Dissertation for the award of:
Master of Science in Games Programming
The University of Hull

Towards the Procedural Generation of Urban Building Interiors

Benjamin Bradley

August 2005

0 Table of Contents

1 Abstract

Computer games which are staged in entire, three-dimensional virtual cities are becoming common. But manually designing such an enormous environment is a huge strain on resources, and as such the buildings in these cities usually exist only as impenetrable external facades.

This paper discusses a practical implementation of a system which, given the exterior shape of a building, uses a procedural algorithm to automatically create a building floor plan and then transform this into a three-dimensional building interior model. The system is most basic, but offers a practical first step towards the very attractive possibility of procedurally generating the interiors for all the buildings in a city based game.

2 Introduction

2.1 Document Structure

Chapter 3 positions this work amongst existing works of research and commercial computer game production techniques. Chapter 4 justifies the research direction taken in this project, that being one of prototyping.

Chapter 5 discusses the initial aims, design and schedule of the project. Chapter 6 then looks more closely at the way in which this design has been implemented, and how and why it differs from that initial plan. The method by which the procedural design algorithm operates is covered in some detail.

With hindsight, Chapter 7 provides an analysis of the results of this project; criticising both the output of the developed software and the development process itself. Chapter 8 considers future research directions and possible improvements to the software. Chapter 9 then concludes the document and reassesses the subject of procedurally generating urban building interiors for computer games.

Chapters 10 and 11 provide a glossary and list of references respectively.

2.2 Definitions

Throughout this document, the term *PICAG* will be used as an acronym for *Procedural Indoor City Architecture Generator*. Further definitions will be introduced within the document structure. Other more commonly accepted acronyms and definitions can be found in the Glossary (Chapter 10).

3 Background

This chapter of the document positions this research work within existing fields of research and commercialism. Section 3.1 introduces procedural content generation and its place within past computer games. Section 3.2 discusses existing relevant works from outside the games industry. Section 3.3 combines these to form a justification for this research project, and section 3.4 concludes.

3.1 Existing Procedural Content Generation in Computer Games

3.1.1 An Introduction to Procedural Content Generation

The *media* of a computer game is the part of the game data which is not computer code. This can include 2D textures, 3D models, environments, animations, sound effects and music.

Procedural content generation is a process of designing this media using computer algorithms rather than directly applying human resources[1]. Such techniques hold a number of useful advantages. Perhaps the most significant of these is that by applying random parameters, one algorithm can often be used to produce a truly enormous amount of content with no further effort.

However, in current computer game production techniques the vast majority of a game's media is still created by a team of artists, musicians and designers. While a procedural generation algorithm has the advantage of quantity, the amount of differentiation between possible outputs is related to the complexity of the algorithm.

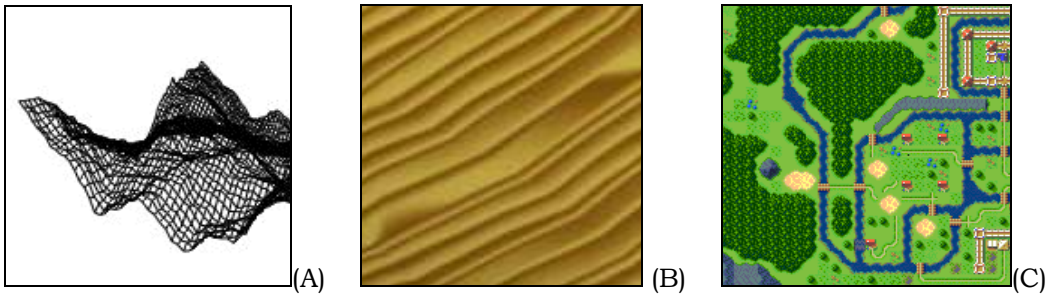
A greater problem is that the algorithm itself is the only link between the human-level design phase and the output. The decisions a designer would make are either reconstructed in code form, or some approximate rule is implemented which produces results that "seem right". Hence such an algorithm often fails to offer an intuitive description of the design, or an obvious correlation with the output… or both! This problem becomes more pronounced as the need for complexity increases.

Deciding whether procedural or manual design is better for any application is very often a question of quality versus quantity.

[1] Except of course, the human resources which are required to create the algorithm.

3.1.2 Common Forms of Procedural Generation in Computer Games

The chief use of procedural generation techniques in computer games of the past has been the creation of 2D maps. This includes height maps (Figure 1A), texture maps (B) and tile maps (C) which are images composed by a grid of smaller images.



(A) (B) (C)

Figure 1 – Procedurally Generated Game Content

These images and environments lend themselves easily to procedural generation because there is only need for one rule – a calculation which takes the x and y location on the grid as its parameters in order to choose the height, colour or image to put in that grid cell. This is called *serialized* procedural generation [1].

Algorithmically generated music [4], animations (such as fire) [13] and plant-life [9] have also been used with success in computer games, though these techniques are not yet widespread.

3.1.3 Procedural Content Generation in the Future

Computer games technology is undoubtedly moving rapidly, and rising team sizes is a familiar trend that looks set to continue. For those who create game artwork, the required level of detail is increasing as computer graphics hardware and software moves forward, but there is also an increase in their workload because the scale of computer games (i.e. the size of their virtual environments, and the length of play time they offer) is also increasing.

Therefore, any preference towards using procedural or manual generation for any particular task must be constantly reassessed. Although algorithmic design is becoming

less popular in areas such as terrain creation, procedural and semi-procedural[2] generation techniques are proving useful in creating other types of media, or specific instances within a type of media (e.g. all the brickwork textures). Research into procedural design techniques is therefore beneficial to the computer games industry.

3.2 Existing Work towards Procedural City Building Generation

3.2.1 Building Exteriors and Cities

Research into procedurally generating city exteriors has already met a great deal of success. *Procedural Modeling of Cities* [5] gives a good example of what can be achieved: a realistic road network which takes terrain variations and bodies of water into account, and within this network, buildings which occupy varying floor spaces, and clearly represent different types of building through their shape, size and location within the city (Figure 2).

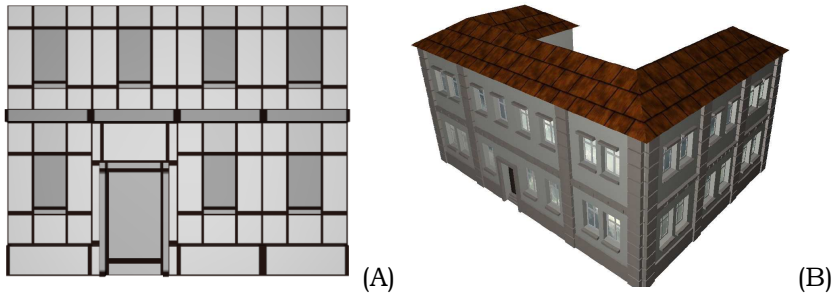
Figure 2 – A City Generated by Software Discussed in [5].

Procedural design is well suited to this because cities are self-similar and orderly. The city generation process in [5] is made more intuitive because it can be broken down into a number of discreet stages which use the output from the previous stage as their input; network of roads, allotments, buildings, and finally textures.

For each of these stages a number of methods are available. *Real-Time Generation of Pseudo Infinite Cities* [3] discusses the combination of a serialized system and a simple polygon clipping algorithm to create interesting building shapes, though this design has

[2] See glossary.

the disadvantage that the roads form an evenly spaced grid. *Procedural Modeling of Cities* [5] represents its rules with *L-Systems*[3] in both its road and building designers.



(A) (B)

Figure 3 – Building Fascias Generated by Software Discussed in [7].

Elsewhere, research projects have also focused more specifically on the generation of more detailed building exteriors for single buildings or small neighbourhoods. In *Instant Architecture* [7] the system which created the detailed building exteriors as seen in Figure 3(B) is discussed. This system is built around repeated subdivision of each facade through a system of rules called *split grammars*. Attributes associated with the design are passed down throughout this process.

