JUMP POINT

ISSUE: 07 10

IN THIS ISSUE →→→

Editor: Ben Lesnick Copy Editor: Martin Driver
Layout: Michael Alder

FROM THE COCKPIT

GREETINGS, CITIZENS!

It goes without saying that October is a big month for *Star Citizen*. I have incredibly fond memories of our very first reveal at GDC Online back in October 2012 and then that first frenzied CitizenCon the following year. And now, the launch of Alpha 3.7 October will mean all the more to *Star Citizen* history! It's exciting to see all sorts of elements come together with each patch and this one seemed especially rewarding, watching players charting enormous cave systems and getting their first flight time with a Banu ship.

You may notice that we're doing things a little differently this issue by having a double-length feature on Server-Side Object Container Streaming. We typically conduct an interview with developers to get the latest on new features, but this time around, the topic ended up being so important and complex that the team behind it offered to write the article themselves. So, this one is straight from the horse's mouth, as it were! That's one of the very exciting, unsung aspects of *Star Citizen* as far as I'm concerned: the teams working on these features are just as excited to share them and to help the community understand them as you are to hear about them. So a very special thank you to Christopher Bolte, who took time out of his extremely busy schedule to put this article together, as well as the whole team who helped make it happen: Carsten Wenzel, Steven Humphreys, Chad McKinney, Clive Johnson, Ivo Herzeg and Silvan Hau.

What, then, is Server-Side Object Container Streaming? Well, they answer that question with a lot more accuracy and information than I possibly could... but in short, it's the technological framework that lets *Star Citizen* become a seamless multiplayer universe. It's something the team has been putting so much work into and is, frankly, too difficult to explain without a special article like this. Maybe it's harder to understand than a new spaceship or environment or career, but it's the invisible umbrella that's going to allow all of those things to matter. But just like we have developers who go above and beyond to explain their work to the community, we have a very special community that goes above and beyond in order to understand this kind of thing. I hope you enjoy the article! If you're missing the usual article about the making of our latest ship, the already-flyable RSI Mantis, check back next month.

On the lore side of things, we've got a brand new Whitley's Guide exploring the history of the Crusader Starlifter. Learning about starfighters and battleships and their magnificent victories is always fun, sure, but I think there's something special about filling in the universe with the history of the support ships that make those grand battles possible. The Hercules is a shift that carries the UEE military on its back and it was fun to explore how it came to be. We've also got A Day in the Life of a G-Loc Bartender, complete with drink recipe. That's it for October save for this small challenge; take your October flare and go scare some unexpecting cave explorers!

I'll see you in the 'verse.

Ben

JumpPoint@cloudimperiumgames.com

THE ROAD TO OBJECT-CONTAINER-STREAMING

CHRISTOPHER BOLTE

CHAD MCKINNEY

CLIVE JOHNSON

Hello, my name is Christopher Bolte, Principle Engine Programmer at Cloud Imperium. As someone involved in nearly all the steps of the development of "Object-Container-Streaming", I would like, on behalf of the whole team, to give an overview about the technical challenges we worked on over the years to deliver this technology.

In this article I will first cover what Object-Container-Streaming is. Afterwards, the technological limits which must be overcome are explained, followed by a short look at how such large and complex features are developed.

Following that, the article will focus on the individual parts and already achieved milestones leading towards Object-Container-Streaming.

WHAT IS "OBJECT-CONTAINER-STREAMING"

Before going into technical details, we should understand what Object-Container-Streaming should provide for the player.

In short, Object-Container-Streaming is the umbrella term for all the technology that makes a vast seamless universe possible, by which we can provide an extremely large virtual world through which the players can move without seeing a loading screen.

