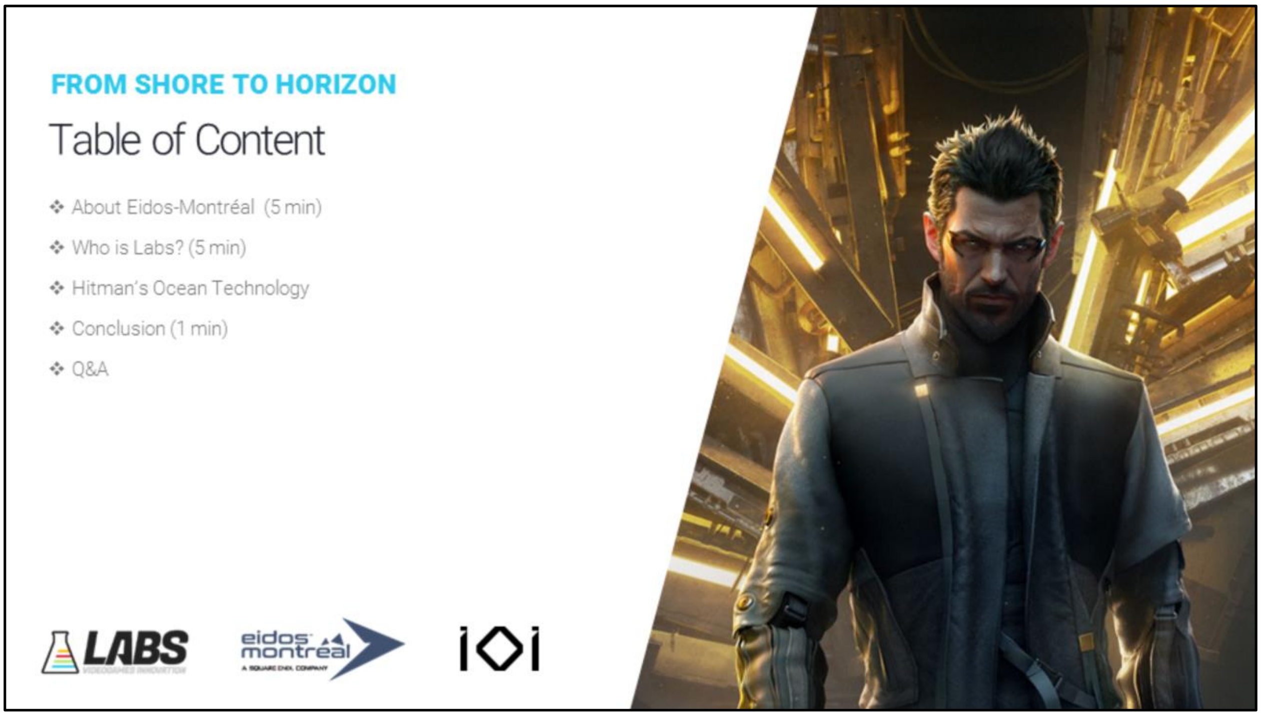
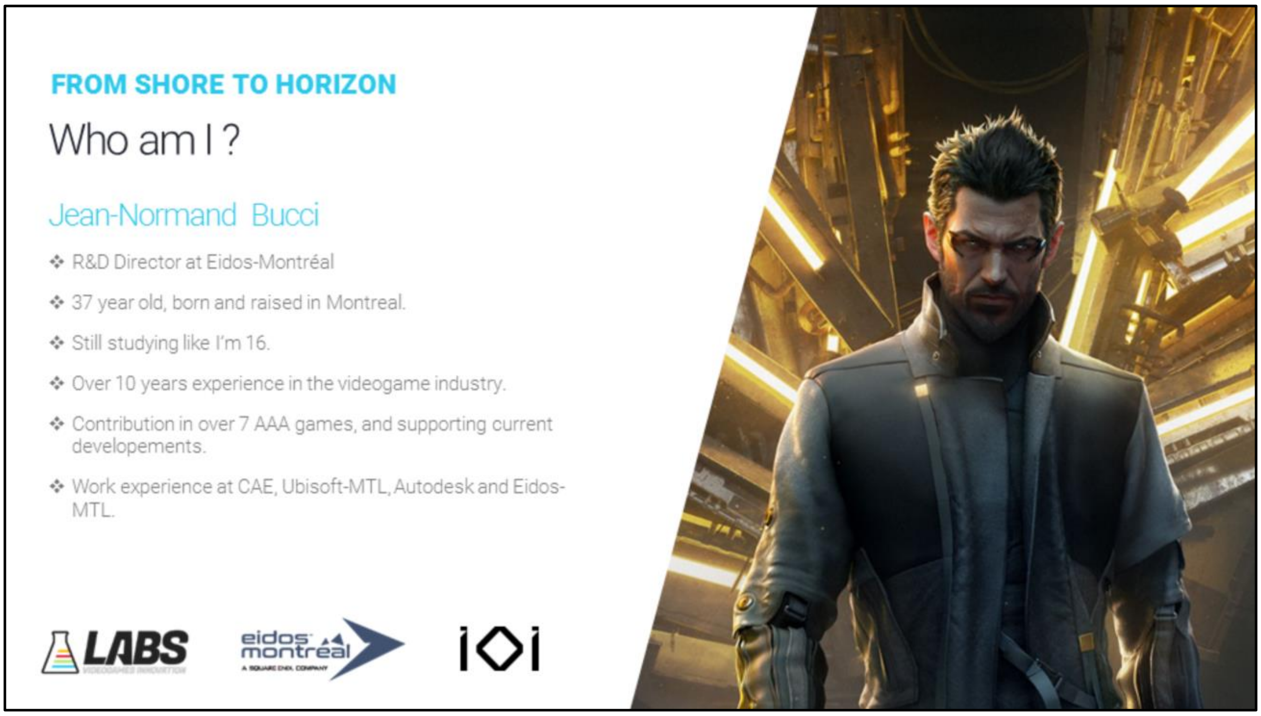
LABS Logo Reveal



FROM SHORE TO HORIZON

Table of Content

Here is the agenda for today!
Let me explain you who we are to better understand our process when attacking a new tech.



FROM SHORE TO HORIZON

Who am I ?

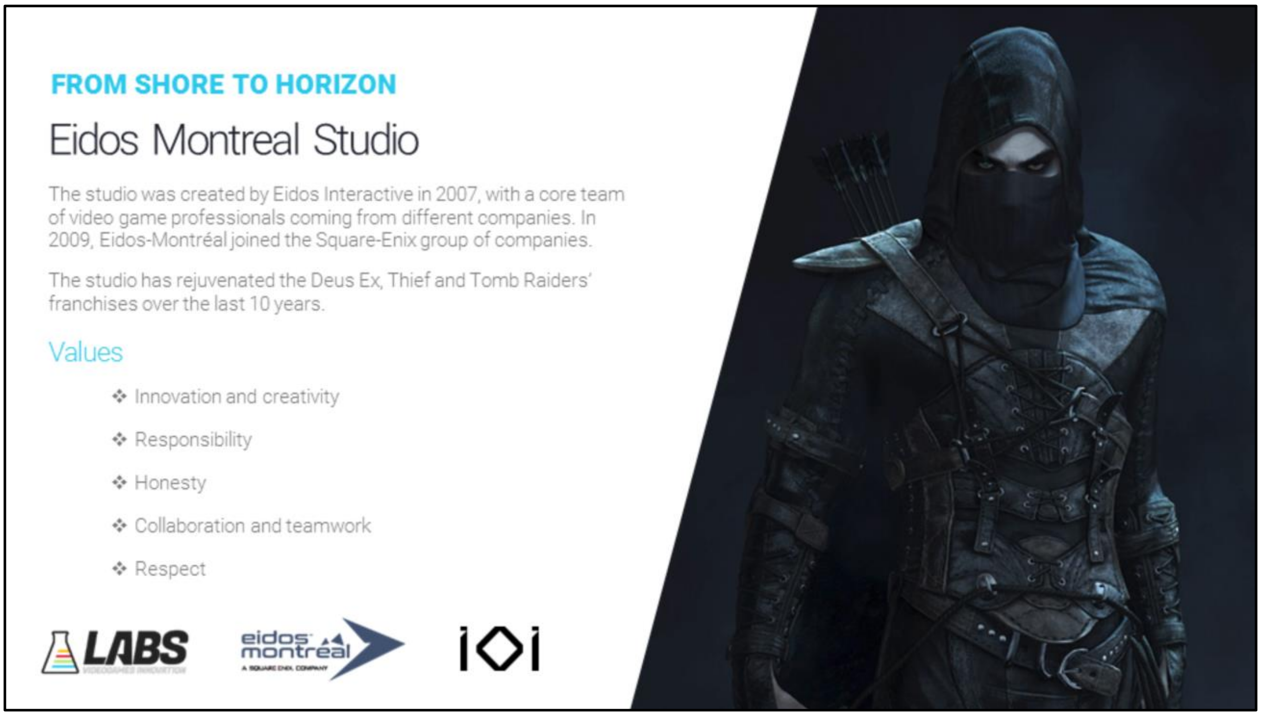
Jean-Normand Bucci

- R&D Director at Eidos-Montréal
- 37 year old, born and raised in Montreal.
- Still studying like I'm 16.
- Over 10 years experience in the videogame industry.
- Contribution in over 7 AAA games, and supporting current developements.
- Work experience at CAE, Ubisoft-MTL, Autodesk and Eidos-MTL.

Hi,
Glad to be here with you all, thanks for choosing us as your kickoff conference of the day
I know it's **early** and I'll try to **keep you all awake until the interesting stuff kicks in**! ☺

I guess this **funnels me to my intro** : **37 yo**, **proud father of two** including a **18 months** old baby boy.
This might only **resonate** to the **parents of young kids in the audience**, but it's a relaxing moments right now with you all! ☺
Born and **raised in Montreal**, by an **Italian father** and a **French Canadian mother**, which **explains** the **weird name** combination there.
I spent **ten years in gaming** specifically, **coming from aerospace** industry. Those ten years where mostly **spent in mainly two studios**.

[CLICK]



FROM SHORE TO HORIZON

Eidos Montreal Studio

The studio was created by Eidos Interactive in 2007, with a core team of video game professionals coming from different companies. In 2009, Eidos-Montréal joined the Square-Enix group of companies.

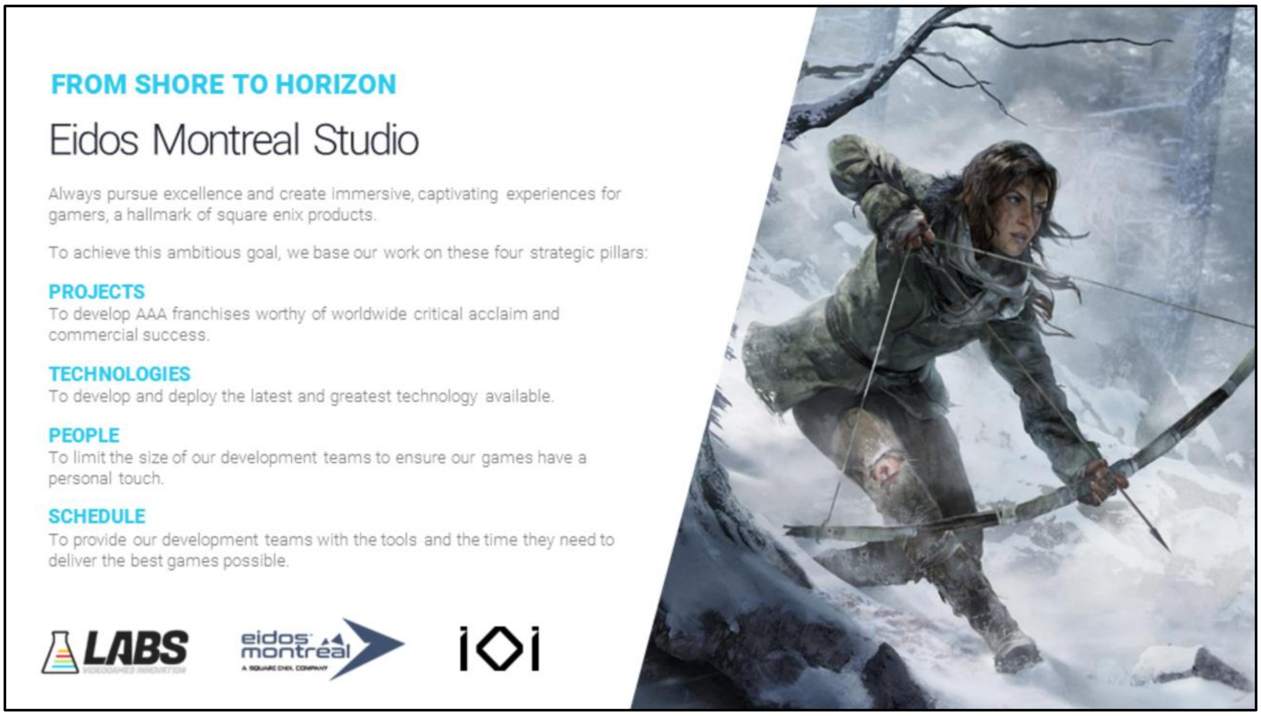
The studio has rejuvenated the Deus Ex, Thief and Tomb Raiders' franchises over the last 10 years.

Values

- ❖ Innovation and creativity
- ❖ Responsibility
- ❖ Honesty
- ❖ Collaboration and teamwork
- ❖ Respect

Apologize in advance as I **extracted** that information from a **corporate deck**.
But let me draw your attention on some of those facts.
Eidos-Montreal moved into its early teenage years, celebrating its 10 years anniversary not too long ago.
Over those years, in which I was **an active participant for over 7** now, the studio built itself a **strong reputation in Montreal** and around the globe.
Strong installments of **beloved franchise** such as **Deus Ex, Thief and Tomb Raider** that helped accomplished this positioning.
The **points here and in the next slides**, that **drives** the most of my **attention** are, of course, **innovation and creativity**, but **also honesty and respect**!

[CLICK]



FROM SHORE TO HORIZON

Eidos Montreal Studio

Always pursue excellence and create immersive, captivating experiences for gamers, a hallmark of square enix products.

To achieve this ambitious goal, we base our work on these four strategic pillars:

PROJECTS
To develop AAA franchises worthy of worldwide critical acclaim and commercial success.

TECHNOLOGIES
To develop and deploy the latest and greatest technology available.

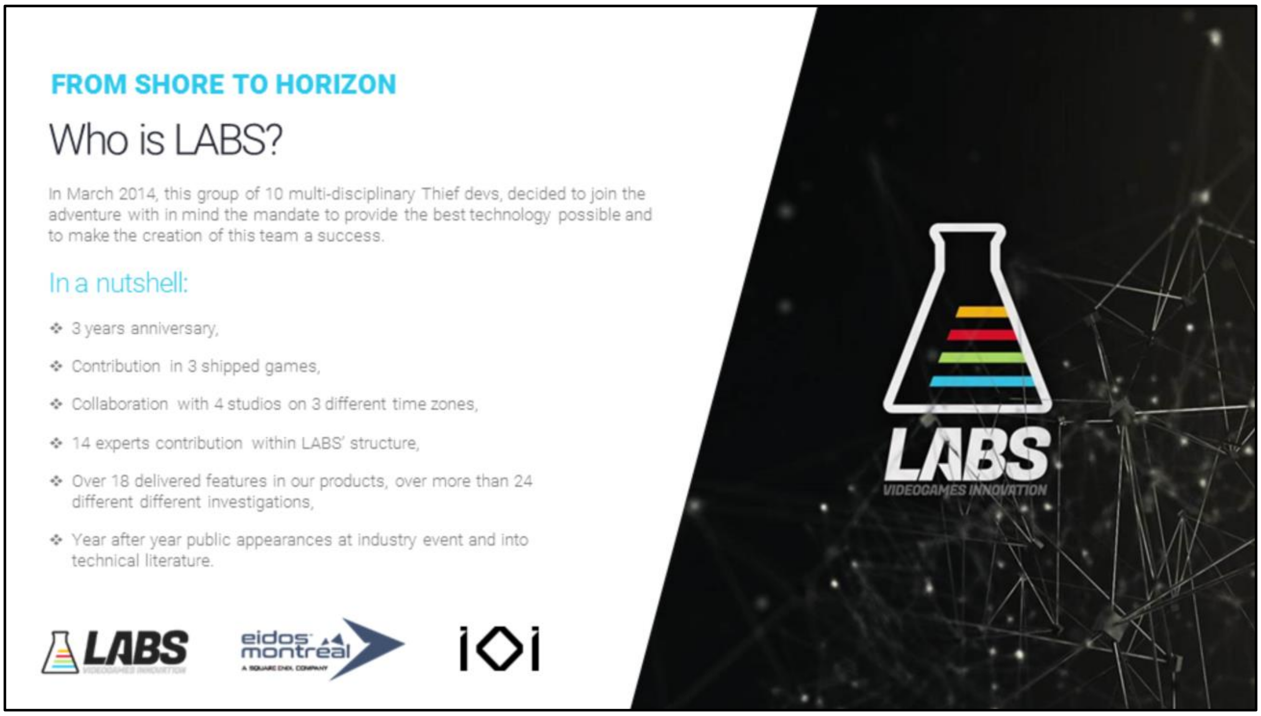
PEOPLE
To limit the size of our development teams to ensure our games have a personal touch.

SCHEDULE
To provide our development teams with the tools and the time they need to deliver the best games possible.

The studio is **creating great titles** from **great IP**, but all of this **done through an enjoyable environment**.
A **respectful approach towards its most valuable asset: its employees**
It's done through great overtime and **crunch management policies** for instance, or **strong balance between work and family** .
Amy Hennings wrote an interresting paper in **GameIndustry.biz a few months back**, which I recommend you if you haven't read it yet.

Let say this straight, **the active developper's community is growing older** and we **have changing expectaction** towards our employers.
Which is fine!
We've seen this trend in **other studio too**, **kindergarden in the office**, **access to doctors**, etc…
We're like a **big familly** and our **returning employees' rate is a great indication**.

[CLICK]


FROM SHORE TO HORIZON

Who is LABS?

In March 2014, this group of 10 multi-disciplinary Thief devs, decided to join the adventure with in mind the mandate to provide the best technology possible and to make the creation of this team a success.

In a nutshell:

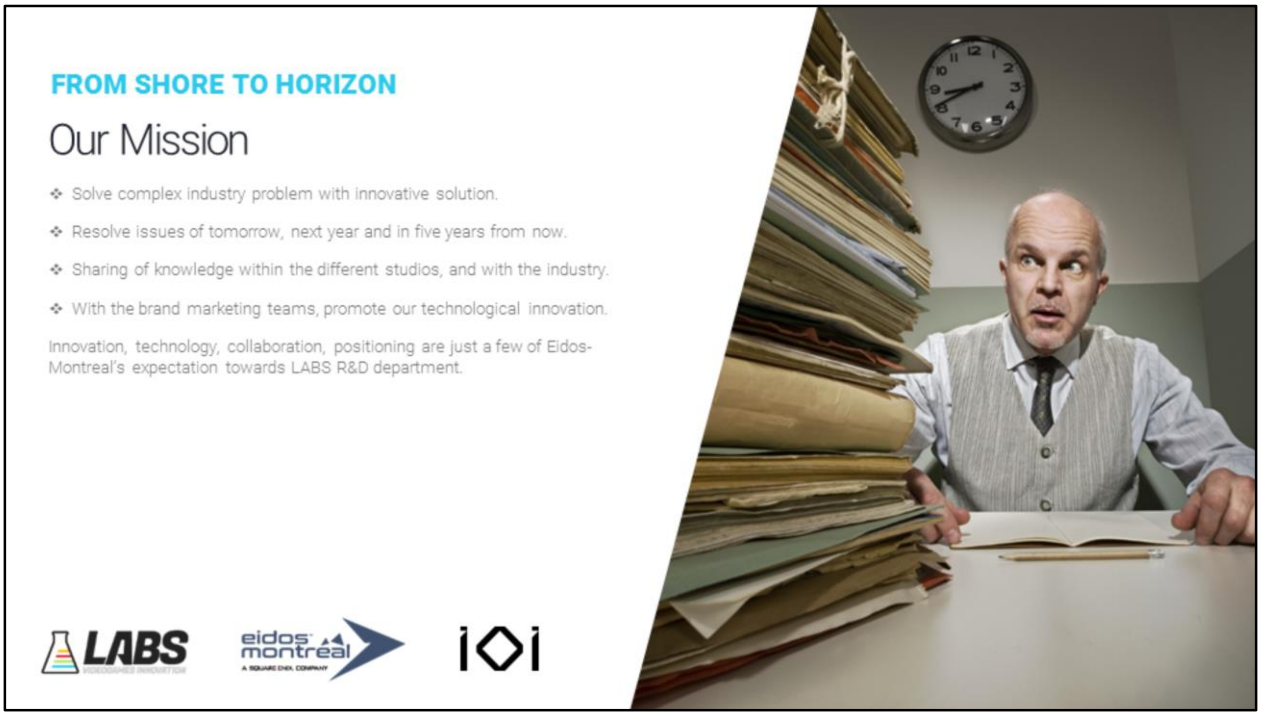
- 3 years anniversary,
- Contribution in 3 shipped games,
- Collaboration with 4 studios on 3 different time zones,
- 14 experts contribution within LABS' structure,
- Over 18 delivered features in our products, over more than 24 different different investigations,
- Year after year public appearances at industry event and into technical literature.

So moving on to **Who is LABS.**
Well, it all **started** when our **Head of Office** regrouped a **bunch of released resources** to **push the technology barrier** and to **act as a single unit**.
We were **folks from different fields of expertise**, who **strongly collaborated together** in the past and **knew each other's strenghts**.
We've had **fighting** moments, **stress** moment, **tears** and all that we **experienced in our respective game teams**, right?

Then, the rest of the numbers are there:

-SLIDE-

[CLICK]



FROM SHORE TO HORIZON

Our Mission

- Solve complex industry problem with innovative solution.
- Resolve issues of tomorrow, next year and in five years from now.
- Sharing of knowledge within the different studios, and with the industry.
- With the brand marketing teams, promote our technological innovation.

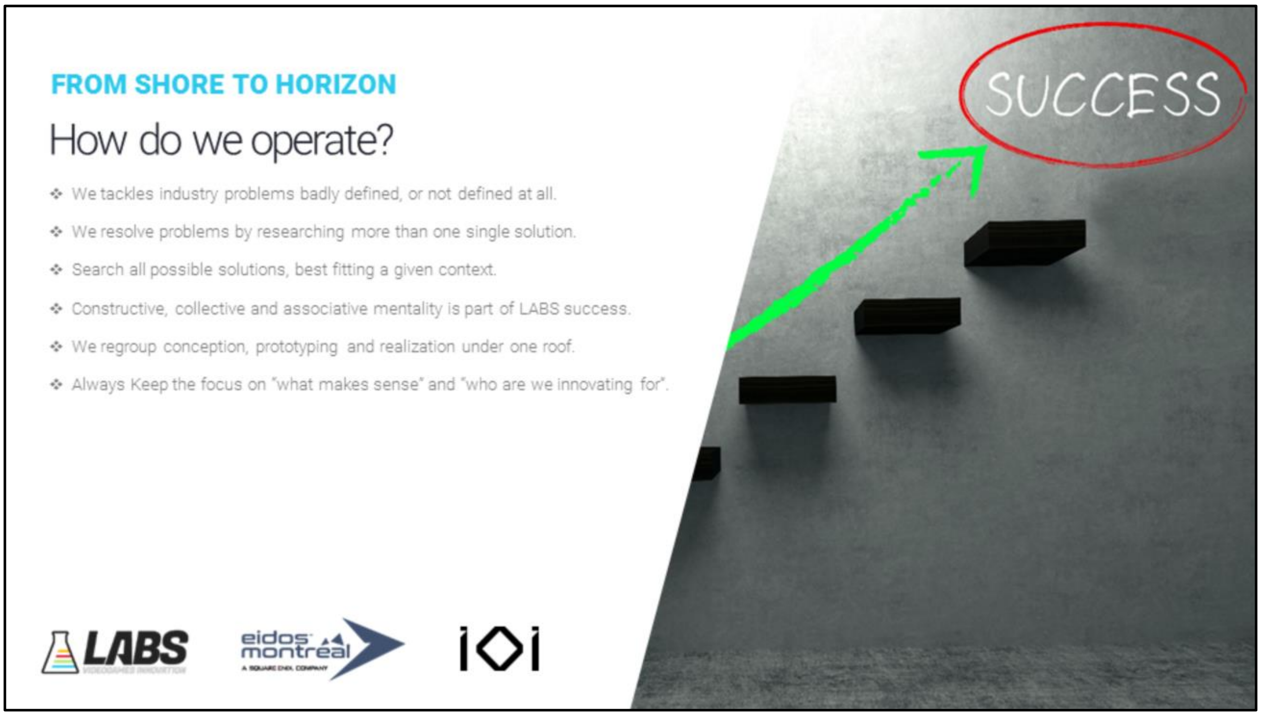
Innovation, technology, collaboration, positioning are just a few of Eidos-Montreal's expectation towards LABS R&D department.

What are the **expectation** for LABS, well **pretty straight forward** and easy to understand… **not so easy** to put into application.

Solve complex industry problems, or at least grow our understanding of those. It's limitations, workaround
There are problems we are all struggling with, independently at which studio you work.

Attempting to **address issues that is an actual problem** we are suffering with but **also being aware of problems to come**.
My team try **investing** the right amount of time **avoiding this to ever become a production showstopper**.

Then **sharing**!
Sharing the **knowledge** gathered or even **proven solution** with sisters' **studio** to **help positioning our products**.



FROM SHORE TO HORIZON

How do we operate?

- We tackles industry problems badly defined, or not defined at all.
- We resolve problems by researching more than one single solution.
- Search all possible solutions, best fitting a given context.
- Constructive, collective and associative mentality is part of LABS success.
- We regroup conception, prototyping and realization under one roof.
- Always Keep the focus on "what makes sense" and "who are we innovating for".

