Optimizing Mobile Games with Gameloft and ARM

Stacy Smith

Senior Software Engineer, ARM

Adrian Voinea

World Android Technical Lead, Gameloft

Victor Bernot

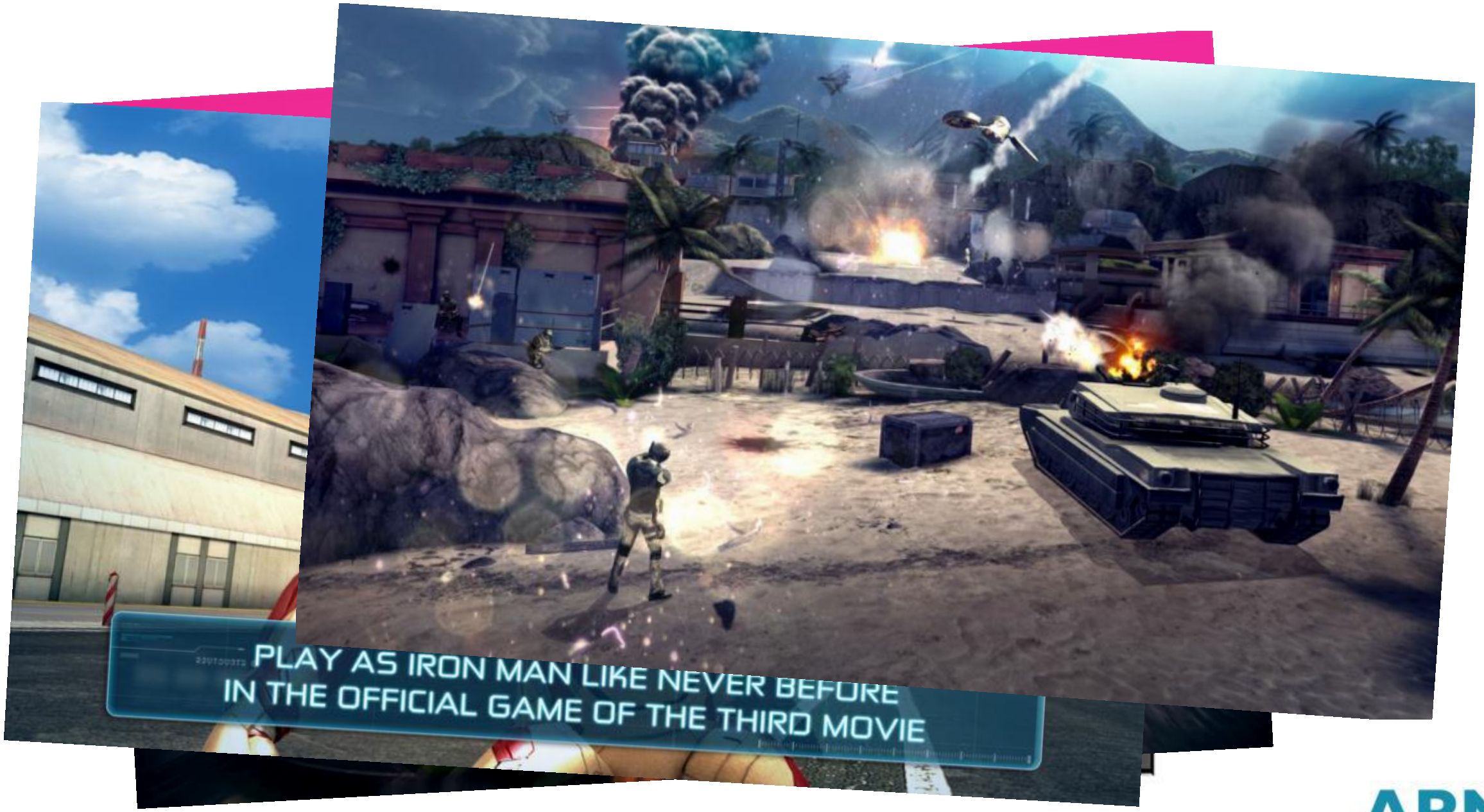
Lead Visual Effects Developer, Gameloft

ARM Ecosystem

- My first role in ARM was in Developer Relations
- Developers came to us to ask for help
- We couldn't share their problems with the world
- We couldn't help everyone on a one to one basis

Developer Education

- Developer Education addresses that need
- Since 2012 we've been sharing advice for graphical development
- Developers working with Developer Relations have remained separate
- Until now!


PLAY AS IRON MAN LIKE NEVER BEFORE
IN THE OFFICIAL GAME OF THE THIRD MOVIE

ARM

Today's Agenda

- Gameloft
  - Batching (Iron Man 3)
  - Culling and LOD (Gangstar Vegas)
  - Texture compression (Asphalt 8)

- ARM (That's me)
  - Entry level implementations
  - How to achieve similar results

ARM

Iron Man 3

Improving Draw Calls and Rendering Techniques

ARM

Sorting Objects Before Rendering

- There is no good or bad way to determine which sorting method works best for a game
- Sorting methods reduce overdraw and material changes at the cost of CPU
- Sorting algorithms used for Iron Man 3:
  - Sorting by material
  - Sorting by distance

What Happens When No Sorting Is Applied?

- Mid-range device: average 18FPS, constant micro-freezes
- Over 35 program changes per frame
- The skybox is rendered in the 27th draw call / 150

ARM

Sorting Objects Before Rendering

- Every shader program change is costly
- It will depend a lot on the number of programs your game has
- Using a texture atlas will create the same materials, thus allowing better material sorting
- Sorting front to back will reduce the overdraw

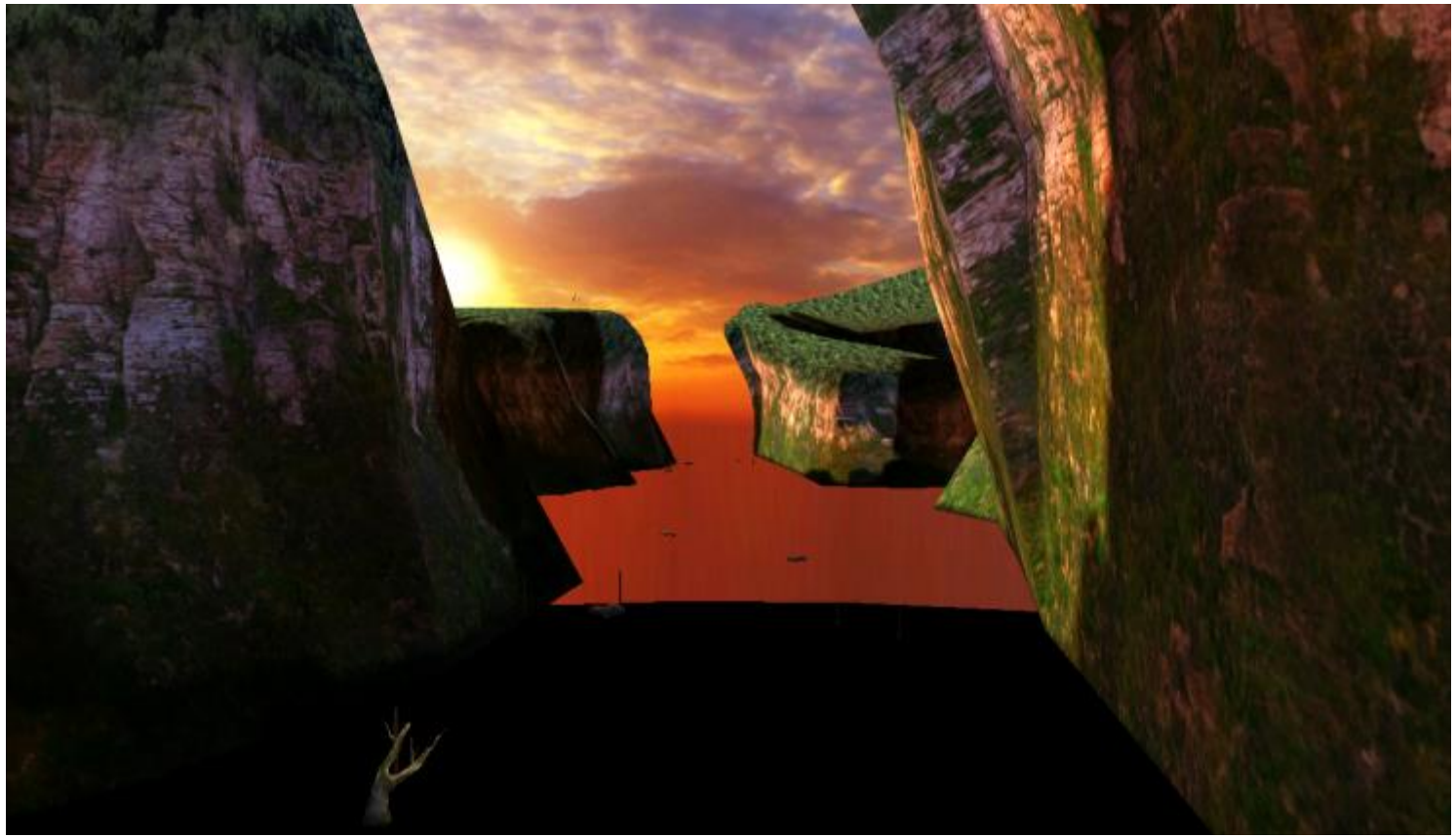
Material Sorting Results

- Reduced program changes to an average of 16
  - Micro-freezes are reduced.
- Average 22FPS, smoother gameplay

ARM

Material Sorting Results

- But we still have a lot of overdraw…



ARM

Front to Back Sorting

- And this is the desired result…

ARM

Front to Back Sorting

- Sorting by distance
  - Front to back sorting will reduce the overdraw, skipping fragment shader execution for all the dropped fragments
  - The transparent objects must be drawn back to front for a correct ordering. The blending is done with what is already drawn in the color buffer
  - For OpenGL® ES 3.0, avoid modifying the depth buffer from the fragment shader or you will not benefit from the early Z testing, making the front to back sorting useless

Front to Back Sorting Results

- Sorting first by material, objects with the same material then sorted front to back
- Constant 24FPS
- The skybox is rendered in the 82nd draw call / 135, being the last opaque object

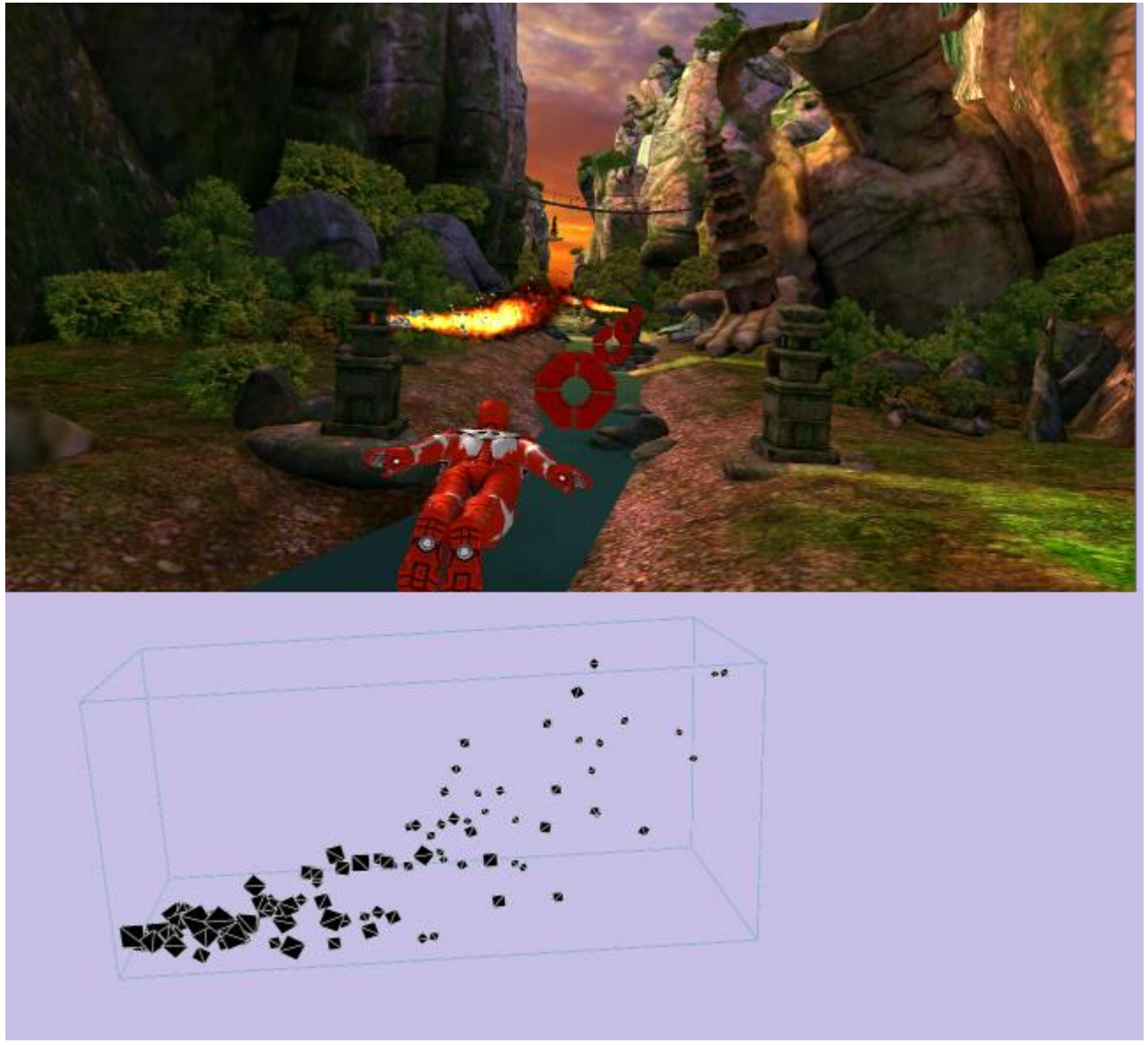
ARM

Dynamic Batching

- Objects that are not static batched and by nature require dynamic movement can be batched at runtime
- Objects need to share the same material to be batched into one single draw call
- Dynamic batched objects require new geometry and the duplication of all the vertex attributes, adding a constraint on the number of vertices

Dynamic Batching

- Should be applied to dynamic objects that share the same material, with a constraint on number of vertices, like particles



ARM

Dynamic Batching

- In order for objects to be batchable at runtime:
  - Objects need to use the same program
  - Objects require identical textures
  - Because of draw order constraint, objects require the same transparency property
  - Because of duplicating vertex attributes, dynamic batching on objects requires a lower number of vertices

Sorting Objects Before Rendering

- Good approach to render your scene:
  - Sort all opaque objects based on material and distance from camera
  - This reduces scene overdraw
  - Reducing material change will help a bit on FPS, depending on the hardware, but also helps with micro-freezes
  - For every consecutive draw call that has the same material, dynamic batching is simple and easy to achieve
  - Reduces the actual number of draw calls
  - Reduced micro-freezes and gained ~6FPS by applying sorting algorithms

The Importance of Batching

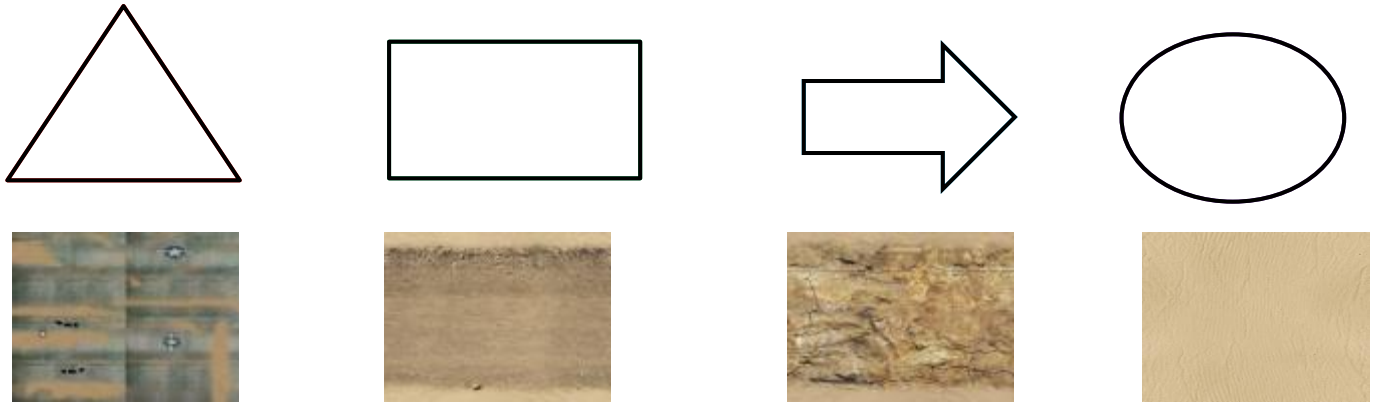
ARM

Batching

- Deferred immediate mode rendering
- glDrawElements and glDrawArrays have an overhead
- Less draw calls, less overhead
- DrawCall class stitches multiple objects into one draw call
- Macro functions in shaders make batching as simple as:

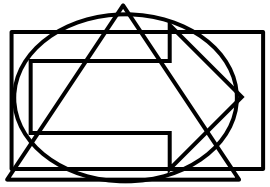
```
vec4 pos=transform[getInstance()]*getPosition();
```

