Investigation into modular design within computer games.

By Scott Jones

BSc (Hons) Computer Games Design

A project submitted in partial fulfilment of the award of the degree of BSc(Hons) Science from Staffordshire University
Supervised by Stuart Butler
May 2011

Faculty of Computing, Engineering and Technology

Aprox Word Count 13000

Abstract

Computer games are ever increasingly becoming more detailed, however this detail is limited by two factors, time to create the higher detail and also the limitations of computer hardware.

One way to overcome this is by using modular assets to build the games levels from, this research paper looks into how modular assets could potentially help to solve those issues.

The goal of the paper is to find out just how beneficial a modular scene is in comparison to a non-modular scene testing performance to see how much more detail can be pushed and also workflow to find out if time can be saved to be able to put the extra detail in.

Content

Content

1 Introduction

This paper will be focusing on the use and implementation of modular assets within computer games, with the end goal of taking the knowledge researched to produce a game-ready level using modular assets.

Modular level design within computer games is already a popular method of creating interactive environments, this can be clearly seen in such games as the Unreal Tournament© series, Mass Effect© series and Halo© series amongst many more, these mentioned games have sold over a million units each based upon VGcharts (2011) with them all using modular design throughout their games.

Fig1.

Fig2.

The reason games such as the above use modular assets are because there are many benefits in doing so, these include but not limited to performance increases, production time reductions and the fundamental way assets are created, these are the main areas of research and discussion that will be presented within the paper however there are many other reasons that will be mentioned throughout to hopefully give a complete and clear insight into level modularity.

Along with these benefits there are some problems using modular assets relating to workflow and visual aesthetics Eipc games (2011) Perry (2011), these will also be discussed to give a fair representation to different methods and ideas.

The aim is to provide an understanding of the full effects of modularity, where and when to use them and also in what circumstances they should be avoided.

To accompany the research covered in the document a practical example will be demonstrated to verify the research, this example will be conducted in Epic Games' UDK and also using Autodesk's 3ds Max and Adobe's Photoshop for content creation.

2 Research

2.1 Modularity

2.1.1 What is modularity?

Epic games defines modularity as (2011):

> "*modular design is concerned with making lots of high-quality chunks of levels and reusing those chunks intelligently*"

The word modular itself comes from the word module which is defined by oxford university press as (2011)

> *"each of a set of standardized parts or independent units that can be used to construct a more complex structure, such as an item of furniture or a building."*

In relation to games, modularity is about the creation of an asset that can be used repeatedly with itself or another modular asset. There are many factors that make an asset modular some of which do not always apply to every circumstance, in section 2.7 different artists methods and views on modularity are discussed.

2.2 Benefits of modularity

There are many beneficial aspects of modularity, this section will cover the main pros and cons of modularity in the different stages of game production and also what effect it has on a game itself.

2.2.1 Workflow benefits

As modularity is the process of re-using the same high detail assets it means that a minimum of only one asset is needed for modularity to work and could potentially be used to make a vast environment, while this may not be visually appealing it does mean that less time is needed in the production of assets for such a level due to the actual quantity of work needed compared to an environment that is populated with multiple unique assets.
If modularity is not taken to such an extreme, in many cases it's still faster to produce a modular level compared to that of level made from unique assets of the same quality and detail, Perry argues (2002)

> *"The modular level design solution arose from the need to have great-looking, high-detail levels without having to build and texture every nook and cranny of the environment"* and also goes on to say *"modelling and texturing an entire level at extremely high detail levels can result in many months if work that may end up needing significant retooling"*

However in some cases this may not be true as Epic states (2011)

> *"If you are just making one room, then a non-modular room may be much faster to create than making separate pieces."*

While this case will be rarer due to the fact most games do not solely focus on only one room or small unique environments, this may still come up from time to time.

On the same note there are also issues relating to the constant re-use of the same asset, Epic games (2011) explains one such problem

> *"A major concern with making modular pieces is the fear that the player will be able to see the re-use of the chunks"*

This is an inherent problem as this is the nature of modular assets. Epic games (2011) suggests using unique attributes (lighting, decals, elevation etc.) to overcome this issue.

In short anything that will stop an area looking repetitive fixes the situation, normally easy to overcome and is fixed in the process of making a normal level as things such as lights solve this issue and lights are in almost every game in some form or other. Examples of methods to overcome these problems will be given in the practical section of the document to determine how much of an effect this can have

Epic games have examples of these methods within their level DM-Deck in UDK, below are renders from this level to demonstrate these techniques in use.

In this render Epic have used props to add variation to the section

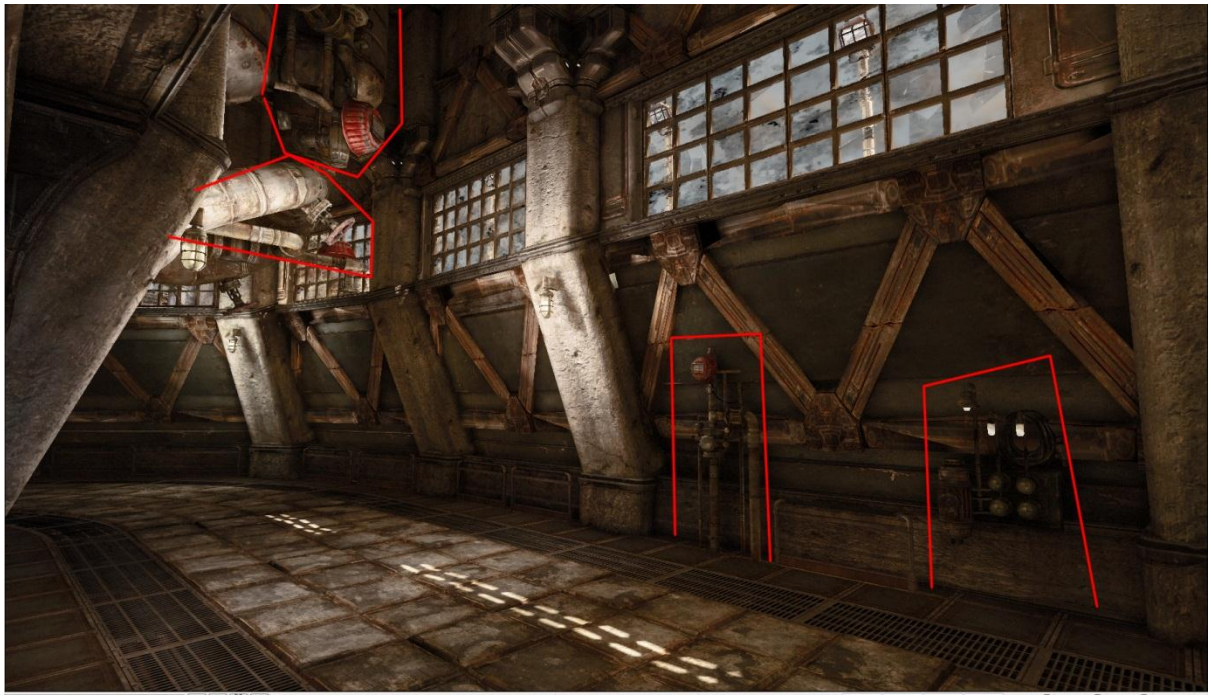


Fig 3.

The Render below demonstrates how lighting has been used to give the illusion that the corridor is not made up of the same pieces.

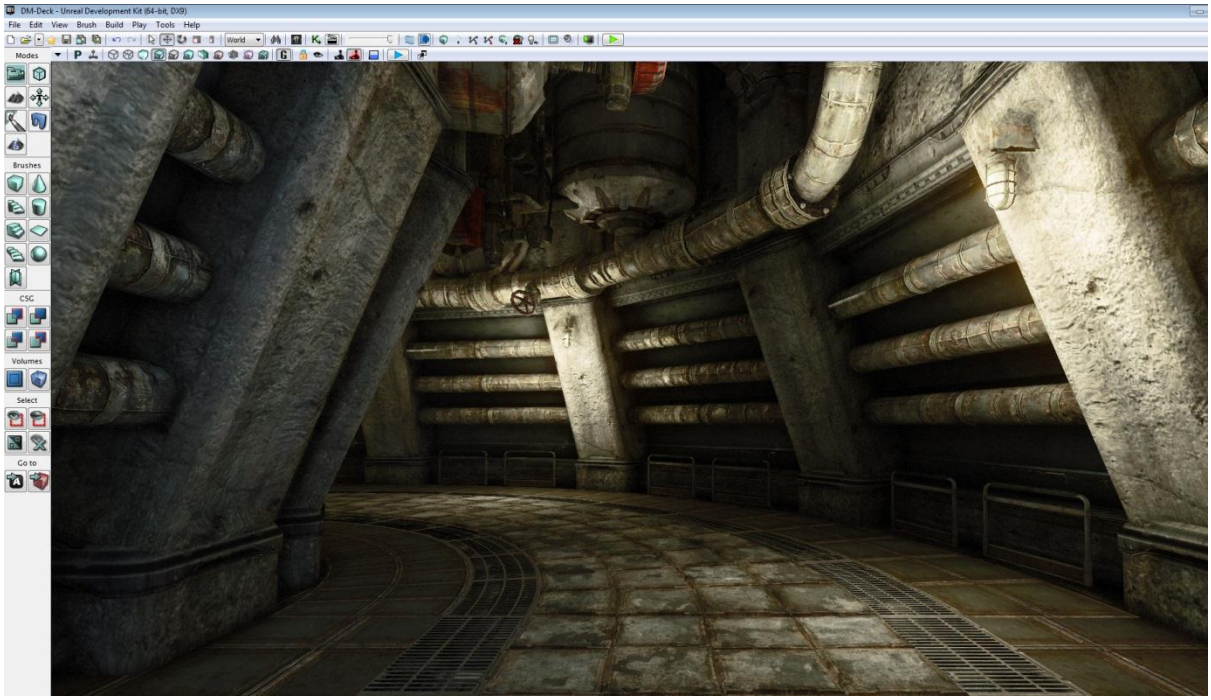


Fig 4.

Below is the same scene with no lighting information, it's a lot clearer to see that the same object has been used repeatedly.

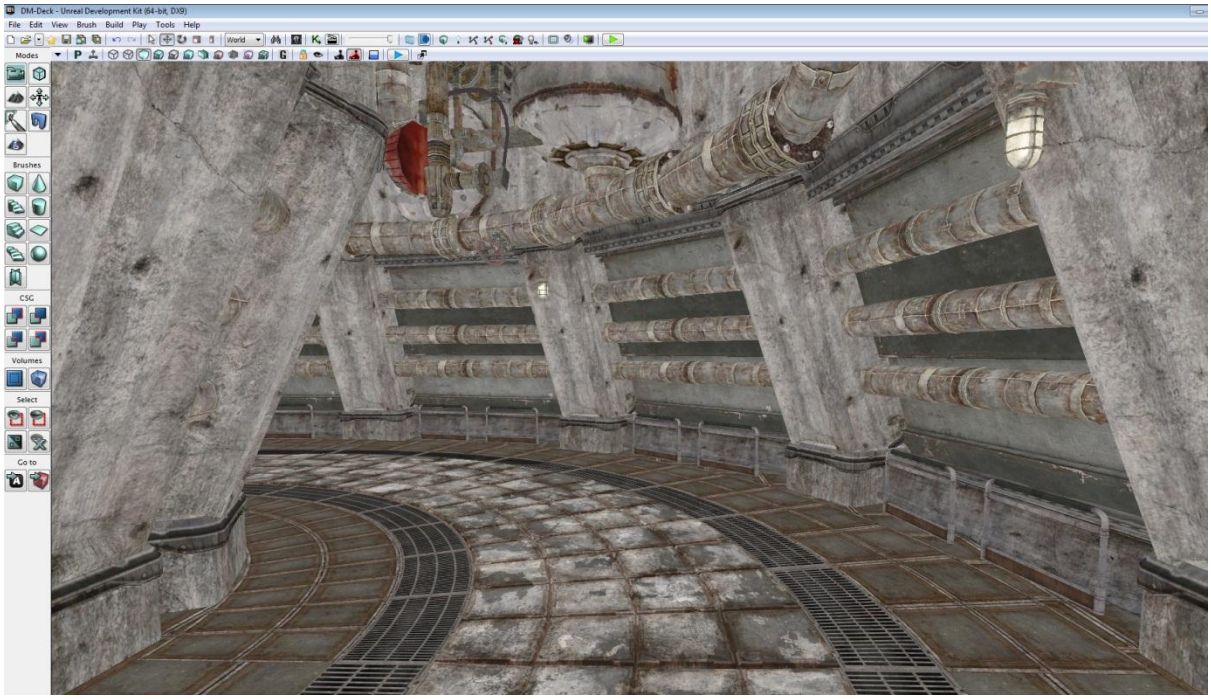


Fig 5.

In the same level they also use height to give the level a unique feeling, the walls and the floor are the same mesh in both areas, the lighting adds some difference along with props, however the areas read different because of height too.

If a player changes height they know they are going to a new location, instead, if the player goes through a series of paths and come to an area that is similar looking depending on the paths it may seem as if they have been there before and possibly causing confusion.

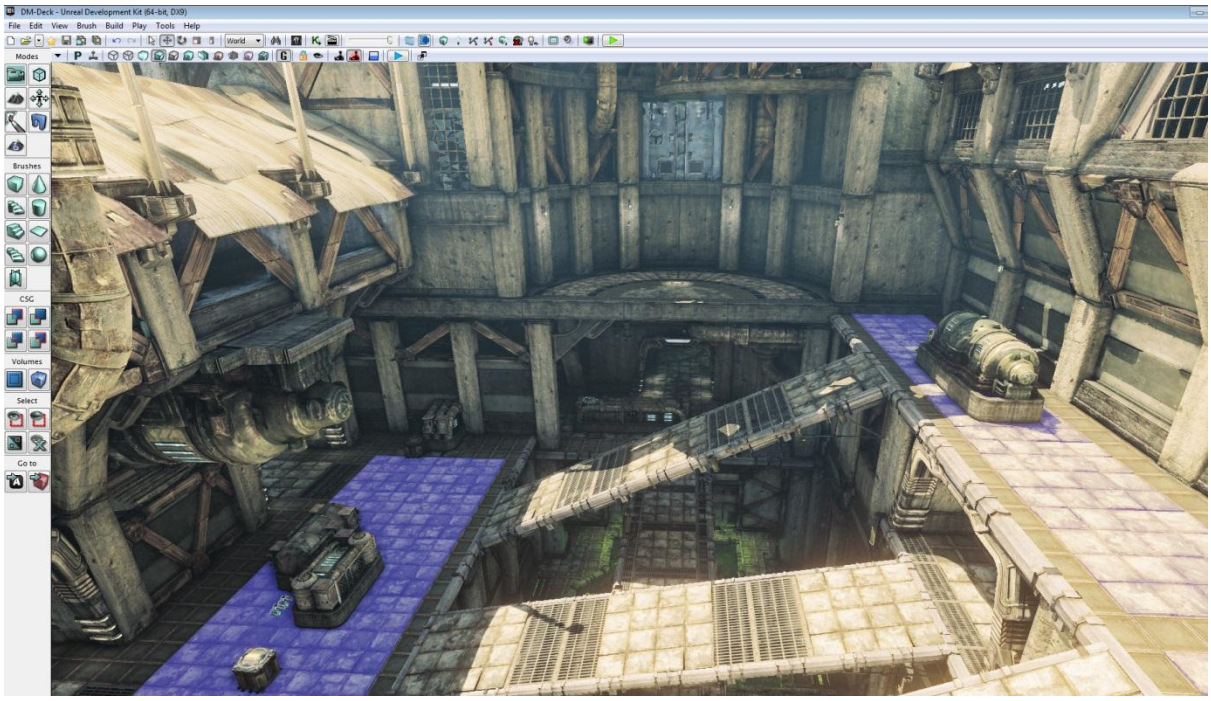


Fig 6.

These are just a few ways to overcome the issue of repetitiveness within a level.

Another issue to consider when creating modular assets is the workflow of the team, changing the way a level is made will also impact the way the project's team will work, an example of this is given by Epic games (2011)

> *"Because the work is now solidly split between artists and level designers, there may be additional layers of communication that need to be established"*

Aspects of this could be seen as a beneficial and also problematic.

The benefit's the fact that the work is now split between the two teams, it means that both teams can be working on the same area of a level at the same time as each other, this allows things as gameplay testing, object interaction and other processes outside of the art department to continue while the art assets are worked upon, then placing the assets in the appropriate places within the level once completed, speeding the development process up.

However on the other hand, the additional communication between the two teams may cause some issues, an example of this is could be seen if the understanding between the two teams falls apart, it

may mean that something is done wrong such as the level designers needing an object to be a specific size, if this is not communicated well to the art team then the asset may need to be redone to fit with the other pieces of the level, creating more work and slowing the development down. Epic games (2011)

Due to the way modularity works, if an object needs changing, it can be done at any point with little effort. Once an artist has update that modular asset, when re-imported into the games engine, it updates all instances of that asset automatically, potentially saving a lot of effort and time.

A by-product of modularity is the potential is for a level designer to be able use assets that where perhaps not originally intended for a specific use, such as a pillar that has been turned into a doorway. Due to the assets looking similar to each other in style this method of level design shouldn't stand out and look odd and gives the level designer a lot of creativity, this method could possibly solve the issue of repetitiveness in some cases.

Fig 9.

2.2.2 Performance benefits.

This section is aimed to clarify and help better understand how important and beneficial modular assets are in regards to performance

There are possibly huge performance gains to be had with a modular design, by using the same objects multiple times but in different locations, a computer system can re-use the original object for all the other instances throughout the level, this is often overlooked and the focus goes on the workflow more than the performance, tests will be conducted to determine how much of a performance boost a level could possibly have through the use of modularity.

Computer Hardware processes 3d assets through the use of graphics processors, early nVidia graphics cards took advantage of instancing, a process of using the same asset in multiple places, allowing the system to load the asset into memory once and render it multiple times increasing performance, while this paper is covering a different area of research, modularity uses the instancing system and so is still a vital part of the subject.

Fig .10

nVidia has been a world leader in graphics processing units (here on in referred to 'GPU') for many years now, often leading the market with ATI/AMD and Intel with their graphics hardware. Below are hardware sales figures of the PC graphics hardware market provided by guru3d.com who has published Jon Peddie Research statistics (Q3 2010).

Vendor	Q3 2009	Q2 2010	Q3 2010
AMD	20.1%	25.0%	22.3%
Intel	53.6%	53.4%	55.6%
Nvidia	25.3%	20.7%	21.2%

However most GPU's use the same systems and work almost exactly the same and not dependent on manufacturer, the only difference that effects games development is the amount of data being calculated at any given time and possibly how specific types of data is used as well. With this in mind modular level design can be implemented on almost any computer device, be it a personal computer, games console or even mobile devices.

To help better explain the graphics pipeline and how instancing works a diagram provided by nVidia below on how the graphics pipeline works.

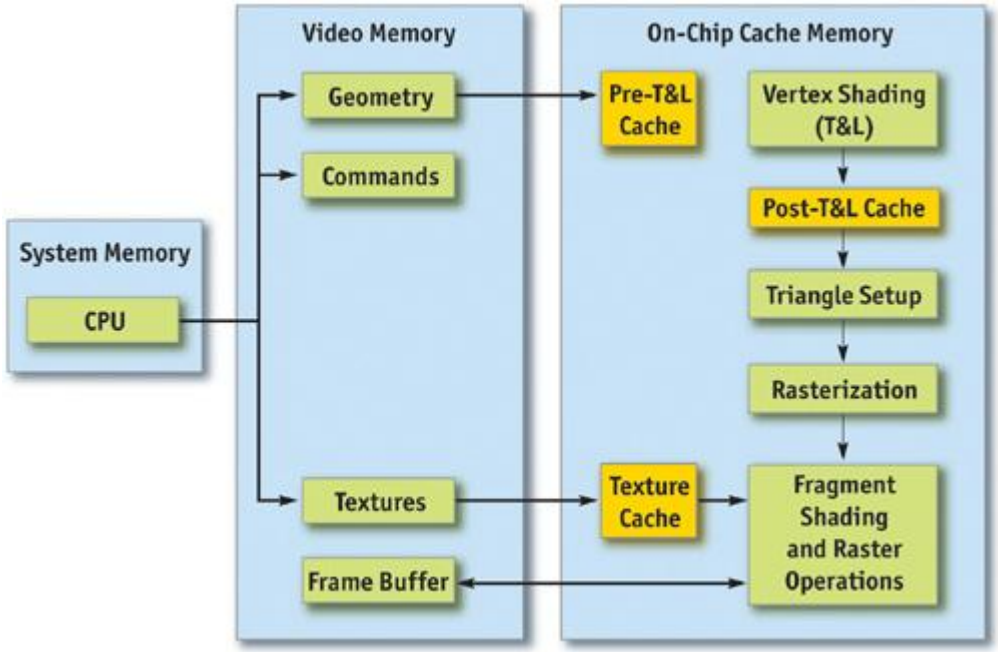


Fig 11

The above diagram shows that the rendering pipeline all starts at the system memory, data is then sent to the CPU, from here the data is then split into two pipelines running simultaneously, geometry and textures, similarly in this way Instancing can be split into these two sections also, the way these two are handled in a game is down to the actual engine rather than the hardware, the hardware gives the potential use of these features only, however as instancing amongst other things have been around for a long time and is built into DirectX and OpenGL (these are programing libraries that enable direct use of hardware features) these features are normally similar in use from engine to engine as most engines will want to take advantage of the power of the GPU.

Everything at some point on screen and off has to go through the CPU, these are referred to batches, modern CPU's can handle about 100,000 batches per second, each object, texture and the other mechanics that are used in games engines are potentially a single batch each. To solve this objects are grouped together and stored as such in the system memory. GPU gems 2 (2005)

When objects are batched together they increase the memory usage as the assets are merged together, this helps performance in frame rate as the GPU can handle large vertex counts and frees up CPU, UDN (2011) Wloka (2003)

Instances of the same mesh though could possibly not take up any more memory in comparison to non-batched items as the mesh still can be sent as 1 object to the GPU with its custom attributes such as position, scale and rotation sent separately. Rege (2004)

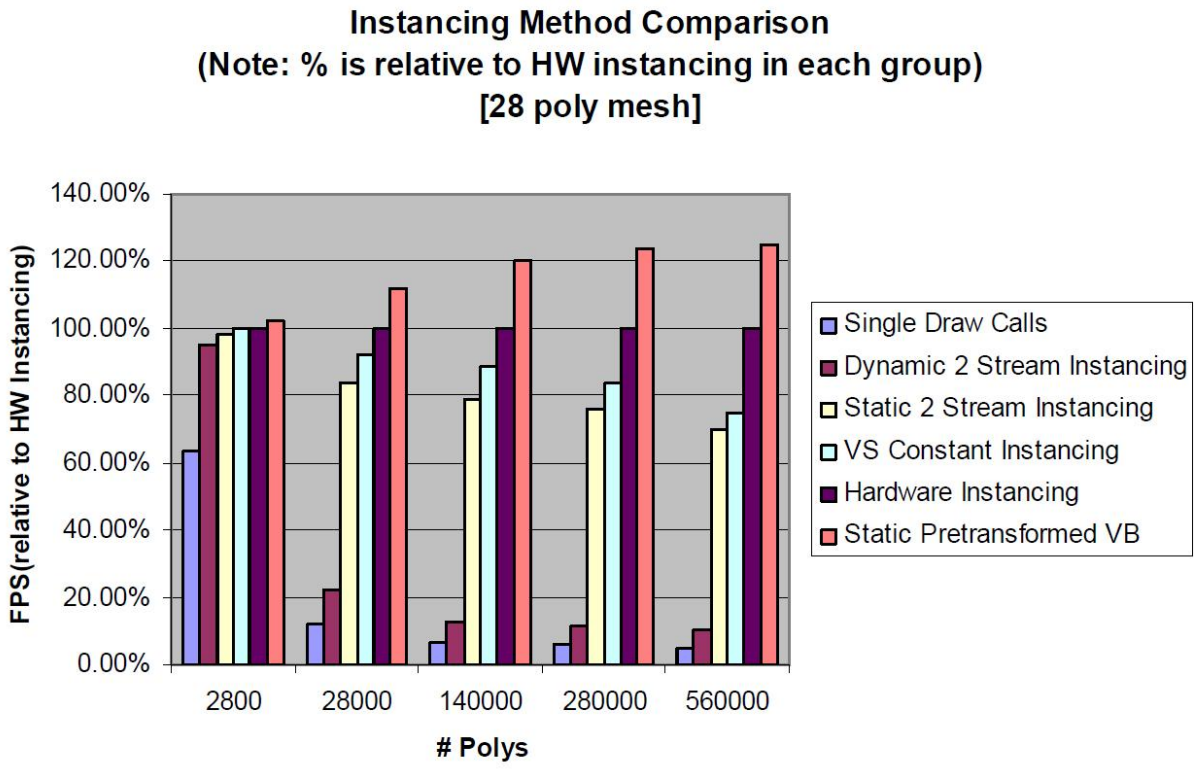


Fig 12

Similarly textures also go through the same process, each texture is sent to the CPU to be told to be handled by the GPU essentially, each texture is again a separate batch, so to help textures can be put into 'atlases', smaller maps combined into one large one, this is handled by the game engine itself and something that artists don't have to particularly worry about but it's something to keep in mind.

