A Scalable Server for 3D Metaverses

Ewen Cheslack-Postava, Tahir Azim, Behram F.T. Mistree, Daniel Reiter Horn, Jeff Terrace, Philip Levis, and Michael J. Freedman

sirikata.com

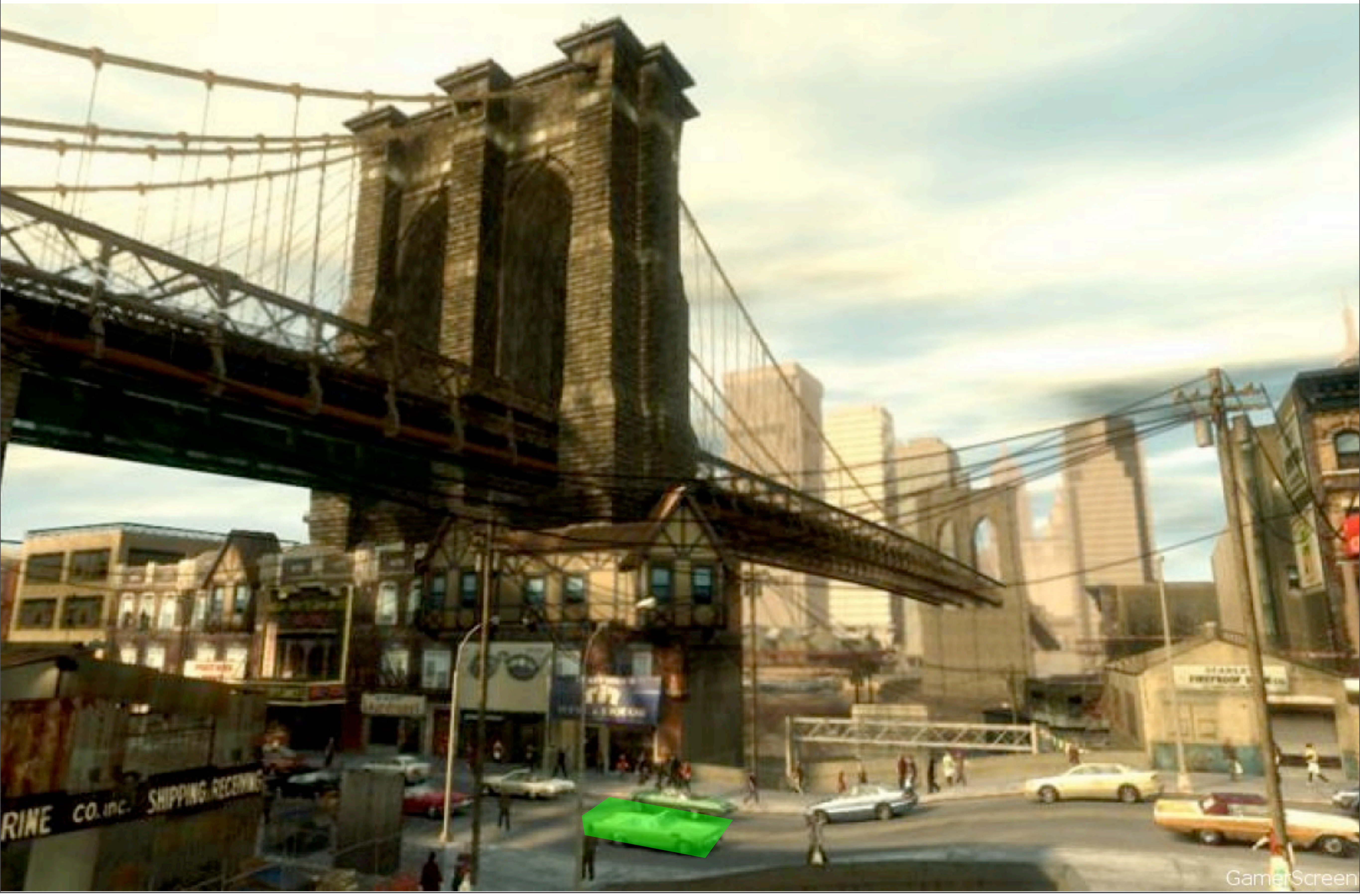
Metaverses

Metaverses are shared 3D spaces with the unique feature that everything is editable and and scriptable by users...

Metaverses

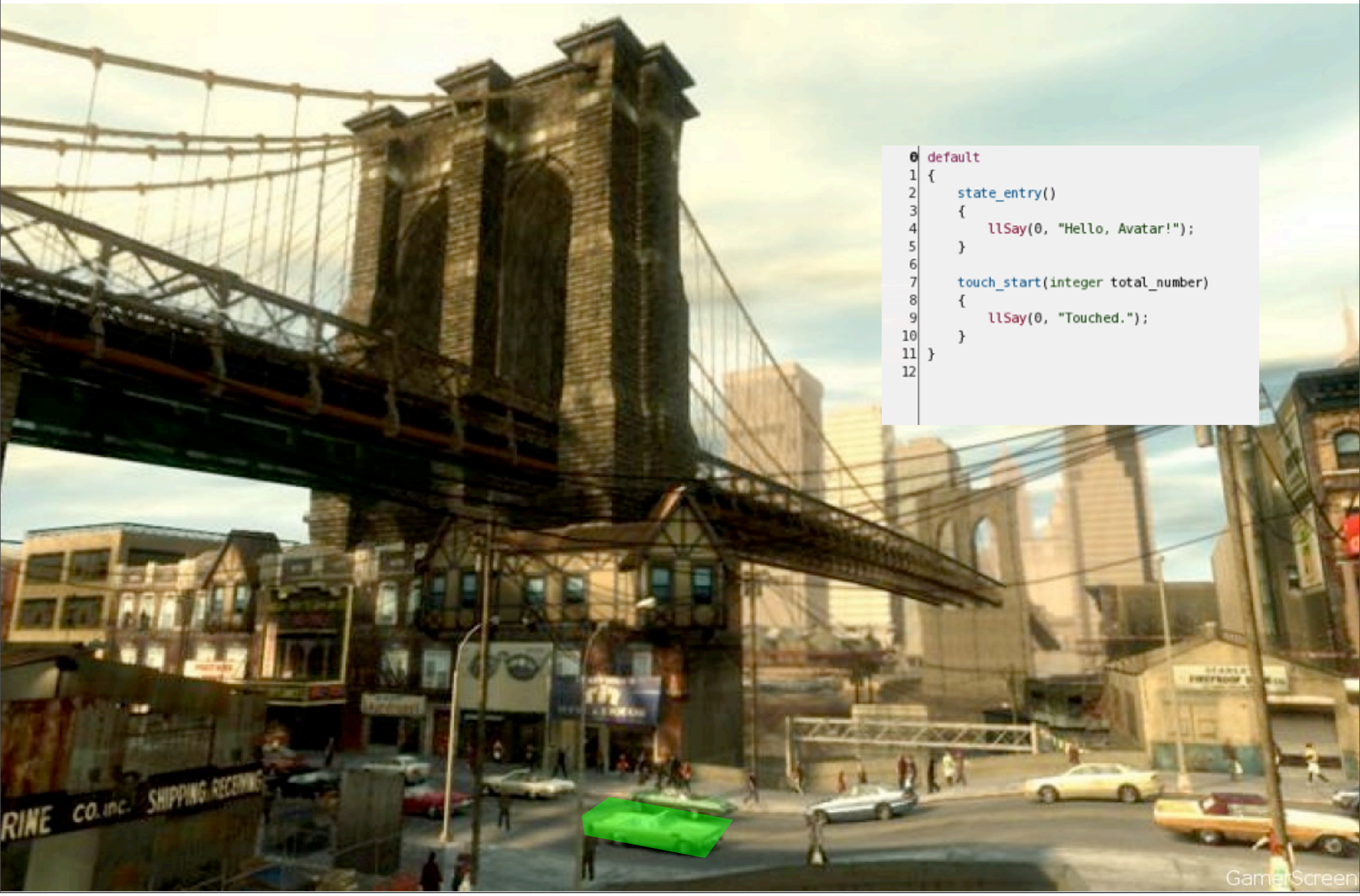
you can just select objects, like this car here [transition], change their appearance, and open scripting windows [transition] to control their behavior -- real-time, live, in the system, and other users participating will see it.

Metaverses



you can just select objects, like this car here [transition], change their appearance, and open scripting windows [transition] to control their behavior -- real-time, live, in the system, and other users participating will see it.

Metaverses



```
default
{
    state_entry()
    {
        llSay(0, "Hello, Avatar!");
    }

    touch_start(integer total_number)
    {
        llSay(0, "Touched.");
    }
}
```

you can just select objects, like this car here [transition], change their appearance, and open scripting windows [transition] to control their behavior -- real-time, live, in the system, and other users participating will see it.

Metaverses

Applications:

- Games
- Augmented reality
- Historical recreations
- Collaborative visualization
- … what will users create?

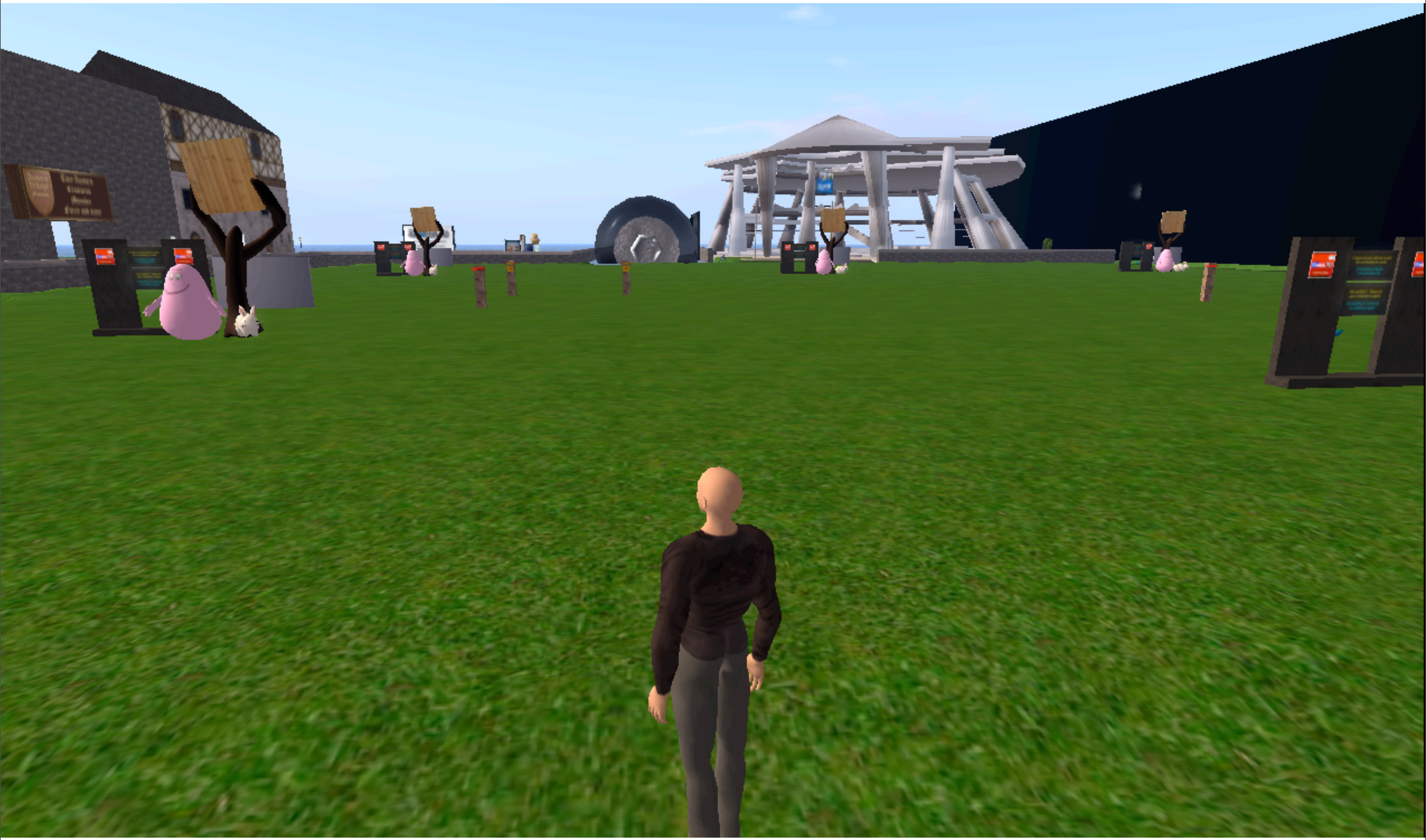
This very flexible environment leads to a wide variety of interesting applications -- games, augmented reality, historical recreations and collaborative visualization to name a few. But more importantly, they open the ability to create applications to **users**, leading to all sorts of unexpected and novel applications. These systems really lower the bar to creating interactive, multiuser 3D applications so you don’t need to be a professional 3D graphics developer to create them.

Metaverses promise such interesting, expansive, immersive worlds...

(Images by Chris Platz)

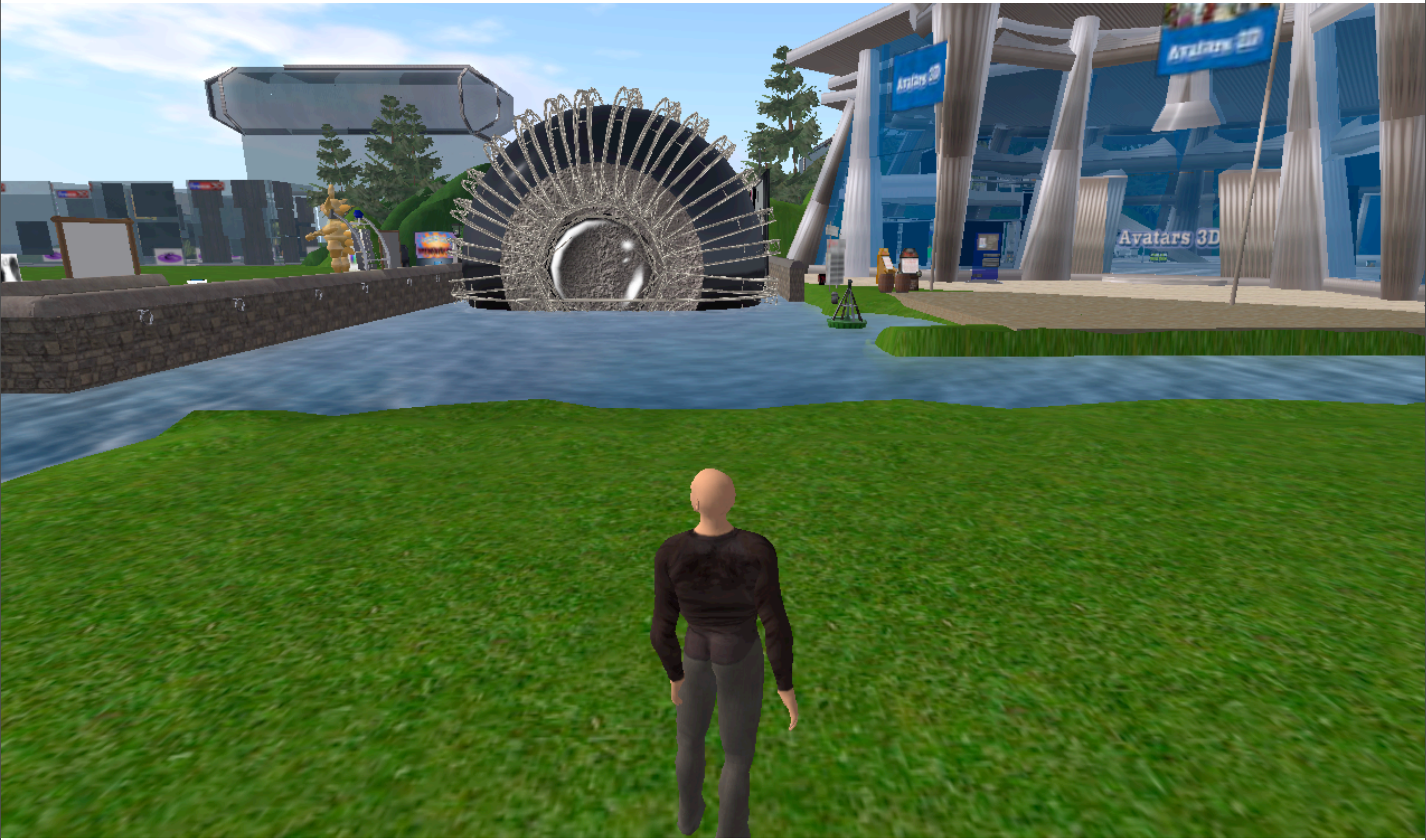
...filled with all sorts of novel, user-defined experiences.

(Images by Chris Platz)



Unfortunately, metaverses today don't even come close to what's been described in fiction.

One particular problem with today's worlds, which we'll look at in detail today, shows up when you first log into Second Life. When you join, stand still for a minute and let everything load, you get something that looks like this. But when you step forward a few meters then suddenly...



... a lot of content pops in that wasn't there before. One of the first things we want to do -- join the world and look around for something interesting -- is crippled or impossible in these systems. And similar problems appear throughout the system, beyond just what to display; for example, we're also limited in how we can communicate with objects, making long distance communication effectively useless.

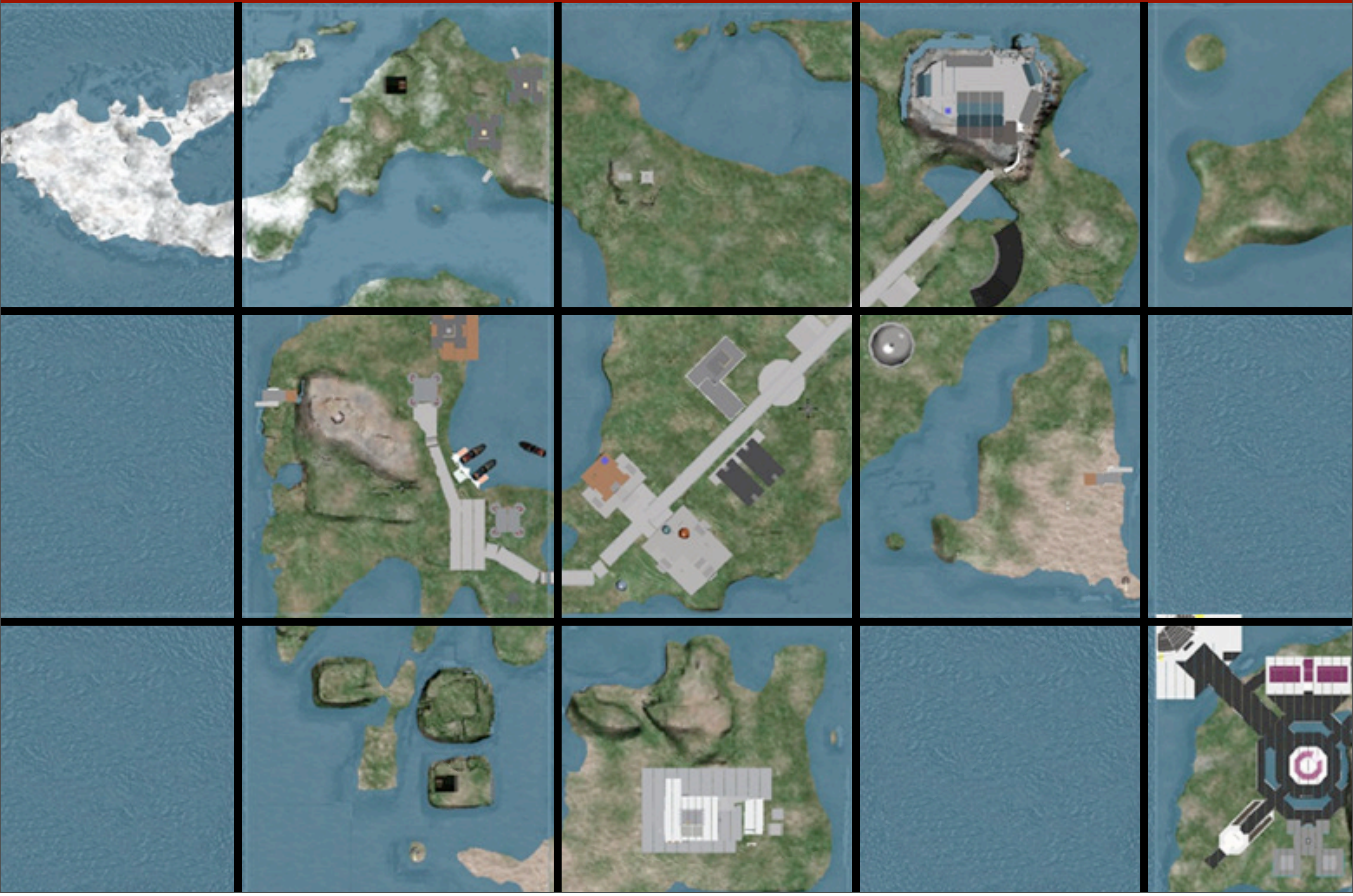
These are systems problems.

These are all fundamentally systems problems. The application may be graphical, but these limitations don't exist because we can't render more objects -- the previous scene clearly wouldn't stress my GPU. The problem is that we don't yet understand how to build these systems in a way which scales without sacrificing the user experience.

Object Discovery

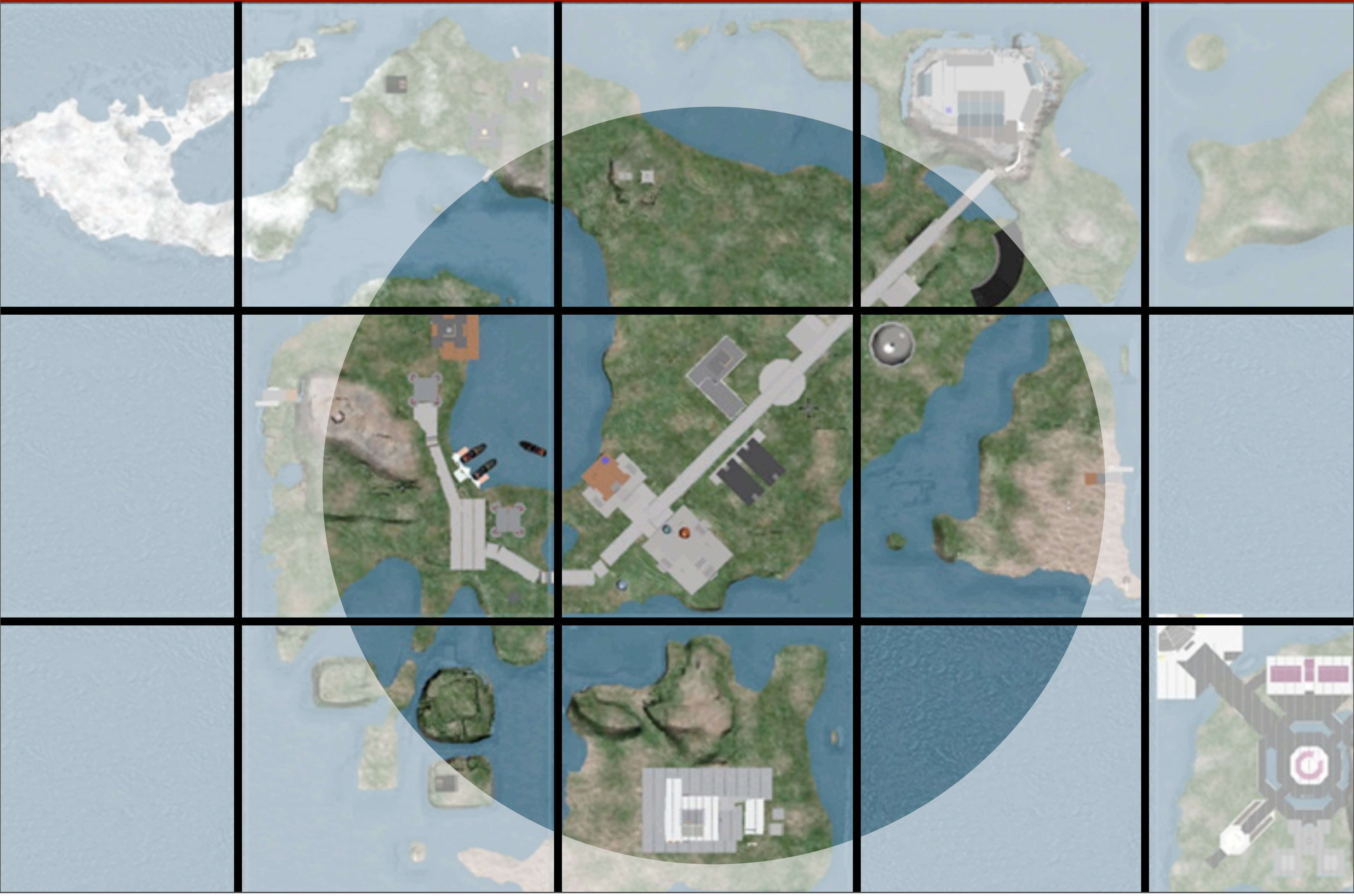
So why can't we see more objects in the world? The reason is that this is how we know how to scale these systems. If we only return nearby objects, then when we carve up the world [transition] among servers, then each server only needs state from neighboring servers [transition]. This does allow the system to scale, but leads to an uncompelling experience.