#1 - Industry problems

Some problems are well documented. Multiple solutions were proposed by expert and big brains already

But others are totally new issues that we need to investigate and build knowledge about it, a better understanding of it.

#2 - Researching different solution.

Aiming at resolving a problem with the first solution that emerges, could show that the solution is not that strong on the long run.

#3 - Search all possible solutions, best fitting a given context.

#4 - Constructive, **collective** and **associative mentality** is part of LABS success.

The collaboration is important in my team.

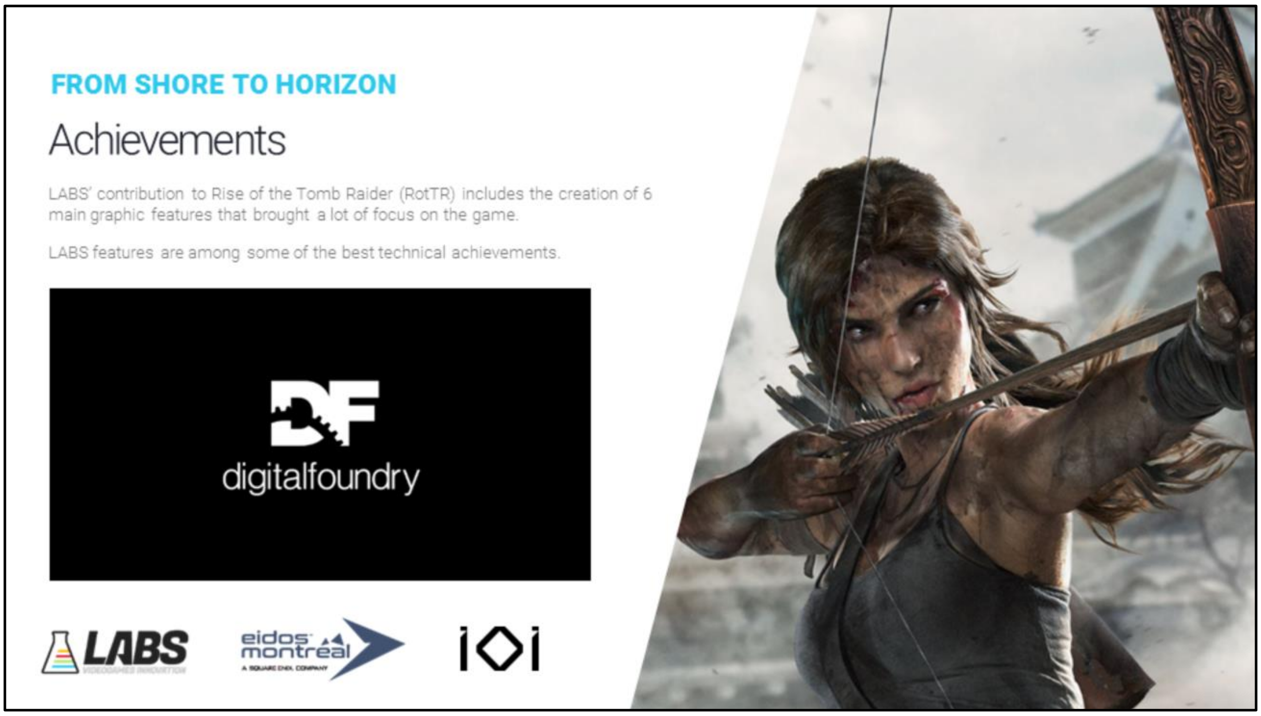
#5 - We regroup conception, prototyping and implementation under one roof.

The idea is to prove its value, quickly. Fail fast and often to find a strong winner.

#6 - “***what makes sense***” and “***who are we innovating for***”.

Keep the focus on the clients’ need and not try to solve some personal fantasy while at it.

Let me **guide you through the team achievements**, done in only **3 years** of existence.
First Rise of the Tomb Raider



FROM SHORE TO HORIZON

Achievements

LABS' contribution to Rise of the Tomb Raider (RotTR) includes the creation of 6 main graphic features that brought a lot of focus on the game.

LABS features are among some of the best technical achievements.

LABS' contribution in **Rise of the Tomb Raider includes** the creation of **6 main graphic features** that brought a lot of focus on the game.

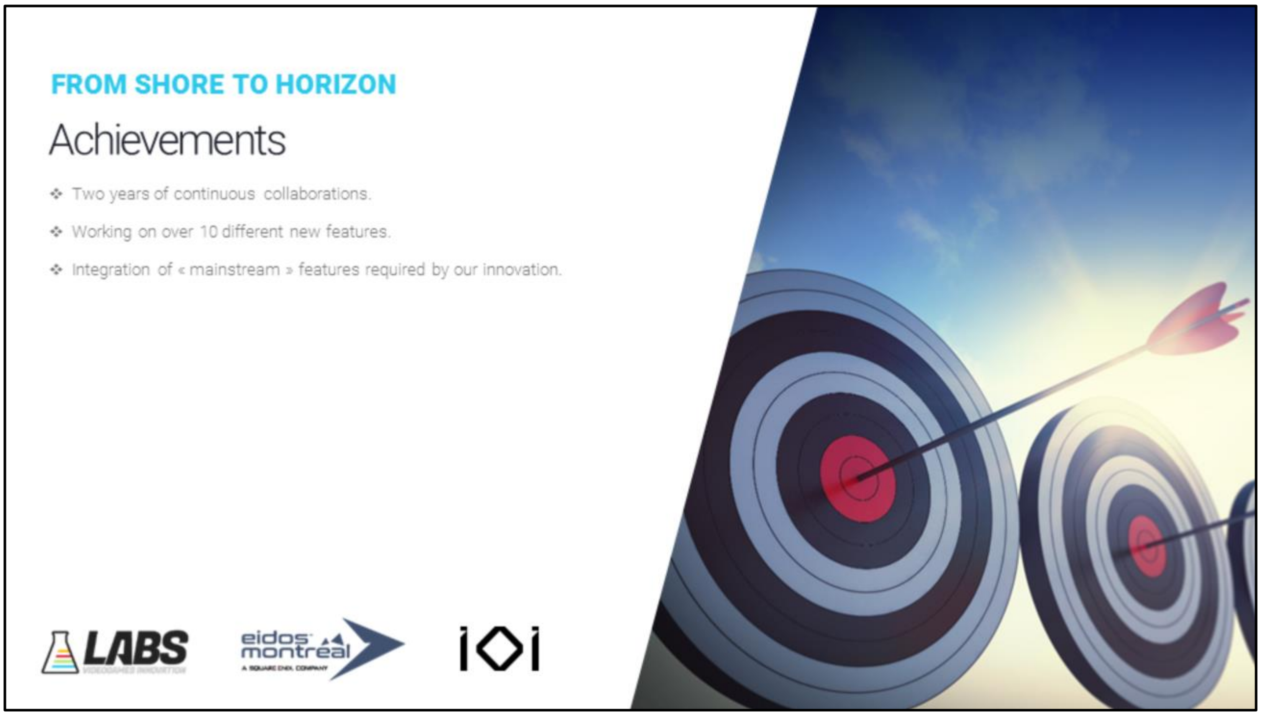
Those features are among some of the **best technical achievements in RotTR according to some press release and my biaised self.**

FROM SHORE TO HORIZON

Achievements

ACMSIGGRAPH
GDC
GPU PRO7
LABS
eidos montréal
A SQUARE ENIX COMPANY
IOI
CRC Press
Taylor & Francis Group
AN A K PETERS BOOK
Advan
GPU PRO7
Edited by Wolfgang Engel

We have lead multiple, **direct and indirect, promotion of the games' technical achievements**, through major industry events and literature.



FROM SHORE TO HORIZON

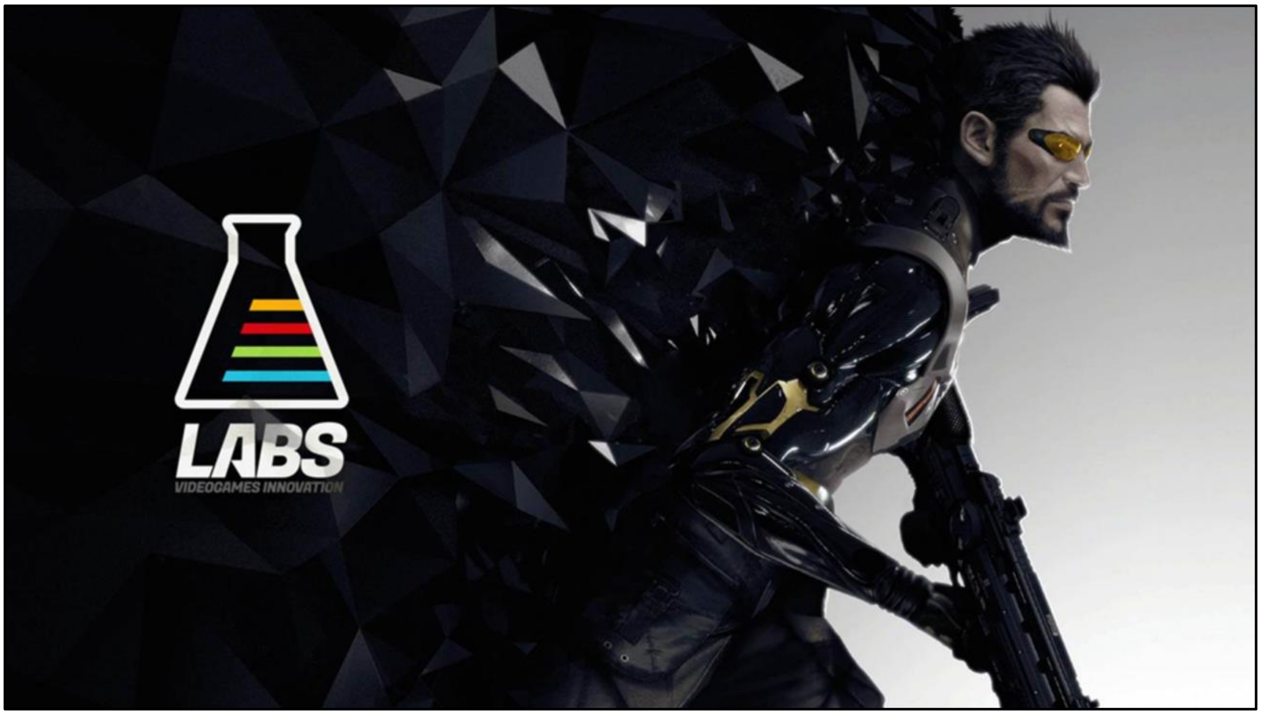
Achievements

- Two years of continuous collaborations.
- Working on over 10 different new features.
- Integration of « mainstream » features required by our innovation.

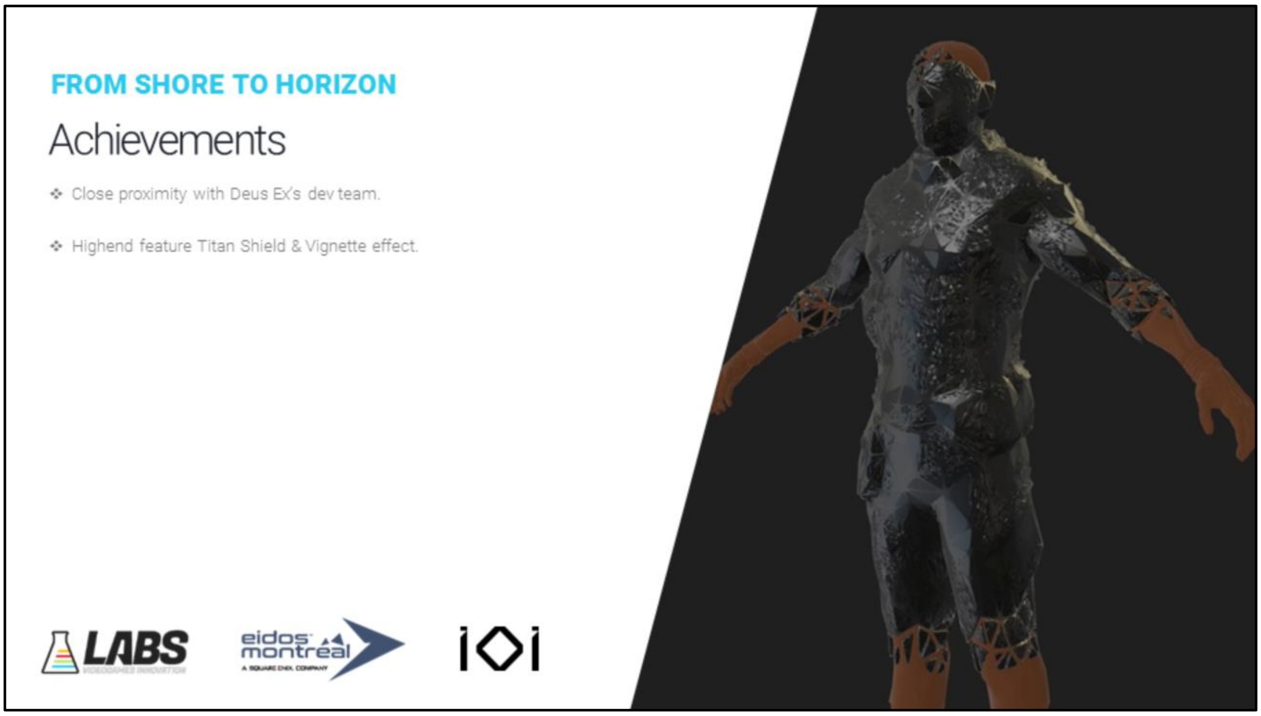
Two years of continuous collaborations with almost all of LABS's member.

Working **on over 10 different new features**, including some that never made it in the final product.

Tackling, at the same time, the **integration** of **« mainstream » features** required by our innovation.



Then moving on now to Deus Ex: Mankind Divided



FROM SHORE TO HORIZON

Achievements

- Close proximity with Deus Ex's dev team.
- Highend feature Titan Shield & Vignette effect.

The proximity with Deus Ex's team was a strong advantage. The different **studio experts were accessible resources, collaborating with us**.

This lead the group in **tackling some of their biggest challenges**:
Such as the iconic feature called **TITAN Shield & Vignette**.

FROM SHORE TO HORIZON

Achievements

- Unique PureHair solution
- Motion Blur
- Volumetric Lights
- Particle Lighting R&D

This mutual confidence **allowed the game team to benefit from many of our previous researches**.

A role expected from LABS is the **sharing of technology** when applicable, and in worst case the **sharing of knowledge gained, as broad as possible**.

With this in mind, **LABS worked hard in providing DXMD with features such:**

- **Unique PureHair** solution
- **Motion Blur** integration
- **Volumetric Lights** integration
- **Particle Lighting R&D**

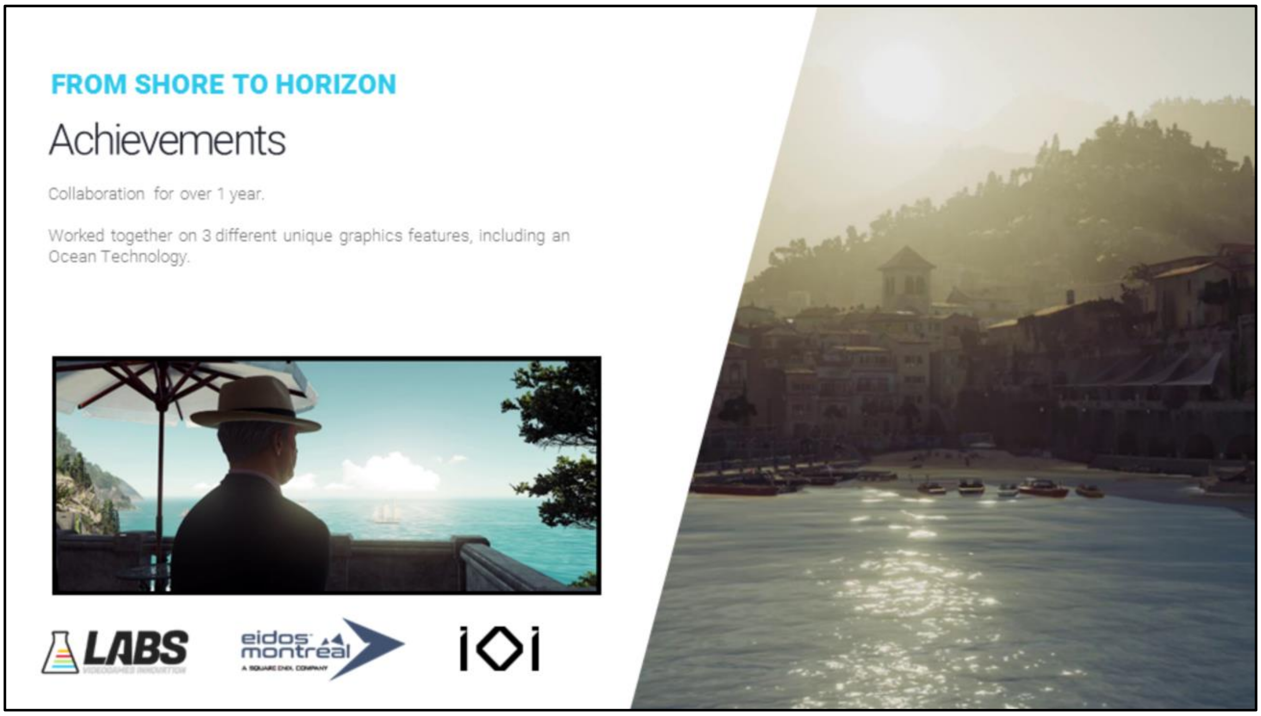
Lots of knowledge shared with this group.

Our collaboration with the team is still active on day-to-day basis.

Being an important **pillar to Eidos-Montreal**, **LABS was heavily involved in sharing back** some of our innovation with the community.

Additionally, LABS co-promoted technology **with** industry leading **partners as AMD, Autodesk** and **Hewlett-Packard**, only to name a few.

And Hitman game



FROM SHORE TO HORIZON

Achievements

Collaboration for over 1 year.

Worked together on 3 different unique graphics features, including an Ocean Technology.

LABS was looking forward to collaboration with Io-Interactive's talented team.

The different work have got us **collaborating for a little over 1 year.**

We had the chance to work together on **3 different unique graphics features**, including an **Ocean Technology**.

FROM SHORE TO HORIZON

Achievements

Promoted the Ocean technology:

GDC 2017

GDC17

Proud of the work done, LABS **naturally promote the Ocean technology** in different ways, explains **our presence here with you today.**

While Nic take the mic, let’s **have a look at the prototype of the work done with a short video**.

FROM SHORE TO HORIZON

Hello.

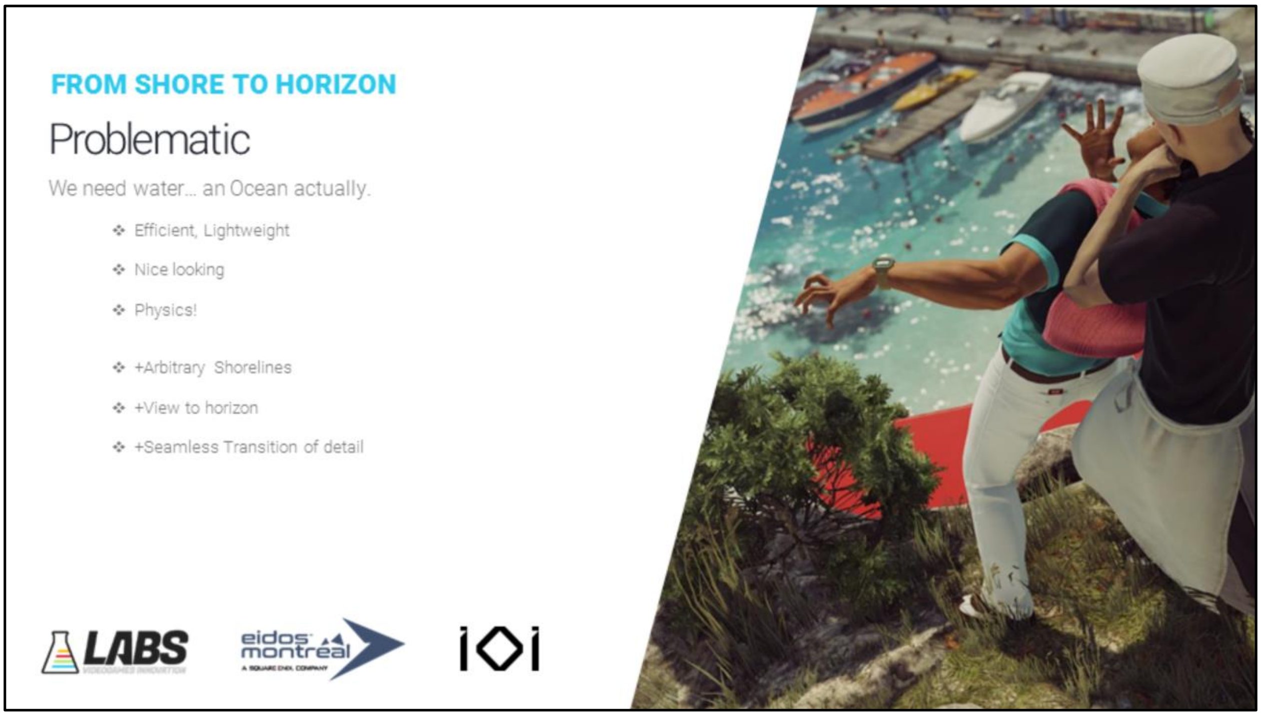
My name is Nicolas Longchamps.

Hi!
My name is Nicolas Longchamps

I am a **Technical Artist** in the LABS group at Eidos Montreal.
I've been in **3D for a little over 16y…** already!
With about **9y** of that spent at **Eidos**
My **Background** is mainly in **Graphics and VFX,**
And on the **technical side** of things, I'm mostly **self-taugh**,
Which I think is **not very unusual** for technical artists.

I'm also a **proud dad**, and I can bake a pretty **decent loaf of bread**.

So to start id like to note that this talk is **not about a revolutionary** ocean implementation **but a practical one.**
Water surfaces are a **difficult topic** in realtime 3d,
And my **hope today** is that I can **add to the community bag of tricks** for creating realtime water surfaces.



FROM SHORE TO HORIZON

Problematic

We need water… an Ocean actually.

- Efficient, Lightweight
- Nice looking
- Physics!
- +Arbitrary Shorelines
- +View to horizon
- +Seamless Transition of detail

During Hitman production,
Roughly around **mid to end 2014**
IO Interactive **approached us** for help to **create a solution** for their in-game **water surfaces**, specifically ocean rendering.

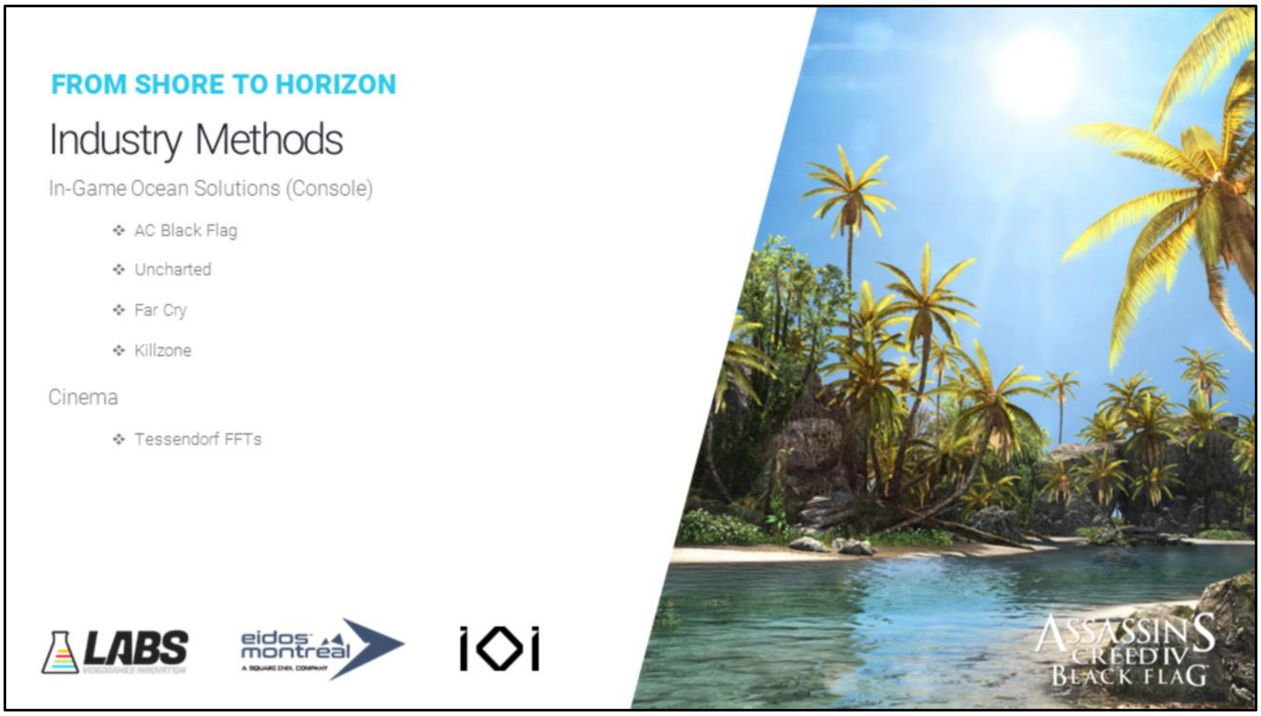
They wanted an **efficient**, **lightweight** solution that was **up to par** with current methods, and could be **easily implemented** in a variety of contexts and have some **support for physics** interactions.

From the supplied maps and reference, I could also see other **non-trivial features** like **transparent water**, an **arbitrary shoreline**, and the fact that you could both walk along the beach and view it from a high vantage point with an **unobstructed view to the horizon** in several directions, hence the title for this talk :-)

This implied a **seamless transition of surface detail** from close up to kilometers away.

