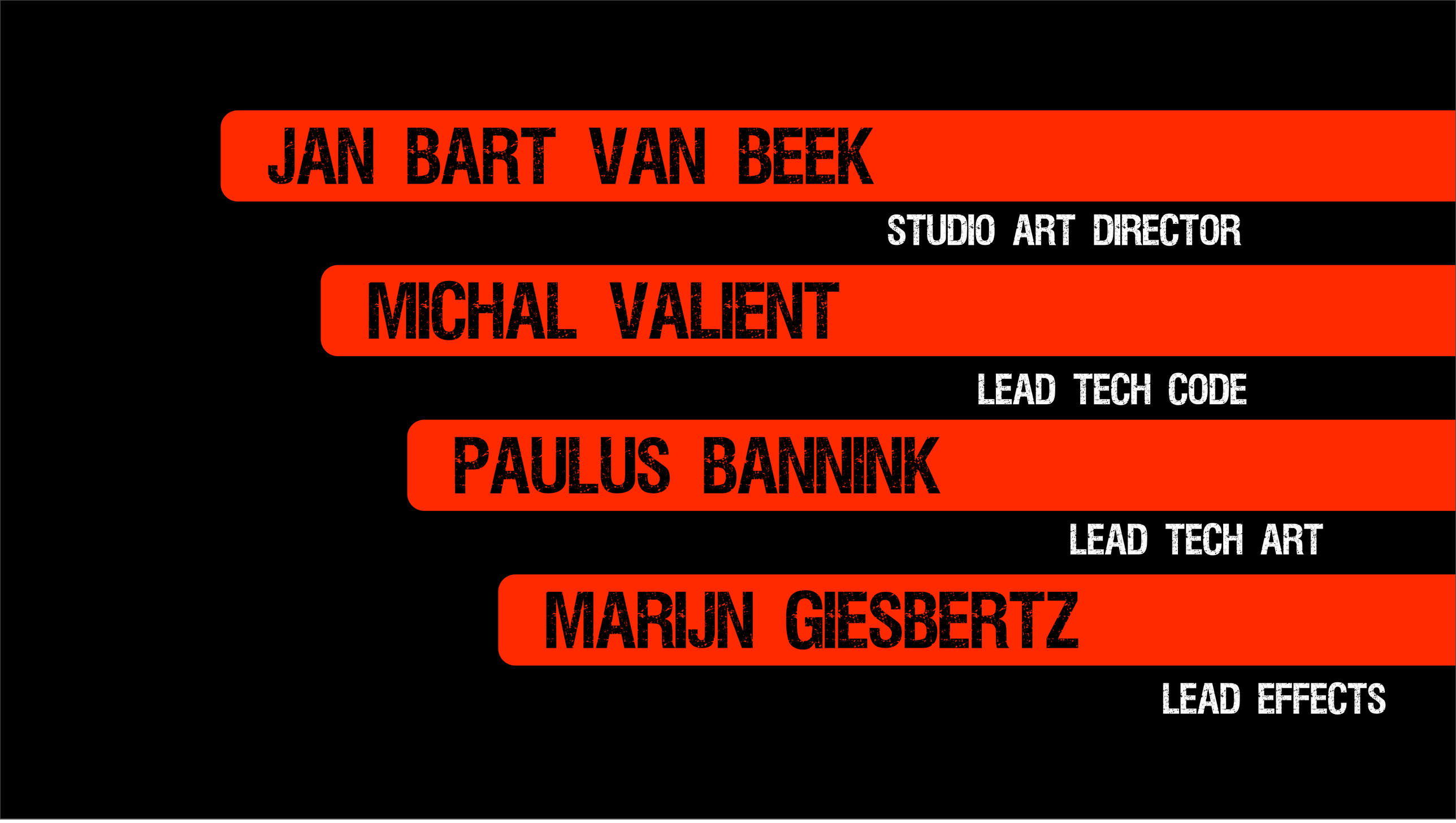
Hello and welcome to this production session
about the making of Killzone 3



My name is Jan-Bart van Beek.
I'm the Studio Art Director
at Guerrilla Games in Amsterdam.

I'm joined on the stage today by Michal, Paulus and Marijn.

And as you can see by their **outlandish** surnames,
we'll be giving this presentation
in a variety of **odd** european accents.

It’s quite an honour to be standing here at Siggraph
and present one of the production session.

Over the last decades
these session have **routinely** been dominated
by such giants as Pixar, ILM, Weta, Dreamworks and Digital Domain.

But it’s only **since** **very recent** that the creators
of videogames have stood here to present their work.

Just 2 years ago Epic Games was the first **gamestudio**
on this stage **when** they presented their work on Gears of War 2.

That was followed last year by one of my collegues at from Sony Santa Monica
.... Ken Feldman...
when he and his team presented God of War 3.

For those of you that don’t know
much about Guerrilla Games,

We were founded in 1999 and have
been a part of Sony Computer Entertainment since 2005.

We’ve been creating games for **more then** a decade now.

Or more precisely,
we been making **one game** for a whole decade.

The Killzone Franchise.

And as you might be able to tell
from our cover art...it's a game
about
shooting
red-**eyed space-nazis**
in the face.

now...10 years is a very long
in video game time

We've seen three generation of consoles arrive onto the market.
And within that time we've seen enormous jumps in technology.

Probably the best way to illustrate
just how much the
games industry has matured
is to
show you what the very first footage of
what Killzone looked like

The following is a snippet
from the 1999
Killzone demo
we gave to Sony.

about 12 years later the third installement of Killzone looks like this.

The jump from PS2 to PS3 has been a very large one.
The hardware became...10 to a 100 times faster.

We went from SD to HD.
From textures to pixel shaders.

From a single static lightsource
to dozens of dynamic shadow casting lights

we went from tens of thousands of polygons
, to hundreds of thousands.

The jump is even bigger when you look at a game like Metal Gear Solid.
one of the few games that has seen every generation of playstation console.

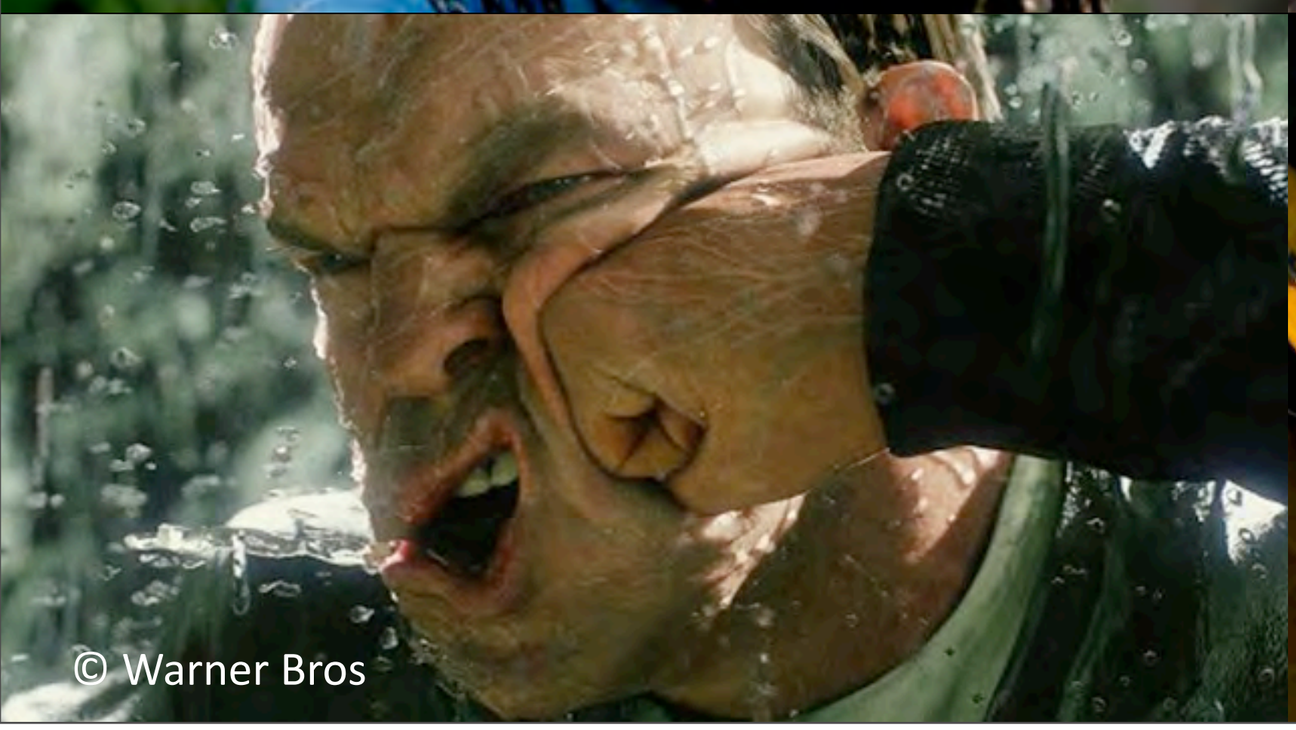
There is exactly 10 years between these two images.

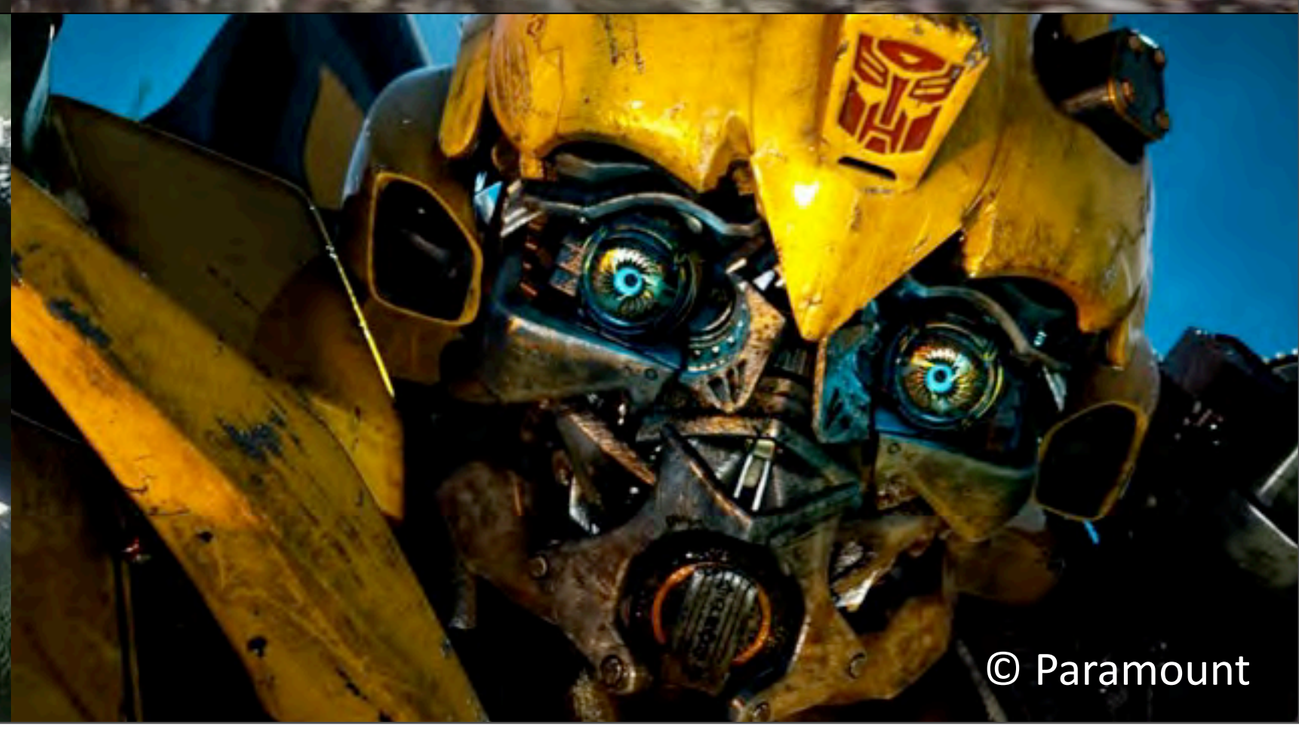
And while movie CG has made
enormous progress over the last 10 years,

I don't think **I will have to**
convince you that games
have seen **an even bigger** jump in fidelity.

Real-time rendering **isn't** photoreal yet,
but **it is** quickly catching up.

For the larger part
that is because of ever faster GPU's





But just as important,

over the last 20 years,
Most of the complicated issues in
off-line rendering have been solved.

Movie CG has pretty much
reached **perfect photorealism**
in every imaginable area.

Now that that milestone has been reached
we can **focus our effort on speed**....

Today we will present you some of the techniques and tools
we've developed during the creation of KIllzone 3
that allowed us to render more,
and create faster.

MICHAL VALIENT

LEAD TECH CODE

Michal Valient,
Guerrilla's Lead Tech Coder
will start off and present some **new engine features**
we **have developed** during the creation of Killzone 3.

He will explain the high level concepts

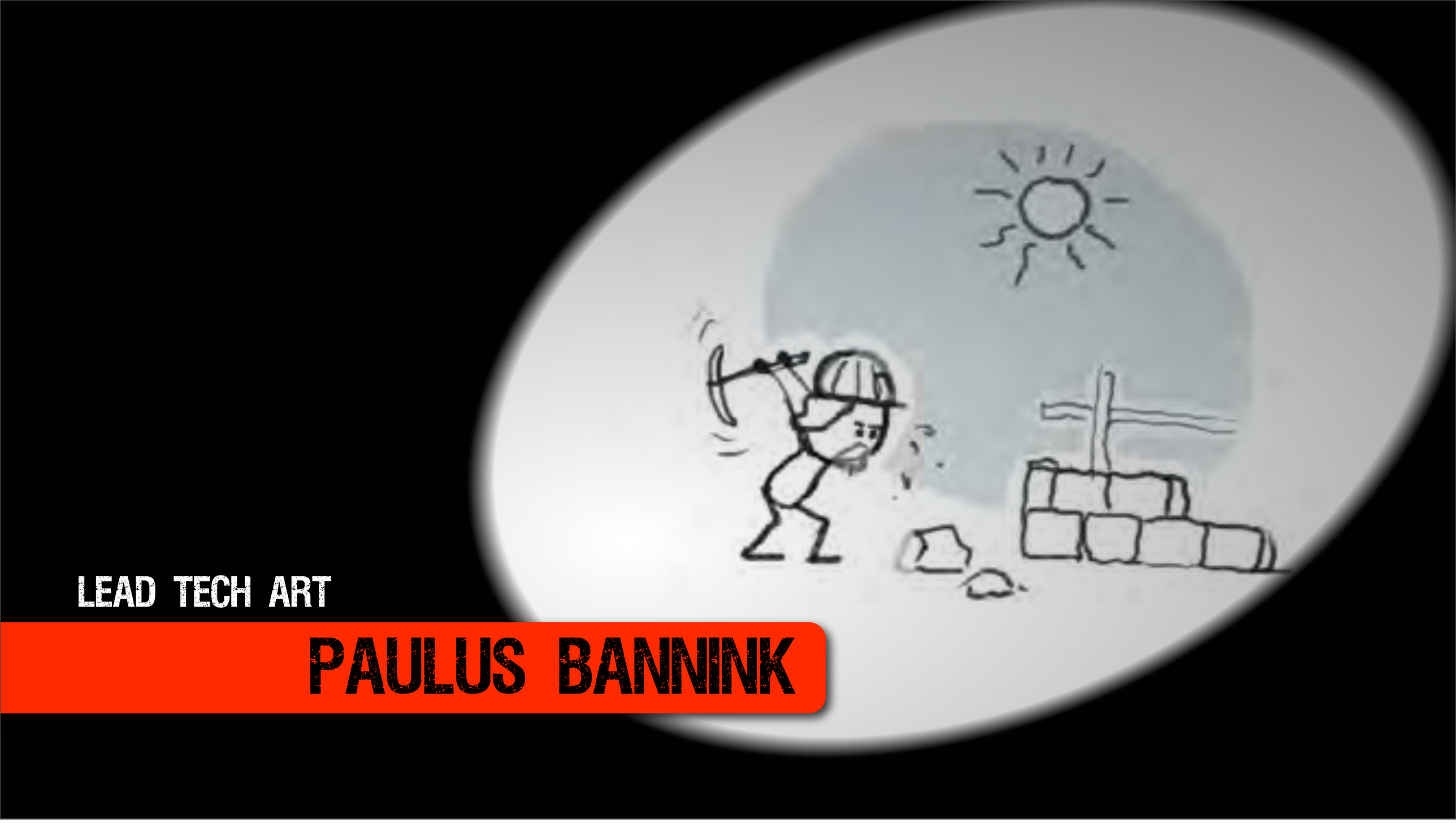
behind our game engine, as well as

the **technical challenges** we had to

squeeze more performance out of a

machine that was **already** using

100% of it's resources.



After him,

Paulus will talk you through the process of how we build our virtual worlds.

He'll explain some of the basic concept of a process

that is designed to allow artists to iterate very fast

while at the same time automating

many of the technical tasks

so artists can spend their time

focussing on the quality instead of technicality

Next,
Marijn will talk about the biggest challenge in games,

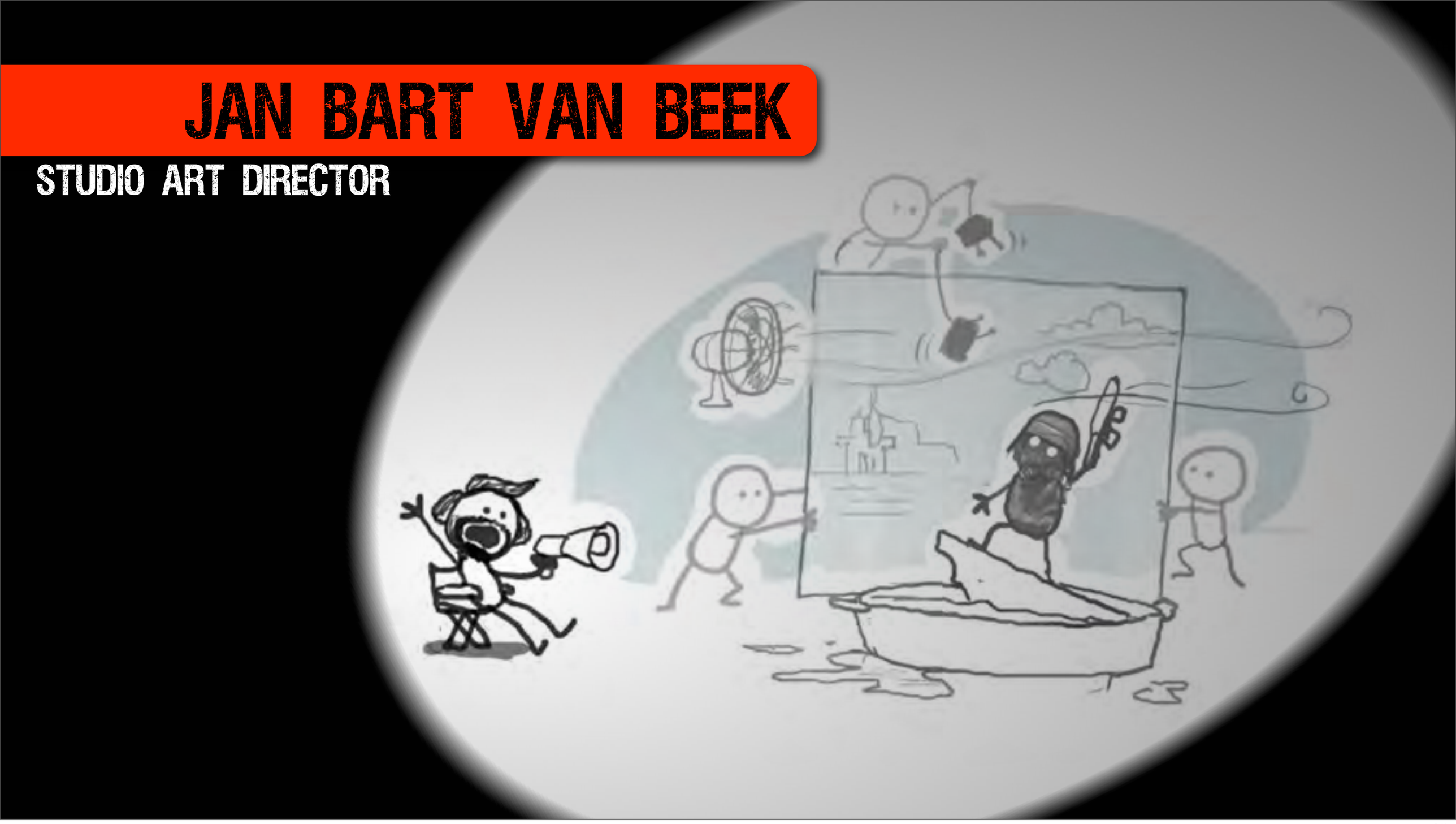
the one thing that makes them different from any other media.

... interactivity.

He will explain the challenges that interactivity
present and some of the tools and techniques we use.

JAN BART VAN BEEK

STUDIO ART DIRECTOR



And finally,

I will come back onto the stage to talk you through
the creation of Killzone's cinematic sequences.

I'll talk about the tools and pipeline that
allowed us to create 72 minutes
of cinematics using our
own game engine
as a renderer.

I'll discuss
the advantages of having a real-time workflow,

but I'll also talk about the
various unexpected hurdles
that we ran into.

And ofcourse I'll talk about some of our experience with 3D.

MICHAL VALIENT

LEAD TECH CODE

Hi I’m Michal

and although that image shows otherwise

I seriously don’t know anything about steam engines…

I hope everybody is familiar with all the intricate details
of Playstation 3 but for the few of you
who don't know, here's a quick overview

As you can see from the specs,
PS3 has you would expect from
a decent workstation back in 2006.

With one little exception.
<Click>

There are 6 very high performance processors
called SPUs that are great for graphics or parallel computations.

These are the reason why governments
and universities buy PS3s to build supercomputers.

DEFERRED FULL PASS

<0:30s – position buffer> <0:36s – normal buffer> <0:45s – specular buffer> <0:52s – diffuse buffer> <0:60s – lightmaps buffer>
<1:08s – lightmaps applied> <1:15s – point lights> <1:22s – sunlight> <1:30s – particles> <1:45s – color correction>

Killzone runs in 720p and it’s locked at 30 frames per second.

We use the increasingly popular Deferred Lighting rendering technique. We first render all opaque geometry, but instead of shading pixels we capture all material properties required by later lighting stages. We for example capture depth used to reconstruct position, normal, specular properties and of course diffuse color. We also render lightmaps with all indirect lighting. Whole pass takes us roughly 12ms.

After that we apply lighting sort of as a post processing step. We use the material information stored in the previous stage to calculate lighting of each pixel.
Typical light pass takes around 12ms of our frame time.
The lighting performance depends only on the number of lit pixels and it does’t really matter whether you have 200 small lights or three really big ones.

After lighting we spend 3ms to render all our transparent objects and particles.

The last stage of our rendering applies post processing. This pass takes 6ms, and most of it runs on SPUs.

By the end of the frame, we spent around 33ms to draw several million pixels
and that is what we in games industry call real-time.
Anything slower is called optimization opportunity.

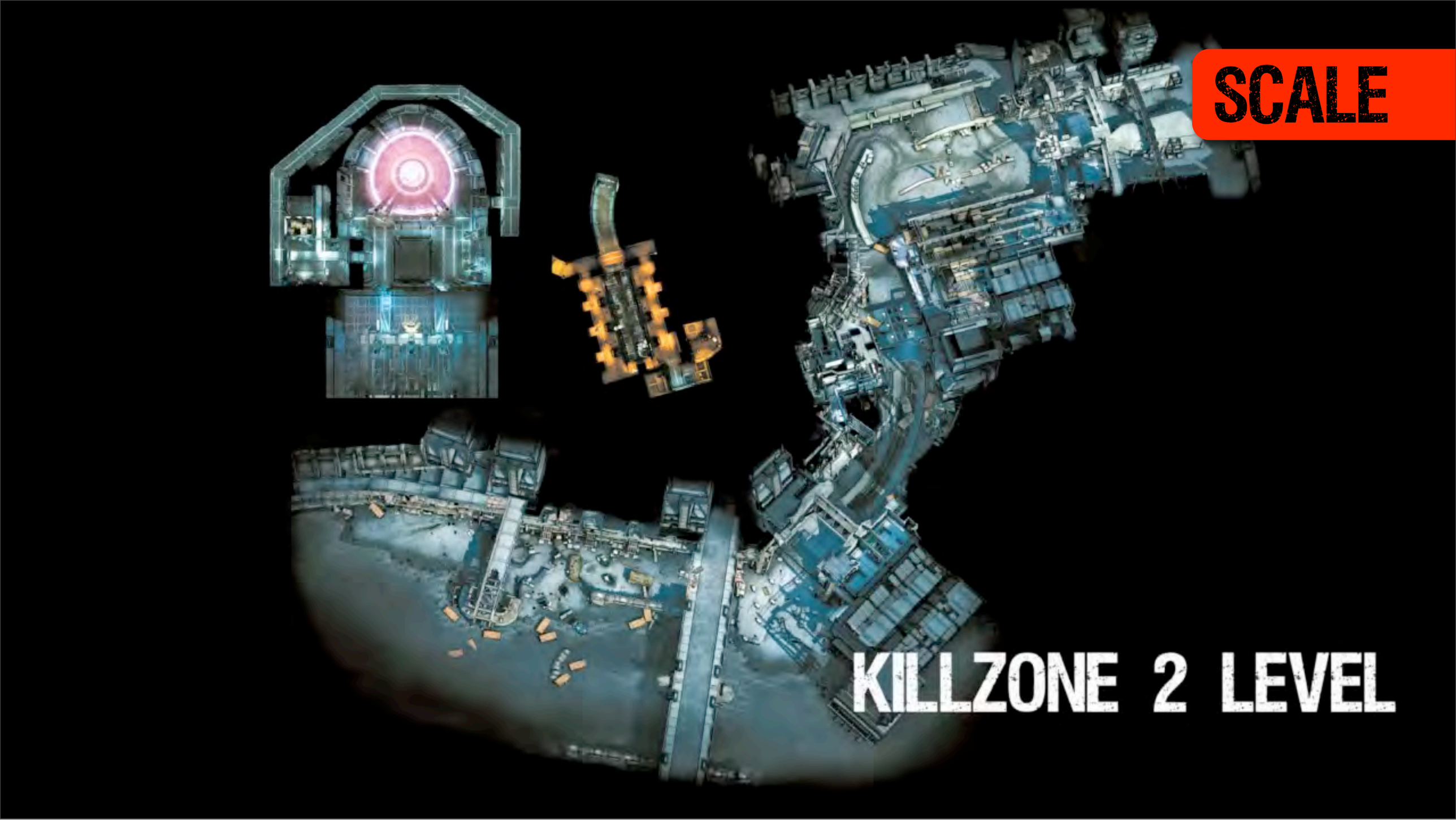
This is our second PS3 game so we already had a pretty
optimized engine and of course we expected a smooth sail.

Boy, were we wrong.

Let’s take look at some of the production goals that
shaped some of the most important new technical requirements.

With Killzone 3 we wanted to give
the player the sense of big game world.

Probably the best way to show this is to
compare it to the previous game, Killzone 2.



Most of the Killzone 3 levels are
several times larger than those in Killzone 2.

Instead of narrow corridors you explore great
open world areas with a lot more happening on screen.

Aside from scale, we also wanted the game
to have more variety and detail.

We wanted to show a clear graphical jump from KIllzone 2 to Killzone 3.

The level you now see on screen is a remake of
Killzone 2 level and again allows for nice comparison.

I hope you noticed on the walls or the ground, we
started to use much more complicated materials
and higher resolution textures and
we added a lot of geometrical detail to the scene.

With Killzone 3 we aimed to bring new experience to
players by fully supporting stereoscopic 3D and
two player splitscreen,
which requires a lot of extra processing power.

By the way, this is us showing
the very first 3D capable build of Killzone 3.

And perhaps most importantly, we
wanted to finish the game in two years.

These goals pretty much defined
the new challenges we needed to solve…

MORE POWER ?

The main problem was to find new ways
to fit this large amount of detail into
the PS3 and still run it at 30FPS.

Upgrading the PS3 would certainly be a great option.
But that’s not how it works.

This time around we had to focus more on
big wins that give us memory and performance
and less on shinny new graphical features.

The main problem was to find new ways
to fit this large amount of detail into
the PS3 and still run it at 30FPS.

Upgrading the PS3 would certainly be a great option.

But that’s not how it works.

So this time around we had to focus more on
big wins that give us memory and performance
and less on shinny new graphical features.

The main problem was to find new ways
to fit this large amount of detail into
the PS3 and still run it at 30FPS.

Upgrading the PS3 would certainly be a great option.

But that’s not how it works.

So this time around we had to focus more on
big wins that give us memory and performance
and less on shinny new graphical features.

The main problem was to find new ways
to fit this large amount of detail into
the PS3 and still run it at 30FPS.

Upgrading the PS3 would certainly be a great option.

But that’s not how it works.

So this time around we had to focus more on
big wins that give us memory and performance
and less on shinny new graphical features.

The main problem was to find new ways
to fit this large amount of detail into
the PS3 and still run it at 30FPS.

Upgrading the PS3 would certainly be a great option.

But that’s not how it works.

So this time around we had to focus more on
big wins that give us memory and performance
and less on shinny new graphical features.

The main problem was to find new ways
to fit this large amount of detail into
the PS3 and still run it at 30FPS.

Upgrading the PS3 would certainly be a great option.

But that’s not how it works.

So this time around we had to focus more on
big wins that give us memory and performance
and less on shinny new graphical features.

The second challenge was simply to find
the areas where we can improve the artist workflows
so that we can make this bigger game in two years.

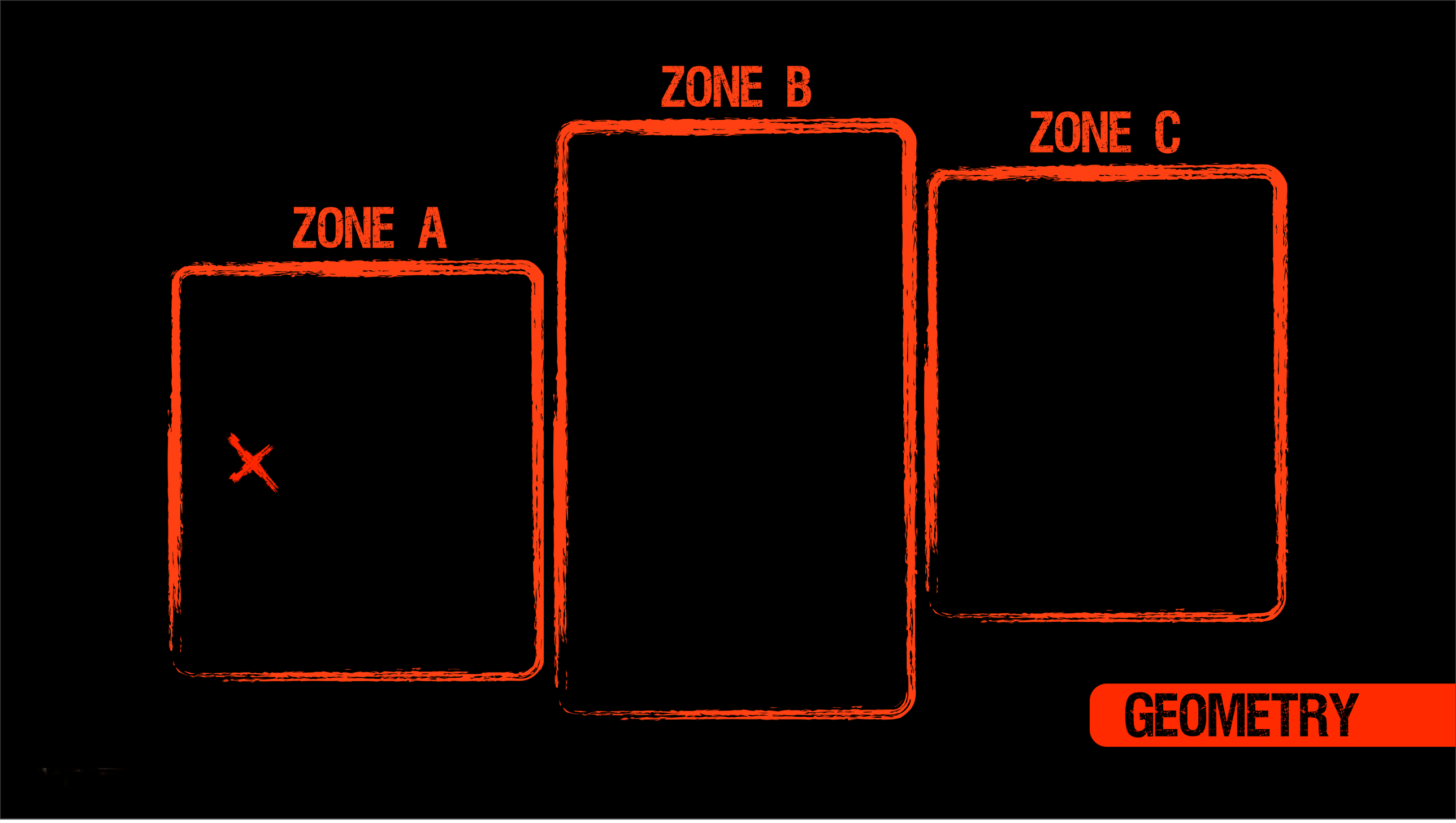
So, those were our problems and challenges.

Let’s take a look at some of the technical solutions we came up with.

One thing you realize during game development is
that you never have enough memory.

If you manage to optimize something and
free up some space, it will be immediately
consumed by new content.

This makes a good streaming system pretty much necessity for current games.



Therefore Killzone engine has several active systems
that stream game data, graphics or audio into memory when it’s needed

The highest level system allows us to split large game
world into smaller sections that fit into memory.

<Click>
As you play through a section of a level - Zone A here.
we’re already loading zone B in the background

<Click>
Once we enter zone B, we dump zone A from memory.
And this creates enough free space to start loading Zone C into memory.

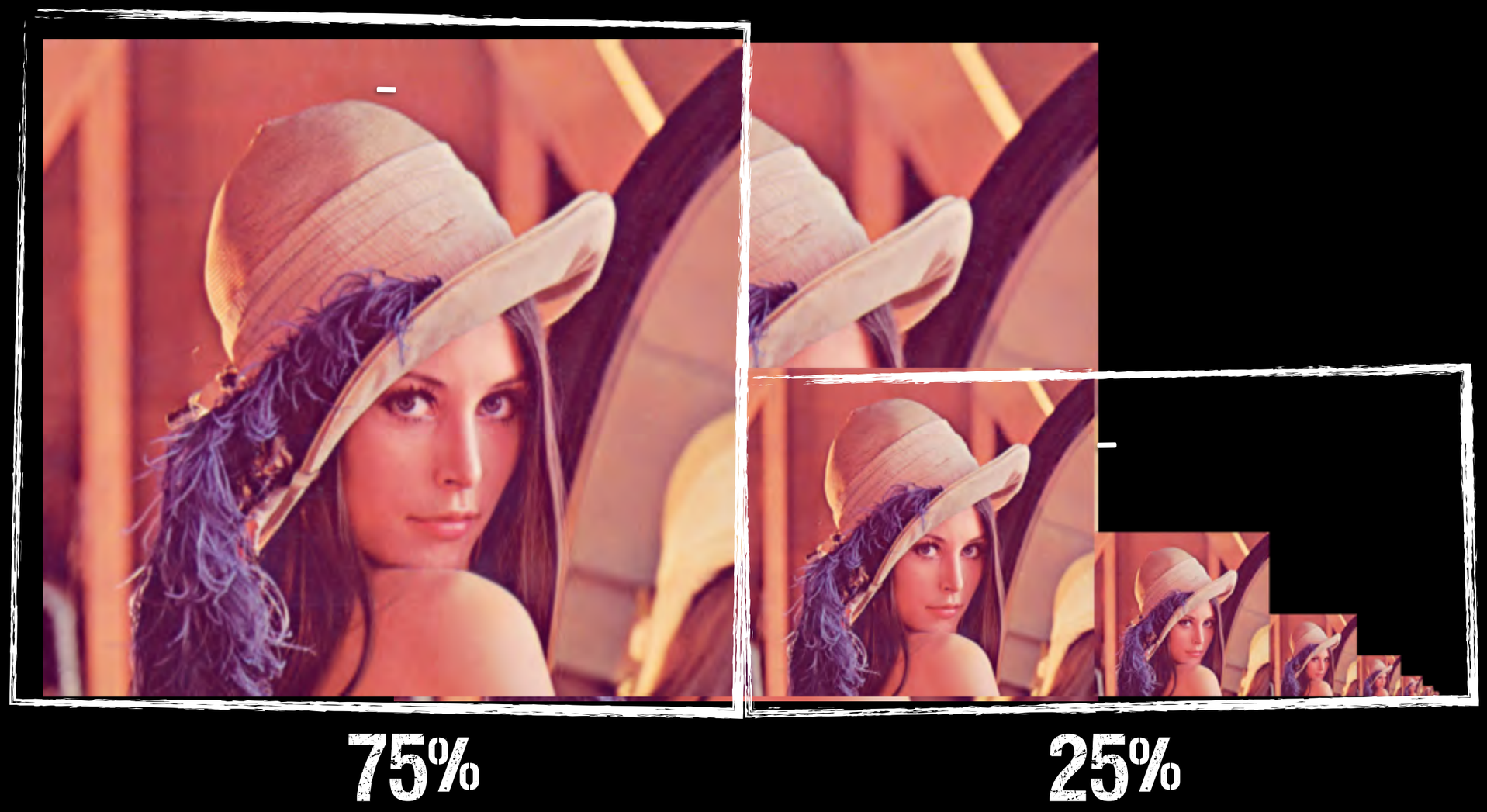
<Click>
And if everything is timed correctly you will never
notice the switch and there are no loading times.

This is the most basic form of streaming in our game.

A completely seperate streaming system exists for sound and music.

Although some sounds are stored in RAM,
most of the music, ambient sounds and even a large amount of the character dialogue is played directly from harddisk.

This allows us to have very large amount of variation without increasing the memory budget.



KIllzone 2 didn’t have a texture streaming system.
And this presented a very clear opportunity to gain back a large amount of memory.

Textures can easily claim half of all the memory that is available.
But as you probably know most of the textures in
video games are stored as mipmap chains....
a sequence of ever smaller versions of the same image.

<click>
This is done to speed up GPU’s so they don’t need to
read large amounts of data when rendering small objects.

<click>
But it also means that 75% of memory is
taken up by textures that are only used 10% of the time.

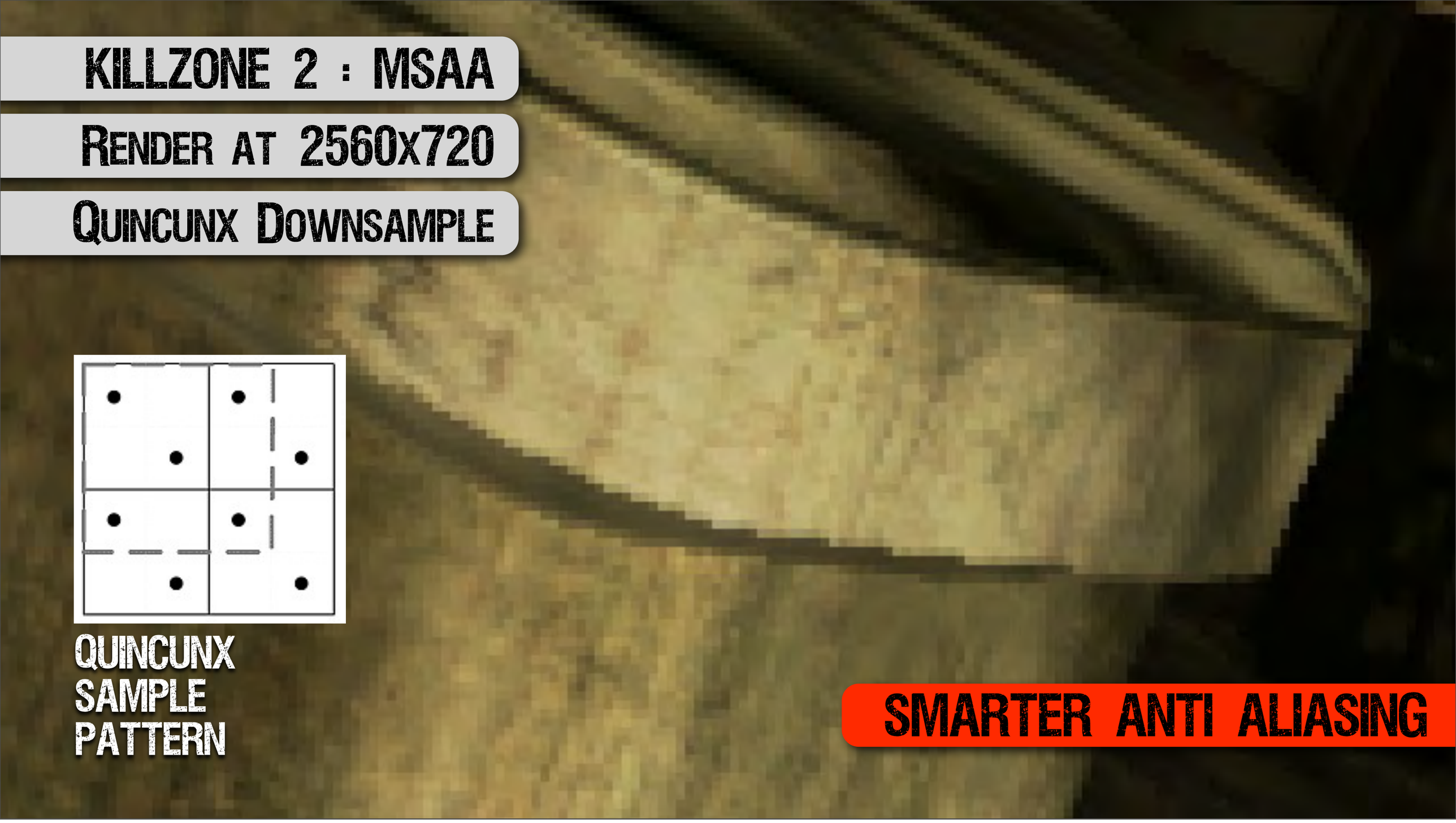
Our solution is probably the simplest you can think of,
but by simply ONLY loading the highest mipmaps into memory when they are needed
we free up a lot of memory and also reduce loading times.

We have also used a new form of streaming that as far
as we know hasn't been done in videogames before.

The sea water we created for Killzone 3 was prototyped in Houdini,
but we quickly realized that trying to get the same results in real-time
would be too computationally heavy.

So we saved out the entire simulation as a mesh per frame.
The output dataset was several hundreds of megabytes large.
Much too large to fit into memory.

