Hello

My name is Dain Saint.

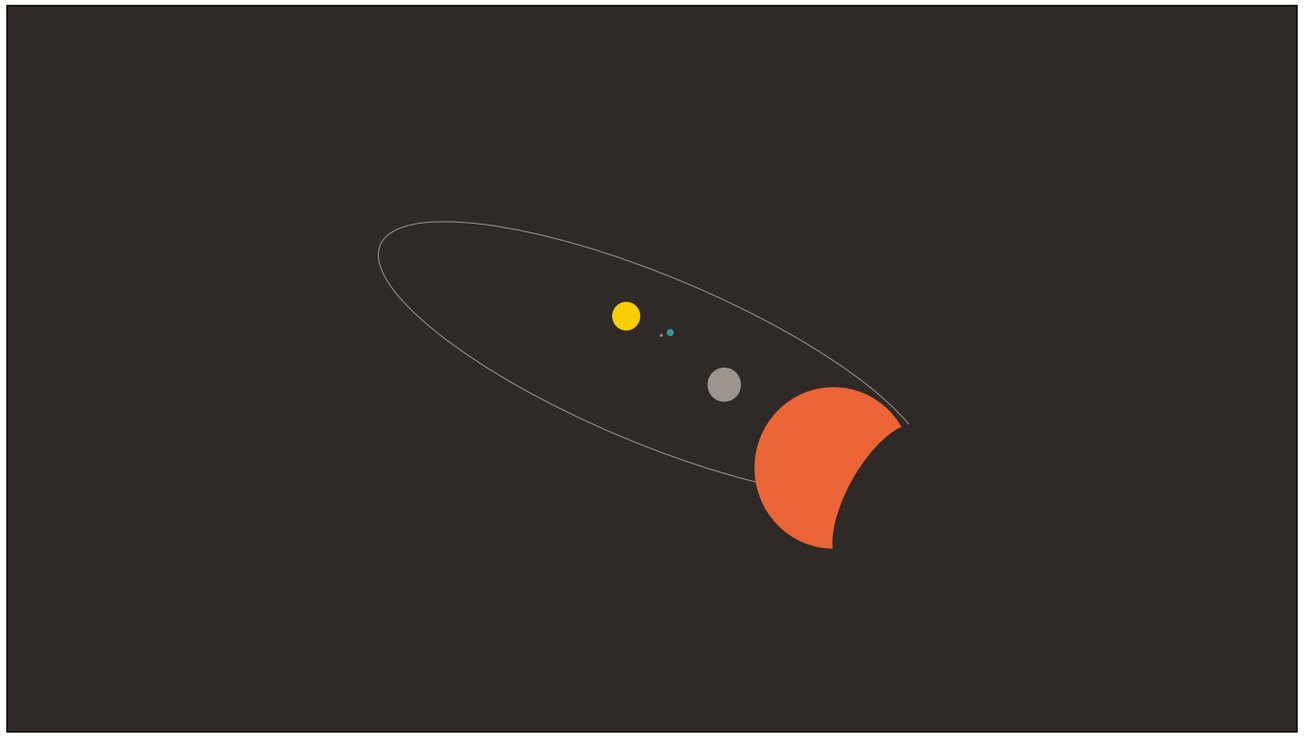
I’m the co-founder of Cipher Prime Studios, best known for Auditorium, Splice, and Intake.

I’m one third of Deep Dark Hole, an experimental games project that’ll be showing off a game called Elbow Room at Wild Rumpus tomorrow

And I’m a director of the Philly Game Forge — Philadelphia’s first and only co-working space devoted exclusively to games.

This talk is about asking the right questions, and getting answers from the natural world. It might be unusual, but I’m hoping it will inspire you to look at things a little differently.

So, without further ado:

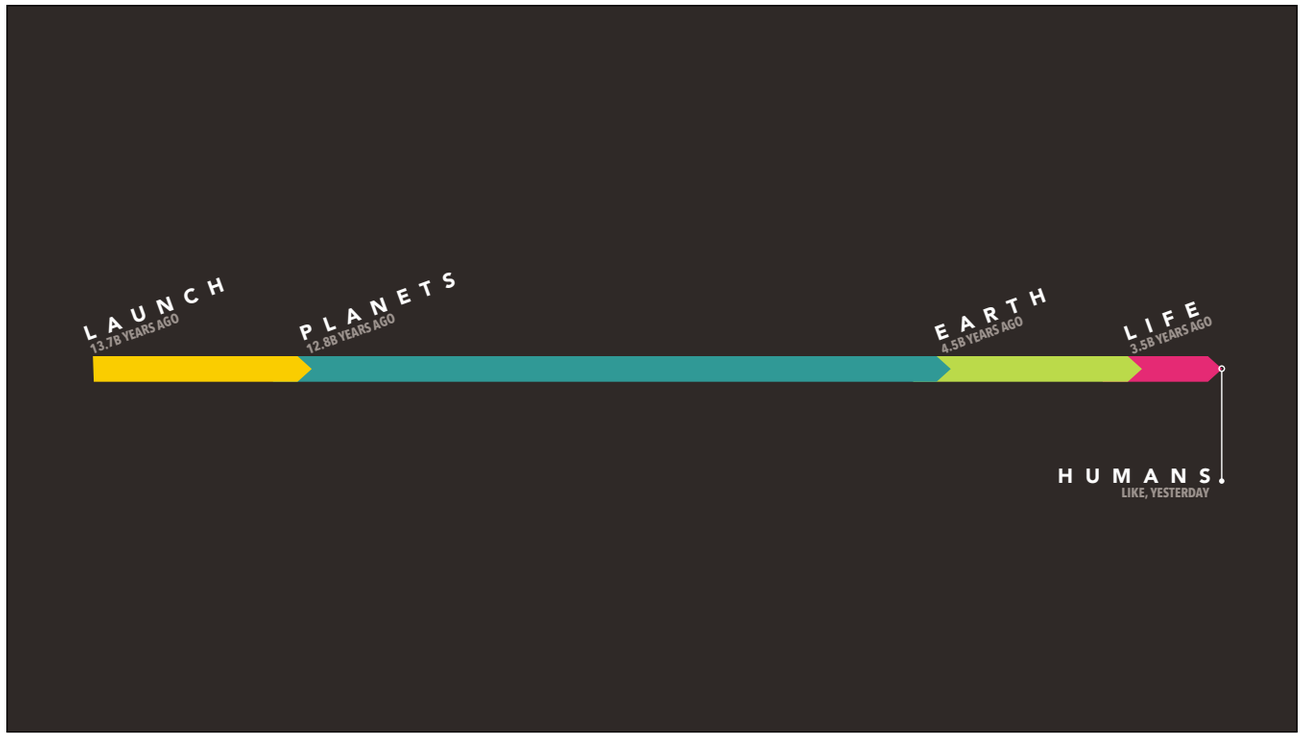


The Universe.

Experiments are currently underway to determine whether our reality is, in fact, a simulation. But what does it mean if it is? I believe that, as game designers, we can answer that question.


THE UNIVERSE
HOW TO BREAK THE GAME

Let’s start at the beginning.

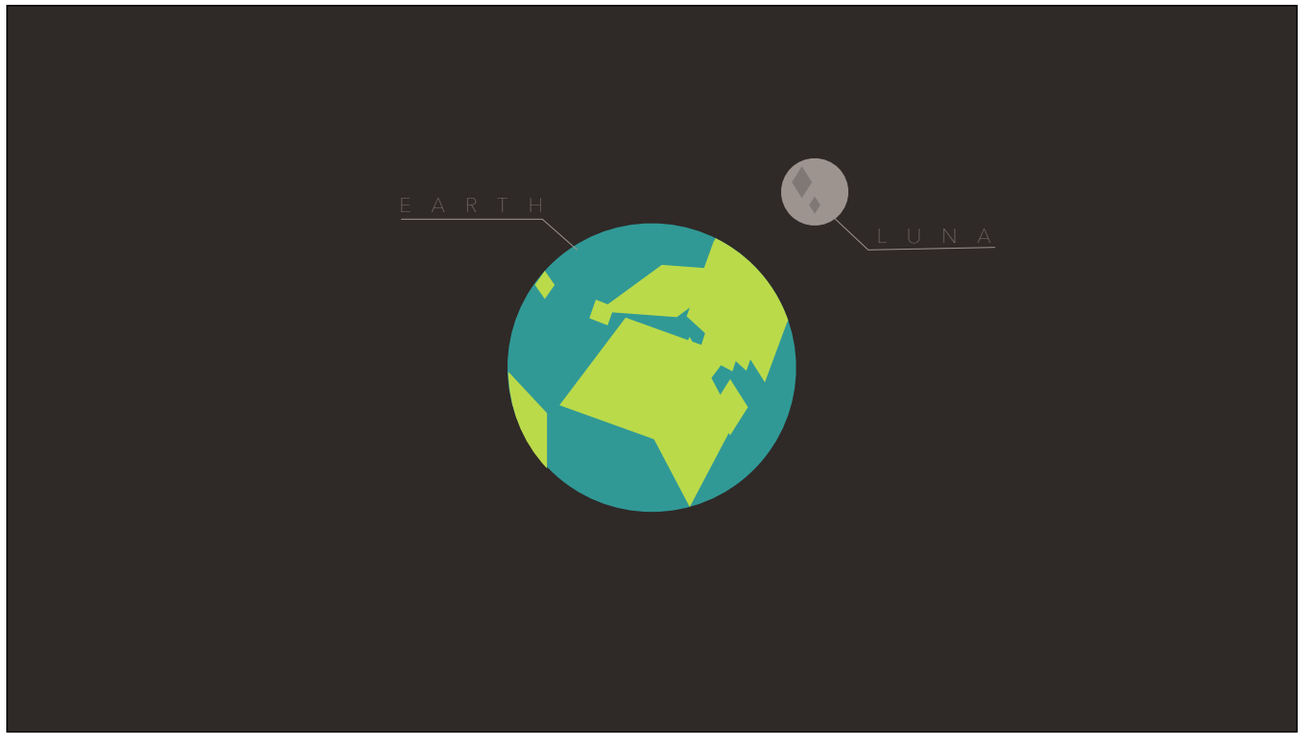


To the best of our knowledge, the Universe hit the market around 13.7 billion years ago.

Now, since humans only started playing it around 200,000 years ago, the early years of the Universe remain somewhat of a mystery.

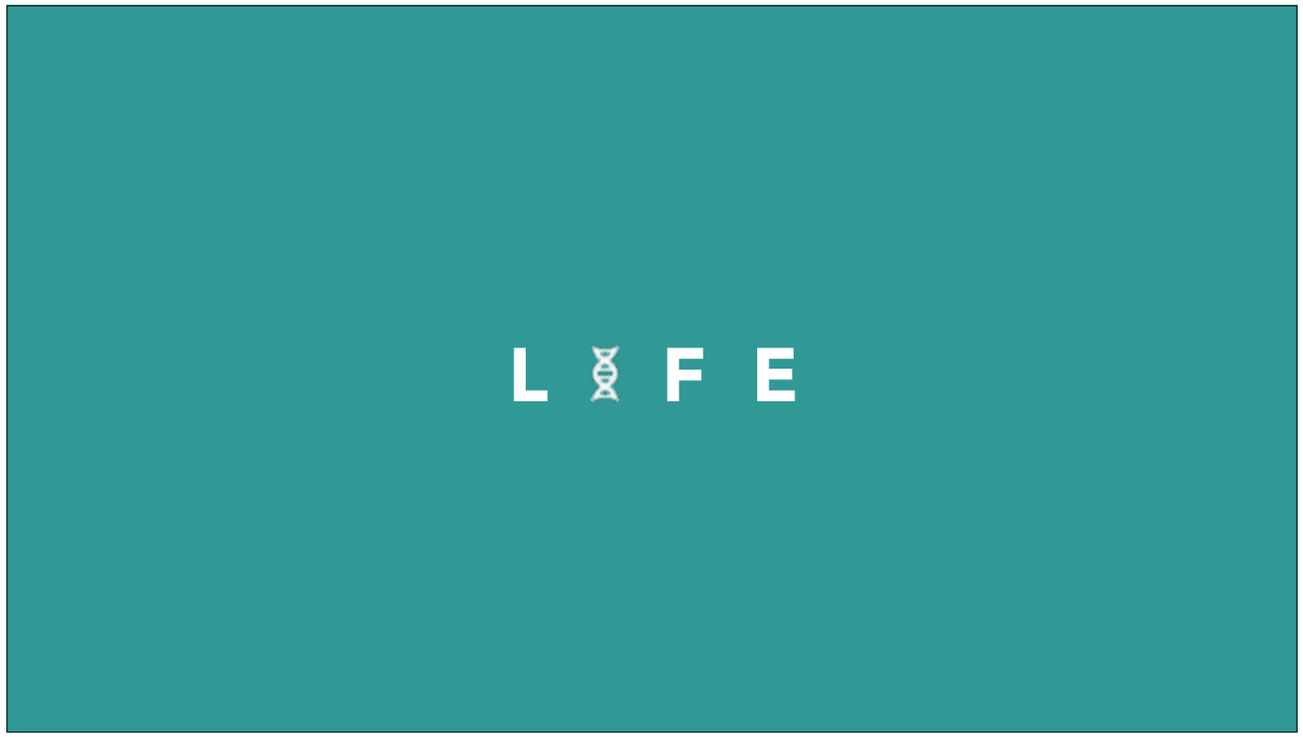
As near as we can tell, the Universe spent about the first two-thirds of its product cycle in Beta, functioning as a sort of planetary-creation game

until the release of the Earth expansion 4.5 billion years ago. This expansion added the killer feature that would change games forever:

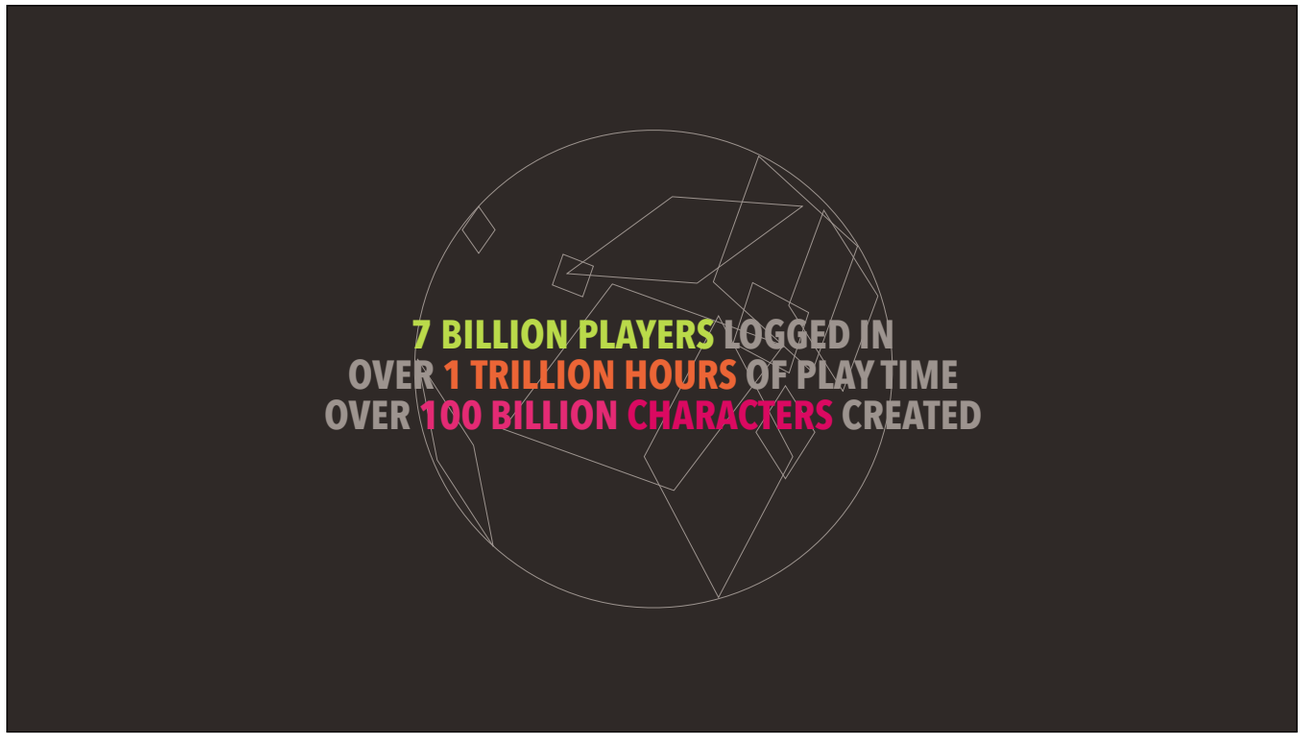


As near as we can tell, the Universe spent about the first two-thirds of its product life-cycle in Beta, functioning as a sort of planetary-creation game

until the release of the Earth expansion 4.5 billion years ago. This expansion added the killer feature that would change games forever:



Life. Though early characters were primitive, this laid the groundwork for exciting new modes of play. Fast forward to today, and the Universe has dominated the market.



Counting humans alone, the Universe has 7 billion players logged in as we speak — despite having no visible monetization strategy.

And since the beginning of the Earth expansion, over 1 trillion hours of playtime have been logged by a 100 billion characters using what has to be the most fun character-creation engine on the market.

And it’s not hard to see why.

Counting humans alone, the Universe has 7 billion players logged in as we speak — despite having no visible monetization strategy.

And since the beginning of the Earth expansion, over 1 trillion hours of playtime have been logged by over 100 billion characters using what has to be the most fun character-creation engine on the market.

And it’s not hard to see why.

The Universe is impossibly good — “Joystiq 5 stars” good. We’re talking about an always on, massively multiplayer experience with zero lag and no downtime.

The Universe is impossibly good — “Joystiq 5 stars” good. We’re talking about an always on, massively multiplayer experience with zero lag and no downtime.

The Universe is impossibly good — “Joystiq 5 stars” good. We’re talking about an always on, massively multiplayer experience with zero lag and no downtime.

The Universe is impossibly good — “Joystiq 5 stars” good. We’re talking about an always on, massively multiplayer experience with zero lag and no downtime.

The Universe is impossibly good — “Joystiq 5 stars” good. We’re talking about an always on, massively multiplayer experience with zero lag and no downtime.

The Universe is impossibly good — “Joystiq 5 stars” good. We’re talking about an always on, massively multiplayer experience with zero lag and no downtime.

Plus, the Universe’s patented Infinity Engine is a technological marvel

The Universe’s patented Infinity Engine is a technological marvel,

boasting infinite resolution, infinite ray cast depth, infinite explorable space, infinitely destructible environments,

boasting infinite resolution, infinite ray cast depth, infinite explorable space, infinitely destructible environments,

and a Level-of-detail algorithm that puts Red Dead Redemption to shame.

boasting infinite resolution, infinite ray cast depth, infinite explorable space, infinitely destructible environments,

and a Level-of-detail algorithm that puts Red Dead Redemption to shame.

boasting infinite resolution, infinite ray cast depth, infinite explorable space, infinitely destructible environments,

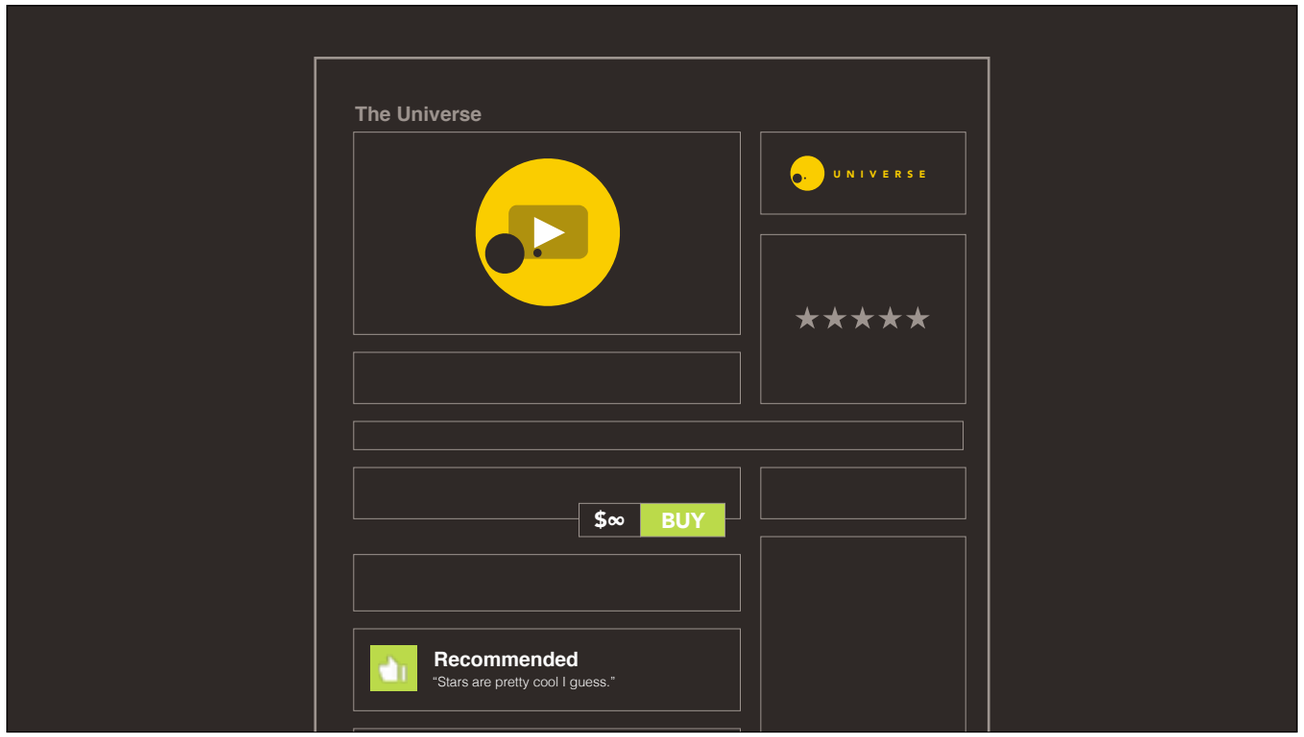
and a Level-of-detail algorithm that puts Red Dead Redemption to shame.

boasting infinite resolution, infinite ray cast depth, infinite explorable space, infinitely destructible environments,

and a Level-of-detail algorithm that puts Red Dead Redemption to shame.

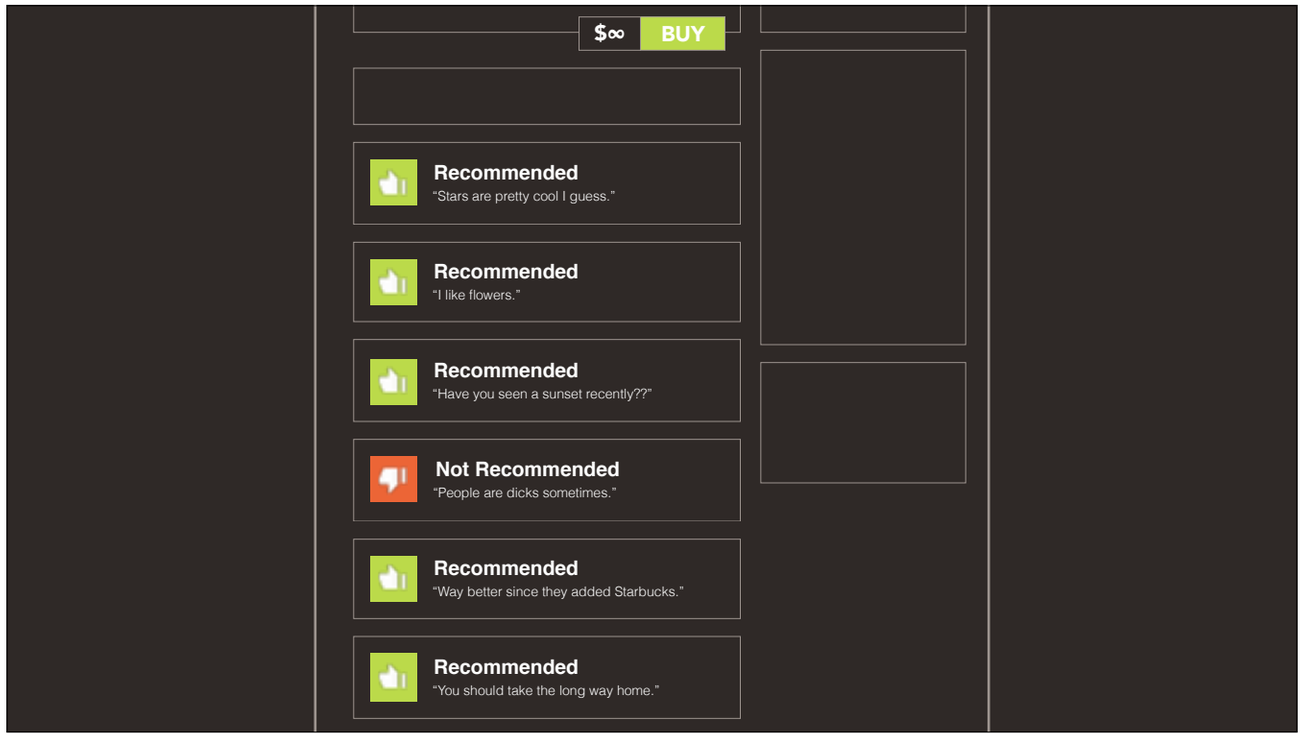
like the name says, it boasts infinite resolution, infinite ray cast depth, infinite explorable space, infinitely destructible environments,

and a Level-of-detail algorithm that puts Red Dead Redemption to shame.



With a game that does so many things right, it’s hard to believe there could be anything wrong.

Most of the player complaints are lodged against the behaviour of other players, not the game itself.



With a game that does so many things right, it’s hard to believe there could be anything wrong.

Most of the player complaints are lodged against the behaviour of other players, not the game itself.

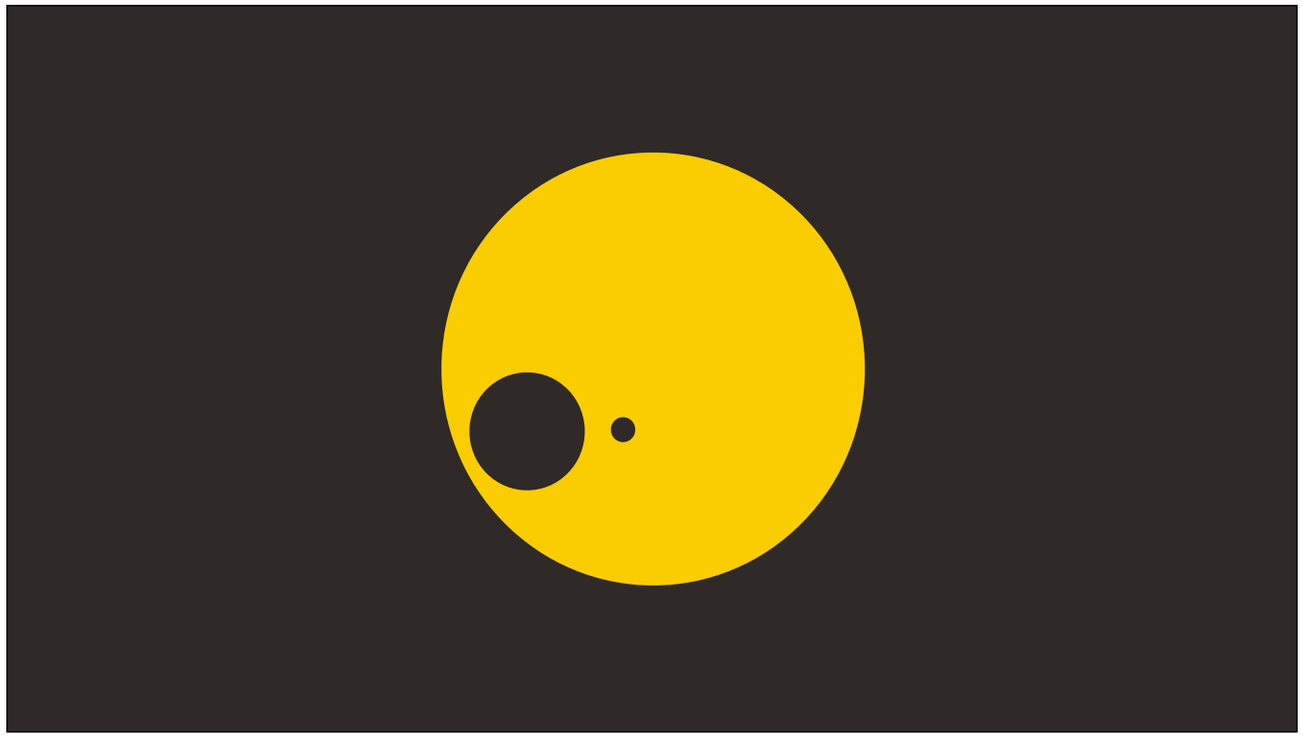
But when you look closely — I mean really closely — you can start to see the seams.

There’s a global speed limit — things can only go so fast, no matter how hard you try.

That infinite draw distance? Turns out it’s not so infinite.

Some pretty major precision bugs start to pop up when things get too small

And the physics system goes absolutely berserk you try to push too much stuff together in one place.



Despite all of this, the Universe is a seamless experience that many of us enjoy every day. So how does one make something this good?

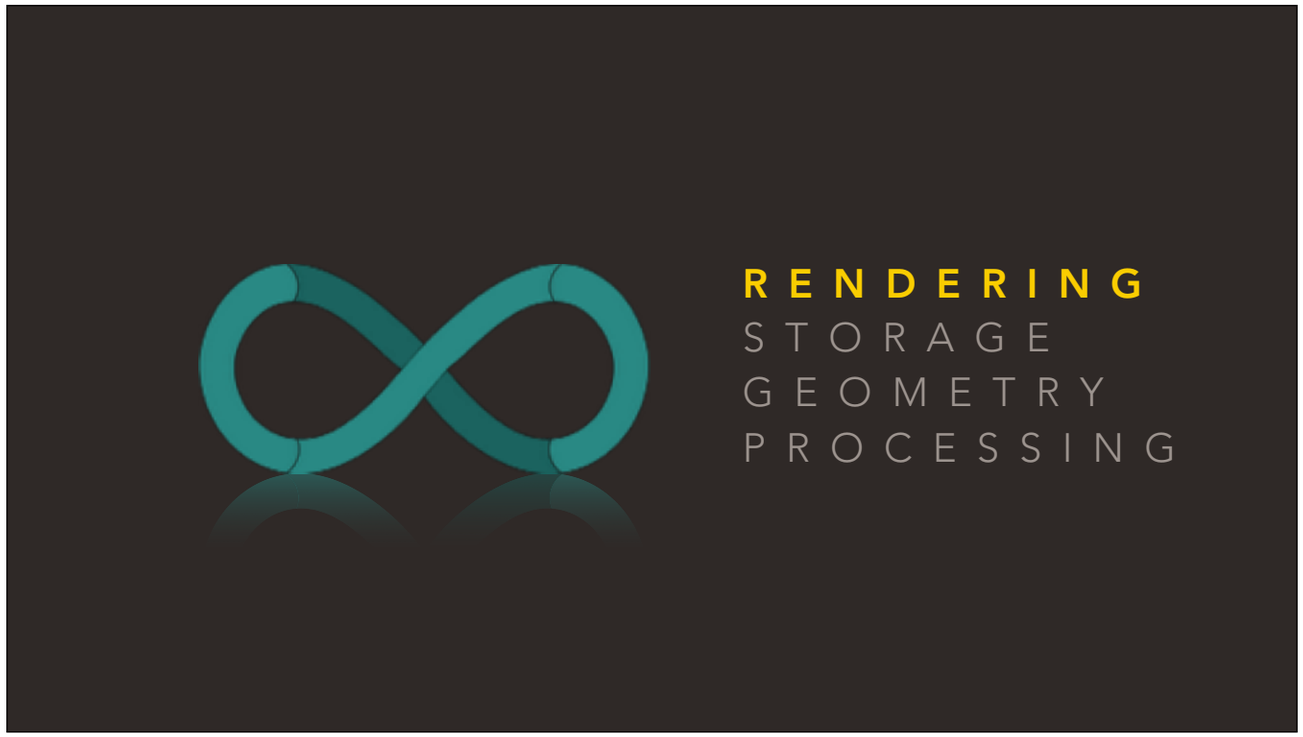
Although we've sent countless emails to the developers, we still haven't gotten a definitive response.

So to answer that question, we're going to have to break the game.

