3D Web Programming

Fivos DOGANIS

whoami

linkedin.com/in/fivosdoganis
fivos.doganis@gmail.com
github.com/fdoganis

ISEP
École d'ingénieurs du numérique

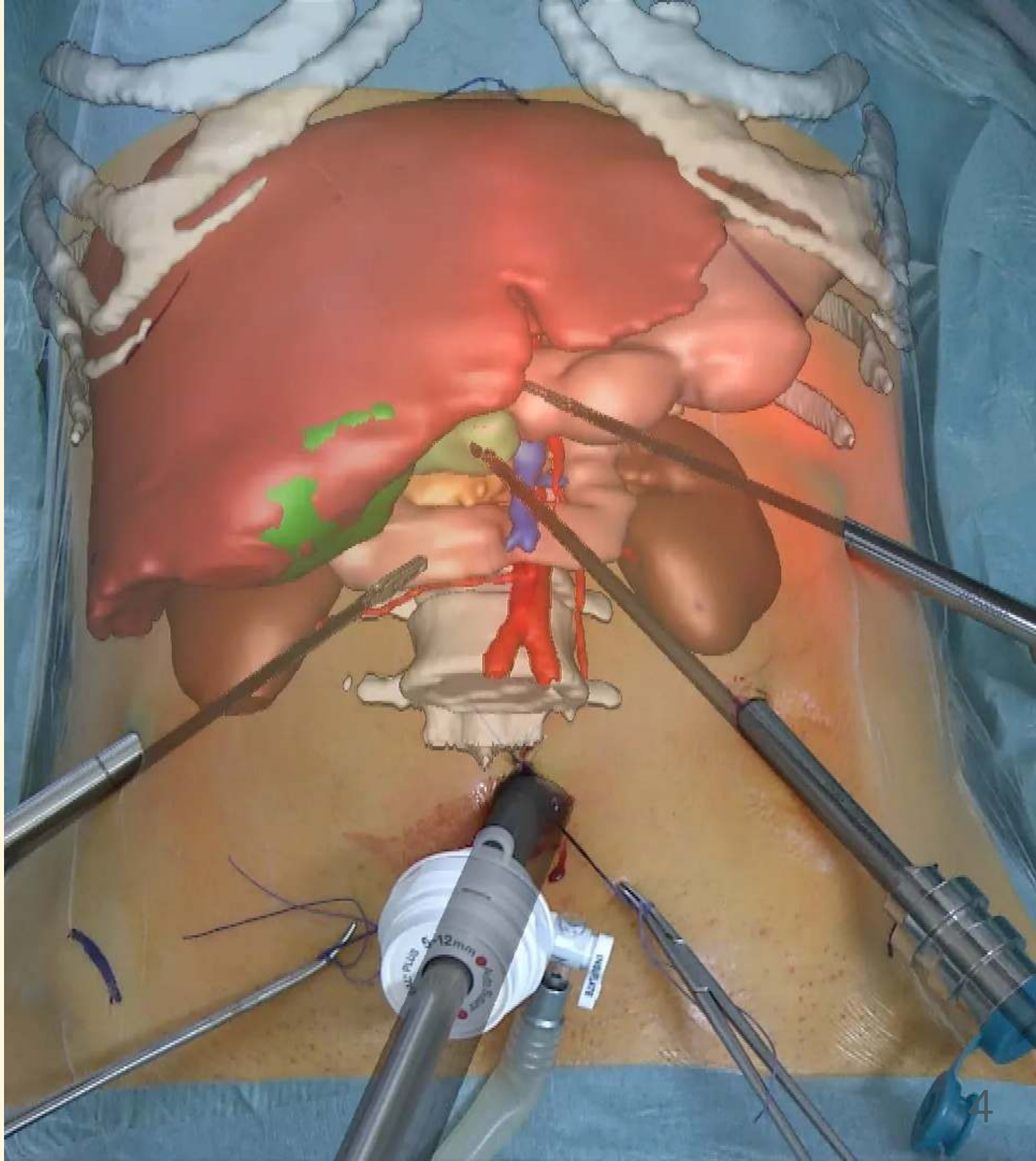
UNIVERSITY
OF HULL

[University of Hull]

- Master of Science by Research (2001)
  Augmented Reality in Archaeology: Registration Issues

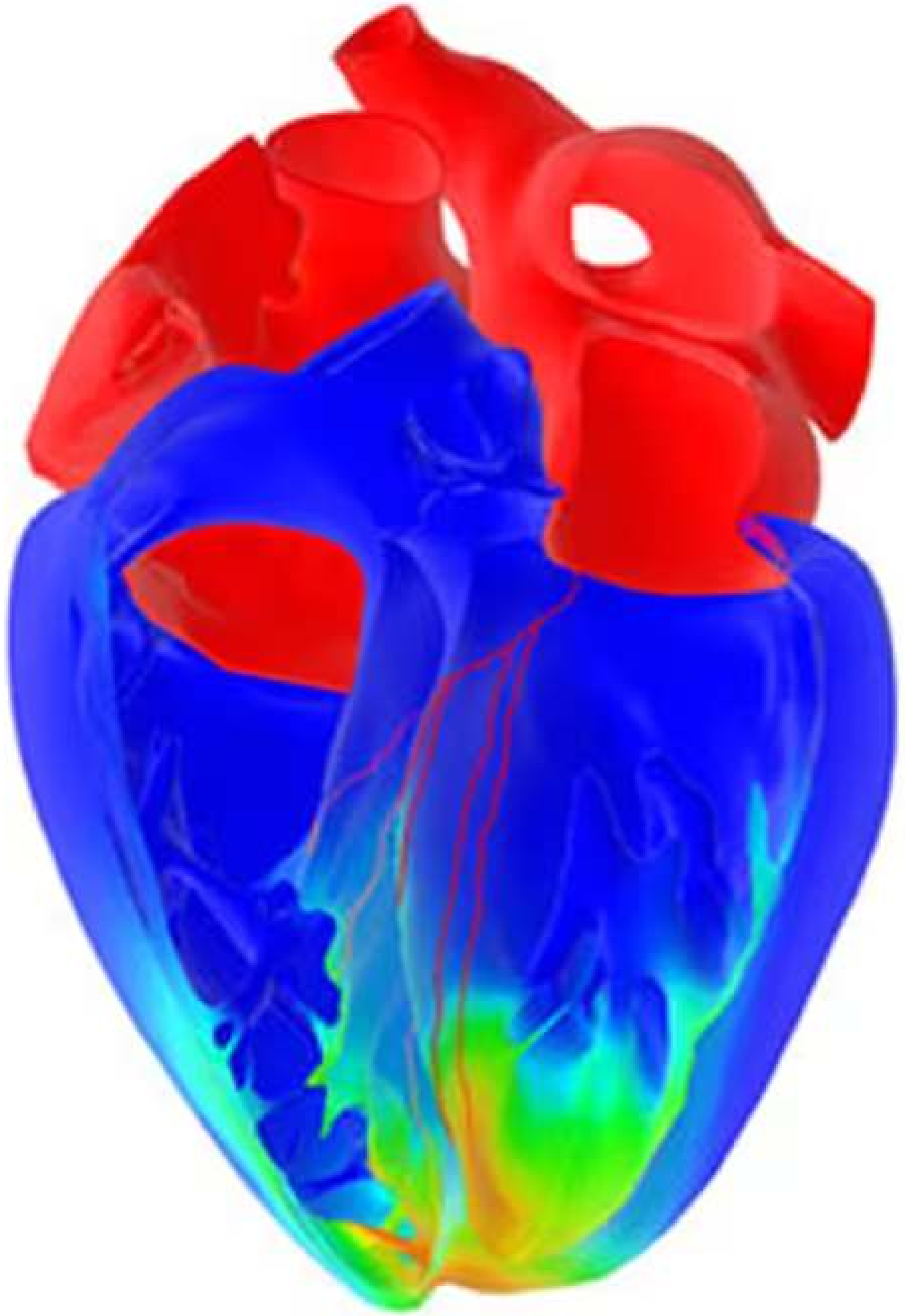
IRCAD (2002 - 2003)

- Institut de Recherche contre les Cancers de l'Appareil Digestif
- Startup
  - Virtual-Surg team
- Augmented Reality Research Engineer



Dassault Systèmes (2003+)

- 3D Visualization Engineer
  - Scenegraph, Materials
  - Geometry, Tessellation
- Virtual and Augmented Reality (XR) Engineer
- XR Research Engineer
- XR Research Manager