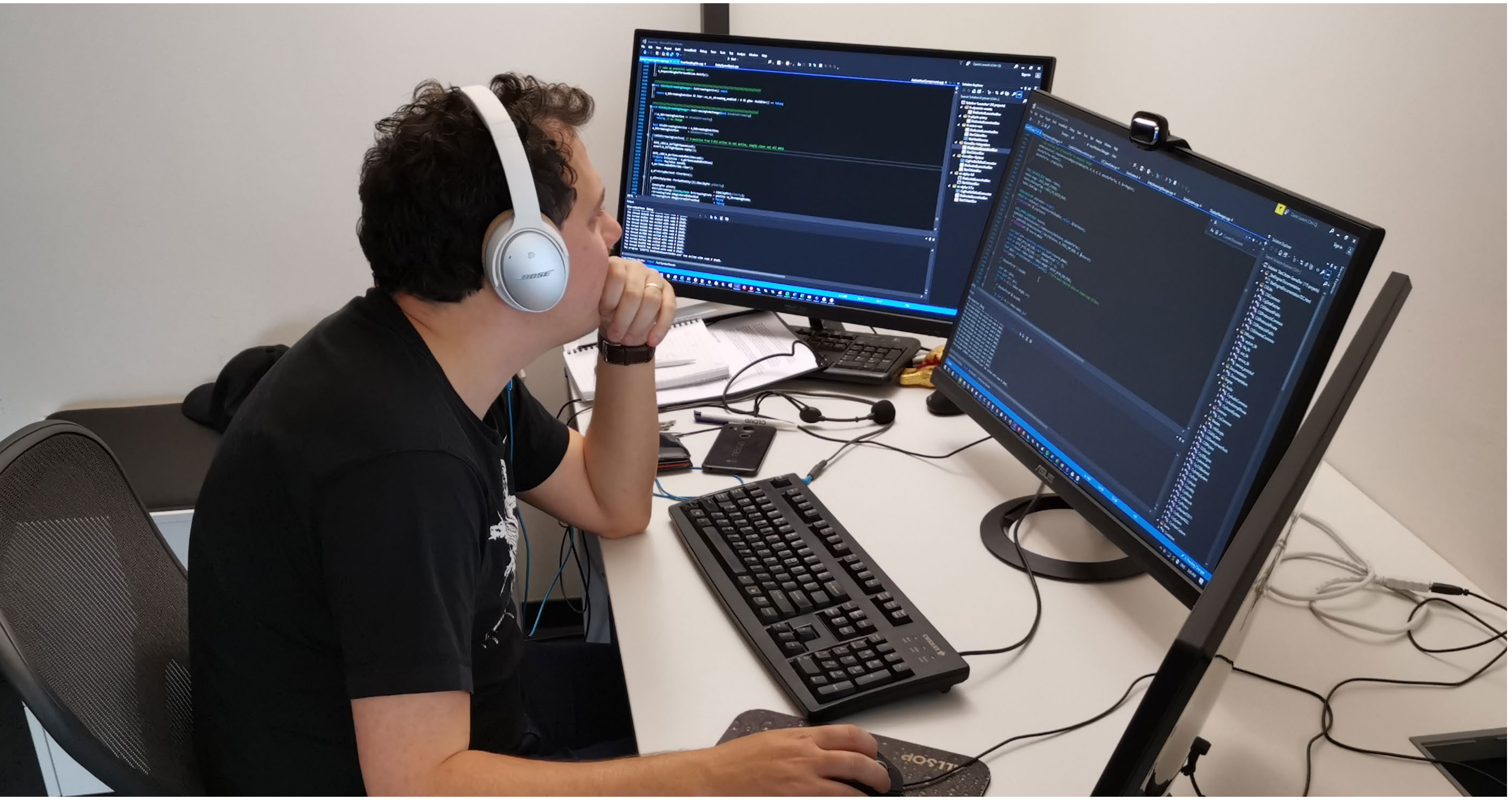
TECHNOLOGICAL LIMITS OBJECT-CONTAINER-STREAMING MUST OVERCOME

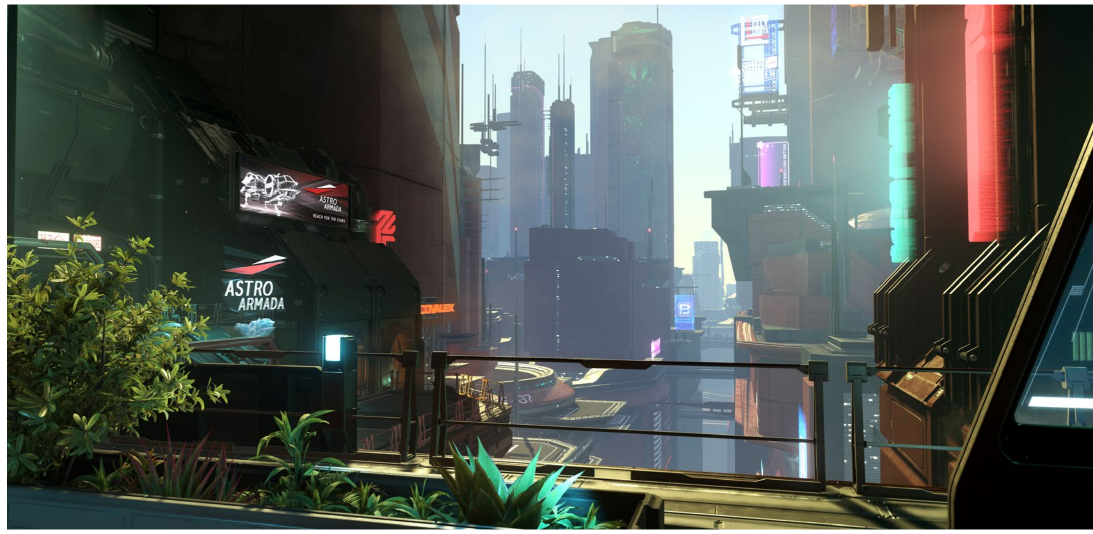
But what implications does that goal have for the engine technology? Foremost, a videogame must refresh its screen at least 30 times a second to give the impression of a fluent experience.

To achieve 30 frames per second (FPS), the game must perform all necessary computations for the whole frame within 0.033 seconds (33 milliseconds). Failing to stay within this time limit will cause stuttering, as the impression of fluent motion is broken when the frame isn't refreshed often enough and the human brain starts to recognize individual images.

LIMITED RAM AND FILE TRANSFER SPEED

Moreover, a PC has a limited amount of 'random access memory' (RAM). RAM is a very fast memory, which can archive 20.000 or more megabyte per second (MB/s) in transfer rate. To ensure a stutter-free experience, it is necessary that all data access is happening inside RAM, and not on the much slower hard drive.





Since RAM is limited and magnitudes smaller than all the game data we want to provide, we must, while the game is running, load data from the hard drive and replace other no-longer-needed data. And that takes time. The transfer of only 10 MB with a fast 500 MB/s hard drive requires 0.2 seconds to load the data, which translates to ~6 frames with 30 FPS, resulting in noticeable stuttering.

Therefore, all file transfers must be done in a way to not affect the game simulation while the game is being played, to ensure a fluent experience while traveling through the seamless universe.

MULTITHREADED PROGRAM EXECUTION

This brings us to multithreading. Each central processing unit (CPU) in the last ~10 Years has multiple CPU cores. Each CPU core can execute program instructions independently from each other. This (and other techniques I omit for easier understanding) allows computers to do more than one thing at the same time. In the case of Object-Container-Streaming, we can load resources in parallel to game logic, thus not affecting the game's frame update rate.

Loading a resource is more than just file transfer time. After loading from the hard drive, the resource data must be initialized - something that can take time and increases the time till the resource can be used. A similar issue arises when we unload the resource, as we then need to de-initialize the resource, which also takes time.

But the worst issue is communication. A game engine has several central managers for certain resource types (like textures or characters). Those managers normally maintain a list of loaded instances of their resource type and provide operations on them. A simplified example would be a character manager, which maintains a list of all loaded characters. Each frame, the game simulation asks the character manager to update all loaded characters. And here it becomes tricky. If our character resource loading finishes in parallel to the character update, it wants to put the newly loaded character into the character manager's list, so that the character will receive updates in the future. Putting the character into the list modifies it. If the game simulation is using this list at the same time to update the characters, this can result in a data corruption and crash the game.

Thus, for a correct program execution, only one execution path should access such a manager at one time, hence mutual exclusion must be applied; if the game simulation is using the manager, resource loading must wait, and vice versa. And waiting takes time, which we don't have due to the 0.033 seconds time limit, bringing us to the main complexity of multithreaded programming: finding the minimal amount of communication necessary while the program still executes correctly. If this is not done correctly, the game simulation could wait for the whole file transfer, which would again result in unwanted stuttering. Or the game simply crashes from time to time due to data corruptions.

Therefore, the whole technological stack must be designed around concurrent resource loading, initialization, and destruction while minimizing communication to not affect the game simulation.

DEVELOPING OBJECT-CONTAINER-STREAMING IN A LIVE PRODUCT

Object-Container-Streaming has been in development for several years. Some steps were very visible to players, like Network-Bind-Culling, others, like removing LUA and reworking legacy code, less so.

One of the major hurdles in developing all those is the fact that we are a live product. We have regular releases (or not so regular in the past, something on which we improved). We also do constant feature development to build the game. Because of feature work and releases, we cannot have a none-working version of the game for several months. At the same time, for Object-Container-Streaming, we are changing the fundamental laws against which those game features are developed. Therefore, for each step we take, we must look at the impact on the schedule and what feature re-work must be done. Based on this, we have to decide or come up with ways to introduce the Object-Container-Streaming changes in a way that allows the game teams to gradually adapt to those new laws while keeping the game working.

THE GROUNDWORK

OBJECT-CONTAINER CONCEPT

Several steps had to be implemented before we could even consider developing Object-Container-Streaming. Roughly five years ago, we began introducing the Object-Container concept. Before that, our engine only supported 'levels'. A level is a list of game objects. A game object itself is a collection of resources typically referred to as an 'entity'. Before a level can be played, all entities must be loaded into RAM from the hard drive and be initialized, which normally happens behind a loading screen.

Object Containers are a level building block. Their concept is that instead of developing one large level, the content creators develop a small section. The final level (or universe in our case) is then made out of many different Object-Containers. This concept allows us, at level building time, to split the world into many smaller building blocks. And building streaming on top of that allows us to load and unload those building blocks at runtime, ensuring we fit into the RAM budget providing the seamless universe. While the Object-Containers didn't provide a noticeable impact to the players (since for a long time, we simply loaded all object containers during level load), they were an important steppingstone.

LUA AND COMPONENTS

Two other major requirements were started around two and half years ago.

First, we began to work on removing LUA from the game code. LUA is a scripting language which was used heavily for all kinds of entity logic within the engine. The problem with LUA was that it was impossible to make multi-thread safe. In other words, as long as we used LUA, we couldn't execute resource loading in parallel to the game simulation without introducing very long wait times due to required mutual exclusion. Hence all LUA code was replaced with C++, which gives us sufficient control to prevent such wait times.

At the same time, we began converting our entities from larger monolithic objects into components. An entity component represents a part of specific game behaviour. With components, the behaviour of an entity is defined by the types of components it has. Without components, all kinds of different logic tends to be interleaved in one monolithic and very complex central logic block.

Using components gives us several improvements on the implementation side. Since they are smaller parts, it is much simpler to make them communicate efficiently with the game simulation while we load them concurrently. Additionally, they split the monolithic game logic into more manageable parts, which played a critical role in allowing a partial roll-out of concurrent entity initialization.

