Super-modular Textures
Comparisons in Practical Applications

John Svetoft

Bachelor Thesis
Game Design and Graphics VT 2012
Supervisors Jonatan Österberg
Tony Oscarsson
Abstract

The purpose of this thesis is to compare the products of two different work flows in a practical environment. The work flows in question are those of building environments using super-modular textures versus the more conventional method of unique textures. The tests will try to establish whether or not the theory of the super-modular work flow holds up in practice, i.e. if the result is actually as optimized has the theory outlines. The data of interest is gathered through tests in the Unreal Development Kit, which is a free version of the commercial Unreal engine. Results are compiled into graphs in order to give a clear overview of the differences between the two products. The results show that in the context of the tests performed the super-modular workflow allocates less memory than what can be achieved using uniquely mapped assets and that the average draw calls remain the same regardless of method.
# Table of Contents

1 Introduction

2 Background

3 Theory
   - 3.1 Hardware
   - 3.2 Bitmaps
   - 3.3 Modular Textures
   - 3.4 Super-modular Structure
   - 3.5 Textures
     - 3.5.1 Normal Map
     - 3.5.2 Colour Map
   - 3.6 3Ds Max
   - 3.7 UDK

4 Purpose

5 Method
   - 5.1 Group 1A & 1B
   - 5.2 Group 2A & 2B
   - 5.3 Group 3A & 3B
   - 5.5 Group 4A & 4B
   - 5.6 Group 5A & 5B

6 Results
   - 6.1 Group 1
   - 6.2 Group 2
   - 6.3 Group 3
   - 6.4 Group
   - 6.5 Group 5

7 Analysis

8 Discussion

9 Conclusion

10 References
   - Literature
   - Sources
1 Introduction

“Less is more” is an aphorism usually associated with an architect by the name of Ludwig Mies van der Rohe (Mies van der Rohe Society 2012). He was most likely referencing his own architectural preferences, but it's one of those sayings that just happens to hold truth to me across a wide array of subjects, even in the quagmire of computer graphics. I find that by following this one simple guideline, if you wish, helps create environments that are not only pleasing to look at, but also gives the game a certain clarity, i.e. there's less clutter on the screen. More specifically so in video game graphics and it's somewhat of a philosophy that I myself as followed rather vigorously when working on developing the visuals for games. But lately I've been pondering if there is even more meaning to this aphorism, if it can be applied to the technical bits of graphics development, beyond the philosophical interpretation of it.

It got me thinking a lot about how I build environments for games. With computer hardware rapidly developing, both in stationary desktops as well as hand held devices, the amount of geometry and effects that can be rendered in real-time is steadily growing, exponentially. During the years it has become possible to build a lot more very detailed and high resolution assets to populate an environment. I realized that I have been putting in a lot more assets into my environments, at the expense of making them smaller in order to cope with the added pressure on the hardware.

This sent me along the path searching for a way to perhaps use less, to create more. After some research I believe to have stumbled upon what might be the answer: super-modular textures.

2 Background

For the purpose of this thesis, what can be described as previous research in the field, or the background, is the method of uniquely unwrapping and texturing assets. There exist a lot of different approaches to this and there is a lot of community experimentation across various platforms. To name what is one of the greatest resources of 3D-modelling theory and techniques there is the Polycount forum, which is an open forum that houses both professionals and hobbyist modellers.

Going over all of the collections would be way beyond the scope of this thesis, so a selected number of articles and threads will be described in short to give a deeper insight into the work that is being done in the field, and in which the super-modular theory is anchored.

What is referred to as the unique workflow within this thesis is the most basic form of asset creation in current generation engines. The workflow is based around modelling a high-poly model, which is a very detailed version of the mesh that usually utilize thousands if not hundreds of thousands times the geometry of the final in-game low-poly model (Holden 2006). The high-poly model is then baked into a normal map, which is normally used to fake the high resolution geometry (Polycount 2012) making a mesh of 100 triangles look as if it was made of 100,000 triangles.

