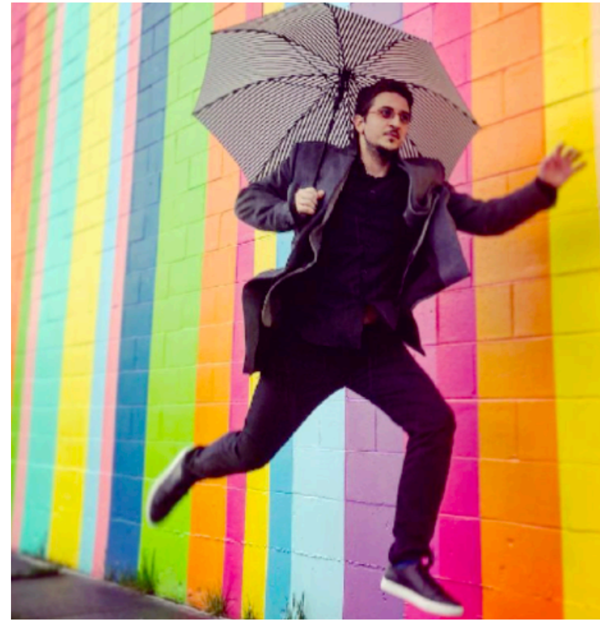
Image: https://www.roblox.com/users/1704534226/profile/#!/creations

#WhoIs
Angelo

- Senior TD @ Roblox
  - Heading our Rendering Team
- Before: AAA
  - Milestone, EA, Relic, Capcom, Activision
  - Many games (13?), lots of grey hair
- Real-Time Rendering: 4th edition
- Also on:
  - Twitter @kenpex
  - c0de517e@blogspot.com



Wisdom?

Lessons learned?

- **Engines** x **Games** x **Companies** x **EndResults** = **Many** (white hairs)
- **Design spaces:**

Low-level

- Graphs or lists? Stateless or stateful?
- Specialized structures per object type?
- A “generic” Mesh+Shaders+Resources object?
- Drawable object with virtual functions?
- Command buffer snippets + sort key?

High Level

- **How to foster innovation?**
  - Shared tech? Or specialized?
  - One engine to rule them all?
- **(Technical) Artist driven or Hardcoded rendering?**
- **Artist time** (baking) **vs Run-time?**

- (Many) **More questions than answers!**
  - Not even sure about Forward vs Forward+ vs Deferred vs VisBuffer etc… 😢😭🤯

Wisdom!

Lessons learned!

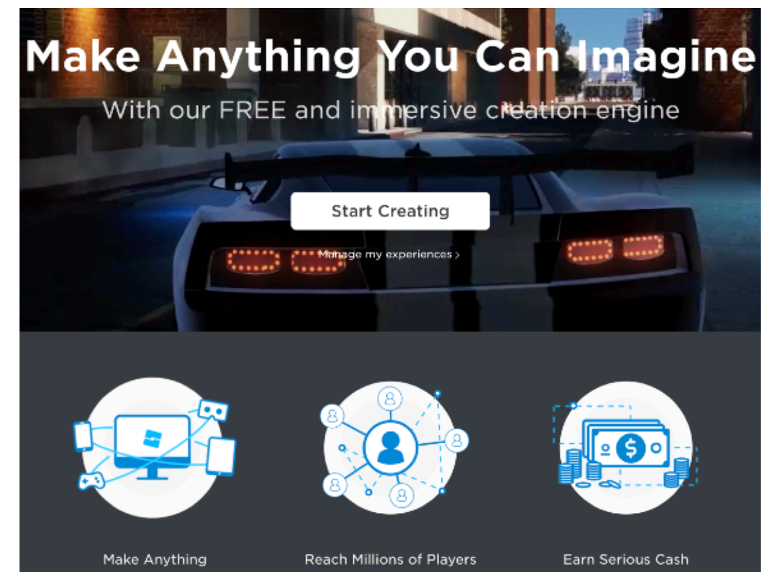
- **Product > Technology & People > Product**
- At best **Technology** serves **People & Product:**
  1. No abstract technological or process “optimum” / beauty
  2. Optimal engineering must leverage a correct amount of tech debt
     - *No great product is ever good*
  3. All non-trivial technology is a matter of **tradeoffs**
     - No single way to success
     - History, culture shapes engineering: **Conway’s law**
       - Any theory re:best practices must consider **context**
  4. If you want to shape technology, you should change people / incentives

Context: Roblox

The “metaverse” in 3 slides

1) Roblox is…
…NOT a Game Engine

- Not a **middleware**
  - Pay $, get tools
  - Use tools to help/create your product
  - Profit
- No way to **"get"** Roblox
  - Roblox is not even just code/ infrastructure…
  - …it's a community!



2) Roblox is…

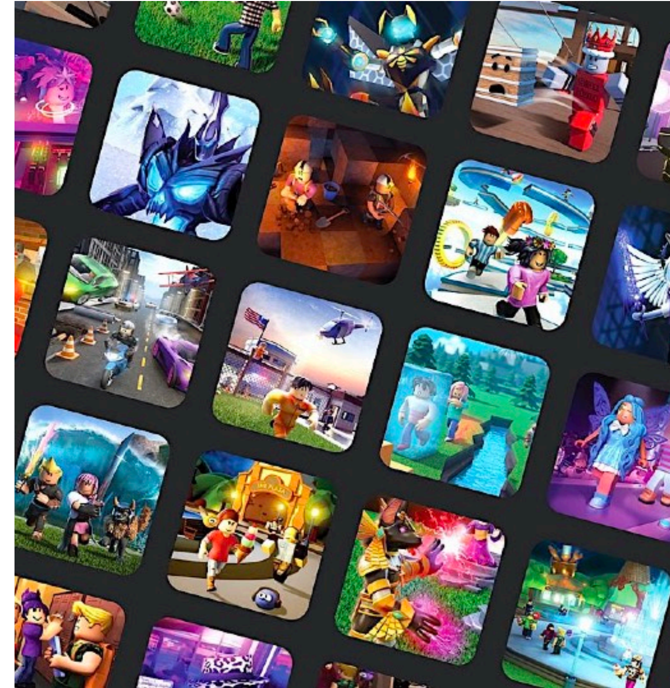
…NOT a Virtual World

- Roblox is not a **game**
  - No single, shared world
  - Not even a default lobby / space
- No **constraints on creativity**…
  - Similar to a game engine, everything's possible
  - We don't create art, dictate style…
  - …but out-of-the-box **defaults** to "realistic" social spaces
    - e.g. 3rd person avatars, physically simulated worlds

