SD322461

Advanced data visualization using the Forge viewer

Kean Walmsley
Platform Architect, Autodesk Research

Learning Objectives

- Understand the history of Autodesk's research into visualization of IoT data in 3D
- See how the Forge viewer supports advanced data visualization
- Identify the open source components published by Autodesk Research
- Learn how to integrate these components into your own Forge application

Description

Over the last decade, Autodesk Research has been developing advanced techniques for data visualization in a 3D environment. One example of this is Dasher 360, a Forge-based application that can display sensor data captured by sensors in buildings (or on infrastructure such as bridges) in a 3D context. In recent months we've been working to publish the core components that display this rich 3D data, for instance to spatially locate sensors in a 3D model and to map large amounts of IoT sensor data to the surface of 3D objects via animated heat-maps. This session focuses on how Forge developers can integrate these components into their own applications, delivering first-class visualizations of their customers' data.

Speaker(s)

Kean Walmsley is a Platform Architect and Evangelist working for Autodesk Research. He blogs and tweets about developing with Forge, AutoCAD and other Autodesk technology, especially with respect to IoT, Generative Design, VR and AR.

kean.walmsley@autodesk.com
https://keanw.com
https://twitter.com/keanw
https://linkedin.com/in/keanw

Introduction

When we submitted the proposal for this AU2019 class, back in May, the Dasher 360 team was 100% behind releasing the source for the entire project via an open-source license. I was a bit wary of proposing a class that relied on this happening, so when I submitted this proposal I thought I was being unnecessarily cautious – feeling it was better to err on the side of caution – opting to talk in a general sense about using open source components published by Autodesk Research.

On the one hand, I'm glad I was cautious: it turned out not to be possible to get the necessary approvals in place for Dasher 360 to be open-sourced in time for the conference. On the other hand, larger discussions around the future of Dasher 360 have been happening just before the conference, meaning it was ultimately not possible to publish *any* open source components before the deadline for posting this handout. Much to my personal chagrin, I might add.

Given the intense interest in this area, the class is absolutely going ahead as planned. However, the last two learning objectives have – at the time of writing, anyway – not proven possible to deliver upon, at least not in this handout. There is some dwindling hope that some components will be published between now and the conference, but we'll see that on the day itself.

While not exactly "open source", we will look at techniques and code that have been published via my blog, over the years, and go into some level of detail that has not been seen before. So I still believe that people will derive enough value from the session, even if not being exactly as originally proposed.

Given the level of uncertainty regarding the specific content of the class, my fallback option for this class handout is to re-publish the excellent material created by Simon Breslav for his Forge DevCon 2017 class entitled "*Using Forge for Advanced IoT Visualization in Dasher 360*". I will follow-up after the conference by posting updated handout material to my blog.

Project Dasher History

Autodesk Research

Autodesk acquired its Research division along with Alias in 2006. The initial team comprised of just 5 researchers, but over time this has grown to over 100 staff in 5 countries: Canada, USA, UK, China and Switzerland. The organization is comprised of a number of groups.

The goal of the department is to conduct cutting-edge research and to represent Autodesk as a thought leader in academic and industrial domains. To do this, researchers in different groups publish research papers in journals and at conferences, as well as forming partnerships with customers and academic institutions to push the boundary of current technologies. Increasingly the output of Autodesk Research's activities is being made available to customers: examples include Autodesk Meshmixer and Autodesk Sketchbook Motion (which was developed as Project Draco). It's our hope to see more such "technology transfers" occur during the coming years.

History of Project Dasher

Here's the original description of Project Dasher that we used when started the project back in 2009:

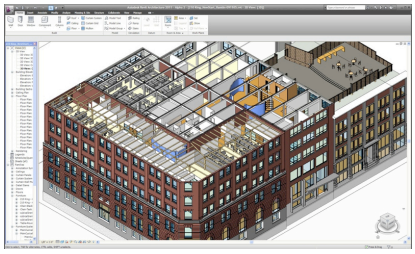
Project Dasher is an Autodesk research project using a BIM-based platform to provide building owners with greater insight into real-time building performance throughout the life-cycle of the building.

A sustainable building is not a fixed ideal, but a moving target that must be reassessed on an ongoing basis in order to respond to the ever changing patterns of its occupants and its context. While building performance tools have traditionally focused on the simulation and evaluation of a specific design, we are witnessing a growing need for tools that can help us to continuously evaluate and verify building performance. Today, most buildings are equipped with sophisticated Building Control Systems (BCS) that collect data from thousands of end-points. These systems help building operation managers maintain buildings by minimizing long-term operational cost ensuring occupants' comfort. However, a key challenge is to define methods of organizing, studying and communicating data, while coping with perpetual changes inherent in any commercial building.

In this context, we need a more integrative approach to maintain the complex balance between our energy-saving measures and occupant comfort. Using BIM as in ideal platform for managing complex building information, Project Dasher aims to go beyond existing building dashboards to represent a comprehensive framework for monitoring building performance. Project Dasher acts as a visualization hub where collected data from various sources is intuitively aggregated and presented in 3D to enhance our ability to infer more complex causal relationship pertaining to building performance and overall operational requirements.

In a nutshell, Project Dasher was intended to be used as a visual analytics tool to help improve the performance of buildings. A detailed as-built Building Information Model was combined with sensors from a Building Management System and cubicle level sensors to give rich, in-context

visualization of building operations. This extends BIM to be used as a tool throughout the lifecycle of the building, rather than just during its conception and creation.



As-Built BIM

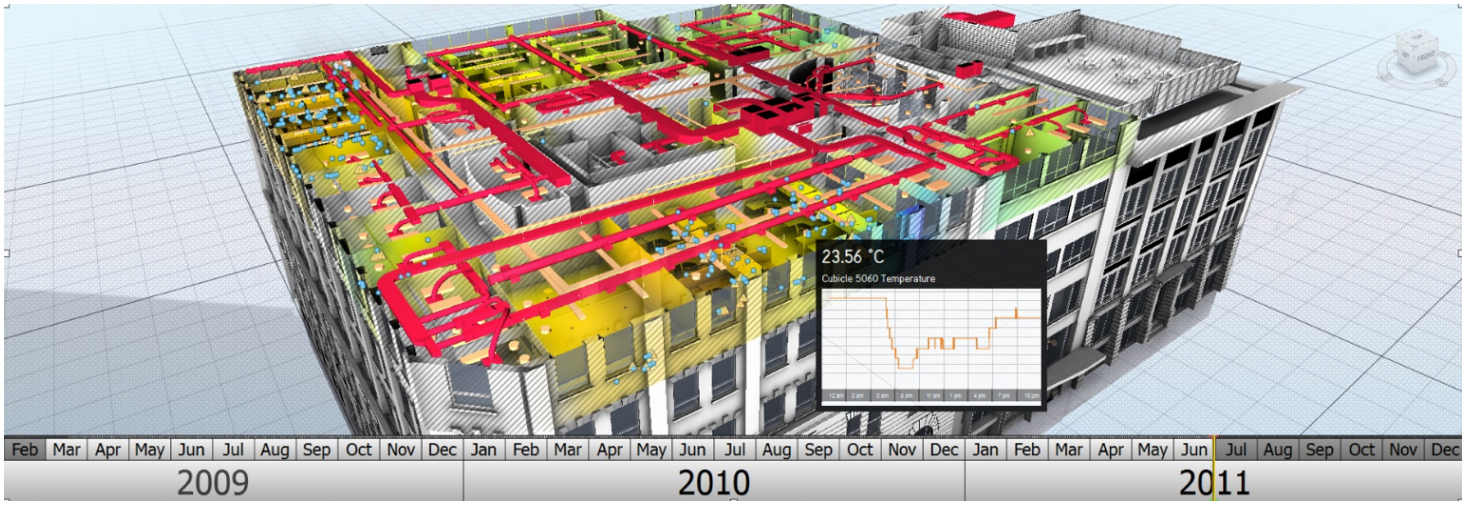
+

Building Data
Building management systems (BMS) and IoT

=

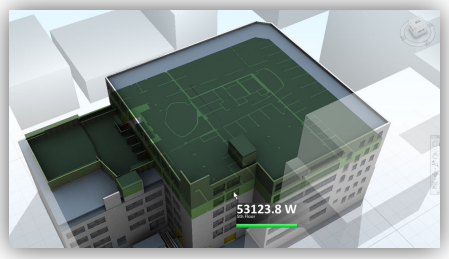
Project Dasher
A visualization too to help customers

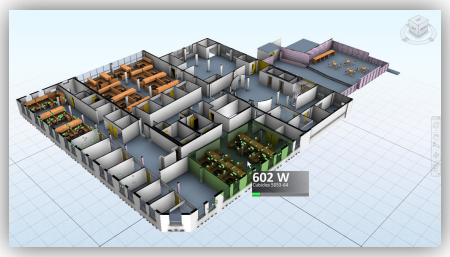
After a few years of working we produced a standalone software tool featuring many advanced visualization techniques.

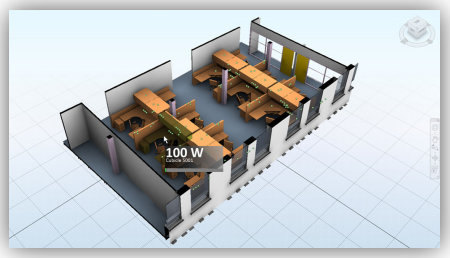


The tool allowed people to easily understand performance at different levels of detail, going from the entire floor of a building to a specific zone. With a key differentiating feature of Dasher being the display of sensor information in a 3D model, making it easier to absorb details of multiple sensors at once and view situations holistically, making it much easier to consume.

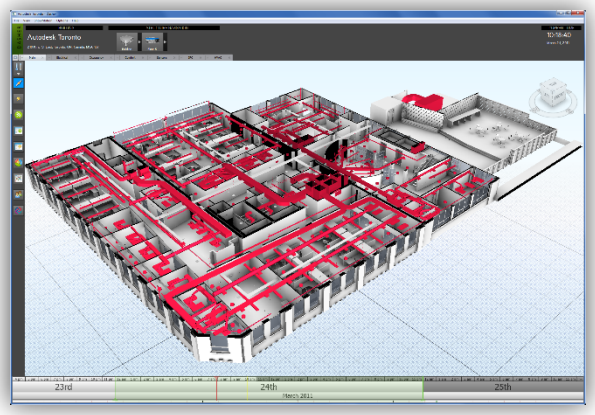
Floor / Area | Workgroup / Zone | Workstation / Individual

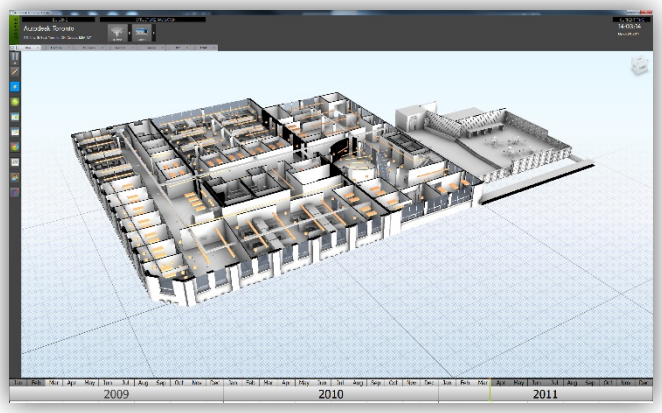






To extend the value of BIM beyond the walls, doors and windows, Dasher integrated with the electrical and mechanical systems, as well as the plumbing data about the building, allowing you to evaluate energy usage and comfort levels in the context of the building's infrastructure.

