UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
TECNOLOGÍA ESPECÍFICA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

APLICACIÓN DE TÉCNICAS GRÁFICAS PROCEDURALES EN EL DISEÑO DE ENTORNOS PARA ROBÓTICA

Autor: Pedro Miguel Luzón Martínez
Directores: Luis Rodríguez Ruiz
Daniel González Medina

Julio, 2016

Declaración de Autoría

Yo, Pedro Miguel Luzón Martínez con DNI 77578641-D, declaro que soy el único autor del trabajo fin de grado titulado *Aplicación de técnicas gráficas procedurales en el diseño de entornos de oficina para robótica* y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual y que todo el material no original contenido en dicho trabajo está apropiadamente atribuido a sus legítimos autores.

Albacete, a 25 de mayo de 2016

Fdo: Pedro Miguel Luzón Martínez

Summary

This project covers the study applied to robotic simulation tools focused on the application of procedural techniques for graphics to build virtual environments that can be used in robotic applications. To this end, several procedural techniques will be studied in depth in order to choose those that are suitable for the design of a virtual office environment.

The document is structured as shown in the table of contents. It is conceived to be easy to read and understand, even by those who are not familiar with procedural approaches, graphic design or programming. Nevertheless, it will not lack of any necessary detail.

The main goals of this work are:

- Study of procedural techniques
- Design and implementation of a system to generate 3D office environments
- Adaptation to robotic simulation tools

Resumen

Este proyecto se centra en el estudio aplicado a herramientas de simulación de robótica de técnicas procedurales de gráficos para construir entornos virtuales que puedan ser usados en aplicaciones de robótica. Para este fin se estudiarán en profundidad varias técnicas procedurales con objeto de escoger aquellas que se adecuan al diseño de un entorno virtual de oficina.

Este documento está estructurado conforme al índice. Está concebido para ser fácil de leer y entender, incluso para aquellos que no están familiarizados en el enfoque procedural, diseño gráfico o programación. No obstante, no carecerá de cualquier detalle necesario.

Las principales metas de este trabajo son:

- Estudio de técnicas procedurales
- Diseño e implementación de un sistema para generar entornos 3D de oficina
- Adaptación a herramientas de simulación de robótica

Acknowledgements

I would like to give my appreciation to my tutors, for introducing me to this fantastic project and all it involves, and giving me great assistance in everything I needed.

Dedications

To family, friends and partners, who supported me from the other side.

.

TABLE OF CONTENTS

INDEX OF FIGURES

INDEX OF TABLES

CHAPTER 1. INTRODUCTION

1.1 MOTIVATION

3D graphic generation has made a great improvement over these recent years. New technologies, tools and methodologies have emerged to make graphic design easier and faster. Nowadays, it is possible to build realistic virtual environments from real world places.

This project focuses on one of many graphic techniques: procedural generation. This technique is well exploited in game development, and it is becoming more and more popular in the computer graphics community. However, its uses beneath entertaining purposes are not widely explored.

Manual graphics require a person with enough knowledge (commonly an expert designer) to design and/or animate all the objects in a scene. Object and scene replicas need to be crafted by hand, resulting in very limited variations and an enormous amount of time consumption. Procedural techniques allow us to converge the main ideas of the objects into a model that generates infinite copies of it with a rich set of changes and in a lesser amount of time. Manual graphic modelling techniques are not then required.

The author has never been engaged in any previous procedural graphics related-work, but knew some basic concepts of graphic design (from an interface-based usage).

1.2 GOALS

First, the field of procedural generation will be studied, which will serve as an introduction to the context in which this work is. As a result of the previous study, several techniques will be chosen based on the appropriateness for the work developed here. The structure of real office environments will be also studied, to know the world this technique will operate in. Then the work progress will be fully explained, following the points including in the table of contents.

This is intended to generate a small environment that may be a starting point for developing bigger an advanced projects based on procedural approaches.

The following list enumerates the goals of this work:

- Study of procedural techniques
- Scenario definition: an office environment
- Design and implementation of a system to generate virtual 3D offices
- Adaptation and testing in robotic simulation tools

1.3 DOCUMENT STRUCTURE

The document will have the following structure:

- Chapter 2: State of the art: an overview of the main technologies and methods used along the project development, including conceptual aspects such as the typical office elements and distributions.

- Chapter 3: Office generation process: a detailed explanation of all the progress for a correct office environment generation. This includes base models design, materials and textures generation and procedural setting.

- Chapter 4: System structure and results: the system architecture is described, as well as generation and simulation results are provided.

- Chapter 5: Conclusion and future work on this topic.

CHAPTER 2. STATE OF THE ART

This sections contains a brief study of the main techniques, methods and tools available for developing this project. This includes procedural generation, office environments, Blender tool, Python programming language and robotic simulators.

2.1 Procedural generation

This section covers the main aspects of procedural generation. The main concepts are described, as well as some applications in different fields, and some of the most important techniques related to this domain, concluding with the technique chosen for this work.

2.1.1 Definition

So far, the usual way to build a 3D scene is based on the use of professional designers with enough knowledge about this field. The 3D models are designed using an interface, via mouse and keyboard, which takes some time. If we wanted to replicate those objects and modify them in different ways in order to create a whole scene, the designer would have to clone the objects and manually perform those modifications which results in a tiresome and time-consuming task.

For instance, if we wanted to make a forest, we would need the full model of a tree. We have to ask an expert to craft it. Next, to fill the forest, the tree needs to be cloned and modified to end up with a set of different trees for the landscape. Model expert has to do all that work by hand.

But there is a way to create that forest without the assistance of a human modeler, therefore saving a significant amount of time. Trees can be modelled, cloned and variated automatically. This can be achieved by using procedural techniques.

We understand procedural techniques as a set of models, techniques and methods that can be used to generate contents of a computer-generated model or effect. Therefore, elements generated this way do not rely on already defined elements, but generate them on their own.

Procedural techniques give us the advantages of:

- Abstraction: in a procedural approach, rather than explicitly specifying and storing all the complex details of a scene or sequence, we abstract them by constructing a model.
- Storage saving: as the details are no longer explicitly specified but implicit in the procedure, we gain a significant amount of space. The time requirements for specification of details are shifted from the programmer to the computer. This allows us to create inherit multiresolution models and textures that can be adapted to the resolution needed.
- Parametric control: procedural generation allows us to facilitate variability and some degree of control on the final result. It also provides amplification of the modeler efforts; thus a few parameters yield large amounts of detail. This parametric control unburdens the user from the low-level control and specification of detail.
- Flexibility: the designer of the procedures can capture the essence of the object, phenomenon or motion without being constrained by the complex laws of physics. Procedural techniques allow the inclusion in the model of any desired amount of physical accuracy. The designer may produce a wide range of effects, from accurately simulating natural laws to pure artistic effects [Ebert03].

Procedural techniques have been used for many years in computer graphics to produce realistic textures (marble, wood, stone, wallpaper, bricks) and objects (water, smoke, steam, fire, planes, tribbles). They are cost-effective alternative to physical simulation [Ebert03].

2.1.2 History and applications of procedural generation

When the first specular reflection models were being formulated, Catmull (1974) generated the first textured computer graphics images, represented as parametric patches [Ebert03].

Procedural geometry generation's first steps were taken in the industry of videogames. Rougelike games, such as *Beneath Apple Manor* (Don Worth, 1978). These kind of games used procedural generation to construct dungeons for ASCII based systems. In Don Worth's game, the player could select the number of rooms per level and the difficulty and the computer generated all dungeons randomly, including treasures and enemies all over the place [TCA]. Figure 2.1A shows a randomly generated dungeon for this game.

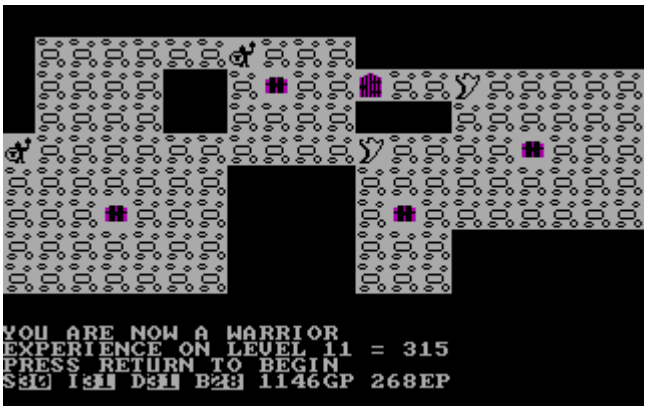


Figure 2.1.A: Randomly generated dungeon on Beneath Apple Manor [TCA]

These dungeons were produced on-the-fly, as there was not enough space to store.

The first videogame to fully generate an artificial world is *Akalabeth: World of Doom* (Richard Garriott, 1979). A seed number was introduced at the beginning of the game and determined almost everything of the gameplay, including the dungeon maps. This was all made in a BASIC code of only 22 Kb [TDA].

.kkrieger (German demogroup, 2004) is a game which makes extensive use of procedural generation methods to make textures. The game only generates one kind of texture, and makes variations to it for every other texture of the game. Thus the history data and the generator code were the only necessary elements to produce them. Therefore, the game only used 97,280 bytes of space in contrast with other games of the same year, that utilized over 2 GB. The *Borderlands* series (Gearbox Software, 2009) use procedural generation to generate the game's weapons, randomly mixing and matching all available gun parts to create an overwhelming number of unique weapons [WSJ].

As a nowadays procedural videogame reference, we cannot forget *Minecraft* (Markus Persson, 2011). Procedural techniques generate really massive worlds. It specifically uses *Perlin Noise* [Ebert03]. It starts out on a very broad level, painting a basic topographic map, and adds noise through finer terrain details like lakes, shrubbery and animals [Engadget]. The amazing results are in Figure 2.1B.

There are several frameworks and tools available to generate procedural content nowadays. One example of this is *Speedtree* (Interactive Data Visualization, Inc., 2002), a set of vegetation programming and modeling products that generates virtual foliage for animations, architecture and real-time simulations [Speedtree].

As in videogames, procedural generation is often used in films to rapidly create visual interesting and accurate spaces.

The Lord of the Rings films (Peter Jackson, 2001-2003) used procedural methods to produce a huge amount of soldiers in their battles. This software's name is *MASSIVE* (stands for *Multiple Agent Simulation System in Virtual Environment*), a tool mostly used for film and television, developed by Stephen Regelous. It has the ability to quickly and easily create thousands of agents that all act as individuals. Through the use of fuzzy logic, the software enables every agent to respond individually to its surroundings, including other agents. The pack also includes cloth simulation, rigid body dynamics and GPU based hardware rendering [Massive]. *World War Z* (Marc Foster, 2013) also took advantage of this tool, and Figure 2.1C shows a notable effect of a crowded city square.

Figure 2.1.B: Minecraft world generation results [TMWS]

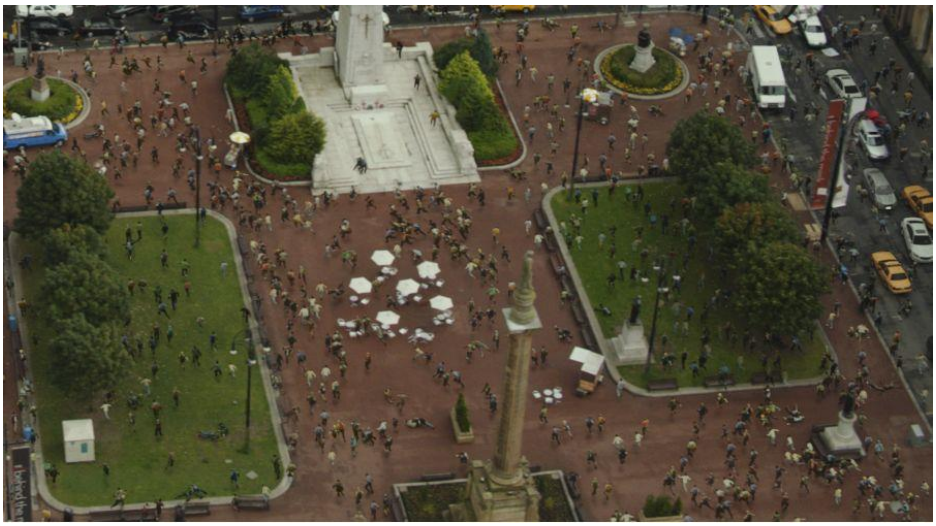


Figure 2.1.C: Crowded square procedurally generated in World War Z [Massive]

2.1.3 Study of procedural techniques

The following sections briefly explains three of the main techniques used for procedural generation: Fractals, *L-Systems* and Perlin Noise.

Fractals

Fractals are irregular and fragmented objects which forms consist of smaller copies of themselves. Thus, they have infinite level of detail. Fractals precursor is the French mathematician Benoit B. Mandelbrot [Aristid].

A good example of a fractal object is the *snowflake curve* proposed in 1905 by von Koch (Figure 2.1D). Mandelbrot restates this construction as follows:

"*One begins with two shapes, an initiator and a generator. The latter is an oriented broken line made up of N equal sides of length r. This each stage of the construction begins with a broken line and consists in replacing each straight interval with a copy of the generator, reduced and displaced so as to have the same end points as those of the interval being replaced*." [Aristid]

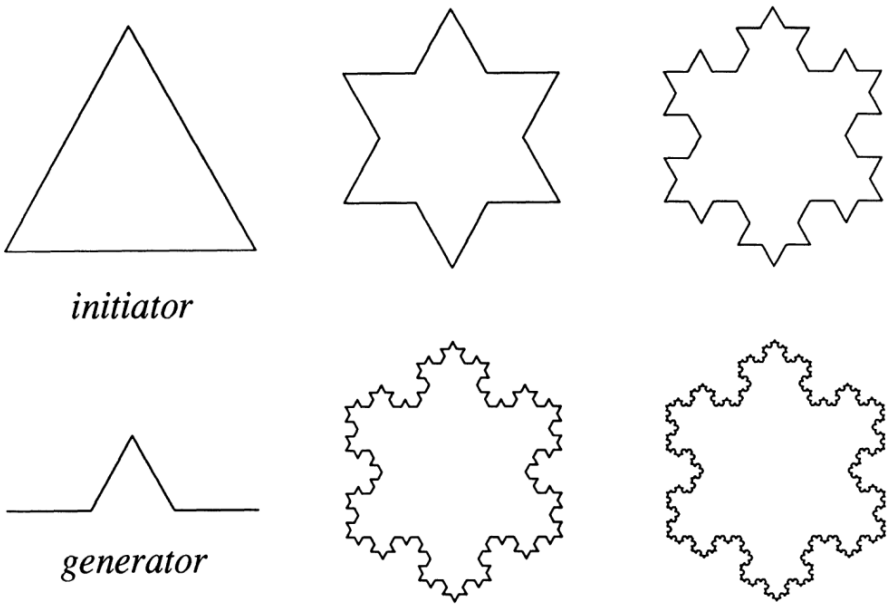


Figure 2.1.D: Snowflake curve construction [Aristid]

One use for this technique is to generate terrain and landforms, as many natural phenomena exhibit some form of statistical similarity that can be modelled by fractal means. Variations in surface texture provide important visual cues to the orientation and slopes of surfaces, and the use of fractal patterns can help create natural looking visual effects. [Lewis] Figure 2.1.E shows a mountainous landscape which is generated using fractals.

Figure 2.1.E: Fractal landscape [Wikipedia]

L-Systems

In 1968, a biologist, Aristid Lindenmayer, introduced a new type of string-rewriting mechanism (grammar), called *L-Systems* [Aristid]. The difference between formal grammars defined by Chomsky in 1950 and *L-Systems* is based on the method of applying productions. Chomsky grammars apply productions sequentially, whereas *L-Systems* do it simultaneously, replacing all the symbols of a given string. Productions are intended to capture cell divisions in multicellular organisms [Aristid].

Formerly, an *L-System* is defined as a tuple $G = (N, \omega, P)$ where:

- N is a set of nonterminal symbols.
- ω is the initial symbol, also called grammar axiom.
- P is the rules or productions set. They specify the way of deriving each symbol of N. Each rule is composed of two parts: a left-handed side called the predecessor and a right-handed side call the successor.

