Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Games and Graphics

Creation of modular 3D assets for videogames

Bachelor Thesis

Alena Mikushina

Field of study: Computer Games and Graphics
Supervisor: doc. Ing. Jiří Bittner, Ph.D.

August 2020

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Mikushina Alena** Personal ID number: **466881**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Graphics and Interaction**

Study program: **Open Informatics**

Branch of study: **Computer Games and Graphics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Creation of modular 3D assets for videogames

Bachelor's thesis title in Czech:

Tvorba modulárních 3D komponent pro videohry

Guidelines:

Bibliography / sources:

[1] Jason Gregory. Game Engine Architecture (3rd edition). CRC Press, 2018.
[2] Michael Schwarz and Pascal Müller. 2015. Advanced procedural modeling of architecture. ACM Trans. Graph. 34, 4, Article 10, 2015.
[3] McDermott, W. M. 'The Comprehensive PBR Guide by Allegorithmic, vol. 1.", 2015.
[4] Daniel Hanák. Tvorba realsitického prostředí pomocí fotogrametrie. Bakalářská práce, ČVUT FEL 2019.
[5] Tomáš Kraus. Přesný 3D model společných prostor DCGI. Bakalářská práce, ČVUT FEL 2014.
[6] Jana Kejvalová. Procedurální generování 3D modelu dle mapových podkladů. Diplomová práce, ČVUT FEL 2019.

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Jiří Bittner, Ph.D., Department of Computer Graphics and Interaction

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **11.02.2020** Deadline for bachelor thesis submission: **14.08.2020**

Assignment valid until: **30.09.2021**

doc. Ing. Jiří Bittner, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I want to express the deepest gratitude to the supervisor of this project Jiří Bittner, associate professor in the Department of Computer Graphics and Interaction, for providing his insight into the topic of computer graphics, enlightening comments and remarks.

Declaration

I hereby declare I have written this thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis.

Prague August 13, 2020

Abstract

In order to keep up with the ever-increasing player's demand for higher visual fidelity of game environments, artists are continually implementing new modelling techniques and production methods into their workflow. One popular contemporary approach that has emerged is based on the notion of modular design. Although it offers many benefits for production workflow, the particular techniques and skills associated with this concept are still not well defined.

This thesis provides an overview of various modelling techniques, 3D modelling software and thorough discussion of the modular design paradigm applied in computer games.

We have combined procedural modelling techniques with the concept of modular design to create several game-ready assets in Houdini. We then assembled a simple test scene in Unreal Engine in order to gain a more in-depth insight into the advantages and disadvantages of the discussed workflow.

Keywords:
3D Modelling Techniques, 3D Modelling Software, Houdini, Procedural Modelling, Unreal Engine, Modular Game Design

Supervisor:
doc. Ing. Jiří Bittner, Ph.D.
DCGI, FEE, CTU in Prague
Praha 2, Karlovo náměstí 13

Hráči počítačových her mají stále vyšší a vyšší nároky na grafické zpracování herního světa a jeho detaily. Aby jim mohlo být vyhověno, grafici musí neustále upravovat svůj přístup a používané modelovací techniky.

Jeden z moderních a populárních přístupů je založen na modularitě a modulárním designu. Přestože tento přístup má spoustu benefitů, přesný popis technik a znalostí spojených s tímto konceptem není stále pevně definovaný.

Tato práce poskytuje náhled na různé modelovací techniky, software pro 3D modelování a detailní popis modulárního přístupu aplikovaného v aktuálních počítačových hrách.

Kombinací procedurálních modelovacích technik a modulárního designu jsme v programu Houdini připravili několik assetů už pouze čekajících na reálné využití. Dále jsme v Unreal Enginu poskládali testovací scénu a tím získali hlubší přehled o výhodách a nevýhodách použitého přístupu k tvorbě grafiky počítačových her.

Klíčová slova:
Modelovací techniky, Modelovací Software, Houdini, Procedurální Modelování, Unreal Engine, Modulárního Herní Design

Překlad názvu:
Tvorba modulárních 3D komponent pro videohry

Contents

List of Figures

Chapter 1

Introduction

The technological advancements in the field of computer graphics over the past decades have facilitated a significant increase in the visual fidelity achievable withing real-time game environments. As a result, vast, flourishing worlds with areas that cover hundreds of virtual kilometres have practically become an industry standard (Figure 1.1).

Figure 1.1: Game environment art in Horizon: Zero Dawn by Ryan Spinney.

In order to keep up with the ever-increasing player's demand for more realistic details and variety, artists have to continually work on achieving even higher visual standards by implementing new modelling techniques and production methods into their workflow.

One particular approach that has been adopted by the majority of game studios - Bethesda, Ubisoft, EA, and BioWare, to name a few, is based on the modular design paradigm.

While modularity, in general, has been around for many decades in disciplines like science, technology and culture, the particular techniques and skills associated with the modular concepts employed in computer games are still not well defined. As a result, artists who want to adopt this approach into their workflow are often left with very little to guide them.

The main goal for this thesis, therefore, would be to provide valuable insight into the modular workflow, through a thorough investigation of techniques and guidelines provided by the industry professionals followed by the practice-led research into the benefits and limitations of this method.

Additionally, the scope of this project will include an overview of various 3D modelling techniques that currently exist, as well as some of the most popular modelling software, that provide tools required for each of them.

Procedural modelling, in particular, would be the key topic of the practical example, where the synergy of modular design and procedural techniques will be implemented into the game assets production workflow.

Chapter 2

3D Modelling

Over the decades, 3D computer graphics went through many developments: from simple wireframe representations of 3D objects to photo-realistic flourishing worlds of modern games.

2.1 Brief History of 3D Modelling

The first advancement in the history of 3D modelling came with the introduction of Sketchpad, also known as 'Robot Draftsman' by Ivan Sutherland in 1963 [6] (Figure 2.1).

Figure 2.1: (left) Ivan Sutherland using Sketchpad on a Lincoln TX-2 computer in 1963. (right) Utah teapot print on a coffee mug.

As a pioneer in commercially available CAD[1] systems, it established that computers could be used not only for engineering but interactively by designers and artists, thus setting to motion the rapid development of other CAD systems and new technologies.

[1] Computer-Aided Design

In the 1970s at the University of Utah Henri Gouraud and Bui Tuong Phong made several breakthroughs of their own that enhanced visual results with revolutionary shading and rendering techniques [52].

Utah Teapot

"*It is a mathematical model of an ordinary, an inside joke, a 3D equivalent of a 'Hello, World!',*" [55] in other words, a legend within computer graphics community!

A humble Melitta-brand teapot first appeared in 1975 at the University of Utah (Figure 2.1). Its intricate shape (solid, cylindrical, and partially convex) is what made it a standard reference object for testing and comparing rendering algorithms.

Four years later, Wolfgang Straßer and Edwin Catmull idpendently discovered the z-buffer algorithm, followed by Turned Whitted and recursive ray tracing in 1979 [14] (Figure 2.2).

What marked the next decade in the history of computer graphics was the development of new 3D modelling techniques such as blobby models by Jim Blinn and fractals. Additionally to that, the goal for 3D software had finally shifted from the sole rendering of objects and scenes to character animations [14].

By the year 1990, CAD software was widespread and tested to its limits. With the introduction of the first IBM PC and further development in solid 3D modelling, it was finally easy and cheap to access professional programs and hardware for any company, freelancer and hobbyist [52].

Figure 2.2: The 2013 Pixar film Monsters University was the first animated film to use ray tracing for all lighting and shading.

Note, that these are just some of the most prominent events and advancements in 3D computer graphics that helped to introduce numerous new technologies, approaches, methods and means for creating 3D content.

2.2 Modelling Techniques

Due to the rapid development in the field of computer graphics over the past years, a great variety of unique modelling techniques and approaches have emerged. However, many novice artists still tend to think of 3D modelling as a standard poly modelling toolset with subdivisions, extrusions, and insets.

Even though the described workflow does, indeed, exist, it is but one of many advanced and versatile modelling techniques that currently exist.

2.2.1 Polygonal Modelling

Polygonal modelling is the oldest 3D modelling technique that remains one of the most popular approaches employed by many game developers in the industry.

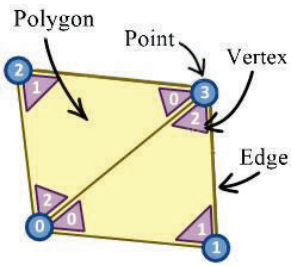


Figure 2.3: Polygon's components in Houdini

Polygons, in essence, are straight-sided shapes comprised of three basic components: vertices (three-dimensional points), edges (lines between two points) and faces (the interior region between them) [4].

When connected, these polygons create a network known as a polygon mesh.

Points & Vertices

Houdini [43] takes the definition of polygons even further and distinguishes between points and vertices (Figure 2.3).

There a point has multiple attributes such as XYZ location, colour, alpha, texture UV, weight, and normal direction.

Vertices, on the other hand, often reference these points by their position in the point list and are only present on polygons, NURBS and primitives. They can also hold diffuse colour, alpha, texture coordinates, and normal information; however, parameters for these attributes would be overridden if they were already defined on points

Box / Subdivision Modelling

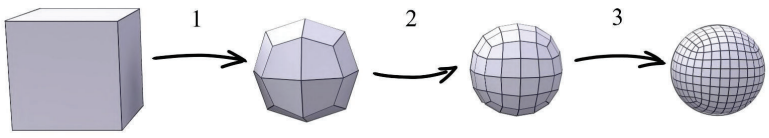


Figure 2.4: A subdivision surface is a polygon mesh where each face has been divided into smaller polygons to give the model a smooth appearance while retaining its general shape.

Box modelling, often combined with subdivision surfaces (Figure 2.4), is the most common form of polygonal modelling technique that often focuses on manipulating whole shapes and large portions of an object at a time.

The production of a model usually starts with a low-polygonal geometric primitive like cube, sphere or cylinder (Figure 2.5).

An artist then gradually refines its shape by controlling individual faces, edges and vertices with traditional modelling tools such as face extrusion, loop cuts and bevel.

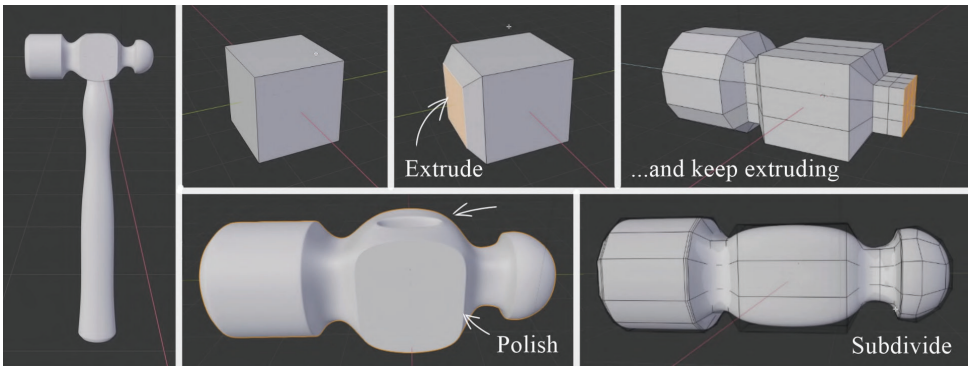


Figure 2.5: Box modelling a hammer

This process of refinement and subdivision repeats until the mesh contains enough polygonal detail to convey the intended concept of the model accurately [46].

Box modelling approach usually works best with hard-surface, or human-made objects and products.

Edge / Contour Modelling

Edge modelling is another polygonal modelling technique, yet, it is fundamentally different from its box modelling counterpart.

In contrast to the previous method, the model is primarily built gradually by placing loops of polygonal faces along prominent contours and then filling any gaps between them with tools like bridge or cap.

However, in order to use this technique to its utmost potential, an artist should plan the model's geometry diligently in advance, which might be especially hard for beginners to achieve.

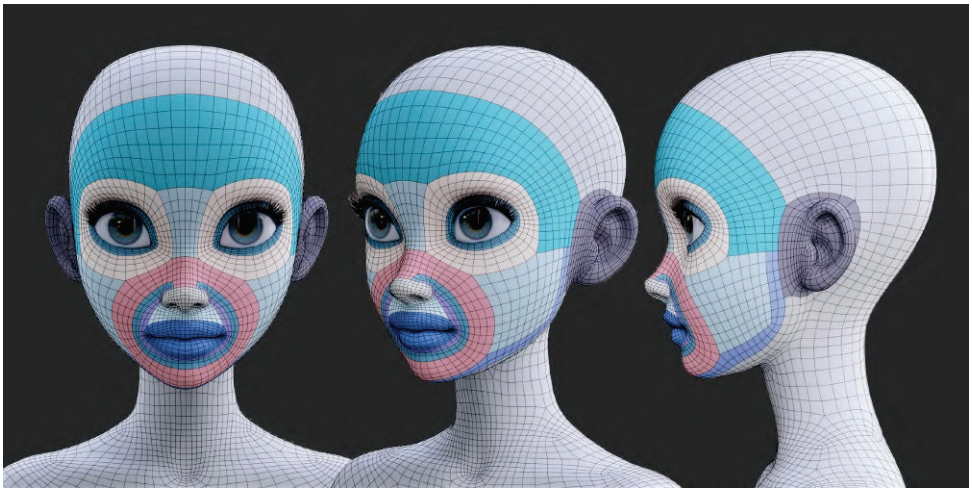


Figure 2.6: Example of an edge flow and topology of a female face by Nazar Noschenko

The precision afforded by edge modelling technique makes it perfect for modelling irregular meshes that require rigorous management of edge flow and topology (e.g., a human face (Figure 2.6)).

2.2.2 NURBS / Spline Modelling

NURBS modelling is an industry standard for designing and modelling surfaces with compound curves (Figure 2.8).

In a sense, it is similar to polygonal modelling techniques; however, instead of operating on standard modelling features such as edges or faces, an artist creates the shape of an object with curves and then forms surfaces around them (Figure 2.7).

NURBS surface objects, therefore, can be described as 3D geometric models that are comprised of patches defined by curves in the U and V directions [2].

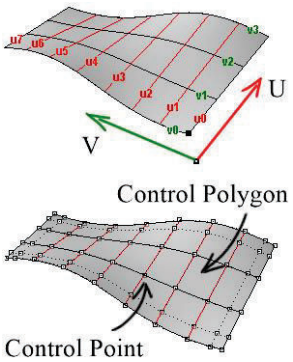


Figure 2.7: NURBS surface

What does NURBS stand for?

"Non-Uniform" refers to the parameterisation of the curve, which allows, among other things, the presence of multi-knots, which are needed to represent Bézier curves.

The term "rational" applies to the underlying mathematical representation. This property allows NURBS to represent exact conics (such as parabolic curves, circles, and ellipses) in addition to free-form curves.

B-splines are piecewise polynomial curves (splines) that have a parametric representation [2].

Due to the smooth and minimal nature of these curves, NURBS surfaces are particularly useful for constructing various types of organic 3D forms.

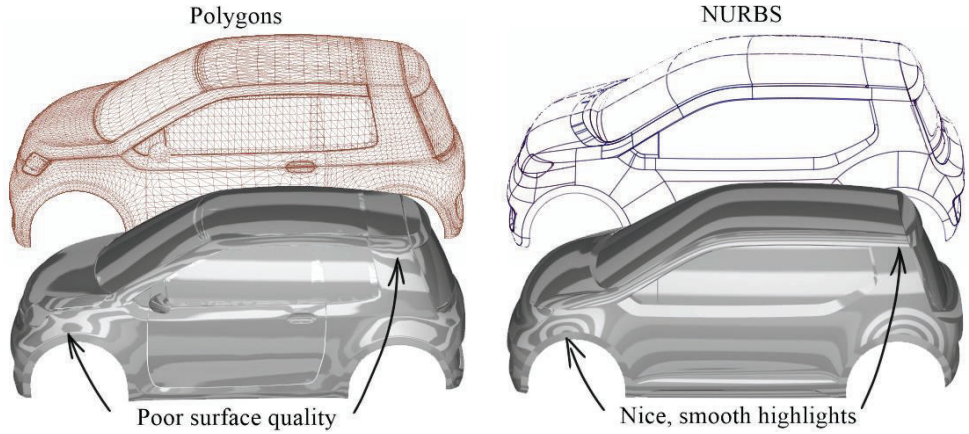


Figure 2.8: Comparison between Polygon mesh and NURBS surface

Additionally, NURBS and spline curves can be used for constructing objects by revolving a profile curve around a central axis [46], or as other elements in the scene (f.e. motion paths for animation or for controlling deformations).

Figure 2.9: Vase, jug and bowl created in Blender with splines

This technique is especially great for radial objects, like vessels and dishes (Figure 2.9), as the artist only needs to be precise with the placement of contours; meanwhile, the software does the rest of the job.

As a result, NURBS are commonplace in the fields of animation, games, scientific visualisation, and industrial design [2].

2.2.3 Sculpting

Digital sculpting is a disruptive technology, a breakthrough in 3D modelling that freed artists from the strict constraints of topology and edge flow.

In comparison to traditional polygon modelling techniques, it implements a more organic approach to asset creation by moulding and shaping the model almost in the exact way as in regular sculpting with clay.

Therefore, artists who have mastered this technique can achieve rather impressive levels of surface detail and natural aesthetics of their models and bring character modelling, in general, to a whole new level [46] (Figure 2.10).

The most rudimentary approach to digital sculpting focuses on the direct manipulation of modelling primitives according to the brush selected, which implies that models geometry have to be very dense from the start.

Figure 2.10: A digital sculpt of Jaemin Kim‘s dragon by Maria Panfilova created in ZBrush

The other way, however, includes more advanced methods such as voxels and adaptive sculpting, which can be further divided into a multi-resolution and dynamic topology.

Multi-Resolution Mesh

Working with multi-resolution topology, in a sense, is very similar to the subdivision surface modelling approach. An artist often starts sculpting on a low-polygonal mesh and increases its resolution level once the limit of detail provided by the model's geometry was reached [7].

Dynamic Topology

The alternative technique to a multi-resolution implements a dynamic subdivision of the mesh into triangles depending on the zoom level or a predetermined absolute detail level, hence the term "dynamic topology" (Figure 2.11).

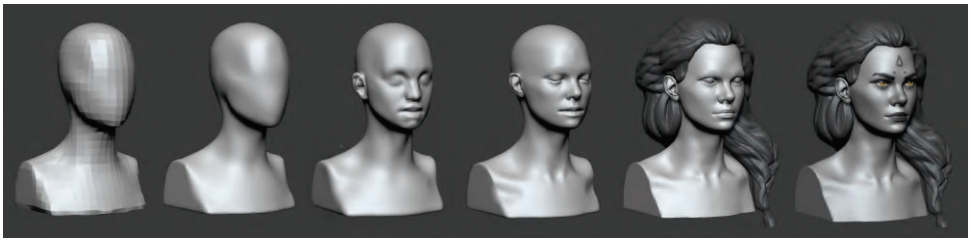


Figure 2.11: Sculpting workflow example in ZBrush by Rodrigo Gonçalves

"*This approach makes it possible to sculpt complex shapes out of a simple mesh, rather than just adding details onto a modelled base mesh.*"[7]

Voxels

While both methods of the adaptive sculpting are closely related to polygonal modelling, with the only difference in the tools applied, sculpting with voxels is an entirely new and significantly different technique.

In the context of digital sculpting, a voxel can be described as a volumetric pixel - a numerical value [0..1] placed in a cubic grid, where the value of 0.5 stands for the object's surface. Similar to pixels, voxels have uniform dimensions along the X, Y and Z-axis, and can be assigned a colour and a shader (Figure 2.12).

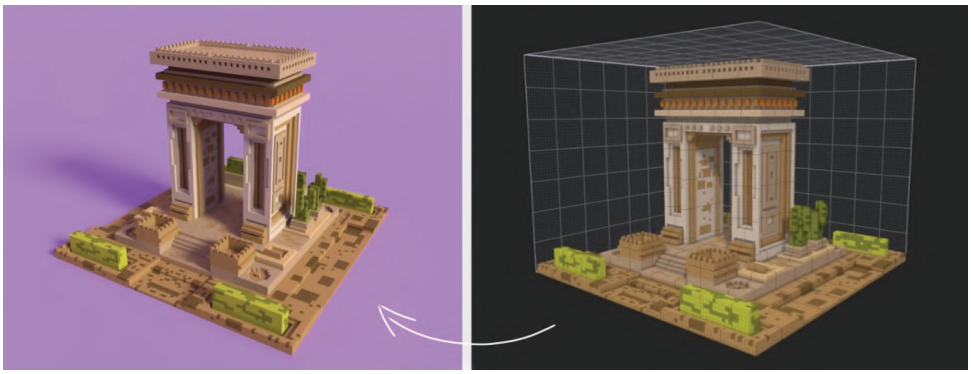


Figure 2.12: Voxel art created in Magical Voxel by narango

Important to note, that only a set number of voxels can occupy the space of the three-dimensional volume at a time.

Sculpting with voxels provides artists with the utmost freedom, allowing them to build up complex objects almost from 'nothing', endlessly add and subtract volume mass, and easily punch holes in the model [27].

2.2.4 Image-Based Modelling / Photogrammetry

Image-Based Modelling (IBM) technique requires the use of algorithms to derive transformable 3D objects from two-dimensional images.

This modelling technique was previously used mostly in situations where time or budget did not allow for manual creation of a fully realised 3D assets [46] (Figure 2.13).

Figure 2.13: Stereophotogrammetry and special effects in The Matrix (1999)

For years many disregarded the technique as being too cumbersome for the technical limitations of game engines, despite the significant rise of application in other fields of the entertainment industry.

It was not until almost a decade later when developers of 'The Vanishing of Ethan Carter' revealed using photogrammetry to create stunning in-game environments [39] (Figure 2.14), followed by EA DICE and their 2015 flagship title 'Star Wars Battlefront'. These and other similar events set to motion the significant popularity growth of the technique in the games industry [48].

Figure 2.14: The Vanishing of Ethan Carter (in-game screenshot)

IBM or Photogrammetry methods can produce high-quality photo-realistic models in a considerably short period, although its disadvantages often cross this benefit.

First is derived from the fact, that in Image-Based Modelling much like in Digital Sculpting, the final mesh often needs to be reworked either by remesh or retopology, which means that the artist might have to fix or create a new UV Map for the model.

Another downside of this method is the need for extensive cleanup work since it is not always possible to photograph the object from all angles or isolate it from its surroundings [42].

2.2.5 3D Scanning

3D scanning represents, perhaps, the most unorthodox approach among all of the discussed modelling techniques so far (Figure 2.15).

Figure 2.15: 3D landscape scanning

Usually, it requires a specific three-dimensional scanner that collects the data about the real-world object (or actor) from multiple angles. A point cloud - raw data of generated points in 3D space - is then used to create an accurate polygon or NURBS mesh [46].

