works for them, and They can then apply their own shading to this geometry that is based off a base hair bxdf.   There are providers for lighting to provide lighting information such as Area Lights and custom sky models in the form of generated sky models. Animated or Baked data can also be fed to Gazebo - we don't have any skinning code inside of Gazebo itself

Instancing

Instancing is an important part of the workflows here and in Gazebo we need to be scalable and flexible in order to accomodate the varying use cases which could range from rendering simple small tree LODs and billboards that might just one or two draw calls, to higher order assets and hero level assets which would involve a huge number of vertices just for a leaf alone, and then that leaf is instanced hundreds of thousands of times for just one tree top.



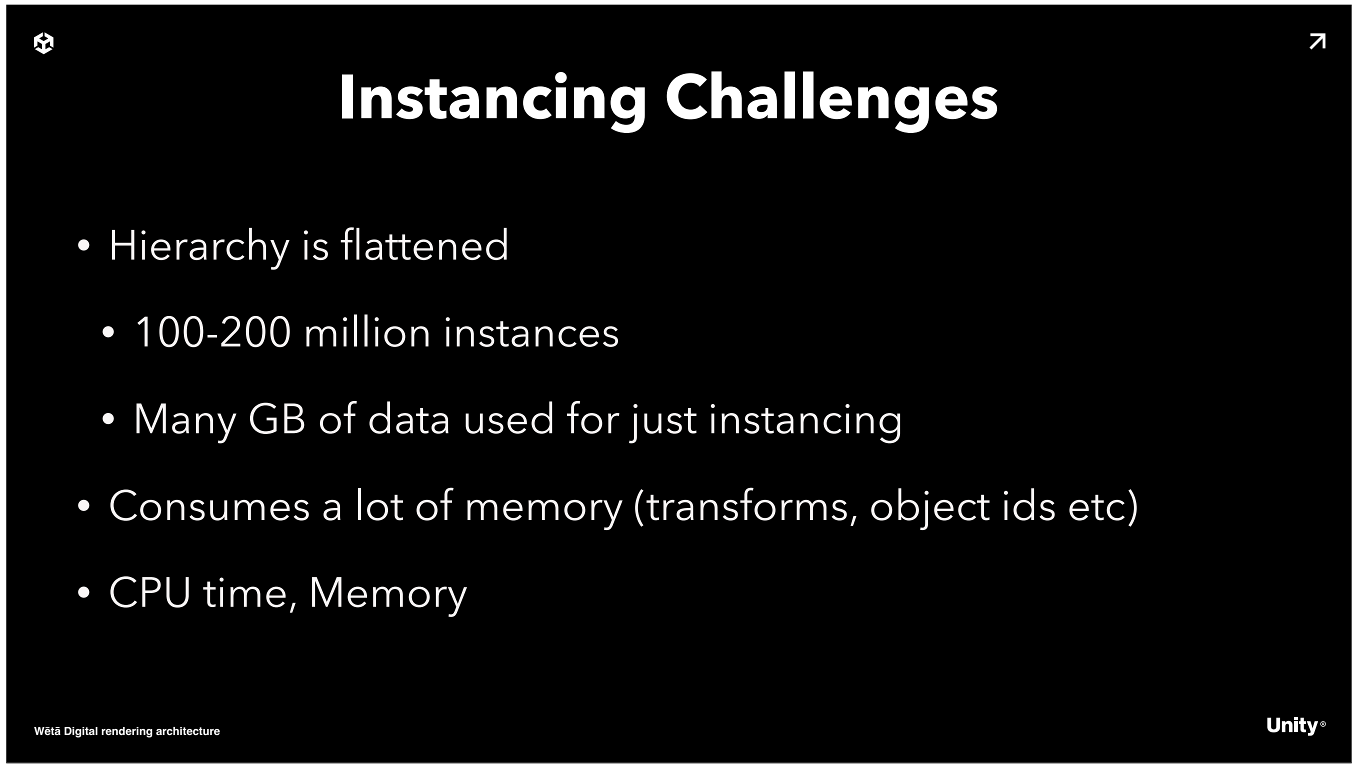
Instancing

- Have to deal with large instance counts
  - Scalability (millions, 10s, 100s)
  - Nested Instancing
  - Allows us to 'approach' Manuka instance levels
- Allows instances to be nested easily
- Reduces amount of work that providers need to do to build instance hierarchies
- Allows Gazebo to make choices about how the data is stored

Wētā Digital rendering architecture

Unity®

To enable this we have a system that allows for the construction of instanced and nested-instanced data to produce complex scene layouts that can then be rendered on the GPU. This system needs to be able handle tens of millions of instances in a performant and memory friendly way. Most of the building of the instancing or nested instancing data is done via our API and this approach allows our providers to create nested instance layouts easily, allowing for re-use and better structure from their domain. Gazebo then has the freedom to decide how this instancing data will be managed, stored and rendered.



