High-Performance **GPU-Driven** Rendering and Hierarchical Culling Architecture

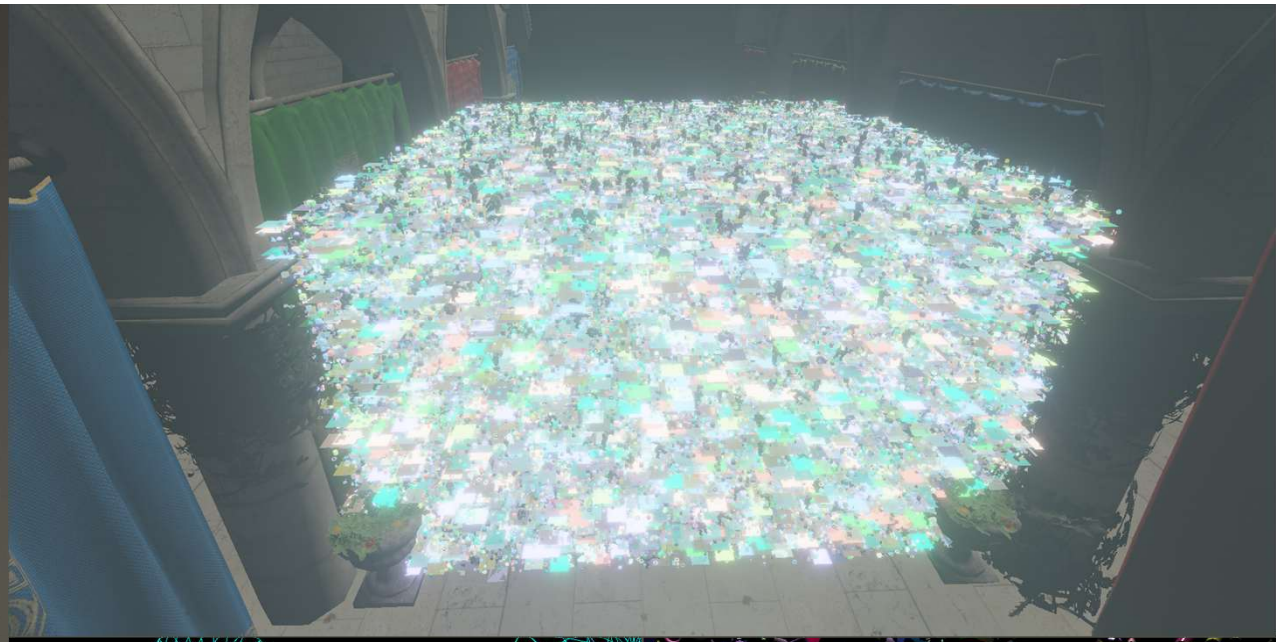
Tamás Péter, Fridvalszky András

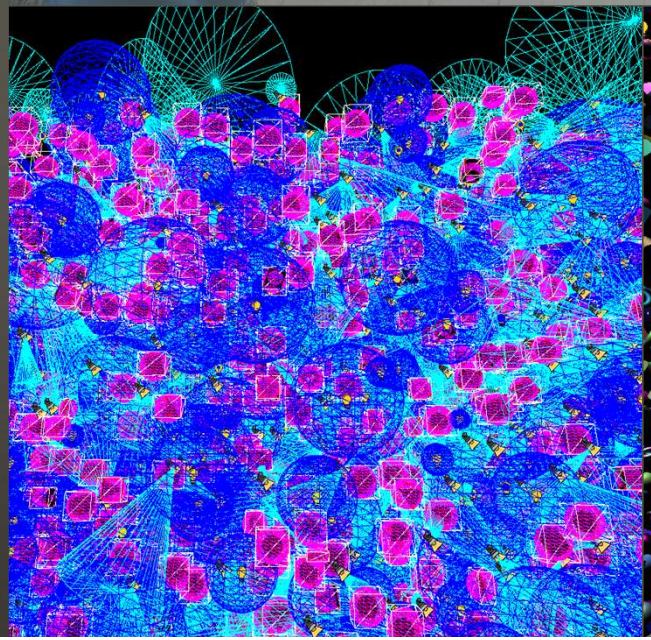
2026, XII. Magyar Számítógépes Grafika és Geometria Konferencia

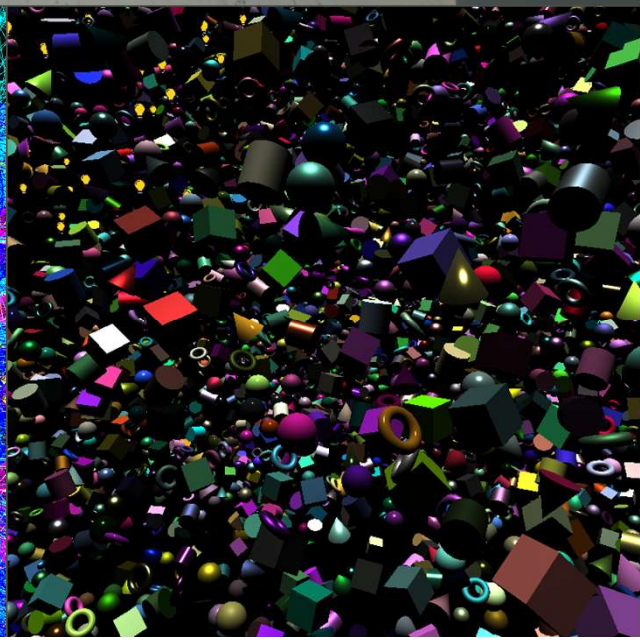
Miről fog szólni az előadás?

- **Hogyan lehet több millió objektumot nagyon hatékonyan kezelni, és megjeleníteni minimális CPU-GPU overheaddel, maximálisan kihasználva a GPU párhuzamosítási lehetőségeit?**

- **Az előadás felépítése**
  1. CPU oldali objektum adattárolás
  2. Bindless GPU adat ábrázolás
  3. GPU-Driven Rendering
  4. GPU-Driven Hierarchikus Culling
  5. Különböző Culling Technikák

