INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Estudo Focado no Utilizador de Software Gratuito para Modelação e Visualização 3D Realista

DIOGO FERREIRA DE CASTRO
Outubro de 2015

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Estudo Focado no Utilizador de Software Gratuito para Modelação e Visualização 3D Realista

Diogo Ferreira de Castro

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas Gráficos e Multimédia

Orientador: Filipe de Faria Pacheco

Júri:

Presidente:

[Nome do Presidente, Categoria, Escola]

Vogais:

[Nome do Vogal1, Categoria, Escola]

[Nome do Vogal2, Categoria, Escola] (até 4 vogais)

Porto, Outubro 2015

Resumo

Computação gráfica um campo que tem vindo a crescer bastante nos últimos anos, desde áreas como cinematográficas, dos videojogos, da animação, o avanço tem sido tão grande que a semelhança com a realidade é cada vez maior. Praticamente hoje em dia todos os filmes têm efeitos gerados através de computação gráfica, até simples anúncios de televisão para não falar do realismo dos videojogos de hoje.

Este estudo tem como objectivo mostrar duas alternativas no mundo da computação gráfica, como tal, vão ser usados dois programas, *Blender* e *Unreal Engine*. O cenário em questão será todo modelado de raiz e será o mesmo nos dois programas. Serão feitos vários renders ao cenário, em ambos os programas usando diferentes materiais, diferentes tipos de iluminação, em tempo real e não de forma a mostrar as várias alternativas possíveis.

Keywords: blender, unreal, render, raytracing, pathtracing, iluminação

Abstract

The field is growing very much in the couple last years, areas like cinematographic, video games, architecture are seeing a big step up in terms of the image quality, and the realism nowadays is huge. Almost every movie, game, is made using computer graphics processes, even some television ads use computer graphics.

The objective of the study is to show two render techniques from two different programs, Blender and Unreal Engine. To do that, some models are going to be model in both programs, then different materials, different types of illumination are going to be tested to show some possible alternatives.

Keywords: blender, unreal, render, raytracing, pathtracing, iluminação

Índice

Lista de Figuras

1 Introdução

Como tema para a tese, do curso de Mestrado em Sistemas Gráficos e Multimédia foi escolhido fazer um estudo sobre dois programas de 3D, Blender e o Unreal.

O Blender trata-se de um programa de modelação 3D, que já algum tempo recebeu um novo motor de renderização e que permite sem grande esforço a criação de conteúdo 3D com bastante realismo. Este motor de renderização tem vindo a crescer e é cada vez mais usado entre os artistas da computação gráfica. O Unreal é parecido tem desvantagens e vantagens como todos os programas, mas com o Blender pode formar um conjunto que permite a realização de conteúdo 3D, para várias vertentes.

O ênfase para o estudo será para o render usado de cada um dos programas e como se complementam. Quais as vantagens e desvantagens de cada um mostrar as semelhanças e as diferenças e também como em conjunto formam uma boa solução.

Este estudo tem como objectivo mostrar as duas alternativas no mundo da computação gráfica, como tal, vai ser modelado um cenário no Blender e posteriormente será exportado para o Unreal Engine. O cenário em questão será todo modelado de raiz e será o mesmo nos dois programas. Serão feitos vários renders ao cenário, em ambos os programas usando diferentes materiais, diferentes tipos de iluminação, em tempo real e não de forma a mostrar as várias alternativas possíveis. Numa primeira fase, será feito um levantamento sobre as áreas que fazem uso da computação gráfica, para que serve e com que objectivo são usadas, quais são as principais técnicas de render usadas, bem como as várias alternativas que existem no mercado actual. O capítulo seguinte aborda a técnica de RayTracing e as suas variantes, que alternativa é esta. O seguinte capítulo descreve de que se trata Global Illumination, vários aspectos referentes à iluminação serão detalhados, quais as diferenças, e algumas das técnicas pertencentes a este grupo. Numa fase final, o relatório visa mostrar ao leitor todos os passos executados para a criação deste projecto, bem como quaisquer dúvidas ou problemas encontrados em todo o processo.

2 Estado da arte

De modo a começar o relatório da forma eficiente, foram feitas pesquisas ao tema da computação gráfica, técnicas, motores de render, software.

O mundo da computação gráfica tem vindo a crescer bastante nos últimos anos, desde áreas cinematográficas, dos videojogos ou da animação, o avanço tem sido tão grande que a semelhança com a realidade é cada vez maior. Tudo de modo a resolver a rendering equation. [Rendering equation, 1986] Praticamente hoje em dia todos os filmes têm efeitos gerados através de computação gráfica, até simples anúncios de televisão. O realismo é cada vez maior, e as possibilidades são enormes, principalmente na área dos videojogos [Andrew Burnes. 2011] que tem vindo a crescer. Recentemente, alguns motores de jogo [Ben Woodhouse, 2012] já começaram a introduzir novas técnicas de render, conseguindo assim uma imagem mais real, e um melhor aproveitamento do hardware existente.

Também na área de arquitectura, tem vindo a tirar partido da computação gráfica, como forma de promover uma forma de virtualização de áreas ainda por construir. Um exemplo onde a computação gráfica tem tido um grande impacto nesta área é o facto de 75% dos produtos que são mostrados nos catálogos do IKEA [Mark Wilson, 2014] são gerados por computação gráfica.

Existem basicamente 2, técnicas, aproximações: Rasterization e RayTracing. Na técnica de Rasterization para cada polígono/triângulo é calculado o número de pixels que são cobertos pela área do polígono, se é visível, onde se encontra e que cor tem. [Rasterization overview, 2015] É bastante usada nos dias de hoje pois permite resultados bastante satisfatórios, mas com o surgimento de novas técnicas começa a ficar ultrapassada. RayTracing funciona de forma inversa, para cada pixel da imagem final é calculado a cor e o polígono/triângulo a que pertence o pixel. A Rasterization trata-se de uma técnica bastante usada e estudada, optimizada para GPU, forte em paralelismo, com suporte para OpenGL e DirectX. RayTracing tem a vantagem de ter melhores reflexos, transparências e sombras, enquanto Rasterization tem a grande vantagem a nível de performance.

Partindo das vantagens de cada uma das técnicas, pode-se pensar em criar um motor de renderização hibrido. À partida elas parecem complementar-se, é fácil imaginar usar a Rasterization para determinar se os polígonos estão visíveis, tirando partido da performance excelente da técnica, e usar RayTracing apenas em certas superfícies para adicionar aquele realismo, como por exemplo sombras ou obter reflexos e transparências exactas. Esta possibilidade já foi executada pela Pixar na produção do filme Cars [Pixar Animation Studios].

Infelizmente, embora pareça muito promissora, uma solução hibrida não é fácil de implementar. Uma das desvantagens do RayTracing é a estrutura de informação necessária de maneira a organizar os polígonos a limitar o número de raios necessários para calcular as intersecções, ou seja, com a utilização de um motor hibrido esta desvantagem continuará presente. Pode-se então considerar usar a Rasterization para renderizar a informação dinâmica e o RayTracing para a informação estática, mas com isto perdemos todas as vantagens do RayTracing. [RayTracing for the movie "Cars", Pixar] Um grande problema em termos de performance está no facto do cálculo dos segundos raios, que são os necessários para manter no nosso motor hibrido, caso contrário não seria usado RayTracing nenhum, perdendo-se assim o realismo do RayTracing e a performance da Rasterization. Na informação dinâmica está o ponto forte do algoritmo RayTracing que são os efeitos de reflexão especular e transparência, que são bastantes difíceis de conseguir com a técnica de Rasterization embora possíveis. Por outro lado será que vale mesmo a pena todo este realismo? [RayTracing for the movie "Cars", Pixar] Uma vez que a nossa visão é facilmente enganada, o resultado aproximado que é conseguido pela Rasterization é já bastante satisfatório. Como tal muitos avanços ainda terão de ser feitos de maneira a que o RayTracing se torne a alternativa credível à técnica Rasterization no que diz respeito a renderização em tempo real.

Todo este realismo tem um custo bastante grande sobre a performance, dependendo da complexidade que se pretende o tempo de renderização pode ser bastante longo. Exemplo disto é o recente filme da Disney, que usou um software de render especialmente criado para renderizar a iluminação, Hyperion. [Disney CODA, 2015] Trata-se de um sistema que simula a luz, e consiste em colocar várias fontes de luz numa cena e, cada uma delas, lança vários raios por toda a cena, de objecto em objecto os raios vão perdendo força e acabam por desaparecer eventualmente alguns raios irão ao encontro da câmera e são mostrados ao utilizador, isto é a versão de Global Illumination da Disney. Pelo caminho estes raios vão iluminando a cena, passando de objecto em objecto, definindo as cores, sombras, etc. Para além de simulação da luz a Disney criou também um simulador de cabelos, para o filme Tangled, um simulador de neve, para o filme Frozen. Para o filme, Big Hero 6 foram precisas 200,000,000 horas de render, 1,100,000 por dia, possíveis graças a 55,000 cores. [Disney Hyperion, 2015] Para compreender melhor os avanços alcançados, o Hyperion conseguiria renderizar o filme, Tangled, em 10 dias. [Disney render, 2015]

Existem ainda outras técnicas que em conjunto permitem todo aquele aspecto realista desejado, de maneira a que cada detalhe, reflexão, transparência, sombra, sejam bem renderizadas. Para iluminação directa é usado RayTracing, para iluminação difusa é usado Radiosity, para Caustics Photon Mapping, etc. Todas estas misturas de técnicas são usadas para chegar o mais perto possível da rendering equation. Das várias alternativas uma tem vindo a crescer muito, PathTracing que por si só permite obter grandes resultados. Tem algumas semelhanças com o RayTracing, mas segue um processo diferente e no fim esse caminho diferente é visível.