Object Discovery



So why can't we see more objects in the world? The reason is that this is how we know how to scale these systems. If we only return nearby objects, then when we carve up the world [transition] among servers, then each server only needs state from neighboring servers [transition]. This does allow the system to scale, but leads to an uncompelling experience.

Object Discovery

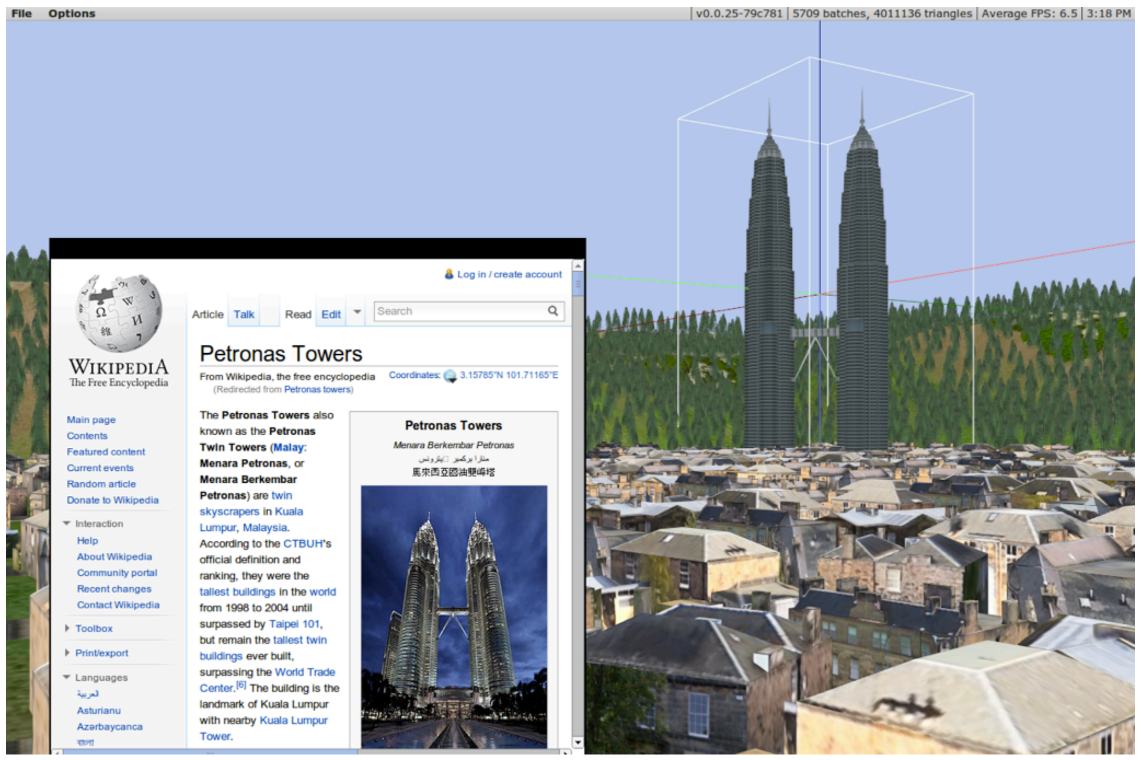


So why can’t we see more objects in the world? The reason is that this is how we know how to scale these systems. If we only return nearby objects, then when we carve up the world [transition] among servers, then each server only needs state from neighboring servers [transition]. This does allow the system to scale, but leads to an uncompelling experience.

How do we scale up the world without limiting the scope of interaction?

We face this more general challenge throughout the system: how can we scale the world up **without** limiting what we can see and what we can interact with? This paper tries to answer that question.

Sirikata



Petronas Towers

From Wikipedia, the free encyclopedia Coordinates: 3.15785°N 101.71165°E
(Redirected from Petronas towers)

The **Petronas Towers** also known as the **Petronas Twin Towers** (Malay: **Menara Petronas**, or **Menara Berkembar Petronas**) are twin skyscrapers in Kuala Lumpur, Malaysia. According to the CTBUH's official definition and ranking, they were the tallest buildings in the world from 1998 to 2004 until surpassed by Taipei 101, but remain the tallest twin buildings ever built, surpassing the World Trade Center.[6] The building is the landmark of Kuala Lumpur with nearby Kuala Lumpur Tower.

Petronas Towers
Menara Berkembar Petronas
馬來西亞國油雙峰塔

Seamless, scalable, and federated metaverses

Sirikata is our new platform for seamless, scalable, and federated metaverses that addresses this challenge. I've shown a few screenshots here from applications built in Sirikata by undergrads in the summer of 2011. The system is still in development, but they were able to quickly build these applications, none of which would have been feasible in other systems due to the constraints on interaction I just described.

Insight

The real world scales.

Our solutions to these problems are based on a simple insight: the real world scales.

Design Principle

Scale by applying real-world constraints to the system.

Therefore, throughout our design, when we encounter a scalability challenge, we can look to the real world for guidance. We apply real-world constraints to scale the system. We apply this principle throughout the system, but I'm going to focus on just one challenge today: object discovery.

Object Discovery

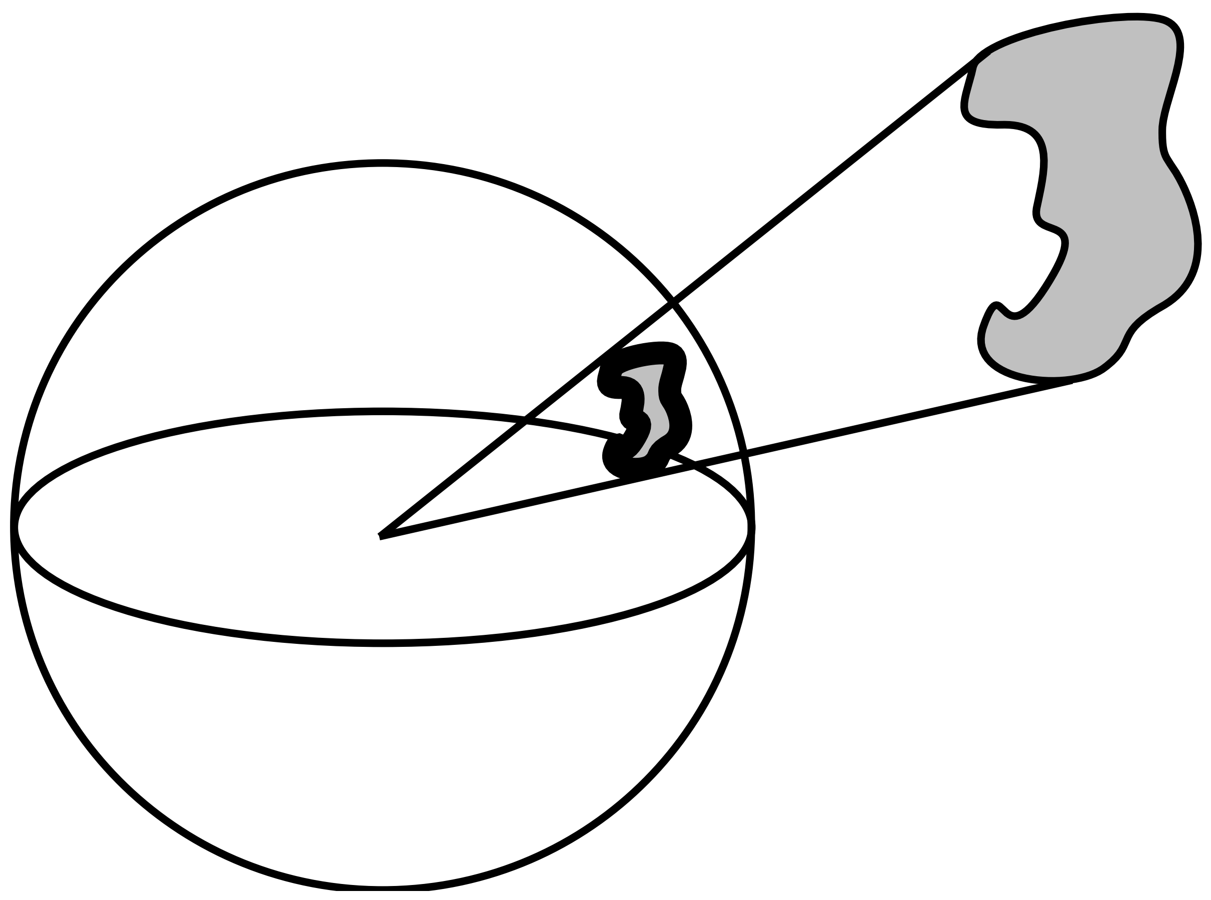
So how can we remove this restriction other systems had to put in place in order to scale? How do we enable display and interaction with objects throughout the world [transition]?

Object Discovery

So how can we remove this restriction other systems had to put in place in order to scale? How do we enable display and interaction with objects throughout the world [transition]?

Solid Angle Queries

Insight: Limited display resolution

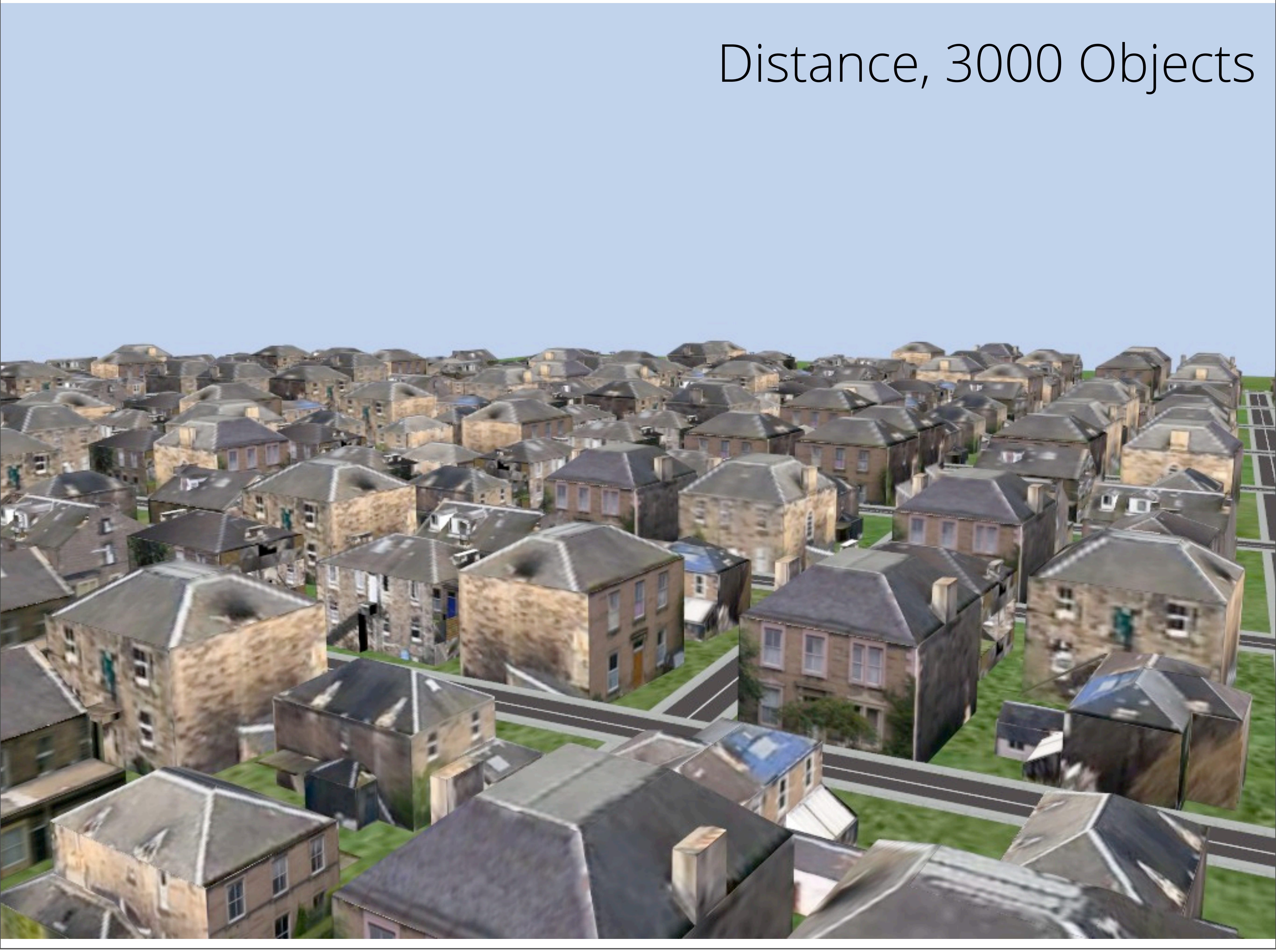


Solid angle: how large an object appears

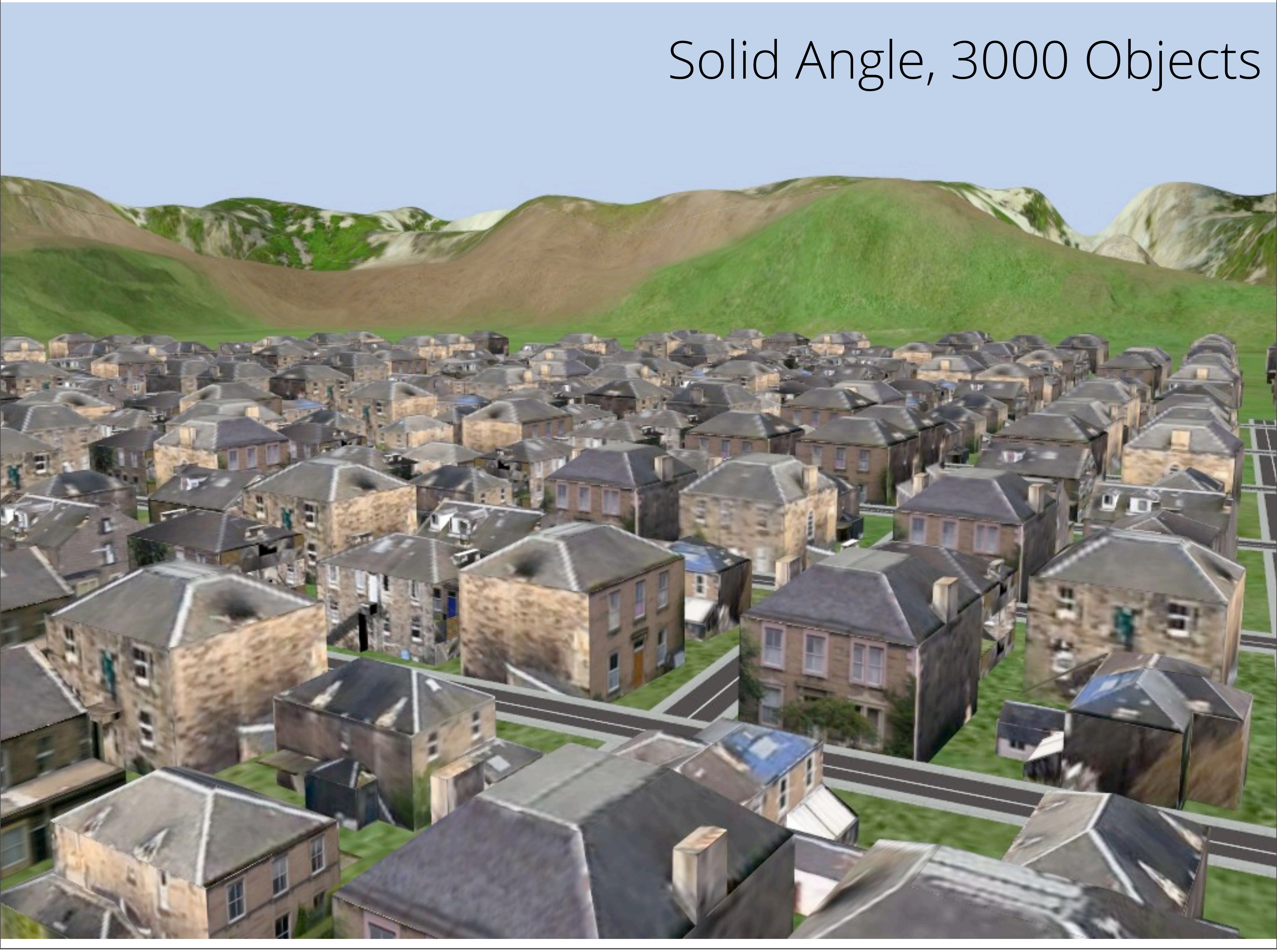
Sirikata exploits the fact that displays ultimately have a limited resolution: the output on your screen will have a limited bandwidth. Therefore, Sirikata uses a different type of query for object discovery called **solid angle queries**. Solid angle measures exactly what we want: how large an object appears to an observer, or, roughly speaking, the number of pixels the object takes up on screen. For example, this type of query will always return the sun no matter how far away the querier is, but small objects like a tree are only returned if they are very large or close by.

To demonstrate the effect of solid angle queries, here's the complete, ideal version of a scene we put together in Sirikata...

Distance, 3000 Objects



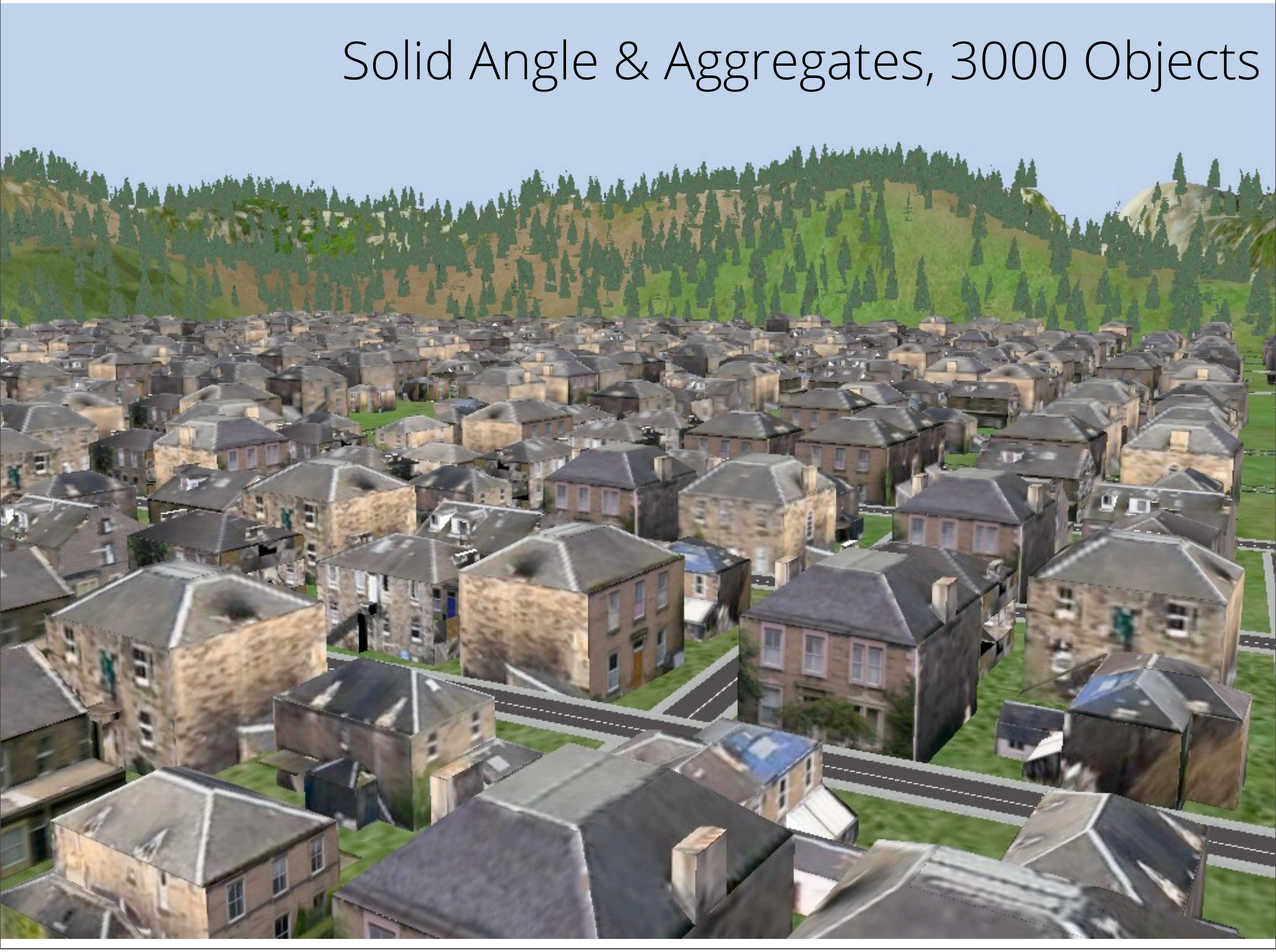
and here’s what distance query returns, making it appear as if the world just drops off...



