TAMPERE UNIVERSITY
OF APPLIED SCIENCES

MODULARITY IN CREATING 3D GAME ENVIRONMENTS

Mike Bernstein

Bachelor's Thesis
May 2017
Degree Programme in Media and Arts
Interactive Media

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Media and Arts
Interactive Media

BERNSTEIN, MIKE:
Modularity in Creating 3D Game Environments

Bachelor's thesis 77 pages
April 2018

In the game industry, environment artists hold a specialized role in that they must understand how to best combine visual fidelity with performance and high-volume productivity. Modularity is a powerful tool in environment art, unlocking the potential of artists who can create more assets in less time. In addition, teams benefit from the flexibility and rapid iteration afforded by working with modular assets. Though exact techniques and best practices vary by the team or artist, it is possible to select a general modular workflow and compare it against a traditional, non-modular workflow.

After presenting research into how modularity is used in modern games, an example 3D modular game environment was created to examine modular workflow and how it applied to small projects. Comparison was made between the potential yield of time spent using a modular workflow versus a non-modular one.

Working on an environment with modular techniques proved beneficial in terms of speed, flexibility, and sense of progress. The results of many tasks were immediately apparent and easily iterated upon. Visual quality was higher than expected even when using unfamiliar techniques. The benefits of adopting a modular workflow for such projects were apparent throughout the process.

Key words: modular, 3D environment, game art, techniques, workflow.

CONTENTS

GLOSSARY

3D	Three dimensional in virtual, computer generated space.
Alpha	Measures the opacity of a pixel in an image,
Asset	A finished unit of 3D art that can be used in a game engine
Baking	Using software to create a texture image
Bottleneck	A situation in which one part of a process limits that process
Blockout	An assembly of simplified placeholder assets as a preview
Color Correction	An overall adjustment to the color range of a scene, applied to each frame when it is rendered
Draw Call	Issued by the game object to graphics hardware each time an object must be drawn on the player’s screen
Environment	A general term for a gameplay location, e.g. the place that a set of assets is intended to portray
Environment Artist	An artist specialized in creating environment assets
Game Engine	Software which facilitates the creation of a game
Gameplay	The experience of playing a game and how players feel in the process
Geometry	The form of a model created by its polygons; its 3D shape
Grid	Virtual space divided by arbitrary units, used as a guide for placement of assets
Immersion	Emotional investment and suspension of disbelief
Invisible Work	Additional, unexpected time expenditure in a work endeavour
Iteration	Refining something by creating and testings versions of it
Level Designer	A person charged specifically with assembling environment assets for the best possible gameplay experience
Material	The result of combining textures of one or more types in a game engine, applicable to assets
Post-Processing	Image-wide effects applied after each frame is rendered
Primitive	A simple, basic 3D form, such as a cube, cylinder, or sphere
Shader	Calculates how a material will be rendered on the screen
Snapping	A software feature in which virtual objects are easily placed, with precision, on a grid or other arbitrary point in space

Texture	A general term for an image that will be projected onto a model to convey surface detail
Texture Atlas	Many textures assembled into a single image
Texture Memory	Computer memory specifically devoted to storing textures during gameplay, usually integrated into graphics hardware
Trim Sheet	A texture image intended to be used piecemeal on many assets
UV	A two-dimensional coordinate defining the boundary of a polygon for the purposes of projecting a texture
UV Shells	One or more polygons treated as a single piece in UV mapping
Workflow	A series of tasks by which a complicated process is broken down into steps

1 INTRODUCTION

The concept of modularity isn’t a new idea in game development, and modular design is present in many industries and human endeavours. Whenever a complicated process can benefit from having many individual elements arranged into larger, more manageable chunks, modularity is a natural conclusion.

Building a 3D game environment is one such process, especially so in the modern day, when advancements in technique and technology allow us to create virtual environments in realistic detail. Artists continue to achieve higher standards, and the expectations of consumers rise to match their accomplishments. In a time when gameplay visuals begin to closely resemble cinematic quality, environment artists are responsible for creating a vast amount of high quality assets. In some genres, teams must achieve a high level of detail over virtual areas covering many square kilometres.

Modular environments were used in such high-profile titles as the Mass Effect, Halo, The Elder Scrolls, Fallout, Assassin’s Creed, and many more. This is an investigation into the modular workflow, how it is used into the game industry today, and its benefits and limitations.

Saving time and increasing productivity is a natural part of running a business, and so modular design has always had a presence in game environment creation. Modern technology has enabled artists to refine their techniques, giving rise to new forms of modular asset creation that overcome the drawbacks of both traditional, non-modular workflows and earlier modular systems.

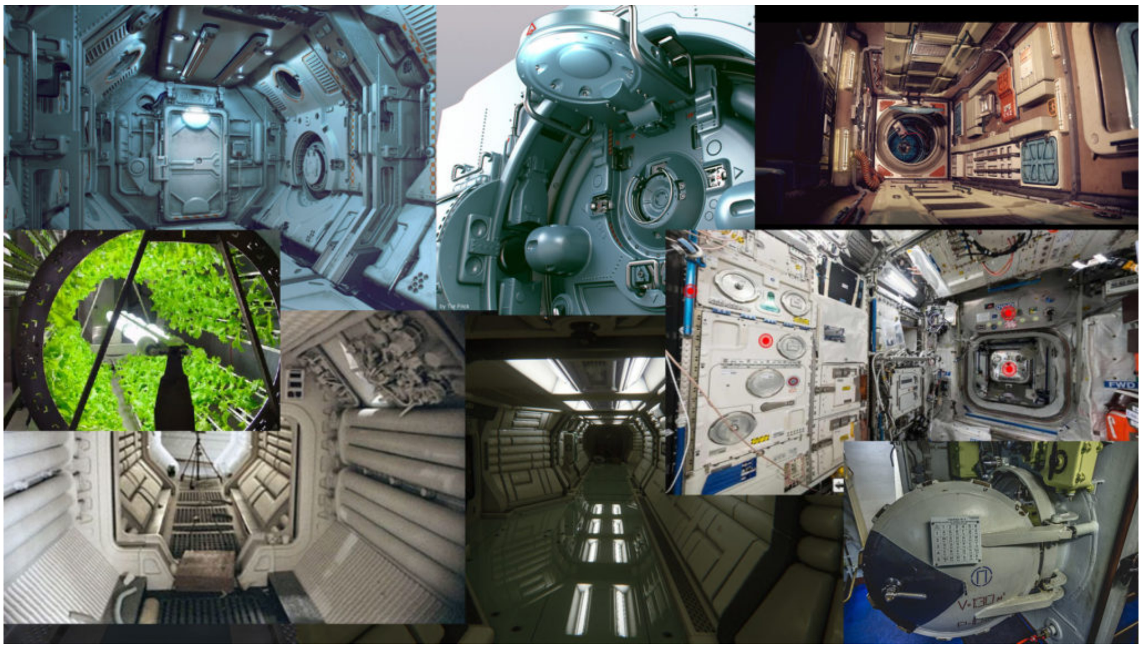
2 MODULARITY OVERVIEW

2.1 Traditional Non-Modular Workflow

To understand modular techniques used by environment artists in the game industry, one must first understand how 3D game assets are traditionally made. For the purposes of providing vital background information, “traditional” is defined as a basic process that is commonly used by professionals and taught to students, which has remained relatively unchanged since 3D games became common. Where many optional component processes exist, those most directly relevant to the rest of this thesis have been chosen as background.

Though the process may seem arcane to the uninitiated, there are certain steps that artists must accomplish to produce a finished asset. This is a high-level overview of the process, but in a game studio of any size, deviation is common in how the steps are achieved. The same steps must be reached from a technical standpoint, but the desired aesthetic style, artists’ working preferences, and other factors may affect how they are accomplished.

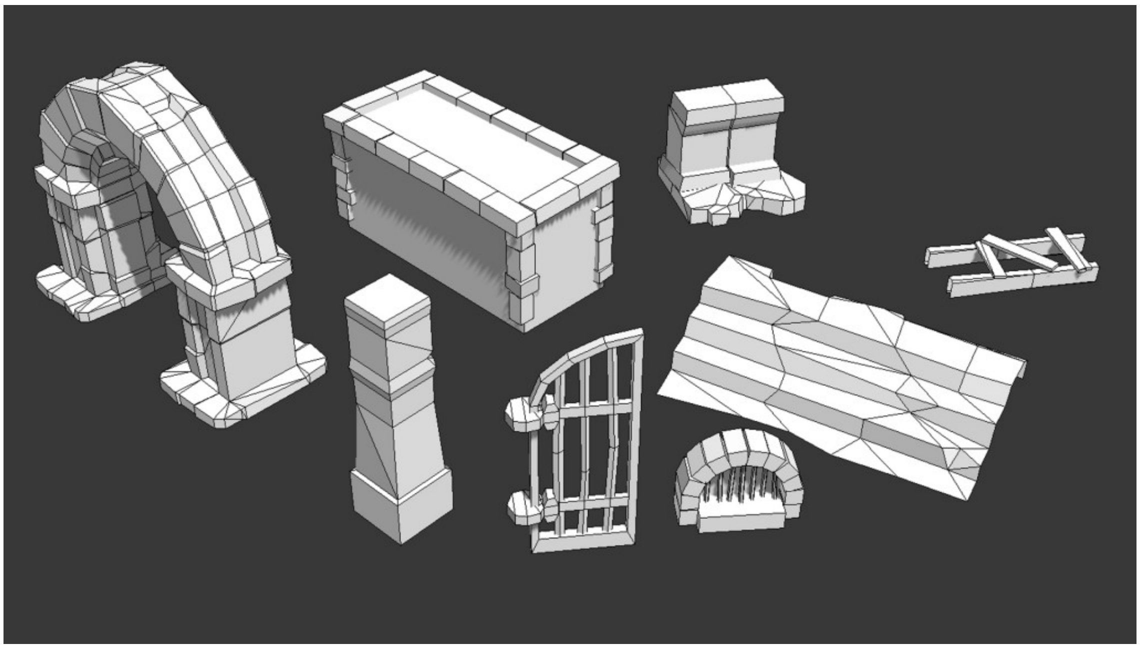
Creating 3D assets is a time-consuming undertaking, just as in more traditional art forms like painting, and the same initial steps are usually taken to prevent investing time acting on a flawed idea. Typically, the artist first gathers reference images from books, online image galleries, and other sources and creates a mood board (picture 1) (Pettit 2015). Examining the assembled image library clarifies what the artist’s vision is and what it is not. An artist may like the aesthetics of a few objects in one image, but take issue with some other aspect, and this kind of comparison allows the artist to move from general to specific ideas about what they intend to create.



PICTURE 1. An arrangement of visual reference material for designing a space capsule environment (Chollet 2017).

Before or after reference gathering, it is economical to sketch ideas on paper or digitally. Communicating ideas, or submitting them to a superior for approval, takes significantly less time with a series of preliminary conceptual drawings. In the case of many studios, a person with the title of Art Lead or Art Director will go through much the same process as the artist initially did with their reference images, providing feedback based on comparison between them. After some iterations, the environment artist is cleared to begin creating models. For each object in the intended scene, the artist must create a model and textures, then assign the textures to the model in a way that the 3D program or game engine understands.

Creating a model is an involved process that requires a developed understanding of all software involved, specific requirements of the final product, and overall principles of computer graphics. In its simplest form, the artist places polygons in virtual space until the desired shape is formed. Manipulating the size, rotation, and position of these polygons or their constituent parts allows the artist to modify simple, primitive shapes, like a cube or cylinder, into more complicated forms (picture 2).



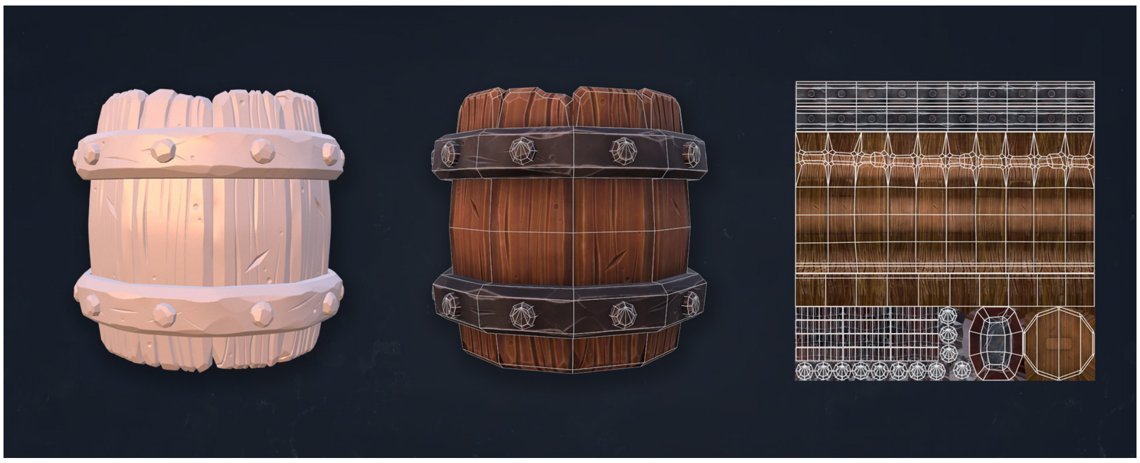
PICTURE 2. Regardless of the complexity of their final form, these 3D models are made up of only triangular and quadrilateral polygons (Damjanov 2016).

The relationship between those shapes to one another must adhere to certain rules, and it is the 3D artist’s job to arrive at a satisfactory result with speed and efficiency, without breaking those rules. Disregarding the rules can lead to a model that doesn’t render properly in a game engine, performs badly, or which cannot be loaded at all. (Silverman 2013.) Further, the artist must constantly achieve the correct balance between the visual quality of a model and how demanding it is to render in real-time. In most cases, using more polygons to describe a shape leads to better visual quality, but increases the work-load of the rendering hardware.

Modelling well is a skill developed over years of practice. It must be remembered that in the game industry, the 3D artist is part of a production pipeline. The success or failure of a project, or entire company, hinges on producing a valuable product within a certain timeframe. Consequently, a good modeller is often judged on a balance of aesthetic quality of the final product, the speed with which it was produced, and the efficiency with which current technology can render it on the end-user’s device (Silverman 2013). A finished model in this traditional workflow is truly purpose-built to fulfil these criteria.

A finished model consists of any number of surfaces which must be mapped to a texture in a process called UV mapping. In this process, the artist will flatten a 3D object into 2D space, so a texture can be applied (Pettit 2015). By manipulating the position of UVs (two-dimensional analogues of the vertices at the corners of each face), the artist assigns part of an image that will be projected onto each face of the model in 3D space.

In the traditional 3D workflow, an artist first projects the model's UVs onto a blank image, resulting in a flattened representation of the models surfaces. It is easy to understand this process by imagining a complicated paper origami figure that has been unfolded, making it flat and understandable, with creases left behind to show where each piece was located when it was folded (picture 3). Once a model is UV mapped, the artist can begin creating texture images to fit the surfaces of the model with any 2D graphics software.


PICTURE 3. A model (left), the model's UV edges (middle), and the model unfolded into UVs on a texture image (right) (Sus 2018).

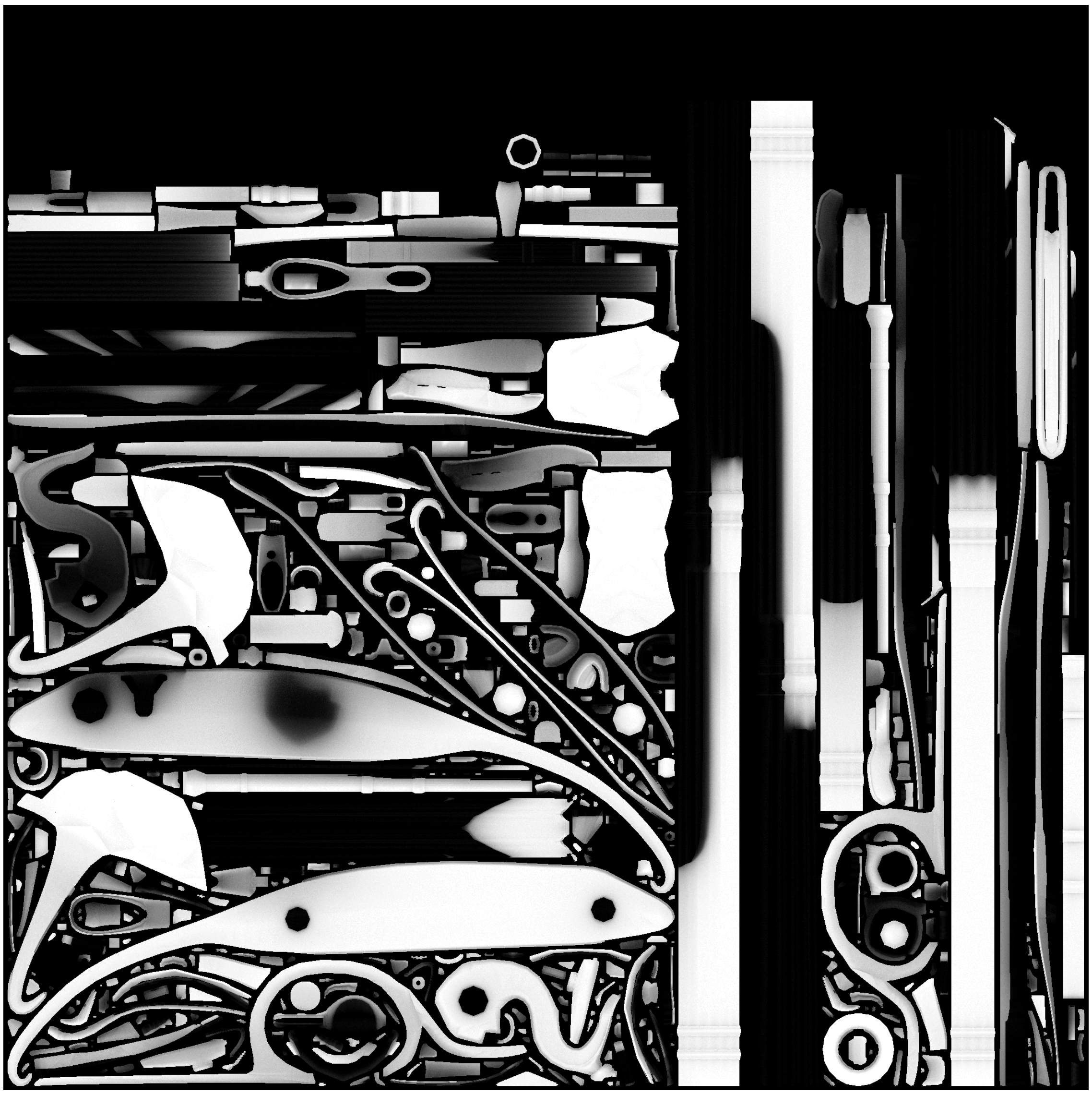
Simplistically, texturing could be equated to painting an unfinished product in the real world, but there is a fundamental difference between the physical world and those we can technologically simulate. In the real world, the workings of light, color, reflection, atmosphere, surface imperfections, and other factors are always perceptually correct. We expect things to look a certain way, and spare little thought to the complexity of what we are seeing, most of the time. The environment artist must create something that approximates our expectations as closely as possible, and potentially influences our perceptions to enhance a game experience. They must do all this in a simulated world that is manufactured to be calculable by home computers, consoles, or mobile phones.

Consequently, texturing an asset for a modern game usually involves several different images used in concert to represent different visual attributes (Silverman 2013). Graphics hardware then interprets and intermixes the effects of these various images, or maps, to simulate realistic conditions. Continuing this overview of the traditional process, we will focus on four of the most basic, representative texture types: diffuse, ambient occlusion, specular, and normal maps. While there are newer, more advanced, and more specialized texture types, these four represent the basic functions that textures achieve in the typical workflow.

Diffuse textures (picture 4) consist of color information and are the most closely analogous to painting an object in the real world. In the case of games designed to demand very little hardware performance, or those with certain stylized aesthetics, a diffuse texture may be the only texture type that is used.

PICTURE 4. A diffuse texture map containing color information.

Ambient occlusion maps (picture 5) are pre-calculated approximations of where parts of the model occlude, or block, light from other areas. This effect enhances the eye's ability to distinguish points of contact between objects, such as inside a corner or crack (Gurney 2010, 54). In many cases, ambient occlusion maps are added to diffuse textures, as they reduce the need for a game engine to calculate these shadows in real-time. Doing so ensures that the diffuse textures are darkened in shadowed areas of the model with computer-aided precision, sometimes exceeding what an artist can achieve by hand in terms of accurately representing 3D forms.

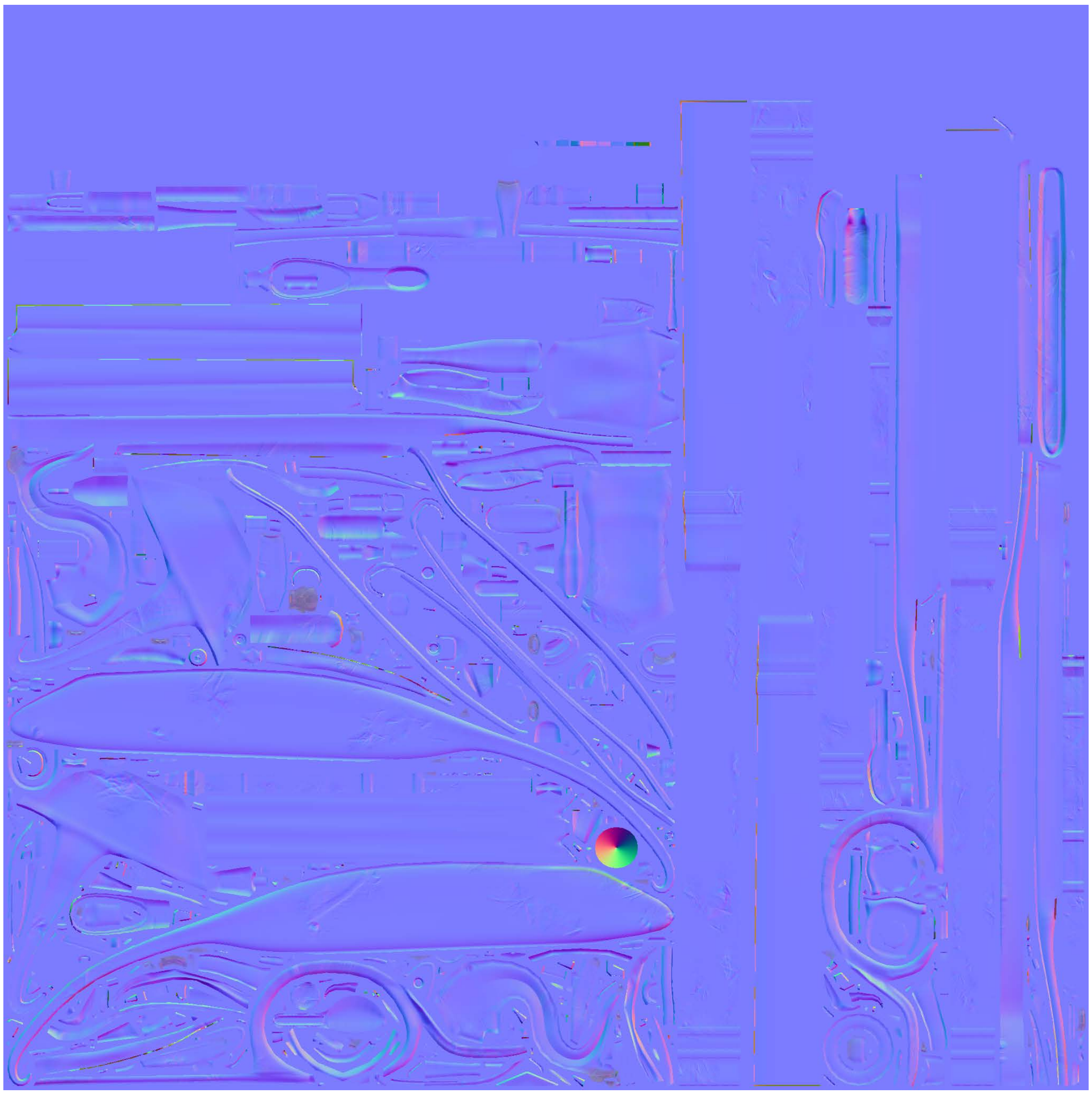


PICTURE 5. An ambient occlusion map. Lighter areas are fully lit, darker areas are in shadow.

Specular maps (picture 6) indicate which parts of the texture reflect light, and to what degree. The intensity and consistency of light's reflection against a surface is extremely important to our brain's perception of that surface, whether it is hard or soft, rough or smooth. Specular maps help us perceive which elements of a surface are smooth, raised, round, wet, or otherwise acted upon by light in a way the human eye can interpret.

