Down To Earth — for dummies

How we made NVIDIA's photo-to-3D-world magic a little better, explained without computer-talk

> *This is the same project that's described over in the main README and the technical proposal — but written for a reader who doesn't know what a "GPU" is, has never heard of a "diffusion model," and doesn't care about megabytes. If you can follow a recipe and you've ever drawn a map of your house from memory, you're qualified.*
>
> No prior knowledge needed. We'll explain everything as we go. The whole thing is one big metaphor about a **photo album** and a **floor-plan map**. If you get that, you get the whole project.

---

1. Hello — and what's all this fuss about?

A team of researchers at a big company called **NVIDIA** (you've probably seen their logo on a green-and-black box, they make the chips that play your video games) released a new piece of AI in April 2026 called **Lyra 2**.

Lyra 2 does something kind of magical: **you give it ONE photograph, and it gives you back a 3D world you can walk through.**

Take a picture of a country lane, hand it to Lyra 2, and it figures out *what's around the corner* — the parts of the lane that weren't in your photo. The trees behind you. The barn you can't see in the original shot. It invents the whole 90-metre stretch of countryside around that one snapshot.

You can then put on a virtual-reality headset and *walk* through that imaginary lane. The 3D world feels real. The bushes have depth. You can crouch and look under the fence. None of it was in the original photo — but it's all consistent, like a memory of a place you've been.

That's the magic trick. Lyra 2 hallucinates a whole environment from a single photo.

The catch is that **it takes a tiny supercomputer to run.** Not a normal computer — a special one with what's called a "GB200" chip inside. Each one of those chips costs about $70,000. And NVIDIA used **64 of them** for two months to teach Lyra 2 how to do its trick. So the cost of *making* Lyra 2 was somewhere around $4.5 million worth of computers.

Once made, *using* Lyra 2 still needs one of those expensive chips. You can't run it on your laptop, your phone, or even a really good gaming PC.

That part — the "you need a $70,000 chip to use it" part — is what we set out to think about.

---

2. The magic trick, broken down

Before we talk about what we improved, let's understand how Lyra 2 actually works. It's not really magic — it's a series of clever steps stacked on top of each other.

Step 1 — Take a photo

You give it any photo. A field. A street corner. A bedroom. Doesn't matter.

Step 2 — "Imagine walking forward a tiny bit"

The AI imagines what the view would look like if you'd taken a step toward the photo. What would you see now? The corner of the building gets bigger. New things appear at the edges. Maybe a wall comes into view that wasn't visible before.

Step 3 — Remember what you've seen

After imagining that next view, the AI **writes down what it saw and where it saw it**, like a tourist taking notes in a guidebook. "*Here's a brick wall. Here's a fountain. Here's the sky.*"

Step 4 — Imagine walking another tiny bit

The AI imagines the next view. But this time, when it imagines, it **looks at its notes** from before, so the brick wall stays a brick wall in the new view. It doesn't suddenly become a different colour. The fountain doesn't move.

Step 5 — Repeat forever

It keeps doing this — taking small imaginary steps, writing notes, looking at notes — until it's built up a whole 90-metre world around the original photo.

Step 6 — Convert the notes into something walkable

At the end, the AI takes all its notes and turns them into a 3D world made of millions of fuzzy glowing dots called "Gaussian splats." Together, those dots form solid-looking trees and walls and floors. You can walk through it.

That's it. **Walk a step, take notes, walk a step, take notes, build the final world.** The whole thing happens in about 30 seconds on the supercomputer.

---

3. The notes problem

That little step where the AI "takes notes" — we're going to focus on that.

Because here's the thing: **how the AI organizes its notes turns out to matter a lot.**

The original Lyra 2 organizes its notes like a **photo album**. Imagine you're a tourist on a long walking tour. Every time you turn a corner, you take a photo, and you stick that photo into an album with a date label: "*Photo 47, from Tuesday at 3pm, taken from the corner of Main Street.*"

When the AI imagines the next view, it has to flip through the album asking "*have I seen this part of town before?*" If you walked past the same fountain three times from three different angles, the fountain appears in **three different photos** — Photo 12, Photo 28, Photo 41. The AI doesn't *know* those are all the same fountain; it has to figure that out by looking at each photo and comparing.

This works! Lyra 2 does it really well, in fact. It's the way it was trained. But it has three quiet problems:

Problem A — The album gets fat

The more you walk, the more photos you take, the bigger the album gets. After a 90-metre walk you've got hundreds of photos. The AI's "memory" (the special fast memory inside the supercomputer, called VRAM) starts to fill up.

Problem B — Finding things in the album is slow

When the AI wants to ask "*what's at this exact spot?*", it has to **flip through every photo** asking "*did this one see that spot?*" The more photos, the longer it takes. By the end of a long walk, this rummaging step takes more time than the actual imagining.

Problem C — The AI has to learn that the same place looks different in different photos

The fountain looks different in Photo 12 (close-up, sunny) than in Photo 28 (far away, in shadow) than in Photo 41 (from a weird angle). The AI has to *learn*, during training, that all three are the same fountain. That's extra work for the AI to figure out — work that costs training time and isn't perfect even after the training is done.