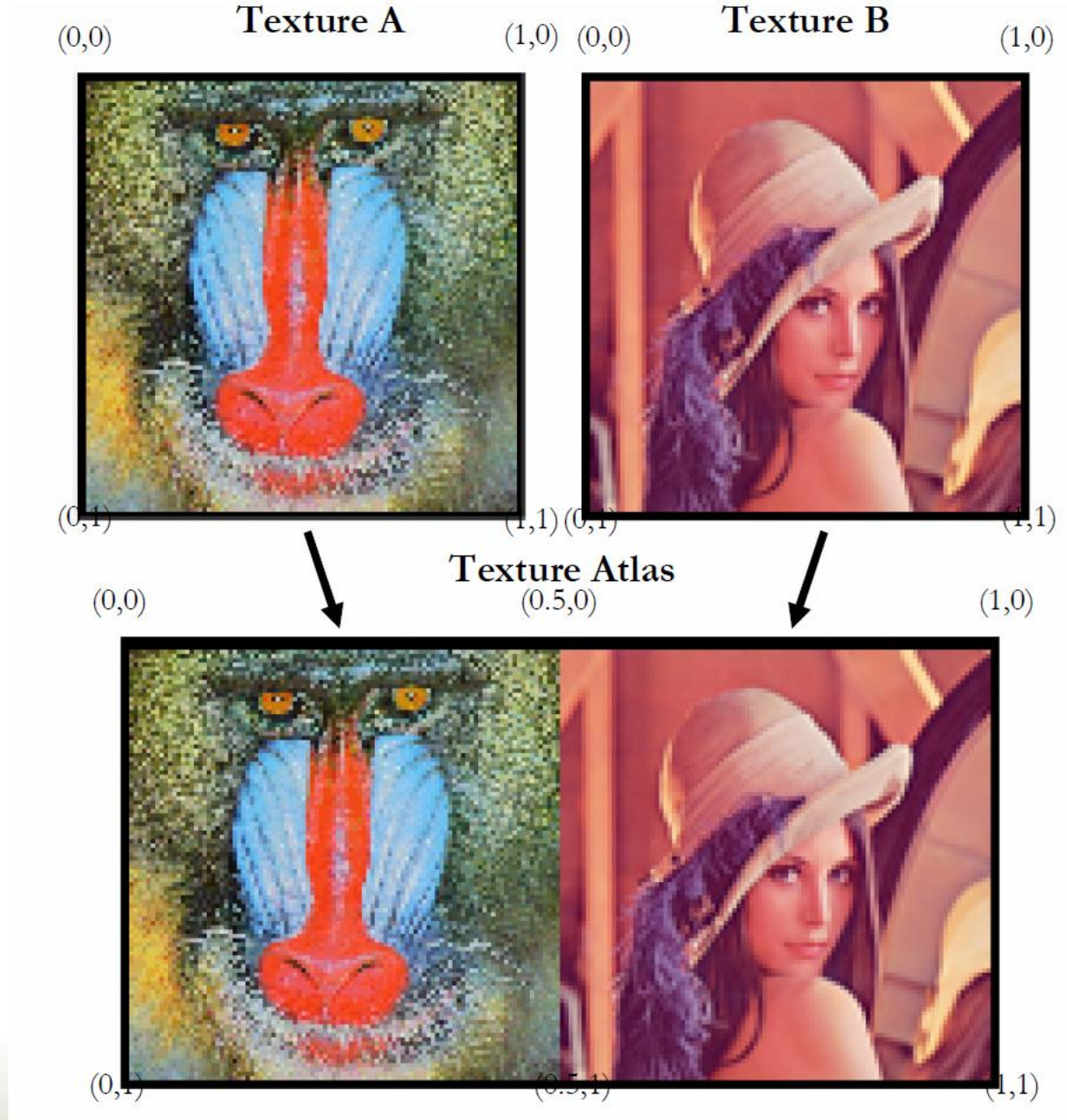


Fig 13

As this is the same process as batching of the same 3D model together, the same can be done to textures, this in terms of performance extends further than just freeing up CPU time to also include freeing up more random access memory (here on in referred to as RAM) on the system and/or GPU memory. If a texture is used 5 times in one scene, each at 2mb each theoretically that quickly eats

up 10mb of RAM where as if the same texture is used just once for all 5 objects, that still equates to only 2mb a huge saving of 80% in memory usage over just 5 objects.

To test this theory demo test was conducted to confirm this above research into performance gains, a simple scene that used a singular asset was created, this asset was then instanced 1000 times in the level and a FPS and memory usage reading was taken, the scene registered at 43 FPS with 2.7 mb of static mesh memory used.

The same scene was then created in 3DS max duplicating the asset 1000 times to the same location as they were in the first test level and then exported as sets into singular asset. These where then imported into the UDK level replacing the instanced assets and then ran the test again.

This time the frame rate was 39 FPS and the memory usage was 106 mb, this means that in this demo there was an increase of 12% in performance and a 3962% reduction in memory usage

Looking at the statistics, it can be seen that the engine is automatically set up to take full advantage of these features, allowing the designer to place the same object multiple times with only storing the object once in the asset package, the same is true with textures as well along with other types of assets.

Understanding these powerful features of how modern GPUs handle game environments can start to give a clearer and more precise understand of how modularity can benefit game design when relating to performance.

if an artist is able to make a level in a modular fashion, where more of the same asset can be added to the world, could potentially only be adding a little extra cost in hardware usage, this enables the game designer more freedom when developing highly detailed levels as they are able to add more assets making for a more richer environment for the player to experience. What's more is this technique can benefit almost any genre of game and gives some understanding to why it has been popular with games developers in the past.

With benefits to several key aspects of game design modularity is extremely powerful, however what has not been mentioned so far is the main drawbacks of modularity, while these may be considered by some to be less of a drawback and more of a way of thinking, the creation of the assets require a different way of construction to work as intended in comparison to making a single, unique standalone asset.

It may be the case that an art team may need to be taught how to create assets in a modular way adding time and cost to a project, though these techniques can be used over multiple projects so the cost of training may not be as high as one would typically think. Section 2.7 will cover techniques used to create modular assets.

A month after this test was conducted a new graphics card was installed, this upgraded the system the test was running from an nVidia 9800 GX2, an old top range card that had built in SLI (multiple graphics cards working together) to a modern nVidia GTX 570, to confirm the test above and also to get more data the test was run again. Surprisingly this time the statistics had changed even though the same level was used, the memory usage was the same however the FPS was higher in the non-instanced test.

To better understand what is going on different hardware settings where applied, overclocking both the CPU and GPU and then underclocking them too and finally a mixture of the two together.

The results of the tests are as follows

CPU	GPU	None Instanced	Instanced
+28%	+15%	117 fps	96 fps
+28%	+0%	115 fps	94 fps
+28	-23%	92 fps	83 FPS
+0%	+15%	119 fps	89 fps
+0%	+0%	117 fps	86 fps
+0%	-23%	90 fps	79 fps
-33%	+15%	116 fps	72 fps
-33%	+0%	112 fps	69 fps
-33%	-23%	94 fps	67 fps

In each instance of the test, the non-modular scene had better frame rate, this was highly unexpected.

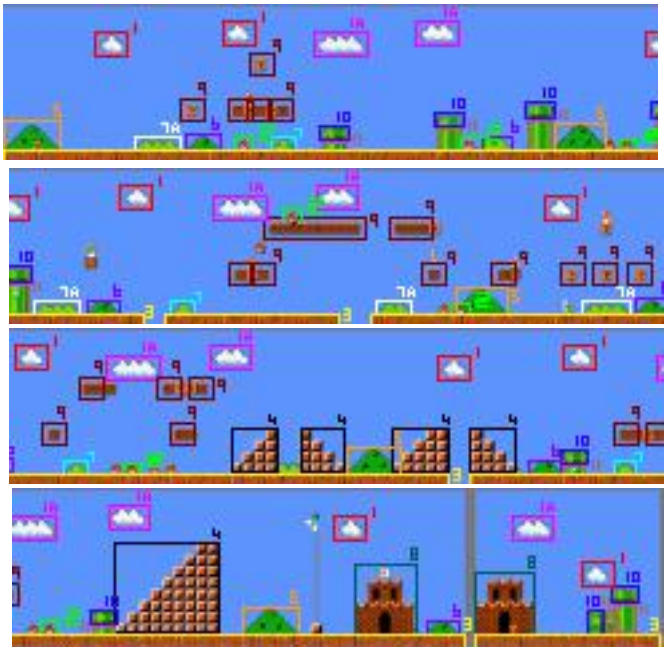
More research into the subject and a better understanding was obtained, according to nVidia and their Games Developer Conference talkers, along with their books they state that performance should be increased, however as epic stated that a batched system would take up more memory further research was done into UDK. From several posts made on the official UDK forums it appears that instancing is not fully supported within the engine, it takes advantage of the main memory savings, however it uses an un-batched rendering system, this would explain why the results above are like they are.

As such, a new testing plan was implemented, this time a scene was made again from 500 assets in a instanced level, however for the un-instanced test, each object was imported as a singular asset, having 500 unique assets replacing the instanced scene, this then tested modularity within UDK without the results being influenced by other factors such as CPU power. Wilson, A UDN official moderator agreed this would be a good way to test modularity.

2.3 History of modularity.

"The modular level design solution arose from the need to have great looking, high-detail levels without having to build and texture very nook and cranny of the environment" Perry (2002)

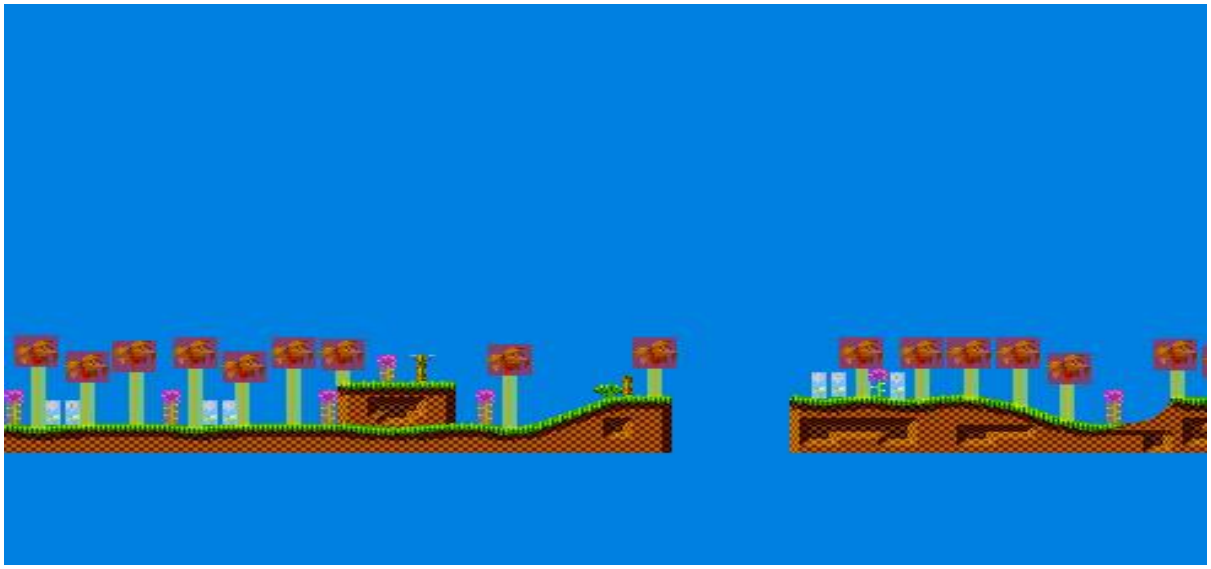
Modularity has been around for a long time now, even back on the Nintendo entertainment system in 2D games such as Mario and also Sega's Sonic used a modular design.

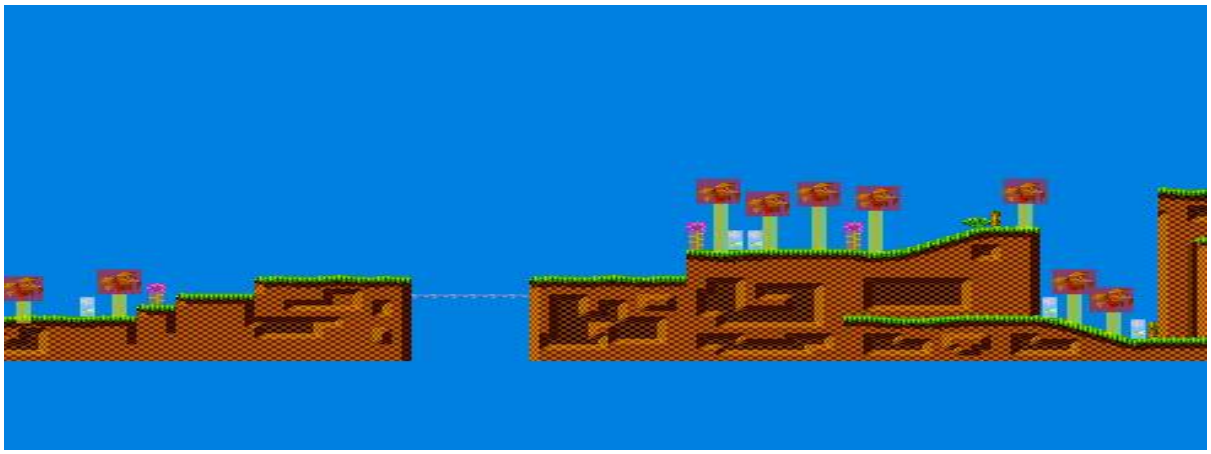


1. Clouds
2. Enemy creature
3. Block floor, repeated along the length of the map
4. Solid colour block, each block repeated to create steps
5. Background large grass mound
6. Medium grass mound
7. Small Grass mound
7a. Small grass mound, possibly repeated and layered
8. Castle, repeated without the flag
9. Floating block
10. Pipe top
11. Pipe Length

Fig 14

Taking a look at the first level of Mario every asset is used several times throughout the level, breaking it down into its unique objects, the most re-used is the floor block, its used as the floor throughout the level, the only time it's not used is when there is a gap in the floor. Along with the floor there are also other interactive objects in the level that are re-used, the pipes, though different sizes use the same material, it also only uses 5 different blocks throughout in different arrangements however the level looks unique throughout. On top of the interactive objects the background objects are also re-used throughout the level, while these are single objects that are randomly placed it still uses the concept of modularity.





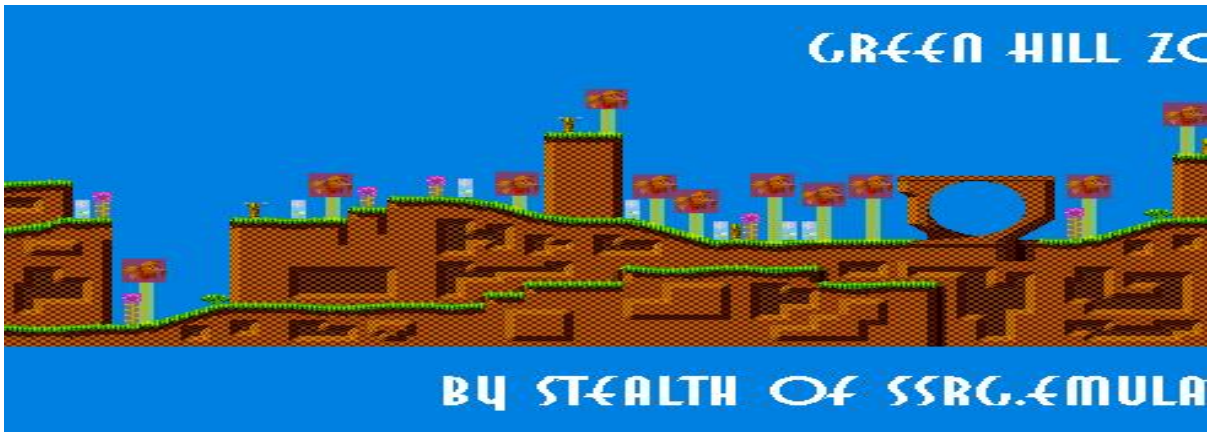

GREEN HILL ZO
BY STEALTH OF SSRG.EMULA

Fig 15 – 19

Similarly the sonic worlds also use a modular design to their levels, this is newer than the Mario level from above and as such uses a bit more complex segments, looking at the overview of the level the ground uses all the same texture/asset throughout, occasionally scaling to fit a better size when required, the trees are also highly modulated throughout the level, however the main difference between Mario and Sonic is the complexity of the levels, Sonics levels are layered on top of each other and also contain some complex shapes for paths that can be followed, joining up at different heights, fitting together nicely due to their modular nature.

This trend of using modular design continued into 3D level design and is now used in most modern games like those mentioned at the beginning.

2.4 Modularity outside of game development.

Modularity is not just limited to computer games, there are many uses indifferent industries, looking at some of these areas outside of games may help to get a better idea where modularity can be beneficial within computer games.

One large area that modularity is used within is the film industry, mainly in sci-fi TV shows and films, such as star trek and star wars, where it would be impossible and impractical to build compete full size sets of everywhere when a lot of the places look the same, so instead re-use stages for different locations, changing small things to fool the viewer that they are filming in a new location. This method also brings down the cost of resources that are needed as the set can potentially be small and thus not much material is used in construction, it also brings down the size of building needed to fit the set in.

Another area that has a lot of modularity within it is the way some houses are constructed, these are pre-fabricated modular houses and have many benefits too, type of these are called Huf Haus

Fig20

Unlike regular buildings these are pre-made in a factory following a set design and constructed offsite, disassembled and reassembled on site taking only a few days to fully construct a house with all fittings and furniture included, the house owner can choose from a variety of fittings all designed to work and fit in the house. As such the house can be customised and personalised quickly and efficiently with no drawbacks due to modularity, bringing the average price down. McCloud, Grand Designs (2003)

There isn't too much that can be applied to games from this section, the most beneficial hint that could applied is the way star trek produce their sets and the ways they make each section different, otherwise this only confirms the benefits of modularity, that it does increase speed and productivity, reducing cost.

2.5 Current use of modular design.

Far from the 2D levels of Mario and sonic are the full, high detailed 3d worlds that people now come to expect from modern games, UDK uses a highly modular approach to levels.

When opening up a level from UDK it only loads a few new packages, this is a system that stores all relevant assets for a level in a singular file, each level is typically split up into multiple packages, with this system the game engine only loads the specific files that are needed to be able to view or play the level in question. For this example DM-Deck has been loaded into the UDK editor.

The quickest way to tell that the level uses a lot of modularity is by looking at the level stats, UDK has many tools to show different statistics that may be helpful to a games designer, looking at the level in question there are 4093 primitives in the level, comparing this to the list of unique items it's clear that there are a lot less than the figure
above.

Type	Name	Count	Sections	Triangles	Inst. Triangles
	Combined Total	4,093	5,025	128,660	1,960,833
StaticMesh	WP_BioRifle.Mesh.S_Bio_Blob_01	37	1	242	8,954
StaticMesh	UN_Trees.Mesh.S_UN_Tree_SM_BurntTree01	12	1	3,258	39,096
StaticMesh	UN_Trees.Mesh.S_UN_Tree_SM_BranchCluster01	1	1	3,644	3,644
StaticMesh	UN_Sky.SM.Mesh.S_UN_Sky_SM_SkyDome03	1	1	112	112
StaticMesh	UN_Liquid.SM.Mesh.S_UN_Liquid_SM_Waterfall_02_1024	3	1	560	1,680
StaticMesh	UN_Liquid.SM.Mesh.S_UN_Liquid_SM_Waterfall_02	1	1	280	280

Fig 21

To look at how much the level has been created through the use of modular assets, the primitives can be ordered by count, this is the amount of times that unique primitive has been used in the level, once ordered it states that the static mesh "StaticMesh'LT_Buildings2.SM.Mesh.S_LT_Buildings_SM_BunkerSupA1" is the most instanced item in the current level (382 times), all of these assets can be selected at once by double clicking on the asset stat line.

Type	Name	Count	Sections	Triangles	Inst. Triangles
	Combined Total	4,093	5,025	128,660	1,960,833
StaticMesh	LT_Buildings2.SM.Mesh.S_LT_Buildings_SM_BunkerSupA1	382	1	552	210,864
StaticMesh	LT_Buildings2.SM.Mesh.S_LT_Buildings_SM_BunkerWallC_STR	252	1	276	69,552
StaticMesh	HU_Supports.SM.Mesh.S_HU_Supports_SM_IbeamA_128	235	1	60	14,100
StaticMesh	HU_Walls.SM.Mesh.S_HU_Walls_SM_Flat	154	1	48	7,392
StaticMesh	LT_Buildings2.SM.Mesh.S_LT_Buildings_SM_BunkerSupA2	151	1	804	121,404
StaticMesh	LT_Buildings2.SM.Mesh.S_LT_Buildings_SM_BunkerSupA3	149	1	48	7,152
StaticMesh	LT_Buildings2.SM.Mesh.S_LT_Buildings_SM_BunkerWallC_CUR2	136	1	320	43,520
StaticMesh	LT_Support.SM.Mesh.S_LT_Supports_SM_YBeam_1	134	1	824	110,416

Fig 22

Going back to the level editor and looking around the asset in question is now selected and highlighted in blue, this particular asset is a concrete pillar and has been used in a lot of different locations, yet still looks visually appealing.

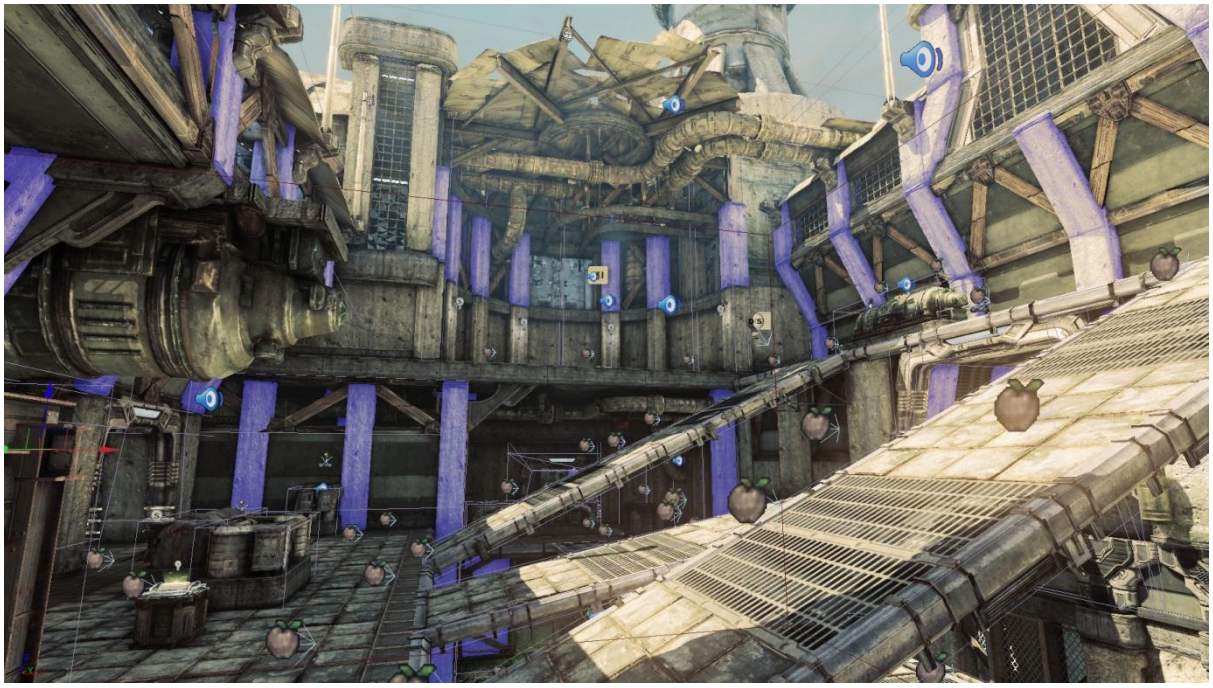


Fig 23

Even though they are exactly the same as each other, because of the way they are used the level doesn't look too repetitive, this has been achieved by making assets look like it belongs with a 'set type' of construction in this instance a sci-fi style set.