The modular workflow is not based around a specific normal map for a specific object, but rather the creation of one normal map or texture that can be reused over a number of different objects
The super-modular texture takes this one step further, as it is a single set of details that are aggressively reused throughout most, if not all of the assets in a map. The theory this thesis is based on is specifically one recent project and experimentation by Tor Frick of the Polycount forums, where a single texture map (two one colour map and one normal map) was used to create an entire environment and populate it with smaller assets, e.g. books, lamps, boxes, microscopes and workbenches (Frick 2012).

3 Theory

This part will briefly explain a few of basics that are needed in order to fully understand the actual process of making super-modular environments and what potential benefits it might have performance-wise. In order to properly explain the potential performance benefits of this work flow, a brief explanation of computer hardware is necessary.

3.1 Hardware

A computer's ability to render 3D graphics depend on three different components. The Random Access Memory (RAM) which is used to temporarily store data, the Graphics Processing Unit (GPU) which is a processing unit specifically designed to handle graphics-related calculations, and the Central Processing Unit (CPU) which is a processing unit designed to handle general calculations (e.g. code).

When a 3D-environment is rendered in a real-time setting (such as a game), the RAM is used to store an instance of all of the different objects relevant to what is going to be displayed on the screen (the RAM is refreshed several thousand times per second). So if a player moves from an indoor environment to an outdoor one for example, the RAM will be flushed and all of the indoor-related assets will be replaced with the outdoor ones.

The GPU will handle most of the graphic specific calculations, e.g. when an object is displayed in the game engine, the CPU will issue a draw call for that object, basically telling the GPU what to draw (Möller, Heins, Hoffman 2008:31). This means that the GPU will draw the object from the RAM and calculate how it will be displayed in relation to its placement in the world and the location of the camera. Each entity of an object will be a unique draw call. So a simple box that uses a single texture will be two draw calls, one for the 3D-mesh and one for the texture, while a box that uses two textures will be cut up before rendering, which results in three draw calls (depending on the engine, it might result in four as some engines will split meshes per texture).
Draw calls can put a very large load on the CPU/GPU as too many would create a bottle neck between the two, resulting in bad frame rate and stuttering performance. As an example we have two different meshes (A & B) that uses two different textures (C&D) in two different combinations. Further calculations and draw calls will be needed depending on a number of different factors, such as if the scene uses dynamic lighting or static lighting, shader complexity and other effects. This may very well vary from engine to engine as they may differ greatly deepening on how it is coded.

To summarize, the RAM is used to store an instance of each model and texture relevant to the scene, while the CPU/GPU performs calculations and the drawing of the scene.

### 3.2 Bitmaps

Another concept that has to be understood in order to explain this workflow is that of a full colour bitmap. In order to create colour in a digital picture, three different layers are blended. These layers are usually referred to as channels. These channels are the Red, Green and Blue, which results in a RGB-colour spectrum. There are other spectrum available such as CMYK (Cyan, Magenta, Yellow and Black), this spectrum is usually used in printing and for on-screen images and colours, RGB is usually preferred.
RGB is comprised of three different channels. Each channel is a black and white version of the image with varying brightness, adding up to a composite colour image (Illustration 2). The theory and workings of this system could of course be described in great length, but for the sake of this thesis the only part that is really relevant in order to describe the super-modular work flow is that a single 24-bit image actually consist of three 8-bit black and white channels, with the option of adding an additional channel, usually known as a alpha channel (making the image 32-bit rather than 24-bit).

3.3 Modular Textures

A modular texture, in contrast to a unique texture, can be reused over and over again for a wide array of different assets. Where the unique texture would map in all of the details, e.g. bolts, hatches, decals etc. in a very specific matter, a modular texture contains these elements in a way that make them non-specific to any particular asset. This can result in a slightly lower texture resolution in of certain areas of the mesh, mainly larger surfaces, whereas smaller surfaces and details can achieve a higher resolution than what would be possible with a sanely scaled unique texture.

The resolution of the modular texture is directly related to the resolution of the mesh. Elements in the modular texture, in this example a couple of small bolts, are of a set resolution which is relatively high in regards to the size of the map. More edge loops in the mesh result in more and smaller surfaces on the mesh, which can then be mapped to the modular details and depending on the size of the mesh, these repeated details come out in a much higher resolution that what would be possible had they been created in a unique texture map (Illustration 3).