Physics interactions included floating objects such as buoys and small boats, ballistics,
and this being a Hitman game of course, **ragdolls a.k.a dead bodies**.

Supporting **physics** became the **initial focus**, as this would drive how the rest of the effect would be designed and function.
Meaning the solution had to be CPU-Physics friendly.



FROM SHORE TO HORIZON

Industry Methods

In-Game Ocean Solutions (Console)

- AC Black Flag
- Uncharted
- Far Cry
- Killzone

Cinema

- Tessendorf FFTs

Assassin's Creed IV Black Flag

So we started looking at what was done at the time,
and there were a lot of implementations to choose from;

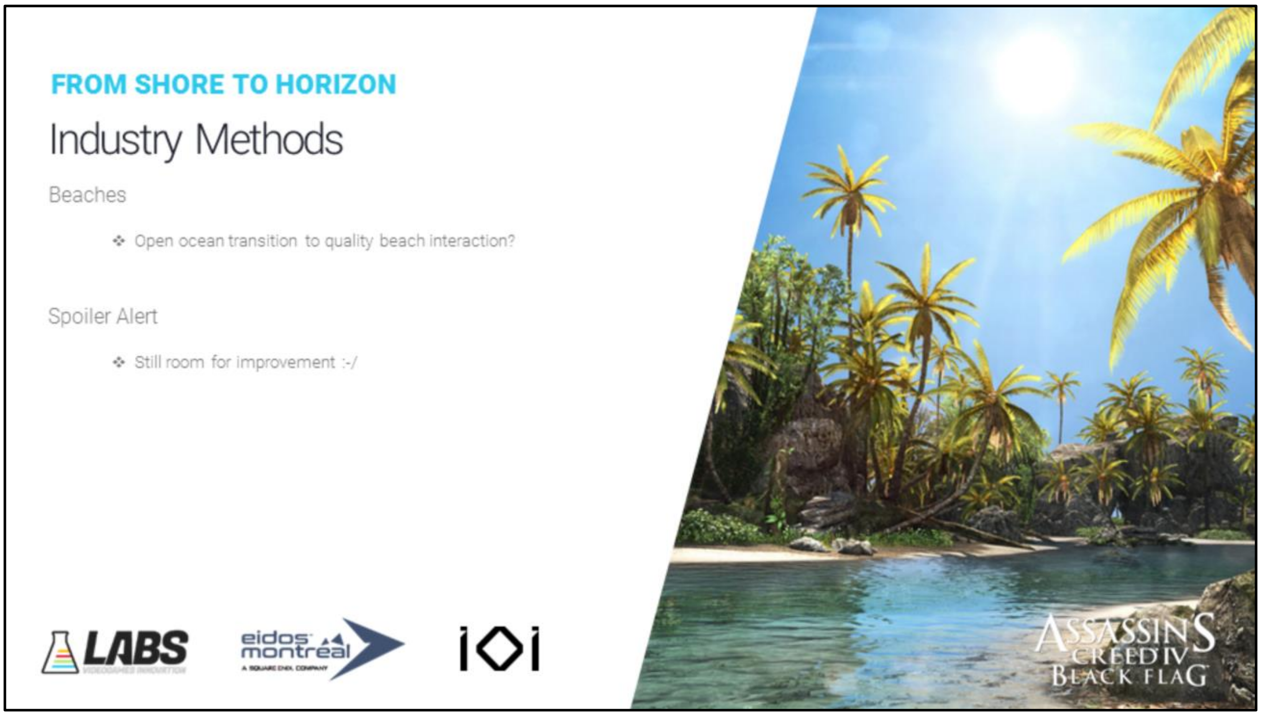
Assassins Creed Black Flag had a nice solution, which sort of became my **visual benchmark** for the effect.
Even though high seas sailing wasn't necessary for us

Uncharted,
Far Cry
and **KillZone** are some that also stood out at the time.

All very nice solutions using interesting techniques. And all with different contexts and requirements.

Of course there was also the obligatory read of **Jerry Tessendorf's** paper on Ocean Rendering,
the **gold standard** for ocean simulation in **Cinema VFX**, but **also** more recently applied in some **AAA** titles.

-“**Assassin’s Creed III: The tech behind (or beneath) the action**”, Mike Seymour, FXGuide 2012
-“**Water Technology of Uncharted**”, Carlos Gonzalez-Ochoa, Naughty Dog, GDC 2012


FROM SHORE TO HORIZON

Industry Methods

Beaches

- Open ocean transition to quality beach interaction?

Spoiler Alert

- Still room for improvement :-/

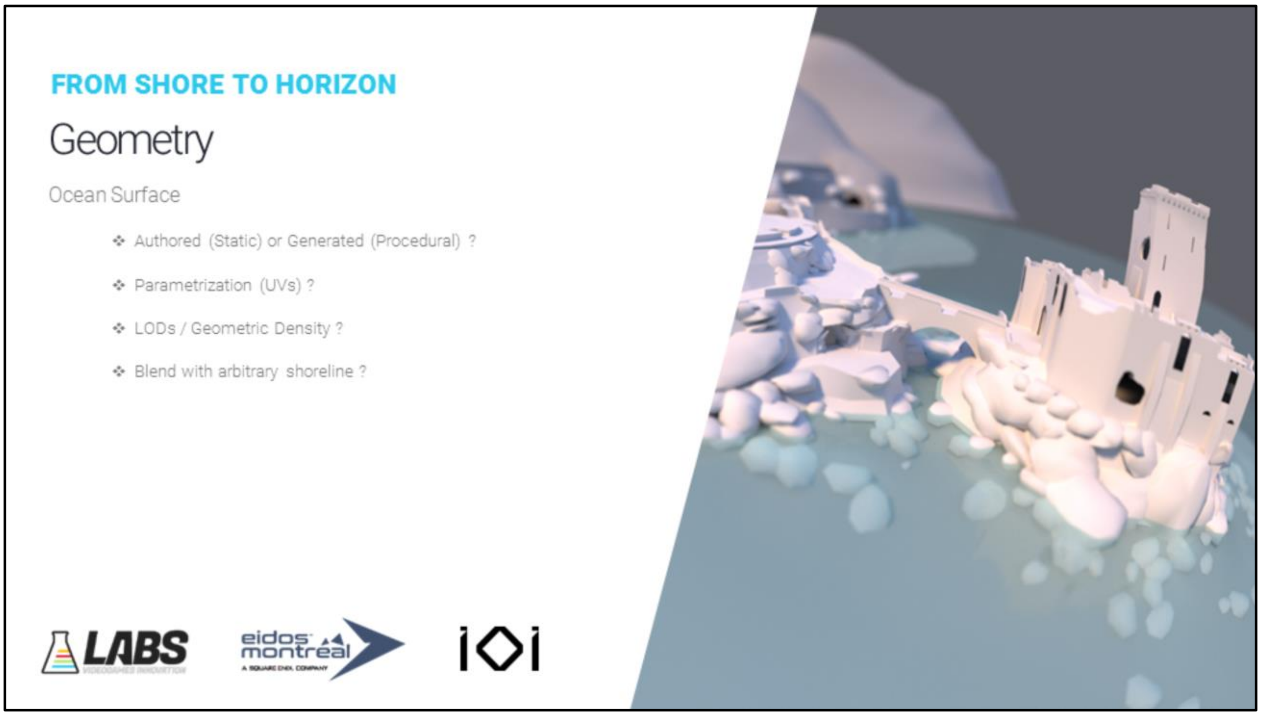
ASSASSIN'S CREED IV BLACK FLAG

There are **a lot of open water solutions**, but unfortunately **few for quality beach interactions**,
except for offline and expensive SPH or FLIP particle simulations.

And it is **difficult to find** any solutions that offer a **seamless integration** of both open water and quality beach interactions.

Solutions that tackle open water well usually have limited beach interactions if any.
The effect sort of lives inside a tank.
And a lot of times the water displacement simply clips the beach, or is attenuated by distance field or water depth textures.

(**Spoiler Alert**!)
I don't think I've completely solved this problem myself, but hopefully I've taken some steps in the right direction.



FROM SHORE TO HORIZON

Geometry

Ocean Surface

- Authored (Static) or Generated (Procedural) ?
- Parametrization (UVs) ?
- LODs / Geometric Density ?
- Blend with arbitrary shoreline ?

So to start: what do we need to make an Ocean surface?

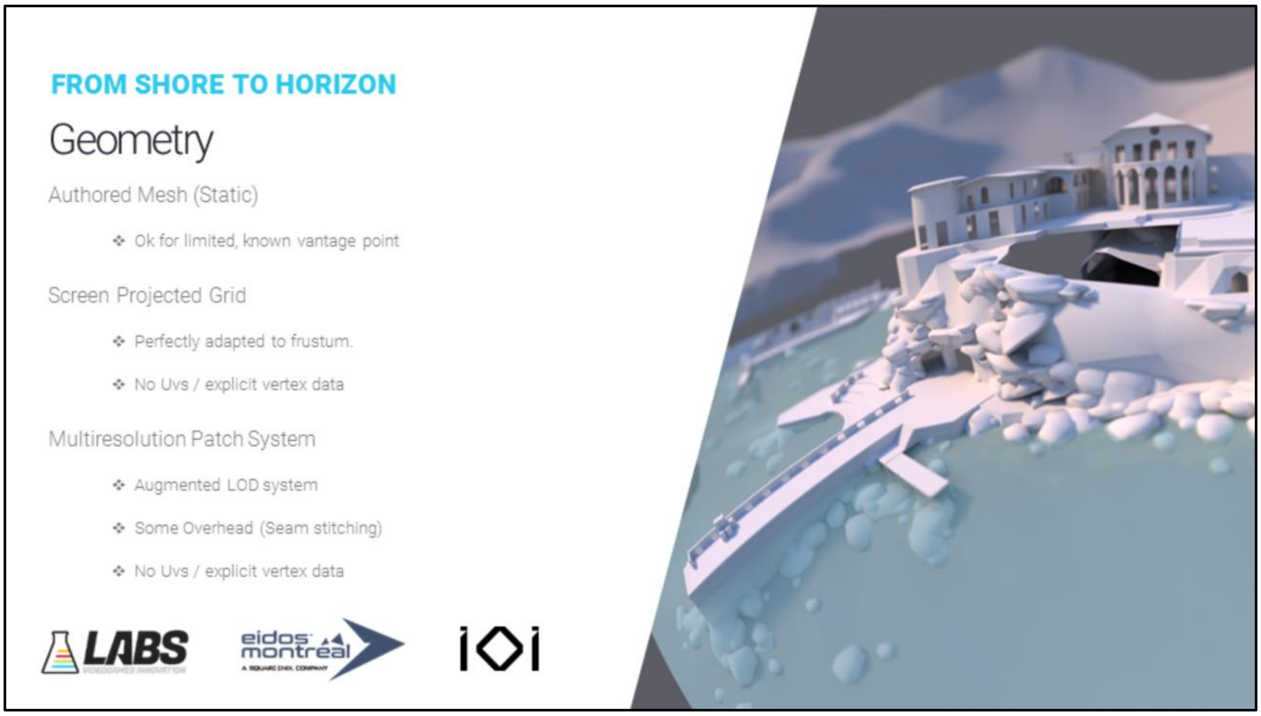
Well… we need a surface, of course
That seems obvious
but what kind of surface? How is it created?

Is it **procedural or static**?

How is it **parametrized**?

How is the **geometric screen density** handled?

Open water is simpler but what about our **beach**?
And how to **transition** from our beach to open water?



FROM SHORE TO HORIZON

Geometry

Authored Mesh (Static)

- Ok for limited, known vantage point

Screen Projected Grid

- Perfectly adapted to frustum.
- No Uvs / explicit vertex data

Multiresolution Patch System

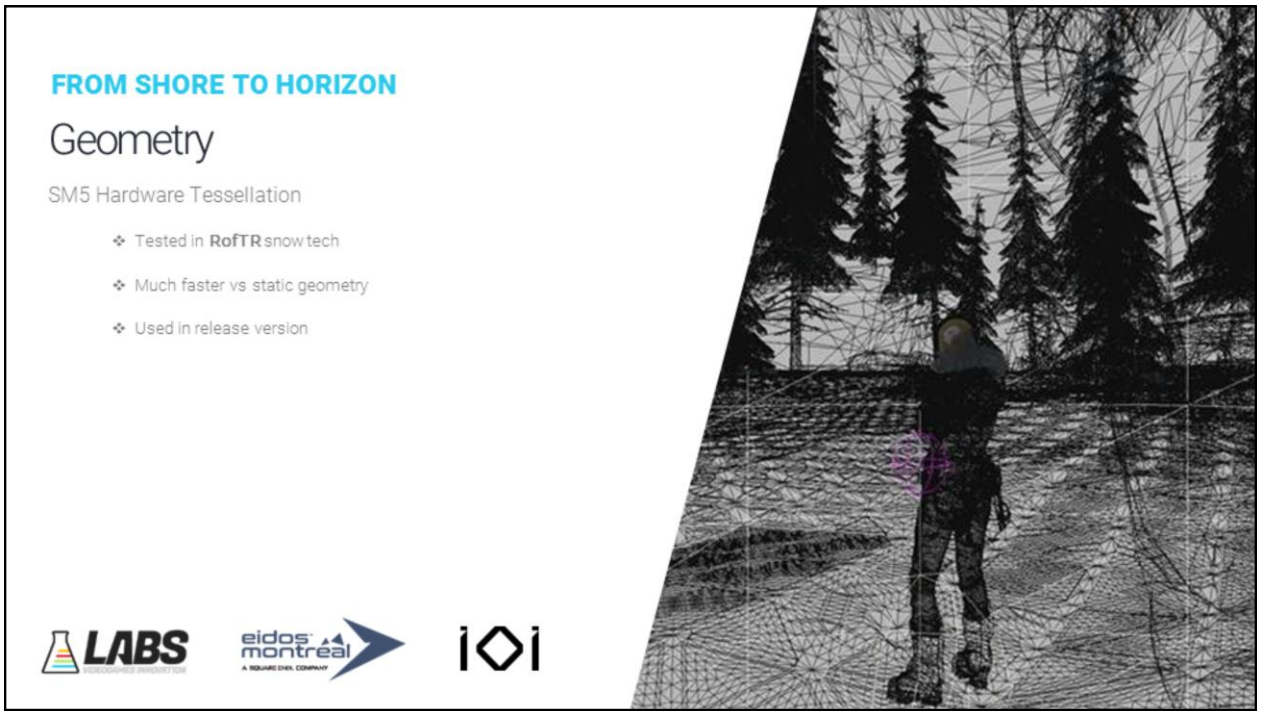
- Augmented LOD system
- Some Overhead (Seam stitching)
- No Uvs / explicit vertex data

Common choices for surface generation are simple **static geometry** (a user authored model asset), a **screen projected grid** or a system of **multiresolution patches**.

Static geometry can be okay if you have a **limited vantage point** and are able to **statically distribute** geometry form a **known point**. This isn't our case at all, and we wanted to avoid handling a LOD system.

The **screen projected grid** is a **popular choice** for good reason. It is by nature restricted to the **frustum**. No frustum culling necessary. Screen geometric density is constant and distribution is more or less ideal. Parametrization for shorelines can be an issue however, as no Uvs or other vertex data can be localized on the mesh itself. Localized data has to be then stored in implicitly mapped textures or by other means.

Patch systems incur some **system overhead** and require **seam stitching** at the borders of patches where different resolutions meet. They also have the same parametrization issues as the screen projected method.



FROM SHORE TO HORIZON

Geometry

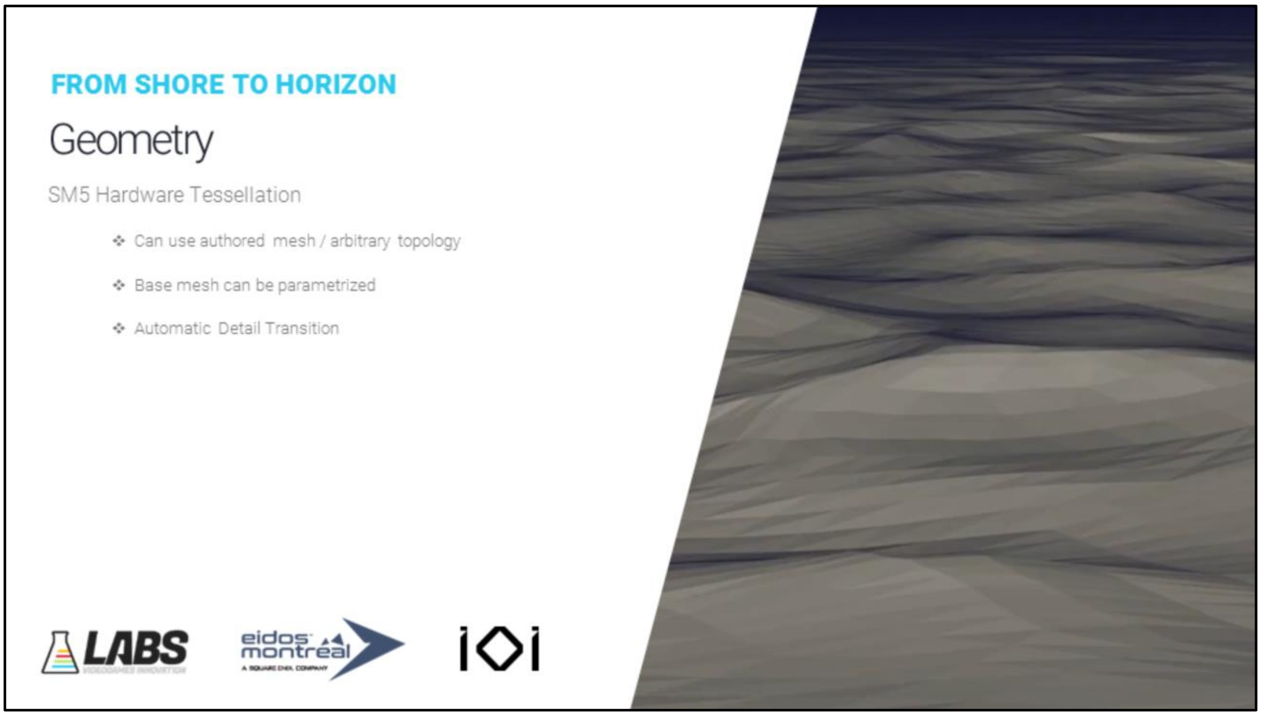
SM5 Hardware Tessellation

- Tested in **RofTR** snow tech
- Much faster vs static geometry
- Used in release version

Hardware tessellation was something I had wanted to apply to water for awhile.

At labs we had **implemented** it for our **Titan Shield** effect in **Deus Ex**, and afterwards more extensively in our **snow deformation** effect in **Rise of the Tomb Raider**.

During the **development** of that feature, we had **tested** hardware tessellation against a comparably dense static mesh and found hardware tessellation to be **much faster**.



FROM SHORE TO HORIZON

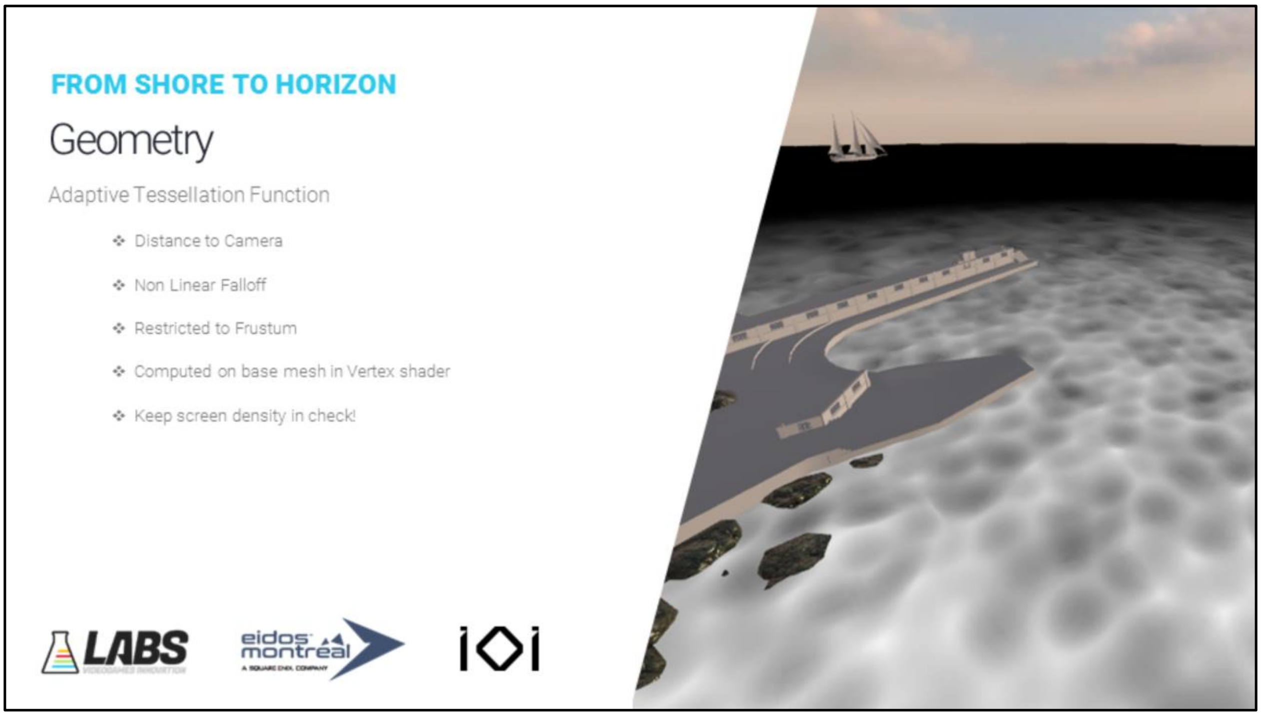
Geometry

SM5 Hardware Tessellation

- Can use authored mesh / arbitrary topology
- Base mesh can be parametrized
- Automatic Detail Transition

Using tessellation would allow for **parametrization** and **arbitrary topology** of the base mesh where necessary, meaning I could UV map my shoreline if I required to do so. Also, transitions in **geometric resolution** would be **automatically** handled, so no stitching algorithms or **LOD** meshes are necessary.

So for the ocean, we are using **authored mesh** as a base for tessellation, but as a very sparse **hull mesh**.


FROM SHORE TO HORIZON

Geometry

Adaptive Tessellation Function

- Distance to Camera
- Non Linear Falloff
- Restricted to Frustum
- Computed on base mesh in Vertex shader
- Keep screen density in check!

For tessellation we employ a simple **distance-to-camera adaptive tessellation** technique with a **non linear falloff**.

Tessellation **density** can be kept **constant to a given distance**. After that, possibly due to angle of **incidence**, I found that I could have the geometric density falloff much quicker than linearly with little noticeable effect in quality.
Tessellation is also kept to **within the camera frustum**.

In the **Shader**, the camera distance function is computed in the **vertex shader** and passed on to other stages for modulating the tessellation factor, displacement amplitude and other shader effect weights.

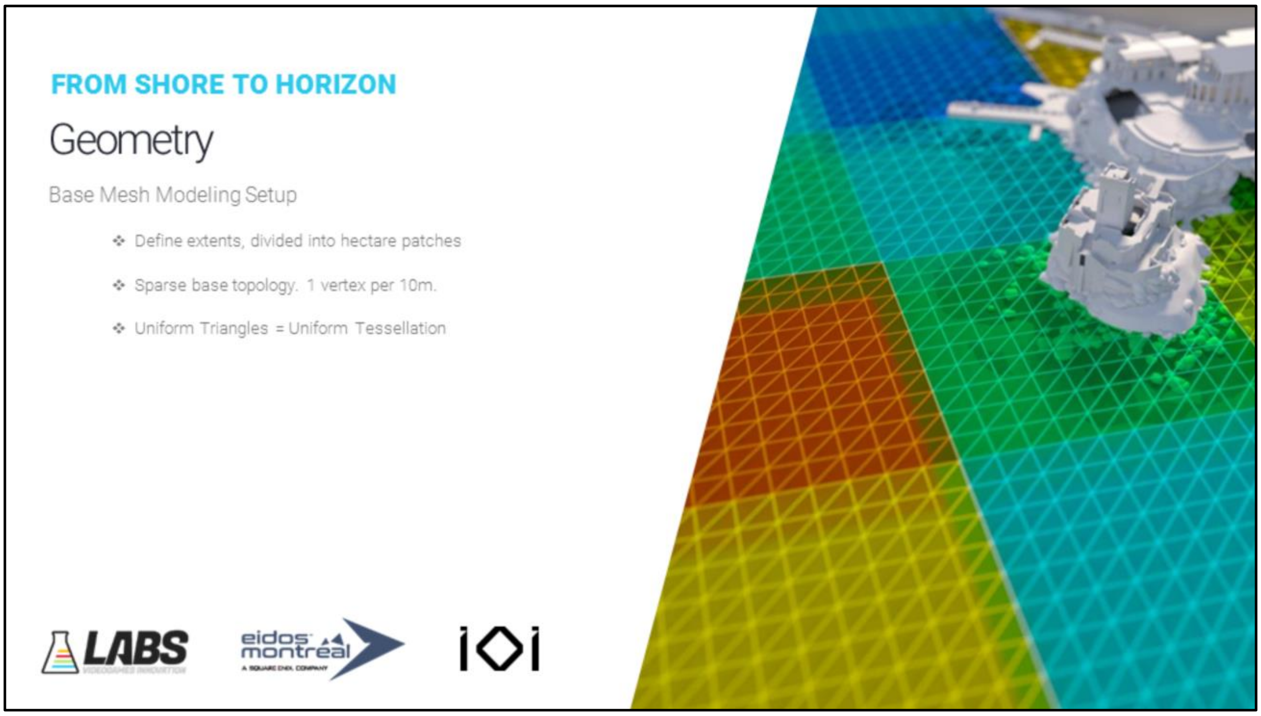
As for the **final tessellation factor**, there are a lot of **guidelines** for tessellation. Most simply state to keep it below a certain number. But this doesn't take into account the initial geometric density. My base mesh has huge triangles. So this is not really helpful.