SERIALIZED VARIABLES

Entities inside a game-simulation have a certain state. Some situations, like network communication, require that we store that state in a format that we can transfer and then restore the same state on a different machine. Other situations, like *Squadron 42* save games, require something similar, except that we store the data on disk. In programmer terms, the process of converting a state into a representation which can be stored on disc/be network transferred is called "serialization". Thus, the name Serialized-Variables. This is a concept in which we take the parts of an entity and put it into a special wrapper object. This wrapper object provides ways to serialize the entity state.

By doing it this way we can have game code writing in a uniform style, regardless of how we want to transfer the serialized data later (also important for Server-Meshing, which will be explained later).

MULTITHREADED ENGINE RESOURCE LOADING

Besides game-related resources (data making up components), the engine also supports many shared resources, which are shared by different components and even different Object-Containers. Those resources include objects like textures or meshes. The engine already supports mesh and texture streaming, which is the process of loading and unloading the GPU data for rendering while the game is being played. But we needed to tackle this on a higher level for Object-Container-Streaming.

In the Object-Container-Streaming context, we also must be able to load the object representing the GPU data in a parallel and organized way, so that all in-parallel loaded Object-Containers can still share the same engine resources. This was all work that went on in the background around two and half years ago.

ALL COMING TOGETHER

The groundwork above took a lot of work, and most of it wasn't really visible for the players, as making those changes without a visible effect was our goal (to verify that we changed it correctly). But all this was very necessary preparation work, moving the technology forward and towards the first very visible effects when Network-Bind-Culling was released to the public in Alpha 3.3.

PARALLEL COMPONENT LOADING AND INITIALIZATION

With all the groundwork done we have:

- Levels split into building block (the Object-Container)
- Entities (which make up a large part of the Object-Container) implemented without LUA and as components
- All entity runtime state organized in Serialized-Variables for easier state communication via network
- Multi-threaded creation of engine-side resources (textures, meshes, characters etc.)

The next steps are building on that foundation. Our first goal was to actually load entities in parallel to the game simulation. This first step didn't include any high-level logic of what to load or unload or when yet, so it was a very dumb streaming system. Nevertheless, it already reduced the runtime stuttering (e.g. spawning a ship at runtime no longer required us to run the initialization code of the ship's entities as part of the game simulation).

At that time, we had roughly 300 to 400 different component types.

If we had tried to execute all those in parallel from the start, we would have drowned in bugs. Therefore, we had to develop a system allowing us to incrementally execute more and more component types in parallel to the game simulation. The more component types we made safe for parallel loading, the less the game simulation would stutter.

The solution we have chosen is utilizing Fibers. A Fiber is an execution state where we can control exactly when and where we want to execute it (parallel loading or game simulation). While Fibers can be very tricky to use, they provided exactly the control we needed. With Fibers, we could move the logic for resource loading between concurrent loading threads and game simulation threads, depending if the component type supported parallel loading or not. And with that, it was possible to step-by-step adjust more and more code to run in parallel while ensuring that everyone uses (and thus tests) the already parallelized code. Those changes were partially rolled out with Alpha 3.2, where they reduced the stuttering caused by loading entity resources at runtime, like spawning ships.

PREPARATION FOR NETWORK-BIND-CULLING

Network-Bind-Culling is how we reference to Object-Container-Streaming on the client. In other words, it is the process of deciding what entities to load/unload on any client. We decided to focus on only the client first, as this allowed us to provide several improvements to the players, develop the whole technology more incrementally, and allow us to solve certain problems later (which are discussed in the Server-Object-Container-Streaming section). But even only focusing on the client, we had to tackle a lot of preparation work.

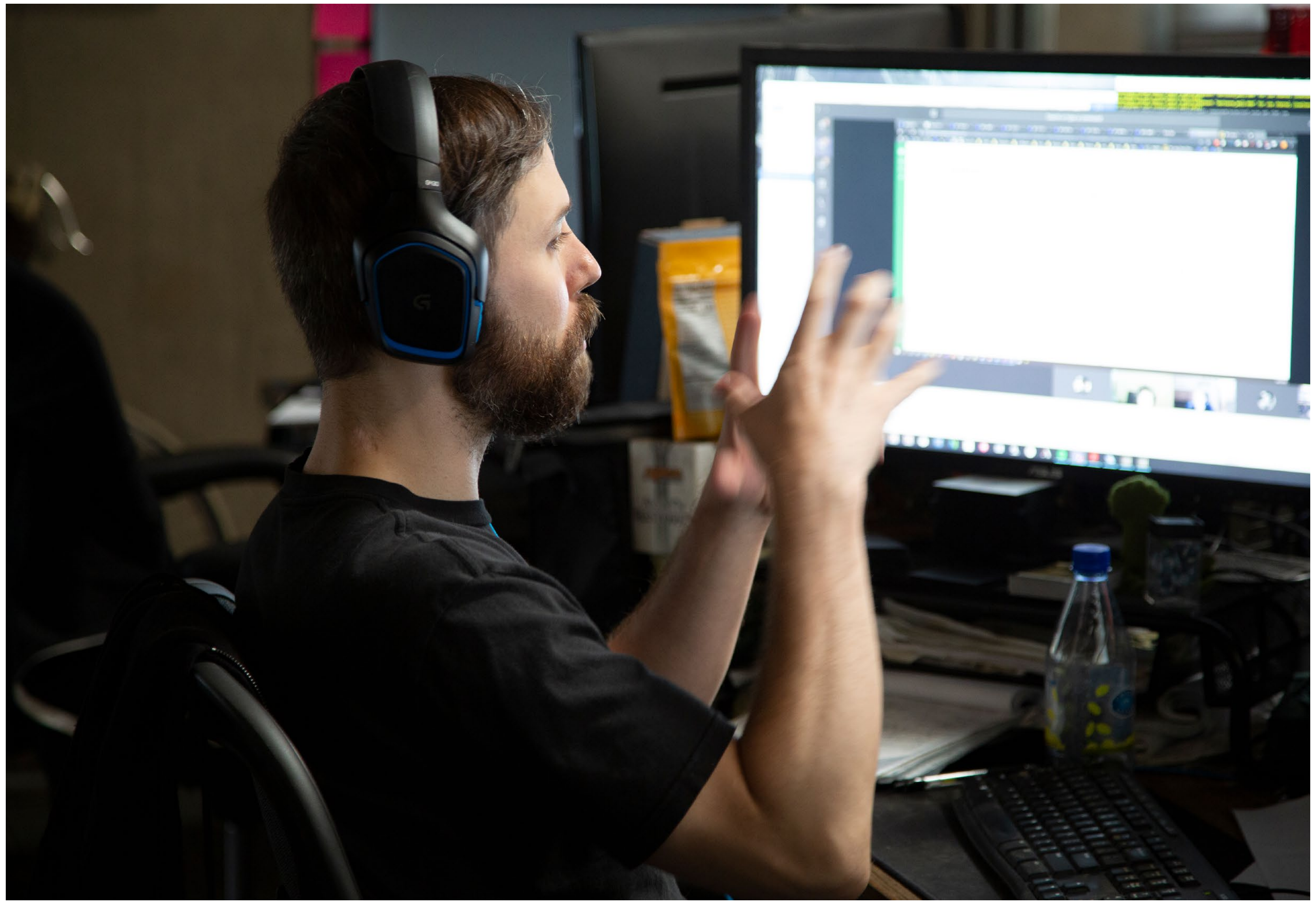
ENTITY-COMPONENT-UPDATE-SCHEDULER

We want to decide which entities to load or unload on a client based on the distance and visibility of that entity in regard to a client. Therefore, we need this information. Luckily, we had already developed such a technology for Alpha 3.0. The Entity-Component-Update-Scheduler is a system designed to control the update of entity components, based on how they are spatially placed relative to the player. By doing this, we can skip updates of components which are too far away (another benefit of the component design, we can do the update policy on a per component basis). It was then natural that the Entity-Component-Update-Scheduler should provide the same information for Network-Bind-Culling.