---

4. Our idea: throw away the photo album, draw a floor plan instead

Here's what we suggested.

Instead of keeping a fat album of photos, **the AI should draw a single floor-plan map** of the place it's exploring. Like the map at a shopping mall: a flat overhead drawing where every spot in the world has *one* address on the map.

When the AI sees a fountain, it marks the fountain's spot on the map: "*there's a fountain at position 47-east, 12-north, 3-up.*" If it sees the same fountain again from a different angle, it doesn't add a new entry — it just refreshes the existing one. "*Yep, still a fountain at 47-east, 12-north.*"

Now when the AI wants to ask "*what's at this spot?*", it doesn't flip through an album — it **glances at the map**. One look. Done.

The fountain has **one address forever**, no matter how many times the AI walked past it. The "I've seen this before" question doesn't need to be learned — it's structural. The map *is* the memory.

That's the whole idea. One paragraph. **Switch from photo album to floor-plan map.**

The clever bit

Here's where it gets a little clever, but stay with me.

When the AI draws on the map, it uses **colours**. A spot might be marked green for grass, brown for a building, etc. — that's the colour that means "what's here."

Our trick is: **the colour you see on the map also IS the address of that spot.**

Imagine a wall map where the top-left corner is painted "1-1" red and the bottom-right corner is painted "100-100" blue, with smooth gradients between. Every shade of every colour on the map IS its own GPS coordinate. The colour and the location are the same number, viewed two different ways.

This sounds weird but it's actually a really practical thing. It means:

- You can **read a coordinate** by looking at a colour
- You can **find a coordinate** by looking up that colour on the map
- The two ways are the same trip, in either direction
- **No translation step needed**

For people who like analogies: it's like if your house address — "742 Evergreen Terrace" — was also literally written on the front of your house in giant letters AND also the exact angle of the sun reflecting off your mailbox at noon. You could find the address THREE different ways and they'd always be the same answer. Redundant in a useful way.

We call this property "**the bytes are the coords**." (A "byte" is a small number, between 0 and 255, that computers use to represent things like colours. So if the colour `red=42, green=87, blue=200` on the map means "this point in the world is at position 42, 87, 200" — the bytes ARE the coords.)

---

5. Why our way is better — five plain answers

Here's what switching from photo album to floor-plan map buys the AI.

One — No more bloat from looking at the same thing twice

The AI can walk in circles around a courtyard, seeing the same fountain fifty times, and the **memory cost stays exactly the same** after the first observation. With the old album, every glimpse added another photo. With the new map, every glimpse just refreshes the existing entry.

For long walks (which is what Lyra 2's whole job is — long, exploratory walks), this is a big deal.

Two — Finding things is one step, not fifty

"*What's at the fountain spot?*" — old way: flip through the album. New way: glance at the map. **One operation instead of many.**

The bigger the cache gets, the bigger the win. After a 90-metre walk with the old way, the *finding* step takes more time than the *imagining* step. With the new way, finding is essentially free.

Three — The same place always has the same name

The fountain at "47-east, 12-north" is always the fountain at "47-east, 12-north," whether you're approaching from the south, the north, or from above. The AI never has to learn that three

different photos show the same thing — it's **structural**, not learned.

This isn't just an efficiency win. It also means **the AI is less likely to make mistakes** like accidentally drawing two slightly-different fountains close together because it didn't realise they were the same fountain in different photos.

Four — A human can read the map

Photo albums are kind of opaque to humans. You'd need to look at every photo to know what's in the album. But our map is just a 2D picture where every spot's colour is its address. **You can print the map and read it.**

For researchers debugging the AI, this matters. They can literally look at the map and see what the AI's "memory" contains, without running anything.

Five — Computers love it

This is the most technical of the five, but the simple version is: computers have a fast way of looking at colours that are next to each other on a screen. Our map is arranged so that **things that are near each other in the real world are also near each other on the map.** Which means the computer can look at a chunk of the map all at once, which is much faster than looking at scattered pieces.

The old album doesn't have this property. Photo 12 and Photo 28 might both contain the fountain, but they're stored at totally different places in the computer's memory.

---

6. The three things that are NOT better (the honest catches)

We promised we'd be honest. Here are the catches.

Catch one — The map has a grid

Real life is smooth, but our map is divided into little squares (grids), like graph paper. Each square represents a small chunk of the real world.

How small? Depends on which "version" we pick:

- **Small map**: each square is 1 centimetre. The whole map can hold about a **2.5-metre cube** of world. About one room.

- **Medium map**: each square is still 1 centimetre. The whole map can hold a **655-metre stretch** — a big village.
- **Large map**: each square is 1 centimetre. The whole map can hold a **167-kilometre area** — a small country.
- **Massive map**: each square is 1 centimetre. The map can hold **the entire Earth's surface**, and still have plenty of room left over.

So the "grid" problem isn't really a problem — we just pick a finer map for bigger worlds.

But here's the catch: at the smallest map (1 centimetre squares), things smaller than 1 centimetre disappear. A spider's leg, a pen tip, a strand of hair — these become invisible to the map. For most things we'd want to do with Lyra 2, this doesn't matter. But it's a real limit.

