AUTODESK UNIVERSITY

AE23563

Behind The Scene – Building The World of Deus Ex : Mankind Divided

Hubert Corriveau
Eidos-Montreal

Learning Objectives

- Understands the pitfalls of designing a navigable 3D interactive world
- Deconstruct an environment into lightweight visual modules
- Organize environment modules into large, memory intensive environments
- Understand the limit of current real-time technologies, and their workaround.

Description

At Eidos-Montreal, the art team is passionate about architecture and it shows in our games. Deus: Ex: Mankind Divided features some of the richest environments ever built in a video game. In this talk, you are invited to take a peek behind the curtain.

Hubert Corriveau, Environment Director will explain the process by which Eidos-Montreal typically builds an environment using Autodesk 3ds Max. He will dive into the details such as the challenge of designing an extremely dense, narrative driven and gameplay-heavy environment while trying to stay as close as possible to real-life architectural constraints. He will also approach how to deconstruct an environment into lightweight visual modules to be able to polish them by adding unique visual elements: clutter, decals and more. How to then re-organize the modules into large, memory-intensive environments, and finally the limits of current real-time technologies and their workaround.

Your AU Expert(s)

Hubert Corriveau is a computer artist based in Montreal, currently working in videogames. Hubert started working in architectural pre-visualization back in 1999, then transitioned to adventure videogames featuring pre-rendered visuals in 2001. He has worked at various studios including Gameloft, Microids and Ubisoft, and is currently the Environment Director on the Deus Ex franchise at Eidos-Montreal.

Understands the pitfalls of designing a navigable 3D interactive world

Why most game environments feels so different for their real-world counterpart?

Character Controller, Camera controller and their relationship with the environment.

Our game switch regularly from a 3rd person perspective to a single player perspective, meaning from a camera inside the head of the protagonist, to a camera about 1,5m behind the character. This distance varies from a game to the other, but in our case, that distance is 1.5 meters.

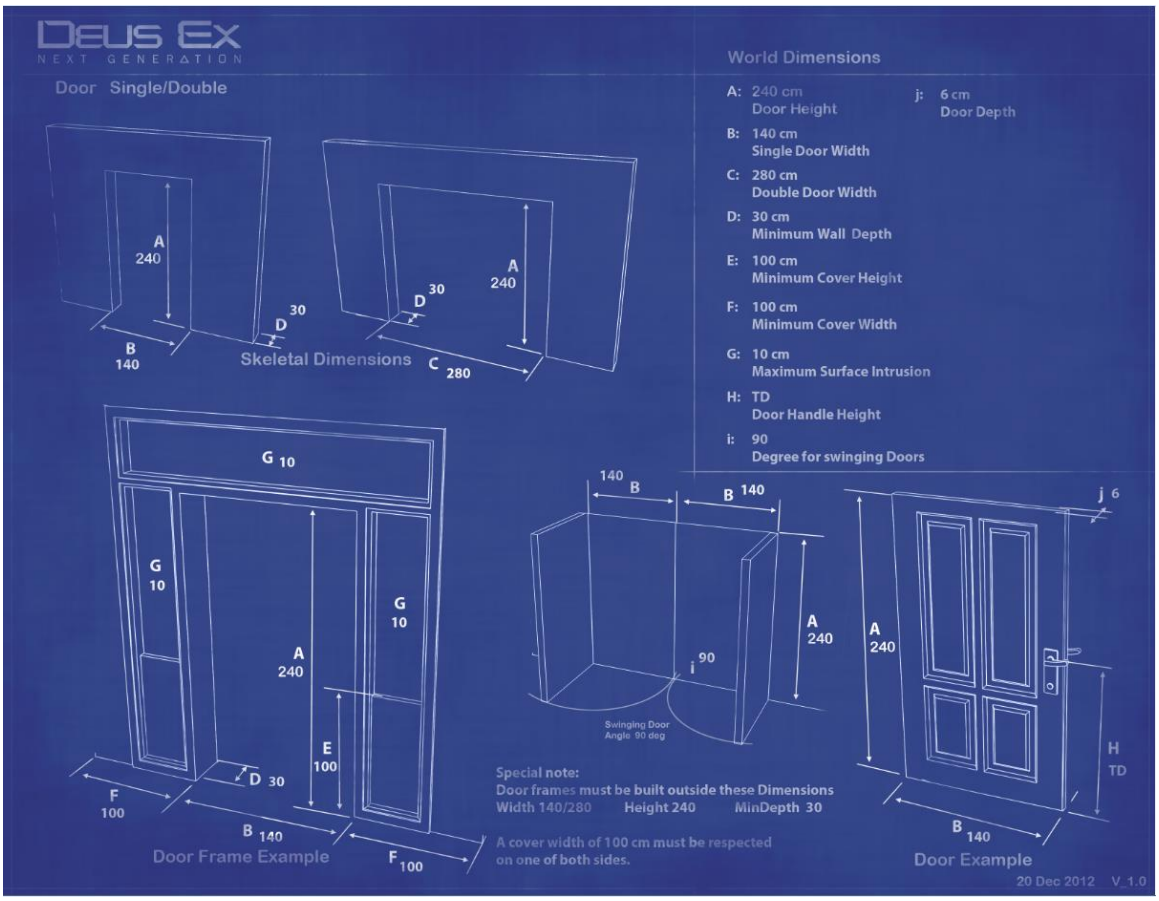
Our Character can also take cover on most surface of the game. This means he can hide behind walls and couch and car and be shielded for bullets, or shielded from enemy's sight.

This leads to a ton of rule set that have nothing to do with life architecture, because of course, in life, car and sofas are not designed to help an Interpol agent into sneaking his way through a bank. But as game designers, and environment designers in a game, we must design the environments with that in mind.

Another constraint is that not only does the player needs to navigate, but the character living in that world needs to navigate alongside the player. For example, where in life you would need stairs to be at least 36 inches wide to make sure 2 persons could reasonably meet in a staircase, cross each other and go on their way, in our virtual world, for various reasons, that stair width is over twice as large, at 2 meters.

This leads to mismatching dimension between interior and exterior environment

Because of those sets of rules, the free space on both side of a door needs to be at least one meter. And the door itself is 1.4 meters. The stairs needs to be at least 2 meters wide. Imagine a single door in your home. Imagine being forced to add 1 meter on each side, now propagate that change throughout the house.



EXAMPLE OF WORLD DIMENSION CONSTRAINTS SHEET WE PRODUCED FOR OUR TEAM.

Then propagate the extra width of the stairs.

Then, imagine a hallway needs to be at least 3,5 meters wide, because in our game, the Interpol agent need to be able to take cover behind a plant, have a 1,5meter camera behind him that “looks” at the scene.

Imagine that the space between the bed and the wall needs to be either 0 (right next to the wall) or 2meter (to accommodate navigation)

THE SMALLEST APARTMENT IN OUR GAME.

And imagine doing this in an apartment that needs to represent a futuristic dystopia. Where you would expect a poor person to live in a very small, undignified area.

You end up with a single bedroom apartment that is about 12 meters long and 12 meters deep. The kind of living space you probably cannot afford if you live in San Francisco or New York, but somehow, it's the smallest possible environment we can give to a "down on his luck, punk, low-life vagrant"

A SMALL, DIRTY, VERY LOW INCOME ENVIRONMENT IN DXMD STILL NEEDS TO BE VERY LARGE BECAUSE OF WORLD DIMENSIONS CONSTRAINTS

Exterior environments are generally easier to deal with; we are used to a larger frame of reference. But since the interior apartments are too big to fits the exterior façade of the building, we have to cheat, this leads to other complication.