Systems for semi-procedural work also exist which mix procedural generation with something which resembles a more traditional 3D modelling package. However, this imposes limitations on both the procedural and manual design elements. Either the procedural and manual design processes apply to distinct parts of the overall design or the modelling package allows the user to design using flexible parameters rather that rigid values, as in [8].

3.2.2 Floor Plan Generation

Work towards procedurally generating the external and internal structures of buildings has been largely independent. The visualisation of a complete procedurally generated city is instantly impressive and hence more valuable from an entertainment perspective, but such systems are inappropriate for town planning – real cities tend to grow rather than "be designed".

[3] An L-System is a string rewriting system, in which substrings are expanded by substitution. [9] provides a good introduction and examples of use.

Research into the generation of floor plans and building designs has grown to a mature stage throughout the seventies and eighties. The most significant research now takes place within the companies who develop such tools to aid architectural designers.

The focus within publicly available research has always been towards semi-procedural systems – those which aid the designer and simplify repetitive tasks, rather than automate the entire process. The most common subject is of space optimisation – finding the most preferable relationship between the available rooms, and their aspect ratios, sizes, locations and access points [10], [11].

In *Infinite Game Universe: Level Design, Terrain and Sound* [4], Lecky-Thompson discusses the procedural generation of floor plans for computer games levels. However, his focus is on the generation of buildings which closely resemble mazes – most modern computer game environments do not follow such a simple structure. To the author, this is the only known work on procedurally generating floor plans specifically for computer games.

3.2.3 Generation of 3D Visualisation from Floor Plan

A natural extension of floor plan design is to be able to visualise the architectural design in 3D [6] in order to allow architectures and clients to comprehend the designed space more clearly. Modern CAD packages are intuitive enough to allow families to model extensions and renovations of their own homes using a combination of simple 2D and 3D interfaces.

3.2.4 Previous Work

A past work of this author – *Procedural Generation of Architectural Game Environments* [12] – began to consider the design and development of an algorithmic system which replaces the manual architecture design process. Although a practical implementation was developed in association with this work its results were inclusive with respect to the themes discussed in the paper and the document remains largely a theoretical piece which discusses the possibilities of procedural architecture generation and the major problems faced. It goes on to suggest a number of possible solutions to those problems in the form a very generic system design for a procedural architecture generator.

This work continues this research, but has a number of crucial differences to the aims in [12]. Firstly, this work focuses on a specific kind of architecture (city building interiors) which lends itself relatively easily to procedural generation for reasons discussed later. Secondly, the system presented in this document is intended to work *with* a manual

design process, rather than *instead of* it. But perhaps most importantly, this project is practical and experimental; the aim being to derive useful results and conclusions from an implementation, rather than create an implementation to support a larger theory.

3.3 Motivation and Area of Application

3.3.1 City Based Games

Due to many factors including the rising expectations of computer game players, the availability of DVD storage, and the increasing size of game teams and profits, 3D game environments have risen to the scale of entire cities. In particular these have been popularised by the *Grand Theft Auto* games series, with each of [G1], [G2] and [G3] replacing it's prequel as the fastest selling game of all time. Such city environments are manually designed, and due to their scale must sacrifice some geometric detail when compared to games set in smaller environments.

Figure 4 – Images from City Based Games

(A)


(B)

(A) A scene from *Grand Theft Auto: Vice City* [G2].

(B) A scene from *Spider-Man 2* [G5].

3.3.2 Motivation

Due to the truly enormous workload that would be created it is impossible for these city buildings to be anything more than external fascias. Although the player is usually in control of a human character within these virtual environments, they are only able to navigate within an exceptionally small number of buildings[4] which have been specifically designed as such. The rest simply exist as physical barriers, with doors which do not function and windows which do not open.

[4] Certainly less than one percent.

City building interiors demonstrate a great deal of organization and repetition, and their design can be thought of in terms of a number of largely independent stages. The number of rooms in a typical city is in the millions. The creation of building interiors therefore seems ideal for procedural generation.

Further potential exists when one considers that each floor of a building is hidden from view by the building's outer walls and the other floors. This means that the building interiors do not need to be stored – it may be possible to generate and regenerate the building interiors at run time, as and when they are required.

3.3.3 Architectural Realism

In *The Role of Architecture in Videogames* [2], Adams suggests that game architecture is fundamentally different in purpose to its real-world counterpart. For example, privacy and shelter from the weather are usually irrelevant in computer games, while constraint, exploration and concealment are more common design considerations. However, another purpose of game architecture may be to provide visual familiarity; cities are an example.

The existence of procedural algorithms within architectural tools shows that this goal is very possible, and there are certainly many useful techniques and algorithms to be found. However, it is not feasible to copy techniques blindly from architectural design software. Urban game environments should appear similar to real cities, but are designed for a different purpose.

3.4 Chapter Conclusion

Procedurally generating urban building interiors for computer games seems a task perfectly suited to procedural development. A large and increasing number of games exist which could benefit from such a system. Existing architectural design software makes use of procedural algorithms which can be drawn upon and demonstrate that it should be quite achievable.

4 Methodology

In the last chapter, research into PICAG design and development was justified. This chapter considers the best means for shedding light on the problems faced by PICAG design. Section 4.1 considers the most important unknowns regarding PICAG design and development, and section 4.2 explains how these have affected the choice of methodology.

4.1 Research Topics

The methodology of research has been chosen based on what are considered to be the chief questions and problems currently posed by PICAG design and development:

4.1.1 Development Time

A PICAG would prove to be worthwhile if its development time undercuts the time it takes to manually design the required quantity of media. An informal consideration of the area of application offered in the previous chapter will certainly suggest that this is the case. If we estimate that there are one million rooms in a city and each take half a day to manually design, the city's interior will take two thousand person-years to create.

Creating a PICAG which produces environments satisfactory for commercial games is still a task too large for this project, and so answering this question has not been a direct target.

4.1.2 Independent Stages

In [12] this author suggested that dividing a procedural architecture generator into a number of largely independent stages was beneficial. This provides an organisation which clarifies the relationship between the design process and the algorithm, and clarifies the algorithm within itself. We must ask how the generation of indoor environments can be divided into stages most aptly.

4.1.3 Development of a Custom Language

It appears that no work has yet been done towards creating a language specifically for designing architecture for use in computer games. Different languages may be necessary for different stages of the algorithm. This project initially aimed to tackle this issue. [12]

identified the careful design of custom scripting language(s) as a major factor in determining the practicality of procedural architecture generation.

4.1.4 Spatial Awareness

Another of the chief problems identified by [12] is the problem of spatial awareness. For example, doorways must be placed within walls, objects must not obscure windows, furniture must not block access to locations, certain rooms and objects should be positioned adjacent to each other.

[12] describes how the quantity of relationships between objects and architectural elements grows exponentially as the complexity of a PICAG increases, and how this creates enormous problems to the processes of describing the design rules, and testing the system. Indoor city architecture has been selected specifically because it simplifies the spatial awareness problem. This will be discussed in more detail in chapter 5.

4.2 Methodology Used

This research project has been highly implementation-oriented. The reasons for this choice are summarised in this section.

However, in actual fact the choice of methodology was not considered critical. Other than the author's own past work in [12], no documentation or existing work on procedurally generating architecture specifically for computer games has been found. This topic is young and although the methodologies discussed herein were not favoured at this time, it is believed they too would have brought forward useful information.

4.2.1 Research Approach

Works which would be most valuable learning tools when designing a PICAG are the professional tools being used for semi-procedural architecture generation. But as discussed in section 3.2.2, development of these systems is now commercialised and this does not appear to be an area of current public research.