Catch two — A single map only goes so far

A *single* sheet of map paper can only hold so much detail before it gets too big to read. For Lyra 2's longest walks (90 metres or so), the "medium map" works fine. For longer walks (a kilometre, a city, a country), you need **folders of maps** — one map for each neighbourhood, with rules for which neighbourhood you're currently in.

We sketched how this would work but didn't actually build it. NVIDIA would need to build it if they wanted to use this idea for, say, a "walk through downtown Manhattan" experience.

Catch three — The AI has to learn a new language

Lyra 2 today has been **trained for hundreds of hours on dozens of supercomputers** to read photo albums. To read maps instead, it needs to be re-taught.

Re-teaching isn't catastrophic — it's not as expensive as starting from scratch. But it's not free either. NVIDIA would need to commit some training time to it.

This is the biggest "catch" of all. Our idea sounds good on paper, but it doesn't actually do anything for anyone until someone with a few weeks of supercomputer time decides to re-teach Lyra 2 to read maps. And the only people who can do that easily are NVIDIA themselves.

So our proposal is basically a polite "*if you ever do a refresh of Lyra 2's training, please consider this idea*" — not a "*here's a thing that works, run it tomorrow.*"

---

7. Tiny, small, big, gigantic worlds — the size question

We mentioned that different sizes of map exist. Let's slow down and look at the size question, because it matters.

A "map" in our scheme is essentially a single image. The image's pixels store the world's information. The bigger the world we want to represent, the bigger or denser the image needs to be.

Here are the sizes available, in plain terms:

The pocket-edition map

Each colour-channel on the map can hold values from 0 to 255 — that's about as much information as a normal computer picture stores per dot.

- World covered: **a 2.5-metre cube** — about one bedroom
- Use case: a single room, a piece of furniture, a character model
- Memory used to store it: about 64 megabytes (about as much as ten music tracks)

The standard map

Each colour-channel holds values up to 65,536. The map is denser.

- World covered: **a 655-metre stretch** — a village, a residential block, several office floors
- Use case: matches Lyra 2's actual published spec (90-metre walks)
- Memory used: about 2 gigabytes (a fast modern graphics card has 16 or 24 of these to spare)

The large map

Each colour-channel can hold integers up to 4.29 billion. The map is much denser.

- World covered: **a 42,000-kilometre area** — bigger than the surface of the Earth, technically (Earth's circumference is 40,000 km)
- Use case: a planet, in principle
- Memory used: a few terabytes — fits on a normal hard drive

The "folder of maps" trick — going bigger than any single map

For things bigger than one big map can hold (like an entire city at very fine detail), the trick is to **keep most maps on the hard drive** and only load the one for "wherever the user currently is" into the fast graphics memory.

This is exactly how Microsoft Flight Simulator handles the entire Earth. They have **2.5 petabytes** (about 2.5 million gigabytes) of world data stored in their servers. When you're flying over Paris, your computer loads the Paris maps. When you fly to Tokyo, it swaps to the Tokyo maps. You never have to fit the whole Earth into your computer.

For our scheme, the same trick works. The map's "address" tells the computer which folder to look in. When the user walks to a new part of the world, the computer fetches the next neighbourhood's map from the hard drive. The fetch happens in about 1/1000th of a second — faster than you can notice.

This means **a $400 graphics card plus a $200 hard drive** could in principle hold "the entire street map of San Francisco at 1-centimetre precision." That's a remarkable amount of world data on a hobbyist's computer.

7.5 How much does it cost to *make* a world?

Storing a world is one question. *Making* one is another. To turn a single starting photo into a walkable 3D environment, Lyra 2 has to do a lot of imagining — and imagining costs computer time, which costs money if you rent the computer, or hours if you use your own.

We surveyed the best-known tricks people have shared publicly (in research papers and open-source projects up through May 2026) for making Lyra 2 run faster. Stacked together — without retraining anything, just plugging existing pieces together — they roughly **60× speed up Lyra 2**.

What that actually buys you, in plain English:

On a rented computer (typical cloud rental: $3.50 per hour):

Size of world	Real-world example	Cost
A single shop or apartment	100 m²	**7¢**
A small village square	5,000 m² (~1 acre)	**$2.45**
A Monkey-Island-sized island	1 km²	**$73**
A five-island archipelago	5 km²	$364
A small district	50 km²	$3,640

A weekend of cloud rental ($100) gets you about a small village plus its surroundings, polished to "hero" quality.

← SCENE DEMO UVW EXPLAINER SOURCE ON GITHUB

On your own computer at home (an off-the-shelf graphics card, $400):

Size of world	Time to make it
A village (1 acre)	~1.5 hours
A Monkey-Island-sized island (1 km^2)	~7 days of leaving the computer running
A whole archipelago (5 km^2)	~36 days

So if you can leave your computer running overnight while you sleep, you can make a village every night. **Over the course of a year, you'd build an island.** Over a lifetime — a continent. The mathematics moves from "this needs a research lab" to "this is a weekend hobby."

