Parallel Computing in Video Games

Oliver Franzke - Double Fine Productions

p1xelcoder.bsky.social

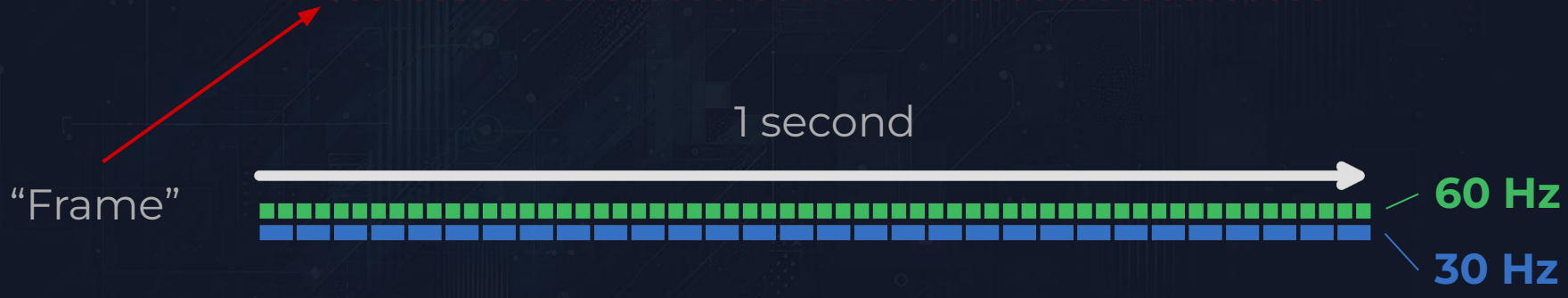

KEEPER
FULL THROTTLE
REMASTERED
PSYCHONAUTS 2
THE CAVE
Day of the Tentacle
REMASTERED
XBOX 360
KINECT
REQUIRES
SESAME STREET
Once upon a MONSTER
STACKING
BROKEN AGE
HEAD LANDER
ATARI
Ballance
PLAYSTATION 3
singstar
Lucidity
LUCASARTS
MONKEY ISLAND
SPECIAL EDITION
2
Michael Schumacher
RACING WORLD
KART 2002
MIDDLE MANAGER of JUSTICE
GRIM FANDANGO
Remastered
AUTONOMOUS
VIDEO GAMES
BUILD ROBOTS • COLLECT ENERGY • SURVIVE
MNEMONIC
Costume Quest 2
OPERATION
CREATURE FEATURE
MONKEY ISLAND
SPECIAL EDITION
THE TRIALS OF TOPOQ
DF 9

Video Games

What are they?

Conceptual Model of a Game

```
while True:
    const playerInput = GetPlayerInput()
    Update(playerInput)
    Render()
```

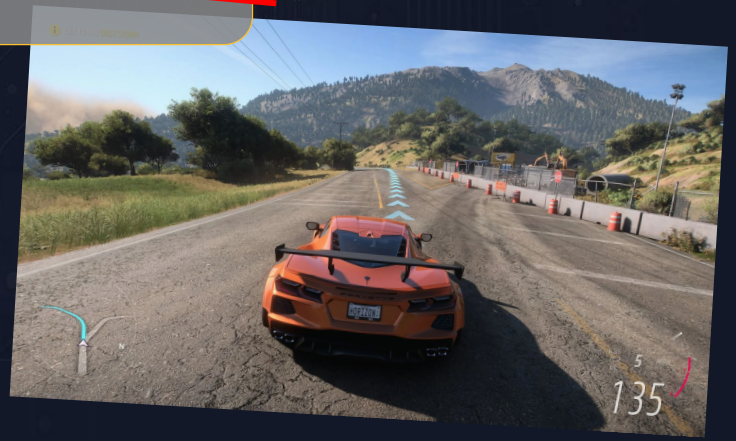


- Hardware Evolution
- Parallelizing Graphics
- Challenge of Game-play Logic
- Alternative Architecture

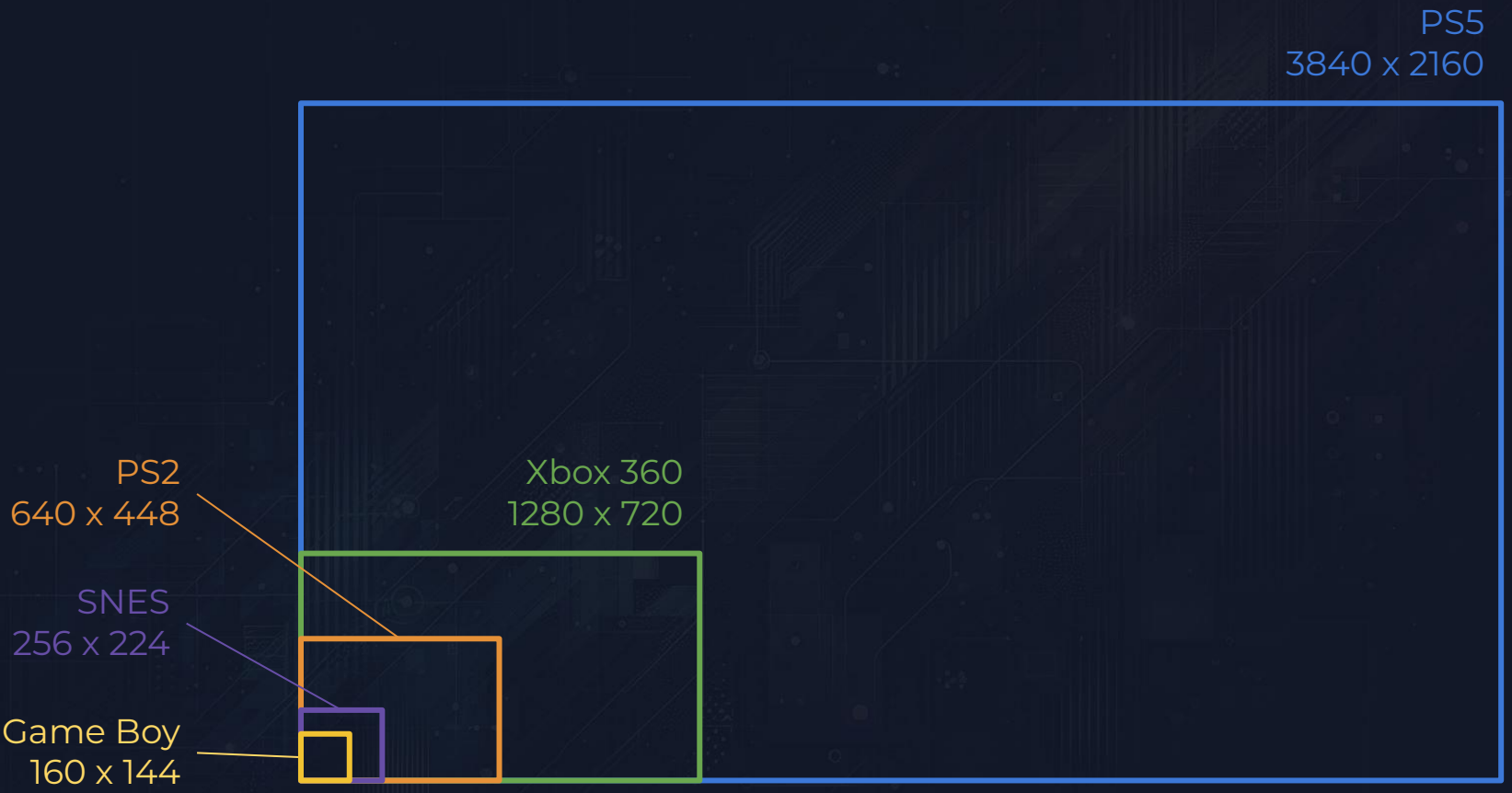
~12.000.000 x faster!

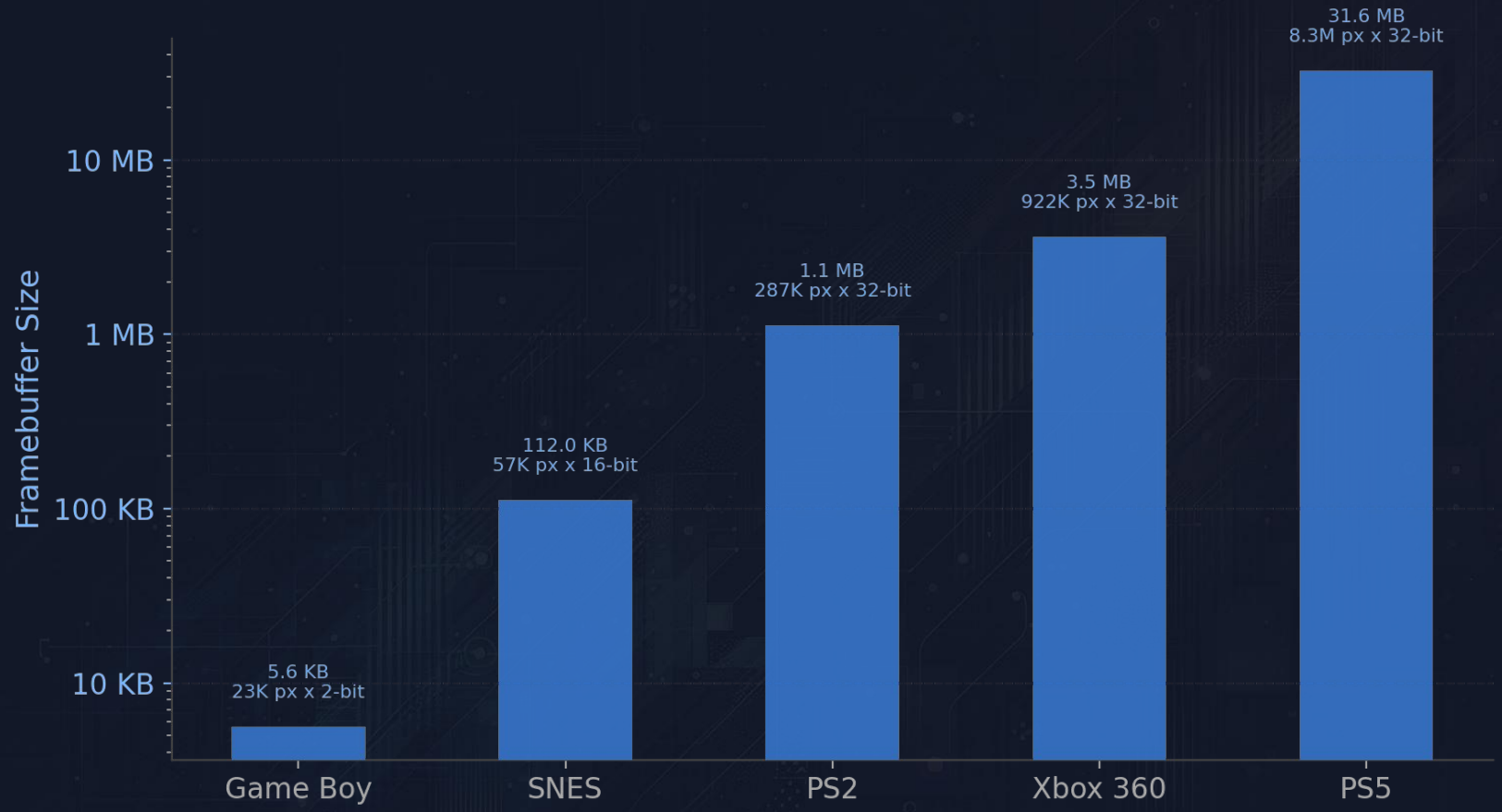
$$Fun = \int_{0}^{t_{end}} \left(\sum_{i=1}^{N} pixels_i(t) + \sum_{j=1}^{M} samples_j(t) \right) dt$$

?

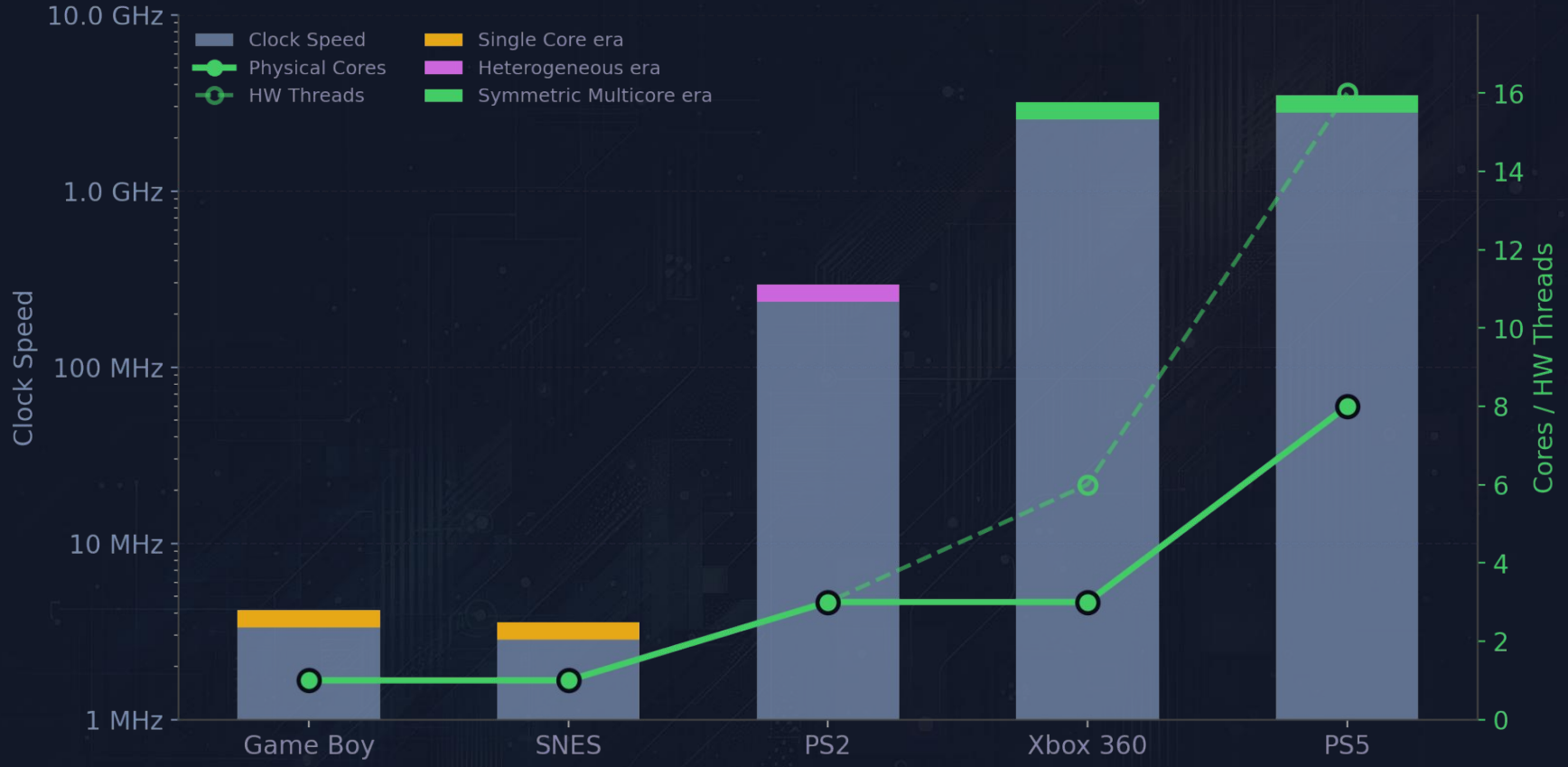


Game Boy	SNES	PS 1	PS 2	Xbox 360	PS 5
1989	1990	1994	2000	2005	2020
	2 x	32 x	5600 x	260.000 x	10.000.000 x