ENTITY-OWNERSHIP-HIERARCHY/ENTITY-AGGREGATES

Another major issue to tackle was dynamic entity groups. Object-Containers split the static level geometry and objects into building blocks at level design time. But our game also consists out of dynamic entities, like players, ships (which can be built out of multiple Object-Containers), or vending machines - basically, everything that can move around in the virtual world. Additionally, those dynamic entities can combine themselves into groups. For example, a player is picking up an object, or a ship is parked in another ship. And that has implications for streaming. In the ships-in-ship case, we don't want to stream the inner ship before the outer ship, as the inner ship's state is partially defined by the outer ship. Therefore, we have to track those dynamic groups and when they are formed or disbanded. This is a concept we call Entity-Ownership-Hierarchy. In this hierarchy, we keep track of entities that are related to each other. If they are related, we treat them as one group - the so-called Entity-Aggregate.

Building on that, Network-Bind-Culling works on units of Entity-Aggregates, using the information of the Entity-Component-Update-



Scheduler to decide what Entity-Aggregates to load or unload on each client.

ENTITY-SPAWN-BATCHES AND ENTITY-SNAPSHOTS

After tracking Entity-Aggregates correctly, we still had to develop a way to efficiently spawn the large number of entities inside an aggregate efficiently on a client. And we need to ensure that entities spawn in a consistent way on the clients so that we end up with the same Entity-Aggregates and entity hierarchy as on the server. For instance, to make sure that a vehicle spawns with its weapons/thrusters already attached, rather than spawning bit by bit over time. To achieve this, we group entities into Entity-Spawn-Batches, which represent a set of entities that should be spawned together and only made active when all spawned.

For each entity we spawn on a client machine, we send an Entity-Snapshot from the server containing the current state of the entity. This is simply the values of the Serialized-Variables belonging to the entity. Snapshots are also used for serializing entity state to persistence, or for *Squadron 42* save games. Because spawning entities asynchronously on the client takes time, a problem we faced was that by the time the client had completed a spawn batch, the Entity-Snapshot could be out of date, as the state of those entities on the server may have changed while the client was spawning them.

To fix this, the server has to send a second set of Entity-Snapshots once the client has spawned all the entities in a spawn batch. When the client receives these secondary Entity-Snapshots, it can perform minor fix-up to the state of the entities and finally add them to the client's simulation, finalizing the network-driven entity spawn.

SERIALIZED-VARIABLE-CULLING

At the end of developing Alpha 3.0, we had already developed various necessary parts, but not all parts were done, and we introduced planets and many major locations like Levski and Grim HEX. At the same time, we only had the update culling of the Entity-Component-Scheduler. While that system helped with server performance, clients still had to pay an unnecessary cost; we didn't yet have a system in place to decide which client requires which information. On top of this, each component had to run its update when it received new network information, resulting in a measurable performance cost. For example, if at that time Player A was at Levski, and Player B at Olisar, Player B would update all its local versions of the NPCs in Levski due to receiving network updates for them as Player A was near them on the server. But we had all the tools ready to provide the required information. Based on that, we decided to build a system that would only send network updates to clients if that client is in proximity of the updated Entity-Aggregate. As for the earlier example, Player B would no longer update its NPCs in Levski, as the server would know that Player B is nowhere near.

Implementing this, under the name Serialized-Variable-Culling, gave us a noticeable performance improvement on the clients. Additionally, it was the first real-life test of running our client game-simulation with only partial information of the whole universe. We wanted to ship this feature with Alpha 3.0, but in defiance of the heroic effort of the network team, we had to let this feature slip into Alpha 3.1.

ENABLING NETWORK-BIND-CULLING

Several years after the first steps were undertaken and multiple game versions shipped, we could harvest the results of all the work.
So far, we have working:
•Levels split into building blocks (the Object-Container)

- Entities (which make up a large part of Object-Container) implemented without LUA and as components
- All entity runtime state organized in Serialized-Variables for easier state communication via network
- Multithreaded creation of engine-side resources (textures, meshes, characters etc.)
- Multithreaded creation of most component types
- Proximity information between all entities and players provided by the Entity-Component-Update-Scheduler
- The ability to track dynamic entity groups (the Entity-Aggregates)
- Efficient ways to spawn Entity-Aggregates on clients
- First real-world usage of client game-simulation with partial world knowledge via Serialized-Variable-Culling

Utilizing all the preparation above, we could develop Network-Bind-Culling. In that system, instead of skipping network updates while having all entities present on each client (as we did for Serialized-Variable-Culling), we will, driven by the server, load and unload entities on the client. In other words, Network-Bind-Culling changes the rules so that each client only has a view into a much larger virtual game world, while with Serialized-Variable-Culling, each client had the full view but only performed partial updates of its local virtual world.

This gives us several advantages, the most noticeable being performance. As each client has a much smaller data set, each local operation whose cost is affected by entity count becomes faster. It also helps with other runtime cost which could not be culled by Serialized-Variable-Culling. Another benefit is that the client uses less memory, since it only has to keep its view in RAM and no longer the whole universe. For many clients, there was a very large performance improvement when we released Network-Bind-Culling to the public with Alpha 3.3.

But the system has another, even more important advantage: it decouples the client from the universe content. As each client has only loaded the small set of entities required for its local view, the clients are no longer affected by the amount of entities we place in the universe. Client performance is now only affected by the actual client's location and surroundings (e.g. empty space vs crowded city). And this gives us the freedom to place as much content as we want into our virtual world without having to worry about clients. Except that, right now, the server still has everything loaded and has to pay the performance cost. And while having bad server performance doesn't affect the clients frame rate, it causes jerkiness when objects move (as it is very likely that client and server disagree with the entities position). Which is also a serious problem, but something we want to tackle with Server-Object-Container-Streaming.

BUILDING SERVER-OBJECT-CONTAINER-STREAMING

With Network-Bind-Culling implemented, the focus shifted over to implementing Object-Container-Streaming on the server.

The basic concept is that if no player is near an object, we can "freeze" that object's state. And instead of keeping the frozen entity in memory (incurring a cost), we can serialize (using Serialized-Variables) its state and store the serialized state in a database. While a client is moving through the virtual world, the server updates its view into the database to restore entities now in proximity as well as free and store away no-longer-needed entities.

Here, Server-Object-Container-Streaming and Network-Bind-Culling go hand in hand. The database contains the whole universe in a frozen state. The server has only a small subset of the universe loaded; in other words, it has a view into the database. Then the client also only has a subset of all server-loaded entities, having a view into the server's virtual world. By this model, we keep everything far away from the players in a frozen state so that those entities don't affect performance or memory. There are also some exceptions to this model, like the Subsumption Universe Simulation, but those are out of scope for this article.

When Server-Object-Container-Streaming is done, we will have a technological solution for content scaling on the server. This means that we can place way more content into the virtual world, while the server performance is only affected by the areas where all players are active (which is a much smaller set than the whole universe). But Server-Object-Container-Streaming also comes with several additional problems, all of which the involved teams are working on in order to deliver it as soon as possible.