The one guideline I found most **useful** and stuck to was **keeping screen density in check**, that is to avoid having triangles smaller than a **16px** screen area in order to maintain **rasterizer efficiency**.
I found this info in an AMD Developpers presentation ; **The AMD GCN Architecture**

- A Crash Course, by Layla Mah.

When setting edge and face **tessellation factors**, only edge factors handled. Face factors are simply and **average of the former**.

The AMD GCN Architecture - A Crash Course, by Layla Mah
https://www.slideshare.net/DevCentralAMD/gs4106-the-amd-gcn-architecture-a-crash-course-by-layla-mah
slides 59-63



FROM SHORE TO HORIZON

Geometry

Base Mesh Modeling Setup

- Define extents, divided into hectare patches
- Sparse base topology. 1 vertex per 10m.
- Uniform Triangles = Uniform Tessellation

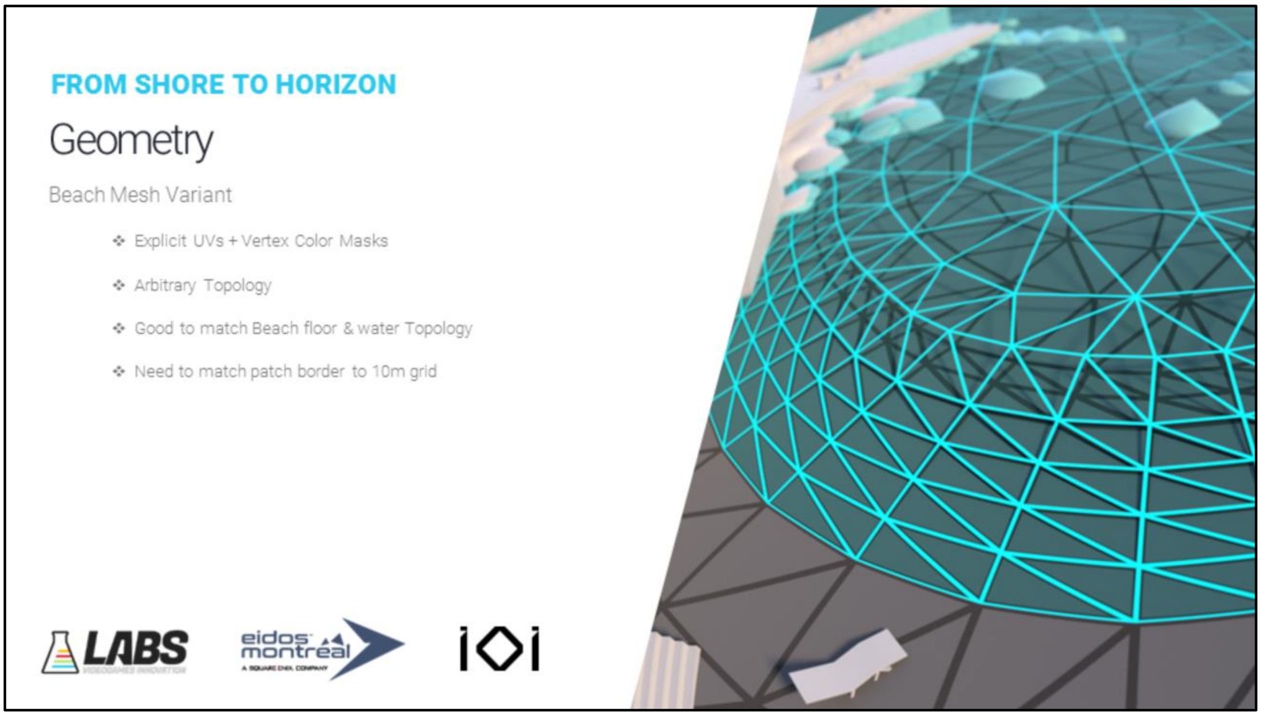
LABS eidos montréal IOI

With the exception of the beach variant base mesh, which ill explained later, the **base geometry** is very **sparse**.

It's a **radial** ocean mesh roughly **800m across**, divided into **hectare patches** (100m x 100m). The total mesh diameter was found by placing circles representing a desired view distance at several extreme positions in the map.
These areas were then **encapsulated** within a larger circle representing the final **extents** of the ocean surface geometry.

Base geometric **density** (before tessellation) is **1 vertex per every 10m**. Keeping a **uniform** distribution of triangles will allow to better control tessellation density.

Permanently **occluded** geometry can be removed, but cuts are kept on the 10m grid as to keep the **uniform** triangle size.



FROM SHORE TO HORIZON

Geometry

Beach Mesh Variant

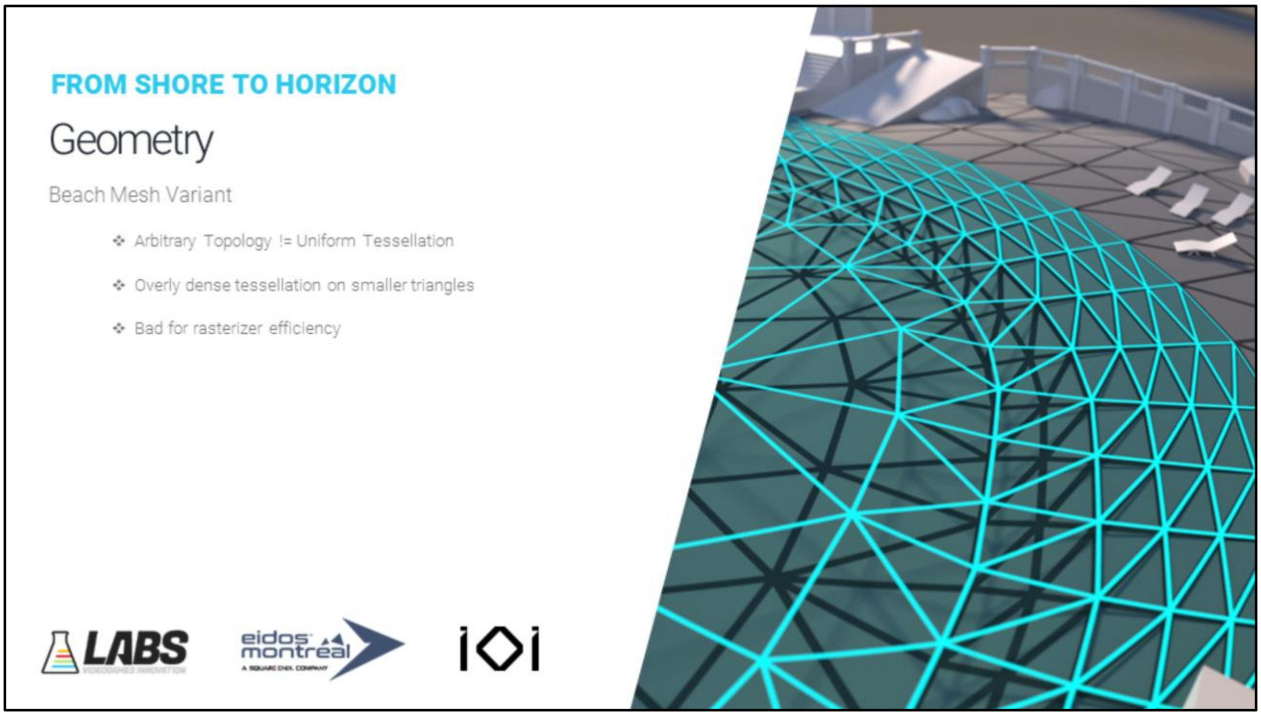
- Explicit UVs + Vertex Color Masks
- Arbitrary Topology
- Good to match Beach floor & water Topology
- Need to match patch border to 10m grid

The **beach** mesh is a **special case**.

Here we want **Uvs** and **vertex color** data for our shore waves.
We also want a **close topology match** with the **shoreline**. So the 10m grid goes out the window in this case.

For modeling, I found it best to model both the water and floor of the shore from the same waterline **curve**.
This allowed for **matching topology and Uvs** on the shore floor, which could be subsequently be synched to the shore water effect and used for a "wet sand" animated effect.

We still want to have the **borders** of our beach patch **match** the 10m grid, as to not have **tears** when displacing.



FROM SHORE TO HORIZON

Geometry

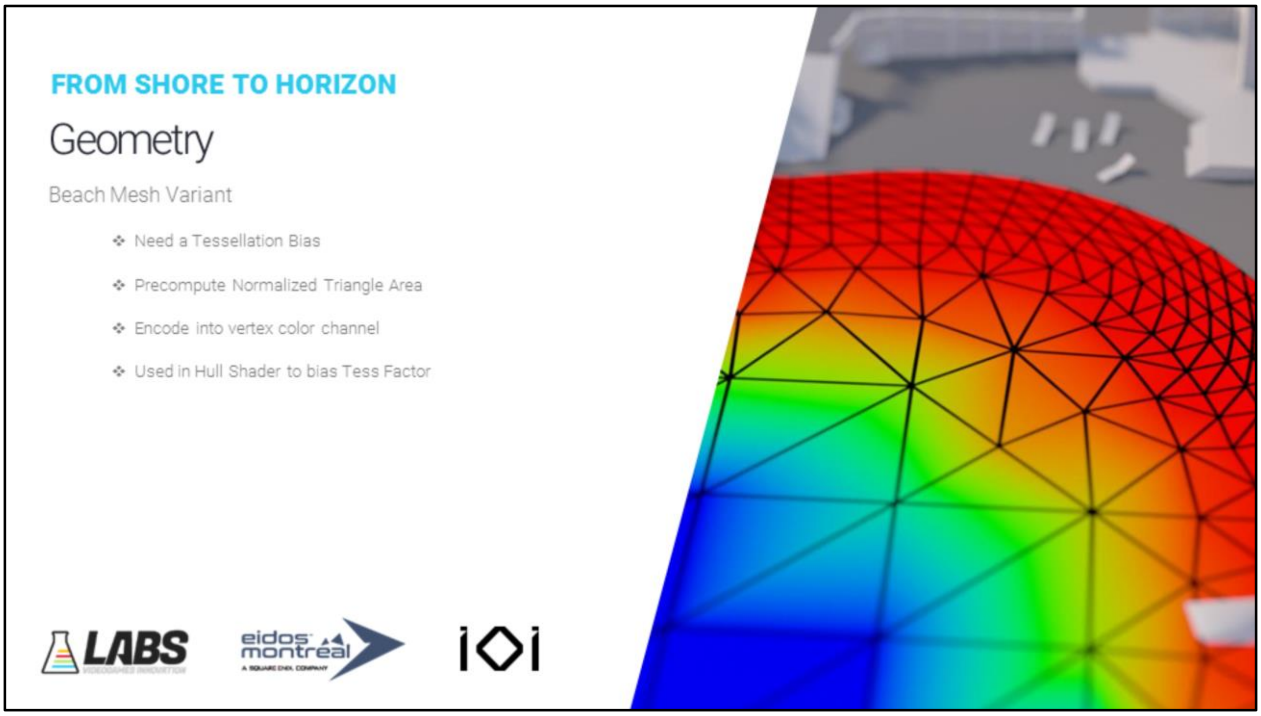
Beach Mesh Variant

- Arbitrary Topology != Uniform Tessellation
- Overly dense tessellation on smaller triangles
- Bad for rasterizer efficiency

However now we have **problem** for tessellation;

Tessellation will divide all triangles by the given amount no matter what their size.

Keeping to a **10m** grid kept a **homogeneous** triangle size and perfectly uniform subdivision. But now we have **smaller triangles** with the same tessellation factor, giving us some very dense and unnecessary geometry. And ultra dense geometry is bad for rasterizer efficiency as we saw.



FROM SHORE TO HORIZON

Geometry

Beach Mesh Variant

- Need a Tessellation Bias
- Precompute Normalized Triangle Area
- Encode into vertex color channel
- Used in Hull Shader to bias Tess Factor

So we want to keep tessellation density relatively uniform, and to do that we encode a **tessellation bias** value into our vertex color data.

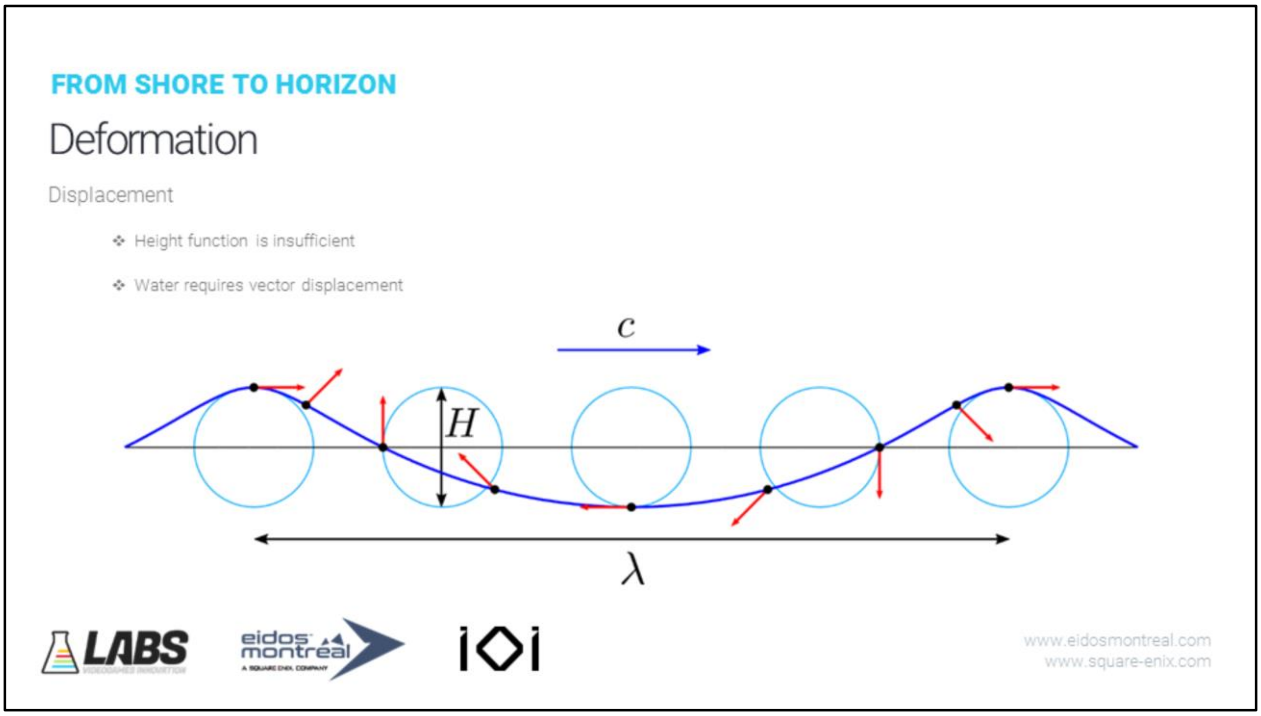
Here in the image Ive applied a **color ramp** so that it looks cool,
but its really a **scalar value**.

For water patches **marked as “beach type”,** this vertex data is used in the **Hull** shader to **bias** the tessellation factor.

To find the bias value, we **process** the geometry in our DCC and find each triangle’s **area**. Going back to our 10m grid, we know a “normal” triangle has an area of (10*10)/2 = **50m2**. Triangle areas are therefor **normalized** to **50m2**, giving a **0-1 size ratio** that can be encoded into a color channel.

Ratios bigger than 1 are clamped.

When applied, smaller triangles are **proportionally** tessellated to a lesser degree.

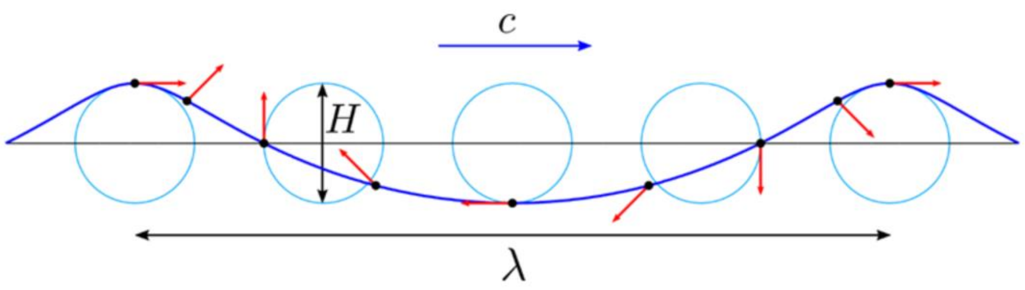


FROM SHORE TO HORIZON

Deformation

Displacement

- Height function is insufficient
- Water requires vector displacement



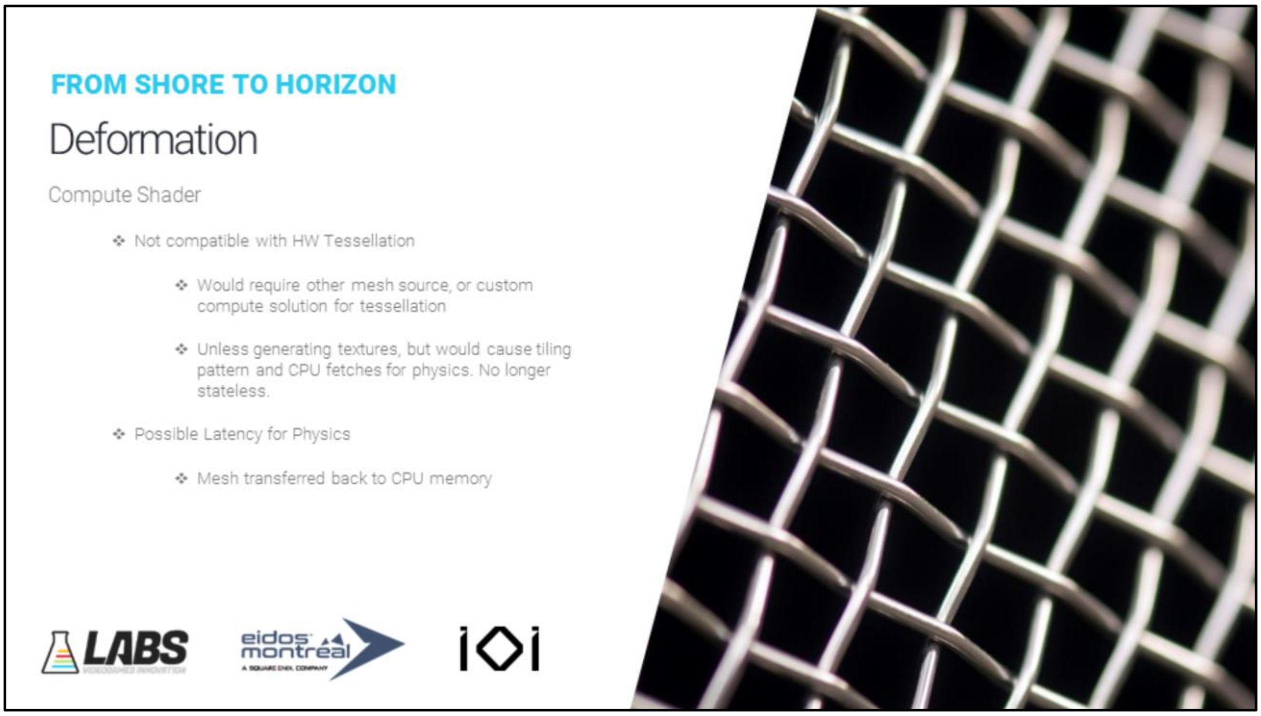
LABS eidos montréal IOI

www.eidosmontreal.com
www.square-enix.com

So we have a surface,
Now we need a **method** to move points our around.

Displacement is pretty **common** in games now, but for **water** it should be noted that a **height function is really not sufficient**.

Water churns and moves; **Lateral movement** is necessary to properly sell the effect. So we’re not looking for simple height displacement but rather **vector displacement.**


FROM SHORE TO HORIZON

Deformation

Compute Shader

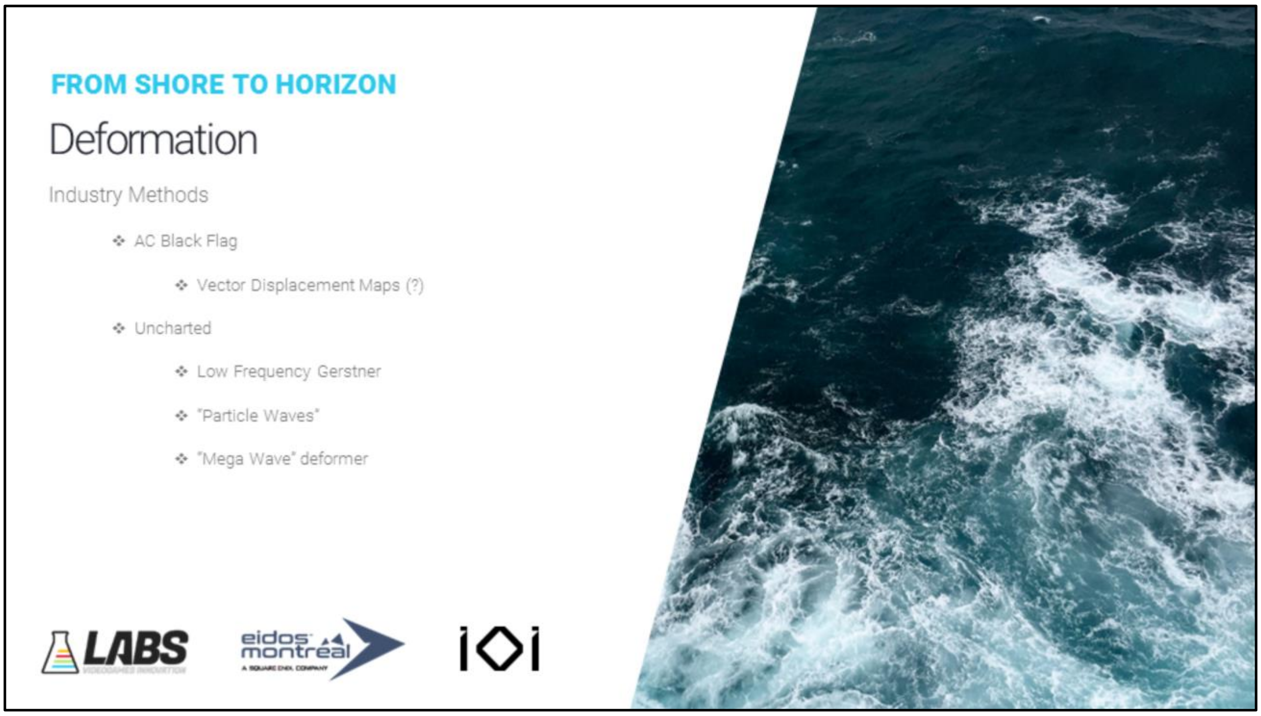
- Not compatible with HW Tessellation
  - Would require other mesh source, or custom compute solution for tessellation
  - Unless generating textures, but would cause tiling pattern and CPU fetches for physics. No longer stateless.
- Possible Latency for Physics
  - Mesh transferred back to CPU memory

A note on **Compute Shaders**,

We did quickly look into a **Compute** approach, but decided **against** it,
fearing **latency** issues that would incur when getting the modified mesh back from GPU in order to perform **CPU Physics**.

Maybe we could have used GPU Compute to **generate** a vector displacement & normal **maps** at runtime, but this would have brought with it a fixed **tiling** pattern that would have to be **managed**.

We also wanted to **avoid** doing **CPU texture fetches** for physics as much as possible.



FROM SHORE TO HORIZON

Deformation

Industry Methods

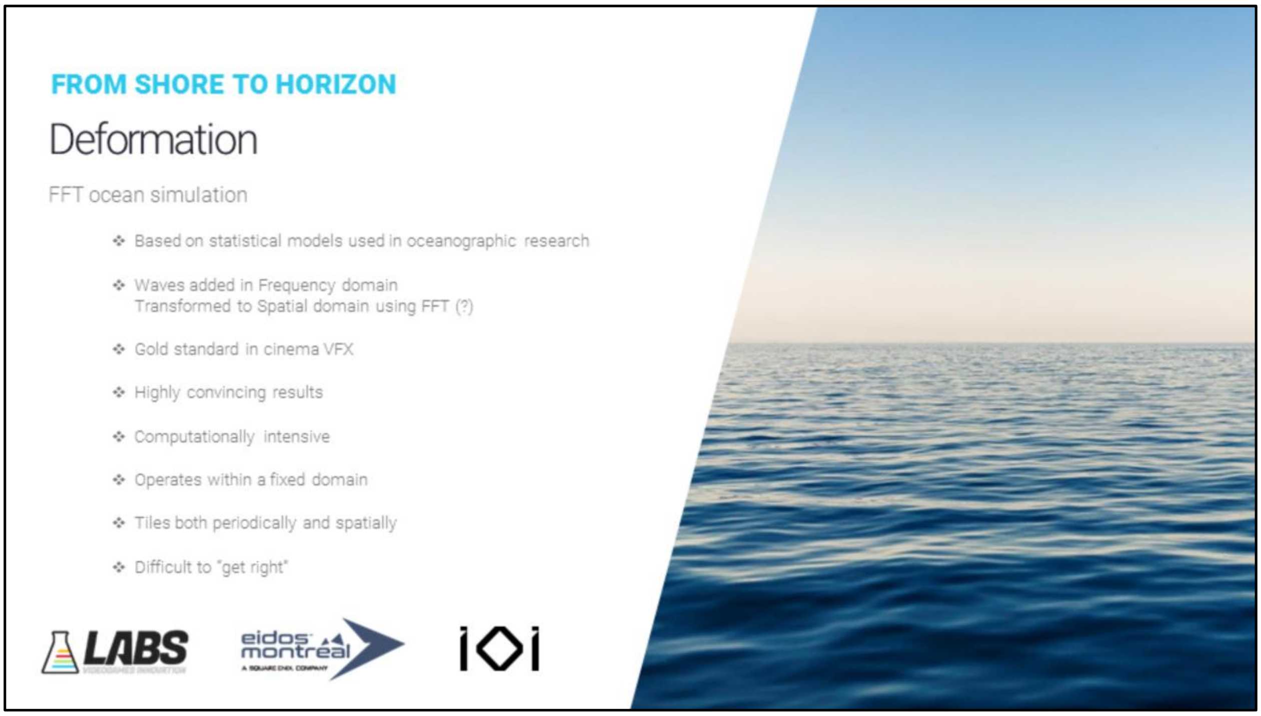
- AC Black Flag
    - Vector Displacement Maps (?)
- Uncharted
    - Low Frequency Gerstner
    - "Particle Waves"
    - "Mega Wave" deformer

Black Flag had a good implementation using I believe offline **generated vector displacement** maps but we ultimately did not explore this route.