Vários tipos de software usam diferentes técnicas de render, para obter os melhores resultados, uns mais completos que outros e com um conjunto alargado de funções. Desde open source a soluções pagas a escolha é muita, as grandes empresas usam software com grande afirmação no mercado e largamente usados, software que tem vindo a melhorar e com isso a tirar melhor partido do hardware existente. Casos como MentalRay [Mental Ray, NVIDIA], RenderMan [RenderMan overview, 2015] são soluções viradas para grandes projectos e por serem pagas, não estão acessíveis a todos os utilizadores. Por outro lado, várias companhias têm adoptado a vertente de software free, como é o caso recente do Unreal 4.

Hoje em dia temos a capacidade para correr cenários, bastante realista em tempo real, coisa que anteriormente só era possível através de computadores muito potentes. Nos tempos actuais, existem vários softwares que permitem em tempo real, obter grandes resultados.

Brigade é um motor que usa PathTracing em tempo real para jogos e muito mais. Este motor usa PathTracing, extensão do RayTracing que permite criar imagens foto realistas, simula vários caminhos da luz a cada pixel para calcular a respectiva cor. Trata-se de um motor bastante completo mas que ainda anda em fase de desenvolvimento. [OTOY render, 2015]

Mais conhecido do que o Brigade é o Unreal Engine 4 que usa uma técnica diferente. Não usa RayTracing puro nem PathTracing, mas tem "presente" um conjunto de técnicas como por exemplo VXGI [Cyril Crassin, 2012], trata-se uma versão de iluminação global da Epic ainda em fase de desenvolvimento, que permite uma iluminação bastante realista, Physical Based Shading que simula o que a iluminação realmente faz aos materiais, ao contrário do que o que pensamos, entre outras. O resultado final é mais preciso e mais natural e funciona em todos os ambientes de iluminação. O Unreal 4 tem vindo também a melhorar de dia para dia, como se trata de um software aberto, o uso é maior e a constante melhoria ajuda a tornar o motor mais robusto. Outro software que tem vindo a crescer, de versão em versão, é o Blender. Modelação, animação, são algumas das tarefas que podem ser realizadas com o Blender. A sua principal função é modelar objectos que posteriormente serão exportados para o Unreal Engine, por exemplo. As recentes versões do Blender introduziram um motor de render novo baseado em PathTracing, Cycles, os resultados são tão bons que a utilização de software pago para o mesmo efeito hoje em dia não faz sentido.

3 Variantes de RayTracing

3.1 Alternativa PathTracing

RayTracing e PathTracing são duas técnicas diferentes, no que diz respeito ao procedimento escolhido para obter os resultados, e iguais no que toca à base de renderização. Ambas lançam os seus raios com início na câmera, com destino a uma fonte de luz.

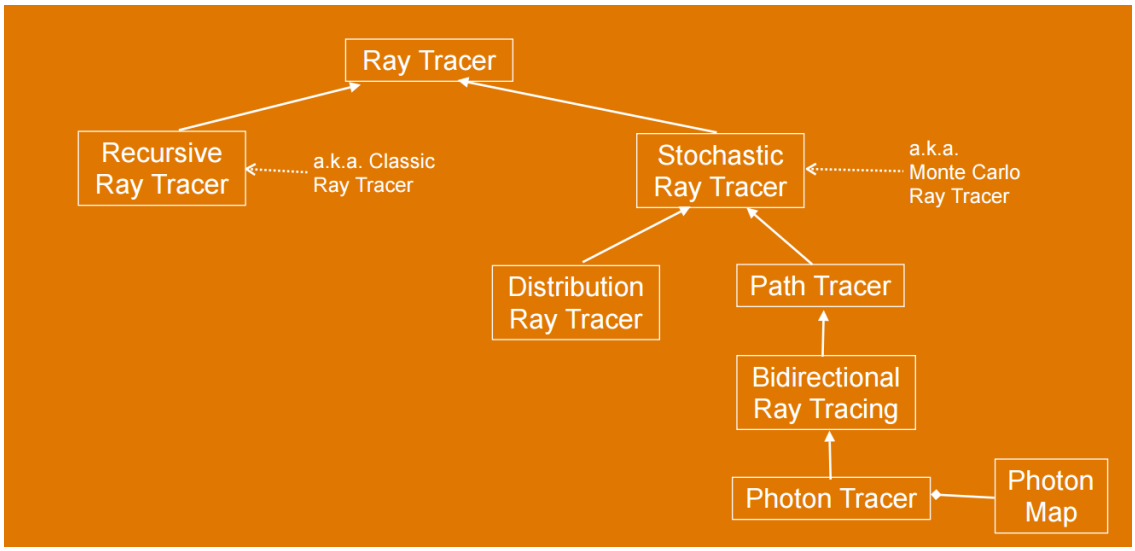

Figura 1 – esquema dos vários algoritmos de iluminação

A imagem acima mostra as variantes do RayTracer, onde existem dois grupos principais. O conhecido como o RayTracer clássico consiste em lançar um raio, vindo da câmera, este atinge uma superfície, novos raios são gerados de três tipos (sombras, transparências, e reflexos). Para os raios da sombra, estes são lançados na direcção de cada fonte de luz, e se um raio específico intersectar algum objecto, a superfície do objecto não é iluminado por esta fonte de luz. Para cada fonte de luz é preciso um "raio de sombra" específico.

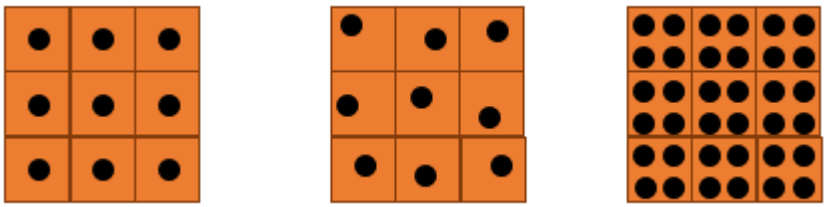

Figura 2 – single-sample, random single-sampling, MSAA

O raio de transparência é lançado para dentro do objecto. O raio dos reflexos é lançado como mirror em relação ao raio que atingiu a mesma superfície, o objecto mais próximo que intersectar esse raio é o que vai aparecer como reflexo.

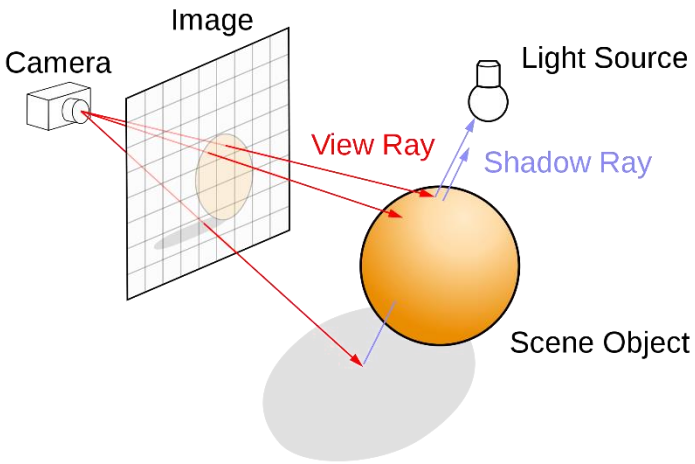

Figura 3[1] – esquema de RayTracer

Ao contrário deste método onde é usado apenas um raio e que não obtém resultados muito realistas, o Stochastic RayTracer, ou Monte Carlo RayTracer, ou Distributed RayTracer, ou simplesmente PathTracer, consegue melhores resultados. [Monte Carlo methods, 2015] Esta técnica consiste em recolher várias amostras de um dado pixel, para isso são lançados vários raios, e com isso conseguir um valor mais aproximado do valor correspondente daquele pixel, tais como a cor, intensidade, etc.

A qualidade final da imagem obtida com render é determinada pelo número de samples, ou seja, quanto mais samples por pixel uma imagem tiver mais definição tem. Quanto mais tempo o render estiver em funcionamento, melhor fica a imagem, até a um ponto em que a diferença é praticamente nula isto no que diz respeito a PathTracing. RayTracing também segue um processo semelhante, para obter uma imagem melhor, mas de forma discreta, ao contrário do PathTracing que vai adicionando os valores. RayTracing tem um "fim" enquanto o PathTracing quanto mais tempo está em funcionamento, mais precisa é a imagem final. A diferença está na forma como cada técnica calcula os valores da iluminação. [Monte Carlo methods, 2015]

[1] http://stackoverflow.com/questions/10012219/how-to-implement-depth-of-field-in-ray-tracer