This is the actual headline. NVIDIA's photo-to-3D-world technology *originally* needed a $70,000 chip and tens of thousands of dollars of compute. With the speedup tricks stacked together, **the same magic becomes a hobbyist hobby on a graphics card a teenager could buy with summer-job money.**

7.6 How your computer actually plays a giant world

Storage is one half of the answer. The other half is even more interesting: once you have a hundred-gigabyte map saved on your hard drive, *how does your computer let you walk around inside it?*

A typical good gaming computer has somewhere between **8 and 24 gigabytes** of fast graphics memory. Our Monkey-Island-sized island — generated, stored, ready to play — is **200 gigabytes**. That's ten times more than the computer can hold all at once. Yet you can walk around in it at full speed, with no waiting, no loading screens.

How? The trick is one of the loveliest in computer graphics, and it boils down to this:

> **You don't need the whole world in memory. You only need the bit of world your eyes are currently looking at.**

The library metaphor

Imagine a huge public library — a thousand bookshelves, a million books. You walk in and sit down at a reading table. Now imagine someone asks: "*How much of this library do you need to read, right this second?*"

The answer is: **one page**. The page you're currently looking at. Maybe two if it's open in front of you. Out of a million books, you need *one page*.

If you turn the page, you need the next one. If you put the book down and grab a different one, you need the first page of that. But you *never* need all the books at once, because **your eyes can only point in one direction and read one page at a time**.

A computer rendering a 3D world is exactly the same. Your screen has about two million tiny pixels. Each pixel points in a slightly different direction. To draw the scene, the computer has to figure out: "*For each of these two million directions, what's the closest thing the camera can see?*"

Two million directions × one cube visible per direction = **two million cubes** that matter, per drawn frame. Out of 50 billion cubes in the whole world.

That's the trick. The world is huge. The visible part is tiny.

Two million out of fifty billion

Let's do the math on what "tiny" means.

A Monkey-Island-sized island has roughly **50 billion 1-cm cubes** (walls, ground, palm fronds, water, all of it). 50 billion. Stored on the hard drive.

A typical computer screen, at the resolution most people play games at, has about **2 million pixels**.

So the ratio of "world data" to "what the screen needs right now" is:

> **50,000,000,000 cubes in the world ÷ 2,000,000 pixels on screen ≈ 25,000 to 1**

For every cube the player can *see* right now, there are 25,000 cubes they can't see. The computer's job is to *quickly find the right 2 million out of the 50 billion*, every sixtieth of a second.

This is doable because of **how cube space is shaped**. The 25,000 hidden cubes per visible cube aren't randomly scattered — they're spatially organised. The cubes you can see are the ones near your camera; the cubes you can't see are the ones far from you, behind walls, around corners, on the other side of the island.

The computer doesn't have to *consider* the 25,000 hidden cubes per visible one. It just *skips* them — by virtue of knowing where the camera is and which direction each pixel is pointing.

How fast can you read pages from a book?

OK, but here's the practical question. The visible cubes change as you walk. If you take one step forward, some cubes that were behind you become visible (when you turn), and new cubes ahead of you become visible (because you're closer to them). The computer has to *fetch* those new cubes from the hard drive into its fast memory.

How fast does this fetching happen?

A modern computer's hard drive (specifically an **SSD** — solid-state drive, the small chip-based fast kind) can read about **5 to 7 gigabytes per second**. That's the speed at which it can pull data off itself and hand it to the rest of the computer.

In one sixtieth of a second (one frame at 60 frames per second), an SSD can hand over about **80 to 100 megabytes**. That's roughly 80 million bytes of data, per frame, from disk to fast memory.

When you walk forward at a normal walking pace (about 5 metres per second), your camera moves **8 centimetres per frame**. That means *new* cubes coming into view per frame: about **6 to 8 of them along the view direction**. At 4 bytes per cube, that's **30 bytes of new cube data per frame**.

Thirty bytes. The SSD can move 80,000,000 bytes per frame. We're using **a tiny fraction of one percent** of what the SSD can do.

There's enormous spare capacity. You could run, sprint, fly, or teleport across the world, and the SSD would still keep up with delivering "the cubes you need right now."

The bookcase metaphor (the chunk system)

The library has another trick: books are organised in **bookcases**. Each bookcase holds about 20 books. The librarian doesn't bring you individual pages — they bring you whole bookcases worth at a time.

The computer does the same thing. The 50 billion cubes in the world are grouped into **chunks** — each chunk is a small cube-shape of the world about 16 × 16 × 16 cubes (roughly fingertip-sized in actual world units). The whole island is divided into millions of these chunks.

When you walk forward, the computer doesn't fetch individual cubes from the hard drive. It fetches whole *chunks* — 16,384 cubes at a time, in one go. Much more efficient.

At any given moment, the computer's fast memory holds maybe **200 chunks** — the ones near your camera. That's about 6 gigabytes of fast memory in use. The other six million chunks (or however many your island has) sit on the hard drive, waiting.

As you walk, the chunks at the *edges* of your visible region get swapped:

- New chunks from the direction you're walking into → loaded from disk
- Old chunks from the direction you're walking away from → forgotten, so the memory they were using can be re-used

It's exactly like a "sliding window" of bookcases, with you in the middle. You always have the same number of bookcases around you, but they're different bookcases depending on where you are.