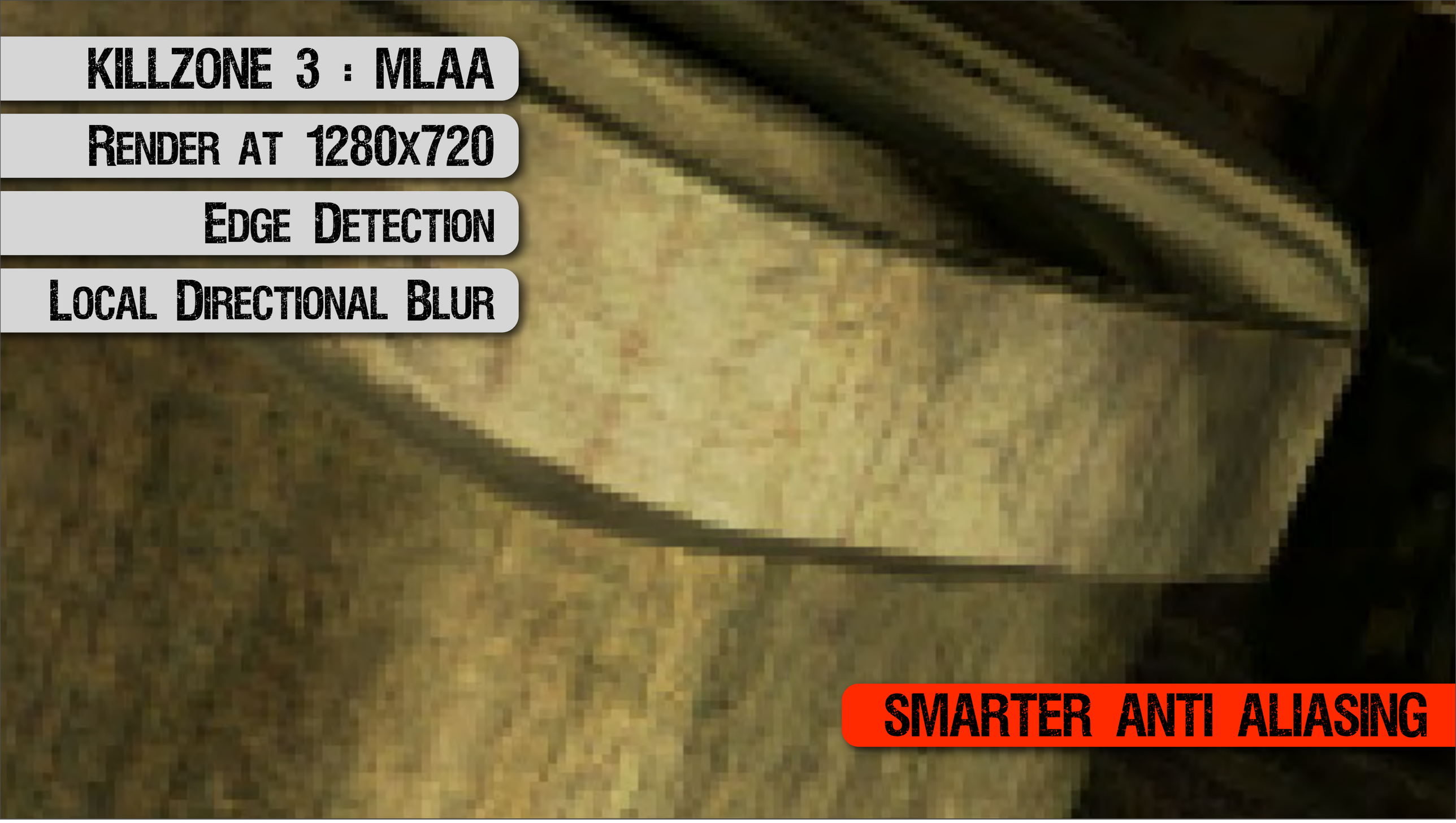
But when we looked at how much data we
would need per second, we realized it would be
less than a megabyte, something we
could very comfortably do alongside all the other streaming.



Our change of anti-aliasing algorithm is a great example of smart optimization
that gave us huge performance boost, better looking game and more memory.

Killzone 2 used traditional form of GPU anti-aliasing.
We essentially rendered scene in twice the horizontal resolution.
Scaling down to native resolution then removed the aliased edges.

This approach is very expensive in the deferred renderer,
because the cost of lighting is proportional to the amount of pixels.



Killzone 3 uses Morphological Anti-Aliasing
implemented by Sony's Advanced Technology Group.

It is an intelligent edge blur post-processing
filter that runs entirely on SPU.

As you can see it softens the edges really well and
most importantly we gained 30% of the GPU time back by
getting rid of overhead of our previous solution.

Since this is a post processing filter and is not PS3 specific
you should definitely consider something similar for your next game or
next time you're waiting for the renderfarm to spill out your image.

Determining which objects are visible is an extremely important topic in games.

As every game developer knows, for every millisecond you
waste on an **object** that is **not visible**, a little **unicorn** dies.

<pause>

Unfortunately our **previous** system was **not perfect**.
<Click>

Killzone 2 used **portals**.
It essentially required artist to **manually tag windows and doors**
so the engine can determine what objects are visible for a frame.

<Click>

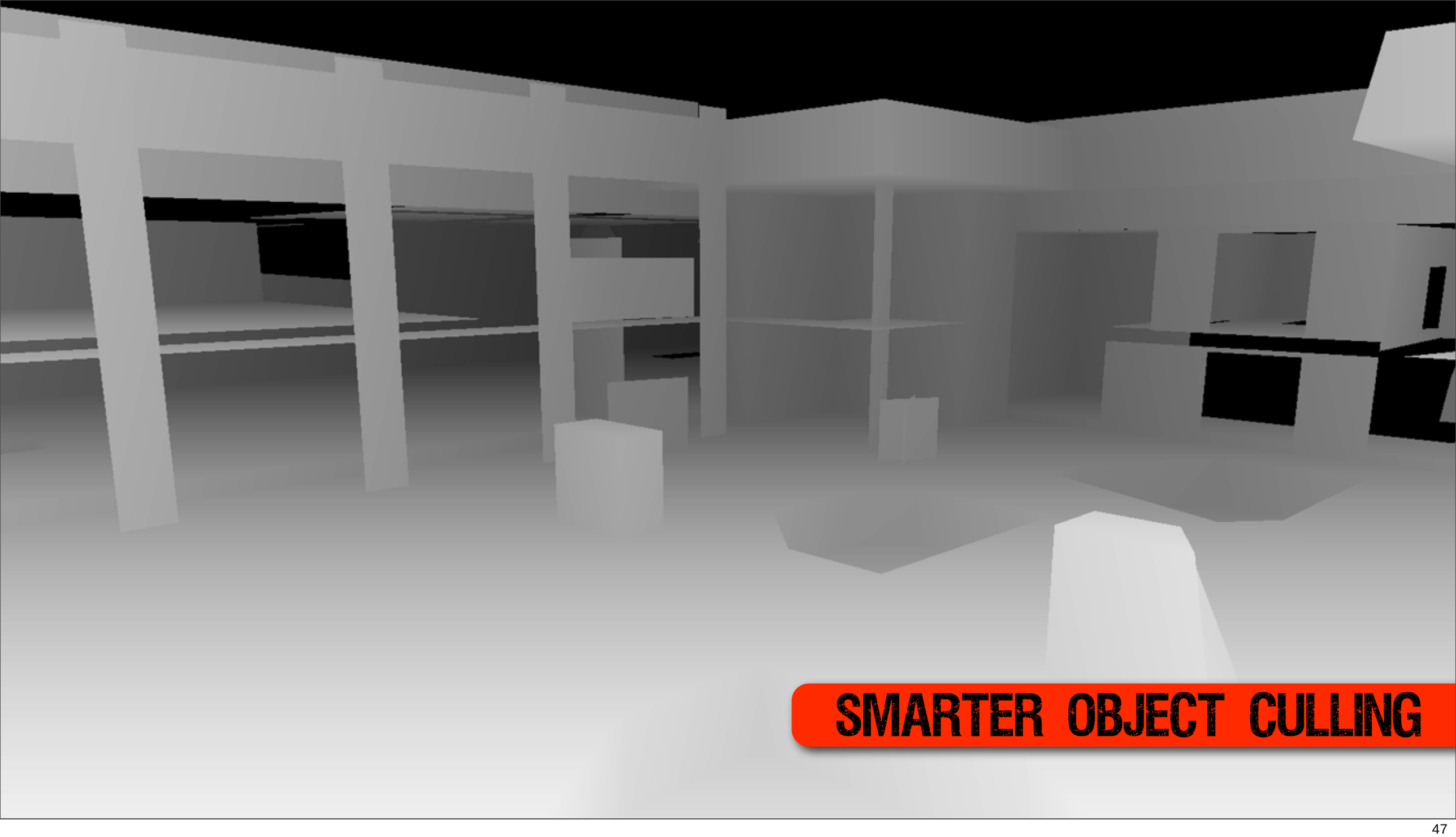
This **hard to understand process...**

<Click>

could only be done on **finished level and**
always required several iterations to get right.

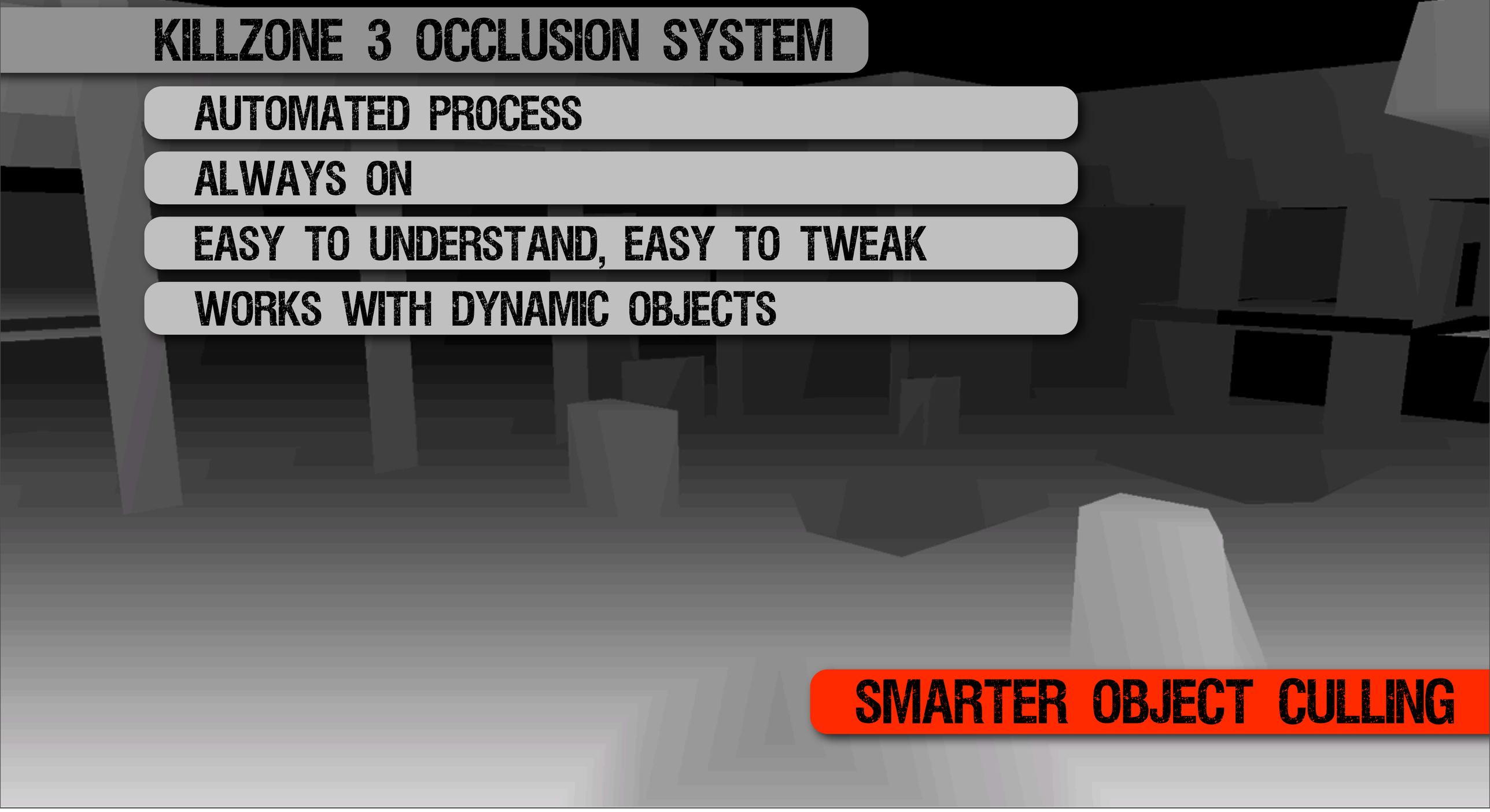
Therefore we worked on easier to use solution.

Before rendering the frame...



....**we use SPUs to render very simplified version of the level geometry into a depth buffer like this.**

We test the **bounding box** of **every object** in the camera frustum
against the **scaled down** version of this depth buffer
to determine whether it's visible or not.



This system has several big advantages compared to a portal-based system

First...<click>
We can generate most the occluder geometry automatically.
It's always on, without the need for manual steps.
<click>
But it's also very simple to understand for artists...
and when the system needs tweaks they can do
so by simply placing additional occlusion geometry

Finally, <click>
Compared to most of the other solution this system is fully dynamic
as any object in the game can become an occluder.

<movie start automatically !>

The following movie will show you how the culling system works
and how effective it can be.

<Wait for inset windows to appear>.
In the left inset you can see the occluders that are being rendered.

While on the right side you can see all the object that have been occluded
and aren’t being renderer in the final frame.

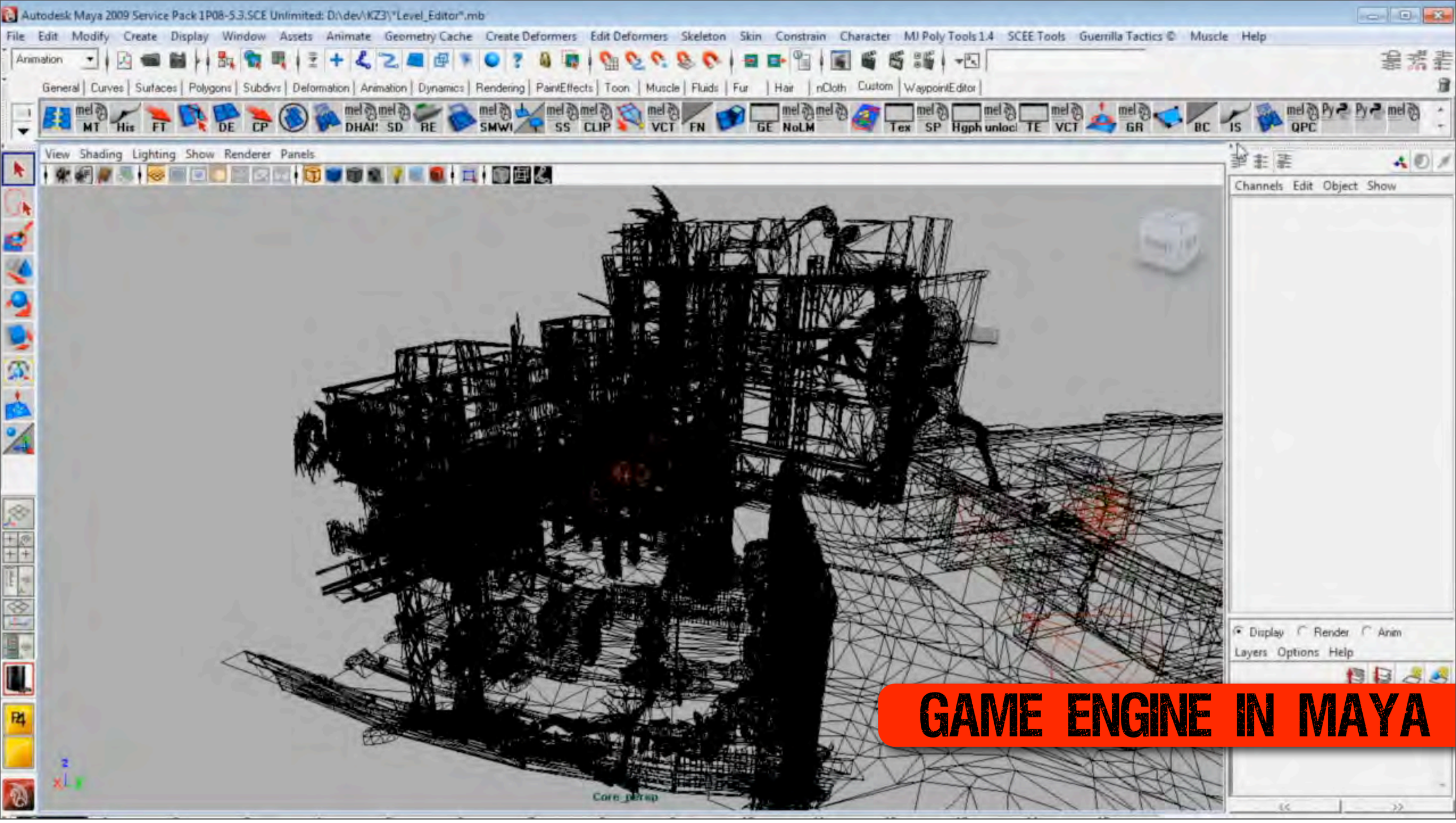
You can see that not every big object is an occluder.
Background mountains or ground itself are missing
from the occluder view.

This is of course because there is nothing worth
to occlude behind these objects.

<wait for movie to stop>
This frame shows a really good example of how effectively the system works.
The wall that our two heroes are hiding is nicely occluding quite a few objects.

So far I talked mostly about the engine,
but it's only one part of the equation.

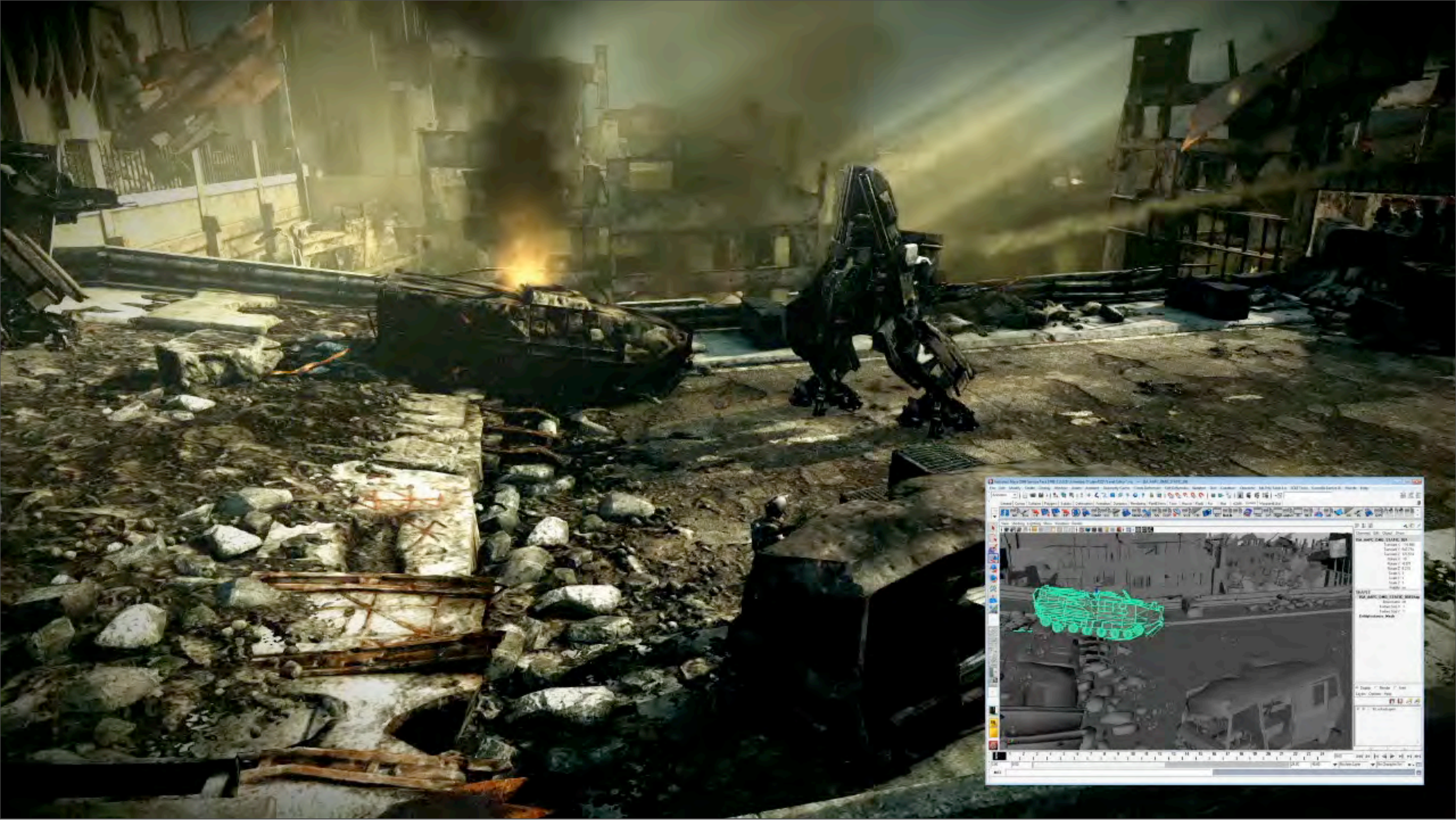
Ultimately it's the quality of your tools that
allow you to make better games... in less time...



We use **Maya** for most of the **level geometry** and shader authoring.
Unfortunately **exporting** data from Maya and then
converting it to gamedata can be **time-consuming**.
And to **avoid** this, we've created a plug-in
that runs the **game engine in Maya** viewport.

This **allows** artists to make **changes** to
level **layouts**, **lighting** and prop **placement**
and **evaluate** those changes within **Maya**,
without the need to export it to the game.

All **shader authoring** is also done completely within **maya**.
Shader artists can create **arbitrary shader** using a combination
of standard maya nodes, as well as various **engine specific nodes**.

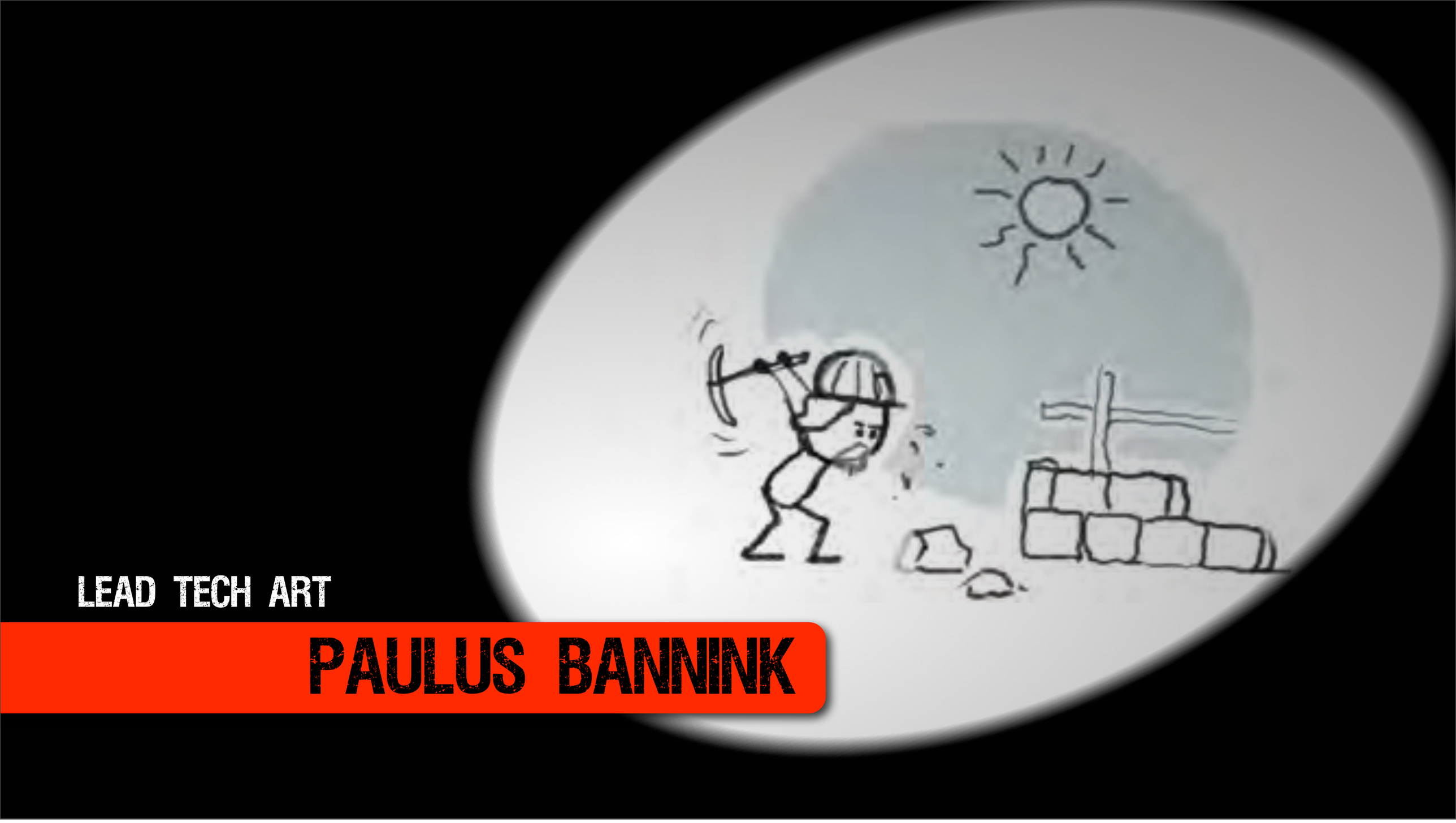


To further **shorten** the iteration **times** we created
a **live connection** between Maya and the running game.

This allows you to **control the camera** in both at the same time.
But it also allows you to **move** objects, change the **lighting** and
change other **attributes** such as LOD distances while the game is live.

You can even start **interacting** with **Artificial Intelligence** agents
and really confuse the heck out of them.

We have a **great plans** to extend the Maya and game **integration** further
and potentially **allow** any kind of **modification** to be available in game real-time.
Because with the **growing complexity** of the games,
this is the most **important bottleneck** any company needs to remove.



Hi, I’m Paulus Bannink and I’ll be talking about how we made the environments that you see in Killzone 3.

When making environments for a game flexibility is the key.

There are so many things that depend on and influence a game environment that it becomes almost impossible to lock it down before production is over.

There is the **development of engine technology** which may change the amount of **detail** that can go into the environment;
there is the **level design** that can radically change the **layout** of the environment
and then there is the **art direction** that may decide to change the whole **theme** of the environment.

<Click>

Having your environments in flux at all times calls for some custom pipelines.

The main goal here is to have anything that needs to be redone every time something is changed as an automated process.

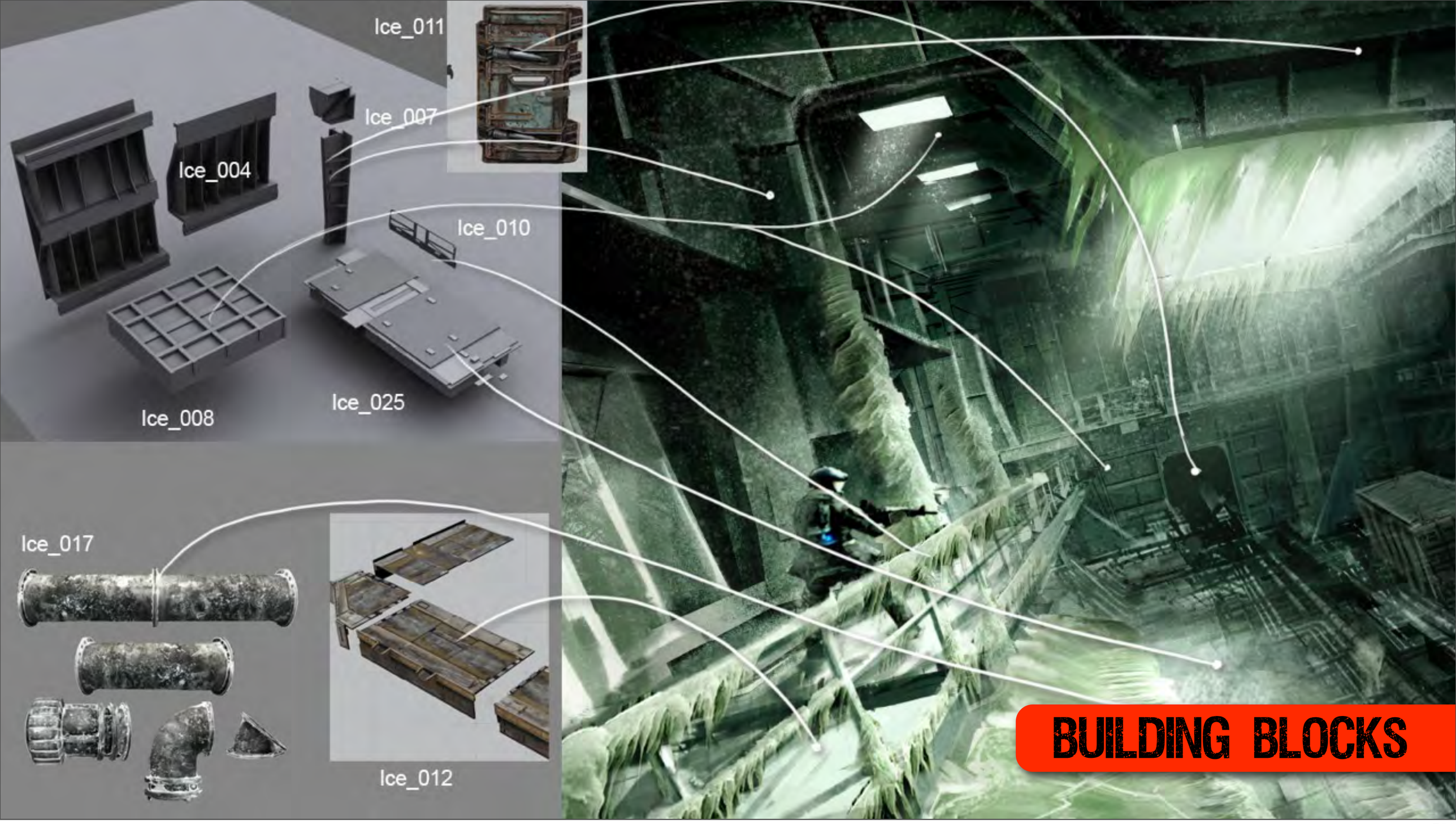
However it’s just as important to have a workflow that allows for quick and easy changes to anything in the environment.

To meet these challenges we developed a pipeline based on what we call “Building Blocks” back when we were working on Killzone 2.

Building Blocks can best be compared to Lego blocks,
in that they are generic objects that can be used and combined in many ways to create unique new objects or in our case complete environments.

These building blocks can be anything ranging from small objects like pillars, railings or pipes to complete buildings or vehicles depending on the requirements of the environment.

These requirements are first derived from the mood boards created by our visual design department.



Once these moodboards have been approved they will be dissected by our environment art team to come up with a list of building blocks that will we required to construct the environment.

This list will be send back to the visual design department so they can create art briefs for every building block.

This is one of the moodboards that was created for the jungle level in Killzone 3

Even a organic environment like this one would in the end be made with building blocks

And here you see a building block brief for one of the required building blocks...

And this is from the ice/sea level that Michal already talked about when he was explaining how we made the ocean

And the matching
brief

This is one of our multiplayer levels

These briefs are usually a single image that shows visualisations of the final building block and also has additional information like material examples and even polycount’s and texture sizes.

By placing all this information into a single file it becomes very easy to outsource and this is also one of the strong points of the Building Block workflow.

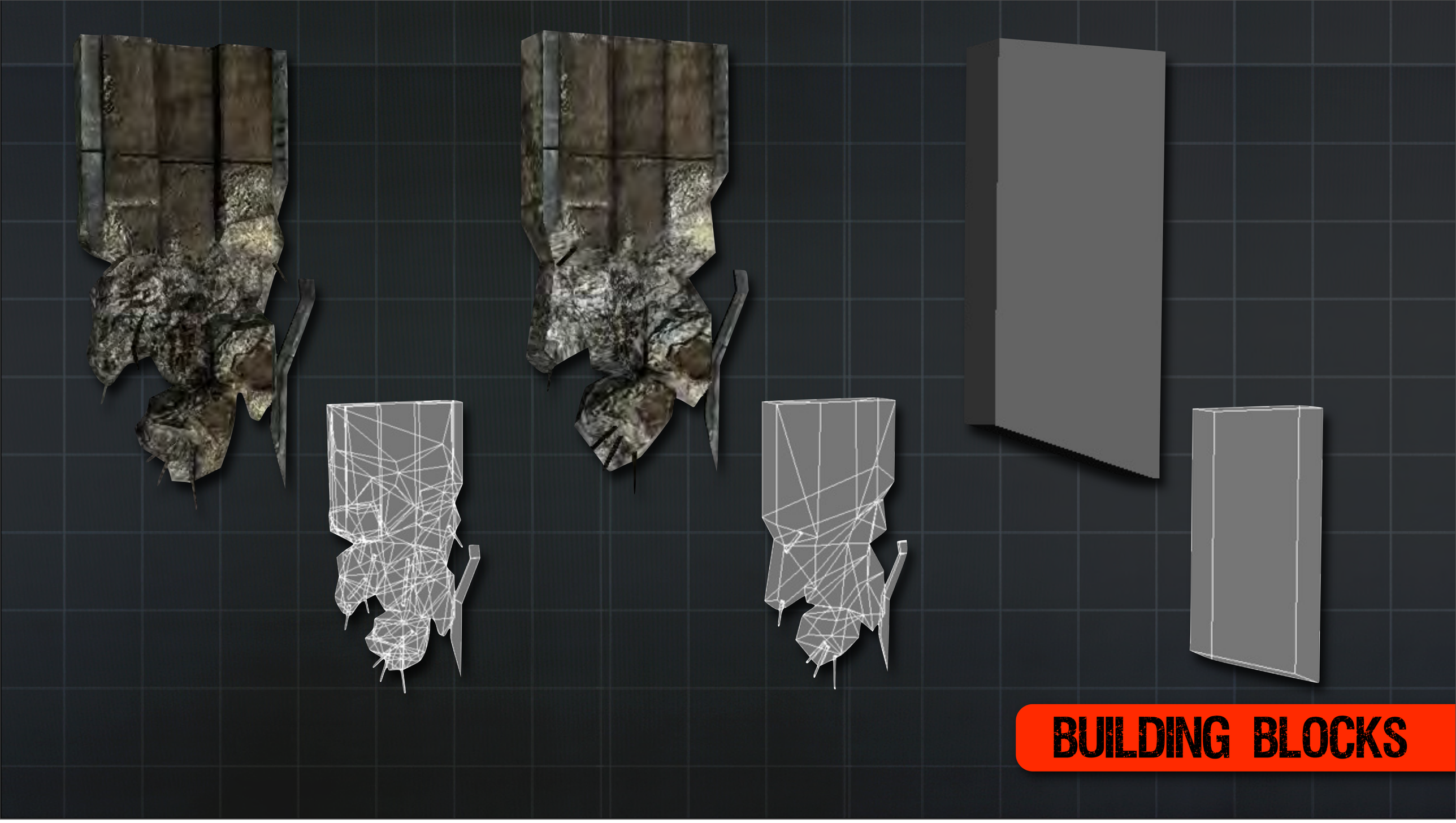
Most game assets have some properties that are unique to the game engine and are therefore quite difficult to outsource without some sort of training/support for the outsource company.

Since these building blocks are so simple by themselves they are not only easy to outsource but also safe;

there is not much that can go wrong.

Based on this brief an artist will create a collection of Building Blocks that are grouped together in a single file.

A collection will have a single shader shared by all the buildings blocks, this is important later on in the pipeline when we start looking at optimisation.



What you will see here are the standard components that make up an example building block:

A high resolution mesh...

a lower resolution mesh...

...and a collision mesh.

[pause]

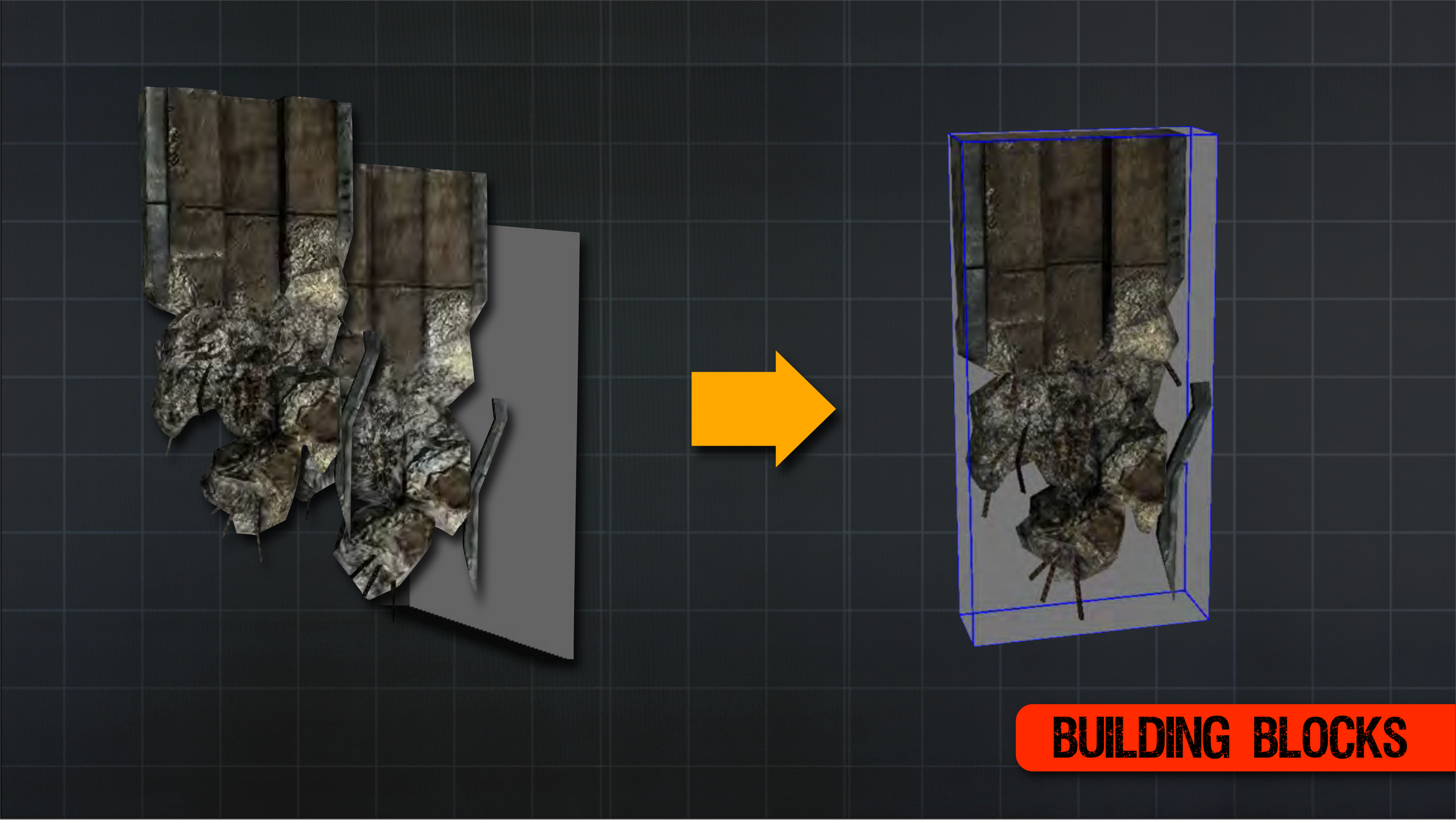
This shows a different building block from the same Family

Once these have been approved by the art director we feed the Building Block components into an export process that,
assuming the objects are named correctly,
will construct the final building block asset.

<click>

This asset can best be seen as a **data container** that carries all the Building Block’s data all through the pipeline.

From here on in the building block will only be handled in this form, and the actual source geometry will be left alone.

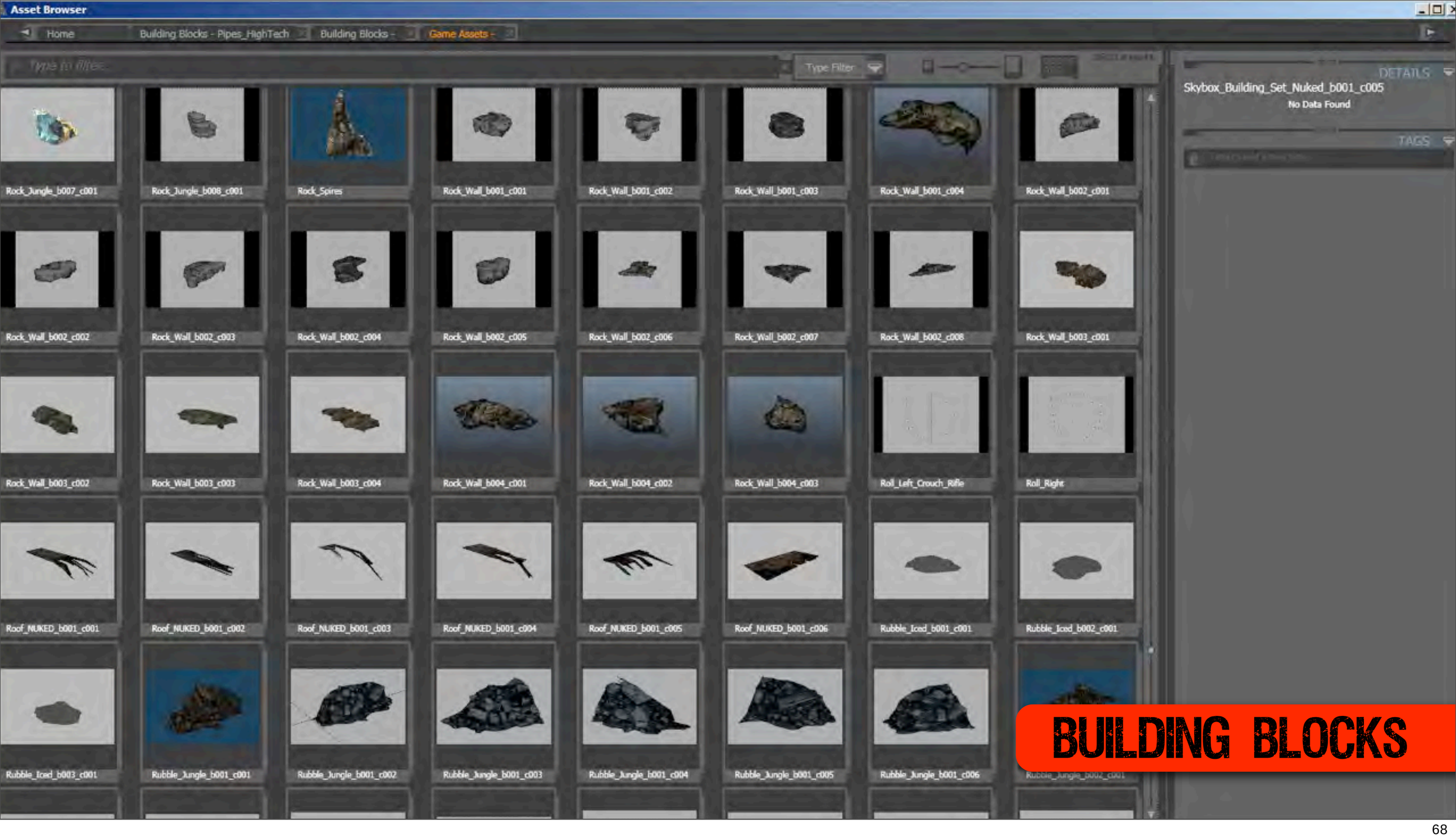


Once these have been approved by the art director we feed the Building Block components into an export process that,
assuming the objects are named correctly,
will construct the final building block asset.