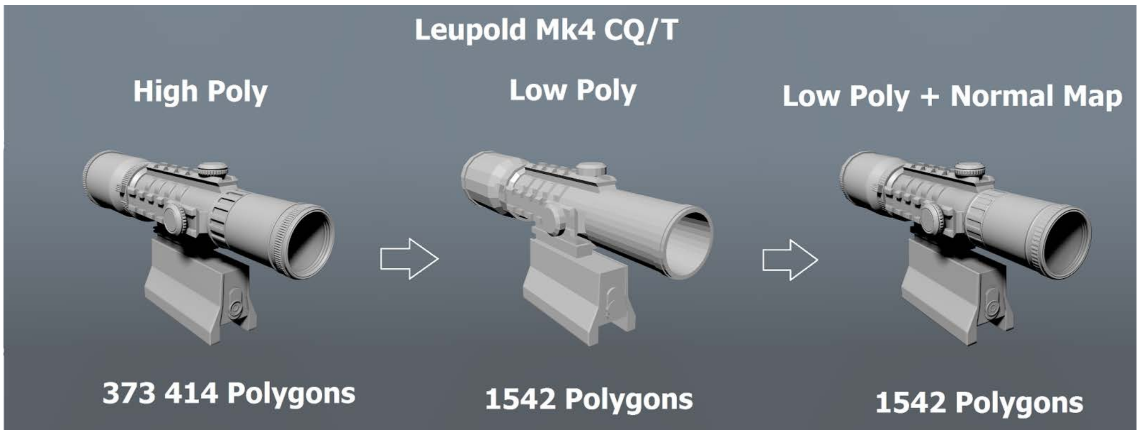
PICTURE 6. Specular map. Lighter areas reflect light more sharply, indicating harder and smoother surfaces. This object has no extremely smooth areas, which could be white.

Normal maps (picture 7), create the illusion of three dimensionality by influencing the way directional light is reflected from each part of the surface. While the use of normal maps has become common, their relatively low impact on game performance has made massive improvements in visual fidelity possible without adding polygons to the game environment. Normal maps make use of the separate color channels used in computer image formats and working with them by hand is difficult. Typically, artists use software to “bake” normal maps, a process in which the software calculates the color value of each pixel in the texture based on its angle, in reverse of how a game engine will later read the texture to simulate light striking those angles. (Hajioannou 2013.)



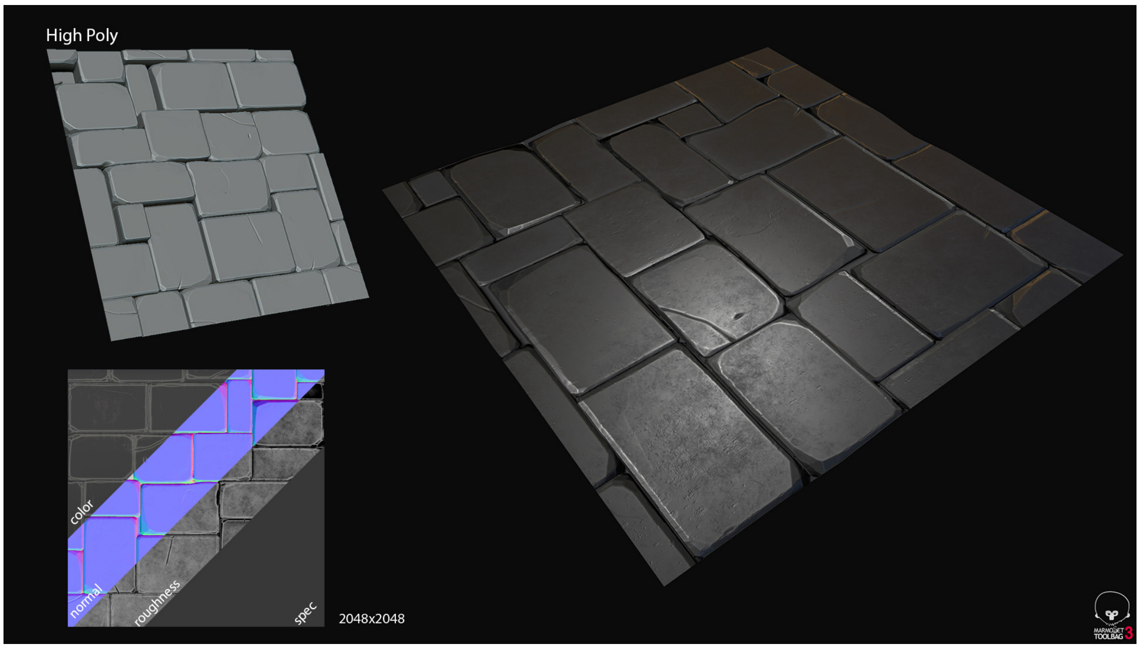
Picture 7. A normal map. The edges of shapes are more red, green, or blue depending on their curvature, indicating the direction at which light reflects from them.

Modern workflows often involve the creation of a more detailed version of each asset with a higher polygon count, which can be sculpted by the artist in extreme detail. Specialized software can then bake a normal map by comparing the high-polygon and low-polygon versions of the model. When done correctly, the final asset can look indistinguishable from the high-polygon version (picture 8), with some limitations. The artist must consider carefully how a model’s normal maps will match up to its silhouette, which cannot be altered short of changing the model itself. Furthermore, once normal maps are baked, any change to the underlying model or UV mapping will result in visual anomalies.



PICTURE 8. An original high-polygon model, low-polygon model for use in a game engine, and the appearance of the latter with a normal map. (Clarry 2015)

Regardless of which texture types are included, texture images can accomplish a variety of visual effects with modern technology. It is typical to use various types in concert with one another to achieve effects that mimic physical materials and variation, as well as the perception that the model is more detailed than it really is. The set of textures used on a given model is layered, one on top of one another, in a game engine. Collectively this is referred to as a material (picture 9).



PICTURE 9. This floor tile (right) is a simple plane with a material applied. The material consists of several textures (lower left), including normal maps baked from a high-polygon model (top left) (Godbille 2018).

At this point, when the model has been created, mapped, and textured to whatever degree the project demands, it can be integrated into a game engine. Frequently, an initial test integration is made for critical assessment and feedback. In some cases, revisions might be requested. Recall the steps of this entire process (figure 1).

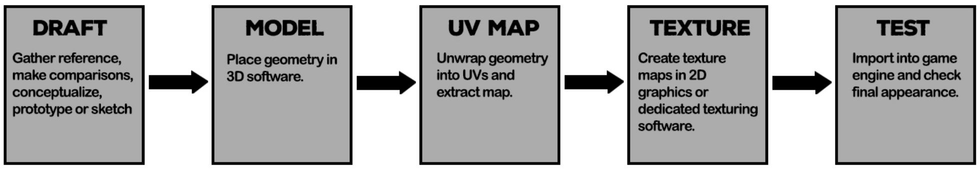

DRAFT
Gather reference, make comparisons, conceptualize, prototype or sketch
MODEL
Place geometry in 3D software.
UV MAP
Unwrap geometry into UVs and extract map.
TEXTURE
Create texture maps in 2D graphics or dedicated texturing software.
TEST
Import into game engine and check final appearance.

FIGURE 1. Five basic steps that make up the traditional non-modular process.

By following these steps, a 3D artist can arrive at a model that is technically sound, aesthetically pleasing, and which has as small an impact as possible on game performance as possible (picture 10). Typically, this series of steps leads to the highest quality end product, with each separate step serving as a quality control point, upon which the next step can begin. This is the benefit of the traditional workflow: the process is straightforward, and the resulting assets are well optimized for their intended purpose.

PICTURE 10. The texture maps in pictures 4 through 7 are combined to texture this finished model.

2.1.1 Challenges of the Non-Modular Workflow

On the other hand, assets created in this manner are comparatively inflexible, increasingly so as they near completion. Consider a case in which a revision of the model is requested. Changing the model invalidates some or all of the UV mapping, which in turn invalidates the texture placement. Baked textures, such as normal maps and occlusion maps, must be re-baked and possibly re-combined with other parts of the intended texture. Entire steps must be repeated by backtracking through an already lengthy process.

Similarly, most of the significant amount of work going into each asset is unique to that asset. Models are tailored to fit together as efficiently as possible, and textures are created with the specific features of that model in mind. Normal maps and occlusion, once baked, are only applicable to the model for which they were made, as it was at the time of the baking. If the artist has created a high-polygon version of the asset to accommodate the baking of more detailed textures, as with normal maps, it may also require significant work to ensure that the alterations to both the high and low polygon version match up.

From the first 3D models in gaming, the workflow of taking an idea to a finished model has remained largely the same and has presented the same difficulties. Over time, technology and innovation have allowed us to enhance process in ways that maximize visual fidelity for a modern audience with modern hardware, but also to create workflows that are more flexible and efficient.

2.2 The Emergence of Modular Design in Games

There is a saying among game artists which states that anything an artist can achieve today will become achievable with much less and effort time within a few years. In every studio, no matter how small, artists are looking for ways to improve their craft. Sometimes it comes in the form of new capabilities offered by advancements in technology. Other times, ideas are shared, even by industry giants, in the form of seminars or presentations at international conferences. There is also a wealth of knowledge available across various internet forums, blogs, and video sharing sites, for those who look for it.

Thusly, the current generation of environment artists are well informed and well prepared to create impressive looking scenes. Meanwhile, the expectations of players increase to meet the new capabilities of art and technology, fuelling an arms race of innovation and consumption. Falling behind the visual expectations of the current generation of consumerss is a dangerous prospect for game developers, who have responded in various ways. There is a high demand for games of many visual styles, from the stylized to the abstract, and many developers find their niche creating them. This thesis is primarily focused on the perspective of game studios who want to achieve a high level of realism across gameplay environments spanning several virtual kilometres or more (picture 11).

PICTURE 11. The gameplay map of The Elder Scrolls IV: Oblivion encompasses an area equivalent to 16 square miles, including cities, landmarks, and wilderness areas (Bethesda Softworks 2006).

Along with innovations in the visual fidelity we can achieve, there is a constant evolution of the techniques studios use to create assets in less time. It is easier now than it has ever been for entry-level environment artists to create assets of high visual quality. To do so across a game environment comparable to those of some modern games requires a considerable investment on the part of the studio. They need more artists, more software licences, more management, and there is more potential for miscommunication and missed deadlines. Creative and clever use of tools increase each artists' productivity, streamlining a studio's creative process and minimizing expenses and delays.

There are several specific factors that have predicated the invention and adoption of modular environment design techniques. Recall the traditional non-modular workflow, a process of steps that allow an artist to purpose-build assets for their intended placement within a scene. As technology, consumer standards, and studio dynamics evolve, so do the parameters by which they choose their environment asset workflow. Increasingly, studios are choosing modular techniques for projects with an ambitious amount of content.

2.2.1 Advancements in Technology

For many years, the visual quality of 3D games was primarily limited by the capability of end-user devices to render them. This remains an important factor in modern games, but advancements in 3D rendering have made it possible to increase visual quality in ways that do not involve adding polygons to the screen. At this moment in time, hardware in the average Steam user's home computer is capable of rendering nearly cinematic quality at framerates that exceed the human eye's ability to perceive individual frames (Vazquez 2018). Other platforms, like home gaming consoles, aren't far behind.

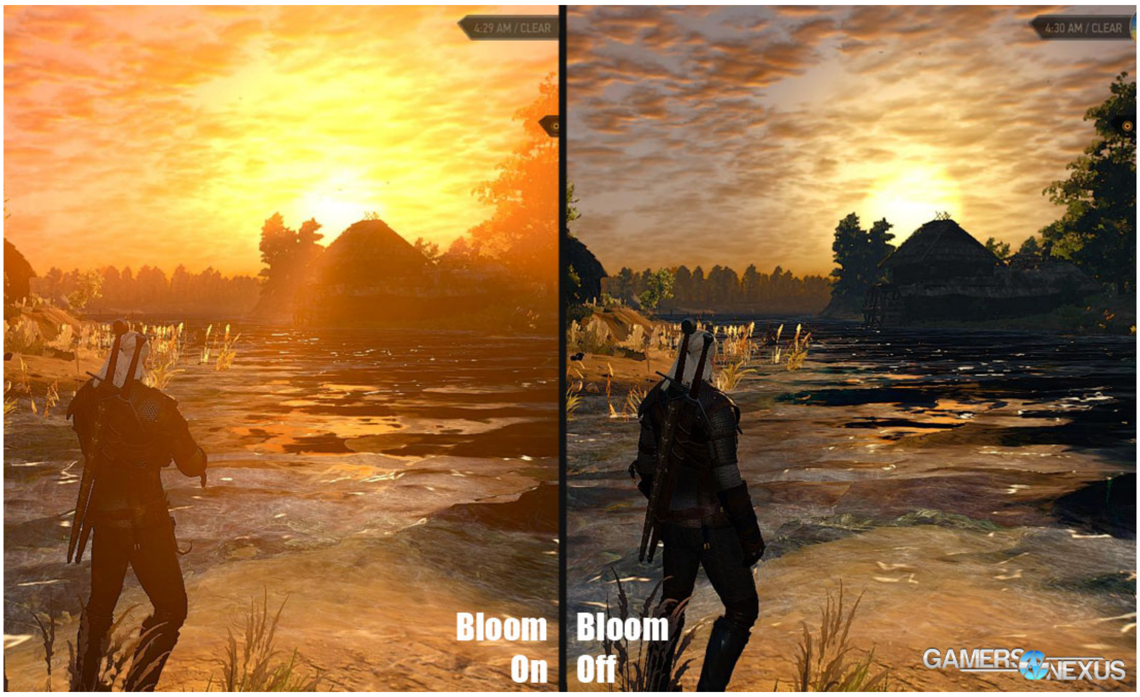
The speed of graphics hardware has increased so much that polygon count is no longer the primary limiting factor, and in fact modern consumers' expectations demand high quality geometry more than ever (Perry 2002, 30). It is still very possible to overwhelm hardware with models of extremely high polygon count, but skilled 3D artists are able to achieve cutting edge visual quality across an entire game environment without exceeding the capability of most end-user hardware.

While it was once necessary to count the polygons used and strictly ration them, it is now possible to show flexibility and indulge the player in more visual detail, selectively. A few hundred extra polygons were an enormous issue when artists were limited to a few thousand for an entire scene, but modern limitations are on the order of millions.

The introduction of bump maps, height maps, parallax maps and normal maps (all of which are techniques for using textures to simulate additional polygon density or surface detail) significantly expanded 3D artists' capabilities. Without physically creating, managing, and rendering additional polygons, computing power can be used to create the illusion of a more detailed model using only an image file (Hajioannou 2013).

Other advancements have come in the form of post-processing, which is a broad term encompassing a wide variety of effects applied to each frame after it is rendered. Some post-processing effects attempt to simulate the workings of light as it travels through the human eye or atmosphere, while others sharpen or smooth areas of each frame (picture 12). In general, post-processing effects are applied across the entire screen image. They affect the “big picture”, by which users perceive the environment as a whole.

Post processing, used judiciously, can greatly improve the visual quality of all assets on the screen, requiring no additional work from the 3D artist. A common example is the addition of atmospheric fog, which mimics particles in the air and adds a sense of scale to any wide shot of an environment (Gurney 2010, 176-177). Some artists consider post-processing such as color correction to be an essential final step in creating an environment; part of the process, not a separate step (Damjanov 2016).



PICTURE 12. Screenshots of The Witcher 3 gameplay with and without bloom, a post-processing technique that enhances light (Burke 2015).

In combination, these advances allow artists to do more with less. Models still need to be efficient in terms of polygon count, but there is less necessity to strictly ration the number used in each asset. It is possible to create scenes which meet current visual standards without pushing the limits of most users’ hardware.

As well, many game engines accommodate the use of multiple textures in a single material. In broad terms, a material in a game engine is the sum of a set of texture maps and shader configurations that determine how objects of said material will be rendered on the screen. Put simply, this enables artists to make use of combined textures that tile seamlessly across certain parts of a model while others are UV mapped to specific features. Previously, the same result could only be achieved by creating each asset out of several individual models, which was less efficient to manage and less performance-friendly in high volumes. This capability is a key factor in the adoption of certain techniques that are becoming standards in the game industry.

Without breakthroughs like these, modular environment assets would be less viable in projects where visual quality is a focal point. With them, artists can potentially achieve the same visual quality as they would with the traditional workflow, with all the productivity and flexibility benefits modularity implies.

2.2.2 Demands in a Studio Environment

In a game development studio, there are typically several people involved in the creation of environment assets if not an entire hierarchy of roles and specialties. In order to remain productive, development teams need to maximize their ability to communicate and standardize their working practices. Modularity is a natural extension of this need. While individual artists might naturally work in different ways, instituting modular standards to work within maximizes their ability to work together by eliminating variables (Perry 2002, 35). The assets created by different team members should require little or no additional work to use together in a scene.

Environment artists need to understand how their assets will be assembled, but in a studio environment, various other team members will view or use those assets in some way. Though the hierarchical structure of studios varies widely, assets usually move through a pipeline of specialists before being finalized, or they may be assembled by someone charged with designing environments according to gameplay, not aesthetics. In some studios, the same assets might repeatedly move back and forth between artists and level designers in pursuit of best mix of aesthetic and mechanical usefulness.

This disconnection is a frequent source of difficulty and invisible work in studios, as gameplay considerations and user testing results must be weighed against aesthetic concerns. Standardized assets allow team members outside of the art team to more easily make use of the assets, and more freely make changes to level design as they meet demands. With open lines of communication, artist’s can ensure that environments look as good as possible without sacrificing gameplay. (Burges and Purkeypile 2016.)

User feedback in the form of monitored user testing and alpha/beta tests is an important part of the development process of many studios (Tan 2012). The saturation of our global entertainment market means that consumers have more choices than ever for how they spend their money, and most studios find it more economical to run testing proceedings than risk consumer backlash against an untested product. This growing need for feedback alone is changing the game industry, as feedback is only actionable if changes can be made and a new iteration tested in a timely manner. Studios therefore increasingly favour workflows that maximize the possibility of rapid iteration, implementing modularity across various aspects of their project.

2.3 Expected Benefits of Using a Modular Workflow

In order to meet modern standards for game visuals and gameplay, studio art teams take advantage of recent technological advances and workflows that maximize their ability to adapt. Barring flawed execution, modular techniques offer benefits that allow studios to eliminate some amount of communication variables, wasted work, and inflexibility. An effective modular system requires significant planning (Klafke 2014).

Modularity can be viewed as an investment, which has been proven successful and profitable in the creation of many high-profile games such as The Elder Scrolls V: Skyrim (Burges and Purkeypile 2016). It costs studios time in which to plan and standardize early in the design process, and maintenance of the workflow that results. While the benefits and limitations of working modular vary with the project, modularity usually enables studios to prototype and iterate faster, make changes more easily, and increase productivity.

2.3.1 Flexibility

The term “modular” is defined by the Merriam-Webster dictionary as something which is constructed with standardized units or dimensions for flexibility and variety in use.

Modular environments in games usually fit this definition exactly; each piece of a scene is designed for quick assembly in many possible configurations, seamlessly, by adhering to certain standards for its size and shape. Once an artist or development team has established these standards, it is possible to rapidly add, remove, or change assembled scenes by moving these fitted pieces.

This form of flexibility allows artists to create fewer individual pieces yet arrange them quickly into new configurations. Consider the traditional non-modular workflow discussed earlier. When alterations need to be made to a non-modular game environment, the artist must potentially alter assets that were purpose-built for their intended position and appearance. This alteration can involve significant work, up to and including having to recreate the asset from scratch. In other cases, they may be required to painstakingly edit the size, shape, or position of many individual assets to accommodate the required changes. A more detailed or complex environment increases the overhead cost of each alteration.

In a modular environment, an artist or level designer can add or remove modular pieces with little effort, altering the configurations of rooms, landscapes, or other features quickly (Perry 2002, 35). Much of the arrangement of individual assets is preconfigured in each module, minimizing the need to repeat the job of arranging and aligning multiple parts into position. For small changes, individual assets can be altered without fear of how they may affect or intersect their surroundings. When a problem is solved by altering an asset or adding a new modular piece, that solution is propagated and repeatable within the game engine, instantly (Perry 2002, 34).

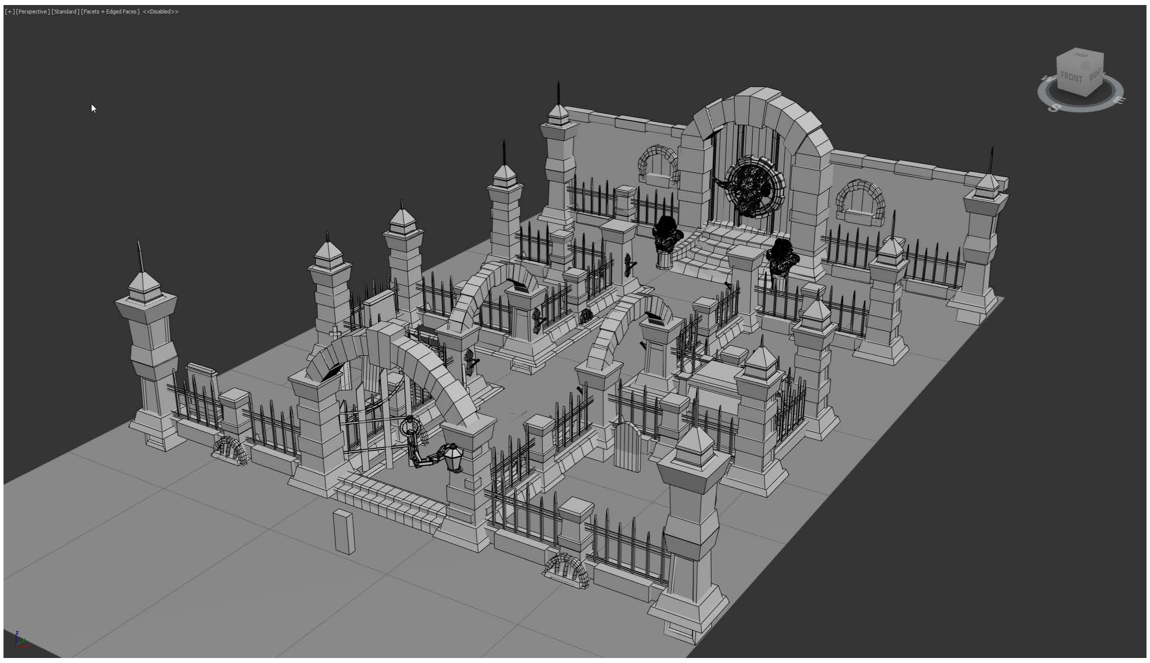
Modularity lowers the overhead of making changes to an environment, resulting in the ability to iterate more freely and rapidly as the project develops. Rapid iteration maximizes the team’s ability to improve their game in response to feedback prior to its release (Goodwin 2017).

2.3.2 Productivity and Time Management

Given the piece-meal nature of modular environment assembly, and the capability of artists to store visual information within specialized textures like normal maps, environment artists can quickly create variant pieces from existing assets. Variations in environments work to increase the immersion of players, minimizing the perception that the same asset has been placed again and again in a game environment. (Pinto 2016.)

In real-world environments, there is variation even in the repetitive elements of our surroundings. In a game environment, the illusion of such variation can only be achieved by creating additional assets. In the case of environment assets such as walls and floors, combining separate pieces and altering UV maps across all textures in a material can create a huge number of variant pieces compared to the number of actual models the artist has created (Klevestav 2010). In a simplified example, a wall section that is comprised of two halves (A and B) can have its halves combined in different ways to create four combinations (AA, BB, AB, and BA) without changing the footprint of the asset. Those four walls can all be used as separate assets with the same shape and general purpose as the original asset, but with added variation. If the artist creates more wall pieces with greater degrees of variation and placement of details, it becomes difficult to recognize re-use of assets.

In a studio environment, it is important to quickly reach an understanding of what a game environment needs to accomplish and how gameplay will happen within it (Pinto 2016). A blockout is analogous to sketching an idea before investing time to paint it; it allows all involved team members to assess how the game environment should be assembled before they invest time in creating the final assets. Typically, primitive 3D shapes such as cubes and cylinders, with little or no texturing, are arranged in a mock gameplay environment to achieve this understanding (picture 13).



PICTURE 13. An environment blockout (Fontaine 2017).

When modular standards exist in creation of the blockout, it is possible to replace the blockout assets seamlessly as they’re completed without altering the scene. Communication between gameplay designers and environment artists is eased this way; the artists know exactly the size, shape, and nature of each module needed, and both parties can refer to the same exemplars when discussing how the environment is taking shape. As assets are finalized, they automatically replace the placeholder assets used for the blockout, allowing immediate visual feedback and perception of the team’s progress. (Pinto 2016.)

