



Shading of Spellsouls: Achieving AAA Quality on Mobile

Srdja Stetic-Kozic
Tech Lead
Nordeus

About Nordeus



Spellsouls

- AAA experience on mobile
- Wide range of devices



Challenges

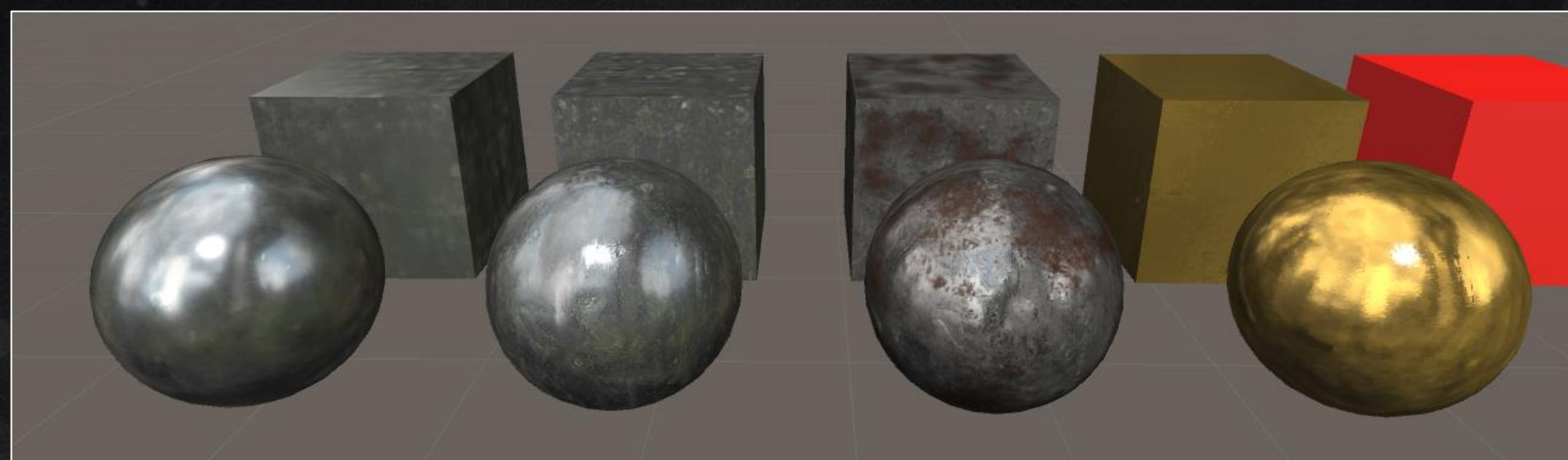
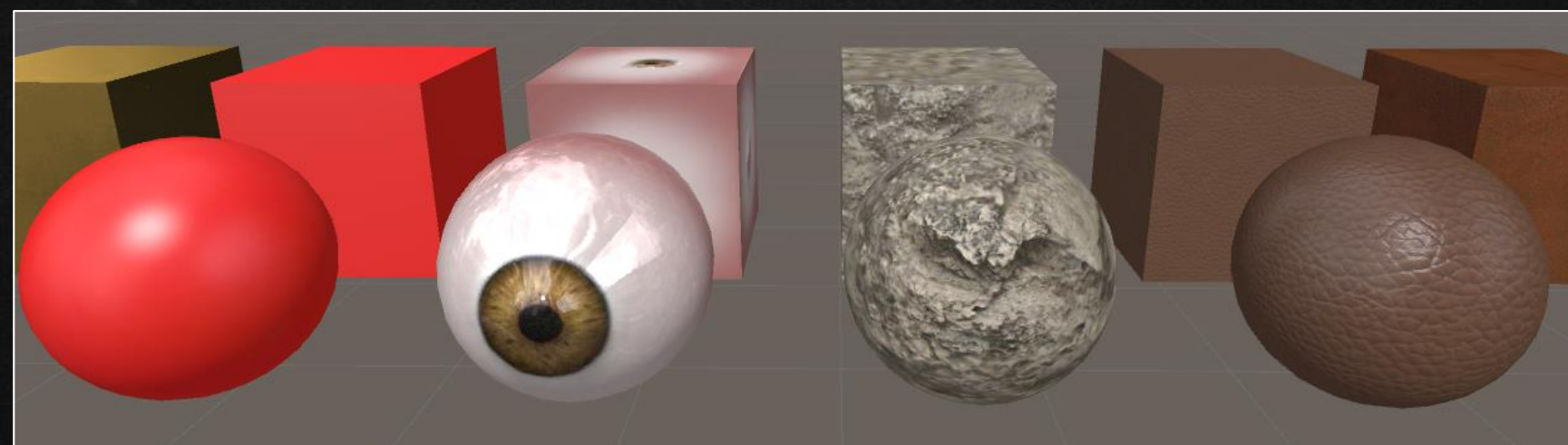
- PBR Shading
- VFX Shading
- Skinning
- Optimizing to 60 FPS



PBR

- Realistic look, different lighting conditions
- Standard GGX approach is expensive
- Normalized Blinn-Phong





Material parameters texture



- Roughness - standard
- Metalness - reflection color
- Reflectivity - environment reflection



Normalized Blinn-Phong

$$specularPower = (512, 1 - roughness)$$

$$\frac{(1.04 - roughness) * (specularPower + 8)}{8} * (N \cdot H)^{specularPower}$$



