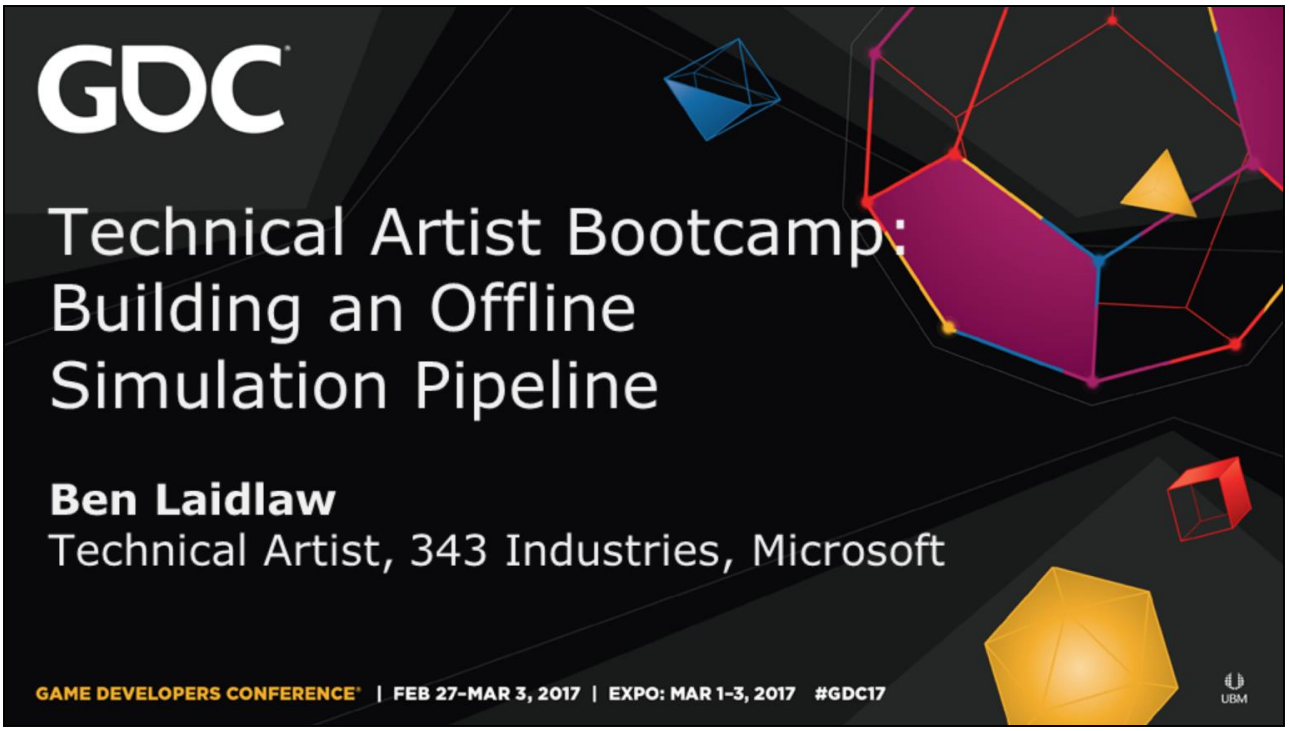


- 00:00, 00:35, 58:15

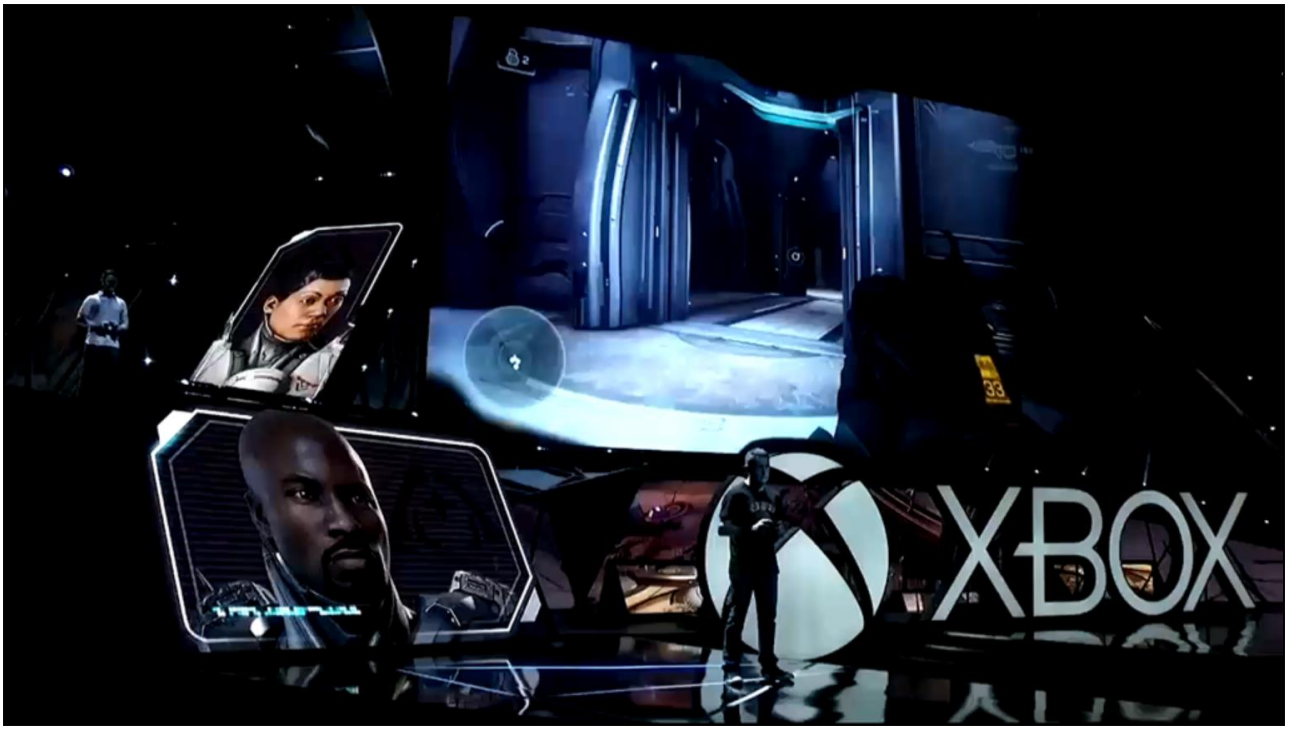
Welcome Everyone,

As TAs we are tasked with keeping up with **consumer expectations**.
At ever demanding Frame Rates, Screen Resolutions,
and for those next big moments.

My name is Ben Laidlaw
and I'm here to talk about using **offline simulations**
to create **complex assets** that you would otherwise
would not be able to author by hand.

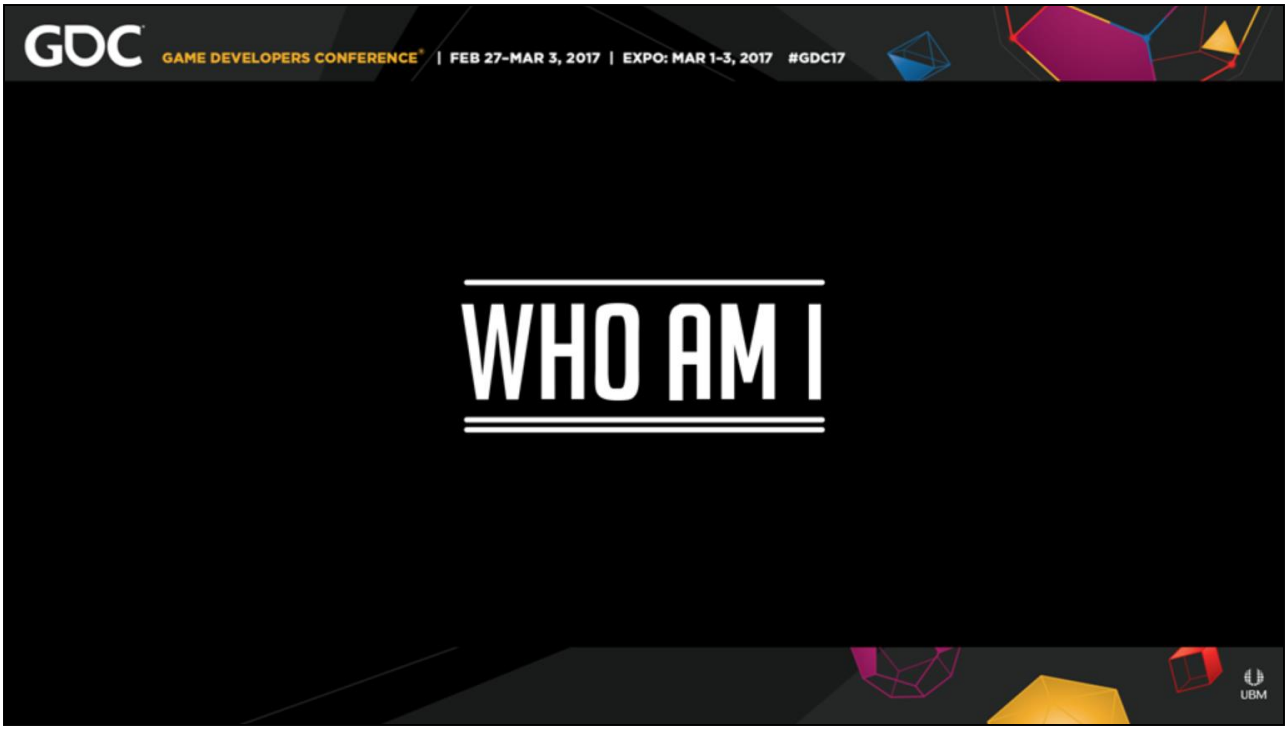
Like Photogrammetry and Motion Capture
We can capture higher fidelity assets
for our games with simulations

So **let me show you** what I'm talking about.
This following video clip
Was from our Halo 5: Guardians E3 Headliner



- 00:35, 01:00, 57:25

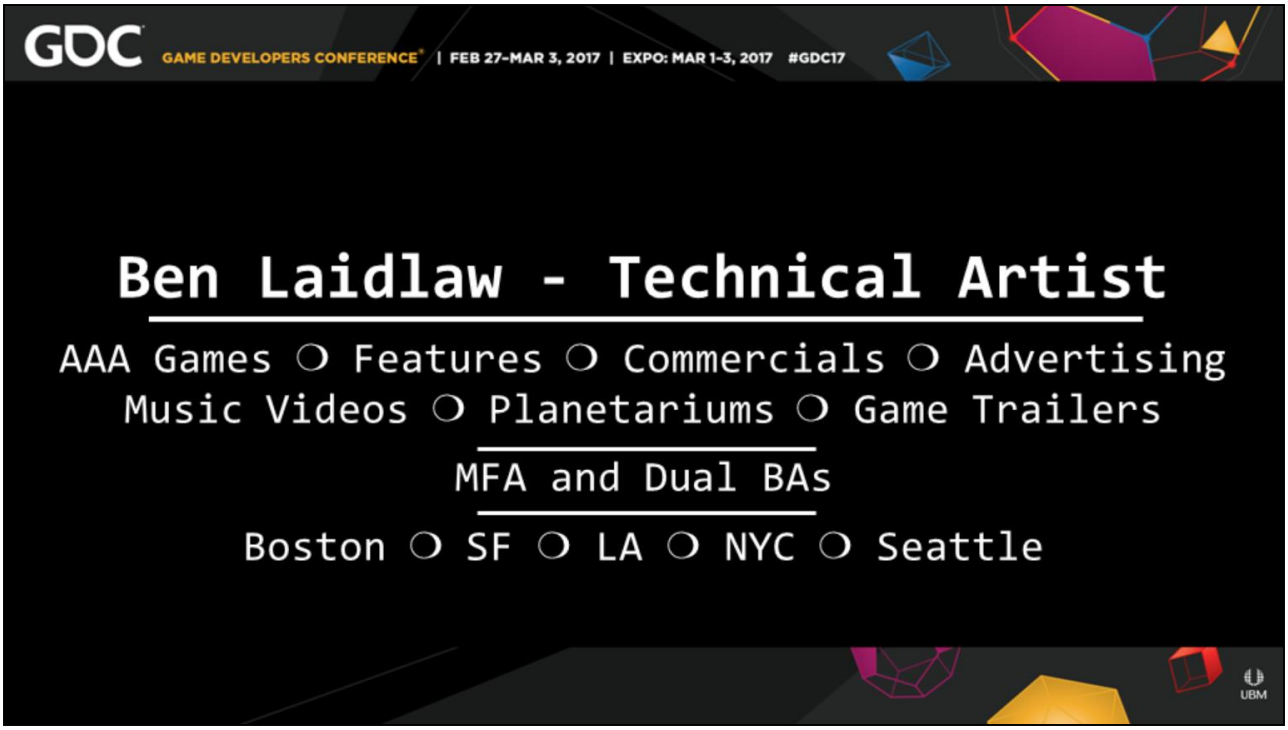
Halo 5: Guardians, E3 Headliner



- 01:35, 00:10, 56:15

So who am I?

For a Technical Artist
This may be the **Greatest existential question**
of all time.



- 01:45, 00:20, 56:55

My name is Ben Laidlaw.

I have 3 Art degrees
including an MFA from across the street
at AAU.

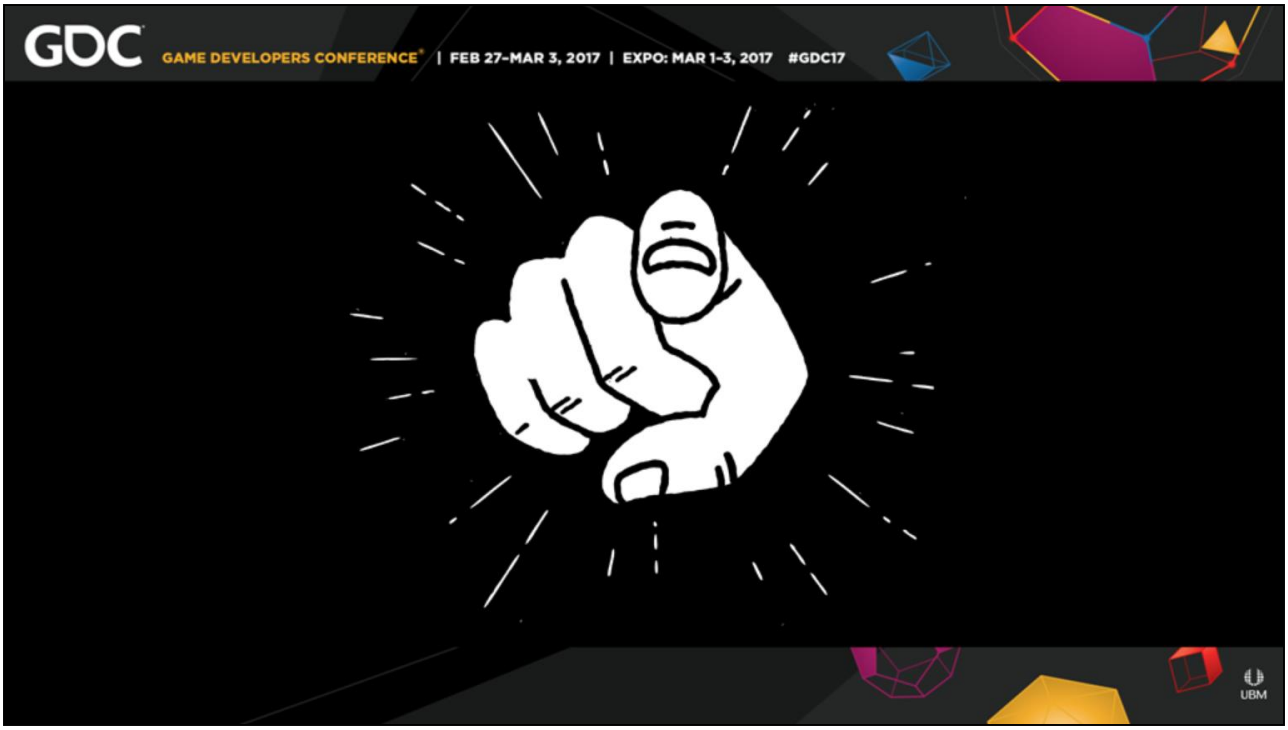
I've worked across North America
at multiple studios.
I've worked the gamut of 3-D
from Commercials, to Films, to AAA Games

However, since we all work in a visual medium
Let me show you a video of

some of my other work real quick.

- 02:05, 01:00, 55:55

Who am I Video



- 03:15, 00:45, 55:10

With a **quick show of hands**,
how many people use simulations in their content creation pipeline?

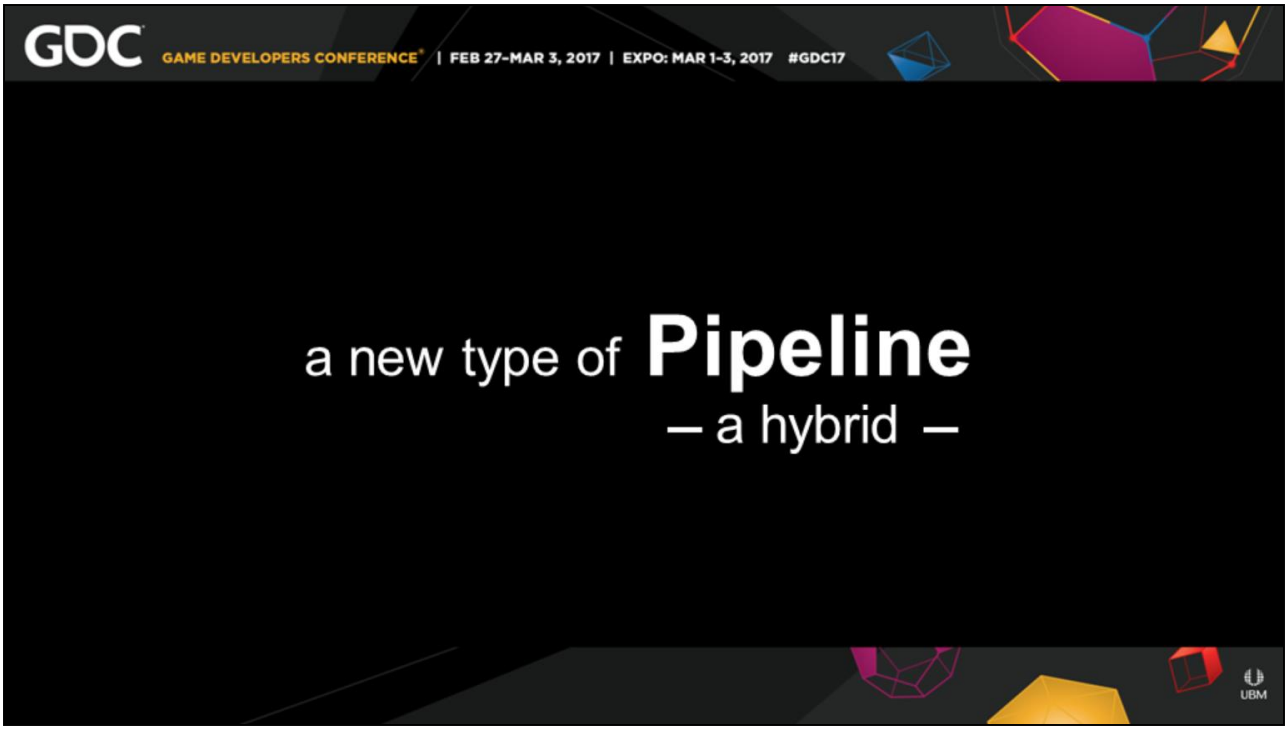
Ok cool about… # of you.
I'll tailor the talk a bit to focus on that.

In general this talk will be more of an **overview** of all the aspects required in simulated content as opposed to one type of simulated content.

So whether you are **using, managing, or implementing** this pipeline,
There should be something for you.

However at the **end**.
Feel free to ask detailed questions on any of it.
It’s a very deep subject.
I will also **point out** some of my tools and setups and some **additional resources**,
To help you get started directly after this talk

There will be **QA** time afterwards,
if you don’t want to step up to the mic.
You can contact me Tech-Art.org



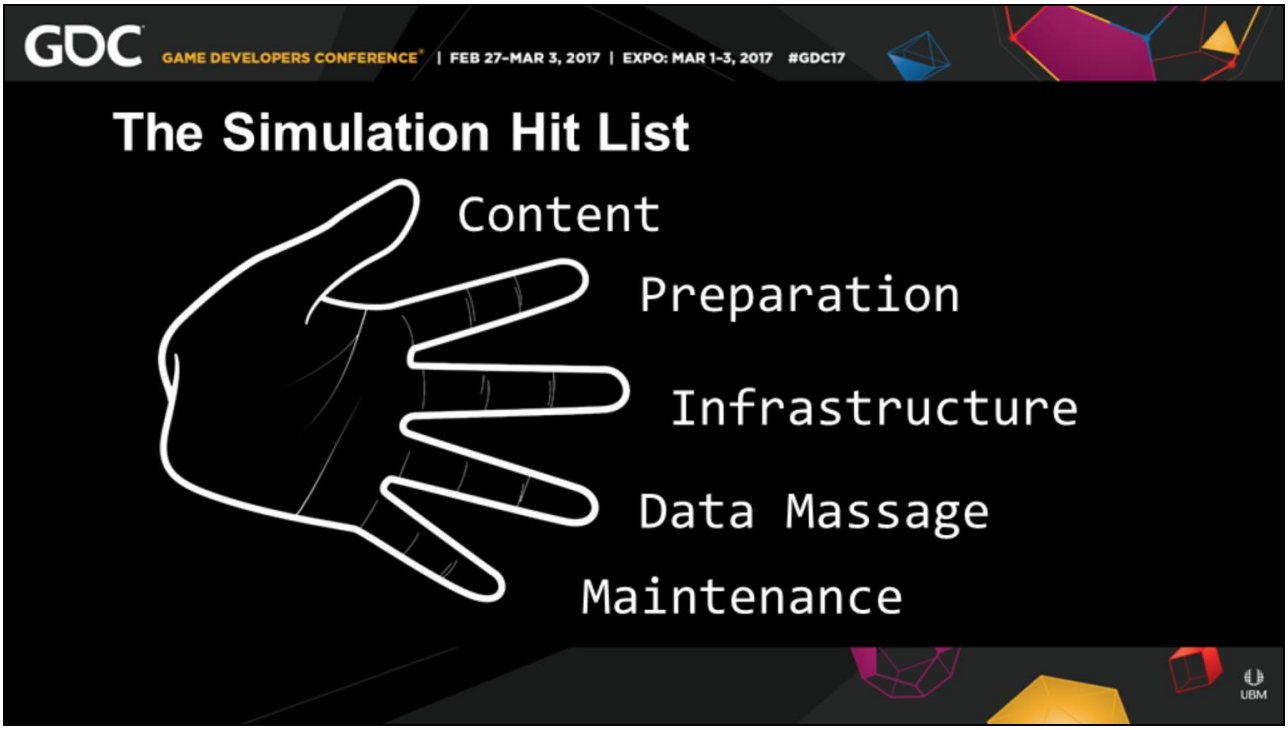
- 03:50, 00:25, 54:45

In general simulated content requires
a **different type of content pipeline**.

You'll be familiar with a lot of the steps,
so it's more of a **hybrid**, than a brand new pipeline.

This will **work at** AAA studios and tiny Indies.
Everything is scalable.
I have set this up on my own at 343 for Halo,
I do this on personal and commercial projects
and I have worked as a cog doing this at larger

studios.



- 04:15, 00:45, 55:30

This is our simulation hit list.
I packed them into 5 simple categories
to remember.

- **Content**
- **Preparation**
- **Infrastructure**
- **Data Massage**
- **Maintenance**

The short elevator pitch for each one is:

What type of content can we make?

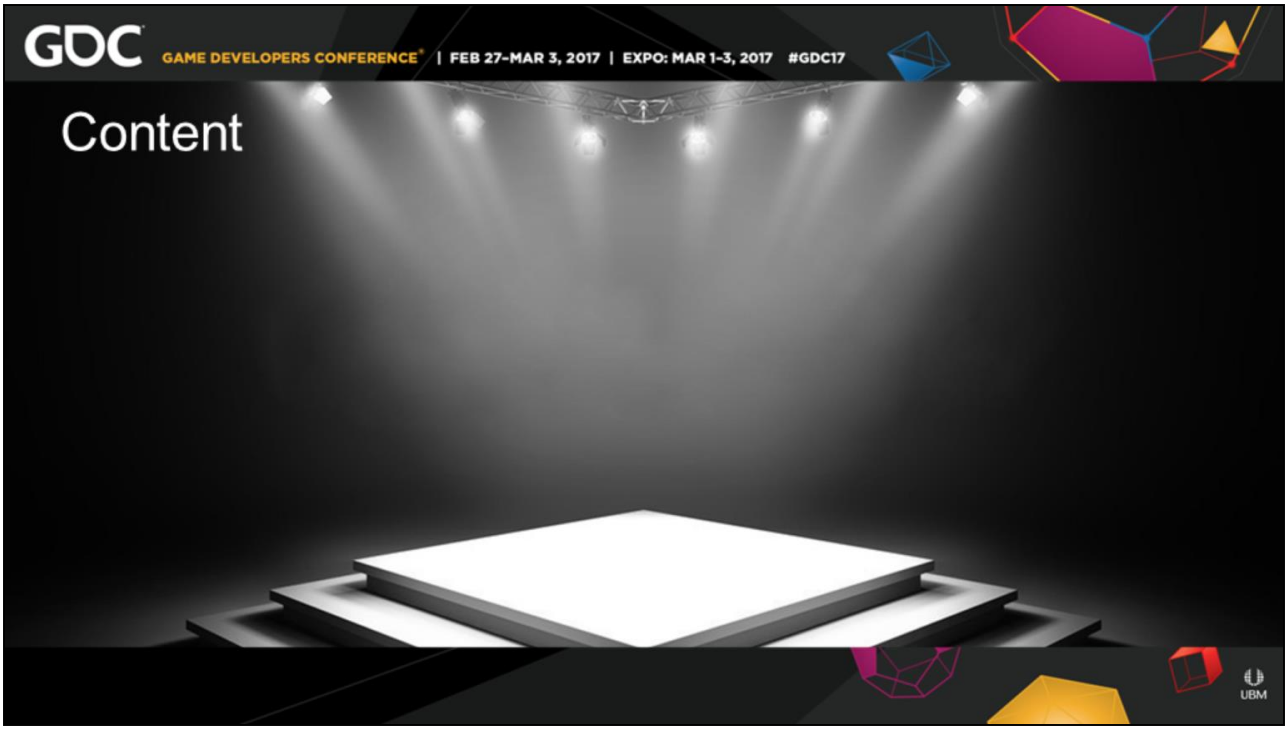
What is the scope of what we **CAN create**?

How to **prepare** for your simulation?
What type of simulations are you doing?

What are the **hardware and the software** needs to make this possible?
Your IT person will enjoy this.

Once your simulations are done. It’s not really over.
You need to give that **data some love**,
give it a massage, sculpt it into what you need it to be

Further once you bring these assets into this world, you need to make sure it survives in this world.
It’s a very cruel world.

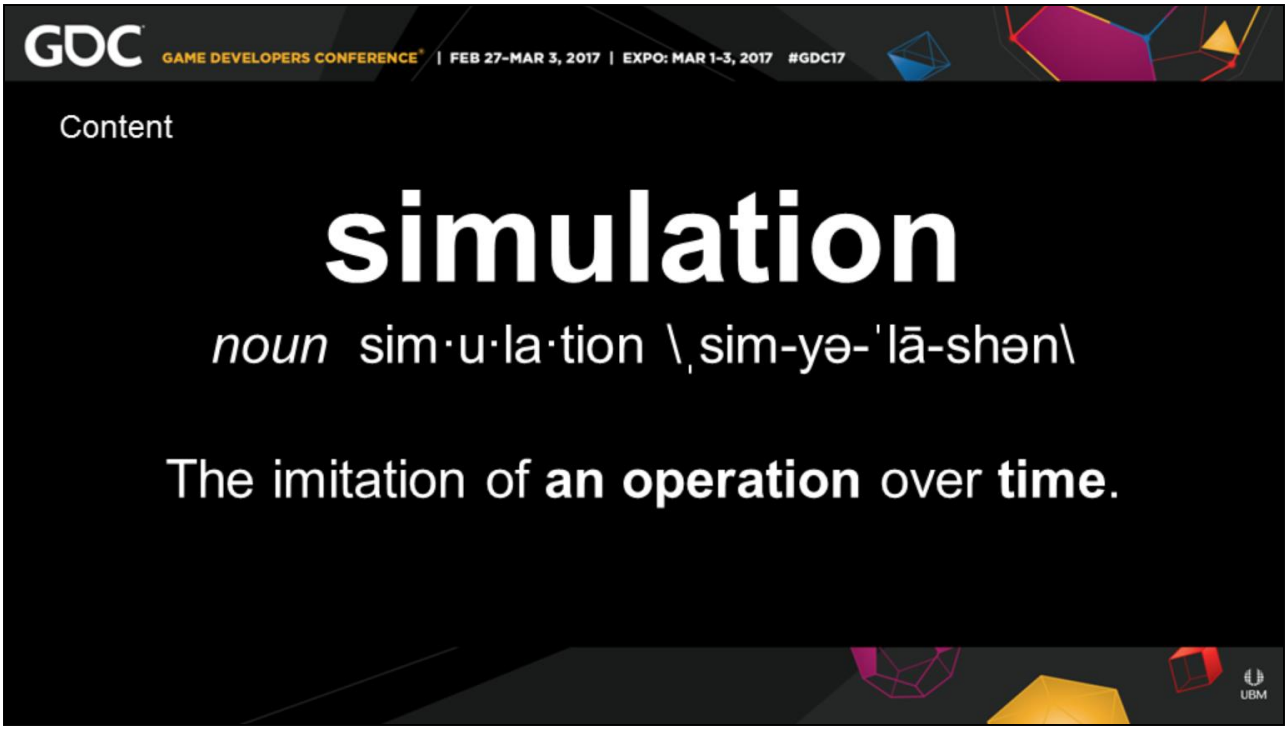


- 05:00, 00:20, 54:00

What is your **final asset type**?