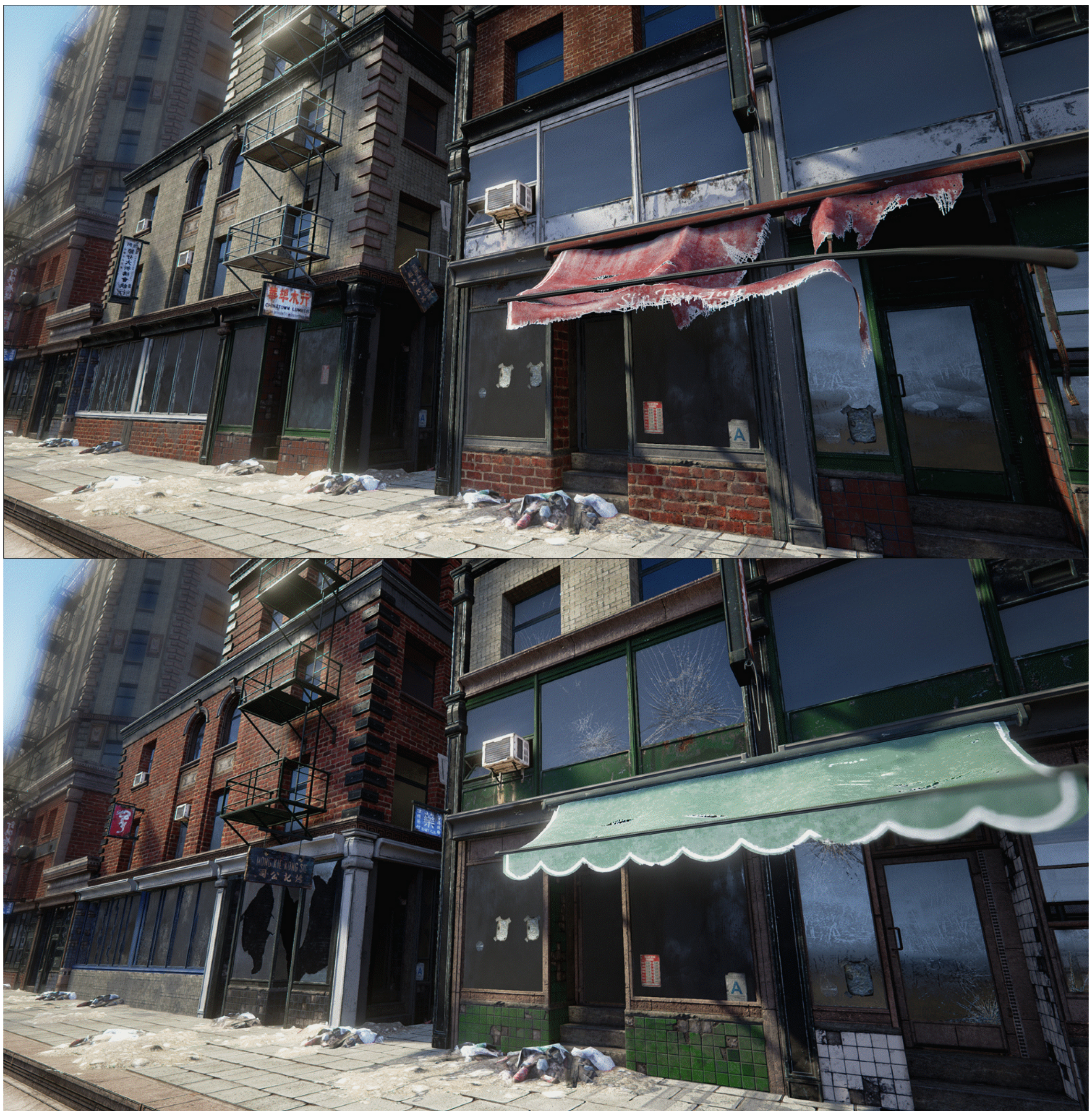
Modularity can greatly increase the speed at which content is created by level designers, which is arguably its most obvious, tangible benefit. According to Burges and Purkeypile (2016), level designers who worked on games like Skyrim and Fallout 4 were the fastest in the industry, so long as the modular pieces they were provided were well-designed.

2.3.3 Interchangeable and Recyclable Assets

In addition to allowing artists to add variation to environments easily, a modular workflow increases the likelihood that assets or parts thereof can be used interchangeably. For example, smoothly transitioning from one style to another within a scene, or retexturing models that already fit together to create a new set of modules that will appear distinct from the original. (Burges and Purkeypile 2016.)

When transitioning between styles, the same modular standards can be implemented across all game environments to ensure that assets can be mixed even in unplanned ways with minimal work. An example of this is the creation of modules that facilitate transitions between two environments across a standardized opening, such as a doorframe that covers the conflicting geometry. These transition assets can be used anywhere the same transition occurs; none of the work involved is applicable only to this original instance.

In addition to standardizing the size and shape of modules, environment artists are able to standardize the UV layout of certain asset types by creating textures that include distinctive features in uniform locations on the texture image. UV layout is stored within the model itself, and thus texture images can be swapped if they are created in the same configuration as the original, seamlessly changing the asset (See section 3.2). Using the various texture types at their disposal, an artist can greatly differentiate assets created in this way, such that the end-user may not recognize that the underlying model has been recycled (picture 14).



PICTURE 14. By swapping materials with identical UV layouts, these modular buildings require no additional modelling but appear completely different (Norris 2015)

2.3.4 Performance Benefits

Every object within a game environment must be rendered at a high framerate each time it is visible to the player. While the performance capabilities of rendering hardware usually come down to speed, texture memory plays a key role in certain situations. Some modular techniques increase the efficiency with which it used. Simplified, the speed at which 3D rendering hardware works today is usually faster than the speed at which each texture can be loaded from its storage location on the hard drive.

As a rule, a game engine loads all textures it expects to need before gameplay begins and manages the loading and unloading of further textures during gameplay. Texture memory is finite, and specifically designed for its purpose. When texture memory is full, a performance bottleneck occurs wherein textures must be constantly reloaded from the hard drive, resulting in intermittent framerate drops as game performance stops to wait for the necessary textures to load (Bavoil 2015).

Some modularity techniques directly alleviate this potential issue. In the traditional non-modular workflow, each object must have its own material, which is uniquely designed for its model and not applicable elsewhere. If there are many different objects on the screen, all of their materials must be managed at once. A key feature of modular design in game environments is the ability to do work once and benefit from it many times over, and this extends to the work of the rendering hardware to some degree. When a modular environment makes use of a common material shared by multiple assets, that material need only be loaded once.

Furthermore, each time an asset is rendered on the screen, a draw call is sent from the game engine to the graphics hardware. Without investigating this complicated process too deeply, it is important to optimize modern games by balancing the draw calls generated with the capabilities of the hardware in the user's device. Generally speaking, fewer draw calls leads to increased performance. Modular assets reduce draw calls in several ways.

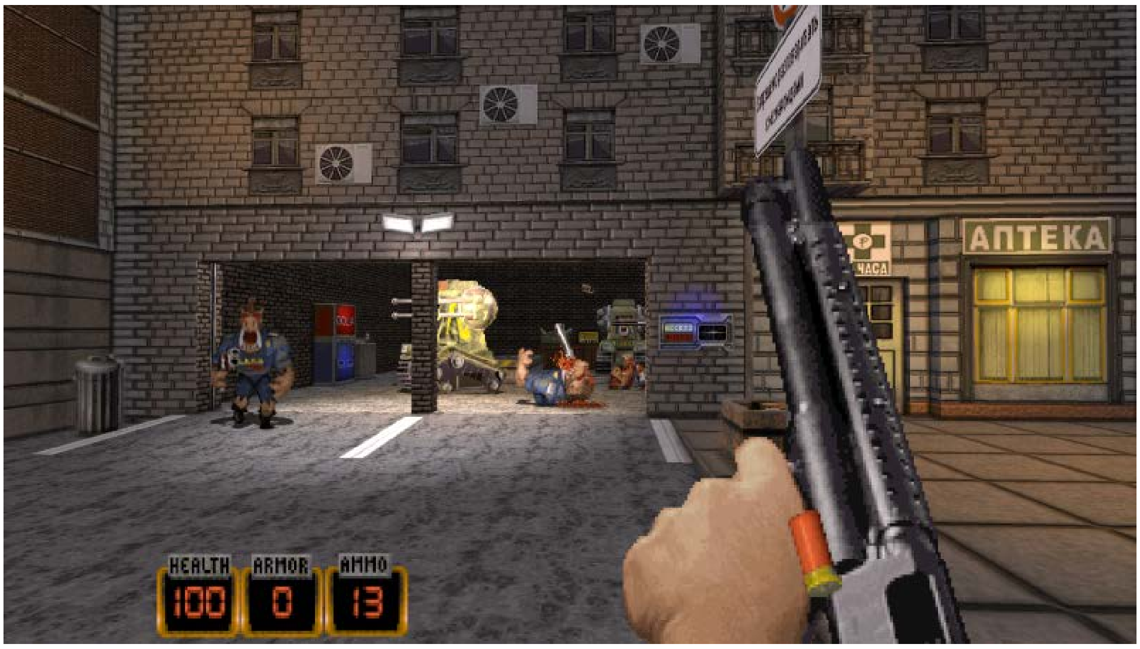
Game engines can consider each placement of a single asset to be an instance, meaning a copy, which should be batched together with all other instances of that asset into a single draw call. Further optimization can be made by batching multiple different assets together into a single draw call, so long as they share the same material, and sometimes other criteria as well. (Unity 2017.)

It follows that using a single material for many assets increases the likelihood that assets can be batched, and modular texturing techniques facilitate this (Piquet 2011). Studios using modular workflows typically optimize their materials for these possibilities, allowing the engine to issue fewer draw call for many objects at once (Burges and Purkeypile 2016).

3 MODULAR ENVIRONMENT WORKFLOW IN GAMES

3.1 Modelling for Coordinate-Based Snapping

Since the invention of the Cartesian coordinate system in the 17th century, we have been able to represent a point in theoretical space with the use of three coordinates (New World Encyclopedia 2018). Naturally, as the internal language of computers is based on mathematics, it follows that the earliest games and game engines used the same coordinate system to determine where an object was located in virtual space (picture 15). Further, the idea of using these coordinates to place objects with mathematical precision is not new. The ability to precisely arrange assets according to a grid or other standardization is generally considered obvious. Ensuring individual assets fit within standard grid units is a foundation of modularity in games (Perry 2002, 32).



PICTURE 15. Early 3D games such as Duke Nukem 3D made heavy use of modular level design, but consumers at the time did not demand as much variation (3D Realms 1996).

Coordinate-based modularity primarily concerns the process of modelling assets and yields great benefit in the later assembly of assets into environments. Industry professionals sometimes refer to these sets of modular assets as environment kits, similar to real-world scale model kits or building blocks. Successful use of the grid snapping technique usually involves more preparation than changes to modelling or texturing workflows.

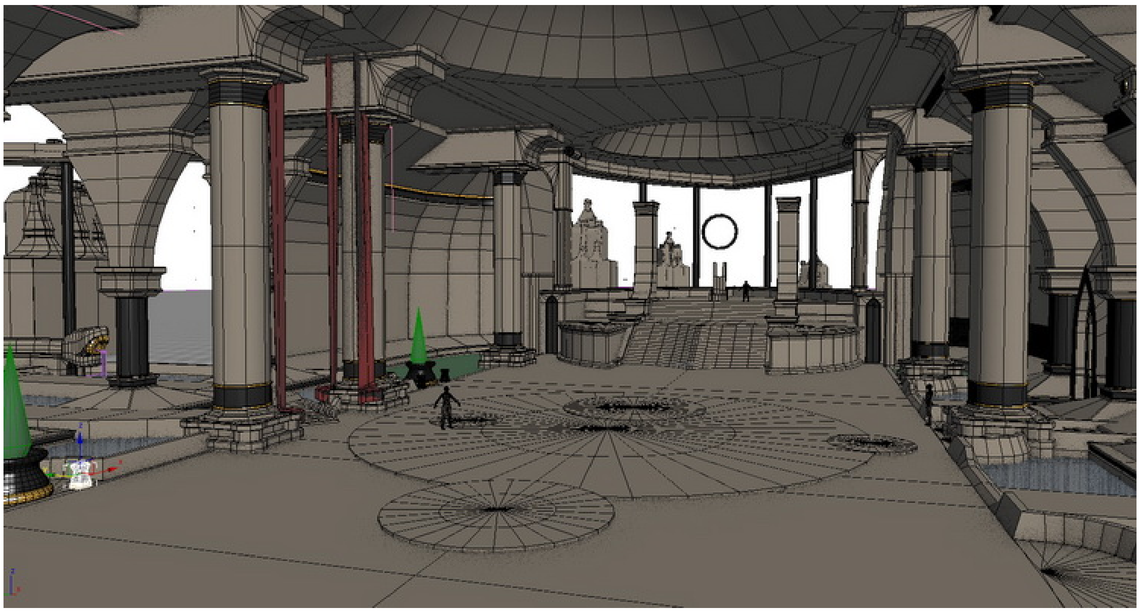
Furthermore, artists creating environment kits must maintain balance at all times between the standardized rules of the modular system and their aesthetic values as an artist. According to Bethesda Softworks level designer Joel Burges (2016), many artists have difficulty working within a set of limitations, and a good kit artist is a rare kind of person who combines expressive and cerebral thinking well.

While the concept is neither new nor surprising, even top game studios who have used modular techniques for decades continue to innovate toward a more flexible and efficient system of snapping assets. There are pitfalls for artists to avoid, and choices which must be made regarding how modular assets fit together in virtual space. Further, designing a modular system within a coordinate grid is a cumulative process. Each decision will have ramifications that will affect, sometimes negatively, decisions made later in the project. (Burges and Purkeypile 2016.)

3.1.1 Preparation

To begin planning for a coordinate-based modular environment, designers must first determine the basic units by which their coordinate plan will be divided. Typically, a number is chosen because it is easily divisible into smaller units, facilitating the creation of assets of various sizes larger and smaller than the default that still fit within the grid. Such a system might be based on real world measurements like square meters, or powers of two (Pinto 2016). In many cases, concept art is used as a basis upon which to understand which pieces need to be created and how their scale can be divided into units.

Designers then consider how to achieve an environment that closely resembles the concept with grid-bound pieces, formulating a basic set of modules that must be created. Using the blockout method, an artist can quickly create a set of pieces that vaguely resembles each intended piece, allowing experimentation within a game engine to determine if changes or additions are needed (picture 16) (Pinto 2016).



PICTURE 16. A blockout made of prototype modular assets. (Holmberg and Klafke 2015).

The difficulty in planning such a system lies mostly in avoiding the player feeling a sense of repetition and disconnection from reality as they move through the environment (Pinto 2016). Certain early 3D games used coordinate-based snapping to procedurally generate huge amounts of content. By today's standards, focusing entirely on visuals and lack of fidelity notwithstanding, the distinct repetition of assets in these early games is extremely distracting to modern users. In their time, early 3D games met the expectations of their players simply by the novelty of operating in 3D space.

Modern audiences expect more realism and immersion into their environment. In the real world, how interesting or compelling a given space feels is determined by many factors, amounting to a science practiced by architects and interior designers. Even in man-made environments like cities, visual interest is found in the form of contrast, harmony, transitions, and details of varying intricacy (Albeluhn 2017). Simplified for the purposes of this statement, overly generic modular assets can lead to a boring space without enough variation to engage players. A well-planned modular system balances variation with standardization, allowing level designers to create distinct spaces with interesting features, with minimal deviation from the intended relationship between modules.

3.1.2 Workflow

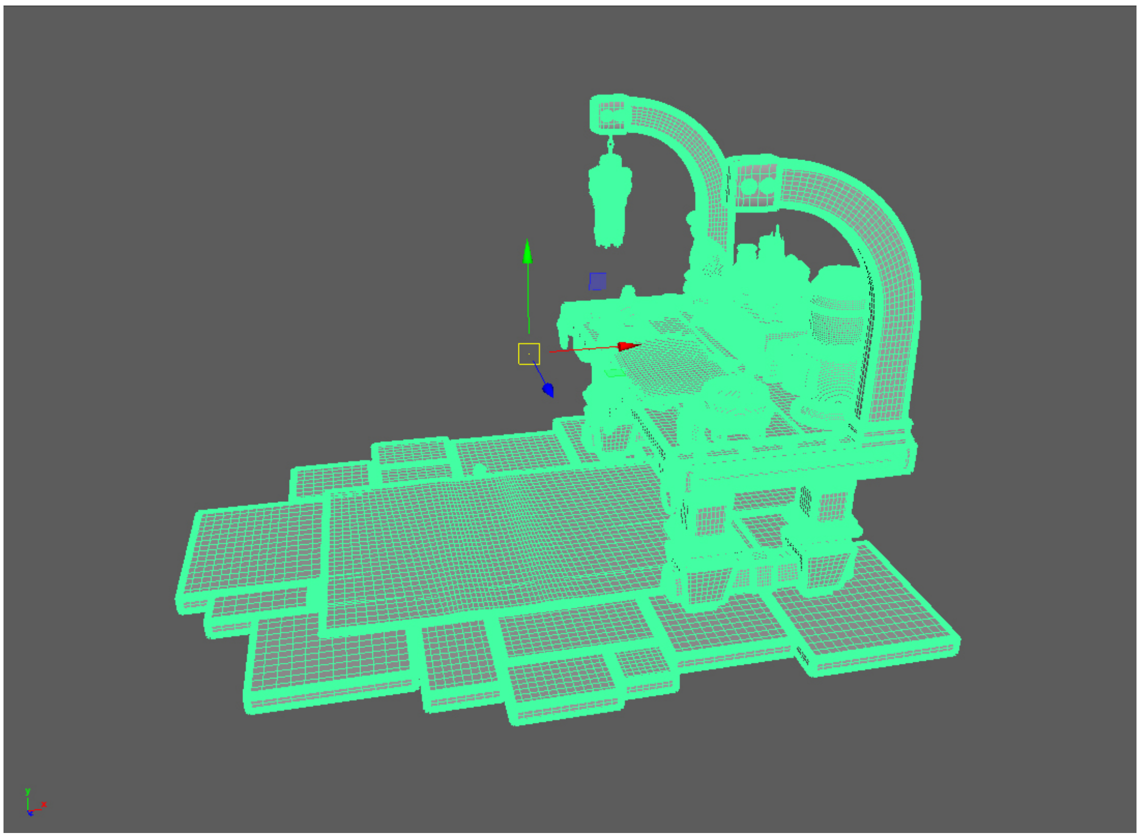
Creating assets for a snapping environment kit involves certain considerations for 3D artists but doesn’t deviate much from the traditional process of creating 3D game assets. Textures are applied after models have been created, and so they fall outside of the scope of this technique, which affects only the modelling stage. Artists must maintain an awareness of how their model fits within the grid space for which it is designed and how it must relate to other kit assets. Standards must be established for where each part of a model should fall in relation to the grid, such as where a wall edge will meet that of a neighbouring piece, or the maximum depth of a depression of protrusion from a given surface (Klafke 2014).

Artists must ensure their assets fit entirely within their allotted grid space, without occupying the absolute edge of it in ways that may conflict with neighbouring pieces. For example, placing walls at the absolute edge of grid space precludes placing similar sections directly adjacent, as two neighbouring walls will occupy the same plane at the shared edge. This results in a form of visual anomaly termed z-fighting, in which hardware attempts to render both objects at the same time. (Burges and Purkeypile 2016.)

Modern 3D modelling software intrinsically operates on a Cartesian coordinate system, easily facilitating the creation of grid-bound assets. An artist can manually enter the desired position of certain points or use in-built features of most 3D software to place edges and vertices on precise measurements within a virtual space. It is possible and sometimes advisable to import finished assets or blockout assets for comparison, observing the way they act when placed in the intended configuration (Perry 2002, 33).

Iteration occurs when an artist or level designer discovers an incompatibility or new requirement. Provided assets have been created according to the standards set in the planning stage, fixing errors typically involves only minor alterations. Where new kit assets are requested, such as in the facilitation of an unexpected transition between two assets, assets can be imported into the 3D software workspace and modelled over directly without disturbing the grid system (Pinto 2016).

A common pitfall in creating modular kits is misplacing the origin point of a model. The origin point is a single point in virtual space by which software understands the model’s location. It is the origin point that is manipulated within a game engine and snapped to points on a grid, and a kit will not assemble correctly if the origin point is misplaced (Mader 2005). Origin points can move frequently during the modelling process, as individual pieces are combined or used as a point of reference, and the artist must take care that it is reset to the correct location before the model is finalized (picture 17). This is a relatively minor consideration for most artists, but noteworthy for the frequency with which it can happen.

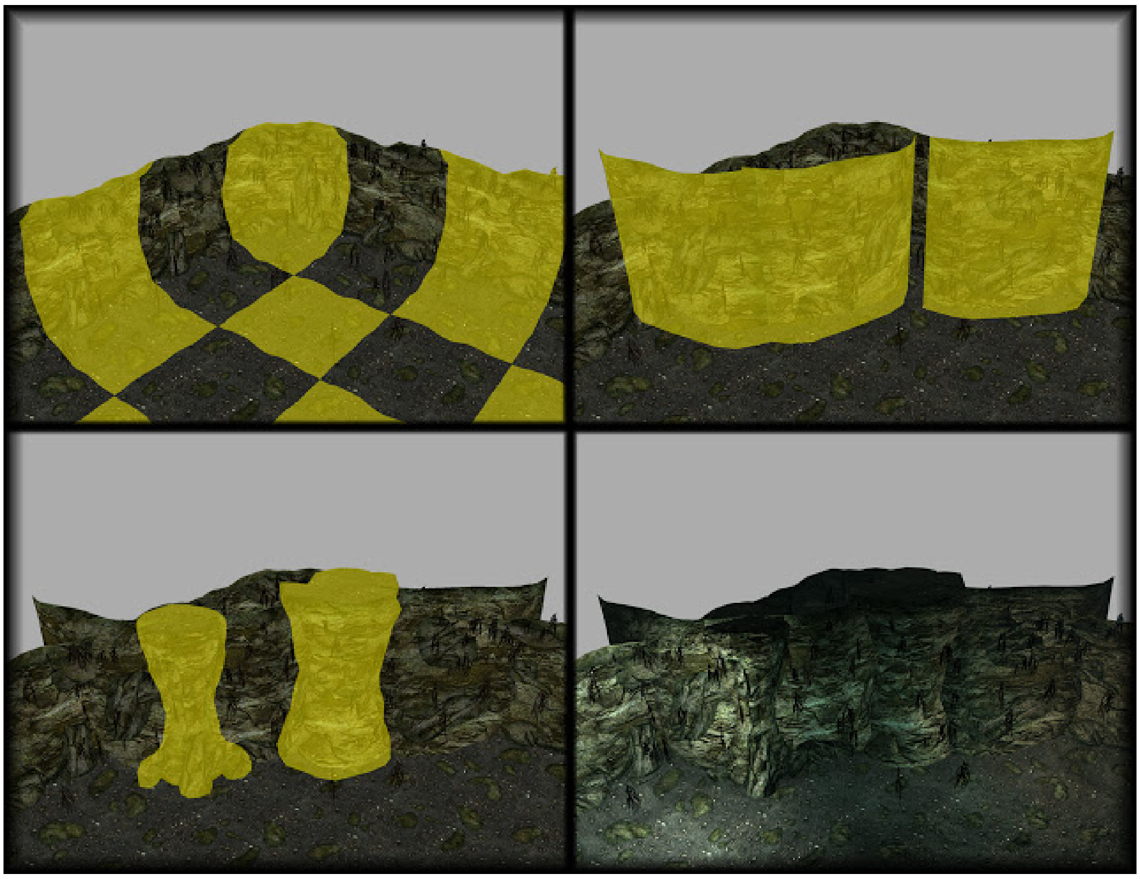

PICTURE 17. Combining many models into a single asset in Maya results in the origin point being moved to the center of the objects. This is not ideal for modular assets, and it must be moved before finalizing.

The placement of origin points can profoundly affect the usefulness of asset kits. In the case of cylindrical objects, origins are best placed on the grid corner around which the asset must be rotated to describe the intended cylinder. Similarly, care should be taken to enable level designers to rotate or mirror assets over their origin point. An environment kit with models that have improper origin points is extremely problematic for level designers, as changes made to the origin point in the name of resolving the issue will propagate to all instances of the asset, potentially invalidating their placement. In general, a level designer should not commit to using an environment kit that has any misplaced origin points, potentially creating a workflow bottleneck in the studio.