Diffuse



Environment light



Specular &
rim lighting



Linear color space

- Necessary for PBR
- Only 50% of devices support it



Linear color space



Gamma = 2.2



Gamma = 1

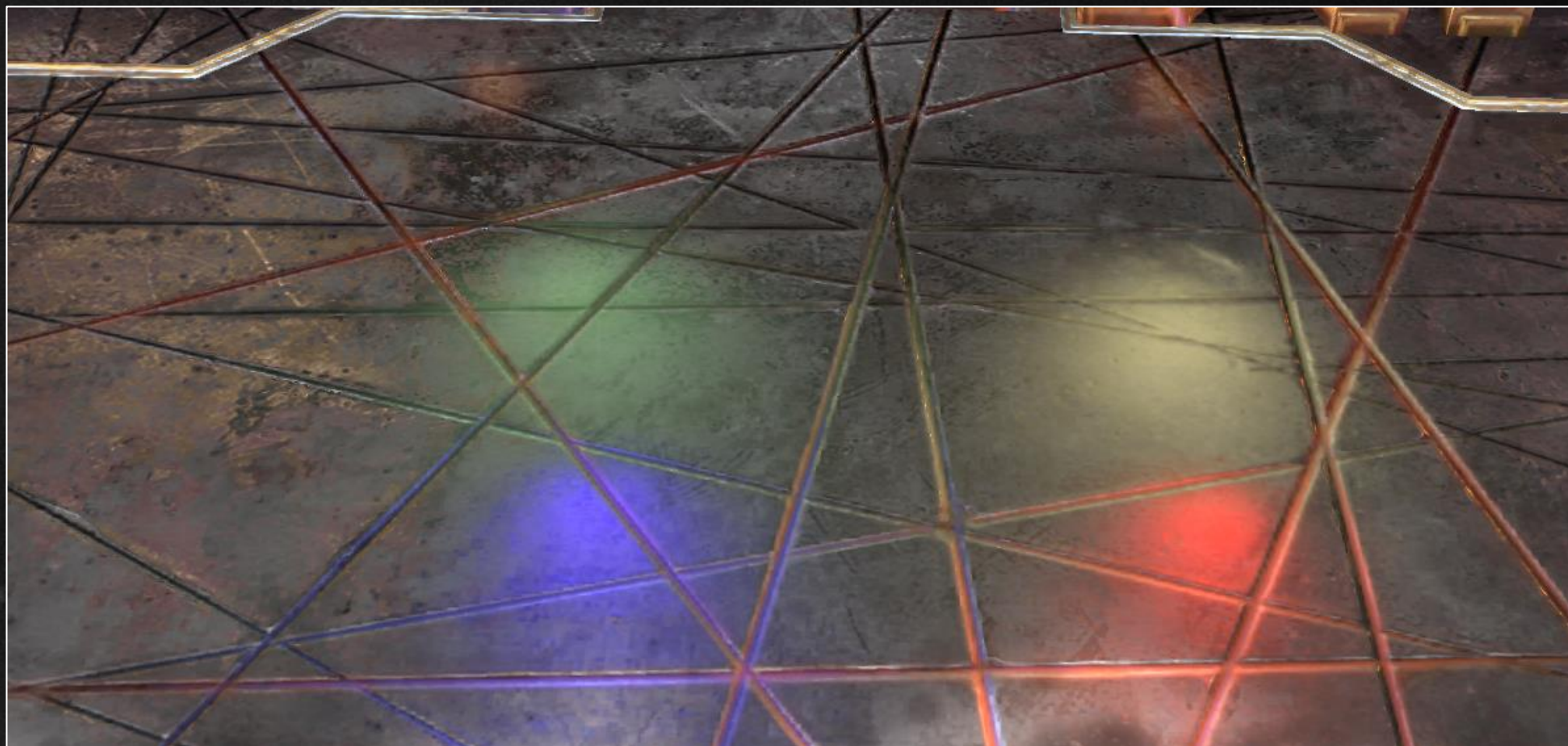
Linear color space

- Went with Gamma = 2.0
- On fetch – $\text{correctedColor} = \text{color} * \text{color}$
- Shader correction – $\text{correctedColor} = \text{pow}(\text{color}, 1/2.2)$



Additional lights

- 4 point lights
- Per object lights?
- Forward+?



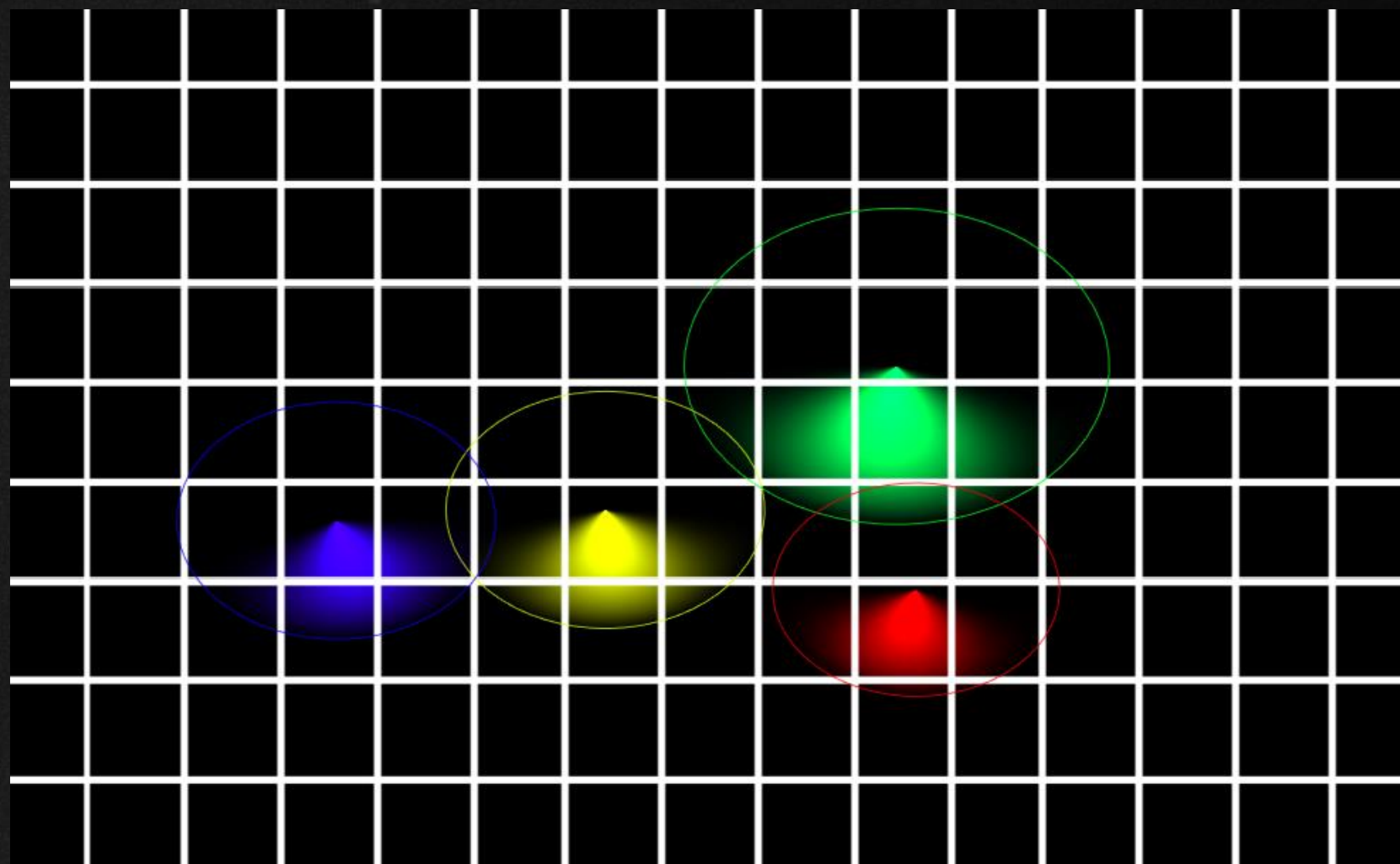
Forward+?