Different strings can be generated from ω applying rules defined in P to symbols in N.

Each generated letter of that string is then mapped into one geometry action, so the sequence within a string can create geometry. Turtle graphics approach defined by Seymour Papert in 1960 clarifies this idea.

L-System adopted the turtle graphics concept for geometry generation. This graphics approach consists in a *turtle* with a determined initial position and orientation. This *turtle* is commanded to move according to Table 2.1.A symbols. Wherever the *turtle* moves, a straight line is drawn following its path, as Figure 2.1.F illustrates [Aristid].

Table 2.1.A: Turtle graphics symbols and commands [Aristid]

SYMBOL	**COMMAND**
F	Move forward a step of length d
+	Turn left by angle ð
-	Turn right by angle ð

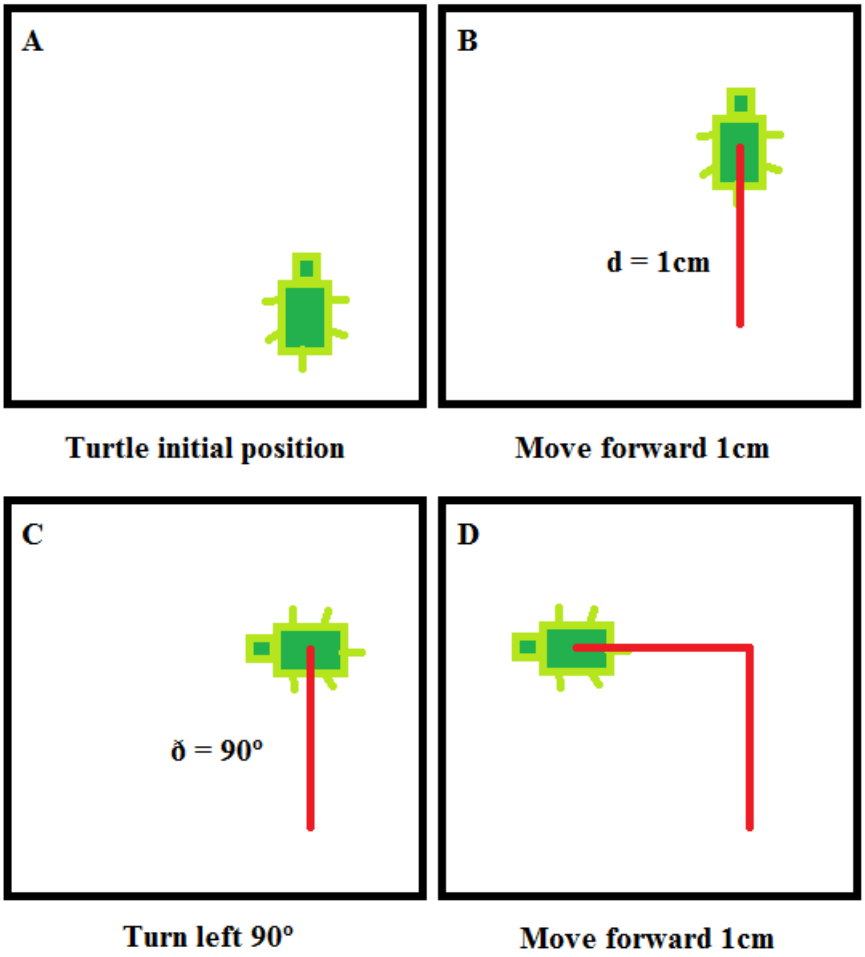


Figure 2.1.F: Turtle drawing sequence

Therefore, the sequence F+F+F+F will make the *turtle* paint a square of d units of side. The usual values for d and ð are 1cm and 90º, as this approach creates sharp-edged figures.

The level of detail of the figures usually depends on the number n of derivations applied to the grammar. More complex grammars and enough number of derivations can spawn impressive figures as Figure 2.1.G shows.

$n = 4,\ \delta = 90^\circ$

$F \rightarrow FF\text{-}F\text{-}F\text{-}F\text{-}F\text{-}F\text{+}F$

$n = 4,\ \delta = 90^\circ$

$F \rightarrow FF\text{-}F\text{-}F\text{-}F\text{-}FF$

Figure 2.1.G: Koch curves [Aristid]

Perlin Noise

Perlin Noise [Ebert03] is a kind gradient noise developed by Ken Perlin in 1983. It is used to increase the appearance of realism in computer graphics. The function has a pseudo-random appearance, but all the details it produces are the same size. Therefore, a great variety of procedural textures and other elements can be obtained. This technique awarded him an Oscar prize from the Academy of Motion Picture Arts and Sciences in 1997.

The algorithm defines an n-dimensional grid. At each point on the grid (node) assign a random gradient vector of unit length in n dimensions. Computation of the (pseudo-) random gradients in one and two dimensions is trivial using a random number generator.

Given an n-dimensional argument for the noise function, the next step in the algorithm is to determine into which grid cell the given point falls. For each corner node of that cell, the distance vector between the point and the node is determined. The dot product between the gradient vector at the node and the distance vector is then computed.

The final step is an interpolation between the second dot products computed at the nodes of the cell containing the argument point. This has the consequence that the noise function returns 0 when evaluated at the grid nodes themselves.

Interpolation is performed using a function that has zero first derivative (and possibly also second derivative) at the second grid nodes. This has the effect that the gradient of the resulting noise function at each grid node coincides with the precomputed random gradient vector there.

Figure 2.1.H shows a sinuous terrain generated using *Perlin Noise*:

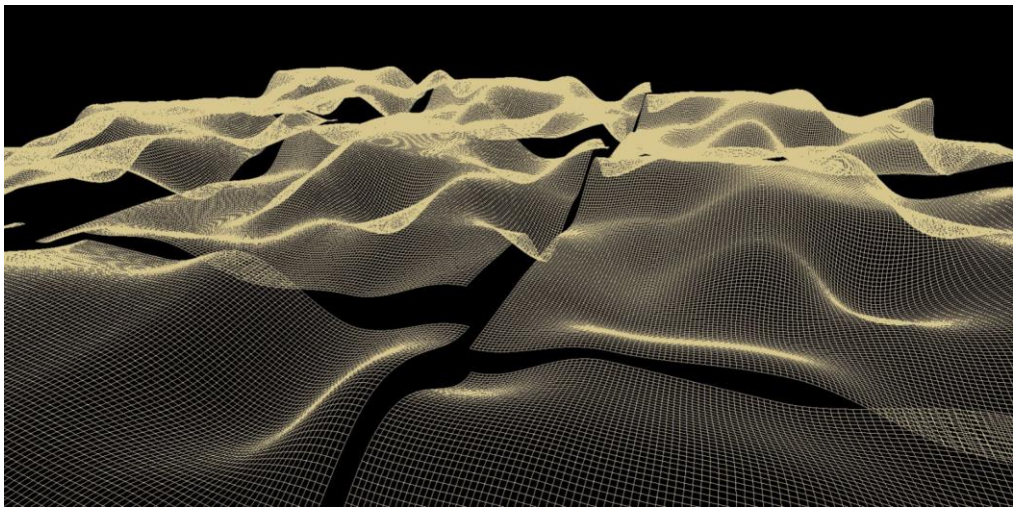


Figure 2.1.H: *Perlin Noise* generated terrain [StackOverflow]

Voronoi Diagrams

Voronoi Diagrams' origin dates back to the 17th century. R. Descartes, in his book *Principia Philosophiae* (1644), claimed that the solar system consists of vortices. He decomposed space into convex regions around fixed stars. Formally speaking, this space P has a set S of fixed stars p, each of them conforming a region. Points x are distributed along P. The region of one single star p consists of all points x for which the influence of p is the strongest over all sets in S. Guerogui Voronoi [Voronoi] used this definition for the study of quadratic forms, where influence is measured by the Euclidean distance.

Voronoi Diagrams [Voronoi] (also called *Dirichlet Tessellation*) are geometric structures that let a Euclidean plane be divided in segments. It uses interpolation joining points between them painting union segment bisections. The intersection of these bisection determine a series of polygons in a bi-dimensional space around a control point set. Each polygon perimeter is equally distant to its neighbors. The most common influence measurement distance is Euclidean distance, but Manhattan distance is also used. Figure 2.1.I shows a diagram using Euclidean distance as influence estimator.

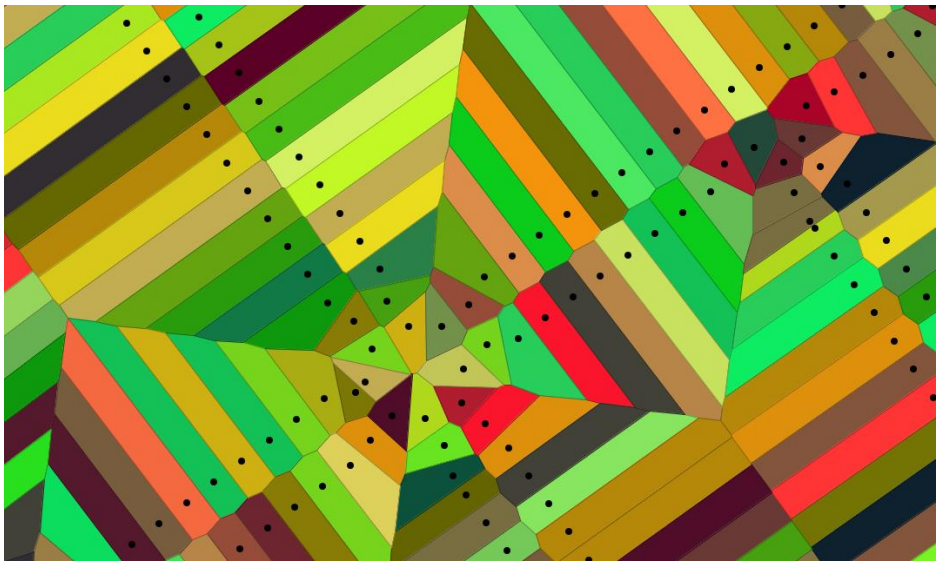


Figure 2.1.I: Voronoi Diagram using Euclidean distance as estimator [Wikipedia]

Various algorithms can be used to build a *Voronoi Diagram*, conditioning the order in which the polygons are defined and the overall efficiency of the progress. The most employed algorithms are incremental construction, divide and conquer, sweep and lifting to 3-space [Aurenhammer].

Voronoi Diagrams are mostly used in studies that determine influential areas for cities (relevant buildings location), communications (mobile reception area control) and biology (vegetal species population analysis).

2.1.4 Techniques used in this project

This work uses *L-System* as the technique for procedural graphics generation.

L-Systems allow to build a full grammar and set the derivation steps to produce variated geometry, nonetheless with enough control to obtain expected results. This technique has evolved from fractals, which is not complex enough for our work.

Perlin Noise adds noise to a previously defined terrain. In addition to the fact that the terrain in our environment does not need to be noisy, the amount of noise this technique applies is hard to control, ending up in random productions. However, it is useful in generating noisy texture images for already crafted elements.

Voronoi Diagrams aggregate a new factor in the generation, which is influence. Influence is not required in our setting, as objects are placed in a fixed location regarding disposal semantics.

The actions mapped to each symbol of our *L-System* do not exactly build new geometry from scratch. Instead, they specify which object is placed in which location of the available space. Changing the order of the productions swipes the orderliness of the elements insertion and locality, generating multiple different environments using the same elements.

From all the techniques described before, noise functions will be used to produce variations of manually designed objects whereas an *L-System* will be developed to define the structure of the office environment by arranging the different elements. *Perlin Noise* will be needed to generate procedural textures for some elements.

2.2 Office environments

In this section we study the structure of typical offices and how the main elements are arranged. This will enable us to build a model that captures the main features of this kind of environments.

2.2.1 Office design definition

Office design is defined by BNet Business Dictionary (2008) as, “the arrangement of workspace so that work can be performed in the most efficient way”. Office design incorporates both ergonomics and work flow, which examine the way in which work is performed in order to optimize layout [PAAM].

2.2.2 Office arrangements

Office arrangement differs according to the office dimensions. Large office environments are not in our concern, and therefore, we will deal with medium and small-size offices only.

Given the great variability in size and shape, the structure of an office depends on the business nature and requirements. However, the specifications given are quite general for every kind of business.

Medium-size offices

Figure 2.2.A shows a typical layout for a medium office. As we can see, it has some characteristic areas:

- Reception: it is a wide area in front of the main door with a reception desk.
- Meeting area: it has the necessary space for meetings, with a large desk and several chairs.
- Open office: this area is where office stands are. The number of stands depends on how big this area is. They consist of the following elements:
    - Working desk.
    - Office chair.
    - Supplies.
- WC.
- Kitchen.

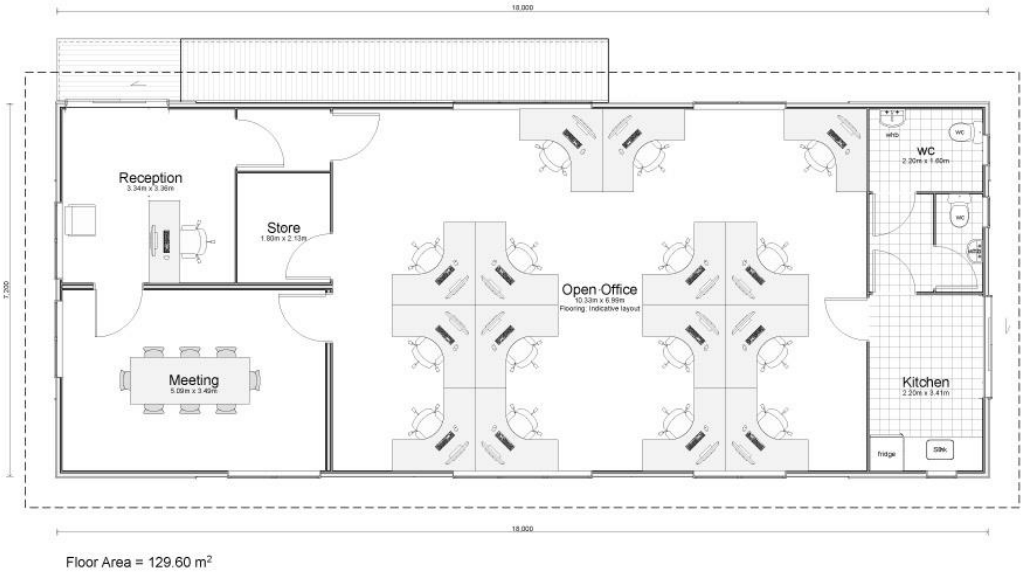


Figure 2.2.A: Medium office layout 1 [Activluton]

In this work we will cope with here are the meeting and open office areas. Working desks are placed to save space and keep workers side to side. It is also noticeable that the usca

In Figure 2.2.B we can see another office arrangement. In this example, the open office area is replaced with small work areas. However, the main elements listed before still remain.

Figure 2.2.B: Medium office layout 2 [Activluton]

Small-size offices

Small-size offices do not have the same extension as the offices seen before, thus some areas are missing.

We cannot find here the kitchen and WC areas, as these are placed outside the office itself. Instead, the two main areas (meeting and working areas) are kept, but in a very reduced space and next to each other. Reception area gets combined with the meeting area.