3.1.3 Capabilities and Limitations

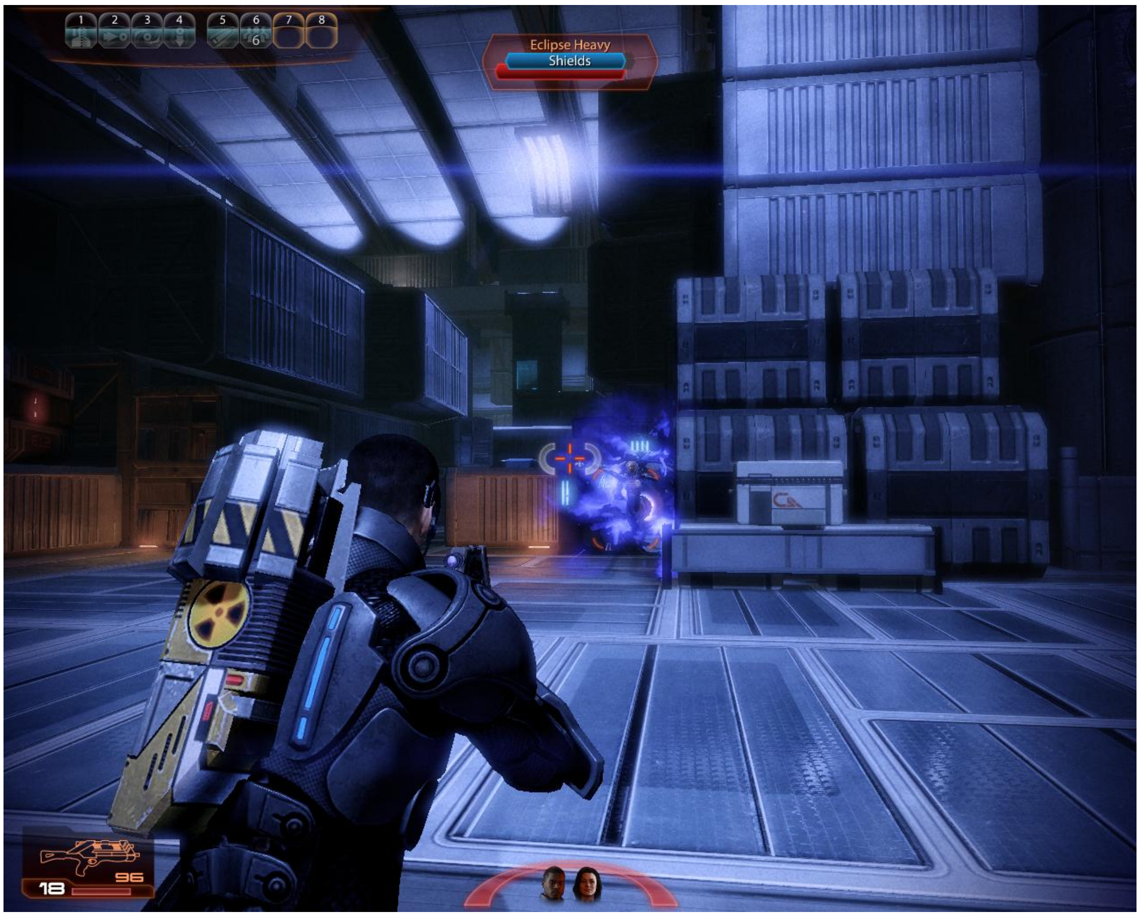
Planned well, a set of assets created for coordinate snapping allows designers to assemble pieces quickly and flexibly, rapidly building a complete environment with the assurance that most configurations will be feasible with no additional work. Snapping environment kits are most useful for creating a high volume of content in a short time. (Pinto 2016.)

In addition to working with the assets as the creator intended, level designers can experiment with mixing assets from different environments in a process commonly called kit-bashing. In most cases, additional work only appears where two kits meet, whereby an artist must create either transitional pieces or disguise any jarring changes in shape or texture by covering them with other geometry (picture 18). Some studios standardize the way kits transition between each other, eliminating this problem early on (Burges and Purkeypile 2016).

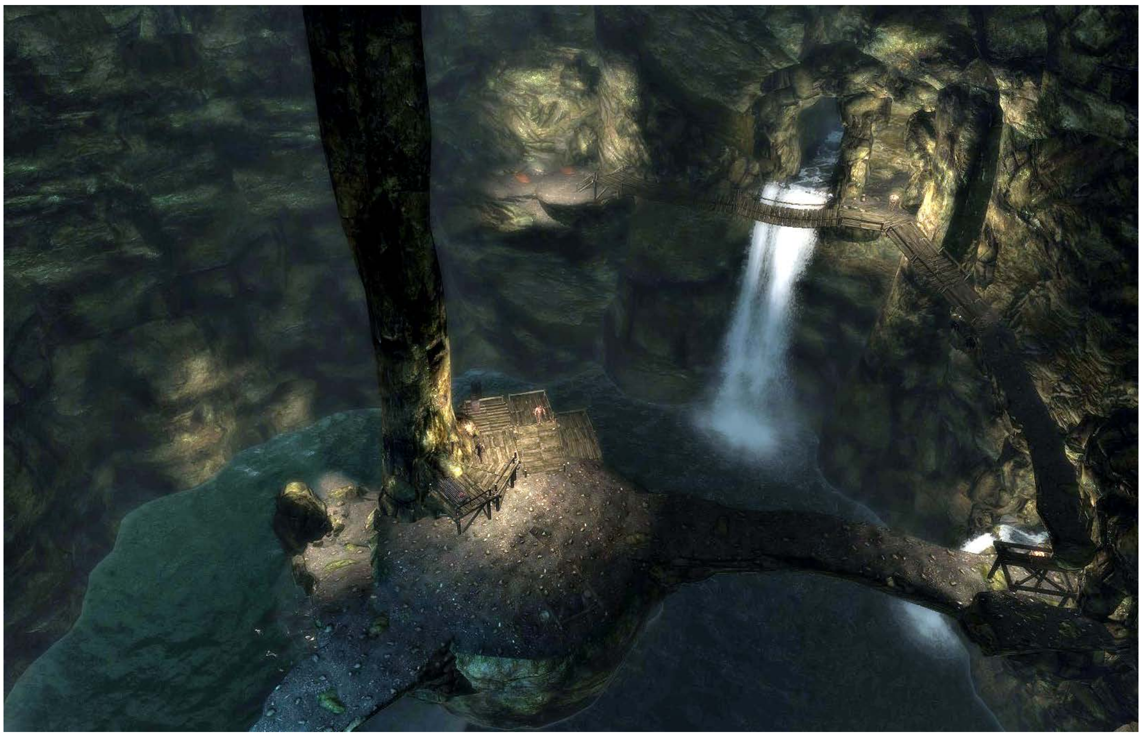

PICTURE 18. Assets in The Elder Scrolls V: Skyrim arranged to intersect one another and conceal a jarringly straight corner (Bethesda Softworks, 2011).

Beyond transitions between kits, it is important to understand that while modular kits have a profound positive effect on the level building process, there are limitations. Cases exist in which the additional planning overhead of creating a modular kit may not be efficient use of time, such as when a particular environment has a very unique aesthetic, small size, or will not represent any significant amount of game content (Klafke 2014).

Grid-based systems of mathematical precision are stylistically suited to modern or futuristic settings, where environments are largely manufactured (picture 19) (Klafke 2014). Creating kits for organic environments such as caves or forests requires more planning and effort from the level designer (picture 20), as well as a more free-form approach to the design of the kit. Even when a kit can be used to lay out a basic environment, care must be taken to minimize any signs of repetition or inorganic geometry. Such a kit may involve irregular shapes that are more difficult to conceptualize within a grid, and artists may need to create a variety of non-snapping assets that can be used to add variation and cover imperfections. The level designer must judiciously and artistically assess how best to use the assets to achieve a convincingly organic environment.



PICTURE 19. Repetition in a sci-fi environment like those of Mass Effect 2 are expected, as such environments are mostly manufactured (Bioware 2010).



PICTURE 20. Organic environments, such as this cave in The Elder Scrolls V: Skyrim, must avoid repetition to create the illusion of a naturally-formed space (Bethesda Softworks 2011).

A studio that chooses a modular approach must also consider the number of objects being individually rendered when a scene is built from many small parts. In such cases, however, large batches of objects can be baked together and drawn as a single object by most engines, alleviating the issue. This process happens as a final pass when the environment is finished and ready for the end-user. (Burges and Purkeypile, 2016.)

3.2 Texturing with Tiling Textures and Trim Sheets

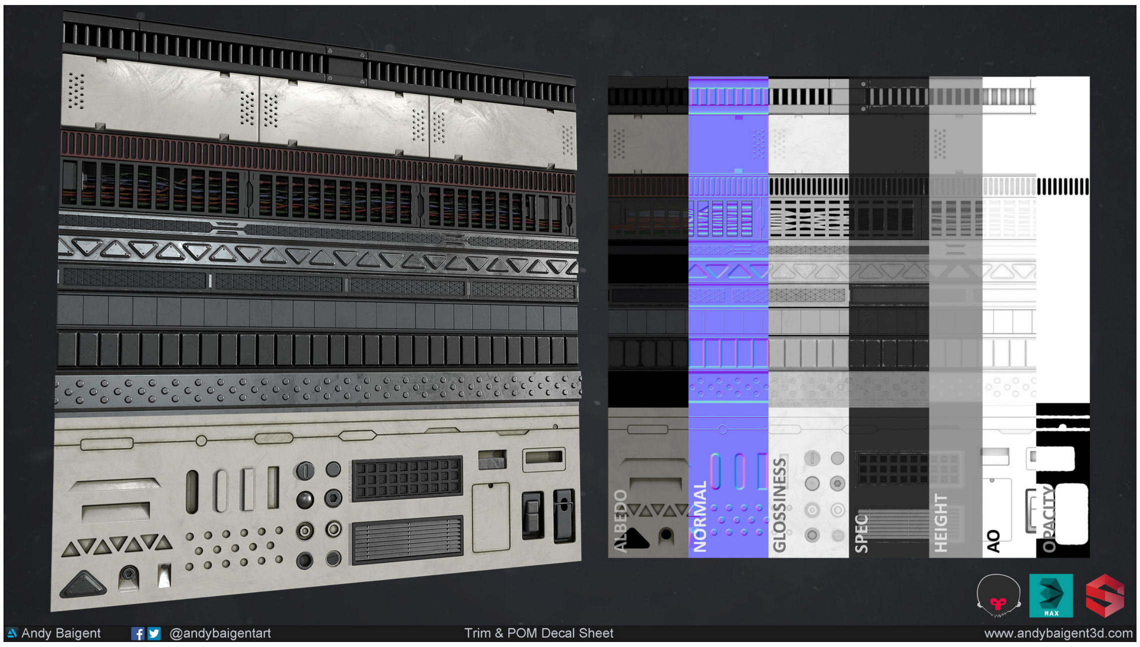
When designing textures for modular environment models, artists commonly break down each object into two simple categories: parts which should use a tiling texture, and parts which should not. Some objects, such as a brick wall, involve flat planes of repetitive texture, and are best represented by creating a tiling texture image. Other objects are more detailed and might include various areas that are distinct from one another and require more specific texturing. Modern game engines allow artists to combine both tiling and other materials in one asset by assigning different materials to different faces of the model (Klafke 2014).

Coupled with the amount of visual information stored in modern game engine materials, including convincing illusions of higher visual detail than has actually been modelled, this capability has given rise to a new industry standard. Increasingly, environment artists are choosing to use a combination of tiling textures and trim sheets to texture models in a modular fashion, without losing any visual fidelity or invalidating baked textures such as ambient occlusion or normal maps.

Trim sheets are an evolution of the texture atlas (picture 21), a technique in which artists combine many textures onto a single image. In addition to offering performance benefits, using a texture atlas can allow artists additional flexibility and production speed under certain circumstances. In the traditional workflow, textures are created for the model which they will cover after it is completed. By contrast, when working with a texture atlas, artists usually map the UVs of a model to an already prepared texture, which allows them to reuse textures across multiple assets that use the same material. Using a texture atlas can provide a wide range of performance benefits (NVIDIA Corporation 2014).

PICTURE 21. An example of a texture atlas. The UVs all models in the scene are mapped to different parts of the image (Ivanov, 2006).

A trim sheet is a form of texture atlas that takes advantage of modern material capabilities. The texture image is precisely divided into areas that encompass all details and variations of the final model or set of models, often including multiple options for certain areas (picture 22). By manipulating the UV mapping, an artist can project different parts of the texture onto parts of each model, rapidly creating variation with no loss in the quality modern textures can achieve in a game engine (Klafke 2014).



PICTURE 22. A trim sheet. The UV maps of different assets will include different parts of this material. All texture maps will apply to all models that use any part of this material. (Baigent 2018).

Other areas of the model are mapped to the tiling texture, mimicking real-world environments which often consist of details set against flat areas of texture, for example the doors and windows on a concrete building. When done well, this technique also allows an artist to texture unplanned assets with relative ease, allowing a single well-crafted texture sheet to cover both current and future assets that will consist of the same materials (Pinto 2016).

3.2.1 Preparation

Planning for the use of trim sheets is dependent upon a complete understanding of the shapes and details a set of models will include. Not only must all needed shapes be anticipated, but texture space must be efficiently divided, room left for variations if desired, and the scale of each area must allow for enough resolution on surfaces without distortion. As this technique has become popular, a number of example layouts that have been presented by accomplished environment artists (Olsen 2015, Pinto 2016).

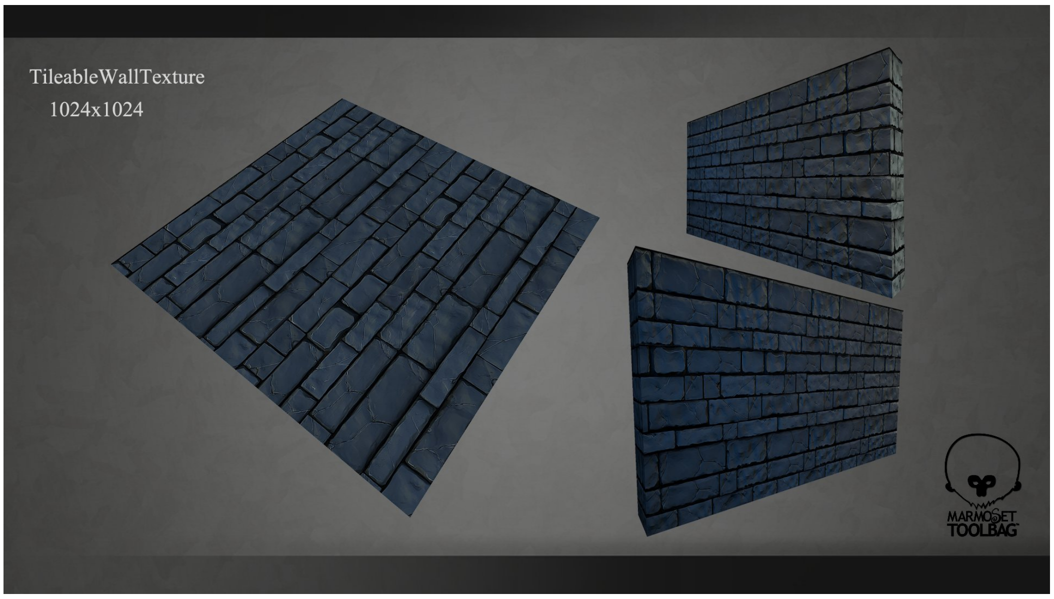
3.2.2 Workflow

Unlike the traditional non-modular workflow, artists using a modular workflow can create their textures before their final models, or concurrently in stages. As with the texture atlas technique, the UVs of each model will be laid out on top of ready-made texture images, and the ability to immediately see textures projected onto the model during this process is one of the main benefits of using this method (Klafke 2014). Using graphics software, the artist precisely lays out the texture, either by creating each within its given space, or creating it separately and combining the textures into one image afterword.

Additional texture maps are created with the same layout, often using a copy of the original diffuse texture as a base and working on top of it. Given the image-wide nature of the adjustment tools in most graphics software, this process can be highly involved. Artist may be required to individually section off and uniquely alter different parts of the image as appropriate for the texture type they're creating and its intended look in a game engine.

If pseudo-3D textures like normal maps are involved in the process, they must usually be baked in one of many ways and in accordance with standards set in the planning stage. An artist may choose to sculpt the forms depicted in the texture image and bake normal maps from the 3D model that results. It is also possible to use software to create normal maps without sculpting, either with software that interprets the image as a 3D surface, or hand-placing features, depending on the tool (Pinto 2016).

Either before or after the trim sheet, the artist must also create one or more tiling textures picture 23). The process usually involves modelling or sculpting with the use of tools or modifiers in modelling and graphics software that aid in ensuring the texture repeats seamlessly along its edges. After creating other texture maps as required, the result is an image that can be infinitely repeated across a surface with the full visual fidelity offered by modern texturing techniques (Silverman 2013).



PICTURE 23: A seamlessly tiling texture (Vermosen 2018).

Laying out UVs must be done with precision, as the surfaces of each model must be adapted to fit the trim sheet layout. This can potentially include cylindrical or circular surfaces that must be made to conform to a linear section of the trim sheet, but so long as this is done with care, the intended texture will be projected seamlessly onto any shape. (Olsen 2015.)

Working with tiling textures and trim sheets has far-reaching consequences for the overall asset creation workflow. Primarily, the rigid structure of the traditional non-modular process (see section 2.1) is reorganized, as modelling and texturing can be done in any order the artist prefers, and both of these as well as UV mapping can be adjusted with minimal consequences. Adjusting the model may require some further adjustment of the UVs, but textures will be undisturbed. Editing the textures will provide instant visual feedback when applied to the model and has no effect on the model or UVs so long as the layout of the trim sheet is not changed. Changing the UV mapping should have no effect on either the model or textures, but will provide instant visual feedback, allowing the artist to make difficult adjustments by eye. These differences fundamentally change the overall workflow of creating the assets to one that allows for iteration at all stages (figure 2.)

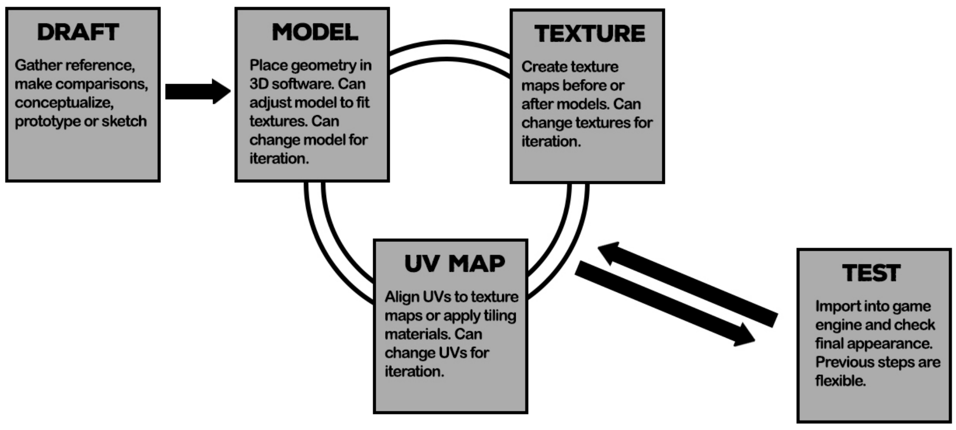

DRAFT
Gather reference, make comparisons, conceptualize, prototype or sketch
MODEL
Place geometry in 3D software. Can adjust model to fit textures. Can change model for iteration.
TEXTURE
Create texture maps before or after models. Can change textures for iteration.
UV MAP
Align UVs to texture maps or apply tiling materials. Can change UVs for iteration.
TEST
Import into game engine and check final appearance. Previous steps are flexible.

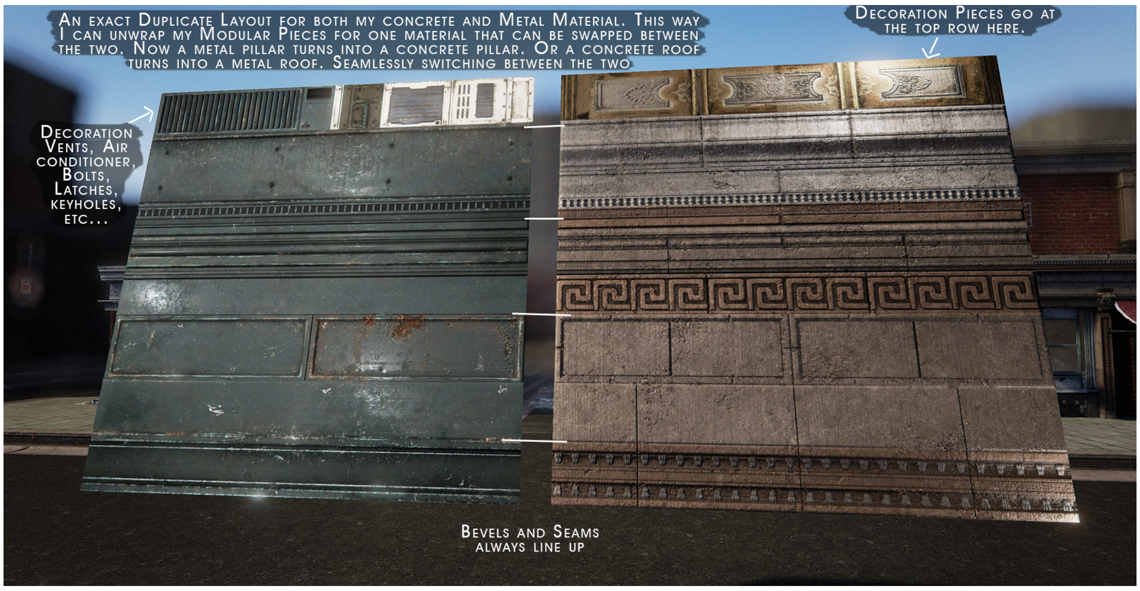
FIGURE 2. The asset creation workflow is flexible and iterative using tiling textures and trim sheets.

3.2.3 Capabilities and Limitations

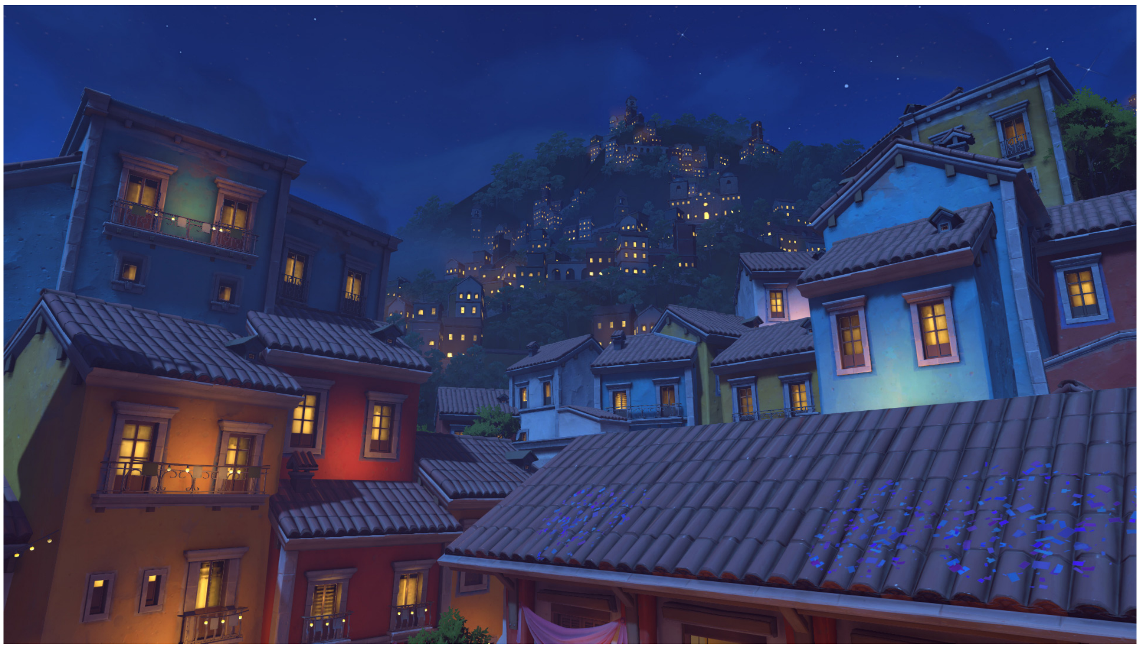
Through careful planning, it is possible to benefit from the use of trim sheets in a variety of ways that cannot be achieved with any other technique. In particular, the use of trim sheets benefits productivity by allowing artists to easily create variations on a single model and use the same texture sheet on almost any asset (Klafke 2014).