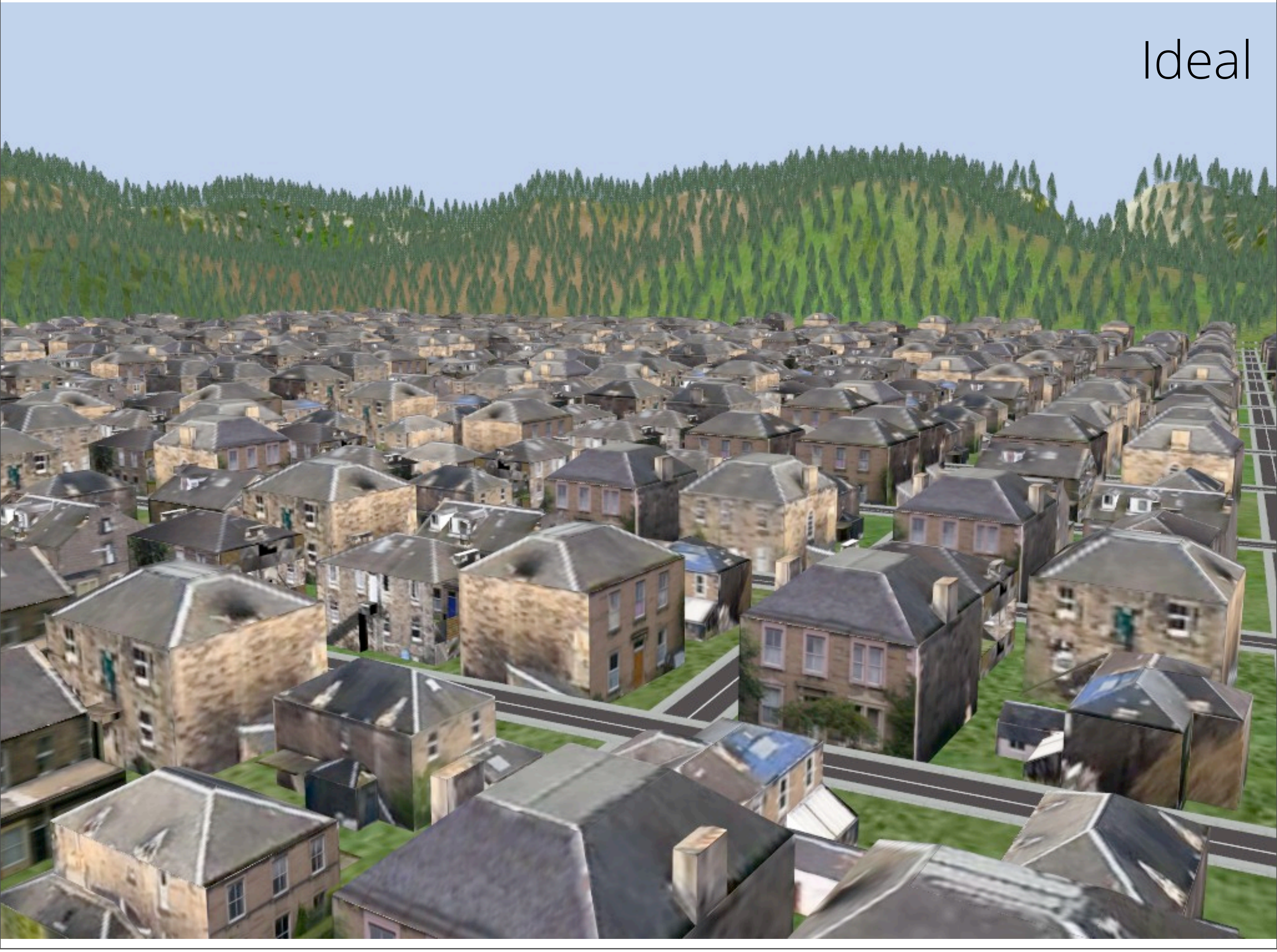
Solid Angle, 3000 Objects

And this is what it looks like with solid angle queries, using the same number of objects as the previous slide. It's a big improvement because the distant mountains are now visible. However, we're still missing all those trees. That's because individually they are too small, so the object discovery service doesn't return them. To make sure we don't miss any of these objects, Sirikata also supports returning aggregates, collections of objects which have been combined and simplified.

Solid Angle & Aggregates, 3000 Objects



This allows us to complete the scene. All the trees have been filled in, although at lower quality, by using aggregates instead of individual objects.



Flipping between the ideal and the version with aggregates that Sirikata uses, we can see some differences, but they are very close.

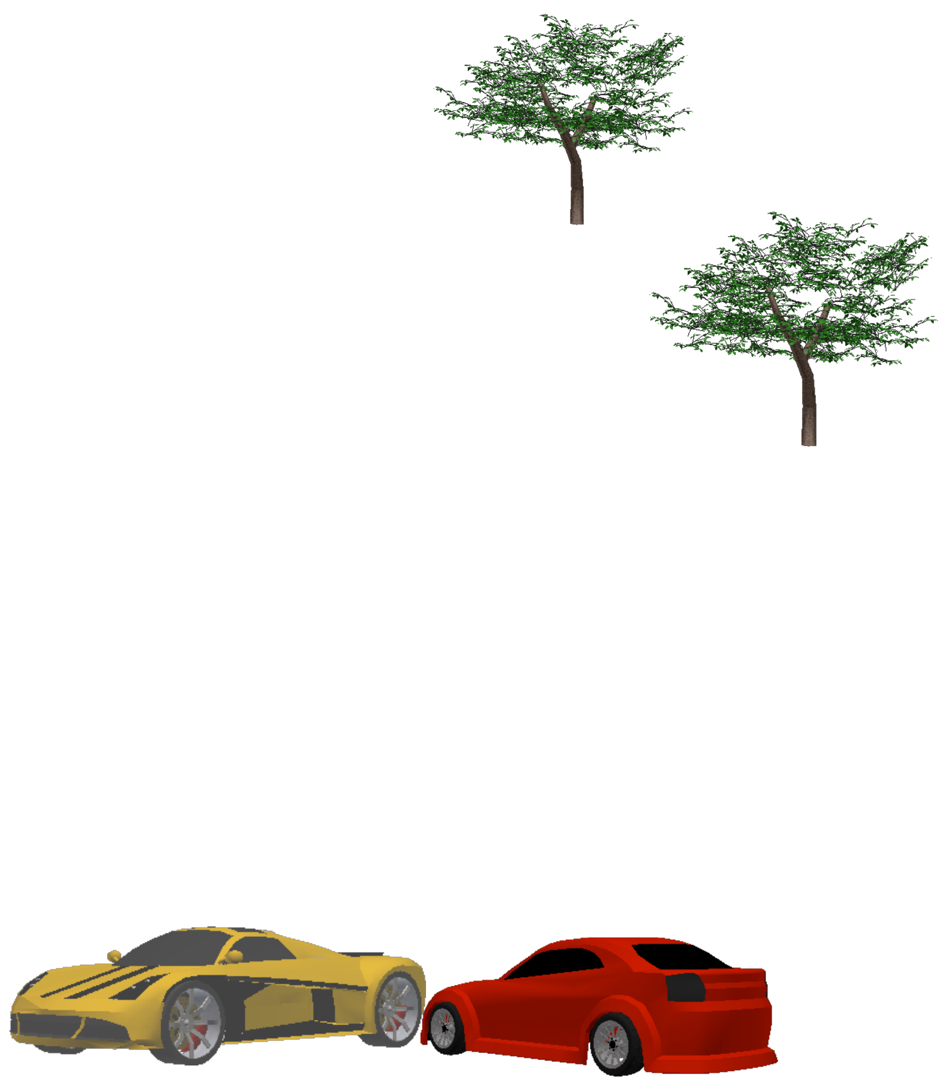
Object Discovery

Solid angle queries are global.

How do we efficiently and scalably evaluate solid angle queries?

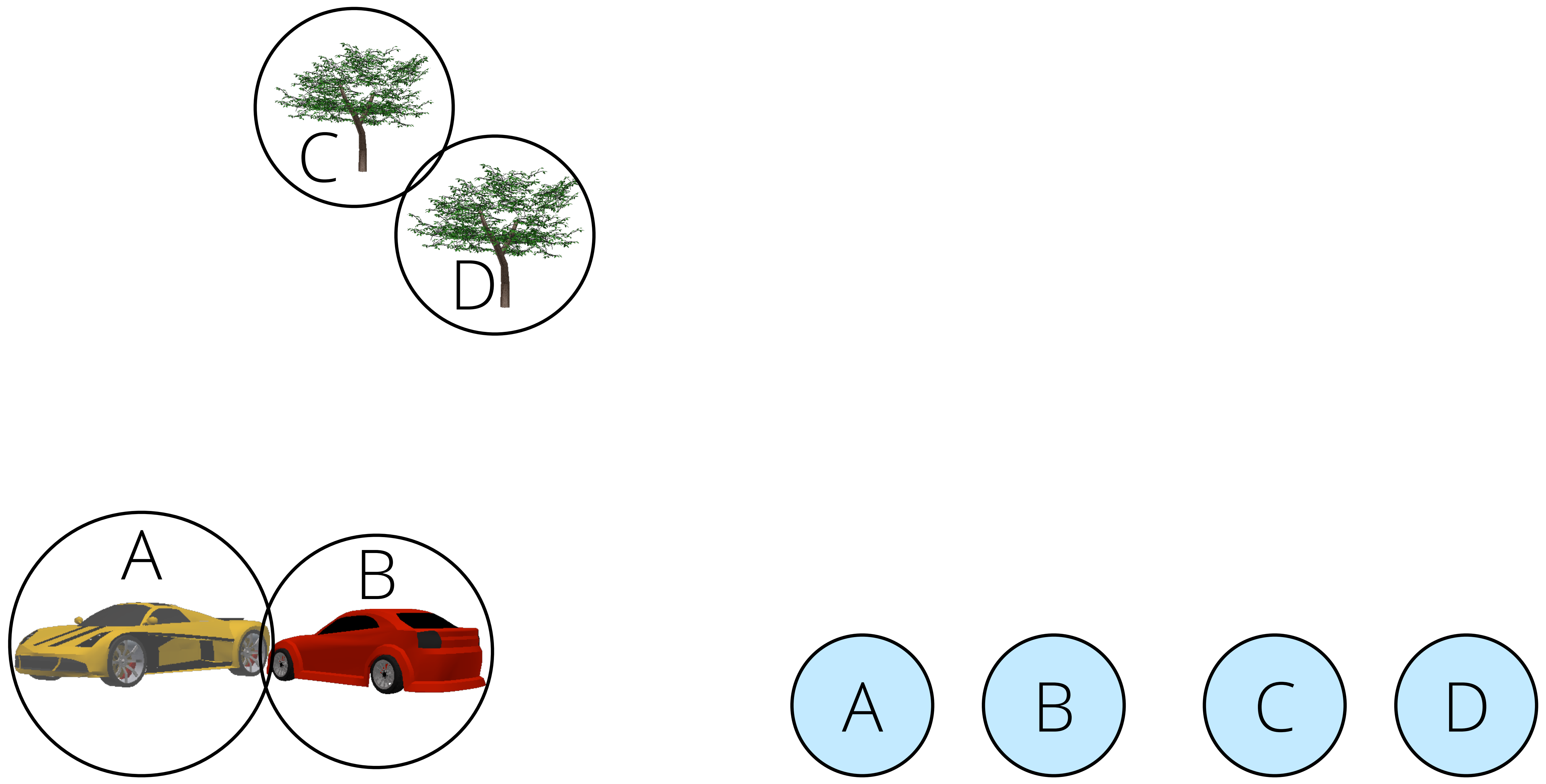
So with the same number of objects we can get much better results. But solid angle queries are global -- an object many servers away, if large enough, could be returned. But we also know there will be a limited output to the query, so our challenge is to build a query data structure and a distributed system upon it which can efficiently narrow the set down to only the necessary objects so the system can scale up.

Data Structure - BVH



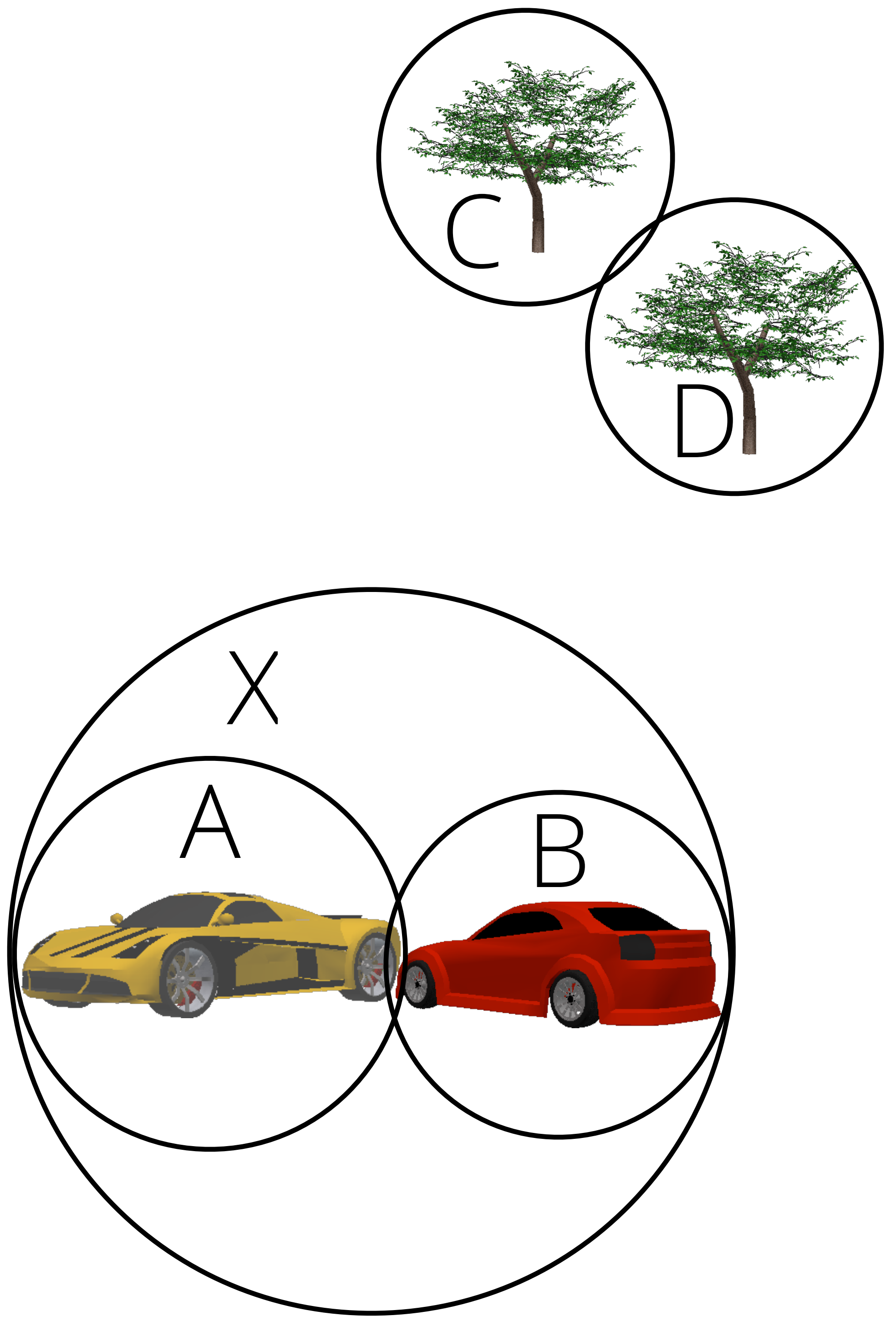
The core of our solution to this is a novel modification of an existing graphics data structure called the bounding volume hierarchy, or BVH. It's called a bounding volume hierarchy because we start with a set of objects, shown geometrically on the left, and create leaf nodes [transition], shown on the right, where each node tracks the 3D bounding sphere that surrounds the object. We then build a hierarchy atop them, where each internal node bounds the objects below it: we add X [transition], whose bounding sphere surrounds A and B, Y [transition] to surround C and D, and then Z [transition] to surround both X and Y. Z is the root and bounds the entire scene.

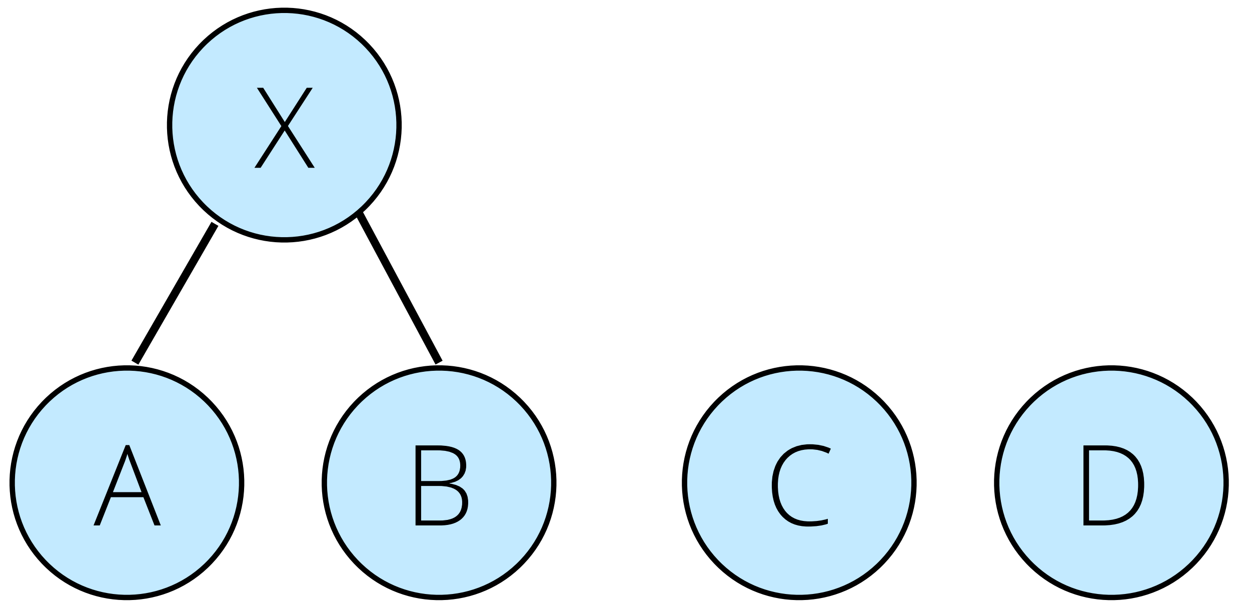
Data Structure - BVH



The core of our solution to this is a novel modification of an existing graphics data structure called the bounding volume hierarchy, or BVH. It's called a bounding volume hierarchy because we start with a set of objects, shown geometrically on the left, and create leaf nodes [transition], shown on the right, where each node tracks the 3D bounding sphere that surrounds the object. We then build a hierarchy atop them, where each internal node bounds the objects below it: we add X [transition], whose bounding sphere surrounds A and B, Y [transition] to surround C and D, and then Z [transition] to surround both X and Y. Z is the root and bounds the entire scene.

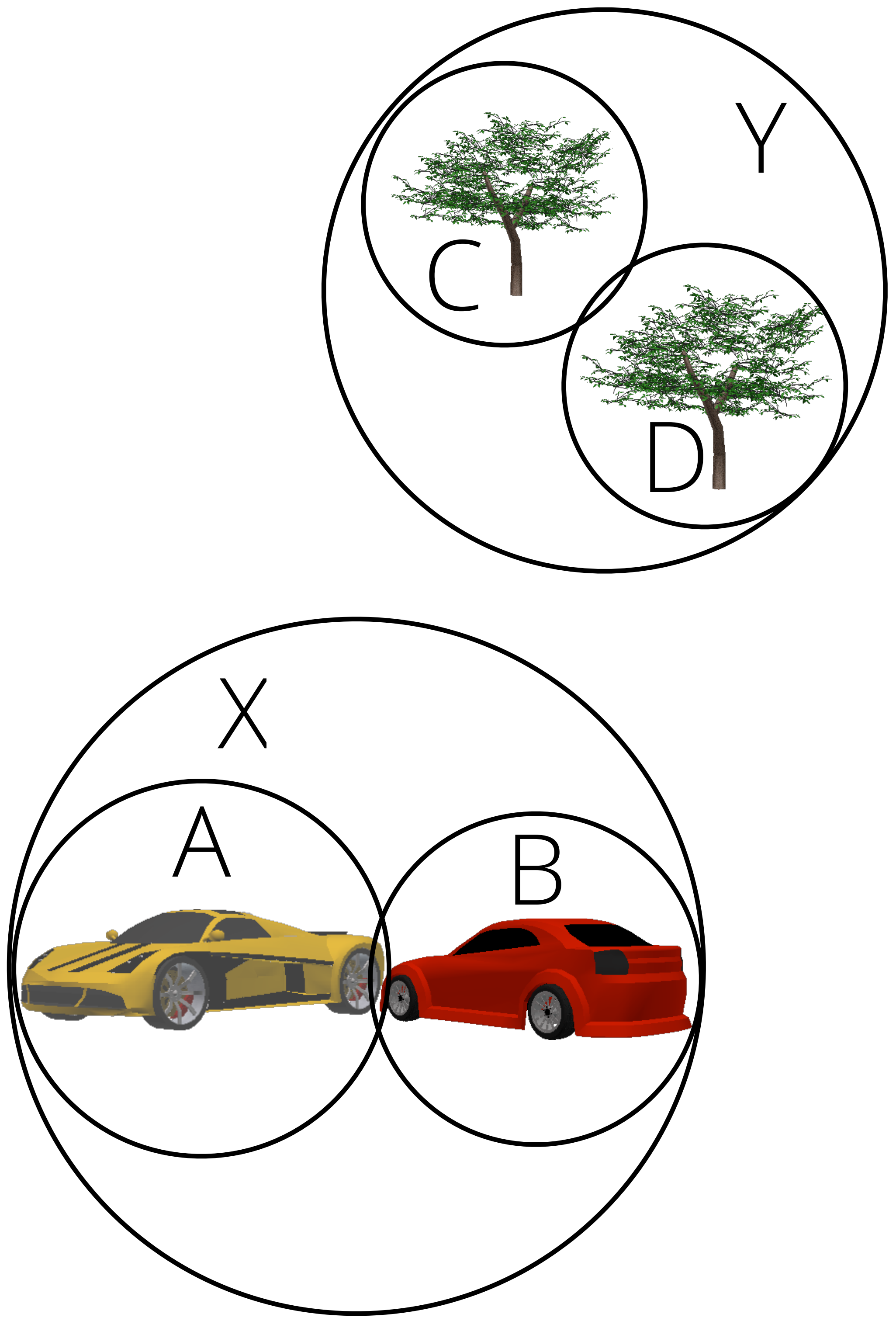
Data Structure - BVH

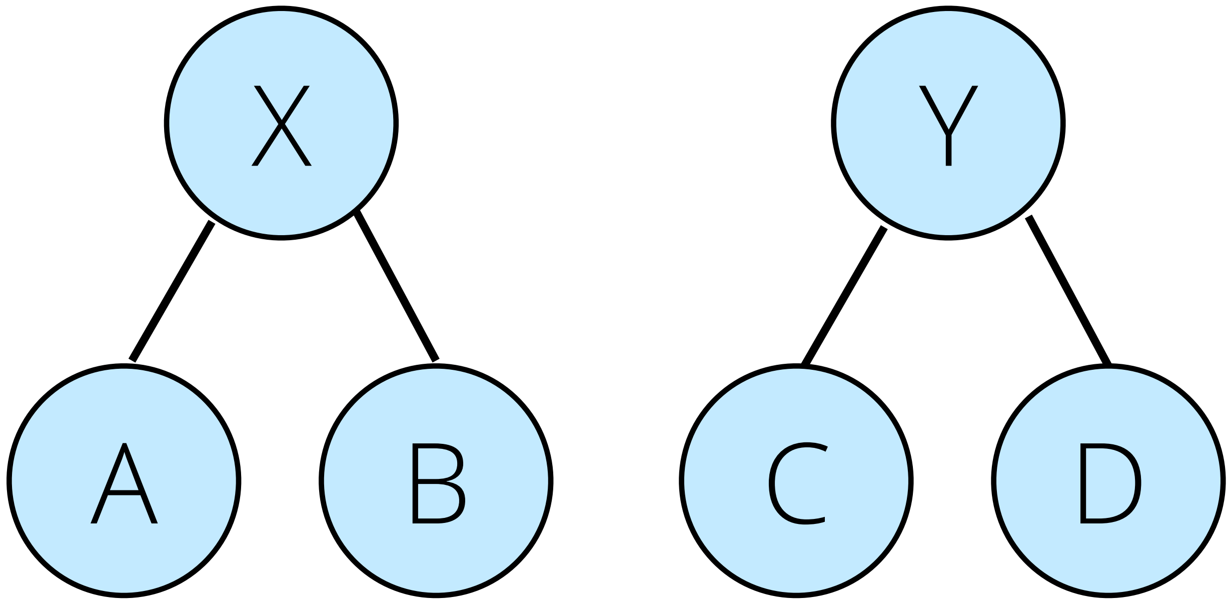




The core of our solution to this is a novel modification of an existing graphics data structure called the bounding volume hierarchy, or BVH. It’s called a bounding volume hierarchy because we start with a set of objects, shown geometrically on the left, and create leaf nodes [transition], shown on the right, where each node tracks the 3D bounding sphere that surrounds the object. We then build a hierarchy atop them, where each internal node bounds the objects below it: we add X [transition], whose bounding sphere surrounds A and B, Y [transition] to surround C and D, and then Z [transition] to surround both X and Y. Z is the root and bounds the entire scene.