EXTERIOR ENVIRONMENTS ARE EASIER TO MAKE AT A REASONABLE SIZE. .

Because of this mismatch, we cannot produce some environments

We spend a lot of time trying to make our environment as architecturally credible as possible. And Deus Ex's environment are meant to be a very large "Doll House" where we want the player to do what he wants to do. We do not hold his hand. The sense of freedom we give to the player is very important.

So, it is sad when we need to place a door somewhere, because there would be a door, but this door cannot be opened. And we don't like non-interactive door. It's a boundary that jumps in the face of the player. It screams "Here is where we stopped building this virtual world"

THIS DOOR IS ONE OF THE RARE "NON-OPENABLE" DOOR IN THE GAME.

In the above example, we are on the 3rd floor of an apartment building, visiting what we call a micro apartment. A single room apartment with the smallest possible area and a bathroom (not on the screenshot). These are not accessible by non-playable characters so, we cheat the dimensions to succeed doing such a "small" environment. Notice the door. It cannot be opened. Our agent went in there by jumping super high, grabbing on the balcony, and crashing through the window. There is no "normal way" for the player to go in this apartment. And we cannot make a normal way. That door is only there for credibility. This door is necessary because conceptually, if there is no door, how would the guy living there get into his apartment?

REMEMBER THAT AN HALLWAY IN MANKIND DIVIDED HAS TO BE LARGE ENOUGH TO LET PEOPLE NAVIGATE ACROSS EACH OTHER

In the case of Deus Ex, we would have the time to produce most environments that are hiding behind doors. But we cannot, because there is likely not enough room behind a given doors. The dimension required for this to work would mean that from the outside, a building would be at least twice as large as you'd expect it to be in life. We would not be able to build the staircase, and make it fit with the actual layout of the virtual city itself. And we don't want the city itself to be infinitely large.

The City Itself is not respecting the actual layout of the city for believability reason

We try to avoid screaming to the player "This is where we stopped building the world" and at some point, we have to stop building the world, because obviously we cannot build infinite worlds. Sometimes, in other types of games, the city is based on an Island, so the limitations are naturals.

Other times, they just build big walls with big invisible collisions.

We call these "world boundaries"

In our case… we try to hide the world boundaries. And we do this by looping the city's layout.

In Human Revolution, our previous game released in 2011, we tried to dress up the "world boundaries" to justify them. For example a street became a highway and went into a tunnel, and an eternal traffic jam was stuck there. This backfired, it doesn't make sense to see no car circulating ever, and then see a ton of cars stuck in the entry of a tunnel "for some reason" so everyone understood the real reason for this meaningless traffic jam.

THIS TUNNEL IS ETERNALLY BLOCKED BY TRAFFIC

By dressing up this world boundary, it actually screamed louder to the player “this is where we stopped building our world.

In Mankind Divided… there are no “end of the roads”

Every street you take will loop back unto another street. You will never get to an “end” you will never get to a point where it says” here is where we stopped building this!!!
Conceptually this makes no sense, as it means there is no way in or out of the city. But this is less “obvious” to the player because he never sees a hint screaming that to his face. And as such we went with that.

The suspension of disbelief is better protected this way.

THIS ENTIRE PLAZA IS A WORLD BOUNDARY, BUT THERE IS A FULL MAP HIDING AT THE END OF IT (THE INTERIOR OF A BANK)

The city is not constructed based on an actual layout of the city, because this would be too big a constraint to the level designers, who are tasked with creating the challenges and scripting the missions for the players in that city. So, we just accept that we are doing a fictional city, and Level Artists tries their best to convey a sense of architectural credibility to make it "believable"

Deconstructing an environment into lightweight visual modules

There are more environments than Prague in Mankind Divided, but by far the most complicated environment is Prague. The density and the size of our virtual Prague makes it a significant technical challenge to achieve on today's hardware. So, we'll keep with that example along the way.

Anything that can be repeated, should be repeated.

Architecture generally have a whole lot of repeating patterns. Even modern parametric architecture usually revolves around modifying pattern with a given algorithm. Today, those algorithms are already getting complicated enough in life to say we can't repeat them easily as mathematical constructs inside a game engine. But if we are doing a large enough environment, we cannot simply save every single building as individual, custom meshes that have been generated with offline algorithm. We need to figure something out.

Vertices cost memory, and we don't have that much memory, even on current gen console. Everything that is easily repeatable should be saved once, and repeated as much as we need.

Dividing a building into sub meshes

This is a vanilla façade that we often use in our city, with how it's being subdivided in meshes.

VANILLA BUILDING

INSTANCED ASSETS

We call this instantiation; this is common in the industry and is quite like the “reference” function in 3DS Max. Each of those meshes are saved once in memory, and then only the different locations of each “instances” are saved in memory. Given that a single “column” on this façade is pushing about 1000 vertices, each with their saved location, indexes and normal information. It is much better to repeat this set of data 5 times by only modifying the base location, and then reapplying the very same set of data at each instances location.

Then we push this a bit further, like I said, it’s a bit like the references in max. It’s not quite like an instance. In our engine, Dawn, we can output any single parameters of pretty much any kind of resource to be modified at runtime, at any point in the game. We use this to modify each instance. So, we do keep the same set of data, then we simply modify one parameter to change something. Anything. The color of the diffuse, the actual texture diffuses inside a material, the way it reacts to light depending on if it’s wet or dry outside.

MODULE VARIANTS

NIGHTTIME VARIANTS

Then Rendering pushes this a bit further. Not only is this saved only once in memory and reapplied multiple time inside the environment, but we are doing what we call “instanced

rendering", where we agglomerate all those instances inside the same Draw Call. This is a significant optimization that is entirely credited to the engineers. In the end, this is how we fit 13500 draw calls inside a more possible limit of 4500.

EXTREME CLUTTER LIKE THIS IS POSSIBLE BECAUSE OF INSTANCE RENDERING

Apply this kind of logic to sets of walls for an interior environment, basements, floors and ceiling. Usually in games, one set of very hard rules are decided upon, which needs to respected, absolutely, for any environments in the game. Like the blueprints we saw above, but meant for "how modules are separated". An easy example is "walls must be divided by either 4m, 2m, 1m, 0,5m and curved assets needs to be based on a circle with a radius of "x" and divided by 45 degrees, 30 degrees, 20 degrees, 10 degrees, 5 degrees.

THE WALLS AND FLOORS OF LONDON ARE ALL MODULES BASED ON A BASE 50CM GRID.

You typically define one, and reuse this set of rule as much as possible. On Deus Ex we don't do that, each environment will likely require its own rules, the level artist will figure out how to best build his environment. It needs to be modular, but the artist could decide walls of 5meters are better suited for this specific environment. And we are not scared of doing custom meshes to "make it fit" but we need to do this when needed, meaning, when 2 sets of modules with different rules meet.

The reason for this willingness to not restrict ourselves to one single rule is that we want architecturally credible environment, and one more set of rule that has nothing to do with life would break it apart a bit more. We do respect rules, but dependent on the given environments requirement. For example, a building's floor height viewed from the exterior, in Prague is 5-meter-high… which is different to the typically expected "base 2" 4 meter high.

Organize environment modules into large, memory intensive environments

If we look at the city from a higher point of view. You can see how we make the streets loop on themselves. If we look at the map of the city, you can kind of see that, at first, the streets where modular, built out of a hexagonal set of tiles. The buildings also had "modular footprints" meaning that even though buildings were built out of modules, they themselves are also modules of a city. This is less obvious on the final, released map. This changed over time because we customized the map heavily during the polish phase, but it still served as a basis for the map creation. This means we could propagate this recipe over a somewhat large area.