3) Roblox is…

…a Social Network for Creativity

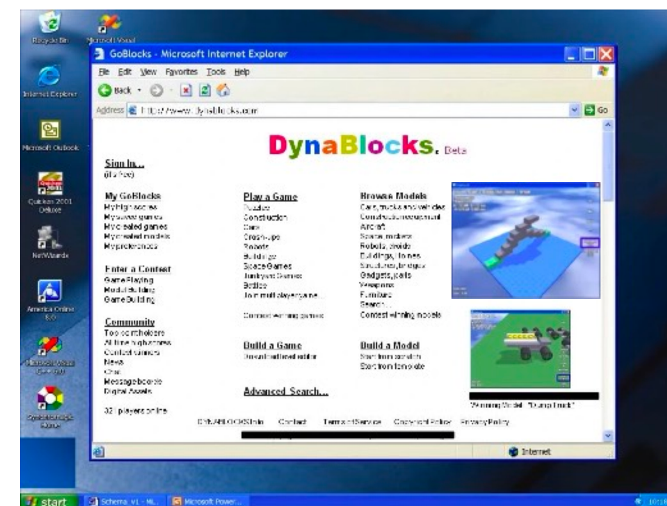
- More YouTube/Instagram than Unity/Unreal or Minecraft/Fortnite
  - I have my **identity** in Roblox (avatars)
  - I **connect** with others:
    - Through creativity (Team Create) and Play
  - Creative media: **Interactive 3D Worlds**
- Worlds are described with **high level primitives**
- **Game-Engine "materializes"** the worlds:
  - Simulation, Delivery, Visualization on (almost) **any device**



Roblox

The company

- Origins: 2004, DynaBlocks
    - Educational Playground
- Small for a long time, still *relatively* small today
- Rendering:
    - Engine started with G3D, then moved to Ogre, now it's bespoke
    - Team? Most of its history: one engineer
        - ~4 in 2018, ~15 in 2021
- ~700 engineers today in 77 teams



Further reading…

- Digital Dragons 2020: **“Rendering the meta verse across space and time”**
    - “Roblox vs AAA”
    - Less engine, more rendering
- GDC 2020: **“Building the Metaverse through User Generated Content and the Cloud”**
- GDC 2018: **“The Mechanics of Explosive Growth”**
- **“Habitat”** ~1990, captures surprisingly many of the same principles

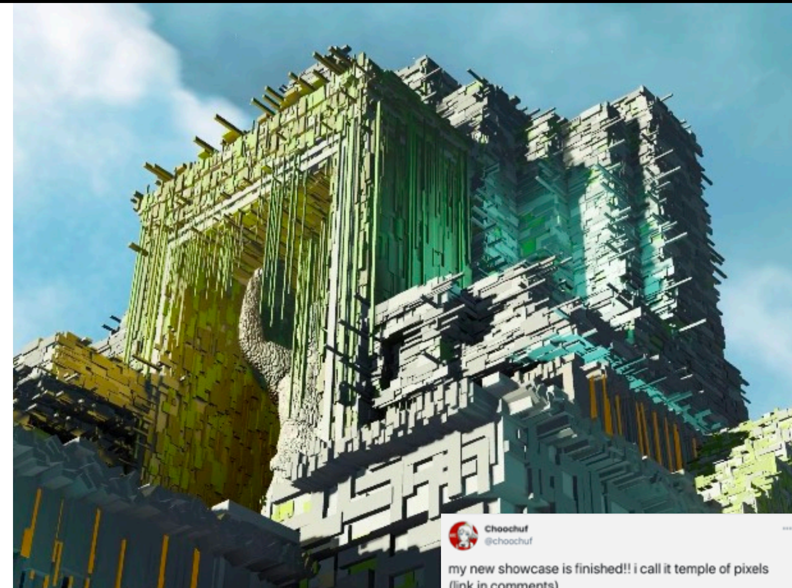
The Lessons of Lucasfilm's Habitat

Chip Morningstar and F. Randall Farmer

Habitat has recently been open sourced and resurrected as “NeoHabitat”. Playable on your nearest c64 (or emulator)

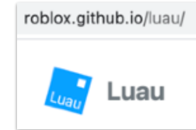
Virtual Worlds

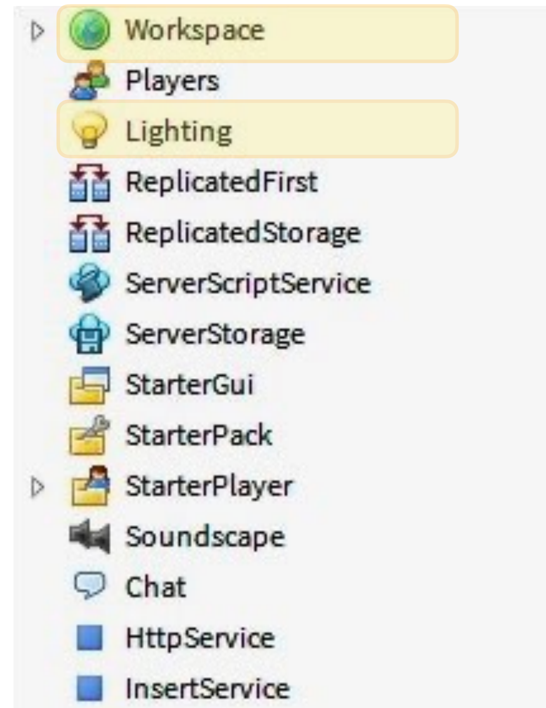
In Roblox



(0,0,0)
The Origin of the World

- **The "DataModel"**
  - Analogy: HTML DOM
    - Lua(u) as scripting
- **Multiple root "Services"**
  - A tree (forest) to keep objects organized
    - No default inheritance semantics
      - (e.g. transforms, attributes)
      - Some objects do affect children
- **Workspace** = The 3D, physically simulated World
- **Lighting:** Visual-only, not part of the "physical" world





(0,0,0)

The Origin of the World

- The DataModel is natively **Client/Server**
  - **RBXL** ~= HTML, static content
  - Clients do not get RBXL; Servers do
    - Objects and Properties are **streamed**
    - Clients do not get the full world / at full fidelity
- **Lua(u) scripts are sandboxed**
  - Scripts are by default server-side, changes replicate
  - ServerStorage objects do not replicate to the Client
  - Local (client) scripts for presentation code
    - Objects created do not replicate