What if you teleport?

Suppose you don't walk — you *teleport* to a completely different part of the island. Now suddenly the chunks the computer had loaded are useless. None of them are near where you appeared. The computer has to load **all 200 chunks of your new region from scratch**.

At 16 kilobytes per chunk, that's about **3 megabytes** of new data needed. The SSD can pull that in **under a thousandth of a second** — much faster than one frame.

So even a teleport has no perceptible loading time on a modern SSD. You blink, you're somewhere else, the world is already there.

The "doorbell" we built earlier — it makes this faster

Remember the doorbell trick from §8.5 — the one-bit "is anyone home?" tag at each cube? It plays a huge role here.

When the computer is figuring out which cubes are visible (the 2 million out of 50 billion), it has to check a lot of cubes along each view direction. *Most* of those cubes are empty (sky, air with nothing in it). The doorbell lets it **skip empty cubes instantly**: ring the doorbell, no answer, move on, never read the contents.

Without the doorbell, the computer would have to read each cube's full colour data just to find out if it was empty. With the doorbell, it reads one bit, finds out it's empty, moves on. **30 times less work.**

And the "did anyone look here" doorbell from §8.5.1 plays the role of **telling the hard drive which chunks to keep ready**. After each frame, the computer can ask: "*Which cubes did the player actually see this frame?*" The answer comes back as a list of cubes (or, more practically, chunks). The chunks NOT in that list can be safely forgotten from fast memory; the chunks IN the list must stay loaded.

This is the system in one sentence: **"Whatever cubes the player just looked at, keep them ready. Forget the rest."**

Even bigger worlds — when SSD isn't enough

For genuinely massive worlds — a whole city at 1-centimetre detail, say — the hard drive eventually fills up. **Microsoft Flight Simulator handles the entire Earth's surface** by keeping the world data on their *servers*, not your computer. When you fly over Paris, your computer streams the Paris chunks from Microsoft's data centre over the internet. When you fly to Tokyo, it streams the Tokyo chunks. The total world is **2.5 petabytes** (2,500,000 gigabytes) — a couple of thousand standard hard drives' worth — and you never need to download it all.

For our project, that's overkill — a single 2-terabyte SSD ($80 at the time of writing) can hold "the entire street map of New York City at 1-centimetre precision." Most people won't need more.

But the same trick works in principle. **The bottleneck stops being "how much can I fit on my hard drive" and starts being "how fast can the internet hand it to me."** At broadband speeds, you can stream a chunk in a few milliseconds — slower than an SSD but still well within the frame budget for not-too-fast camera motion.

The complete picture

So when you're walking around in our Mêlée-Island-sized world:

1. **The whole world** (200 gigabytes) sits on your SSD, waiting.
2. **About 200 chunks** (6 gigabytes) are in your graphics card's fast memory at any moment — the chunks around your camera.
3. **About 2 million cubes** (160 megabytes — but mostly the doorbells are read, not the full data) are *actually examined* per frame — the cubes your screen pixels point at.
4. **About 30 bytes per frame** are streamed from SSD to fast memory as you walk — the new chunks arriving at the edge of your visible region.
5. **The visible part is drawn 60 times per second.** All you see is the scene; the streaming and chunking happens silently underneath.

The dream is that the whole thing is invisible to the player. You walk, you see, the world is there. **You never know there's a 200 GB library being silently sliced into the bookcase you're standing in front of, 60 times every second.**

That's how a graphics card with 16 gigabytes of memory plays a 200-gigabyte world. **Not by getting bigger memory — by being *clever about what to look at*.**

---

8. The "don't look behind you" trick

Here's another small improvement we suggested.

When you're walking forward and you want to know what's around you, **you don't need to think about what's behind you**. Your eyes are pointed forward; the stuff behind you is invisible until you turn around.

The original Lyra 2 doesn't quite work that way. When it's deciding which photo from its album might be relevant to the current view, it **checks every photo**, including ones that are clearly behind it. Even photos showing things in the totally opposite direction get scored, just to be sure.

We suggested: **just don't bother with the ones behind you.** Skip them entirely. The AI never needed them anyway; the math was wasting time.

This is the smallest possible improvement in the whole proposal, and the *only one that doesn't require any retraining of the AI*. NVIDIA could turn this on tomorrow and get a small speed boost.

How big a boost? About 5-15% faster on long walks. Not huge — but free.

The fancy name for this is **"axis-aligned octant culling."** The plain version: don't pack things you're walking away from.

8.5 A tiny bonus — the "doorbell" trick

While we're here, one more small idea that sits nicely on top of everything else.

Most of the spots in our map are **empty**. Think of a 3D map of a hamlet square — the spots that are actually a wall, a tree, a fountain, a cobblestone are a tiny fraction of the total volume of air around them. Most of the map is just "sky" or "air with nothing in it."

When the computer wants to draw the scene, it walks through every spot asking *"is there anything here?"* For empty spots, it currently has to read **three bytes** (the colour information of that spot) just to find out the answer is "no, nothing here."

What if there was a **doorbell** at each spot — a one-bit yes/no that says *"is anyone home?"* — before you bother to read the three bytes of contents? If the doorbell says no, you skip the spot entirely. If yes, then you read.