The arrangement of this kind of offices depends on the shape of the office and personal taste, so different configurations can emerge. In Figure 2.2.C the most common ones are shown:

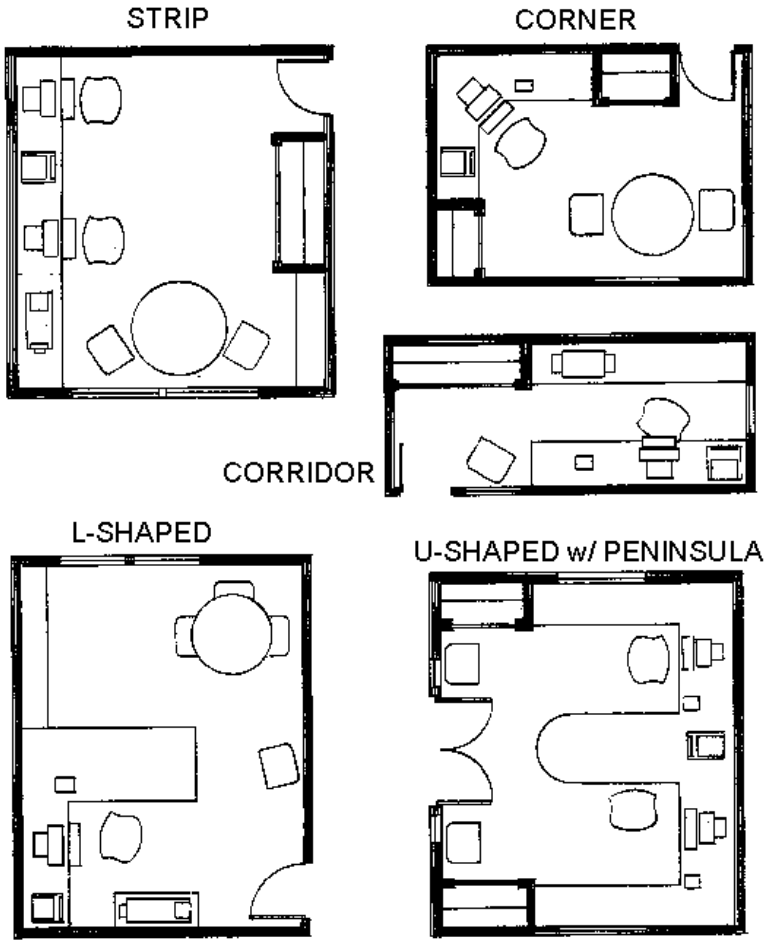


Figure 2.2.C: Small-size office most common configurations [Activluton]

Round-shaped desks are intended to be meeting points, while the rest of the desks are left for working purposes.

Some small-size offices do not have meeting areas, and they are only reduced to a single desk. These kind of arrangement (Figure 2.2.D) is too simple for our interest.

The kind of office this project is based on is a small-size office set with meeting and working areas, next to each other. As in medium-size office configurations, the meeting area is the closest one to the entrance. The desired setting is similar to Figure 2.2.E.

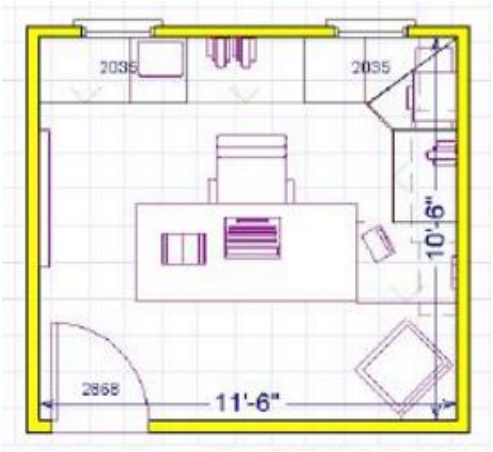


Figure 2.2.D: Home office [Activluton]

Figure 2.2.E: Office setting in which this project is based on [Alkiloo]

2.2.3 Office furniture and supplies

allBusiness webpage [AB], a very influential business site, states a list of all furniture, supplies, cleaning goods and technology an office needs. Table 2.2.A shows a reduced list taken from this website:

Table 2.2.A: Office elements list [AB]

SUPPLIES	CLEANING	TECHNOLOGY	FURNITURE
• Binders • Boards & easels • Calendars & planners • Envelopes & forms • Labels • Mailroom, shipping, & moving supplies • Paper • Post-it® Notes • Storage & organizers • Writing & correction • Paper and binder clips	• Paper towels and napkins • All-purpose cleaners • Coffee makers; cups; lids; stirrers; creamer; sugar • Trash bags and cans; light bulbs; air freshener • First aid kit	• Batteries and surge protectors • Computer accessories: mouse pads; monitor and machine stands; speakers and headsets • Computers & PDAs: PCs or notebooks • Copiers & fax machines • Digital cameras & scanners • GPS & satellite radio • Ink & toner cartridges • Office machines & calculators • Peripherals & memory • Printers • Shredders • Telephone & communication	• Storage cabinets • Bookcases and shelving • Chair mats & floor mats • Chairs • Desks & file cabinets • Lamps & lighting: incandescent; fluorescent; halogen • Office decor & plants: • Panel systems/accessories

This list is intended for medium-big offices and, because we are only concerned about small-size offices, only a small subset of these elements will be taken. Also most of these elements are small features that can be unnecessary in our environment. Efficiency is also a factor to consider: the generation must be as fast as possible, so only those items that are needed for a robotic simulation will be adopted.

Taking all of this into account, the relevant objects for our office generation are:

- Meeting and working desks
- Shelving
- Cloth hanger
- Rubbish bin
- Chairs
- Cork panel
- Reading lamp
- Monitor, keyboard and mouse

These elements are enough for a typical office place.

2.3 Blender

Blender (logo in Figure 2.3.A) is a free and open source 3D creation suit. It supports modeling, rigging, animation, simulation, rendering, compositing and motion tracking tasks.

Blender is cross-platform, running well on Linux, Windows and Macintosh computers. It is based on OpenGL. Blender has been actively developed for more than twenty years.

Figure 2.3.A: Blender logo [Blender]

Its current version is 2.77a (as of April 6, 2016). The common usage of this tool is via user interface. This interface is composed of several windows, each one with the possibility of choosing between various panels and tools, providing the user with working flexibility. The default interface set is in Figure 2.3.B. The user can interact with the tool using only the mouse. However, Blender is keyboard-oriented, providing a set of shortcuts covering its main functionalities, aimed at for time saving.

Besides, programmers can use Blender API in Python to make scripts that benefit from the same functionality the user interface provides, as every control in the GUI is associated with a Python function. This is useful to automatize modelling process to save

a notable amount of time, and craft regular and/or procedural geometry that could be very hard or even impossible to do by hand [Blender].

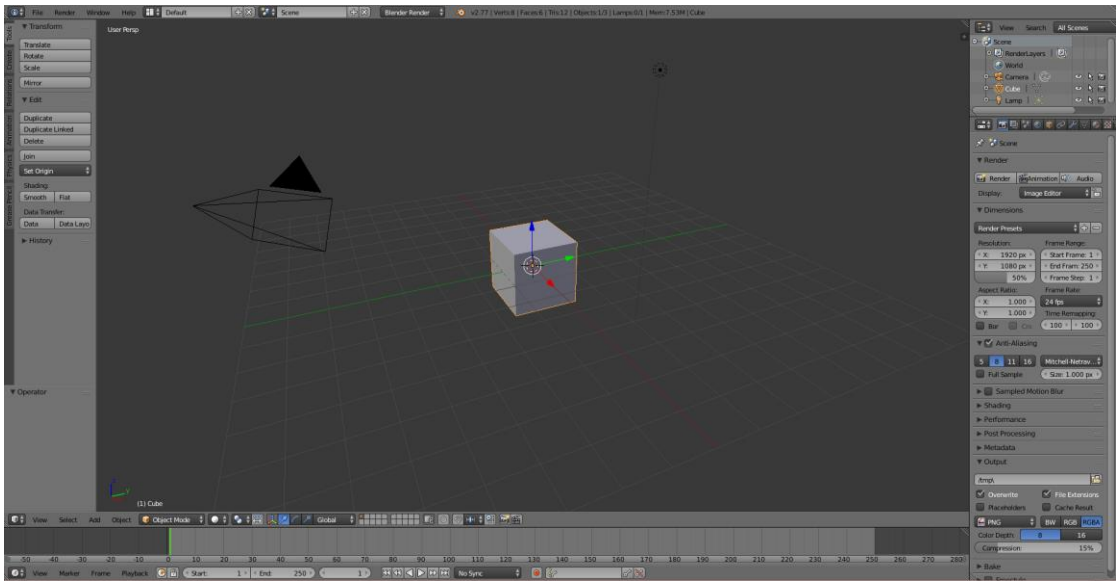


Figure 2.3.B: Blender default interface setting [Blender]

2.4 Python

Python is a widely used high-level interpreted programming language. It is for general purpose and its philosophy is based on code readability and simplicity: coders can express concepts in fewer lines of code than in languages such as C++ or Java.

Interpreters are available for many operating systems; therefore, Python can run in a huge variety of machines.

Its most recent stable releases are Python 3.4.4 (December 21, 2015) and Python 2.7.11 (December 5, 2015). These two versions are differentiated due to the big differences between them [Python].

Blender API uses 2.6.6 version [Blender] and, hence, this is the version we will use for building the scripts for this project.

2.5 Robotic simulators

This sections verses on robotic simulator tools, an important component of this work, since the 3D virtual environments produced will be adapted to be used in robotic simulation platforms.

2.5.1 Description

Robotics simulators are used to create embedded applications for robots without depending physically on the actual machine, therefore saving cost and time. In some cases, these applications can be transferred on the real robot without modifications.

One of the most popular industrial applications for robotics simulation is for D3 Modelling and rendering of a robot and its environment. This type of robotics software has a single unit or a set of virtual robots that are capable of emulating the motion of an actual robot in real life. Some simulators use a Physics engine for realistic renderings and movements of the robot inside the 3D space.

Physics engines provide precise physics models with variables such as mass, velocity and friction, and can simulate and predict effects under conditions that approximate the real world.

In robotics laboratories, access to equipment and testing environments is limited and there are few trial-and-error experiences possible due to the risk of injury to personnel and/or damage to equipment. Robotics simulation software enables programmers to write their own robot programs, modify the environment and use the available sensors. [Robologix]

2.5.2 Advantages and disadvantages of simulation

These are the benefits and disadvantages of using a simulator for robotics purposes [SR]:

Advantages

- Reduces cost involved in robot production.
- Diagnoses source code that controls resources.
- Simulates various alternatives without involving physical costs.
- Robot or components can be tested before implementation.
- Simulation can be done in stages, which is beneficial for complex projects.

- Allows to make demonstrations of a system to determine if is whether viable or not.
- Simulators are compatible with a wide range of programming languages.

Disadvantages

- An application can simulate just what it is programmed to simulate. That means it will not simulate internal or external factors which are ignored in the development phase.
- A robot can encounter many more scenarios in the real world than there can be simulated.

2.5.3 MORSE

MORSE is a generic open-source simulator for academic robotics. It focuses on realistic 3D simulation of small to large environments, indoor or outdoor, with one to tenths of autonomous robots. MORSE can be controlled from the command-line, and simulations are generated using Python scripts. It comes with a set of standard sensors, actuators and robotic bases, which enable fast creation of simulation scenarios. [MORSE]

It extends the Blender Game Engine (a render engine which features shader-based 3D rendering and physics simulation), so it allows for semi-realistic simulation of complex environments (Figure 2.5.A). It integrates in the Blender 3D model and switches to Game Engine. The user can also edit the properties of the simulation in Blender tool before running rather than executing it right away.

Figure 2.5.A: MORSE outdoors simulation [MORSE]

Therefore, MORSE is not considered as a physically accurate simulator. It is not expected to accurately simulate robot arm dynamics or fine grasping

MORSE components mentioned before (sensors, actuators and bases) exchange data wilt the robotics software via middleware buildings. The actual supported middleware are ROS, YARP, Pocolilbs, MOOS, HLA, Mavilink and generic socket-based protocol. [Brugali]

It is mostly supported and developed on Linux. It is known to also run on MacOSX and Windows, but only limited support can be provided for these platforms. [MORSE]

This tool was chosen for this project due to its full compatibility with Blender. It is also open-source, simple and easy to use, Python-scripted and supports useful middleware modules.

CHAPTER 3. PROCEDURAL OFFICE GENERATION

3.1 Main elements modelling

Initially, we planned to use a procedural method to obtain the elements (furniture) of the initial office. However, because most of the office elements have a very characteristic shape, the use of a procedural approach was discarded.

Instead, we opted for building a set of base objects, manually (that is, using the Blender graphical interface) and, then, apply on those objects a technique to introduce variability.

Models do not have excessive level of detail, as it could increase computational cost in an unnecessary way.

An introduction to polygonal modelling is first encompassed, to better comprehend the description of the covered elements. For the shake of clarity, we will show in detail how one of these elements was modelled.

3.1.1 Polygon modelling

Polygonal modelling is based on conceiving 3D objects as collections of polygons. The objects created using these techniques have three main elements that make up their structure (Figure 3.1.A):

- Vertex: it is a point defined in the 3D space.
- Edge: it is a segment that connects two vertices.
- Face: it is a surface delimited by the vertices and edges.

A close loop of edges constitutes a face. These faces are formally named polygons, and are the visible parts in a model. Models that are composed of one or more polygons are properly called meshes.

Each vertex is a tuple of three values, representing the location of it in 3D coordinates (x, y and z). The whole mesh location is the median value of its vertex coordinates, and the resulting point is considered as the origin of the object.

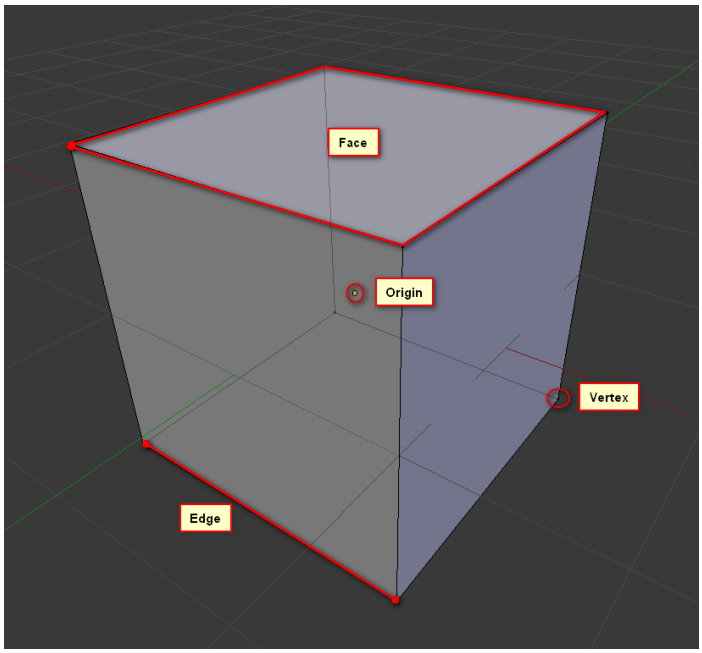
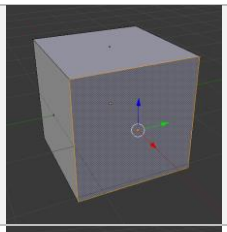
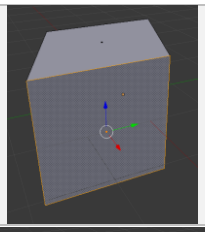
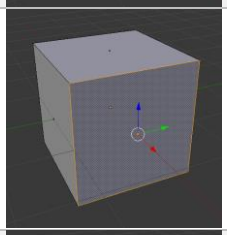
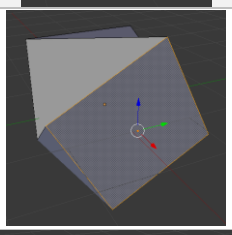
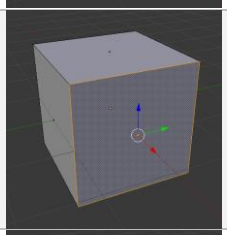
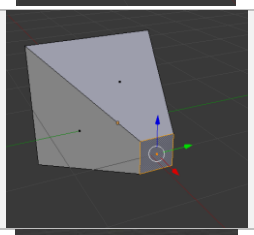
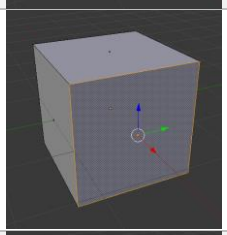
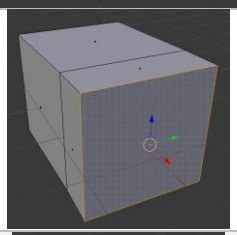
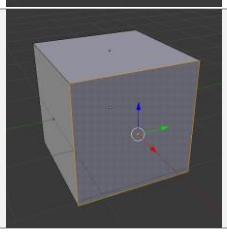
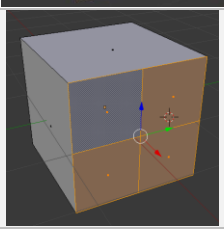


Figure 3.1.A: Mesh elements description

Operations are applied to meshes to modify their shape or to add more geometry to them, increasing their level of detail. The most used and common actions are listed in Table 3.1.A, including examples of each operation effects. The kind of modelling operations we can use in polygon modelling is determined by the structure of the 3D object. This way we can apply transformations to vertices, edges and faces.