Like the TV show star trek a lot of the props have a similar feel and style as each other, they all look like they were created with each other in mind.

Fig 24

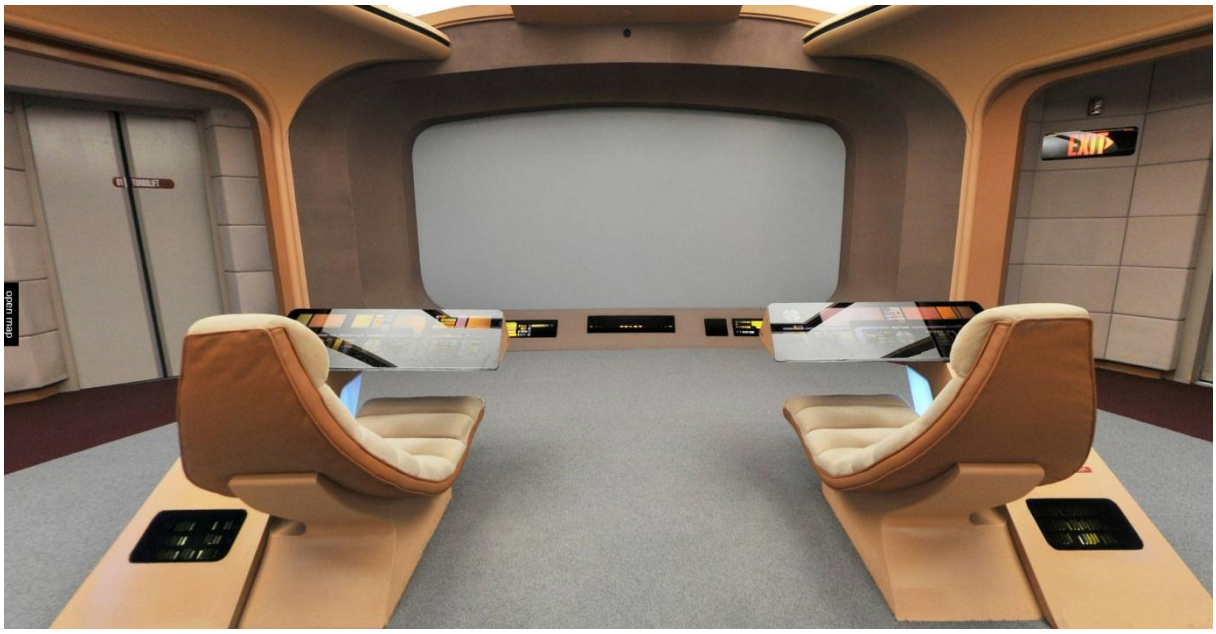


Fig 25

Fig 26

Where it differs from Mario and sonic is they use the fact that they are also designed to modulate in more than just a 2D plane for the environment games used to be limited to, instead they are designed so that they can be rotated in a 3D world, allowing them to also be modular in the depth axis too.

What is also important to note is the surrounding objects, these are not always the same as each other, this breaks up the repetitive nature of modularity and creates a scene that looks unique throughout, sometimes this is not important however in a lot of cases it is, the normal case when making a level for any given game is to make it feel realistic and keep the player engaged at the task at hand, if the scene was continuously the same throughout, repeating the same little bit over and

over again probably would not be very visually appealing, making what could be an exciting game very boring and dull, it also doesn't give a natural feel to the level due to the fact, nothing is exactly the same in reality.

One type of game that this may not be true for is car racing games or simulators, as the environment is normally moving past the player at a fast pace, it's most likely the case that they will not notice the repetitiveness of assets in the level, instead the only concern for this genre of game is having an appropriate asset in the environment as the player would then notice if it was not there. Epic also picked up on this too.

"If your player will be rushing past the pieces (as in a racing game), then you can afford to make larger, less detailed chunks"

Another prominent feature of the techniques used in UDK is the fact that the assets cut into each other and don't necessarily perfectly align with each other.

The following renders from the level demonstrate this method in different areas and locations throughout the map

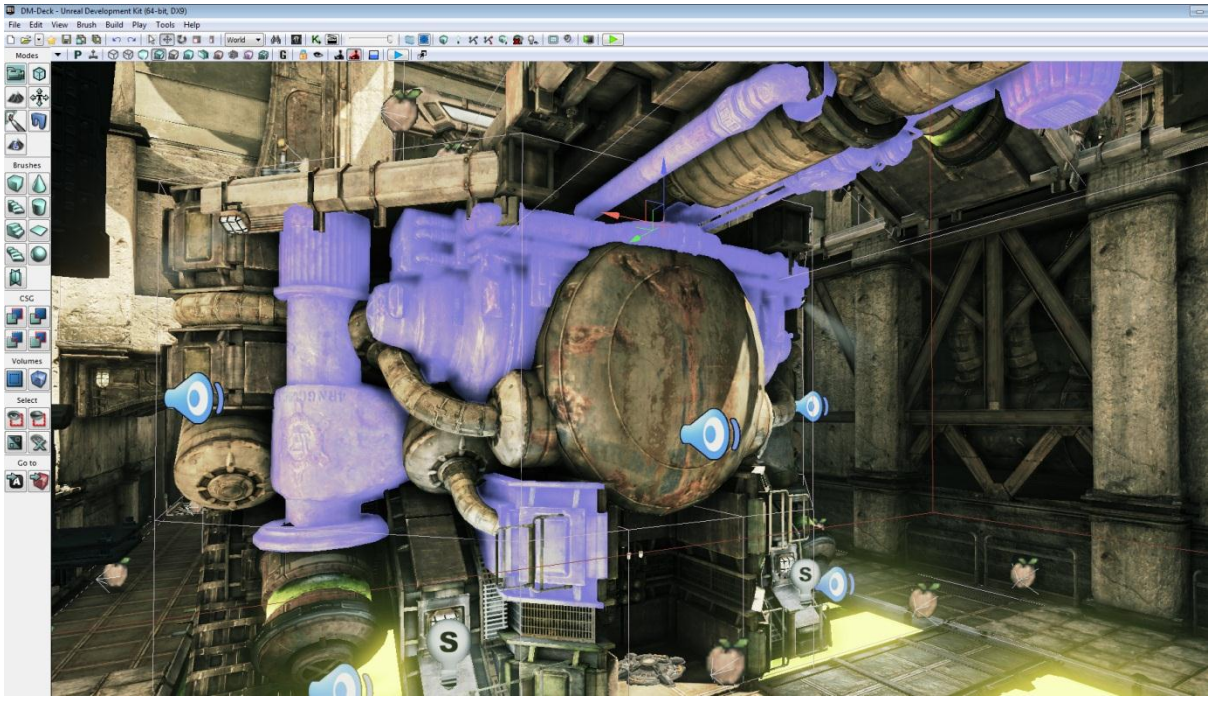


Fig 27

Above is a render from the level with several assets selected to demonstrate the amount of intersection and yet there are no graphical errors neither does it look out of place

Here there are a lot of intersecting objects, they are placed around to add detail to the map, due to these assets being used throughout the level it doesn't add any more work for the artist to do and can be quickly done by the level designer

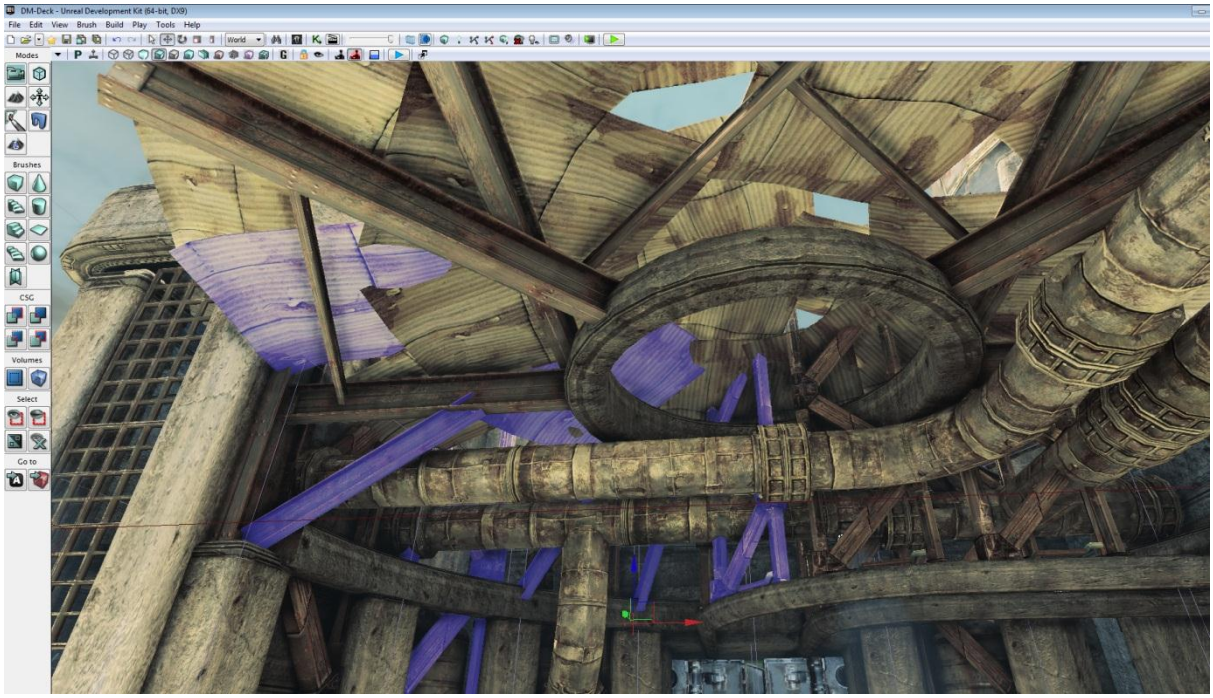


Fig 28

Below assets are intersecting through the entire wall to add detail to an otherwise simple plain wall

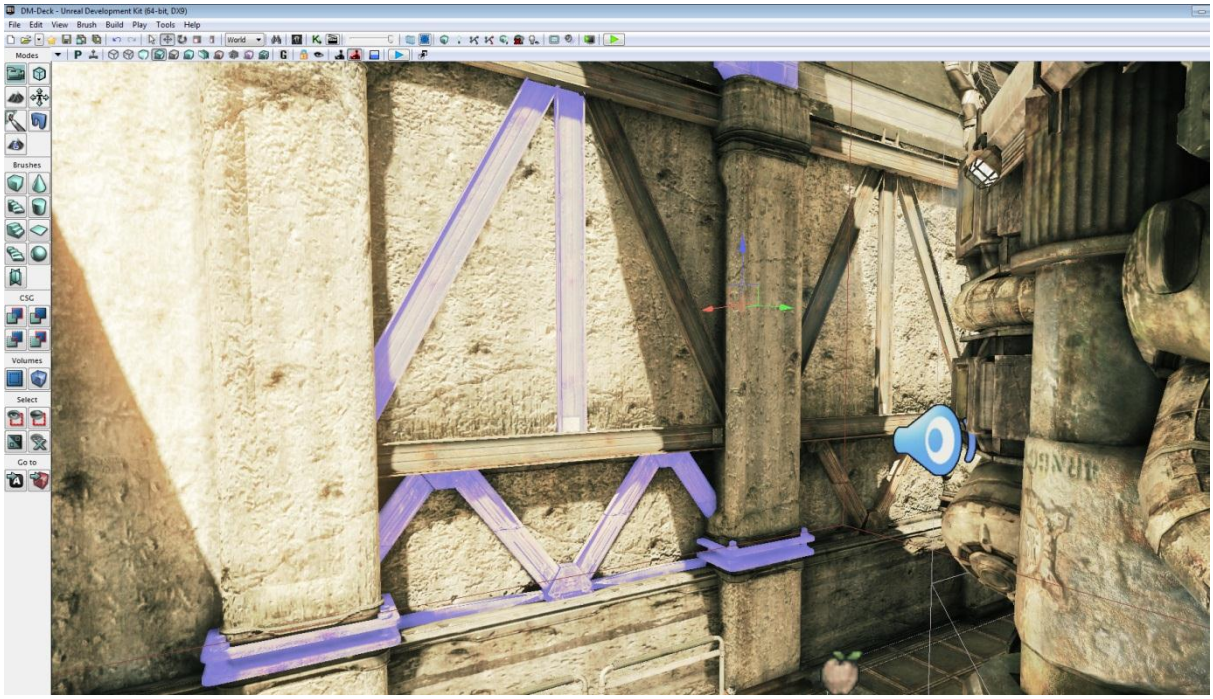


Fig 29

In the render below are the assets that are cutting through the wall, they are a lot bigger and go a fair distance into the wall however this is not noticeable to the player and causes no rendering issues and as such is a cheap way to add detail.

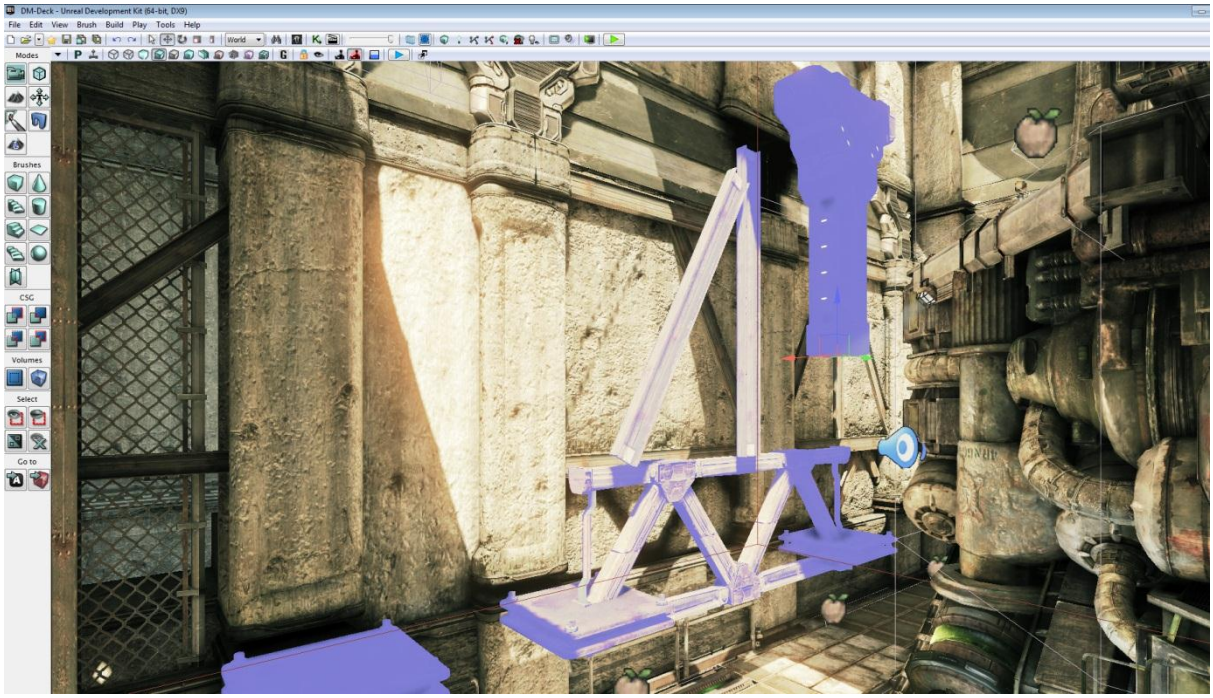


Fig 30

UDK is not the only game that uses modularity, as mentioned at the beginning Microsoft's Halo and BioWare's Masss Effect amongst other games also use modularity.

Fig 31

Mass effect is a game where the repetitiveness of modularity is well hidden and not too noticeable, this is mainly due to more unique assets, however it's also the strategic position of the modular assets and their shapes to give a feel that the level is not repetitive, some of the trees are used multiple times, some scaled and rotated so from any one angle they do not look the same, the use of steps are also modular, joined up with the floor neatly along with the large support structures.

When it comes to multiplayer levels its normally the case the level needs to be balanced for each team, in halo 3 the level designers have used modularity to their advantage for creating their multiplayer maps, mirroring half the level to form the opposite side, due to modularity both ends fit nicely together to form a complete map, within each side modular assets are used extensively to create the bulk of the environment, some of these are highlighted in the images below, colour co-ordinated to demonstrate the possibility that they are the same asset

Fig 32

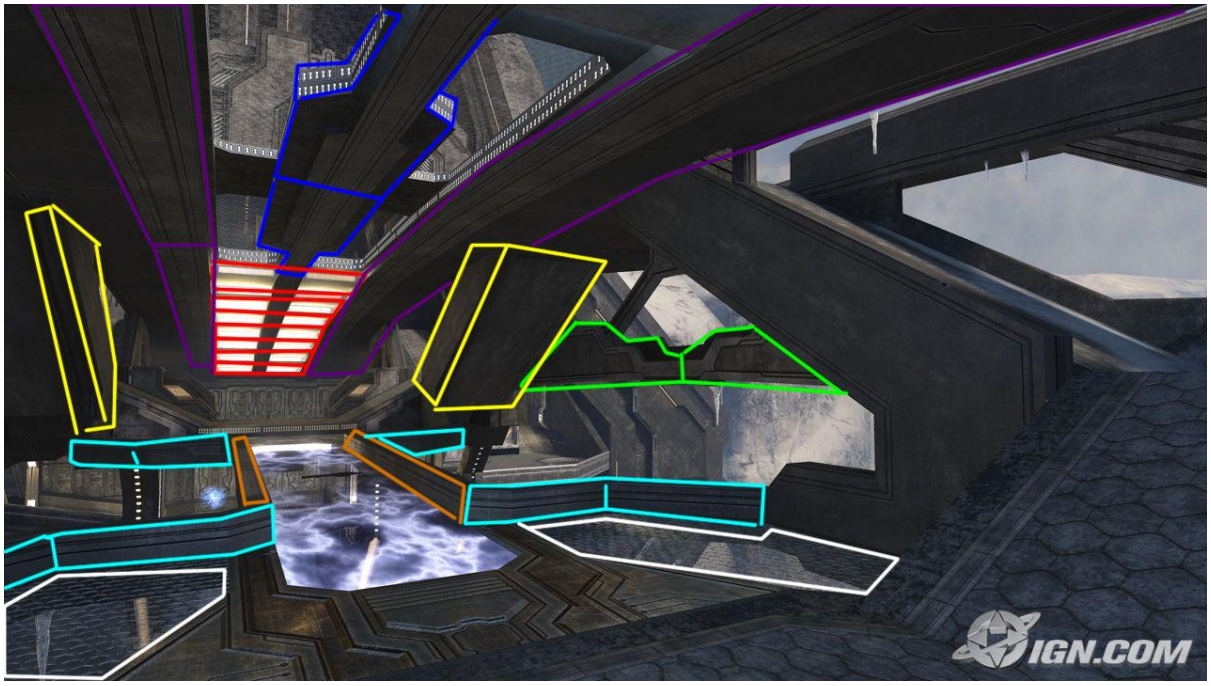


Fig 33

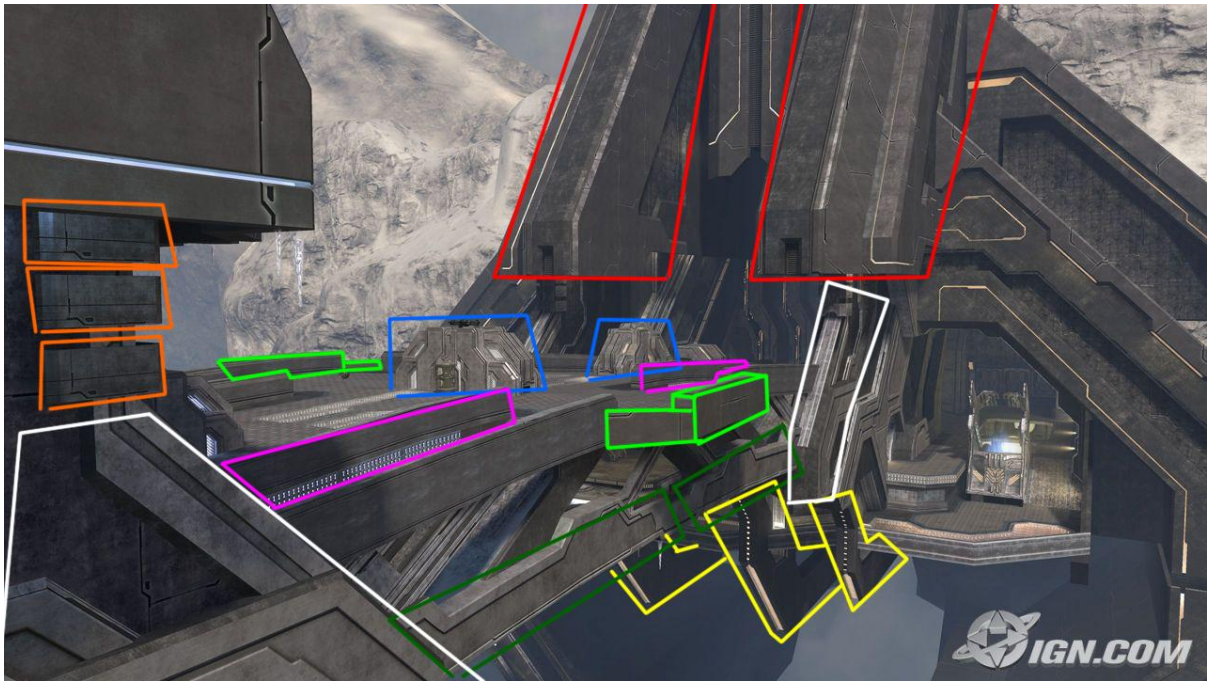


Fig 34

2.6 When and where to use modular assets

As stated previously, modularity is a powerful method of level designing and has a wide range of uses in almost all games, so when is the best place to use modularity.

Due to evidence reported in this document, there are more benefits to modularity than disadvantages and in short it's easier to state when not to use it.

So where is modularity not appropriate? Well the most obvious is where performance is not an issue and where it would be quicker for an artist to create unique assets. Epic Games (2011) gives a good example of an inappropriate time to use modularity.

> *"If you are just making one room, then a non-modular room may be much faster to create than making separate pieces. Once that number grows to 100 rooms, however, you'll understand the time savings that can result when you adopt these practices"*

However this is partially true, if this room is part of a larger level and is supposed to look different from the rest of the game or alternatively for some reason a singular room is required for an entire level, then this may be true because it may be hard to use existing assets to give a unique feeling, so it may be quicker to make the room out of unique assets, but if the room is part of a level or looks similar to another room within the game, modularity is still an option, it may be the case that a mix of unique and modular assets could be combined to create such a room.

Other situations that may determine how modular a level is will be down to how much time the player will be spending in that particular environment, similar to the racing example in section 2.5, if the player is not spending too long in the area, the artist can make large modular pieces, that may not be too modular in nature, an example may be there could be a building for each module, these put together to make a town that the player may pass through quickly, these in a sense are only partially modular as they will not fit together like previous examples however having streets to separate them they become somewhat modular. So modular can be worked on in different scales depending on the situation. The following statement from Epic Games should help understand what size of modularity is best for different situations.