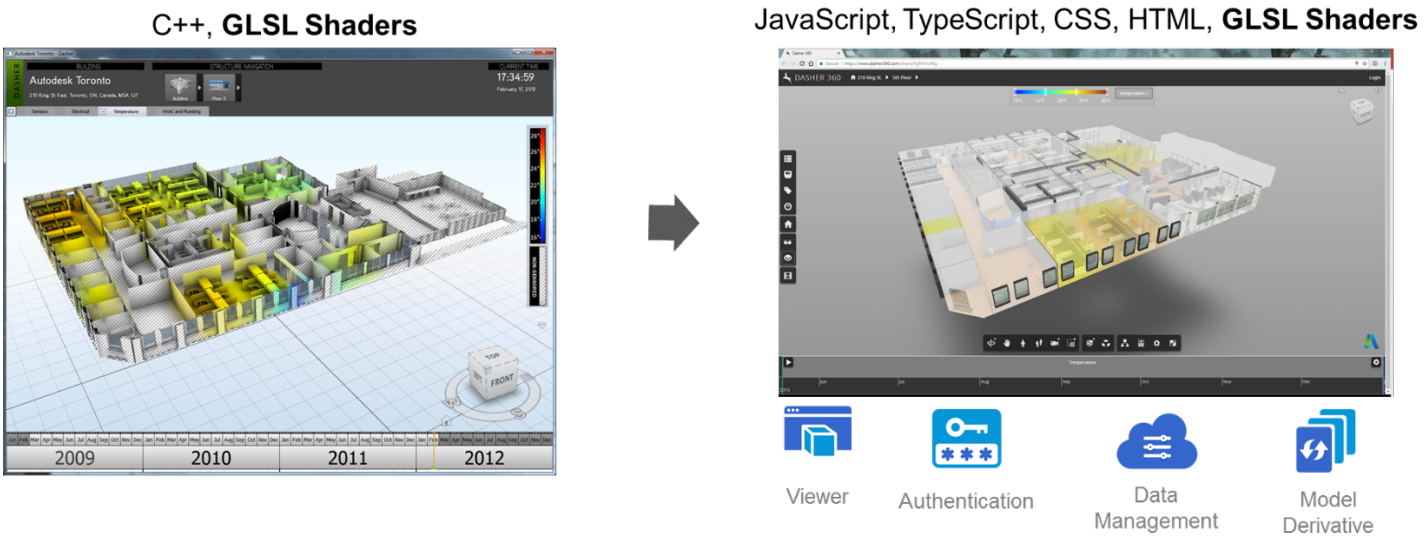




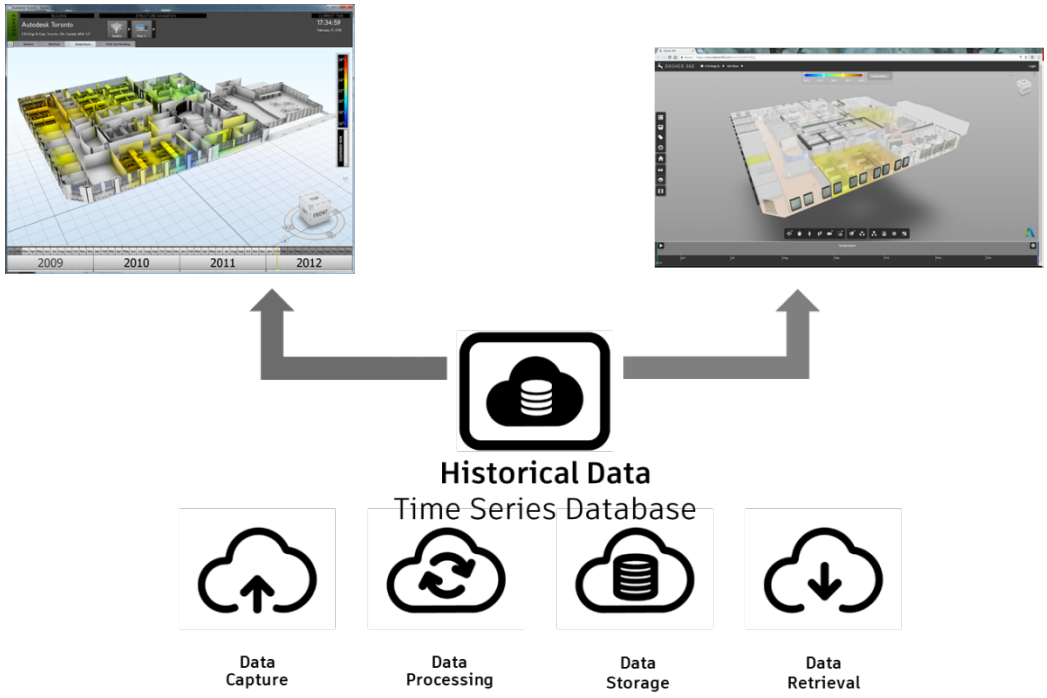
In summary, while existing Building Management Systems provide some access to sensor information, the way they present often lacks the context of 3D building information. Dasher builds on Autodesk's strength in working with spatial data to present this data in an intuitive way to building operators and facility managers.

Migrating to Dasher 360

With the company moving to the cloud, the question came if Dasher should really be a web application. The move would increase accessibility and scale. For example, it would grant building occupants the ability to see their resource consumption, not just building operators and facility managers. With a desktop application that requires high-end hardware, this is not really an option. It would also allow for new workflows and applications such as sensor commissioning, and construction safety analysis, which may require trades people have access to Dasher using mobile devices in the field. So, In April 2016, we started a proof of concept for the web-based version of the Dasher. Aside from some aspects of the GLSL shaders in the desktop, sadly most of the code from the desktop could not be used directly.



The underlying database for time-series data became a separate project, as it offered an opportunity of a separate service that could feed both desktop Dasher and the web-based version. We called that effort Project Data 360, a scalable time-series database for



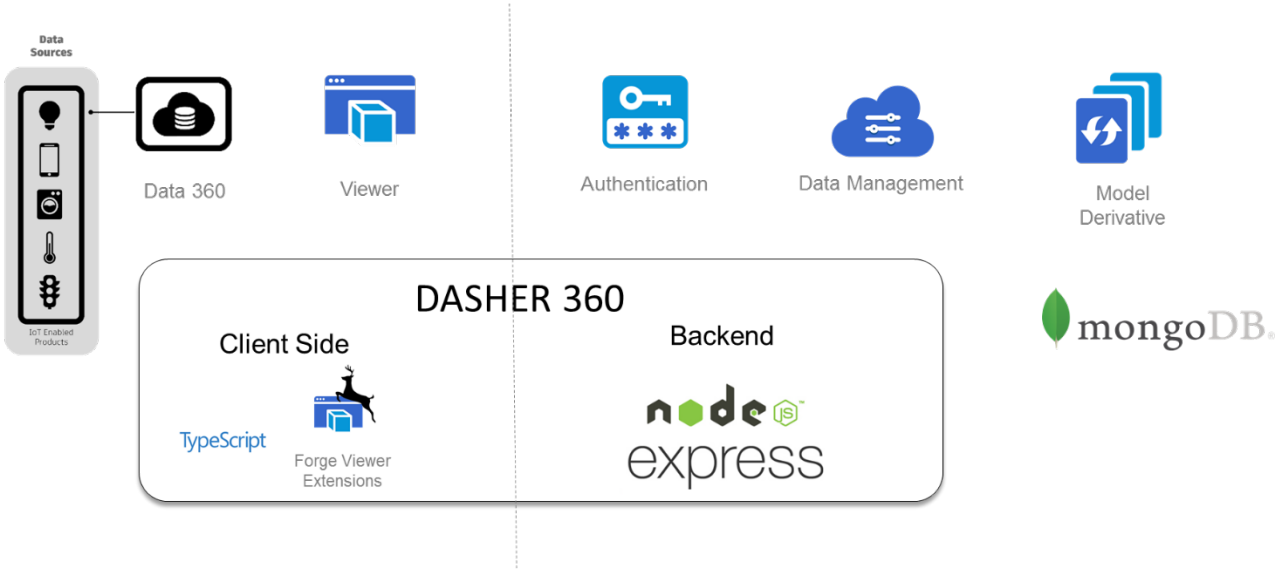
big data.

Dasher 360 Forge Usage

Architecture Overview

Dasher 360 is Project Dasher re-imagined for the web using Autodesk's Forge platform. The back-end of Dasher 360 is built as a Node.js (nodejs.org) application, on top of the Express (expressjs.com) framework. If you are new to node.js development, a nice tool that helps you get started on a project quickly is Yeoman (yeoman.io) which helps to create boilerplate code for a basic application, with many generators depending on the technology you want to use. For example, https://github.com/petecoop/generator-express makes a basic express in just a few steps. Looking at different generators can provide a good sense of different standard practices for organizing projects.

On the back-end, we use a number of Forge services, specifically Authentication, Data Management, and Model Derivative APIs. While on the front-end, we organize our code as a set of Forge Viewer Extensions. And although we write our extensions in TypeScript, this gets compiled to regular JavaScript before being deployed. Also, on the frontend we connect to the Project Data 360 service for time-series data.

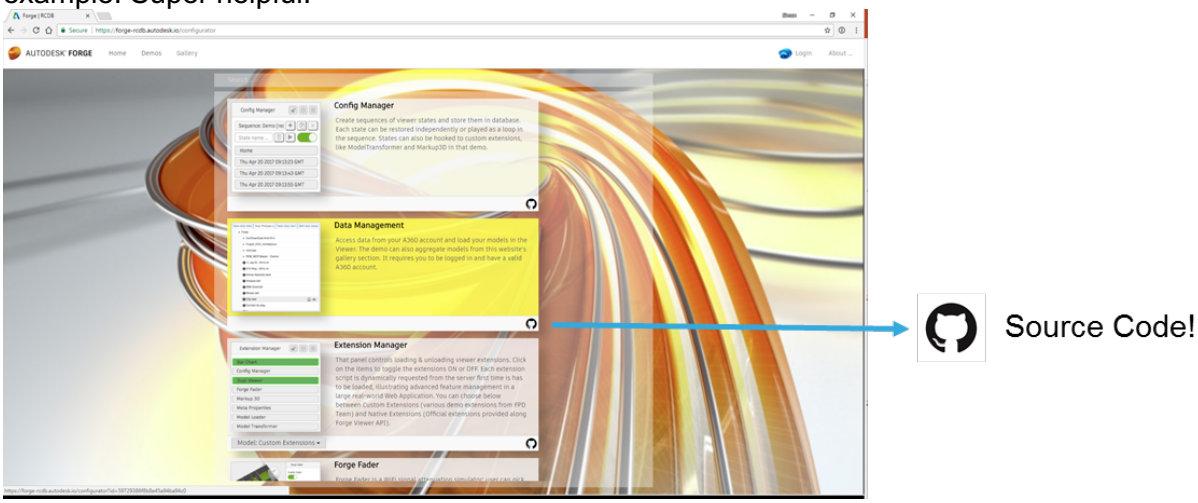


Forge Examples

The Forge team created a number of examples for using Forge services using different APIs. https://autodesk-forge.github.io/ or https://developer.autodesk.com are good places to start exploring Forge. The base organization that hosts all the open source Forge examples is at https://github.com/Autodesk-Forge

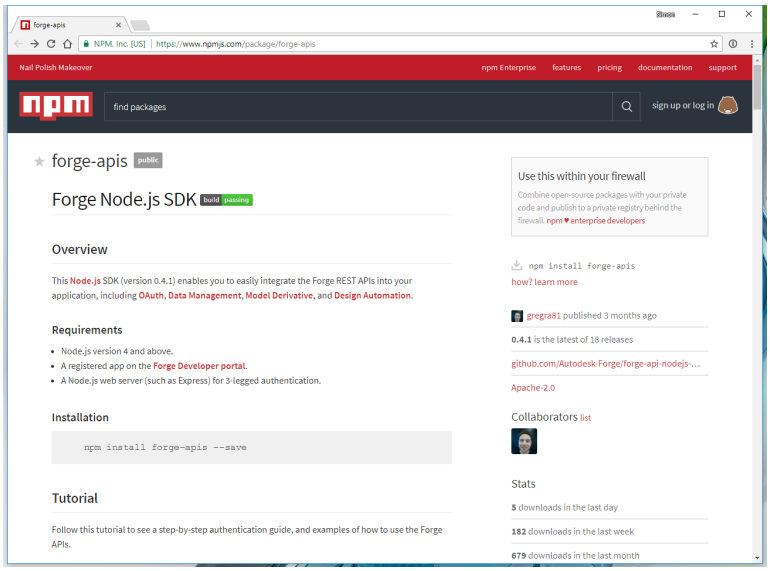
We specifically recommend exploring these two projects:

- https://github.com/Autodesk-Forge/forge-rcdb.nodejs Contains a lot of the examples needed to recreate most features in Dasher. A live demo of the examples from this project can be found at https://forge-rcdb.autodesk.io. One tip to note is that each separate sample has a little GitHub link, that jumps to the specific location in the codebase on GitHub, so that you don't have to search through the code to find the example. Super helpful!