DEFINITE STATE

With Network-Bind-Culling, we always had an authoritative version of each entity loaded on the server. This allowed us some "lazy" solutions, since we could get away with smaller issues, since we would always correct them after the entity is loaded on the client.

One example of this is teleporting the player. A teleport is an instant move from one place to another in the universe. This is the worst case for streaming, but we have it in some situations, like player respawn, or when using development tools. After a teleport, everything around the player must be loaded. We didn't have any priority for the order in which we spawn those entities. This resulted in NPCs falling through the not-yet-loaded floor. With Network-Bind-Culling this was fine, as we could depend on the server sending us the correct NPC position (as the floor exists on the server). With Server-Object-Container-Streaming we cannot do that. As the server is authoritative, if the NPC is spawned before the floor, the NPC will be gone, resulting in boring empty cities. Therefore, we had to ensure that we always spawn the floor before the NPCs. Another issue is component types we only execute on the server. Previously, we didn't unload them, so have to make sure they restore their state correctly from serialized data.

Those, and all other kinds of small problems are the things we have to fix before we can ship Server-Object-Container-Streaming.

ENTITY-STREAMING-MANAGER / STARHASH / STARHASH-RADIXTREE

Another problem arises when we unload all entities and store them in a database. We need a way to perform spatial searches on those entities to ensure we only load those in proximity to any client. Therefore, it was necessary to develop a lookup scheme that allows us to store a huge number of entities with enough spatial information. For this, we adapted the Geohash algorithm (used by all map applications to find places around the users) for our needs by making it larger (our virtual world at 2m solution needs more data than the real earth) and 3D. We called it a StarHash.

This StarHash provides us with an efficient tool to store our entities in a way allowing efficient searches for all entities in an area of space by utilizing a data structure called a RadixTree. The Entity-Streaming-Manager is then the logic-driving the StarHash-RadixTree queries to trigger loading and unloading of entities on the server, based on the positions of all connected clients.

LOCATION-IDS

The last major issue we had to tackle was spawning locations. To spawn a player, the game logic requires a SpawnPoint, which is also an entity. But we only load entities if a player is nearby, thus we need to spawn a player to load the SpawnPoint to spawn the player. Since it is also not possible to exclude SpawnPoints from streaming (as they are part of other larger constructs like space stations) we had to find another solution.

Here we decided on a two-phase spawn process. When a player connects, we first find their spawn Location-ID. A location is a higher-level concept of a point in space. So we first load all entities at this point, which will also load the required SpawnPoint. Afterwards we can safely spawn the player at their destinated SpawnPoint. Lastly, the streaming logic will switch over to the player from the Location-ID so that the database view of that player will move with them.

CURRENTLY ONGOING WORK

At the time of writing this article, not all work for Server-Object-Container-Streaming is finished. We have implemented the Entity-Streaming-Manager as well as the StarHash logic. The Location-ID work is nearly done and should be finished soon. Because of that, Server-Object-Container-Streaming can already be used to a certain degree. And doing that shows us all the problems we have with missing Definite State and all the areas we still have to fix. Most major areas where we have such problems are known and are actively being worked on.

NEXT STEPS

The work won't be over when the first iteration of Server-Object-Container-Streaming is delivered. While the first release should give us way better content scaling on the server, we will still have several areas to work in.

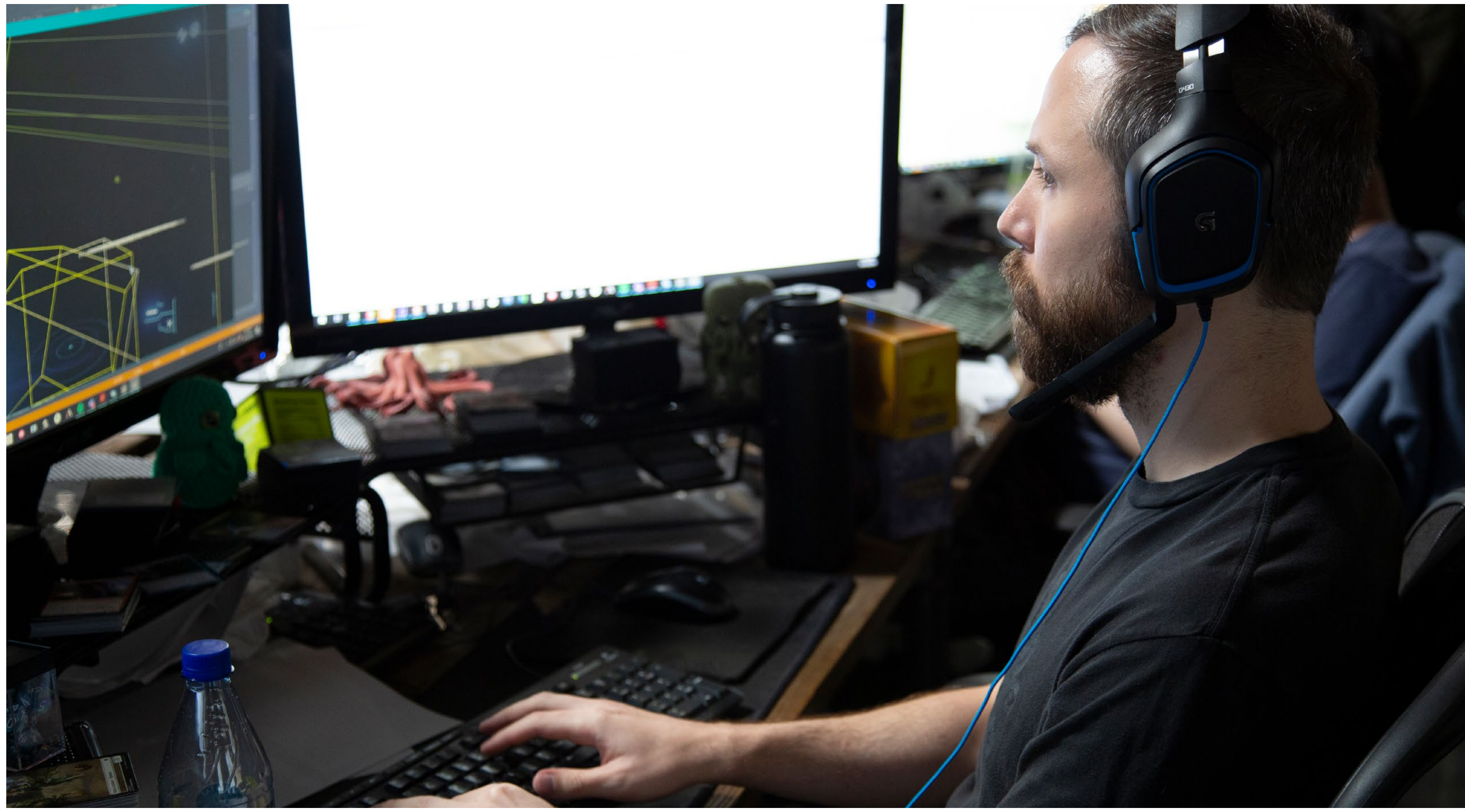
CROSS SESSION PERSISTENCE

Server-Object-Container-Streaming doesn't affect how and what kind of data we serialize to persistence. We will store the entity in a frozen state in an in-process database. This implies that the state is lost when the server crashes or is restarted (besides the state we already persist). So, the next steps are to develop an efficient network access layer to allow the storing of the entity in a database on a different machine. When we implement that step, object state will persist over server restarts and crashes (until we delete the persistence database), moving the whole game towards a persistent experience.

SERVER-MESHING

With Server-Object-Container-Streaming, a single server is responsible for managing the database views of all clients. Thus, while the Server-Object-Container-Streaming should improve the server performance (as we load less entities), it ultimately won't solve the problem of more players per server.