This may seem like it should be blatantly obvious.
A texture pipeline builds a texture.
A modeling pipeline builds geometry.

Simulation, however, is a **method**
that can be used by Any pipeline
for Any asset type.
It’s a means to an ends



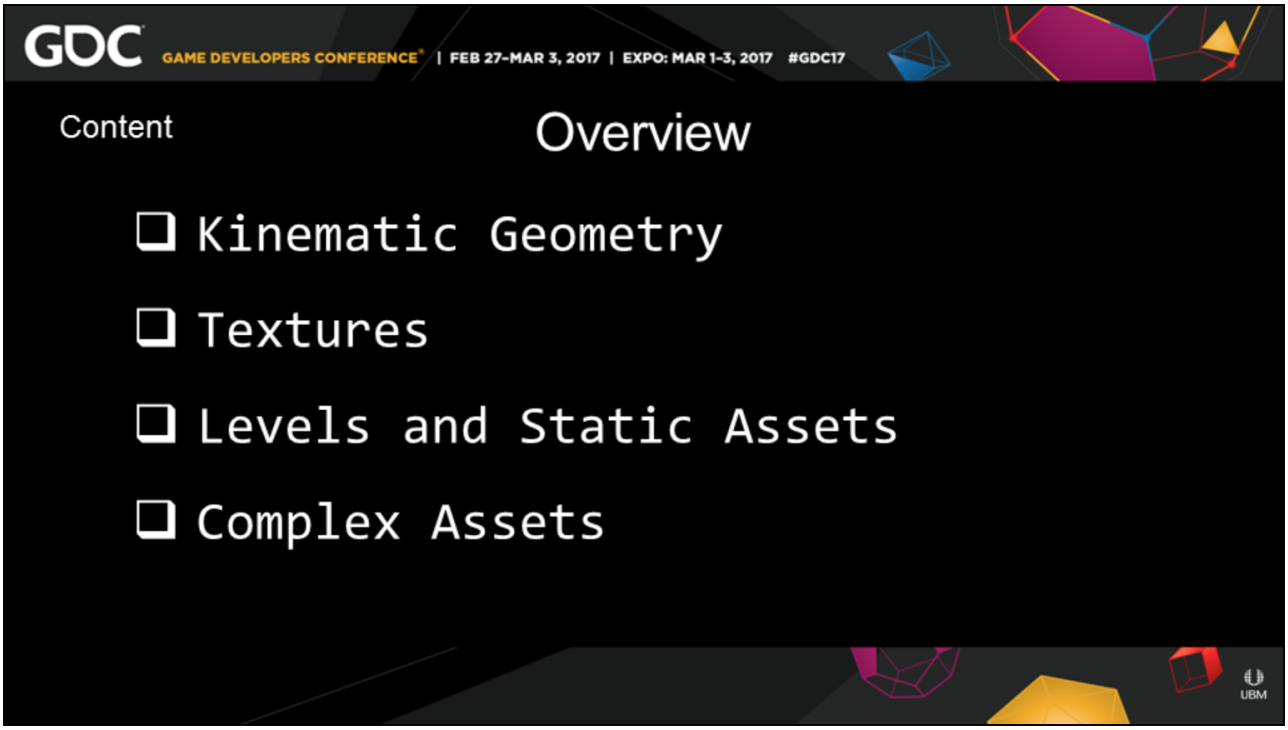
- 05:20, 00:20, 53:40

Apologies for the **stereo-typical slide**.
It is an important thing to see
And to consider.

Simulation are the imitiation of an operation over time.

Will first be talking about **operations**,
and we will follow up with the **TIME**
and I will stress the TIME part now,
write this down if you want.
We will circle back to it later.

Simulations are a method,
the output of which could be any content.

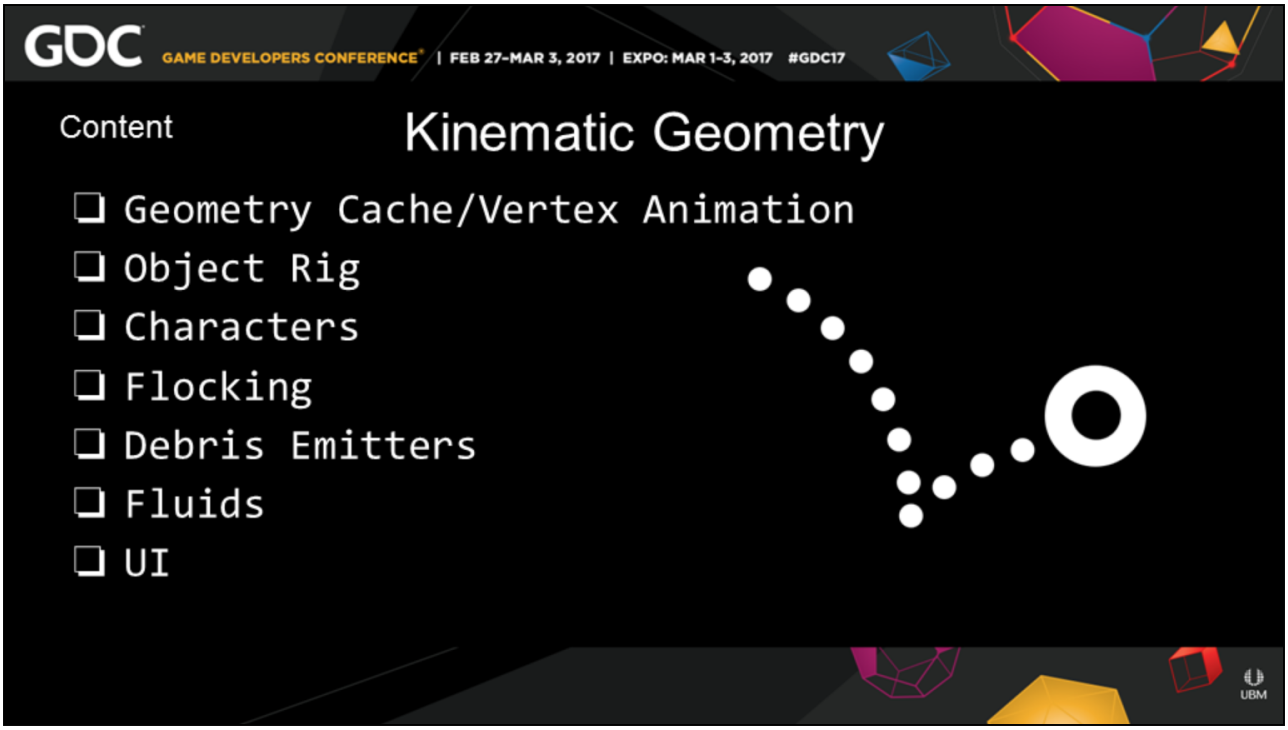


- 05:00, 00:20, 54:00

Some of the **categories** of content
that are directly applicable to simultion are:

Animated Geometry,
Textures,
Levels and **Static Assets**
similar to what
Luiz lead with
in the previous talk.

Will talk a bit about **Complex Asset**
for instance state machines and sequences



- 06:00, 01:00, 52:00

Animated Geometry can encompass a large swath of asset types.

The most common personally
for our Team on Halo 5 was **Geometry Caches**
We learned to push over a million transforms through the GPU at 60 FPS
It became our bread and butter, to squeeze any simulation through
like the E3 Headliner.

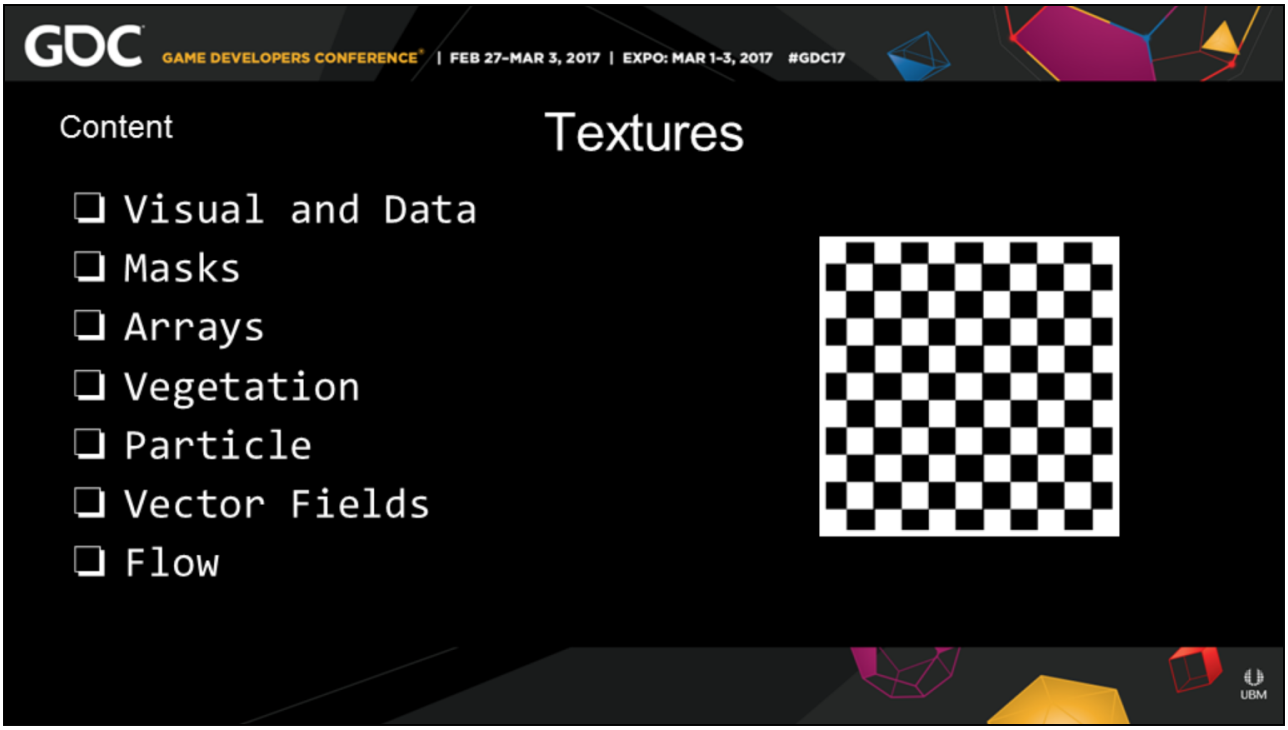
For object **Rigs**, think moving platforms and mechanical motion.

Characters using physics to define fat jiggling, and hair flopping,
stuff a character animator would have to do time consumingly by hand.

Flocking Algorithms for cyclical birds flying through the sky.
Fleets of Ships fighting.
Stadiums of people doing the wave.

Debris emitters, to fill the environment with extremely cheap detris.
Fluids such as Water Fountains, rivers, and lava falls
With UI, making Iron Man’s HUD can be immensely pleasing.

The list can go on endlessly as does the lexicon for moving geometry.



- 07:00, 00:40, 51:20

Textures can be simulated as either a direct visual map, or as a data field.

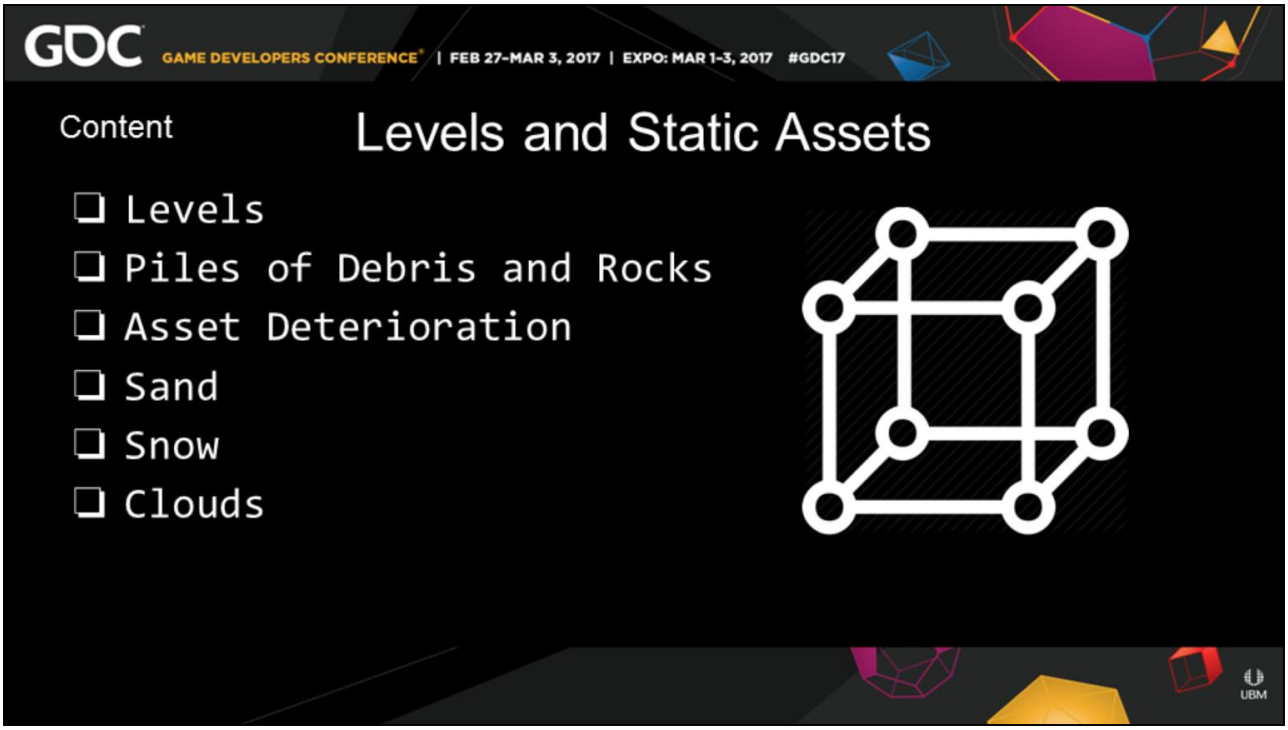
As far as visual maps, think of Substance with **particle painting**,
imagine applying that deterioration and wear directionally
and layered across a whole level.

Visual Textures can be responsible for rain, erosion, drips, scratches, dings and dents. To name a few.
Why have an **artist paint** them on thousands of

assets?
When you can simulate a whole levels worth of masks at the same time.

Fire Simulations
get rendered as a texture array.
You can make particle emitters,
or capture particle data in textures.

As data you can use these textures to
propagate vegetation of all sorts.
You can design **vector fields** for wind currents.
You can turn your river sims into flow maps.



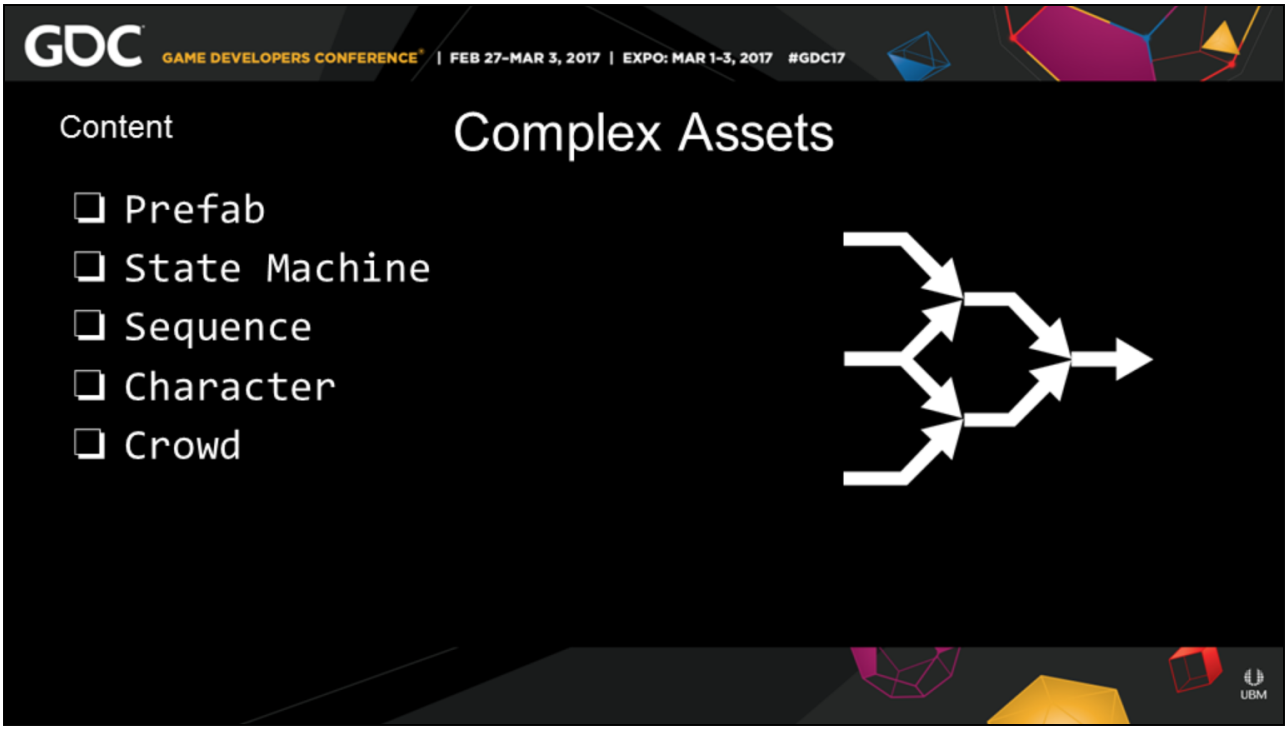
- 07:40, 00:30, 50:50

Luiz in the previous talk
Talked about this, so I'll be a bit on shorter.

Procedural level creation is an amazing tool.
But on top of just creation,
you can performs **sims on these**.
You can pile debris or rocks.

You can actually **erode entire levels** based on water flow,
or the sands of time.
Watch cities crumble and grow over the millennia.

You can **make it snow**,
so that trees actually bend with the weight of their loads.
You can create clouds
that you can light from day to night as they evolve and change.

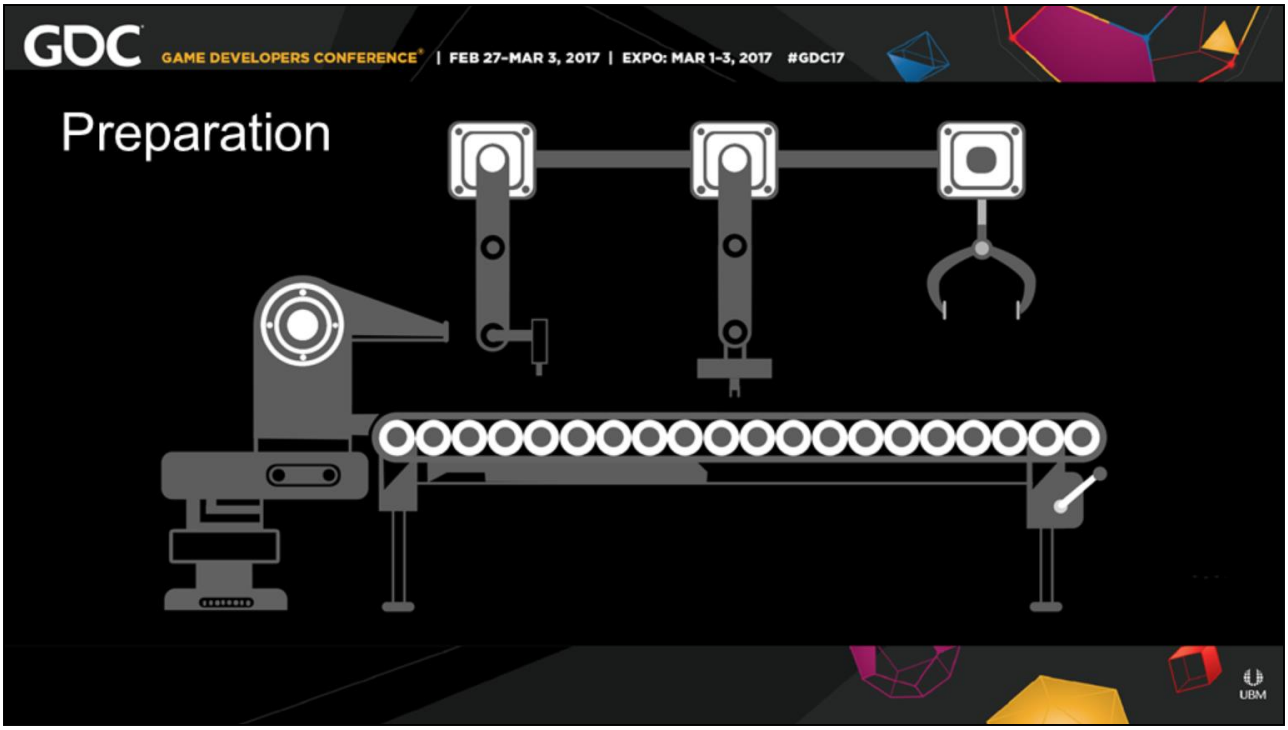


- 08:10, 00:30, 50:20

You can even use simulations to create **complex assets**.
For instance the perpetual door problem,
Cover, walls, floors, ceiling.
Even though these are offline sims,
you can make event based triggering mechanism,
that work on regions, with multiple permutations.
That are layered on top of particle and havok geometry.

The E3 headliner was a complex assembly of **trigger volumes, state machines, and sequences.**

You can even simulate
not only the character animations themselves,
but whole characters, with their rigs, in large
crowds.

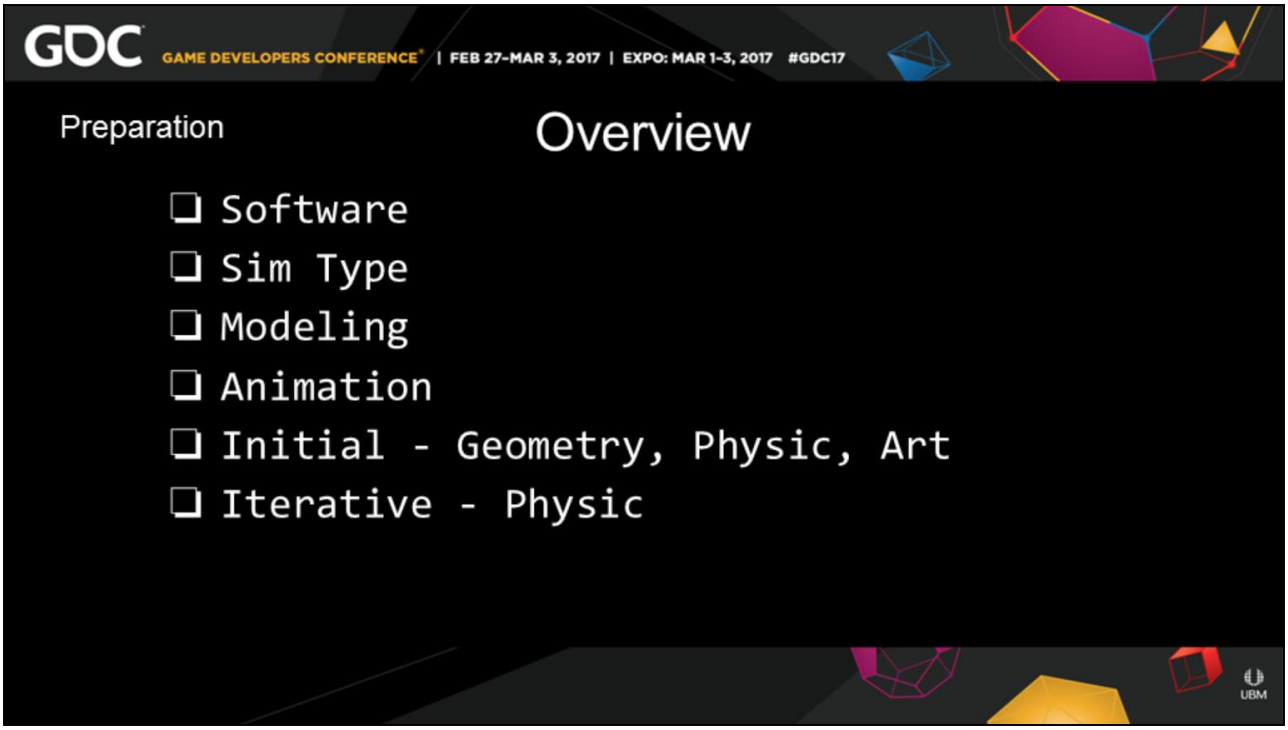


- 08:40, 00:20, 50:00

With a bunch of these content types floating through your heads
I want you to grab hold of one of these
And write it down on a piece of paper;
If you are taking notes

Something like:
Asset Deterioration,
Geometry Caches,
or Flocking

With this item in our heads,
will begin the next step to **prepare** if for simulation.



- 09:00, 00:30, 49:30

We will split up **preparation** into few **categories** to keep it digestible.
First off is the **software** you want to work with.

Then the **type of simulations** you want to run.

You need to do prepare your **models** and your **animation**.

Then we need to set up the sim, not necessarily the pure physic algorithms
But establishing **gravity, friction**, collision,
And basic **art direction** for your game

- 09:30, 00:30, 49:00

Our first decision is the **software and plugins** we want to use.

This can depend on the artist you have available to you,
Your financial constraints.
Your studio history
Also what you need to get out of it.

There are many good software and plugins.
We used **Houdini**
so you know the **bias** point
and the comparable lexicon so you can A and B

your options
However, the presentation has been abstracted
So that you could use any application

In order to help you pick the best applications
There are **6 main points** I want to bring up
These are aimed towards working with a team,
this makes it scalable to grow
and shrink to any size team.
From indie to corporate.

- 09:55, 00:25, 48:40

You need a software that can **simulate anything you need**,
and give you a range of options.

You'll often find people run themselves straight into a **black box** wall
with some software when they are asked to make certain types of art.

You also need to be able to post-process the simulation,
however you need to do that type of data manipulation.

You’re simulation will never be perfect the first time, and the more options you can have to edit them the better.

- 10:15, 00:20, 48:15

Iterative Level Design will kill you in the realm of simulations.
Imagine trying to do a flow map for a river,
and the art director each week is changing the rocks.
You need to be able to slurp up the level
and kick that sim off with a decent tolerance threshold
that you don't have to hand author the sim
every time the director moves a rock.

- 10:35, 00:20, 47:55

Say you make a real cool debris emitter.
I made one for my lead Chris Woods that was a full 3-D bullet solver,
but we need to share this with each other as we use it with every asset.

I can not just be copying and pasting from different scene files,
I need to use this as **proper tool** throughout all productions.

- 11:05, 00:30, 47:25

Once I make a tool we don't want to be making the same tools
for every other software under the sun.

Plus the next artist that walk into the studio
The are going to have a completely different background,
as that is how we get diversity and better problem solving into your studio
so you need to plan for this and allow for it

Your tools need to be **accessible across** all dcc's and engine's

If you don’t plan for this you will build yourself into a box,
and you are going to notice the walls close in on you very fast.

- 11:35, 00:25, 47:00

When you are making assets
Or those big moments in your game,
those unique elements,
the islands in the sea of your environment art,

You need to be able to **adapt your tool set** to be able to author that content.
So we need a data agnostic tool set
where the tools can be procedural
and swapped around.

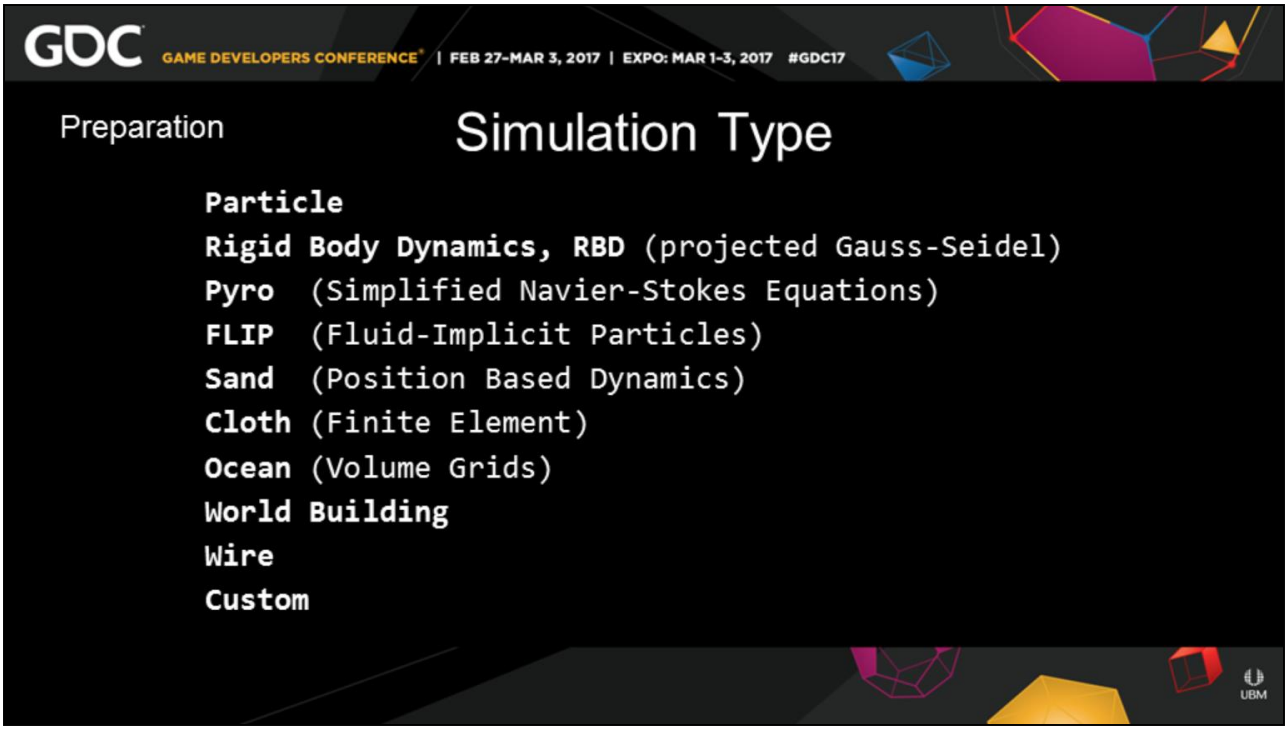
For your E3, your intro's, your end games,
those little unique blips in the game that connect

your player to your world

- 12:00, 00:20, 46:40

At the end of the day your pipeline team can only **support a limited number** of packages,
so if you have a half dozen programs that begins to become more maintenance than it is worth.