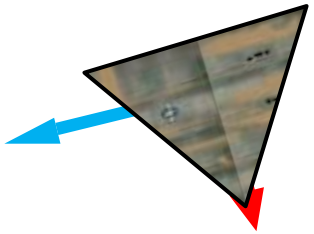
Batching

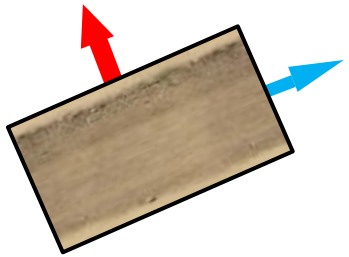


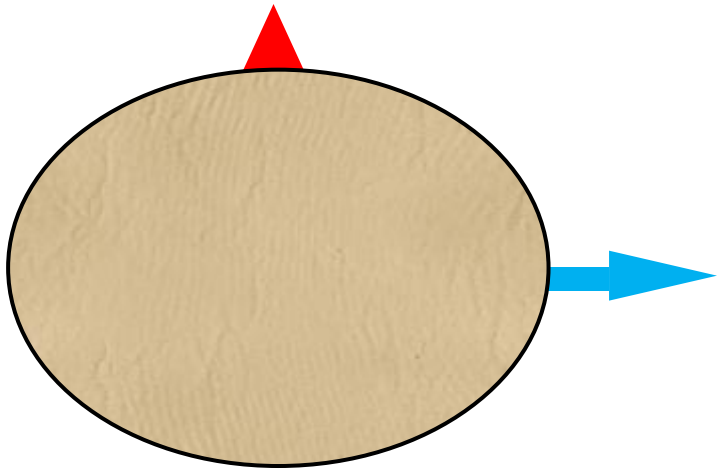
ARM

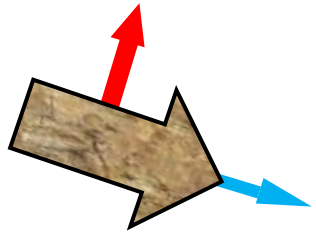
Batching











uniform mat4 transforms[4];

Mixing up a Batch

Mesh 1 [(1,1),(0,1),(0,0),(1,0)]
Mesh 2 [(1,2),(0,2),(0,0)]
Mesh 3 [(2,2),(2,1),(1,1)]
Index1: [0,1,2,0,2,3]
Index2: [0,1,2]
Index3: [0,1,2]

Attrib1:[(1,1),(0,1),(0,0),(1,1),(1,2),(0,0),(1,0),(2,2),(2,1),(1,1)]
Attrib2:[0,0,0,0,1,1,1,2,2,2]

Index: [0,1,2,0,2,3,4,5,6,7,8,9]

ARM

Serving up a Batch

Vertex shader:

```
uniform mat4 transforms[3];
attribute vec4 pos;
attribute float id;
void main(){
    mat4 trans=transforms[(int)id];
    glPosition=trans*pos;
}
```

GL Code:

```
float transforms[16*instanceCount];
.
. /* Load matrices into float array */
.
glUniformMatrix4fv(transID,4,false,transforms);
```

Scene Batching

- Multiple geometries drawn in a single draw call:
- `drawbuilder.addGeometry(geo1);`
- `drawbuilder.addGeometry(geo2);`
- `drawbuilder.addGeometry(geo3);`
- `drawbuilder.Build();`

- Always draws in the same order from a start and end index. Unused objects can be scaled out to zero
- Can lead to inefficiency of vertex shaders

Remixing a Batch

Mesh 1 [(1,1),(0,1),(0,0),(1,0)]
Mesh 2 [(1,2),(0,2),(0,0)]
Mesh 3 [(2,2),(2,1),(1,1)]
Index1: [0,1,2,0,2,3]
Index2: [0,1,2]
Index3: [0,1,2]

Attrib1:[(1,1),(0,1),(0,0),(1,1),(1,2),(0,0),(1,0),(2,2),(2,1),(1,1)]
Attrib2:[0,0,0,0,1,1,1,2,2,2]

Index: [0,1,2,0,2,3,4,5,6,7,8,9]

ARM

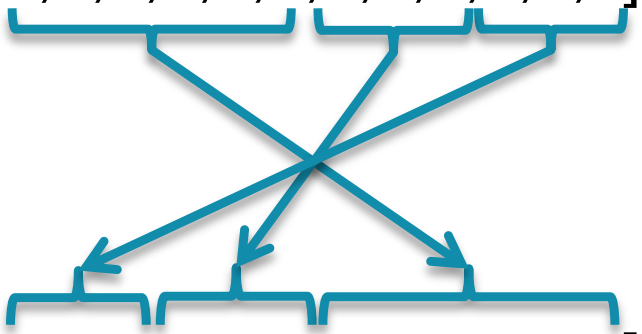
Remixing a Batch

Attrib1:[(1,1),(0,1),(0,0),(1,1),(1,2),(0,0),(1,0),(2,2),(2,1),(1,1)]

Attrib2:[0,0,0,0,1,1,1,2,2,2]

These IDs are the same

Index: [0,1,2,0,2,3,4,5,6,7,8,9]



Index: [7,8,9,4,5,6,0,1,2,0,2,3]

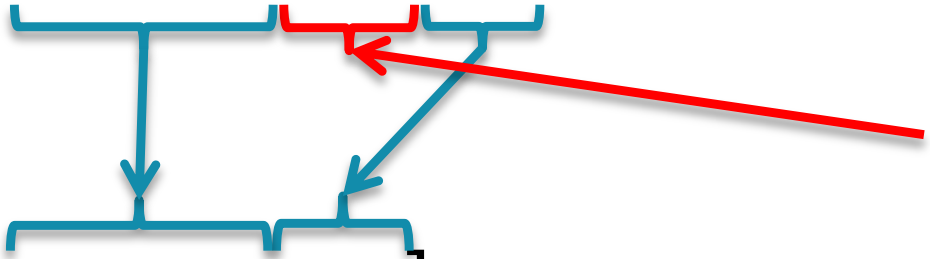
But now the draw order is different

Remixing a Batch

Attrib1:[(1,1),(0,1),(0,0),(1,1),(1,2),(0,0),(1,0),(2,2),(2,1),(1,1)]

Attrib2:[0,0,0,0,1,1,1,2,2,2]

Index: [0,1,2,0,2,3,4,5,6,7,8,9]



Index: [0,1,2,0,2,3,7,8,9]

These vertices are still processed

…But never get drawn

Object Instancing

- Multiple geometries or single object instances:

```
for(int i=0;i<50;i++)
  drawbuilder.addGeometry(geo1);
drawbuilder.Build()
```

- Can implement LOD switching when objects are sorted front to back and correctly culled

- So let's talk about culling

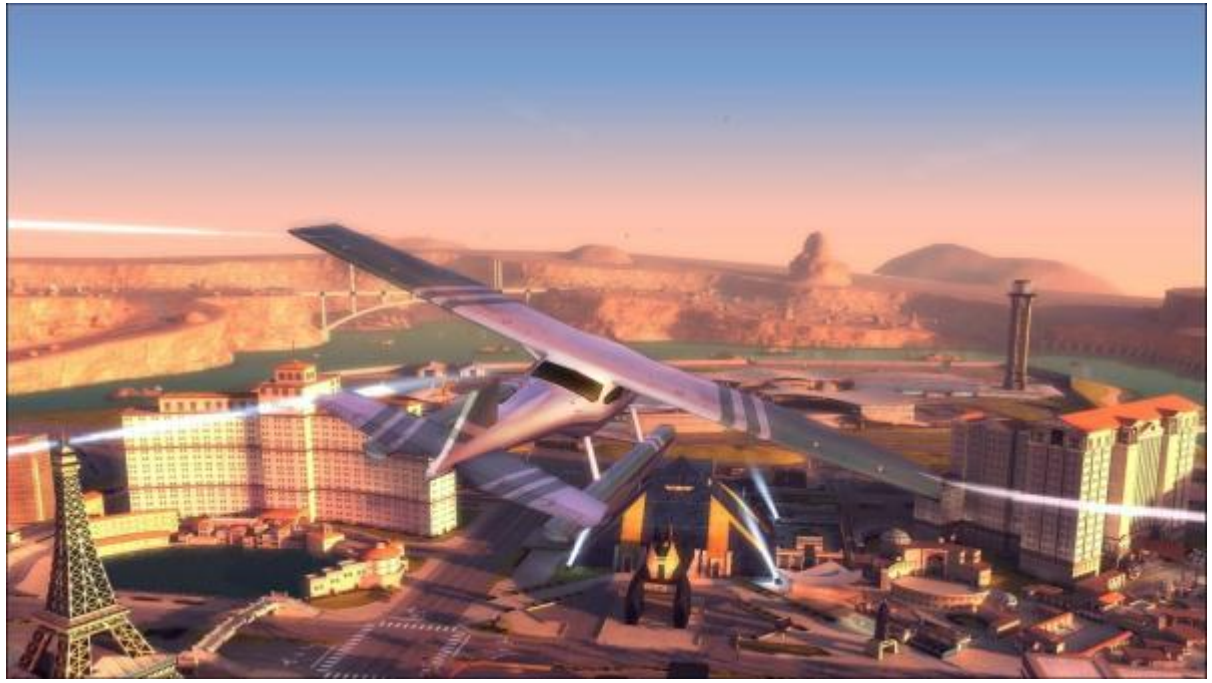
Batching / Draw Call Reduction

Case Study: Gangstar Vegas

ARM

Challenges

- Big map: ~16 km²
  - Move by foot / by car / by plane
  - Interior/exterior environments



ARM

Streaming

- Constant streaming is needed :
  - Meshes / LODs
  - Streaming thread with second OpenGL® context
  - Textures
  - Same streaming thread as meshes
  - Lower mipmap level always available

- Also streamed :
  - Animations
  - Sounds
  - Physics data

Reducing Draw Call Count & Cost

- Draw calls are one of the major bottlenecks on mobiles
  - On a very powerful device, stay under 300 draw calls
  - On most devices => 100 to 200 at max
- Techniques used to reduce draw call count:
  - Static batching
  - 2D grid culling
  - Near/far city
- To reduce draw call cost:
  - Index & Vertex Buffer Objects (IBO / VBO) for static geometry

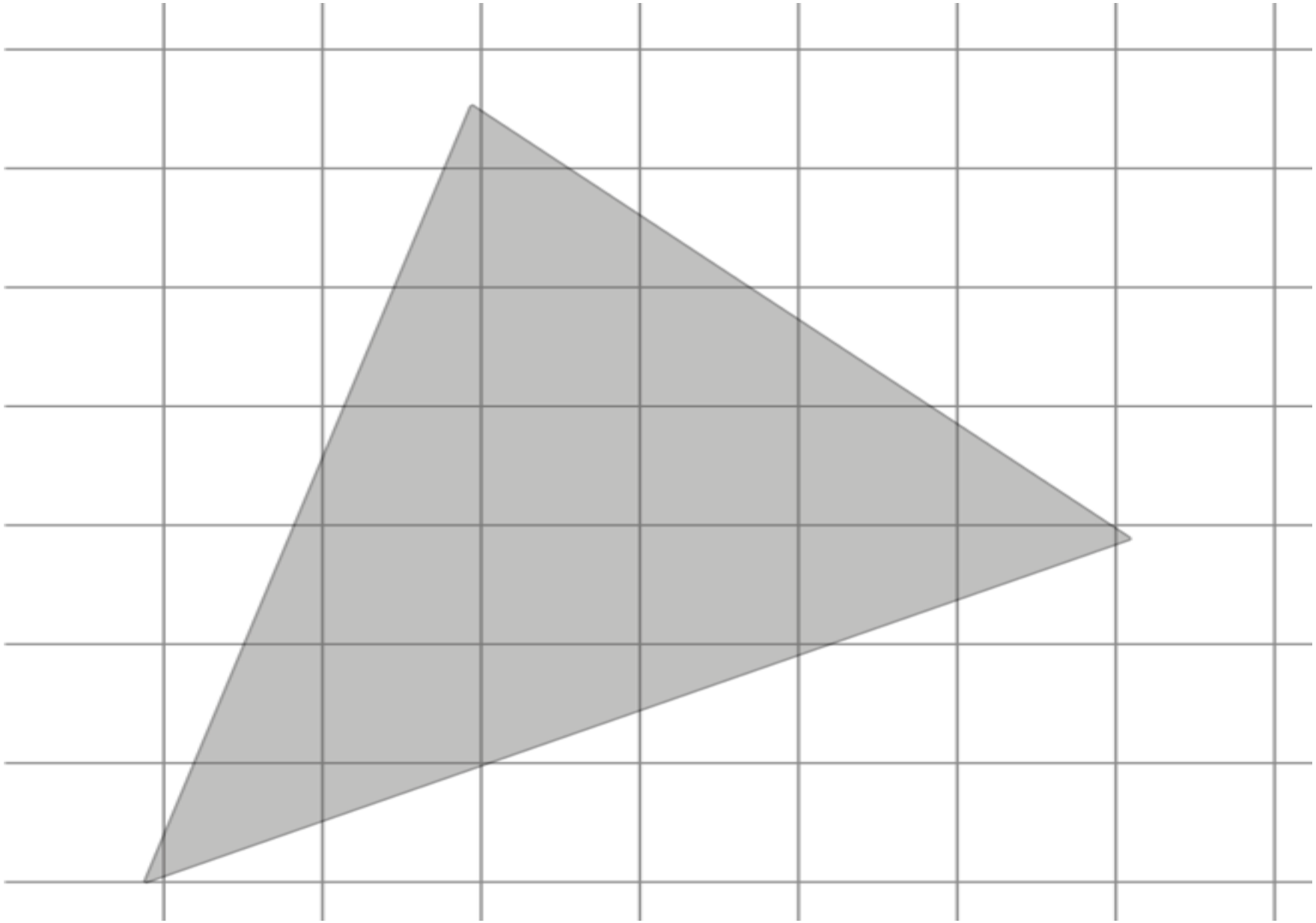
2D Grid Culling

- Objects are linked to the cells of a grid
  - A simple 2D grid is used:
  - 128x128 cells
  - Cell size : 30m x 30m
  - An object can belong to one or more cells

- Culling process:
  - Get cells visible by camera's frustum 2D projection on grid
  - Middle cells: fully in frustum
  - Border cells: partially in frustum
  - Middle cell objects: directly added to render list
  - Border cell objects: AABBoxes are tested vs camera frustum

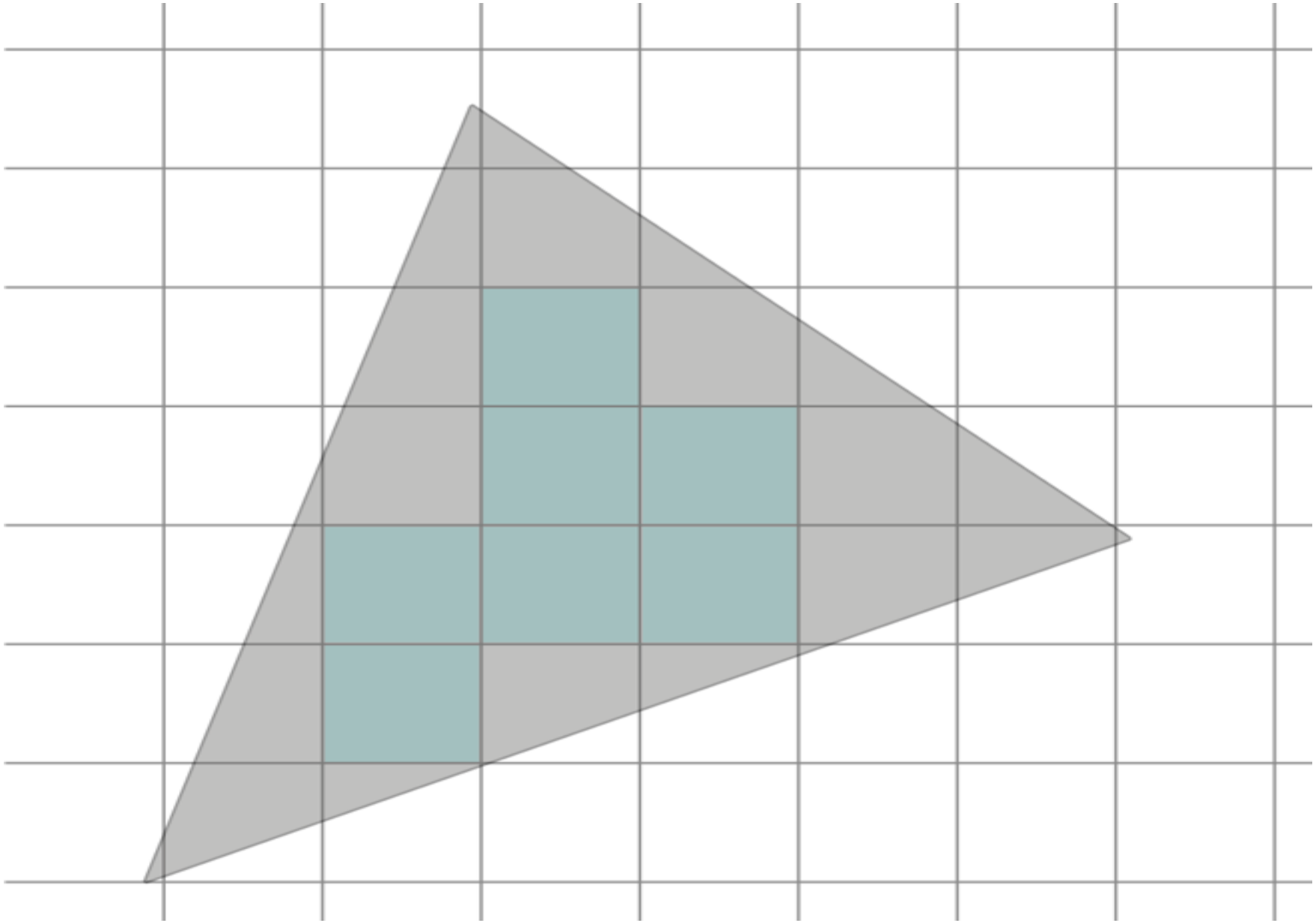
- Grid is also used for streaming:
  - Radius Add (175m) / Radius Safe (205m)