The mesh consists of a 512x512 plane, which has been divided into 16 squares. The mesh area consumed by the rivets is relatively small, but since the area of the texture it samples is of a higher resolution than what would be possible had the rivets been mapped uniquely (assuming a sane texture resolution), the result will be much sharper details.

This is the same principle that modular textures are based on, however modular textures evolve around sets of several texture maps with different purposes, e.g. a texture map with generic rock texture that can be reused over several different assets (Klevestav 2010), whereas the super-modular is based on a single, concentrated set of details to be reused.

Illustration 3: Modular texture mapping
3.4 Super-modular Structure

The super-modular structure utilizes the ability to have up to four separate grey scale images in a single bitmap image, there is no set structure for this and what is described in this section is simply the version used for the purpose of this thesis. There are likely a lot of different viable options for what to actually fit into the map. For the purpose of this thesis, the super-modular structure will be described using three different software packages:

• Autodesk 3Ds Max (Max) – 3D-modelling suite
• Adobe Photoshop (PS) – Image editor
• Unreal Development Kit (UDK) – Game engine

While some of the herein described methods and functions may be unique to any of these software packages, the theory remains the same, regardless of program used.

3.5 Textures

The primary texture file is a composite of four grayscale images, each located in a separate channel. As for the secondary texture file, the normal map, the same theory applies. However, due to the nature of the normal map, two of the four channels have to be set, that is the green and red channels which dictate up and down, left and right. The blue channel and the alpha channel can be replaced with other images. The blue channel stands for depth in a normal map and is usually of no use and can instead be replaced with pure white in the engine (in this case in the material editor of the UDK).

Here follows an example of a super-modular texture to clarify the actual theory which will be described through a series of illustrations.

3.5.1 Normal Map

Illustration 4: Super-modular normal map structure
The red channel contains a default normal map pass, left and right normals. The green channel contains a default normal map pass, up and down normals. The Blue channel contains a black and white mask used to separate the metal parts of the texture (Illustration 4).

3.5.2 Colour Map

![Image of Colour Map](Illustration 5: Super-modular colour map structure)

The red channel is a simple tileable metal texture, this will be tiled out over all of the assets adding detail and texture to the asset when viewed in close proximity.

The green channel is a ambient occlusion pass of the normal map, this adds contact shadows and further definition to the asset. This channel can also be used to add dirt to crevices and contact areas. The Blue channel contains different decals that can be mapped on to assets (Illustration 5).

3.6 3Ds Max

In Max the main difference between the modular and the unique work flow is the use of more UV-channels, as the super-modular set up will need at least three. The first UV-channel (Illustration 6) will be used to store the light map unwrap, that is, every surface of the model will need its own space on this UV in as few islands as possible. The reason this UV is set to 1 is because the UDK will calculate specular lightning based on whatever is in that channel. So having a cut up UV and overlapping faces would cause major problems in the engine.
The second UV-channel (Illustration 7) is used to map out the decals, this channel can have as many overlapping faces as need, so decals can be reused several times for the same asset.

Illustration 7: UV-channel 2 - Decals

In this set up the third UV-channel is used to map out the details from the normal map (Illustration 8). This map can have as much overlapping and stretching as needed to properly map in the details. Note that there might be a need to add extra edge loops in the mesh in order to map details properly.
This might mean that the mesh will have a slightly higher triangle count than if it was created with unique detailing.

Illustration 8: UV-channel 3 - Normal Map

3.7 UDK

The material system in the UDK allows the artist to make use of a wide array of functions, which in turn creates a very flexible system. The super-modular structure is based around a master material with several parameters. When an instance of the master material is created these parameters can then be turned on and off or otherwise edited, allowing the artist to change a number of different aspects.
In this example (Illustration 9), the artist is free to change the colour of the different surfaces, turn reflections on and off as well as turning parts of the texture emissive (this will cause them to glow). The checkboxes can turn decals and several material effects, such as reflections on and off.

4 Purpose

The purpose is to conduct a number of tests in an attempt to conclude whether or not the super-modular theory holds up in practice, i.e. does it hold up in an actual game engine. The main questions this thesis will attempt to answer are the following:

- Does the complex nature of the super-modular material affect texture memory allocation?
- Does said nature of the material affect the number of draw calls?
- What, if any, is the difference between the super-modular and the unique workflow in memory allocation and draw calls?