You need a single tool that can handle it all,
which is why I pick Houdini



- 12:20, 00:30, 46:10

Now that we know about some of your software needs.
We should have an idea on the **types of simulations** algorithms you may run.

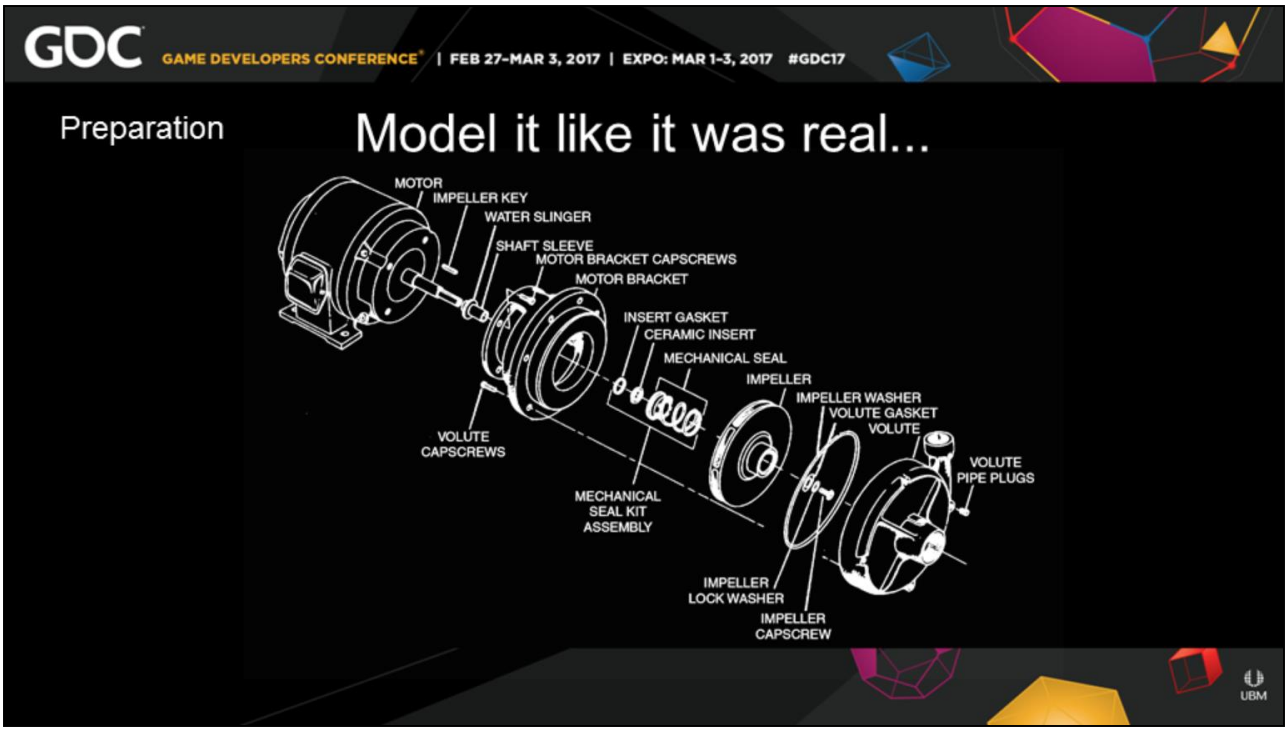
I've listed a few with their **houdini names**
and the main parts of the **algorithm** that they are based on,
so you can compare it with your own simulation options.

Each software package optimizes these algorithms **different**ly

So no two packages will produce the exact same results.

These simulation types can run the gamut from gas simulations for fire, smoke and dust to,
Finite Element for soft deformations and cloth.
And custom unique solvers that can help define the rules of your world building.

Pick a sim type and write it down, so that we can apply it to our content type.

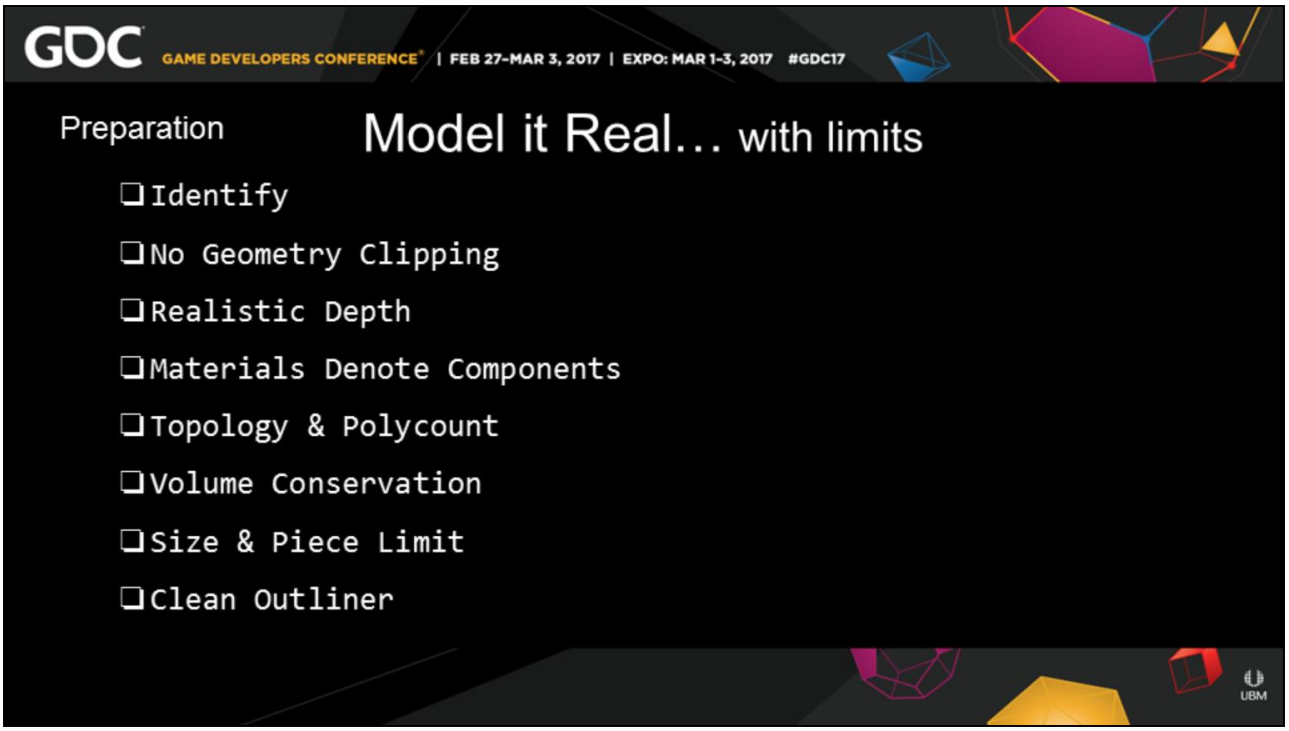


- 12:50, 00:25, 45:45

vis-à-vis the exploded diagram

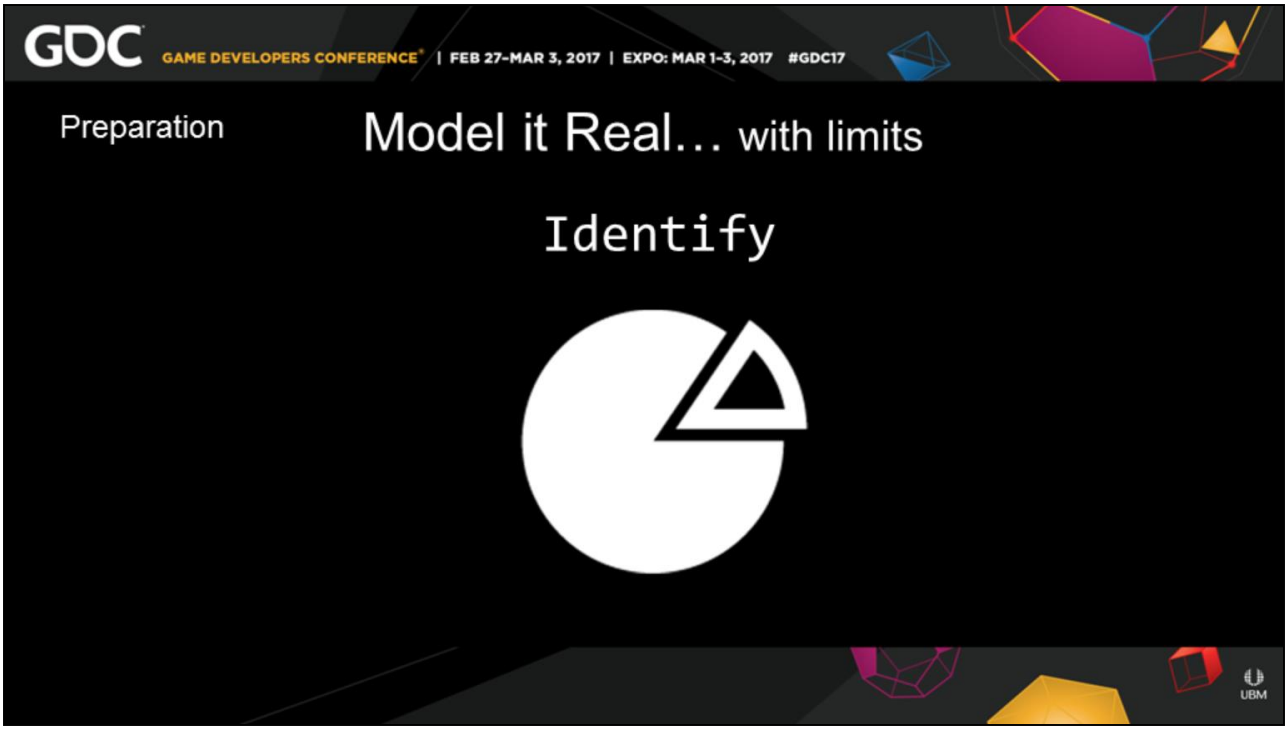
The common refrain is to build it **water tight**.
This only makes sense if it’s composed of one element.
A cement wall or an iron pipe.

However, if you build it like it was real,
when we go to shatter it.
It will break apart realistically.



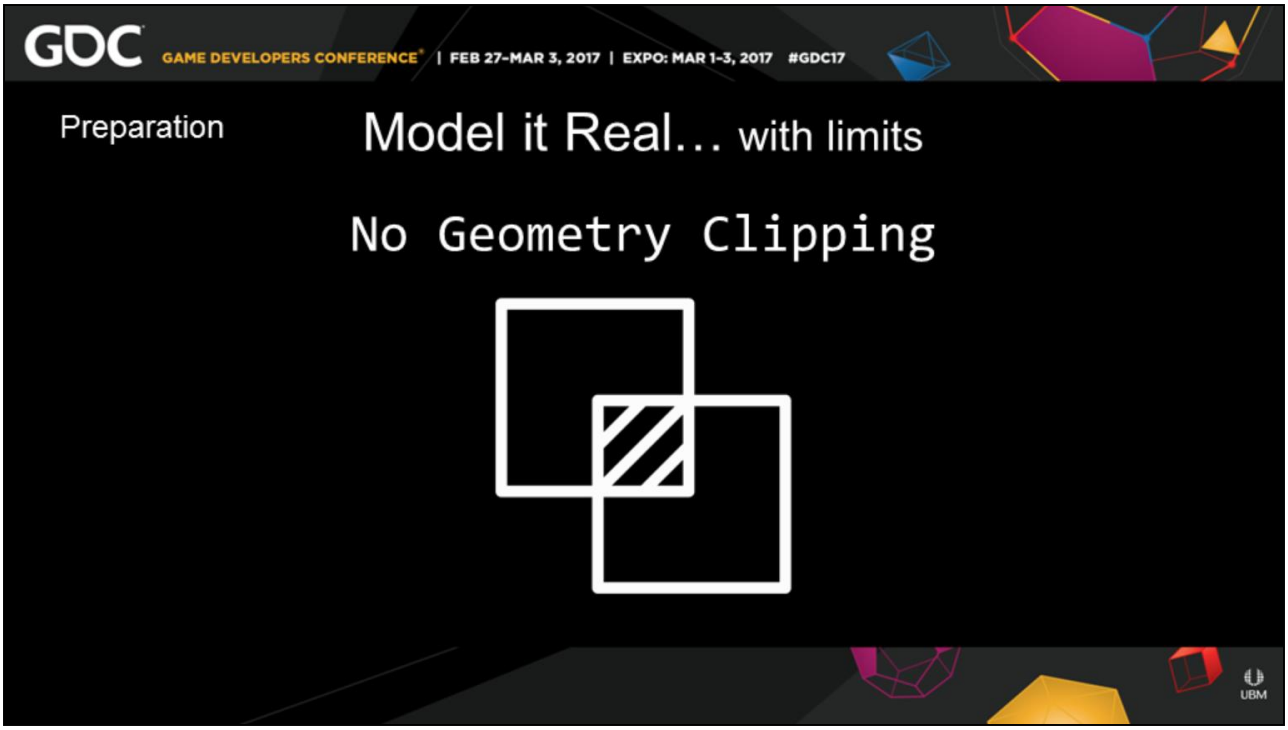
- 13:15, 01:35, 44:10 “2:00”

For those that raised a little red warning light about building it real.
Don’t worry, I have some **general guidelines** to make that production safe.
So take that asset you have written down,
And mentally apply these.



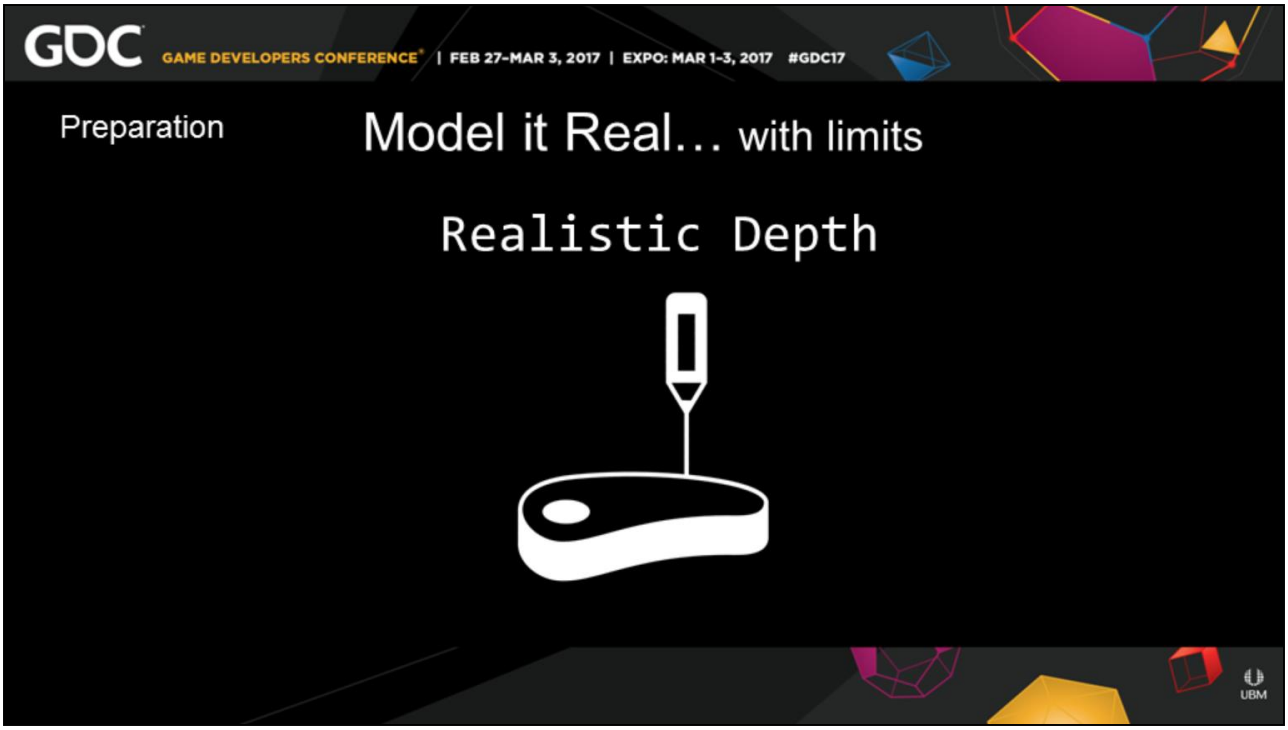
- 13:15, 01:35, 44:10

Identify what is **earmarked for simulation**,
This is general a specific asset,
or this may be a general area for a moment.



- 13:15, 01:35, 44:10

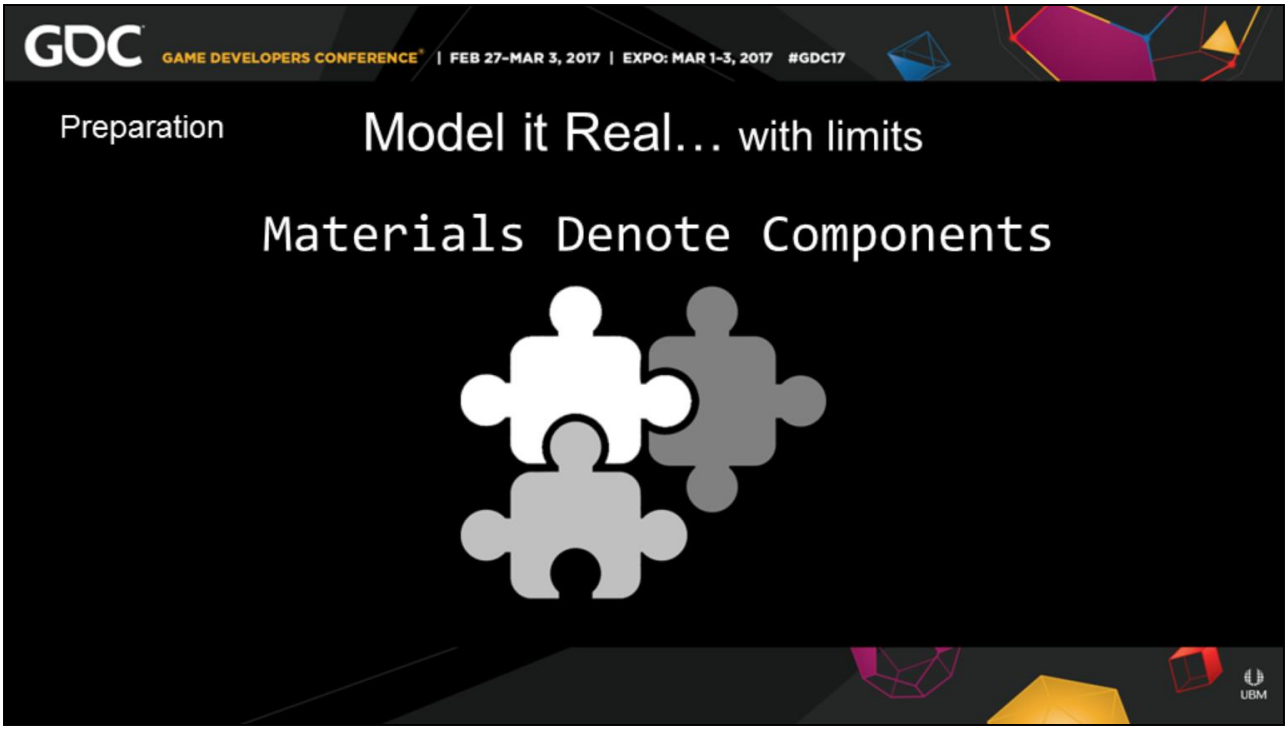
Use the snap to features so the geometry does not have **overlapping** edges.
These finicky edge overlaps are the most time consuming to post cleanup.



- 13:15, 01:35, 44:10

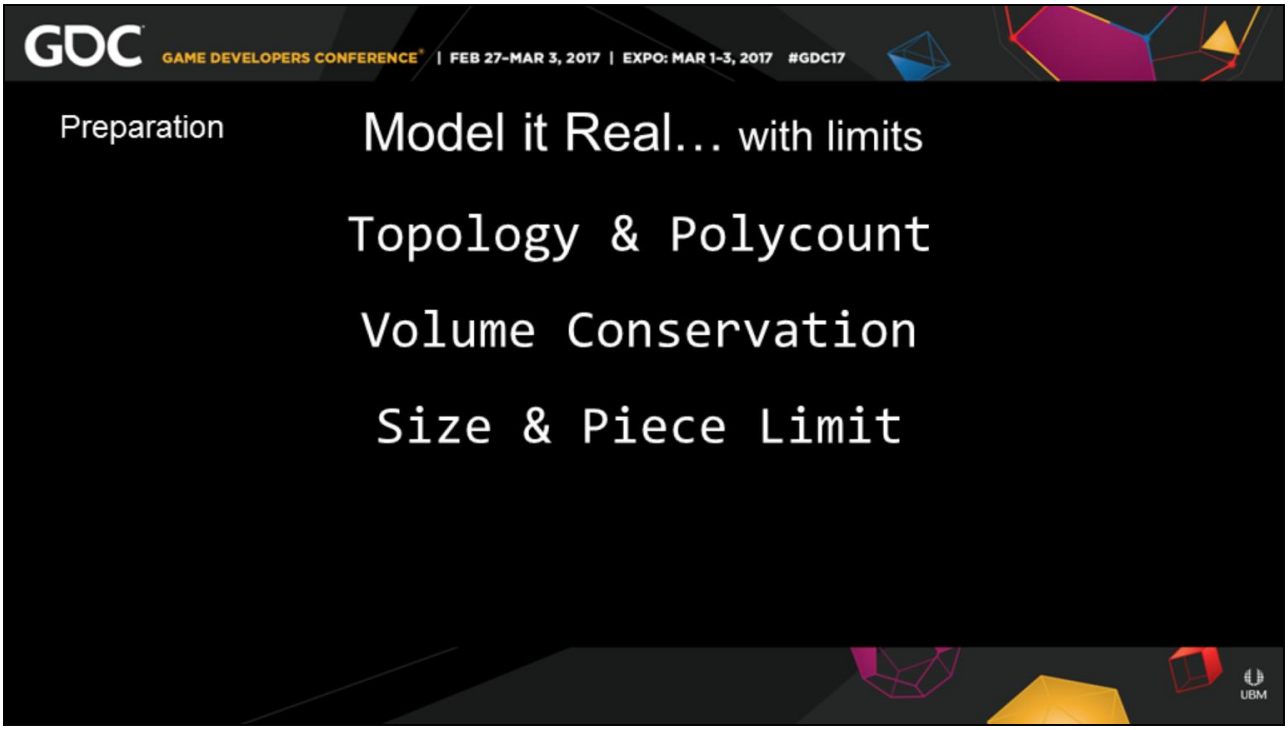
Give the outside shell a **realistic material depth**
The depth issue is dependent on material type.
Sheet metal may be safe with double sided material,
But heavy metal or wood paneling has an observable thickness,
That wraps the internal structure.
Think inside the walls of this very room.

Keep the outside shell in an attribute group.
The outside shell is your traditional game model and your beginning and end states.



- 13:15, 01:35, 44:10

Separate based on **material type**.
The separation on material type is the first extremely unique issue,
Half the time this denotes an actual geometric component like a door knob
Versus the main part of the door.
You can test this by exploding view based on material names,
“plus keep the material count down per an asset.”



- 13:15, 01:35, 44:10

These three are a triplet,
Topology & Polycount
Volume Conservation,
Size & Piece limit,

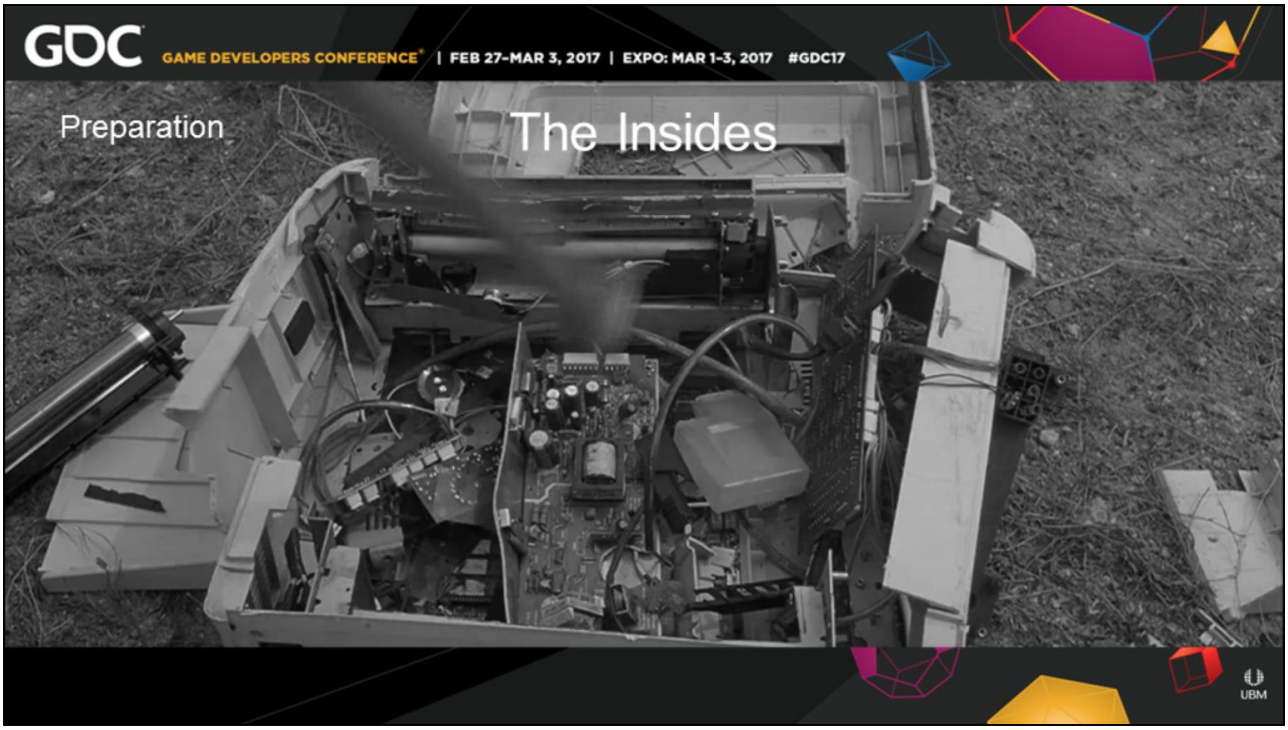
Obviously don’t build micro features.
For us it was .04 fusing in world space
For the vertices that do not matter.
Keep your poly count relative to world space,
This can be a simple diagnostic.

Build the primary key components that fit into the object.

Under a car hood there should be an engine block.
And then when you go to spit out the pieces,
Do not spit out arbitrarily more piece than can fit inside the vehicle.
The consumer knows,
This is why it’s comedic for unfeasible amount of gore to come from monster in
“B” rated horror films.
It’s comedic not realistic.
We are in a generation more sophisticated than cause in effect for “reality” style games.

- 13:15, 01:35, 44:10

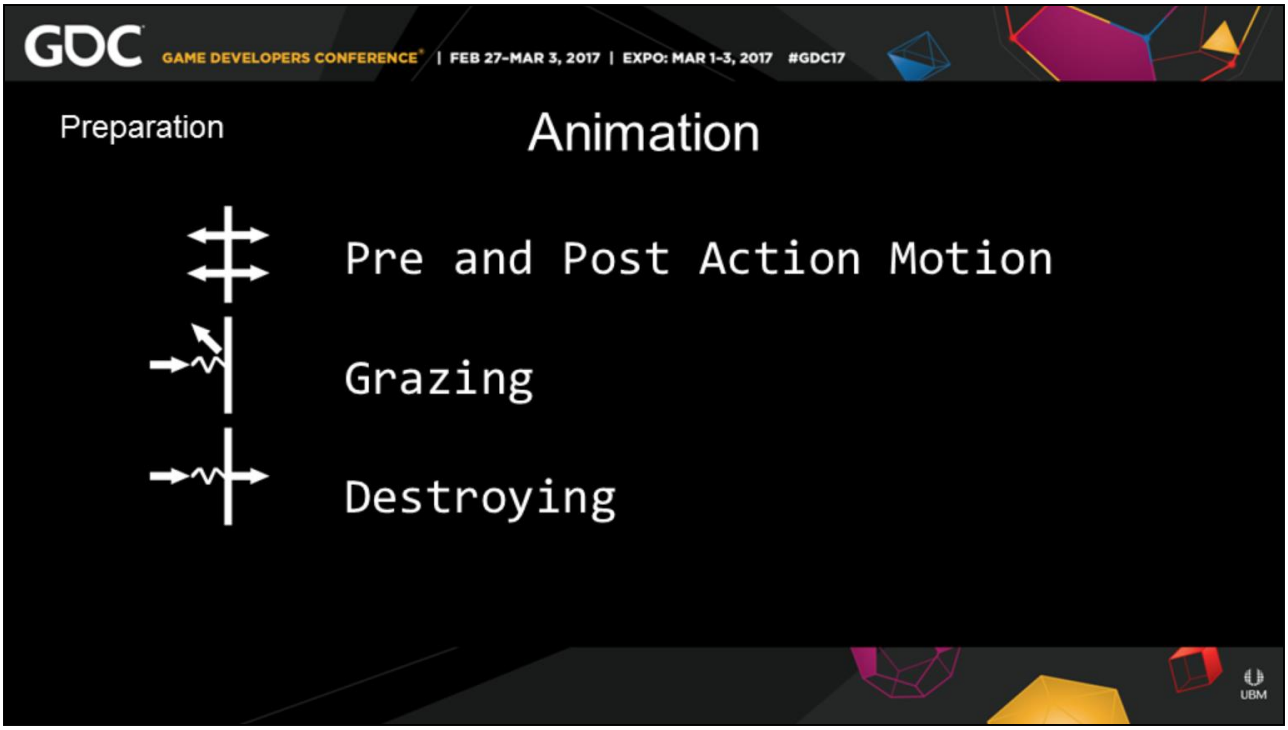
Lastly keep your **outliner** and hierarchy clean and well named.
Like point snapping and hygiene,
it is a clean and healthy habit
Also depending on your pipeline your simulations could be dependent on
the naming, materials, or attribute tags,
Just like a rig as will go into next.