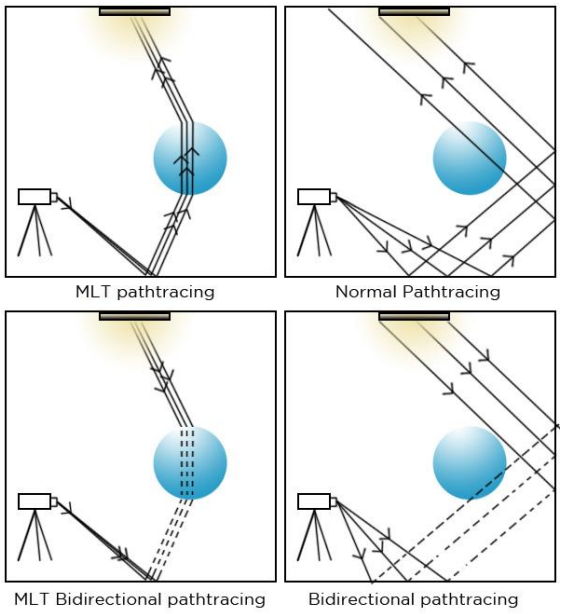

Figura 4[2] – esquema das variantes de PathTracing

Em PathTracing um raio é lançado da câmera e à medida que vai "saltando" de objecto em objecto, recolhe a informação necessária, tais como a cor, valor de reflexão, refração, para resolver a rendering equation. [Rendering equation, 1986] Cada vez que um raio entra em contacto com um material, é dividido em vários raios, dependendo do material do objecto que intersecta, este processo é depois repetido por esses mesmos raios. Na parte final, estes raios intersectam uma fonte de iluminação, recolhendo assim o último valor necessário para resolver a rendering equation, a energia inicial necessária. A equação é completa e o computador trata de renderizar a cor dos pixels com valor baseado na soma total da equação. Para tornar este processo mais rápido é possível para cada raio intersectado, traçar um raio na direcção de cada fonte de iluminação, em vez de andarem a "saltar" de objecto em objecto. [PathTracing methods, 2015]

Existe também uma variante da técnica de PathTracing, que se chama Bidirectional PathTracing que consiste em traçar raios com ponto de partida da câmera e também das fontes de iluminação. [Bi-directional PathTracing, 1993] PathTracing normal por vezes tem dificuldade em renderizar a iluminação, especialmente quando existe caustics ou a fonte de iluminação está escondida. A razão para isto acontecer, é devido ao facto de não ser possível disparar um raio na direcção da luz, uma vez que é improvável que o raio atinge a fonte de iluminação visto que a sua direcção pode variar ao intersectar outras superfícies. Com isso alguns renders podem resultar em imagens com granulado.

[2] http://indigorenderer.com/node/1135

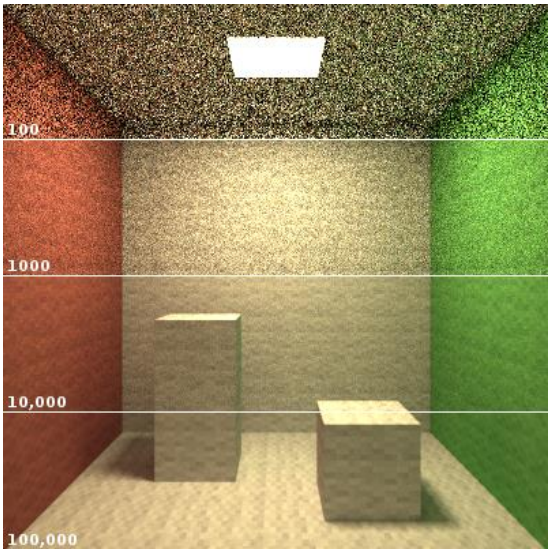

Figura 5[3] – diferença de samples em pathtracing

Em RayTracing, um raio é lançado da câmera, como foi referido acima, mas quando estes raios intersectam um objecto, múltiplos raios são lançados na direcção das fontes de iluminação e é então calculado o pixel tendo em conta as propriedades do material intersectado e a quantidade iluminação que está a receber de cada fonte de iluminação. Isto prova que RayTracing apenas consegue renderizar iluminação directa. Para ter todo aquele aspecto realista, é necessário usar um conjunto de técnicas adicionais, como foi referido acima.

[3] http://chunky.llbit.se/path_tracing.html

Figura 6[4] – pathtracing cena

À medida que as placas gráficas têm vindo a ficar mais rápidos, técnicas como PathTracing [Vilém Otte, 2013] começaram a ser mais interessantes como forma de estudo e implementação em diversos projectos. Recentes tentativas de rendering interactivo são bastante complicadas, especialmente no que diz respeito à componente da iluminação, PathTracing vem ajudar bastante nesse aspecto.

Para resumir, PathTracing é uma alternativa mais realista, baseado em força bruta para calcular a iluminação. RayTracing é mais rápido e eficiente, graças ao conjunto que forma com outras técnicas. Diferentes soluções para diferentes aplicações. PathTracing resolve o problema de ser necessário o uso de várias técnicas e com os processadores gráficos cada vez mais eficientes esta técnica pode vir a ser bastante usada no futuro.

[4] http://www.disneyanimation.com/technology/innovations/hyperion

3.2 Real-Time PathTracing

De forma a alcançar o fotorealismo na área de computação gráfica, são necessários vários truques e combinações de forma a obter sombras suaves, ambiente occlusion, reflexos reais, bem como a componente da iluminação, que tanta diferença faz. A junção de vários efeitos é muitas vezes complicada e pode levar a problemas de código. Como foi descrito no capítulo anterior a técnica de PathTracing [PathTracing methods, 2015], resolve este problema uma vez que contém todos esses efeitos. Até aos tempos de hoje PathTracing, era considerado muito pesado a nível computacional, para criar efeitos especiais em filmes e animações sem contar com aplicações em tempo real, como por exemplo jogos. Contudo, e como está descrito no capítulo anterior os processadores gráficos têm vindo a ganhar grandes capacidades de processamento, principalmente no que toca a capacidade de processamento paralelo.

PathTracing através da placa gráfica [Vilém Otte, 2013] é hoje uma realidade para renderizar cenários em tempo real, com todos os efeitos que fazem parte da técnica. Embora continue a ser uma técnica bastante pesada para usar em tempo real, já existem motores de render em tempo real, que usam PathTracing, como é o caso do motor Brigade [Jacco Bikker, Jeroen van Schijndel, 2012], [Jim Thacker, 2014]. Usa renderização através da placa gráfica o que torna os tempos de render bastante reduzidos, suporta qualquer tipo de mesh, ou seja, desde árvores, carros a outro tipo de mesh e do ponto de visto de tempo necessário de render é o mesmo. Centenas, milhares ou milhões de polígonos, o Brigade é totalmente dinâmico, suporta também para a realização de outras tarefas, a possibilidade de realizar animações. Conta ainda com um diverso conjunto de efeitos que podem ser adicionados à cena, desde bloom, filtros de cor, etc, de maneira a criar o ambiente desejado. No que diz respeito aos materiais, todas as propriedades são personalizáveis em tempo real.

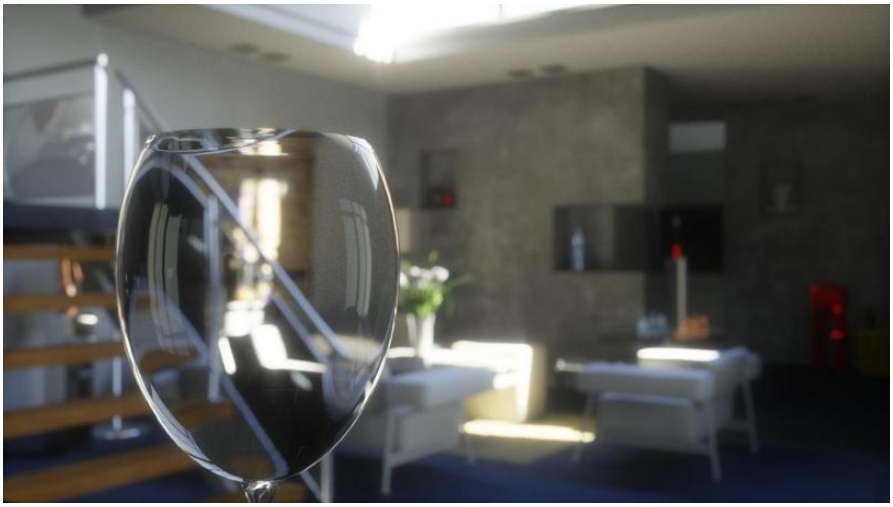

Figura 7[5] – caustics no brigade

[5] http://raytracey.blogspot.pt/2012/09/brigade-new-screenshots.html

Como se trata de um motor de fase de desenvolvimento, ainda tem algum daquele efeito granulado do PathTracing, mas comparado com as versões finais as melhorias são notáveis. Sendo já possível nos dias de hoje renderizar imagens em tempo real usando PathTracing, o único problema esta nos requisitos para todo este cálculo, embora não seja preciso grandes quantidades de render farms, são necessários placas gráficas de gama alta.

Anteriormente era impensável pensar renderizar em tempo real*,* em *RayTracing* eram necessárias horas para renderizar um simples bola de cristal com uma animação e hoje temos cenas renderizadas usando PathTracing em tempo real com todos os efeitos agradáveis à vista. O próximo passo é conseguir diminuir ou eliminar o efeito granulado e conseguir frames mais estáveis.

Figura 8[6] – iluminação global no brigade

Para além da área dos vídeo jogos que pode vir a utilizar PathTracing como base, a área cinematográfica já começa a tirar partido desta técnica. O filme Constructor [Jim Thacker, 2014] usa PathTracing para renderizar cenas em tempo real, com motion capture. Todos os frames capturados em tempo real servem para verificar o aspecto final que podem ter, ou seja, algo que pode ser usado para filmes com computação gráfica. Tudo isto, era possível à algum tempo atrás, mas demorava horas, senão dias a renderizar uma cena sem ter o aspecto aceitável para produção.