The initial research period of this project unearthed many documents, the most significant of which have been referenced in chapter 3. Their subject matter and age are sporadic. Indoor architecture generation (for entertainment) and outdoor architecture design are almost totally independent fields. Further research work may have continued to bring up related material which can be learnt from, but ultimately this approach was

not considered best since it seemed unlikely that a solid PICAG design could be composed through research alone, due to these complaints.

4.2.2 Design Approach

Creating a theoretical detailed design of a PICAG system would prove useful at this stage of research. [12] suggested a design which is generic enough to cover any kind of game architecture. This can be enhanced to provide a more detailed and specialised design aimed at city building interiors. Such a design would give an indication of the development time (4.1.1), independent design stages (4.1.2) and the design of custom scripting language(s) (4.1.3).

However, its practicality would remain unproven. Not least, the suitability of the designed scripting languages is difficult to prove given the unruly nature of the rule sets which they are intended for. A detailed design can not give a sense of the complex relationships between geometrical elements (4.1.4).

4.2.3 Prototyping Approach

Instead this project has taken a prototyping approach, the aim being to develop a simple PICAG in the allocated time and deduce any lessons from the development that can be made. Naturally, the initial design was constructed in order to direct development towards answering the key questions in section 4.1. This design is discussed in the following chapter.

5 Initial Project Design

Due to a strict timeframe, it transpires that the software developed did not fulfil initial expectations. It will be useful for the reader to consider the project's original aims, design structure and time plan, as the reasons why this project proved to be over-ambitious are behind some of its most influential conclusions. Section 5.1 gives an overview of the features in the target PICAG software, section 5.2 describes some code resources used in development, and section 5.3 shows the original development schedule.

5.1 Project Aims

The target software is best described in terms of its input, internal process, and output.

5.1.1 Input

The program would take two inputs. The first of these is a set of ASCII files containing a description of possible environmental features to be included. Each file would contain the heuristics for one stage of the design process (see section 4.1.3). Due to time constraints (as well as the purpose of experimentation) this language would be purely definitional.

The second input to the algorithm is created within the application. Here, the user would draw the plan view outline of a building, and dictate how many floors it has.

5.1.2 Process

The design process would be hard-coded, and broken down into four stages:

- Floor Stage: The building outline is extrapolated into 3D and divided into numerous floors. A part of all the floors is reserved for a stairwell and/or lift shaft.
- Room Stage: Floors are divided by walls into corridors and rooms.
- Negative Stage: Windows and doorways are removed from walls.
- Furniture Stage: 3D furniture models which have been pre-designed in a modelling package are positioned at appropriate points in appropriate rooms.

These stages were selected because of their physical boundaries. The floors of a building are physically divided. The rooms of the building are also divided by walls. Hence if an object is placed within a room we know that it will not interfere with the geometry in any other room or on any other floor. In this way, the design stages become largely independent.

There are times when these boundaries are virtually broken – for example we may wish to show repetition across a number of bedrooms within a hotel. This kind of information exchange is an exceptional extension of what is otherwise a procedural design process of independent stages, and has not been a goal in the development of this prototype.

5.1.3 Output

The target output was a three dimensional building which can be navigated in a first-person walk-through mode. The building would have a practical network of corridors, lifts and stair cases which allowed access to any of the rooms. The level of visual detail would be most basic for this purpose. Greater variety would be achieved in the output buildings by expanding the definitional rule set.

5.2 Resources

Part of the research portion of the project was spent searching for existing code libraries which could be useful, but none found were directly applicable to the design. However, this author had previously created two useful works:

- A constructive solid geometry algorithm which can be used to find the union, difference, and intersection of two convex 3D blocks. The results of the algorithm rendered unreasonably slowly, and produced erroneous results in certain cases.
- An application for creating simple urban outdoor environments. This program was complete and bug-free, and featured saving and loading routines. However, modification would once be again necessary, in this case to change the floor plan input and representation from triangles to polygons.

It appeared that both of these applications could prove useful in development; the first for removing solids from one another, e.g. taking windows and doorways out of walls; and the second as a means of inputting building exterior shapes.

5.3 Initial Time Plan

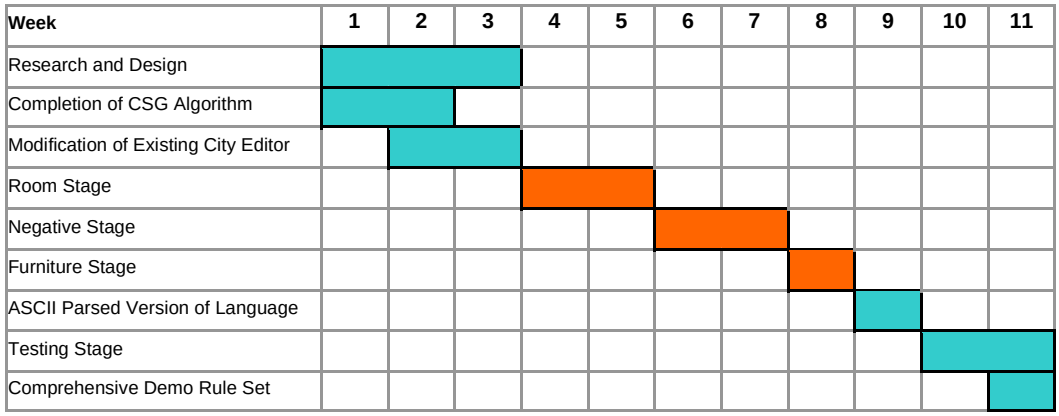


Figure 5 – Initial Development Schedule

Figure 5 shows the initial development time plan. Tasks in orange indicate the development of the code which forms the core PICAG process. Tasks in green indicate the development of supporting code required to develop these process, or other non-programming tasks. Since many tasks have not been implemented, it would be unproductive to articulate this entire design in detail.

6 Implementation

Chapter 5 has given an overview of the original project design. This chapter describes that part of the design which has actually been implemented in greater detail. Section 6.1 uses a class diagram to show the key features of the static program structure, and section 6.2 uses a program flowchart and images output by the program to give an overview of the PICAG design process, and its path of execution.

The following sections examine particular parts of the implementation more closely. Section 6.3 discusses the development of two different CSG algorithms. Section 6.4 provides an in-depth discussion of the room placement strategy, which is at the heart of the developed software.