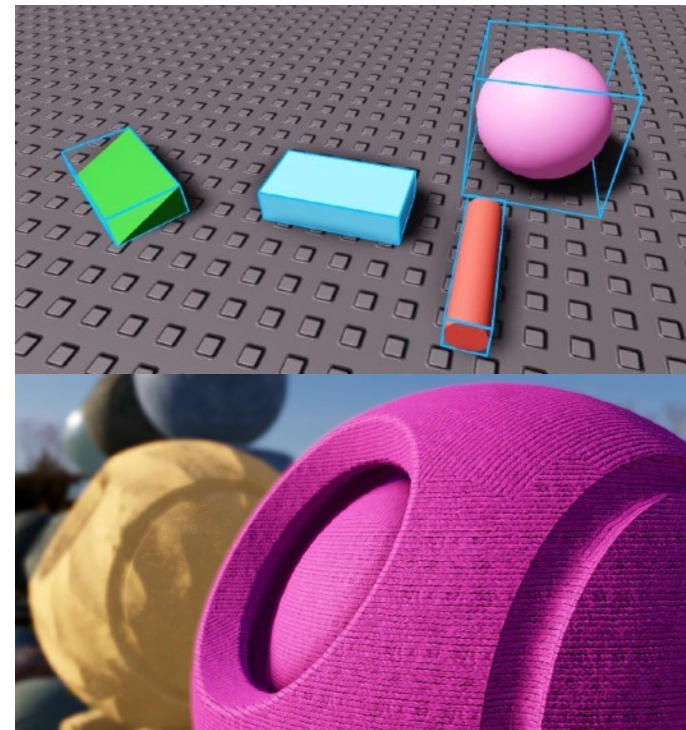
“at full fidelity” - example: today, far models can be replaced by voxel imposters to reduce streaming lag

Note: Both Luau sandboxing and the RBLX streaming are security features - clients cannot inject gameplay code, nor download the source of the game…

What are worlds made of?

(Rendering) Primitives

- **Parts**
  - Cubical projected textures + Decals
  - Can CSG (in runtime too). Exact collisions
- **MeshParts**
  - UV mapped. Convex decomposition & automatic LOD
  - First attempted in **2006**, pulled in 2008, finally back in **2016!**
    - **2021** introducing Skinning
- **Others:** Voxel Terrain. Particles / Beams / Trails…
- **Materials:** not just textures/shaders, but physics, audio…



Distributed computation?

Client + Servers + Cloud Services

- **Servers: Roblox Cloud Compute**
  - Linux build of the Game Engine
- **Cloud Services**
  - Developer-facing: Analytics, Data Store, Marketplace…
  - Internal: Too many to list…
  - **Notably, for today: Asset Service**

Assets

UGC Content Pipeline

- **Assets are Immutable**
  - (but support uploading new versions)
  - CDNs & Content-Addressable Storage (Amazon S3)
- E.g. Image, Meshes, Sounds, Animations, Static CSG…
- Immutable = Can be moderated!
  - Humans + AI…
  - Bespoke review tools per asset type

CDN: High latency, high bandwidth

Assets

Lifetime

- **Upload**:
    - Validation & Normalization (web endpoints)
- **Moderation:**
    - Approved = Asset is accessible
    - Kicks **initial processing**, e.g. thumbnail generation…
- **Client** requests an **AssetID:**
    - Resolve to the best available version for a given platform
    - If needed, kick **“lazy” optimizations**
        - Internal versioning!

Lazy optimizations - E.g.:
- Server-side texture compression (dxt, etc…)
- PBR material transcoding (specular AA, packing, automatic micro-AO…)
- Mesh processing (LOD…)

Internal versioning - we can change formats, tools, and assets will lazily re-process to update. We can invalidate certain versions and force re-processing…

Assets always fall-back to the unoptimized format (e.g. a PNG for textures, instead of DXT/BCT) if needed…

Everything is

Dynamic

- 99.9% of the properties are mutable, **exposed to Lua(u)**
  - Roblox Studio = GUI on top of the runtime DataModel
- **Plus** everything has **physics by default**
  - Server-side + Client ownership of nearby objects
- Examples:
  - Static objects? No! Anchor constraints
  - Models? Just weld together parts…
  - Even Avatars are (mostly) a bunch of parts & physics

Nothing that Studio can do is not doable in games or via other tools! E.g. all modeling is available to games, many games are creative in nature…

The DataModel, its objects and mutable properties, is 99% of the Roblox meta-verse.

Everything is

Dynamic

- Rendering must **infer optimizations** from the DM
  - No instancing / references
    - Find ways to reduce draws…
  - Not even a transform hierarchy!
- Scale, today?
  - Roblox Battle Royale ~50k parts, fully destructible
  - Townhall ~ 500 players in the same location…

Everything is
Tech-Agnostic

- Tech changes over time / across devices
  - ~15 years old platform already!
  - Delivering worlds (currently) from GLES2/DX9 devices up to DX11/Vulkan/Metal
- Build worlds with **high-level, universal concepts:**
  - **No authored optimizations**
    - e.g. No Occluders, Portals, Lightmaps, Probes…
  - Hints are acceptable - e.g. how important is the quality of a given object?
  - **No authored tech-specific parameters**
    - e.g. Shadow-map bias, SSAO samples…

Platform split is roughly equal between PC, iOS and Android

Side effects

Constraints or features?

- Fundamentally easier:
  - Real-time rendering couples technology and creativity, making game making hard
- More social:
  - Strong semantics
  - Objects and scripts made for one place can be shared and be expected to work in others

…in fact, creators that share models usually package in the model also the relevant scripts that control its behavior!

Engine design
The “anti-AAA"

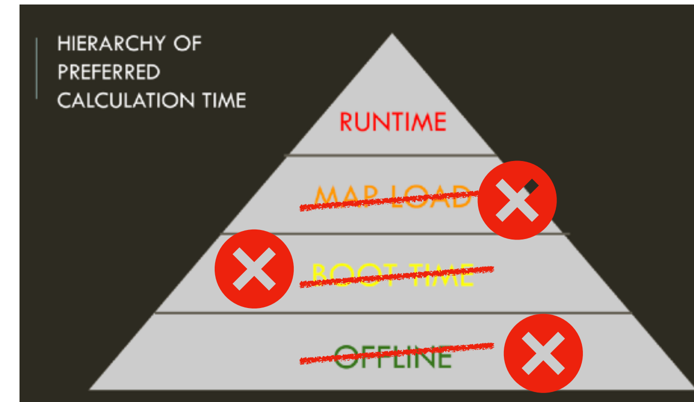
ΛΓTBLOX @ArtBlox_406 · Oct 30
Cyberpunk 2077 but not delayed.
#Roblox #RobloxDev

This is a hard problem!