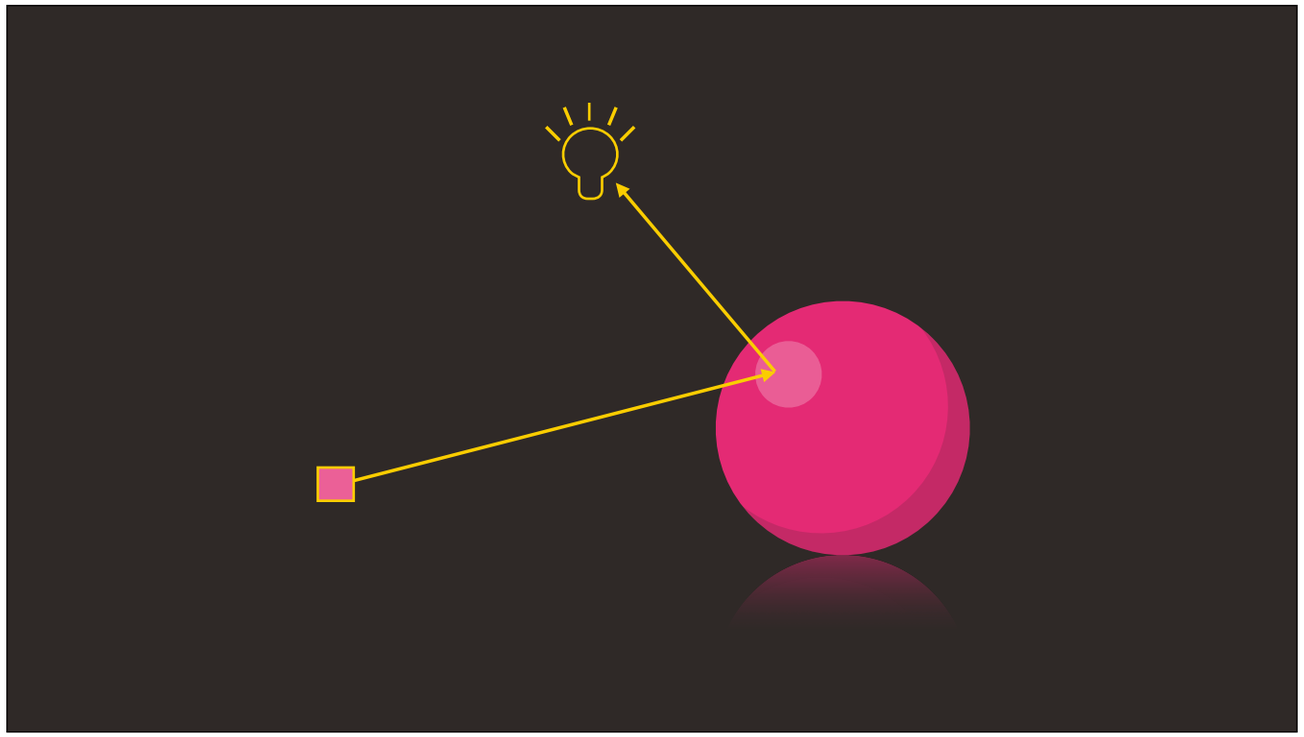
PART TWO

BREAK IT

To start hacking at the game, we have to first understand the limitations of the Infinity Engine. It seems natural to start with the rendering pipeline, so lets start there.



To start hacking at the game, we have to first understand the limitations of the Infinity Engine. It seems natural to start with the rendering pipeline, so lets start there.



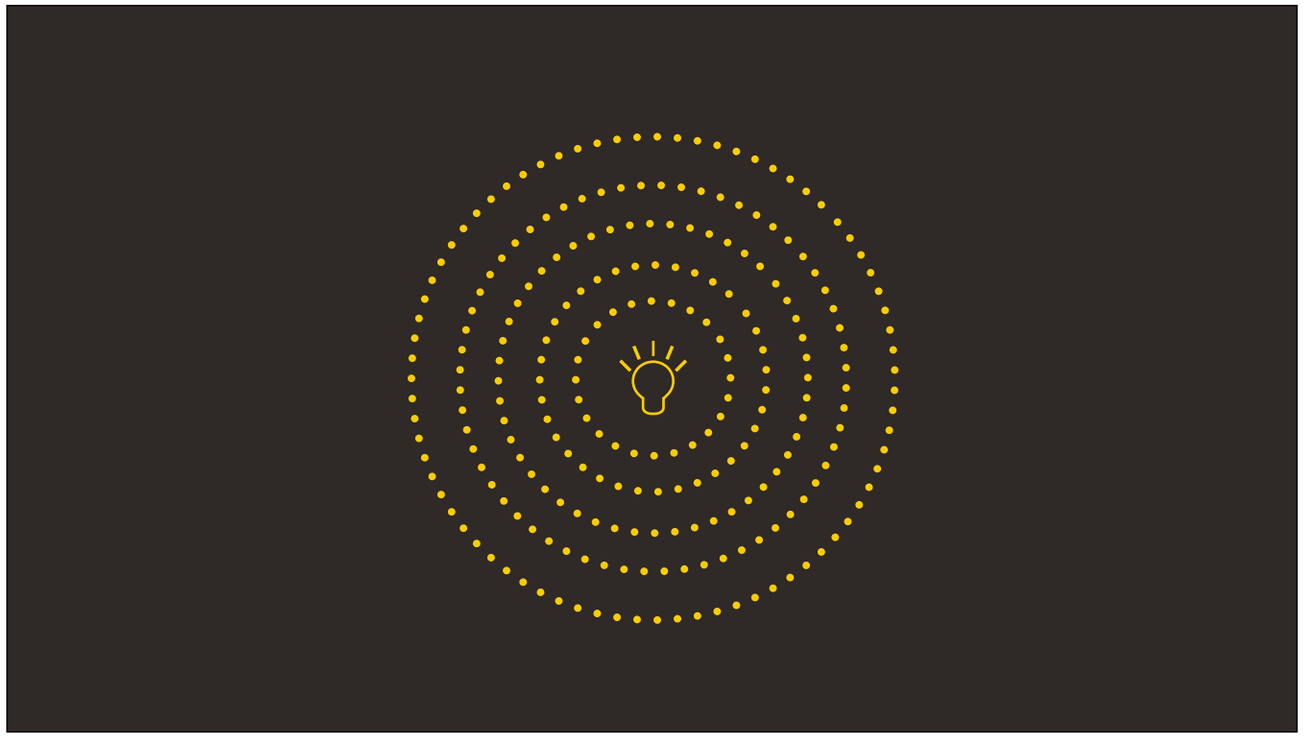
As you may know, traditional engines utilise one of a few variations on a technique called “Raycasting”, where a ray is sent out from each pixel that needs to be rendered, bouncing off objects until it reaches a light source.

The Infinity Engine turns this paradigm upside down, using a proprietary system called Photon.

The Infinity Engine turns this paradigm upside down, using a proprietary system called Photon.

In this, particles of light, called “photons”, are actually sent from out the light source, accumulating color data as it goes along it’s way.

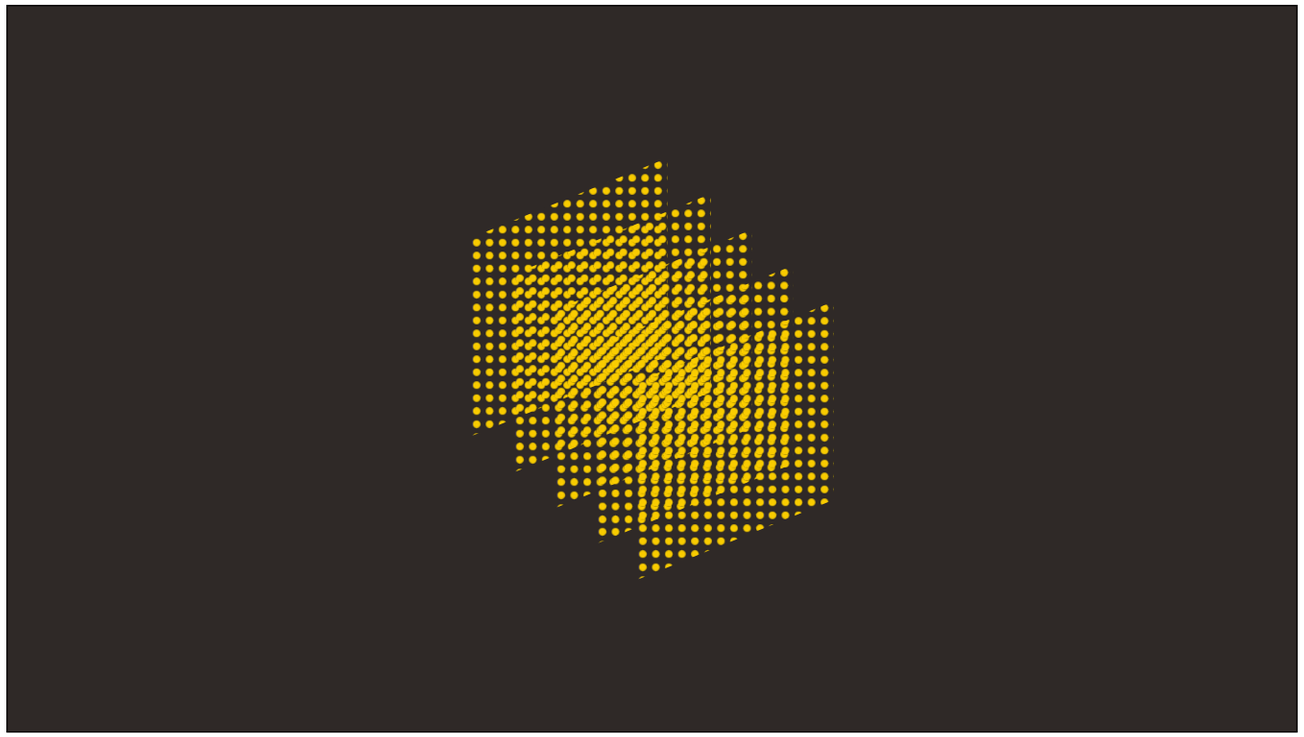
The volume is filled with photons, and images are constructed by querying the photons that exist in a particular cross-section.



The Infinity Engine turns this paradigm upside down, using a proprietary system called Photon.

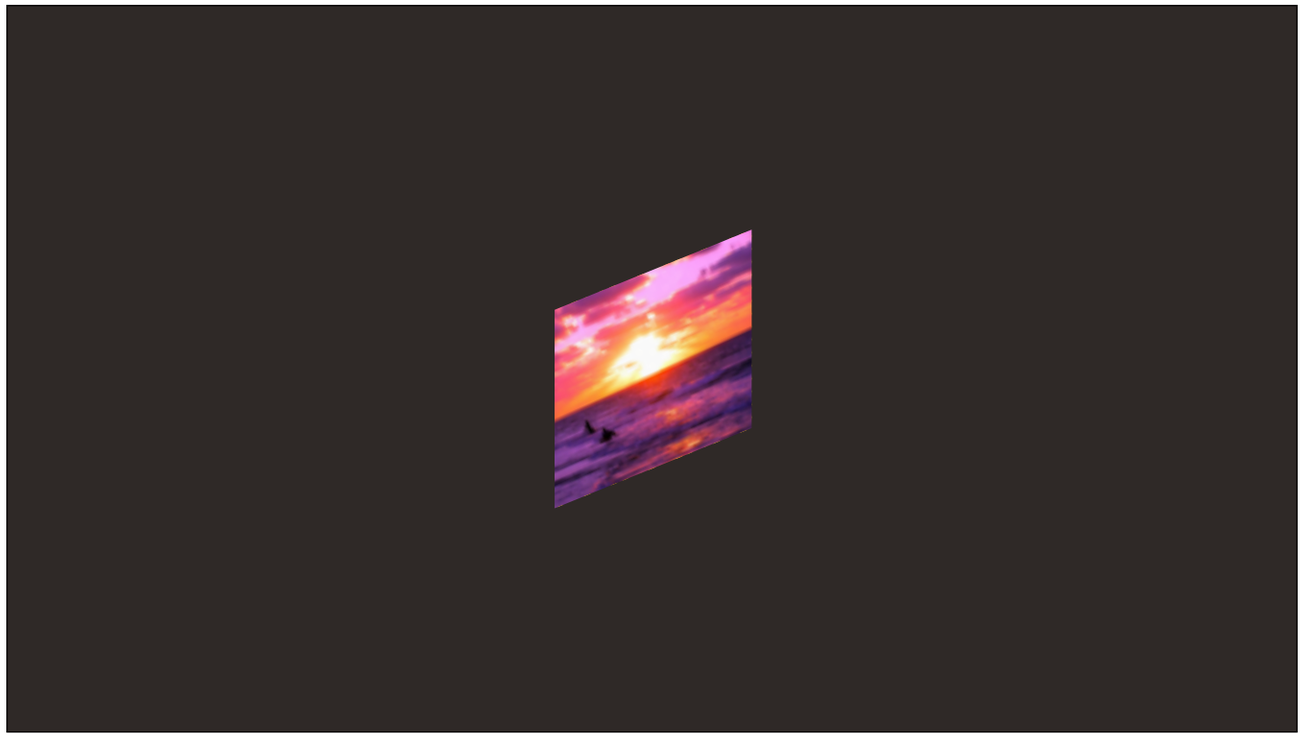
In this, actual particles of light, called “photons”, are sent from out the light source, accumulating color data as it goes along it’s way.

The volume is filled with photons, and images are constructed by querying the photons that exist in a particular cross-section.



The volume is filled with photons, and images are constructed by querying the photons that exist in a particular cross-section.

Because the images rendered are time-dependant, the photons themselves have a speed, referred to as “c” — roughly 300 million meters per second.



The volume is filled with photons, and images are constructed by querying the photons that exist in a particular cross-section.

Because the images rendered are time-dependant, the photons themselves have a speed, referred to as “c” — roughly 300 million meters per second.

Because the images rendered are time-dependant, the photons themselves have a speed, referred to as “c”, the speed of light — roughly 300 million meters per second.

Now this is where it gets interesting. Photon is completely dependent on this constant — which means we’ve found the first seam we can start hacking at.

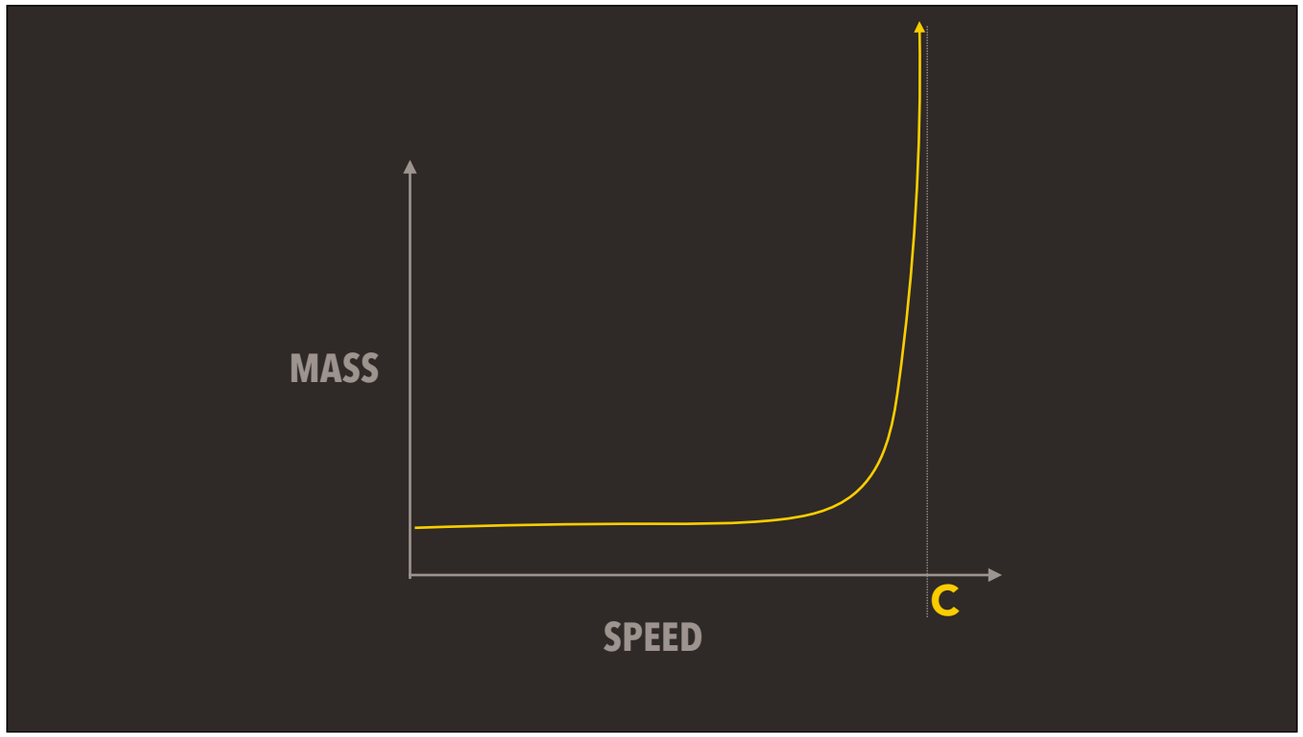
If the maximum refresh rate of the universe is limited by this number, what happens when we go faster than it?

Think about Quake — players have been using bunny-hopping to go faster than the “max velocity” for so long that it’s essential to high-level play.

So what happens when we try to go MLG-Pro by surpassing the speed of light?

Well? It turns out — you can’t. Nothing can. No matter how much energy you use, nothing in the Infinity Engine will ever go faster than the speed of light. Hell, even getting *close* is damn near impossible.

What gives?



What gives? When we throw out a few traces to see whats going on, the result is bizarre: as an object approaches the SPEED of light, it GETS heavier and heavier. This is really strange behaviour, I've never seen a game do this before, but its actually a pretty elegant hack! The heavier something is, the harder you have to push for it to gain speed.

So by making objects heavier the faster they get, you get a smooth way of preventing something from hitting that MAX velocity cap.

AND you only start to really notice the effects at near-light speeds.

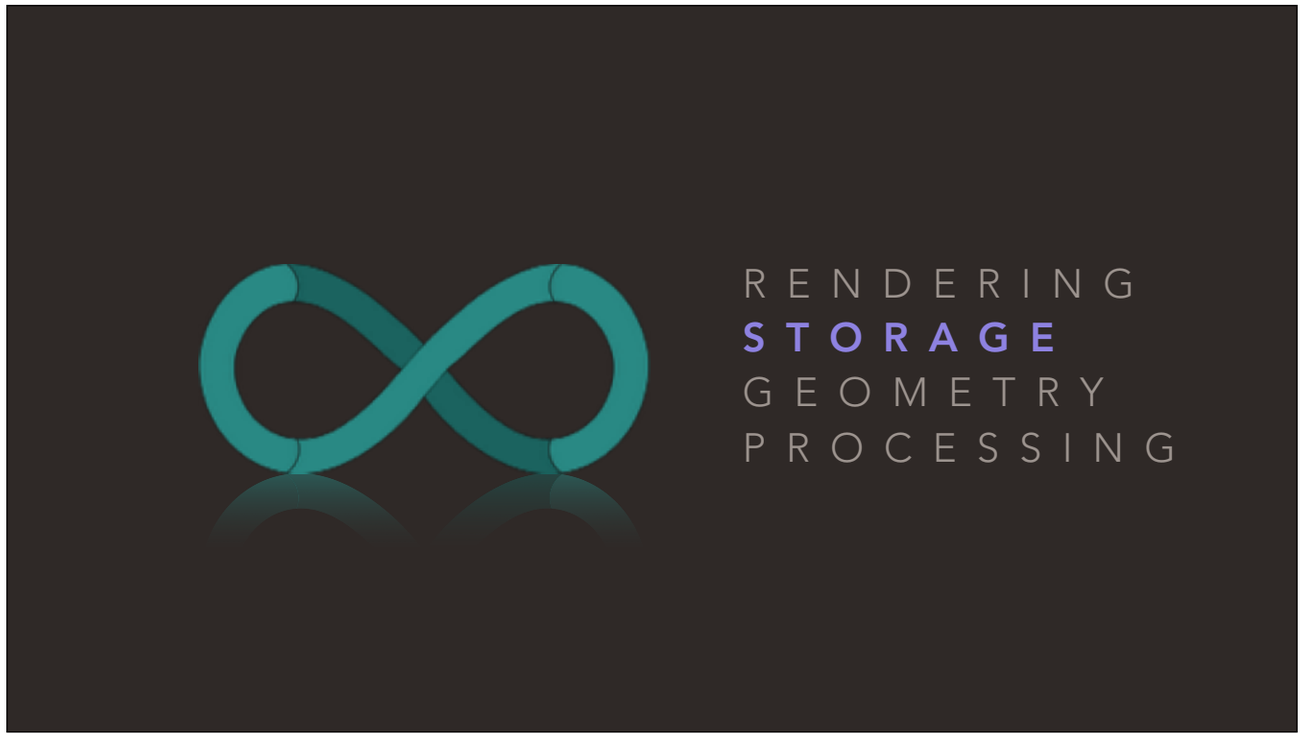
Kudos to the devs for that one.

AND you only start to really notice the effects at near-light speeds.

Kudos to the devs for that one.

Now, the fact that light has a speed is also partially responsible for another peculiarity of the Infinity Engine — the maximum draw distance.

Let's look at the storage engine to see why.

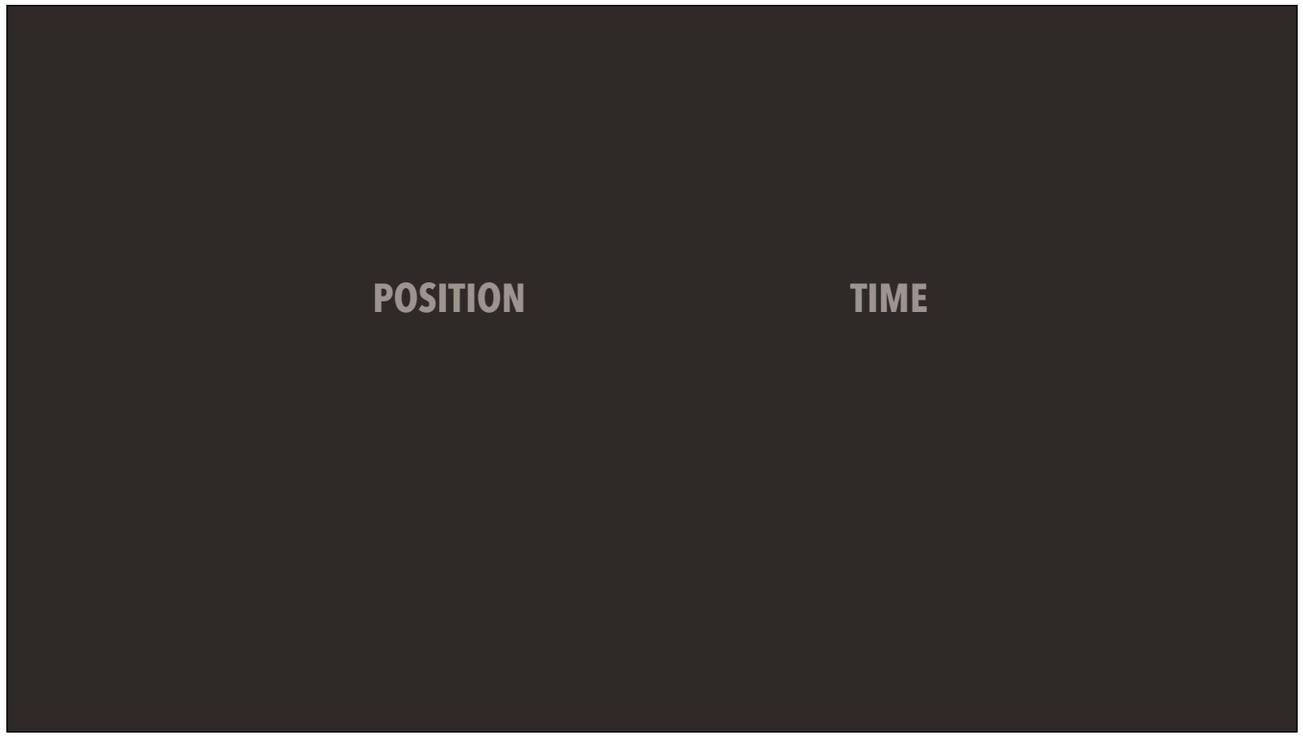


Now, the fact that light has a speed is also partially responsible for another peculiarity of the Infinity Engine — the maximum draw distance.

Let’s look at the STORAGE engine to see why.

In games, you need to be able to specify both WHERE a thing is, and WHEN it is occurring. Typically, engines store these as two variables — POSITION and TIME. 2D games store position as a VECTOR2, and 3D games store it as a VECTOR3. TIME is usually kept as a separate variable, and managed by a different system altogether.

Conversely, the Infinity Engine stores both position and time TOGETHER in a single Vector4 using it's proprietary storage MODULE , appropriately named "Spacetime".

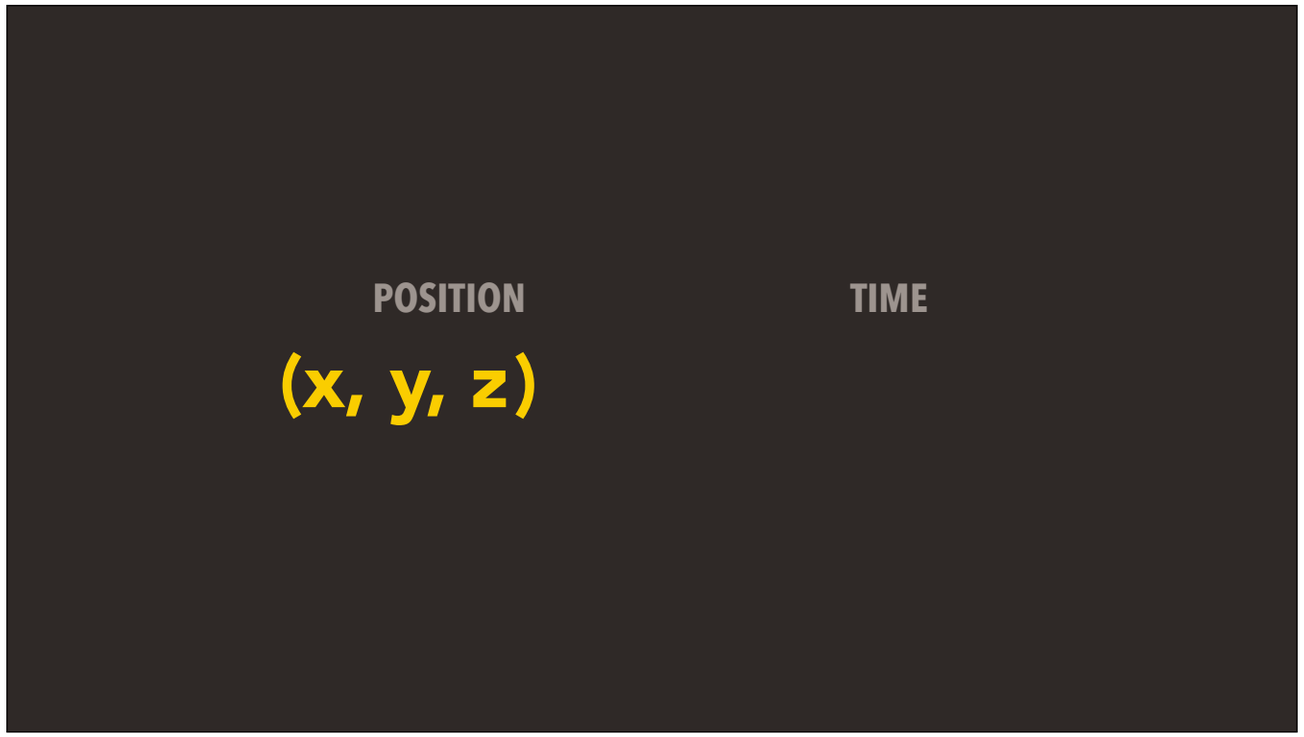


In games, you need to be able to specify both WHERE a thing is, and WHEN it is occurring. Typically, engines store these as two variables — POSITION and TIME. 2D games store position as a VECTOR2, and 3D games store it as a VECTOR3. TIME is usually kept as a separate variable, and managed by a different system altogether.

Conversely, the Infinity Engine stores both position and time TOGETHER in a single Vector4 using it's proprietary storage MODULE , appropriately named "Spacetime".

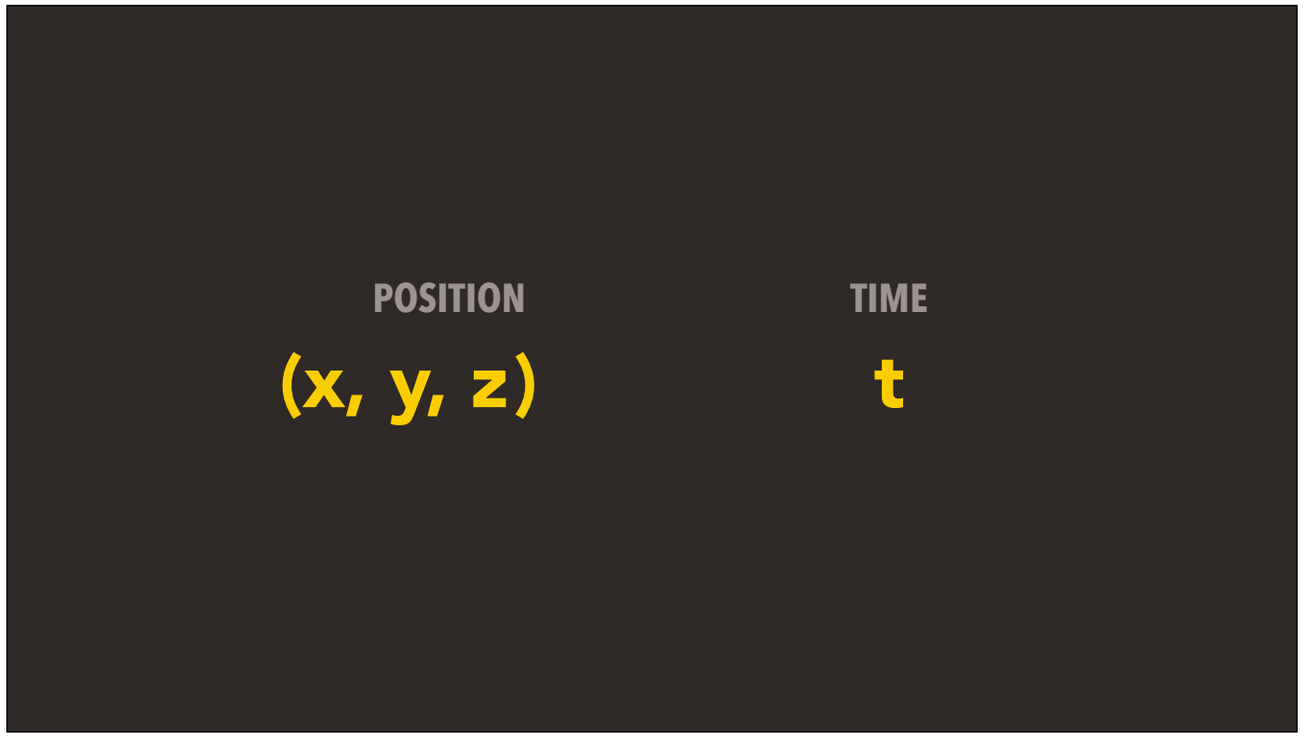
In games, you need to be able to specify both WHERE a thing is, and WHEN it is occurring. Typically, engines store these as two variables — POSITION and TIME. 2D games store position as a VECTOR2, and 3D games store it as a VECTOR3. TIME is usually kept as a separate variable, and managed by a different system altogether.

Conversely, the Infinity Engine stores both position and time TOGETHER in a single Vector4 using it's proprietary storage MODULE , appropriately named "Spacetime".