- 15:50, 00:10, 43:00

Before we go deeper into simulation land
I want to leave you with an easy to reference reminder
if someone ask you why you need those pieces.

Turn on the printer smashing scene from **office space**
and tell them to pay attention to the printer.



- 16:00, 01:00, 42:00

For assets that are already animated,
There are three simple things I would be aware of

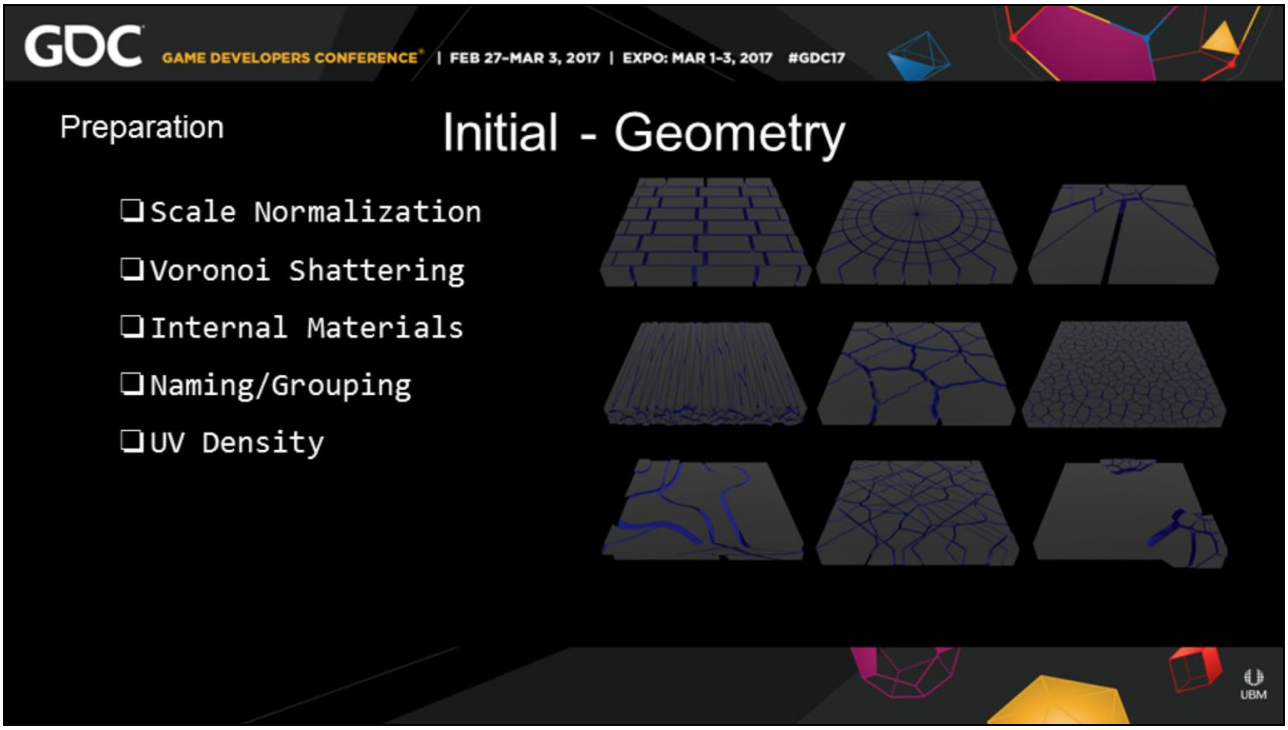
The first is **pre and post action motion**,
Especially if you are using physics for secondary animation on your character rig.
This means your motion must begin before
and carry after your required animation.

Remember that physic book analogy,
An object in motion stays in motion.
An object at rest stays at rest
So if you start your sim when it is static like a statue

Your sim will be static like a statue.
Even fire sims have a long pre-roll sometimes of hundreds of frames.

The next two are about **grazing** and destroying
If you are going to impact, step on, or interact with an object,
Animate the reaction and make sure you don't penetrate.
You can use your character sims if needed.
Simulations hate, Hate, Interpenetrating objects,
Two objects are not meant to exist in the same space.
The sim will explode, and not in a controlled way.

However, if you are **destroying** something,
You need to animate the anticipation, interaction and then penetration.
If your sim is secondary, it will not feedback into the character.



- 17:00, 01:15, 40:45

Now we begin the true **prep of the simulation**.
So remember the asset you had written down
Where going to becomes riggers of simulations now.

Imagine how big that asset is in “our human world” space.
We want to **normalize the asset** to this space.
Like the concept of PBR shading and rendering.
Physic work real well based in real world units.
Think of gravity, You can break it after you have it correct. Not before.

Often there are many ways to weather or **shatter an**

object.
This is where you should look up reference, and lock on your reference.
If you are breaking marble, rebar lined concrete, cinder blocks, wood, metal facade
how the item will break is highly important.

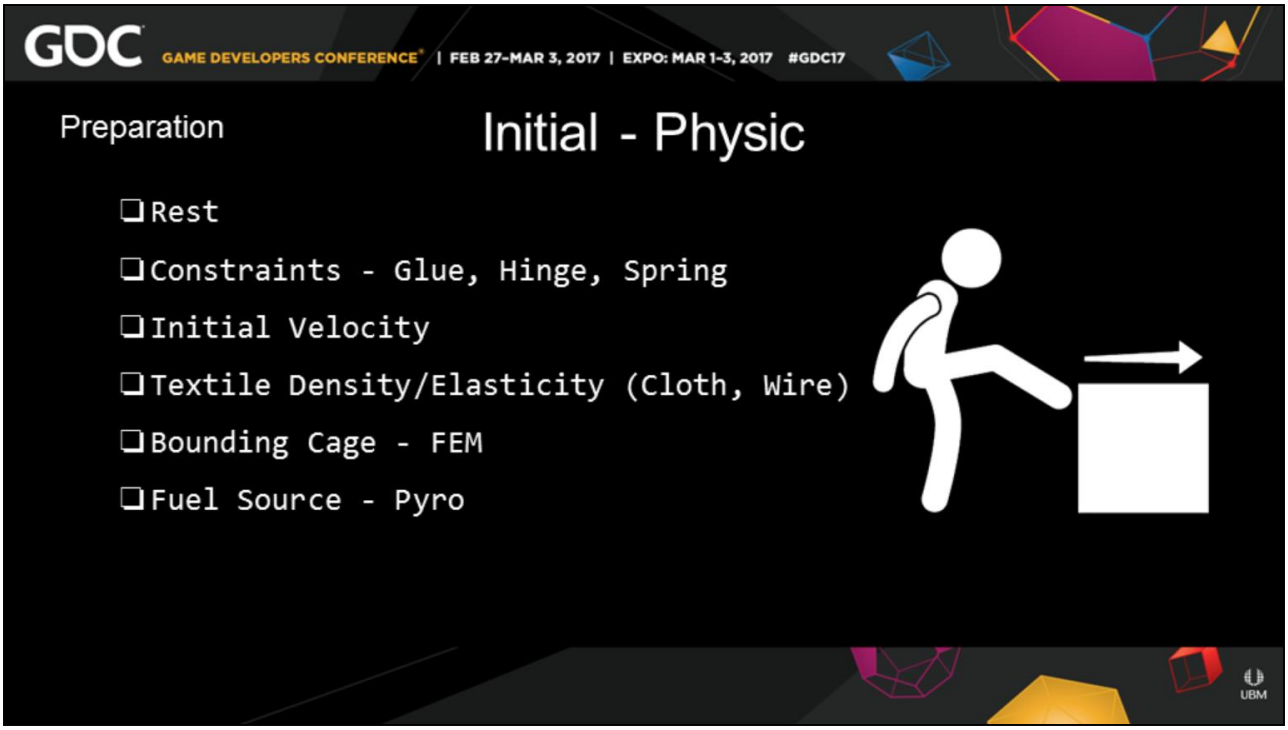
For games especially it is important to **pre-break your assets**,
this way you have maximum control over your part counts,
And it is far more expensive at run time.

You also need full control of your **newly created surfaces**.
this means you need materials, uvs, engine side properties, and normals to name a few.
If you have a good material naming convention, you can actually automate this.
Concrete to ConcreteInside for instance.

You need to be able to **name/group** these pieces, so when you feed them into a simulation
you can identify, to the physics program, this is part A. This is part B.

UV density is actually a funny thing to bring up as the more surface you create

The smaller the pixel density becomes in the normalized space.
So need to tile this like your environment ground.
You can automate this by normalizing it based on your world size.



- 18:15, 01:20, 39:25

Now that your geometry is preped, you need to apply the **initial state of physics**.
This is like setting up a character rig, For some sim types this can be simple,
But for Rigid Body simulations, this can be quite complex.
Will use a building such as our Sangheli towers.

First set the T pose of your sim. We call this the **rest** position,
It’s the point position when everything is standing up

clean and proper,
We use this for shading, and transform extraction.

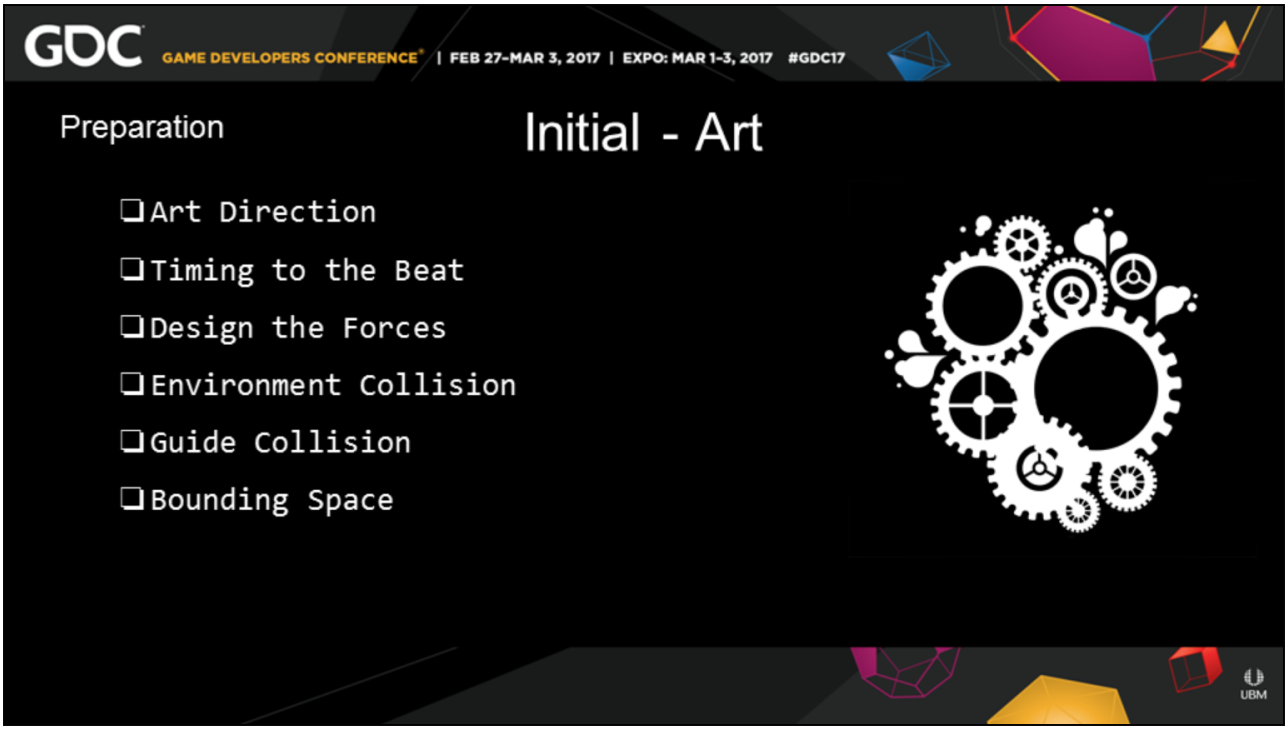
Constraints are the FK/IK and bones of your sim,
Imagine a character with several thousand pieces.
Constraints connect all of those pieces together
So when you apply force on one end,
Everythign flexies, hinges, breaks and accordingly and timely.

Thanks to newtonian physics, that darn physic book…
an object at rest tends to stay at rest.
So those piece you toss pass your player are statues,
And will fall flat with gravity, ::Sound effect… shh splat::
until you append the appropriate **velocity**,

If you are handling **cloth and wires**
the elasticity can direct flexibility by uvs, color, etcetera.
Or in general define why your tshirt is different than a cable

To get your girders to bend, depending on your **Finite Element Sytem**
you may need to create your own bounding cage.

For the torches along your precipist you need to create realistic **fuel sources**.
Whether it acts like the surface area of a log, or like gasoline spreading on the ground.



- 19:35, 01:20, 38:05

Our sim is ready for **art direction**. This is when you are the composer. The true artist,
Think of **V for Vendetta** and the blowing up of parliament to the 1812 overture
This can be you directing where your terrain will erode and avalanche
Or the maticulus grooming of your hair sim

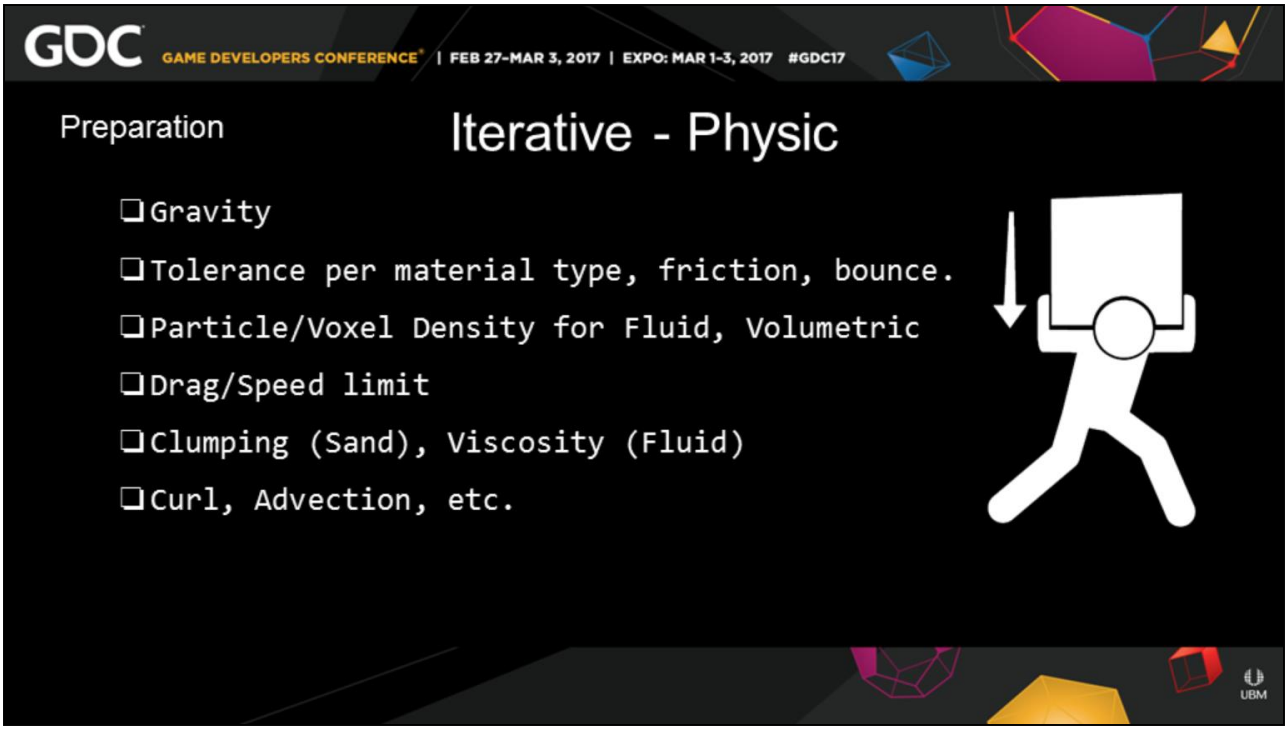
For **sequential assets**, like the building,
this is often the directing, the release of forces, to the beat of the sequence.
This can be eroding your contraints, as the ground parts to consume your city

You need the **related geometry alongside** your sim, to be integrated for interactions.
For iterative level design we just threw a grid over the mesh and **projected** on to it
As the level changes the ground would keep on updating

You may need **guide geometry**, whether moving or static to help your sim along
I'm very fond of the pokey stick method. In order to get my simulation working sometimes,
i will carve a hole in the back of my environment and literally hit it with a stick.
Impulse forces are great and all, but sometimes, you just want to hit it with a stick.

For any collision geometry, make sure the mesh, or volume, that is created is correct
If you have a **concave surface** make sure the default is not convex, the geometry will not be happy with you.
It will actually explode out, not good, happy accidents aside.
This was a big issue with the bullet solver when it first came out.

Also sometimes it best to build **a bounding cage** on your sim, A piece can literally go interstellar,
The solver doesn’t care you screwed up, So for really complex sim, I enclose them in a box
Keeping all my eggs in the same basket.



- 20:55, 01:05, 37:00

Finally this is where we work with the core of the actual simulation algorithm.

This is the part of the sim that **loops over time** each frame.
This is where **gravity** is defined in your sim,
Make sure your level is **normalized** to the human world,
Otherwise your gravity will be off,
You should try and avoid art directing gravity,
As we **perceive** gravity in a hieghten sense,
So what we think the rate should be is often not,

This also includes **per object tolerance**, such as, density, friction, bounce.
Plus you need to manage particle and voxel **density**. Start small, scale up.
Work on the general motions first,
So that higher order details can be revealed with higher density
Just like animation. You don't want to be worried about secondary animation
if you don't have your blocking correct.

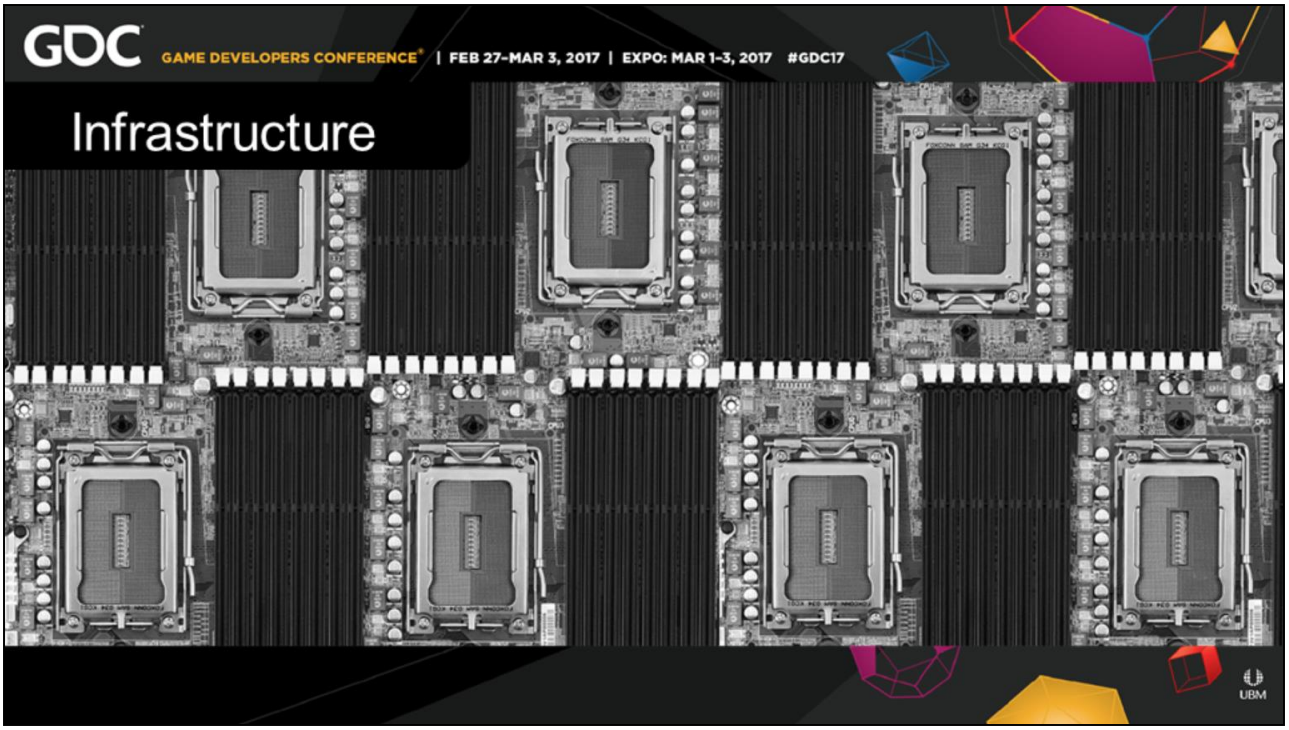
You need to impose natural resistance on your motion
Air and water resist movement via drag or **speed limits**.

Clumping and Viscosity, are a type of **stickiness** quotient,
getting this right really makes the difference between stuff accumulating like snow and sand or water and lava.

As your simulations perculate **forces**, like wind, curl noises, and advections
will affect how particles and volumes move through a space.
The difference from cigarette smoke to pyroclastic smoke is only a few settings.

- 22:00, 00:10, 36:50

Now you can press your **simulate button**.
Your little assets is on their journey to be molded
By that custom sim you have created.



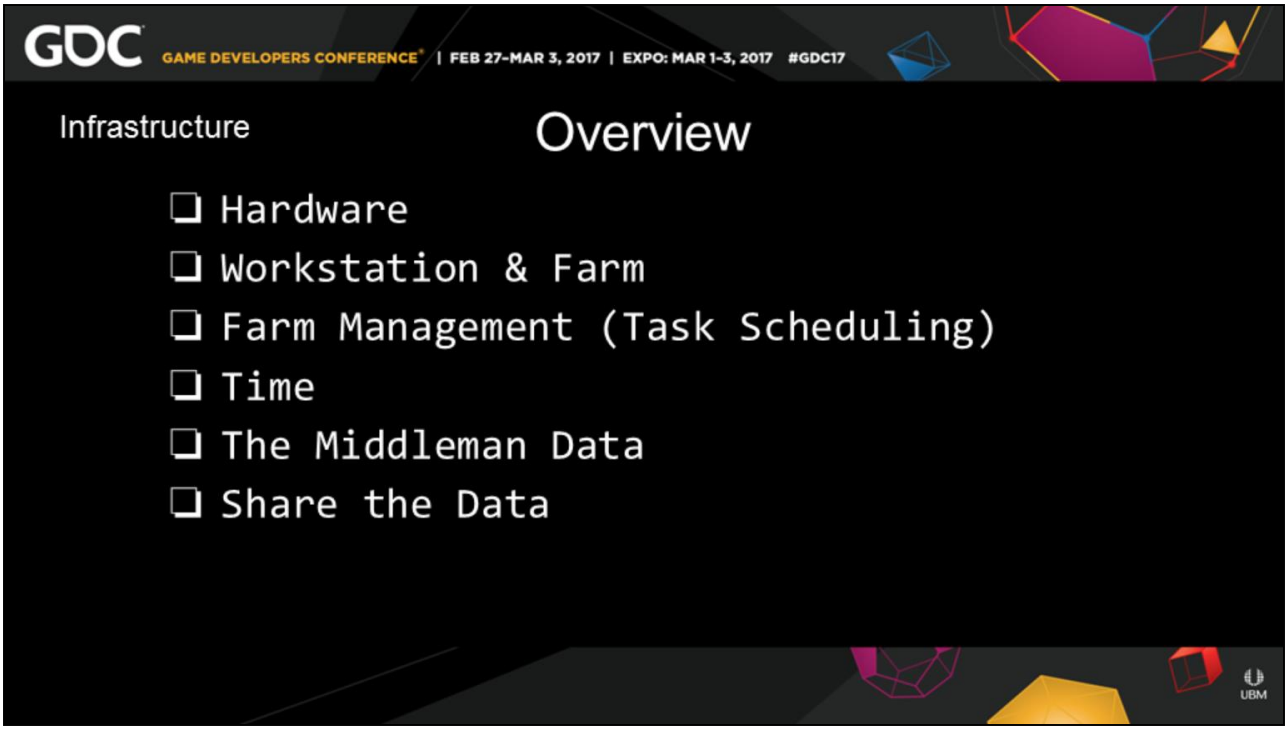
- 22:10, 00:25, 36:25

As your simulation run let's talk about your **Infrastructure** to support your simulation.

Not all simulations are designed equal.
Each algorithm and software package
handles a **simulation slightly different**.

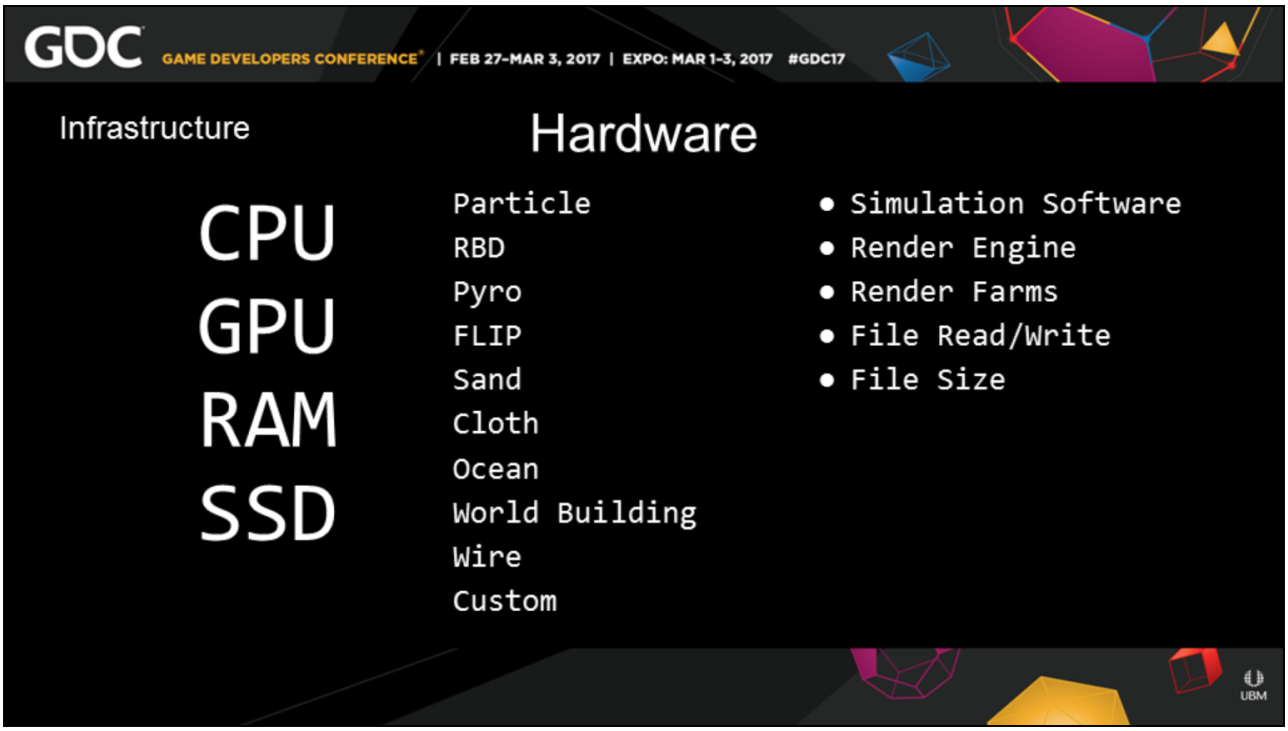
They have different needs and wants like real people.
I'm not saying you need to **commune** with your boxes,
Please go ahead if it will help you,
but it's important to know if the
Algorithms are CPU, GPU, Memory, or disk dependent.

-In games we live in a world of microseconds,
-in simulations we live in a world of microseconds too.



- 22:35, 00:10, 36:15

We're going to hit up your basic **hardware** options,
Percolate about the requirements of your **workstations and Farms**
Discus this pesky thing called **Time**,
And chat about the **data** we are creating.



- 22:45, 00:55, 35:25

Your hardware per each box or node, **The CPU, GPU, RAM, and SSD** are going to be taxed more than any other form of 3-D content creation. Usually people simulating have special computers to account for this, and the needs are usually quite different than the needs of a modeler.

If you are rendering a **texture array**,
most “offline” renderers are designed for the logical cores of a CPU, the more the merrier,
in commercials and films we often deal 24, 32, or 64 core boxes at our desk.

However some parts of algorithms are **single threaded**,
Like Rigid Bodies depending on the implementation,
less so each day, but high clock cores won’t hurt.