They are always applied in at least one dimension, up to three, with a certain amount of units. For instance, a rotation operation can be completed around x, y or z axis or either a combination of these, rotating a particular number of degrees on each. Figure 3.1.B clarifies this idea using a colored pyramid seen from above. (a) is the original object, while in (b), (c) and (d) a 90º rotation in z, x and y axis is applied, respectively. (e) is a combination of 30º x and 30º y rotation, and (f) is a blend of 45º z and -30º y spin.

Table 3.1.A: Most common mesh operations

NAME	DESCRIPTION	BEFORE	AFTER
TRANSLATION	Moves the selected feature (E.g: face translation)		
ROTATION	Rotates the selected feature relative to a pivot point (E.g: face rotation)		
SCALE	Changes the size of the selected feature (E.g: face scaling)		
EXTRUSION	Generates new geometry from the selected feature (E.g: face extrusion)		
SUBDIVISION	Splits the selected feature (E.g: face subdivision)		

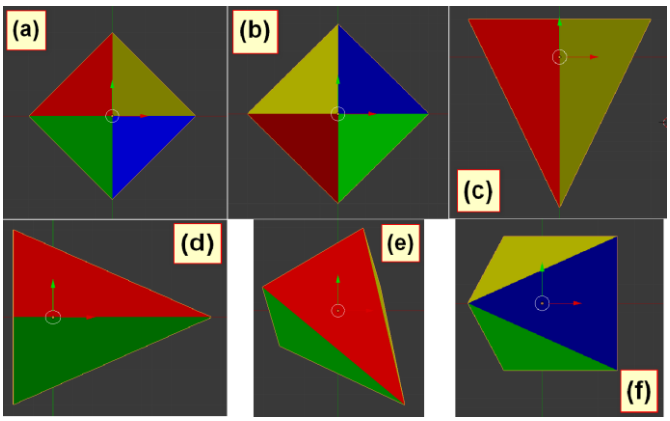


Figure 3.1.B: Rotation examples

Example: Modelling a rubbish bin

Below is the process followed to model a rubbish bin, an object included in this work. Each step is illustrated in Figure 3.1.C.

a. The model starts from a default cylinder provided by Blender tool.
b. The upper face is scaled up, to achieve the basic bin shape.
c. The same face is extruded (to generate a new independent face) and scaled up. This drafts the top part shape of the bin.
d. This new face is extruded to reach the top.
e. After that, it is extruded again, to create another face, which is scaled down. The amount of scaling determines the width of the borders of the bin.
f. This smaller face is grabbed to the bottom of the model and scaled down to adjust to the outer faces. This determines the capacity.
g. The object is now completed.

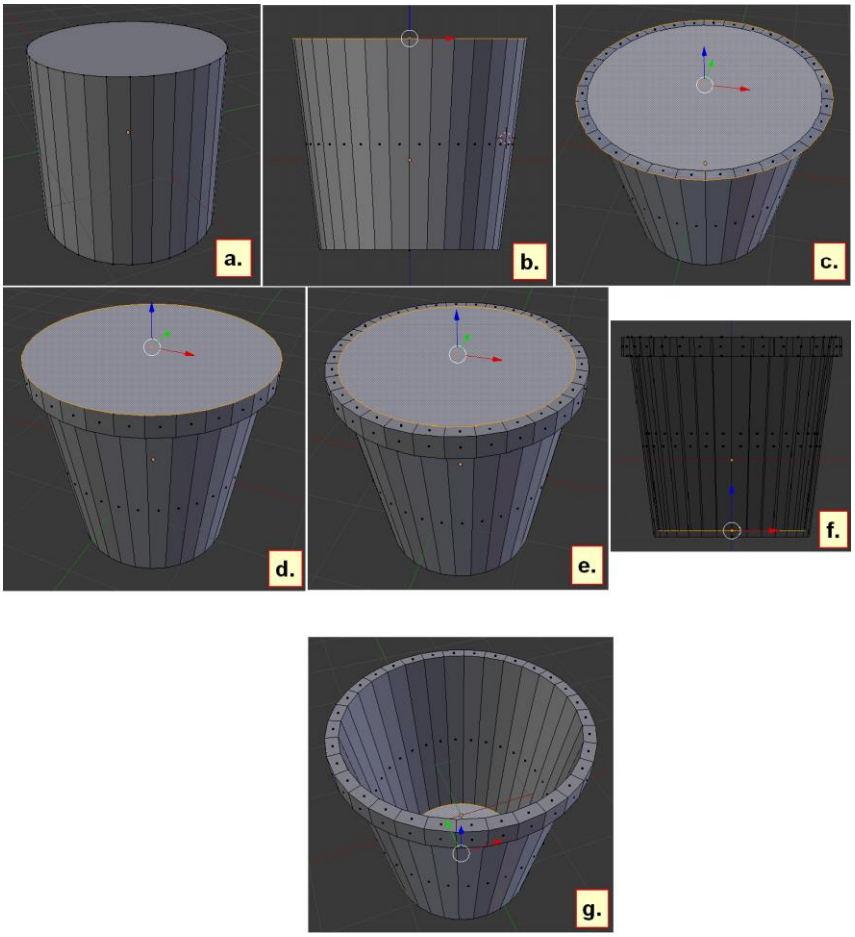


Figure 3.1.C: Rubbish bin modelling process

3.1.2 Furniture

Meeting desk

A simple meeting desk was made from a cube basis. Its lower face has been subdivided so its legs can be extruded in a regular way. The result is shown in Figure 3.1.D

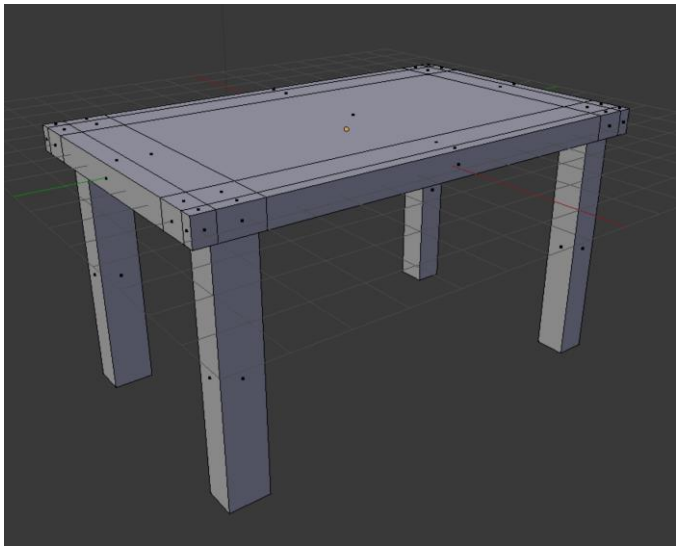


Figure 3.1.D: Meeting desk modelling result

Working desk (first type)

It is a desk inspired in typical working desks with drawers on its side. It is thinner than the meeting desk. Its basis is a cube subdivided in the drawers' area, the lower part of the table and the right basis, as shown in Figure 3.1.E. Three drawers were created subdividing the widest face after extruding the areas, as it could be seen in Figure 3.1.F. Handles were modelled using a plane as a basis, and extruding to achieve its proper shape. Result is appreciable in Figure 3.1.G.

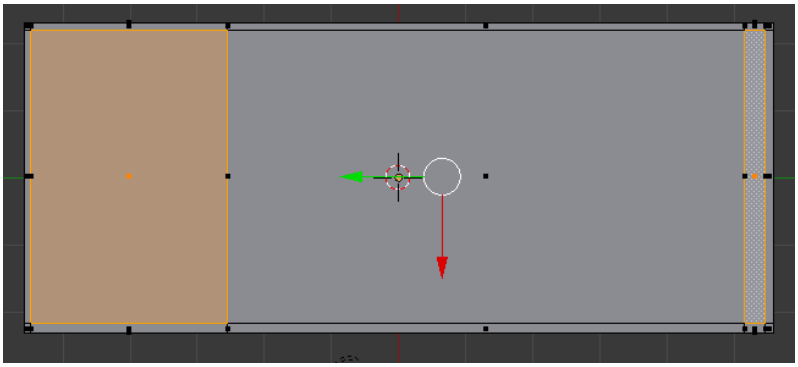


Figure 3.1.E: Working desk subdivision

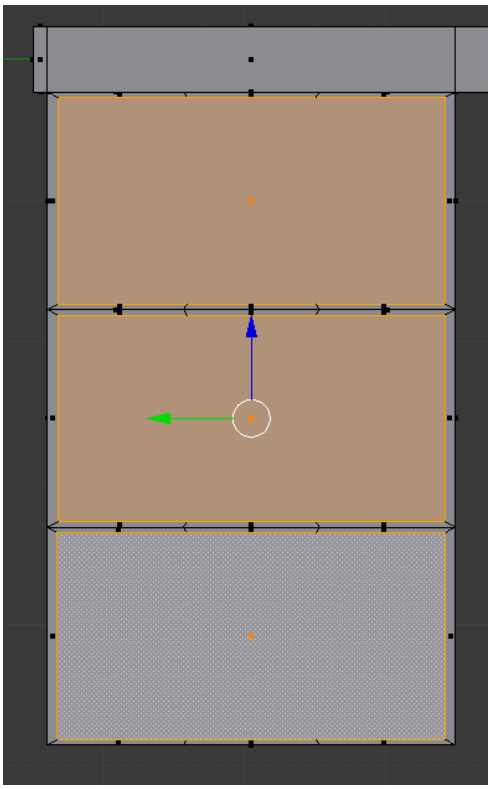


Figure 3.1.F: Drawers subsivision

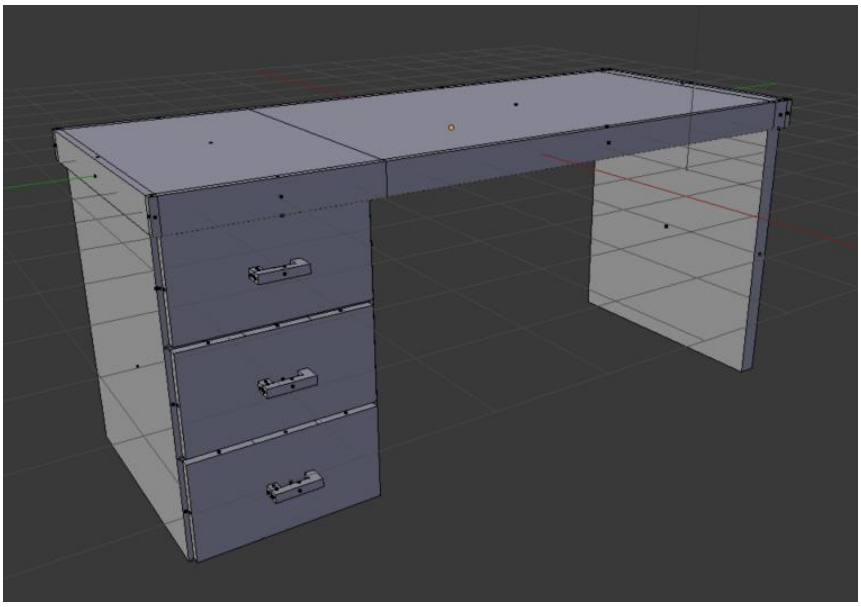


Figure 3.1.G: Working desk (first type) modelling result

Working desk (second type)

This is also a common working desk very similar to the first time, but this desk has two drawer sections instead of one. The basis and the way of modelling are identical to the first type. The result is shown in Figure 3.1.H.

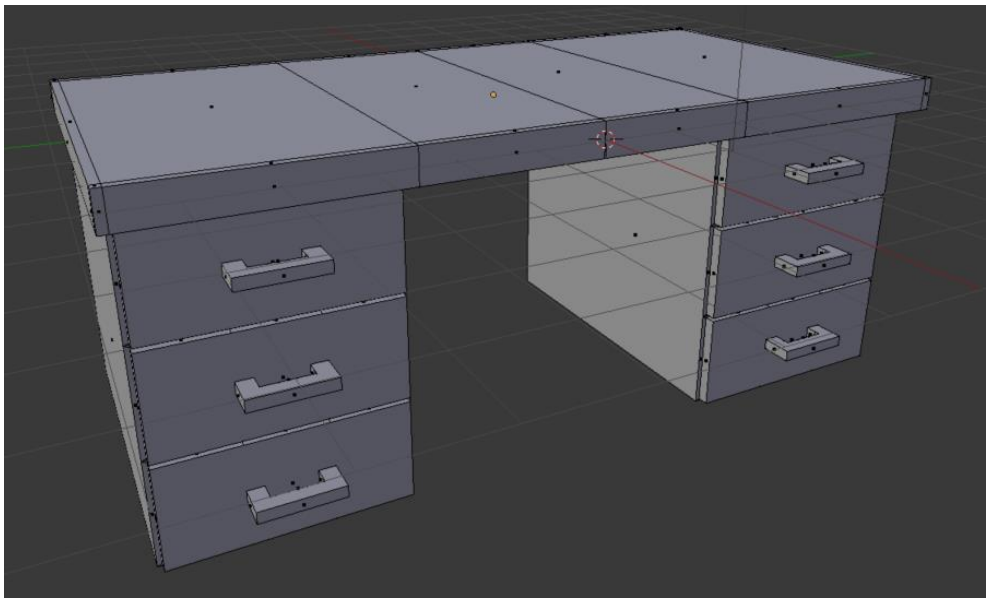


Figure 3.1.H: Working desk (second type) modelling result

Shelving

It is a classic shelving design with four cells. The mesh basis was a cube subdivided and extruded in a regular way to achieve what Figure 3.1.I shows.

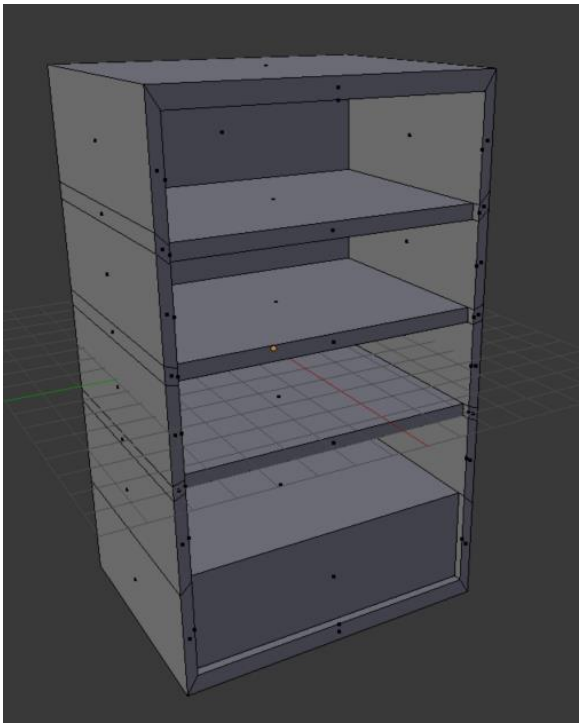


Figure 3.1.I: Shelving modelling result

Cloth hanger

It is a hanger stand usually found in small offices. Its basis is formed from a cube that has a cylinder attached to it. Hangers are placed alongside. These consist of thin extruded cubes with spheres on the tip. It has a symmetric appearance seen from the top in Figure 3.1.G and the whole hanger design appears in Figure 3.1.J.