Despite the great results that 3D scanning can provide, the technology for this method is still in development and, therefore, has some serious issues, to name a few: the algorithms are imperfect and prone to bugs, and the hardware is costly and unusable by the majority of companies [51].

However, as a unique modelling technique, it certainly offers a glimpse in the future of 3D visualisation.

2.2.6 Procedural Modelling

Procedural modelling [22], in general, is an umbrella term for various techniques in computer graphics used to create 3D models and textures from sets of rules that are either embedded into the algorithm, configurable by parameters or separate from the evaluation engine.

Fractals

In computer graphics, fractals are geometrically complex objects, the complexity of which arises through the repetition of form over some range of scale [16].

In general terms, these are models with a significant degree of self-similarity, which is achieved through sub-pieces of that model - scaled-down, translated and rotated versions of the original object [15] (see Figure 2.16).

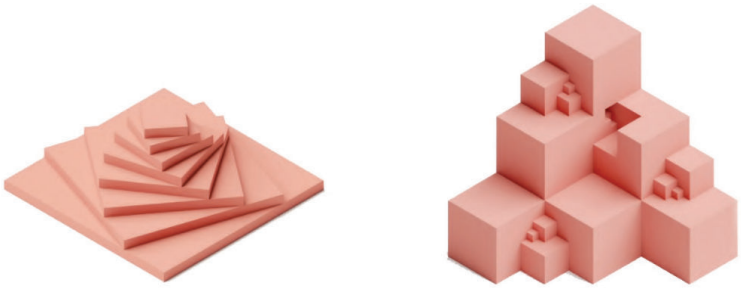


Figure 2.16: Simple fractal geometry created in Blender by David Morris

Fractals can be divided into two categories: non-deterministic and deterministic.

Non-deterministic also called random fractals are most notable for modelling different types of terrain. This process often starts with a simple surface that is divided equally into subparts; then, new vertices are added and pseudo-randomly displaced from the original. The displacement magnitude gradually decreases with each iteration; therefore, towering peaks are formed at the start of the algorithm, and only following subdivisions add fine details [16].

Apart from terrain generation, other applications of fractals in 3D modelling include vegetation, trees, coastlines, mountains, clouds and water surface.

Grammar-Based Modelling

Grammar-Based modelling, known initially as L-systems, uses shape grammars as a method of two and three-dimensional shape generation used for creating remarkably realistic looking 3D models and 2D images of trees (Figure 2.17), plants, and seashells [40].

Figure 2.17: 3D tree model evolution

Shape grammars, essentially, are a set of shape transformation rules that are symbolic and non-deterministic, meaning that they are not limited to a single set of rules from which artist can choose [13].

L-systems

L-systems were first introduced in late 1960 by Aristid Lindenmayer as a mathematical theory of plant development.

Over the years several subsequent geometric interpretations of this theory have emerged, which helped to turn L-systems into a versatile tool for modelling various types of foliage and trees [40]. Besides, modern L-systems can also be used for generating streets and buildings [23].

The grammar-based modelling method often consists of two parts: a generative and an interpretative process (Figure 2.18).

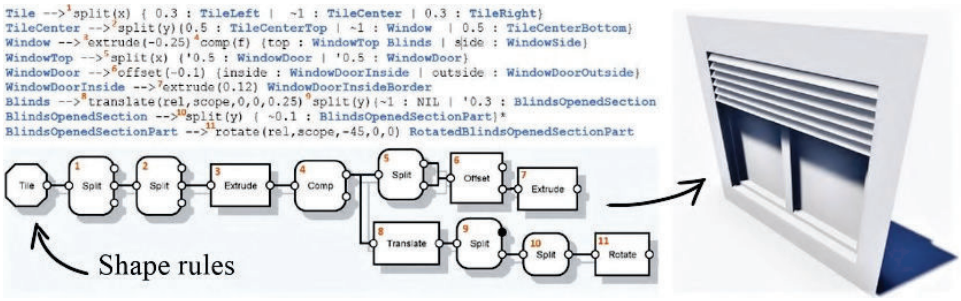


Figure 2.18: Shape rules as text and graph used for generating a window model

The fundamental concept behind the generative part is string-rewriting - a technique in which letters that comprise an initial string are replaced in parallel by new letters according to pre-defined rules. Combined these new letters form the next generation subject to repeated use of the same replacement rules over and over again.

The most familiar example of rewriting technique is the snowflake curve proposed by von Koch in 1950 [40].

As the name suggests, the second step of the process interprets the letters of the last generation of string to create the final geometry [23].

Moreover, the presence of self-similarities in shapes created by L-systems suggests their close relation to deterministic fractals [15].

OL-systems are an example of the more sophisticated shape generation method. It originates from the traditional L-systems, yet, includes context-sensitivity, wordage information, and probabilistic rule evaluation, which allows each plant created to be even more unique [15] (Figure 2.19).

Despite the time-consuming preparation of the rules, the model production process with shape grammars, in general, is less labour-intensive compared to more traditional techniques, such as polygonal or NURBS modelling.

Figure 2.19: The Plantation by Alex Patel

Implicit Surfaces

In contrast to the explicit representation of objects' surfaces, which is commonly used in the majority of discussed modelling techniques, implicit surfaces, as their name suggests, are represented by implicit equations of form

$$F(x, y, z) - I = 0,\ I \in \mathbb{R}$$

Here, the function **F** is the scalar field function, that determines the value at every point in space due to some underlying primitive geometric objects (e.g., points, line segments, and polygonally bounded planes) [35]. Different surface shapes, depending on the function **F**, can be achieved by merely changing the value of I; hence the term "iso-surface".

Tracking the Origins

Jim Blinn was first to introduce implicit surfaces, or blobby molecules as they were called at the time, into computer graphics when he used them to visualise electron density field [35].

Shortly after, metaballs were developed in Japan at Osaka University, followed by soft objects introduced in Canada [8].

Later, Bloomenthal discussed the application of these techniques for modelling organic forms, such as trees, leaves and arms, and for polygonising them using adaptive surface-tracking octrees [8].

Currently, there are three major approaches to implicit surface modelling: simple, skeletal and complex primitives.

The first technique uses primitives defined around a single point to create

models (e.g. a sphere or an ellipsoid). These are blobby molecules, metaballs (Figure 2.20) and soft objects.

The second, skeletal primitives, define primitives around rigid skeletons, such as lines, curves, polygons or points.

The latter approach is comprised of algebraic (e.g. a sphere, a quadric surface, a torus or sine wave), procedural and empirical primitives. Procedural primitives can also include conditions or loops, and empirical primitives are generally gathered data, for example, from medical scans.

Figure 2.20: Cosmic Fox by Rino Dal Farra created using metaballs in Blender.

One of the most prominent features of implicit surfaces is that they offer a rather compact representation of a model, which is especially valuable when working with complex surfaces.

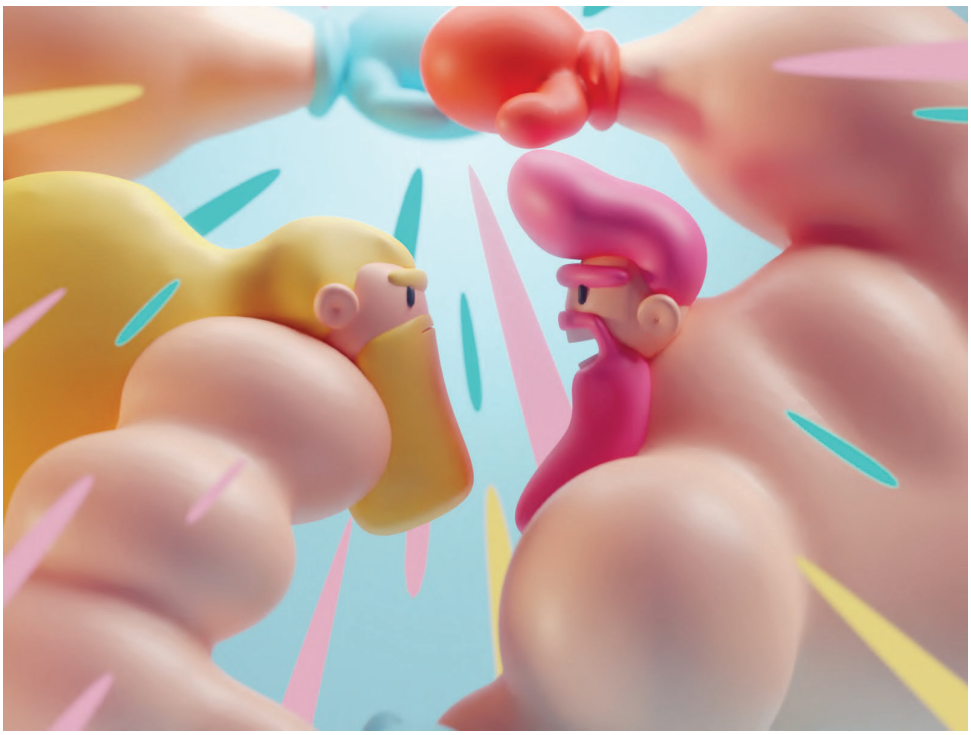


Figure 2.21: Fighters by Mohamed Chahin with subdivided objects and metaballs created in Blender

Implicit surfaces are widely used in the fields of computer simulation and physically-based animation [35] and can even be applied to character modelling (Figure 2.21) and animation [15].

Additionally, as metaball technology proved to be the most successful over the past years, it is currently used in several 3D modelling software, like 3ds Max and Houdini.

2.2.7 Other Modelling Techniques

Figure 2.22: Paint Splashes: Particles, Smoke Sim and Metaballs in Blender by Gleb Alexandrov

Volumetric Modelling

Volumetric modelling is another type of procedural technique that uses algorithms to define and animate three-dimensional volumetric objects such as fire, smoke, clouds, fog, and water.

The most significant advantage of this method is that the small amount of input data is needed to create complex volumetric phenomena [15].

Simulations

Simulations, in general, tend to lean more towards VFX[2], than modelling; however, it is safe to consider them as a 3D modelling technique since they can help artists to create or deform objects.

There are many kinds of digital simulations, each with a unique purpose. Some of the most popular include cloth, particles, soft body, fluid and smoke simulation. Often the result of these techniques can be used as a stand-alone animation, scene or an object [42] (Figure 2.22).

[2]visual effects

Particles

Similarly to volumetric models, particle systems are most commonly used to represent natural phenomena, such as clouds, snow, fire, grass, and trees.

A particle system object is defined by a large collection of simple geometric particles that change stochastically over time. However, all parameters assigned to them (location, birth, death) are controlled algorithmically, which allows the fast and efficient control over the object [15] (Figure 2.22).

2.3 3D Modelling Software

Apart from a great variety of different modelling techniques, an increasingly rich array of software for creating 3D content is now available.

When previously, only large, multi-purpose 3D software packages such as Autodesk's 3ds Max or Maya dominated the market; today, a new crop of stand-alone 3D products have arrived [5]. These products are prominent for their task-focused workflow and technology specifically optimised for a particular job.

Autodesk Maya

In the field of 3D entertainment industries, Maya is most renowned for its comprehensive tools for character design, animation and awe-inspiring effects.

Many big AAA game development studios rely on Maya as an essential part of their asset production pipeline [24].

Its vast feature set includes 3D modelling with different types of objects (polygons, NURBS, and subdivision surfaces), particles, hair, solid body physics, cloth, fluid simulations, lighting, shading, rendering and, most importantly, advanced character rigging and animation. The latest release of Maya also includes Bifrost procedural effects that use graphs and dynamic solvers to create stunning effects; Arnold RenderView system for photo-realistic renders; remesh and retopology tools.

Following the trend of 'best-of-breed' approach in the development of 3D modelling software, Autodesk is now focused more on improving existing systems and implementing features voted on by the community, rather than trying to bring more versatile tools to the 3D suite. This new approach gradually transforms Maya into a much better, all-round application which continues to evolve and improve in a more focused way.

Autodesk 3ds Max

Probably the oldest all-purpose 3D modelling software currently on the market, that predates almost every other program [19].

Over the years 3ds Max has been employed by many notable game studios including Ubisoft, Eidos-Montreal and Activision [24].

Like Maya, it boasts a very robust toolset for modelling, fluid and cloth simulations, hair, fur, VFX, skinning, rigging, animations and a vast range of rendering systems, including Renderman by Pixar, mental ray and Arnold. Another prominent feature of 3ds Max is the spline system that is regularly regarded as the best of its kind [30].

3ds Max is equipped with tools that support both direct and procedural modelling approaches [25], and work with different types of models, such as conventional polygon construction, NURBS and patch surfaces [37].

It further provides its users with a wide variety of plugins to make up for any shortcomings it might have. Even though these third-party modules can often significantly enhance the modelling process, there is always a fear that some of them might cause crashes and unstable system behaviour instead.

Autodesk Mudbox

Mudbox is, yet, another powerful software by Autodesk, equipped with a simple, intuitive and tactile toolset for editing and sculpting [19].

In addition to that, its feature set includes dynamic tessellation, map baking, retopology and texture painting with a layer-base workflow [21].

Modo by Foundry

Since its release in 2004, Modo has rapidly grown from a basic subdivision surface modeller to the fully-featured digital content creation program [25]. It is considered to be one of the fastest modelling software in the industry, with a highly customisable and user-friendly interface.

Modo's 3D modelling toolset supports both direct and procedural modelling techniques enhanced with the award-winning MeshFusion Boolean and built-in sculpting tools.

Among its other most prominent feature are tools like photo-realistic viewport, progressive renderer, WYSIWYG look-dev for Unity and Unreal, and Tool Pipe for creation custom modelling tools through the combination of already

existent.

Modo is currently used primarily by small indie studios like Wooga [20] - a mobile game company.

Cinema 4D by Maxon

Maxon's Cinema4D is another serious competitor to most 3D modelling programs [19], best known for its overall stability, intuitive design and one of the easiest learning curve compared to other CG apps [25].

Even though it is primarily designed and used for creating perfect motion graphics, visual effects and illustration, it can still satisfy the needs of game artists.

Cinema 4D provides the majority of features prevalent among standard 3D modelling programs, such as UV unwrapping, texturing, rigging, animation, and rendering.

In addition to that, it includes a toolset for parametric and volumetric modelling [25], as well as a multichannel texture painting tool - BodyPaint 3D, which makes Cinema 4D the industry-standard for texturing [47].

LightWave by NewTek

LightWave was once an industry-leading modelling, animation, and rendering package frequently used for visual effects in commercial advertising, television, and film. Today, however, it is more prevalent among freelance artists and smaller studios [47].

Perhaps, the most distinguishing characteristic of this program, that developers in NewTek have not changed over the years of its existence is that LightWave still operates as two apps, Modeler – for building assets – and Layout - for texturing, lighting, animation and rendering [25].

Since the first updated version of the 3D modelling package was finally introduced at the start of 2019 after the app lay fallow for several years, new features such as a built-in Bullet, Hypervoxels, and ParticleFX were added to its toolset [47].

However, in a sense, it is still playing catch-up with most of its competitors and is yet to gain back its popularity in the visual and entertainment industry.

ZBrush by Pixologic

ZBrush is a standalone sculpting and modelling app, that is popular across a vast range of industries, especially so in film and game developer sectors [37].

It provides the most robust and comprehensive set of award-winning brush system, and have virtually no competitors when it comes to digital sculpting.

Zbrush is capable of manipulating models with a polycount of 20,000,000 — making it ideal for detailed, high-poly work [30], best suited to the creation of organic form. However, in recent updates, its hard-surface abilities have been gradually improved as well.

While most 3D sculpting software usually requires an already existing object to work on or lets artist start entirely from scratch, ZBrush features a particularly exciting tool called ZSpheres. It allows artists to create a sketch of the object which then well be skinned, transformed to a mesh and ready to be sculpted on [21].

Another artist-friendly feature is a non-linear production path that makes it possible to revert to a previous iteration, make changes to design, and then roll forward again [37]

The only major downfall of ZBrush is in its workflow and user interface; both are not intuitive and hard to master, especially for a beginner [25].

Rhinoceros 3D

Rhinoceros was the first 3D modelling system to integrate NURBS in its modelling workflow.

Its current version can sculpt objects, adapt LIDAR scans, work perfectly with meshes from other systems and even render scenes and complex animations using raytracing, as well as other tools and features one would expect from the 3D modelling software [19].

Furthermore, Rhinoceros is equipped with Grasshopper - a tool for making form generation algorithm without any extra coding needed.

Note

Rhinoceros focuses primarily on producing mathematically precise representations of curves and freeform surfaces, which makes it more suitable for prototyping mechanical parts, creating concept designs or models for 3D printing, rather than for modelling asset for games [37].

Blender

Blender is the free and open-source 3D creation suite that has been around since 2002 [19].

Over the years it has built the large community with more than 500 000 downloads per month, an army of artists, teachers, and enthusiasts behind its continued development [30]. As a result there is not a single aspect of graphics pipeline that Blender cannot incorporate [47].

Indeed, while the majority of other programs are focused on modelling, animation, rendering, or other parts of the 3D process at a time, Blender provides the entire pipeline in one package [37].

It supports modelling, sculpting, rigging, animation, simulation, rendering, compositing and motion tracking, video editing and even 2D animation pipeline.

With its most recent update, Blender 2.8 got a more consistent and polished user interface, high-quality viewport and real-time interactive rendering powered by EEVEE [25].

Even though Blender is not a part of the AAA game industry, yet, it is gradually gaining more popularity among indie game developers, small and medium studios.

In conclusion, Blender is, perhaps, the best option for new-coming artists and game developers, since it provides the most intuitive toolset which can be an utterly viable alternative to the majority of paid modelling programs [25].

Built-in Game Engine

Previous versions of Blender were also integrated with a game engine. However, its development has been discontinued recently and removed from the official release of Blender 2.8.

Houdini by SideFx

Houdini is, yet, another industry-standard software that ranks among some of the best 3D modelling packages like Autodesk Maya, 3ds Max and ZBrush [30].

Among its many features is 3D modelling, animation, effects, simulation, rendering and compositing. It also supports different types of geometry, such as polygons, NURBS and Bézier, various geometric primitives and metaballs. However, what Houdini really stands out for is its procedural approach, which makes it the only major 3D suite designed around a wholly procedural

development environment [47].

Even though Houdini is equipped with tools for more traditional direct manipulation of the model on screen, its fundamental workflow involves artists to work with networks of connected nodes to accomplish their task.

Developers in SideFx compare these networks to a computer file system, where networks represent folders and nodes are like files in them. Each node can be further described as a single procedural instruction, or a subnet, which would holds inside a whole other network.

Example

In contrast to other 3D modelling software, Houdini has a two-level scene design.

The first level `/geo` contains objects such as characters, props, and lights. The second is the geometry object itself - a subnetwork that contains various geometry nodes that define the object's geometry.

Each node has various parameters, inputs and outputs.

Wires between nodes in the network have a different meaning depending on the network type. Usually, they represent that the data created or processed by one node is passed to the next node. However, at the scene level, wiring objects establishes parent relations [44].

The changes done to previous nodes in the network automatically propagate through it to alter the final result, meaning that its parts can be reused, reconfigured or swapped without fear of losing any work already done. When finished, these networks can be packaged up to create new tools with their own interface, called digital assets, which can be later used in other scenes or projects [44].

This node-based procedural approach provides artists with an unprecedented level of control, power, and flexibility [25].

Another notable feature of Houdini is its ability to manage relatively large and complex scenes, with support for generating and loading geometry, and adding detail at the render time instead of keeping everything in memory [44].

Due to its procedural approach, Houdini requires a rather steep learning curve[3], although, once mastered, it can produce solutions that cannot be achieved in any other 3D software [47].

All in all, Houdini is a solid choice for any VFX artist and CG content developer.

[3] meaning that it takes longer for an artist to become productive compared to almost any other 3D modelling software

According to the most recent game reel [45] published by SideFx, Houdini has been employed by game studios of all sizes, from Indie (Figure 2.23) to AAA, such as EA Bioware, Ubisoft, Insomniac Games, and Sega.

Figure 2.23: Everwild is an upcoming adventure video game by Rare, which demonstrates that even small procedural modelling gradually gains popularity even among small indie developers.

Houdini Apprentice & Houdini Engine Education

Houdini Apprentice is a free version of Houdini FX, which can be used by students, artists and hobbyists to create personal non-commercial projects. Even though it offers all the same power and flexibility of the original program, there are some significant downfalls.

The scene file format is unique to Apprentice version and therefore cannot be used in other software (e.g., export to 3ds Max for further refinement or use as a digital asset in a game engine), which implies that only Houdini Engine Education can serve as a bridge between Houdini and other programs.

Chapter 3

Modular Level Design

Over the decades, video game design and development has evolved into the profitable and lucrative creative field that operates within ever-increasing technical capabilities, which in turn facilitate the rapid growth of game environments' visual fidelity (Figure 3.1). Although, despite the technological advancements, the amount of detail a game can have remains restricted by two main factors: the time required on assets creation, and hardware limitations [18].

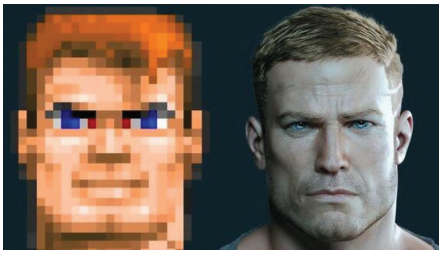


Figure 3.1: Wolfenstein: 1992 vs 2017

These constraints make art one of the most expensive commodities in video games, and challenge artists and level designers to always look for a trade-off between what is visually appealing and how much load it puts on the hardware. Moreover, the ever-growing expectations the current generation of players set for games' graphics only exacerbates this dilemma (Figure 3.2).

Figure 3.2: Horizon Zero Dawn: Mountain Landscapes by Lucas Bolt

Falling back behind these demands is a dangerous prospect for any game studio, no matter the sizes. Therefore, artists are persistently improving their workflow and searching for new ways to maximise system resource allocation, aesthetic quality and make the more efficient use of the production time.

These factors have eventually led to the introduction of the modular design paradigm into computer games.

3.1 The Concept of Modular Design

The term "modular" itself comes from the word "module", which means"*one of a set of separate parts that, when combined, form a complete whole [32].*"

The concept of modularity in design fits this description perfectly, as in its core is the idea of breaking complex systems into smaller components - modules - that can be then brought together to more efficiently organise intricate designs and processes [50].

Throughout the years, the modular design approach has been applied in several fields of science, technology, industry, and culture, each with its nuances (Figure 3.3).

Figure 3.3: Construction of Carmel Place - the first micro-unit modular building in NY.

3.2 Modularity in Games

"Tile graphics" are considered to be the earliest example of the modular concepts applied in computer games that helped game developers to enhance

players' in-game experience regardless the limited hardware and rendering capabilities of their machines. The underlying principle of the technique they employed was as follows:"*tiles of a set size were used and reused, and different modular assets were swapped in and out to create the illusion of a much bigger world than one that actually existed [41].*"

In games, like SimCity (Figure 3.4), Mario and Sonic, these tiles were used to create entire levels, including buildings, landscape/terrain, and even characters.