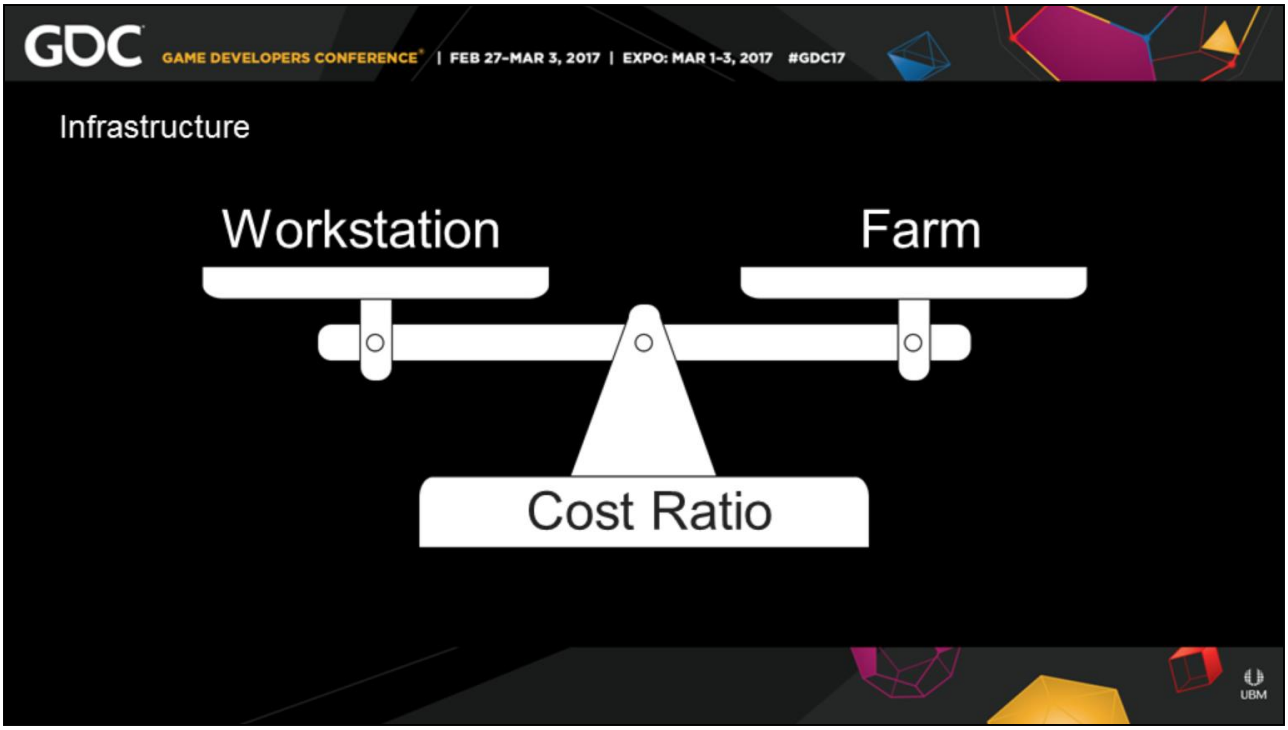
If you are running **particles, volumetrics**, or flip fluids,
these algorithms, inherently loves to be distributed through the GPU.

Other simulations like to store a lot in **RAM**
imagine several million particles, you hold it in easy to access memory

So each frame you can continue their journey uninterrupted

Some simulations are so fast that the **read write** to your disk will actually be largest impact.

I was going to graph this, but it’s got ugly real fast, and for each possibly setup, my page had white lines all over it quickly so this is very dependent.



- 23:40, 01:05, 34:15

The goal of your artist **workstations** is to efficiently use a person time,
and to do **less web** browsing.

The hardware to process your sim can be split into two categories,
a workstation, an artist dedicated computer,
And **a farm**, a series of machines used to distribute processing.
These do not need to be mutually exclusive.
Leaving the hardware in a server room or in the cloud is a great option.

Cost Ratios are what differentiate the hardware for each of these.
It also the major difference between indies and major studios.

Your goal between your workstation and your farm is to reduce the time your artist are **idle**,
Usually the shorter sims are the biggest culprit, It's easier to get distracted and loose peoples attention.
An extra set of cores or a better GPU for a few hundreds dollars, For your artist workstations
Can save you tens of thousands over the course of a year from distraction and time wasted.

Longer sims are definitely a burden in their own right. These should never really go longer than "overnight"
We are in games after all, and trying to make a profit
Plus the quality to cost ratio is never in your favor for longer sims

Also be aware that traditionally there are no GPU on

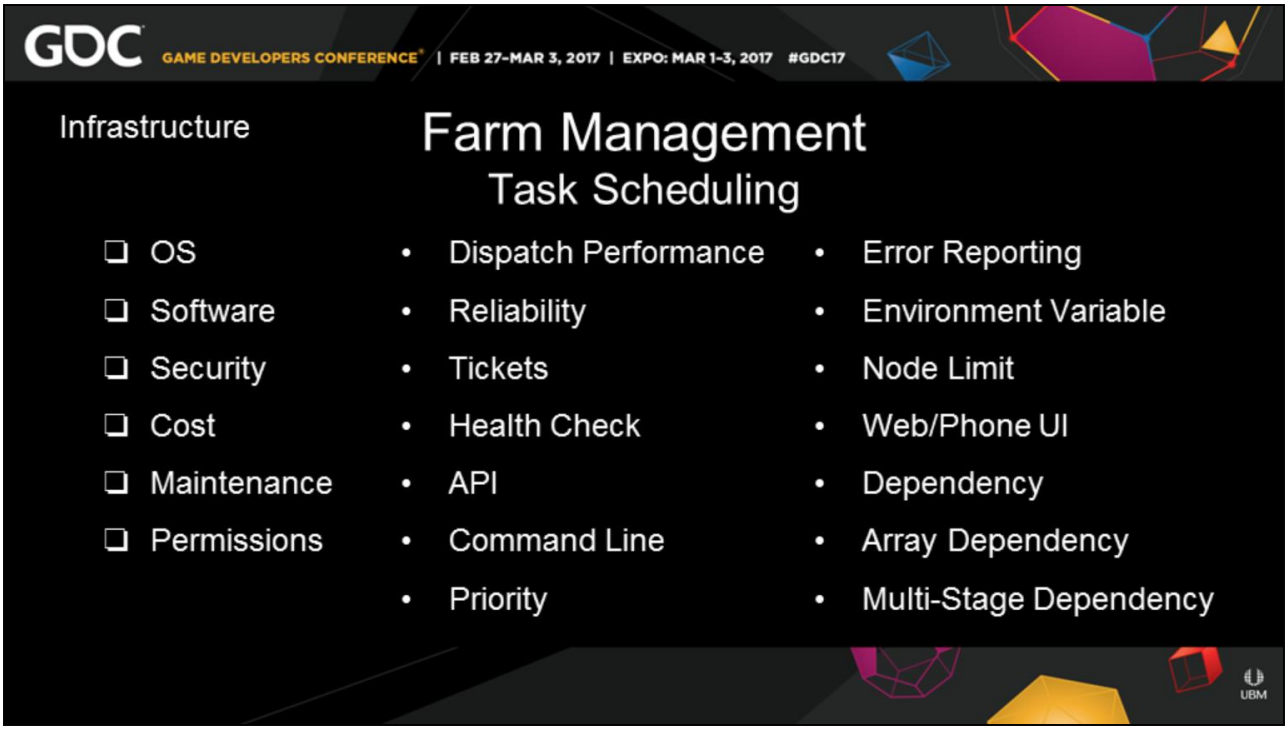
a server farm,
GPU have generally been associated with display units, and there are no monitors in a server rack
This is changing, but be aware of it for your simulation needs.

- 24:45, 00:25, 33:50

Your Farm…
This could easily be a talk in and of itself.
These are a series of questions to ask yourself
You can very easily spend a lot of money here.
And either see the return, or just throw away your money.

To start off from categorically.

Do you have **money to pay** for a farm?
Do you have the time for **open source**?
Does your **DCC** have a native farm, or are you using multiple DCC?
Is the **cloud** an option?
What would be the cost of sending your stuff to the cloud,
to process it and get it back.



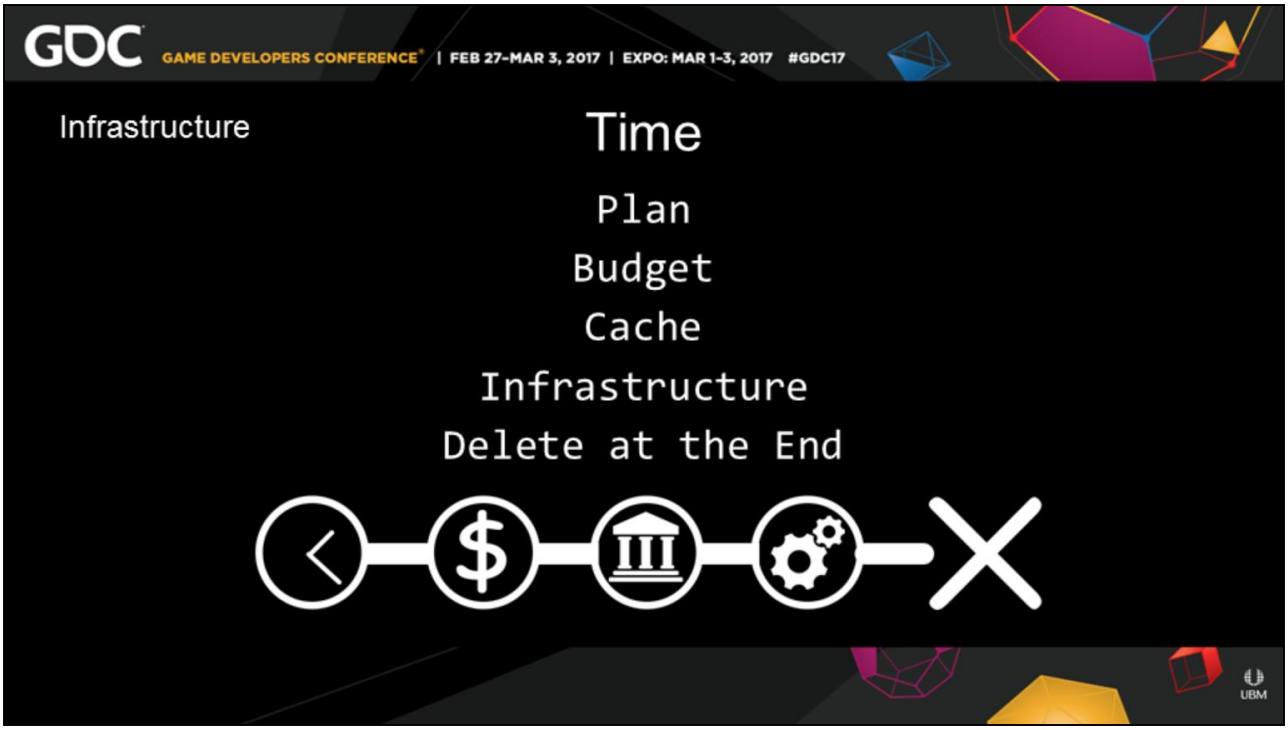
- 25:10, 00:25, 33:25

The follow up **questions** are onto the more **technical** nature.

What is your **operating system**?
How **secure** does your data need to be?
Are you having your friend do it at his house?
What software, simulation, and **hardware** do you need **on the farm**?
Whose going to **maintain** this farm?
You'll need hardware, IT, and **wranglers skill sets** or people depending on your scale.
In the end you do not want to give some one a **free bit coin farm**.
I have listed a few other bullets to ask about, too.
As these are some of the hundreds of questions to ask.

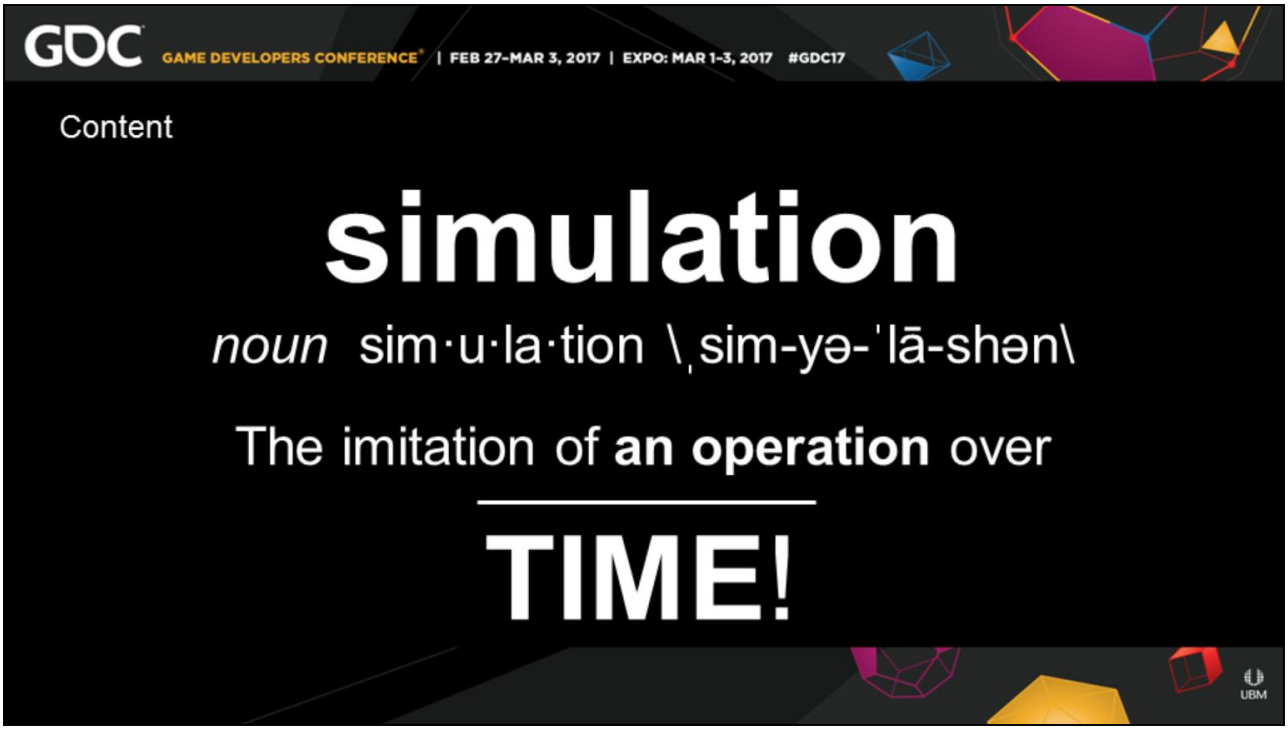
- 25:35, 00:10, 33:15

As there are **hundreds of farms** out there you can use nowadays,
And they all service a very diverse set of options
So spend your money wisely.



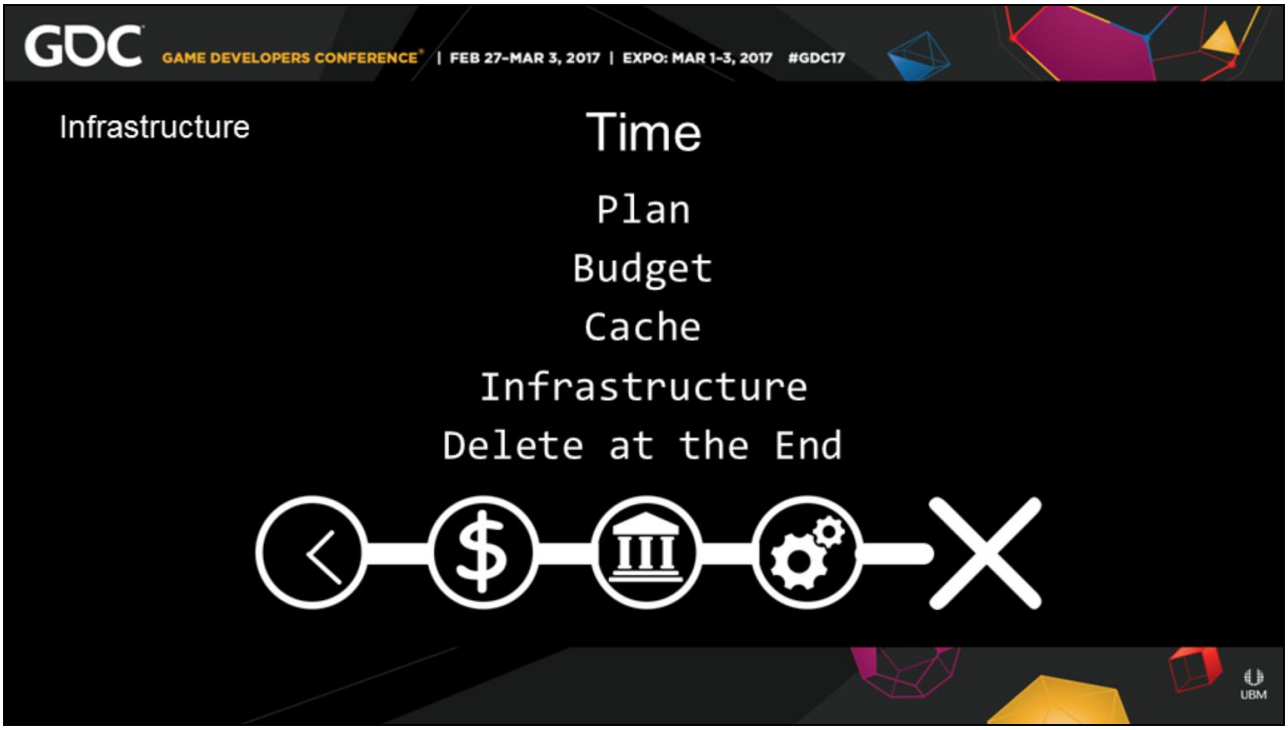
- 25:35, 00:10, 33:15

Time is the key component of your simulation at this point.
By **Simulations very definition**,



- 25:40, 00:10, 33:05

Do you remember that **stereotypical slide** I had before?
Simulation is the imitation of an Operation over **TIME!**



- 26:00, 00:25, 32:35

This is what no one **plans** for or **budgets** for.
In games we work in real-time, even content creation is WYSIWYG

As this time
….stretches on…
We don't want to relive it,
So we **write out the data per each frame**
This creates intermediate data that can become huge
This could be more than the rest of your combined studio output.
Simulations are Big Data the Art edition.

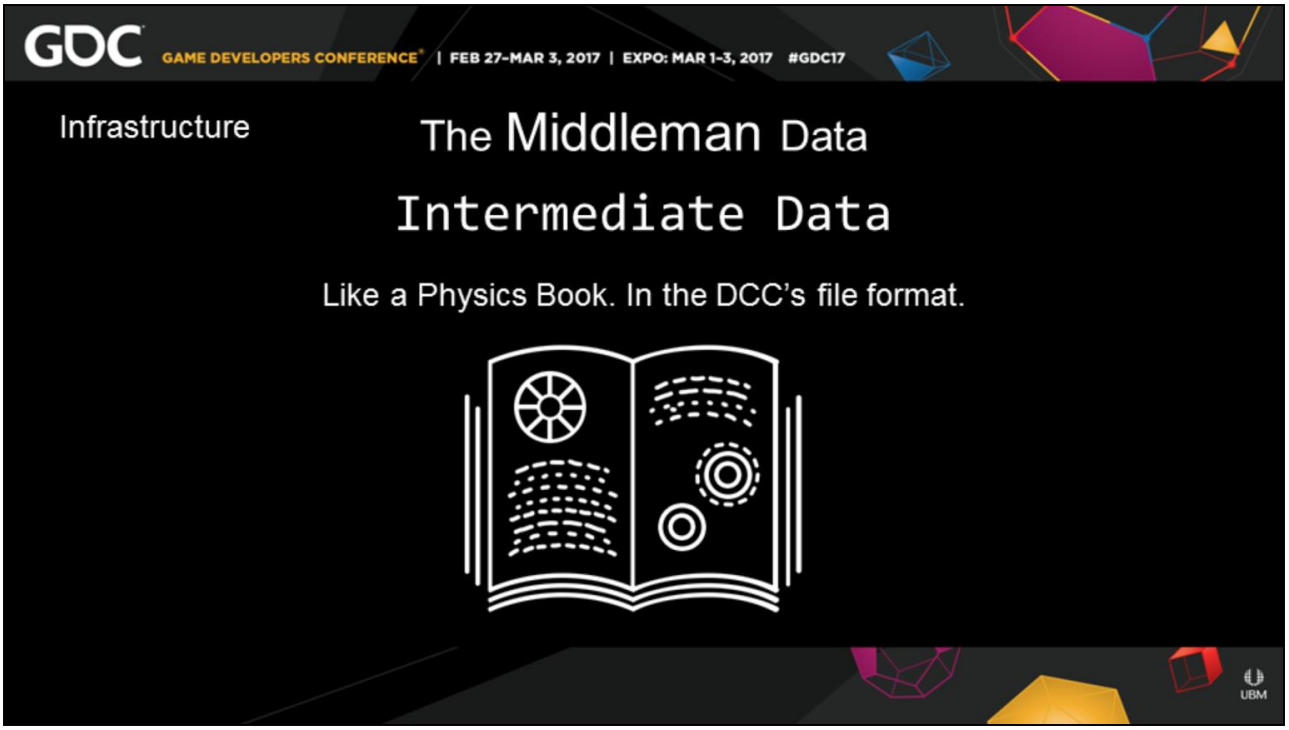
We're talking about gigabytes and terabytes of data.
Which if you are moderate size studio **requires infrastructure**.

But do not worry for the long term, this is **temporary data**. You can delete it at the end of the Game
As long as you make sure to check in all the source, and scene files.

- 26:25, 01:15, 31:20

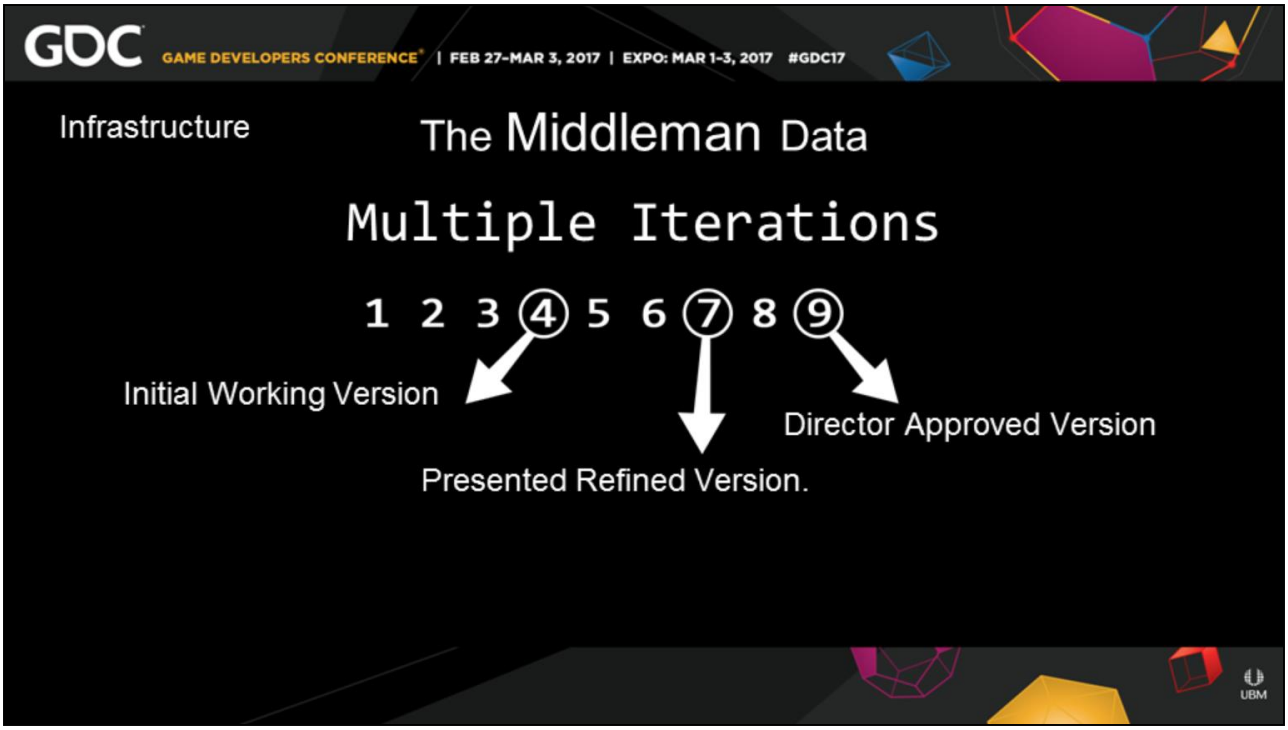
Let's talk about this **middleman data** we are storing.

It's actually a very **complex story**.



- 26:25, 01:15, 31:20

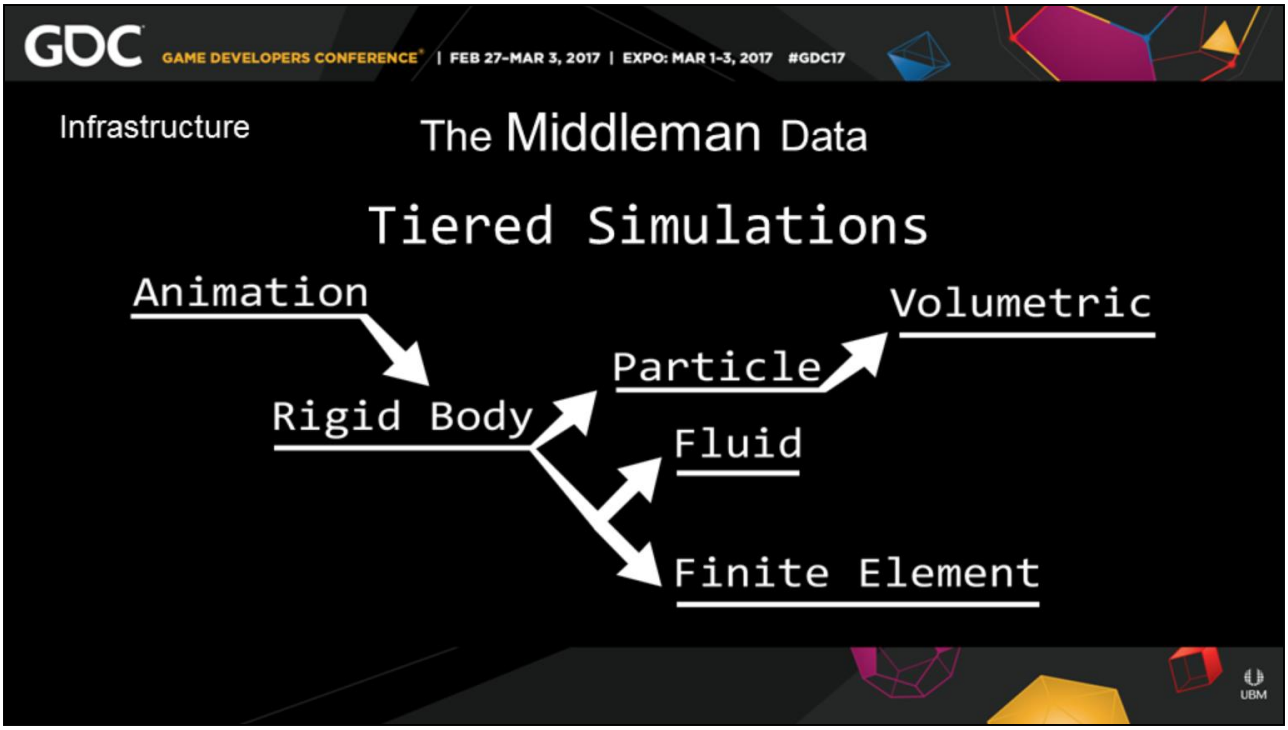
This **data is intermediate**,
you can think of it as the equivalent of your highschool physics book
It's a unique book for each content creation package,
It's with you until your class is over,
and hopefully it's in your brain in a much lighter format in the end.



- 26:25, 01:15, 31:20

Every version of your sim creates a new set of this data.
Don't stomp on the previous sim however,
As you use the previous version to compare the difference,
Or when there are major changes, or you present a version.
You won't need to **save every iteration**,
because then your truely saving terabytes of crap.
But you need to save the version that matter
version 4, 7, and 9.
I usually only keep 3 for disk space issues.

Your scene file is the important thing to save in this case, not the data.
With your scene file you can always re-export it, especially when the art director likes the first version with a small tweak.