6.1 Static Structure

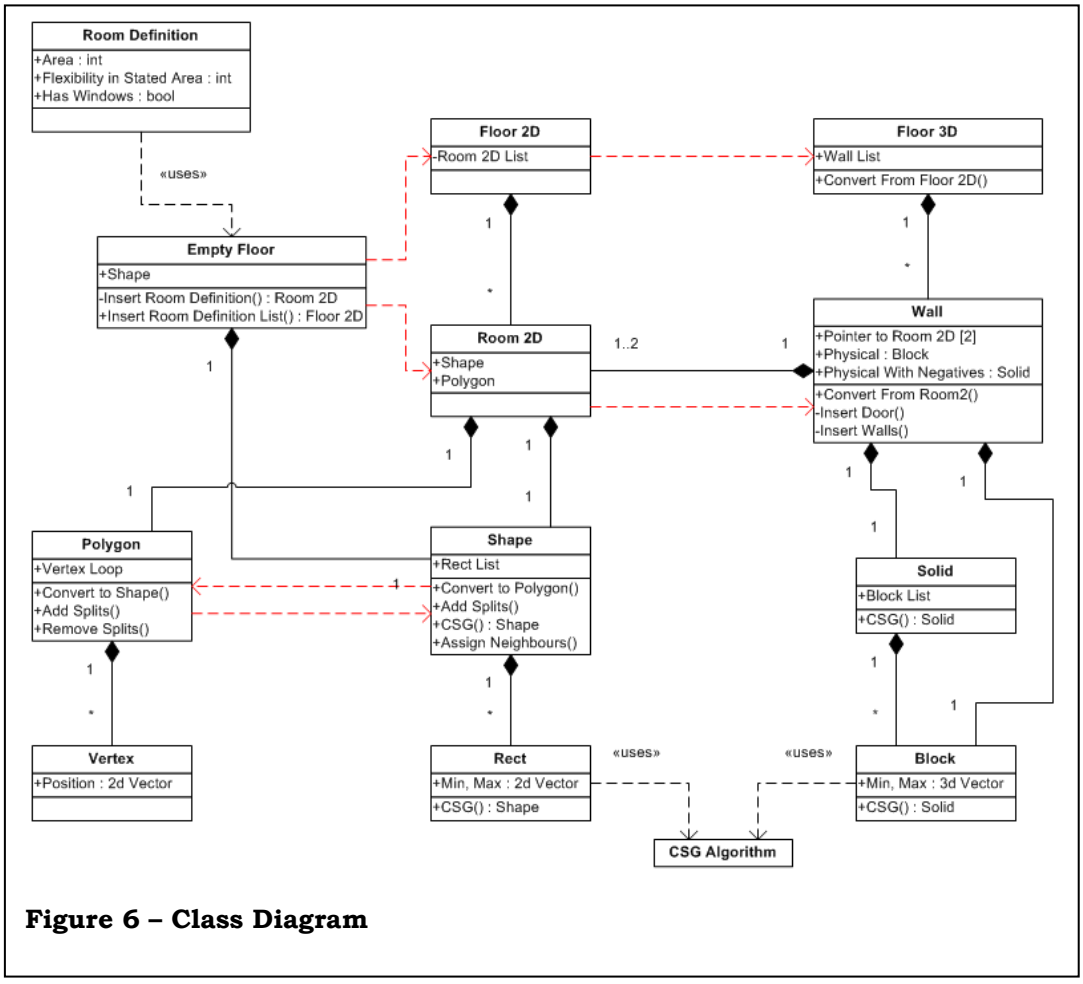


Figure 6 – Class Diagram

Figure 6 shows the main classes within the developed software and their most fundamental attributes and functions. In addition to this, a library of basic maths classes exists along with code to handle rendering and input. Apart from `Room Type`, all the classes shown in this diagram contain a function called `Render()`, which allows the visualisation of the class contents. `Shape`s are composed of `Rect`s (rectangles), and are the 2D equivalent of 3D `Solid`s and `Block`s (cuboids) respectively. Red dotted lines show where conversions are possible.

6.2 Execution

The developed software is better understood by considering its path of execution rather than its class structure. After all, it is procedural by nature; and the majority of the design and implementation time has been spent on a few relatively large algorithms. Many of the classes in Figure 6 (particularly `Rect`, `Shape` and `Polygon`) are used frequently and for varying purposes throughout these algorithms.

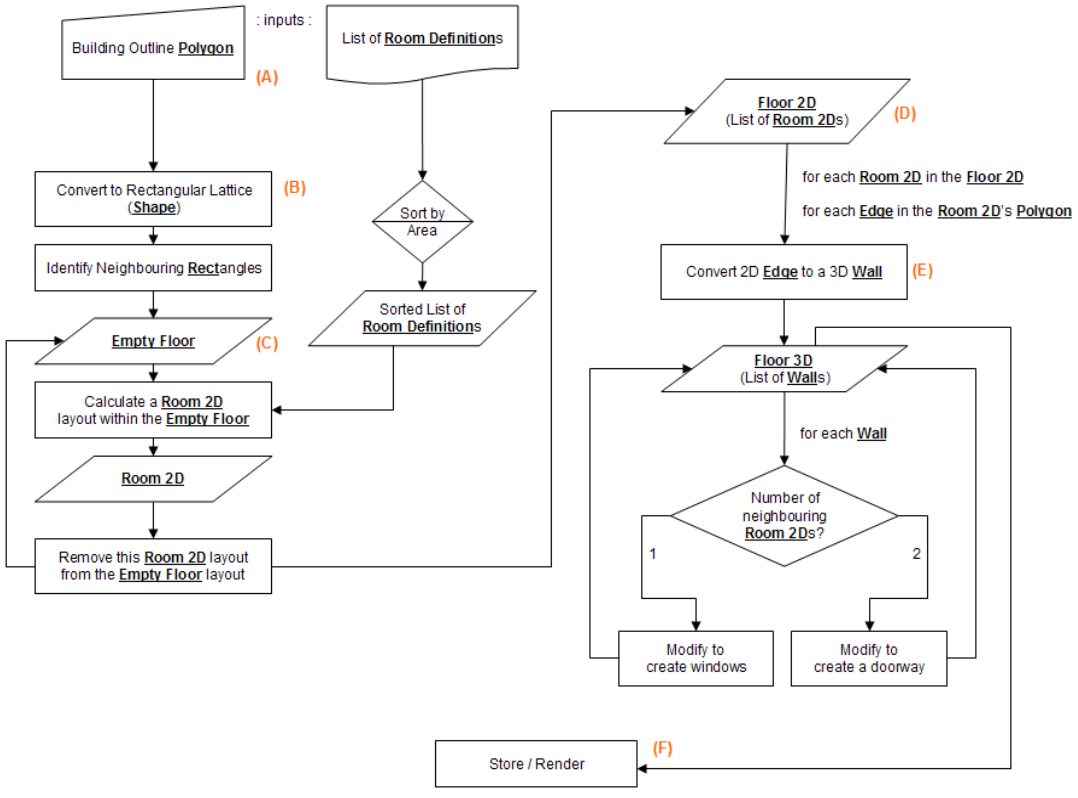


Figure 7 – Flowchart

Figure 7 provides a flowchart showing the steps in the architecture generation process. Class names are bold and underlined. The loop on the far left is executed recursively for

each `Room Definition` in the list. The images by the bracketed orange letters in Figure 8 show the PICAG's visual output when interrupted at that stage of the algorithm in Figure 7 with the same letter.

The following sections describe many of the algorithms shown in Figure 7 in more detail.