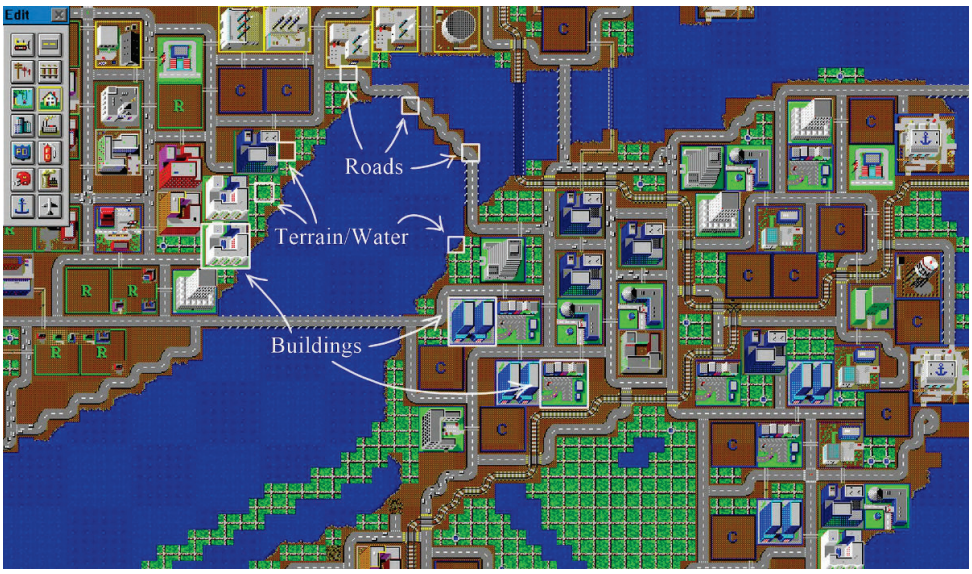
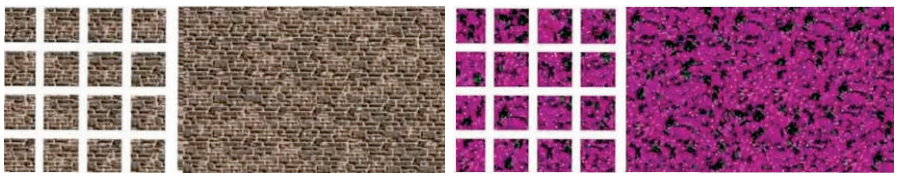


Figure 3.4: Tiles in SimCity, 1989

As time passed, tiling graphics had outgrown its primary role of being solely an asset reduction technique and found a new purpose in today's games as an alternative strategy used to optimise textures [57].

Tile-Based Texture Maps

The key principle of tile-based textures (Figure 3.5) can be described as follows: a program dices up the object's geometry into smaller polygons and then assigns a different texture tile or texture coordinate transformations to each of these pieces [57]. Additionally, several nonperiodic tiling methods can be applied to minimise visual repetition (e.g., Wang tiles).



(a) Bricks (b) Flowers

Figure 3.5: Tile-based texture maps

The most prominent advantage of this approach is that almost an infinite compression ratio (in theory) can be achieved, as an arbitrarily large output can be produced from just a few tiles [57].

The fundamental ideas of modularity applied in modern computer games, however, remained unchanged from the original concept of modular design; a complex scene or an object is broken down into small pieces with the intent of being repeatedly used with itself or other modular components (Figure 3.6). Designed by adhering to a specific set of rules, each of these pieces is fit for quick assembly in many possible configurations seamlessly.

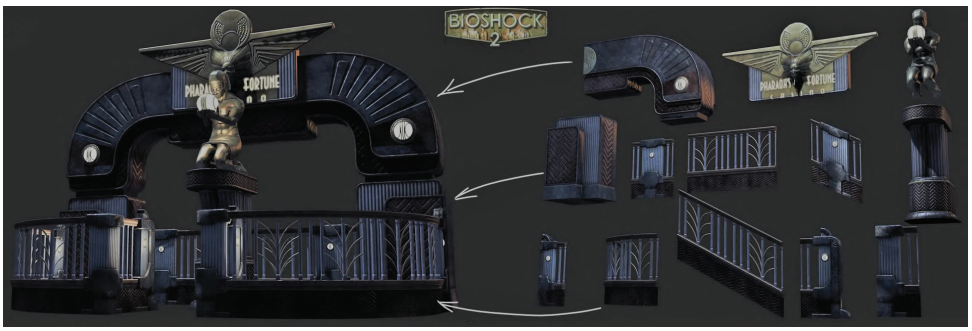


Figure 3.6: Modular environment pieces for Bioshock 2 by Vincent Joyau

Artists, who adopted this approach into their workflow, were finally able to create great-looking, high-detailed levels "*without having to build and texture every nook and cranny of the environments [36].*" Scenes build from reused art, in general, helped to save memory, significantly improve load times through the reduction in unique draw calls, and streamline asset production [28].

Moreover, considering that for modularity to work a minimum of one asset is needed [28], game developers could focus more on the game itself rather than the visuals, yet still, be able to build vast game environments for players to explore with less time spent on the creation of unique assets.

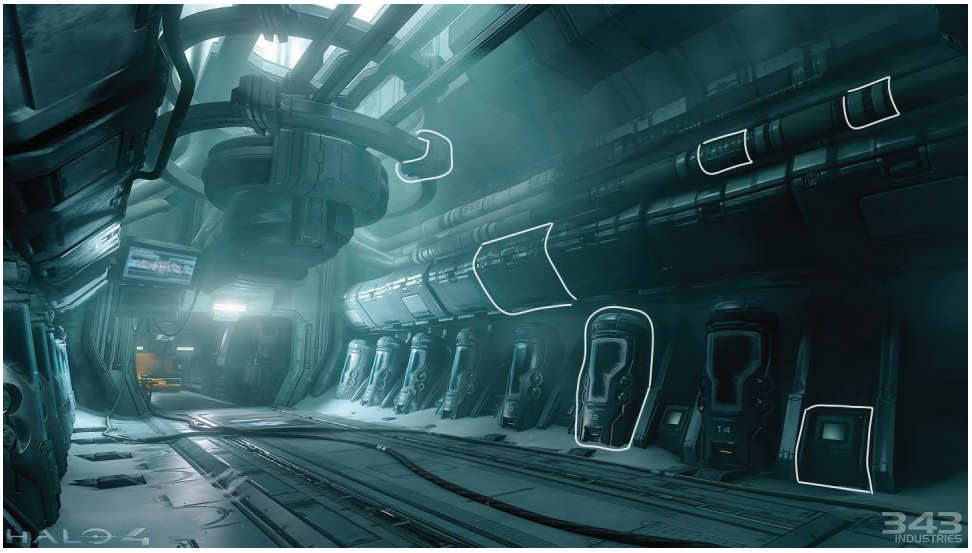


Figure 3.7: Modular assets in Halo 4: Forward Unto Dawn Cryo Room by Paul Pepera

Due to these and other factors, the concept of modular level design has eventually become commonplace in the game industry. Titles like The Elder Scrolls V: Skyrim, Fallout 3 and 4, Assassin's Creed series, Mass Effect and

Halo series (Figure 3.7) are just some of the examples, where modular design techniques were implemented into the production pipeline of a studio.

However, "*modularity is not a shortcut, nor is it a solution for all scenarios*"[11]; for a game developer, it is more likely to be viewed as an investment, similar to new software or technology.

Whereas this approach has proven itself to be a successful and profitable technique in some projects and studios [11], it might not perform that well for the others, as goes the saying, "*one size does not fit all.*" Therefore, it is worthwhile to consider all the benefits and shortcomings of the modular level design paradigm before adopting it.

3.2.1 Advantages

Level Design

The first and foremost benefit of the modular approach is the reusability of game assets (Figure 3.8). That is the quality particularly valuable when building large, open-world environments, like the one in Skyrim or Fallout 4, where not having to create and import every level component for a specific area can save a tremendous amount of time and work.[9]

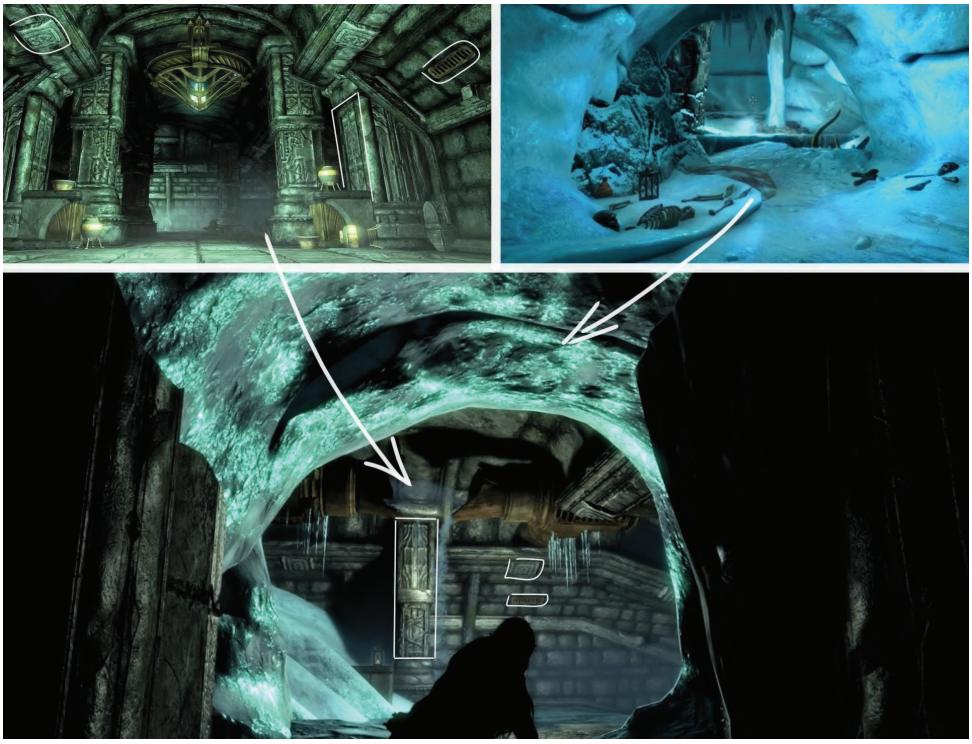


Figure 3.8: Dwemer Ruins kit combined with Ice Caves demonstrates how modular assets can be reused and mixed together to create new environments, The Elder Scrolls V: Skyrim

Additionally, rules and standards established early on for all pieces of a modular kit - a set of modular assets - provide artists and level designers even more flexibility as they can rapidly add, remove, alter, and swap game objects as necessary. That, in turn, enables for a faster iteration over the environment while the project is in the development stage.

Game Production

From a game production viewpoint, modularity helps artists and level designers to efficiently divide their work on a game level making it possible for each group to focus on what they do best without having to wait for one another. Meaning that things like gameplay testing, object interactions, and other tasks outside of the art department can continue while game assets are in production (Figure 3.9) [11].

Figure 3.9: Level designers often work with placeholders to create the base game level layout (blockout). Abandoned Library by Jonah Pankonin

Performance

Apart from design and production benefits, the modular approach gives developers a solution to some of the most significant hardware limitations on the design process: memory allocation and rendering.

Since most of the objects within modular environments are usually highly instantiated, it enables the full use of **GPU Instancing**. The core idea behind

this technique is that with each draw call only identical assets in the scene are rendered; each instances can also have different parameters (Figure3.10), such as colour or scale, to add variation and reduce the appearance of repetition [54].

Figure 3.10: Instances of warriors in Total War: Attila with decals (tattoos, dirt) and gribbles (beards).

Consequently, this approach reduces the total number of draw calls used per scene, and, therefore, significantly improves the overall rendering performance of the project. Besides, having multiple instances of an asset in a game level implies that each update to the original mesh will propagate through all its copies automatically, which in turn can save even more time and work for a designer.

Example

In order to better understand how GPU Instancing works, imagine a forest where all trees are unique assets within that scene. When rendered, each tree will be submitted to the GPU separately - one set of triangles per batch.

A modern CPU can do around 1,000 to 4,000 batches per frame at 30 frames/sec, meaning that only 4,000 trees will be rendered - with no CPU time left for the rest of the game [12].

However, if that forest is built from instances of the original tree, the same geometry will be rendered multiple times within the same batch in a single draw call. That consequently will "*minimise CPU time spent in submitting batches and free up that time for other systems, such as physics, artificial intelligence, and game logic.*" [12]

3.2.2 Disadvantages

While the modular design approach has profound positive effects on both performance and game development process, there are some ramifications which are important to understand and learn how to cope with them.

Repetitive Art

One such drawback is closely related to the earlier addressed inherent nature of modular assets - reusability - which in extreme cases can make a game level look blocky and repetitive (Figure 3.11), taking away the authentic feel of the environment [9].

Furthermore, the lack of enough visual stimulus to persuade a player into thinking that every location in the game is unique might confuse the player's navigation through the level and negatively affect the overall experience from the gameplay.

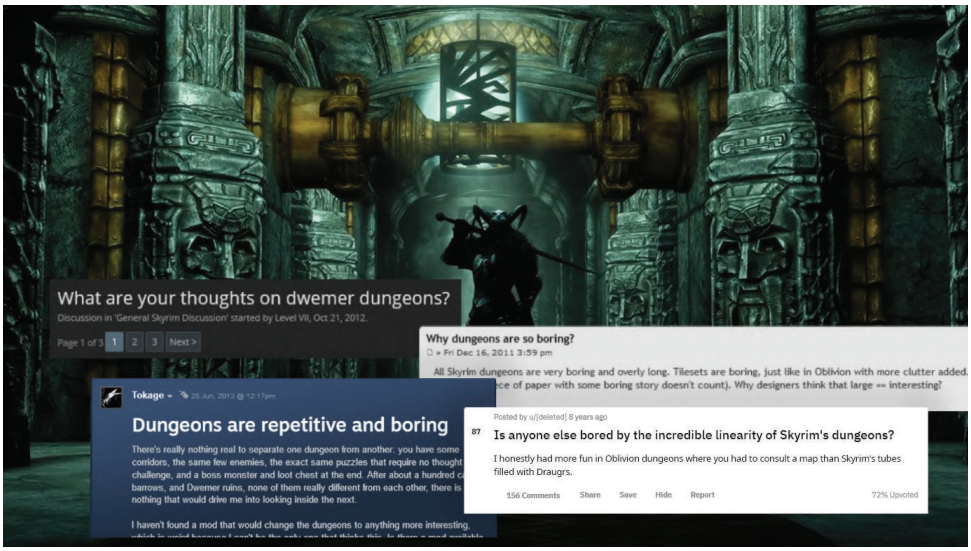


Figure 3.11: Needless to say, how frustrated players can become due to repetitive game levels. The Elder Scrolls V: Skyrim, Dvemer ruins

Generic Layout

"*With word 'modular', initially, old-school, graph-paper-designed Dungeons & Dragons levels come to mind,*"[36] which screen into designers' nightmares.

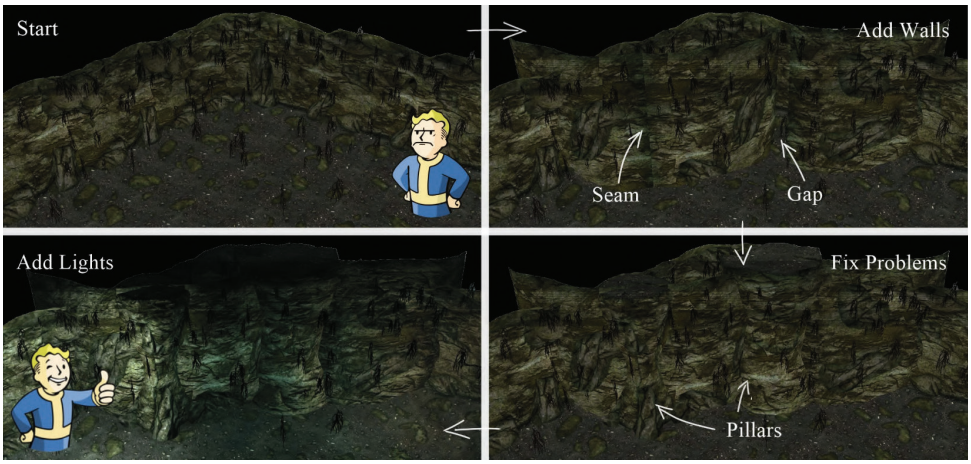


Figure 3.12: Tips on how to make the modular level layout look less generic. Bethesda

Grid-based modular assets are, indeed, very restrictive when it comes to their placement within the game environment. However, that limitation can be easily overcome with some additional kit pieces explicitly designed with the intent to hide imperfect tiling or overlapped geometry in places where level designers decided to go off the grid (Figure 3.12).

Planning

Compared to traditional, non-modular design approaches, modular assets often require significantly more time to create, due to the additional planning overhead. Therefore, game developers should always ask themselves whether the modular level design is something their project will benefit from before adopting it into their workflow.

Example

A single room is faster to create from assets designed to solve one particular task, rather than combining separate modular pieces. A large building, on the other hand, will be faster and more efficient to assemble from smaller generic elements, such as rooms, staircases and halls.

3.3 Art Fatigue

Perhaps, the biggest challenge many artists and level designers have to face when working modular comes from the fact that modular game environments are often built primarily from a reused art that can quickly become repetitive and, therefore, degrade the visual fidelity of the game.

The term “art fatigue” is commonly used to describe these situations where repetitive geometry of environmental components becomes noticeable to a player, and as a result, erodes the authenticity of the world [9].

Figure 3.13: An example of a generic environment in Destiny 2 by Kyoungche Kim.

However, not all game environments suffer the same fate. For example, in

areas that are generic by its nature, such as sci-fi bunkers, spaceships (Figure 3.13) and factories, it is even desirable to have some extent of repetition in the scene.

It is arguable that players tend to relate their in-game experience to the real world where objects rarely repeat themselves in the exact matter; in which case, games are only deviations from realism following predictable patterns that make art fatigue especially hard to overcome [26]. Bringing a massive amount of detail to 3D models, however, does not necessarily have to make a scene look more visually appealing, although, other steps might help to delay, if not prevent completely, the onset of repetitive geometries.

Lighting

The easiest way to hide repetitive geometry is by emphasising the object's silhouette with light and shadow, since it is in the mid-tones where most details are going to be visible [26].

Lighting, in general, plays an essential part in any type of video games, as it has a profound effect on how players perceive the space and mood of an environment [38]. Therefore, apart from helping players to navigate through a game level, lights can conceal otherwise obvious repetition of modular assets and even diminish art fatigue.

Figure 3.14: Notice how lights and shadows add more detail to the otherwise dull concrete environment. Portal 2: Aperture Laboratories.

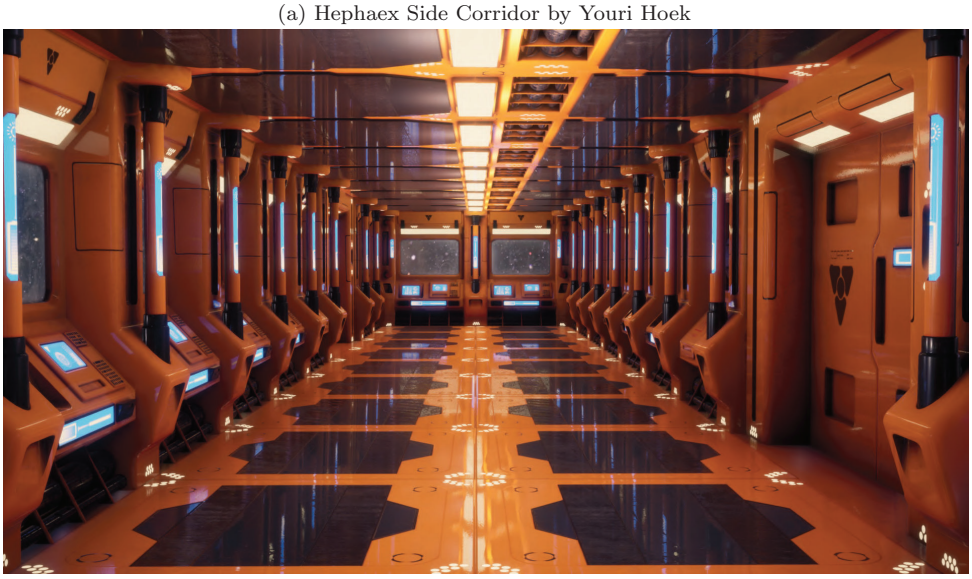
Colour

Using right colour tones and their combinations (Figure 3.15) across different objects can help further disguise duplicated details of environmental

components.

The rule of thumb is to use full and bright colours only on so-called 'hero pieces'; that would draw players attention away from the rest of the setting, making modular assets less noticeable [26].

(a) Hephaex Side Corridor by Youri Hoek



(b) Sci-Fi Modular Corridor by Alessio Verolini

Figure 3.15: Compare how the same four colours (orange, light blue, light and dark grey) can create different look and feel of the environment depending on how artists used them.

Props

In video games, the word "prop" refers to many types of objects within an environment. The best definition would be as follows; "*a prop is anything that supports the scene but is not part of a level layout or character set [33].*"

Props, in general, can be created as components of a separate modular kit (Figure 3.6), or as unique set pieces, design to solve particular problems [11].

As it was pointed out by Joel Burgess, a level designer in Bethesda, players are more likely to notice the repeated detail elements - props - within a game level, rather than broad architectural elements (Figure 3.16) [9].

Figure 3.16: Uncharted 4 Treasury Courtyard (fanart) by Uditraj Vadher.

Therefore, by merely populating the scene with additional assets known as 'accessory pieces' level designers can break the appearance of tiling and even fool players into thinking that a particular location is unique, regardless the modular nature of its components.

There are two types of accessory pieces that are most commonly used in level design: concealment and hero pieces.

Concealment Pieces

As the name suggests, the primary goal of concealment pieces is to cover seams or gaps between objects that do not tile or snap accurately (e.g., pillars in Figure 3.12). This technique is most common in situations where a designer decides to go off the grid to create a more natural or intricate look of the environment.

Hero Pieces

Hero pieces, on the other hand, are designed to be a focal point of the scene, add context, create visual interest and distinguish it from a similar area (Figure 3.17). However, as important as they might seem, these assets should be kept to a minimum [38]. The main reason is that building a single hero

piece is significantly more time consuming than creating several modular components, as it often requires more geometry details and custom textures.