[6] http://raytracey.blogspot.pt/2012/09/brigade-new-screenshots.html

4 GPU render

No passado a melhor forma de tornar mais rápido uma aplicação era conseguida através do aumento do CPU clock, que foi assim aumentando até aos dias de hoje. Contudo, nos tempos de hoje temos CPU com clocks de 4ghz, mas em contrapartida com um elevado aumento de consumo de energia e um problema de temperaturas. Os fabricantes de CPU passaram então a aumentar o número de cores, em vez do clock mas isto não fez com que as aplicações fossem executadas mais rápido, uma vez que foram programas para correr em apenas um core.

Com o evoluir das placas gráficas, a área da computação gráfica deu um salto bastante grande, isto deve-se ao facto de ser possível usar a GPU para fazer render em vez do CPU. Muitas das aplicações hoje em dia usam aceleração com base em GPU o que faz com que sejam executadas mais rápido e seja também possível permitir acções que antes não eram possíveis. Muitas destas aplicações tem diferentes pressupostos e pertencem a diferentes áreas, tais como na medicina, das finanças, da física, da química, da imagem, da geográfica entre outras.

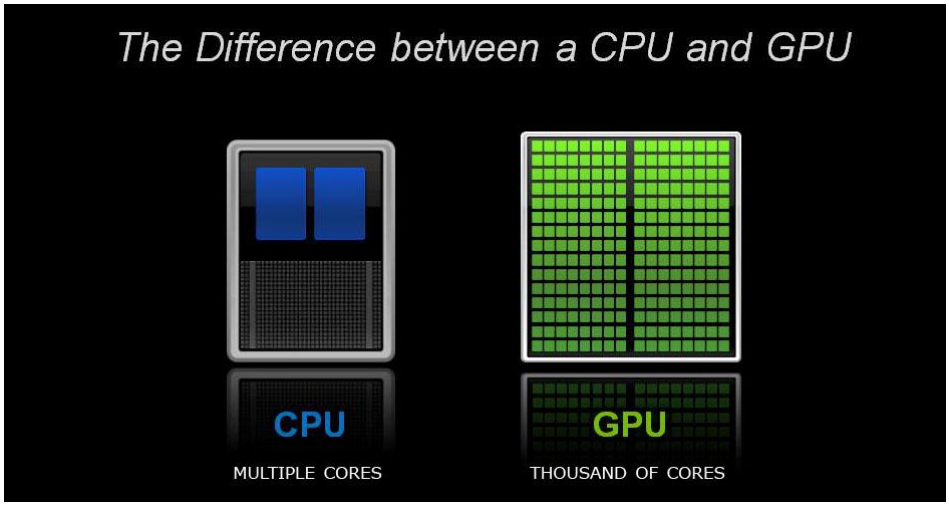

Figura 9[7] – cpu e gpu cores

Inicialmente a GPU tinha como função desenhar pixéis, hoje em dia é muito mais do que isso, trata-se um novo método de computação que é totalmente programável. Em conjunto, CPU e GPU, conseguem gerir de forma mais eficiente a execução de uma aplicação, graças a um modelo de programação que integra os dois componentes e à possibilidade de operações assíncronas.

Computação através da GPU, tem uma grande vantagem que, é ter acesso a um grande conjunto de processadores, tratam-se de componentes que são fortes em paralelismo e como tal suportam muitas threads em simultâneo. Para que isto seja possível é preciso um modelo de programação que possa de forma eficiente gerir todo este paralelismo. A mais conhecida é talvez a OpenCL, esta API serve para comunicar com os shaders de qualquer GPU. As GPUs da AMD utilizam em grande parte a API OpenCL, enquanto a NVIDIA tem a sua própria que se chama CUDA, a Microsoft tem um conjunto de várias, DirectX.

CUDA é uma linguagem de programação que tira partido do GPU possibilitando assim que os cálculos sejam feitos em paralelo de modo a serem executados de forma mais rápida. GPUs correm um kernel, cada kernel possui um conjunto de blocos que são compostos por um grupo de ALU(Arithmetic Logic Units). Cada bloco é composto por um conjunto de threads, que trabalham em conjunto para um valor.

[7] http://furryball.aaa-studio.eu/aboutFurryBall/whyGpu.html

O ray tracing é facilmente paralelizável de forma simples atribuindo uma tarefa a cada pixel: o código é o mesmo entre todas as tarefas, os dados comuns entre as várias tarefas são a descrição da cena que é igual para uma frame, e é preciso guardar o resultado final de cada pixel que é muito rápido comparado com os cálculos intermédios.

No mundo real as coisas são mais complicadas devido por um lado às questões de sincronização entre as várias tarefas [Unreal rendering], mas também devido às grandes diferenças de processamento que podem existir entre os vários pixéis: no caso de um pixel sem objectos é apenas necessário pesquisar a cena uma vez, mas no caso mais normal de múltiplos objectos e luzes cada intersecção vai gerar novos raios e a computação aumenta rapidamente.

Isto implica que é necessário ter sistemas avançados de balanceamento entre as várias tarefas e, eventualmente, gerar novas tarefas para os novos raios criados e as coisas ficam ainda mais complexas com outras optimizações necessárias para aumentar a velocidade de cada tarefa [Stéphane Marchesin and Catherine Mongenet and Jean-Michel Dischler].

De qualquer forma mesmo que o processamento de cada core de uma GPU seja relativamente modesto quando comparado com um core de um CPU a possibilidade de processar centenas ou até milhares de pixeis em paralelo é o que permite a renderização em tempo-real que vemos hoje em dia mesmo em computadores de secretária.

Aplique-se o mesmo princípio para "farms" de renderização com utilização de GPUs e os ganhos em termos de tempo e qualidade são evidentes mesmo para sistemas relativamente modestos [Construct, NVIDIA QUADRO 2015].

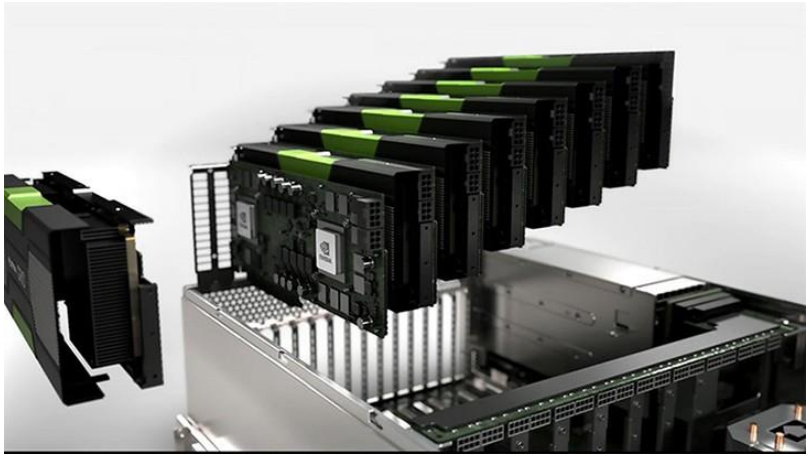

Figura 10[8] – nvidia renderfarm

[8] http://blog.miragestudio7.com/the-correct-graphic-cards-for-autodesk-3d-renderings-and-cloud-computing-render-farm/5770/

5 Global Illumination

Para se conseguir obter resultados mais realista em computação gráfica é preciso ter em atenção à iluminação, é ela que nos vai permitir visualizar os elementos presentes numa cena. Existem vários modelos de iluminação que usados em conjunto permitem obter resultados melhores, mas por outro lado afectam bastante o tempo de render. Existem dois tipos de iluminação: directa e indirecta.

A iluminação directa diz respeito apenas à iluminação que provem das fontes de luz (ex. spot light). Esta luz directa vinda de uma fonte de luz, segue num raio e vai iluminar um ponto de uma superfície ou de um volume. Neste modelo de iluminação é usado apenas a contribuição de cada fonte de luz para a iluminação final. Mas como é do conhecimento geral, o mundo que nos rodeia não tem apenas iluminação directa, conta também com a iluminação indirecta. Este modelo diz respeito a toda a iluminação que é reflectida numa cena, em computação gráfica o termo designado é Global Illumination.

Com este tipo de iluminação, a contribuição da luz que é reflectida de outros objectos é usada para calcular a iluminação final e para as cores dos pontos nos objectos que não estão a receber iluminação. Global Illumination ocorre quando a luz é reflectida ou é passada através de uma superfície para outra superfície. Como se trata de uma técnica bastante exigente a nível computacional, particularmente em cenas bastante detalhadas, Global Illumination tem sido primeiramente utilizado para renderizar cenas de computação gráfica exigentes, como filmes, usando render farms.

Reflexos, transparências e sombras, são exemplos de iluminação indirecta. Uma superfície que não esta exposta directamente a uma fonte que emite luz será visível se os objectos próximos dela estão iluminados. As múltiplas reflexões da luz desses objectos próximos combinadas produzem uma iluminação uniforme e designada por luz ambiente ou luz de fundo. As fontes de luz nada mais são que direcções na qual a luz atinge os objectos da cena.

A luz difusa é designada por reflexão difusa e resulta de várias características das superfícies: da sua rugosidade e granulosidade; superfície baça = reflexões difusas; superfície aparece igualmente brilhante de todas as direcções de visualização. Os pontos de luz criam também highlights ou manchas brilhantes designadas por reflexão especular, que é mais acentuado em superfícies brilhantes.

Figura 11 – iluminação indirecta, directa e global

Global Illumination diz também respeito a um grande conjunto de técnicas, Radiosity, Caustics, PathTracing, entre outras e cada uma com a sua especificidade para renderizar imagens. Nos tempos de hoje, existe muitas alternativas, variantes de iluminação global dinâmica e semi-dinâmica.