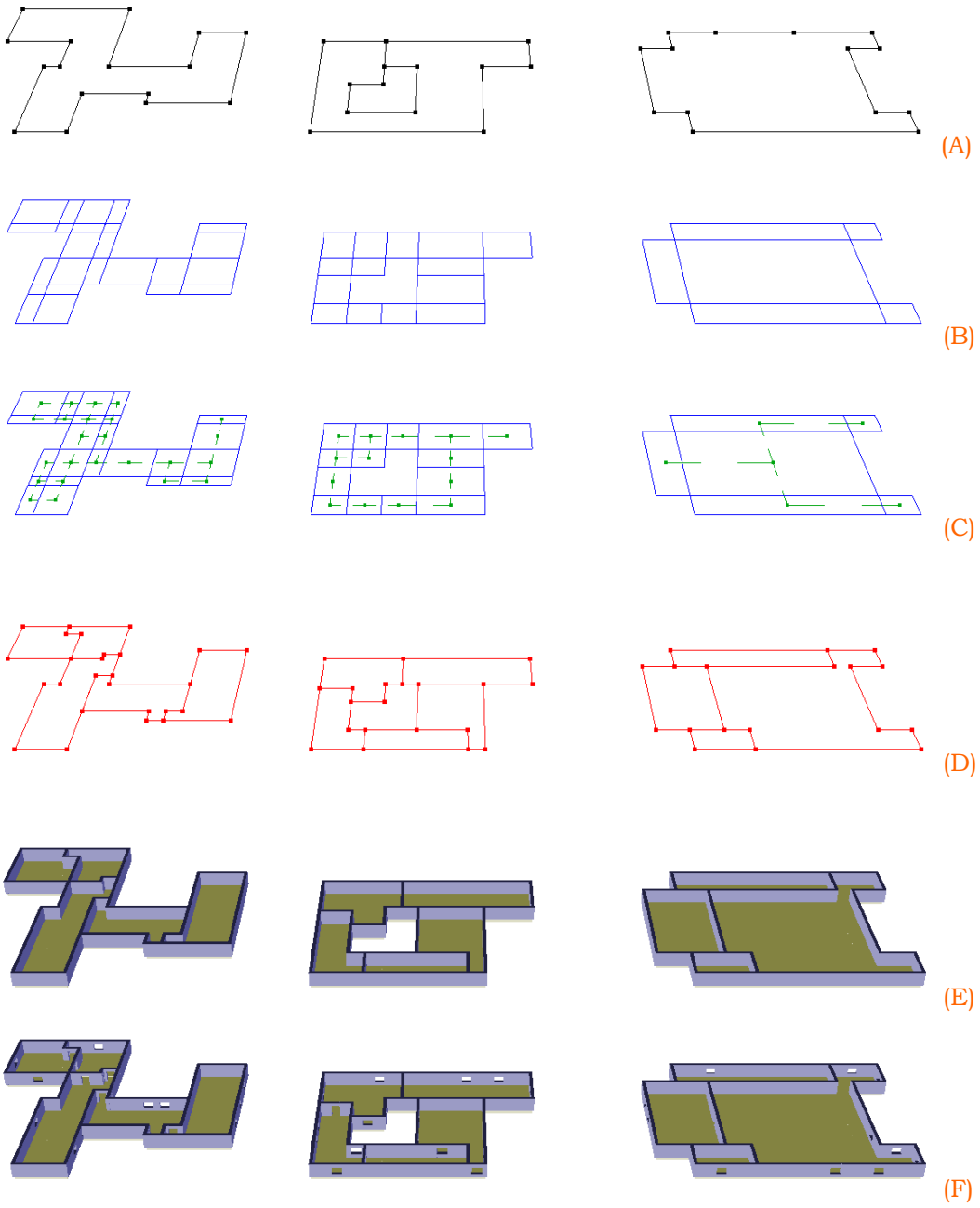


Figure 8 – Three possible visual outputs of the developed PICAG, when interrupted at various stages of its design algorithm

6.3 Constructive Solid Geometry Algorithms

6.3.1 Failed Modification of Existing Code

As stated in section 5.2, it was intended that an existing CSG routine be debugged in order for it to be used in this project. Solving both of its faults (slow render times and occasionally erroneous results) was crucial as the render times for just one CSG result were around one frame per second. It was decided that the render times should be solved first as this would aid the debugging of the second problem.

After two weeks of work, the problem domain had been narrowed down significantly but render times had not been improved. This was simply because code which was written by this author just two years ago now seemed both unruly and unfamiliar.

6.3.2 An Orthographic Implementation

Unfortunately the design relied heavily on the availability of 3D and 2D CSG algorithms[5]. With two weeks work on the modification of old code proving unfruitful, the difficult decision was made to abandon it and attempt to build an entirely new algorithm.

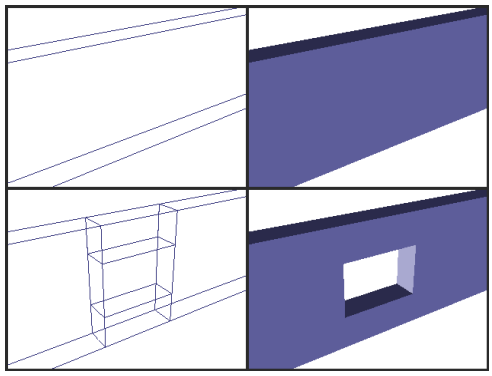


Figure 9 – Orthographic CSG

Left: wireframe. Right: flat shaded. Top: before CSG. Bottom: a cuboid window section removed. The result is in terms of multiple cuboids.

The new CSG implementation worked only with axis-aligned cuboids. This provided a welcome simplification; the convex blocks were now represented by two vectors indicating the minimal and maximal corners, rather than a collection of multi-sided surface polygons.

The restriction which this brought – that all corners within the environment would need to be at right-angles – has not been detrimental for two reasons. Firstly, this is often the case in urban architecture, and secondly, time which had been lost modifying old CSG code was no longer available for allowing the PICAG design process the complexity to handle acute corners.

[5] Developing a 2D CSG (Constructive Shape Geometry) algorithm was not considered difficult, as it is a direct simplification of the 3D problem: one dimension removed.

In hindsight, neither of these decisions are considered wrong. It could not have been foreseen that the existing algorithm would prove so difficult to amend, and the decision to build a simpler algorithm in its place has proved to be acceptable.

6.3.3 Explanation of the Algorithm

An explanation of the CSG algorithm will not be given here since its method is the same as that given in Appendix A in [12]; the only differences being that all faces of the `Block`s are confined to be parallel to the world axes, and that each `Block` is represented and manipulated by the CSG algorithm only by the positions of two corners – those closest and furthest from the world origin.

6.4 Room Placement Strategy

The room placement strategy – the crux of the developed software – is a recursive algorithm. It operates by creating a room shape which fits within the provided floor space, and then using CSG to subtract this room shape from the entire floor shape. This leaves only the unoccupied floor space remaining, allowing the algorithm to place another room into this.

6.4.1 Preparing an Empty Floor Polygon for Room Insertion

The `Polygon` provided by the user is converted to a `Shape` (section 6.5) – a collection of `Rect`s (rectangles) which cover the same region as this `Polygon`.

This conversion process creates a `Shape` in which the edges of the `Rect`s meet, but the corners do not necessarily do so. The `Shape::Add Splits` method subdivides all the `Rect`s in the `Shape` by all of the `Shape`'s `Rect`'s vertices. The result is a lattice where corners can only join to other corners, as in Figure 8B.

Each `Rect` in the `Empty Floor::Shape` is provided with a pointer to its neighbouring rectangles and a flag indicating in which direction that neighbour lies. This is visualised in Figure 8C, and completes the preparation of the `Empty Floor`.

6.4.2 Placing a Room within an Empty Floor

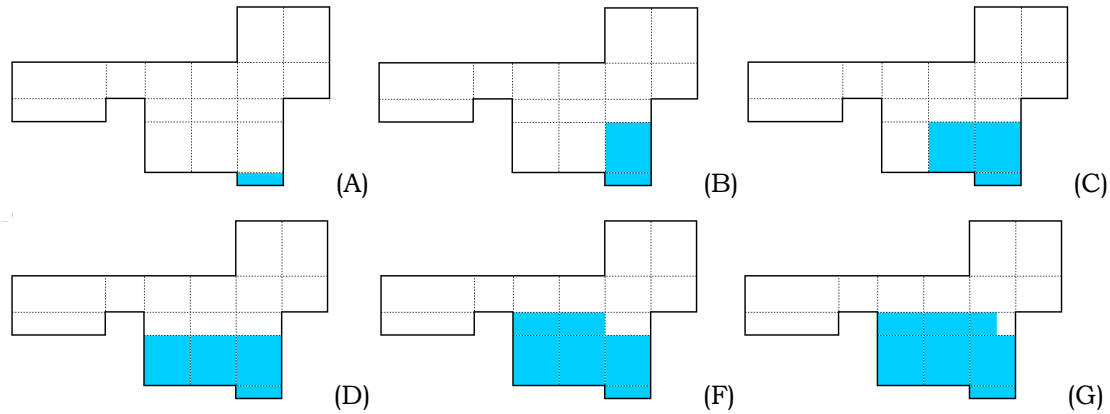


Figure 10 – Filling an `Empty Floor::Shape` with a `Room 2D::Shape`

The goal is now to create a `Room 2D::Shape` which is a subset of the `Empty Floor::Shape`. This is done by considering the area which we require the room to occupy. One `Rect` at a time is added to the solution until this room area is met, as in Figure 10. There is no randomisation in the selection of `Rect`s. The next `Rect` to add to the solution is selected using two rules:

The first priority is to find the `Rect` which has the most neighbours already in the solution. This helps to reduce the number of corners in the final `Room 2D`. It also means that the algorithm favours `Shape`s which have a squarer aspect ratio, as this has been considered preferable to creating rooms which are long and narrow.