THE CITY HUB'S STREET LAYOUT WAS AT FIRST BUILT OUT OF HEXAGONAL TILE, THEN HEAVILY CUSTOMIZED TO FIT LEVEL DESIGN REQUIREMENTS.

But then, we are doing Deus Ex, and in this game, we want a lot of storytelling. The map is filled with bits of story everywhere. Our agents jump on a Micro Apartment, hacks a computer, and reads a story about how a kid lost his mom in an explosion, and is now on the verge of being deported because is dad is augmented and already deported to Golem City. This speaks of the theme of Deus Ex Mankind Divided.

Now, when we walk on that Micro Apartment, we do not see the kid, we only read the mails, and the apartment itself. We see toys on a mattress on the floor… with a box of cereal half opened. You can see in your mind, the kid playing with the train, begging his mom for his cereal box, her mom giving him the cereal box, and him going back unto his mattress, crashing there while looking at the TV and eating crunchy pirates.

You can almost get the sense that for him… life was comfortable until the day before.

I BET WHOEVER SLEEPS ON THIS BED LOVES PAW PATROL!!!

This is completely irrelevant to the game itself. The player is never required to go there; he's never asked by anyone to go there. But we know players explore, and those that explore will see dozens upon dozens of little sites like that throughout the game. Where something is shown, and vaguely told through mails, to the player. It builds on most of the "universe" of Deus Ex, give a sense of coherent world, to a much greater extent than most games out there.

A KINDERGARTEN IN THE VERY SLUM-LIKE GOLEM CITY ENVIRONMENT.

INSIDE, YOU WILL SEE A WOMEN WHO HAD HER KID REMOVED FROM HER CUSTODY, A BUNCH OF DOLLS BUILT OUT OF COAT-HANGERS, AND ROBOT THEMED CHILDREN CEREALS.

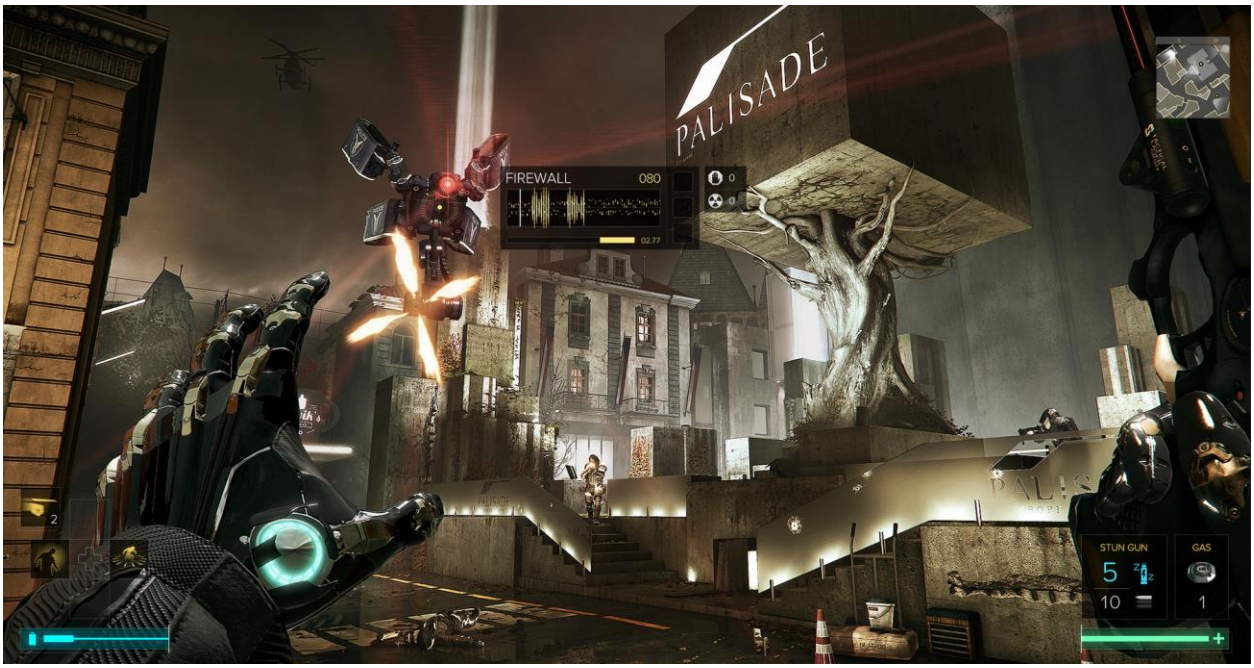
We want those moments, as much as possible, and we crammed Prague full of those tidbits. Anywhere you can just look around, you will likely find something we wanted you to recognize.

This cost memory. More than the console has. And there, we need to do dynamic loading and streaming of assets.

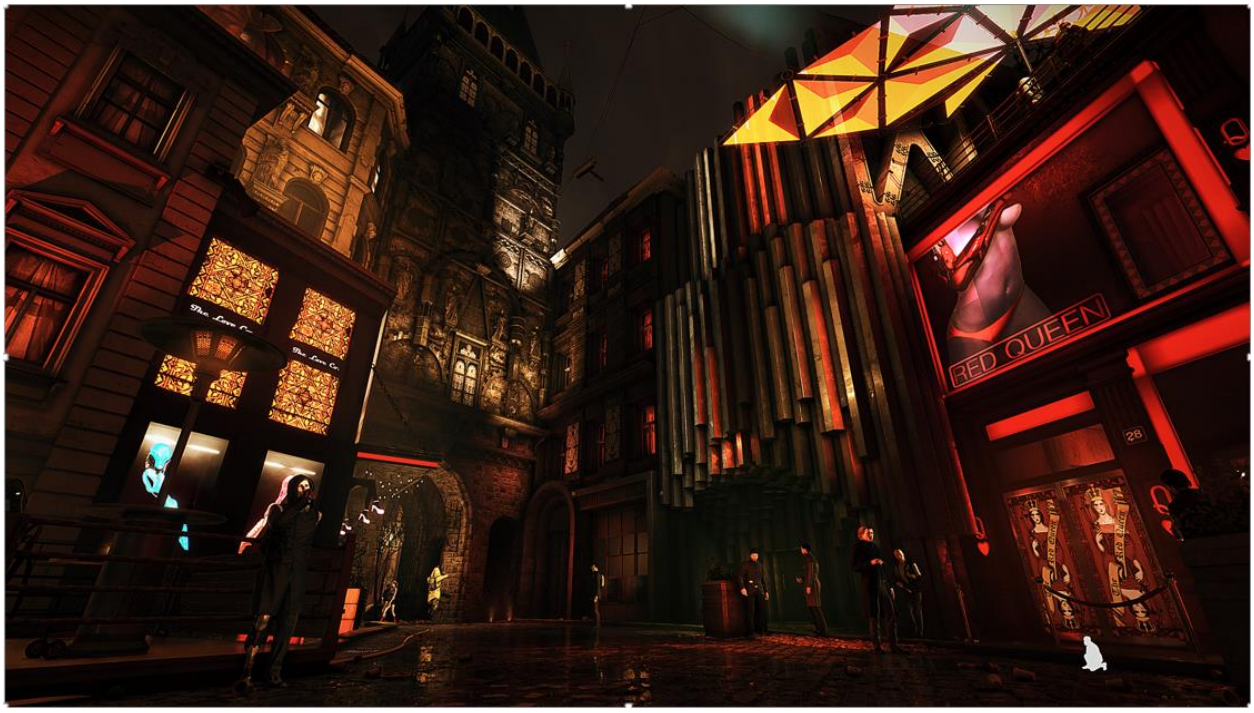
Dynamic Loading

If you go from one end of Prague to another, that theater you saw 4 minutes ago has been flushed out of memory, to make place for this bank. And if you go back, the bank must be flushed again, and the theatre must be reloaded.