Frustum & Grid



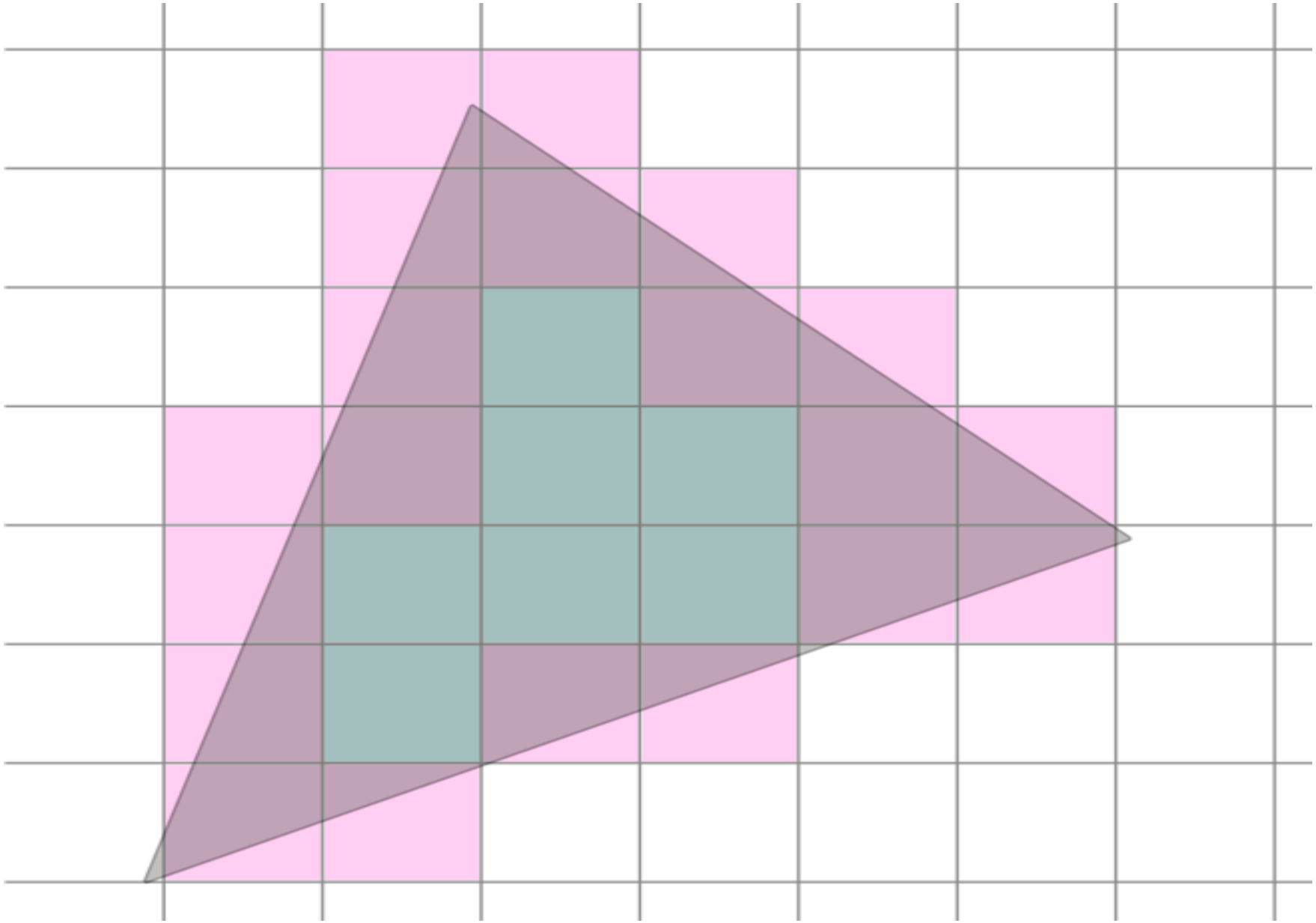
ARM

Middle Cells



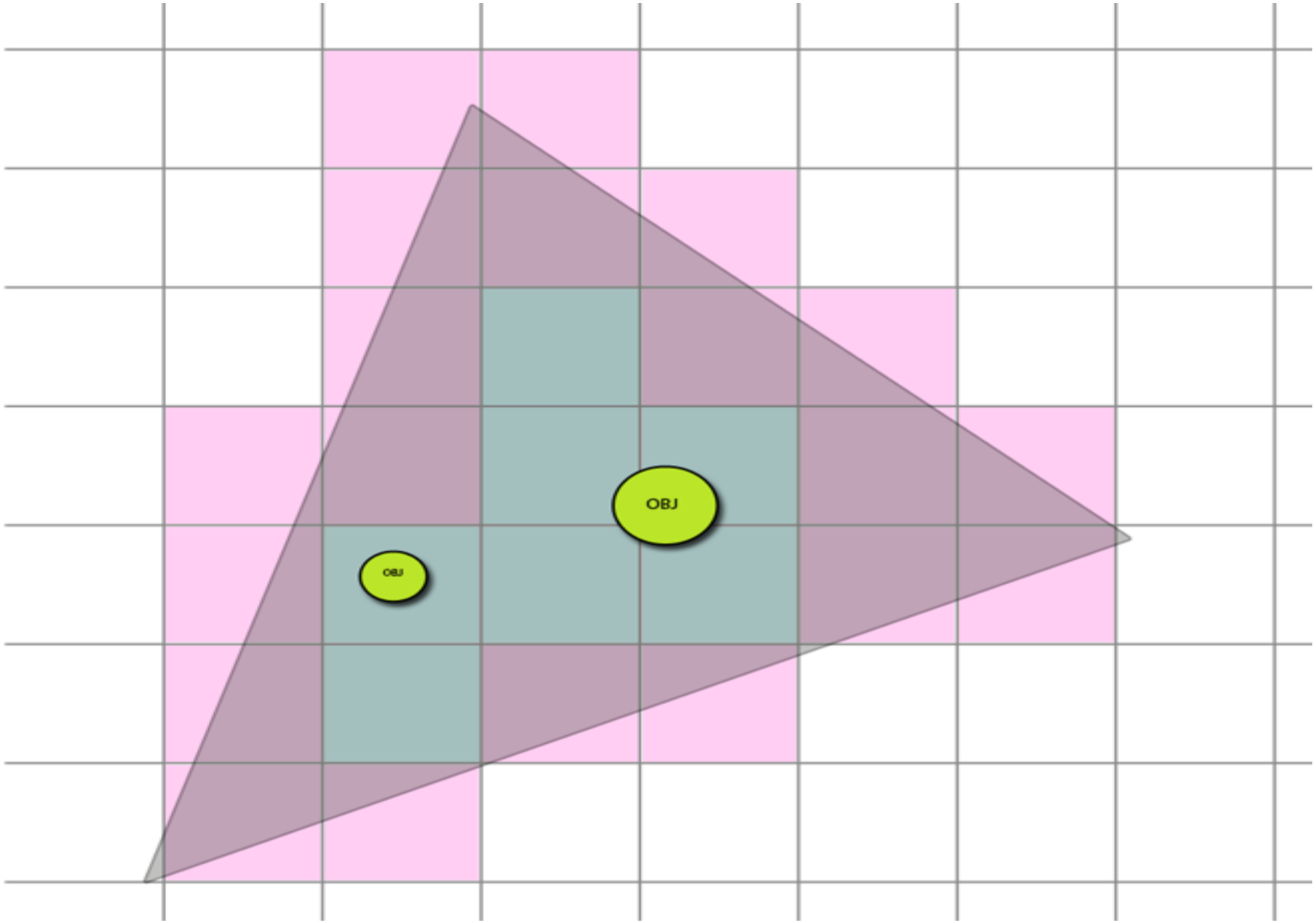
ARM

Border Cells



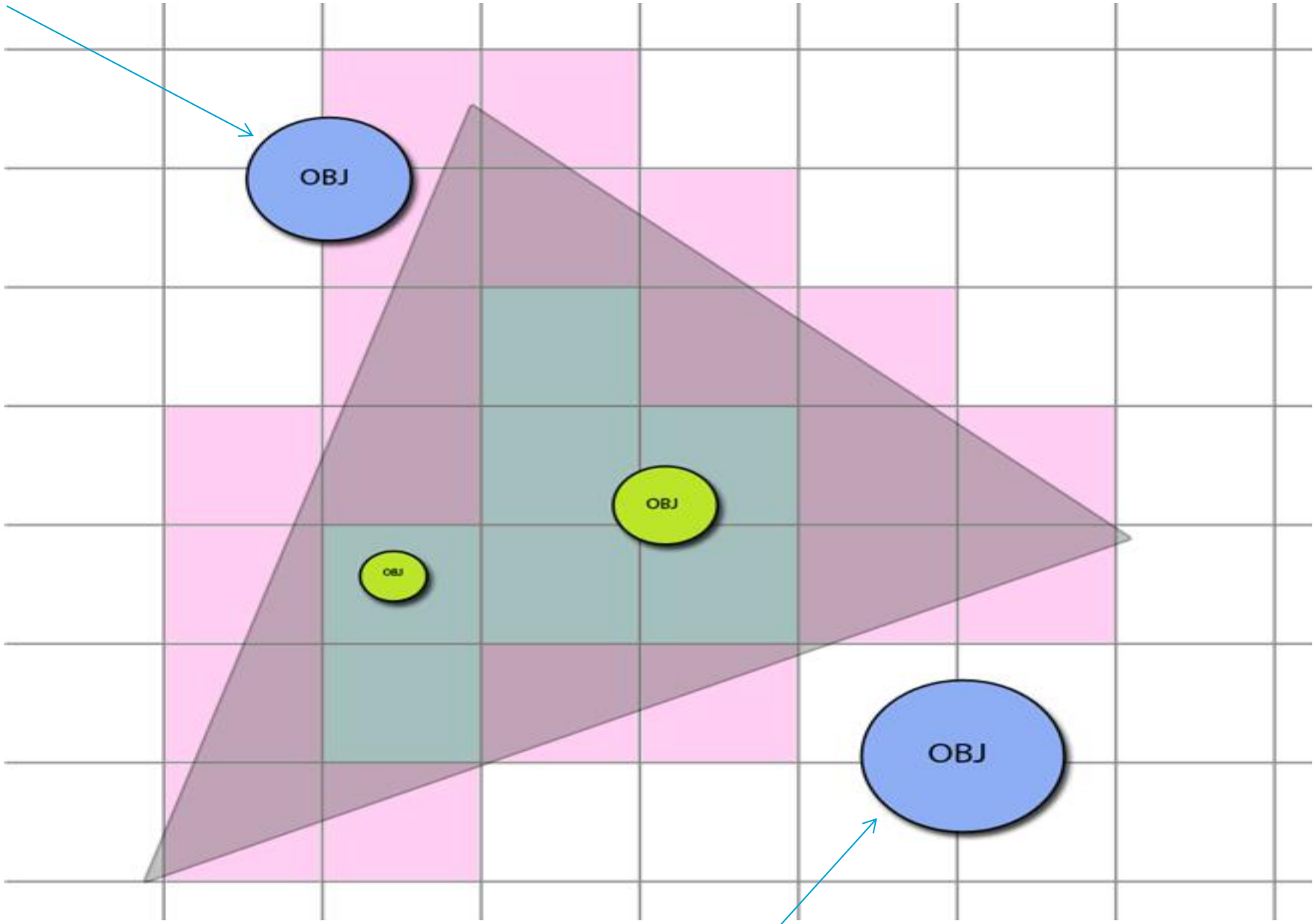
ARM

Objects will be drawn directly



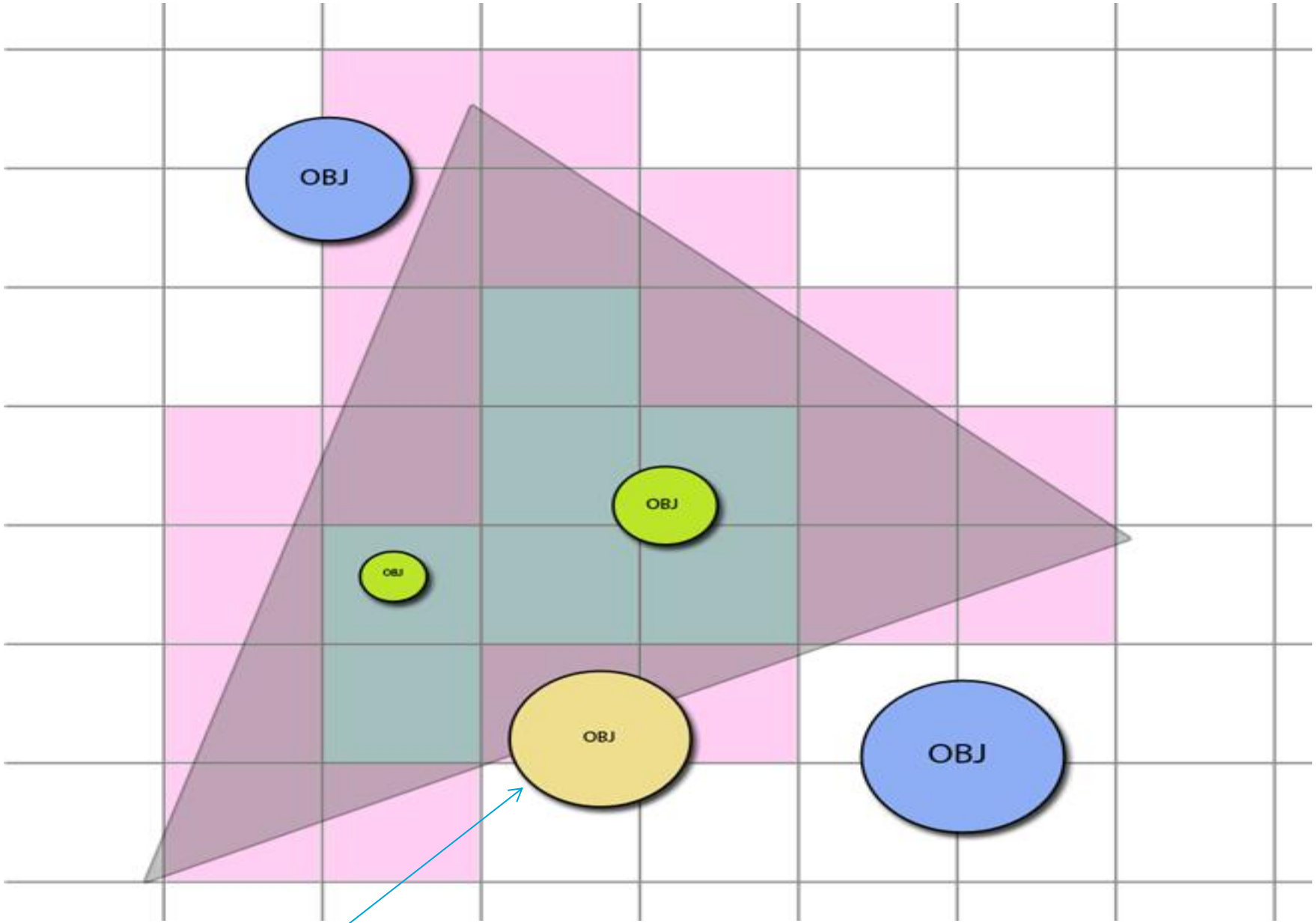
ARM

AABOX culling => culled



Outside frustum => directly culled

ARM



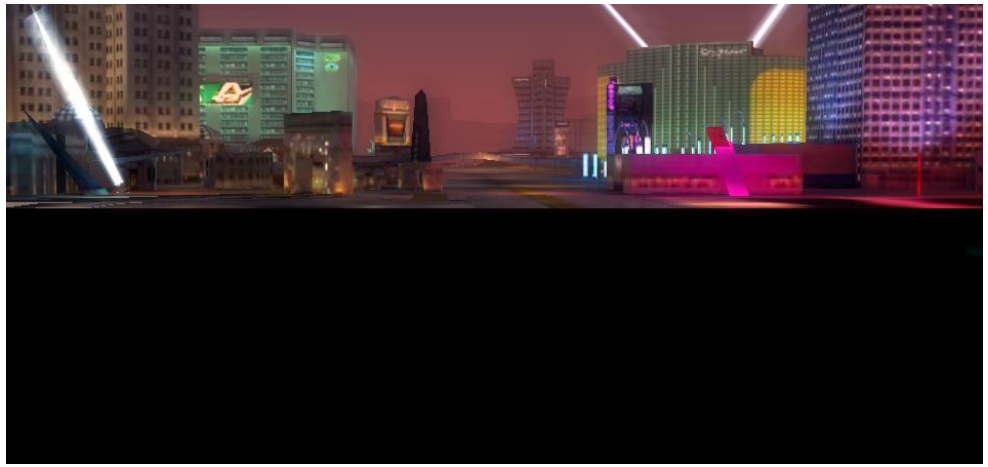
AABOX culling => visible

ARM

Near/Far Cities

- Two different city meshes are used:

1. Near city:
   - High resolution
   - Streamed
2. Far city
   - Low resolution
   - Always available



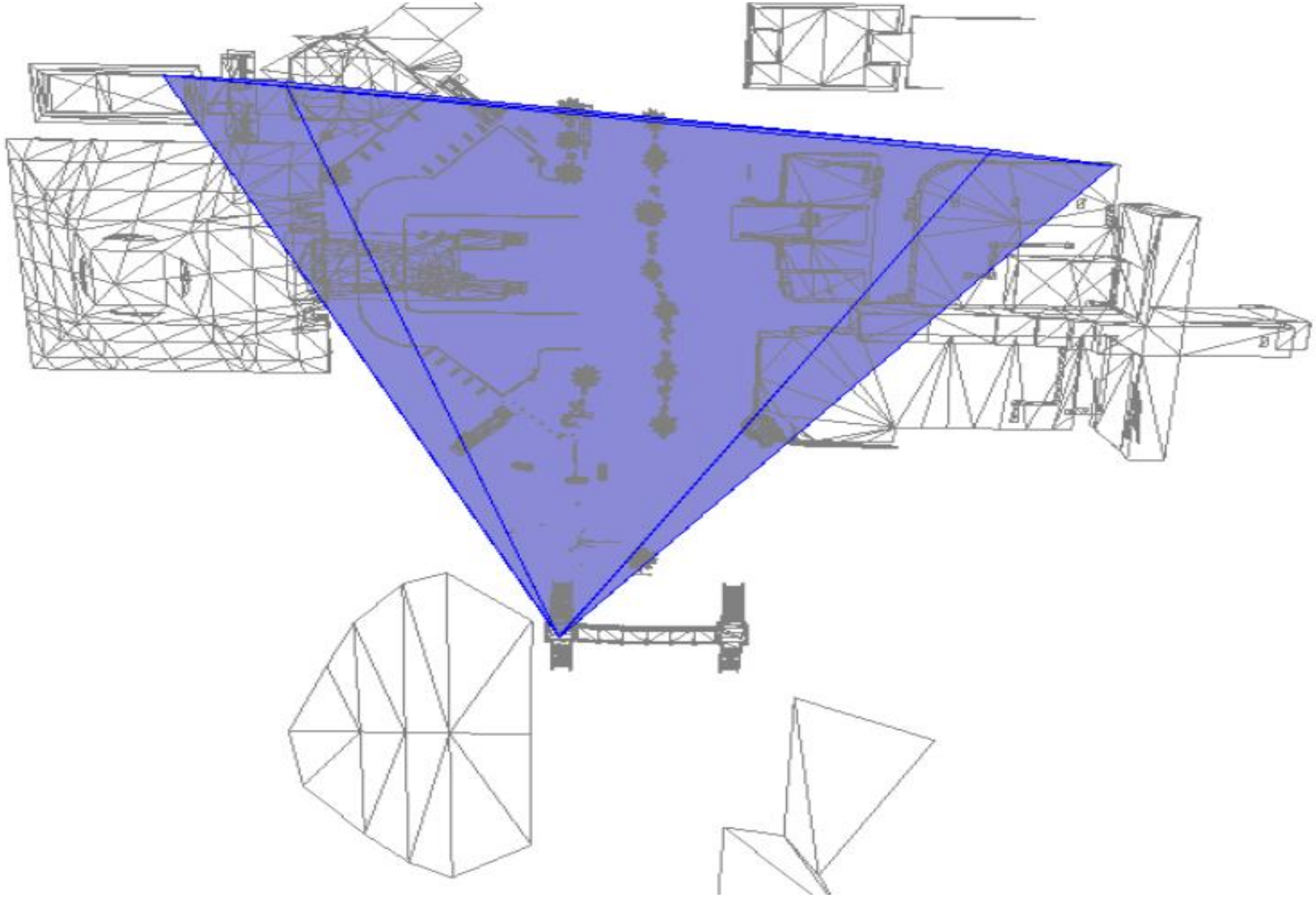
ARM

Full city: 77 draw calls

ARM

Near City : 48 draw calls

ARM

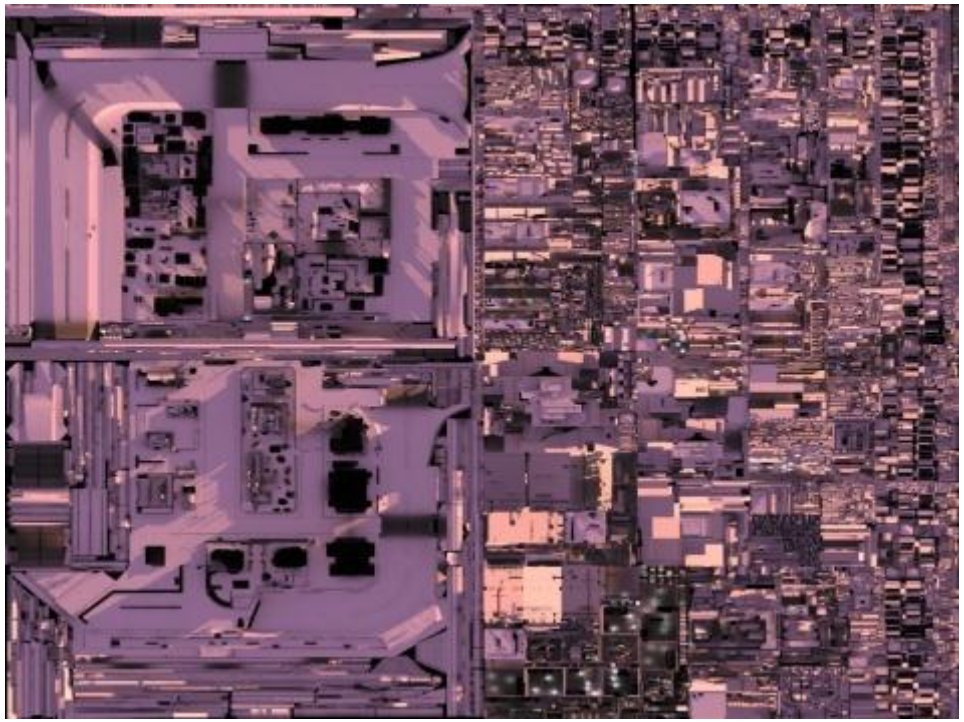


Top view, near city slice

ARM

Near City Materials

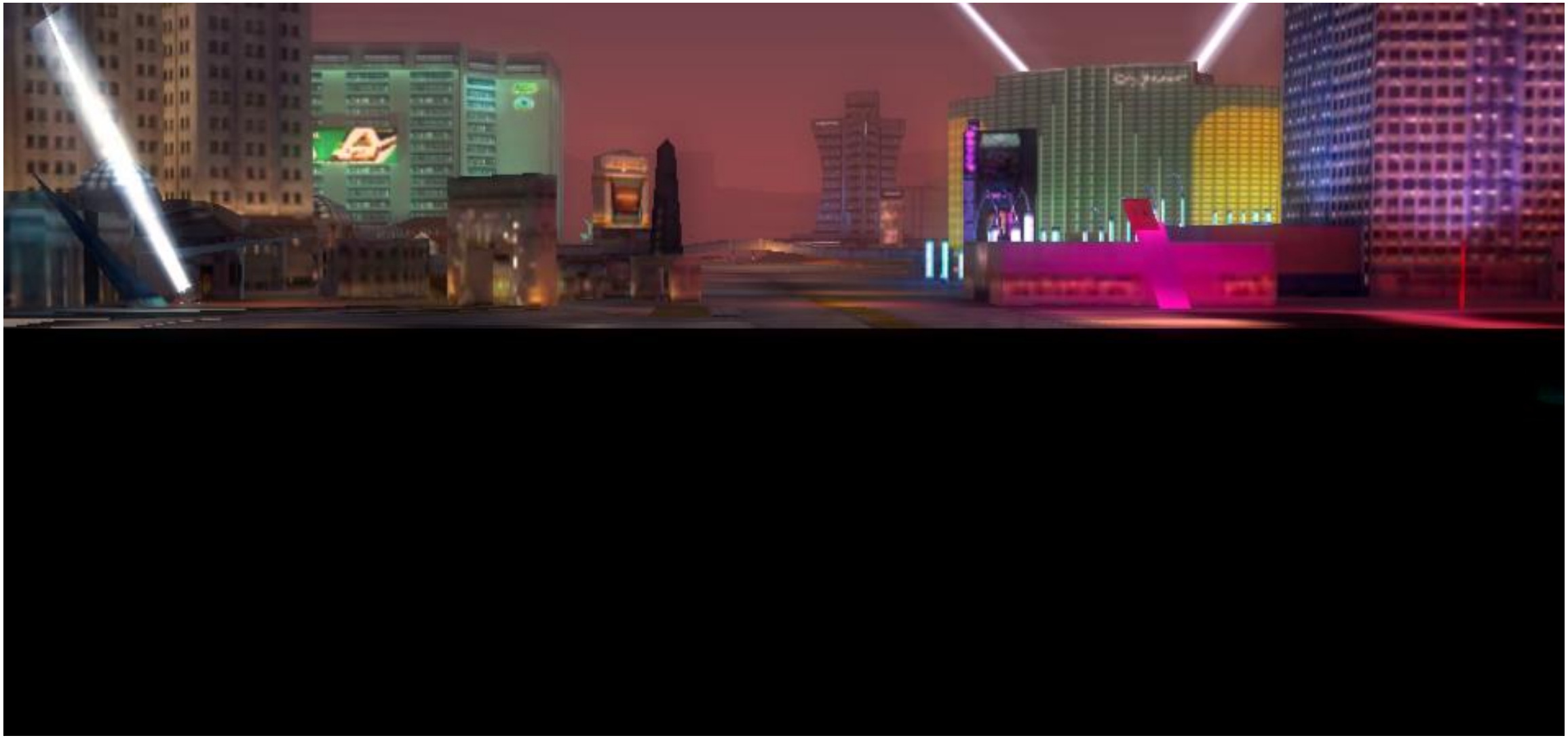
- Atlasing of albedo textures (batching & memory)
- Lightmaps
- Complex shaders (e.g. specular / reflection)



ARM

Full City: 77 draw calls

ARM



Far City: 29 draw calls

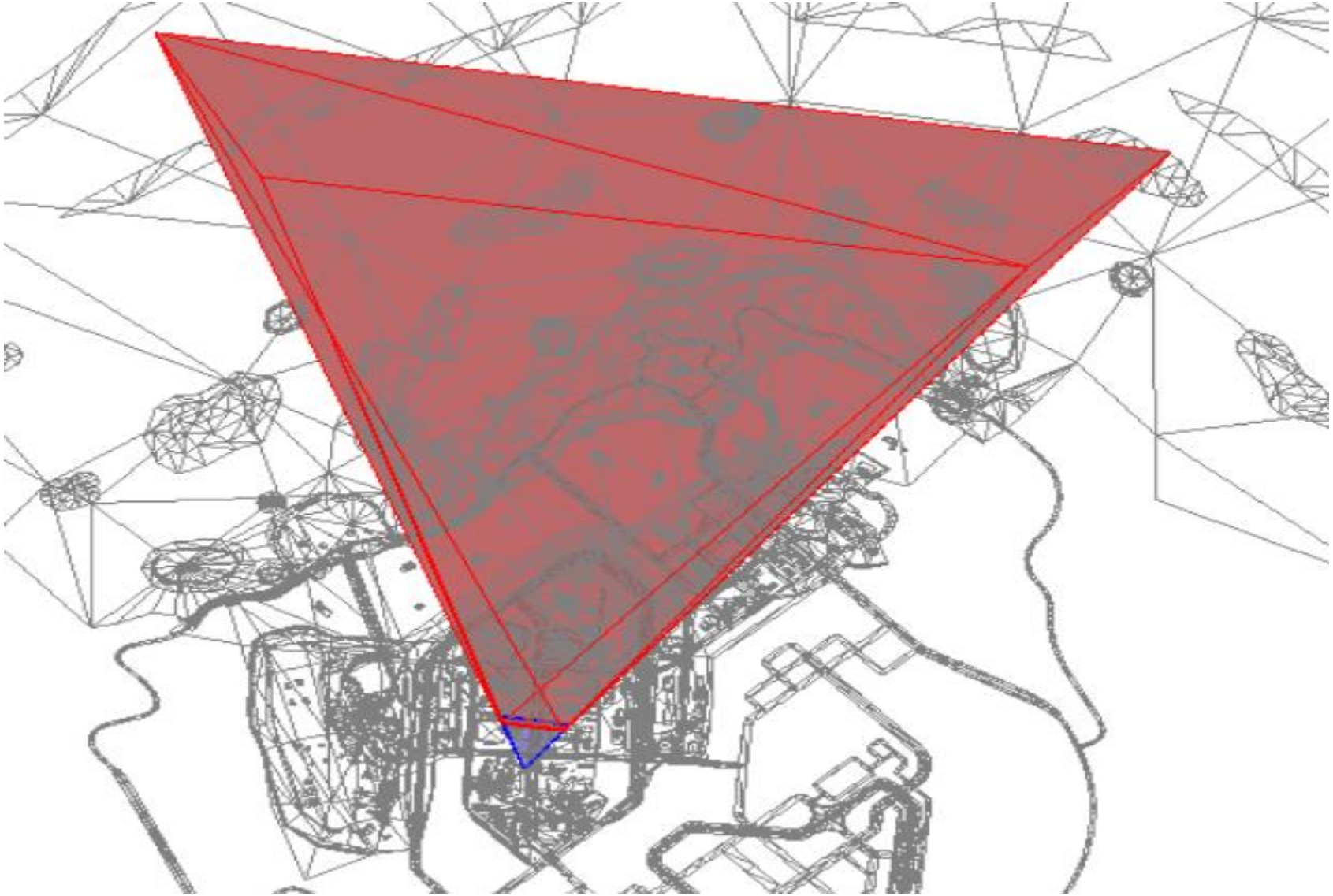
ARM

Cities Depth Slices

- Each city version has its own « depth slice »
  - Using **`glDepthRange`**
  - Little overlapping zone between the two (20m).

```
glDepthRange(0.0, 0.08);
CamZnear = 0.2;
CamZFar  = 160;
RenderHighresCity();

glDepthRange(0.08, 1);
CamZnear = 140;
CamZFar  = 2000;
RenderLowresCity();
```

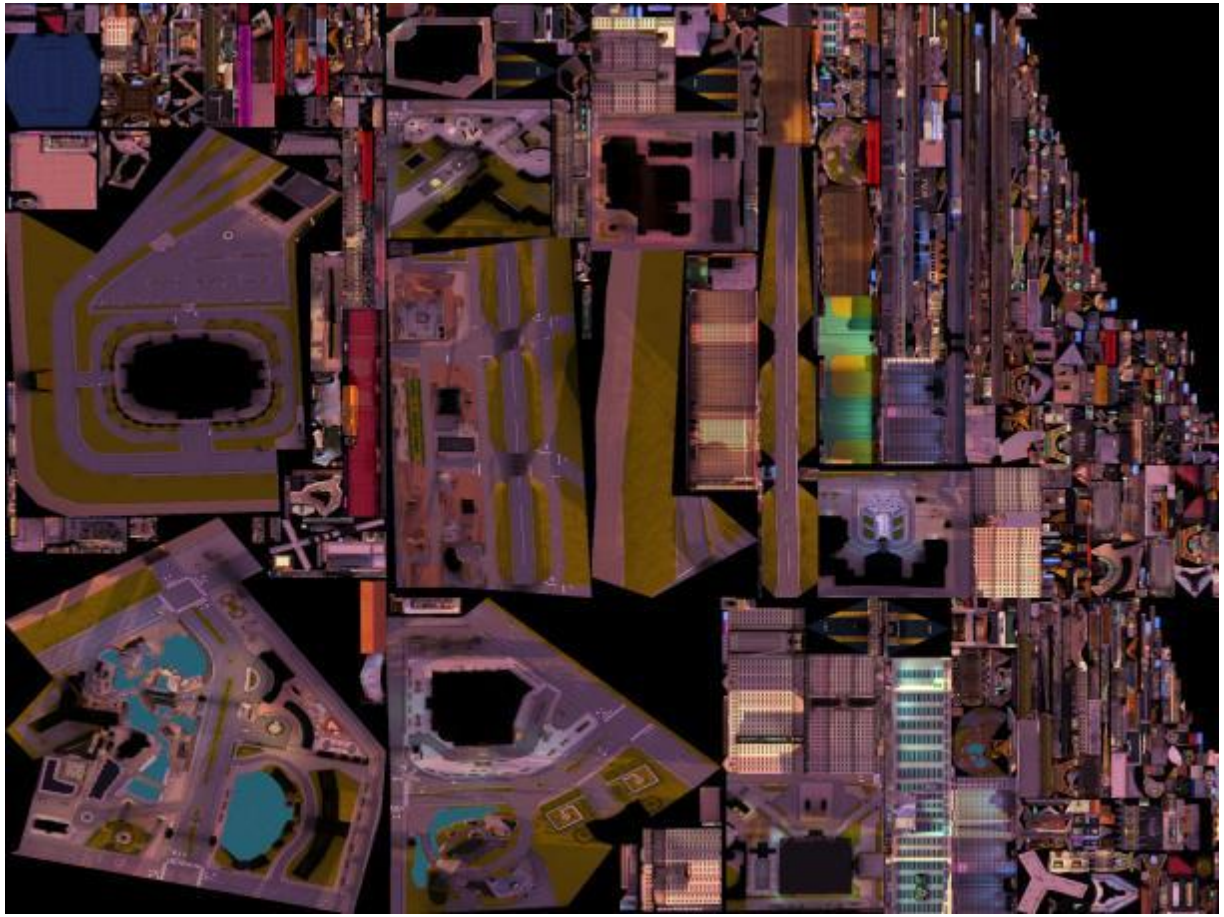


Top view: far city and full frustum

ARM

Far City Materials

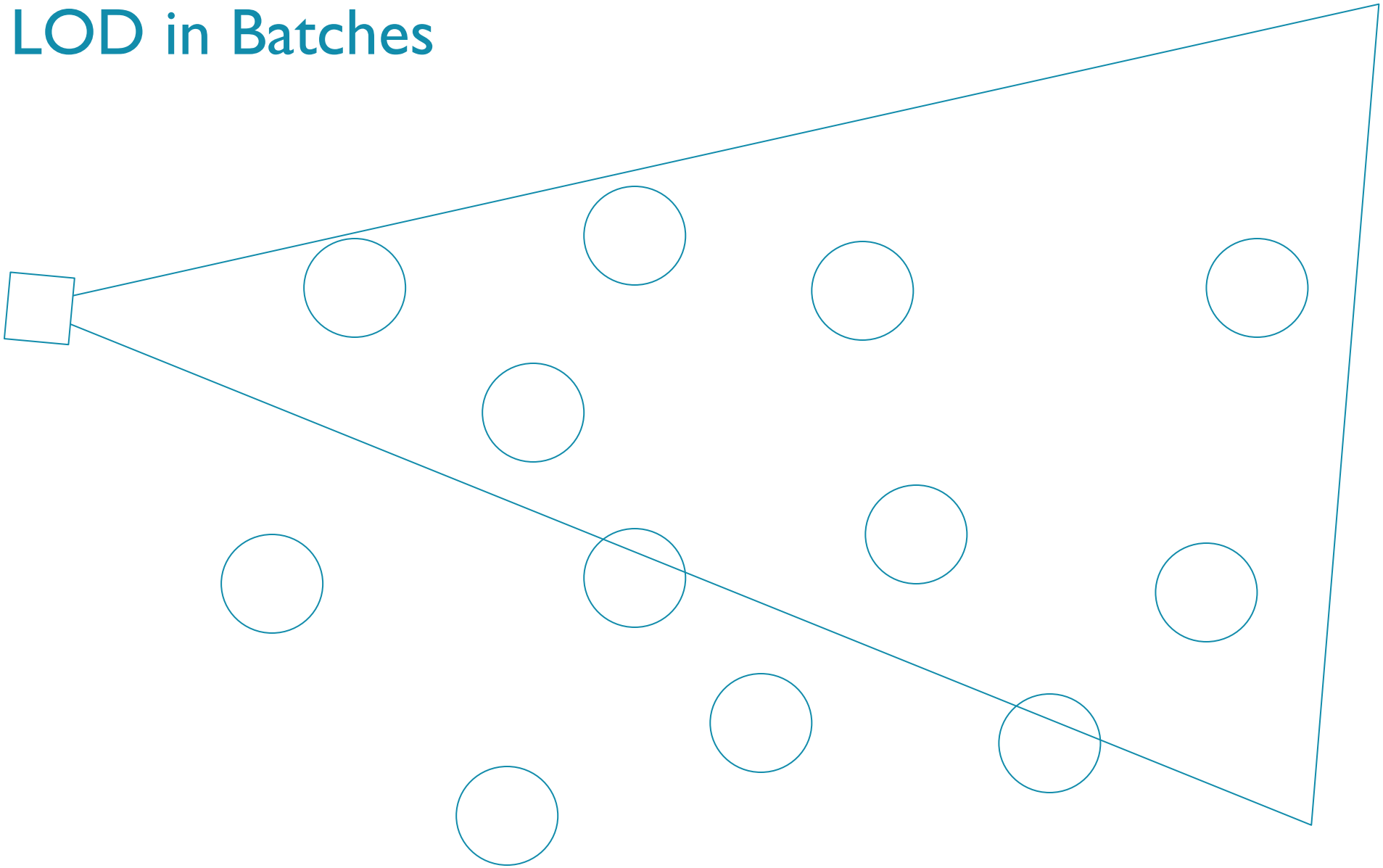
- Heavy atlasing
- Premultiplied Lightmaps
- Low cost shaders