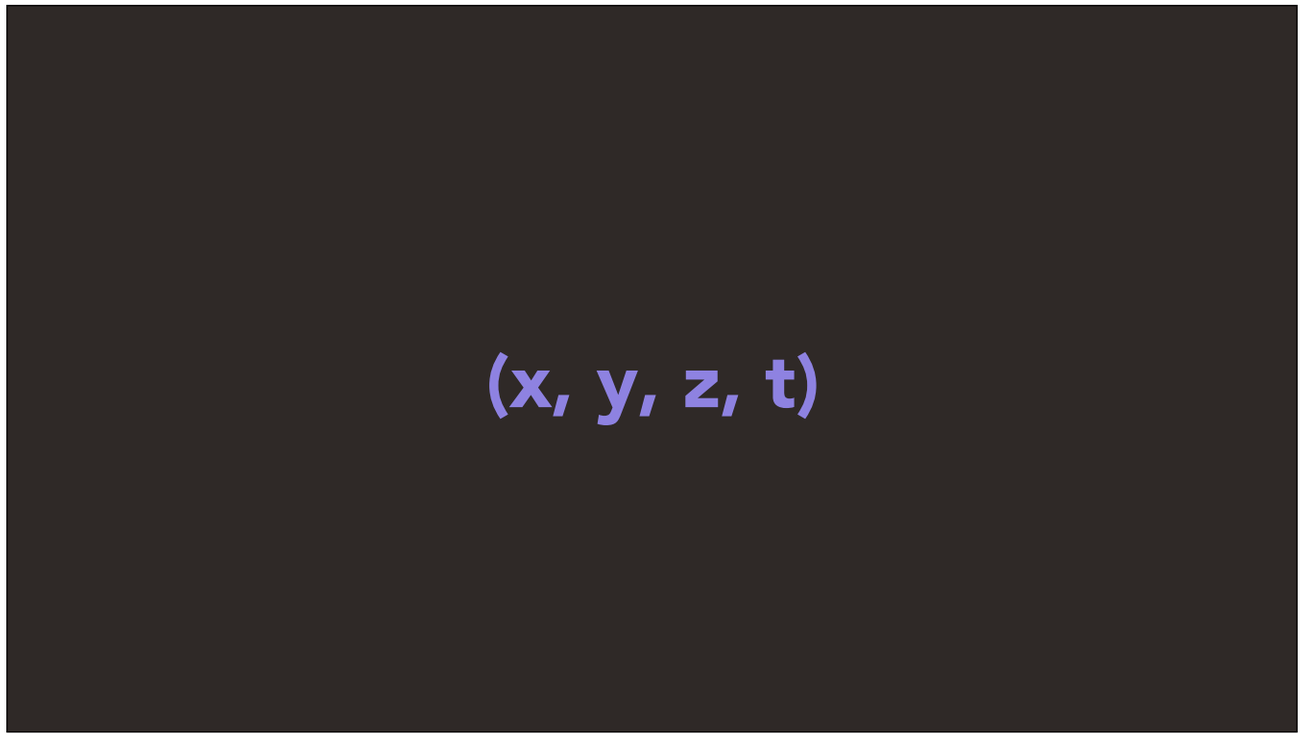
In games, you need to be able to specify both WHERE a thing is, and WHEN it is occurring. Typically, engines store these as two variables — POSITION and TIME. 2D games store position as a VECTOR2, and 3D games store it as a VECTOR3. TIME is usually kept as a separate variable, and managed by a different system altogether.

Conversely, the Infinity Engine stores both position and time TOGETHER in a single Vector4 using it's proprietary storage MODULE , appropriately named "Spacetime".



In games, you need to be able to specify both WHERE a thing is, and WHEN it is occurring. Typically, engines store these as two variables — POSITION and TIME. 2D games store position as a VECTOR2, and 3D games store it as a VECTOR3. TIME is usually kept as a separate variable, and managed by a different system altogether.

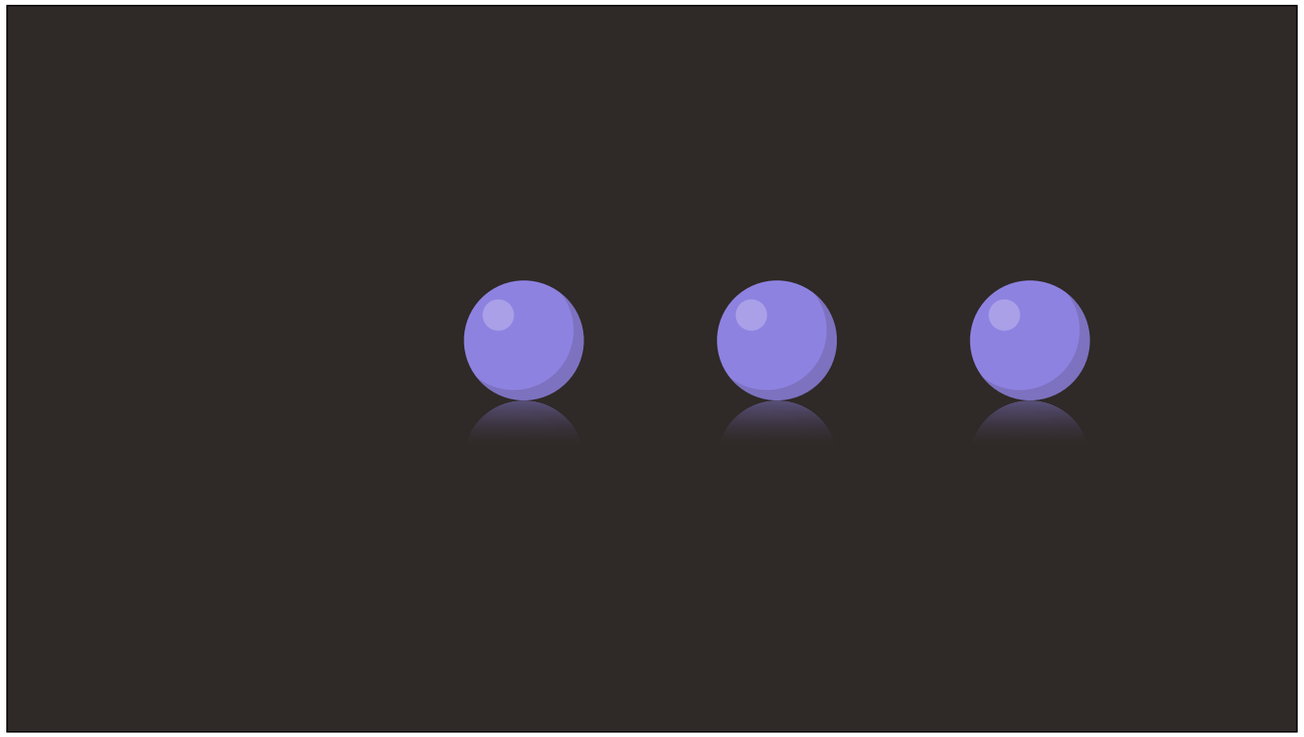
Conversely, the Infinity Engine stores both position and time TOGETHER in a single Vector4 using it's proprietary storage MODULE , appropriately named "Spacetime".



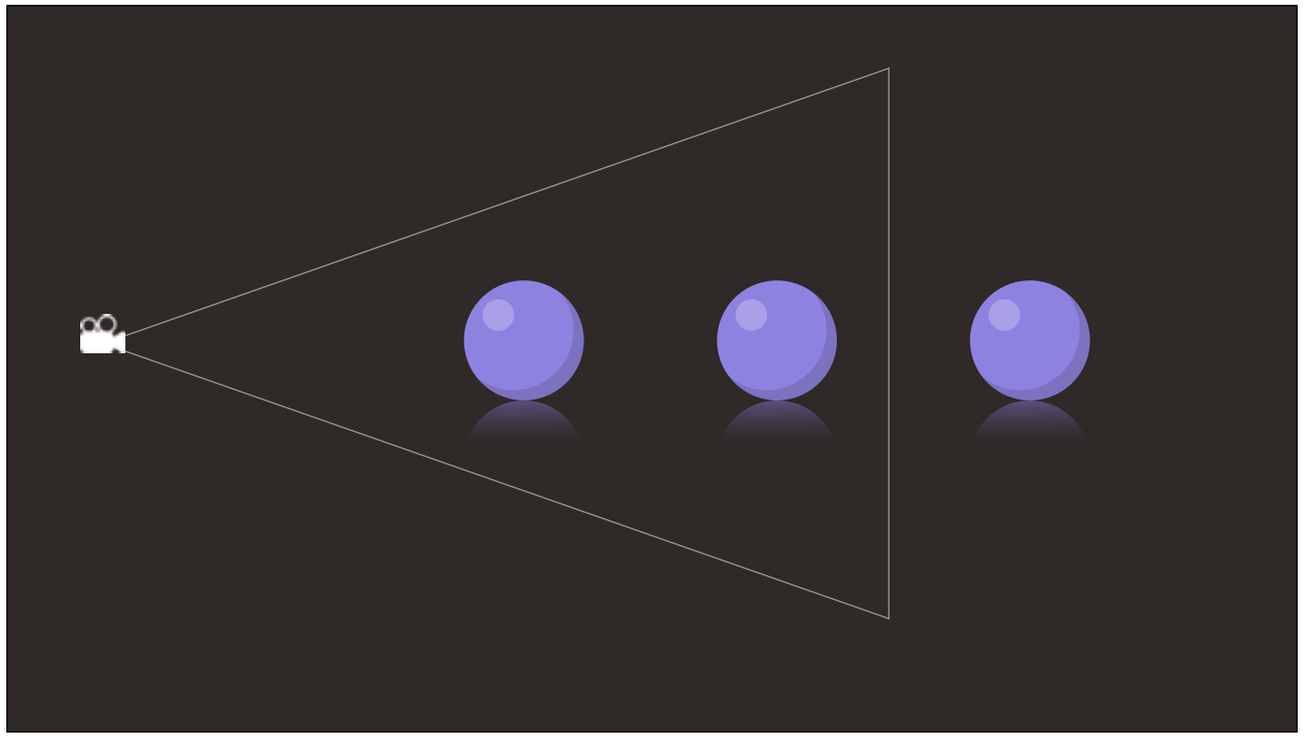
In games, you need to be able to specify both WHERE a thing is, and WHEN it is occurring. Typically, engines store these as two variables — POSITION and TIME. 2D games store position as a VECTOR2, and 3D games store it as a VECTOR3. TIME is usually kept as a separate variable, and managed by a different system altogether.

Conversely, the Infinity Engine stores both position and time TOGETHER in a single Vector4 using it's proprietary storage MODULE , appropriately named "Spacetime".

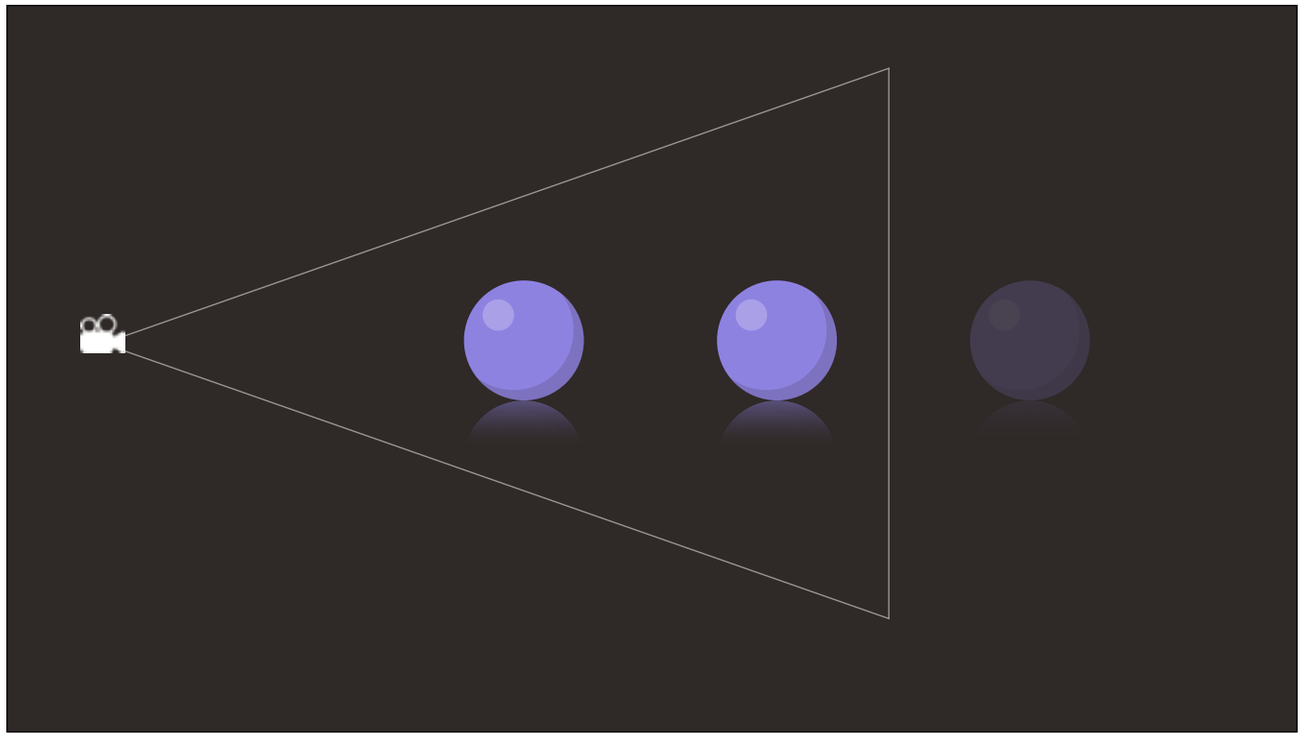
Conversely, the Infinity Engine stores both position and time together in a single Vector4 using it’s proprietary storage module, appropriately named “Spacetime”.



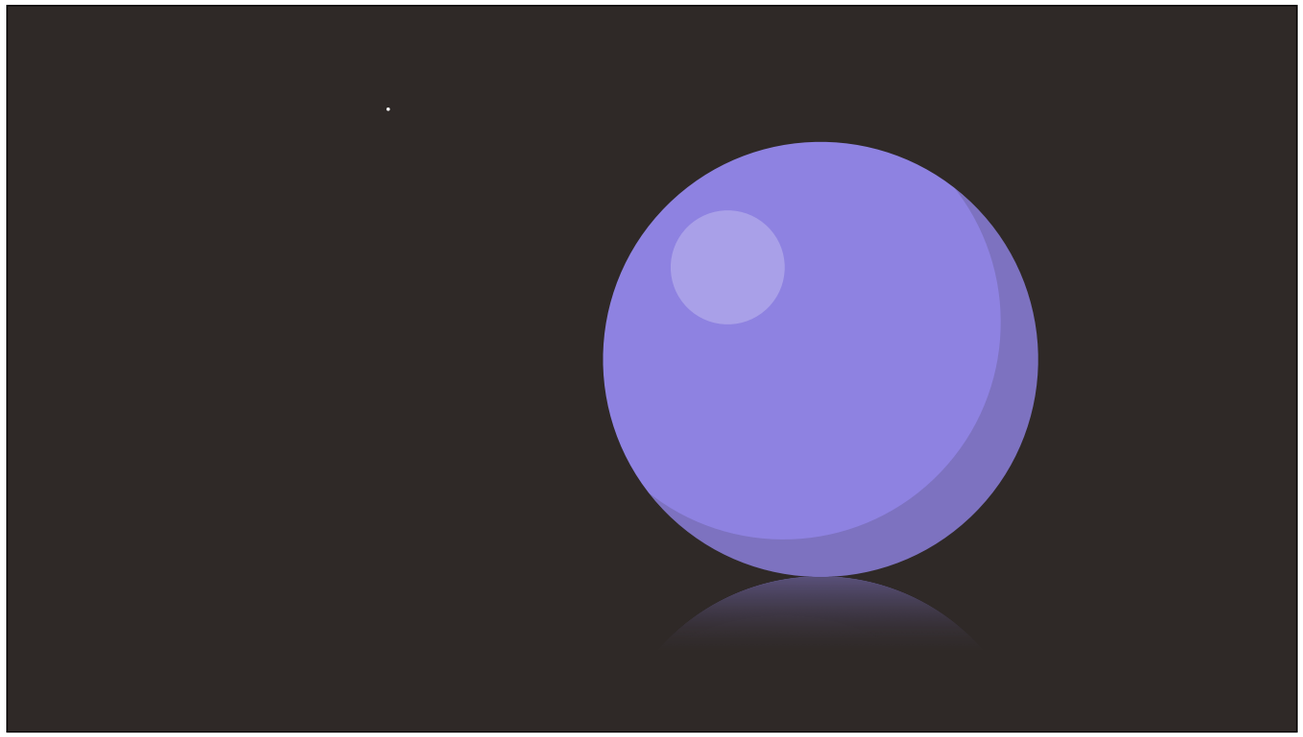
Now, most games have a culling distance — items that are sufficiently far from the camera are simply ignored by the renderer. This is done to speed up rendering. But as we mentioned earlier, Photon doesn't work like most renderers — all possible viewing paths have to be ready-to-observe by any player at any time.



Now, most games have a culling distance — items that are sufficiently far from the camera are simply ignored by the renderer. This is done to speed up rendering. But as we mentioned earlier, Photon doesn't work like most renderers — all possible viewing paths have to be ready-to-observe by any player at any time.

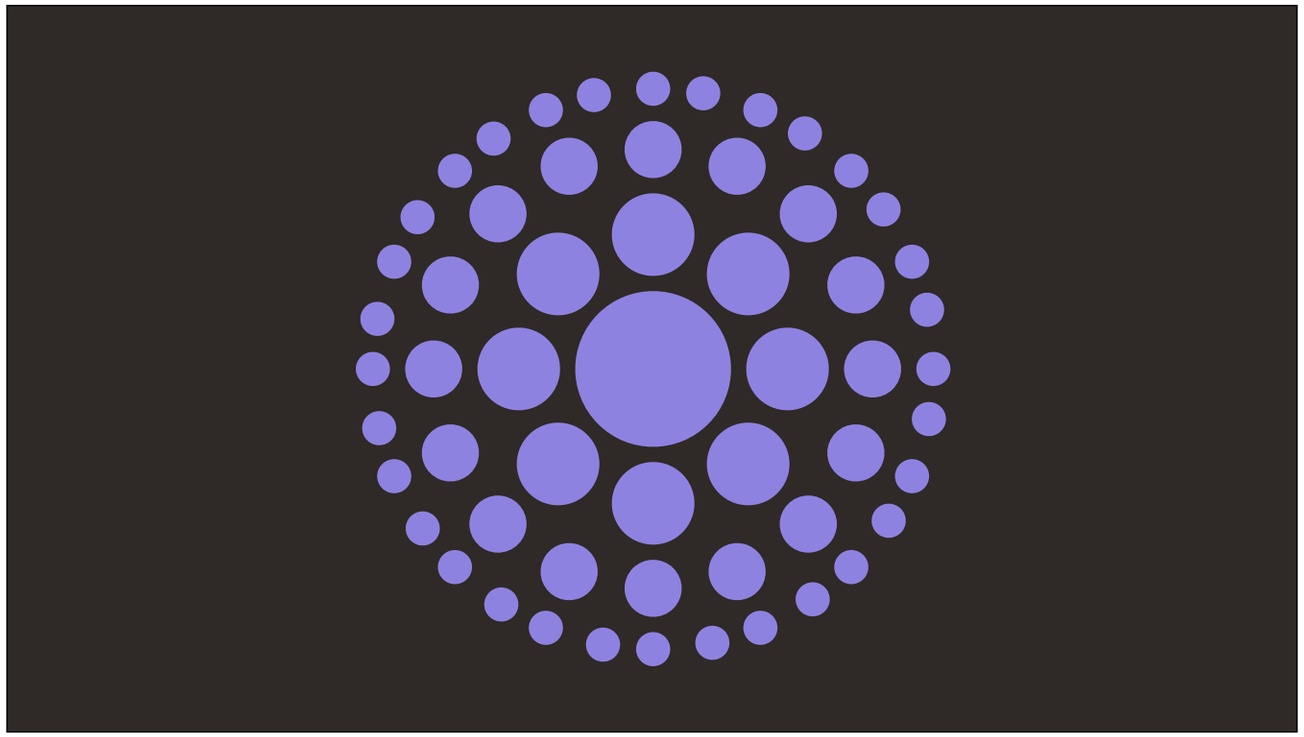


Now, most games have a culling distance — items that are sufficiently far from the camera are simply ignored by the renderer. This is done to speed up rendering. But as we mentioned earlier, Photon doesn't work like most renderers — all possible viewing paths have to be ready-to-observe by any player at any time.



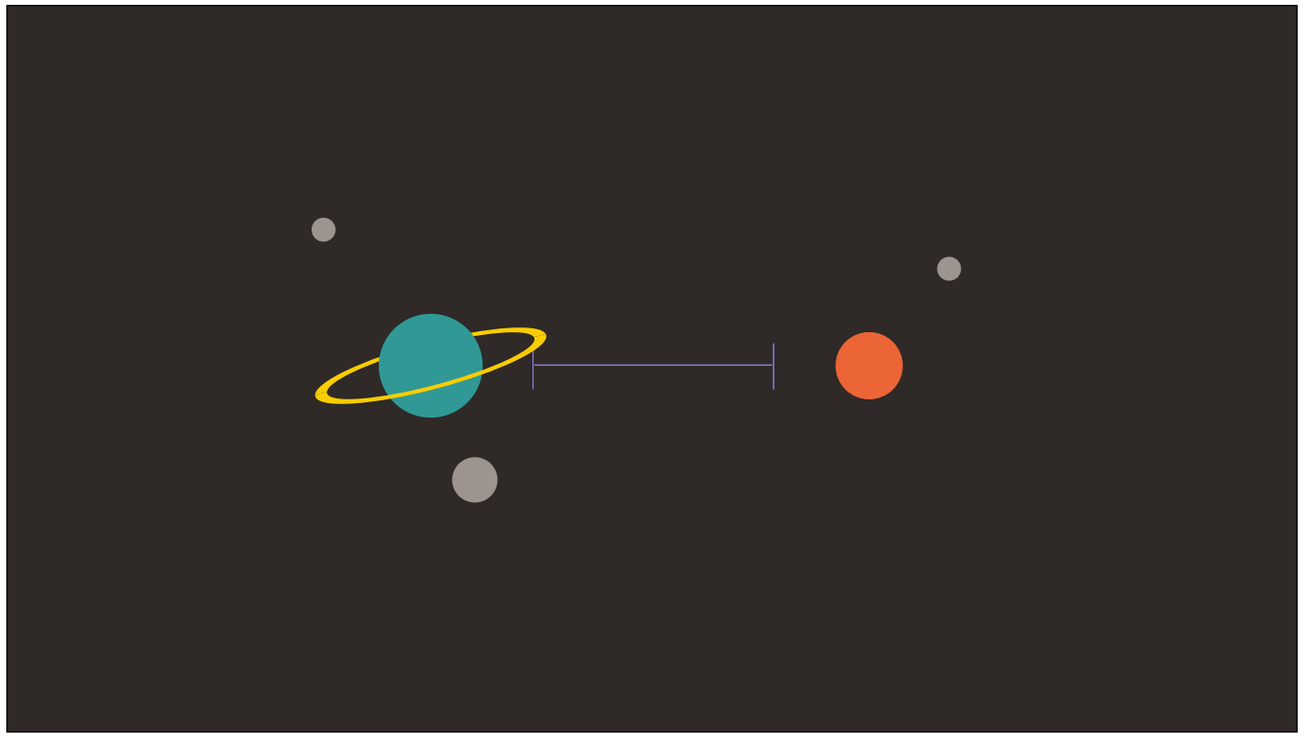
In an engine that advertises infinite space, this means that items close to the viewer have the same rendering priority as things that are billions of miles away— and this gives us the next seam to pick at.

Is it possible to construct a view with so much stuff in it that we can get renderer to lag — maybe even crash?



In an engine that advertises infinite space, this means that items close to the viewer have the same rendering priority as things that are billions of miles away— and this gives us the next seam to pick at.

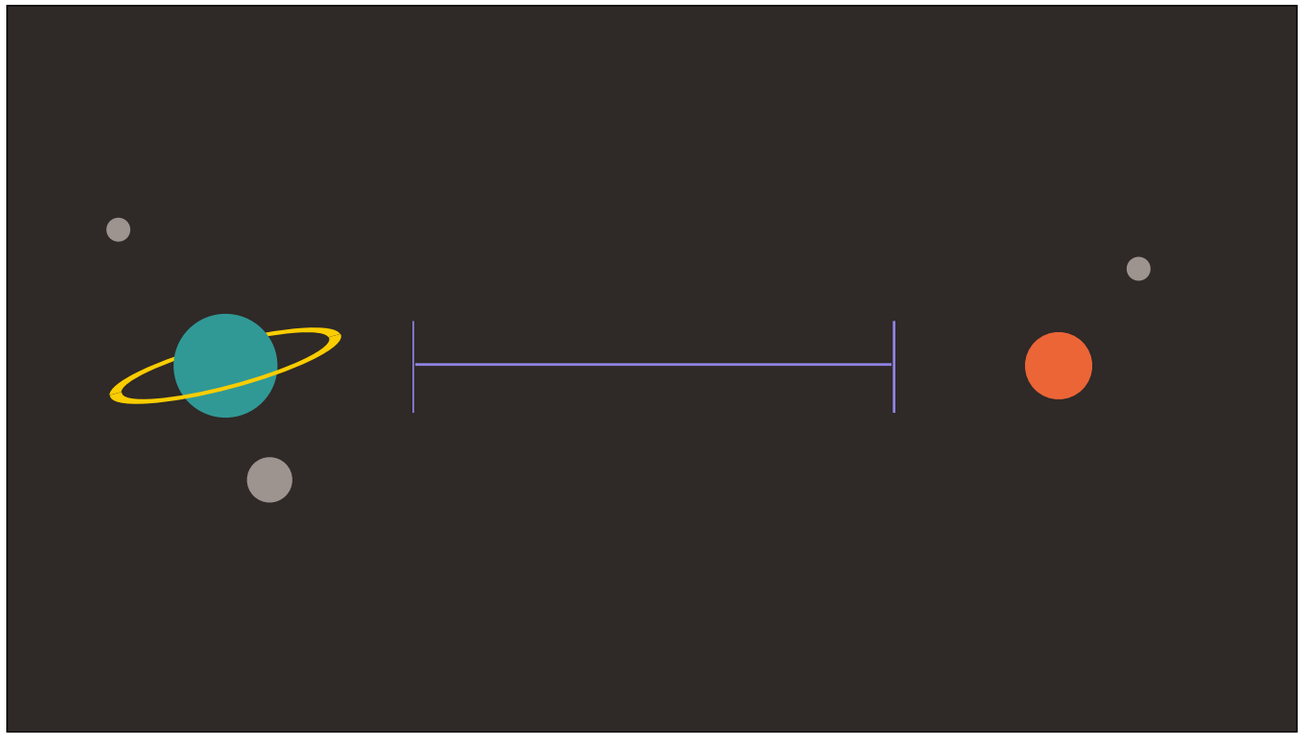
Is it possible to construct a view with so much stuff in it that we can get renderer to lag — maybe even crash?



And again, we hit a “no”. Why?

There are two clever systems at work here.

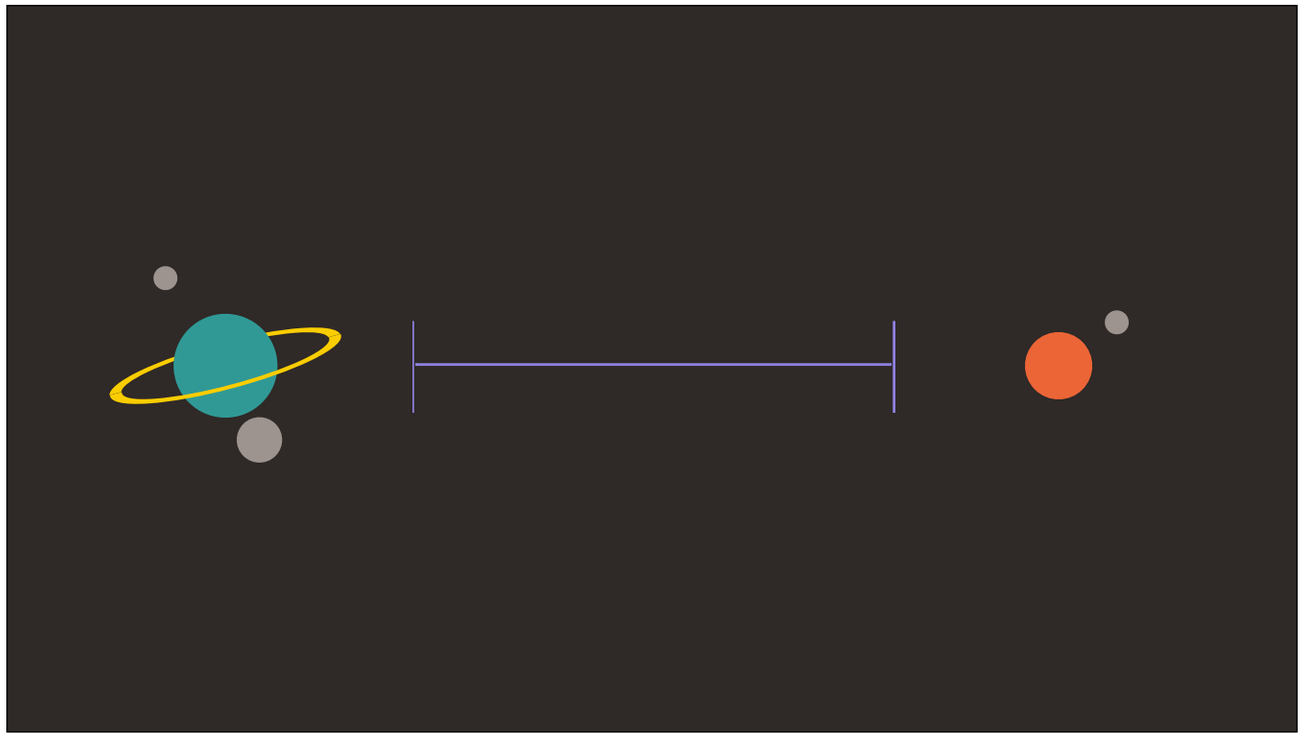
First off, Spacetime is constantly expanding, putting more and more space between the objects being stored, while the objects themselves are drawn together with gravity. This has the effect of making most the objects in the universe clump together in clusters, separated by an enormous space that grows every day — the same kind of optimisation algorithm used in sparse voxel octrees.



And again, we hit a “no”. Why?

There are two clever systems at work here.

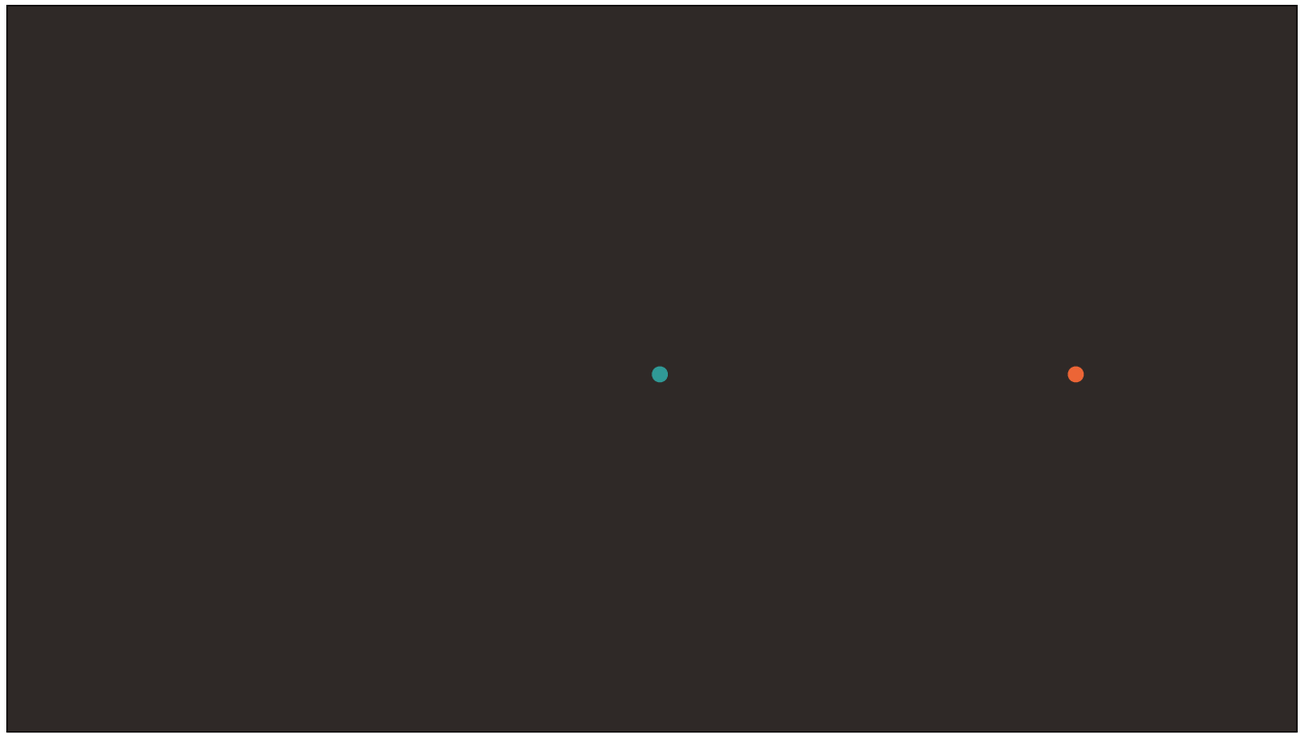
First off, Spacetime is constantly expanding, putting more and more space between the objects being stored, while the objects themselves are drawn together with gravity. This has the effect of making most the objects in the universe clump together in clusters, separated by an enormous space that grows every day — the same kind of optimisation algorithm used in sparse voxel octrees.



And again, we hit a “no”. Why?

There are two clever systems at work here.