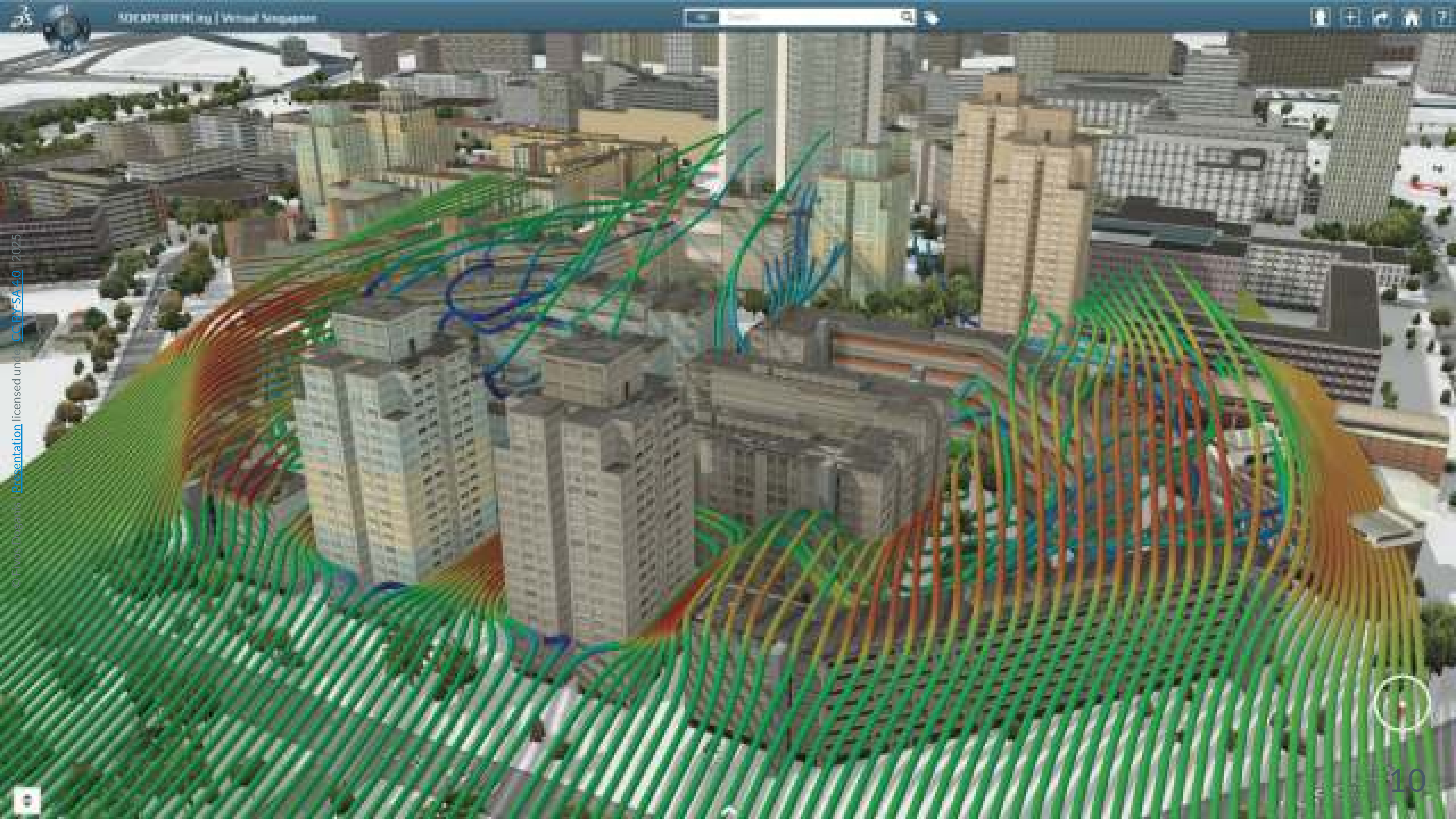


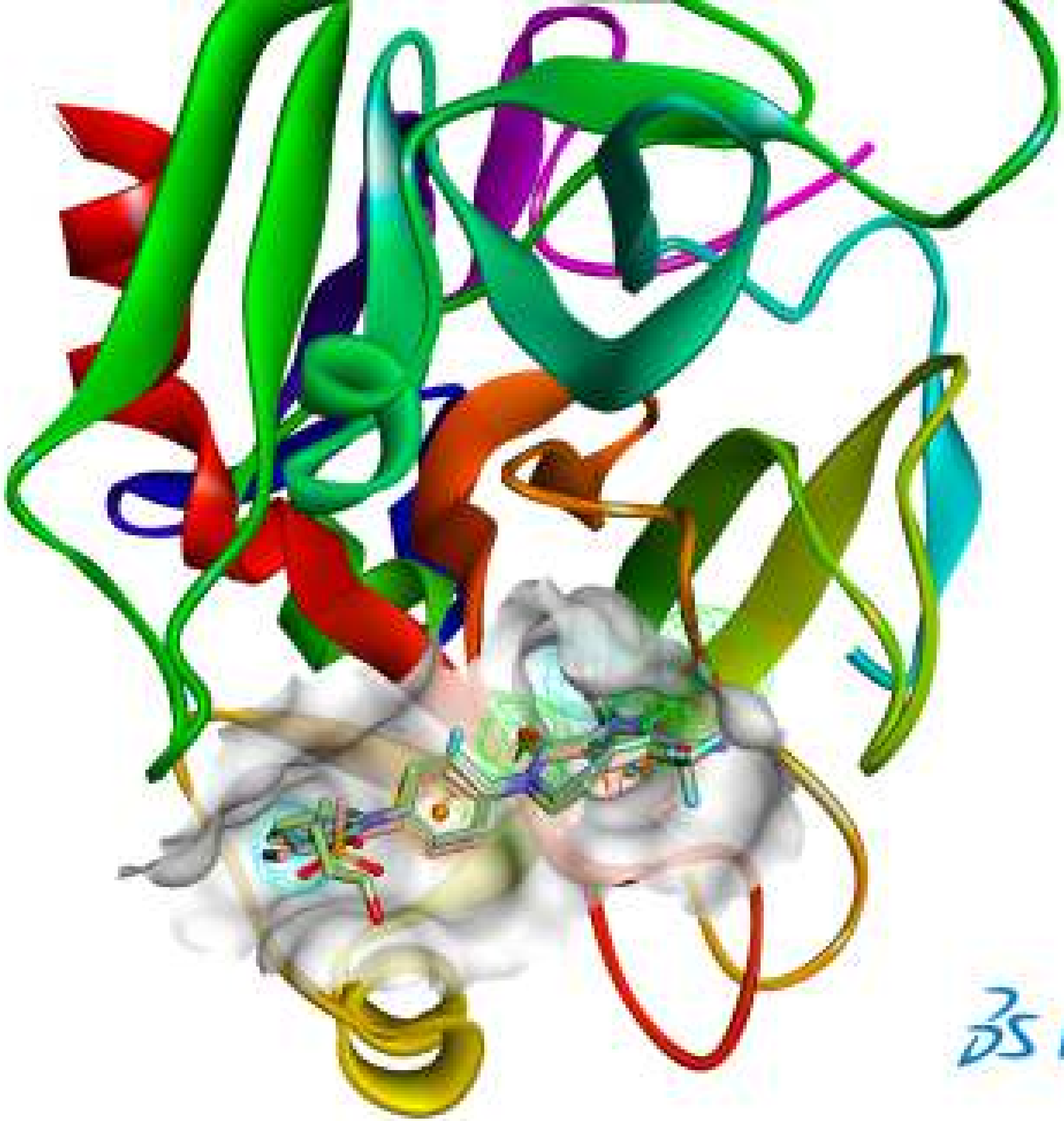
Dassault Systèmes

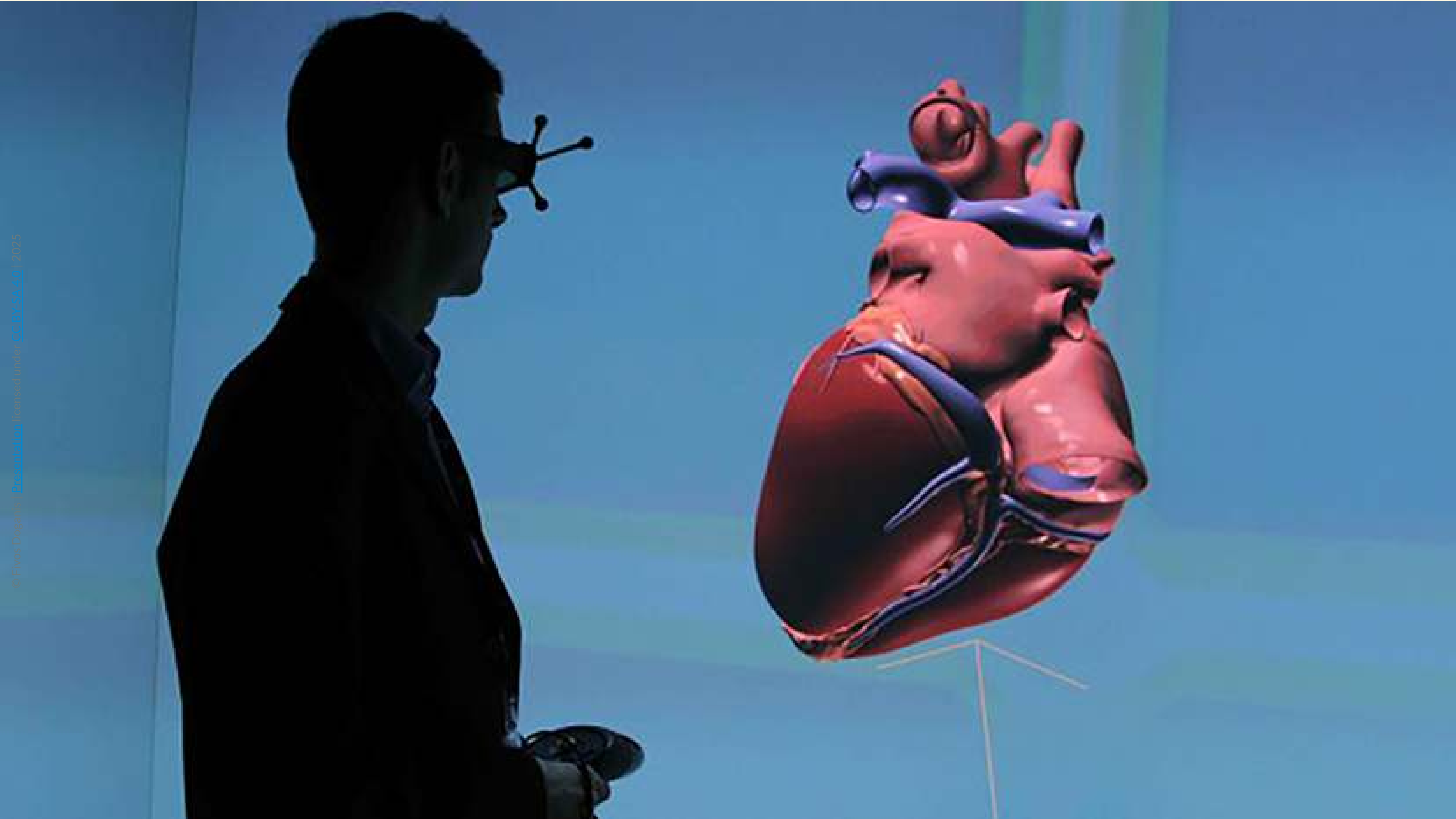
From Shape to Life


pininfarina
CATIA


SOLARIMPULSE ACROSS AMERICA


3DEXPERIENCity | Virtual Singapore





Course audience

- **Computer Science students** learning Computer Graphics, Physics or Machine Learning 🧑‍🎓
  - and who are afraid to ask "what's a GPU?", "what's a shader?"
- **Web designers** wishing to add 3D graphics to their sites 🎨
- **Game developers** wishing to conquer the Web 👩‍💻
- Anyone looking for a **simple introduction to 3D**
  - and who has no clue where to start 🧐

➡️ Feel free to skim through technical sections and use this course as future reference

Course prerequisites

We'll start from scratch but these should help:

- **Math** 🔢
  - 3D vectors and matrices
- **Programming** 💻
  - **JavaScript** notions, or any similar language (HTML kept minimal)
- 3D API 🧊
  - **OpenGL**, DirectX, Metal
- 3D Software (Blender, Unity, Unreal Engine, Godot Engine)
- Desktop / Laptop + VSCode

Course contents

- **Demo** 🕹️
  - Existing **3D Web Apps**
- **Theory** 📖
  - A brief **history** of 3D on the Web
  - **Concepts**: Architecture, Pipeline, APIs
- **Practice** 💻
  - 3D Web **Programming**
    - **WebGL**, **THREE.js**
    - Other APIs

📆 Planning

- **Day 1** (6 hours)
  - 📖 **Theory**
  - 💻 **WebGL exercises**
  - 🍜 Lunch
  - 🎮 **THREE.js Theory** + full **exercise**
  - 🛸 Explore examples + choose a personal **project**
- **Day 2** (6 hours)
  - 🖋 Evaluation: **Quizz** 20 questions / ~20 min ➡️ 20 points max
  - ⭐ **Personal project / game jam** ➡️ +5 bonus points!

Project evaluation

- ⭐ Project can be finished at home 🏠
- 👤 1 person per project
- send git repo link by mail (invite if private) with:
  - README
  - LICENSE
  - illustration (image / GIF / mp4)
  - link for live testing
  - source code

Project evaluation criteria

- originality 👀
- interactions 👋
- physics 💥 / animations 🏃 / sounds 🎶 / eye-candy 🎇
- GIS 🌍
- code quality ✨, tricks 😏, performance ⏱️
- fun 🎉

Project grading

- up to **5** bonus points
- choose features from previous slide
- for each implemented **feature**:
  - not done: **0 pt** 🥱 💤
  - nice try / buggy: **0.5 pt** 🤨 🐛
  - basic / good enough: **1 pt** 😐
  - great / polished: **1.5 pts** 🙂
  - impressive: **2 pts** 🤠 ⭐

3D Web Apps

- Games
- e-Commerce
- 3D content creation
- 3D data exploration
- Interactive art

Minecraft Classic

Aviator

Aviator 2

Heraclos, (Gobelins)

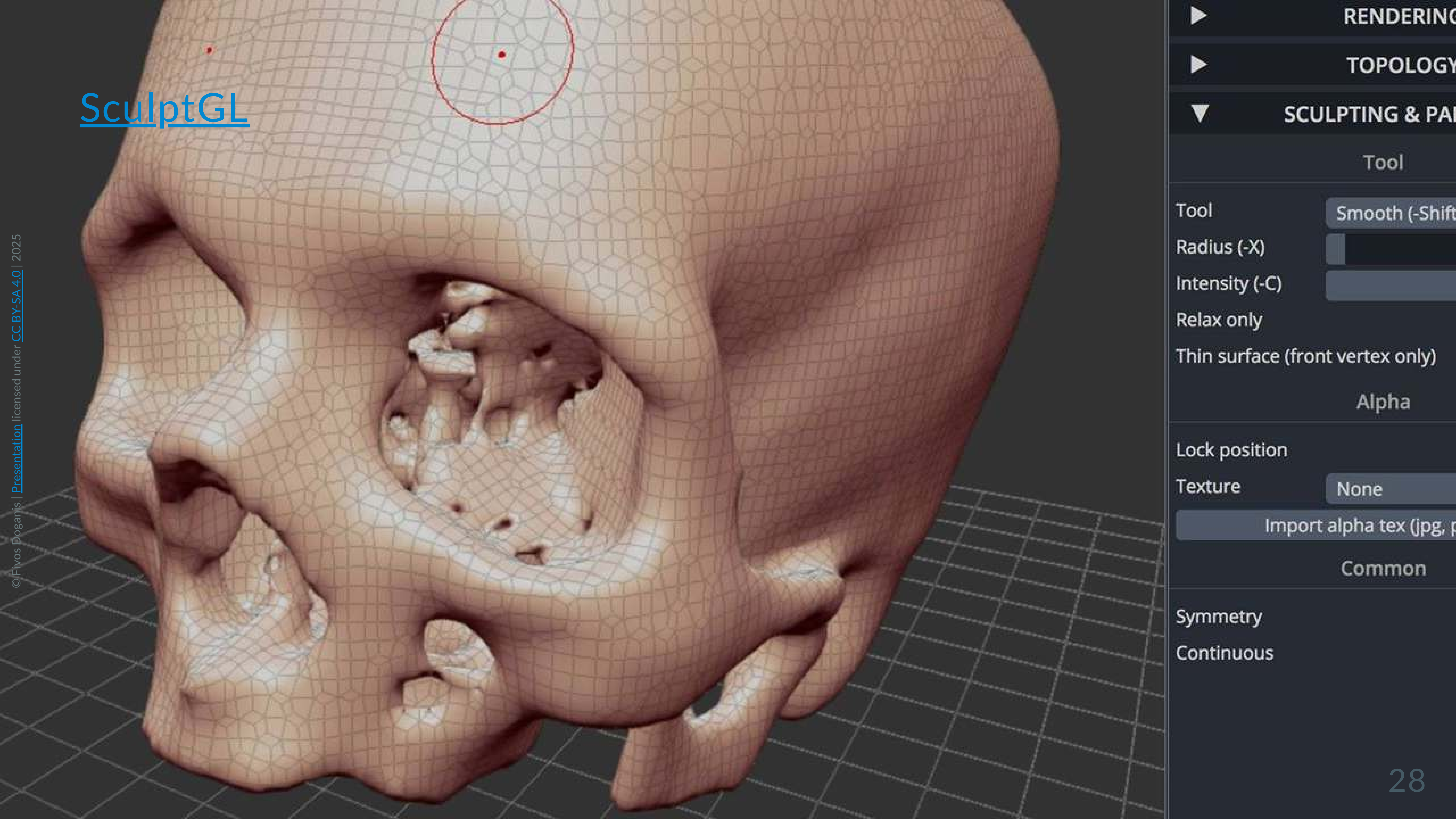
Blob Opera (Google)

Apple iPhone Studio

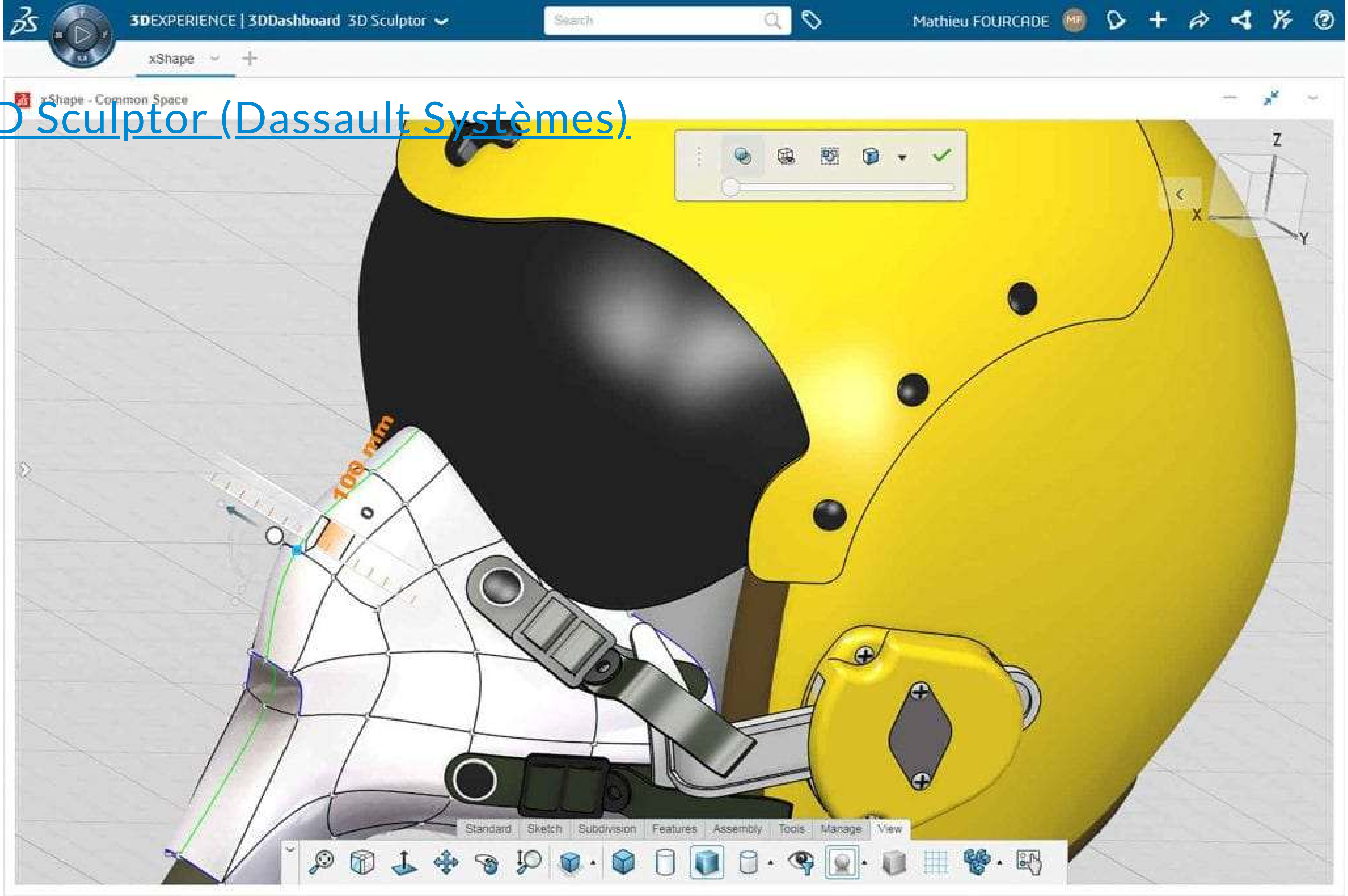
BMW

[OUIGO]

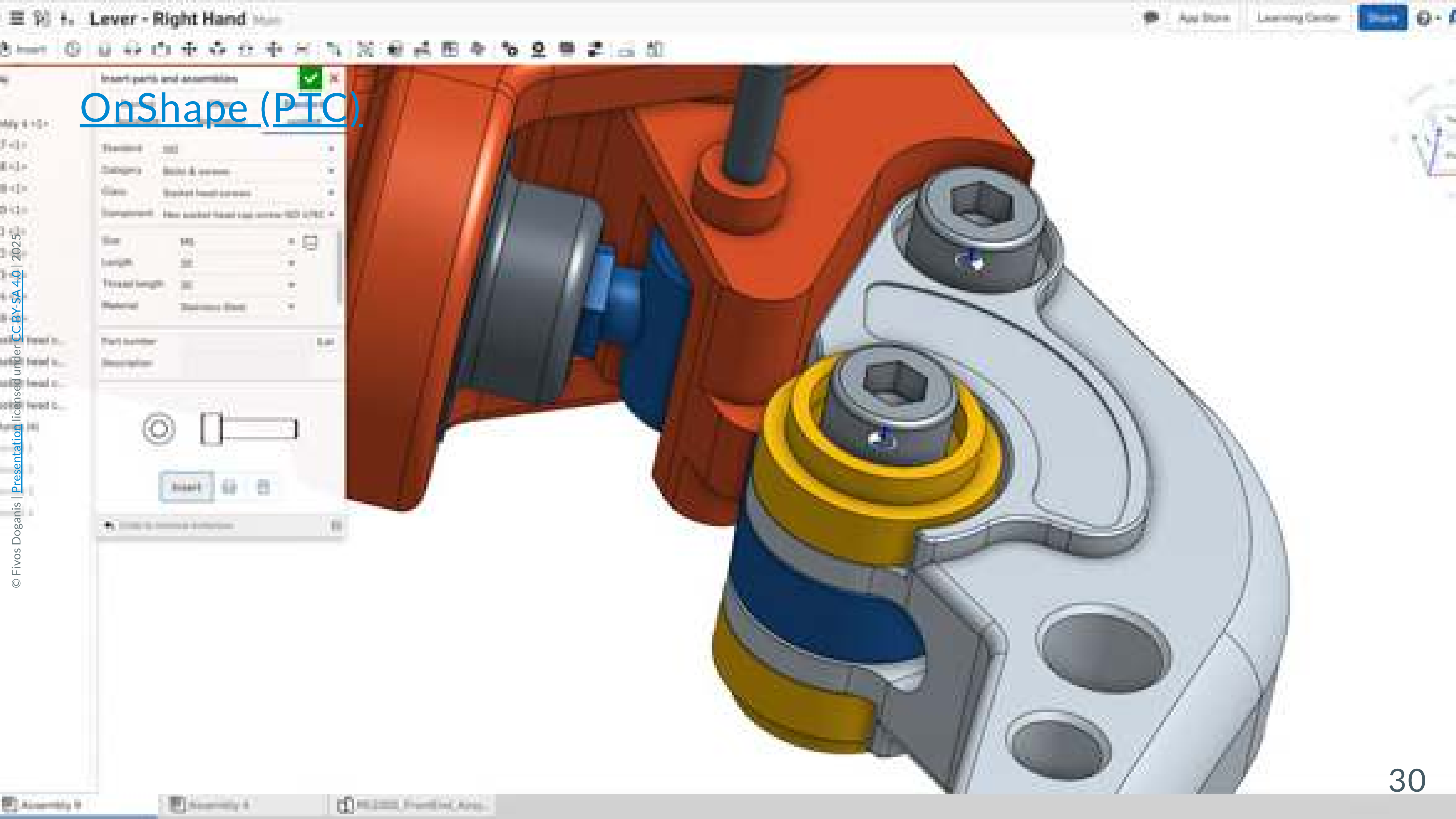
SculptGL


RENDERING
TOPOLOGY
SCULPTING & PAI
Tool
Tool
Smooth (-Shift
Radius (-X)
Intensity (-C)
Relax only
Thin surface (front vertex only)
Alpha
Lock position
Texture
None
Import alpha tex (jpg, p
Common
Symmetry
Continuous