> *"If you'll be spending a lot of time in an environment, then a smaller scale of modularity with more complicated pieces is called for. For comparison, imagine a town that one would fly over in a helicopter, as opposed to the interior of a spaceship that you wander about it. In both of these cases, the use of modular meshes can speed up the development of a game considerably."*

2.7 Methods of modular creation.

This section covers different methods of modular asset creation within the industry, examples are taken from different sources all within the games industry.

2.7.1 Epic's UDN (2011) on modularity

On Epic's UDN website they give a general overview of modularity, Epic suggests the first step to take is to set the scale of modularity, depending on the game or level will determine what scale to use, according to Epic, if the player is to be passing through the environment then larger less detailed modular assets can be created, if instead the level is inside of a building then the chunks can be smaller such as walls, floors and roofs etc. instead of whole buildings modulated together.
Epic states that early on in the process it's very important to communicate between level designers and artist to state the focus of the level. From this the artist should then work on the most useful assets that can be immediate given to the level designers so they can start creating the level from the different parts.

A recommendation given is if the team is confident that it could be beneficial to start with placeholder parts that then can be swapped out if they are approximately the same shape and size, possibly speeding up development.

The next step that Epic takes is to get the size of objects in comparison to the player, these are things such as the height and width of assets such as walls and also to work out things such as how high the player can jump, these numbers are important to all parties involved even people such as animators, prototype testing can be also helpful here.

Epic continues on to mention the grid, this is a large section in comparison to the others, they set up the grid in max and Maya to match up with the game engines grid, in this case, a version of Unreal Tournament. In UDK or UT3 assets have the option of snapping to the grid, this can be helpful when creating a modular set, if the grid was off it would be hard for the level designer to perfectly place each asset next to each other and may also cause graphical errors, however with the assets on the grid they can perfectly snap into position relatively quickly and with no side effects such as the graphical errors mentioned above.
If the same grid is used in the games engine and modelling package it helps planning and synchronisation, as if it fits

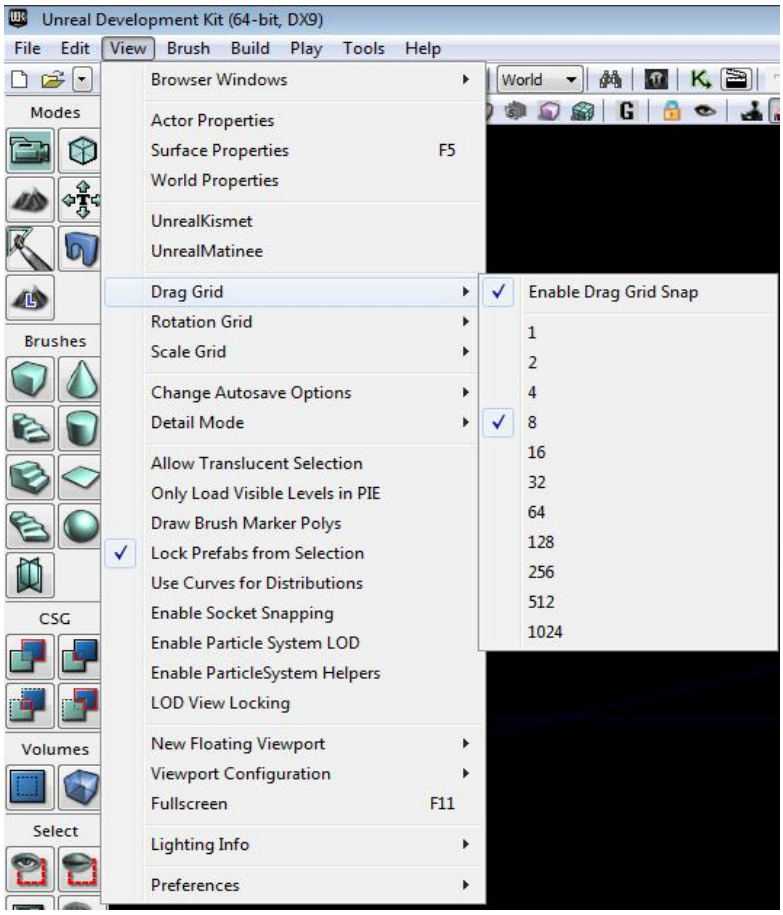


Fig 35

together in the modelling package it's guaranteed to work in the games engine. A good analogy made by epic is that a modular setup is like Legos, from simple building blocks complex structures can be made.

With the grid setup and working Epic make it clear that it should hardly ever be turned off as it can cause problems even if the assets are on the grid in the modelling package, however they do mention that the rotational grid is not as important to follow, especially for organic assets.

CG society member and award winner Stefan Morrell (2007) says this is the most important aspect of modularity especially at the beginning.

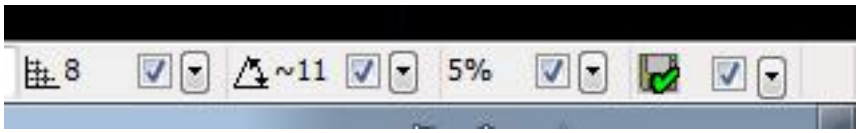
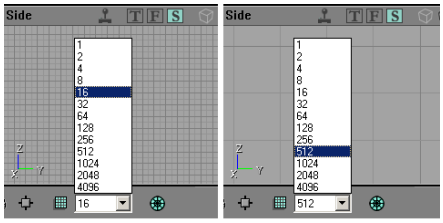


Fig 36

Epic makes an point that the grid used is best to be a power of 2, UDK doesn't allow any other grid size that is not a power of 2, however it's possible to divide the grid down, in their example an object can still be made to a size of 1 unit as this is still a power of 2 and as such on this grid level any object that fits on the grid will fit on this lower grid regardless of length, however it's also important to note that the smaller the grid size the harder it becomes to place objects next to each other.



16-Unit Grid 512-Unit Grid

Fig 37

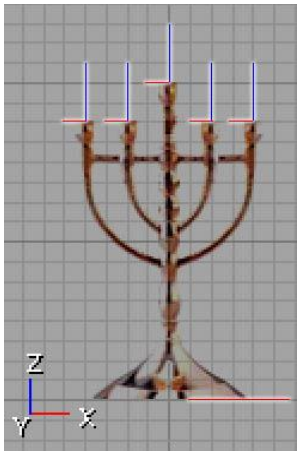


Fig 38

Epic continues the creation guide by explaining that modular assets are not just objects that sit side by side, but could also be possibly put on top of each other or underneath, an example given is of a candle on a candelabra, with the top of the candelabra on the grid, placing the candle itself will be easier as it will snap on the grid.

Some helpful tips are given on how to avoid problems , mainly issues referring to repetitiveness, they suggest to mirror objects to reduce the problem and that they are not just limited to mirroring side to side either, assets have the potential to be mirrored in any axis. Text will not work with mirroring however as the text itself would also be mirrored and be backwards or up-side down.

The origin is also very important Epic explains, The origin of the model is important when keeping things on the grid, if the origin is in a random place of the model then the model being on the grid doesn't matter much, the origin is best located in the same place on every model, Epic suggests to place it in the bottom right corner of all assets, this is to help with rotation and scaling, this allows the corner to constantly be on the grid when scaling it to fit odd gaps, most modelling package keep the origin of the model in the middle, if left here the assets would have to be realigned in the games engine to work on the grid, somewhat defeating the point of being on the grid.

The origin for assets within UDK is the centre point of 3ds max and Maya, to get the objects working how they should each asset has to be moved so the bottom right corner of the asset is exactly on the world origin of the modelling application.

2.7.2 Paul Mader

Mader wrote an article for gamasutra (2005) on ways to speed up modularity, at the time of writing the document he was working on the unreal 3 engine, while the article doesn't go through the entire process of creating modular assets it does cover key points, these are the same that Epics UDN mentioned above, however he expands on one key area, the pivot point.

For the most part he confirms what Epic stated, however for objects that are curved and rotate to meet up with the next part like that shown below, the pivot point is placed in the centre of a complete circle version of that particular model.

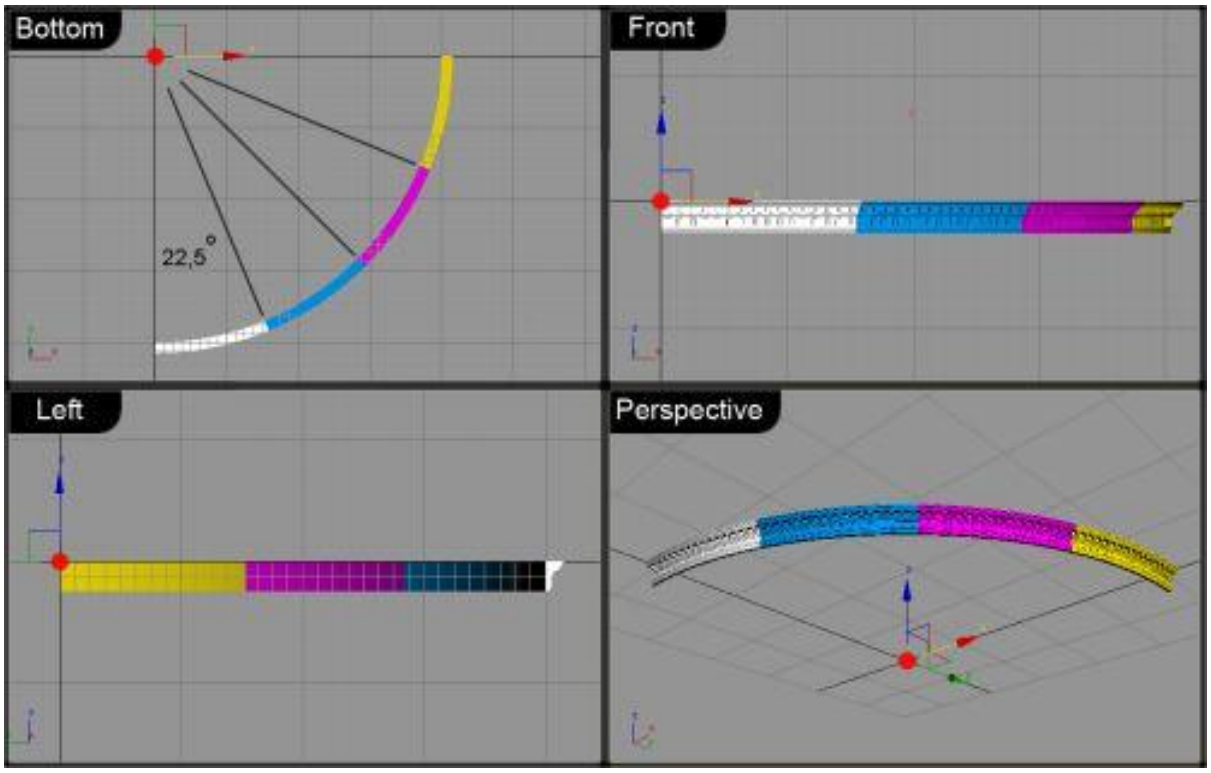


Fig 39

2.7.3 Lee Perry

Perry wrote an article for game developer magazine covering modularity, Perry at the time of writing the article in 2002 was working for Epic Games, as such the content from the UDN website is similar however there are a few more hints on how to improve modularity

One point that Perry expands on is planning, working with level designers to prioritise assets and construct an asset list to include things such as props to landscapes, he suggest to do this after organising the scale and grid, at this point it's also advisable to plan for end parts, Perry gives an example of a modular river, there will need to be an asset that naturally makes the modularity finish, for walls this could be a pillar or a door way if needed.

Following on from that, Perry states to start with the basics, at this point it's not advisable to start making transition assets from area to area or complex junctions, leave these till later when the levels are more developed.

Perry finishes up with ideas to overcome the repetitiveness of modularity, he explains that adding accessories to things can go a long way to making places feel unique, for example adding broken beams, or posts etc. to a castle hallway. Modularity can be taken further at this point, making props from modular parts, this can help break the level up even more, having different props that can mix and match together.

These accessory assets can also be used to conceal problems that may arise with modularity, they could be strategically placed to cover thing such as holes or seams in the texture or seams where objects may intersect such as natural rock formations.

Once all of the above is done, Perry then starts making the unique assets for the level, creating areas that need to be different from the rest of the game, as these are more one off bits the grid is less important here, however it's still a good idea to finish the edges up on the edge as it helps to fit the objects together perfectly. Another factor to think about here is to look at the asset and see if it can't be broken up into chunks to be used somewhere else as well, this is maximising the use of all objects.

Finally level assembly takes place, this is where all the benefits of the above come in, from the different sets of assets a level now can be made. Due to the modularity of the assets there is a lot more flexibility when constructing the levels, before requesting the addition of more assets, take a look at the assets that have already been made, see if any of them could be used outside their main purpose for whatever is needed. This could save a lot of time and speed production up overall with the added bonus of creating a more unique environment from using pieces that were not expected to be used.

2.7.4 Tyler Wanlass at 3DMotive

At 3dmotive.com Wanlass produced a video tutorial on how to create modular assets (2011), unlike UDN and Perry he takes a different approach to creating modular assets.
Instead of starting with the 3d model first like a lot of artists do, he starts with the texture first in Photoshop, again he uses the grid but instead of using it in max it's used in Photoshop first.

Using photo textures Wanlass breaks them up into their parts such as bricks, trim detail, windows etc. and places these on the grid in Photoshop, using standard texture power of 2 size this then forms the texture for the object, again the grid size can be set to any power of two but as before the smaller it is the more tedious it becomes to place the objects. It's important to use the grid here as it would have been used in the modelling packages, having the size of objects in proportion to each other, while the actual size of the texture is down to the texture size allotted for the item, this is then relative space in a modelling package and as such can be made bigger or smaller, but its best to keep it all at the same scale.

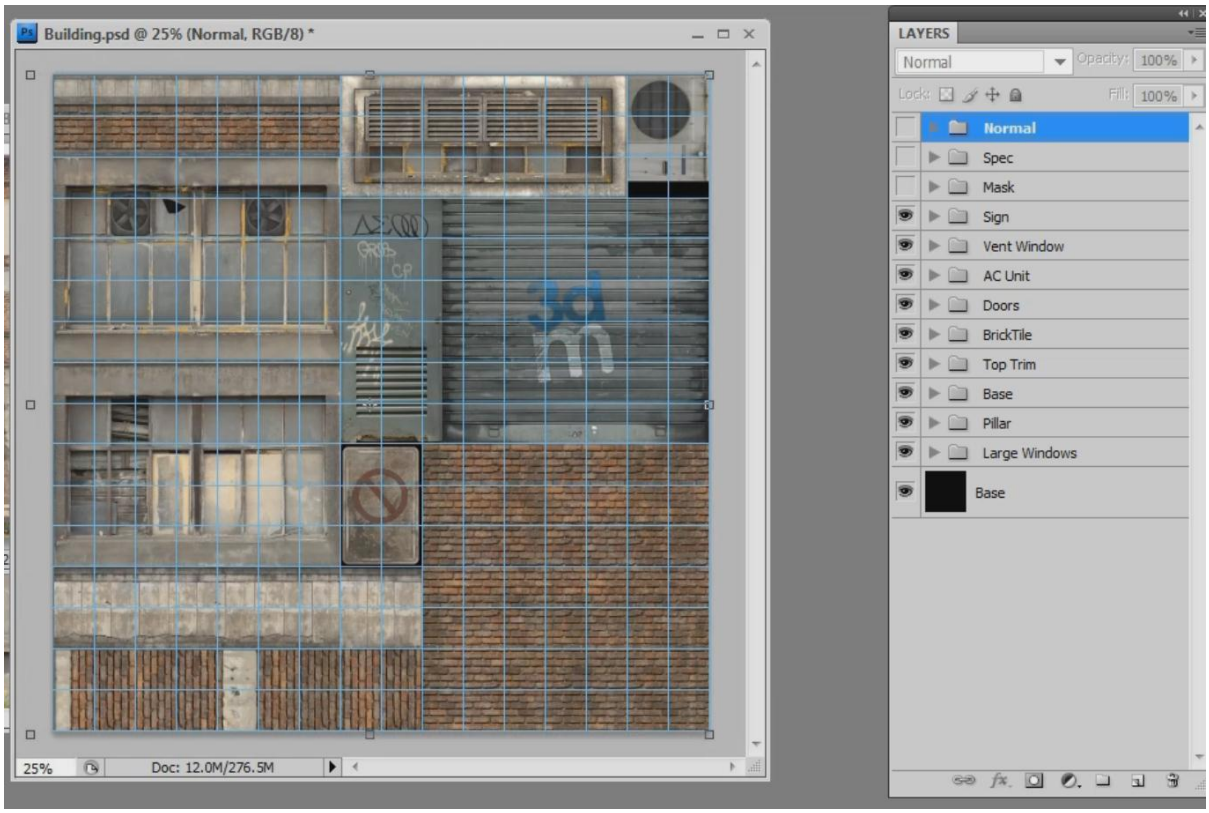


Fig 40

This grid is then set up in the modelling package to match that of Photoshop. As the texture is aligned to this grid it will then align to the grid of the modelling package.
A flat plane is then made to the size of the texture, so if the texture was a 1024x1024 pixel texture then the plane would be 1024x1024 units in the modelling package, the material is then applied to the plane.

This is now set up ready to model the pieces from, he starts by splitting the mesh into segments to the same amount of that in Photoshop's grid, then breaking up the plane into separate sections using the different parts of the texture.

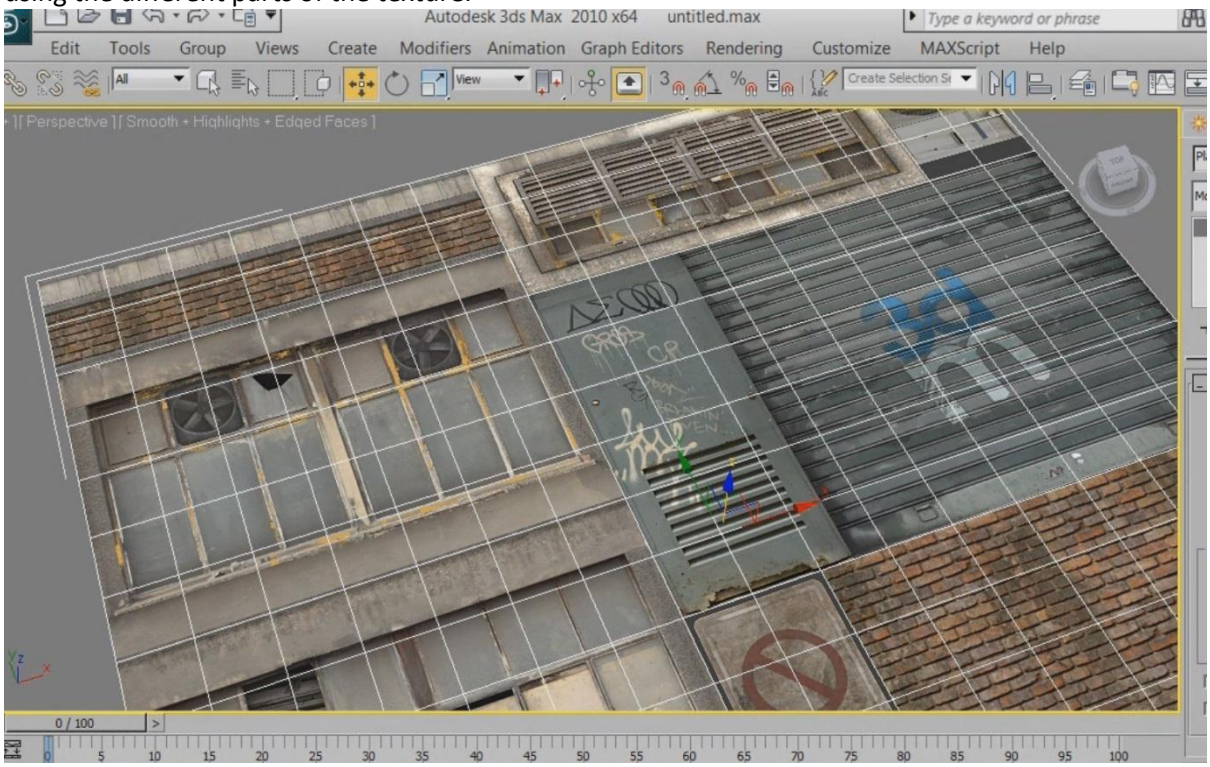


Fig 41

This has now separated each part into objects that have perfect pixel ratio and perfectly align to the grid. Using these parts the model can be created from using extrude, cut, bevel and other tools to give this flat plane depth and 3D shape, while most of the UV map will be good, after altering the mesh like the above causes some problems, the extruded parts do not have any new UV co-ordinates and so have to be manually placed into position, this may be a little difficult as instead of making the texture based on the model, the model has to be positioned in the best location, with time this may not cause a problem however there is a chance it causes a seam in the model as these are removed in creating the texture, as this normally happens on a corner though, depending on its position in the level it might not be noticeable.

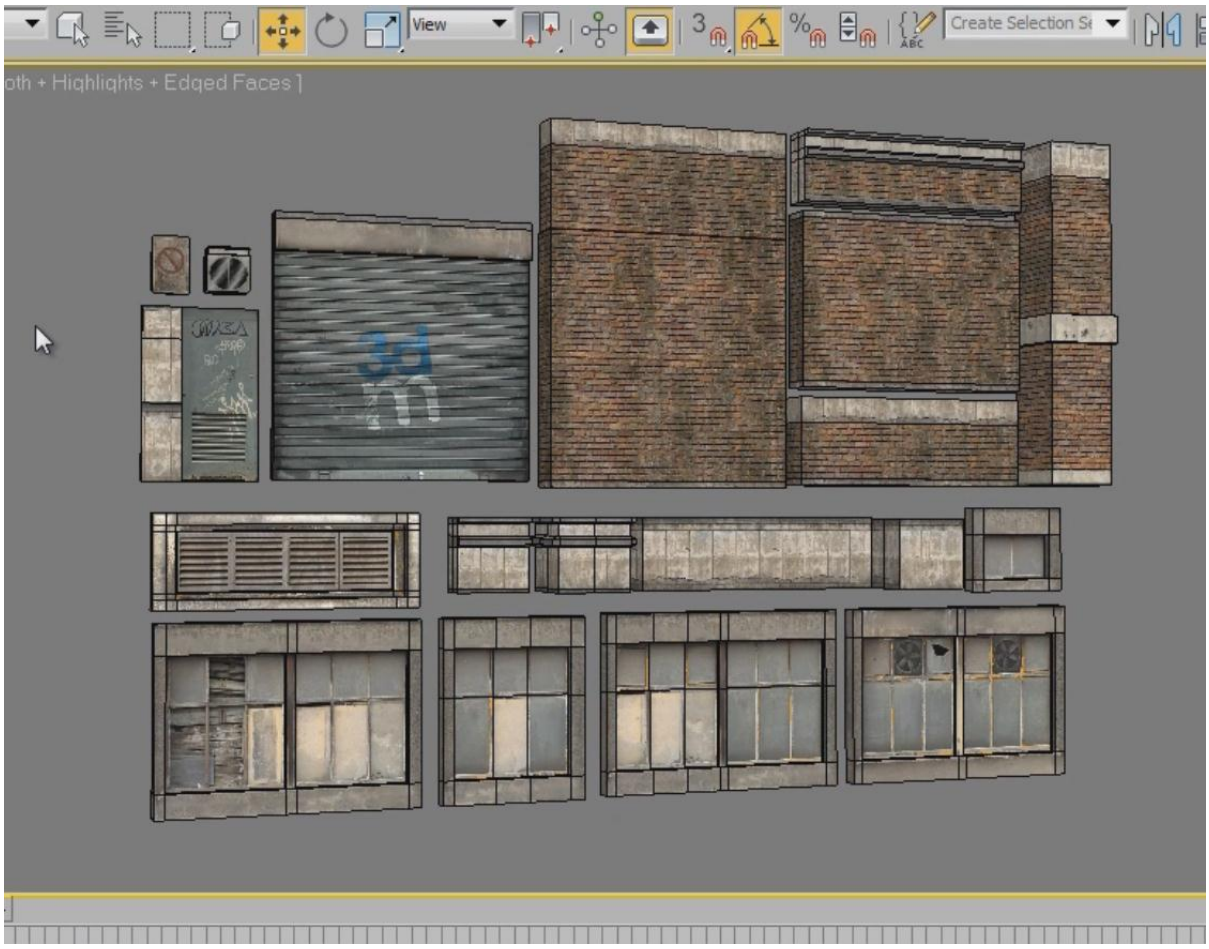


Fig 42

These parts are then exported to the game engine, due to the unit scale being setup the same as the engine the assets automatically fits onto the game engine grid, making placement of the meshes the same technique as with UDN and Perry above. Due to the parts being same size as each other such as the door height, wall height and wall variants height it's easy to quickly swap them out and start breaking up the relativeness of the level caused by modularity.