<click>

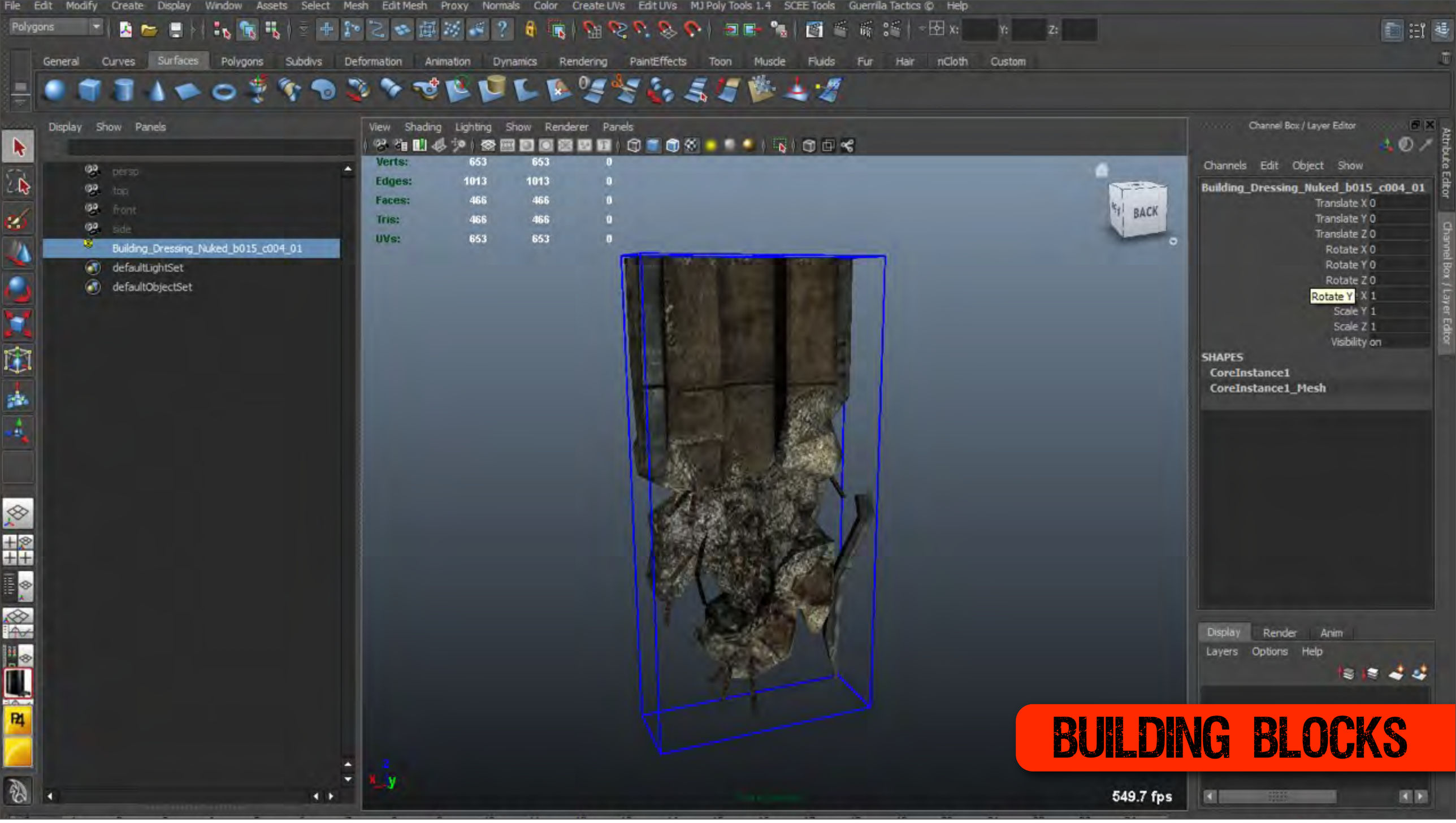
This asset can best be seen as a **data container** that carries all the Building Block’s data all through the pipeline.

From here on in the building block will only be handled in this form, and the actual source geometry will be left alone.



This export process also pushes the building block to our asset manager;
this will be the starting point for the environment artists to build their scenes.

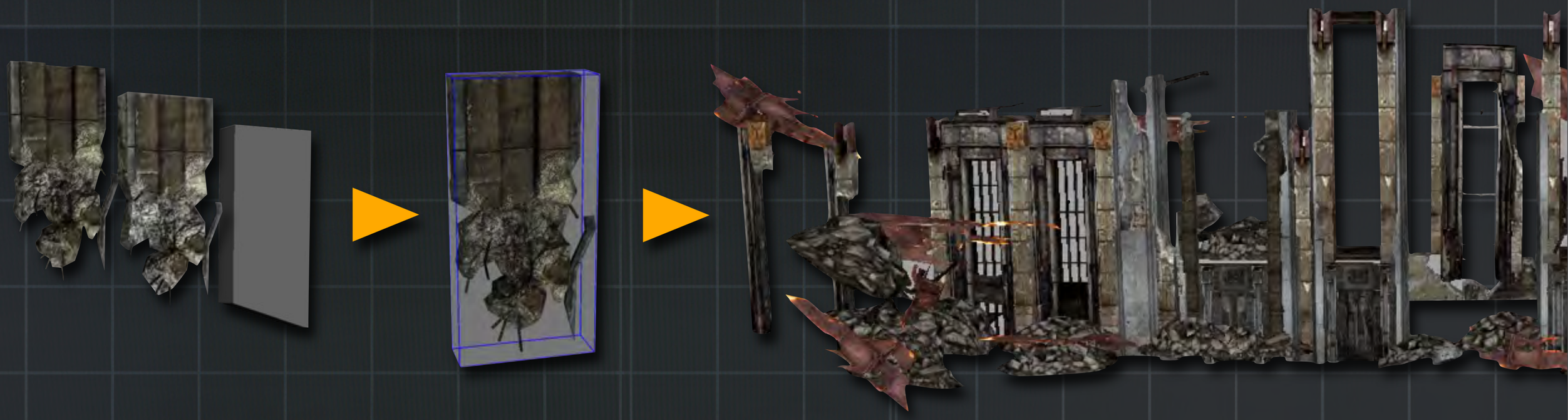
They will search through the asset manager for the most appropriate building blocks and instance them into the Maya scene.



This instancing is done through a custom Maya node that can load an exported file and show a representation of the geometry in the Maya scene.

It’s important to realise that the model you see here is not a maya model but an actual in game mesh.

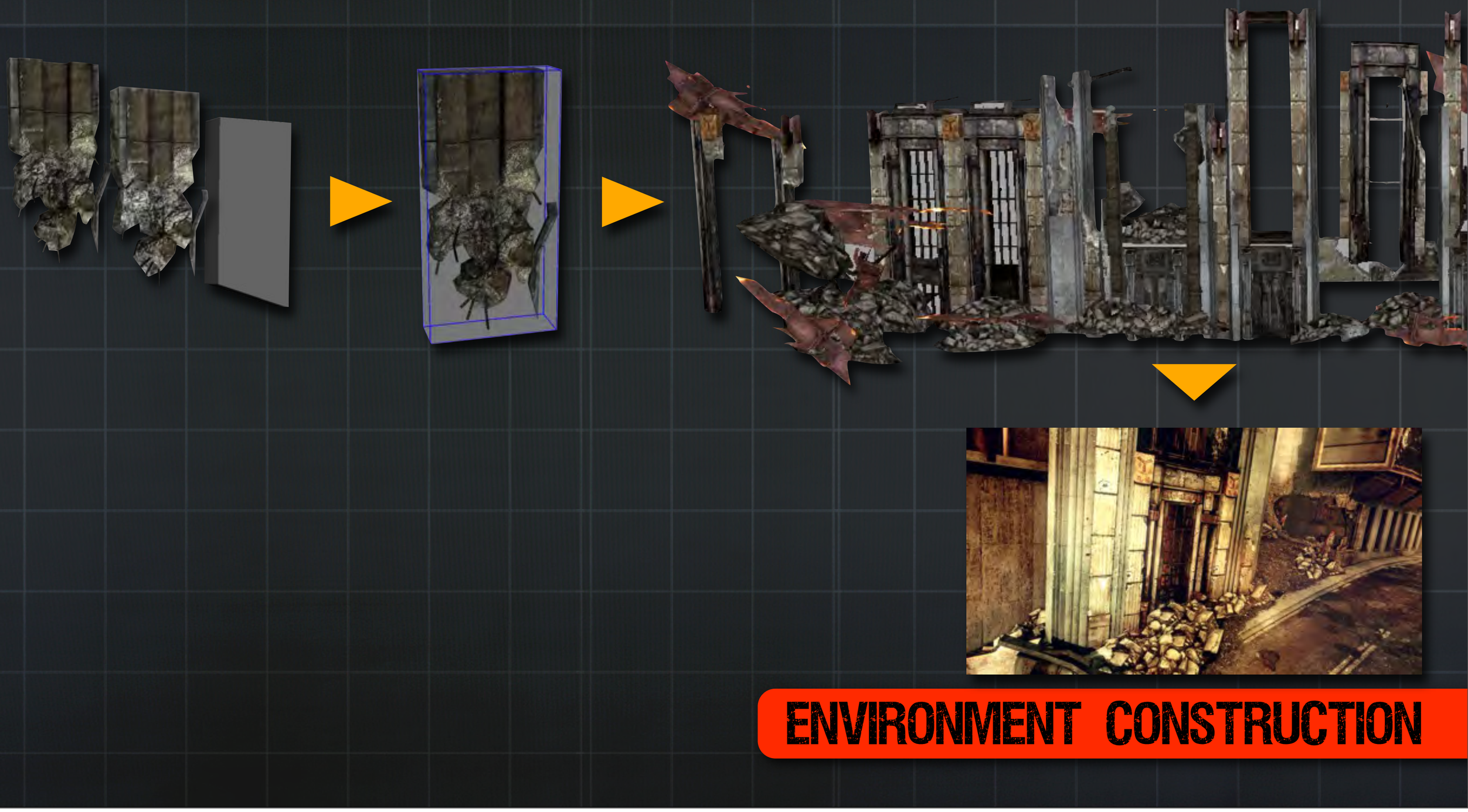
Through various custom render modes in the Maya viewport it is also possible for the artist to visualise additional data of the Building Block like the collision mesh or the shader properties.



ENVIRONMENT CONSTRUCTION

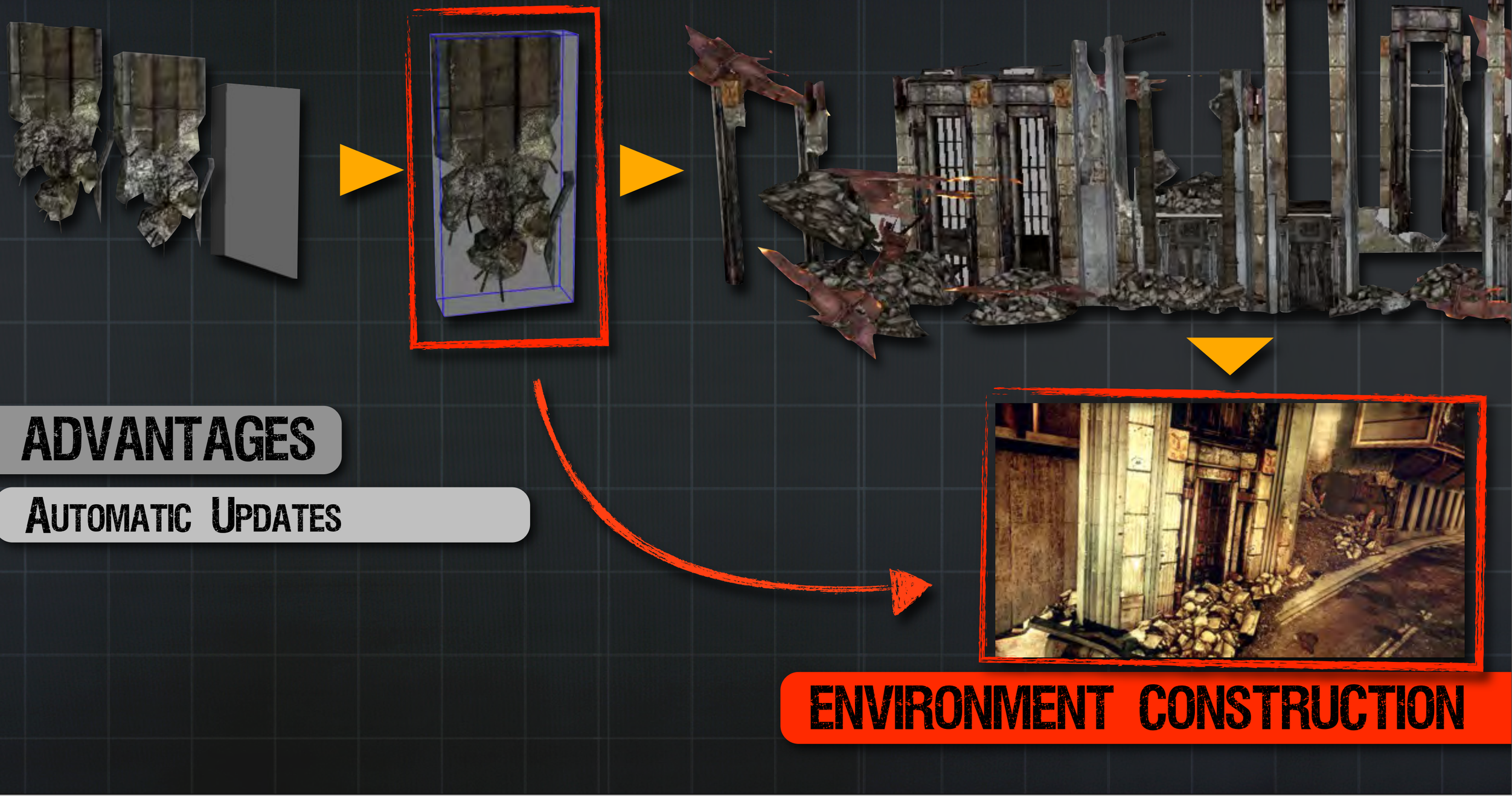
The only interaction the artist has with this instance is transformation,

so by moving rotating and scaling the instance they will start building the environment.



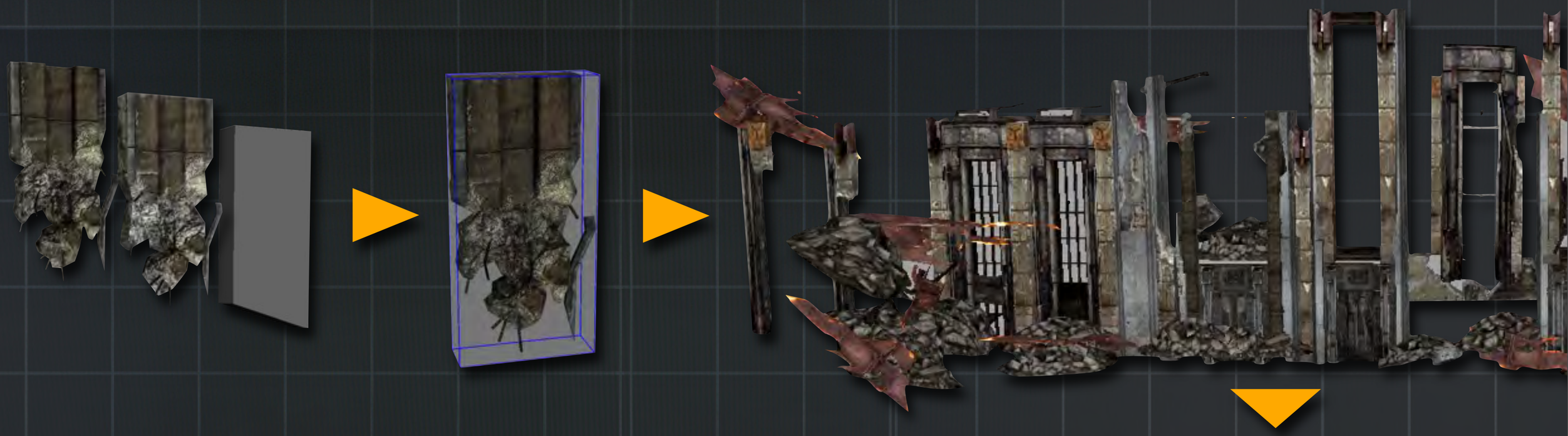
Once he or she is satisfied with the environment the scene is exported to the engine and the environment can be loaded in the game.

Apart from making is fast and easy to construct a new environment this pipeline introduces the following benefits...



Since the building block instance in the game is still referencing the original asset,
any changes to that asset will be automatically updated in the game.

So any update to this asset will automatically propagate through the whole game.



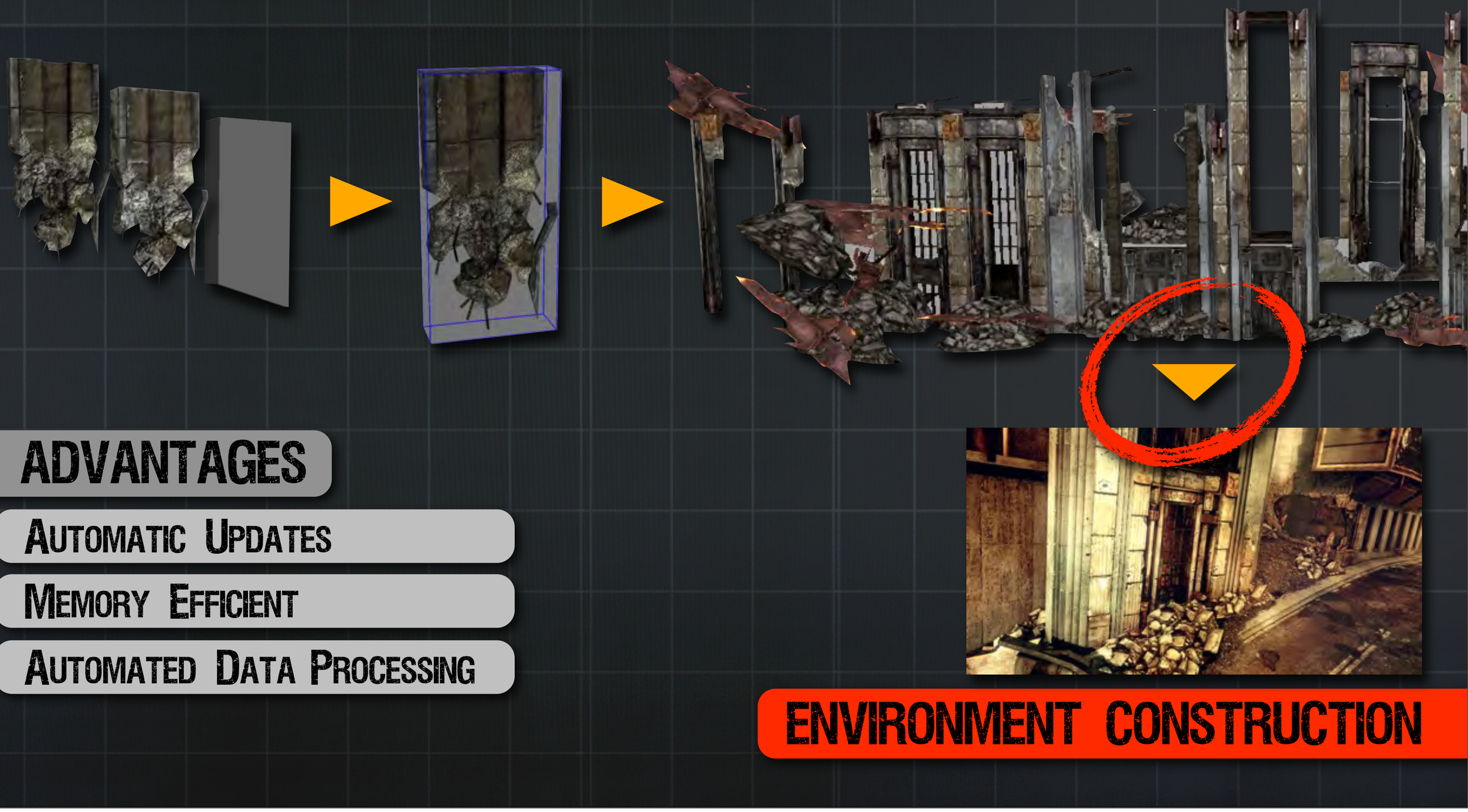
ADVANTAGES

- Automatic Updates
- Memory Efficient

ENVIRONMENT CONSTRUCTION

Secondly, we only have to store the meshes once and then render them multiple times with various transformation matrices applied.

This means that we can save a lot of memory.



Finally, every time we start the game and load the level the exported data is converted to run-time data.

This conversion step gives us the opportunity to insert any automated processing we want to do to our data.

I’d now like to talk about some of these processes.

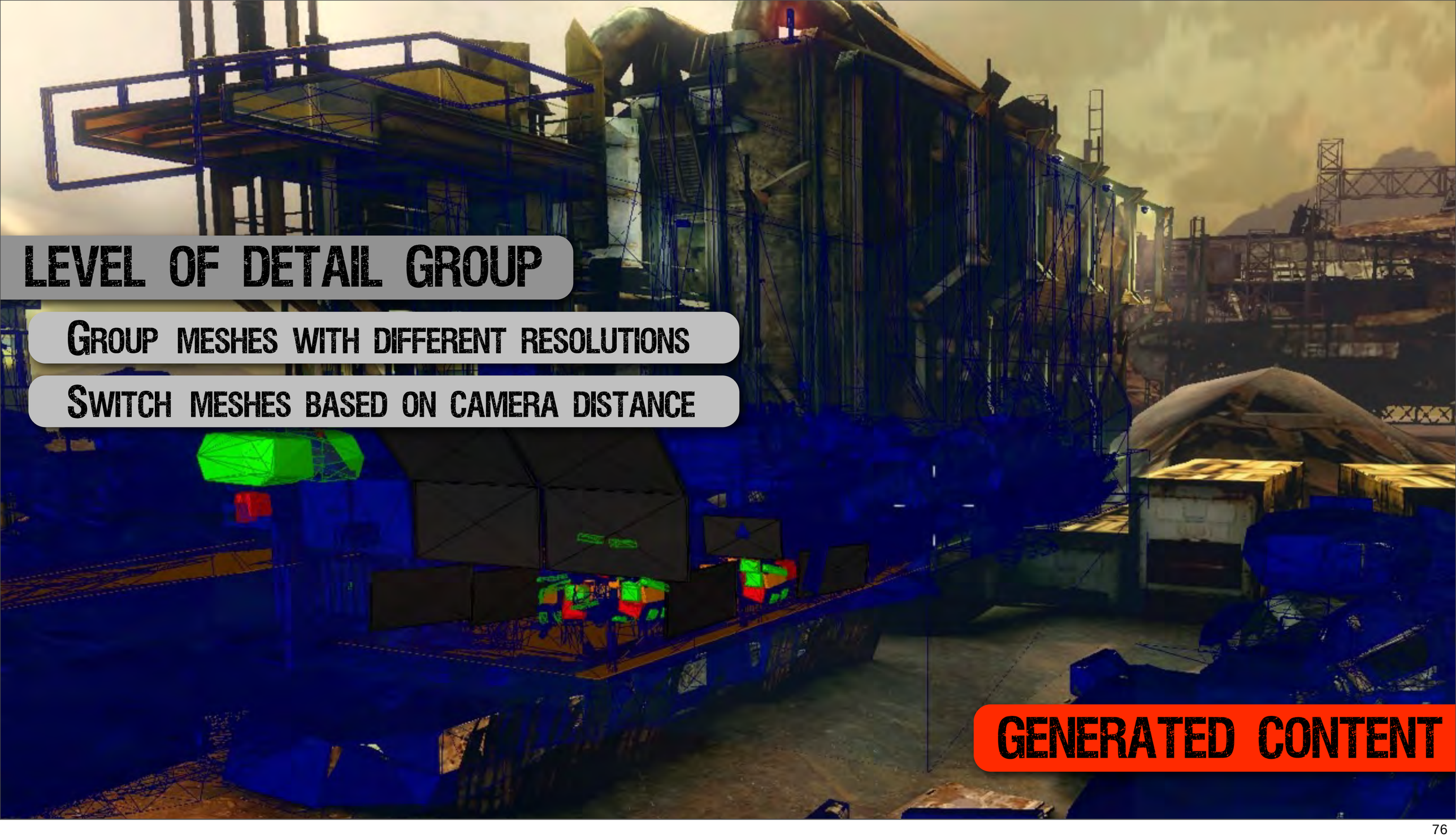
Creating game environments involves the building of a lot of content that may not be noticeable for the player but is required to make the game run real time or work at all.

Examples of such content are collision meshes for physics interaction, occlusion meshes for polygon culling or optimised low resolution meshes for far away objects.

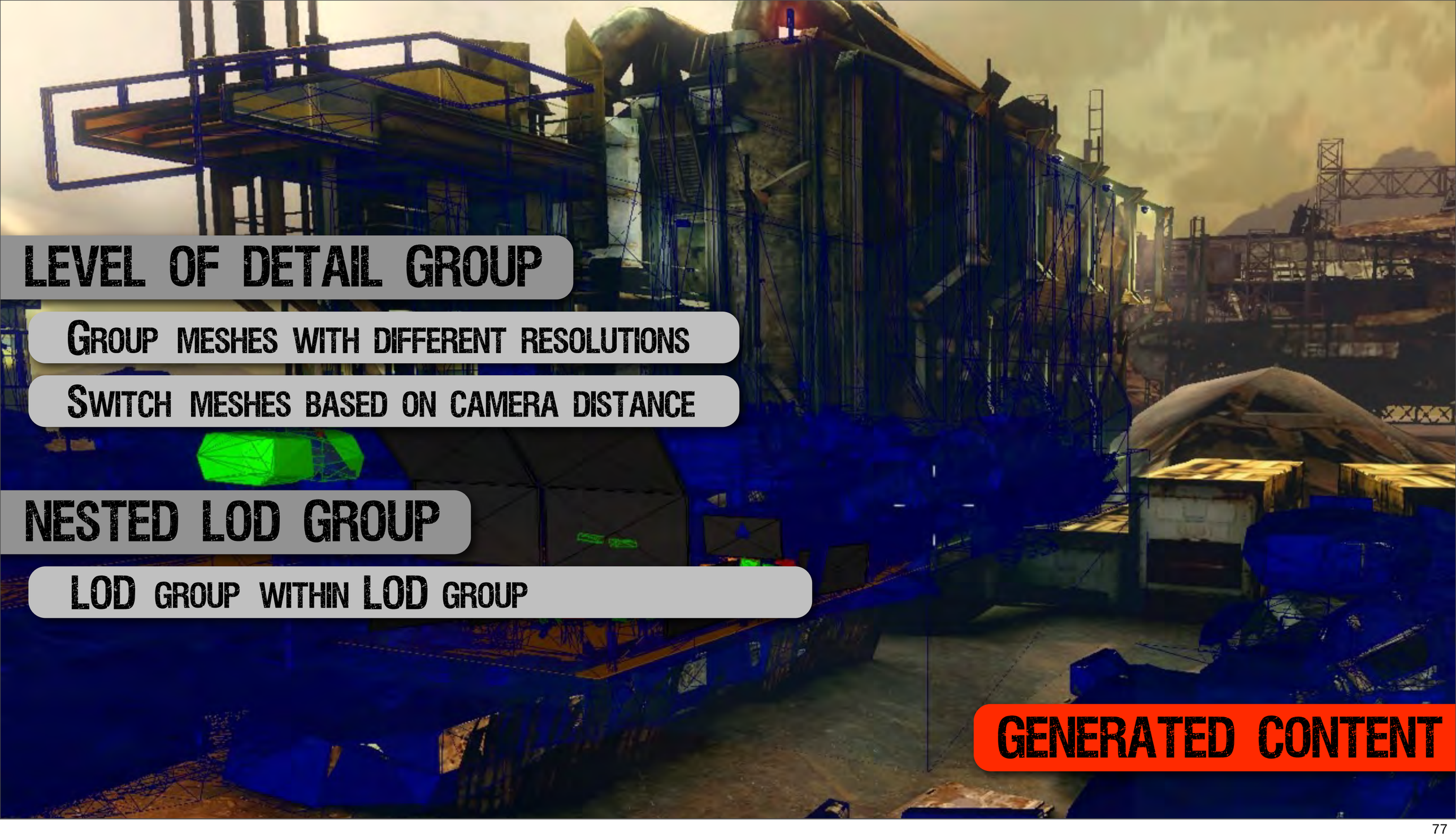
For both the collision meshes and the occlusion meshes we are able to extract the relevant data from the source building block during conversion and construct a combined mesh without the artist touching the content.

The reason why we combine these meshes into one is that it is faster for the engine to handle one big mesh then multiple small meshes.

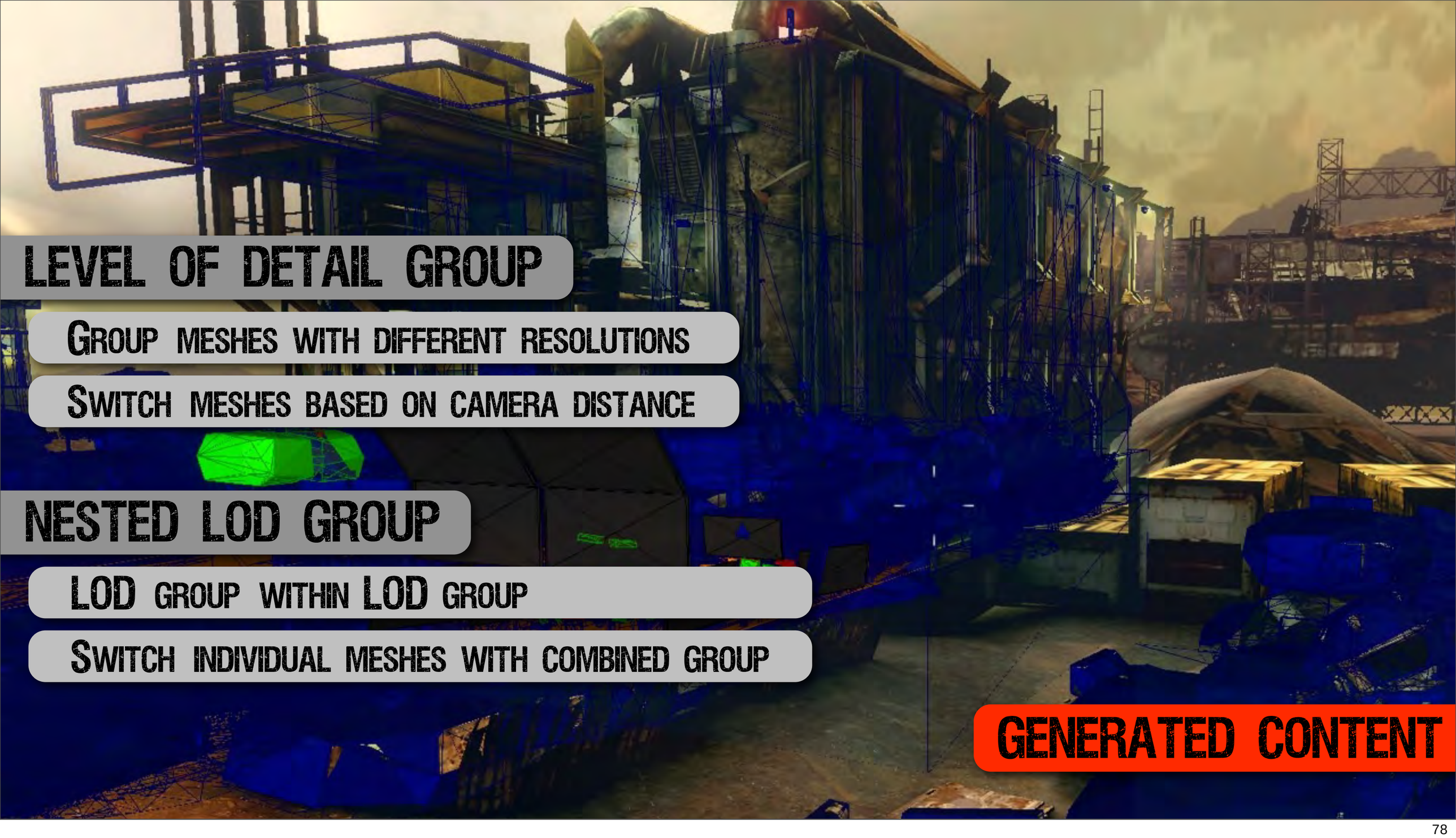
For the low resolution meshes the process is a bit more involved, and we use the conversion process to generate what we call “nested LOD’s”.



For those of you who unfamiliar with the term LOD, it stands for “level of detail”,
a LOD group is nothing more than a group of meshes with different resolutions that switch based on camera distance.



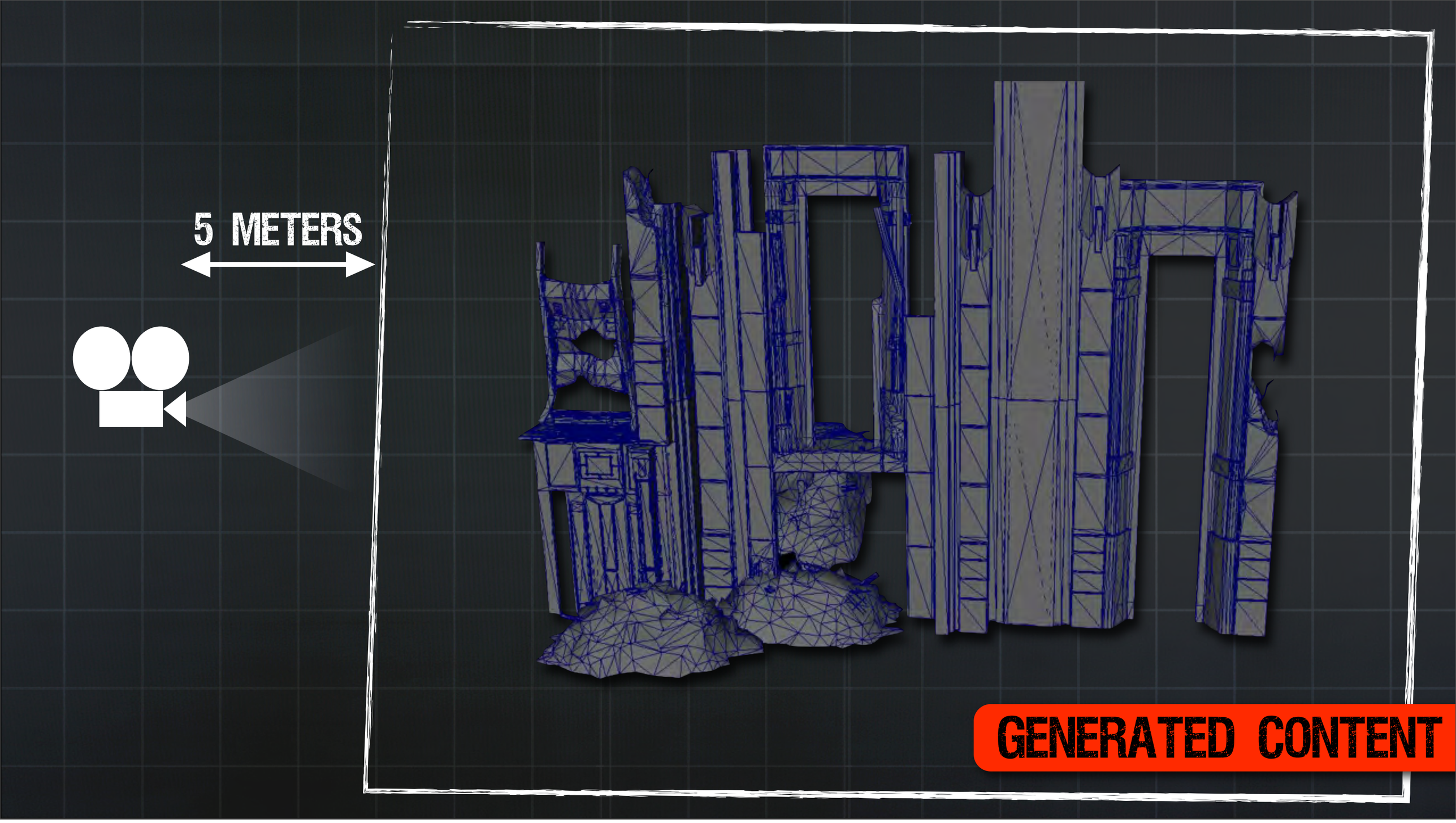
A nested LOD consists of a LOD group inside of a LOD group.



This means that when you are close to a group of building blocks you will see them as individual objects with their own LOD setup.

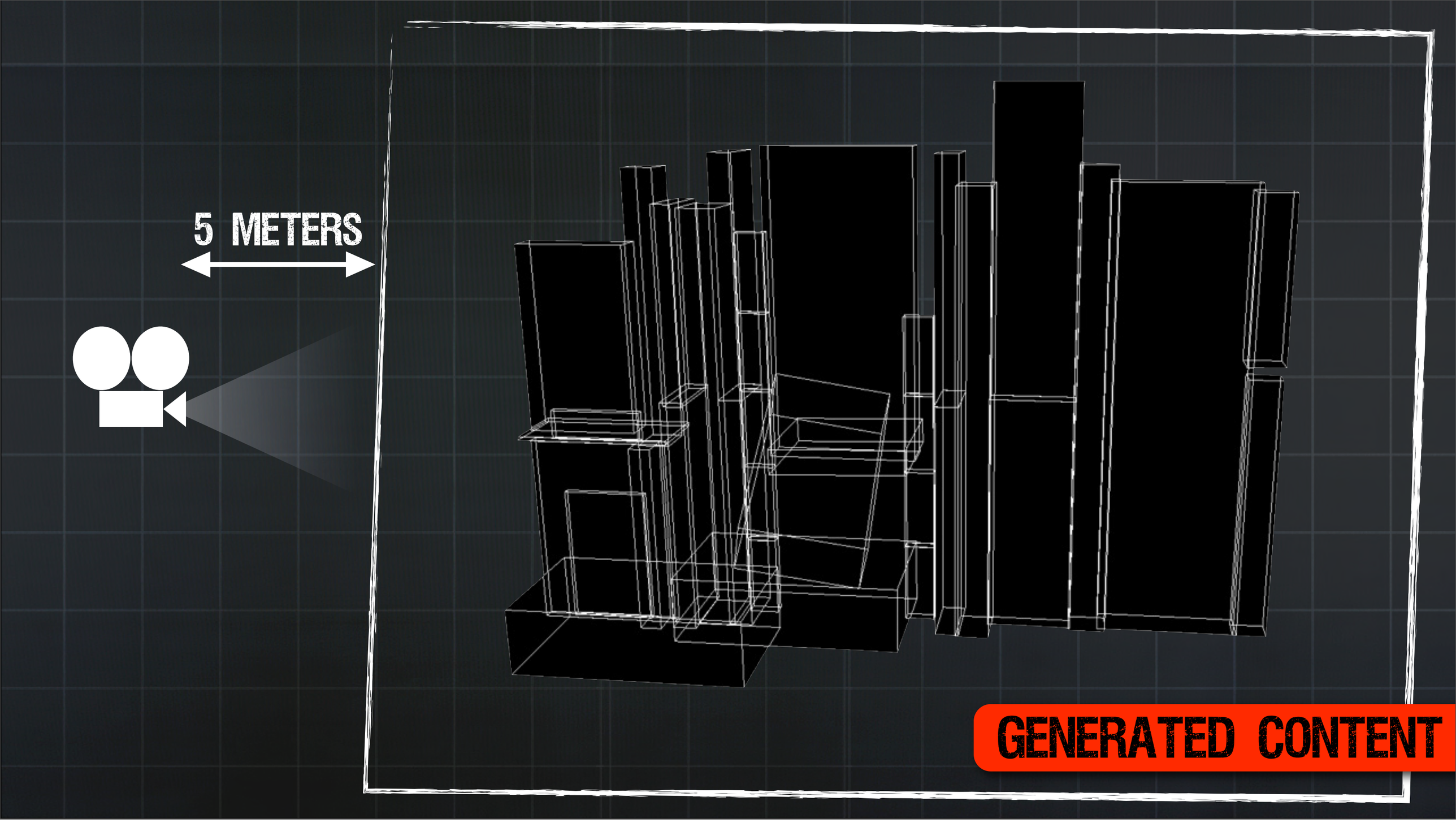
However when you are sufficiently far away from the whole group we remove all the individual building blocks and replace it with this combined mesh representation.

Let me try to illustrate this setup.

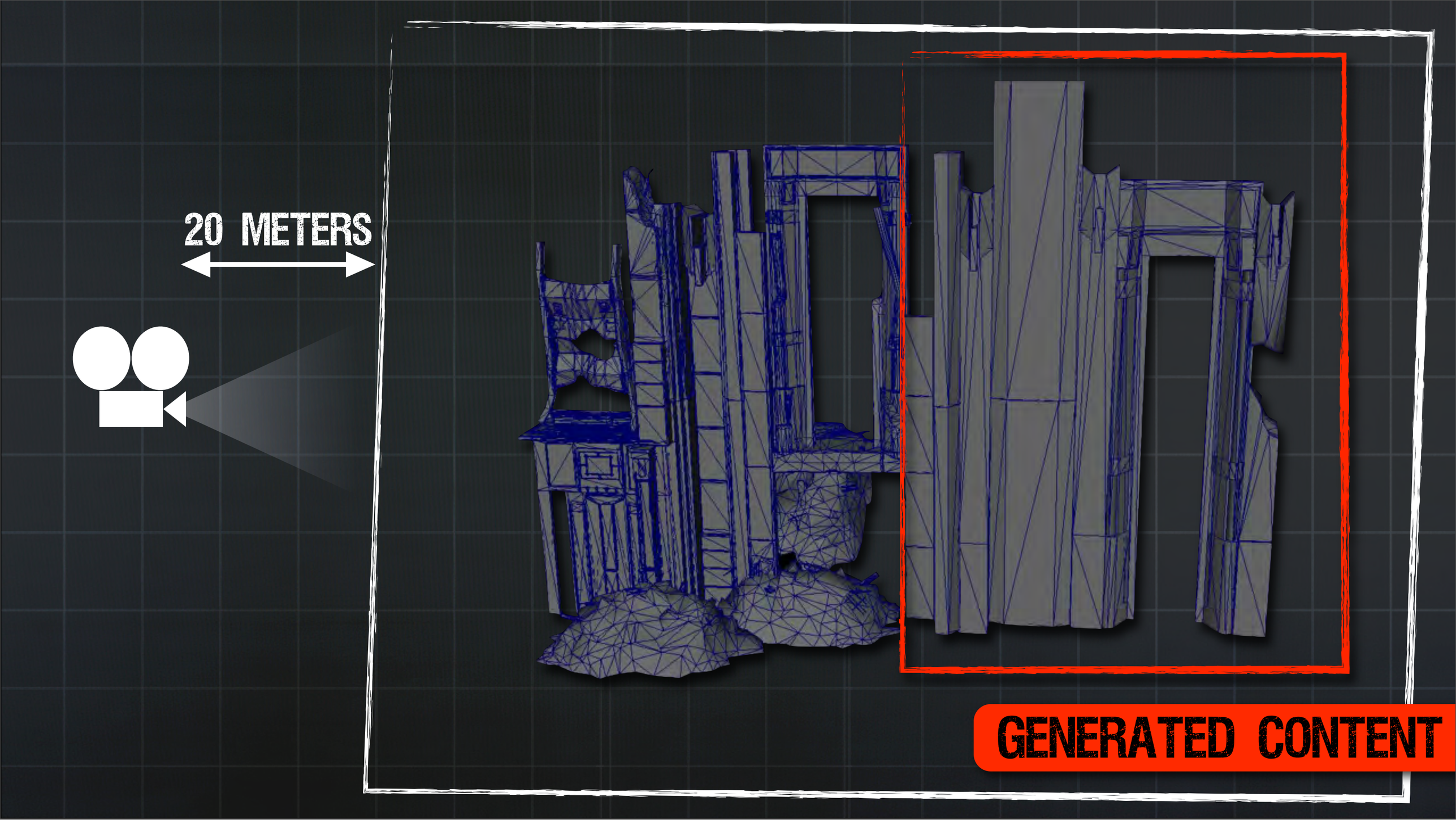


What you see here is a collection of building blocks that make a wall.