Precomputed light transport – este conjunto de técnicas consiste no pré processamento do transporte da luz a todos os pontos de um cenário estático. Um dos problemas deste tipo de técnicas é o facto de, praticamente, todo o conteúdo em computação é dinâmico, logo não faz muito sentido. Embora seja relativamente rápida para ser processada, dependendo claro da complexidade do cenário, ainda é usado nos dias de hoje, principalmente em vídeo jogos.

Instant radiosity based methods – consiste em representar a iluminação indirecta como um conjunto de VPL (virtual point light). Photons são traçados da fonte de iluminação para o cenário, cada intersecção de cada photon é tratado como um VPL (virtual point light) e para cada um deles é gerado uma aproximação de radiação difusa. A desvantagem está no facto de serem necessários vários shadow rays de maneira a se conseguir um render preciso, sem problemas.

Photon-mapping based methods – baseados na técnica de photon mapping, não são muito usados por requererem constante actualização dos mapas de photons, devido à dinâmica das cenas.

5.1 Radiosity

A produção de uma imagem renderizada, o mais perto possível da realidade, é umas das preocupações no campo da computação gráfica. Efeitos como luz especular, difusa, caustics, entre outros são essenciais para fazer com que uma imagem seja credível. Estes efeitos devem-se em grande parte às interações que a luz tem com as várias superfícies, e regra geral são bastante pesadas a nível de processamento. Tudo isto para um dia se vir a chegar a resolver a rendering equation.

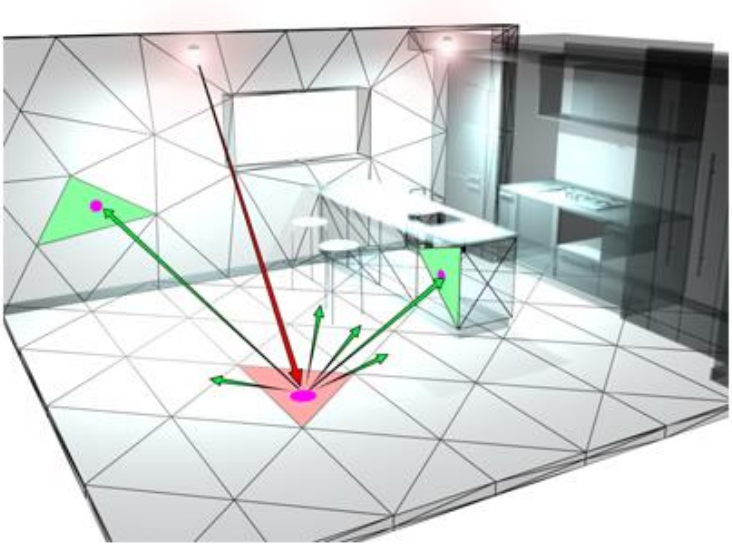

Figura 12[9] – radiosity

Uma das principais técnicas conhecidas para tentar resolver essa equação, RayTracing é capaz de produzir imagens com excelente luz especular. Por outro lado a renderização de luz difusa ambiente é bastante fraca, este facto vem comprovar assim o aspecto plástico das imagens produzidas.

Uma das alternativas para renderizar uma luz difusa com resultados bastante realista é a técnica Radiosity [Fredo Durand and Barb Cutler]. Esta técnica consiste em guardar informação dos valores da intensidade da iluminação nos objectos, à medida que a luz faz o seu percurso, partindo de uma fonte de iluminação.

Como foi referido no capítulo do estado da arte, grande parte das imagens produzidas que usam RayTracing têm um resultado bastante realista por são usadas várias técnicas em conjunto. Logo em conjunto RayTracing e Radiosity conseguem obter resultados muito bons, visto que cada técnica é forte num determinado modelo de iluminação.

[9] http://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/3DSMax-Archive/files/GUID-C5A3C77B-794B-4444-9783-7F2EA11C16BD-htm.html

Radiosity não renderiza a luz com base num número de raios, pelo contrário, o seu foco está em como a energia é transferida de patch para patch em cada polígono. Para isso cada polígono tem de ser subdividido de forma a conter mais divisões nas suas faces. Logo em Radiosity não se fala em luz mas sim em transferência da energia, como base está a transferência do calor que é o estudo da emissão e transmissão do calor e a luz trata-se de uma espécie de calor, energia. [Modeling Global Illumination with radiosity, 3DS MAX 2015]

Portanto a Radiosity de um patch consiste na quantidade de energia que deixa esse patch num dado momento, pode ser considerado o brilho desse patch. Em Radiosty não se distingue objectos que reflectem luz, nem fontes de iluminação, uma vez que cada patch pode emitir luz. Trata-se de um processo que é repetido várias vezes, o que faz com cada vez mais patchs sejam iluminados, resultando numa iluminação mais suave. Em contra partida é pesado, e não é usado em tempo real, mas pode ser pré processado para usar depois numa cena em tempo real. [Modeling Global Illumination with radiosity, 3DS MAX 2015]

5.2 Photon mapping

Muitos dos complexos algoritmos baseados em RayTracing podem ser considerados, algoritmos de Monte Carlo, que dizem respeito a um conjunto de algoritmos que usa repetições aleatórias de amostras como forma de obter resultados. Um dos grandes problemas destes algoritmos de Monte Carlo RayTracing é a quantidade de granulado na imagem, efeito bastante percetível quando se está a renderizar caustics.

Uma solução passa por aumentar o número de amostras, mas a complexidade aumentaria o que por consequência levaria a tempos de render mais elevados. Um modelo de iluminação global que que corrige esses problemas, contêm ainda todos os efeitos de iluminação global e que consome menos memória é o PhotonMapping.

Numa primeira fase, são lançados vários photons das fontes de iluminação, estes photons são lançados com uma intensidade, representada pela cor. As fontes de luz suportadas por este modelo são praticamente todas, luzes de ponto, luzes direcionais, luzes de área, independentemente da forma da luz a função é sempre a mesma, lançar photons em todas as direcções de forma uniforma para a cena. Assim que um photon é lançado, este salta de objecto em objecto, conforme o número de bounces lhe é atribuído e pode ser absorvido ou reflectido dependendo do material de cada objecto. A técnica usada para determinar o que acontece com um photon quando este colide com um objecto, chama-se Russian Roulette. Na segunda fase a cena é renderizada com base na informação que é obtida através do mapa de photons.

Dependendo do tamanho da cena em que está a ser usada esta técnica, alguns photons podem ser lançados na direcção oposta dos objectos da cena, isto irá fazer com que o número de cálculos aumente, para evitar e optimizar o processo de render, é possível lançar photons apenas na direcção da geometria. Para calcular caustics, reflexos e refracções são criados mapas à parte.

De maneira a optimizar este processo, existe uma técnica que se chama Irradiance Caching [M.C. Escher, 1898–1972] que consiste em obter a informação de cada photon com base nos photons mais perto, resultado assim numa iluminação indirecta mais suave.

6 Anti-Aliasing e texturas

6.1 Anti-aliasing

Anti-aliasing (AA) trata-se de uma técnica usada em computação gráfica, que tem como objectivo minimizar o efeito serrilhado dos objectos. Este efeito é visível devido às baixas resoluções usadas nos motores de render 3D. Quando AA é usado a qualidade de imagem aumenta substancialmente, mas em contra partida o cálculos necessários para renderizar uma imagem são bastante elevados, levando assim a que a performance seja afectada.

Existem várias técnicas alternativas de AA, mas todas tem o mesmo principio renderizar vários pixels por cada pixel da imagem final. A diferença entre as várias técnicas encontra-se na forma como determinam que pixels são "aliased" e como misturam os vários pixels para calcular o pixel final. Muitas das várias alternativas de AA, têm no seu nome a quantidade de pixels usados para cada pixel, por exemplo, 2x, 4x ou até 8x.

AA está divida em duas formas de aplicação, durante o render ou depois do render. Aplicado durante o render produz-se melhores resultados mas com um impacto na performance maior. Como efeito aplicado depois do render, o impacto é muito menor, a qualidade é pior.

- Multisample Anti-Aliasing (MSAA) – este tipo de anti-aliasing trata-se de uma versão barata de supersampling, ao contrário de Supersample Anti-Aliasing (SSAA), MSAA optimiza todo o processo avaliando cada pixel apenas uma vez. Apenas pixels que necessitam de um aliasing maior é que são alvos de SSAA. O resultado de MSAA melhora a qualidade de imagem com um impacto pequeno na performance, mas a qualidade é também inferior a SSAA.