5 Method

The method by which tests will be conducted is a rather straight forward lab exercise in order to collect data. The desired data is primarily the video memory (RAM), with the secondary being the amount of draw calls (CPU/GPU) in order to try and establish the difference, if any, in performance. Frame rate will not be measured as it is a much too hardware specific unit that can differ greatly between different set ups (the frame rate measures how long it takes for a single frame to render, in milliseconds).

Balancing these tests may prove somewhat difficult, since in order to get trustworthy data, things like material instructions, texture resolution and triangle count may be to be somewhat different between the assets. The tests will be performed in groups using assets of similar nature on both
sides and steadily increasing the quantity of assets in increments of five (arbitrarily selected number) to be able to measure the curves relative to the quantity.

The UDK engine counts memory usage in megabyte (MB) and draw calls in an averaged unit where 1u = 1 call, since the count is averaged, values may end up with decimals which for the sake of clarity will be rounded to the nearest whole (≥0.5=1).

Between each batch of tests of one type of asset, the engine will be completely restarted in order to ensure that the collected data is not contaminated by any lingering files in the RAM (simply switching scenes back and forth was found to cause the texture memory in the last opened scene to carry over to the next one, creating a faulty reading).

Each group of assets will be idle for five minutes to ensure that everything is loaded in properly. After which the highest and lowest value will be noted down for one minute for the draw calls and the highest value for the texture memory.

These tests will utilize a stripped version of the UDK, where a blank scene will use 15 MB texture memory, and average at 46 draw calls. These values will be subtracted from the gathered scores after which the final values are added to an index with an accompanying a graph.

The super-modular assets will use two 512x256 24-bit TARGA (TGA) maps, while the unique assets will be using up to three 512x512 24-bit TGA textures maximum, and two 64x64 24-bit textures minimum (these sizes will be determined in relation to the size of the mesh).

The scene will be lit by a single point light and feature baked light maps, the UDK measures light map memory usage in a separate string so the allocated texture memory should be unaffected.

5.1 Group 1A & 1B

Group 1 will be simple boxes, i.e. not an actual object, but a simple box mesh. This is to make a fair comparison between the textures, their draw calls and memory allocation, effectively taking the mesh out of the equation. Group 1 A will be the super-modular texture (A will be used for the super-modular assets throughout testing), while group B will be a unique texture, consisting of one diffuse map, one normal map and a coloured specular map.

The first round will be one box on either side, then five, ten and so on until there are 50 boxes in each group.

5.2 Group 2A & 2B

The second test group will also be using boxes as the mesh component. This group will start off with five boxes on both sides, each box utilizing it's own material instance, i.e. group 2A will have five material instances with different diffuse colours, while 2B will have five material instances each using its own texture maps (three maps per instance, in contrast to the super-modular which will still utilize the same two texture maps).

This group will start out with five boxes, increasing the amount in increments of five until 50 is reached.
5.3 Group 3A & 3B
This is the first group that will utilize actual modelled meshes of various designs. The design of the meshes is largely irrelevant, instead they will be compared on a strictly technical level meaning that they will be of a similar triangle count and size. The idea is to make a more practical comparison between the two methods.

The group will start out with five different meshes on both sides, the uniquely textured meshes (group 3B) will each use a separate material instance, while group A will be using one instance. So the texture count on each side will be:

- Group A – 2x 512x256@24-bit texture maps
- Group B – 5x3 512x512@24-bit texture maps

Each group will start out with five meshes and increase in increments of five until a total amount of fifty is reached.

5.5 Group 4A & 4B
Group 4 will have a similar set up as group 3, but this time group A will be using several material instances. This is to better simulate a built environment and to further test out the impact of material instances on the video memory and draw calls in a way that better reflects actual conditions.

Each group will start out with five meshes and increase in increments of five until a total amount of fifty is reached.

5.6 Group 5A & 5B
Group 5 will be a test of some more elaborate conditions where each side will have ten fully modelled meshes, each using a separate material instance, each mesh in group A will have decals and reflections enabled and group B will have reflections enabled.