The math is wild:

Spots in the map	Three-byte storage	One-bit doorbell	Doorbell is smaller by...
16 million (a small room at 1cm)	50 megabytes	2 megabytes	**24 times**
1 billion (a village block)	3.2 gigabytes	134 megabytes	24 times
8.6 billion (a city block)	25.8 gigabytes	1 gigabyte	24 times

The doorbell is **24 times smaller** than the colour data for every size of map. Reading it is much faster, takes much less memory, and the computer's quick cache memory can hold tens of thousands of doorbells at once — instead of just hundreds of colour entries.

And here's the elegant bit: for sparse scenes (where most spots are empty, which is *most* scenes), pairing the doorbell with **only-store-colour-for-actually-populated-spots** gets you the best of both worlds:

- **The doorbell** (always there, very small) lets you ask "is anything here?" instantly for any spot
- **The colour data** (only stored for spots that have something) takes only the memory you actually need

For a typical scene where 5% of spots are populated, the combined cost drops from **50 megabytes** down to about **5 megabytes** — *and* the AI can still answer "what's at this exact spot?" instantly. About 10× smaller than the original layout, with no loss of capability.

This is one of those "wait, why don't we already do this everywhere" tricks. It's actually a standard pattern in computer graphics (called an "occupancy bitmap") but it isn't currently in Lyra 2. Adding it is part of our proposal.

✅ **And we've actually built it** — the working code lives in our project at `pipeline/uvw_atlas.py` as a class called `OccupancyBitmap`. You can hand it a list of populated spots and it builds the doorbell map; ask it about any spot and it tells you yes/no instantly. We verified the math works for all 16.7 million possible spots in a small-room-sized map (took about a second on a regular laptop). The pattern can be lifted straight into Lyra 2 with minor changes; it doesn't need any retraining of the AI to add it.

8.5.1 An even better doorbell — the "did anyone look here?" tag

While building the rest of this project, we noticed the doorbell could be slightly bigger and do twice the work. Instead of just one bit saying "*is anyone home?*", give each spot **two bits**:

- **Bit 1:** is anyone home? *(occupancy — same as before)*
- **Bit 2:** did someone just look at this spot? *(touched-this-moment)*

The "did someone look here" bit gets set by the program drawing the picture every time it actually sees that spot — like a smudge on a visitor log. At the end of each picture, the computer can ask: "*which spots were smudged?*" and it gets back **exactly the list of places the viewer cared about this instant.**

Why is this useful? Because the computer can keep only those "recently-touched" spots in its fast memory and let the rest sit on the hard drive. As the viewer looks around — turning their head, walking forward, looking up — different spots get smudged each moment, and the fast memory swaps in the new ones from disk.

The trick that makes virtual reality work without lag. A headset-wearer can turn their head 180 degrees in a tenth of a second. If the computer pre-loaded only the spots in front of them, they'd see nothing when they turned. With the smudge-bit, **whatever spots the viewer's eyes touch are automatically the ones in memory** — for ANY direction they choose to look, no advance warning needed.

Storage cost: a 4 MB doorbell map instead of 2 MB. Doubles the doorbell, but still 12× smaller than the original colour data. ✅ **Built** — `pipeline/uvw_atlas.py`'s `OccupancyBitmap` now supports both modes (1-bit and 2-bit) via a constructor option. The 2-bit mode is what runtime virtual-reality rendering would use.

There's one more piece of cleverness worth surfacing: **the "did anyone look here" bit lives at the same address as the spot's colour.** Remember the whole project's headline trick — the spot's *colour* IS its *address* in the map. Well, the spot's "did anyone look here" flag is *also* stored at that same address. So when the computer hits a spot during drawing, **one address lookup gives it both the colour to paint AND the slot to mark as 'just looked at.'** One trip to the index, two answers. Cheap and elegant — exactly the shape of payoff the original colour-coord map was built for.

---

9. So… did it actually work?

This is the question, isn't it.

Honest answer: we don't know for sure.

We wrote the code that would change Lyra 2 to use the map instead of the photo album. The code **compiles cleanly** — meaning the computer can read it without choking. We wrote it

carefully so that **turning our new features off** produces exactly the same behaviour as the original Lyra 2 — meaning we haven't broken anything.

We wrote a small **self-test** that confirms the bijection (the math that says "this colour IS this coordinate, both ways") works perfectly for all 256 × 256 × 256 = **16.7 million possible addresses on the pocket-edition map**. Every single combination round-trips correctly. The math is solid.

But we **couldn't test the AI itself with our changes** — that would have required one of those $70,000 supercomputer chips, and a few weeks of training time. We don't have those. So whether our changes make Lyra 2 *better*, *worse*, or *the same in terms of quality*, we honestly don't know. We can argue from the structure of the problem that it should help, but arguing isn't measuring.

We were very careful in our proposal to NVIDIA to say this clearly: "*We think this is a good idea. Here's why. We haven't measured it. If you measure it and find it's a bad idea, that's useful too — please tell us.*"

NVIDIA's research team can choose to measure it, ignore it, or send us a polite "we tried this in 2024 and it didn't help." Any of those outcomes is fine. The point of writing the proposal carefully was to be *useful documentation either way*.

