



Maximizing Graphics Performance with Flexible Virtualized Geometry

Alexis Vaisse

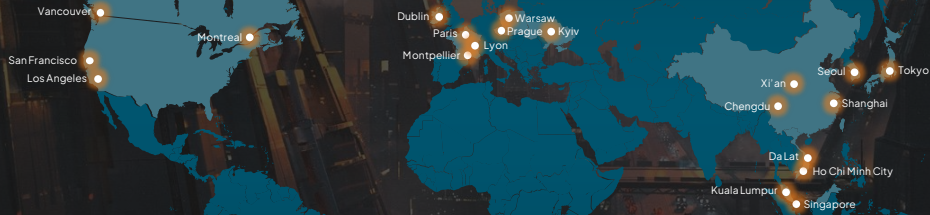
Senior Technical Director

Marios Michaelides

Engineering BU Director

VIRTUOS CONFIDENTIAL

One of the largest independent developers



3700+
PEOPLE

22
LOCATIONS

900+
CLIENTS

20
YEARS

1500+
TITLES

VIRTUOS CONFIDENTIAL

One of the largest independent developers





We're going to start by explaining why we took the decision to develop this technology.

The second part, which is the biggest one, will cover the technology in more detail.

One specificity we have at Virtuos is that we work on different game engines.

Developing the technology for multiple game engines was a must have. We'll see briefly how we achieved this.

We are constantly improving the technology. I will conclude with an overview of the subjects we are currently working on.

WHY USE VIRTUALIZED GEOMETRY?



Artists' dream

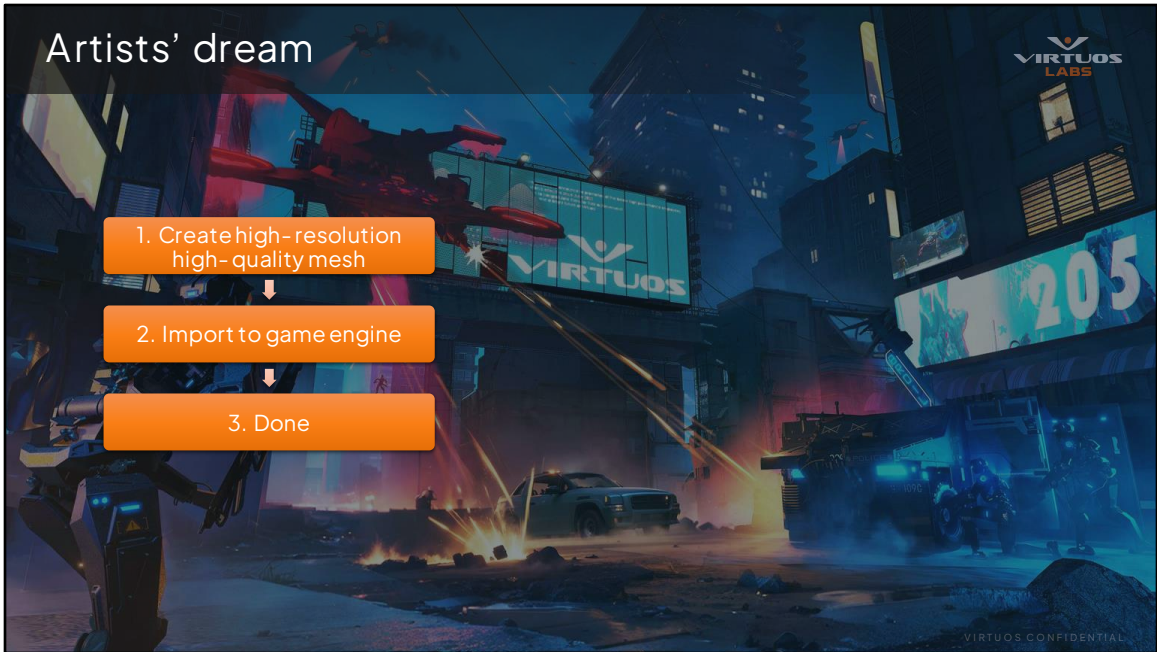
1. Create high-resolution
high-quality mesh



2. Import to game engine



3. Done



Artists' dream vs reality



VIRTUOS
LABS

CONFIDENTIAL



What can we do to improve this?

We want artists to only work with high-resolution meshes.

And it's up to the game engine to manage to get good performance and respect the memory budget.

Don't forget that we work with multiple game engines.

So, we want the technology to be compatible with different game engines.

It must be compatible with forward rendering, deferred rendering, virtual shadow, cascade shadow, raytracing, various streaming systems, etc.

Expected results



Better quality

- Allow artists to use higher-resolution meshes
- Use better resolution at runtime compared to classical rendering

Save production time

- Spend much less time optimizing meshes

Better performance

- Dynamically adjust LOD level to reach expected performance
- More stable performance compared to classical rendering

Less out-of-memory crashes

- Use a fixed memory budget
- Reduce out-of-memory crashes on consoles during production

VIRTUOS CONFIDENTIAL

We want to reach a better quality because artists will use higher-resolution meshes and the graphics engine itself will use higher-resolution compared to classical rendering.

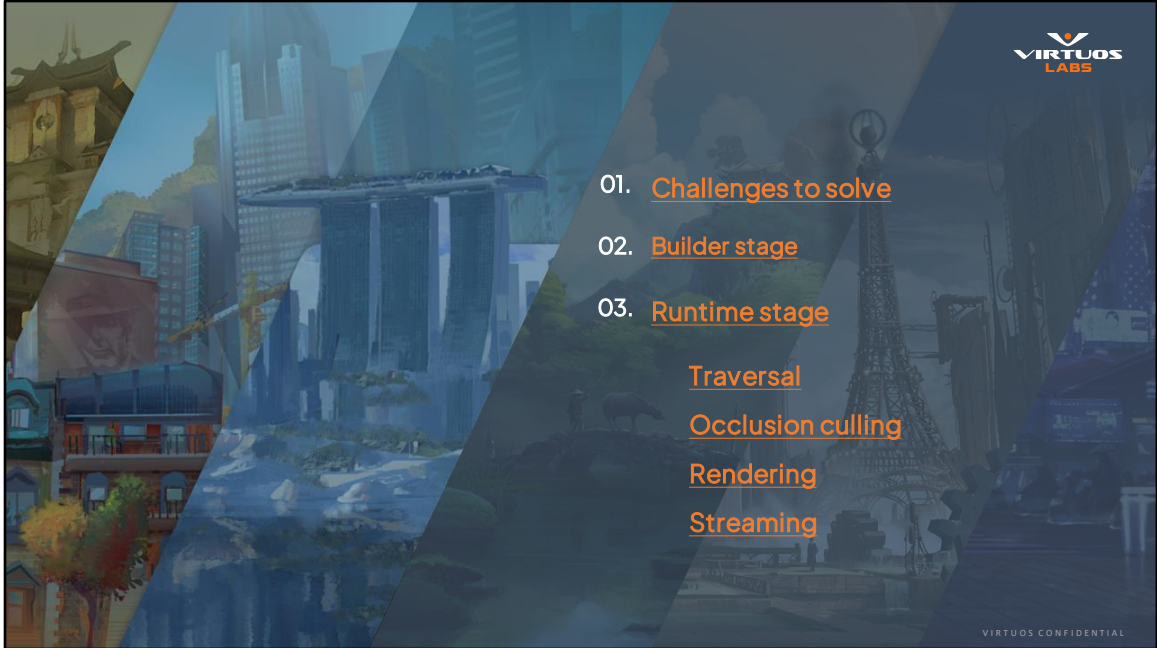
We also want to get better performance and, probably as important, more stable performance compared to classical rendering.

We also expect to save production time because we won't need to optimize meshes, or, at least, we'll spend less time to optimize them.

And the last point is that we'll decrease the number of out-of-memory crashes on consoles during production because we use a fixed memory budget.



OK. We set quite ambitious objectives. Let see what we did to achieve them.



I will start with the different technical challenges to solve, then we'll dive into the different parts of the technology itself. The first one is the builder stage, and the second one is the runtime stage with the traversal, the occlusion culling, the rendering, and the streaming.