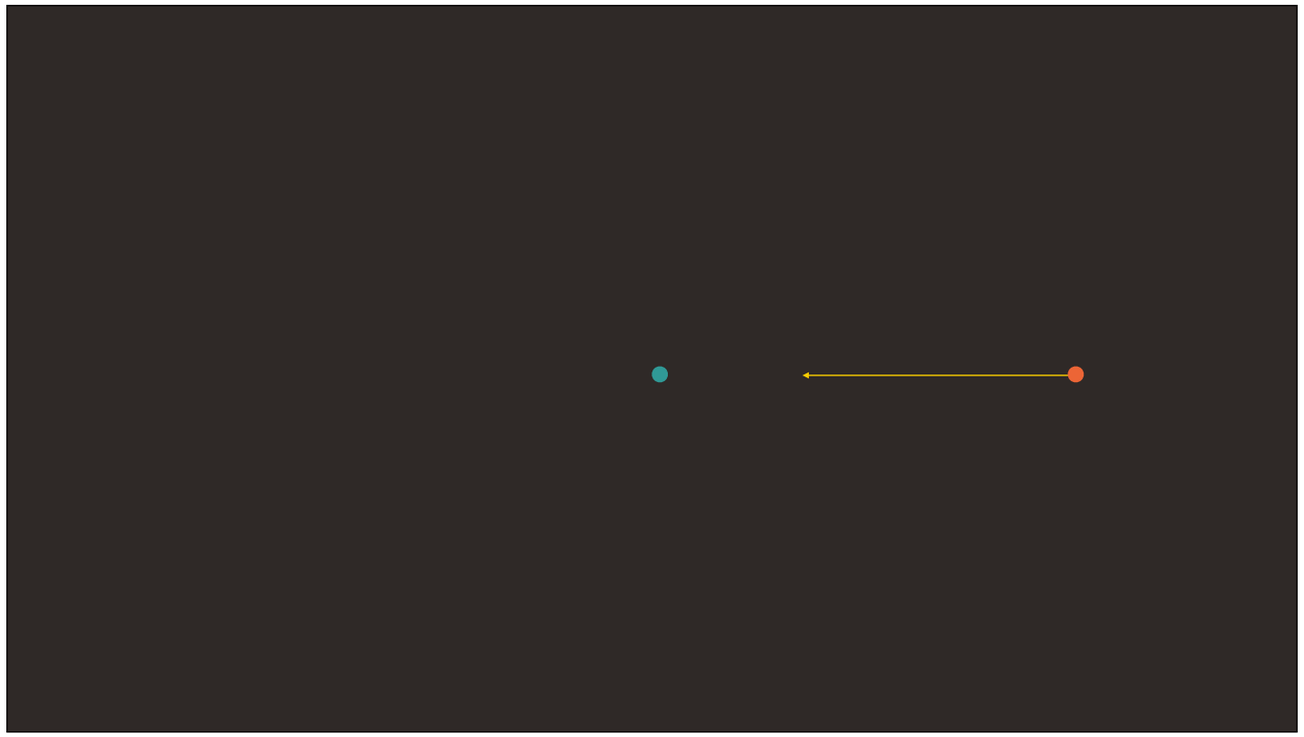
First off, Spacetime is constantly expanding, putting more and more space between the objects being stored, while the objects themselves are drawn together with gravity. This has the effect of making most the objects in the universe clump together in clusters, separated by an enormous space that grows every day — the same kind of optimisation algorithm used in sparse voxel octrees.



Secondly, the expansion interacts with the Photon engine in an unexpected way: since photons have a speed, objects can be so far away that the the light from them simply hasn't had enough time to reach us.

Since the universe is around 13.7 billion years old, and light can travel roughly 5 trillion miles a year, any objects further than 65 billion trillion miles away are, very naturally, not rendered.

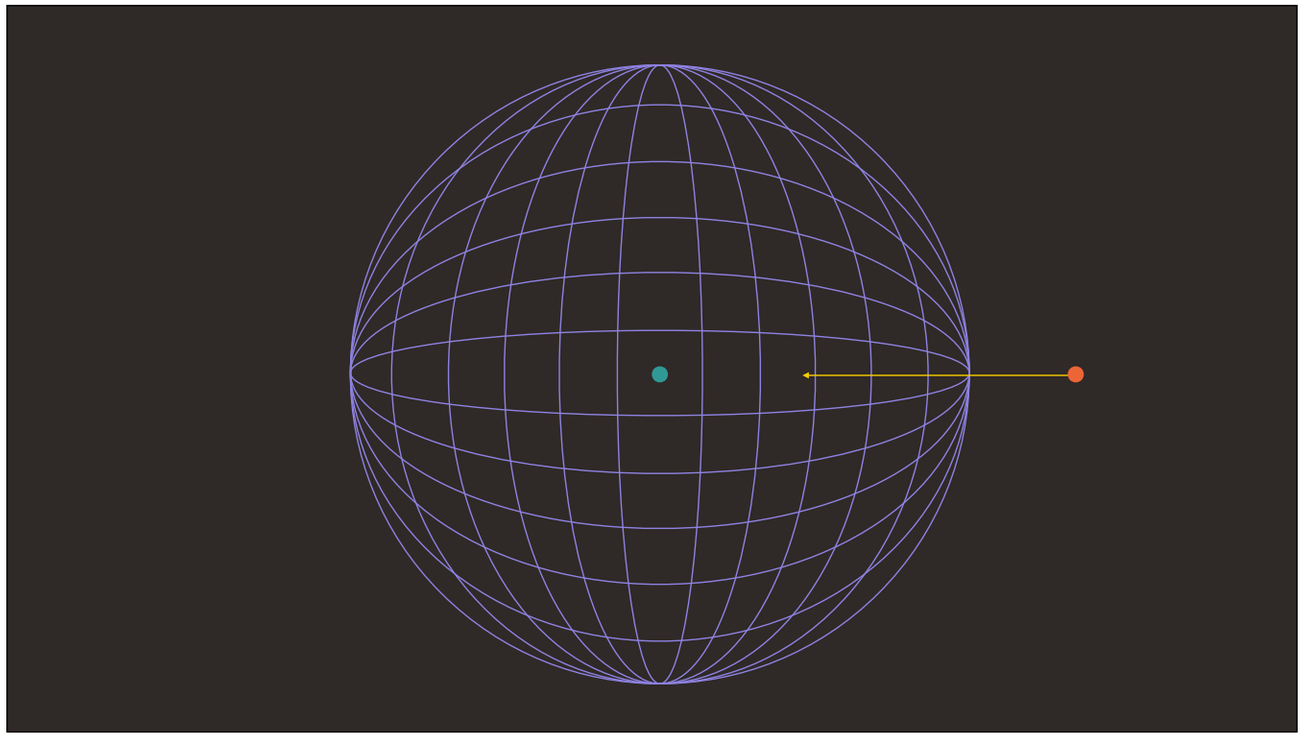
When you work in the expansion factor, this creates a kind of "rendering bubble" that's around 230 billion trillion miles across — that's one hell of a skybox.



Secondly, the expansion interacts with the Photon engine in an unexpected way: since photons have a speed, objects can be so far away that the the light from them simply hasn’t had enough time to reach us.

Since the universe is around 13.7 billion years old, and light can travel roughly 5 trillion miles a year, any objects further than 65 billion trillion miles away are, very naturally, not rendered.

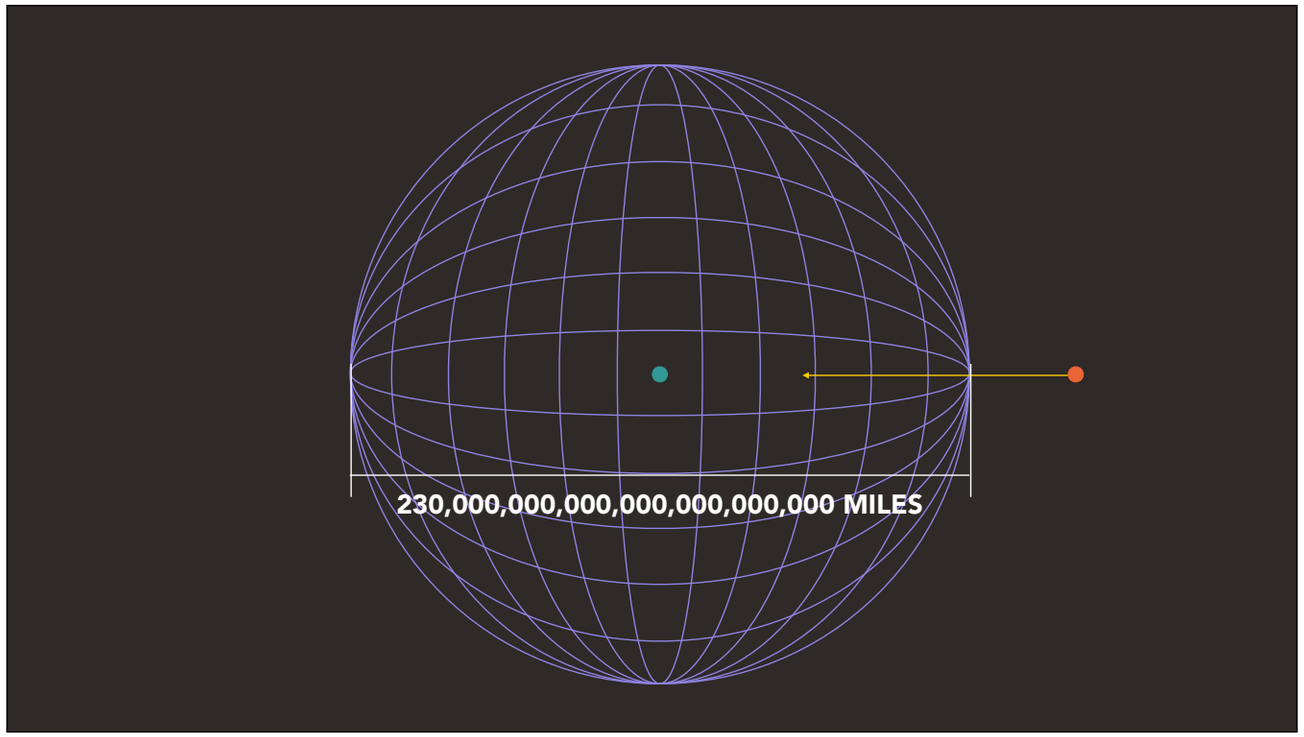
When you work in the expansion factor, this creates a kind of “rendering bubble” that’s around 230 billion trillion miles across — that’s one hell of a skybox.



Secondly, the expansion interacts with the Photon engine in an unexpected way: since photons have a speed, objects can be so far away that the the light from them simply hasn’t had enough time to reach us.

Since the universe is around 13.7 billion years old, and light can travel roughly 5 trillion miles a year, any objects further than 65 billion trillion miles away are, very naturally, not rendered.

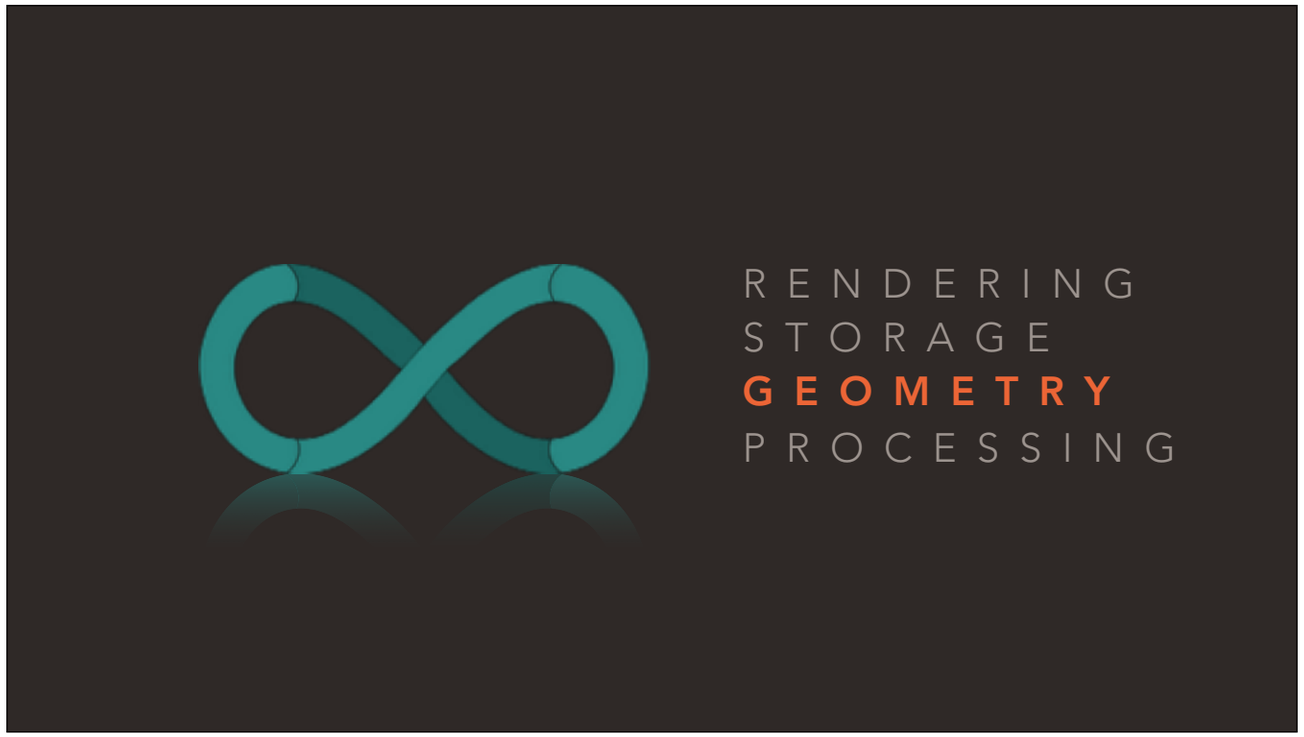
When you work in the expansion factor, this creates a kind of “rendering bubble” that’s around 230 billion trillion miles across — that’s one hell of a skybox.



Secondly, the expansion interacts with the Photon engine in an unexpected way: since photons have a speed, objects can be so far away that the the light from them simply hasn't had enough time to reach us.

Since the universe is around 13.7 billion years old, and light can travel roughly 5 trillion miles a year, any objects further than 65 billion trillion miles away are, very naturally, not rendered.

When you work in the expansion factor, this creates a kind of "rendering bubble" that's around 230 billion trillion miles across — that's one hell of a skybox.



Okay, so the Universe can handle massive. What about tiny? What happens when we start messing around with the smallest objects in the game?

For that, we'll need to look at how the Universe handles geometry.

There are two main methods of storing 3D data in games today.

There are two main methods of storing 3D data in games today.

The first and by far most popular is the vertex-mesh method — discrete points, linked together by edges to form surfaces. This is a very quick algorithm to render, which has led to its popularity, but it fails to describe things without explicitly defined surfaces, like clouds and water.

There are two main methods of storing 3D data in games today.

The first and by far most popular is the vertex-mesh method — discrete points, linked together by edges to form surfaces. This is a very quick algorithm to render, which has led to its popularity, but it fails to describe things without explicitly defined surfaces, like clouds and water.

There are two main methods of storing 3D data in games today.

The first and by far most popular is the vertex-mesh method — discrete points, linked together by edges to form surfaces. This is a very quick algorithm to render, which has led to its popularity, but it fails to describe things without explicitly defined surfaces, like clouds and water.

There are two main methods of storing 3D data in games today.

The first and by far most popular is the vertex-mesh method — discrete points, linked together by edges to form surfaces. This is a very quick algorithm to render, which has led to its popularity, but it fails to describe things without explicitly defined surfaces, like clouds and water.

There are two main methods of storing 3D data used in games today.

The first and by far most popular is the vertex-mesh method — discrete points, linked together by edges to form surfaces. This is a very quick algorithm to render, which has led to its popularity, but it fails to describe things without explicitly defined surfaces, like clouds and water.

The second, starting to become popular with the likes of Minecraft, is the volumetric method

The second, starting to become popular with the likes of Minecraft, is the volumetric method. Here, objects are described by functions, and surfaces are implied by the intersection of these volumes and the rays used to render them.

This renders volumes very naturally, but has it’s own limitations — it’s computationally expensive, rotation is difficult, and it makes physics much more difficult to calculate.

The Infinity Engine, once again, has a unique solution — Matter.

The second, starting to become popular with the likes of Minecraft, is the volumetric method. Here, objects are described by functions, and surfaces are implied by the intersection of these volumes and the rays used to render them.

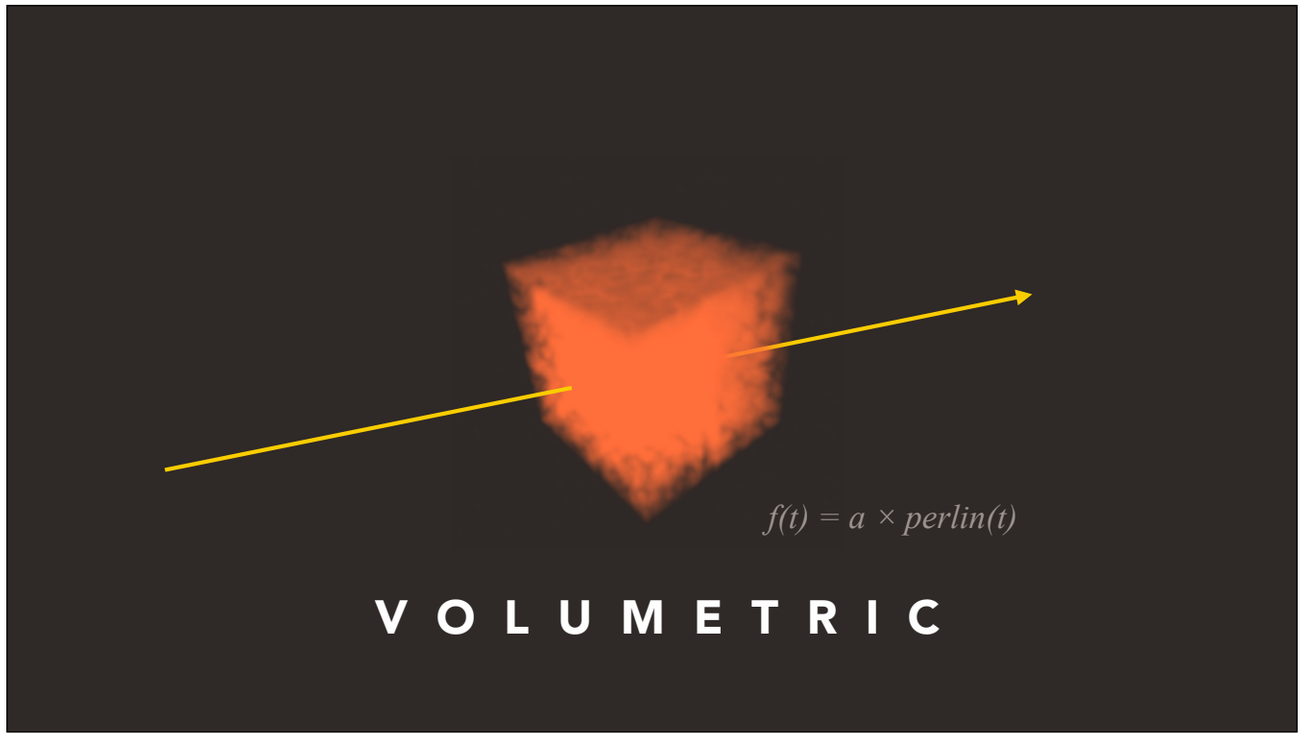
This renders volumes very naturally, but has it’s own limitations — it’s computationally expensive, rotation is difficult, and it makes physics much more difficult to calculate.

The Infinity Engine, once again, has a unique solution — Matter.

The second, starting to become popular with the likes of Minecraft, is the volumetric method. Here, objects are described by functions, and surfaces are implied by the intersection of these volumes and the rays used to render them.

This renders volumes very naturally, but has it’s own limitations — it’s computationally expensive, rotation is difficult, and it makes physics much more difficult to calculate.

The Infinity Engine, once again, has a unique solution — Matter.



The second, starting to become popular with the likes of Minecraft, is the volumetric method. Here, objects are described by functions, and surfaces are implied by the intersection of these volumes and the rays used to render them.

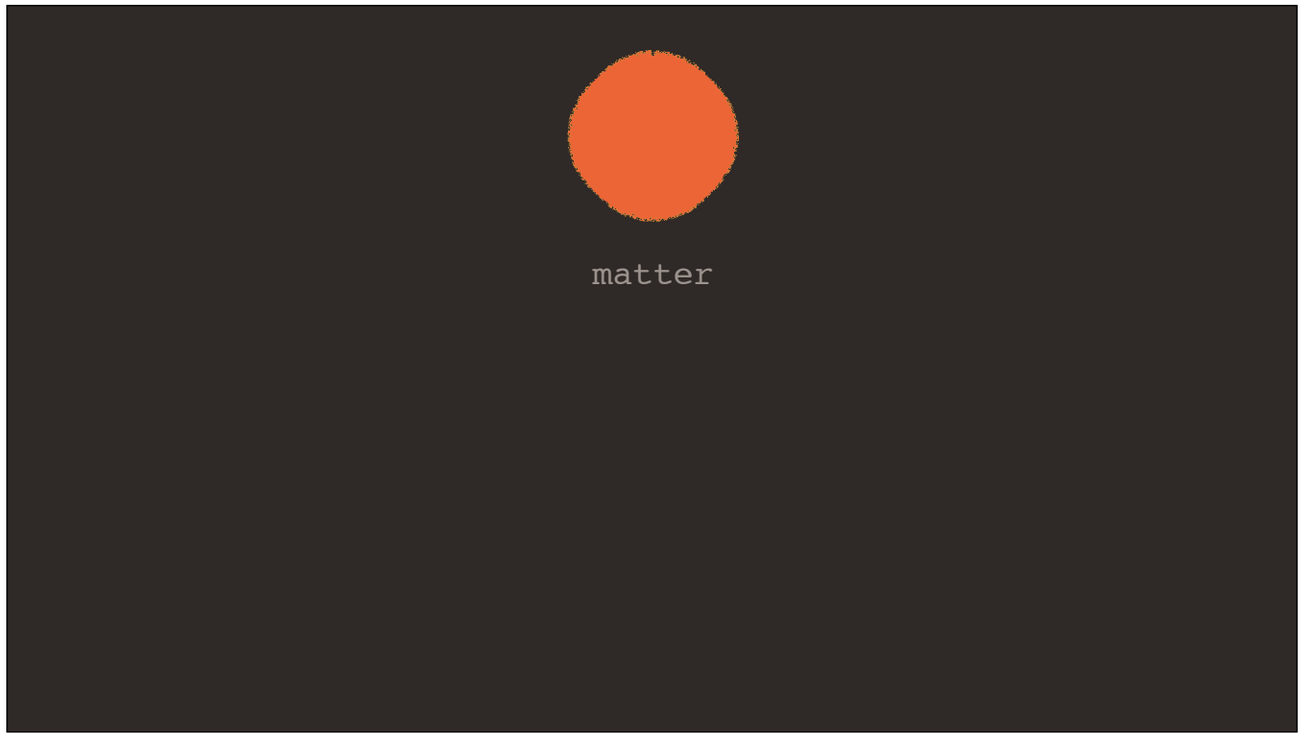
This renders volumes very naturally, but has it’s own limitations — it’s computationally expensive, rotation is difficult, and it makes physics much more difficult to calculate.

The Infinity Engine, once again, has a unique solution — Matter.

The Infinity Engine, once again, has a unique solution — Matter.

In this paradigm, all objects are stored as something which shares properties of both discrete vertex geometry and continuous volumetric geometry.

The API allows for it to be cast as discrete for some calculations, and continuous for others, all without changing the underlying data.

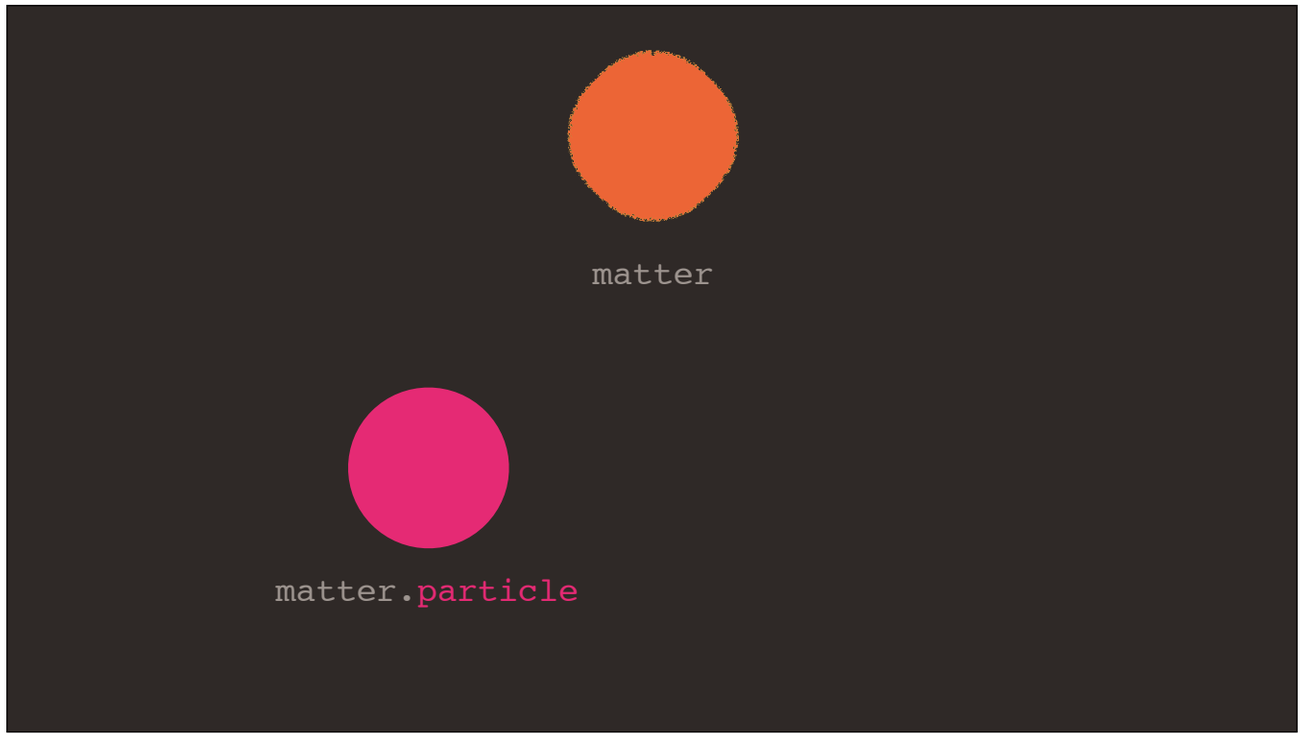


The Infinity Engine, once again, has a unique solution — Matter.

In this paradigm, all objects are stored as something which shares properties of both discrete vertex geometry and continuous volumetric geometry.

The API allows for it to be cast as discrete for some calculations, and continuous for others.

This is an incredibly flexible approach, but it has a very weird side effect. Since you can only get at the data in these two very different ways, it can MAKE things really ambiguous.

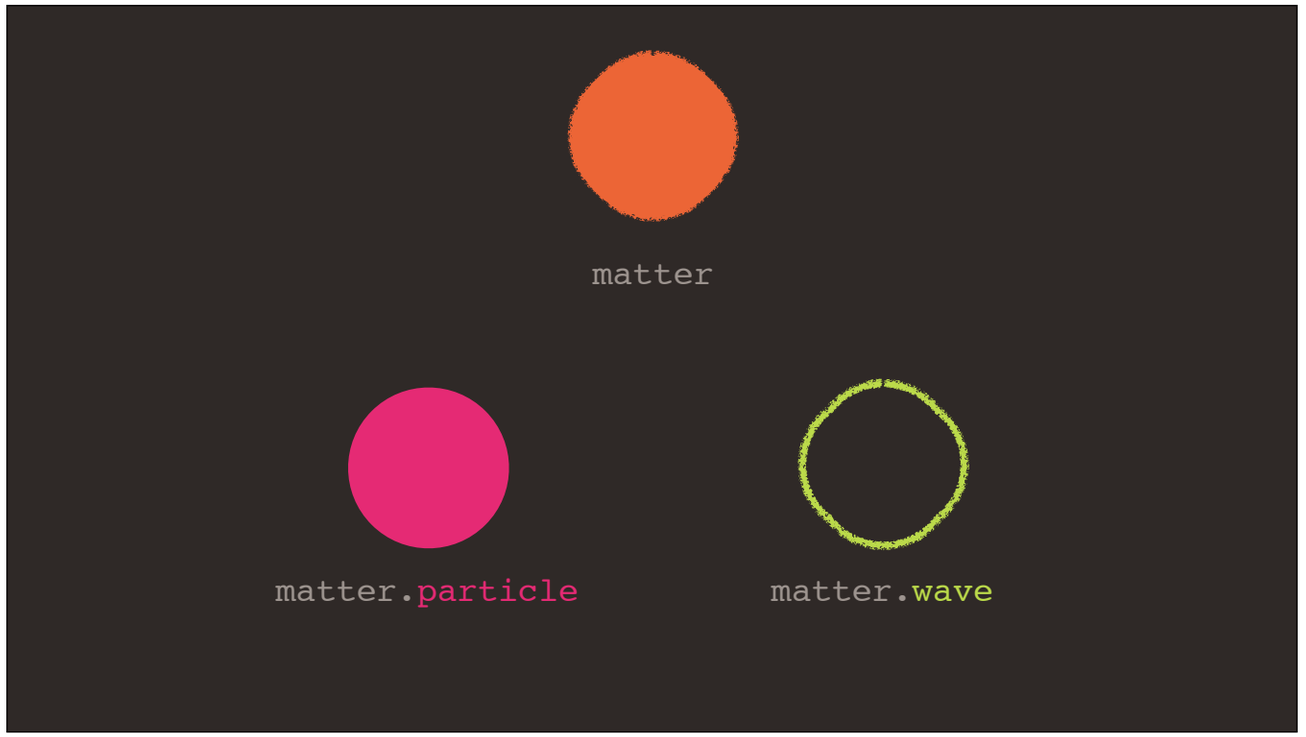


The Infinity Engine, once again, has a unique solution — Matter.

In this paradigm, all objects are stored as something which shares properties of both discrete vertex geometry and continuous volumetric geometry.

The API allows for it to be cast as discrete for some calculations, and continuous for others.

This is an incredibly flexible approach, but it has a very weird side effect. Since you can only get at the data in these two very different ways, it can MAKE things really ambiguous.

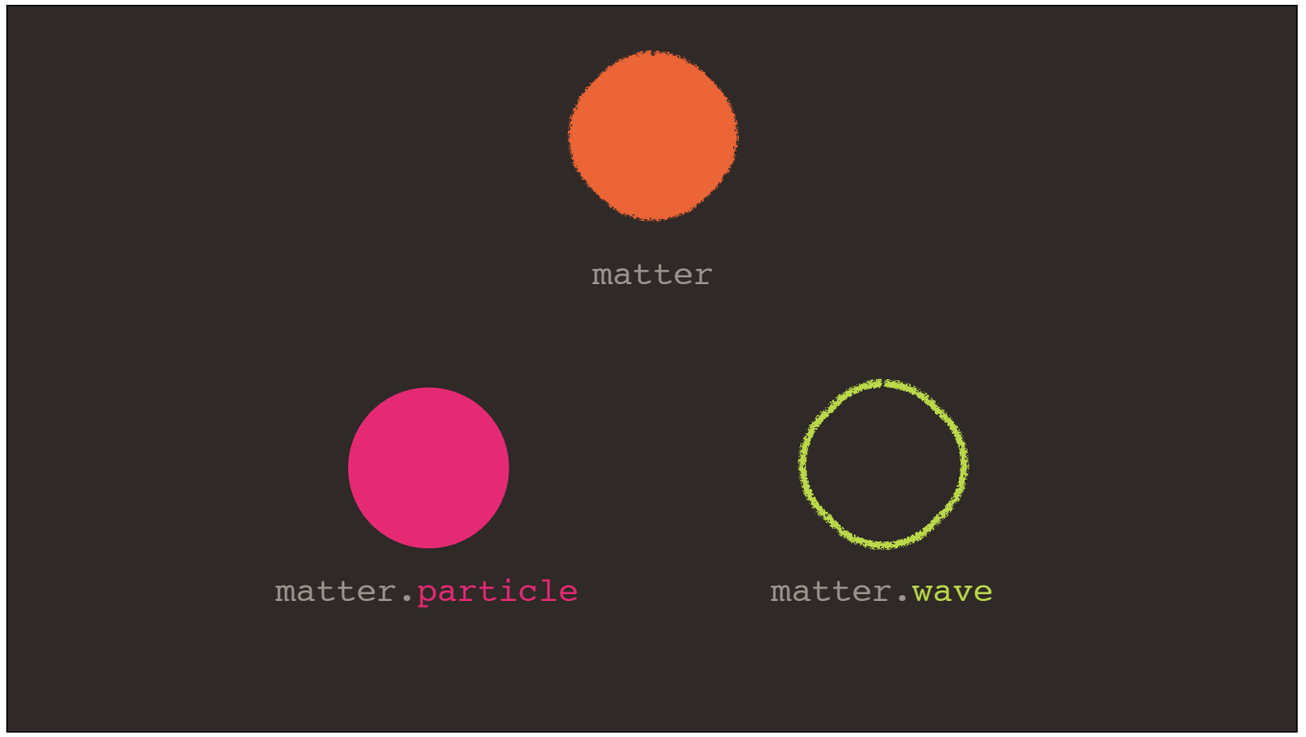


The Infinity Engine, once again, has a unique solution — Matter.