Fig 43

Fig 44

Fig 45

2.7.5 Chris Robson at 3D-Palace (2011)

On 3D-Palace Robson has created a short tutorial on how to make modular architecture, where this differs from the other workflows is that it's not games orientated, however has been modelled and constructed purely in the modelling package, as such the tutorial specifically starts out saying that it does not form to the grid as the others do, however has techniques on how to make a modular scene.

In the video tutorial he starts off with a basic shape block for the structure from this he models out pieces like he would usually modelling, not concerning with the scale and grid but just keeping a relative scale between parts

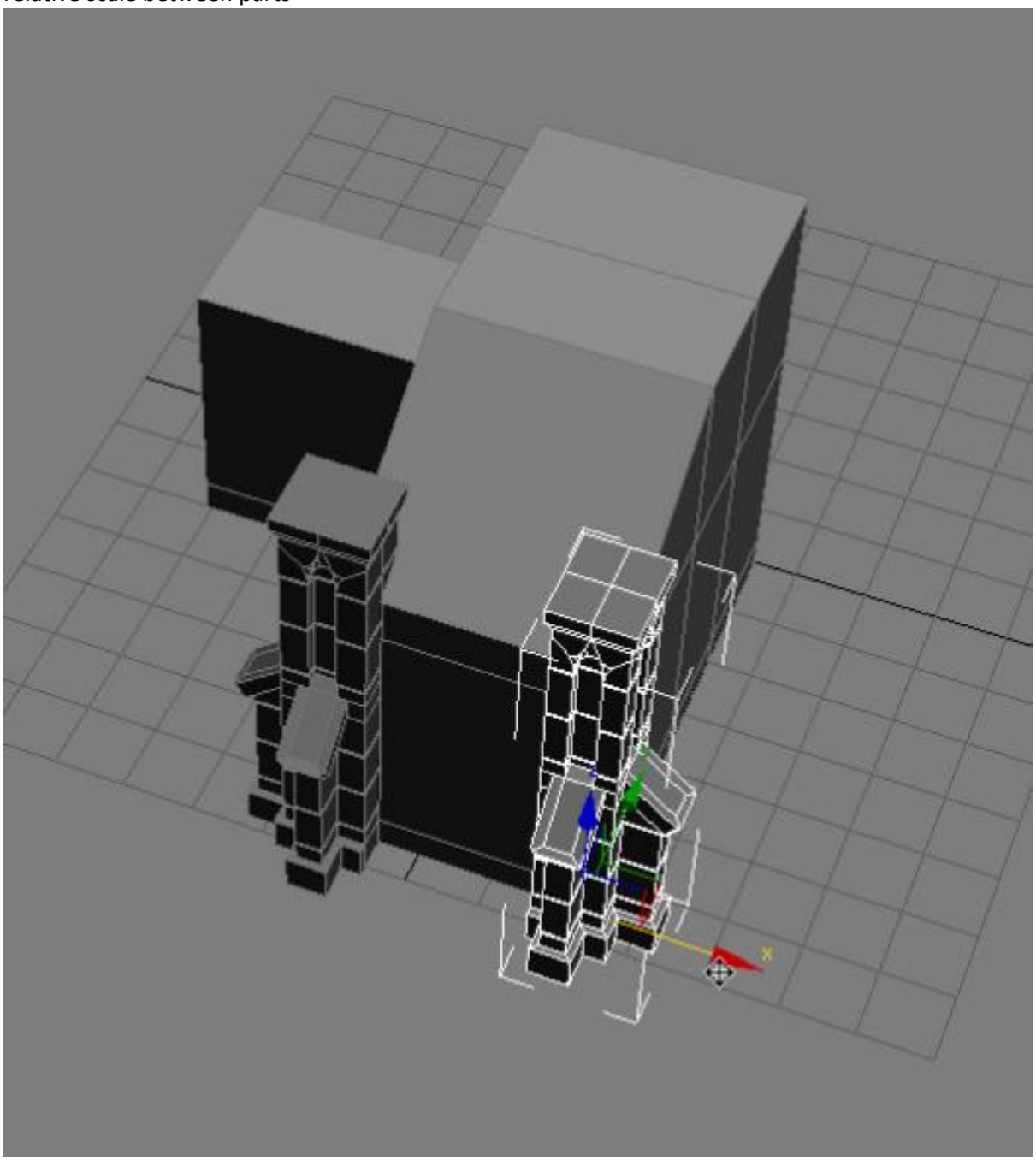


Fig 46

There are issues with this method such as the ease of use within an engine however it demonstrates its possible to make a modular scene without following the rules, this could be adapted to work with unique areas that wouldn't normally be modular, as each object is modelled normally it wouldn't add any more time and would in fact save time when building the level as the assets would be re-used and as epic has proven in their level, if they don't intersect too much then the grid may not be an issue in this instance

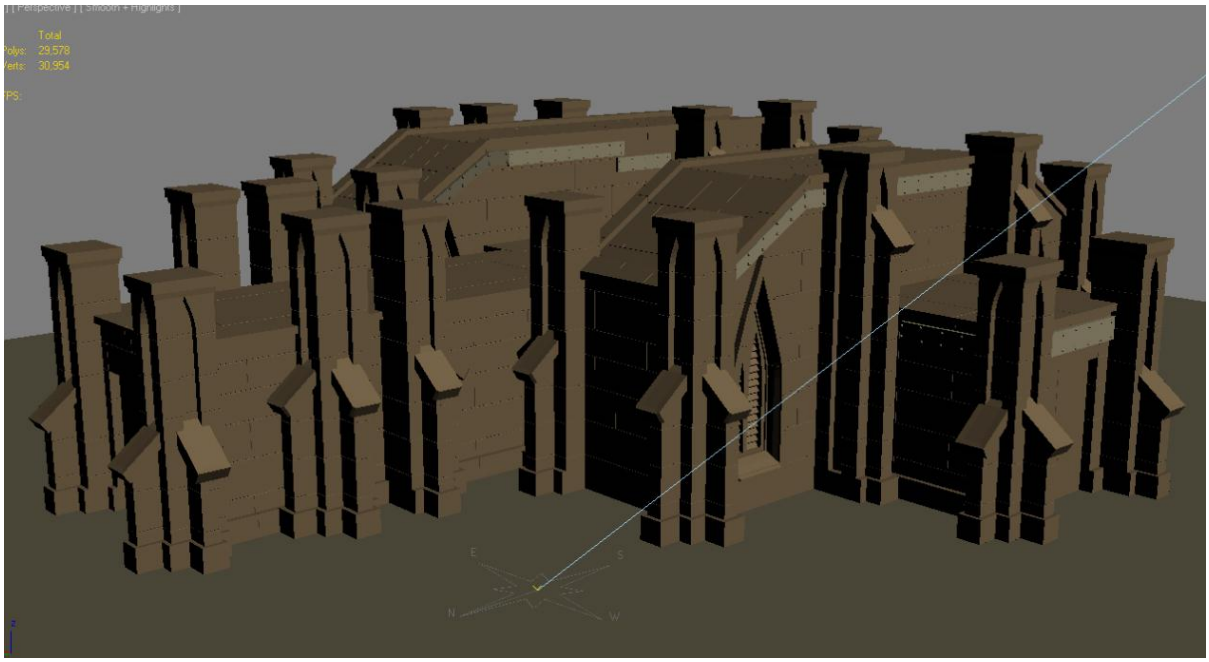


Fig 47

2.8 Section 2 brief conclusion.

That covers most of the theory behind modular assets, a lot of information was taken from the UDN website and as such may be bias towards their engine, this though should be highly beneficial due to the practical example will be made in UDK.

Similarly with the information on performance benefits was mostly took from nVidia, sections 2.2 and 2.6 are not documented often and finding contrasting views for these sections are few and far apart, this paper hopefully fills these areas of research in a little more, with that said the paper covers most aspects of modularity that will help in the next section "Practical Example"

The research gathered in section 2.7 will be used to create the assets for the next section, as such most of the views and ideas from that section will be used, however the concepts provided by Wanless and Robson, will not be used much, while Wanless shows a good way to go about asset creation there are concerns that fixing some of the texture issue will be less productive than other methods, similarly Robson's methods are aimed towards pre-rendering, due to this assets have larger polycounts and doesn't take into account placement in engine or interior design.

3 Practical Example and Testing

For this section a mixture of the methods mentioned above will be used to find a fast and easy approach to a modular level. A sci-fi space station level will be constructed from modular assets in UDK to test the effectiveness of modular assets and find any issues that may not have been covered within the previous sections, following from this example another small game scene consisting of a few hallways will be created to conduct specific test such as workflow benefits and performance benefits.

3.1 Planning

3.1.1 3ds max planning

As mentioned early on, planning is very important and is the first stage to this practical example. Autodesk's 3DS max modelling software will be used to create very simple block parts to make a quick outline of a level, these will be made on a grid size of 16 in 3ds Max as unreal allows this unit of grid size.

If the level designer has not created any plans yet then creating basic primitives first can help them create a level outline, for this example there has not been a level plan so will start by creating these primitives. To help the level designer the most, its best to create as many different possible components of the level, such as walls, floors, ceiling, doors etc.

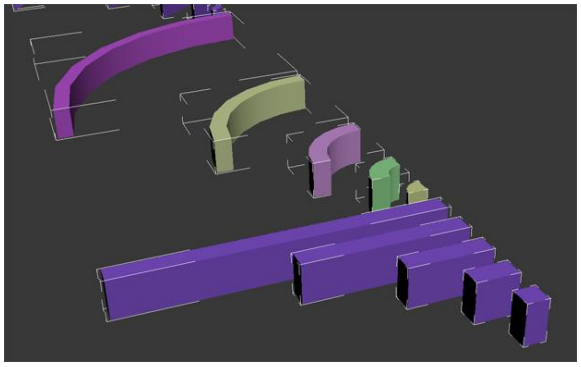


Fig 48

For this example the walls will have a length of 512 units and a height of 128 units with the average player in UDK being 92 units high, both of these fit on the grid of 16.

As the walls where created first, the floors, Doors, Ceiling, Steps and other parts are again created as basic geometry but the dimensions are based upon this first wall, if another part was created first, base the rest of the parts around that instead, making sure that the edges are based on the grid so they are easily placed within UDK.

From this more complex primitives can be created, using an FFD 2x2x2 modifier on the model in 3DS max, vertical and diagonal variants of the original assets can be quickly created, however it's also important to keep the FFD control points on the grid too.

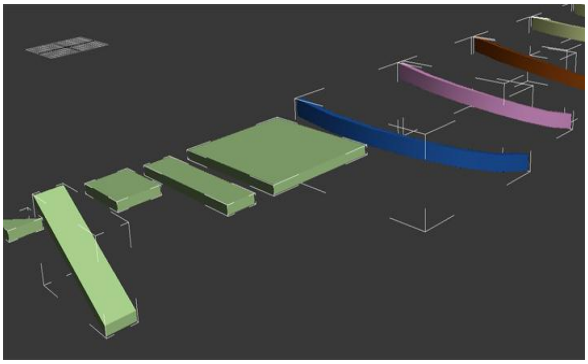


Fig 49

Along with these variants for this example curved walls have been created in many different diameters along with their variants using the FFD modifier again. There may be other sizes or basic shapes needed, create as many as might be useful and give them a good naming convention to help the level designer recognise what they are designed for.

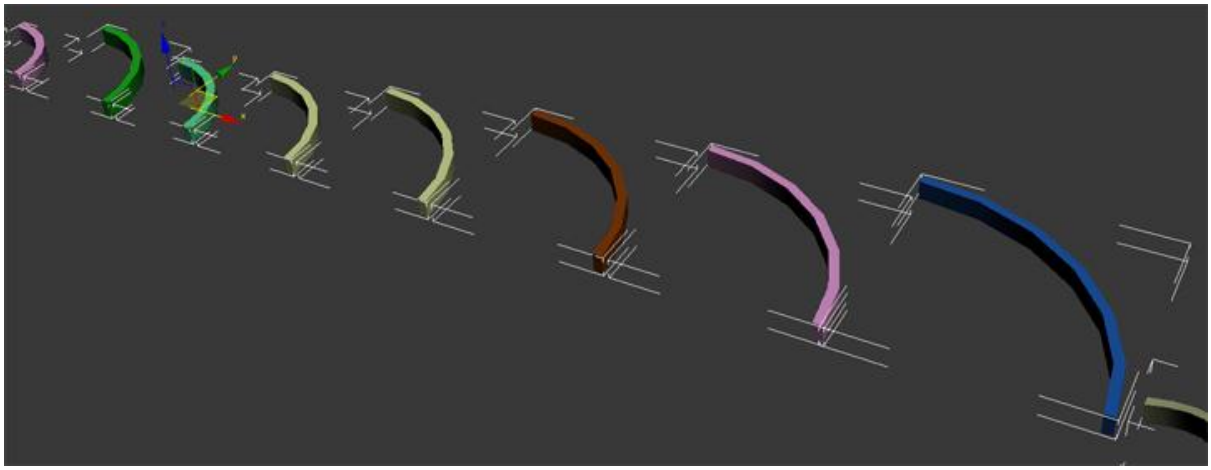


Fig 50

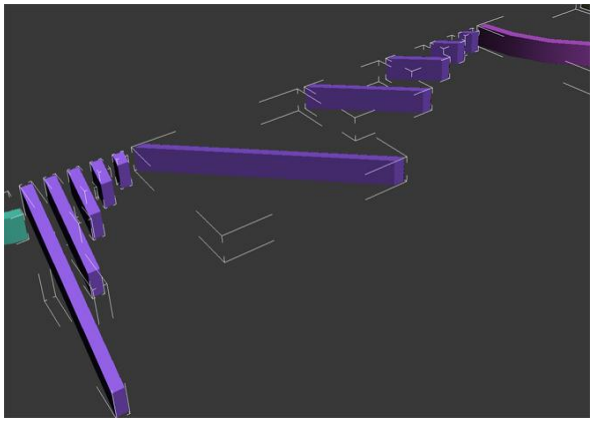


Fig 51

If there is already a level plan made on paper or other media, create these basic shapes to compliment the level design size and shape. This will help speed production up as early feedback can be given such as if an area is too small or too high, possibly to the extent that gameplay can be tested to find if the level layout works or if changes have to be made.

These are then exported out to your game engine model type, depending on engine there may be different steps to exporting, the main important thing to do at this point is to have the pivot point in a corner of the model that is on the grid, use this corner for all the assets created to help with continuity. For UDK the pivot point used in the engine for static meshes is 3DS max's world centre, so all that is needed at this point is for each object to be moved so the corner is on the world origin within 3DS max and exported individually, this process is fairly simple with snapping to grid points and snapping to vertex is turned on

3.1.2 Implementation of planning objects

Once the objects have been exported for use they need to be imported into engine, this part is easy for UDK as it's a simple task of clicking import in the content browser. These assets can then be placed within the level, with these basic set of primitives a basic level outline can be constructed, at this point useful feedback can be given back to the artist from the level designer based on what objects are used the most and what objects are probably not going to be needed at all. This will help organise and get a good basis of a game as time wouldn't be spent on an asset that is not all that important.

If the level is already planned out by other means and this section is not needed, having the primitives from section 3.1.1 can still be useful when creating the game ready assets in section 3.2. but is not necessary.

Fig 52

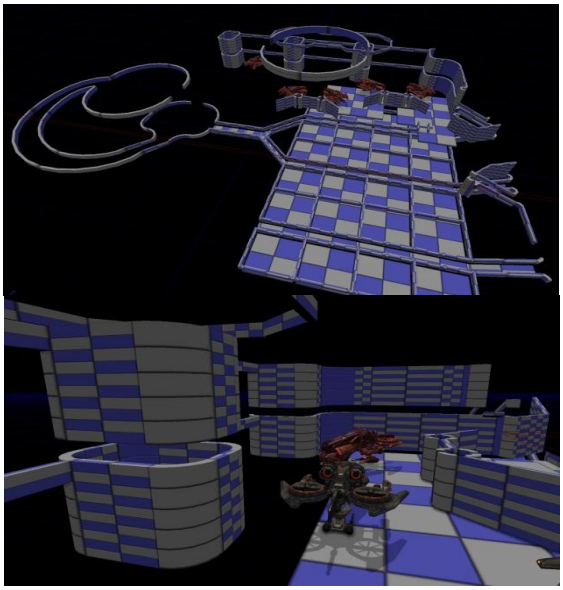


Fig 53

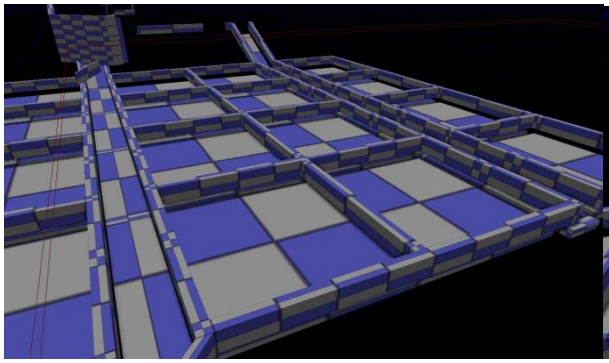


Fig 54

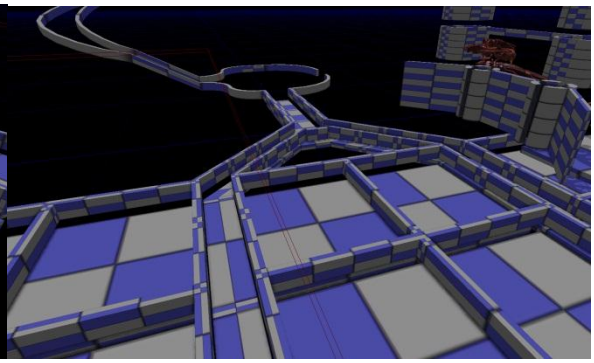


Fig 55

3.2 Modular asset creation.

3.2.1 Creating assets in 3DS max

Once the level designers have the planning objects the game ready assets can be created, at this point it's best to start with the most common items such as a section of wall, something that is guaranteed to be used, this could be the assets that the level designer has said they are going to use the most if by this point they have given feedback as suggested by Epic (2011)

The first piece to be made is the wall section, the basic primitive that was given to the level designer was used as a basic starting place, the rest of the scene was hidden so it was easier to work with this one asset, the primitive was duplicated and used to create the asset itself, as such this insured that it was the correct size and was still on the grid as this was the most important rule to follow from the research gathered, making sure that the edges and corners didn't move the asset was created through the use of several tools such as cut, bevel, extrude and chamfer. Within the confines of this space though there is a lot of freedom to create the asset.

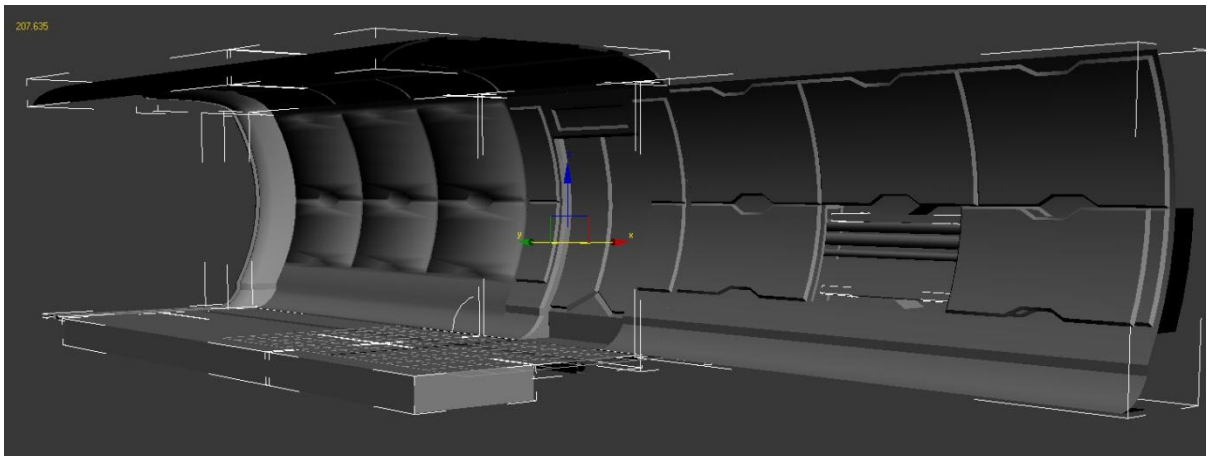
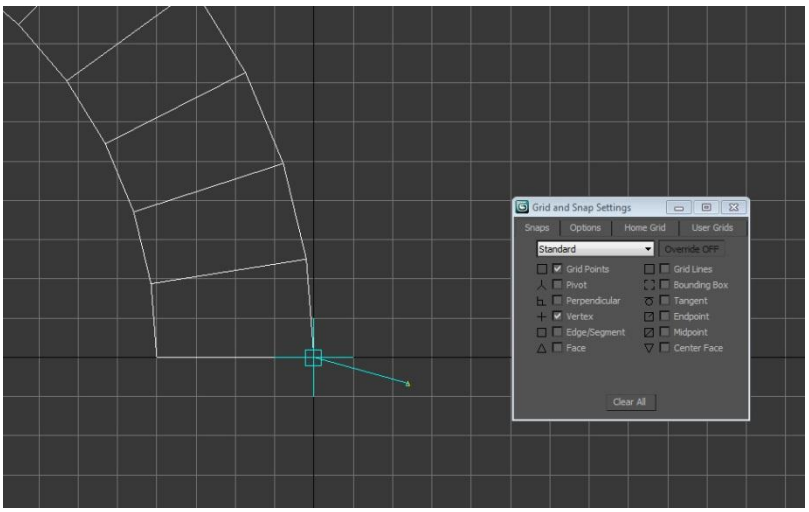


Fig 56

Whilst creating the assets there was the issue that occasionally the model came off the grid, to fix this the edge that had gone over was manually positioned back to the correct place with the assistance of the snap tool, snapping the edge vertex to the grid.

Fig 57



With a couple of the assets this made them look odd, this was down to a fairly large size change compared to the original planning object, instead of scaling it back up to the original size a second accompanying asset was created to fill in the gap, these two assets could of easily been combined together to make it a singular asset, however having them separate gives the level designer more options when creating the level by giving them another asset to use with.

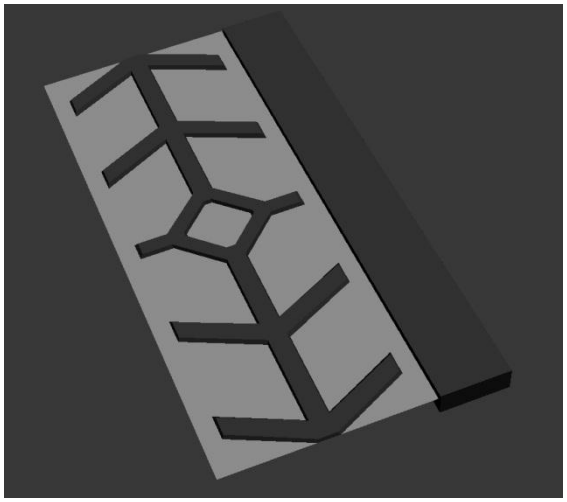


Fig 58

At this point it's a good idea to keep the assets relatively simple, more detail can be added later if time permits, however with whatever detailing is added, an objective look is taken at the model to find out if the asset could be broken down more to become more modular.

Below is an example of this situation, the wall was created as a singular piece with all the detail attached, however this was then broken up into smaller parts, the wall detailing, pipes and wires where all separated, this allows the combination and swapping of different parts giving the level designer more assets to play with and be more creative.

