FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Modelação Procedimental para Desenvolvimento de Jogos de Computador

Pedro Amorim Brandão da Silva

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Prof. António Fernando Coelho

28 de Junho de 2010

Modelação Procedimental Para Desenvolvimento de Jogos de Computador

Pedro Amorim Brandão da Silva

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Rui Rodrigues (Professor Auxiliar Convidado)

Vogal Externo: Maximino Bessa (Professor Auxiliar)

Orientador: António Fernando Coelho (Professor Auxiliar)

_____

31 de Julho de 2010

Resumo

A modelação de ambientes virtuais constitui um dos processos mais exigentes no desenvolvimento de jogos de computador, na medida em que requer a criação de uma elevada quantidade de conteúdos com grande detalhe e qualidade, resultando em esforços e custos elevados, bem como em longos períodos de desenvolvimento. Recentemente, uma das tendências que se tem sentido cada vez mais em jogos é a tentativa de recriação de ambientes urbanos existentes que, provocando um sentimento de familiaridade do jogador com o local, contribuem para uma maior imersão no jogo. No entanto, tal também apresenta dificuldades adicionais na sua concepção, dado é necessário consultar informação real para a sua criação.

A utilização de métodos procedimentais para geração de conteúdos tridimensionais em jogos de computador tem vindo a tornar-se numa prática recorrente, com resultados bastante interessantes. Tal prática deve-se ao facto de estas técnicas requerem um esforço bastante menor na medida em que são capazes de gerar, de forma automática (ou pelo menos, com reduzida interacção humana), modelos tridimensionais. Para tal, é necessário que sejam introduzidas algumas orientações e indicações pelo utilizador, ou que se utilizem fontes informativas que descrevam os espaços de forma detalhada. No entanto, podem ser necessárias formas rápidas de lhes aceder e máquinas poderosas para as operar se estas tomarem dimensões e quantidades elevados. Por outro lado, tais características nem sempre implica que tais fontes sejam suficientes detalhadas para se obter modelos precisos, pelo que é necessário recorrer a métodos estocásticos ou empíricos sobre o ambiente urbano para ampliar a informação existente.

Nesta dissertação apresenta-se o modelador PG3D, uma solução de modelação procedimental de ambientes urbanos virtuais para aplicação a jogos de computador, cuja principal característica consiste na sua implementação em sistemas de gestão de bases de dados com capacidades espaciais e análise geográfica. Este tipo de plataforma serve assim, por um lado, como local de armazenamento das fontes de informação necessárias, e por outro como local de armazenamento dos modelos criados, sendo operados por um conjunto de procedimentos internos às bases de dados, reduzindo assim os tempos de acesso aos dados e a carga de dados a operar em cada momento.

Derivada das várias potencialidades que o conceito encerra, surge a Gramática PG3D que possibilita a construção de ambientes urbanos de grande dimensão e detalhe, baseada em vários procedimentos de modelação e num crescimento iterativo, condicional e característico, na medida em que se apoia em informação real e na relação geográfica que existe entre os seus elementos.

Com o intuito de complementar e facilitar os meios de interacção e exportação de informação que este tipo de sistemas de armazenamento de dados consegue proporcionar, apresenta-se também uma ferramenta dedicada à gestão dos parâmetros necessários aos processos de modelação, bem como à exportação da informação dos ambientes gerados para formatos interoperacionais, permitindo a sua utilização em diversas aplicações, entre as quais os jogos de computador. Esta integração das duas partes constitui assim o Sistema PG3D.

Abstract

The creation of virtual environments is one of the most demanding processes regarding the development of computer games, in the way that it requires the design of large amounts of high detail and quality contents, therefore leading to great costs and a great deal of effort, as well as long development periods. Recently, one of the trends that have been growing in games is the attempt to recreate existing urban environments which, by causing the player's recognition of the scene, may lead to a greater immersion in the game. However, this presents also greater difficulties in building these environments, since real information must be used.

The use of procedural methods for generating three-dimensional content in computer games has become a recurring practice, with interesting results, requiring much less effort, by generating, automatically (or at least, with little human interaction) three-dimensional models. Some guidelines and directions should be introduced by the user, or information sources that are used to describe the spaces in detail. However, sometimes the size and number of such sources is too large, which requires quick ways to access them and powerful computers to operate on them. On the other hand, this does not necessarily mean that such sources are detailed enough to obtain accurate models, which makes it crucial to use stochastic methods or empirical data on the urban environment to amplify the existing information.

This dissertation aims to present the PG3D modeler, a solution for procedural modeling of virtual urban environments for computer games, whose main characteristic is its implementation in spatial databases management systems. This type of platform serves firstly as a container for information sources, and secondly as a storage location for the created models, whereas these are operated by a set of stored procedures in the databases, thereby reducing the time access to data.

Due to the large potential hold by this concept, the PG3D Grammar arose, enabling the construction of large detailed urban environments, based on various modeling procedures and an iterative, conditional and characteristic growth, by relying on real information and the geographical relationship that exists between its elements.

In order to complement and facilitate the means of information transaction and export that the data storage systems possess, a management tool has been conceived and will be presented. This allows the configuration of the necessary parameters of the modeling processes, as well as the export of generated data into a set of interoperable formats, allowing its use in various applications, including computer games. These two parts together form the PG3D System.

Agradecimentos

Gostaria de começar por agradecer ao meu orientador, o Prof. António Fernando Vasconcelos Cunha Castro Coelho, por, antes de mais, me ter introduzido à modelação procedimental, uma matéria que já há muito que me fascinava na área da computação gráfica, pela seu apoio, disponibilidade e pelos vário conselhos, materiais e ensinamentos que me foi transmitindo ao longo dos vários encontros que tivemos, cuja frequência só demonstra o seu contínuo interesse pelos alunos, qualidade esta que já tinha testemunhado em experiências anteriores.

Agradeço também à direcção da Faculdade de Engenharia da Universidade do Porto e do Departamento de Engenharia Informática e Computação pela disponibilização de um espaço de trabalho próprio, e que se tornou num espaço de desenvolvimento especialmente frutuoso.

Gostaria ainda dirigir um especial obrigado aos meus colegas e amigos da Faculdade de Engenharia, pelo companheirismo e ajuda sempre demonstrados e pelos momentos de pausa proporcionados que, devido à minha natureza por vezes exageradamente dedicada ao trabalho, normalmente não faria, mas que reconheço terem sido momentos importantes de partilha e concepção de ideias.

Finalmente, aos meus pais e irmã, bem como todos os familiares e amigos, quero agradecer pelo seu interesse e apoio constante, especialmente nos momentos mais difíceis, bem como pela sua compreensão e paciência face aos vários momentos de convívio ultrapassados.

Pedro Amorim Brandão da Silva

Conteúdo

Índice de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

Blob – *Binary Large Object*
COLLADA – *COLLAborative Design Activity*
CGA – *Computer Graphics Architecture*
DEM – *Digital Elevation Model*
FPS – *First Person Shooter*
GB - *Gigabyte*
GIS – *Geographics Information System*
GiST – *Generalized Search Tree*
GTA – *Grand Theft Auto*
KB – *Kilobye*
L-System – *Lindenmayer System*
ODBC – *Open Data Base Connectivity*
OGC – *Open Geospatial Consortium*
PCG – *Procedural Content Generation*
PG3D – *PostgreSQL, PostGIS, Procedural Generation 3D*
PL/SQL – *Procedural Language/Structured Query Language*
RAM – *Random Access Memory*
RGBA – *Red, Green, Blue, Alpha*
RPG – *Role Playing Game*
SQL – *Structured Query Language*
TB – *Terabyte*
X3D – *Extensible 3D*
XNA – *XNA's Not Acronymed*

Capítulo 1

Introdução

A indústria dos jogos de computador apresenta um dos maiores crescimentos a nível mundial. Anualmente, são lançadas centenas de novos títulos, cada um tentando apresentar algo de novo ou mais desenvolvido do que os anteriores. Efectivamente, os jogos de computador têm apresentado uma evolução impressionante, providenciando uma jogabilidade cada vez mais avançada, grafismos extremamente detalhados e complexos, bem como enredos profundos. Estes encontram-se encapsulados na definição de conteúdos variados, desde os elementos sonoros, às texturas e modelos aprimorados, cuja criação individualizada requer especial atenção por parte dos artistas ou designers que os concebem.

Em jogos mais recentes tem-se assistido a uma forte aposta na concepção de conteúdos em larga escala, com o objectivo de conferir uma maior longevidade, diversidade e imersão aos jogos criados. No foco desta abordagem tem estado o desenvolvimento dos ambientes virtuais tridimensionais em que os jogos se desenrolam.

Tem-se procurado desenvolver espaços cada vez maiores e mais amplos, recheados com grandes diversidades de conteúdos, de forma a instigar a exploração do terreno por parte dos utilizadores e promovendo uma experiência menos linear e mais duradoura. Adicionalmente, a modelação com um elevado nível de detalhe e realismo tem sido outro dos objectivos da produção de ambientes virtuais. Títulos como The Elder Scrolls: Oblivion e Fallout constituem apenas alguns exemplos em que a liberdade disponibilizada, as possibilidades de interacção com o ambiente virtual, e o grande detalhe gráfico constituíram alguns dos elementos mais aclamados desses jogos. Recentemente, outros títulos como Prototype e GTA IV apresentaram ambientes virtuais baseados em zonas urbanas reais. Estas abordagens tiveram uma recepção muito positiva, na medida em que os jogadores se identificaram com espaço representado e se sentiram mais imersos no mesmo.

Apesar do impacto positivo, a criação de ambientes virtuais deste calibre não surge sem um custo elevado. Produções desta dimensão obrigam ao recurso de grandes equipas que produzam cada um dos conteúdos, o que corresponde a um esforço elevado e consequentemente, a elevados períodos e custos de desenvolvimento.

Com o intuito de acelerar estes processos, têm sido desenvolvidas técnicas e ferramentas que consigam automatizá-los, carecendo apenas da definição de um conjunto pequeno de instruções ou regras. É neste sentido que surge o conceito de modelação procedimental. Este engloba diversas técnicas para gerar, de forma automática (ou pelo menos com menor intervenção humana), modelos tridimensionais. Tal poderá resultar numa extrema redução nos recursos necessários à concepção de ambientes virtuais.

1.1. Motivação

Nos primórdios da história dos jogos de computador, o desenvolvimento destes realizava-se de forma relativamente simples. Os jogos eram criados por um conjunto pequeno de pessoas, cada uma dedicando-se a qualquer tarefa de programação ou design. Certos jogos chegavam a ser programados de raiz por uma única pessoa e atingiram níveis de sucesso surpreendentes.

Com a evolução das capacidades computacionais que o hardware e software têm apresentado, a fasquia relativamente aos jogos de computador tem subido, resultando na dificuldade cada vez maior de exceder as expectativas dos jogadores, que aspiram a mais, melhores e maiores conteúdos gráficos, sobretudo no âmbito tridimensional. De forma a conseguir responder a tais exigências, não é raro encontrarem-se equipas de 30, 50 ou 100 programadores e designers a trabalharem durante anos na criação de um jogo. Em equipas deste tamanho, é necessária a divisão por especialidades, sendo a concepção dos ambientes virtuais, em que o jogo decorre, uma delas. Os designers desta área têm, por isso, uma responsabilidade consideravelmente maior nos dias de hoje.

Contudo, esta carência em termos de tempo e recursos humanos tem tornado o desenvolvimento de jogos de computador num investimento pesado (e também arriscado) em termos financeiros. Adicionalmente, a tarefa de criação de ambientes muito extensos torna-se, em muitos casos, um trabalho desgastante e propenso a falhas, não só em termos de criação, mas também na realização dos testes, que exigem uma análise exaustiva.

As ferramentas de modelação procedimental apresentam-se como uma solução de criação de conteúdos tridimensionais, na medida em que são capazes de os gerar de forma quase automática, sendo necessária apenas a definição de um conjunto de regras por parte de um utilizador sendo este, preferencialmente, o *designer* que imaginou o resultado pretendido.

A utilização destes métodos de criação desde há muito que têm sido aplicados em jogos de computador ao longo do tempo, com diferentes alvos e intensidades, incluindo em alguns aspectos de ambientes virtuais. Em alguns casos, têm sido utilizados para a simples disposição de inimigos num terreno, noutros para a geração de diversos tipos de árvores. Tal permitiu não só a rapidez de geração, mas também uma maior variedade de hipóteses, dada a forte componente aleatória que algoritmos de modelação procedimental podem conter.

Apesar da sua larga disseminação e aplicabilidade, a modelação procedimental ainda apresenta algumas dificuldades, especialmente na modelação de elementos realistas. Em jogos mais recentes, tem-se vindo a assistir à tentativa de representação de ambientes reais, sobretudo de meios urbanos. Apesar de constituírem um desafio maior, já têm surgido algumas abordagens de modelação procedimental, capazes de produzir ambientes bastante próximos da realidade. Contudo, estas abordagens encontram-se ainda numa fase inicial, sendo por isso necessárias soluções inovadoras.

O grande impacto dos ambientes urbanos virtuais realistas em jogos de computador, associado ao seu elevado custo e tempo de desenvolvimento, motivam o desenvolvimento de aplicações de modelação procedimental que ajudem na sua criação. Estas provaram ser extremamente úteis em aplicações diversas de jogos de computador, havendo por isso fortes perspectivas de aplicabilidade na concepção de ambientes deste calibre.

1.2. Descrição do Problema

A utilização de métodos procedimentais para geração de conteúdos tridimensionais em jogos de computador tem vindo a tornar-se numa prática recorrente, propondo facilidades de criação de ambientes, na medida em que gera uma boa parte destes de forma automática (ou pelo menos, com reduzida interacção humana). Não obstante, é necessário que sejam declaradas regras e parâmetros que orientem essa mesma geração.

Embora as abordagens procedimentais já tenham produzido resultados interessantes na criação de ambientes urbanos virtuais fictícios, para a reprodução de ambientes reais ainda não existem soluções com fidelidade visual suficientemente elevada. Uma das suas maiores dificuldades consiste na utilização de fontes informativas que descrevem os espaços reais. Tal pode ser encontrado em fotografias, mapas variados ou em sistemas de informação geográfica. Contudo, a dimensão e número de fontes a utilizar pode assumir valores elevados, tornando assim insuportável o tempo de acesso às mesmas ou mesmo impossível a operação sobre elas, dadas as limitações de *hardware* que as máquinas possam possuir. Por essa razão, surge a necessidade de implementar metodologias de acesso e operação rápidas e eficientes. Tal consiste um problema que é importante atender.

Por outro lado, a quantidade e dimensão das fontes não implica que estas possuam detalhe suficiente para descrever os ambientes na sua plenitude, pelo que é necessário, em determinados casos, que se recorra a métodos estocásticos ou a conhecimento sobre o ambiente a modelar para ampliar a informação existente. Um dos métodos consiste na percepção do ambiente envolvente de cada elemento, que conjuntamente com regras e noções de arquitectura, é capaz de produzir resultados mais plausíveis. Abordagens neste sentido já têm sido exploradas, mas o seu emprego extensivo e eficiente ainda constitui um desafio a que se pretende responder nesta tese.

1.3. Objectivos

O objectivo principal deste projecto é o desenvolvimento de uma aplicação para modelação procedimental de ambientes urbanos para jogos de computador. Para tal será necessário concretizar os seguintes objectivos específicos:

- Investigar o estado da arte ao nível da modelação procedimental, em especial de ambientes urbanos realistas, e as suas aplicações em jogos de computador;
- Desenvolver uma aplicação que faça modelação procedimental dos diversos elementos que as compõem, tais como estradas, edifícios e mobiliário urbano;
- Integrar os diversos elementos do ambiente urbano com informação geoespacial de ambientes reais e com modelos digitais de elevação do terreno.
- Integrar os elementos gerados num contexto de utilização em jogos de computador

1.4. Estrutura da Dissertação

Esta dissertação encontra-se organizada em 6 capítulos, adicionados de referências e anexos técnicos. O primeiro e presente capítulo consiste na descrição geral do problema, na motivação que levou à realização deste trabalho e na definição dos objectivos propostos para a sua realização.

No segundo capítulo é realizado um levantamento do estado da arte ao nível da geração procedimental de conteúdos, sendo primeiro explanado o seu conceito, benefícios e contrapartidas, seguido de alguns exemplos de aplicação em jogos. Segue-se uma abordagem à modelação de ambientes virtuais, sendo referidos alguns desafios e processos gerais na sua concepção. Passando ao desenvolvimento de ambientes urbanos virtuais, são apresentados vários trabalhos realizados a vários níveis de construção: terreno e redes rodoviárias, edifícios, fachadas, bem como outros conteúdos urbanos. O capítulo termina com a explicação de algumas questões que ainda permanecem em aberto neste ramo de investigação.

O terceiro capítulo é dedicado à descrição da solução proposta, começando pelas principais dificuldades com que os vários autores se têm defrontado e procurado responder. Segue-se a explicação conceptual do Sistema PG3D e a sua implementação sobre bases de dados espaciais. Com vista a complementar o seu funcionamento lógico, são descritas as gramáticas PG3D, começando pela definição dos seus conceitos fundamentais, seguindo-se o seu funcionamento e potencialidades na definição de instruções de modelação. Termina-se com a algumas explicações referentes às formas de visualização e exportação dos dados gerados.

O quarto capítulo consiste na descrição de alguns detalhes da implementação da solução apresentada no capítulo anterior. É descrita a arquitectura do Sistema PG3D desenvolvido, apresentados alguns detalhes de implementação e sobre as tecnologias empregues, bem como mencionadas algumas das funcionalidades disponíveis.

O quinto capítulo consiste na apresentação dos resultados produzidos, sendo realizada uma avaliação dos mesmos e comparados a vários níveis, tais como os tempos de geração, a qualidade dos modelos produzidos e da aplicabilidade do Sistema PG3D aos jogos de computador.

O sexto e último capítulo visa apresentar um pequeno resumo das constatações e conclusões a realizar sobre esta dissertação. É feita uma avaliação do nível de satisfação atingido e descritas algumas perspectivas de desenvolvimento futuro do sistema PG3D.

Capítulo 2

Modelação Procedimental em Jogos de Computador

O termo *procedimental* refere-se ao processo de computação de uma função em particular. Procedimentos, por vezes também denominados por rotinas, subrotinas, ou métodos, contêm um conjunto de instruções computacionais a serem executadas. Um dado procedimento pode ser chamado várias vezes e em qualquer momento durante a execução de um programa, por outros procedimentos ou recursivamente por si próprio.

A aproximação procedimental na geração de conteúdos para jogos de computador pretende seguir os mesmos princípios, na medida em que encapsula, de forma modular, um conjunto de passos a serem realizados, por vezes repetidamente, sobre um conjunto de dados de entrada. Cada um destes passos pode equivaler a uma operação simples, resultando o conjunto destes numa realização bastante mais complexa. Isto reverte numa redução de esforço necessário a ser realizado por interacção humana, uma vez que, sendo especificadas as regras de para obtenção do resultado, este pode ser criado de forma rápida por um computador.

Esta abordagem tem sido fundamental em múltiplas áreas, tais como o desenvolvimento de jogos de computador. Com a crescente complexidade, variedade e longevidade que os jogos pretendem oferecer, tem sido cada vez mais importante criar elevadas quantidades de conteúdos: personagens, objectos e sobretudo ambientes, que aspiram, para além de grande dimensão, conseguir simular ambientes realistas, tais como naturais e urbanísticos. Assim sendo, estes últimos constituem efectivamente a componente que exige mais esforço, tempo e recursos no desenvolvimento de jogos, tendo-se por isso procurado soluções de criação rápidas e o mais automáticas possível. Nesse sentido surge a modelação procedimental.

Numa primeira secção, apresentar-se-á brevemente o conceito, enquadrado na definição da geração procedimental de conteúdos em jogos, sendo referidos aspectos, estratégias e exemplos de utilização já existentes. Numa segunda secção, será abordada a modelação procedimental de ambientes em jogos de computador, com foco na tendência mais recente em jogos, nomeadamente a de concepção de ambientes reais.

2.1. Geração Procedimental de Conteúdos

A geração procedimental de conteúdos é um termo cada vez mais popular e recorrente em jogos de computador. Uma vez que, como se irá referir de seguida, a modelação procedimental é uma particularização deste conceito, pretende-se assim analisar a geração

procedimental em geral, em termos de benefícios, contrapartidas e estratégias, cujas abordagens são mais amplas e genéricas e por isso inspiradoras para a modelação procedimental.

2.1.1. Conceito

O conceito de geração procedimental nos jogos de computador tem vindo a adquirir várias definições que, embora não contraditórias, indicam aproximações a níveis ligeiramente diferentes [HLWL08, INSW07, PCGW09a, PCGW09b, YOUN09a]. De uma forma geral, é possível afirmar que a geração procedimental de conteúdos procura criar elementos (modelos, texturas, sons, música, objectos, etc.) de forma automática (ou, pelo menos, com muito reduzida interacção humana) através de um conjunto de técnicas, algoritmos computacionais e um determinado nível de aleatoriedade.

Algumas definições fazem corresponder geração procedimental a "totalmente aleatório" e "não determinístico" [YOUN09a]. Embora seja verdade que a geração procedimental pode produzir uma infinidade de resultados diferentes com a definição de dados aleatória (dentro das limitações computacionais de gerar essa aleatoriedade), não é absolutamente necessário que assim seja. É bastante trivial fazer o computador gerar a mesma sequência de números aleatórios, produzindo assim um conteúdo "aleatório" que surja sempre igual cada vez que seja visualizado. Assim sendo, o geração de conteúdo pode ser determinística, sendo o resultado automaticamente e aleatoriamente produzido pelo computador sempre igual, se desejado [YOUN09b].

Outras definições fazem distinção entre "geração procedimental" e "geração procedimental de conteúdos"[DOUL08, PCGW09a, PCGW09b], sendo a principal diferença o facto de a geração procedimental de conteúdos produzir resultados efectivamente diferentes em cada execução do jogo, afectando assim a experiência sentida em cada geração. Embora seja uma diferença pertinente, o factor de alteração de jogabilidade não constitui o foco desta análise, interessando apenas o factor de automatismo e redução de interacção humana que a geração procedimental de conteúdos oferece.

Como referido anteriormente, os conteúdos possíveis de serem gerados procedimentalmente incluem som, música, texturas, modelos tridimensionais etc., sendo sobretudo a geração destes últimos o foco principal deste documento, processo este que especificamente se denomina por *modelação procedimental.*

2.1.2.Benefícios

A geração procedimental possui então, entre outros, os seguintes benefícios [HLWL08]:

- **Redução de trabalho, custos e aumento de produtividade:** Para produzir um determinado conteúdo, um sistema procedimental requer menos interacção/entradas, resultando num trabalho menor por parte do *designer* responsável, aumentando assim, a sua produtividade.
- **Compressão de dados:** No contexto actual, tal fará mais sentido na necessidade de envio da informação através de uma rede de dados, apesar da redução da compressão de dados ser bem-vinda em qualquer meio, se fizer sentido. A possibilidade de reprodução exacta da informação do lado receptor a partir de um

pequeno conjunto de dados diminui a necessidade de se guardar informação redundante.

- **Geração de conteúdos por parte de utilizadores:** Nem sempre os jogadores possuem o tempo e perseverança necessária para produzir conteúdos de qualidade para jogos. Contudo, se através de um sistema procedimental for necessário apenas algumas entradas, tal torna-se simples e acessível para qualquer um. Os exemplos mais clássicos são os de configuração de personagens em *Role Playing Games.*
- **Geração de conteúdos por parte de programadores:** Mesmo as equipas com falta de *designers* poderão dar-se ao luxo de produzir conteúdos de grande qualidade e dimensão criando métodos de geração procedimental que produzam uma quantidade infinita de hipóteses através de um conjunto pequeno de entradas aleatórias.

2.1.3. Desvantagens

Apesar das vantagens enumeradas, nem sempre é simples nem apropriada ou, por vezes, nem mesmo possível a aplicação de métodos procedimentais para a geração de conteúdos. Por outro lado, é igualmente possível que a utilização de tais processos resulte em outros tipos de dificuldades:

- **Complexidade de Implementação:** Apesar de poder resultar na redução de recursos a nível de pessoal dedicado à criação dos vários conteúdos, a implementação de métodos de geração procedimental requer normalmente mão-de-obra mais qualificada, que possua conhecimentos tanto na área artística como na de programação [DANC07, REMO08]. Muitas ferramentas genéricas de geração procedimental existentes poderão suprimir esse problema, mas nem sempre serão capazes de cumprir todos os requisitos de geração pretendidos na implementação de um jogo.
- **Repetitividade e falta de criatividade:** Sendo criados a partir de um conjunto de regras limitadas e de um certo grau de aleatoriedade, os conteúdos gerados procedimentalmente correm o risco de se tornarem repetitivos e pouco imaginativos ao longo do tempo. Se não for cuidadosamente programado, os mesmos conteúdos poderão ser gerados em locais muito próximos, tornando-os pouco interessantes [DANC07].
- **Tempos de espera elevados:** Quando utilizados na altura da execução do jogo, os métodos de geração procedimental, que exigem à partida um processamento mais intenso, podem resultar, em máquinas menos potentes, em períodos de espera mais longos, algo que o jogador dificilmente estará disposto a despender [INSW07].
- **Factores Sociais:** De uma forma geral é possível afirmar que os métodos de geração procedimental falham na produção de conteúdos de cariz humano. Diálogos, vozes, descrições e narrativas são alguns dos exemplos que ainda não é possível gerar automaticamente, mas que se trata de uma tarefa exigente em tempo e recursos [DANC07].

- **Teste aos Conteúdos:** A introdução de geração procedimental possibilita assim a fácil criação de conteúdos de larga escala (tais como ambientes) através de um conjunto reduzido de entradas e de regras de geração. No entanto, a introdução de uma nova regra poderá ter repercussões menos evidentes noutras. Por outro lado, a verificação desse facto, dada a grande dimensão do produto gerado, pode-se tornar também mais complicada [REMO08]. Neste sentido, são ainda necessários métodos de teste adequados.

2.1.4. Exemplos de Geração Procedimental em Jogos

O conceito de geração procedimental não é novo na área dos jogos electrónicos. Possivelmente um dos primeiros casos de utilização terá surgido na década de 80, denominado *Elite*, um simulador de naves espaciais, que oferecia 8 galáxias, cada uma contendo 256 planetas e respectivas estações espaciais, todas à espera de serem exploradas, sendo todo este conteúdo gerado procedimentalmente [JNSN09].

Muitos outros títulos foram surgindo ao longo do tempo. O muito aclamado *Diablo* e a sua sequela; jogos da saga *The Elder Scrolls*, como *Morrowind* e *Oblivion; Hellgate:London* são apenas alguns dos exemplos de jogos que proporcionaram ambientes muito extensos, criados procedimentalmente [DOUL08]. Já outros jogos utilizaram tais métodos para a fácil criação de personagens, nomeadamente o jogo *Spore [TSPR09]*. Recentemente, o jogo *Borderlands* demonstrou ser capaz de oferecer mais de meio milhão de armas diferentes, criados de forma procedimental [BLGE08].

Figura 1 - Spore - Um jogo que ficou conhecido pela variedade de personagens geradas procedimentalmente [BENZ09]

Outros títulos foram ao longo do tempo recorrendo a métodos procedimentais para auxiliar no desenvolvimento, mas tal como nos exemplos antes mencionados, a maioria concentrou a sua aplicação apenas uma área.

Recentemente, têm surgido tentativas mais profundas de implementação de meios procedimentais em quase todos aspectos de desenvolvimento. Um exemplo será o jogo *.kkrieger* [TPKT04], um *First Person Shooter*, no qual, todos os conteúdos, nomeadamente texturas, modelos, animações e som são criados no início da execução do programa, permitindo assim a

compactação de todo o jogo num executável com menos de 100 *KB* [BENZ09]. Este constitui assim uma das apostas de geração procedimental mais profundas até ao momento e demonstra até que ponto é que se poderá chegar com métodos procedimentais.

Figura 2 – .kkrieger, um jogo cujos conteúdos são inteiramente criados procedimentalmente [YOUN06]

2.2. Modelação Procedimental de Ambientes Virtuais

Desde o aparecimento dos jogos que têm sido necessários ambientes onde estes são jogados. Mesmo os mais tradicionais, como o xadrez, carecem de um tabuleiro que ajude a definir a forma como se deve jogar. Na verdade, o conceito de "jogo de tabuleiro" surge dessa mesma dependência da jogabilidade do ambiente que, quando inexistente, perde parcial ou completamente o seu sentido.

Na sua passagem para vertente digital em computador, máquinas de arcada e consolas de jogos, esta necessidade não desapareceu. Na verdade, dadas as possibilidades que as plataformas disponibilizavam, a criação de ambientes diversos começou a tornar-se num aspecto essencial não só conferia um aspecto único ao jogo, mas permitia a expansibilidade de um jogo através da definição de "níveis". Dois jogos bastante conhecidos são o *Pacman* e *Lemmings*, que apesar das regras e objectivos do jogo se manterem ao longo dos níveis, a disposição dos vários elementos do ambiente (obstáculos, inimigos, saídas) conferiam uma experiência única a cada um, fornecendo assim horas e horas de diversão ao jogador.

À medida que as plataformas de jogos electrónicos foram evoluindo, com maior capacidade de armazenamento e processamento, o mesmo foi acontecendo com os designs dos ambientes. Estes tornaram-se mais interactivos e complexos, passando-se da bidimensionalidade para a tridimensionalidade. Foi-se encorajando o jogador a explorar melhor os terrenos que lhe eram propostos, fornecendo inclusivamente prémios na descoberta de câmaras secretas. Mais tarde, com a introdução de narrativas e objectivos mais rebuscados, a disponibilização de um ambiente imersivo e convincente tornou-se fundamental para o sucesso dos videojogos. Muitos foram os títulos que se destacaram especialmente pelos ambientes complexos que providenciavam.

Figura 3 – *Fallout 3*: Um título que providencia ambientes detalhados e de grande escala (Autodesk, 2009).

Contudo, o desenvolvimento de jogos deste calibre torna-se muito exigente em termos de recursos, sendo assim pouco viável a sua criação através de ferramentas e métodos de modelação manuais. Neste sentido, a modelação procedimental tem mostrado ser uma solução interessante, com resultados promissores. Pretende-se assim explorar algumas alternativas existentes e de que forma é que têm respondido aos vários desafios de criação de ambientes tridimensionais em jogos de computador.

2.2.1. Conceitos Gerais no Desenvolvimento de Ambientes Virtuais de Jogos

O crescente desenvolvimento de ambientes virtuais para jogos tem contribuído para o aumento das expectativas dos jogadores em relação a eles. De forma a tornarem-se imersivos, interessantes e imersivos, devem ser concebidos com alguma ponderação, tomando bem em conta o seu objectivo no jogo em que são integrados para decidir quais os elementos a serem utilizados e como os dispor. A criação destes não é, assim, uma tarefa mundana, pelo que é necessário estar a par de algumas noções e conceitos para se conseguir produzir bons ambientes virtuais.

2.2.1.1. Desafios na Concepção

Quando introduzido num contexto de um jogo de computador, o conceito de ambiente virtual impõe um conjunto superior de processos de forma a cumprir o seu papel na aplicação. Enquanto num simulador como o *Google Earth* [GOGL10], o propósito reside na simples exploração do ambiente recriado com fins informativos, os ambientes virtuais em jogos de computador já impõem requisitos a nível de entretenimento. Um estudo realizado [SWTR04] apresentou alguns elementos que os jogadores consideram essenciais num ambiente virtual de um jogo:

- **Imersão e Suspensão de Descrença:** Um dos elementos que os jogadores consideram aumentar a sensação de imersão é a existência de pormenores gráficos e sonoros que se ajustem à atmosfera que se pretende transmitir num determinado momento. Da mesma forma que sons podem aumentar o *suspense* num filme de terror, é importante que questões gráficas como iluminação e texturas os

acompanhem. Da mesma forma, é necessário que, uma vez estabelecido um certo tema, o ambiente virtual do jogo seja ajustado. É necessário também que o grafismo seja consistente, pois nada saltará mais à vista do que um elemento que visivelmente não se enquadra naquilo que o circunda.

- **Interactividade:** Um dos aspectos muito criticados pelos jogadores é o facto de as suas interacções com o ambiente serem muito limitadas. Na maioria dos casos, os objectos presentes em ambientes virtuais constituem simplesmente o cenário, não podendo ser movidos ou afectados. As peças de mobiliário não são amovíveis, as paredes não podem ser danificados e os papeis não podem ser lidos. Poucos têm sido os títulos que implementaram um nível de interactividade mais avançado, mas a verdade é que os resultados têm sido bem recebidos.
- **Consistência:** Um dos aspectos mais importantes é a percepção de que objectos da mesma natureza actuem da mesma forma. Um exemplo do não cumprimento desta regra serão "os vidros que por vezes partem, e outros que não partem". Estas inconsistências podem causar dificuldades ao jogador em aprender as regras de um jogo, que aparenta estar em constante mudança. É importante assim, que quando um objecto tem uma aplicação diferente, que possua alguma característica que demonstre que um tipo diferente de interacção é possível.

O cumprimento destas e outras necessidades em ambientes virtuais em jogos de computador constitui uma vertente que ainda não tem sido muito trabalhada no desenvolvimento através de métodos procedimentais. Por outro lado, constitui também uma potencialidade, na medida em que, através da definição de um conjunto de regras, se poderá rapidamente recriar uma determinada atmosfera num ambiente virtual, no qual bastantes elementos poderão possuir meios de interacção diferentes e consistentes.

2.2.1.2. Processos de Concepção

Uma vez que na maioria toda a jogabilidade passa pelos ambientes, a criação destes abarca a maioria dos processos de desenvolvimento de um jogo. Embora varie conforme o género, estilo e título, é possível definir alguns passos mais comuns na criação de ambientes tridimensionais [BYRN05, GLZN09]:

- **Criação do espaço para o movimento das personagens:** Em ambientes externos, tipicamente passará pela concepção de uma camada térrea do mapa, com algum relevo. Em ambientes internos, é mais comum o desenvolvimento de um conjunto de salas, corredores, átrios, túneis e outros complexos.
- **Passagens:** Criação de elementos de transição entre áreas externas e internas (ou entre áreas internas), tal como portas, janelas ou elementos mais complexos.
- **Plataformas Movediças:** Definição de plataformas amovíveis que sirvam, por exemplo, para transporte do jogador entre diferentes locais ou que abram novas passagens.

- **Decoração:** Distribuição de objectos que compõem o ambiente (árvores, rochedos, caixas, etc.), podendo contribuir de alguma forma para o desenvolvimento da personagem.
- **Objectos Usáveis:** Disposição de objectos que o jogador pode recolher e utilizar, tal como armas, munições, caixas ou outros recursos.
- **Entidades:** Colocação das personagens com que o jogador poderá interagir (tais como aliados ou inimigos)
- **Início e Fim:** Definição dos pontos que indicam o ponto de começo e de fim.
- **Pontos Objectivo:** Estabelecimento de um conjunto de pontos que o jogador tem de atingir de forma a completar o desafio proposto pelo jogo.
- **Regiões:** Definição de regiões ou áreas com características específicas (zona de recolha de recursos, zonas de água).
- **Áreas de Eventos:** Definição de locais onde determinados eventos poderão ocorrer.

Embora já se tenha assistido à recorrência, em alguns jogos, de métodos de modelação procedimental para a automatização de alguns destes processos, outros ainda são realizados manualmente. Tal reside sobretudo nas próprias limitações das ferramentas de modelação procedimental, que ainda não são capazes de reproduzir esses processos, sendo assim necessária a procura de novas soluções.

2.2.2. Geração de Ambientes Urbanos Virtuais

O desenvolvimento de ambientes urbanos em ambientes de jogos de computador tem constituído sempre um desafio para os criadores. Tal explica-se sobretudo pela necessidade de modelação de um elevado número de edifícios, os quais deverão possuir características distintas, variadas e preferencialmente detalhadas, de forma a evitarem a monotonia e conferir alguma unicidade a cada secção, bem como um maior realismo e imersão nos ambientes.

Uma das tendências que se tem sentido cada vez mais em jogos recentes é a tentativa de recriação de ambientes urbanos existentes. Apesar de já se ter assistido à modelação de cidades reais no passado em muitos jogos, tais como simuladores de aviões, a aproximação à realidade nunca foi muito explorada, tendo ficado limitada à introdução de um conjunto pequeno de edifícios reais mais emblemáticos como forma de permitir ao jogador o reconhecimento da cidade que se encontrava a visitar.

A introdução destes elementos, no entanto, tem tido uma recepção bastante positiva, na medida em que o sentimento de familiaridade do jogador com o local contribui para uma maior imersão no jogo. Um exemplo de sucesso foi efectivamente o jogo *GTA IV*, cujos ambientes em que apresenta possuem muitas semelhanças com a cidade de Nova Iorque.

Figura 4 - Uma visão sobre a cidade de Nova Iorque no Jogo GTAIV [JOEY09]

Este facto, contudo, não surgiu sem um preço elevado em termos de concepção, na medida em que foram necessários 3 anos para a sua modelação completa [GDSN08]. A equipa de desenvolvimento reuniu vários arquitectos e *designers* que investiram muito tempo a investigar e reproduzir todo o cenário urbano do jogo.

É no sentido de ajudar neste processo que a modelação procedimental tem evoluído, tendo vindo a surgir várias soluções por diversos autores.

2.2.2.1. Fontes de Informação para Geração da Ambientes Urbanos

De forma a obter uma aproximação à realidade na modelação procedimental de ambientes urbanos virtuais é necessária a importação de alguma informação verídica. Contudo, os seus formatos, métodos de aquisição e nível de aplicação têm variado entre as abordagens dos vários investigadores, consoante o nível de detalhe pretendido e a correspondência desejada a ambientes existentes.

Muitos autores nesta área [FKZR08, MULL06, MULL07, WNKA03] têm optado pela construção de um conjunto de regras integradas em gramáticas formais (tal será abordado nas secções seguintes) que definam os elementos constituintes mais comuns dos edifícios (janelas, portas, colunas, pilares, telhados, arcos, paredes, ornamentos, etc.) e a forma como estes se interligam. A determinação destas regras tem sido realizada manualmente sobretudo através da observação das características de edifícios em geral, consulta de livros da área da arquitectura [WATS08] ou, mais recentemente, de fotografias das fachadas dos edifícios [MULL07]. Estas abordagens têm demonstrado resultados de qualidade elevada, possibilitando a geração rápida de uma grande diversidade de áreas urbanas virtuais. No entanto, para estabelecer uma correspondência entre estes e áreas urbanas reais, estas fontes de informação nem sempre são suficientes.

Na sua aplicação denominada CityEngine [PRCI09], Parish e Müller [PRSH01] tentaram reconstituir cidades com base em alguns aspectos característicos das mesmas, tais como a distribuição demográfica e económica, que muito contribuem para a organização espacial dos edifícios. Assim sendo, mapas demográficos e estatísticos foram suficientes para obter resultados aproximados, mas certamente não exactos.

No entanto, muitos outros autores [BRNT08, COEL07, JNHU03, SLVR06] têm procurado utilizar informação mais precisa para a criação dos ambientes virtuais, com o intuito de os aproximar à realidade, tais como:

- Informação Global obtida através fotografias aéreas e mapas bidimensionais (orográficos, rodoviários, etc.)

- Informação local da geometria dos edifícios e texturas das fachadas captada através de dispositivos de varrimento laser e câmaras fotográficas
- Modelos Digitais de Elevação de Terreno (DEM)
- Sistemas de Informação Geográfica (GIS)
- Outras bases de dados urbanísticas

No entanto, apesar da grande diversidade de fontes de informação, nem sempre é simples a automatização do processo interpretação e junção dos vários dados. Neste sentido, [COEL07] sugere a integração de alguns desses dados através das propriedades geoespaciais dos objectos neles contidos, nomeadamente a sua posição absoluta ou relativa no planeta (georreferenciada).

Um facto importante a reter é que nem sempre todas estas fontes são suficientes para descrever todo um espaço urbano. É neste sentido que a introdução de técnicas de amplificação de informação é necessária, tais como a inclusão de alguma aleatoriedade em partes dos modelos em que a informação é escassa e em que o nível de detalhe não é essencial [COEL07].

2.2.2.2. Modelação de Terreno e Redes Rodoviárias

Em qualquer ambiente virtual tridimensional, é elementar a criação de uma superfície que serve como base onde todos os objectos se apoiam e deslocam. Num edifício fechado tal será o chão de cada piso, uma superfície essencialmente plana e limitada, tipicamente com uma textura simples e constante. Contudo, já em ambientes exteriores, não é incomum que esta tenha variações ao longo de toda a sua extensão, tal como diferentes tipos de terreno e variados níveis de elevação.

A modelação procedimental de terrenos de elevação irregular é um processo já muito explorado, com resultados muito interessantes por parte de diversos autores e produtos comerciais [DISC09, ZHOU]. Hoje em dia é possível encontrar facilmente um conjunto de algoritmos de aplicação simples, tal como através da introdução de ruído, capazes de gerar terrenos montanhosos semelhantes ao que se podem encontrar na natureza [MRTZ96].

No entanto, com vista a reproduzir terrenos existentes, é preferível a recorrência a informação real tal como fotografias ou mesmo sistemas de informação geográfica, que por sua vez possuem normalmente a informação do relevo num modelo digital de elevação de terreno (DEM – Digital Elevation Model). Este tipo de modelo representa a superfície sobre a forma de uma malha poligonal, contendo em cada ponto o valor correspondente à sua altitude. O facto de os pontos se encontrarem dispostos de forma regular permite que sejam inferidas associações bastante interessantes, tal como o cálculo de visibilidade a partir de um determinado ponto ou a determinação de encostas mais íngremes [HRDU09].

Os ambientes urbanos, que se encontram assentes neste tipo de superfícies, estão assim sujeitos às dificuldades de colocação de estradas e edifícios sobre as mesmas. No exemplo da cidade do Porto, cuja localização junto ao mar e a um rio teve implicações na orografia do terreno, poder-se-á reparar como as diferenças no relevo contribuíram para a disposição da rede rodoviária, para a distribuição dos edifícios e para a arquitectura das construções. Estas questões constituem assim ainda bastantes desafios para a modelação procedimental.