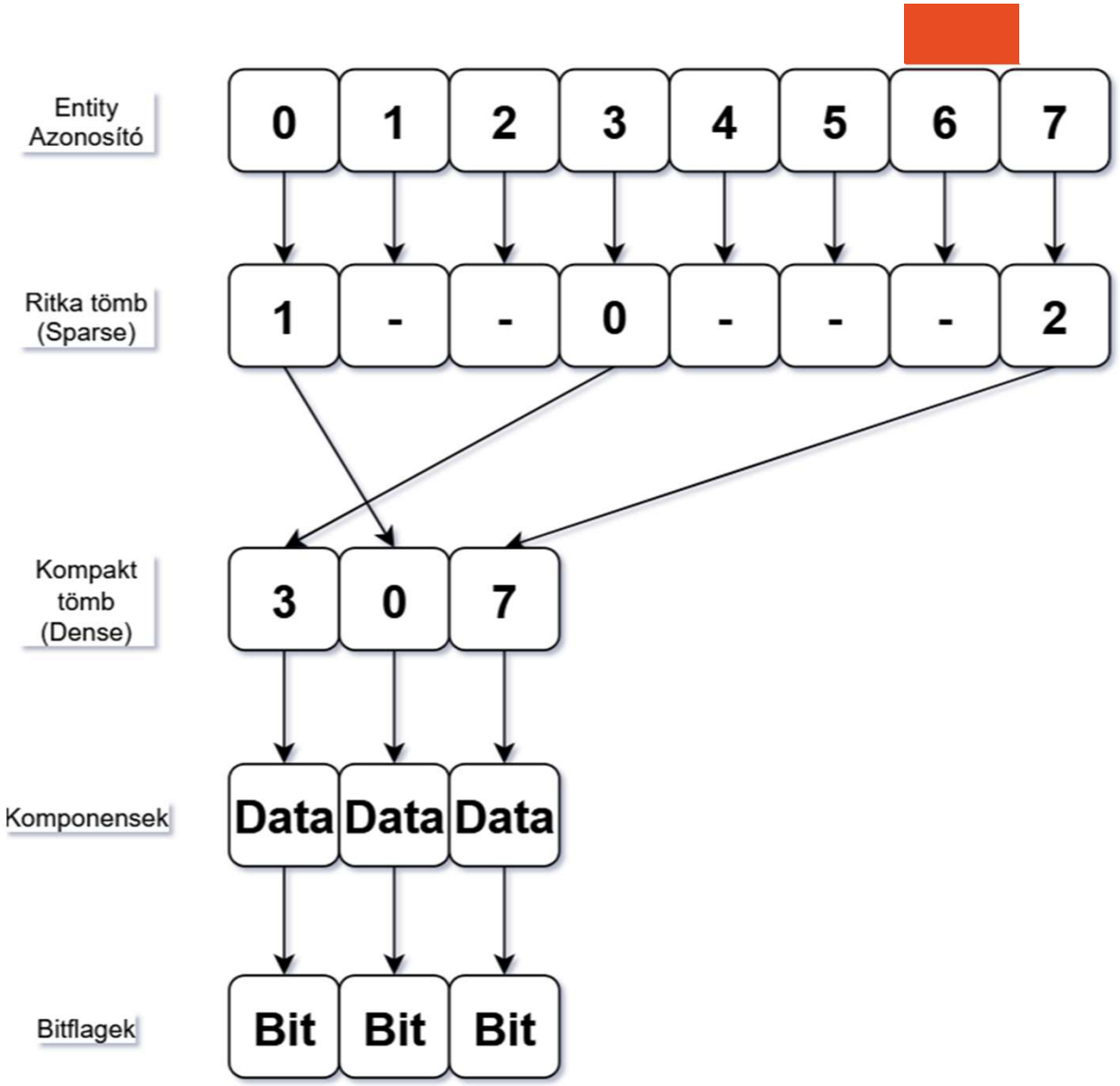






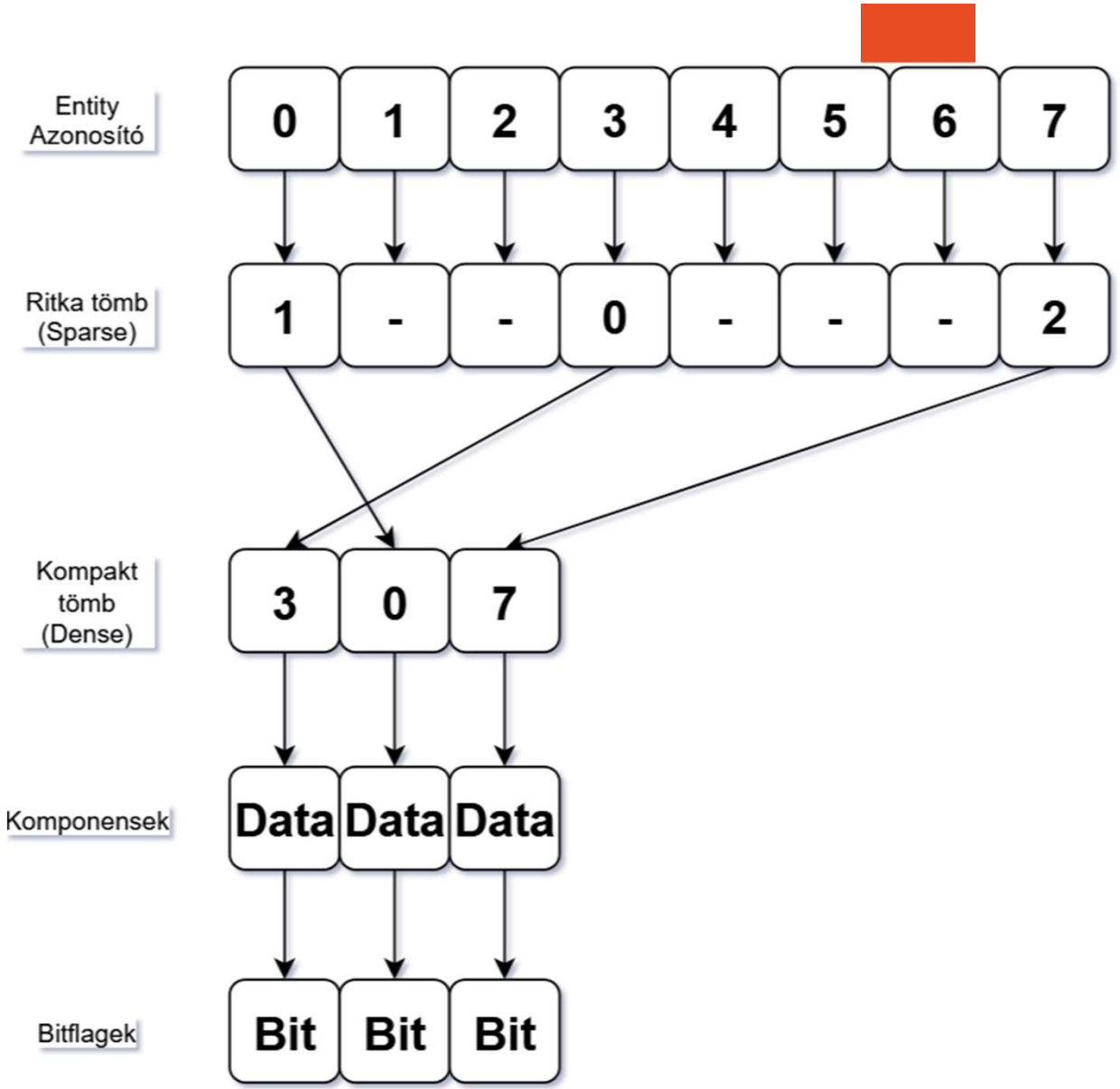
Adatorientált Entity-Component-System I.

- **Entitások**: unsigned int azonosítók *(UINT32_MAX = invalid entity)*
- **Komponensek**: tisztán adatot tartalmazó struktúrák
- **Systemek**: adott típusú komponenseket frissítenek, rajtuk műveleteket végeznek
- **Adatorientáltság előnyei**:
  - *Maximum **cache lokalitás***
  - ***Valós konstans O(1)** indexelés*
  - *Systemek **párhuzamos futtatása***
  - ***SIMD** párhuzamos komponens frissítés*
  - ***Kompakt adat tárolás**: GPU számára tökéletes ábrázolás*



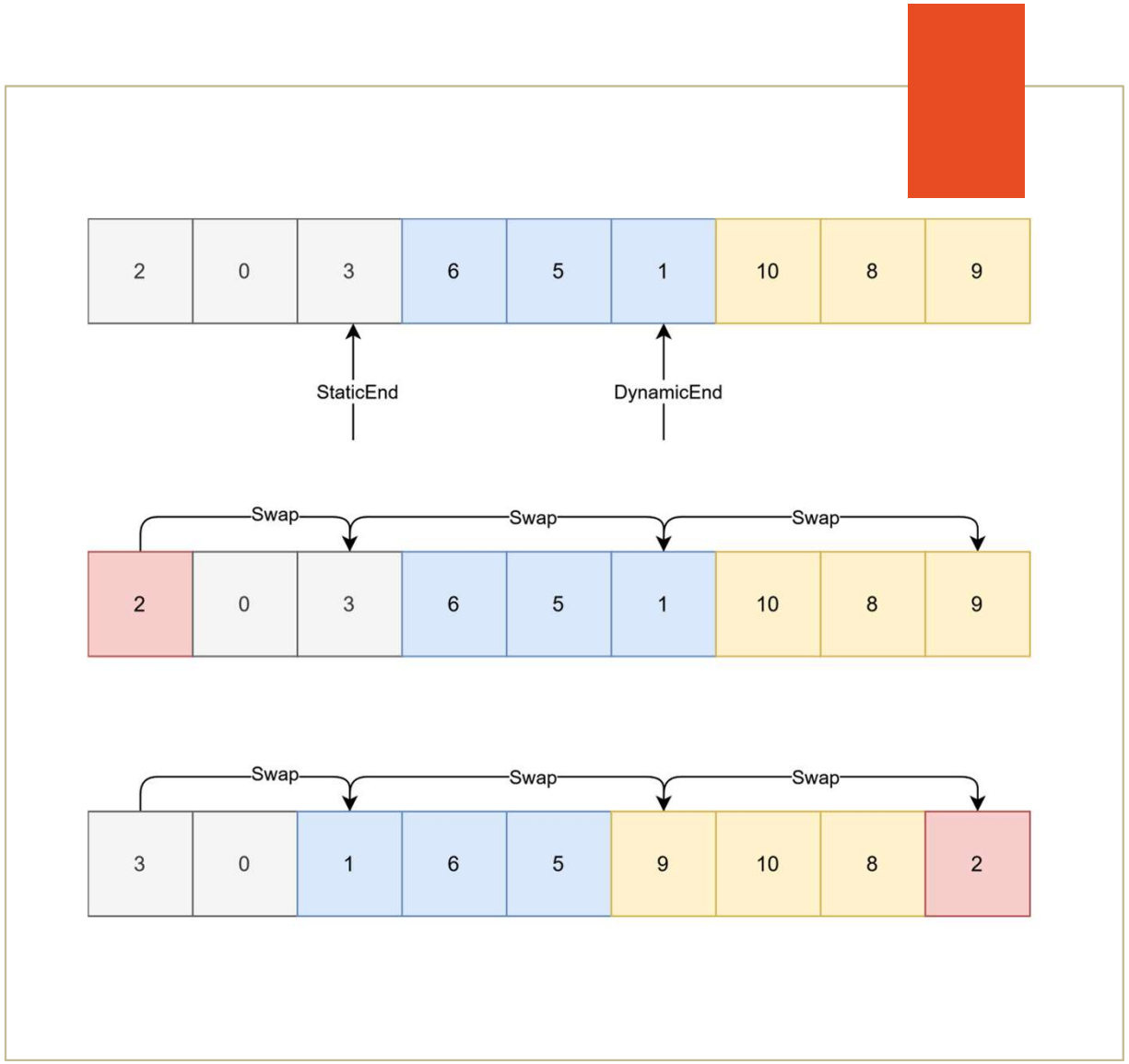
Adatorientált Entity-Component-System II.

- **Probléma**: Minden egyes frameben az **teljes kompakt listán végigmegyünk** és frissítjük, akkor is ha egyáltalán **nem változott** az adat.
- **Megoldás**: Állapotbit bevezetése
  - **UpdateBit**: Jelzi, hogy megváltozott egy változó, ezért frissíteni kell a komponenst.
  - **ChangedBit**: Jelzi, hogy frissült. Hierarchikus egymásrahatások kezelhetőek más systemekkel.
- **Optimalizálás**: Teljes pool szintű állapot tracking.



Adatorientált Entity-Component-System III.

- **Probléma**: A teljes kompakt listán végig kell még mindig iterálni.
  (1 millió entitás esetén 1 millió iteráció)
- **Megoldás**: Szegmentált adattárolás frissítési gyakoriság szerint
  - **Static**: Nagyon ritkán változik, szinte soha.
    (Dirty List + Atomic Counter)
  - **Dynamic**: Mérsékelten sokszor változik.
    (Állapotbit + Teljes régió állapotbit)
  - **Stream**: Minden frameben változik. (fizikai szimuláció például).
- **Nagyon hatékony törlés és hozzáadás!! Maximum 3 csere.**



Adatorientált Entity-Component-System IV.

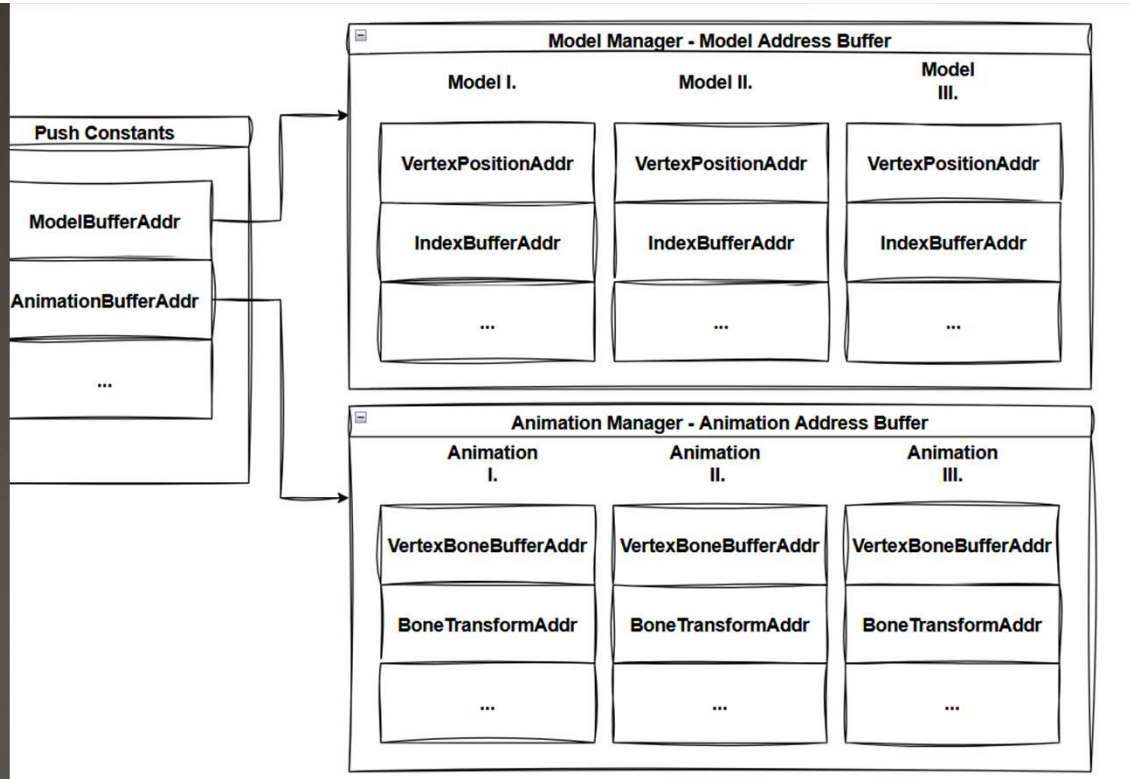
- **Implementációs megoldások**
  - **Teljesen template alapú**, egyáltalán nincsen virtuális függvény!!
  - Teljesen **Mixin alapú** moduláris komponens storage implementáció.
  - Típus bíztos megvalósítás template **CRTP pattern** megoldással.
  - Magas modularitás **Template Extension Pattern** megvalósításával