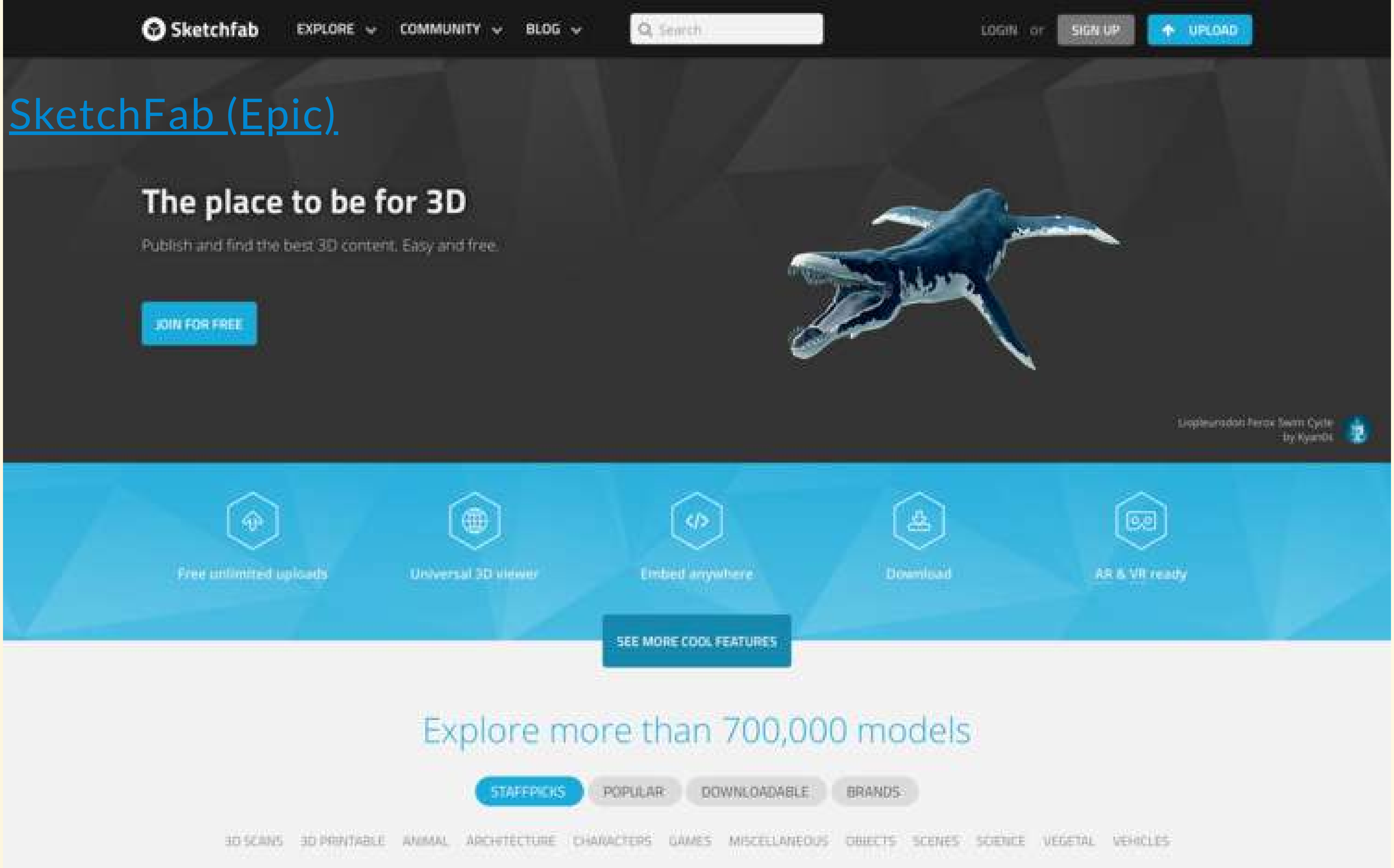
3D Sculptor (Dassault Systèmes)



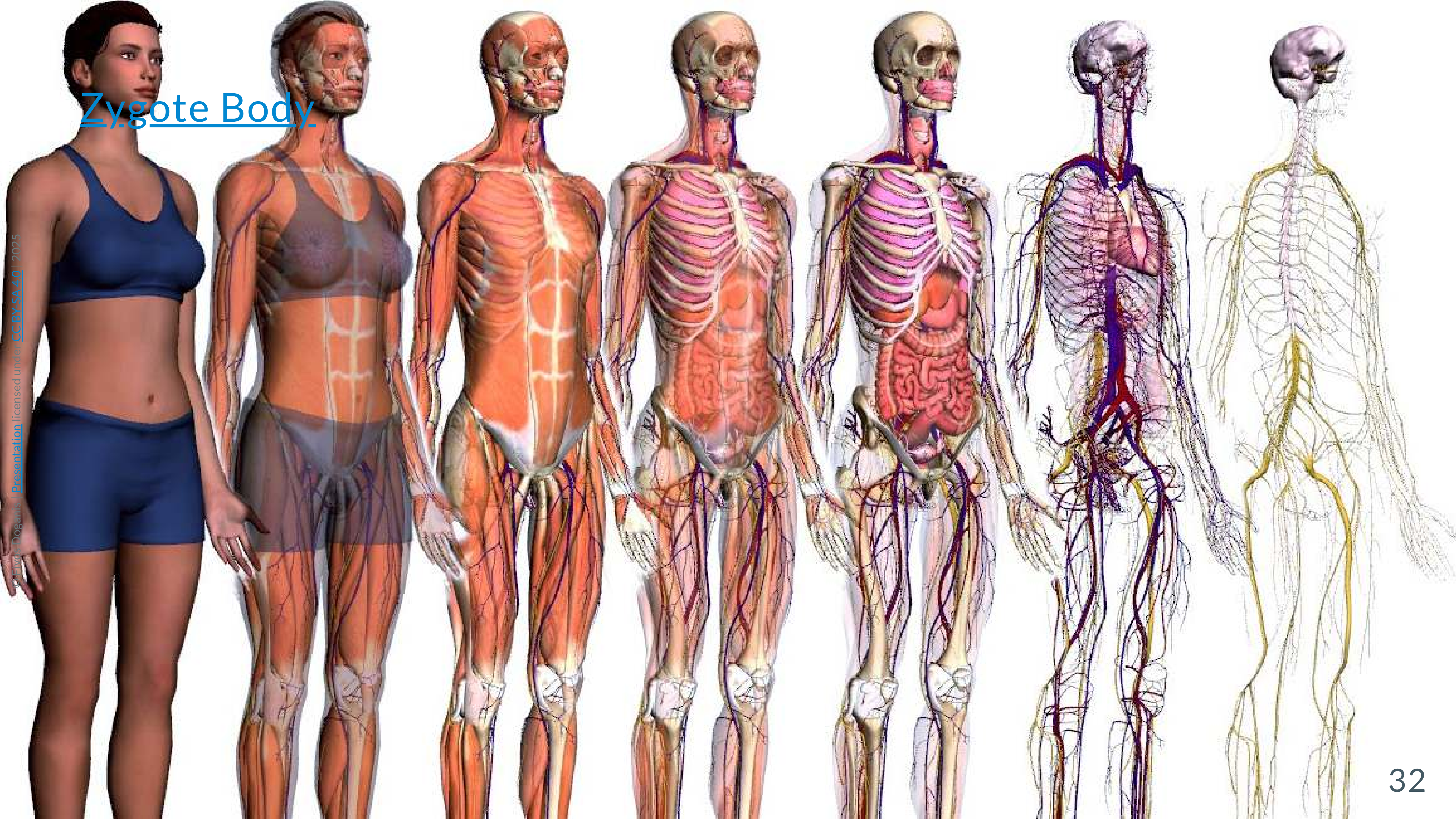
OnShape (PTC)



SketchFab (Epic)

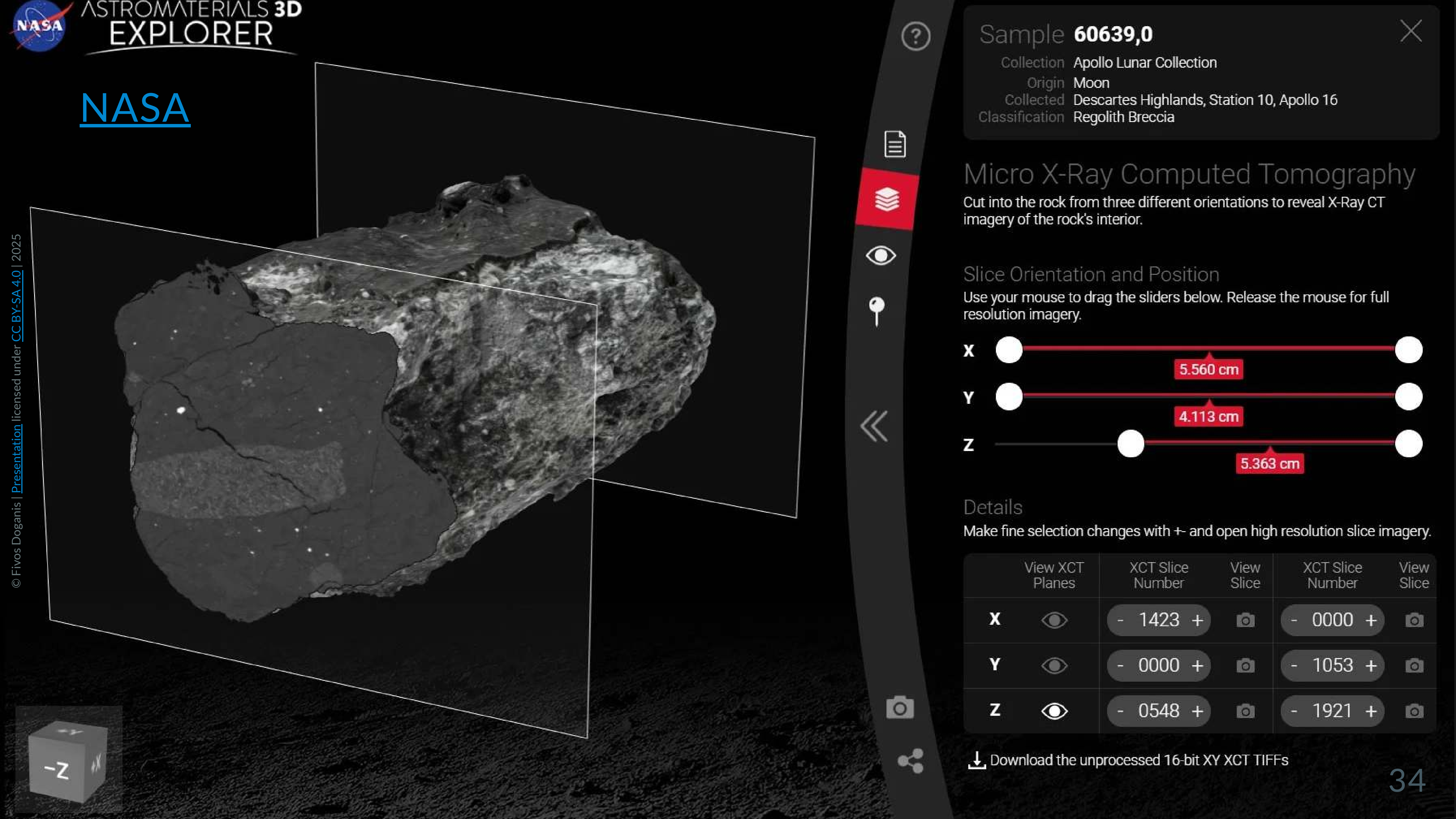


Zygote Body



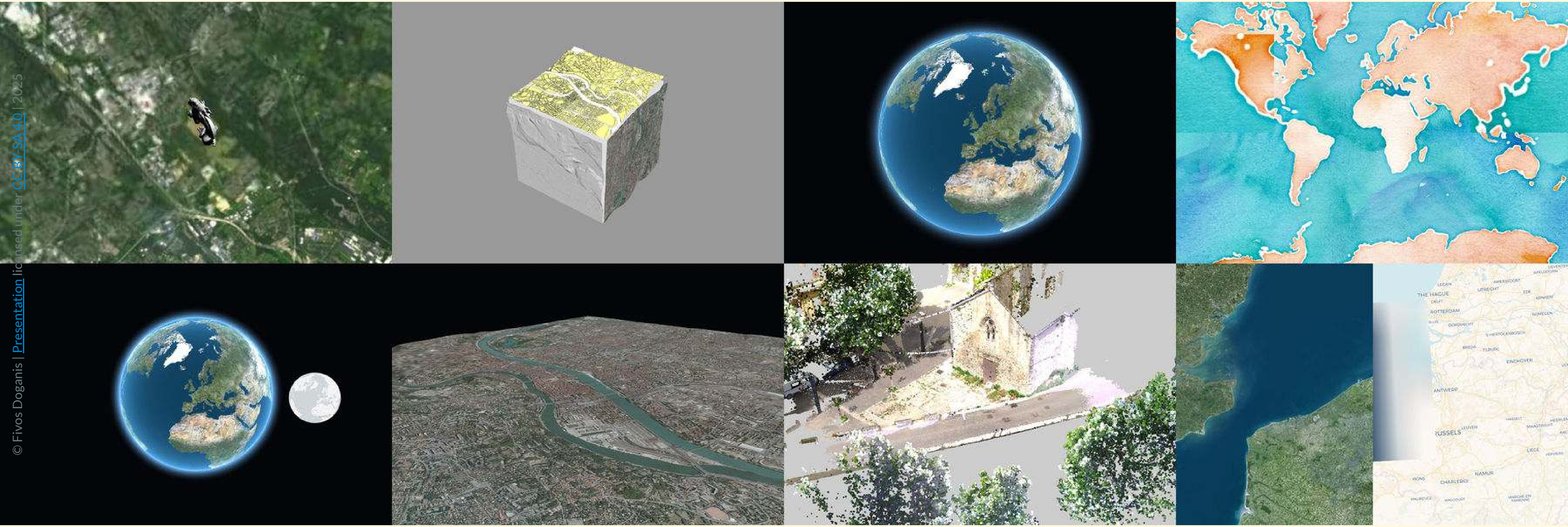
Z-Anatomy

NASA



ASTROMATERIALS 3D EXPLORER
Sample 60639,0
Collection Apollo Lunar Collection
Origin Moon
Collected Descartes Highlands, Station 10, Apollo 16
Classification Regolith Breccia
Micro X-Ray Computed Tomography
Cut into the rock from three different orientations to reveal X-Ray CT imagery of the rock's interior.
Slice Orientation and Position
Use your mouse to drag the sliders below. Release the mouse for full resolution imagery.
X 5.560 cm
Y 4.113 cm
Z 5.363 cm
Details
Make fine selection changes with +- and open high resolution slice imagery.
View XCT Planes
XCT Slice Number
View Slice
X - 1423 + - 0000 +
Y - 0000 + - 1053 +
Z - 0548 + - 1921 +
Download the unprocessed 16-bit XY XCT TIFFs
-Z

iTowns (IGN)


