



Applying NASA code guidelines to JavaScript

Airspace is closer than you may think

Europa's surface

@PixelsCommander


denis.radin@gmail.com



MyHammer

The universe respects unification

Planets shape, the laws of physics,
and star system as a unit
are constant everywhere



“Unification — effective method to decrease diversity. Aim for unification is to place elements in particular order building strict system which is comfortable to use.”

(c) Wikipedia



**There are sixtillions of
standardized units...**

It can be hard to distinguish them
without being an expert

Milky Way, Salamnca, Spain



The same works for engineering

Unification never stops

**Diversity decreases
since only best
solutions survive...**

Is it a Boeing or Airbus?



A Germanwings Airbus A320neo aircraft is shown on a runway. The aircraft is white with maroon and yellow stripes on the tail. The text "Germanwings" is written in maroon on the side of the fuselage, and "Lufthansa Group" is written in smaller letters below it. The registration "D-APX" is visible on the tail. The aircraft is parked on a runway with a grassy field and hills in the background.

Can you distinguish
them at all?

I bet you can not because of unification...



We develop software for 60 years

But such a common thing as UI definition
is not standardized yet

Souyz space ship docks ISS





All planets
are round

All UIs will be unified

Timelapse taken from ISS



Front Side Imagery



Data in Record Grooves

Standard?

Let`s guess...

Voyager golden disks

The background of the slide features three fighter jets flying against a clear blue sky. At the top is a P-51 Mustang, a single-engine propeller-driven fighter, with yellow and blue markings and the tail number 'FU-834'. In the center is an F-22 Raptor, a stealth fighter, shown from a side-on perspective. At the bottom right is an F-16 Fighting Falcon, a multirole fighter, with '50' and '1st Wild Weather' on its tail. The text 'Aviation uses UIs' is overlaid in a large, white, serif font across the middle of the image.

Aviation uses UIs

Which tend to be standardized
as whole industry is

F-22, F-16, P-51

Why HTML for flight instruments?

Unification, reliability, accessibility

● Bombardier Q400 electronic dashboard

An MQ-1 Predator drone is shown in flight against a dramatic sky at sunset or sunrise. The drone is silhouetted against the bright horizon, with its wings and V-shaped tail clearly visible. The sky is filled with dark, textured clouds, and the sun's glow creates a strong backlighting effect.

Network accessibility

Highly valuable in the age of drones

MQ-1 Predator

A photograph of three technicians in white cleanroom suits working on a large, circular, black spacecraft component with white markings. The component is mounted on a stand. The technicians are in a cleanroom environment with various equipment and cables visible in the background.

Unification

Decreases development cost standardizing
development flow and technologies stack

Work on NASA's InSight Lander

A wide-angle photograph of NASA's Mission Control Center. The room is filled with long rows of consoles, each equipped with multiple computer monitors. Several staff members are seated at the consoles, working. The walls are decorated with numerous NASA mission patches and large posters of space missions. In the background, large projection screens display various data, including a map of Earth and a close-up of a spacecraft. The overall atmosphere is professional and high-tech.