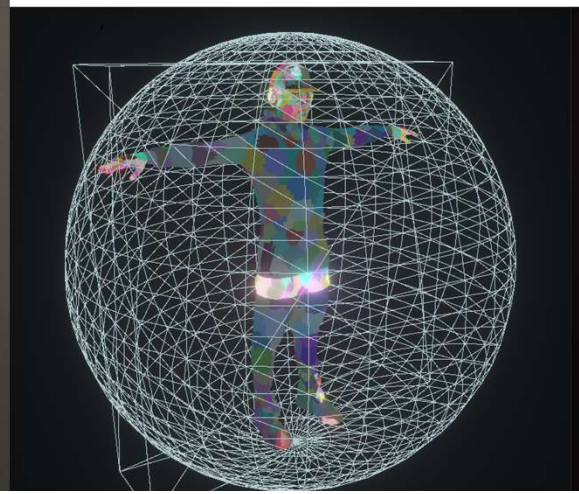
Table 1: *Performance comparison between the Custom Segmented ECS (Synapse) and EnTT. CPU time in ms (lower is better).*

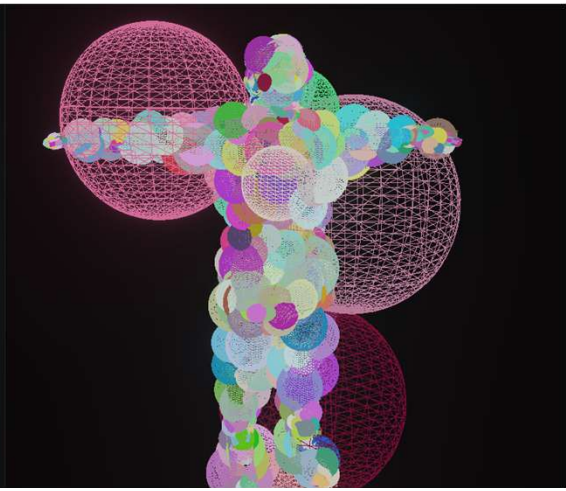
Operation	Entities	Synapse (ms)	EnTT (ms)
Add	100k	6.08	2.99
	1M	59.66	35.59
Remove	100k	2.31	3.29
	1M	26.18	37.01
Iterate (Seq)	100k	0.24	0.20
	1M	2.46	2.25
Iterate (Par)	100k	0.25	0.25
	1M	2.15	2.26
Engine Update	100k	0.11	0.14
	1M	1.15	1.76

Betöltött Modelek kezelése – Bindless Architektúra

- **Pipeline**: Adatorientált Raw/Cooked/Gpu 3 szintű pipeline feldolgozás (Mesh szintű párhuzamosság)
- **Legenerált adatok**:
  - Vertex adatok külön szedve
    1. Vertex Positions
    2. Vertex Attributes
  - Index Adatok (4 LOD)
  - Meshlet Adatok (4 LOD)
  - Colliderek (Sphere + AABB)
  - Leíró indexelő struktúrák…
- **MINDEN ADATORIENTÁLTAN BATCHELVE A GPU BUFFERBEN, DE MESH SZINTŰ AZONOSÍTHATÓSÁGGAL**

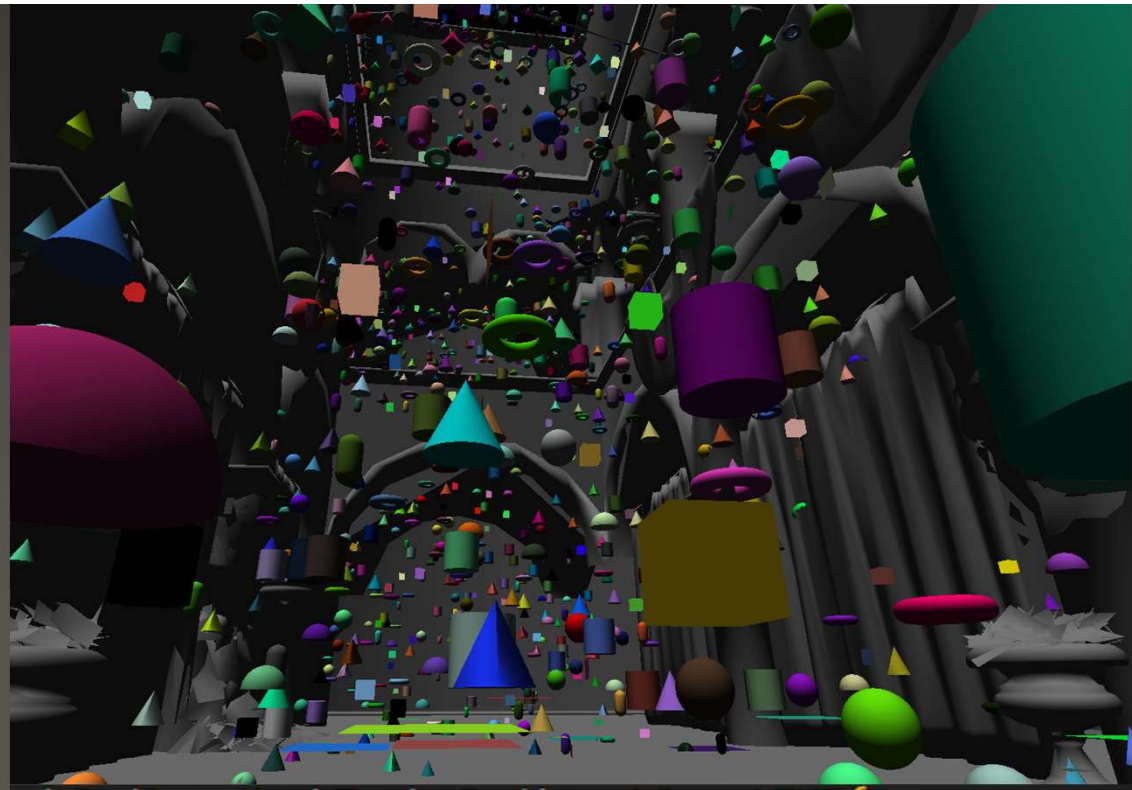






GPU-Driven Rendering

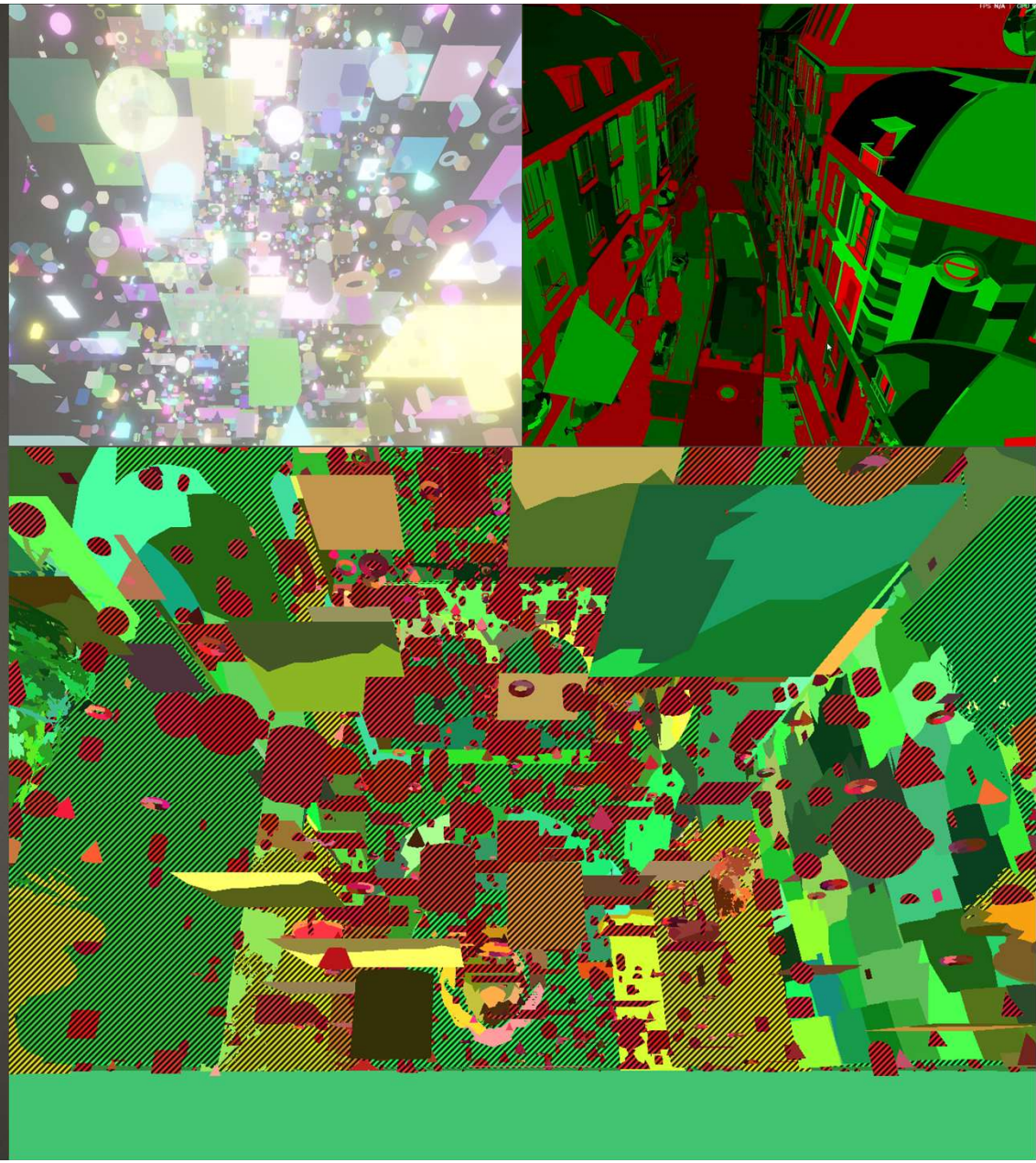
- **Indirect Draw**: A CPU csak kiadja a rajzoló utasítást, de semmit nem tud a részletekről. Minden **draw command GPU bufferben** van tárolva.
- **Indirect Draw Command**: Leíró ami a draw paramétereket tárolja (VertexCount, InstanceIndex, stb…)
- Minden mesh-hez tartozik saját draw command! A **renderer meshekben gondolkodik**, míg a ECS modelekben!
- Minden **meshez fixen 4 draw command** a megfelelő LOD szintekhez! Batchelve egymás után vannak tárolva.
  4 * meshIndex + lodIndex



```
vkCmdDrawMeshTasksIndirectCountEXT(
    context.cmd,
    indirectBuffer,
    indirectOffset,
    countBuffer,
    countOffset,
    maxCommandCount,
    sizeof(VkDrawMeshTasksIndirectCommandEXT)
);
```

8-Bucket Draw Command felosztás

- Összes draw command 1 nagy bufferben batchelve.
- **2-Bucket Rendszer**: Traditional és Mesh shader felosztás.
- **8-Bucket Rendszer**: Tovább osztás material típus szerint.
  1. Opaque 1 sided
  2. Opaque 2 sided
  3. Transparent 1 sided
  4. Transparent 2 sided
- Transparencia: **Weighted Blending Order Independent** technikával