THE HAGUE
UTRECHT
ROTTERDAM
EINDHOVEN
ANTWERP
NAMUR
CHARLEROI

The Cursed Library

History

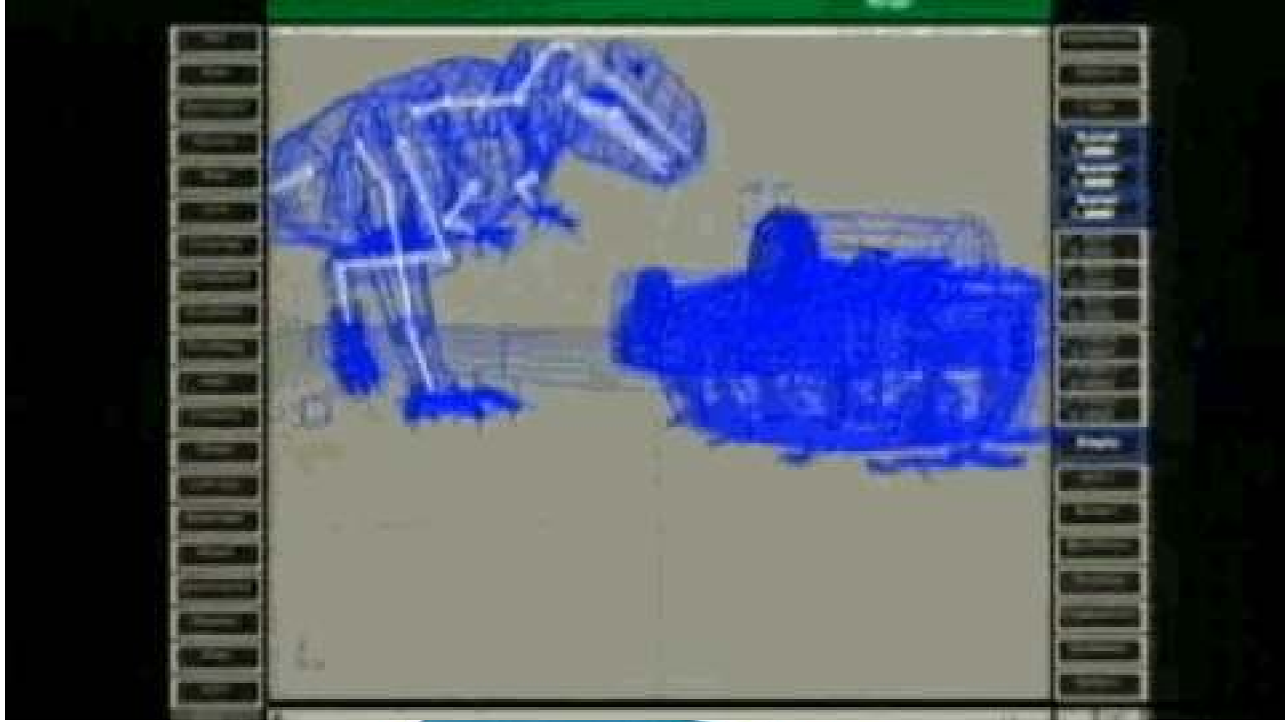
The past, present and future of Web 3D

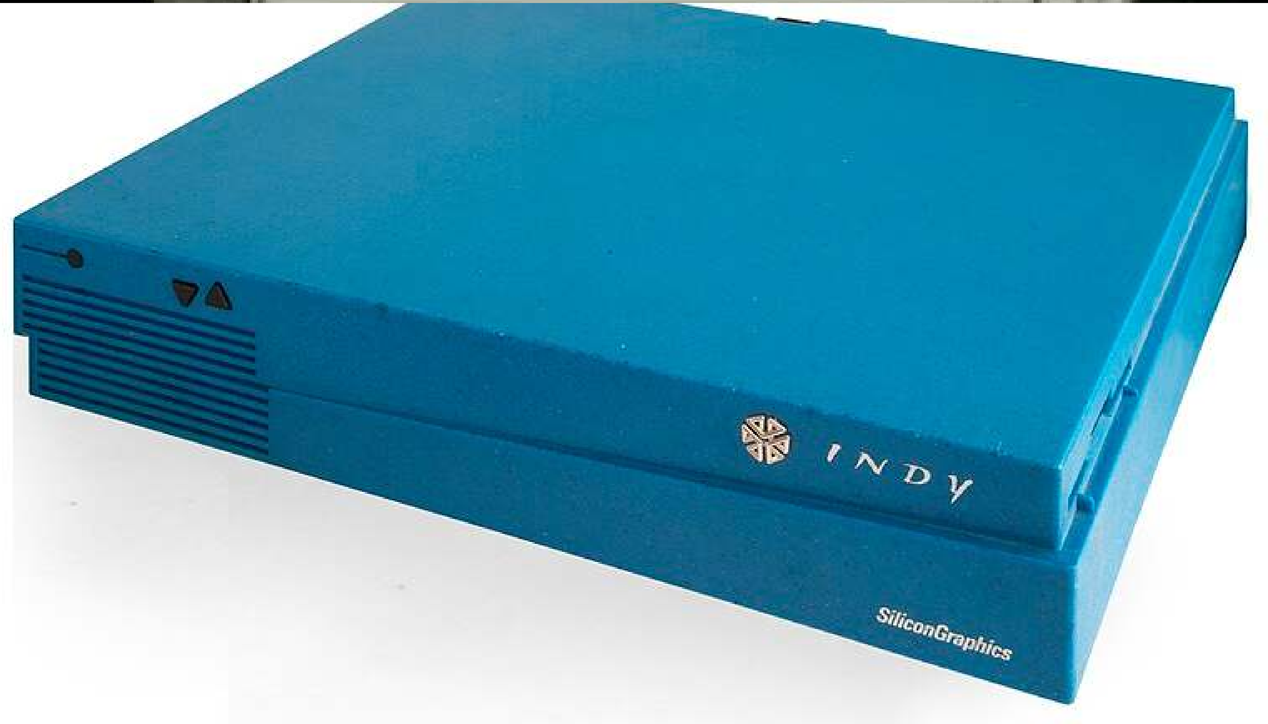
"Dis Papy, c'était comment la 3D avant?"

Prehistory

(1983 - 1993)

- Silicon Graphics (SGI) hardware only
    - IRIX OS
- IRIS GL (1983)
    - API **close to hardware**
- IRIS Inventor (1988)
- **OpenGL 1.0** (1993)
    - Open API, Multi-OS





Fixed Pipeline (1993 - 2004)

- 3dfx **Glide** API (1996)
  - Voodoo: "hardware 3D acceleration" for all
- Microsoft **Direct X** API (1997)
  - Windows-only 🙁
- **OpenGL ES** (2004)
  - **Subset** for "**E**mbedded **S**ystems" 📱 🎮
  - *"most widely deployed 3D graphics API"*
- **OpenGL 2.0** (2004)
  - **GLSL** Shaders 🎉

- Foundation of **nVIDIA** (1993)
- NV1 in **SEGA Saturn** (1994)
- GeForce 256 (1999)
  - democratizes the **GPU**: **G**raphics **P**rocessing **U**nit, Transform & Ligthing
- GeForce 3 (2001): NV2A in Microsoft's **Xbox**, **programmable shading**

Other players

- **SGI** + **Nintendo**: Project Reality / **N64** (1996)
  - SGI ends in 2006 💀
- **Imagination Technologies** (**PowerVR** GPU) + **Sega**: **Dreamcast** (1998)
- Intel i740 (1998) : OS in 3D
- **ATI (AMD)** + **Nintendo**: **Gamecube** (2001)

Shaders, Mobile, Web (2004+)

- OpenGL ES 2.0 (2007)
  - **Mobile subset with shaders**
- Canvas 3D (2007), **WebGL** ancestor
  - created by **Vladimir Vukićević** at Mozilla
- **WebGL 1.0 (2011)** ⭐ 🎉
  - OpenGL ES 2.0 functionality for the Web!
- OpenGL ES 3.0 (2012), **3.1** (2014): **not for Apple** 😢
- **WebGL 2.0** (2017)
  - OpenGL ES 3.0 exposed to the Web

“ **Before WebGL**, you couldn’t really do 3D on the web at all.
There was **powerful 3D hardware** everywhere on both desktops and mobile phones, but **the web couldn’t tap into any of it.**
There were some **plugins**, but users had to do an extra **installation step** that was a huge obstacle to adoption.
All the browser vendors knew that this was a **challenge** that needed to be resolved, which is why we came together as a **Khronos** Working Group. ”

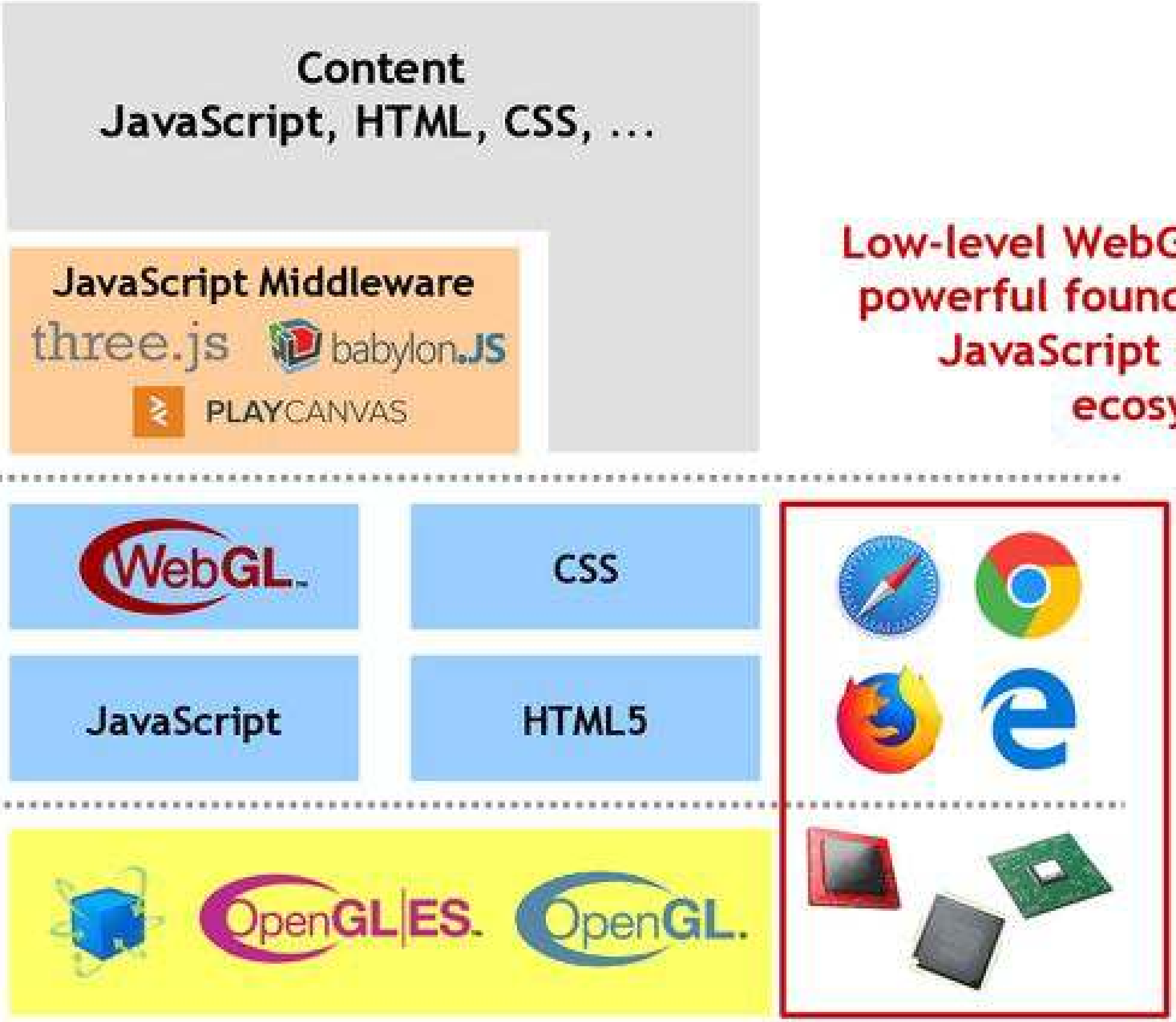
WebGL Stack

Content downloaded from the Web

Middleware provides accessibility for non-expert programmers
E.g. three.js library

Browser provides WebGL 3D engine alongside other HTML5 technologies - no plug-in required

OS Provided Drivers
WebGL uses native OpenGL or OpenGL ES or
Angle = OpenGL ES over DX9/11



Low-level WebGL API provides a powerful foundation for a rich JavaScript middleware ecosystem

Reliable WebGL relies on work by both GPU and Browser Vendors
->
Khronos has the right membership to enable that cooperation

WebGL architecture: software stack

- **Code**: HTML + CSS + JS
  - JS code inside the web page makes WebGL API calls
- **Browser**:
  - browser interprets JS code (using JS Engine)
  - turns WebGL calls into OpenGL calls (binding)
- **OS + Driver**: converts OpenGL calls to
  - DirectX calls on Windows, Metal on Apple (using ANGLE)
  - OpenGL or OpenGL ES calls on other OSes
- **CPU + GPU**: run the **hardware accelerated** code!

Binding example: from JS to C++

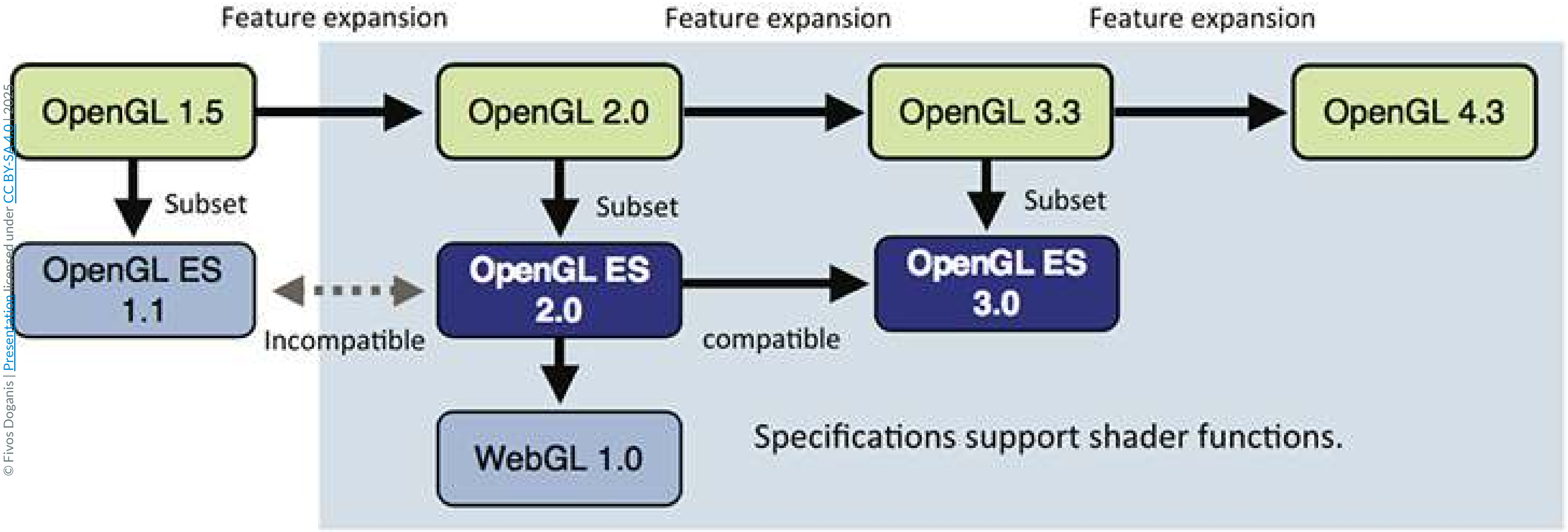
```
gl.drawElements(primitiveType, count, indexType, offset);
```

```
JSValue JSCanvasRenderingContext3D::glDrawElements(JSC::ExecState* exec, JSC::ArgList const& args)
{
    unsigned mode = args.at(0).toInt32(exec);
    unsigned type = args.at(1).toInt32(exec);

    unsigned int count = 0;

    // If the third param is not an object, it is a number, which is the count.
    // In this case if there is a 4th param, it is the offset. If there is no
    // 4th param, the offset is 0
    if (!args.at(2).isObject()) {
        count = args.at(2).toInt32(exec);
        unsigned int offset = (args.size() > 3) ? args.at(3).toInt32(exec) : 0;
        impl()->glDrawElements(mode, count, type, (void*) offset);
    } else {
```

Impact of stack on performance

- Interpreted JS code is ~10x slower than native code
  - unless you use WebAssembly ("only" 2x slower than native)
- On mobile devices, native code is ~10x slower than on desktop
- **Performance tips**
  - reduce processing in JS code, let the shaders do the hard work
  - once shaders are **compiled** and rendering data is on the GPU, the code runs at near **native speeds**
  - GPU memory is limited: use Draco geometry compression, and Basis GPU texture compression

Evolution



Feature expansion
Feature expansion
Feature expansion
OpenGL 1.5
OpenGL 2.0
OpenGL 3.3
OpenGL 4.3
Subset
Subset
Subset
OpenGL ES 1.1
OpenGL ES 2.0
OpenGL ES 3.0
Incompatible
compatible
WebGL 1.0
Specifications support shader functions.

WebGL's Evolution

After WebGL 2.0?
W3C is working on WebGPU
Layering over Vulkan/DX12/Metal
Possibly leveraging SPIR-V IR
https://www.w3.org/community/gpu/

Pervasive OpenGL ES 2.0
OpenGL and OpenGL ES ships on every desktop and mobile OS. 3D on the Web is enabled!