---

10. Where this goes from here

We did several things to share this with NVIDIA:

A formal request on their public website

The bit of the internet where programmers share code is called **GitHub**. NVIDIA has a public page there where they publish Lyra 2. We filed what's called a "pull request" — basically a formal "I'd like to propose this change to your code" suggestion.

It's marked as a **draft**, which is the polite way of saying "*I'm proposing this for discussion, not asking you to merge it tomorrow.*" NVIDIA's team can comment on it, ask questions, accept it, reject it, or just ignore it. Either way it exists publicly and anyone in the world can read it.

You can see it here:

https://github.com/nv-tlabs/lyra/pull/61

A written technical document

Alongside the code, we wrote a **paper** explaining the idea in detail — with diagrams, tables of numbers, and a list of experiments NVIDIA could run to test if the idea actually helps.

You can read it here:

/proposal (rendered, dark theme) or **PDF download** (482 KB)

A more polished private version for direct sharing

There's also a longer, more detailed version sitting on the author's local computer, ready to email directly to specific NVIDIA researchers if they're interested in a deeper conversation. That version isn't published publicly — it's for one-to-one exchange.

A live web demo

To show that the underlying idea (the bidirectional colour-coordinate map) actually works, there's a **live interactive demo** anyone can click. It shows a small 3D scene rendered two ways — once normally, once with the "colour = coordinate" trick — so you can see the map's address on every cube on screen.

https://downtoearth-9lq.pages.dev

Two demos there:

- The **photoreal scene demo** (a small village, made of 167,000 glowing dots)
- The **bijection explainer** at /uvw (with the map address visible on every cube)

A voxel-world walking demo (new, 2026-05-20)

To show that the **runtime** side also works — the bit where you fly around inside the saved world after it's been made — we built a small self-contained demo that runs in any web browser, no installation needed.

Open the file `voxgaussian/voxel_renderer/index.html` in any modern browser. You'll see a tropical island scene (sand floor, central tower, palm trees) made of 16 million tiny cubes, that you can fly around with the keyboard and mouse. Pick a different scene with the 1, 2, or 3 keys (island / stepped tower / glowing caverns).

The interesting trick: **the computer never stores the whole island at once.** It only thinks about the cubes the camera is currently looking at. Press F to turn that trick off — you'll see the frame rate drop. Press G to colour the parts the trick is skipping in red, so you can see exactly what gets ignored.

This is the same trick that lets a $400 graphics card play a hundred-square-kilometre world at full quality. The computer only ever holds the bit of world the player is looking at *right now*.

---

11. The complete list of everything we did

For completeness (this is the "full feature set" section the introduction promised), here's everything that went into this project, in plain language:

1. **Designed the colour-coord map** — the bidirectional bijection between a 3D coordinate and a 2D map pixel, with the property that the pixel's colour is the coordinate.
2. **Proved the math works** — wrote a self-test that exhaustively checks all 16.7 million possible addresses in the pocket-edition map. Zero round-trip errors.
3. **Built a small toolkit** — Python code that anyone can use to convert between coordinates, map pixels, and colours. Open-source under a permissive licence (MIT) so anyone can copy it for their projects.
4. **Made a live interactive demo** — a webpage where you can drag and zoom a small 3D scene and watch the colour-coord trick happen in real time. Works on phones, laptops, VR headsets.
5. **Built a tiny version of Lyra 2 for normal computers** — called "voxgaussian", it does a simpler version of what Lyra 2 does but on consumer hardware. Takes a photo, turns it into a small 3D scene with 167,000 fuzzy dots. Whole thing runs on a $430 graphics card in 5 minutes.
6. **Wrote three writeups** — a technical proposal aimed at NVIDIA researchers, a more polished private version for direct emailing, and this dummies version for everyone else.
7. **Generated PDFs** of everything for offline reading and emailing.
8. **Filed a formal proposal** with NVIDIA at https://github.com/nv-tlabs/lyra/pull/61, signed properly, with the right formal language and the right respect for their process.
9. **Made the code modifications real** — not just descriptions. We forked NVIDIA's repository, applied our changes carefully, signed off on the commit per their rules, pushed to a public branch, and opened the formal pull request.
10. **Designed the "folder of maps" trick** for going beyond a single map — a way to handle arbitrarily-large worlds by streaming chunks from a hard drive on demand.