Bruneton e Neyret [BRNT08] apresentaram um método que possibilita a integração detalhada de vários elementos tais como estradas, rios e lagos em terrenos de vasta dimensão, combinado para tal um modelo digital de elevação do terreno com informação vectorial dos

vários objectos de forma a tornar a junção de ambos mais realista. De forma a conseguir enquadrar a possibilidade de terrenos extensos com detalhes precisos, é utilizado um esquema de refinamento baseado em quadtrees dependente do campo de visão e nível de aproximação. Dependendo do detalhe necessário é efectuada uma rasterização da informação vectorial na resolução apropriada, permitindo assim a correcta adaptação dos vários elementos ao relevo da superfície.

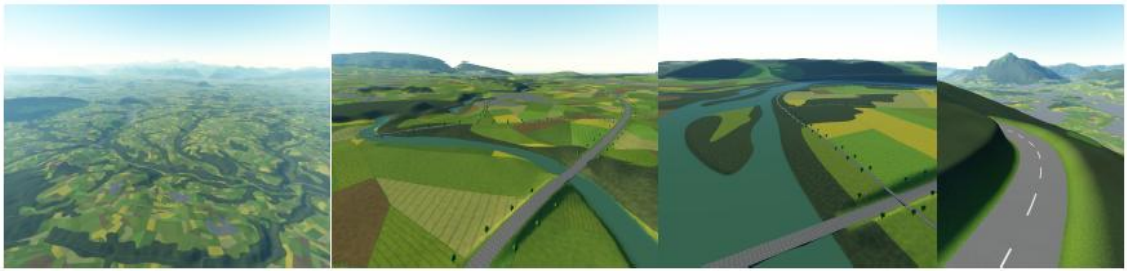

Figura 5 - Colocação de elementos (estradas, pontes, rios) em superfícies de grande dimensão e terreno irregular [BRNT08].

No entanto, esta abordagem, apesar de ser muito indicada para a colocação de estradas, um elemento fundamental em ambientes urbanos, ainda não contempla a disposição de edifícios, cuja maior dimensão apresenta também maiores desafios de adaptação ao terreno.

Na concepção de áreas urbanas, as estradas constituem os elementos fundamentais que definem a distribuição dos edifícios numa determinada área. Por essa mesma razão, em algumas abordagens, a modelação destas tem sido efectuada como primeiro passo [PRSH01, SLVR06].

Parish e Müller [PRSH01] apresentaram numa primeira versão da sua aplicação, CityEngine, um sistema capaz de modelar grandes cenários urbanos através de um conjunto relativamente pequeno de dados estatísticos e geográficos e possibilitando a definição adicional de um conjunto de regras definidas pelo utilizador. Esta informação é depois utilizada na definição dos parâmetros que definem Sistemas L, que por sua vez procedem à criação da grelha rodoviária.

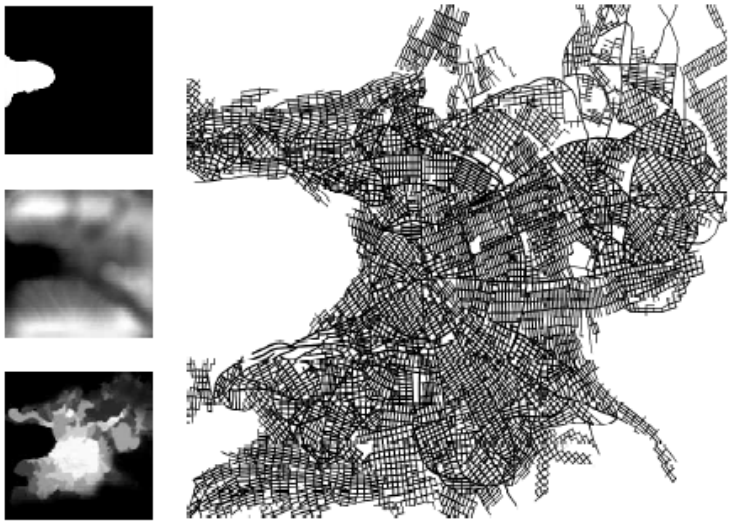

Figura 6 - Esquerda: Mapas com água, elevação e densidade populacional. Direita: exemplo de um mapa rodoviário gerado a partir destas fontes de informação [PRSH01].

Os sistemas L (um acrónimo para Sistemas Lindenmayer) foram desenvolvidos por Aristid Lindenmayer na simulação de crescimento de plantas [PSKZ96]. Os sistemas L baseiam-se no conceito de reescrita de cadeias de caracteres. A ideia consiste em utilizar um conjunto de regras de produção para criar objectos complexos através de uma substituição sucessiva de partes de um objecto simples. Trata-se assim de uma forma de amplificação de dados. Embora partindo do mesmo princípio das gramáticas de Chomsky [CSKY56], difere delas na medida em que as gramáticas são aplicadas sequencialmente, enquanto em sistemas L as mesmas são aplicadas em paralelo [FKZR08].

Recorrendo às facilidades produção paralela proporcionada pelos sistemas L, Parish e Müller [PRSH01] conseguiram facilmente implementar o esquema ramificado de estradas, uma vez criadas as regras de produção. A partir de um pequeno segmento de estrada, novos segmentos são adicionados procedimentalmente, crescendo assim uma rede de estradas, da mesma forma que uma planta cresce. No entanto, apesar de este algoritmo produzir resultados de grande qualidade, o utilizador não tem grande poder de alteração sobre os elementos criados, sendo difícil a definição de pontos ou estradas com determinadas características.

Nesse sentido, uma alternativa foi apresentada em [CHEN08], que permite ao utilizador a definição de um campo tensorial que guia a geração da rede rodoviária. Um exemplo que permite uma melhor visualização do processo encontra-se na figura seguinte:

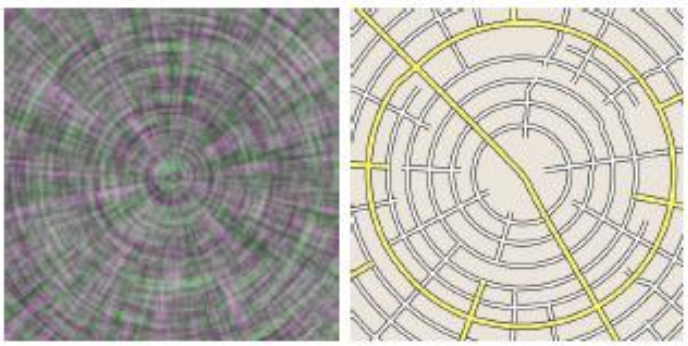

Figura 7 - Um padrão de ruas gerado procedimentalmente (direita) a partir de um campo tensorial (esquerda) [CHEN08].

Também destinado a conferir um maior poder de configuração ao utilizador foi uma solução apresentada por [KLLY] no seu sistema *CityGen*. Este procura ser interactivo, na medida em que permite ao utilizador realizar alterações a diversos parâmetros e verificar os seus resultados em tempo real. O esquema de funcionamento consiste na geração inicial das estradas principais e auto-estradas, sendo depois as estradas secundárias geradas entre as anteriores.

A rede de estradas principais é representada por um grafo não direccionado e por listas ligadas, usando nós que podem ser definidos e alterados pelo utilizador em tempo real usando o programa de geração. As estradas são então geradas entre os nós, adaptando-se ao terreno sobre as quais são colocadas. A geração de estradas secundárias começa pela divisão do espaço em células, preenchendo depois estas áreas com ruas através de Sistemas L. O procedimento pode ser alterado de forma a gerar vários tipos diferentes de ruas (Figura 8).

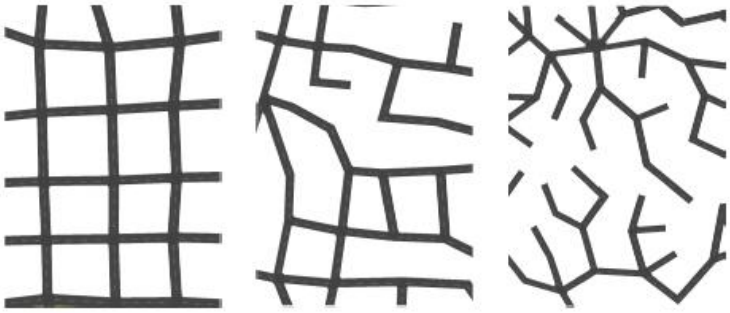

Figura 8 - Padrões de desenho de ruas existentes no Citygen: Raster, Industrial e Orgânicas [KLLY]

2.2.2.3. Modelação Procedimental de Edifícios

Nos vários trabalhos enunciados na secção anterior, a geração de redes rodoviárias constitui o primeiro passo de modelação de ambientes urbanos, sendo o seguinte a geração de edifícios nos lotes resultantes da divisão do espaço pelas estradas criadas.

A abordagem em [PRSH01] apresentava uma solução para construção dos edifícios também através de Sistemas L, consistindo estes de simples primitivas e recorrendo a *shaders* para produzir detalhe nas fachadas. Contudo, segundo Wonka e Müller [MULL06, WNKA03], os sistemas L não mostraram ser uma boa alternativa para este fim, uma vez que se aplica sobretudo à simulação de crescimento em espaços abertos. Os edifícios, por sua vez, possuem restrições de espaço maiores e a sua estrutura normalmente não reflecte um processo de crescimento [MULL06, WNKA03].

Em [WNKA03], Wonka introduziu gramáticas de cisão (*split grammar*), um tipo de gramática formal que se baseia em formas geométricas, para a modelação de estruturas arquitecturais. Partindo de uma forma não-terminal simples e de um conjunto de regras de produção, é realizada uma cisão da forma em múltiplas não-terminais mais pequenas, e finalizando em formas terminais como janelas ou outros ornamentos em paredes. Este procedimento encontra-se representado na figura seguinte:

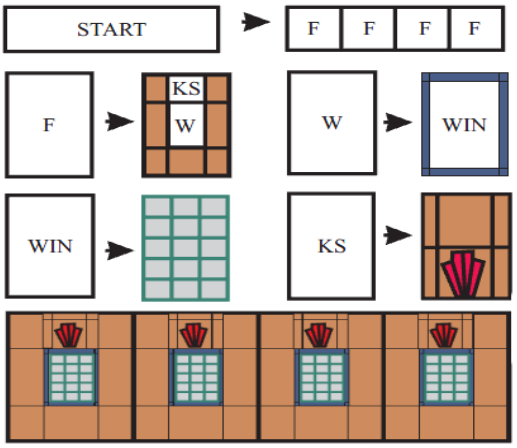

Figura 9 - Esquema de funcionamento de uma gramática de cisão, sendo a forma START o ponto de partida e o conjunto de janela em baixo o resultado final [WNKA03]

Utilizando uma base de dados recheada com largas quantidades de regras, pode ser produzida uma variedade de resultados diferentes, não sendo necessária a definição de uma gramática individual para cada objecto a ser modelado. Devido à complexidade de uma gramática deste calibre, existem várias regras que podem ser escolhidas em cada passo da derivação, pelo que é utilizada uma segunda gramática, denominada de gramática de controlo, que permita coerência nas escolhas realizadas em cada derivação, de modo a que todos os elementos das casas sigam um estilo comum.

A abordagem sugerida por Wonka permitiu assim a modelação procedimental de edifícios com um elevado nível de detalhe geométrico. Contudo, a utilização de gramáticas de cisão apenas se aplicavam a modelos iniciais simples. Com o objectivo de evoluir estas gramáticas para ultrapassar esta limitação, Müller apresentou o conceito de *CGA Shape*, que gera procedimentalmente variações do modelo de massa usando formas volumétricas (cubos, cilindros, etc.), procedendo depois à geração de detalhe consistente com o modelo [MULL06].

A gramática funciona com a configuração de formas. Uma forma consiste de um modelo (uma cadeia de caracteres), geometria (atributos geométricos) e atributos numéricos. Formas são identificados pelos seus símbolos que podem ser terminais ou não-terminais. O sistema de produção realiza-se através da modificação e substituição de formas, numa evolução iterativa. É atribuída a cada regra uma prioridade de derivação, para que esta ocorra com um nível de detalhe crescente mas controlado. Tal como nas gramáticas de Chomsky [CSKY56], a aplicação das regras de produção é realizado de forma sequencial.

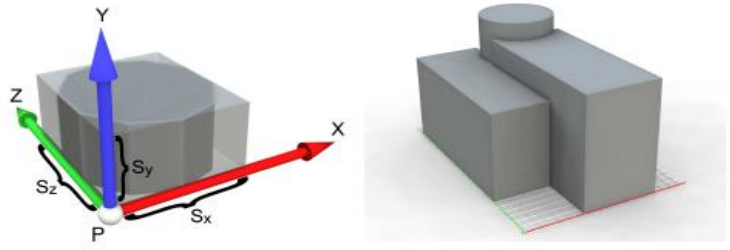

Figura 10 – Esquerda: Âmbito de uma forma. Direita: Um modelo composto de três primitivas [MULL06].

Os modelos em massa podem ser vistos como a união de formas geométricas mais simples. Assim sendo, através da aplicação de um conjunto de transformações geométricas sobre blocos primitivos, formas geométricas mais complexas poderão ser criadas.

Dois mecanismos existem com o objectivo de tornar a qualidade dos edifícios mais consistente: o teste de sobreposição espacial e o teste de aproximação de linhas ou planos importantes. O teste da sobreposição espacial, ou oclusão, verifica se uma forma se encontra visível, parcialmente oculta ou totalmente oculta por outras. O teste de aproximação de linhas ou planos importantes permite evitar situações tais como a sobreposição parcial de uma janela por parte de uma parede, ou seja, situações que arquitecturalmente e esteticamente não se encontram em ambientes reais.

A aplicação deste novo processo de criação de ambientes urbanos virtuais encontra-se enquadrada na versão mais recente do sistema CityEngine, anteriormente referido. Os resultados produzidos encontram-se entre os mais detalhados apresentados até ao momento por métodos de modelação procedimental.

Figura 11 – Aplicação das CGA Shapes na modelação de edifícios detalhados[MULL06].

Uma outra abordagem mais simples mas também interessante foi apresentada em [MREL07] e baseia-se no conceito de síntese de modelos, semelhante à técnica de síntese de texturas. O objectivo consiste em que o *designer* particione um pequeno exemplo usando uma grelha tridimensional. Esta partição depois gera um conjunto de restrições automaticamente: duas partes só podem ser adjacentes no modelo gerado se estiverem adjacentes no modelo exemplo. Embora seja bastante limitada devido à sua disposição em grelha (que na criação de áreas urbanas raramente é aplicável), os resultados são bastante complexos mas simples de criar.

Figura 12 - Dados alguns edifícios num modelo exemplo (esquerda), cidades complexas podem ser geradas (direita)[MREL07].

As abordagens referidas anteriormente revelaram ser apropriadas para a modelação procedimental de ambientes urbanos fictícios para jogos de computador, uma vez que apresentam níveis de detalhe impressionantes, bem como a possibilidade de criação de uma grande variedade de edifícios. No entanto, tal como referido no início da secção, ainda reside o problema de correspondência dos ambientes urbanos virtuais aos reais. Na modelação de uma cidade real, tais abordagens podem ser extremamente difíceis de empregar, uma vez que se carece de um conjunto extenso de regras ou exemplos, surgindo assim a necessidade de uma grande intervenção humana.

Com vista a colmatar este problema, a integração com sistemas de informação geográfica tem sido sugerida por alguns autores [COEL07, DOLN05, SLVR06], uma vez que muita informação georreferenciada de áreas urbanas poderá ser automaticamente extraída. Desta forma, é possível modelar edifícios em posições e tamanhos reais sem necessidade de intervenção humana.

Adicionalmente, a existência de informação possibilita o conceito de contextualização geoespacial [COEL07], ou seja, a habilidade de perceber as relações espaciais entre os vários

elementos urbanos distintos e o ambiente, permitindo assim a definição de algumas regras básicas de construção. Por exemplo, os edifícios cujas paredes estão muito juntas não deverão possuir varandas.

Figura 13 – Vista sobre uma reprodução virtual da Praça da República, no Porto [COEL07]

No trabalho de Coelho [COEL07] são apresentados os Sistemas L Geoespaciais, uma extensão dos Sistemas L paramétricos que incorpora contextualização espacial. Desta forma, é aproveitada a capacidade de ampliação de dados que os sistemas L proporcionam com os sistemas geoespaciais, que possibilitam a estruturação e análise espacial de informação georreferenciada. Esta solução encontra-se incorporada numa ferramenta, o modelador XL3D, que integra várias fontes de informação e possibilita assim a reprodução de ambientes urbanos existentes.

2.2.2.4. Modelação Procedimental de Fachadas

Várias são as características que, num ambiente urbano, definem cada edifício de forma única e distinta das suas congéneres. Entre as maiores estarão certamente as suas formas volumétricas e a sua disposição no espaço, bem como a disposição dos vários elementos na sua fachada, cujos pormenores são, por vezes, essenciais para constituir diferença.

Na recriação virtual de um ambiente urbano existente, é essencial a introdução dessas características, de forma a conferir ao observador um sentimento de familiaridade ao observador. É nesse sentido que certos autores têm apresentado solução de modelação procedimentais mais pormenorizadas, concentrando-se na representação realista das fachadas.

Com vista a expandir os seus trabalhos anteriores, Müller apresentou um sistema que, dada uma imagem de baixa resolução da fachada de um edifício, capaz de criar um modelo tridimensional de maior resolução, possuindo grandes semelhanças com a imagem introduzida [MULL07]. Adicionalmente, é realizada uma interpretação semântica das imagens, permitindo a inferência de regras gramaticais.

Figura 14 - Esquerda: Fotografia de uma fachada de edifício. Direita: modelo tridimensional da fachada[MULL07]

O processo é dividido em quatro fases individuais [MULL07]:

- **Detecção da estrutura das fachadas:** Subdivide a imagem com a fachada em andares (verticalmente) e em secções (*tiles*, as divisões horizontais dentro de um andar) através da detecção de informação mútua. O conceito de *tile* é importante na modelação procedimental que simboliza um elemento arquitectónico tal como uma janela ou porta incluindo a parede em volta.
- **Refinamento das secções:** Uma vez obtida uma divisão da fachada em secções, em que as mais semelhantes se encontram agrupadas, é realizada uma segmentação de secções em rectângulos mais pequenos. Este passo deriva das gramáticas de cisão.
- **Reconhecimento de elementos:** É realizada uma correspondência dos pequenos rectângulos com objectos tridimensionais de uma biblioteca de elementos arquitectónicos. Depois é gerado um modelo tridimensional com texturas, juntamente com a estrutura semântica numa árvore de formas.
- **Edição e Extracção de regras gramaticais:** Interpretação semântica de fachadas pode ser usado em várias operações de edição, incluindo a extracção de regras gramaticais de formas a partir da árvore de formas criada.

Apesar dos resultados de elevada qualidade e da grande verosimilhança entre as imagens e os modelos gerados, o sistema de processamento de imagem é muito sensível ao ruído nas imagens, o que dificulta a sua utilização em muito casos.

Uma outra aproximação de modelação procedimental de fachadas foi apresentada por Finkenzeller [FKZR08] com vista a produzir elementos de fachadas bastante mais detalhados do que Müller, tais como as armações, beirais e ornamentos que muitas vezes se podem encontrar em edifícios.

Nesta abordagem, é necessária uma actividade bastante maior por parte do utilizador, na medida em que lhe caberá a modelação de um esboço do edifício. Contudo, a especificação deste deverá ser bastante simples, servindo apenas para delimitar os contornos do edifício, o seu objectivo e os estilos pretendidos (Figura 1Figura 15, a).

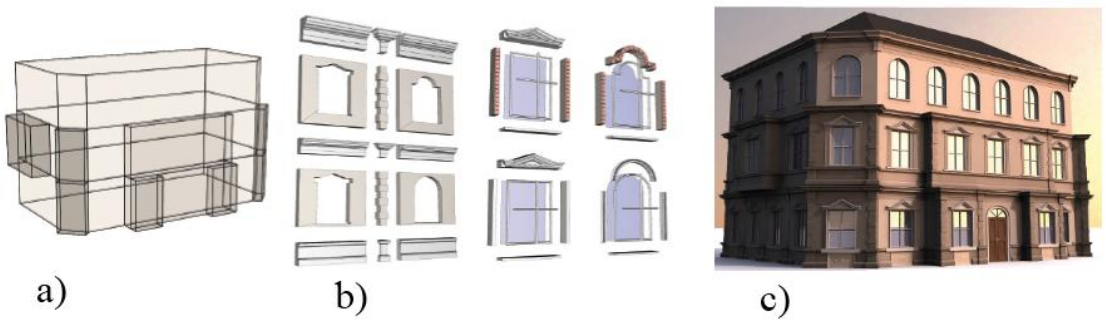

Figura 15 - Processo de modelação de fachadas sugeridas por Finkenzeller [FKZR08]

De seguida, o sistema extrai toda a informação espacial de todos os elementos da fachada e automaticamente identifica e adapta estruturas adjacentes, evitando a necessidade de cálculos de oclusão, como em [MULL06]. Só depois é que gera procedimentalmente elementos detalhados e com colocação precisa, tais como portas, janelas, beirais e outros ornamentos (Figura 15, b).

Por fim, o sistema guarda a fachada numa estrutura hierárquica que reflecte a representação simbólica da mesma, ou seja, gera uma parametrização de toda a fachada. Como consequência, os utilizadores podem alterar parâmetros da fachada num nível mais elevado e produzir estruturas mais complexas em menos tempo (Figura 15, c).

2.2.2.5. Modelação Procedimental de outros Conteúdos Urbanos

Um ambiente urbano é muito mais do que edifícios rodeados de estradas. É neste sentido que é relevante a modelação de outros conteúdos comuns nestes espaços, tais como postes, bancos, semáforos, placas, e outro mobiliário urbano de utilidade pública. A introdução de árvores e outros elementos verdes são também essenciais na constituição de paisagens urbanas virtuais [COEL07, SLVR06].

O processo de modelação deste tipo de conteúdos é relativamente simples, na medida em que as variantes, que ocorrem dentro de cada classe de objectos, são simples de catalogar. Por exemplo, em termos de caixotes do lixo, não existe uma grande variedade de alternativas, visto estes serem encomendados em grandes quantidades e espalhados pelas cidades. Tal procedimento é recorrente na maioria do mobiliário urbano, podendo os modelos ou as instruções de geração destes ser guardada numa base de dados e reutilizada conforme desejado.

Outros conteúdos naturais, tais como árvores, que já apresentam variações maiores mesmo dentro de cada classe, a utilização de diferentes modelos para árvores do mesmo tipo é essencial para conferir alguma noção de diversidade. Neste sentido, a utilização de Sistemas L para criação desta diversidade tem sido abordada em diversos trabalhos [MECH, PSKZ94, PSKZ96].

Embora a modelação dos conteúdos referidos seja bastante simples, o mesmo não se aplica à sua colocação e orientação no terreno, visto esta seguir, em ambientes urbanos reais, regras bastante específicas. Por exemplo, não faz sentido a existência de objectos deste tipo no meio de estradas, nem à frente das portas das casas, ou a orientação de placas ou semáforos no sentido contrário. É, por isso, essencial que a disposição destas não seja realizada de uma forma puramente aleatória, mas que seja dirigida por um conjunto de regras. Um exemplo de regras

será a limitação de colocação de objectos às zonas pedestres e à orientação de estações de autocarro para as ruas.

Uma outra possibilidade para além da definição de regras é a utilização de informação georreferenciada sobre o mobiliário urbano, que pode estar presente em bases de dados e sistemas de informação geográfica. No seu trabalho, Coelho utiliza as duas [COEL07]. Como visto anteriormente, na sua abordagem é utilizada a informação geoespacial para o correcto posicionamento dos vários elementos, bem como o estabelecimento de relações entre eles (contextualização geoespacial), que permite a sua correcta orientação no espaço. No caso de a informação necessária não estar disponível ou ser insuficiente, o recurso aos Sistemas L geoespaciais que Coelho introduz permite assim ampliar a informação que se possui, através da definição das regras e de uma certa componente de aleatoriedade.

Figura 16 - Vista sobre uma área verde repleta de mobiliário urbano e vegetação, adquirida do trabalho de Coelho [COEL07].

2.2.2.6. Questões em Aberto na Modelação Procedimental de Ambientes Urbanos Virtuais

O recurso a métodos de modelação procedimental com vista ao desenvolvimento de ambientes urbanos virtuais em jogos de computador tem vindo cada vez mais a apresentar soluções inovadoras, provando que o seu emprego na área de desenvolvimento de jogos de computador é cada vez mais recompensador.

As abordagens referidas apresentam todas métodos únicos, que de certa forma contribuem para resultados cada vez melhores, mais realistas, ou mesmo mais simples de gerar. Assim sendo, a combinação destas estratégias possibilita assim a concepção de soluções melhores e mais abrangentes, na medida em que poderá aproveitar o melhor de cada um.

No entanto ainda existem algumas questões em aberto para serem revolvidas e aspectos a serem melhorados, sobretudo na interacção humana necessária, tais como:

- **Aprendizagem automática de regras gramaticais:** Um dos desafios referentes à utilização de modeladores procedimentais que recorrem a gramáticas formais é a definição das regras de produção que gera os modelos desejados. Apesar de algum trabalho já ter sido efectuado relativamente às fachadas [MULL07], ainda é necessária investigação neste sentido. A interpretação e estabelecimento de relações entre objectos e edifícios, através de fontes como GIS, são hipóteses a serem exploradas.

- **Bibliotecas de regras:** Alterar uma regra existente (como uma gramática) de forma a corresponder a novos requisitos é mais simples do que analisar um determinado *design* e defini-lo desde o início. Assim sendo, a criação e disponibilização de bibliotecas regras pode ajudar no processo de criação de novas cidades.
- **Definição de regras através de linguagem natural:** A criação de regras de gramáticas nem sempre é intuitiva devido à forma como estas têm de ser construídas. Assim sendo, a definição destas através de descrições em linguagem natural poderiam tornar tais processos mais acessíveis para todos. Por outro lado, possibilitaria a utilização de fontes como livros e outros relatórios arquitectónicos. Tal abordagem já começou a ser explorada em [RDGS09], mas ainda carece de algum desenvolvimento.
- **Definição de interfaces gráficas de construção de regras:** Para que os designers e arquitectos possam utilizar as ferramentas de modelação procedimental de forma mais simples e intuitiva, é necessário a que a sua intervenção seja realizada um nível mais alto [COEL09]. Tal poderá ser resolvido através da introdução de interfaces gráficas mais intuitivas e/ou através da modelação de exemplos.

2.3. Sumário

A modelação procedimental tem-se tornado cada vez mais uma ferramenta indispensável no desenvolvimento de jogos de computador. Cada vez mais editoras têm apercebido que, de forma a conseguirem competir num mercado repleto de títulos de grande sucesso, é necessária a satisfação de certas exigências crescentes por parte dos jogadores, tais como os conteúdos. A concepção em grande quantidade e qualidade destes é efectivamente um deles, algo que através de meios manuais se torna extremamente custoso e em muito casos inviável.

Neste capítulo verificou-se que modelação procedimental é uma solução cada vez mais utilizada para auxiliar nas tarefas mais exigentes de desenvolvimento de jogos de computador, e sobretudo na criação dos ambientes virtuais em que os jogos se desenrolam. As ambições sobre estes têm crescido, procurando-se já a reprodução fiel de ambientes urbanos existentes. Como se constatou, muitos autores têm vindo a apresentar soluções de modelação procedimental com vista à automatização quase total do processo de desenvolvimento desses ambientes, mas ainda há muito terreno para ser explorado. Muitas são as questões que permanecem em aberto e que necessitam de ser resolvidas.

Capítulo 3

A Solução PG3D

O problema da modelação de ambientes urbanos realistas apresenta vários desafios aos quais muitos autores têm tentado responder ao longo do tempo. Estes englobam por exemplo a correspondência visual em ambientes reais, a utilização das fontes de informação, ou mesmo a definição de ferramentas suficientemente poderosas capazes de modelar estruturas existentes. No desenvolvimento de jogos de computador são impostas questões adicionais ao nível da qualidade dos modelos, do desempenho na sua visualização e no seu formato para poderem ser empregues.

Neste capítulo pretende-se apresentar uma solução de modelação procedimental de ambientes urbanos virtuais para aplicação a jogos de computador, baseada numa abordagem tecnológica inovadora que, por sua vez, possibilita a conjunção de conceitos existentes numa vertente original.

Numa primeira parte abordar-se-á em pormenor as várias dificuldades inerentes à modelação de ambientes urbanos, apresentando-se de seguida o conceito de sistema PG3D como contribuição para a resolução dos mesmos, no âmbito das bases de dados espaciais. Depois será abordado o emprego das gramáticas formais no domínio da computação gráfica, como introdução às Gramáticas PG3D, e às formas como a sua criação se baseia em conceitos diversos já explorados por diversos autores, mas pouco trabalhados conjuntamente. Seguidamente será realizada uma especificação destas novas gramáticas, explanando os conceitos envolvidos. Terminar-se-á o capítulo com as formas de interpretação gráfica dos resultados produzidos por esta abordagem, nomeadamente no âmbito da sua visualização, exportação e utilização por jogos de computador.

3.1. Modelador PG3D

Um dos denominadores comuns nas aplicações de modelação procedimental consideradas na revisão bibliográfica consiste na sua plataforma de execução e na sua plataforma de execução e na forma como opera sobre ela. Geralmente, são concebidas sob a forma de um programa independente, que realiza os procedimentos de modelação em memória e efectua a gravação final dos seus dados no sistema de ficheiros que o sistema operativo suporta. O acesso às fontes de informação real é feito de forma remota, incorrendo-se assim em tempos de consulta periódicos e que, dada a sua frequência, se tornam elevados.

O conceito PG3D apresenta assim uma metodologia de desenvolvimento de aplicações de modelação procedimental, sendo a sua implementação concreta denominada de Modelador PG3D.

3.1.1. Dificuldades na Modelação de Ambientes Urbanos Virtuais

Apesar de vários autores já terem conseguido obter resultados com fidelidade visual bastante elevada, em dimensões consideráveis e, em certos trabalhos, com grande correspondência a ambientes reais, as aplicações criadas para modelação ainda continuam a encarar dificuldades nos seus processos de modelação que ainda limitam o alcance dos resultados, ou mesmo a sua aplicabilidade na área dos jogos de computador. Entre outros, podem-se enumerar alguns dos mais relevantes:

- **Definição da estrutura:** Para conseguir construir e representar os vários elementos que constituem os ambientes virtuais, é necessário conceber uma estrutura suficientemente poderosa e genérica, suportando uma panóplia de operações suficientemente abrangente que possibilitem a reprodução de qualquer estrutura que exista na realidade.
- **Facilidade de expressão através de definição de regras**: Para que a seja fácil a um utilizador conceber uma estrutura que imagine, convém que a definição das instruções de modelação seja realizada da forma mais simples e intuitiva possível. No entanto, tal nem sempre é possível se se pretender corresponder ao ponto anterior.
- **Correspondência a ambientes urbanos existentes**: A modelação de ambientes reais com grande fidelidade visual carece obviamente de fontes de dados que contenham essas informações (ver secção 2.2.2.1.). Contudo, essas fontes nem sempre se encontram disponíveis ou com níveis de detalhe suficientemente elevados.
- **Gestão de múltiplas fontes de dados**: O acesso interoperável a múltiplas fontes de dados tem vindo a ser trabalhado [COEL07], mas dado a diversidade de formatos que cada fonte pode assumir, é necessária sempre a transformação dos mesmos para um formato comum, algo que exige sempre uma certa perícia e experiência daquele que os pretende utilizar.
- **Dimensão e rapidez no acesso às fontes de dados**: Uma vez que as fontes de dados poderão ter dimensões demasiado elevadas para ser carregadas completamente para memória, é necessário o recurso a técnicas que permitam minimizar o número de acessos às mesmas ou, simplesmente, realizá-las com maior rapidez.
- **Rapidez dos processos de modelação:** Embora seja sempre mais rápido e compensador do que a modelação manual, os processos procedimentais poderão também ter tempos de geração bastante extensos, pelo que a redução dos mesmos é sempre um elemento a aperfeiçoar.
- **Dimensão dos dados carregados e dados gerados:** Tipicamente, para permitir uma maior velocidade no processamento dos dados, estes são carregados para a memória, onde o seu acesso é obviamente mais rápido. Contudo, apesar das constantes evoluções de hardware a este nível, a quantidade deste recurso é consideravelmente limitada, especialmente quando comparado com a informação que pode ser necessário carregar e operar. Assim sendo, este aspecto pode constituir

uma restrição que impede a realização deste tipo de processos em qualquer máquina, e especialmente para pedidos de maior dimensão.

- **Formato dos dados gerados em aplicações de visualização em tempo real:** Os jogos de computador, por exemplo, contam-se entre as aplicações mais carentes de recursos computacionais, dado o facto de requerem uma visualização e interacção constante com o ambiente. A utilização dos dados gerados requer assim o recurso a técnicas de visualização e organização das estruturas de dados bastante complexas, para que o uso dos mesmos não tenha grande impacto no desempenho da aplicação.
- **Expansibilidade para múltiplas plataformas:** A possibilidade de exportação dos dados para múltiplas plataformas é um factor chave para o seu grau de aplicabilidade. Dado o facto de até agora ainda não ter sido possível impor um formato padrão para distribuição de conteúdos tridimensionais (embora existam algumas tentativas com mais sucesso, tais como o X3D ou COLLADA), convém que a aplicação seja suficientemente flexível para conseguir exportar os dados gerados para qualquer formato. No ramo dos jogos de computador, este aspecto é fundamental, dado que cada motor de jogo costuma apenas operar sobre o seu formato específico.

3.1.2. Definição do Conceito PG3D

O conceito de PG3D surge com o intuito de tentar abordar as dificuldades acima enumeradas, empregando para tal algumas ideias tentadas por diversos autores na área da modelação procedimental, tal como enumeradas no capítulo 2.2.

O nome PG3D é um acrónimo para ***P****rocedural* ***G****eneration* ***3D*** (Geração Procedimental Tridimensional), o que aparentemente poderá ser uma designação demasiado genérica e aplicável a qualquer aplicação de modelação procedimental. Na realidade, as iniciais correspondem a mais do que um significado, tendo nascido no processo de implementação do mesmo, sendo por isso parcialmente relacionado com as tecnologias em que foi concebido (como irá ser visto mais à frente).

A ideia fundamental por detrás do conceito PG3D consiste no desenvolvimento dos processos de modelação procedimental directamente na plataforma de armazenamento de dados, tipicamente num sistema de gestão de base de dados, recorrendo para tal à programação de *stored procedures* em linguagens de programação próprias da plataforma, para realizar as várias operações de consulta e processamento dos dados. Uma vez que todos os dados a processar se encontrarão georreferenciados, é necessário que a plataforma possua, não só o suporte para informação espacial, mas também operações para a consulta e manipulação optimizada da mesma.

Visto que o objectivo consiste na modelação de ambientes urbanos reais, é imperativa a consulta de fontes de dados reais. Estes deverão ser importados para a plataforma de armazenamento de dados, podendo ser consultados directamente e rapidamente sem grande *overhead*. Da mesma forma, uma vez iniciado o processo de modelação, os resultados serão gravados na base de dados, tanto em fases finais como intermédias, permitindo assim a realização de consultas mais complexas sobre os mesmos, expandindo assim o leque de possibilidades de modelação a ser realizadas.

A metodologia PG3D vem assim contrastar com a maioria das aplicações de modelação procedimental, cujos processos operam sobretudo em memória RAM, limitando bastante a extensão dos limites da área de modelação em cada instância de execução. Adicionalmente, torna-se complicado (se não mesmo impossível) prever se uma sequência de operações com considerável nível de detalhe não correrá o risco de exceder as capacidades de memória da máquina. Por seu lado, a constante gravação dos dados na base de dados permite eliminar esse risco, dado que apenas pequenas quantidades de dados serão operadas de cada vez em memória, e as restantes constantemente gravadas em disco (que apesar de possuir limitações, são muito incomparavelmente maiores do as da memória RAM).

Contudo, a introdução desta característica vem obviamente com um preço em termos de desempenho, dada a velocidade bastante superior da memória RAM face ao acesso em disco. No entanto, os recentes desenvolvimentos nas tecnologias de bases de dados têm resultado em aumentos bastante significativos no desempenho das mesmas. Assim sendo, apostando-se nas potencialidades de indexação de dados actuais para bases de dados, o PG3D será capaz de conseguir uma performance não muito distante das aplicações de modelação existentes (juntando-se a compensação pelo tempo de acesso reduzido às fontes de dados).

A metodologia PG3D induz assim as seguintes vantagens:

- **Rapidez de acesso às fontes de dados**: Este é possivelmente um dos pontos principais a apontar. Como mencionado, a consulta da informação geográfica referente a ambientes urbanos constitui sempre um *overhead*, que se poderá sentir sobretudo em processos de modelação mais extensos.

- **Maior dimensão dos limites de modelação:** Embora as operações sobre dados apenas em memória RAM sejam consideravelmente mais rápidas do que sobre dados em disco, a memória é também mais limitada, pelo que a gravação constante dos dados em disco possibilita que apenas pequenas parcelas sejam operadas de cada vez.

- **Maior segurança**: Dada a quantidade de informação a ser trabalhada, o tempo para modelação procedimental de uma determinada área poderá atingir tempos consideráveis, de minutos a horas ou dias. Como em qualquer sistema computacional, a máquina a executar o processo é sempre sujeita a falhas que interrompam o processo após um longo período. No caso do PG3D, dado que todas as fases intermédias serão gravadas na base de dados, será possível partir do ponto de interrupção (dependendo, claro das potencialidades de recuperação da base de dados onde foi implementada).

- **Fácil transformação das fontes de dados**: Uma das principais dificuldades referidas é a adaptação da estrutura das fontes de informação de origem diversa para um formato pronto a ser processado pelo modelador. A introdução das mesmas em bases de dados permite que as transformações se dêem sobre a forma de consultas SQL, com o auxílio de funções de índole espacial. Adicionalmente, será possível, caso se pretenda, realizar certas operações de modelação, em vez de as realizar no início de cada instância de modelação.

- **Operações mais complexas através de consultas personalizadas:** A estruturação em bases de dados relacionais permite realizar consultas mais complexas do que em

simples linguagens orientadas a objectos, populares para este tipo de aplicações. Este tipo de questão será vista em mais pormenor mais à frente.

- **Fácil tradução de dados gerados em dados de consulta:** Dado que os dados resultantes da execução de uma instância de modelação também se encontram na base de dados e com referências espaciais, é simples transformá-las em novas fontes de dados para posterior utilização.
- **Fácil integração com qualquer plataforma:** Estando directamente ligada à plataforma de armazenamento de dados, os processos de modelação podem ser facilmente integrados com qualquer cliente ou serviço que procure dispor das suas capacidades de modelação procedimental.

Dado este conjunto de vantagens surge a questão sobre a falta da popularidade deste tipo de metodologia para este tipo de aplicação. É possível assim pensar algumas razões que podem levar muitos a achar esta abordagem pouco atractiva:

- **Complexidade de implementação:** As linguagens de programação integradas em base de dados são bastante menos populares do que as linguagens de mais alto nível, existindo por isso um menor número de recursos, documentação e ferramentas para o seu desenvolvimento. Não sendo normalmente concebidas para fins de processamento tão complexo, estas linguagens possuem funcionalidades mais limitadas, conduzindo à necessidade de produção de um conjunto de instruções mais longo, mesmo para operações mais simples.
- **Dificuldade de integração com sistemas de baixo nível:** No âmbito da computação gráfica, é muitas vezes mais eficiente realizar operações de manipulação tridimensional recorrendo directamente à placa gráfica, que as realiza de forma bastante mais rápida. A integração directa com a plataforma de armazenamento de dados torna esta tarefa extremamente difícil, se não mesmo impossível de efectuar.

3.1.3. Bases de dados Espaciais

A modelação de ambientes urbanos virtuais correspondentes a ambientes reais conduz directamente à necessidade de fontes de informação que descrevam os diversos elementos que os constituem de alguma forma, tal como a sua posição, a sua altura e aspecto. Esse tipo de informação, contudo, dificilmente se encontrará organizada numa única fonte ou num único local e disponível para qualquer utilizador. Por outro lado, a variedade de formatos digitais em que tais dados se poderão encontrar torna o acesso interoperável ainda mais complicado.

Como tal torna-se essencial que todas essas fontes possuam, antes de mais, um referencial comum, para que possam ser estabelecidas relações entre as mesmas e conjugada de forma útil toda a informação nelas contidas. É neste sentido importante a georreferenciação dos dados, ou seja, que os dados possuam a referência da sua localização na superfície da terra.

Um método utilizado para descrever a posição de um objecto na superfície da terra é usando as coordenadas esféricas de latitude e longitude que descrevem os ângulos (em graus) do centro da terra até ao ponto na sua superfície. Este sistema referencial é denominado de sistema de coordenadas geográficas. Os ângulos de latitude são medido nas direcção norte - sul, tendo o

equador o valor ângulo igual 0, e sendo mais vulgarmente atribuído valores positivos ao hemisfério norte e negativos ao sul. As medidas de longitude são realizadas na direcção este-oeste e baseadas tradicionalmente no meridiano de Greenwich (onde o ângulo de longitude é 0). Contudo, apesar de a longitude e a latitude serem capazes de determinar posições exactas na superfície do globo, estas não são unidades uniformes de medida. Tal acontece devido ao facto de apenas o paralelo correspondente ao equador ser tão comprido como os meridianos. Acima e abaixo desta linha, os paralelos vão ficando mais pequenos até se tornarem num único ponto na zona dos pólos. Por essa razão, torna-se complicada a utilização deste sistema de coordenadas em diversas operações de cálculo e análise.

O recurso a sistemas de coordenadas projectadas é uma forma de contornar este problema, tentando representar a superfície da terra de uma forma plana com coordenadas cartesianas, processo este denominado de "projecção". No entanto, dado que a terra possui uma forma elipsoidal, é impossível realizá-lo sem distorções da forma ou criando áreas de descontinuidade. Por outro lado, a natureza irregular do geóide terrestre e as constantes transformações de terreno que ocorrem com o decorrer do tempo, torna-se necessária criar múltiplas projecções sobre áreas mais reduzidas, e datadas para definir o espaço com maior precisão. No entanto, esta diversidade de projecções obriga também ao desenvolvimento de formas de relacionar ou transformar dados que se encontrem definidos em sistemas de projecção diferentes.