Instancing Challenges

- Hierarchy is flattened
  - 100-200 million instances
  - Many GB of data used for just instancing
- Consumes a lot of memory (transforms, object ids etc)
- CPU time, Memory

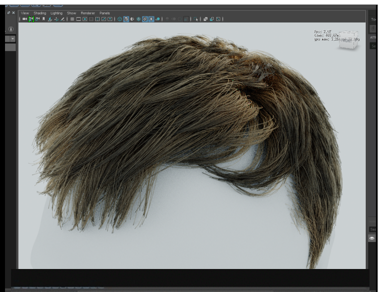
Wētā Digital rendering architecture

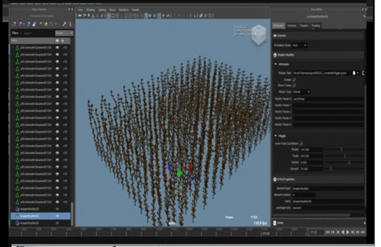
Unity®

This does come at a cost as we currently need to flatten this data down so that it is in a format that we can render with. This is less than ideal and mostly a legacy issue which we have plans to improve in the near future. Memory is a huge cost also when reaching 100s of millions of instances and often compression is not an option for us currently due to the sheer size of some of our shots.

Other features

- OIT
- Shading Modifiers
- Colourspaces/Imaging
- Physlight



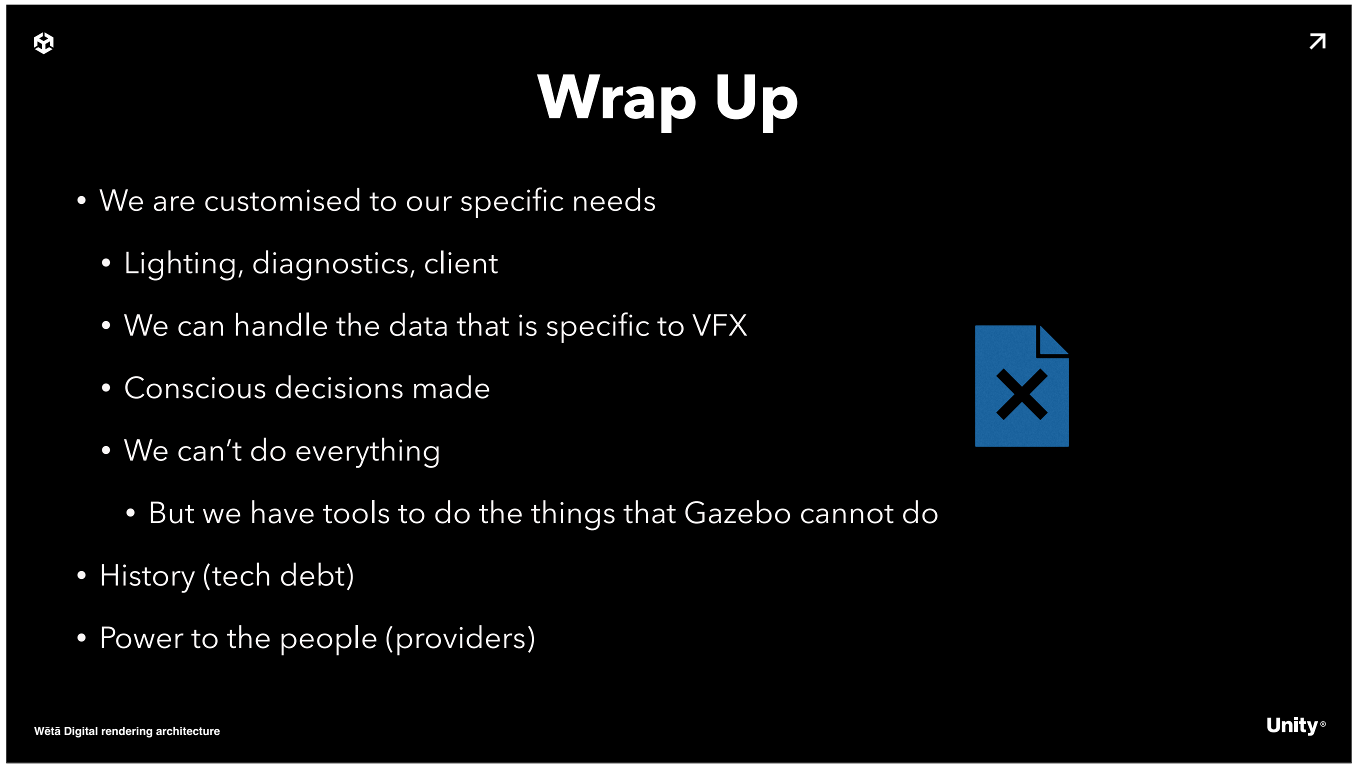


There are many more unique systems within Gazebo which we won't have time to fully cover here which include:

Order Independent Transparency is achieved in Gazebo via Depth Peeling and Moment Transparency. Moment Transparency was developed here internally at Wētā although there was a similar technology designed around the same time independently.