Challenges to solve



Buddha statue

4.5-meter
height

200 tons

3.5M triangles

Let's take this wonderful Buddha statue, 4.5-meter height, 200 tons, and 3.5 millions triangles.

Let's get closer.

Challenges to solve

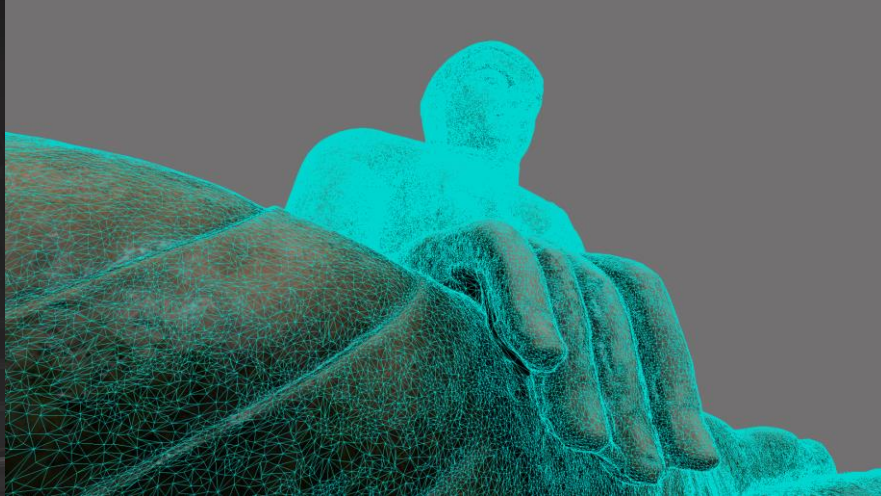
VIRTUOS
LABS



Which **LOD**
to use?

Which LOD should we use to render it?

Challenges to solve



High LOD

Good quality

Poor performance

If we use a higher LOD, we'll get good quality but poor performance.

Challenges to solve

VIRTUOS
LABS



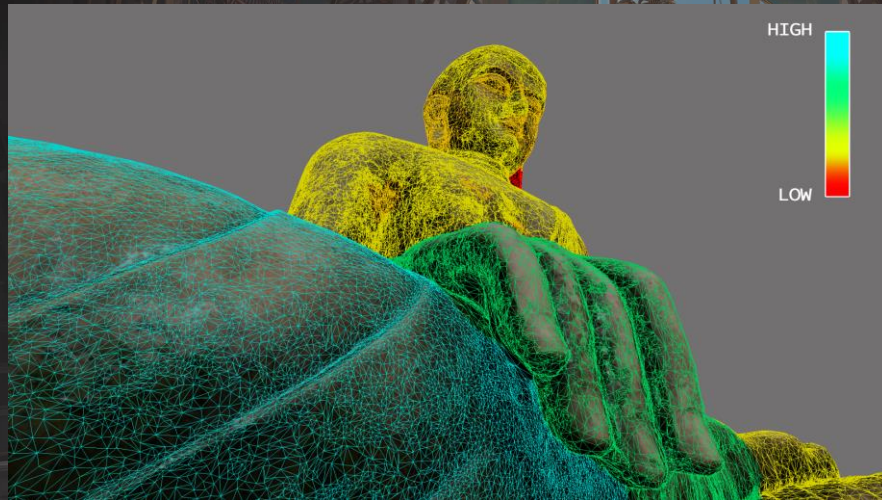
Low LOD

Good
performance

Poor quality

And if we use a lower LOD, we'll get good performance but a poor quality.

Challenges to solve



We want to render an object with different LODs at the same time

I'd like to have this, which is different LODs based on the camera distance.
So, we want to render an object with different LODs at the same time.

Challenges to solve



We want to render an object with **different LODs** at the same time

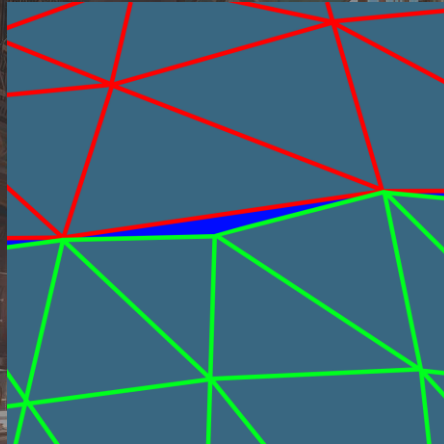
Let's get closer.

What will happen if I use a lower LOD for the upper part of the mesh?

Let's try it.

Challenges to solve

VIRTUOS
LABS



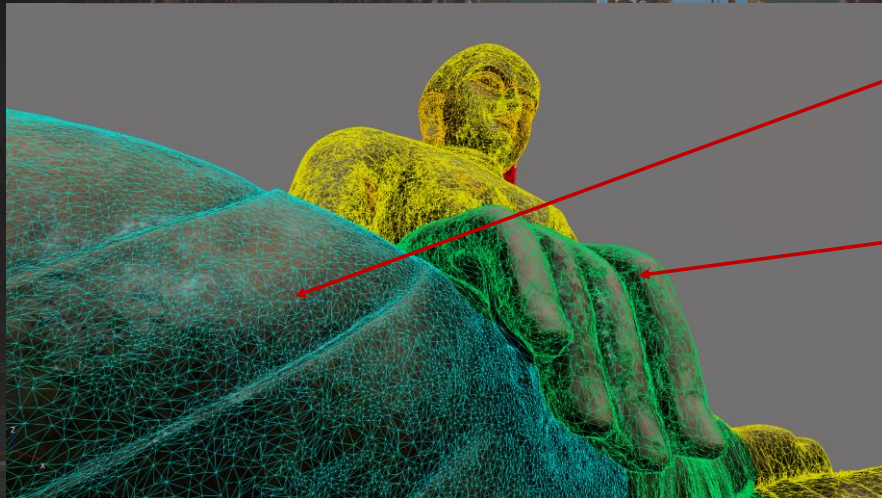
We want to
render
an object
with
different
LODs
at the
same time
with no
seam!

Oops, we get a hole between LODs.

Let's rephrase our objective: We want to render an object with different LODs at the same time... with no seam.

Challenges to solve

VIRTUOS
LABS



Even if this part
is displayed
using the best
LOD,

we don't want
the best LOD to
be loaded for
the rest

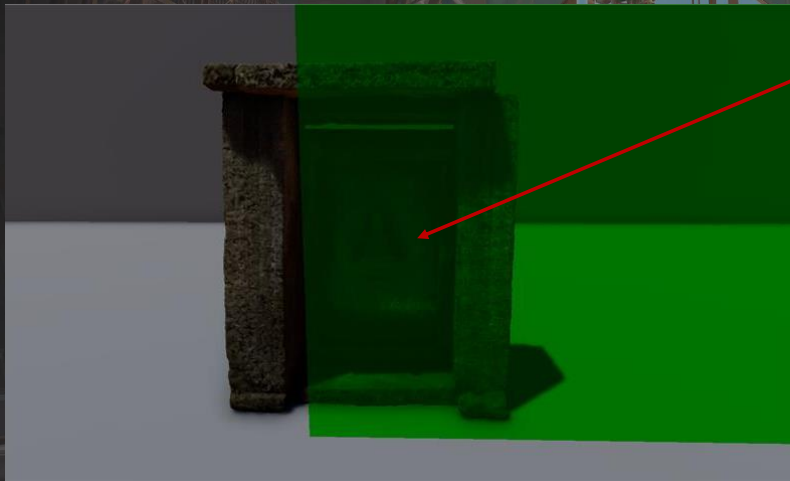
We want
streaming at
sub-object
level

Let's go back to our wonderful statue.

Even if a part of the mesh is displayed using the highest LOD, I don't want the highest LOD to be loaded for the rest of the mesh.

Which means that we want streaming at sub-object level.