Mobile Graphics
Programmable Vertex and Fragment shaders

Desktop Graphics
Textures: NPOT, 3D, Depth, Arrays, Int/float
Objects: Query, Sync, Samplers
Seamless Cubemaps, Integer vertex attributes
Multiple Render Targets, Instanced rendering
Transform feedback, Uniform blocks
Vertex array objects, GLSL ES 3.0 shaders

Apple does not ship OpenGL ES 3.1
Cannot bring compute shaders into core WebGL

Compute Shaders

2007
OpenGL ES 2.0

4 years

2011
WebGL 1.0

2012
OpenGL ES 3.0

5 years

2017
WebGL 2.0

2014
OpenGL ES 3.1

Work in Progress
Compute Context
Multiview extension

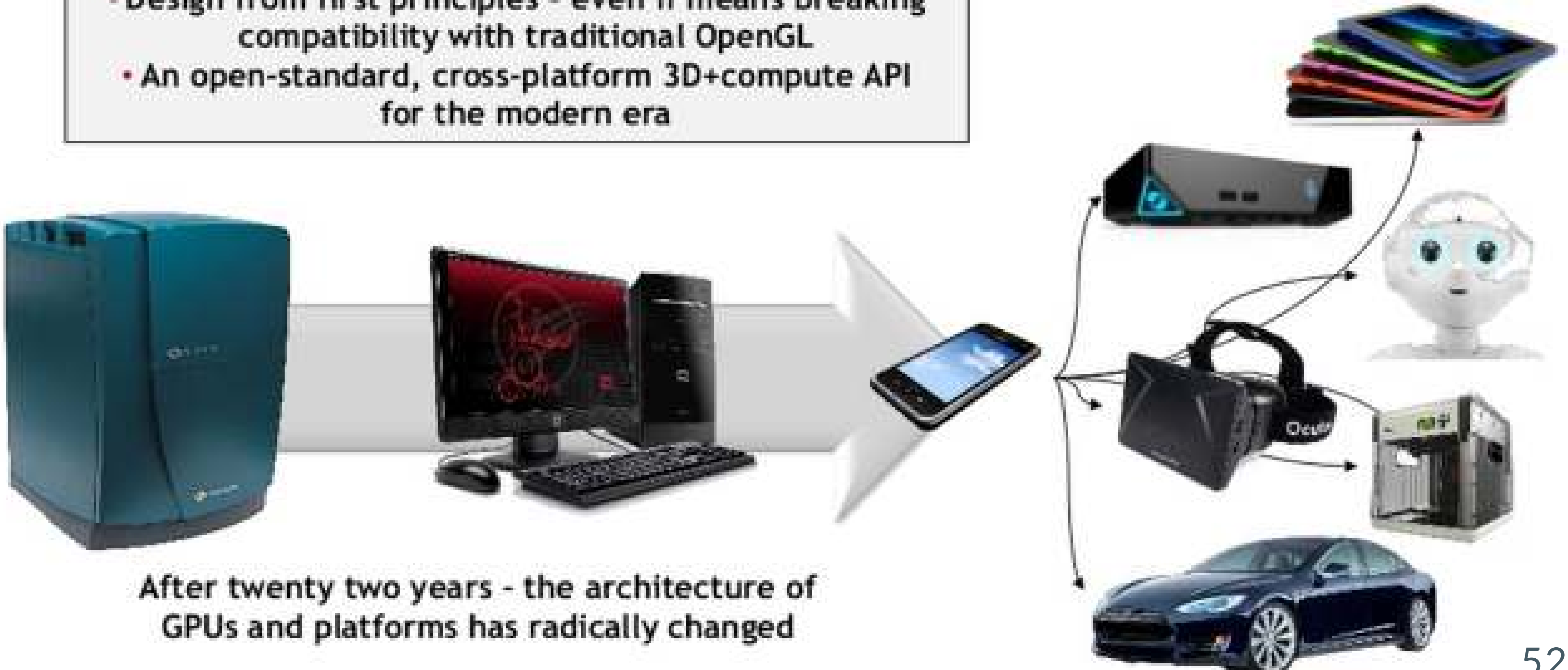
Conformance Testing is vital for Cross-Platform Reliability
WebGL 2.0 conformance tests are very thorough 10x more tests than WebGL 1.0 tests

The end of an API?

Next Generation OpenGL Initiative

- Ground up re-design of API for high-efficiency access to graphics and compute on modern GPUs and platforms
- Design from first principles - even if means breaking compatibility with traditional OpenGL
- An open-standard, cross-platform 3D+compute API for the modern era

Platform Diversity and need for cross-platform API standards *increasing*



After twenty two years - the architecture of GPUs and platforms has radically changed

Evolution issues

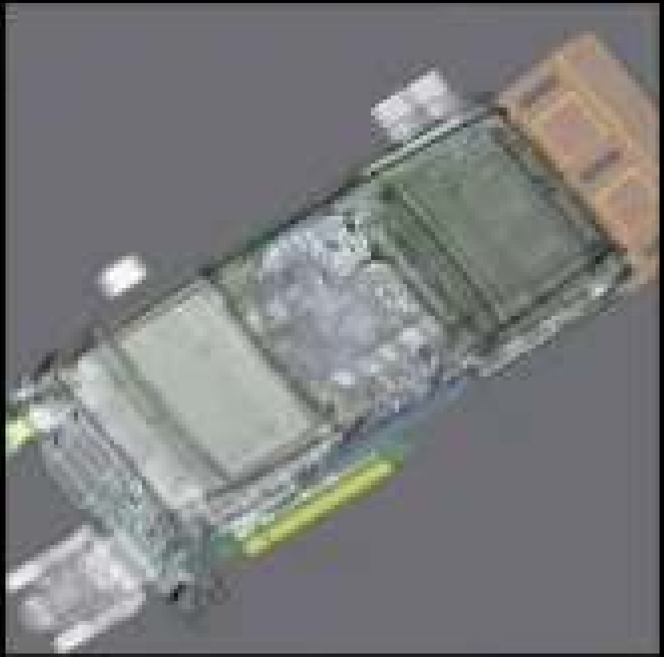
- New hardware, new needs since 1993
  - **mobiles**, wearables 📱 ⌚ 🕶
  - **embedded systems**, AI, Vision 🚗 🚀
- API has become more and more **complex**
  - coding fast and bug-free **drivers** is hard
  - OpenGL **extensions** are not universal
  - API **subset** needed to deprecate old and slow APIs
- ➡️ New API needed, and it should be
  - **low-level, universal, fast and abstract**

But do we really need a new API?

- Not really, see **AZDO**: Approaching Zero Driver Overhead (2016)
  - using the "right" OpenGL subset and the "right" extensions, we can squeeze as much performance as possible from the GPU
    - https://fr.slideshare.net/CassEveritt/approaching-zero-driver-overhead
    - https://fr.slideshare.net/tlorach/opengl-nvidia-commandlistapproaching-zerodriveroverhead
  - main idea
    - free the CPU ➡️ fewer DrawCalls
    - keep the GPU busy ➡️ send more data at once

Challenge of Issuing Commands

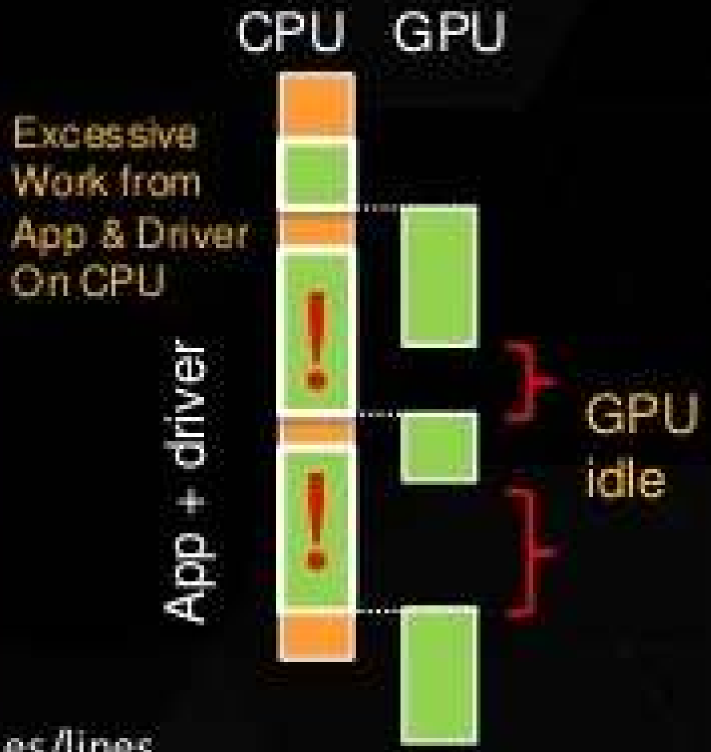
- Issuing drawcalls and state changes can be a real bottleneck



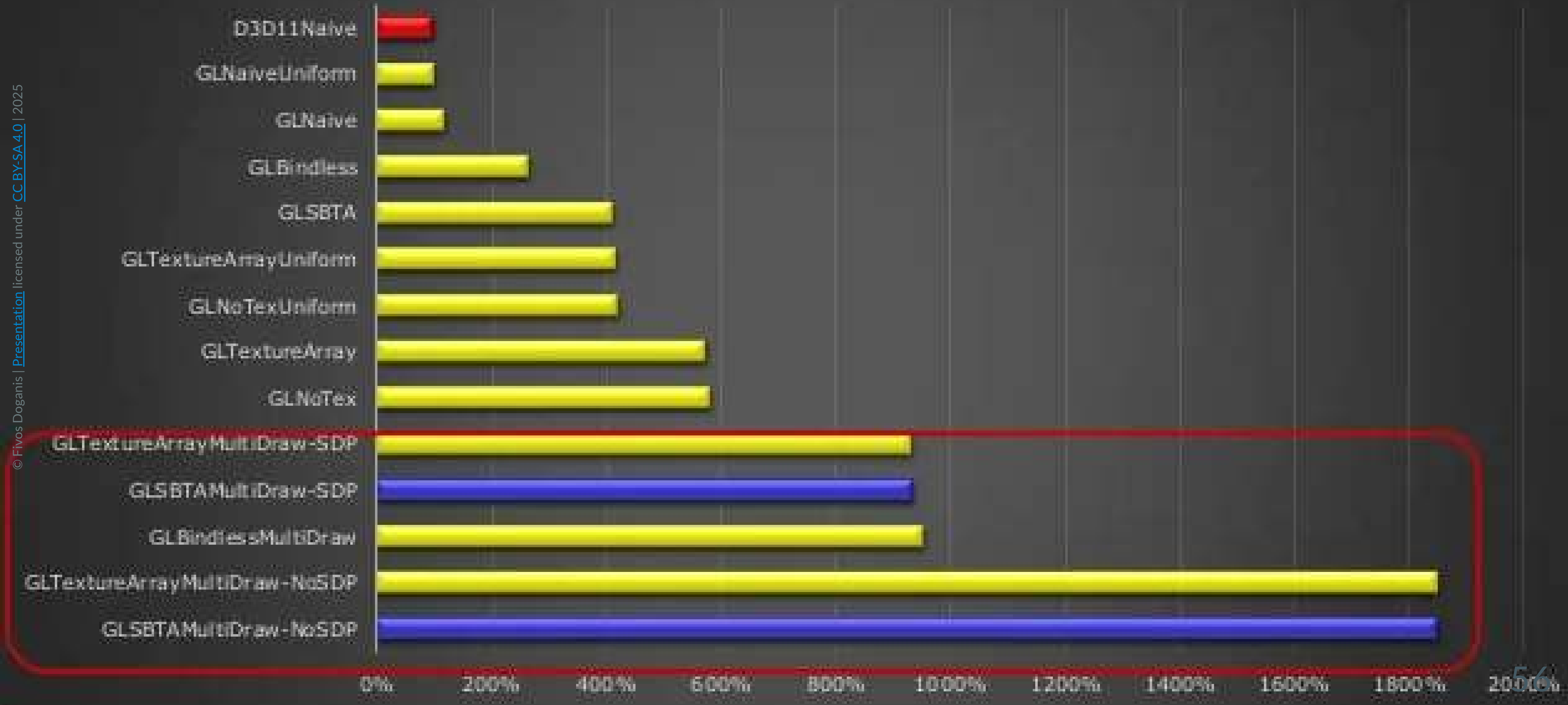
- 650,000 Triangles
- 68,000 Parts
- ~ 10 Triangles per part

- 3,700,000 Triangles
- 98 000 Parts
- ~ 37 Triangles per part

- 14,338,275 Triangles/lines
- 300,528 drawcalls (parts)
- ~ 48 Triangles per part



TexturedQuads – Normalized Obj/s



API Fragmentation 💥

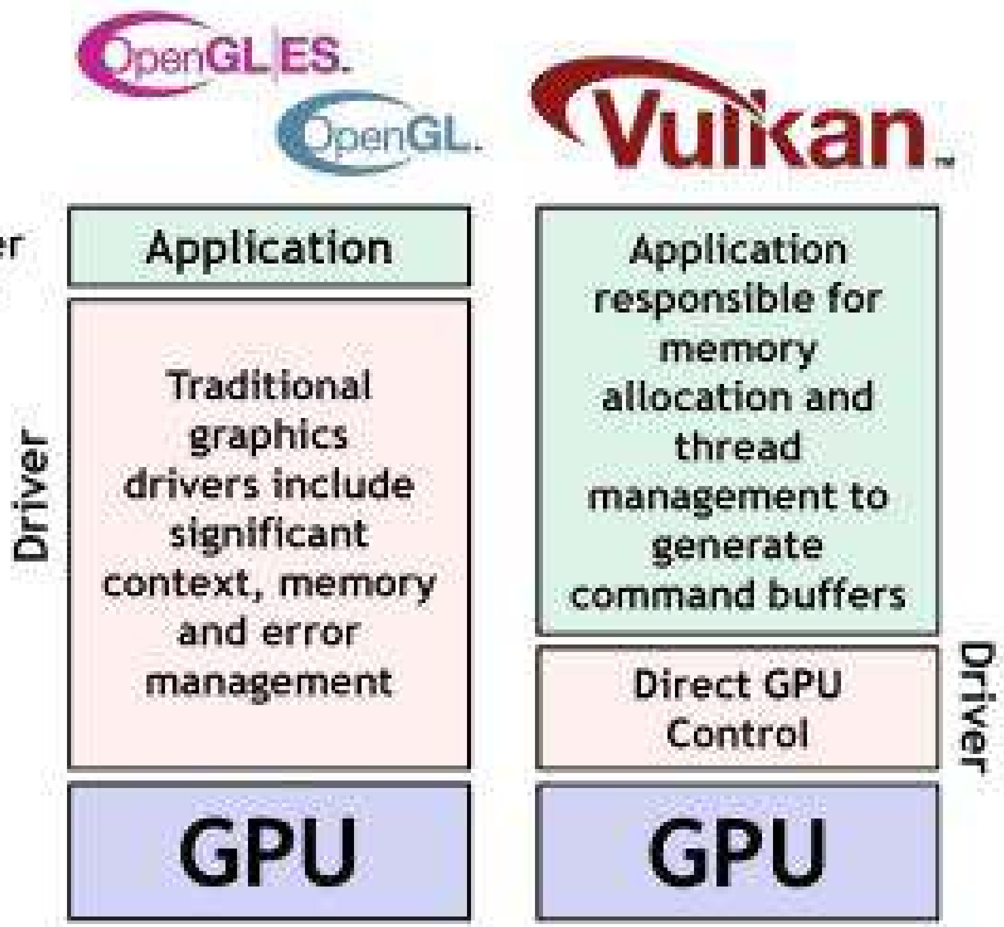
- Proprietary API proliferation ➡️ no portability 😦
  - close to the GPU ➡️ fast
  - new ➡️ "clean"
- **Direct X 12** (Microsoft) : Windows, Xbox
- **Mantle** (AMD) transferred to **Khronos** to become **Vulkan**
  - **new open low-level standard** ⭐
- **Metal** (Apple)
  - looks like Vulkan, Metal was created first