If multiple `Rect`s exist with the same number of neighbours in the solution, then the second priority is to choose the `Rect` which is "closest to the outside" of the `Empty Floor`. This is, in fact, the `Rect` which has the fewest *actual* neighbours (neighbours which may or may not already be in the solution).

If the last `Rect` found does not meet the remaining required area exactly, it is divided, and a part of it is added to the solution. This is shown in Figure 10(G). However, this was not an intentional implementation choice, as will be discussed in section 6.3.3.

6.4.3 Recursion

In order to prepare the `Empty Floor::Shape` for the insertion of another room, the `Room 2D::Shape` which has just been found is removed from it using a CSG difference operation. This algorithm is able to deal with the partial `Rect` which was created in the `Room 2D`; this results in a further subdivision of the entire floor. Figure 11 shows the `Empty Floor` from Figure 10 when prepared for the creation of a second room.

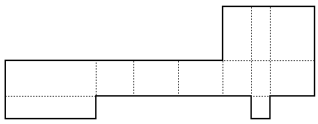


Figure 11 – After Removing the `Room 2D`

6.4.4 Converting Floor Plan to a Collection of 3D Walls

All `Room 2D::Shape`s are converted to `Room 2D::Polygon`s. The `Room 2D`s are then loaded into a `Floor 2D` class structure. A routine is then used to convert the `Floor 2D` to a `Floor 3D` which is composed of `Wall`s (not "`Room 3D`"s as one might first expect).

These `Wall`s are initially created in the form of singular `Block`s. Each `Wall::Block` is found by widening and extrapolating the `Edge`s of the `Room 2D::Polygon`s into 3D space. This can be seen by comparing Figure 8D and Figure 8E.

6.4.5 Positioning Doors and Windows

During this process, if the algorithm attempts to create the same `Wall` twice, this is an indication that it is shared by two rooms, and this is flagged within that `Wall` object. If the algorithm does not try to create the `Wall` twice, it is therefore an outside wall.

It is this flag which is used to decide whether a `Wall` can accommodate a window or a doorway. Using the 3D CSG algorithm, the software attempts to remove a doorway from the centre of every internal wall, and to remove between zero and three window sections from each wall, depending on the length of the wall and a random variable.

Placing doorways in every internal wall and randomly placing windows is not considered to be realistic, and the possibility of developing a better means of selecting their locations certainly exists. However, it is at least practical, in that it allows a route of access between any of the rooms.

6.5 Other Noteworthy Implementations

6.5.1 Lists and Loops

Past experience has shown that the commonly available list classes such as `std::vector` and `std::list` suffer two problems. Firstly they are unreasonably slow when creating lists of lists due to their tendency to try to arrange and rearrange their data sequentially in memory. Secondly, the creation and use of iterators to step through their contents can occupy several lines of code for what is a common and simple task.

This was foreseen in this project, and the very first implementation was two new doubly-linked list class templates: `TList` and `TLoop`. Within `TList`, there are three pointers to the first and last items, and one other item which is the iterator. Each item is responsible for its previous and next items via two pointers. `TLoop` is identical except that there is no last item; instead the end of the list points back to the start to create a seamless loop (this has been used for the `Polygon` class – a loop of `Vertex`s).

This implementation has been a positive effort. Keeping the iterator within the list class has made the code to cycle through lists uncomplicated. And when one considers that a `Floor 2D` has a `TList` of `Room 2D`s, and that each `Room 2D` contains a `TList` of `Polygon`s (which is a `TLoop` of `Vertex`s) and a `TList` of `Shape`s (which is a `TList` of `Rect`s), it essential to realise that the software will not attempt to try to rearrange this data in memory at any time.

6.5.2 Polygon-to-Shape and Shape-to-Polygon Conversions

These two algorithms each took just under a week to develop. Both work in a recursive manner, moving individual `Rect`s out of or into the solution data structure.

`Polygon`-to-`Shape` conversion operates by looking for rectangular enclosures at the top of the polygon, creating a `Rect` of the same size, and then removing these vertices from the `Polygon`. In this way, the algorithm, piece by piece, constructs a `Shape` while deconstructing the `Polygon` until the entire area is converted.

`Shape`-to-`Polygon` conversion operates by converting each `Rect` which neighbours the solution `Polygon` into a four point `Polygon`. It then adds this to the solution by removing the common edge between the solution and this additional `Rect` to generate a larger solution `Polygon`. This is repeated until all `Rect`s are removed from the original `Shape`.

6.5.3 Modification of Level Editor

Approximately a week was spent beginning the modification of the level editor. However, once the lost time on the CSG implementation was realised, this was abandoned as the need for a user-friendly interface to input the building plan outlines was a low priority. This subject is revisited in the following chapter.

6.6 Known Outstanding Bugs

6.6.1 Placing the Final Room

The algorithm seems unable to use the entire building floor space when placing a room. An attempt to creating a room which has exactly the same floor size as is remaining will result in the software crashing. As an alternative, the system has been recoded so that the final room is created by transforming the `Shape` containing the remaining region of floor space directly into a room, rather than try to place a room within it.

6.6.2 Edge Comparison

Doors and windows are often not inserted into walls where they should be. This is because the code currently only identifies two polygon edges as 'the same' if their end vertices are identical. Instead, it should identify where a part of the edges may share the same space and treat this as a shared wall. The author is adamant that this is the cause, and that changing this comparison would eradicate the problem.

6.6.3 'Flexibility in Stated Area' in 'Room Definition'

Room placement has been designed to use not a fixed area, but a flexible area. A range would be given in which the room area can lie, such that when the last `Rect` in a `Room 2D` solution is being considered:

- If none of the `Rect` area is within the room area's range, none of it will be used.
- Else if all of the `Rect` area is within the room area's range, all of it will be used.
- Else the `Rect` will be divided.

This system was operational earlier in development, but further modification seems to have caused a bug which now prevents this from working. The purpose of this system was to reduce the number of corners in a room. It also helps avoid the creation of another division of the `Empty Floor::Shape` after the creation of each `Room 2D`.

7 Results

This chapter analyses the project in hindsight. It is divided into two sections, the first covering the program itself, and the second covering the development process.

7.1 The Software

7.1.1 Answering the Research Topics

In section 4.1, the four principal concerns surrounding PICAG design and development were given. Here these are considered against the final software.

- **Development Time:**

The program is no way near the complexity that would be required for it to be used in commercial video games and so it gives virtually no insight into the development time of such a system. This was expected.

- **Independent Design Stages:**

The software does indeed demonstrate that, at least in this instance, dividing the process into a number of discrete stages is possible. The developed PICAG can be considered in three stages: the first creating a floor plan, the second converting this to a three dimensional building, and the third removing the negatives (windows and doors).

The outputs from one stage directly form the inputs for the next, making the design stages truly independent in the sense given here. This is demonstrated by the creation of the outputs shown in Figure 8D, E and F.

This is not expected to be true when developing more comprehensive PICAGs. When furnishing a bathroom, it would be useful to know whether that bathroom is in a family home or a five star hotel, meaning that some communication across the stages of development would be necessary. But in the case of this PICAG design, it is certainly evident that the relationships between design decisions and geometric elements do not form the nightmarish spider-webs envisioned by the author in chapter 3 of [12], and this is encouraging for future PICAG research.