Reliability

Browser is a GUI rendering system tested by billions users daily

NASA's Mission Control Center

Components market

Possibility to establish competitive UI components (flight instruments) market

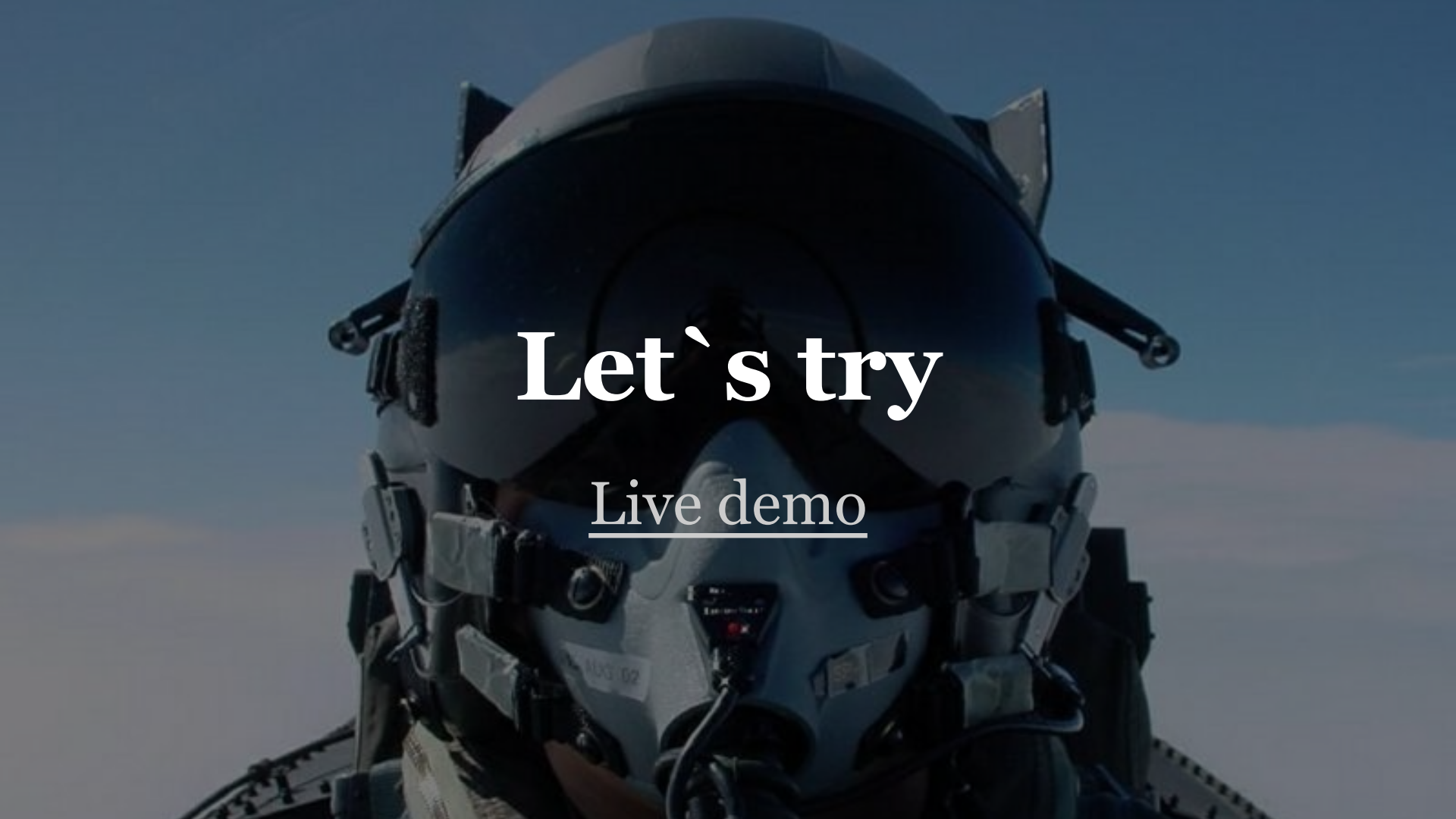
Mir station modules scheme

A Diamond aircraft DA40 is shown in flight against a sunset sky. The aircraft is a high-wing, four-seater light aircraft with a tricycle landing gear. It is flying from the left towards the right, with its wings and tail visible. The sky is a mix of blue and orange, with some clouds at the bottom.