LODs and Culling

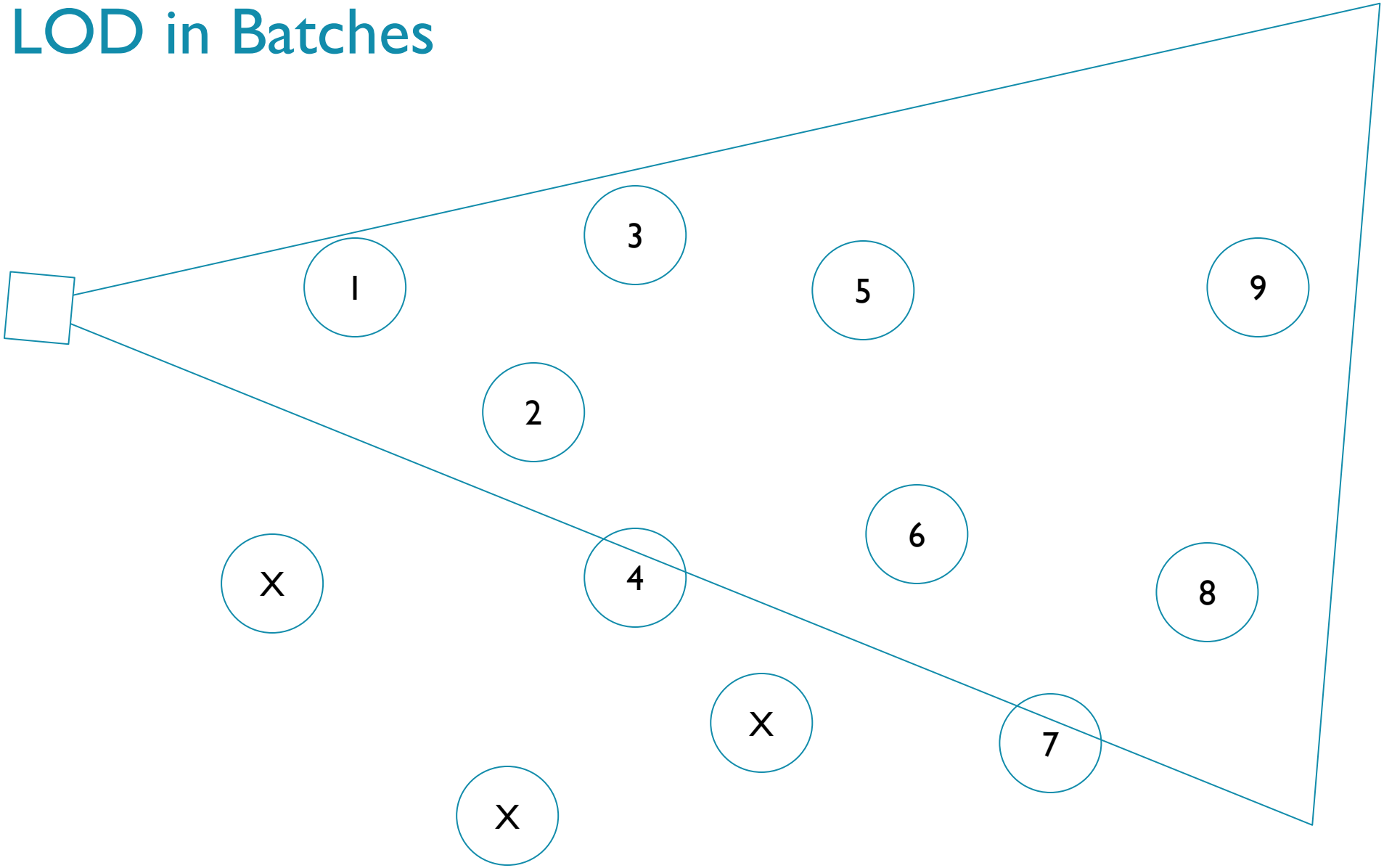
ARM

LOD in Batches



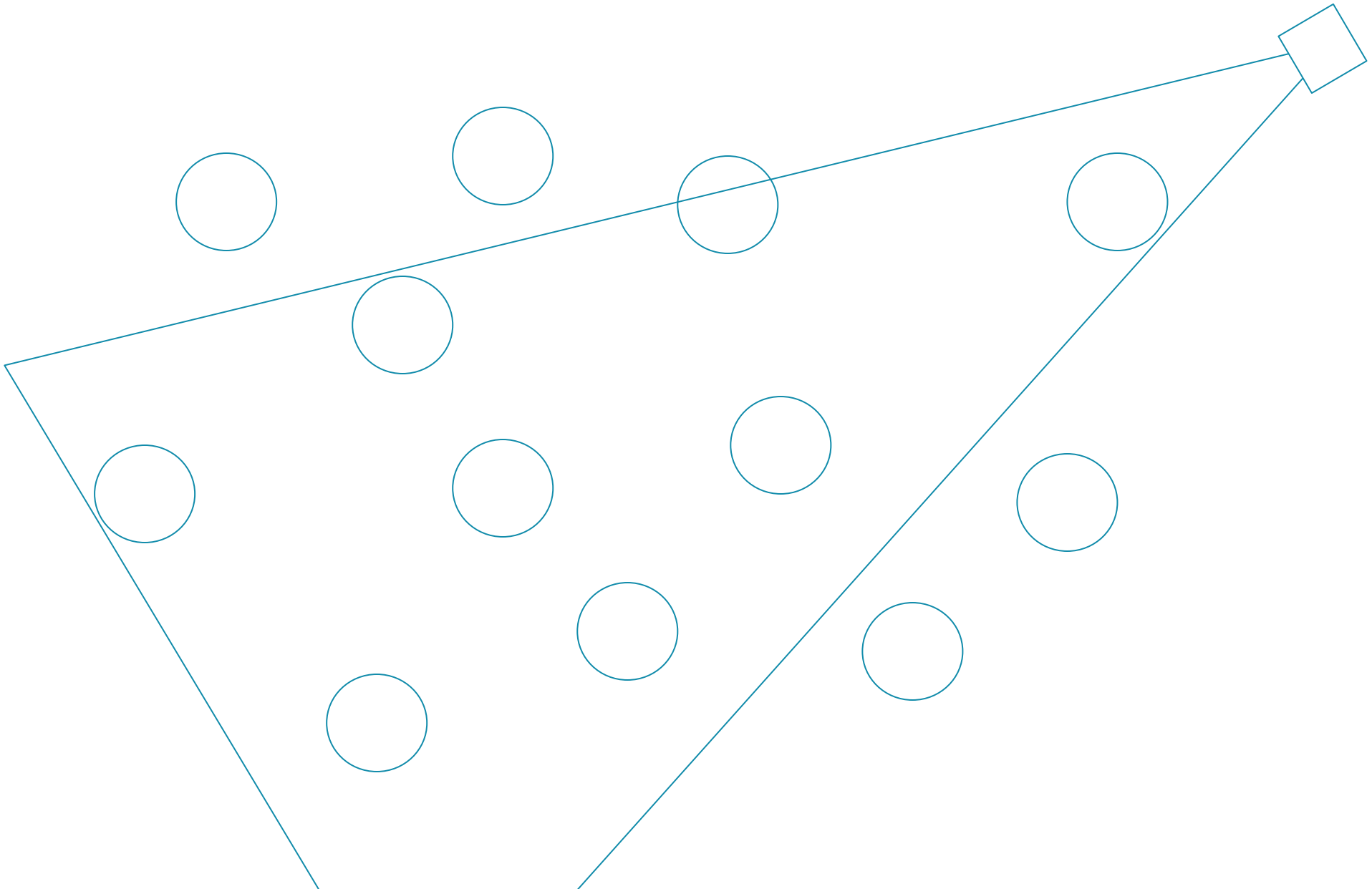
ARM

LOD in Batches



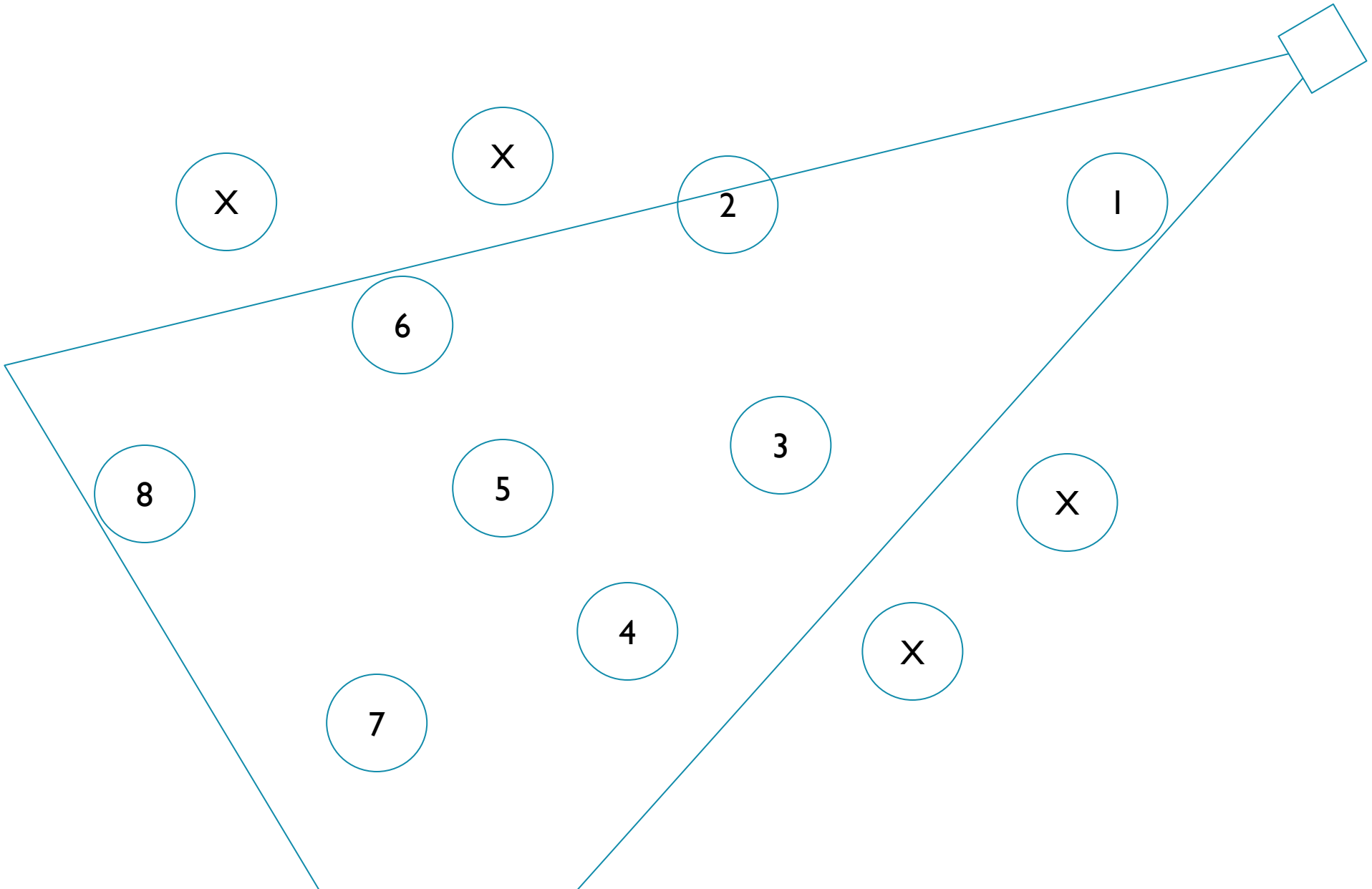
ARM

LOD in Batches



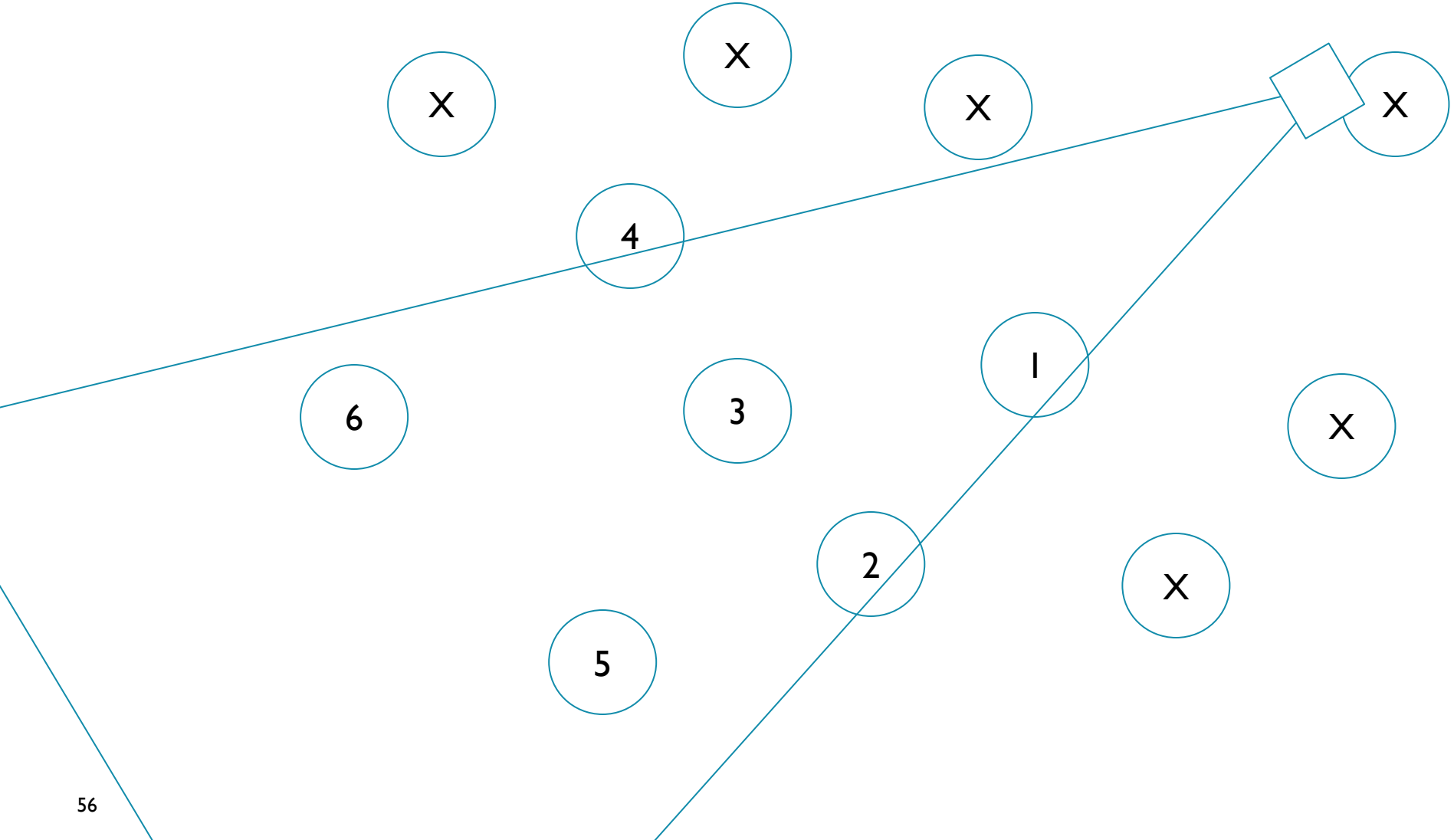
ARM

LOD in Batches



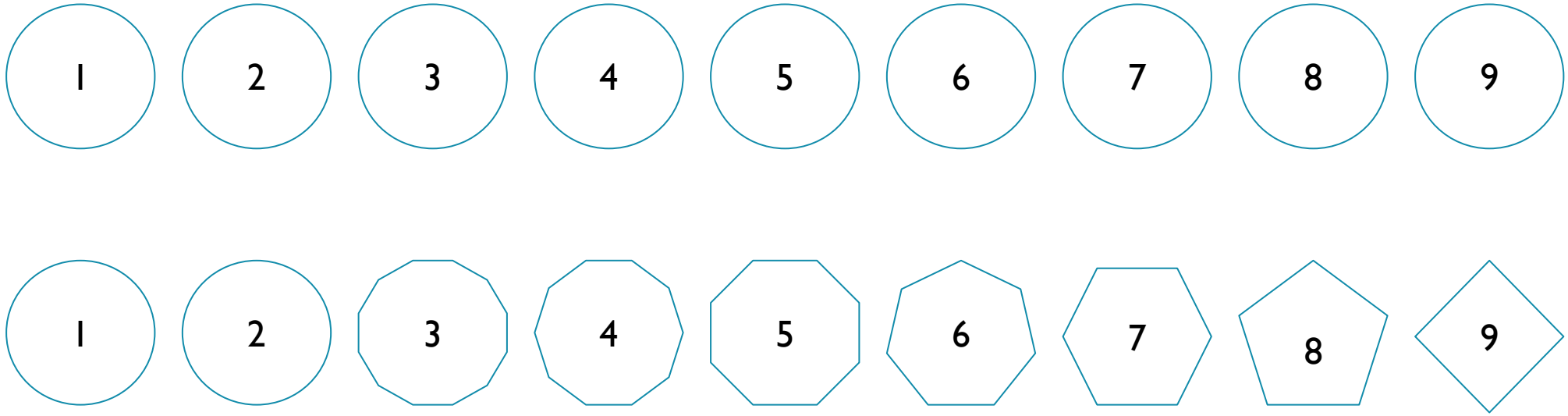
ARM

LOD in Batches



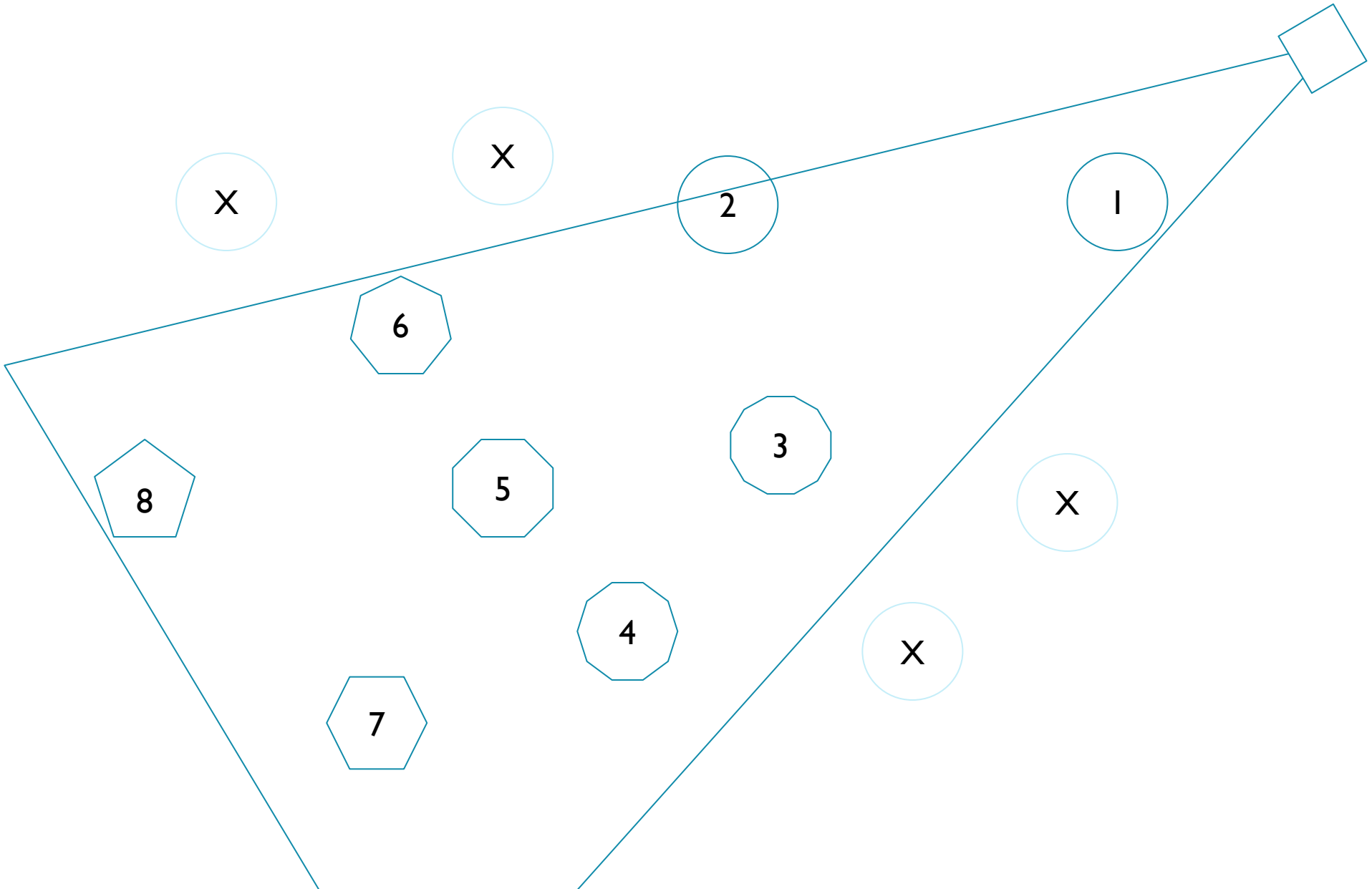
ARM

LOD in Batches



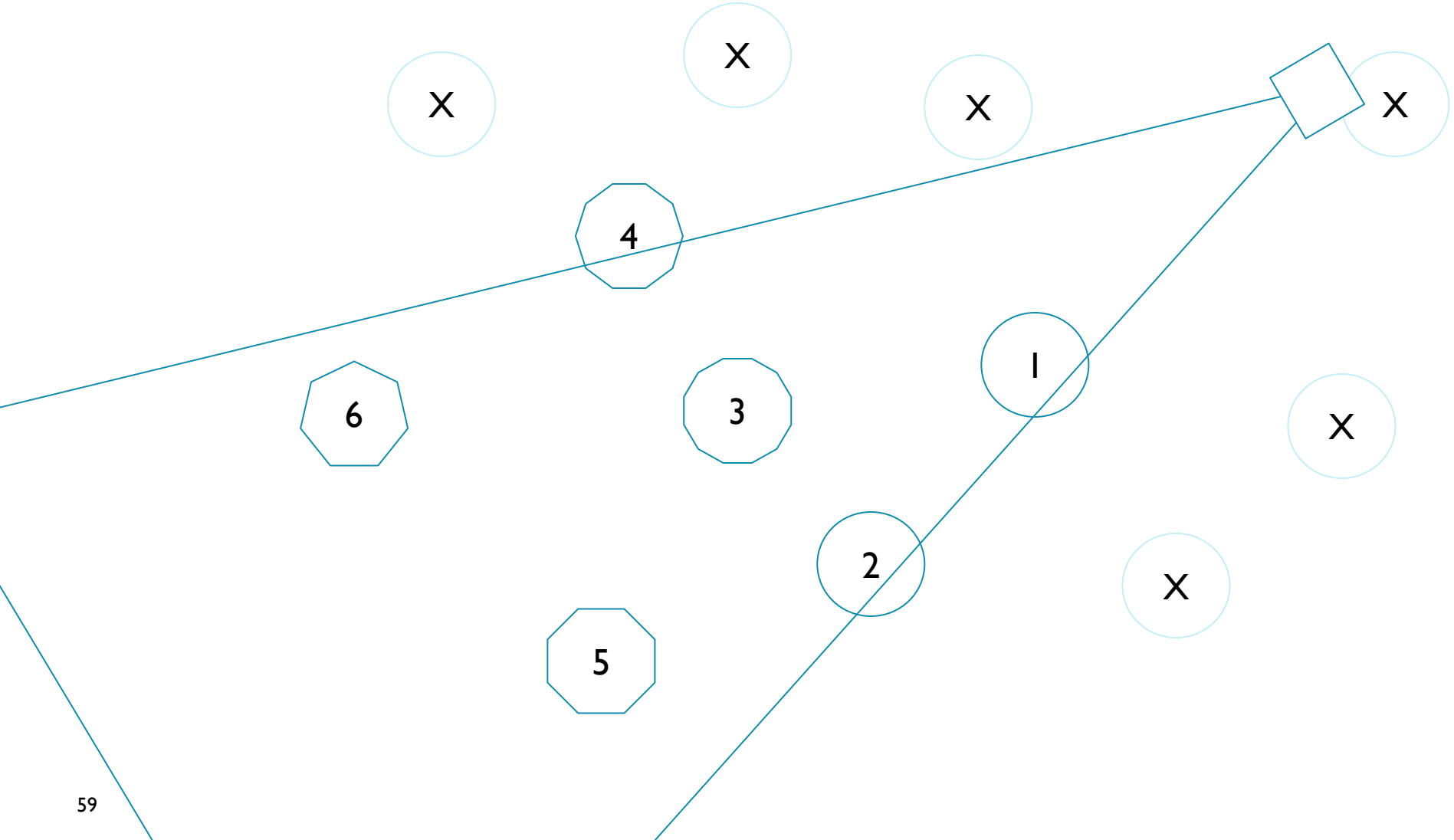
ARM

LOD in Batches



ARM

LOD in Batches



ARM

A View to a Cull

Scene Batching

Timbuktu has 22 objects per environment layer

S= Skipped R=Rendered X=Scaled out

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
		R	R	R							R	R						R		R	

ARM

Imposterable

- Timbuktu's Foliage:
  - 4 types of alpha blended geometries, depth sorted
  - Unpredictable interleaving of Grass, Tree, Bush and Shrub
- First suggestion:

  GTBSGTSBGTSBGTSBGTBSGTBSGTSBGTSBGTS
  - No more than 3 dead models in a row
  - Clumping of similar entities makes worst case more common

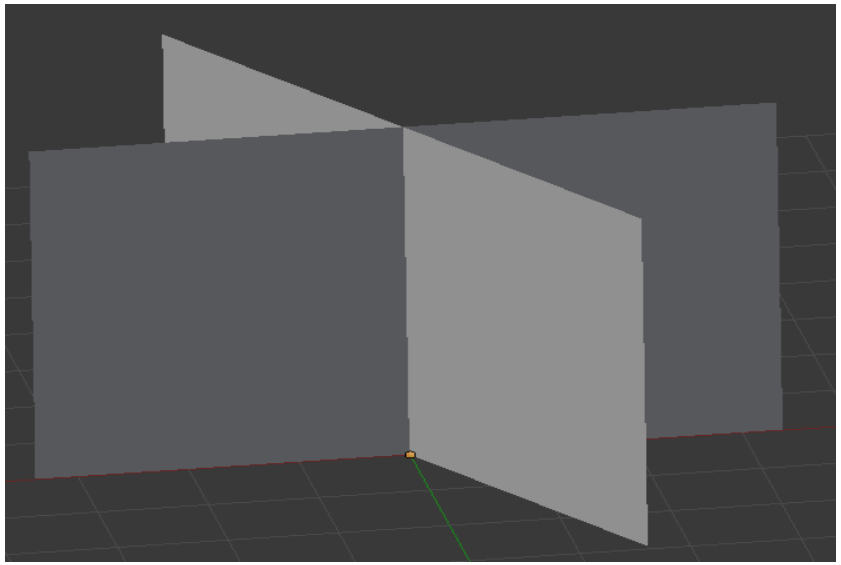
- Second suggestion:

  GGTTBBSSGGTTBBSSGGTTBBSSGGTTBBSS
  - Allows consecutive pairs more cheaply
  - Worst case still arises frequently, and is now even worse

ARM

Imposterable

- Solution:
  - Foliage only varies in texture and scale
  - All models topologically identical
  - Zero skips
  - Extra vec4 per instance



ARM

Asphalt 8

Effects, Draw Calls & Texture Data

ARM

The Game

- Explosive action racing
  - Multiplayer
  - Physics engine
- Main FXs :
  - Realtime soft shadows
  - Realtime reflections
  - Paraboloid
  - Road reflections
  - Proxies
- Main post FXs:
  - Motion blur
  - Lens flare dirt
  - Color grading
  - Vigneting

ARM

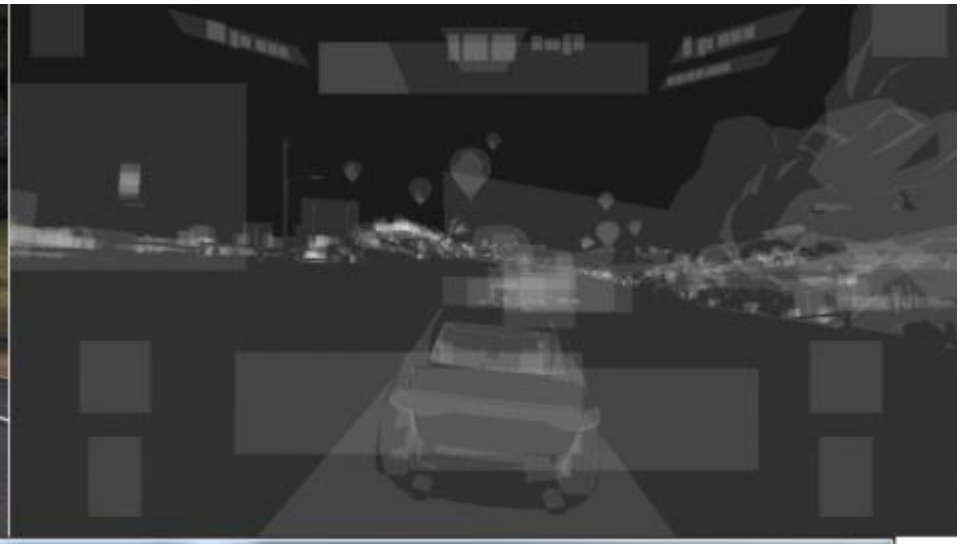
Game Effects

ARM

Optimizations / Draw Call Reduction

- Culling adapted to a racing game :
  - PVS (Potential Visibility Sets)
  - AABBox culling
  - Software occlusion culling
  - LODs (distance or screen projection size)
  - VBOs / IBOs for 95% of mesh data