The meshes will be duplicated in increments of ten until one hundred is reached.

Depending on the outcome of these five tests, additional tests may be performed to further investigate some aspects of the results.

6 Results
The results of the tests will be presented in graphs together with an analysis. Each graph shows the data collected in memory allocation and draw calls.
6.1 Group 1

The purpose of this group was to get a basic comparison between a super-modular and a uniquely textured asset. 1A illustrates a 512x256 super-modular texture map applied to a simple primitive cube while 1B is a 512x512 unique texture applied to the same cube.

The super-modular cube is mapped across three UV-channels while the unique cube utilizes two. By themselves these graphs are nothing particularly noteworthy, but confirms that the super-modular structure does not affect the number of draw calls in any way, despite using a much heavier material that samples the texture map several times (in contrast to the unique material which only samples each texture once).

As far as memory allocation goes, the super-unique texture (1A) has a slightly lower base value due to its dimensions being half the height of the unique texture (1B).

![Group 1 Memory Allocation Graph](image)

The steady climb in memory allocation could be attributed to the increase in the dimensions of light maps as the UDK will pack these together into larger files (both assets use light maps of equal dimensions). The results of the first test group points towards the assets being more or less equal in the sense of both memory allocation and draw calls.

Any differences between the tested assets are small and could be attributed to a number of different sources, not to mention the fact that testing the same asset several times can yield a difference of up to a few hundred kilobytes.

The draw calls increase in a linear fashion, with an average of two draw calls per asset. The super-modular asset showed a slightly higher base with four calls to the unique asset which had a base of two.
These results show that on a basic level one super-modular asset is more or less equal to a uniquely textured asset.

6.2 Group 2

The purpose of group two was to make a comparison between super-modular assets using several material instances with different parameters in order to create variations and assets using several different textures to do the same. The material instances for the super-modular assets do not have reflections enabled, but three of them utilizes decals.
The results show the uniquely textured assets (2B) uses a little more than twice the texture memory as the super-modular assets (2A). As with group 1 the allocation slowly rises in relation to how many assets are present in the scene. The difference in material complexity is still very high with 47 instructions per instance for the uniquely mapped assets versus 132 for the super-modular assets with no visible impact to the results in either the memory allocation or the draw calls.

The draw calls average at two draw calls per asset with values comparable to the result of group 1, More materials doesn't seem to have any substantial effect on draw calls for either the super-modular or unique assets.
6.3 Group 3

Group 3 is the first group to use actual meshes instead of simple boxes to conduct tests, the purpose of this is to bring testing successively closer to the actual construction of a game ready asset being rendered. Each group uses meshes of a similar triangle count, but they differ in design (i.e. they are not a visual representation of the same type of object).

Group A uses a single material instance, whereas group B uses five different instances (these instances are very simple, and the only thing that changes is the diffuse and normal texture maps).

This was the first test where there was a great difference in memory allocation between the two groups and is where it became harder to properly collect data. The unique group uses a fairly heavy, but not as heavy as the super-modular, master material

**Group 3**

![Memory Allocation Graph](image)

Group B showed a much higher memory allocation (shown on the previous page), these results were so high that the test was redone after a complete reboot in order to see if the numbers were persistent.

Below is the result of the control test which show that the numbers from the previous test were in fact false. It is hard to say what caused this, but it may be related to a previous scene still being loaded into the memory (this was discovered during initial testing where creating new scene without restarting the application would cause the memory allocation to grow to very high numbers, this could be repeated until all of the available RAM was being allocated).

Above is the control test of group 3, both groups were retested with the same specifications. The interesting parts is how the super-modular assets, compared to the first testing of this group is no longer growing as rapidly. This is most likely because the light maps had not been configured
properly the first time the group was tested. While the UDK counts light map memory usage in a separate statistic, it is also added to the texture memory statistic. All of the assets have been configured to use a resolution of 32x32 pixels for the light maps as they aren't really relevant for what is being tested until a later state.

The steep initial climb in group 3B is odd, and can't be attributed to light maps. The memory allocation increases by roughly 1 MB per asset instance, before evening out before another increase by a couple of megabytes. The cause of this is due what is most like a bug which occurs within certain groups with no real pattern (it occurs across different materials, even the most simple ones super-modular or unique), making it seems completely random.