All these building blocks are separate objects and with the camera 5 meters away they will all be in their high resolution state

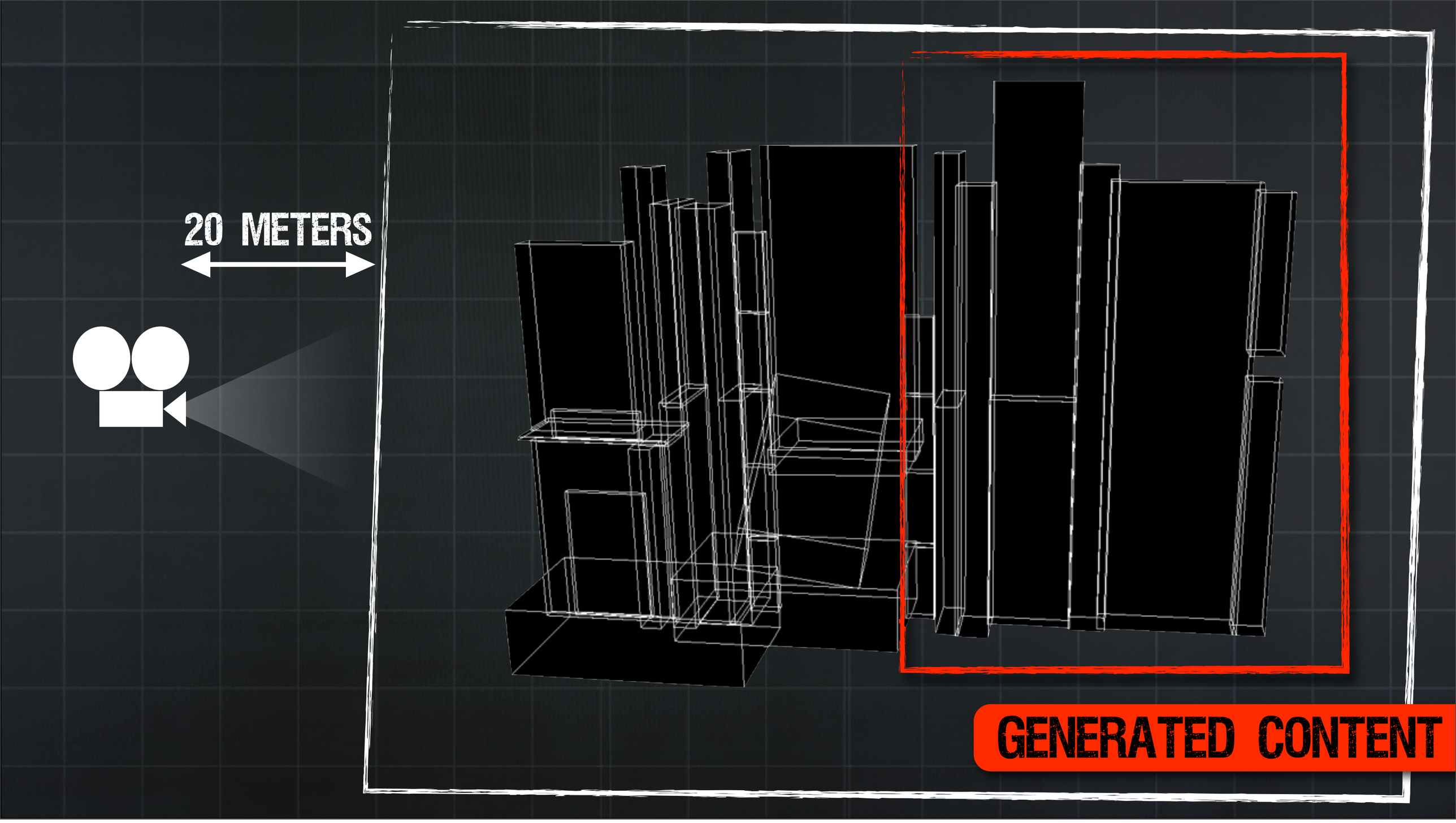


When we look at the bounding boxes you can see that they are indeed separate objects

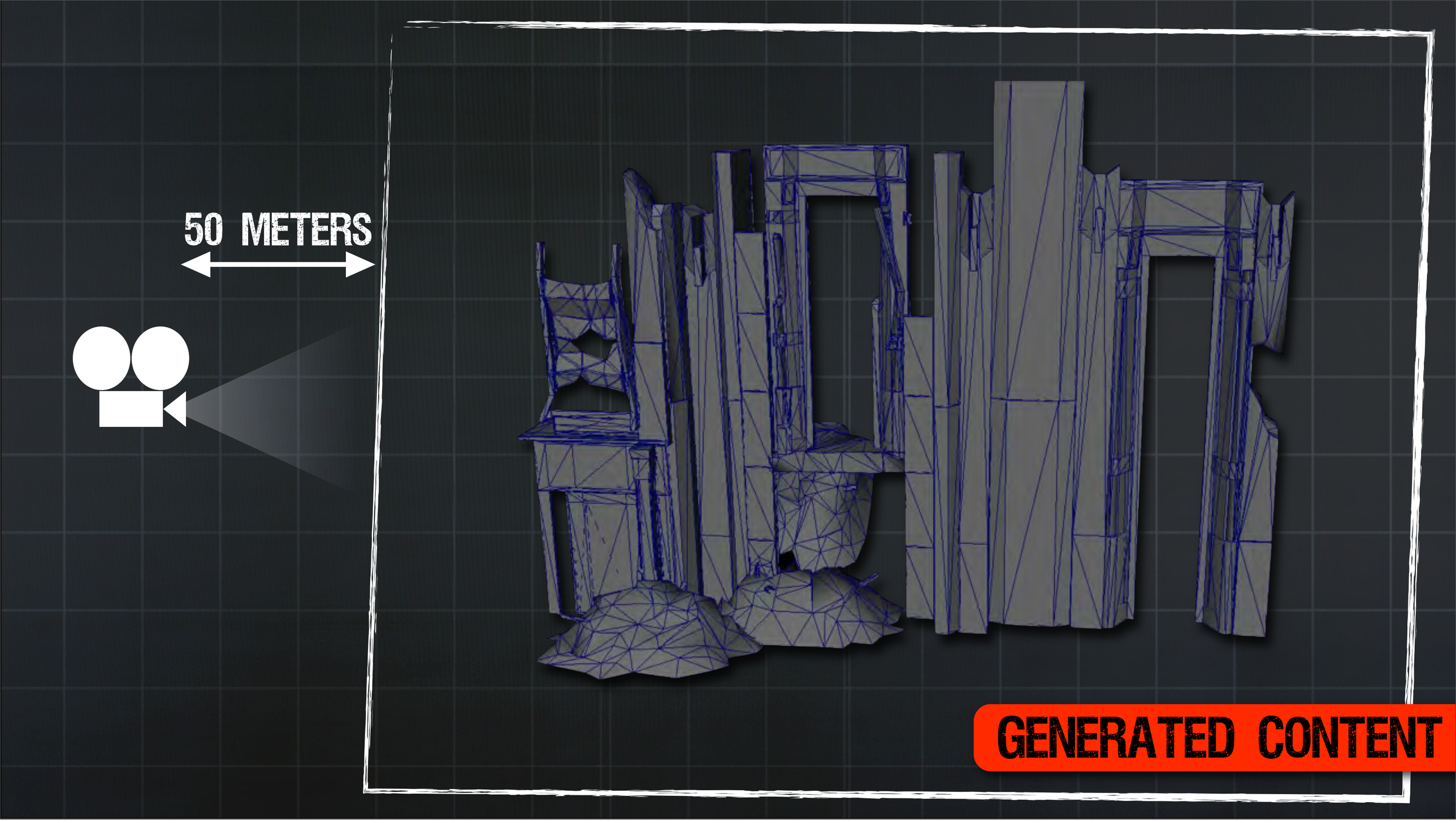


Now we move the camera further away and some of the building blocks at the far end of the collection will now swap to their low resolution state.

This way we are reducing the amount of polygons that we have to render.

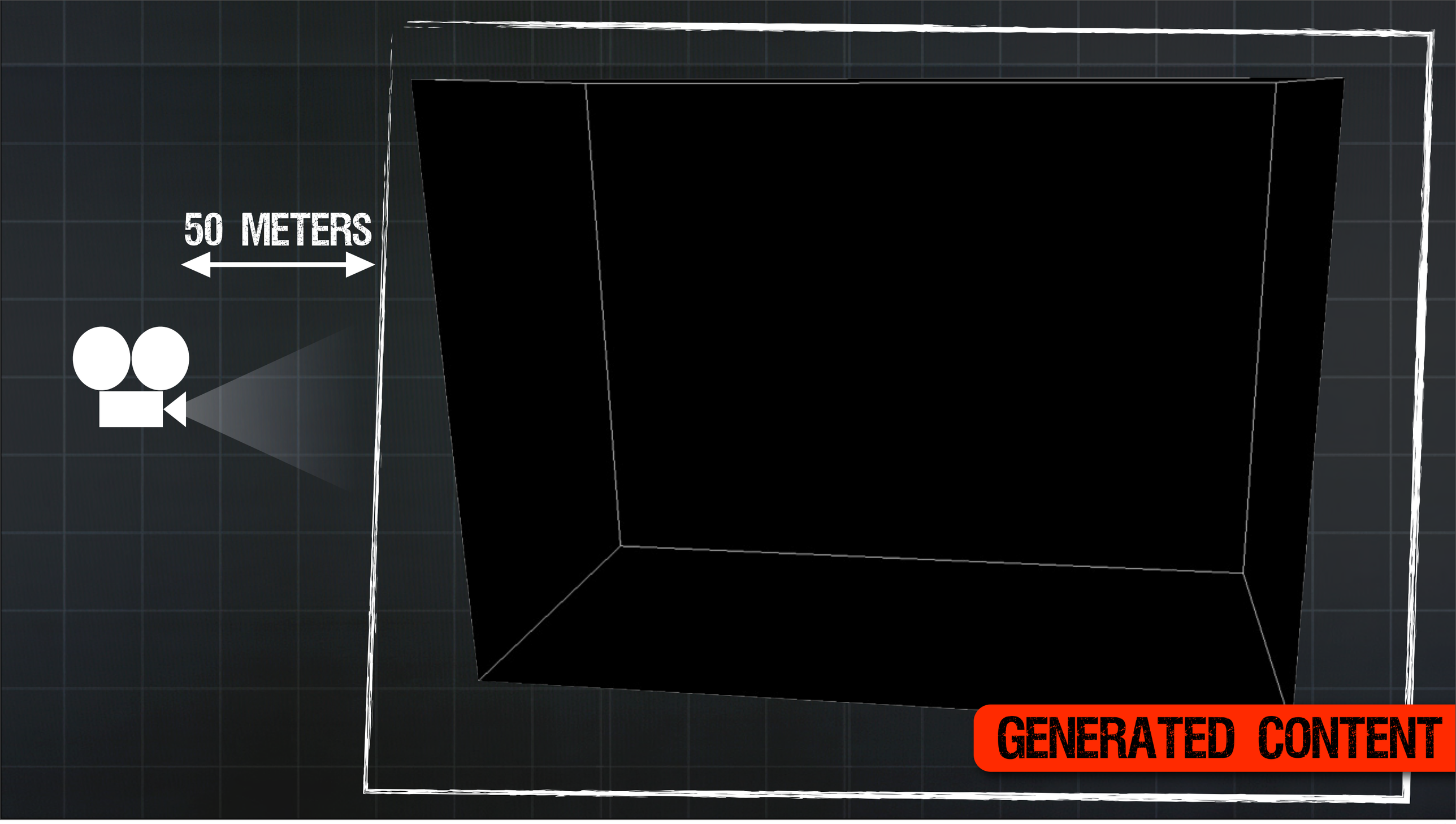


When we look at the bounding box again we see that they are still separate objects.



Now the camera is sufficiently far away for all the building blocks to be switched to the low resolution mesh.

Since they are all in the same state now it would be much more efficient to render them as a single object.



So when we look at the bounding box you can now see that we actually swapped out the whole collection of building blocks and replaced them with a single mesh.

The whole setup including the collapsed mesh is completely generated during our conversion process the only manual step is configuring the distances at which we switch the meshes.

Naturally there are environmental features that would be unpractical to create by instancing these generic objects.

Generally larger simple surfaces like walls or streets are so specific to their location that it’s a lot easier and faster to model them as unique geometry.

So this is what we do.

In an average a Killzone 3 level building blocks represent about 80% of the poly budget and 20% is used by unique geometry.

Instead of instancing this geometry we instance the shaders used by this geometry.

BUILDING BLOCK SHADER

UNIQUE GEOMETRY

Building Block shaders tend to be fairly simple because we can use the geometry much more to convey the nature object and by mixing and matching various building blocks it’s easy to create an interesting visual image.

This unique geometry is different, as I mentioned it’s usually used to define quite a large surface so shader detail is much more important.

This is why these shaders are created by specialised “shader artists” and usually contain multiple types of material that can be blended between by the artist creating the environment.

By instancing these shaders we not only make it possible for the shader artist and the environment artist to work in parallel, but also gain similar benefits to our building block pipeline.

We can for example use the conversion process to generate optimised version of the shader by removing things like normal-mapping and reflections for far-way shaders.

Finally, like the building blocks it allows us to optimise the shaders towards the end of production without touching or re-exporting any environment geometry.

88

As I mentioned before, this pipeline also helps a lot with our optimisation process.

No matter how hard you try, when any environment is finished it will always use too much memory and run too slow.

The image you see here is our performance statistics, basically “red is bad” and the circled number is the current frame rate;
not good.

So this is where our optimisation artists step in.

Since we can optimise very effectively by modifying the source assets of our pipeline (building blocks, shaders) it becomes important interesting to know which assets are used where and how many times.

AUTOBOT MP07OP - MP07 Operations - default | Memory Errors Performance Textures Shaders Meshes Physics Entities

Snapshot 2702 generated from content of 15 hours 3 minutes ago, changelist 1401224

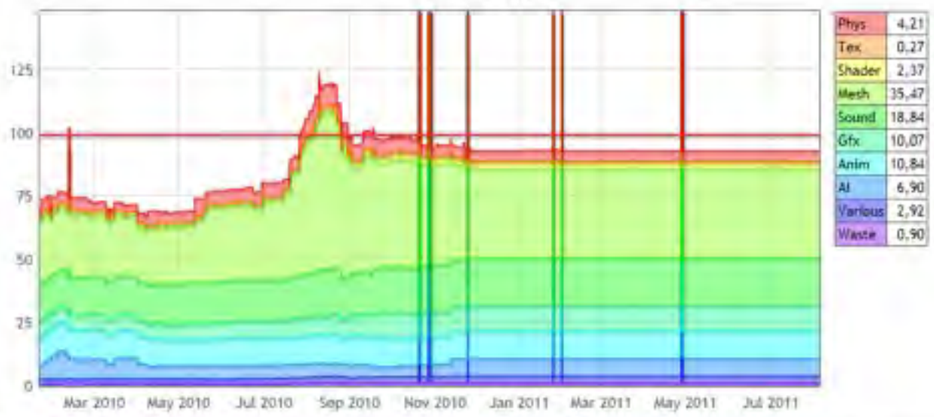
Vital stats

LevelName	Multi_Player/MP_07
SectionName	default
Configuration	Operations
Command Line	-level=Multi_Player/MP_07#Operations -universe_url=test:// -game_name=Test-Multi_Player/MP_07
Objects	55.743
Waypoints	5.032
Optimized assets	True
Development status	Polish
Waypoint Area	5,7 ha

Heap

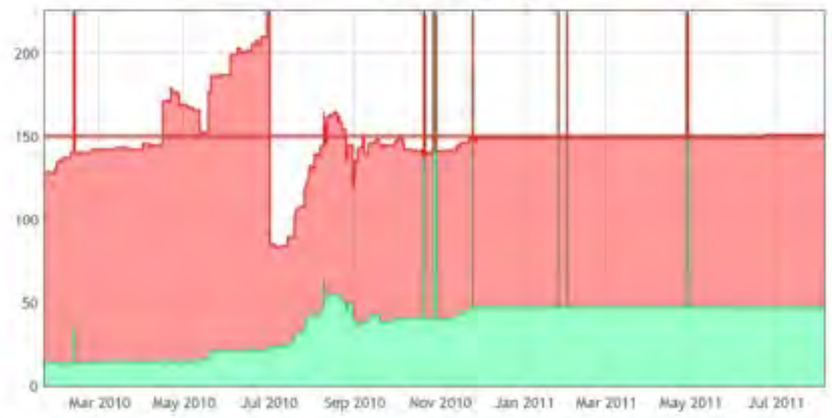
Heap on load	2.048,00
Heapblock on load	2.048,00
Heap on init	38,89
Heapblock on init	38,89
Heap on player spawn	2.048,00
Heapblock on player spawn	2.048,00
Heap in game	2.048,00
Heapblock in game	2.048,00

XDR Usage



Phys	4,21
Tex	0,27
Shader	2,37
Mesh	35,47
Sound	18,84
Gfx	10,07
Anim	10,84
AI	6,90
Various	2,92
Waste	0,90

VRAM Usage



Tex	103,40
Mesh	47,55

Date	20-jan 20:04	29-jun 19:20	26-aug 13:07	12-okt 11:44	13-nov 16:56	12-jan 13:05	29-mrt 12:23	1-aug 17:17	2-aug 15:09	3-aug 04:47	3-aug 12:16	3-aug 20:05	4-aug 04:46	23h 14m ago	15h 03m ago
Changelist	868002	1023670	1079966	1140719	1190722	1248399	1310758	1398327	1398995	1399304	1399598	1400321	1400397	1400697	1401224
Phys	3,61	5,47	6,59	5,88	4,37	4,21	4,21	4,21	4,21	4,21	4,21	4,21	4,21	4,21	4,21
Tex	0,21	0,22	0,25	0,24	0,26	0,27	0,27	0,27	0,27	0,27	0,27	0,27	0,27	0,27	0,27
Shader	1,42	1,52	2,00	2,56	2,40	2,52	2,37	2,37	2,37	2,37	2,37	2,37	2,37	2,37	2,37
Mesh	24,56	31,26	49,95	42,64	37,89	35,47	35,47	35,47	35,47	35,47	35,47	35,47	35,47	35,47	35,47
Sound	15,27	16,44	16,29	18,11	18,64	18,84	18,84	18,84	18,84	18,84	18,84	18,84	18,84	18,84	18,84
Gfx	5,89	6,64	7,48	10,02	9,89	10,10	10,10	10,07	10,07	10,07	10,07	10,07	10,07	10,07	10,07
Anim	11,37	10,54	9,63	10,33	10,83	10,84	10,84	10,84	10,84	10,84	10,84	10,84	10,84	10,84	10,84
AI	5,13	5,39	5,11	4,63	6,95	6,90	6,90	6,90	6,90	6,90	6,90	6,90	6,90	6,90	6,90
Various	1,94	2,14	2,46	2,55	2,86	2,91	2,91	2,92	2,92	2,92	2,92	2,92	2,92	2,92	2,92
Waste	0,69	0,72	1,06	0,98	0,91	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90	0,90
Total	70,10	80,33	100,80	97,95	95,01	92,95	92,80	92,78	92,78	92,78	92,78	92,78	92,78	92,78	92,78
Bins	71	81	101	98	95	93	93	93	93	93	93	93	93	93	93

Date	20-jan 20:04	29-jun 19:20	26-aug 13:07	12-okt 11:44	13-nov 16:56	12-jan 13:05	29-mrt 12:23	1-aug 17:17	2-aug 15:09	3-aug 04:47	3-aug 12:16	3-aug 20:05	4-aug 04:46	23h 14m ago	15h 03m ago
Changelist	868002	1023670	1079966	1140719	1190722	1248399	1310758	1398327	1398995	1399304	1399598	1400321	1400397	1400697	1401224
Tex	114,11	213,14	94,13	100,78	100,80	102,19	102,27	103,40	103,40	103,40	103,40	103,40	103,40	103,40	103,40
Mesh	14,20	23,56	44,72	40,76	43,36	47,55	47,55	47,55	47,55	47,55	47,55	47,55	47,55	47,55	47,55
Total	128,31	236,70	138,85	141,53	144,17	149,74	149,82	150,95	150,95	150,95	150,95	150,95	150,95	150,95	150,95

OPTIMISATION

89

To get to this information we have developed a server side tool that will constantly convert the game and generate reports.

These reports contain things like graphs about memory usage and frame-rate at predetermined locations in a level.

By tracking these changes over time it becomes relatively easy to see when and why destructive changes made their way into the game.

AUTOBOT SP01 - Flooded Pyrrhus - HGH Burning Memory Errors Performance Textures Shaders Meshes Physics Entities

Snapshot 2704 generated from content of 8 hours 40 minutes ago, changelist 1401369

BuildingBlock

Instances	6,516
TotalMin	452.202
TotalMax	2.159.253
LodRatio	20,94%
BaseFiles	109

BSP

Instances	63
TotalMin	34.592
TotalMax	95.257
LodRatio	36,31%

Prop

Instances	407
TotalMin	88.810
TotalMax	402.196
LodRatio	22,08%

LowCollapsed

Instances	1.452
TotalMin	675.619
TotalMax	4.678.147
LodRatio	14,44%

BaseFiles BuildingBlock BSP Prop LowCollapsed

Count	Name	Location	TrisMin	TrisMax	TotalMin ▼	TotalMax	LodRatio	Candidate	Savings	PhysTris	TotalPhys	PhysRatio
207	Debris_NUKED_b005_c002	models/building_blocks/urban_nuked/debris/components/debris_nuked_b005_c002_resource	264	774	54.648	160.218	34,11%	True	6.624	48	9.936	6,20%
111	Debris_b001_c015	models/building_blocks/scrapyard/debris/components/debris_b001_c015_resource	230	770	25.530	85.470	29,87%	False	0	12	1.332	1,56%
159	Steps_Nuked_b001_c002	models/building_blocks/urban_nuked/steps/components/steps_nuked_b001_c002_resource	126	464	20.352	73.776	27,59%	False	0	32	5.088	6,90%
231	Building_Dressing_b007_c002	models/building_blocks/urban/building_dressing/components/building_dressing_b007_c002_resource	68	142	15.708	32.802	47,89%	True	6.006	16	3.696	11,27%
64	Fence_NUKED_b001_c003	models/building_blocks/urban_nuked/fence/components/fence_nuked_b001_c003_resource	196	576	12.544	36.864	34,03%	True	1.536	12	768	2,08%
76	Shrapnel_Nuked_b001_c002	models/building_blocks/urban_nuked/shrapnel/components/shrapnel_nuked_b001_c002_resource	158	1.043	12.008	79.268	15,15%	False	0	78	5.928	7,48%
272	Column_NUKED_b008_c006	models/building_blocks/urban_nuked/columns/components/column_nuked_b008_c006_resource	40	136	10.880	36.992	29,41%	False	0	20	5.440	14,71%
51	Debris_b001_c016	models/building_blocks/scrapyard/debris/components/debris_b001_c016_resource	200	668	10.200	34.068	29,94%	False	0	0	0	0,00%
65	Broken_Wall_b002_c001	models/building_blocks/urban_nuked/walls/components/broken_wall_b002_c001_resource	155	432	10.075	28.080	35,88%	True	1.690	39	2.535	9,03%
149	Column_NUKED_b013_c001	models/building_blocks/urban_nuked/columns/components/column_nuked_b013_c001_resource	60	316	8.940	47.084	18,99%	False	0	12	1.788	3,80%
49	Column_NUKED_b007_c001	models/building_blocks/urban_nuked/columns/components/column_nuked_b007_c001_resource	174	1.697	8.526	83.153	10,25%	False	0	146	7.154	8,60%
21	Generic_Plate_NUKED_b018_c002	models/building_blocks/urban_nuked/generic_plating/components/generic_plate_nuked_b018_c002_resource	340	730	7.140	15.330	46,58%	True	2.541	114	2.394	15,62%
35	Rocks_SCRAPYARD_b001_c005	models/building_blocks/scrapyard/rocks/components/rocks_scrapyard_b001_c005_resource	168	1.030	5.880	36.050	16,31%	False	0	66	2.310	6,41%
284	Building_Dressing_NUKED_b016_c001	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b016_c001_resource	20	210	5.680	59.640	9,52%	False	0	20	5.680	9,52%
57	Column_NUKED_b013_c002	models/building_blocks/urban_nuked/columns/components/column_nuked_b013_c002_resource	96	360	5.586	20.520	27,22%	False	0	32	1.824	8,89%
33	Full_Cover_Wall_NUKED_b001_c010	models/building_blocks/urban_nuked/walls/components/full_cover_wall_nuked_b001_c010_resource	164	924	5.412	30.492	17,75%	False	0	87	2.871	9,42%
64	Floor_Tiles_NUKED_b002_c020	models/building_blocks/urban_nuked/tiles/components/floor_tiles_nuked_b002_c020_resource	84	465	5.376	29.760	18,06%	False	0	7	448	1,51%
132	Framework_Nuked_b004_c002	models/building_blocks/urban_nuked/framework/components/framework_nuked_b004_c002_resource	40	92	5.280	12.144	43,48%	False	0	12	1.584	13,04%
165	Building_Dressing_NUKED_b007_c008	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b007_c008_resource	32	710	5.280	117.150	4,51%	False	0	24	3.960	3,38%
22	Building_Dressing_NUKED_b014_c001	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b014_c001_resource	230	1.036	5.060	22.792	22,20%	False	0	24	528	2,32%
173	Shutter_b010_c002	models/building_blocks/urban/shutters/components/shutter_b010_c002_resource	26	76	4.498	13.148	34,21%	False	0	34	5.882	44,74%
120	Shrapnel_Nuked_b001_c009	models/building_blocks/urban_nuked/shrapnel/components/shrapnel_nuked_b001_c009_resource	37	124	4.440	14.880	29,84%	False	0	30	3.600	24,19%
171	Building_Dressing_Nuked_b015_c011	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b015_c011_resource	24	108	4.104	18.468	22,22%	False	0	12	2.052	11,11%
59	Windows_Nuked_b009_c001	models/building_blocks/urban_nuked/windows/components/windows_nuked_b009_c001_resource	68	326	4.012	19.234	20,86%	False	0	48	2.832	14,72%
18	Pipes_b006_c001	models/building_blocks/urban/pipes/components/pipes_b006_c001_resource	217	1.222	3.906	21.996	17,76%	False	0	10	180	0,82%
44	Full_Cover_Wall_NUKED_b001_c009	models/building_blocks/urban_nuked/walls/components/full_cover_wall_nuked_b001_c009_resource	82	626	3.608	27.544	13,10%	False	0	44	1.936	7,03%
19	Broken_Wall_b003_c002	models/building_blocks/urban_nuked/walls/components/broken_wall_b003_c002_resource	182	1.932	3.458	36.708	9,42%	False	0	32	608	1,66%
108	Building_Dressing_Nuked_b015_c001	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b015_c001_resource	32	138	3.456	14.904	23,19%	False	0	12	1.296	8,70%
11	Shrapnel_Nuked_b001_c003	models/building_blocks/urban_nuked/shrapnel/components/shrapnel_nuked_b001_c003_resource	314	846	3.454	9.306	37,12%	True	671	130	1.430	15,37%
40	Railing_b005_c003	models/building_blocks/urban/railings/components/railing_b005_c003_resource	82	468	3.280	18.720	17,52%	False	0	10	400	2,14%
25	Broken_Wall_b003_c003	models/building_blocks/urban_nuked/walls/components/broken_wall_b003_c003_resource	130	1.160	3.250	29.000	11,21%	False	0	16	400	1,38%
29	Shrapnel_Nuked_b001_c001	models/building_blocks/urban_nuked/shrapnel/components/shrapnel_nuked_b001_c001_resource	108	616	3.132	17.864	17,53%	False	0	128	3.712	20,78%
71	Windows_b009_c001	models/building_blocks/urban/windows/components/windows_b009_c001_resource	44	400	3.124	28.400	11,00%	False	0	22	1.562	5,50%
65	Shutter_Nuked_b010_c002	models/building_blocks/urban_nuked/shutters/components/shutter_nuked_b010_c002_resource	48	128	3.120	8.320	37,50%	True	650	20	1.300	15,63%
29	Column_NUKED_b008_c012	models/building_blocks/urban_nuked/columns/components/column_nuked_b008_c012_resource	107	788	3.103	22.852	13,58%	False	0	60	1.740	7,61%
11	Fence_b001_c001	models/building_blocks/urban/fence/components/fence_b001_c001_resource	279	576	3.069	6.336	48,44%	True	1.177	12	132	2,08%
14	Floor_NUKED_b002_c004	models/building_blocks/urban_nuked/floor/components/floor_nuked_b002_c004_resource	214	934	2.996	13.076	22,91%	False	0			
35	Building_Dressing_NUKED_b014_c003	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b014_c003_resource	84	270	2.940	9.450	31,11%	True	105			
28	Building_Sets_Framework_NUKED_b002_c014	models/building_blocks/urban_nuked/building_sets/components/building_sets_framework_nuked_b002_c014_resource	96	153	2.688	4.284	62,75%	True	1.428			
5	Shutter_Nuked_b008_c011	models/building_blocks/urban_nuked/shutters/components/shutter_nuked_b008_c011_resource	530	1.074	2.650	5.370	49,35%	True	1.040			
88	Building_Dressing_NUKED_b008_c008	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b008_c008_resource	30	168	2.640	14.784	17,86%	False	0			
41	Door_NUKED_b014_c004	models/building_blocks/urban_nuked/doors/components/door_nuked_b014_c004_resource	60	1.080	2.580	46.440	5,56%	False	0	20	860	1,85%
9	Fence_b001_c010	models/building_blocks/urban/fence/components/fence_b001_c010_resource	254	486	2.286	4.374	52,26%	True	981	12	108	2,47%
10	Statue_b001_c002	models/building_blocks/urban_nuked/statues/components/statue_b001_c002_resource	227	960	2.270	9.600	23,65%	False	0	80	800	8,33%
102	Ledges_NUKED_b002_c002	models/building_blocks/urban_nuked/ledges/components/ledges_nuked_b002_c002_resource	23	134	2.244	13.648	17,74%	False	0	8	816	6,45%

OPTIMISATION

Say that we need to remove some polygons to get the memory usage under control again.

It’s much more efficient to optimise a building block that occurs 300 times than a building block that only occurs once.

AUTOBOT SP01 - Flooded Pyrrhus - HGH Burning Memory Errors Performance Textures Shaders Meshes Physics Entities

Snapshot 2704 generated from content of 8 hours 40 minutes ago, changelist 1401369

BuildingBlock

Instances	6.516
TotalMin	452.202
TotalMax	2.159.253
LodRatio	20,94%

BSP

Instances	63
TotalMin	34.592
TotalMax	95.257
LodRatio	36,31%

Prop

Instances	407
TotalMin	88.810
TotalMax	402.196
LodRatio	22,08%

LowCollapsed

Instances	1.452
TotalMin	675.619
TotalMax	4.678.147
LodRatio	14,44%

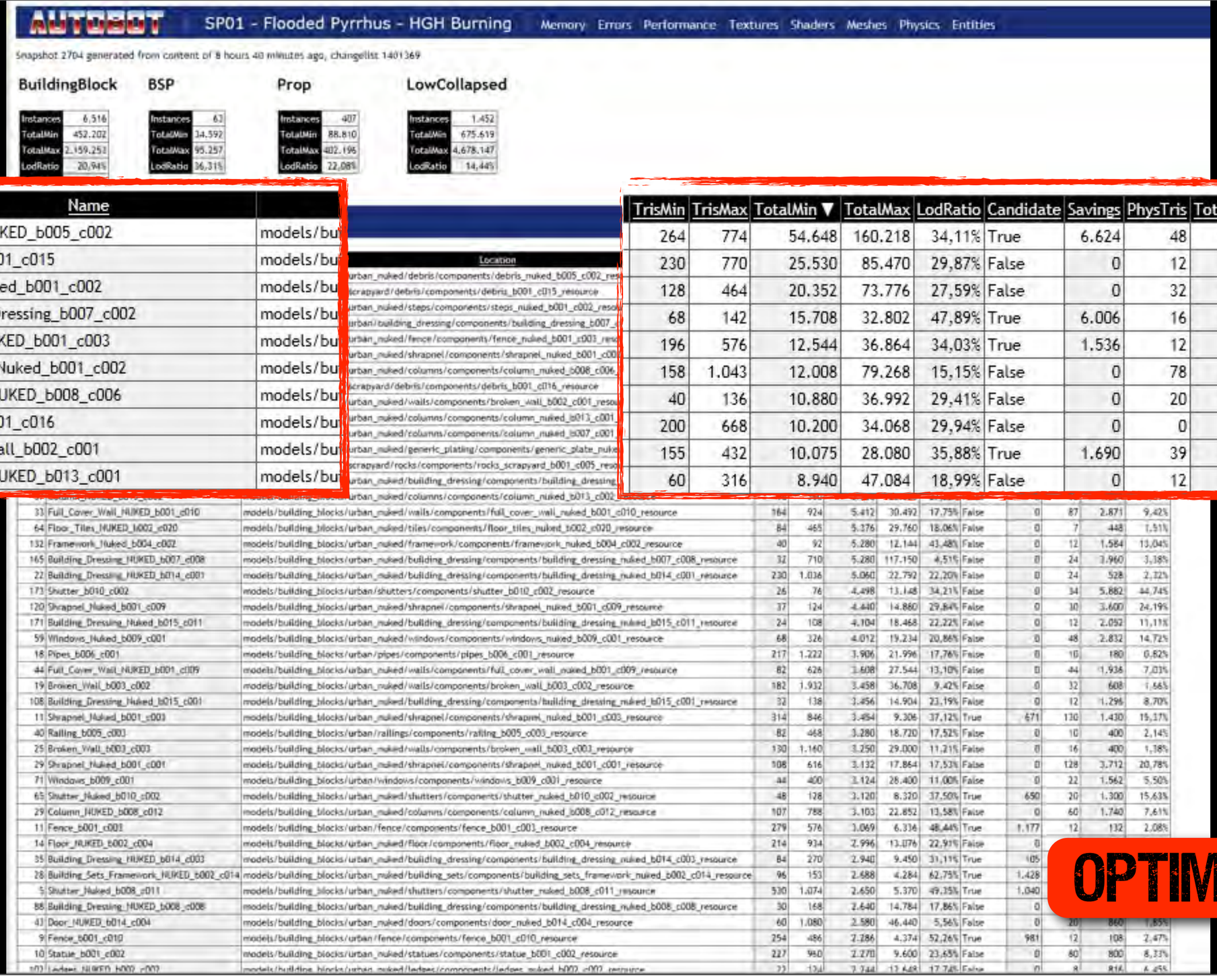
Count	Name	
207	Debris_NUKED_b005_c002	models/bui
111	Debris_b001_c015	models/bui
159	Steps_Nuked_b001_c002	models/bui
231	Building_Dressing_b007_c002	models/bui
64	Fence_NUKED_b001_c003	models/bui
76	Shrapnel_Nuked_b001_c002	models/bui
272	Column_NUKED_b008_c006	models/bui
51	Debris_b001_c016	models/bui
65	Broken_Wall_b002_c001	models/bui
149	Column_NUKED_b013_c001	models/bui

		Location	TrisMin	TrisMax	TotalMin ▼	TotalMax	LodRatio	Candidate	Savings	PhysTris	TotalPhys	PhysRatio
		/urban_nuked/debris/components/debris_nuked_b005_c002_resource	264	774	54.648	160.218	34,11%	True	6.624	48	9.936	6,20%
		/scrapyard/debris/components/debris_b001_c015_resource	230	770	25.530	85.470	29,87%	False	0	12	1.332	1,56%
		/urban_nuked/steps/components/steps_nuked_b001_c002_resource	126	464	20.352	73.776	27,59%	False	0	12	5.088	6,90%
		/urban/building_dressing/components/building_dressing_b007_c002_resource	68	142	15.708	32.802	47,89%	True	6.006	16	3.696	11,27%
		/urban_nuked/fence/components/fence_nuked_b001_c003_resource	196	576	12.544	36.864	34,03%	True	1.536	12	768	2,08%
		/urban_nuked/shrapnel/components/shrapnel_nuked_b001_c002_resource	158	1.043	12.008	79.268	15,15%	False	0	78	5.928	7,48%
		/urban_nuked/columns/components/column_nuked_b008_c006_resource	40	136	10.880	36.992	29,41%	False	0	20	5.440	14,71%
		/scrapyard/debris/components/debris_b001_c016_resource	200	668	10.200	34.068	29,94%	False	0	0	0	0,00%
		/urban_nuked/walls/components/broken_wall_b002_c001_resource	155	432	10.075	28.080	35,88%	True	1.690	39	2.535	9,03%
		/urban_nuked/columns/components/column_nuked_b013_c001_resource	60	316	8.940	47.084	18,99%	False	0	12	1.788	3,80%
		/urban_nuked/columns/components/column_nuked_b007_c001_resource	174	1.697	8.526	83.153	10,25%	False	0	146	7.154	8,60%
		/urban_nuked/generic_plating/components/generic_plate_nuked_b018_c002_resource	340	730	7.140	15.330	46,58%	True	2.541	114	2.394	15,62%
		/scrapyard/rocks/components/rocks_scrapyard_b001_c005_resource	168	1.030	5.880	36.050	16,31%	False	0	66	2.310	6,41%
		/urban_nuked/building_dressing/components/building_dressing_nuked_b016_c001_resource	20	210	5.680	59.640	9,52%	False	0	20	5.680	9,52%
57	Column_NUKED_b013_c002	models/building_blocks/urban_nuked/columns/components/column_nuked_b013_c002_resource	96	360	5.586	20.520	27,22%	False	0	32	1.824	8,89%
33	Full_Cover_Wall_NUKED_b001_c010	models/building_blocks/urban_nuked/walls/components/full_cover_wall_nuked_b001_c010_resource	164	924	5.412	30.492	17,75%	False	0	87	2.871	9,42%
64	Floor_Tiles_NUKED_b002_c020	models/building_blocks/urban_nuked/tiles/components/floor_tiles_nuked_b002_c020_resource	84	465	5.376	29.760	18,06%	False	0	7	448	1,51%
132	Framework_Nuked_b004_c002	models/building_blocks/urban_nuked/framework/components/framework_nuked_b004_c002_resource	40	92	5.280	12.144	43,48%	False	0	12	1.584	13,04%
165	Building_Dressing_NUKED_b007_c008	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b007_c008_resource	32	710	5.280	117.150	4,51%	False	0	24	3.960	3,38%
22	Building_Dressing_NUKED_b014_c001	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b014_c001_resource	230	1.036	5.060	22.792	22,20%	False	0	24	528	2,32%
173	Shutter_b010_c002	models/building_blocks/urban/shutters/components/shutter_b010_c002_resource	26	76	4.498	13.148	34,21%	False	0	34	5.882	44,74%
120	Shrapnel_Nuked_b001_c009	models/building_blocks/urban_nuked/shrapnel/components/shrapnel_nuked_b001_c009_resource	37	124	4.440	14.880	29,84%	False	0	30	3.600	24,19%
171	Building_Dressing_Nuked_b015_c011	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b015_c011_resource	24	108	4.104	18.468	22,22%	False	0	12	2.052	11,11%
59	Windows_Nuked_b009_c001	models/building_blocks/urban_nuked/windows/components/windows_nuked_b009_c001_resource	68	326	4.012	19.234	20,86%	False	0	48	2.832	14,72%
18	Pipes_b006_c001	models/building_blocks/urban/pipes/components/pipes_b006_c001_resource	217	1.222	3.906	21.996	17,76%	False	0	10	180	0,82%
44	Full_Cover_Wall_NUKED_b001_c009	models/building_blocks/urban_nuked/walls/components/full_cover_wall_nuked_b001_c009_resource	82	626	3.608	27.544	13,10%	False	0	44	1.936	7,03%
19	Broken_Wall_b003_c002	models/building_blocks/urban_nuked/walls/components/broken_wall_b003_c002_resource	182	1.932	3.458	36.708	9,42%	False	0	32	608	1,66%
108	Building_Dressing_Nuked_b015_c001	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b015_c001_resource	32	138	3.456	14.904	23,19%	False	0	12	1.296	8,70%
11	Shrapnel_Nuked_b001_c003	models/building_blocks/urban_nuked/shrapnel/components/shrapnel_nuked_b001_c003_resource	314	846	3.454	9.306	37,12%	True	671	130	1.430	15,37%
40	Railing_b005_c003	models/building_blocks/urban/railings/components/railing_b005_c003_resource	82	468	3.280	18.720	17,52%	False	0	10	400	2,14%
25	Broken_Wall_b003_c003	models/building_blocks/urban_nuked/walls/components/broken_wall_b003_c003_resource	130	1.160	3.250	29.000	11,21%	False	0	16	400	1,38%
29	Shrapnel_Nuked_b001_c001	models/building_blocks/urban_nuked/shrapnel/components/shrapnel_nuked_b001_c001_resource	108	616	3.132	17.864	17,53%	False	0	128	3.712	20,78%
71	Windows_b009_c001	models/building_blocks/urban/windows/components/windows_b009_c001_resource	44	400	3.124	28.400	11,00%	False	0	22	1.562	5,50%
65	Shutter_Nuked_b010_c002	models/building_blocks/urban_nuked/shutters/components/shutter_nuked_b010_c002_resource	48	128	3.120	8.320	37,50%	True	650	20	1.300	15,63%
29	Column_NUKED_b008_c012	models/building_blocks/urban_nuked/columns/components/column_nuked_b008_c012_resource	107	788	3.103	22.852	13,58%	False	0	60	1.740	7,61%
11	Fence_b001_c001	models/building_blocks/urban/fence/components/fence_b001_c001_resource	279	576	3.069	6.336	48,44%	True	1.177	12	132	2,08%
14	Floor_NUKED_b002_c004	models/building_blocks/urban_nuked/floor/components/floor_nuked_b002_c004_resource	214	934	2.996	13.076	22,91%	False	0			
35	Building_Dressing_NUKED_b014_c003	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b014_c003_resource	84	270	2.940	9.450	31,11%	True	105			
28	Building_Sets_Framework_NUKED_b002_c014	models/building_blocks/urban_nuked/building_sets/components/building_sets_framework_nuked_b002_c014_resource	96	153	2.688	4.284	62,75%	True	1.428			
5	Shutter_Nuked_b008_c011	models/building_blocks/urban_nuked/shutters/components/shutter_nuked_b008_c011_resource	530	1.074	2.650	5.370	49,35%	True	1.040			
88	Building_Dressing_NUKED_b008_c008	models/building_blocks/urban_nuked/building_dressing/components/building_dressing_nuked_b008_c008_resource	30	168	2.640	14.784	17,86%	False	0			
41	Door_NUKED_b014_c004	models/building_blocks/urban_nuked/doors/components/door_nuked_b014_c004_resource	60	1.080	2.580	46.440	5,56%	False	0	20	860	1,85%
9	Fence_b001_c010	models/building_blocks/urban/fence/components/fence_b001_c010_resource	254	486	2.286	4.374	52,26%	True	981	12	108	2,47%
10	Statue_b001_c002	models/building_blocks/urban_nuked/statues/components/statue_b001_c002_resource	227	960	2.270	9.600	23,65%	False	0	80	800	8,33%
102	Ledges_NUKED_b002_c002	models/building_blocks/urban_nuked/ledges/components/ledges_nuked_b002_c002_resource	23	134	2.244	13.668	17,74%	False	0	8	816	6,45%

OPTIMISATION

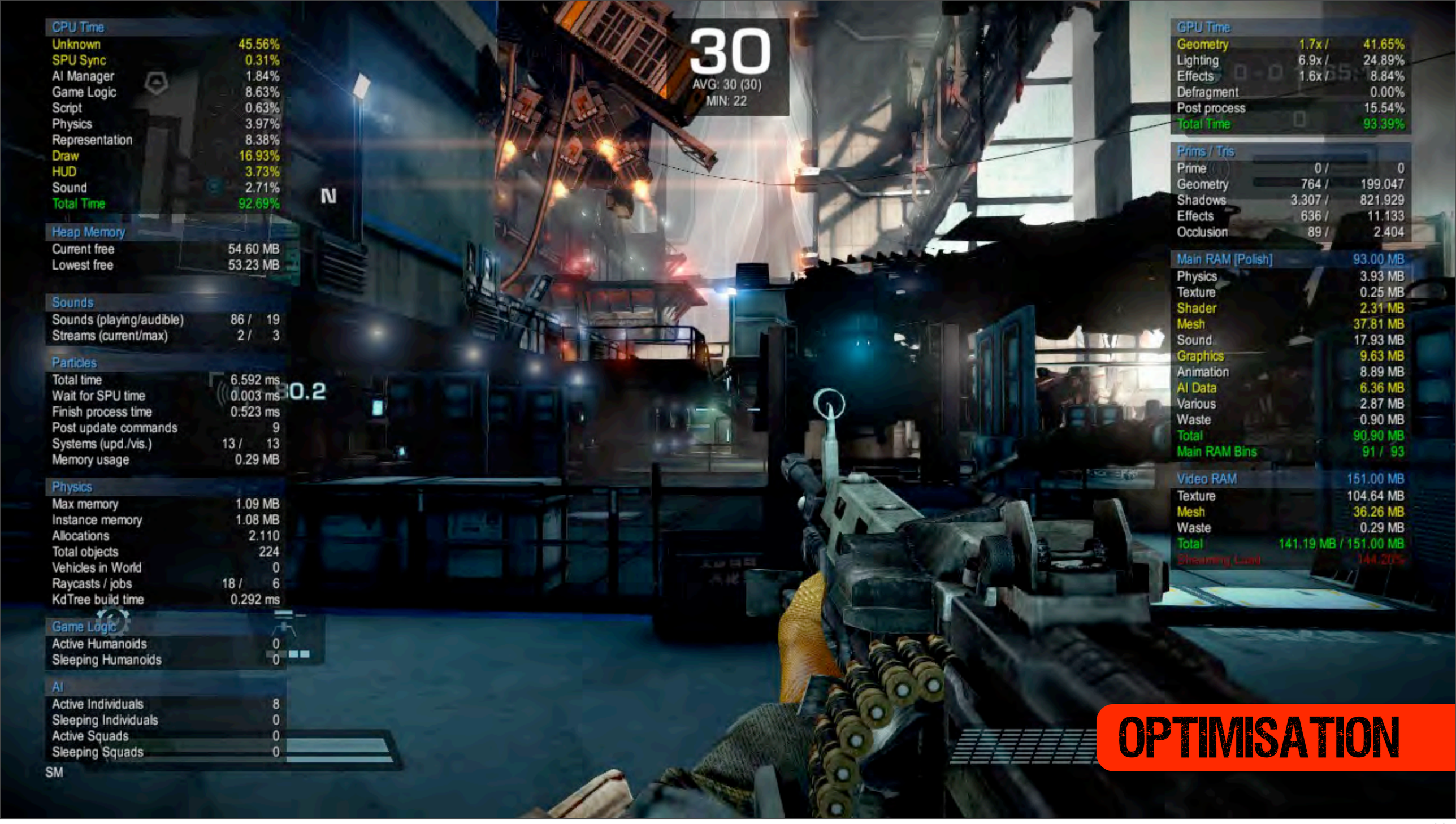
To get to this information the user can drill down into the level specific data and get lists of the used building blocks and even individual statistics per building block.