- Physics on a separate thread
- Texture streaming to reduce loading times
- On a medium device (Samsung Galaxy SIII):
  - 120 to 250 draw calls, depending on track

ARM® Mali™ Graphics Debugger (on a Samsung Galaxy SIII)



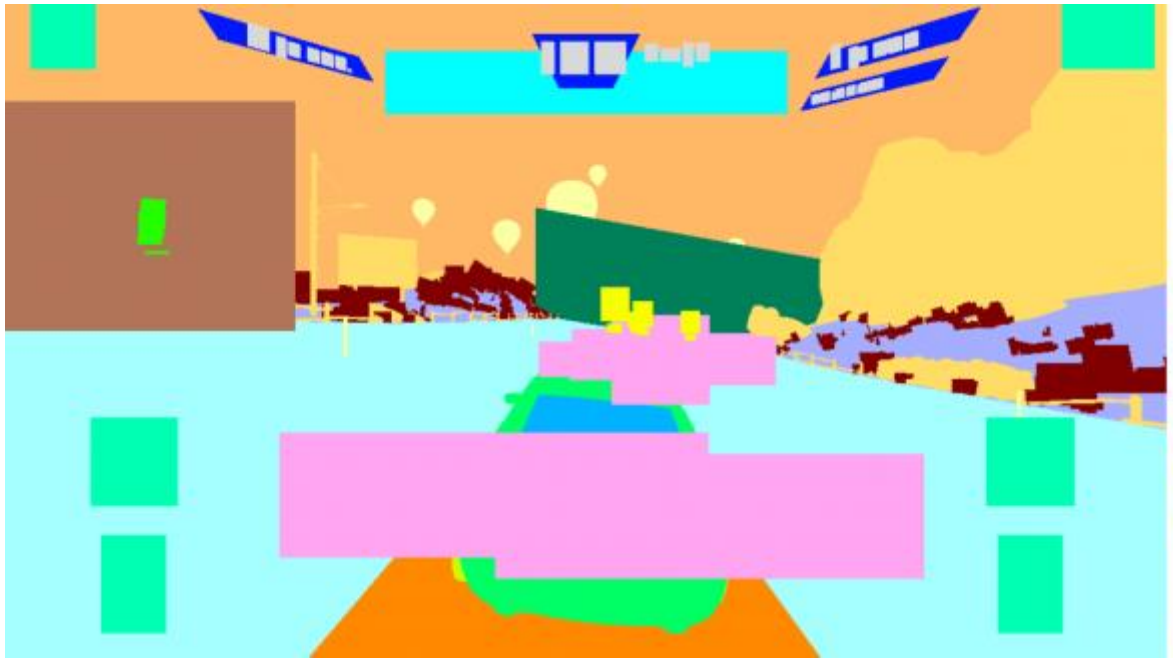
Statistics

Total number of API function calls	624902
Average vert/frame	323250,00
Average instanced vert/frame	0,00
Average draw/frame	138,00
Frame #	419
Number of API calls in current frame	2757
Number of draw calls in current frame	139
Number of vertices submitted in current frame	323658
API call #	619473
Number of vertices in current draw call	n/a

- 300k vertices
- 140 draw calls
- Not much overdraw
  - Stop lights
  - HUD

ARM

ARM® Mali™ Graphics Debugger (on a Samsung Galaxy SIII)



Fragment Shaders

Program	Name	Instructions	Shortest path	Longes...	Instances run	Total cycles
129	Shader 128	14	14	14	272330	3812620
60	Shader 59	7	7	7	145816	1020712
108	Shader 107	18	18	18	49211	885798
159	Shader 158	1	1	1	665856	665856
9	Shader 8	6	6	6	99483	596898
177	Shader 176	9	7	9	58480	467840
105	Shader 104	17	17	17	23389	397613
132	Shader 131	9	9	9	30708	276372
36	Shader 35	2	2	2	136553	273106
138	Shader 137	5	5	5	45264	226320
174	Shader 173	3	2	2	68854	137708
147	Shader 146	4	4	4	24442	97768
210	Shader 209	3	3	3	27593	82779
21	Shader 20	5	5	5	16384	81920
123	Shader 122	9	9	9	7617	68553
102	Shader 101	2	2	2	28038	56076
213	Shader 212	3	3	3	12493	37479
198	Shader 197	3	3	3	11470	34410
111	Shader 110	4	4	4	7311	29244
207	Shader 206	10	8	10	2896	26064
24	Shader 23	5	5	5	4096	20480
114	Shader 113	3	3	3	4542	13626
204	Shader 203	3	3	3	2466	7398
135	Shader 134	11	11	11	210	2310
192	Shader 191	2	2	2	851	1702
153	Shader 152	4	4	4	324	1296
117	Shader 116	1	1	1	537	537
195	Shader 194	4	4	4	76	304

- Post FXs disabled here
- Shaders covering large screen areas must be cheap :
  - `Road => 14 cycles (specular / normal map)`
  - `Rocks => 7 cycles`
  - `Sky => 1 cycle`
  - `Cars => 18 cycles`
  - `Lights/Smoke => 2/4 cycles`