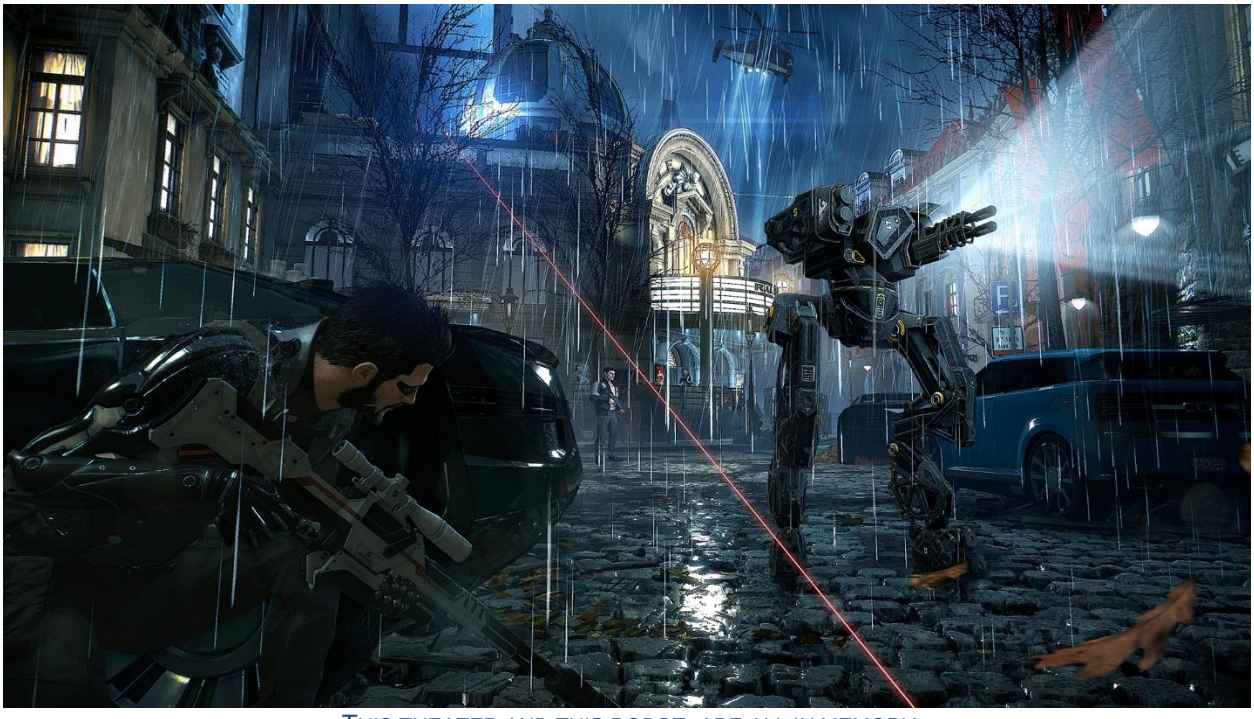
MAP OF ONE SECTION OF PRAGUE. THE LOOK OF THE LEFT HAND AREA AND THE LOOK OF THE RIGHT HAND AREA ARE QUITE DIFFERENT.



THIS PLAZA, THIS DRONE



THE RED-LIGHT DISTRICT, THOSE CHARACTERS



THIS THEATER AND THIS ROBOT, ARE ALL IN MEMORY

This is all managed without the player knowing. Similarly, there is around 80 characters being “simulated” always around you, but a whole lot more than that in the city. What happens when you walk away from the theater, all the characters around it gets saved at their current position, and then flushed out of memory, to make place for those at the banks, then we

simulate those. Because we believe you are heading to the bank. But the player could very well stop right there, and walk back to the theater. And it gets more complicated when you think that there is around 9 different location of interest like that which may get flushed out of memory.

What is Streaming?

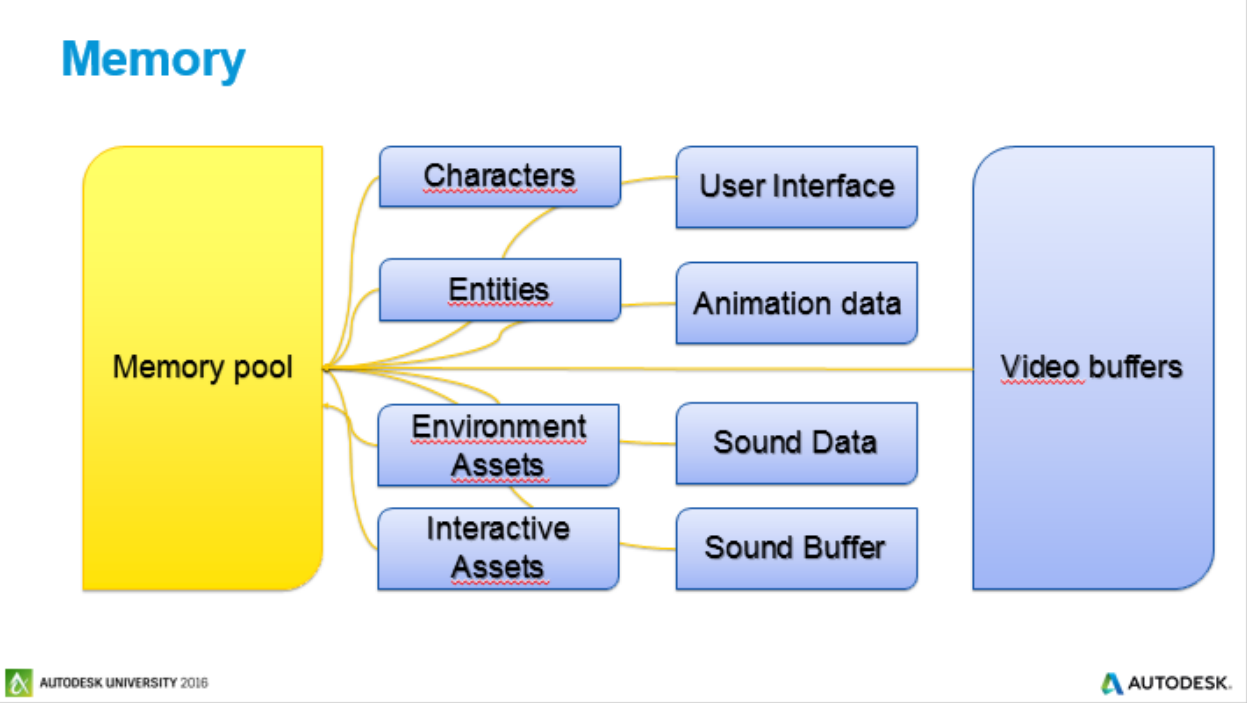
Streaming is quite akin to what you would expect with a YouTube video. But instead of streaming videos, you stream a bunch of individual textures and individual meshes and sets of sound files for all those character that talks and even stream video files being displayed on a TV in the game. There is a limit to how much can be streamed away from the hard drive each second. This bandwidth is limited and in Prague, we are often reaching that limit.

Understand the limit of current real-time technologies, and their workaround.