What happens is when moving the camera closer to the assets the texture memory will start filling up rapidly, simply moving the camera back and forth in close proximity to the assets will continue to fill up the memory allocation to a seemingly limitless degree (it only affects the texture memory, light map memory, pixel shader memory and other statistics are static while this occurs) . Upon further testing, assets which initially used around 13 megabytes of RAM could be pushed to use over 300 megabytes. Several suspected causes, such as level of detail maps and other possibilities were investigated and disabled in order to find the cause of the extreme memory allocation. None of these tests showed any improvement and the problem is still in need of further investigation.

This does of course put the validity of the results in question which will be discussed further under both 6 Analysis and 7 Discussion.
The draw calls are still linear and show little variation from previous groups. The interesting part about the draw calls is that they are seemingly unaffected by material complexity, object count (having several different objects in one mesh) as well as things like reflections etc. All of which could potentially force the engine to have to draw the same object several times in order to display the desired material effects.

As the graph above shows, the difference in draw calls is almost non-existent at this level of testing.

### 6.4 Group

This test group is very close to test group 3, the difference is that this time several material instances is used in the super-modular group. As the memory allocation graph shows, the super-modular group rises in memory allocation in somewhat equal increments for every five assets.
present in the scene. The values of group B are identical to those used in group 3 as this group has not changed in its set-up.

6.5 Group 5

The results of the draw calls are linear, averaging at two calls per asset. The collected data shows practically no difference between the different assets as far as draw calls are concerned. This points towards the average draw calls for these assets to be mesh and material, where the engine doesn't have to issue any draw calls for the textures more than once.
Group 5 attempts to push the assets to further resemble a built environment. The super-modular assets use their own material instances, each with both decals and reflections enabled, while the unique assets all use their own textures, with reflections and detail normal maps enabled. The differences in memory allocation is now very apparent, the super-modular assets still allocate a very small amount of memory with an almost constant rate, while the unique assets allocate more than ten times the memory when both scenes are populated by ten assets. The unique assets require more and more memory with every new instance in the scene.
The results change some when it comes to the draw calls, where, as the graph shows, the super-modular assets uses more draw calls for the same amount of assets. The explanation for this is simple as one of the super-modular assets uses two different materials, this asset was included in order to illustrate one of the things that will occur in an actual super-modular environment (this is discussed further under 7 Discussion). This comes from something that is purely related to the design of the particular asset, but is something that may very well be a part of any environment built with this kind of work flow.

7 Analysis

The data gathered in these tests supports the hypothesis of super-modular assets allocating a lot less texture memory than their uniquely textured counterparts. The data suggests that any of these benefits aren't really that apparent until a higher number of assets are placed in the scene. This does not rule out that the complexity of the material doesn't have a larger impact on other parts of the memory allocation, as the tests have been focused entirely on two specific statistics.

Despite the fact that the super-modular textures use a much heavier material which re-samples the texture maps several times, this isn't something that will have a clear impact on any of the data gathered through these tests.

Looking over the graphs, it's interesting how the number of draw calls are almost identical between the super-modular and unique groups, despite the fact that they differ some in both triangle count, object count (several objects in the same asset, i.e. the mesh is not one piece) as well as big differences in the material structure, where the super-modular material has to sample several different channels in several different arguments.

Some of the data presented in the results might however not be of total veracity, as was mentioned briefly in 5.3 Group 3 the Unreal Development Kit displayed some sort of bug where even a single asset could allocate a hundred time more texture memory than what could be expected, with a seemingly endless roof.

Whether this is simply a mere fault in the statistics and it is just displaying erroneous numbers or if the memory is actually being allocated is at this point pure speculation, but it still compromises the integrity of the whole study. All of the groups were tested more than once in an attempt to verify the data, which was for the most part successful, but there is still reason to be quite critical of the data collected in these tests.

This memory allocation was first attributed to level of detail (LOD) maps, i.e. rescaled versions of the textures, where moving in closer to the object would load a higher resolution version of the texture and moving the camera further away again would load a smaller resolution map, stacking the two in the RAM. After more extensive testing this turned out to be false, as the bug appeared even on assets that were forced to LOD 0 (all of the tested assets were in fact forced to LOD 0), and was absent in assets created with several LOD-layers.