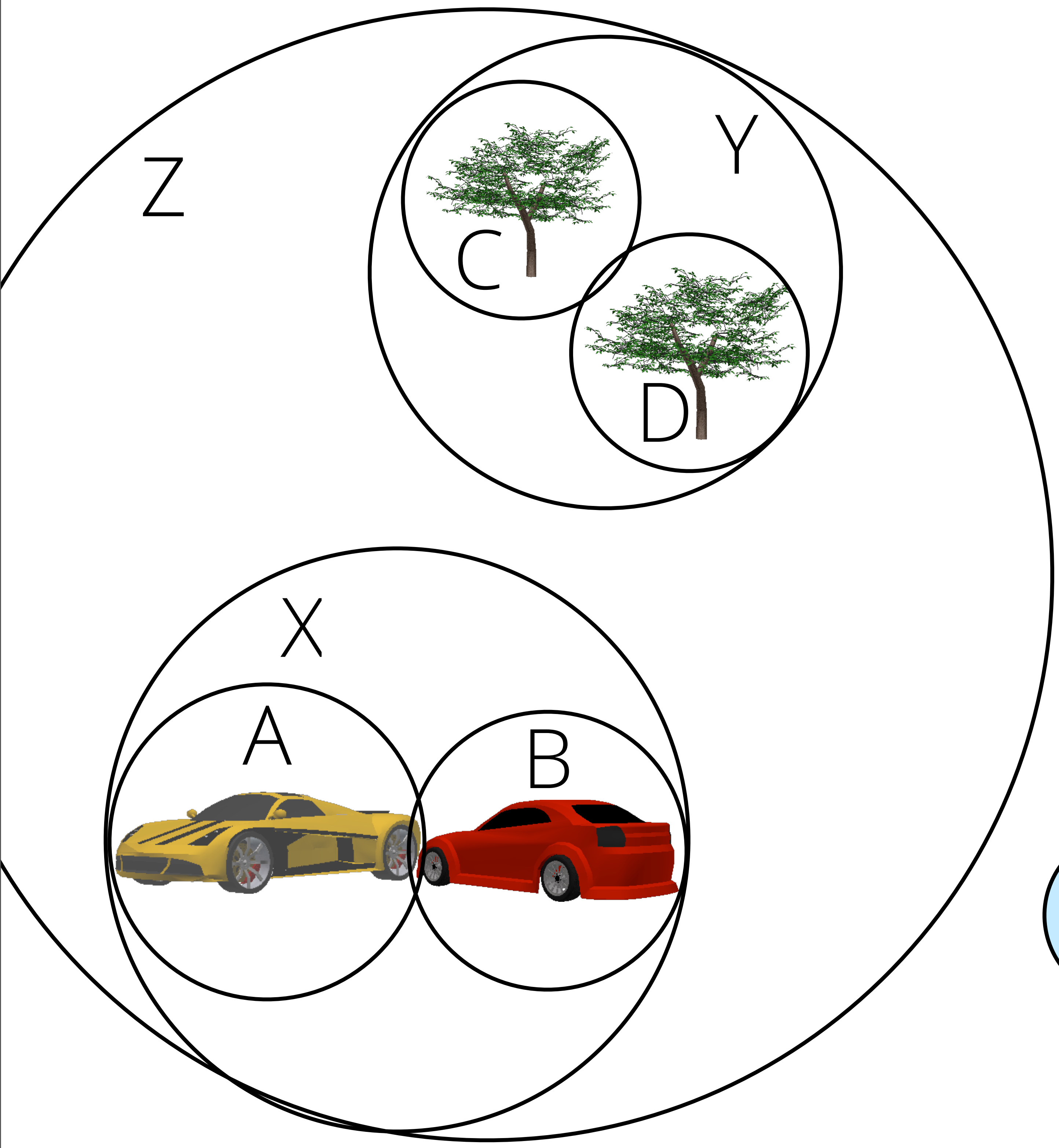
Data Structure - BVH

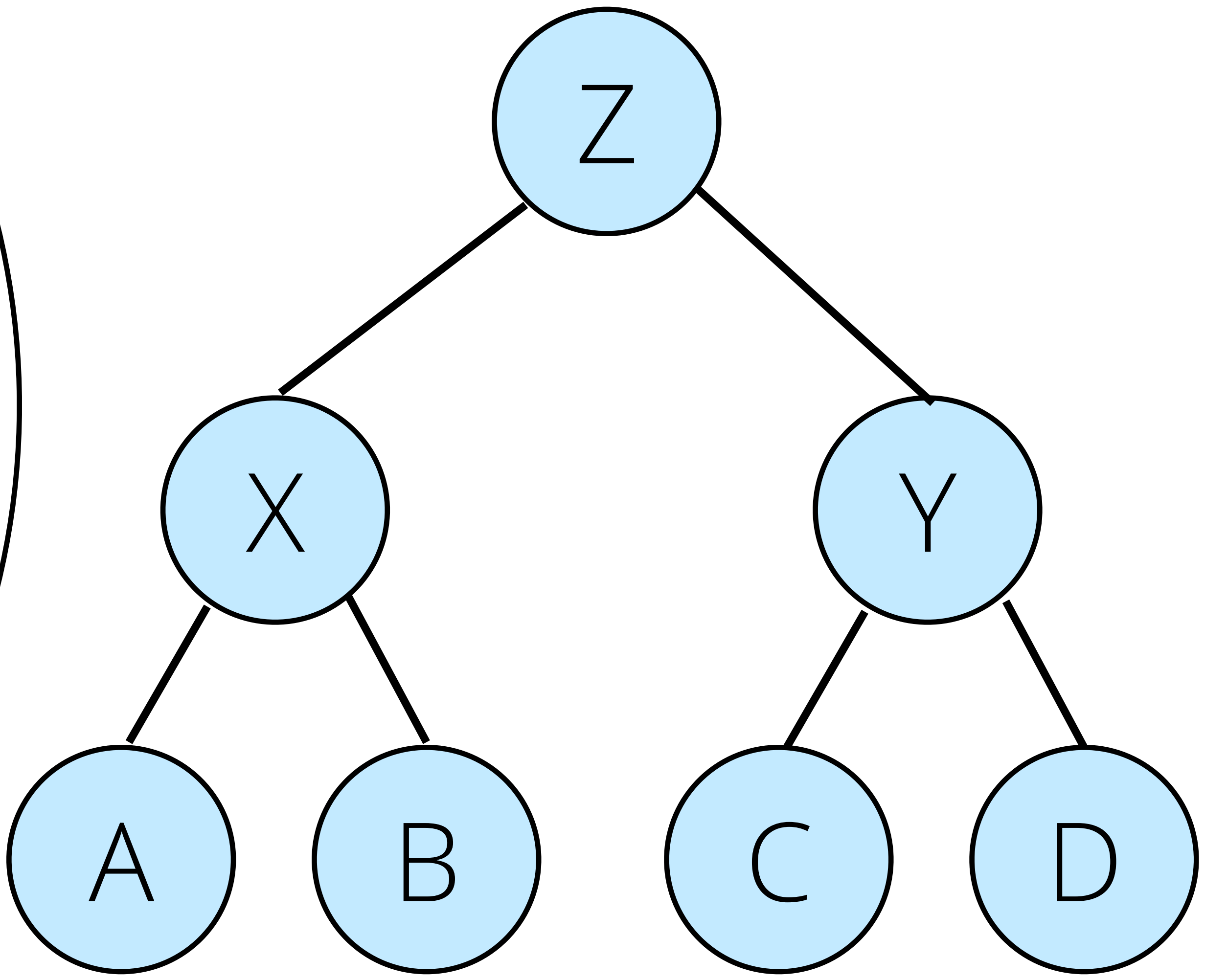




The core of our solution to this is a novel modification of an existing graphics data structure called the bounding volume hierarchy, or BVH. It's called a bounding volume hierarchy because we start with a set of objects, shown geometrically on the left, and create leaf nodes [transition], shown on the right, where each node tracks the 3D bounding sphere that surrounds the object. We then build a hierarchy atop them, where each internal node bounds the objects below it: we add X [transition], whose bounding sphere surrounds A and B, Y [transition] to surround C and D, and then Z [transition] to surround both X and Y. Z is the root and bounds the entire scene.

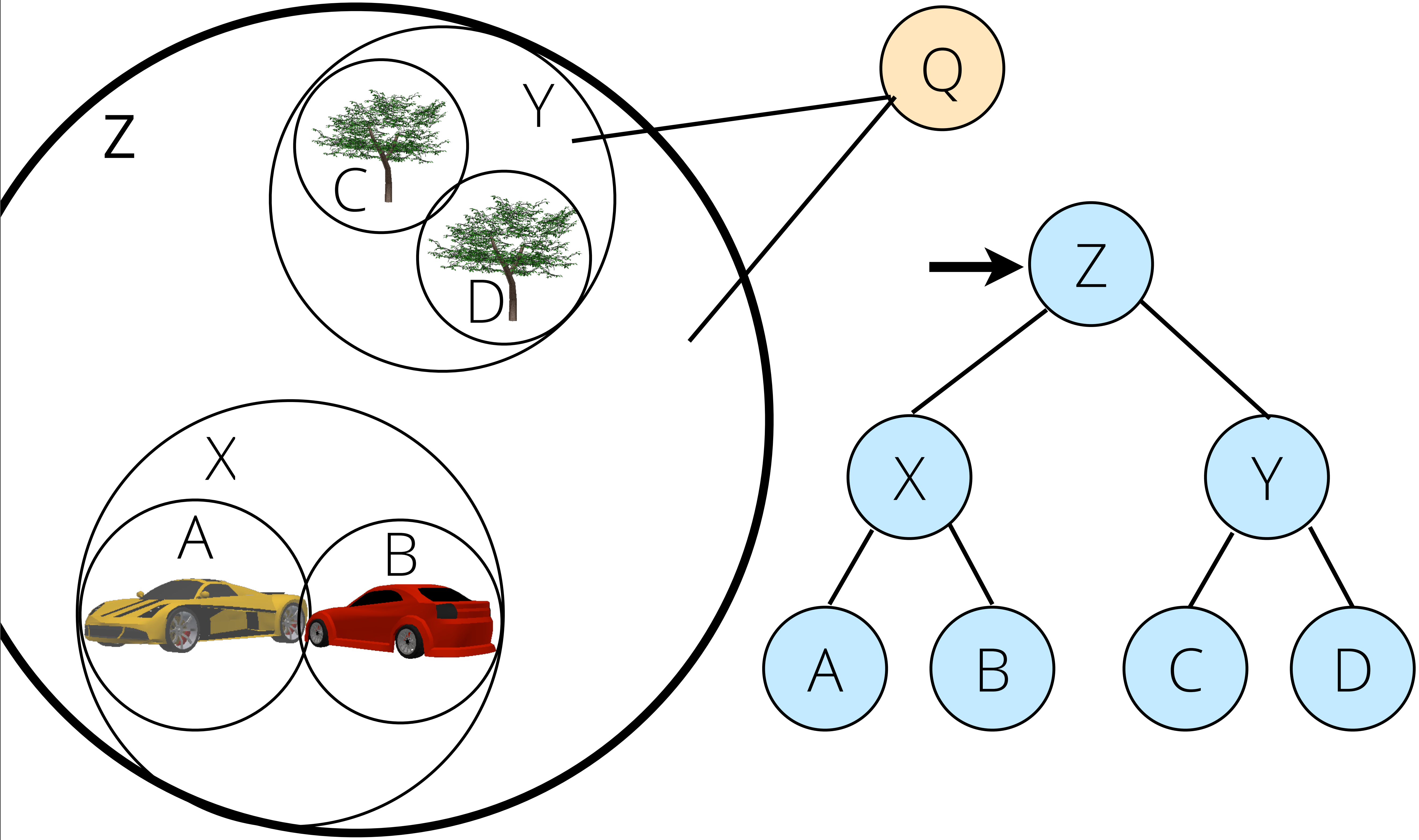
Data Structure - BVH





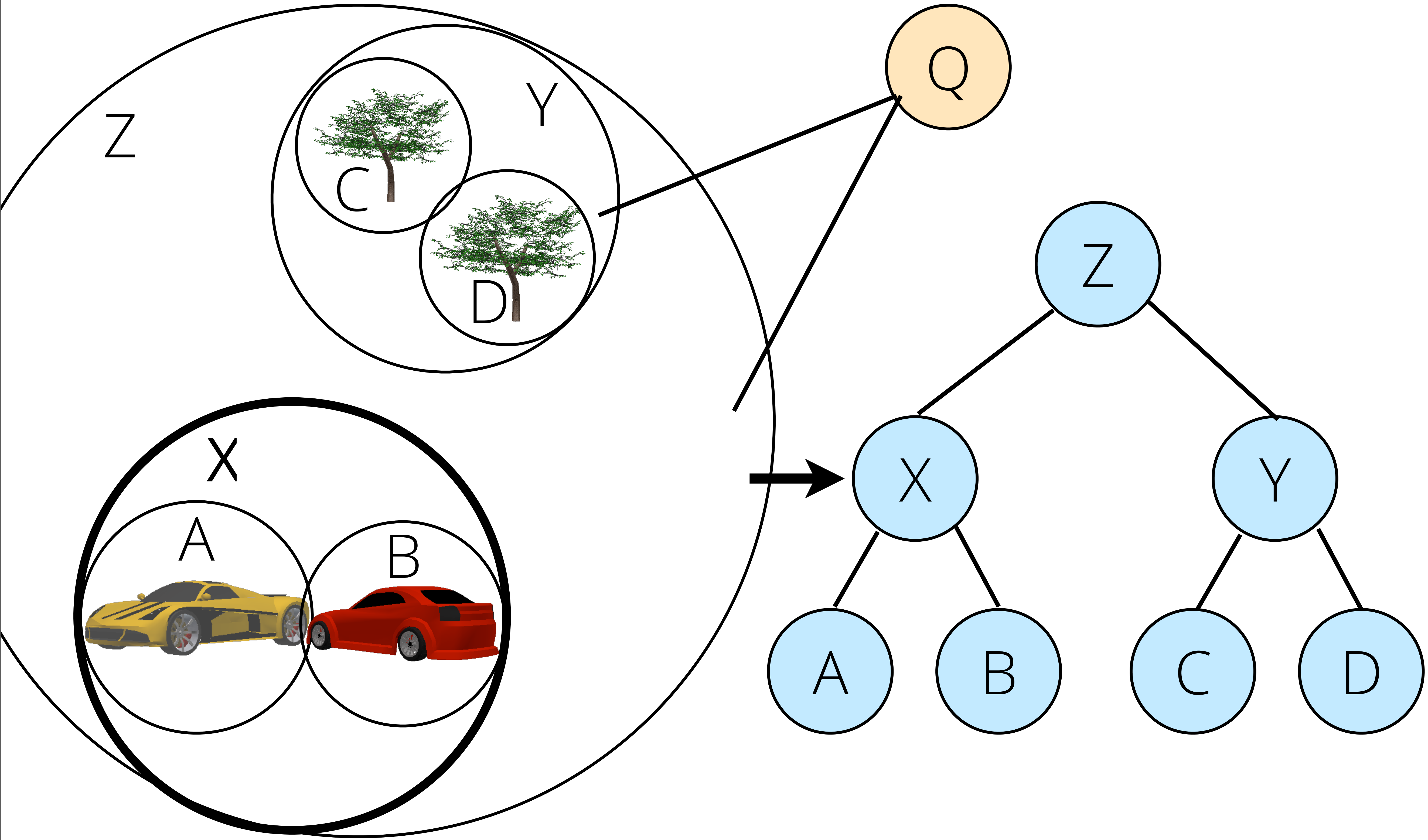
The core of our solution to this is a novel modification of an existing graphics data structure called the bounding volume hierarchy, or BVH. It’s called a bounding volume hierarchy because we start with a set of objects, shown geometrically on the left, and create leaf nodes [transition], shown on the right, where each node tracks the 3D bounding sphere that surrounds the object. We then build a hierarchy atop them, where each internal node bounds the objects below it: we add X [transition], whose bounding sphere surrounds A and B, Y [transition] to surround C and D, and then Z [transition] to surround both X and Y. Z is the root and bounds the entire scene.

Data Structure - BVH



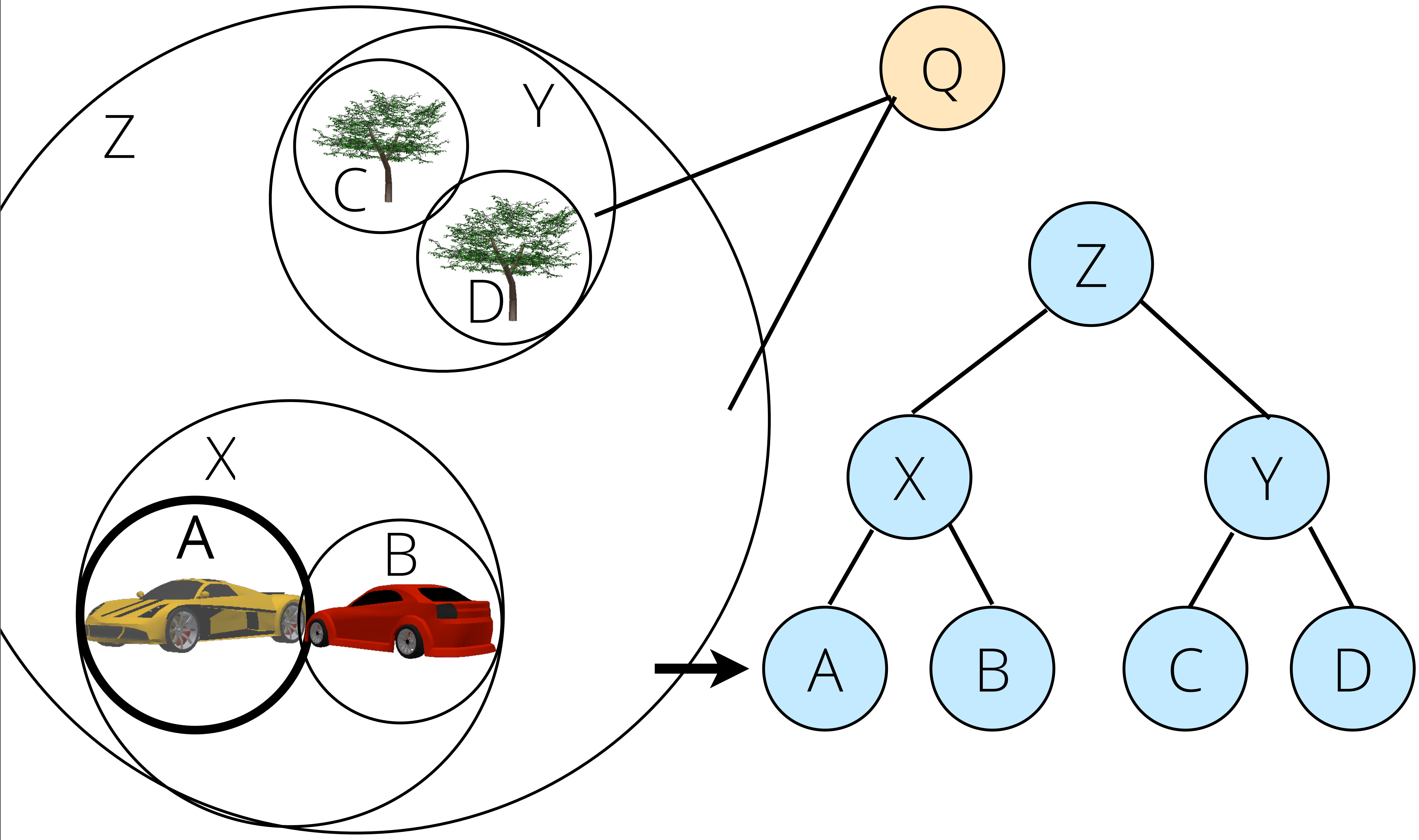
But now if we add a querier looking for objects that appear larger than the angle shown, we start at the root and recurse as nodes satisfy the query. Unfortunately, with these large objects, we'll quickly cover the whole tree, hitting Z easily…

Data Structure - BVH



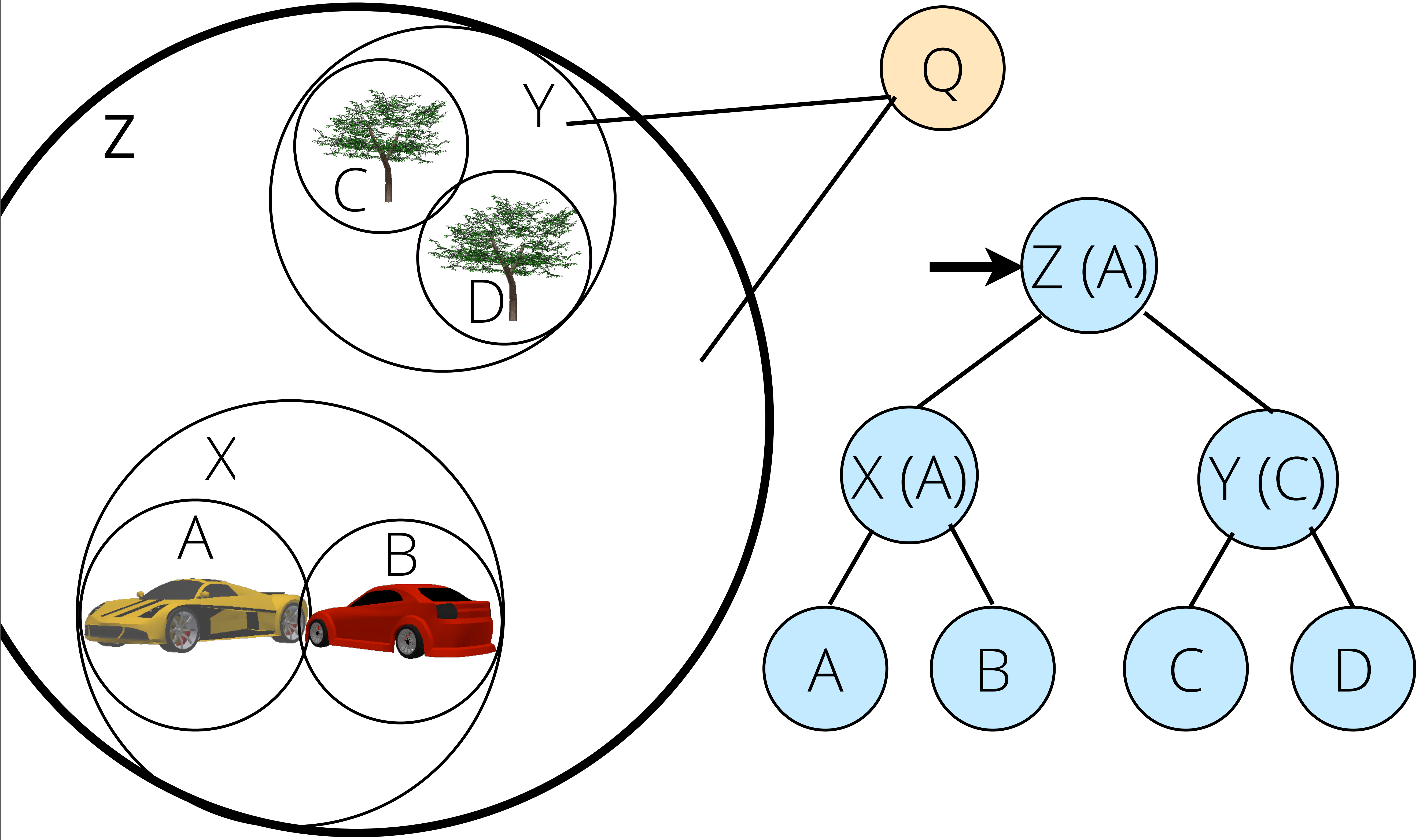
then aiming towards X and also recursing for it...