Here you see that we list the occurrence count for each building block in this level...



And here you can see the individual statistics like polycount, LOD ratio and many more.

By using this data we can make sure that we are targeting the areas of the game where an actual difference can be made without unnecessarily sacrificing quality.



And this is what the performance statistics should look like before we can release the game.

That’s it for me talking about our environment creation process,
I hope you enjoyed it and I’ll now hand you over to my colleague Marijn Giesbertz who will talk about interactivity in Killzone 3.

Thank you.

LEAD EFFECTS

MARIJN GIESBERTZ

Hi, my name is Marijn Giesbertz and
I'm Lead visual FX @ guerrilla games and
I'll be talking about interactivity in our game
with a focus on our AI, animation and particle system.

Let's start of by making a quick comparison between Movies and video-games.
and you'll notice that there are at least some simularities

GAMES VS MOVIES

We both make use of a **set**....

lights.....

cameras....

actors.....

GAMES VS MOVIES

director.....

but there is 1 crucial difference between our worlds.... we have 1 actor on set that we're not able to control... **the player**

..... or gamer..... and in case of a multi-player game we have even more of these uncontrollable actors!

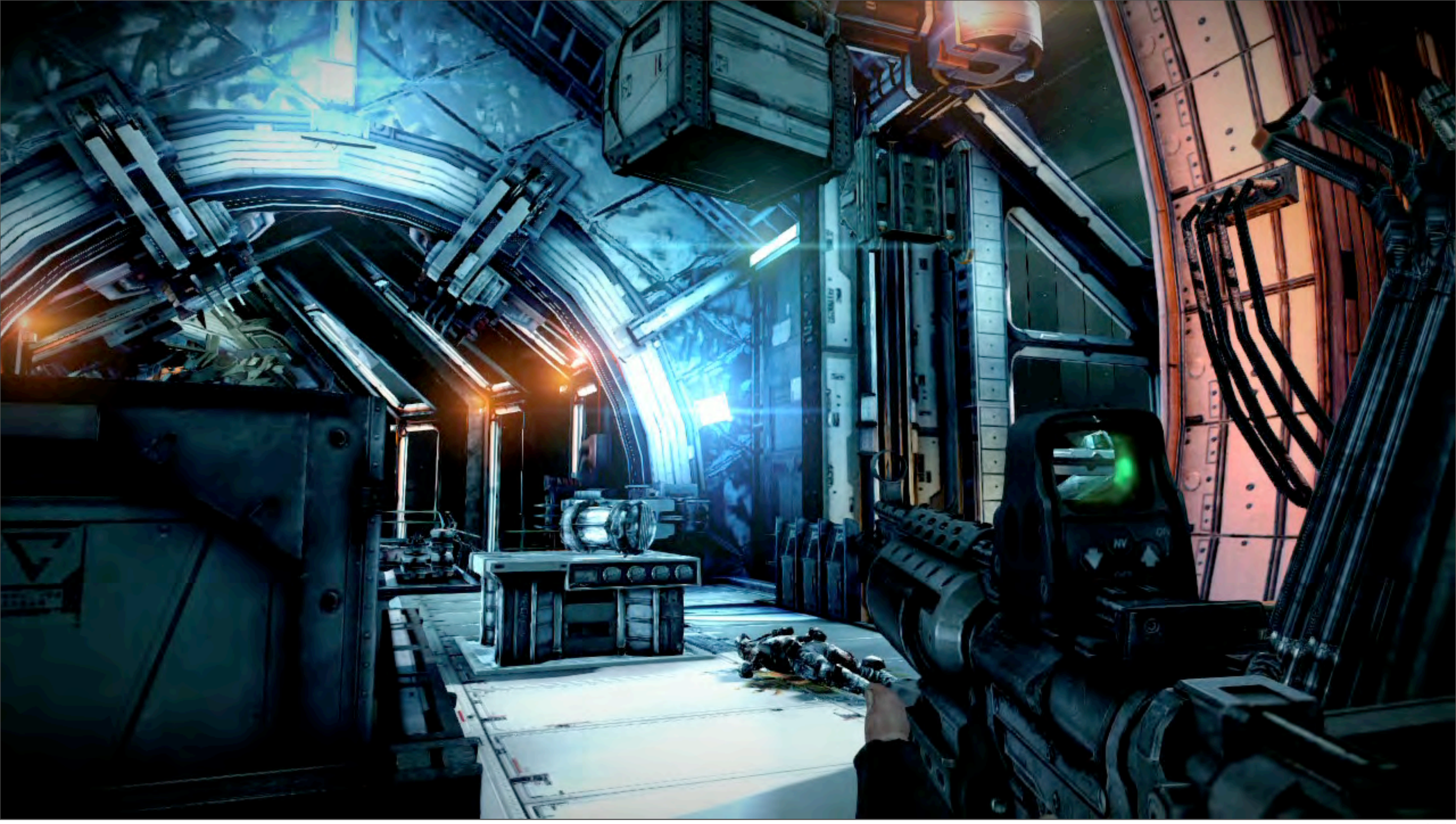
But to clarify this a bit better, i'll start of by showing a couple of examples from our game :

Here we see our main character controlled by the player.

He can move and look around freely and explore the world that we've created for him.

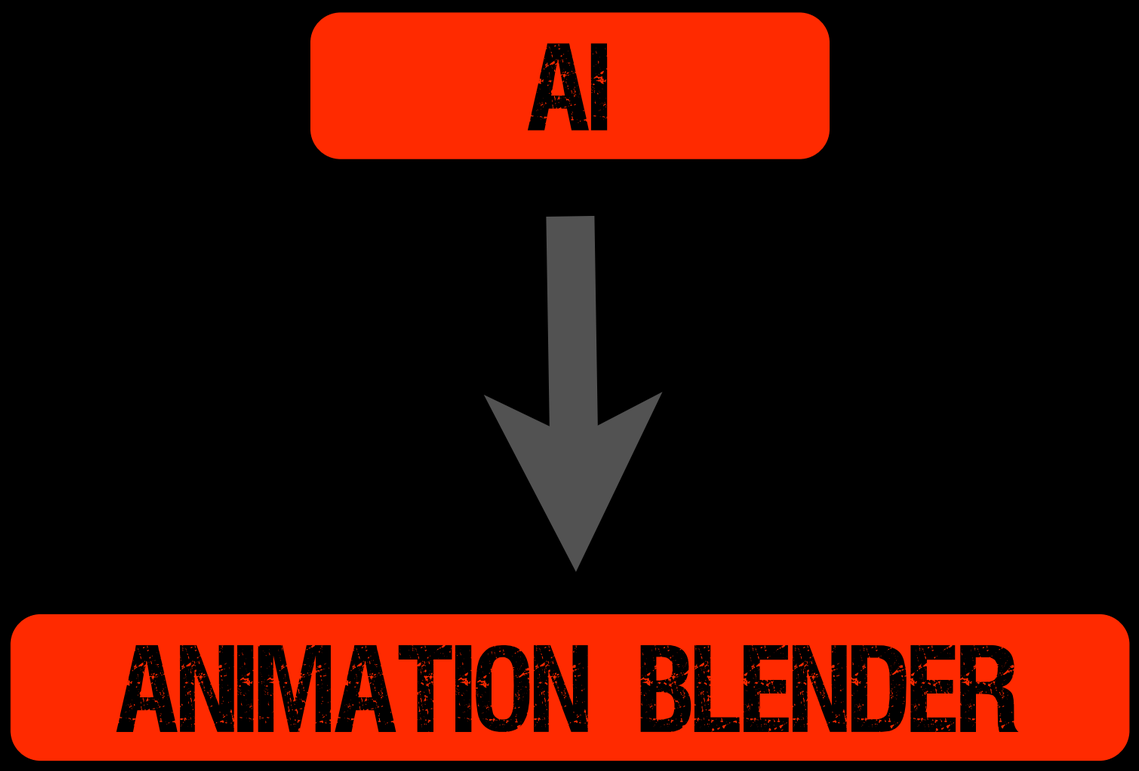
Besides the player there are a couple of other ‘actors’ that we don’t have complete control of.....

The other characters living in our world are controlled by our AI system and they'll react differently every time you play through the game depending on the situation.



we also deal with lots of dynamic objects in the world that the player is free to interact with.

Hopefully these examples give you a better understanding of what we're dealing.



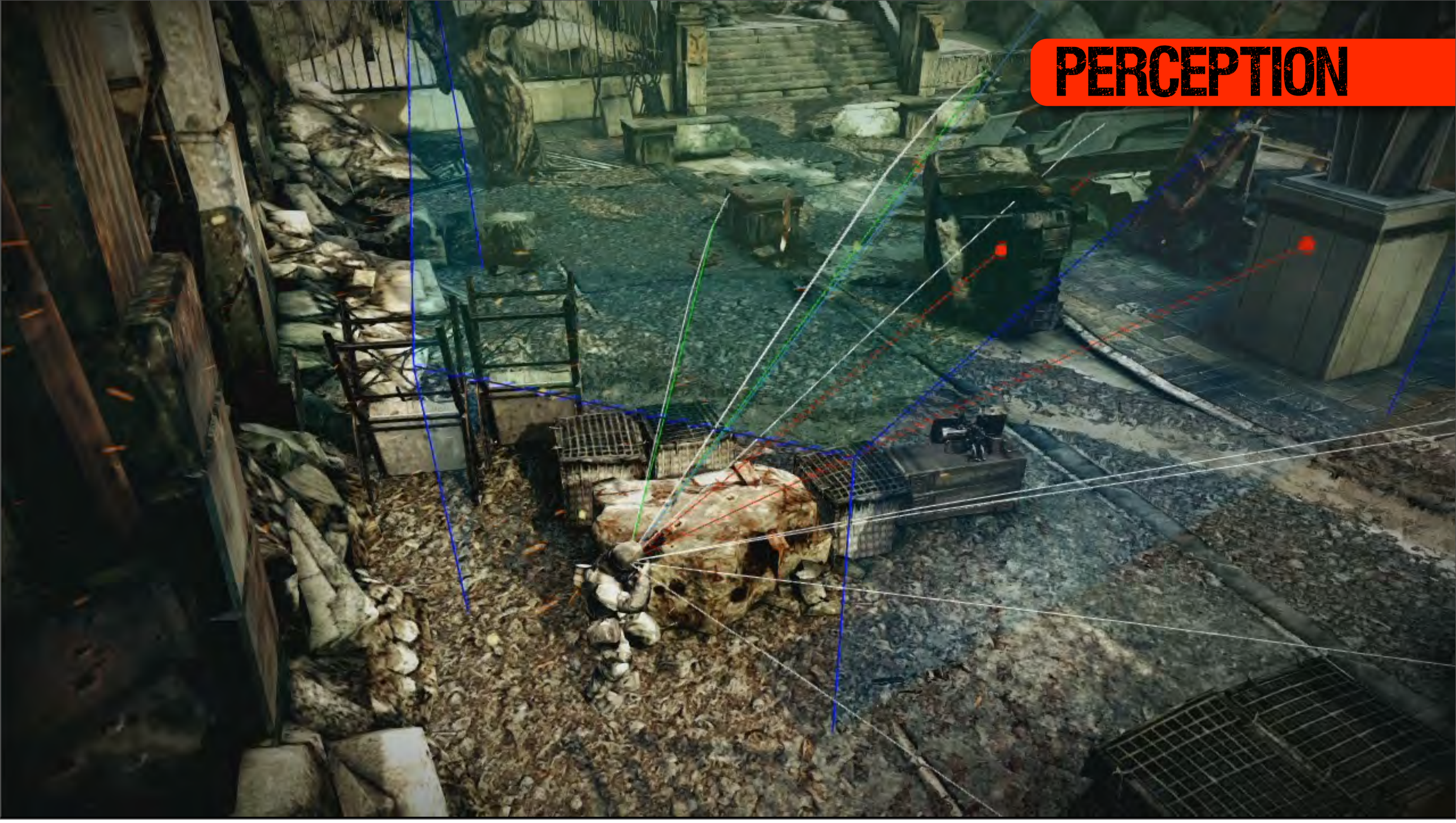
Next I'll briefly explain how our **AI** system works and how this is triggering the animations on the characters... after that I'll go into more detail on our Animation blender tool.

PERCEPTION

In this image you can see typical encounter situation, where we have one of our **enemies** taking cover behind a large concrete block while being attacked by one of the **good guys**.

Our AI Agents obtain knowledge about the world through a sensory system. This sensory system tries to mimic human perception.

PERCEPTION



The **vision part** of the sensory system consists of a cone (represented by the blue lines in this image), in which the agent can directly see things and a **hemisphere** which represents the peripheral vision of the agent. How well an agent can see is affected by distance... velocity... and atmosphere (like for example fog and smoke). In its peripheral vision an agent can only perceive motion.
The **hearing** part of the sensory system perceives sounds such as footsteps, explosions, bullet impacts and bullets whizzing by.

Terrain information

In order to navigate around the world agents need to know where they can walk and what areas are reachable. We use a graph of nodes called waypoints to describe the walkable areas of a level, represented by the green grid in this image.
The agents use these waypoints to decide where to move and how to get there.

The waypoints themselves contain information about cover. Determining whether or not a position provides cover is expensive to do at run-time. In order to be in cover you need your entire body to be obscured. It would require a huge amount of raycasts to check this.

We solve this problem by precalculating a cover map for each waypoint. A cover map is a **cube map** containing depth values. The cover maps are generated in such a way that we can guarantee the entire body is in cover with a single check.

The waypoint graph and especially the cover information are expensive to generate and therefore are created offline. The calculation of the cover maps is distributed over multiple machines.

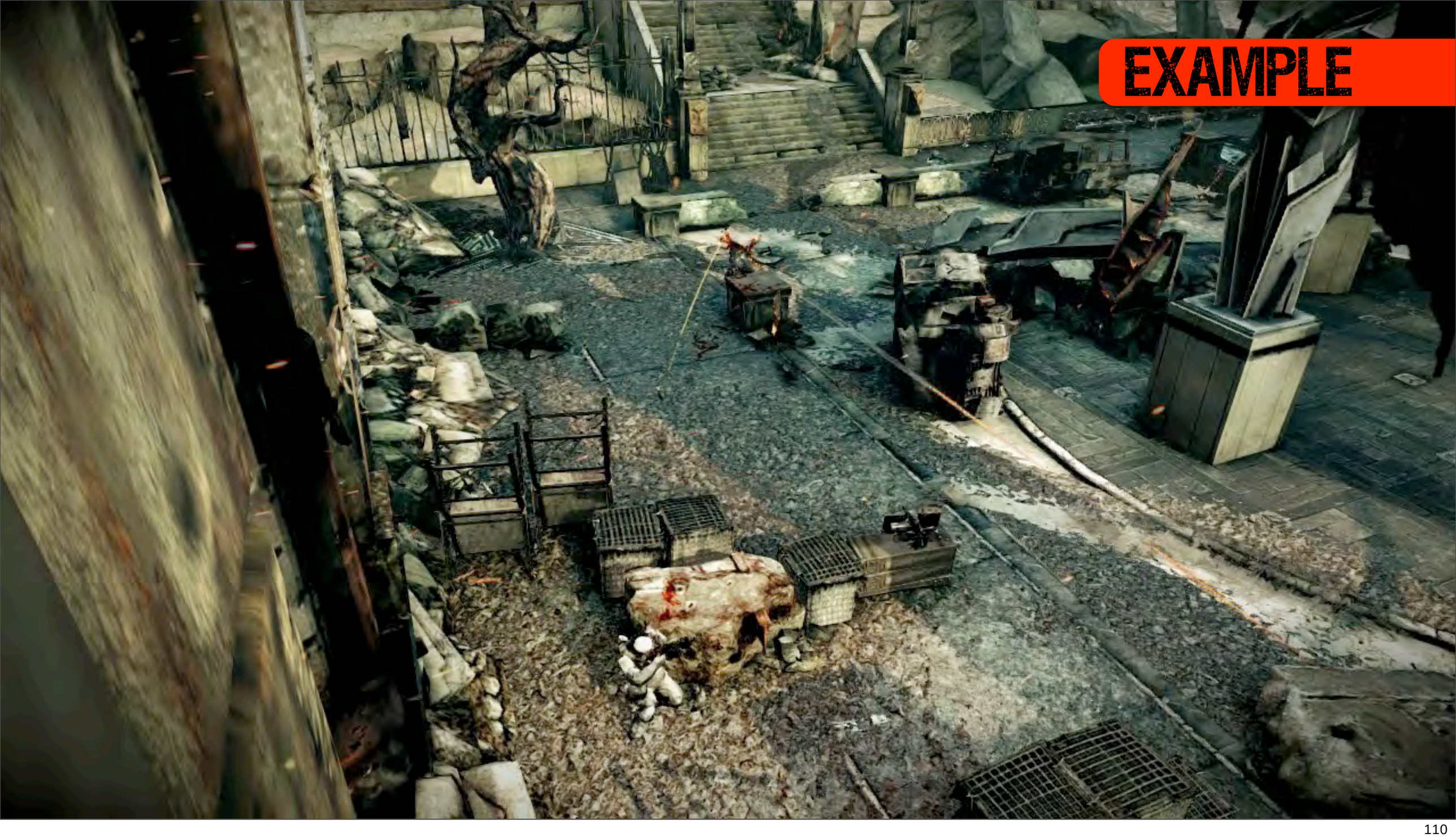
execute plan

After the perceived data has been processed the agent forms a plan of action .

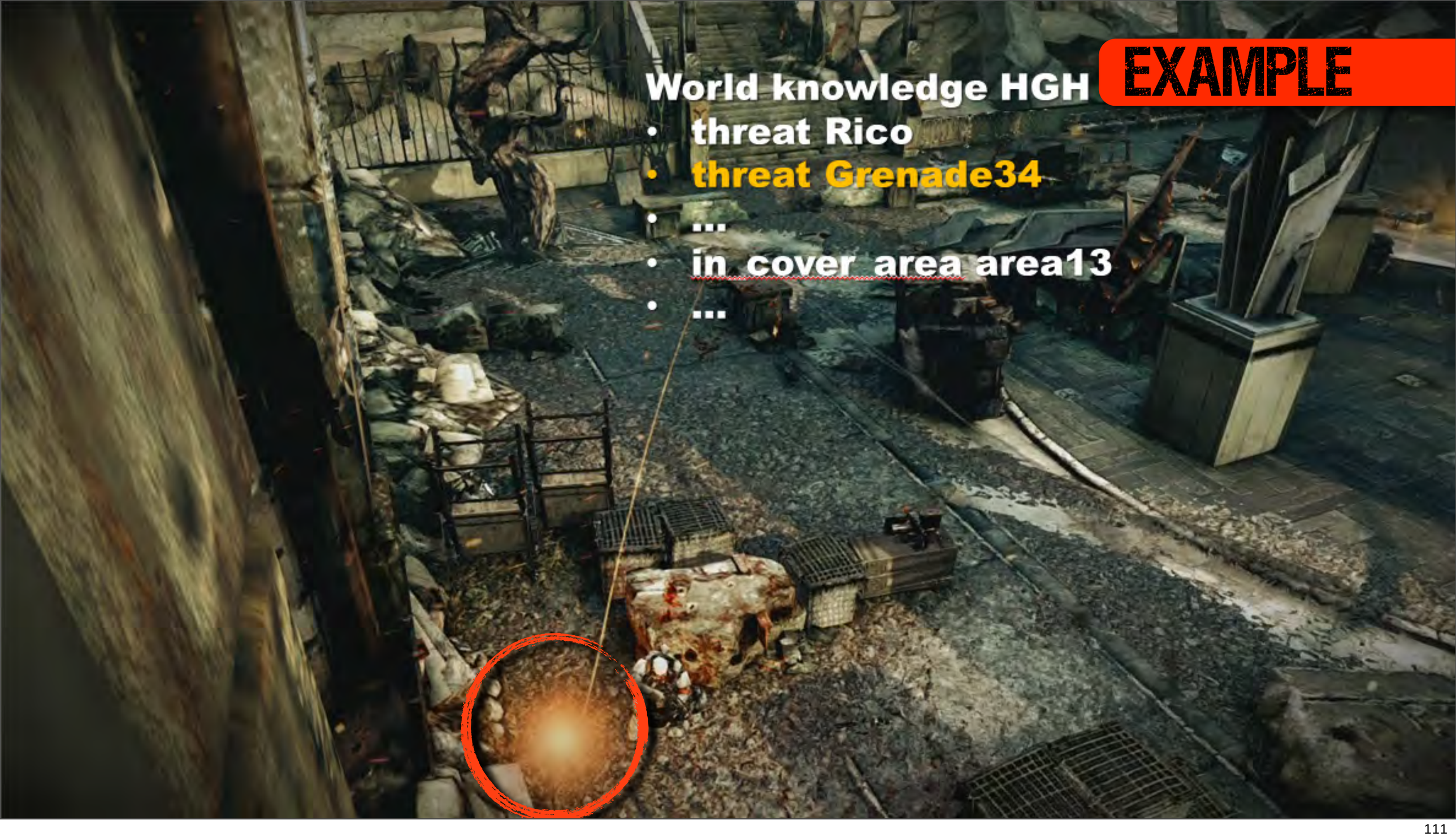
In the process of creating a plan we search for good positions in the **vicinity** of the agent. We **score** positions based on cover and attack opportunities....
distance to walls and enemies ....and many more.

The cover maps provided by the waypoints are essential to this process. In this image you can see the lowest scoring points in red and highest in green.

The executed plan is a sequence of actions the agent will perform, let’s take a look at a example:



Here you can see a grenade getting thrown near our agent.



our agent is aware of the new threat.

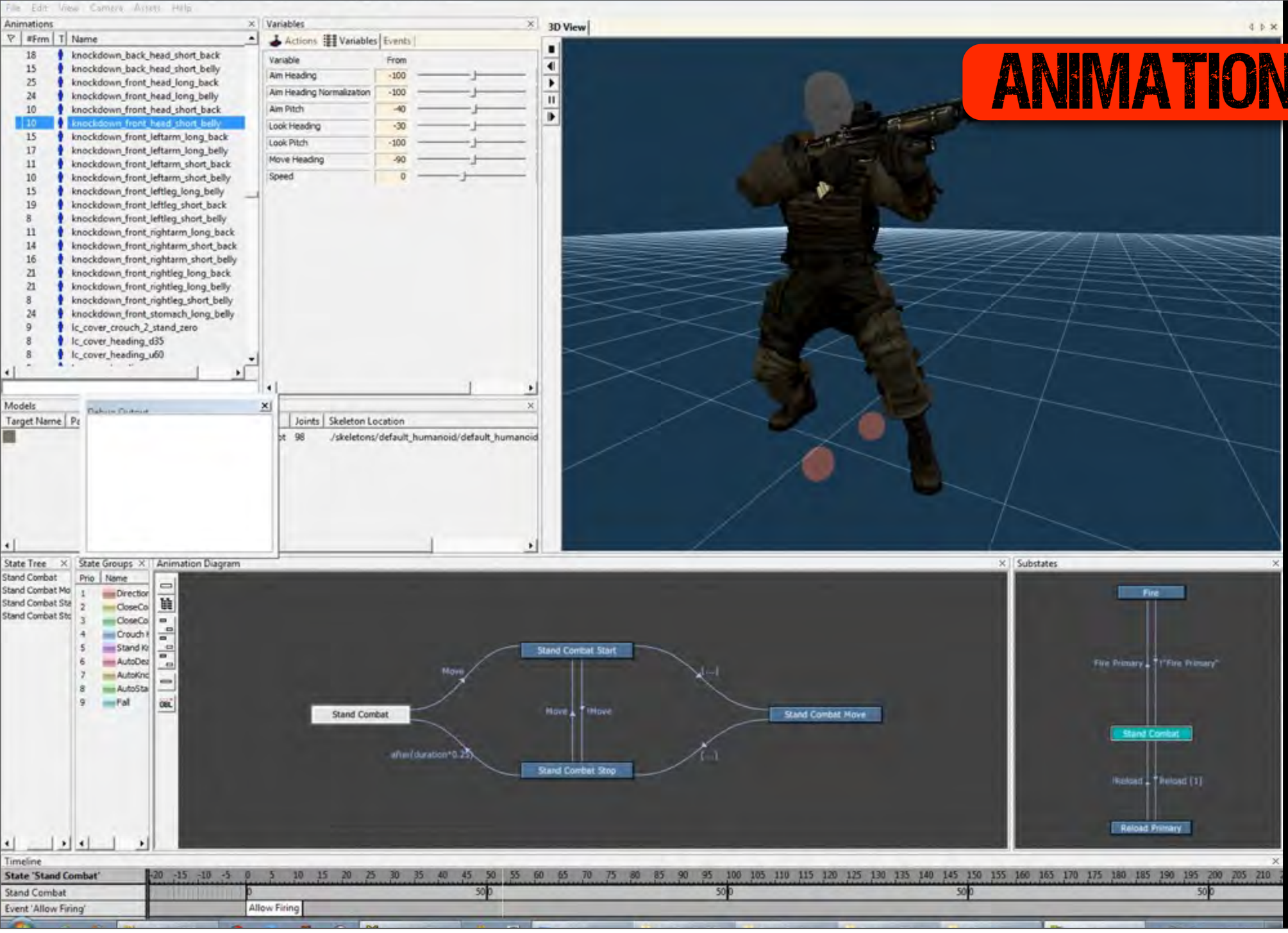
Based on this plus the terrain knowledge a decision is made to go out of cover and flee to a safer spot, represented by the green waypoint.

here you can see this in progress..

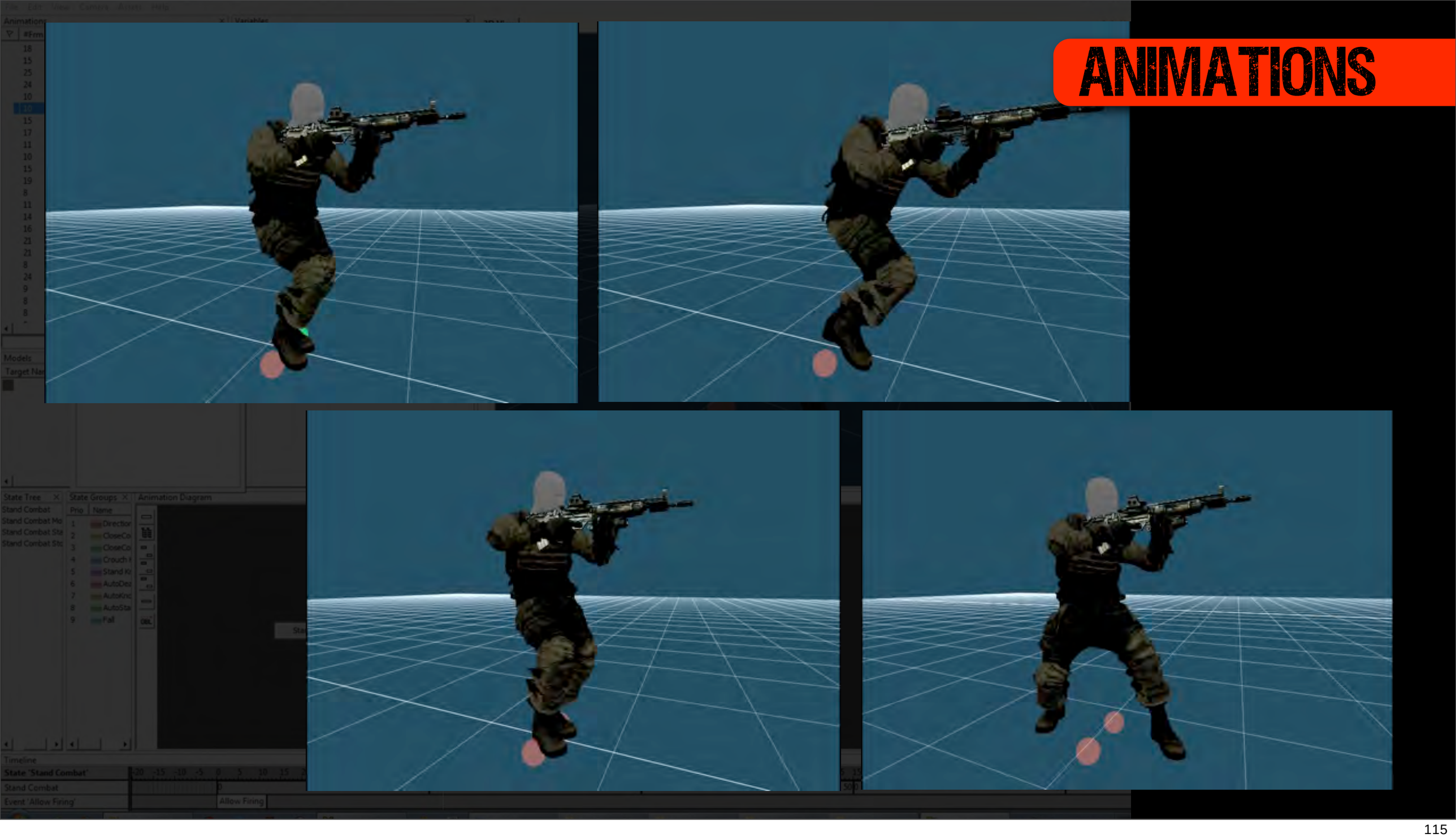
The plan executed contains the following series of actions :

- run from cover
- run to safe position and ....attack threat.

These actions are then sent and executed by our animation system.



Here you can see a screenshot from our Animation Blender tool,
We us this to define the motor skills of our game characters.
We start of by loading in our animation data..

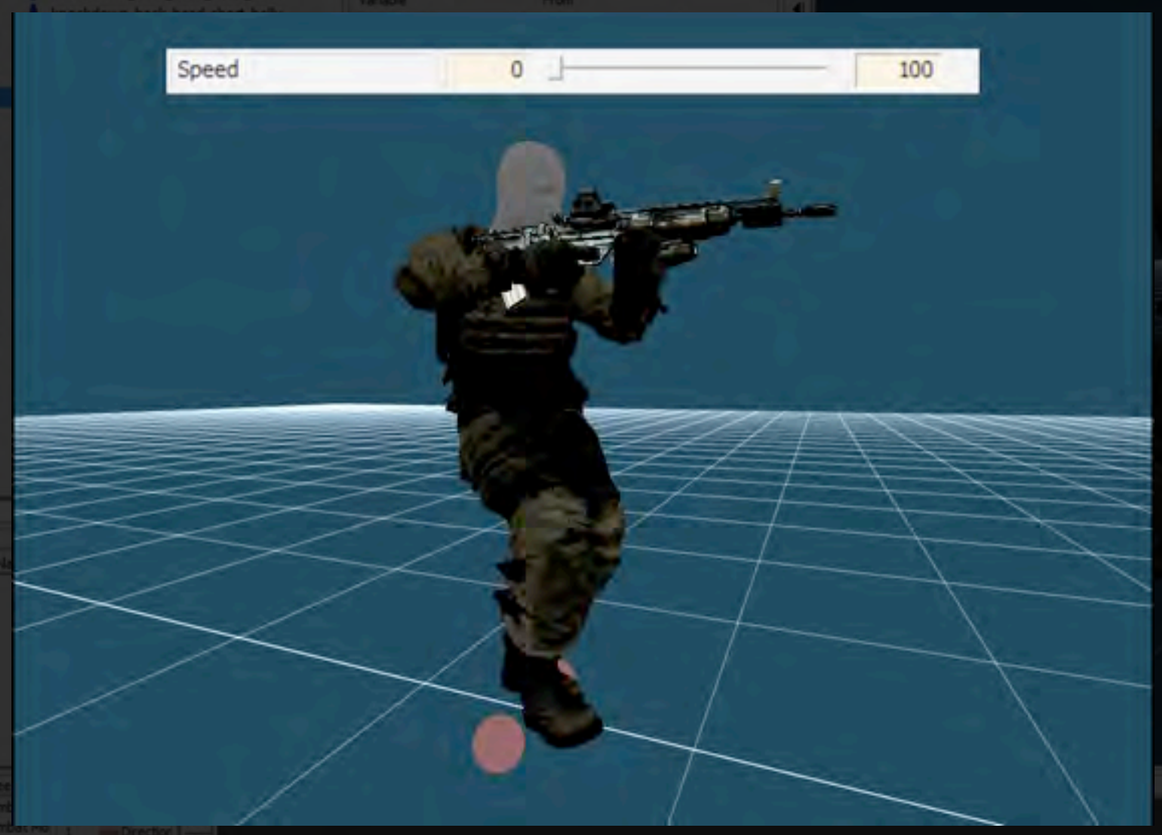


here can see a couple of example animations.

In total we have about 3500 1st person ....
and 5000 3rd person animations

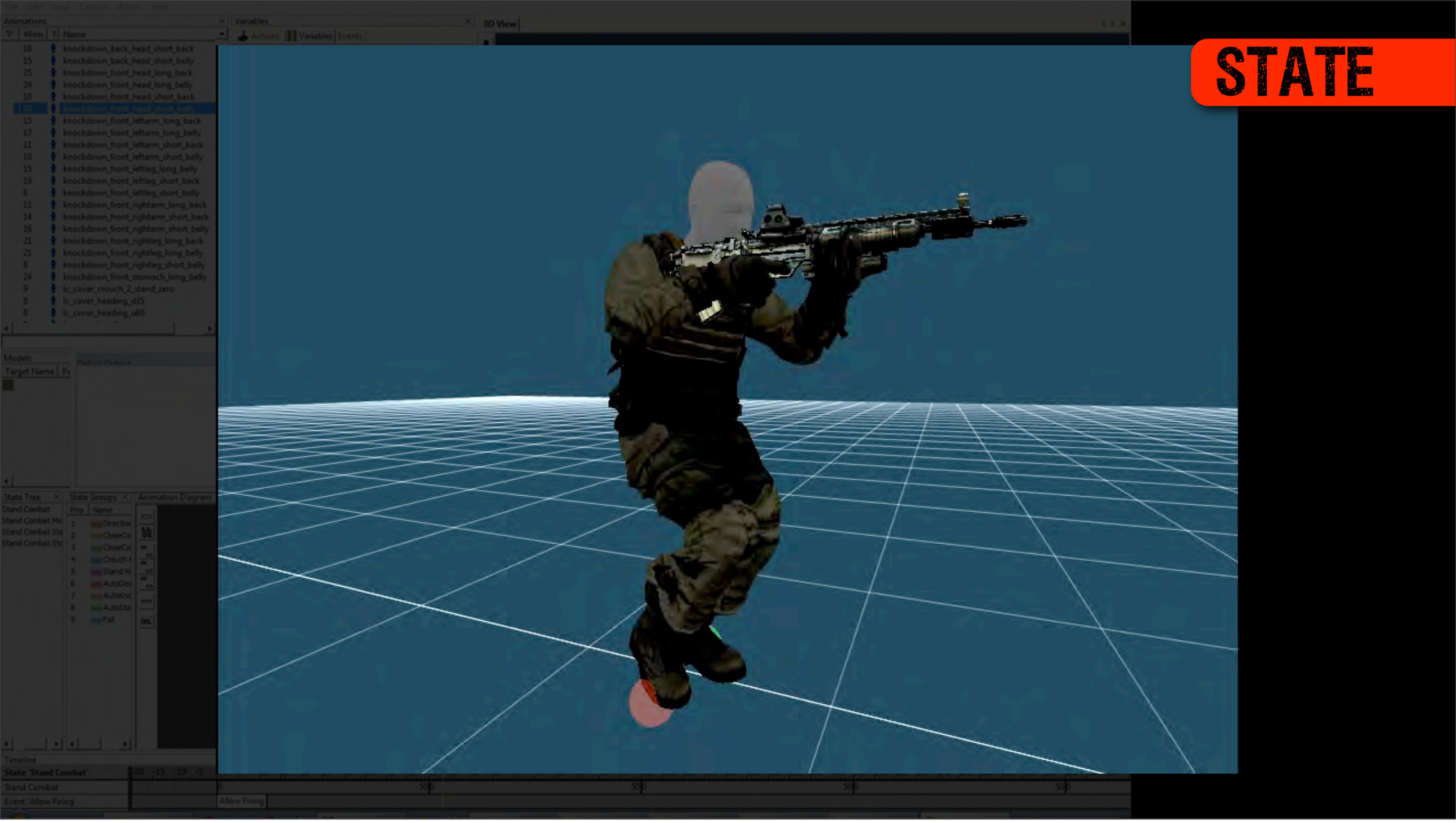
We define **Variables**, with these we're able to drive the blend factor between our animations as you can see in **the following** examples.

VARIATIONS



The speed value of the walk cycle seen on the left is a blend between 3 animations.

The weapon aim heading and pitch uses 5 animations for blending.