Standardizing the angle of bevelled elements before creating normal maps can enable artists to seamlessly match up the edges of UV shells to create the illusion of higher polygon density. Doing so requires a high degree of precision in both sculpting, texture baking, and UV placement. Some artists advise using only 45-degree angles in all cases, ensuring compatibility along any edge without anomalies. Once completed, modular assets textured in this way are nearly impossible to distinguish from assets that have been sculpted and baked from a high-poly variation. (Olsen 2015.)

It is also possible to completely change a given model by changing its material, so long as the textures of both materials are laid out identically (picture 24). Using this method, an artist can create endless color variations or alterations to the material of an asset (picture 25).


An exact Duplicate Layout for both my concrete and Metal Material. This way I can unwrap my Modular Pieces for one material that can be swapped between the two. Now a metal pillar turns into a concrete pillar. Or a concrete roof turns into a metal roof. Seamlessly switching between the two
Decoration Pieces go at the top row here.
Decoration Vents, Air conditioner, Bolts, Latches, keyholes, etc...
Bevels and Seams always line up

PICTURE 24. These two trim sheets are laid out identically and can be swapped between models without changing the model or UV map (Norris 2015).

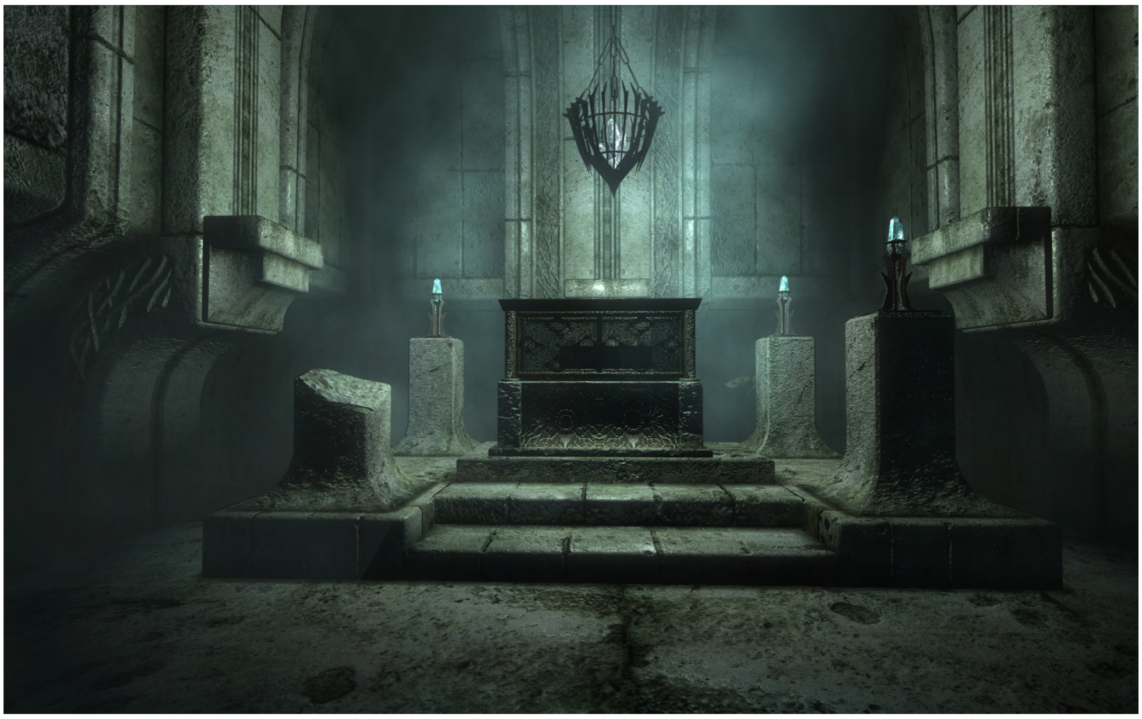


PICTURE 25. In this screenshot from Overwatch, note the number of identical building models that have been assigned different materials to change their color and mask their repetition (Blizzard Entertainment 2016).

The benefit of using tiling textures is straightforward. Environment assets can seamlessly connect to create the perception that they form one large object. If a large object, like a building, were created as a single asset without the use of tiling textures, a texture would need to be large enough to cover its surface without appearing distorted. This would potentially impact performance by requiring a larger texture image and may also require more time to create due to the volume of work involved.

3.3 Minimizing Art Fatigue

Art fatigue is a term used at Bethesda Studios to describe the conditions of a player becoming complacent and losing interest in an environment due to excessive reuse of assets, repetitions, or an overall sense of sameness. It is difficult to create an environment asset kit that, in and of itself, avoids this feeling. Modular environments are completely dependent on the reuse of assets, so modular workflows typically include steps taken to avoid art fatigue. Neglecting this part of the process can lead to broken player immersion, or in worse cases, the perception that the game is of lower quality than similar products without this issue. Such was the case with some of the criticism levelled at dungeon environments in The Elder Scrolls IV: Oblivion (picture 26). (Burges and Purkeypile 2016.)



PICTURE 26: This room configuration from The Elder Scrolls IV: Oblivion is visually interesting but was placed in many dungeons without any variation (Bethesda Softworks 2006).

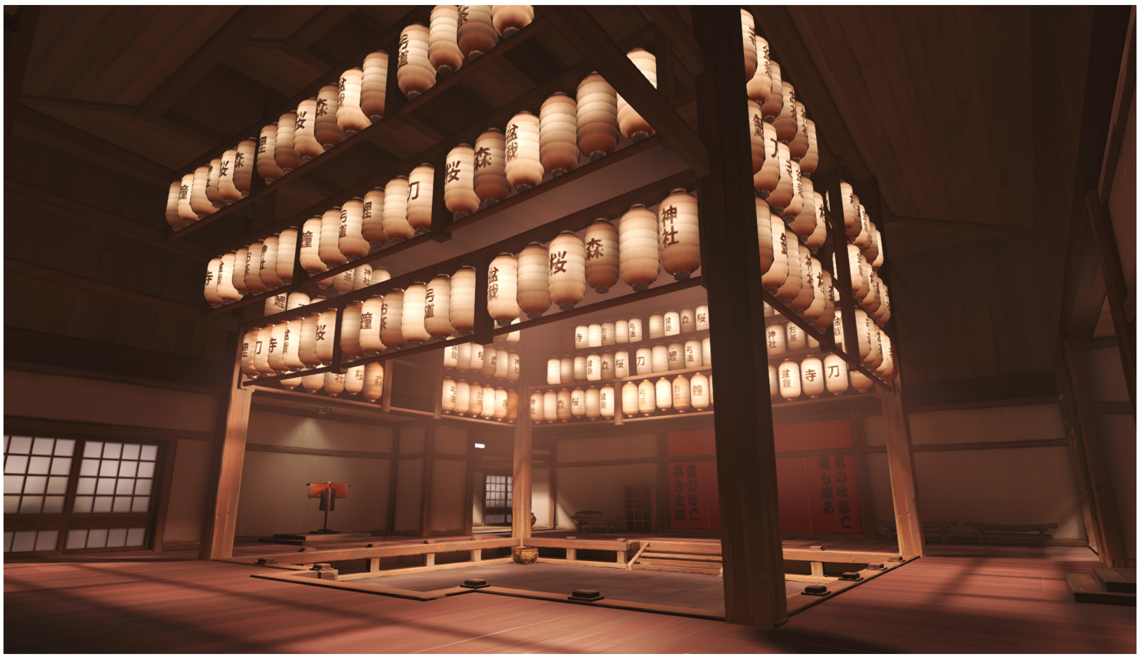
The perception of repetition and onset of art fatigue is more immediately noticeable in repeated details within a scene, more so than “big picture” elements. For example, in a modular room environment, players are more likely to notice four identical pieces of furniture than the modular nature of the walls. This could be linked to our tendency to expect and become complacent with large features of our environment, like walls and floors, instead creating focal points around the objects which occupy our spaces.

What follows is an examination of three common techniques used to combat art fatigue in game environments. It bears pointing out that the ultimate method of avoiding repetition is to create as many different asset variations as possible (Pinto 2016). Unfortunately, doing so increases the difficulty of effectively using and understanding an asset kit, may negatively impact device performance, and most obviously increases the workload involved in creating each kit. There is a point at which creating additional assets becomes less economical than using other techniques to minimize art fatigue, and studios must strike a balance between the two.

3.3.1 Hero Assets

Assembling an environment from an asset kit sometimes yields a complete and well-optimized space for gameplay, but one which is comparatively empty and lacking in detail that distinguishes it from similar spaces. For reasons of both aesthetics and gameplay, it can be beneficial to add features that add context to a specific space and create visual interest. Frequently, artists create unique assets for this purpose, which fit the rest of the assets used in the area but are not intended to be used as a repeatable module. Borrowing a term from the film industry, these unique, high quality pieces are intended to be a focal point of the scene and are called hero assets.

Combining hero assets with an otherwise modular environment distinguishes one space from others, and hero assets are often centrally placed and well-lit to draw players' attention (picture 27) (Pinto 2016). This mix of modular and unique assets benefits from all that coordinate-based kits add in terms of efficiency and flexibility, while also directing players to perceive the space as unique and memorable.



Picture 27. This environment from Overwatch is built from modular pieces, except for the central element which draws enough focus that most players will not notice repetition in the walls and floors (Blizzard Entertainment 2016).

Hero assets are an important part of creating visual interest and breaking up repetition in modular environments. However, artist Jerryn Goodwin (2017) cautions that they must be used sparingly, as they usually demand more system resources to render and much more time to create. Goodwin suggests (2017) that hero assets are best placed in areas of low pressure or pacing, to allow players to appreciate the artwork.

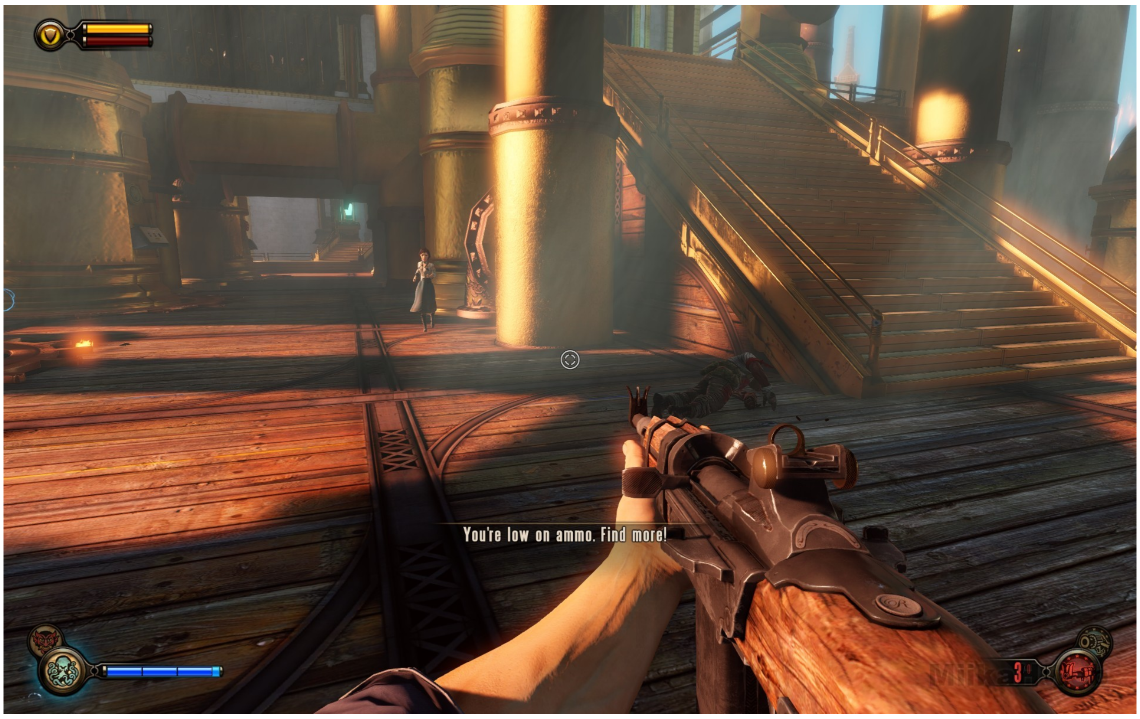
3.3.2 Lighting

The science of light and vision is broad in scope, yet for the average person, these things are taken for granted. The way we perceive light is a primal sense that affects our interpretation of our surroundings deeply. Along with developing techniques for creating assets of higher visual quality, many 3D artists develop their understanding of light, and some studios even employ lighting specialists.

Aside from allowing a player to find their way from one place to another, lighting has a profound effect on how humans perceive space and form. It is possible to manipulate mood and perception by placing, tuning, and moving light within a scene (Pinto 2016). Some elements can be brought forward as focal points, while others can be pushed into the background. The color of light and contrast of shadows directly influence our perception of an object’s color and texture.

Increasingly, game engines are able to realistically represent the behaviour of light in space, and modern hardware can handle the calculations required to do so. While high quality lighting effects are one of the most performance-demanding ways to increase visual fidelity, they are also one of the most visually striking. Modern game engines make it possible to calculate high quality lighting and bake it into another form of texture image, a light map, that is applied to all assets within a scene. This process can provide some of the fidelity of real-time lighting calculations across a scene with significantly less performance impact. The technique is seen extensively in modern games such as Halo 4, where it was used to great effect (Pepera 2012).

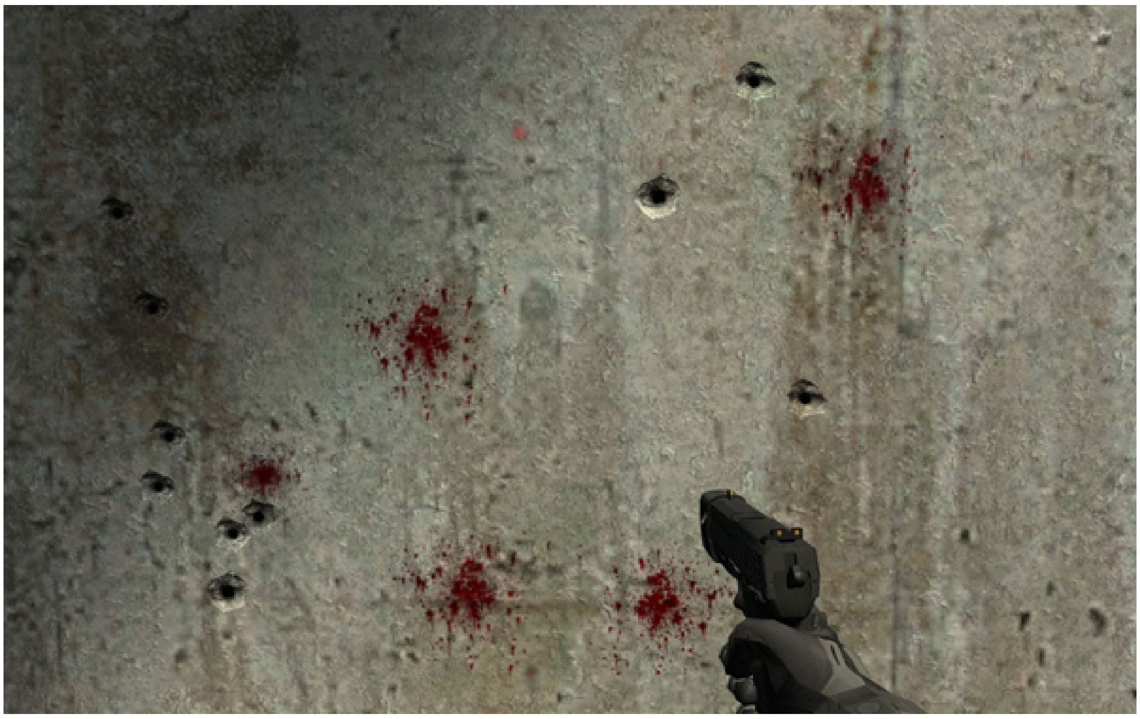
In either case, lighting that is varied and affects surrounding assets is capable of diminishing art fatigue (picture 28) (Perry 2002, 34). In the case of real-time lighting, reflections and shadows move across environments, and colors blend and refract across surfaces, disguising repeated details. Baked lighting creates variation by applying the influence of light as a texture, locally, on top of any repetitious asset textures beneath.



PICTURE 28. The lighting in this Bioshock: Infinite environment adds layers of variation and contrast to the underlying modular elements (Irrational Games 2013).

3.3.3 3D Decals

The word decal is used to describe an asset that consists if an extremely simple model, such as a flat plane, that is placed close against the surface of another asset such that the player perceives the decal as an attached detail (picture 29). At its most basic level, the use of decals enables artists to place images arbitrarily on other surfaces, similar to hanging a poster on a wall. Placing decals is an effective technique for covering repetitive details or creating a focal point on an object where one hasn't been modelled, or possibly covering one that has.

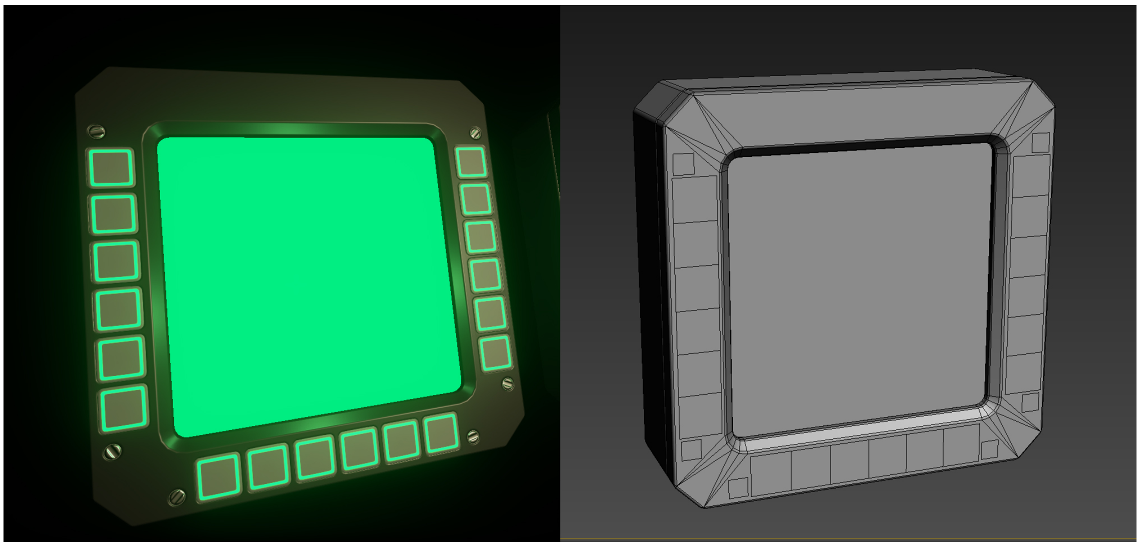


PICTURE 29. The blood and bullet holes on this wall are decals placed dynamically by the player’s interactions with the environment. The underlying model has not changed (Silverman 2013).

Furthermore, decals aren’t limited to flat surfaces. An artist can create specific shapes which can be placed around curved or irregularly shaped surfaces to add details that are not desired on the model itself.

By using images with alpha transparency, which determines the opacity of each pixel in a texture image, the borders of non-rectangular images are rendered invisible, allowing complicated 2d shapes to be seamlessly placed, such as graffiti on a building. The addition of other texture maps to a decal’s material make a wide range of effects possible. (Silverman 2013.)

Mostly-transparent decals with a specular map may indicate the presence of wetness on a sidewalk. Normal maps can be added to decals to create the appearance of a protrusion or depression in a flat surface, where none has been modelled, for example to create the appearance of surface damage. Decals can also include an emission map, which can be understood as an image that is rendered in a game engine as if it was producing light, unaffected by surrounding lighting conditions. This technique is often used to create the impression of things like heated metal elements, glowing embers, neon signs, or electronic screens (picture 30).



PICTURE 30. Every button and screw on this screen is a decal placed on a very simple underlying model in Star Citizen (Lovas 2015).

Decals express the capability of texture images to store visual information in a very direct way. Used well, they lend artists the ability to change the appearance of an asset without creating a full variant of the asset, usually requiring far less work. As a simple individual asset, a decal can be moved or toggled in and out of existence easily by game engines. Whether the decal is used in a temporary fashion, to break up repetition, or in other ways, they can be a powerful tool for minimizing art fatigue.

4 CREATING A MODULAR ENVIRONMENT

4.1 Goals and Scope

Creating a modular environment kit will provide insight into other benefits and drawbacks of using these techniques. Specifically, it was hoped that experiencing this process would make it possible to draw conclusions for comparison between modular and non-modular environment asset creation. It should be noted that nearly all game environments make use of modularity to some degree. Some may be working with grid coordinate, using several techniques but not all, or using them on a case by case basis without formalizing the process. The definition of an environment is also subject to many interpretations. Therefore, it is difficult to define what constitutes a completely non-modular environment without making a forced comparison of games that are fundamentally different in genre and purpose.

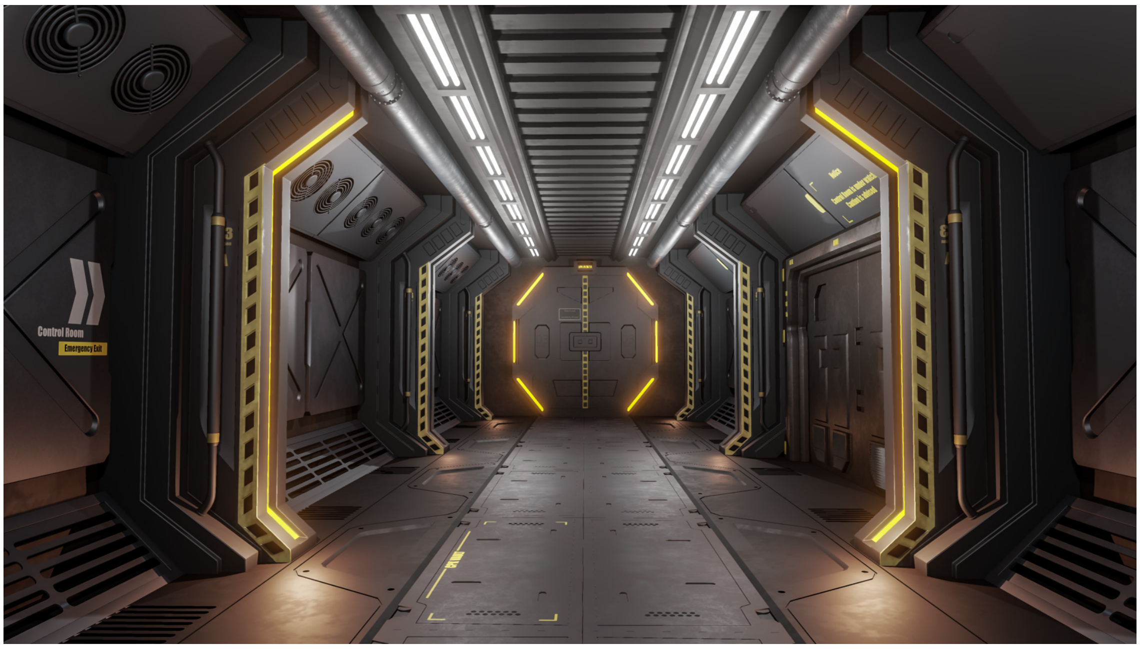
To mitigate this issue, comparison was made between the workflow and process of creating modular versus non-modular assets, not the results of doing so. This example process focused specifically on using all the techniques discussed in this thesis.

While it is possible to achieve high visual fidelity in a modular environment, the focus of this example was on the workflow, flexibility, and speed. Creating a 3D environment is time consuming and being overly concerned with aesthetics may work against this focus. Therefore, the result was expected to demonstrate the use and benefits of each technique in a readable way, without achieving any set visual standard.