Challenges to solve

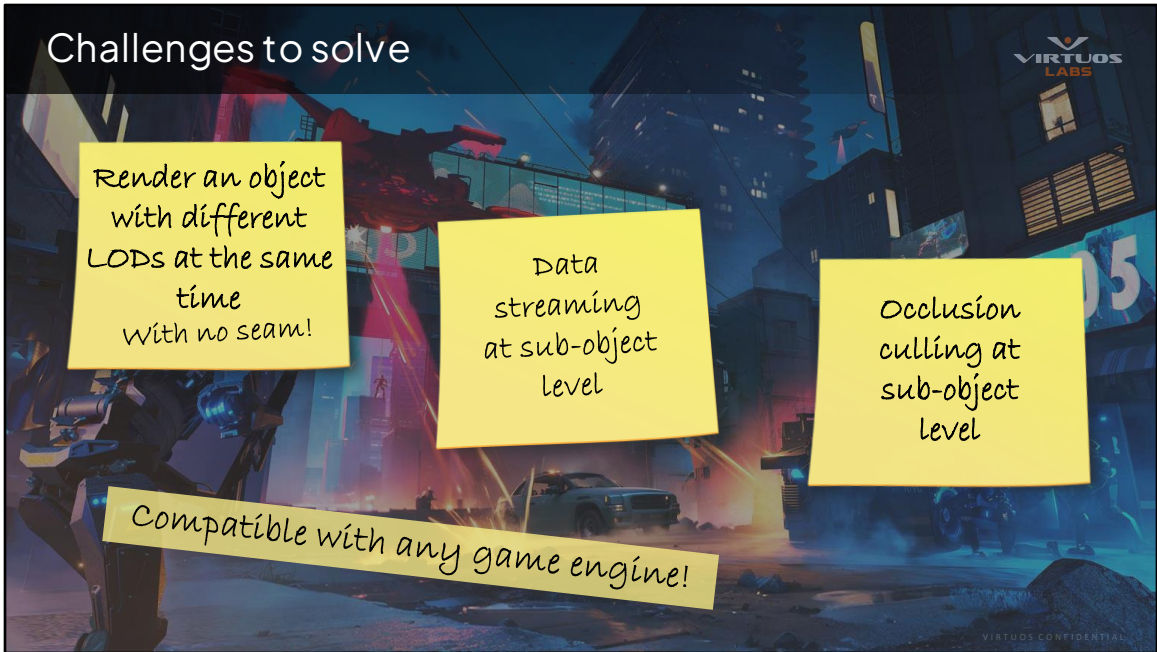


Even if this object is only partly occluded, I would like the occluded part to not be rendered at all to improve performance

We want occlusion culling at **sub-object level**

Even if an object is only partly occluded, I'd like the occluded part to not be rendered at all to improve performance.

Which means that we want occlusion culling at sub-object level.

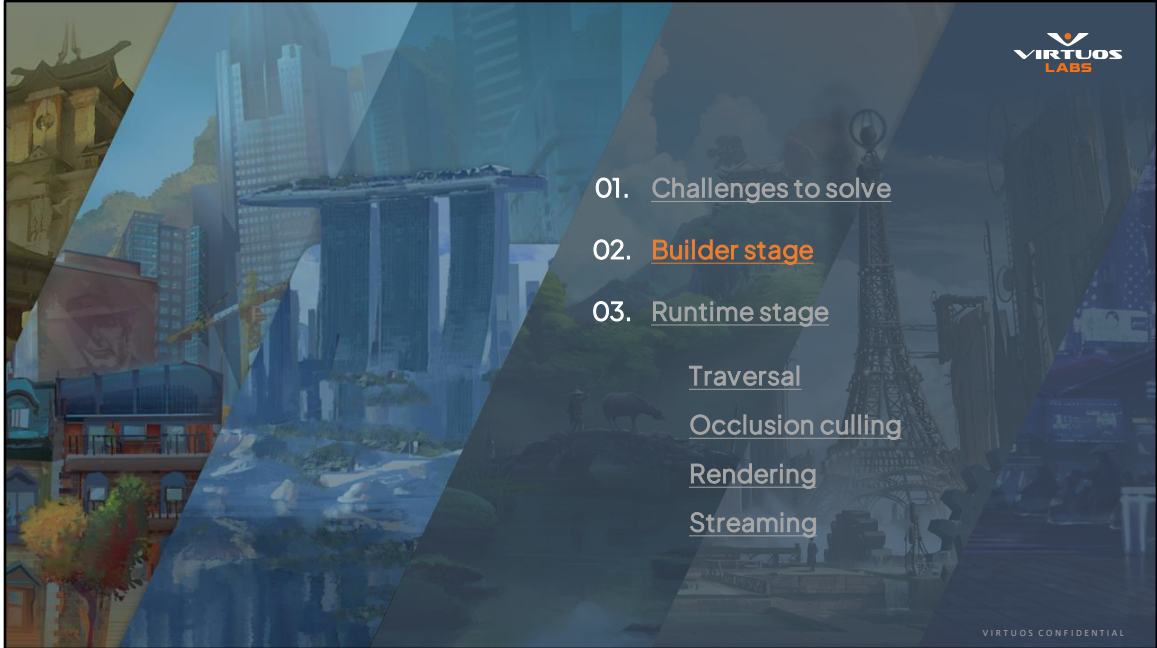


Let me summarize the different challenges that we've identified:

We want to render an object with different LODs at the same time with no seam.

We want data streaming at sub-object level, and occlusion culling at sub-object level as well.

Still compatible with multiple game engines.



The builder stage is usually integrated into the import process.

Builder stage



Step 1: Cluster creation

- We use the **METIS** library to split the mesh into clusters
- Each cluster contains **128 triangles** (user-defined)



What do we do here?

The first step is to split a mesh into small parts that we call clusters.

We use the METIS library for this.

The size of the clusters is user-defined, but 128 triangles usually give the best performance.

Builder stage



Step 1: Cluster creation

- We use the **METIS** library to split the mesh into clusters
- Each cluster contains **128 triangles** (user-defined)




Here are the different clusters that we get for this mesh.

Builder stage



Step 2: Decimation

- Merge group of **contiguous** clusters (typically, 32 clusters)
- Remove edges to **divide** the number of **triangles** by 2  Edges at the border are **preserved** to avoid holes between LODs
- Create **16 clusters** to form the next level



The next step is what we call the decimation.

For this, we merge a group of contiguous clusters. A typical value is 32 clusters.

Then we remove edges to divide the number of triangles by exactly 2.

What is very important is to preserve edges at the border to avoid holes between LODs.

Then we create 16 new clusters. These clusters will be part of the next LOD.

Here, you can see the new clusters of the object.

Builder stage



Step 2: Decimation

- Merge group of **contiguous** clusters (typically, 32 clusters)
- Remove edges to **divide** the number of **triangles** by **2**
- Create **16 clusters** to form the next level

One of the **biggest challenges!**

A lot of **tweaking** here to reach good quality

The difficult part here is the edge collapse phase. Basically, you have an error function that you use to select the edges to remove. We have spent months to tweak this error function to improve the quality of the generated LODs.

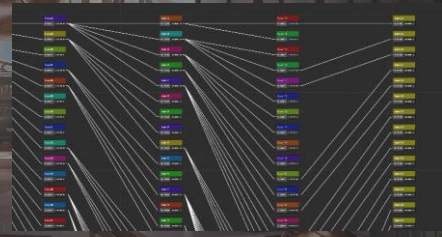
Builder stage

Step 3: Graph building

- The 16 clusters of the next level are the **parents** of the 32 previous clusters
- Clusters form a **Direct Acyclic Graph**

Larger triangles

Smaller triangles



Step 3 is the creation of the graph.

The 16 clusters of the next level are the parents of the 32 previous clusters.

The clusters form a direct acyclic graph.

Direct means that edges are one-way edges.

Acyclic means that there is no cycle.

It's not a tree because a node can have several parents.

On the top, we have clusters with larger triangles.

On the bottom, we have clusters with smaller triangles.

Here, the graph is very simple.

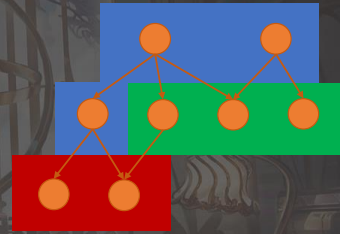
In practice, we have graphs like this, which are more difficult to debug.

Builder stage



Step 4: Chunk creation

- Group clusters to create **chunks**
- All chunks have the **same size on disk**
- 1 chunk = 1 block on disk to be streamed



The last step is the creation of the chunks.