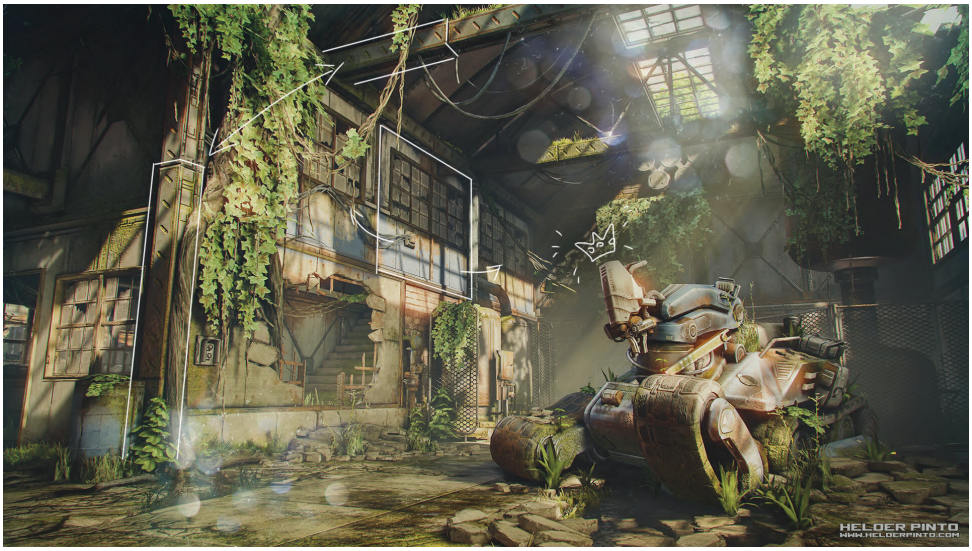


Figure 3.17: Post Apocalyptic Warehouse by Helder Pinto demonstrates a hero asset (with a crown doodle) and some modular walls and windows highlighted for comparison.

Decals and Greebles

Decals are, essentially, materials projected, or "sprayed" onto existing surfaces either by a level designer or a game engine.

"*For instance, a decal applied downward onto a staircase would cascade down onto the top (but not front) of each step.*"[56].

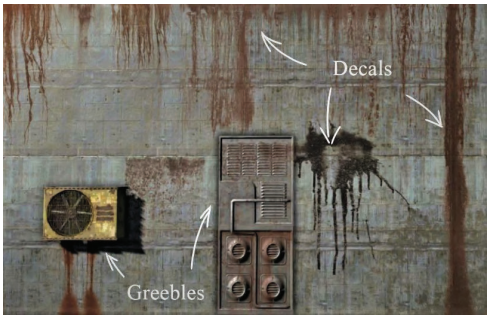


Figure 3.18: Example from Fallout 4

Greebles, on the other hand, are usually small objects that stick to the surface of a larger asset to give it more detail (Figure 3.18).

Mix-and-Match

Finally, if a project has multiple kits, each designed specifically for a particular location or purpose in game, techniques such as kit-bashing, kit-jamming, or kit mashup exist to break the repetitive look of the environment.

Despite the different naming, the core idea behind all these methods is to "mix-and-match" assets from different kits in one scene [11], as demonstrated in Figure 3.8

3.4 Modular Assets

According to the technical interpretation set by game engines, like Unity or Unreal, the word "asset" means a piece of a content used in a game or a project, which can represent an item of any supported type, such as a 3D model, an audio file, or an image [53] (Figure 3.19).

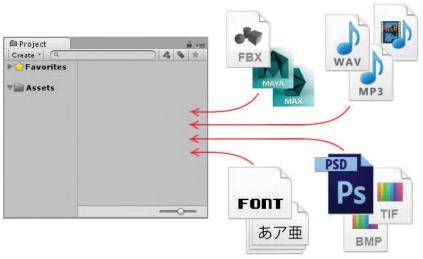


Figure 3.19: Example of supported assets in Unity.

However, when discussing game art for level design, in particular, the definition of an asset can be narrowed down to a character or an object that is meant to appear in a game environment [49].

In contrast to the non-modular approach, modular assets are parts and pieces built with a set of strict rules in mind so that they fit together flawlessly to create easy to assemble game levels or new, more complex objects. This form of modularity often draws comparison with a plastic construction toy 'Lego' (Figure 3.20).

Figure 3.20: Lego brick's dimensions and a Bike shop from LEGO modular buildings series.

Considering that "*the design of each Lego brick is also highly specific and that specificity is what promises it to work as a part of a system,*" [11] the analogy appears to be remarkably accurate and, therefore, is frequently used by artists and level designers to describe the concept of modular assets.

A modular kit, on the other hand, has several interpretations across various types of games and tools. Although, according to the analogy based on the Legos, the level-building kit is a set of modular assets (Figure 3.20). Each piece from the kit can be combined with itself or other modular components to build game environments, compose more complex objects, or, perhaps, create something original, which might exceed the boundaries of the initial idea or intention of the designer of the kit itself.

3.4.1 Modular Kit Fundamentals

Modular kits have many applications across various types of games (both 3D and 2D). Components of those kits not only empower level designers working on their project by hand but also can serve as building blocks for procedurally generated environments.

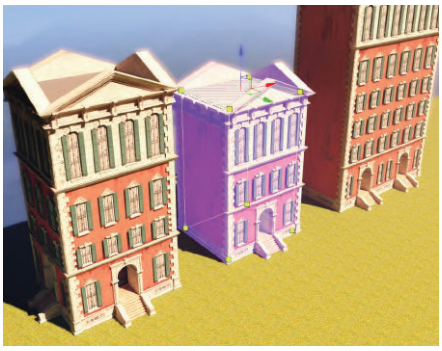


Figure 3.21: Procedural buildings: editable volumes that add/subtract meshes.

Some examples where modular assets were used to procedurally generate assets or whole game levels include Daggerfall, Path of Exile, and BioShock Infinite (Figure 3.21 and Figure 3.22).

Figure 3.22: Buildings for BioShock Infinite by Calen Brait.

Therefore, when adopting a kit-based approach into production workflow, the most important thing to start with is by establishing the purpose of the modular kit and the impact it would have on a player [10].

Scale

Although modularity can be applied to a variety of scales (Figure 3.23), it is often the scope of the environment that defines the exact proportions of all components inside a level-building kit. Depending on whether it is a first player shooter or a racing game, objects can be created as single prefabricated modular blocks or as a combination of smaller fine-detailed pieces, such as walls, floors, and rooftops [36].

Note that the granularity of the kit's assets can be taken even further; for instance, it can result in a corridor wall that incorporates several variations of tiny modular components (Figure 3.24).

Figure 3.23: Modular Old Industrial Building Asset by kangarooz 3d

This decision regarding the base scale and proportion of the kit should not be taken lightly, however, as it can directly affect both the amount of time an artist will spend working on the assets and the visual fidelity of the final scene.

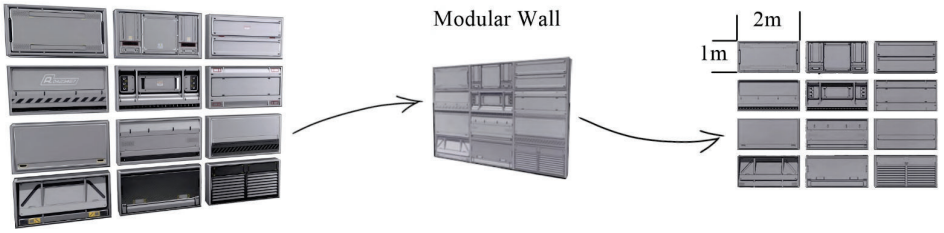


Figure 3.24: Concept art: Horizontal wall panels 01 by Richard Nixon for After Reset

Example

Some of the most visually demanding game environments are enclosed spaces such as spaceships or dungeons, as they require the finest level of detail and variety. Therefore, modular kits designed for these areas typically consist of either multiple variants of a single piece or much smaller components than usually, e.g. different parts of a wall (Figure 3.15a and 3.24).

Grid

"*Regardless of scale, when creating a set of modular components, nothing is as important as the almighty grid [36].*"

Every level editing and 3D modelling program has some kind of grid system. For example, game engines such as Far Cry, Max Payne, and Unity, use a grid system with meter units. Id Software's Doom series and Epic Games' Unreal series, on the other hand, tend to work with the unit grid system [31].

While it does not usually concern an artist what kind of grid a particular program has, for a level designer to manage a project efficiently, the grid is fundamentally important.

To guarantee seamless and fast level design experience, it is, therefore, crucial to set up the grid system of 3D modelling package to correspond with the one used in a game engine before any actual work on the game environment starts.

Taking this step will ensure consistent workflow and prevent any undesired scaling and tiling issues [17].

Further Considerations

While making sure that the object's geometry aligns perfectly with grid lines is important, it is also worth considering possible interactions the asset will have with other objects in the environment and make necessary adjustments in advances.

- If there is a chance that any other item in the scene will be placed on top of the piece, or the asset itself is built to be stackable, its geometry along the Z-axis should also be on the grid.
- A piece that will be pressed flush against another object (e.g., a balcony or ledges for a building) should have all its parts that will connect to that object aligned with gridlines as well.
- Finally, when creating a modular component of a different size than the rest of the kit, only even divisions of the grid should be used at all times.

Footprint

Some industry professionals suggest that when considering creating a modular kit, the first thing to decide and understand about it should be a footprint, not the grid, nor the scale.

"*The volumetric footprint of the kit is, essentially, the grid; the size of pieces and the shape on which they are going to snap and tile on each other.*"[11] (Figure 3.25)

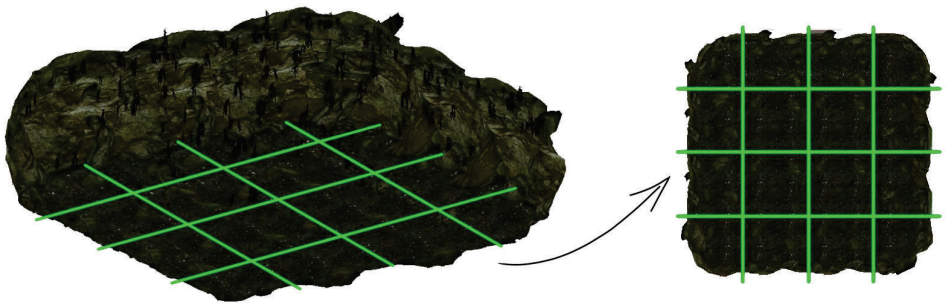
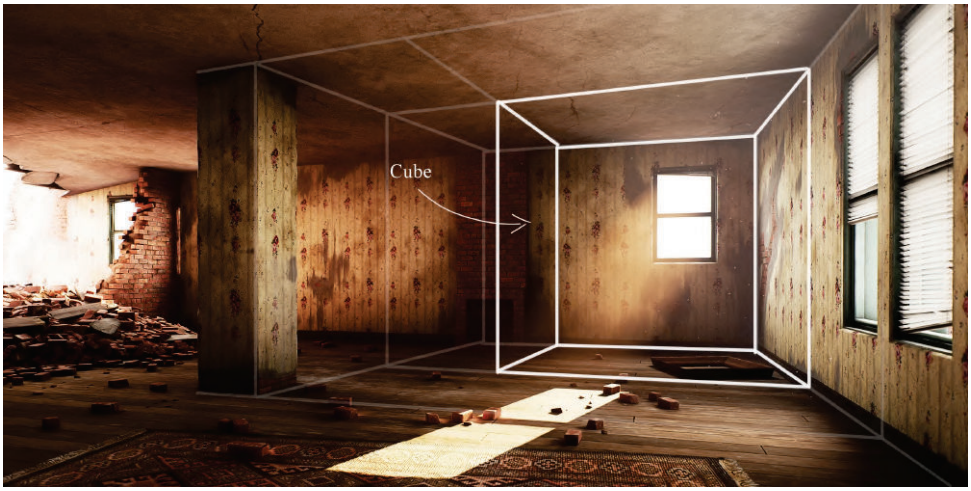


Figure 3.25: Example of a Cave kit's footprint provided by Bethesda

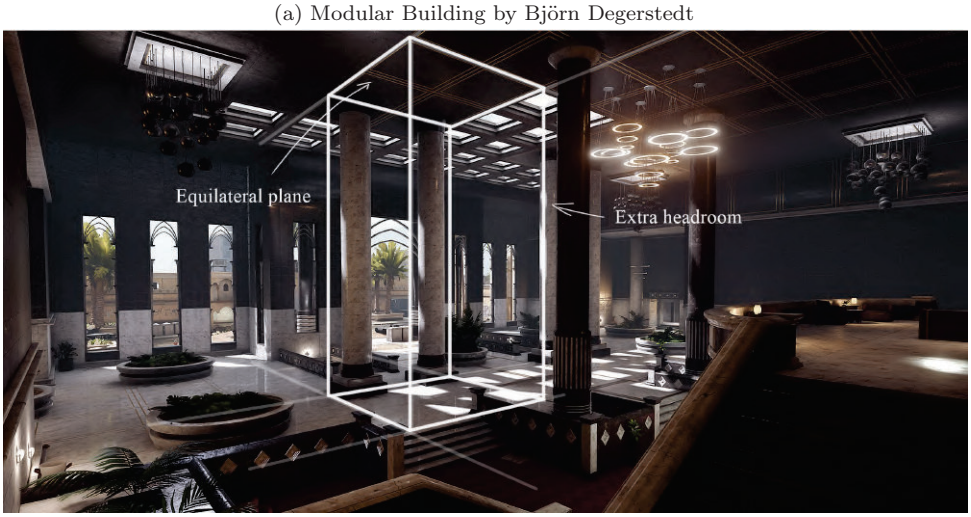
The most common type of footprint is an equilateral cube (Figure 3.26a), since it can tile with itself cleanly in all directions and can give a certain kind of aesthetic and proportion of space to a game level. However, the downfall of this approach is that an equilateral kit tends to look the most repetitive and generic.

The non-uniform footprint, on the other hand, gives designers more building freedom, although can limit the ability of a piece to tile and snap back on

itself, often resulting in a so-called one-directional flow kit.



(a) Modular Building by Björn Degerstedt



(b) Insurgency Sandstorm: Ministry by Vuk Banovic

Figure 3.26: Example of a uniform kit and a kit with extra height and equilateral base

The most reliable approach to the problem, therefore, is to change only the height of the footprint (Figure 3.26b), which would immediately make it possible for a designer to create a different look and feel with additional headroom while keeping the advantages of the equilateral plane [11].

Sub-kit's footprint

Various sub-kits of one kit do not have to share the same footprint. Although, those footprints should be multiples of each other to avoid possible complications with tiling or the kit's ability to loop back on itself [9].

Another important property of the footprint is that it represents the full bounds of an object' s geometry - the maximum extent of a modular piece that can not be traversed (Figure 3.27).

Therefore, when working with a kit, an artist should make sure that all its components fit entirely within their allotted grid space. The only time when a piece can occupy the absolute edge of the footprint, however, is when it is intended to snap with another piece from the kit.

Example

Consider the wall piece built along the edge of the footprint. If an artist attempts to create two adjacent hallways with it, there would be no room left for the walls themselves to exist.

That would place unnecessary restrictions on a layout or result in overlaps, and z-fight flickering (co-planar planes).

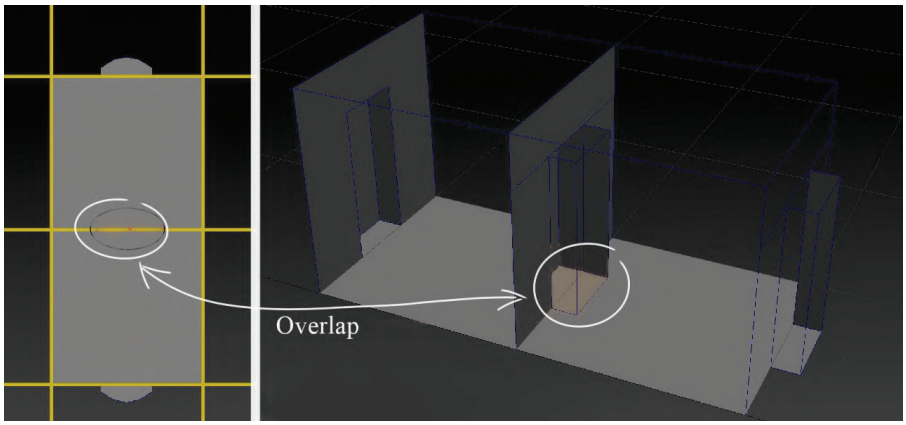


Figure 3.27: Example of a wall piece that exceeds the boundaries of the footprint (yellow grid) provided by Bethesda.

Tiling Rules

Depending on the purpose of a kit, its tiling rules can vary. For example, hallways tend to tile on only one axis, while rooms typically tile on two, and an all-axis tiling kit can easily snap on itself and create spaces that stretch out in all directions.

Although, for an additional tiling, instead of the all-axis kit, it is generally wiser to build a separate sub-kit with a fixed plane size along the desired axis [9].

Pivot Point

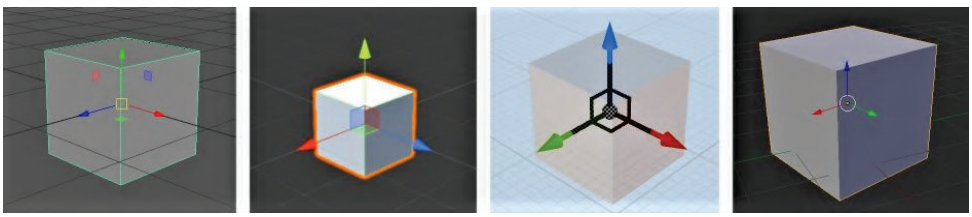
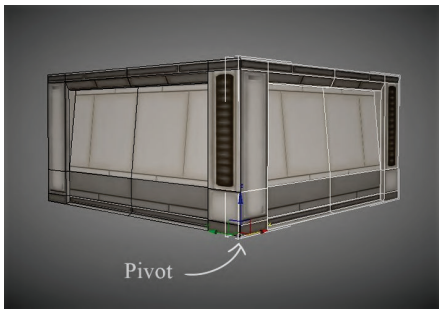


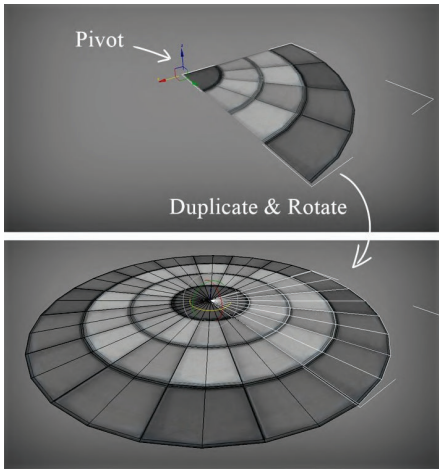
Figure 3.28: Pivots of the model in Maya, Unity, Unreal and Blender respectively.

A pivot (Figure 3.28), also the origin, is a single point in virtual space, that represents the object's local centre and local coordinate system, and is primarily used for rotation, translation and transformation of the object in a game engine and a 3D modelling software [3].

In a kit-based modelling approach, pivot points are especially important, since their position has a profound impact on a level designer's workflow, and snapping properties of modular assets (Figure 3.29). Therefore, the placement of the pivot point should be carefully planned and consistent throughout the modular set. Besides, once its location is established on an individual piece, it should stay that way; otherwise, any alteration during the production phase might undermine any work a level designer has already done with that asset [11].



(a) Corner



(b) Circle centre

Figure 3.29: Example of correct pivot placement by Thiago Klafke.

Regarding the correct placement of the pivot, a general rule-of-thumb is to put it in one of the plane extremities - lower corners of the mesh [29] (Figure 3.29a). For example, in Skyrim and Fallout 4, most of the pivot points are positioned at the bottom centre or a bottom edge of a piece.

However, exceptions do exist; a pipe kit is one of them. There the pivot is often placed at the edge of a pipe piece to provide better snapping and rotation [9].

The universal rule says, that "*as long as the pivot position is mathematically accurate, it is, to a certain extent, a matter of personal preference.*" [11]

"The Five Steps to a Perfect Pivot", Paul Mader [31]

1. Place pivot in the focal point of a model.
2. Check for any symmetry lines and position the pivot on one of them or in the point of intersection.
3. Determine which type of alignment the model has with an environment (ground, ceiling, wall, or their combination)
4. For a tiling segment, place the pivot to a point where it is supposed to snap with the next piece.
5. If a modular component is created as a part of a circle, skip all previous steps and shift the pivot to the circle centre right away (Figure 3.29b).

Kit Elements

Modular kits are time-consuming to create, due to all the planning and considerations they require. One way to minimise the time spent building them is to determine early on which modular components will be needed to create the desired environment and how important they are.

In general, all elements of a level-building kit can be divided into three categories (Figure 3.30) [11]:

- Hero Pieces - low-use high-detail unique pieces, designed to draw the player's attention (Figure 3.17).
- Utilitarian Core Pieces - walls, floors, staircases - all the parts needed to create a basic layout, establish the flow of a game environment (Figure 3.23).
- Variant Pieces - cosmetics, alternates, and pieces that might be a part of the core kit but are used less frequently or specialised to accommodate particular needs of the base layout (e.g., props in Figure 3.16 and concealment pieces in Figure 3.12).

Figure 3.30: Modular House kit by Knife Entertainment with some highlighted elements.

Despite the stigma attached to words "generic" and "repetitive", level-building set assets are the most versatile and flexible to use in comparison to hero pieces, which are often designed to solve a specific problem and rarely occur more than once in a game environment.

Therefore, the best practice is to focus on the core pieces first and depending on the needs of a level designer gradually work up to other two types of the kit's components.

This approach generally saves a considerable amount of time for both level designers and artist working on a project.

3.4.2 Assets Creation Techniques

As the concept of modular level design has been used in a diverse number of computer games of various genres and styles, artists and game studious have developed several approaches and techniques tailor-made for their unique purposes, which they continue to innovate toward more flexible and reliable systems.

Regardless of the great diversity among these methods, it is possible to divide them into three base categories according to some fundamental aspects of a modular kit, such as shape and granularity of its components.

Planar

The planar method is, arguably, the most popular technique in the game industry. As the name suggests, planar modular assets are single-sided uniform modules that can be snapped together in a 'jigsaw' style to build modular environments [18].