- **Development of a Custom Language:**

It had been hoped that a definitional language would be developed during this project, and that its analysis would form the centrepiece of the project's evaluation. This language did not transpire to be anything more than a single value dictating the areas of

the rooms to be included in the building. But such a value is instantly gratified by the fact that all rooms must have areas, whether we are defining them, picking them randomly, or simply setting them all equal. The author does not consider this to be a language and it does not give any insight into the benefits or restrictions of a definitional PICAG language. Rooms are not created with any higher purpose than to define a region of the floor.

- **Spatial Awareness:**

It is possible to argue that little effort has been exhausted worrying about problems of spatial awareness. But it is also possible to argue that the entire software development has been about spatial awareness!

The PICAG algorithm has been designed with a very much common sense approach. The problem of fitting a room within the boundaries of a floor, and ensuring that it does not overlap other rooms is not just an aim of the floor placement algorithm, but its purpose. Section 7.2.1 below discusses the value of geometric algorithms in more detail.

Perhaps a better example of "spatial awareness" is the task of deciding whether walls should feature doors or windows by noticing the number of surrounding rooms. This is the only part of the algorithm which needs to have an awareness of existing physical objects in order to decide on the position or existence of *other types* of physical objects. However, the software is so simple that this is the only case of its kind, and so it has not added a great deal of complexity or confusion to the program design.

7.1.2 Program Output

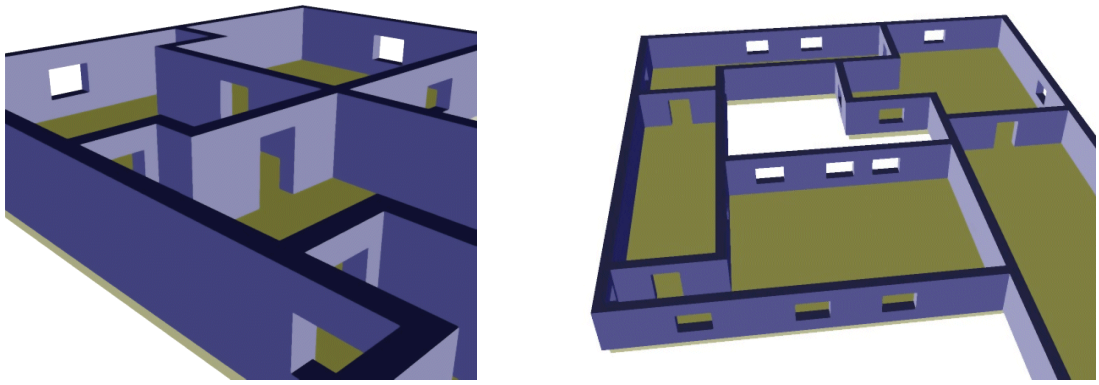


Figure 12 – Output Images of the Developed PICAG

Ignoring the edge comparison bug discussed in section 6.6.2, the output of the program is functionally correct. The existence of spaces which can be classified as "rooms" is

instantly recognizable to the human eye and mind. Doorways within walls ensure that the spaces are accessible, yet should still be considered separate spaces.

A consideration by anyone who is familiar with western architecture will suggest that the outputs have no glaring flaws. If the shapes shown in Figure 12 were the walls, windows and doorways of a real building, that building could not be considered completely useless. It is possible to imagine these forming a part of a school, office complex, or home.

However, the algorithm is not extensive enough to be used universally. It has no concept of corridors[6], lifts, stairs or halls. Corridors in particular are a common, fundamental and irreplaceable concept in western architecture. It can hence be said that although this PICAG is suitable for creating some types of buildings, it needs to be replaced, modified or expanded in order to handle a wider variety of room layout designs.

7.1.3 Dependency on Building Exteriors

The process of subdividing the `Empty Floor::Shape`, and sequentially selecting entire `Rect`s to use in a `Room 2D` has a number of unfortunate side effects.

Not least is the fact that if the building outline is rectangular, no `Split`ting occurs. This implies that when the floor plan is created, only one rectangle is available to choose from. The end result is that rooms are created by slicing this floor rectangle in a linear fashion. This is highly unrealistic, and it is not easy to think of a type of building which is simply divided laterally. This is even more unfortunate considering that many buildings are indeed simply rectangular.

7.1.4 Number of Corners

For reasons of cost, we are used to seeing rooms with a minimal number of corners. Figure 8 and Figure 12 show that the rooms generated by the software often appear to have an unusually large number of corners. The left-most floor in Figure 8 shows a result in which no room has four corners.

[6] A routine for creating a corridor network which could be used in conjunction with the final room placement algorithm was designed, and a small portion was developed, though it was soon abandoned in favour of other development priorities.

In defence of this, the error described in section 6.6.3 is partly responsible. If this functioned correctly then room in the example in Figure 10 would have one less corner, as would the room which neighbours it.

Furthermore, the aim is to create environments for entertainment. This problem is not serious since it has the positive side-effect that a room with more corners is often a more interesting space to visualise.

7.2 The Development Process

7.2.1 Geometric Algorithms

If this author can offer only one piece of advice to others considering developing a PICAG of any scale, it would be to start with a comprehensive and robust library of geometric structures and algorithms. In the initial plan (chapter 5), the modification of the CSG algorithm was assigned two weeks, and the creation of any necessary 2D geometric algorithms were to be developed within the two weeks assigned to the development of the room creation stage (see section 5.3).

In reality developing algorithms to analyse and modify 2D and 3D geometry consumed 75% of the development time. The procedural design algorithms (i.e. the innovative parts of the project) consumed the remaining quarter.

A small amount of time was spent at the beginning of the project trying to locate a suitable library of geometry. In hindsight, using half of entire project time to find (and possibly modify) such a library may have been a more constructive approach!

7.2.2 Initial Scheduling

Human error was made in the act of scheduling. The project schedule (section 7.2.2) should have been by organised by priority. The opinion was held that by doing the modifications to existing code first, the code base would become larger very early in the project. Although CSG was necessary and was used effectively, work on the Editor was a waste, as a user-friendly interface for building polygon design was a very relatively low priority and this work was later abandoned completely.

Furthermore, if the software had not reached the stage where a floor plan was designed correctly (as seemed the likely case at one point during the development), then 3D CSG would also have been a wasted effort.

In summary, the initial schedule should have been prioritised more strictly. When creating an experimental prototype, it is highly unwise to make the assumption that the software will reach a certain state in a given time!

8 Future Work

Having identified the limitations of the developed PICAG in the previous chapter, here section 8.1 suggests methods of improving the software, and section 8.2 suggests directions for future research work in this field.

8.1 Suggested Expansions of the Developed Software

8.1.1 High Dependency of Floor Shape

The core of the developed software is its room placement algorithm, and the largest flaw within this its high dependency on the building shape in determining the pattern of its rooms.

The simplest way to fix this seems to impose a maximum edge size for `Rect`s. Any `Rect` which exceeds this size should be divided into component `Rect`s. This will allow the algorithm a more varied network of `Rect`s in which to build the `Room 2D` solution. It will be effective towards solving the problems regarding the design of rectangular buildings. Other methods of fixing this problem exist and may provide better results.

8.1.2 Corridors

The second largest flaw in the output is the absence of corridors – an extremely common feature in western architecture. A method to find suitable routes for corridors within a floor plan was designed during the project, but has not been fully implemented and to describe its design is beyond the scope of this document.

It would be interesting to observe how effective the room placement algorithm becomes when the floor is initially divided into much smaller and remote pieces by the network of corridors. The chances are it would need some modification, as *no* code for handling the division of the floor plan has yet been written; although the floor plan can already be divided by the creation of rooms, testing has so far shown the algorithm appears to "just work anyway". This is unlikely to remain the case when the building's floor area is divided more aggressively by corridors.

Adaptation of the room placement algorithm to pick the best piece of the floor plan in which to create a particular room seems the most obvious solution.