A chunk is simply a group of clusters.

Here, we have a first chunk, then a second one, and the last one.

Why are we doing this?

Remember, a cluster is very small because it only contains 128 triangles.

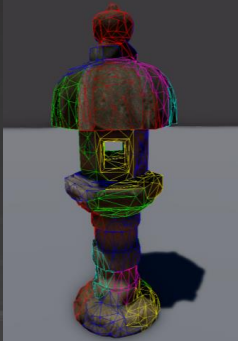
To be efficient, we must stream larger blocks of data.

All chunks have the same size on disk, and a chunk is the minimum amount of data that we can stream from the disk.

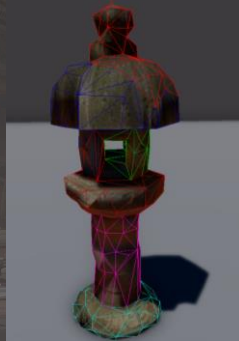
Builder stage



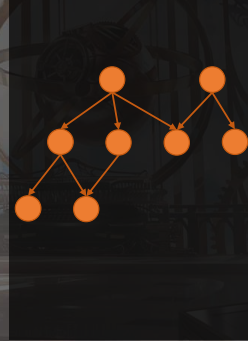
Step 1:
Cluster
creation



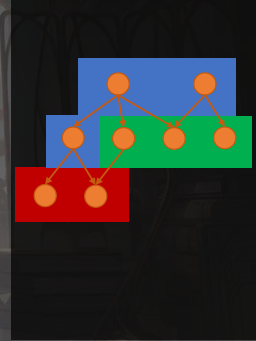
Step 2:
Decimation



Step 3:
Graph building



Step 4:
Chunk
creation



VIRTUOS CONFIDENTIAL

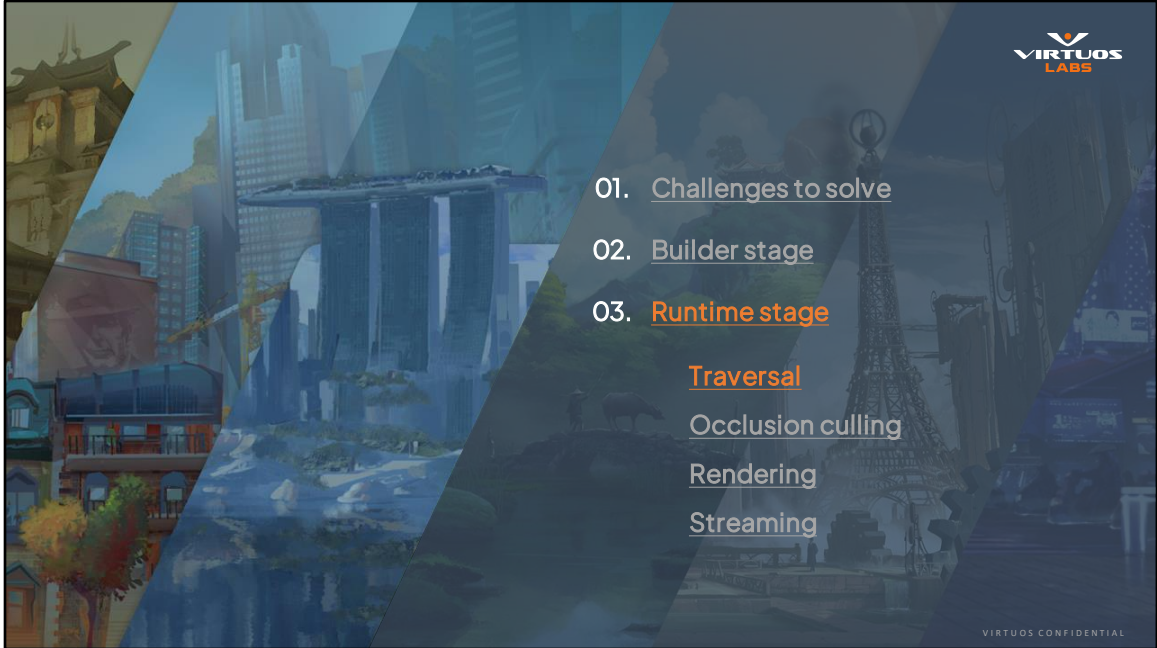
Let me summarize the different parts of the builder phase:

First, we split the mesh into small clusters.

Then we decimate to reduce the number of clusters to create the different LODs until we have a single cluster left.

We build a graph with the different clusters.

And we group clusters together to create chunks that will be streamed from disk.



Let's switch to the runtime stage.
The first step is the traversal.

Runtime stage – Traversal



- A compute shader which traverses the graphs to determine for each object:

- The clusters to render

Output: List of pairs (clusterID, objectID)

- The clusters to stream

Output: List of clusterID

- Clusters are selected based the projected bounding volume size on the screen

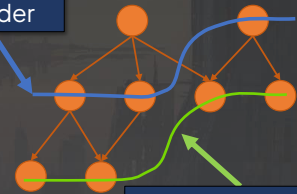
Clusters to render

Clusters to stream

Will be read by the next compute shader

Will be sent to RAM

Goal: each triangle has the same size on screen



The traversal is a compute shader which traverses all graphs to compute for each object the list of clusters to display.

We start from the graph's root, and we go down until we reach a cluster whose size on the screen is small enough.

The output is a list of pairs of cluster ID and object ID. We need the object ID to get the transform to apply, as well as the other per-object properties.

This list will be used by the next compute shader.

We also compute the list of clusters that need to be streamed from disk.

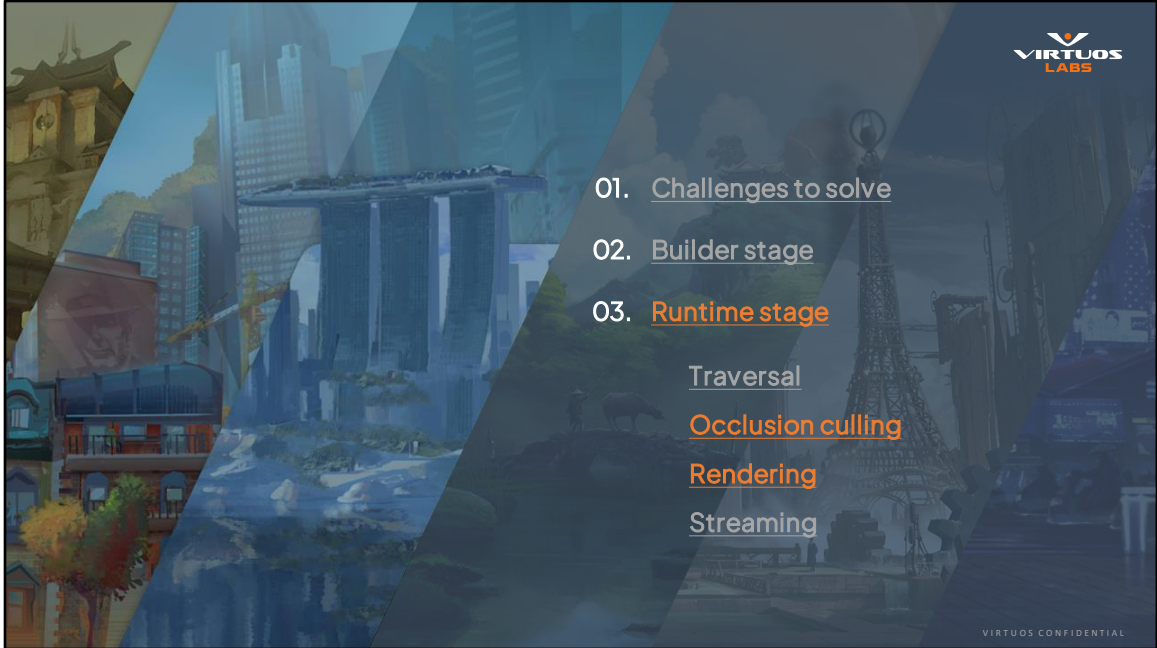
This time, we only need the cluster IDs.

This list will be sent to RAM and used to generate the streaming requests.

We select the clusters based of the size of their bounding spheres projected onto the screen.