As far as visual representation is concerned in the gaming world, we have 3 main resources, which are limited, we must deal with; Main/GPU memory, GPU workload and CPU workload. Each have their different “raison d’être”

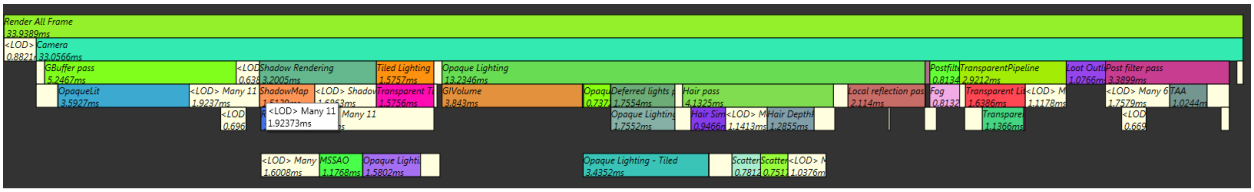
What are each constraint referring to; Memory

In layman’s term, **the memory** refers to how many different assets you can stock inside the running console at a given time. Each piece of mesh and textures requires memory space, and everything on display, or potentially on display at a very quick notice, needs to be stocked in memory. Every single piece of road, sidewalk, architecture, characters, weapons and other interactive objects… the pause menu, the inventory screen, all of this needs to be stocked in memory. Go beyond the available memory, and the console crashes.



What are each constraint referring to; GPU Workload

The GPU is the graphic processor; it oversees drawing everything you are seeing on screen. It gets fed a bunch of input from the CPU, it has access to all the memory, it takes all this and crunches thousands of little jobs we call draw calls, all rendered as various buffers. The GPU then composite all those buffers together and displays them as a single, polished image. All this for every frame, and a frame lasts for 33ms (or 16ms if you aim for 60-fps, but let's keep to our 30-fps example here). Too many draw calls, and any one of those buffers being a bit too heavy with data, and your games slows down to a crawl.



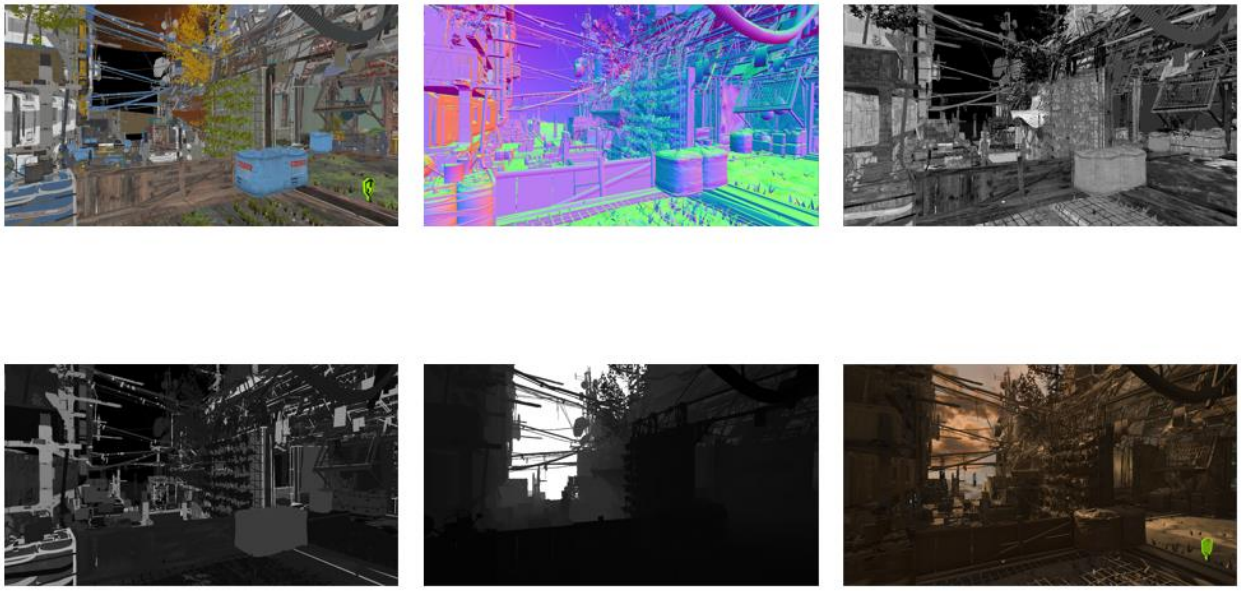
MULTIPLE BUFFERS OF THE SAME IMAGE BEFORE BEING COMPOSITED AS A WHOLE

Draw Calls and Buffer

Draw Calls are a single primitive (a fragment of mesh) that has a single material applied on it. There are multiple ways to optimize that, in our case, we batch each primitive that are similar as a single draw call. We typically render about 4000 draw calls on the XB1/PS4 on a single frame. (33ms)

ONE OF THE FEW SCENES WHERE WE GET REALLY CLOSE TO OUR 4500 DRAW CALLS LIMIT

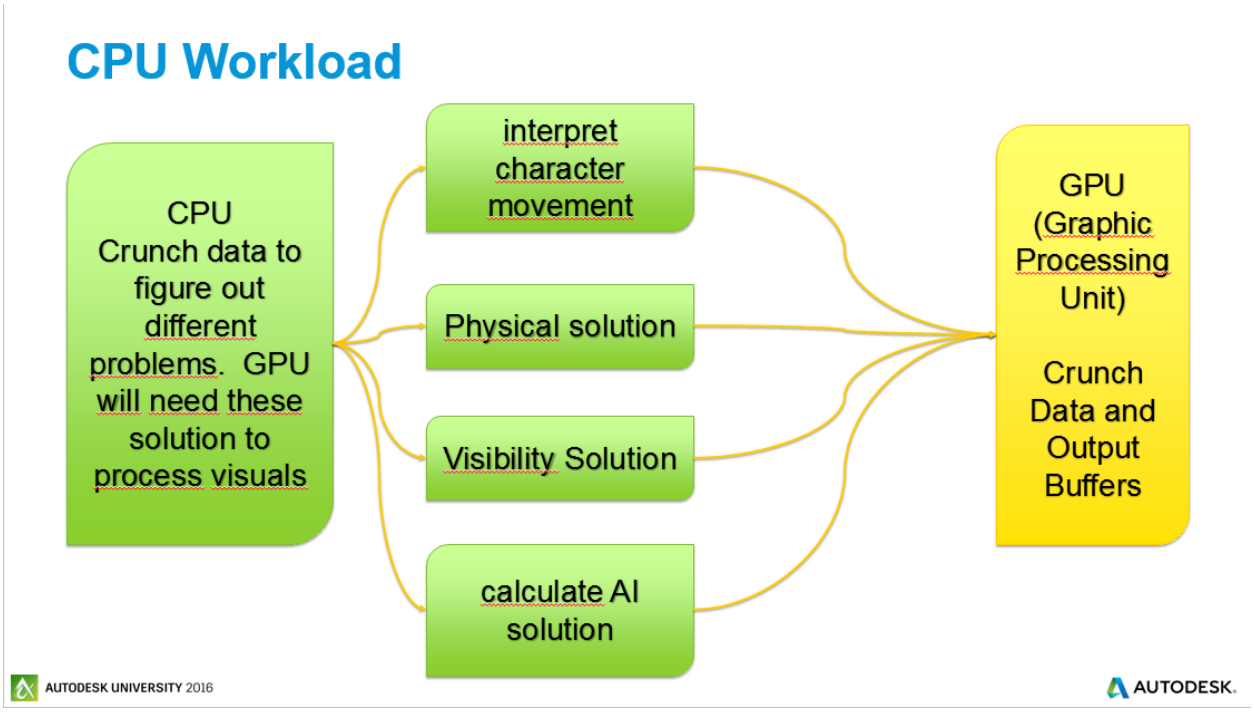
Buffers are like layers in a compositing software. Imagine having an albedo pass, a depth pass, a shadow pass, a specular pass, and compositing them all together to have a final image. This is what happens in a GPU, it must do this compositing work (with many other buffers) in about 4-5 ms.