- Spotty compute shader support
- Already GPU bound



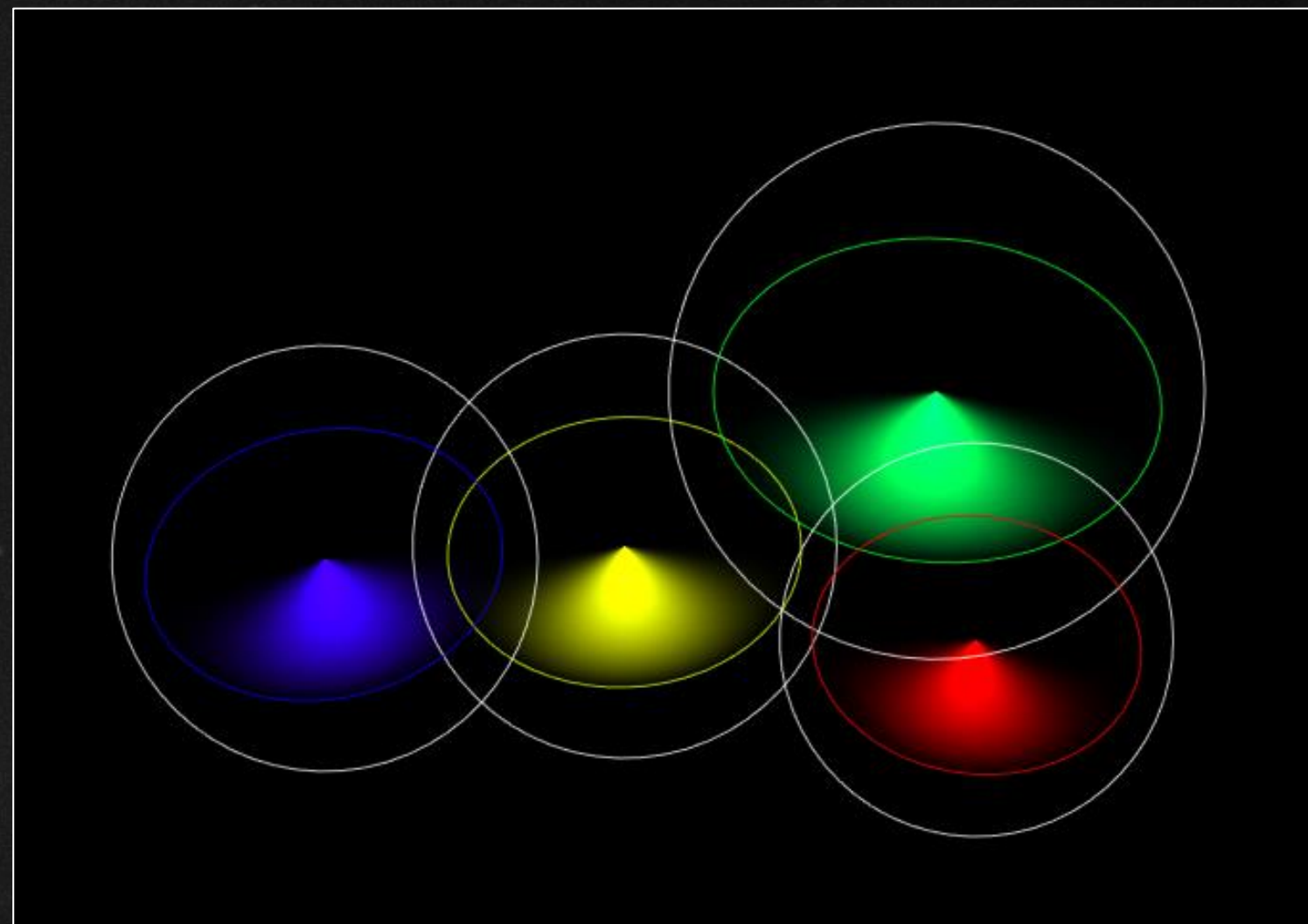
Forward+?

- Try to simplify?
- No depth pre-pass
- CPU light culling
 - 64x64 pixel tiles
 - Approximate light area with spheres



Bounding circle

- Can use when perspective is not strong
- Project assuming camera is orthographic and increase radius for safety
- Strong perspective? – project into an AABB