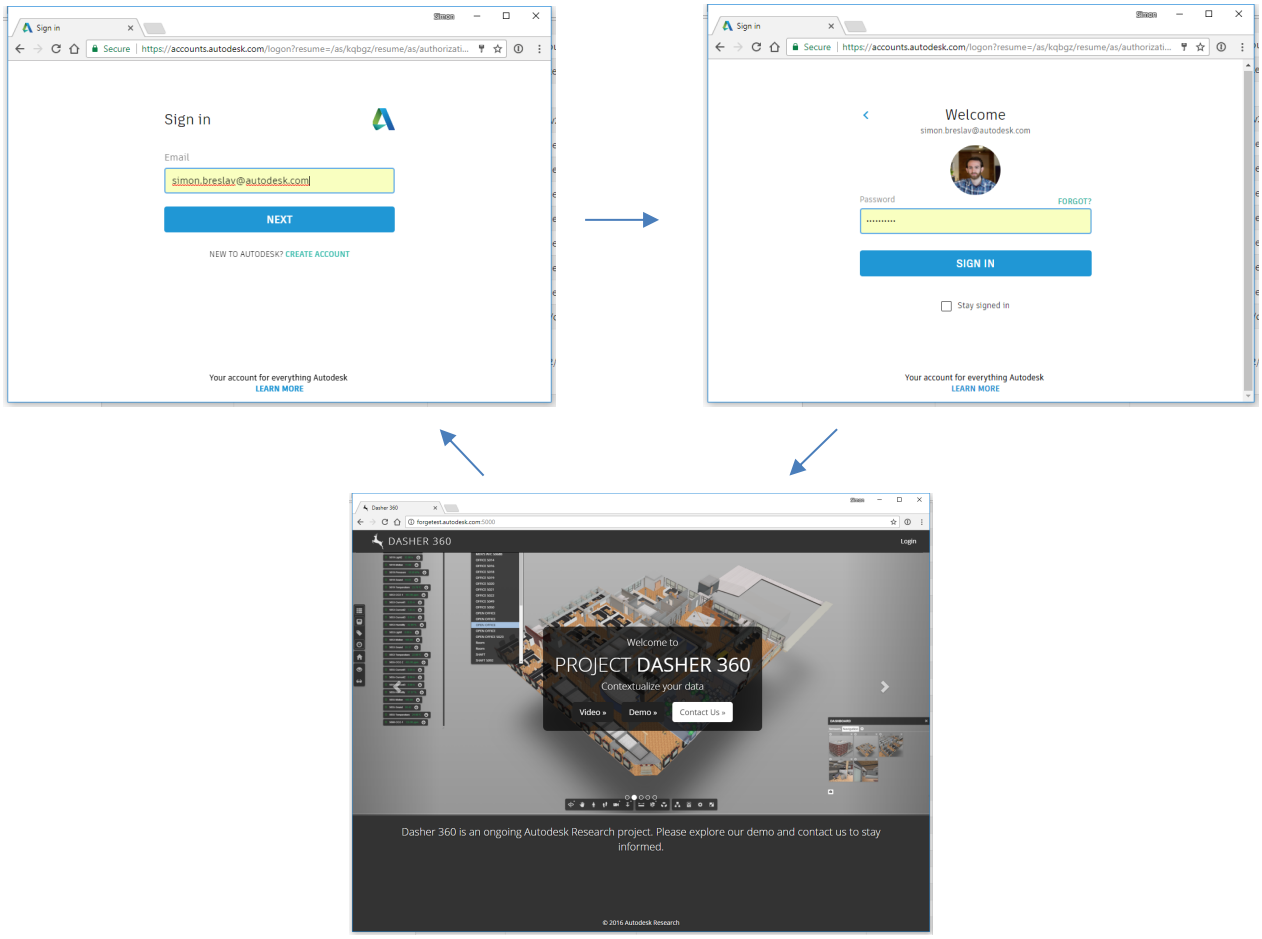
- https://github.com/Autodesk-Forge/model.derivative-nodejs-sample A simpler example that focuses on Data Management and Model Derivative APIs, and might be a good starting point. Live demo at https://derivatives.autodesk.io
- https://github.com/Autodesk-Forge/models.autodesk.io. A sample that shows how to work with Model Derivative API, where users can upload files to App-managed data buckets. Live demo at https://models.autodesk.io

Both of the projects demonstrate how to use a node.js npm package called forge-apis (https://www.npmjs.com/package/forge-apis) which wraps all the Forge services in one package with a uniform API.



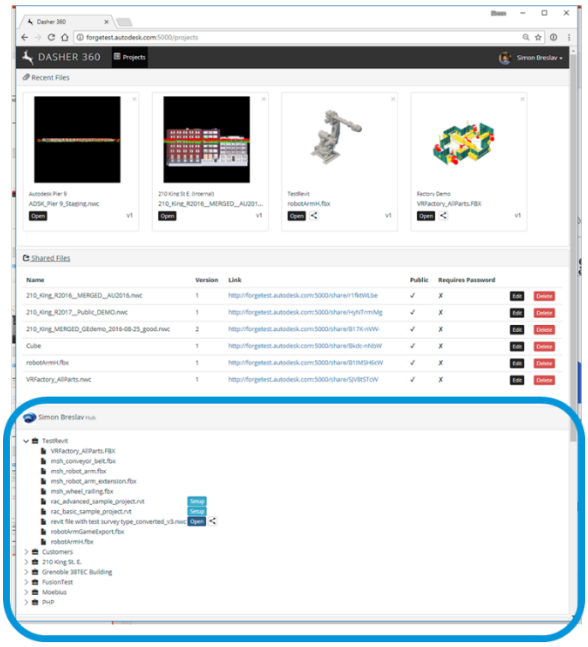
Authentication

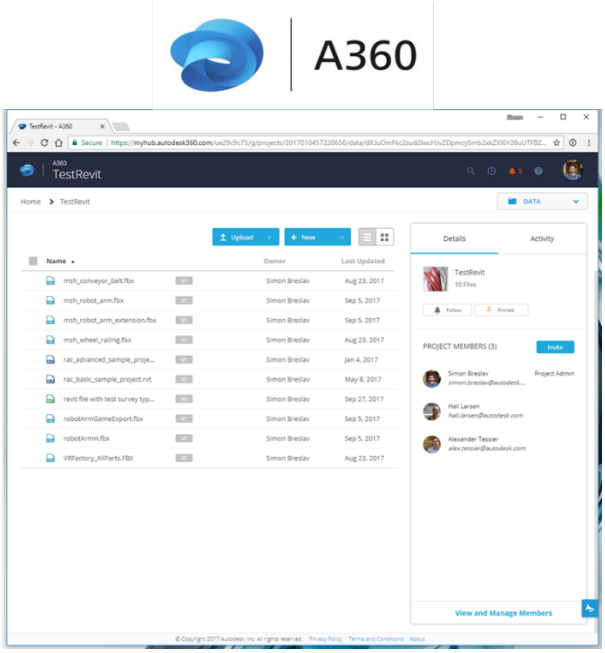
The first benefit that Dasher 360 gained from using Forge is that we don't have to worry about creating our own login mechanism. When a user clicks the login link, they are taken to accounts.autodesk.com to login into their Autodesk Account. Given the current status of the project (i.e. it is not open for just anyone to upload or manage project data), we did implement an email whitelist that lives in our local database, but all the authentication is done by Forge.



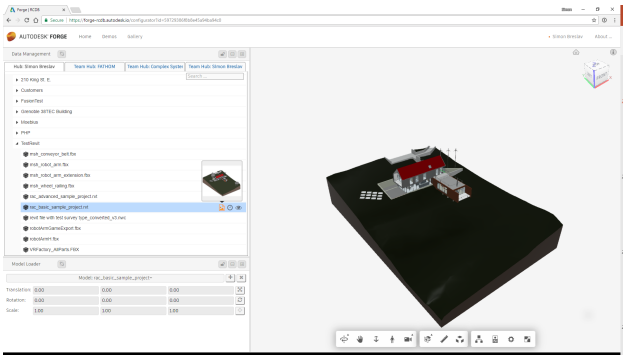
Data Management

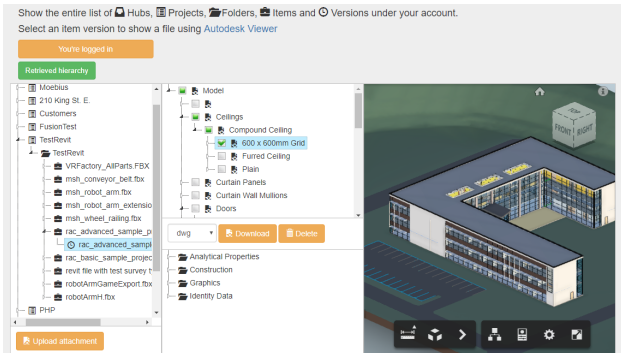
Using Autodesk authentication meant that we didn't need to re-implement the various group collaboration features available in A360 or BIM360 Team for Dasher 360's own project management workflows. When a user sign-ins, they can see all their Hubs, Projects, Folders, and Files.





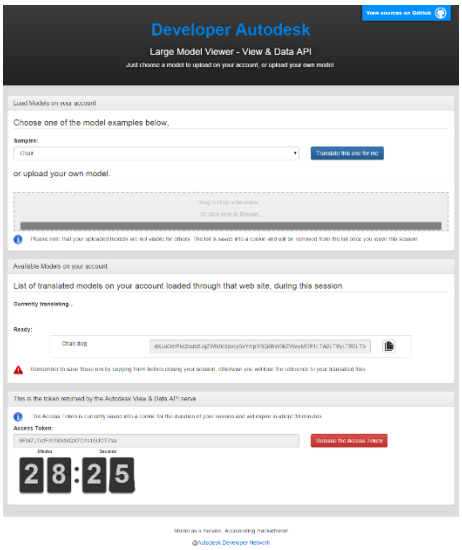
The Forge GitHub examples mentioned above show how this can be done.
So try out https://forge-rcdb.autodesk.io (Source: https://github.com/Autodesk-Forge/forge-rcdb.nodejs) and https://derivatives.autodesk.io (Source: https://github.com/Autodesk-Forge/model.derivative-nodejs-sample). The Derivative Demo is a bit simpler and focuses on the Authentication, Data Management and Model Derivative APIs, so we recommend looking at that first.





Model Derivative

Since we used the Data Management API to get access to an authenticated user's A360 account, we didn't have to create an interface to upload or convert model files. However, if you are interested in a workflow where your users don't require an A360 account, take a look at the code in this example: https://github.com/Autodesk-Forge/models.autodesk.io and a live demo of the example here: https://models.autodesk.io. Instead of connecting to A360 files, it shows how you can create App-managed file workflow.



Dasher 360 Forge Viewer Extensions

The bulk of the work that went into Dasher 360 was creating custom Viewer Extensions. We will now go through most of them, describing what went into making them, and provide links to blog posts and examples that will help you develop your own.

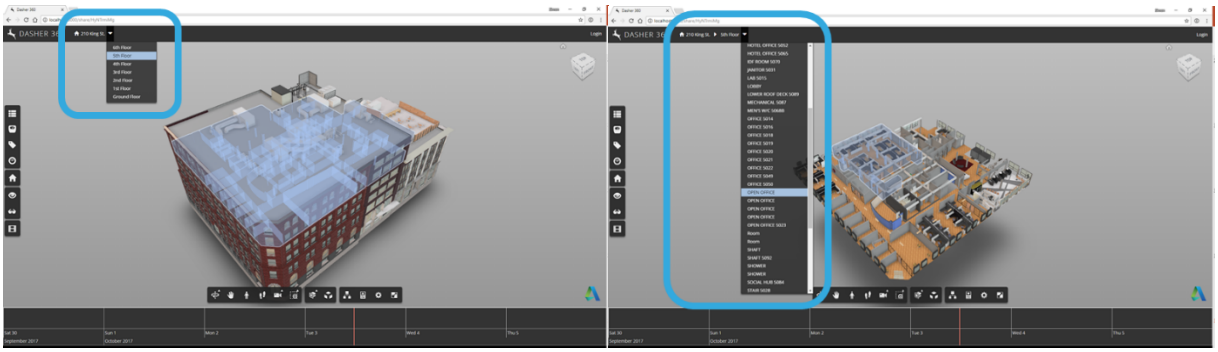
Toolbar

To have a clear visual separation of Dasher-specific tools, we created an extension to display a vertical toolbar similar to the default Viewer buttons. While we currently only have toggle buttons that turn on and off different functionality, it is also possible to have dropdown menus that expand vertically similarly to the ones in the default Viewer menu.

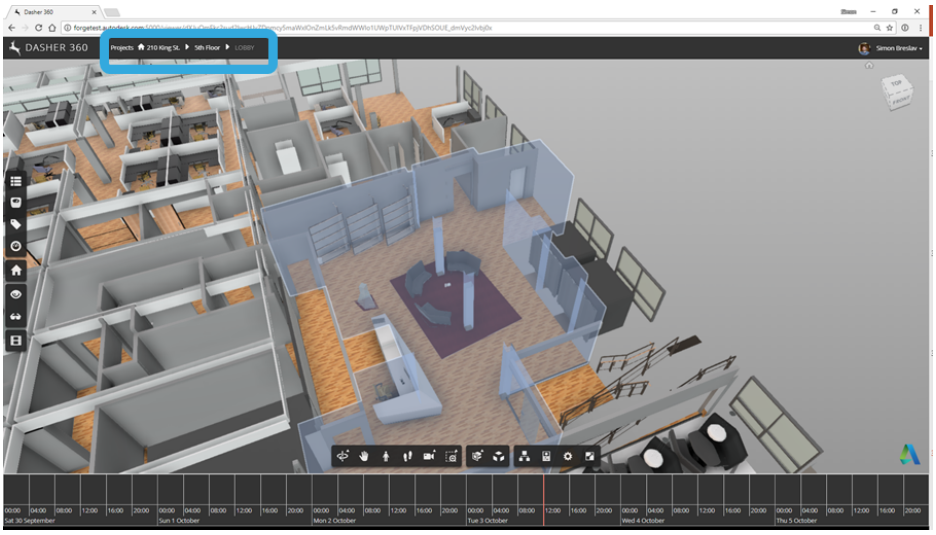
The implementation details are published on Kean's blog. Creating a vertical toolbar extension for the Autodesk viewer. May 04, 2016. Through the Interface. *Kean Walmsley*. http://through-the-interface.typepad.com/through_the_interface/2016/05/creating-a-vertical-toolbar-extension-for-the-autodesk-viewer.html

Building Navigation Breadcrumbs

To improve navigation for buildings, we implemented an extension that adds a breadcrumb navigation widget in the top navigation bar. As the user hovers over the dropdown with the list of floors, all the rooms that belong to that floor are highlighted in the model.