- Temporal Anti-Aliasing (TXAA) – trata-se de uma técnica, desenvolvida pela NVIDIA, que reduz temporal AA. Usa a base de MSAA mas com um conjunto de filtros, que lhe transmite aquele aspecto de filme. [TXAA NVIDIA, 2015]

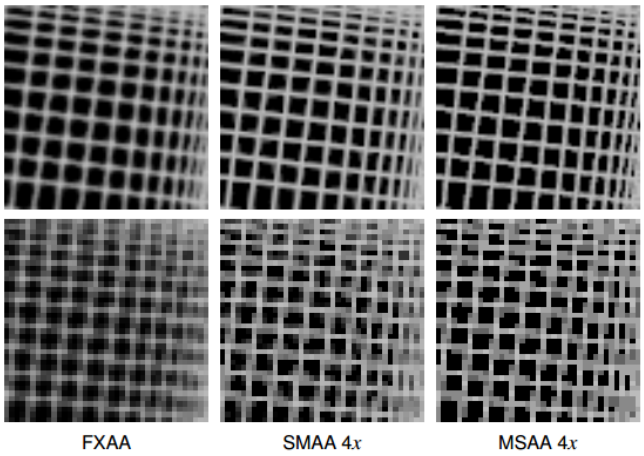

Figura 13[10] – fxaa vs 4x smaa vs 4x msaa

- Coverage Sampling Antialiasing (CSAA) – técnica desenvolvida pela NVIDIA concorrente ao MSAA. Em teoria esta técnica equivale a 8x de MSAA, usando recursos equivalentes a 4x MSAA. Este resultado é conseguido porque são reduzidos o número de características para cada sample; [CSAA, Peter Young, NVIDIA]

- Supersampling (SSAA) – também conhecido como Fullscreen Anti-Aliasing tem vindo a ser substituído por MSAA, em causa está o impacto grande na GPU. Comparado com MSAA, SSAA produz a imagem produzida é mais suave e realista. SSAA é a técnica original de AA, em grande parte dos casos onde é aplicada pode-se considerar, que renderiza uma imagem a uma resolução superior e depois faz down sampling, ou seja, diminui a resolução da imagem. Por exemplo uma imagem renderizada a 2x SSAA a 800x600, significa que foi renderizada a 1600x1200 e depois diminuída para 800x600. Isto significa que para cada pixel da imagem, foram renderizados quatro pontos e depois combinados.

- Subpixel Morphological Anti-Aliasing (SMAA) – versão melhorada de Morphological Anti-Aliasing (MLAA), é aplicada como post-process e produz melhores resultados do que FXAA; [SMAA, Crytek 2012]

- Fast Approximate Anti-Aliasing (FXAA) – trata-se de uma versão post-process de AA, ou seja, o AA é aplicado apenas depois da imagem renderizada, ao contrario das técnicas melhores como por exemplo MSAA. Tem uma grande compatibilidade visto que se trata de um efeito aplicado posteriormente. Contudo a melhoria não é muito substancial comparado com as outras técnicas. [Timothy Lottes, 2009]

[10] http://bondarev.nl/?p=67

6.2 Processamento de texturas

Considerada uma das grandes inovações, na área da computação gráfica, o processamento de texturas veio trazer um detalhe grande, às imagens renderizadas. A partir de um array de pixels, pinta esses mesmos pixels numa superfície trazendo assim outro realismo. O array de pixels consiste numa imagem, que pode ser uma textura de roupa, madeira, ou muitas outras superfícies. Bump mapping altera a superfície de um objecto, dando a ilusão de diferentes relevos na superfície. A vantagem do uso de texturas é que adiciona muito mais detalhe a um objecto, enquanto o tempo de render aumenta muito pouco.

Trata-se de uma operação de rasterização, onde são aplicadas texturas a superfícies 2D de um objecto 3D e depois o sistema gráfico trata de mudar cada pixel, a este processo chama-se "scan conversion". Como o processamento das texturas acontece durante a rasterização da imagem, durante o mesmo render a GPU tem de:

- Determinar as coordenadas dos pixels de cada canto;
- Determinar os pixels das arestas do polígono;
- Determinar a cor dos pixels das arestas na forma de uma linha;
- Determinar a cor dos pixels que estão abaixo dessa linha;

Existem coordenadas do mundo e coordenadas do objecto e ambas são importantes para mapear as texturas. Nas coordenadas dos objectos a origem e as coordenadas dos eixos, permanecem fixas independentemente da posição e orientação do objecto. Grande parte das técnicas de mapeamento usam coordenadas do objecto caso contrário o resultado não seria o esperado, como se pode verificar abaixo. Quando se usa coordenadas do mundo, a textura espalha-se pelo objecto à medida que este se move pelo mundo. [Fundamentas of Texture Mapping and Image Warping, Paul S.Heckbert, 1989]

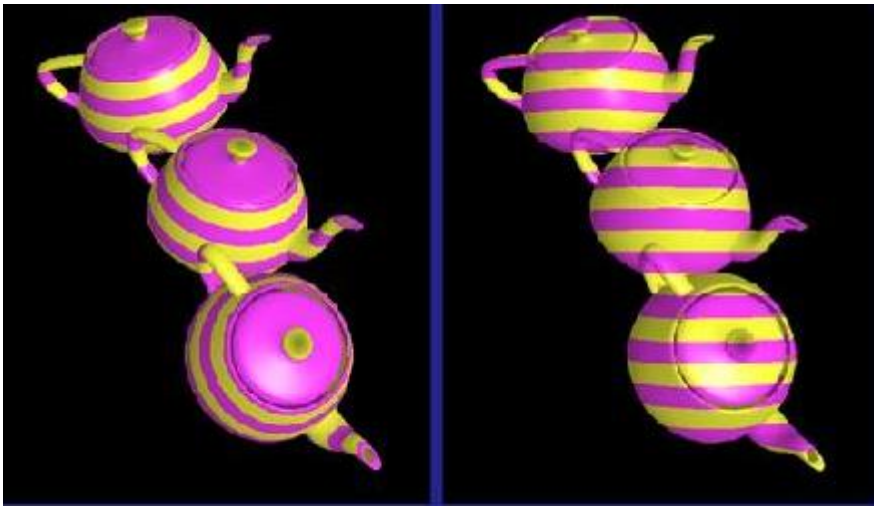

Figura 14[11] – coordenadas do objecto / coordenadas do mundo

Two dimensional mappings são usados para definir os parâmetros de uma superfície e para descrever as transformações entre, o sistema de coordenadas das texturas e o sistema de coordenadas da cena. [Fundamentas of Texture Mapping and Image Warping, Paul S.Heckbert, 1989]

11 https://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_1.htm

7 Criação do cenário

O lançamento do Unreal 4 teve um grande impacto na indústria de computação gráfica, principalmente na área dos jogos, devido às suas capacidades que parecem ser infinitas. O sistema de Physical Based Rendering e o método de importação fácil, como forma de importar modelos 3D para o Unreal Engine foi uma das razões que me levou a - para além da qualidade que se consegue obter - escolher como forma a mostrar que se trata de uma alternativa viável para criar cenários 3D.

O Blender e o Unreal são dois programas diferentes mas em conjunto podem servir para criar cenários multimédia bastante bons. Embora possam ser feitas as mesmas coisas no Blender ou no Unreal, cada um deles é específico numa parte. Ambos têm o seu próprio motor, ambos servem para criar animações, modelar, entre outras tarefas. Ambos possuem um motor de jogo, embora o do Blender esteja muito ultrapassado, a modelação é permitida nos dois, mas a do Unreal é muito limitada, na área da animação as semelhanças já são maiores, ambos possuem funções muito avançadas nesta área. Para tratar das texturas o Blender também é o melhor, embora o Unreal seja bastante semelhante.

O foco desta tese está em mostrar o mesmo cenário a correr em motores diferentes, em técnicas diferentes, em programas diferentes, mostrar o forte de cada programa. Como já foi descrito nesta tese, o Blender usa o render Cycles que tem como base a técnica PathTracing, enquanto o Unreal usa um conjunto de várias técnicas, entre as quais está a light propagation volumes.

Real time PathTracing é algo que ainda não é bem possível, pelo menos no Blender, ou seja, criar cenários usando iluminação global e usá-los para correr em tempo real, está fora de questão. Por outro lado o Unreal que não é baseado em nenhuma variante de RayTracing, é muito melhor para correr em tempo real, mesmo com a sua versão de iluminação global.

Neste capítulo vai ser detalhado todo o processo de criação, começando pela modelação no Blender bem como todo o processo de criação do render final. Vai ser ainda descrito a passagem do cenário para o Unreal e a criação da cena final no mesmo.