PS5
3840 x 2160
Xbox 360
1280 x 720
PS2
640 x 448
SNES
256 x 224
Game Boy
160 x 144


Framebuffer Size
10 MB
1 MB
100 KB
10 KB
5.6 KB
23K px x 2-bit
112.0 KB
57K px x 16-bit
1.1 MB
287K px x 32-bit
3.5 MB
922K px x 32-bit
31.6 MB
8.3M px x 32-bit
Game Boy
SNES
PS2
Xbox 360
PS5


Framebuffer (log)
GPU Parallel Units (log)
Framebuffer Size
GPU Parallel Units
10 MB
1 MB
100 KB
10 KB
1,000
100
10
1
5.6 KB
112.0 KB
1.1 MB
3.5 MB
31.6 MB
Game Boy
SNES
PS2
Xbox 360
PS5


Clock Speed
Physical Cores
HW Threads
Single Core era
Heterogeneous era
Symmetric Multicore era
10.0 GHz
1.0 GHz
100 MHz
10 MHz
1 MHz
Clock Speed
16
14
12
10
8
6
4
2
0
Cores / HW Threads
Game Boy
SNES
PS2
Xbox 360
PS5

Where to go from here?

```
while True:
    const playerInput = GetPlayerInput()
    Update(playerInput)
    Render()
```

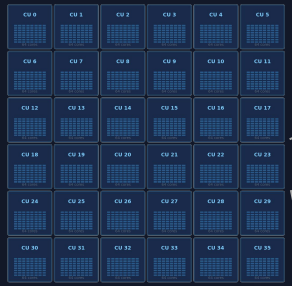
Let’s start here!

CU 0	CU 1	CU 2	CU 3	CU 4	CU 5
CU 6	CU 7	CU 8	CU 9	CU 10	CU 11
CU 12	CU 13	CU 14	CU 15	CU 16	CU 17
CU 18	CU 19	CU 20	CU 21	CU 22	CU 23
CU 24	CU 25	CU 26	CU 27	CU 28	CU 29
CU 30	CU 31	CU 32	CU 33	CU 34	CU 35

64 cores

36 Compute Units × 64 cores each = 2,304 shader cores

Shader core clock speed: 2.23 GHz



36 Compute Units × 64 cores each = 2,304 shader cores

INSTRUCTION DECODER

Fetches & decodes 1 instruction

WAVEFRONT SCHEDULER

Assigns work to cores

▼ broadcasts to all 64 cores ▼

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

64 ALUs

LOCAL DATA SHARE

64 KB shared cache

REGISTER FILE

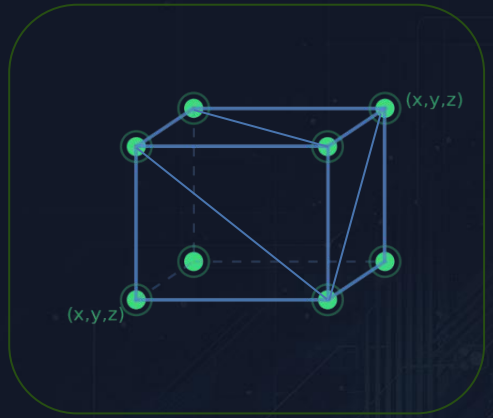
256 KB private data

TEXTURE UNITS

4 TMUs


Buffer A
Buffer B
read
Compute Shader
write
Buffer C
Buffer D
Buffer E

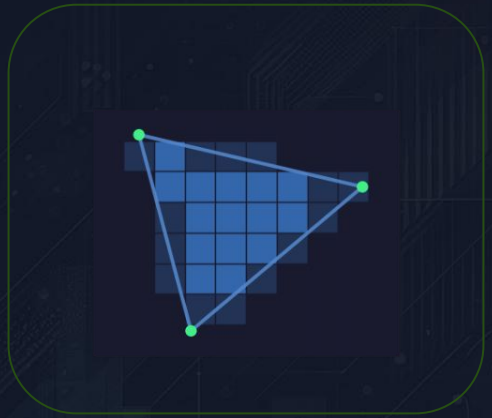
1



VERTEX PROCESSING

Transform → Project → Clip

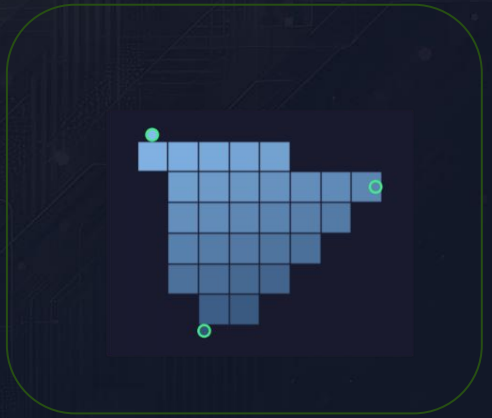
2



RASTERIZATION

Triangle → Fragments (pixels)

3



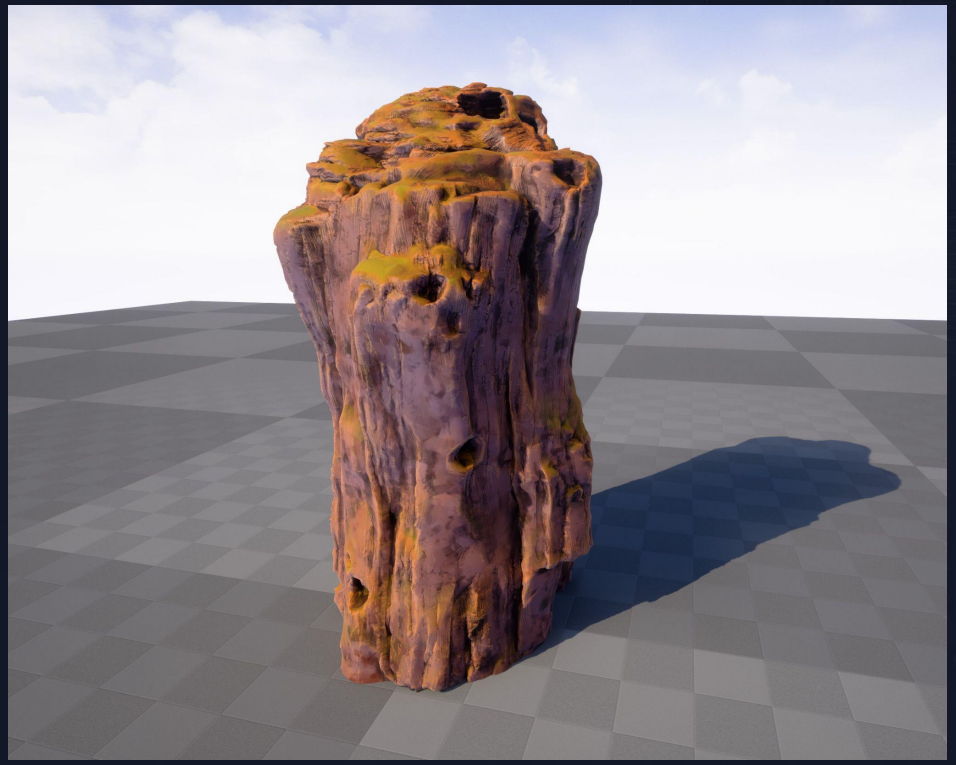
PIXEL SHADING

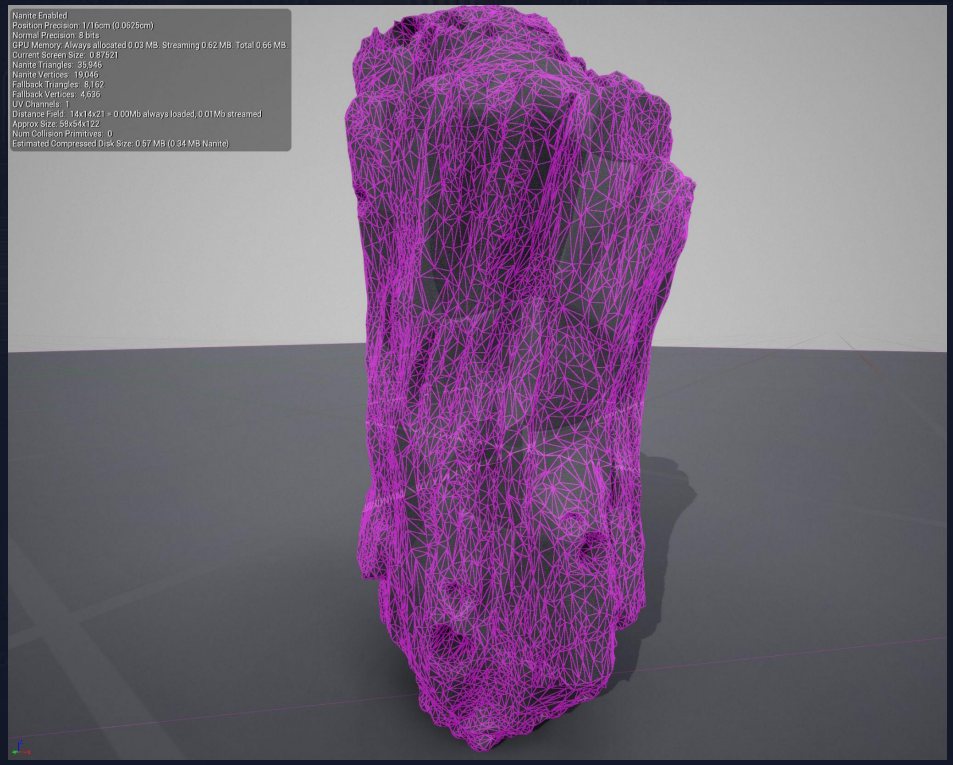
Per-pixel: lighting, color, texture

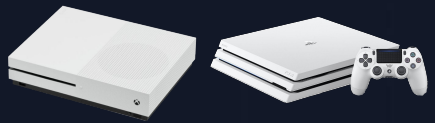

KEEPER
© Double Fine Productions


KEEPER
© Double Fine Productions


KEEPER

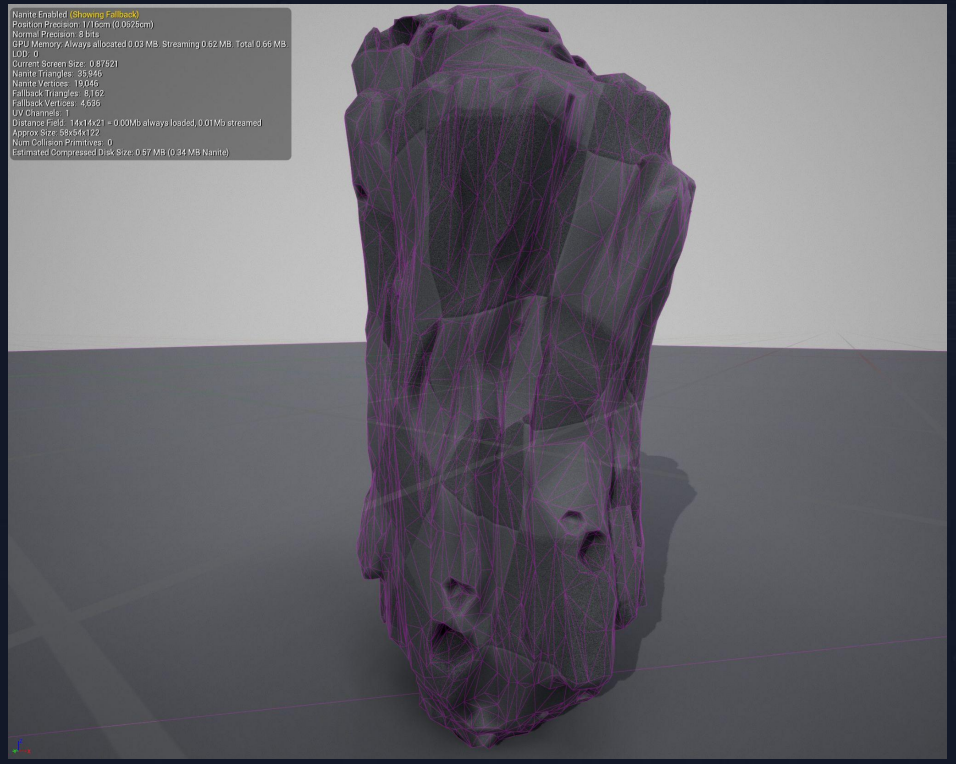


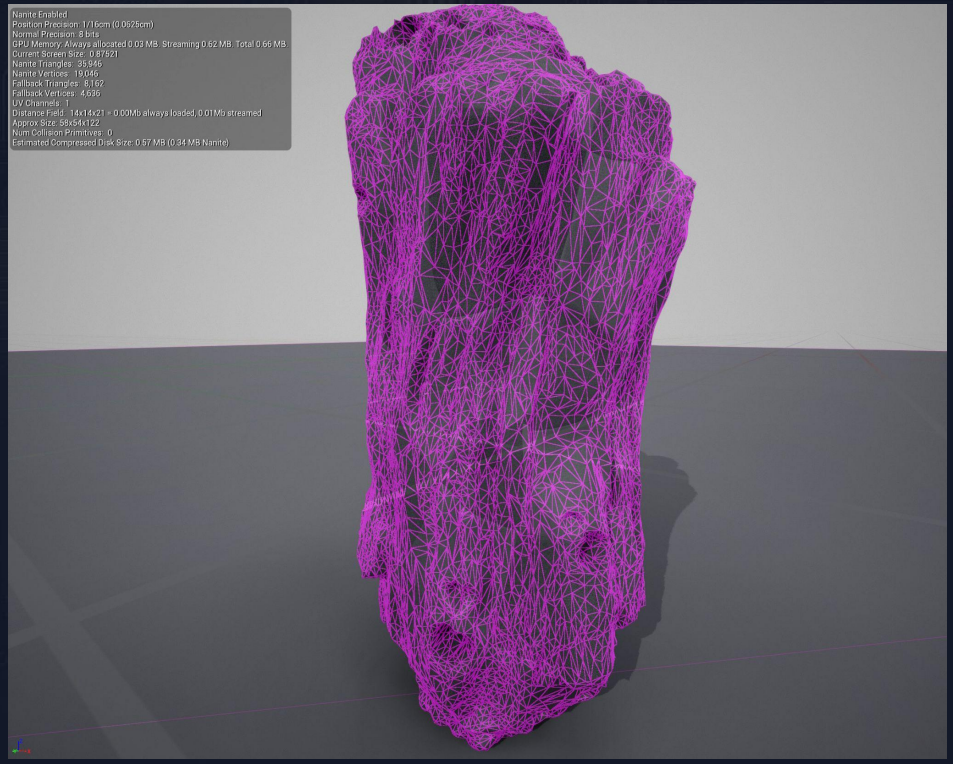

Nanite Enabled
Position Precision: 1/16cm (0.0625cm)
Normal Precision: 8 bits
GPU Memory: Always allocated 0.03 MB, Streaming 0.62 MB, Total 0.66 MB
Current Screen Size: 0.87521
Nanite Triangles: 35,946
Nanite Vertices: 19,046
Fallback Triangles: 8,162
Fallback Vertices: 4,636
UV Channels: 1
Distance Field: 14x14x21 = 0.00Mb always loaded, 0.01Mb streamed
Approx Size: 58x54x122
Num Collision Primitives: 0
Estimated Compressed Disk Size: 0.57 MB (0.34 MB Nanite)