Vulkan Explicit GPU Control

Complex drivers lead to driver overhead and cross vendor unpredictability

Error management is always active

Driver compiles full shading language source



Simpler drivers for low-overhead efficiency and cross vendor consistency

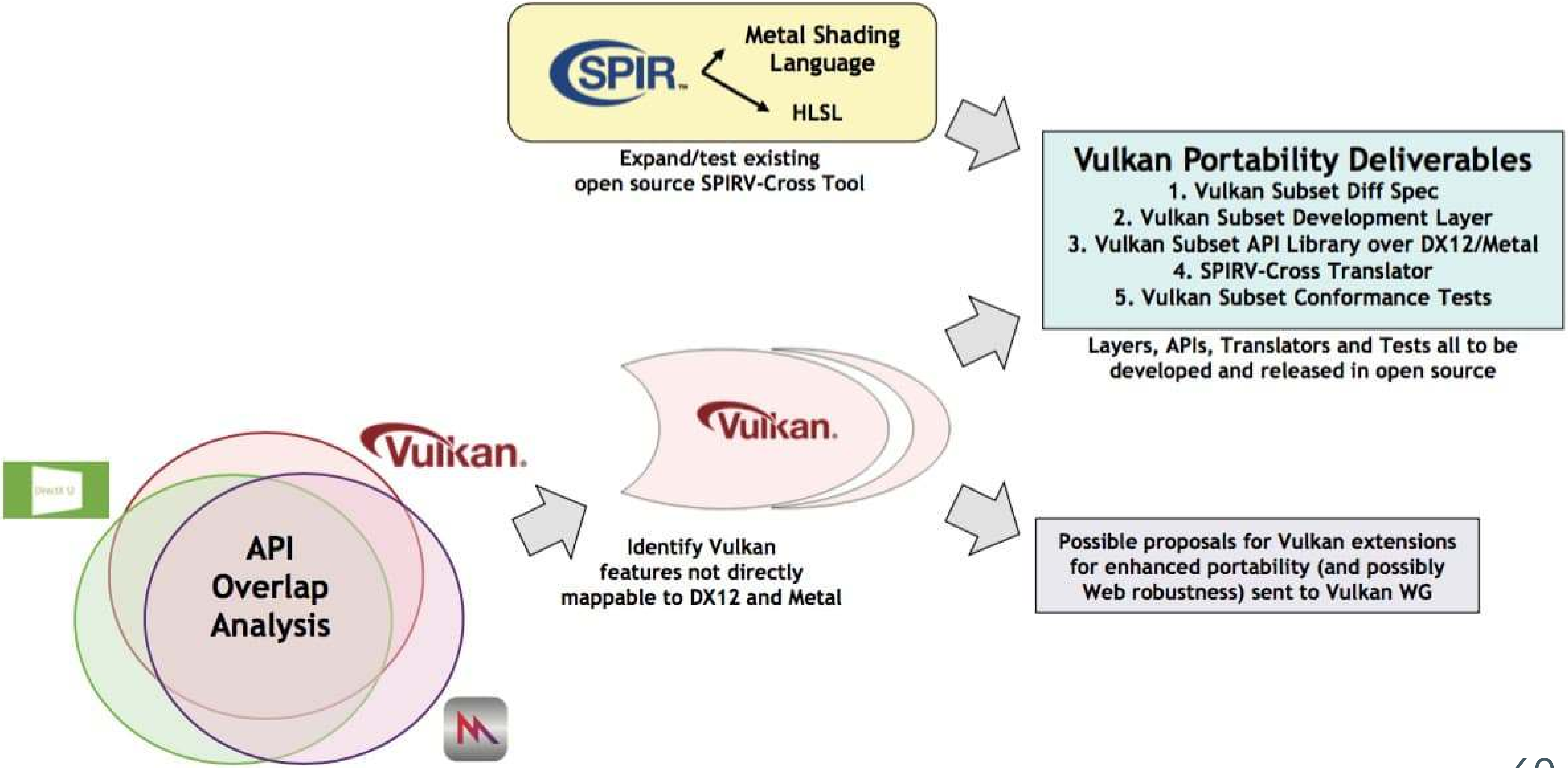
Layered architecture so validation and debug layers can be loaded only when needed

Run-time only has to ingest SPIR-V intermediate language

Can we reunite all these new APIs?

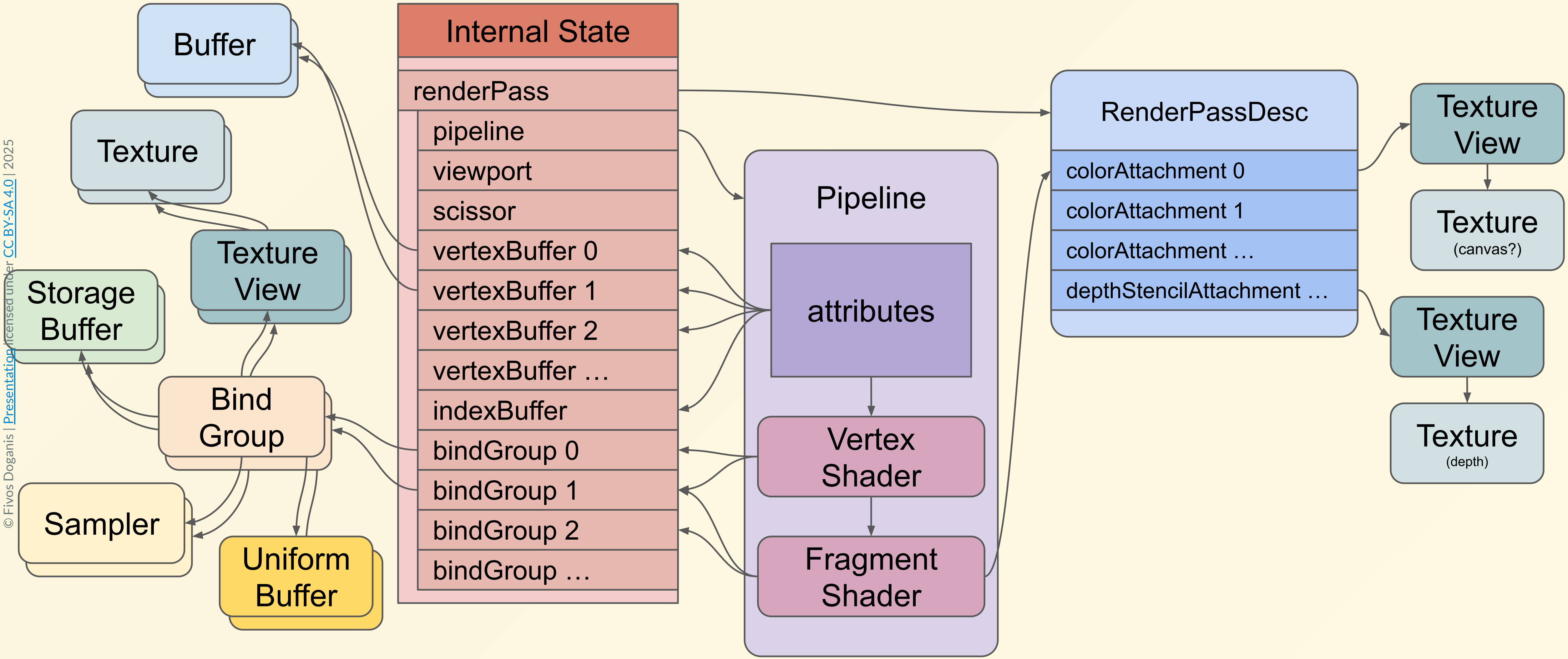
Which common subset to use?

Vulkan Portability TSG Process



WebGL Next == WebGPU ?

- Apple's "WebMetal", API similar to Metal
- API partially reused and renamed **WebGPU**
  - temporary name, API still being defined
  - both **low-level** and **object-oriented** (no global state!)
  - **fast**
  - subset for **web AND native**, despite the "web" in the name
    - a bit like "Vulkan ES" / "Metal ES"
- Might replace WebCL (Compute Shaders), abandoned 😢
- Compatible with WebAssembly



https://webgpufundamentals.org/webgpu/lessons/webgpu-fundamentals.html

All this is so confusing, which API should I actually use ⁉️

Well…

> We hope for **universal availability of WebGL 2.0 soon**. If you need to ship your product **today, WebGL 2.0 is the way to go**. WebGL will be **supported indefinitely**. You do not need to worry about it going away.

> **WebGPU**'s timeline is discussed in the answer to the previous question. WebXR and WebAR are already working on WebGPU integration.

Khronos WebGL meetup, November 18, 2020

https://www.khronos.org/blog/webgl-happenings

> **WebGL 2.0** can now be considered **universally available** across browsers, operating systems and devices.
> As an application author, you can **target WebGL 2.0 with confidence.**
> We encourage you to **migrate to WebGL 2.0**
> It's no longer necessary to maintain a WebGL 1.0 fallback path unless you need to reach absolutely every device.
> In particular, **older Windows machines and Android devices.**

> **WebGPU** standardization continues; conformance testing in high gear
> Aiming to reach 1.0 in 2022 Q2 (spec and conformance tests).

Khronos WebGL + WebGPU meetup, January 25, 2022

https://www.khronos.org/events/webgl-webgpu-meetup-january-2022

Can I use webgl ? Settings

50 results found

WebGL - 3D Canvas graphics - OTHER

Usage % of all users ?

Global 97.94%

Method of generating dynamic 3D graphics using JavaScript, accelerated through hardware

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baic Brow
		2-3.6	4-7	3.1-5	10-11.5										
	12-18	4-23	8-32	5.1-7.1	12.1-18	3.2-7.1									
6-10	79-83	24-78	33-83	8-13	19-68	8-13.3		2.1-4.4.4	12-12.1				4-11.2		
11	84	79	84	13.1	69	13.5	all	81	46	84	68	12.12	12.0	10.4	7.1
		80-81	85-87	14-TP		14.0									

WebGL 1.0 ✅

- available to **98,45%** of users!
  - data from January 2024: https://caniuse.com/webgl
  - even 99.72% according to https://web3dsurvey.com/webgl
- including **mobile** browsers
  - Chrome for Android
  - iOS Safari

➡️ **available everywhere!** 🎉

WebGL 2.0 - OTHER

Next version of WebGL. Based on OpenGL ES 3.0.

Usage % of all users ?

Global 92.3%

Current aligned | Usage relative | Date relative | Filtered | All

Chrome	Edge *	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
			2-24													
			[1 2] 25-41													
4-42		3.1-10	[1] 42-44				3.2-11.4									
[3] 43-55	12-18	[4] 10.1-14.1	[1 5] 45-50	10-42			[4] 12-14.8	4-6.4								
56-102	79-102	15-15.4	51-101	43-85	6-10		15-15.4	7.2-16.0		12-12.1		2.1-4.4.4				
103	103	15.5	102	86	11	103	15.5	17.0	all	64	12.12	103	101	[3] 10.4	7.12	[1 5] 2.5
104-106		15.6-TP	103-104	87			16.0									

WebGL 2.0 ⚠️

- available to **96.34%** of users!
  - data from january 2024: https://caniuse.com/webgl2
- even 97.92% according to https://web3dsurvey.com/webgl2
- Standard at Apple since iOS 15! (september 2021) 🎉

➡️ **official, you can start coding with it!** (check availability)

- retrocompatibility: WebGL 1.0 works in a WebGL 2.0 context
- WebGL 1.0 polyfill to support a subset of WebGL 2.0 (⚠️ shaders)
- you should learn WebGL 1 to understand existing code.

WebGL 2.0 on iOS

➡️ Test WebGL 2 support here:

- https://webglreport.com/?v=2
- https://get.webgl.org/webgl2/

WebGL 2.0 : standardized extensions

```
Depth Textures (WEBGL_depth_texture)
Floating Point Textures (OES_texture_float/OES_texture_float_linear)
Half Floating Point Textures (OES_texture_half_float/OES_texture_half_float_linear)
Vertex Array Objects (OES_vertex_array_object)
Standard Derivatives (OES_standard_derivatives)
Instanced Drawing (ANGLE_instanced_arrays)
UNSIGNED_INT indices (OES_element_index_uint)
Setting gl_FragDepth (EXT_frag_depth)
Blend Equation MIN/MAX (EXT_blend_minmax)
Direct texture LOD access (EXT_shader_texture_lod)
Multiple Draw Buffers (WEBGL_draw_buffers)
Texture access in vertex shaders
```

➡️ when WebGL 2.0 is not supported, we can get close to it using WebGL 1.0 + **these extensions**!

WebGL 1.0.1 ✅

WebGL 1.0.1 == WebL 1.0 + omnipresent extensions

```
ANGLE_instanced_arrays
EXT_blend_minmax
OES_element_index_uint
OES_standard_derivatives
OES_vertex_array_object // use it!
WEBGL_debug_renderer_info
WEBGL_lose_context
```

➡️ **always available, use them!**

WebGL 1.0.2 ✅

WebGL 1.0.2 == WebL 1.0.1 + [omnipresent extensions (since 2021)]

```
EXT_texture_filter_anisotropic
OES_texture_float
OES_texture_float_linear
OES_texture_half_float
OES_texture_half_float_linear
WEBGL_depth_texture
```

➡️ **always available, use them!**

WebGL 1.0.3 ⚠️

WebGL 1.0.3 == WebL 1.0.2 + omnipresent extensions (since 2022):

```
EXT_shader_texture_lod
EXT_sRGB
EXT_frag_depth
```

- but one very useful extension is not supported on Android 😢

```
WEBGL_draw_buffers
```

➡️ **check availability before use**

WebGL 2.0 extensions ⚠️

WebGL 2.0 omnipresent extensions since 2022:

```
EXT_texture_filter_anisotropic
OES_texture_float_linear
WEBGL_debug_renderer_info
WEBGL_lose_context
```

- but one fails on Safari ❌ (EXT_float_blend ?): problem for **GPGPU**

```
EXT_color_buffer_float
```

➡️ **check availability before use**

⚠️ Available != no bugs

> “ **#WebGL2** is a rubbish job on **#IOS14**. On Ipad pro more than half of the conformance tests fail (153553 over 260803 - tested here: https://khronos.org/registry/webgl/sdk/tests/webgl-conformance-tests.html). For GPGPU I still use WebGL1 for IOS devices. ”

@xavierbourry, July 2020

WebGPU (reminders)

- low-level API, fast, promising, introduced by Apple
- close to **Metal**, Vulkan and DirectX 12
  - read Metal docs to understand the concepts
- new shader language: **WGSL**
  - text format, gets compiled as SPIR-V (Vulkan)
- **version 1.0 released on Chrome in april 2023**
- official W3C spec, demos, minimalistic code sample

➡️ probably the future Web 3D API, **keep an eye on it!** 👀

Conclusion: which API should I use?

- 1️⃣ Start with **WebGL 1.0**, many exampled (13-year-old API!)
  - to understand the ***concepts***
    - state machine, pipeline, buffers, shaders
    - see OpenGL course!
  - and **how WebGL interacts with a web page**
    - see HTML / JavaScript / CSS course
- 2️⃣ Check new features introduced in **WebGL 2.0**
- 3️⃣ Use high-level APIs: **THREE.js**, Babylon, A-Frame etc.
  - for a smooth transition to WebGPU!