Recap:

- We don't allow content to be (hand) optimized
- We don't allow content to know about technology
- We deliver to ~100k different devices
    - Mostly low power & obsolete (kids)
- Everything is dynamic, zero precomputation
- We can distribute computation:
    - Low-latency / power = Client
    - Mid-latency / power = Server
    - High-latency / power = Cloud

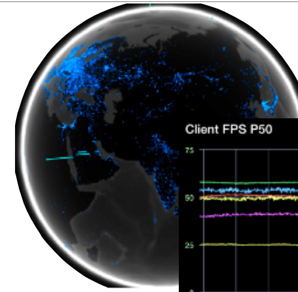
& we have a relatively small engineering team…

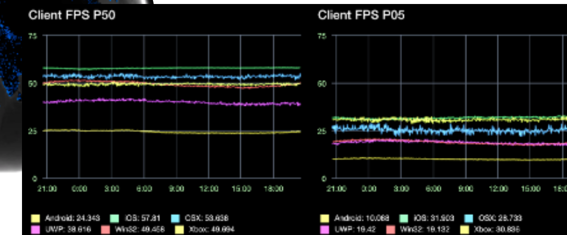


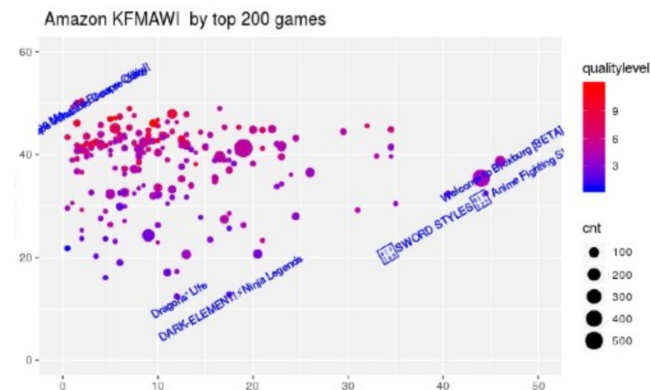
How hard is this, really?

Example: Profiling

- At scale? **Data Analytics**
  - Still an open problem for us!
- “Usual” pipeline has limited usefulness
  - Testing on device(s), vendor tools
  - QA + Automated smoke tests
  - Mostly for correctness, not tuning
- Most changes, when deployed, "average out" in global graphs!
  - Too many devices x Games x Settings
  - On-device behavior are hard to understand
  - Lots of interactions: engine load-balancing, streaming/memory, hw thermal throttling…



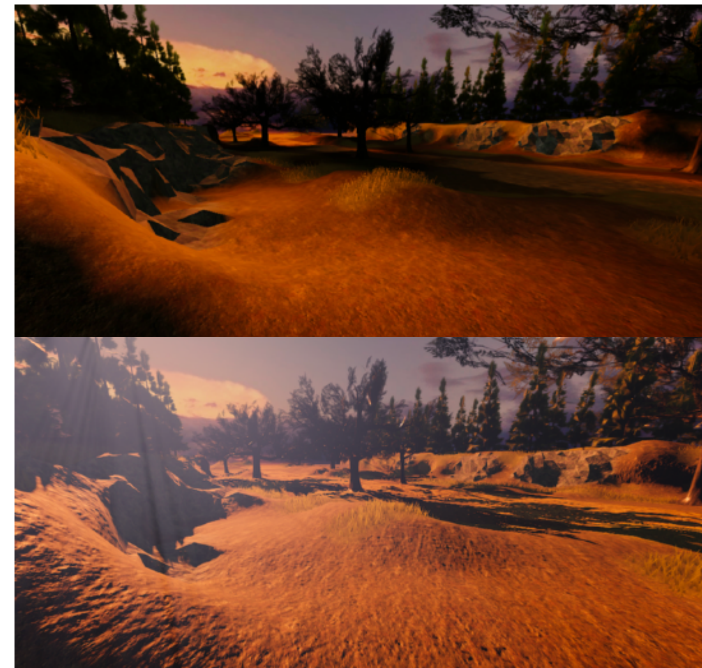




Where do we cheat?

~~Optimal~~ -> Scalable

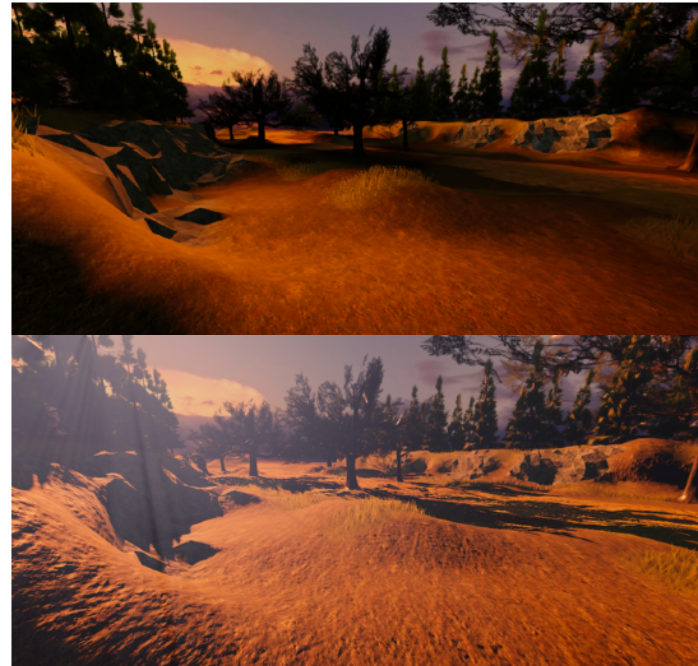
- **Graceful degradation:**
  - Clients always work, try to never crash
  - Ruthlessly reduce quality
- Easiest way: disable effects, fallback
  - Ok if it does not impact gameplay
- **Time-slice everything!**
  - Incremental updates & fixed budgets:
  - Shadowmaps… probes… voxels…
  - …even vertex buffer creation!



Where do we cheat?