Wanted a **stateless** function.

Other than vector displacement maps, what other **methods** have been applied? **Uncharted** made use of low frequency **gerstner** waves along with higher frequency "**particle waves**" and an additional "**mega wave**" **deformer**.

Our mandate did not call for such rough seas, but I liked the idea of artist **placeable** waves.


FROM SHORE TO HORIZON

Deformation

FFT ocean simulation

- Based on statistical models used in oceanographic research
- Waves added in Frequency domain
  Transformed to Spatial domain using FFT (?)
- Gold standard in cinema VFX
- Highly convincing results
- Computationally intensive
- Operates within a fixed domain
- Tiles both periodically and spatially
- Difficult to "get right"

We did look into **FFTs**, or Fast Fournier Transform ocean deformation as outlined in Mr. Jerry **Tessendorf's** 2001 paper, "Simulating Oean Water".

FFT waves are maybe not a good name. The fast fournier transform here is a **component** of a larger set of equations based on **statistical models found in oceanographic literature**. There is a reason FFT waves are pretty much the gold standard for ocean simulation in cinema, the method produces some very **convincing results**.

Although **advancements** in available CPU-GPU power in games **hardware** have recently made FFT waves **more viable**, as seen in the more recent Just Cause 3's Nvidia **Waveworks** implementation, they remain **computationally intensive**.

(And for a poor technical artist, a bit mathematically intimidating!)

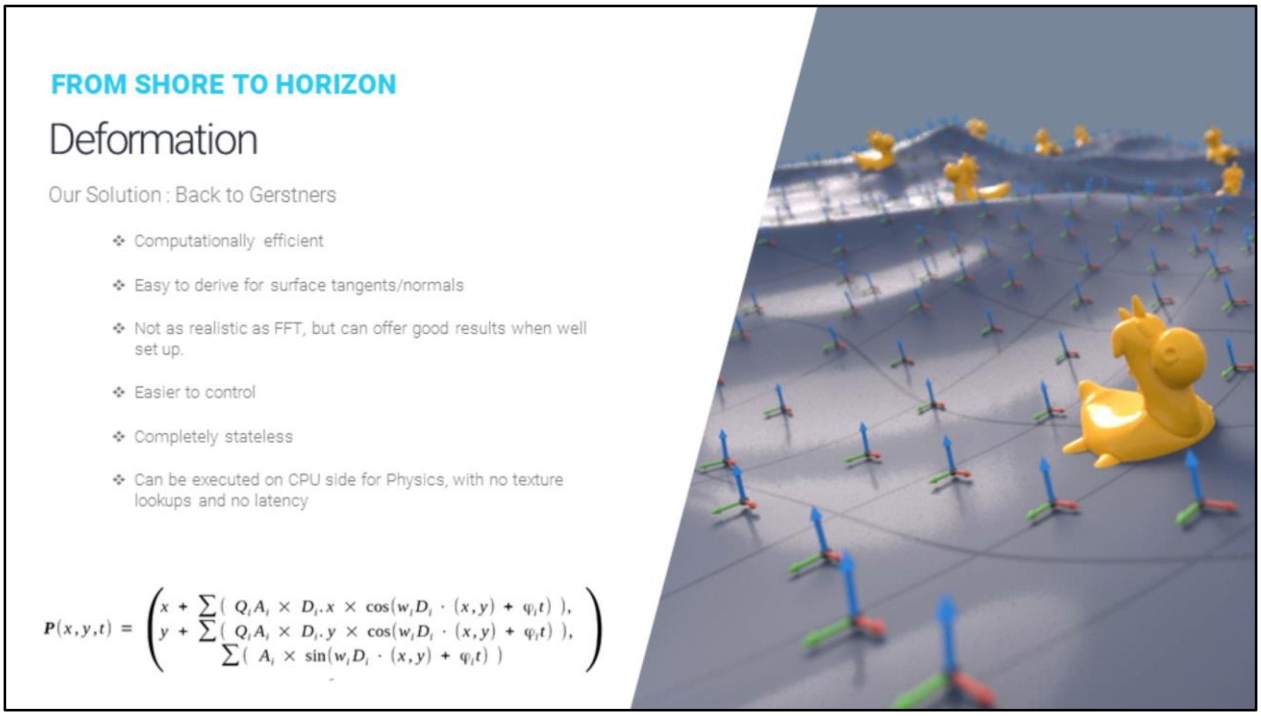
Math complexity put aside though, FFT waves have some **issues** to consider:

By nature of existing within a **domain**, they **tile** both spatially and periodically.
The spatial tiling **cant be fixed**, only **managed.**

They can be **difficult** to “get right”
and **generating normals** or CPU-side surfaces for **physics** requires **more FFTs**.

For very **water-centric games**, the cost may be **worth** it.

“Simulating Ocean Water” 2001 , Jerry Tessendorf



FROM SHORE TO HORIZON

Deformation

Our Solution : Back to Gerstners

- Computationally efficient
- Easy to derive for surface tangents/normals
- Not as realistic as FFT, but can offer good results when well set up.
- Easier to control
- Completely stateless
- Can be executed on CPU side for Physics, with no texture lookups and no latency

$$P(x,y,t) = \begin{pmatrix} x + \sum(\ Q_i A_i \times D_i.x \times \cos(w_i D_i \cdot (x,y) + \varphi_i t)\ ), \\ y + \sum(\ Q_i A_i \times D_i.y \times \cos(w_i D_i \cdot (x,y) + \varphi_i t)\ ), \\ \sum(\ A_i \times \sin(w_i D_i \cdot (x,y) + \varphi_i t)\ ) \end{pmatrix}$$

But for us, I went back and revisited **Gerstner** waves to drive our effect, starting from the well known GPU Gems article,
“Effective Water Simulation from Physical Models”, by Mark Finch.

Despite maybe being considered **obsolete,** …for cinema at least,
Gerstner waves have some **nice properties** that shouldn’t be overlooked ;

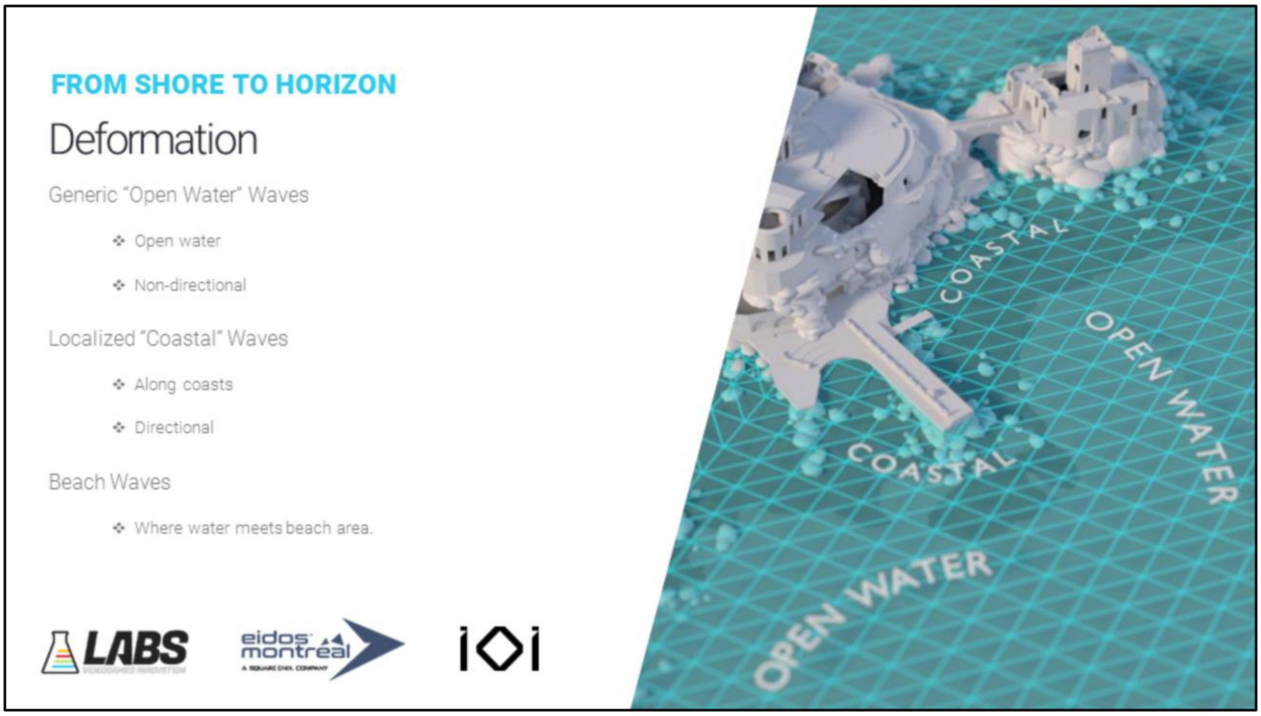
Gerstner Waves are computationally **efficient**,

and the function is easy to **derive** in order to generate a **surface normal** and **tangent basis**.

They might **not be as realistic as FFT** waves, but they can also offer convincing results given the **proper setup** and parameters.

Although there is also a **periodic element** to Gernstner waves, each wave is **independent** and not tied to a domain, so it’s easier to manage repetition.
One easy way to break repetition, which we implemented, is to employ a low frequency **phase noise texture** to the waves.

“Effective Water Simulation from Physical Models” , Mark Finch, GPU Gems 2004 NVidia



FROM SHORE TO HORIZON

Deformation

Generic "Open Water" Waves

- Open water
- Non-directional

Localized "Coastal" Waves

- Along coasts
- Directional

Beach Waves

- Where water meets beach area.

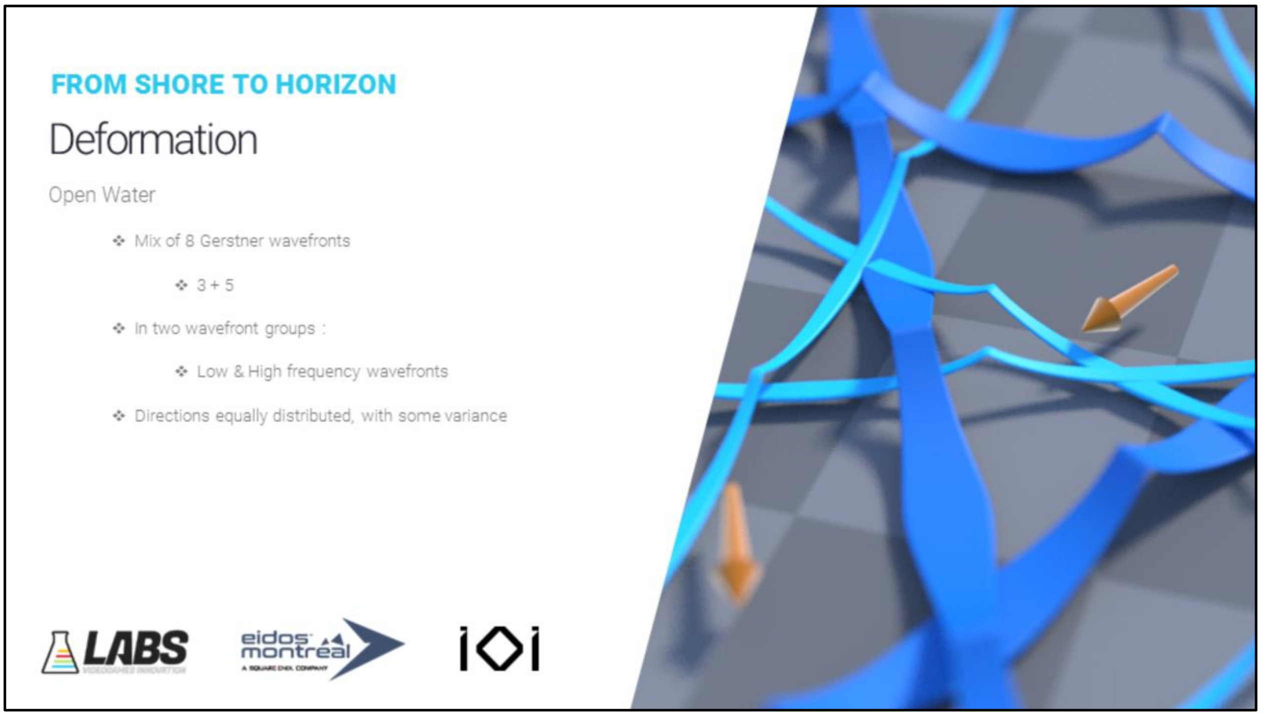
The deformation effect was separated into three water **categories**, (or behaviors) ;

Generic "open water" waves, areas where there is no directionality or influence from land masses.

Coastal waves would still be open water, but here you would have some **directionality** and **conformity** to the **coastline**.

& **Beach** waves. Where water actually makes **contact** with a land mass and the contact line needs to be **handled** somehow.

Water towards the **horizon** has no displacement so we'll talk about that later.



FROM SHORE TO HORIZON

Deformation

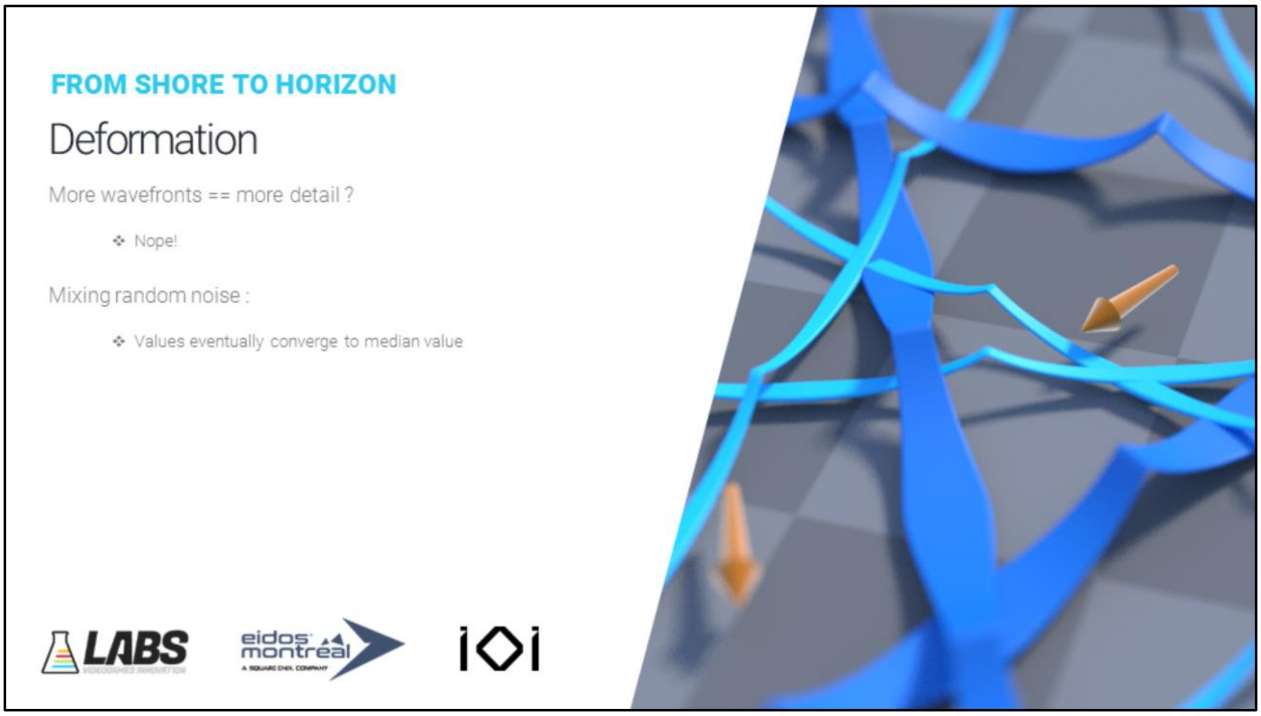
Open Water

- Mix of 8 Gerstner wavefronts
  - 3 + 5
- In two wavefront groups :
  - Low & High frequency wavefronts
- Directions equally distributed, with some variance

Open water is the **generic**, non-directional motion of the water. This is the **base layer** of the effect.

Its a mix of **8 Gerstner** waves arranged in a way to cover all directions without canceling each other out and create wave motion **not biased** to any particular **direction**.

While we’re talking about mixing Gerstner waves, there are a few points I can share on the topic…



FROM SHORE TO HORIZON

Deformation

More wavefronts == more detail ?

- Nope!

Mixing random noise :

- Values eventually converge to median value

LABS
eidos montréal
IOI

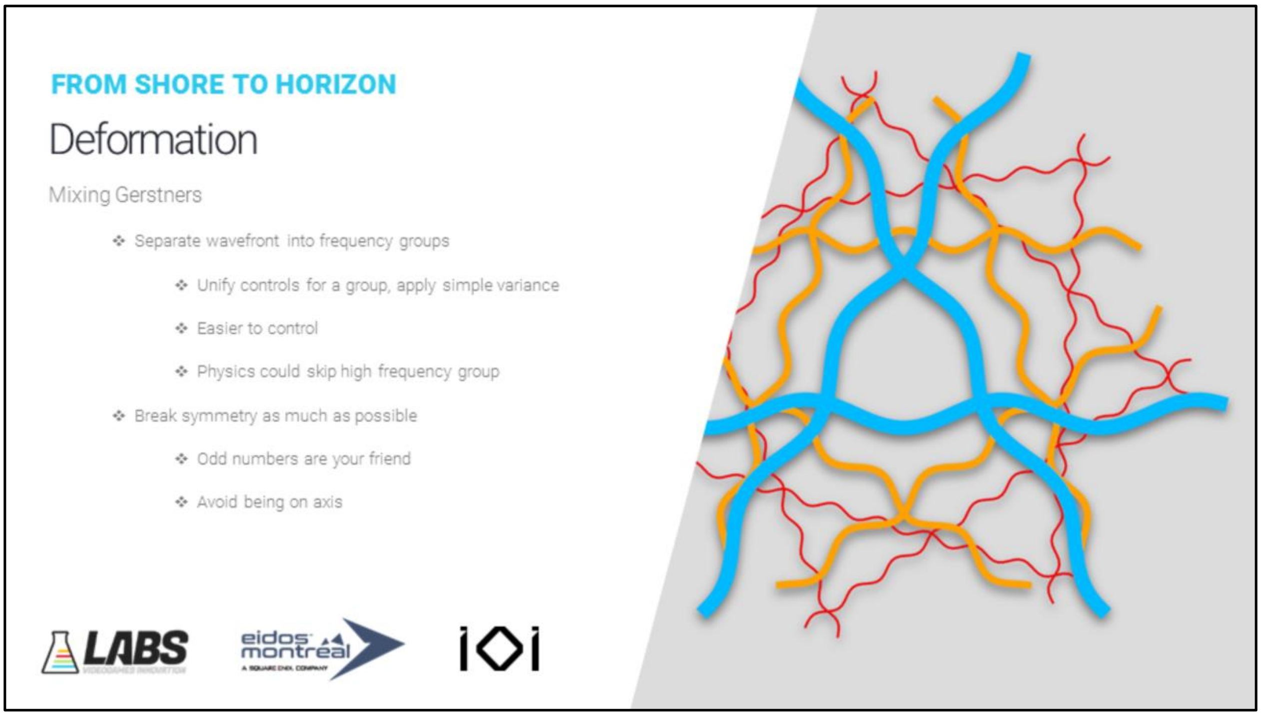
If you're like me, your first **instinct** would probably have been to mix **many** wavefronts
Just randomly pack them on,
thinking **more waves = more detail**!

Not necessarily it turns out,

What happens is like **adding noise** on top of noise, values eventually **converges** to the **median value**.

And in our case, since we're doing **bidirectional** displacement, the **median** value is **zero**. So the more wavefronts I added to the system, the more my water surface got flatter. Basically all of the delta positions were **cancelling** each other out.

So more is **not** better,
Not at similar **frequencies** at least. Which brings the second point…



FROM SHORE TO HORIZON

Deformation

Mixing Gerstners

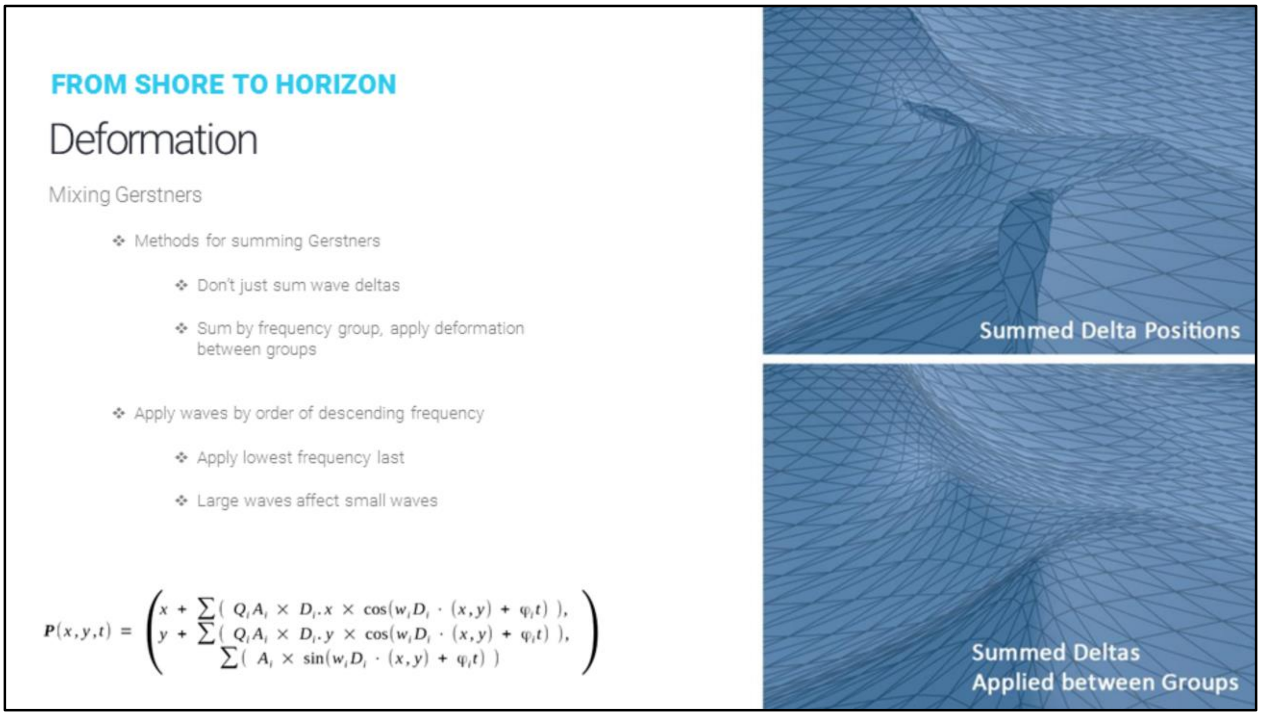
- Separate wavefront into frequency groups
    - Unify controls for a group, apply simple variance
    - Easier to control
    - Physics could skip high frequency group
- Break symmetry as much as possible
    - Odd numbers are your friend
    - Avoid being on axis

Separate your wavefronts into **frequency groups**.
For example, we have a low frequency group of 3 wavefronts, and a high frequency group of 5 wavefronts.

Generally the **lower** the **frequency**, the **less wavefronts** you want to use.

Doing this I found Is a good way to **manage the complexity** of the effect:
Separating into groups **allows** to **unify the controls** for the waves in that group, and simply apply variance to the individual waves, making the overall effect **easier to control**.
Additionally, you may want to **skip** a high frequency group when **sampling** positions on CPU for physics. Or any other instances where you might need a low fidelity height sample.

And finally the **obvious** :
You want to **break symmetry** as much as possible, so **odd numbers** are your friend. Add **variance** where you can and **avoid** having anything on **axis**. Our 3 wavefronts are radially distributed, and our 5 high frequency wavefronts start from the reverse direction, so no directions in the same group overlap and overall the directions are well distributed.



FROM SHORE TO HORIZON

Deformation

Mixing Gerstners

- Methods for summing Gerstners
  - Don't just sum wave deltas
  - Sum by frequency group, apply deformation between groups
- Apply waves by order of descending frequency
  - Apply lowest frequency last
  - Large waves affect small waves

$$\mathbf{P}(x,y,t) = \begin{pmatrix} x + \sum (Q_i A_i \times D_i.x \times \cos(w_i D_i \cdot (x,y) + \varphi_i t)), \\ y + \sum (Q_i A_i \times D_i.y \times \cos(w_i D_i \cdot (x,y) + \varphi_i t)), \\ \sum (A_i \times \sin(w_i D_i \cdot (x,y) + \varphi_i t)) \end{pmatrix}$$

Conceptually, ocean surface is composed of an infinite amount of random wavefronts.
But **realistically**, waves **interact**. Collide, exchange energy, they die, separate, merge. Of course **our waves don't do that**. They simply add. This is also the case for an FFT ocean surface I believe. I'm inclined to say that these are not technically simulations, like you would have with SPH or FLIP particle solutions, where a finite data set is updated through a solver.
We have **no data set to update**, but we can control **how** the waves are **added** and the base positions from which they are computed.