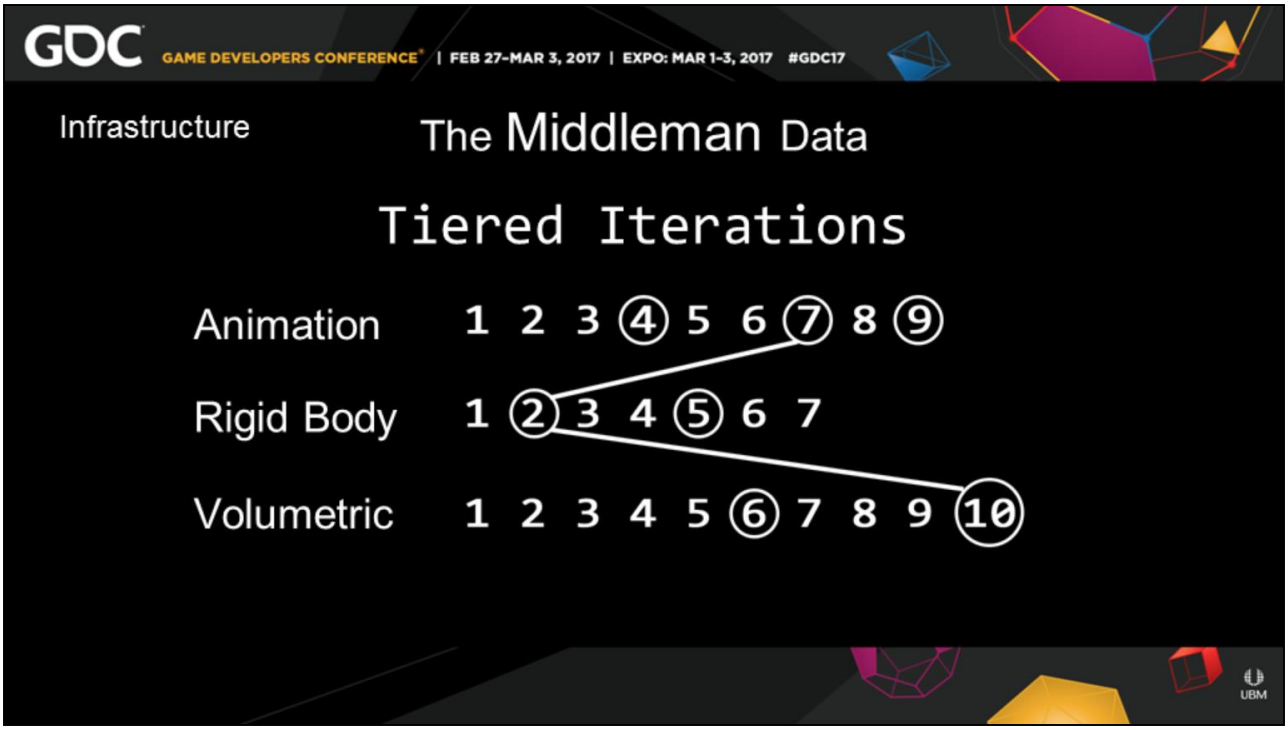


- 26:25, 01:15, 31:20

The complexity can go up as you have **tiered simulations**.
One simulation drives the next simulation.
A character animation that drives a rigid body simulation,
That spins off a Flui and Finite Element Solver
And a particle solver,
That also drives a volumetric solver,

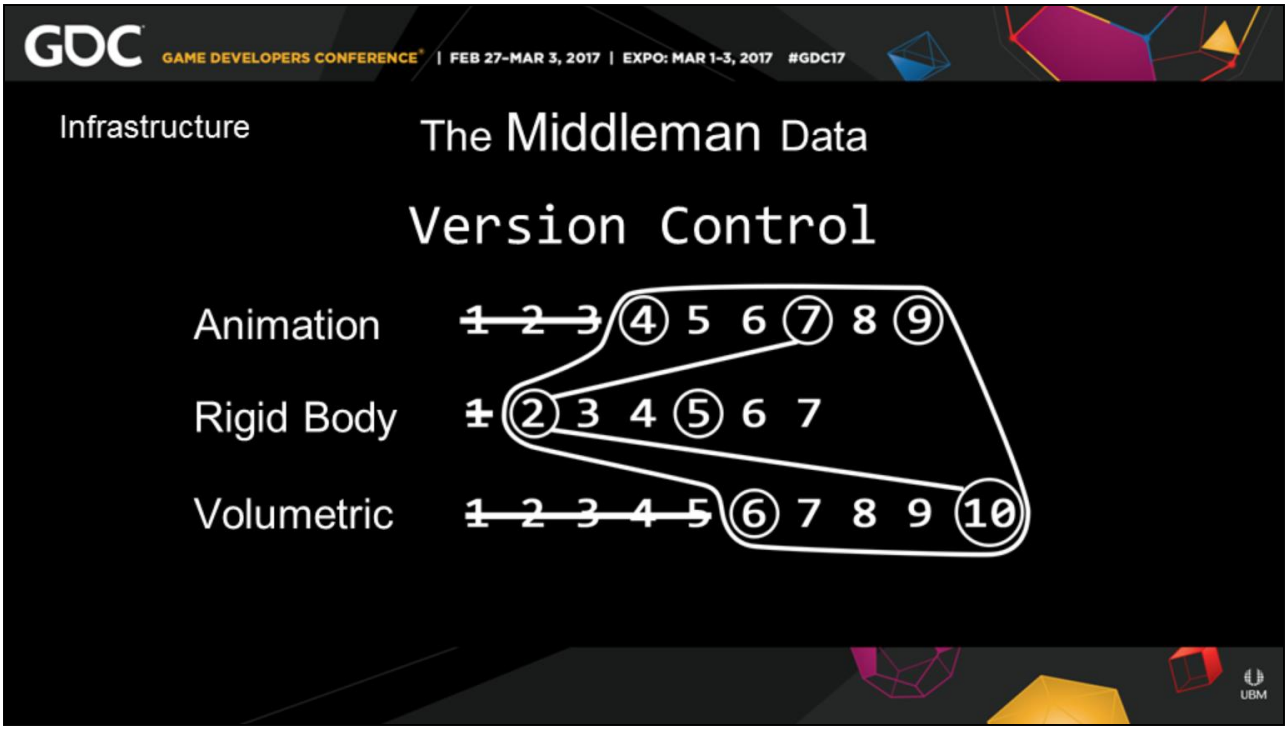
All so that the car can crash
into a baby carriage that crumbles
That is full of bottles,
That go flying everywhere,

Spraying liquid,
And the car blows up in Michael Bay fashion



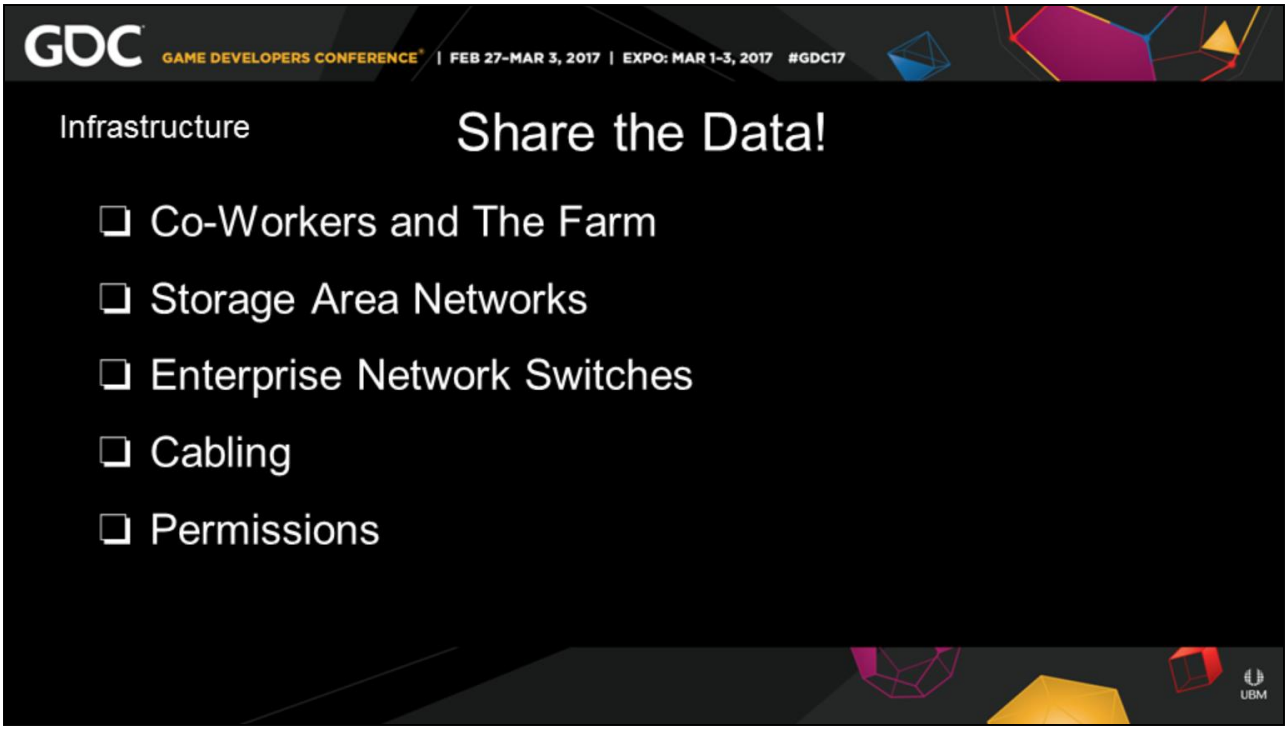
- 26:25, 01:15, 31:20

With this many simulations, you will approves each **iterations in tiers**.
As the first sim gets approved on version 7,
the next in the series may not get approved until version 2 or 10.



- 26:25, 01:15, 31:20

So in consequence to all of this,
The data needs to be **version controlled**,
And uniquely stored so that you can access,
Multiple versions and iterations at any time.



- 27:40, 01:15, 30:05

Now that you have all this data sorted, you need to **share it**.
Either with your **co-workers or the farm**.
If you are a solo operation this may not be as large of an issue.

But imagine you're a moderate size company, you higher in contractors to do work for you.
They come and go as the project changes.
If you are in crunch you don't want to re-sim everything they just did.
Or if they need help you should be able to open it from your desk.

This is where you will need some fat pipes to share your data.
The simplest setup is to just have **a virtual drive** that everyone can access
and work directly from this drive.

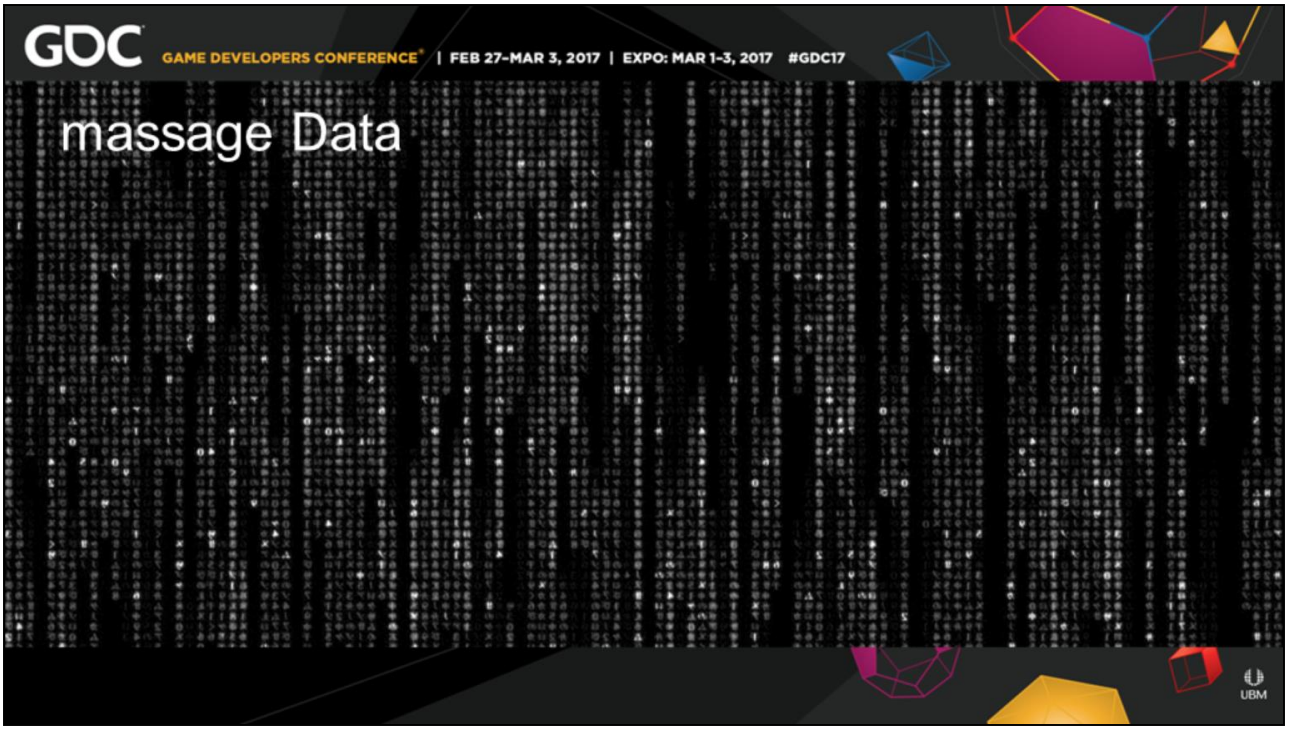
When you start getting to a larger studio you'll need **Storage Area Networks**, and **Enterprise Network Switches**,
not those little ethernet hubs that you and your co-workers kick at your desk.

I'm talking about serious rack mounted machinery.
Ask your friendly IT person they'll be able to help.
At a large enough capacity you become a full on data center,
imagine google's and amazon's warehouses of computer.

This may seem silly, but check your **ethernet cables**,
And make sure they were not made in the 90s
Plus install a dual network card on the workstations
You do not want to be bottle necked when you are not working locally.

If you do keep your sims with all their assets inside

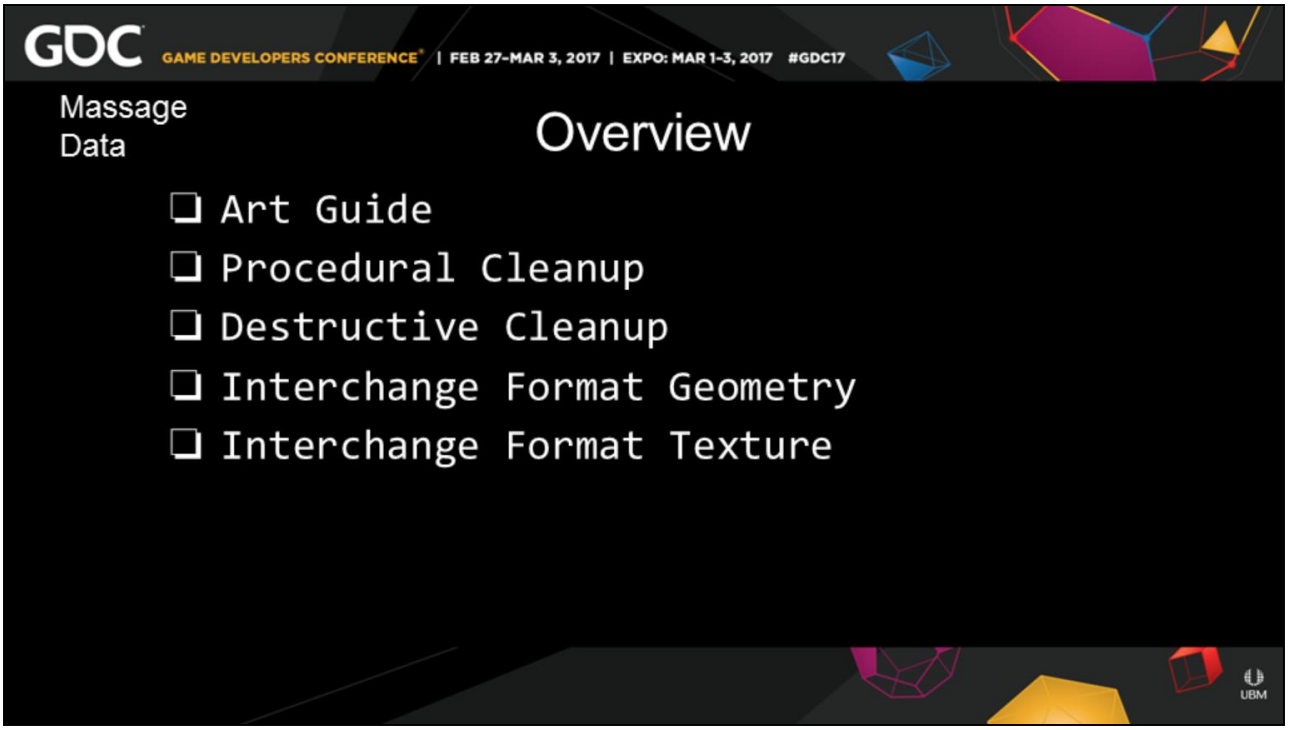
one machine it will be faster,
but it’s hard to give yourself dynamically more space.



- 28:55, 00:10, 29:50

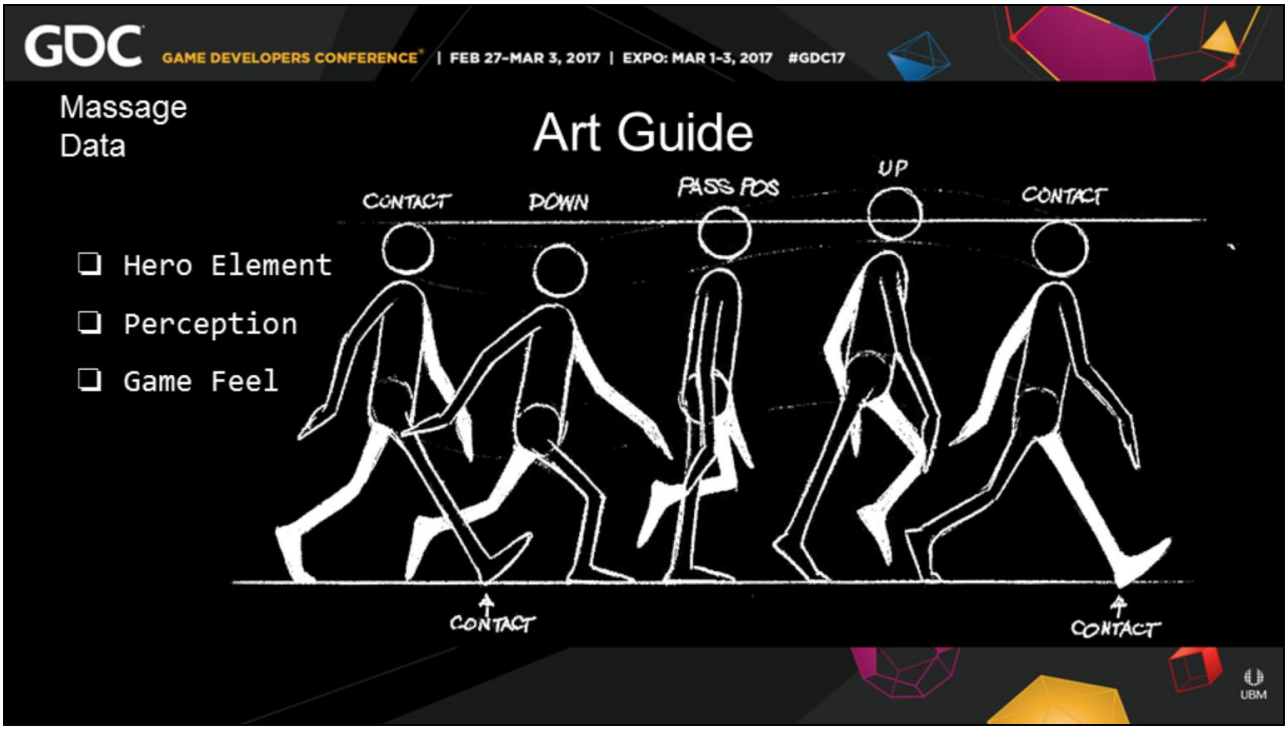
So your asset has been sliced and diced,
and now you have some clay to mold.

Odd's are your sim will not be perfect right after it sims.
However the data is like **clay to a sculptures** hand.



- 29:05, 00:15, 29:35

As you massage this data
There is a few different items to get your hands dirty with.
A last bit more on the **art** of your sim,
then **procedural** cleanup,
followed by **destructive** hand editing.
And your **interchange formats**



- 29:20, 00:45, 28:55

Depending on how much you art direct, and how hero the asset is,
your simulation may just be an **art guide** for your animators,
and that is OK.

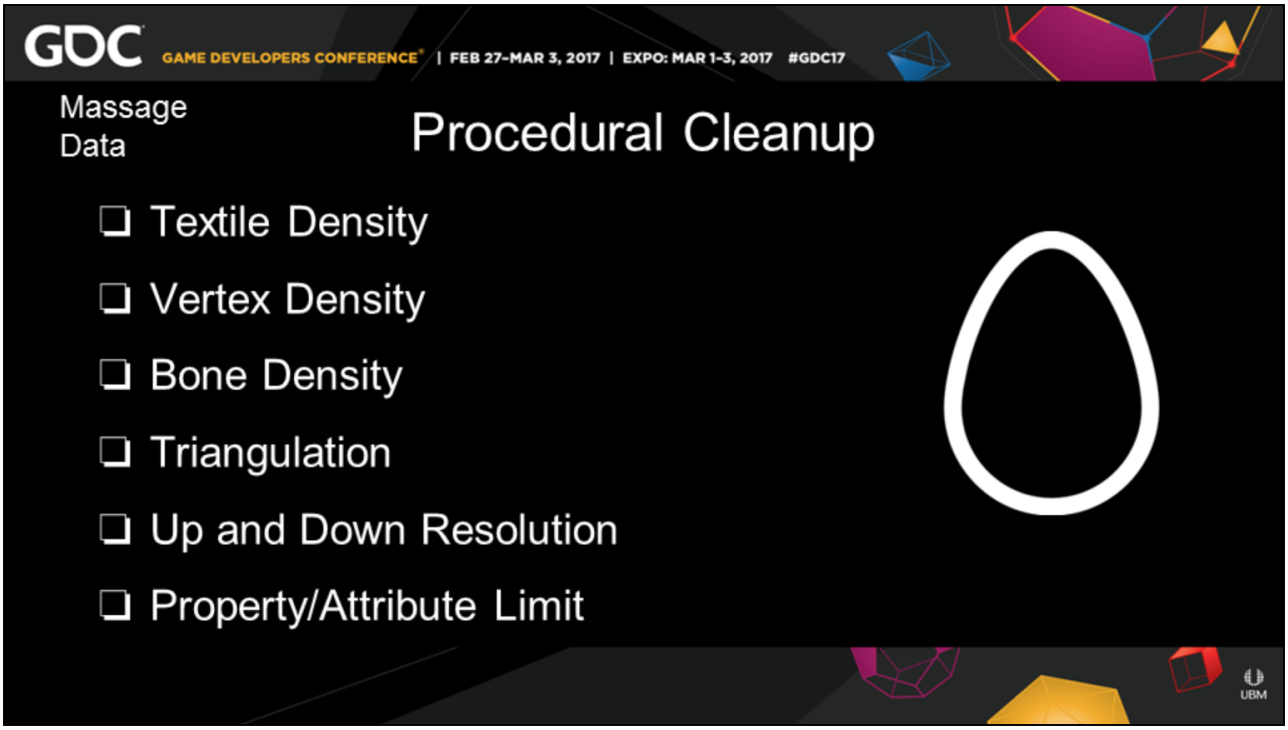
So during the course of your simulation you **did your best to get it right** the first time,
this is never easy as simulations can be complex monstrosities
there is a reason we use **computers to handle the complexity**

inherent with calculating the velocity, mass, collision, forces, et al.

The alternative is to hand animate every piece,
while a skilled animator could handle this,
the turn around time for them between iterations would be unprofitable.

However your **animators are your best friends** in this,
they have spent years studying motion, and they can tell you when motion feels correct.

They can even supply you with basic motion, timing, and beats to best art direct the simulation.
To the extreme point you need to be comfortable with,
it may in the end be best for an animator
to take your preped geometry
and hand animate on top of it.



- 30:05, 0:45, 28:10

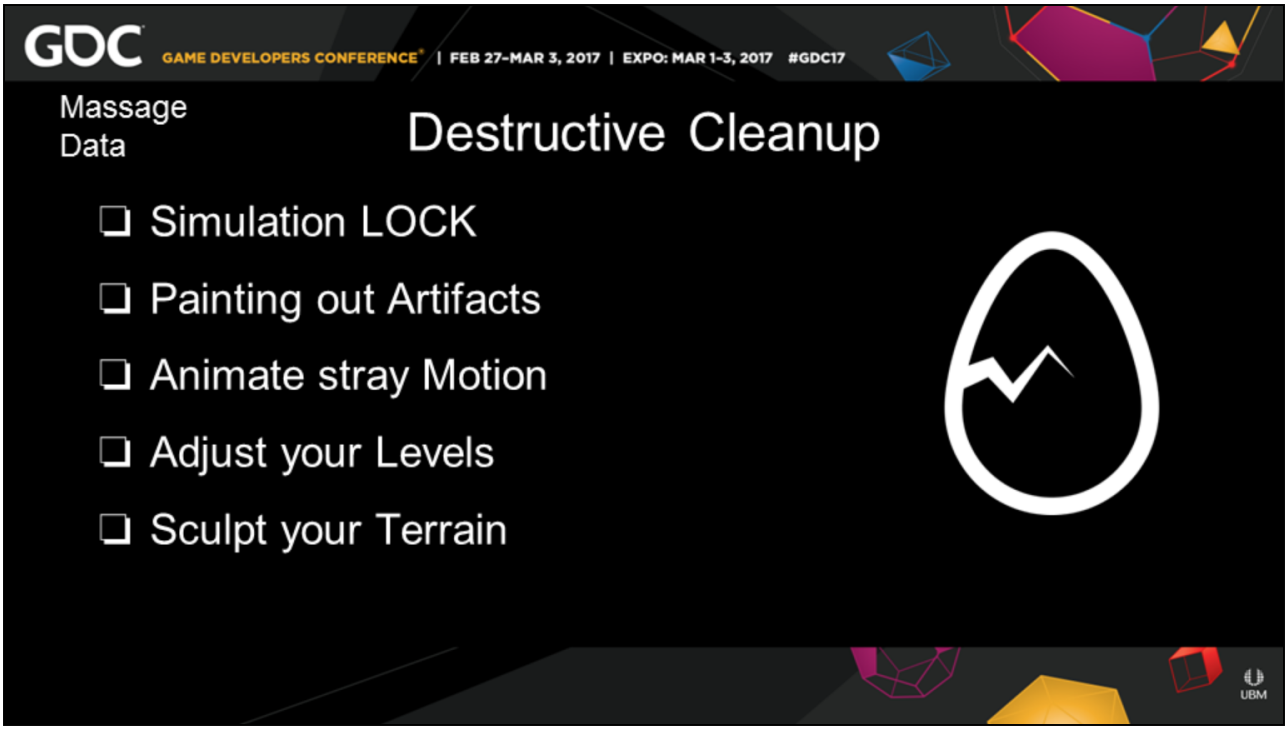
After you have art directed and corrected
You want to do some **procedural cleanup**
This is a type of automated cleanup that is dependent on your sim method.

For instance, we often did sims with a thousand pieces, That look great in the dcc,
but when you put it in the game and you go storming past it, it's not as cool as you remembered it.
So you need to do cleaning and reduction.

You have to deal with textile **density**. Vertex

density.Bones density.
The order of operations of creation and deletion of your data.
Traingulation over motion
Regional Culling of stray pieces
Swapping out your simulated data for **higher or lower resolution** final geometry
This could even be removing geometry from it’s entire existence of your sim
Doing any **attribute/property** cleanup or remaping.

And any Diagnostic operations depending if your exporter handles it,
or if you expect your artist to do it.



- 30:50, 00:55, 27:15

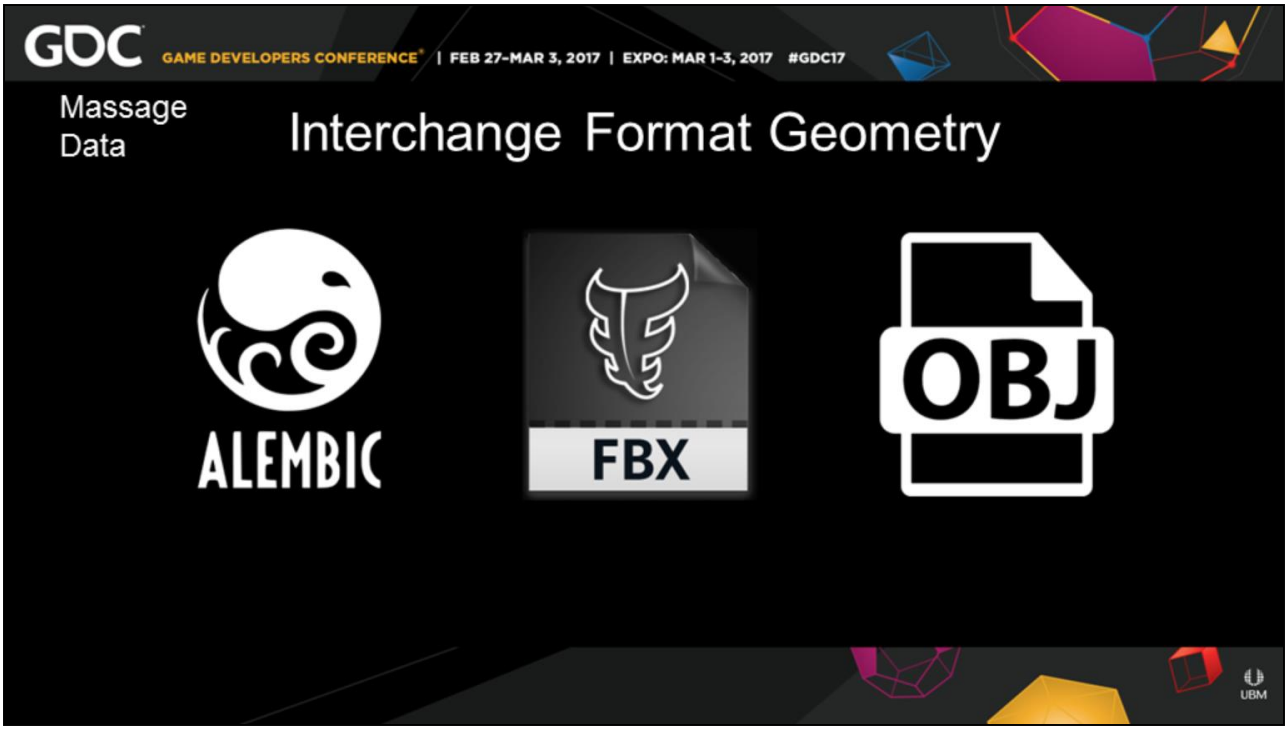
The alternative to procedural is **destructive cleanup**
You need to be very clear to all teams that the sim is **LOCKED**
so that no new geometry can flow through it. As it will no longer dynamically update.

If you are authoring textures
Where you take it into photoshop and just paint out artifacts in your normal maps
You can do this with your texture arrays in after effects.
If you are just remapping colors, you can do this

procedurally and automate it

This is **not a bad thing**, but be warned if you process requires this everytime
it will quickly become a time suck.
For adjustments like these survery your artist and make sure this is not a common task.
If it is repetative then you should develop a task that you can automate it.

This can be **animating stray geo** that rolled funny and is too hero to cull.
This can be using your world builder, and doing some fine tuning to it.
Or this can be scultping edges of your terrain.



- 31:45, 01:00, 26:15

Once your sim is cleaned up you need to transfer it to your game engine
and amongst your other dcc.
This is where your **interchange file formats** come in.