~~Optimal~~ -> Scalable

- Automatic rendering quality: “**FrameRateManager**"
  - Monitors device performance
    - ~ Dynamic render resolution in an AAA game
    - But controls all rendering parameters!
    - Originally designed to control CPU performance
- E.g. Which rendering backend:
  - From direct sun only, no shadow-maps…
  - …all the way to all dynamic lights + EVSM F+
  - LOD & Cull distances, Post-fx…
  - Budgets (milliseconds, memory)
- Problem: Hard to tune!
  - Cache settings for a given game, on device



Again… data analytics & learning

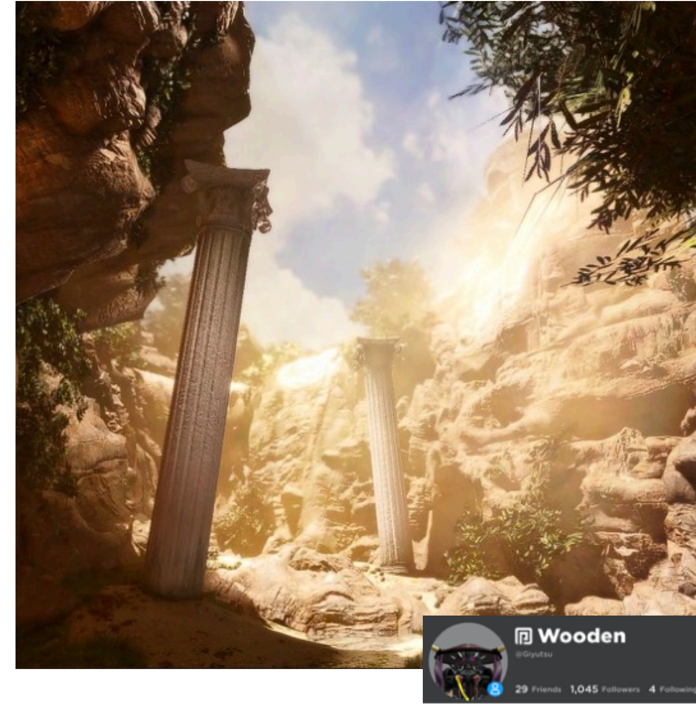
Where do we cheat?

"Take the long view"

- Will never "beat" any AAA game when they come out…
  - …but our clients never stop evolving!
- Future: What prevents us from beating them all?
  - We can always add detail!
  - Power of tech-agnostic authoring…

Recap:
Technology Principles

- Performance has to be primarily about **scalability**
  - Never crash
  - Disable effects / fallback on low quality
  - Amortize aggressively over time
- **Cache** optimized rendering data
  - Everything can change…
  - …assume that few things will change every frame
  - *Scripting is not that fast*
- Work with **layers of abstraction**
  - Code quality matters, always evolving, always running codebase
  - We cannot target single HW anyways…



https://www.roblox.com/games/6855297473/Semiarid-Showcase-5-24-21

Engine, Today

Implementation

Rendering Architecture

Threading

- *“Oldschool”: two-thread model*
- Main thread sync: **Prepare**
  - Locks the DataModel
  - Fetches any relevant changes
- Render thread: **Perform**
  - In parallel with next-frame simulation, cannot access DM
  - Takes the changes and creates the appropriate rendering structures
  - Kicks actual draws

Rendering Architecture
Threading

- *“Oldschool”: threadpool / jobs*
  - Not a graph. Jobs that can create other jobs
  - Originally made to async computation over multiple frames/steps!
- Sub-optimal?
  - Latency/Throughput … mostly about performing optimizations
  - We’re far from pushing the current system to its limits
  - Oldschool approach is simpler to reason about
- Doubtful that even at its limits, it would be less optimal or more complex (for similar performance) than a graph system

Step by Step

DataModel to Rendering

- DataModel -> Rendering: Hybrid Event/Push system and Pull…
- **Part/MeshPart(s)**
  - All objects in the DM derive from an “**Instance**” type
    - Provide **reflection** support and **events**
  - **“GfxBinding”**
    - Rendering interface that listens to property changes and add/remove child
  - Every rendering subsystem **implements a GfxBinding** for the DM objects it manages
    - Typically, mark/queue relevant rendering structures to be invalidated
- **2D/3D GUI “Adorns”: “GfxGui”**
  - Record GUI VM commands…

Step by Step
Prepare

- Read-only DM lock
  - Processing can still be parallelized *(Alas, currently, it is not)*
  - As light-weight as possible, no rendering work
- **Process invalidation queues:**
  - e.g. Part/MeshPart(s)
    - **Pull** the relevant DM changes, e.g. matrix transforms etc…
- **Process global state:**
  - Directly copy all the data over, no delta/events…
  - E.g. Lighting settings, post-fx etc

Step by Step

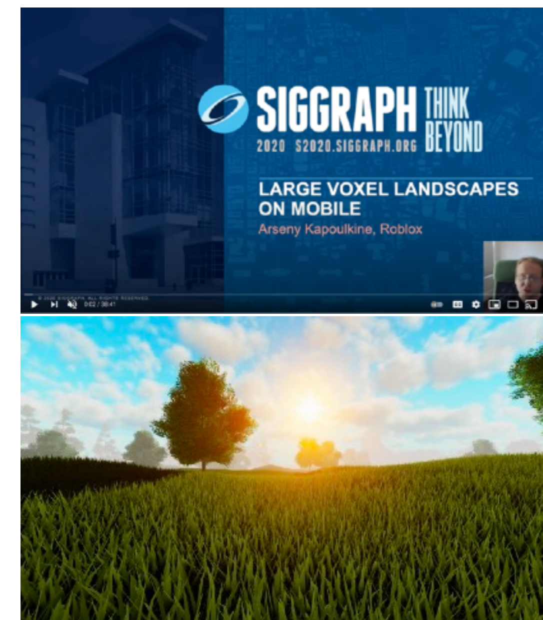
Perform: *Clusters*

- Never draw DM objects 1:1
  - **Want to achieve ~100:1 objects to draws**
- We always create “**clusters**”
  - Different systems per **object type** & **API capabilities**
- Assumption: lots of dynamic objects in a scene, but only a tiny fraction actually change per frame
  - Lua(u) is not that fast…
    - …& no methods for bulk updates / hierarchical changes 😉
  - *Physics/Animation are fast, but we can optimize these paths internally…*

Step by Step

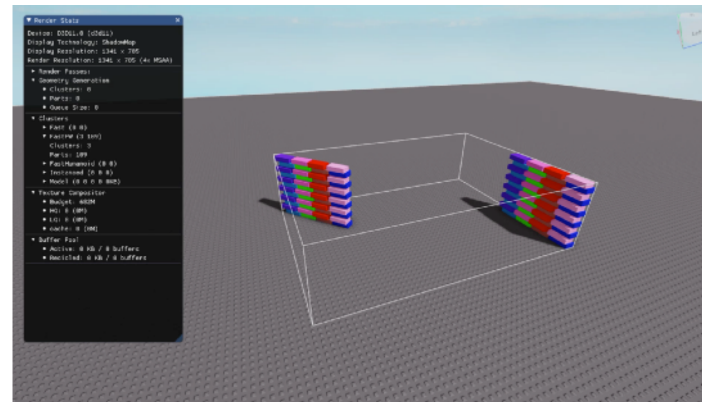
Perform: *Clusters*

- Today: we will see the **two main clustering systems**
  - Parts & MeshParts = Cluster spatially
  - Rendering: **FastCluster** or **InstanceCluster**
- Many more exist!
  - **Terrain (voxels) = “SmoothCluster”**
    - With another cluster for geometric “decorators” (grass)
  - Streaming Imposters = “ModelCluster”
    - No textures, just vertex colors. De-duplicate identical instances
  - 3D for GUIs (Picture in Picture) = ViewportFrames / GUICluster
    - Basic lighting only
  - 2D/3D GUIs = “VertexStreamer”