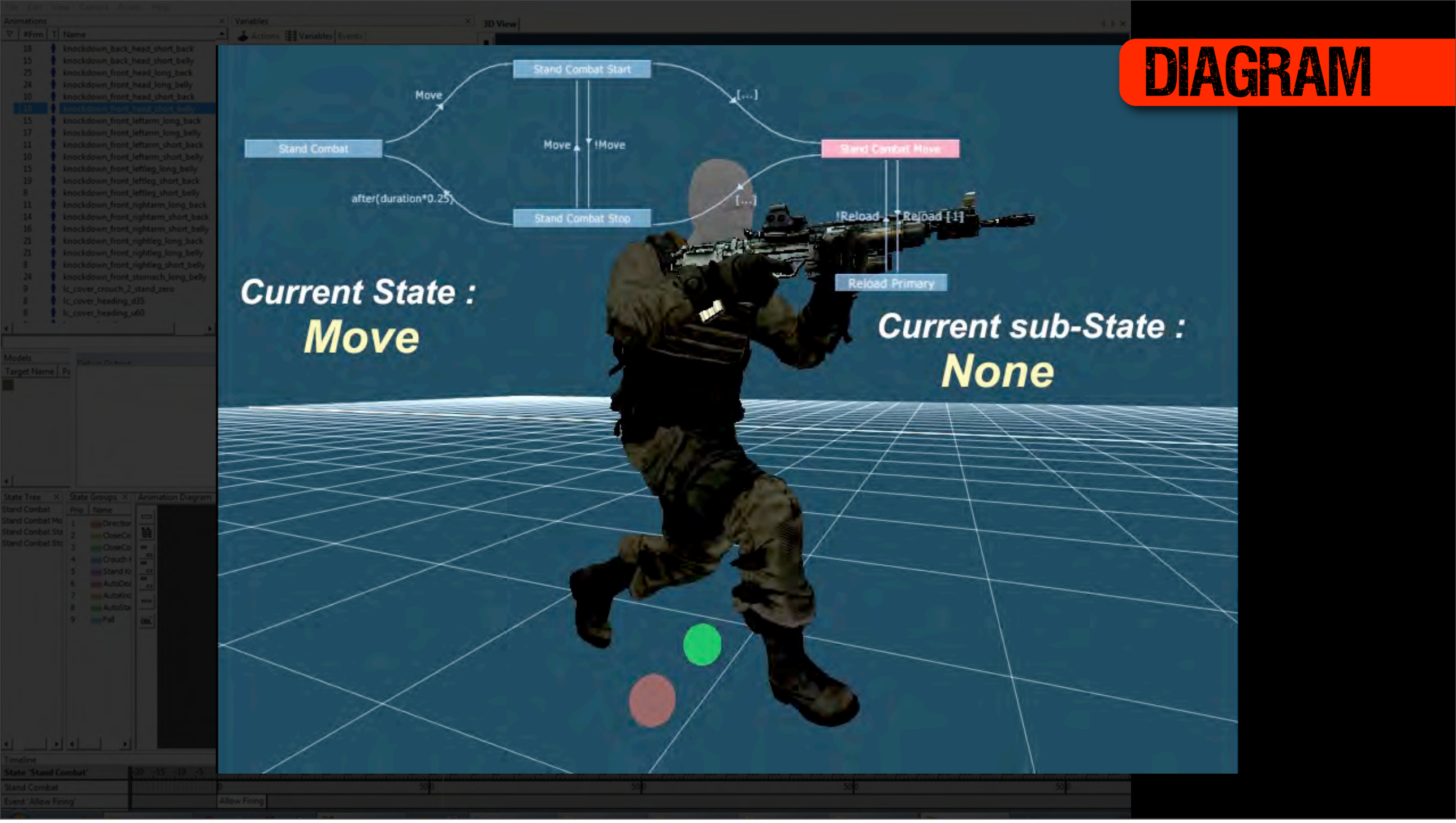


here you see them used in **combination**. A specific group of animations define a state.

All of these States are put into a diagram that pretty much resembles the motor-skills part of the brain.

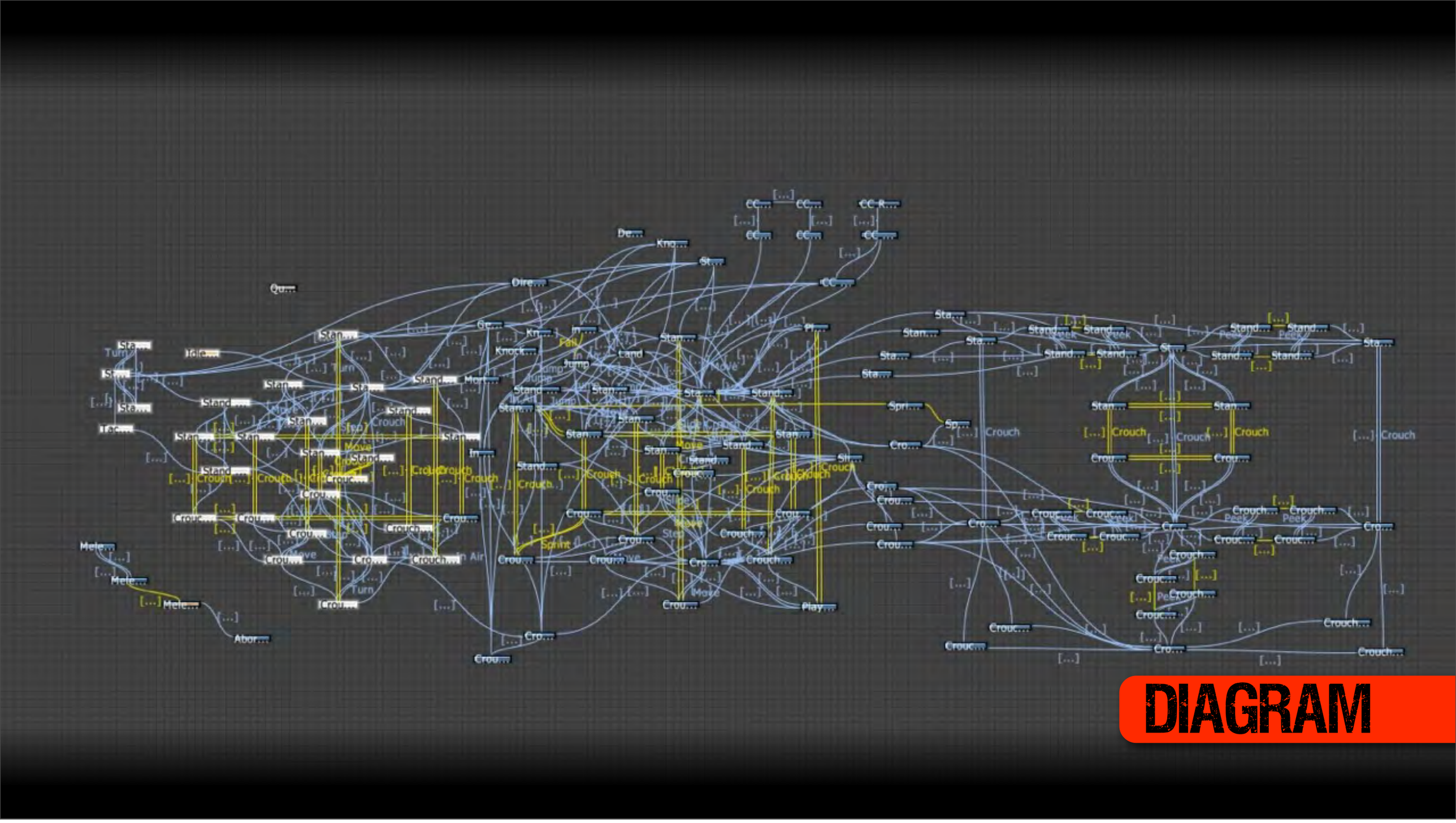
The highlighted states in the diagram get executed and the animations on our character updated.

States keep changing depending on the executed action.

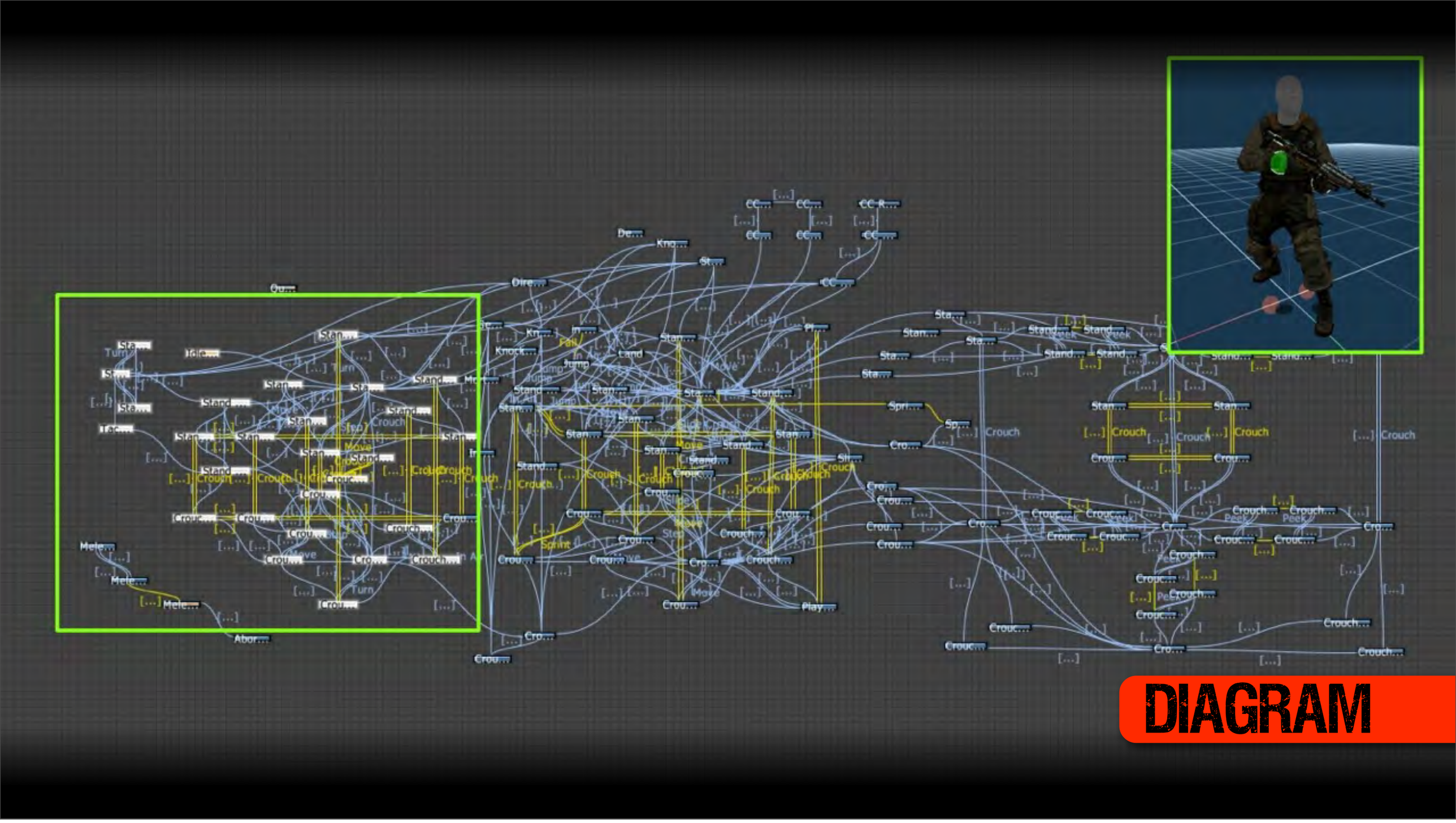


If you go even deeper in the diagram you'll find sub-States, for example : while the character is in his moving state he can execute a **sub-state** to reload his weapon.

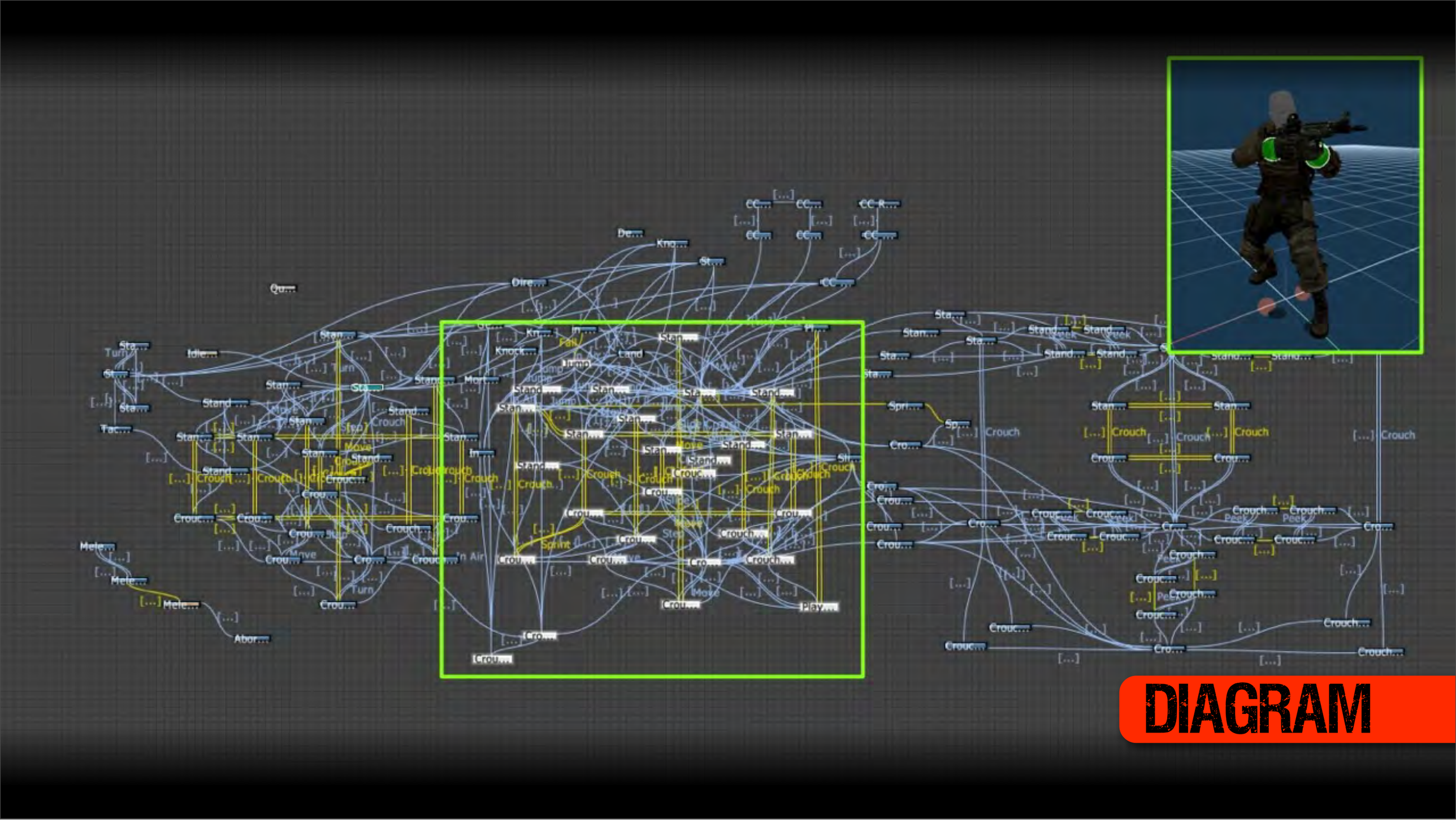
This example only shows you an extremely simplified diagram. The actual character diagrams are a lot more complex..



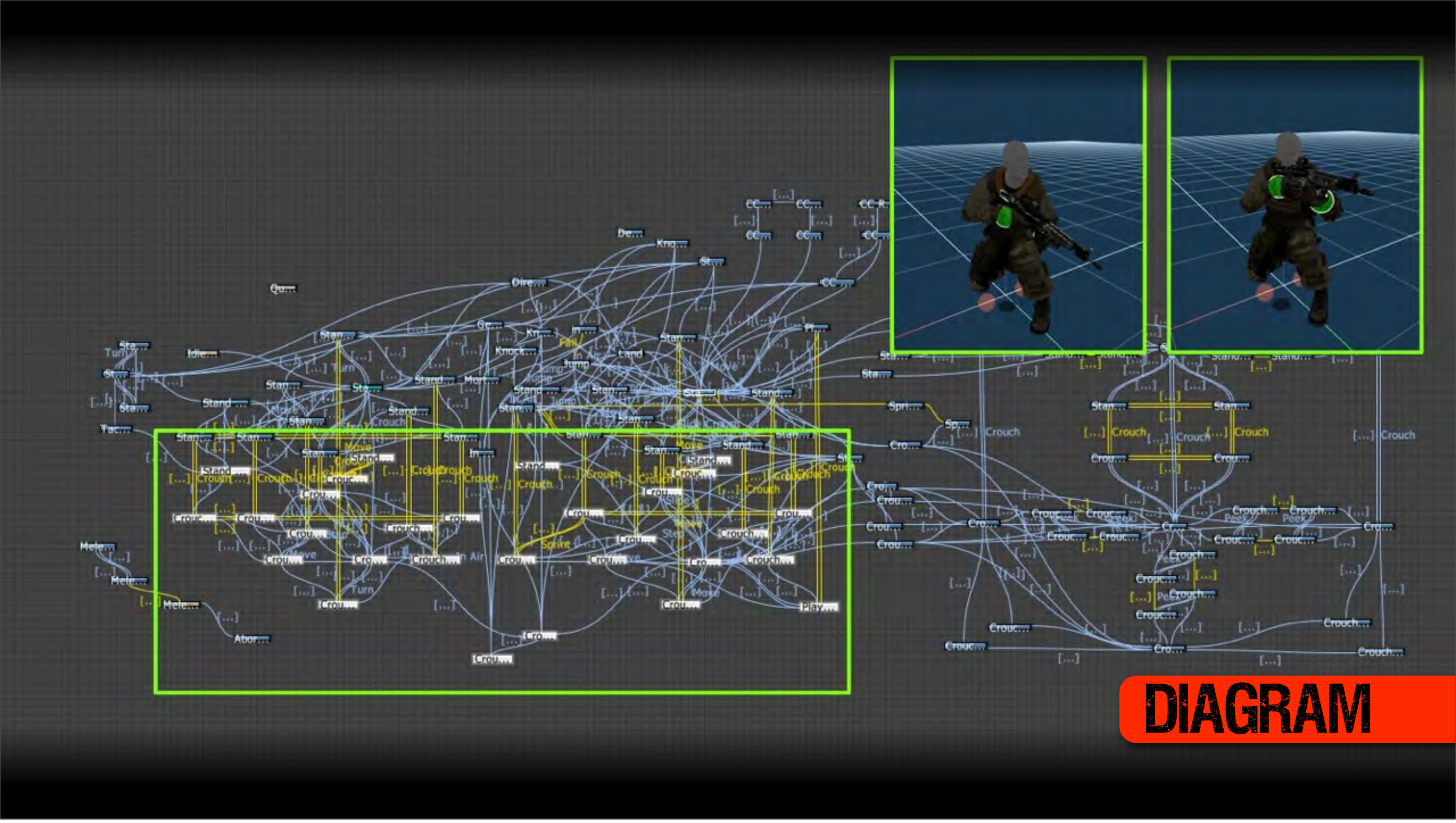
....I know what you're thinking... but YES..... There is definitely logic in this....



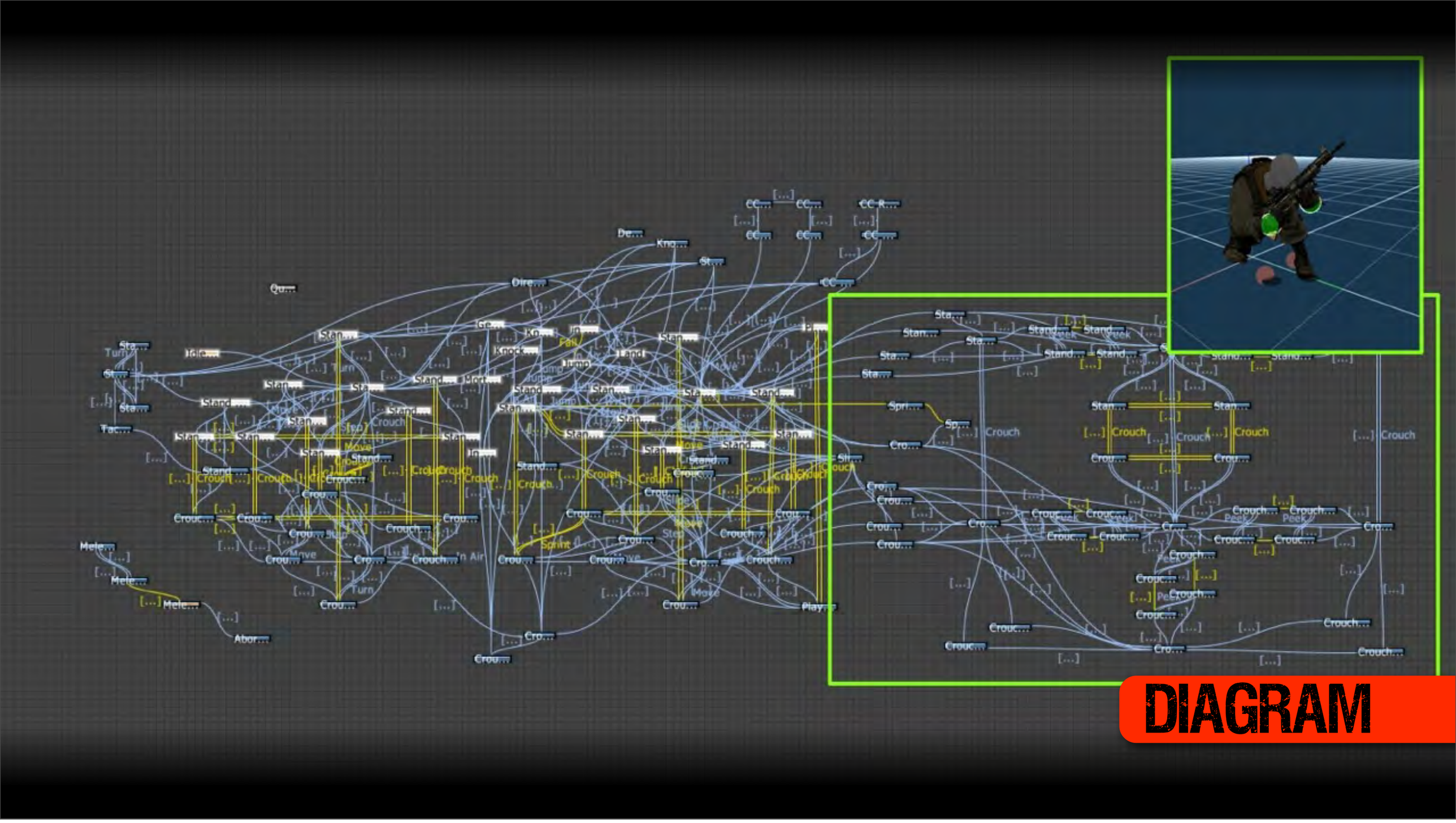
this area represents the tactical side and contains all animations for non-combat situations. (gun lowered)
You can see an image of what the character looks like in idle mode during these states.



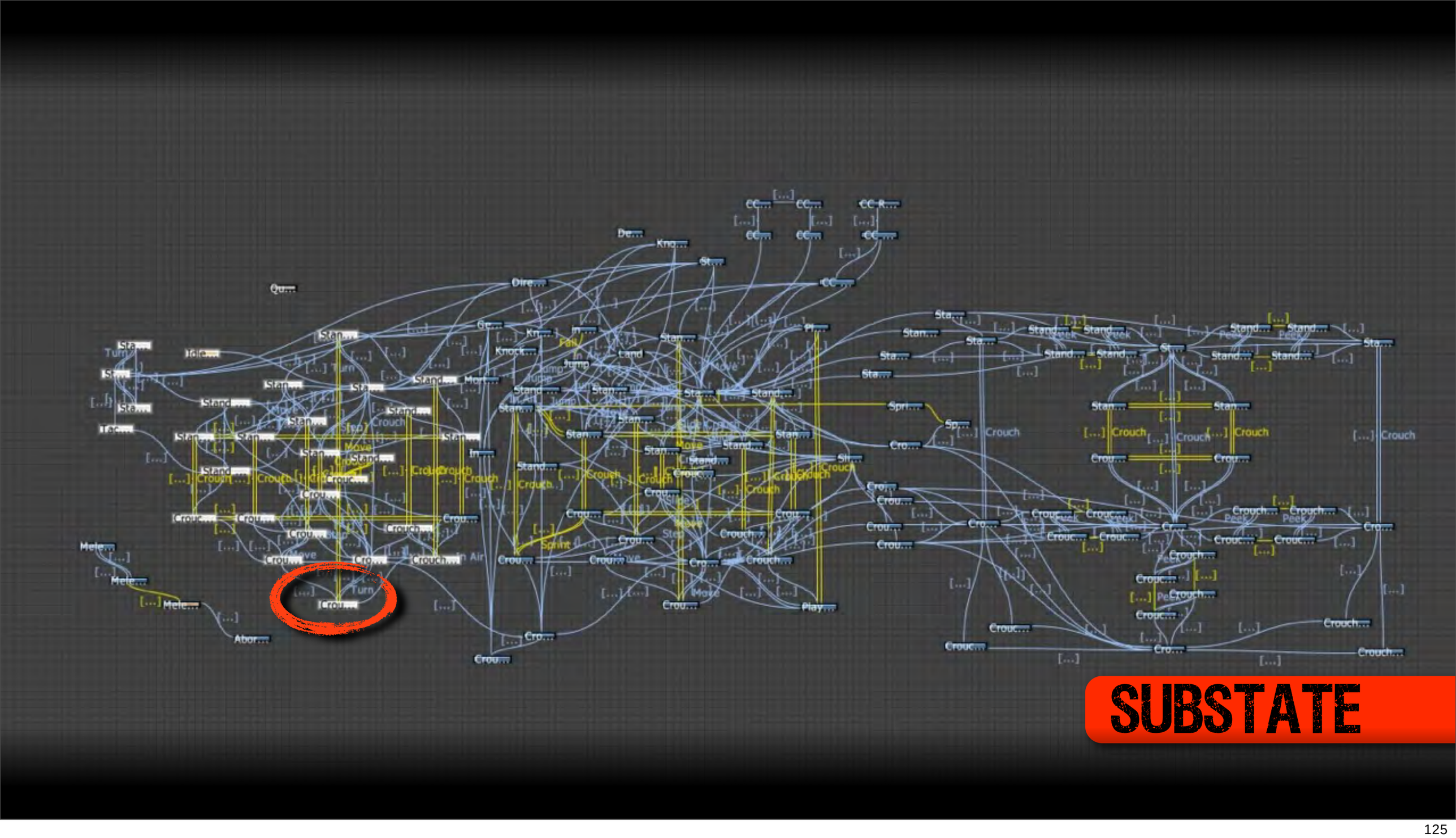
this area contains all combat animations.



this contains all crouched animations for both tactical and combat mode



and the furthest right area represents the lean and peek animations for use with cover objects



This shows one of the many **sub-states** the diagram contains. Part of this shows the firing and reloading of the weapon. The reloading of the weapon has some simple logic that makes sure to only give the player ammo ones it's finished playing the animation for x amount of percent.

Here you can see the diagram for 3rd person jumping states and playing the appropriate animation...

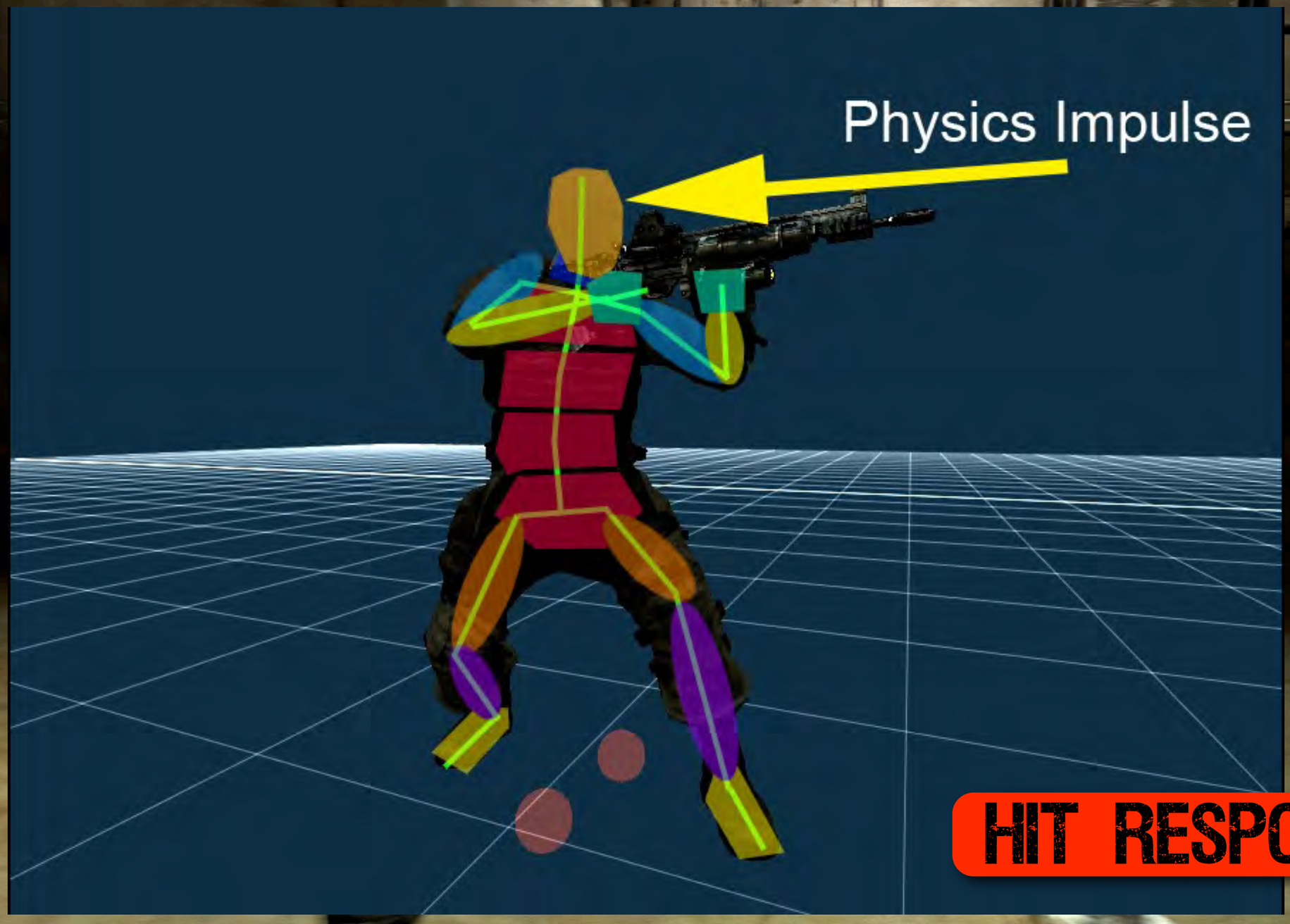
Next we'll take a look at our Hit-Response system.

HIT RESPONSES

When we started working on KZ2 we wanted every bullet to have a dramatic impact. The next video shows some very early KZ2 footage demonstrating the Hit-response system.

Let’s take a closer look at how this was achieved....

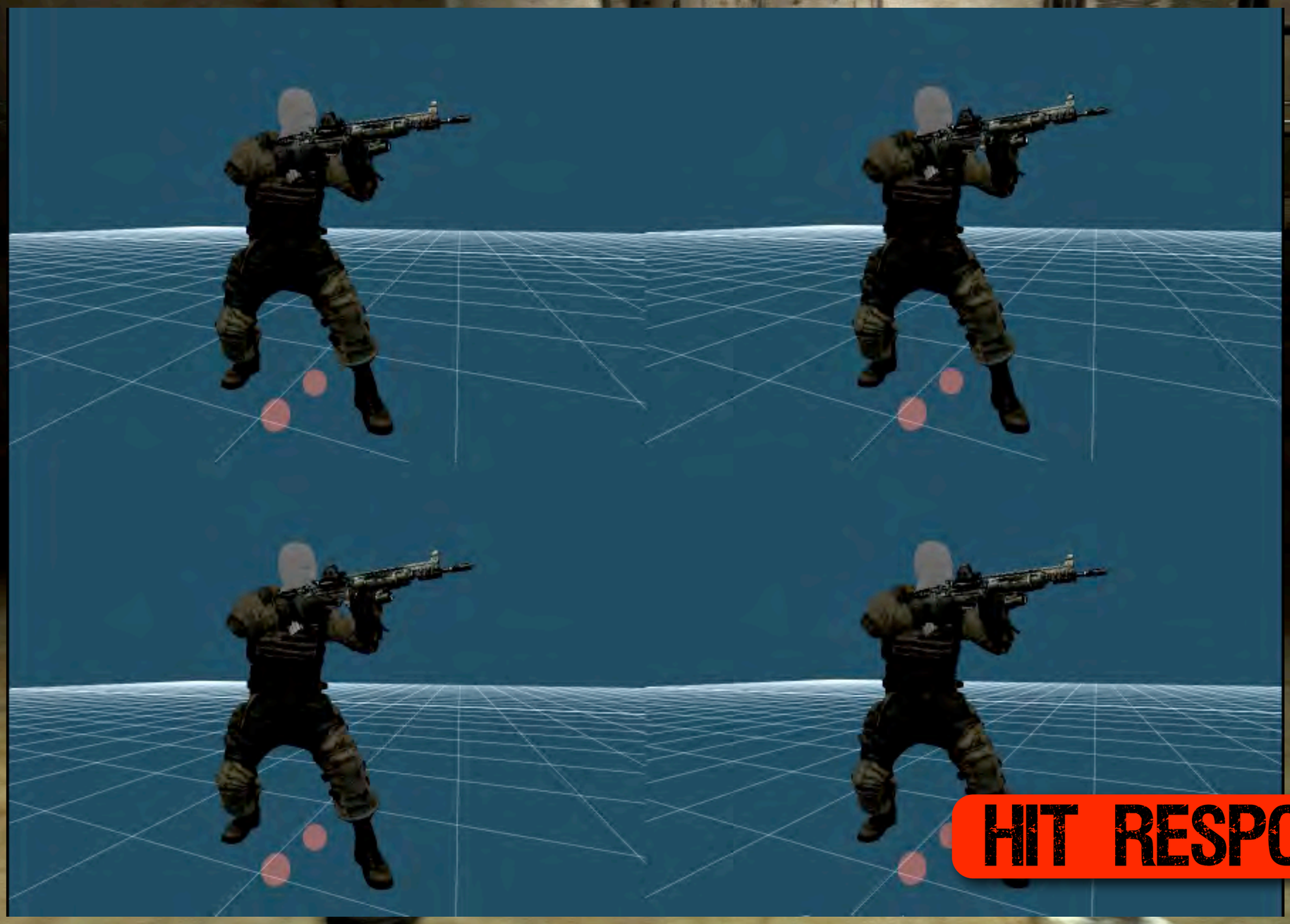
All of our characters have a **ragdoll** physics setup.....
When a bullet hits the character,



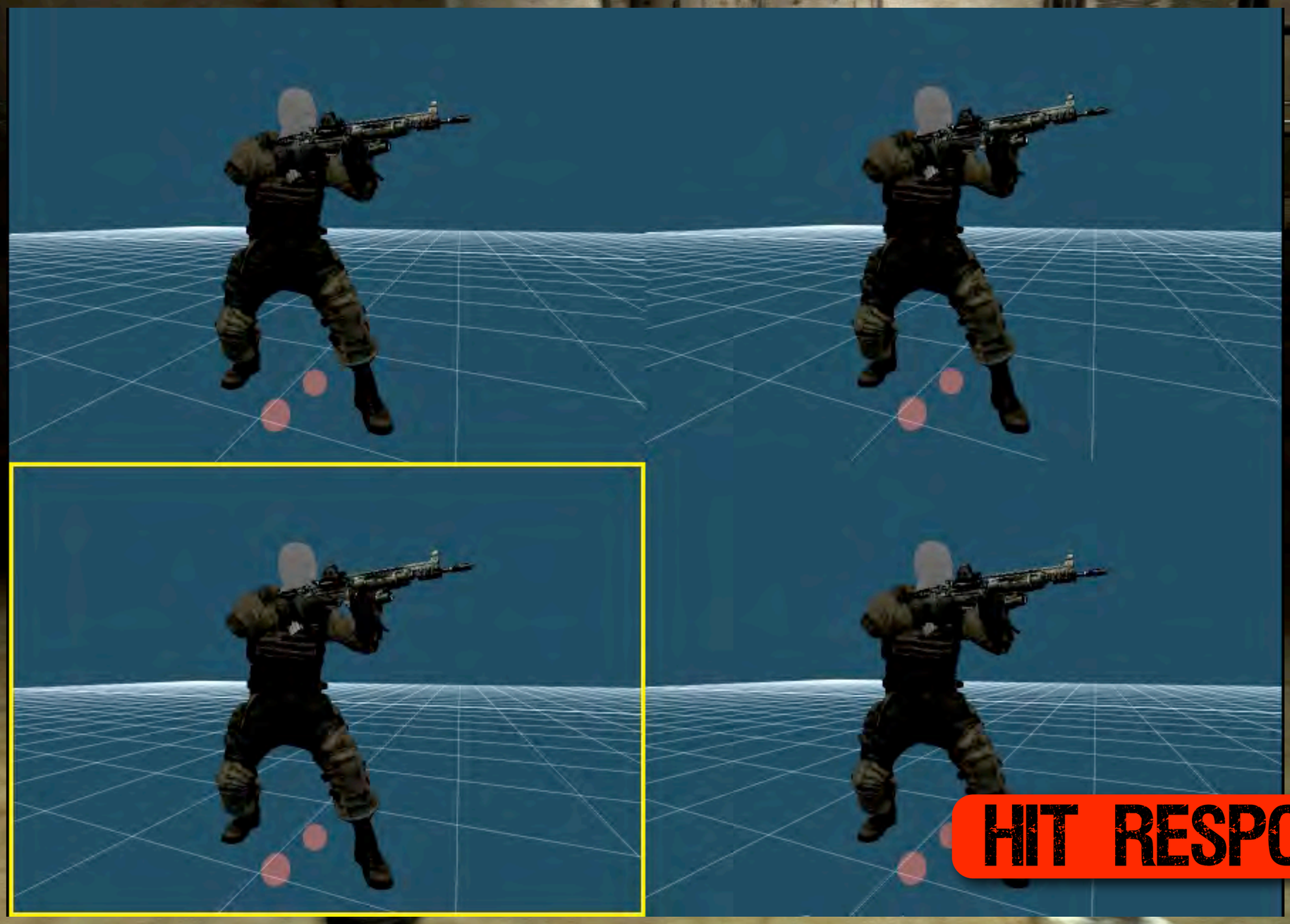
a physics impulse is given to the ragdoll according to the point of impact, velocity, angle.

Once the impulse has been given to the ragdoll it’s **pose** will change.

This pose is then sampled and compared to a set of predefined impact animations.



Here you can see a couple of those animations. After it’s done comparing the best matching animation is **selected……..**



and used for blending. The system is also able to blend half-way animations.

When a character dies the system will play 1 of the best matching **death animations**.....

When you see the character freeze in this video that’s when it will start to try and mimic a death pose

This shows you a couple of examples

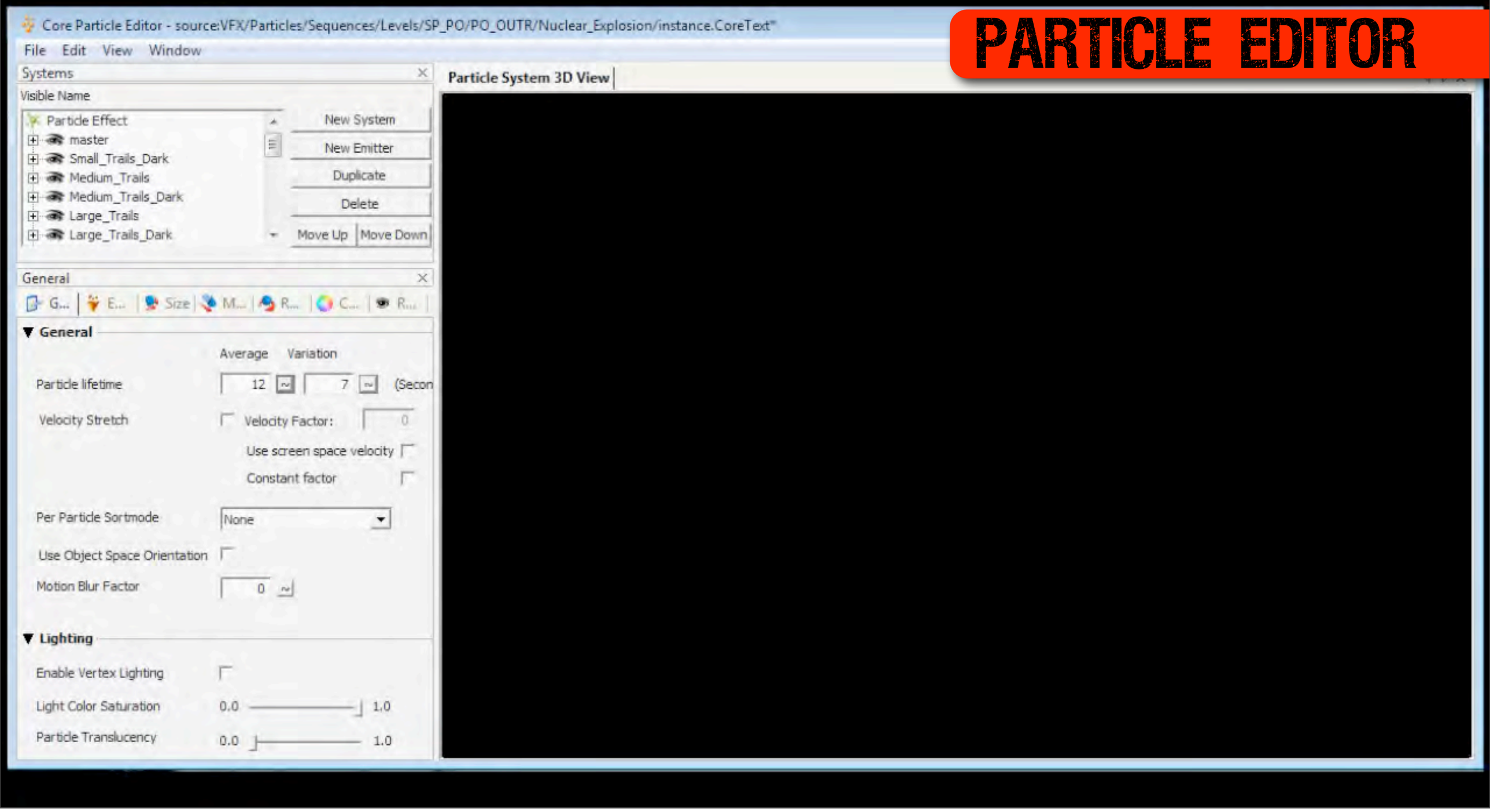
The end-result in game will look something like this

Next I’d like to take a closer look at our particles system.

Let’s start off by viewing part of a FX heavy cutscene....

after that I’ll go over some of our particle features.

In this next video you can see the effect of this explosion loaded in our particle editor.



One of the nicest things about working on a video game as an artist, is the instant feedback you get while tweaking values.... this really speeds up the process significantly.
At the moment video game particle effects are still mainly sprite based but this seems likely to change in the future.
Our particle editor pretty much has all of the basic features you'd expect : setting values for particle lifetime, size, emission rate, movement, color values, etc.

Let's take a closer look at the elements of this explosion..