Your simulation data depending on the content you are creating,
can be extremely **different** than your normal data.
You can not use a lossy format like in your engine.
And you don’t need the editing capacity of the dcc.

So which file format for sims are the **best???**

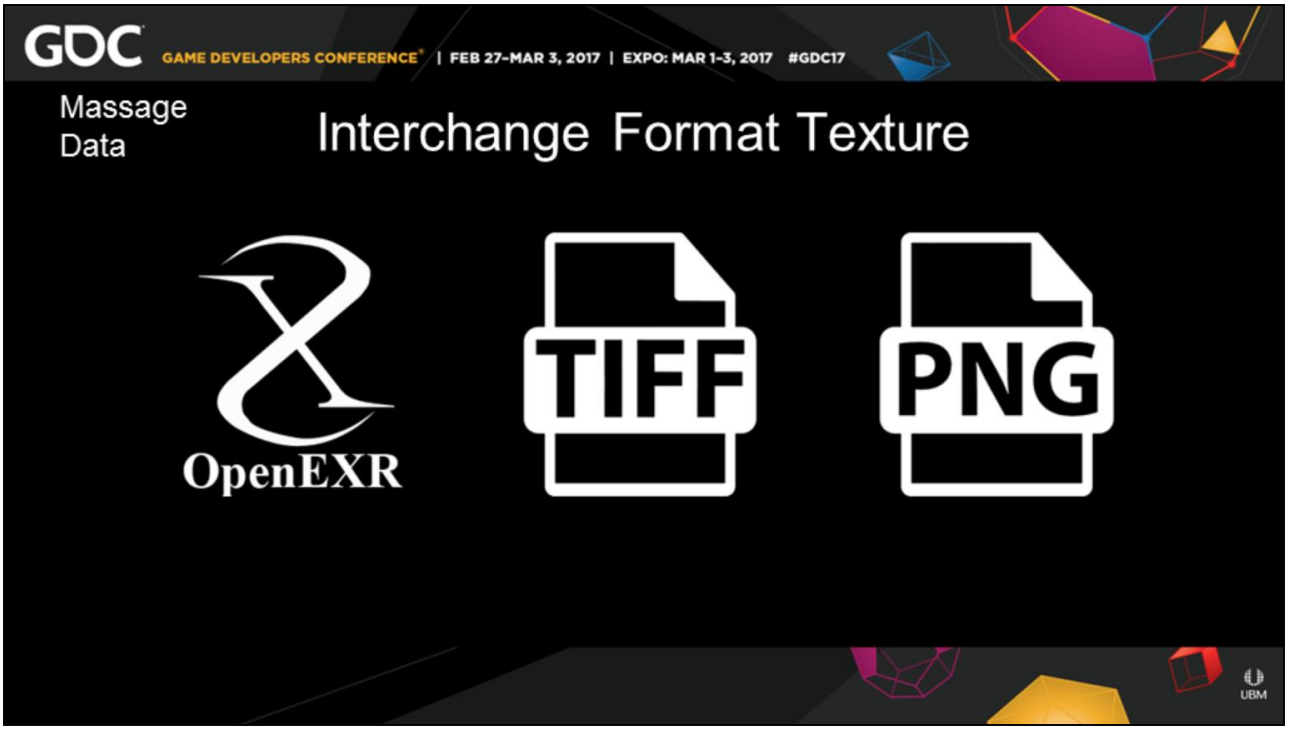
We found for geometry **Alembic** was the best choice for our studio,

First it is **open source**, so we don't have to pay for it.
Secondly we can edit the internal format to how we use it best.
We don't need to piggy back additional xml files along side it like obj.
It can read any line or frame of data at any point
For a vastly better **Read/Write** ability than FBX.

I can play back a whole destruction scene of several buildings in real-time
and an animator can still do their job.

One issue is the fact is that it **does not natively support rigs**,
For your character style rigs I would still recommend FBX
However, you can amend alembic to handle this.

One inherent issue with all interchange format
is the fact that no two packages natively read and write them the same,
so you need a wrapper to prep the data and saves button clicks.



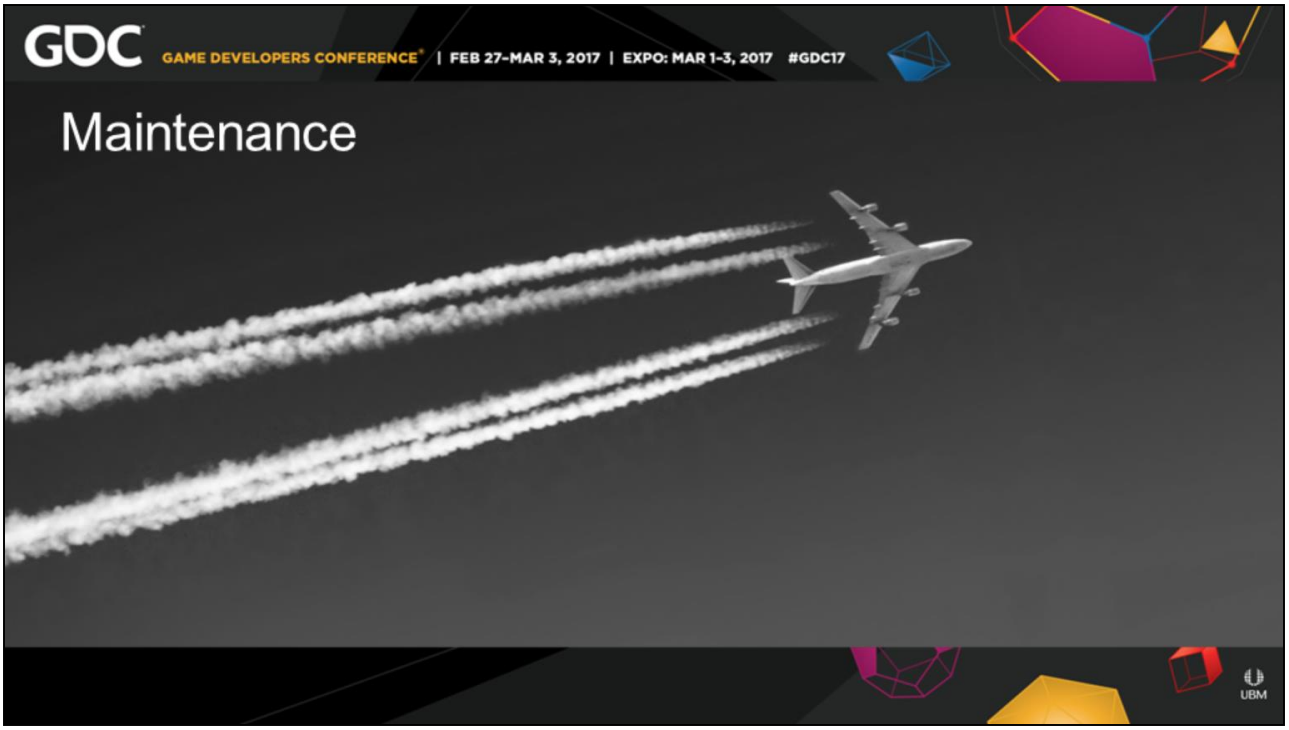
- 32:45, 00:45, 25:30

For texture based mediums
You often be exporting **non-standard data** with your sims,
This could be geometry caching data. Displacement maps.
Or even more arbitrary engine data
So please never use a **lossy format**, such as, jpg.

You'll need ranges that can hit up to **32 bits of data**.
So unless you are having hard drive limits
And/or are writing straight texture color,
Png style formats are not good.

The next wrung are files like tiff or targa which are workable.
And are a known quantities
However if you are building from scratch,
Or are you revamping for **HDR**,
OpenEXR is a format from this century.
Proven and designed to handle this type of data.

It’s highly flexible and it’s **open source** so you can re-arrange the data how you need it.



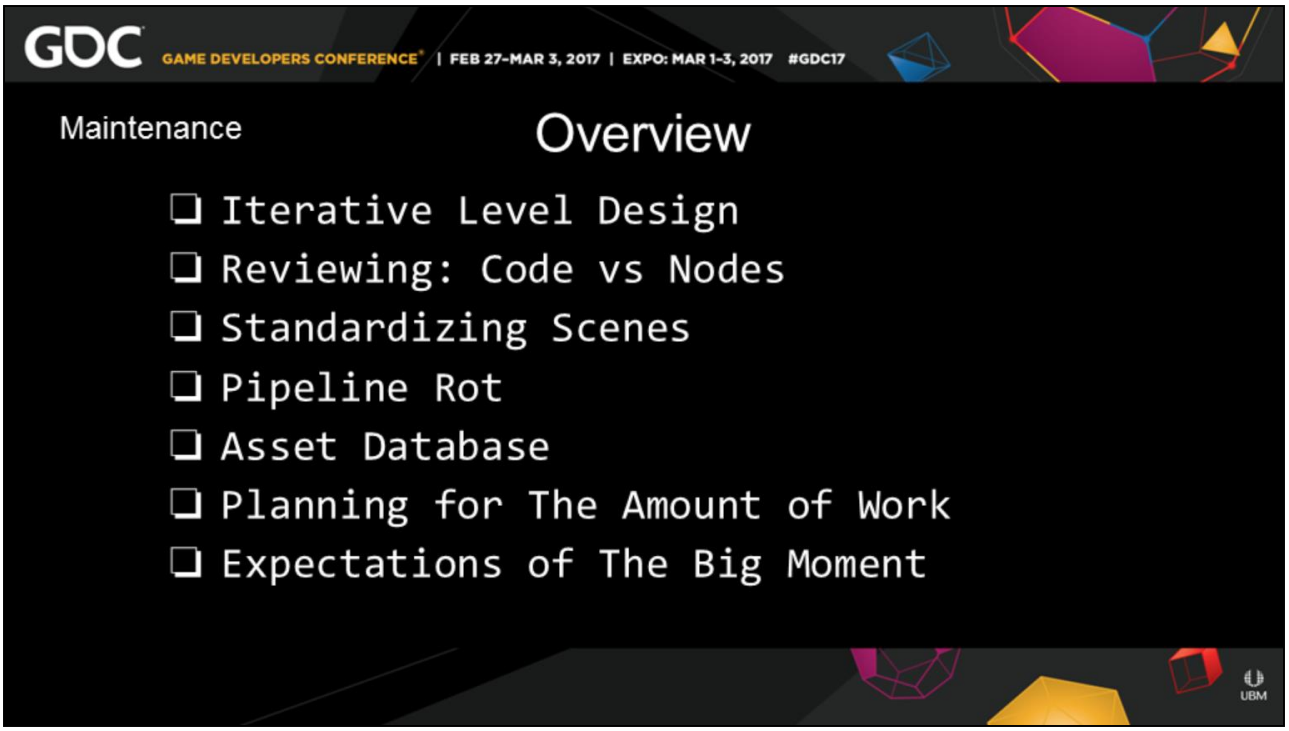
- 33:30, 00:40, 24:50

Now comes the long haul.
You need to be able to **maintain** these asset over the course of a production

Once **an asset** has been created **it’s not the end of it’s life**,
as the levels evolve and the engine changes you need to keep on fine tuning it for shipping.
To squeeze it into an acceptable download, or on to disk.

We've been talking about a dynamically adaptable pipeline to this point,
so this would seem by it's nature it will cause chaos.
And YES in large portions there is some janitor work involved to unify some of these scenes..

What you are making is a **living set of code**, more than a static model.
So as the level updates and changes from concept, to design, to block out, to your first and second art pass,
through your bug fixing passes.
And that final tweak before shipping.
You need to manage this data.



- 34:10, 00:30, 24:10

Will cover handling that data from a few different areas.
Iterative level design, and geometry updates in general affecting your sim.

I’m going to talk about the concept of **code review versus node reviews**,
this is about bridging programing and art at the scene file level.

Will talk about ways to **standardize your scene**,

we are doing a lot of unique items, but inherently there is a lot of similarities.

Then will talk about **pipeline rot** with this system, **asset databases** relevant to simulation work.

Planning and budgeting for the amount of work.
And **managing people's expectations**
when you are working on the big moment.

- 34:40, 01:20, 23:00

In our example case I'll take the our moment of the Kraken Reveal.
This was one of the first levels to be set up for Halo 5
early in the production
so it inherently had it's issues to begin with,
and we we're just setting up this pipeline in that time.

This scene was done by two artist,
we ended up switching shots after the first concept

of this moment evolved.
However we didn’t design anything at this point to be dynamically updated as the level changed.
And we learned right away why this was a bad idea, as this was the opening salvo of shots updating.
And there was no way in hell we could repeat setting scenes up from scratch each time something changed.
and get anything to ship.

- 36:00, 01:15, 21:45

So we setup a process to handle **dynamically changing geometry**.
From iterative level design in our setups.
Will go from the micro level on this and expand it out to the macro over the next few slides.
This section will be a bit more Houdini focused.

The first thing was to **standardize our imports**, from our downstream dependencies.
If your familiar with asset tracking this was our first step.

Knowing and being notified when your downstream level is updated.
Via communication, e-mails, and software notifications.

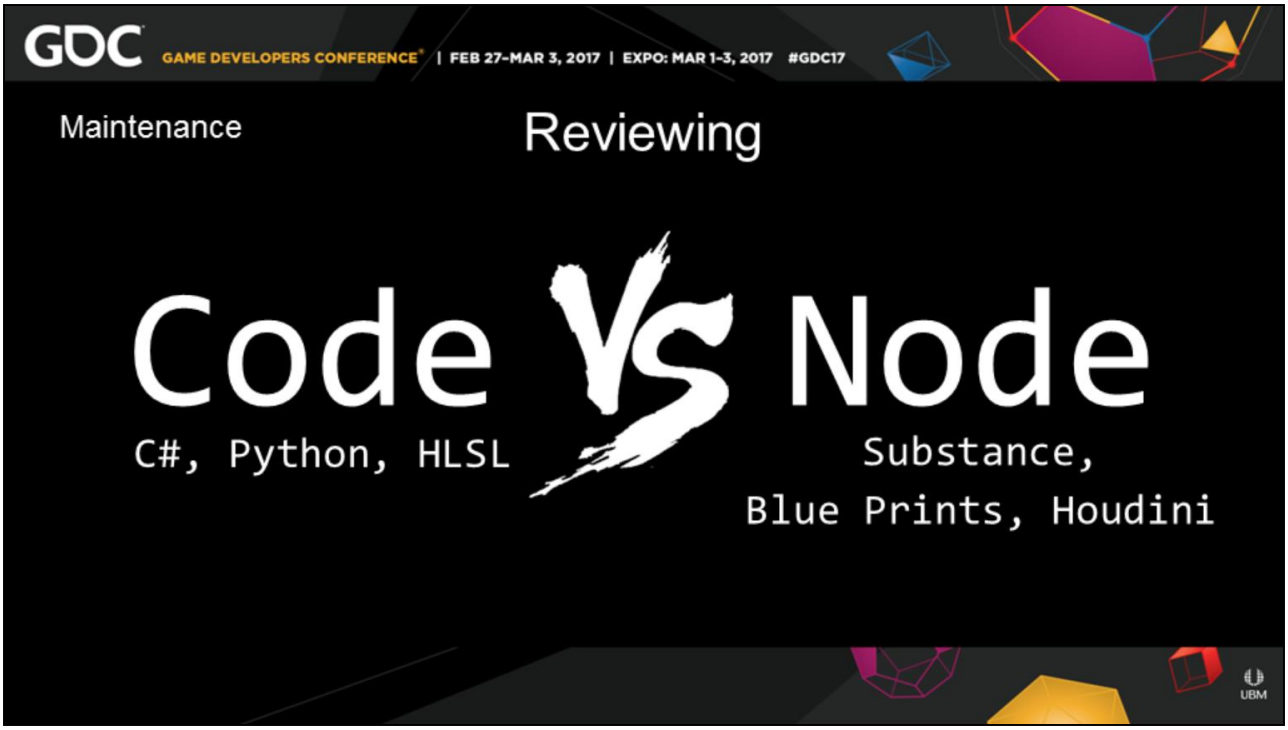
Our asset pipeline does not have a rugged taging per a primitve system,
For dependent teams, which I highly recommend.
So we got flexible and made all the **selections based on relative** bounding areas.
It allowed us to have a flexible tolerance for a door changing designs,
As long as the door remained within a reasonable range.

We standardized our **disk caching** and game asset exporter.
So that no matter what scene you are in you could easily find them.

We made a rule of building a complete asset within in **one container.**
So if you needed to share it, duplicate it, or navigate it there was no missing files.

Including designating our in and out points for each context. So they were clearly readable.

We set up standard internal scene based **dependency triggers**
To be able to remotely reprocess the scene.
This helped beyond just updating incoming geometry
As our intermediate and final game files updated their formating over the project.
For the Kraken Reveal that was two dozen assets update and set up for review by clicking on one button.



- 37:15, 01:15, 20:30

So our simulations can be described as very organic code, as opposed to just straight content.
As the scene file is more important than the final output.
So with this similarity it was suggested to do a node review.
just like a code review is done in programming.

At first I was hostile to this, I'm still mostly an artist.
However, it is a valid argument,
that is for any submitted code you do you have it

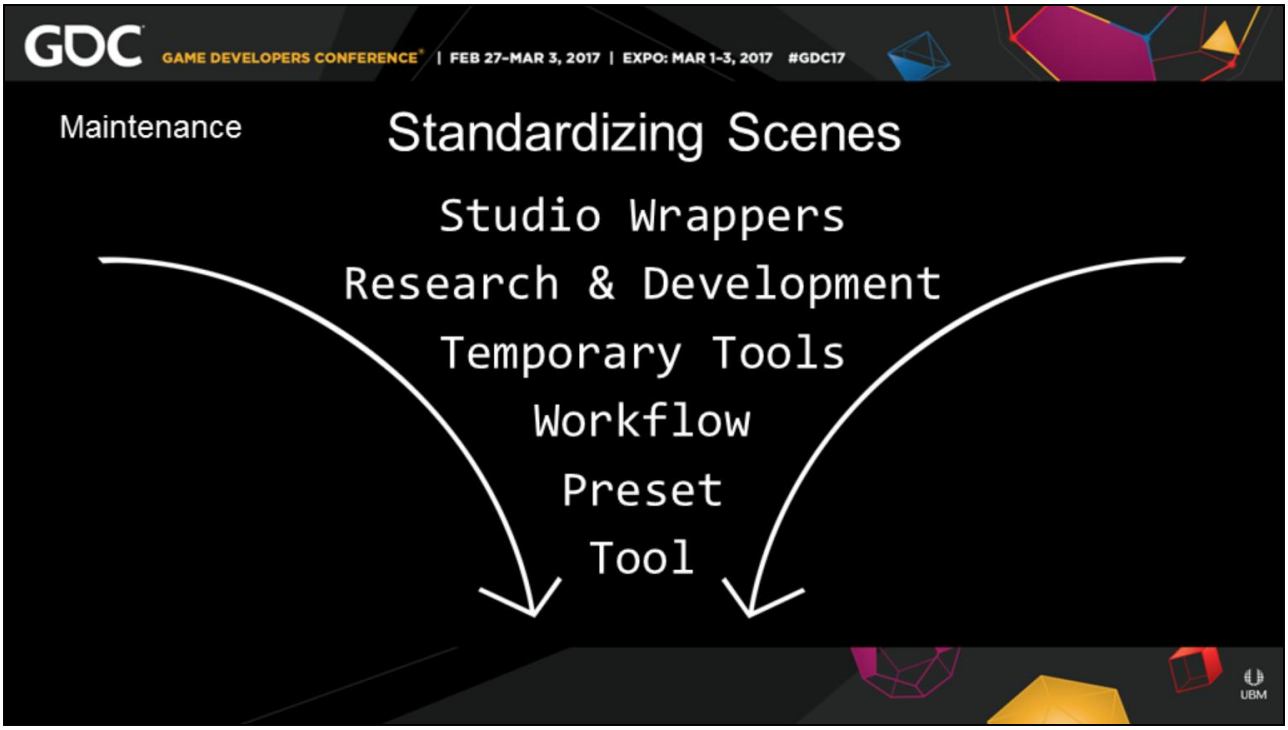
reviewed by your peers,
the same way would work for node setups.
Plus, **Node sound like Code**…

This would prevent having vastly different method of doing the same thing.
In principal this would save a lot of maintenance and head ache.
The other side is we have a methodology while technical is still more Art than code.
How ever you are forcing your artist to work in “the” way.

To balance this out I asked a jury of my peers,
who deliberated and came back to me with this.
If a studio has pre-defined standards, workflows, and tools
before they get there this is acceptable.

So as Halo 5 wrapped up we made presets of our most standard workflows,
we made tools out of anything that was common amongst the workflows,
and we setup up coloring and naming standards so

that the easiet way forward was together.
This reduced our janitorial work, in what we currently think is a happy medium.
And we were abe to easily out perform when it came time for a year of DLC content.
I’ll reference these are available of Orbolt.com later.



- 38:30, 01:20, 19:10

So in order to make a happy medium i broke down our setup into a couple of descending basket.
First is the **studio wrapper**, this controls my larger environment variables,
scene sets up, node color, naming configurations, etc.

Research & Development or prototyping time is adhoc.
You are going to try a dozen different example files, a whole bunch of different methods

as you fail fast. Most of this work is throw away, but you make sure people version up,
branch, and document what they did. You don't leave around giant trees of dead nodes and scene files.

Temproary Tools, are parts of your R&D that have coalesced into a repeatable workflow.
This is a proto workflow into something you really want to give to any one else yet.
You ahve figured out enough of the common buttons to partially wrap it up.
These are usualy the bi-product of a sequence or level area
where you will use the temporary tool setup for a short time,
but it might not be used elsewhere.

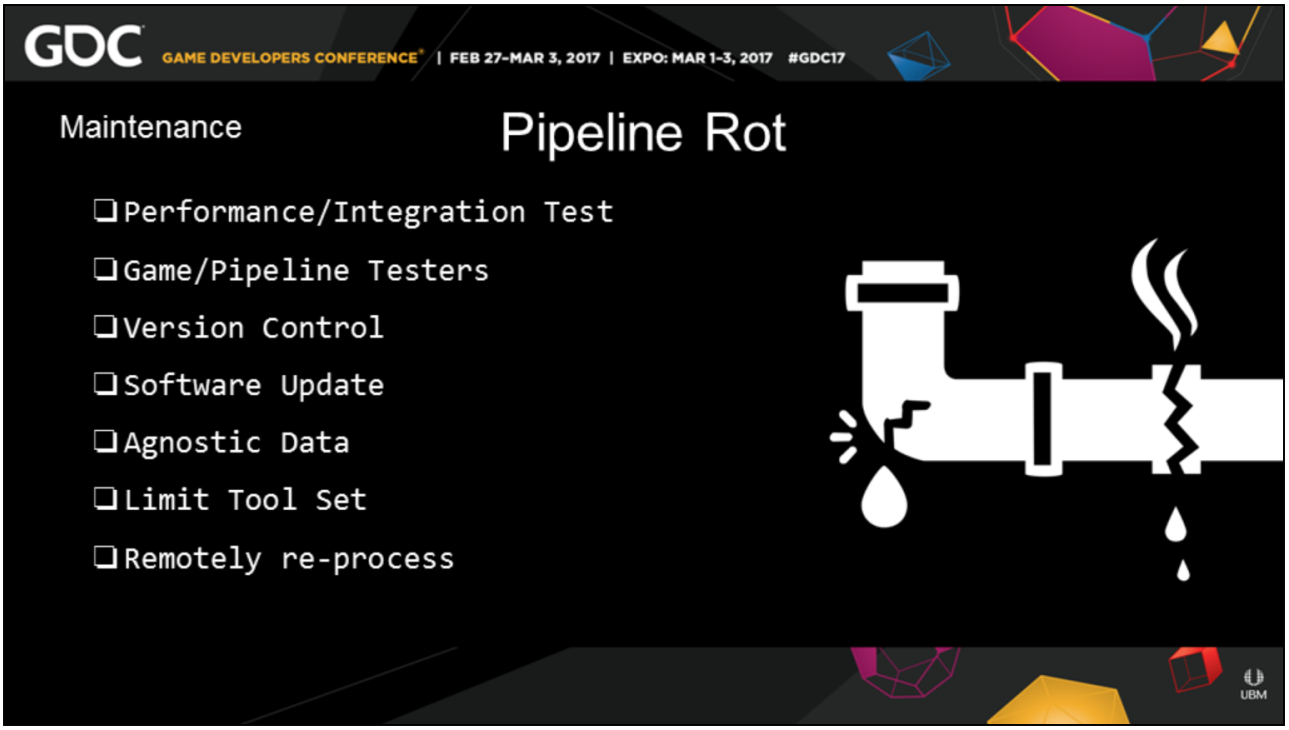
A **workflow** is a repeatable pattern of work, this is usually a sequence of smaller tools
that you can link together to make an item. I've created Texture Array Setups, RBD setups etc.

The next phase are **presets**. This is an array of

tools,
that give you an asset right away, you can still move things around, but there is less need.

Tools are when every element of your system is setup
and you can just pound a button to create hundreds of assets.
These may also be smaller components that do very specific pieces of work like exploded views, or fuel setup.

With all of these inplace prior to some one working at the studio
you can safely catch them up and walk them through the setup.



- 40:50, 00:45, 17:25

Pipeline rot happens as the engine and tools update After content has been completed.

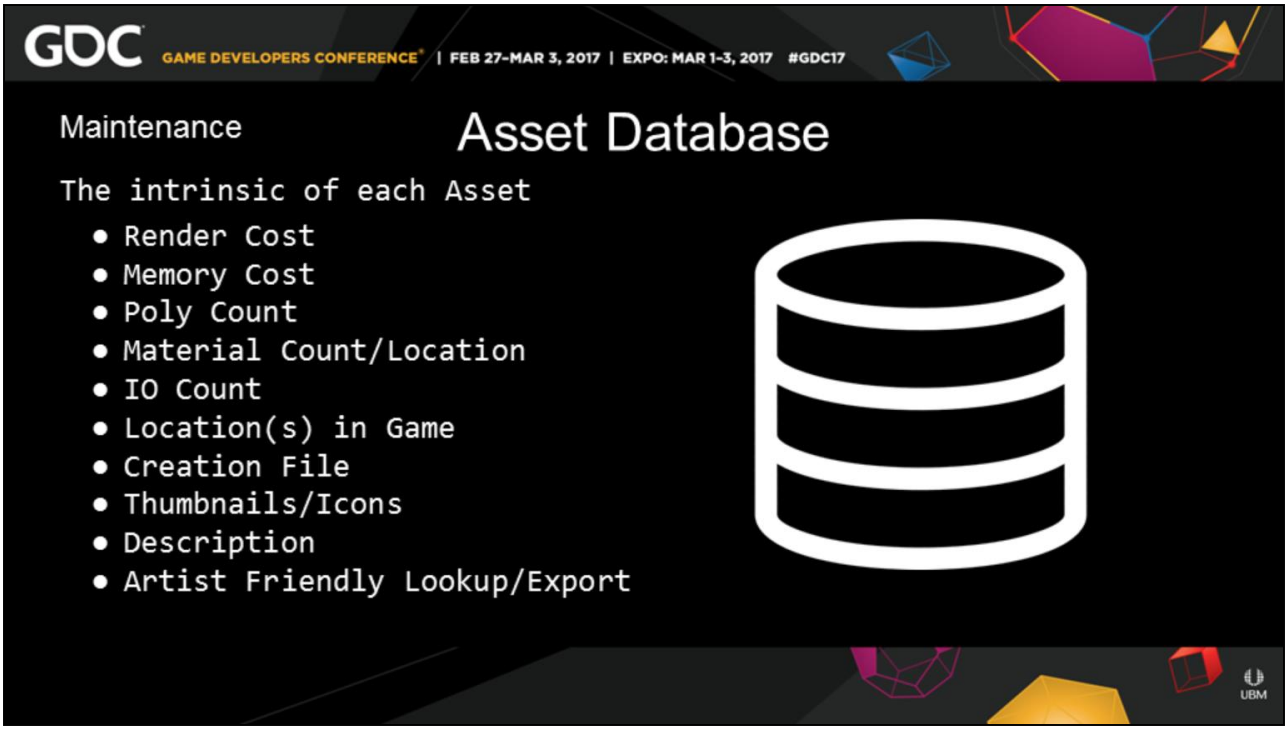
Your **Game/Pipeline Testers** will discover almost all these bugs and issues,
that can cause you to re-run a sim as you notice something doesn't look right.
Give a shout out to Dan for his help.

On over a hundred unique scene files this can get pretty severe.

All those tools need to be very carefully **version controlled**,
and when you choose to **update your software** you need to be able to re-verify everything is working.
with **performance testing** tools

Carefully calabrating these can be painful for tools that consume **agnostic data.**
Keeping your **tool set simple** is key.
My current tool set does not range more then 25 to 40 tools the artist can access,
and I continuously hide or update tools depending on their usage history.

I have also setup batch scripts to re-open every scene files and **re-process** all the assets.
You may only need to re-cook the middleman assets into their final formats.



- 41:50, 01:00, 16:10

Maintain a database of your assets and their intrinsinc properties.
This becomes really important towards the end of your project.