WebGL

Concepts

Concepts > Syntax

- APIs evolve
- GPUs change too
- But all modern APIs and GPUs have **a lot in common**
- **Common, transposable concepts** are more important than **syntax** and APIs

➡️ **understand how GPUs work** for **maximum performance**

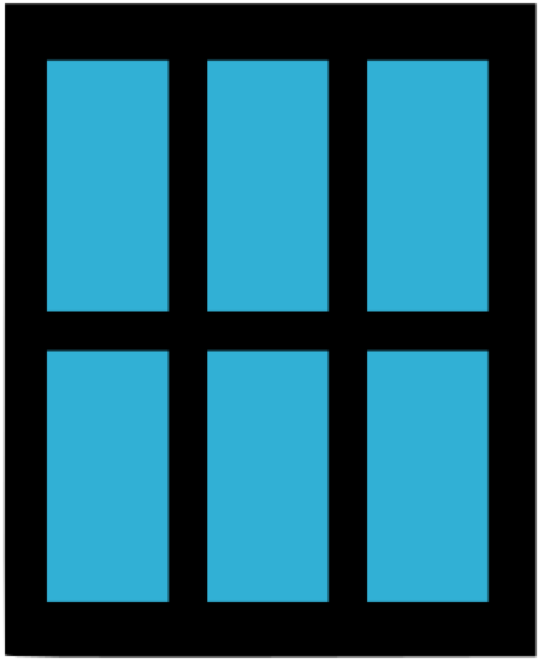
➡️ **transpose your knowledge easily** to other APIs, OSes or architectures

CPU vs GPU

Reminders

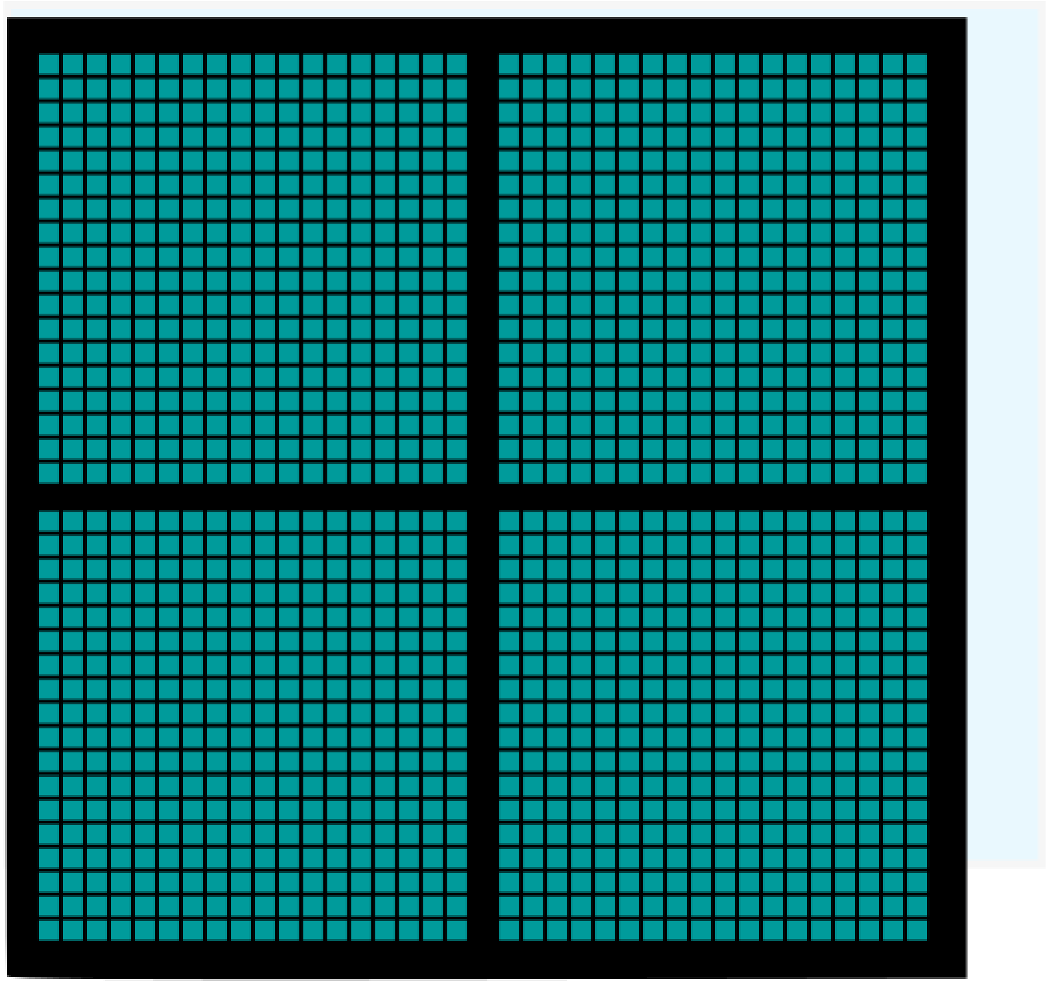
> “ *There's a freaking supercomputer in your browser, and nobody seems to have noticed!* ”

[Steve Sanderson]



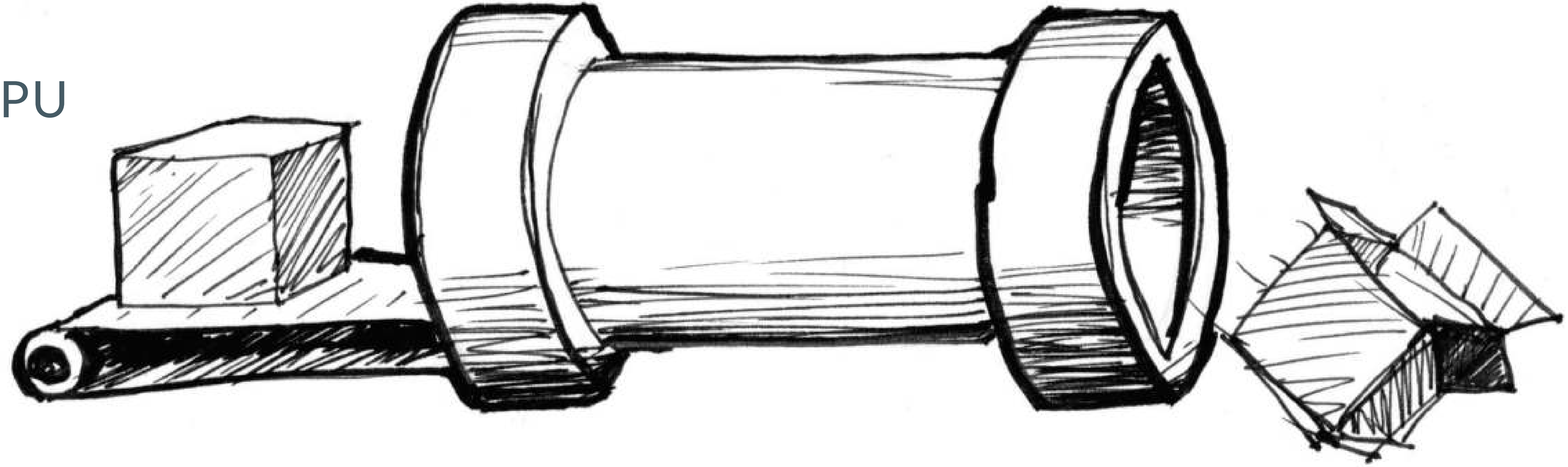
CPU
Multiple Cores

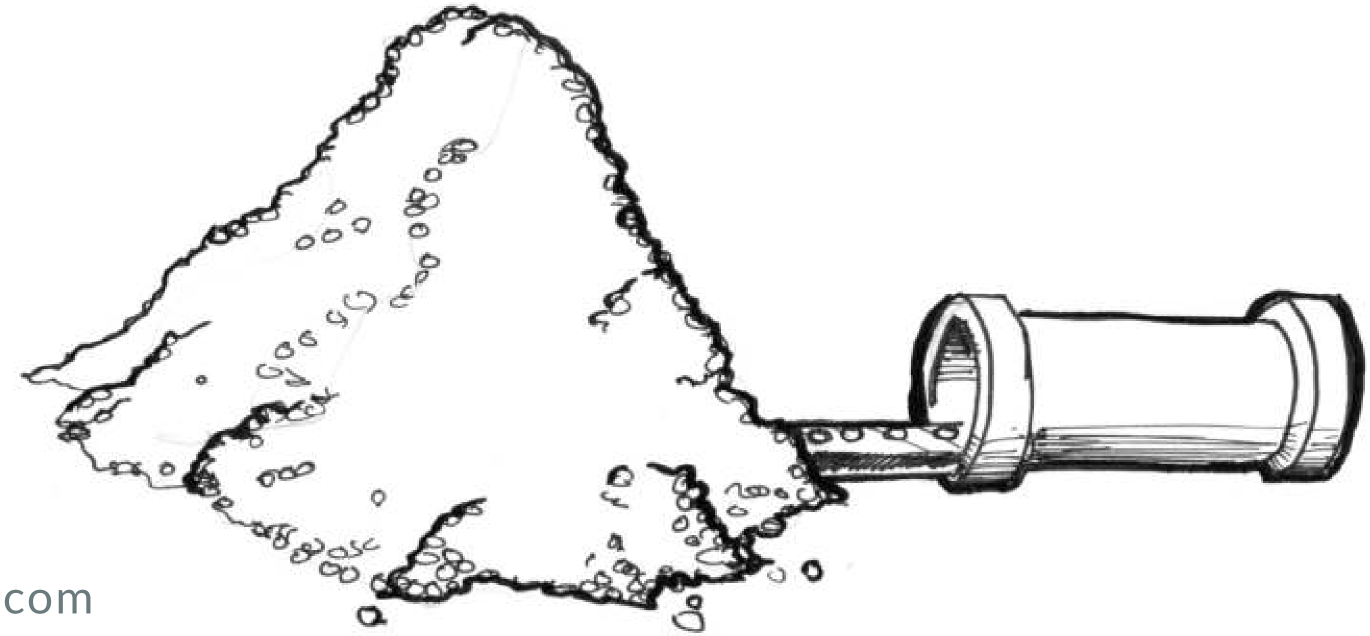
+



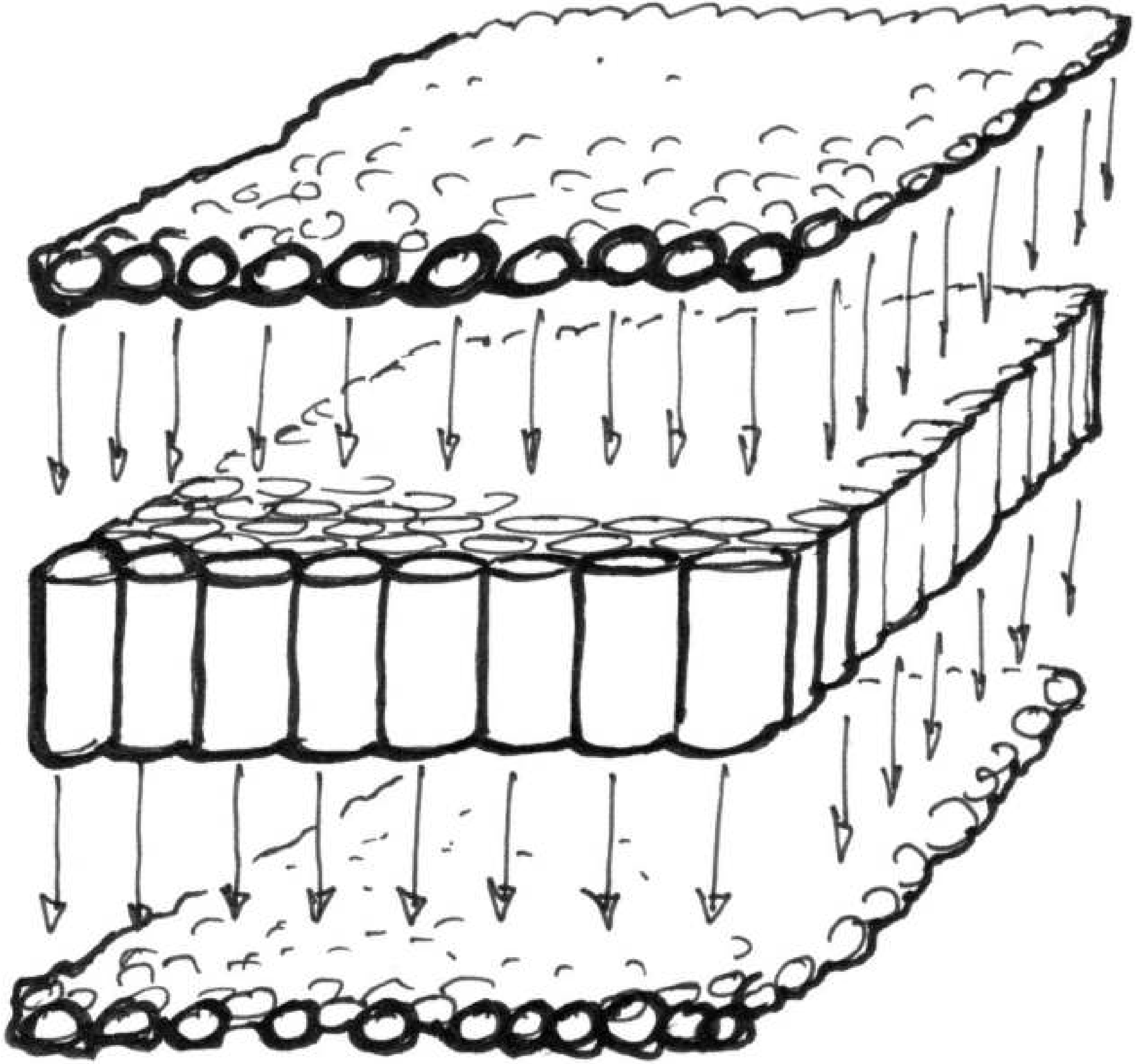
GPU
Thousands of Cores

CPU





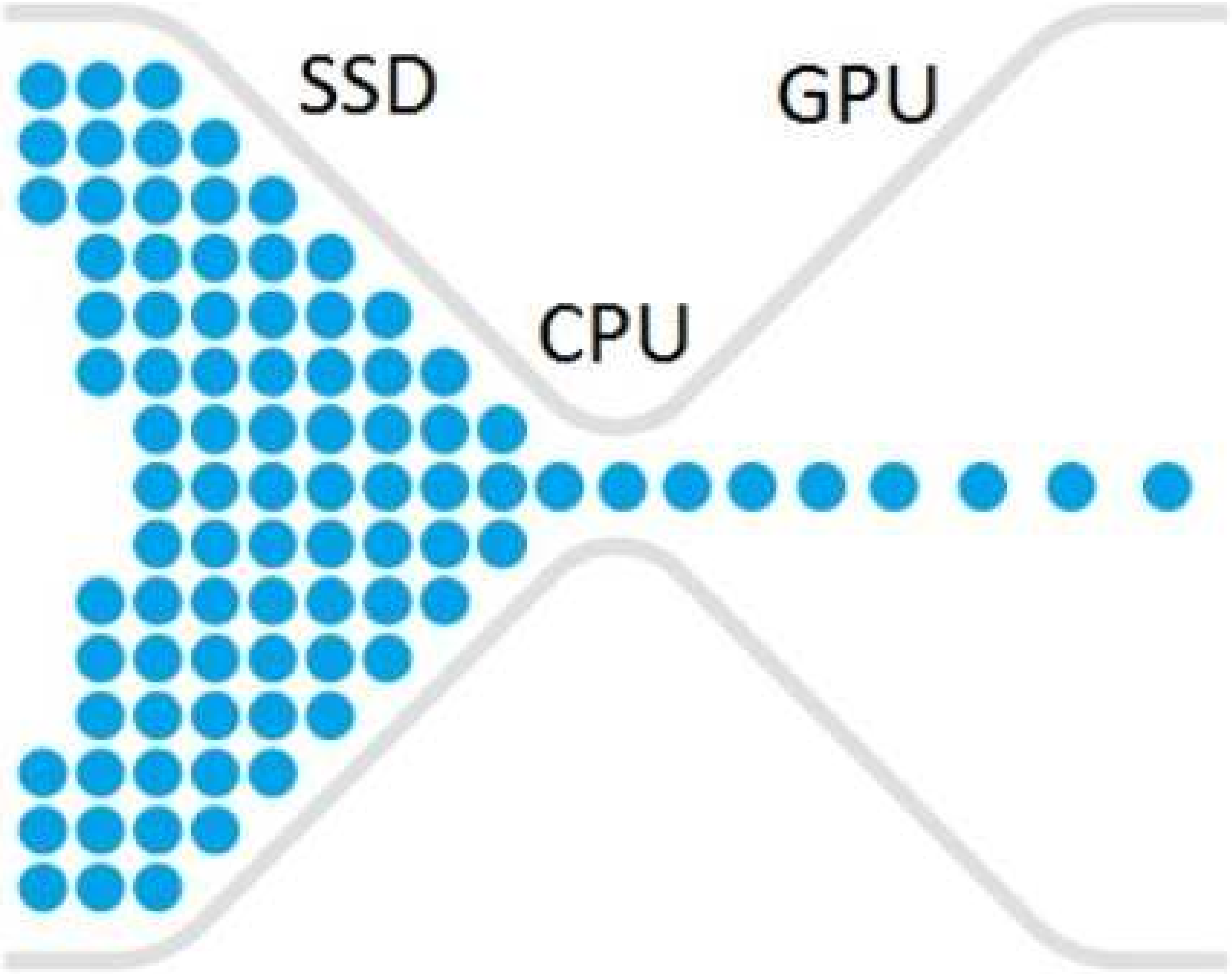
GPU



Goal

Send **as much data as possible to the GPU**, for **fast** processing

- "**upload**" (CPU ➡️ GPU) is **slow** 🐢
  - group data into **buffers** before transfer
- **GPU processing is very fast**
  - using **shaders**
    - working in **parallel**
    - **simple** instructions
    - **compiled** in native low-level GPU code