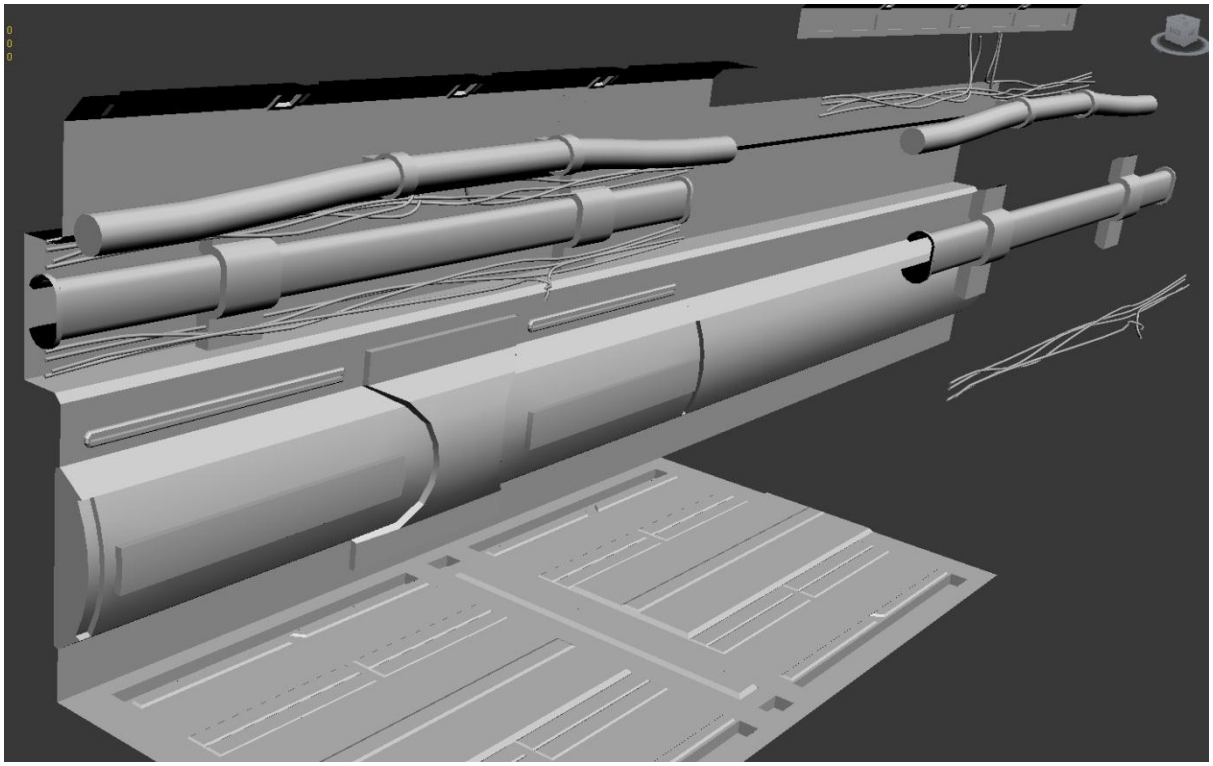


Fig 59

It's also important to keep these parts modular too, if they touch an edge of the specified object then the opposite side has to be the same so they align together.

The wires where created using renderable splines, these started off with a line spline, holding shift creates a straight line this spline was then converted into an editable spline with more points added throughout, however making sure the end points are kept it the same location.

The middle points to this spline where then moved around and each one converted to a Bezier corner to round the hard points off, these where made in bunches of 3 or 4 then several versions of these bunches where created with the middle points at different locations, the pivot point, while not set at the corner vertex of the model was instead put in the corner of the corresponding wall. This insured that they stayed on the grid and easily moved into the correct position within engine having the same pivot point as the wall. Once the spline sets where complete the make renderable in viewport checkbox was checked and then converted to an editable poly ready to be exported to UDK.

One issue with this would be that they wouldn't be easy to position for any other use, however there are no plans to use these assets within this example and as such would be quicker the way stated above.

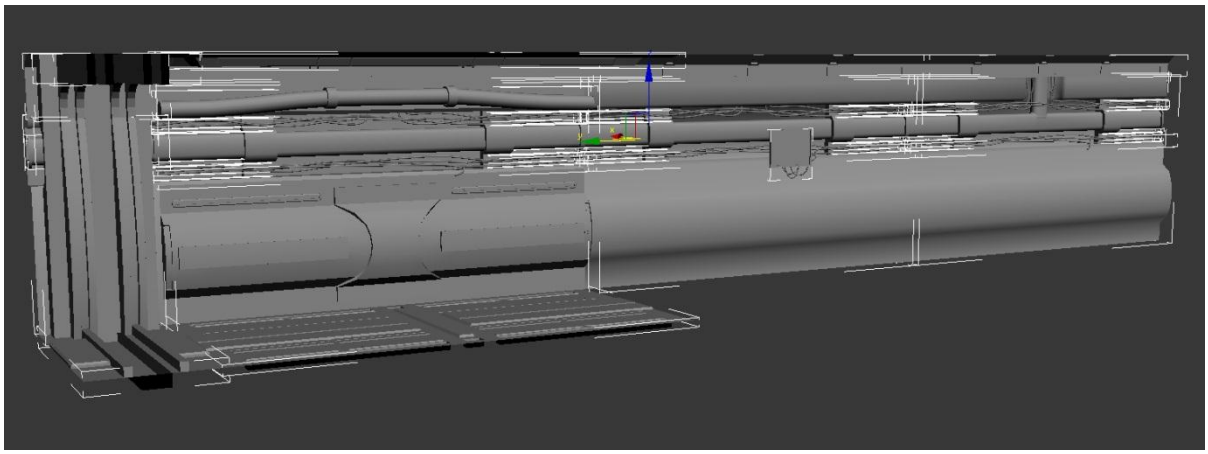


Fig 60

Another important asset to create are the end pieces, these are used to give a natural finish rather than just stopping at some random point.

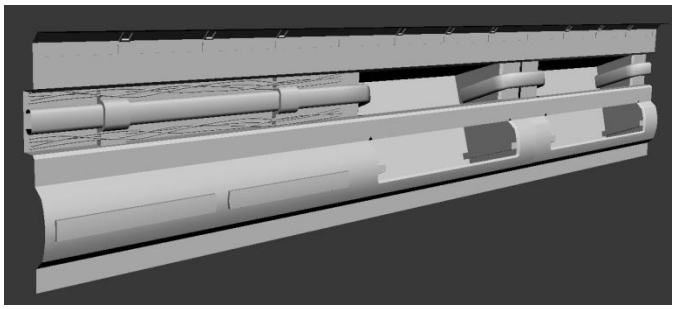


Fig 61

The other variant of the pipe was a corner piece set to rotate up 90 degrees to make it look as if it was supposed to end, this gives the level designer control over what sections have or have not got the pipe over, again because this pipe was straight they matched up on both sides allowing for it to be used in a modular fashion, only rule to follow is to make sure both ends are on the grid so there are no gaps between the sections.

A couple of versions where created in total for this set to add variation, as these where originally part of the main asset in some cases they could use the same texture sheet, as such the performance hit on memory was minimal in these instances, for those that didn't fit on the same sheet, small textures where used.

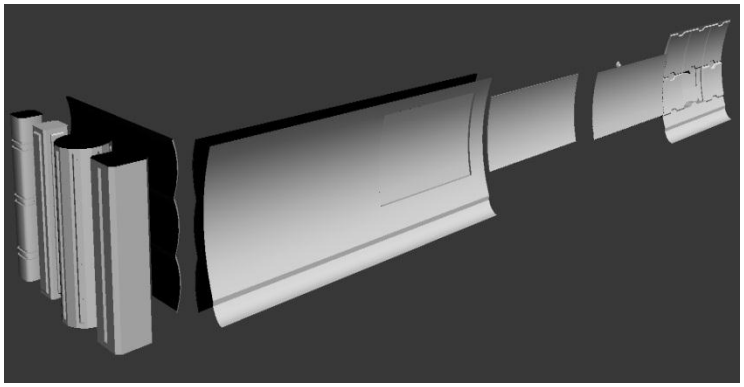


Fig 62

These last points have helped create diversity and break up the repetitiveness of the assets even before they have got to the game engine or even texturing stage, going some way to solving the issue of a repetitive level.

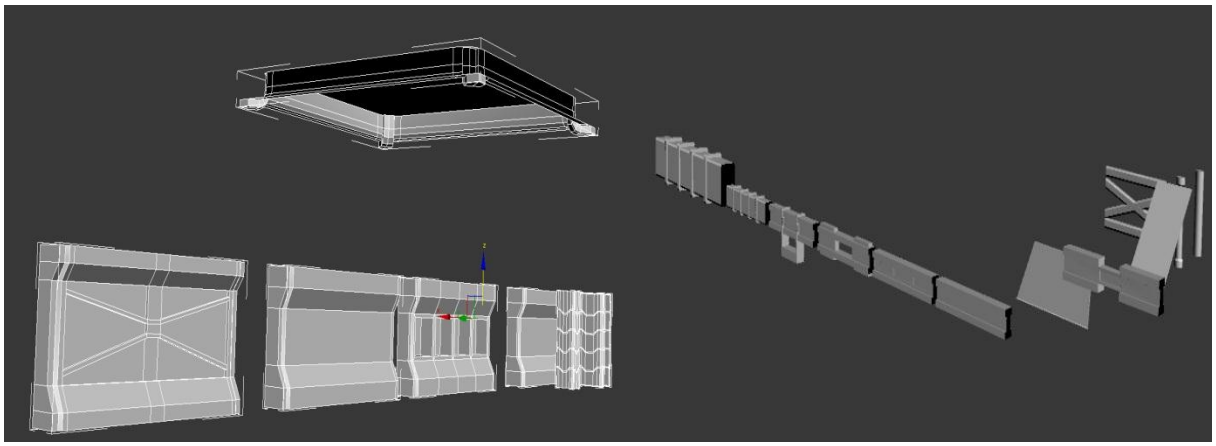


Fig 63

Fig 64

3.3 UDK Level construction

At this point the texturing and unwrapping can be done at the same point as the level creation, once the new UV's are done the new model can replace the old one quickly within the games engine.

As some of the texturing work is done through shaders, they have to be in engine first, as such the document will cover the level creation first.

Each mesh was imported into the engine separately, if the mesh doesn't animate then it was exported under the .ase file format and imported as a static mesh, if the object had animation attached to it, then it was exported using the actorX tool and imported as a skeletal mesh. All textures where imported as either .tga's or PSD's, UDK has the capability of importing PSD's and backing them down into one texture from their separate layers, this allows them to be edited in Photoshop and easily be updated in the engine. All textures are at game friendly resolution, these are dimensions that are the power of 2.

An issue occurred when importing a skeletal mesh and its animation, when first imported into UDK the engine doesn't use any movement information from the animation, there was a check box that allowed the use of movement from the animation as well as rotation, as the door in this example used both then the b_usemovement is ticked to allow the use of movement.

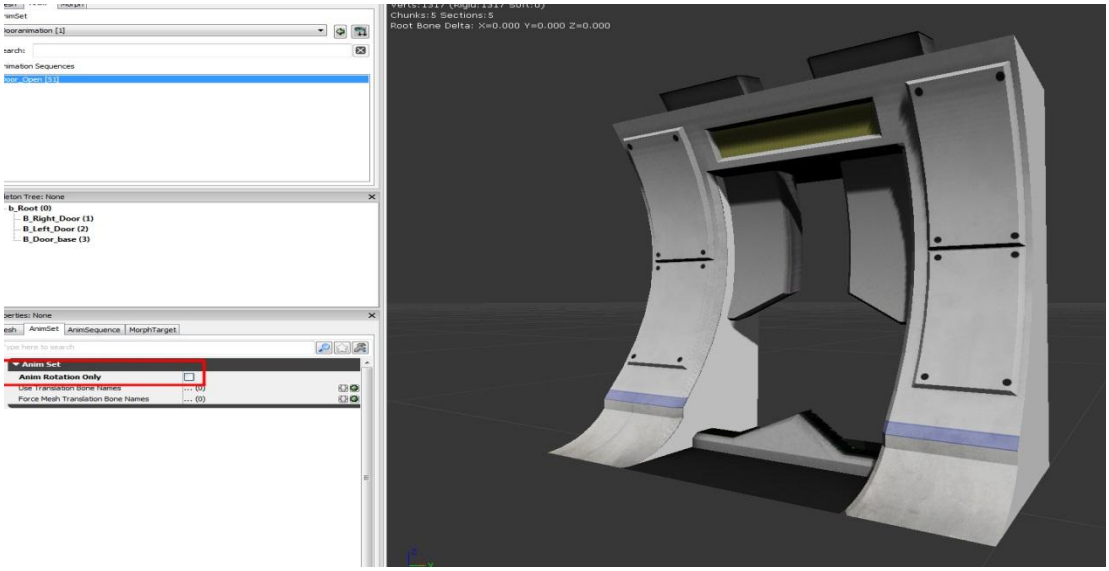


Fig 65

For this paper only a few assets where imported and placed at a time, this is because the assets where created a few at a time and then imported in stages, this would be different if there was a team working on the project, instead these tasks would be given to specific job rolls and sections would be worked on simultaneously.

The first assets to be imported are the Living Area set (assets named LA_), more specifically the major level outline assets such as the corridor walls, floors and ceilings. These where placed and moved to their location following the base level design created with the basic shapes, the straight walls where the first to be placed down 3 main corridors, at certain intervals where the basic level outline stated bulkheads where placed to break up the repetitiveness.

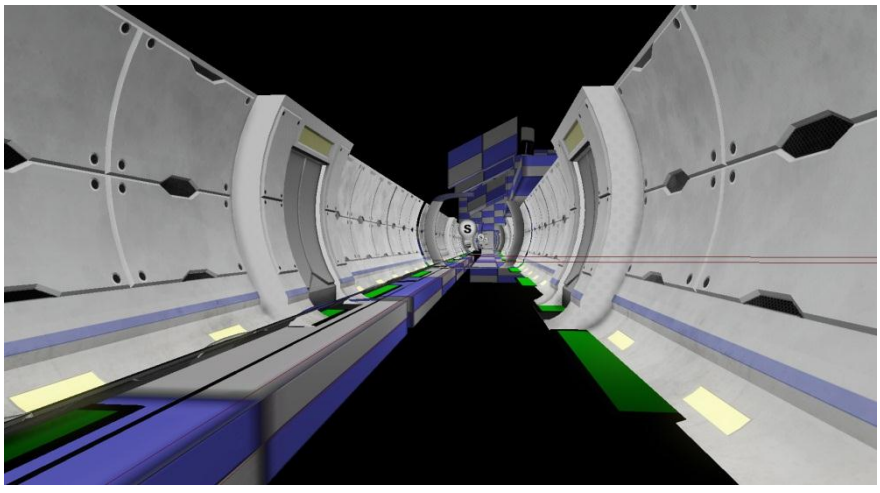


Fig 66

An issue that came up at this point within UDK specifically is the way shadows are created, there seems to be an issue with mip-mapping within the engine

Mip-mapping is where the engine takes the full size texture for the model and then scales it down and uses it in the distance where less detail will be visible, this saves memory and increases

performance, however when 2 objects are flat and lined up, in the distance there are dark shadows appearing when there isn't any expected to be, an example of this can be seen on the polycount website(2010).

The solution for this seems to be to make assets with bevelled edges, this may only be related to UDK so it might be an idea to test this within the engine being used, if it occurs and there are programmers on the team, request them to try and fix this at the engine level, or use the workaround mentioned above.

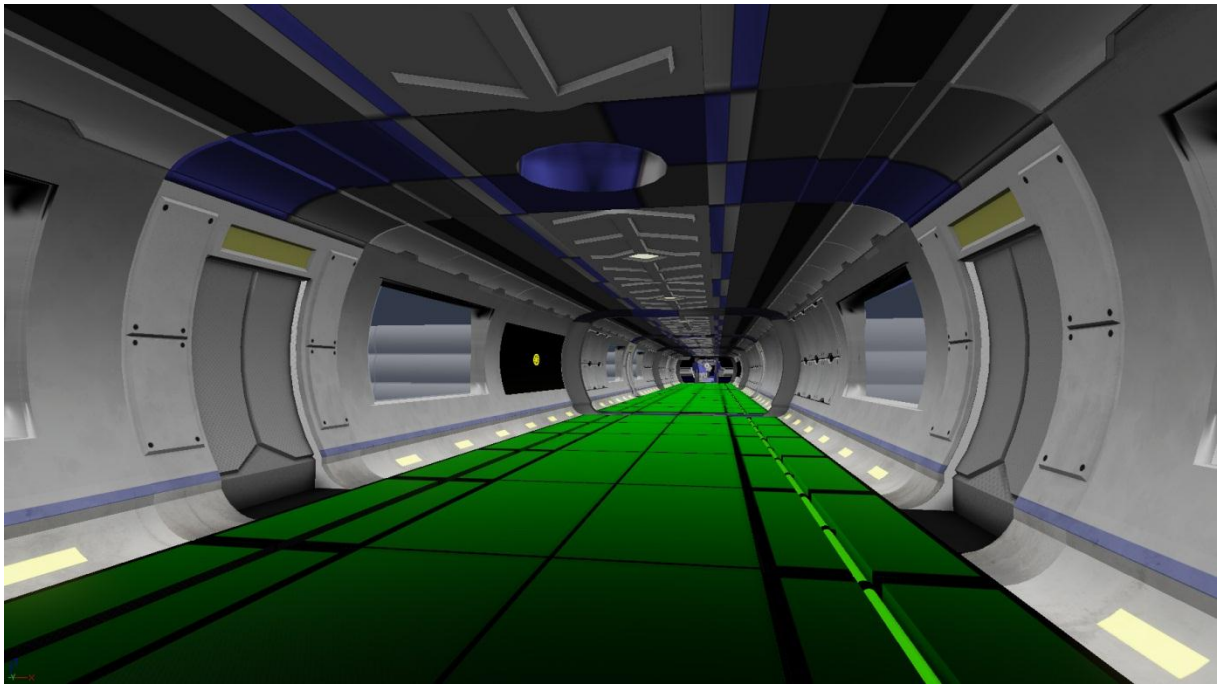


Fig 67

Following on from this the diagonal walls were placed in, these where made from the original walls in 3ds max using an FFD modifier, this became the first major problem while developing the level, as stated in Epic games UDN, when working with diagonal pieces the length is longer and needs to be compensated.

With the level design plan having the bulkheads directly opposite each other there is no quick solution to this, rotating the unreal asset 45 degrees meant that the edges of the wall didn't meet up correctly and also the length was incorrect, though this last issue could have been solved with fiddling around with the scale tool. Several different solutions where thought up with in 3ds max however due to the shape of the wall asset being concave, getting a viable shape working with no gaps became a big deal, below are the solutions explored.

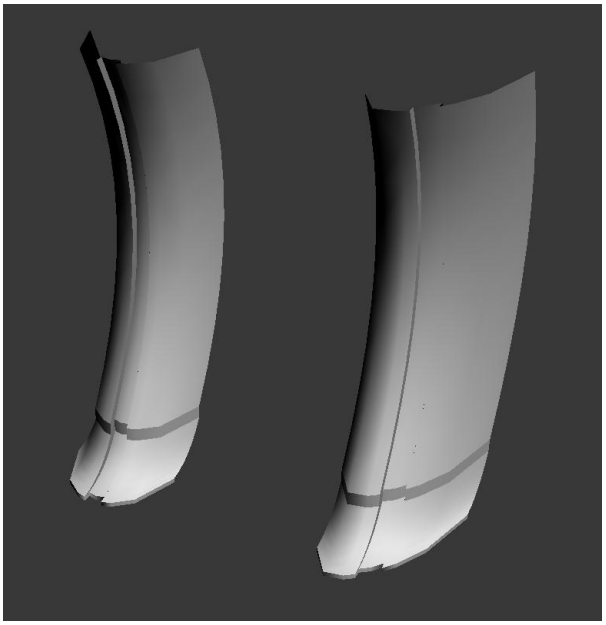


Fig 68

It was chosen that an FFD modifier would be used as this was quick and didn't have too many side effects, the only noticeable problem being the panels not extruding out as far as the normal straight walls do.

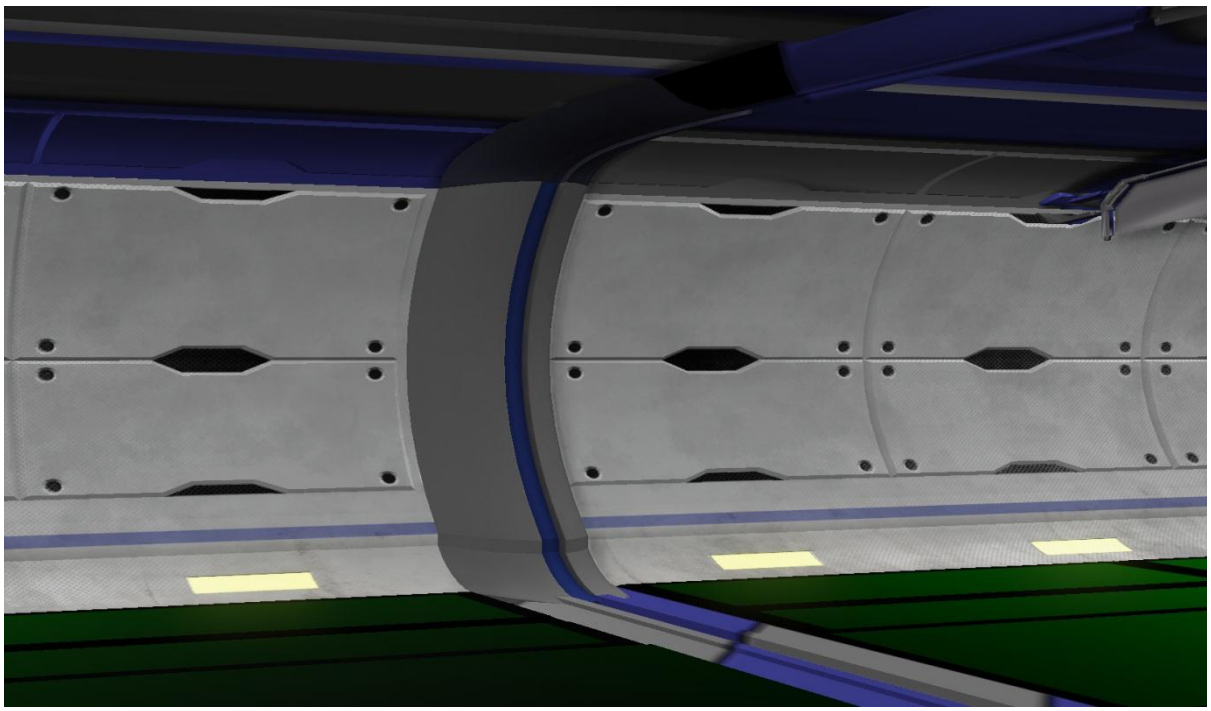


Fig 69

The corresponding floor assets didn't run into as many issues as they have a more simple shape being flat, as such an FFD modifier worked well, this made lining up the floor between the two sections easier in comparison to the walls.

With the new angled wall another problem arose with the bulkheads down this corridor, whereas before the walls have been in a straight line the bulkheads symmetrical design that continued to follow the shape of the main wall worked well, however with them now being offset by the 45 degree angle they didn't line up well, leaving gaps due to the concave nature of the wall so a new bulkhead was created to compensate.

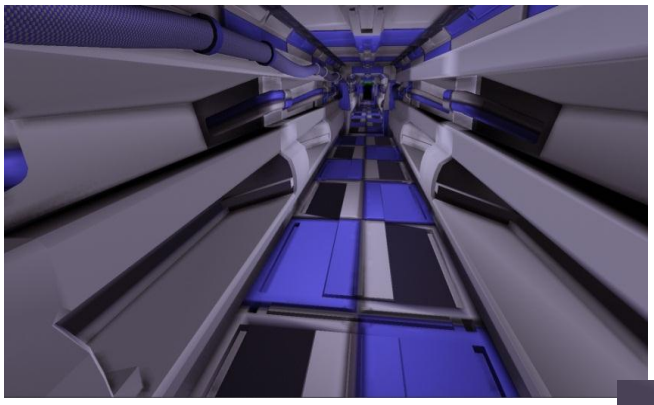
Again using the FFD modifier to build this bulkhead, because of this, the roof also had to be re-done in the same manner to match up with the bulk head walls.