Data Structure - BVH



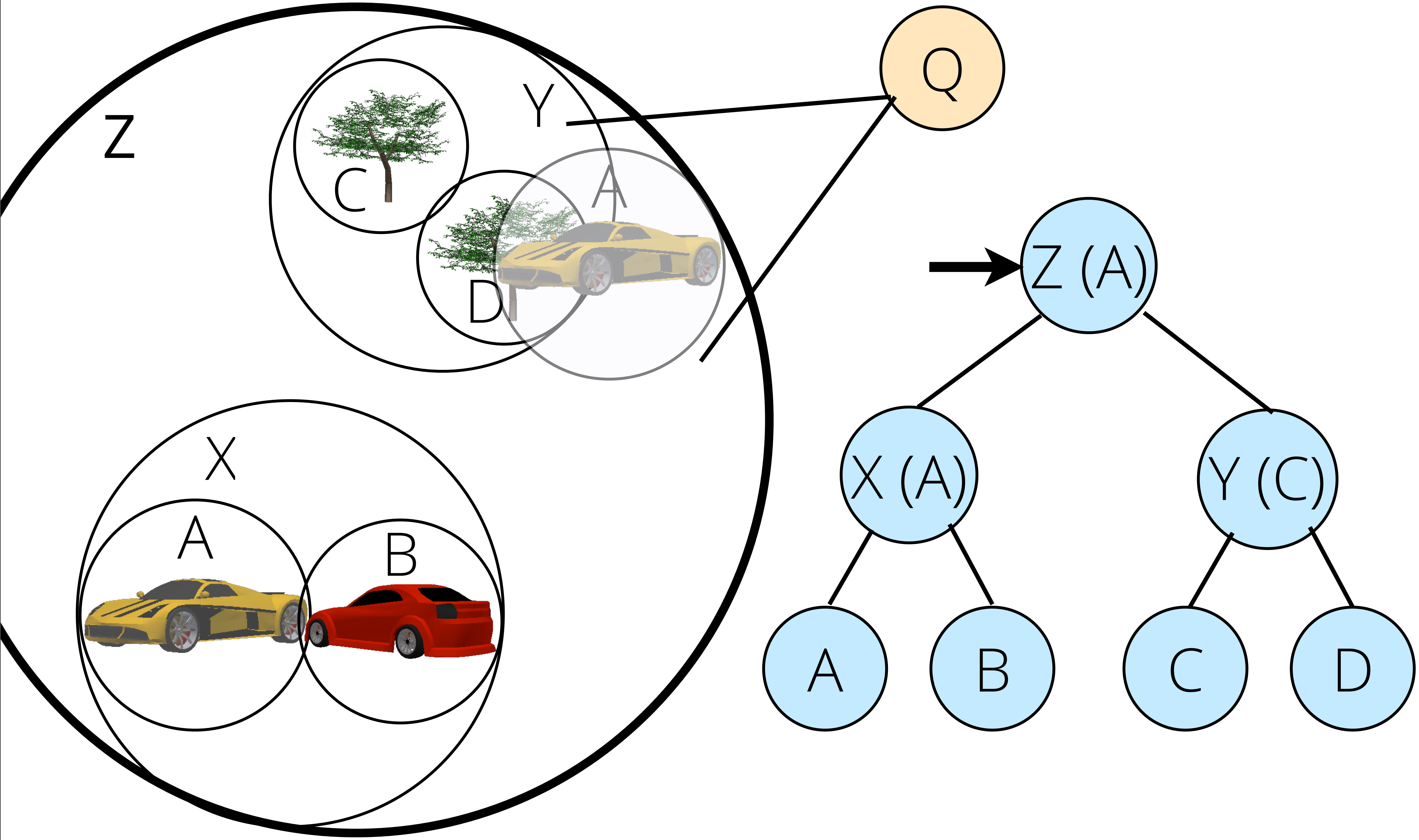
and testing A and B, even if they don't satisfy the query. In fact, I only showed one branch, but in this case we'll actually check the entire tree. This happens because the bounds are very large compared the objects below them -- not a problem for distance queries, but a big problem for solid angle queries, which are heavily influenced by object size.

New Data Structure - LBVH



To make query processing efficient, we augment each node with the largest object in the subtree, creating the largest object bounding volume hierarchy, or LBVH. This is a minor modification, but allows for a much more efficient test. When we test query Q against a node, for example Z, instead of testing the entire bounds, we move the largest object A as close to the querier as possible within the bounds Z [transition] and test that object. This is a much smaller object and therefore less likely to satisfy the query.

New Data Structure - LBVH



To make query processing efficient, we augment each node with the largest object in the subtree, creating the largest object bounding volume hierarchy, or LBVH. This is a minor modification, but allows for a much more efficient test. When we test query Q against a node, for example Z, instead of testing the entire bounds, we move the largest object A as close to the querier as possible within the bounds Z [transition] and test that object. This is a much smaller object and therefore less likely to satisfy the query.

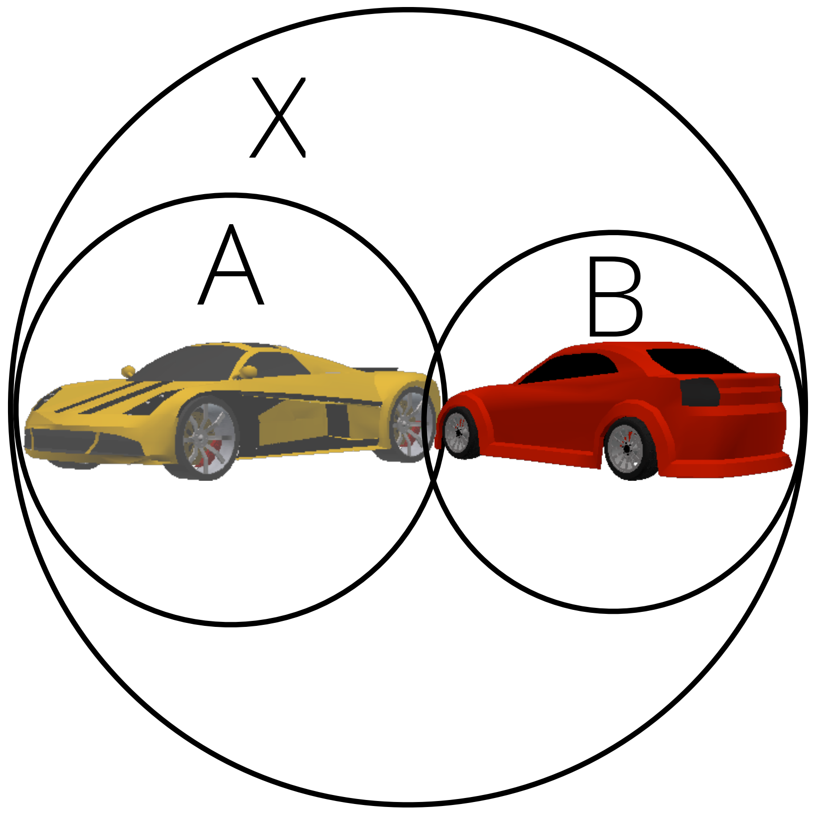
LBVH

75 - 90% fewer nodes tested
than with BVH

Overall, the LBVH reduces the number of nodes tested, and therefore the cost, of evaluating a query by 75-90% over the corresponding BVH. This modification is really what makes it possible to reasonably evaluate solid angle queries.

Dynamic Objects

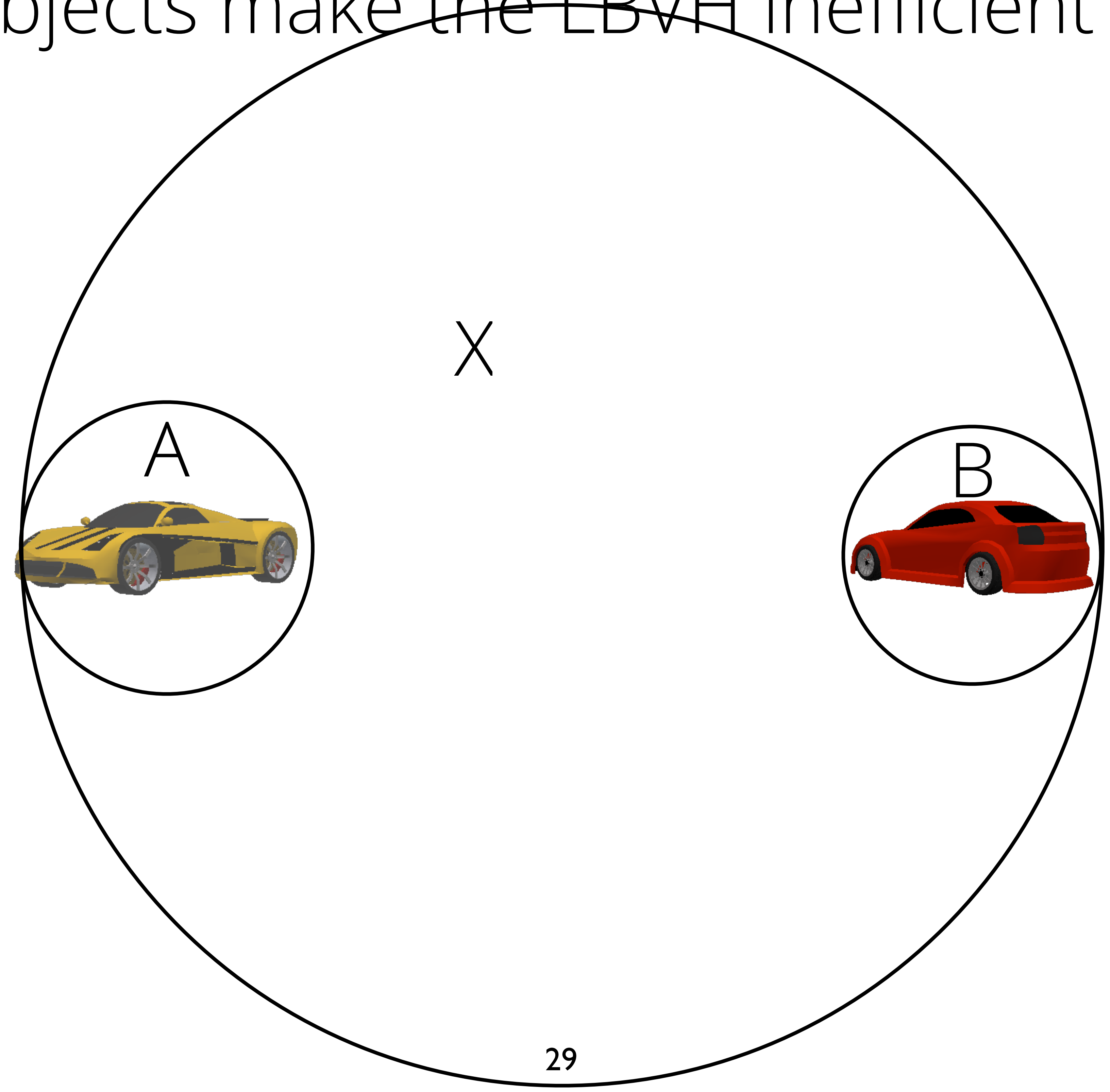
Moving objects make the LBVH inefficient over time



However, the LBVH alone isn’t sufficient. One major problem with the LBVH is that moving objects make it inefficient over time: as objects move apart, maintaining the same tree structure but updating the bounds results in the bounding spheres getting stretched out [transition].

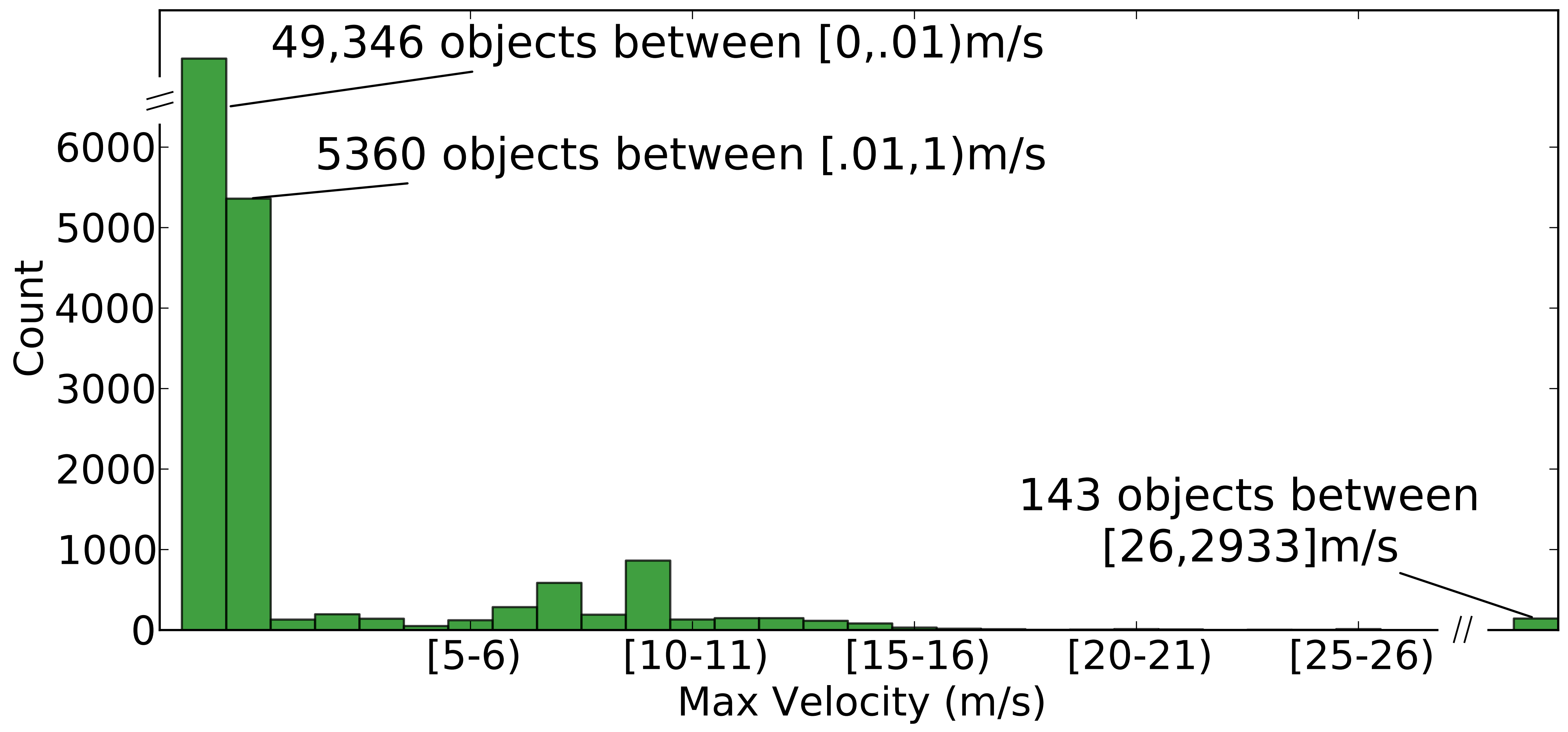
Dynamic Objects

Moving objects make the LBVH inefficient over time



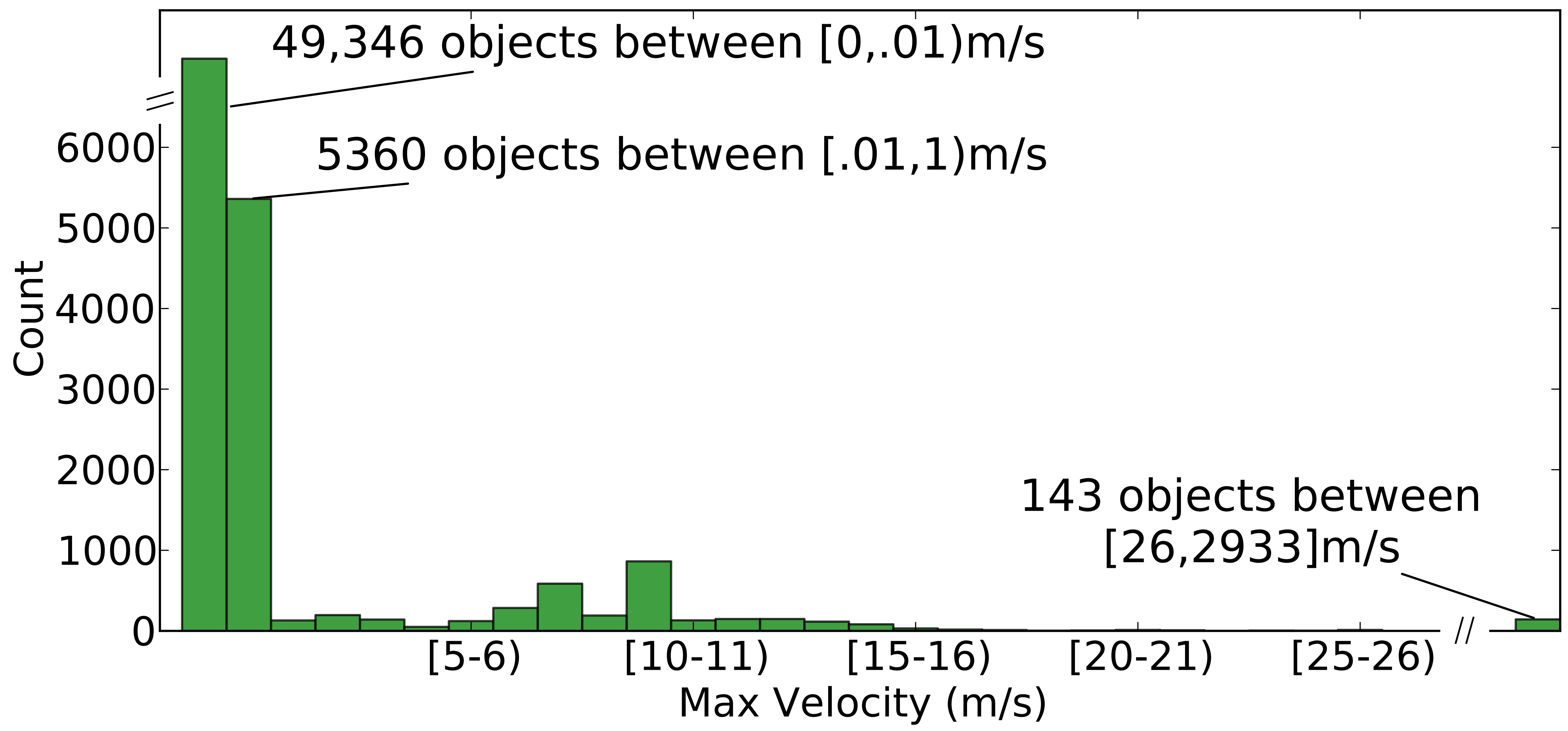
However, the LBVH alone isn’t sufficient. One major problem with the LBVH is that moving objects make it inefficient over time: as objects move apart, maintaining the same tree structure but updating the bounds results in the bounding spheres getting stretched out [transition].

Dynamic Objects



We could try to use a complicated approach for updating and reorganizing the tree to keep it efficient, and in fact we investigated that for awhile. But it turns out natural object movement distributions have a nice property: they're heavily static. This histogram, collected from objects in Second Life, shows that this is true -- note the broken Y-axis. We can exploit this very simply, just separating static objects into their own tree. The dynamic tree is much smaller and can safely become a bit inefficient, while the static tree changes very slowly and covers the vast majority of objects.

Dynamic Objects



Split between static and dynamic objects

We could try to use a complicated approach for updating and reorganizing the tree to keep it efficient, and in fact we investigated that for awhile. But it turns out natural object movement distributions have a nice property: they're heavily static. This histogram, collected from objects in Second Life, shows that this is true -- note the broken Y-axis. We can exploit this very simply, just separating static objects into their own tree. The dynamic tree is much smaller and can safely become a bit inefficient, while the static tree changes very slowly and covers the vast majority of objects.

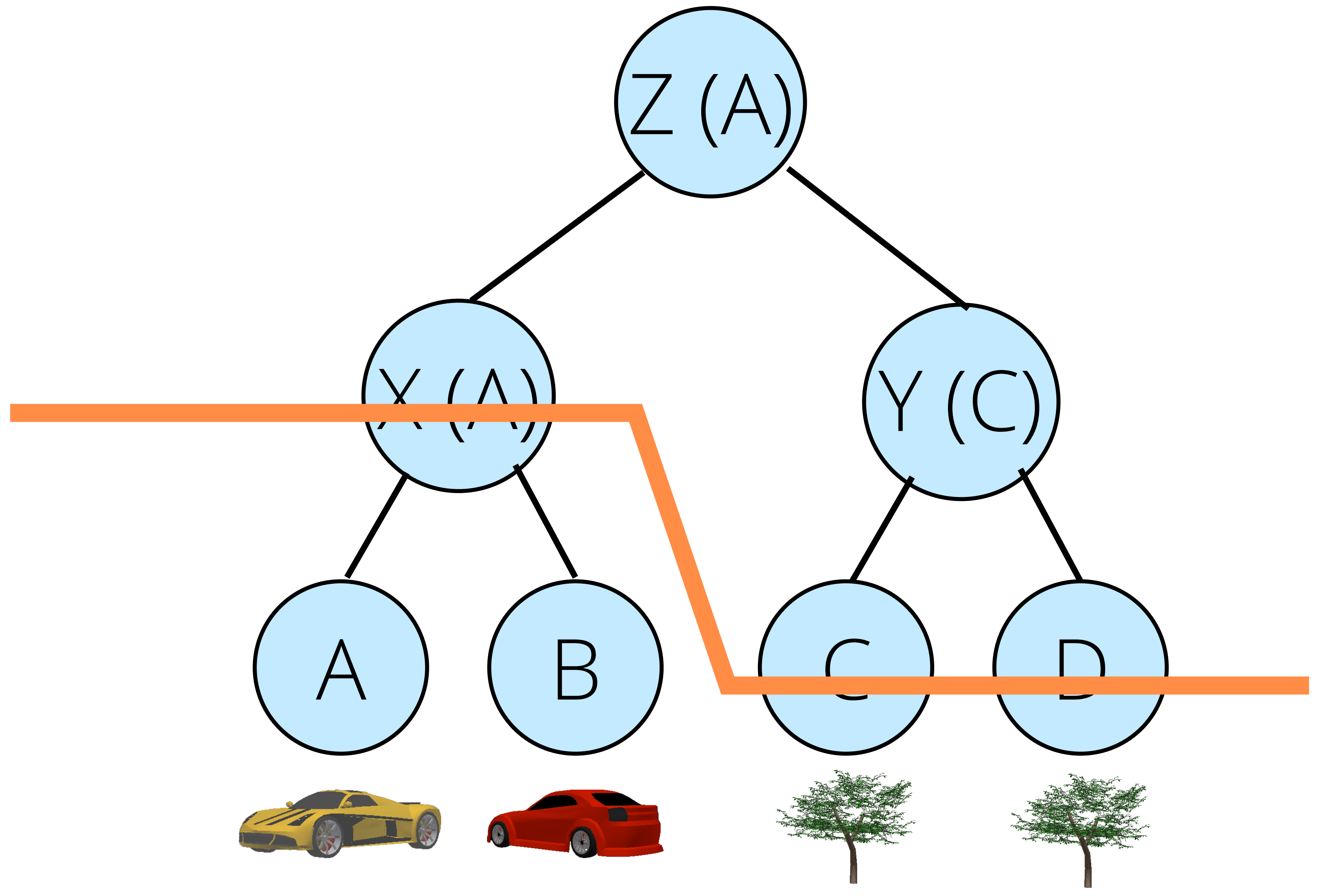
Dynamic Objects

10 - 15% less expensive during short, 100 second experiment

Benefit improves over time

A simple experiment of just 100 seconds shows a 10 - 15% reduction in cost in evaluating a query compared to a single tree. But the benefit actually improves over time because the bounds get worse over time.