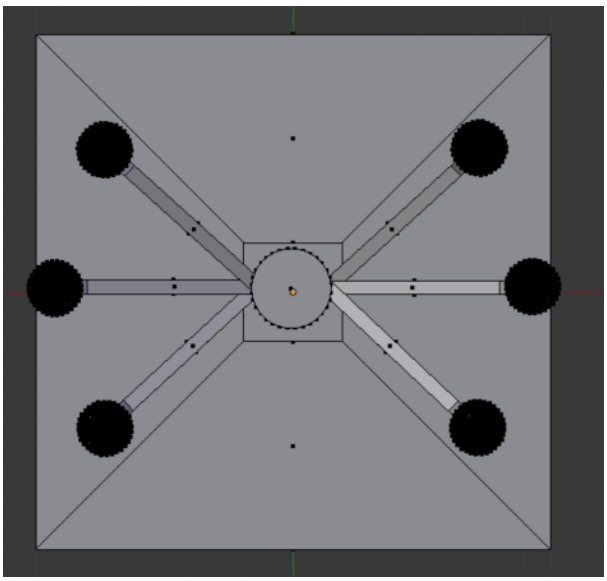


Figure 3.1.J: Cloth hanger seen from above

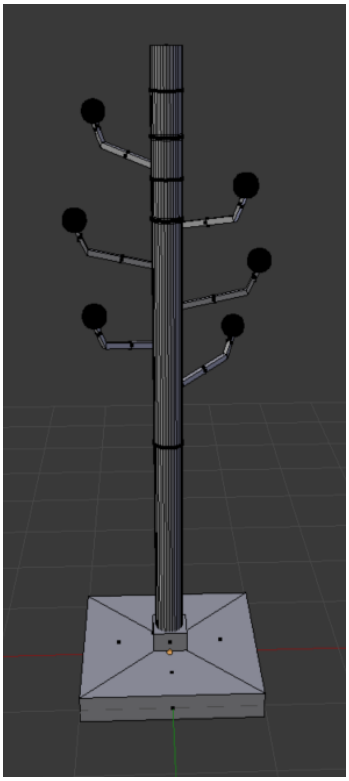


Figure 3.1.K: Cloth hanger modelling result

Chair (first type)

It is inspired in a classic office chair with adjustable height seat and back and with two armrests.

The bottom part of the chair is made from a cylinder with 5 faces extruded in a regular way, that represent the legs. Small cylinders were attached to the bottom to simulate the wheels. The upper part consists of the chair's seat skeleton made from a cube extruded using a spin tool to get nice circular edges (Figure 3.1.L) and cubes modified to act as chair's foam. The result is in Figure 3.1.M.

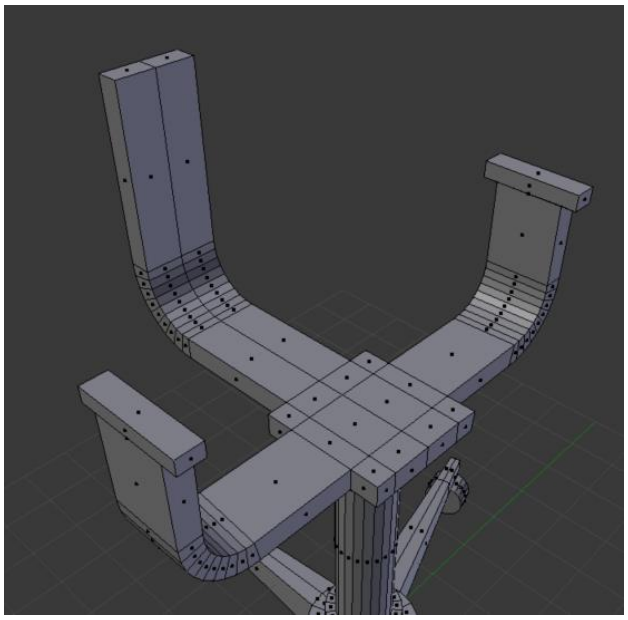


Figure 3.1.L: Upper part skeleton

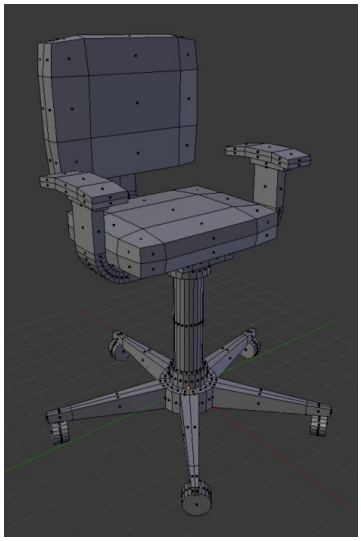


Figure 3.1.M: Chair (first type) modelling result

Chair (second type)

This is a straighter and more classic kind of chair taken from a cube with extruded faces generating its legs and back skeleton. Two cubes were shaped to represent foam seats and back. A mirror modifier was used for better results. The output is in Figure 3.1.N.

Figure 3.1.N: Chair (second type) modelling result

3.1.3 Supplies

Cork panel

A cube shaped as a panel emulating a basic cork panel, as shown in Figure 3.1.O.

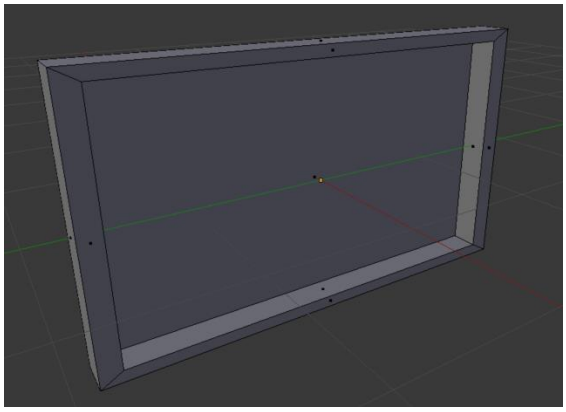


Figure 3.1.O: Cork panel modelling result

Rubbish bin

A cylinder modified to emulate an office bin, displayed in Figure 3.1.P.

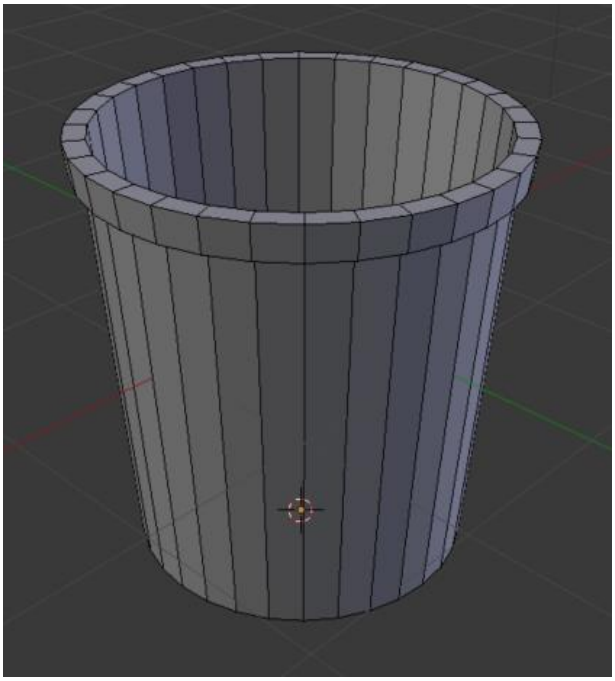


Figure 3.1.P: Rubbish bin modelling result

Reading lamp

A typical standing desk reading lamp, made with two low-poly cylinders as the base and support and half of a UV sphere as the lamp bell. The outcome is Figure 3.1.Q.

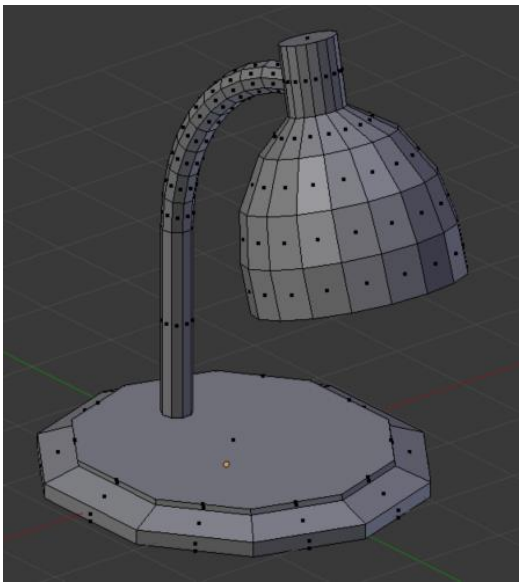


Figure 3.1.Q: Reading lamp modelling result

Monitor

A cube-based nowadays display, which is captured in Figure 3.1.R.

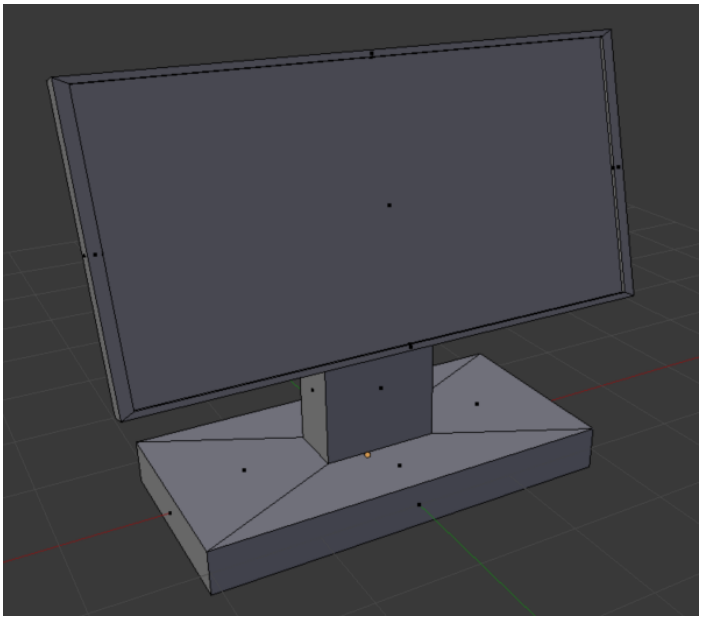


Figure 3.1.R: Monitor modelling result

Mouse

A classic wireless mouse, using a cube for the body and a torus as the mouse wheel. It can be seen in Figure 3.1.S.

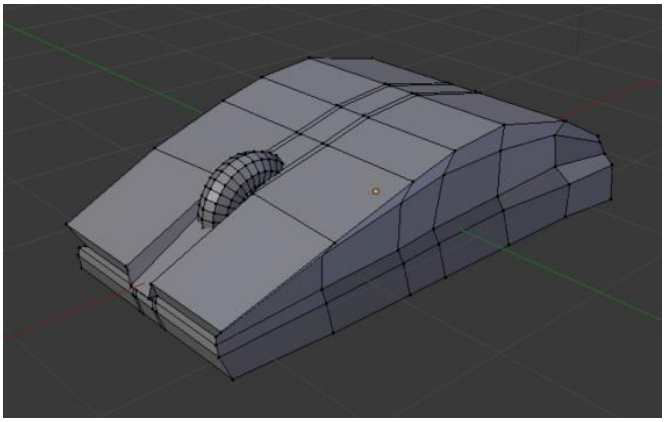


Figure 3.1.S: Mouse modelling result

Keyboard

A wireless keyboard with numeric pad, made from cubes. The keys arranging was achieved using a real-life keyboard key map picture. The result is in Figure 3.1.T.

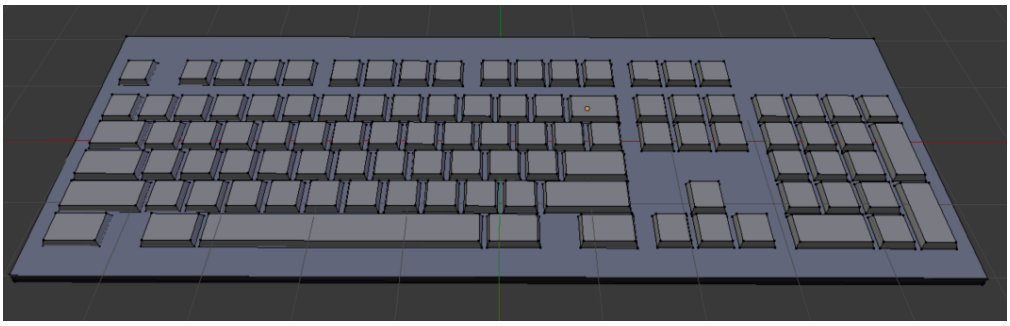


Figure 3.1.T: Keyboard modelling result

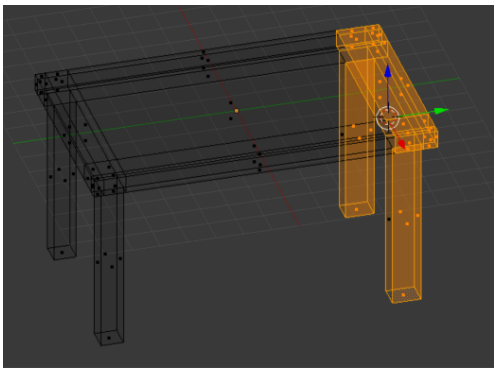
3.2 Object variations

As it was specified in the introduction, one of the main goals of this project is to automatically generate 3D objects, using a procedural approach. In this section we are going to discuss how to apply procedural techniques to get objects similar to the manually design elements described in the previous section.

For instance, we can generate multiple desks with different length, width and height dimensions every time from the available desk object.

The method to achieve this is based on modifying the faces. The available operations for a face are location, scale and rotation. These modifications (the operations to apply) vary depending on the characteristic of the object that we want to modify. The set of faces to modify is obtained through an interactive process where a user marks the faces to be changed. This way, each model keeps a list of faces that can be modified to obtain a new model.

For instance, to change the length of the meeting desk, the side faces have to be displaced along the axis that correspond to the largest edge in the face. To get those faces, a user has to select the faces to be changed first. Then, a Python script is executed, saving the selected faces in a list. This list is later stored for further use. Figure 3.2A shows se set of faces for this particular example.



```
[4, 5, 6, 13, 17, 18, 28,
34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 59, 60,
61, 62, 63, 64, 65, 66, 67,
68, 72, 73, 74, 78, 81, 83]
```

Figure 3.2.A: Faces involved in a length variation on the meeting desk

Once the faces are selected, noise is applied to them to perform the modifications. This noise is achieved by means of random numbers that determine the units of the operation. For instance, this determines the amount of units that the selected faces are going to be displaced to modify the length of the desk. Units can be either positive (which enlarges the length of the desk) or negative (which shrinks the length of the desk).

Random numbers can be obtained from three different distributions. Zero-mean Gaussian distributions are used to perform relatively small changes to the object on both directions with high probability, while large changes ae not as usual. This prevents the object from deforming drastically along several scene generations. Non-zero-mean Gaussian distributions give more importance to changes on a certain direction, depending on the sign of the mean of the distribution. Finally, uniform distributions grant the same importance to all possible changes.

Variance in Gaussian distributions and range in uniform distributions stablish the span of the modifications. These are determined taking into account the amount of deformation a model can stand before it loses its regular proportions.

This approximation is an initial step to semi-automated variation generation for 3D models.