The goal is to have triangles with approximately the same size on screen.

VIRTUOS CONFIDENTIAL



Next step is the occlusion culling.

Runtime stage – Per-cluster occlusion culling



- 1 compute shader for per-cluster frustum culling

Uses the results of the traversal phase

- 1 compute shader for per-cluster occlusion culling

Must be done after the Z-Prepass

Tests the bounding sphere of each cluster against the hierarchical Z Buffer

Output: List of DrawIndirect commands with (cluster ID, object ID)

VIRTUOS CONFIDENTIAL

We have a compute shader which takes the input of the traversal phase, does the frustum culling for the different clusters, and generates a new list of clusters. Then another computer shader which does the same for the occlusion culling. It uses the result of the Z-Prepass, that's why it must be executed after the Z-Prepass. Basically, we test the bounding volume of each cluster against the hierarchical Z Buffer. The output is a list of DrawIndirect commands with pairs of cluster ID and object ID.

Runtime stage – The rendering



- We simply execute the list of DrawIndirect commands
- Pixel shaders are not modified

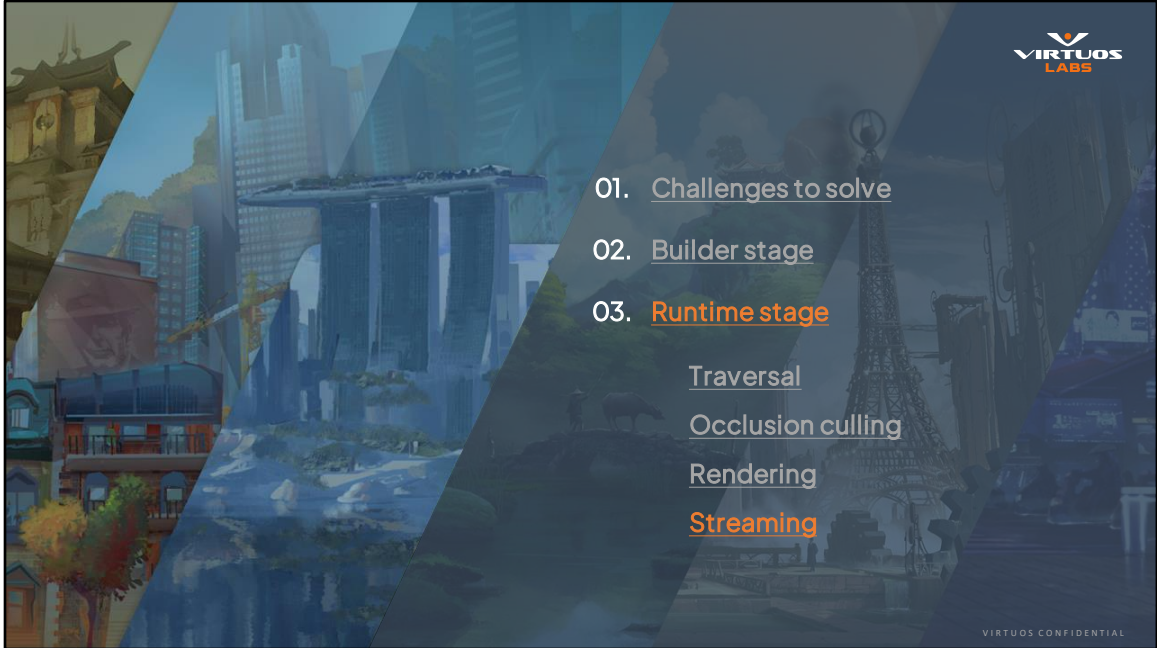
Very important to stay compatible with all materials

The rendering phase itself is pretty simply because we simply execute the DrawIndirect commands.

We don't modify the pixel shaders.

This is very important because we need to stay compatible with all materials, whether hardcoded or generated.

VIRTUOS CONFIDENTIAL



The last step is the streaming.

Runtime stage – Streaming

VIRTUOS
LABS

1 chunk = a list of clusters

All chunks have the same size on disk

- We work at the chunk level
- Fixed memory budget = fixed number of chunks
- For each chunk, store the time of last use
- Use LRU (*Least Recently Used*) algorithm to choose chunks to unload
- Chunks can have a priority level

Remember, for the streaming, we work at the chunk level.

A chunk is a list of clusters, and all chunks have the same size on disk.

We used a fixed memory budget, so we have a fixed number of chunks that can be in memory at the same time.

For each chunk, we store the time of last use, and we use a classical LRU – Least Recently Used – algorithm to choose a chunk to unload when we need to load a new chunk.

This part was not too difficult to implement, until we had to add a notion of priority. Then things became more complex.

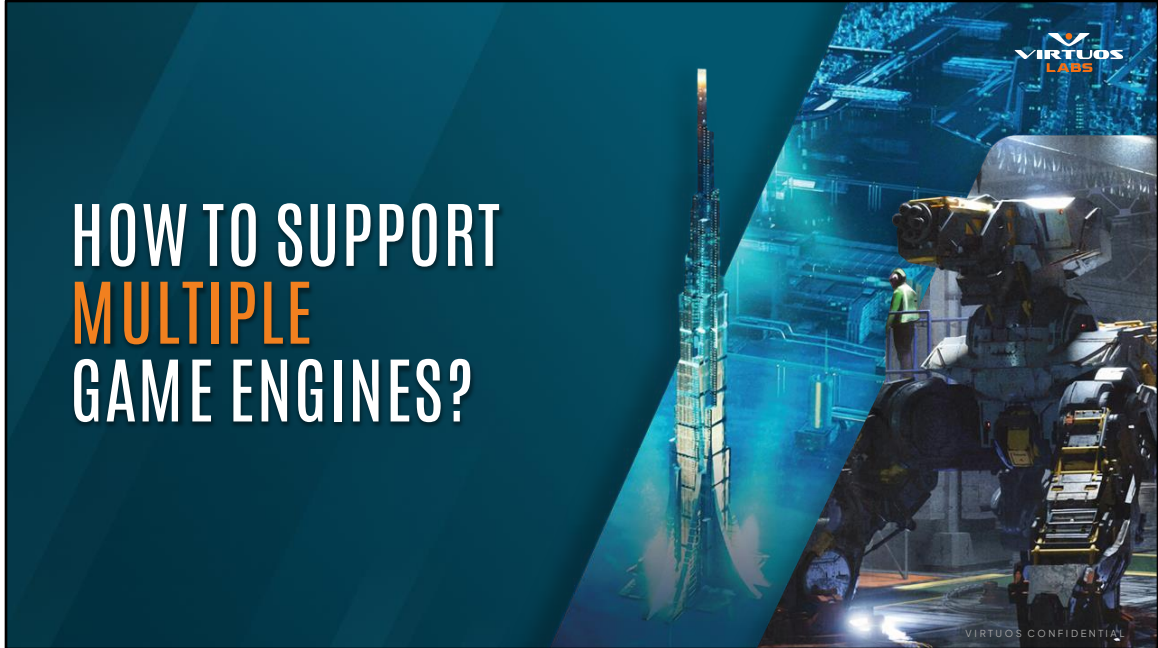


TABLE OF CONTENTS

- 01. Why use virtualized geometry?
- 02. Creating Hi-res Vision
- 03. **How to support multiple game engines?**
- 04. Going further: geomorphing and future work
- 05. Conclusion & takeaway