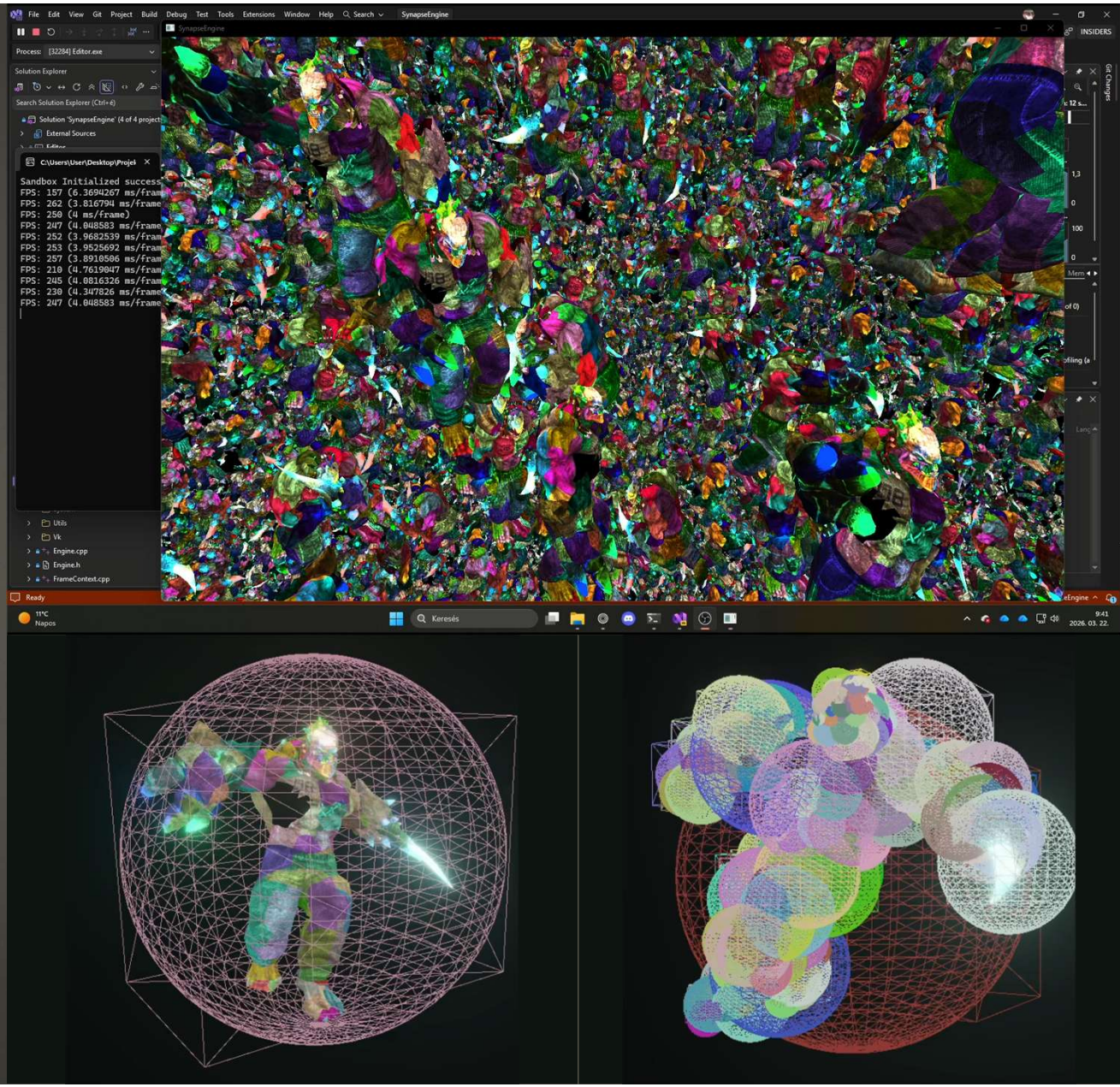


Level-Of-Detail (LOD)

- **Mesh LOD**: Meshoptimizerrel automatikusan generálva. (Csak index adatok redukálása)
- **Meshlet LOD**: Mesh LOD-ból generált meshletek!
- **Dinamikus LOD számítás**: projektált screen-space méret alapján
  - >512px : 0. szint (100% részletesség)
  - >256px : 1. szint (50%)
  - >128px : 2.szint (25%)
  - > 64px : 3. szint (12.5%)

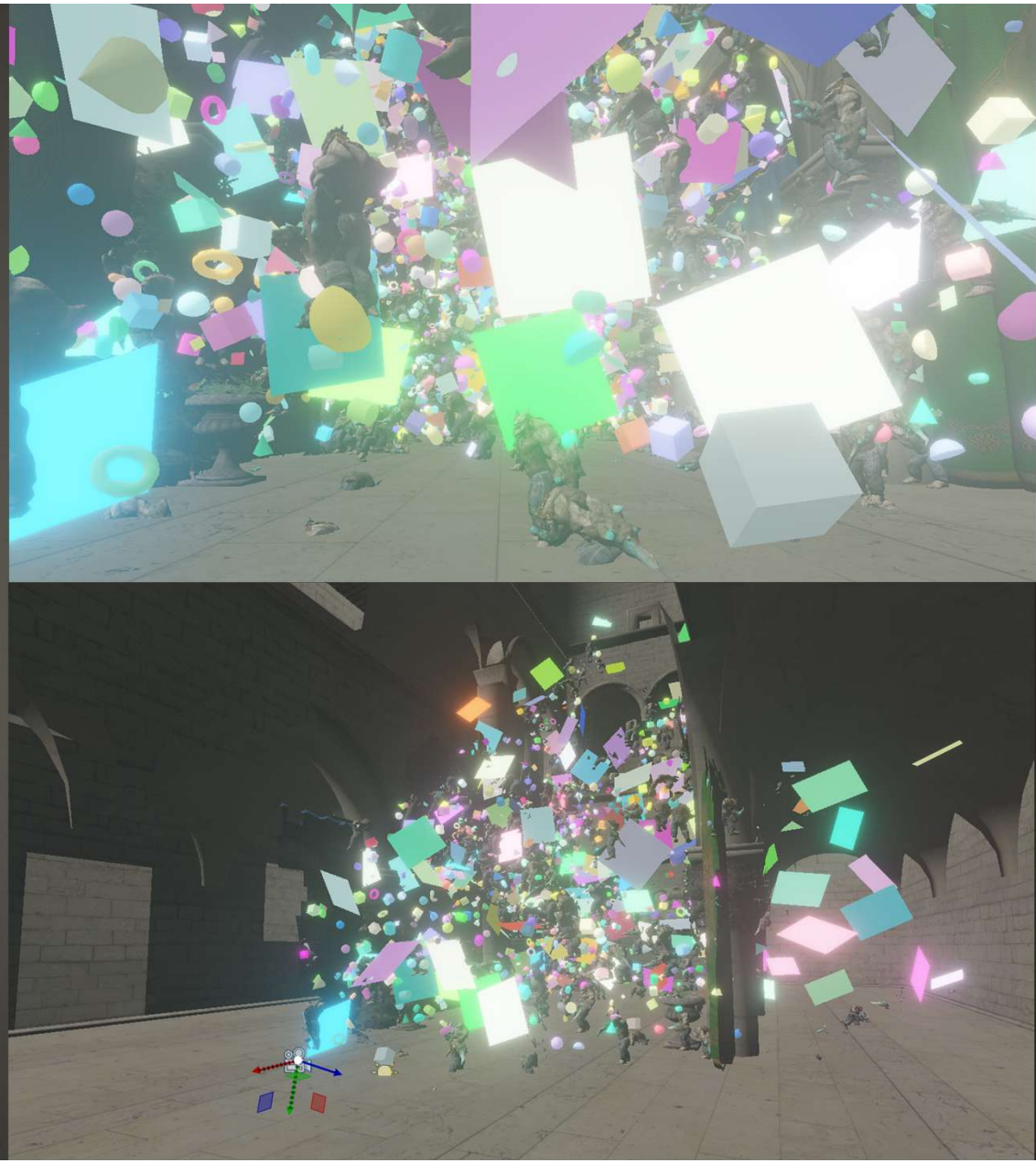
Animáció

- **Statikus és Animált objektumok közös kezelése** rendering shaderekben és culling compute shaderekben is!
- **Bindless Animáció architektúra**: **Statikus alap Model közös**, bindless módon érhető el a hozzá tartozó betöltött animáció shaderben.
- **Baked Animation**: Minden framehez előre ki van számolva az animációs mátrix, és a model/mesh/meshlet colliderek!
- Több **100.000** animált objektum, akár különböző animációval!



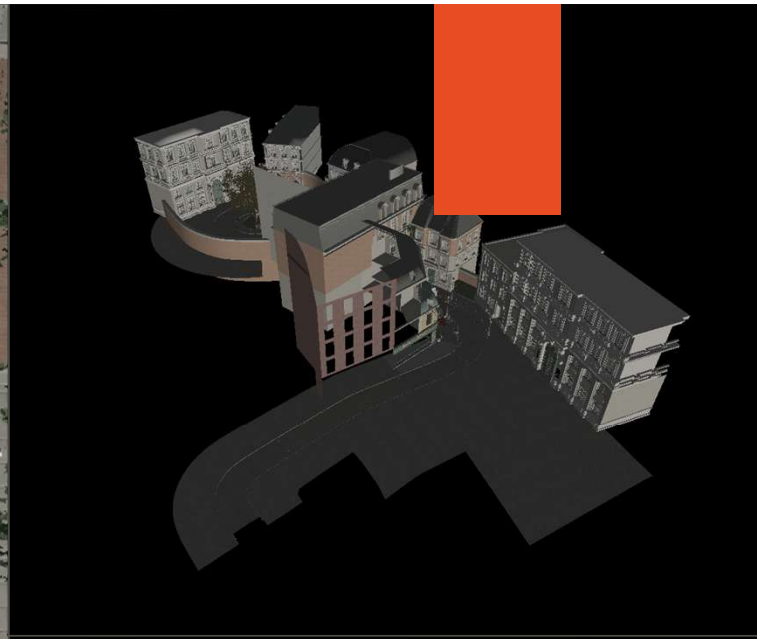
GPU-Driven Culling
Model Compute Pass

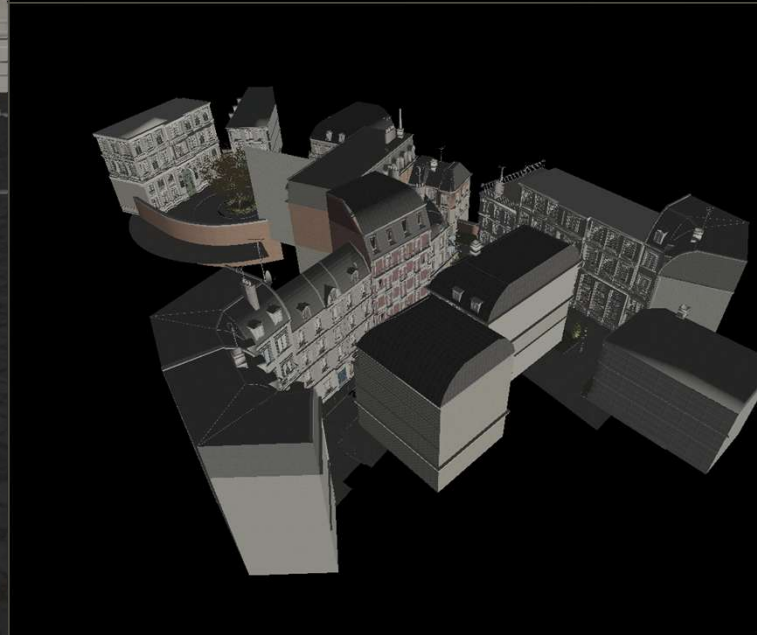
- Minden draw command GPU bufferben, tehát Compute Shaderből írható, láthatóságot és takarást vizsgálunk!!
- Draw Command **InstanceCount**-ot növeljük atomic módon, ha látható egy mesh + Instance index buffer frissítése az instanced renderinghez!
- **Model Culling**:
  - **Fast Path**: Ha kevés meshből áll egy model, akkor a model compute shaderben fogjuk a mesheket is feldolgozni.
  - **Slow Path**: Ha sok meshből áll a model, akkor egy gyüjtő bufferbe feljegyezzük az azonosítóját, majd egy következő körben futó kollaboratív mesh szintű compute logikával fogjuk feldolgozni a mesheket.
- Újdonság: **Work-Graph**!!