All modifiable objects with their available variations and specific operations to achieve them are in the following table (Table 3.2.A):

Table 3.2.A: Available object variations and operations

OBJECT	VARIATIONS	OPERATIONS
Cork panel	• Length modification • Width modification	• Y Displacement • Z Displacement
Rubbish bin	• Height modification	• Z Displacement
Shelving	• Width modification • Height modification	• Y Scale • Z Scale
Meeting desk	• Length modification • Width modification • Height modification	• Y Displacement • X Displacement • Z Displacement
Working desk (first type)	• Length modification • Width modification	• Y Displacement • X Displacement
Working desk (second type)	• Length modification • Width modification	• Y Displacement • X Displacement
Monitor	• Width modification • Height modification • Orientation modification	• X Scale • Z Scale • X Rotation
Chair (first type)	• Seat height modification • Back height modification	• Z Displacement • Z Displacement
Chair (second type)	• Chair width modification • Seat height modification • Back height modification	• X Displacement • Z Displacement • Z Displacement

This is an example of executing three times this algorithm on the meeting desk (Figure 3.2B):

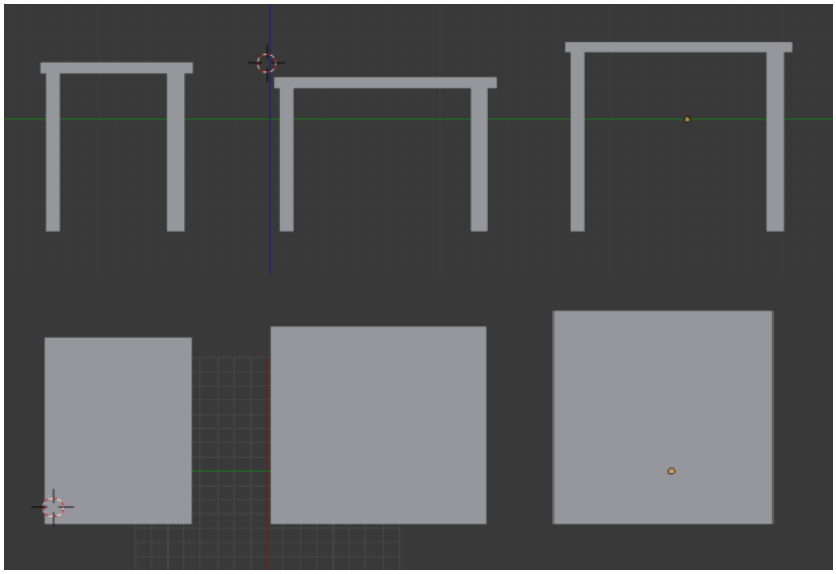


Figure 3.2.B: Three executions result of object variation algorithm on the meeting table mesh

As seen, there are some morphological changes, but the main desk shape is still appreciable.

3.3 Materials and textures

Models have materials attached to them to give them color and specular properties. These are:

- Wood material: imitates real-life wood with a procedural noise texture (better detailed below).
- Color material: serves as color template for the reading lamp and foam parts of chairs.
- Grayscale material: tones of gray for the reading lamp, keyboard, mouse, monitor body and rubbish bin.
- Walls material: assigned to the room walls when generated (explained in 3.4).
- Floor material: assigned to the room floor when generated (explained in 3.4)
- Black material: fixed black color and specular for the monitor's screen.

- Cork material: fixed brown color for the cork panel.

These materials are already predefined and can be modified (except for walls and floor materials, that are generated with the room). This is to allow the user to change the material color and let the algorithm modify some parameters according to those changes, to combine customization with procedural generation.

The algorithm dedicated to this takes all the predefined materials and changes their parameters depending on the material type. The diffuse intensity value is changed leaving the diffuse color values intact. This is due to prevent the randomness of the algorithm generating a color too different from the original predefined color. Although the user may leave the same diffuse color for two consecutive iterations, the colors will not be exactly the same, thus lighter and darker tones will be generated.

Wood material uses procedural textures provided by Blender to make it similar to real wood. The configuration of these textures changes every room generation, so it has an infinite variety of wood types. The basic components of this texture are the following:

- Basis wood texture. It is a simple wood texture layer with no noise. It can be either bands or rings. This serves as the basis shape of wood weave (3.3A).

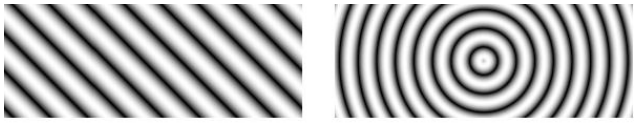


Figure 3.3.A: Basis wood texture. (a) is band basis while (b) is ring basis.

- Complementary wood texture. As its name implies, it complements the basis wood texture, adding some noise with the same shape (bands or rings). It can be either band noise or ring noise, and it is coherent with its base. That means it will be bands noise when the base are bands, and ring noise when the base are rings (3.3B).

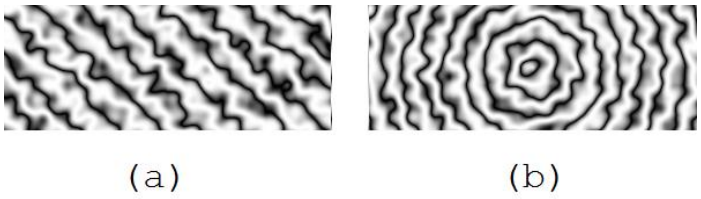


Figure 3.3.B: Complementary wood texture. (a) is band noise while (b) is ring noise.

- Pure noise texture: It is a *Perlin Noise* texture, having long patterns, that generates variety to the wood surface when weaves are not present (3.3C).

Figure 3.3.C: Pure noise texture

These layers are provided with a random offset, so weaves are not exactly round and uniform (which is unnatural), but somewhat stretched to resemble real wood. They also adapt to the user's desired wood color, so the noise color gets the material's diffuse color value and modifies it slightly to generate noise with similar color.

The result, given Figures 3.3A (b), 3.3B (b) and 3.3C, looks like this (Figure 3.3D):

Figure 3.3.D: Resulting procedural wood material

Figure 3.3E shows some execution examples with the same diffuse color in all of them.

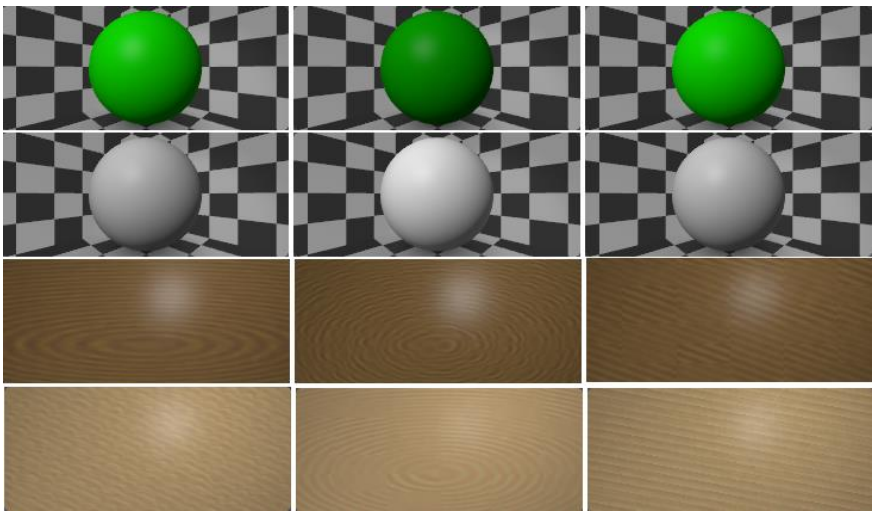


Figure 3.3.E: Generated materials in three executions

3.4 Procedural office generation

In this section, we will describe how the structure of office room is generated from the objects described in previous sections.

The room generation consists in creating the floor and walls of the room, the door and window holes, and placing the elements in their respective location.

The generation workflow is:

- Clearing the scene.
- Generating/creating materials.
- Generating room's walls and floor.
- Setting the structure and place the furniture and other objects.

Room bounds are generated with random but coherent dimensions, giving enough space for all objects to be placed inside while keeping a standard small office dimensions. After that, walls and floor textures, which were generated early, are attached to their respective faces. Then, we can define the office structure.

Bounds are represented by a list of four 2D points (XY coordinates) which are the upper-left corner, the upper-right corner, the lower-left corner and the lower-right corner.

Objects are placed inside the room's bounds following regular patterns. These patterns are determined by the space needed by each item, based on a basic office structure. In this case, room's dimensions are divided in two main areas (Figure 3.4.A) *meeting area* and *work area*. The meeting area is next to the door. It is meant to be the place where people have work meetings. And therefore it needs the following elements: meeting desk, chairs, a shelving and a cloth hanger. The working area is on the opposite side to the door and is conceived as the place to work, so it has a working desk with its supplies, a trash bin and a cork panel. Given this basic structure, the procedural model will distribute all these elements so we can obtain variations in the office structure.

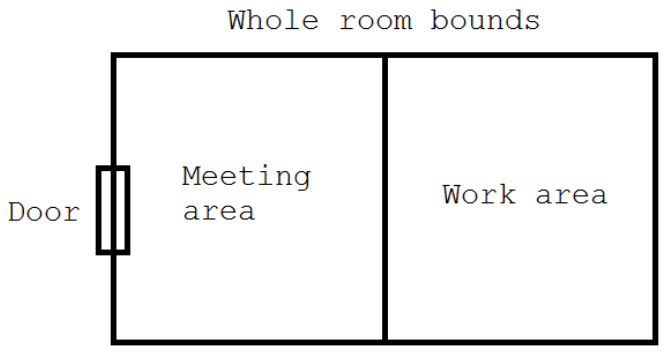


Figure 3.4.A: Room space distribution

A noise parameter is used to generate both areas. Depending on its value, it will produce them with the same size or one (either meeting or working area) larger than the other. The difference between both sizes is not meant to be high, in order to have enough space in each area.

In order for the objects to be placed, the model sets the zones where they are going to be positioned inside. It divides the two main areas (meeting and working areas) following a regular pattern based on typical office arrangements. Then, each element is assigned to its corresponding zone. An example of this distribution is shown in Figure 3.4.B):

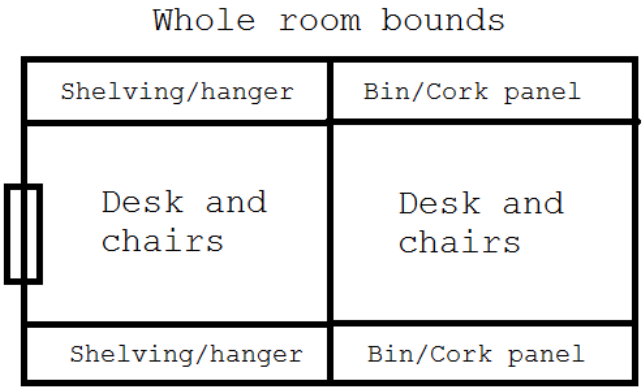


Figure 3.4.B: Object allocation distribution

The procedural model takes the elements described in Section 3.1 as an input parameter. Next, it modifies their morphology using the approach defined in Section 3.2. Then, it designates them a zone where they are going to be placed, and finally sets their final location using a uniform distribution with range between the bounds of the zone. To

preserve the original shape of each element, the model makes a copy of them to work with.

The task of setting the bounds and taking the objects is carried out by an *L-System*. As discussed in Section 2.1.3, they are formal used to generate geometry. In this case, however, the *L-System* will produce the structure of the office by determining where to place each item.

As a reminder, an L-System is defined as a tuple $G = (N, \omega, P)$ where:

- N is a set of nonterminal symbols.
- ω is the initial symbol, also called grammar axiom
- P is the rules or productions set. They specify the way of deriving each symbol of N. Each rule is composed of two parts: a left-handed side called the predecessor and a right-handed side call the successor.

Different strings can be generated from ω applying rules defined in P to symbols in N.

To fully integrate this technique into this project, the following steps had to be followed:

- Defining the alphabet, as a set of symbols, where each symbol is associated with a specific action.
- Defining grammar from previously defined symbols, ω and rule set.
- Deriving strings from the previously mentioned grammar. It generates a string to be translated into geometry operations.
- Translating those symbols into actions previously defined. In this case, each action is related to the definition of the office structure.

This system in particular employs a stochastic version of *L-System*. Deterministic grammars will be able to produce a single model, as a symbol can only derivate in one string only. Stochastic grammars afford us more flexibility, as one symbol can generate several strings. This results in different geometry actions produced by the same symbol in different executions.

Successor parts of each rule have an associated probability number, which represents the probability for them to be derived. The sum of all probabilities of the right-handed parts of a rule with the same symbol as the predecessor is 1.

Taking this change into account, the grammar used for this system has a different representation than other deterministic grammars. Although N and ω sets remain the same, the rules in P are written in a different way:

S → (0.2 A) (0.3 B) (0.5 CS)

This is a stochastic rule conceived for this system. Each successor of S is shown in parenthesis. Each parenthesis is a duple composed of a probability number and a series of symbols. The number stats the probability for the series to be derived from S. For example, 'A' would be derived from S 20% of the time, while 'B' would do it 30% of the time and 'CS' would do it 50% of the time. Chances are based on a uniform distribution. Therefore, three different strings can be generated from the same symbol.

All grammar symbols and actions associated to them are gathered in Table 3.4.A:

Table 3.4.A: Grammar symbols and actions

Symbol	**Action**
S	(Initial symbol)
R	Generate floor and walls
M	Generate meeting bounds
W	Generate work bounds
V	Generate shelving
K	Generate meeting desk
T	Generate working desk (first type)
Y	Generate working desk (second type)
H	Generate chair (first type)
A	Generate chair (second type)
B	Generate rubbish bin

P	Generate cork panel
L	Generate reading lamp
N	Generate monitor
O	Generate keyboard and mouse
G	Generate cloth hanger
[	Save current bounds
]	Recover previous bounds
F	Finish room

When room's walls and floor are created at the beginning of the generation process, the floor area is stored as a bounding box variable, saving its four corners coordinates in a list. When another zone or object is going to be generated, these coordinates are modified. This modification creates a subzone inside the current area, thus it can only reduce the bounds size. The procedural model can only place an element inside the bounds that are represented by the current coordinates in the list. Therefore, it is necessary to recover the previous area where the subzone was generated to continue placing the rest of the items.

Saving the current bounds means pushing the current list of coordinates into a stack. This keeps a copy of them and prevents them from being overwritten. Recovering the previous bounds makes a pop action to the stack, claiming the coordinates that were saved before. Thus, saving is made before generating a subzone inside a certain area, and recovering is performed when another operation needs to be made in the same area as before.

When an object is generated (in a generate action), the procedural model takes the designate object as an input. It randomly modifies its shape and sets its final location within the current coordinates. The exact location can either be randomly calculated between the current space, be proportional to the amount of available space (avoiding placing elements next to each other) or have a fixed coordinate value. This depends on the kind of object to be generated.

Finishing the room makes the walls thicker to achieve a realistic aspect and generates the door and window holes. Three lamps are placed on top, covering the length of the room. This is supposed to be done at the very end of the office generation process.

Let this stochastic L-System grammar be an example:

$N = \{S, R, M, V, D, T, Y, [,]\}$

$\omega = S$

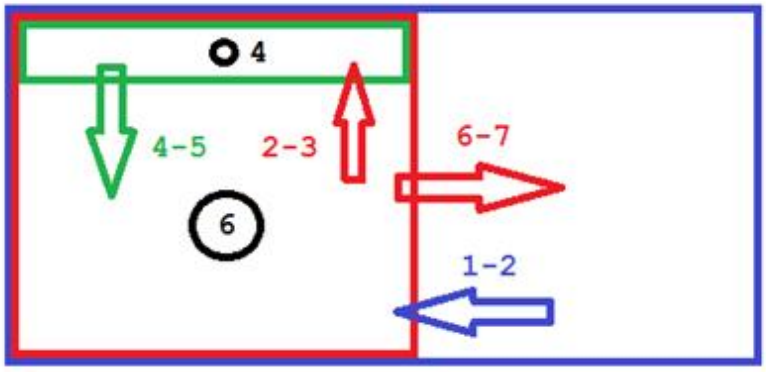
$P = \{S \rightarrow (1.0\ R[M[V]K])\}$

Deriving the rules of P will generate 'R[M[V]K]' with 100% probability. According the Table 3.4A, these symbols are translated into these generation steps:

- Generating room's floor and walls (R)
- Saving current bounds ([)
- Generating meeting bounds (M)
- Saving current bounds ([)
- Generating shelving (V)
- Recovering previous bounds (])
- Generating meeting desk (K)
- Recovering previous bounds (])

Room is generated and floor bounds are saved. Then these are modified to generate the meeting bounds, previously saving their values. Then, a shelving is placed. To place the shelving, a zone needs to be set for it from the coordinates of the meeting bounds. Therefore, the current coordinates must be saved, and then recovered to place the meeting desk. Finally, original floor bounds are recovered, and the office will be considered fully generated.