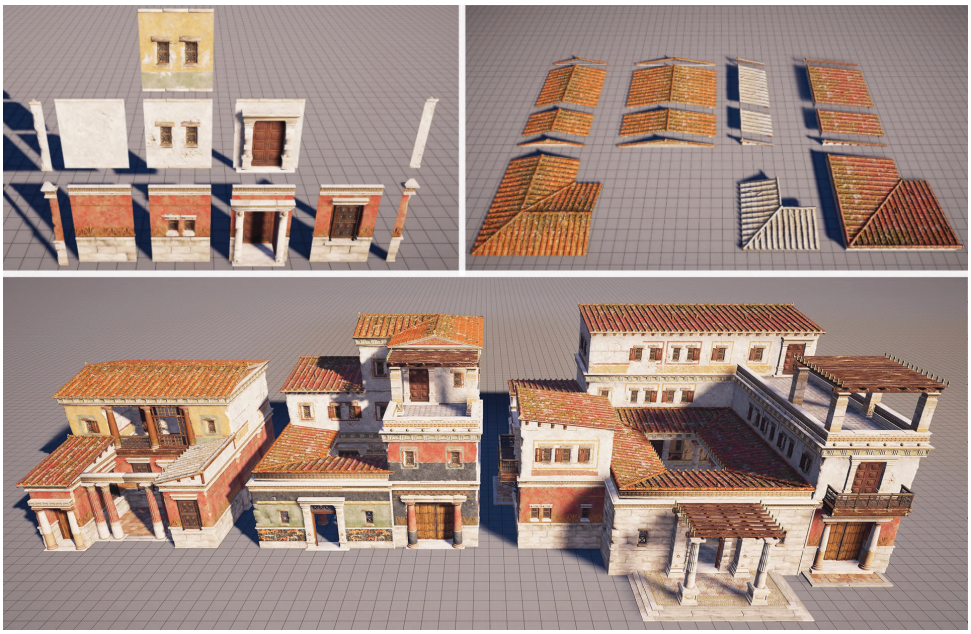


Figure 3.31: Assassin's Creed Odyssey: Rich Villas Architectural Kit by Olivier Carignan

Examples of these assets can be found in games like Skyrim, Assassin's Creed series (Figure 3.31), Dark Souls 3, Bloodrayne 2, Fortnite, and many others.

The key advantage of the planar method is that it is fast to develop and

texture due to the simple topology of individual components. Additionally, it reduces the total number of unique modules needed to be produced, yet, maintaining enough flexibility and variety of assets required to create rich and engaging environments.

However, despite the higher level of control over the scene achieved through the use of small interchangeable parts, the workflow of level designers became significantly more timeconsuming (Figure 3.32).

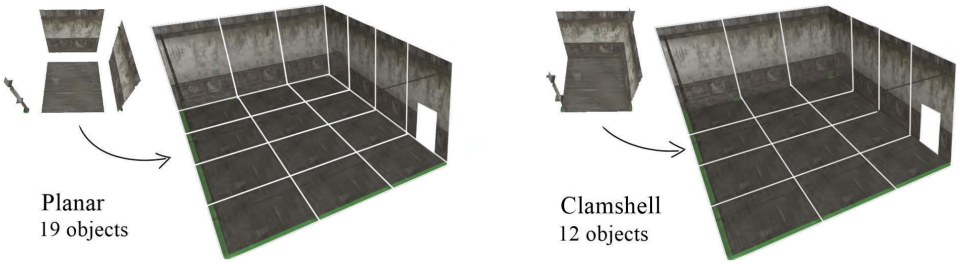


Figure 3.32: Comparison of two methods

Box

In contrast to planar, the fundamental idea of the box method is to create larger modular components which can be used independently or intersecting with other assets in the scene to build a variety of structures and more complex game areas [18] (Figure 3.33).

Figure 3.33: The Witcher 3, building kit

Games like Diablo 3, Spider-Man, Total War: Warhammer and The Witcher series (Figure 3.34) are some of many examples, where box modularity was employed.

The primary advantage of this technique is that, compared to the planar method, it requires a smaller number of objects in the scene to achieve a satisfying amount of visual fidelity, which results in faster level design iterations.

However, areas filled with these assets are more prone to the onset of art fatigue, and therefore, require more time and resources spent on concealing repetitive geometries.

Figure 3.34: The Witcher 3: Wild Hunt - Blood and Wine DLC by Maciej Caputa

Hybrid

Another approach proposed by the team of artists and level designers from Bethesda is based on concepts of the planar method and the power of prefabs. In other words, planar pieces are often assembled inside an engine into complete modular components - boxes [11].

This new way of assets creation is said to be beneficial for both level designers and artists, as all modular pieces remain highly granulated and yet, similar to the box method, the total count of unique assets placed in a scene is significantly reduced.

At this point, however, it is hard to judge how well this method performs in comparison to other techniques since it is known to be implemented in only one game so far.

Clamshell Method

A clamshell method is an intermediate step between planar and box methods which combines small single-sided modules into bigger, more detailed modular components (Figure 3.32). Each such asset often represents a particular part of a game level or an architectural structure.

While this approach decreases the total amount of separate objects in the scene, reducing the time spent on building the game environment, it requires artists to produce more unique assets to accommodate all needs of level designers [11].

Chapter 4

Combining Procedural Techniques with Modularity

The most reliable way to a better understanding of any new concept is to apply everything learnt in practice.

Therefore, the goal of the following chapter would be to create a modular environment kit, that is hoped to provide further insight into other benefits and drawbacks of this approach as well as help to conclude comparison between modular and non-modular workflow.

Why focusing only on the workflow?

The most straightforward explanation for the decision made would be that nearly all games make use of modularity to a certain extent. For example, some might benefit from just working on the grid, while others tend to implement several but not all modular techniques on a case by case basis.

Thus, the interpretation of what constitutes a modular and non-modular environment becomes especially hard to define without making a forced comparison of fundamentally different genres and purposes of games.

Moreover, as the performance benefits of modular environments were already explained and thoroughly studied in the Section 3.2.1, additional testing might not yield any new information, apart from what has been already discussed.

4.1 Defining the Scope of the Project

The first step of the project was to determine its theme and chose the right tools that would aid in achieving the desired results.

Even though sci-fi corridors, spaceships and dungeons are some of the most

popular examples of environments that work best with modular concepts, they were quickly dismissed from consideration mostly due to personal preferences. Instead, it I have decided to create modular assets suitable for constructing buildings similar to those found in Amsterdam (Figure 4.1).

Figure 4.1: Canal houses in Amsterdam. Photo by Alfons Taekema on Unsplash

The next step was to collect as many reference images as possible (Figure 4.2) since a good reference collection can not only give inspiration and shape a better idea about the project but also can help to solve some problems concerning its creation visually [1].

Figure 4.2: Reference images collected in PureRef

As for the tools, the majority of articles found regarding modular design uses polygonal modelling techniques, sometimes combined with digital sculpting. While those have proved to be sufficient for achieving desired visual fidelity of modular assets, the overall look, dimensions and number of kit components

were often limited to what an artist regarded as most appealing and significant to include.

Procedural modelling, on the other hand, can produce an almost unlimited amount of variants, shapes and sizes of objects. That combined with modular concepts would provide an even more flexible building kit, with modular components that could be easily adapted to the needs of any artist or level designer, who might later work with it.

Therefore, the final scope of the project was set to create a building kit, which would consist of procedurally generated modular assets suitable for a first or third person game.

4.2 Planning the Modular Environment

The next step of the project was focused on implementing all the modular techniques discussed in the Section 3.4.1.

4.2.1 Scale

The most common approach to architectural modular kits is to create walls that are already combined with different types of windows and doors (Figure 4.3).

Figure 4.3: Modular Assets for a Commercial Modular Street Pack by Finlay Pearston.

However, according to the scope of the environment, following this method would require a significant number of variants to achieve a similar visual fidelity, which in turn might work against the concept of modularity. Thus, a different level of assets granularity was adopted; The final modular kit would consist of walls, doors and windows as separate elements (Figure 4.4). Among its other components would be cornices, pillars, stairs and gabels (divided into a “neck” and a “hat” part).

Figure 4.4: Example of modular assets needed for the building kit.

Determining the proportions of these assets, was perhaps, the first significant challenge of the project, as the dimensions of the windows would vary from small basement awnings to windows as large as doors and even higher.

The primary strategy was, therefore, to take a reference image, put a grid over it and outline parts of the facades; then measure the space occupied by each of these elements and compare the results gathered from other images (Figure 4.5).

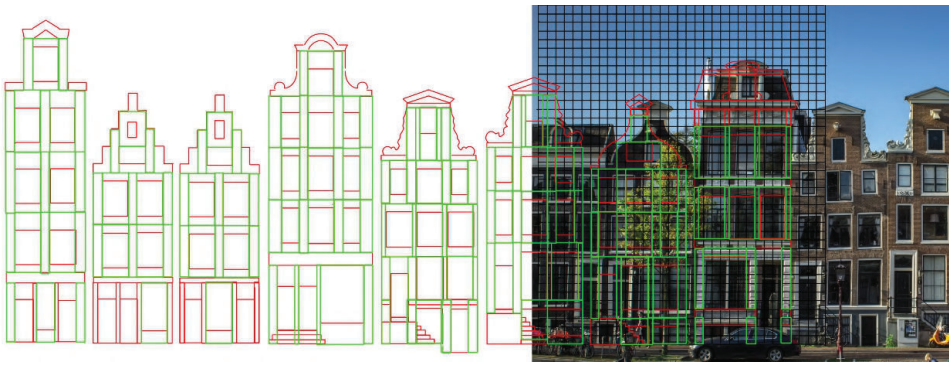


Figure 4.5: Example of the described process.

Unfortunately, all that this approach did was consuming a significant amount of time, without yielding any useful information (Figure 4.6).

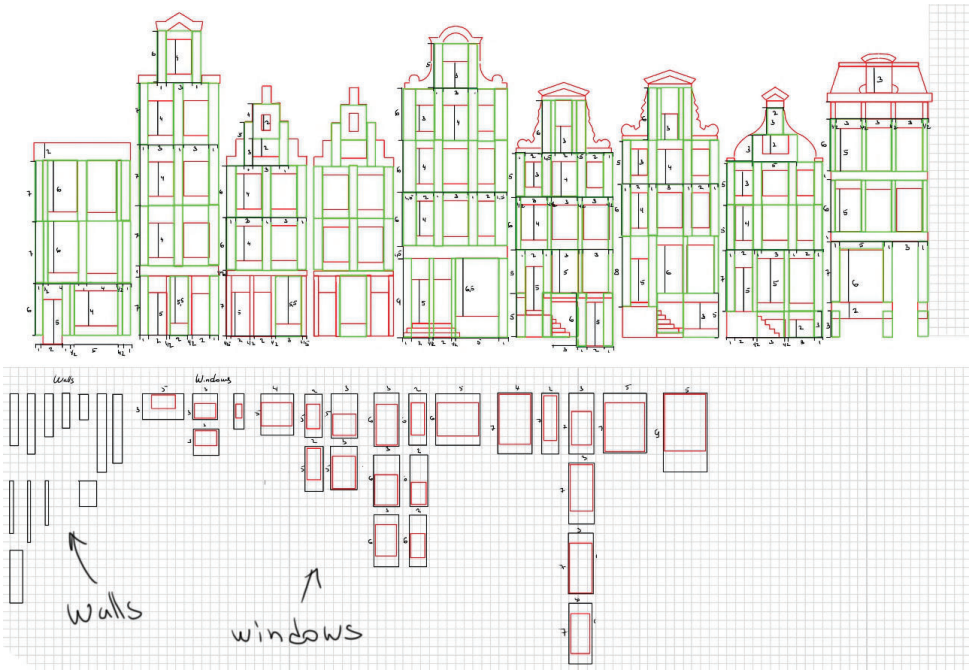


Figure 4.6: Below are walls divided into various sizes and windows with their unique dimensions and positions on the wall. Tiny numbers on the facades indicate the amount of grid space each component occupied.

4.2.2 Grid

The grid system was set to 16cm increment in both Houdini and Unreal Engine to ensure a smooth workflow between both programs (Figure 4.9).

Since some references included images with people standing next to buildings, it was possible to approximate the size of various facade component according to the average height of men, women and even children (Figure 4.7). These results were then combined with some standard dimensions found for doors and windows.

Figure 4.7: Example of using reference images with people to determine assets dimensions

The last stage of the process was performed in the Unreal Engine with a mannequin model for reference, which is roughly 180cm height, stairs and two boxes, that substituted for a window and a door asset (Figure 4.8).

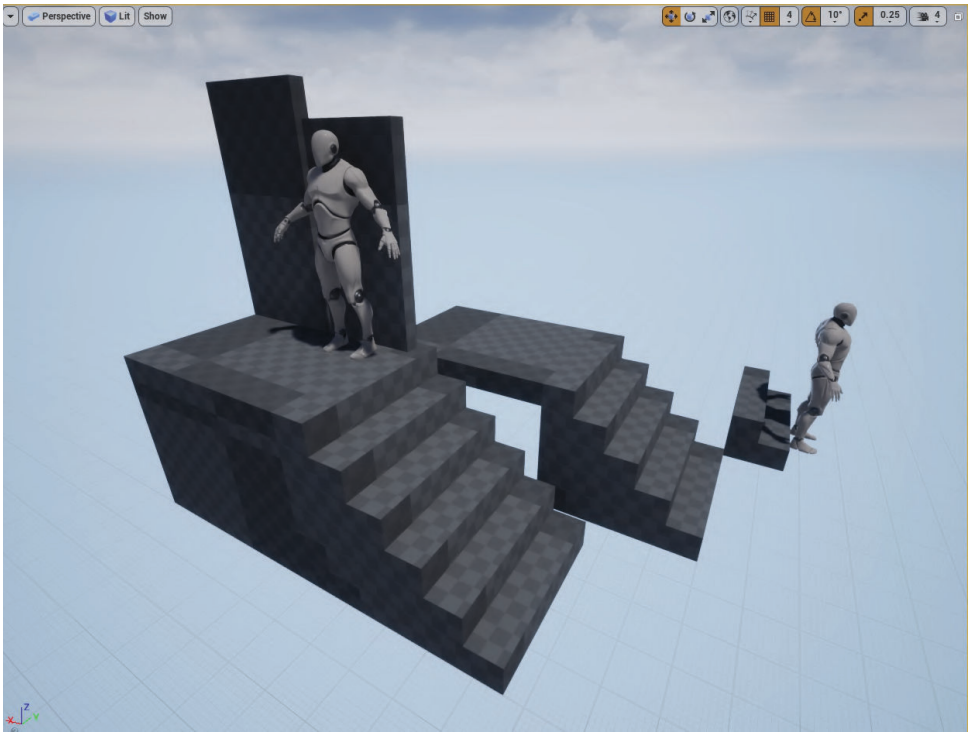


Figure 4.8: A scene in Unreal Engine with two mannequins, several stairs of different heights and number of steps, a door and a window box (top left corner behind the mannequin.)

After several experiments with various grid spacing settings, maximum and minimum proportions of models required for the kit, and tests performed on the assets ability to snap on each other, it became clear that 16cm icrement for the grid space would be the ideal number ruquired by the scope of the project's environment.

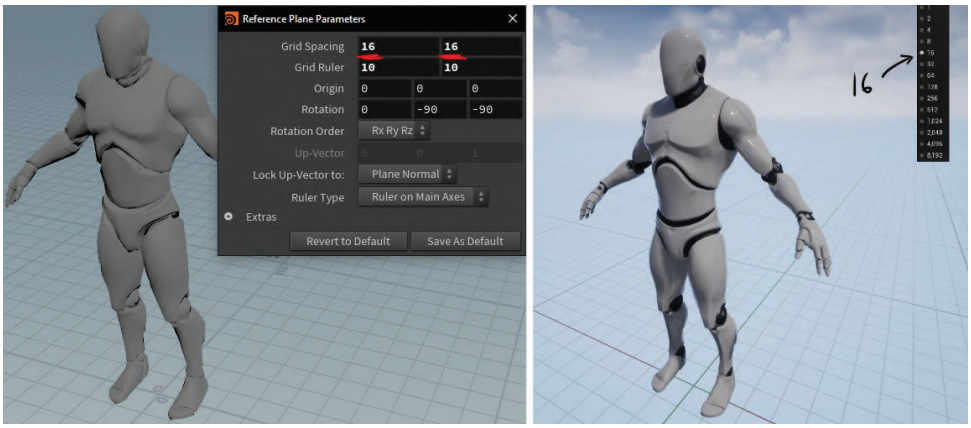


Figure 4.9: Grid system settings on the left in Houdini, on the right in Unreal Engine. A mannequin model was used in both programs for size reference.

4.2.3 Footprint

For the kit's footprint, I have decided that an equilateral cube would look too generic and destroy the authentic look of the facades. Therefore, another

type of footprint, as described in the Section 3.4.1, was chosen. It was hoped to provid a reliable tiling of assets on the plane along x and y-axis with additional headroom for buildings to grow along the z-axis.

4.2.4 Pivot

For the majority of modular assets in this project, the bottom left corner of the object's geometry proved to be the ideal position for the pivot point. However, some exceptions to this rule accured later during the project, which would be covered in the following sections.

4.2.5 Tiling

According to the research conducted on building facades at the very start of the project, it was clear that a two-side tiling along x and y-axis would be the most desirable. There was no need to provide additional tiling along the z-axis as each building could have a different height depending on the type of floors and facade components used to build it.

4.3 Creating Kit Elements

This section would cover each step of the production process for all the modular kit's components from the planning to the actuall modelling inside Houdini.

4.3.1 Windows

The first step was to determine what types of windows would be needed to create a believable game environment.

Figure 4.10: Some windows from reference images were cut and sorted according to their shape, size and style.

The process shown in Figure 4.10 helped to identify six most common types of frames: simple, with either transom or mullion and three combinations of the last two (Figure 4.12). Additionally, frame heads were also divided into several shapes: straight, elliptical, round and a combination of an arch with straight corners (Figure 4.11).

Figure 4.11: Example of each frame head shape

Figure 4.12: Example of each frame type

Before creating a procedural, modular asset, after studying some of the main features of windows, another thing to consider was how the model would

be structured, which parts would be modifiable and to what extent. It was then decided to create the procedural asset with editable parameters for all essential parts of the window so that the end-user would be able to create any type and style of the asset required

The procedural workflow then started with creation of a simple plane object with a grid node; its dimensions along x and y-axis were set to the increment of 16cm.

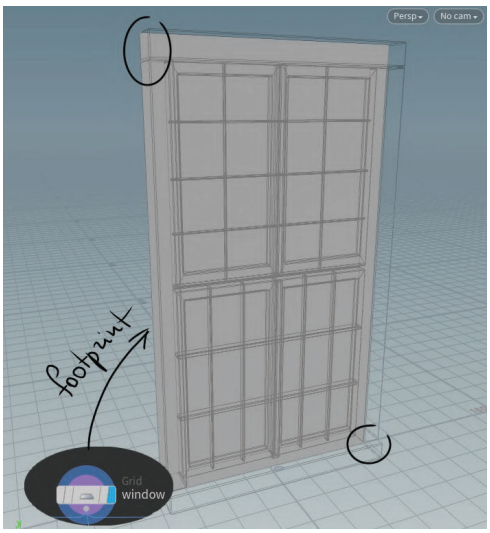
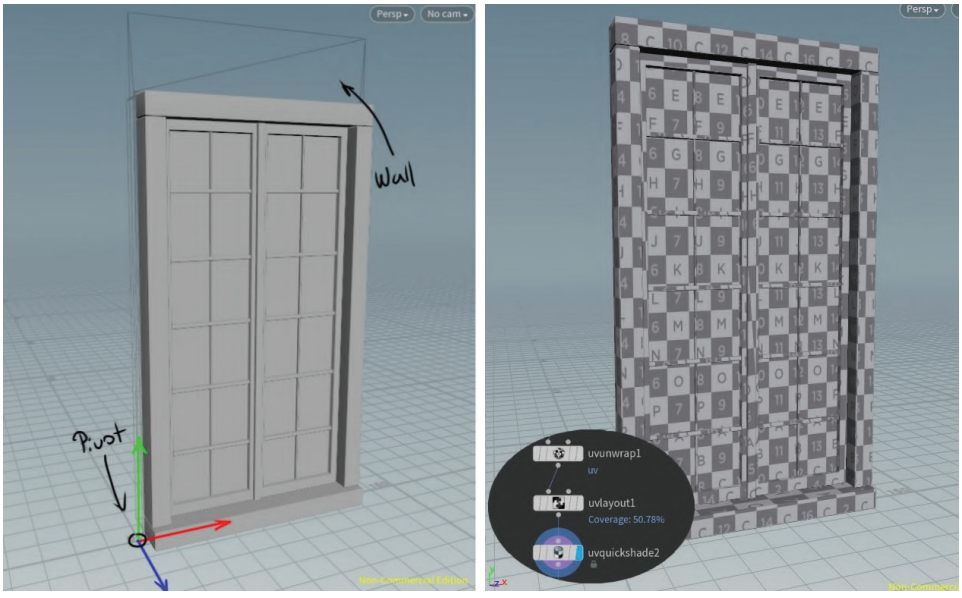


Figure 4.13: Example of a grid node and window model in Houdini

This plane then served as a maximum extent to which the modular assets geometry could grow. In the Figure 4.13 we can seen that all elements of the final model (some of which are circled) were kept inside the space occupied by the plane's geometry.

The main reason for this approach was to keep the asset's ability to tile and snap with other kit's components intact, while at the same time provide enough flexibility regarding the inside shapes of the window.



(a) Pivot (b) Checked texture

Figure 4.14: Example of the pivot point position of the window and uv-unwrapping nodes with a test texture applied to the model.

This step was then followed by the modelling process of each part of the window separately. When these elements where finished, they were combined

into the final model with boolean and merge nodes used according to the particular need of the geometry.

The pivot point for the window was then put at the bottom left corner of the wall footprint. As seen in the Figure 4.14a, the placement of the model's origin is slightly shifted compared to where the actual geometry. Despite how strange and unusual it might seem, in practice, the asset's tiling and snapping worked just the way it was expected.

The unwrapping of the model was also done procedurally with the uv-unwrap and uv-layout nodes (Figure 4.14b).

4.3.2 Doors

The planning phase for the door asset was very similar to the window asset. It started with gathering as much information as possible from reference images, then deciding on what types of parameters would be needed and how many.

Same rules as for the window's tiling, footprint and pivot position were applied to the door.

Copying the workflow from the window asset first was created a plane, which represented the maximum extent to which the door's geometry could grow.

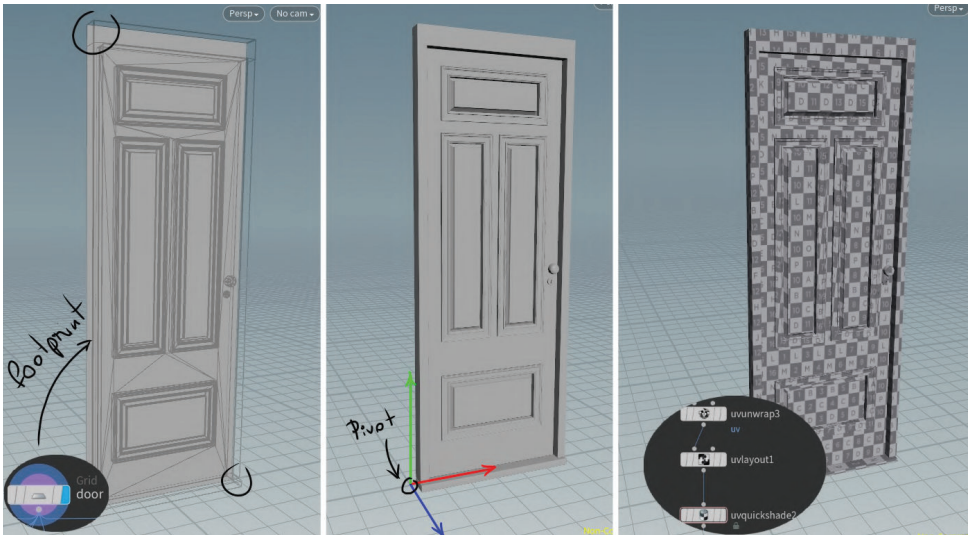


Figure 4.15: Example of a footprint, pivot point and uwrapped uvs.