Shadows

Dynamic shadows for moving objects

Static shadows for the environment

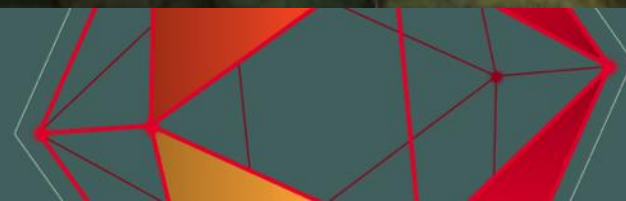


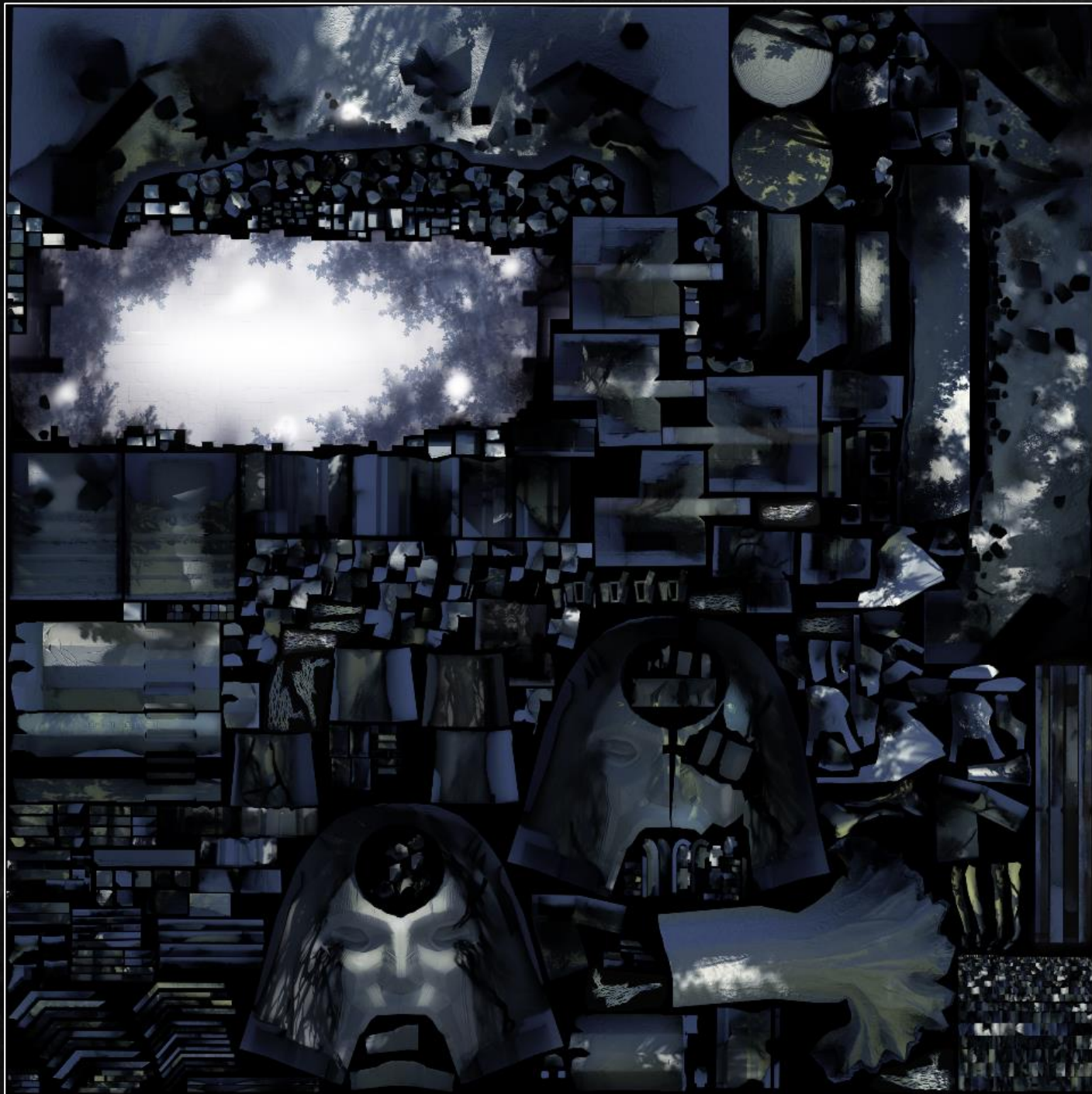
Dynamic shadows

- Dynamic shadowmap
- Hardware 4-tap PCF

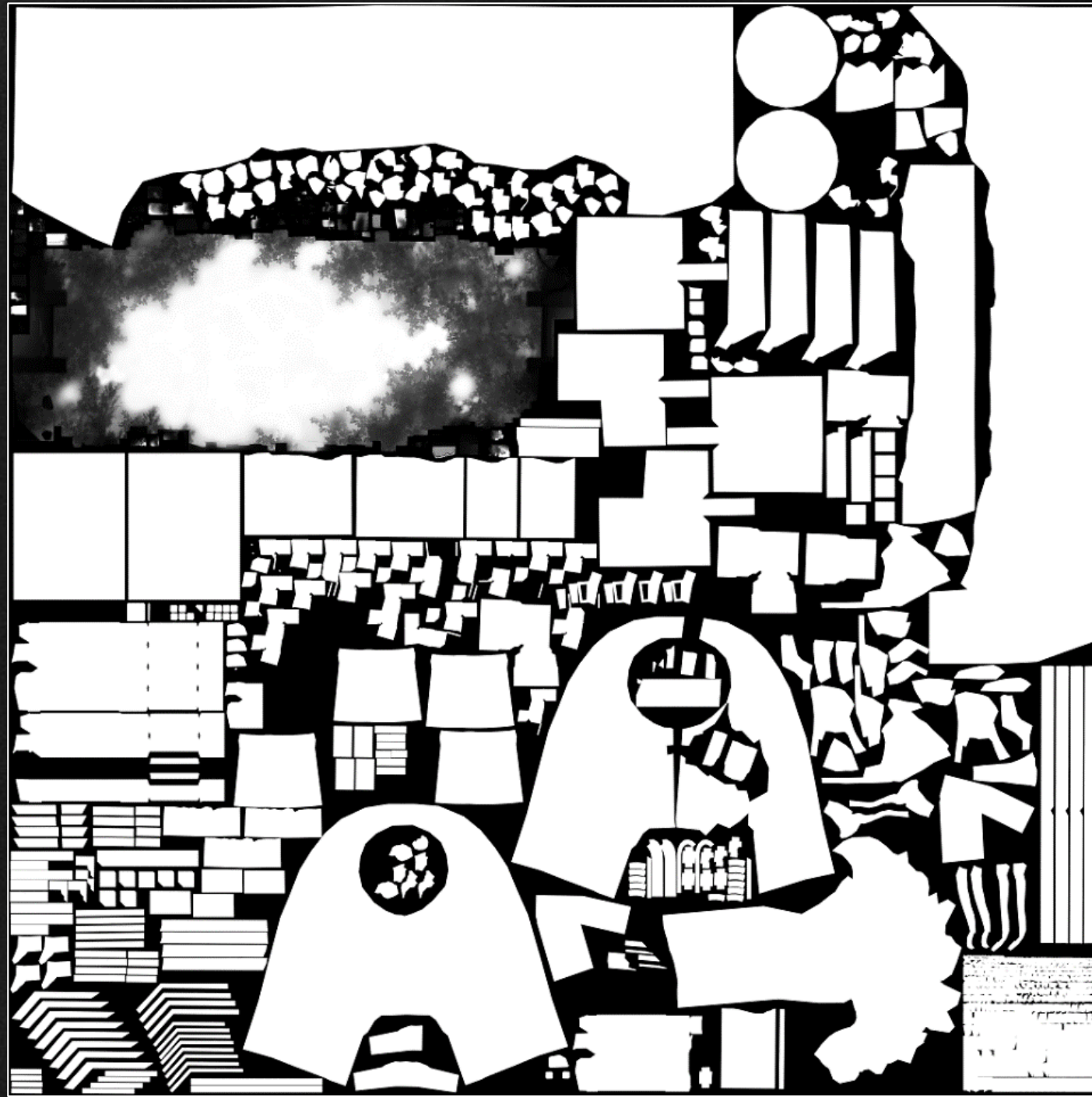


Static shadows





Lightmap



Shadowmap



Combining shadows

- Combining is done when rendering the floor
- Both lightmap color and dynamic shadow color are evaluated for every pixel



Combining shadows

$$color = albedo.rgb * lightmap.rgb$$
$$color = color * lerp(lightColor, shadowColor, lightmap.a * \min(N \cdot L, dynamicShadow))$$



VFX



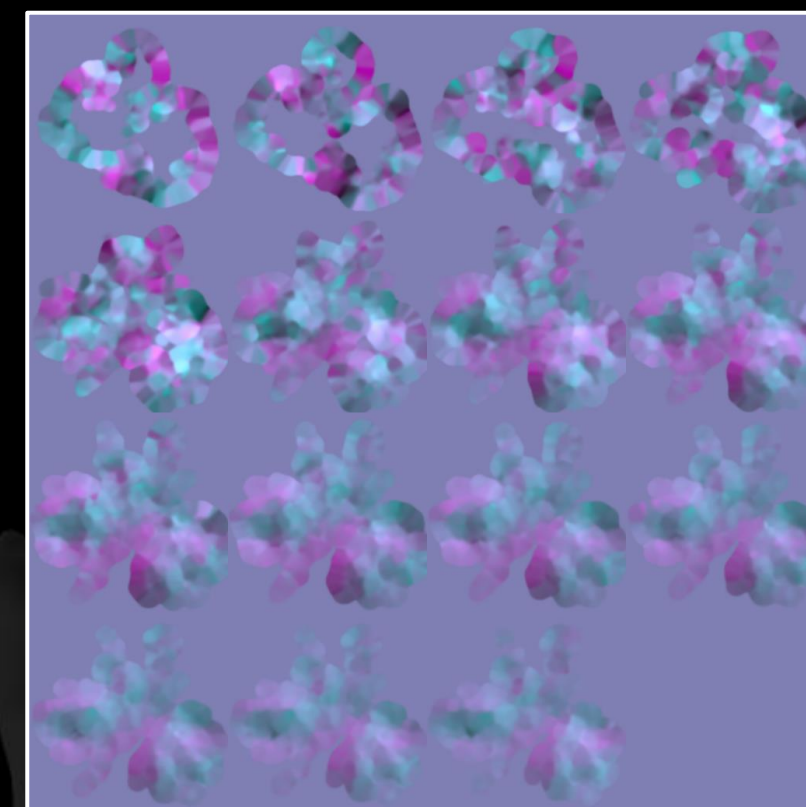
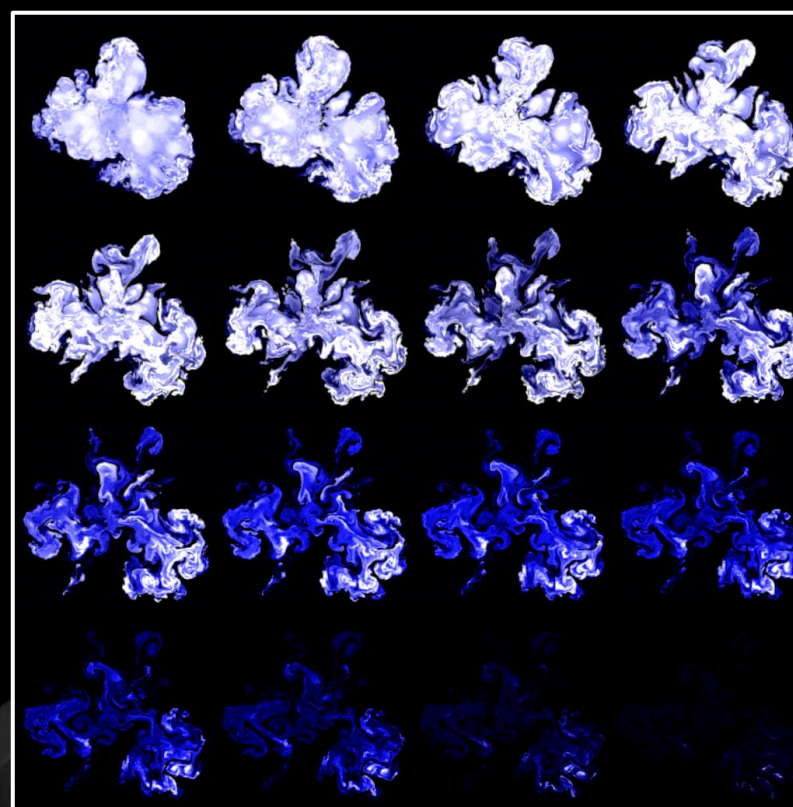
VFX

- No postprocessing
- Particle systems are expensive
- Spritesheets!