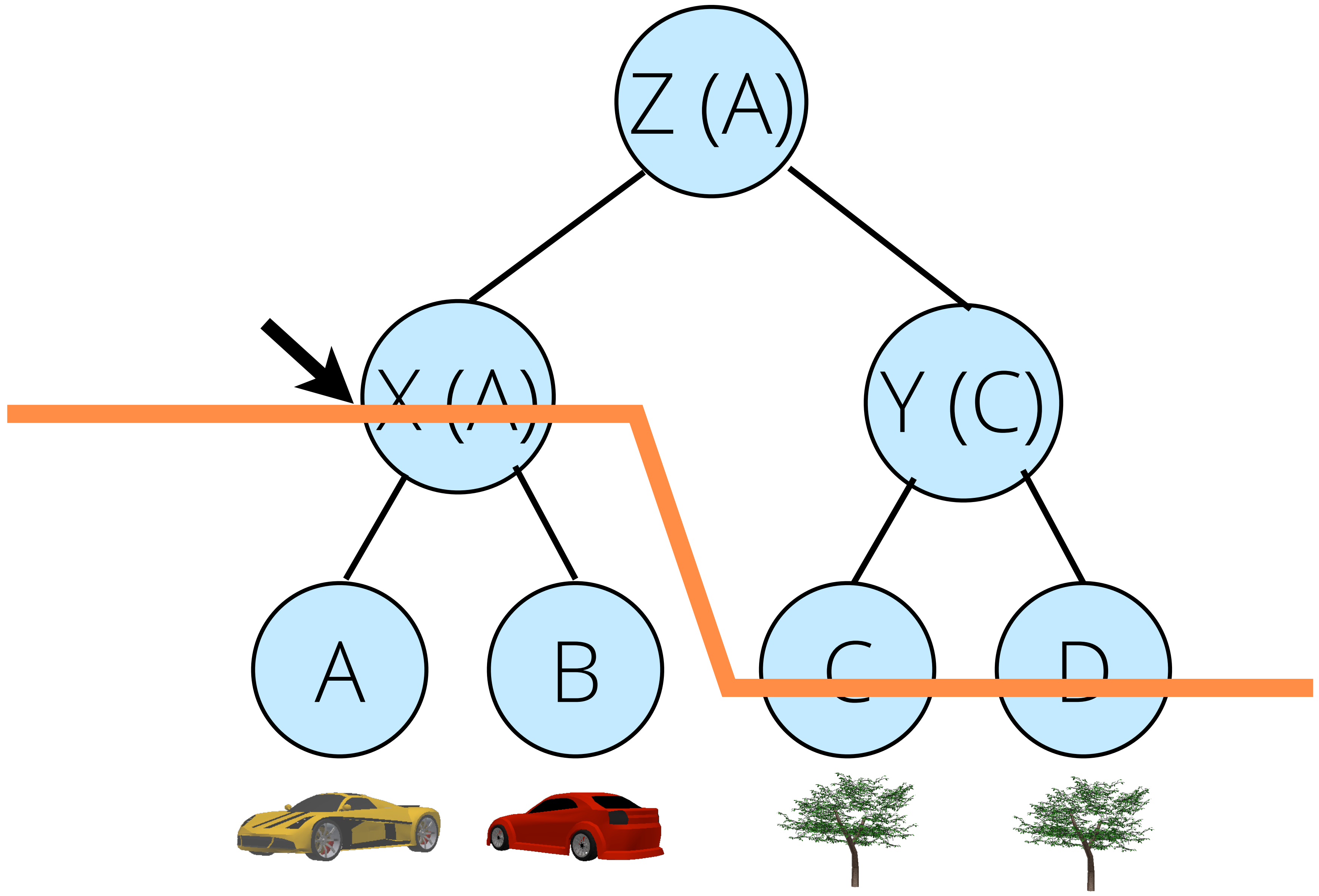
Standing Queries



Cuts avoid redundant work

We also exploit the fact that queries are standing, meaning that we register it once and then continue to receive updates, such as when an object comes closer to us and becomes relevant. To avoid redundant work when updating queries, we maintain a cut through the tree, indicating where we finished evaluating the query on the last iteration. To update the query, instead of starting at the root, we just walk and update the cut, moving it down or up the tree. For example if the querier moved, we might update this cut by walking horizontally, [transition] finding that node X does now satisfy the query...

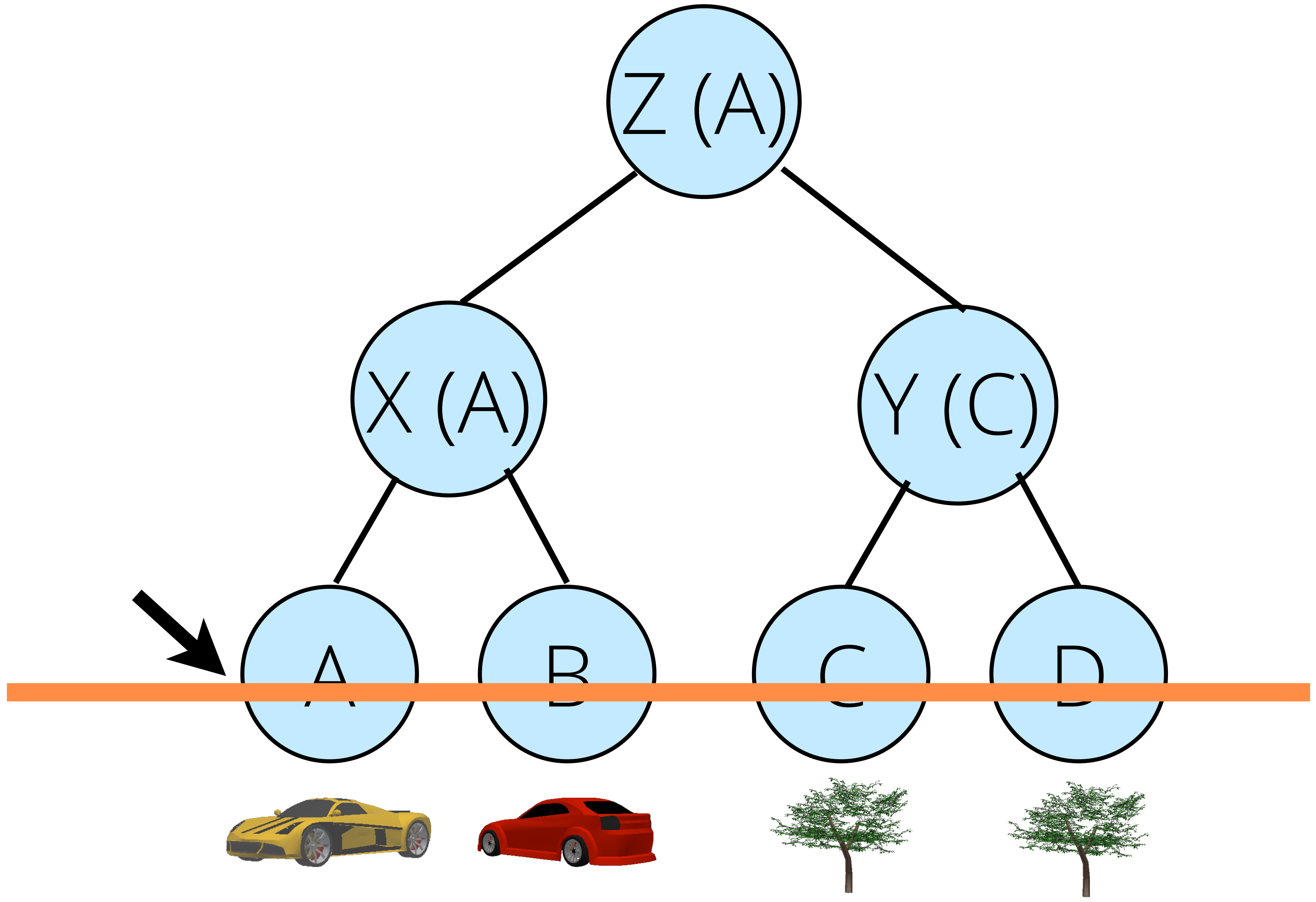
Standing Queries



Cuts avoid redundant work

We also exploit the fact that queries are standing, meaning that we register it once and then continue to receive updates, such as when an object comes closer to us and becomes relevant. To avoid redundant work when updating queries, we maintain a cut through the tree, indicating where we finished evaluating the query on the last iteration. To update the query, instead of starting at the root, we just walk and update the cut, moving it down or up the tree. For example if the querier moved, we might update this cut by walking horizontally, [transition] finding that node X does now satisfy the query...

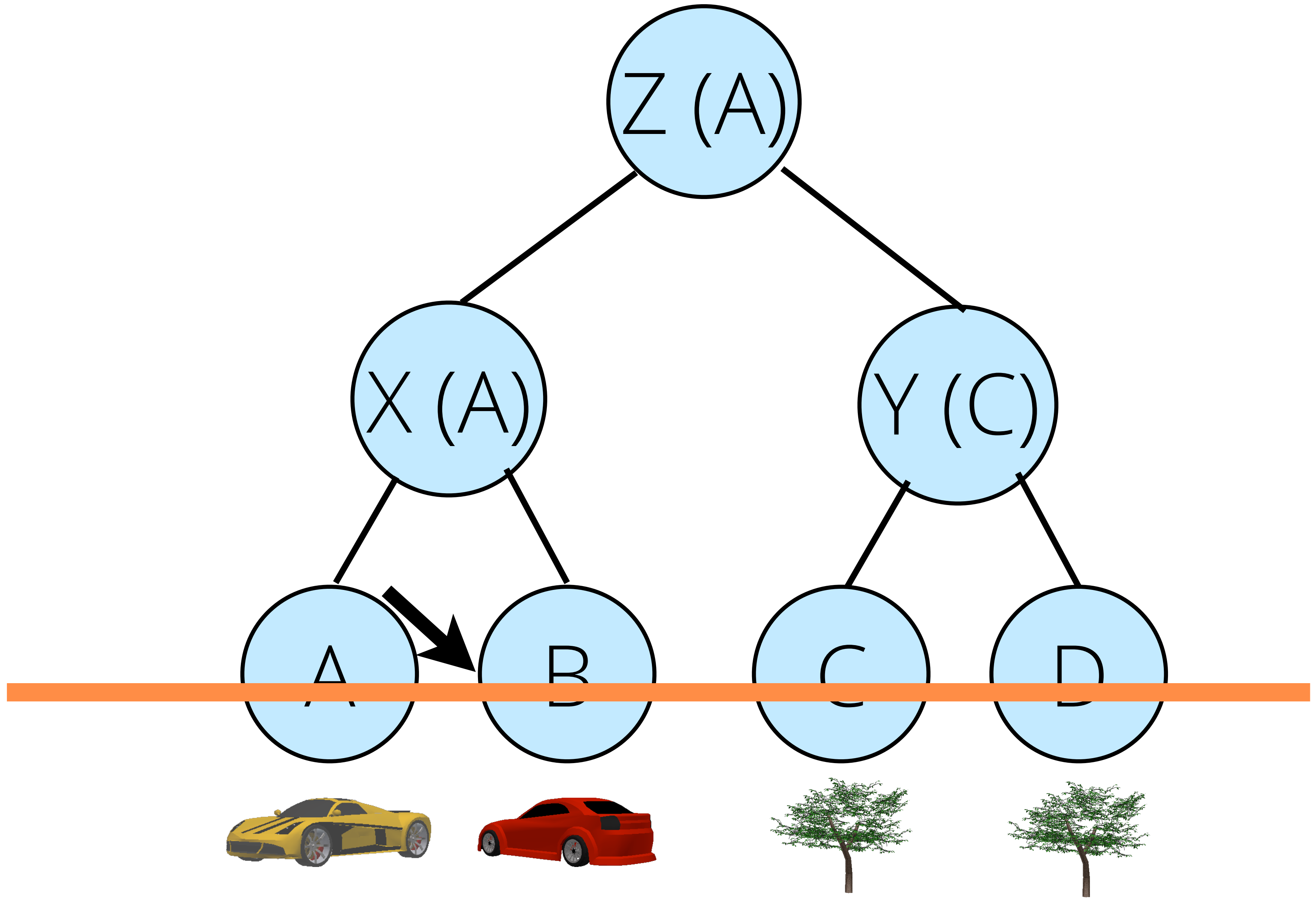
Standing Queries



Cuts avoid redundant work

refine the cut by splitting to nodes A and B. We then recursively evaluate them, checking A and adding it to the results...

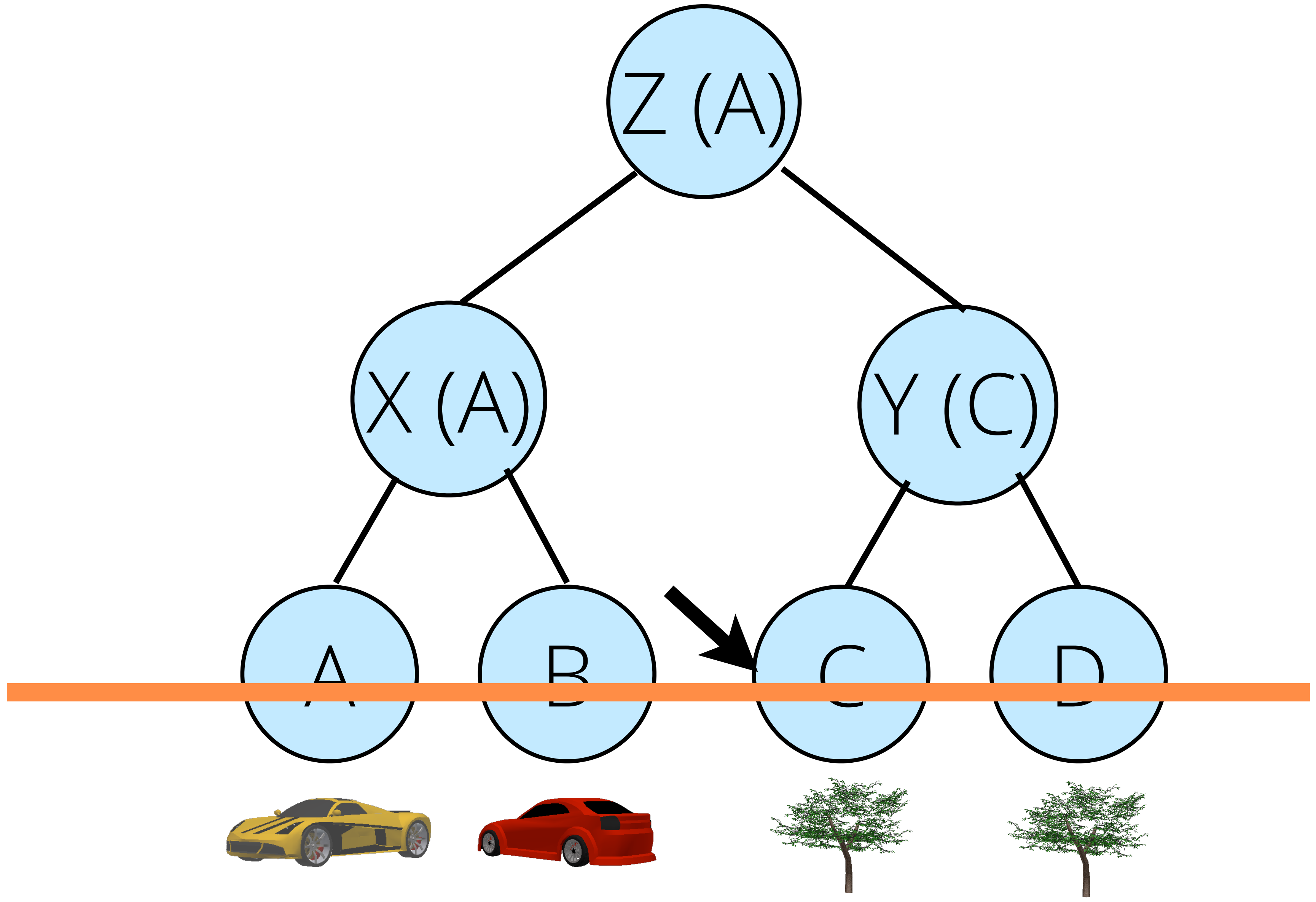
Standing Queries



Cuts avoid redundant work

and the same for B...

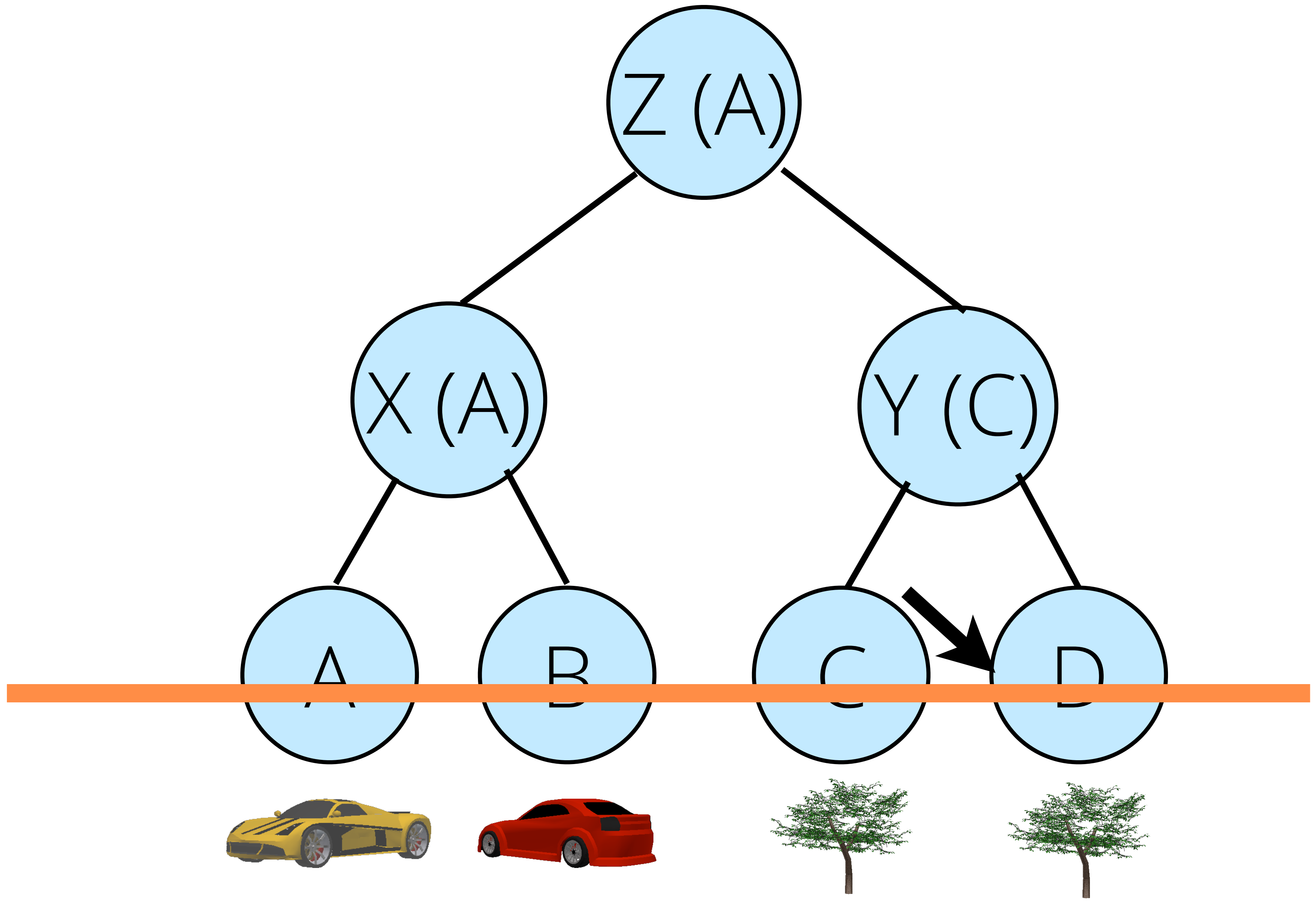
Standing Queries



Cuts avoid redundant work

Now we continue, and let’s say for this example that C now fails the test -- it no longer appears large enough to the querier. We remove it from the results, but don’t change the cut...

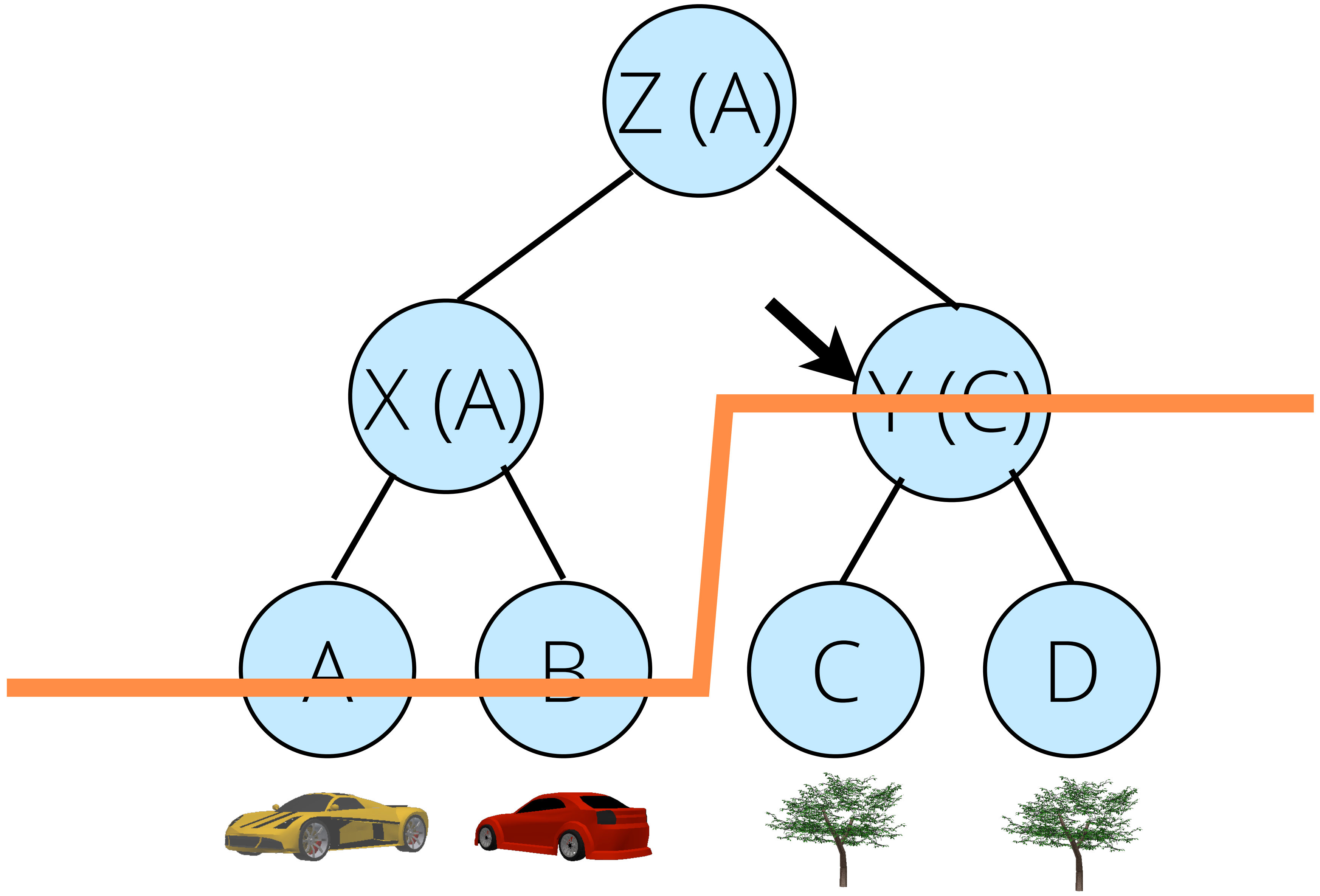
Standing Queries



Cuts avoid redundant work

because we still need to evaluate D, finding it is also no longer part of the results. Now that we've found all children of Y do not satisfy the query, we can move the cut up...