MULTIPLE BUFFERS OF THE SAME IMAGE BEFORE BEING COMPOSITED AS A WHOLE

What are each constraint referring to; CPU Workload

The CPU has many jobs (and we are now dealing with many CPUs working in parallel) Not all of them have anything to do with visuals but all those jobs will end up affecting visuals in one way or another.

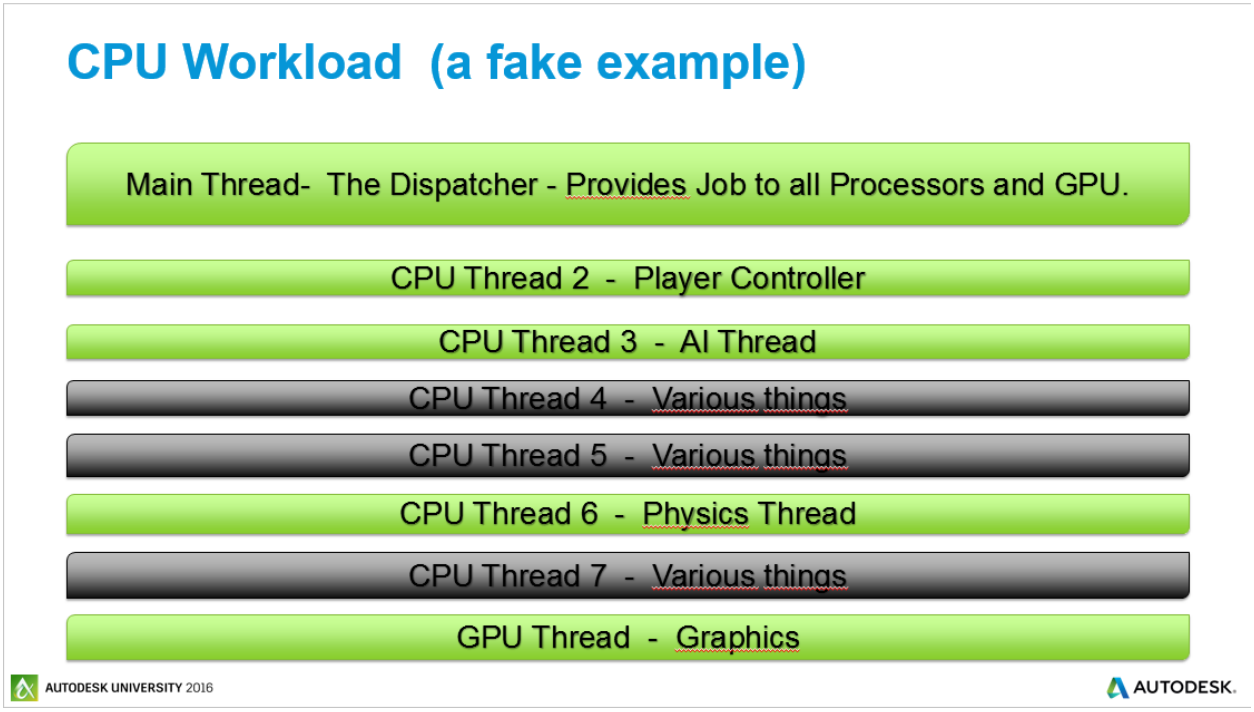


THE CPU CRUNCHES A LOT OF MATH TO PROVIDE RESULTS TO THE GPU, SO THE GPU CAN START HIS OWN SETS OF MATH...

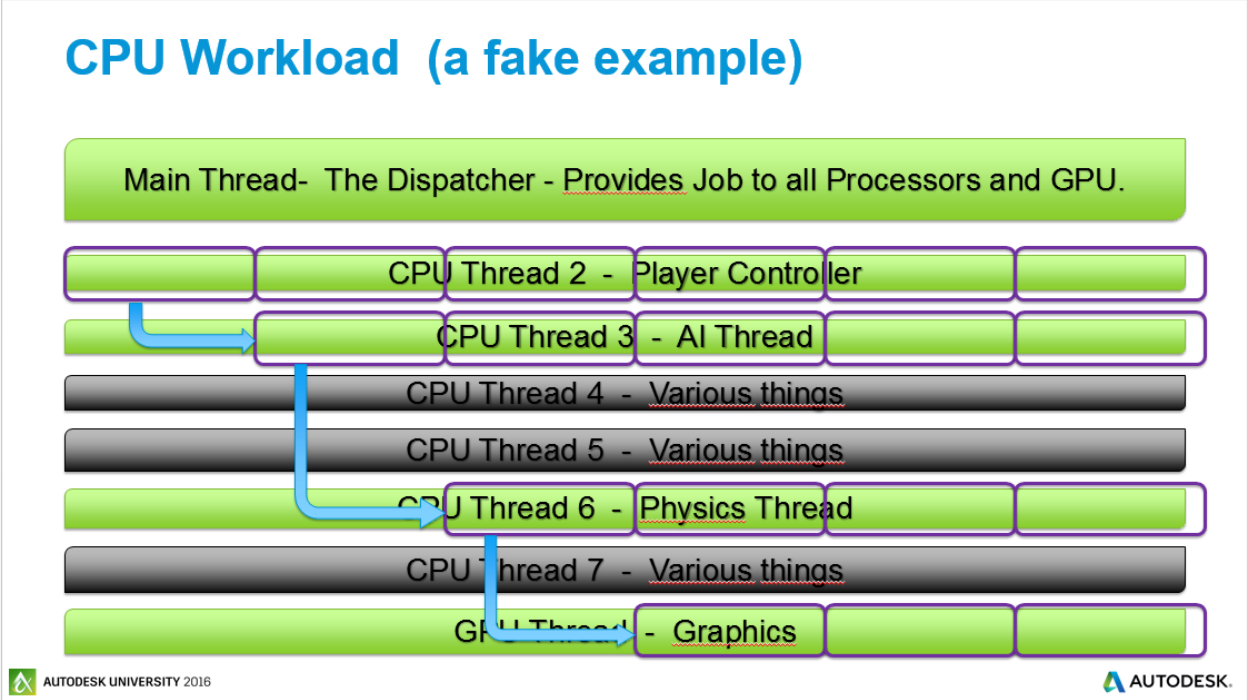
For example, AI and Physic needs to be resolved for the CPU to know where the character and the physical crate are currently located, so that it can pass that information to the GPU (which will render the character and the crate).

If the CPU needs to resolve too many character AIs, or too many physical crates, or too many particles simulation, their results will not arrive on time to the GPU, and the entire game will come to a crawl.

There are many processes in parallel here, and many possible points of failures.



More than that, the CPU crunches many math problems at once, in parallel, BUT a certain order or process is still necessary.