Motion Vectors

Spritesheet - Low FPS

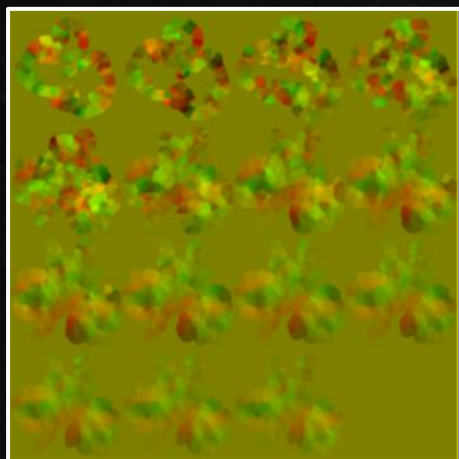


Motion Vectors



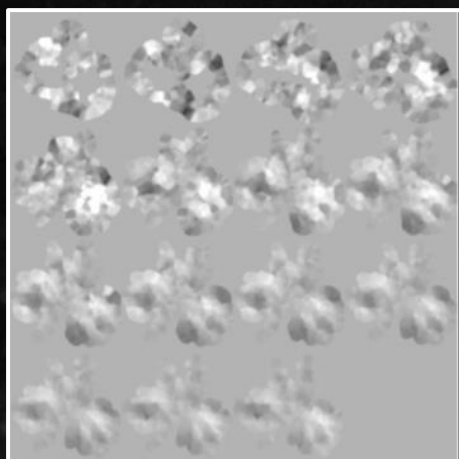
Motion Vectors

norde.us/vfxmv



Red & Green channels

UV coordinates for next pixel to read



Blue channel

Scale factor for Red and Green channels

1. Read current frame pixel and motion vectors value from same UV coordinates
2. Determine UV coordinates to read from next frame using motion vectors
3. Interpolate between current pixel and the next frame's one



Skinning

- CPU skinning was taking 2 full cores
- Uploading meshes to GPU every frame was killing performance



Endgame scenario with 40 units



GPU Skinning

Requirements:

- No GPU mesh uploads
- Supports instancing
- Fast



Texture Based Matrix-palette GPU Skinning

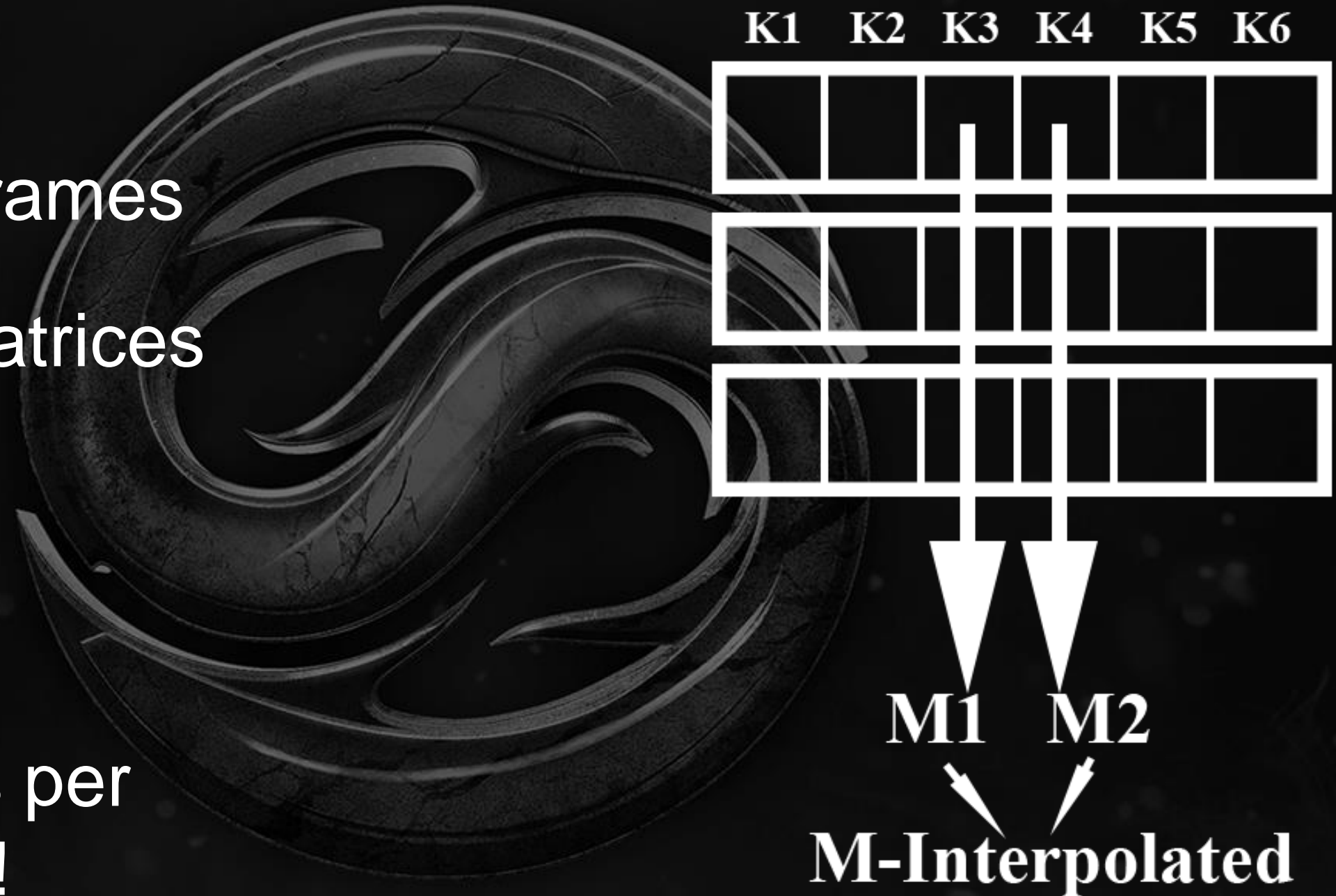
- Sample bone TRS at regular intervals
- Bake bone TRS into textures
- 3x4 floats – 3 textures needed
- Per Instance data – 1 float, U coordinate
- How to interpolate?