Shading modifiers are a unique solution to making changes to authored shaders once they are in use on a show. These modifiers allow end users to modify input parameters and shading behaviour on a shader without having to touch the source shader. This allows for visual variation on materials on Stage without the need to re-author the source assets and paying the time cost of waiting for this. This has been a huge boon for us for Avatar: The Way of Water where on-the-fly modifications to materials on stage needed to be rolled out in a timely fashion and we've been constantly surprised by what has been achieved by our users with this system including a range of effects such as shoreline on water, boat-wakes, and changing displacement on the water around boats and shoreline. This also helps with per-instance variation as it adds another layer of configurability that we don't have to worry about.

Colourspaces and Physlight are areas that have been presented in other talks as they are their own complex topics but integrated when in to Gazebo this gives us the ability to consume data in the many different colour formats and output in to the space required by the show and or viewport.



Wrap Up

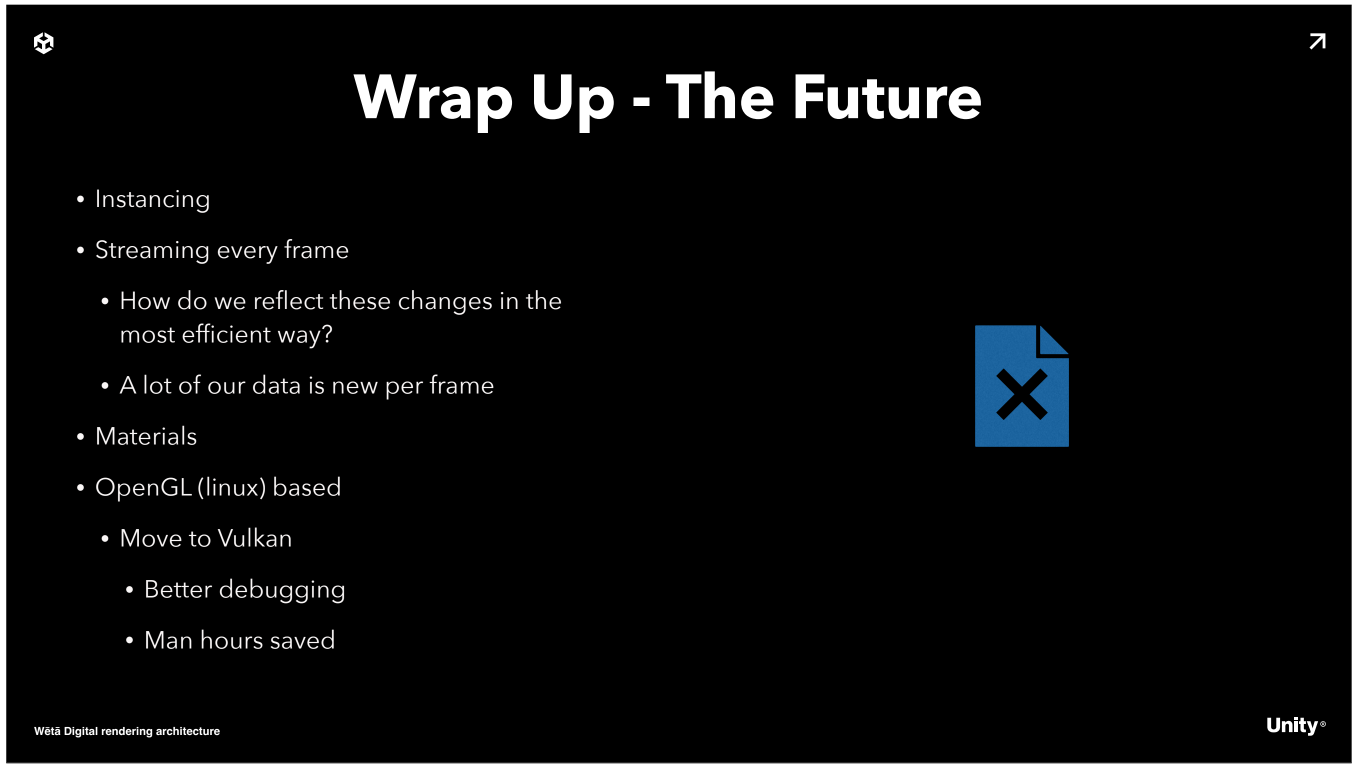
- We are customised to our specific needs
  - Lighting, diagnostics, client
  - We can handle the data that is specific to VFX
  - Conscious decisions made
  - We can't do everything
    - But we have tools to do the things that Gazebo cannot do
- History (tech debt)
- Power to the people (providers)