ARM

ETC1 and Alpha

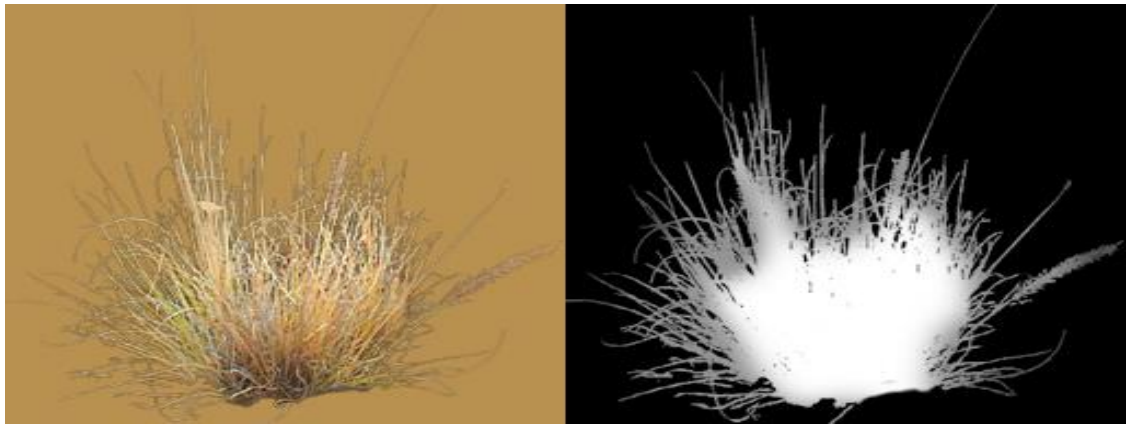
- ~80% of the textures are ETC1 compressed
  - We keep uncompressed :
    - Splash / loading screens
    - Cars & tracks illustrations
    - Lens flare textures, LUTs…
- ~20% of the textures need alpha:
  - Menus, HUD
  - Billboards
  - Specular maps …

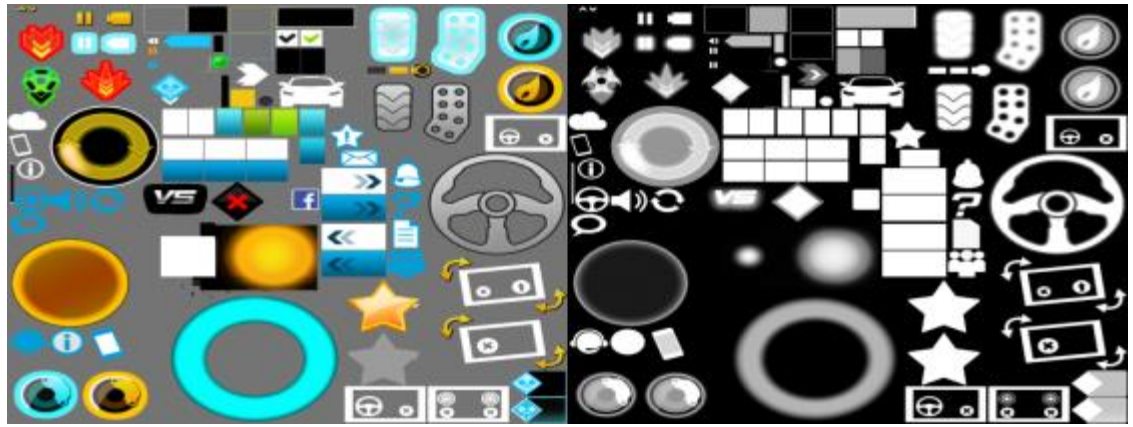
Split Alpha on Android™

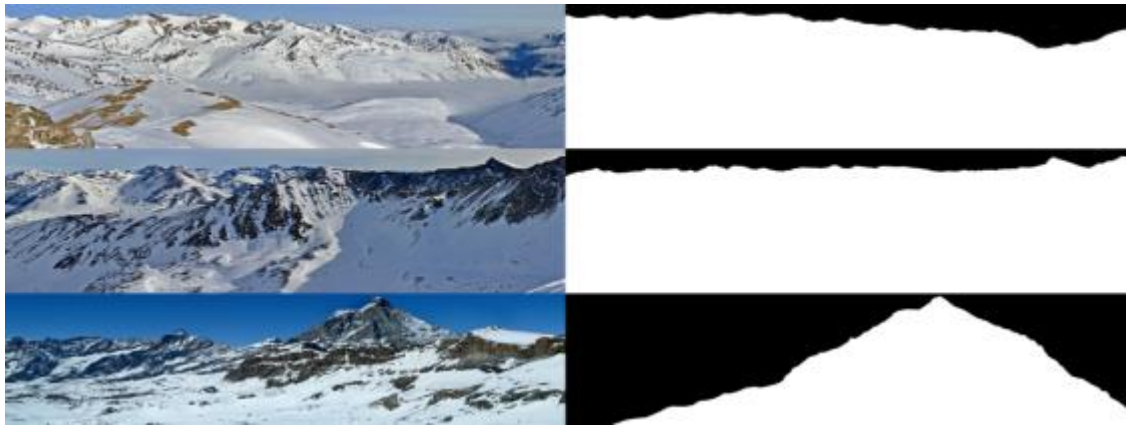
- Using two ETC1 compressed textures
  - One for Alpha
  - One for RGB
- ETC1 compression ratio: 4bpp (24bits)
  - RGB: 6x
  - RGBA: 3x

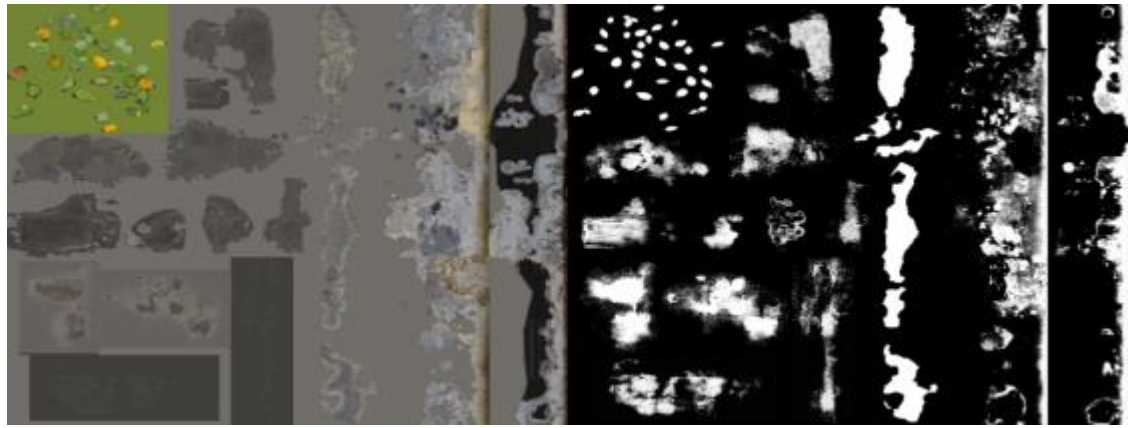
ARM

Billboards / HUD



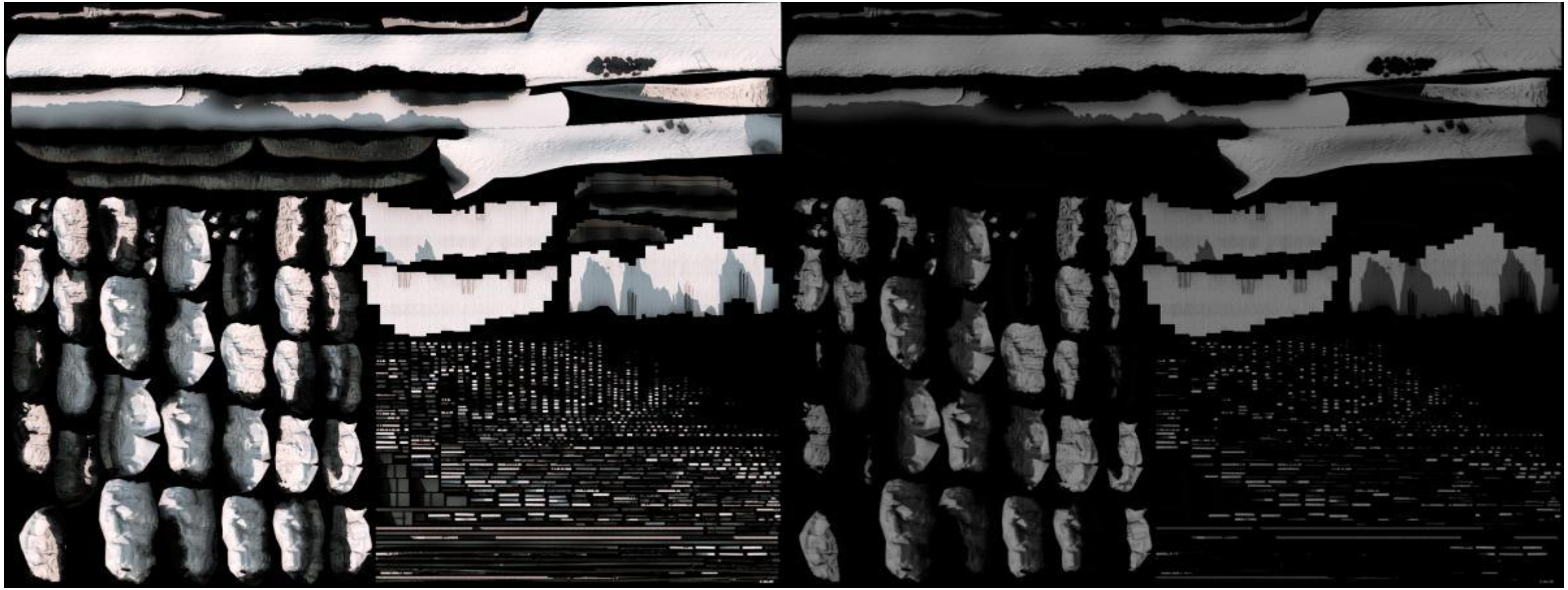






Additive particles / lens flare textures do not need alpha !

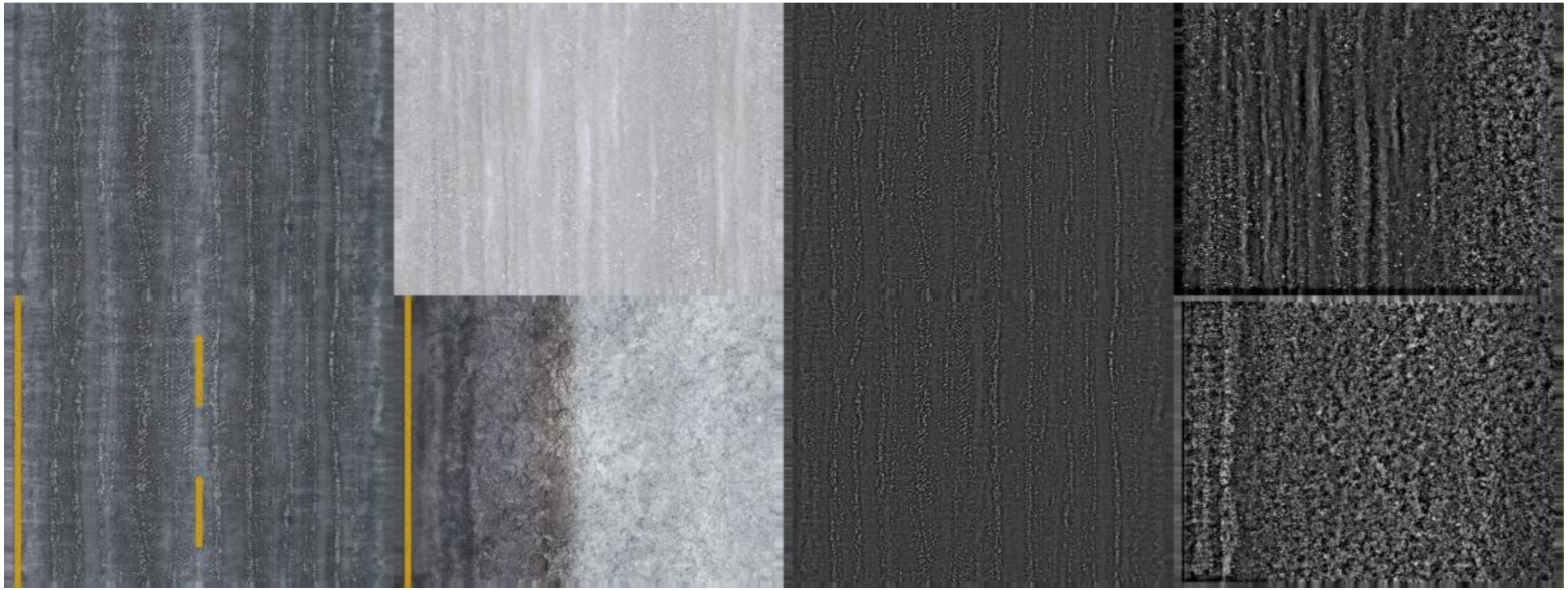
Light Maps



```
Complete = DF * (LM + (LM * LM.a * brightBurn)) * overbright;
```


ARM

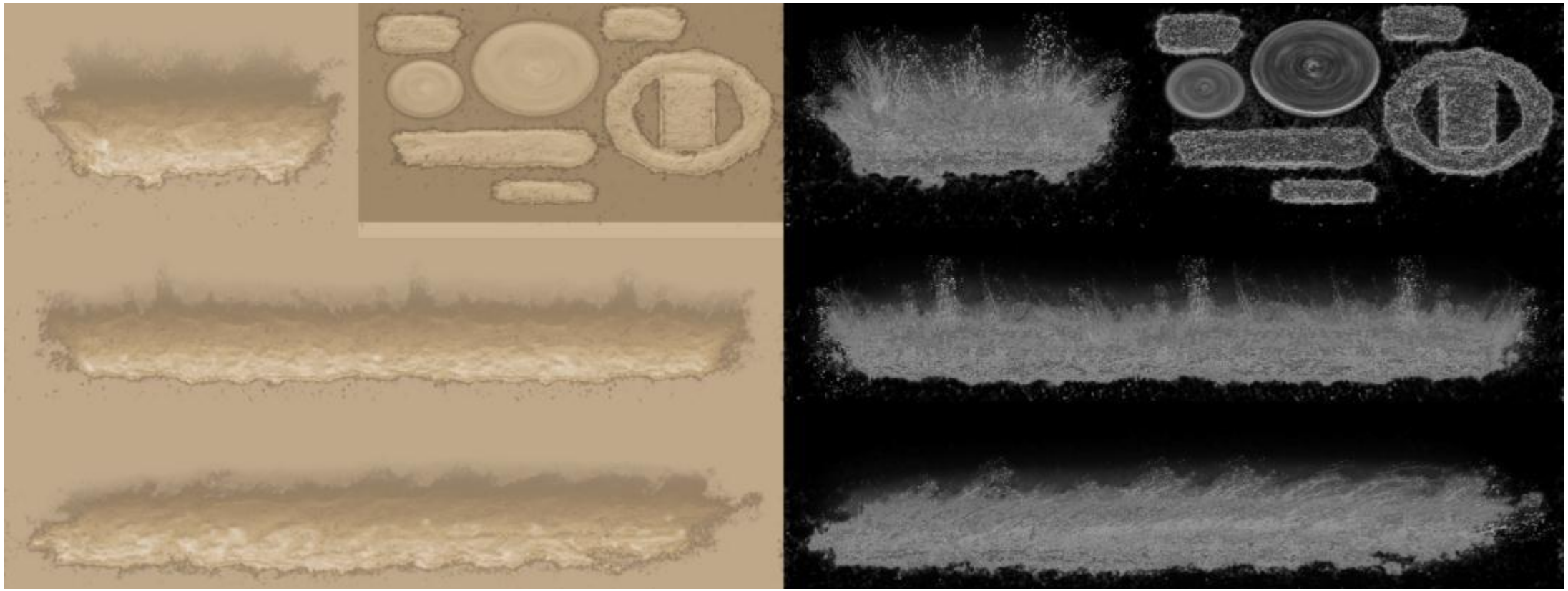
Road Specular Map



Alpha = AO + Specular Intensity

ARM

Dirt Map



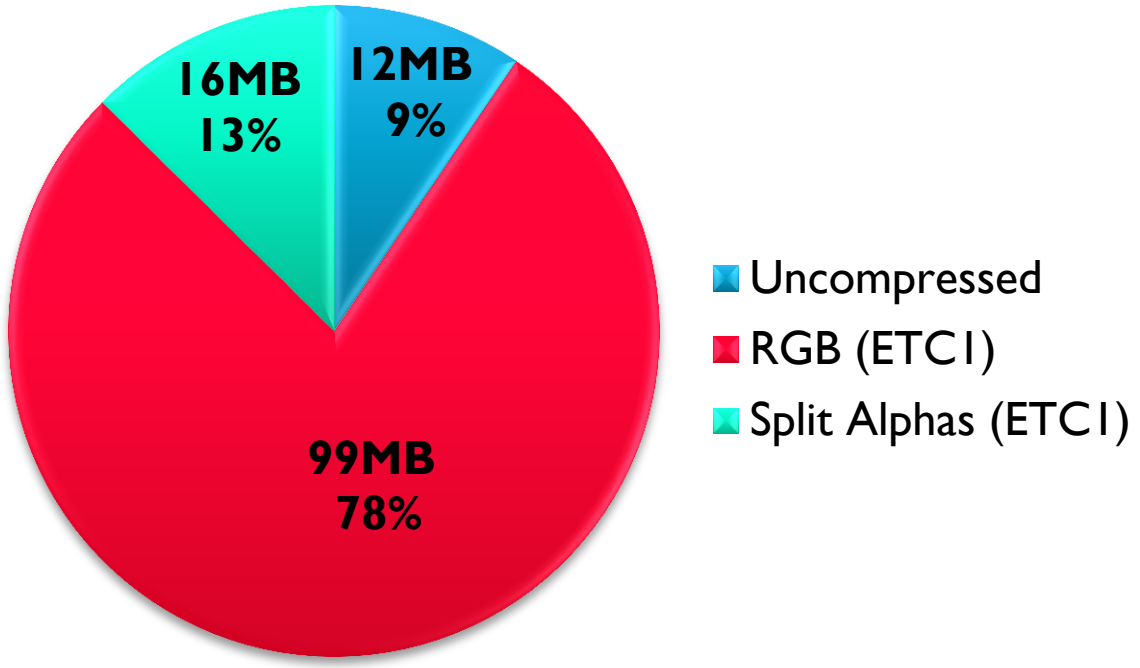
```
vec3 dirt = aoFactor * light * DIRT.rgb;
float dirtFactor = min(dirtAmount * DIRT.a, 1.0);
color = mix(color, dirt, dirtFactor);
```