The following Figures give a graphical summary of this process. Figure 3.4.C shows the bounds reservation and object placement sequence, while Figure 3.4.D tells the final result in Blender. This time, the shelving took the upper space of the meeting bounds.



1: Floor bounds
2: Meeting bounds
3: Shelving bounds
4: Shelving is placed
5: Meeting bounds recovered
6: Meeting desk is placed
7: Floor bounds recovered

Figure 3.4.C: Generation workflow

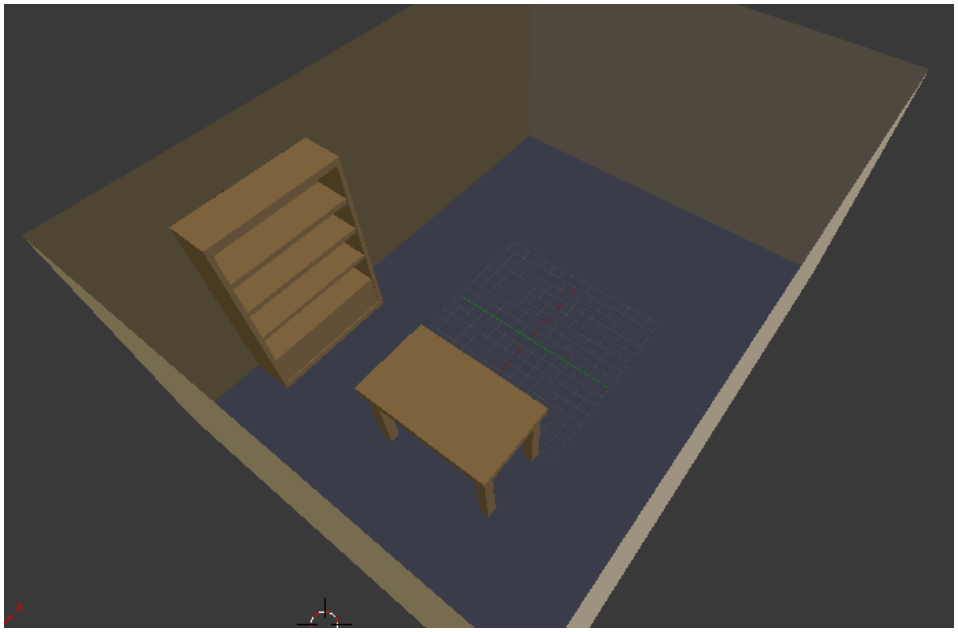


Figure 3.4.D: 3D generation result

We can generate as many office rooms as needed in one single scene. The number of generated rooms is determined by two control parameters (which are not part of the

grammar). The first parameter determines the number of rooms on the right side of the scene, while the second parameter controls the number on the left side. The additional created office rooms are placed next to each other, forming a row of offices.

In addition to this, a small floor area is built automatically beside the entrances, which is intended to serve as a corridor to connect all rooms that are on both sides. It is specially aimed at providing a more realistic environment for the robotic simulation.

Finally, there is a noise control parameter that defines the amount of disorder inside the rooms. It can be either a single number, which will affect to all rooms in the scene, or a comma-separated list with the same length as the number of rooms needed, which assigns a unique noise value for each room. Each value must be between 0 and 1. The noise affects to the arrangement of the chairs that are aside the meeting desk.

CHAPTER 4. SYSTEM WORKFLOW AND RESULTS

In this chapter, the workflow and architecture of the developed system is going to be discussed. The main results achieved will be also discussed.

4.1 System structure

As explained in Chapter 1, this project focuses on office environment procedural generation and the usage of this environment for robotic simulation purposes. The first part is achieved using the Blender API in Python. All the algorithms are coded in Python scripts, that are executed to build and save a full 3D office model. Next, the result is sent to the MORSE tool. MORSE takes the produced models as a testing environment where the virtual robot can interact. This structure is captured in Figure 4.1.A.

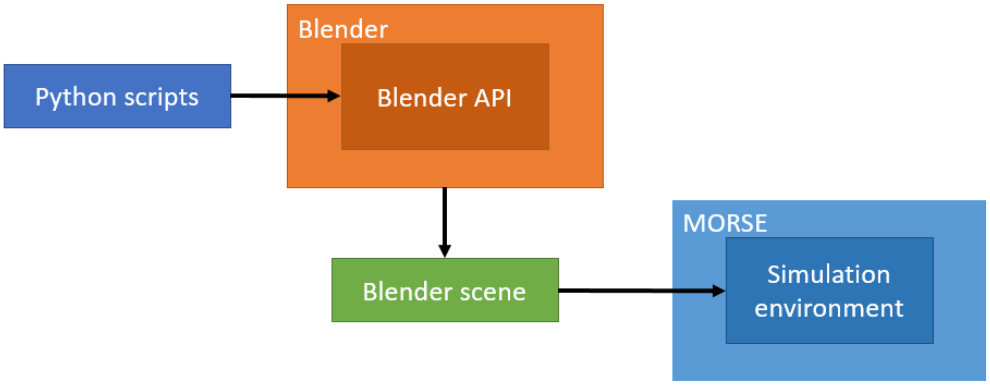


Figure 4.1.A: System architecture

The fact that we are using MORSE for this project does not mean that other simulation tools cannot benefit from the environment created. The 3D scene produced by the procedural generation system can be exported to several formats that can be used by most simulation tools.

As mentioned before, the generation is accomplished using Python scripts embedded into a Blender file that contains the manually designed objects described in Section 3.1.2. There are five different scripts involved in the generation process:

- *Office* script: it is the main script and it has to be ran by the user. It controls the rest of the scripts involved. It clears the scene and calls the *Materials* script to reset and generate the textures and materials in the file. Then uses the Grammar class of the *LSystem* script to read the grammar passed as argument and creates a string. This string is employed later on by the *ProceduralSetting* script. Finally, it saves the outcome in a Blender file.

- *Materials* script: it deals with the definition, resetting and generation of all the materials and associated textures needed by the 3D models.

- *LSystem* script: it reads a grammar file and the control parameters in it. It generates the string with the symbols mapped to actions contained inside the *ProceduralSetting* script. It also saves the value of the control parameters and passes them to the previously mentioned script.

- *ProceduralSetting* script: it is the most important piece of the process, as it contains all the actions and methods to generate a whole scene. It takes the string generated in the *LSystem* script and performs the translation process by sequentially executing the operation associated with each symbol in the string. It holds the bounds, parameters and properties of everything that can be generated. Some of these methods require to bring an object to the scene. In that case, the *Elements* script is called.

- *Elements* script: it collaborates with the *ProceduralSetting* script. Its main functionality is to duplicate, modify and allocate the predefined 3D models in the scene. The *ProceduralSetting* script calls for a specific object, thus the corresponding procedure is executed. There is a procedure for each available element, and perform the actions described above.

A simplified flow diagram describing this process is shown in Figure 4.1.B.

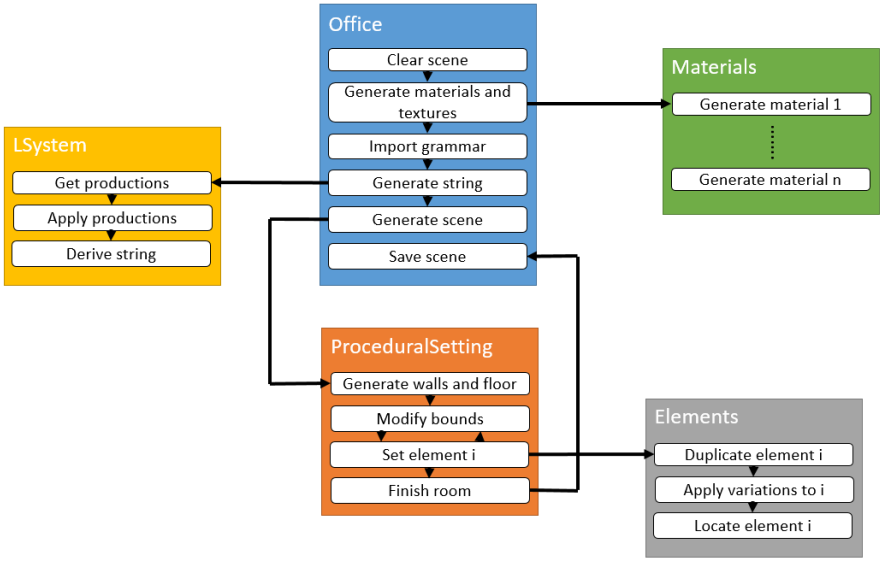


Figure 4.1.B: Office generation workflow

4.2 Project results

This section shows the main results of the stand-alone office environment production and its integration in the simulator.

4.2.1 Office environment

Figure 4.2.A shows the result of the generation of a single office room. From this perspective we can measure the proportions of it, as the grid is visible. The generated room proportions need to be reduced enough for the robot to have a proper size relative to the rest of the objects, but with the sufficient space for it to move freely.

Figure 4.2.B pictures multiple room generation. The scene holds three offices on the right side of the scene. Please notice how the corridor enlarges to fit and connect all of them. There is a minor separation between the walls of each room. This emulates real-life room separation, as a similar breach is left for electric setting and other pipes that are amid the walls.

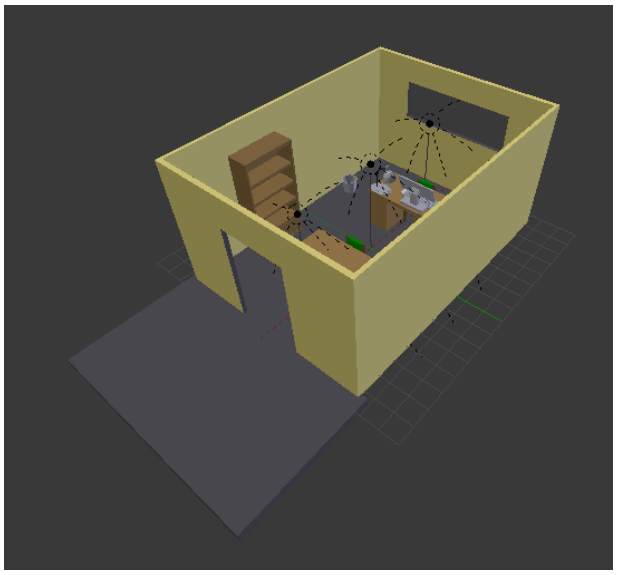


Figure 4.2.A: Single room result

Figure 4.2.B: Multiple rooms result on one side

Figure 4.2.C exhibits another kind of multiple room generation. This time, there are two rooms on both sides of the scene. Now the environment forms a corridor of offices.

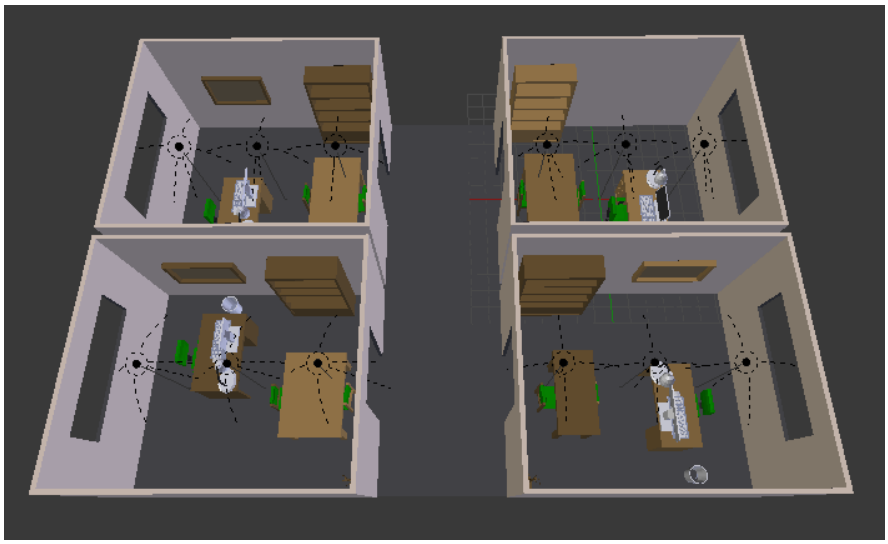


Figure 4.2.C: Multiple rooms result on both sides

Figure 4.2.D shows a single room setting in four different executions. Light effects are active as well as wood texture is visible. These are generated from the same grammar using the probability values in it. As a result of this, some elements change or disappear. For instance, the type of chair and working desk changes from one generation to another, and supplies such as monitors, keyboards and mouse peripherals do not appear in second and last execution.

The color of chairs and wood remains intact along all iterations. This does not imply monotony, as the algorithm changes the intensity of them in every execution. In addition to this, it creates two replicas of the already predefined wood texture and distributes them randomly to the objects which have this texture assigned. These replicas consist on darker and lighter tones of the color of the main wood texture, also with unique wood patterns. This way, the furniture will not be composed with the same kind of wood on every generation.

Window holes are inspired in the window shapes found in the offices of our Center.

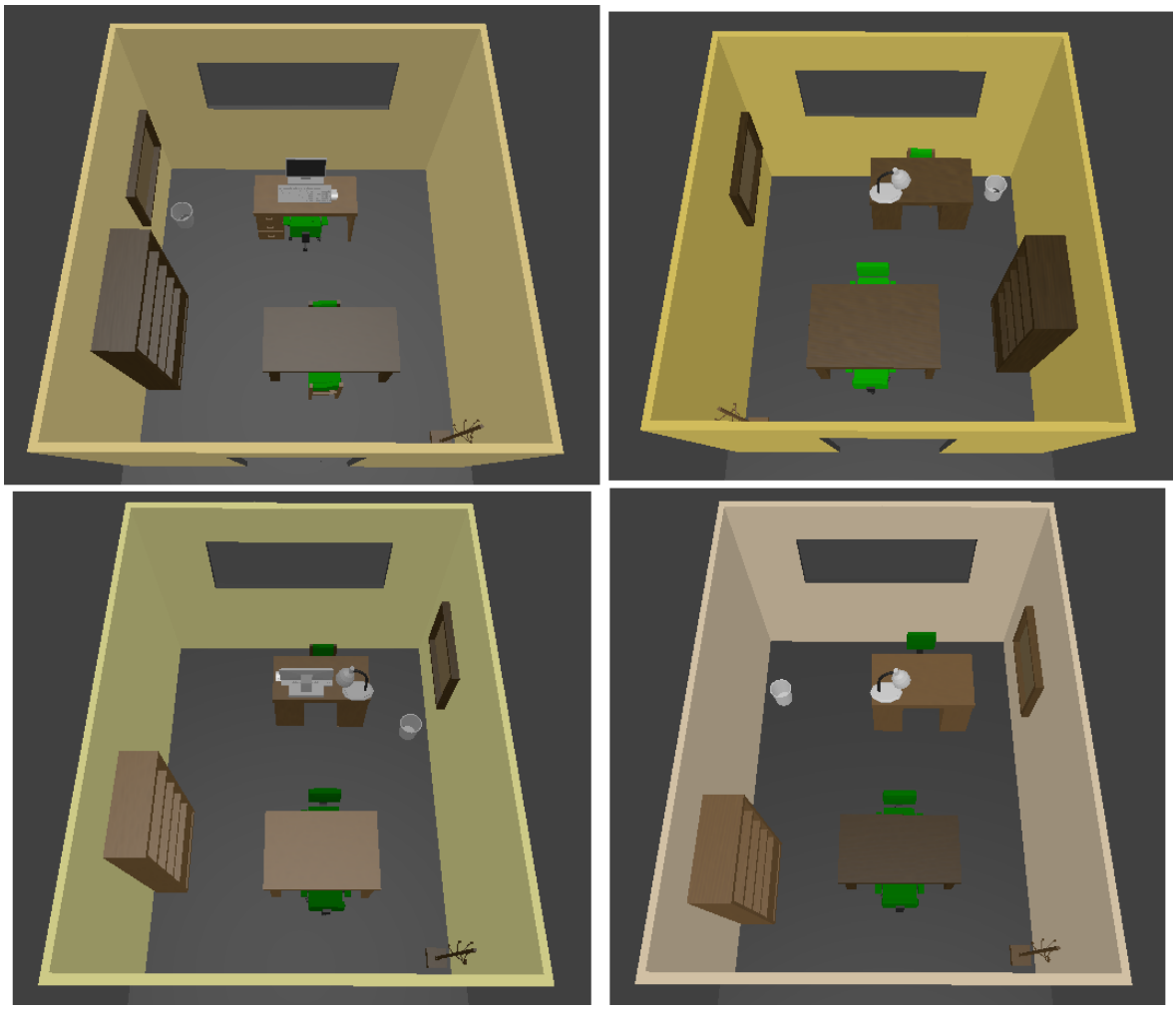


Figure 4.2.D: Single room generation in four different executions

Figure 4.2.E is an example of how the noise parameter affects the chairs arranging. The rotation and displacement of them increases as the value of this parameter is higher. This provides realism to the environment, as chairs are rarely arranged perfectly in real world offices.