Of course this thesis is not meant to be a exercise in speculative debugging, but it is well worth mentioning as this bug might have had a (in some cases) substantial impact on some of the collected data.
Further analysis of the results suggests that depending on the design of the assets, though completely up to the artist, the super-modular assets might have a larger impact on draw calls as the tested method of creation doesn't support operations such as more than one colour per material instance.

8 Discussion

There are a many points of discussion related to this thesis. First and foremost is the super-modular workflow itself. Looking over the results it's easy to get the impression that this method is a lot better than what it is being compared to uniquely textured assets. While it is true that it uses a lot less texture memory and a comparable amount of draw calls, these are only the purely technical side of it.

Working with super-modular textures is really limiting and the assets built with them can be rather lacking. While working on a small test environment using the super-modular workflow, I found it to be really fast and effective, all while making it easy to create assets with a homogeneous result. I was able to create quite a small array of different assets with a generally decent result, visually. What became lacking was the complete absence of any kind of material definition, where most of the assets would have pretty much the same finish. This is possibly the main drawback in my opinion, the lack of variation, which of course it kind of comes with the territory, but it severely limits what can actually be created. Still, it is impressive that the same texture can be used to create both the floor and walls of a room and things to populate it.

Another drawback is possibly that the super-modular workflow will require a lot more triangles in order to (arguably) look its best. One of the uses of the normal map in the regular, unique workflow is to give the illusion of smooth edges, without actually adding that geometry. This is possible, but quite difficult to achieve with the super-modular workflow, in my opinion. Apart from having to add geometry in order to get smooth edges, with the super-modular workflow it is also necessary to add additional geometry in order to be able to properly map in details from the texture. This can lead to a higher triangle count in super-modular assets than what would be necessary had the assets been created with unique textures.

To further expand on the topic of limitations in the super-modular workflow design-wise, any kind of outdoor environment is out of the question, as trying to make any kind of representation of an organic material is impossible, unless the entire environment is made of that single material (e.g. an entire landscape made of wood). Working with the super-modular workflow also takes a lot of planning, as the set of details available will be limited it becomes important to make sure that whatever is placed on the texture map can be reused over and over again while not looking awkward or out of place, i.e. it has to make sense for the design.

Just testing the viable applications of the workflow would be a really lengthy task, and would possibly yield even more concrete results of the advantages and disadvantages of the workflow, this thesis is merely a small stepping stone that tries to gather a very limited, but important set of data.
which can be used to compare the super-modular workflow to other counterparts. While this
discussion may bring up a number of the advantages and disadvantages, these are in no way a
proven fact or something that should be taken as such. This discussion exists solely because in order
to share some experience which could help further explain this thesis and why it exists, but also
point to some aspects which I feel should be investigated further, as the potential benefits of doing
so could be of great use to the field, which is at the time of writing very small.

In further explanation of the advantages and disadvantages of this workflow, it can be said that the
super-modular workflow work really well if it is utilized with a design that suits it. Many of the
examples you can find of environments built with this type of workflow will be quite impressive to
look at. It works particularly well with minimalistic designs (due to the nature of the workflow), and
while most of the designs I have seen have had basically the same kind of science-fiction setting. As
an example there is two of Tor Frick's environments (both shown below) which are both great
examples of where this particular workflow works extremely well.

(Images by Tor Frick, www.torfrick.com)

With the state of the current hardware there is no actual need to limit the amount of texture maps
this heavily, but loading a large amount of textures per frame can still make it very difficult to keep
a constant frame rate (Möller, Haines, Hoffman 2008:173). Which is what makes this particular
workflow very powerful, as the hardware will only have to load two textures.

To re-iterate on the purpose of this thesis and its creation, it does an intentional choice in cutting its
scope short. The aspects that could be tested and the method of doing so could be expanded greatly
and possibly yield bigger and more interesting results, but due to the lack of earlier research doing
so in a single thesis would be really hard to support properly and the risk of errors in the collection
of data would greatly increase.