	K1	K2	K3	K4	K5	K6
Bone 0						
Bone 1						
⋮						



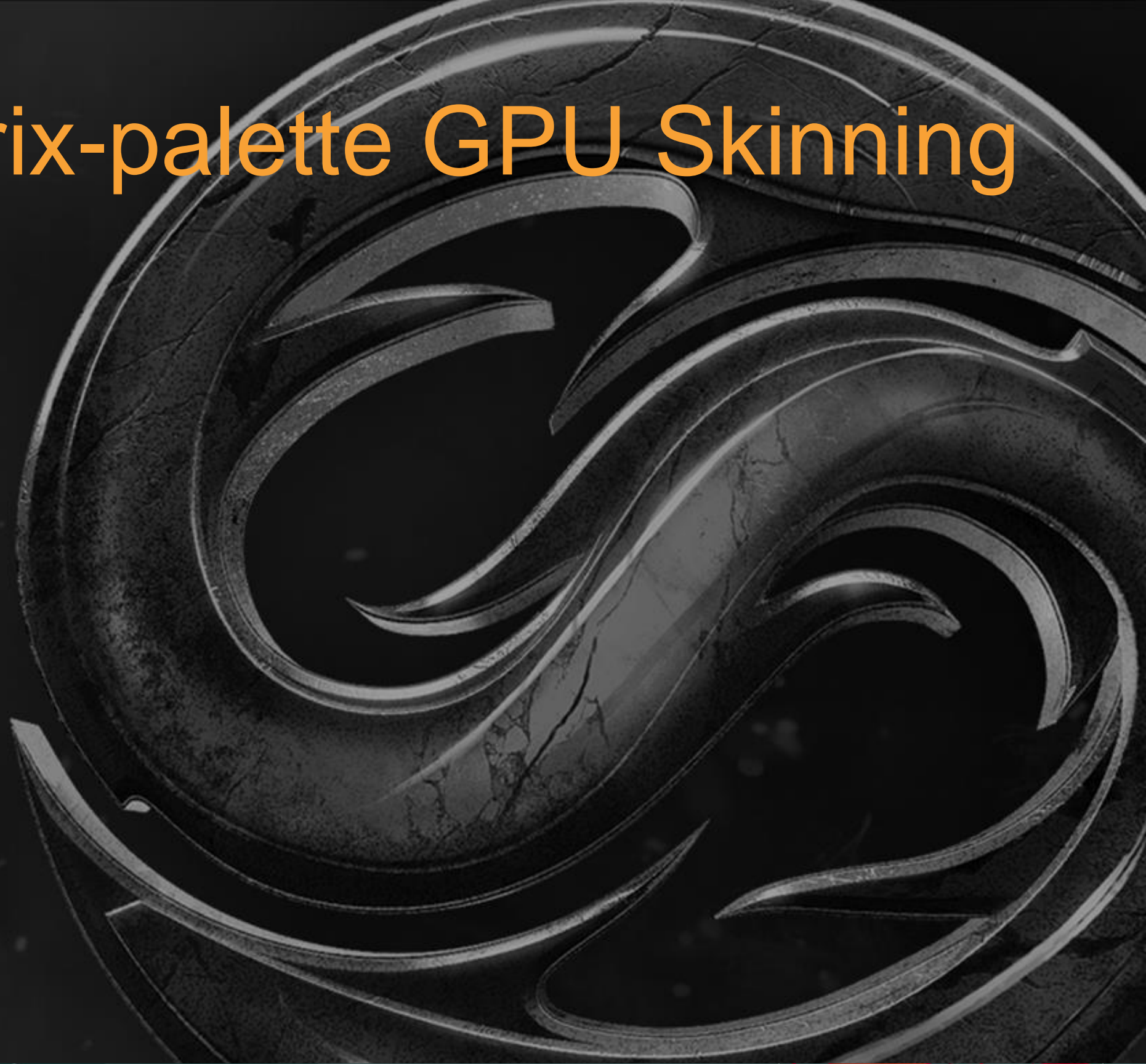
Texture Based Matrix-palette GPU Skinning

1. Read two keyframes
 2. Reconstruct matrices
 3. Interpolate
- 6 texture reads per bone influence!



Texture Based Matrix-palette GPU Skinning

- Problems to solve:
 - 6 texture reads
 - 3 textures
 - Interpolation math



Texture Based Dual Quaternion GPU Skinning

- Dual quaternions! - norde.us/dualq
- 2 textures – 3rd scaling texture optional
- Bilinear filtering
- Looks good with low sampling rate – 15FPS for us
- Blending animations? – Double texture reads





Performance & optimization

We optimize for:

- Framerate
- Heating
- Battery drain



Graphics Profilers

Good for fast
shader iteration

Shader Viewer

Send Revert Retain-Modified

Modified Original

```
51 void main()
52 {
53     u_xlat10_0.xyz = texture(_BumpMap, vs_TEXCOORD0.xy).xyz;
54     u_xlat16_0.xyz = u_xlat10_0.xyz * vec3(2.0, 2.0, 2.0) + vec3(-1.0, -1.0, -1.0);
55     u_xlat16_1.xyz = u_xlat16_0.yyy * vs_TEXCOORD4.xyz;
56     u_xlat16_1.xyz = vs_TEXCOORD3.xyz * u_xlat16_0.xxx + u_xlat16_1.xyz;
57     u_xlat16_1.xyz = vs_TEXCOORD5.xyz * u_xlat16_0.zzz + u_xlat16_1.xyz;
58     u_xlat16_31 = dot(u_xlat16_1.xyz, u_xlat16_1.xyz);
59     u_xlat16_31 = inversesqrt(u_xlat16_31);
60     u_xlat16_1.xyz = vec3(u_xlat16_31) * u_xlat16_1.xyz;
61     u_xlat0.xyz = vs_TEXCOORD7.xyz + (-WorldSpaceCameraPos.xyz);
62     u_xlat30 = dot(u_xlat0.xyz, u_xlat0.xyz);
63     u_xlat30 = inversesqrt(u_xlat30);
64     u_xlat0.xyz = vec3(u_xlat30) * u_xlat0.xyz;
65     u_xlat16_31 = dot(u_xlat0.xyz, u_xlat16_1.xyz);
66     u_xlat16_31 = u_xlat16_31 + u_xlat16_31;
67     u_xlat16_2.xyz = u_xlat16_1.xyz * (-vec3(u_xlat16_31)) + u_xlat0.xyz;
68     u_xlat0.x = dot(u_xlat16_1.xyz, (-u_xlat0.xyz));
69 #ifdef UNITY_ADRENO_ES3
70     u_xlat0.x = min(max(u_xlat0.x, 0.0), 1.0);
71 #else
72     u_xlat0.x = clamp(u_xlat0.x, 0.0, 1.0);
73 #endif
74     u_xlat0.x = (-u_xlat0.x) + 1.0;
75     u_xlat0.x = log2(u_xlat0.x);
76     u_xlat0.x = u_xlat0.x * _RimLightCoeff;
77 }
```

Shader is valid:

Device: Qualcomm Adreno (TM) 330
Total Instructions 250

Full Precision ALU Instructions . 63
Half Precision ALU Instructions . 144
Interpolation Instructions 0
Texture Fetches 5
Memory Load Instructions 0
Memory Store Instructions 0
Flow Control Instructions 1
No-Op Instructions 37
Synchronization Instructions 0
Short Latency Sync Instructions . 7
Long Latency Sync Instructions .. 3
Number of Registers 8
Number of Full Registers 3
Number of Half Registers 9
EFU Instructions 14



Shader instructions

- Never use fixed precision
- Convert between precisions sparingly
- Watch out for No-ops

```
Total Instructions ..... 250
Full Precision ALU Instructions . 63
Half Precision ALU Instructions . 144
Interpolation Instructions ..... 0
Texture Fetches ..... 5
Memory Load Instructions ..... 0
Memory Store Instructions ..... 0
Flow Control Instructions ..... 1
No-Op Instructions ..... 37
```