FastClusters
Circa 2012

- Supported on **all devices**
    - Create on the fly new IB/VB to aggregate chunks of geometry
- **Heuristics** for when to split / aggregate
    - Grid-based (+ physics constraints)
    - Small vs Large parts go into different cells.
- Reclustering is expensive - generating new geometry
    - Done **incrementally**, over time…
- Further opt: **Static and Moving** cluster per each spatial cell
    - Moving = "Dynamic FastClusters" - up to 72 transforms as constants
    - Some material properties passed as vertex data to further minimize draws

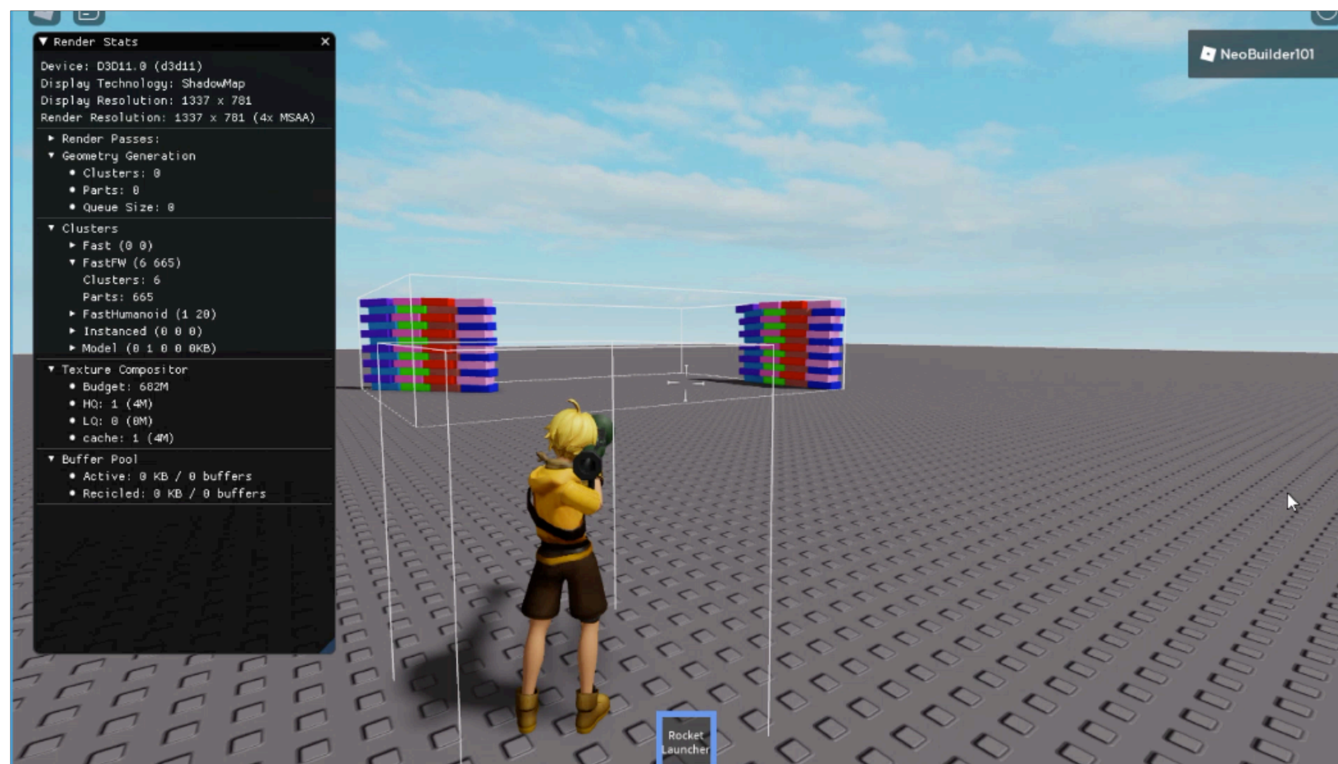


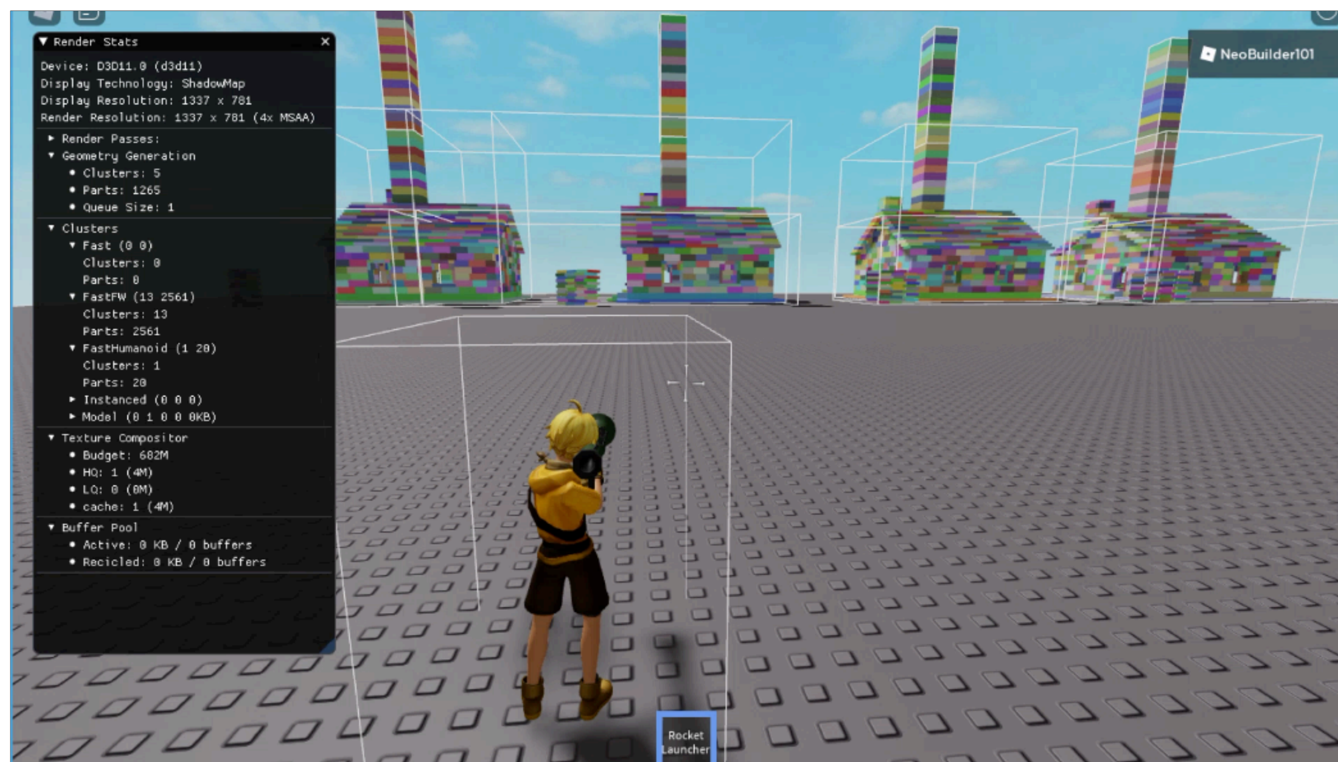
This was done ~the time we switched out of OGRE as the rendering system
OGRE was our second renderer, first was McGuire’s G3D!
Current GfxCore/GfxRender/GfxBase is ~10x smaller than OGRE was!