Unity

In the end, we are a realtime rendered that is highly customised for the needs of the Weta pipeline. This, I think is a good thing. We're agile, and flexible to our users needs and bug reports. We can handle data that is specific to VFX, and that's all we need to do. Gazebo is a product of many conscious decisions made over time, and we can't do everything. But.,, that's ok - we have tools to do the things that Gazebo cannot do.

Being such an old product we do have to deal with a lot of Technical Debt, and we've probably not been as good at resolving some of it as we should be, but it's a complex problem and we are slowly making strides to address this with varying levels of success on how disruptive we are to our end users.

The provider concept within Gazebo is very powerful, and as mentioned there are advantages and disadvantages but as a whole it works quite well in our situation and we have future plans to give more power to these providers to allow them to leverage more of Gazebo for their specific situations and scenarios



Wrap Up - The Future

- Instancing
- Streaming every frame
  - How do we reflect these changes in the most efficient way?
  - A lot of our data is new per frame
- Materials
- OpenGL (linux) based
  - Move to Vulkan
    - Better debugging
    - Man hours saved

We've learned a lot from Avatar, and are already starting our journey in to the next generation of rendering within Gazebo. Instancing and instancing related data plus shading continues to be a huge focus for us since our users do and should be able to push the current limits within Gazebo both in our architecture and the hardware they use.

A lot of our data is streamed in per frame - how do we deal with the growing requirements and use cases in the most efficient way that is acceptable to both ourselves and our users? Instancing is another huge issue for us, how do we handle the massive amounts of instancing that is required of us, in a performant and efficient way? This is all ongoing research and development for us.

Materials are also a key area here - some of our material tech has stagnated a bit over the years so how do we continue to strive to push the state of the art in this area whilst attempting to better and more closely match what Manuka can do so that we can further bridge the gap between our two technologies.

Gazebo is traditionally an OpenGL renderer on Linux, and has served us quite well for a long time, but as we started using extensions that were more outside the core scope of OpenGL and we started pushing requirements for Wall rendering we've encountered hard to track down crashes and bugs that have bled an uncountable number of man hours from our development. Along with this, vendor support is less, and new features are not always made available to OpenGL so we made the decision to do the move to Vulkan which has been happening over the last few years as we still need to support OpenGL for some various reasons. Also, there's probably another talks worth of content talking about the reasons we moved, the pros and cons.., and pain of retro-fitting an existing engine vs starting a new and the amount of work involved in doing this but that will have to be for another time.

There are many exciting challenges on the road ahead and Gazebo is in a unique position to investigate and potentially implements some of these due to where we exist in the realtime rendering engine space.

Thanks!

- Wētā FX
- Wētā Digital Rendering & Engineering Dept
  - All past and present
- Unity

Wētā Digital rendering architecture

Unity

Thanks for tuning in to this talk and seeing a glimpse in to the world of Manuka and Gazebo - we'd like to thank the artists at weta, without whom we would not be able to produce the beautiful images, but also the render and engineering teams both past and present. The work presented here in both of these talks spans a decade of work by everyone, thanks again - we'll be happy to answer any questions