A complexidade por detrás da correcta leitura e interpretação dos dados justifica a utilização de uma plataforma de dados que, para além de capaz de os armazenar, possua meios de os relacionar e de realizar operações de análise sobre eles. É neste sentido que é o recurso a bases de dados espaciais.

Enquanto qualquer base de dados é capaz de processar dados numéricos ou textuais, as bases de dados espaciais possuem funcionalidades acrescidas para o processamento de dados espaciais. Estes são tipicamente chamados de geometrias (*geometry*). O consórcio Open Geospacial (OGC) criou a especificação "Simple Features", que define vários tipos de geometria [OGCI06], como se poderá observar na figura seguinte:

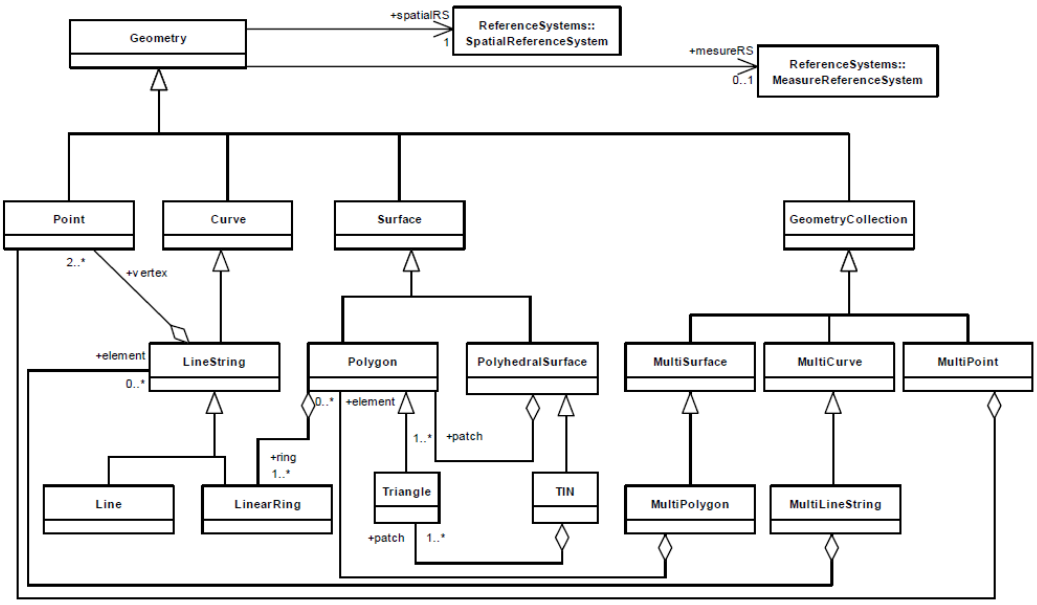

Figura 17 - Especificação Simple Features pelo Open Geospatial Consortium [OGCI06]

Actuando sobre este tipo de dados, as bases de dados espaciais são igualmente capazes de realizar consultas no âmbito espacial, tal como:

- **Construção**: Possibilitando a geração de novas geometrias a partir de descrições textuais, binárias ou da conciliação de valores numéricos;
- **Acesso**: Permitindo a leitura dos valores que integra ou das suas características (número de elementos constituintes, identificador da referência espacial, etc.);
- **Transformação**: Facilitando assim a relação entre dados de projecções geográficas diferentes através da sua transformação para projecções comuns.
- **Edição**: Facultando a alteração dos valores, adição ou remoção de pontos bem como operações de simplificação, segmentação ou transformação geométrica (translação, rotação ou escalamento);
- **Visualização:** Através do retorno da informação em diferentes formatos (binário, textual, GML);
- **Relação:** Permitindo, por um lado, a percepção sobre a intersecção, sobreposição, encerramento ou existência a determinada distância, e por outro a ligação entre geometrias através de uniões, subtracções, etc.;
- **Medida:** Introduzindo facilidades de medida de área, distância, comprimento e perímetro.

Da mesma forma que as bases de dados alfanuméricas podem desfrutar da indexação dos dados para optimizar os processos de pesquisa sobre os mesmos, as bases de dados possuem capacidades de indexação espacial dos dados. Tal é especialmente útil especialmente nas operações de relação espacial acima enunciadas.

As bases de dados espaciais são assim uma solução para o armazenamento, organização, operação e gestão de múltiplas fontes de dados georreferenciados. Dadas as suas adicionais capacidade de edição e transformação dos dados, é possível da mesma forma resolver incompatibilidades de formatos e estruturas de dados.

Por outro lado, as várias faculdades de gestão de geometrias tornam assim este tipo de plataforma a ideal para o Sistema PG3D, uma vez que os seus próprios processos de modelação actuam sobre estruturas de dados semelhantes, podendo dispor por isso da existência das várias operações de gestão, consulta optimizada e manipulação.

3.2. Gramática PG3D

Derivada da sua relação próxima com a base de dados e a sua implementação com linguagens de programação integradas com a mesma, nasce a Gramática PG3D. Esta vem assim definir as estruturas e operações que, por um lado, aproveitam o conceito PG3D, e por outro, expandem o alcance desta metodologia.

As gramáticas formais têm sido alvo de variadas aplicações no ramo da computação gráfica, tendo-se revelado, efectivamente, como imprescindível para a realização de determinadas operações de criação de conteúdos gráficos, especialmente nos mais complexos, extensos e facilmente parametrizáveis. Contudo, dada a variedade de problemas e especificidade de cada um, é necessário o emprego de soluções è medida, e por essa mesma razão têm surgido

diversas abordagens dependendo do tipo, formato e fim dos conteúdos a serem criados. No âmbito da modelação de ambientes urbanos, algumas soluções têm vindo a ser concebidas, tendo-se tornado fortes influências no desenvolvimento da Gramática PG3D.

3.2.1. As Gramáticas Formais no Domínio da Modelação de Ambientes Urbanos

As gramáticas formais são constituídas por conjuntos de regras de produção que actuam sobre cadeias de símbolos no âmbito de uma determinada linguagem. As regras descrevem como é que as cadeias podem ser criadas a partir do alfabeto de símbolos, e de acordo com a sintaxe da linguagem definida. As gramáticas formais caracterizam-se também pela definição de um axioma inicial, de onde parte a aplicação das regras de produção. Esta pode realizar-se em qualquer ordem, até que a cadeia de caracteres deixe de possuir símbolos iniciais ou símbolos não terminais. As gramáticas formais não pretendem definir o significado ou semântica, mas apenas a forma dessas mesmas cadeias [HPCT00]. Em 1956, Noam Chomsky formalizou e classificou algumas das gramáticas mais recorrentemente usadas nos dias de hoje, especialmente no ramo da computação [CSKY56].

Uma das variantes de gramáticas formais que tem ganho mais popularidade no ramo da computação gráfica são os Sistemas L (um acrónimo para Sistemas Lindenmayer, segundo o seu autor, Aristid Lindenmayer), cuja aplicação mais conhecida é na simulação dos processos de crescimento de plantas [PSKZ96] ou no desenvolvimento de comunidades de organismos.

O conceito central dos sistemas L é o de reescrita de cadeias de caracteres. Em geral, trata-se de uma técnica que permite a construção de objectos complexos através da sucessiva substituição de partes de um objecto inicial simples através de um conjunto de regras de reescrita ou produção [PSKZ96]. Estas são aplicadas iterativamente, e em maior número possível, a cada cadeia, partindo de uma inicial, designada de axioma. A cadeia final resultante necessita, posteriormente, de uma interpretação gráfica, na qual cada um (ou cada conjunto) dos símbolos da cadeia corresponde a uma operação de desenho ou transformação gráfica, de forma a obter-se a estrutura da cena num formato mais convencional para poder ser apresentada graficamente. Um dos mecanismos mais populares será a interpretação tartaruga [COEL07, PSKZ96]

Uma das características mais importantes dos sistemas L consiste portanto no facto de as regras de produção serem aplicadas em paralelo, sendo todos os caracteres de uma dada cadeia substituída simultaneamente, ao contrário das gramáticas de Chomsky [CSKY56], em que são aplicadas sequencialmente. Esta diferença reflecte a motivação biológica dos sistemas L, na medida em que as produções permitem a simulação de divisões celulares em organismos multicelulares, em que divisões podem ocorrer ao mesmo tempo.

Com a exploração dos sistemas L e da sua aplicabilidade, foram desenvolvidas extensões ao conceito inicial, sendo possível enumerar diversas classes de sistemas L: sistemas L paramétricos, parentéticos, estocásticos, livres ou sensíveis ao contexto, entre outros [COEL07, PSKZ96]. Cada uma introduz um conjunto mais vasto de oportunidades, e que permitem a melhor aplicação e adaptação a cada problema. Exemplos incluem a simulação dos processos de crescimento de organismos, a sua interacção com o ambiente exterior ou mesmo o desenvolvimento de cada um dos seus elementos constituintes.

De forma análoga à modelação de organismos naturais, alguns autores procuraram utilizar os Sistemas L para modelar ambientes urbanos virtuais, afirmando que estes

compartilham lógicas de desenvolvimento semelhantes. Em [COEL07] é descrito como, tal como as comunidades de organismos, os elementos constituintes de ambientes urbanos (pessoas, vegetação, terreno, ruas, edifícios e outros objectos) se organizam, distribuem e estruturam com base na interacção entre si, ou seja, cada elemento não se desenvolve individualmente, mas sim dependente do seu ambiente em que se encontra. Este conceito de dependência é designado no trabalho de [COEL07] de **"**contextualização geoespacial", aproveitando as potencialidades dos Sistemas L Sensíveis ao Contexto e os Sistemas L Geoespaciais, tendo nomeado a sua conjunção de "Sistemas L Geoespaciais".

Um outro tipo de gramáticas formais recorrentes na área da computação gráfica é a gramática de forma (*shape grammar*), que se distingue sobretudo pelo facto de operar directamente sobre formas geométricas. Criados por George Stiny e James Gips em 1971 [STNY72], as gramáticas de forma têm vindo a ser estudadas especialmente para aplicações de planeamento arquitectónico assistido por computador. Enquanto os sistemas L se baseiam na substituição de caracteres, as gramáticas de forma operam pela substituição de formas (ou *shapes*).

Dos trabalhos que procuraram recorrer às gramáticas de forma para a concepção de ambientes urbanos virtuais, destaca-se [WNKA03] e [MULL06] com resultados de elevado detalhe e fidelidade visual. Actualmente, o sistema *CityEngine*, cuja criação envolveu estes mesmos autores, encontra-se entre as aplicações mais conhecidas no ramo da modelação procedimental para desenvolvimento de ambientes urbanos virtuais com vista para a área de jogos de computador. O conceito de *CGA Shape* nele contido [MULL06] suporta um conjunto vasto de operações de selecção e operação de formas geométricas, sendo estas aplicadas de formas forma sequencial (de forma semelhante às gramáticas de Chomsky [CSKY56]).

3.2.2. Conceitos fundamentais das Gramáticas PG3D

As gramáticas PG3D são um desenvolvimento das gramáticas de forma (*shape grammar*) [STNY72], mais concretamente, a *CGA Shape* [MULL06], dotadas das capacidades de contextualização geoespacial e desenvolvimento relacional, provindas dos Sistemas L Geoespaciais [COEL07]. Tendo sido fortemente influenciadas por ambos, constituem uma tentativa de, dentro do possível, conjugar as suas maiores potencialidades.

Antes de abordar a estrutura das gramáticas PG3D, é necessária a explanação de alguns conceitos básicos que são utilizados na definição das mesmas.

3.2.2.1. A P*G3DShape*

A *PG3DShape*, ou *shape* (forma) para abreviar, constitui a estrutura de dados principal de todos os processos de modelação procedimental da gramática PG3D. É inspirada no conceito de *shape* das gramáticas *CGAShape* definidos em [MULL06], possuindo algumas características em comum com esta.

Cada *PG3DShape* é identificada por um nome, ou símbolo, iniciado por letra maiúscula, não necessariamente único que descreve tipicamente a entidade que pretende personificar ou características que possa possuir. Uma *shape* pode ser assim uma superfície, rua, edifício, ou mesmo apenas uma parcela destas, como uma parede ou o canto de uma janela. Uma *shape* contém um conjunto de geometrias que descreve graficamente o tipo de elemento que pretende representar. Estas geometrias são tipicamente polígonos, que por sua vez contêm vértices, cada

um com uma panóplia de propriedades essenciais para a sua representação (posição, normais, cor, mapeamento de texturas, etc,). A definição da textura é realizada ao nível da *shape* e por transitividade é aplicada a cada um dos seus elementos constituintes.

Figura 18 - Vários Exemplos (com diferentes complexidades) de *PG3DShapes*

Apesar de todas as geometrias já possuírem informação georreferenciada que descreve bem a localização da *shape* no espaço, estas encerram adicionalmente a indicação do seu âmbito, designado de *PG3DScope*, que corresponde a uma *bounding box* de todas as geometrias que possui. Desta forma é possível realizar verificações espaciais (intersecções, uniões) entre *shapes* de forma mais rápida, além de permitir ao utilizador impor condições sobre o desenvolvimento da *shape* com base na sua localização e as suas dimensões. O *PG3DScope* serve igualmente para definir um sistema de eixos próprio, tal como uma origem de coordenadas para facilitar a realização de diversas operações de modelação.

Outra propriedade interessante da *shape* é o seu pivô. É utilizado essencialmente em transformações geométricas, nomeadamente a rotação e o escalamento, para que possam ser realizadas em torno de um ponto arbitrário. As coordenadas do pivô são sempre relativas à origem de coordenadas do *PG3DScope*. Por definição, o pivô encontra-se na origem, mas pode ser redefinido a qualquer altura.

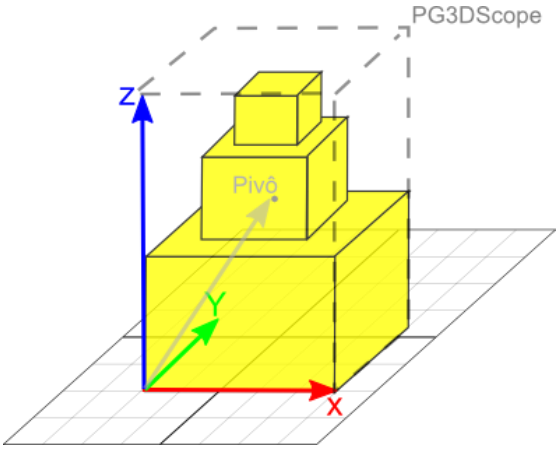

Figura 19 - Representação do PG3DScope e o seu pivô

A *PG3DShape* funciona de forma hierárquica. Como irá ser descrito em seguida, as *shapes* evoluem por substituição sucessiva. Assim sendo, é possível que cada *shape* tenha conhecimento do seu antecessor, ou seja, da *shape* de que deriva. Tal permite realizar consultas sobre *shapes* que descendam de um determinado antecessor.

A PG3DShape é um objecto que se desenvolve em contexto. Como se verá a seguir, o desenvolvimento das *shapes* pode ser condicionado pelo seu ambiente envolvente (contextualização geoespacial). Para que tal seja possível, a informação sobre a sua localização e dimensão encontra-se optimizada de acordo com as potencialidades oferecidas pelas bases de dados espaciais.

3.2.2.2. A *PG3DLayer*

A *PG3DLayer* existe fundamentalmente no âmbito da leitura de uma fonte de dados, como agregadora das *shapes* que dela tenham surgido. Uma *layer* trata-se portanto de uma camada, uma estrutura de dados que engloba uma ou mais *shapes* que possuam a mesma fontes de dados e um significado comum. Trata-se assim de uma forma de organizar as *shapes* criadas ao longo da sua criação e consulta, facilitando a sua gestão. As *PG3DLayers* são igualmente elementos constituintes do ponto de partida de um processo de modelação procedimental, na medida em que nelas são definidas as fontes de dados a serem trabalhadas.

Apesar de serem carregadas independentemente e os seus elementos se poderem igualmente desenvolver de forma autónoma, a intenção principal é que estas possam ser combinadas, conforme a figura:

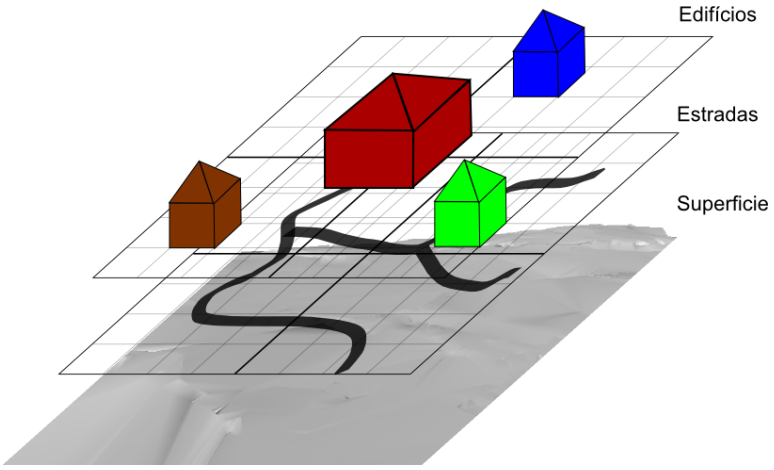

Figura 20 - Inter-relacionamento entre várias *PG3DLayers*

O georreferenciamento das *shapes* contidas nas *layers* permite o seu inter-relacionamento, sendo possível, tal como no exemplo acima, juntar a informação sobre o plano rodoviário e as bases dos edifícios com a informação da superfície, encaixando umas nas outras de uma forma adequada.

3.2.2.3. *A PG3DBoundary*

Em ambientes tão complexos como as áreas urbanas, é comum notar-se discrepâncias em estilos e formatos de construção ao na extensão de uma cidade, sobretudo entre o seu centro e os subúrbios. Por essa mesma razão, torna-se necessário realizar distinções nas instruções de desenvolvimento de cada *shape* conforme a sua localização. Visto que a definição pontual deste tipo de variações é um processo trabalhoso, por vezes é mais fácil (e igualmente correcto) realizá-lo através da definição de um perímetro. Este conceito, denominado de "limite" ou, mais

especificamente, *PG3DBoundary*, faculta assim a aplicação de operações de modelação de forma diferente de uma zona para a outra.

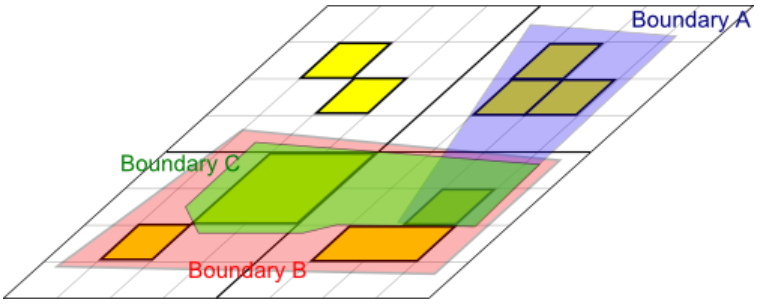

Figura 21 - Representação de *PG3DBoundaries* no espaço

As *boundaries* são definidas em coordenadas georreferenciadas e podem ser mutuamente exclusivas, intersectantes ou mesmo coincidentes, como se pode ver na figura acima. O seu uso não é imperativo, mas pode ser extremamente útil para precisar a selecção de *shapes* para a aplicação de determinadas operações.

3.2.2.4. *A PG3DTag*

Uma das maiores dificuldades que surge da manipulação de uma tão grande quantidade de *shapes*, que podem assumir as mais diversificadas figuras com a sucessiva aplicação de variadas transformações, é a gestão individualizada das propriedades de cada uma. É por isso essencial que existam formas de seleccionar apenas componentes das *shapes* com base nas suas características e realizar operações sobre esses componentes apenas.

Um dessas formas é impondo condições de aplicação com base nas propriedades de cada componente. Como exemplo considere-se uma superfície correspondente à elevação de um terreno: supondo-se que se deseja colorir de forma diferente os vértices que se encontrem abaixo do nível do mar e os vértices acima, é possível aplicar condições na operação de coloração – os vértices que possuam a componente da altura com valores acima de *x* terão a propriedade de cor alterada para a cor A e os com valores abaixo desse limiar terão essa mesma propriedade para outra cor.

Contudo, nem sempre pode ser tão simples seleccionar certas partes de *shapes* através das suas características. É neste sentido que surge a ideia de "etiquetar" componentes na sequência da realização de operações de modelação sobre as mesmas. Estas etiquetas são assim chamadas de *PG3DTags*. Um exemplo do seu emprego pode ser consultado no exemplo que se segue, que demonstra a aplicação de *tags* numa operação de extrusão:

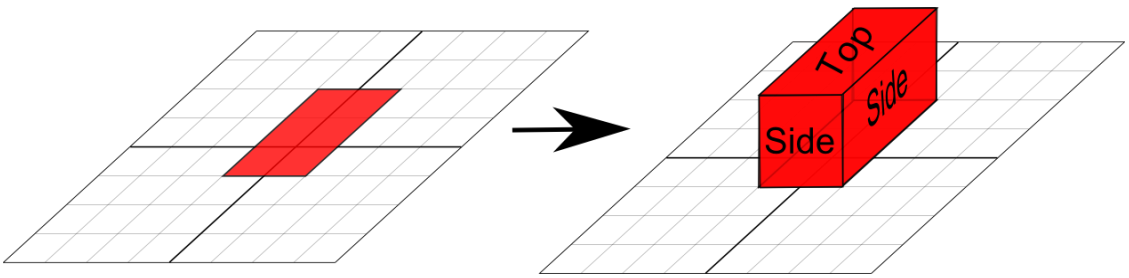

Figura 22 - Exemplo da aplicação automática de *PG3Dtags* a uma *shape* após a sua extrusão

Como se pode ver, cada face gerada neste processo é etiquetada, facilitando assim a sua selecção para a aplicação de operações posteriores. É possível que cada polígono contenha mais do que uma *tag*, de modo a tornar as selecções mais precisas.

Da mesma forma que são aplicadas, as *tags* podem ser removidas conforme desejado, caso se pretenda evitar conflitos e confusões, algo que pode acontecer após um elevado conjunto de operações de modelação.

3.2.3. Definição de Gramática PG3D

Como enunciado previamente, a criação das gramáticas PG3D foi fortemente influenciada pelas gramáticas *CGA Shape* [MULL06] e pelos Sistemas L Geoespaciais [COEL07]. Apesar de ser sobretudo uma extensão das *CGA Shape*, conta também com algumas particularidades dos Sistemas L Geoespaciais, tais como o desenvolvimento das regras em paralelo e, em especial, o conceito de contextualização geoespacial.

A gramática pode ser então descrita pela sua constituição nos seguintes elementos:

- Um conjunto de *PG3DShapes*, que agrega um ou mais elementos com representação gráfica. As *PG3DShapes* podem ser consideradas terminais, caso não possuam regras que se apliquem a si, ou variáveis, caso contrário.
- O axioma, ou seja o elemento de partida dos processos de modelação, definido por uma ou mais *PG3DLayers*. Cada *layer*, correspondente a uma fonte de informação pode conter agregar também múltiplas *PG3DShapes*.
- Um conjunto de regras de produção, denominadas *PG3DRules*, que definem as instruções de modelação, mais concretamente os processos de substituição e desenvolvimentos das *PG3DShapes*.

O processo consiste, em suma, da substituição sucessiva de *shapes* de uma forma iterativa, partindo do axioma definido, e seguindo as instruções contidas nas regras de produção. Na primeira iteração do processo, são analisadas as *layers* que fazem parte do axioma, que indicam quais as fontes de dados que deverão ser carregadas, e em que limites. Embora seja possível carregar toda a informação nelas contidas, é muitas vezes mais conveniente trabalhar em pequenas parcelas de cada vez. A definição de tais parcelas pode ser realizada através de delimitações do espaço definida por *PG3DBoundaries*.

A partir daí, possuindo já um conjunto de *shapes* com o potencial para serem desenvolvidas, o procedimento continua assim de uma forma simultaneamente sequencial e paralela. Em cada iteração são abordadas todas as *shapes* que tenham sido criadas na iteração anterior, procurada a regra de produção que se adapta a cada uma, que depois é aplicada. Estando esta operação terminada, passa-se à iteração seguinte onde se tratarão as *shapes* recentemente criadas na iteração anterior. Este processo repete-se até não existirem mais regras aplicáveis a nenhuma *shape* nessa iteração ou até ter sido atingido o limite de iterações imposto pelo utilizador. Trata-se assim de um processo de desenvolvimento em **paralelo**, na medida em que todos os elementos são substituídos de uma vez em cada iteração. No entanto, as regras de produção podem ser realizadas de uma forma composta, na medida em que podem ser conjugadas múltiplas transformações e substituições de *shapes*. Isto significa, que de uma forma interna às regras de produção, o seu processo realização de forma **sequencial.**

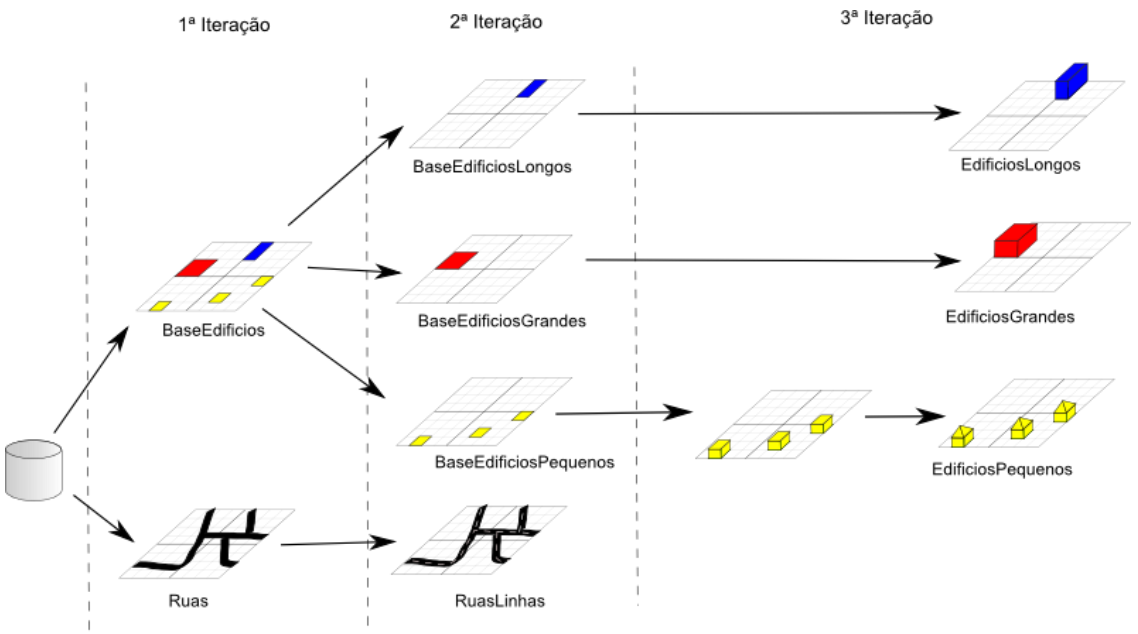

Figura 23 - Evolução em árvore da aplicação de regras em várias iterações

A figura acima ilustra o procedimento sobre um axioma constituída por duas *layers* correspondentes aos edifícios e às estradas. Como se pode observar, na primeira iteração é realizado o carregamento dos dados das fontes de dados, sendo criadas duas *shapes* com grandes dimensões, uma contendo todas as bases dos edifícios e outra contendo as formas das ruas. A segunda iteração consiste, no caso das *shapes* pertencendo à *layer* de edifícios, da divisão em mais pequenas, tendo cada uma apenas as bases de edifícios que correspondam a uma certa condição. Neste caso foram divididas em bases de grande ou pequena área ou de comprimento maior. No caso da *shape* pertencendo à *layer* de estradas, estas continuaram agregadas na mesma *shape*, tendo no entanto sido adicionadas as linhas rodoviárias identificativas das faixas de rodagem. Como se pode verificar, estas transformações deram-se de forma paralela e em simultâneo.

Na segunda iteração foi realizada uma operação de extrusão nos 3 tipos de edifícios, concebendo volumes sólidos, tendo e na terceira *shape* sido também executada uma operação de criação de telhado sobre cada um dos edifícios, dentro da mesma operação. Torna-se assim uma operação composta (que poderia ter mais operações) com aplicações sequenciais. Chama-se à atenção que seria igualmente possível ter realizado as operações da segunda e terceira iterações numa só, através da composição de operações e de estruturas de controlo apropriadas dentro das regras de produção.

A figura representa apenas um quadro possível de modelação procedimental. Contudo, é possível já visualizar alguns aspectos fundamentais do seu desenvolvimento e que conferem um enorme potencial na extracção dos dados criados. Como se poderá reparar, o processo criou uma árvore de geração, com cada nó correspondendo a uma *shape* representável graficamente. Dada a sua adicional identificação dos ramos da árvore através de *layers*, da aplicação de nomes nos seus nós e à numeração do grau de profundidade com o número da iteração, é possível executar consultas mais complexas sobre os dados.

Por exemplo, se se pretender obter o resultado final do processo de modelação, será necessário apenas obter as folhas da árvore (nós sem descendente) o que, no caso da figura, corresponde às *shapes* "RuasLinhas", "EdificiosLongos", "EdificiosGrandes" e "EdificiosPequenos". Da mesma forma, é possível obter os elementos criados até uma

determinada iteração, aos pertencentes apenas a uma *layer* ou mesmo aos descendentes de um determinado nó. Este tipo de potencial confere uma maior flexibilidade à gestão dos dados criados, algo que poderá mostrar-se especialmente útil se pretende observar o desenvolvimento da modelação, e assim compreender, por exemplo, as razões por detrás de desvios que possam ocorrer na evolução de uma *shape*.

3.2.4. Regras de Produção

A ideia básica de uma regra de produção consiste na substituição de uma *shape* por um conjunto de uma ou mais *shapes*. Formalmente, as regras estruturam-se da seguinte forma:

Predecessor → Sucessor

Na sua forma mais básica, a definição do predecessor consiste na indicação de um símbolo, composto por caracteres alfanuméricos, e começando com letra maiúscula. Esta regra será a partir daí aplicável a todas as *shapes* que possuam um símbolo coincidente.

O sucessor, por seu lado, pode ser bastante mais complexo, contendo uma sequência de operações de modelação e terminando tipicamente num novo símbolo que será o nome da nova *shape* criada. Esta será o resultado da aplicação das operações sobre o predecessor e irá substitui-lo.

Para uma compreensão mais fácil, considere-se o seguinte exemplo:

Shape1 → Shape2

Esta regra de produção constitui o tipo mais simples de substituição de *shapes*, tratando-se de uma simples alteração de nome. Para cada *shape* cujo nome seja "Shape1" será criada uma nova designada de "Shape2", que mantém as mesmas características que a "Shape1" exceptuando o seu nome.

Observe-se um exemplo mais complexo:

Shape1 → translate(vector3(0,10,20)) Shape2

Neste novo caso, é realizada uma transformação geométrica de translação. A "Shape2" será assim o produto da aplicação da operação de translação sobre a "Shape1". Da mesma forma, pode-se encadear mais do que uma operação de modelação:

Shape1 → translate(vector3(0,10,20)) scale(vector3(5,5,5))
colorShape(rgba(255,0,0,255) Shape2

Neste caso mais complexo, é aplicada uma translação, depois um escalamento, e por fim uma cor (vermelha neste caso). A "Shape2" é assim produto de um conjunto de operações complexa sobre a "Shape1", e por isso encerra uma menor semelhança com ele.

A definição desta última regra implica a criação de uma *shape* com novo nome no final da aplicação de todas as operações de modelação. A seguir a cada transformação será criada uma versão temporária e anónima da "Shape1". No entanto, apenas na altura da definição do

símbolo de uma *shape* é que o resultado até aí será "nomeado" e poderá ser utilizado em iterações seguintes. Assim sendo, o seguinte exemplo:

Shape1 → translate(vector3(0,10,20)) scale(vector3(5,5,5))
colorShape(rgba(255,0,0,255)

não resultará na definição de qualquer nova *shape*, viste que não existe nenhuma indicação sobre *shape* que deverá ser criada. Para distinguir as definições de uma *shape* de uma operação, a primeira é sempre iniciada com letra maiúscula, enquanto a segunda começará sempre por uma letra minúscula. Esta definição será extremamente importante na secção seguinte, para realizar distinções para com regras paramétricas.

É possível da mesma forma criar múltiplas *shapes* em cada regra através da nomeação sucessiva, como no exemplo:

Shape1 → translate(vector3(0,10,20)) Shape2 scale(vector3(5,5,5)) Shape3
colorShape(rgba(255,0,0,255) Shape4 extrude(20)

Segundo esta regra, a "Shape1" será substituída por um conjunto de novas *shapes:* a "Shape2", que é igual ao predecessor mas deslocada no espaço, a "Shape3" que é 5 vezes maior que a anterior (em todas as coordenadas) e "Shape4" que é igual à anterior, mas com cor vermelha. A última operação, de extrusão, não sendo seguida da determinação de uma *shape*, não tem qualquer seguimento.

3.2.4.1. Regras Paramétricas

Da aplicação de uma regra para a seguinte pode ser necessário transmitir determinadas informações. Por essa razão, as regras de produção aceitam a definição de parâmetros. De uma forma semelhante à especificação de uma função, estes deverão ser declarados no predecessor da regra, a seguir ao símbolo:

Shape1(xValue float, zValue float)→ translate(vector3(xValue,0, zValue)) Shape2

Os argumentos definidos na regra poderão ser depois utilizados múltiplas vezes ao longo da especificação do sucessor.

No processo iterativo de aplicação de regras, o número e tipo de argumentos é igualmente consultado, permitindo assim a definição de regras com o mesmo símbolo inicial, desde que variem nestes pontos. Trata-se assim de *overloading* de regras, de forma análoga à definição de funções em determinadas linguagens de programação.

Os argumentos que são passados são alojados nas próprias *shapes* a que as regras se aplicam. A sua passagem consiste em definir, na componente sucessora da regra, valores a seguir à nomeação da nova *shape*. Considere-se o seguinte exemplo que o demonstra, juntamente com a capacidade de *overloading* de regras discutida:

Shape1→ translate(vector3(10,40, 50)) Shape2(vector3(10,40, 50))) extrude(50.0) Shape2(50)

Shape2(vValue vector3)→ scale(vValue) Shape3

Shape2(height float)→ scale(height,0, height) Shape4

Note-se que esta abordagem permite também realizar, de certa forma, estruturas de controlo sobre o desenvolvimento das *shapes*.

3.2.4.2. Regras Condicionais

O estabelecimento de condições é essencial para o desenvolvimento de *shapes*, especialmente quando carregadas de fontes de informação exteriores ou quando submetidas a processos estocásticos. As gramáticas PG3D incluem assim suporte para estruturas de controlo, que funcionam de forma muito semelhante a certas linguagens de programação.

Shape1(arg integer)→ IF arg > 30 THEN translate(vector3(10,40, 50)) Shape2 ENDIF;

Este exemplo demonstra a forma mais simples de construção condicional. É possível igualmente definir mais alternativas ou mesmo uma condição adicional, caso nenhuma das condições anteriores se cumprir.

```
Shape1(arg integer) → IF arg < 0 THEN
                          translate(vector3(10,10, 10)) Shape2
                      ELSIF arg < 30 THEN
                          rotate(90, 'X') Shape2
                      ELSIF arg < 50 THEN
                          colorShape(rgba(0,0,255,100)) Shape3
                      ELSE
                          extrude(40) Shape4
                      ENDIF
```

É possível encaixar múltiplas definições IF…THEN…ELSE umas nas outras sem limite, de forma análoga a certas linguagens de programação.

Embora até agora se tenha mostrado condições simples, é possível definir expressões booleanas mais complexas, recorrendo para tal aos operadores 'and', 'or', 'not'. Mais pormenores sobre o desenvolvimento condicional serão vistos mais adiante.

3.2.4.3. Regras Estocásticas

O recurso à aleatoriedade torna-se, quando controlada e utilizada eficazmente, uma forma de amplificação de dados. Dado que muitas vezes é difícil, se não mesmo impossível, encontrar dados que descrevam objectos existentes de forma completa, torna-se necessário definir características que, dentro de determinados parâmetros, colmatem essa falha. Um exemplo desse tipo de parâmetros poderá ser a frequência conhecida sobre a ocorrência de uma determinada característica num certo meio, o que propõe o recurso de probabilidades. É nesse sentido a estrutura STOC…ENDSTOC traz grandes potencialidades, como se poderá constatar no exemplo que se segue:

```
Shape1→ STOC 20% extrude(20) Shape2 50% extrude(40) Shape3 30% extrude(60) Shape4
ENDSTOC
```

Na aplicação desta regra, existe 20% de probabilidades de a *shape* criada ser extrudada em 20 unidades, 50% de ser extrudida em 40 e 30% de ser extrudada em 60 unidades.

É possível, da mesma forma que na declaração de estruturas condicionais, encaixar múltiplas definições deste tipo, tal como no exemplo seguinte:

```
Shape1→ STOC 20% extrude(20)
            STOC
                  33.3% color(rgba(255,0,0)) Shape2
                  33.3% color(rgba(0,255,0)) Shape2
                  33.3% color(rgba(0,0,255)) Shape2
            ENDSTOC
            50% extrude(40) Shape3
            30% extrude(60) Shape4
      ENDSTOC
```

3.2.4.4. Regras Parentéticas

As possibilidades de encadeamento de operações de modelação implicam igualmente a necessidade de as conseguir gerir de forma útil. Suponha-se o seguinte caso:

```
Shape1 → translate(vector3(0,10,20)) Shape2 scale(vector3(5,5,5)) Shape3
            colorShape(rgba(255,0,0,255) Shape4
```

Neste exemplo, são geradas 3 novas *shapes* a partir da "Shape1". A "Shape2" corresponde a uma simples reprodução da "Shape1", deslocada 10 unidades em Y e 20 em Z, a "Shape3" corresponde às mesmas características que a "Shape2", mas com um tamanho 5 vezes maior e por fim a "Shape4" é igual à *shape* anterior, mas dotada de uma cor vermelha. Como se pode ver, cada *shape* herda assim todas as alterações efectuadas pela sequência de operações de modelação definida. Contudo, tal poderá nem sempre ser o pretendido. Supondo-se que se pretende, no caso acima, que a "Shape4" tenha a mesma dimensão que a "Shape2". É necessário assim anular o efeito do escalamento realizado entre as duas definições. Tal pode ser realizado através de uma operação de escalamento com valores simétricos ou simplesmente gravando o estado antes do escalamento e recuperando esse mesmo estado a seguir à definição da "Shape3". Com esse objectivo existem os operadores PUSH e POP que colocam numa pilha (*stack*) o estado actual e permitem a sua recuperação posteriormente. A sua utilização para solucionar o caso referido seria:

```
Shape1 → translate(vector3(0,10,20)) Shape2 PUSH scale(vector3(5,5,5)) Shape3 POP
            colorShape(rgba(255,0,0,255) Shape4
```

Desta forma, a "Shape4" não será escalada.

A denominação de "parentético" deriva do conceito aplicado aos Sistemas L que possuem o mesmo objectivo. Nesses casos são utilizados os operadores "[" e "]" para esse efeito, mas que nesta gramática, para evitar conflitos com a definição de *arrays*, não é utilizado.

3.2.4.5. Atributos, Limites e Texturas

A utilização dos mesmos valores em mais do que uma regra de produção é uma prática comum. De uma forma análoga a linguagens de programação, é conveniente a declaração destes valores num local à parte, podendo depois ser referenciado mais do que uma vez. Assim, as gramáticas PG3D suportam a seguinte definição de atributos:

```
valorAltura := 30;
eixoRotacao := 'X';
corAmarela := rgba(255, 205, 0);
```

A declaração de atributos não requer a definição do seu tipo, uma vez que a sua definição é normalmente explícita. Uma enumeração destes tipos será vista numa secção mais à frente. A sua utilização nas regras de produção realiza-se através do recurso ao seu nome, precedido por uma arroba ('@'), como exemplificado em seguida:

```
Shape1→translate(vector3(0,10,@valorAltura)) rotate(90,@eixoRotacao)
colorShape(@corAmarela) Shape2
```

Para os casos vistos acima, a declaração de variáveis é bastante simples. No entanto, para outras estruturas de dados, tais como a definição de limites (*PG3DBoundaries*) e de texturas (*PG3DTextures*) que requerem a especificação de múltiplos pontos e de caminhos para ficheiros, respectivamente, é por vezes mais conveniente que estes sejam definidos de forma mais interactiva (através de qualquer cliente possua essa capacidade) e gravados na base de dados para posterior consulta. A sua referência poderá ser realizada da seguinte forma:

```
Shape1→ setTexture(@t_texturaChao)) createSkybox(@b_cidadePorto) Shape2
```

Para estes casos, o indicador 't_' ou 'b_' a seguir à arroba declaram o tipo de dados de que se trata. Mais alternativas poderão ser definidas consoante a implementação e potencialidades da aplicação cliente.

3.2.5. Transformação, Leitura e Acesso aos Dados Reais

O objectivo de modelação de ambientes urbanos virtuais baseados em ambientes reais moldou a estrutura das gramáticas PG3D. A necessidade de os processos de modelação se basearem em dados reais tornou assim imperativa a criação de métodos para a sua simples integração.

O primeiro passo por detrás da utilização de fontes de informação geográficas consiste na compreensão do seu formato e capacidade de aproveitamento. No entanto, serão certamente raros os casos em que estes poderão ser imediatamente usáveis sem recurso a transformações prévias. Neste sentido, a sua implementação sobre bases de dados é extremamente vantajosa, na medida em que, através da execução de funções de cariz variado (operadores aritméticos, de manipulação de geometria ou mesmo de modelação) em consultas (por exemplo, em SQL ou na linguagem usada) se poderá criar novas fontes com formatos usáveis.

Como referido, as instruções sobre as fontes de informação a carregar é realizada nas *PG3DLayers*, que depois agregam os dados em uma ou mais *shapes*. Cada polígono destas *shapes* iniciais corresponde a um registo da fonte de dados.