Previous Gen

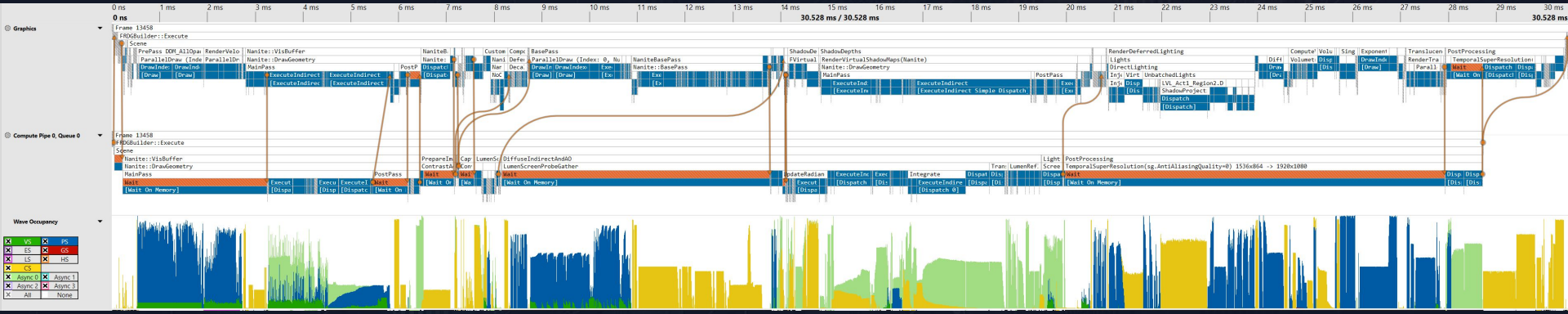
Current Gen

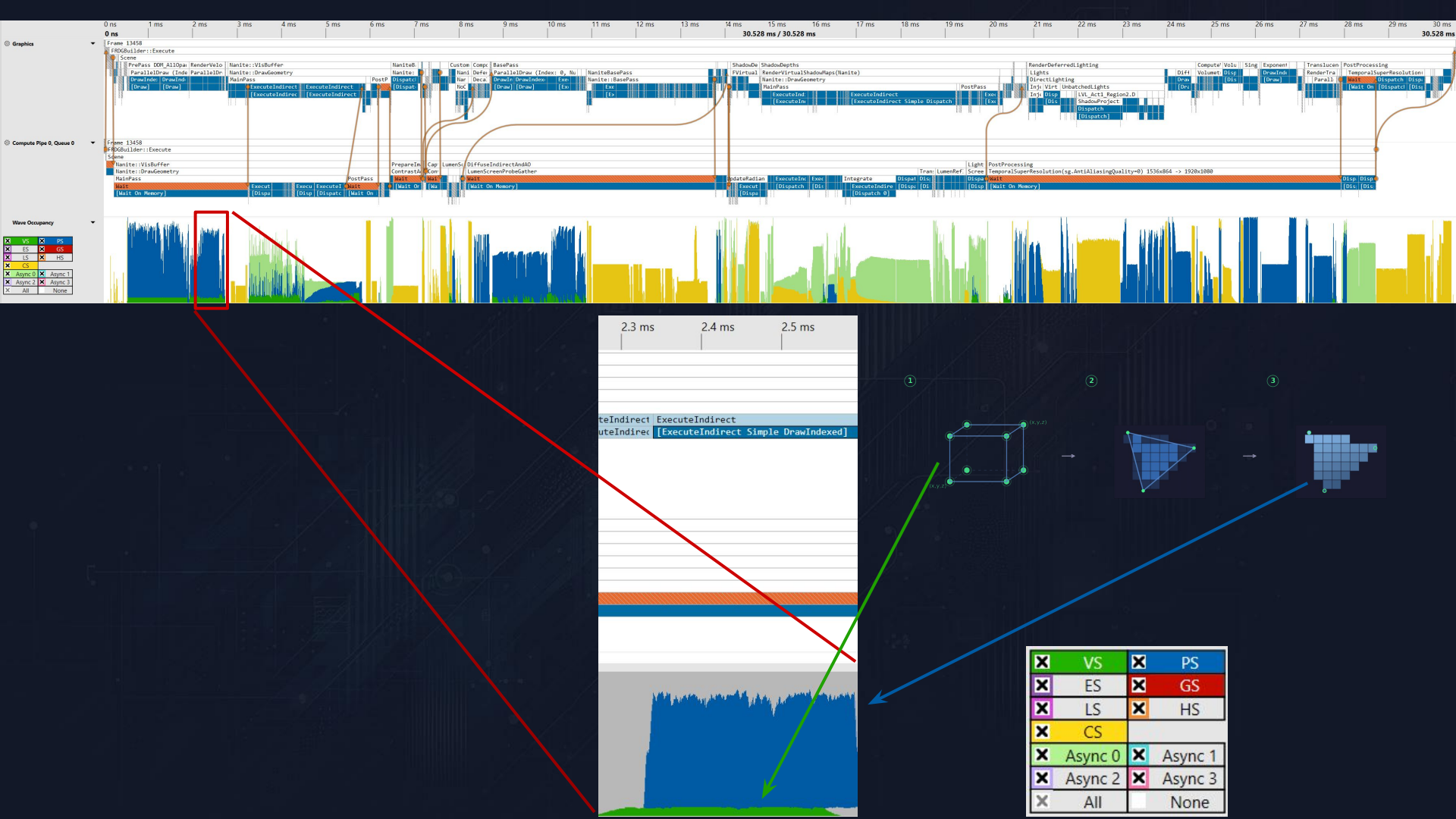


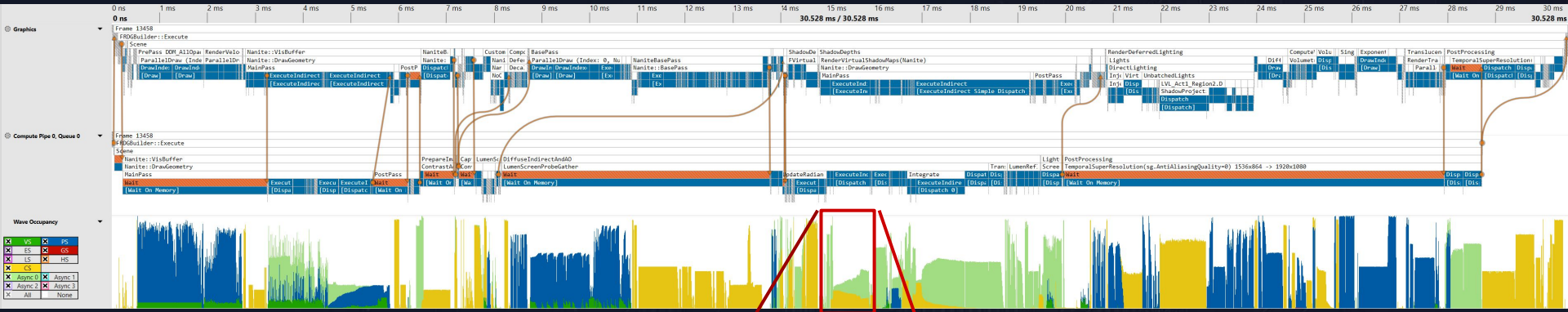


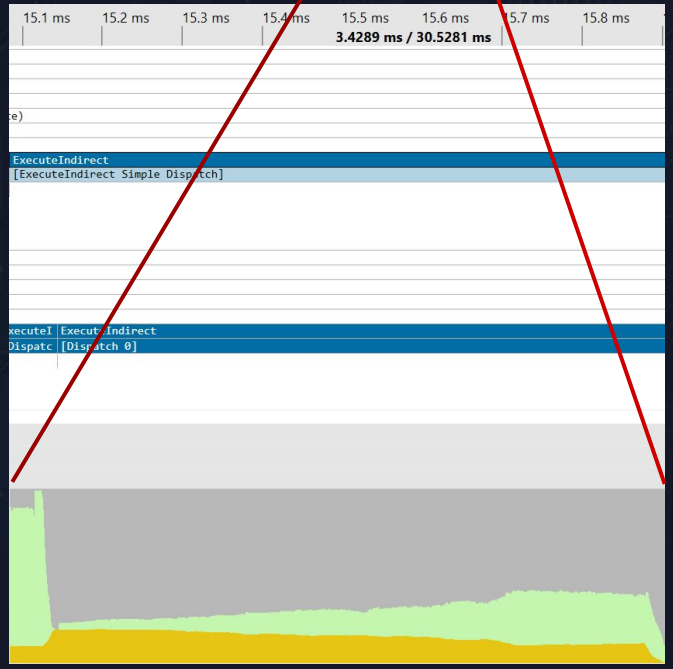

KEEPER
© Double Fine Productions


KEEPER

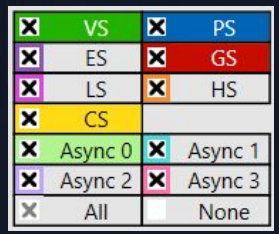

0 ns
1 ms
2 ms
3 ms
4 ms
5 ms
6 ms
7 ms
8 ms
9 ms
10 ms
11 ms
12 ms
13 ms
14 ms
15 ms
16 ms
17 ms
18 ms
19 ms
20 ms
21 ms
22 ms
23 ms
24 ms
25 ms
26 ms
27 ms
28 ms
29 ms
30 ms
0 ns
30.528 ms / 30.528 ms
30.528 ms
Graphics
Frame 13458
FRDGBuilder::Execute
Scene
Compute Pipe 0, Queue 0
Frame 13458
FRDGBuilder::Execute
Scene
Wave Occupancy
VS
PS
ES
GS
LS
HS
CS
Async 0
Async 1
Async 2
Async 3
All
None

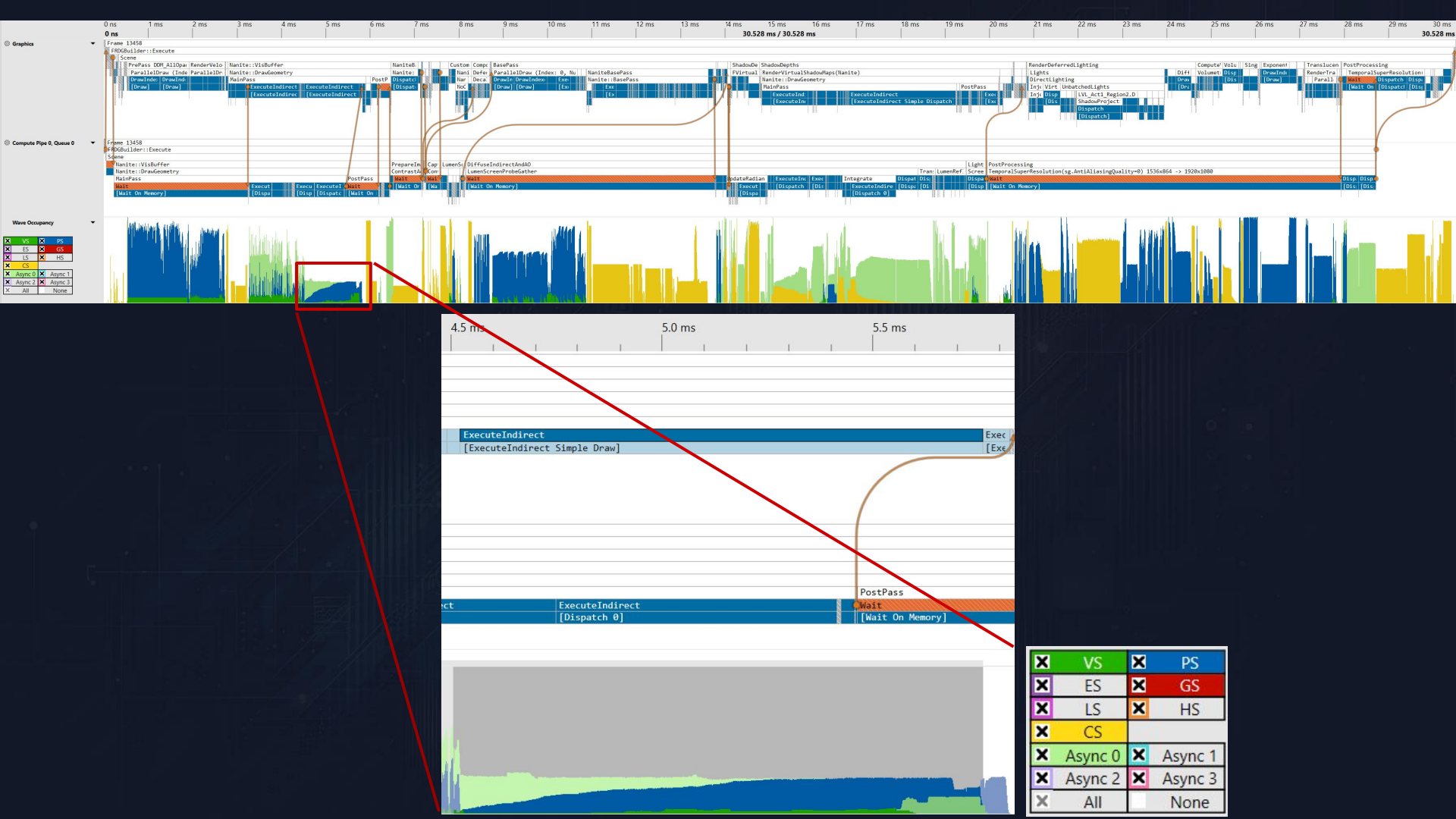

Graphics
Compute Pipe 0, Queue 0
Wave Occupancy
VS
PS
ES
GS
LS
HS
CS
Async 0
Async 1
Async 2
Async 3
All
None
2.3 ms
2.4 ms
2.5 ms
ExecuteIndirect
[ExecuteIndirect Simple DrawIndexed]