8.1.3 Multiple Floors

Algorithms to create stairs and lifts would quickly allow for the possibility of multi-floored buildings. Implementing this is not difficult – a lift shaft can be considered as a special type of room which exists at the same location on every floor, and in which the ground is removed (possibly using CSG).

8.2 Future Research Directions

8.2.1 Alternative Room Placement Strategies

The room placement algorithm developed here is only one solution – one which subdivides the floor into a lattice and uses a search algorithm to locate suitable extensions until the required room area is achieved.

With some consideration it is not hard to come up with quite different methods of positioning rooms. Different algorithms may be better suited towards different shapes, purposes and sizes of rooms. Research which compares and contrasts room placement algorithms would be welcome.

8.2.2 Development of a Definitional Scripting Language

A language allowing the descriptions of types of buildings, rooms and furniture needs to be developed. This may contain information such as what types of building may contain a particular room, what floor that room should be on, what shape and size it is, what kind of furniture exists in it, and how that furniture might be arranged. This will bring PICAG design towards a stage where it is customisable, allowing it to develop new types of buildings by scripting new rules rather than altering the program.

8.2.3 Geometry Library

The development of an open-source, comprehensive geometry library with routines for analysing, converting and modifying 3D and 2D geometry would be useful as a sound basis for developing future PICAG projects (along with may other 3D graphics projects).

8.2.4 Prototyping

In this case, prototyping has brought about useful observations and lessons. The software however is extremely basic, and (as discussed in section 7.1.1) the answers to the most pertinent questions surrounding PICAG design have at best been hinted at, and at worst not been answered in any form at all. A prototyping approach is therefore still recommended as the best route forward.

8.2.5 Research

However, anyone considering creating a much larger PICAG development is also advised to study the field of semi-procedural architecture design more closely. Not all of the techniques which have been used there are likely to be applicable to the creation of game environments, but it seems likely that some algorithms exist which can either be used directly, if not, learnt from.

9 Conclusion

This chapter closes the document by summarising the state of this research topic with the hindsight of the project. Section 9.1 covers the form of the developed software, section 9.2 reiterates the principle lessons learned, and section 9.3 encourages future research work in this area.

9.1 State of the Final Software

This paper has presented a system for procedurally generating indoor urban architecture. Its results are functional and technically correct[7], but their use is limited. Of the two inputs to the system, the design process is, unfortunately, influenced more by the exterior shape of the building than by the descriptions of the rooms to be included. This is due to the failure to create a scripting language within the timeframe, along with the nature of the room placement algorithm.

The buildings created by the algorithm are by no means useless, and although visually unadorned, do represent practical building designs.

9.2 Chief Findings

The development of this software seems to suggest that city building interiors, as suspected, may be a relatively favourable form of architecture for procedural development. The algorithm has been divided effectively into independent stages, and many of these make use of simple, recursive algorithms with great success.

However, the need for a comprehensive library of routines for modifying and analysing 2D and 3D geometry is paramount. In this project, it has absorbed the vast majority of development time, and the algorithms created do no more than is necessary for the purpose of the procedural design process. The author recommends that any future work into PICAG creation begins by acquiring or developing such a library.

[7] If ignoring bugs which are well understood.

9.3 Future Directions

Prototyping still appears to be the best means by which to continue research in this field, mainly since the simplicity of this project has left many larger questions still unanswered. The author suggests that this is supported by a closer study of existing research papers on semi-procedural architecture design tools.

Although the software did not meet the expectations set by its initial design, valuable lessons have been learned. The outputs are basic and only useful in and for particular circumstances, but they are most valuable in demonstrating that procedurally generating the indoor architecture for computer game city environments is as plausible as it is desirable.

10 Glossary

- **CSG [Constructive Solid Geometry]** is a collection of algorithms which can be used to find the logical union, difference or intersection of two three-dimensional spaces.

- **CSG [Constructive Shape Geometry]** is as above, but for two-dimensional regions.

- **Level** refers to an environment within a computer game.

- **Media** refers to the portion of computer game software which is not executable code. This can include databases, artwork and sound.

- **PICAG [Procedural Indoor City Architecture Generator]** is a term defined by this document, as an automated or semi-automated computer program which designs urban indoor architecture. This project has attempted to design and implement a simple PICAG system, with computer game creation as the anticipated beneficiary.

- **Semi-Procedural** generation refers to a design process which is carried out partly by a designer, and partly by an algorithm.

11 References

The chapter lists the literature in section 11.1 and the computer game software in section 11.2 which have been referenced throughout this document.

11.1 Literature

[1] LECKY-THOMPSON, G. W. 2001. *Infinite Game Universe: Mathematical Techniques*. Hingham, MA: Charles River Media Inc. ISBN: 1-58450-058-1

[2] ADAMS, E. 2002. *The Role of Architecture in Videogames*. Gamasutra: http://www.gamasutra.com/features/20021009/adams_01.htm

[3] GREUTER, S. ET AL. 2003. *Real-Time Generation of Pseudo Infinite Cities*. Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia: pp. 87-94.

[4] LECKY-THOMPSON, G. W. 2002. *Infinite Game Universe: Level Design, Terrain and Sound*. Hingham, MA: Charles River Media Inc. ISBN: 1-58450-213-4

[5] PARISH, Y. I. H. & MÜLLER, P. 2001. *Procedural Modeling of Cities*. Proceedings of the 2001 Conference on Virtual Reality, Archeology, and Cultural Heritage (ACM): pp. 187-196.

[6] SO, C., BACIU, G. & HANQIU, S. 1998. *Reconstruction of 3D Virtual Buildings from 2D Architectural Floor Plans*. Proceedings of the ACM Symposium on Virtual Reality Software and Technology: pp. 17-23.

[7] WONKA, P. ET AL. 2003. *Instant Architecture*. ACM Transactions on Graphics, 22(3): pp. 669-677.

[8] BIRCH, P. J ET AL. 2001. *Rapid Procedural Modelling of Architectural Structures*. Proceedings of the 2001 Conference on Virtual Reality, Archeology, and Cultural Heritage (ACM): pp. 187-196.

[9] PRUSINKIEWICZ, P. & LINDENMAYER, A. 1990. *The Algorithmic Beauty of Plants*. New York: Springer-Verlag. ISBN: 0-387-97297-8

[10] SHMUEL, S., KAREN, I., & CEDERBAUM, I. 1988. *Optimal Aspect Ratios of Building Blocks in VSLI*. Proceedings of the 25th ACM/IEEE Design Automation Conference.

[11] SIMPSON, R., 1980. *A Survey of Space Allocation Algorithms in Use in Architectural Design in the Past 20 Years*. PhD Thesis. Southern Connecticut State College and Yale School of Architecture, New Haven, Connecticut.

[12] BRADLEY, B. 2004. *Procedural Generation of Architectural Game Environments*. BSc Thesis. University of Abertay Dundee, Bell Street, Dundee.

[13] RHOADES, J. ET AL. 1992. *Real-Time Procedural Textures*. Department of Computer Science, University of North Carolina at Chapel Hill. http://www.cs.umd.edu/projects/gvil/papers/i3d92.pdf

11.2 Computer Games

[G1] ROCKSTAR NORTH, 2002. *Grand Theft Auto III*. Rockstar, Take Two.

[G2] ROCKSTAR NORTH, 2003. *Grand Theft Auto: Vice City*. Rockstar, Take Two.

[G3] ROCKSTAR NORTH, 2004. *Grand Theft Auto: San Andreas*. Rockstar, Take Two.

[G4] REFELECTIONS, 2004. *Driv3r*, Atari.

[G5] TREYARCH, 2004. *Spiderman 2 – The Movie*. Activision.

[G6] REFLECTIONS, 2000. *Driver*. Atari.