This is where Server-Meshing comes in. In this concept, instead of having a single server manage all the views, we will distribute the individual views over multiple servers. Doing this will reduce the load on each participating server. When we then place those servers on different machines, we get a nice and practical way to scale with player



count. To implement Server-Meshing, we will build on what we are building right now: entities will be moved between servers by using the serialization code provided by Serialized-Variables, depending on the code from Server-Object-Container-Streaming, to ensure that we can restore those moved entities correctly on a different server.

EDITOR SUPPORT

More hidden from the public but very important is the game editor. The editor is a custom engine tool which is used to build our Object-Containers and place them in the universe. It is also used to test-play the newly developed content while working on it, which is extremely important to develop good quality content. Unfortunately, the editor itself is not yet streaming-aware. Thus, the content creator can create and develop content, but suffer from long loading times and bad frame rate. And this will become worse the more content we place into the game.

Therefore, a very important next step is to make the editor streaming-aware to give the content creators the same benefits we gave the clients (via Network-Bind-Culling) and server (via Server-Object-Container-Streaming). But as the editor is having its own additional logic on top of the whole game-simulation, we can only tackle that after doing Server-Object-Container-Streaming.

SQUADRON 42 SUPPORT

Squadron 42 will be the easiest additional work. In *Squadron 42*, the client will be the server as well. Therefore, we will execute the same code as we do on the *Star Citizen* server. In fact, we do that already internally. And as *Squadron 42* and *Star Citizen* share the same code base, fixes for Server-Object-Container-Streaming for either product will benefit the other.

CLOSING WORDS

I hope this introduction provided a helpful explanation of the multi-year-long voyage of "Object-Container-Streaming" and an understandable explanation of all the technical challenges we had to face and overcome during this journey.

Please also forgive me for omitting most of the very technical nitty-gritty details, but laying out all those would turn this article into a book. And I think it is better to write the technology than writing a book about the technology we want to build.

Thank you for taking the time to read this.

Best Regards,

Christopher Bolte
Principle Engine Programmer, Cloud Imperium Games

WHITLEY'S GUIDE

The following extract is from the 2949 Whitley's Guide to Spacecraft's Crusader Hercules Development History. Reprinted with permission. Whitley Guide is the property of Gallivan Publishing, 2860-2949, all rights reserved.

CRUSADER INDUSTRIES

THE CRUSADER INDUSTRIES HERCULES STARLIFTER

HERCULES STARLIFTER - DEVELOPMENT HISTORY

Development of the spacecraft that would become the modern Hercules began in the mid-28th century during a particularly introspective period for UEE military leadership. Keen to examine the potential lessons of the last war, UEE commanders undertook an unprecedented analysis of the Second Tevarin War followed by a series of simulated wargames covering major battles. One of the outcomes of this effort was a new understanding of the impact of support logistics on interstellar warfare. During the Tevarin wars and prior, interplanetary operations meant establishing an initial beachhead on a hostile world using small, heavily armored landing assault craft. Once a base was established, heavier equipment would be brought in using a support column of freighters and transporters not specially equipped for combat. Analysis of this practice in action suggested it had created a major choke point that had significantly delayed necessary assets in several cases. Not only did transporting weaponry crated aboard traditional transports slow the ability to deploy heavier artillery, missile launchers, and armored tanks, it also limited their immediate range once deployed. Even successes like the famed 2605 Battle of Koren Pass were cited as examples of situations where casualties resulted from a lack of logistics: if the UEE had the lift capacity to deliver fighting vehicles directly from orbit, losses on the ground could have been significantly reduced.

The solution, military leaders determined, was two-fold. The first was organizational. In an attempt to reduce time lost to inter-service confusion, the decision was made to establish UEE Starlift Command - a cross-service support framework intended to better coordinate the UEEN assets responsible for delivering personnel and materiel that would address the UEEA and UEEM's granular battlefield needs. The second was to set forth the specifications for a complete quantum-to-battlefield support spacecraft that could deploy armored units and other assets to a variety of alien terrains while under fire. Instead of amphibious operations focusing on establishing individual fire bases to bring in heavier assault weaponry, this command and its theoretical spacecraft could deliver advanced units directly to active theaters. The plan would prove incredibly effective and significantly alter the shape of planetary-scale battlefields. Additionally, this new spacecraft could be maintained locally and be used to quickly relocate already deployed assets should flashpoints evolve.

The formal request for a proposal was issued in 2814. It asked for a large, well-protected transport that was jump-capable, able to sustain concentrated artillery fire, and able to deploy multiple armored vehicles quickly. Significant proposals were developed by both Aegis Dynamics and Crusader Industries. Crusader, then a premiere manufacturer of civilian starliners and associated industrial conversions, was expected to adapt their serving Saturn-class starliner for combat operations. Aegis was expected to develop a bespoke design specific to the UEEN's needs. In an unexpected twist, the opposite proved true: Aegis suggested adapting existing military freighters with armor and defensive turrets, while Crusader developed a much more expensive proposal to create an entirely new design that would eventually become the Hercules starlifter. Despite Crusader's proposal having three times the price tag of the Aegis conversion, the feeling was that such a major reorganization of tactical doctrine would be better supported with an entirely new spacecraft. The military decided to invest, despite the cost of developing such a system and the inevitable organizational issues that would come with its adoption. With that, Crusader Industries launched an all-out four-year program to develop their first dedicated military support spacecraft.