First HTML/JS flight instrument

And first ever flight using
HTML/JS for displaying flight information

Diamond aircraft DA40



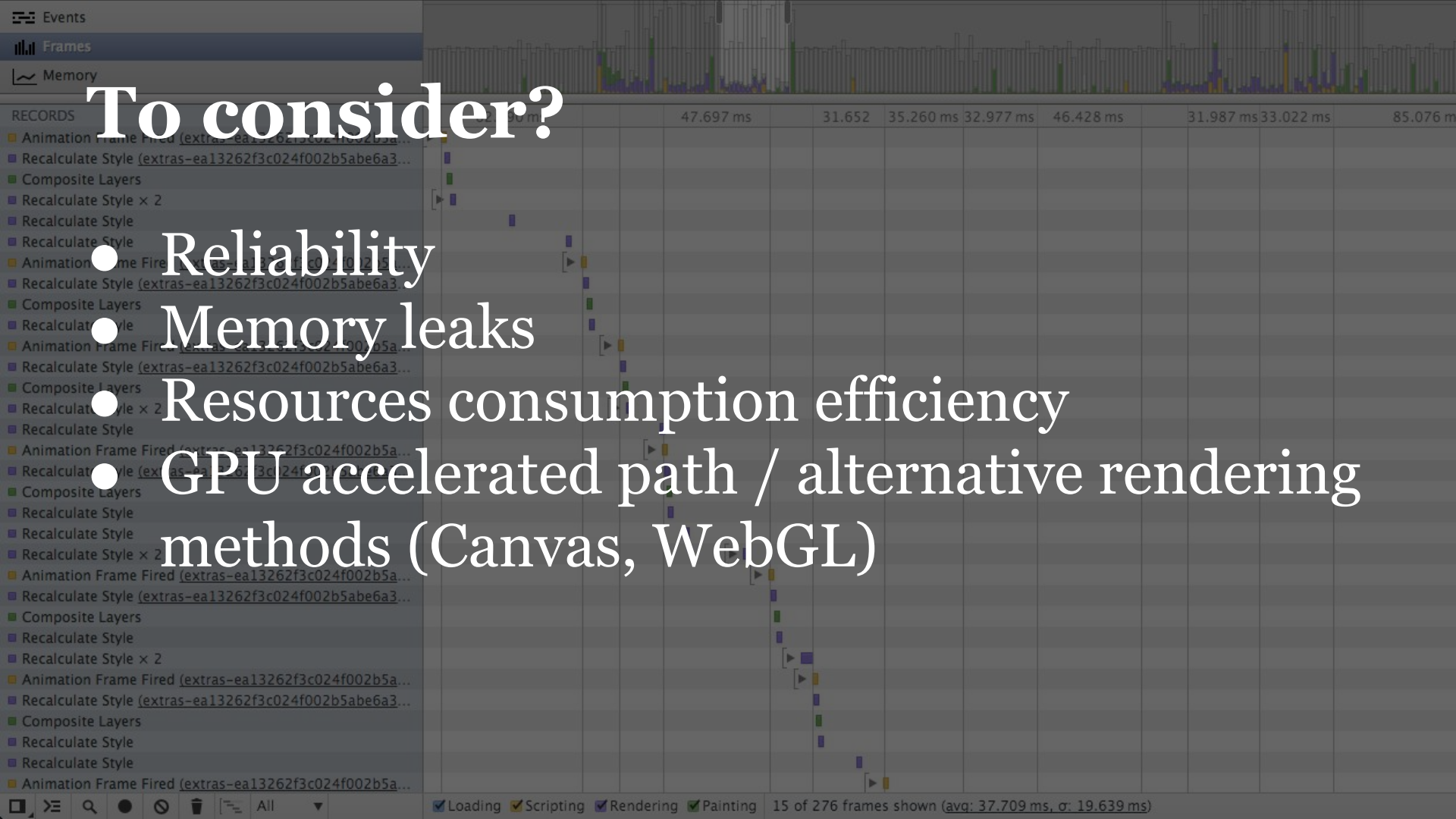
Let`s try

Live demo



To consider?

- Reliability
- Memory leaks
- Resources consumption efficiency
- GPU accelerated path / alternative rendering methods (Canvas, WebGL)



Scared flying JS driven airplane?

Most likely because of your
expectations from JS developers



ST 7R - 25L

A silhouette of a parent catching a child against a cloudy sky. The child is jumping from the left, and the parent is on the right, reaching out with their hands to catch the child. The background is a dark, cloudy sky with some light rays visible in the upper left.

This is all about trust

Trust is based on
expectations of what is normal for JS



More guidelines?

Let`s have a look at
Jet Propulsion Laboratory

A detailed CGI rendering of the Voyager 1 spacecraft against a dark blue space background filled with stars. The probe is shown from a three-quarter perspective, highlighting its complex structure including the main body, various instruments, and the long, thin boom extending from the side. The large, circular high-gain antenna is prominent. The text is overlaid on the central part of the image.