Then the pivot point's position was set to the bottom left corner of the wall's geometry. Finally, the model was uv-unwrapped with a checker board texture applied to it for testing purposes (Figure 4.15).

For frames around panels and door lite, a separate subnetwork was created, which included around 44 different curves that could be swept along the line of the frame to create the desired geometry (Figure 4.16).

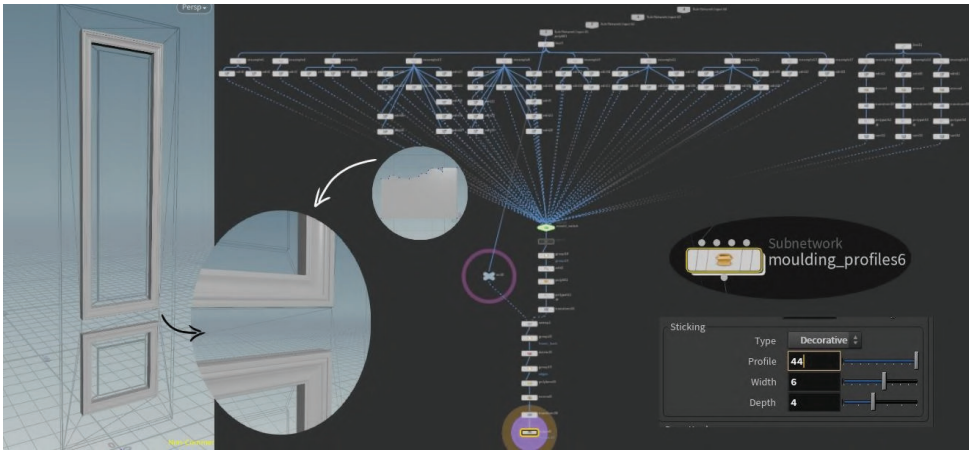


Figure 4.16: Example of the moulding profile subnetwork graph, its parameters and the final look of the frame around door panels.

That subnetwork then was used throughout the project whenever it was needed with some modifications made either to the curves it included or some other parts of the graph to suite specific needs of the particcular model.

4.3.3 Stairs

Since stairs were already used as a scaling reference object in Unreal Engine, it was possible to skip the planning part of the process and dive into modelling right away.

The staircase, steps and the top platform were all created procedurally, using similar tools and techniques as for the first two assets. However, the rail post was instead modelled manually by gradually extruding and scaling faces of the cube's geometry (Figure 4.17).

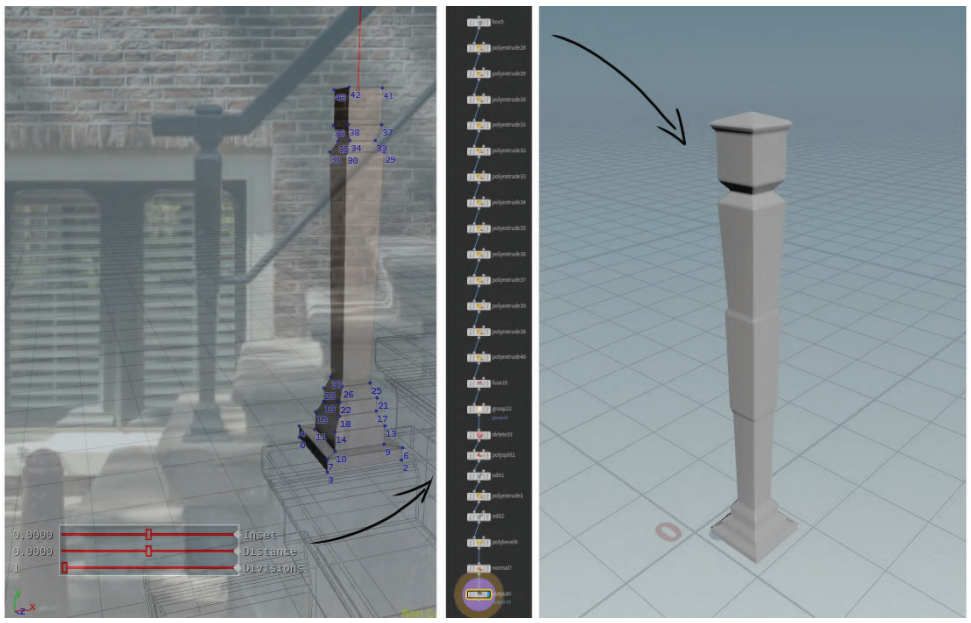


Figure 4.17: Example of the modelling technique used to create rail post for stairs.

Then the sweep node was used to create rails by sweeping the shape created from the curve along the line (Figure 4.18).

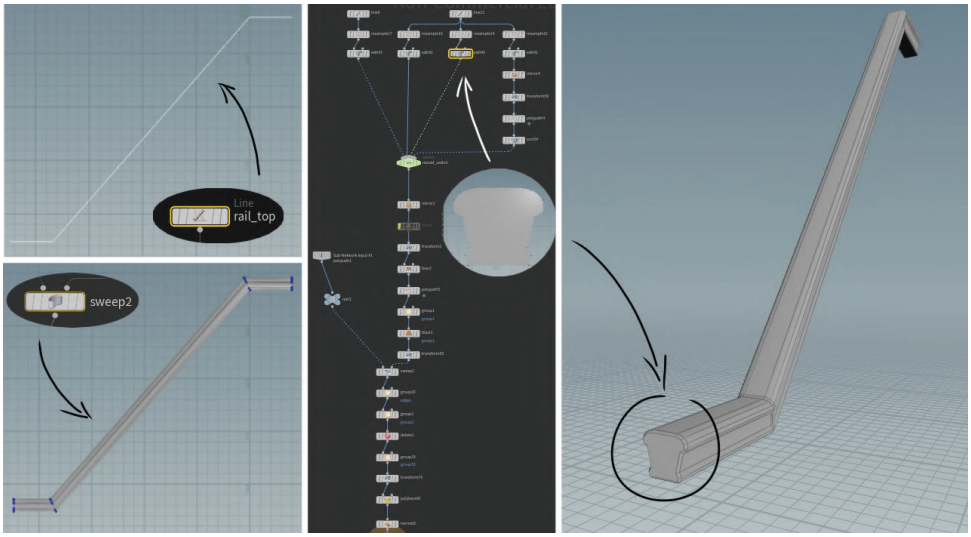
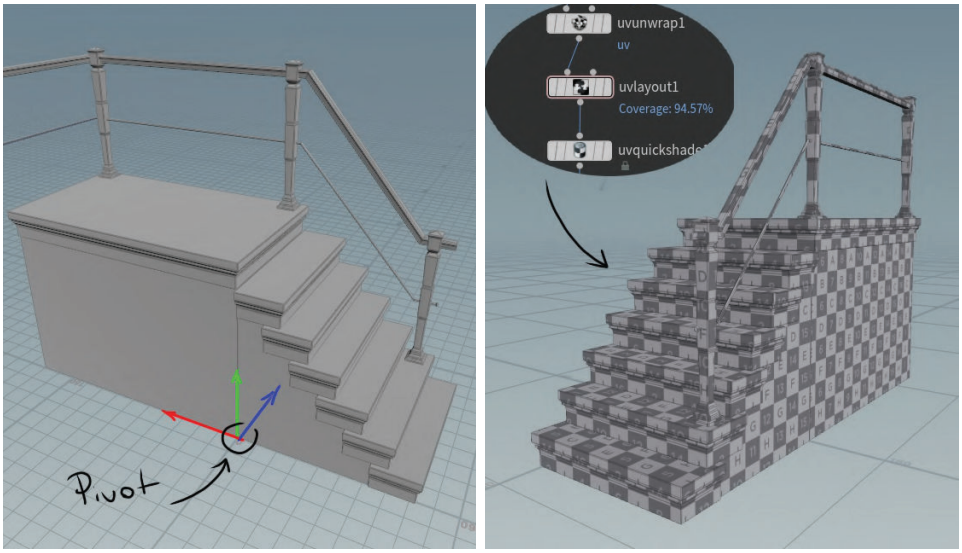


Figure 4.18: Example of the methods used to create rails for the stairs.

The pivot point for the stairs was placed at the bottom left corner of the platform, on the side, which would be close to the building wall (Figure 4.19).



(a) Pivot (b) Unvraped uvs

Figure 4.19: Example of the pivot placement for the stairs and the unwraped uvs.

4.3.4 Cornices

After the thorough study of reference images, I have noticed that cornices, in general, does not have to be limited to large individual pieces that can be only used on the uppermost part of a building. Smaller variants of the same asset might work as rims or even door frames, consequently, saving time and effort needed to create unique assets for each purpose.

Additionally, this approach was hoped to provide better performance results as more instances of the same model would be used in the final scene.

I have also decided that, apart from the main tileable cornice asset, two other variations would be required for the modular kit — a cap piece with the set length of 16cm and a corner (Figure 4.20).

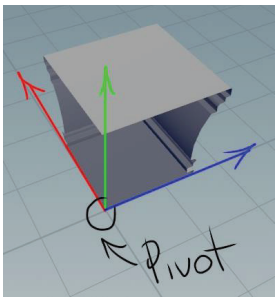


Figure 4.20: Pivot on a cornice corner

The workflow, that was then used to create cornices was, in essence, the same as the one used for rails. All three types of cornice assets started from a simple line geometry along which a curve was swept to create the desired shape of the model.

The pivot point was set to the bottom corner of the left side of the geometry for all variants of the cornice asset.

4.3.5 Pillars

Modelling of a pillar asset was one of the easiest and the most straightforward processes.

Curves used in the moulding subnetwork were mirrored along one of the axes to provide some additional variations to the model, and then again just swept along the line.

Position for the pivot point was clear from the very start, as the model's geometry and the purpose were very similar to a cornice piece (Figure 4.21).

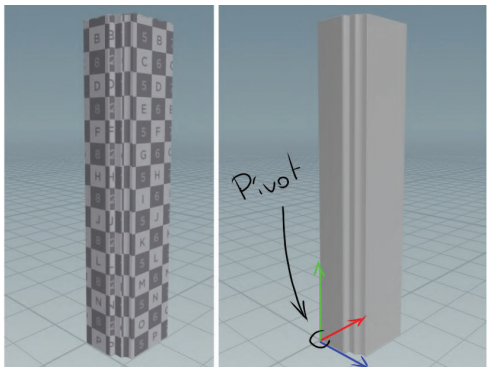


Figure 4.21: Pillar asset's UV's example and pivot

UV unwrapping was done using the same set of tools (nodes).

4.3.6 Walls

How to create a wall asset was the topic left unanswered for a long time, almost until the very last stages of the project were approached.

The original idea was to build walls around doors and windows with extra space left on both sides along the x and y-axis. However, after many hours of drawing over reference images, it became clear that even though this approach

often works for the majority of building kits, this particular project would require some out-of-the-box thinking.

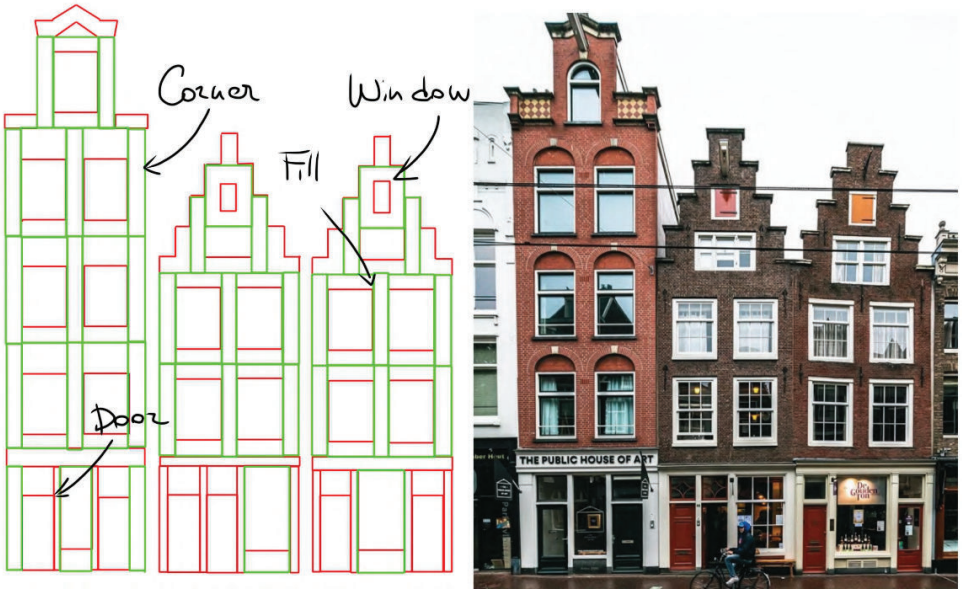


Figure 4.22: Example of several wall assets, that would be needed for the project to create building facades.

To accommodate all the needs of the facades, I have eventually decided to build several walls for each of the four types: corner, fill, window and door (Figure 4.22).

Walls were then created as single-sided pieces from the simple plane geometry. For each of four variants, its geometry was placed on the extremities of the footprint to provide a seamless tiling with other similar assets (Figure 4.23).

Doors and windows were cut from the surface of the piece with a boolean node.

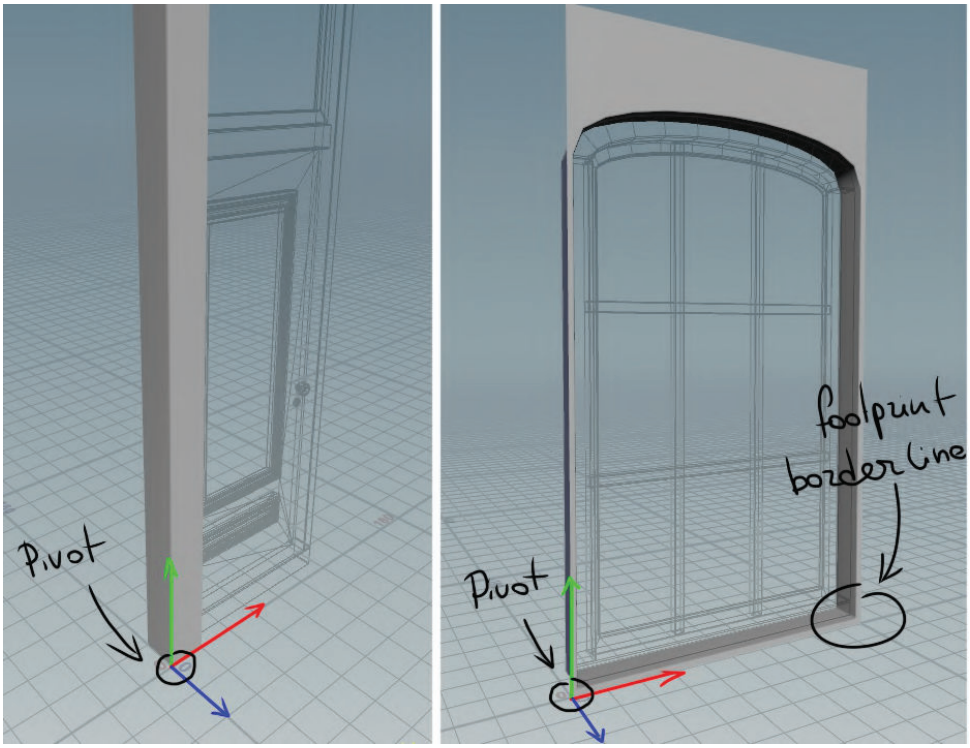


Figure 4.23: Pivot poits positions for some wall pieces

The pivot point was placed on the bottom left corner, with an exception made for a corner piece (Figure 4.23); and finally, all faces of the assets were UV unwrapped (Figure 4.24).

Figure 4.24: Door, window and corner wall assets with checker board textures.

4.3.7 Gables

In general, "*the gable is a section of wall between the edges of a dual-pitched roof*" [34]. It is also one of the most prominent features of the facade. The most common shapes of the Dutch gable include bell, spout, stairs and neck (Figure 4.25).

Figure 4.25: Example of different gable styles.

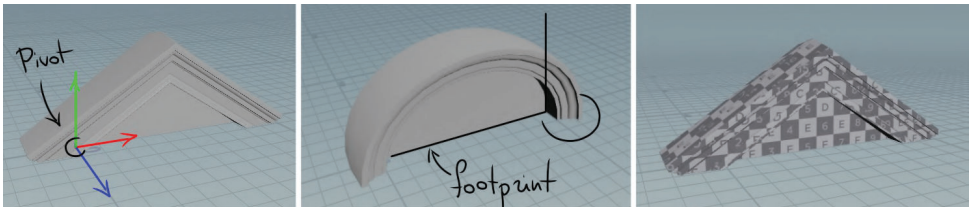
At first, it seemed that all four variants would have to be modelled individually. However, after a closer look, it became clear, that some shapes, such as neck and stairs could be assembled from assets that are already a part of the modular kit (walls, cornices and pillars).

Gables were then further divided into two parts: a side frame and a "hat", to provide more flexibility and variety without having to create countless unique assets.

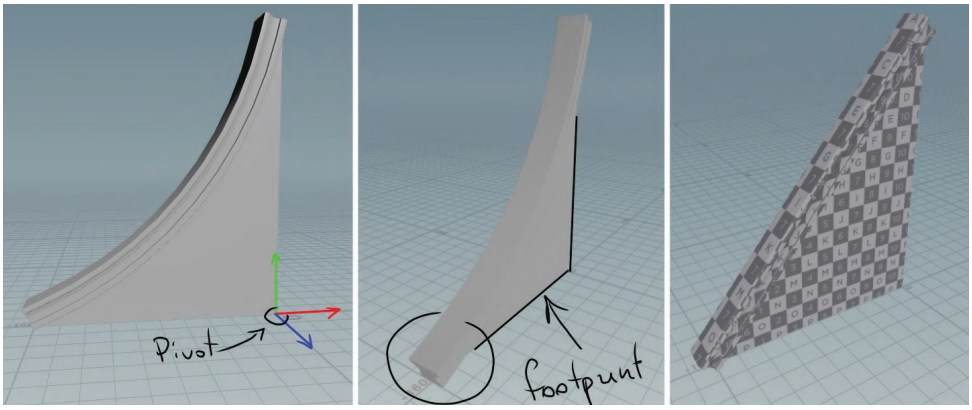
All parts of the gable were modelled procedurally from a simple line and a plane. The workflow was very similar to rails, cornices and pillars.

What stood out, however, was the placement of the pivot point and how assets geometry would fit into the footprint.

First, the pivot - its position was set at the bottom left corner of the plane below the frame for the "hat" (Figure 4.26a), and at the right bottom corner for the side piece (Figure 4.26b).



(a) "Hat"



(b) Side frame

Figure 4.26: Example of a pivot point position, footprint and UV maps for each part of the gable's comonents

What was uncommon for the footprint, is that in contrast to other assets, all components of the gable were created inside this imaginary grid space with both front and backside stretched equally to its extremities.

4.4 Assembling a Scene in Unreal Engine

4.4.1 First Export (and challenges that came with it)

After all procedural assets were finished, the next question was how to export them into the Unreal Engine, as all the work was done in the Apprentice version of Houdini, which came without any support for exporting .fbx files or digital assets.

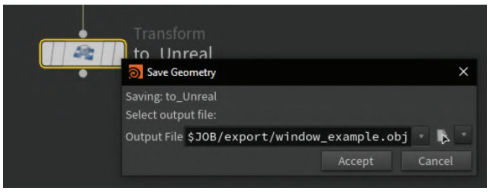


Figure 4.27: The transform node that rotates a model 90 degrees along the x-axis, followed by the Save Geometry panel

Hopefully, it was possible to at least save geometry from a selected node on the disc in the .obj format (Figure 4.27).

This method, however, had some significant flaws, that took a reasonable amount of time and additional research to solve.

First was the difference in the coordinate system used in Houdini (Y-up right-handed) and Unreal Engine (Z-up left-handed). Houdini Engine would

typically take care of the conversion between these systems; however, since none was available at the time, another approach was required.

After several trials and errors, I have found, that translation node with 90 degrees rotation along the x-axis was enought to successfully solve the problem (Figure 4.28).

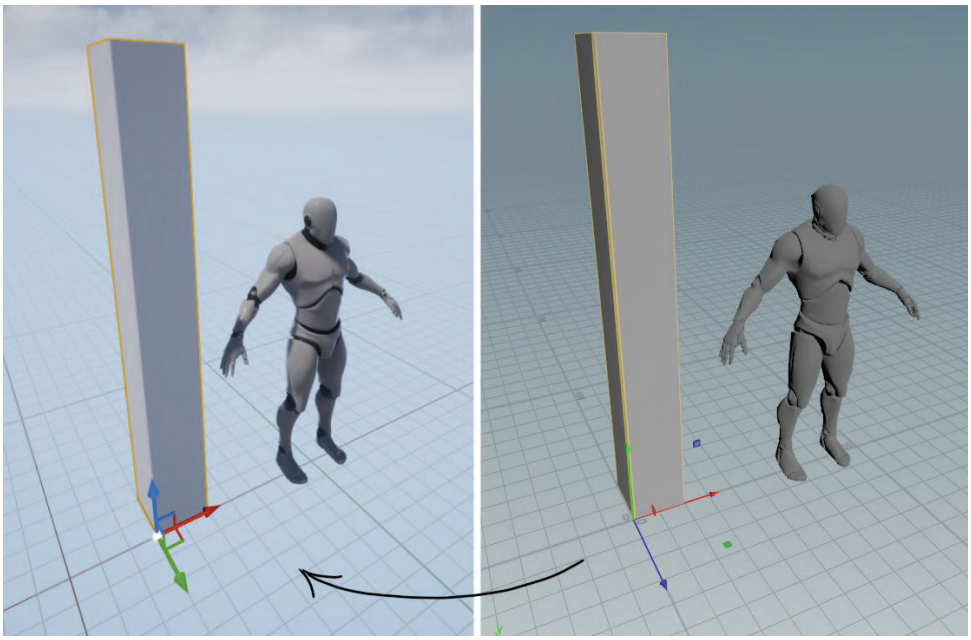


Figure 4.28: Comparison between a wall corner asset inside Houdini (left) and after it was imported into the Unreal Engine with 90 degrees rotation along the x-axis (right).