No processo de leitura dos dados, apenas dois campos são carregados: a chave primária, que identifica o registo de forma única e a geometria. Estes campos constituem os elementos chave e fundamentais que deverão estar constar em qualquer tabela a ser trabalhada pelo modelador PG3D. A sua existência permite assim, por um lado, operar sempre sobre geometrias (que possuem obrigatoriamente informação geográfica) e, por outro, não deixar de possuir uma referência para o registo lido, para que seja possível aceder às outras colunas da mesma linha.

A razão por detrás do carregamento de apenas 2 campos consiste na poupança de recursos. É possível através dos dois campos carregados aceder ao registo completo da base de dados a que o polígono da *shape* se aplica, eliminando assim a necessidade de conter o registo completo. Para que não existam polígonos sem informação associada, a referência do registo é propagada sempre que sejam efectuadas operações de modelação sobre polígonos. Um exemplo poderá ser na realização de extrusões: ao ter uma relação de derivação para com as geometrias originais, os novos polígonos criados receberão igualmente a referência ao registo.

Estando criadas facilidades de preparação e acesso aos dados, tudo depende da forma como as regras de produção forem definidas e da forma como os dados se encontrarem.

3.2.6. Desenvolvimento Característico e Condicional

Uma das principais características das gramáticas PG3D é a sua capacidade de desenvolvimento característico e condicional baseado nas propriedades que cada *shape* ou mesmo cada elemento constituinte seu (os seus polígonos ou vértices) possa possuir. Por outras palavras, a evolução de uma *shape* poderá depender do seu estado actual ou de dados que possam provir do carregamento dos dados das fontes de informação.

No capítulo anterior foi descrita a estrutura de controlo IF…THEN…ELSE que constitui uma forma de verificação destas propriedades. A outra existe ao nível das operações, na medida em que certas incluem um parâmetro booleano, permitindo que sejam aplicadas apenas a partes de uma *shape* condigam com uma certa condição.

Para indicar que se pretende actuar com base no estado da *shape*, no dos seus polígonos ou vértices são utilizadas as seguintes referências:

%s – Para indicar a alusão à *shape* no seu estado actual

%p – Para se referir aos polígonos dentro da *shape* actual

%v – Para se referir aos vértices dentro da *shape* actual

Tratam-se assim de apontadores, podendo os seus campos ser consultados através de funções especiais, tais como posição, dimensão, cor, textura, normais, etc. É possível igualmente consultar dados que provenham do carregamento dos dados.

Considere-se o seguinte exemplo simples da condição baseada nas propriedades de *shapes*:

```
Shape1 →IF height(scope(%s)) > 30 THEN scale(vector3(0,0,30 / height(scope(%s))))
Shape2 ENDIF;
```

Na regra acima, pretende-se garantir que a altura de uma *shape* não seja maior do que 30 unidades. Essa propriedade é consultada e caso a condição não se realize, é realizado um escalamento em Z para reduzir essa altura para 30 unidades. Este caso transmite bem a ideia do poder do acesso aos dados. Permite por um lado consultar se um valor está dentro do exigido e depois agir correspondentemente. O valor da altura é assim usado tanto para verificações como acções, nomeadamente em argumentos das operações de modelação.

Considere-se agora um exemplo diferente, mas desta vez incluindo dados dos polígonos e vértices:

Embora este exemplo faça pouco sentido em termos práticos (pois mistura valores do pivot com cores), apresenta bastante bem o poder por detrás da consulta das propriedades e o seu uso nos vários argumentos das operações de modelação.

```
Shape1→ IF width(scope(%s)) >10 THEN
            colorPolygon (rgba(100,recd(%p,'bluecolor'), 10 + x(pivot(%s)), 255),
                y(pivot(%s) ) = 20 or top(normal(scope(%p)))) Shape2
    ELSE
        colorVertex(rgba(100,100,0,255), height(scope(%p)) > 10 and
                top(normal(%v))) Shape3
    ENDIF
```

De facto, a operação de coloração é um dos raros casos ponderados em que é possível estabelecer condições sobre tanto as propriedades dos polígonos como dos vértices. A regra indica que caso a largura da s*hape* exceda 10 unidades, então será realizada a coloração a nível dos polígonos. Esta operação recebe como argumentos a cor, e uma condição, indicando que esta só será aplicada aos polígonos para os quais a condição for verdadeira. Os componentes da cor são definidos por constantes, por acesso ao campo 'bluecolor' do registo respectivo do polígono (ver secção 3.2.5.) e do valor da coordenada x do pivô da *shape*. Por seu lado, a condição aplica-se sobre os dados da *shape* e sobre os do polígono (o conceito de "top" baseia-se na orientação do eixo z do âmbito do polígono).

Antes de se proceder à conclusão sobre as potencialidades, passe-se à segunda expressão do exemplo, que aplica a cor a vértices que possuam determinadas características. Para este caso já faz sentido a utilização da referência a %v, acedendo aos dados da sua normal. Adicionalmente, apesar de a função se dirigir aos vértices, é possível igualmente recorrer aos dados do polígono a que pertencem (*height(scope(%p)) > 10*). Existe portanto uma sequência hierárquica – uma função ou operador que aceite a referência a %v aceita também a %p e %s. O mesmo se aplica de das que aceitem %p relativamente a %s.

Em suma, a utilização da referência %s pode surgir nas em condições em estruturas de controlo IF…THEN…ELSE e nos argumentos ou condições dentro das operações de modelação. Por outro lado, %p e %v só podem ser utilizadas nos argumentos e condições das operações que lhes digam respeito, tal como nos casos acima.

Para concluir, pretende-se apenas chamar a atenção para a constituição das condições. Como se pode observar, estas podem ser compostas, utilizando para tal os operadores booleanos

‘and’, ‘or’, ‘not’ ou mesmo parêntesis, de forma semelhante a várias linguagens de programação, permitindo assim a formulação de condições complexas.

3.2.6.1. Contextualização Geoespacial

O conceito de contextualização geoespacial nas gramáticas PG3D deriva da sua aplicação em Sistemas L Geoespaciais [COEL07]. A ideia base consiste em que o desenvolvimento dos processos de modelação não ocorra de forma independente em *shapes* diferentes, mas que seja influenciada pela relação entre elas.

Suponha-se a construção de edifícios cujas bases se encontram juntas. Um dos primeiros passos seria a extrusão da base, obtendo-se paredes com faces opostas mas coincidentes em posição. O passo seguinte seria por exemplo a criação de janelas. Contudo, a construção de janelas em paredes tapadas por outros edifícios não pode ser considerada uma construção válida. Neste sentido, é imperativo que se possa realizar consultas sobre as *shapes* que se encontrem na mesma vizinhança. Este é um dos casos em que o desenvolvimento das *shapes* se deve realizar “em contexto”, e não independentemente, sobre o risco de modelação de estruturas irrealistas.

A integração de contextualização geoespacial dá-se no âmbito do desenvolvimento condicional, através da chamada de funções de consulta sobre os dados geoespaciais das *shapes* e polígonos. Observe-se o seguinte exemplo, baseado no caso mencionado anteriormente.

```
QuadranteParede →IF NOT hasInFront (%s,’Parede’,1) THEN
QuadranteBomParaJanela ENDIF;
```

Esta regra aplica-se a um quadrante de uma parede, no qual se pretende abrir uma janela. Para realizar a verificação sobre se é possível, é averiguado se existe alguma parede a menos de um metro de distância, e em caso negativo, será criada uma nova *shape* que irá realmente realizá-lo (“QuadranteBomParaJanela”).

Tal com a função que constata se existem elementos em frente, existirão outros que verificarão na vizinhança, acima ou abaixo da *shape*. Uma vez que elas se encontram georreferenciadas (podendo ser os dados referentes à sua localização indexados em bases de dados espaciais) trata-se de uma operação simples de construir.

O facto de as *shapes* se desenvolverem em paralelo contribui para que seja mais simples acompanhar o estado de cada uma em cada iteração. Por exemplo, para o caso anterior, se para todas as *shapes* tiver sido aplicada uma operação de extrusão, criando a suas paredes, seguida de uma divisão em quadrantes, é certo supor que as paredes já terão sido criadas na altura em que está a analisar a criação de quadrantes (pois a criação de paredes já foi executada para todas as bases numa fase anterior).

Considere-se o seguinte caso, em que se possuírem duas *layers* correspondentes à superfície do terreno e edifícios:

```
Superficie → setTexture(@t_stone) SuperficieComTextura;
SuperficieComTextura → surfaceOptimize() SuperficieComTexturaOptimizada;
Edificio → placeOnSurfaceShape(‘SuperficieComTexturaOptimizada’)
EdificioEmSuperficie;
```

Este exemplo demonstra outra forma de interacção entre *shapes* e recurso à contextualização geoespacial, na medida em que os edifícios são colocados sobre outra *shape*, mais concretamente de superfície. No entanto, o caso acima contém um problema relacionado com o desenvolvimento em paralelo. Suponha-se que na primeira iteração, o axioma é carregado e processado, criando uma *shape* chamada "Superficie" e um conjunto de *shapes* chamadas "Edificio". Na segunda iteração, será aplicada a 1ª regra à "Superficie", sendo criada uma nova designada "SuperficieComTextura", que só será analisada pela 2ª regra na iteração seguinte. De seguida, será aplicada a 3ª regra às *shapes* de nome "Edificio". No entanto, a sua execução possui um problema. Esta depende da *shape* "SuperficieComTextura" que na realidade ainda não foi criada nesta iteração (só será na seguinte). Este problema pode ser resolvido de três formas:

- **Junção das duas primeiras regras**, recorrendo assim ao funcionamento em sequencia:

```
Superficie → setTexture(@t_stone) surfaceOptimize()
        SuperficieComTexturaOptimizada;
Edificio → placeOnSurfaceShape('SuperficieComTexturaOptimizada')
        EdificioEmSuperficie;
```

Dado que a aplicação das regras é coincidente com a ordem da sua definição, não existe o risco de a primeira regra ser aplicada antes da segunda.

- **Verificar a existência da *shape* em falta, através da função 'existsAny':**

```
Edificio → IF existsAny('SuperficieComTexturaOptimizada') THEN
placeOnSurfaceShape('SuperficieComTexturaOptimizada') EdificioEmSuperficie
        ELSE Edificio;
```

Esta função verifica se existe alguma *shape* com o nome pretendido. Se não existir, cria uma *shape* com o mesmo nome, garantindo assim a sua revisão na iteração seguinte.

- **Controlar o número da iteração,** através da função 'iterationLevel', aplicada à *shape* actual:

```
Edificio → IF iterationLevel(%s) > 2 THEN
                placeOnSurfaceShape('SuperficieComTexturaOptimizada')
                EdificioEmSuperficie
        ELSE Edificio;
```

Esta função verifica se a *shape* está a ser analisada na altura pretendida. Caso contrário, cria uma *shape* com o mesmo nome, garantindo assim a sua revisão na iteração seguinte. Este método requer no entanto um bom controlo e certeza sobre o número da iteração em que quer que a *shape* seja revista.

Como se pode ver, as gramáticas PG3D possuem uma vasta variedade de soluções de modelação, permitindo um bom domínio sobre as operações a realizar. Tal é apenas possível devido às suas características de desenvolvimento condicional e característico.

3.2.6.2. Selectividade

Uma forma simplificada de actuar sobre *shapes* consiste na sua decomposição em *shapes* de menor dimensão e complexidade, e manipulando-as na iteração seguinte. Para permitir esse tipo de acção, o PG3D contém os operadores de selectividade, também designados de "selectores". O seu modo de actuação é semelhante a uma simples "nomeação" de *shape*, mas permitindo a aplicação de condições sobre polígonos. Suponha-se o seguinte caso, em que a "Shape1" consiste num conjunto de polígonos:

```
Shape1→ rotate(90,'Y')  { maxZ(scope(%p)) < 10 : Shape2 }
```

Neste caso é aplicada uma rotação de 90 graus à "Shape1" e tipicamente todo o resultado da sua transformação poderia ser guardado numa *shape*. No exemplo, contudo, é consultado o âmbito de cada um dos polígonos, bem como o ponto com coordenada Z mais elevado. Apenas os polígonos que possuam esse valor abaixo de 10 unidades farão parte da nova *shape*, designada de "Shape2".

Analise-se agora a seguinte regra, aplicável ao caso apresentado anteriormente na Figura 23, mais especificamente na segunda iteração:

```
BaseEdificios → { ySize(scope(%p)) >= 3: BaseEdificiosLongos,
                  area(%p) >= 4: BaseEdificiosGrandes,
                  %rest: BaseEdificiosPequenos}
```

Esta regra limita-se a dividir a *shape* em múltiplas parcelas, cada uma agregando o conjunto de polígonos cujas propriedades correspondam à condição. As bases dos edifícios com dimensão Y maior que 3 foram todas agregadas numa *shape* chamada "BaseEdificiosLongos", os de área igual ou superior a 4 foram colocados numa *shape* designada "BaseEdificiosGrandes" e todos os polígonos que não foram incluídos em nenhum dos grupos anteriores foram colocados numa *shape* denominada de "BaseEdificosPequenos". Esta palavra-chave 'rest' serve assim para garantir que não ficam partes da *shape* perdidas.

Os conjuntos de polígonos abarcados pelas condições podem ser mutuamente exclusivos, intersectantes ou coincidentes (no exemplo da figura, não é feita tal representação, mas é possível).

Os selectores possuem outra palavra-chave, 'each'. Esta é especialmente útil se se pretender separar cada um dos elementos constituintes da *shape* em várias individuais, como no exemplo:

```
TodasBasesEdificios → { true %each : BaseEdificioIndividual}
```

O valor 'true' indica que se aplica para todas as regras, e 'each' indica que vai atribuir a cada polígono da *shape* "TodasBasesEdificios" uma *shape* própria.

3.2.7. Utilização das geometrias espaciais

Para além da sua presença nas fontes de dados, nas quais são usadas para descrever um ponto, linha ou forma, as geometrias espaciais são utilizadas em várias outras ocasiões na gramática PG3D.

Como já foi mencionado, no processo de leitura das fontes de dados espaciais é carregado o identificador de cada registo, juntamente com a geometria a ele associada. Esta geometria é importada para uma *shape*, e convertida num seu elemento constituinte, nomeadamente um polígono ou vértice. Nesta conversão são criadas estruturas de dados PG3D mais fáceis de manipular, igualmente com referências geoespaciais, mas isentas das capacidades de indexação espacial optimizada que a base de dados espacial possa oferecer.

Por essa razão, cada *shape*, polígono e vértice possui na sua definição uma geometria de especificação "Simple Features" [OGCI06] correspondente, respectivamente *multipolygon*, *polygon* e *point*. Estas são geradas após cada modificação efectuada sobre as estruturas PG3D, para que permaneçam actualizadas. É assim importante a fácil transformação das estruturas PG3D para as geometrias. Partindo destas é fácil e rápida a consulta das relações espaciais entre *shapes* no seu todo, ou mais precisamente ao nível dos seus elementos constituintes, possibilitando assim as fortes capacidades de contextualização geoespacial referidas na secção anterior.

Dadas as várias operações de análise e transformação espacial que as bases de dados espaciais incluem, por vezes torna-se mais compensador operar sobre as geometrias espaciais do que sobre as estruturas PG3D, pois além de pouparem a necessidade da sua reimplementação, são por vezes bastante mais eficientes.

A coexistência das geometrias da base de dados espacial com as estruturas PG3D torna-se, de certo modo, numa duplicação de informação e, consequentemente, do espaço necessário. Contudo, dado o tempo de conversão razoavelmente elevado entre as duas, esta abordagem é mais vantajosa, já que as consultas podem incorrer com maior frequência e intensidade (ou seja, abrangendo grandes áreas) do que as operações de modelação.

3.2.8. Operações de Modelação da Gramática PG3D

Para permitir a modelação dos vários componentes que constituem os ambientes urbanos, torna-se necessária a implementação de uma panóplia de operações geométricas suficientemente ampla para os conseguir representar da forma mais próxima da realidade possível.

Pretende-se nesta secção apresentar algumas das operações consideradas mais importantes para este fim. Não se pretende concretizar já a forma como estas deverão ser implementadas mas apenas referir qual o seu efeito.

3.2.8.1. Carregamento de dados

Esta operação é uma das mais importantes no processo de modelação, na medida em que constitui o seu ponto de partida. Os dados geométricos são carregados de uma fonte de dados indicada e agregados numa única *shape*, conforme descrito na secção 0.

3.2.8.2.Coloração

A operação de coloração funciona a três níveis distintos: *shape*, polígono e vértice. É possível a aplicação de cor a vértices que possuam determinadas características (recorrendo para tal à referência %v), resultando em polígonos com alteração progressiva de cor. Por outro lado,

a cor também é aplicável a polígonos individuais (usando a referência %p) ou a *shapes* completas. Por definição é atribuída a cor branca às *shapes*.

3.2.8.3.Aplicação de Texturas e definição de Coordenadas UV

Como alternativa à cor, as *shapes* podem ter associada uma textura (por definição não possuem), sendo o mapeamento delas sobre os seus polígonos constituintes realizado automaticamente, e de acordo com a definição do seu âmbito. Embora não seja permitido personalizar a disposição do mapeamento, é possível rodar ou escalar os seus valores para que o mesmo aconteça na visualização da textura. Uma vez que esta aplicação se realiza ao nível do polígono, é sobretudo indicada para a definição de componentes de edifícios, não tanto para estradas ou superfícies.

3.2.8.4.Extrusão

A operação de extrusão consiste na conversão de uma figura plana num volume de base correspondente à figura e de altura indicada. A extrusão pode ser realizada a dois níveis: em superfície ou em afunilamento, sendo esta última utilizada para criar estruturas piramidais (e por isso muito úteis para representar telhados, por exemplo).

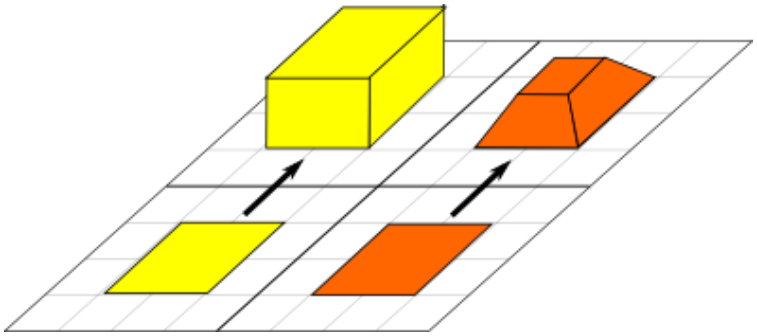

Figura 24 – Exemplo de extrusão em superfície e extrusão em afunilamento

Embora a operação de extrusão poder por definição ser executada em qualquer direcção, por definição esta é realizada na direcção da normal do polígono a que se aplica. O valor correspondente à altura constitui um dos argumentos da operação e, no caso de afunilamento, adiciona-se o factor de diferença entre o tamanho da superfície superior da base.

3.2.8.5.Transformações Geométricas: Translação, Rotação e Escalamento

A manipulação da posição, direcção e dimensão de geometrias através de transformações geométricas constituem algumas das operações mais básicas na área da computação gráfica, e como tal são indispensáveis. Em PG3D, este tipo de operações pode realizar-se a dois níveis, nomeadamente à *shape* completa ou aos seus polígonos constituintes (através da referência %p).

3.2.8.6.Optimização de Superfície e Redes Rodoviárias

A modelação de superfícies e de redes rodoviária possui algumas características particulares e por isso requer abordagens especiais. O facto de os seus elementos constituintes possuírem orientações muito semelhantes (pelo menos sempre, de uma forma ou outra,

direccionadas verticalmente) e não se intersectarem, torna assim mais fácil, por um lado, a simplificação da sua estrutura (numa malha triangular, com vértices comuns), e por outro a mapeamento de texturas ou mesmo a definição das normais. Uma vez que estes procedimentos possuem uma base comum, encontram-se concentrados nesta operação, capaz de modelar uma malha optimizada.

3.2.8.7.Colocação sobre a superfície

A superfície constitui a base sobre a qual os restantes objectos assentam. Por essa razão, é imperativa a existência de uma operação que execute essa acção. Este consiste basicamente em procurar, para cada vértice das bases das *shapes*, o ponto correspondente na superfície e copiar o seu valor, algo facilmente realizável uma vez que todos os dados se encontram georreferenciados e indexados na base de dados espacial.

3.2.8.8.Recorte

A operação de recorte consiste em desenhar uma figura geométrica convexa de n pontos com centro coincidente com o polígono a ser recortado. Embora extremamente útil, poderá causar falha e invalidez da geometria, caso a dimensão da figura exceda a da geometria a ser recortada.

3.2.8.9.Cisão

A operação de cisão consiste em dividir polígonos através do corte horizontal ou vertical com espaçamentos definidos. Esta operação é extremamente útil para definir zonas de operação mais pequenas, para poderem ser usadas em processos de recorte ou extrusão.

3.2.8.10.Triangulação

Para que possam ser visualizados mesmo nas plataformas gráficas mais básicas, convém dividir os polígonos com mais de 3 vértices em malhas triangulares. A operação de triangulação pretende cumprir esse objectivo, indicando os índices dos vértices que podem formar triângulos. Esta operação é executada sempre que um polígono complexo é carregado de uma fonte de informação ou gerado por outros meios (por meio de cisão ou recorte).

3.3. Interpretação Gráfica PG3D

Outra das potencialidades mais relevantes por detrás do conceito PG3D encontra-se relacionada com a interpretação gráfica das *shapes* modeladas. Como já foi mencionado, as PG3DShapes são constituídas por conjuntos de geometrias que descrevem o aspecto das mesmas. Para ser possível a sua visualização em qualquer biblioteca ou ferramenta gráfica, é necessário que a sua estrutura seja "traduzida" previamente. Contudo, esta transformação pode ser relativamente mais complicada dependendo das exigências e potencialidades da plataforma alvo.

3.3.1. Estrutura PG3D como Factor Chave

Em alguns modeladores tem-se procurado utilizar primitivas geométricas mais desenvolvidas, tais como por exemplo cubos, cilindros ou esferas para auxiliar nos processos de modelação, algo que quando tendo em vista plataformas e ferramentas de desenvolvimento gráficos mais específicos pode ser compensador em termos de desempenho, se elas as suportarem. Contudo, tal abordagem torna-se igualmente limitada para outras, pois nem sempre tais funcionalidades existem em todas as plataformas.

É neste sentido que o PG3D pretende operar sobre as formas mais básicas de dados gráficos sobre as quais a computação gráfica costuma operar: vértices e a sua constituição em malhas poligonais, especialmente em malhas triangulares. Tal estrutura torna-se assim muito mais facilmente integrável com plataformas de desenvolvimento, edição ou visualização gráficas, especialmente motores de jogos de computador.

Este tipo de estrutura de dados, integrada com as potencialidades de realização de consultas complexas sobre a base de dados, induz capacidades de optimização acrescidas, as quais não seriam tão fáceis se implementadas em simples estruturas de dados orientadas a objectos. Fala-se pois, da capacidade de detecção de vértices com características iguais em conjuntos largos de *shapes*, podendo assim reduzir drasticamente a quantidade de dados a serem desenhados pela placa gráfica no *rendering* de toda a cena. Este tipo de abordagem, conciliada com algumas técnicas de optimização hoje disponíveis, tais como *vertex buffers* e *index buffers* pode permitir a visualização de ambientes urbanos bastante extensos.

3.3.2. Visualização e Exportação dos Dados

A sua implementação em bases de dados torna bastante difícil, senão mesmo impossível a sua visualização gráfica da informação directamente em ferramentas da própria base de dados. Assim sendo, a implementação PG3D em base de dados consiste apenas numa parte da ferramenta, nomeadamente a destinada ao armazenamento dos dados e à realização dos processos de modelação. Como tal, as operações dirigidas à utilização dos dados gerados tem de ser realizado através de aplicações externas à base de dados, tais como serviços ou clientes que acedem à base de dados para a extracção e consequente visualização e exportação da informação. Juntamente com o modelador, estas aplicações constituirão um Sistema PG3D.

A recorrência a ferramentas externas conduz assim novamente a dois problemas referidos anteriormente que ocorrem, em muitas aplicações de modelação procedimental já nos próprios processos de geração de conteúdos: o tempo para acesso e descarregamento dos dados por um lado, e os limites de memória para acomodar essa mesma quantidade de informação. Visto que, no âmbito da modelação de ambientes urbanos, se lida com grandes quantidades de informação, estes podem tornar-se problemas sérios que exigem alguma estratégia, especialmente quando se fala de aplicações de visualização e interacção em tempo real, tais como aplicações móveis ou jogos de computador.

Uma das estratégias que tem sido utilizada ao longo do tempo em jogos de computador tem sido a divisão por "níveis". Desta forma, pequenas parcelas dos ambientes são carregadas em alturas chave do jogo (entrada ou saída de um edifício, passagem por um portão, etc.), sendo comum a apresentação de um ecrã de carregamento ("*loading*"). Desta forma resolvem-se ambos os problemas da velocidade de descarregamento dos dados como da quantidade de dados

a carregar de cada vez. Trata-se assim de uma forma básica que induz a esperas periódicas, mas à qual a maioria dos jogadores já se encontra habituada.

Outra estratégia que se tem observado cada vez mais em jogos que possuem ambientes de grande dimensão (por exemplo, *The Elder Scrolls: Oblivion*), é o recurso à técnica de *prefeching [GRSS99]*. A ideia por detrás deste conceito consiste no carregamento "em avanço" de parcelas do ambiente do jogo durante a própria sua execução. Este processo, executado em paralelo com o recurso a *threading*, elimina assim a necessidade de carregar os ambientes por completo de uma vez, podendo operar de uma forma mais *memory-friendly*, e com redução ou mesmo eliminação dos tempos de espera.

Embora estas questões sejam importantes na implementação de uma aplicação cliente que aceda aos dados à base de dados dotados com capacidades PG3D, a verdade é que, em jogos de computador, é mais comum que cada motor de jogo possua os seus próprios métodos de carregamento dos ambientes e que seja de pouco interesse o acesso constante a bases de dados (embora, em jogos *online*, esta seja uma alternativa bastante viável). É assim mais importante a possibilidade de exportação dos ambientes para formatos compatíveis com cada jogo, no qual o problema do tempo de consulta não se coloca, e pouco provavelmente as questões de memória (uma vez que a informação é directamente gravada em ficheiro no disco).

A existência do PG3D em bases de dados, que dispõe de estruturas de dados bastante optimizadas e genéricas para qualquer aplicação torna assim a sua utilização por qualquer indivíduo, equipa ou empresa bastante flexível. Dependendo da plataforma de visualização alvo que se deseja utilizar, será fácil a criação de exportadores ou interpretadores específicos.

3.4. Sumário

Neste capítulo foi apresentada a solução PG3D para a modelação de ambientes urbanos virtuais baseados em informação real, com vista ao desenvolvimento de jogos de computador. Numa fase inicial foram apresentadas várias dificuldades que assombram este tipo de aplicações, e explicado o conceito PG3D, na sua implementação sobre plataformas de armazenamento de dados como forma de as abordar.

Derivada das potencialidades que o conceito encarcera, foi enunciada a gramática PG3D que, influenciada por trabalhos de vários autores, possibilita a construção de ambientes urbanos de grande dimensão e grande detalhe, baseada em vários procedimentos de modelação e num crescimento iterativo, condicional e característico, apoiando-se em informação real e na relação de dependência mútua dos seus elementos. Foram explicados alguns conceitos fundamentais a ter em conta no seu desenvolvimento, bem como as possibilidades de desenvolvimento das regras de produção das gramáticas.

Numa fase final foi abordadas questões mais ao nível da visualização e exportação dos dados e das estruturas que o PG3D possui para integrar mais facilmente a informação criada em outras plataformas, como forma de empregar os dados num contexto de utilização prática.

Capítulo 4

Implementação PG3D

Face aos objectivos de investigação propostos para esta dissertação de mestrado foi desenvolvido um protótipo do sistema PG3D para modelação de ambientes urbanos virtuais para jogos de computador. Pretende-se então neste capítulo apresentar o trabalho realizado ao nível da implementação da solução proposta no capítulo anterior.

Como mencionado no capítulo 4.1, o ponto fundamental por detrás do modelador PG3D consiste na sua implementação em sistemas de gestão de bases de dados, aproveitando assim sobretudo as capacidades de armazenamento e velocidade de consulta aos dados que este tipo de plataforma propicia. Esta componente é, por isso, responsável pelas seguintes tarefas:

- Armazenamento das fontes de dados
- Armazenamento de todos os parâmetros de modelação e respectivos resultados
- Realização dos processos de modelação

No entanto, apesar das várias vantagens que este tipo de sistemas possa apresentar, as suas interfaces de gestão nem sempre são as mais amigáveis para a administração das tarefas mais específicas, e raramente possuem ferramentas para criação das mesmas. Por outro lado, certas operações não são sequer realizáveis nas linguagens de programação que estas plataformas oferecem, sendo por isso necessário o recurso a aplicações externas para esse fim.

Por essa mesma razão foi concebida adicionalmente uma aplicação cliente, destinada à realização das funcionalidades mais difíceis de conceber em sistemas de gestão de bases de dados:

- Gestão dos parâmetros de modelação
- Visualização dos dados criados
- Exportação dos dados criados

O funcionamento de cada uma destas tarefas será vista em pormenor nas secções seguintes.

Este capítulo inicia-se com a especificação da arquitectura e das estruturas de dados, bem como de alguns conceitos que são comuns aos dois componentes da aplicação PG3D. De seguida abordar-se-á com mais detalhe a finalidade e o funcionamento de cada um, referindo o âmbito tecnológico em que foram desenvolvidos.

4.1. Arquitectura, Conceitos e Estruturas

O Sistema PG3D implementado possui uma arquitectura cliente – servidor, sendo o servidor a simples base de dados. O servidor é evidentemente o que possui os encargos mais pesados, nomeadamente os de armazenamento e processamento dos dados, entre as quais as de operações de modelação procedimental.

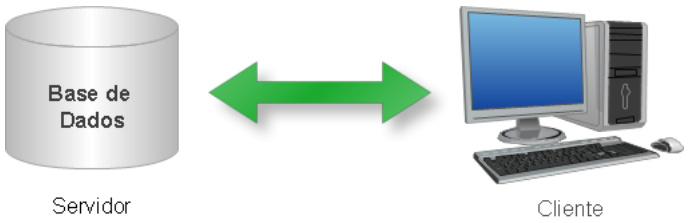

Figura 25 - Arquitectura do Sistema PG3D

O cliente, por outro lado, destina-se sobretudo à visualização e interacção com os dados da plataforma de armazenamento, o que justificará a sua classificação como *thin client*. Contudo, dado que o cliente é ainda responsável pela exportação para ficheiro e pela visualização gráfica, a sua utilização poderá ainda requerer uma máquina mais poderosa, o que poderá contrastar com o ideal do conceito. Por outro lado, a verdade é que praticamente nenhuns dados são gravados na máquina cliente, com a excepção dos dados utilizados para estabelecer a ligação ao servidor (por fins de usabilidade).

Tratando-se de uma base de dados que disponibiliza um serviço, o modelador PG3D pode ser utilizado por múltiplas pessoas em simultâneo, embora com objectivos de modelação diferentes. Essa variedade induziu a criação do conceito de "projecto", muito comum em aplicações de desenvolvimento.

Um projecto é uma divisão lógica da base de dados de modelação, que agrega um conjunto de definições e parâmetros que podem ser utilizados em múltiplas instâncias de modelação procedimental. Todas estas definições são gravadas do lado do servidor, e carregadas pelas aplicações cliente nos seus momentos de inicialização. Visto não ser gravada qualquer informação nos clientes, é possível a operação concorrente de múltiplos utilizadores no mesmo projecto. No entanto, uma vez que os processos de modelação são bastante intensos, tal poderá resultar num desempenho menor de cada um deles. Assim sendo, é recomendável o recurso a uma máquina poderosa se se pretender partilhar as capacidades do modelador PG3D.

Como especificado na Gramática PG3D (ver secção 3.2.3.), os parâmetros necessários para os processos de modelação procedimental são:

- Um conjunto de fontes de dados (transformados ou na forma original);
- Axioma, ou seja, o ponto de partida definido por um conjunto de *layers;*
- Regras de produção que constituem as instruções de modelação.

Adicionalmente é possível definir:

- Um conjunto de *boundaries* para definir mais facilmente áreas de operação;
- Um conjunto de texturas para aplicar sobre *shapes.*

Os dados que resultam da execução dos processos de modelação encontram-se, por sua vez, sobre o formato de *shapes*, que são constituídos por polígonos e vértices.

Como foi explanado, todas estas definições e dados só se encontram no lado servidor, pelo que a aplicação cliente se limita a consultá-los e actuar sobre eles quando pedido. Assim sendo, a arquitectura de dados é comum às duas vertentes de implementação (com apenas ligeiras diferenças) pelo que fará sentido enunciá-la aqui:

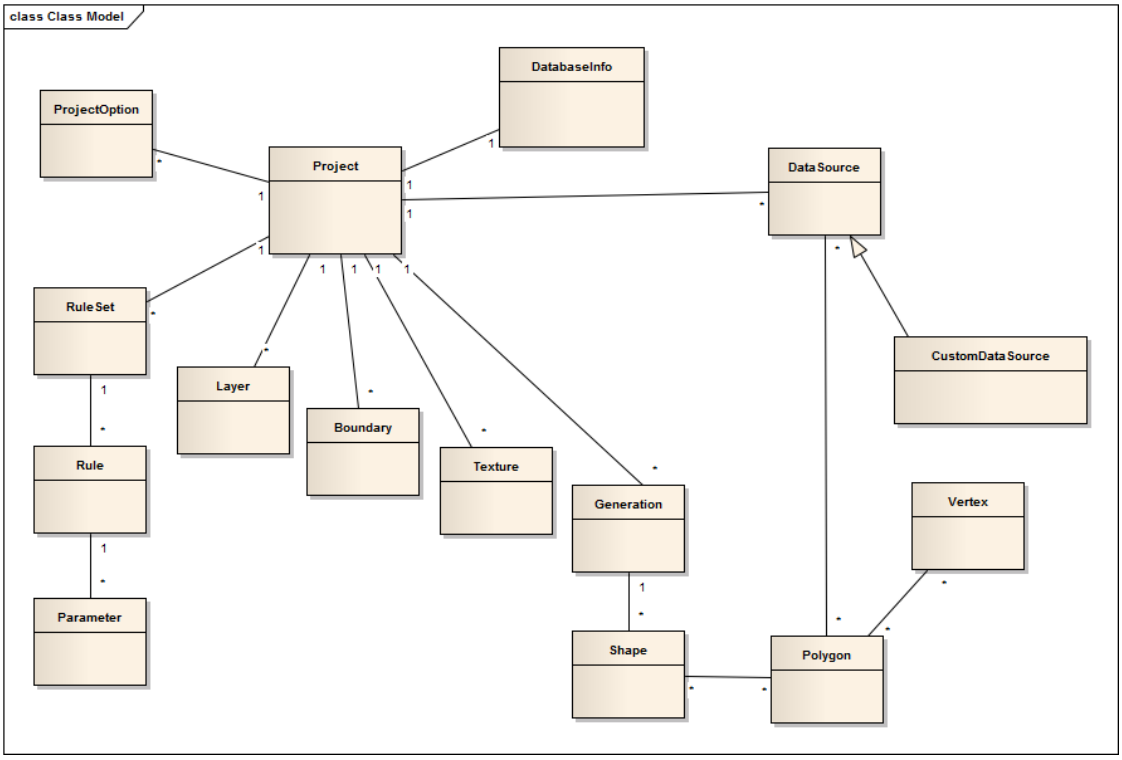

Figura 26 - Arquitectura comum do sistema PG3D

Para fins de simplificação, o diagrama de classes acima demonstra apenas as várias ligações entre as várias entidades, abstendo-se dos pormenores como os atributos ou métodos de cada uma. O projecto é assim o elemento central, que contém antes de mais um conjunto de opções (*ProjectOption*) associada a si, permitindo o armazenamento de quaisquer preferências especiais.

Para além disso, os projectos agregam texturas (*Texture*); limites (*Boundary*); camadas (*Layer*); conjuntos de regras (*RuleSet* e *Rule*), que por sua vez contêm parâmetros (*Parameter*); fontes de dados (*DataSource*), que podem ser variantes personalizadas (*CustomDataSource*); e os resultados dos processos de modelação (*Generation*). Estes últimos contêm múltiplas *shapes* (*Shape*) que podem ser constituídas por polígonos (*Polygon*) e vértices (*Vertex*). Para fins de optimização e redução de informação, os polígonos podem aparecer em mais do que uma *shape*, para evitar repetições. O mesmo pode acontecer com os vértices. Na verdade, neste último caso, a possibilidade de cada vértice estar associado a mais do que um polígono encontra-se relacionado com as questões de optimização mais avançadas, referidas na secção 3.3.

4.2. Modelador PG3D

O modelador PG3D, implementado directamente em sistemas de gestão de bases de dados, constitui o elemento principal por detrás do conceito PG3D.

Como foi explicado anteriormente, a escolha do seu nome baseou-se, para além da óbvia referência ao seu objectivo, nas tecnologias em que foi concebido na sua primeira implementação. PG3D é assim um acrónimo para a tríplice ***P**ost**g**reSQL, **P**ost**GI**S, **P**rocedural **G**eneration 3D,* sendo o número 3 não só uma referência à tridimensionalidade dos dados que são operados, mas também ao triplicidade do significado das iniciais no seu nome.

O sistema de gestão de base de dados escolhida foi o PostgreSQL, recorrendo à extensão espacial PostGIS, permitindo assim não só o carregamento e armazenamento das fontes de informações geográfica a serem utilizadas pelo utilizador, mas também a execução das operações espaciais nas estruturas de dados usadas. Como linguagem de programação foi escolhida o PL/PgSQL, que aproveita a sua implementação nativa na base de dados para aumentar o poder, flexibilidade e o desempenho das funções criadas.

Pretende-se aqui abordar, numa primeira fase, as tecnologias utilizadas para a implementação do protótipo, enumerando as várias funcionalidades e características que motivaram a sua escolha. Nesta secção pretende-se abordar a forma como foram aproveitadas, referindo alguns aspectos mais relevantes do funcionamento do modelador PG3D.

4.2.1. Plataformas de Desenvolvimento

4.2.1.1. PostgreSQL

PostgreSQL é um sistema de gestão de bases de dados *open source*. Com mais de 15 anos de existência e de desenvolvimento contínuo, possui uma arquitectura comprovada e que tem obtido uma forte reputação em termos de fidelidade, integridade e qualidade [PGDG10a]. Actualmente, o PostgreSQL é compatível com os sistemas operativos mais populares, tais como Linux, UNIX e Windows.

O PostgreSQL inclui a possibilidade de definição de chaves primárias (*primary keys*), chaves estrangeiras (*foreign keys*), junções (*joins*), vistas (*views*), gatilhos (*triggers*) e procedimentos armazenados (*stored procedures).* Engloba igualmente a realização de subconsultas, podendo os resultados destas ser realizado de forma parcial, através dos delimitadores LIMIT / OFFSET. Características a nível integridade dos dados incluem chaves primárias (simples ou compostas), chaves estrangeiras com restrições e actualizações ou remoções de dados em cascata, restrições de unicidade ou nulidade, entre outros.

Para além de possuir um catálogo de sistema totalmente relacional que suporta vários esquemas por base de dados, o seu catálogo também pode ser acedido através do esquema de informações (*Information Schema*), tal como definido no padrão SQL.

PostgreSQL segue um conjunto de padrões de conformidade. Inclui a maioria dos tipos de dados SQL: 2008, incluindo *integer, numeric, boolean, char, varchar, date, interval, e timestamp*. Suporta também o armazenamento de grandes objectos binários (*blobs*), incluindo imagens, sons ou vídeo. É possível igualmente definir novos tipos de dados, domínios, conversões e operadores sobre os mesmos.

Possui interfaces de programação nativa para *C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC,* entre outros e é dotado de documentação extensa e detalhada [PGDG10a].

O PostgreSQL apresenta um conjunto de características sofisticadas que permitem a sua elevada escalabilidade tanto ao nível da informação que consegue gerir como no número de utilizadores concorrentes que consegue acomodar. Os limites das suas capacidades são bastante extensos, como se pode constatar na tabela que se segue [PGDG10a]:

Tabela 1 – Capacidades do PostgreSQL [PGDG10a]

Limite	Valor
Tamanho máximo de bases de dados	Ilimitado
Tamanho máximo de Tabelas	32 TB
Tamanho máximo de cada entrada	1.6 TB
Tamanho máximo de cada campo	1 GB
Número máximo de entradas por Base de Dados	Ilimitado
Número máximo de colunas por tabela	250 - 1600 consoante o tipo de dados
Número máximo de entradas por tabela	Ilimitado

De forma a optimizar os processos de consulta dos dados que contém, o PostgreSQL oferece um conjunto vasto de alternativas de indexação de dados: B-tree, R-tree, hash, ou métodos de armazenamento GiST. A indexação por GiST (Generalized Search Tree) é um sistema que reúne uma grande variedade de escolhas de diferentes algoritmos de pesquisa. Permite a criação de tipos de dados personalizados, bem como métodos de consulta extensível para a procura dos mesmos. Assim, GiST oferece a flexibilidade de especificação do que é que se pretende armazenar, como se armazenar, tal como definir novas formas de procurar informação, excedendo assim as potencialidades oferecidas por outros algoritmos genéricos de pesquisa [PGDG10a].

Este conjunto de características que o PostgreSQL possui constituiu o motivo da sua escolha. Trata-se de uma base de dados *open source* com um leque de funcionalidades vasto ao nível de muitos produtos comerciais. Tem igualmente grandes capacidades em termos de extensibilidade, através das linguagens procedimentais e da habilidade de criação de tipos personalizados. Para além disso, os seus extensos limites na dimensão e complexidade dos dados constituem assim outras das vantagens e que, aliadas às capacidades de indexação e consulta sobre essas grandezas tornaram o PostgreSQL ideal para servir como base de desenvolvimento do modelador PG3D.

4.2.1.2. PostGIS

PostGIS é uma extensão para PostgreSQL que introduz o suporte para objectos geográficos, potenciando a sua utilização por sistemas de informação geográficos (GIS). PostGIS tem vindo a ser desenvolvido pela Refractions Reseach [RSCH08] no âmbito de um projecto em tecnologia de bases de dados espaciais *open source*.