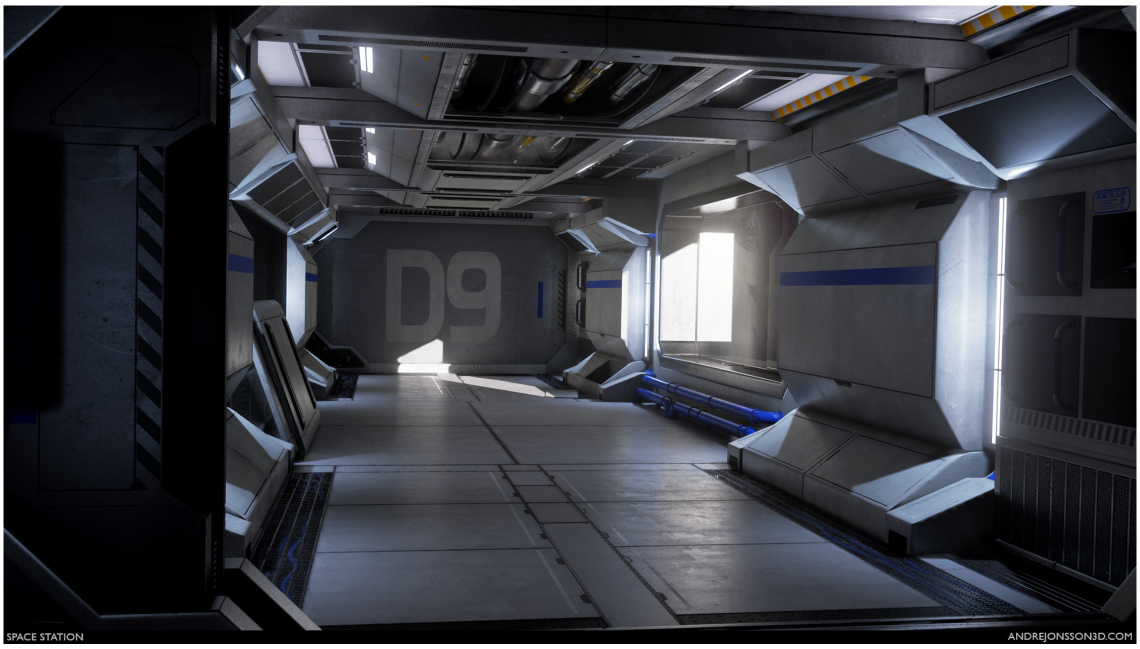
A science fiction theme was chosen, as futuristic environments lend believability to a modular kit by their manufactured nature. The kit needed to include several wall, floor, and ceiling modules, to create a corridor with a corner and a door at the end.

4.2 Planning and Blockout

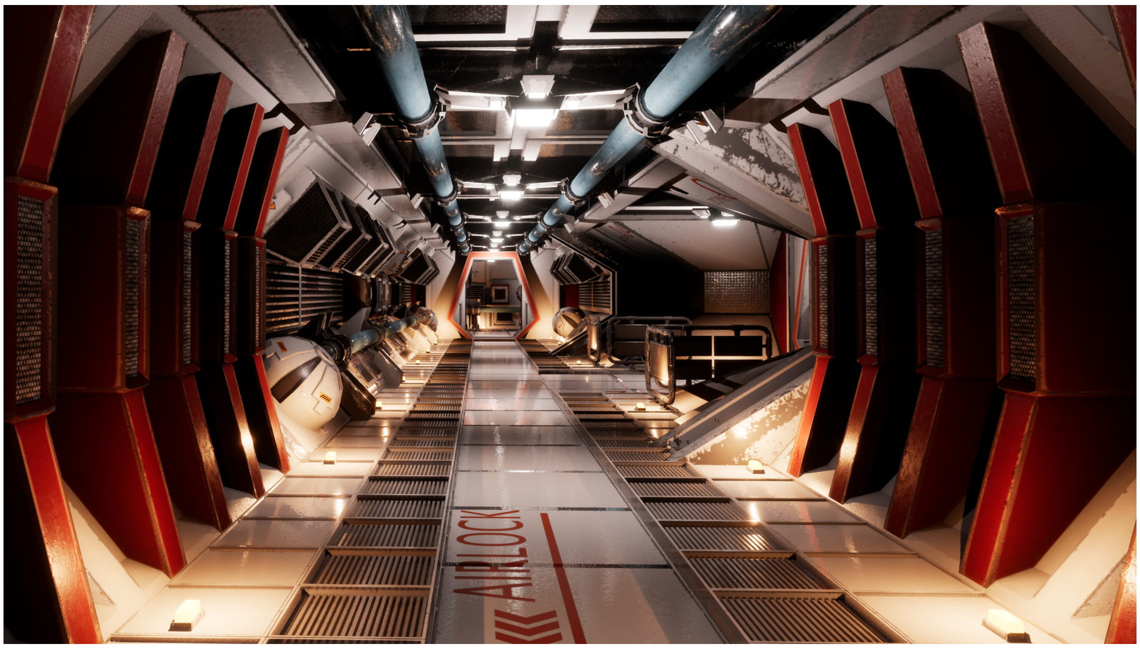
As suggested in section 2.1, this process began with a planning stage. Finding reference images allowed tangible aesthetic choices to be made by comparing what was and was not desired. The theme was narrowed down to facilitate a quick search: a space station corridor with sectioned, panelled walls, compartments and access hatches, and ventilation. Below are some of the images chosen as reference for this theme (pictures 31, 32, and 33).



Picture 31. A simple but effective level of detail, open planes for decals, neutral metallic colors with an accent color. Linear elements describe the shape of the hallway and lead the eye to the end (Oakes 2017.)



Picture 32. Very utilitarian; ceiling space used as conduit, all panels have an obvious means of access. Removed panel breaks up unrealistically pristine shot (Jönsson 2015).



Picture 33. No common shape language. Viewer’s eye bounces around to areas of high contrast in color and conflicting lines. Viable visual style, but not desired for this project (Williams 2018.)

As the scope of this project was clearly defined, it was not necessary to create a wide variety of different asset types. The corridor consisted of wall, floor, and ceiling sections, a door, support structures that join wall sections, and a corner section. Variation in the walls, floors, and ceilings was achieved by making minor changes to geometry, then by mapping the UVs to the trim sheet differently. Doing so took less time than creating new pieces, while creating actual geometry variation

As advised by industry experts, a working scale was chosen to guaranty uniformity of assets in grid space. A measure of 16 units was selected for wall, floor, and ceiling elements. While not an industry standard, a project of this size did not need such granular modularity as offered by units of 128 or 256.

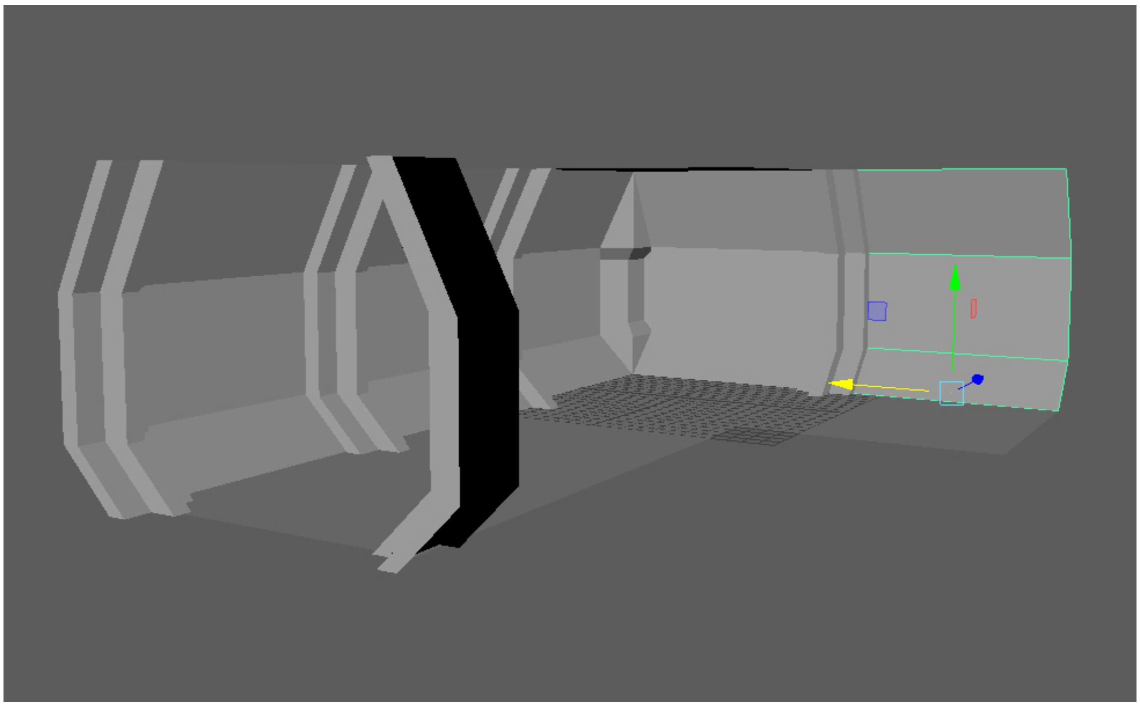
The door at the end of the corridor was intended be a hero asset. In this project, however it is impossible to differentiate the hero asset from other assets, as only one scene is shown and there is no context under which to encounter it. Viewers of a project result like this, consisting of a few renders, have no way of knowing that this is a unique piece that is not repeated throughout a theoretical game. However, it was planned to create a focal point via lighting, contrast, and location, achieving its purpose even without these points of reference.

Lighting was to be kept somewhat dim, with focal points created via small light sources in the ceiling. The contrast of well-lit features and shadows was expected to further reduce the appearance of tiling and repetitive elements. Colored lighting was considered as an option to further enhance contrast, but this was better decided after the assets were assembled.

Decals would be used to differentiate certain planes. Labels and other text are fitting to the chosen theme, but emissive decals were planned for various indicators and screens throughout the assembled corridor. These high-contrast points of light were meant to draw attention, placed randomly on some indicators and not others to help break up repetition.

In terms of planning this environment, the process of choosing aesthetics was the same as it would have been when working non-modularly, or creating a prop, character, or other asset that is not an environment. As well, dividing the environment into individual assets that must be created is normal. In this project, however, perfect formalized modularity was desired, which is not always a chosen objective in game studios. The thought process was different in that each asset had to fit within its intended grid space

A blockout was created in Maya using untextured primitives with minimal modelling (picture 34). Due to the simplicity of the layout chosen, a detailed blockout was not necessary, but this process did provide an opportunity to make decisions concerning overall shape. It was expected that it would be possible to model and refine these initial blockout shapes directly, without changing their place, ensuring there was no deviation from the grid.



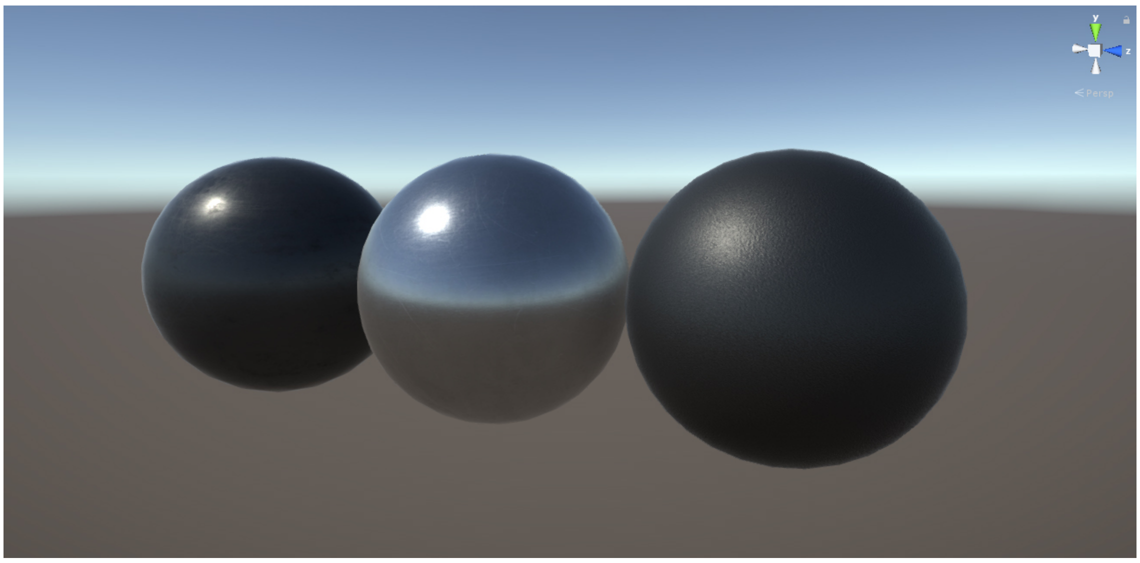
PICTURE 34. A blockout of the hallway, with some pieces hidden for viewing.

4.3 Creating Textures

As some artists advise, textures were created before assets were modelled. Using tiling textures and trim sheets enables this workflow, as well as requires it to some degree, as UV mapping is done on a finished texture image with this method. To facilitate creating realistic textures quickly, texture libraries were used as a base and altered, combined, or otherwise manipulated to achieve a desired aesthetic.

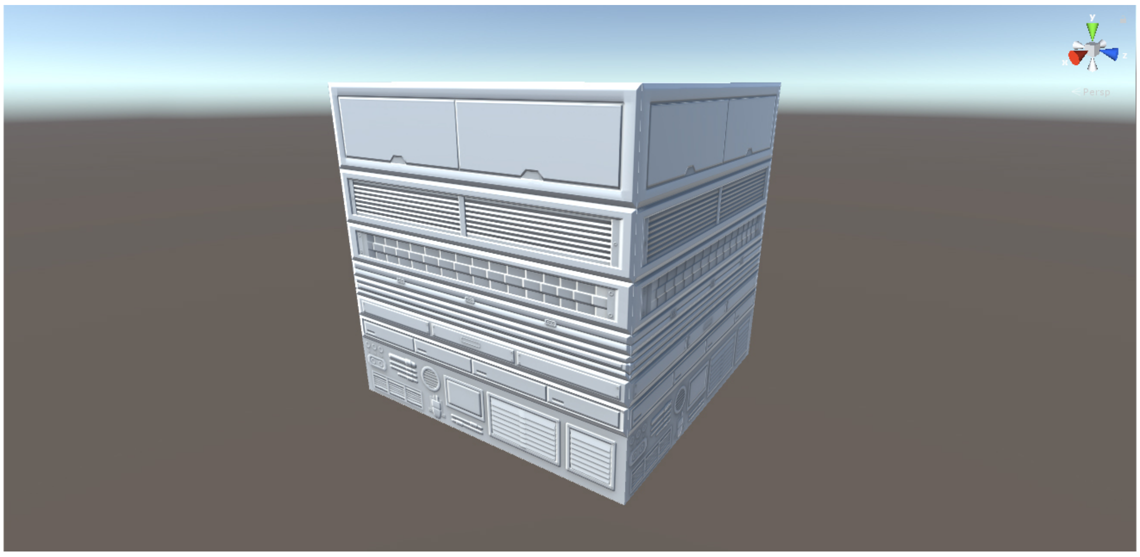
Three tileable materials were created first: a main metallic wall material, an accent metal material, and a floor material. This choice was made arbitrarily, but with insight gained from reference images. There is no guideline save for the sensibilities of the artist to determine how many tiling materials would be needed. In this case, reference images have revealed a desire to maintain a smooth and uniform look, without excessive variation. The finished tiling materials were going to be used as a base for the trim sheet, and so had to be completed first.

Creation of tiling materials is straightforward, and identical whether textures are chosen from a texture library or hand-made. If the image doesn't already tile seamlessly, it can be made to do so with features of most graphics software. Here, Adobe Photoshop's offset filter was used to non-destructively shift the image directionally and cover the image seams using tools like clone brush or blur. Alternately, some 3D sculpting software can facilitate the task of sculpting tileable textures by automatically placing planes as instances adjacent to the working plane. In this way, work along one edge of the model is mirrored on its opposite edge, ensuring a perfect tiling effect. For this project, sculpting wasn't necessary: the needed tiling textures were smooth, with little variation. The three textures created for this project can be seen below (picture 35). Note that the final look of each material was intended to be refined when it was possible to view them together on finished assets.

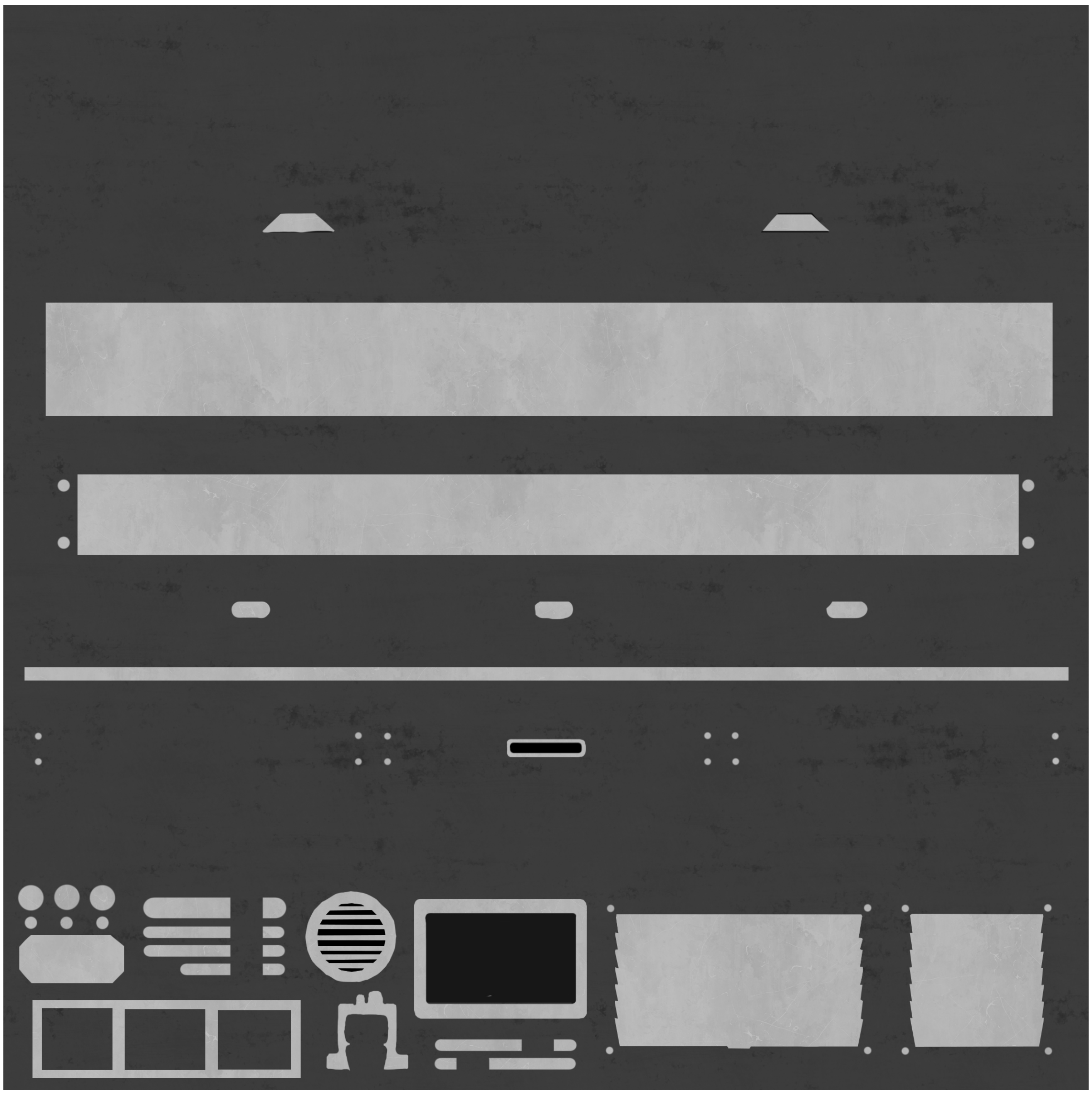


PICTURE 35. Three tileable textures created for this project, applied to spheres in Unity. Plastic, stainless steel, and rubber.

A trim sheet layout was chosen from various examples available, and a square plane was divided into the chosen layout. Details were then added to the trim sheet via hard surface modelling in Autodesk Maya 2016. The hard-edged, manufactured nature of this environment precluded the need for sculpted detail as might be appropriate for more organic settings. The nature of the details was chosen after generating a list of ideas and sketches. The finished model was used to bake a normal and ambient occlusion map, then inspected in Unity.

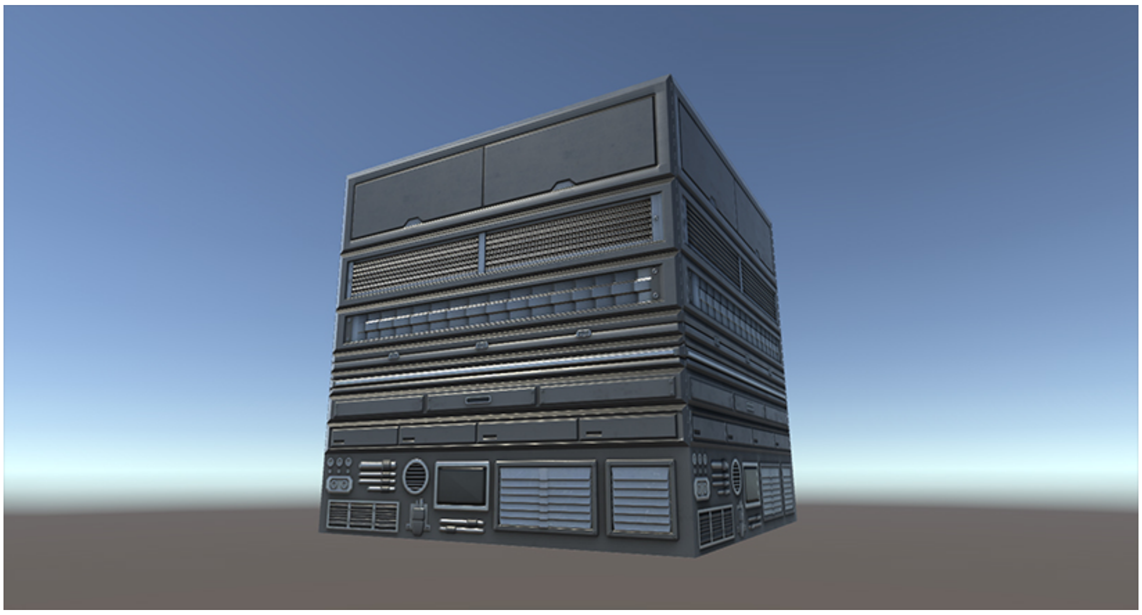

PICTURE 36. The finished trim sheet normal map and ambient occlusion map applied to a primitive cube for inspection in Unity.

The baked normal map was then desaturated to remove distracting colors and used as a base for diffuse textures (picture 37). Tiling textures made earlier were applied to this layout to ensure uniformity. An ambient occlusion map was baked in Maya and added as a multiply layer on top of the diffuse textures, automatically placing shadows in the appropriate locations. Details and embellishments could be added at this point, as well as fixing intended effects that texture baking did not capture artistic intentions, such as the openings of a vent leading into darkness. Care was taken to avoid using details at this stage that are too noticeable when repeated on other instances of the same asset.



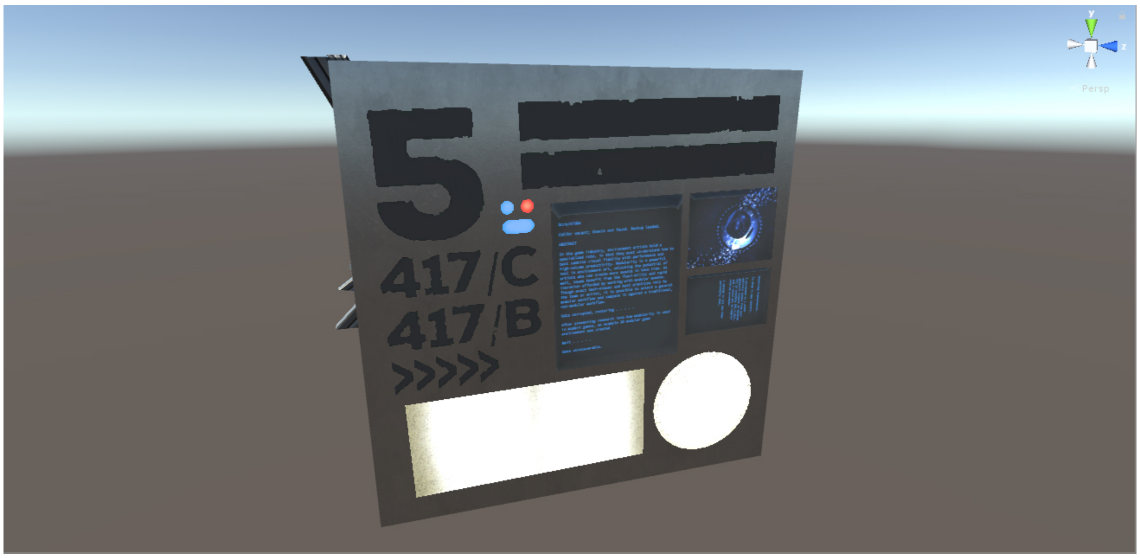
PICTURE 36. The diffuse texture for the trim sheet, made from the three tileable materials as a base. Ambient occlusion was kept separate at this stage to take advantage of Unity’s ability to tune its opacity.