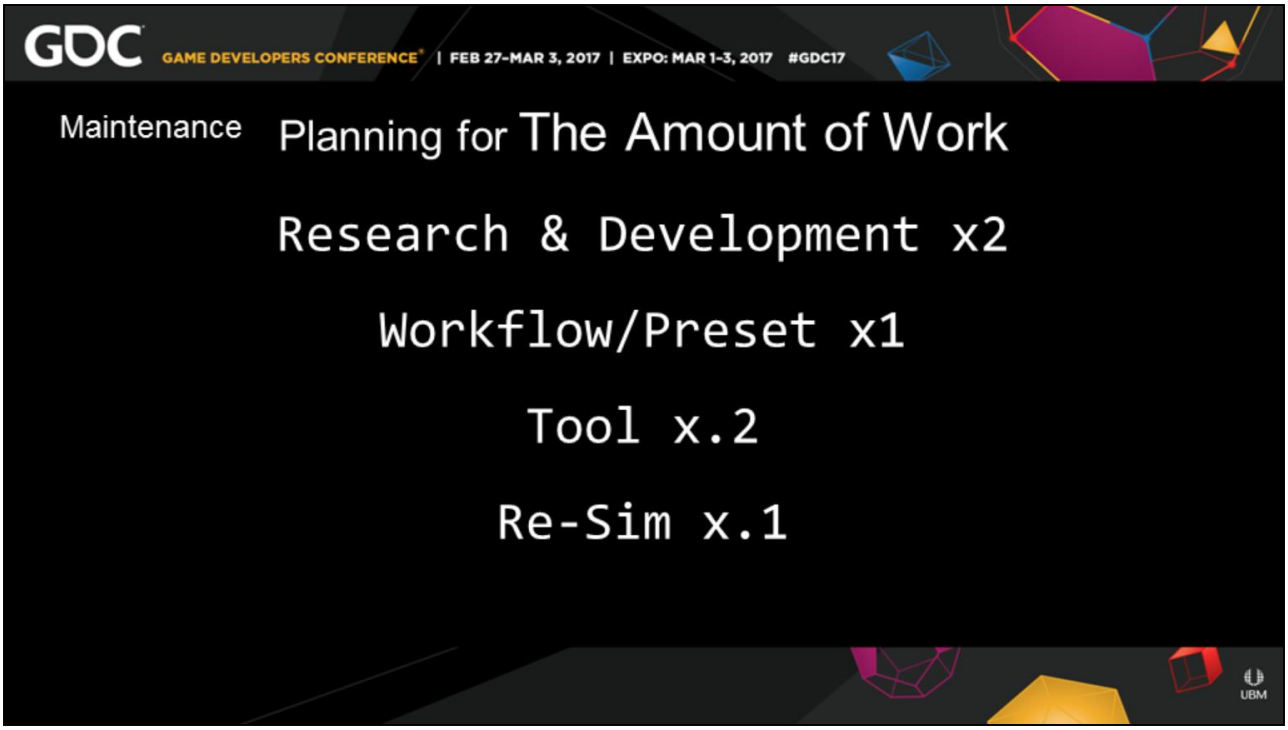
When you have a large enough team,
and you can’t keep track of every thing they have produced.
Over the long haul **shit breaks**, it ROTs,
you need to clean and optimize stuff, your golden path shifts.

Some one tells you that once you won't touch something you are now up close in your face.
But if all your contractors are gone, or you buddy that you are working with is on vacation,
you need to be able to track down those assets find where they were created,
What software was installed, and what they need to fix.
Having this databased is awesome.

Also since you might be working on somethign that is fairly complex
it will prbably have a larger than average footprint.
This footprint is usually the first thing noticed at performance reviews.
So if you can track every piece of info on that asset from a webpage in the performance review
you are that much closer to being out of the production eye.
Plus yu can make intelligent decision on what should be cut.

I would highly recommend that this is auto

populated and updated on every export.
The simplest ui you can give to an artist to fill out
And other intrinsics on what this assets is for is helpful.



- 42:50, 01:00, 15:10

When it comes time to budget for work,
I have a series of multipliers I use.
The base component defined by your team skillset.
But these work as a general rule of thumb,

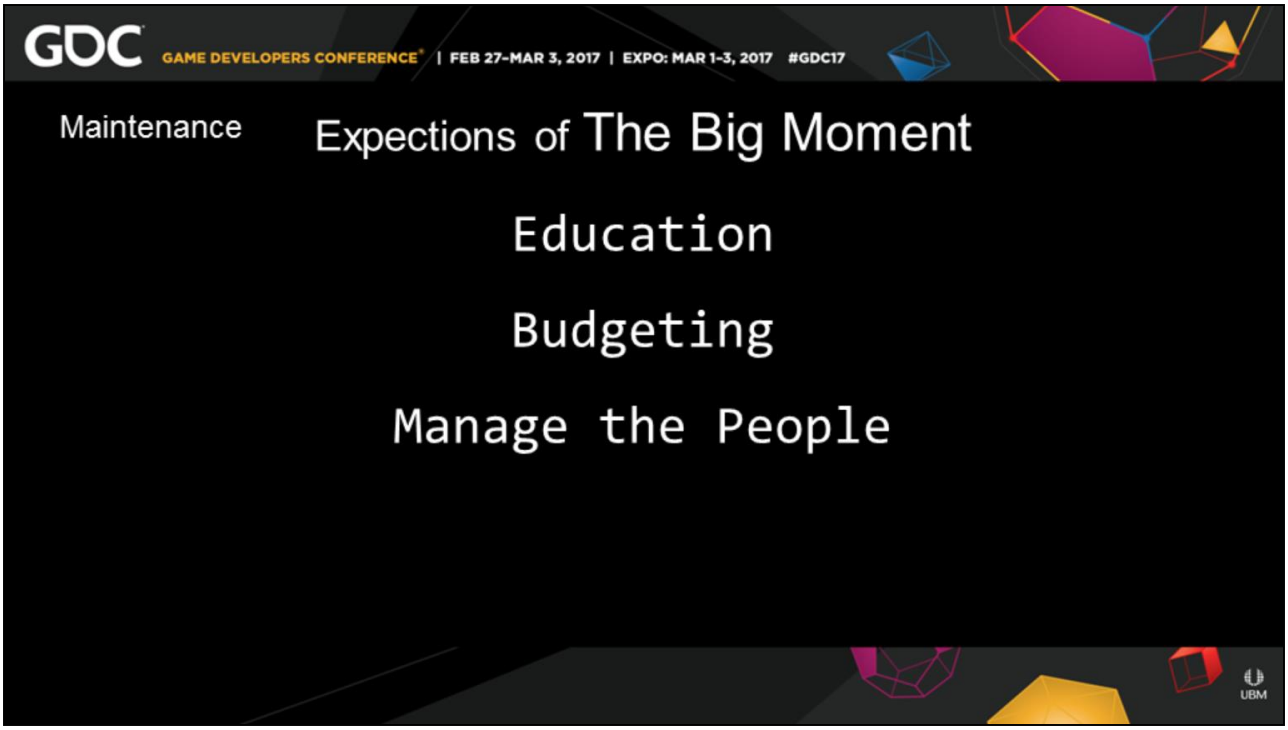
Research & Development is relative to the effect you are working on,
but you should be able to get something out in a day,
2-3 days for a middle level R&D,
and a week to a sprint for a pretty complete asset you have never done before.
For an entire level building system at a AAA studio

this could take a year,
but you are talking about across the studio changes, not a single moment.

Workflows and presets are equal to your base amount of work,
As theses are the known quantities you have produced before,
And you can budget based on that task
The more workflows and presets you have the quicker projects becomes.

If the work has been done often and has been encapsulated into a single **tool**
You could literally create hundreds of thousands of versions,
and these can be torn down and updated quite easily.

Finally a **re-sim** should take the smallest perecentage,
However,This is relative to the complexity of the sim.



- 43:50, 01:00, 14:10

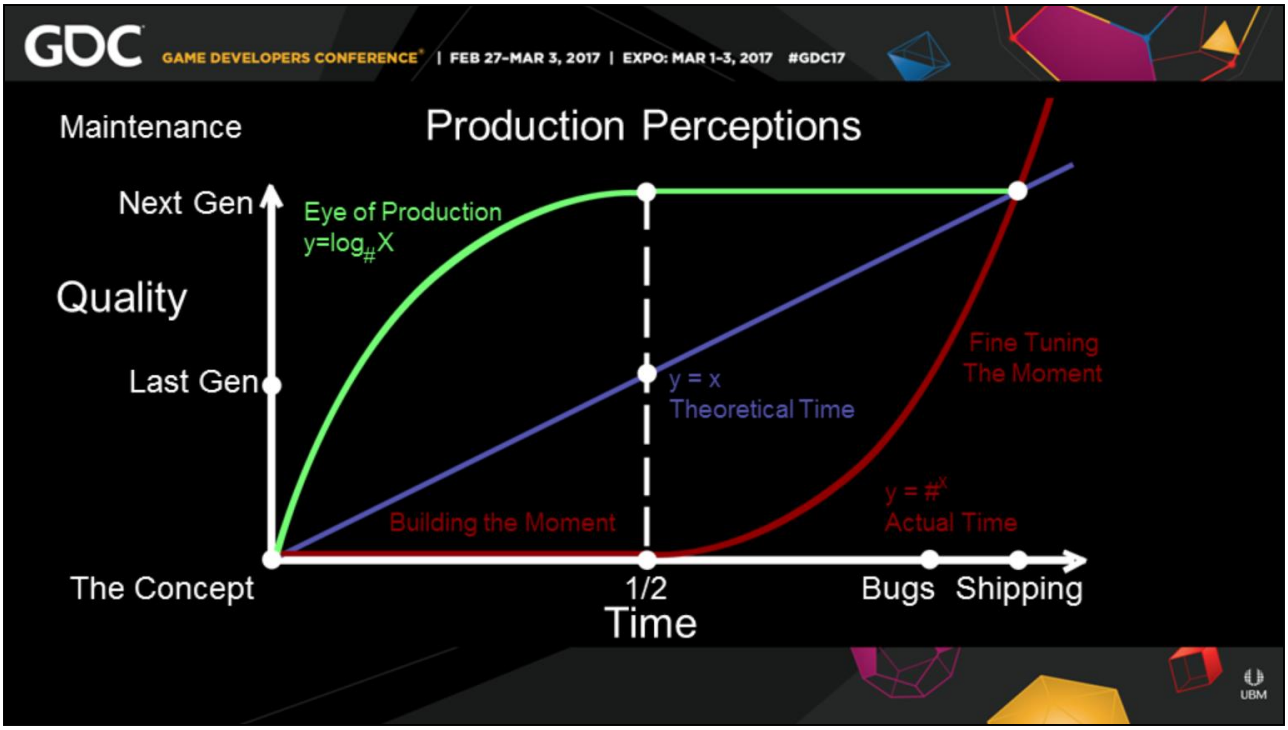
The other thing we constantly encountered with this pipeline
is that we were working on the bigger momments of the game.
These are the point in time where the consumer sits back and takes a breather,
or else they are in the true middle of the shit.

In order to handle this you need to help
Educate people on your process,
Budget your time accordingly and **manage the people** involved.
This is on top of your simulation work.

A lot of our simulations we did were the eye candy.
And when eye candy happens you want to do something new and unique.
Nobody wants to see the same thing over and over again
as these big moments are when the story builds up.

As we described our pipeline needs to be adaptable for these moments,
So thus we work out of workflows and presets to make these custom unique options,
So we inherit a lot more pereceived R&D time naturally in our process.
This means we need to budget for it accordingly
This also means you need to manage peoples expectations for each review
That the quality may not change drastically from their last viewing

If your quality does not perceive to change,
It is hard to judge from an outside point of view why it did not change,
So this is where clear education and communication matters.
It helps keep level heads, and allows you to adequately make proper decisions



- 4:50, 01:00, 13:10

I made this graph on a napkin note originally during a production meeting
To convey our workflow expectations, as a generalized production curve.
I've found it helpful to handle perceptions

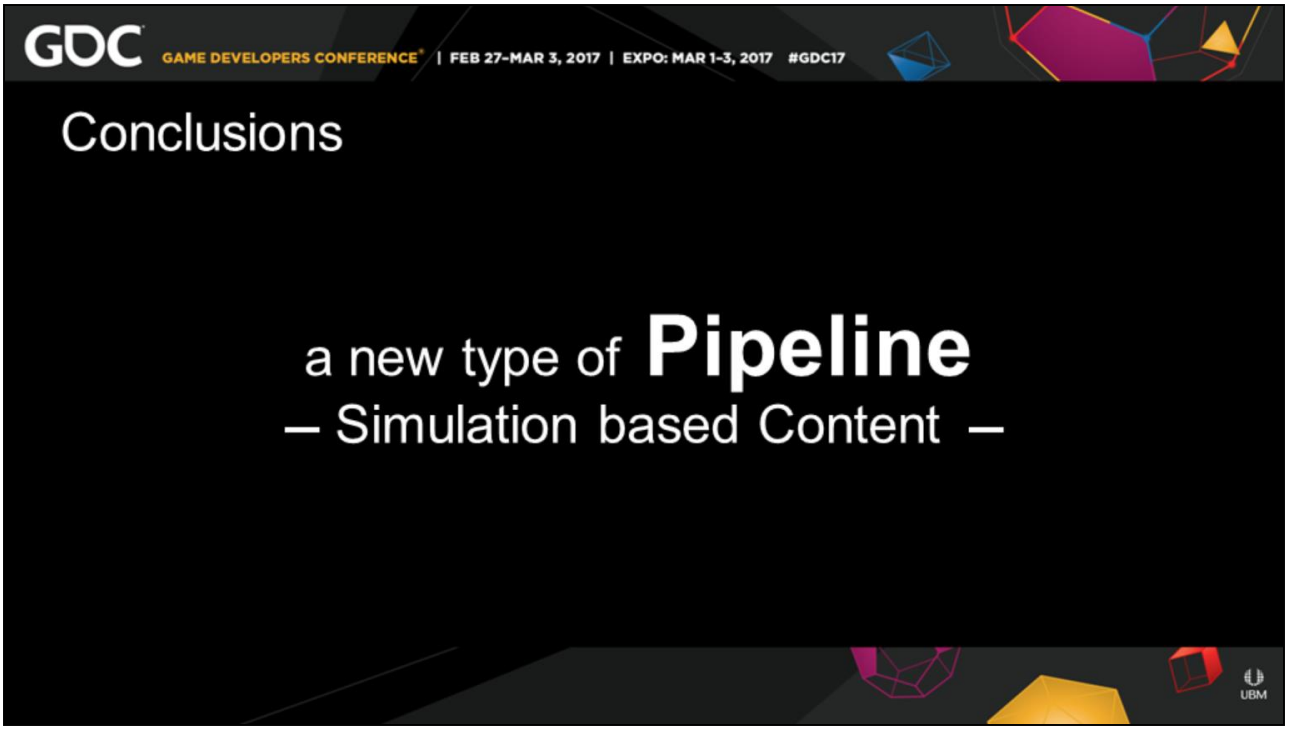
The **blue line**, in a traditional pipeline, such as texture painting, is a very common workflow,
The quantities are known and it's a bit linear,
Half way through the process you expect to see 50% of the quality,

The **red line**, is usually simulated content, especially for building unique assets,
Where even with a known quantity you need to prepare,
And do the necessary simulation work which takes half the time
Before you start over working on quality and iterations.

The most important line is the **green line**, this is the Eye of Production.
You are working on that extremely unique moment or asset, that E3, that level show piece.
All the eyes are on you!

The rest of studio will thank you as they have a breather as you are constantly in review
This is where content will get killed, even if you are meeting or exceeding your deadlines.
It’s extremely important to take a minute, Breath and get everyone on your team on the same page.
On what their expectation of your process are.

Everyone is invested in making the best product you can
Within the scope you made available.
But this can only be conveyed in a simple upfront manner.



- 45:50, 00:30, 12:40

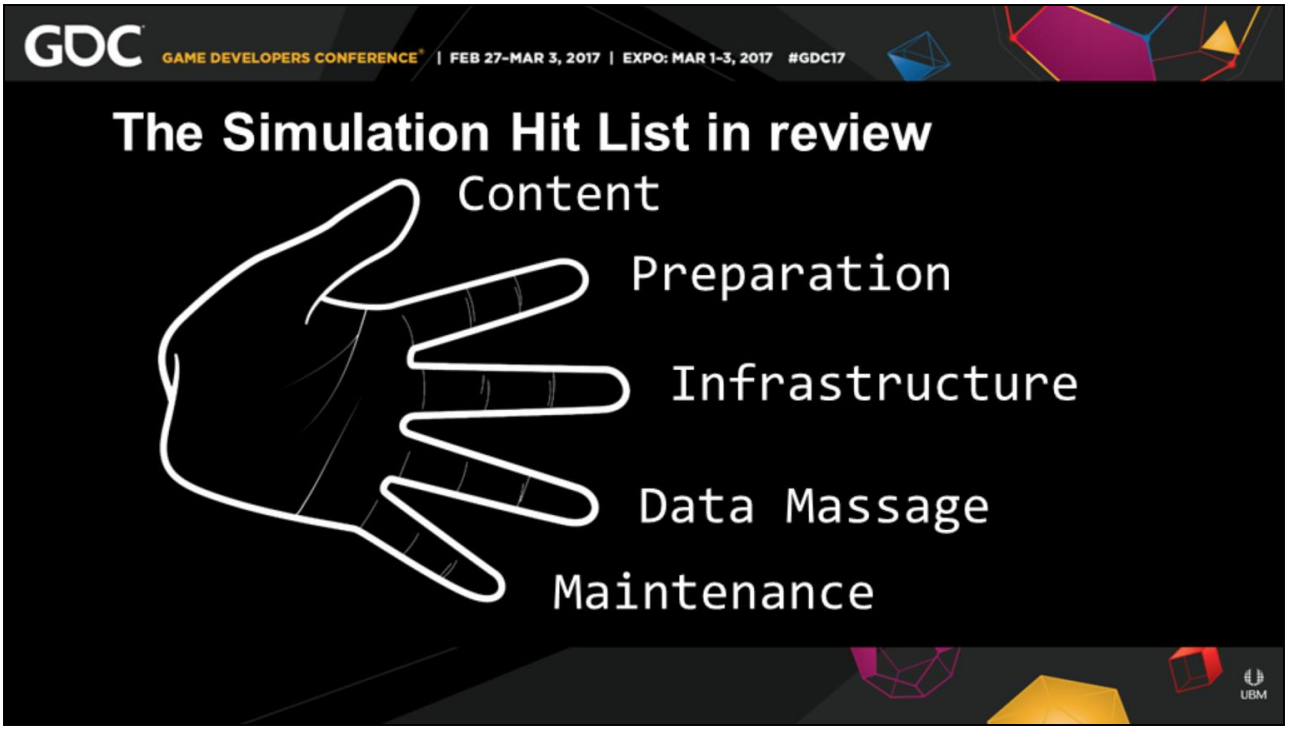
Now we have hit the **conclusion** of the talk,

So we covered today how to create a hybrid type of pipeline
A simulation based pipeline.

I hope you learned something new,
or refreshed something for your studio

No matter the size and the capacity of your production.

Simulations can allow you to create **far more complex content** than
You would normally be able to author by hand,
and I hopefully I shared some of my experiences to make it easier.



- 46:20, 00:30, 12:10

So coming in to the end of the talk,
I hope you have your questions ready, I have 4 slides left.
::point to the mic::

So in a final review of the categories we talked about.
If you remember the random items I had you write down.
Those will be helpful talking points for you to take back to your peers.

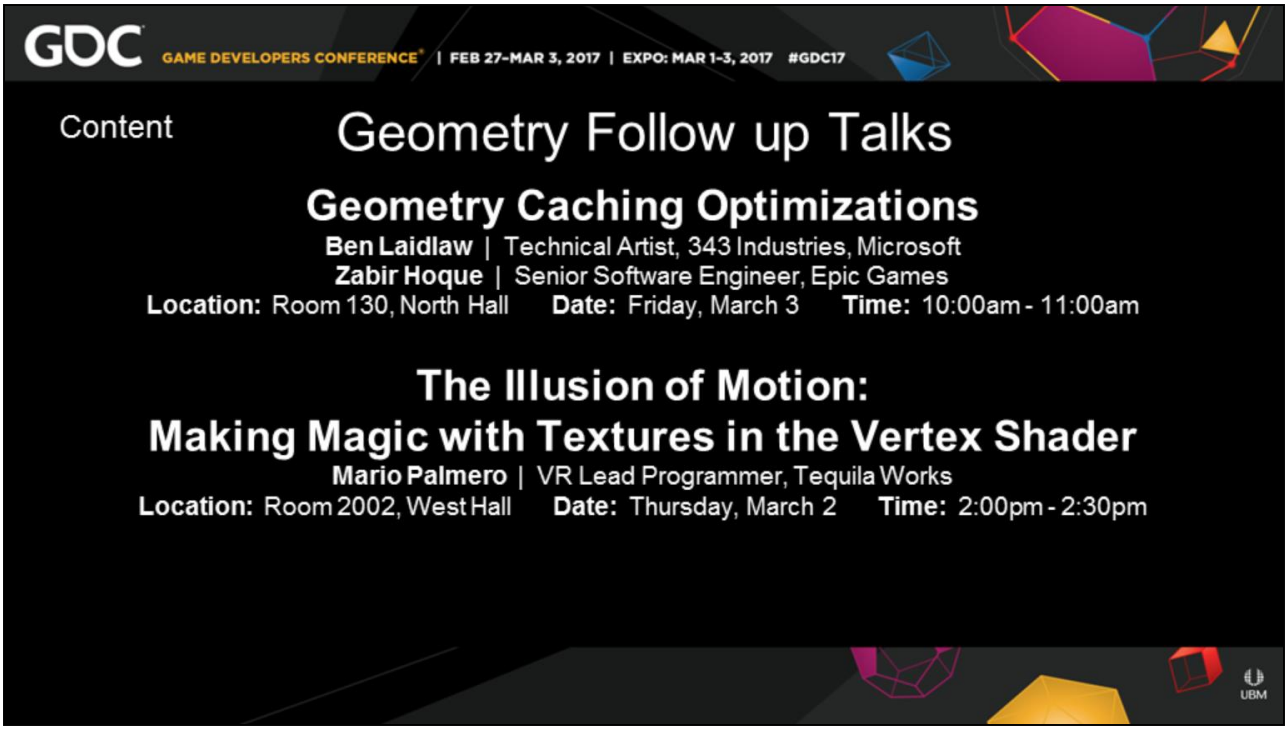
First of we have your type of **content** to focus on.

How to prepare that content for your **simulation** of choice.

How to best set up the **infrastructure** for that sim.

How to **massage** the data,
and reveal its inner beauty.

And then how to **maintain** that data
For a happy and long relationship.



- 46:50, 00:45, 11:25

I'm going to **recommend** two talks right now this week,
if moving geometry does interest you.

If you want to learn how for Halo 5 my team got over a million transforms at 60FPS
In order to play those simulations in real-time
Please come to Zabir and mine talk on
Geometry Caching Optimizations on Friday at 10am in the North Hall.
This is a bit more programing and engine than this

talk.
Zabir will show a lot of details our runtime compression method.

I would also highly recommend as a precursor talk to
Mario Palmeron and Norman's talk on "**The Illusion of Motion**"
It was work down in unknown parallel to ours.
But it is a great introductory to core concepts prior to us cranking that technology to 11.

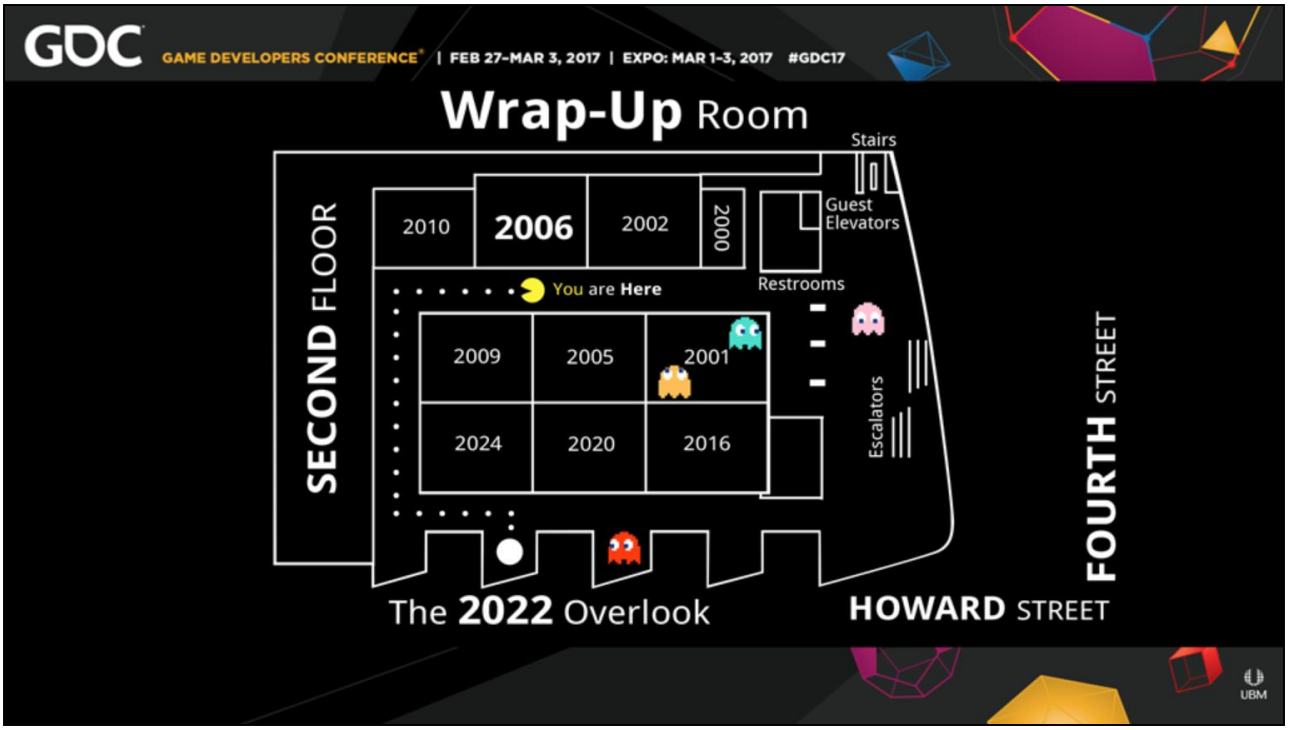
- 47:35, 00:40, 10:45

Since this talk was an overview of simulated content,
And if you want to dive in and start simulating content today,
Here are a few **recommended** jump starts.

As far as access to the tools I have used in production,
You can get them directly from inside Houdini
Via http://www.**orbolt.com**/search/?q=LaidlawFX

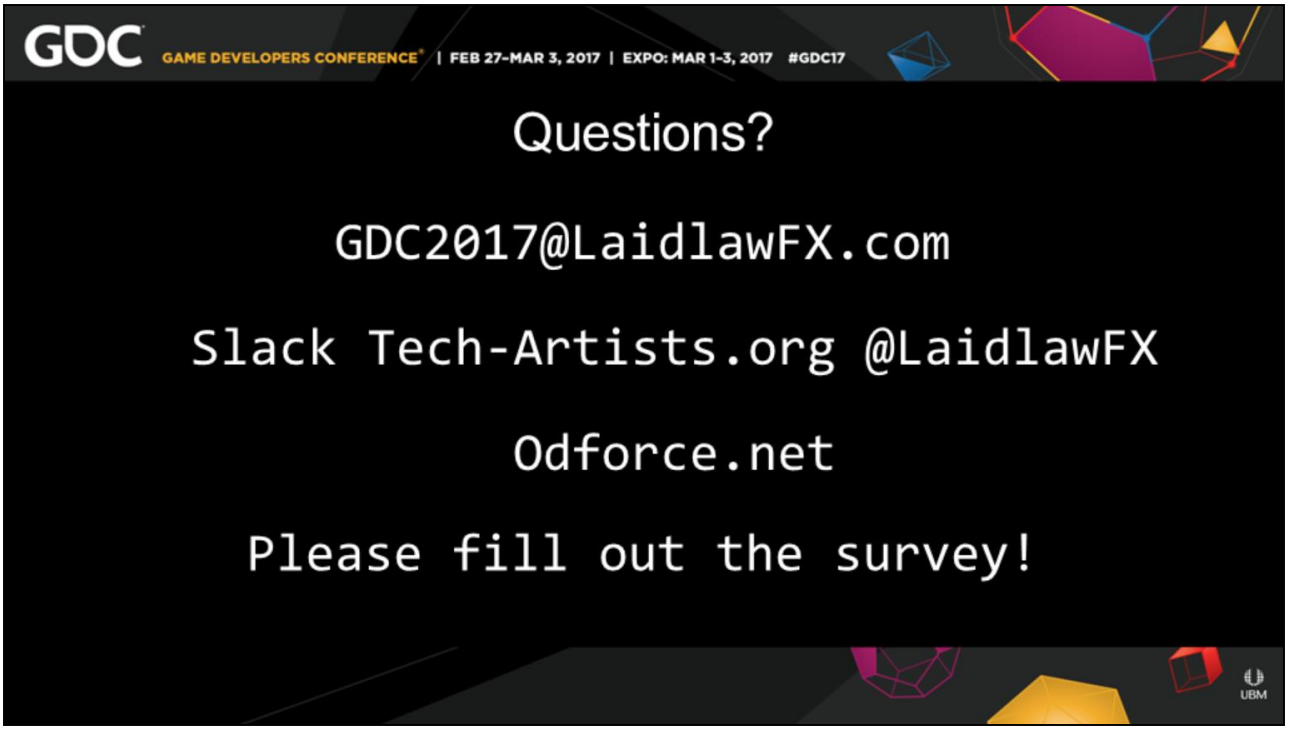
In addition with the launch of **Houdini 16**
A lot of games setups have been fully integrated into

Houdini,
Meaning you get direct support from SideFX now.
For any additional production help with Houdini stop by the **SideFX booth**,
They can gladly point you towards specific tutorials related to simulated content.



- 48:15, 00:15, 10:30

So we are about to break,
So if you do not want to get up in front of the room on the microphone
And you want to ask a question after this talk
Please join me in the **wrap up room** down the hall.



- 48:30, 00:40, 09:50

I'm also available after GDC to communicate with through a few methods.
First via **Slack** through Tech-Artists.org @LaidlawFX
Or through **Odforce.net** if you like the forum format.
Or you can even **e-mail** me at GDC2017@LaidlawFX.com

Thank you all for coming!

And I hope if you enjoyed the talk.

And if you did, Please fill out the **Survey**.
I and all the speakers would greatly appreciate it.

I am happy to answer any questions now
And for the next few minutes.