O PostGIS segue a norma "Simple Features" onde assentam as diversas especificações definidas pelo *Open Geoespatial Consortium*. Como tal PostGIS inclui o suporte para vários tipos de geometria, nomeadamente [OGCI06]:

- ***Geometry:*** É a estrutura que se encontra na raiz da hierarquia, abstracta e não instanciável. Todas as restantes derivam desta classe.

- ***Point:*** Representa uma geometria de dimensão espacial nula, correspondendo por isso a um simples ponto no espaço.
- ***LineString:*** Representa uma curva com interpolação linear entre pontos (*Points*). Cada par consecutivo de pontos define um segmento de linha.
- ***Polygon:*** Superfície plana delimitada por uma curva correspondente ao seu perímetro e possivelmente recortada por 0 ou mais curvas interiores.
- ***Multipoint, Multilinestring, Multipolygon*** e ***GeometryCollection***, que aglomeram essencialmente múltiplos dos tipos referidos anteriormente, sendo esta última capaz de armazenar diferentes tipos na mesma estrutura. Note-se que em qualquer uma destas agregações é necessária que os seus elementos constituintes possuam o mesmo identificador de referenciação espacial.

Para além de possibilitar o armazenamento deste tipo de estruturas, o PostGIS inclui um conjunto de operadores espaciais capazes de realizar diversos tipos de medições tais como área, distância, comprimento e perímetro. É igualmente possível efectuar operações mais complexas, tais como união, diferença, intersecção, simplificação, entre outras.

Através de uma optimizada integração com o sistema de gestão de bases de dados em que opera, o PostgreSQL, o PostGIS inclui um suporte poderoso para indexação sobre os dados geométricos e geográficos, sendo capaz de realizar consultas complexas com elevado desempenho.

As potencialidades oferecidas por esta extensão espacial *open source* para a base de dados PostgreSQL, motivaram a sua escolha como plataforma para desenvolvimento do modelador PG3D. Uma vantagem adicional que surgiu com a sua escolha, residiu no facto de a extensão PostGIS ser cada vez mais suportada por vários sistemas de informação geográficos, permitindo assim a visualização imediata (pelo menos, no âmbito bidimensional) dos resultados de modelação concebidos.

4.2.1.3. PL/pgSQL

PL/pgSQL é uma linguagem procedimental concebida para o sistema de gestão de base de dados PostgreSQL, que introduz as seguintes funcionalidades [PGDG10b]:

- Criação de funções e gatilhos;
- Adição de estruturas de controlo à linguagem SQL;
- Faculdade de realizar computações complexas;
- Acesso a tipos de dados personalizados, funções e operadores;
- Facilidade de integração e utilização.

A linguagem SQL é das linguagens mais vulgarmente utilizadas em bases de dados relacionais para a execução de consultas. Apesar de portável e simples de usar, a linguagem SQL possui o inconveniente de cada expressão ter de executada de forma individual pelo servidor. Tal obriga a que uma aplicação cliente envie à vez cada uma das instruções, espere pelo seu processamento, receba os resultados, decida com base neles e apenas depois envie a

instrução seguinte. Tal obriga assim à comunicação constante, induzindo um *overhead* adicional a todo o procedimento [PGDG10b].

O PL/pgSQL permite a seriação de várias consultas dentro do servidor, para além da definição de estruturas de controlo típicas de uma linguagem procedimental. Tal reduz a necessidade de processos de comunicação entre o cliente e o servidor, eliminando assim o *overhead* imposto por estes, conduzindo consequentemente a um aumento no desempenho.

4.2.2. Arquitectura e Gestão de Projectos

A implementação do modelador PG3D num sistema de gestão de bases de dados como o PostgreSQL conduziu ao desenvolvimento de uma arquitectura especial, que muito difere das vulgarmente encontradas em aplicações orientadas a objectos. Tal deve-se também ao facto de na maioria dos seus casos de uso raramente serem concebidas aplicações realmente complexas. Por outro lado, a sua maior orientação ao armazenamento contribuiu para uma estrutura própria e mais ajustada, com funcionalidades mais direccionadas à manipulação de informação, mas também recheada com uma certa dose de flexibilidade e dinamismo que, quando bem aproveitadas, podem constituir oportunidades interessantes.

Assim sendo, o modelador consiste sobretudo em tabelas e estruturas de dados compostas e num conjunto de funções que sobre elas opera. Todos estes elementos podem encontrar-se agregados em esquemas, que residem basicamente em métodos para os organizar, à semelhança do que acontece em espaços de nomes em linguagens de programação. Esta característica que o PostgreSQL possui tornou-se essencial para o desenvolvimento do conceito de "projecto" mencionado anteriormente e sobre o qual se desenvolvem os processos de modelação.

Um projecto corresponde a um esquema numa base de dados. Cada base de dados pode conter um conjunto de projectos, cada um possuindo as suas tabelas, funções e estruturas de dados próprias. Cada esquema usado no âmbito PG3D é dotado das várias tabelas de parâmetros e de dados gerados (tal como definidos na secção 4.1) no momento da sua criação.

No PostgreSQL, os elementos criados sem indicação de esquema são colocados por defeito num esquema designado '*public*' (público). Este facto contribuiu para a ideia da introdução de um esquema adicional comum a todos os projectos, podendo conter todas as funções e fontes de dados necessárias, e evitando a sua redefinição em cada esquema de projecto. Assim sendo, cada projecto tem acesso a dois esquemas: o seu próprio, no qual poderão ser adicionadas novos elementos, e no esquema público, que consiste num repositório contendo:

- **Um conjunto de fontes de dados:** Estas são tipicamente imutáveis, mas poderão ser acedidos múltiplas vezes e usados na criação de dados personalizados, como origem para posterior transformação.
- **Funções de modelação e de sistema:** As funções de modelação procedimental e respectivas operações geométricas, matemáticas, bem como as responsáveis pela contextualização espacial e desenvolvimento característico e condicional.
- **Componente espacial:** Sendo uma extensão, o PostGIS consiste essencialmente num conjunto de funções e tipos de dados que são copiados para a base de dados em que se pretende utilizá-la.

4.2.3. Criação e Gestão de Regras de Produção

Uma das principais vantagens que a linguagem de programação PL/pgSQL oferece é a facilidade e dinamismo com que as suas funções podem ser criadas. Tal revelou ser um elemento extremamente importante que contribuiu para um processamento mais simples, rápido e directo das regras de produção.

Uma regra de produção da Gramática PG3D corresponde a uma função PL/pgSQL. Ao contrário do que acontece em vários modeladores, em que a cada regra de produção da gramática é avaliada, processada e verificada com métodos próprios, na implementação do modelador PG3D tal é realizado através da sua conversão para PL/pgSQL, podendo assim desfrutar das capacidades de interpretação e processamento que o PostgreSQL já possui para esta linguagem.

Por outro lado, a utilização das regras torna-se mais eficiente na medida em que não é interpretada "em bruto" durante a execução do próprio processo de modelação procedimental. Apesar de a linguagem PL/pgSQL ser interpretada e não compilada (o que tornaria em princípio a sua execução bastante mais lenta), a verdade é que esta prossegue à realização de planos de execução, na qual define uma estratégia de interacção com tabelas, índices e conjuntos de resultados. Tal contribui para um desempenho bastante superior na medida em que o seu código não necessita de ser reinterpretado de cada vez que é executado.

Parcialmente com vista a esta implementação, o conceito da gramática PG3D possui várias semelhanças com a linguagem PL/pgSQL. A declaração de regras com parâmetros, por exemplo, é algo que existe em qualquer linguagem de programação que declare funções, não sendo esta uma excepção. Por outro lado, existem as operações booleanas e matemáticas que actuam de forma análoga. Outros exemplos serão as estruturas de controlo IF…THEN…ELSE cujo funcionamento é praticamente idêntico em ambos os casos. Contudo, uma vez que a gramática PG3D é ainda consideravelmente dinâmica, possuindo capacidade de acesso e consulta às características dentro das funções de operação (através das várias referências, veja-se a secção 3.2.6.), ainda subsistem alguns pontos que só podem ser interpretados na altura da execução.

4.2.4. Classes de funções e operações PG3D

O desenvolvimento do modelador PG3D rodou essencialmente em volta do desenvolvimento de funções, dada a natureza do PostgreSQL. Embora organizadas pelo sistema de forma aparentemente desconexa e independente, estas encontram-se ligadas e relacionadas, quer pelos dados sobre os quais operam, quer pela sua mútua dependência ou mesmo pelo tipo de actividade que desempenham.

É possível, pois, classificar as funções e operações criadas nas seguintes classes:

- **Funções de sistema:** Controlam o processo de modelação procedimental, gerindo as *shapes* criadas, as regras de produção a serem utilizadas e todo o fluxo iterativo;
- **Funções de carregamento de dados:** São as responsáveis pelo início das operações de modelação, carregando os dados das fontes indicadas;
- **Funções de modelação:** Realizam os vários tipos de transformações e construções geométricas. O seu desenvolvimento é o mais expansível, na medida em que a

contínua criação de mais membros só contribui para uma maior facilidade e potencial de construção de modelos tridimensionais;

- **Funções de acesso:** Encarregadas de aceder aos dados respectivos às *shapes*, polígonos e vértices, permitindo o desenvolvimento condicional e característico dos processos de modelação;
- **Funções de construção:** Correspondentes ao conceito de construtor para os tipos de dados compostos a utilizar;
- **Funções contextuais:** Apontadas para a verificação de intersecções, vizinhanças, sobreposições, etc.;
- **Funções matemáticas:** Designadas a executar operações de cariz matemático (aritméticas, algébricas, trigonométricas, etc.);
- **Funções de conversão ou *casting*:** Para garantir o uso dos tipos de dados correctos nos parâmetros da várias operações;
- **Funções de produção:** Não existem no modelador *a priori*, sendo apenas criadas com base na definição das regras de produção especificadas pelo utilizador (que como foi indicado na secção anterior, são convertidas em PL/pgSQL).

Uma referência completa das várias funções criadas no âmbito do modelador PG3D encontra-se no Anexo B.

4.2.5. Tipos e Estruturas de Dados PG3D

Nas várias classes de operações mencionadas na secção anterior, é necessária a noção da variedade dos tipos e estruturas de dados que são manipulados. A admissão específica de determinados tipos em certos parâmetros de operações, bem como a imperativa declaração dos tipos de dados nos parâmetros das operações obriga à sua correcta utilização sob a pena de erros ou em alguns casos a perdas de precisão ou mesmo inversão do seu significado (caso ocorra *overloading* de operadores). Os tipos de dados mais básicos encontram-se, como é óbvio, determinados pelos que o PostgreSQL contém [PGDG10c], podendo mesmo ser consultados em documentação própria. Entre outros, enumeram-se:

- ***integer, float, double, numeric***: Tipos de dados numéricos, com diferentes tipos de precisão de vírgula flutuante.
- ***boolean***: Tipo de dado que pode assumir os valores *true* ou *false.*
- ***text***: Cadeia de caracteres de tamanho ilimitado. A sua definição carece de uma inclusão entre plicas (‘’).
- ***geometry*:** Tipo de dados correspondente a uma geometria espacial, seguindo a especificação “Simple Features” [OGCI06]. As *PG3DBoundaries* são especificadas neste tipo de dados.

Para além destes tipos, o modelador PG3D inclui um conjunto de estruturas de dados adicionais que são, essencialmente composições dos tipos anteriores. No entanto, dada a

existência de campos não obrigatórios, por vezes é conveniente ter conhecimentos dos vários construtores existentes para sua definição (uma referência completa pode ser consultada no anexo B):

- ***vector3:*** Determinação de um vector com 3 coordenadas, XYZ e com identificador de referência geoespacial (Srid).
- ***rgba:*** Descrição de cor e valor *alpha* para transparência.
- ***scope:*** Definição de um âmbito. Possui indicação da direcção dos 3 eixos XYZ, com tamanhos correspondentes à dimensão do âmbito (sendo apenas necessário normalizá-los para a obtenção de vectores com tamanho 1) e um ponto correspondente à origem dos eixos.
- ***vectorUV:*** Definição das coordenadas UV para mapeamento de texturas.
- ***record:*** Definição de uma entrada numa tabela de base de dados.

Adicionalmente, em alguns casos é imperativa a definição de colecções de objectos do mesmo tipo, algo já suportado em PostgreSQL:

- ***array:*** Permite agregar conjuntos de dados do mesmo tipo. A sua definição é realizada através de parêntesis rectos (por exemplo, para inteiros seria '[1,3,4,5]').

Como definido no conceito de gramática PG3D, por vezes é conveniente a consulta de dados de *shapes*, polígonos ou vértices. Tal pode ser realizado através das referências %s, %p e %v. Contudo a sua utilização não é suportada em todas as operações, pelo que é conveniente a consulta da sua documentação (Anexo B).

4.3. Aplicação de Gestão PG3D

O sistema de gestão de bases de dados PostgreSQL inclui já variadas formas de aceder aos dados armazenados, bem como de executar consultas, criar e utilizar tipos de dados personalizados e de chamar funções, sendo assim suficiente para a utilização das funcionalidades de modelação PG3D. No entanto, tal dificilmente poderá ser considerada a forma mais amigável e eficiente de dispor das mesmas, para além de não incluir formas de exportação e visualização dos dados gerados. Assim sendo, a criação de uma ferramenta de gestão do modelador constituiu um dos pontos importantes na implementação.

Com vista a aproveitar grande familiaridade com a linguagem a as possibilidades de construção de interfaces apelativas, foi utilizado para o desenvolvimento desta aplicação a linguagem C# da plataforma .NET em *Visual Studio 2008.*

A aplicação de gestão PG3D poderá ser denominada de *thin client*, visto não realizar grandes processamentos para além de ajudar na criação, edição e visualização dos dados presentes no servidor, a base de dados. Da mesma forma, todos os dados que ela opera não se encontram localmente, mas sim no servidor, carregando-os sempre que é inicializado ou forçado a tal por pedido de actualização do utilizador.

Como já foi mencionado, todos esses dados encontram-se agregados no conceito de "projecto", que agrega fontes de dados transformadas, camadas, conjuntos de regras, limites,

texturas e os resultados conseguidos. Cada instância de execução do cliente é capaz de operar sobre um projecto de cada vez. Ao ser iniciada, a aplicação possuirá a possibilidade de configurar os dados de acesso a uma base de dados com suporte PG3D e guardá-los-á numa lista própria (sendo esta a única informação a ser guardada no lado cliente). Uma vez conectada à base de dados, a aplicação lista os vários projectos que nela já foram criados.

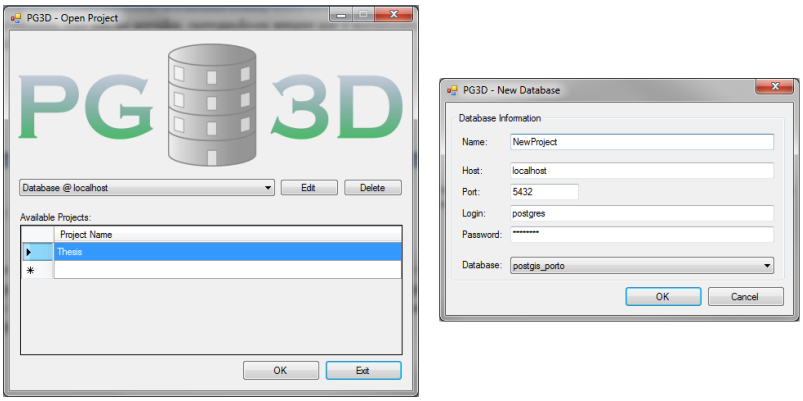

Figura 27 - Janelas iniciais para criação e selecção de um projecto

Uma vez escolhido um projecto, o utilizador será defrontado com a interface de gestão do projecto. Não pretende nesta fase apresentar em pormenor como utilizar a aplicação (tal pode ser consultado no anexo A), mas sim demonstrar algumas das facilidades que este cliente pretende proporcionar.

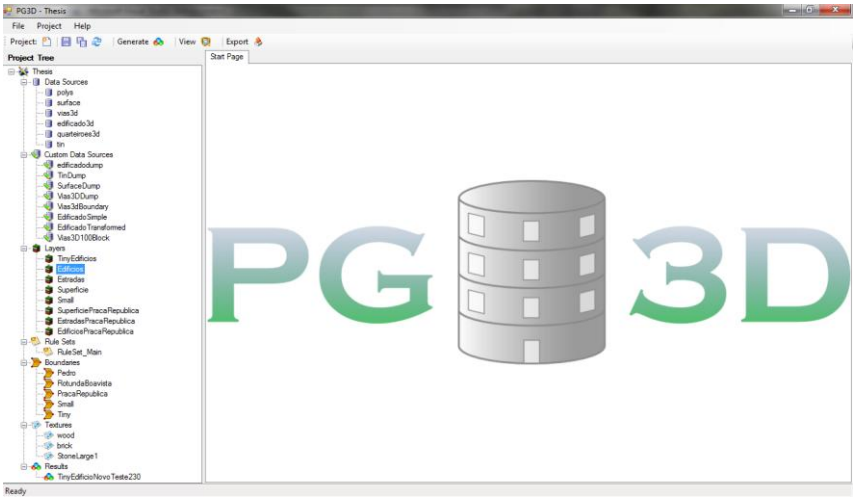

Figura 28 - Interface Gráfica da Aplicação de Gestão PG3D

A interface de gestão de projecto é semelhante à de muitas ferramentas de desenvolvimento de código, na medida em que possui uma árvore com os vários objectos que podem ser abertos no gestor de separadores do lado direito, e lá consultados ou trabalhados. A barra de ferramentas ou o menu de trabalho em cima constituem os acessos para criação de novos objectos, para iniciar processos de modelação ou para visualizar ou exportar os dados.

4.3.1. Transformação das fontes de dados

Cada fonte de dados corresponde a uma tabela na base de dados, e esta pode ser partilhada por vários projectos ou exclusiva ao projecto, se for resultado da transformação de

um das anteriores. É possível realizar igualmente transformações sobre fontes já transformadas até se obter os resultados pretendidos.

A interface para a sua criação pretende ser relativamente simples de utilizar, requerendo no entanto conhecimentos de SQL para descrever as operações de transformação.

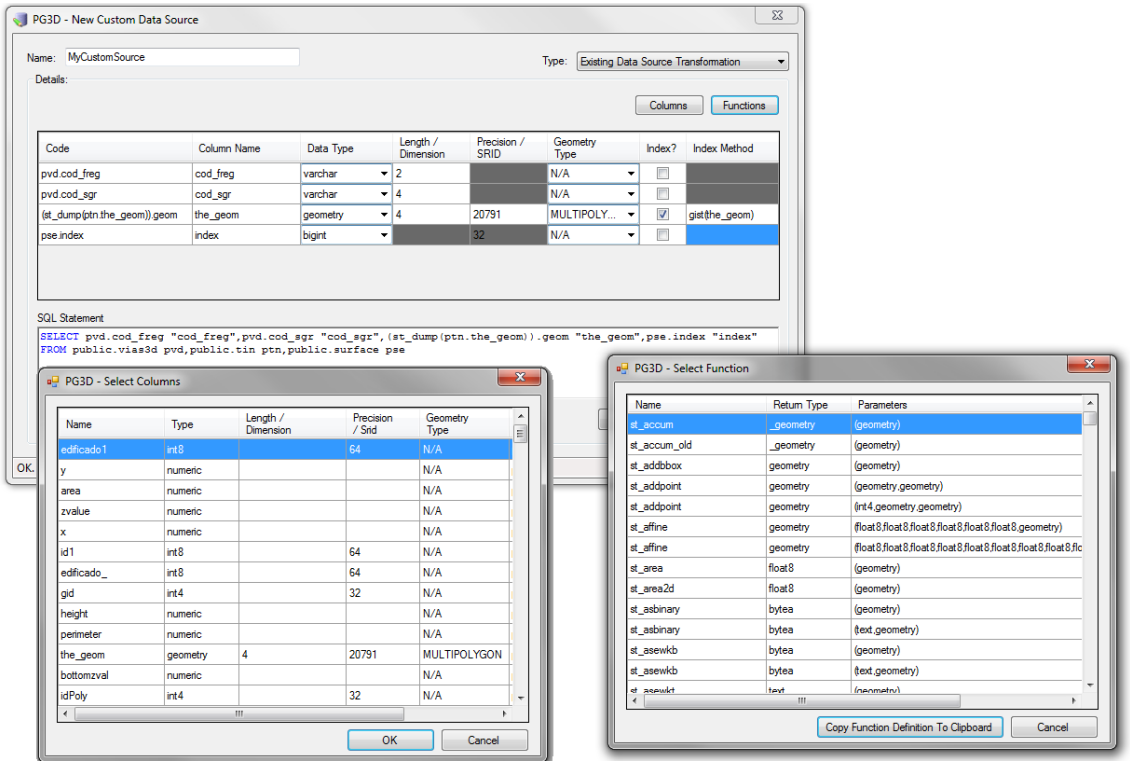

Figura 29 - Interface de Criação de fontes de dados personalizados

A interface permite a selecção de colunas de tabelas existentes, sendo mostrados pormenores sobre o tipo de cada uma. Uma vez seleccionadas, estas serão adicionadas à lista e o código SQL para as referenciar é criado automaticamente. A partir daí, é possível chamar funções sobre as colunas, havendo para isso uma referência que contém o seu tipo e parâmetros. No entanto, dado que não é possível obter descrições delas automaticamente, é conveniente que o utilizador tenha conhecimento da sua aplicabilidade.

É possível, adicionalmente, manipular o nome, tipo e dimensão das colunas ou no caso das geometrias, o seu identificador de referência geográfica. Por fim, visto a rapidez de acesso aos dados ser de extrema importância, esta interface disponibiliza a aplicação de índices a colunas, sendo apenas necessário indicar a função de indexação que se pretende utilizar.

4.3.2. Edição de Regras de Produção

Para a edição das regras de produção, foi construída uma interface muito simples, semelhante a um editor de texto, mas com realce de cor em certas palavras e expressões, conforme a sua utilidade na gramática PG3D.

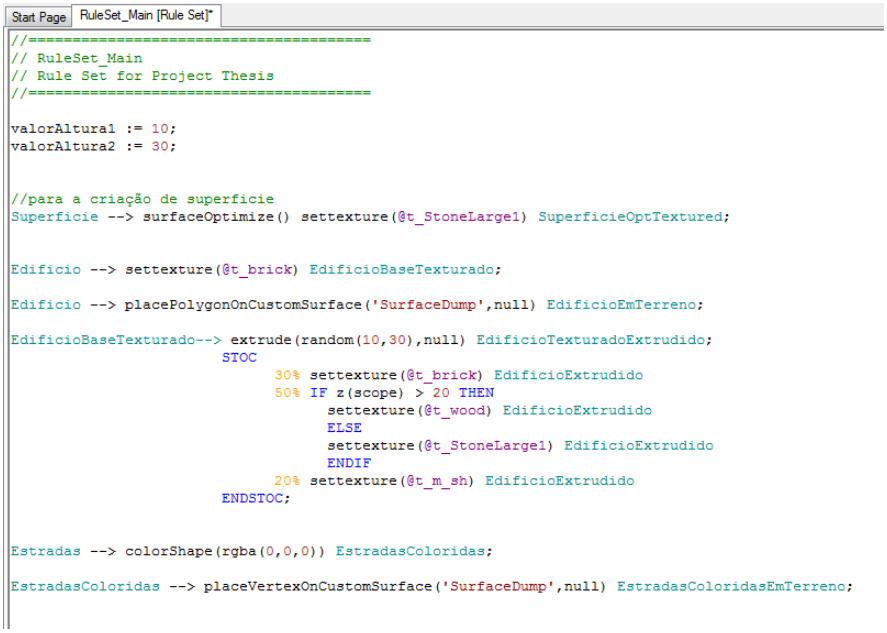

Figura 30 - Editor de regras de produção

A definição das regras é independente do espaçamento, da tabulação e da mudança de linha. É possível assim alterar a disposição das expressões conforme for de maior agrado ao utilizador. A aplicação de comentários funciona ao nível de linha - iniciado pelos caracteres '//', podendo ser iniciado em qualquer parte da linha mas apenas comentando os conteúdos depois da linha, ao estilo das linguagens de programação da família C ou Java.

A definição das regras necessita de ser realizada imperativamente antes da sua utilização. Seguindo a definição da gramática PG3D, as variáveis são referenciadas pela arroba ('@'), tal como as texturas e os limites são referenciados pelo '@t_' e '@_b', respectivamente.

As regras são assim processadas num agregadora, chamado *ruleset*. Este define, de certa forma, um espaço de nomes (*namespace*) à semelhança do que se passa em várias linguagens de programação. É possível assim definir regras com os mesmos nomes em *rulesets* diferentes. A gravação das alterações realizadas no *ruleset* consiste na eliminação de todas as regras deste espaço de nomes, na sua reanálise e recompilação. Como foi dito antes na secção 4.2.3., as regras são convertidas para funções *PL/pgSQL* na base de dados.

4.3.3. Edição de *Layers*

A edição de *layers* não difere muito da edição de regras de produção, exceptuando o facto de apenas suportar a definição de uma regra e esta não necessitar de predecessor. A interface de escrita é igualmente baseada num editor de texto, com suporte para coloração da sintaxe e palavras mais importantes.

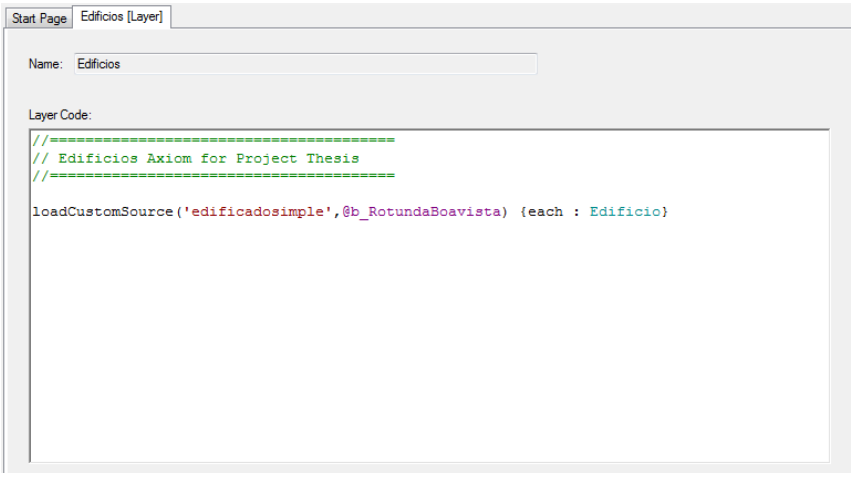

Figura 31 - Editor de *Layers*

4.3.4. Importação de Texturas

A definição do caminho para os ficheiros de texturas pode constituir uma tarefa demorada, especialmente quando em grande quantidade. Por essa razão, uma forma mais eficiente de carregamento foi criada, de forma a acelerar o processo:

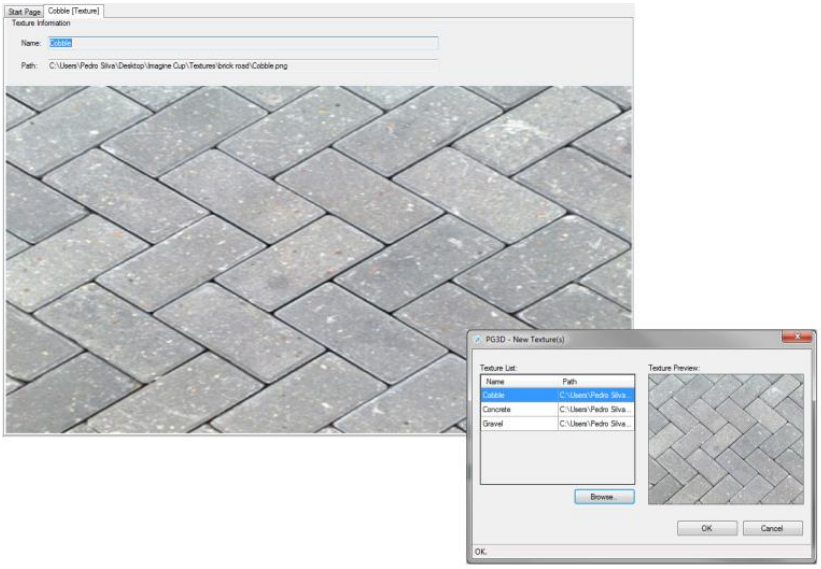

Figura 32 - Visualização e carregamento de texturas

A ideia consiste em poder seleccionar vários ficheiros ao mesmo tempo, sendo os identificadores das texturas depois atribuídos com base no nome do ficheiro (mas podendo ser manualmente alterados). Depois de importadas para o projecto, as texturas podem ser consultadas em qualquer altura.

4.3.5. Edição de Boundaries

As *boundaries* constituem provavelmente o tipo de dados cuja definição implicaria um maior esforço, dado que as coordenadas que a constituem devem possuir grande precisão, que só poderia ser consultada num mapa ou sistema de informação geográfico. Para este efeito, foi implementado nesta ferramenta uma integração com o *Google Maps*, que permite a definição de polígonos com facilidade, e disponibilizando os dados em latitude e longitude, logo proporcionando uma utilização quase directa.

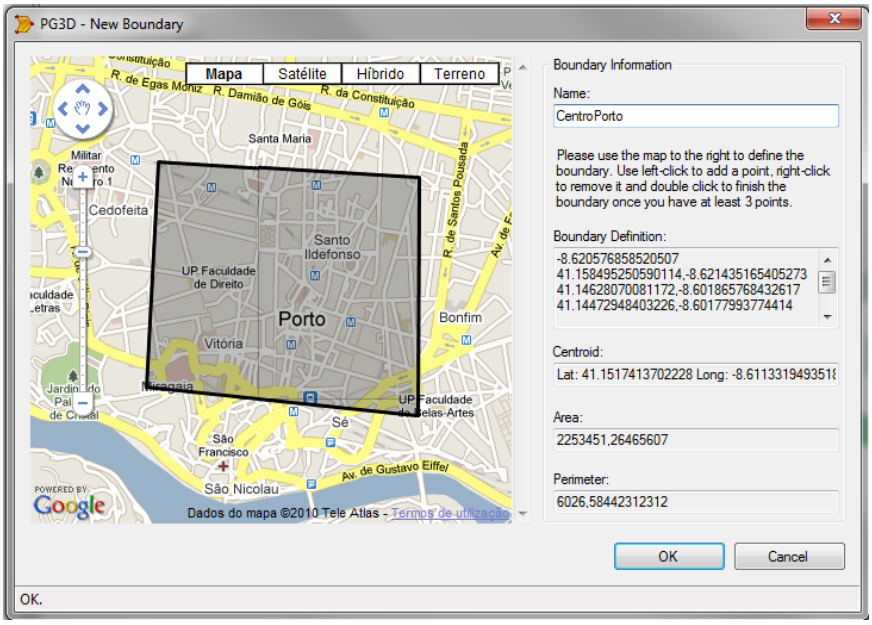

Figura 33- Janela de definição rápida de *boundaries*

Para além de disponibilizar facilidade de desenho, esta janela permite igualmente consultar os valores das coordenadas, bem como a localização do seu centro. O perímetro e a área da zona marcada são também mostrados, permitindo assim obter a noção da dimensão do espaço seleccionado.

4.3.6. Iniciação do Processo de Modelação

Estando todos os ingredientes necessários prontos, é possível realizar processos de modelação procedimental. Os detalhes podem ser consultados na figura que se segue: é necessário atribuir um nome à instância de modelação, para além da óbvia definição do conjunto de regras e das *layers* que compõem o axioma do processo. É igualmente possível definir a iteração máxima que pode ser atingida. Estando este campo em branco, não será imposta qualquer limitação neste sentido.

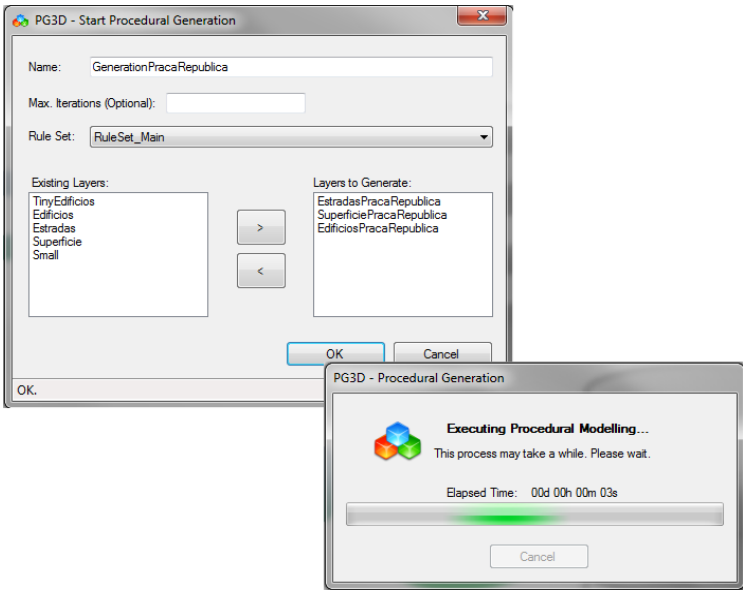

Figura 34 - Janela para definição dos parâmetros de modelação e consequente janela de progresso

Uma vez iniciado o processo, é mostrado uma janela de progresso com um contador de tempo decorrido. Uma vez que é complicado determinar o tempo restante, não é transmitida qualquer informação nesse sentido, sendo apenas garantido, através do movimento da barra de progressos, que a operação ainda se encontra em execução.

4.3.7. Visualização dos Dados

A visualização de dados pode realizada directamente na aplicação de gestão PG3D, característica esta que constitui uma das razões principais por detrás da sua criação. Nesta fase de implementação, apenas se encontra disponível uma alternativa de visualização, nomeadamente a *Framework* XNA. Esta possui infelizmente limitações a partir de um determinado número de polígonos sendo por isso pouco aconselhável em quantidades de dados maiores. Porém, o visualizador XNA encontra-se implementado sobre forma de um módulo, de forma que a criação de novas formas de apresentação dos dados gerados é uma tarefa relativamente acessível.

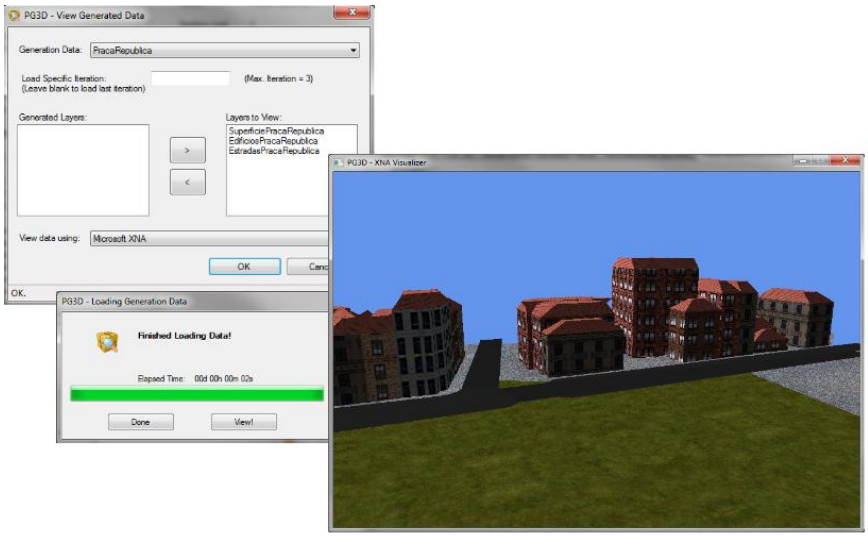

Figura 35 - Visualizador PG3D

Como se pode observar na interface, a visualização realiza-se sobre os resultados de uma instância de geração, daí a importância que estes possuam um nome. É possível visualizar apenas parte das *layers* geradas, bem como descarregar os resultados apenas até uma certa iteração, algo que poderá ser extremamente útil para constatar a evolução dos procedimentos de modelação.

Antes de os dados poderem ser apresentados, estes têm de ser descarregados da base de dados e colocados em estruturas de dados próprias (nomeadamente *vertex*- e *indexbuffers*), pelo que o processo poderá ser mais demorado para quantidades de dados maiores. Por essa razão é apresentado uma janela de progresso (de forma semelhante à geração) durante a execução destas tarefas e estando elas prontas, bastará recorrer ao botão de visualização. A razão por detrás da visualização não imediata após o carregamento deveu-se ao facto de muitas vezes ser mais conveniente poder abrir e fechar a janela de visualização XNA múltiplas vezes sem necessitar de se recarregar os dados.

4.3.8. Exportação dos Dados

A exportação dos dados constitui outra das razões principais por detrás da criação de uma aplicação cliente e constitui uma das principais funcionalidades desta ferramenta. A sua janela de interface é semelhante à de visualização, na medida em que actua sobre resultados de gerações, que permite a inclusão de apenas algumas *layers* e que faculta a selecção de uma iteração que não imperativamente a final. Adicionalmente, e como é óbvio, a janela pede também o ficheiro de destino, bem como o tipo de formato para o qual o ficheiro deve ser exportado. De momento, o único suportado é o COLLADA.

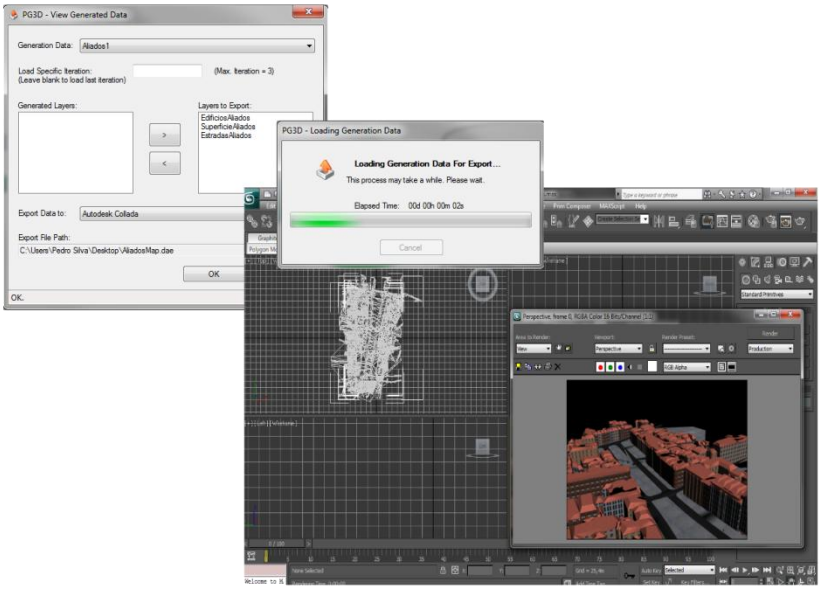

Figura 36 - Exportação dos dados e respectiva consulta na ferramenta 3ds Max

A escolha sobre o COLLADA deve-se ao seu recente crescimento em popularidade no ramo dos jogos de computador. De facto, para além de suportado por alguns motores de jogos, é possível importar este tipo de ficheiro para uma ferramenta de modelação como o 3Ds Max, que é um ponto de ligação entre muitos outros.

4.4. Sumário

Neste capítulo foram abordados alguns dos aspectos mais importantes do sistema PG3D, cuja constituição consiste no modelador, implementado no sistema de gestão de bases de dados PostgreSQL, dotado da extensão espacial PostGIS e da linguagem de programação Pl/PgSQL; e da sua aplicação cliente criada como forma de gerir os parâmetros e conteúdos do modelador. Ambos possuem uma arquitectura comum, dada a maior centralidade das operações na componente da base de dados. Foi discutido o conceito de projecto, muito importante para a utilização e organização dos dados do PG3D, bem como os vários elementos que o constituem.

Relativamente ao modelador, foram enunciadas as várias tecnologias utilizadas para o seu desenvolvimento, bem como as razões para a sua escolha. Seguiu-se a explicação de alguns aspectos de implementação mais importantes, especialmente ao nível da interpretação das regras de produção e da gestão de dados. Foram referidas também as várias classes de funções que constituem o PG3D, bem como as estruturas de dados mais recorrentes.

Quanto à aplicação cliente, foi feita uma observação mais superficial das suas funcionalidades de gestão de parâmetros, e mais especialmente sobre as capacidades de visualização e exportação dos dados.

Em suma, apesar de se encontrar numa fase de protótipo, tanto o modelador e a sua aplicação cliente já possuem uma larga quantidade de funcionalidades, com forte correspondência ao conceito PG3D apresentado na solução, e capazes de produzir resultados bastante interessantes, como se poderá constar no capítulo que se segue.

Capítulo 5

Resultados

O modelador PG3D foi desenvolvido com o intuito de facilitar a modelação de ambientes urbanos virtuais através da redução da carência de interacção humana no processo, contribuindo assim para um menor esforço, menor tempo de concepção e, consequentemente, menores custos de produção, sendo por isso ideal em aplicações que recorram a este tipo de conteúdos, tais como os jogos de computador.

Para poder comprovar as vantagens que o PG3D enuncia é necessário executar testes a diversos níveis, e comparar os seus resultados com outras aplicações que proporcionem as mesmas faculdades, nomeadamente de modelação de ambientes urbanos virtuais, correspondentes a ambientes urbanos reais.

Neste capítulo pretende-se assim apresentar vários exemplos de modelação criados pelo modelador PG3D e a sua ferramenta de gestão, analisando-os nos seguintes aspectos:

- Facilidade da definição, manutenção e actualização e utilização dos modelos produzidos;
- Qualidade, nível de detalhe e fidelidade visual dos modelos criados;
- Tempo de execução dos procedimentos de modelação, carregamento e exportação dos dados.
- Possibilidades de Contextualização Geoespacial
- Aplicabilidade em Jogos de Computador

Como elementos de comparação será considerado o modelador XL3D [COEL07], o modelador CityEngine [MULL07, PRCI09] e outras ferramentas de modelação manual, tais como o 3Ds Max ou Maya. Salienta-se no entanto o facto de que estas avaliações apenas pretendem mostrar a vantagem relativa de uma ferramenta para outra em cada uma das abordagens, mostrando em que áreas em que cada um se destaca. Assim, Estas avaliações foram sobretudo realizadas de forma empírica, dada a experiência obtida com as ferramentas.