As the user navigates to floors and consequently to rooms, the breadcrumb provides visual feedback of the current location, as well as a shortcut to move upwards through the hierarchy. As the user continues to navigate through the building using standard Viewer tools, such as panning, orbit, and zoom, and First-Person navigation, we try to update breadcrumbs to be consistent with where the user is located. This is fairly straightforward in first-person mode: we simply listen to camera-changed events, and figure out which room the camera is located in. In third-person mode it can be a bit more ambiguous, as it's not always clear which room the user is looking at (we haven't perfected the heuristics for this yet).



Since this visualization method relies on having room geometry, only Navisworks files are currently supported, since the process of exporting Navisworks files from Revit adds these room geometries to the building. Currently if Revit file is imported into Forge viewer directly, room

geometries are not available. It is hoped that in the future this will change, but for now you may need to use the Navisworks NWC Export Utility to be able to export NWC files (it's free).

Floor and Room Hierarchy

While there is nothing particularly fancy about the widget itself – we are using a styled Bootstrap dropdown widget – the tricky part of the extension is figuring out a good way to highlight and isolate floors and rooms. To do this, we create model-specific settings that identify which floors exist in the building, and the match strings to be used when processing properties of the building. Here is an example of that settings data:

```
"levels": [
{ "match": "SIXTH FLOOR", "value": 6, "name": "6th Floor" },
{ "match": "FIFTH FLOOR", "value": 5, "name": "5th Floor" },
{ "match": ["LEVEL 4 FLR. FIN.", "FOURTH FLOOR"], "value": 4, "name": "4th Floor"
},
{ "match": ["LEVEL 3 FLR. FIN.", "THIRD FLOOR"], "value": 3, "name": "3rd Floor"
},
{ "match": "LEVEL 2 FLR. FIN.", "value": 2, "name": "2nd Floor" },
{ "match": "LEVEL 1 FLR. FIN.", "value": 1, "name": "1st Floor" },
```

To see actual examples of processing properties, take a look at the **Meta Properties** demo on https://forge-rcdb.autodesk.io

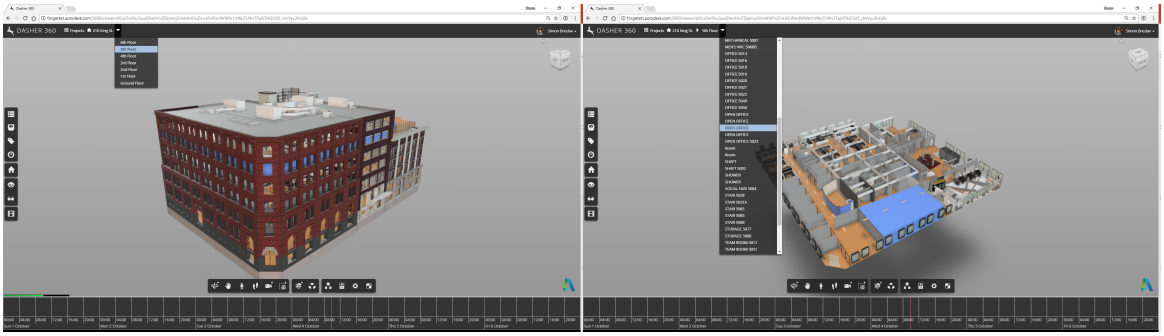
This method is not perfect, often there are objects that span multiple floors, such as the building envelope, and while we have experimented with using Revit's Parts functionality, it's still a bit of an open problem for us. If you are interested in tackling this issue, take a look at the **Wall Analyzer** example in https://forge-rcdb.autodesk.io. This example looks very promising,

performing geometry analysis. This has the potential to save quite a bit of manual work, and we will explore this method in the future.

Selection Highlight

The last noteworthy piece of functionality relating to Floor & Room Navigation is the highlighting of floors and rooms. Initially we simply implemented this functionality by using default viewer selection mechanism. Meaning these two Viewer API functions:

viewer.select(dbId)
viewer.clearSelection()



However, changing selection forces the whole building to redraw, causing quite a bit of flickering, plus – when looking at the whole building – the room geometries are inside and can be hard to see if (for example) the windows are loaded.

In order to overcome this, we recreated and adapted some private/internal functionality that drives the low-level selection mechanism. Our custom selection geometry now ignores the depth buffer and just draws on top, while the redraw only invalidates Overlays, not the geometry. A snippet of the core code is below, just keep in mind that we are using an impl object of the viewer, which is intended for internal use, since it is undocumented and subject to change, please only use it for experimentation.

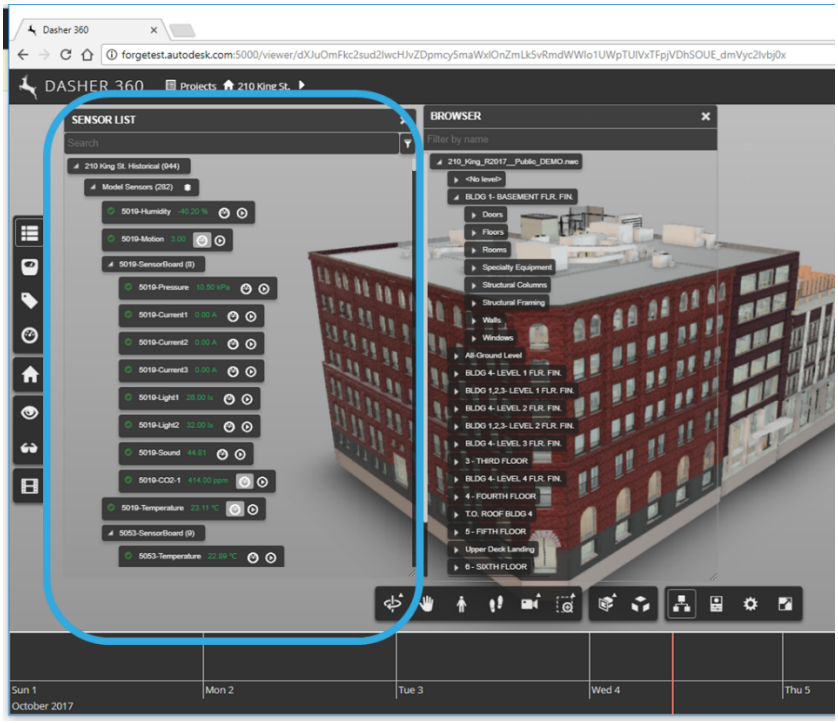
```
let material = new THREE.MeshBasicMaterial(
```

```
{color: 0x6e98ee,
 transparent: true,
 opacity: 0.5 });
material.depthWrite = false;
material.depthTest = false;
viewer.impl.createOverlayScene('CustomSelection');


let renderProxy = this.viewer.impl.getRenderProxy(viewer.model, fragId);
let meshProxy = new THREE.Mesh(renderProxy.geometry, material);
meshProxy.matrix.copy(renderProxy.matrixWorld);

viewer.impl.addOverlay('CustomSelection', meshProxy);
viewer.impl.invalidate(false, false, true);
```

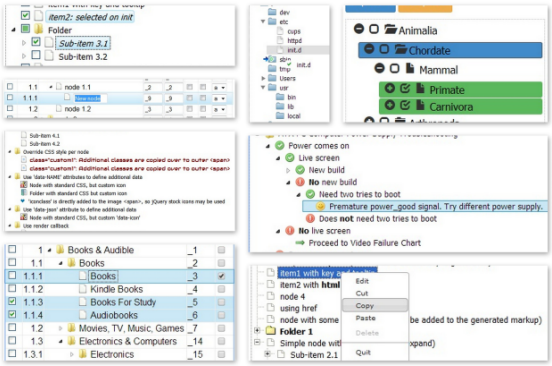
Sensor List

Dasher 360's Sensor List contains a list of all the sensors in the model and in the database, and provides a set of core sensor management features:

- Filter the sensors based on type or a name.
- Hide sensors that are not connected to the database.
- "Go to Sensor" buttons that take you to a sensor's 3D location.
- "Add to Dashboard" buttons that add the sensor to a Bookmark Dashboard.
- Show the latest value of a sensor.
- "Positioning Add/Remove" buttons that let you position sensors that don't have a position, or remove positioning information for sensors that do.



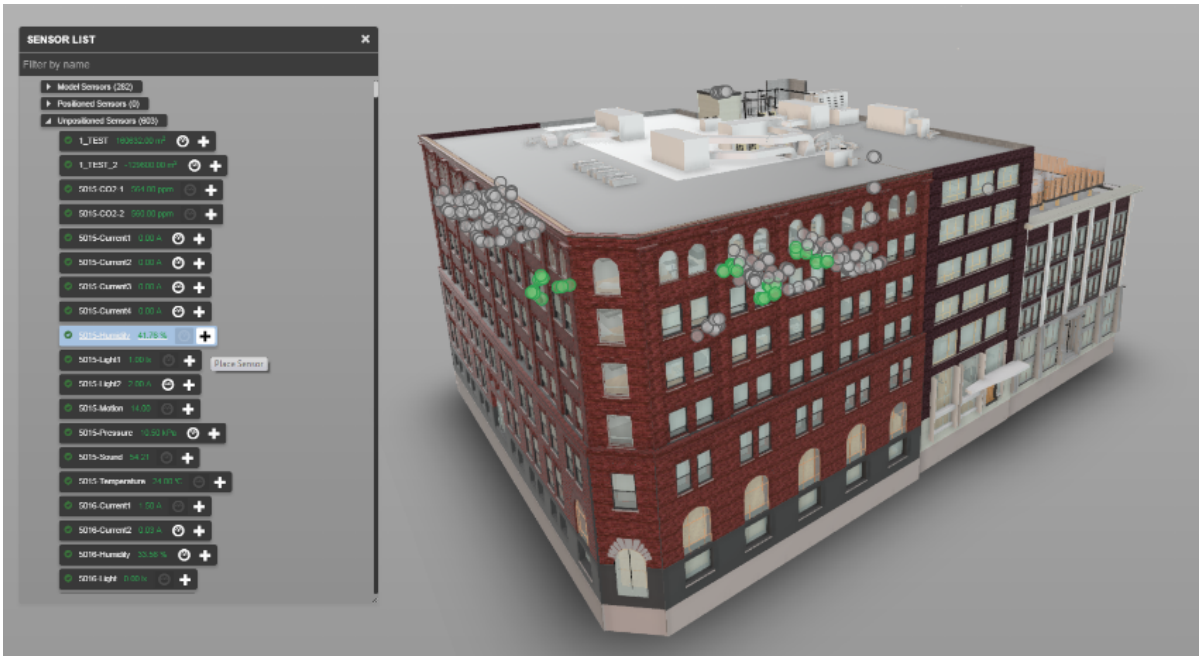
To implement the list we derived from the tree class in the Viewer code. Autodesk.Viewing.UI.Tree for the tree, and Autodesk.Viewing.UI.TreeDelegate. However this was mainly to provide a consistent look and feel: there's no reason why a different library such as fancytree (https://github.com/mar10/fancytree) couldn't be used. A Viewer Panel can contain arbitrary HTML code, so you can insert any content into your panel.



Sensors that exist in the database but are not in the Revit model can be positioned from within the Dasher 360 user interface. The key to implementing such a feature is to be able to map a 3D location from mouse click. The function we are using is

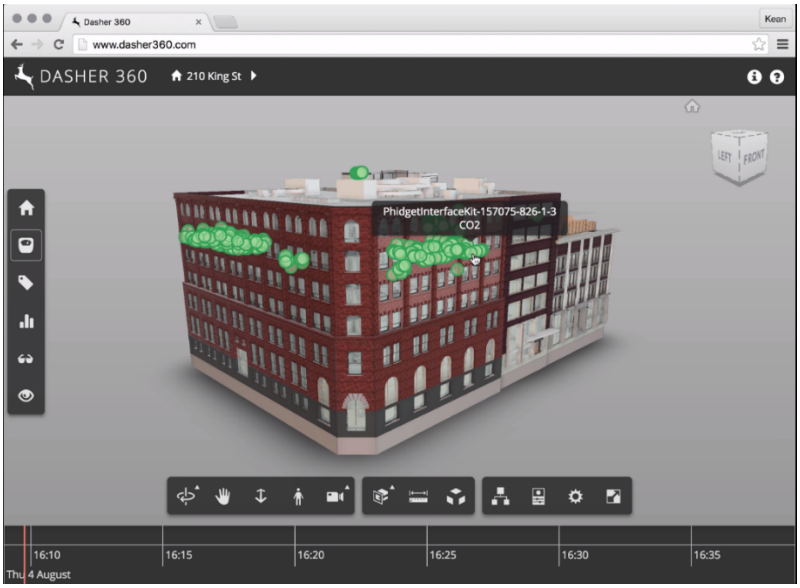
```
viewer. clientToWorld (event.canvasX, event.canvasY, false);
```

Note, the last parameter of the function is *ignoreTransparent*: a Boolean to control if transparent objects should be ignored by the hit test. The result of the clientToWorld will give you an object with a 3D location as well as the ID of the geometry that was intersected.