SSD
GPU
CPU

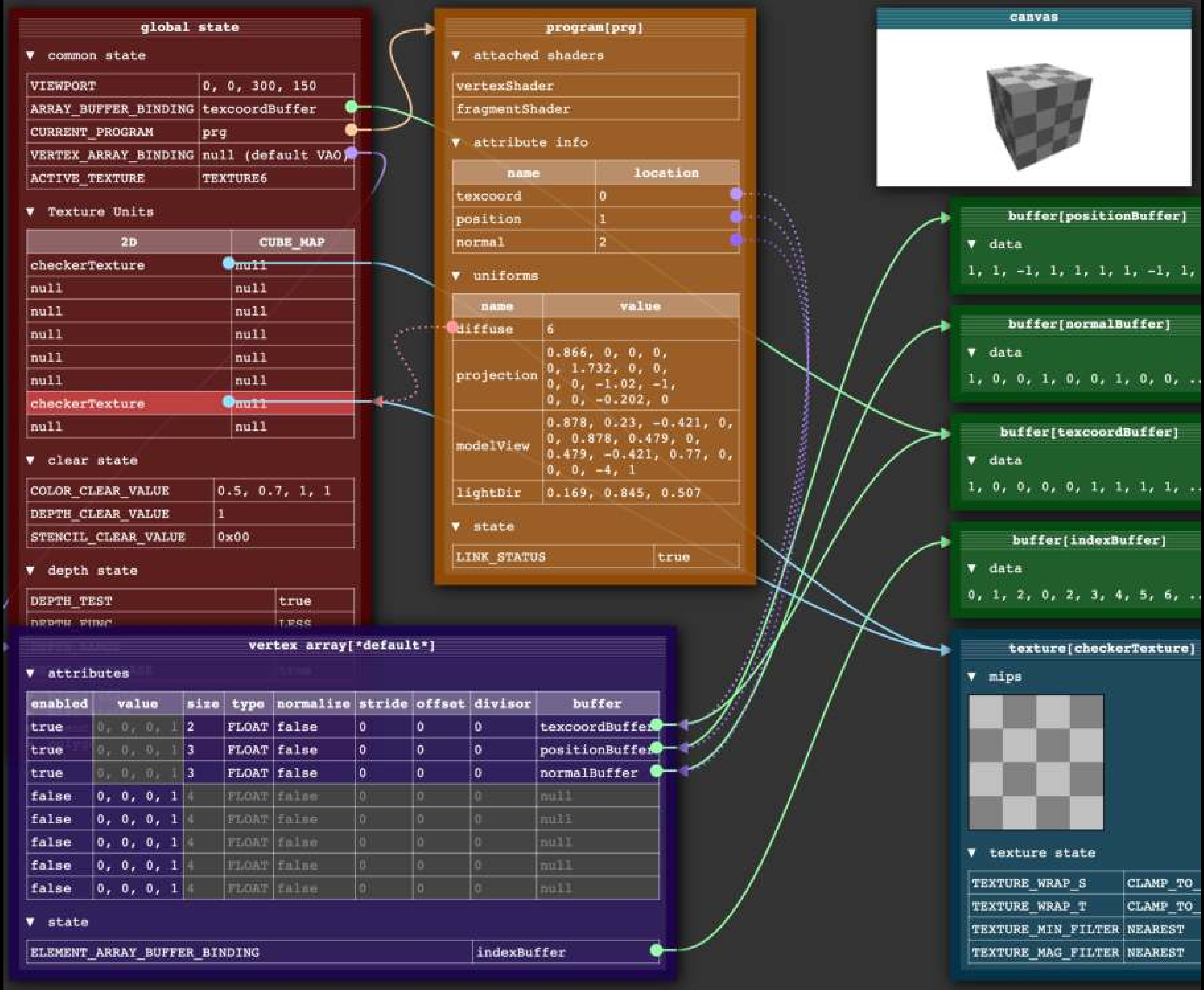
Constraints

- Rendering is fast but "**download**" (CPU ⬅️ GPU) ***VERY* slow** 🐌
- **Buffers are not flexible** for **dynamic** data
- **Arrays** must be converted to **textures** (for **GPGPU**)
  - **conversion** takes time especially for dynamic data
  - possible loss of **accuracy**
- **Shaders are complex** to write
  - pixels are **isolated**, processed in parallel, independently
  - instructions are **limited**
  - **optimizing** and **debugging** is not trivial!

CPU

OpenGL, is a **state machine**

Reminders

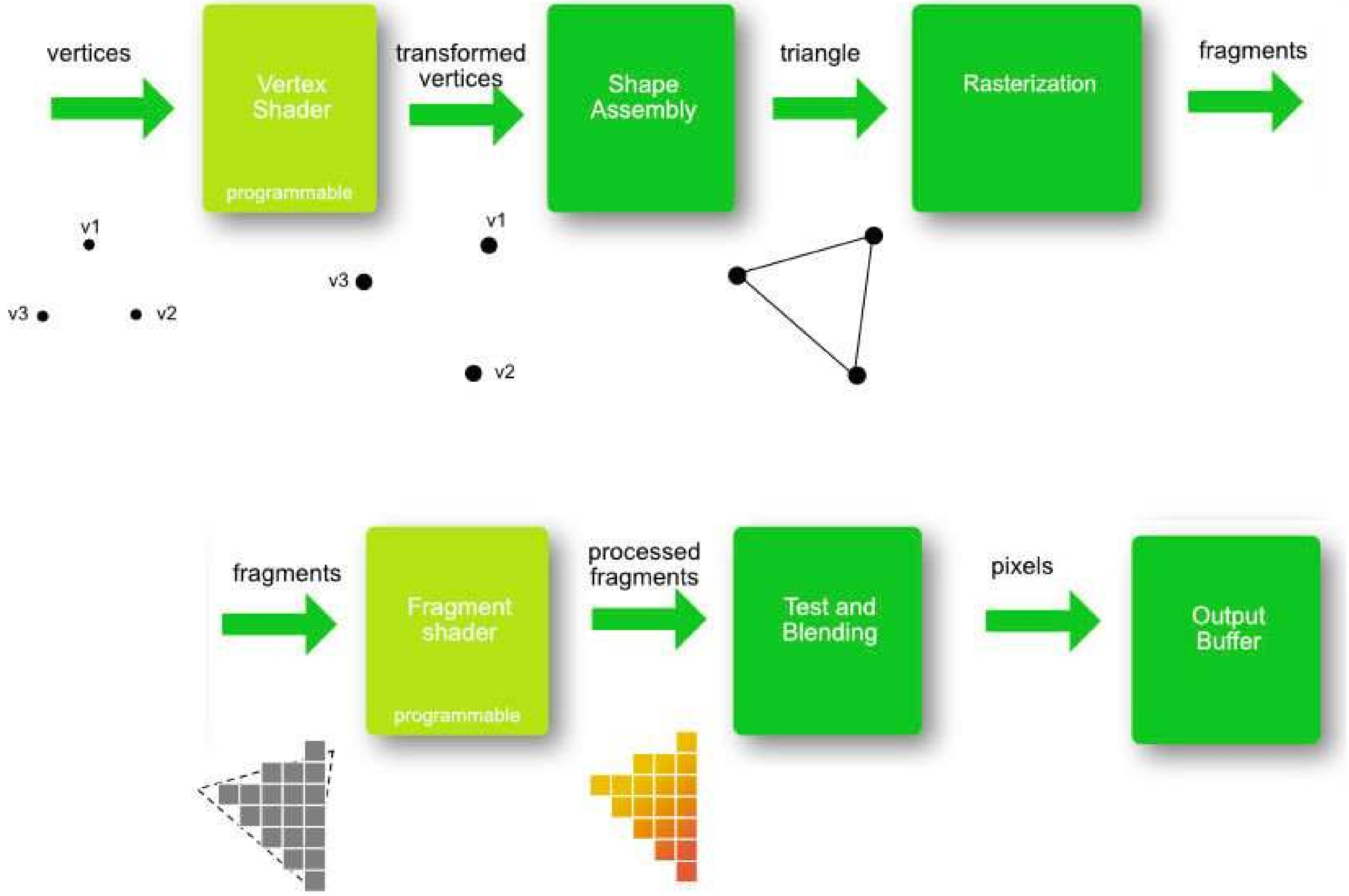

global state
common state
VIEWPORT 0, 0, 300, 150
ARRAY_BUFFER_BINDING texcoordBuffer
CURRENT_PROGRAM prg
VERTEX_ARRAY_BINDING null (default VAO)
ACTIVE_TEXTURE TEXTURE6
Texture Units
2D CUBE_MAP
checkerTexture null
null null
null null
null null
null null
null null
checkerTexture null
null null
clear state
COLOR_CLEAR_VALUE 0.5, 0.7, 1, 1
DEPTH_CLEAR_VALUE 1
STENCIL_CLEAR_VALUE 0x00
depth state
DEPTH_TEST true
program[prg]
attached shaders
vertexShader
fragmentShader
attribute info
name location
texcoord 0
position 1
normal 2
uniforms
name value
diffuse 6
projection 0.866, 0, 0, 0, 0, 1.732, 0, 0, 0, 0, -1.02, -1, 0, 0, -0.202, 0
modelView 0.878, 0.23, -0.421, 0, 0, 0.878, 0.479, 0, 0.479, -0.421, 0.77, 0, 0, 0, -4, 1
lightDir 0.169, 0.845, 0.507
state
LINK_STATUS true
canvas
buffer[positionBuffer]
data
1, 1, -1, 1, 1, 1, 1, -1, 1,
buffer[normalBuffer]
data
1, 0, 0, 1, 0, 0, 1, 0, 0, ..
buffer[texcoordBuffer]
data
1, 0, 0, 0, 0, 1, 1, 1, 1, ..
buffer[indexBuffer]
data
0, 1, 2, 0, 2, 3, 4, 5, 6, ..
texture[checkerTexture]
mips
texture state
TEXTURE_WRAP_S CLAMP_TO_
TEXTURE_WRAP_T CLAMP_TO_
TEXTURE_MIN_FILTER NEAREST
TEXTURE_MAG_FILTER NEAREST
vertex array[*default*]
attributes
enabled value size type normalize stride offset divisor buffer
true 0, 0, 0, 1 2 FLOAT false 0 0 0 texcoordBuffer
true 0, 0, 0, 1 3 FLOAT false 0 0 0 positionBuffer
true 0, 0, 0, 1 3 FLOAT false 0 0 0 normalBuffer
false 0, 0, 0, 1 4 FLOAT false 0 0 0 null
false 0, 0, 0, 1 4 FLOAT false 0 0 0 null
false 0, 0, 0, 1 4 FLOAT false 0 0 0 null
false 0, 0, 0, 1 4 FLOAT false 0 0 0 null
false 0, 0, 0, 1 4 FLOAT false 0 0 0 null
state
ELEMENT_ARRAY_BUFFER_BINDING indexBuffer

WebGL is a state machine too!

- **data** preparation
  - format, type, buffers...
- **global state** preparation
  - color, blending...
- rendering
  - **send to GPU** for **shader** processing

OpenGL Pipeline

Reminders


vertices
Vertex Shader
programmable
transformed vertices
Shape Assembly
triangle
Rasterization
fragments
v1
v3
v2
v1
v3
v2
fragments
Fragment shader
programmable
processed fragments
Test and Blending
pixels
Output Buffer

Rasterization

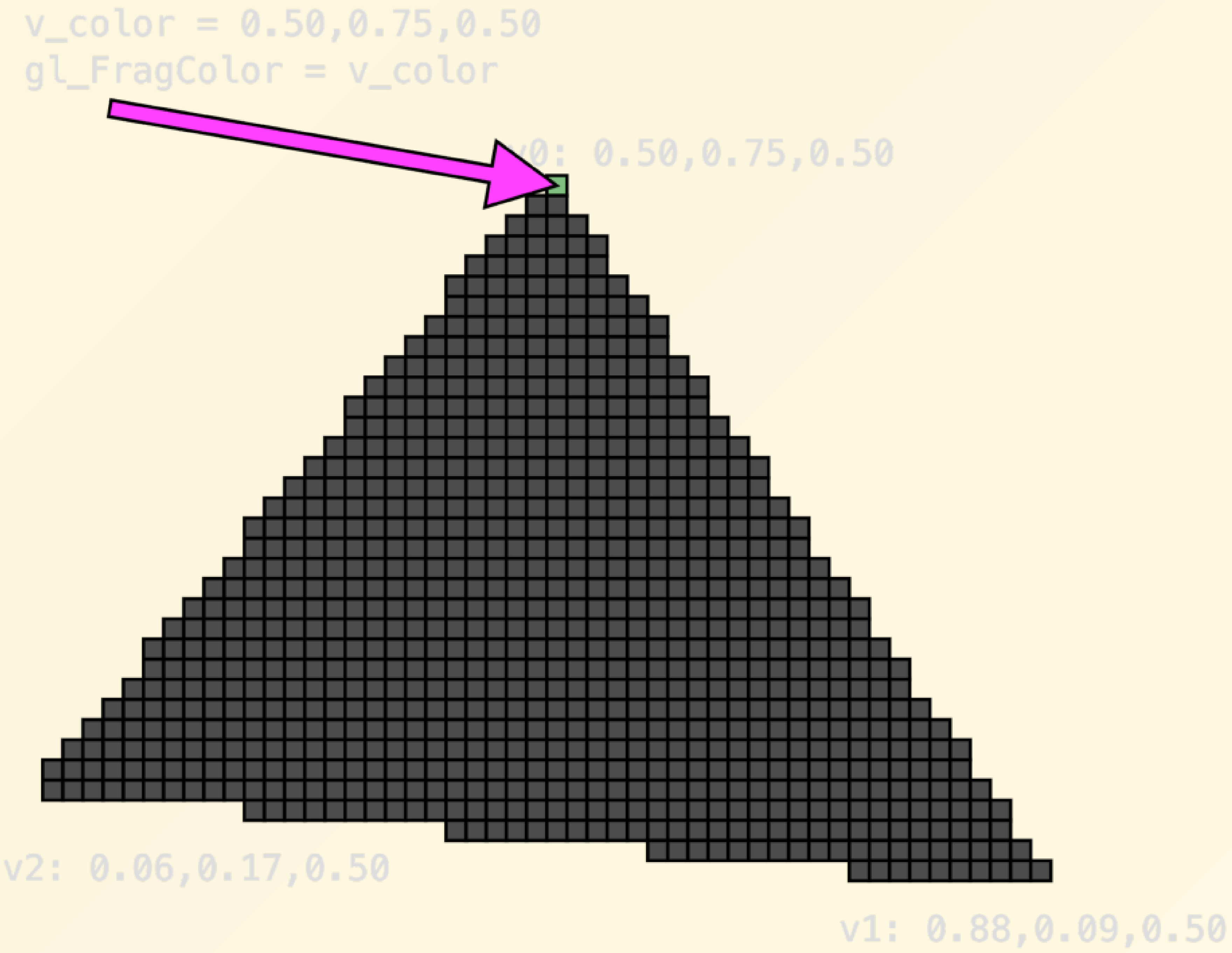
Interactive illustration

WebGL Fundamentals

by

Gregg Tavares (@greggman)
Chrome WebGL implementor

https://webglfundamentals.org/webgl/lessons/resources/fragment-shader-anim.html



v_color = 0.50,0.75,0.50
gl_FragColor = v_color
v0: 0.50,0.75,0.50
v2: 0.06,0.17,0.50
v1: 0.88,0.09,0.50

Shaders are essential

They allow to unleash the power of the GPU

- Shader code is **fast**
  - **thousands of specialized cores** inside a GPU!
  - once the data and the compiled code have been sent to the GPU, the performance is the same regardless of thelanguage or API
- Rendering is **flexible**
  - the rendering pipeline was fixed, not programmable before 2001
  - "rendering" has been hijacked to perform fast parallel physics and machine learning computations on the GPU (**GPGPU**)

Shader programming steps

- the application sends to the GPU:
  - **buffers** (vertices, normals, connectivity info…) and **textures**
  - **shaders** to compile and run
- **vertex shaders are called once per vertex** ⭐
- each primitive (point, line, triangle) is converted to fragments(*rasterization*)
- ~~pixel~~ **fragment shaders are called once per fragment** ⭐
  - their inputs (color, depth, normal) have been previously **interpolated** using the points defining the primitive!

From the triangle to the pixel

Interactive illustration

Making WebGL Dance

by

Steven Wittens

https://acko.net/files/fullfrontal/fullfrontal/webglmath/shaders.html

Note: these slides use cutting edge CSS 3 and WebGL features. It is recommended to use Google Chrome to view them.

The Rise Of The

Shaders

GLSL: Shader Programming Language

- **variable types** ⭐
  - **uniform**: **input**, sent by the app, **constant** in the shader code
  - **attribute**: **input** of the vertex shader, sent by the app: **data of the vertex buffer**, **varies per vertex**
  - **varying**: **output** of the vertex shader / **input** of the fragment shader
- **functions**: C language dialect
- GLSL for WebGL 1.0, cf [page 3]
- GLSL pour WebGL 2.0: version OpenGL ES 3.0, cf [page 8]

WebGL

Let's code!

BREAK

30' ⌛

Appendices

- WebGL History

https://web.eecs.umich.edu/~sugih/courses/eecs487/lectures/20-History+ES+WebGL.pdf

- WebGL 2 Course

https://perso.univ-rennes1.fr/pierre.nerzic/IAI2/IMR2 - Synthèse d'images - CM2.pdf

- Tools for analyzing, debugging, checking and dumping WebGL

https://github.com/greggman/webgl-helpers#webgl-gl-error-checkjs