In this paradigm, all objects are stored as something which shares properties of both discrete vertex geometry and continuous volumetric geometry.

The API allows for it to be cast as discrete for some calculations, and continuous for others.

This is an incredibly flexible approach, but it has a very weird side effect. Since you can only get at the data in these two very different ways, it can MAKE things really ambiguous.



The Infinity Engine, once again, has a unique solution — Matter.

In this paradigm, all objects are stored as something which shares properties of both discrete vertex geometry and continuous volumetric geometry.

The API allows for it to be cast as discrete for some calculations, and continuous for others.

This is an incredibly flexible approach, but it has a very weird side effect. Since you can only get at the data in these two very different ways, it can MAKE things really ambiguous.

Let’s look at this tiny piece of matter. It has a position, and a velocity. So far so good. But I want to be precise, so I need to know its exact position.

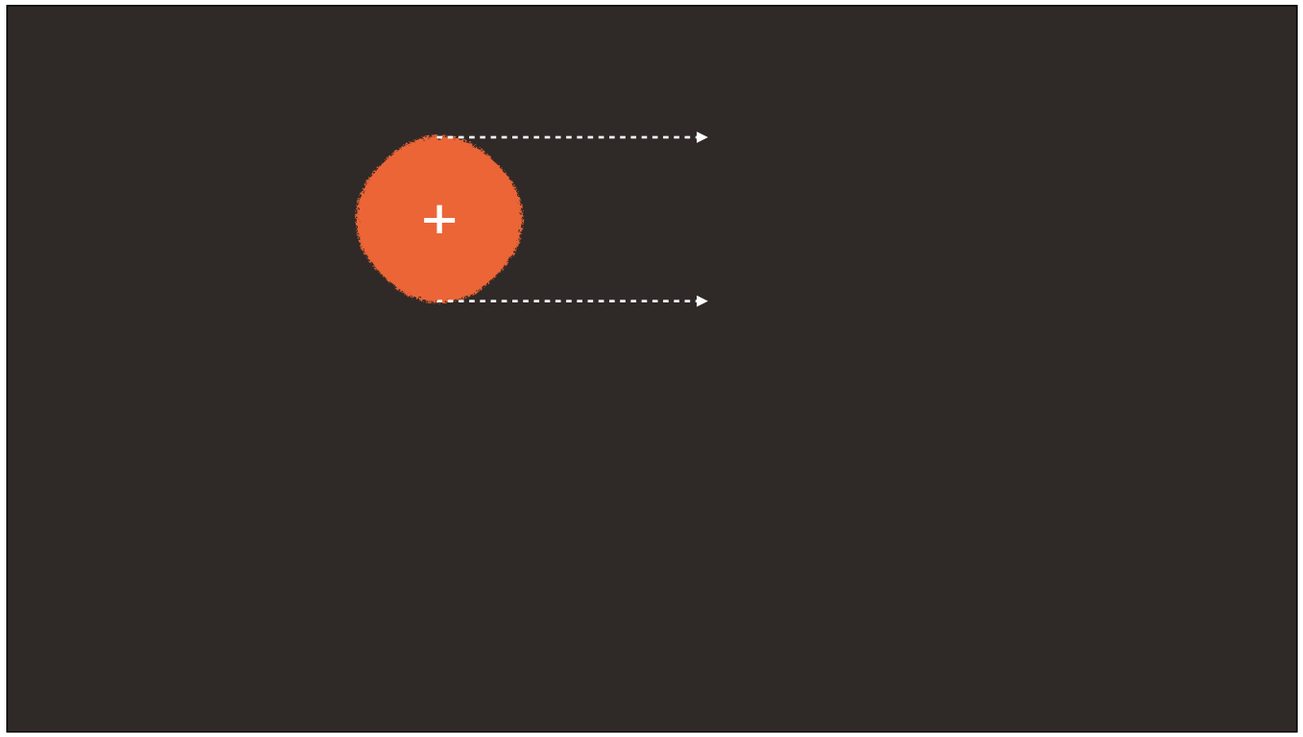
To do that, I’ll make it a discrete object, by calling matter.particle.GetPosition(). Let’s see what happens. Ok, now I know exactly where it is, but I can’t figure out its exact velocity.

But what if I WANT its exact velocity? Well, I have to make it a continuous object: matter.wave.GetVelocity(). When I do that, I now know its exact speed and direction, but I can no longer tell you exactly where it is. This is totally a bug, right?

Let’s look at this tiny piece of matter. It has a position, and a velocity. So far so good. But I want to be precise, so I need to know its exact position.

To do that, I’ll make it a discrete object, by calling matter.particle.GetPosition(). Let’s see what happens. Ok, now I know exactly where it is, but I can’t figure out its exact velocity.

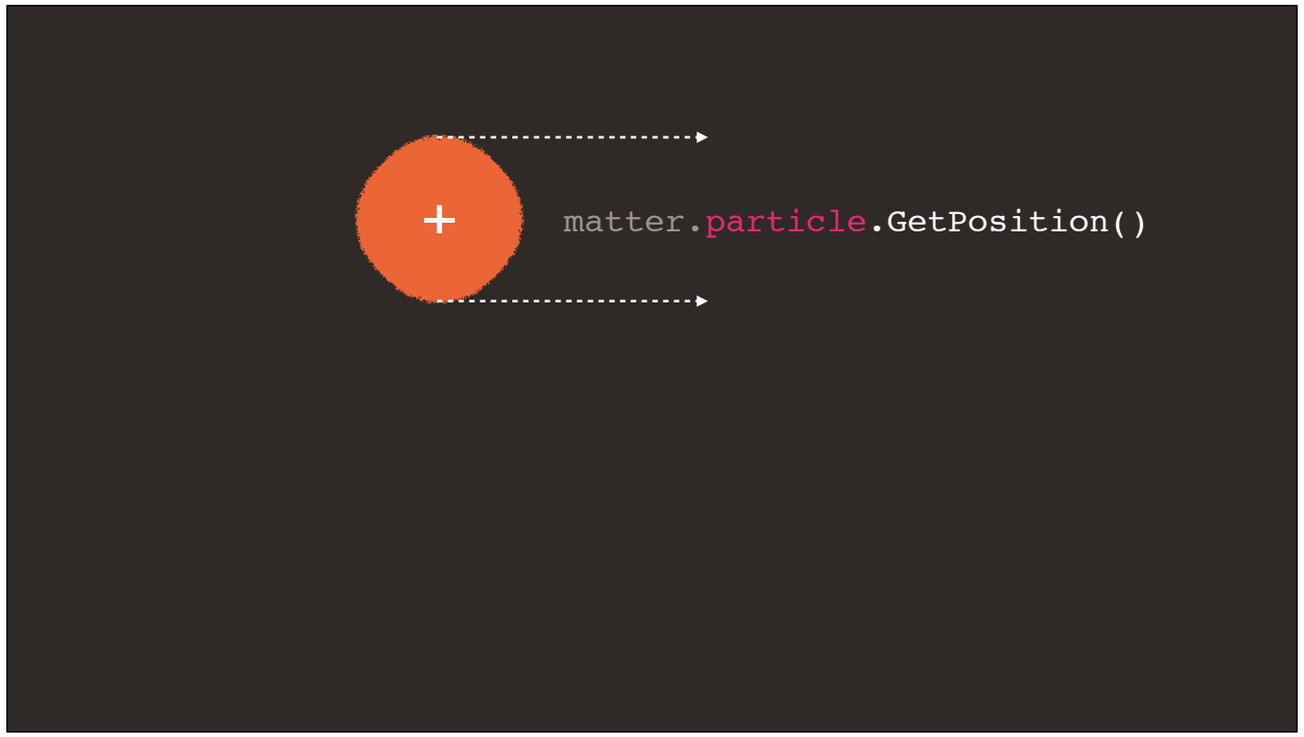
But what if I WANT its exact velocity? Well, I have to make it a continuous object: matter.wave.GetVelocity(). When I do that, I now know its exact speed and direction, but I can no longer tell you exactly where it is. This is totally a bug, right?



Let’s look at this tiny piece of matter. It has a position, and a velocity. So far so good. But I want to be precise, so I need to know its exact position.

To do that, I’ll make it a discrete object, by calling matter.particle.GetPosition(). Let’s see what happens. Ok, now I know exactly where it is, but I can’t figure out its exact velocity.

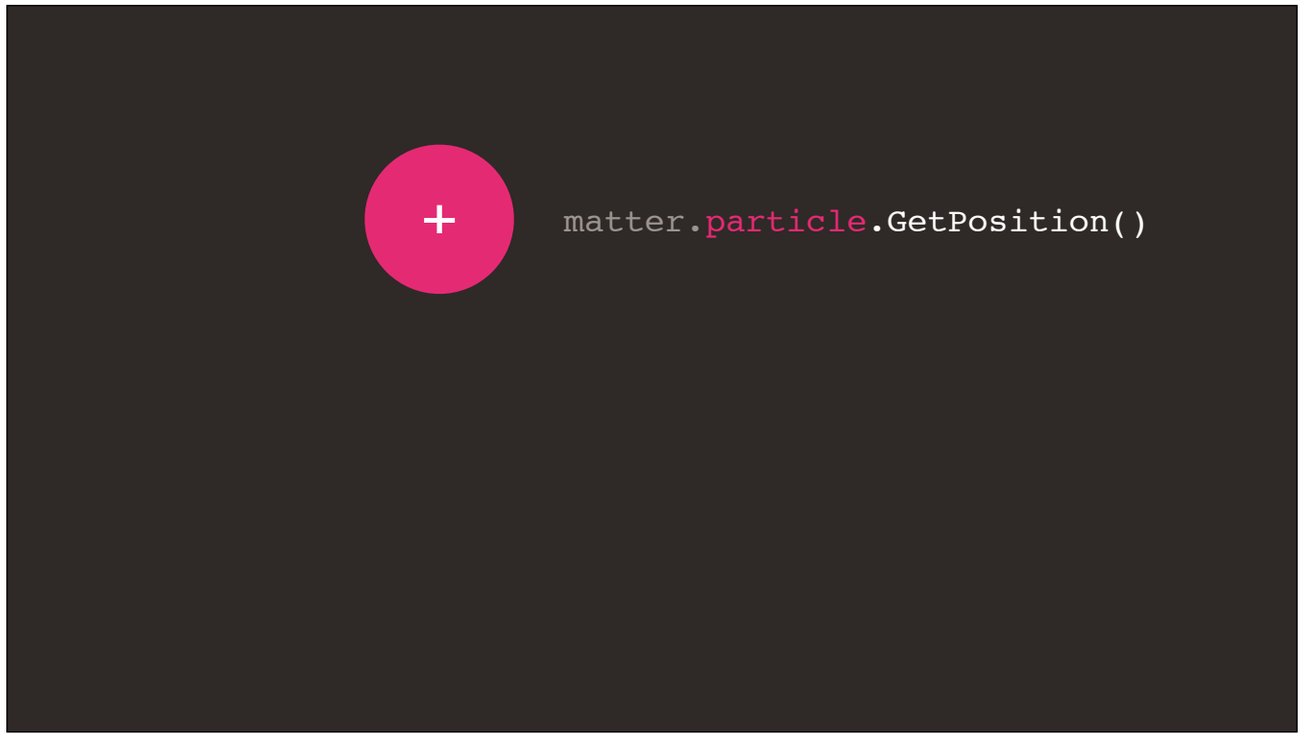
But what if I WANT its exact velocity? Well, I have to make it a continuous object: matter.wave.GetVelocity(). When I do that, I now know its exact speed and direction, but I can no longer tell you exactly where it is. This is totally a bug, right?



Let’s look at this tiny piece of matter. It has a position, and a velocity. So far so good. But I want to be precise, so I need to know its exact position.

To do that, I’ll make it a discrete object, by calling matter.particle.GetPosition(). Let’s see what happens. Ok, now I know exactly where it is, but I can’t figure out its exact velocity.

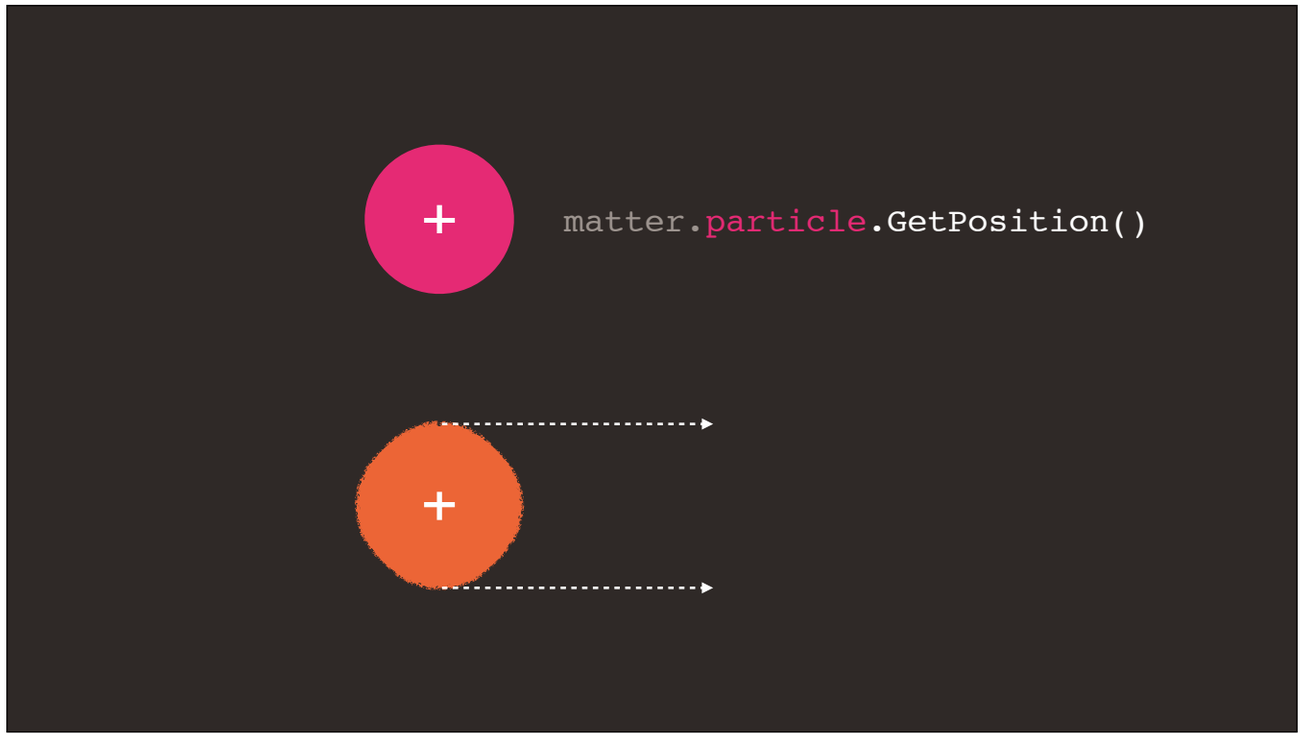
But what if I WANT its exact velocity? Well, I have to make it a continuous object: matter.wave.GetVelocity(). When I do that, I now know its exact speed and direction, but I can no longer tell you exactly where it is. This is totally a bug, right?



Let’s look at this tiny piece of matter. It has a position, and a velocity. So far so good. But I want to be precise, so I need to know its exact position.

To do that, I’ll make it a discrete object, by calling matter.particle.GetPosition(). Let’s see what happens. Ok, now I know exactly where it is, but I can’t figure out its exact velocity.

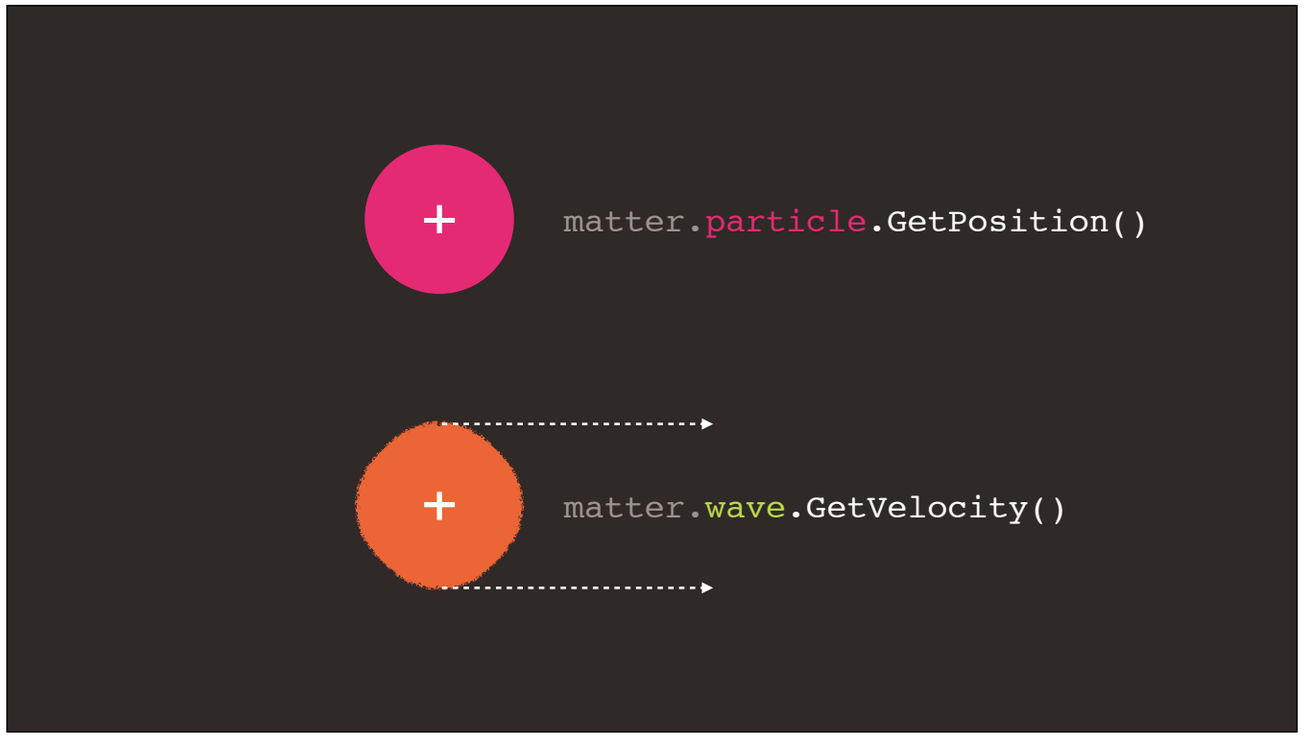
But what if I WANT its exact velocity? Well, I have to make it a continuous object: matter.wave.GetVelocity(). When I do that, I now know its exact speed and direction, but I can no longer tell you exactly where it is. This is totally a bug, right?



Let’s look at this tiny piece of matter. It has a position, and a velocity. So far so good. But I want to be precise, so I need to know its exact position.

To do that, I’ll make it a discrete object, by calling matter.particle.GetPosition(). Let’s see what happens. Ok, now I know exactly where it is, but I can’t figure out its exact velocity.

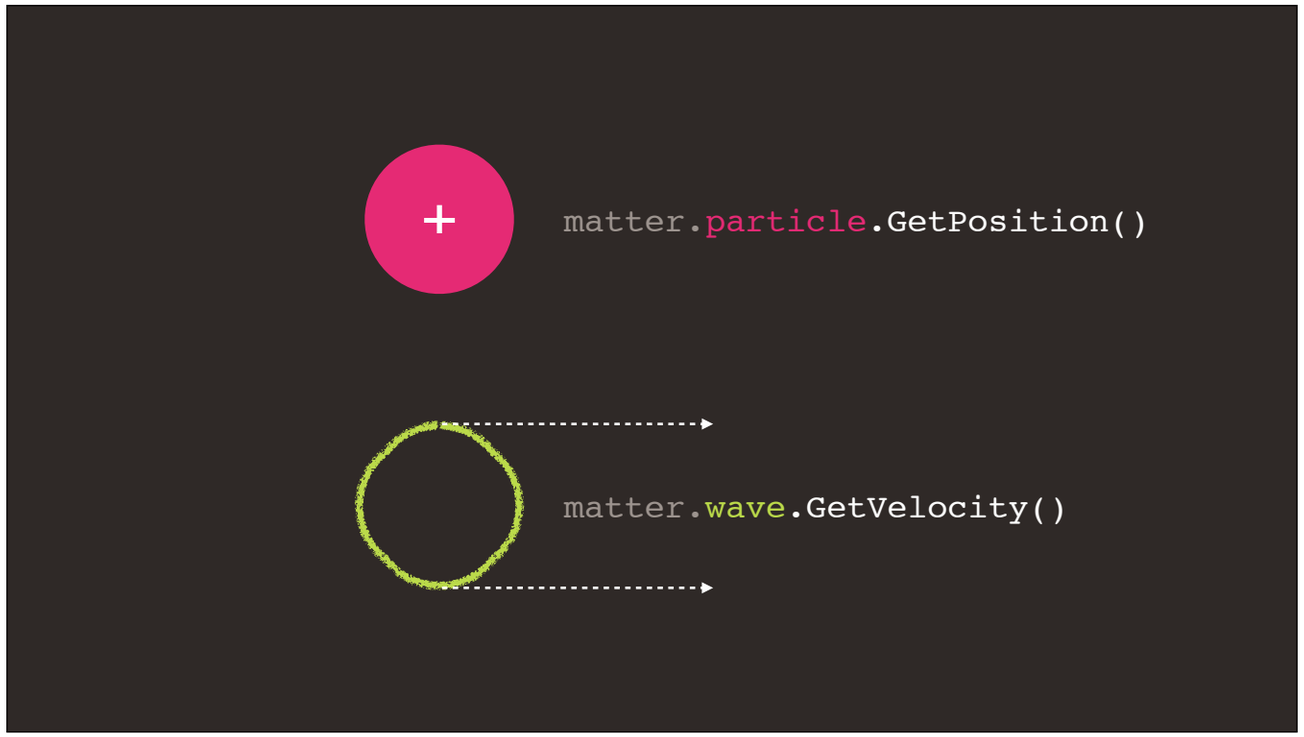
But what if I WANT its exact velocity? Well, I have to make it a continuous object: matter.wave.GetVelocity(). When I do that, I now know its exact speed and direction, but I can no longer tell you exactly where it is. This is totally a bug, right?



Let’s look at this tiny piece of matter. It has a position, and a velocity. So far so good. But I want to be precise, so I need to know its exact position.

To do that, I’ll make it a discrete object, by calling matter.particle.GetPosition(). Let’s see what happens. Ok, now I know exactly where it is, but I can’t figure out its exact velocity.

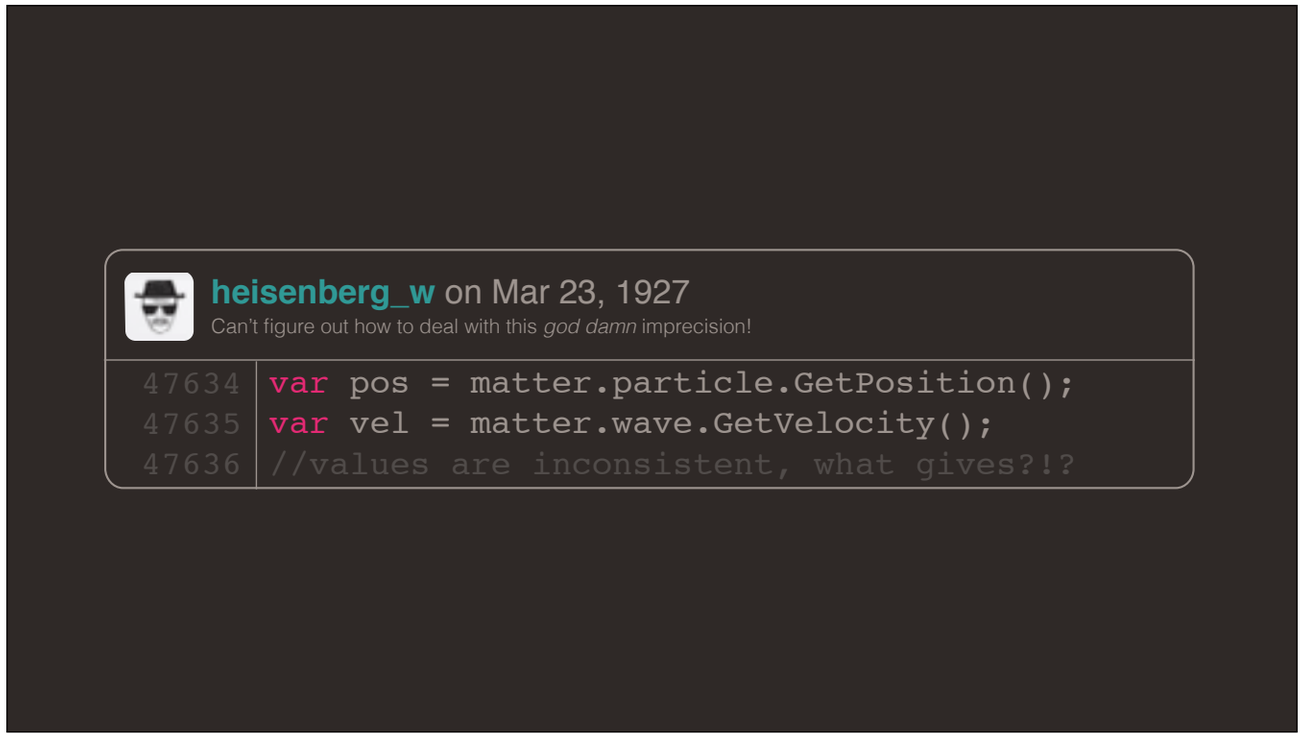
But what if I WANT its exact velocity? Well, I have to make it a continuous object: matter.wave.GetVelocity(). When I do that, I now know its exact speed and direction, but I can no longer tell you exactly where it is. This is totally a bug, right?



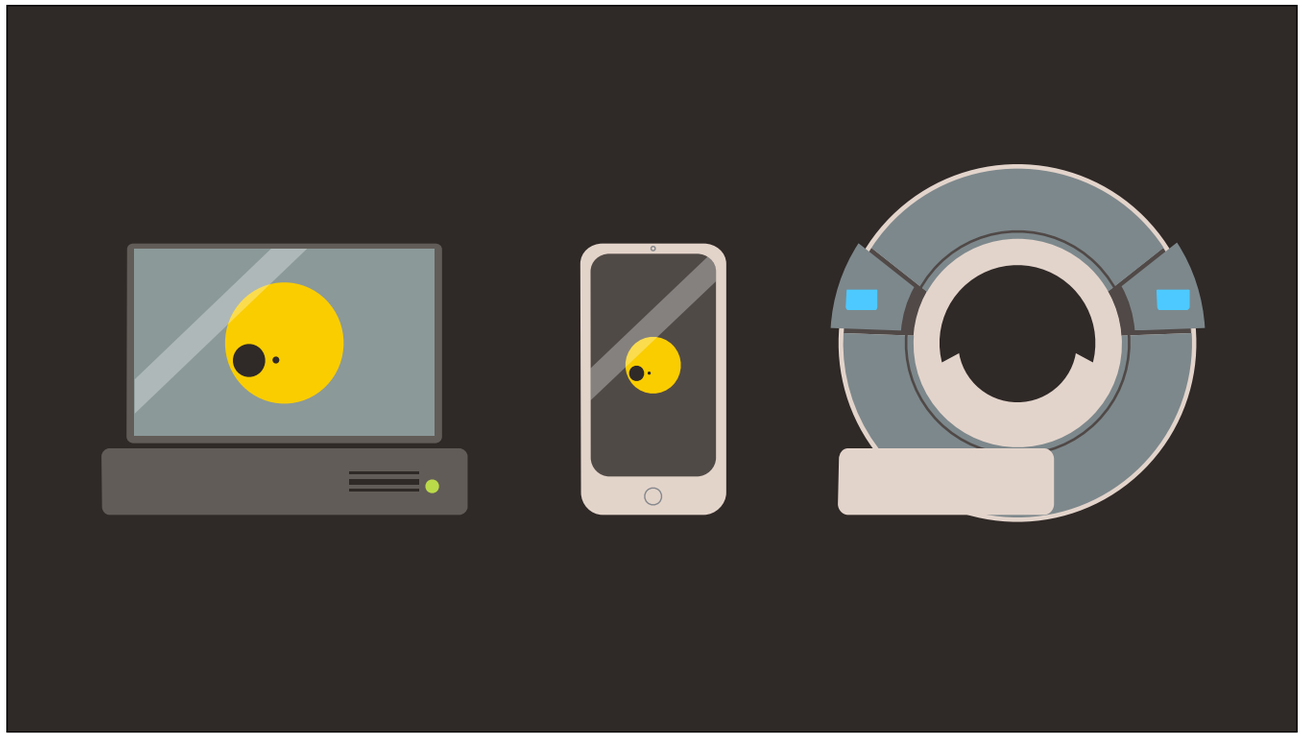
Let’s look at this tiny piece of matter. It has a position, and a velocity. So far so good. But I want to be precise, so I need to know its exact position.

To do that, I’ll make it a discrete object, by calling matter.particle.GetPosition(). Let’s see what happens. Ok, now I know exactly where it is, but I can’t figure out its exact velocity.

But what if I WANT its exact velocity? Well, I have to make it a continuous object: matter.wave.GetVelocity(). When I do that, I now know its exact speed and direction, but I can no longer tell you exactly where it is. This is totally a bug, right?



This is totally a bug, right? It definitely feels like the kind of thing that can be exploited.



Turns out, it is! This ambiguity is actually responsible for a lot of our technology — computers, cell phones, and MRI machines all rely on clever exploits of this bug at the subatomic scale.

And at normal scales, the uncertainty kind of just kinda… works itself out. The average player never notices, because the ambiguity is extremely small. This is the prime example of a shippable bug.

Turns out, it is! This ambiguity is actually responsible for a lot of our technology — computers, cell phones, and MRI machines all rely on clever exploits of this bug at the subatomic scale.

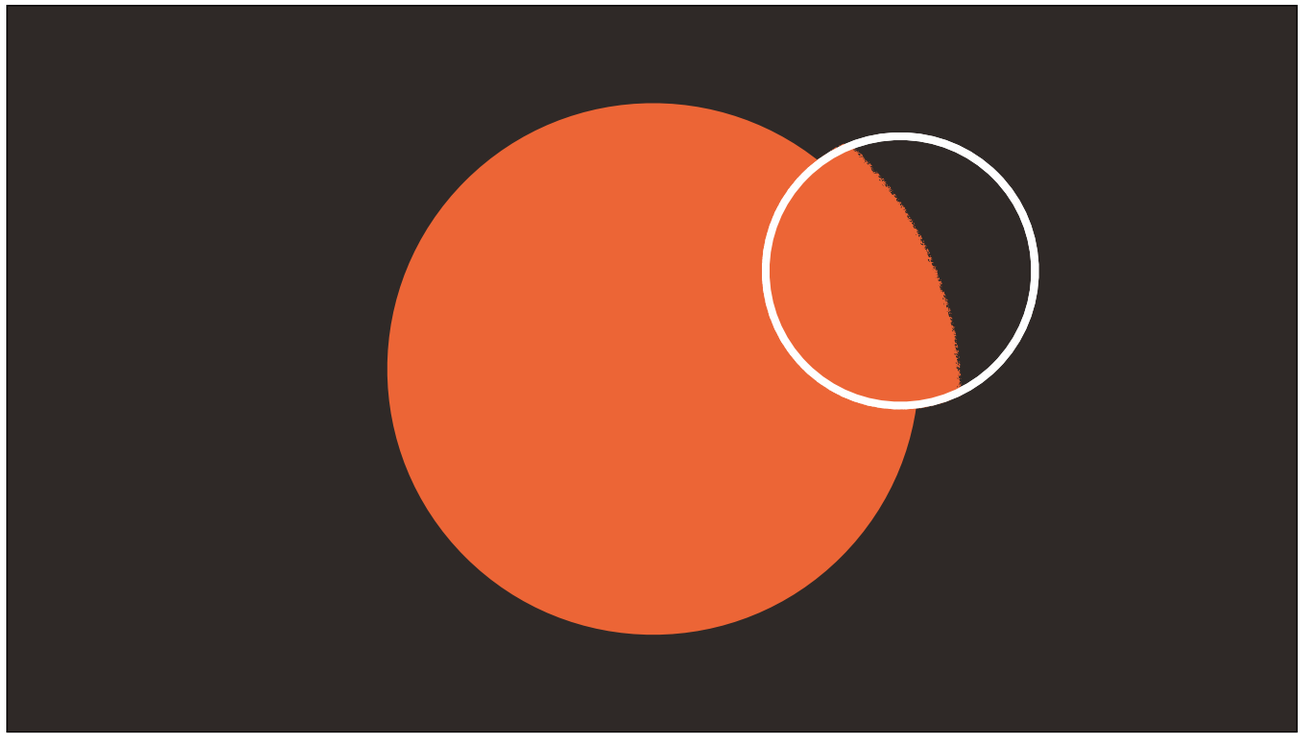
And at normal scales, the uncertainty kinda just… works itself out. The average player never notices, because the ambiguity is extremely small. This is the prime example of a shippable bug.