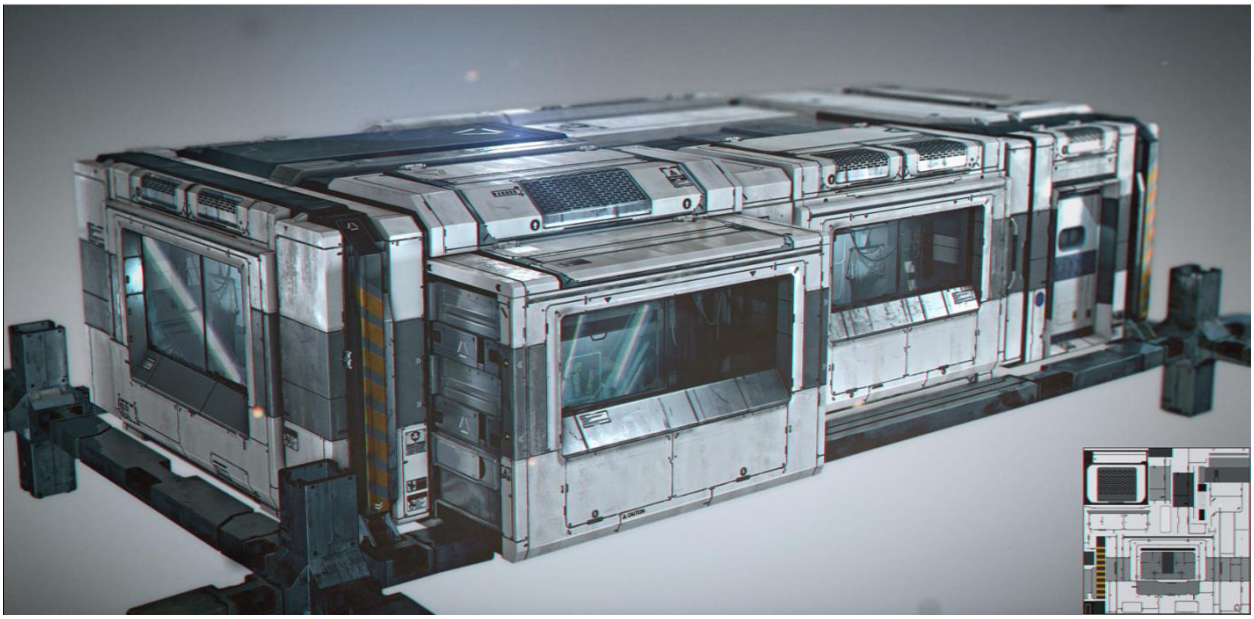
And since everything need to fit inside 33 MS, we can actually stagger the process over a few frames. If there is 2 or 3 frames of delay between a button press and the result on screen, a human can't really realize it (we have about 120ms of reaction time)

The Workaround; Many Lightweight Modules, Template data organization, and Compressed texture mapping.

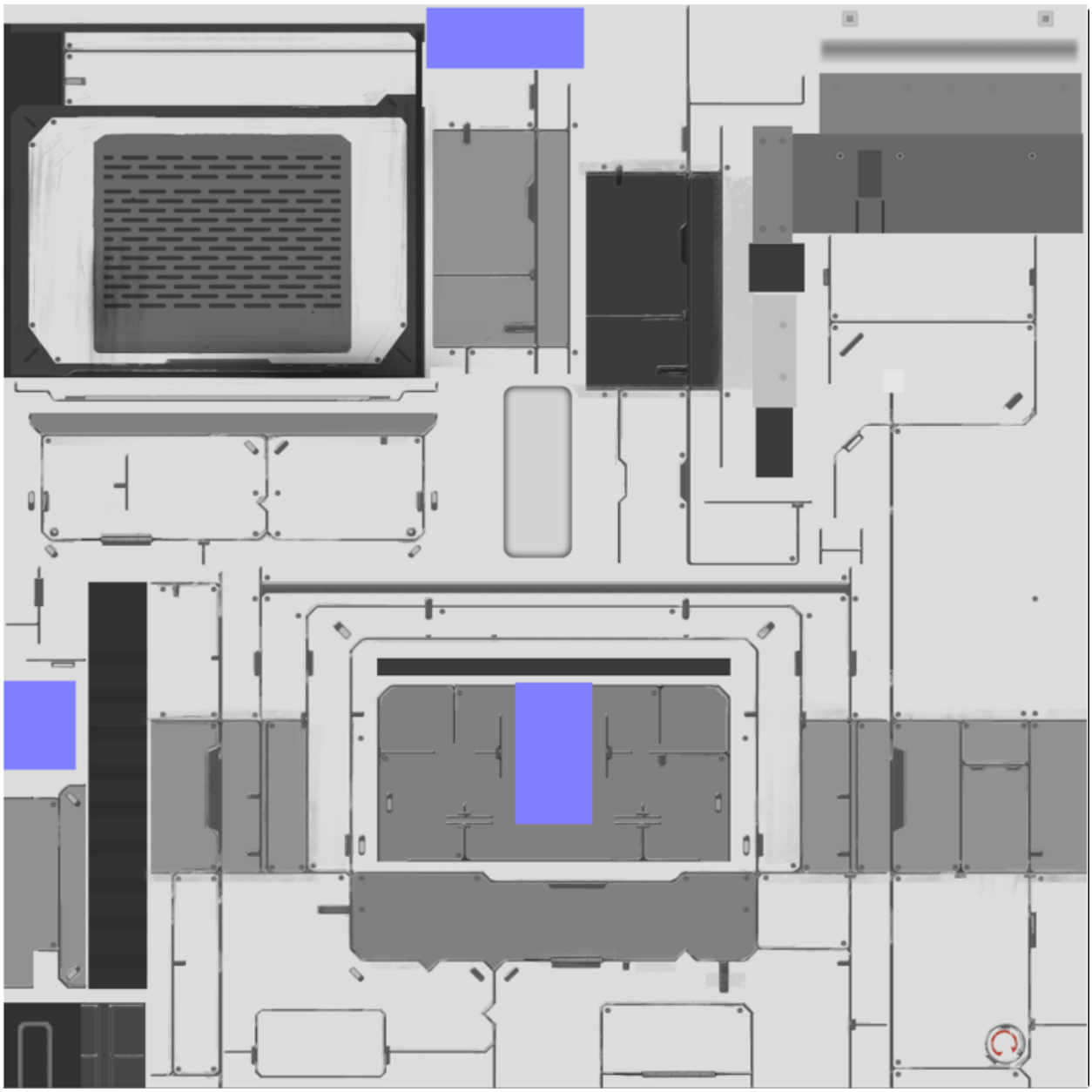
In the case of Deus Ex: Mankind Divided, we have a significant number of parallel features, a lot of choice offered to the player, which the player can enact when he wants, and a mostly non-linear structure. We are stretched very thin on most resources and need to use some workarounds. Other games may not have to do as many sacrifices to achieve the same visual quality as what we are achieving.

Other than the template Data Organization and the lightweight modules, which we went in detail above, the compressed texture mapping is probably our biggest way of saving memory.