Textures in Iceland Track

ARM

Iceland Track: Memory Usage

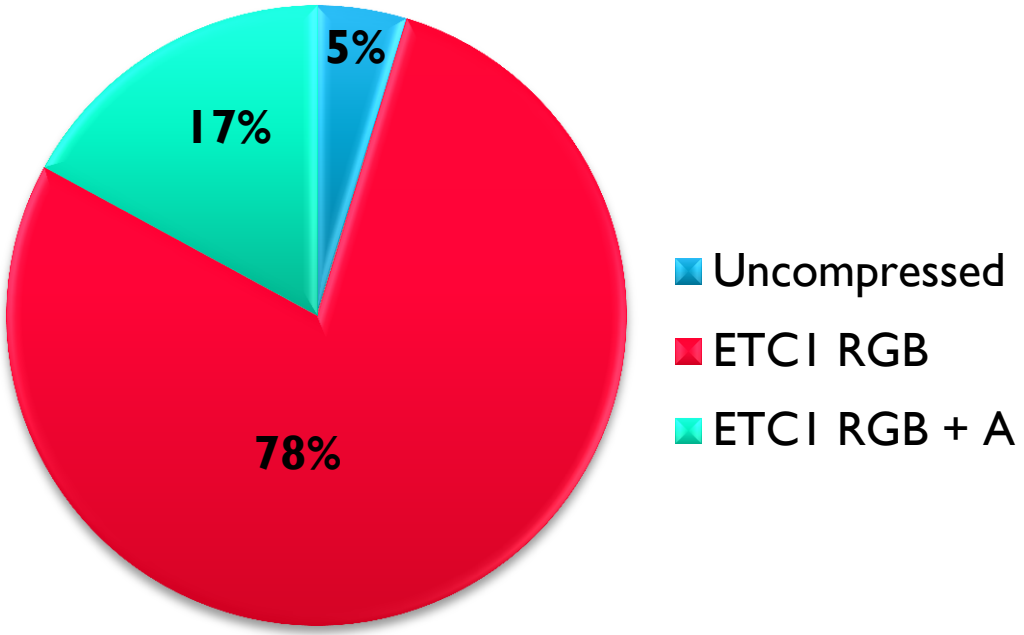
- Total : 127 MB
  - mipmaps not counted
- 13% of texture data is used for alphas (16MB)



ARM

Iceland Track: Texture Repartition

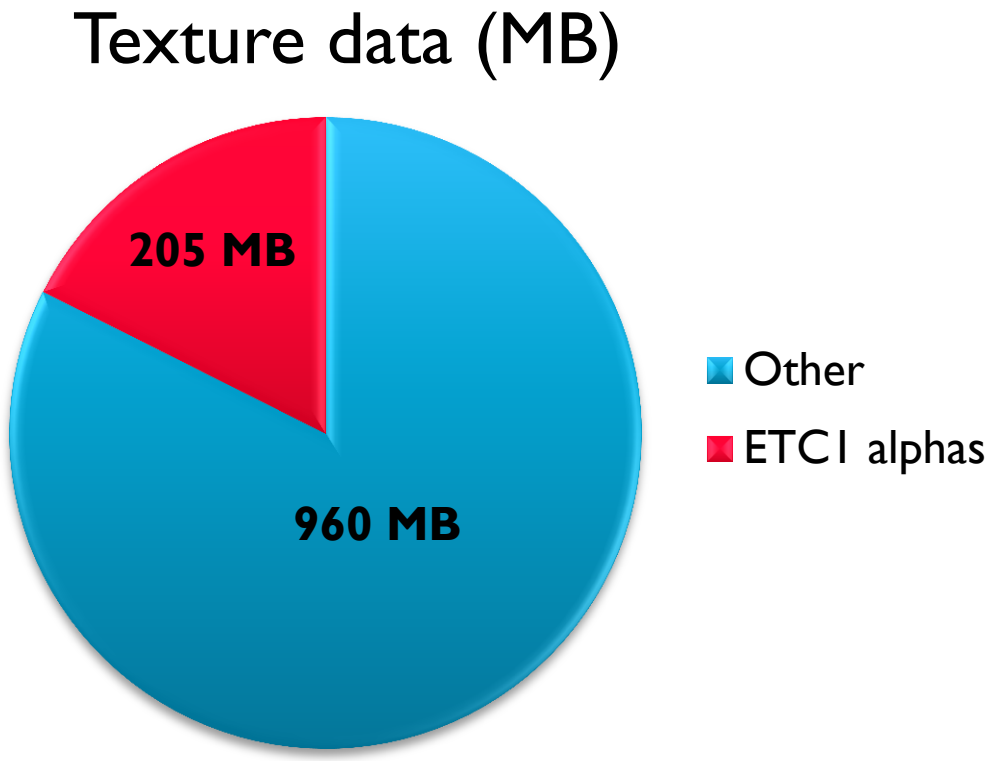
- Things to take care of with ETC1 split alpha technique :
  - More bandwidth usage
  - One extra sampling operation in shaders:
    - More fragment instructions
    - Still better cache efficiency than uncompressed, but not as good as ETC1 RGB



ARM

Total Texture Size on Disk

- Total texture size in game package: 1165 MB
- 17.5% of package (205MB) is ETC1 alphas data



ARM

Texture Compression

ARM

Texture Compression Formats

- ETC – ARM® Mali™-400 GPU
  - 4bpp
  - RGB No alpha channel
- ETC2 – ARM Mali-T604 GPU
  - 4bpp
  - Backward compatible
  - RGB also handles alpha & punch through
- ASTC – ARM Mali-T624 GPU and beyond
  - 0.8bpp to 8bpp
  - Supports RGB, RGB alpha, luminance, luminance alpha, normal maps
  - Also supports HDR
  - Also supports 3D textures

ARM

ETC

- Khronos standard
- Most widely supported
- Doesn't support alpha channels
- If you need alpha, the ARM® Mali™ Texture Compression Tool has options to store a separate alpha image

ARM

ETC 2

- Extension of ETC
- Less widely supported
- More block modes
- Alpha / punch through
- Punch through tip:
  - Use `discard` and sort front to back

ARM

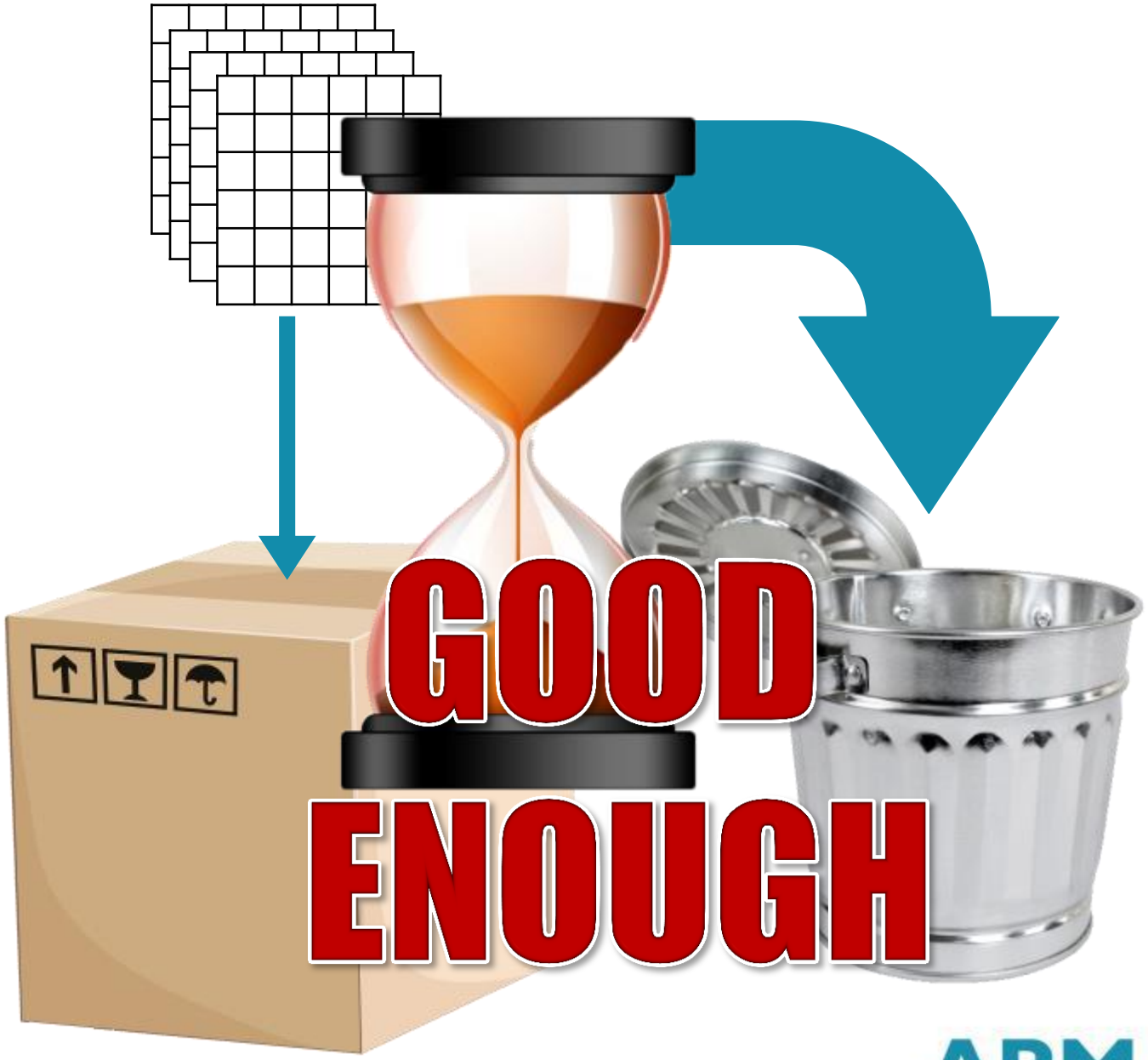
ASTC

- Developed by ARM, now a Khronos Standard
- Variable bit rate 8bpp to 0.8bpp
- ASTC is on the cutting edge so only the most up to date hardware supports it
- This time next year ASTC will be everywhere

ARM

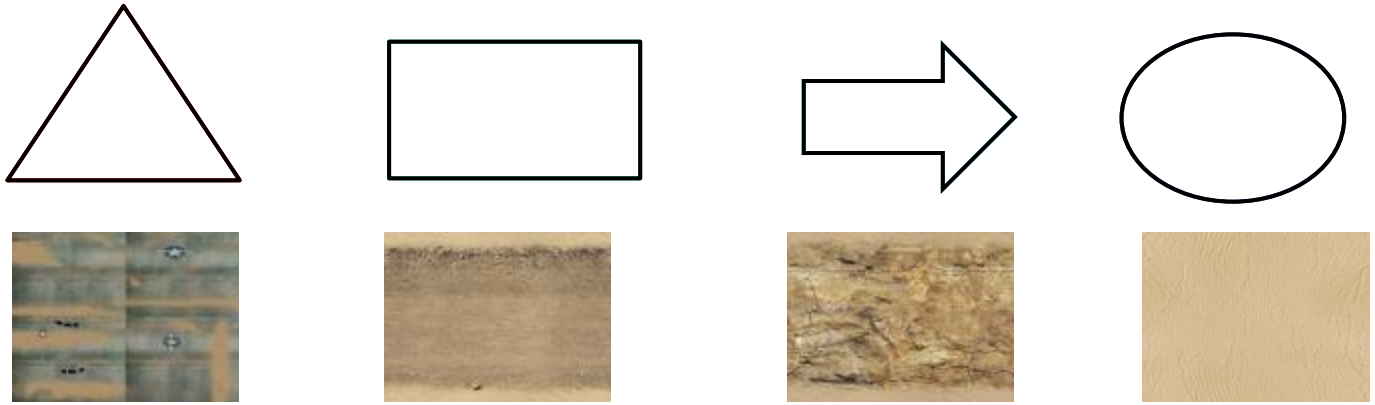
General Tips

- Quality setting is a time trade off
- Iterate low quality, ship high quality
- Compress in the asset pipeline
- Familiarize artists with the tools



ARM

Atlassing?

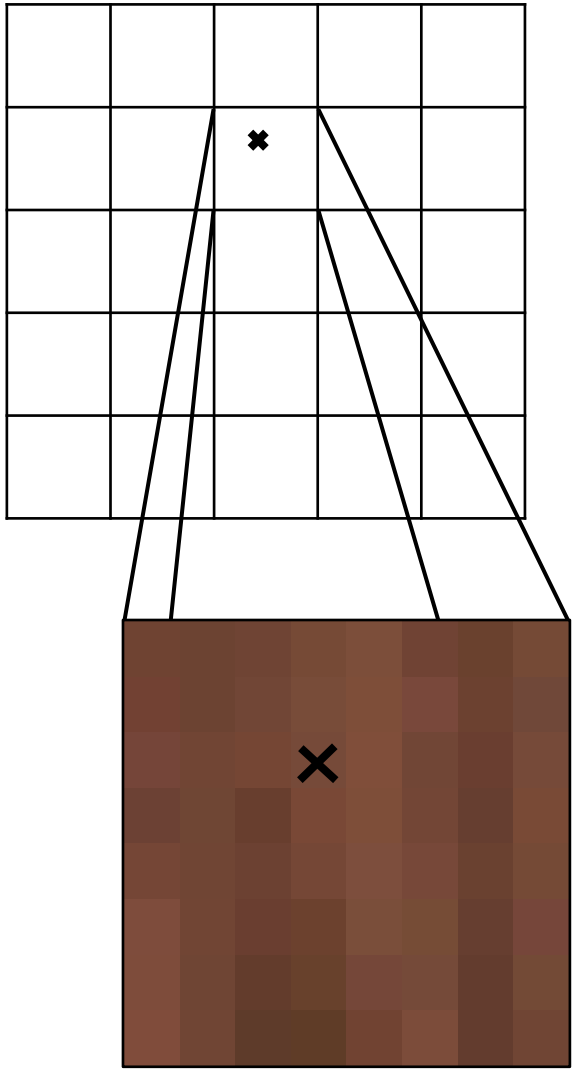


Atlassing Compressed Textures

- The conventional wisdom is that compressed textures cannot be altered
- If you change the content it has to be recompressed
- This isn't *entirely* true

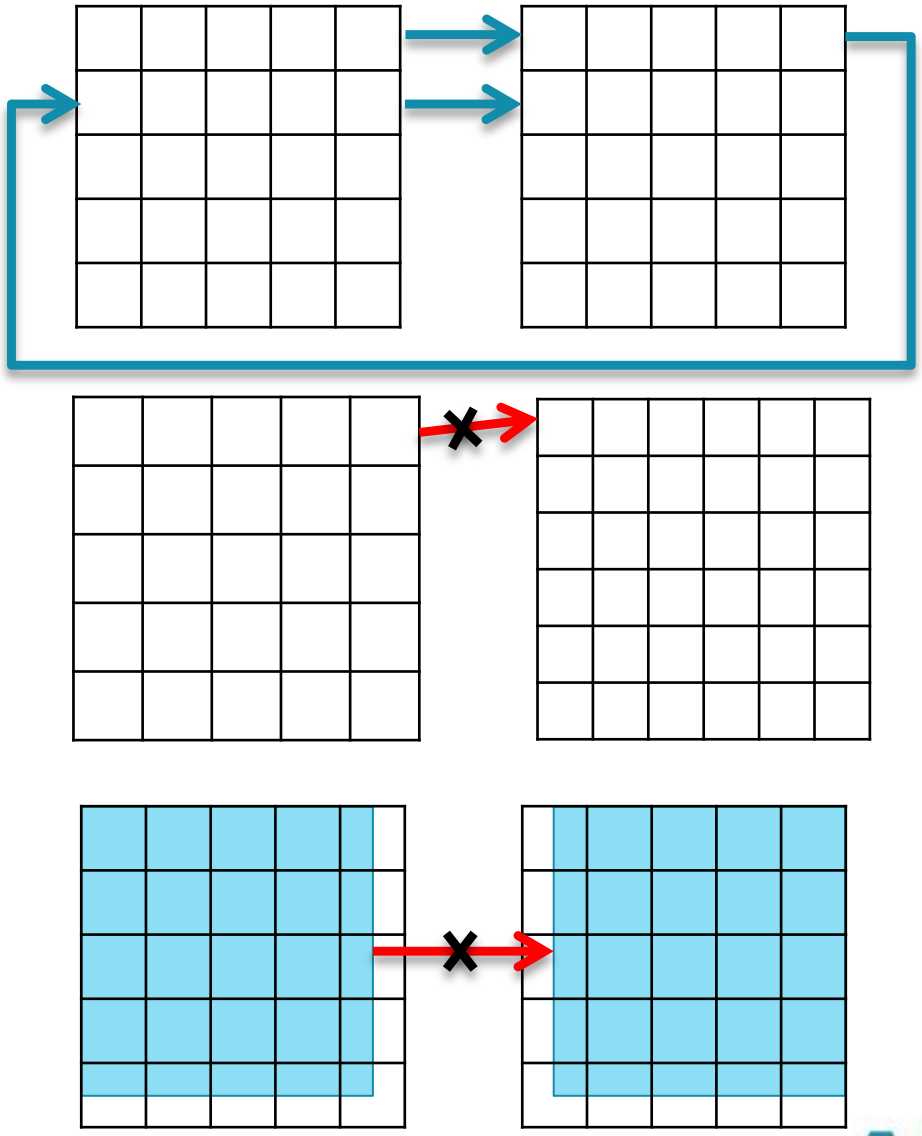
Atlassing Compressed Textures

- Compressed textures are block based
- Hardware decompression uses this
  - Use coords to look up block
  - Decompress block
  - Use relevant pixels
- Blocks are wholly independent
- Textures with the same block size can be stuck together



Atlassing Compressed Textures

- Combined textures must be in the same format, with the same global settings
  - Encoding
  - Block size
- Quality settings can differ
  - Purely an argument of the compression
- Textures must stitch at block boundaries



ARM

malideveloper.arm.com

community.arm.com

Thank You

Any questions?