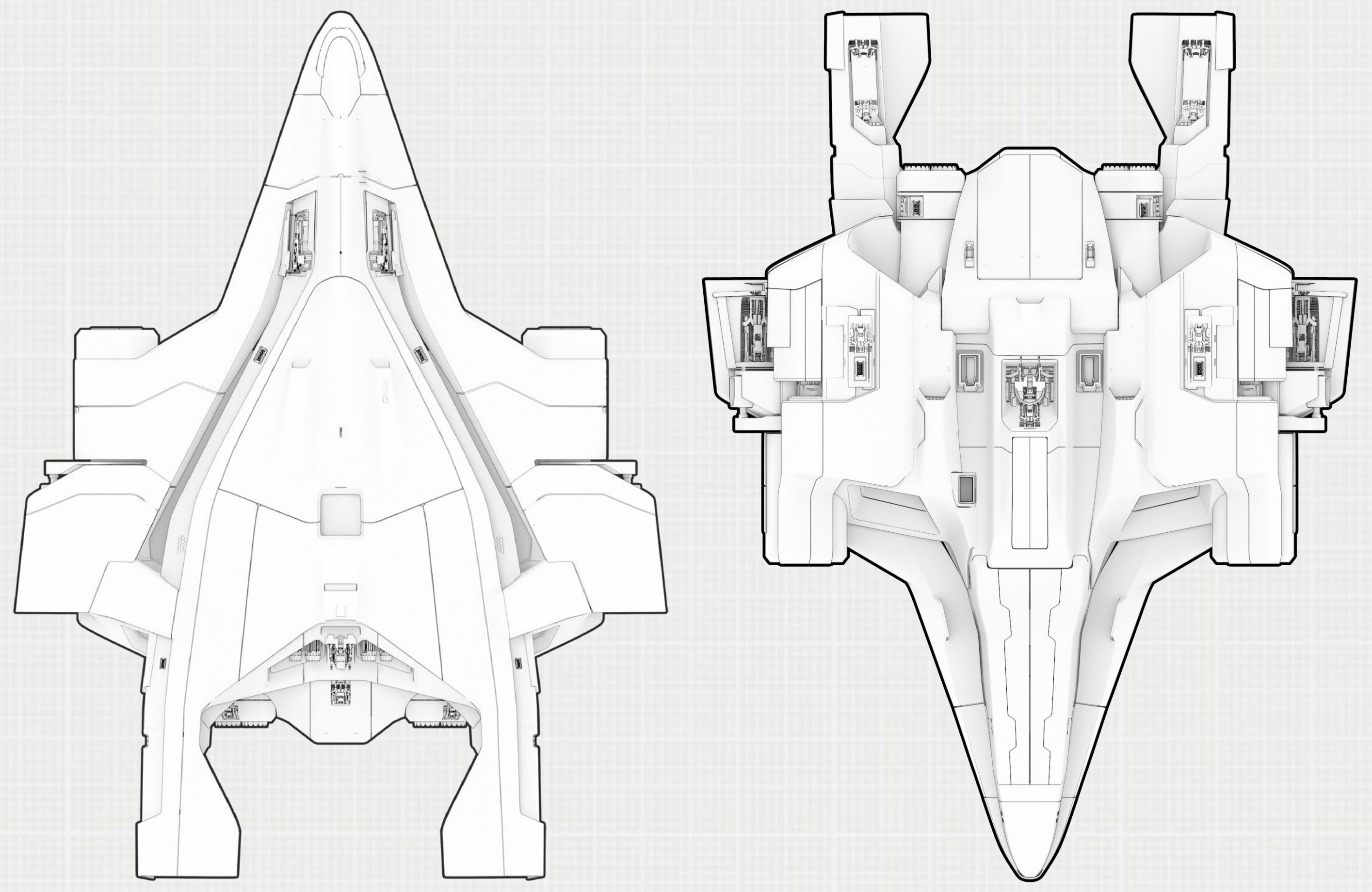
The first active-duty starlifter unit was formed in May 2821 with a dozen first model spacecraft (formally designated the 'M2 Hercules'). In initial training exercises, the new ship worked perfectly. Capable of taking sustained fire and deploying a tank or armored car in minutes, the Hercules met the military's requirements and then some. However, delays to Hercules deployment occurred due to the difficulty of integrating the new interservice command, with those involved facing a great deal of bureaucracy in order to allow these new processes to supplant the tried-and-true support chain. Nevertheless, the wisdom of the decision became clear in March 2824 with the first active combat deployment of the Hercules system, when UEE armed forces were called upon to put down a heavily armed group of pirate forces located on a frontier world near the Xi'an border. Instead of attacking the site from orbit, planners determined that it would be worthwhile to capture assets intact in order to pursue further antipiracy operations elsewhere. Two Hercules squadrons, escorted by deep space support fighters, quietly deployed troops and an armored column which defeated the stunned criminal forces in short order. The battle, previously thought to be a particularly hazardous prospect, was won with no losses of UEE personnel and the resulting capture of information would lead directly to the destruction of two pirate outposts and a small capital ship.

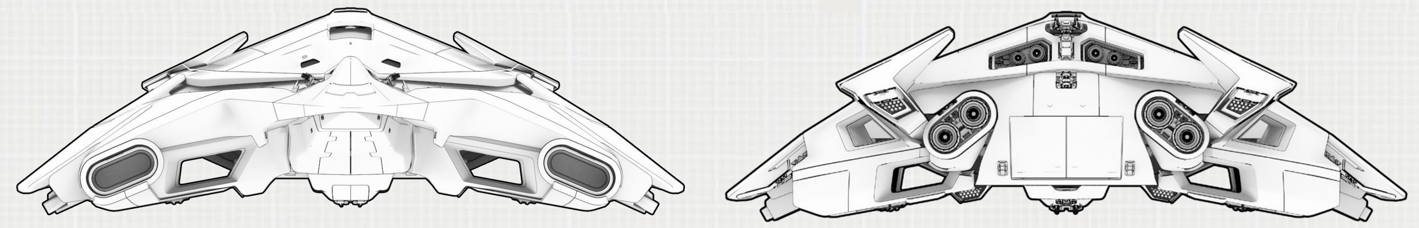
As use of the Crusader starlifter normalized, it quickly became a favorite among pirates and ground crews. Crusader's experience with civil space transport meant they understood how to build a spacecraft intended for ease of maintenance. Additionally, the hulky, armored appearance of the Hercules became a comfort to soldiers and marines, who came to associate it with much safer deployments. The sight of a Hercules on the battlefield inevitably meant the delivery of additional supplies or reinforcements. Within two decades, Starlift Command had organizational structures in place across the empire that would allow the rapid movement of Hercules to any battlefield within a jump of a currently settled star system. Several units of starlifters are kept on

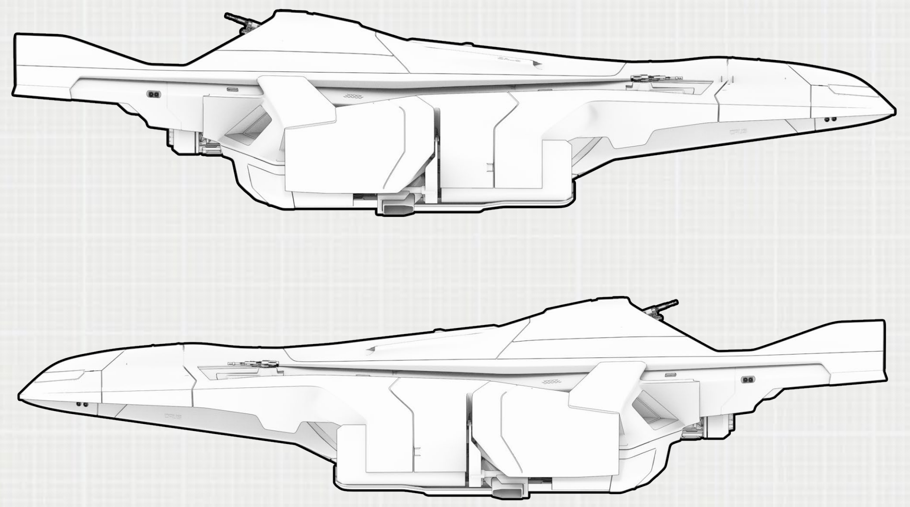
'ready five' status around the Empire already loaded with tanks and missile launchers and teamed with special operations troops that can be used to address rapidly developing situations.

Over the decades, Crusader has continued to update and enhance the original Hercules design and has made a tidy profit performing fleet enhancements and producing battlefield update kits in the progress. This steady dedication to modernizing the fleet has been strongly supported by Starlift Command and has allowed individual examples to remain in service well past their intended retirement. As of 2948, a significant number of first and second generation Hercules hulls were still being operated thanks to these extensive maintenance processes. Similarly, Crusader has continued to apply their 'frame-and-role' design process developed in starliner construction to the Hercules line, which has allowed the rapid creation of a number of role-specific variants including refuelers, heavy armor support ships, and information runners. Crusader's philosophy allows the creation of variants to proceed rapidly as the need requires without disrupting existing production lines. This has allowed role-specific Hercules to be constructed as needed and retired just as quickly. One of these variants has become a significant part of the UEEN inventory: the A2 is a dedicated heavy gunship that adapts the Hercules' heavy armor and other defensive systems for more a sustained combat role and uses the design's extensive cargo capacity for munitions storage. The A2 Hercules is now constructed on its own factory line and has seen extensive combat operations against planetside forces.