No-ops

```
float4x4 sum = (boneMatrices[index0] * weight0) + (boneMatrices[index1] * weight1);
```

250 No-ops

```
float4x4 matrix0 = boneMatrices[index0];  
float4x4 matrix1 = boneMatrices[index1];
```

20% Faster

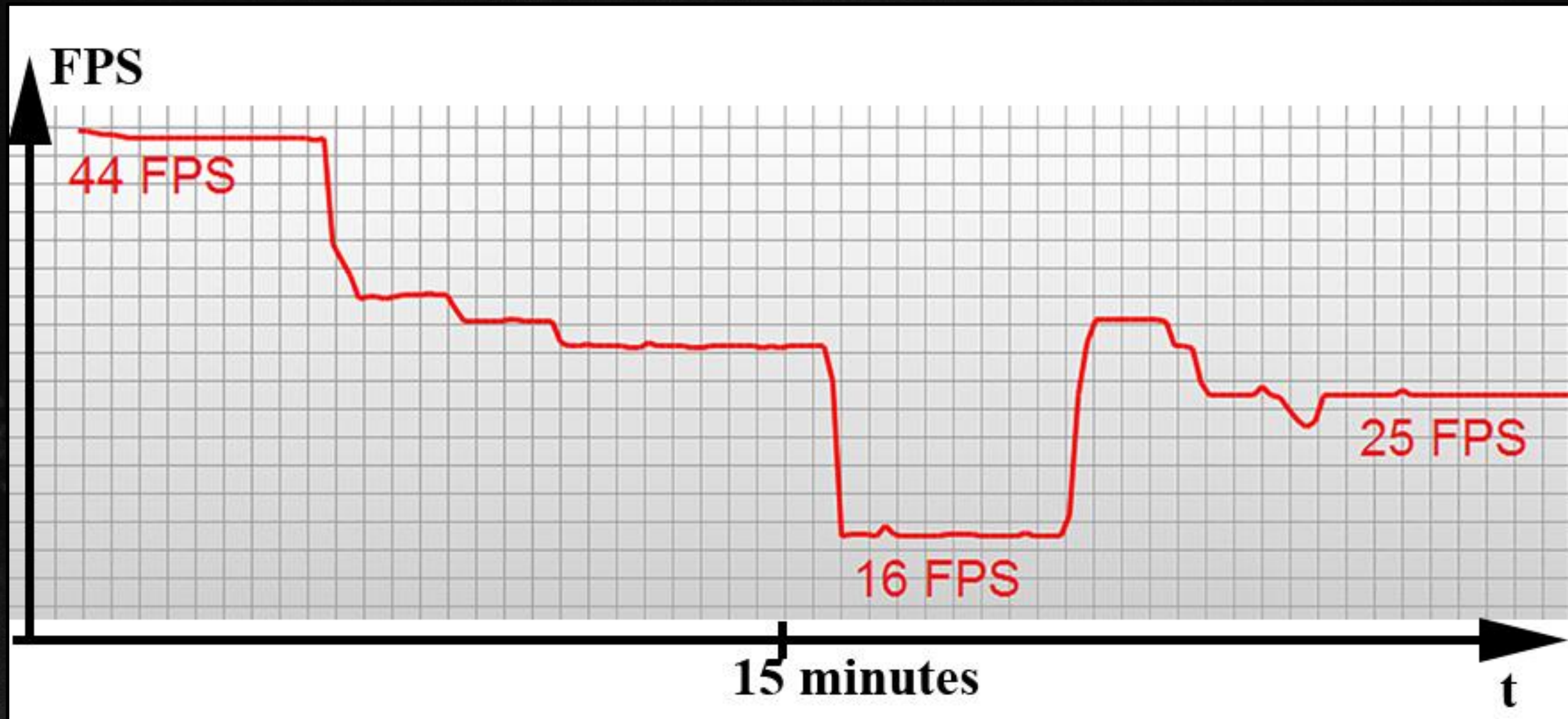
180 No-ops

ALU here →

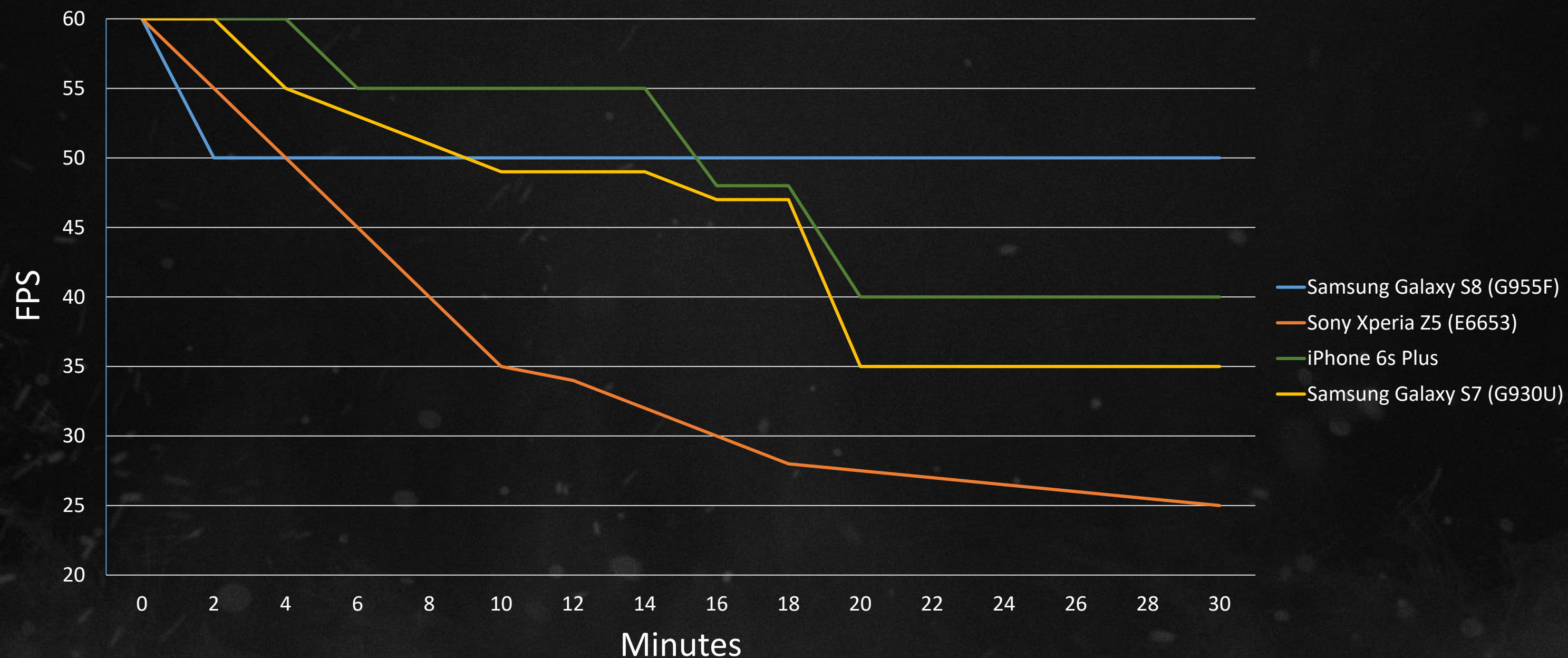
```
float4x4 sum = (matrix0 * weight0) + (matrix1 * weight1);
```