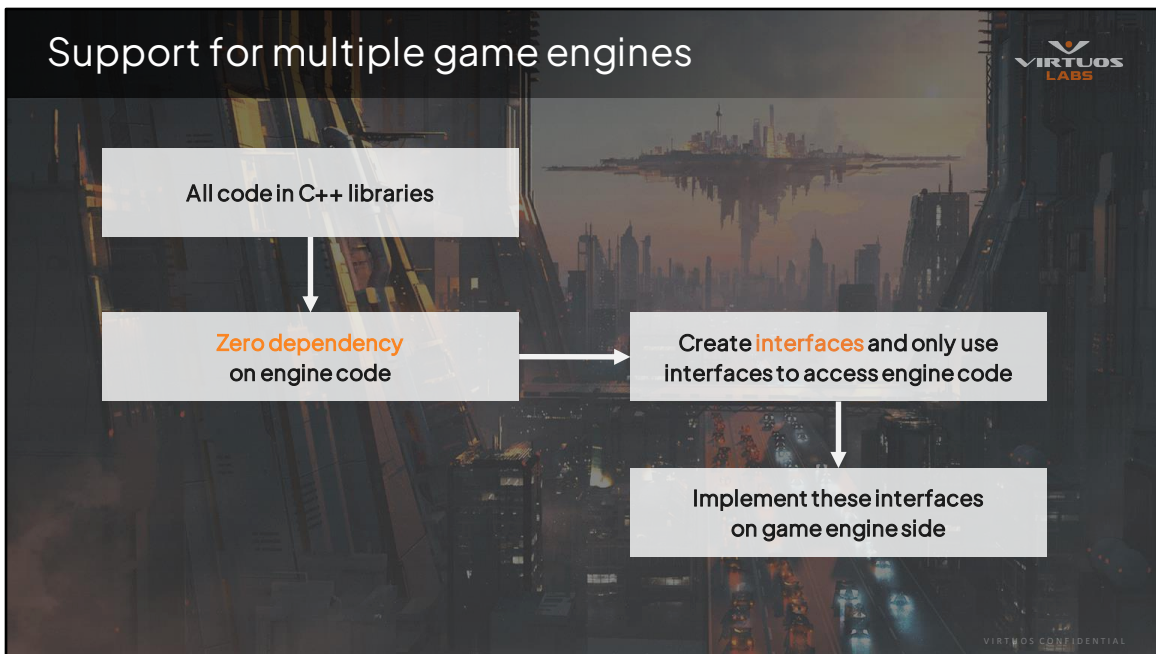
VIRTUOS
LABS

VIRTUOS CONFIDENTIAL



We're going to see some techniques that we used to support several game engines.

Support for multiple game engines

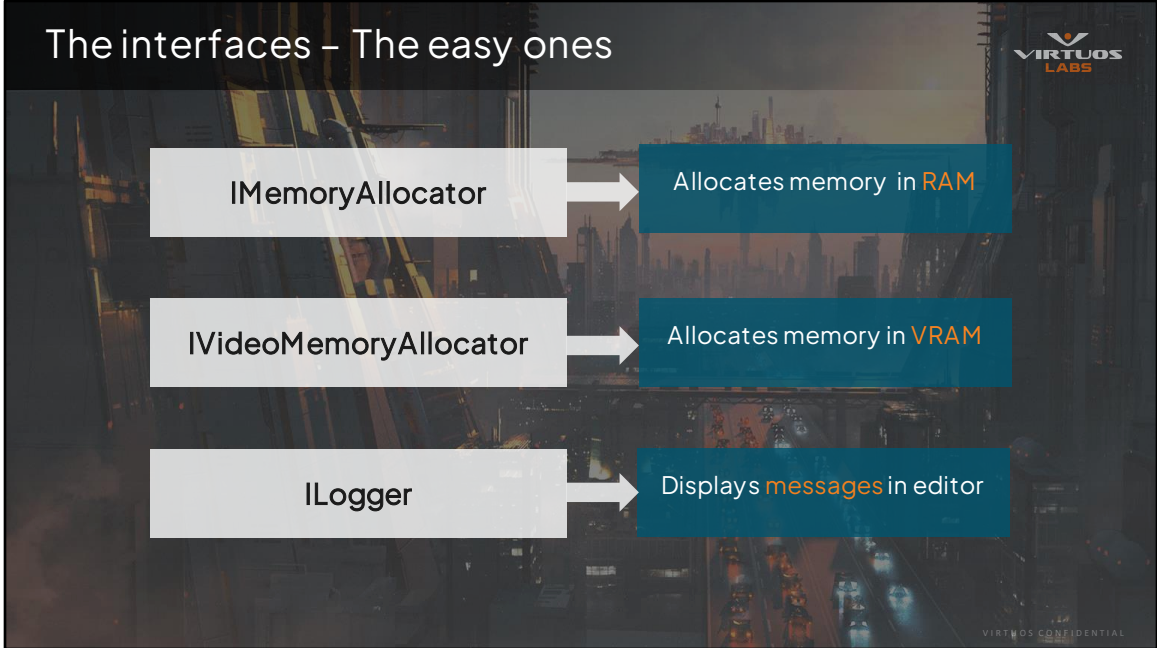


The main idea is that we put all code in C++ libraries, that have no access to the engine code.

We can access the engine code only through interfaces that we created, basically abstract classes.

And we implement these interfaces on the game engine side.

The interfaces – The easy ones



Let see some of these interfaces.

Some are very simple, such as a memory allocator to allocate memory in RAM, a video memory allocator, or a logger to display messages.

The interfaces - The streaming



IStreamingManager

- Send request to stream data
- Be called when data has been loaded

Use small files and always read entire files?
Use bigger files and read parts of files?

We need to
support both

We need a generic
way to identify a data

IIdProvider

- Provides ID to identify data
(file, part of file)

An ID is fully generic (array of bytes)

It can be a GUID, a file name, a full path or anything

For the streaming, we have a streaming manager to send a request to stream a block of data and register a callback to be warned when the data has been loaded.

Should we put one chunk per file? Or all chunks in a single file?

Well... It depends on the game engine.

We need to support both.

So, we need a generic way to identify a data to stream.

To do so, we created another interface to provide an ID to identify a data.

The ID must be generic. In fact, it's just an array of bytes.

It can be whatever we want, a path, a file name, a GUID, etc.

The interfaces – The rendering



IBinder

Binds a buffer to a shader

The rest of the rendering is engine-specific

For the rendering phase, we have a binder interface to bind a buffer to a shader.
And the rest of the rendering code is engine specific.

The shaders

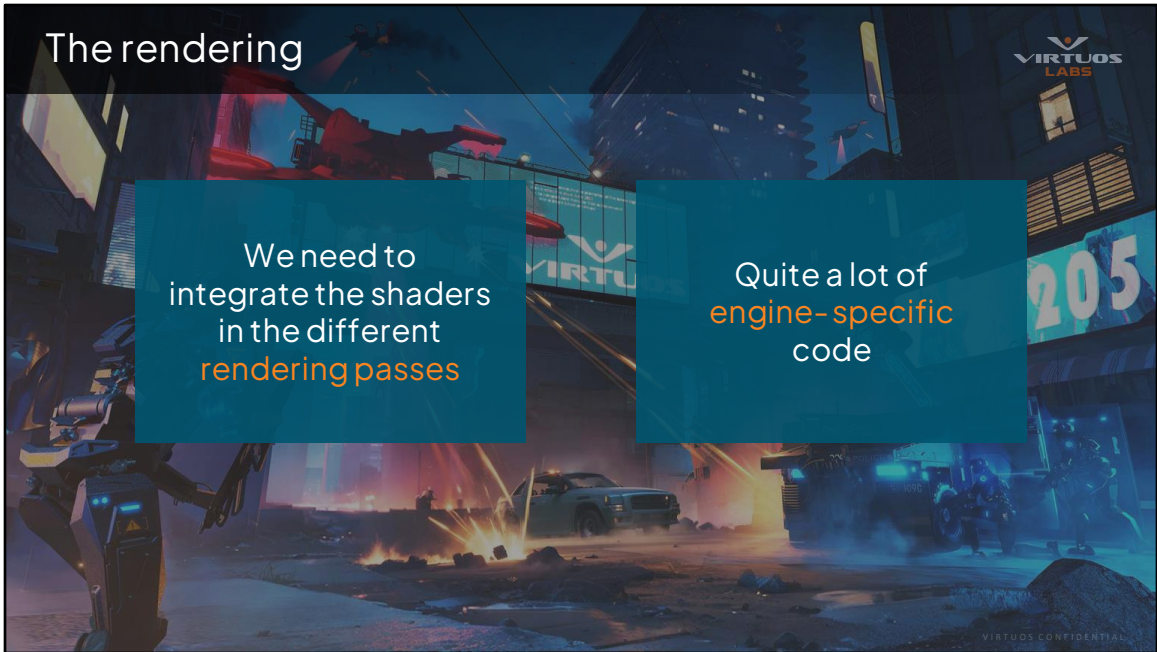
Game engines usually have their own shader language

Shaders are written in **HLSL** in the library but **rewritten** for each game engine

Usually a straightforward operation

As for the shaders, game engines usually have their own shader language on top of HLSL.