The rest of the set was created using these techniques, while this is a modular set, its ability to modulate is very low, where things didn't line up more assets had to be made to the correct specifications to fit the gaps, thus adding more time and removing the creativeness of the level designer forcing them to use specific assets at points.

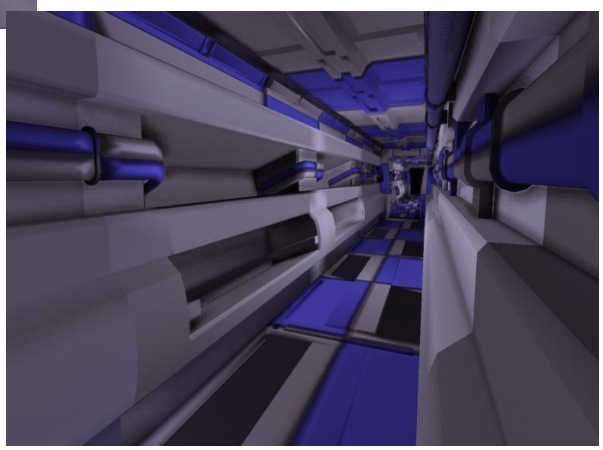
Fig 70

The next set was designed slightly different, as the main issue with the last set was the fact the walls where concave, lining up diagonal pieces caused many different issues, as such this set will be more generic in shape and aim to avoid this problem. This should speed up production not having the need to go back and create assets to fix these problems

This second set is designed for an engineering section, this idea for this section will to have a very tight and close feel with narrow corridors compared to the living area, the feel will also be darker and dirtier with more dents, scratches and detail added, these will have a modular aspect of them so they can be used with each other easier than the last set.



Again for this set the standard 128 units high walls where used, this time though the floor was half as wide, the main wall was created using the same techniques too, using a combination of extrude, cut and chamfer to get the shape, however keeping the wall straighter than before to avoid issues of having to make specific parts fit each other.



The same main sections that where created for the living area was also created for this engineering section, this time more emphasis has gone on the modularity of the detail.

Fig 71 above Fig 72 right

After their creation in 3ds max they were again imported into UDK and the beginning of this second sections was created, this was done in the same method as before but this time using more props to create variation as stated earlier in section 3.2.1

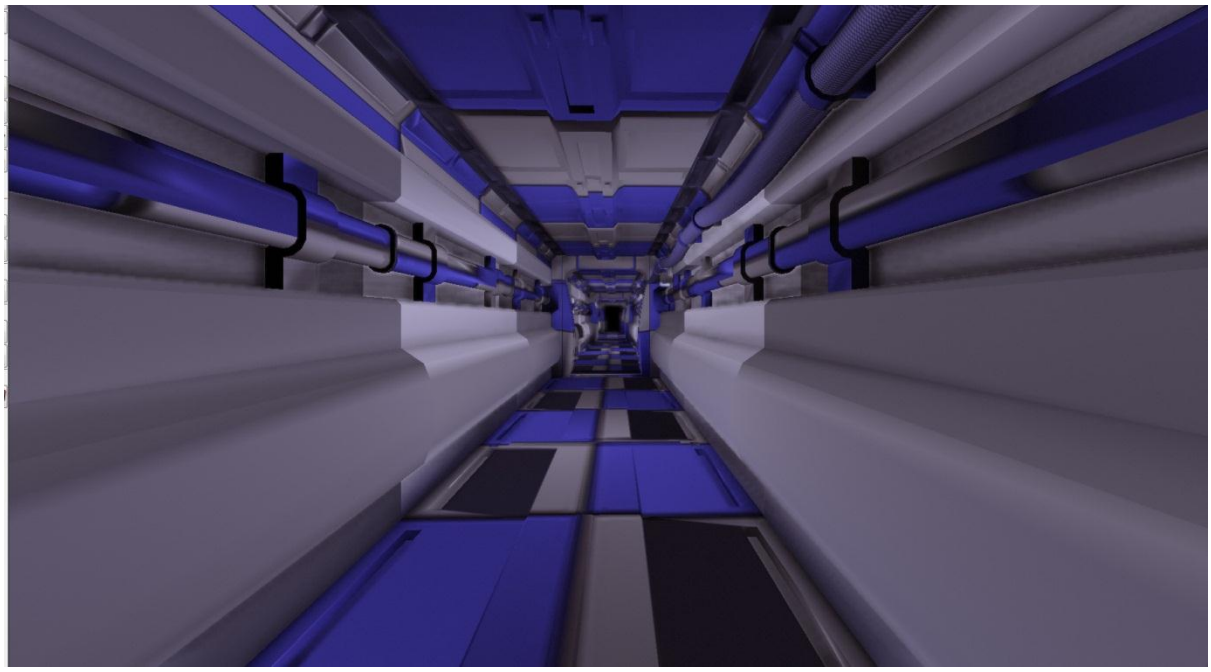


Fig 73

While this section was more modular than the living area, compared to the UDK levels these are still not as modular. At this point time was running short and to fully explore modularity to its full potential this second set, the engineering area was dropped so another more modular set could be constructed.

When comparing UDK's assets to the previous sets there is one large noticeable difference, the assets in UDK are somewhat generic and blocky, the walls are used as floors and ceilings and the support beams used as door ways with other props being used for different purposes, these assets are based on basic shapes, as such the ends are created in a way that they do not require specific assets to finish that section off and as such they can be intersected together and still look fine, these assets also modulate in all directions instead of just sideways like most of the first 2 sets do.

The key point for this set is that they didn't have huge forms and shape to them, that they were generic enough to modulate like the assets in UDK, instead this set contains a lot of pillars, beams, pipes and general generic pieces.

This time, instead of having a level planed while making it in 3ds max the perspective of a level designer was took, so the models where created as such as described in section 3.2.1 but the level design was left till all the assets where created, then building the level from the separate parts.

With this method the assets became naturally more generic and was a lot better for the level creation processes, duplicating bits and placing them together, there were still certain aspects of this set that could have been more modular, the wall is a good example here while it fitted with more objects, the style was dissimilar to the other assets and so while it may of looked ok, its modularity aspect was lower than the rest. This time around there wasn't many assets needed to be created to

fix areas, only a couple of specific things such as lights for the ceiling and energy field for the ship hanger.

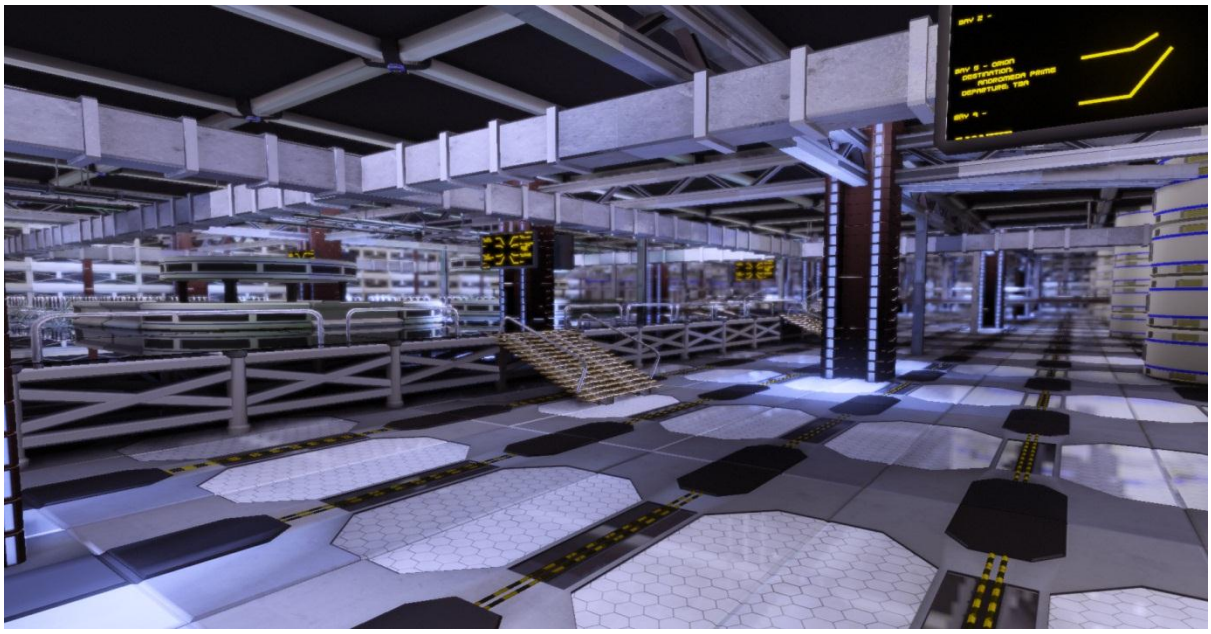


Fig 74

Fig 75

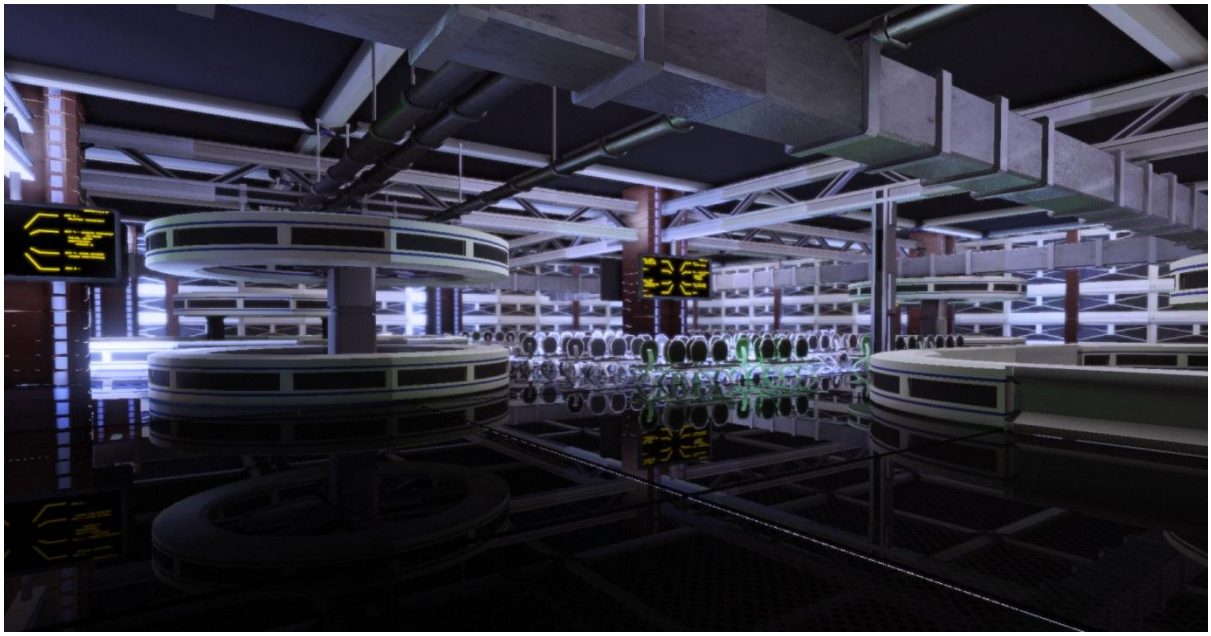


Fig 76

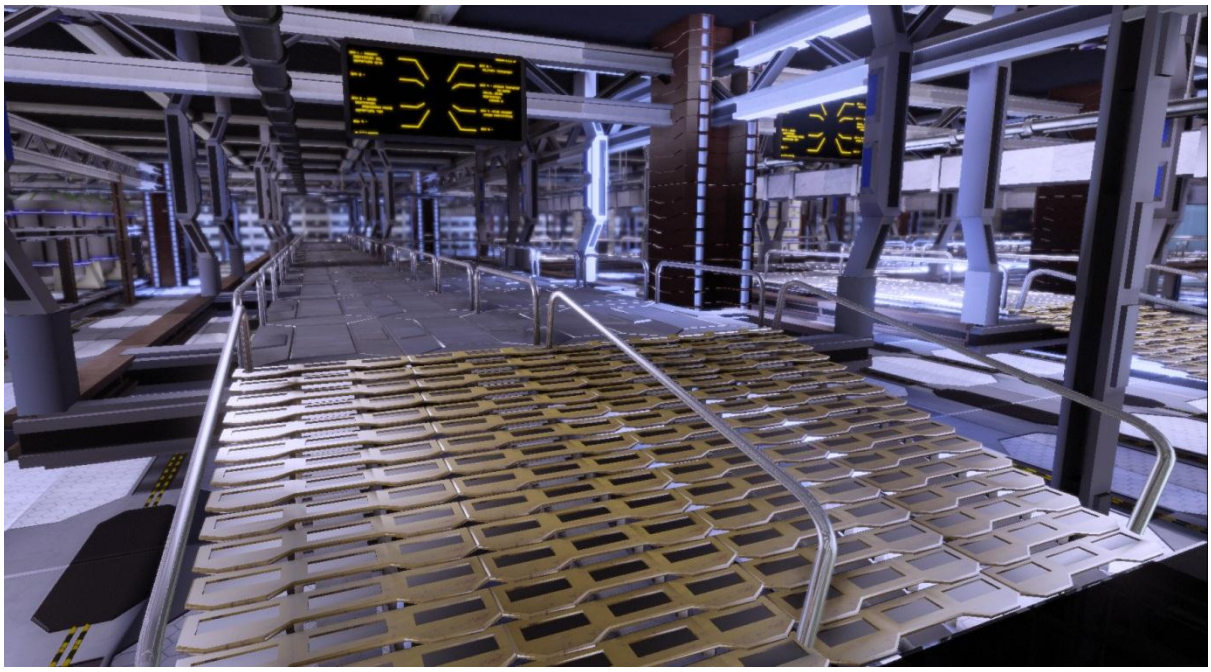


Fig 77

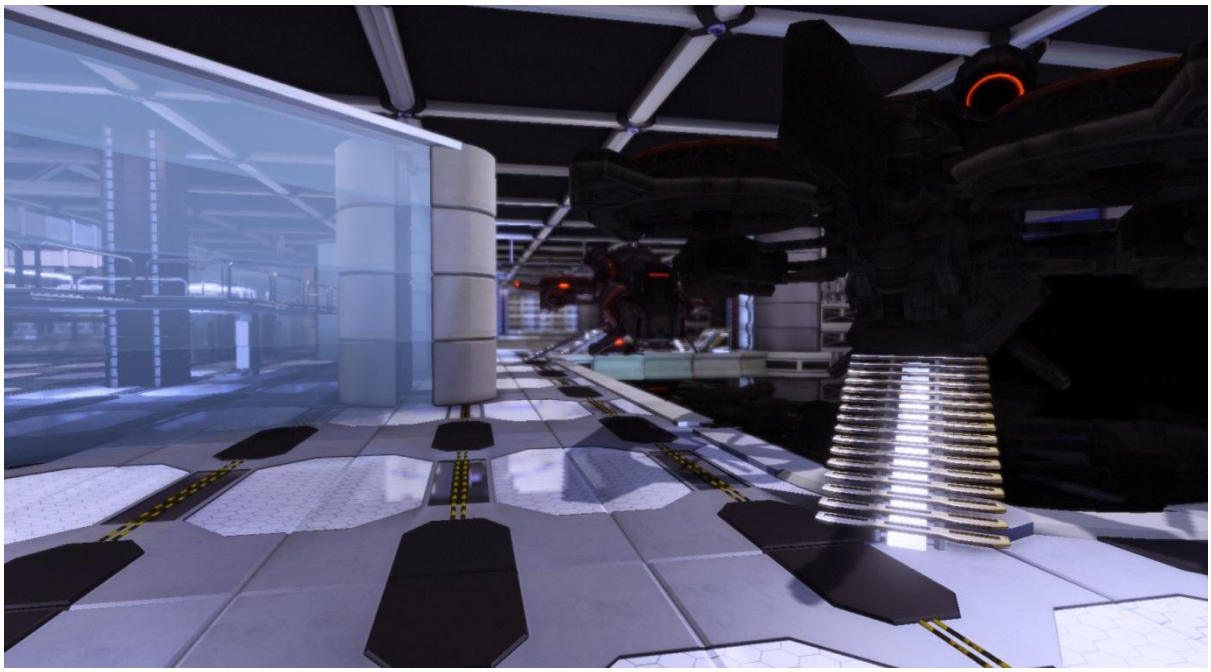


Fig 78

3.4 Unwrapping and texturing.

There are no special pre-requisites for this and can potentially be approached normally, however there are some circumstances where it may be beneficial to break the textures into layers, just like in Photoshop, however in the games engine instead.

Most texture artists compress the entire texture down into a singular material, however as the same asset is being re-used, this would mean the same texture would be used on all of these pieces making the level look very repetitive, in this instance textures can go a long way to make these assets look unique.

In UDK and other games engines they are able to do complex calculations based on materials, these are called shaders.

A shader is a term for mathematical equations done to a material, they can be written in code or in the case of UDK there may be a visual editor to assist in the creation, the end product alters the input of the calculation to create an effect based on the calculations done, some of these are common and have been around for a long time, some well-known shaders include bump maps, normal maps and specular maps, they also get a lot more complex and can start creating effects such as reflections in mirrors to fire material used in particle effects.

With these shaders UDK is able to do similar abilities as a texturing application such as Photoshop, however as these are done in real time, if the artist is not careful, they can dramatically affect the performance of a game or level. However this opens up opportunities that may possibly help a modular scene look more unique.

There are 100’s of ways shaders can help due to them being completely customisable and are not limited in complexity, however only few examples will be explored for this research paper.

This first shader comprises of 3 parts, the main diffuse, extra detail and text

Starting at the back of the chain there is the main texture node, from here it is then multiplied by a second texture, this one is set as standard to white and adds no detail, however instead of a standard texture node this one is a parameter meaning that in a material instance this can be swapped out for another material that might have more detail.

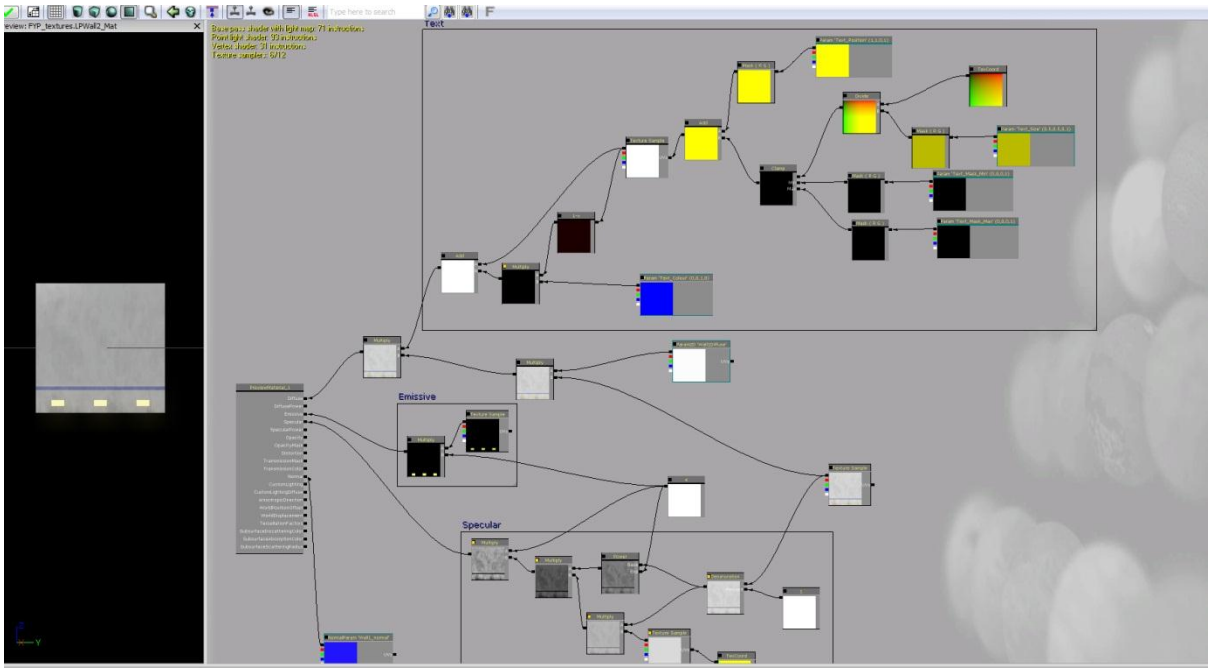


Fig 79

From here it is then multiplied to a large shader chain that controls text, this text depending on the values entered can change colour also what the text says, this is done by having a texture with set writing on and masking out the parts that are not needed, then moved to the location desired, as such these can be controlled through material instances, this means that extra detail can be quickly added without the need to create the entire shader again, it also brings up the possibility that because it is using the same calculations at time, that each material could be processed quicker.

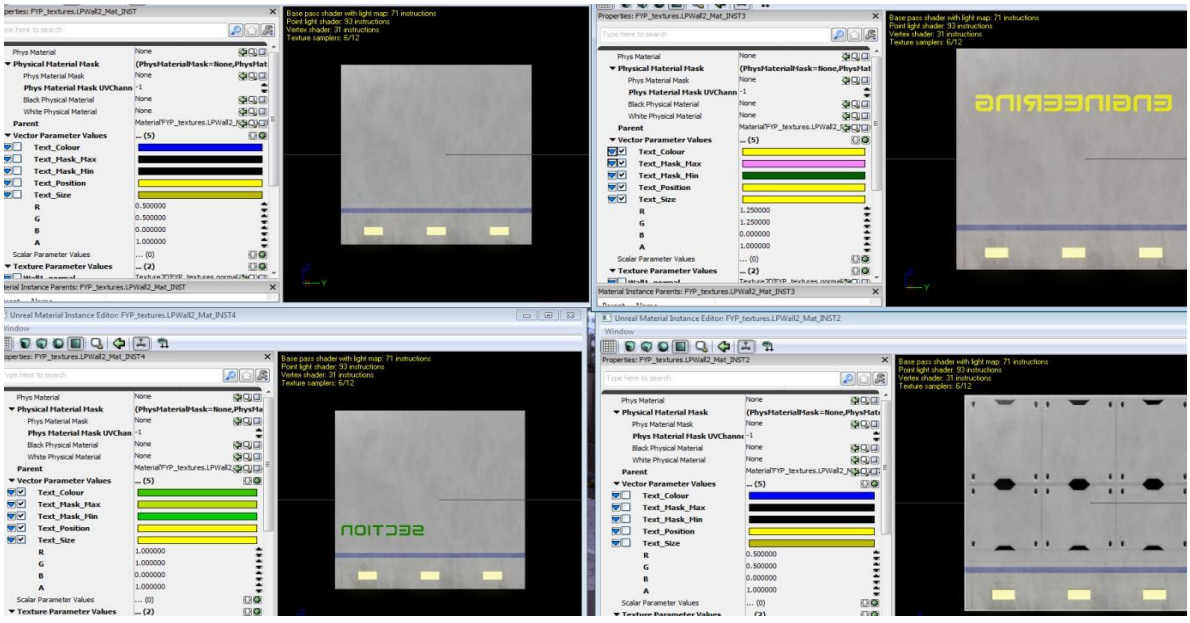


Fig 80

This shader technique took some time to setup, many assets may not need complex shaders like this, instead these can assets can use a standard texturing technique and simply plugging the different textures into the apropriate slots.