Standing Queries



Cuts avoid redundant work

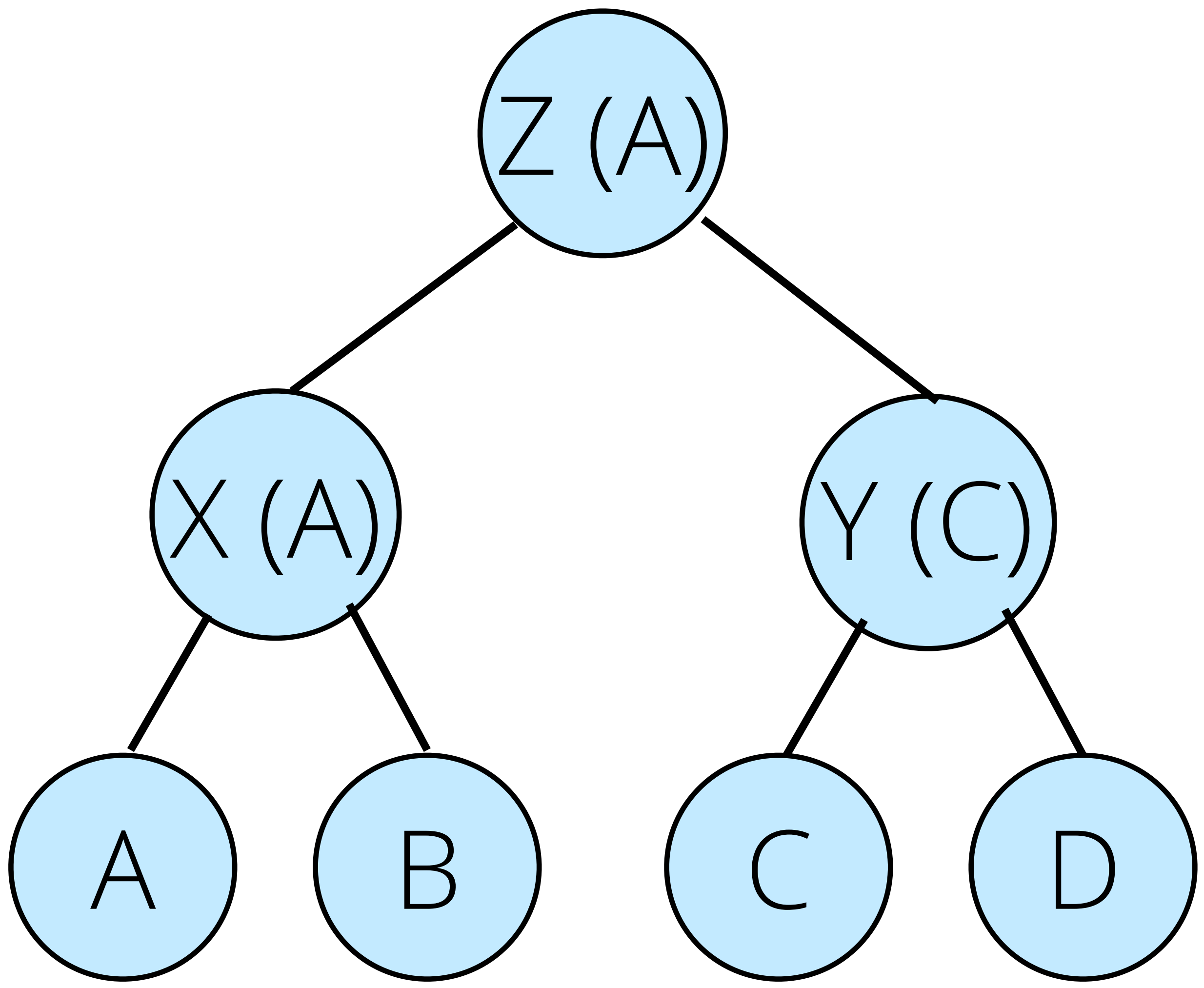
and check Y, then finally finish by leaving the end of the cut.

Standing Queries

20 - 56% increase in
query evaluation rate

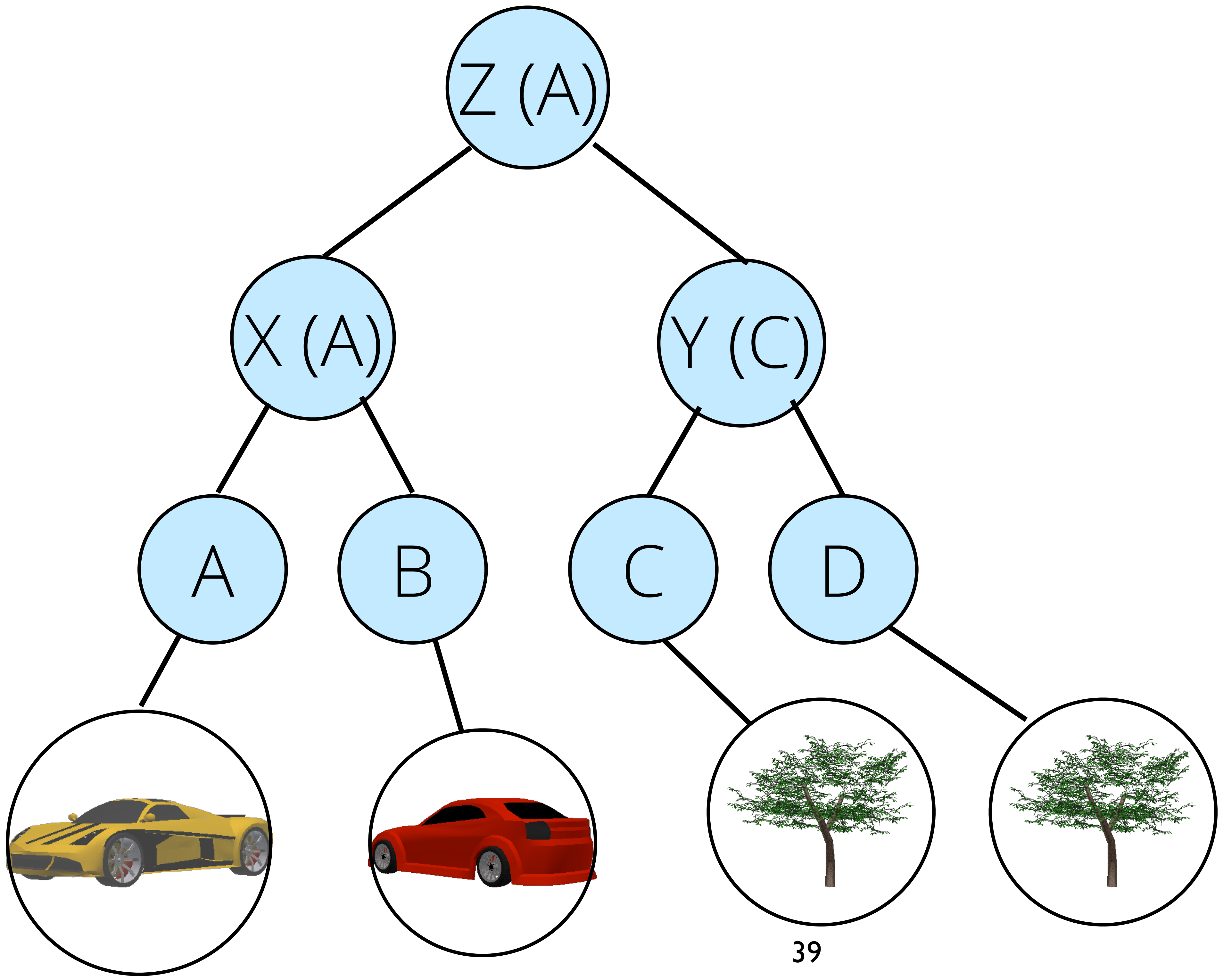
This example doesn’t make it look like a lot of savings, but they can be substantial when you have deeper trees as you avoid retesting all the internal nodes. Exploiting standing queries with cuts improves query evaluation rate by 20% with an LBVH. Aggregates, which I’ll talk about next, change the way cuts work a little, resulting in even more improvement, up to 56%.

Aggregation



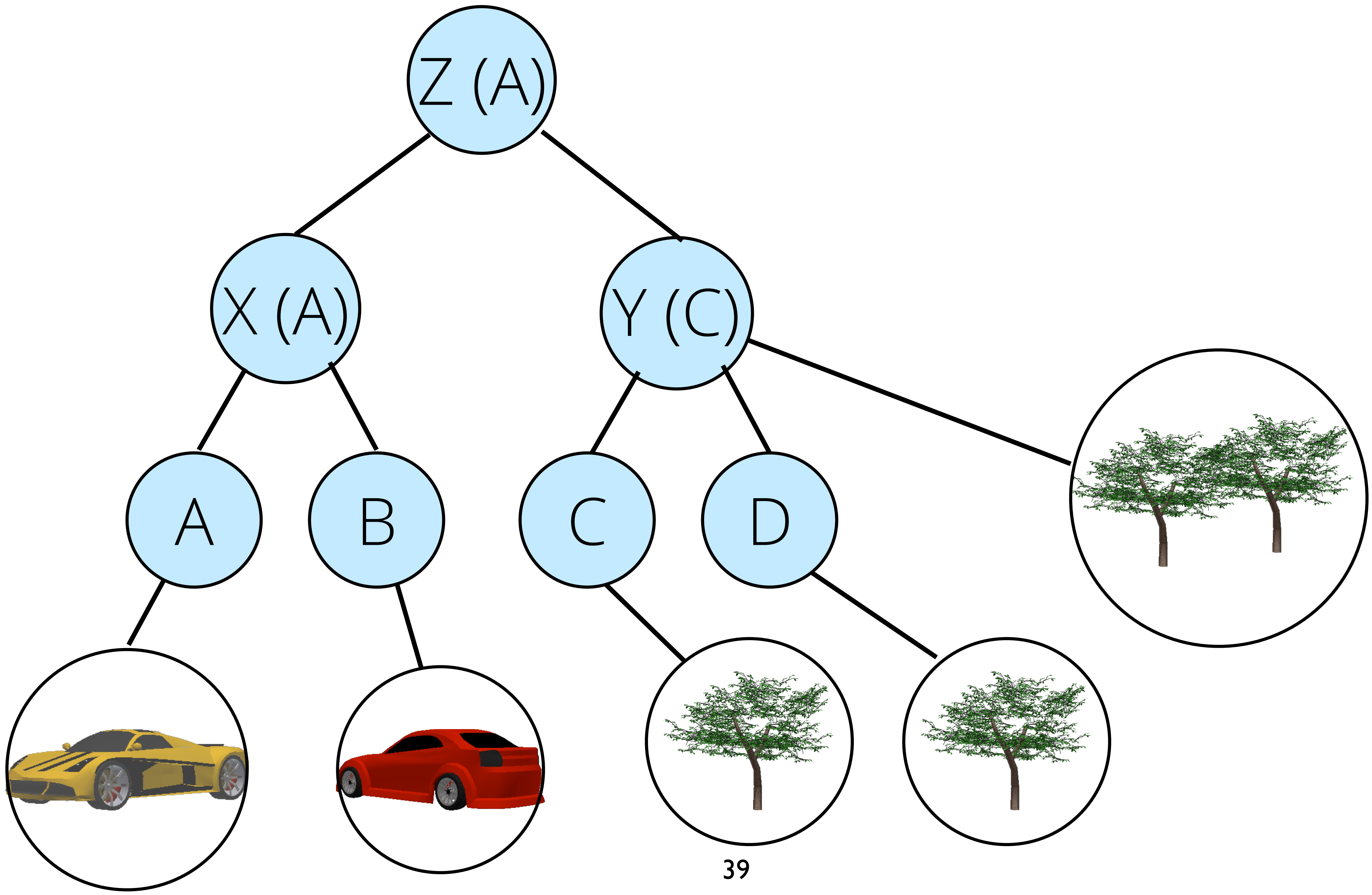
Finally, we use this data structure to generate aggregates. Recall that aggregates are collections of objects which are used if individual objects are too small. Leaf nodes contain single object meshes [transition]. An internal node represents an aggregate of all leaf nodes below it. We generate an aggregate mesh and simplify it so internal nodes have approximately the same complexity. The node Y [transition] above C and D contains 2, simpler trees and the node Z [transition] contains the entire simplified scene. [transition] We use the cut to decide which aggregates to return [transition]. With aggregates we see the entire scene, but some is lower quality.

Aggregation



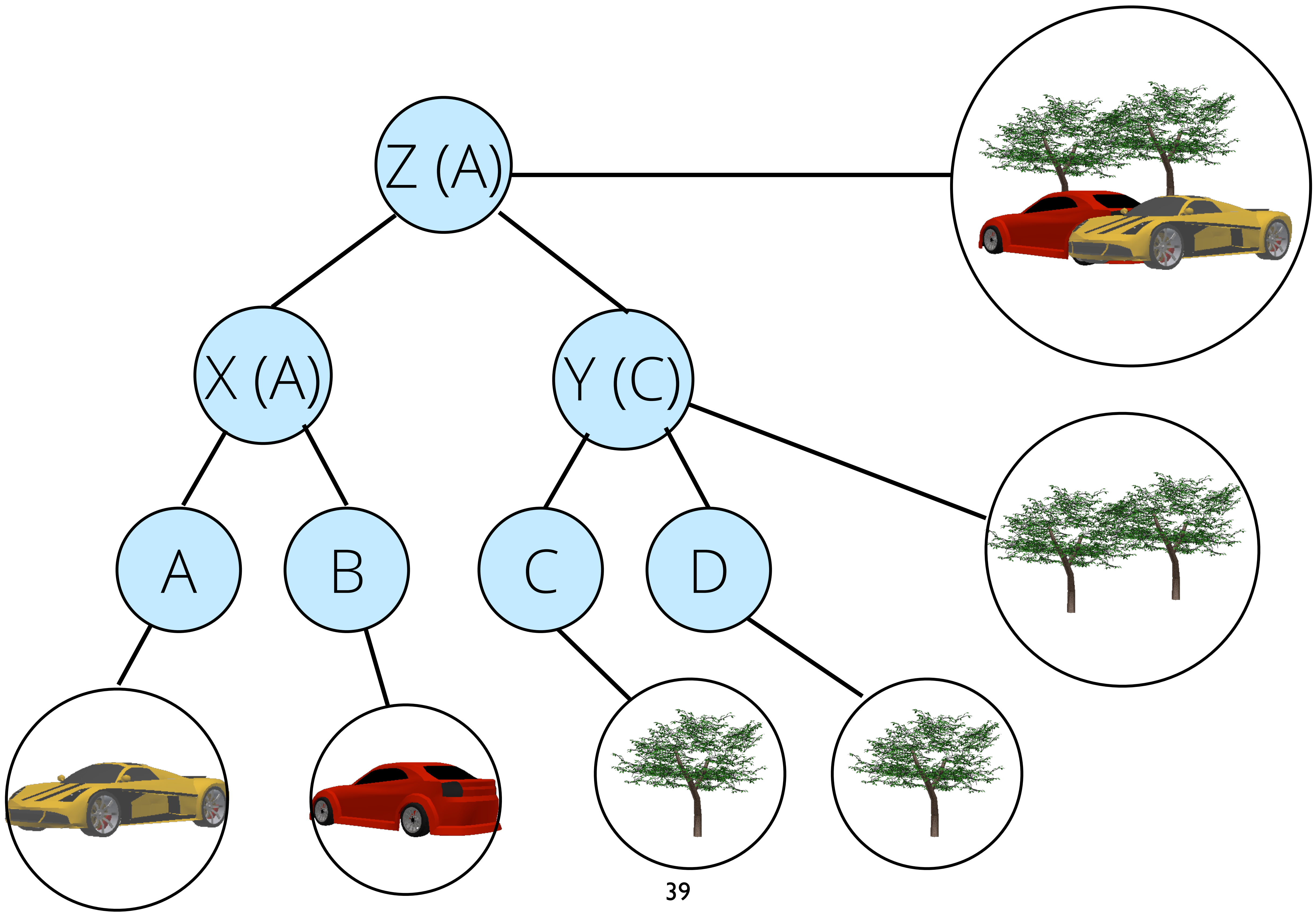
Finally, we use this data structure to generate aggregates. Recall that aggregates are collections of objects which are used if individual objects are too small. Leaf nodes contain single object meshes [transition]. An internal node represents an aggregate of all leaf nodes below it. We generate an aggregate mesh and simplify it so internal nodes have approximately the same complexity. The node Y [transition] above C and D contains 2, simpler trees and the node Z [transition] contains the entire simplified scene. [transition] We use the cut to decide which aggregates to return [transition]. With aggregates we see the entire scene, but some is lower quality.

Aggregation



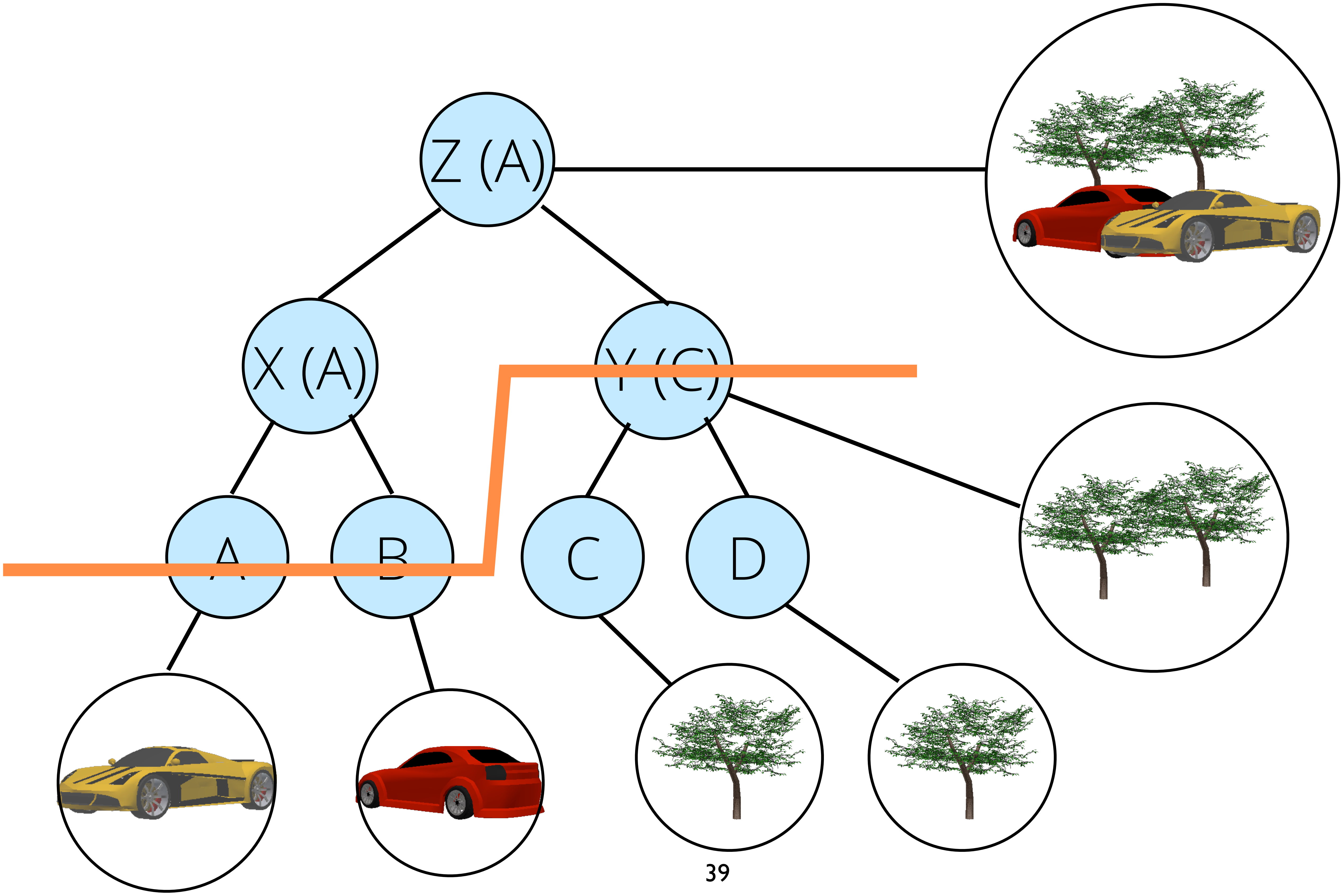
Finally, we use this data structure to generate aggregates. Recall that aggregates are collections of objects which are used if individual objects are too small. Leaf nodes contain single object meshes [transition]. An internal node represents an aggregate of all leaf nodes below it. We generate an aggregate mesh and simplify it so internal nodes have approximately the same complexity. The node Y [transition] above C and D contains 2, simpler trees and the node Z [transition] contains the entire simplified scene. [transition] We use the cut to decide which aggregates to return [transition]. With aggregates we see the entire scene, but some is lower quality.

Aggregation



Finally, we use this data structure to generate aggregates. Recall that aggregates are collections of objects which are used if individual objects are too small. Leaf nodes contain single object meshes [transition]. An internal node represents an aggregate of all leaf nodes below it. We generate an aggregate mesh and simplify it so internal nodes have approximately the same complexity. The node Y [transition] above C and D contains 2, simpler trees and the node Z [transition] contains the entire simplified scene. [transition] We use the cut to decide which aggregates to return [transition]. With aggregates we see the entire scene, but some is lower quality.