Voyager: 36 years without bugs

Can your JavaScript do this?

horizon



unitymedia



Performance and stability are priorities



Month without a reset ?



Code guidelines to the rescue...



Rule #1

No function should be longer than what can
be printed on a single sheet of paper



MARGARET HAMILTON
next to printed source code
of Apollo mission



Rule #1 - Do one thing

Long functions: less readable, not reusable,
harder to test, harder to refactor

Rule #1 - Readability

```
makeCoffeeAndCookEgg(){  
  let teapot = new Teapot();  
  let cup = new Cup();  
  let pan = new Pan();  
  let egg = new Egg();  
  teapot.on().then(teapot.fill.bind(cup));  
  pan.on()  
    .then(egg.breake)  
    .then(egg.fill.bind(cup));  
}
```

```
makeCoffeeAndCookEgg(){  
  makeCoffee();  
  cookEgg();  
}
```

Rule #1 - Easy to refactor

```
makeCoffeeAndCookEgg(){  
  let teapot = new Teapot();  
  let cup = new Cup();  
  let pan = new Pan();  
  let egg = new Egg();  
  teapot.on().then(teapot.fill.bind(cup));  
  pan.on()  
    .then(egg.breake)  
    .then(egg.fill.bind(cup));  
}
```

```
makeCoffeeAndCookEggAfter(){  
  makeCoffee().then(cookEgg);  
}
```

Rule #1 - Reusability

```
makeCoffeeAndCookEgg(){  
  let teapot = new Teapot();  
  let cup = new Cup();  
  let pan = new Pan();  
  let egg = new Egg();  
  teapot.on().then(teapot.fill.bind(cup));  
  pan.on()  
    .then(egg.breake)  
    .then(egg.fill.bind(cup));  
}
```

```
makeCoffeeAndCookEgg(){  
  makeCoffee();  
  cookEgg();  
}  
  
makeCoffeeAndDoToast(){  
  makeCoffee();  
  doToast();  
}
```




Rule #1 - Do one thing

Long functions: less readable, not reusable,
harder to test, harder to refactor

Rule #2

Restrict all code to very simple control flow constructs – do not use goto statements and direct or indirect recursion

Rule #2 - Predictability

Restrict all code to very simple control flow constructs – do not use goto statements and direct or indirect recursion

Rule #2 - Predictability

- If you want to write reliable code – drop to write cool one and write predictable
- Define coding standard and follow it
- Use static analysis to support standard and reduce chance for defect: ESLint + whole lot of plugins, presets
- Collect metrics: SonarQube, Scrutinizer, Plato
- Analyze types: Flow/ Closure Tools / TypeScript



Rule #3

Do not use dynamic memory allocation after
initialization



Rule #3 - Respect RAM

GC might become your enemy



Measure

DevTools / Timeline



Compare

DevTools / Profile / Take heap snapshot

Rule #3 - Respect RAM

- Manage your variables with respect. Declare at the top of scope to increase visibility, ESLint vars-on-top. Sort for predictability sort-vars
- Watch for memory leaks, clean listeners and variables when not needed anymore
- ESLint no-unused-vars
- Switch JavaScript to static memory allocation mode via object pooling

Object pooling?

No new objects in run time.

```
const pool = createObjectsPool(256);  
let object = pool.getObject();  
pool.releaseObject(object);
```


A background image showing a sunset or sunrise from the International Space Station (ISS). The horizon is a thin line of orange and yellow light against a dark blue sky. The foreground is a solid black area.

Rule #4

All loops must have a fixed upper-bound

A grayscale image of Jupiter's Great Red Spot, a massive storm system, serving as a background for the text.

Rule #5

The assertion density of the code should average to a minimum of two assertions per function

A large, circular, grayscale image of Jupiter's Great Red Spot, which is a massive storm on the planet. The image is centered in the background of the slide.

Rule #5 - Test well

The assertion density of the code should average to a minimum of two assertions per function

Rule #5 - Test well

- Higher tests density is less defects you get
- Minimal amount of tests is 2 per function
- Watch for anomalies in system state during run time. Generate and handle errors in case of critical failures
- Measure coverage but be aware, 100% coverage does not necessarily mean you have well tested code

A satellite image of the Earth at night, showing the glowing lights of cities and towns across the landmasses. The lights are concentrated along the coastlines and in major urban centers, creating a pattern of bright yellow and orange spots against the dark background of the oceans and unlit land.