11. **Designed the "don't look behind you" optimisation** — a smaller, free improvement that doesn't require retraining and could be adopted immediately.
12. **Designed the size family** (pocket / standard / large / massive) so the same scheme can be applied to single-room, village, or planet-scale problems by just picking different settings.
13. **Wrote a permissive licence** on everything we did — meaning anyone, including NVIDIA, can use any part of this work without paying anything or asking permission.
14. **Hosted everything on a public website** — https://downtoearth-9lq.pages.dev — so the whole thing is browseable, downloadable, shareable.
15. **Wrote this dummies version**, because the technical writeups are useful but only to people who already know what a "diffusion model" is. This page is for everyone else.
16. **(May 2026) Added the "did anyone look here?" tag** to the doorbell — a 2-bit version of the occupancy bitmap that records which spots the player's view actually touched each instant. The data structure for virtual-reality-safe streaming (head can turn 180° instantly and nothing pops in).
17. **(May 2026) Built a voxel-world walking demo** — a single-file webpage that flies around a 16-million-cube test world at full speed on any modern graphics card. Proves the runtime renderer pieces work on consumer hardware. Includes the "only think about cubes the camera is looking at" trick.
18. **(May 2026) Surveyed the speedup stack** — independently researched what people have shared publicly about making Lyra-2-class AI go faster. Found a stackable combination of tricks (already-shipped distillation + content-aware step skipping + compiler optimisations + native fp8 maths) that gives roughly **60× speedup** over stock Lyra 2, with no retraining required. This is what makes a Monkey-Island-class 1 km² world cost ~$73 of cloud computing instead of ~$1,400.
19. **(May 2026) Designed entity-structured prompts** — instead of giving the AI a freeform sentence ("a fishing village at dawn"), give it a small list of named things it should include (`sky`, `dock`, `boat`, `palm`, `fisherman`) with canonical expanded descriptions. Self-organising so the most-used entities in your project automatically get the cheapest storage. Plugs into the same colour-coordinate atlas as a per-voxel semantic tag, so every cube knows *what* it is, not just what colour.

---

12. Words that might have flown by — a glossary

In case you want to look up what any of these meant later:

← SCENE DEMO UVW EXPLAINER SOURCE ON GITHUB

Word	What it means in plain English
AI	A computer program trained on a lot of data to do tasks (like recognising faces or generating images) that would normally need a human
GPU / Graphics Card	The special chip in a computer that does a lot of math at once. Mostly used for video games and AI
VRAM	The fast memory inside a graphics card. It's much faster than your computer's regular memory, but there's less of it
Diffusion model	A specific type of AI that's good at generating images from descriptions. The kind that powers things like Midjourney and DALL-E
Voxel	A small cube of 3D space. Like a pixel but in three dimensions
Gaussian splat	A fuzzy glowing dot used to render 3D scenes. Modern 3D AI uses millions of these together to make photoreal scenes
Bijection	A fancy math word for "a perfect translation that works in both directions." Our colour-coordinate trick is a bijection — colour goes to coord, coord goes to colour, both perfect
ControlNet	A "leash" on an AI image generator that tells it "draw a tree HERE specifically." Used to add structure to AI-generated images
GitHub	A website where programmers share code. NVIDIA published Lyra 2 there. We filed our proposal there
Pull request / PR	A formal proposal to add some code to a project. Like a politely-formatted "here's an improvement, what do you think?"
Draft PR	A pull request marked as "I'm proposing this for discussion, please don't merge yet." Polite RFC stance
DCO	A formal sign-off saying "I have the right to submit this code." Required by NVIDIA's contributing rules
Octant culling	The "don't look behind you" trick. Halves the work the AI has to do per step, for free
Streaming	Loading data from a slow place (hard drive, internet) into a fast place (graphics card) just-in-time, only when you need it

Word	What it means in plain English
Lyra 1 / Lyra 2	The names of the AI we're talking about. Lyra 1 came first (September 2025), Lyra 2 followed (April 2026) and is the one we focused on

13. A note for the proud reader

This whole project — the data structure, the code, the demos, the writeups, the proposal to NVIDIA, this very page — was built by one person on a $430 graphics card in their evenings.

That's not a humblebrag. It's a comment on what's become possible. The same project ten years ago would have needed a research lab, a team of three to five people, hundreds of thousands of dollars of hardware, and several years. Today, with the right open-source pieces and the right architecture, one motivated person with consumer hardware can produce something that's at least *a serious conversation* with one of the best-funded research teams in the world.

The conversation might not go anywhere. NVIDIA might not respond. The proposal might sit in a queue for months and eventually quietly close. **That's fine** — the artifact exists publicly. Someone will Google "Lyra 2 architecture extensions" in two years and find it. The contribution is made.

But if NVIDIA *does* respond — even with "we considered this and decided not to do it because X" — that's still a win. That's a public document about why a specific architectural shift wasn't worth pursuing. Future researchers will benefit from that.

And on the small chance the response is more positive — that some engineer at NVIDIA actually picks up the cheap "don't look behind you" optimisation, or even folds the colour-coord-map idea into Lyra 2 v3 — well, that would be the kind of "lone hobbyist's idea shipped in a major commercial product" story that doesn't happen very often but used to happen all the time in the early days of the internet.

Either way, you now have a working understanding of one of the more interesting AI-architecture questions of 2026, and you have it without needing to learn what a "tensor" is or what "GPU memory bandwidth" means.

Thanks for reading. Tell a friend. The technical version is at /readme; the proposal is at /proposal; the live 3D demo is at the main page; the formal NVIDIA submission is at pull request #61. If anything here was unclear, that's our fault — drop us a note and we'll improve it.

🙏

Document written and maintained by MiLO + Opie. Hosted at https://downtoearth-9lq.pages.dev/dummies. PDF version at https://downtoearth-9lq.pages.dev/DUMMIES.pdf. Last updated 2026-05-19.