5.1. Definição dos processos de modelação

A definição das regras de produção constitui a tarefa central a ser cumprida pelo utilizador que pretenda obter modelos criados procedimentalmente. Tal, contudo, pode não ser uma tarefa fácil, dada a necessidade de uma consciência prévia dos elementos que dispõe e que pretende manipular. Entre outros, inclui-se as fontes de informações disponíveis e a forma como

estão estruturadas; os materiais, texturas e limites declarados; ou mesmo os modelos que vão sendo criados, cujas estruturas complexas requerem boas capacidades espaciais por parte de quem os pretende conceber. Dada que a declaração das regras se efectua de forma textual, ou seja, sem possibilidade de visualização imediata, este pode ser um factor menos atraente no sistema PG3D para a maioria dos utilizadores.

Por outro lado, apesar de a sua declaração textual, a sua declaração é relativamente fácil em ambientes mais básicos, tal como a definição de simples blocos de edifícios colocados sobre uma superfície. Considere-se o seguinte ambiente na imagem seguinte que apresenta o resultado da modelação da avenida dos aliados e que demonstra as capacidades de modelação dos vários ambientes e a sua integração com os dados de elevação do terreno:

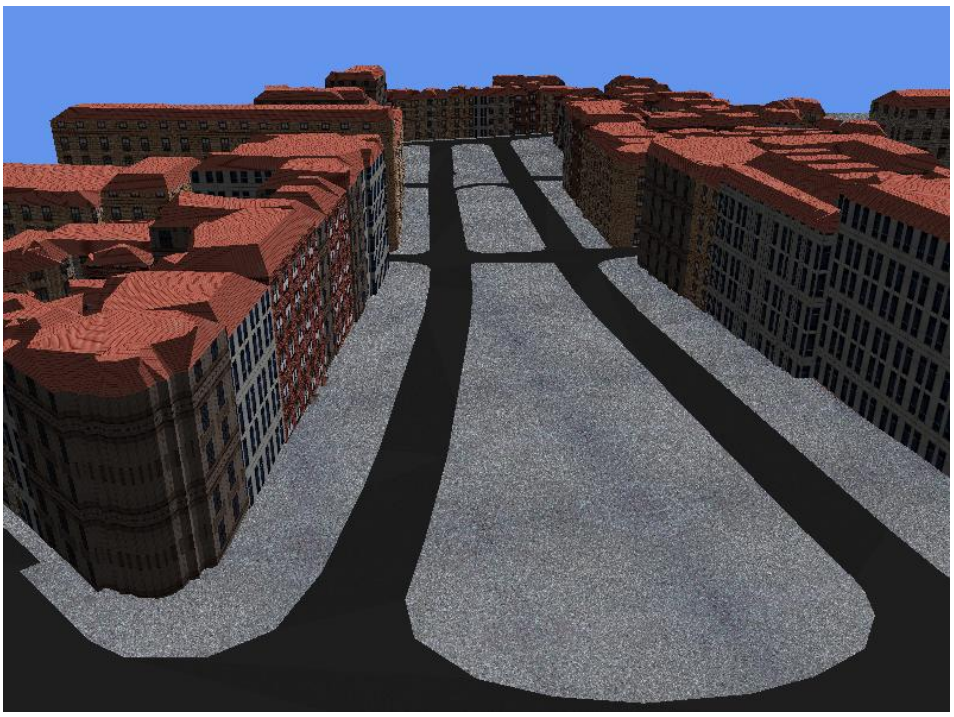

Figura 37 - Modelação da Avenida dos Aliados

Apesar de já ter um aspecto relativamente composto, a sua definição é extremamente curta e simples, algo que se reflecte na qualidade dos modelos que, para além de não se tratarem de edifícios realistas, é possível notar fachadas menos bem conseguidas. As regras de produção utilizadas foram as seguintes:

```
Superficie --> surfaceOptimize(20,20) setTexture(@t_Stone1) SuperficieCimentoTexturado;

Estradas --> surfaceOptimize(20,20) setTexture(@t_Asphalt)
               placeVertexOnCustomSurface('SurfaceDump') EstradasTexturadas;

Edificio --> placePolygonOnCustomSurface('SurfaceDump')
              extrude(double(record(%p),'height'))
             STOC
                    15% setTexture(@t_Window1)
                    15% setTexture(@t_Window2)
                    15% setTexture(@t_Window3)
                    15% setTexture(@t_Window4)
                    15% setTexture(@t_Window5)
                    25% setTexture(@t_Window6)
             ENDSTOC
             setUV(5,5)
             {hasTag(%p,'Edificio.Top') %each : EdificioTopo, %rest : EdificioLados};
```

```
EdificioTopo --> extrudeTaper(3,5) setTexture(@t_RoofRed) EdificioTelhado;
```

Para além do axioma (que se limita a carregar os edifícios), foram definidas 4 regras de produção. A primeira, consiste em optimizar a superfície, convertendo-a numa malha poligonal e definindo o seu mapeamento (indicando que 20 unidades do seu tamanho corresponderão a uma unidade de mapeamento *(s,t)*) e a textura a ser aplicada à mesma (a textura de pedra). A segunda realiza o mesmo respectivamente às estradas (colocando uma textura de asfalto) e ajusta o nível das estradas à orografia do terreno. A terceira regra é mais complexa: coloca as bases dos edifícios sobre o terreno e depois executa a operação de extrusão para lhe dar altura. O argumento da altura é carregado das fontes de informação de onde o registo relacionado com o edifício em questão foi extraído através da função "*record*". De seguida, é determinada a textura a aplicar sobre as fachadas dos edifícios. No entanto, como não se possui informação sobre esse aspecto, é aplicado aqui um método estocástico, sendo a textura escolhida de forma aleatória de um conjunto de 6 possíveis, cada uma com igual probabilidade de ser escolhida (exceptuando a última, para completar os 100%). É definido o mapeamento das texturas e logo de seguida é feita uma triagem dos polígonos. Os que se corresponderem aos lados permanecerão como estão e os correspondentes ao topo serão tratados na quarta regra. Nesta última é construído um telhado através de uma extrusão que afunila, e decidida a textura a ser aplicada.

Como se pode constatar, mesmo através de um número pequeno de regras é possível obter resultados consideravelmente satisfatórios. No entanto, para níveis de detalhe superiores, tal já poderia necessitar de regras mais complexas que, para além de mais difíceis de conceptualizar, são mais propensas a falhas e declarações erróneas.

O diagrama seguinte apresenta um comparativo das várias ferramentas enunciadas, no âmbito da facilidade de criação de regras de produção para modelação de ambientes de grande dimensão:

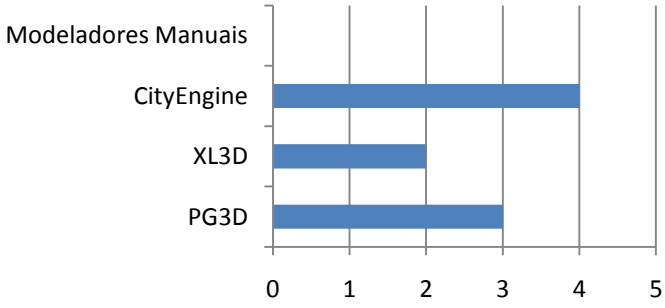

Figura 38 - Avaliação do nível da facilidade de criação de regras de produção numa gama 0-5

Nesta análise, os modeladores manuais não são considerados, dada a ausência destas funcionalidades. Nos restantes casos, o CityEngine poderá ter uma melhor avaliação dada a sua capacidade de visualização quase instantânea da criação e alteração das regras. Já entre o PG3D e o modelador XL3D a diferença não é grande, com a excepção o PG3D fornecer algumas facilidades de criação adicionais (por exemplo, a criação de *boundaries* ou visualização iterativa das várias fases de modelação).

5.2. Manutenção e Actualização dos modelos

O método de modelação através de regras de produção concede uma facilidade maior, especialmente quando considerando a dimensão dos ambientes a criar. Assim, a alteração de um elemento que se possa repetir em várias instâncias requer apenas a modificação de regra responsável pela sua criação, enquanto num modelador manual tal operação exigiria a acção sobre todas as instâncias, tarefa esta que seria extensa e desgastante.

Ao modelar ambientes existentes, surge a relação com as fontes de informação que os descrevem. Dadas as constantes evoluções e transformações que ocorrem nos vários elementos que compõe os ambientes urbanos, é necessário que os modelos virtuais que as representam também acompanhem essas alterações. Assim, a sua relação para com dados reais é fundamental e muito mais conveniente, na medida em que bastará a reaplicação das mesmas regras sobre os novos dados para se realizar a actualização dos ambientes virtuais.

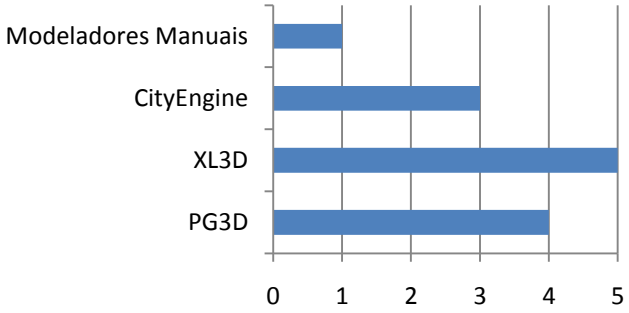

Figura 39 - Avaliação do nível da facilidade de manutenção dos modelos numa gama 0-5

Neste comparativo torna-se óbvio que as ferramentas procedimentais são muito mais vantajosas no que diz respeito à manutenção dos modelos criados, pelas razões acima referidas. Apesar de as três restantes alternativas possuírem ligação a fontes de dados, a sua relação não é igual, já que no caso do PG3D e *CityEngine* ser necessário uma reimportação e retratamento dos dados reais e, no caso deste último, o aproveitamento dos dados reais não ser tão poderoso. No entanto, o modelador XL3D é o que possui a maior vantagem, na medida em permite o acesso dinâmico e remoto às fontes de dados, bastando que estas se encontrem actualizadas para realizar a manutenção instantânea dos seus modelos.

5.3. Capacidade, Qualidade e Nível de detalhe

Uma dos factores importantes na modelação é a qualidade e nível de detalhe dos modelos criados. Para que seja possível atingir certos níveis de pormenor é necessário que as suas ferramentas de produção sejam suficientemente capazes de reproduzir edifícios existentes na realidade (ou pelo menos, a sua maioria) com elevado nível de detalhe. O desenvolvimento dessa capacidade foi um dos principais objectivos do modelador PG3D. Um exemplo de um edifício criado no seu âmbito encontra-se na figura seguinte:

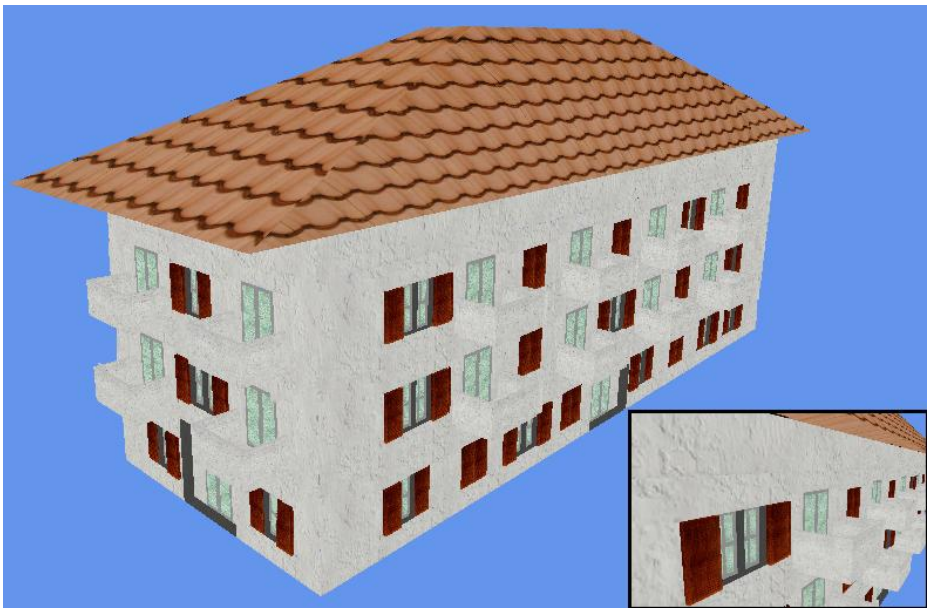

Figura 40 - Modelo detalhado de uma casa concebido em PG3D

O modelo da imagem mostra como é possível definir com bastante detalhe os vários elementos da fachada de uma casa. Nesta foram modelados com pormenor as várias janelas e varandas, portas, portadas, com um grau razoável de variabilidade – é possível notar que o número de portadas abertas em cada janela varia. São se trata da simples aplicação de texturas, mas de elementos com volume (as janelas têm beirais, as varandas têm profundidade, etc.). A concepção deste tipo de estruturas necessita, contudo, da definição de um maior número de regras (que pode ser consultada no Anexo C), tal como mais tempo para o seu processamento, tendo sido por isso utilizado apenas um modelo para este exemplo.

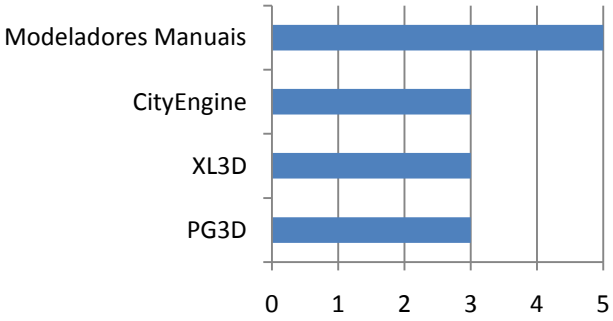

Figura 41 - Avaliação ao nível do detalhe e qualidade dos modelos criados numa gama 0-5

Dada a falta de conhecimento profundo das potencialidades de cada ferramenta neste sentido colocou-as neste comparativo ao mesmo nível, pois acredita-se que o seu potencial não difira muito. No entanto, dadas as funcionalidades um modelador manual é possível atingir níveis de detalhe, qualidade e efeitos muito superiores ao de um modelador procedimental.

Por outro lado, há que sublinhar as capacidades de exportação de dados que o sistema PG3D oferece. Tal permite que, mesmo possuindo capacidades inferiores, possibilita sempre o posterior melhoramento dos seus modelos em ferramentas manuais, dado que consegue converter os seus dados para um formato por eles suportado.

5.4. Fidelidade Visual

O conceito de fidelidade visual refere-se à forma como os ambientes virtuais conseguem recriar ambientes reais, capazes de serem reconhecidos por quem os observe. Até ao momento, devido à falta de fontes de informações detalhadas, tem sido complicado até ao momento conseguir produzir ambientes característicos, que constituem fundamentalmente os factores chave para uma maior fidelidade. No entanto, é ainda possível estabelecer alguma relação com a realidade, através da disposição dos vários elementos que a compõem, da sua altura e da sua forma, como se pode ver na figura seguinte:

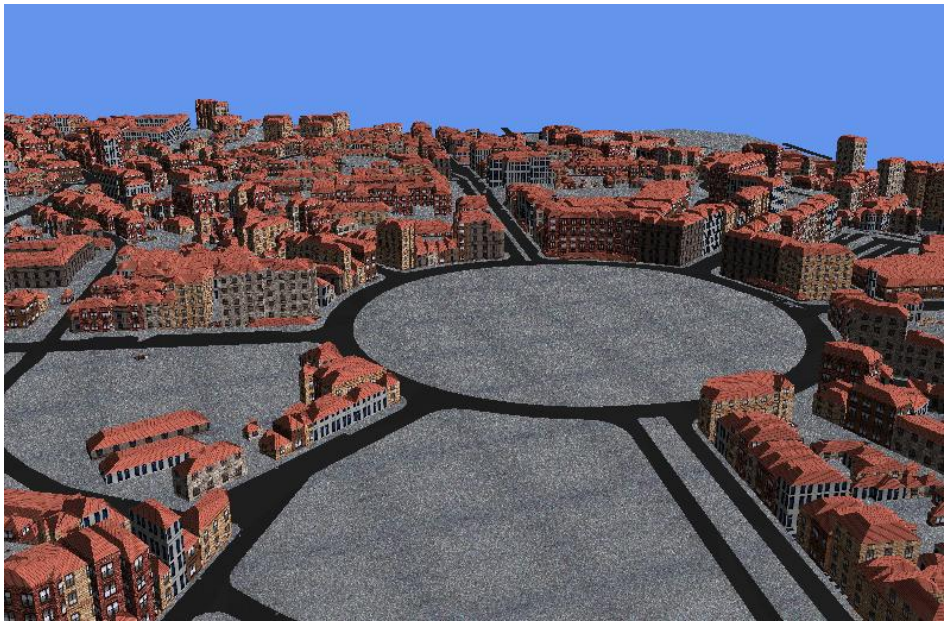

Figura 42 – Modelação da Rotunda da Boavista

Na imagem encontra-se modelada a rotunda da Boavista. Não tendo sido obtida informação mais precisa sobre os jardins ou da zona correspondente à Casa da Música, o seu aspecto é ainda relativamente pobre, mas identificável.

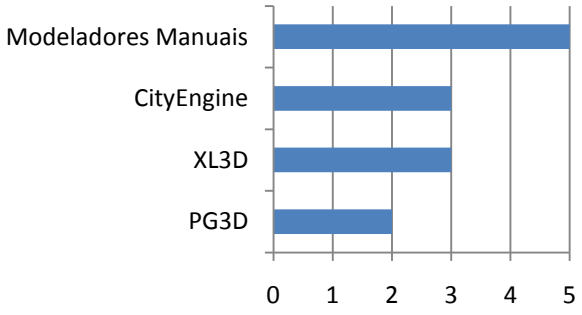

Figura 43 – Avaliação do ao nível da fidelidade visual numa gama 0-5

Em termos de comparação com outras ferramentas, o PG3D ainda não apresenta aparentemente grande concorrência, mas espera-se que, com a obtenção de mais dados se consiga obter resultados mais interessantes. Por essa razão, o comparativo acima refere-se mais aos resultados apresentados pelas ferramentas, e por isso pode não reflectir imperativamente ao seu verdadeiro potencial.

O CityEngine, não tendo como principal objectivo a representação exacta de ambientes urbanos existentes (mas apenas de ambientes urbanos verosímeis), possui um suporte limitado à inclusão de informação geográfica, tendo, por outro lado, a capacidade de geração de edifícios com aspecto bastante realistas (embora não correspondentes à realidade). O XL3D é sem dúvida a ferramenta de modelação procedimental que melhor consegue reproduzir elementos existentes, através da sua integração com informação de fontes de dados em formatos e plataformas diversas. Por outro lado, dado que um utilizador poderá modelar individualmente cada um dos edifícios através de fotografias, de medições ou da sua memória, as ferramentas são as mais capazes de produzir resultados de elevada fidelidade. Evidentemente, a modelação individual é igualmente possível em ferramentas procedimentais, mas tal não reflecte seu intuito original.

5.5. Tempos de Modelação, Carregamento e Exportação

Nesta primeira prototipagem do sistema PG3D, o tempo necessário para os processos de modelação não constitui um dos seus pontos mais fortes, já que o seu maior foco era a implementação do conceito PG3D. Tal também já se poderia esperar de certo modo, dada a sua operação constante sobre a base de dados e, consequentemente, do disco. É conveniente sublinhar também que tal também contribui para um potencial de criação e verificação maior, na medida em que é capaz de gerir áreas de extrema dimensão e de demonstrar o estado em cada iteração. Ainda assim espera-se ainda conseguir reduzir drasticamente os tempos de geração, pela remodelação de algumas estruturas de dados, da optimização de muitas funções e da afinação de muitas definições que o PostgreSQL contém.

A seguinte tabela contém alguns exemplos de instâncias de modelação sobre várias áreas de variada dimensão e complexidade. São indicados igualmente os tempos necessários ao carregamento para visualização e exportação dos dados (no qual é construída a malha poligonal). Estes tempos foram obtidos num portátil com processador Intel Core Duo 2 2,53 GHz, com 3GB de memória, usando o Windows 7 32-bit:

Tabela 2 – Medição do Desempenho do modelador PG3D

Nome	Área (m^2)	Total Shapes Gerados	Total Polígonos Gerados	Tempo Geração	Tempo Carregamento
Casa Complexa (Figura 40)	320	1583	8518	40s	2s
Praça da República (Figura 35	41444	555	14995	45s	2s
Avenida dos Aliados (Figura 37)	88265	1643	22181	2m 32s	5s
Rotunda da Boavista (Figura 42)	926750	11483	188299	15m 10s	1m
Polo São João e Arredores	10377907	88696	1141620	3h 1m 8s	30m 25s

Dada a dimensão e complexidade dos modelos, os seus tempos são razoáveis, e sem dúvida óptimos quando comparados com ferramentas de modelação manual, que requereriam tempos muito superiores, mesmo para modelos com pouco detalhe. Assim sendo, o modelador PG3D parece ser capaz de cumprir o seu objectivo.

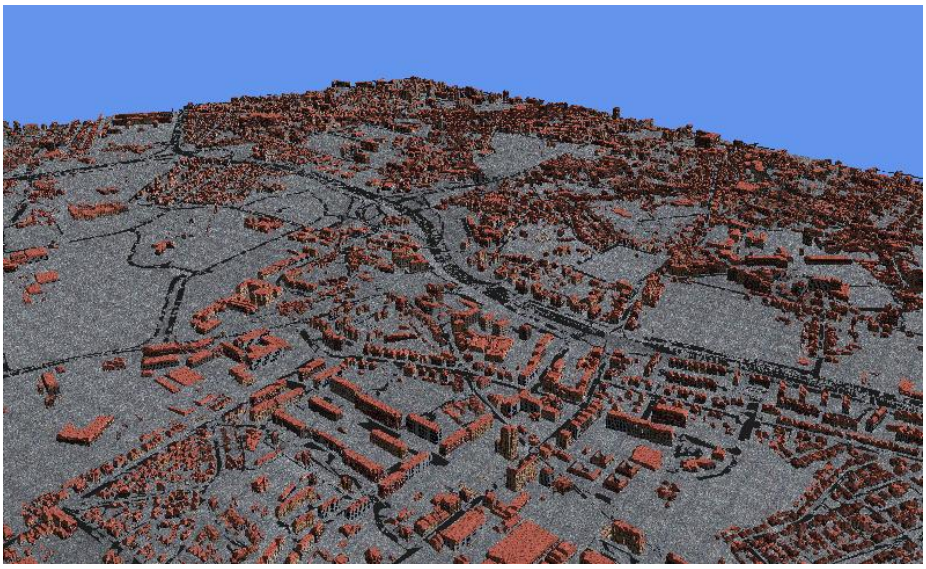

Figura 44 - Modelação da Zona do Pólo S. João

A imagem acima representa representa uma area de dimensão bastante elevada, tendo para isso também requerido bastante mais tempo quer de geração (mais de 3 horas), quer de consulta (30 minutos). É evidente, que para fins de geração, é perfeitamente possível confiar no seu processamento sem preocupação com os limites de memória. Quanto à consulta, é mais compensador que esta seja realizada em parcelas mais pequenas.

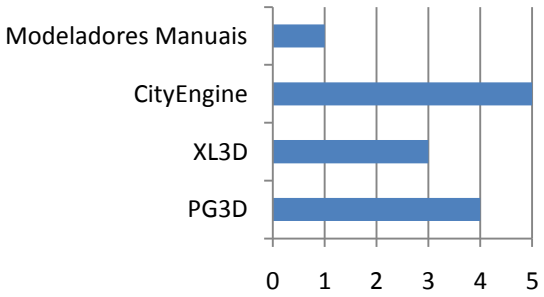

Figura 45 – Avaliação do nível da rapidez de modelação numa gama 0-5

Como é evidente, os modeladores manuais perdem bastante para com os modeladores procedimentais, quanto maior for a dimensão do ambiente em causa. Experiências anteriores com o CityEngine votam a seu favor, enquanto segundo as próprias indicações em [COEL07] o modelador XL3D aparenta requerer de mais tempo.

5.6. Possibilidades de Contextualização Geoespacial

Como referido, um dos maiores potenciais do sistema PG3D é a sua capacidade de as regras se desenvolverem com base nos ambiente envolvente de cada elemento. Tal só é possível se existirem estruturas apropriadas para esses tipos de consulta, algo que se reflecte no PG3D graças à natureza geoespacial da base de dados sobre a qual opera, nomeadamente o PostGIS, e com a indexação dos seus dados. No entanto, esta extensão possui a limitação de as suas operações só funcionarem sobre um nível plano, ou seja, não considerando a coordenada da altura. Como tal, embora seja possível constatar se um elemento tem outro como vizinho, não é possível a confirmação da sua altura, algo que terá de ser realizado num nível diferente. Assim sendo, nesta altura só é possível realizar constatações sobre os vizinhanças e não, por exemplo, de oclusões parciais (para determinar quais exactamente as parcelas das paredes desimpedidas).

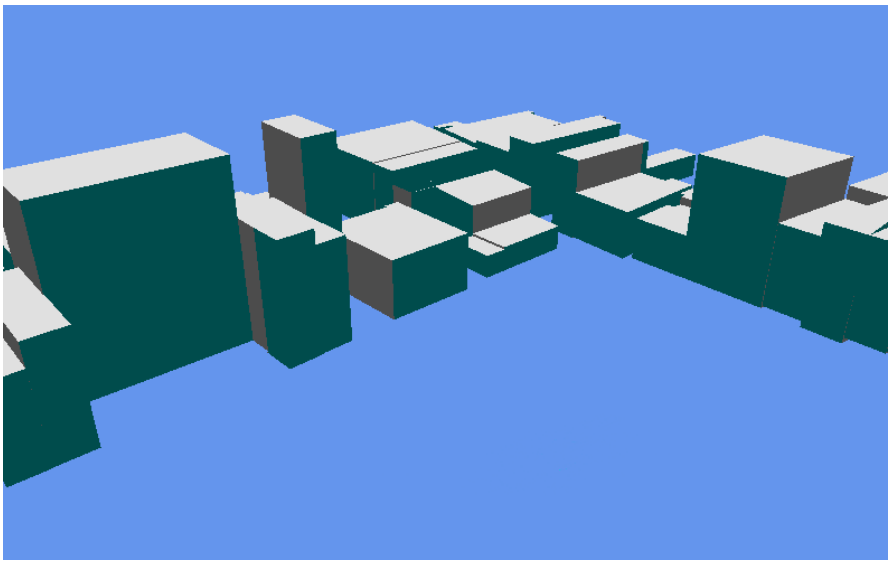

Figura 46 - Blocos dos Edifícios com faces desimpedidas coloridas de verde

Na imagem acima encontram-se coloridas de verde as fachadas dos edifícios da praça da república que não possuem nenhum obstáculo até uma distância de 3 metros à sua frente. Este tipo de constatação confere já a possibilidade de filtrar as fachadas dos edifícios, aplicando a determinadas regras a este grupo mais restrito.

5.7. Aplicação em Jogos de Computador

Dada orientação desta dissertação à modelação procedimental de ambientes urbanos virtuais ao desenvolvimento de jogos de computador, não faria sentido apresentar os seus resultados num contexto de utilização prática. Várias figuras apresentadas neste capítulo (Figura 35,Figura 37,Figura 40,Figura 42,Figura 44) mostram os modelos em XNA, que já é em si uma ferramenta de desenvolvimento de jogos, o que prova a usabilidade dos ambientes virtuais criados em jogos concebidos com esta *framework*.

Adicionalmente, procurou-se aplicar os ambientes gerados pelo menos num jogo, para provar a sua usabilidade e facilidade de integração. Para esse fim, escolheu-se o motor de *Unreal Tournament 3* que, além de ser capaz de operar sobre extensas áreas modeladas, possui ferramentas de criação de níveis bastante poderosas (*UDK - Unreal Development Kit* [EPCG09]).

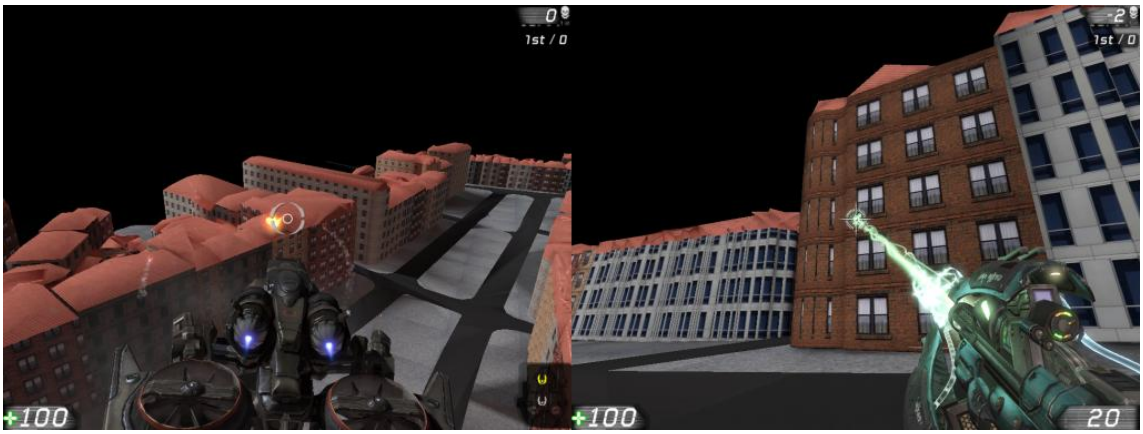

Figura 47 - Utilização dos Ambientes gerados no motor gráfico de Unreal Tournament 3

As imagens acima demonstram a jogabilidade na Avenida dos Aliados, mostrada na Figura 37. Como se pode verificar, a qualidade dos modelos é idêntica, sendo possível a interacção com os mesmos. O processo de adaptação e integração dos modelos com o jogo é relativamente simples, sendo apenas necessário reimportar e converter as texturas para materiais usáveis (algo que o próprio *UDK* não faz automaticamente) e ajustar algumas definições de colisões e iluminação. Para além disso, a utilização dos modelos é quase directa graças à conformidade das malhas exportadas pelo PG3D, definidas em ficheiro *COLLADA*.

5.8. Sumário

Nesta secção foram apresentados alguns resultados conseguidos através do sistema PG3D. Realizaram-se análises ao nível da facilidade de criação, manutenção e utilização dos modelos, verificando-se que o PG3D, apesar de ser capaz de facilmente criar ambientes urbanos virtuais a partir de regras simples, pode ser melhorado através da concepção de meios com mais suporte visual.

Quanto à qualidade dos ambientes criados, constatou-se que o PG3D possui efectivamente bastante capacidade e potencial, restando assim o teste com fontes de dados mais detalhados para confirmar a fidelidade dos mesmos, ou seja, a sua correspondência a ambientes reais.

Foram apresentadas medições temporais para avaliar o desempenho do modelador, tendo-se verificado que apesar de muito vais vantajoso que as ferramentas manuais, poderá ainda ser bastante optimizado, especialmente na tentativa de concepção de ambientes mais detalhados.

De seguida analisou-se os resultados conseguidos no âmbito do desenvolvimento condicional baseado em contextualização geoespacial, constatando-se assim o seu potencial e consequentes vantagens, mas que dadas algumas limitações da plataforma em que foi desenvolvida, poderá ainda ser melhorado.

Por fim foi apresentada a aplicação dos modelos criados num jogo de computador, comprovando assim a usabilidade dos ambientes gerados em jogos de computador.

Capítulo 6

Conclusões e Trabalho Futuro

Este documento, desenvolvido no âmbito de uma dissertação de Mestrado em Engenharia Informática e Computação, na área de Computação Gráfica, consistiu na exposição do conceito e desenvolvimento de uma aplicação de modelação procedimental de ambientes urbanos virtuais, correspondentes a ambientes reais, com vista à sua aplicação em jogos de computador: o Sistema PG3D.

Numa primeira fase deste documento foi realizado o levantamento do estado da arte ao nível da modelação procedimental em jogos de computador. A utilização de meios procedimentais tem-se tornado cada vez mais popular, e cada vez mais um requisito em jogos onde a possibilidade de exploração constitui um elemento central. É necessária a concepção de uma grande quantidade de conteúdos, o que se pode tornar extremamente custoso e, em muito casos, inviável através de meios manuais, especialmente se estes deverem corresponder a elementos reais. Neste sentido, muitos autores têm vindo a trabalhar, produzindo soluções interessantes, particularmente para a criação de ambientes fictícios, mas ainda com dificuldades na reprodução de ambientes reais.

A principal contribuição inovadora, apresentada no capítulo da solução, consta na sua implementação em sistemas de gestão de bases de dados, dotadas de capacidades espaciais e geográficas, que servem como local de armazenamento das fontes de informação relativas a áreas urbanas existentes. É baseando-se nestas que os processos de modelação, operando através de um conjunto de procedimentos internos à base de dados, são capazes de desenvolver ambientes virtuais com teor real, e guardando os seus modelos igualmente em base de dados. Desta forma, o todo o acesso aos dados é feito internamente, reduzindo-se o tempo necessário para a sua realização.

A realização do modelador neste tipo de plataforma induziu a criação da Gramática PG3D, através da qual são definidas as regras de produção e símbolos sobre os quais os processos de modelação operam para a criação dos ambientes urbanos. A dimensão e o detalhe destes dependem assim das fontes de dados disponíveis e da forma como são usados. O seu desenvolvimento, que se realiza de uma forma iterativa, possibilita uma evolução condicional e característica, uma vez que se apoia em informação real e na relação geográfica existente entre os seus elementos.

Com o intuito de complementar e facilitar os meios de interacção e exportação de informação que este tipo de sistemas de armazenamento de dados consegue proporcionar, apresentou-se também uma ferramenta de gestão dedicada à gestão dos parâmetros necessários aos processos de modelação, bem como à exportação da informação dos ambientes gerados para formatos interoperacionais, permitindo a sua utilização em diversas aplicações, entre as quais os jogos de computador.

6.1. Satisfação dos Objectivos

Nesta fase da conclusão da dissertação, será correcto afirmar que todos os objectivos propostos inicialmente foram cumpridos, com resultados bastante satisfatórios.

O primeiro objectivo consistia na investigação do estado da arte ao nível da modelação procedimental, em especial de ambientes realistas, e as suas aplicações em jogos de computador, processo este que se encontra relatado no segundo capítulo deste documento. Este estudo foi extremamente importante, resultando no aproveitamento de múltiplos conceitos propostos por vários autores, nomeadamente os Sistemas L Geoespaciais [COEL07] e as *CGA Shapes* [MULL07].

O segundo e terceiro objectivo consistiam no desenvolvimento de uma aplicação capaz de realizar modelação procedimental dos diversos elementos que compõem os ambientes urbanos, tais como estradas edifícios e mobiliário urbano, e integrando-o com modelos digitais de elevação de terreno. A criação do modelador PG3D vista corresponder a estes objectivos, tendo provado no capítulo de resultados a sua capacidade de modelação dos vários componentes de ambientes urbanos, baseados em informação geoespacial, os quais se encontravam assentes na orografia do terreno (Figura 35,Figura 37,Figura 40,Figura 42,Figura 44).

O último objectivo visava aplicar os modelos gerados num contexto de utilização prática, nomeadamente em jogos de computador. Como foi apresentado no final do capítulo de resultados, a possibilidade de exportação dos dados para o formato COLLADA, incluída na aplicação de gestão PG3D, tornou a sua utilização bastante imediata, tendo-se conseguido empregar os modelos no jogo *Unreal Tournament 3* (ver secção5.7. Aplicação em Jogos de Computador).

6.2. Trabalho Futuro

O Sistema PG3D concebido encontra-se ainda numa fase de protótipo, mas apresentando um enorme potencial no que diz respeito à modelação procedimental para o desenvolvimento de jogos de computador. Assim sendo, é possível desenvolver muitas funcionalidades partindo desta primeira abordagem.

Como se pôde verificar no capítulo de resultados, o modelador PG3D é capaz de criar ambientes de grande dimensão e qualidade, apesar da junção destas duas características ter repercussões fortes sobre os seus tempos de geração. Nesse sentido, uma das primeiras modificações a realizar seria ao nível de optimização dos processos de modelação. Tal poderá ser feito a vários níveis, desde o simples reajustamento das definições da base de dados ou à definição das operações a um nível mais baixo como C em vez de Pl/PgSQL para funções menos dinâmicas e que não requeiram um acesso tão frequente (ou mesmo nenhum) às tabelas de dados. Por outro lado, há que chamar a atenção para o forte poder de rastreamento que o PG3D oferece através da gravação de cada estado e iteração: a possibilidade de escolha entre um modo dedicado a esse fim e um com fins de produção em massa seria igualmente uma alteração possível. Da mesma forma que se fazem diferenciações ao nível do processo por completo, o mesmo poderia existir ao nível das operações, fornecendo estruturas e métodos optimizados para cada tipo de intenção.

Ao nível da fidelidade visual, o PG3D ainda não proporcionou resultados convincentes, pelo que o seu teste sobre fontes de dados mais extensas e pormenorizadas é ainda uma acção a

tomar. Tal irá requerer o contacto com organizações que possuam essas fontes, bem como o tratamento apropriado das mesmas, algo que para já o PG3D possibilita até certo ponto, mas que pode ser desfrutar de um leque de opções superior.

A edição de regras da gramática é sem dúvida um dos pontos que poderá ser mais trabalhado em projectos futuros. Actualmente, a sua edição possui um interpretador relativamente básico, sendo pouco capaz de identificar correctamente e localmente erros de sintaxe e de semântica. Trata-se, por isso, de uma área que carecerá de bastante atenção. Por outro lado, e como também já foi discutido, a definição puramente textual das regras de produção constitui um facto pouco apelativo por detrás da definição de regras de produção, algo que poderá e deverá ser melhorado particularmente ao nível da aplicação cliente. Assim sendo, a visualização em tempo real de uma parcela de dados exemplo poderá constituir um editor mais interactivo e usável.

Ao nível da exportação para modeladores manuais e para jogos de computador, o sistema PG3D já mostrou ser uma ferramenta poderosa, através da possibilidade de exportação para *COLLADA*. Porém, introdução de mais formatos, tais como X3D ou mesmo alguns mais específicos para certos motores de jogos, poderá ser sempre uma mais-valia para a aplicação.

Na matéria de contextualização geoespacial, a implementação actual do Sistema PG3D já possui certamente algumas potencialidades, mas, tal como foi explicado no capítulo de resultados, esta ainda não se realiza em três dimensões, requerendo assim ainda algum trabalho, mas prometendo vir a ser uma das funcionalidades mais interessantes que o PG3D disponibilizará.

Em suma, existem muitas perspectivas de desenvolvimento para o sistema PG3D, a partir do actual trabalho. Contudo, o conceito da sua implementação sobre bases de dados não restringe a plataforma sobre a qual deve actuar, pelo que, de futuro, seria interessante experimentar o seu funcionamento em outros sistemas de gestão de bases de dados com capacidades espaciais. Da mesma forma, a sua aplicação não é exclusiva aos jogos de computador, pelo que o emprego noutras áreas poderia constituir outros projectos no futuro.

Referências

[BENZ09] Benzin. *Procedural Generation in Infinity* 2009; Disponível em: http://forgottenportal.com/infinityblog/2009/01/procedural-generation/.

[BLGE08] Borderlands Guide. *How Do You Make Over Half A Million Guns?* 2008; Disponível em: http://borderlandsguide.com/news/how-do-you-make-over-half-million-guns.

[BRNT08] Bruneton, E. e F. Neyret, *Real-time rendering and editing of vector-based terrains.* Computer Graphics Forum, 2008. **27**(Eurographics 2008): p. 311-320.

[BYRN05] Byrne, E., *Game Level Design*. 2005, Hingham: Charles River Media, Inc.

[CHEN08] Chen, G., et al. (2008) *Interactive Procedural Street Modeling*.

[COEL07] Coelho, A., et al., *Expeditious Modelling of Virtual Urban Environments with Geospatial L-systems.* Computer Graphics Forum, 2007. **26**(4): p. 769-782.

[COEL09] Coelho, A. *Procedural Modeling of Buildings - Automating Virtual World Creation with Procedural Techniques*. 2009.

[CSKY56] Chomsky, N. (1956) *Three Models for the Description of Language*. 113–124.

[DANC07] Danc. *Lost Garden: "Content is Bad"*. 2007; Disponível em: http://lostgarden.com/2007/02/content-is-bad.html.

[DISC09] Discoe, B. *Artificial Terrain Generation*. 2009; Disponível em: http://www.vterrain.org/Elevation/Artificial/.

[DOLN05] Döllner, J. e H. Buchholz, *Continuous level-of-detail modeling of buildings in 3D city models*, in *Proceedings of the 13th annual ACM international workshop on Geographic information systems*. 2005, ACM: Bremen, Germany. p. 173-181.

[DOUL08] Doull, A. *The Death of the Level Designer: Procedural Content Generation in Games*. Ascii Dreams 2008; Disponível em:

http://roguelikedeveloper.blogspot.com/2008/01/death-of-level-designer-procedural.html.

[EPCG09] Epic Games. *Unreal Development Kit*. Unreal Development Kit 2009; Disponível em: http://www.udk.com/.

[FKZR08] Finkenzeller, D. (2008) *Detailed Building Facades*. 58-66.

[GDSN08] Goldstein, H. *GTA IV: Building a Brave New World*. IGN 2008; Disponível em: http://xbox360.ign.com/articles/863/863028p1.html.

[GLZN09] Galuzin, A. *World of Level Design*. 2009; Disponível em: http://www.worldofleveldesign.com/.

[GOGL10] Google. *Google Earth*. 2010; Disponível em: http://earth.google.com/.

[GRSS99] Gross, O. *Pentium III Prefetch Optimizations Using the VTune Performance Analyzer*. 1999; Disponível em: http://www.gamasutra.com/view/feature/3364/pentium_iii_prefetch_optimizations_.php.

[HLWL08] Halliwell, L. *Procedural content generation*. Luke Halliwell's Weblog - Modern Game Development 2008; Disponível em: http://lukehalliwell.wordpress.com/2008/08/05/procedural-content-generation/.

[HPCT00] Hopcroft, J., R. Motwani, e J. Ullman, *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*. 2000: {Addison Wesley}.

[HRDU09] Harvard University. *Digital Elevation Models*. Harvard University 2009; Disponível em: http://www.gsd.harvard.edu/gis/manual/dem/.

[INSW07] Introversion Software. *Procedural Content Generation*. GameCarrerGuide.com 2007; Disponível em: http://www.gamecareerguide.com/features/336/procedural_content_.php.