Our shader are written in HLSL in the library but rewritten for each game engine. This is usually a straightforward operation.



For the rendering itself, we need to integrate the shaders in the different rendering passes.

This requires quite a lot of engine-specific code.



TABLE OF CONTENTS

- 01. Why use virtualized geometry?
- 02. Creating Hi-res Vision
- 03. How to support multiple game engines?
- 04. **Going further: geomorphing and future work**
- 05. Conclusion & takeaway



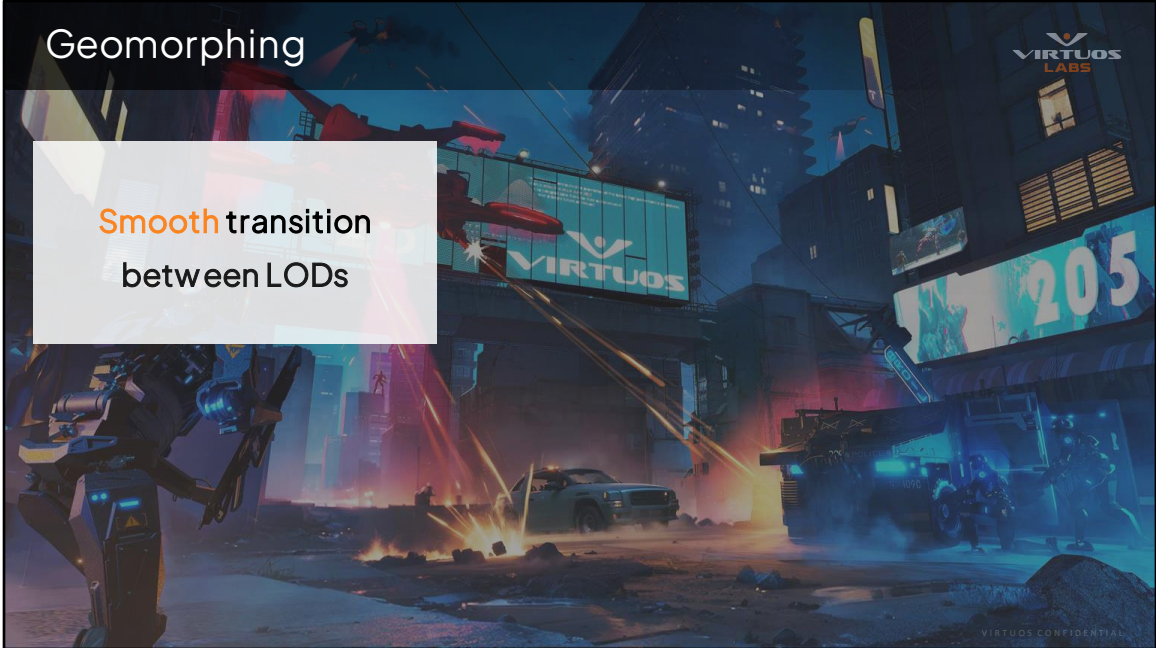
VIRTUOS CONFIDENTIAL



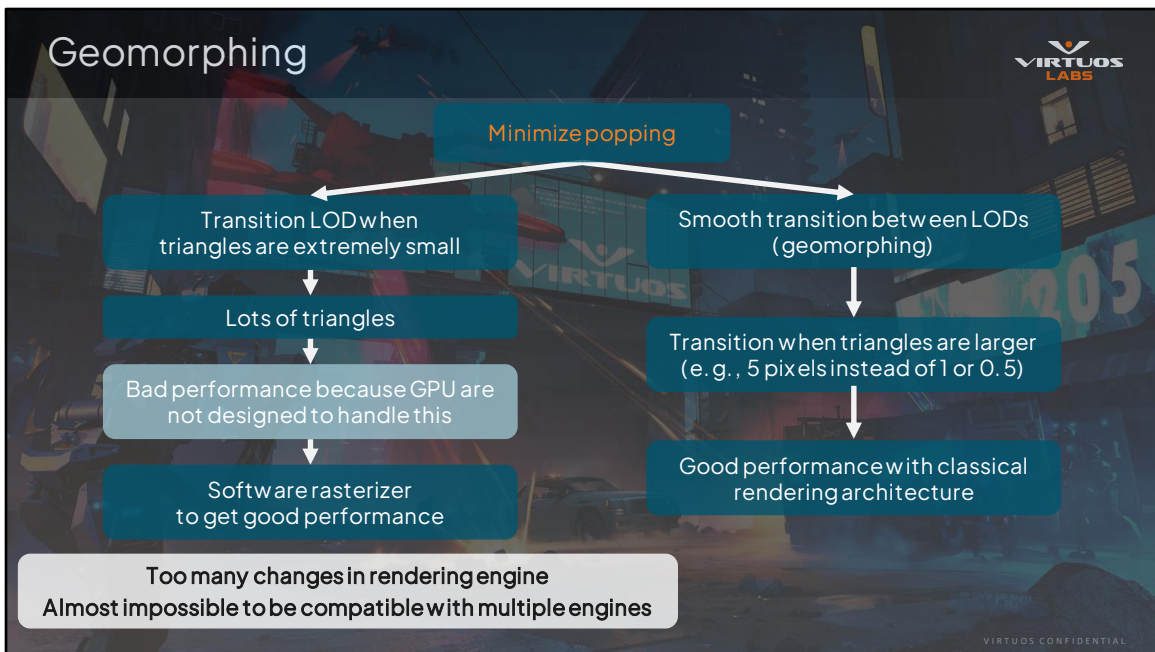
I'm going to finish this presentation with our current work in progress.

Geomorphing

Smooth transition
between LODs



We are currently working on geomorphing.
What is geomorphing?
It's simply a smooth transition between LODs.



What can geomorphing be useful for?

One challenge we have is to minimize popping when switching from LOD to another. One possible solution is to transition between LODs when triangles are very small, typically smaller than one pixel.

The drawback is that we have a lot of triangles, which is bad for performance.

A solution is to implement a software rasterizer.

We didn't go this way because it has a big impact on the rendering engine, and it would be very difficult to have an implementation that is compatible with multiple game engines.

Another possibility is to have smooth transition between LODs, which is exactly what geomorphing does, and switch from LOD to another when triangles are larger, for example 4 or 5 pixels instead of 1 or less than 1.

We did some tests: there is no quality increase when using 1-pixel triangles compared to 5-pixel triangles.

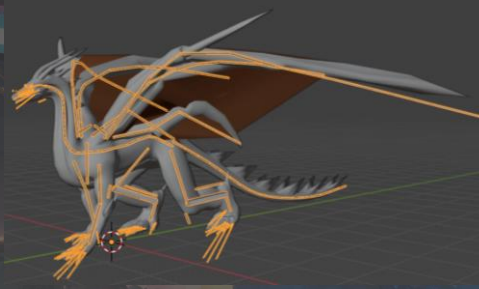
Future work



Foliage support



Animated mesh support



Our R&D team is also working on foliage support, which isn't too difficult, and animated mesh support, which is a bigger challenge.

VIRTUOS CONFIDENTIAL



TABLE OF CONTENTS

- 01. Why use virtualized geometry?
- 02. Creating Hi-res Vision
- 03. How to support multiple game engines?
- 04. Going further: geomorphing and future work
- 05. **Conclusion & takeaway**



VIRTUOS CONFIDENTIAL



What are the most relevant topics to remember?

Conclusion & takeaway



Virtualized geometry is becoming standard

Builder stage

Cluster
creation

Decimation

Graph
building

Chunk
creation

- METIS is your friend, don't reinvent the wheel
- Quite complex algorithms to implement
- The quality of the decimation is key
- Expect a *long* tweaking phase to get good results
- Mesh build times are important

I do believe that virtualized geometry is becoming standard in video games. I'm recalling here the different steps of the builder stage: cluster creation, decimation, graph building, and chunk creation. METIS is a very powerful library, it's open source, use it when you need it. The algorithms of Builder stage are not so easy to implement, it can take a long time to achieve a good quality for LODs, with a lot of specific cases to handle. And a point I didn't mention yet: the time to build meshes can be significant.