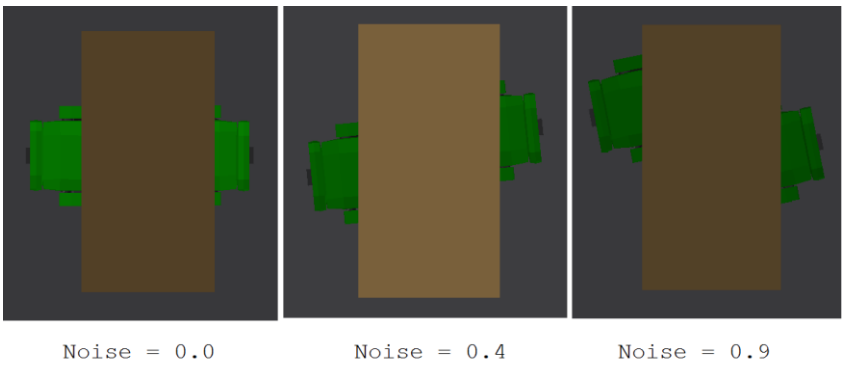


Figure 4.2.E: Noise value comparison

4.2.2 Robot simulation

The robot simulation was conducted using the environment shown in Figure 4.2.F. It consists of three rooms on both sides, with a noise parameter of 0.4 for each room. The colors of chairs and wood were the same as the predefined values.

Figure 4.2.F: Robot simulation environment

The simulation consisted in putting a virtual wheeled robot provided by MORSE into the previously mentioned environment (Figure 4.2.G). Robot direction was controlled by the keyboard, and it was lead about the surroundings for one minute and forty-five seconds. An external middleware to ROS (YARP) was connected to the simulator to acquire the image published by the robot's camera in real time (Figure 4.2.H). This way, we obtain the robot perspective to have an idea of the approximate range of vision the robot could have in real life.

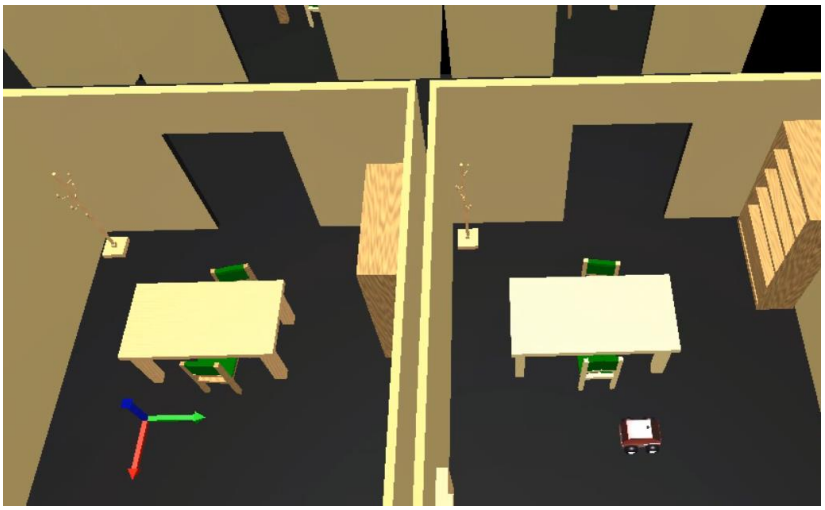


Figure 4.2.G: MORSE simulation

Figure 4.2.H: Robot camera image

There were no issues in the integration of the environment into the simulator tool, nor were in the operating of the robot. It could be moved freely throughout the simulation. It has been demonstrated the full compatibility of the procedurally generated scene with robot simulation tools. These kind of indoors environments are useful to train artificial vision systems for assistant robots or path planner algorithms for surveillance units. As small-size offices in real life do not differ excessively from these models, the adaptation of the robot to real-life environments is coherent.

CHAPTER 5. CONCLUSIONS AND PROPOSALS

In this chapter we will draw some conclusions of the work developed as well as some future work.

5.1 CONCLUSIONS

The goal of this work was to generate a virtual environment using procedural graphics techniques for robot simulation purposes.

From the point of view of graphic – generation, we can say that the results are adequate, as they represent a typical center office distribution or a small-size company office setting. They do not exceed in exhaustibility, but this can be considered acceptable since no high-detailed models were needed. This is reasonable, for the reason that the scenes are used in simulators that render their environment in real-time.

That also applies to textures and materials. They are adequate to represent the essence of each element's nature. Bevel or high-quality shadow effects are not implemented, due to the render engine used in the simulation. There is no need to add extra visual features if the emulator will not take advantage of them.

This work uses L-System as the procedural generation technique for building the rooms. This system is employed in a different point of view. Instead of mapping operations for crafting geometry from scratch, it takes pre-defined elements. This saves a significant amount of time in each generation step.

Moving right along to the simulation part, a robot was put into the scene successfully. It has been demonstrated that the environments created can adapt to robot simulation tools. While that is true that only one type of software was used, this is not a hindrance for other ones thanks to Blender's flexibility in file format exporting.

5.2 FUTURE WORK AND POSSIBLE PROJECT EXTENSIONS

As this project focuses on making a small scene serving as a base, these are some development thoughts for further progress:

- Including more objects in the scene: adding more elements inside the rooms, such as new furniture or some additional modifications of the current set, as well as an interesting set of props and supplies.
- Extending the room replication: taking the idea of generating a row of offices and expanding it to craft a whole office flat or even an entire building.
- Experimenting with more robot simulators: testing the environment with even more simulators and new file formats.
- Optimizing generation process: given that an image file needs to be baked and written for every scene, the production time raises. It could be interesting finding more alternatives to this step.

BIBLIOGRAPHY

BOOKS AND ARTICLES

[Aristid] Przemyslaw Prusinkiewicz, Aristid Lindenmayer. The Algorithmic beauty of Plants. Springer Science & Business Media, 6 dec. 2012

[Aurenhammer] Franz Aurenhammer, Rolf Klein. Voronoi Diagrams. FernUniversität in Hagen

[Burgali] Davide Brugali, Jan Broenink, Torsten Kroeger, Bruce MacDonald. Simulation, Modeling, and Programming for Autonomous Robots. Springer, 19 sept. 2014

[Ebert03] David S. Ebert. "Texturing & Modeling: A Procedural Approach". Morgan Kaufmann, 2003.

[Lewis] J. P. Lewis. Is the Fractal Model Appropriate for Terrain? Disney's The Secret Lab.

[PAAM] Public Affairs, Administration and Management: Impact of Office Design on Employees' Productivity. A Case study of Banking Organizations of Abbottabad, Pakistan. Volume 3, Issue 1, 2009.

[Voronoi] G. M. Voronoi. Nouvelles applications des paramètres continus à la théorie des forms quadratiques. Deuxième Mémoire: Recherches sur les parallélloèdres primitifs. J. Reine Angew. Math. 1908

WEB LINKS

[Activluton] Home Interior Design Ideas: Executive Office Layout Design http://activluton.com/executive-office-layout-design/ (Last access on 27th May 2016)

[Alkiloo] Alkiloo http://www.alkiloo.com/ (Last access on 27th May 2016)

[AB] AllBusiness: Office Supply Checklist. https://www.allbusiness.com/office-supply-checklist-3779097-1.html (Last access on 27th May 2016)

[Blender] Blender Foundation https://www.blender.org/about/ (Last access on 27th May 2016)

[Engadget] Engadget: Here's how 'Minecraft' creates its gigantic worlds. Jon Fingas, March 3, 2015 http://www.engadget.com/2015/03/04/how-minecraft-worlds-are-made/ (Last access on 27th May 2016)

[Massive] MASSIVE http://www.massivesoftware.com/index.html (Last access on 27th May 2016)

[MORSE] MORSE – Openrobots https://www.openrobots.org/morse/ (Last access 25th June 2016)

[Python] Python https://www.python.org/ (Last access on 27th May 2016)

[Robologix] Robologix https://www.robologix.com/ (Last Access on 25th June 2016)

[Speedtree] Speedtree http://www.speedtree.com/ (Last access on 27th May 2016)

[SR] SmashingRobotics http://www.smashingrobotics.com/ (Last access on 25th June 2016)

[StackOverflow] Stack Overflow http://stackoverflow.com/ (Last acces on 7th June 2016)

[TCA] The CRPG Addict: Game 79: Beneath Apple Manor (1978). December 9, 2012 http://crpgaddict.blogspot.com.es/2012/12/game-79-beneath-apple-manor-1978.html (Last access on 27th May 2016)

[TDA] The Digital Antiquarian: Akalabeth. December 18, 2015 http://www.filfre.net/2011/12/akalabeth/ (Last access on 27th May 2016)

[TMWS] Top Minecraft World Seeds http://topminecraftworldseeds.com/triple-crater-caverns (Las access

27th m May ay 2016)

[Wikipedia] Wikipedia, The Free Encyclopedia https://en.wikipedia.org (Last access 7th June 2016)

[WSJ] The Wall Street Journal: Why Borderlands 2 Has the Most Stylish Guns in Gaming. Ryan Kujo, April 19, 2012 http://blogs.wsj.com/speakeasy/2012/04/19/why-borderlands-2-has-the-most-stylish-guns-in-gaming/ (Las access on 27th May 2016)

ANNEX I: CD CONTENT

We can find the following resources on the CD content which accompanies this document:

- PDF format document inside the *Document* directory.
- Source code, grammar file and Blender base file inside the *Source* directory.
- Grammar usage manual inside the *Manual* directory.
- Some examples of generated scenes in Blender and image file formats, as well as a video of the robot simulation inside the *Examples* directory.

ANNEX II: EXECUTION DETAILS

This annex explains the requirements and process steps to successfully generate an office environment.

This system is optimized to work only in Linux operative systems.

Before starting, all the needed files must be inside the same directory. These files are the Python scripts, the grammar file and the Blender base file. Otherwise, the system will miss parameters and libraries. The 2.7.1 release version or a newer version of Blender tool must be installed.

If these requirements are met, we can proceed to start the Blender file. It has to be launched using the following console command inside the same directory the Blender file is, as executing it from the file explorer could result in failure:

```
blender Scene.blend
```

A similar interface to Figure II.A will be shown. It consists in a *File Editor* area with the main script loaded by default, as well as a *3D View* area, *Python Console* area, *Mesh Selection* panel and *Properties* panel.

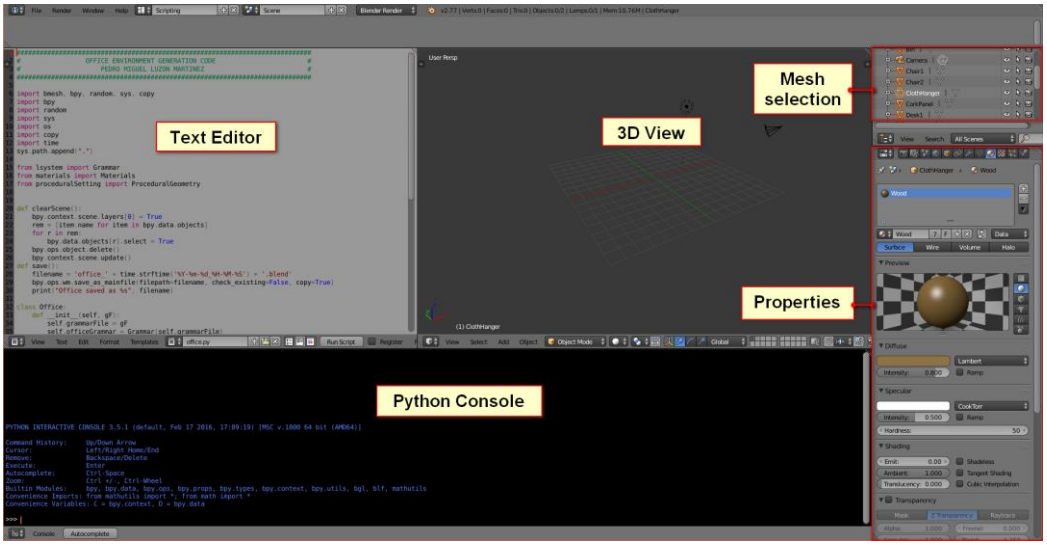


Figure II.0.A: Blender file interface

To start generating an environment, click with the right mouse button over the *File Editor* area and select *Run Script*, or simply place the mouse over the *File Editor* and press Alt-P. The progress is displayed in the console that was used to launch the Blender file. When the process is finished, the resulting scene will appear inside the *3D View* area.

A Blender file is automatically saved inside the same directory when a scene is fully generated. It is named after the day and hour when the generation was finished. Three wood texture images are saved as well, which correspond to the generated wood materials and patterns. These four files need to be in the same directory for the scene to be re-opened.

It is not necessary to close and restart the Blender base file every time a new office environment needs to be generated, as it clears the scene before proceeding. It is neither necessary if the grammar file has been changed between executions, as it takes the current version of the grammar file. Simply assure that the grammar changes are saved before re-executing the script.

The wood material tab is open in the *Properties* panel by default. To change the color of wood material, click on the *Diffuse* color property and select a new tone (Figure II.B). To change the color of the chairs, select one of the chair meshes (either *Chair1* or *Chair2*), select the *Color* material and click on the *Diffuse* color property (Figure II.C). These parameters need to be set before generating a new scene. To go back to the wood material, select a mesh that uses it (such as *ClothHanger*). As the procedural model generates two replicas of the original wood material (named *Wood*) after an execution, modify the Diffuse color property of this material and leave the others intact.

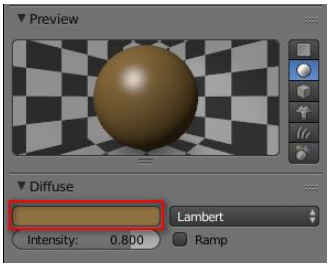


Figure II.0.B: Wood color modification

Wood texture adapts to the defined Diffuse color. However, it is optimized for wood-like colors. An odd color tone (such as green wood) could produce undesired or strange effects.

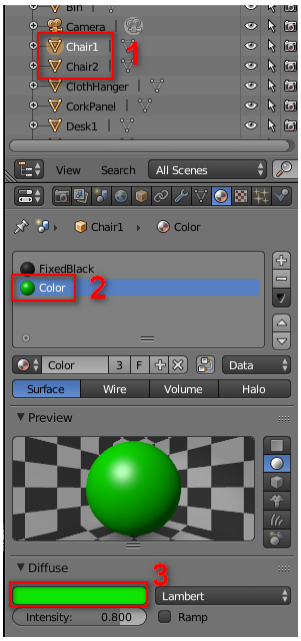


Figure II.C: Color material modification