[JNHU03] Hu, J., S. You, e U. Neumann, *Approaches to Large-Scale Urban Modeling*. IEEE Comput. Graph. Appl., 2003. **23**(6): p. 62-69.

[JNSN09] Johnson, A. *By the Numbers: The Lost Art of Procedural Generation*. The Game Reviews 2009; Disponível em: http://www.thegamereviews.com/articlenav-1639-page-1.html.

[JOEY09] Joey. *Top 7 Open-World Video Game Environments*. ScrawlFx 2009; Disponível em: http://scrawlfx.com/2009/01/top-7-open-world-video-game-environments.

[KLLY] Kelly, G. e H. McCabe (2007) *Citygen: An Interactive System for Procedural City Generation*.

[MECH] Měch, R. e P. Prusinkiewicz, *Visual models of plants interacting with their environment*, in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, ACM. p. 397-410.

[MREL07] Merrell, P., *Example-based model synthesis*, in *Proceedings of the 2007 symposium on Interactive 3D graphics and games*. 2007, ACM: Seattle, Washington. p. 105-112.

[MRTZ96] Martz, P. *Generating Random Fractal Terrain*. Game Programmer 1996; Disponível em: http://www.gameprogrammer.com/fractal.html.

[MULL06] Müller, P., et al., *Procedural Modeling of Buildings*, in *ACM SIGGRAPH 2006 Papers*. 2006, ACM: Boston, Massachusetts. p. 614-623.

[MULL07] Müller, P., et al. (2007) *Image-based Procedural Modeling of Facades*.

[OGCI06] Open Geospatial Consortium Inc., *OpenGIS® Implementation Specification for Geographic Information - Simple feature access - Part 1: Common Architecture*, J.R. Herring, Editor. 2006.

[PCGW09a] PCG Wiki. *What PCG Is*. Procedural Content Generation 2009; Disponível em: http://pcg.wikidot.com/what-pcg-is.

[PCGW09b] PCG Wiki. *Procedural Generation*. Procural Content Generation 2009; Disponível em: http://pcg.wikidot.com/pcg-algorithm:procedural-generation.

[PGDG10a] PostgreSQL Global Development Group. *PostgreSQL: About*. 2010 [citado 2010 15-6-2010]; Disponível em: http://www.postgresql.org/about/.

[PGDG10b] PostgreSQL Global Development Group. *PostgreSQL 8.4.4 Documentation - PL/pgSQL - SQL Procedural Language*. 2010; Disponível em: http://www.postgresql.org/docs/8.4/static/plpgsql.html.

[PGDG10c] PostgreSQL Global Development Group. *PostgreSQL: Documentation: Manuals: PostgreSQL 8.4: Data Types*. 2010; Disponível em: http://www.postgresql.org/docs/8.4/interactive/datatype.html.

[PRCI09] Procedural Inc. *3D Modelling Software for Urban Environments*. Procedural 2009; Disponível em: http://www.procedural.com/.

[PRSH01] Parish, Y.I.H. e P. Müller (2001) *Procedural Modeling of Cities*. 301–308.

[PSKZ94] Prusinkiewicz, P., M. James, e R. Měch, *Synthetic topiary*, in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. 1994, ACM. p. 351-358.

[PSKZ96] Prusinkiewicz, P. e A. Lindenmayer, *The Algorithmic Beauty of Plants*. 1996: Springer-Verlag.

[RDGS09] Rodrigues, R., A. Coelho, e L.P. Reis. *Modelação Expedita de Edifícios Monumentais a Partir de Descrições Textuais*. 2009.

[REMO08] Remo, C. *MIGS: Far Cry 2's Guay On The Importance Of Procedural Content*. Gamasutra 2008; Disponível em: http://www.gamasutra.com/php-bin/news_index.php?story=21165.

[RSCH08] Research, R. *Refractions Research : Home*. 2008; Disponível em: http://www.refractions.net/.

[SLVR06] Silveira, L.G.d. e S.R. Musse, *Realtime Generation of Populated Virtual Cities*, in *Proceedings of the ACM symposium on Virtual reality software and technology*. 2006, ACM: Limassol, Cyprus. p. 155-164.

[STNY72] Stiny, G. e J. Gips. *Shape Grammars and the Generative Specification of Painting and Sculpture*. in *Information Processing '71*. 1972.

[SWTR04] Sweetser, P. e D.M. Johnson (2004) *Player-Centred Game Environments: Assessing Player Opinions, Experiences and Issues*. 321-332.

[TPKT04] .theprodukkt. *Kkrieger*. 2004; Disponível em: http://www.theprodukkt.com/kkrieger.

[TSPR09] Total Spore. *Spore: Procedural Generation*. Total Spore 2009; Disponível em: http://totalspore.com/game-information/procedural-generation/.

[WATS08] Watson, B., et al., *Procedural Urban Modeling in Practice.* IEEE Computer Graphics and Applications, 2008. **28**: p. 18-26.

[WNKA03] Wonka, P., et al. (2003) *Instant Architecture*. **22**, 669–677.

[YOUN06] Young, S. *The Procedural World, Part 1*. Twenty Sided 2006; Disponível em: http://www.shamusyoung.com/twentysidedtale/?p=537.

[YOUN09a] Young, S. *Fuel: Defining Procedural*. Twenty Sided 2009; Disponível em: http://www.shamusyoung.com/twentysidedtale/?p=5134.

[YOUN09b] Young, S. *The Future is Procedural*. The Escapist 2009; Disponível em: http://www.escapistmagazine.com/articles/view/columns/experienced-points/6418-The-Future-is-Procedural.

[ZHOU] Zhou, H., et al., *Terrain Synthesis from Digital Elevation Models.* IEEE Transactions on Visualization and Computer Graphics, 2007. **13**(4): p. 834-848.

Anexo A

Manual de Utilização da Aplicação de Gestão PG3D

Neste anexo pretende-se realizar uma visita guiada simples sobre a aplicação de gestão PG3D para mostrar os vários menus e opções, demonstrando os procedimentos necessários para a criação e visualização de um ambiente urbano virtual.

Ao iniciar a aplicação surge a seguinte janela de gestão de projectos.

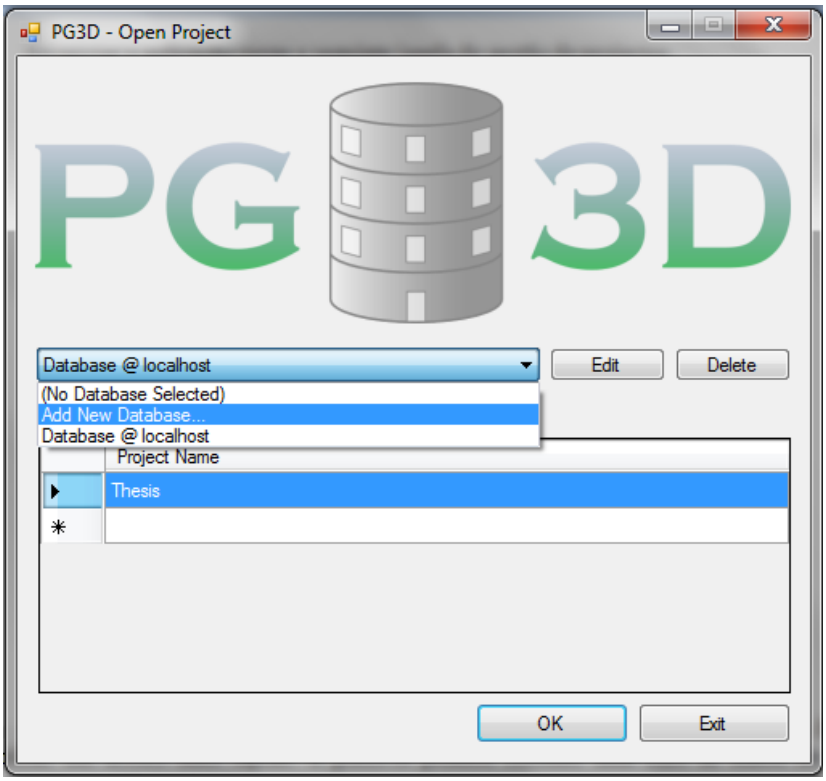

Figura 48 – Janela de gestão de projectos

A caixa de selecção existente contém uma lista de bases de dados já acedidas, para possibilitar um acesso mais rápido. É possível adicionar a definição de novas bases de dados através da escolha da hipótese “Add New Database”, pelo que surgirá a imagem a seguir.

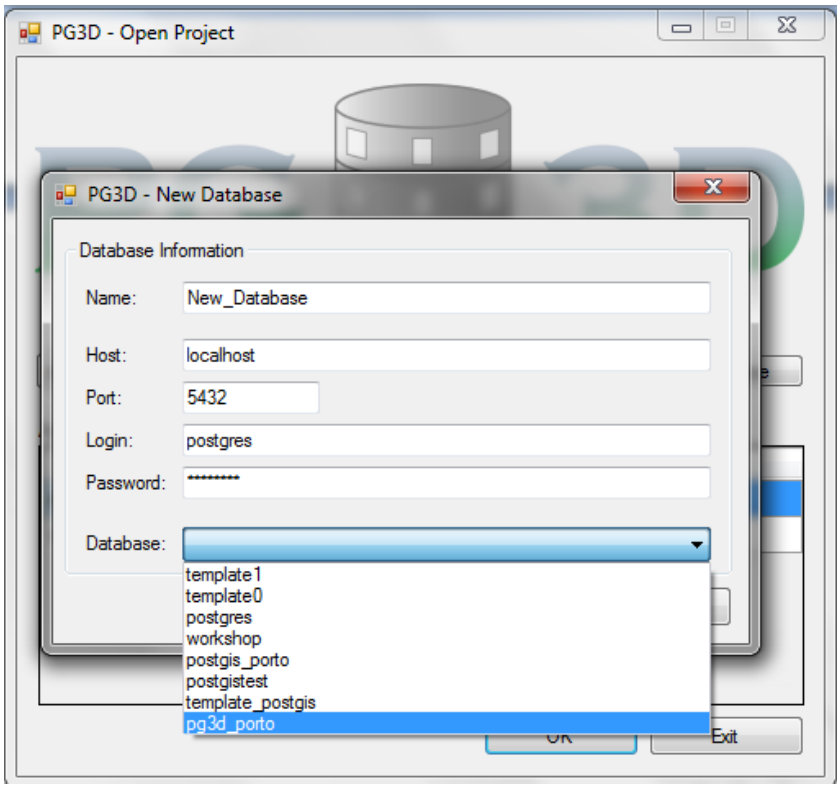

Figura 49 – Formulário para definição de novo acesso

Neste formulário pode ser escolhido o endereço ou nome do servidor que contém a base de dados, a porta, bem como o nome de utilizador e palavra-passe. Depois de preenchido estes campos, é possível consultar com o controlo de selecção as bases de dados que existem no servidor, e escolher uma delas. No entanto, se ela não possuir suporte para PG3D, não poderá ser usada.

Figura 50 – Caso de base de dados não suportada

É importante também escolher um nome que identifique a ligação, antes que as definições possam ser gravadas.

Uma vez aceites os dados inseridos, a lista será actualizada, sendo possível escolher o nome do elemento criado. Este pode ser eliminado ou editado posteriormente através dos botões ao lado. Ao seleccionar o elemento, aparecerá na listagem em baixo a lista de projectos disponíveis. Ao escrever um novo nome na linha vazia ou ao eliminar uma linha existente será adicionado e removido um projecto com esse nome, respectivamente. Não é possível, contudo, editar os nomes dos projectos uma vez criados.

Tendo seleccionado uma base de dados e um projecto, é possível seleccionar o botão "OK" para abrir o projecto, pelo que surgirá a janela de gestão de projecto, como mostrado na figura que se segue:

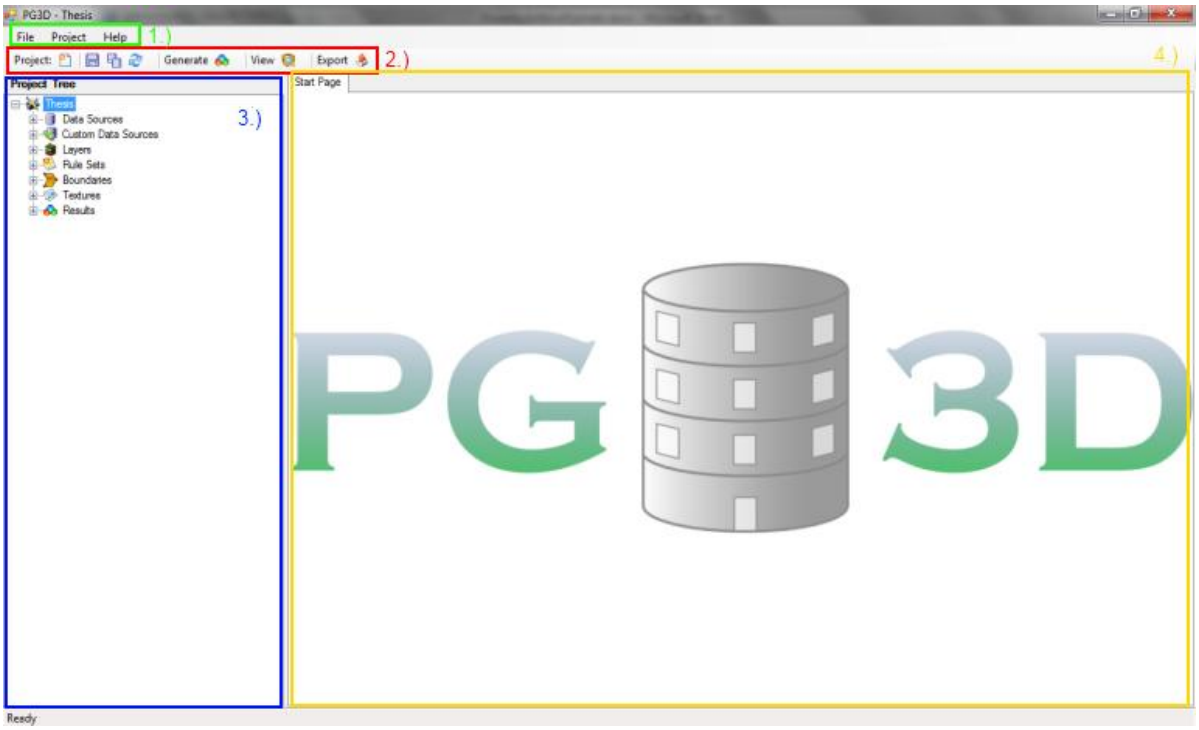

Figura 51- Janela de Gestão de projecto

Na imagem encontram-se marcadas com cores e números identificando as várias partes da interface. No canto superior esquerdo, na zona marcada a verde, encontra-se o menu de topo, que permite o fecho do projecto (fazendo surgir a janela anterior) ou da aplicação; a criação de novos objectos, bem como a gravação de alterações dos mesmos; a actualização dos objectos da base de dados e o acesso ao menu de detalhes do programa.

Abaixo, marcado pela caixa vermelha, encontra-se a barra de ferramentas, possibilitando algumas das possibilidades do menu de topo, tais como a criação de novos objectos e a gravação destes. Contêm igualmente os botões que permitem a geração, visualização e exportação de dados modelados.

A terceira zona, na região lateral esquerda, marcada a azul, encontra-se a árvore de projecto contendo todos os elementos que podem ser manipulados: fontes de dados públicas, fontes de dados personalizados, *layers*, ficheiros de regras, *boundaries*, texturas e resultados. À medida que vão sendo criados novos objectos, serão aqui dispostos, sendo que o duplo-clique sobre cada elemento causará o aparecimento de um separador no seu respectivo gestor na zona marcada a amarelo. Este possuirá informação sobre o objecto, podendo, em certos casos, ser editada.

Ao carregar sobre um elemento que descenda do nó referente às bases de dados públicas ou personalizadas (denominado “Data Sources” e “Custom Data Sources”, respectivamente, na árvore de projecto), é mostrado um separador semelhante ao seguinte:

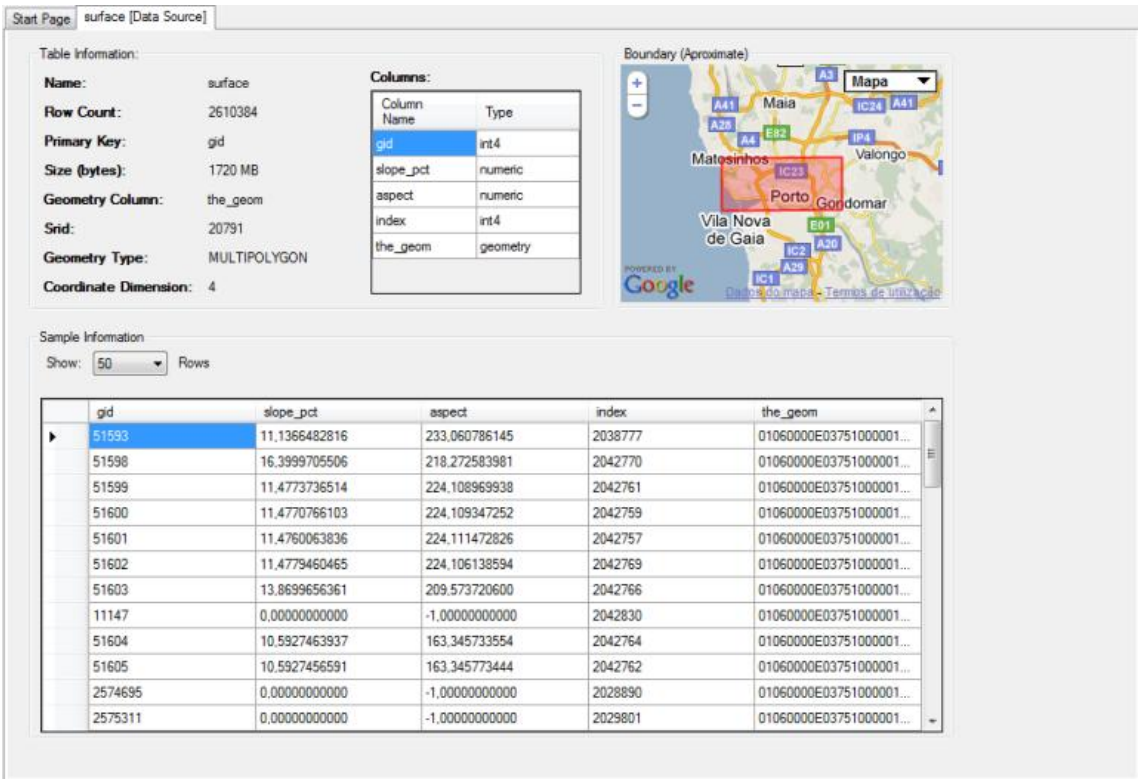

Figura 52 – Separador com informação de uma tabela de dados

Aqui são mostrados os vários dados referentes à tabela escolhida: o seu nome; o número de entradas; a sua chave primária, o seu tamanho em disco, a coluna em formato geográfico (bem como o tipo e dimensão da geometria), o seu identificador de referência espacial, as colunas que o descrevem bem como uma amostra dos dados da tabela. Encontra-se também, num controlo no canto superior direito, um mapa que recorre ao Google Maps (possibilitando assim a ampliação e visualização mais interactiva) para mostrar uma aproximação da área abrangida pelos dados.

É possível criar novas fontes de informação a partir das tabelas existentes. Para tal, é necessário aceder ao menu de topo, na opção "Project"-> "New Object" ou à barra de ferramentas, no primeiro botão, que fará surgir a seguinte janela:

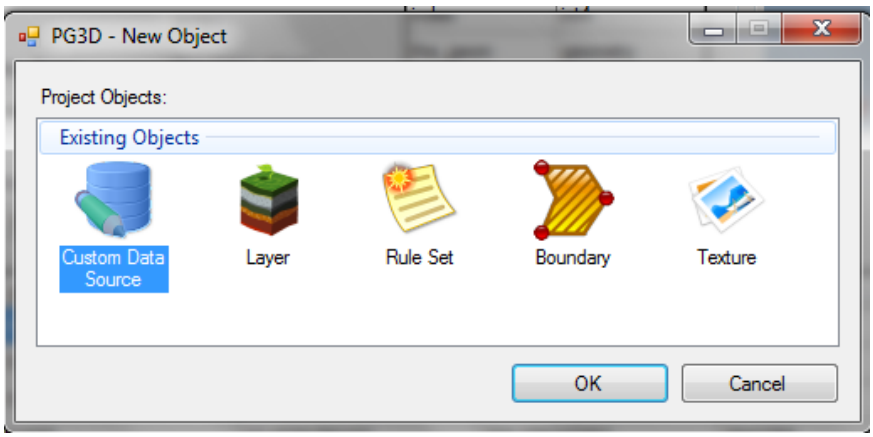

Figura 53 – Janela de criação de novo objecto

Ao seleccionar a primeira opção “Custom Data Source” e premindo “OK”, surge a seguinte janela que permite a criação de novas fontes:

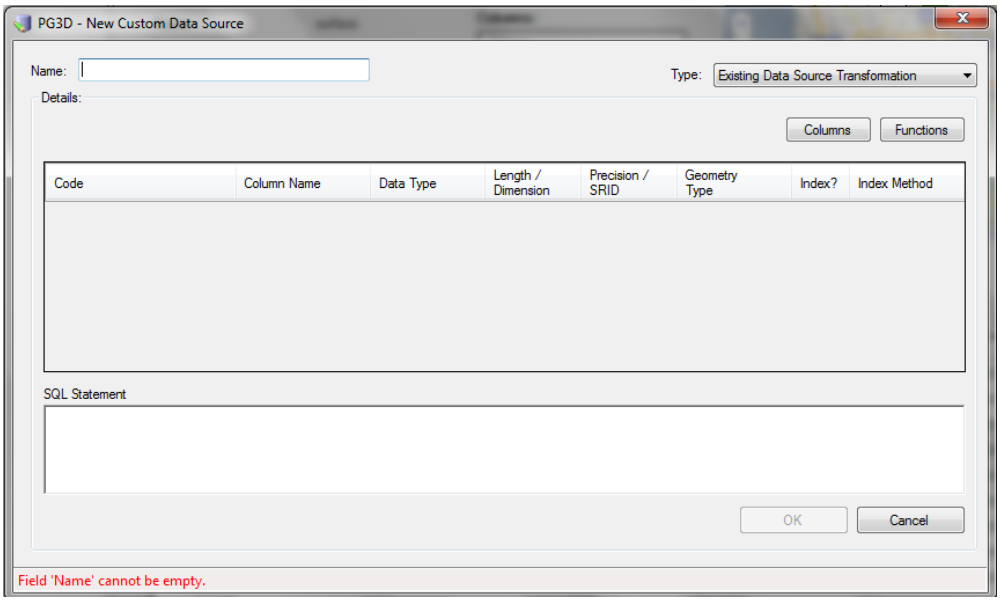

Figura 54 – Janela de Criação de novas fontes de dados

Como em todas as janelas de criação da interface PG3D, procura-se reduzir a recorrência a caixas de diálogo, normalmente usadas para indicar elementos em falta ou a ocorrência de erros. Em vez delas, existe na zona inferior de cada janela a mensagem em vermelho que indica o estado actual do preenchimento do formulário, indicando os campos em falta ou dados inválidos. Se existir pelo menos um problema no formulário, o botão de aceitação (“OK”) não será activado.

A criação de novas fontes surge normalmente da transformação de dados existentes. Como forma de facilitar a consulta desses, existe um botão chamado “Columns” que permite consultar todas as colunas de todas as tabelas com dados.

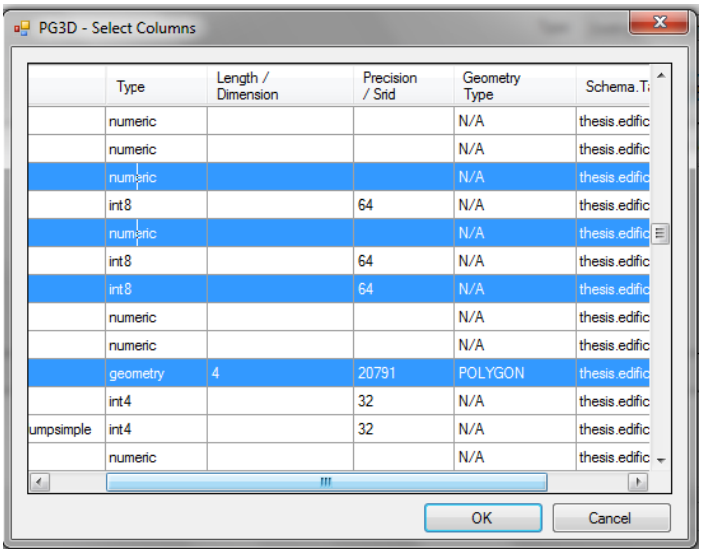

Figura 55 – Janela com lista de colunas

Esta janela possui várias informações sobre cada coluna, e possibilita a escolha de mais do que uma de cada vez. Uma vez seleccionadas e aceites, são adicionadas à lista da janela anterior. Em baixo, o código SQL referente a essas colunas é gerado automaticamente.

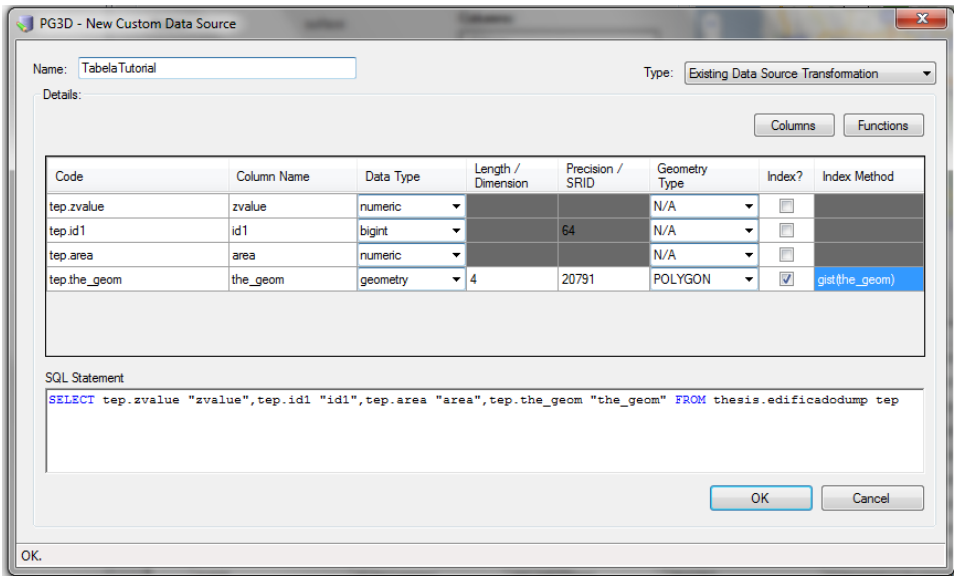

Figura 56 – Adição de colunas e geração automática de código SQL

Uma vez presentes na lista, é possível alterar os seus nomes, valores ou definir uma forma de indexar a coluna, sendo apenas preciso indicar qual a função de indexação pretendida. As alterações realizadas nesta fase também serão reproduzidas na expressão SQL em baixo. Na primeira coluna da listagem, é possível aplicar operações ou funções, pelo que o acesso a uma referência é igualmente possível. O botão "Functions" conduz a uma listagem de funções que podem ser utilizadas, sendo indicados os nomes e argumentos necessários. Uma vez que não é possível obter descrições das funções automaticamente, é conveniente que o utilizador possua algum conhecimento prévio do seu funcionamento.

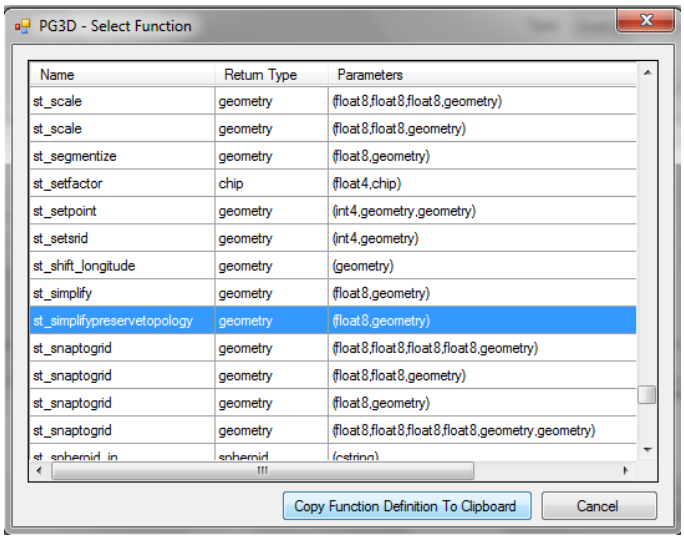

Figura 57 – Listagem de funções

Neste caso, a selecção de elementos da lista funciona de forma diferente, na medida é que é apenas possível copiar o protótipo da função para a área de transferência, para ser depois utilizada em qualquer dos campos de código da janela anterior.

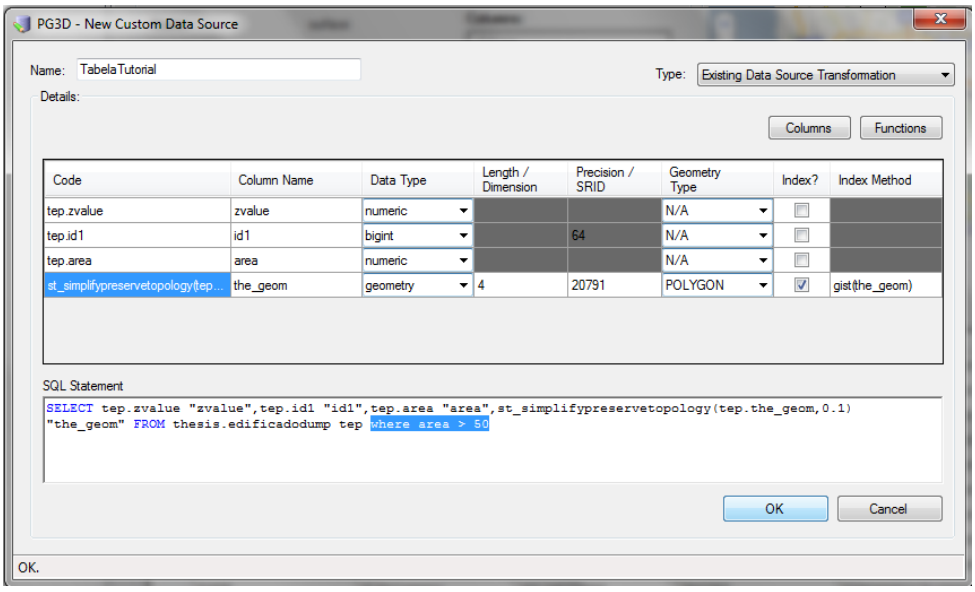

Figura 58 – Adição de funções e de código SQL manualmente

Como se pode ver na imagem, foi aplicada, por um lado, a função à última coluna com base na informação obtida e por outro código SQL manualmente, provando assim que é possível aplicar filtros e condições sobre os dados. Uma vez aceites todas estas definições, surge uma caixa que indica o progresso de criação da fonte de dados (que poderá eventualmente demorar algum tempo):

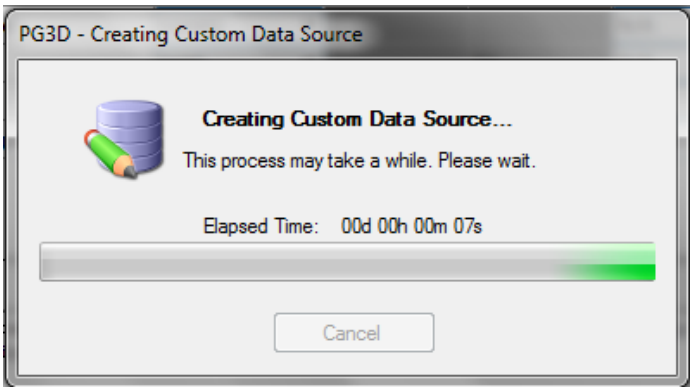

Figura 59 – Caixa de progresso da criação da tabela

Finalizado o processo, é possível consultar e utilizar a tabela de dados criada da mesma forma que as restantes.

A interface PG3D possibilita também a definição de limites ou *boundaries*, que podem ser utilizadas na concepção de regras de produção. Para criar uma nova *boundary* é necessário aceder menu da Figura 53 e escolher a opção “Boundary”. A janela que surge daí é a seguinte:

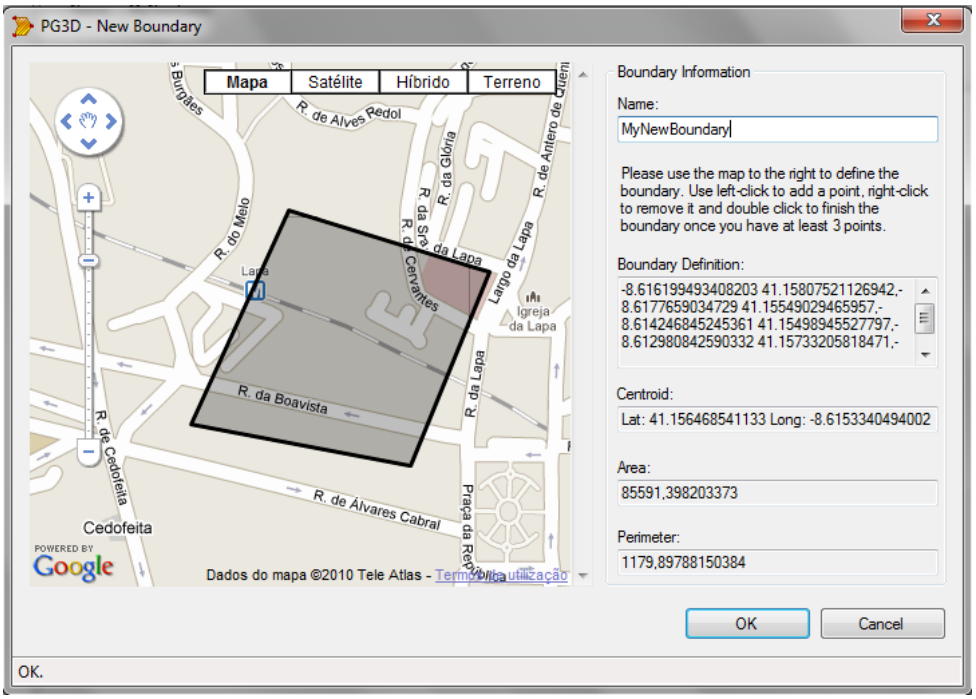

Figura 60 – Janela de criação de nova *boundary*

A criação é bastante interactiva, na medida em que recorre ao Google Maps, que possibilita a visualização e o desenho de polígonos e áreas. Através da utilização do rato é possível definir um conjunto de rectas (recorrendo ao botão esquerdo para definir uma nova linha e o direito para voltar atrás) que serão fechadas ao fazer duplo-clique. É criada uma área mais escura que define a área definida pela *boundary*, sendo os dados dela, tais como as suas coordenadas, o seu centro, a área e o perímetro. É conveniente que as linhas criadas não se intersectem, pois tal pode causar problemas na utilização da *boundary*. Tendo-lhe atribuído um nome, esta poderá ser aceite e utilizada de futuro.

Outro objecto que pode ser criado é a *layer* através do menu já descrito na Figura 53. Uma vez escolhida essa opção no menu, surge uma simples janela, onde o nome deve ser escolhido:

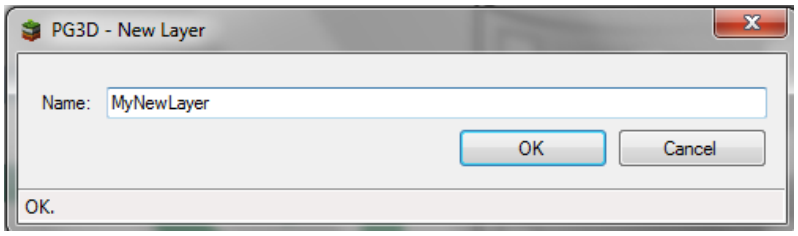

Figura 61 – Janela de criação de nova *layer*

A *layer* é um elemento que pode fazer parte do axioma dos processos de modelação, sendo por isso responsável pela importação de informação. O seu código é semelhante à de uma regra de produção, mas sem predecessor. Assim sendo, a edição de uma *layer* é feita através de um editor de texto que pode ser acedido através do duplo-clique sobre a *layer* na árvore de projecto. O separador que surge é o seguinte:

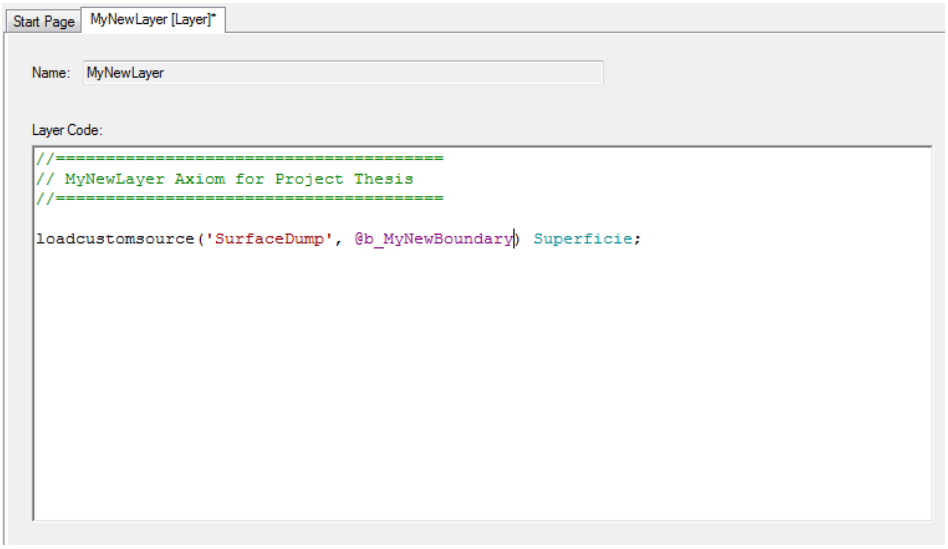

Figura 62 – Separador de edição de *layer*

A imagem acima mostra o código utilizado para o carregamento de informação de uma tabela de superfície, relativa à *boundary* criada no exemplo acima. A referência de funções e operações a utilizar nesta fase pode ser encontrada no Anexo B.

É possível igualmente importar texturas, através do menu já mencionado. A sua importação é feita na seguinte janela:

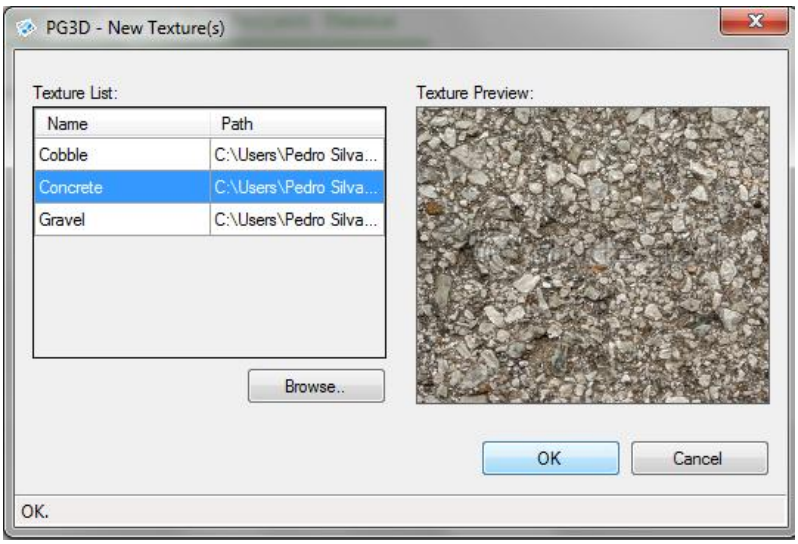

Figura 63 – Janela de importação de texturas

O botão "Browse…" permite a procura de múltiplas texturas no disco do utilizador. Uma vez escolhidas, serão adicionadas à listagem que se pode ver na figura acima, podendo depois os seus nomes e caminhos serem adicionados, se desejado. A escolha de uma textura despoleta a sua visualização na parte direita da janela.

Por fim, a criação de conjuntos de regras pode ser realizado através da mesma janela que para os anteriores casos. É apenas necessário indicar o nome, tal como para as *layers:*

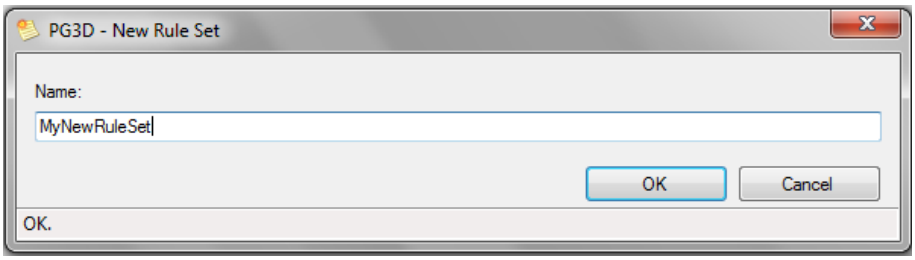

Figura 64- Janela de criação de ficheiros de regras

A sua forma de edição é semelhante a uma *layer*, através de um editor de texto:

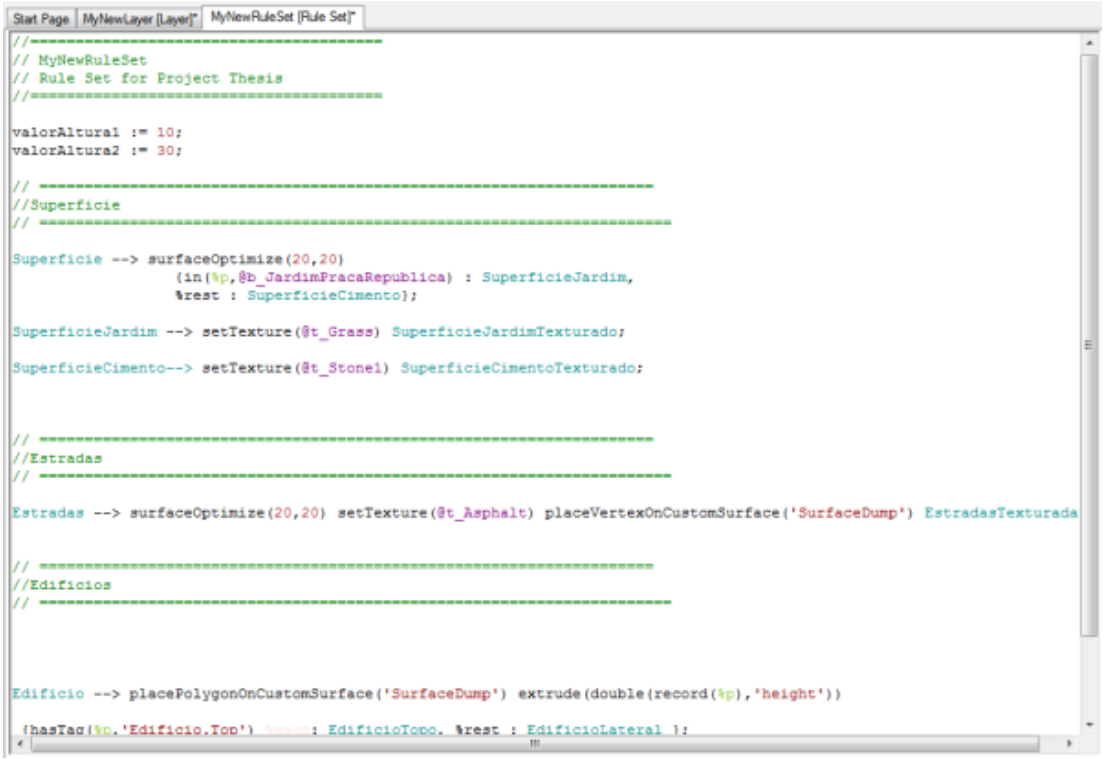

Figura 65 – Edição das regras de produção

O editor possui capacidades de realce de palavras e expressões especiais, sendo assim fácil distinguir os vários elementos na composição das regras de produção.