Mesh Culling Compute Pass

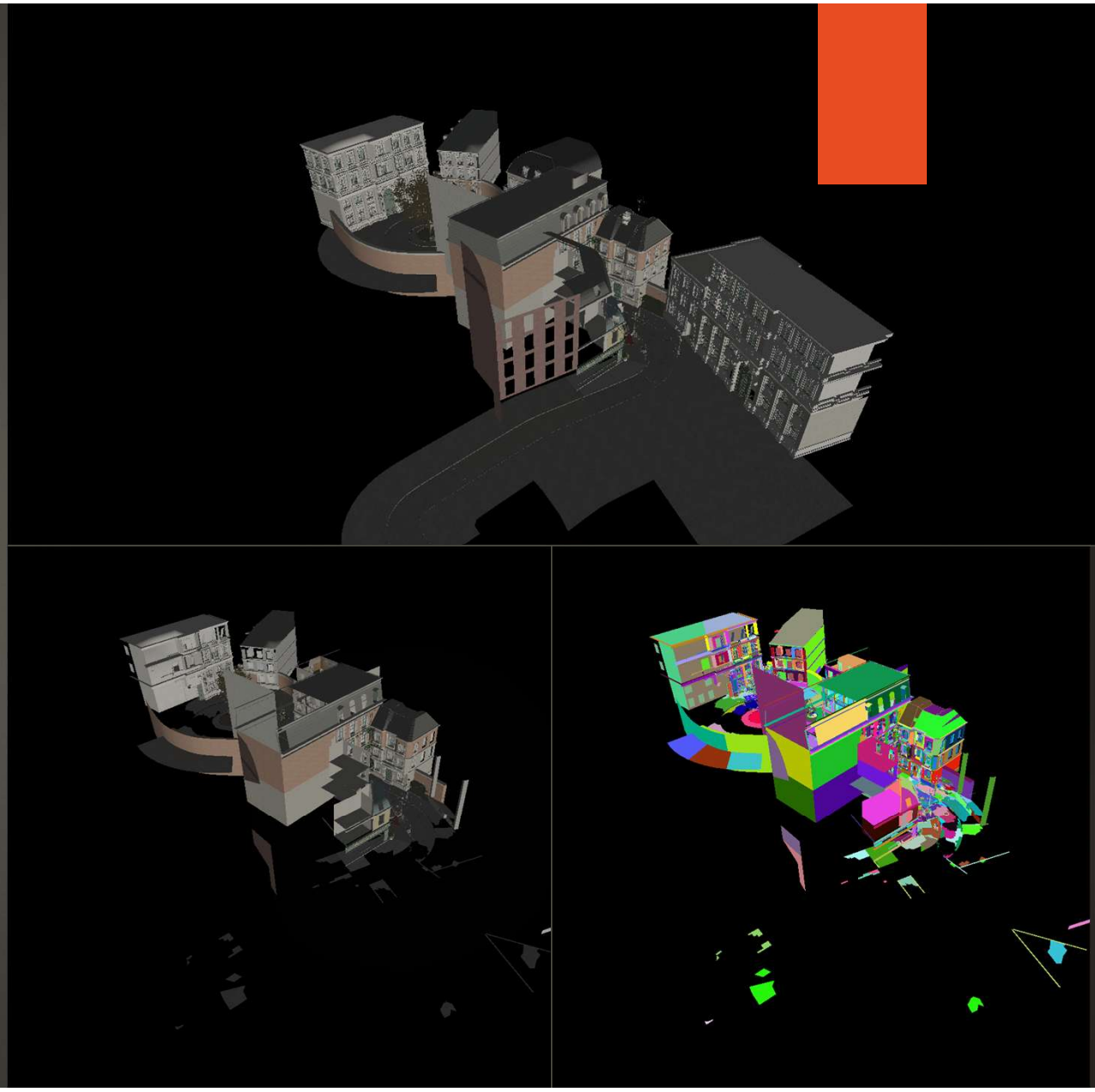
- **Collaborative** Compuite Mesh Culling.
- **32 thread** dolgozik 1 modellen, és dolgozzák fel a hozzá tartozó mesheket.
- Láthatóság, takarás vizsgálat.
- **LOD meghatározása**, material visszafejtése
- Draw Command frissítése





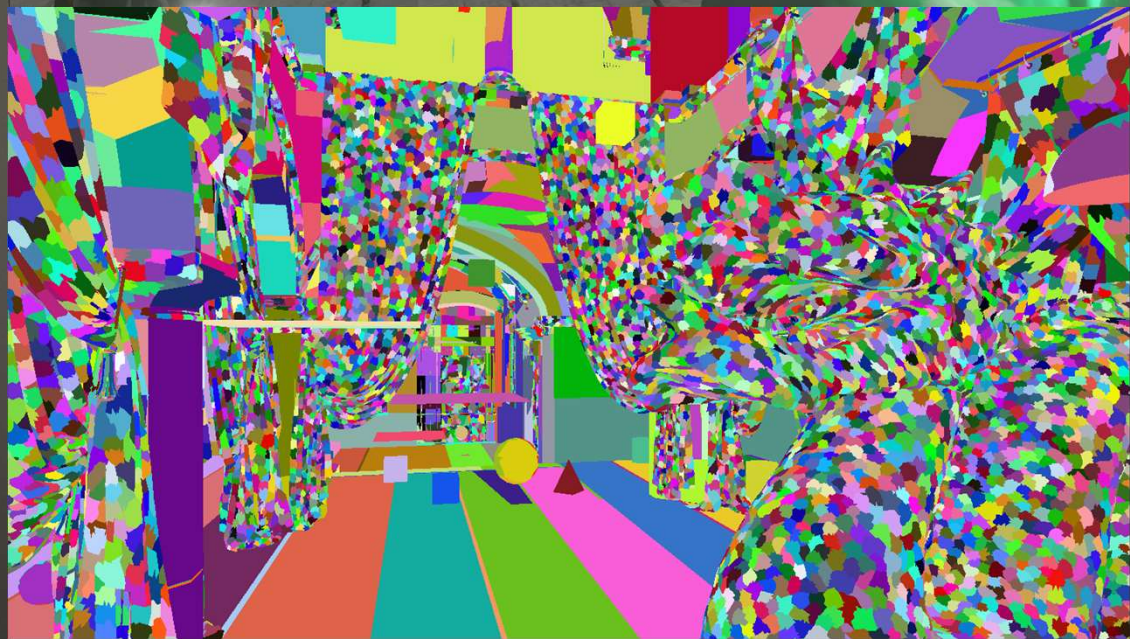
Meshlet Culling Task Shader Pass

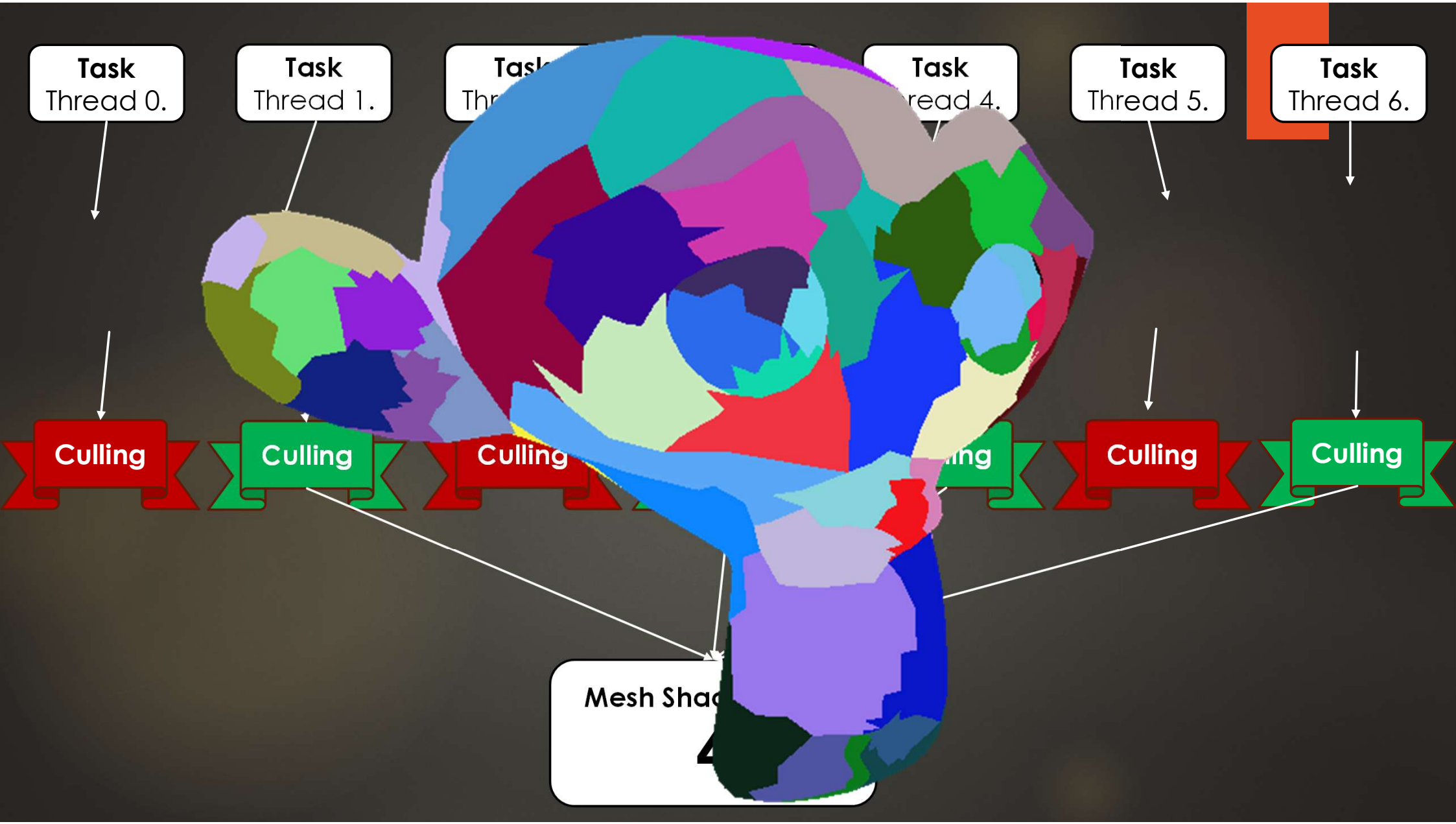
- **Task Shader:** Mesh shader előtt fut. Dinamikusan képes valahány db. mesh shadert indítani.
- **GroupX** = Látható mesh száma
- **GroupY** = Meshlet száma (32)
- **Task shader threadek meshleteken futnak** és vizsgálnak láthatósági és takarási teszteket.
- A látható, **túlélő meshletek** számát összegyűjtjük, és indítjuk a **mesh shadereket**.
- A mesh shaderek pedig egy meshlet vertexeit dolgozzák fel.