For this project, a specular map was not used, favouring instead a newer technique called physically-based rendering (PBR). The purpose of PBR is essentially the same as a specular map; it informs the shader of how areas of a material react to direct light, whether they are smooth and shiny or rough and nonreflective. PBR, however, is a new industry standard that better simulates the behaviour of light by separating traits into multiple maps, and the result looks more realistic. This did not affect the results of this project in terms of comparing modular and non-modular workflow. The final resulting materials are shown below, again applied to primitives in Unity (picture 37).



PICTURE 37. The finished trim sheet material after applying PBR maps, and normal and ambient occlusion maps made earlier, to a cube primitive.

Finally, a decal sheet was created manually in Photoshop (picture 38). Creating decals for manual placement (not projection) requires that they be stored in a file type that includes alpha transparency. Aside from this condition, creating decals would only require UV mapping a flat plane to the appropriate section of the decal sheet. Additional texture maps were also created to ensure the decals looked natural in Unity, including an emissive map. To keep the project as manageable as possible, the decal map also contained emissive textures for light sources, one round and one flat, or cylindrical.



PICTURE 38. A transparent decal material applied to a plane and set in front of a cube.

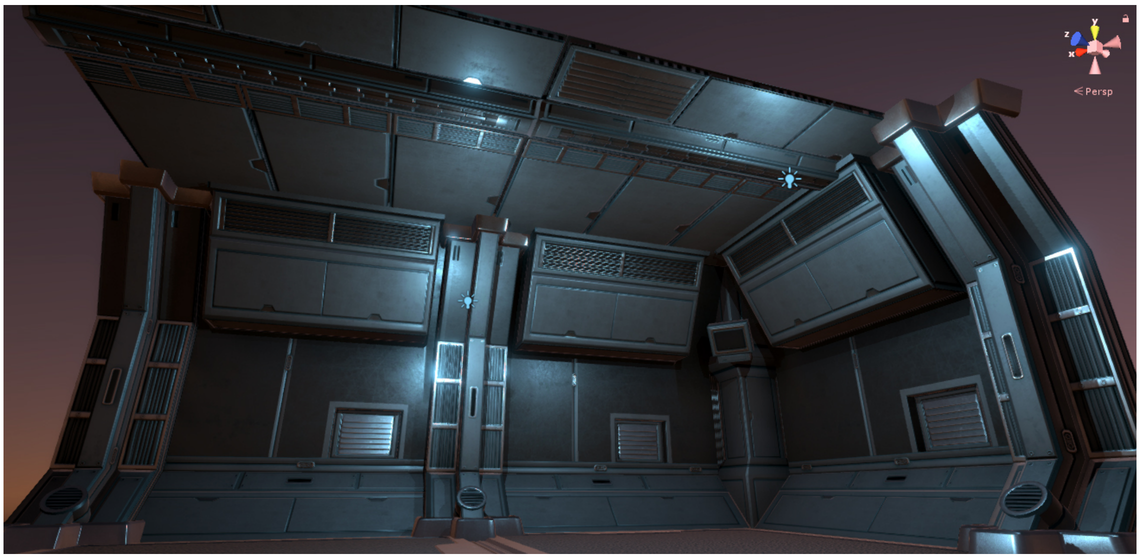
In terms of workflow, it can be said that creating textures this way took less time than it would when working non-modularly. There was no need to account for specific shapes and peculiarities of a given model, and less concern over the appearance of seams, as the location of tiling edges are known. At the same time, uniform 45-degree bevels used on the trim sheet will always match up along neighbouring edges, so there was less concern over the compatibility of specific model edges. In hindsight, this was of extreme benefit, as models could be kept relatively simple, reducing the workload in terms of creating geometry. The bevels effectively doubled the perceived polygon density of every edge, without actually creating and managing it.

Creating textures in this way was both liberating and nerve-wracking. This may have been the result of having never worked this way before. While it was encouraging to see the results of creating materials in-engine this rapidly, it was also difficult to trust that they would be sufficient to map all elements of models that hadn’t yet been created.

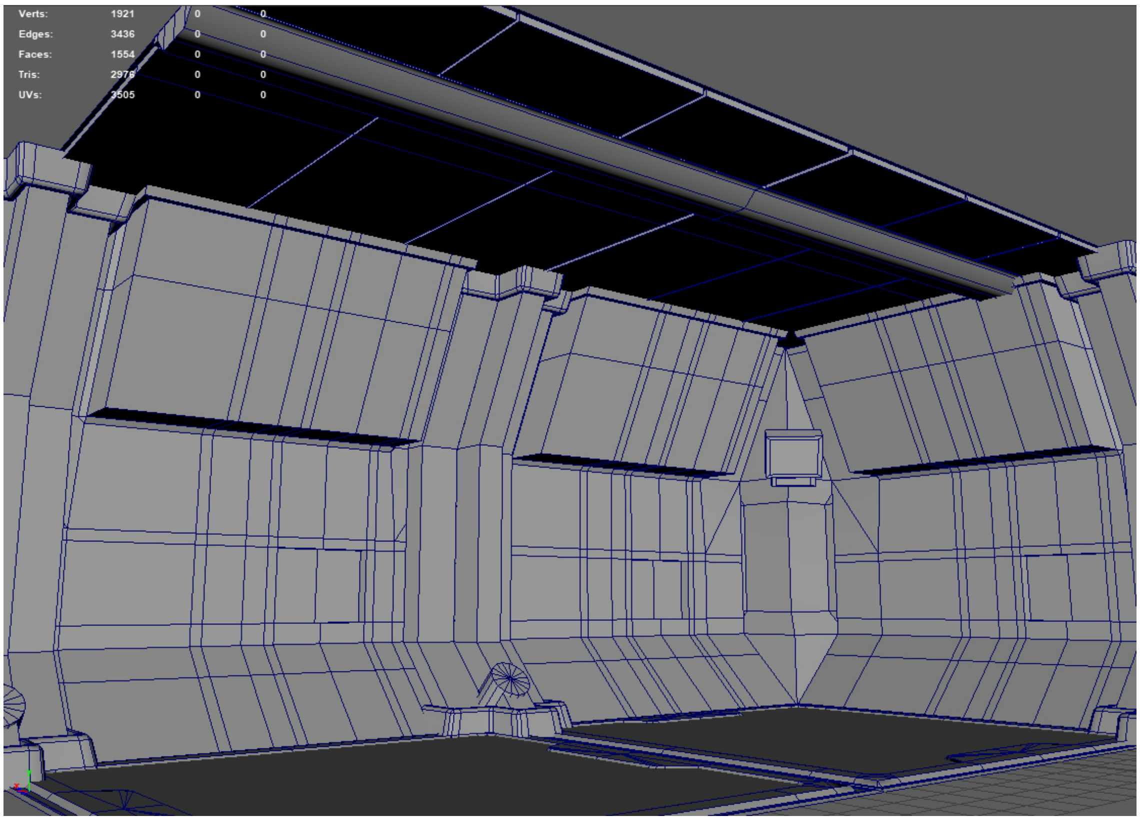
4.4 Modeling the Assets and UV Mapping

At the outset, there was concern over how adaptable one could be when modelling while working with a trim sheet. Assets were created with basic hard-surface modelling techniques in Maya and kept simple as proof of concept, easing early work with the unfamiliar technique. The blockout assets were modelled directly into simple forms for testing and mapped.

It became immediately apparent that there was a psychological benefit to mapping UVs to a finished texture. Keeping these early assets from becoming too complicated allowed free experimentation with the trim sheet, and dismissed concerns about texture stretching or restrictions on the size and shape of polygons. It became increasingly comfortable to use the bevels and details of the trim sheet in unexpected ways. Further development came with the realization that planes could be carefully sectioned to place trim sheet details on the same plane as tiling textures seamlessly. The immediacy of the visual result was encouraging, and the process became fun. The first prototype assets of each type were imported into Unity for viewing in game-engine lighting (picture 39). The polygon density of these assets was low, but the carefully prepared trim sheet proved that no visual quality had been lost (picture 40).



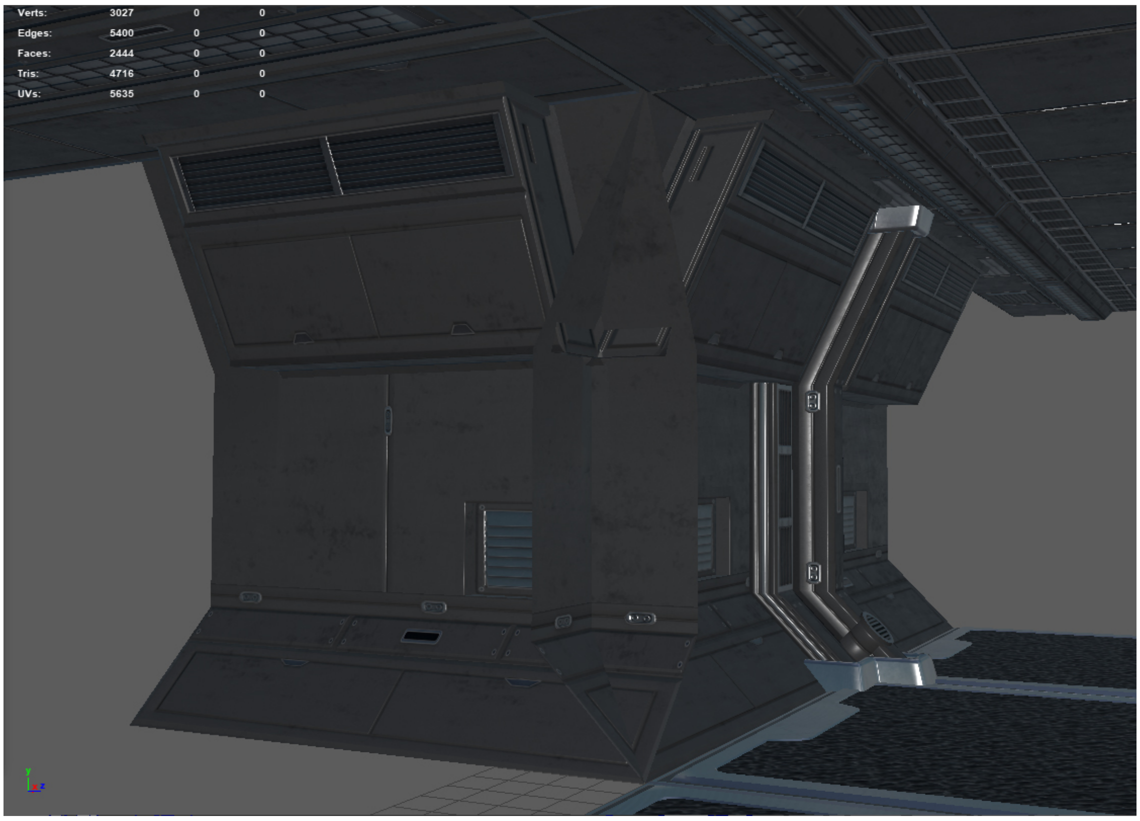
PICTURE 39. Early test in Unity using simple models UV mapped to the trim sheet.



```
Verts:   1921   0   0
Edges:   3436   0   0
Faces:   1554   0   0
Tris:    2976   0   0
UVs:     3505   0   0
```

PICTURE 40. A wireframe view of the above test, revealing the low number of polygons used.

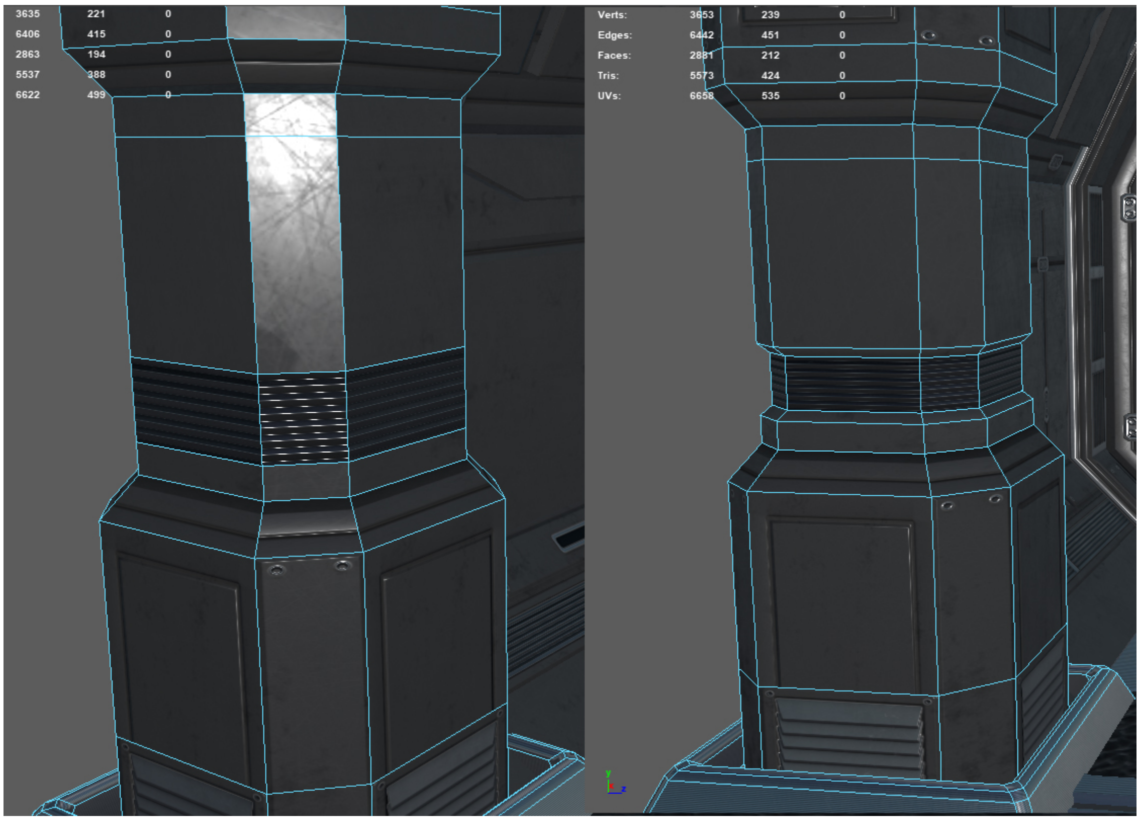
In terms of adaptability modelling strictly according to the grid also benefitted this process. Assembling early assets revealed a need for an additional corner piece that would cover the overlapping geometry of two wall sections (picture 41). Covering geometry is an industry standard technique and creating new or altered wall assets for use on corners would have defeated the purpose of working modularly. Using the grid, a new asset was modelled directly over the extraneous geometry.



```
Verts:   3027   0   0
Edges:   5400   0   0
Faces:   2444   0   0
Tris:    4716   0   0
UVs:     5635   0   0
```

PICTURE 41. Placing two wall sections on an inside corner revealed overlapping geometry that could be modelled over.

After creating this corner asset, another benefit of using a trim sheet was discovered. In this case, the final look of the corner seemed too simplistic. A section of the column had been mapped to an area of the trim sheet that looked like it should be recessed into the geometry, representing some form of supporting structure beneath the outer metal. It was possible to add geometry extremely quickly by creating edge loops on the textured asset and extruding inward (picture 42). The new polygons that were created could be immediately remapped in minimal time, while the intended part of the texture was left undisturbed by the process.

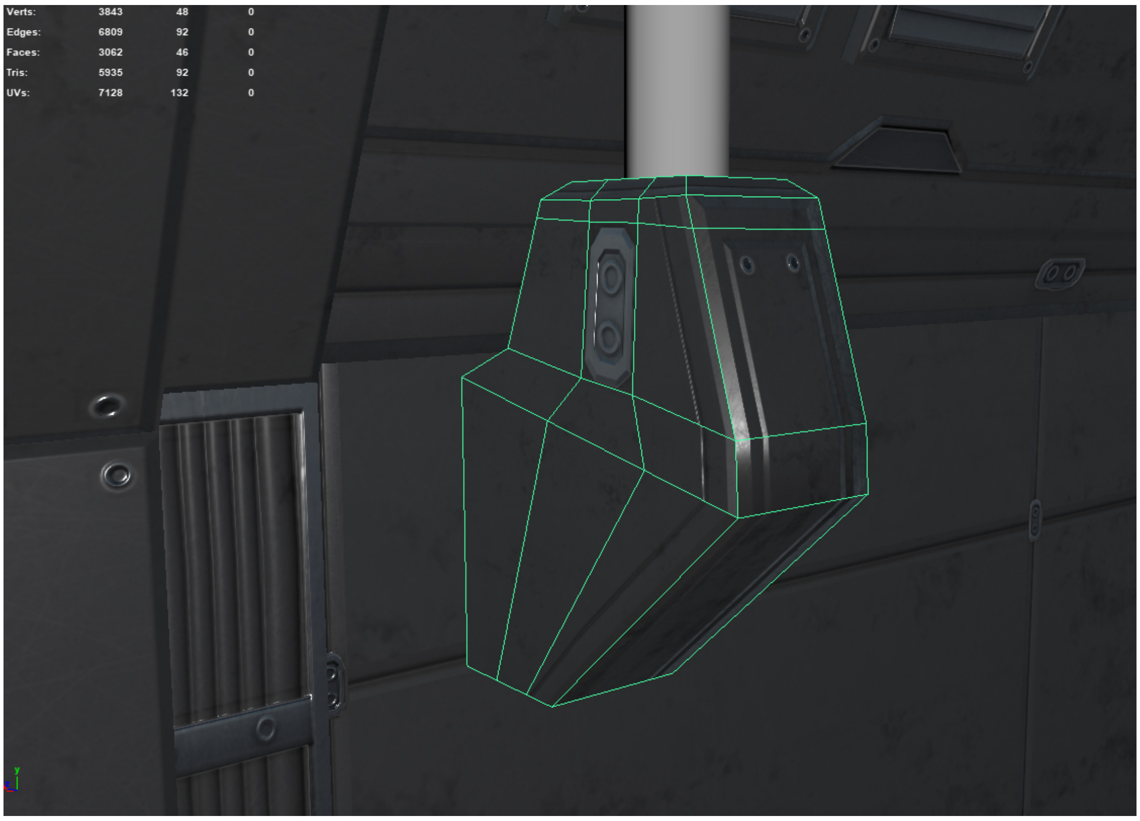


```
3635    221    0
6406    415    0
2863    194    0
5537    388    0
6622    499    0
```

```
Verts:    3653    239    0
Edges:    6442    451    0
Faces:    2881    212    0
Tris:     5573    424    0
UVs:      6658    535    0
```

PICTURE 42. Geometry added with precision to an asset textured with the trim sheet.

As the process continued, models became more experimental. By carefully placing extra edges to creating polygons for trim sheet mapping, it was possible to effectively map even complex, oblong, rounded, or sharply angled shapes (picture 43). To be noted, there is some element of underlying geometry visible when mapping UVs to a trim sheet. Straight, linear edges and bevels on a trim sheet can be mapped to a curving series of polygons, but in some cases may still appear less convincingly round than normal baked from a high-polygon version of the asset. Thorough planning and mixing of the two techniques could eliminate this issue.



```
Verts:    3843    48    0
Edges:    6809    92    0
Faces:    3062    46    0
Tris:     5935    92    0
UVs:      7128   132    0
```

PICTURE 43. UV mapping had to be done with care, but even complicated shapes could be mapped to the straight bevels of the trim sheet.

Modelling decals was straightforward; a plane was created, assigned the correct material, and UV mapped to the correct part of the decal sheet. The plane was then resized to minimize any distortion, matching as closely as possible to the original proportions of the decal.

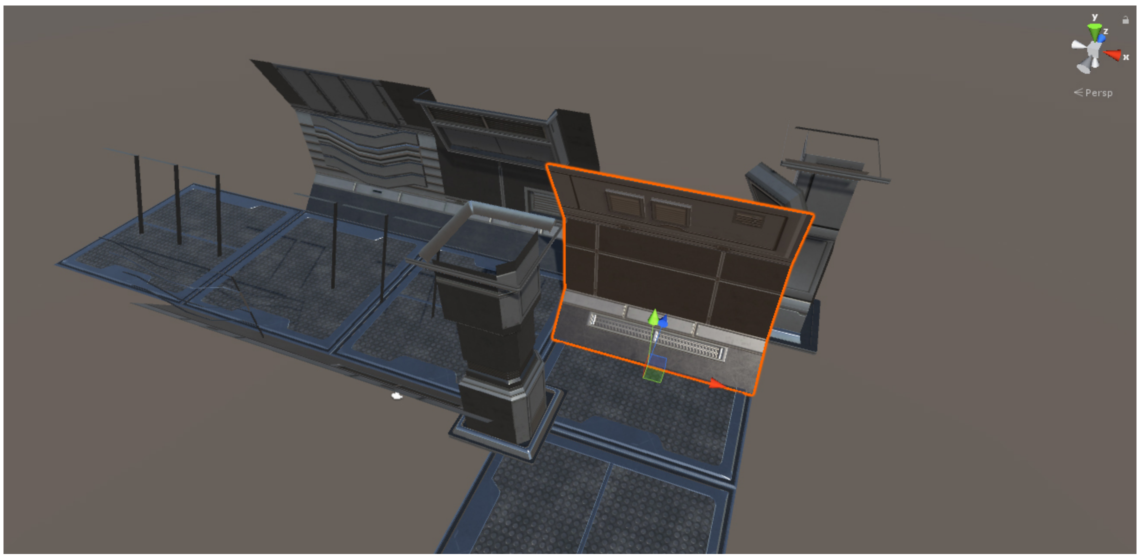
At this stage, it was possible to draw many comparisons between modular and non-modular workflows. Overall, the process was exceptionally fast and enjoyable. Many artists dislike the process of UV mapping, doing it only out of necessity. UV mapping on the trim sheet was like a new medium for the artist, creatively arranging the shapes with immediate visual feedback. It is likely that this alone could be helpful for some artists who find this middle stage of the non-modular process difficult to stay motivated through.

The flexibility of the trim sheet was surprising. In the case of the corner column with added geometry, it was clearly demonstrated. In the traditional process, the model would have been completed, textured baked, and UV mapping finished already. All of those steps would need to be repeated in order to change the underlying geometry. Furthermore, seeing materials applied to models immediately allows the artist to make well-informed decisions concerning where extra geometry was needed. Testing finished assets in the game engine made it possible to infer the viewing angle of the player and the position of lights. From there, it was possible to correct a model's silhouette, or add shadow-casting geometry to enhance the sense of three-dimensionality.

Working on the grid was helpful, but the effects of doing so are more apparent when later assembling the finished kit. No additional time or forethought was necessary for creating assets to fit the grid. Blockout assets could be directly used as a base, and in most cases, duplicates were made for each new variant, ensuring an original, correctly sized copy was always available. In the final stages of the modelling process, all model origin points had to be set to the correct place, and the model itself then moved to coordinate 0,0. Basic features of Maya, and presumably other similar software, made this a simple task.

4.5 Assembly in Unity

Assembling the assets on Unity's grid was predictably easy. At this stage, it was the benefit of working on the grid that all pieces were snapped into position quickly (picture 44). Any error in the placement of origin points would have been immediately apparent. All final materials needed to be created in Unity and applied to the assets as each was imported, but this is also the case in a non-modular workflow. Though the scope of this project does not include any large-scale assembly, it would be comfortable using this kit for a larger environment.

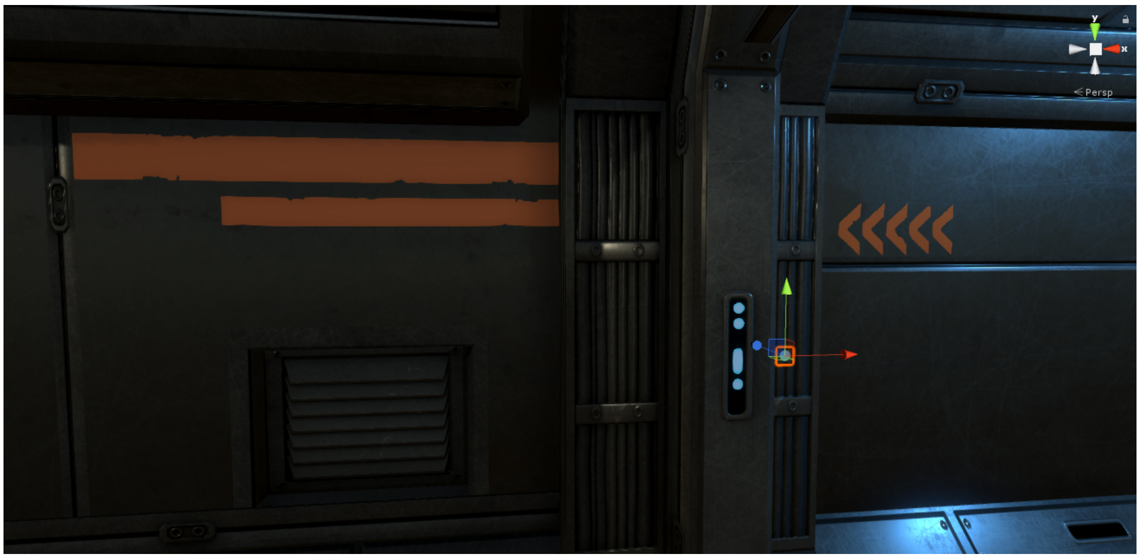


PICTURE 44. Assets snap together on Unity's grid.