Conclusion & takeaway (cont'd)



Rendering

- Naïve implementation quite easy to write
- Optimized version took a much longer time

Streaming

- Classical LRU algorithm easy to write
- Things become more complex when dealing with priorities

Geomorphing, foliage, animated mesh

- We will disclose our results when available

VIRTUOS CONFIDENTIAL

For the rendering stage, a simple implementation is not so difficult to write, the challenge is more to get an optimized version.

As for the streaming part, a LRU algorithm gives good results, but managing priorities is more complex.

About our current work, we will disclose our results when they are available.

Stay tune!

References



- [1] R. Li, « A lightweight 3D viewer: real-time rendering of multi-scale 3D surface models ».
- [2] A. Beacco, N. Pelechano, et C. Andújar, « A Survey of Real-Time Crowd Rendering », Computer Graphics Forum, vol. 35, n° 8, p. 32-50, déc. 2016, doi: 10.1111/cgf.12774.
- [3] A. E. W. Mason et E. H. Blake, « Automatic Hierarchical Level of Detail Optimization in Computer Animation », 1997.
- [4] E. Danovaro, L. De Floriani, E. Puppo, et H. Samet, « Clustering Techniques for Out-of-Core Multi-resolution Modeling », in IEEE Visualization 2005 - (VIS'05), Minneapolis, MN, USA: IEEE, 2005, p. 113-113. doi: 10.1109/VIS.2005.15.
- [5] V. Semenov, V. Shutkin, et V. Zolotov, « Conservative Out-of-Core Rendering of Large Dynamic Scenes Using HDLODs », in Proceedings of the 31th International Conference on Computer Graphics and Vision. Volume 2, Keldysh Institute of Applied Mathematics, 2021, p. 105-115. doi: 10.20948/graphicon-2021-3027-105-115.
- [6] Y. Zhang et X. Chen, « Constructing and Rendering of Multiresolution Representation for Massive Meshes with GPU and Mesh Layout », JSW, vol. 8, n° 8, p. 1968-1975, août 2013, doi: 10.4304/jsw.8.8.1968-1975.
- [7] H. Hoppe et S. Marschner, « Efficient Minimization of New Quadric Metric for Simplifying Meshes with Appearance Attributes ».
- [8] H. Legrand, J. Thiery, et T. Boubekeur, « Filtered Quadrics for High-Speed Geometry Smoothing and Clustering », Computer Graphics Forum, vol. 38, n° 1, p. 663-677, févr. 2019, doi: 10.1111/cgf.13597.
- [9] C. Zach, « Integration of geomorphing into level of detail management for realtime rendering », in Proceedings of the 18th Spring Conference on Computer Graphics, Budmerice Slovakia: ACM, avr. 2002, p. 115-122. doi: 10.1145/584458.584478.
- [10] G. Debunne, M. Desbrun, A. Barr, et M.-P. Cani, « Interactive multiresolution animation of deformable models », in Computer Animation and Simulation '99, N. Magnenat-Thalmann et D. Thalmann, Éd., in Eurographics. , Vienna: Springer Vienna, 1999, p. 133-144. doi: 10.1007/978-3-7091-6423-5_13.

VIRTUOS CONFIDENTIAL

I put here a list of papers that we read or we used to build our technology.

References (cont'd)



- [11] E. Danovaro, L. De Floriani, P. Magillo, E. Puppo, et D. Sobrero, « Level-of-detail for data analysis and exploration: A historical overview and some new perspectives », *Computers & Graphics*, vol. 30, n° 3, p. 334-344, juin 2006, doi: 10.1016/j.cag.2006.02.006.
- [12] A. Dietrich, E. Gobbetti, et S.-E. Yoon, « Massive-Model Rendering Techniques », *IEEE Computer Graphics and Applications*, 2007.
- [13] M. Wand et W. Straßer, « Multi-Resolution Rendering of Complex Animated Scenes ».
- [14] S. Zhang et E. Wu, « Multiresolution Animated Models Generation Based on Deformation Distance Analysis », in 2009 International Conference on Computer Modeling and Simulation, Macau, China: IEEE, févr. 2009, p. 73-77. doi: 10.1109/ICCMS.2009.43.
- [15] M. Garland, « Multiresolution Modeling: Survey & Future Opportunities ».
- [16] H. Hoppe, « New quadric metric for simplifying meshes with appearance attributes », in *Proceedings Visualization '99 (Cat. No.99CB37067)*, San Francisco, CA, USA: IEEE, 1999, p. 59-510. doi: 10.1109/VISUAL.1999.809869.
- [17] Y. Zhang et D. Xu, « Parallel construction and rendering of multi-resolution representation for massive meshes with GPU », *IFS*, vol. 33, n° 5, p. 3165-3172, oct. 2017, doi: 10.3233/JIFS-169368.
- [18] H. Hoppe, « Progressive meshes ».
- [19] J. Popović et H. Hoppe, « Progressive simplicial complexes », in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, Not Known: ACM Press, 1997, p. 217-224. doi: 10.1145/258734.258852.
- [20] A. Ghazanfarpour, N. Mellado, C. E. Himeur, L. Barthe, et J.-P. Jessel, « Proximity-aware multiple meshes decimation using quadric error metric », *Graphical Models*, vol. 109, p. 101062, mai 2020, doi: 10.1016/j.gmod.2020.101062.
- [21] M. Garland et Y. Zhou, « Quadric-based simplification in any dimension », *ACM Trans. Graph.*, vol. 24, n° 2, p. 209-239, avr. 2005, doi: 10.1145/1061347.1061350.

VIRTUOS CONFIDENTIAL

References (cont'd)



- [22] D. L. James, Twigg Christopher D., « Skinning Mesh Animations ». 2005.
- [23] H. Hoppe, « Smooth view-dependent level-of-detail control and its application to terrain rendering », in Proceedings Visualization '98 (Cat. No.98CB36276), Research Triangle Park, NC, USA: IEEE, 1998, p. 35-42., doi: 10.1109/VISUAL.1998.745282.
- [24] E. Gobbetti et E. Bouvier, « Time-critical multiresolution rendering of large complex models », Computer-Aided Design, vol. 32, n° 13, p. 785-803, nov. 2000, doi: 10.1016/S0010-4485(00)00068-3.
- [25] Xinyue Li et Han-Wei Shen, « Time-critical multiresolution volume rendering using 3D texture mapping hardware », in Symposium on Volume Visualization and Graphics, 2002. Proceedings. IEEE / ACM SIGGRAPH, Boston, MA, USA: IEEE, 2002, p. 29-36. doi: 10.1109/SVG.2002.1226507.
- [26] Yuanchen Zhu, « Uniform Remeshing with an Adaptive Domain: A New Scheme for View-Dependent Level-of-Detail Rendering of Meshes », IEEE Trans. Visual. Comput. Graphics, vol. 11, n° 3, p. 306-316, mai 2005, doi: 10.1109/TVCG.2005.50.
- [27] R. Li, « View-dependent Adaptive HLOD: real-time interactive rendering of multi-resolution models », 2023.
- [28] H. Chen, S. Zhan, Y. Gao, et W. Zhang, « View-Dependent Out-of-Core Rendering of Large-Scale Virtual Environments with Continuous Hierarchical Levels of Detail », in 2008 International Conference on Computer Science and Information Technology, IEEE, août 2008, p. 313-321. doi: 10.1109/ICCSIT.2008.192.
- [29] Junho Kim, Seungyong Lee, et L. Kobbelt, « View-dependent streaming of progressive meshes », in Proceedings Shape Modeling Applications, 2004., Genova, Italy: IEEE, 2004, p. 209-391. doi: 10.1109/SMI.2004.1314508.

VIRTUOS CONFIDENTIAL



Maximizing Graphics Performance
with Flexible Virtualized Geometry

THANK YOU

Alexis Vaisse

Senior Technical Director
avaisse@virtuosgames.com

Marios Michaelides

Engineering BU Director
mmichaelides@virtuosgames.com

VIRTUOS CONFIDENTIAL