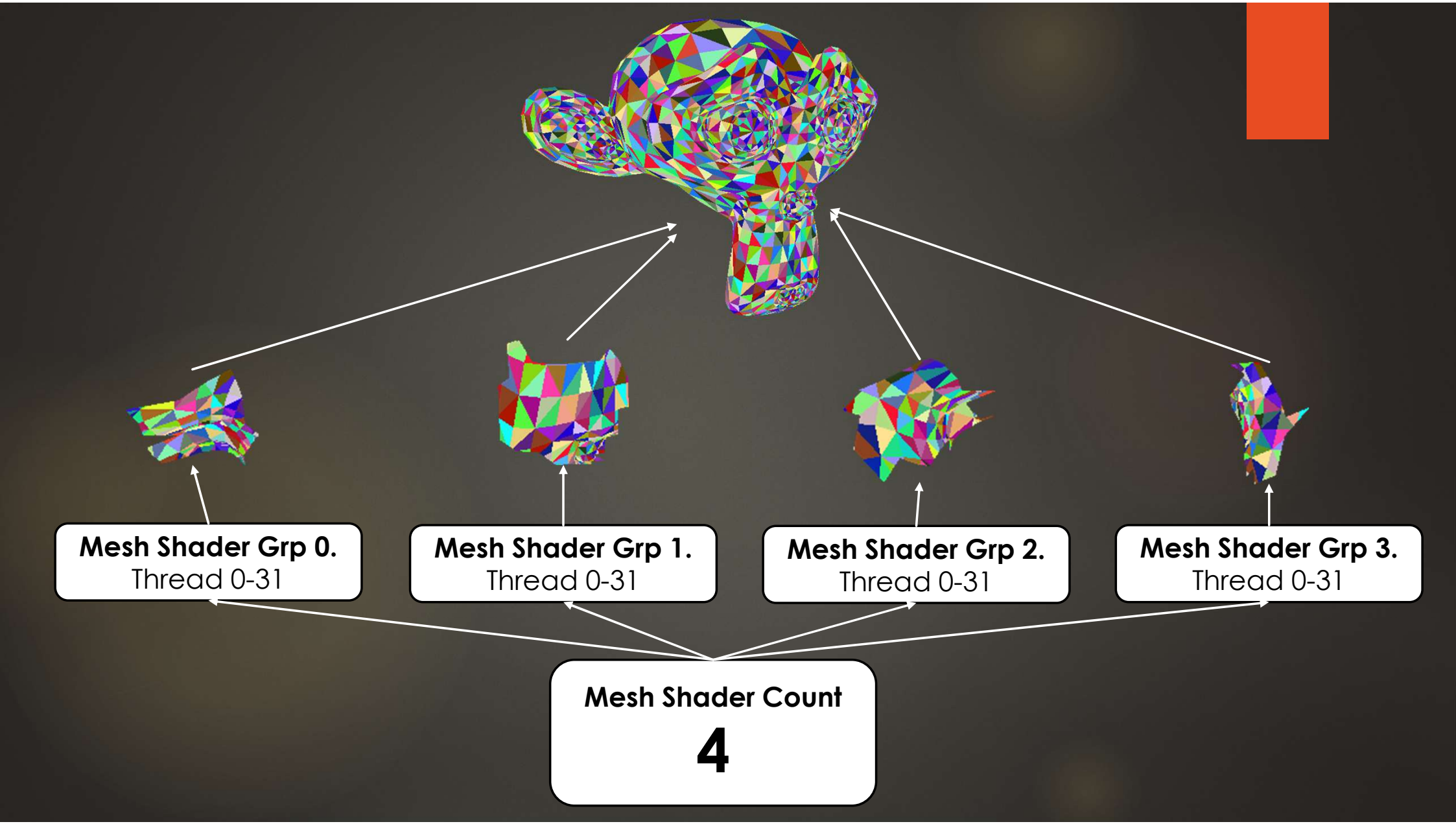


Meshlet + Mesh Shader

- **Mesh shader**: A régi grafikus pipeline-t helyettesíti újfajta compute alapú működéssel. Kollaboratív módon 32 szál dolgoz fel egy meshletet.
- **Meshlet**: 64 vertexből max 126 háromszögből álló kis geometriai felosztás. (Meshoptimizer generálja index buffer alapján)
- **Vertex caching**: GPU-driven renderingnél nincsen bindolt index buffer! Vertex shaderben ez lassulást jelent, de mesh shaderben nem!
- **Performancia**: **1.5-2.0x gyorsabb** a mesh shader a vertex shadernél.




Task
Thread 0.
Task
Thread 1.
Task
Thread 5.
Task
Thread 6.
Culling
Culling
Culling
Culling
Culling
Mesh Sha


Mesh Shader Grp 0.
Thread 0-31
Mesh Shader Grp 1.
Thread 0-31
Mesh Shader Grp 2.
Thread 0-31
Mesh Shader Grp 3.
Thread 0-31
Mesh Shader Count
4

GPU-Driven Culling Performance

- GPU-Driven culling és Rendering **~20x gyorsabb** a hagyományos CPU-Driven működéstől.
- 2 millió objektum esetén is a teljes gpu pipeline culling + rendering **1ms alatt**!!!
- **NVIDIA ResizeBar** engedélyezése szükséges!!!

Table 2: *CPU vs. GPU culling performance (1M entities).*

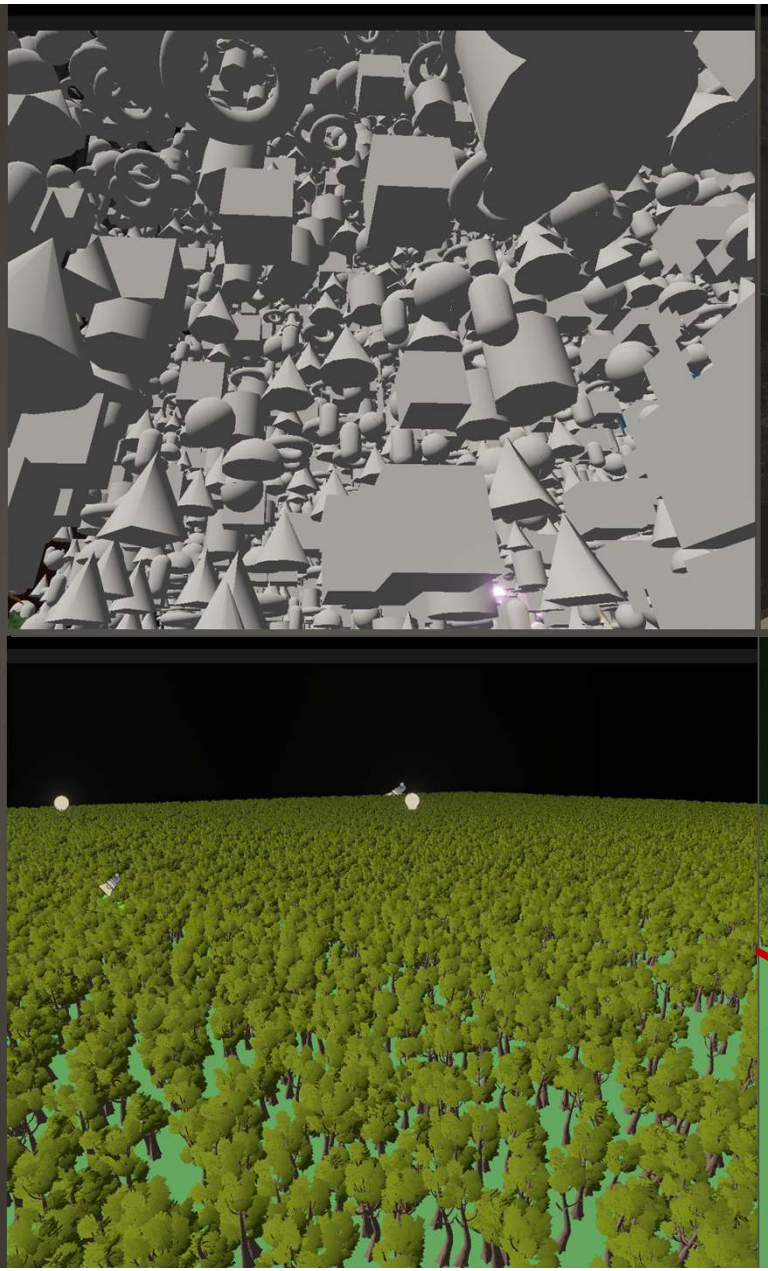
Scenario	CPU (ms)	GPU (ms)	FPS
CPU-Driven	**12.244**	-	**79**
– *Full Mesh*	-	3.481	-
– *Mesh (no occ.)*	-	4.504	-
– *Traditional*	-	7.687	-
GPU-Driven	**0.153**	**0.677**	**1344**

Table 4: *Hardware scalability on RTX 4060.*

Entities	Model Culling (ms)	Total GPU (ms)
10,000	0.013	0.253
100,000	0.050	0.549
1,000,000	0.360	0.671
2,000,000	0.680	0.992

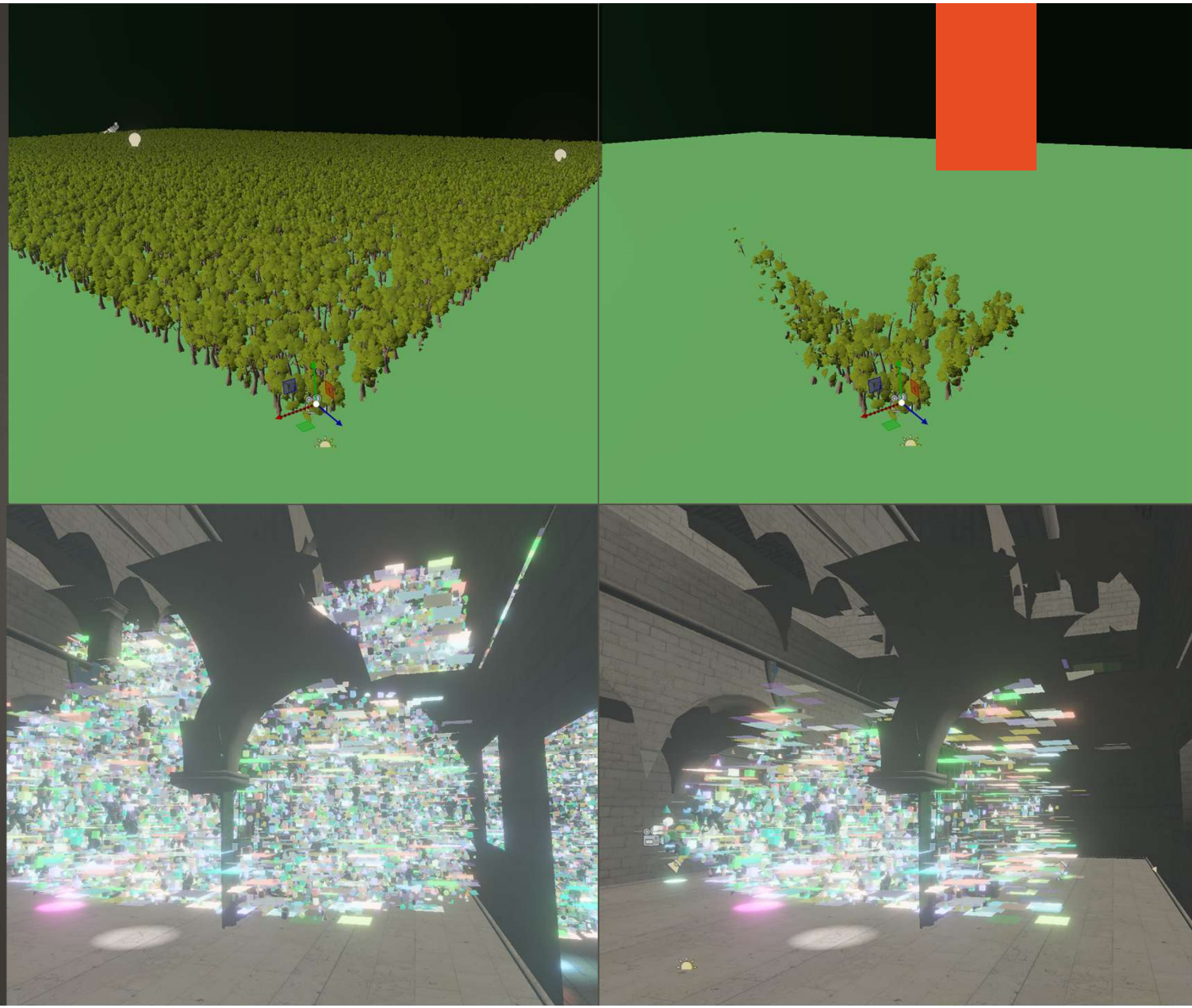
Frustum Culling

- Camera látógúla vs Model Sphere/AABB metszés teszt
- **Optimalizáció**:
  - Inside/Outside/Intersecting állapot.
  - Ha a **Model teljesen a frustumban**, akkor az összes Mesh és Meshlet is. **Nem kell frustum teszt alsó szinten**
  - Entity index 31. bit jelzi a **fully inside** állapotot!