When several assets were finally imported into the scene, I have noticed that some of these objects were divided into multiple parts inside the content browser. As it was discovered later, the cause for this problem was various geometry groups that were used inside each node tree in Houdini to distinguish between different parts of the model (Figure 4.29). To clean-up all undesirable selections and groups a "delete" node was added to each graph.

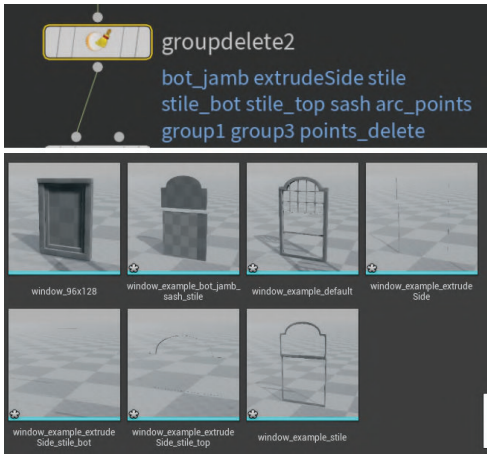


Figure 4.29: Example of a group delete node that helped to fix the problem with an asset being divided into several parts inside the Content Browser

While these were just some minor obstacles, the real challenge came later, after the first door asset was imported.

As seen in the Figure 4.30, all faces of the model where rendered except for one. That issue with the face being invisible was not present in the Houdini, which complicated the debugging

process even more.

Eventually, the case of the disappearing geometry was found to be caused by the boolean node, which by default detriangulates all faces, regardless of whether they were modified during the process or not.

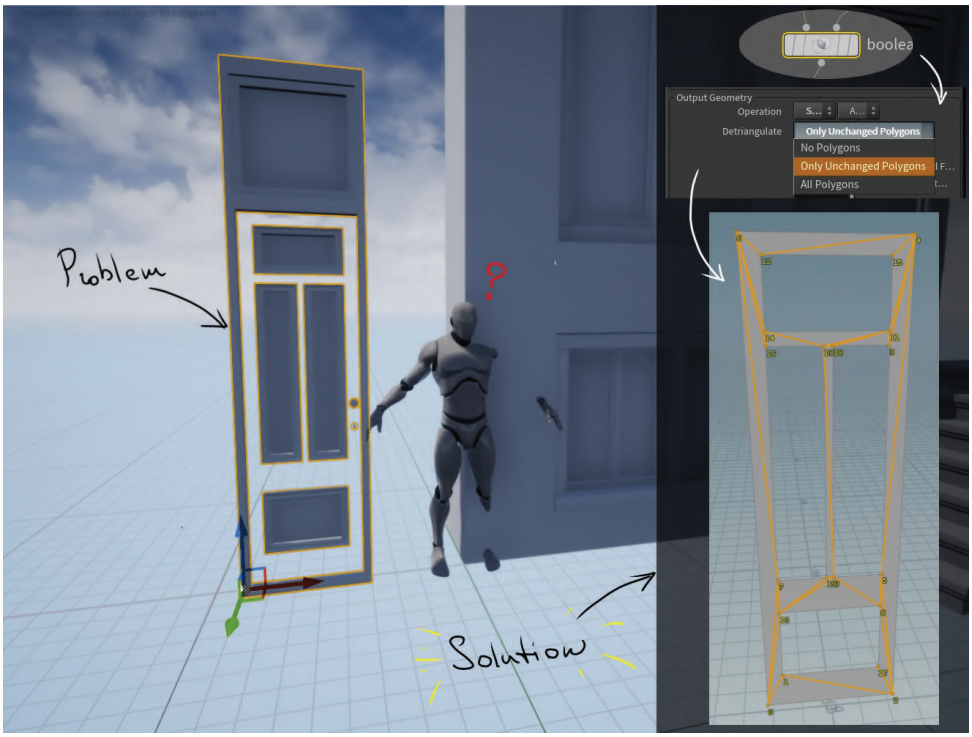


Figure 4.30: Example of the door object in Unreal Engine with invisible geometry on the right and the possible solution with setting different parameter for detriangulation performed by the boolean node.

It was possible to mitigate this problem by disabling the detriangulation option (Figure 4.30) and creating a custom node tree, that would fix issues with the output geometry that might occur after the boolean node.

4.4.2 Assembling a Test Scene

Before creating a final modular kit, it was important to test how each procedurally generated model would behave in an actual game environment.

For this purpose,I have decided to build a simple test scene with a single building in it.

The overal process of assembling the facade in Unreal Engine was perhaps the smoothest and the most enjoyable part of the project.

Due to the modular nature of assets combined with procedural modelling techniques, fixing various problems with wrong pivot point's position or

object's dimensions, even production of any additional models and assembling them in the scene would take the minimum time possible, in some case even less than a minute (!).

Figure 4.31: (from right to left) A reference image for the building, the final model and its front view representation on the grid inside Unreal Engine. Highlighted in orange are unique assets, while the rest of purple-ish geometry are their instances.

As shown in Figure 4.31, the final result required a total of 76 assets, 32 of which were unique, while the rest 34 where their instances.

Compared to the reference image, the final model of the facade did lack some detail, primarily due to the fact that no textures were provided for these assets, as their creation was out of the scope of this project.

The overall visual quality of the building's facade, however, was satisfying, as it was possible to achieve the utmost resemblance between the 3D model and its real-world counterpart.

Chapter 5

Results

5.1 Procedural Systems for Modular Assets

This section will provide a short description for each modular asset in terms of statistics regarding the number of nodes inside each procedural graph system and how many modifiable parameters each of them have.

5.1.1 Windows

The procedural window asset was created with a total number of approximately 443 nodes (Figure 5.2) and 51 parameters (Figure **??**).

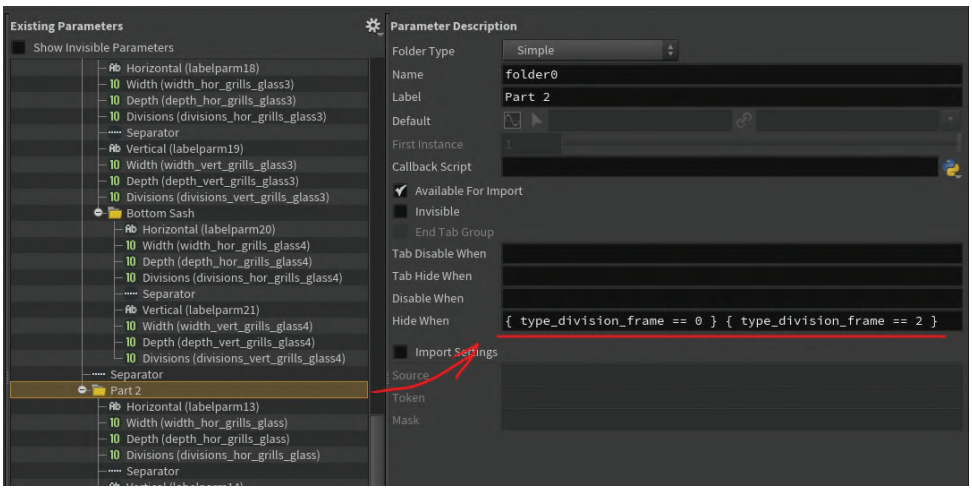


Figure 5.1: Example of the parameter description menu, where rules were set for some parameters to ensure that they would be rendered only when needed.

The interface of the node, which holds all these parameters, was implemented in the way that is hoped to be intuitive and easy to use (Figure 5.1).

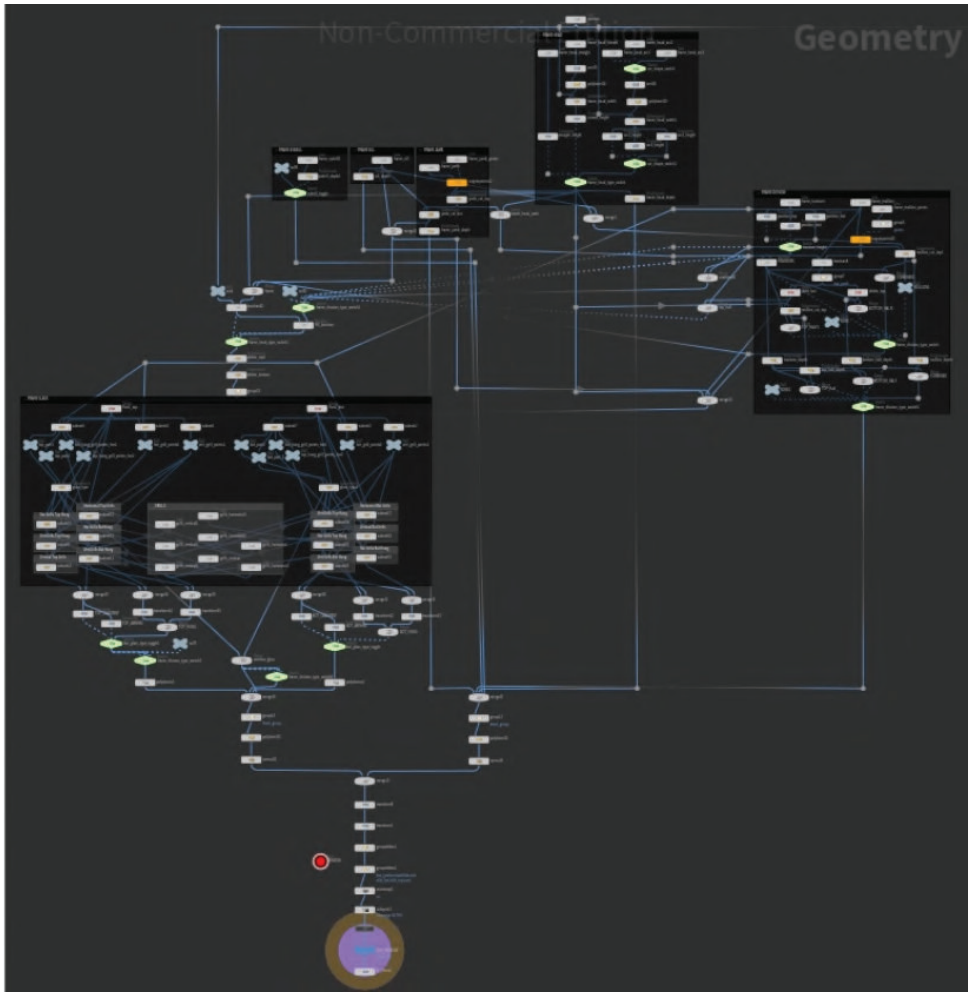


Figure 5.2: Example of the node-graph for the window model.

A network box was put around each section of the graph responsible for creating a particular part of the model (Figure 5.3).

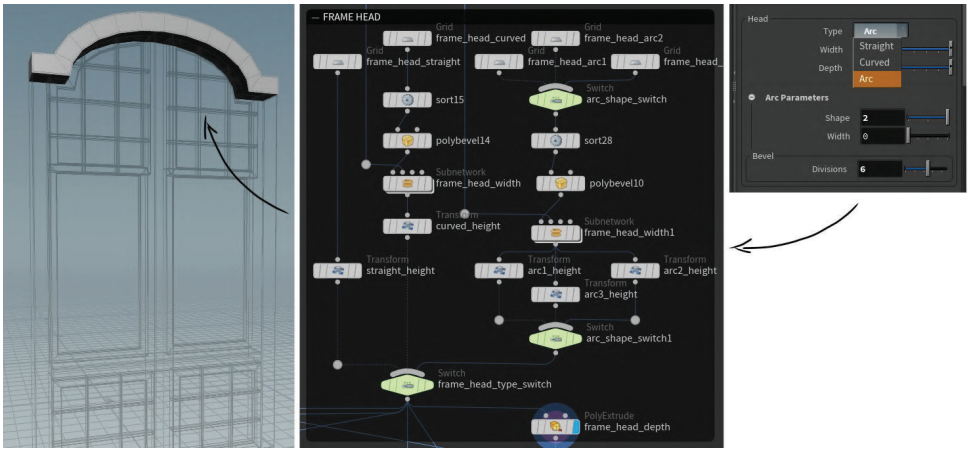


Figure 5.3: Example of the paraeters menu, network box and a frame head model of the window.

The naming of these boxes, as well as some nodes inside of them, was set to

correspond with what user will see in the parameters menu.

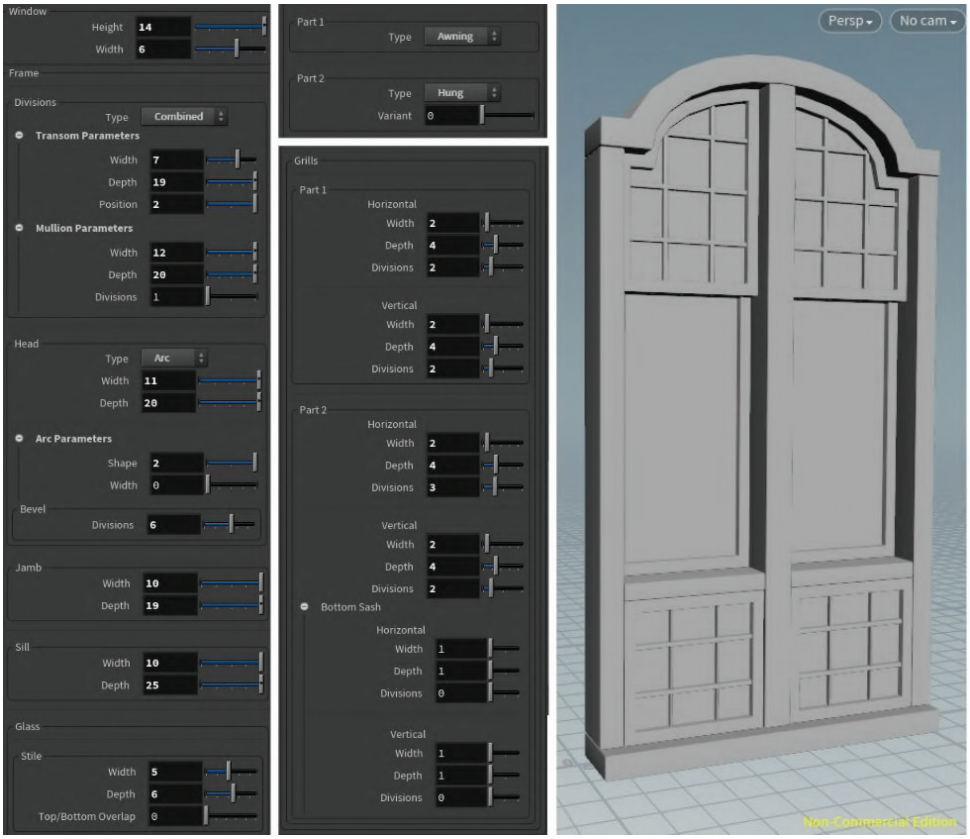


Figure 5.4: Example of the parameters interface with set values on the left for the window model on the right.

With only a couple of clicks and in less than a minute, it was possible to create a couple of windows according to a randomly selected reference image (Figure 5.5).

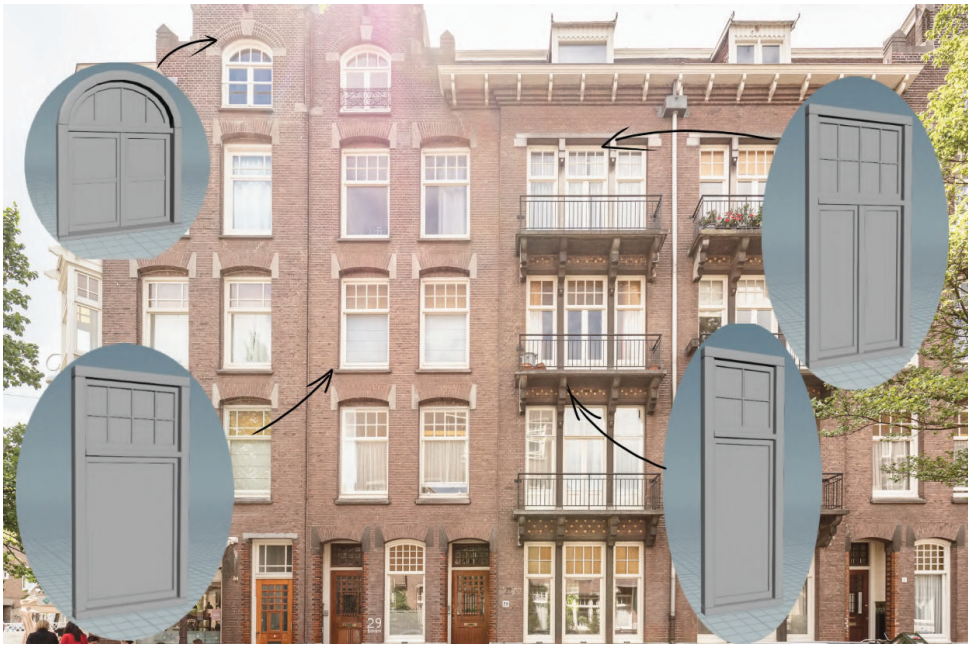


Figure 5.5: Some procedurally generated modullar windows.

5.1.2 Doors

Around 526 nodes were used to create the door asset, the number significantly larger than the one needed for the window (Figure 5.6).

A total of 71 parameters were added to control each essential part of the model (Figure 5.8).

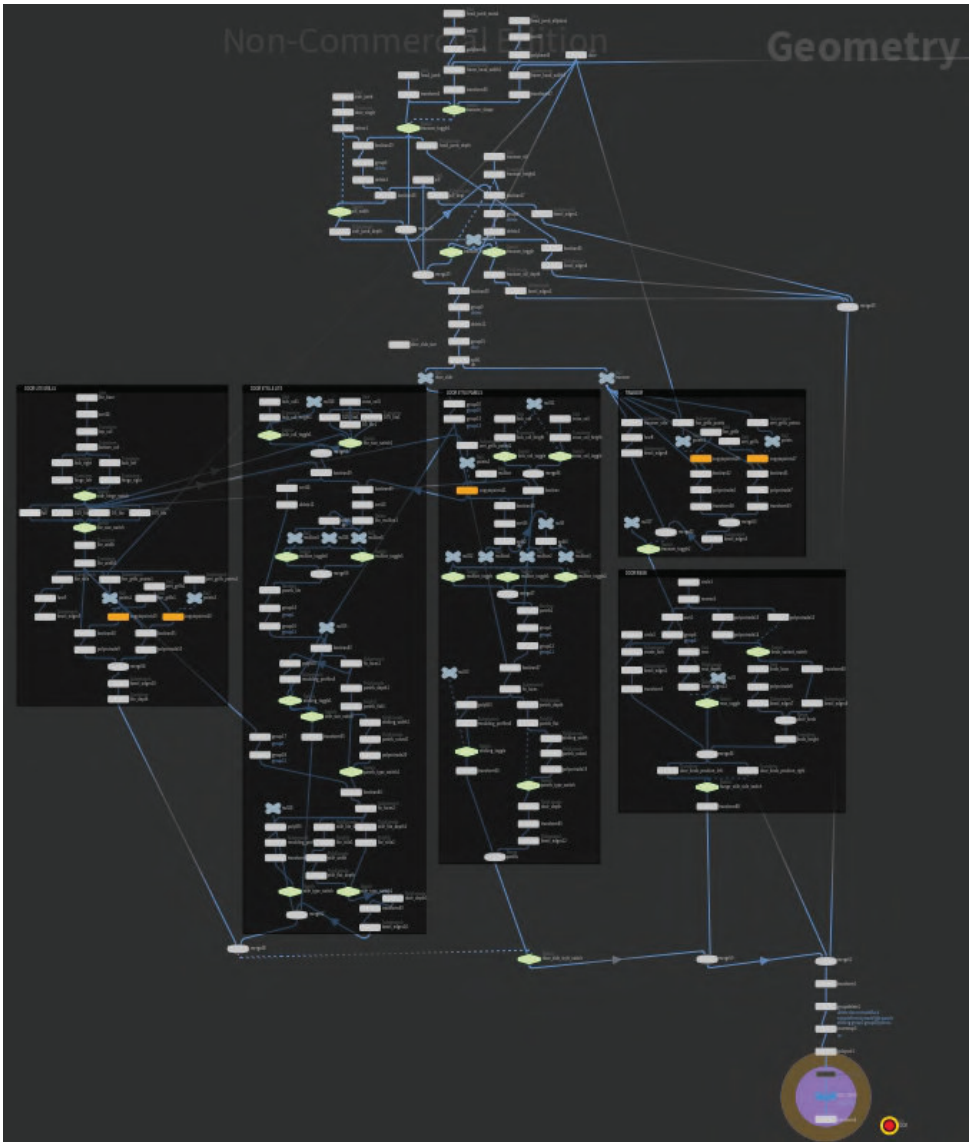


Figure 5.6: Example of the node-grap used for the modular door asset

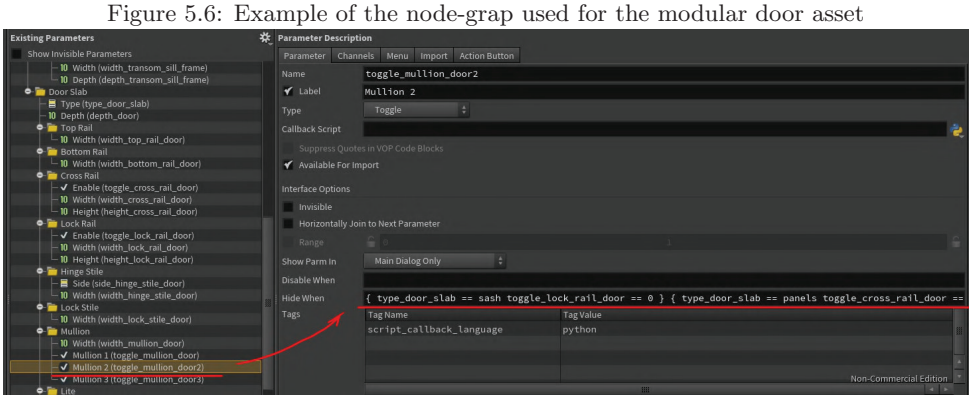


Figure 5.7: Example of the parameter description menu, where rules were set for some parameters to ensure that they would be rendered only when needed

The interface of the parameters menu was again set concerning the needs of the possible end-user (Figure 5.7).