Thermal throttling

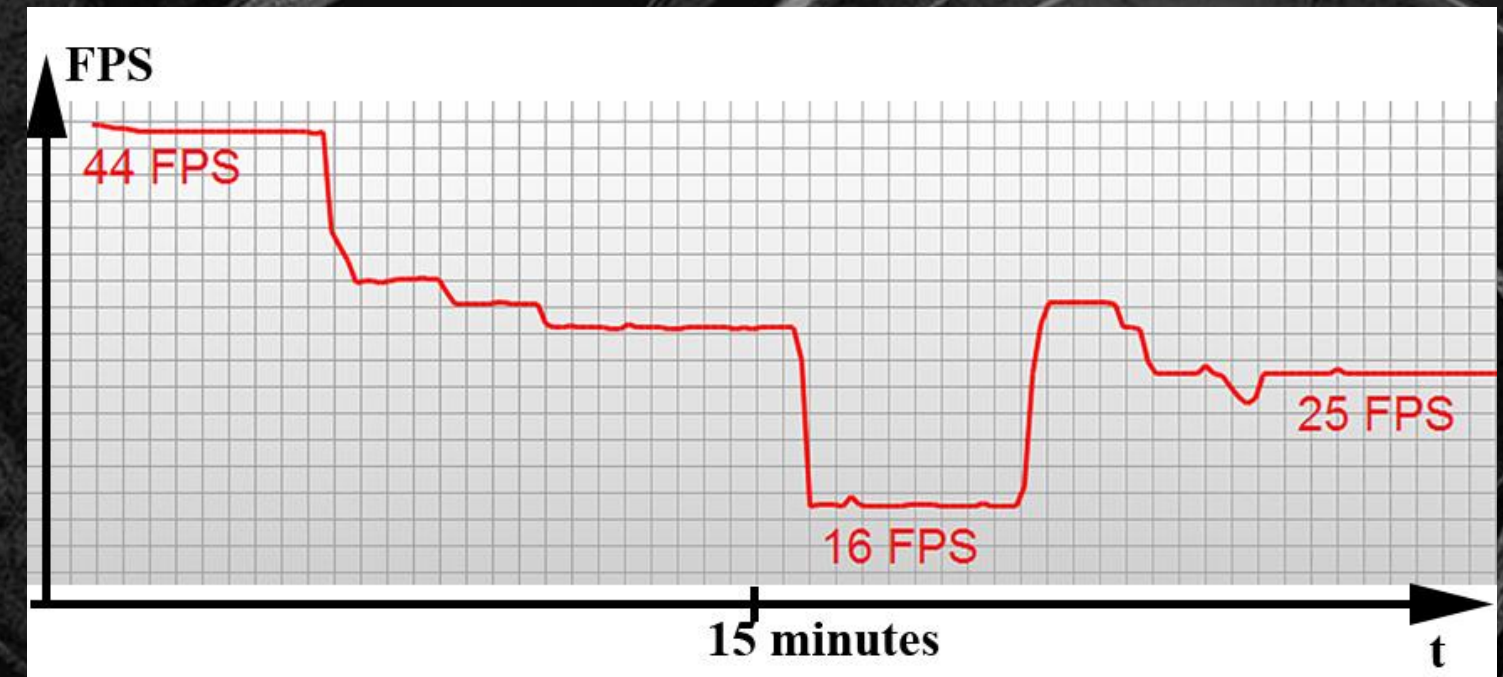


Thermal throttling



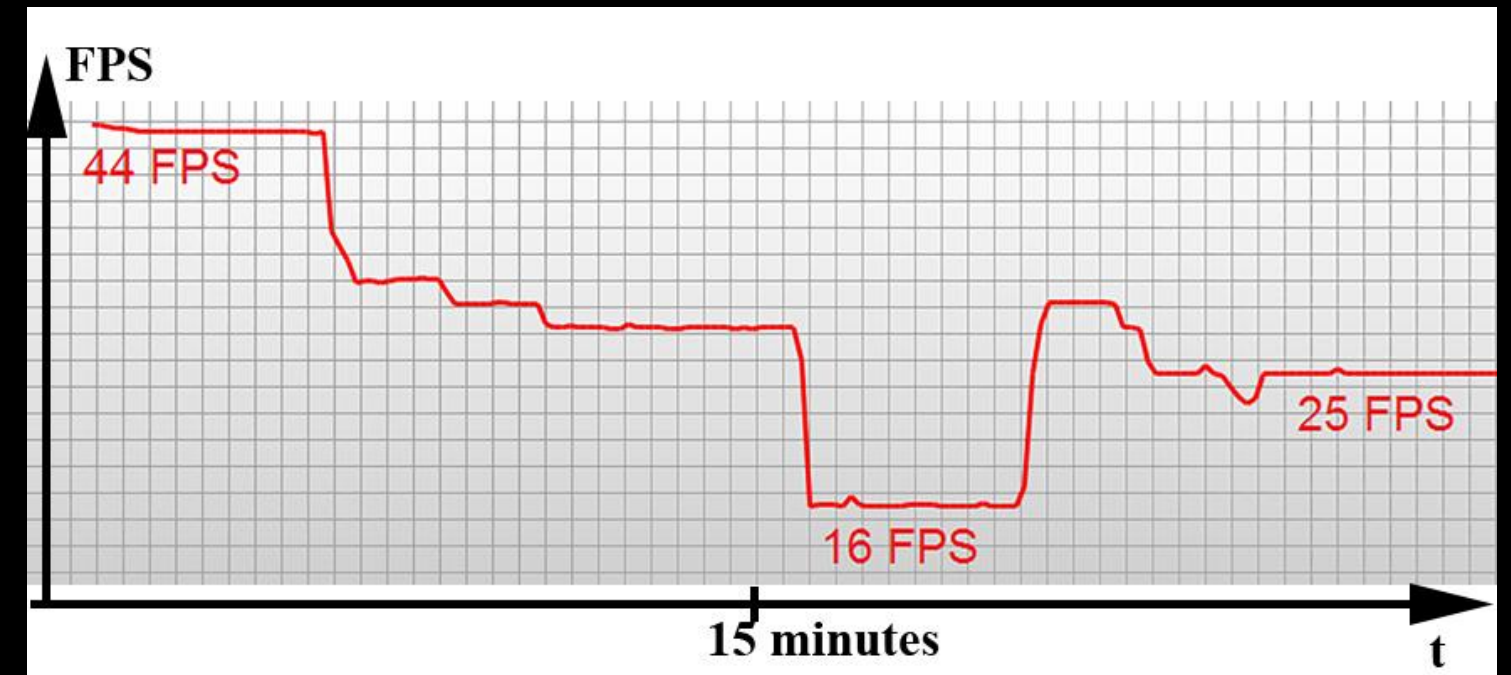
Thermal throttling

1. Wait until the device goes into stable state (25FPS)
2. Determine target framerate – in this case 30FPS
3. Set graphical quality so FPS is 30% above the target – 40FPS
4. Cap framerate to target – 30FPS



Thermal throttling

- Benefits:
 - We are not using all of the computational resources of the device
 - Amortizes frametime spikes
- Quality settings determined per GPU



Performance tracking

- Analytics tracking:
 - Average FPS
 - Battery drain
- Battery drain is correlated with heating
- New settings distributed on every game start from our server



Recap

- Specialize your techniques
- It's worth it to go old-school
- Be mindful of heating





Questions?

srdjas@nordeus.com

GDC®

Thank You!

GAME DEVELOPERS CONFERENCE® | MARCH 19-23, 2018 | EXPO: MARCH 21-23, 2018 #GDC18