0 ns
30.528 ms / 30.528 ms
Graphics
Compute Pipe 0, Queue 0
Wave Occupancy
VS
PS
ES
GS
LS
HS
CS
Async 0
Async 1
Async 2
Async 3
All
None

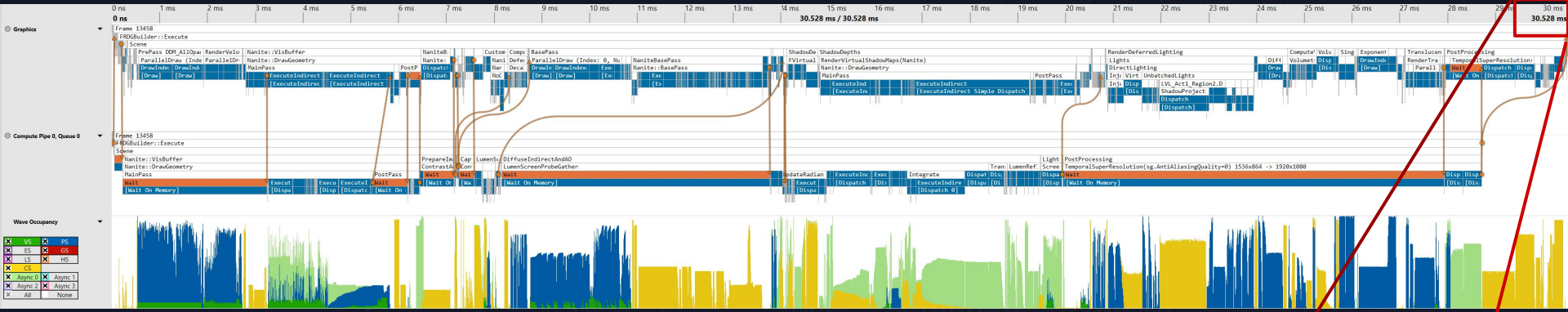

15.1 ms
15.2 ms
15.3 ms
15.4 ms
15.5 ms
15.6 ms
15.7 ms
15.8 ms
3.4289 ms / 30.5281 ms
ExecuteIndirect
[ExecuteIndirect Simple Dispatch]
[Dispatch 0]


Buffer A
Buffer B
read
Compute Shader
write
Buffer C
Buffer D
Buffer E


VS
PS
ES
GS
LS
HS
CS
Async 0
Async 1
Async 2
Async 3
All
None


Graphics
Compute Pipe 0, Queue 0
Wave Occupancy
VS
PS
ES
GS
LS
HS
CS
Async 0
Async 1
Async 2
Async 3
All
None
4.5 ms
5.0 ms
5.5 ms
ExecuteIndirect
[ExecuteIndirect Simple Draw]
PostPass
Wait
[Wait On Memory]
[Dispatch 0]


0 ns
30.528 ms / 30.528 ms
Graphics
Compute Pipe 0, Queue 0
Wave Occupancy
5.45 ms
1.5024 ms / 30.5281 ms
ExecuteIndirect
[ExecuteIndirect]
Barrier
[Draw]
PostPass
Wait
[Wait On Memory]



30 Hz

33.33 ms / frame

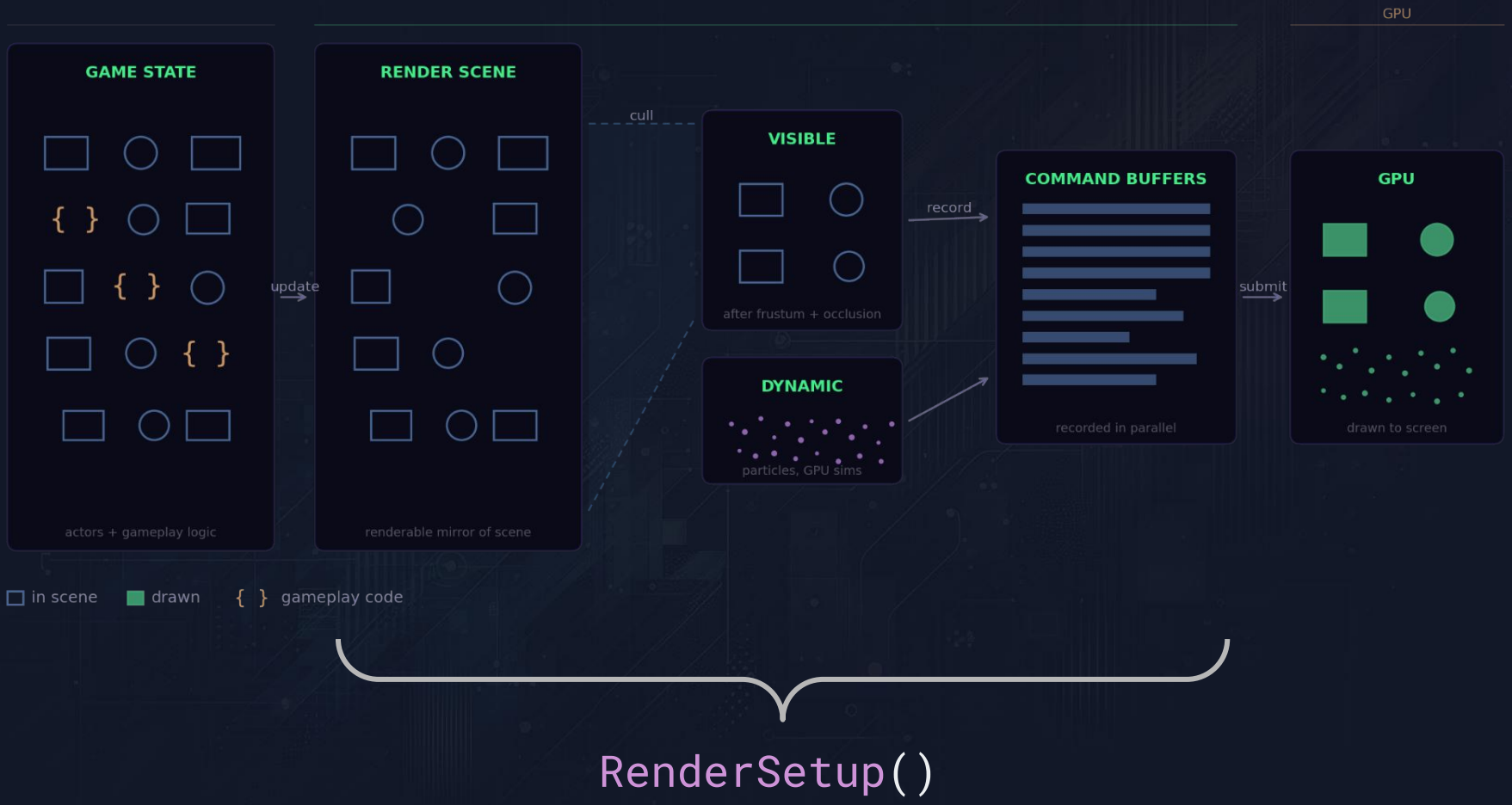
30 ms

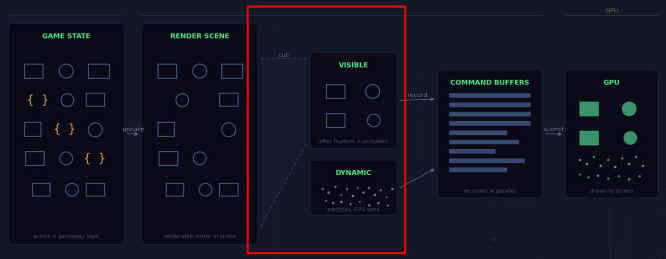
30.528 ms

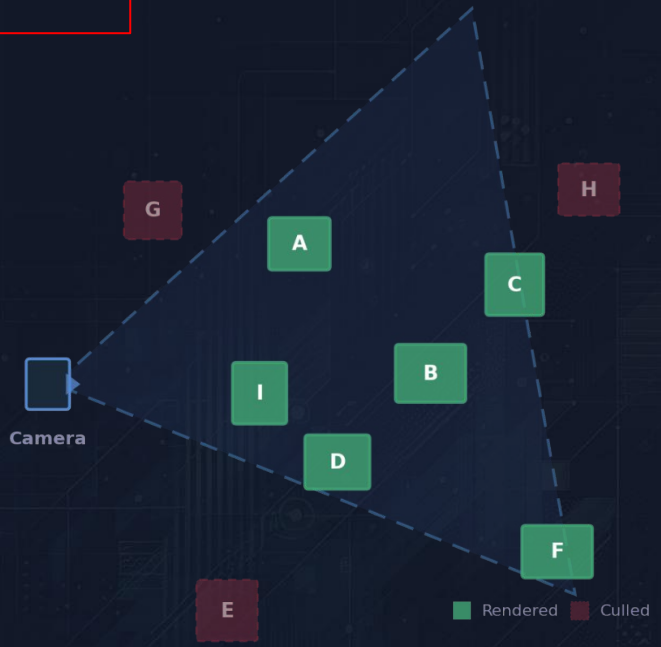

```
while True:
    const playerInput = GetPlayerInput()
    Update(playerInput)
    Render()
```

```
while True:
    const playerInput = GetPlayerInput()
    Update(playerInput)
    Render()
```

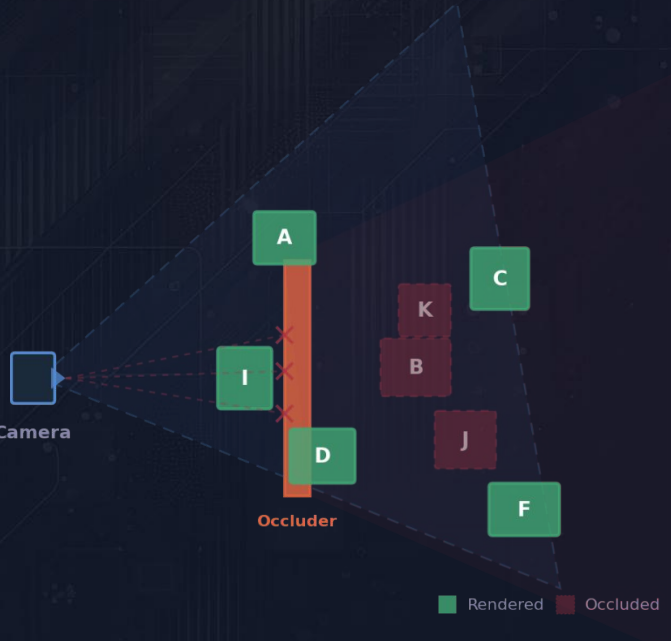
```
while True:
    const playerInput = GetPlayerInput()
    Update(playerInput)
    RenderSetup()
```


GPU
GAME STATE
actors + gameplay logic
update
RENDER SCENE
renderable mirror of scene
cull
VISIBLE
after frustum + occlusion
DYNAMIC
particles, GPU sims
record
COMMAND BUFFERS
recorded in parallel
submit
GPU
drawn to screen
in scene
drawn
{ } gameplay code
RenderSetup()