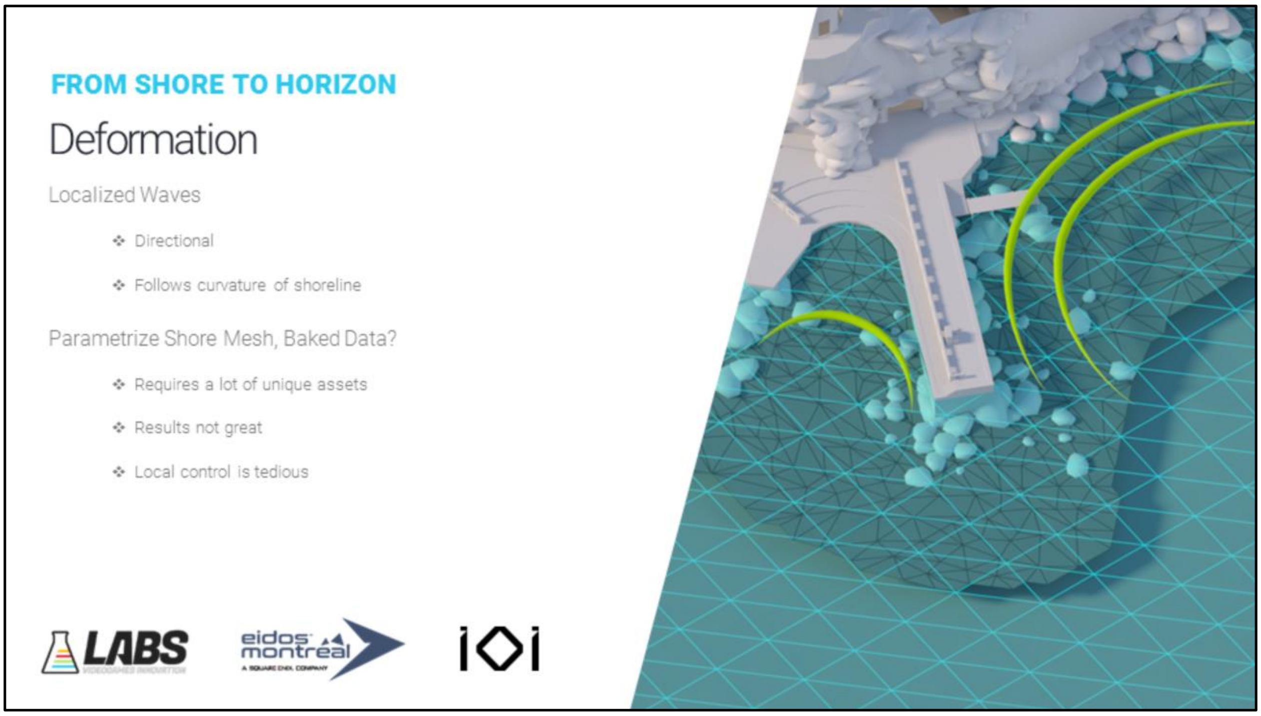
So I find there's **basically 3 methods to sum Gerstner waves**, in reference to when they get applied to the base :
Sequentially, **Summed delta positions** or **Summed deltas by Group**.

The **first** method is to simply **sequentially add** the waves, so that each wave is computed from positions deformed by previous waves. This isn't a great method and is difficult to control. One reason I dislike it is that the first wave is deformed by all the subsequent waves, but the last wave isn't deformed at all. So the "weight" of each wavefront is not really equal.

The **second** method, as seen in GPU Gem’s Gerstner water surface equation, is computing all waves from the base position and then **adding the summed delta positions** in one go. This way all waves have an equal effect weight. However since they all are computed from the base position, the waves don’t deform each other, which is a nice effect to have.

The **third** method takes **advantage** of the fact that we’ve **separated** our wavefronts into **frequency groups**. In this method we sum deltas, **but** we apply them to the base position between groups.
So the **base** position here is **deformed** by previously applied groups.

And we go in **descending frequency order**, which is **important**.
This is so that small waves get deformed by large waves, as would make sense in reality. The bigger the wave, the more inertia.
I also found this method to be much less sensitive to **pinching** at the **crests** than simply summing deltas,
And still allowed for waves to be deformed by each other, like with the sequential method,
but in a more **controllable** manner.



FROM SHORE TO HORIZON

Deformation

Localized Waves

- Directional
- Follows curvature of shoreline

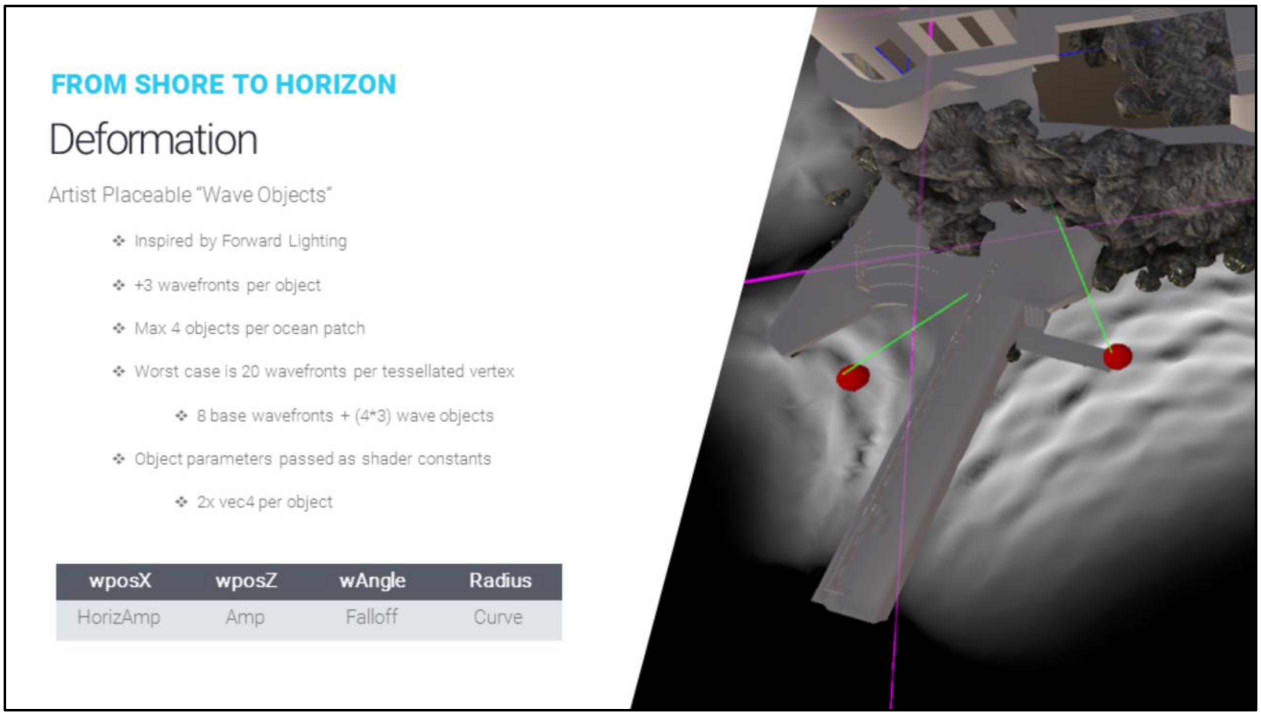
Parametrize Shore Mesh, Baked Data?

- Requires a lot of unique assets
- Results not great
- Local control is tedious

For **coastal** waves, you want to have more **directional** waves, fitted to the **curvature** of the **shoreline**.

To do this we **first** looked into **parametrizing the mesh**. But that would require a lot of unique assets and the results were not great. It was also difficult and tedious to have localized control for things like intensity, speed, etc.

We also wanted a **friendlier** process than baking into texture assets.



FROM SHORE TO HORIZON

Deformation

Artist Placeable "Wave Objects"

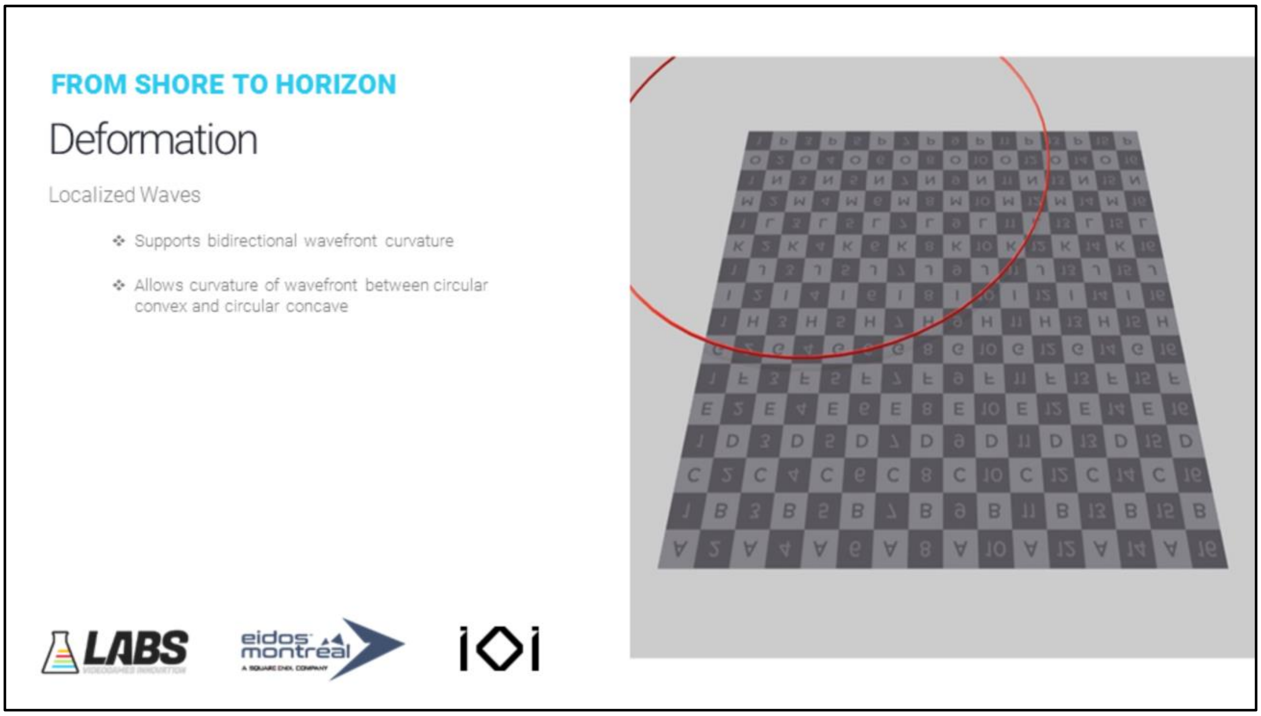
- Inspired by Forward Lighting
- +3 wavefronts per object
- Max 4 objects per ocean patch
- Worst case is 20 wavefronts per tessellated vertex
  - 8 base wavefronts + (4*3) wave objects
- Object parameters passed as shader constants
  - 2x vec4 per object

wposX	wposZ	wAngle	Radius
HorizAmp	Amp	Falloff	Curve

Inspired by **Forward Lighting**, we had the idea to create a simple system of **artist-placeable "wave objects".**

These objects work a bit like forward **lights** in the shader.
Each wave object adds **3 wave samples per vertex**, and the shader allows for **up to 4 wave objects per mesh**.
The **worst case** scenario is therefore **20 wave samples** (8 omnidirectional + 4 * 3 localized) per tessellated vertex.

Each wave object holds a set of parameters and passes these to the shader as constants. This allows the user to control area of effect size (radius), direction, wave amplitudes and curvature of the wavefront.



FROM SHORE TO HORIZON

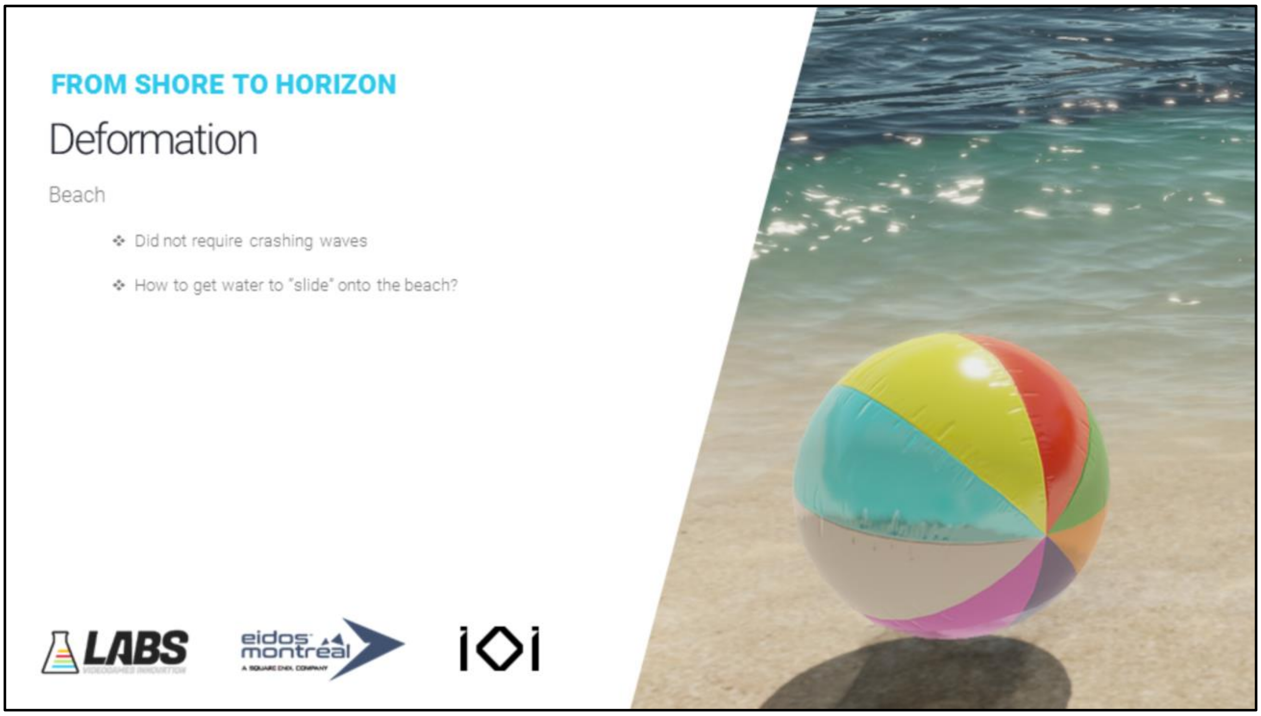
Deformation

Localized Waves

- Supports bidirectional wavefront curvature
- Allows curvature of wavefront between circular convex and circular concave

In order to support **curved** coastlines, it is possible to **warp** the wavefront along the wave direction:

The wave curve parameter allows for **bidirectional** warping in order to obtain a **convex** or **concave** curved wavefront.
This also helped to reduce the number of wave objects necessary to follow the shoreline.



FROM SHORE TO HORIZON

Deformation

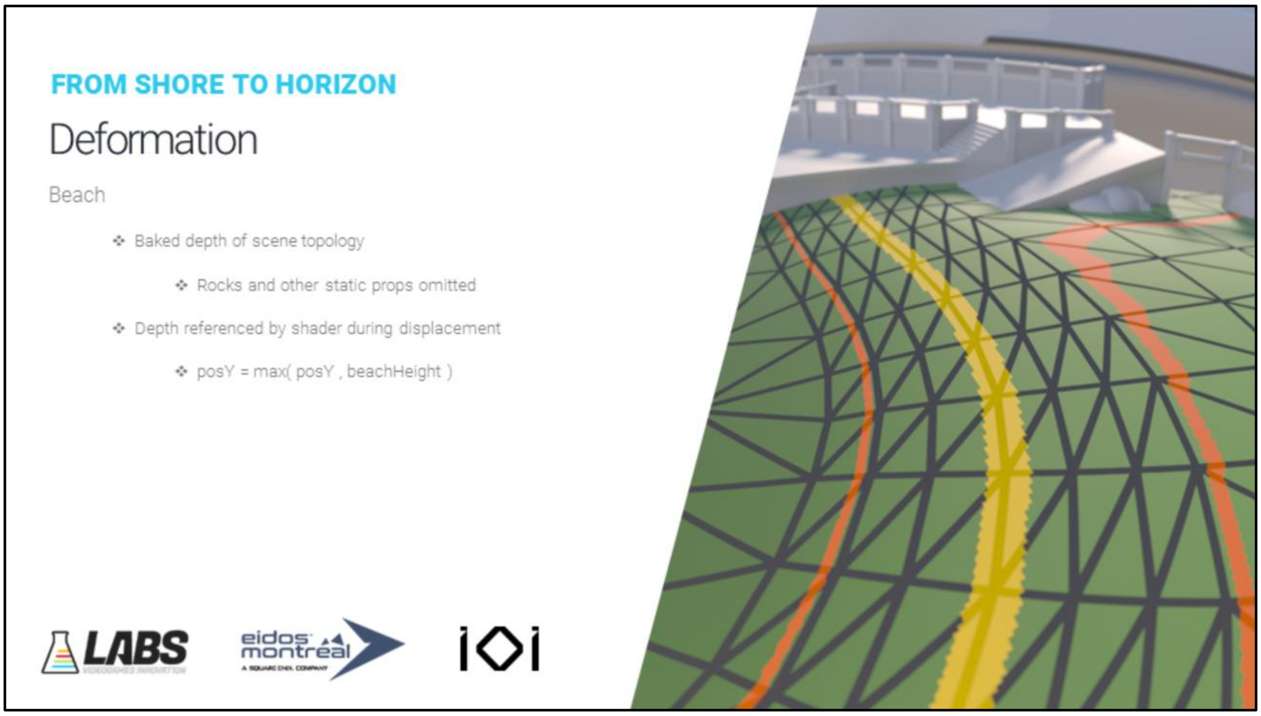
Beach

- Did not require crashing waves
- How to get water to "slide" onto the beach?

So we **solved** the tessellation issue for **arbitrary** geometry

And we **dont require crashing waves**, which makes things easier

But how to get the water to **slide** onto the beach?
The player can walk along the beach, right up to the water, so just letting the water clip is not an option.
Attenuating the deformation to a static position along the shore is common, but I wanted to go for that sliding effect.



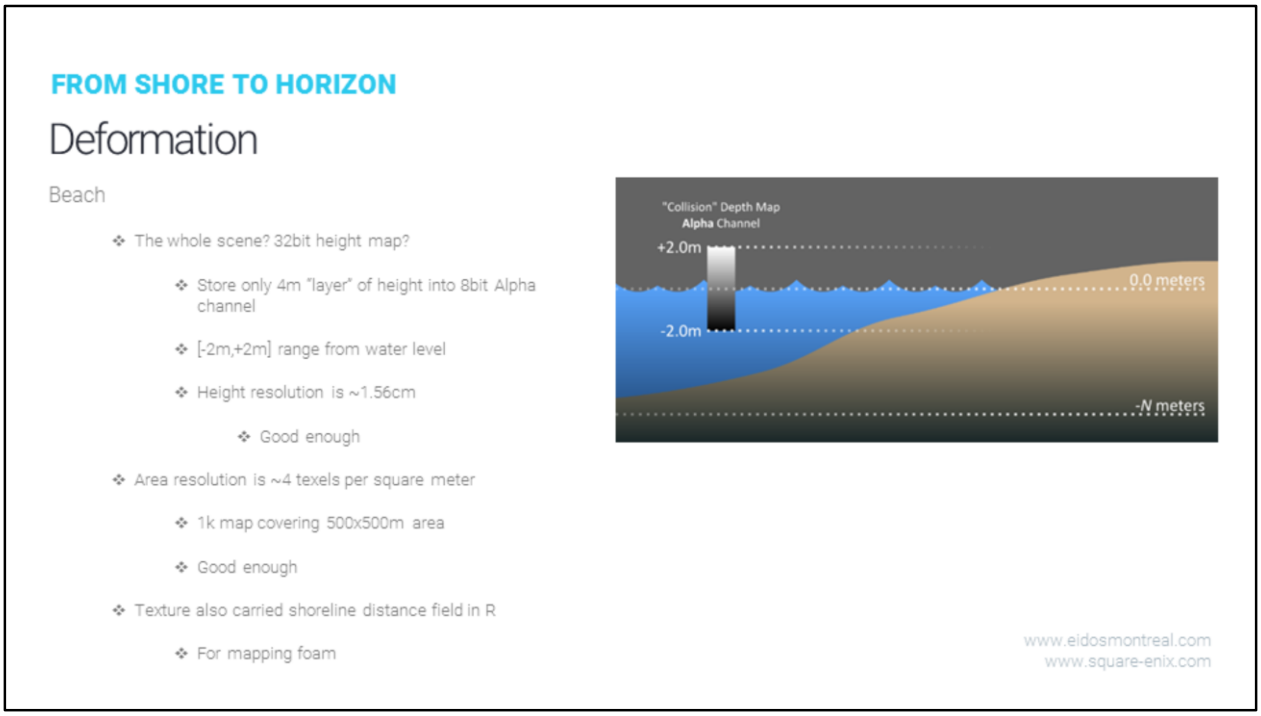
FROM SHORE TO HORIZON

Deformation

Beach

- Baked depth of scene topology
    - Rocks and other static props omitted
- Depth referenced by shader during displacement
    - posY = max(posY , beachHeight)

And to do this,
The beach interaction relies on a **baked depth slice** of the scenery, which is sampled by the ocean shader for the **beach variant** and allows the vertices to **know the height** of the beach at their location during **displacement**. This allowed to have the ocean mesh “**slide**” onto the beach mesh when moving.
This is as simple as selecting the **maximum height** between the **displaced vertex** and the **sampled beach height.**

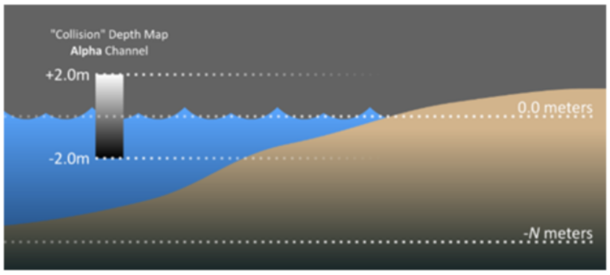


FROM SHORE TO HORIZON

Deformation

Beach

- The whole scene? 32bit height map?
  - Store only 4m "layer" of height into 8bit Alpha channel
  - [-2m,+2m] range from water level
  - Height resolution is ~1.56cm
    - Good enough
- Area resolution is ~4 texels per square meter
  - 1k map covering 500x500m area
  - Good enough
- Texture also carried shoreline distance field in R
  - For mapping foam



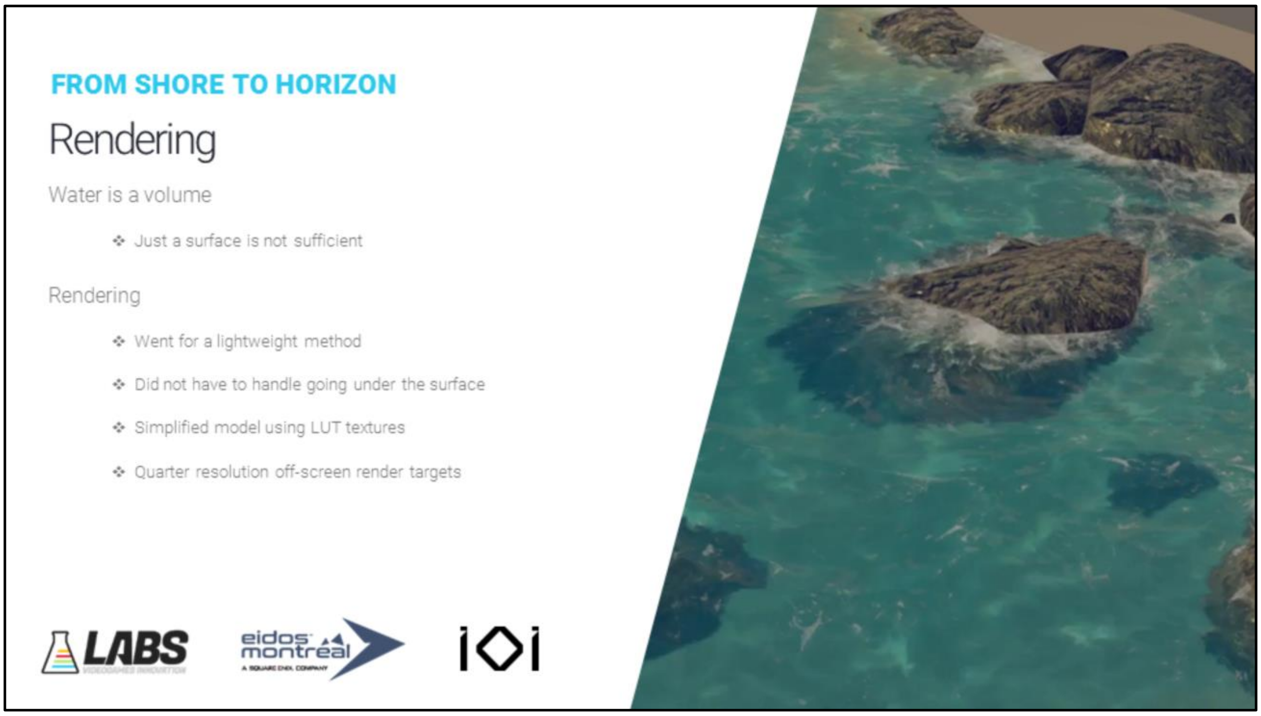
www.eidosmontreal.com
www.square-enix.com

A **full scene height** map sounds a bit **intense**, but we know the water height, so from there we can cut around that and **maximize depth resolution** in the small strip of height that we really need.

The height map only stores a **4m wide layer of height**, a 2m spread from the water height, which allows for an acceptable height resolution of ~1.56cm when packed into an 8bit texture alpha.

For **area**, roughly 4 texels per square meter seemed sufficient.

Since we now have this texture covering the map, we also encode a distance field from the shoreline for use when masking foam.