As for the purpose it ends up merely answering the questions it posed with a quite positive result.
While the difference in allocated texture memory can easily be calculated by simply looking at the
file size of the used textures and adding them together, these tests show that this carries over into a
practical application, i.e. it remains true in an actual game engine instead of in pure numbers.

While setting up these tests, the actual optimization of the super-modular material was somewhat
doubtful due to how complex it is, having to call a wide array of different instructions in order to
display the desired result. And while these results neither proves or disapproves the actual impact of
the material due to the limited scope of the collected data, it points to a reason to believe that the material complexity has no impact on either the texture memory allocation or the draw calls. The latter being something that was something that couldn't be properly predicted even in a theoretical manner until actually tested.

Since the tests performed in this thesis were all conducted in the same engine (UDK), it is hard to say if the results would carry over to another engine. This is mainly due to the fact that all engines work differently and the actual code for rendering assets can vary widely which doesn't invalidate the collected data, but rather narrows the impact of it. And in the context of the problems encountered with the engine, what with the memory allocation bug described at earlier chapters it's hard to say if running the exact same tests again would yield an identical result. So with that it stands to say that none of the numbers or graphs in this thesis should be treated as approximates rather than definitive.

One thing that this thesis achieved is that it made me, as and artist a lot more aware of the impact every asset can have on the performance of a game, which is something that I've come to argue is a really important attribute for any artist. It's basically a whole new perspective of the craft as a whole, as my own focus has always been to create the most insanely detailed environment possible not giving any second thought to throwing in an additional texture map. Just seeing how far it's possible to push a single texture map in terms of what can be created with it really is a huge change in perspective, and the theoretical appliances of this workflow when jointed with the more traditional approach opens up huge possibilities in creating not only good looking environments and assets, but also performance friendly.

It should also be noted that this thesis is written by an artist for artists, that is to say the actual technicalities of the hardware operation is, for a lack of better words a bit simplified in order to make it more accessible to the target audience, if you wish.

On the note of further research and where to go from here, there is in my opinion still a lot to be done, and many more aspects that should be researched in more detail before reaching any definite conclusions. The subject of modularity in game engines isn't anything per se, but the documented research is very small, to the degree that finding anything to base most parts of this thesis on required a huge effort. It's important to remember though that this is a field that is rapidly expanding and keeping the documentation and research up to date becomes difficult. In the time it takes to properly research a certain area, the field might have moved on to something new. To the degree where most of the analysis and progress in regards to how artists create isn't really made in any kind of formal context, but instead most of the progress within the field of game art creation is made across various platforms on the internet, by both professionals and laymen alike.

The collective knowledge of game art is usually found in public wiki-sites, rather than in reports and papers. These sites are usually used to pool together a number of different resources, spanning from tutorials and techniques to more theoretical and technical documents, all of which can be edited by anyone, thus building consensus by the experience and tests of the majority.
9 Conclusion

The conclusion that can be draw from this thesis is that in the Unreal Development Kit, during the conditions tested, the super-modular material allocate less texture memory, and averages the same amount of draw calls on the GPU (unless it's utilizing more material instances due to design decisions) despite the much heavier material.

The difference in memory allocation is not apparent until the difference in materials used, i.e. as the amount of different assets needing different texture maps gets higher. On a basis of one versus one, there is only a negligible difference between the two, which is due to the fact that in these tests both assets used an equal amount of maps, with only a slight difference in resolution (512x512 and 512x256).

These results answer all of the questions asked in Chapter 3, in the context of the Unreal Development Kit. Further research is required in order to see if these results carry over to engines that have been written in a different way, and at this point the results of such a test is pure speculation.

Further tests are also required in order to test other aspects of this workflow and to establish some sort of formalization of the whole theory.


10 References

Literature


http://chrisholden.net/tutor/cover_tutor01.htm

http://www.miessociety.org/legacy/

http://www.thiagoklafke.com/modularenvironments.html

http://www.philipk.net/tutorials/modular_rocks/modular_rocks.html

http://wiki.polycount.com/NormalMap

Sources
http://usa.autodesk.com/

www.adobe.com

http://www.udk.com

http://www.torfrick.com/info/datacore.html

*Scifi Lab*
http://www.torfrick.com/info/lab.html