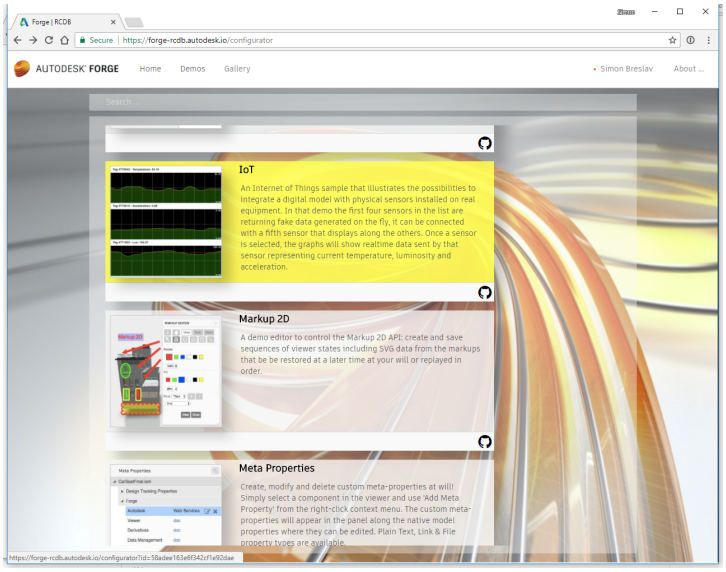


Sensor Dots

The core value added by Dasher 360 over a standard Forge Viewer is the ability to display sensor data in a 3D context, and the main method to visualize placement of these sensors is using small icons that can be hovered over to display some basic information, or clicked to open a detailed graph.

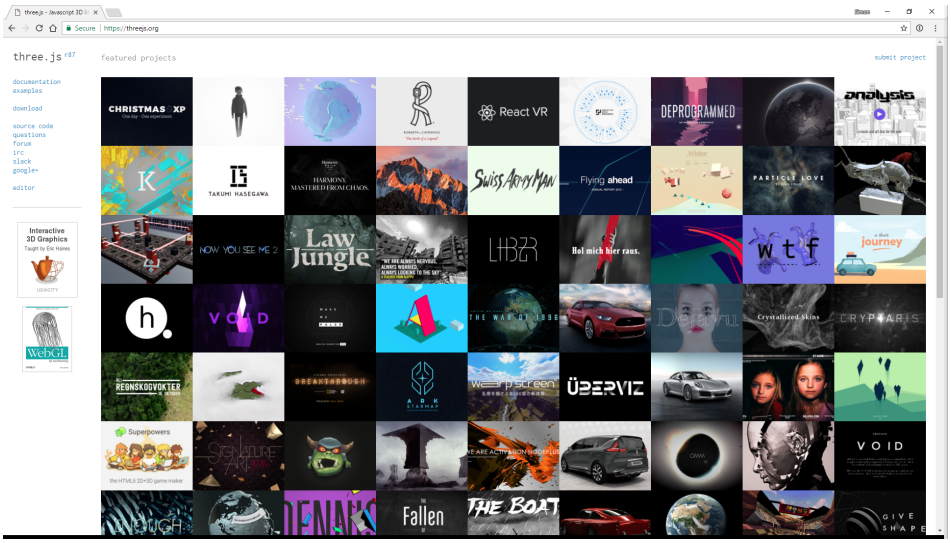


In the initial implementation, we placed the dots as SVG elements in the HTML page above the viewer, and on any camera movement these elements were updated to new locations. To see implementation details of such an approach, see the **IoT** Example in the https://forge-rcdb.autodesk.io/ project.

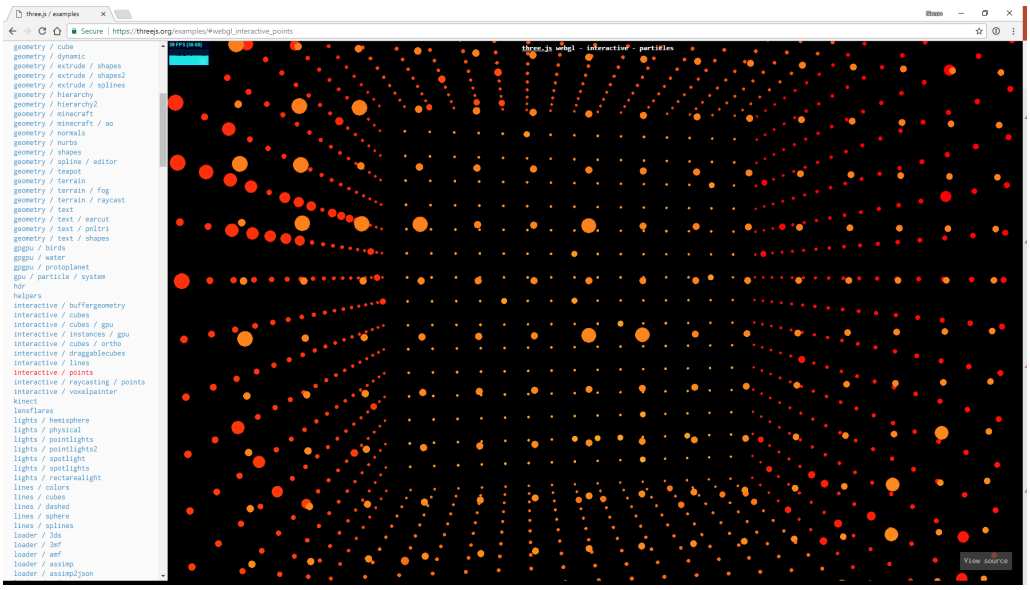


While this approach makes it easy to make the dots interactive (hovering, clicking), if you have a lot of sensors (thousands), it will become computationally expensive when (for instance) rotating the model. While one workaround might be to hide sensors while the camera is changing (and only to draw sensors when the model is static), it was not the user experience we were looking for. We therefore decided to explore a different way to draw sensor dots that would let us scale to thousands of sensors.

The great thing about the Forge Viewer is that it's built on top of Three.js (https://threejs.org/), an active open source library with a large community, good documentation, and lots of examples.



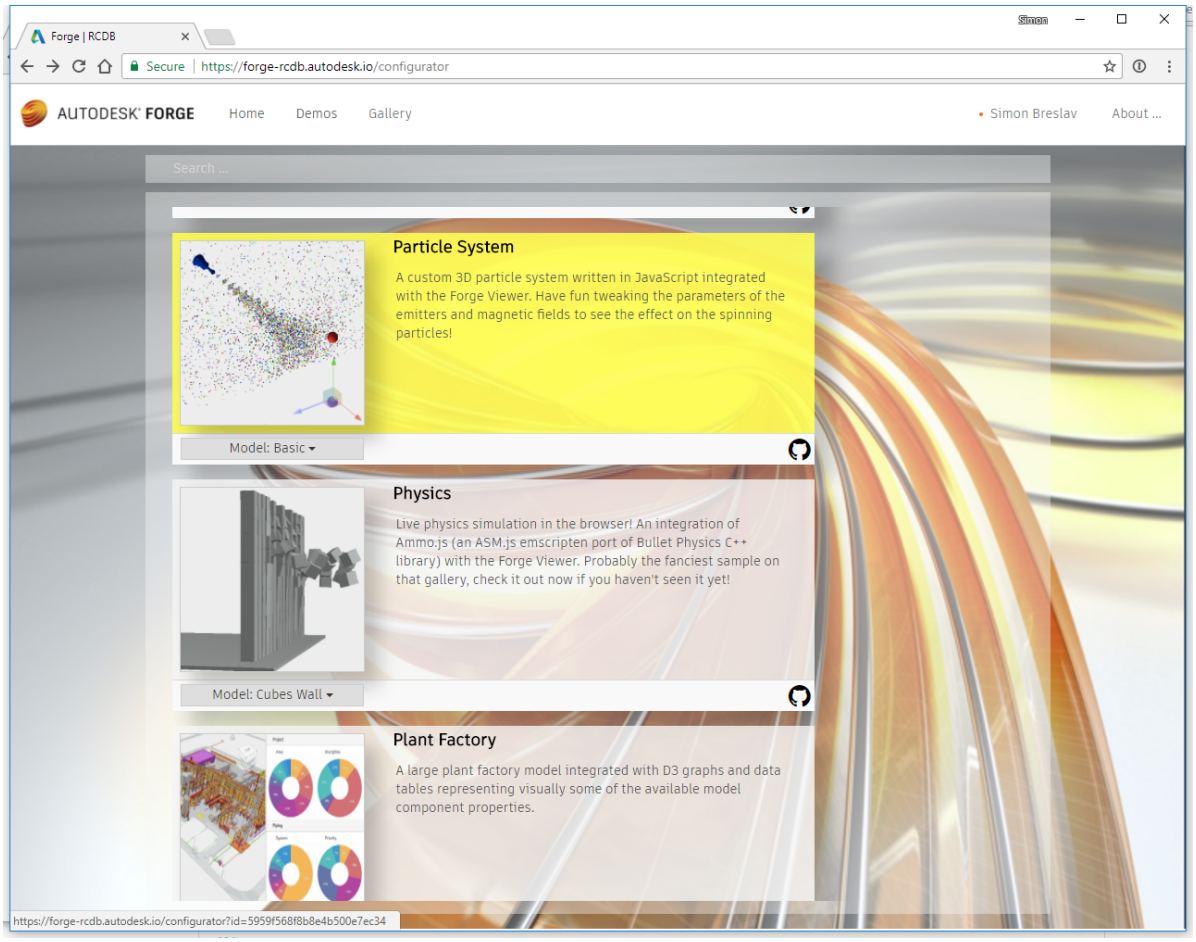
When looking at the examples, and getting some advice from Forge experts (many thanks to Philippe Leefsma), we've found an example that demonstrated exactly what we were looking for, interactive points using a point cloud (https://threejs.org/examples/#webgl_interactive_points).



Of course, in terms of getting the example translated to the way the Viewer does things, it's important to keep in mind that the Viewer might be using an older version of Three.js. The version of Three.js is displayed in the Debug Console (press F11 in Chrome while on a page that has the Viewer.) At the time of writing this, our version of the Viewer was using v71, so a way to get corresponding examples is to get threejs release from GitHub of that version, and run the examples locally.
https://github.com/mrdoob/three.js/releases/tag/r71

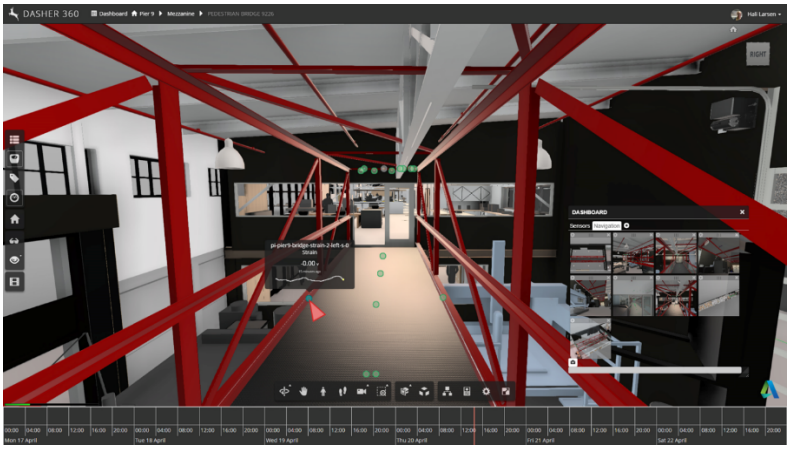
The corresponding documentation will also be in the release folder.

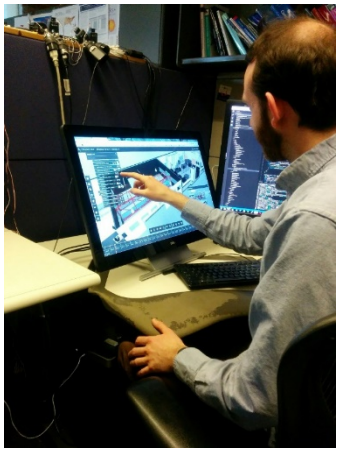
However, luckily for all of us, the Forge team has a made an example of using particle systems in the Viewer, so just take a look at the **Particle System** example in the https://forge-rcdb.autodesk.io/ project to figure out how to get it working.



To get a bit more context, here is Kean's blog on the topic: [Optimizing Dasher 360: using a point cloud to display sensor markers.](#) January 23, 2017. Through the Interface. Kean Walmsley. http://through-the-interface.typepad.com/through_the_interface/2017/01/optimizing-dasher-360-using-a-point-cloud-to-display-sensor-markers.html

Tooltips





When you hover over the sensor dots, we show some basic information about the sensor, and the latest value, and sparkline of the data. To get the tooltip dialog we used a library called Tooltipster (http://iamceege.github.io/tooltipster/).