FROM SHORE TO HORIZON

Rendering

Water is a volume

- Just a surface is not sufficient

Rendering

- Went for a lightweight method
- Did not have to handle going under the surface
- Simplified model using LUT textures
- Quarter resolution off-screen render targets

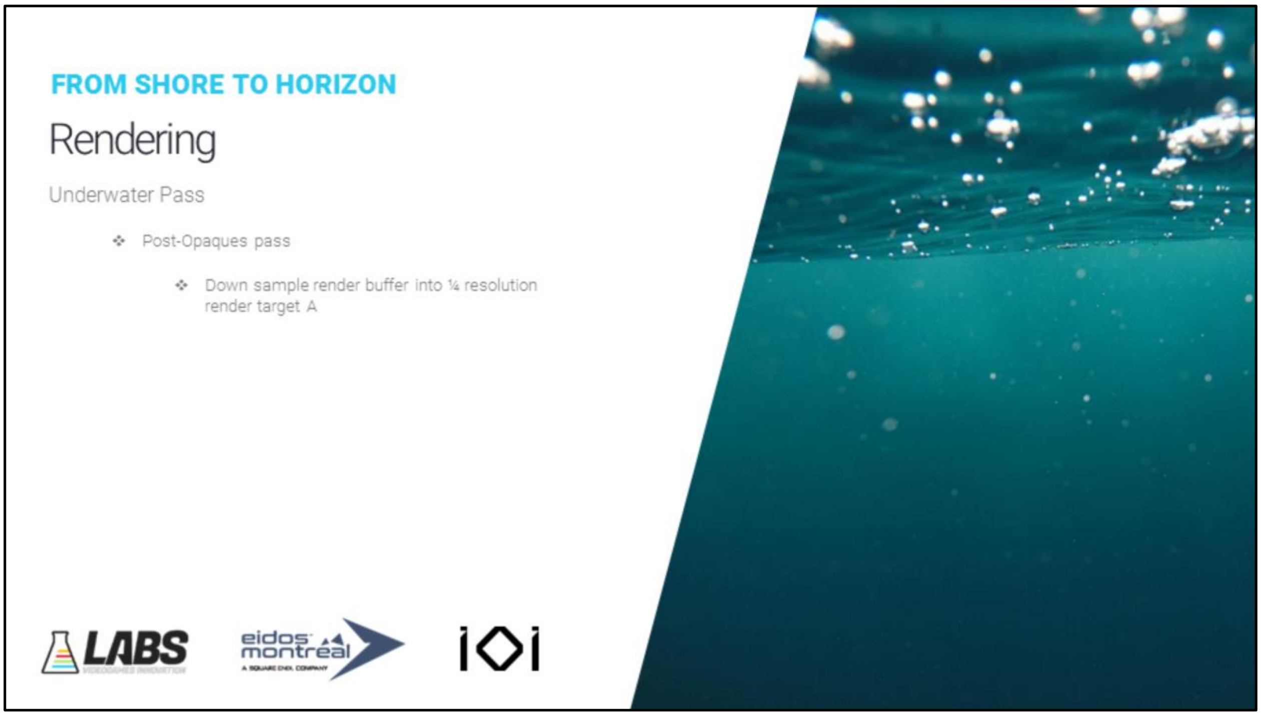
OK, onto rendering.

Water is a **volume**,
And because of that a lot of the shading **effects** are not on but rather **under** the surface. So if not volume rendering, which in games is probably the case, you have to **at least process** what's **behind** the water surface.

As per the **requirements**, we went for a **lightweight** method,
And thankfully we did not need to manage going into or under the water surface.

Shading **parameters** were mostly simplified to **absorption** and **scattering**, represented using a **LUT** texture mapped to linear view depth.

The shader also made use of a **quarter resolution render target** for **underwater** geometries, allowing blurring, refraction, caustics and optionally dispersion.



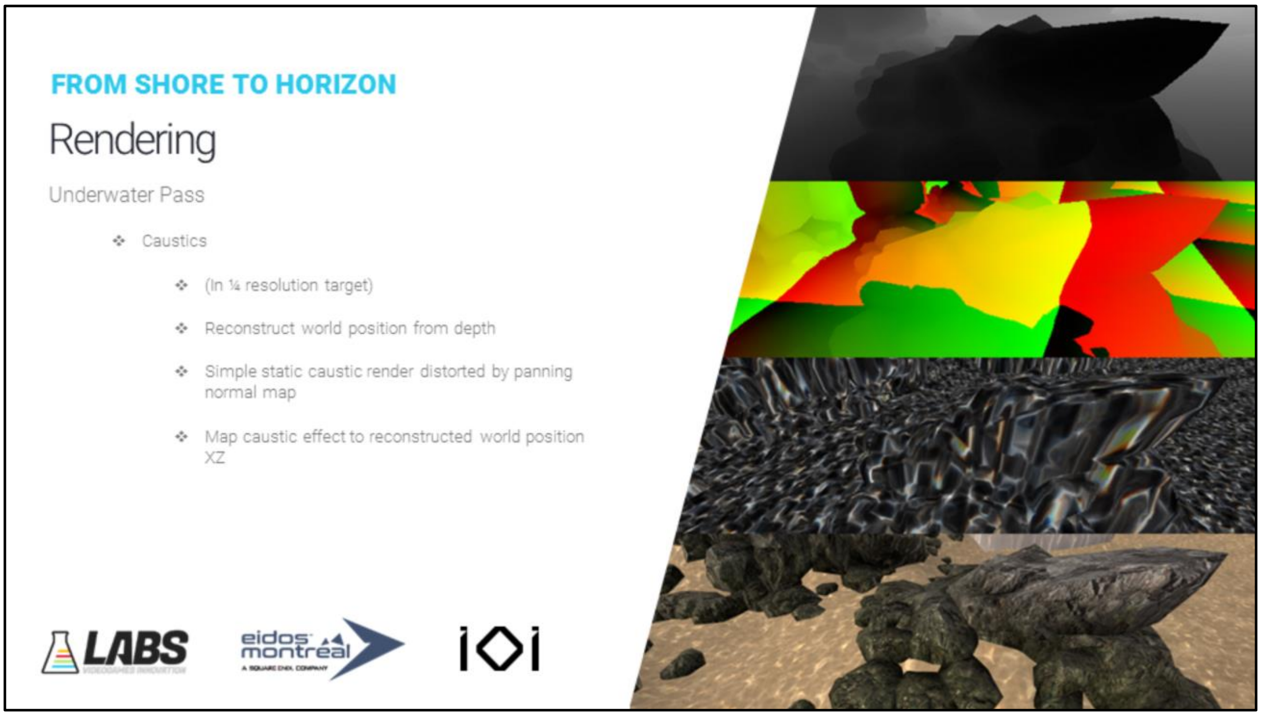
FROM SHORE TO HORIZON

Rendering

Underwater Pass

- Post-Opaques pass
  - Down sample render buffer into ¼ resolution render target A

For the underwater plate,
we render the scene into a **quarter resolution** offscreen buffer before the translucent passes.



FROM SHORE TO HORIZON

Rendering

Underwater Pass

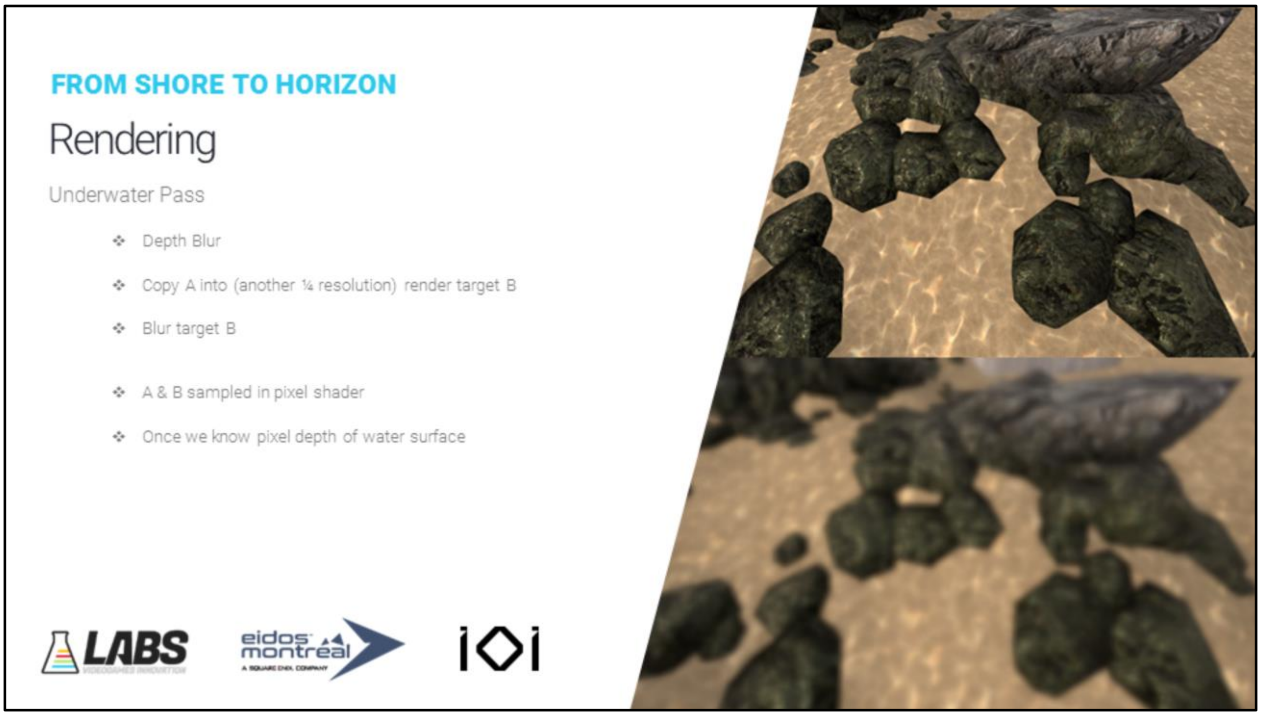
- Caustics
  - (In ¼ resolution target)
  - Reconstruct world position from depth
  - Simple static caustic render distorted by panning normal map
  - Map caustic effect to reconstructed world position XZ

From here we **reconstruct world position** from depth in order to **generate world coordinates** over all of the scene geometry

The **caustic** effect itself is very **simple**; its a static caustic render, just a single image, distorted by a panning low frequency normal map. It does the job pretty well, and besides the effect will be further perturbed by the surface refraction effect.

We then use the generated worldpos as Uvs in order to **project** our caustics effect over everything.

We don't really care about **masking** or **distance** here :
Elements above the water obviously wont be visible in the underwater image.
And elements far off in the distance will be obstructed by absorption.


FROM SHORE TO HORIZON

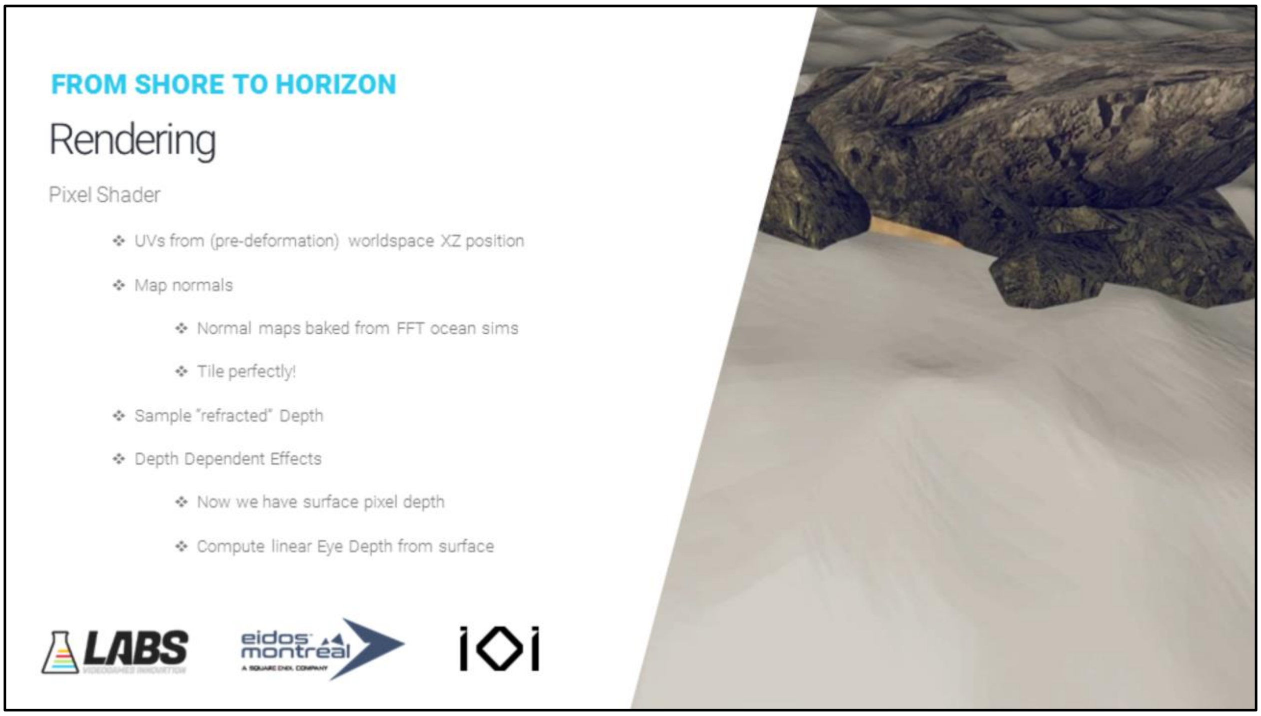
Rendering

Underwater Pass

- Depth Blur
- Copy A into (another ¼ resolution) render target B
- Blur target B
- A & B sampled in pixel shader
- Once we know pixel depth of water surface

After the **caustics** are applied, we make a **copy** of the texture and **blur** it, so we have a blurred and unblurred version of our underwater, which we can later **blend** using **depth**.

Further effects will need to be applied in the surface pixel shader, since we will need the surface pixel location.
This is because we cant just use depth, we need **depth from surface**.



FROM SHORE TO HORIZON

Rendering

Pixel Shader

- UVs from (pre-deformation) worldspace XZ position
- Map normals
  - Normal maps baked from FFT ocean sims
  - Tile perfectly!
- Sample "refracted" Depth
- Depth Dependent Effects
  - Now we have surface pixel depth
  - Compute linear Eye Depth from surface

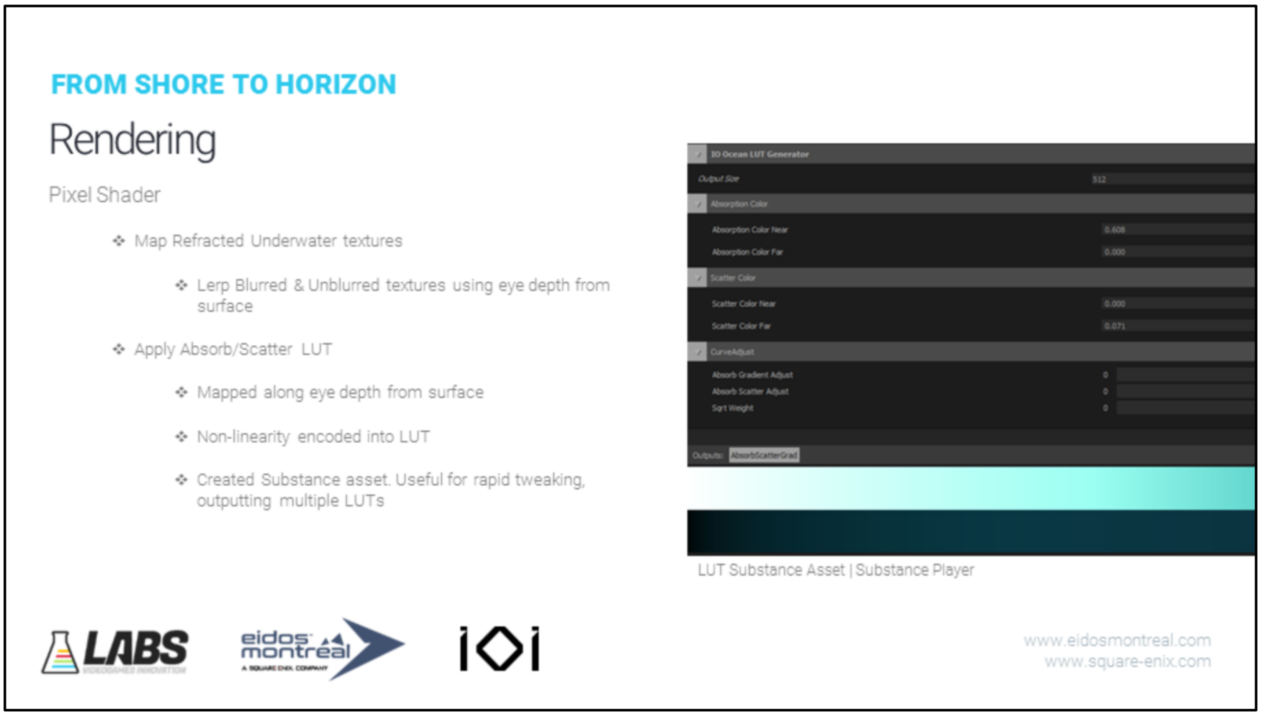
When rendering the surface in the pixel shader

We get 2 **UVs** based on world position. **Pre & Post deformation**.
Pre-deformation coordinates will (*unintuitively*) be deformed by wave motion, so this is what we use to map surface textures like foam and normals.

I do use an **offline FFT ocean** surface in order to **bake normal map**s for water. This really makes the best water normal maps I find.
The **periodic nature** of FFTs works to our advantage when baking normal maps : they **tile** perfectly.
We use these wave normal maps to represent the highest frequency waves on top of the deformation waves.

Once we have normals, we can (quote-unquote) **refract** our **UVs** for sampling our **scene depth** and **underwater plates**.

In the pixel shader we also have the **pixel depth**, so we can compute **eye depth from surface**. Obviously if you want to accumulate an effect like absorption, you don't want to start from the eye, but rather the point at which the eye ray **intersects** the surface.

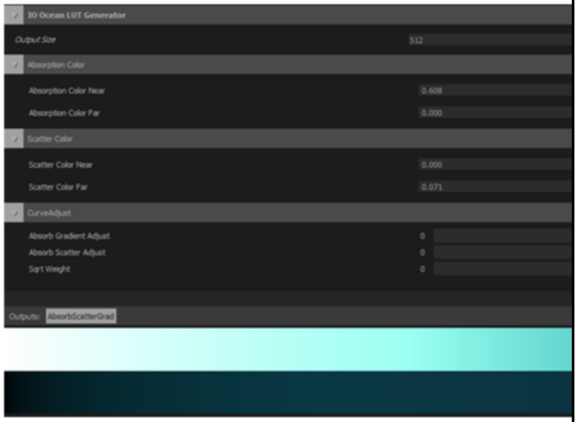


FROM SHORE TO HORIZON

Rendering

Pixel Shader

- Map Refracted Underwater textures
  - Lerp Blurred & Unblurred textures using eye depth from surface
- Apply Absorb/Scatter LUT
  - Mapped along eye depth from surface
  - Non-linearity encoded into LUT
  - Created Substance asset. Useful for rapid tweaking, outputting multiple LUTs



```
IO Ocean LUT Generator
Output Size                 512
Absorption Color
  Absorption Color Near     0.608
  Absorption Color Far      0.000
Scatter Color
  Scatter Color Near        0.000
  Scatter Color Far         0.071
CurveAdjust
  Absorb Gradient Adjust    0
  Absorb Scatter Adjust     0
  Sqrt Weight               0
Outputs: AbsorbScatterGrad
```

LUT Substance Asset | Substance Player

www.eidosmontreal.com
www.square-enix.com

With linear distance from surface, we then **map both blurred and unblurred underwater plates** and blend them along depth to a given input distance.

So why **blur** the water, is that **PBR**?
Probably not.
But blurring I find **helps** a lot. Real ocean water has a lot of effects going on. Underwater haze, micro refraction from very small surface ripples and and under the surface from microscopic bubbles, dispersion and other difficult effects.

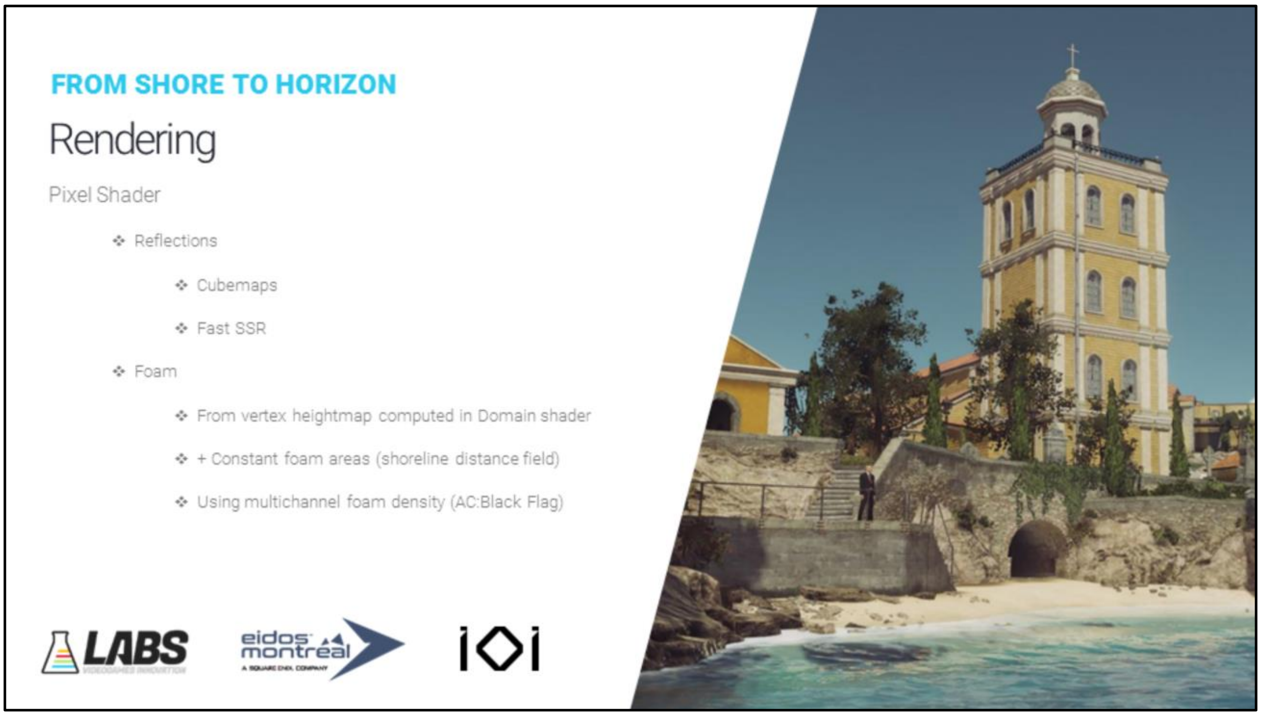
I find blurring works visually to **convey** the impression of a different, **heavier medium** below the surface.
Without I find often water rendering looks like just air behind a deforming glass pane.

So after this, we apply our **Absorb/Scatter LUT**

Absorption works along eye depth from surface, and is multiplicative. The far point is **black**, as that is where light would be mostly absorbed underwater. For ocean water this depends on the wavelength, but 200m is generally the point at which almost no light passes in open water.

Scattering is a more **complex** effect and doesn't really work along eye depth, but for simplicity and efficiency, that's what we did.

A downside to LUT textures is the iteration rate when tweaking values. In my case I made the LUT as a function in **substance** designer so I could rapidly regenerate the texture when changing the gradients.


FROM SHORE TO HORIZON

Rendering

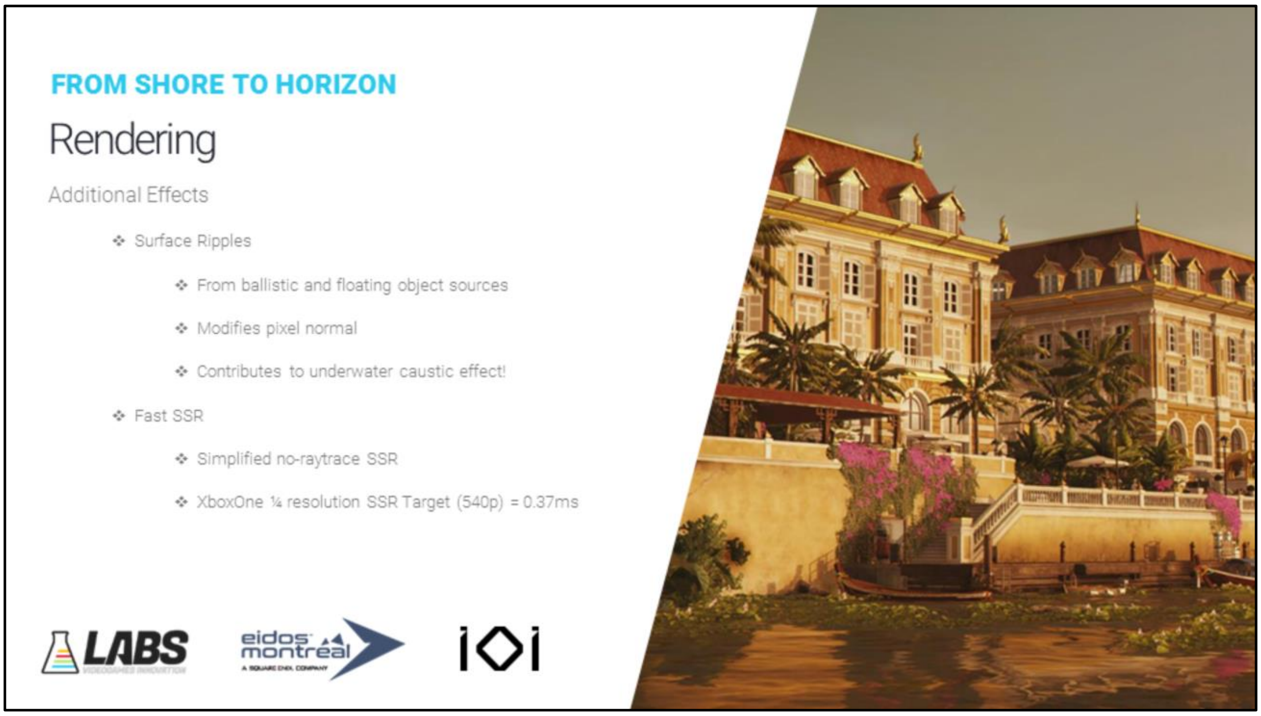
Pixel Shader

- Reflections
  - Cubemaps
  - Fast SSR
- Foam
  - From vertex heightmap computed in Domain shader
  - + Constant foam areas (shoreline distance field)
  - Using multichannel foam density (AC:Black Flag)

We then apply our cubemap **reflections** & **specular** highlights and an SSR pass is done in post.