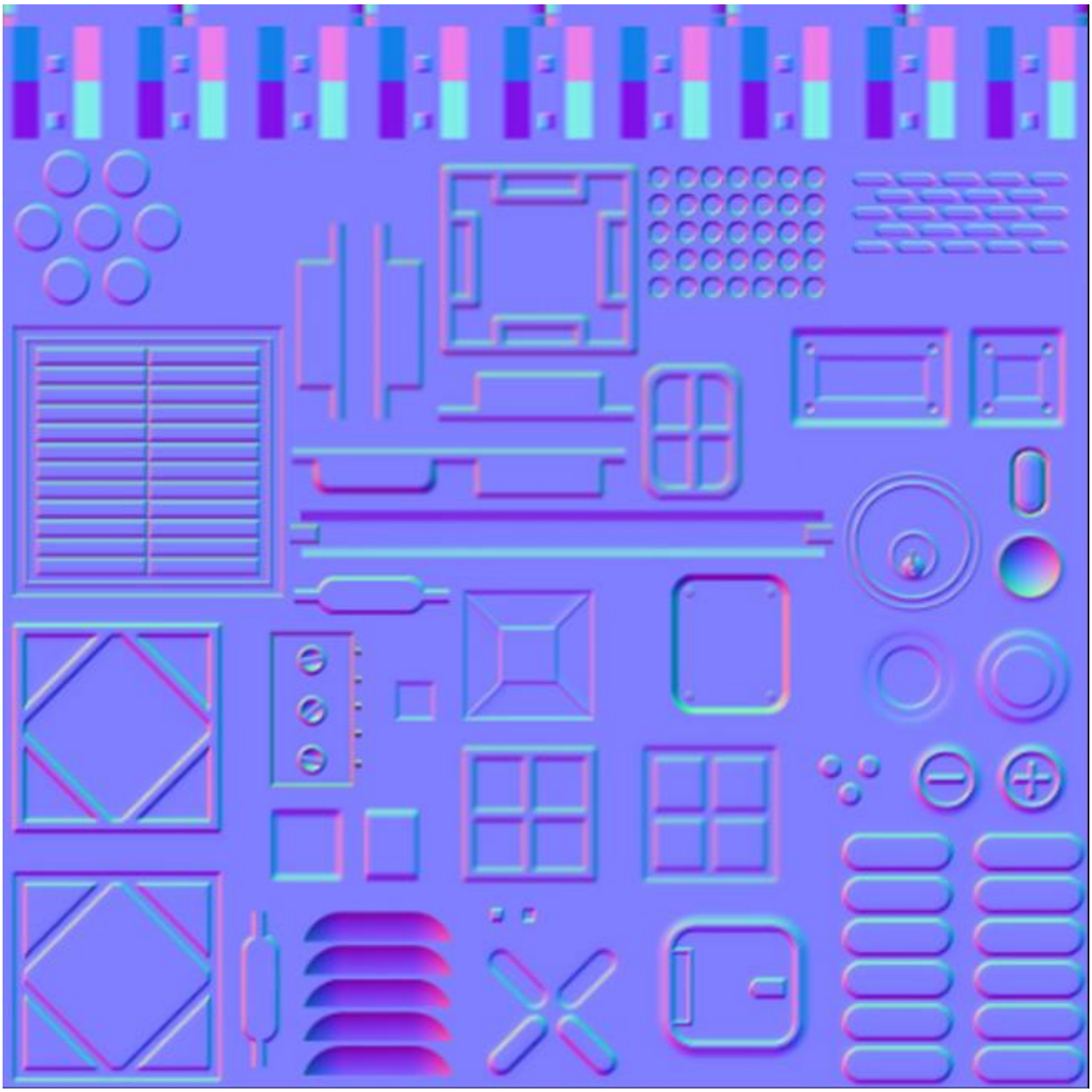
What we do here is, we use 2 sets of UVs for every single asset which requires specific details, at specific location. 1 UV is meant to map those specific, generally mechanical detail like construction lines, vents, bolts, stickers, latches, etc… at one specific location. And a second UV, more evenly mapped out, is used to apply the "material" details, meaning the wood grain, the metal scratches, the rust, and other, more tile-able visual elements.



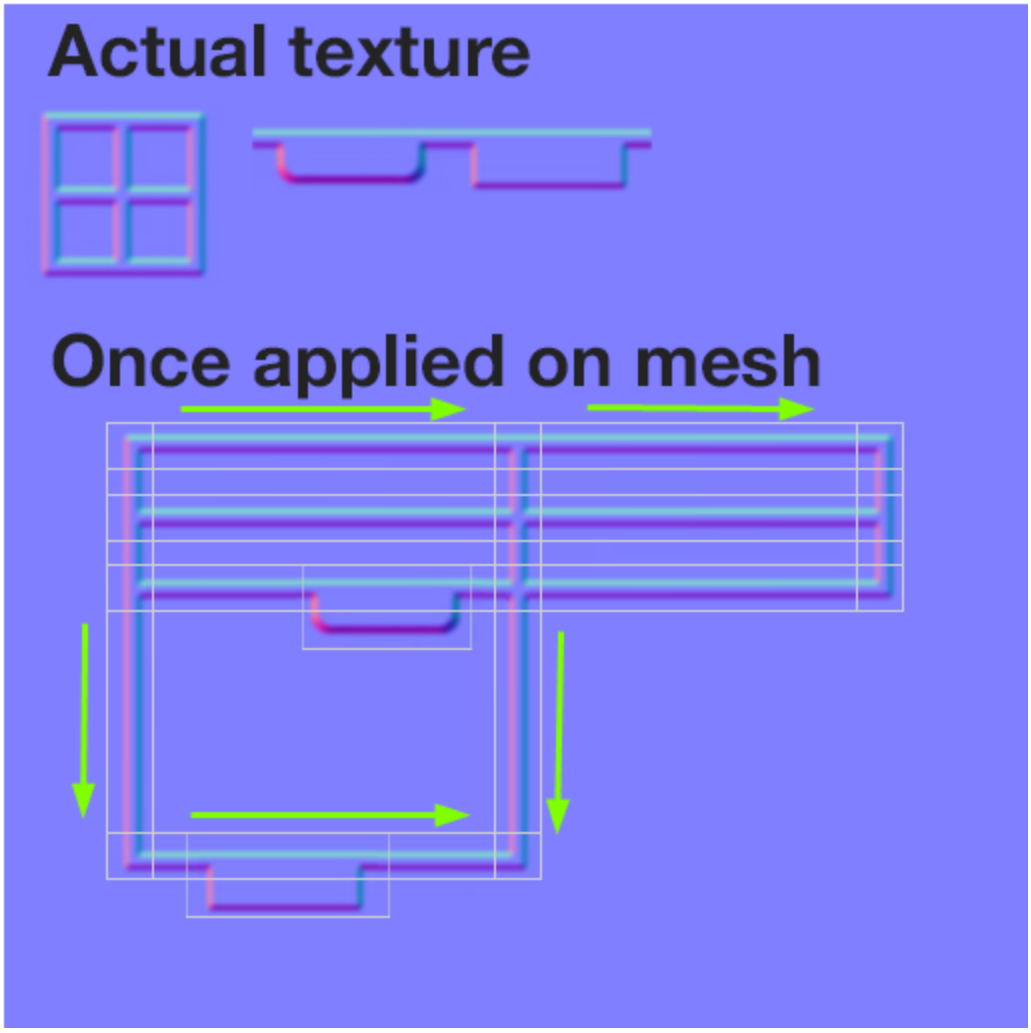
GOLEM CITY'S HOUSING BLOCK, THE MAIN BUILDING BLOCK OF GOLEM CITY, MEASURE 8M X 16M X 8M



THE ALBEDO TEXTURE OF THE HOUSING BLOCK. WE STRETCH MECHANICAL LINES AS MUCH AS WE CAN TO COVER ALL OF THE SURFACE OF THE LARGE HOUSING BLOCK



THE MAIN HENGSHA NORMAL TEXTURE. ALL THE ASSETS OF THAT CITY WERE REFERRING TO THAT SINGLE TEXTURE



AN EXAMPLE OF WHAT WE MEAN BY TEXTURE MAPPING COMPRESSION. WE USE TINY BIT OF TEXTURE, AND STRETCH IT ON THE GEOMETRY AT A VERY GRANULAR LEVEL. PROPAGATED THROUGHOUT A CITY, THIS IS EXTREMELY EFFICIENT MEMORY WISE.