The tricky part about tooltips is that on mobile and touch devices, there are no mouse hover events. So, we added an alternative behavior, where first click would open the tooltip, and second would open the plot. Take a look at Kean's blog here:

Handling both mouse and touch events in Forge viewer applications. April 13, 2017. Through the Interface. *Kean Walmsley.* http://through-the-interface.typepad.com/through_the_interface/2017/04/handling-both-mouse-and-touch-events-in-forge-viewer-applications.html

Note, that we have tweaked this code a bit, to better support situations where there is both mouse and touch (e.g. touch screen on a desktop computer). Mainly, an additional event listener was added for *touchstart* event on the canvas element of the Viewer. This way, touch behavior can more easily be isolated from the mouse workflow.

```
viewer.canvas.addEventListener('touchstart', callbackFunction, false)
```

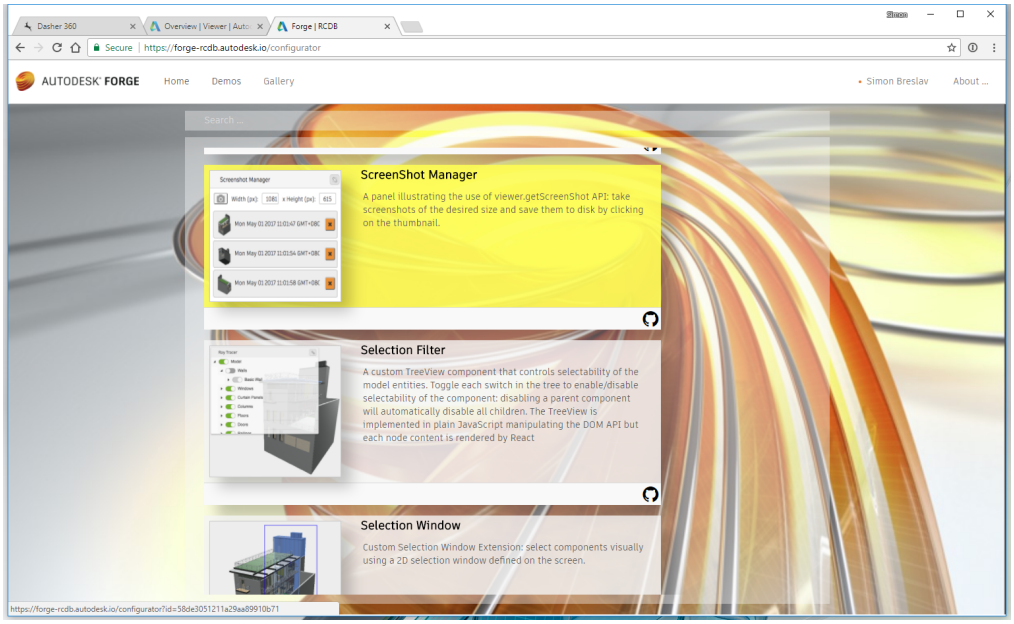
Sensor & View Bookmark Dashboard

The sensor and view bookmark dashboard lets us create sensor and navigational dashboards. We used a library called gridstackjs (http://gridstackjs.com/) for a draggable and resizable multi-column grid.

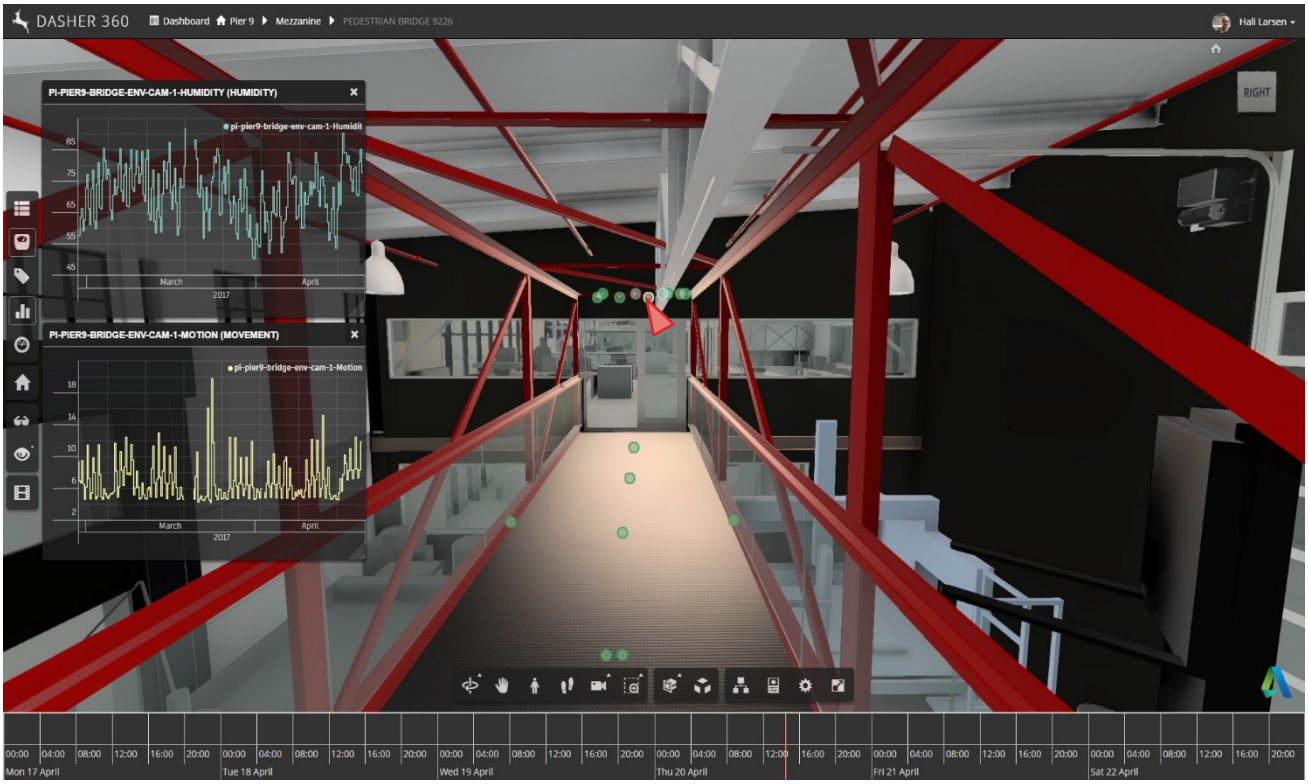
To implement navigational bookmarks, the viewer has all the needed information, mainly, current state of the viewer (camera data and etc.) and a generated screenshot of the current view. Take a look at the documentation, here are the functions to look for.

```
viewer.getScreenShotBuffer(width, height, callback)
viewer.getState(filter)
viewer.restoreState(viewerState, filter, immediate)
```

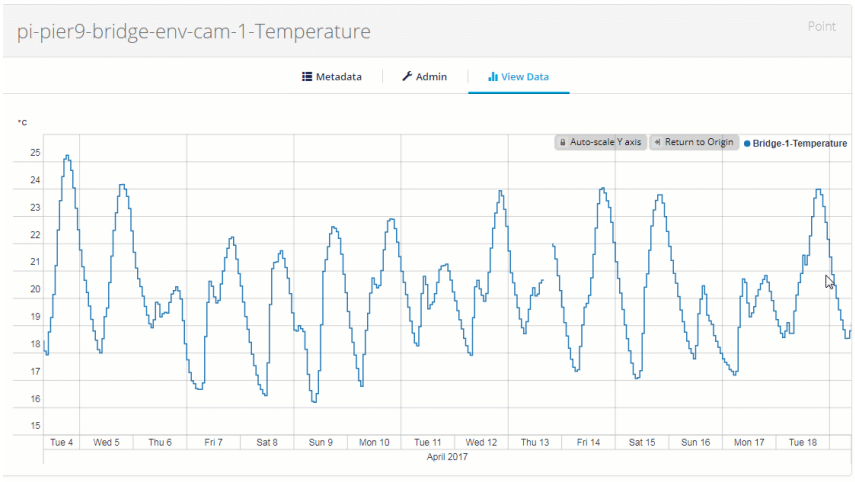
To get a live demo and source code of screenshot functionality, refer to the **ScreenShot Manager** example in https://forge-rcdb.autodesk.io/ project.



Time Series Plots (Splash)

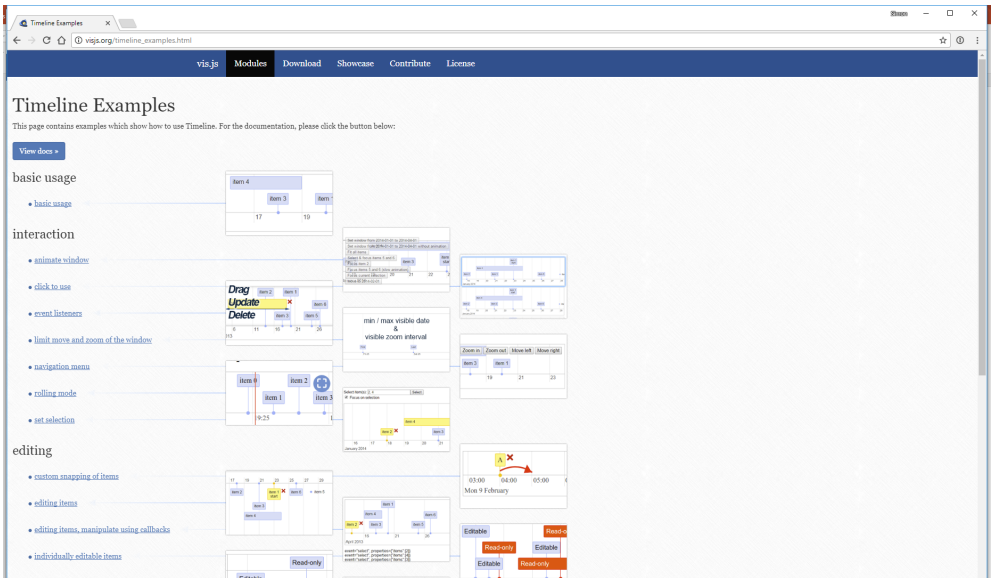


To display detailed sensor time-series data, we use line plots. To get up and running, we used a library called Vis.js (http://visjs.org/), which worked well, and we highly recommend it. However, we wanted to have the ability to quickly display data at various levels of detail. We therefore integrated a research project that our group developed called Splash (https://autodeskresearch.com/publications/splash). This library initially loads coarse data in place of fine data, to improve the user experience of navigating through large datasets quickly.



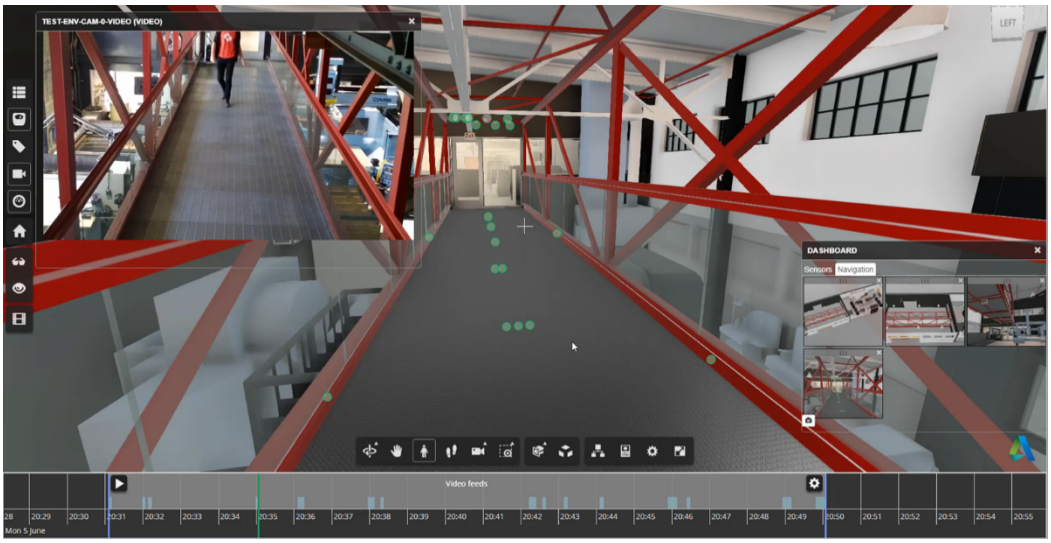
Timeline

In Dasher, our focus has been on time-dependent (sensor) data, so we needed a global timeline to put everything into a temporal context. To get started we used the Vis.js (http://visjs.org/) Timeline widget. Which we still use, as it's API is quite flexible.