Ok. So far we've explored speed, space, and size. Let's bring it all together and take a look at what makes the Universe TICK — the central processing module, called Quantum.

Turns out, it is! This ambiguity is actually responsible for a lot of our technology — computers, cell phones, and MRI machines all rely on clever exploits of this bug at the subatomic scale.

And at normal scales, the uncertainty kinda just… works itself out. The average player never notices, because the ambiguity is extremely small. This is the prime example of a shippable bug.

Ok. So far we've explored speed, space, and size. Let's bring it all together and take a look at what makes the Universe TICK — the central processing module, called Quantum.



Turns out, it is! This ambiguity is actually responsible for a lot of our technology — computers, cell phones, and MRI machines all rely on clever exploits of this bug at the subatomic scale.

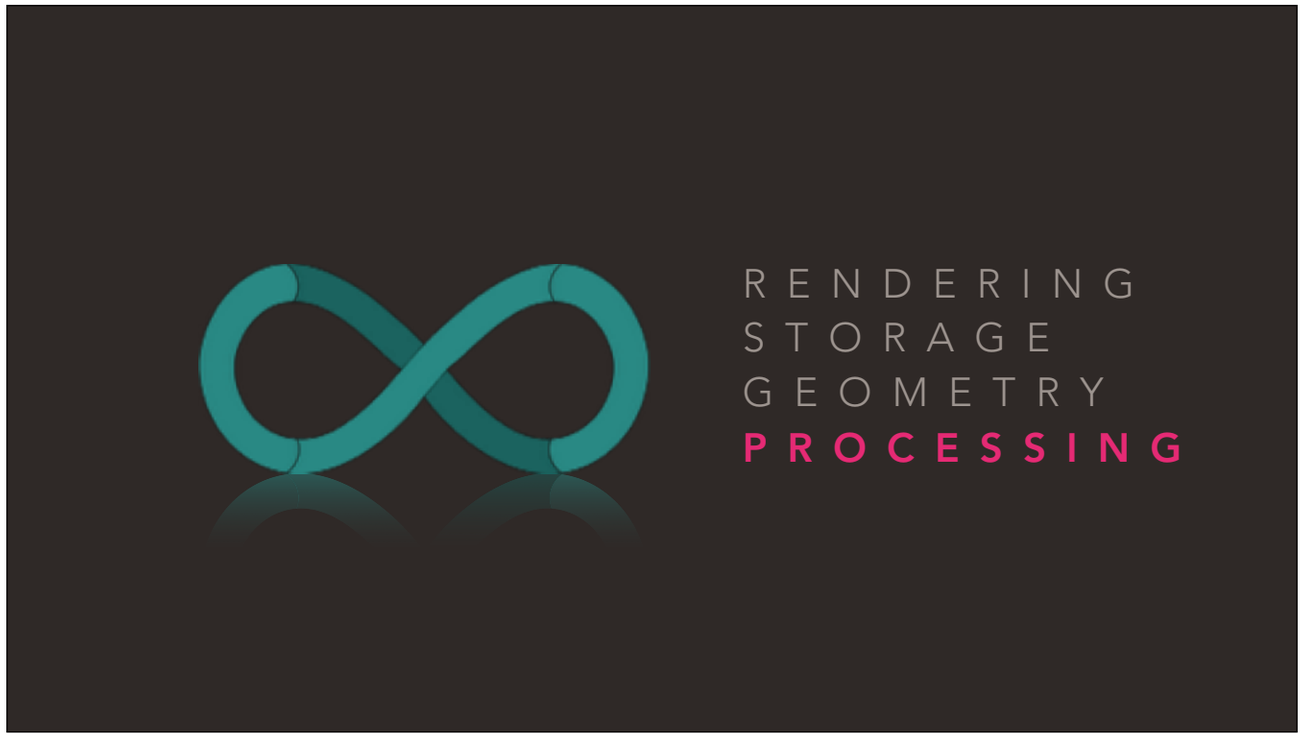
And at normal scales, the uncertainty kinda just… works itself out. The average player never notices, because the ambiguity is extremely small. This is the prime example of a shippable bug.

Ok. So far we've explored speed, space, and size. Let's bring it all together and take a look at what makes the Universe TICK — the central processing module, called Quantum.

Turns out, it is! This ambiguity is actually responsible for a lot of our technology — computers, cell phones, and MRI machines all rely on clever exploits of this bug at the subatomic scale.

And at normal scales, the uncertainty kinda just… works itself out. The average player never notices, because the ambiguity is extremely small. This is the prime example of a shippable bug.

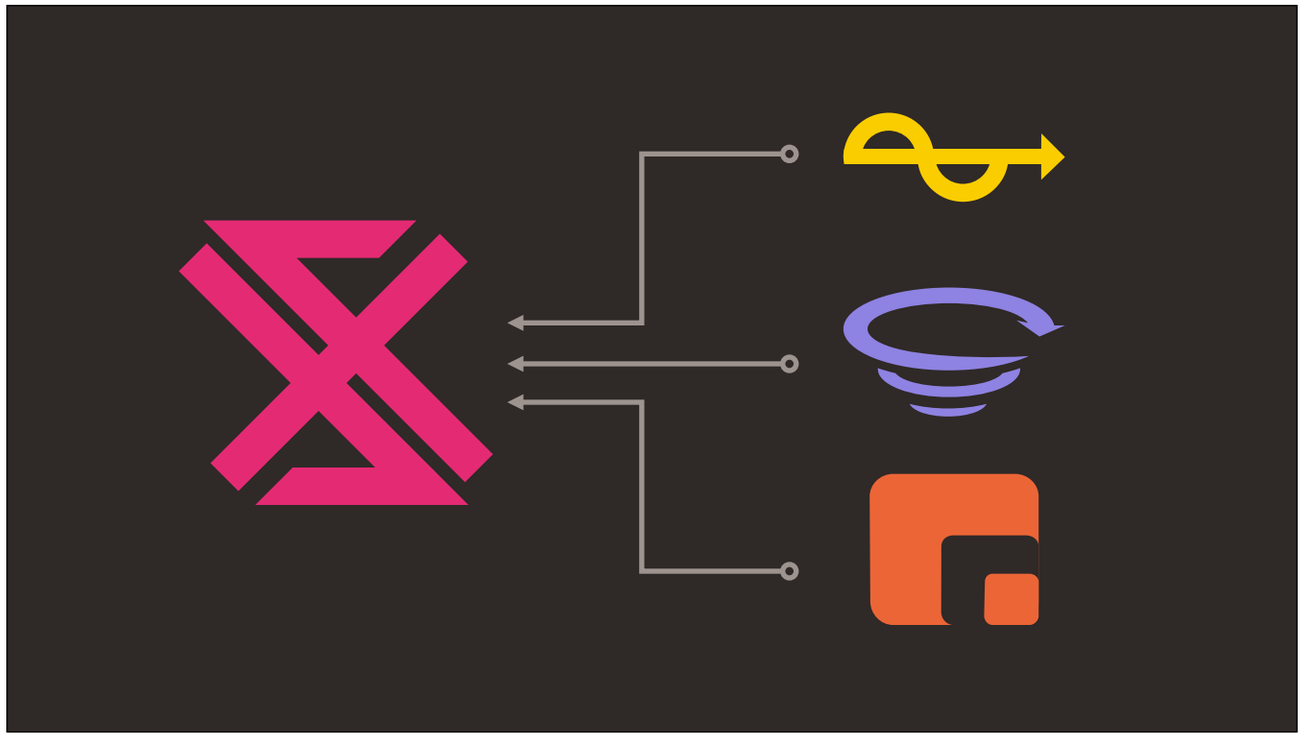
Ok. So far we've explored speed, space, and size. Let's bring it all together and take a look at what makes the Universe TICK — the central processing module, called Quantum.



Ok. So far we've explored speed, space, and size. Let's bring it all together and take a look at what makes the Universe TICK — the central processing module, called Quantum.

Ok. So far we've explored speed, space, and size. Let's bring it all together and take a look at what makes the Universe tick — the central processing module, called Quantum.

Quantum is by far my favorite part of the Infinity Engine.

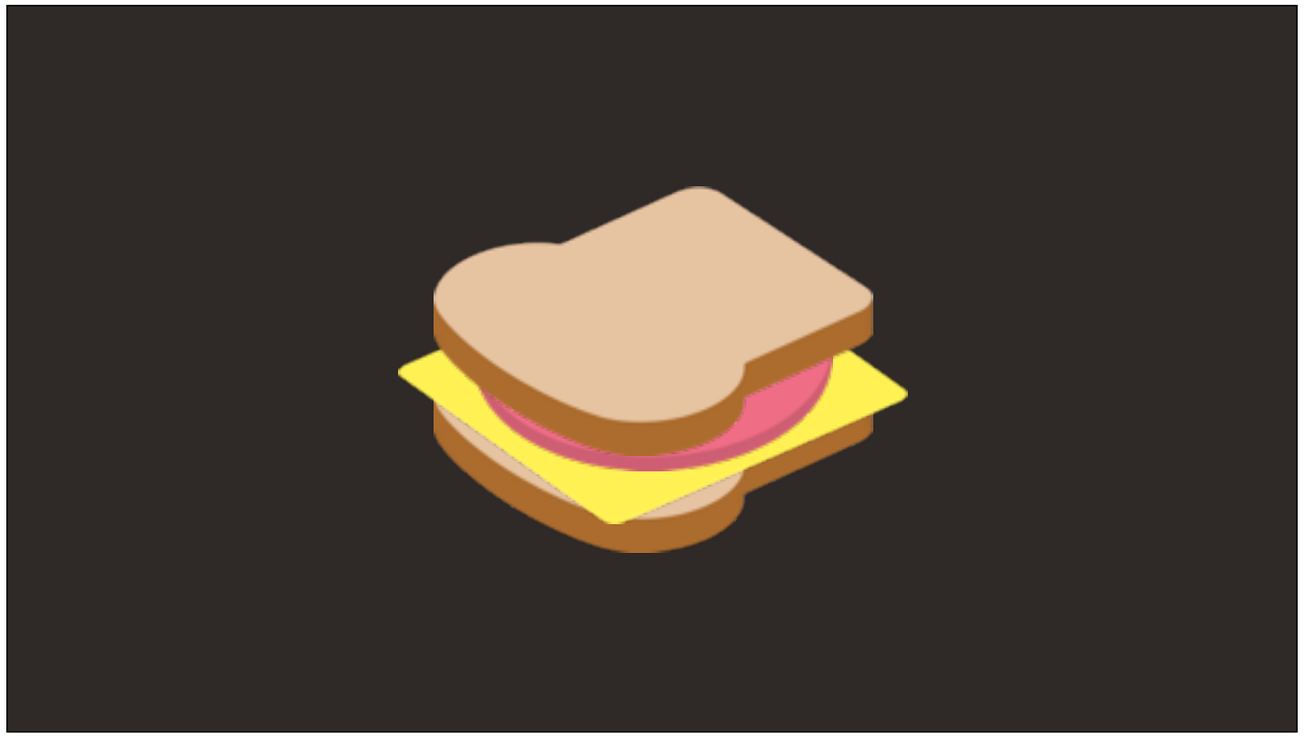


Quantum is by far my favorite part of the Infinity Engine.

A breathtakingly unique approach to computation, all physical interactions are handled by directly interfacing Photon, Spacetime, and Matter at the assembly level.

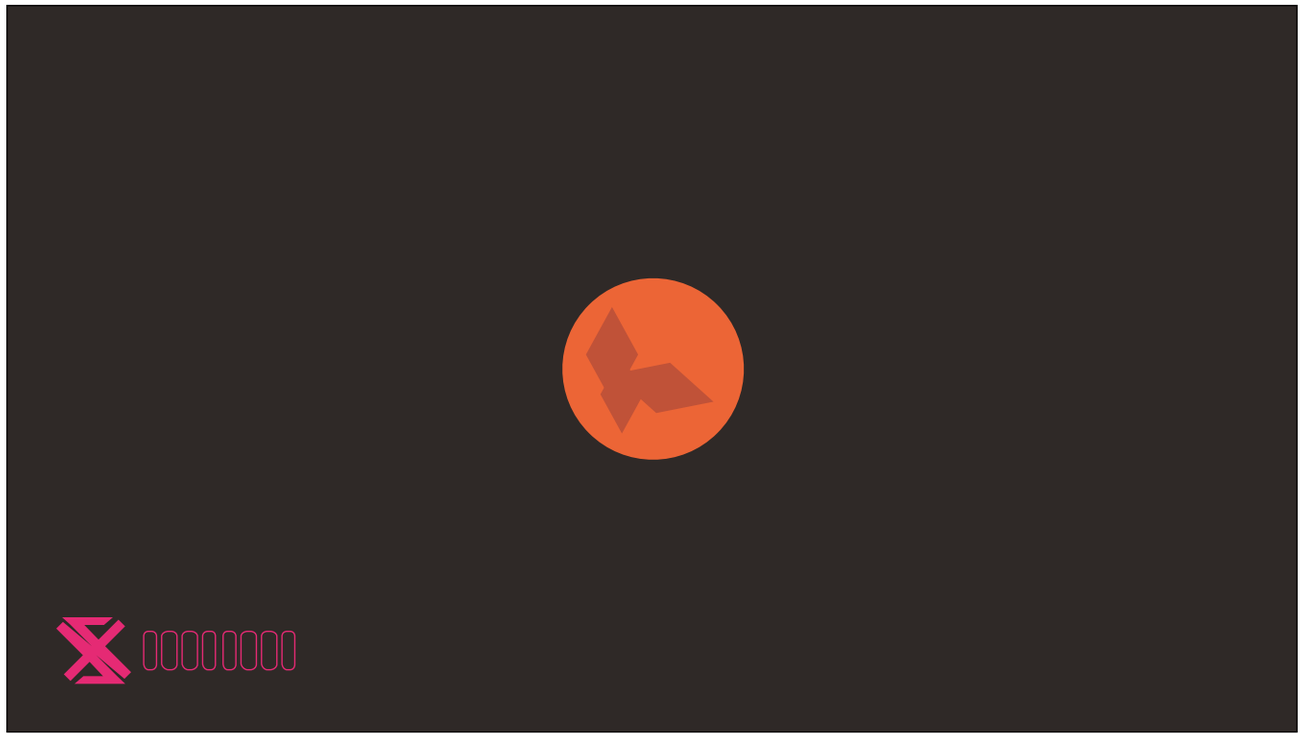
For instance, the photons used in the rendering system are actually small bits of matter, spawned from energy allocated by other piece of matter.

More and more complicated systems have been built on top of surprisingly few fundamental interactions — it’s nice that you don’t have to directly address a few billion particles every time you want to, say, make a sandwich.



More and more complicated systems have been built on top of surprisingly few fundamental interactions — it’s nice that you don’t have to directly address a few billion particles every time you want to, say, make a sandwich.

Quantum runs the very base ruleset that keeps the simulation going.

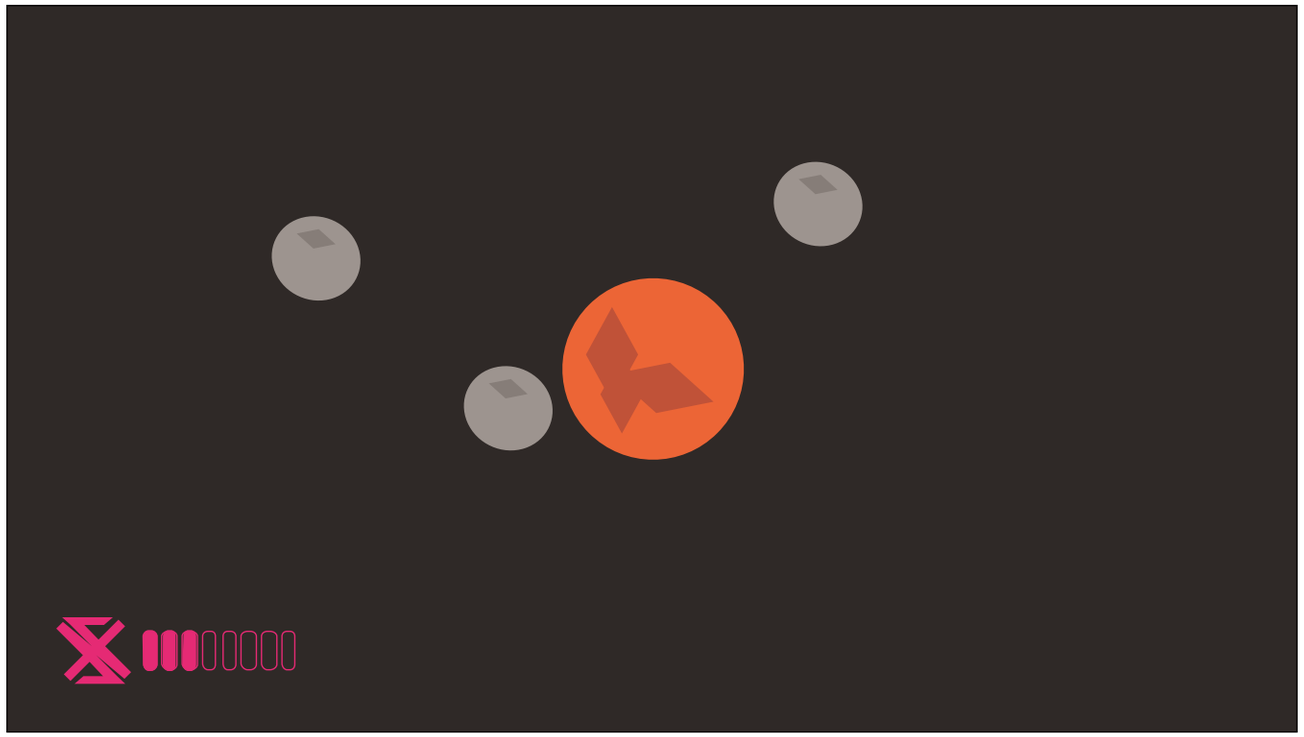


Quantum runs the very base ruleset that keeps the simulation going.

In fact, gravity is sort of a side effect of this computation.

As matter occupies space near other matter, more interactions need to be calculated, keeping the matter near each other for longer periods of time.

To me, this sounds like a situation that could easily lead to an infinite loop. So here we find the last seam we're going to look at today.

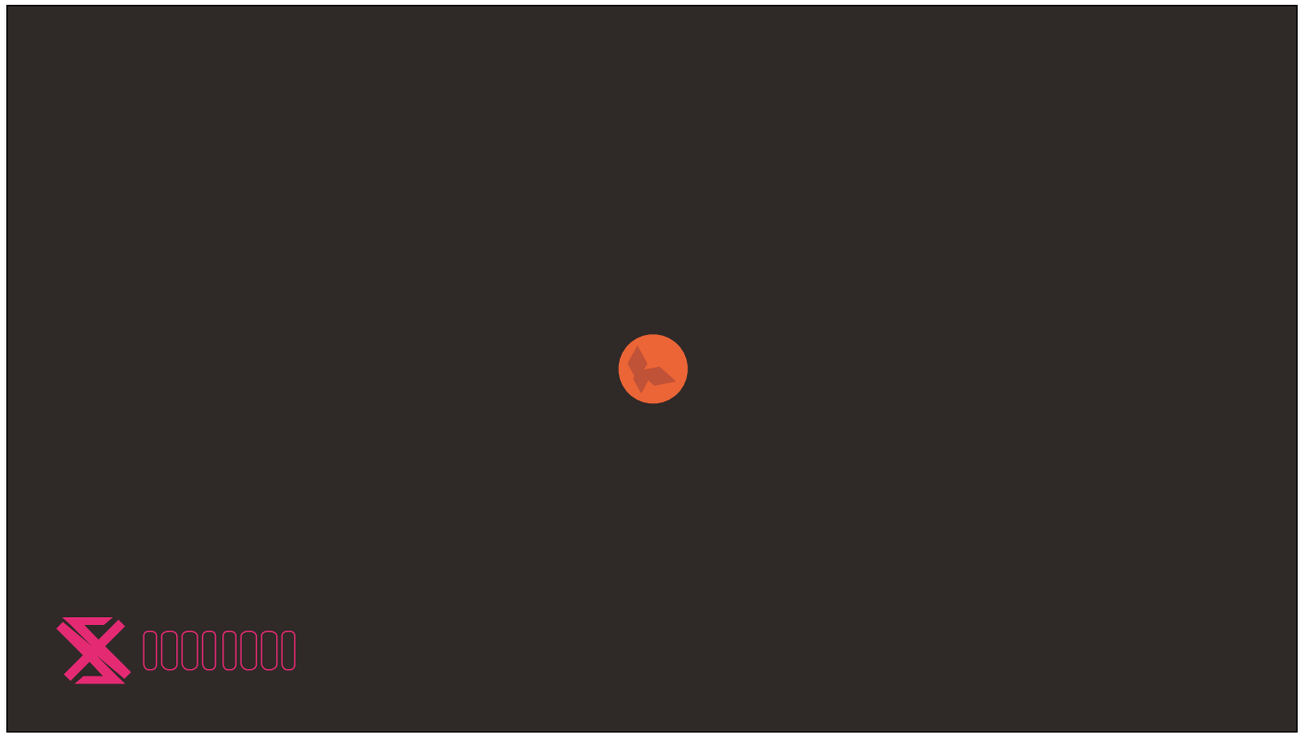


Quantum runs the very base ruleset that keeps the simulation going.

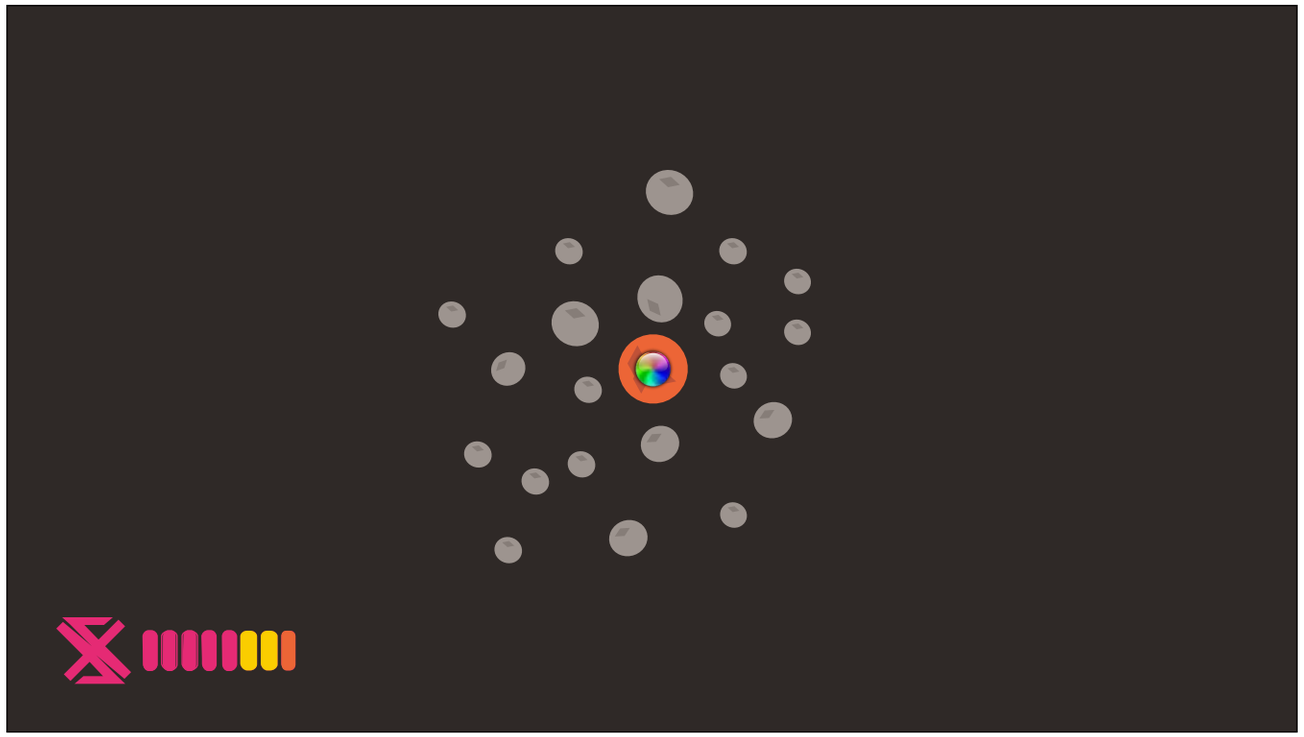
In fact, gravity is sort of a side effect of this computation.

As matter occupies space near other matter, more interactions need to be calculated, keeping the matter near each other for longer periods of time.

To me, this sounds like a situation that could easily lead to an infinite loop. So here we find the last seam we're going to look at today.



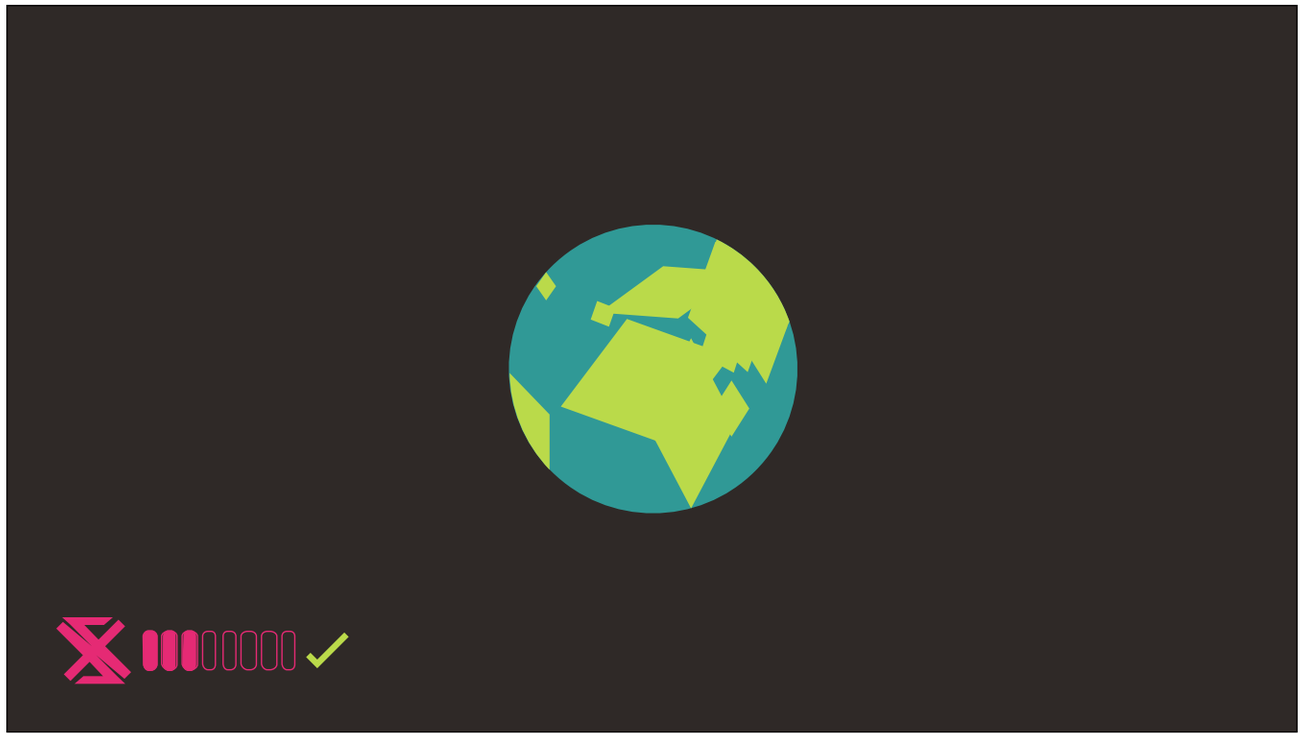
To me, this sounds like a situation that could easily lead to an infinite loop. So here we find the last seam we’re going to look at today. Is it possible to put so much matter in one location that we crash the Quantum computation engine?



To me, this sounds like a situation that could easily lead to an infinite loop. So here we find the last seam we’re going to look at today. Is it possible to put so much matter in one location that we crash the Quantum computation engine?

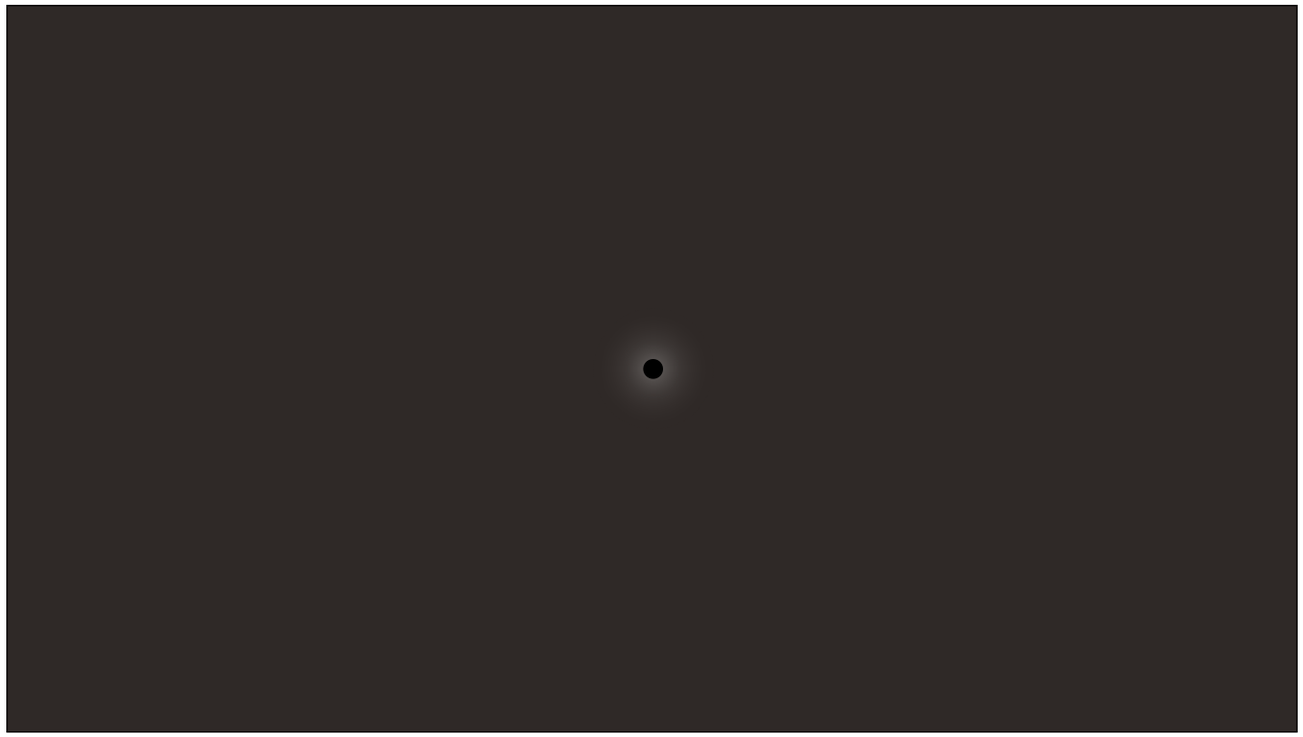
It’s not possible on Earth, simply because there’s already quite a bit of Matter here and it seems to be processing just fine.

But in the depths of space, we’ve been able to find impossibly massive objects that seem to answer the question for us. What happens when too much matter occupies one space?



It’s not possible on Earth, simply because there’s already quite a bit of Matter here and it seems to be processing just fine.

But in the depths of space, we’ve been able to find impossibly massive objects that seem to answer the question for us. What happens when too much matter occupies one space?