FastClusters
Circa 2012

- **Humanoids** are a special case
  - Assume all unique
  - Always dynamic
- Atlas textures - allow single draws, one cluster per humanoid
- WIP - **2021**
  - Compressed atlas (DXT…)
  - SW raster (in UV) to mark used texture blocks, copy them over
    - Support 3 levels of mip
  - Occlusion culling to remove parts hidden by clothing

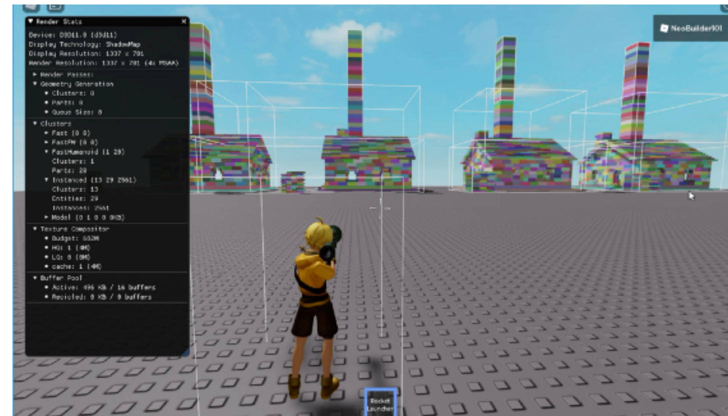
FastCluster actually originated for character rendering (humanoids), and today they are still the way we render them even when InstanceCluster is available on the HW


Render Stats
Device: D3D11.0 (d3d11)
Display Technology: ShadowMap
Display Resolution: 1337 x 781
Render Resolution: 1337 x 781 (4x MSAA)
Render Passes:
Geometry Generation
Clusters: 0
Parts: 0
Queue Size: 0
Clusters
Fast (0 0)
FastFW (6 665)
Clusters: 6
Parts: 665
FastHumanoid (1 20)
Instanced (0 0 0)
Model (0 1 0 0 0KB)
Texture Compositor
Budget: 682M
HQ: 1 (4M)
LQ: 0 (0M)
cache: 1 (4M)
Buffer Pool
Active: 0 KB / 0 buffers
Recicled: 0 KB / 0 buffers
NeoBuilder101
Rocket
Launcher


Render Stats
Device: D3D11.0 (d3d11)
Display Technology: ShadowMap
Display Resolution: 1337 x 781
Render Resolution: 1337 x 781 (4x MSAA)
Render Passes:
Geometry Generation
Clusters: 5
Parts: 1265
Queue Size: 1
Clusters
Fast (0 0)
Clusters: 0
Parts: 0
FastFW (13 2561)
Clusters: 13
Parts: 2561
FastHumanoid (1 20)
Clusters: 1
Parts: 20
Instanced (0 0 0)
Model (0 1 0 0 0KB)
Texture Compositor
Budget: 682M
HQ: 1 (4M)
LQ: 0 (0M)
cache: 1 (4M)
Buffer Pool
Active: 0 KB / 0 buffers
Recicled: 0 KB / 0 buffers
NeoBuilder101
Rocket Launcher

InstanceClusters
~2018

- For devices with **working hardware instancing**
  - In some situations might not batch as aggressively, leading to more drawcalls
  - In practice not a problem & these devices can cope with more draws
  - 112 bytes per instance
- **Hybrid static / dynamic**
  - Big enough clusters (70+ parts) statically upload instance data to GPU
  - Small clusters, single parts are aggregated dynamically
    - Especially useful for transparencies!
- **Big win** - Use much less memory, can render bigger views
  - Not bindless (yet)
- Supports Mesh LODs (switch entire cluster - uniformly)



Step by Step

Perform: Issuing Drawcalls

- Multiple abstractions to keep sanity
- **GfxRender: Drawing**
  - Culling (spatial hash) & sorted Queues…
    - Most clusters implement common interfaces for this
  - Some other rendering subsystems draw more directly / using specialized systems
    - e.g. UI, Particles
- **GfxCore: HW API**
  - All rendering goes through this common HW abstraction API
    - Resources, Techniques, Geometry…
    - Cap bits

GfxCore

- Supported APIs:
  - DirectX 9 (shader model 2.0) - Deprecated
  - DirectX 11 / 11fl10 (shader model 4.0 / 4.1 / 5.0)
  - Durango DX11 (shader model 5.0)
  - OpenGL 2.x, 3.x
  - OpenGL ES 2.0, ES 3.x
  - Apple Metal (iOS / macOS)
  - Vulkan (Desktop / Mobile)
  - Driver/GPU bug workarounds :)
- Roblox Shader Compiler
  - Shaders are written using HLSL
  - Small number of permutations, hardcoded (by design)
  - Backends: HLSL, SPIR-V, Metal Shading Language, GLSL

XBOX

Vulkan.