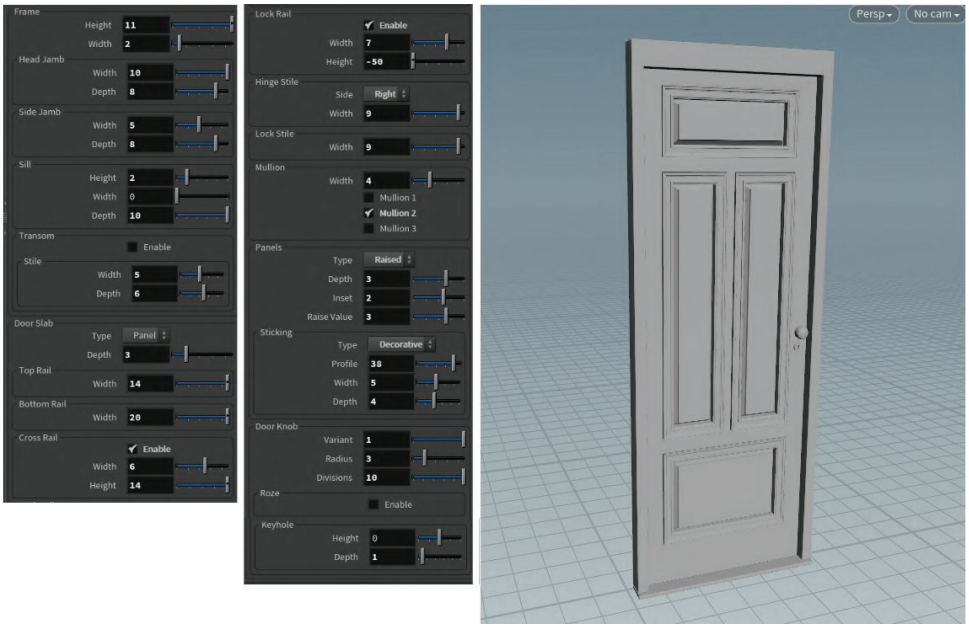


Figure 5.8: On the left are parameters with set values for the door model on the right

Several network boxes were created around each section of the graph, which was responsible for creating a particular part of the door model (Figure 5.9). The naming of these boxes, as well as some nodes inside of them, was again set to correspond with what user would see in the parameters menu.

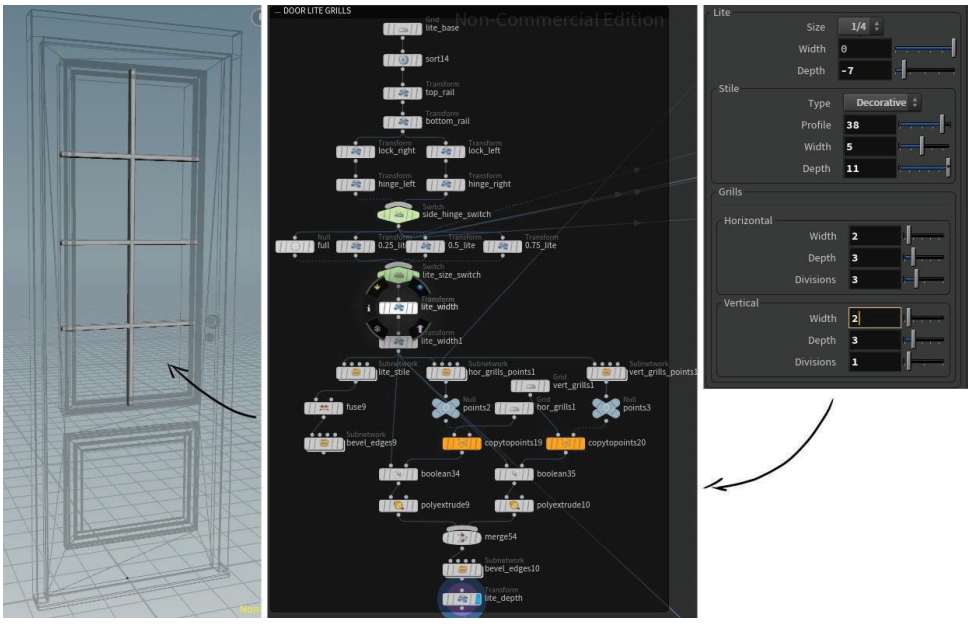


Figure 5.9: Example of the paraeters menu, network box and a grills for the door lite

The result from the procedural door system was not as versatile as from the window, as the graph and the model itself was already quite complicated it was decided to keep the possible variations to the most necessary minimum (Figure 5.10). Furthermore, since doors only occur once, maximum twice on each building, the repetition between their geometry would not be as

noticeable as if it was on windows.

Figure 5.10: Some examples of door assets created according to the reference images

5.1.3 Stairs

Stairs were created with only 83 different nodes and required 11 parameters to define the specific shape, size and style of the moulding, steps and the platform (Figure 5.11).

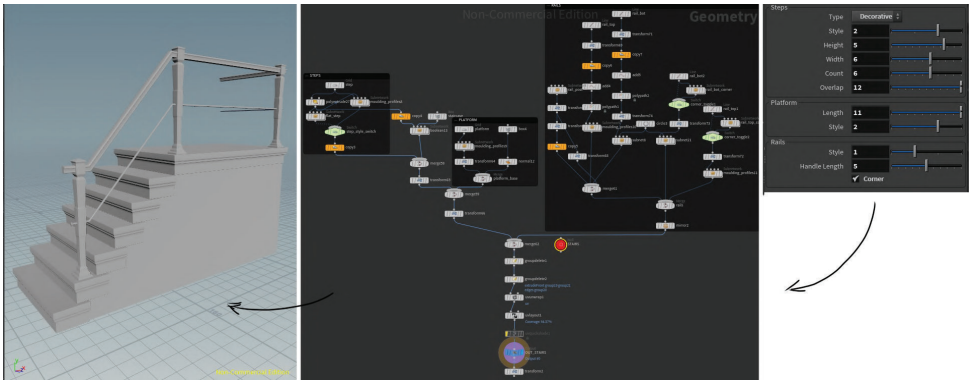


Figure 5.11: Example of the modular stairs asset, procedural graph and parameters interface for this model

5.1.4 Cornices

77 nodes and 6 parameters were used to create three types of cornice pieces: cap, corner and simple tilable rim (Figure 5.12).

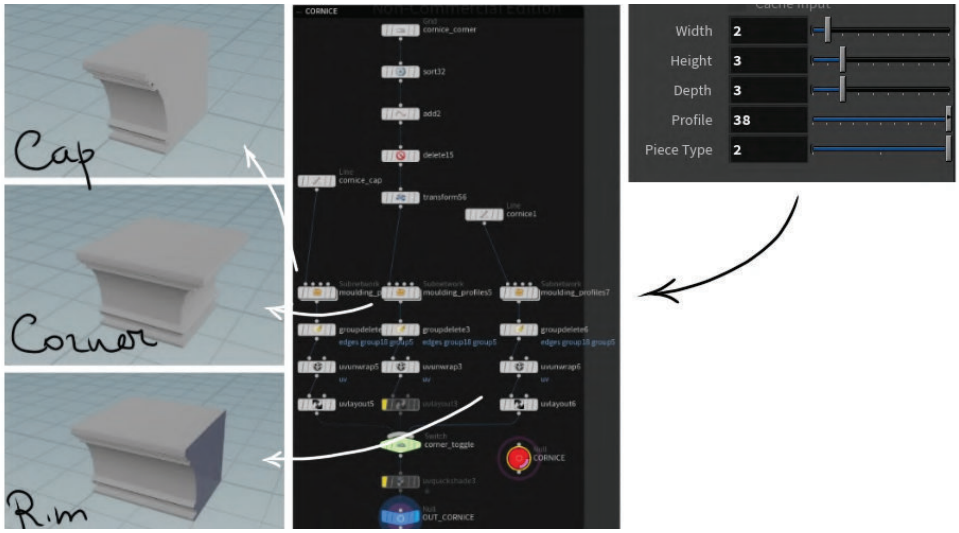


Figure 5.12: Modular cornice asset types, graph and parameters

5.1.5 Walls

For the wall model, the total of 26 nodes and five parameters were used.

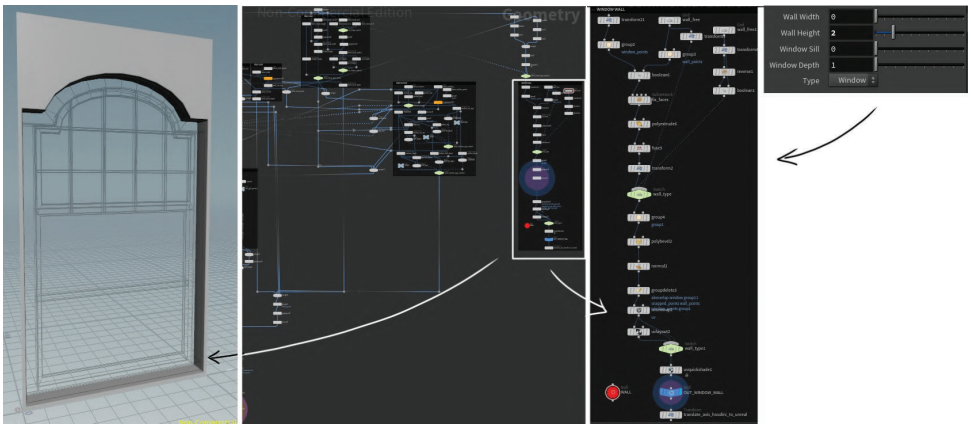


Figure 5.13: Node graph and parameters menu for the modular wall asset inside the window geometry node.

In contrast to all the previous assets, the node graph for the wall was put directly into the window geometry node (Figure 5.13) and then copied to the door. This step ensured that for each window or door created there would be no need to adjust the wall's parameters so that the newly created geometry would still fit into the cut.

5.1.6 Pillars

Pillars had roughly 24 nodes and five standard parameters for the height, width depth and the shape of the model (Figure 5.14).

Figure 5.14: Procedural graph for a pillar asset and its parameters, with example of different symmetry variants and their effects on the object's geometry

5.1.7 Gables

Each part of a gable was created separately without any dependencies that would affect their dimensions, shape or any other parameters. However, both a "hat" and a side frame of the gable were stored inside the same geometry node and shared the same parameters menu.

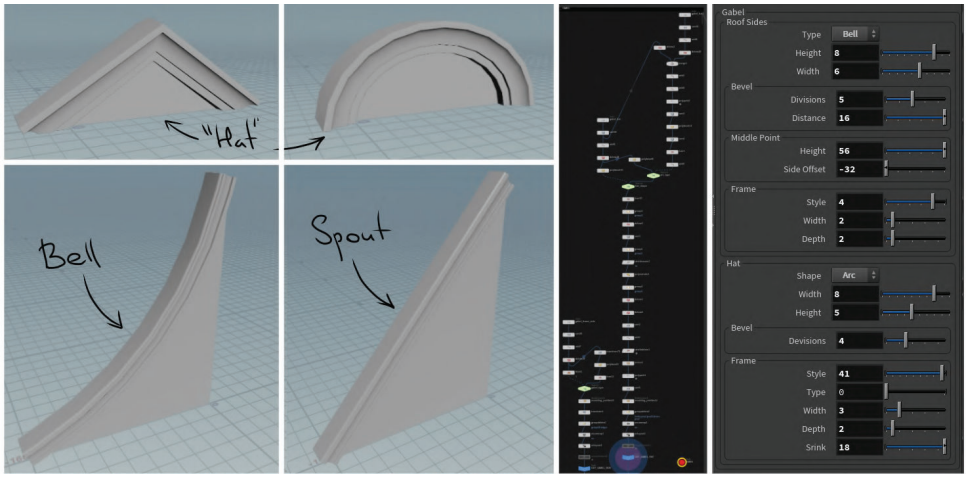


Figure 5.15: Example of two different shapes for a "hat" and side frame of a gable with a node graph and parameters for both assets

To create both assets a total of 106 nodes was required, and 19 different parameters (Figure 5.15).

5.2 Workflow Overview

As hoped, the practical part of the project has helped to gain a more in-depth insight into the advantages and disadvantages of both modular and procedural workflow.

Modular techniques adopted early on during the asset production process

have provided a consistent and fast level design workflow.

The procedural approach used for the creation of these assets, on the other hand, have provided both negative and positive results.

The tools required for this technique are rather complicated, especially for a beginner; therefore, the process of modelling of some of the assets took an enormous amount of time. However, compared to polygonal modelling, which would provide a faster production workflow, once these procedural systems where finished, it was possible to generate any amount of assets as well as their variants in just a couple of minutes, which would be almost impossible to achieve with any other modelling technique.

5.3 Additional Use-Case for Procedural Systems

Another possible use-case for created procedural systems might be the random generation of various assets or even the composition of whole buildings inside Houdini.

For this purpose, however, a separate node-graph would be needed, that would take care of the building process according to the set of the rules defined for each facade.

Since, all modular assets, as well as their node-graphs, were carefully planned and implemented so that each system would be able to provide the most reliable result possible, there should not be any significant amount of extra work that would be required to adopt them into a fully procedural workflow.

Chapter 6

Conclusion

In the first part of this thesis, we have discussed different modelling techniques, addressed some of their most prominent features and took a look at multiple 3D modelling software.

Then, we have conducted thorough research on the concept of modular design in computer games. Through the examination of different approaches used by industry professionals, we have identified and described some of the essential aspects of the modular workflow and asset production.

In the practical part of this project, we have combined everything that was discussed about modular design with the procedural modelling techniques.

As a result, we were able to create procedural systems that can produce various modular assets according to the parameters set by the user.

This modular-procedural approach guaranteed both the rapid level design iterations, thanks to the modular nature of the kit's components, and the efficient asset production workflow that can generate a large variety of objects under a short amount of time.

Bibliography

[1] Anlauf, Tyler: Complex Modular Architecture Environment in UE4. In: *80lv* (2018), December

[2] Autodesk Inc.: *NURBS overview*. Maya Documentation. 2016. – URL https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2016/ENU/Maya/files/GUID-5EC05798-3F28-4AD2-8154-36BC444A4DC9-htm.html

[3] Autodesk Inc.: *Pivot Point*. 3Ds Max Documentation. 2017. – URL https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2017/ENU/3DSMax/files/GUID-A4853FF5-8657-4ED0-92E0-98F4CFB445A3-htm.html

[4] Autodesk Inc.: *Polygonal Modeling*. Maya Documentation. 2020. – URL https://help.autodesk.com/view/MAYAUL/2020/ENU/?guid=GUID-7941F97A-36E8-47FE-95D1-71412A3B3017

[5] Bennett, Bob: Evolution of the 3D Industry. In: *CGW* 29 (2006), December

[6] Blackwell, Alan ; Rodden, Kerry: Sketchpad: A man-machine graphical communication system / University of Cambridge. September 2003 (574). – Forschungsbericht

[7] Blender: *Adaptive Sculpting*. Bender Manual. – URL https://docs.blender.org/manual/en/latest/sculpt_paint/sculpting/adaptive.html

[8] Bourke, Paul: *Implicit surfaces*. June 1997. – URL http://paulbourke.net/geometry/implicitsurf/

[9] Burgess, Joel: *Skyrim's Modular Approach to Level Design*. Gamasutra blog. 2013. – URL https://www.gamasutra.com/blogs/JoelBurgess/20130501/191514/Skyrims_Modular_Approach_to_Level_Design.php

[10] Burgess, Joel: Building Huge Open Worlds: Modularity, Kits & Art Fatigue. In: *80lv* (2018), August. – URL https://80.lv/articles/building-huge-open-worlds-modularity-kits-art-fatigue/

[11] Burgess, Joel ; Purkeypile, Nathan: Fallout 4's Modular Level Design. In: *Game Developers Conference*, March 2016

[12] Carucci, Francesco: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Kap. Inside Geometry Instancing, Addison-Wesley Professional, 2005. – ISBN 0321335597

[13] College of Earth and Mineral Sciences at The Pennsylvania State University (Veranst.): *The Concept of Procedural Modeling*. 2018. – URL https://www.e-education.psu.edu/geogvr/node/534

[14] Digital School Technical Design College: *A History of Computer Graphic Modelling*. – URL https://www.digitalschool.ca/a-history-of-computer-graphic-modeling

[15] Ebert, David S.: Advanced Modeling Techniques for Computer Graphics. In: *ACM Computing Surveys* 28 (1996), 12

[16] Ebert, David S. ; Musgrave, F. K. ; Peachey, Darwyn ; Perlin, Ken ; Worley, Steven: *Texturing & Modeling: A Procedural Approach*. Morgan Kaufmann, 2003 (Morgan Kaufmann series in computer graphics and geometric modeling). – ISBN 9781558608481

[17] Epic Games Inc: *Using Workflow Techniques and Modularity*. Unreal Engine Documentation. 2012. – URL https://docs.unrealengine.com/udk/Two/WorkflowAndModularity.html

[18] Fenech, Braiden: *Approaches to Modular Construction for Real-time Game Environments*, University of Canberra, Diplomarbeit, 2015

[19] Format Team: Our Top 19 3D Modeling Software Picks: Free and Paid. In: *Format Magazine* (2019), March. – URL https://www.format.com/magazine/resources/design/3d-modeling-software

[20] Foundry: *How Wooga Games develops a creative, engaging and rich experience with Modo*. video. December 2019. – URL https://www.youtube.com/watch?v=4Q1ar6rIMAA&feature=youtu.be

[21] Gaget, Lucie: Battle of Software 2020: Mudbox vs ZBrush. In: *Sculpteo* (2019), February

[22] Ganster, Dr. B. ; Klein, Prof. Dr. R.: *Procedural Modeling*

[23] Hansmeyer, Michael: L-Systems in Architecture. In: *Computational Architecture* (2003). – URL http://www.michael-hansmeyer.com/l-systems

[24] inspirationTuts: *3D Modeling Software used for Video Games*. April 2020. – URL https://inspirationtuts.com/2020/04/21/3d-modeling-software-used-for-video-games/

[25] Jarratt, Steve: The best 3D modelling software 2020. In: *CREATIVE BLOQ* (2019), June

[26] Johansson, Isabell: *Defining what is visually dynamic for modular assets in level design*, Luleå University of Technology, Diplomarbeit, 2017

[27] Jones, Javis: *What Are Voxels*. 3DCoat Documentation. August 2019. – URL https://3dcoat.com/manual/sculpt/81-voxels/

[28] Jones, Scott: *Investigation into modular design within computer games.*, Staffordshire University, Diplomarbeit, May 2011

[29] Klafke, Thiago: *Creating Modular Environments in UDK*. Blog. – URL https://www.thiagoklafke.com/tutorials/modular-environments/

[30] Kutz, Sarah: *3D Modeling Software used for Video Games*. October 2018. – URL https://medium.com/imeshup/comparison-the-top-6-most-popular-3d-modeling-soft-1c6a9ed204a4

[31] Mader, Paul: *Creating Modular Game Art For Fast Level Design*. Gamasutra blog. 2005. – URL https://www.gamasutra.com/view/feature/130885/creating_modular_game_art_for_fast_.php

[32] Module: *Cambridge Dictionary*. Cambridge University Press, 2020

[33] Nelson, Shawn: *Photoshop for Games: Creating Art for Console, Mobile, and Social Games*. Kap. Using Photoshop to Create Art for a Mobile Game, New Riders, 2014. – ISBN 978-0-321-99020-4

[34] Nigel: *Canal House Gables*. Amsterdam for Visitors, Blog. – URL https://www.amsterdamforvisitors.com/canal-house-gables/

[35] Opalach, Agata ; Maddock, Steve: An Overview of Implicit Surfaces / Department of Computer Science, The University of Sheffield. May 1995. – Forschungsbericht

[36] Perry, Lee: Modular Level and Component Design: How I Learned to Stop Worrying and Love Making High-Detail Worlds. In: *Game Developer* (2002), November

[37] Pickavance, Mark: (Best 3D modelling software of 2020). In: *TechRadar* (2019), October

[38] Pinto, Helder: *3D Environment Art for Video Games: Artist Panel*. Gnomon on YouTube. 2016. – URL https://www.youtube.com/watch?v=kGm_xhu42tU

[39] Poznanski, Andrzej: Visual Revolution of The Vanishing of Ethan Carter. In: *Ethan Carter, Game Design* (2014), March. – URL http://www.theastronauts.com/2014/03/visual-revolution-vanishing-ethan-carter/

[40] Prusinkiewicz, Przemyslaw ; Lindenmayer, Aristid: *The Algorithmic Beauty of Plants*. Springer-Verlag, 2012 (The Virtual Laboratory). – ISBN 9781461384762

[41] Salmond, Michael: *Video Game Design: Principles and Practices from the Ground Up*. Bloomsbury Publishing, 2017. – ISBN 9781474255455

[42] Selin, Erik: *10 Different types of 3D modeling techniques*. – URL https://artisticrender.com/10-different-types-of-3d-modeling-techniques/

[43] SideFX: *Geometry attributes*. Houdini Documentation. 2019. – URL https://www.sidefx.com/docs/houdini/model/attributes.html

[44] SideFX: *Houdini Documentation*, 2019. – URL https://www.sidefx.com/docs/houdini/basics/intro.html

[45] SideFX Houdini: *2020 Houdini Games Reel*. video. March 2020. – URL https://vimeo.com/397201248

[46] Slick, Justin: 7 Common Modeling Techniques for Film and Games: An introduction to 3D modeling techniques. In: *Lifewire* (2019), October

[47] Slick, Justin: List of Full-Featured 3D Applications. In: *Lifewire* (2019), January

[48] Statham, Nataska: Use of Photogrammetry in Video Games: A Historical Overview. In: *Games and Culture* (2018), July. – URL https://journals.sagepub.com/doi/full/10.1177/1555412018786415#

[49] Treehouse Island, Inc: *Asset Workflow for Game Art: 3D Modeling*. Blog. 2015. – URL https://blog.teamtreehouse.com/asset-workflow-game-art-3d-modeling

[50] Tseng, Mitchell M. ; Wang, Chenjie: *CIRP Encyclopedia of Production Engineering*. Kap. Modular Design, S. 895–897. Berlin, Heidelberg : Springer Berlin Heidelberg, 2014. – ISBN 978-3-642-20617-7

[51] UFO 3D: *Advanced 3D Modelling Techniques: Top 6 For Product Visualization*. blog. – URL https://ufo3d.com/advanced-3d-modelling-techniques-top6

[52] UFO 3D: *History of 3D Modeling: from Euclid to 3D printing*. blog. – URL https://ufo3d.com/history-of-3d-modeling

[53] Unity Technologies: *Asset Workflow*. Unity3d Documentation. 2017. – URL https://docs.unity3d.com/560/Documentation/Manual/AssetWorkflow.html

[54] Unity Technologies: *GPU instancing*. Unity 3d Documentation. 2020. – URL https://docs.unity3d.com/Manual/GPUInstancing.html

[55] Utah teapot: *Wikipedia*. – URL https://en.wikipedia.org/wiki/Utah_teapot

[56] Valve Developer Community: *Decals*. Valve Documentation. 2019. – URL https://developer.valvesoftware.com/wiki/Decals

[57] Wei, Li-Yi: *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems)*. Kap. Tile-Based Texture Mapping, Addison-Wesley Professional, 2005. – ISBN 0321335597