Rule #6

Data objects must be declared at the smallest possible level of scope

Rule #6 - No shared state

Mutable shared state decreases predictability, testability since any part of system can write there without notifying the rest

Rule #6 - No shared state

ESLint pureness plugin

Rule #7

The return value of non-void functions must be checked by each calling function, and the validity of parameters must be checked inside each function

Rule #8

The use of the preprocessor must be limited to the inclusion of header files and simple macro definitions

Nice to know when using transpilers

Performance of ES6 features relative to the ES5

The cost of transpiling ES2015

Rule #9

The use of pointers should be restricted.

Specifically, no more than one level of dereferencing is allowed. Function pointers are not permitted

Rule #9 - LoD

Dog.body.legs.run();

vs

Dog.run();

Rule #9 - Call chains

```
Object1.object2.object3.method();
```

vs

```
const object3 = Object1.object2.object3;  
object3.method();
```




Rule #10

All code must be compiled, from the first day
of development, with all compiler warnings
enabled



Rule #10 - Keep green

All code must be compiled, from the first day of development, with all compiler warnings enabled

The background of the slide is a deep space image of the Hourglass Planetary Nebula (MyCn18). It features a central bright green and blue core surrounded by concentric rings of glowing orange and red gas. The overall shape is hourglass-like, with a narrower waist in the center.

Rule #10 - If red?

What if we are already fu**ed up?



Rule #10 - If red?

Do not panic.
Simply, prioritize,
refactor and add tests piece by piece.

A black and white photograph of an astronaut in a full spacesuit standing on the lunar surface. The astronaut is positioned in the center of the frame, facing forward. The ground is covered in lunar dust and small rocks. The background is a dark, featureless sky.

Small step for developers but...

Big step for web platform
to be perceived as reliable

A Eurofighter Typhoon fighter jet is shown in a steep climb, banking sharply to the left. The aircraft is heavily armed with a variety of missiles and bombs mounted under its wings and fuselage. The background consists of a cloudy sky. The text "Ok, not that far yet..." is overlaid in a large, white, serif font.

Ok, not that far yet...

But still, why not HTML/JS for instruments?

Eurofighter maneuvering

A Sukhoi SU-30 fighter jet is shown in a steep climb against a blue sky with scattered white clouds. The aircraft is grey with a red, white, and blue Indian Air Force roundel on the fuselage. The tail fin features the number 'SB 107' and the Indian national flag. The background shows a rugged, brown mountain range.

Ok, not that far yet...

But still, why not HTML/JS for instruments?

SU-30 take-off

The background image shows a close-up of an aircraft's Electronic Flight Instrument System (EFIS) displays. On the left, a Primary Flight Display (PFD) shows a heading scale from 240 to 300 degrees, with a heading indicator needle pointing around 270 degrees. To its right is a Multifunction Display (MFD) showing a vertical speed scale from 200 to 300 feet per minute, with a vertical speed indicator needle pointing around 240 fpm. Further right, another MFD displays a horizontal speed scale from 10 to 100 knots, with a horizontal speed indicator needle pointing around 10 knots. The top of the displays shows various flight parameters like altitude (11585 feet) and speed (38000). The overall scene is dimly lit, typical of a cockpit at night or in low light conditions.

Ok, not that far yet...

But still, why not HTML/JS for instruments?

EFIS displays

A large red offshore supply ship, the XANTHIA, is shown from a high-angle perspective, sailing on a dark blue, choppy sea. The ship's hull is a deep red, and its superstructure is white. The name "XANTHIA" is visible in white lettering on the red hull near the bow. The ship is equipped with various deck structures, including cranes, ladders, and railings. A "NO SMOKING" sign is visible on the white superstructure. The ship is moving towards the left, leaving a white wake behind it. The text "And ... what about ships?" is overlaid in white serif font in the center of the image.

And ...
what about ships?

@PixelsCommander

denis.radin@gmail.com