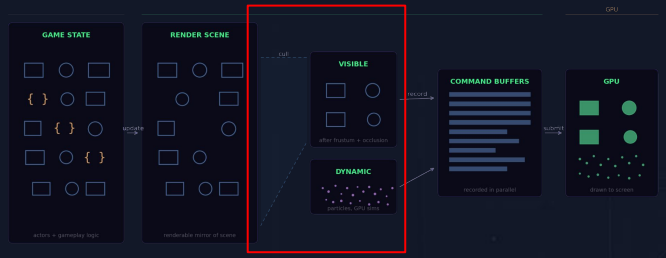


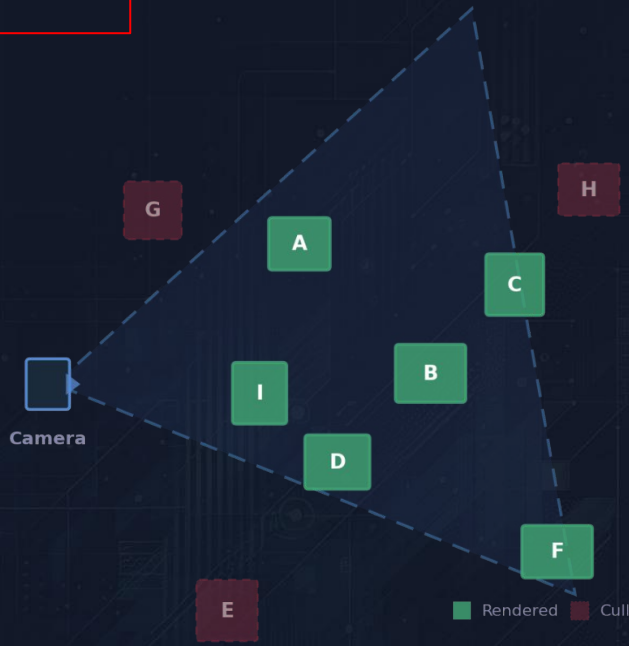
then



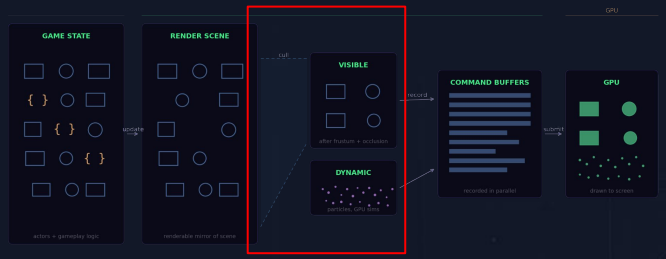
FRUSTUM CULLING

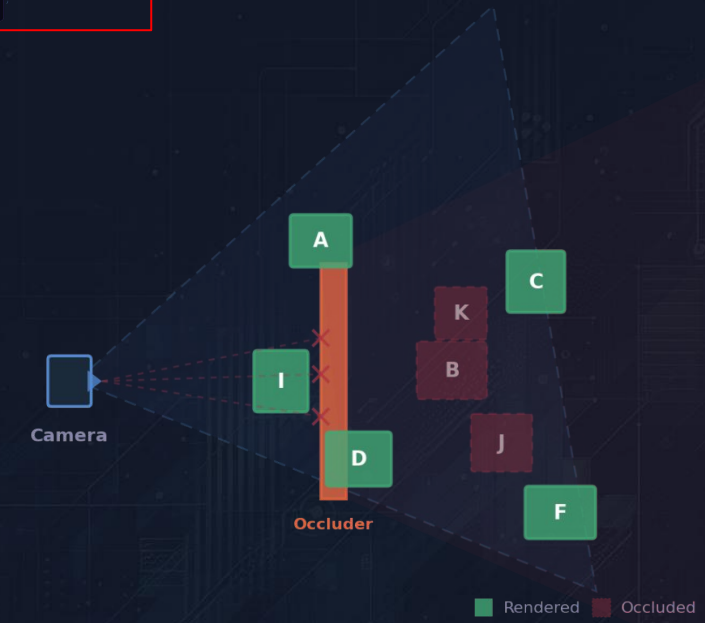
OCCLUSION CULLING

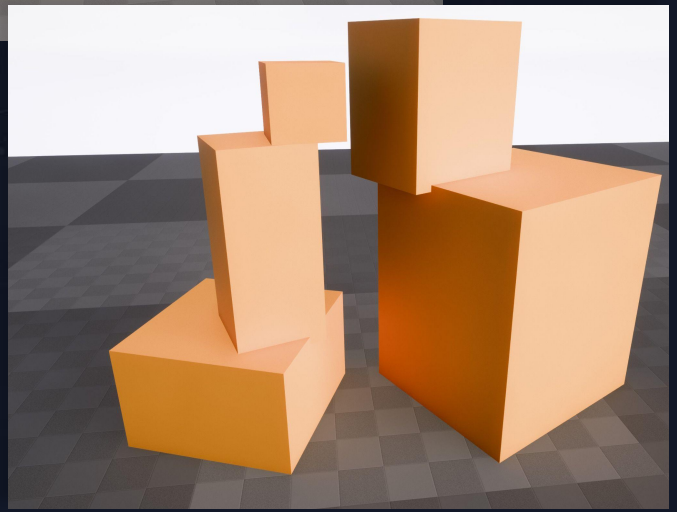




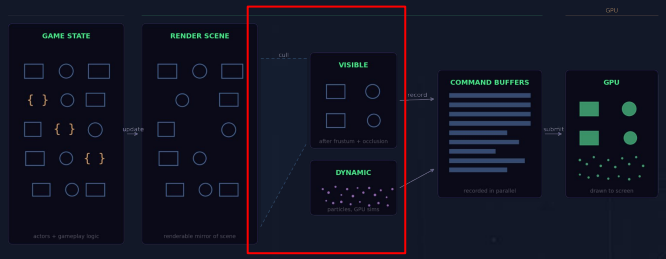
FRUSTUM CULLING

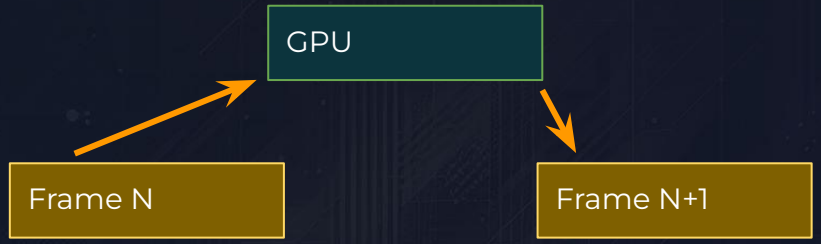


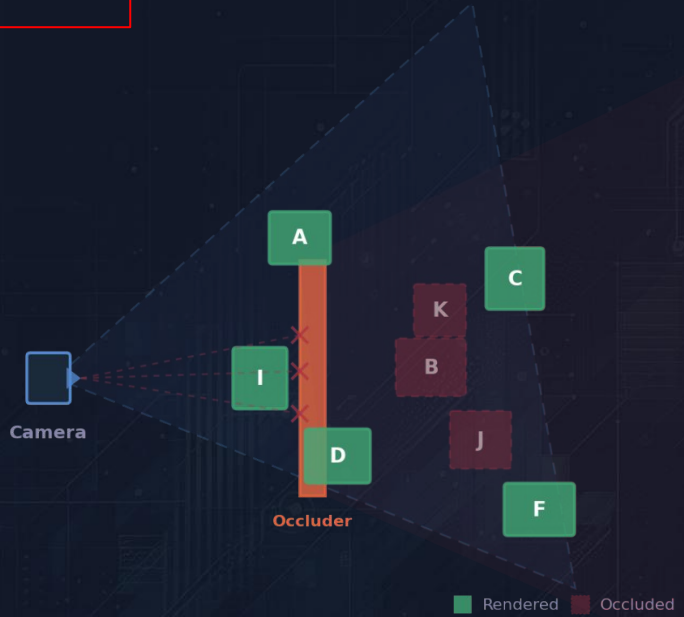


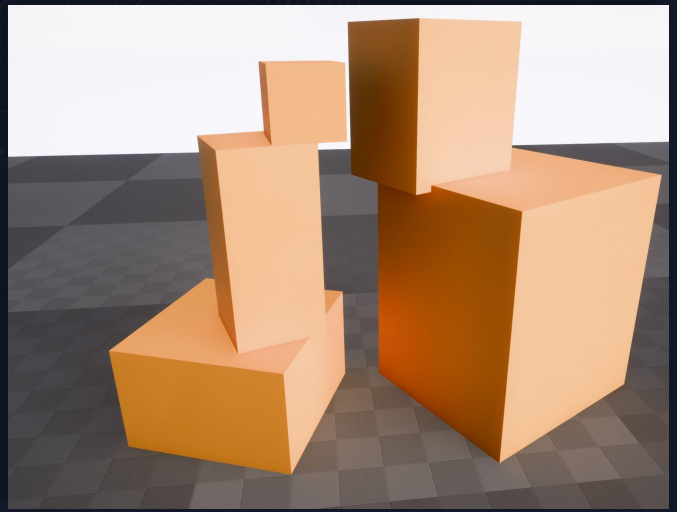


OCCLUSION CULLING

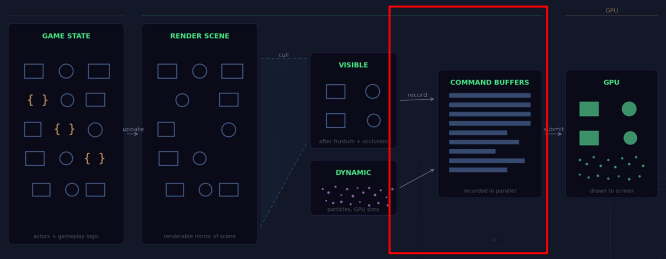




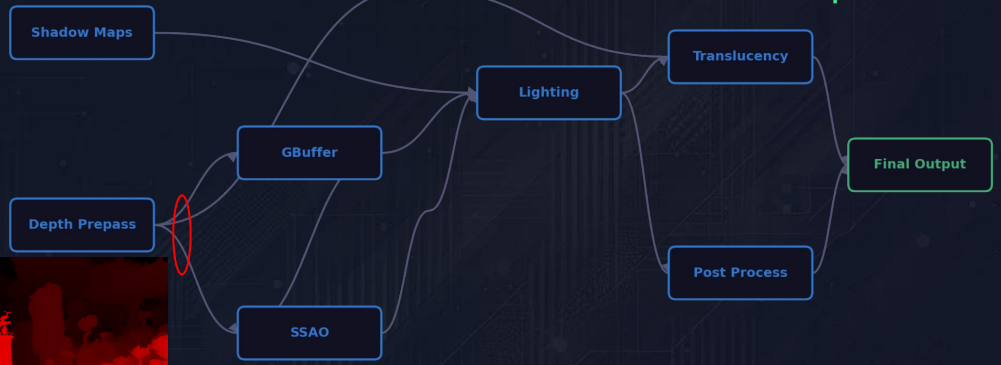




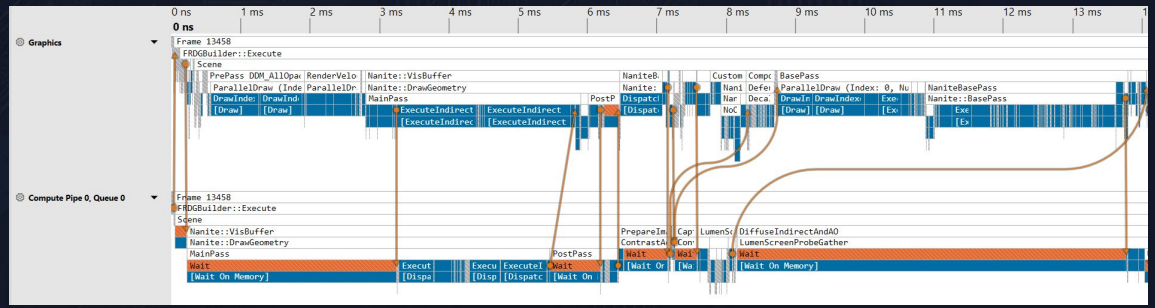
OCCLUSION CULLING

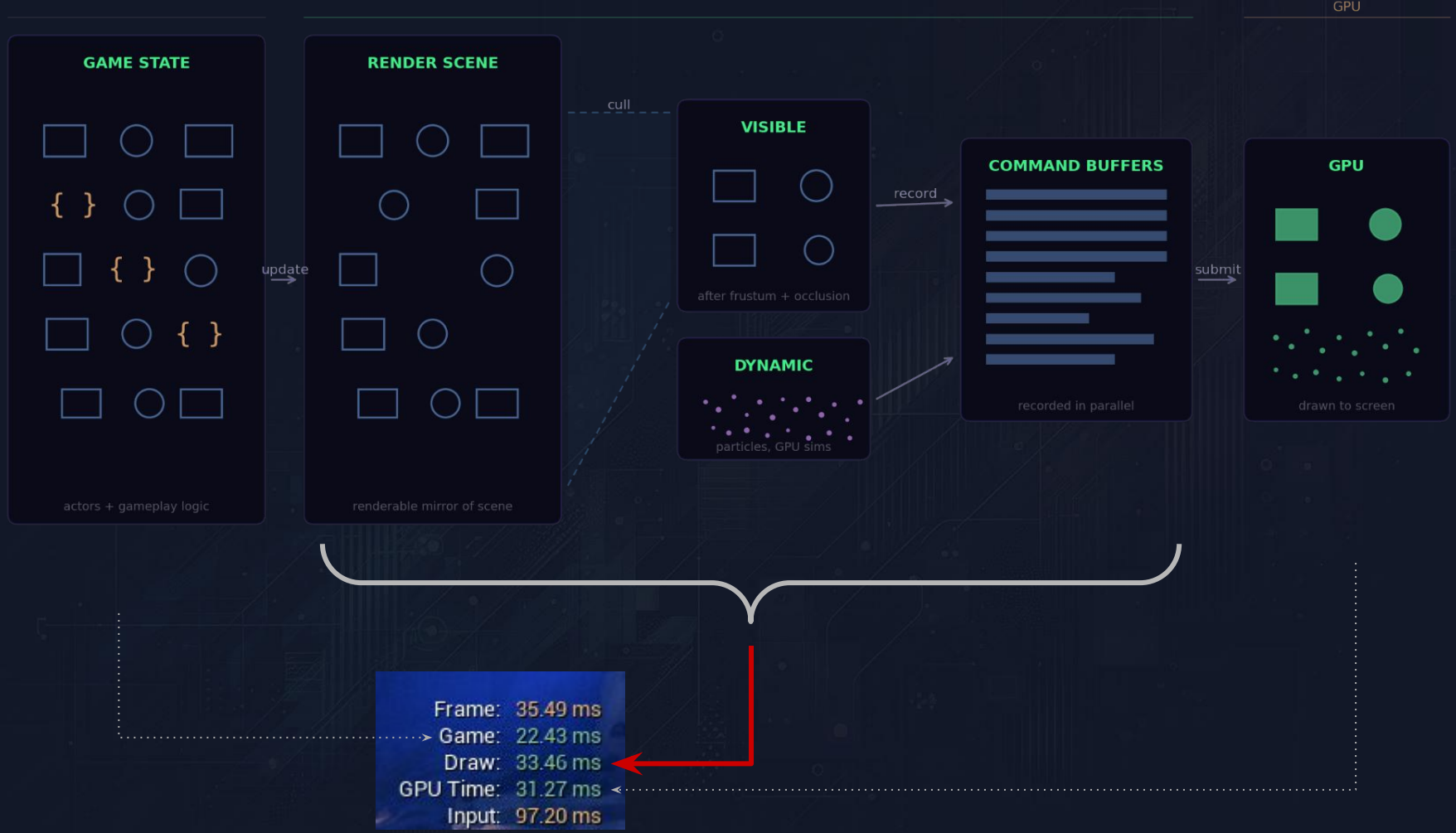


Render-Graph

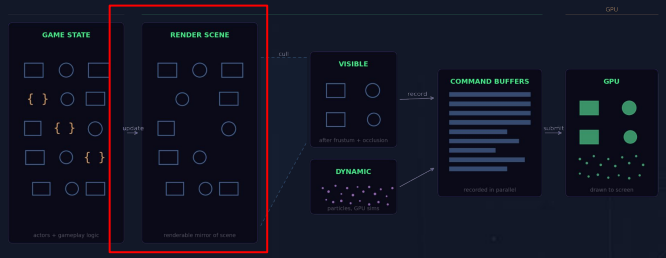


Current Render State

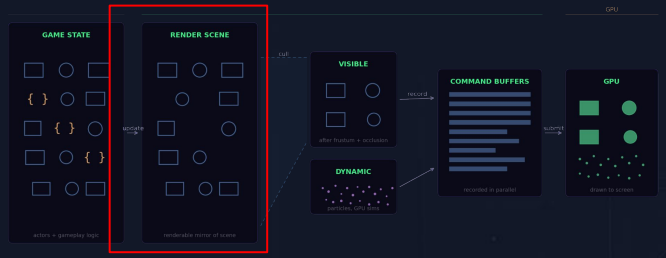



GPU
GAME STATE
actors + gameplay logic
update
RENDER SCENE
renderable mirror of scene
cull
VISIBLE
after frustum + occlusion
DYNAMIC
particles, GPU sims
record
COMMAND BUFFERS
recorded in parallel
submit
GPU
drawn to screen
Frame: 35.49 ms
Game: 22.43 ms
Draw: 33.46 ms
GPU Time: 31.27 ms
Input: 97.20 ms