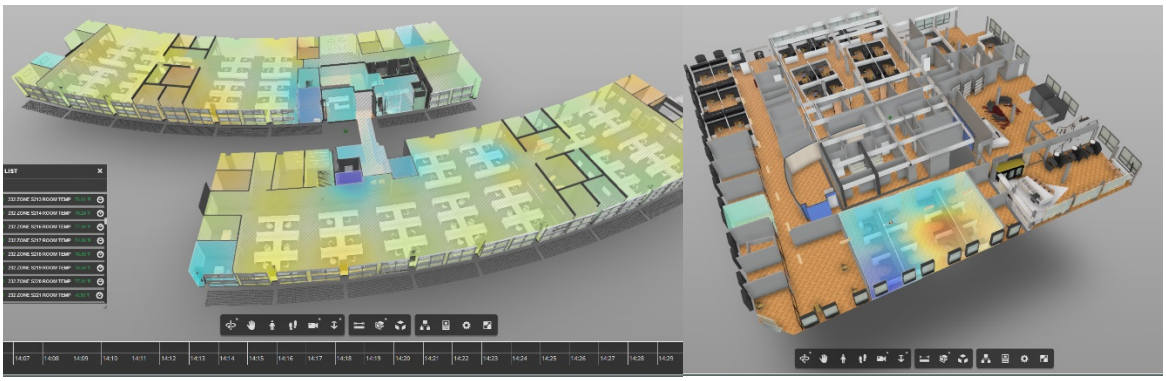
Video

That said, we did start to see performance issues with the timeline when we started looking at displaying video data instead of basic sensor data. Since video surveillance data can be quite sparse, we wanted to add items into the global timeline indicating where data is available. However, with a long period loading all the items became too slow, since all the items are implemented as HTML elements. In the future we will probably implement a custom item to show where data is available, and where it is not.

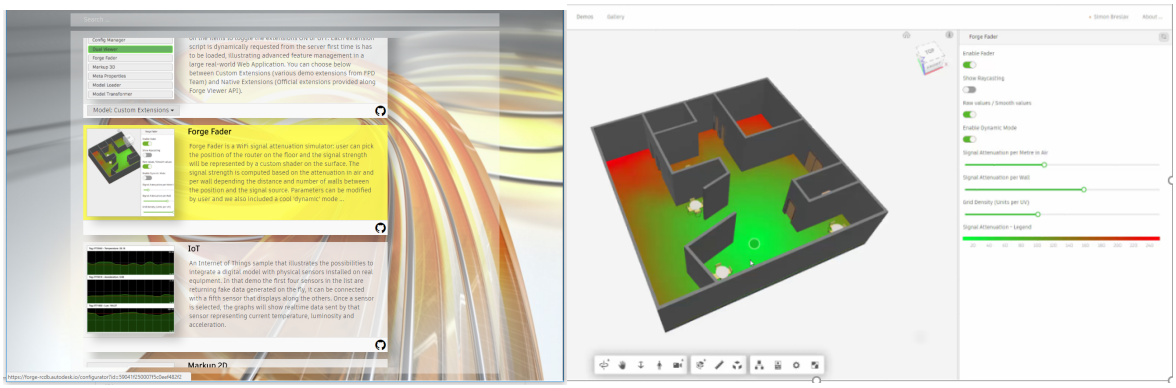


Surface Shading/Heatmap

Surface shading (also known as a heatmap) is a very useful way to visualize data from a large set of sensors, since at a glance you're able to see patterns and anomalies with enough spatial context to infer a possible environmental reason for the effect.

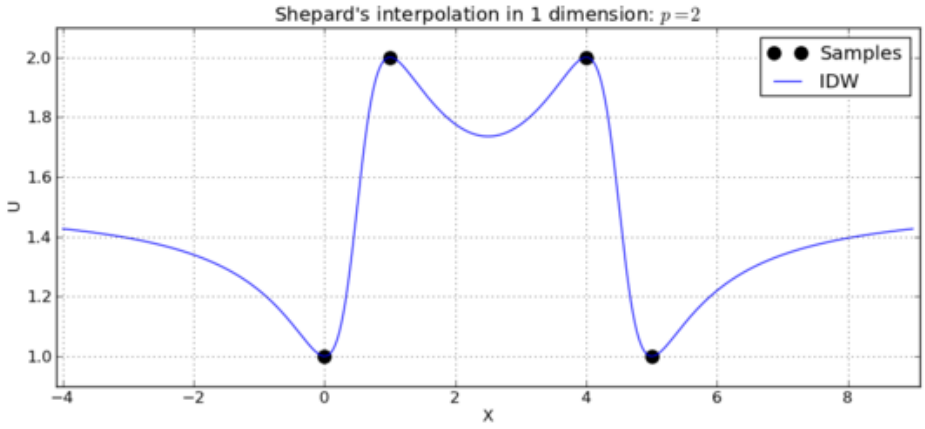


To get up and running with Surface Shading, you should check out the **Forge Fader** example in the https://forge-rcdb.autodesk.io/ project. It shows an example of WiFi signal attenuation. A user picks a position of the router on the floor and the signal strength is represented by a custom shader on the surface.



Under the hood it does raycasting at a specified grid resolution, and interpolates values on the GPU. This example should be helpful to get you started, however, with Dasher 360 we took a slightly different approach.

Before talking about what we did, let's first clarify what problem we are trying to solve. Given a sparse set of sensors positioned in different spatial locations, we want to interpolate these values to everywhere we have some surface that we can use to communicate that sparse sensor data. To illustrate this, the 1D plot below shows four black samples (e.g. sensors), and a curve that connects the lines, interpolates. Of course there are an infinite number of different curves that can go through those points, and the only true way to know what values are in between is to add more samples (add more sensors). However, we can reasonably approximate values for visualization purposes.



While, there are lots of different methods to do such interpolation, take a look at the Wikipedia article on Multivariate Interpolation (https://en.wikipedia.org/wiki/Multivariate_interpolation) if the topic interests you.

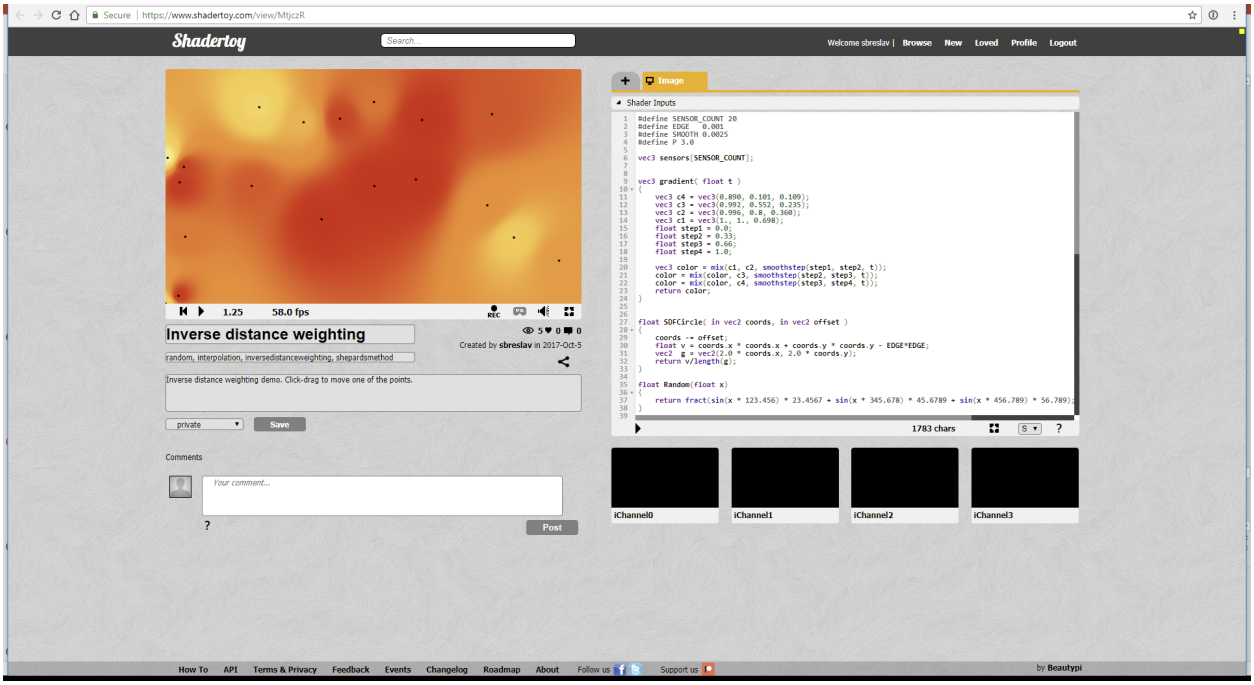
We decided to use the simplest, called Inverse Distance Weighting (https://en.wikipedia.org/wiki/Inverse_distance_weighting) This method is also called Shepard's Method, since it was introduced by Donald Shepard in 1968 in a paper titled "A two-dimensional interpolation function for irregularly-spaced data". The main idea of the method is that values for

unknown points are calculated with a weighted average of the values available at the known points.

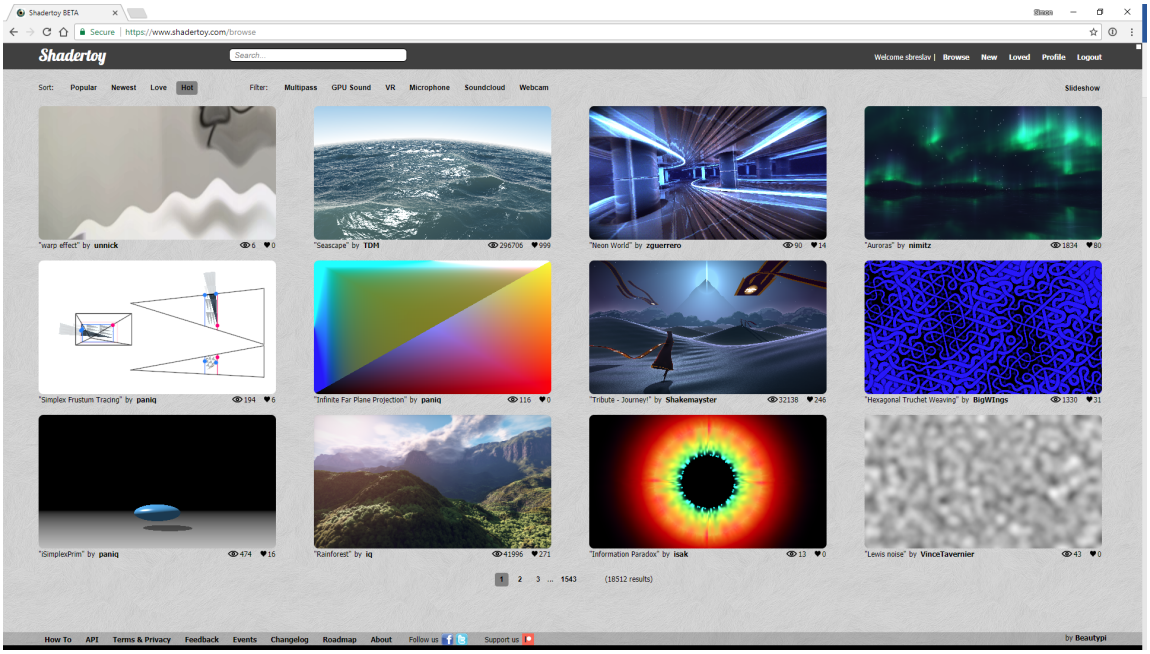
One nice aspect of this method that it can easily be performed fully by the GPU, which allows it to be executed in a highly efficient manner. Here is our 14 line implementation in GLSL.

```
…
float IDW(vec2 fragPos) {
  float sum, wsum = 0.0;
  for (int i=0; i < SENSOR_COUNT; i++) {
    float dist = distance(fragPos, sensors[i].xy);
    if( dist > 0.0) {
      float w = (1.0 / pow(dist, P));
      sum += (sensors[i].z * w);
      wsum += w;
    } else {
      return sensors[i].z;
    }
  }
  return sum / wsum;
}
…
```

Note, this is a 2D version, but it doesn't change much for the 3D case. To see the shader in action, take a look at this ShaderToy Demo we made: https://www.shadertoy.com/view/MtjczR



If you haven't heard of ShaderToy, it's a cool website to learn more about making GPU Shaders in the browser. People share lots of really cool demos, and all the source code is included with each demo. Some demos are just amazing.



Kiosk Mode

- Kiosk Mode is designed for cases where you want to leave Dasher 360 in "demo mode", for instance on a large screen in your building lobby.
- Displays a fake cursor to simulate the operation of Dasher 360.
- The mode loops through various Dasher 360 features and is interruptible by the mouse.
- Functional for any screen configuration/resolution.
- Easy to modify/extend to add new features.
- View-independent.
- Not tied to a specific model.