Aggregation



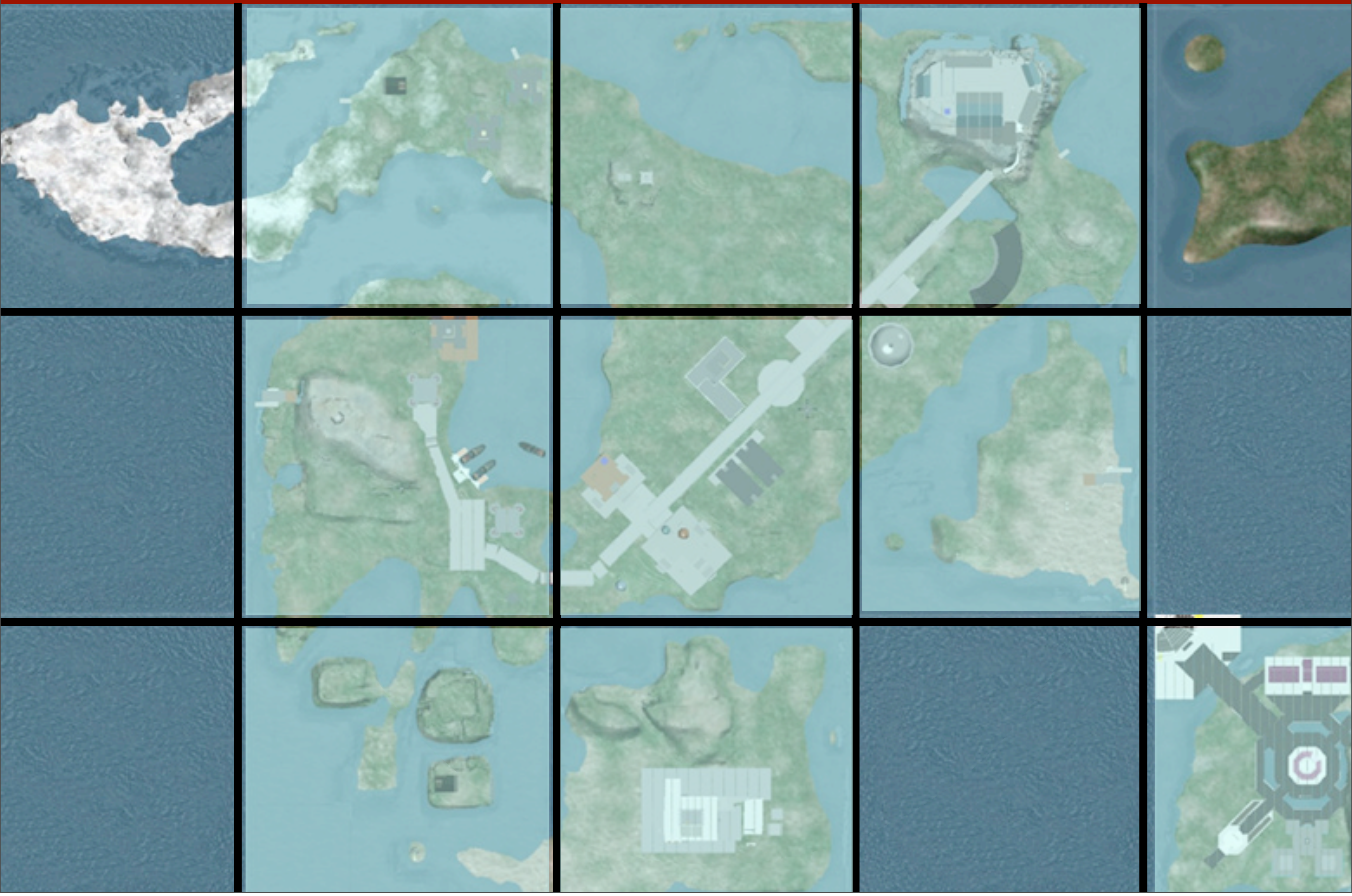
Finally, we use this data structure to generate aggregates. Recall that aggregates are collections of objects which are used if individual objects are too small. Leaf nodes contain single object meshes [transition]. An internal node represents an aggregate of all leaf nodes below it. We generate an aggregate mesh and simplify it so internal nodes have approximately the same complexity. The node Y [transition] above C and D contains 2, simpler trees and the node Z [transition] contains the entire simplified scene. [transition] We use the cut to decide which aggregates to return [transition]. With aggregates we see the entire scene, but some is lower quality.

Aggregate Queries

- Queries on a server are all similar
- Aggregate queries to reduce inter-server querying load
- Filter results further before returning results to querier

Extending to distributed queries is complicated so I'm going to gloss over it a bit -- see the paper for details. At a high level, we keep distributed queries cheap in two ways. First, we recognize that all queries from objects on a server will be similar because all the queriers are near each other. Therefore, instead of having every server answer every query, we aggregate all queries on a server and send a single query to other servers. The origin server then filters results further for individual objects.

Server Discovery

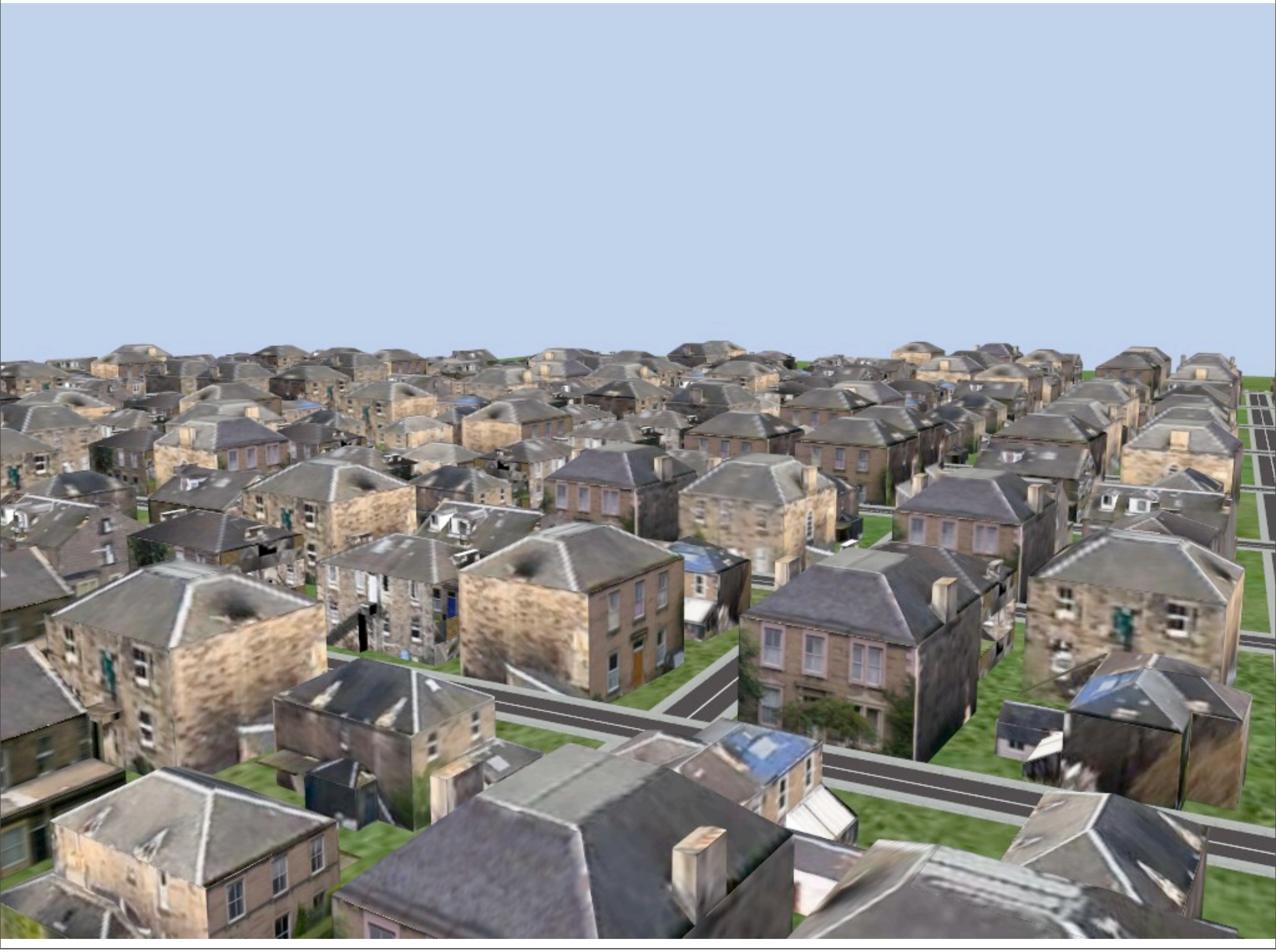


This only requires 1 query for each other server, but that's still a lot if your world is large and run by thousands of servers. However, the natural distribution of object sizes lets us do better. The queries from a server will mostly get results from nearby servers, and maybe a few results -- very large objects -- from a few, more distant servers. Therefore, we can collect the same information about servers -- bounds and largest objects -- and reduce the number of servers that need to be queried by ignoring servers that don't even have a large enough object to be returned.

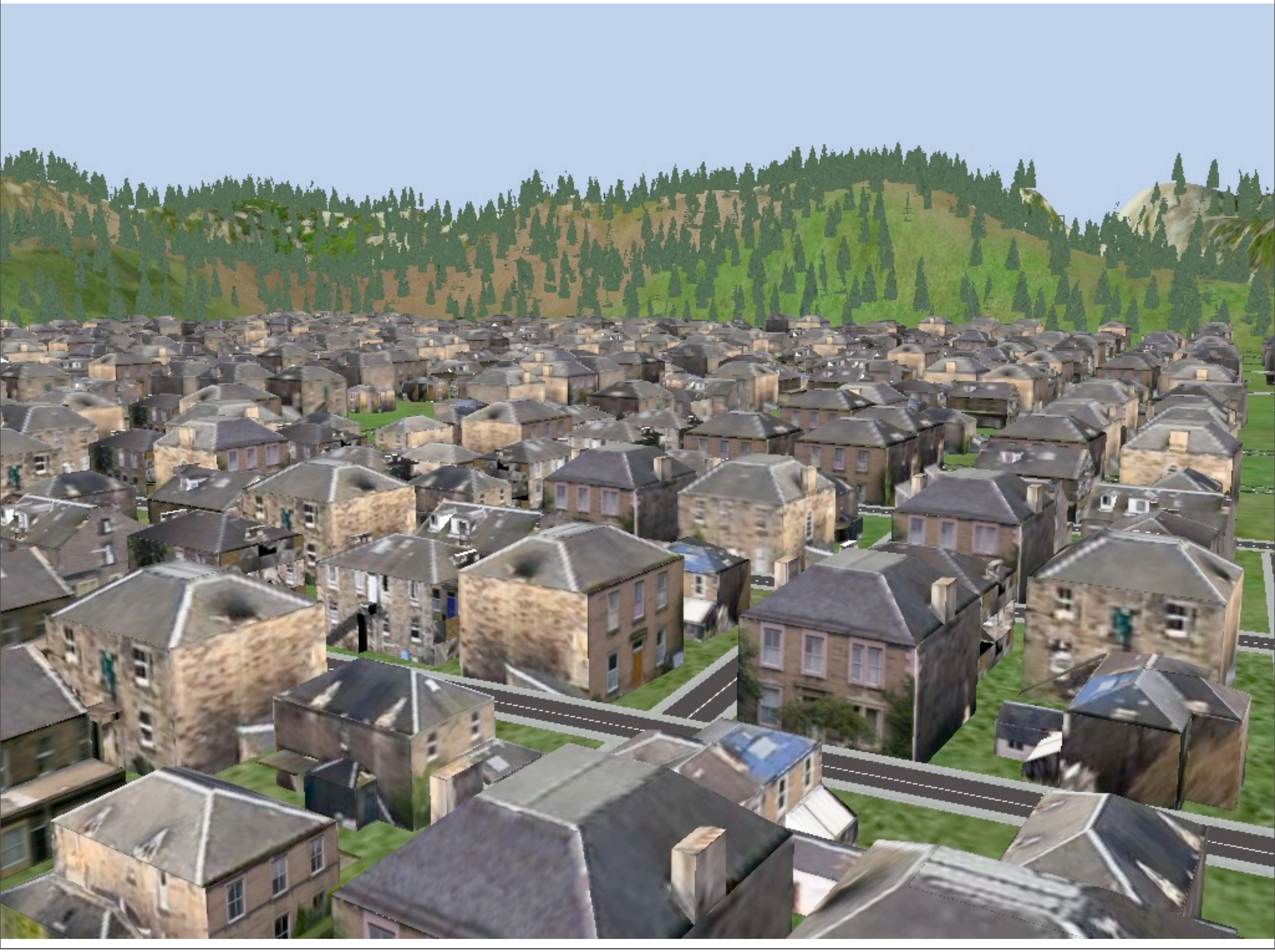
Server Discovery

- Geometric server discovery
- Determine which other servers need to be queried
- Built on same LBVH data structure

We call this geometric server discovery. This is a natural extension of the LBVH to servers and has a similar effect: it can reduce the number of servers queried by an order of magnitude.



Once we put these all together, we go from this type of image in current systems...



....to this final display of a large world with many objects. The view is complete, although at lower quality for some regions. This querying isn't just for display either, it also bootstraps further interaction through messaging. The query format works well for that as well -- you're only able to start communicating with objects you were able to make out visually (or learned about indirectly).

Also in the Paper

- Globally consistent distributed data structure mapping regions to servers
- Global routing table enabling all-pairs communication
- Forwarder with intuitive, physically-based weighting emphasizing local traffic

The paper addresses other scalability challenges as well. Here I'll just give the flavor of each.

* First, we need a globally consistent distributed data structure that maps regions of the world to servers, allowing any server in the system to direct objects to the appropriate server for a location. This data structure exploits the fact that at large scales, object distributions move slowly.

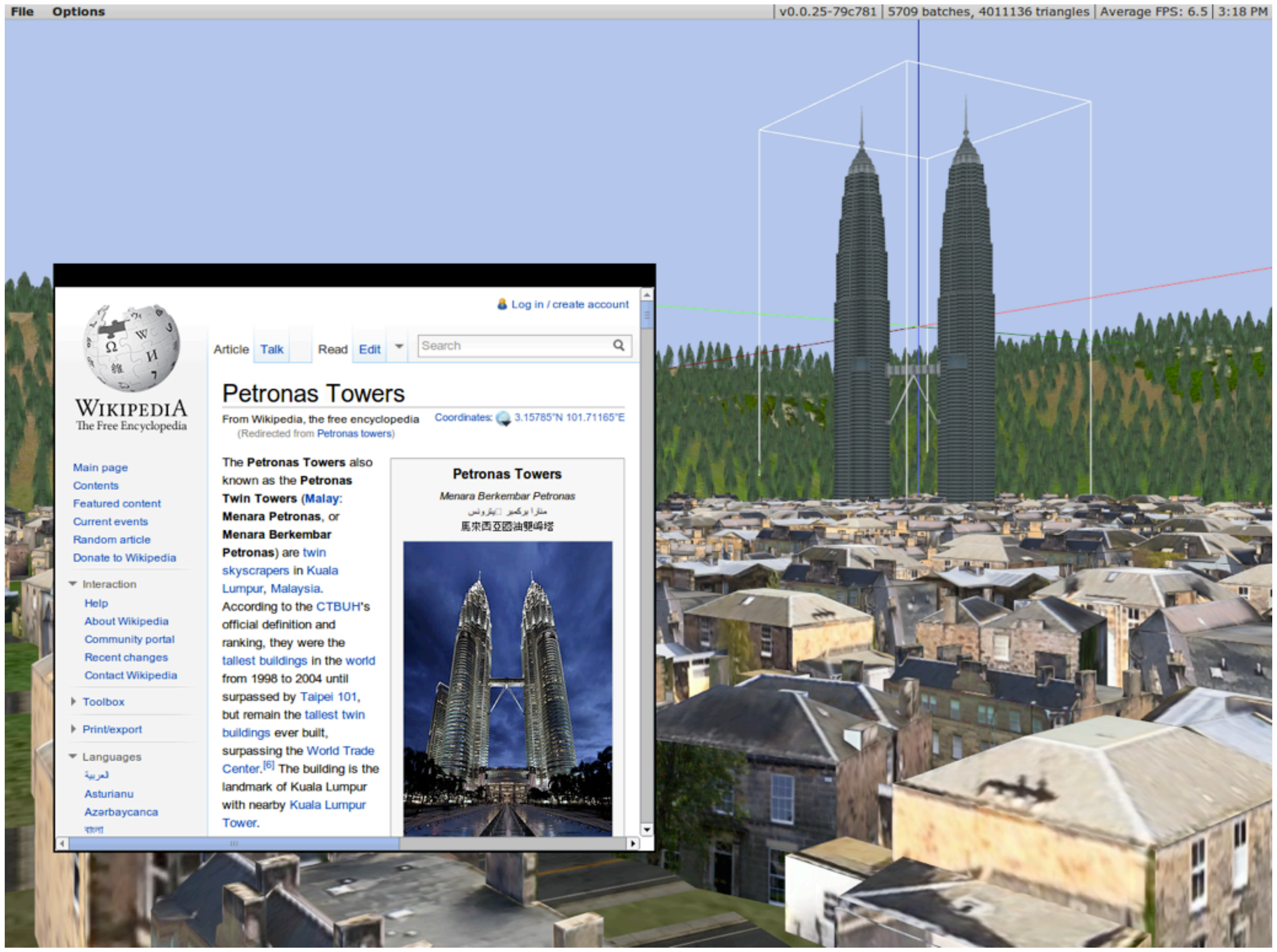
Also in the Paper

- Globally consistent distributed data structure mapping regions to servers
- Global routing table enabling all-pairs communication
- Forwarder with intuitive, physically-based weighting emphasizing local traffic

* To allow all pairs of objects to communicate, ensuring we don't restrict interaction through messaging, we have a global routing table enabling routing to any object in the system. Aggressive caching allows us to scale this to billions of objects.

* Finally, to get messages to their destination, or drop them under load, the forwarder uses an intuitive, physically-motivated weighting scheme which emphasizes local traffic, giving much nicer behavior than naive uniform weighting or the drastic cutoffs used in other systems.

Wiki World



Petronas Towers

From Wikipedia, the free encyclopedia Coordinates: 3.15785°N 101.71165°E
(Redirected from Petronas towers)

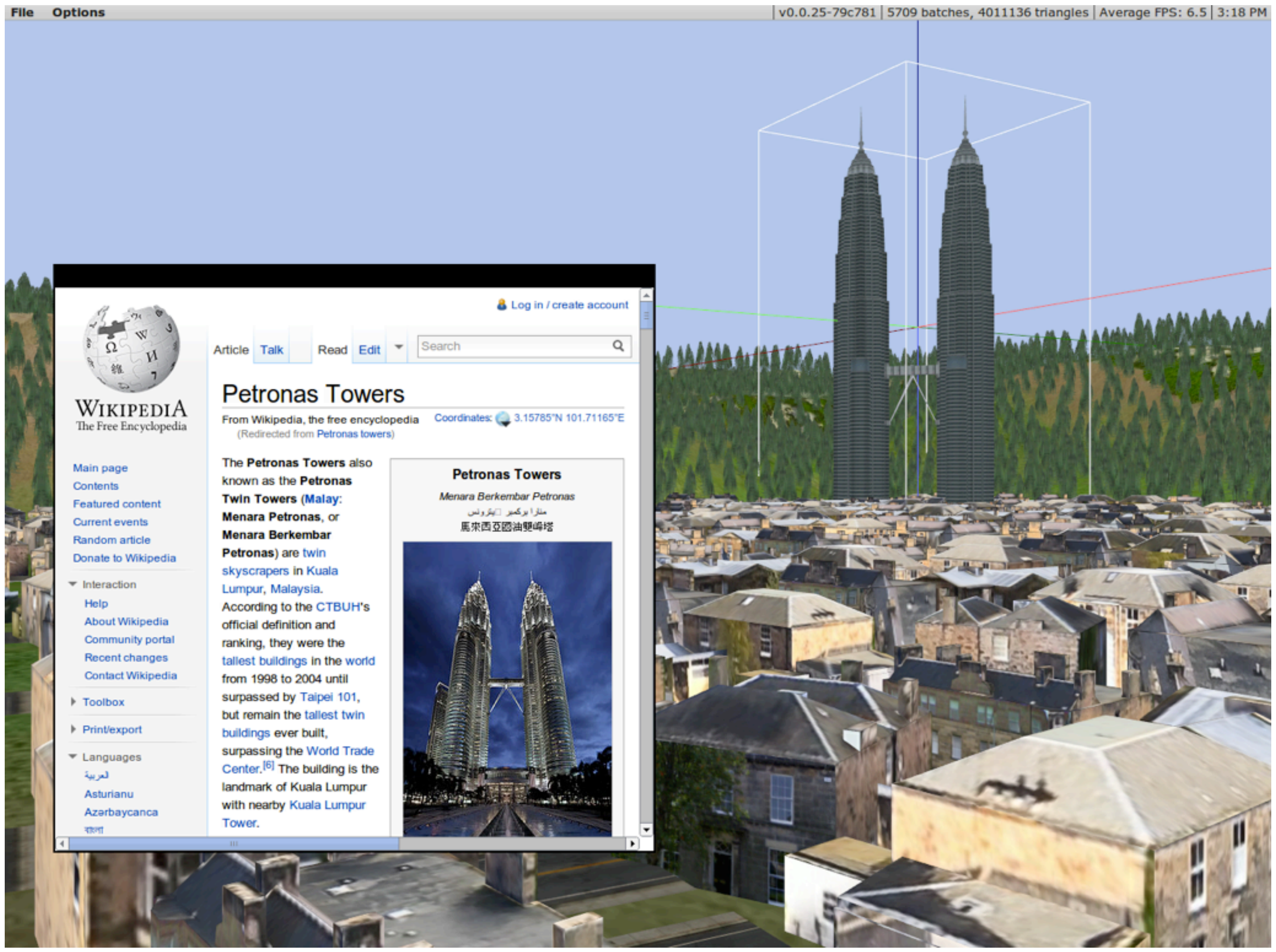
The **Petronas Towers** also known as the **Petronas Twin Towers** (Malay: **Menara Petronas**, or **Menara Berkembar Petronas**) are twin skyscrapers in Kuala Lumpur, Malaysia. According to the CTBUH's official definition and ranking, they were the tallest buildings in the world from 1998 to 2004 until surpassed by Taipei 101, but remain the tallest twin buildings ever built, surpassing the World Trade Center.[6] The building is the landmark of Kuala Lumpur with nearby Kuala Lumpur Tower.

Petronas Towers
Menara Berkembar Petronas
مناارا بركمبر ڤيترونس
馬來西亞國油雙峰塔

Automatically find information about objects on Wikipedia

So what kinds of applications does Sirikata enable with these new features? One example that a student built last summer is called wiki-world and it helps with that first task a user performs when they join a world -- finding something interesting to interact with. For example, when I join the world, as objects load, I might click on a distant building and pull up wiki-world. It collects tags and search terms from the object using messaging, searches wikipedia for relevant articles, and presents them to the user.

Wiki World



Petronas Towers

From Wikipedia, the free encyclopedia Coordinates: 3.15785°N 101.71165°E
(Redirected from Petronas towers)

The **Petronas Towers** also known as the **Petronas Twin Towers** (Malay: **Menara Petronas**, or **Menara Berkembar Petronas**) are twin skyscrapers in Kuala Lumpur, Malaysia. According to the CTBUH's official definition and ranking, they were the tallest buildings in the world from 1998 to 2004 until surpassed by Taipei 101, but remain the tallest twin buildings ever built, surpassing the World Trade Center.[6] The building is the landmark of Kuala Lumpur with nearby Kuala Lumpur Tower.

Petronas Towers
Menara Berkembar Petronas
馬來西亞國油雙峰塔

Automatically find information about objects on Wikipedia

This is a very simple application, but the results you get in Sirikata wouldn't have been possible in other systems. For a large class of objects which are very relevant to the user, like the Petronas towers seen here, other systems would not have returned them because they limit visibility to only nearby objects. Other constraints in those systems would limit further interaction, through messaging with this object.

But wait, there's more...

There are many more systems challenges at the intersection of systems, graphics, PL, databases, ...

A few examples:

- Audio: distant siren, roar of a crowd
- Efficient property updates

We've come a long way over the past couple of years, but we still have a lot to do. We hope Sirikata can be the basis for additional research at this interesting intersection of systems, graphics, programming languages, databases, and other areas.

But wait, there's more…

There are many more systems challenges at the intersection of systems, graphics, PL, databases, …

A few examples:

- Audio: distant siren, roar of a crowd
- Efficient property updates

To give a taste for some other challenges we've encountered, here are a few other examples of challenges. How would you add audio support to this large distributed system that can convey both a loud, distant siren and the roar of a crowd, handling thousands of streams, mixing them spatially, and doing so under the latency constraints of real-time audio? Or how can we most efficiently propagate property updates, where we might not care about intermediate values, but the stable state must be reliably provided?

But wait, there's more...

There are many more systems challenges at the intersection of systems, graphics, PL, databases, ...

A few examples:

- Audio: distant siren, roar of a crowd
- Efficient property updates

Novel challenges like these arise throughout the system, and we think this community has a lot to offer in this space.

Thank You

Download and code at

sirikata.com

Questions?

If you’re interested in seeing Sirikata for yourself, we have binaries for download and a couple of test worlds, or videos if you don’t want to install the client. We also hope you’ll check out the code and we’d love to see contributions and new extensions. All of this can be found at sirikata.com.