WHAT IS THIS?
A custom 3D open world game world loading, managing, and rendering solution. Its made in the Unity3D game engine but the project itself is very similar creating a game engine of my own.

WHO MAKES THIS?
I'm Kriсти Mäenna and I make this. I want to make my own Open World RPG and creating this world is taking one stepping stone along that path. I have been developing this project for 6 months with around 500 hours having been sacrificed for its development.

PROBLEM
The problem is that open world games worlds are big and loading the whole world in at once, if possible, is too slow. There are two widely accepted general solutions for this: The first one being to subdivide your worlds into levels and have only one level loaded at a time. Classic games such as Mario use this approach. The second way is to subdivide your worlds into chunks which are a collection of world data tied to a specific world coordinate. So then what you do is only load in the chunks that are close enough to the players coordinate. Virtually all open world games use a variation of this. Both of these solutions work well, but they each have drawbacks. While level based world system are very easy to create and manage they require some sort of loading transition to go from one level to another and that is not very open world like. The problem with a chunk based solution is that to get from A to B naturally you have to cross the chunks between A and B and that while being very realistic isn’t always fun. As a game designer I would like to have the freedom of the level based world system and to have the seamlessness of the chunk based world system without using any loading screens.

SOLUTION
My proposed solution to this is basically have chunk based worlds but instead of the chunks having their coordinate in global space they are defined by their relations to other chunks. And I call them nodes not chunks. So basically it is a node that can have a chunk of terrain data which you have to convert their relative transforms to the world transforms. This way the game designer can connect any nodes they want to create their world. But this solution is not without its drawbacks with the main one being that it is a custom solution and you have to implement it yourself. For most cases implementing this system will be overkill but if you want to have a lot of freedom in designing your open world game or distort loading screens as much as I do this system can be considered.

RENDERING TERRAIN
So each node can have some transforming data. When the nodes are loaded that transforming data is blended with other nodes transforming data and converted into a grid of terrain cells. Each cell has a bunch of values like height, color, texture and so forth. Then these cells are converted into planes where a plane is a plane mesh tied to 8x8 cells so that each vertex corresponds to a cell. Then the planes are rendered using instanced rendering. Instanced rendering is a way of batching together draw calls that use the same mesh and this can give massive performance gains over using a single draw call per object. When it comes to rendering the planes I ran into a lot of trouble because I can’t use the geometry shader step so I have to sample the cells per fragment instead of per vertex. My biggest mistake was to make the shader too much at foreach fragment I wanted the shader to blend nearby cell values, use trigonar shading, and have a bunch of cool things like emission and a height map. The way I did I had meant that foreach fragment I then had to do 4*4*3 = 144 texture reads to combine all the texture values together and this wasn’t exactly fast. So I decided to scrap most of the features and make a fast and stylized terrain rendering system. In the end I reduced the texture reads down to 6 per vertex — 3 cells and 2 textures per cell for diffuse and normal.

MULTITHREADING
Another thing that frequently gets me into the habit of bad code is coding for performance. I decided early on that this project was going to be multi threaded. This would allow me to keep my code simple because there would be enough performance to go around. So how my threading system works is that you have three parts to it. Part one is the thread pool, it does queued work once each frame, this is ensured by making the worker threads wait for the main thread to open a barrier when they have finished their work. The second part is the systems which have two states of execution — working and syncing. Working state is done on subthreads and syncing state on the main thread. The second part of the systems themselves are all systems that influence each other finish their current state. The third part is what allows me to do working state. So all systems can read the global data of other systems and on the sync state they write their local state to their global state. So this allows me to really easily create systems that can access each other’s data while being run on multiple threads simultaneously.

CONCLUSION
This project has been a wild ride with the rendering of the terrain planes and creating the editor system taking up way more resources than I expected but that just made it so much better. I didn't finish these. If you were somewhat inspired by this poster and want to create your own world thing use the Unity game engine then I would strongly suggest that you firstly plan out your project, secondly not underestimate how much time making a custom editor solution is going to take and thirdly remember that no matter what you do writing shaders will always be a pain in the ass.