For implementation details take a look at these blog posts:

Implementing a demo mode inside the Forge viewer – Part 1. March 15, 2017. Through the Interface. Kean Walmsley. http://through-the-interface.typepad.com/through_the_interface/2017/03/implementing-a-demo-mode-inside-the-forge-viewer-part-1.html
Implementing a demo mode inside the Forge viewer – Part 2. March 16, 2017. Through the Interface. Kean Walmsley. http://through-the-interface.typepad.com/through_the_interface/2017/03/implementing-a-demo-mode-inside-the-forge-viewer-part-2.html

Implementing a demo mode inside the Forge viewer – Part 3. March 17, 2017. Through the Interface. Kean Walmsley. http://through-the-interface.typepad.com/through_the_interface/2017/03/implementing-a-demo-mode-inside-the-forge-viewer-part-3.html

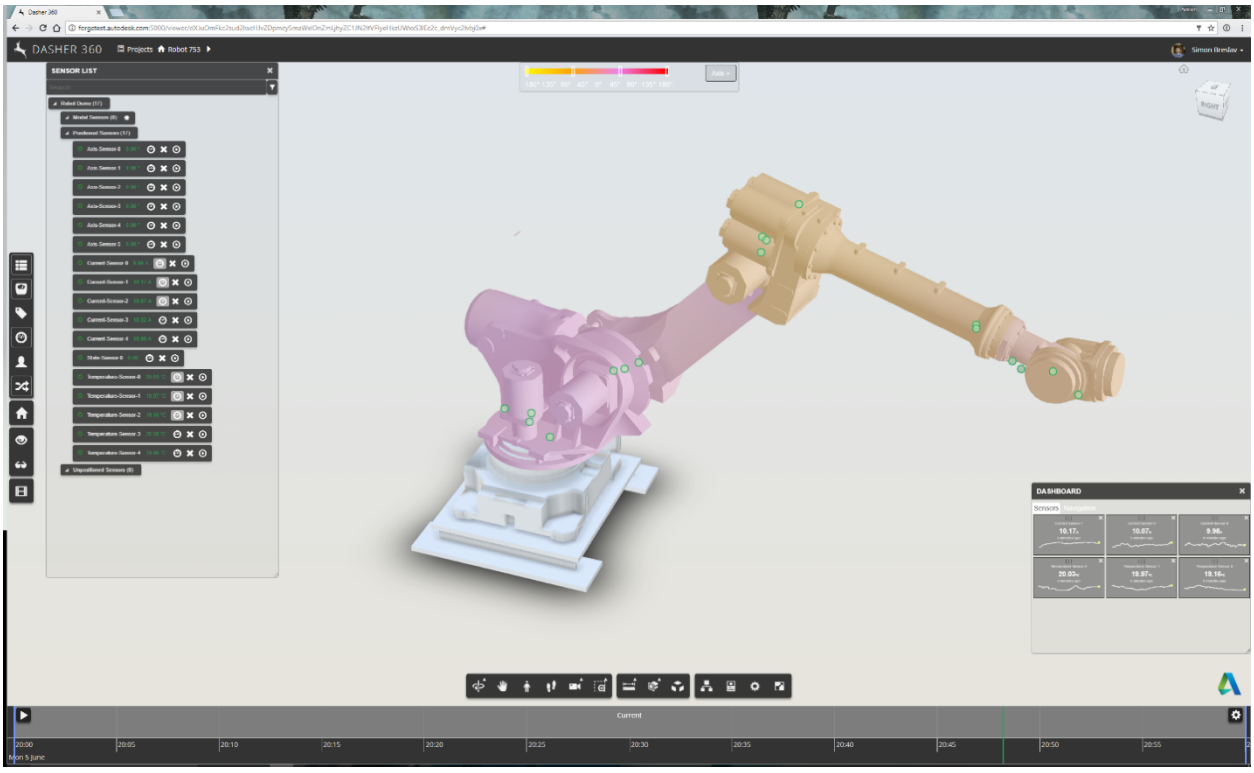
Logo

A small but useful extension provides custom branding by overlaying a logo bitmap on the Forge Viewer. See Kean's blog for implementation details:

Overlaying a logo on the Forge viewer. February 17, 2017 Through the Interface. Kean Walmsley.http://through-the-interface.typepad.com/through_the_interface/2017/02/overlaying-a-logo-on-the-forge-viewer.html

Work In Progress

Animated Robot Demo

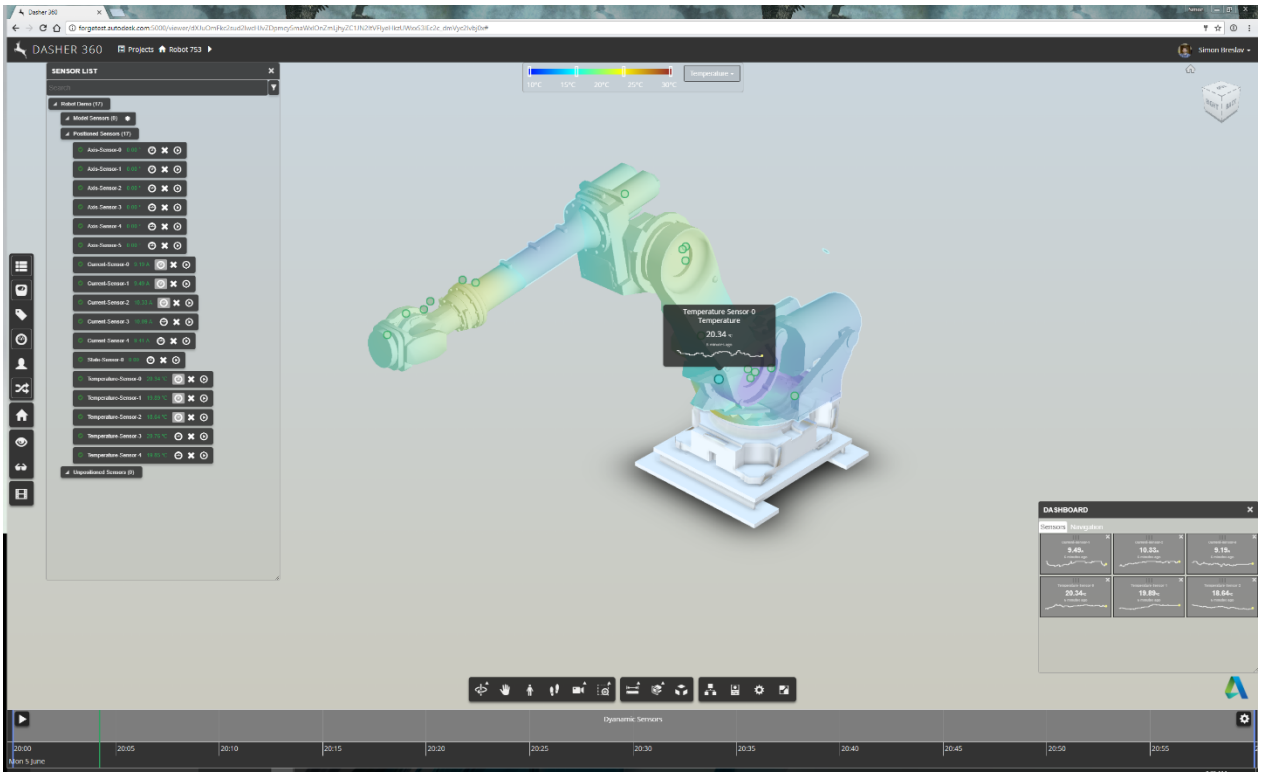


The goal of the demo was to give a glimpse of Factory of the Future, where Forge Viewer would be part of a toolchain for robot programmers and operators to debug and monitor robot operations. To show the feasibility of such an application, we created a small demo extension with generous help from Philippe Leefsma. Since the robot arm model didn't have an object hierarchy to do proper transformations, we had to manually create one in our code. Here are the steps we took:

- Manually figure out 3D locations and IDs of all the parts: use `viewer.clientToWorld (event.canvasX, event.canvasY, false)` function to figure out IDs of all the parts of the robot that need to move, and 3D locations where the pivot locations are. To get axis of rotation, we used the normal of the mesh at the intersection point when getting the location of the pivot.
- Create a hierarchy of pivots located at position figured out in last step, with each pivot being a THREE.Object3D(), which contains all the child pivots, as well as a copy of the robot part that is connected to that pivot.
- Copies of the robot arm parts can be created the same way as it was done for Custom Selection method in Floor and Room Hierarchy section, it used `viewer.impl.getRenderProxy(viewer.model, fragId)` method to get a copy of the geometry. The original geometry should be hidden.
- A root pivot can be added as an overlay (`viewer.impl.addOverlay('Robot', root_pivot)`) or to an actual main scene (`viewer.impl.scene.add(root_pivot)`). While in some cases, such as custom selection, and surface shading using overlays was useful, in this case, it doesn't offer much advantage.
- Once the hierarchy is created, rotating pivot objects, rotates all the child geometry and pivots. Here is snippet of code, where axis is THREE.Vector3 around which we are rotation, and angle is just an angle in radians.

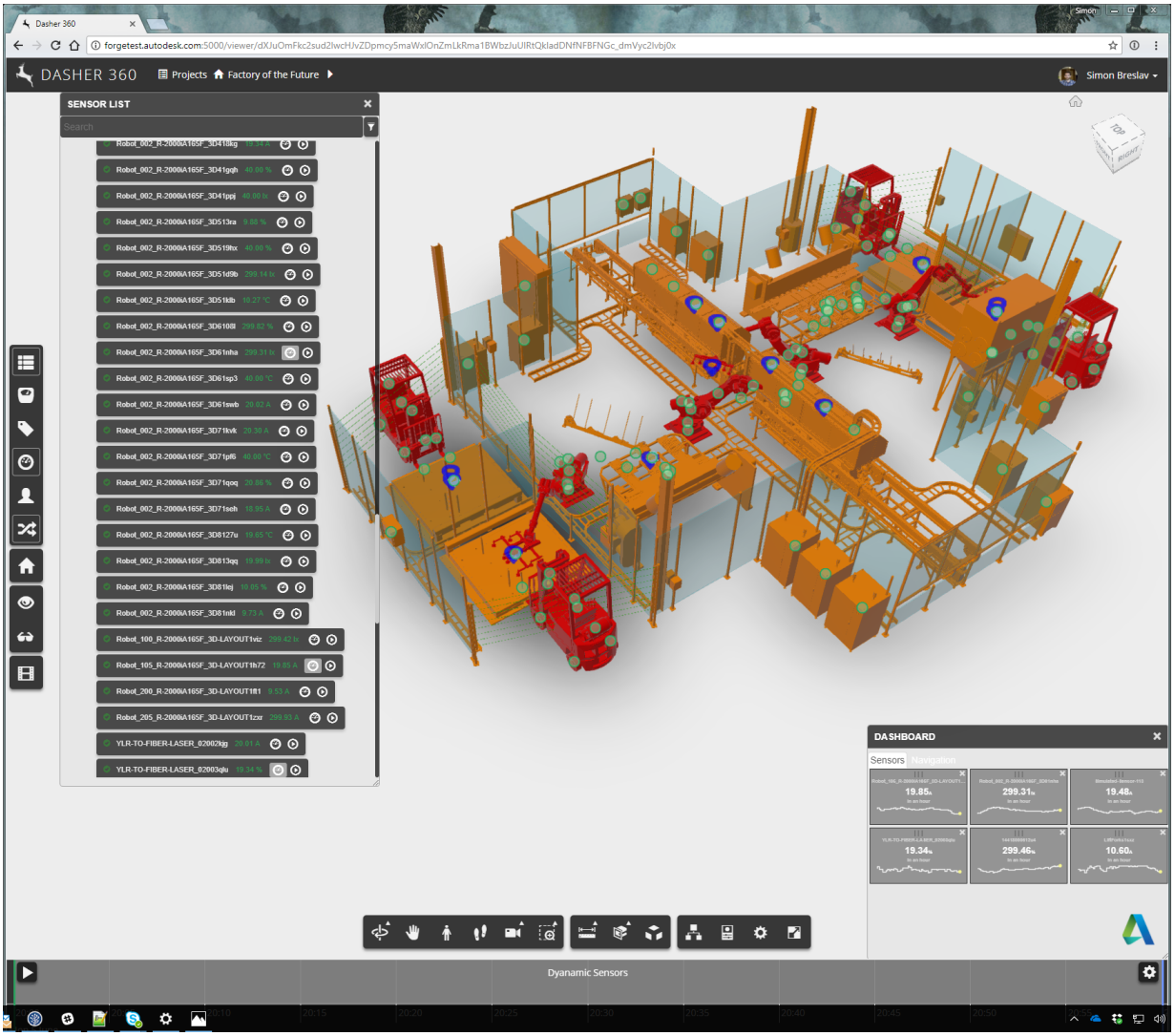
```
let quaternion = new THREE.Quaternion();
quaternion.setFromAxisAngle(axis, angle);
pivot.rotation.setFromQuaternion(quaternion);
```

While this only gives the bare-bone approach we used, it should be enough to get you started.



Factory Floor Demo

While animating a single robot is useful, putting that robot in the context of a whole factory significantly increases the usefulness of the demo for our Factory of the Future. Such visualizations will be crucial both for simulation of possible factory designs and for operation of existing factories



Conclusion

In this class, you've learned the history of Project Dasher and how we benefited from using Forge when reimplementing it for the web. Not only did Forge significantly speed up development, but also made it much easier given the quality of the examples and the library documentation we used for inspiration. By using Viewer Extensions as basis of our framework, it was possible for us mix and match multiple pieces of well encapsulated functionality into one coherent experience.