BREAKDOWN

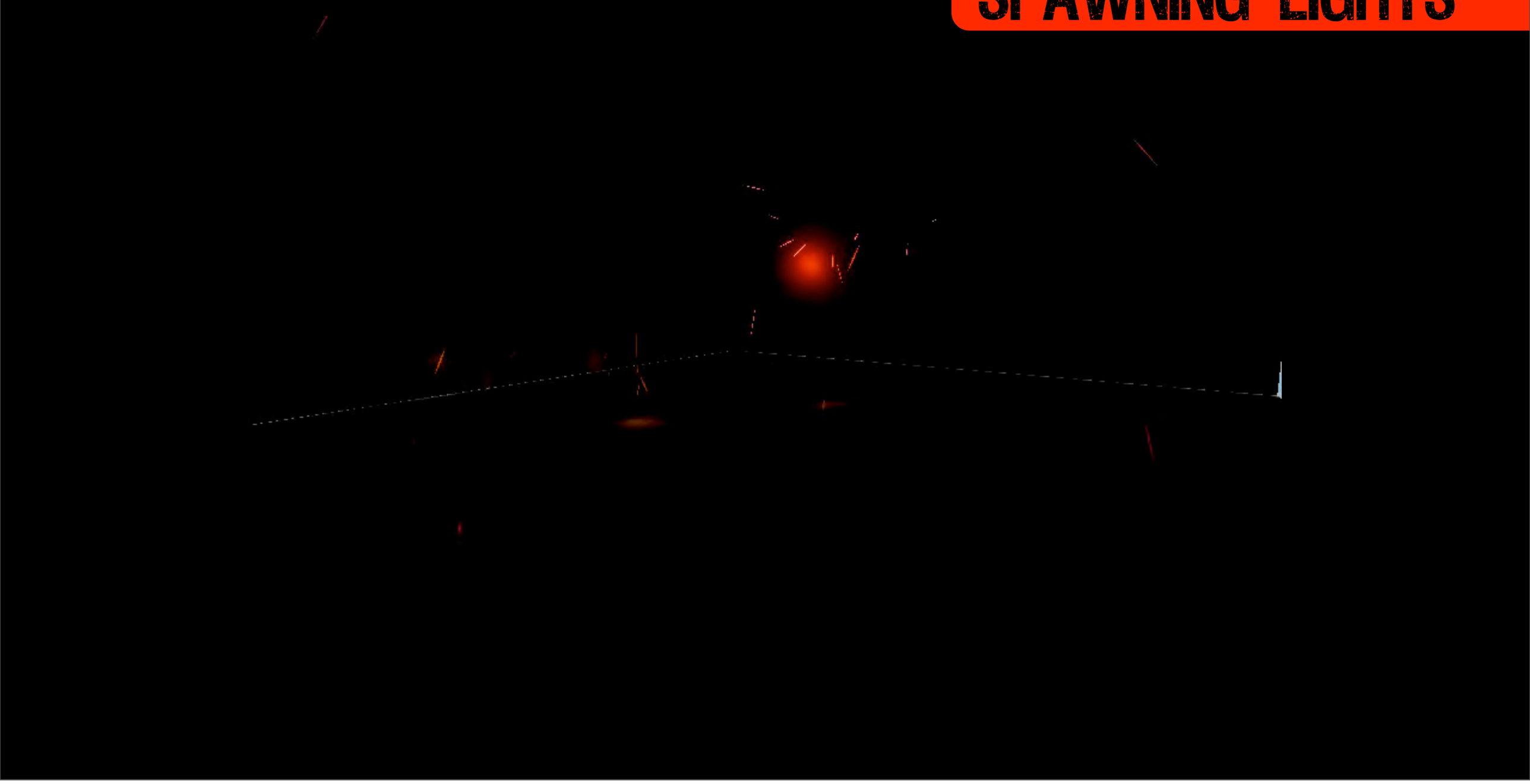
For this particular effect we've used about 30 particle systems and spawn about 2500 particles. This particular effect was made for a cutscene and the nice part about working on cutscenes like these is that we can really push our particle systems much further. We don't have as much performance restrictions as we do with FX during gameplay. JanBart will explain more about this process in a few minutes.

Let me show you a couple of nice features we have...

POLYGON TRAILS

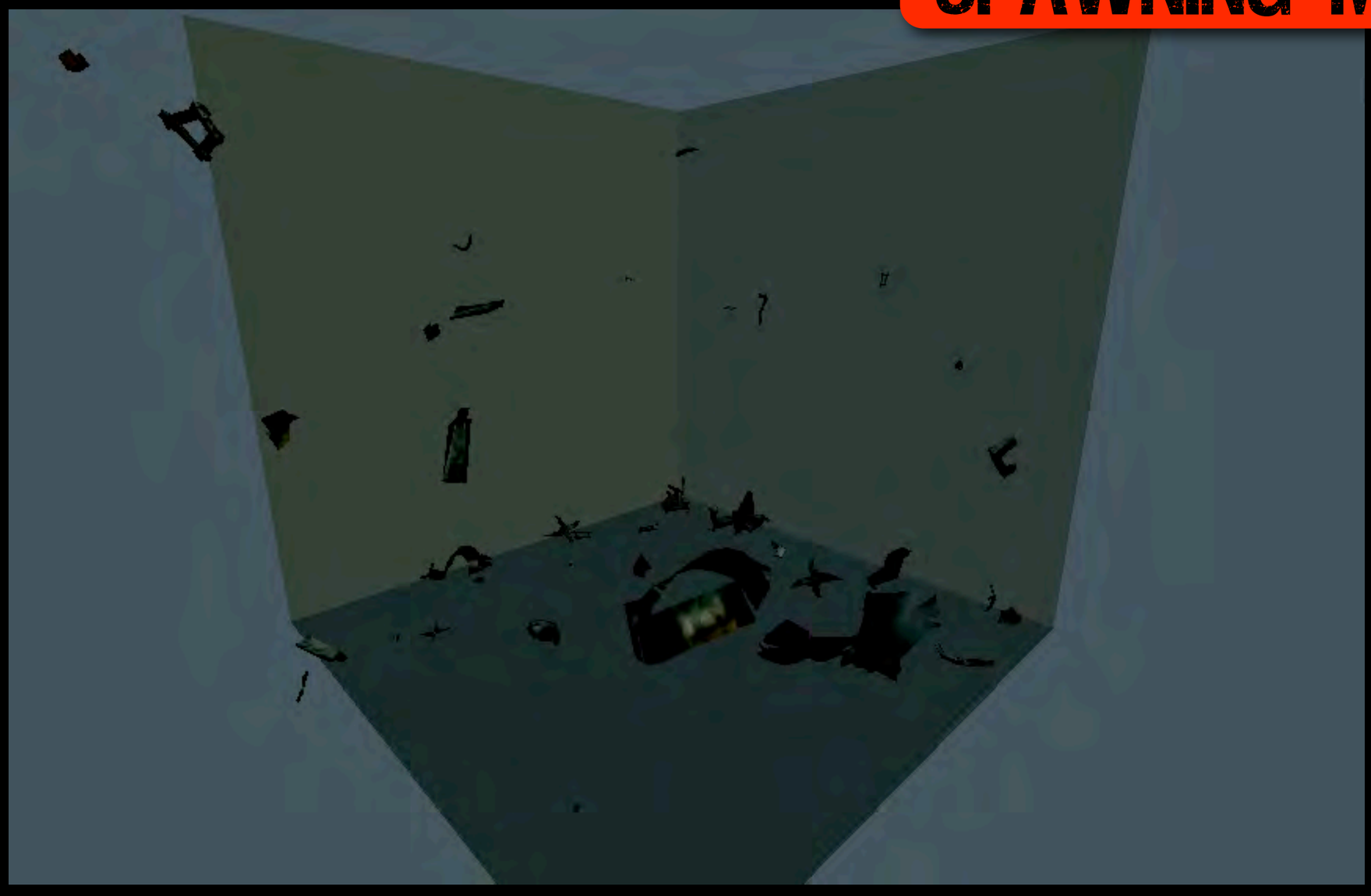
One of the new features on KZ3 was the option of polygon trails, these really helped in getting much more continues missile and dust trails. It generates 2 vertices every x amount of steps and connects these to the previously generated points. You can then set the generated polygon to face the camera.

SPAWNING LIGHTS



Since we’re using deferred rendering we can spawn lots of small lights with our particles without any significant performance drop.

SPAWNING MESHES

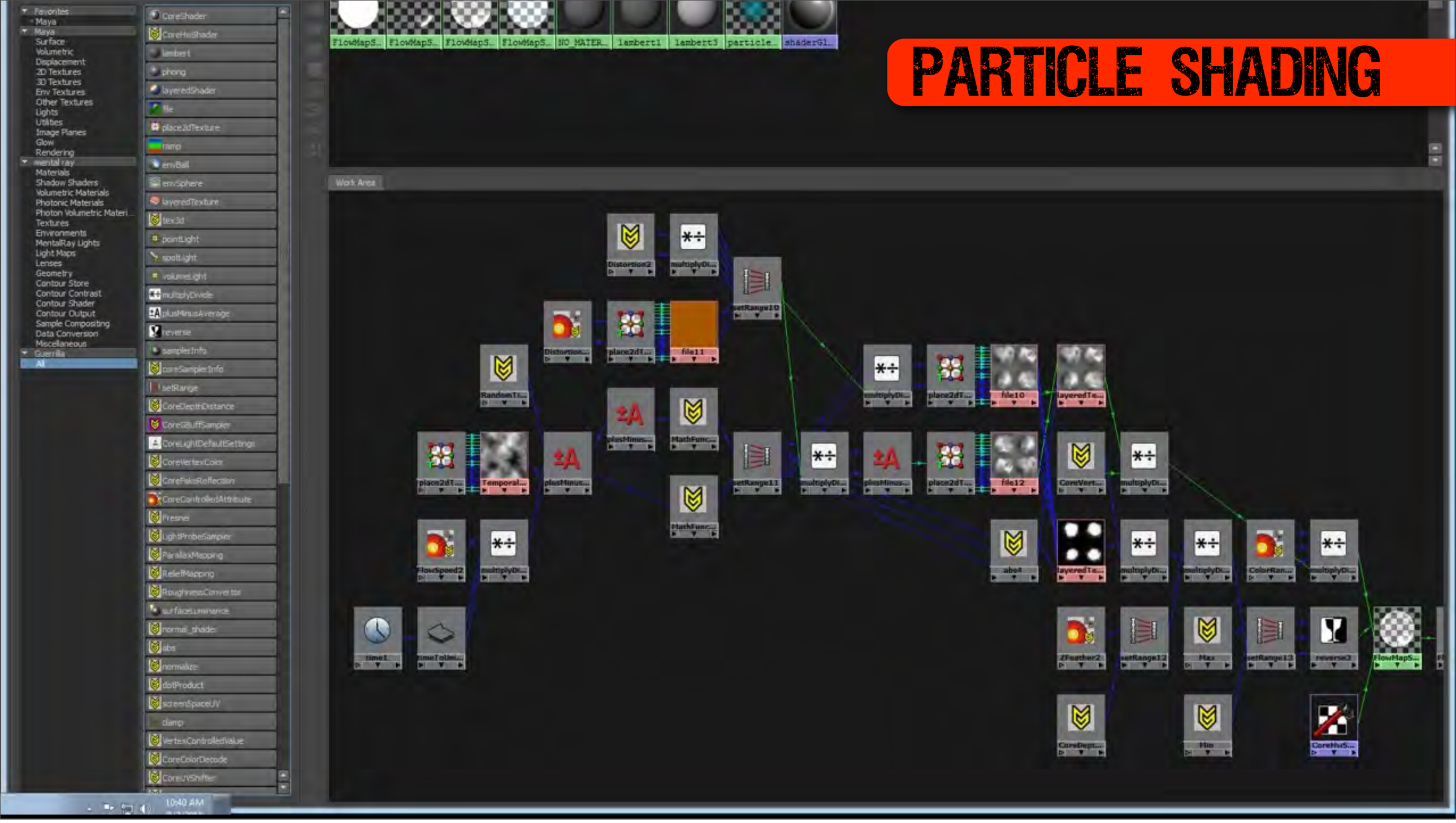


Mesh spawning is also a possibility....

Adjusting emitter settings is actually only half of the work in creating effects.

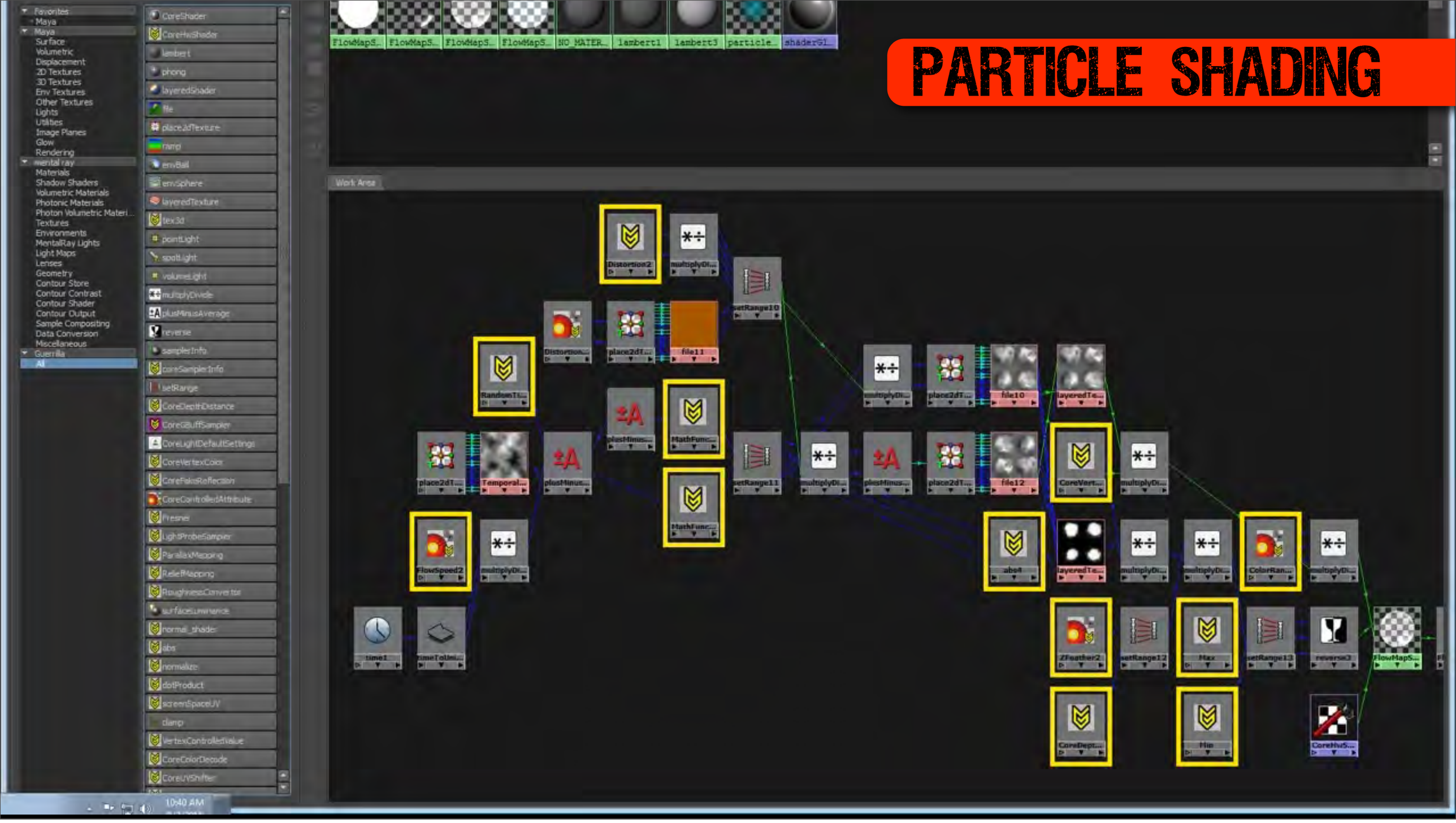
The other half lies in creating the shaders, like all of the shaders in our game these are set up in Maya....

PARTICLE SHADING

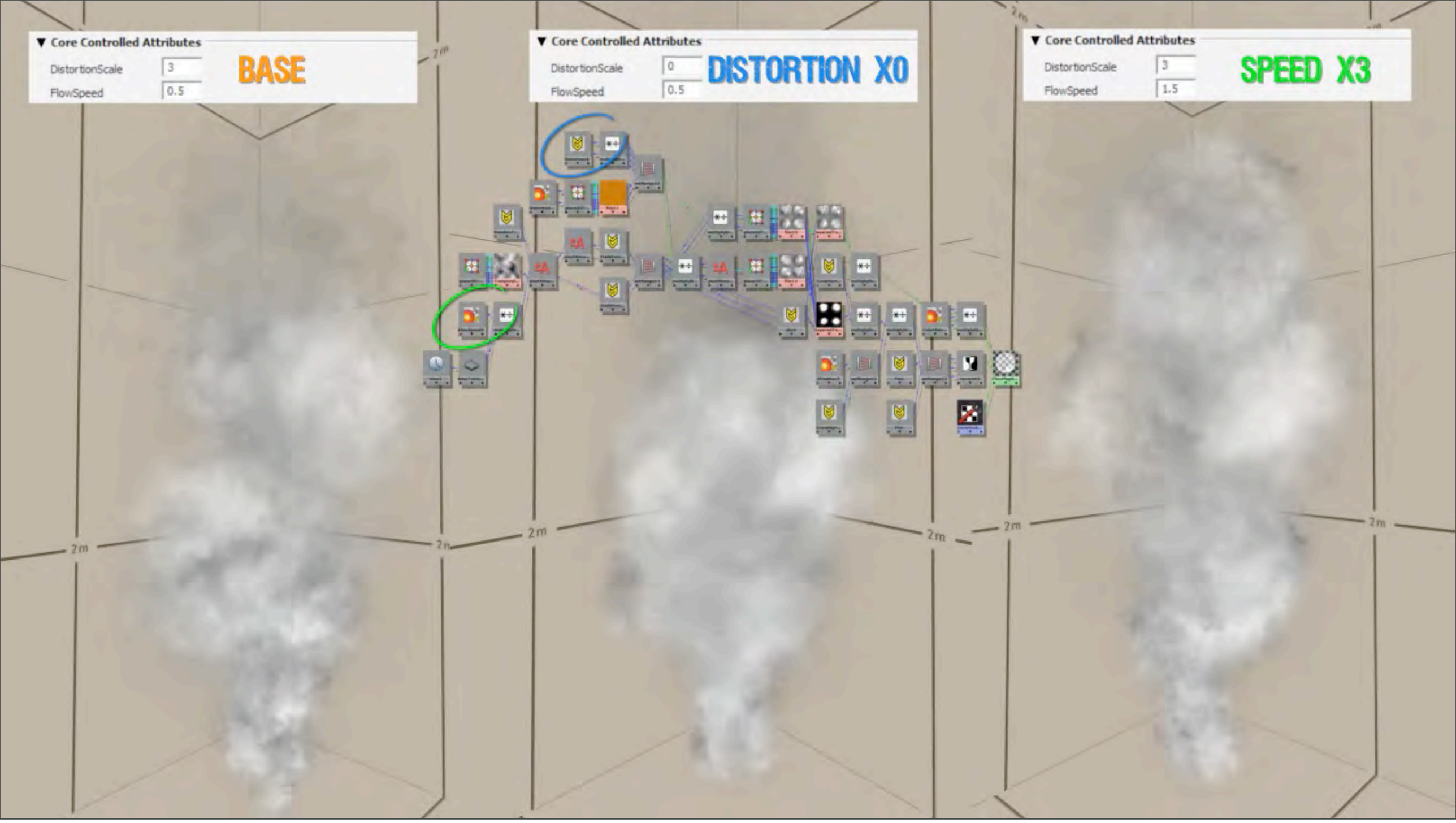


Here you can see a typical particle shading network. You'll notice a lot of familiar Maya nodes and a couple of our **custom nodes**.

PARTICLE SHADING



These allow us to set/control variables over time in our particle editor.



The next example show the result of that shader on the particle system. For the effect we only emit about 10 particles. The shader makes use of a flow map technique, you can find more on this in an awesome paper by Alex Vlachos called "Water Flow in Portal 2" presented at last years Siggraph.

In the middle particle effect we've adjusted the scale of the UV distortion.

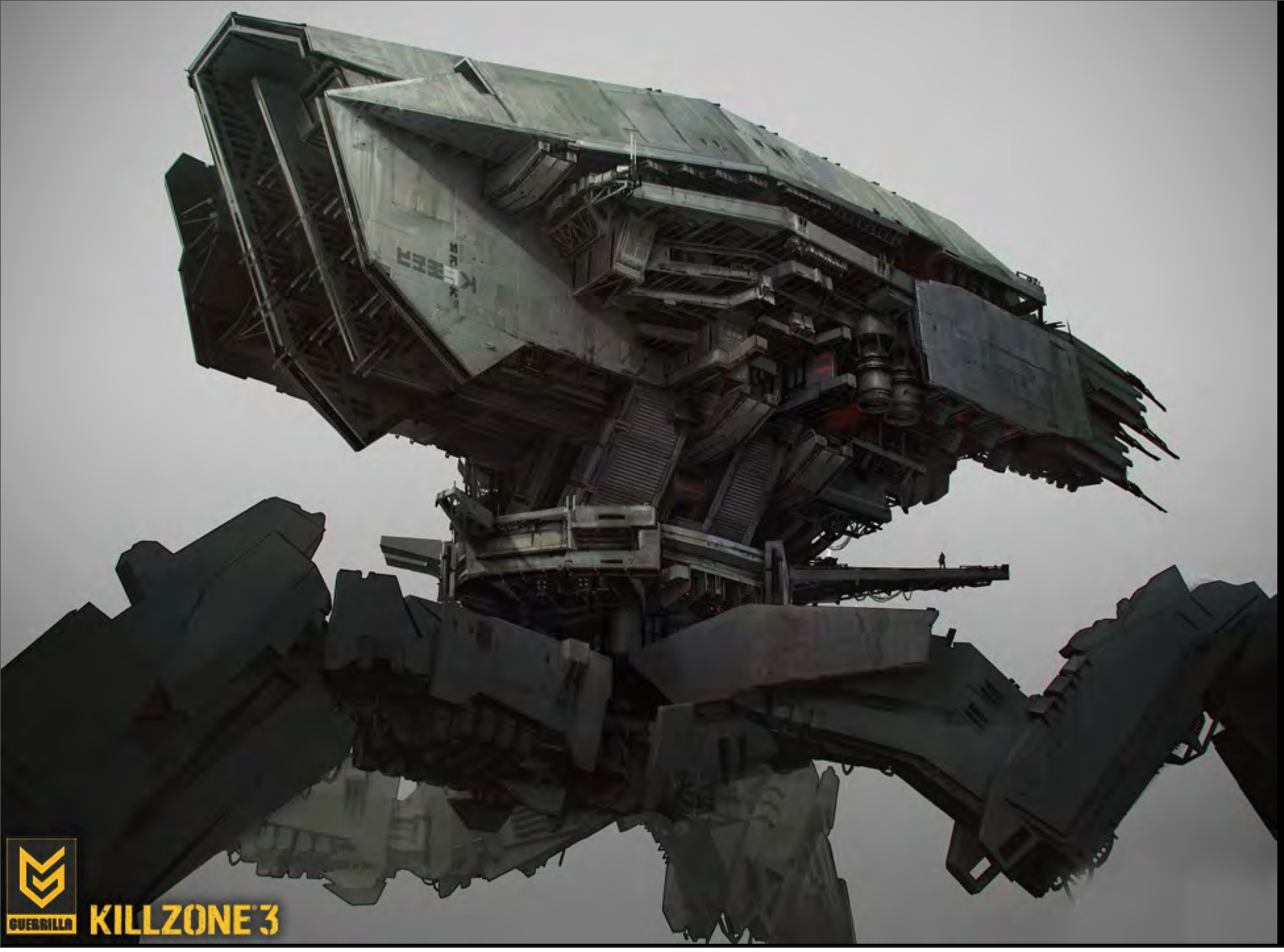
On the right we've adjusted the UV scrolling speed.

In this example you can see The color of our particles are controlled by multiplying a texture with the vertex colors. The vertex values over time are represented by the gradients at the top. These values can be adjusted in the particle editor.

This video shows the result of a short fun experiment that we never ended up using in the game. The goal was to see if we could use our particle systems to generate large scale background battlefields, really pushing our particle system to it’s limits. All the tracers, explosions and even characters you see are particles. Top right video shows you a close up of the battlefield.
The quality of art is very poor, but the point was to see if the particles would work.

Eventually we decided not to use it in the game because the lack of quality.

Hopefully this short demonstration gives you a glimpse of the possibilities we have with our particle systems.



To conclude my part of the presentation I want to show you 2 small parts of a level where you're fighting this metal beast.

These clips showcase all the topics I've just discussed.

Since everything on Killzone 3 needed to be bigger, that also included this end-level boss called the MAWLR. This 80 meter high tank turned out to be quite the challenge.

It was the geometry heaviest character we've ever made at about 250000 polygons for the hi lod and a 100000 for the low lod model. It took a good 6 months for our modeler to finish.

After these 2 examples JanBart will tell you more about the cutscene process.

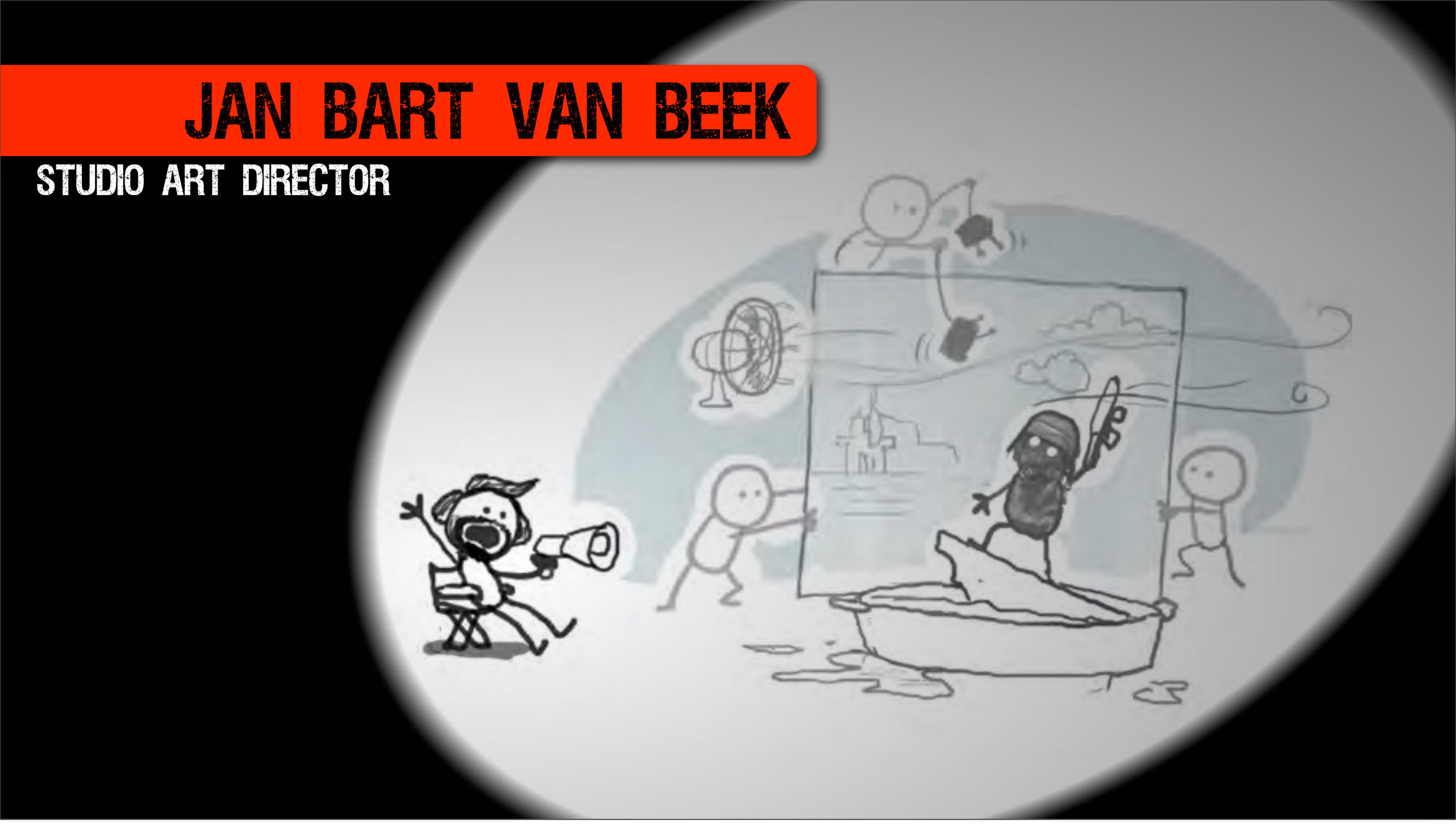
This first sequence shows you a typical encounter and the MAWLR in action....
in the topright corner you'll see a view of our AI agents at work.
For this particular encounter we have about 20 individual agents each perceiving, planning and executing actions.

and in this final video you can see some of the particle effects during gameplay...
you'll notice the particle trails we use on the missiles fired and debris falling down after impact...
and you can see the enormous amount of detail on the MAWLR model....

This concludes my part of the presentation, next JanBart will talk about cutscenes.

JAN BART VAN BEEK

STUDIO ART DIRECTOR



Hello again.
In our next segment I'd like to talk you
through our cinematics pipeline, as well
as some of the lessons we learned while building it.

As I mentioned during the introduction,
we have a real-time rendering pipeline.

Which means we output the frames
from our in-game engine.

There are a couple of direct advantages to this work flow.

The first of course is is a WYSIWYG workflow

Our artist are always looking at the final quality image
while they are working on it.

Although they still have to wait
on certain data conversion processes,

most of their changes and tweaks can
be seen instantaniously

Another advantage is that there
is no inconsistency in the final look.

I’ve got two clips of Killzone 2 here.

One is from the intro movie, the other is from the in-game cinematics.

It’s quite obvious to see that there are both
stylistic and qualitative differences in the renders.

This may be fine for the intro cinematic,
as the audience only sees it at the start of the game.

But these changes are quite jarring during the game itself.
They diminish the immersion in the world.
So we prefer to render
everything with the same renderer and maintain consistency.

Doing this also saved us a lot of time having
to convert in-game content to be able to
render properly in an off-line renderer.

During the creation of Killzone 2,
we also ran into some clear disadvantages
to rendering out the cinematics in real-time
on a consumer PS3 console.

The main problems are with the performance and memory;
all the cutscenes had to fit into the RAM of an
consumer PS3 and also run at 30 fps.

This put a lot of strain on the already
overtasked optimization team, so for
Killzone 3 we decided to use our
game engine as an offline renderer.

For Killzone 3,
we wanted to remove that disadvantage while
keeping all the goodness of a real-time renderer.

So we decided to use our game engine as if it was an offline renderer.

Although most of the KZ3 scenes ran at at 30 fps,
some of them dropped below 20 fps.

For Killzone 2 that would have meant a lot of work to get it running smoothly.

But for Killzone 3, we didn't have to worry about it, as we
would simple write every frame to disk as it got rendered.

Memory also became much less of an issue.
First of all, because we didn't need to load all the data
that the game would need to run.

But more importantly, because we don't need to run it on consumer-level PS3.

Because, we have are own special PS3's.
They look exactly the same, but it says tool on it.

And they have twice as much memory.
Which makes a lot of difference.

WHAT YOU SEE IS WHAT YOU GET

CONSISTENT LOOK ACROSS GAME

SPEED OF ITERATION

ADVANTAGES

A WYSIWYG workflow and consistency of
look are certainly important,

but they're almost nice-to-haves
compared to our biggest advantage.

<click>

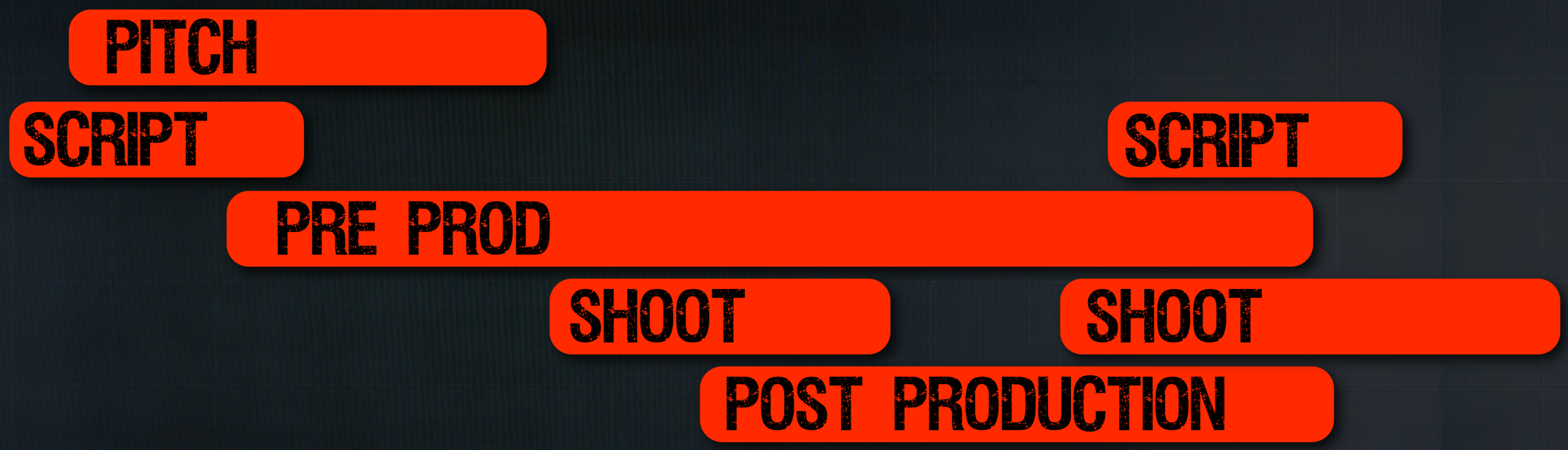
Which is our ability to create and iterate very quickly.

IDEALIZED MOVIE PRODUCTION TIMELINE

In traditional movie production...
Scriptwriting, pre-production, shooting, and post-production
are all clear stages of the process..
and ideally you don't want to move forward
unless the previous stage is done.

MOVIE PRODUCTION TIMELINE

Ofcourse in reality it might look more like this and the stages overlap more.



PROBLEMATIC MOVIE PRODUCTION TIMELINE

But most producers will try and avoid
somethign like the following scenario as much as possible.

MOVIE PRODUCTION TIMELINE

In game development terms this is called a classic waterfall development methodology.
And it doesn't really work for us.

The reason for that is that
some of the similarities between movie production
and game production are only superficial.

At the end of the day, we're making software.

And the quality is of software is highly dependent on constant testing, feedback and iteration loops.

The strongest force in game production isn't the director,

it’s the playtesters.

And if they say it sucks, it will have to change.

This requires us to be very flexible and
be able to change things very quickly and at any stage of production.

KILLZONE™ 3 PRODUCTION TIMELINE

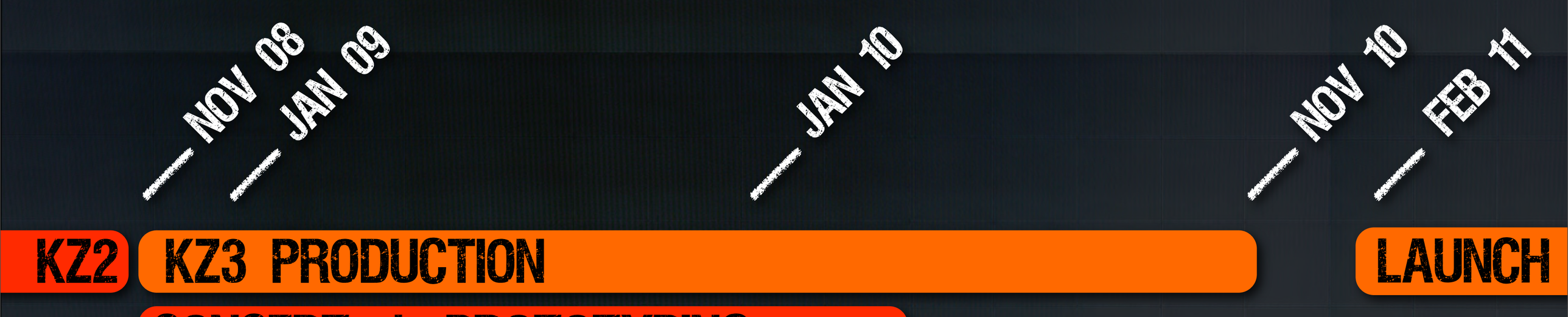
Let’s take a quick look at Killzone 3’s production timeline.

We finished KZ2 in November 2008, in time for a launch in February 09.

KZ3 was already slated for a launch in Feb 11.
So we had till about november 2010 to work on it.

Defining the high level concept and
then building gameplay prototypes
takes a considerable amount of time.

Building a drivable tank for example can
take over a year to tweak and get it playable and fun



KILLZONE™ 3 PRODUCTION TIMELINE

Once we have enough working prototypes
to start on certain levels, we're off into production.

This ofcourse last till the very end of our schedule.
Litterally the day before we start printing the discs.

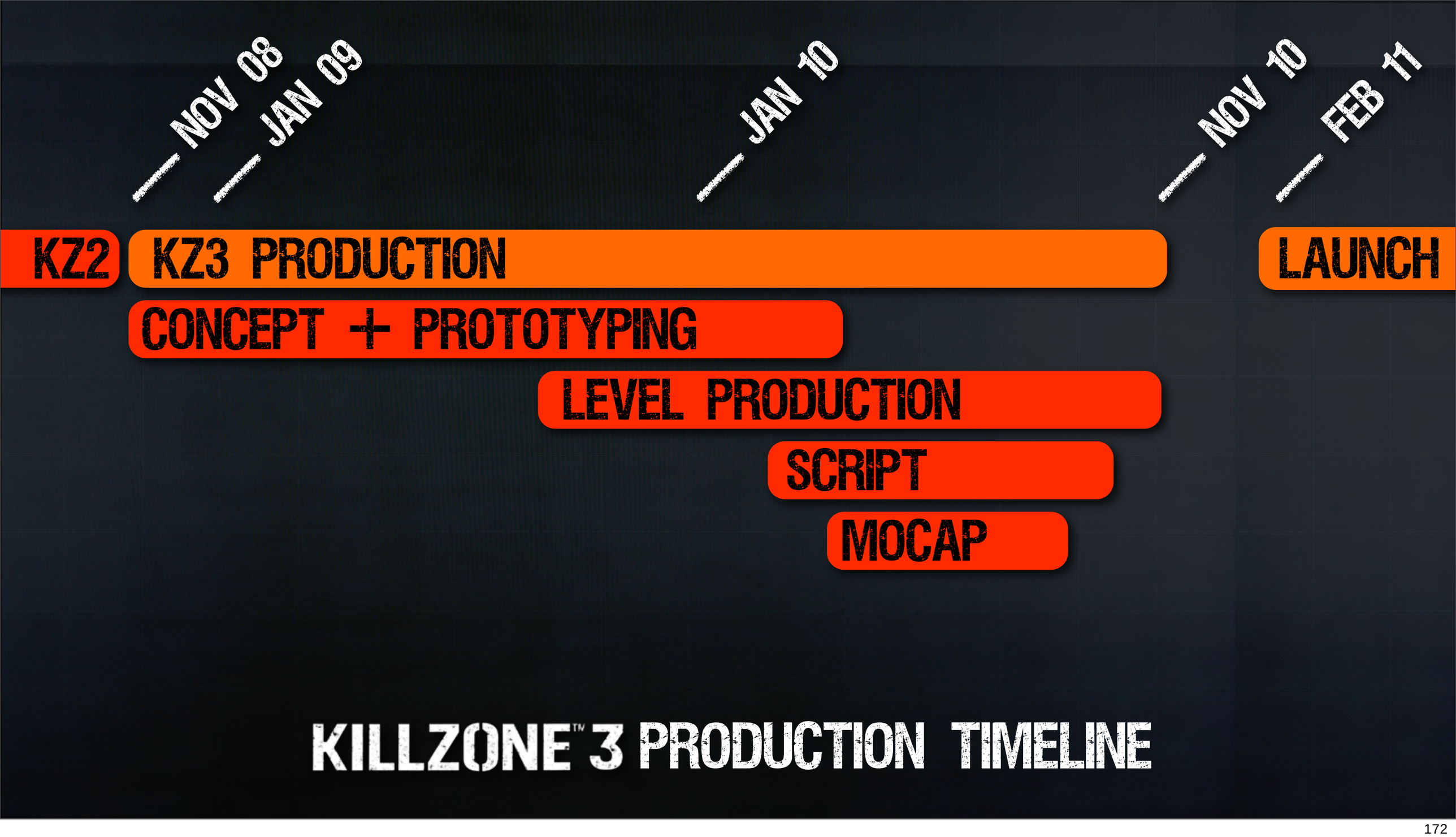
Usually we start with 3 level or so... about a third of the game.
At this point we don't know what will end up in the final game.
Something features might still fail.
And that may require big changes to the game.

But still at some point, even though the game isn't finished.
we'll to commit to the story
and get the cinematic process started.

This also means that while the scripts are being written.
They have to constantly be adjusted as the level are still changing.

We very quickly go from script to an 2D animatic.
This way we can test the story as well,
as players can see at least very rough storytelling.

Here how that stage looks.
All the dialogue is mock-up and usually done by the animators.
but it gives us a fair idea whether the scene is going to work.



Once we have a couple of scripts we'll start motion capture.
This is not a 2 or 3 week event were we capture everything,
because that would require us to have all scripts done.

With some levels not being designed yet, that's simply impossible.
And we can't wait for the level to be done either,
cause it might only be done a week before we ship finish
and that leaves us with too little time to create the cinematic.

Motion capture happens over a 4-5 month period,
usually there are about 10 session, each of about 3 days long.

And as you can see Mocap end before the last scripts are written.
As there is still ADR to be written and in-game dialogue.

The mocap session can get quite complicated
because of these weird shooting schedules.

It's logistically almost inpossible to have all
the actors available on the same day.

So the shoots for a single scene are often spread out over multiple sessions.

In some cases actors even need to play
multiple parts of the scene, or stand in for another actor.

In this case for example Malcolm MacDowel
who played the role of Yoram Stahl, the game's villain,
wasn't available in the day that we needed to shoot.

So he had to record his performance on a later date.
So Andrew Bowen, who normally plays the role of Sev, the game's hero,
stood in for Malcolm.

It far from ideal...
And we try to avoid this as much as possible,
but it's economically unfeasible to lock an actor down for 4 or 5 months.

<Click>

As the first mocap trickles in,
we can start the actual production part of the cinematics.
And...as you can see, at that point we only had 6 months left.

We had planned for this and our sister Studio in San Diego
was contracted to help us out. T

hey were responsible for all animation, scene assembly and camera work.

While Guerrilla would focus on effects, lighting and integration into the game engine.

Aside from cleaning up all the mocap data,
and hand animating any parts that could not be captured,

The San Diego studio also had two intersense virtual camera capture stages.

And this has helped a lot to create a more cinematic feel as we could hire profesional directors of photography to define the camera layout.

Halfway through the process we get something back like this
<Click>

At this point we get it back and bring it into the engine.

Now It’s already clear that the amount of time left at this point is very minimal.