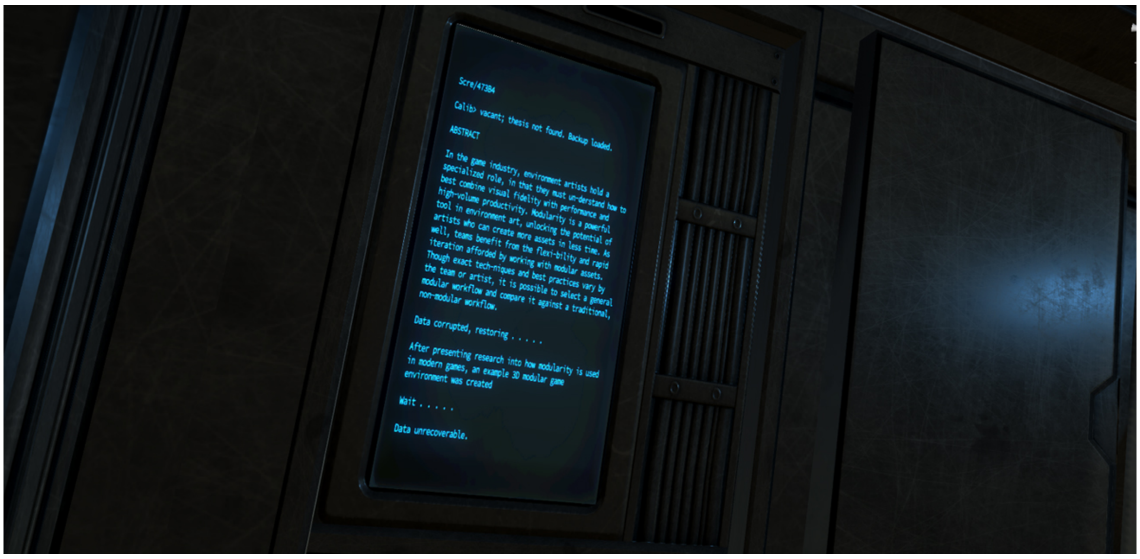
As noted in section 3.3.2, lighting is a complicated subject, and it could be said that choosing how to light this environment was the most difficult for the artist. The time required to find a more aesthetically pleasing lighting scheme would have been at odds with the goals of this project. Baked lighting may have done more to enhance the form and reflectivity of the assets, but frequently requires repeated tuning and re-baking to reach a good result. Furthermore, dark, high contrast lighting would have been appropriate for the environment, but difficult for the reader to view with this text, so a compromise was made.

Decals were a minor source of frustration, and it seems that correctly using decals in a game engine may require tuning to get the best result. In this case, it took time to find the correct shader settings and alter the material textures for a good result. There were other anomalies, for example the decals fade out more readily as they become distant than expected, and they interact strangely with light. Repeated tuning of the shader eventually mitigated some of these issues. Overall, they benefit the scene more than detract from it.

Placing the decals was tedious but not difficult (picture 45). One the decal was aligned to the proper plane and scaled to an appropriate size, it could be duplicated and moved along one axis to reach similarly aligned planed on other wall sections. Emissive decals were placed on monitors in the environment experimentally. While the visual result is passable from some distance, close inspection reveals small artefacts around the edges. The sharpness of detail up close was unaffected (picture 46). Further experimentation might yield more information concerning when a decal should and should not be used. In terms of aesthetics, the monitors may have been better textured via UV mapping. Decals offer the possibility of being easily turned on, off, or changed during gameplay, however.



PICTURE 45. Decals placed along wall sections.



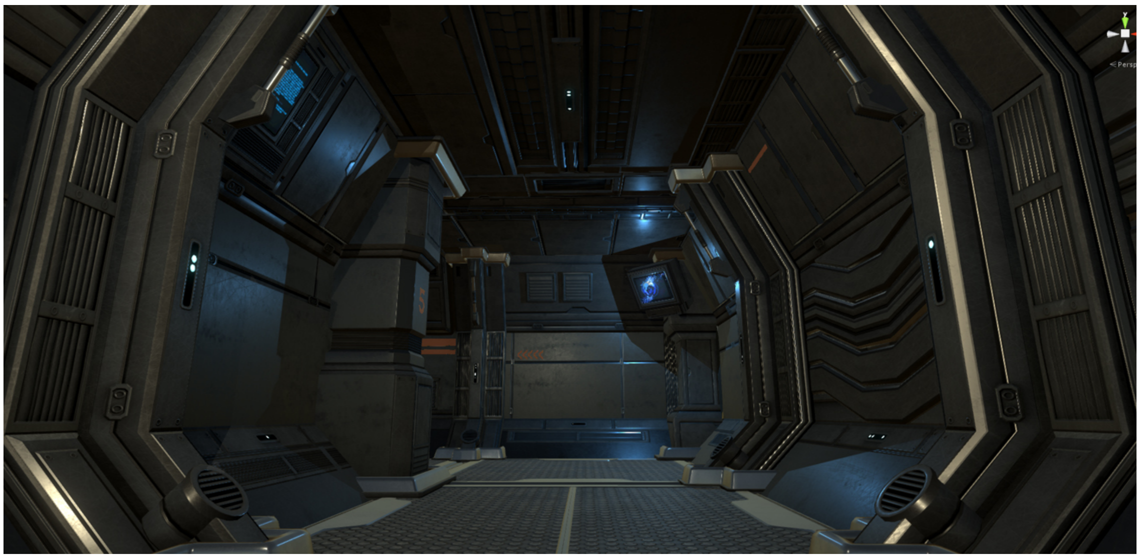
```
Scre/47384

Calib> vacant; thesis not found. Backup loaded.

ABSTRACT

In the game industry, environment artists hold a
specialized role, in that they must un-derstand how to
best combine visual fidelity with performance and
high-volume productivity. Modularity is a powerful
tool in environment art, unlocking the potential of
artists who can create more assets in less time. As
well, teams benefit from the flexi-bility and rapid
iteration afforded by working with modular assets.
Though exact tech-niques and best practices vary by
the team or artist, it is possible to select a general
modular workflow and compare it against a traditional,
non-modular workflow.

Data corrupted, restoring . . . . .

After presenting research into how modularity is used
in modern games, an example 3D modular game
environment was created

Wait . . . . .

Data unrecoverable.
```

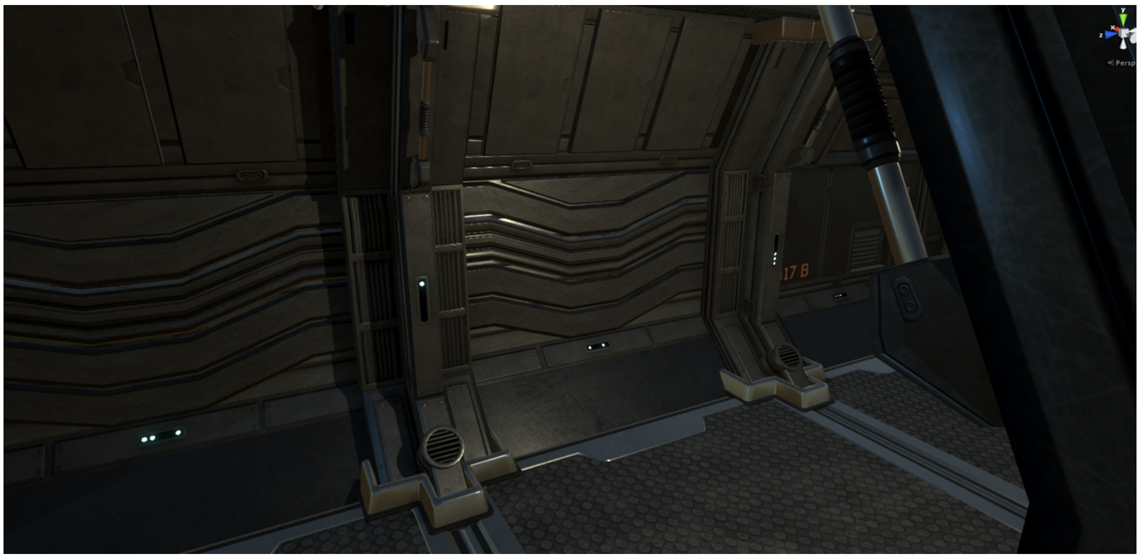
PICTURE 46. A monitor decal at medium distance.

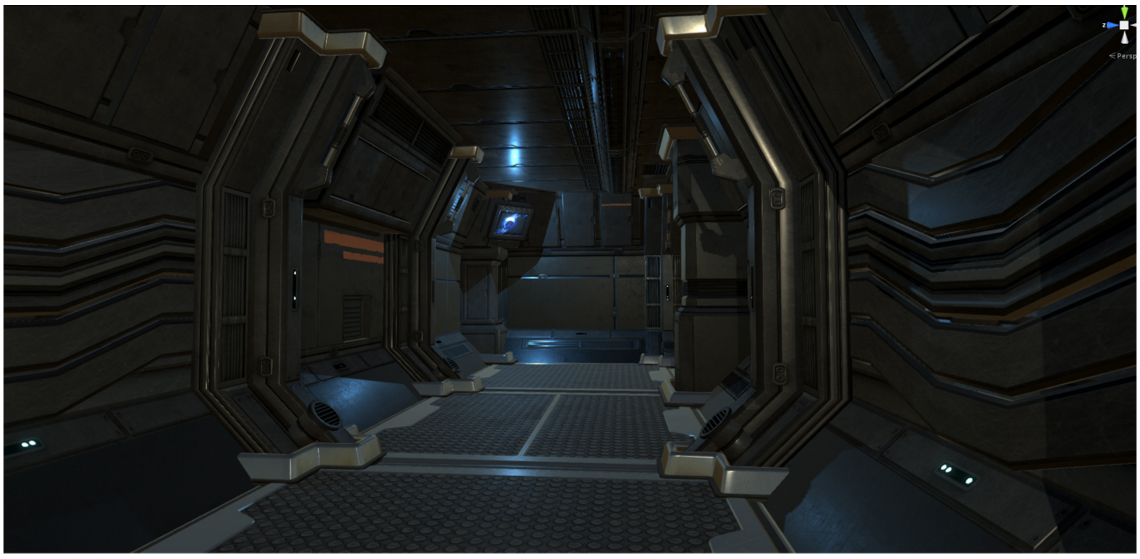
As expected, the small scale of this project, and through the medium of this thesis, made it difficult to differentiate a hero asset in this scene. There is no context by which the reader can judge how often these assets appear in a theoretical wider environment. The intended door stands out from a distance, through a combination of bright lighting, converging lines along the corridor, and strong reflection. A player passing through this corridor over the course of the game would theoretically focus their attention on the door, and not the surrounding walls. Therefore, the door, with the help of the scene, accomplishes the goal of gaining the player’s attention, but does not provide and special context to this corridor. The monitor on the corner of the corridor is arguably as visually interesting due to its colored lighting, but there is again no context for how often this would appear in a theoretical game.

4.6 Final Result

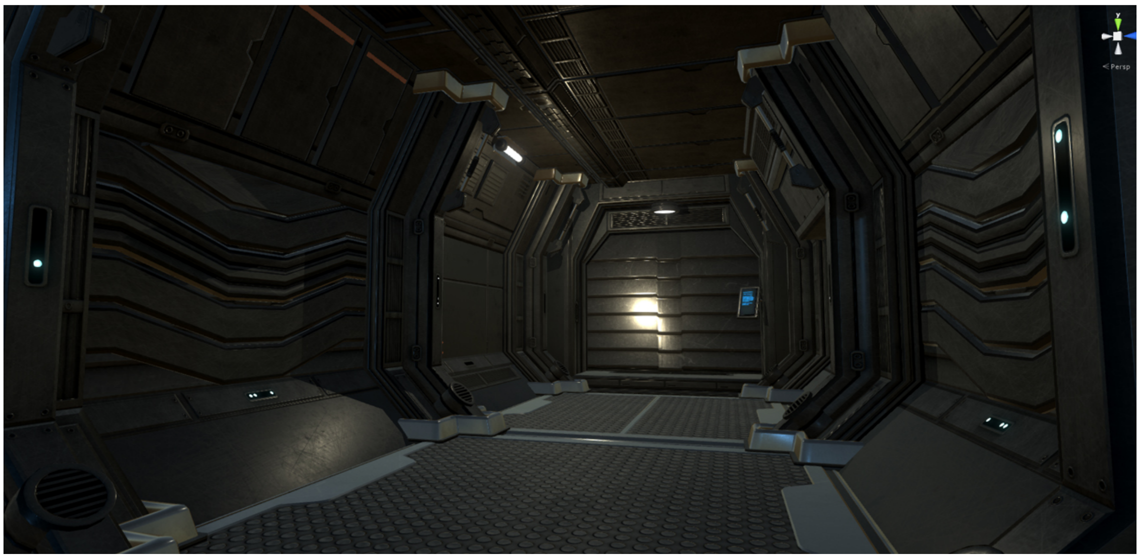
Creating this environment with the modular techniques in this thesis proved to be exceptionally fast. Consequently, it was possible to ensure a good level of visual quality was reached despite setting no goals in terms of aesthetic value. It is still possible to detect repetition in the assets and textures, but not enough to stand out to most players. Below are five screen captures of the finished environment for reader assessment of how well project goals were achieved (pictures 47-51).


PICTURE 47. Screen capture 1.

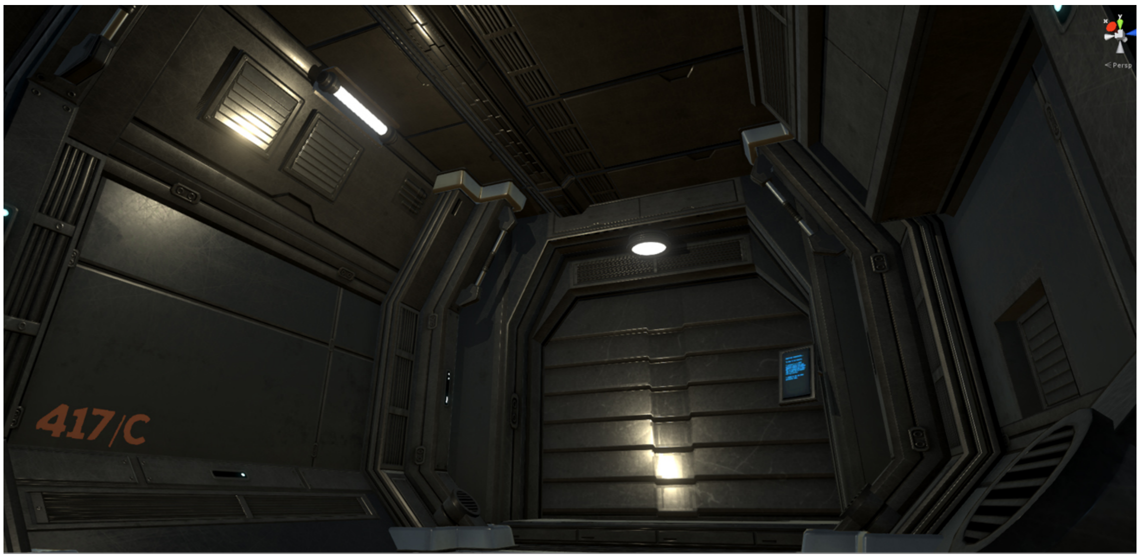

PICTURE 48. Screen capture 2.



PICTURE 49. Screen capture 3.



PICTURE 50. Screen capture 4.



PICTURE 51. Screen capture 5.

5 DISCUSSION

Using a combination of modular environment art techniques, it was possible to create a game environment much more quickly than expected. Using the trim sheet made rapid progress not only possible but encouraging. Working on the grid, while not revolutionary, functioned exactly as expected and revealed no unexpected results. The addition of lighting, focal points, and decals adds life to the scene and provide some illusion that the assets are less repetitive than their true nature. Of the expected benefits of working modularly, flexibility, increased productivity, and interchangeability were demonstrated. Though expected, the degree to which flexibility and productivity were improved was surprising. The overall visual quality of the result was also surprising and owed to a combination of rapid iteration and time savings that could be spent on improving models.

In terms of providing an example of modularity in action, the results of this project were mixed. The desired qualities and benefits of the workflow were seen, but the chosen nature of the environment created may not have provided as clear a picture as possible. Mitigating repetition is a key factor in successfully combatting art fatigue, and a corridor does not provide wide views of many assets at once. More initial research could have been conducted on the proper use of decals in a game engine. It would have allowed more informed decisions concerning where and how to use them.

If the project were to be undertaken again in the future, improvements could be made. Instead of a corridor, a wide-open room would allow ample opportunity to observe and conceal repetition using these techniques. A set time limit may have benefitted the process in terms of measuring the specific benefits of modularity, as time was unavoidably spent on aesthetics at intervals. Further improvement might possibly be gained from working with a single chosen environment concept from a professional concept artist. Working with an existing concept would have better included the planning stages of a modular environment by requiring the artist to break down the concept and achieve the same vision with a kit.

In conclusion, the project successfully demonstrated key benefits expected of using a modular workflow to create a 3D game environment. The workflow benefitted the artist throughout the process by providing early visual feedback, easy iteration, and allowing work to be reused across many assets. Whether in a studio environment or personal project, it is possible for artists, level designers, and others to benefit from adopting a modular workflow.

REFERENCES

Albeluhn, C. 2017. Visually appealing building guide. Read on 4.4.2018. http://www.chrisalbeluhn.com/Building_Layout_Guideline_Tutorial.html

Baigent, A. 2018. Sci-fi Trim & POM Decal Sheet. Read on 12.4.2018. https://www.artstation.com/artwork/RGPWW

Bavoil, L. 2015. Are You Running Out of Video Memory? Detecting Video-Memory Overcommitment Using GPUView. Read on 25.4.2018. https://developer.nvidia.com/content/are-you-running-out-video-memory-detecting-video-memory-overcommitment-using-gpuview

Burges, J. and Purkeypile, N. 2016. Fallout 4s Modular Level Design. Seminar. Game Developers Conference on 14.5.2016. UBM Game Network. https://www.youtube.com/watch?v=QBAM27YbKZg

Burke, S. 2015. The Complete Witcher 3 Graphics and Optimization Guide & Performance Benchmarks. Read on 19.4.2018. https://www.gamersnexus.net/game-bench/1952-complete-witcher-3-graphics-optimization-guide-and-performance

Chollet, M. 2017. Creation of the Zero Gravity Capsule. Read 2.4.2018. https://80.lv/articles/creation-of-the-zero-gravity-capsule/

Clarry, J. 2015. Hard surface modelling and high poly meshes to make UV baking maps. Read on 2.4.2018. http://jordanclarryba5.blogspot.fi/2015/01/hard-surface-modelling-and-high-poly.html

Damjanov, N. 2016. Game artist talks making the first step toward UE4. Read 3.4.2018. https://80.lv/articles/game-artist-talks-making-the-first-step-toward-ue4/

Fontaine, S. Stylized Meshes and Textures for Games. Read on 4.4.2018. https://80.lv/articles/stylized-meshes-and-textures-for-games/

Godbille, A. 2018. UE4 - Snowy Mountain Temple. Read on 12.4.2018. https://www.artstation.com/artwork/1vZe3

Goodwin J. 2017. My Approach to Level Design: Building Experiences. Read on 3.4.2018. http://www.jerryngoodwin.com/process/

Gurney, J. 2010. Color and Light: A Guide for the Realistic Painter. 1st Edition. Kansas City: Andrews McMeel Publishing.

Hajioannou, Y. 2013. Gamedev Glossary: What Is a “Normal Map”?. Read on 22.4.2018. https://gamedevelopment.tutsplus.com/articles/gamedev-glossary-what-is-a-normal-map--gamedev-3893

Holmberg, M. and Klafke, Y. 2015. Building the Temple of Utu: Blizzard Devs on Great Environmental Design. Read on 29/3/2018. https://80.lv/articles/blizzard-designers-on-good-level-design/

Ivanov, I. 2006. Practical texture Atlases. Read on 3.4.2018. https://www.gamasutra.com/view/feature/130940/practical_texture_atlases.php

Pinto, H. 2016. Artist Panel for 3D Environment Artists. Panel discussion. Hollywood, CA on 26.5.2016. Gnomon School. https://www.youtube.com/watch?v=kGm_xhu42tU

Jönsson, A. 2015. Space Station. Read on 25.4.2018. https://www.artstation.com/artwork/JexrD

Klafke, T. 2014. Creating Modular Environments in UDK. Read 25.3.2018. http://www.thiagoklafke.com/modularenvironments.html

Klevestav, P. 2010. Working With Modular Sets. Read on 3.4.2018. http://www.philipk.net/tutorials/modular_sets/modular_sets.html

Lovas, Kristof. 2015. Decal technique from Star Citizen. Read on 12.4.2018. http://polycount.com/discussion/155894/decal-technique-from-star-citizen/p1

Mader, P. 2005. Creating Modular Game Art for Fast Level Design. Read on 2.3.2018. http://www.gamasutra.com/view/feature/2475/creating_modular_game_art_for_fast_.php

New World Encyclopedia. 2017. Cartesia Coordinate System. Read on 1.5.2018. http://www.newworldencyclopedia.org/entry/Cartesian_coordinate_system

Norris, J. 2015. UE4 Modular Building Set Breakdown. Read on 4.4.2018. http://polycount.com/discussion/144838/ue4-modular-building-set-breakdown/p1

NVIDIA Corporation. 2004. SDK White Paper: Improve Batching Using Texture Atlases. Read on 22.4.2018. http://download.nvidia.com/developer/NVTextureSuite/Atlas_Tools/Texture_Atlas_Whitepaper.pdf

Oakes, J. 2017. Sci-Fi Corridor (Unreal Engine 4). Read on 25/4/2018. https://www.artstation.com/artwork/Jn1VA

Olsen, M. 2015. The Ultimate Trim: Texturing techniques of Sunset Overdrive. Presentation. Game Developers Conference on 2.3.2015. UBM Game Network. http://gdcvault.com/play/1022324/The-Ultimate-Trim-Texturing-Techniques

Pepera, P. 2012. The Environment Art of Halo 4. Read 3.4.2018. http://polycount.com/discussion/159954/the-environment-art-of-halo-4/p1

Perry, L. 2002. Modular Level and Component Design. Read on 25.3.2018. https://api.unrealengine.com/udk/Three/rsrc/Three/ModularLevelDesign/ModularLevelDesign.pdf

Pettit, N. 2015. Asset Workflow for Game Art: 3D Modeling. Read on 4.4.2018. http://blog.teamtreehouse.com/asset-workflow-game-art-3d-modeling

Piquet, F. 2011. Modularity and Draw Call. Read on 8.4.2018. http://www.froyok.fr/blog/2011-11-modularity-and-drawcall

Silverman, D. 2013. 3D Primer for Game Developers: An Overview of 3D Modeling in Games. Read on 4/4/2018. https://gamedevelopment.tutsplus.com/articles/3d-primer-for-game-developers-an-overview-of-3d-modeling-in-games--gamedev-5704

Sus, S. 2018. Orcish forge fan art in UE4. Read on 21.4.2018. https://www.artstation.com/artwork/LYWPr

Tan, M. 2012. Understanding User Research: It's Not QA or Marketing!. Read on 6/4/2018. https://www.gamasutra.com/view/feature/168114/understanding_user_research_its_.php

Unity. 2017. Unity Manual 2017.4: Draw Call Batching. Read on 8.4.2018. https://docs.unity3d.com/Manual/DrawCallBatching.html

Vazquez, S. 2018. Steam Reveals the Most Popular PC Specs for 2017. Read on 20.4.2018. http://www.gameinformer.com/b/news/archive/2018/01/06/valve-reveals-the-most-popular-pc-specs-for-december-2017.aspx

Vermosen, M. 2018. Wall Tileable Texture. Read on 28.4.2018. https://mickaelvermosen.artstation.com/projects/58Kqz?album_id=106800

Williams, S. 2018. Low-Poly Sci-Fi Environment WIP 2. Read on 25.4.2018. https://www.artstation.com/artwork/b14Er