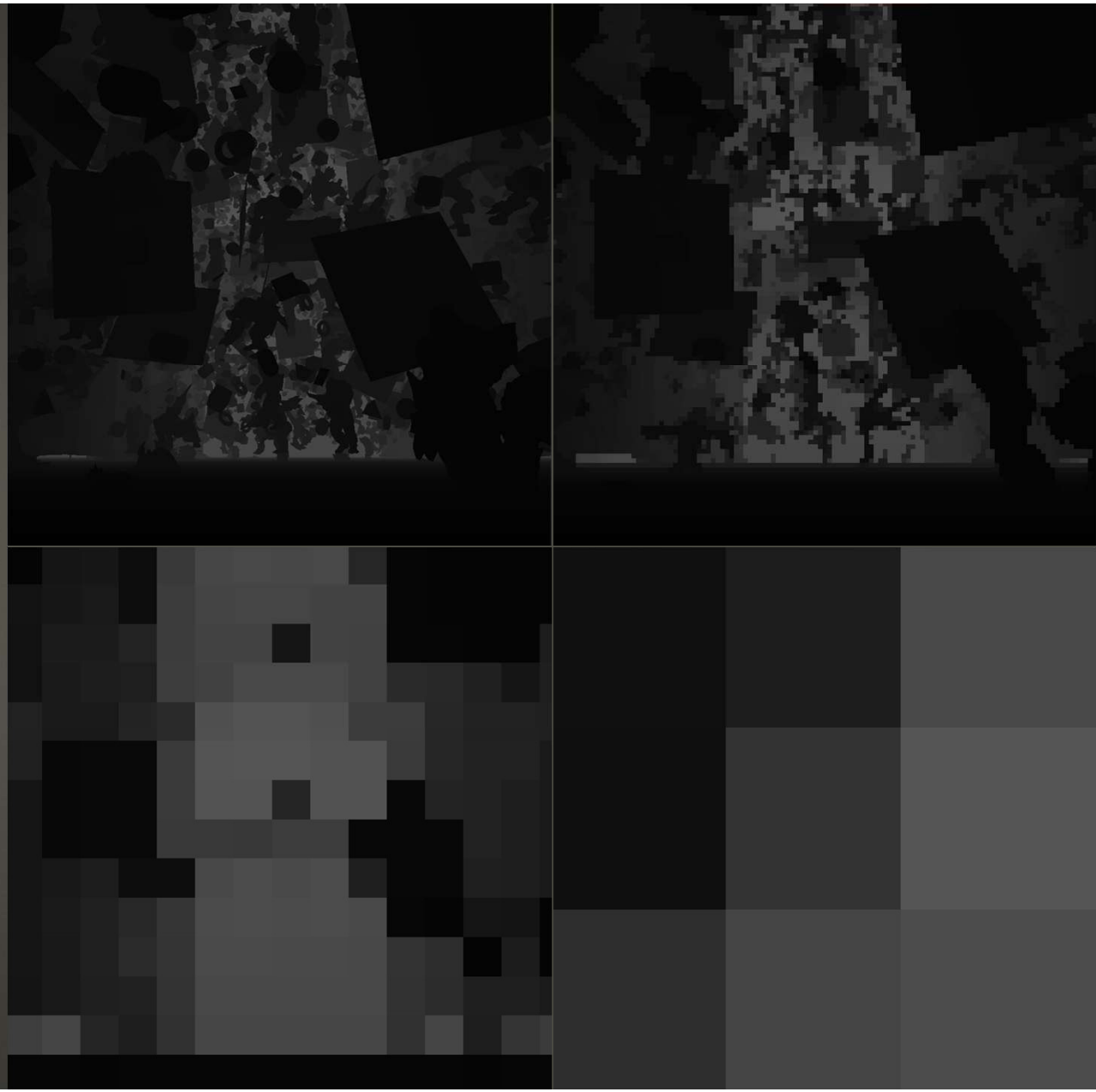
Occlusion Culling I.

- **Nem látható, kitakart objektumok kizárása.**
- Megvalósítás: Mélység textúra tartalmazza a szükséges geometriai adatokat!
- **Alkalmazása**:
  1. **Model szinten**: Kevésbé hatékony, túl nagy a model.
  2. **Mesh szinten**: Viszonylag hatékony, de még mindig nagyok a meshek
  3. **Meshlet szinten**: Nagyon hatékony.



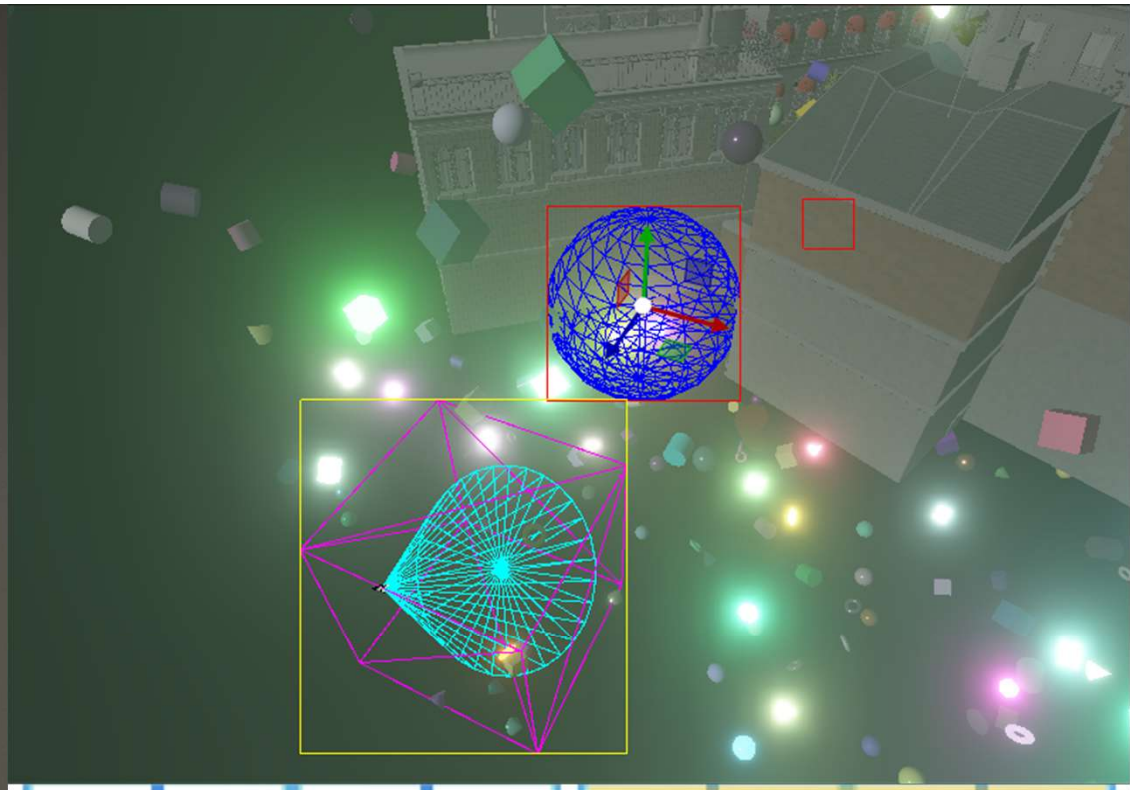
Occlusion Culling II.

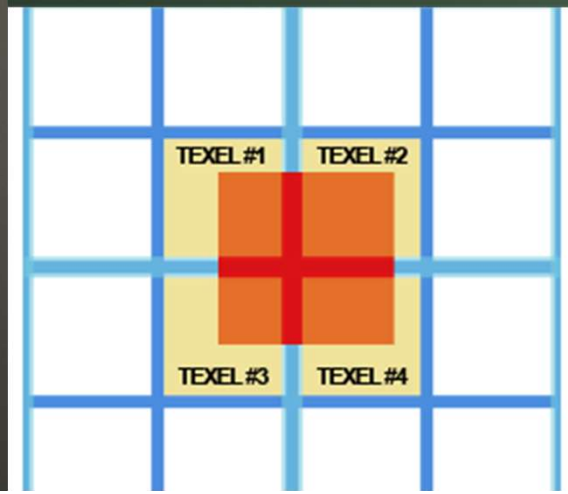
- Hierarchikusan max reduction mip chain készítése.
- **2x2 régiók maximumát vesszük**. (Széleken figyelni kell, hogy konzervatív maradjon!!)
- Ha egy objektum a méretéhez tartozó **maximum mélység érték mögött** helyezkedik el, akkor **takarásban van**!
- **Vulkan Max Reduction Sampler Extension**

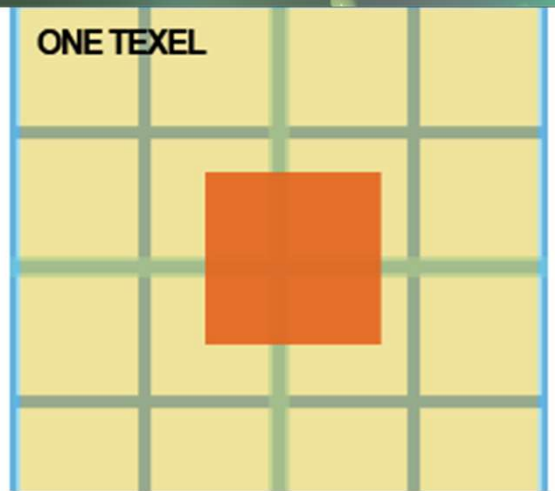


Occlusion Culling III.