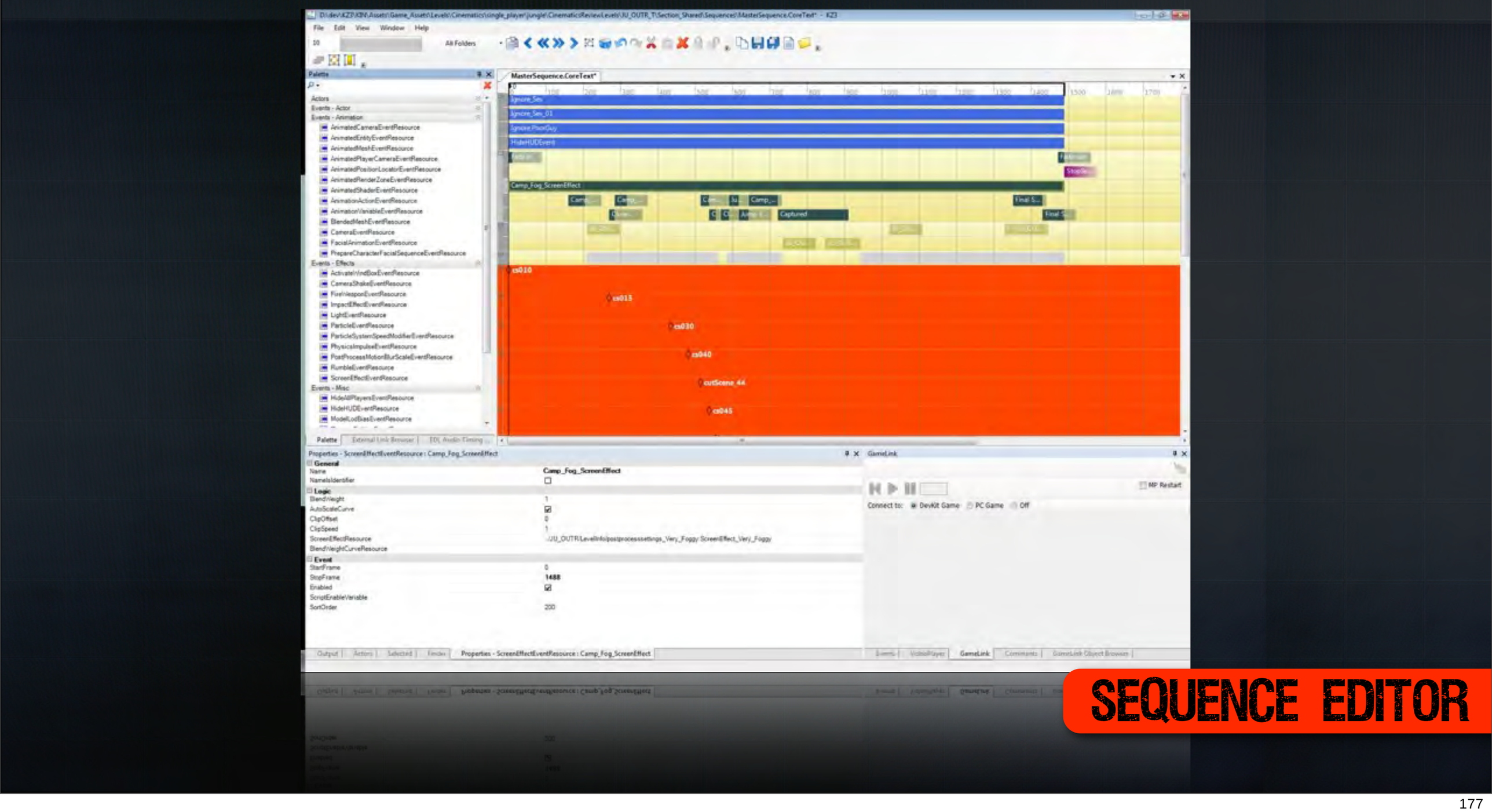
All together we have about 6-7 months to do
all post-production on 72 minutes of cinematics.

But that time constraint isn’t necesarily the biggest challenge.

<click>
It’s the enormous amount of stuff going on in paralel.

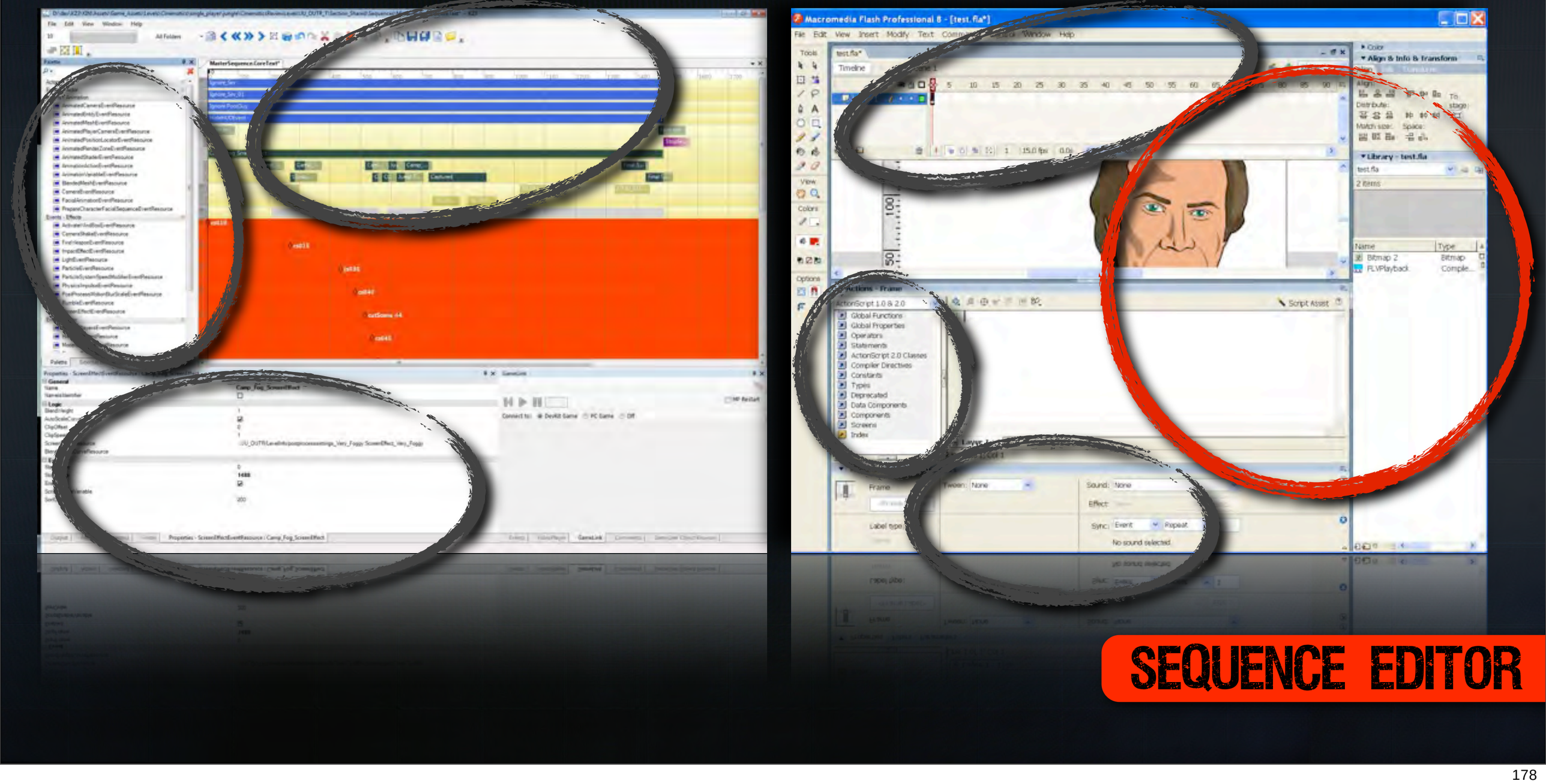
The scripts aren’t done yet, but we’re already in post production.
Your producer’s worst nightmare.

Now that we’re getting final data back, we can start putting the content into the game.



For this we have a custom set of internal tools.
The most important one is the SequenceEditor.

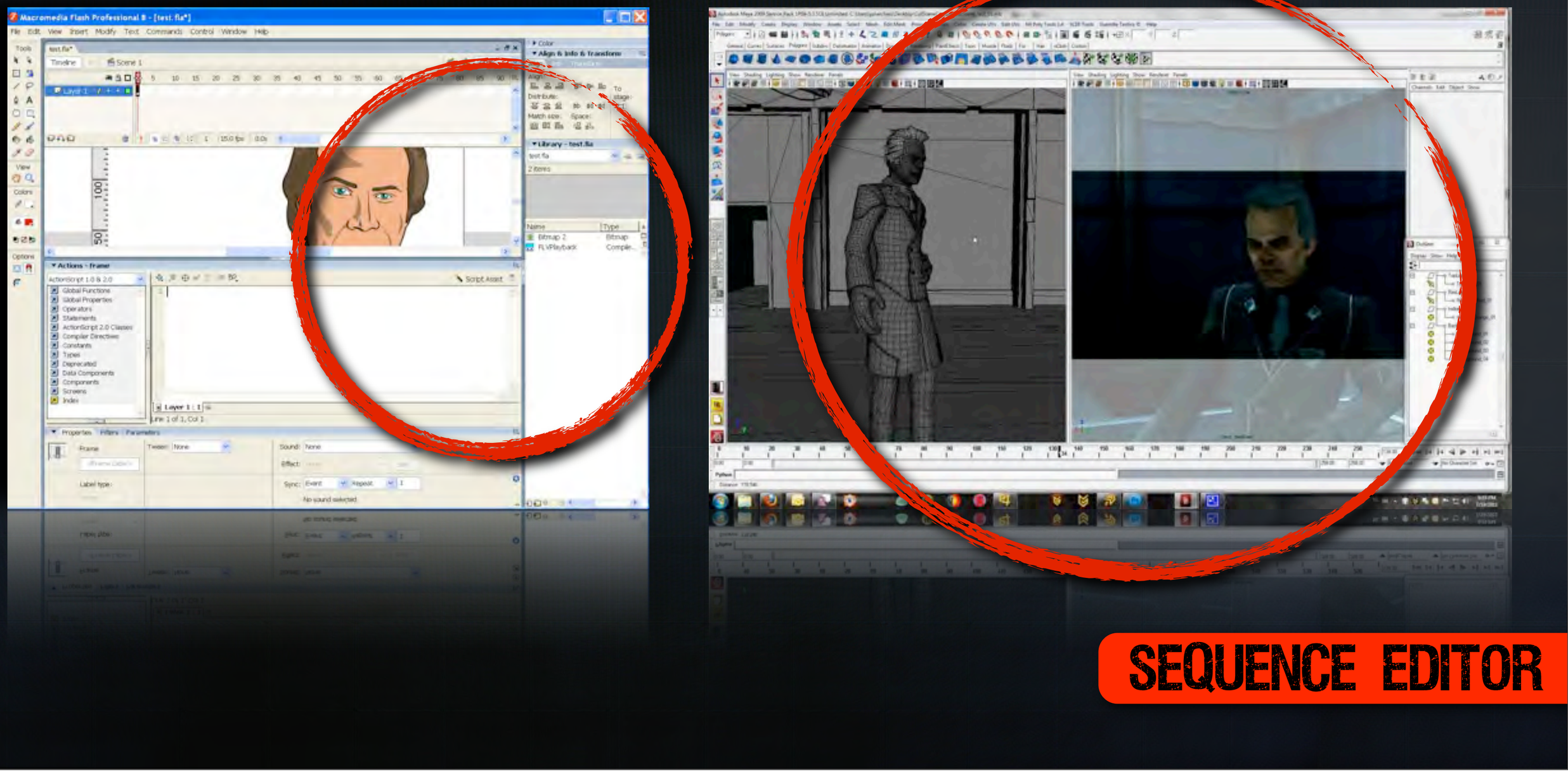
It may look a bit like a video editing tool,
but it’s actually a lot closer to interactivity tools like Flash.



Similar to Flash,
there is a timeline, an list of events and a property editor for those events.

The tool doesn't cut footage together, like a editing program. Instead you place events on a timeline.
Such as "creates a character",
"plays an animation"
"trigger a particle effect or a sound"
"pause the player or the kill and enemy"

<Click> This part of flash...
Where all the content creation, positioning and animation is done..
isn;t done in this tool, but is done part is in Maya.



Maya is still out primary tool.
Michal already showed how we use maya to place objects and lights
and can see the results immediattly both in maya, as well as on the ps3.

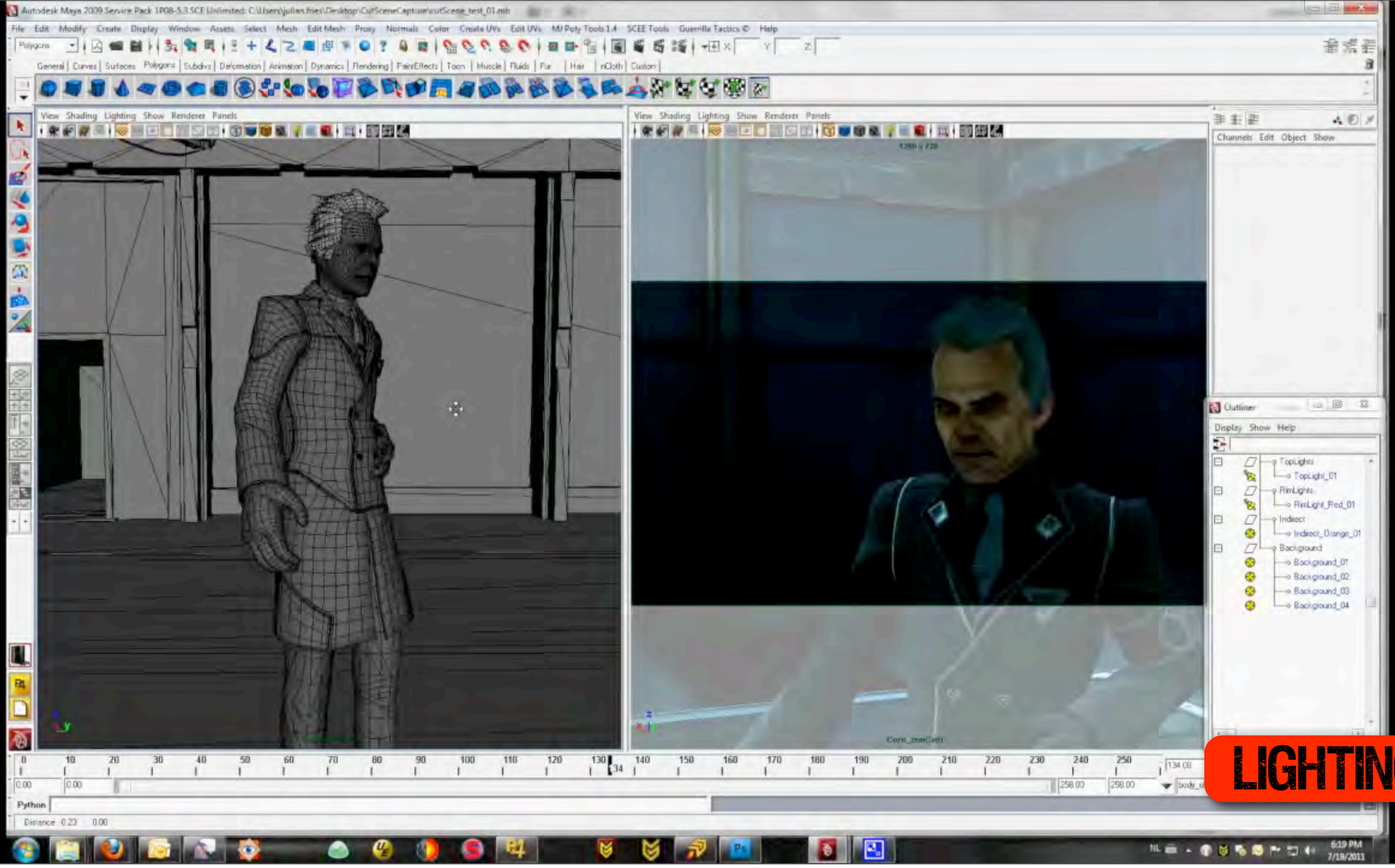
Paulus also mentioned how our level creation pipeline is all based around maya.

For some game developers this may sound a bit unusually,
especially as most First-Person shooters are build with
custom editors like Epic's UnrealEd, Crytek's Sandbox or in
the case of Call of Duty, a heavily
modified version of the old quake engine.

For us the choice was three-fold.
We wanted our own engine, but we didn;t want to
invest in building an editor, which is dozens of manyears of work.

Second, we figured it would be easier to train artists
of all their tools lived within maya.

And third, anyone who can program a bit of Mel or Python
could help build the tools and tailor them to our needs.



Let's get back to the cinamatics.
Once we have everything in game we can start lighting the shots.

As Michal has told you our game engine is a deferred renderer,
which gives us much more flexibility and speed during
lighting then a traditional game engine.

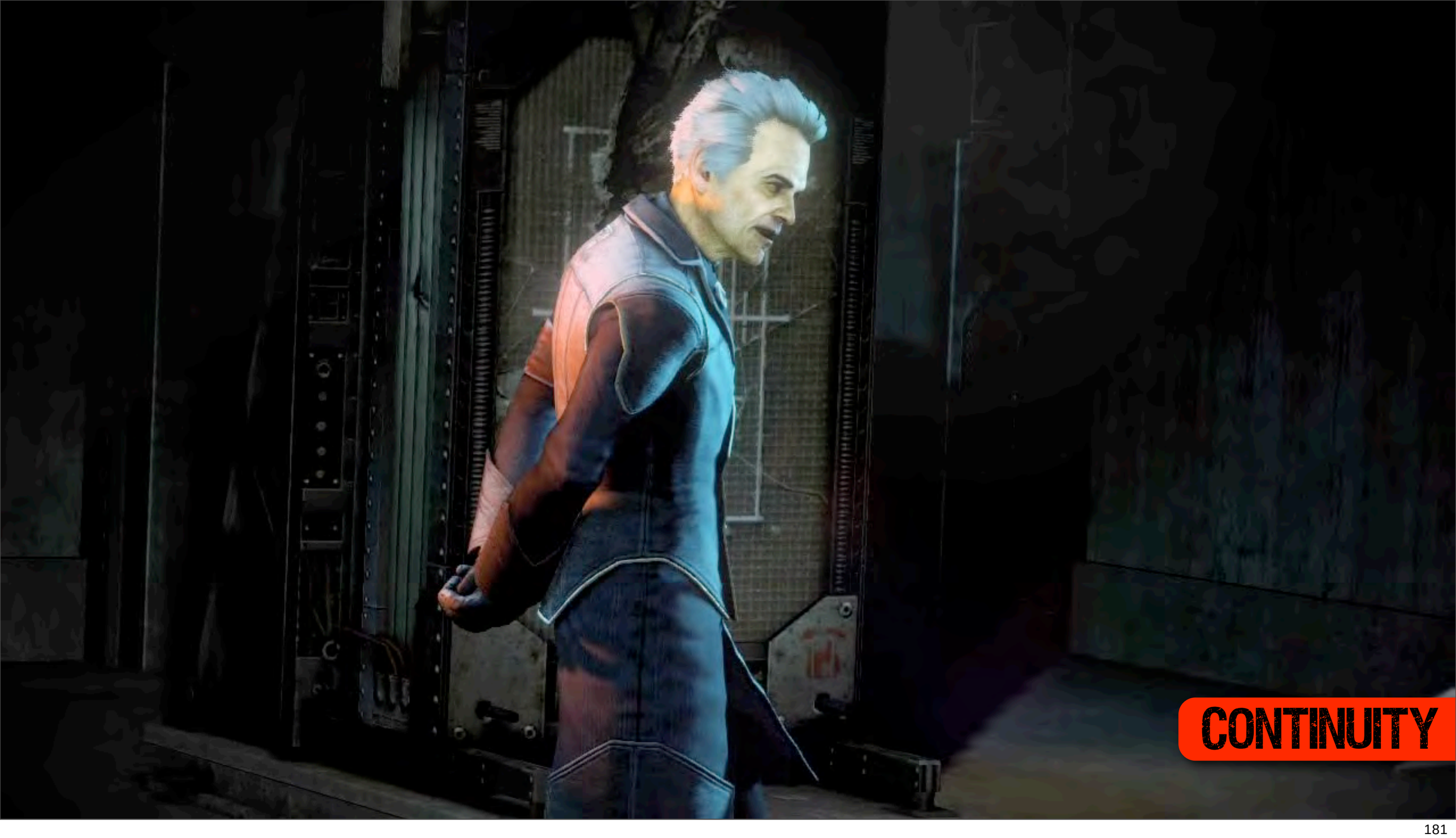
But an even bigger advantage is that it runs directly in the maya viewport.
This allows us to add, copy, and tweak the lighting of the scene in real-time.

The system only support omnilights and spotlights,
but we had little need for more advanced lights.

Also as the system didn't support any real-time
radiosity or radiance, we had to use secondary omni lights
to create the illusion of bounced light.

The nice thing about a defered renderer is that
you can go pretty nuts with the amount of lights in your scene,
as the systems performance is mostly dependend on how many lights you have per pixel.

That means you can have hundreds of lights
in your scene while still remaining interactive.



An important aspect to keep in mind is
that we're not working with footage.

It all's 'live' game content.
It remains an interactive three-dimensional
scene throughout the whole process.

At any moment we can take the camera and start walking through the scene.

Now, If you do that you would see a lot of continuity issues.

Characters jump around,
light jump all over the place,
effects suddenly pop in and out.
Props appear out of nowhere,
whole sets disappear

It's quite bizar looking.

And this is also where a lot of our problems began.

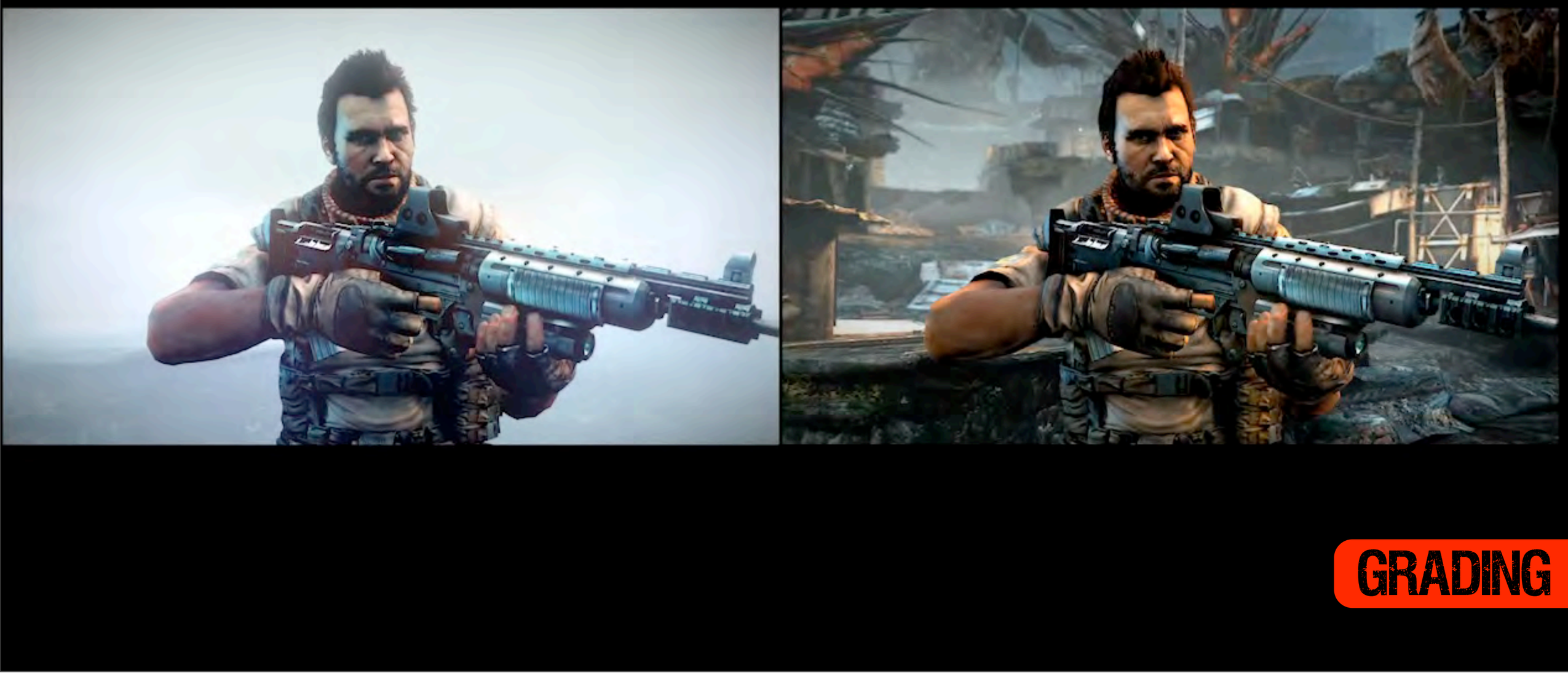
Because any effect that is temporally based,
will start looking quite broken in such discontinous time.

Motion blur would carry over from one shot to another, so every first frame was always blurred.
Particle effects would carry over into the next shot.
Lights from one shot would still turn on one frame too late.

The worst problem we had had to do with a legacy problem in our game logic.
that would only allow us to place event on every second frame,
so at 15 fps, instead of 30.

Before we finally fixed this, animators had to spend an enormous amout of time
rejigging all the camera cuts to happen on odd frames only.

It’s one of those things we really didn’t anticipate.



A very important part of our post processing is the grading process.
<click to start movie>

COMPOSITING

<movie start automatically>

The last stage in our process was compositing.
As the entire scene is created directly in the PS3.
we were hoping not to have to do any compositing.
and we didn't really prepare for it.

Unfortunatly the script called for a couple of scenes with fairly complicated holographic projections that couldn't be done in our engine.

Compositing something like this in Nuke or Combustion is fairly straight forward.

Because we have a deferred renderer we could use many of the buffers it automatically creates during it's normal rendering process.

All we had to do is write some code to out put depth-buffers, or other buffers to disk as well so we could use those during compositing.

Where we really ran into the problem was when we tried to composite the same scene for the 3D version.
And this had a lot to do with how we implemented 3D.

Killzone was one of the first major titles to support 3D.
And because of that we got to show Killzone 3 on the Jimmy Fallon show.

Jimmy was kind enough to give our marketing a little boost by
shouting
<click>
Oh, my Gosh, oh my gosh ! for about 5 minutes straight
till his jetpack crashed into the sea at which his dying word where
<click>
“look at the bubbles !“

To a certain extent supporting 3D in real-time is fairly straightforward.

We don’t have live footage elements that need to be combined with CG.
in that regard our compositing needs aren’t that heavy,
so it simplifies the process.

Also, we were already planning to support the game
in 2-player split-screen mode,

so...most of the complicated work of heaving to render
the game from two different cameras was already done.

Over the last three days I've seen various talks about
how developers have implemented 3D in their game.

And pretty much everyone is doing some form of reprojection.
We're not doing that, we're actually rendering the screen twice.

The decision was part circumstance. We were one of the first to go 3D
and reprojection techniques hadn't really been developed yet.

But once reprojection became an option, we noticed quite some artifacts that
we felt wouldn't be acceptable.

Namely : because of the need to "fill in the holes" the amount of depth needs to be much more limited then two camera renders.

Also transparancies. All the reprojection guys have sorta glossed over it,
but in a game where you're constantly looking at
explosions, gunflares, smoke and other participating media
making sure that all of those work in 3D is vital to the experience.

Finally there are some other nice-to-haves.. water, reflections, speculars,
all works as they are supposed to work.

But there are drawbacks as well.

<Click>

To deal with the additional rendering overhead
while still keeping a 30 fps framerate
we had to render the screen at a resolution of 640 by 720.

This is how the game itself rendered it's content,
and although for the cut-scenes we could have chosen
to render both frames at full resolution,
we decided to stay consistent with the in-game quality.

Also it would have made the cinematics
twice as big on the blu-ray disc and
we were already using 42 of the available 50 gigabytes

Now, although you can be cheeky and say it's still 720p,

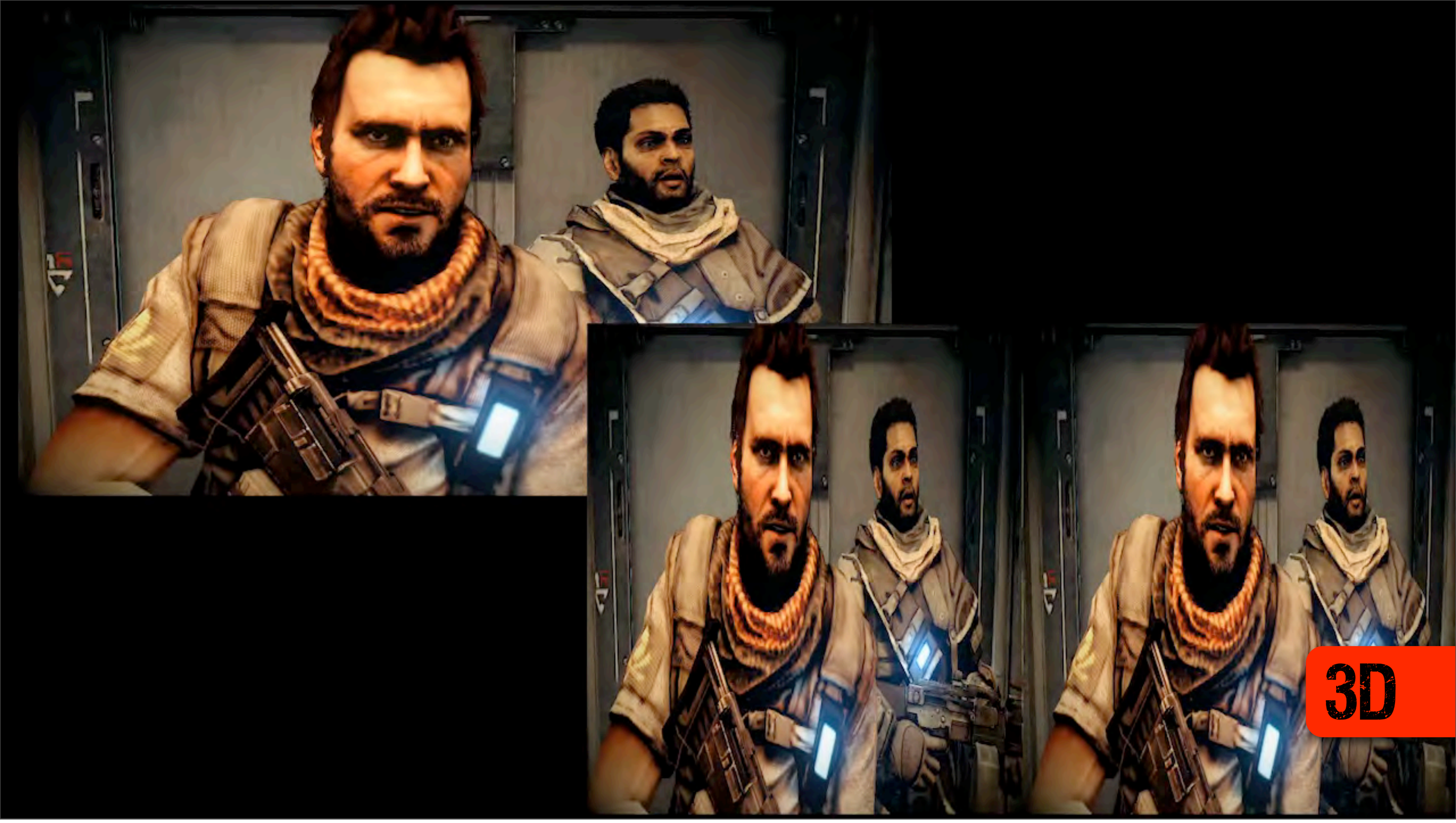
If you look at the image here, you can see there is some slight blurring on the textures,

<click>

but the most noticable artifact come from aliasing effects on vertical lines.

In general no one really noticed though, as the artifacts tended to be hidden by motion blur and the compression of the movie itself.

Also there is some level of natural anti-aliasing going.
The aliasing in both images is different, so when someone focusses on the artifacts
both eyes see a different form of aliasing and the brain naturally merges those two together.

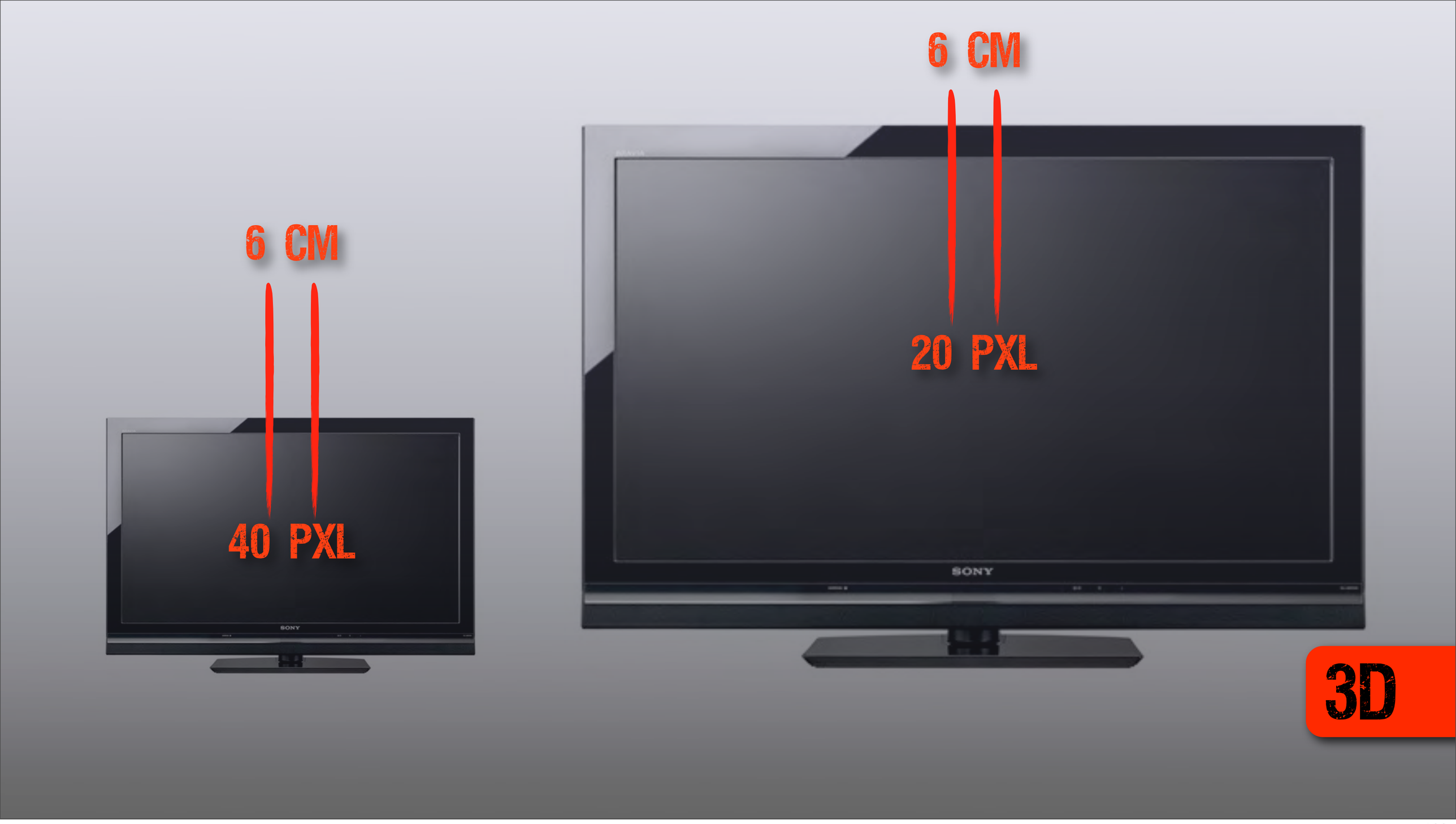


When rendering for 3D for cinema,
it's traditional to render a "right eye" frame that is
the primary frame and is also the output for a 2D version.

This is a good cost saving measure
as you only have to render one additional camera for the 3D version.

For us there is practically no additional rendering cost.
So, we chose to offset both cameras from the center axis.

This helped with close up object like the gun that is always on-screen.



Our target medium is a television,
and we have to take into account that our audience
has an enormous variety in different sized television.

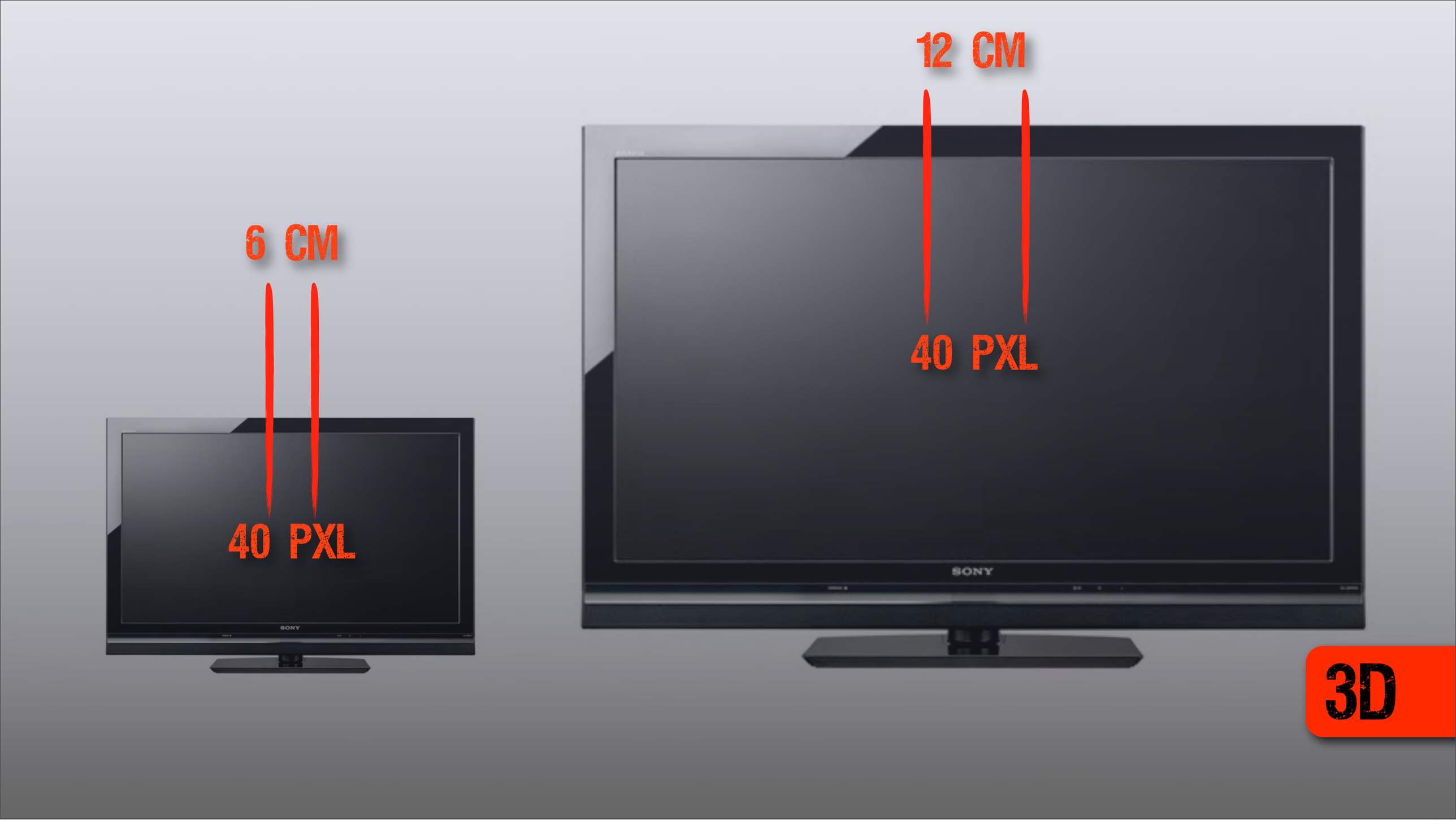
The size of a television has a huge impact on the 3D effect.
Because the farthest point in your scene should always
have a seperation of about 6 centimeters on screen,
no matter how big the screen.

<click>

So while on a small 20 inch screen you may have need 40 pixels seperation, on a large 40 inch screen, you need to have 20 pixels.

<Click>

This is also the reason why you see very
little seperation between the frames when
looking at 3D on a cinema screen.



If you don’t account for the screen size,
people won’t be able to focus their eyes on distant objects.

The will litterally have to diverge their eyes,
which is fairly difficult and headache inducing.

The reverse can also happen, this results in a reduced sense of depth.

In the game we adapted for this dynamically,
by reading out the specification from
the TV via the HDMI connection.

But as our cutscene are rendered out,
we told the system to always assume a 40inch screen.
It’s not ideal, but it was a good middle ground.

And this is where our compositing problem in 3D came from.
As the cameras where being created by our game code,
it was really hard to replicate their exact placement in maya.

The solution was ultimately to lock the camera seperation for all our cut-scenes.
And then spend several days tweaking the placement of
the compositing plates by hand to get it to match up.

As the overhead of this was simple to big
and we had several other shots that also had compositing requirements,
we decided to hack a very rudimentary image
streaming system into our engine,
so we could show externally rendered texture
sequences onto simple objects.

There was a size limitation of about
128 by 64 pixels and a 64 frame limitation,
but we've recently expanded the system so there are no more limitation
and we're one step closer to removing our need for
compositing outside of our engine.

CONCLUSION

We’re getting to the end of this presentation
as well as the end of a great Siggraph.

It seems that each year more and more attention is given to
real-time rendering techniques.

REAL TIME RENDERING

There have been many game developers that
have presented their latest findings.

But also Dreamworks, Pixar and ILM have all presented their
tools for real-time lighting at Siggraph over the past years.

If you're working in the movie industry,
It's likely that your R&D team is already looking into real-time rendering.

Probably **not yet** as a replacement for final quality rendering,
but as a way to improve the workflow
and make it **faster to iterate.**

QUALITY
IS
ITERATION

All quality is dependend on iteration.
Which usually means that faster iteration improves quality.

A friend of mine mentioned the other day that improvement to iteration time
aren't gradual, but they're based on human time scales.

a 10% or 20% speed-up doesn't mean anything.

but going from over-night rendering
to twice a day is a big leap.

You can now have two iteration a day.



Going from twice a day to every hour... that's a big step as well
That's the amount of time you take to eat lunch.

After that it's 5 minutes.
<click>
The amount of time to get coffee, or check your email.

But the next step isn't 10 or 20 seconds.
waiting 10 seconds is just as disruptive to your flow as waiting 5 minutes.

<click>

The next step is that you don't
have to take
your eyes of the screen
as you tweak your sliders.

But real-time interaction is something the game industry already has.
Our biggest challenge is yet to come.

Avatar was a bit of a watershed moment
for some of us in the games industry

James Cameron showed us a new world,
and a lot of us wanted to go explore that world.

Avatar has become a the new benchmark.
And many in our industry have the intention to take on that challenge.

Before the end of the decade
We will render world like avatar in real-time.
There is very little doubt about that in my mind.

But we'll add that one ingredient to it that only games can do.

The freedom to leave the camera path

and to go and explore the world one their own.

Thank you.