In 2940, Crusader surprised the aerospace industry by announcing the development of the first standalone civilian variant of the Hercules, the C2. Long seen as a military-only spacecraft design, the decision was especially unexpected as Crusader's factories did not have the immediate capacity to produce more than the Hercules already requisitioned by the military. In order to produce the C2, three more Hercules lines would need to be opened. Crusader, however, saw this as less of a gamble, believing that even if interest in a civilianized Hercules was not immediately apparent, the investment would ultimately be useful as military demand increased in the face of increased conflict with the Vanduul. The C2 Hercules design drops some of the armor and specialized hardware from the current-generation military type in exchange for an overall improvement in cargo. Formally targeted at frontier concerns, the C2 variant has been positioned as a way for planets with less developed infrastructures to rapidly move vehicles from place to place. In their example study, Crusader imagined a mining corporation seeking to reallocate heavy equipment to sites around a newly explored planet in order to make use of claims before unlicensed jumpers could move in. The move proved to be a success, with civilian organizations quickly taking to the sturdy spacecraft design and corporate partners happy to have a ship with such a well-developed lineage and extant support apparatus. In addition to miners and explorers, the C2 Hercules quickly proved to be popular among militia groups eager to move small spacecraft and ground vehicles from place to place on individual worlds.







HERCULES STARLIFTER

MANUFACTURER	CRUSADER INDUSTRIES
MAXIMUM CREW	C2 2 M2 3 A2 8 (INCLUDING 5 STATIONS INTHE REAR)
MASS	114,591KG
LENGTH	94M
HEIGHT	23M
WIDTH	70M
ROLE	C2 TRANSPORT M2 MILITARY A2 GUNSHIP

OBSERVIST LIFESTYLE: G-LOC BAR

Greetings, traveler. The universe is full of unique stories waiting to be told. We here at the OBSERVIST LIFESTYLE are eager to provide a firsthand, up-close look at the fascinating people who live among the stars and the amazing adventures they have.

Today, we travel to the Stanton system and enter atmosphere above ArcCorp. Below us, urban sprawl stretches in each direction, making it an intimidating and overwhelming destination for many travelers. Most visitors, whether for business or pleasure, set a course for Area18, a public commercial district with its spaceport open to all. This bustling commerce hub draws many to this unique corner of the Empire.

The G-Loc Bar sits just off Area18's central plaza. Popular with both locals and off-worlders for its spectacular views and stiff drinks, the bar sees a steady flow of customers from all walks of life and from almost every system in the UEE. Bartender Brant Weiss has been slinging drinks and swapping stories with G-Loc customers for over eight years. He spends an average of twelve hours a day, six days a week at the bar, but claims he almost never gets bored. Weiss credits the diverse and unpredictable customer base for keeping him engaged and interested in his job. I was eager to hear what life was like for someone who deals with such a wide range of Humanity on a daily basis.

Arriving during a mid-shift lull to find G-Loc lively but not too crowded, I grab an empty stool at the far end of the L-shaped bar and survey the scene. The bar blends an industrial yet elegant aesthetic with retired and refurbished ArcCorp thrusters hanging above comfortable faux leather booths. An elevated area at the back contains a currently empty dance floor and massive windows offering an impressive view of Area18's unending sprawl.

Today, Weiss works alone behind the bar. He greets each customer warmly before taking their order. Some he knows, but most are new. Then he quickly mixes their drink and moves to the next guest. Once everyone's been served, Weiss retrieves a few used glasses left on the bar, runs the sanitizer, restocks the rails, and makes himself a quick drink before joining me.

"It's just a peach Fizzz. I need to keep my wits about me. Plus, knowing my tastes, I'd drink away half my pay if it became a habit. There's a bottle of Radegast on the top shelf that keeps calling to me. But in all seriousness, this is a delicate and detail-oriented job. Most don't think of it like that, but once the rush is on, trying to remember who ordered what and getting it all out fast can be a real herculean effort. Plus, you gotta make sure you stay fully stocked, keep everything not just clean but sanitary… I mean, the list just goes on and on. That's the easy part though. It's managing the customers that's the true challenge and, for me at least, the real joy of this job."

Weiss claims the G-Loc doesn't have a typical customer. Its location near Area18's busy central plaza brings in a wide range of foot traffic. It attracts locals looking to unwind after work, haulers killing time between gigs, and tourists drawn in by positive reviews of their expertly crafted and potent cocktails.

"We always do good business during the Murray Cup or sataball season. Those crowds are always fun but a bit rowdy. There's one long hauler that makes us her first stop after spending who knows how long trapped alone aboard her ship. Says she needs a heavy dose of Humanity with her whiskey. And the other day, I delivered a round of shots to a quiet table in the back only to find out that they just hammered out the final

details on some multi-billion credit merger. No two days are ever the same around here, that's for sure."

As Weiss describes his wide-ranging clientele, an individual wearing full armor and a helmet sprints into the bar. They quickly survey the scene before running up the steps to the dance floor. They do a brief hip shaking, finger wagging dance before running out almost as quickly as they came in. The incident doesn't faze Weiss at all.

"That happens more often than you think. Don't know what drives folks to do it. Sometimes I think staying on a ship for too long can mess with people's heads… or it's just the drugs. (Laughs) Anyways, I've gotten pretty damn good at reading people with a glance. Even those that come tearing through here in full armor. I mean, that right there tells you a lot about the person. Who wears a helmet to go get a drink?"

A steady stream of customers continues to trickle in and out. While Weiss fills drink orders, he chats amicably across a wide range of topics from the TDD's current price for astatine to the best beaches on Goss. Based on his grounded repartee and in-depth knowledge about areas and issues affecting all corners of the UEE, most would be shocked to know he's never left the Stanton system.

"I went to microTech once. Those biomes are beautiful but everything's so damn expensive. My family never had a ship, and I sure as hell can't afford one. Never let that kill my curiosity though. Someone once told me that you can learn a lot about life by traveling, reading, and engaging in good conversation. I don't have the means to travel, so I focus on the other two. That's why I'm always talking to folks about their adventures and keeping a list of all the interesting places they mention. Maybe one day when I'm retired I'll visit a few. I'd like that. But for now, I'm happy to have the rest of the universe come here and visit me."

And from where I'm sitting, it seems the rest of the 'verse is happy to do just that. As long as Weiss remains content mixing strong drinks and engaging in spirited discussions about the wider universe with those that wander into the G-Loc, there seems to be no end to the flow of potential customers looking for a respite from the non-stop metropolis outside. Getting ready to order a second round, I ask about his favorite drink on the menu. His answer is immediate and emphatic.

"Nick's Mistake. It's delicious, packs a punch, and something you can only get here. It uses our secret Nova mix, which we make in-house. Seriously, if I told you the recipe, I'd be fired faster than a projectile from a tachyon cannon."

For those that want the G-Loc experience without traveling to Area18, Weiss happily shared the recipe for his second favorite drink on the menu, the Wingman's Hangover, with one caveat.

"The name says it all. So, let me give 'em the spiel I reel off here. Enjoy responsibly and never, ever fly while intoxicated."

WINGMAN'S HANGOVER (SHAKEN, ROCKS)

1oz	Jynx gin
1oz	Starlight Idris Cuvee cognac
1oz	Lionheart Martian whiskey
.75oz	Lime
.5oz	Simple syrup
	Rothman's Ginger Lime

Add all ingredients besides Rothman's soda to shaker, shake with ice. Strain into a Collins glass filled with ice. Top with Rothman's. Garnish with lemon wedge.

DISCOVER YOUR NEW VISION

Whether exploring uncharted territory or going out on the town, there's a style for every situation with the Element Authority 2949 eyewear collections.

URBAN COLLECTION

ADVENTURER COLLECTION

FEATURED BRANDS

SPV RAMBLER STEGMAN'S

Supplies and availability of 2949 Eyeware Collections may vary by Element Authority store location. Contact your local Element Authority store for more information.