- **Sphere Collidert projektáljuk Screen-Space** koordináta rendszerbe, ebből kapunk egy 2D AABB-t. (Zeux projection)
- **Textúra mintavételezés**:
  1. **Projektált AABB középpontja**: Textúra mintavételezési (u,v) pont.
  2. **X/Y mérete**: HiZ mip map level meghatározása
- **Fontos**: Mindig egy szinttel nagyobb textúrából kell olvasni, hogy a 4 sarok megfelelően legyen lefedve!
- **Zero-Pixel triangle culling**

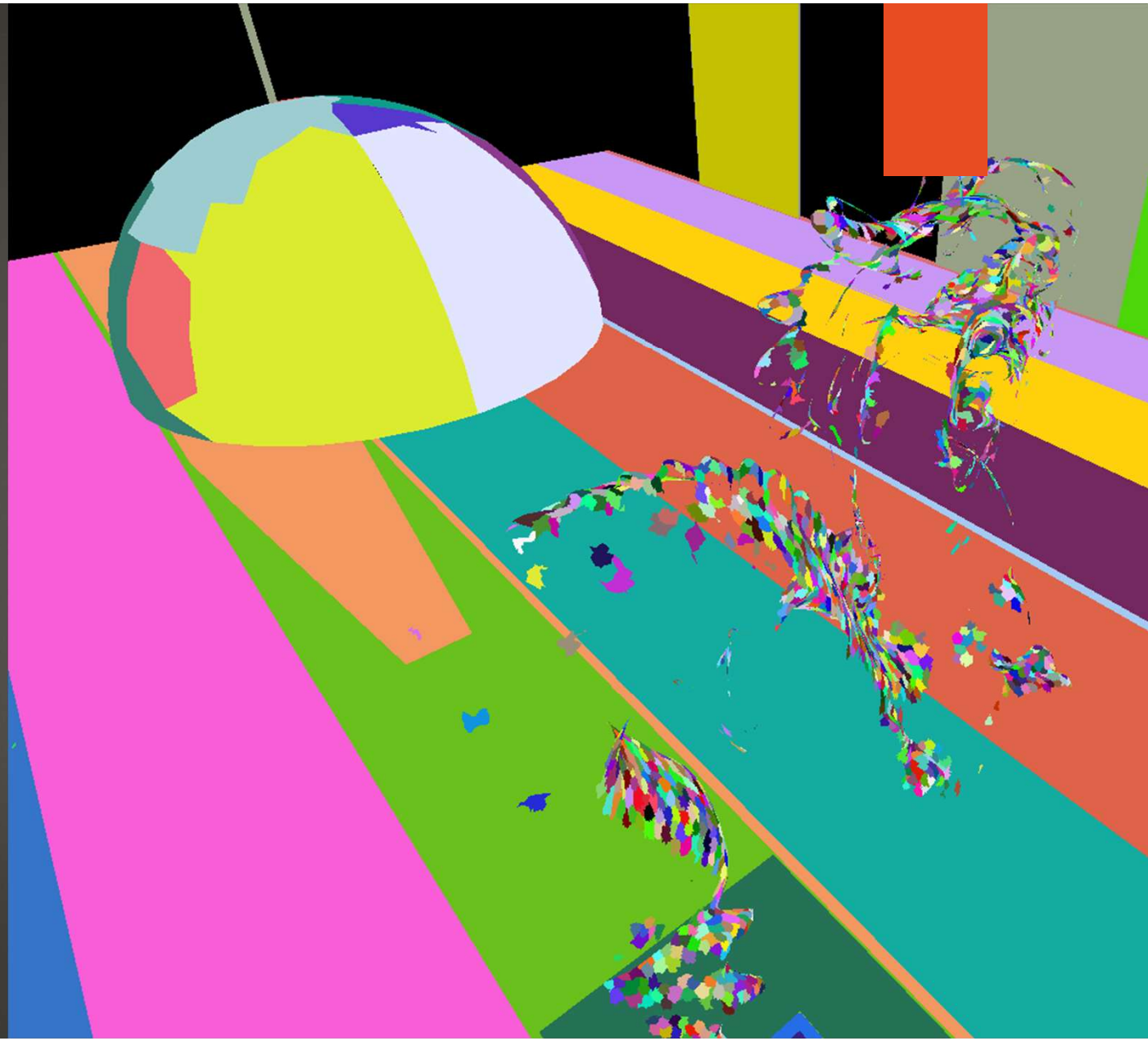






Meshlet Cone Culling

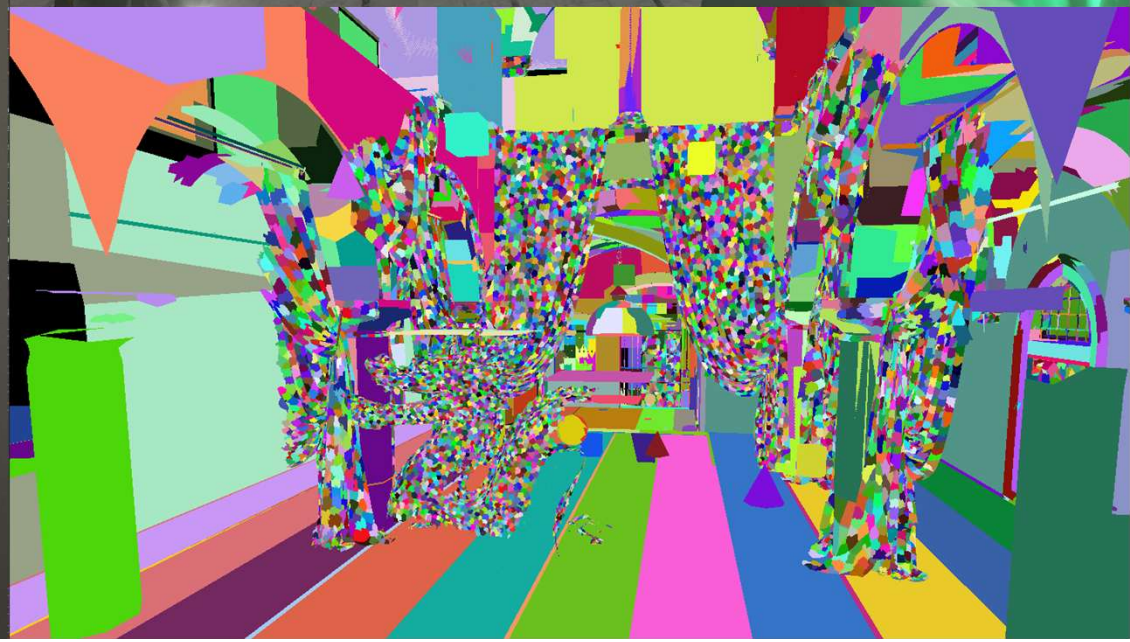
- Minden meshlethez egy **körbezáró kúp**. (Meshoptimizer generálja.)
- A meshletben lévő **háromszögek normálisának átlag iránya**, és a maximális eltérése szögben.
- **Backface culling**: Kamera iránya, és a kúp iránya és kiterjedéséből eldönthető hogy a hátrafele néz-e a meshlet.



Culling Végeredmény, Performancia

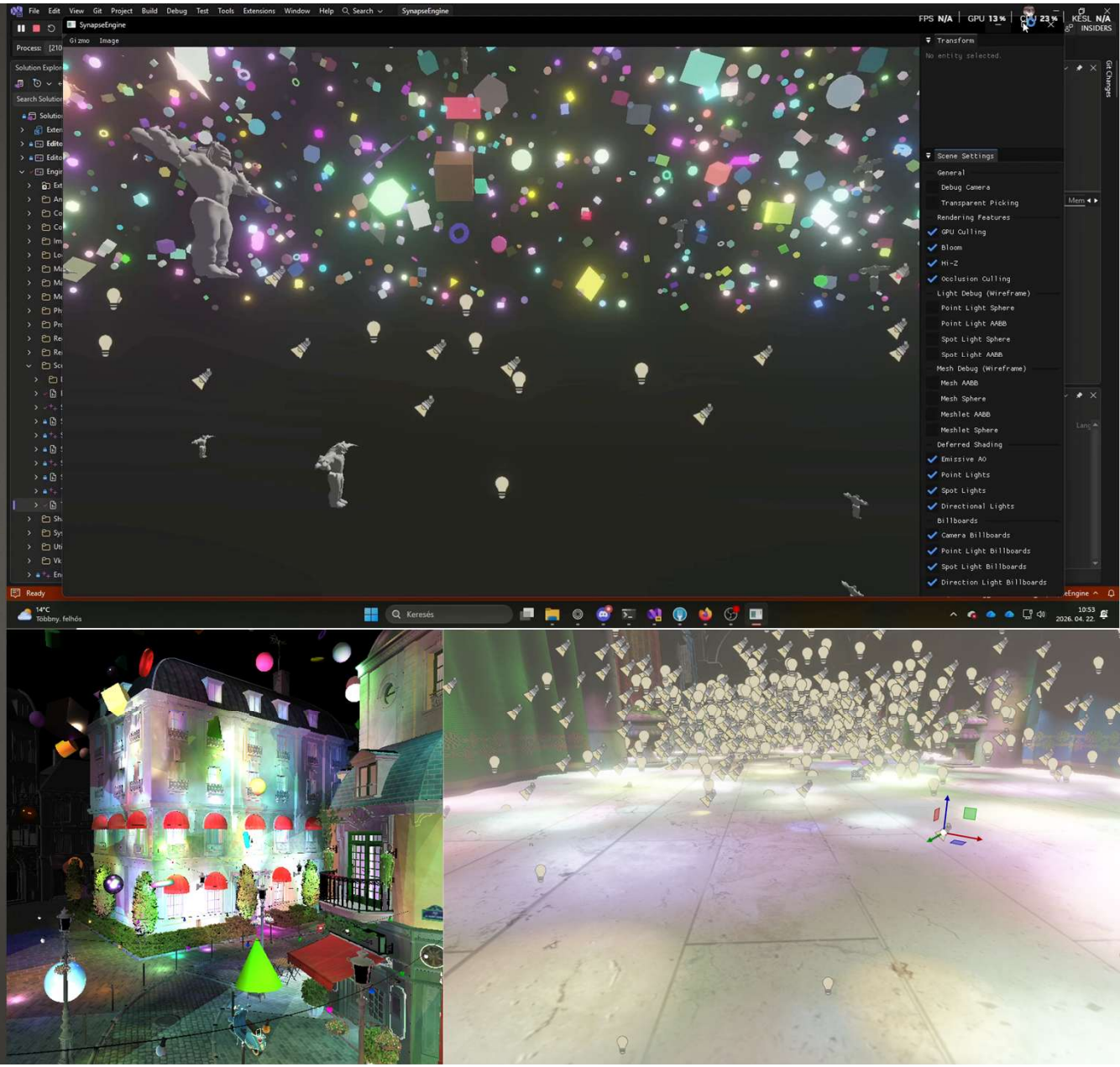
- Hierarchikusan **kizártuk a nem látható modeleket, meshеket,** és nagyon alacsony szinten a **meshleteket** teljesen a GPU-n compute és task shadereket használva.

- Culling nélkül: 124.407 ms
- + Cone Culling: 94.123 ms
- + Frustum Culling: 2.878 ms
- + Occlusion Culling: 0.681 ms
- + Dynamic LOD: **0.653 ms**



További fő fejlesztések

- Fizikai szimuláció: Jolt Physics **(Kész)**
- Light szimuláció:
  - Deferred Shading **(Kész)**
  - Forward+ Shading **(In Progress)**
- GPU-Driven shadow: shadow light culling and rendering **(Not Started)**
- **+ Rengeteg minden más ☺**



Kérdések?

Köszönöm a megtisztelő figyelmet!

SOURCE CODE: HTTPS://GITHUB.COM/TAMASPETII/SYNAPSEENGINE