A definição das regras é independente do espaçamento, da tabulação e da mudança de linha. É possível assim alterar a disposição das expressões conforme for de maior agrado ao utilizador. A aplicação de comentário, porém, funciona ao nível de linha - iniciado pelos caracteres '//', podendo ser iniciado em qualquer parte da linha mas apenas comentando os conteúdos depois da linha, ao estilo das linguagens de programação da família C ou Java.

A definição das regras necessita de ser realizada imperativamente antes da sua utilização. Seguindo a definição da gramática PG3D, as variáveis são referenciadas pela arroba ('@'), tal como as texturas e os limites são referenciados pelo '@t_' e '@_b', respectivamente.

No processo de alteração de ficheiros de regras ou layers, o separador fica marcado com um asterisco, indicando que as últimas modificações ainda não foram gravadas (como se pode ver na última figura). Para o fazer, deve-se recorrer ao menu de topo, nas opções em "Project"-> "Save Object" ou "Save All Objects" para gravar o separador actual ou todos os abertos, respectivamente. As mesmas opções estão disponíveis através da barra de ferramentas, nos ícones com disquetes.

Tendo criado *layers* e declarado regras, é possível gerar modelos. A opção encontra-se disponível na barra de ferramentas, a seguir à indicação de "Generation".

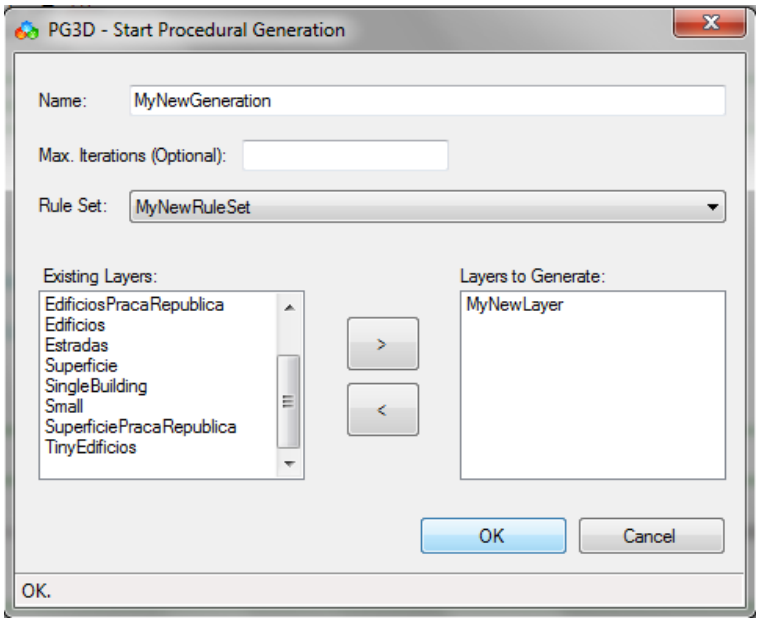

Figura 66 – Menu de geração

A janela que surge contém uma lista com os vários conjuntos de regras e *layers* criadas, sendo possível escolher apenas um para o primeiro caso, e múltiplas para o segundo. Por defeito, a geração decorre até não existirem mais regras que consigam processar as *shapes* criadas, mas é possível aplicar também um limite de iterações. Por defeito, este campo encontra-se vazio. É, no entanto, essencial a atribuição de um nome à geração, para que possa ser identificada mais tarde. Ao aceitar os dados actuais, é mostrada uma janela de progresso que, apesar de não ser capaz de determinar o tempo restante, mostra o tempo que já decorreu:

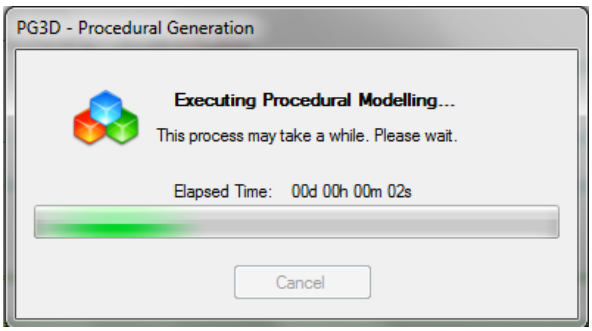

Figura 67 – Janela de progresso dos processos de modelação procedimental

Uma vez terminado o processo, é adicionada uma entrada à árvore de projecto referente aos resultados criados, que pode ser consultada num separador:

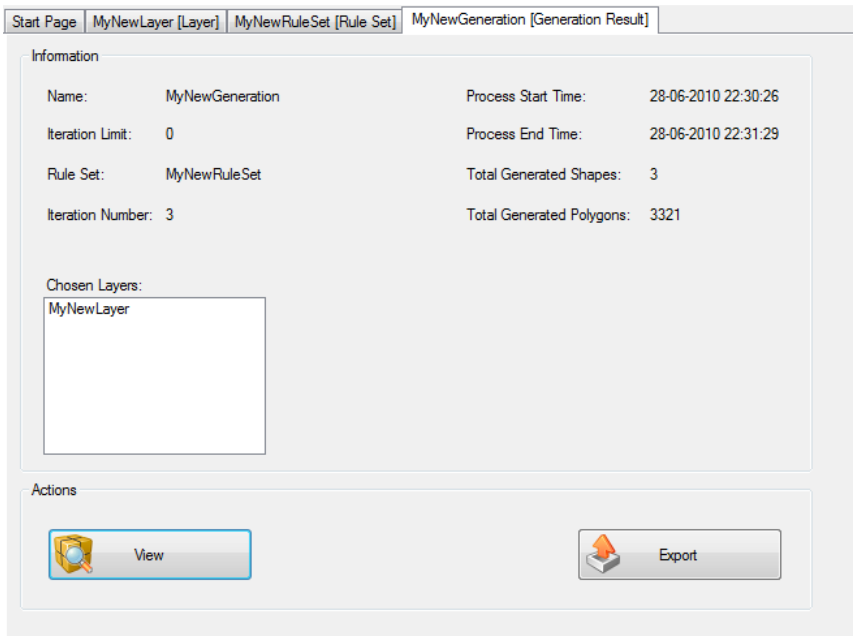

Figura 68 – Separador com informação de uma instância de modelação

Aqui é possível consultar vários dados sobre a geração, tais como o tempo demorado, o número de *shapes* e polígonos criados, e o número de iterações que decorreu. Em baixo encontram-se dois botões que permitem a visualização e exportação dos dados desta geração. Estas opções também são acessíveis a partir da barra de ferramentas.

Ao pretender-se visualizar os dados, surge a seguinte janela:

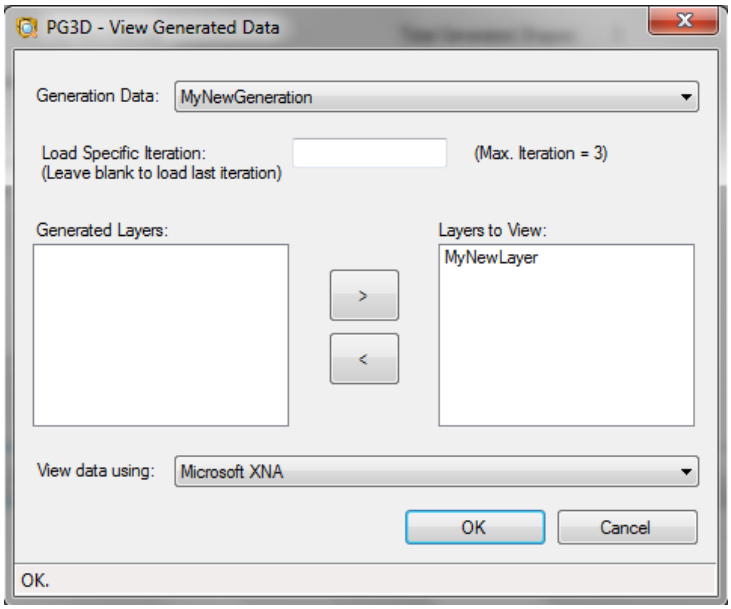

Figura 69 – Janela de para visualização dos dados

Esta permite a escolha de apenas uma das *layers* (no caso acima existe apenas uma, por isso apenas essa foi escolhida), ou mesmo de apenas o resultado obtido até uma determinada

iteração, permitindo assim consultar o desenvolvimento da geração em cada passo. Como plataforma de visualização, apenas se encontra disponível o Microsoft XNA, mas a própria interface já se encontra preparada para futuras adições nesta área. Os resultados podem levar também algum tempo a carregar, pelo que surgirá uma caixa de progresso, semelhante à da Figura 67. Uma vez terminado, bastará carregar o botão "View" para fazer surgir a seguinte janela.

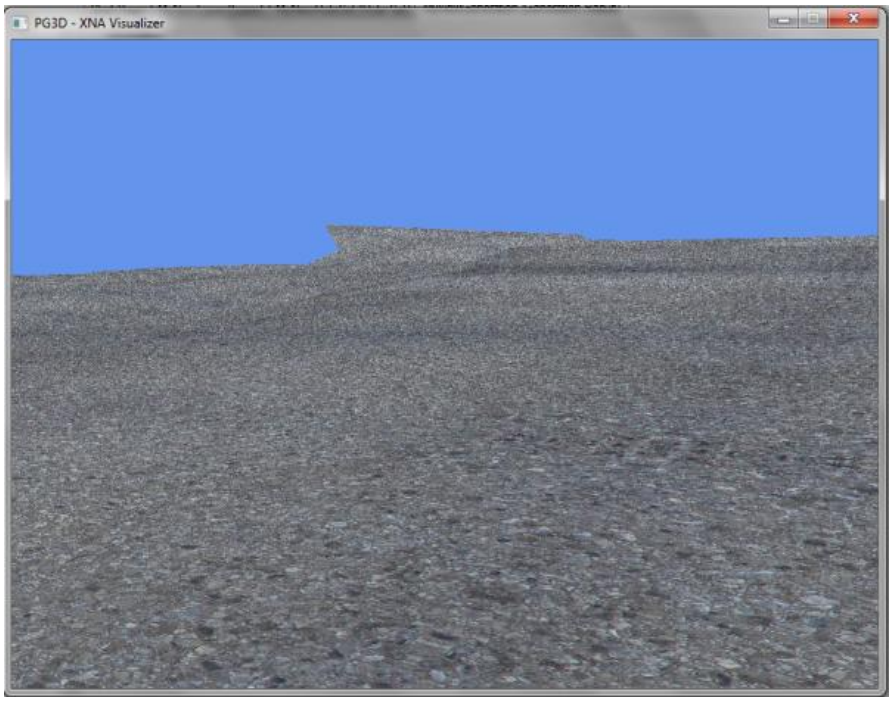

Figura 70 – Janela de Visualização da Cena em XNA

A cena acima é simples, tal como foi o exemplo testado.

Por fim, resta referir a funcionalidade de exportação, que possui fortes semelhanças com a de visualização. Acessível também a partir da barra de ferramentas, mostra a seguinte janela:

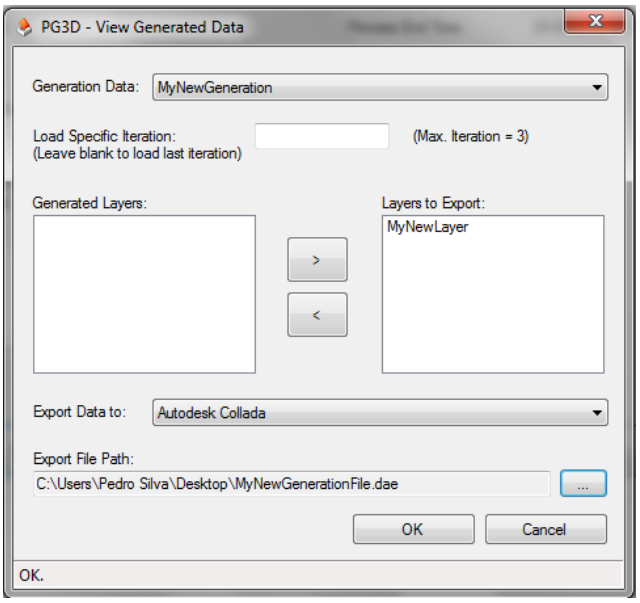

Figura 71 – Janela de exportação de dados gerados

As opções são semelhantes ao caso anterior, com a excepção de neste caso ser necessária a indicação do ficheiro para onde se deve exportar, para além da escolha do formato. De momento o único suportado é o COLLADA, mas esperam-se futuras adições no futuro. Para começar o processo de exportação, resta pressionar o botão "OK", que despoletará o processo de exportação. Como para o caso de geração e visualização, poderá ser necessário esperar algum tempo, pelo que será mostrada uma janela de progresso durante esse período.

Anexo B

Referência de Funções PG3D

Esta referência pretende apenas referir as funções necessárias para a utilização nas Gramáticas PG3D, não constando por isso as funções de sistema ou intermédias, que não devem ser de qualquer das formas acedidas pelos utilizadores.

Como referido, os parâmetros que aceitam as referências a %s, %p e %v (estão declaradas com o sufixo _s, _p e _v, respectivamente. Relembra-se que as referências de nível inferior aceitam referências de nível superior, ou seja, é possível utilizar as referências às *shapes* em funções _v e _p, bem como polígonos nas funções _v.

Funções de Carregamento de Dados

loadsource(*tableName_s* text)
Carrega todos os dados de uma tabela pública de nome *tableName_s*.

loadsource(*tableName_s* text, *boundary* geometry)
Carrega de uma tabela pública os dados referentes ao limite definido em *boundary*.

loadcustomsource(*tableName_s* text)
Carrega todos os dados de uma tabela personalizada (ou seja, criada dentro do projecto) de nome *tableName_s*.

loadcustomsource(tableName_s text, *boundary* geometry)
Carrega de uma tabela personalizada (ou seja, criada dentro do projecto) os dados referentes ao limite definido em *boundary*.

Funções de Modelação

vector3 rotateX(*v1* vector3, *angle* double precision)
Roda o vector em torno do eixo X pelo ângulo definido no parâmetro. O ângulo deve ser definido em radianos.

vector3 rotateY(*v1* vector3, *angle* double precision)

Roda o vector em torno do eixo Y pelo ângulo definido no parâmetro. O ângulo deve ser definido em radianos.

vector3 rotateZ(*v1* vector3, *angle* double precision)
Roda o vector em torno do eixo Z pelo ângulo definido no parâmetro. O ângulo deve ser definido em radianos.

slicex (*xvalues_p* double precision[], *tags_p* text[])
Corta todos os polígonos da *shape* em várias partes, de tamanhos definidos em *xvalues_p*, ao longo do eixo x do seu âmbito, e atribui-lhes as etiquetas definidas em *tags_p*.

slicex (*xvalues_p* double precision[], *tags_p* text[], *condition_p* boolean)
Corta os polígonos da *shape*, que correspondam à condição condition_p, em várias partes, de tamanhos definidos em *xvalues_p*, ao longo do eixo x do seu âmbito, e atribui-lhes as etiquetas definidas em *tags_p*.

slicey (*xvalues_p* double precision[], *tags_p* text[])
Corta todos os polígonos da *shape* em várias partes, de tamanhos definidos em *xvalues_p*, ao longo do eixo y do seu âmbito, e atribui-lhes as etiquetas definidas em *tags_p*.

slicey (*xvalues_p* double precision[], *tags_p* text[], *condition_p* boolean)
Corta os polígonos da *shape*, que correspondam à condição condition_p, em várias partes, de tamanhos definidos em *xvalues_p*, ao longo do eixo y do seu âmbito, e atribui-lhes as etiquetas definidas em *tags_p*.

surfaceOptimize ()
Remove os vértices repetidos de uma *shape*, correspondente a uma superfície, e define as coordenadas de mapeamento (u,v) das texturas a cada um dos pontos, de forma a que a textura fique esticada e cuba toda a superfície.

surfaceOptimize (*xvalue_p* double precision, *yvalue_p* double precision)
Remove os vértices repetidos de uma *shape*, correspondente a uma superfície, e define as coordenadas de mapeamento (u,v) das texturas a cada um dos pontos, de forma a que a aplicação da textura tenha sempre o mesmo tamanho e igual ao indicado nos argumentos.

clearAllTags()
Elimina todas as *tags* aplicadas a todos os polígonos da shape.

clearAllTags (*condition_p* boolean)
Elimina todas as *tags* aplicadas aos polígonos da shape que correspondam à condição indicada.

clearAllTags (*tags_p* text[])

Elimina as tags declaradas de todos os polígonos da shape.

clearAllTags (*tags_p* text[], *condition_p* boolean)
Elimina as tags declaradas dos polígonos da shape que correspondam à condição indicada.

setTexture (*texture_s* text)
Aplica uma textura a uma shape.
setUV ()
Realiza o mapeamento das texturas em todos os polígonos da *shape*, esticando a textura para que cada unidade de textura corresponda ao tamanho de cada polígono.

setUV (*condition_p* boolean)
Realiza o mapeamento das texturas nos polígonos da *shape* aos quais a condição se aplique, esticando a textura para que cada unidade de textura corresponda ao tamanho de cada polígono.

setUV (*xvalue_p* double precision, *yvalue_p* double precision)
Realiza o mapeamento das texturas em todos os polígonos da *shape*, esticando a textura para que cada unidade de textura corresponda ao tamanho definidos nos argumentos.

setUV (*xvalue_p* double precision, *yvalue_p* double precision, *condition_p* boolean)
Realiza o mapeamento das texturas nos polígonos da *shape* aos quais a condição se aplique, esticando a textura para que cada unidade de textura corresponda ao tamanho definidos nos argumentos.

extrude (*height_p* double precision)
Realiza uma extrusão de todos os polígonos contidos na *shape* pelo valor indicado no parâmetro.

extrude (*height_p* double precision, *condition_p* boolean)
Realiza uma extrusão dos polígonos da *shape* que correspondam a à condição, pelo valor indicado no parâmetro.

extrudeTaper (*height_p* double precision, *taperValue_p* double precision)
Realiza uma extrusão de todos os polígonos contidos na *shape*, pela altura definida em *height_p* e com afunilamento definido em *taperValue_p*.

extrudeTaper (*height_p* double precision, *taperValue_p* double precision, *condition_p* boolean)
Realiza uma extrusão de todos os polígonos da *shape* que correspondam a à condição, pela altura definida em *height_p* e com afunilamento definido em *taperValue_p*.

placeVertexOnSurface (*tableName_s text)*

Assenta a *shape* sobre a os dados de um terreno, definido numa tabela pública com nome *tableName_s*, com precisão aos seus vértices (ou seja, com cada vértices assente sobre a superfície).

placeVertexOnSurface (tableName_s text, condition_p boolean)
Assenta a *shape* sobre a os dados de um terreno, definido numa tabela pública com nome *tableName_s*, com precisão aos seus vértices (ou seja, com cada vértices assente sobre a superfície) para os quais a condição seja verdadeira.

placeVertexOnCustomSurface (*tableName_s text)*
Assenta a *shape* sobre a os dados de um terreno, definido numa tabela personalizada (ou seja, criada dentro do projecto) com nome *tableName_s*, com precisão aos seus vértices (ou seja, com cada vértices assente sobre a superfície).

placeVertexOnCustomSurface (tableName_s text, condition_p boolean)
Assenta a *shape* sobre a os dados de um terreno, definido numa tabela personalizada (ou seja, criada dentro do projecto) com nome *tableName_s*, com precisão aos seus vértices (ou seja, com cada vértices assente sobre a superfície) para os quais a condição seja verdadeira.

placePolygonOnSurface (*tableName_s text)*
Assenta a *shape* sobre a os dados de um terreno, definido numa tabela pública com nome *tableName_s*, com precisão aos seus polígonos (ou seja, assentando os vértices de cada um ao mesmo nível).

placePolygonOnSurface (tableName_s text, condition_p boolean)
Assenta a *shape* sobre a os dados de um terreno, definido numa tabela pública com nome *tableName_s*, com precisão aos seus polígonos (ou seja, assentando os vértices de cada um ao mesmo nível) para os quais a condição seja verdadeira.

placePolygonOnCustomSurface (*tableName_s text)*
Assenta a *shape* sobre a os dados de um terreno, definido numa tabela personalizada (ou seja, criada dentro do projecto) com nome *tableName_s*, com precisão aos seus polígonos (ou seja, assentando os vértices de cada um ao mesmo nível).

placePolygonOnCustomSurface (tableName_s text, condition_p boolean)
Assenta a *shape* sobre a os dados de um terreno, definido numa tabela personalizada (ou seja, criada dentro do projecto) com nome *tableName_s*, com precisão aos seus polígonos (ou seja, assentando os vértices de cada um ao mesmo nível) para os quais a condição seja verdadeira.

colorShape (*color_s* rgba)
Colora toda a *shape* da cor indicada.

colorPolygon (color_p rgba, condition_p boolean)

Colora todos os polígonos, que correspondam a uma determinada condição, da cor indicada.

colorVertex (color_v rgba, condition_v boolean)
Colora todos os vértices, que correspondam a uma determinada condição, da cor indicada.

translateShape (*offset_s* vector3)
Desloca toda a *shape* pelo valor indicado.

translatePolygon (offset _p vector3, condition_p boolean)
Desloca todos os polígonos, que correspondam a uma determinada condição, pelo valor indicado.

rotateShape (*angle* double precision, *axis* text)
Roda toda a *shape* pelo valor e em volta do eixo indicado.

rotatePolygon (*angle* double precision, , *axis* text, *condition_p boolean*)
Roda todos os polígonos, que correspondam a uma determinada condição, pelo valor e em volta do eixo indicado.

scaleShape (*offset_s* vector3)
Realiza um escalamento de toda a *shape* pelo valor indicado.

scalePolygon (offset _p vector3, condition_p boolean)
Realiza um escalamento de todos os polígonos, que correspondam a uma determinada condição, pelo valor indicado.

Funções de Acesso

double precision x(*v1* vector3)
Devolve a coordenada x do vector indicado.

double precision y(*v1* vector3)
Devolve a coordenada y do vector indicado.

double precision z(*v1* vector3)
Devolve a coordenada z do vector indicado.

integer iterationLevel(%s)
Devolve o nível de iteração em que a *shape* foi criada.

scope scope(%s)
Devolve o âmbito da *shape*.

text symbol (%s)
Devolve o símbolo da *shape*.

%s parent(%s)
Devolve o *shape* que originou a *shape* indicada.

vector3 pivot (%s)
Devolve o pivô da *shape*.

text texture (%s)
Devolve a textura da *shape*.

integer numPolygons (%s)
Devolve o número de polígonos da *shape*.

%p firstPolygon (%s)
Devolve o primeiro polígono da *shape*.

%p largestPolygon (%s)
Devolve o polígono de maior dimensão da *shape*.

double precision r(*color* rgba)
Devolve a componente vermelha da cor.

double precision g(*color* rgba)
Devolve a componente verde da cor.

double precision b(*color* rgba)
Devolve a componente azul da cor.

double precision b(*color* rgba)
Devolve a componente *alpha* da cor.

scope scope(%p)
Devolve o âmbito do polígono.

vector3 normal(%p)
Devolve a normal do polígono.

vector3 firstPosition(%p)
Devolve a primeira posição do polígono.

record firstPosition(%p)
Devolve a primeira posição do polígono.

boolean hasTag(%p,*tagName* text)
Indica se o polígono possui uma determinada etiqueta aplicada.

text text (*rec* record, *field* text)
Obtém o campo *field* de tipo de texto de um registo.

integer integer (*rec* record, *field* text)
Obtém o campo *field* de tipo inteiro de um registo.

double precision double (*rec* record, *field* text)
Obtém o campo *field* de tipo precisão dupla de um registo.

geometry geometry (*rec* record, *field* text)
Obtém o campo *field* de tipo geometria de um registo.

numeric numeric (*rec* record, *field* text)
Obtém o campo *field* de tipo numérico de um registo.

vector3 normal(%v)
Devolve a normal do vértice.

vector3 position(%v)
Devolve a posição do vértice.

vectorUV uv (%v)
Devolve o mapeamento uv do vértice.

rgba rgba (%v)
Devolve a cor atribuída ao vértice.

Funções de Construção

vector3 vector3 ()
Devolve uma instância de um vector de coordenadas XYZ e o identificador de referência espacial srid, destinado a designar uma posição no espaço cartesiano. Os valores das coordenadas são aqui colocados a zero e o srid a -1 (significando que não possui nenhuma referência espacial).

vector3 vector3 (*srid* integer)
Devolve uma instância de um vector de coordenadas XYZ e o identificador de referência espacial srid, destinado a designar uma posição no espaço cartesiano. Os valores das coordenadas são aqui colocados a zero e o srid igual ao valor indicado no parâmetro.

vector3 vector3 (*x* double precision, *y* double precision, *srid* integer)
Devolve uma instância de um vector de coordenadas XYZ e o identificador de referência espacial srid, destinado a designar uma posição no espaço cartesiano. Os valores das coordenadas e srid são correspondem às definidas nos argumentos, sendo a coordenada Z deixada a 0.

vector3 vector3 (*x* double precision, *y* double precision, *z* double precision, *srid* integer)
Devolve uma instância de um vector de coordenadas XYZ e o identificador de referência espacial srid, destinado a designar uma posição no espaço cartesiano. Os valores das coordenadas e srid são correspondem às definidas nos argumentos.

rgba rgba()
Devolve uma instância de uma estrutura que define uma cor em formato rgba (red, green, blue, alpha), como todos os valores igual a 255, ou seja, definido a cor branca.

rgba rgba(*red* integer, *green* integer, *blue* integer)
Devolve uma instância de uma estrutura que define uma cor em formato RGBA (*red, green, blue, alpha*), com valores correspondentes aos parâmetros e o campo alpha igual a 255.

rgba rgba(*red* integer, *green* integer, *blue* integer, *alpha* integer)
Devolve uma instância de uma estrutura que define uma cor em formato RGBA (*red, green, blue, alpha*), com valores correspondentes aos parâmetros.

vectorUV vectorUV ()
Cria uma instância de um vector UV, destinado ao mapeamento de texturas, com valores nulos por defeito.

vectorUV vectorUV (*u* double precision, *v* double precision)
Cria uma instância de um vector UV, destinado ao mapeamento de texturas, com os valores correspondentes aos parâmetros.

vectorUV vectorUV (*u* double precision, *v* double precision)
Cria uma instância de um vector UV, destinado ao mapeamento de texturas, com os valores correspondentes aos parâmetros.

scope scope (*srid* integer)
Cria um âmbito de dimensão unitária e com srid correspondente ao parâmetro.

scope scope (*translation* vector3)
Cria um âmbito de dimensão unitária, com a posição definida pelo parâmetro.

scope scope (*xaxis* vector3,*yaxis* vector3,*zaxis* vector3)
Cria um âmbito de dimensão definida pelos parâmetros, e colocada na origem.

scope scope (*xaxis* vector3,*yaxis* vector3,*zaxis* vector3, *translation* vector3)

Cria um âmbito de dimensão e posição definida pelos parâmetros.

Funções Contextuais

in(%s, *boundary* geometry)
Verifica que se uma determinada *shape* se encontra localizada dentro do limite indicado.

in(%p, *boundary* geometry)
Verifica que se um determinado polígono se encontra localizado dentro do limite indicado.

hasInFront(%p, *shapeName* text, *distance* double precision)
Verifica que se um determinado polígono possui alguma *shape* com o nome indicado a menos da distância declarada.

Funções Matemáticas

vector3 multVal (*v1* vector3, *val* double precision) ou vector3 * double precision
Realiza a multiplicação das coordenadas do vector por uma constante.

vector3 sum (*v1* vector3, *v2* vector3) ou vector3 + vector3
Realiza a soma das coordenadas de dois vectores. É imperativo que os vectores possuam o mesmo identificador de referência espacial srid.

vector3 sub (*v1* vector3, *v2* vector3) ou vector3 - vector3
Realiza a soma das coordenadas de dois vectores. É imperativo que os vectores possuam o mesmo identificador de referência espacial srid.

vector3 multVal (*v1* vector3, *val* double precision) ou vector3 * double precision
Realiza a multiplicação das coordenadas do vector por uma constante.

double precision size (*v1* vector3)
Devolve o tamanho de um vector.

vector3 angle (*v1* vector3, *v2* vector3)
Calcula o ângulo entre dois vectores.

vector3 cross (*v1* vector3, *v2* vector3)
Calcula o produto de dois vectores.

vector3 normalize(*v1* vector3)
Devolve o vector normalizado.

double precision area(*geom* geometry)
Calcula a área de uma geometria.

integer random (*value1* integer, *value2* integer)
Devolve um valor inteiro aleatório entre os limites definidos (inclusive).

double precision random (*value1* double precision, *value2* double precision)
Devolve um valor de precisão dupla aleatório entre os limites definidos (inclusive).

integer abs (*value1* integer)
Devolve o valor absoluto de um do parâmetro inteiro.

double precision abs (*value1* double precision)
Devolve o valor absoluto de um do parâmetro de precisão dupla.

double precision rad2deg (*value1* double precision)
Converte de radianos para graus.

double precision deg2rad (*value1* double precision)
Converte de graus para radianos.

double precision exp (*value1* double precision)
Calcula o valor de 2^{value}.

double precision sin (*value1* double precision)
Calcula o seno do valor.

double precision asin (*value1* double precision)
Calcula o inverso do seno do valor.

double precision tan (*value1* double precision)
Calcula a tangente do valor.

double precision atan (*value1* double precision)
Calcula o inverso da tangente do valor.

double precision cot (*value1* double precision)
Calcula a co-tangente do valor.

Funções de Conversão

integer i(*val* double precision)
Devolve o valor convertido para inteiro.

integer i(*val* numeric)
Devolve o valor convertido para inteiro.

integer i(*val* text)
Devolve o valor convertido para inteiro.

double d(*val* numeric)
Devolve o valor convertido para precisão dupla.

double d(*val* text)
Devolve o valor convertido para precisão dupla.

Anexo C

Exemplo de Código PG3D para Criação de uma Casa Detalhada

```
//=======================================
// Ficheiro de Regras para construção de casa complexa
//=======================================

Edificio -->
        extrude(double(record(%p),'height'))
        setTexture(@t_Wall1)
        setUV(10,10)
        { hasTag(%p,'Edificio.Side') %each : EdificioLateral,
                hasTag(%p,'Edificio.Top') %each : EdificioTopo,
                %rest : EdificioResto};


EdificioTopo -->
        extrudeTaper(0.1,-3)
        extrudeTaper(5,15,hasTag(%p,'EdificioTopo.Top'))
        settexture(@t_RoofRed) setUV(10,10) EdificioTelhado;

EdificioLateral-->
        slicey([3,%f],['Baixo','Cima'])
        { hasTag(%p,'EdificioLateral.Cima') : EdificioLateralCima(1),
                hasTag(%p,'EdificioLateral.Baixo') : EdificioPiso(0) };

EdificioLateralCima(y integer) -->
        IF size(zaxis(scope(%s))) < 3 THEN
                EdificioQualquerRestoParede
        ELSE
                clearAllTags() slicey([3,%f],['Baixo','Cima'])
                { hasTag(%p,'EdificioLateralCima.Baixo') : EdificioPiso(y),
                        hasTag(%p,'EdificioLateralCima.Cima') : EdificioLateralCima(y + 1) }
        ENDIF;


EdificioPiso(y integer) -->
        IF size(xaxis(scope(firstpolygon(%s)))) < 3 THEN
                EdificioQualquerRestoParede
        ELSE
                slicex([%f, 3,%f],['LadoEsq','QuadranteJanela', 'LadoDir'])
                { hasTag(%p,'EdificioPiso.QuadranteJanela') : EdificioQuadranteJanela(0,y),
                        hasTag(%p,'EdificioPiso.LadoEsq') : EdificioPisoLadoEsq(-1,y),
                        hasTag(%p,'EdificioPiso.LadoDir') : EdificioPisoLadoDir(1,y)}
        ENDIF;



EdificioPisoLadoEsq(x integer,y integer) -->
        clearAllTags()
        IF size(xaxis(scope(firstpolygon(%s)))) < 3 THEN
                EdificioQualquerRestoParede
        ELSE
                slicex([%f, 3],['LadoEsq','QuadranteJanela'])
        { hasTag(%p,'EdificioPisoLadoEsq.QuadranteJanela') : EdificioQuadranteJanela(x,y),
                hasTag(%p,'EdificioPisoLadoEsq.LadoEsq') : EdificioPisoLadoEsq(x-1,y)}
        ENDIF;

EdificioPisoLadoDir(x integer,y integer) -->
        clearAllTags()
        IF size(xaxis(scope(firstpolygon(%s)))) < 3 THEN
                EdificioQualquerRestoParede
        ELSE
                slicex([3, %f],['QuadranteJanela','LadoDir'])
                { hasTag(%p,'EdificioPisoLadoDir.QuadranteJanela') : EdificioQuadranteJanela(x,y),
                        hasTag(%p,'EdificioPisoLadoDir.LadoDir') : EdificioPisoLadoDir(x+1,y)}
```

```
        ENDIF;


//=================================================
//Porta

//=================================================
EdificioQuadrantePorta -->
        extrude(-0.4)
        slicex([%f, 1.5, %f],['Resto','PortaMeio', 'Resto'], hasTag(%p,'EdificioQuadrantePorta.Top'))
        slicey([2,%f],['ZonaPorta','Resto'],hasTag(%p,'EdificioQuadrantePorta.PortaMeio'))
        {hasTag(%p,'EdificioQuadrantePorta.ZonaPorta'): EdificioQuadranteZonaPorta,
                hasTag(%p,'EdificioQuadrantePorta.Resto') : EdificioQualquerRestoParede,
                hasTag(%p,'EdificioQuadrantePorta.Side') : EdificioQualquerRestoParede};



EdificioQuadranteZonaPorta -->
        slicex([0.75,0.75],['Porta','Porta'])
        {hasTag(%p,'EdificioQuadranteZonaPorta.Porta') %each : EdificioPorta};


EdificioPorta -->
        slicex([0.1, %f,0.1],['Resto','ZonaVidroPortaX','Resto'])
        slicey([0.1, %f,0.1],['Resto','ZonaVidroPorta','Resto'],hasTag(%p,'EdificioPorta.ZonaVidroPor
taX'))
        {hasTag(%p,'EdificioPorta.ZonaVidroPorta'): EdificioPortaVidro,
                hasTag(%p,'EdificioPorta.Resto'): EdificioPortaRestos};

EdificioPortaVidro -->
        setUV(1,1) setTexture(@t_Glass) EdificioPortaVidroTerminada;

EdificioPortaRestos -->
        setUV(1,1) setTexture(@t_Metal1) EdificioPortaRestosTerminada;



//=======================================================
//Janelas
//=======================================================
EdificioQuadranteJanela(x integer,y integer) -->
        IF x = 0 and y = 0 THEN
                 EdificioQuadrantePorta
        ELSIF (x + 1) % 2 = 0 and y > 0 THEN
                EdificioQuadranteVaranda
        ELSE
                slicey([%f, 1.5,%f],['Resto','ZonaMeio', 'Resto'])
                slicex([%f,1.4,1.4,%f],['Resto','ParteJanelaEsq','ParteJanelaDir','Resto'],
                                                hasTag(%p,'EdificioQuadranteJanela.ZonaMeio'))
                { hasTag(%p,'EdificioQuadranteJanela.Resto') : EdificioQualquerRestoParede,
                        hasTag(%p,'EdificioQuadranteJanela.ParteJanelaEsq') :
EdificioJanelaParteEsq,
                        hasTag(%p,'EdificioQuadranteJanela.ParteJanelaDir') :
EdificioJanelaParteDir}
        ENDIF;

EdificioJanelaParteEsq -->
        slicex([0.7,%f],['PartePortada', 'ParteJanela'])
        STOC
               50% { hasTag(%p,'EdificioJanelaParteEsq.PartePortada') : EdificioQualquerRestoParede,
                                hasTag(%p,'EdificioJanelaParteEsq.ParteJanela') :
EdificioJanelaPortadaFechada}  //portada fechada
               50% { hasTag(%p,'EdificioJanelaParteEsq.PartePortada') : EdificioJanelaPortadaAberta,
                                hasTag(%p,'EdificioJanelaParteEsq.ParteJanela') : EdificioJanela}
        //portada aberta
        ENDSTOC;

EdificioJanelaParteDir -->
        slicex([0.7,%f],['ParteJanela', 'PartePortada'])
        STOC
               50% { hasTag(%p,'EdificioJanelaParteDir.ParteJanela') : EdificioJanelaPortadaFechada,
                                hasTag(%p,'EdificioJanelaParteDir.PartePortada') :
EdificioQualquerRestoParede}  //portada fechada
                50% { hasTag(%p,'EdificioJanelaParteDir.ParteJanela') : EdificioJanela,
                                hasTag(%p,'EdificioJanelaParteDir.PartePortada') :
EdificioJanelaPortadaAberta}      //portada aberta
        ENDSTOC;

EdificioJanelaPortadaAberta -->
        extrude(0.1) settexture(@t_Door1) setuv() EdificioJanelaPortadaAbertaTexturada;

EdificioJanelaPortadaFechada -->
        settexture(@t_Door1) setuv() EdificioJanelaPortadaFechadaTexturada;

EdificioJanela -->
        extrude(-0.2)
        { hasTag(%p,'EdificioJanela.Side'): EdificioJanelaBeiral,
```

```
                hasTag(%p,'EdificioJanela.Top') : EdificioJanelaMesmo};

EdificioJanelaMesmo -->
        slicex([0.1,%f,0.1,%f,0.1],['Resto', 'Vidro','Resto','Vidro','Resto'])
        slicey([0.1, %f,0.1,%f,0.1],['Resto',
'Vidro','Resto','Vidro','Resto'],hasTag(%p,'EdificioJanelaMesmo.Vidro'))
        { hasTag(%p,'EdificioJanelaMesmo.Resto'): EdificioJanelaMetal,
                hasTag(%p,'EdificioJanelaMesmo.Vidro') : EdificioJanelaVidro};

EdificioJanelaMetal -->
        setUV(1,1) setTexture(@t_Metal1) EdificioJanelaMetalTerminada;

EdificioJanelaVidro -->
        setUV(1,1) setTexture(@t_Glass) EdificioJanelaMetalTerminada;

//=======================================================
//Varanda
//=======================================================

EdificioQuadranteVaranda -->
        slicey([0.2,0.2,2,%f],['Resto','ZonaBaseVarandaToda', 'ZonaPorta','Resto'])
        slicex([ %f, 1.5, %f],['Resto','PortaMeio',
'Resto'],hasTag(%p,'EdificioQuadranteVaranda.ZonaPorta'))
        slicex([ %f, 2.5, %f],['Resto','ZonaBaseVaranda',
'Resto'],hasTag(%p,'EdificioQuadranteVaranda.ZonaBaseVarandaToda'))
        { hasTag(%p,'EdificioQuadranteVaranda.Resto') : EdificioQualquerRestoParede,
                hasTag(%p,'EdificioQuadranteVaranda.PortaMeio') : EdificioQuadranteZonaPorta,
                hasTag(%p,'EdificioQuadranteVaranda.ZonaBaseVaranda') : EdificioVarandaBase};

EdificioVarandaBase -->
        extrude(1.5)
        { hasTag(%p,'EdificioVarandaBase.Side') and z(normal(%p)) > 0:  EdificioVarandaBaseCima,
                %rest : EdificioQualquerRestoParede} ;

EdificioVarandaBaseCima -->
        if size(xaxis(scope(%s))) > size(yaxis(scope(%s))) THEN
                slicex([%f, 2.3, %f],['LateralBase','Meio', 'LateralBase'])
                slicey([0.1, %f],['LateralBase','Resto'],hasTag(%p,'EdificioVarandaBaseCima.Meio'))
        ELSE
                slicey([%f, 2.3, %f],['LateralBase','Meio', 'LateralBase'])
                slicex([0.1, %f],['LateralBase','Resto'],hasTag(%p,'EdificioVarandaBaseCima.Meio'))
        ENDIF
                { hasTag(%p,'EdificioVarandaBaseCima.Resto') : EdificioQualquerRestoParede,
                        hasTag(%p,'EdificioVarandaBaseCima.LateralBase') : EdificioLateralBase};

EdificioLateralBase -->
        extrude(1.0) EdificioLateralBaseFim;

//=======================================================
//Tudo o que sobrar da parede
//=======================================================
EdificioQualquerRestoParede -->
        setUV(10,10) EdificioQualquerRestoParedeUV;
```