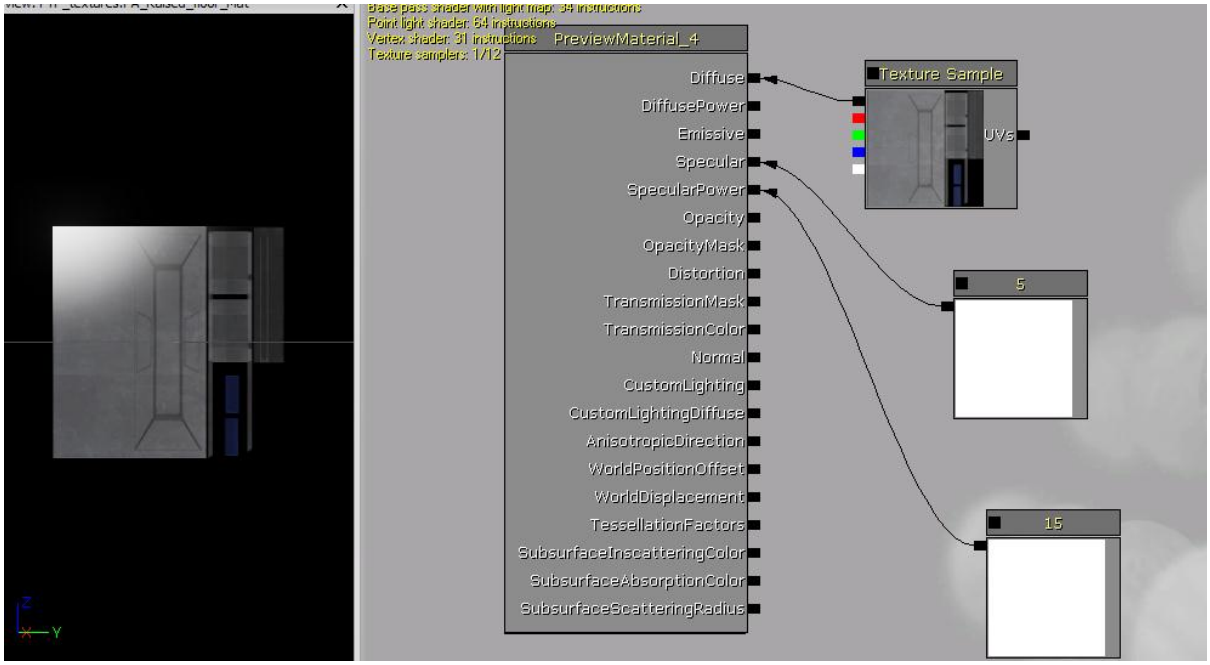


Fig 81

Similar techniques where used for other materials, using masks to control different aspects of the material.

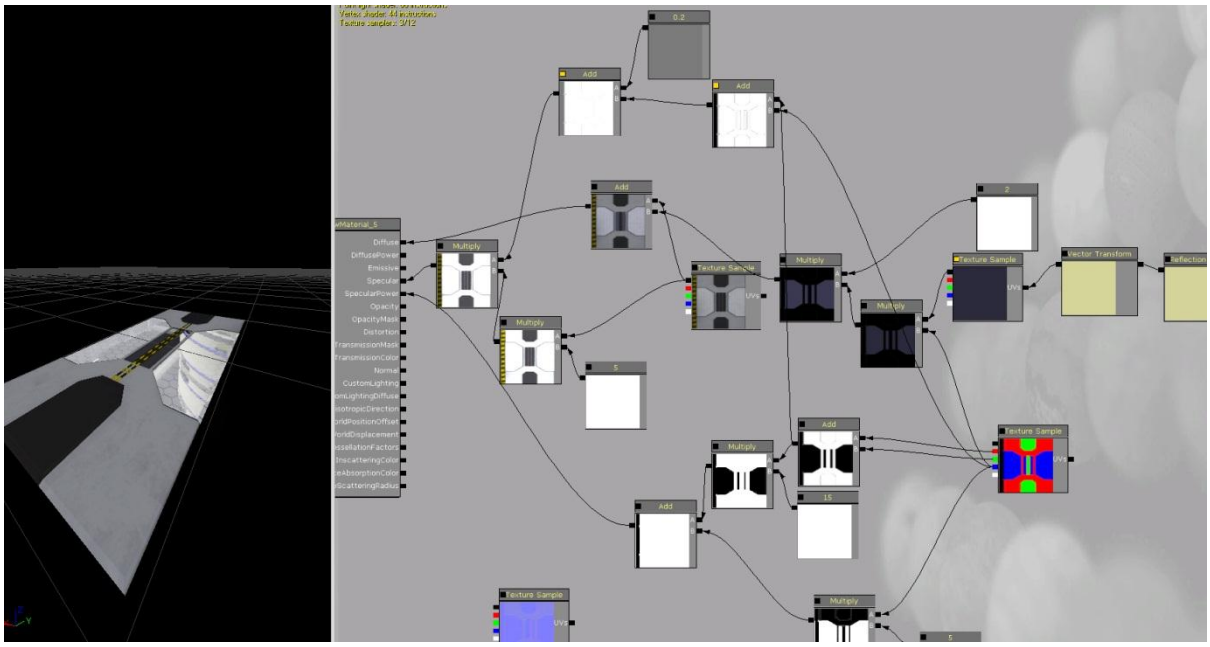


Fig 82

3.5 Research practical support.

To back up the research covered in the paper, alongside the portfolio practical example to show how far modularity can be taken, a small level will be created in both a modular and none modular method to demonstrate the workflow

3.5.1. Test Planning

To demonstrate this the modular assets that where created for the previous practical example will be used again in this section, however to demonstrate the performance benefits a copy of these assets have been made, they have then been scaled and taken of the grid, as such these are similar to non-modular assets. This should demonstrate the performance and workflow benefits within UDK.

As the asset creation is not all that much different from creating normal assets for a level, the time difference here is negligible once an artist is used to following a few rules, while modularity may seem restrictive at first, it's not all that much different from making a unique asset that needs to be in a certain size to fill a gap in a level, the only time that modularity may cause a longer development time is when assets are needed to fix problems in a modular scene. Depending on the level in question this may also be true for non-modular assets, as such the time difference here will be down to the quantity of assets required to be built.

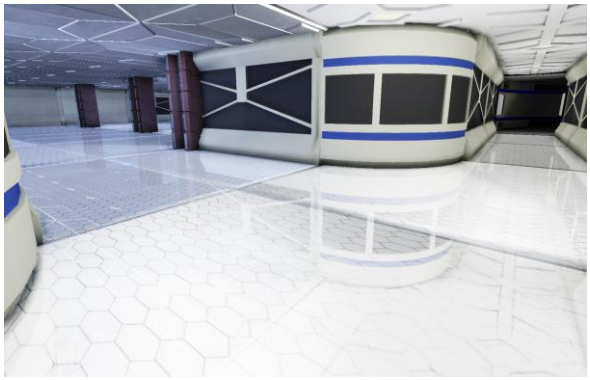
The time it took to create the building blocks at the beginning of the project was recorded, this time is extra time added to the workflow stage of an artist, however it may be the case that this is done any way for a none modular level, furthermore the benefits provided to other areas of the game creation process may outweigh the time spent here.

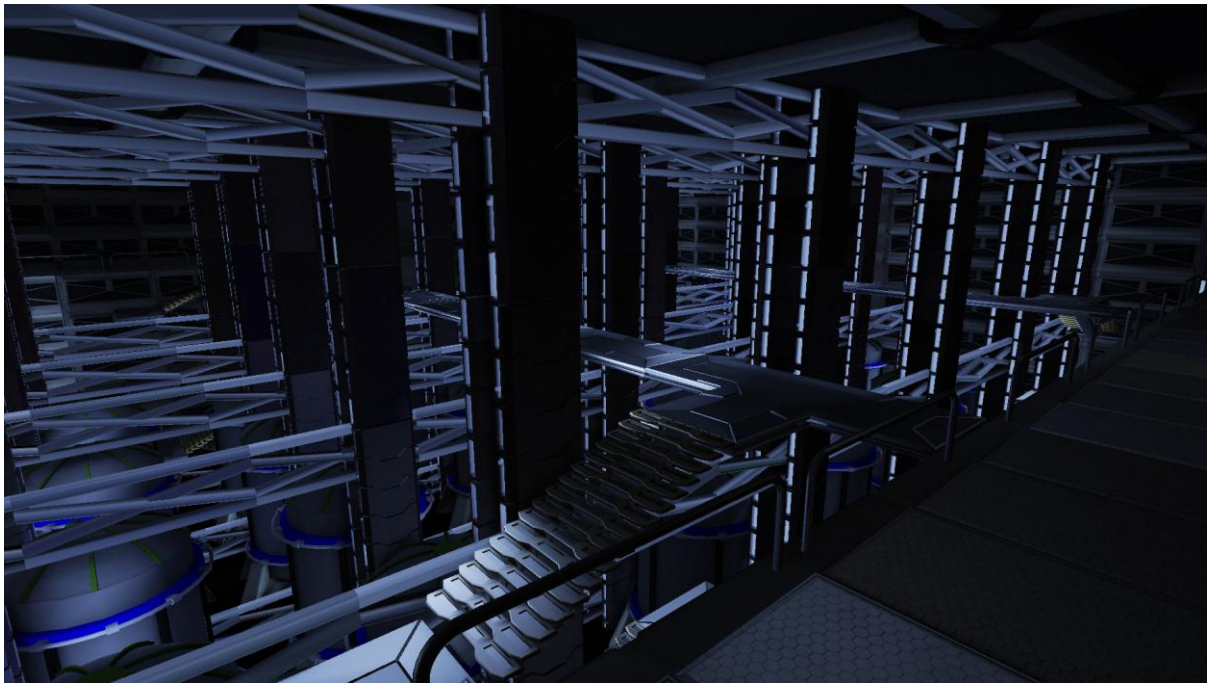
3.5.2. Testing Method

To test the workflow benefits within UDK a small level will be made to demonstrate a modular scene, this will not look aesthetically pleasing due to it being a test, the same time that was spent on this scene will then be spent on a non-modular scene and comparing the difference between the amount complete, these objects will be taken of the grid and uniformly scaled to be similar to a non-modular asset would be.

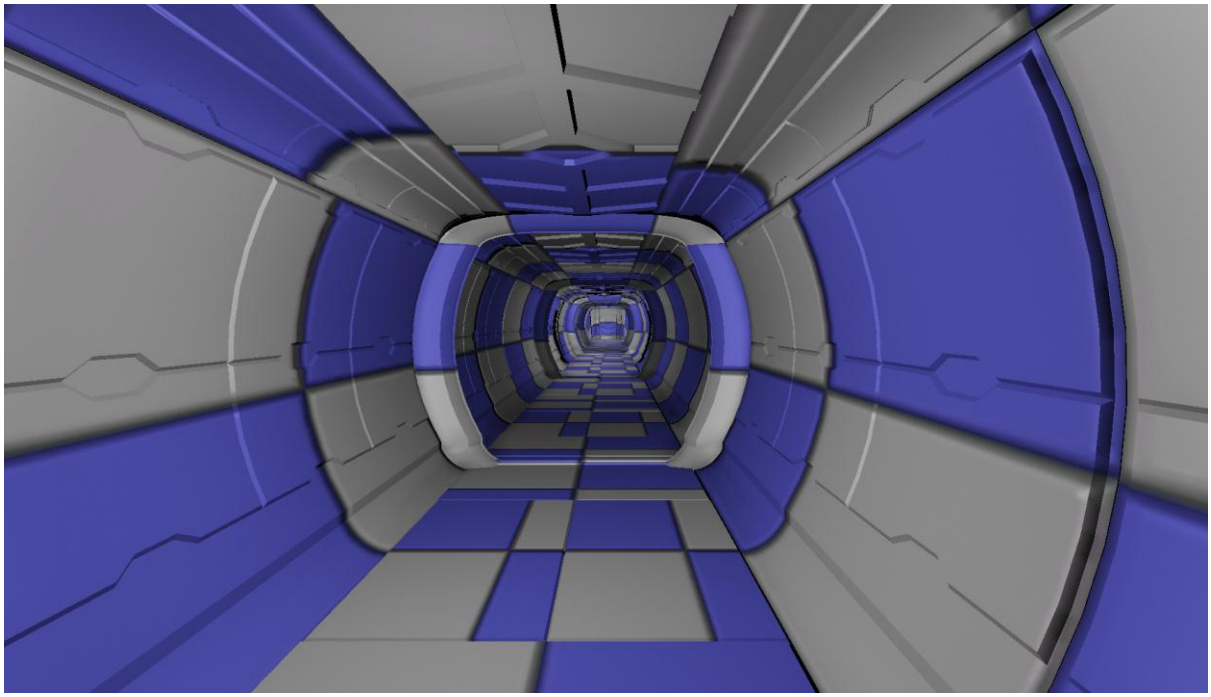
3.5.3. Testing results

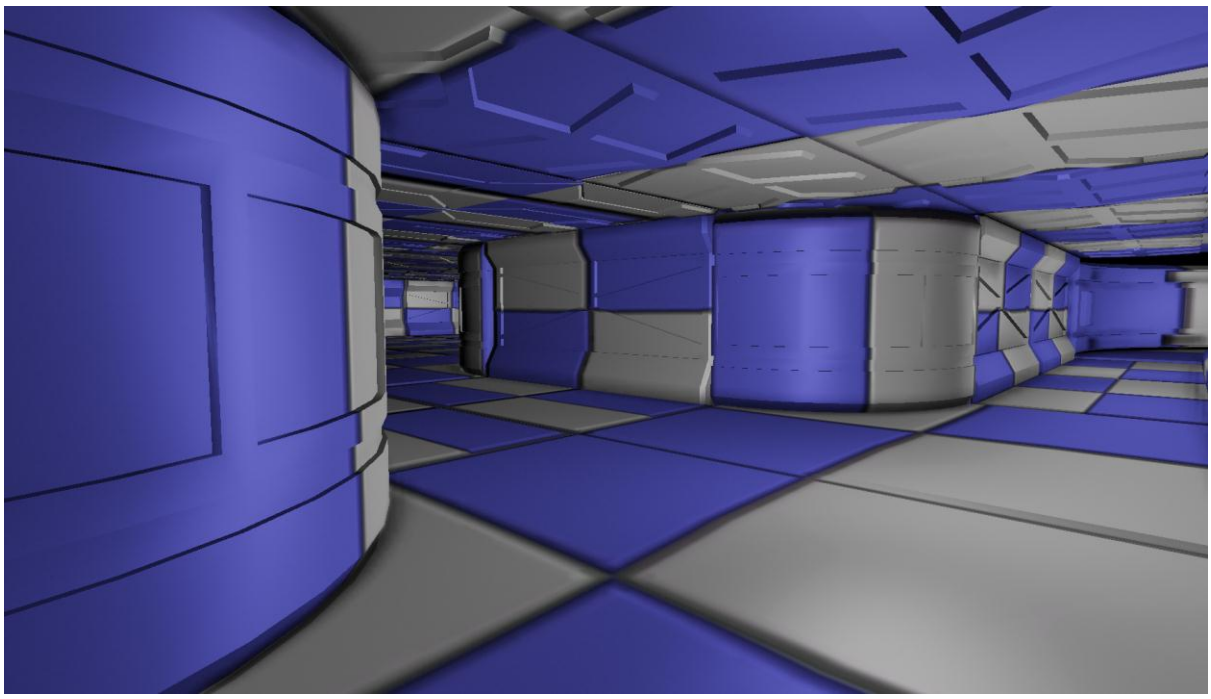
Below is a picture of the test level, it took two and a half hours to create including the lighting

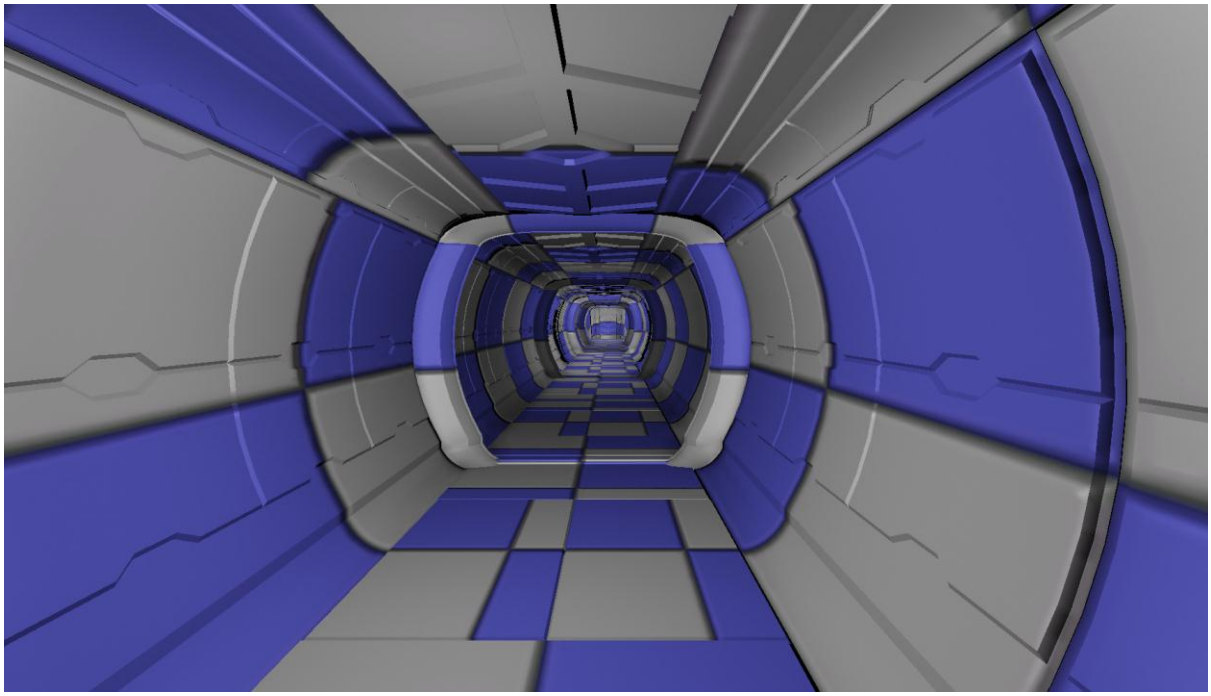




Below is the scene created in a non-modular attempt, the same time was spent on this as the above.







In this scene, only 395 objects where placed in the same time, there was a lot of trouble with getting them to fit together due to them not being on the grid, however the overall outcome of the non-modular scene wasn't too difficult to do, that is a 75% increase in time taken, note that in the renders above the objects use the UDK standard material in comparison to the modulated scene, this was because the materials are not being considered here, as such these where new objects that needed the materials re-applied to all objects.

This test demonstrates it is far quicker to use a modular aspect at times, as aligning assets perfectly is a lot more simple, however what this test fails to consider is that in a real life example some of these assets will be grouped together and as such would not take as long as it did in this test.

4 Conclusion and recommendations

Overall the project was able to demonstrate the benefits of modularity, some areas more than others.

The research covered a lot of different methods artists use and tips that might be useful to anyone who has read this document, though there are methods that conflict with each other in places the research has tried to show that they are all valid ways of construction but there may be only special circumstances for their use.

Researching into the graphics pipeline became complicated as this is something that until now was unexplored and there was little previous knowledge to draw from, it was made more difficult due to the fact that the area of research this was located in was different from the usual area of research an artist looks into. As such it became apparent that each engine has its own feature set, something that was made aware near the end of the project with UDK not supporting batch rendering, by this point there was insufficient time left to test performance in another engine. Even though this was the case the research and accompanying the practical project successfully reached its aim of proving that there are performance benefits to be had using modularity. Due to this it's highly recommended to communicate with the programmer in charge of the engine that is currently in use to find out the best way to optimise and implement modular assets, possibly requesting features that have been mentioned in this document to be added to the engine.

Due to the above reason it was hard to test the performance of modularity as it would of meant in the practical testing stage that 1600 separate objects would have had to of been imported separately and placed individually, if they were grouped together then the way the engine batches objects together would have been the key factor.

One aspect of research that could have been looked into more is modularity outside of games, due to time constraints this section was only lightly covered and not much was gained from it in the end, there were more areas of research in mind to expand on this such as car construction and also real life space station construction.

As the modular construction method was something that had not been attempted before there was issues in the beginning of the practical example where the design of the assets where hard to come up with and ultimately didn't work out as a hugely modular set with it only being modulated by the sides and not in all 3 dimensions, however it did stand to prove that modularity can be used in many different scenarios, there was a plan to make a 3rd set of assets, this was started after the 1st set to improve on the modularity aspects of the 1st set, however while it was more modular it was dropped due to time constraints and the fact that while it was more modular, the modularity aspect could still be improved more, as such this was then demonstrated in the 3rd set.

Within the first set there where a few issues relating to the design, due to the shape of the walls it became increasingly hard to create rotated sections, especially relating to the floor and the bulkheads. A good recommendation is to keep the building blocks as generic as possible, not having hugely concave wall sections such as those in the 1st set and also make it so each asset doesn't

require an exact same shape to meet up with it next to it, as such the 1st set had a curved theme running throughout it, this limited its use between areas and sets and to a point making it more difficult to get transition sections between different sets.

With more time the practical side would have been taken further with more detail being pushed into the objects, along with this detail props would have been added to break the scenes repetitiveness up more.

Finally while not too important, it would have been good to of been able to upload the level to the internet for people to review and comment on it, however due to the length of time it took to create the assets there wasn't the time neither the capability to upload it as the final file size was large too.

List of figures

References.

VG charts Network. (2011). *UT3 Sales figures.* Available: http://www.vgchartz.com/worldtotals.php?name=unreal+tournament&publisher=&console=&genre=&minSales=0&results=50&sort=Total . Last accessed 2011.

VG charts Network. (2011). *Halo Seires Sales figures.* Available: http://www.vgchartz.com/worldtotals.php?name=halo&publisher=&console=&genre=&minSales=0&results=50&sort=Total . Last accessed 2011.

VG charts Network. (2011). *Halo Seires Sales figures.* Available: http://www.vgchartz.com/worldtotals.php?name=mass+effect&publisher=&console=&genre=&minSales=0&results=50&sort=Total . Last accessed 2011.

Oxford University. (). *Dictionary Definition.* Available: http://www.oxforddictionaries.com/definition/module?view=uk . Last accessed 2011.

Perry, L. (2002). *Modular Level and Component Design.* Available: http://udn.epicgames.com/Three/rsrc/Three/ModularLevelDesign/ModularLevelDesign.pdf . Last accessed 2011.

Lin, Lentz, Sturgill, Reed. (). *Using Workflow Techniques and Modularity.* Available: http://udn.epicgames.com/Two/WorkflowAndModularity.html . Last accessed 2011

Hagedoorn, H. (2010). *AMD GPU marketshare declining again.* Available: http://www.guru3d.com/news/amd-gpu-marketshare-declining-again . Last accessed 2011.

Reed, A & Lin, T. (). *Mesh Optimization.* Available: http://udn.epicgames.com/Two/MeshOptimization.html . Last accessed 2011.

Cebenoyan, C. (2004). *Graphics Pipeline Performance.* Available: http://http.developer.nvidia.com/GPUGems/gpugems_ch28.html . Last accessed 2011.

Wloka, M. (2005). *Batching 4EVA.* Available: http://http.download.nvidia.com/developer/presentations/2005/GDC/Direct3D_Day/D3DTutorial03_Optimization.pdf . Last accessed 2011.

Rege, A. (2004). *Optimization for DirectX9 Graphics.* Available: http://http.download.nvidia.com/developer/presentations/GDC_2004/Dx9Optimization.pdf . Last accessed 2011.

Carucci, F. (2005). *Inside Geometry Instancing.* Available: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter03.html . Last accessed 2011.

Wloka, M. (2003). *"Batch, Batch, Batch:" What Does It Really Mean.* Available: http://origin-developer.nvidia.com/docs/IO/8230/BatchBatchBatch.pdf?q=docs/IO/8230/BatchBatchBatch.pdf . Last accessed 2011.

Robson, C. (2010). *MT7 modular structure building in 3ds max.* Available: http://www.3d-palace.com/2010/11/29/mt7-modular-structure-building-in-3ds-max-the-bunker/ . Last accessed 2011.

Wanlass, T. (2011). *Modular Building Workflow.* Available: http://www.3dmotive.com/product-modular-building-udk . Last accessed 2011

Mader, P. (2005). *Creating Modular Game Art For Fast Level Design .* Available: http://www.gamasutra.com/view/feature/2475/creating_modular_game_art_for_fast_.php . Last accessed 2011.

Smith, G. (2010). *Modular Floor Tiles - Outlined in Shadow .* Available: http://www.polycount.com/forum/showthread.php?t=76476&highlight=modular . Last accessed 2011.

McCloud, K. (2009). *Grand Designs Season 6 Episode 9.* Available: http://www.channel4.com/programmes/grand-designs/4od#2921453 . Last accessed 2011.

Jones, S. (2011). *Technical performance question .* Available: http://forums.epicgames.com/showthread.php?t=784144 . Last accessed 2011.

Bibliography