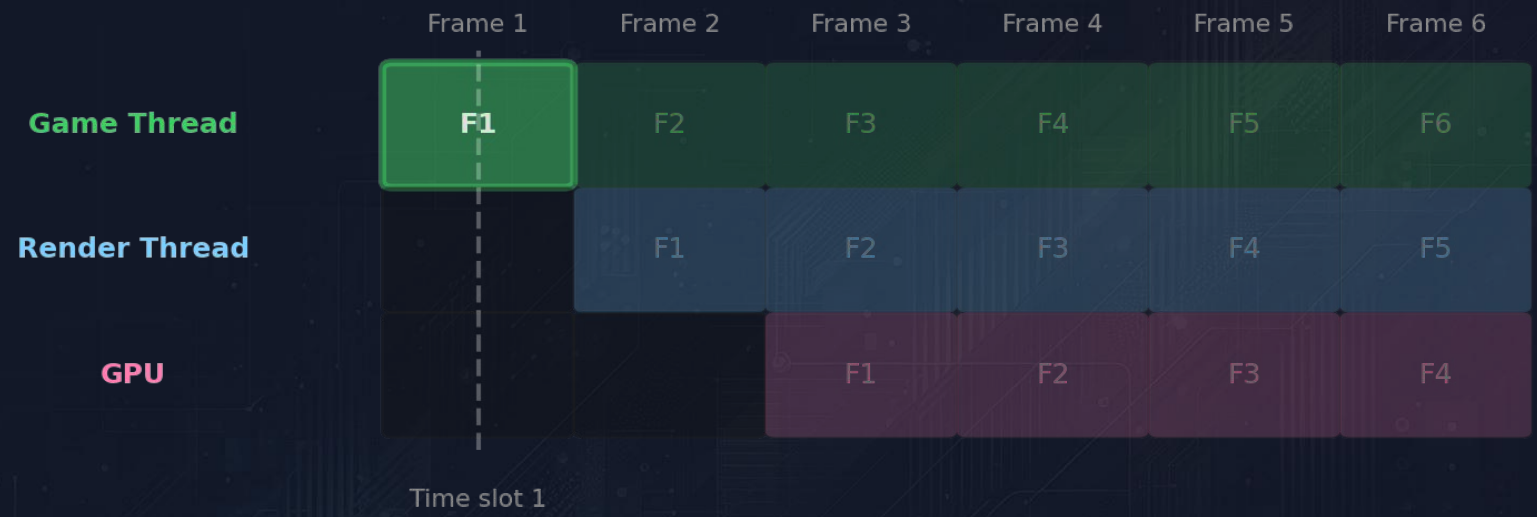
```
StartRenderThread()
while True:
    const playerInput = GetPlayerInput()
    Update(playerInput)
```

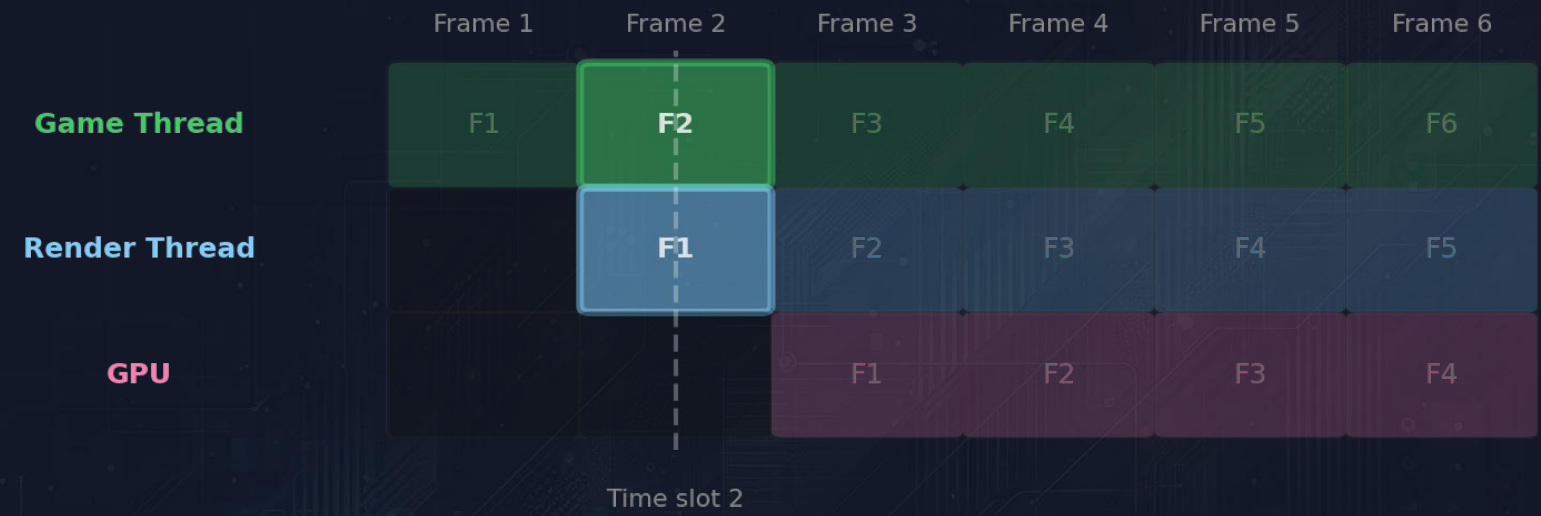


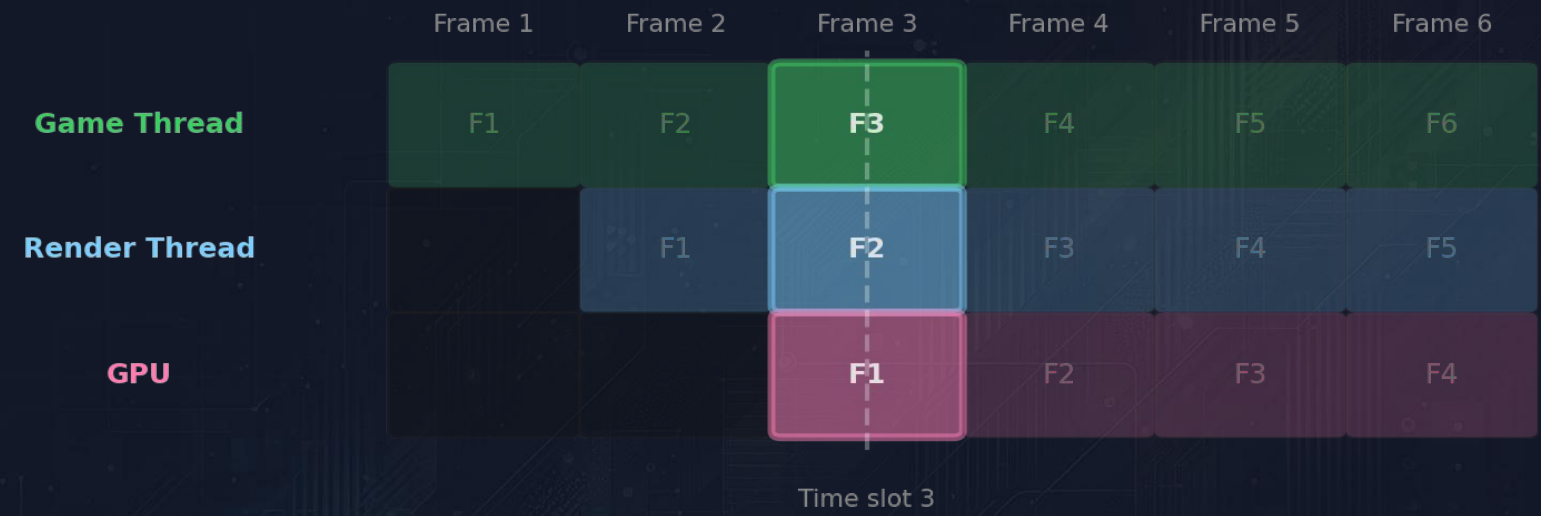

```
FLumenVirtualLightSceneInfo* LightSceneInfo = Proxy->LightSceneInfo;
ENQUEUE_RENDER_COMMAND(AddLumenVirtualLight)(
    [Scene, LightSceneInfo](FRHICommandListImmediate& RHICmdList)
    {;
        Scene->AddLumenVirtualLightSceneInfo_RenderThread(LightSceneInfo);
    });

FLumenVirtualLightSceneInfo* LightSceneInfo = Proxy->LightSceneInfo;
ENQUEUE_RENDER_COMMAND(UpdateLumenVirtualLightColorAndBrightness)(
    [LightSceneInfo, Scene, NewColor](FRHICommandListImmediate& RHICmdList)
    {
        Scene->UpdateLumenVirtualLightColorAndBrightness_RenderThread(LightSceneInfo, NewColor);
    });
```

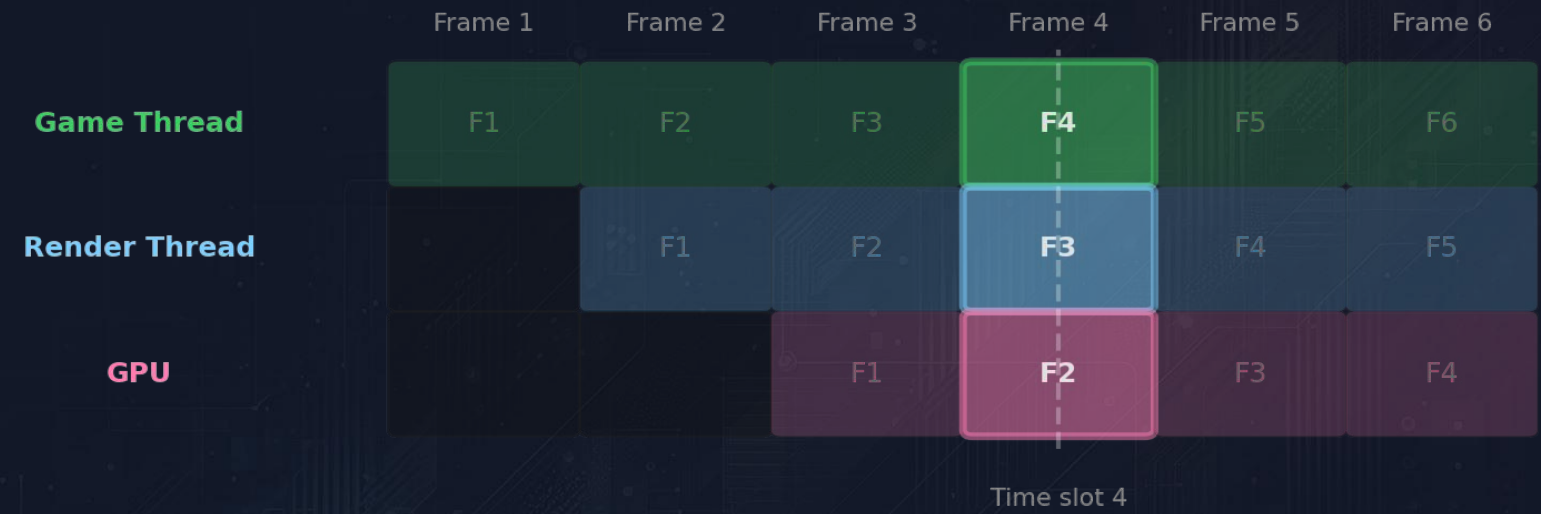


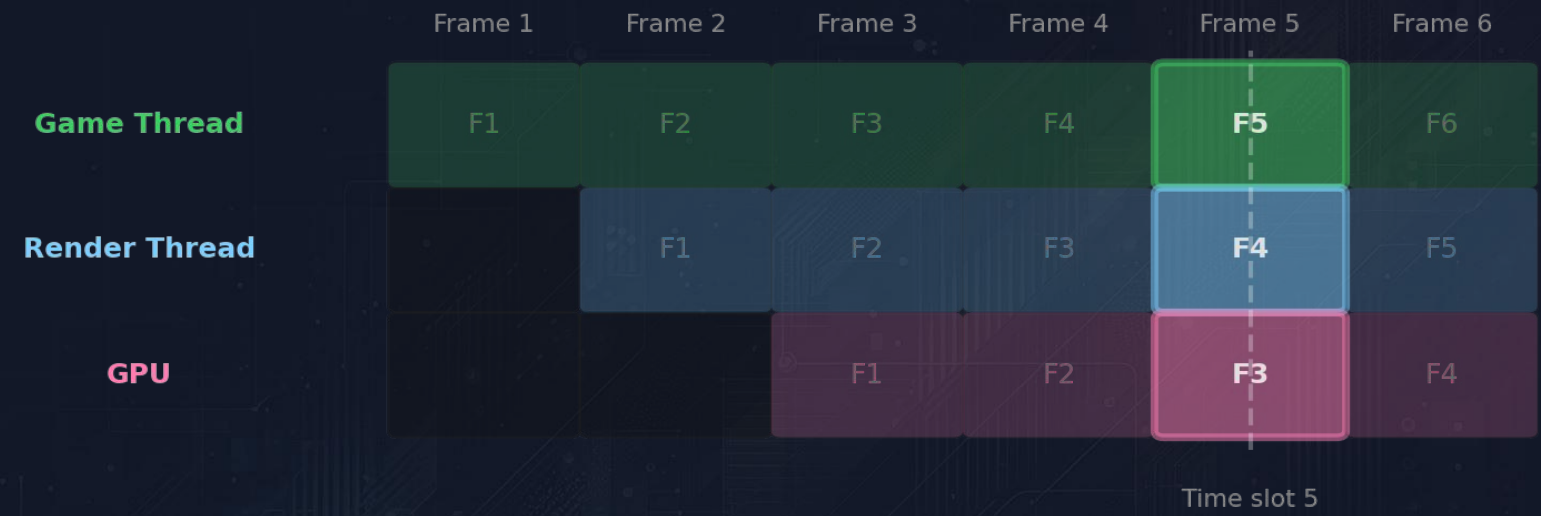

```
FLumenVirtualLightSceneInfo* LightSceneInfo = Proxy->LightSceneInfo;
ENQUEUE_RENDER_COMMAND(AddLumenVirtualLight)(
    [Scene, LightSceneInfo](FRHICommandListImmediate& RHICmdList)
    {;
        Scene->AddLumenVirtualLightSceneInfo_RenderThread(LightSceneInfo);
    });

FLumenVirtualLightSceneInfo* LightSceneInfo = Proxy->LightSceneInfo;
ENQUEUE_RENDER_COMMAND(UpdateLumenVirtualLightColorAndBrightness)(
    [LightSceneInfo, Scene, NewColor](FRHICommandListImmediate& RHICmdList)
    {
        Scene->UpdateLumenVirtualLightColorAndBrightness_RenderThread(LightSceneInfo, NewColor);
    });
```