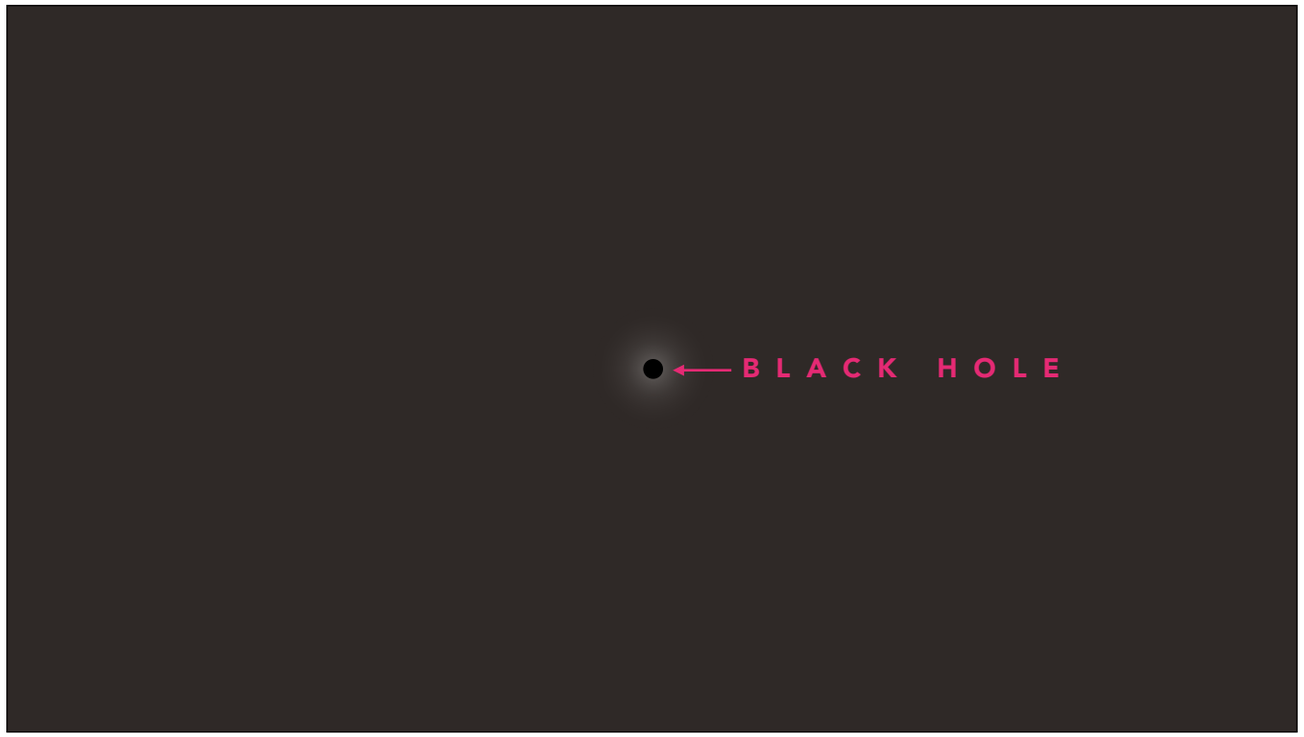


Essentially, dead pixels.

A Scientist by the name of Stephen Hawking gave them a more foreboding name: a black hole, because photons never leave these areas. This areas require so many calculations that they effectively have a sort of boundary, referred to as the “event horizon”, and any objects that enter are lost to the infinite loop, presumably forever.

And yet, the game keeps on ticking. Why?

Well, if you think this looks suspiciously similar to our “rendering bubble” from earlier, you’re not alone.

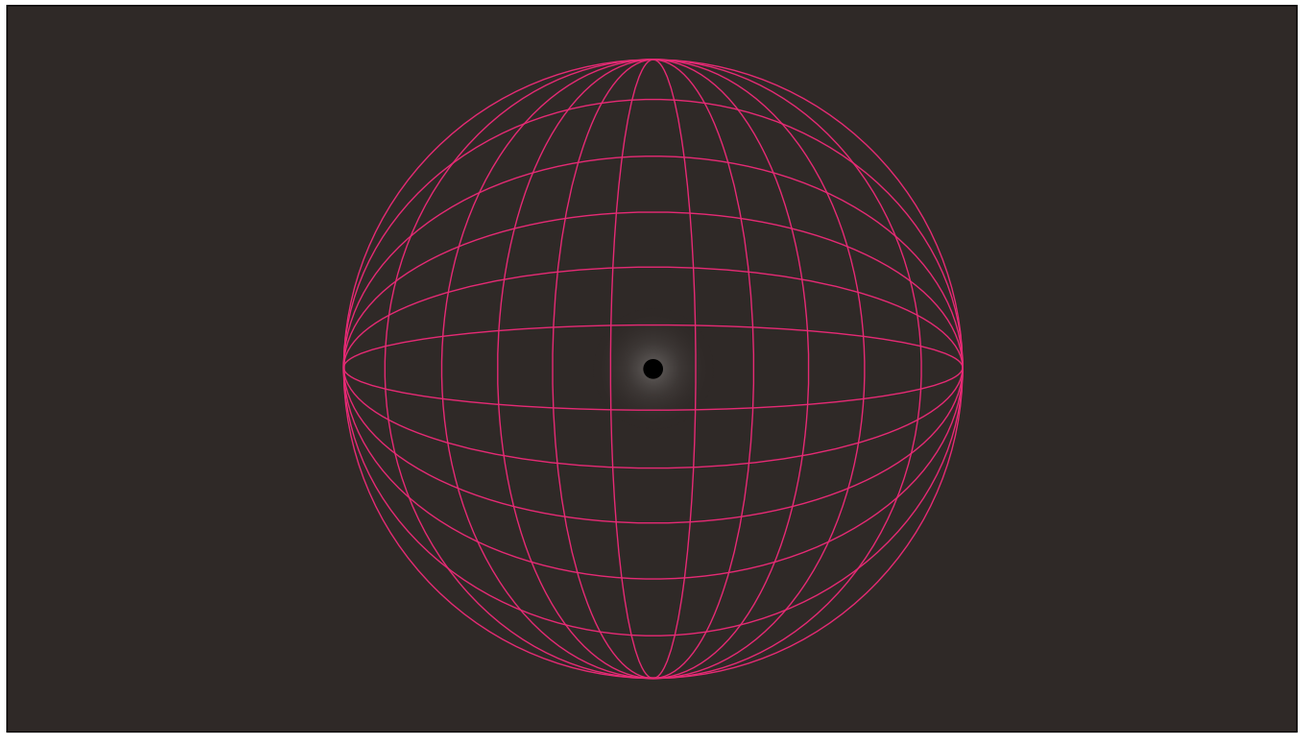


Essentially, dead pixels.

A Scientist by the name of Stephen Hawking gave them a more foreboding name: a black hole, because photons never leave these areas. This areas require so many calculations that they effectively have a sort of boundary, referred to as the “event horizon”, and any objects that enter are lost to the infinite loop, presumably forever.

And yet, the game keeps on ticking. Why?

Well, if you think this looks suspiciously similar to our “rendering bubble” from earlier, you’re not alone.

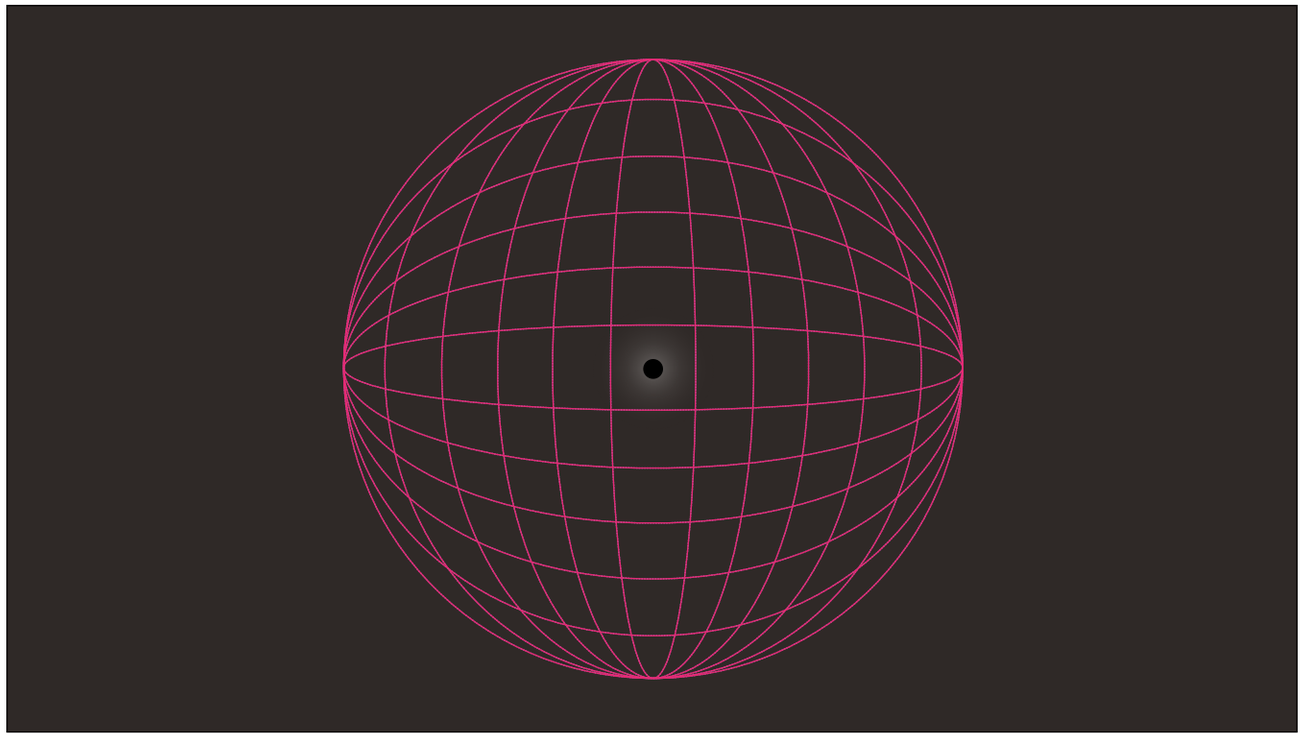


Essentially, dead pixels.

A Scientist by the name of Stephen Hawking gave them a more foreboding name: a black hole, because photons never leave these areas. This areas require so many calculations that they effectively have a sort of boundary, referred to as the “event horizon”, and any objects that enter are lost to the infinite loop, presumably forever.

And yet, the game keeps on ticking. Why?

Well, if you think this looks suspiciously similar to our “rendering bubble” from earlier, you’re not alone.

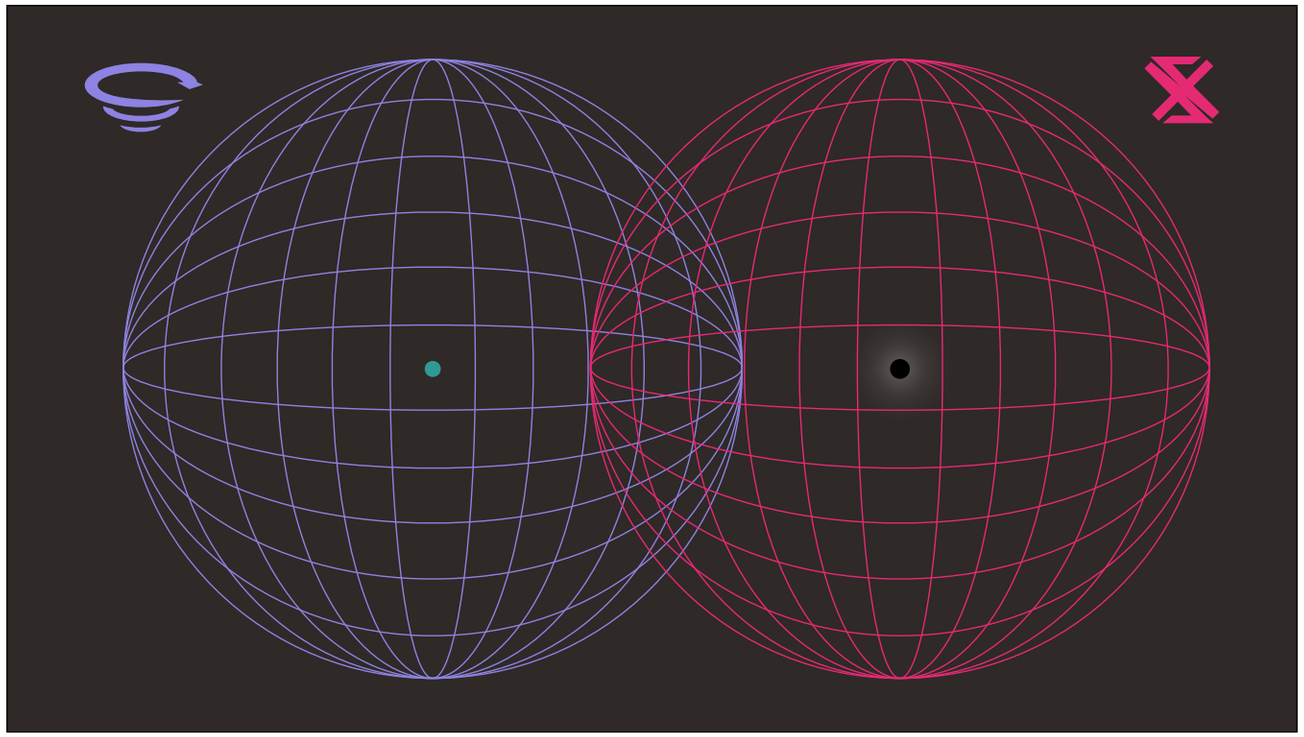


Essentially, dead pixels.

A Scientist by the name of Stephen Hawking gave them a more foreboding name: a black hole, because photons never leave these areas. This areas require so many calculations that they effectively have a sort of boundary, referred to as the “event horizon”, and any objects that enter are lost to the infinite loop, presumably forever.

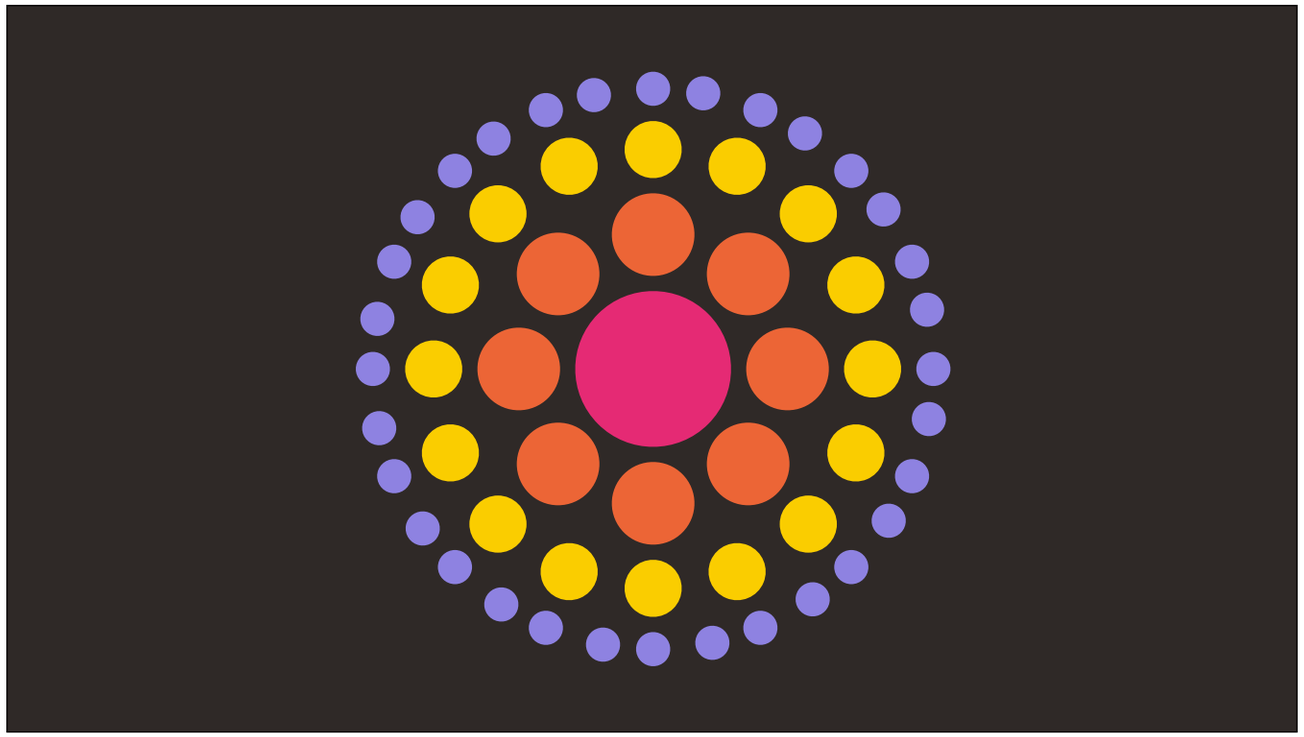
And yet, the game keeps on ticking. Why?

Well, if you think this looks suspiciously similar to our “rendering bubble” from earlier, you’re not alone.

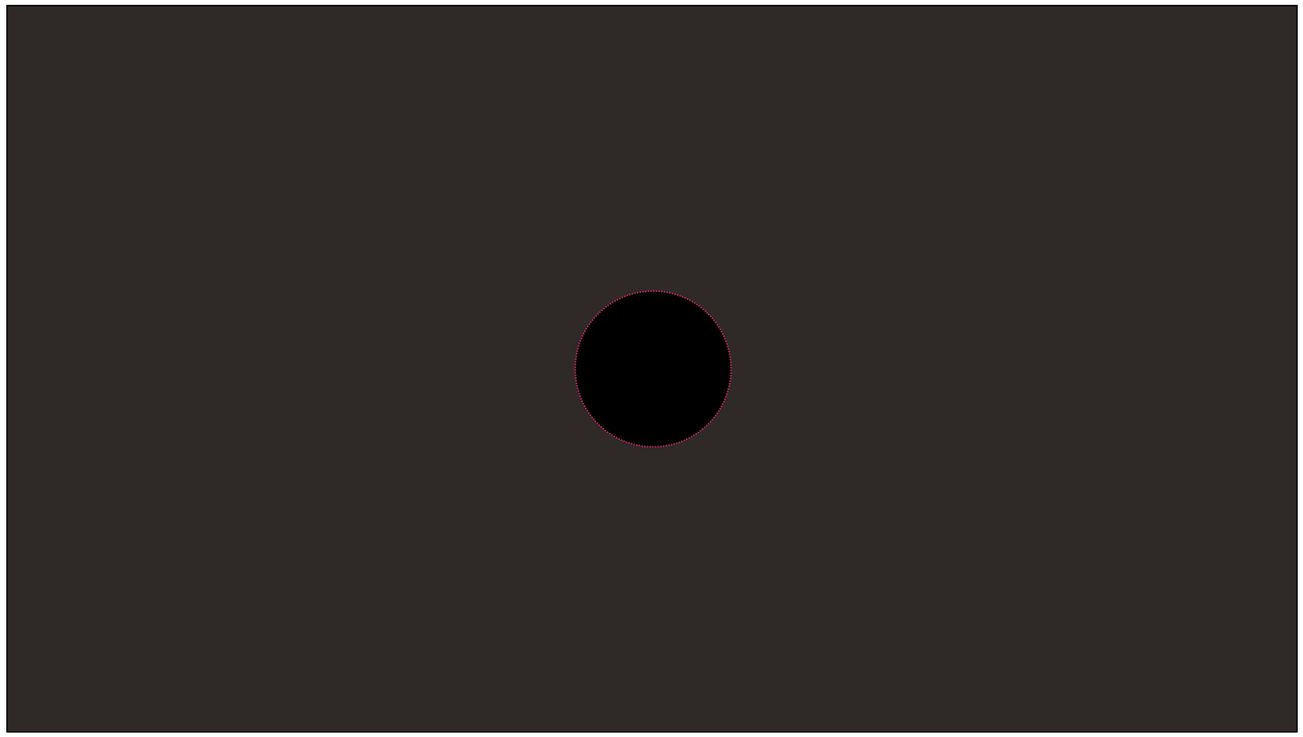


Well, if you think this looks suspiciously similar to our “rendering bubble” from earlier, you’re not alone.

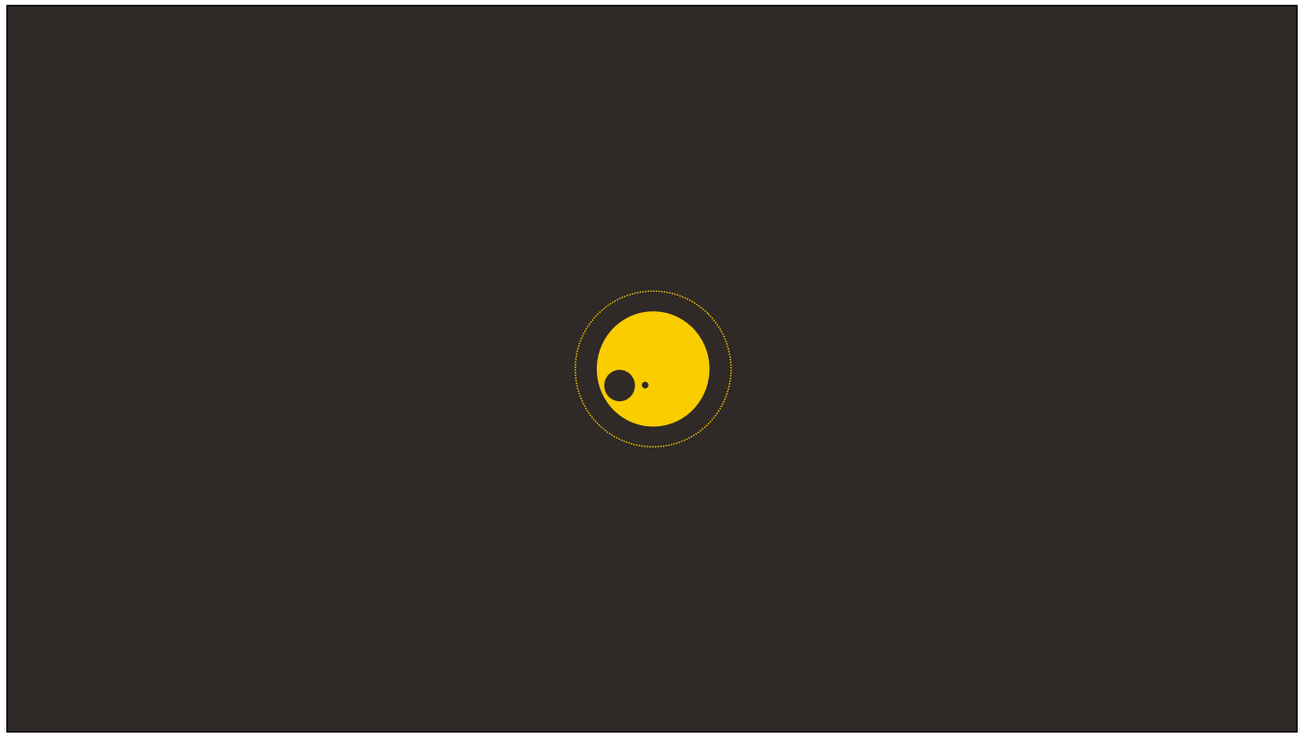
Let’s face it — the more matter you have crammed together, the more interactions you need to calculate.



If that number passes a certain threshold, it makes sense to spin up a separate server to handle the extra load. And it makes even more sense to just run a new copy of The Universe on that server, since it already has all of the logic needed to run exactly this sort of simulation.

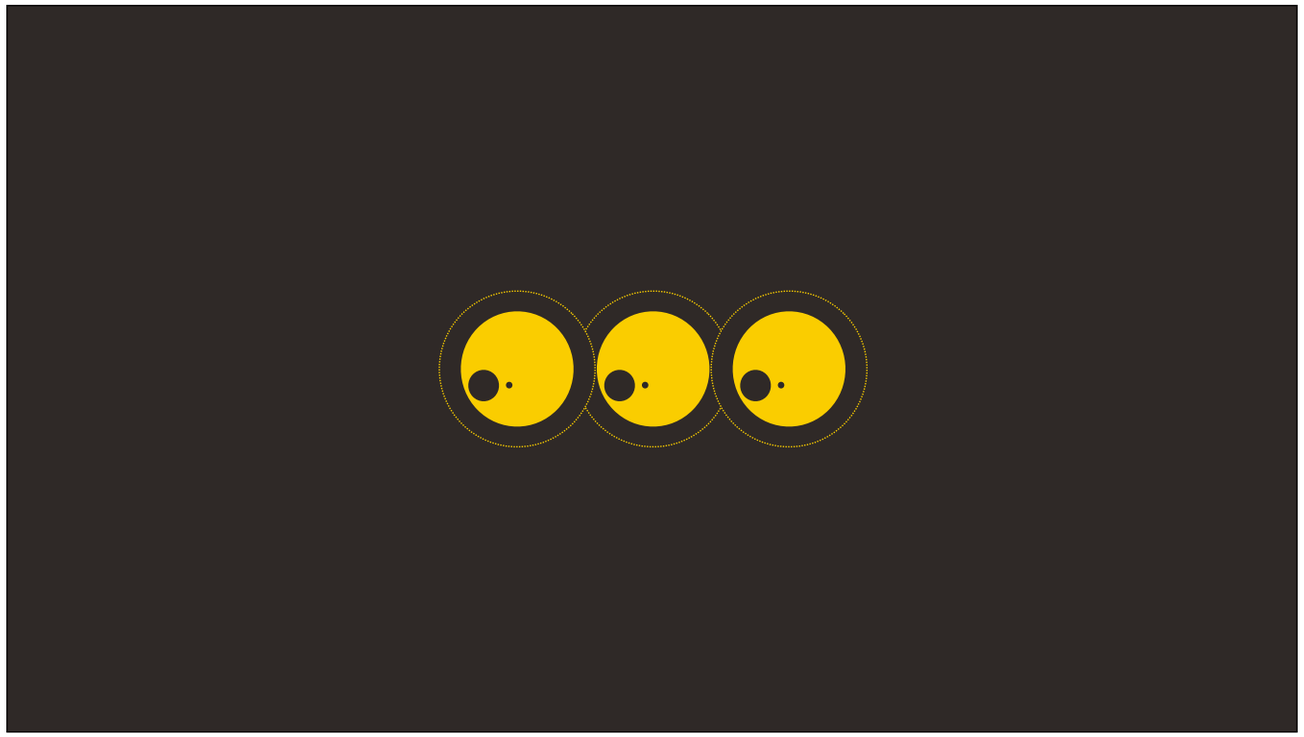


If that number passes a certain threshold, it makes sense to spin up a separate server to handle the extra load. And it makes even more sense to just run a new copy of The Universe on that server, since it already has all of the logic needed to run exactly this sort of simulation.

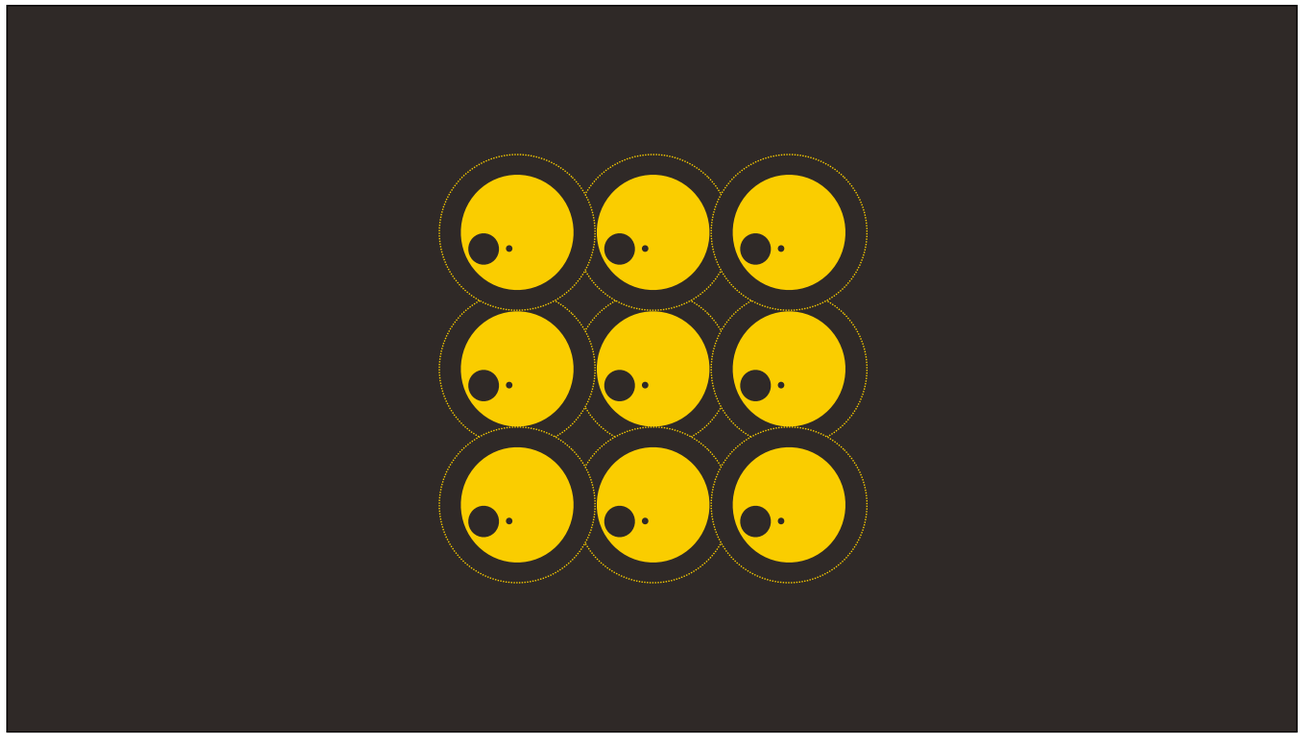


If that number passes a certain threshold, it makes sense to spin up a separate server to handle the extra load. And it makes even more sense to just run a new copy of The Universe on that server, since it already has all of the logic needed to run exactly this sort of simulation.

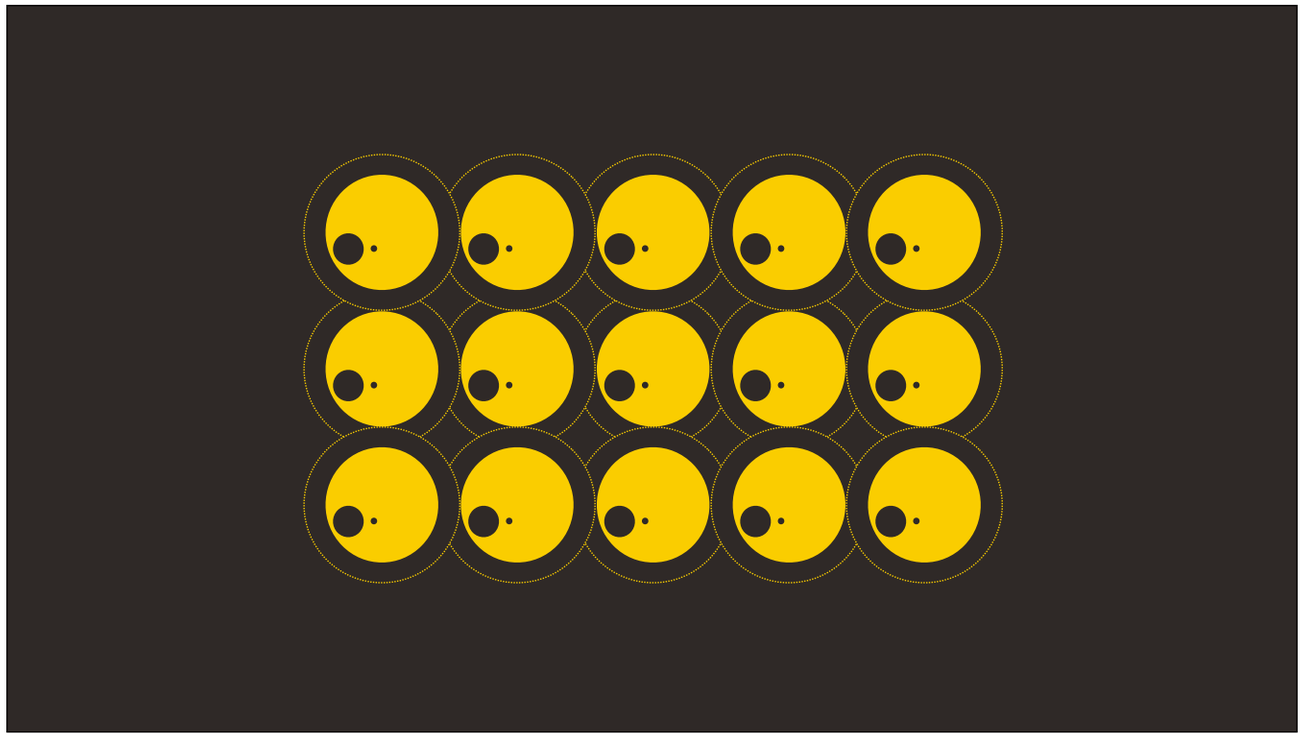
Picture it. Multiple Servers.