For **Foam**, its masked using both a shoreline distance field, and a vertex height map computed from our waves.
While accumulating height in the domain shader, total added height is tracked in order to normalize the final height value.

For **texturing**, a trick taken from Black Flag : **Mutliple** densities of foam are packed into a single texture, and mixed together in the shader depending on the grey value of the surface foam mask.



FROM SHORE TO HORIZON

Rendering

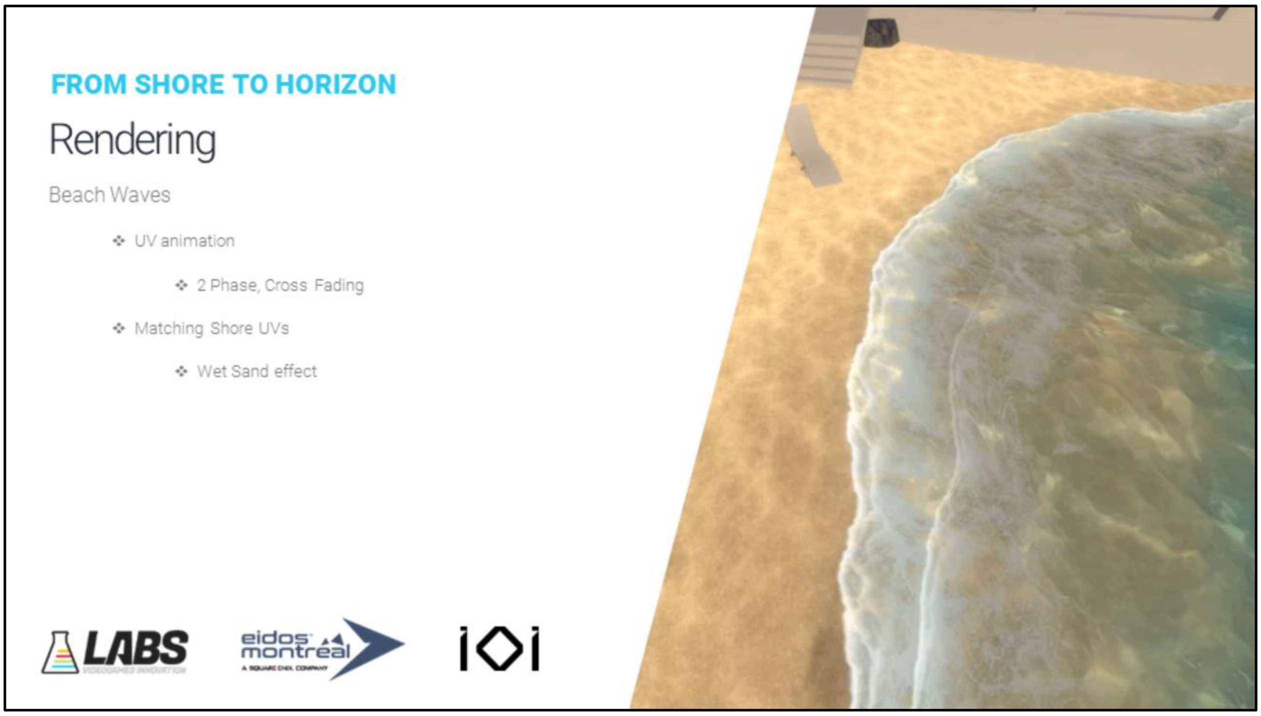
Additional Effects

- Surface Ripples
  - From ballistic and floating object sources
  - Modifies pixel normal
  - Contributes to underwater caustic effect!
- Fast SSR
  - Simplified no-raytrace SSR
  - XboxOne ¼ resolution SSR Target (540p) = 0.37ms

The effect supports dynamic surface **ripples** from interacting floating physics objects as well as **ballistic** impacts. These don’t deform the surface but rather contribute to the pixel normal.

Whats cool though is that they also contribute to the underwater **caustics.**

The **SSR** method mentioned before is a fast, simplified no-raytrace SSR which uses the plane equation of the water surface rather than the real surface and normal.



FROM SHORE TO HORIZON

Rendering

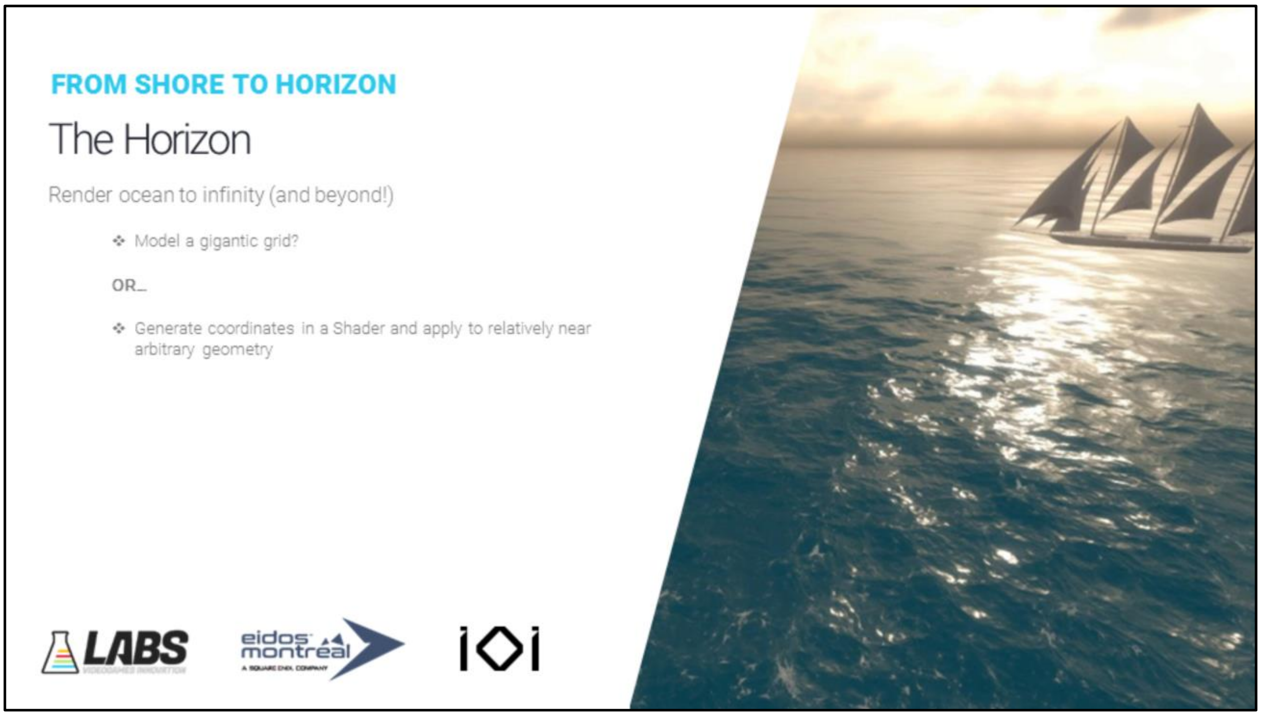
Beach Waves

- UV animation
  - 2 Phase, Cross Fading
- Matching Shore UVs
  - Wet Sand effect

For the **beach** waves, we already have our sliding water effect from the deformation.
And the beach mesh also has **UVs** along that strip of polygons.
So to **complete** the effect, a **two-phase cross fading uv animation** is used.

To **desync** the animation along the **length** of the shoreline, the cosine of UV.x **offsets** the **phase** of the animation.

And because we **matched** the **topology** of the water edge and beach, and have **matching UVs,** we can **implement** the **same animation** on the sand in order to achieve a "**wet sand**" effect.



FROM SHORE TO HORIZON

The Horizon

Render ocean to infinity (and beyond!)

- ❖ Model a gigantic grid?

OR…

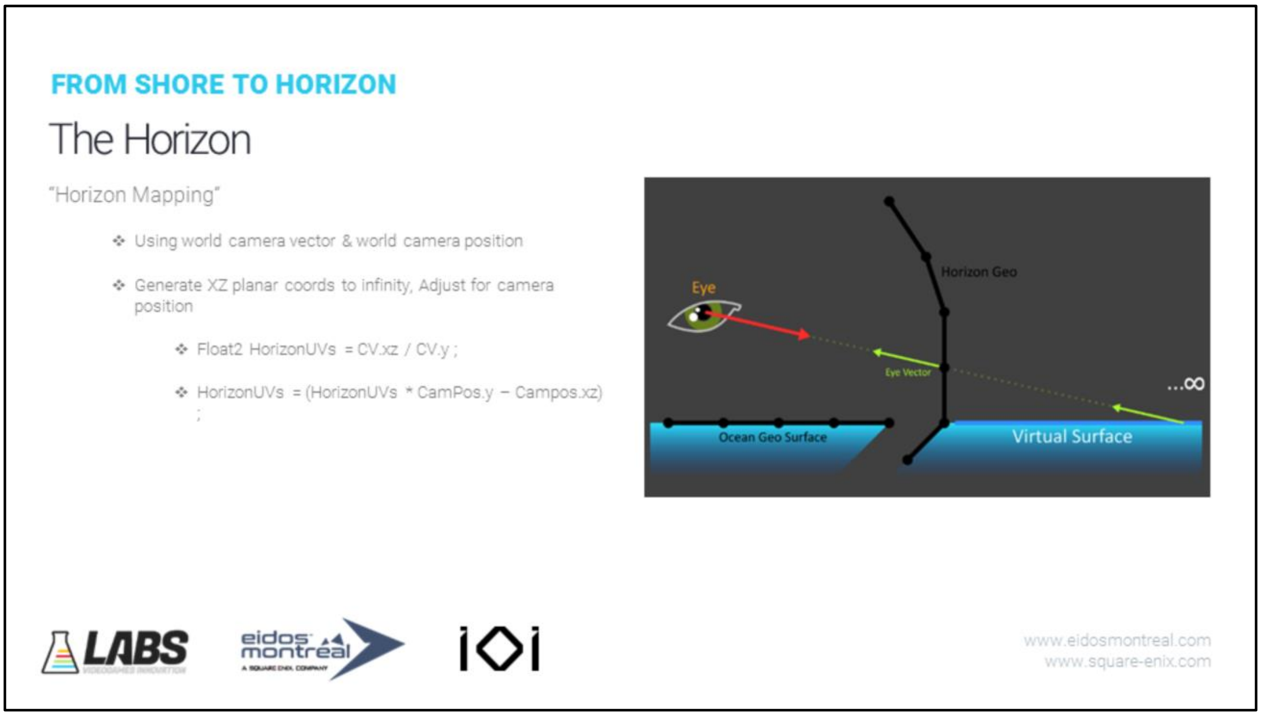
- ❖ Generate coordinates in a Shader and apply to relatively near arbitrary geometry

For the Horizon…

Although the **base** geometry **extended** out several hundred meters, the edge of the mesh could be seen especially from certain **elevated vantage** points.

One traditional solution is to simply create a **gigantic grid** and scale it until you just cant see the edges anymore.
This however requires a **large far clip value** which does affects **depth distribution.**

So Instead I recycled a trick I used to map procedural skies.

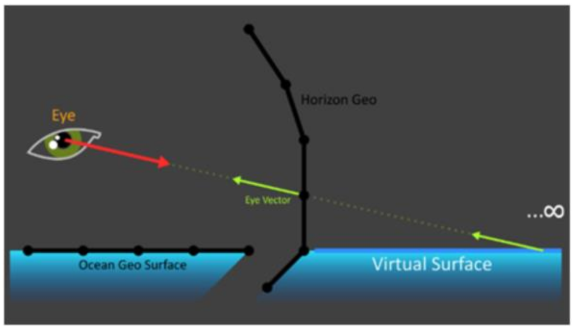


FROM SHORE TO HORIZON

The Horizon

"Horizon Mapping"

- Using world camera vector & world camera position
- Generate XZ planar coords to infinity, Adjust for camera position
  - Float2 HorizonUVs = CV.xz / CV.y ;
  - HorizonUVs = (HorizonUVs * CamPos.y – Campos.xz) ;

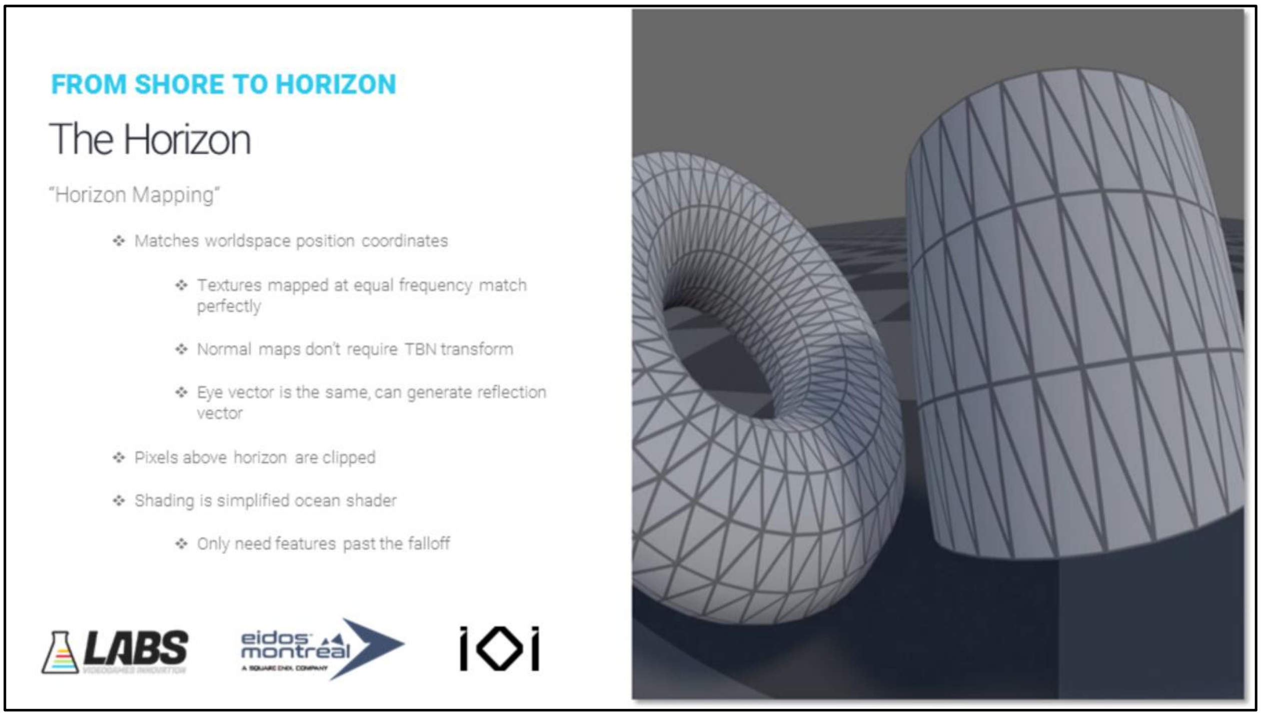


www.eidosmontreal.com
www.square-enix.com

For lack of a better name I dubbed the effect "**horizon mapping**", simply because it generates planar coordinates to the horizon line
but I think the term might **already** used elsewhere in CG. (So if anyone has a better name for it, let me know)

Here we use the world camera vector and world camera position.
(and assuming a Y-up coordsys) we **divide** Camera Vector XZ by Y which gives us infinite planar coords on the XZ plane.

They're **camera-centric** though, so we adjust using world camera position to **compensate**.



FROM SHORE TO HORIZON

The Horizon

"Horizon Mapping"

- Matches worldspace position coordinates
    - Textures mapped at equal frequency match perfectly
    - Normal maps don't require TBN transform
    - Eye vector is the same, can generate reflection vector
- Pixels above horizon are clipped
- Shading is simplified ocean shader
    - Only need features past the falloff

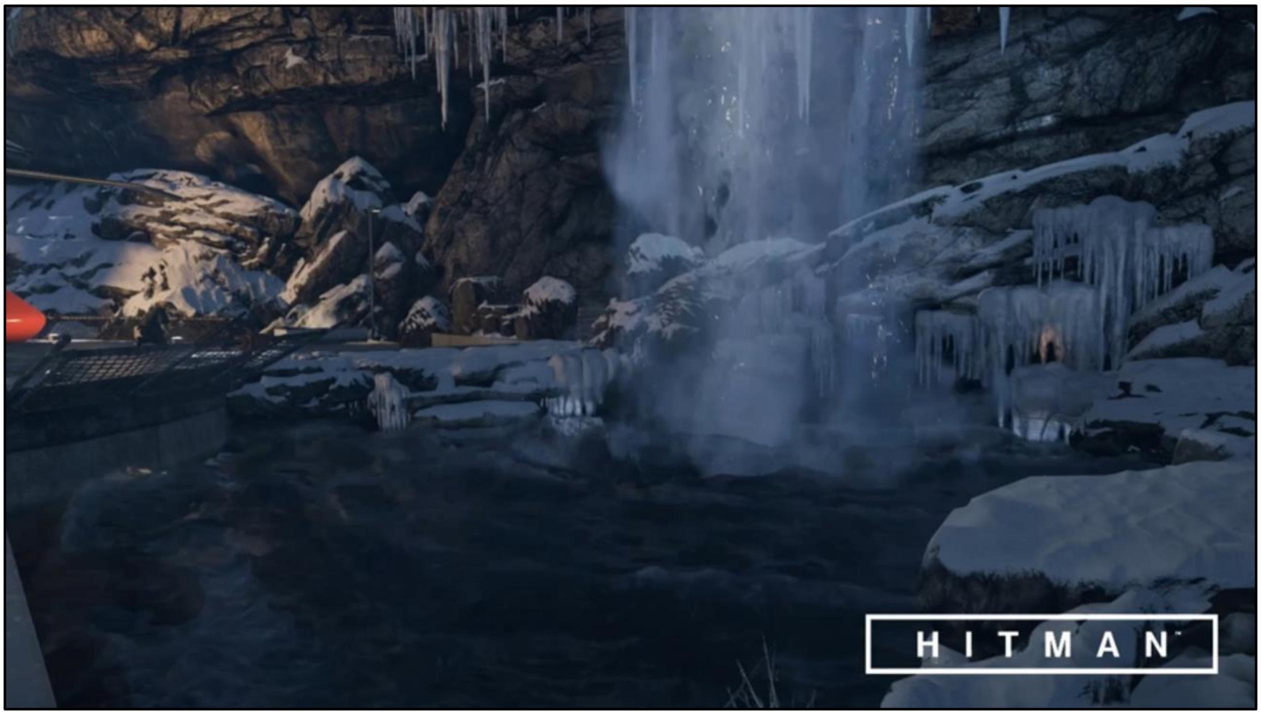
The result is planar coordinates that **match world coordinates** perfectly,
so if you are using world coords to map textures, such as is the case for our ocean, the textures mapped to the horizon mapping should line up perfectly and there will be no visible seam (assuming the pixels are shaded the same way of course).

Normal maps here don't require transforming, other than swapping Y for Z or other coordinate adjustments, since were already aligned with the world axis.

The world camera vector on the support geometry should be exactly the **same** as on our virtual plane, so with the mapped normal we can get a reflection vector on the virtual surface as well. This gives us enough to shade the surface and match our real ocean geometry.

Because the effect's support geometry can be **relatively close**, the **camera far plane** doesn't need to be at a very high value which is better for your scene's **depth distribution.**

Pixels above the horizon line are **clipped** simply using Camera Vector Y,
and the virtual plane is **shaded** using a **simplified** version of the **ocean** shader.

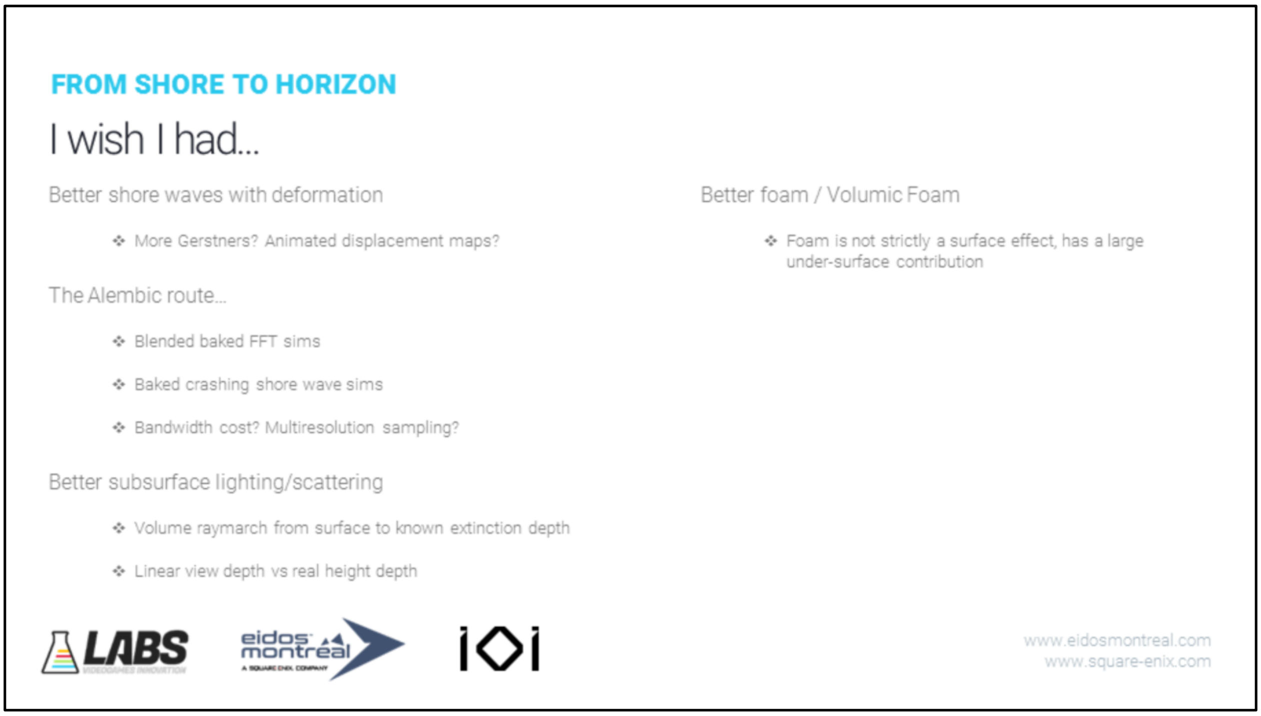


Reception of the effect by the team was pretty good. The **placeable wave modifiers** were especially liked.
Here we can see them used as waves from the waterfall.

The effect proved **versatile** enough to be used in many contexts across the game.

One nice example here is the **bubbling** water in the natural spring.
I believe they cleverly used an **inverted** Gerstner function here.
It looks pretty good!

Many **size** contexts as well,
Anything from small pools to lakes to oceans



FROM SHORE TO HORIZON

I wish I had…

Better shore waves with deformation

- More Gerstners? Animated displacement maps?

The Alembic route…

- Blended baked FFT sims
- Baked crashing shore wave sims
- Bandwidth cost? Multiresolution sampling?

Better subsurface lighting/scattering

- Volume raymarch from surface to known extinction depth
- Linear view depth vs real height depth

Better foam / Volumic Foam

- Foam is not strictly a surface effect, has a large under-surface contribution

www.eidosmontreal.com
www.square-enix.com

Of course eventually we had to **ship** the feature… with still a lot of interesting **ideas** to **explore**.
And lots of ideas come after a **step back**, and id like to **share** some of those;

Better shore waves.
Our effect did not call for surf waves or crashing waves
But still I wish it had a bit more deformation there.
More gerstners? Probably difficult to control for shorewaves.
Animated displacement maps? Maybe a better idea given our uv animation is already there.

Alembic
Not really an option at the time, but Alembic offers some interesting avenues to explore.
Mesh caches could allow to precompute and play back complex simulations and surface deformations, deferring the computation cost to bandwidth.
Precomputed crashing wave simulations would be nice.
Lots of things to solve and explore on this end though. Bandwidth cost would be a question, how to apply to a mesh with dynamic resolution would be another. How to blend Alembic deformations together or with other deformations.

Better subsurface lighting/scattering
Absorption works with linear eye depth from surface, but not scattering, that's a complete **hack**. A volumic ray march from surface would have been nice here, maybe in the ¼ resolution buffer using a uniform water level.

Finally
Better Foam
If you look at photos of ocean surf, you see that water is bright where the foam is.
And this is that foam is really not strictly a surface thing like it is usually represented;
There's a lot of air bubbles mixed in just under the surface.
So I don't think you can really get realistic foam until you also capture this effect.
Maybe a raymarch through a foam heightmap from the surface?
Or maybe a simple parallax effect with blurred foam would do.

So That's all for me
Ill now pass the floor back to Jean Normand

FROM SHORE TO HORIZON

Thank you!

Collaborators

Eidos-Montréal:	Io-Interactive:
David Gallardo	Jonas Meyer
Nicholas Fréchette	Martin Vestergaard Kümmel
Samuel Delmont	Martin Bjarke Kragh Nielsen
Anton Kai Michels	Joakim Mellergård

Special Thanks

David Anfossi
Julien Bouvrais
Frédéric Richard
Isabelle Tremblay

Billie, Luka, Milo & Sarah X2

Of course, we haven’t done this alone and we’d like to thanks some great brains who helped us achieve this all.

WE ARE HIRING
eidos montreal
A SQUARE ENIX COMPANY
SQUARE ENIX
SEE OUR OPENINGS AT WWW.EIDOSMONTREAL.COM/JOBS