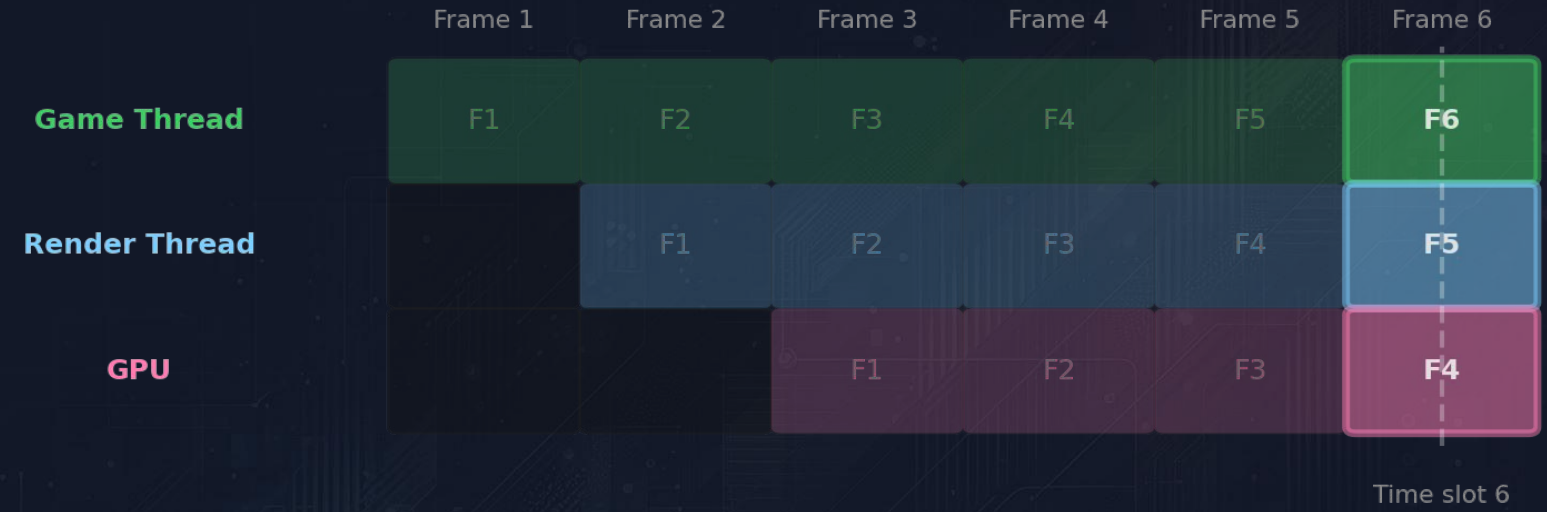

Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Game Thread
F1
F2
F3
F4
F5
F6
Render Thread
F1
F2
F3
F4
F5
GPU
F1
F2
F3
F4
Time slot 1


Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Game Thread
F1
F2
F3
F4
F5
F6
Render Thread
F1
F2
F3
F4
F5
GPU
F1
F2
F3
F4
Time slot 2


Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Game Thread
F1
F2
F3
F4
F5
F6
Render Thread
F1
F2
F3
F4
F5
GPU
F1
F2
F3
F4
Time slot 3


Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Game Thread
F1
F2
F3
F4
F5
F6
Render Thread
F1
F2
F3
F4
F5
GPU
F1
F2
F3
F4
Time slot 4


Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Game Thread
F1
F2
F3
F4
F5
F6
Render Thread
F1
F2
F3
F4
F5
GPU
F1
F2
F3
F4
Time slot 5


Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Game Thread
F1
F2
F3
F4
F5
F6
Render Thread
F1
F2
F3
F4
F5
GPU
F1
F2
F3
F4
Time slot 6

Where to go from here?

```
StartRenderThread()
while True:
    const playerInput = GetPlayerInput()
    Update(playerInput)
```

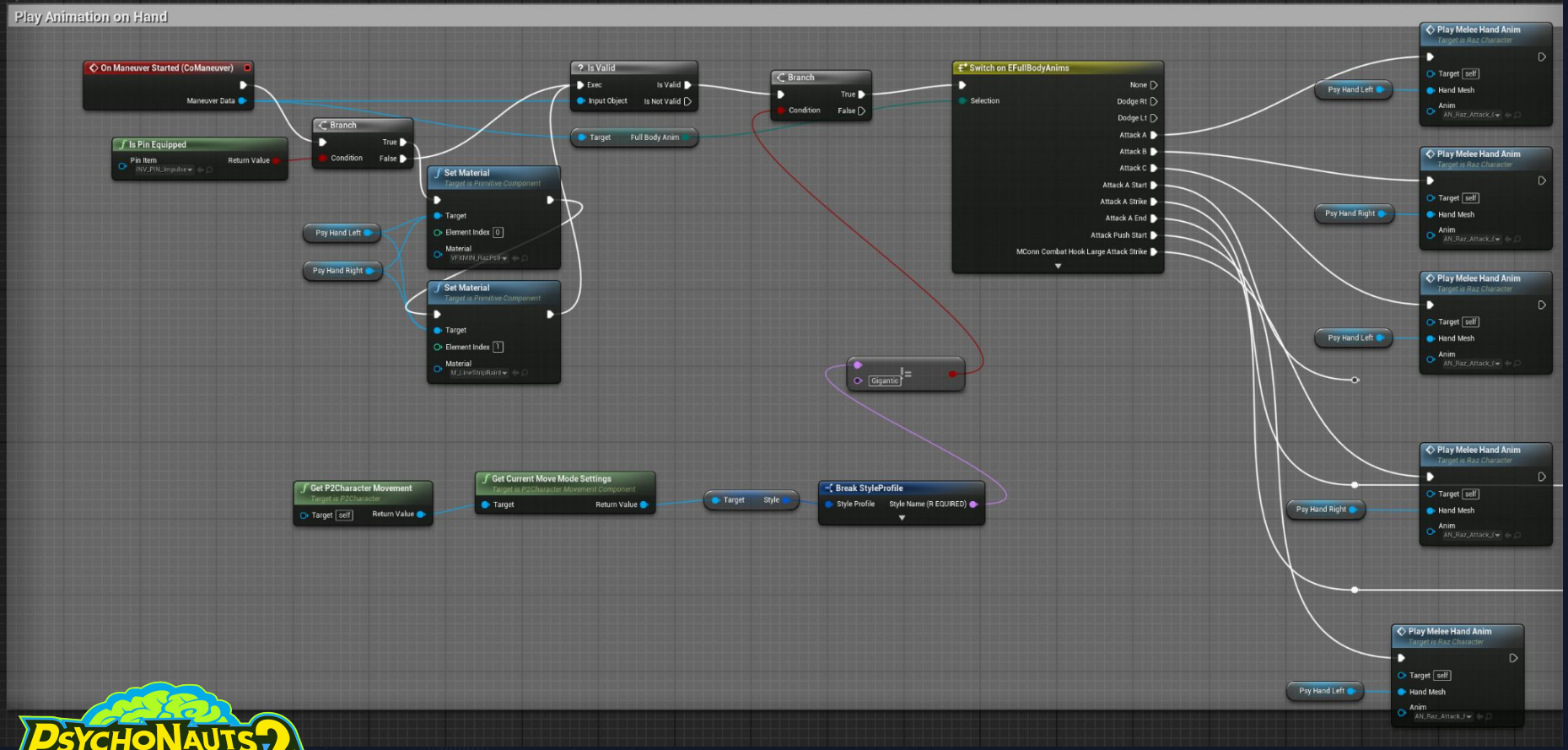
Parallelizing game-play logic...
How hard can it be?


PSYCHONAUTS 2
NOODLE BOWL
© Double Fine Productions


PSYCHONAUTS 2
NOODLE BOWL
1100658
© Double Fine Productions

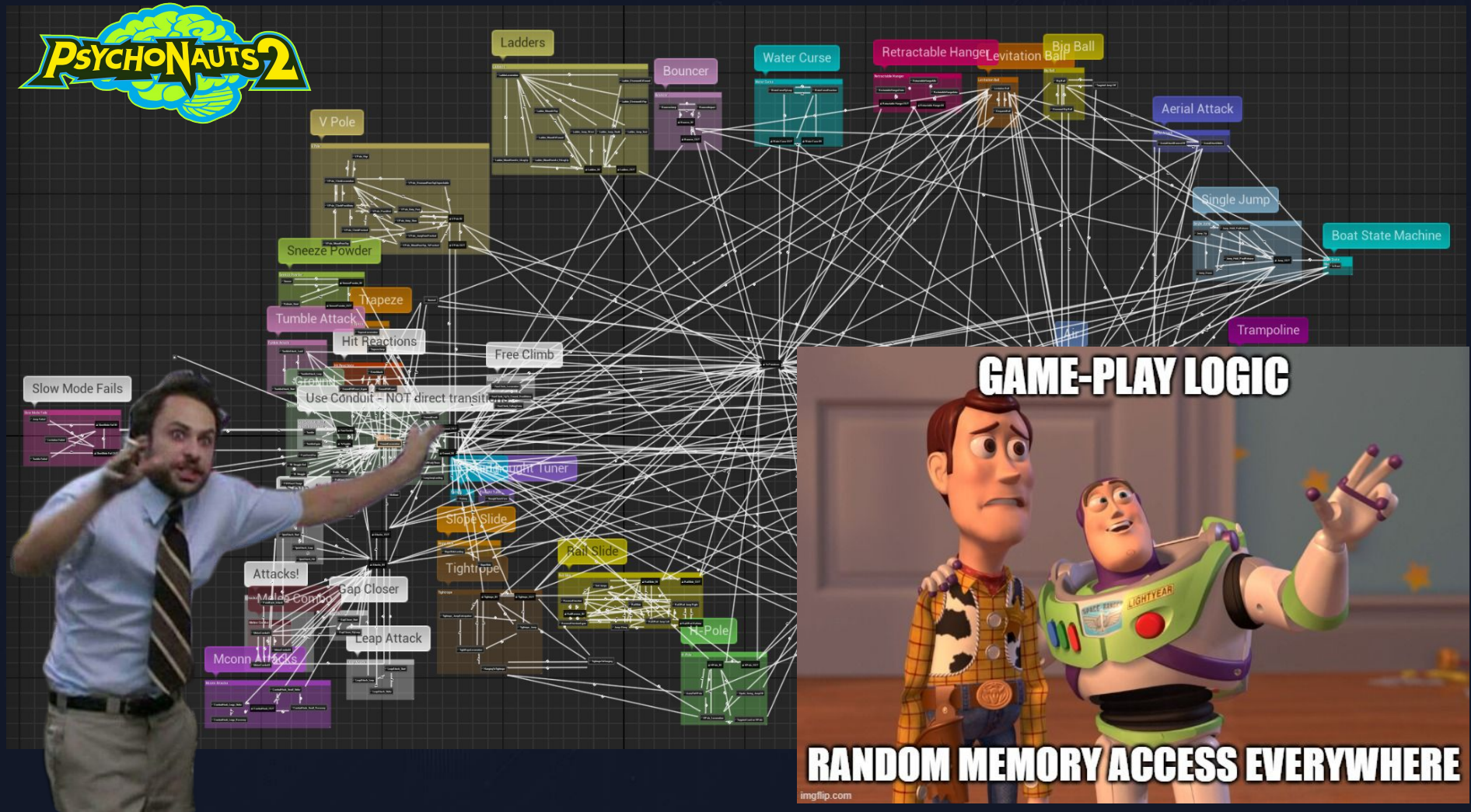

PSYCHONAUTS 2
NOODLE BOWL
1100658
© Double Fine Productions

Play Animation on Hand

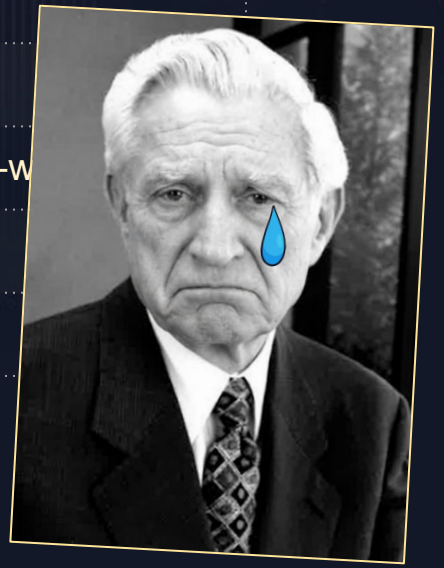


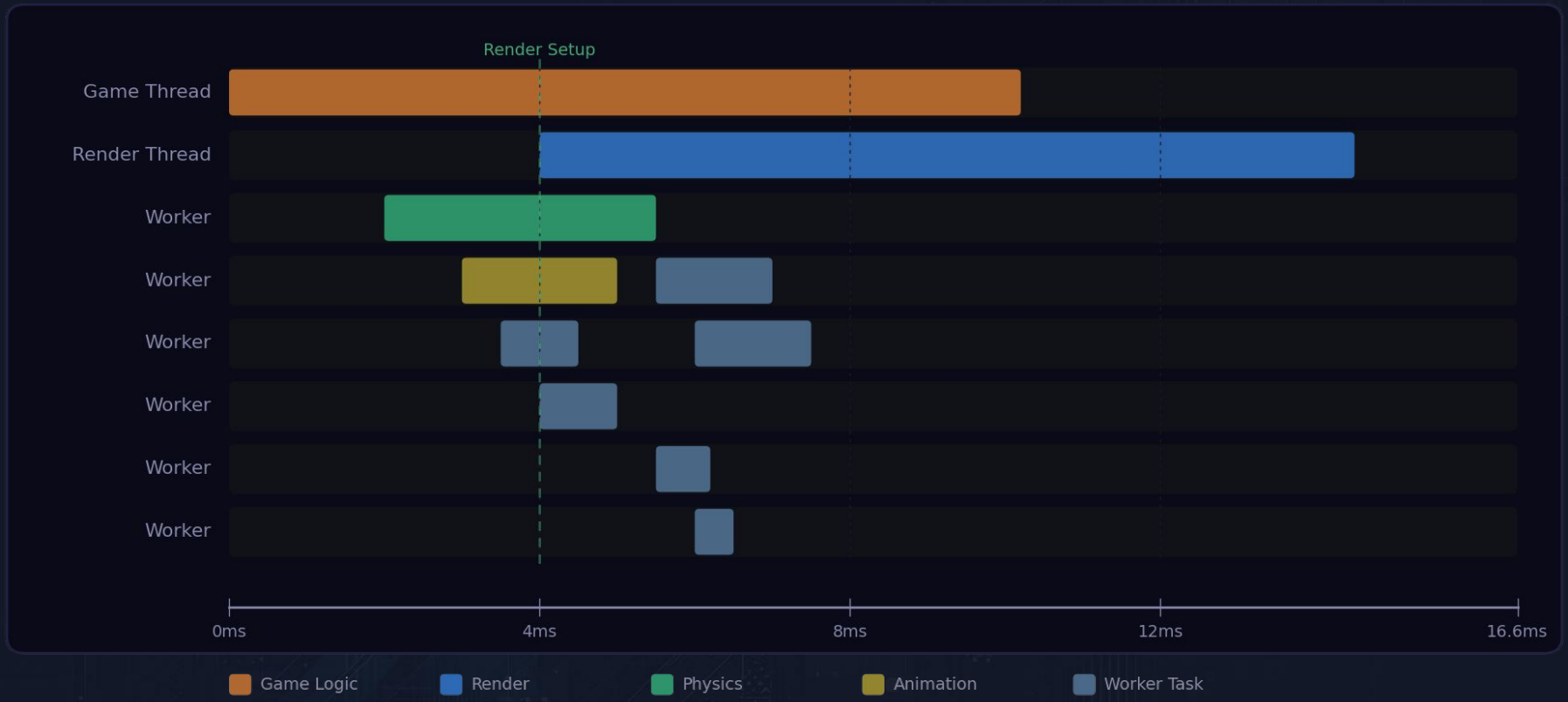
PSYCHONAUTS 2

Blueprints: 6316
Functions: 26912


PSYCHONAUTS 2
Ladders
Bouncer
Water Curse
Retractable Hanger
Levitation Ball
Big Ball
Aerial Attack
Single Jump
Boat State Machine
Trampoline
V Pole
Sneeze Powder
Trapeze
Tumble Attack
Hit Reactions
Free Climb
Slow Mode Fails
Use Conduit - NOT direct transition
Slope Slide
Rail Slide
Tightrope
Attacks!
Gap Closer
Leap Attack
H-Pole
GAME-PLAY LOGIC
RANDOM MEMORY ACCESS EVERYWHERE
imgflip.com