OpenGL|ES™

Lighting
and Shading

https://www.roblox.com/games/4422066367/Morning-Light-Showcase

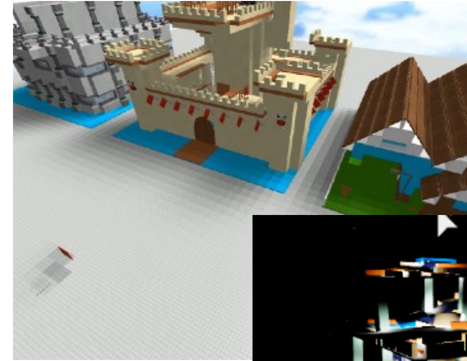
Shading the Metaverse
Principles

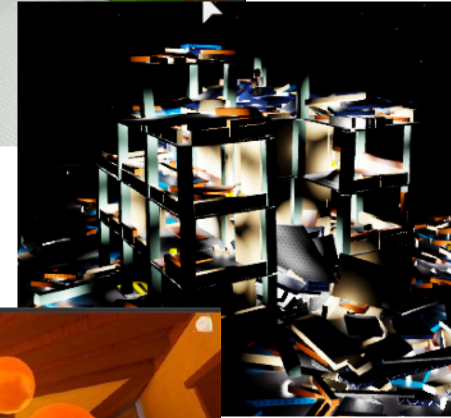
- **Decouple everything**
  - Draws: Clustering - Culling/Sorting - HW API
  - Lighting/Shading from Draws
  - *But certain features are implemented only for the newer InstanceClusters*
- Not about devices!
  - We don't control game complexity..
  - Even powerful devices might need to use simpler rendering
  - APIs also do not tell the whole story:
    - Some low-power devices do support modern APIs

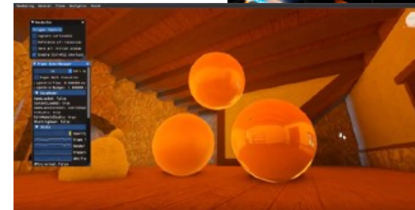
Lighting Technologies

As of Today…

- **Voxel Lighting**
  - CPU, SIMD, incremental
    - Voxel chunks with skirts
  - Local lights, Sun shadows, Sky Occlusion
    - Only sun is analytics
    - Other lights “diffuse” ~ LPV
  - *Experimented w/GPU voxels, not shipped*
- **ShadowMaps**
  - Replaces voxel shadows for sun
  - Cached (tiled) EVSM CSM
- **Forward+**
  - Replaces all shadows and forces all lights to analytic







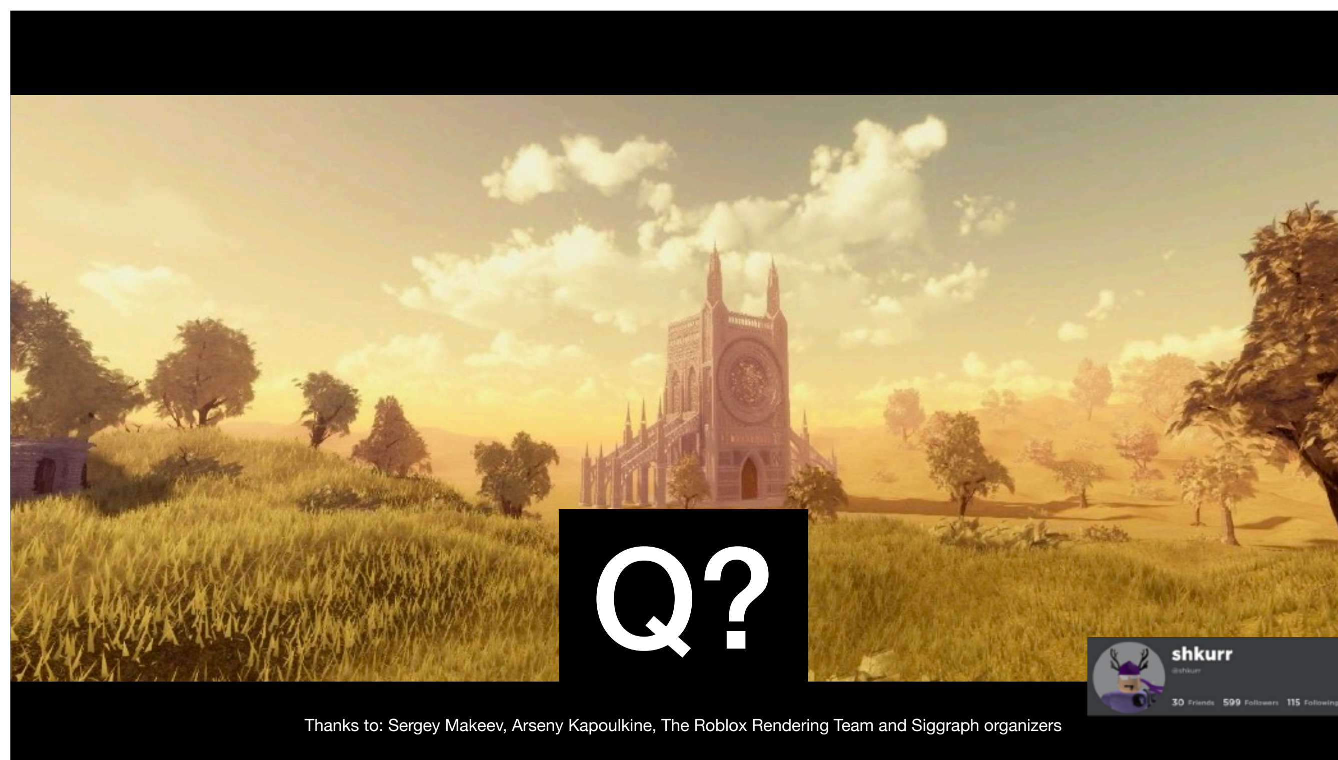
Shading
Analytic Lights

- All analytic lights use PBR
  - Custom approximation of GGX w/EC
  - …Spherical Gaussian approx for low-end
- From lowest to highest quality:
  - Voxel lights + Sun w/voxel shadows + Ambient indoor/outdoor based on voxel sky visibility. Vertex Shaded!
  - Move to PS, add Environment Cubes / Ambient, indoor/outdoor still leverages voxel sky
  - Add Sun w/shadow maps
  - Add F+ lights, all lights w/shadow maps, voxels only for ambient
  - Add Dynamic Probes

Conclusions?
Roblox so far

- Not "just" an engine for all platforms, but content/delivery that scales
  - A new idea of 3d content production
- Long view:
  - Make sure we do not paint ourselves into corners
  - Careful when adding new DM/API concepts
  - Design our systems so the delivery technology can change
    - Not just client, but server/cloud too
- Actual implementation:
  - Incremental computation, caching & sensible fallbacks

https://www.roblox.com/games/6396202508/Islands-Showcase-4-27-21



https://www.roblox.com/users/6583472/profile#!/creations