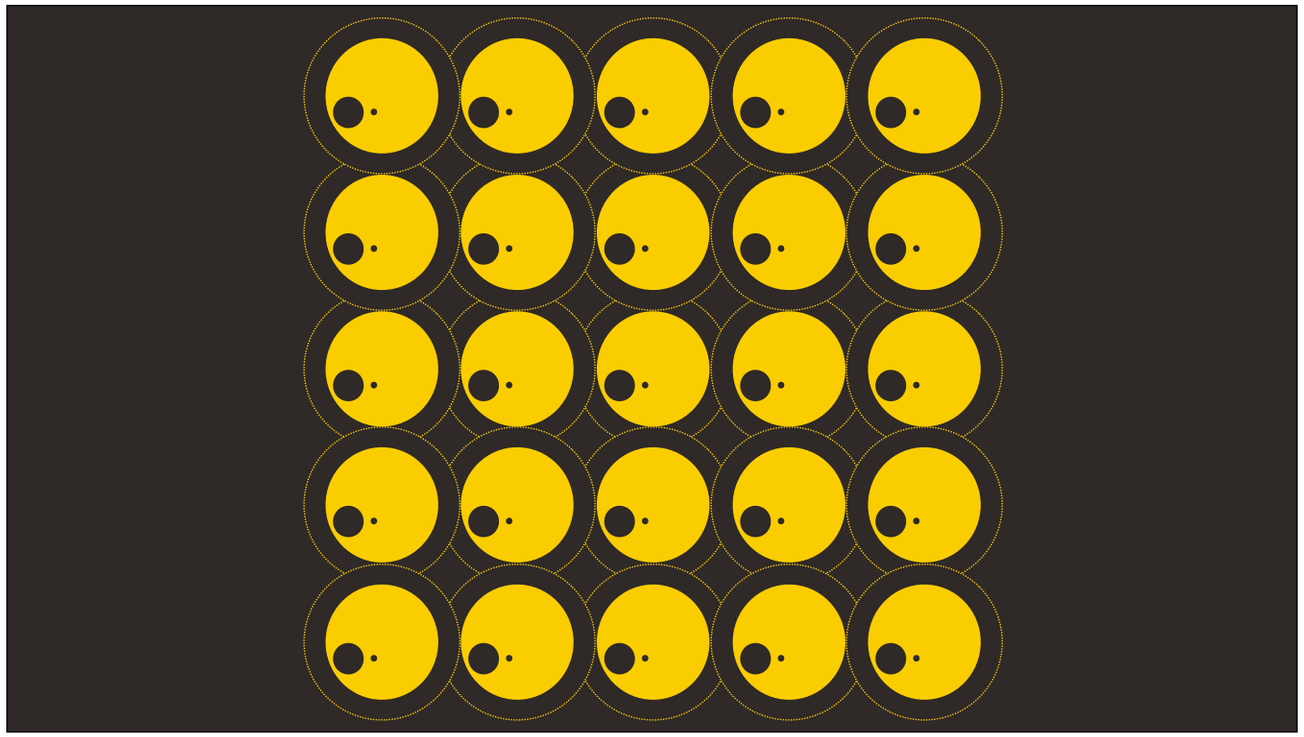
Picture it: Multiple servers, each running a recursive instance of The Universe, seeded with a starting mass and left to blossom. Each new instance a beautiful reflection of the things that were and the things that will be; servers all the way down.



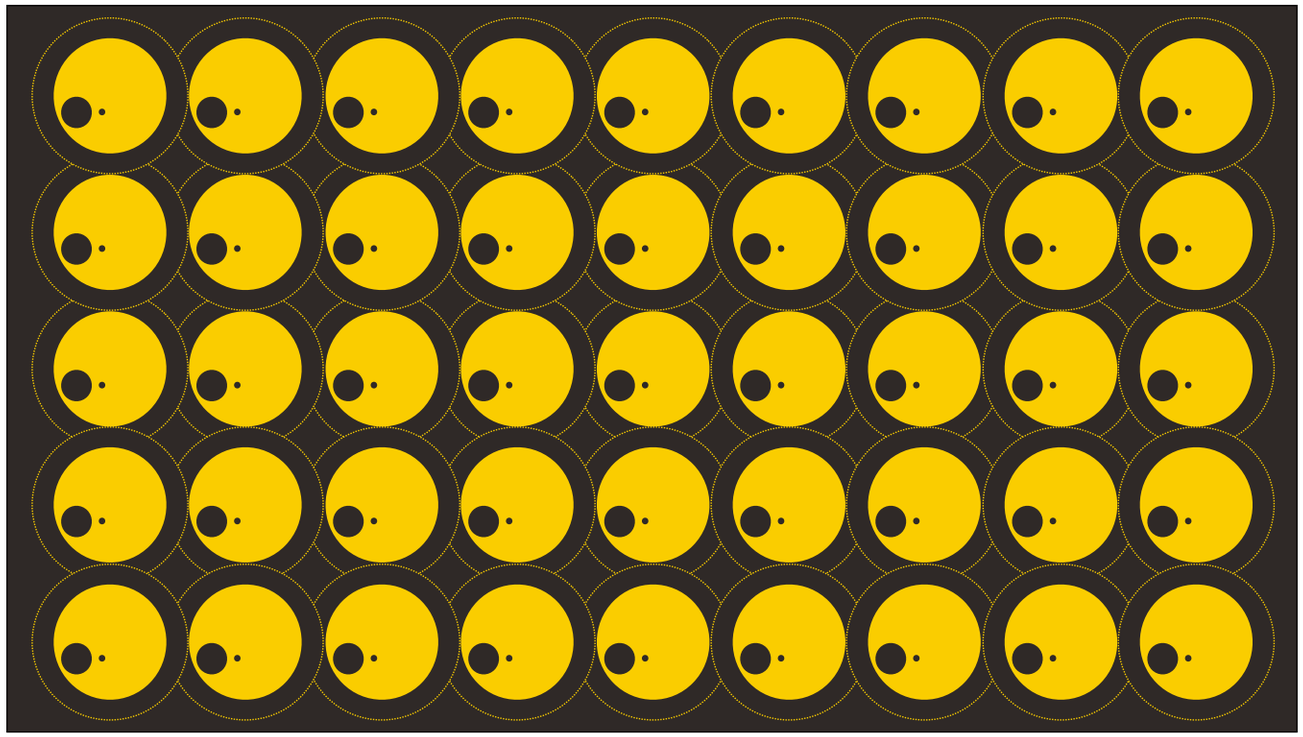
Picture it: Multiple servers, each running a recursive instance of The Universe, seeded with a starting mass and left to blossom. Each new instance a beautiful reflection of the things that were and the things that will be; servers all the way down.



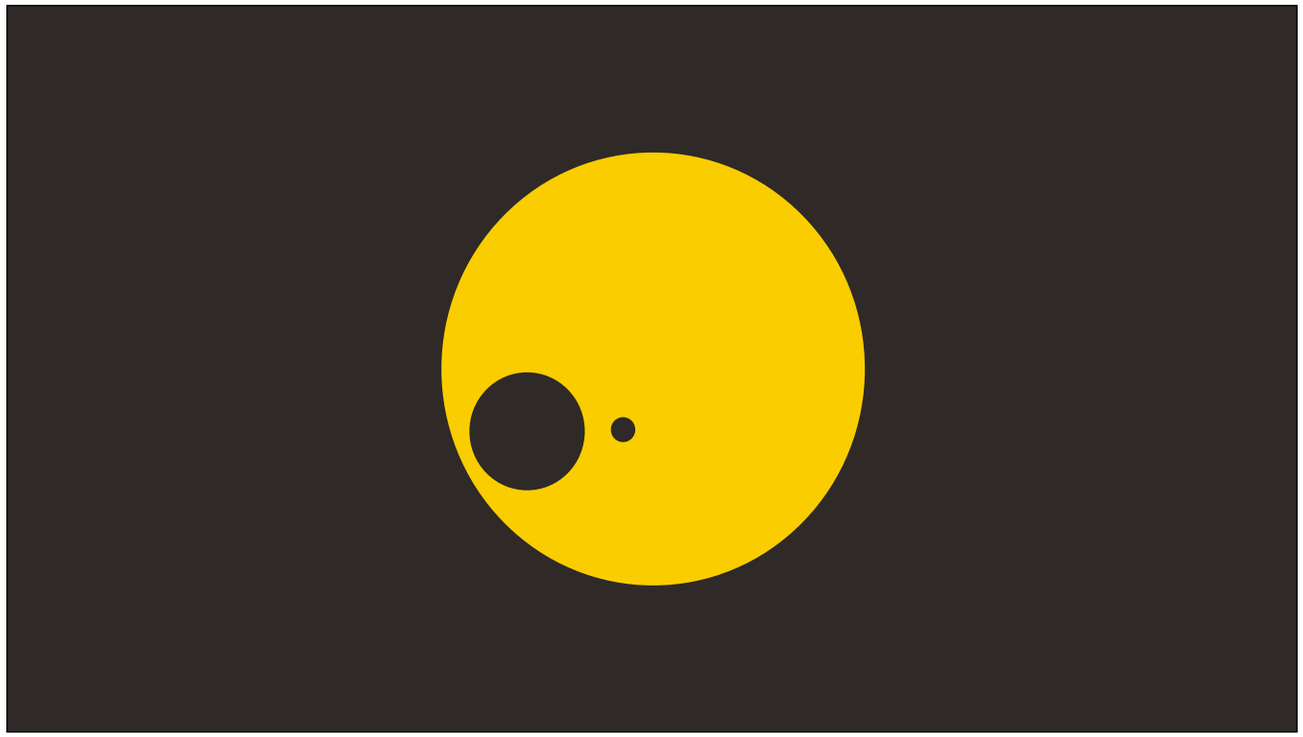
Picture it: Multiple servers, each running a recursive instance of The Universe, seeded with a starting mass and left to blossom. Each new instance a beautiful reflection of the things that were and the things that will be; servers all the way down.



Picture it: Multiple servers, each running a recursive instance of The Universe, seeded with a starting mass and left to blossom. Each new instance a beautiful reflection of the things that were and the things that will be; servers all the way down.



Picture it: Multiple servers, each running a recursive instance of The Universe, seeded with a starting mass and left to blossom. Each new instance a beautiful reflection of the things that were and the things that will be; servers all the way down.



Picture it: Multiple servers, each running a recursive instance of The Universe, seeded with a starting mass and left to blossom. Each new instance a beautiful reflection of the things that were and the things that will be; servers all the way down.

PART THREE

LESSONS

So what have we learned? Obviously there is so much more to cover, but we only have so much time. So let’s review:

1. Use limitations creatively

The speed of light, “c”, is the major limitation of everything that runs in the Universe, and poorly planning around it could have been disastrous. But from the weird mass hack to the maximum draw distance, the developers consistently found creative ways to use the limitations to their advantage. Limitations breed creativity — embrace them early and often.

2. Control your scope

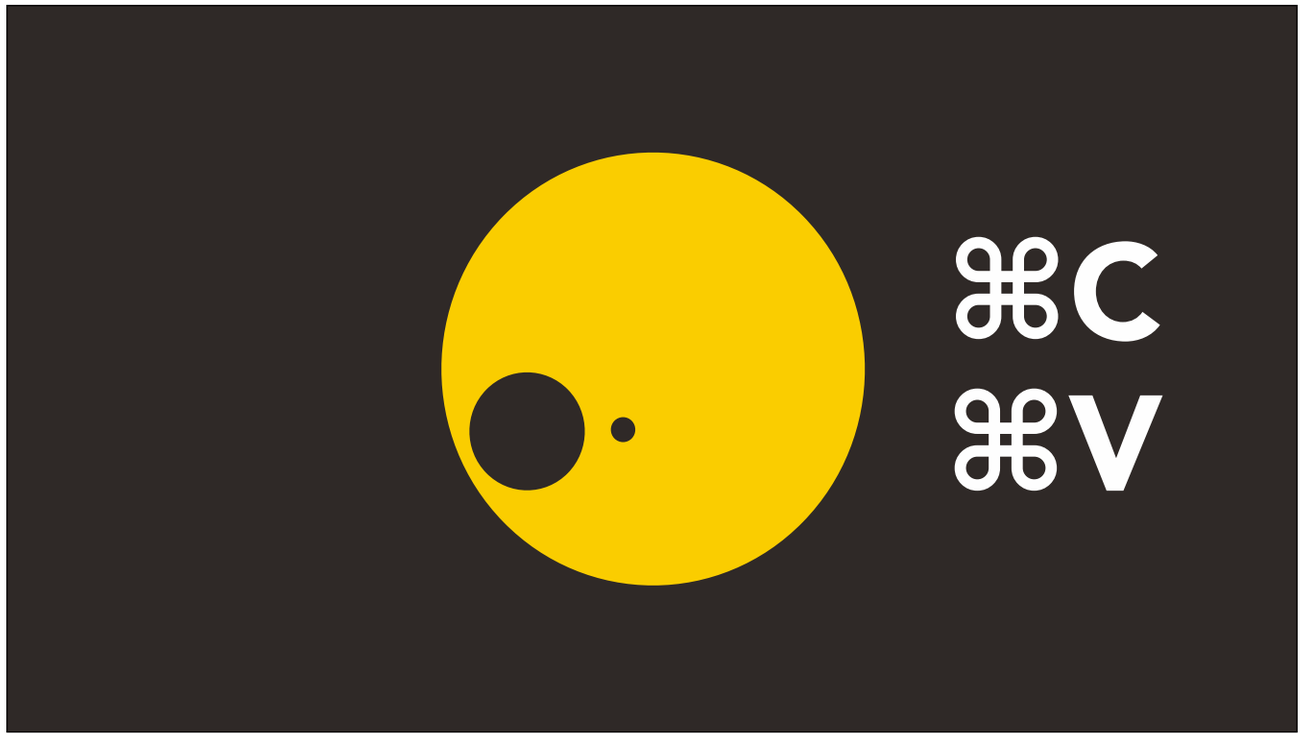
The Universe advertises infinity, but in practice, it settles for “very big.” Likewise, when making your games, keep a clear idea of what you’re trying to achieve. It’s really easy to try and make your game all things to all people, and it’s natural to want to add every feature.But remember — even something as expansive as the Universe knows when things are simply out of scope.

3. Don’t sweat the small stuff

The weird precision bugs that pop up from time to time are frustrating, and I know many a programmer that would work well into the night to try and fix it. But remember to look at the big picture — at the end of the day, if it doesn’t make the game worse, it probably doesn’t need fixing.

4. Embrace emergent phenomena

In games, you have so many distinct systems interacting in so many ways that it can be hard to predict what will happen. And as game designers, we'll often try and enact exacting control over every aspect of the experiences we make. But sometimes you have to take a step back and let the game become what it wants to become. It may surprise you.



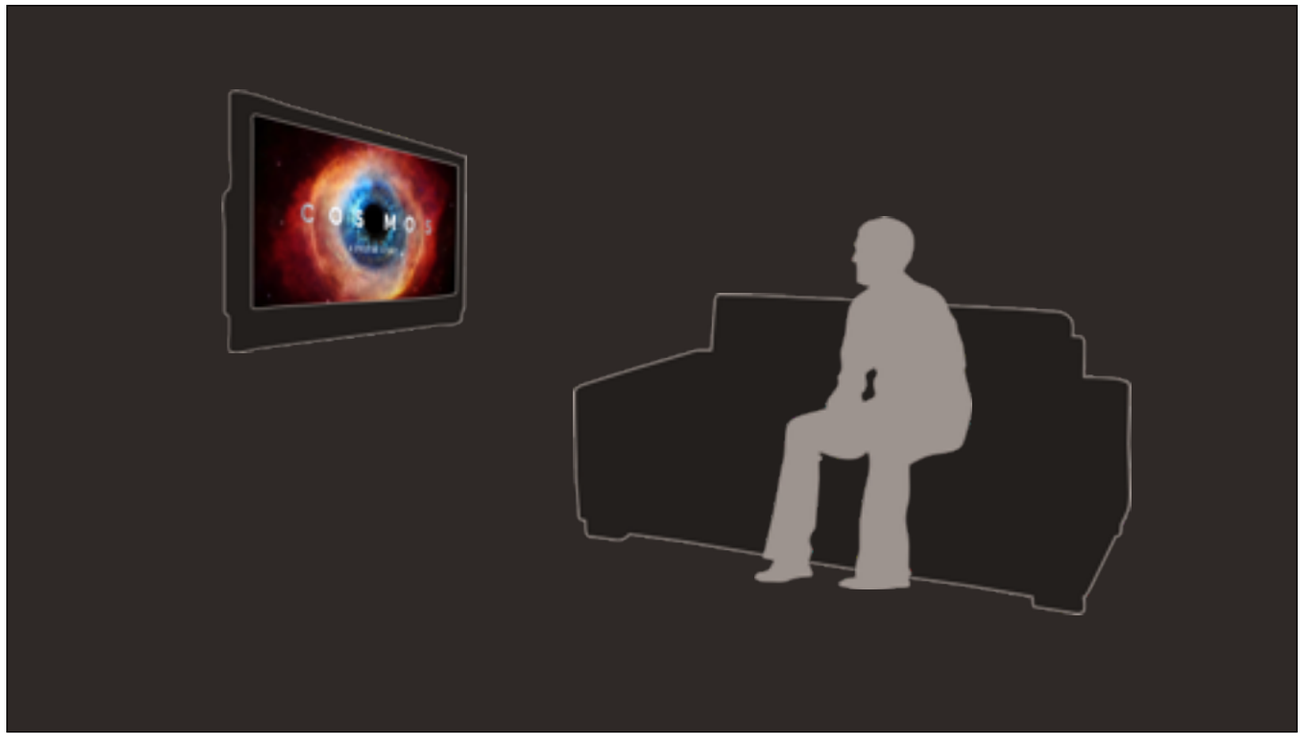
The great thing about the Universe is that its literally everywhere. And as far as I know, the Universe is open source — or at least, it has yet to sue anybody. So the next time you run into a game design problem you're struggling to fix, do what I do: steal from the Universe.

If I may, I’d like to switch gears briefly and talk about what drove me to write this talk.

It wasn’t the game industry, an over-hyped game, or a console launch.

It wasn’t a release schedule, and it damn sure wasn’t GamerGate.

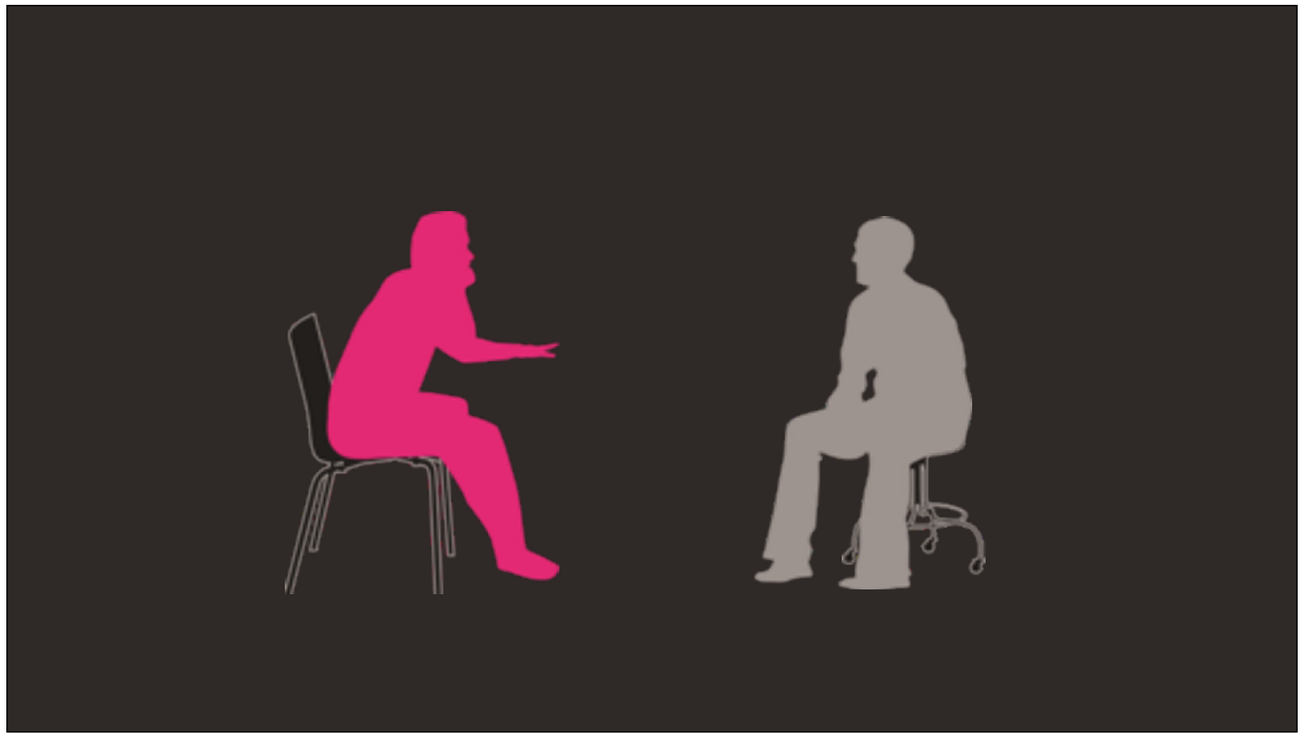
It was an episode of Cosmos;



It was an episode of Cosmos;

a conversation with a friend.

It was a walk through the city on a fall afternoon that got me thinking about all the things I normally wouldn’t.



It was an episode of Cosmos;

a conversation with a friend.

It was a walk through the city on a fall afternoon that got me thinking about all the things I normally wouldn’t.

It was an episode of Cosmos;

a conversation with a friend.

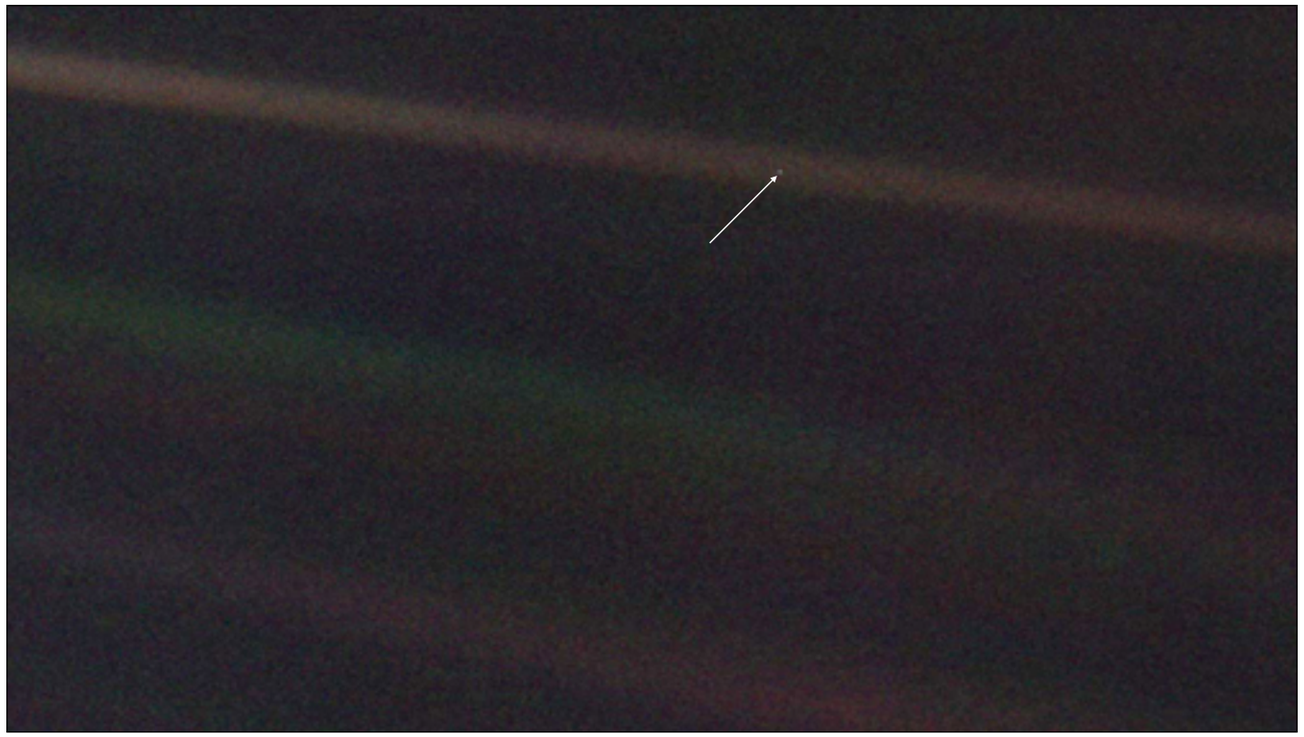
It was a walk through the city on a fall afternoon that got me thinking about all sorts of things I normally wouldn’t.

And that’s really what I came to talk to you about today — the things that inspire us to create.

Earth is our entire world — but it is only a microscopic piece, of a minuscule galaxy, in the tiniest fraction of the universe.

And the games industry is tinier still.

Simply put, there’s so much more to life than games.



Earth is our entire world — but it is only a microscopic piece, of a minuscule galaxy, in the tiniest fraction of the universe.

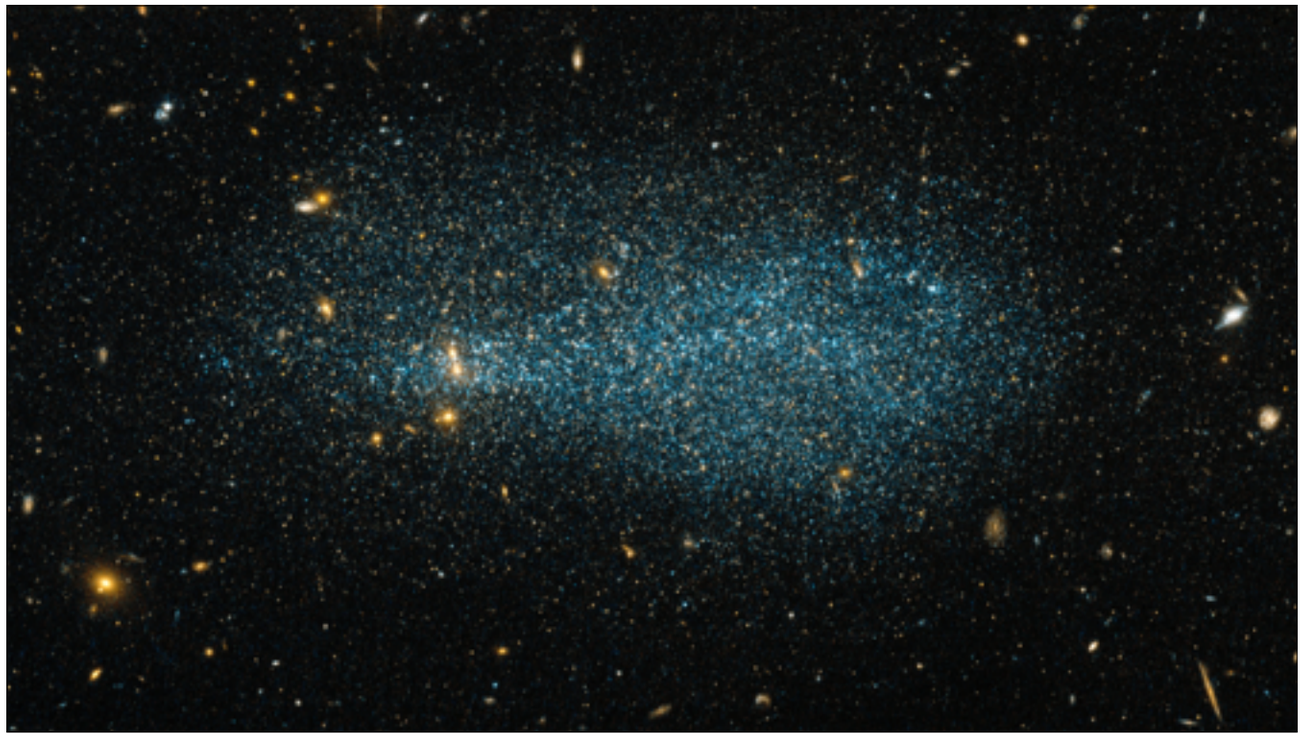
And the games industry is tinier still.

Simply put, there’s so much more to life than games.

Earth is our entire world — but it is only a microscopic piece, of a minuscule galaxy, in the tiniest fraction of the universe.

And the games industry is tinier still.

Simply put, there’s so much more to life than games.



Earth is our entire world — but it is only a microscopic piece, of a minuscule galaxy, in the tiniest fraction of the universe.

And the games industry is tinier still.

Simply put, there’s so much more to life than games.

Don’t let the commenters fool you. People don’t give two shits about Frame rates

Earth is our entire world — but it is only a microscopic piece, of a minuscule galaxy, in the tiniest fraction of the universe.

And the games industry is tinier still.

Simply put, there’s so much more to life than games.

Don’t let the commenters fool you. People don’t give two shits about Frame rates

Don’t let the commenters fool you. People don’t give two shits about frame rates, DLC, DRM, or in-app purchases.

They care about having an experience that matters — and the only way for you to give it to them is to get out there and have one yourselves.

Don’t let the commenters fool you. People don’t give two shits about frame rates, DLC, DRM, or in-app purchases.

They care about having an experience that matters — and the only way for you to give it to them is to get out there and have one yourselves.

Don’t let the commenters fool you. People don’t give two shits about frame rates, DLC, DRM, or in-app purchases.

They care about having an experience that matters — and the only way for you to give it to them is to get out there and have one yourselves.

Don’t let the commenters fool you. People don’t give two shits about frame rates, DLC, DRM, or in-app purchases.

They care about having an experience that matters — and the only way for you to give it to them is to get out there and have one yourselves.

Don’t let the commenters fool you. People don’t give two shits about frame rates, DLC, DRM, or in-app purchases.

They care about having an experience that matters — and the only way for you to give it to them is to get out there and have some yourselves.

So go. Be inspired. Level up. Do some quests. Expand your skill tree. Get every last achievement.

So go. Be inspired. Level up. Do some quests. Expand your skill tree. Get every last achievement.

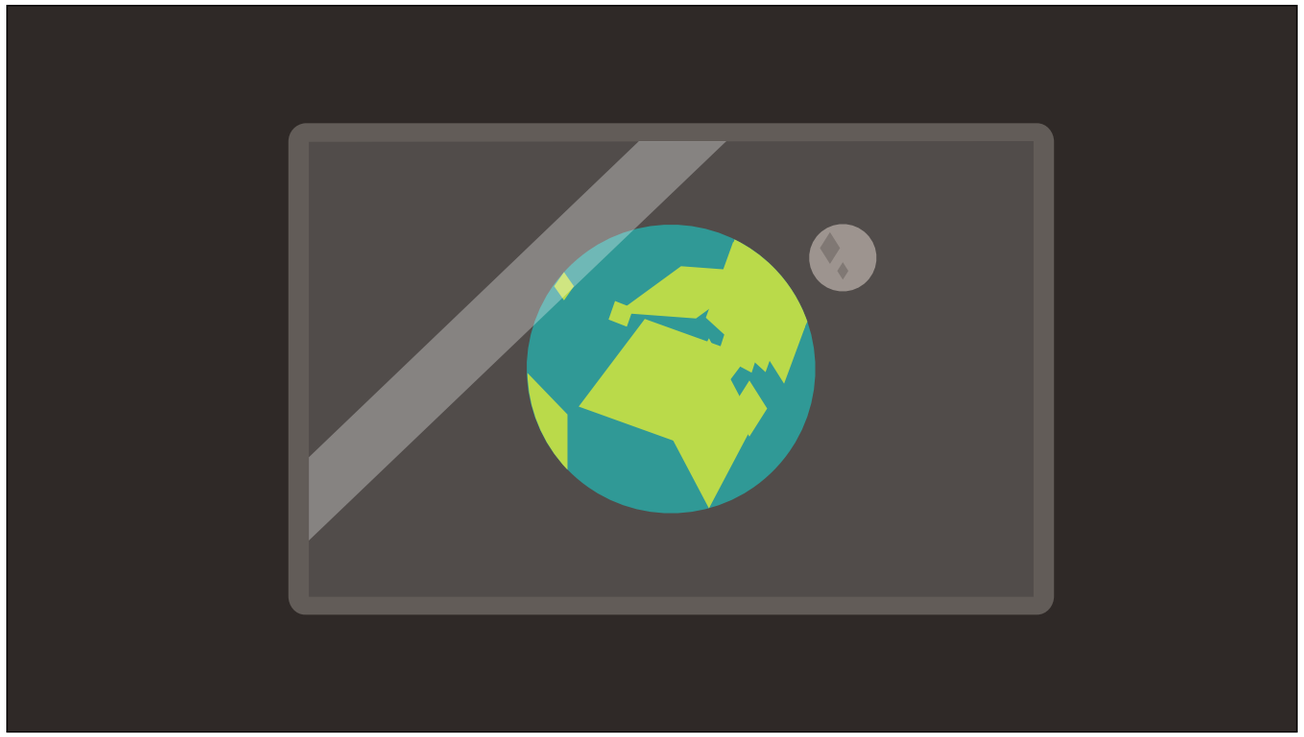
So go. Be inspired. Level up. Do some quests. Expand your skill tree. Get every last achievement.

So go. Be inspired. Level up. Do some quests. Expand your skill tree. Get every last achievement.

So go. Be inspired. Level up. Do some quests. Expand your skill tree. Get every last achievement.

So go. Be inspired. Level up. Do some quests. Expand your skill tree. Get every last achievement.

Because if it is true that our entire reality — everything we ever have or ever will know — is a game…



Because if it is true that our entire reality — everything we ever have or ever will know — is a game…

Because if it is true that our entire reality — everything we ever have or ever will know — is a game…

Shouldn’t we be playing more of it?


THE UNIVERSE
HOW TO BREAK THE GAME
@dainsaint • @cipherprime