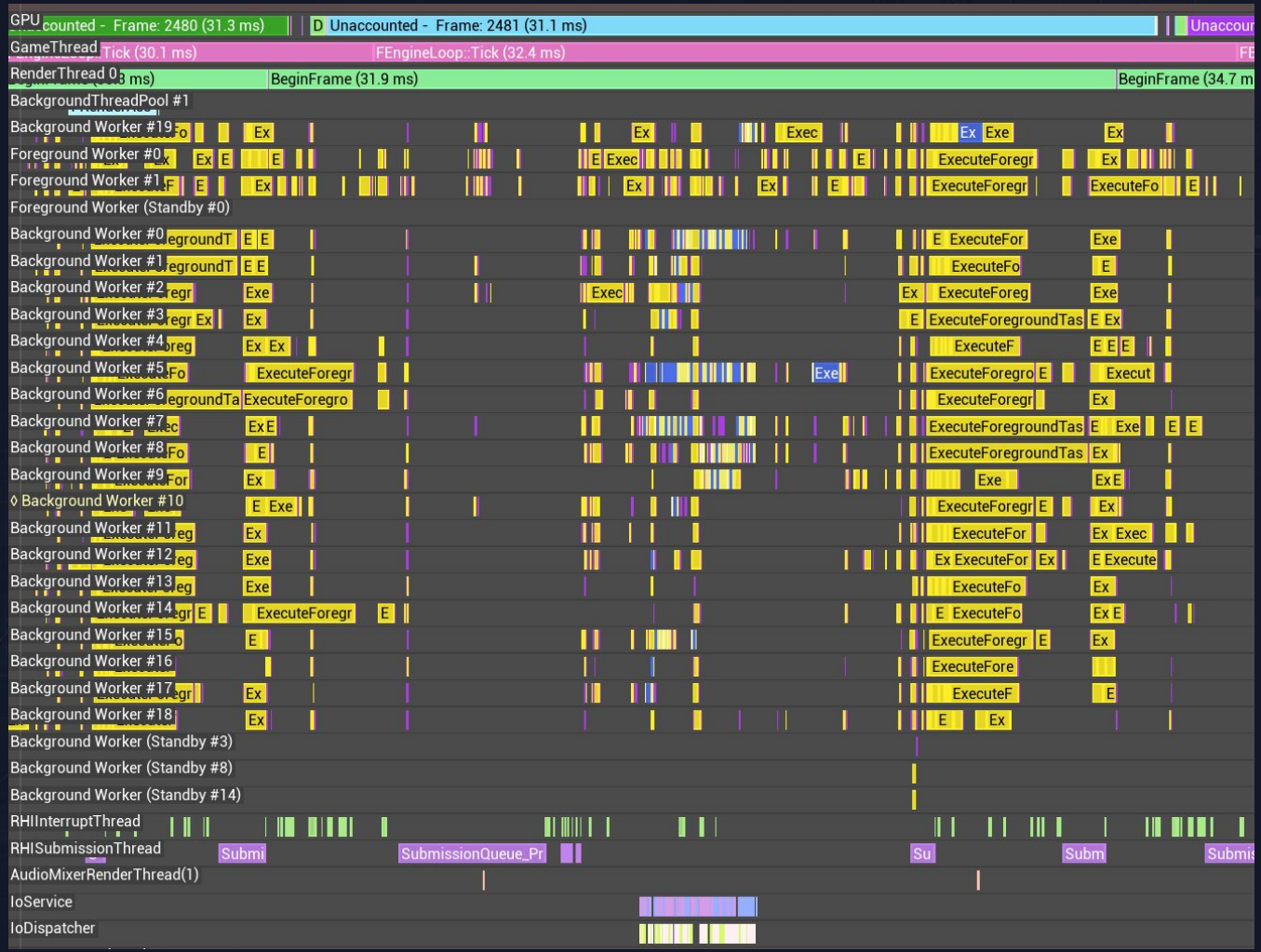
	Graphics / Compute	Game-play Logic
Data Layout	Contiguous, aligned buffers	Scattered across heap
Access	Predictable	Random
Operation per Element	Same	Different
Branching	Minimal	Heavy
State mutation	RO input → WO output	Read-modify-w
Dependencies	None	Complex
Execution order	Doesn’t matter	Matters




Render Setup
Game Thread
Render Thread
Worker
Worker
Worker
Worker
Worker
Worker
0ms
4ms
8ms
12ms
16.6ms
Game Logic
Render
Physics
Animation
Worker Task

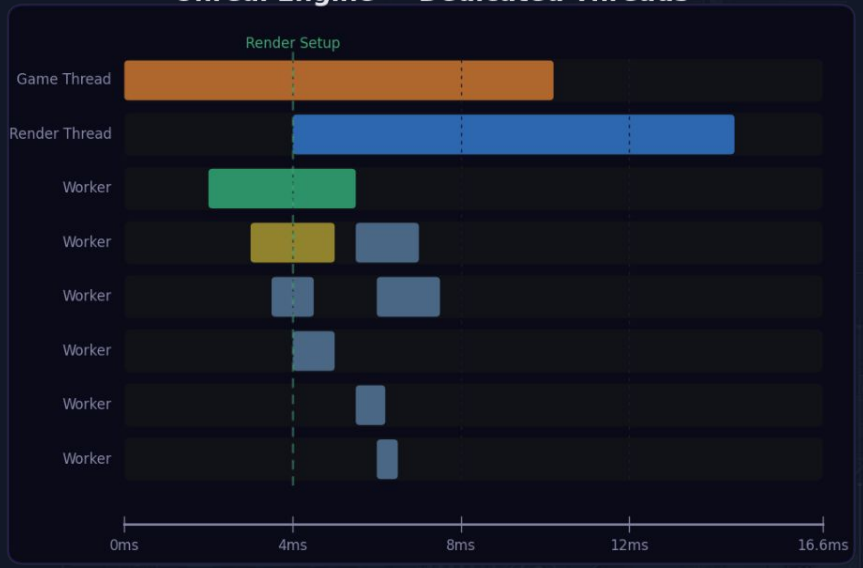
KEEPER

Unaccounted - Frame: 2556 (24.1 ms)
NaniteVisB
Lumen
Basepass (1.3
ShadowDepths (1.5
RenderDeferredLighting (11.1 ms)
Postprocessing (3.5 ms)
NaniteBasePa
LumenScreenProbeGather (7 ms)
LumenRefl
Lights (2.3 ms)
TemporalSuperResolution (2.9 ms)
ShadowPr
FEngineLoop::Tick (25.3 ms)
Frame (25.3 ms) - 2558
FrameTime (25.2 ms)
GameEngine Tick (18.4 ms)
Frame Sync Time (2.1 ms)
UWorld_Tick (16.9 ms)
RedrawVie
GameThreadWaitForTask
World Tick Time (16.9 ms)
FViewport
Game thread idle time (2.1
Sequence Ti
Queue
Tick Time (14.5 ms)
Runner Flu
TG_PrePhysics (10.5 ms)
Start TG_DuringPhysics (2.5 ms
TG_Post
ReleaseTickGroup Block (10.2 ms)
TickCompletionEvents (10.2 ms)
WaitUntilTasksComplete (10.1 ms)
ExecuteT
FActorCo
CharMov
UCharact
Char Mov
Char Tick
Char Non
Char Perf
USkelet
AnimGa
Anim Ti
Update
Lightho
Bluep P
Bluep L
Bluep
Execu
IKFo
BeginFrame (25.7 ms)
FDrawSceneCommand (23.9 ms)
RenderViewFamily (23.7 ms)
Scene::UpdateAllPrimitiveSceneInfos (7.8 ms)
FDeferredShadingSceneRenderer_Render (6.9 ms)
FRDGBuilder::Execute (8.4 ms)
FScene_UpdateAllPrimitiveSceneInfos (7.8 ms)
Scene (6.9 ms)
Collect Resource
FTaskBase::WaitImpl_StateChangeEvent_WaitFor (5.2 ms)
UpdatePrimitive (7.8 ms)
VisibilityCo
RenderDeferredLight
UpdateAllPrimitiveSceneInfos (7.8 ms)
FDeferredS
DeferredShadingSce
LaunchVisibilityTasks (6.8 ms)
InitViews
RenderLighting (1.6
MapOcclusionResults (6.8 ms)
STAT_MapHZBResults (6.8 ms)
STAT_RHIMETHOD_MapStagingSurface_Flush (6.8 ms)
RHICmdList_Submit (6.8 ms)
RHICmdList_FlushRHIThread (6.7 ms)
Explicit wait for RHI thread (6.7 ms)
WaitUntilTasksComplete (6.7 ms)
WaitForTasks (6.7 ms)


GPU
Unaccounted - Frame: 2480 (31.3 ms)
Unaccounted - Frame: 2481 (31.1 ms)
GameThread
Tick (30.1 ms)
FEngineLoop::Tick (32.4 ms)
RenderThread 0
BeginFrame (31.9 ms)
BeginFrame (34.7 m
BackgroundThreadPool #1
Background Worker #19
Foreground Worker #0
Foreground Worker #1
Foreground Worker (Standby #0)
Background Worker #0
Background Worker #1
Background Worker #2
Background Worker #3
Background Worker #4
Background Worker #5
Background Worker #6
Background Worker #7
Background Worker #8
Background Worker #9
Background Worker #10
Background Worker #11
Background Worker #12
Background Worker #13
Background Worker #14
Background Worker #15
Background Worker #16
Background Worker #17
Background Worker #18
Background Worker (Standby #3)
Background Worker (Standby #8)
Background Worker (Standby #14)
RHIInterruptThread
RHISubmissionThread
AudioMixerRenderThread(1)
IoService
IoDispatcher


Axel Gneiting
@axelgneiting
Fun fact: Doom Eternal does not have a main or render thread. It's all jobs with one worker thread per core.
DOOM
ETERNAL
© id Software

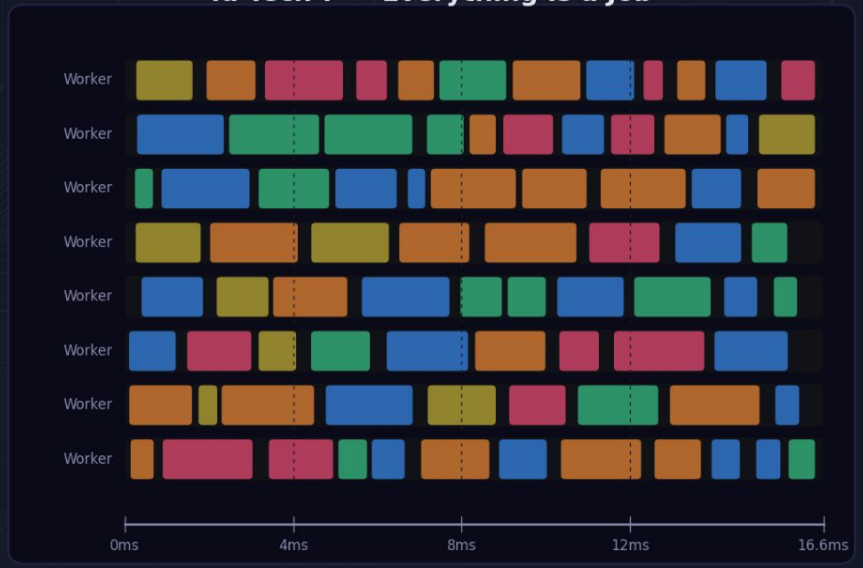
Unreal Engine — Dedicated Threads



Predictable pipeline: Game Thread → Render Thread → GPU. Workers handle leaf tasks.

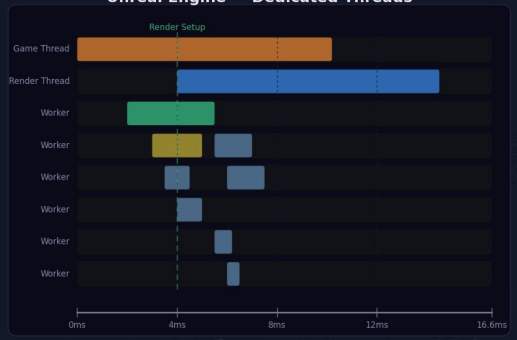
vs

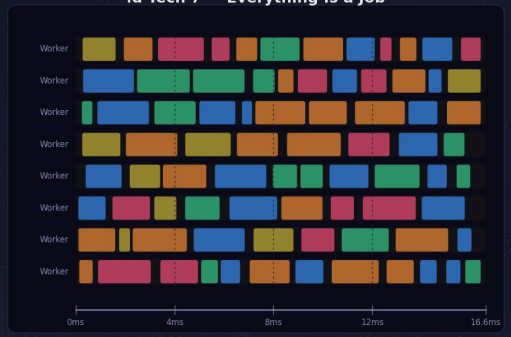
id Tech 7 — Everything Is a Job

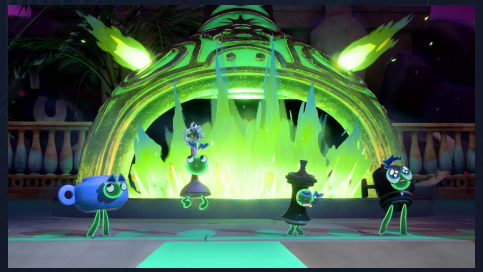


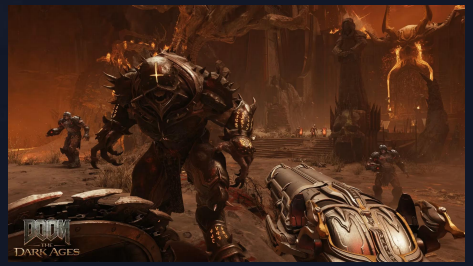
No dedicated threads. ~100 jobs per frame distributed across all cores.

Game Logic · Render · RHI / Submit · Physics · Animation · Audio









Tha n k you!

p1xelcoder.bsky.social