7.1 Blender

7.1.1 Modelação

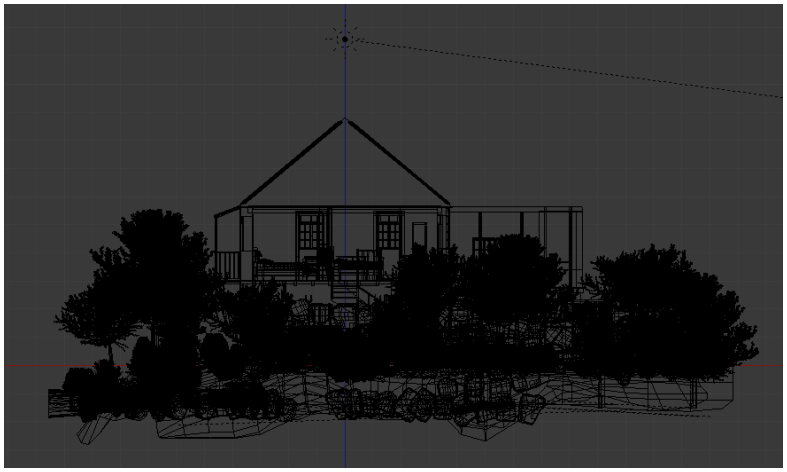

Figura 15 – vista frontal do cenário

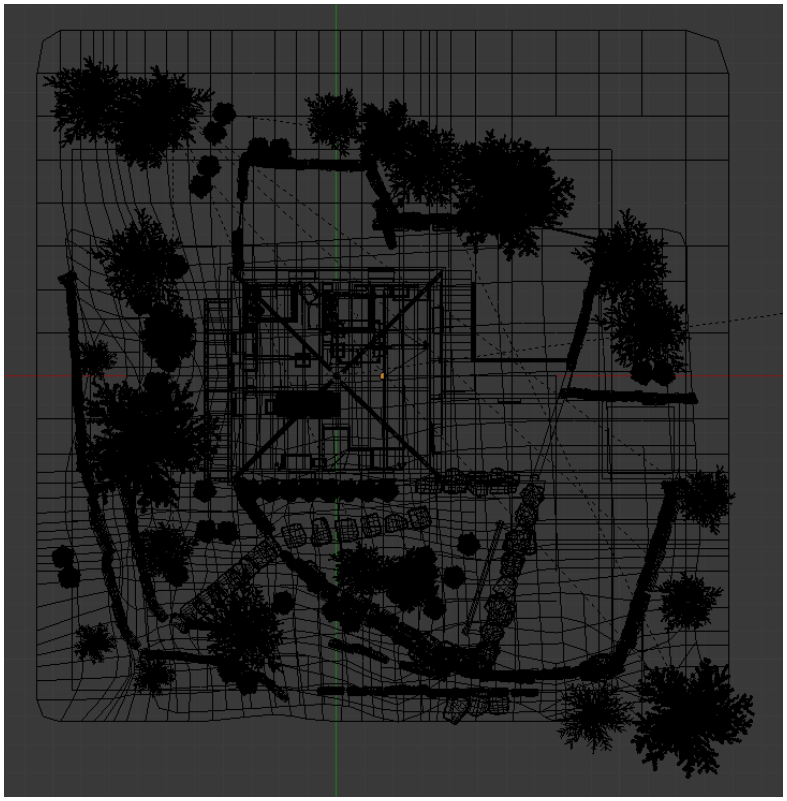

Figura 16 – vista superior do cenário

O processo de modelação foi todo feito no *Blender*, como foi referido acima, todos os objectos foram trabalhos de forma a obter modelos realistas e proporcionar uma visão realista do cenário. Desta maneira o uso de texturas para "simular" algum relevo, como por exemplo *bump textures* nos vários modelos não foi preciso. Como se pode ver nas seguintes figuras é possível comprovar isso mesmo, dá para verificar a existência de bastantes objectos.

Alguns objectos foram modelados com recurso a background image ou então "livremente". Para modelar alguns detalhes, em certos objectos, o uso de background image ajuda bastante a este processo, pois permite ter uma visão exacta das várias vistas disponíveis, ou seja, da topo, lateral e frontal bem como os detalhes dos objectos. A imagem de background foi inserida carregando na tecla de atalho N, depois foi preciso carregar no botão background image depois escolher a imagem que se pretende. É possível depois atribuir uma imagem a um ponto de vista específico, ou então ao conjunto de vistas. Esta possibilidade de usar imagens no background podem usar bastante em objectos bastante detalhados e permite também passar o mais parecido possível um objecto que foi desenhado num papel. As imagens, usadas para background, foram obtidas do Google ou então desenhadas em programas, como por exemplo Photoshop.

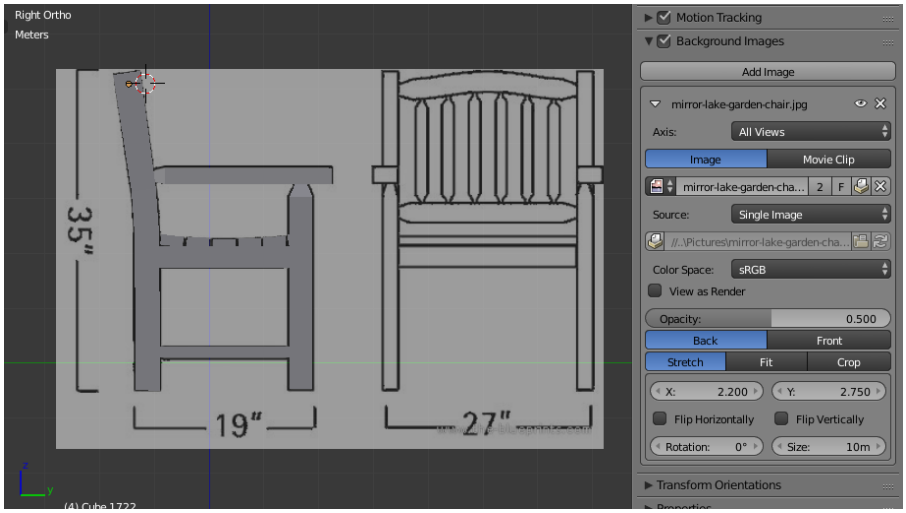

Figura 17 – background image

No Blender existe uma secção, com o nome de modifiers, que permitem ao utilizador uma modelação mais rápida e mais correcta da cena. Foram usados vários desses modifiers que possibilitaram assim um processo de modelação bastante melhor. O uso destes não é obrigatoriamente necessário mas o uso deles possibilita uma liberdade maior quando se está a modelar um objecto. Outra vantagem prende-se com o facto de quando se está a tentar modelar objectos complexos, com a ajuda dos modifiers essas dificuldades são diminuídas.

Um dos mais usados foi o modifier mirror pois permite replicar o modelo 3D, no qual está aplicado nas várias coordenadas, desta maneira os objectos são simétricos, por exemplo não existe cadeiras com pernas de diferentes tamanhos, entre outros. Para usar este modifier é preciso selecionar o objecto, entrar em edit mode, dividir ao meio o objecto, por exemplo inserindo um conjunto de vértices no centro do objecto, com as teclas de atalho ctrl+r e depois eliminar um dos lados. Desta maneira fica apenas disponível um dos lados do objecto que está a ser modelado.

Depois, no separador dos modifiers escolher o que diz mirror, escolher o eixo que se pretende reproduzir, existe depois ainda a possibilidade de fazer merge aos vértices que se encontram no meio, de maneira a que no centro do objecto não existam vértices duplicados e existe também a possibilidade de prevenir a modelação para o lado onde está a ser replicado o objecto. Este modifier é muito usado e permite ao utilizador modelar um objecto sem se preocupar com a simetria do modelo, focando assim toda a atenção em apena um dos lados sabendo que o outro lado vai ser exactamente igual.

Mas no meio destas melhorias todas, existem também alguns “problemas”. Sempre que está concluída a modelação do objecto, é preciso carregar em apply de maneira a confirmar o modifier mirror. Caso não seja feito o apply, o mesmo objecto quando é exportado para outro programa, vai aparecer todo deformado visto que o seu processo de modelação não foi concluído. Existe ainda outro problema, que é quando o utilizador carrega no apply, mas depois mais tarde decide alterar a geometria do objecto. O utilizador tem assim de aplicar outro modifier mirror, ou então faz as alterações necessárias com a dificuldade de manter sempre simétrico o objecto. Por experiência própria, antes de exportar cada objecto faço apply ao objecto, desta maneira está sempre disponível para fazer alterações no objecto.

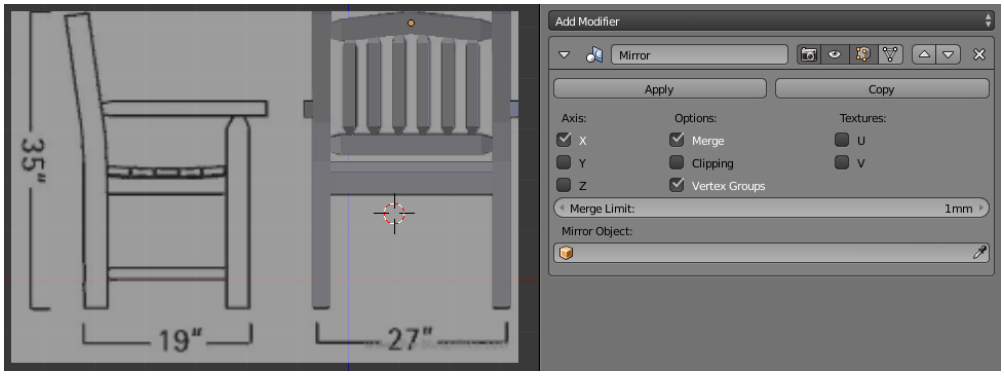

Figura 18 – modifier mirror

Outro modifier usado foi o subsurface que consiste em suavizar uma mesh e para isso criar mais subdivisões nas faces, quanto mais subdivisões forem criadas mas pesado fica o objecto mas em contra partida fica com mais detalhe. Este modifier foi usado em quase todos os objectos da cena, pelo menos o primeiro nível foi activado. Uma forma rápida de activar este modifier, é usando as teclas de atalho ctrl+1, onde o número significa o número de subdivisões. Quantos mais níveis forem aumentados, mais pesado fica o objecto mas do ponto de vista visual fica mais agradável. Este modifier é bastante útil, o nível básico é um ponto essencial para o objecto ter um bom aspecto, níveis superiores a 2 ou 3 são um pouco desnecessários porque a diferença é impercetível e o impacto na performance é enorme.

O problema é que este, modifier, requer que o objecto seja subdividido, ou seja, contenha mais vértices caso contrário pode piorar o aspecto bastante e não obter o resultado pretendido. Por exemplo, um cubo com o modifier subsurface no valor 1 ou 2, vai parecer uma bola, para corrigir isto são necessários mais vértices. Para aumentar o número de vértices, o utilizador pode usar as teclas de atalho ctrl+r para definir pontos específicos, ou então usar a teclas w e escolher subdivide. Portanto no geral o objecto fica com melhor aspecto, mas fica também mais pesado. Tal como o modifier mirror, este também necessita de apply para concluir o processo. No geral este problema está presente em todos os modifiers.

Em object mode o objecto aparece com o modifier activo mas quando se entra em edit mode é criada uma "layer" que permite editar o objecto sem o modifier a interferir com a edição. Em edit mode podemos comprovar que o objecto continua igual, mas no resultado final, o render, o objecto aparece com uma modelação "suave". É possível ainda especificar o nível que queremos para o render final e para o uso em preview, de maneira a que quando o utilizador está a modelar o objecto possa ter uma vista do resultado final mas ao mesmo tempo não perca a nível de performance.

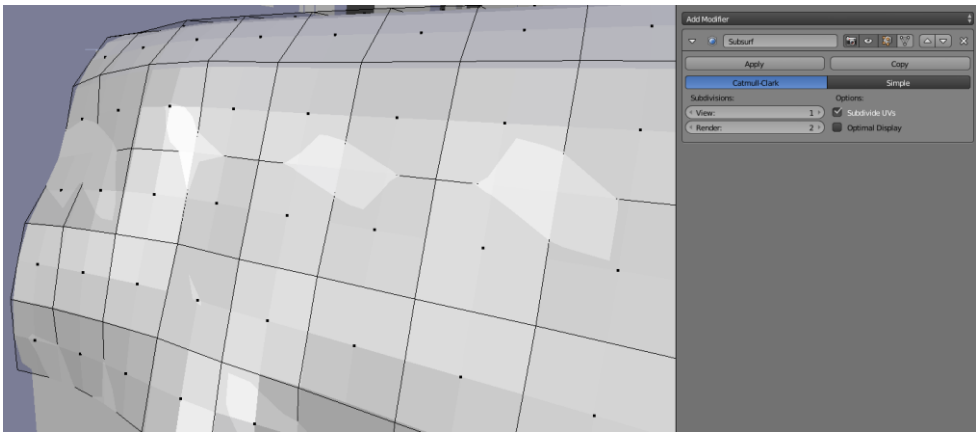

Figura 19 – modifier subsurface

O modifier multiresolution permite tal como o subsurface subdividir a mesh de um objecto mas permite depois que esse detalhe seja trabalhado, basicamente permite modelar um objecto com um detalhe superior. Este modifier é muito parecido com o subsurface pois suaviza o objecto e ainda adiciona subdivisões ao mesmo. A utilização de ambos é possível ou apenas de um só, o multiresolution é mais completo enquanto o subsurface exige a subdivisão posterior do objecto, é uma questão de escolha. Como se pode ver na imagem abaixo existem alguns campos extra em relação ao subsurface, com principal destaque para o valor do sculpt.

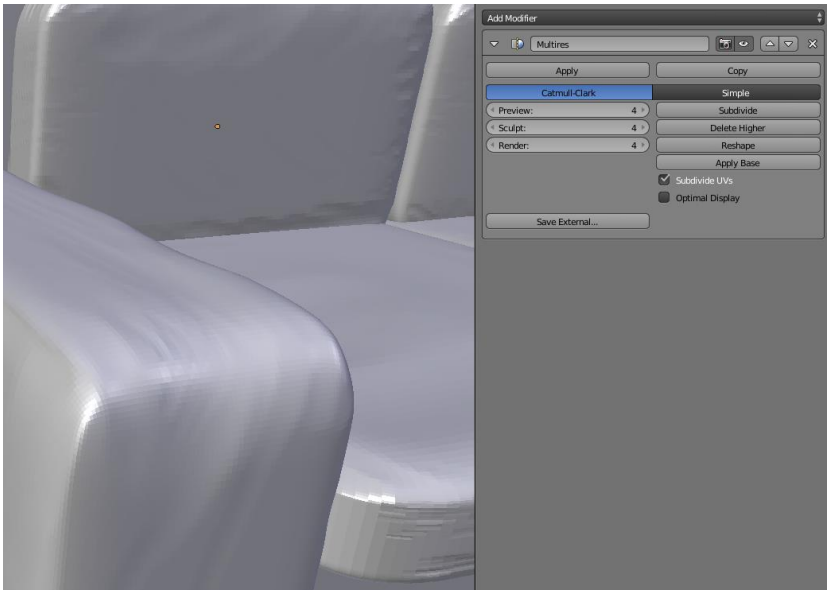

Figura 20 – modifier multiresolution

Este valor vai definir o nível de resolução a ser utilizado no modo sculpt que foi bastante usado para modelar o objecto sofá. Começou com um quadrado normal depois procedeu-se à modelação básica, sem grandes detalhes. Depois de modelado aplicou-se o modifier multiresolution que permite como o próprio nome indica multiplicar a resolução. O modo sculpt foi depois usado para adicionar o detalhe, a partir de um objecto modelado com o modifier multiresolution é possível neste mesmo modo “esculpir” o modelo. Existem vários modos de sculpt com diferentes efeitos no objecto, como se vê na imagem abaixo, cada um com o seu efeito.

Figura 21 – modo sculpt

Estes vários perfis disponíveis permitiram tornar o que era um conjunto de quadrados sem detalhe em algo mais agradável, mas com um custo a nível de performance algo elevado. Este modo é muito bom para adicionar pequenos detalhes, era possível também apenas com um quadrado modelar o sofá inteiro mas ia requerer muito mais tempo.

Outro objecto que podia ter sido modelado usando o modo sculpt era os objectos com tecidos, mas não é a forma mais eficaz uma vez que existe no Blender a possibilidade de simulação. Em conjunto com a simulação cloth usa-se também a simulação de collision que permitir definir que objectos são activos ou passivos, desta maneira podemos ter interacção entre objectos. Um ter um bom resultado quando usa a simulação cloth, precisa de ser subdivido, pois quantos mais vértices tiver é permitido ao objecto deformar-se de forma mais realista. Depois de no último separador, physics, activar-se a opcção cloth podemos depois definir um conjunto de parâmetros. O Blender já trás por defeito alguns tecidos predefinidos o que facilita a tarefa de termos de escolher que campos precisamos de alterar. Portanto depois de escolhido o tecido, podemos então alterar alguns valores que são gerais a todas as opções de tecidos, como comprova a imagem abaixo. Quanto maior o valor dos steps, quality melhor qualidade tem a simulação, os valores do grupo material e damping são alterados com base no preset que se escolhe, mas também podem ser alterados. Os campos start e end definem o número de frames para a animação que começa quando se carrega em bake.

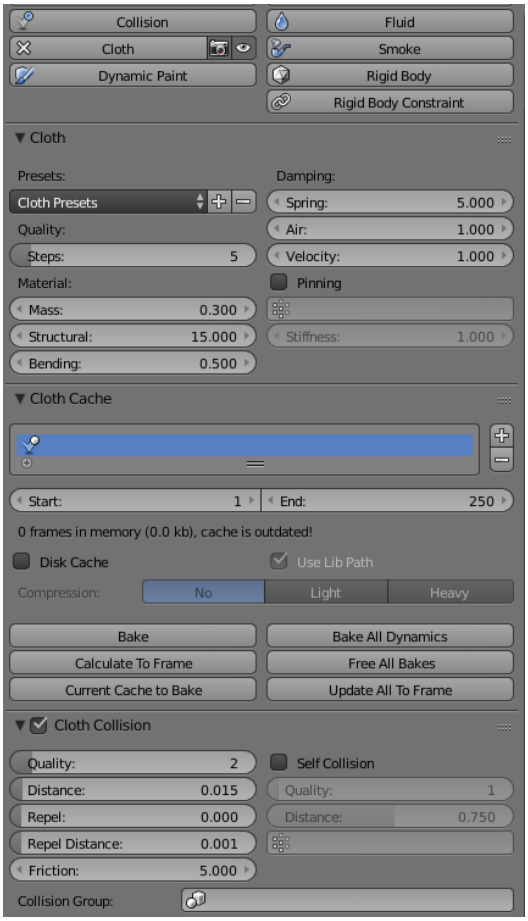

Figura 22 – características da simulação cloth

Depois do processo de bake acabar é possível na timeline escolher o momento em que o tecido se encontra na forma que queremos, encontrado o momento é depois necessário fazer apply no modifier. A simulação cloth permite fazer muitas animações, tem o factor contra de ser pesada mas trata-se de uma ferramenta poderosa.

A vegetação do cenário foi dividida em duas partes, a parte do solo e a parte das árvores. A parte do solo foi feita a partir de um plano que através do modifier subsurface permitiu modelar os diferentes declives do cenário, desta maneira foi possível escolher os vértices no ponto certo de maneira a ficar como foi previsto. A relva foi criada através do sistema de partículas do Blender, como mostra a imagem abaixo. Depois de selecionado o objecto ao qual se pretende adicionar as partículas, escolhe-se o tipo que neste caso foi hair depois com a opção advanced activada é possível ter acesso a um conjunto de campos que permitem uma maior personalização. O campo hair length permite definir o tamanho enquanto o number permite definir quantas partículas vão ser renderizadas. Outros campos necessários para se conseguir o resultado semelhante a relva são o root e o scaling. Estes vão definir o tamanho da origem da partícula na face do objecto, valores pequenos tamanho pequeno, foi o escolhido.

Figura 23 – particle system no blender

A parte das árvores foi realizada graças a um plugin, de nome sapling tree, que permitiu criar vegetação credível, caso contrário a vegetação seria criada com poucos planos, com as texturas certas mas sem o mesmo resultado. Contudo o detalhe que este plugin permite tem um impacto muito grande no número de vértices e por consequência na performance.

Este plugin permitiu criar vegetação bastante realista, que se fosse realizada "manualmente" seria um processo muito demoroso. O conjunto de opcções é variado, desde o número de ramos, o angulo das folhas, entre outras opcções.

Durante o processo de construção da árvore, houve alguns problemas. O processo de construção tem de ser feito de uma vez só, caso contrário não se pode editar mais, o que implica recomeçar o processo todo uma vez mais. Por exemplo, se sair do edit mode da árvore e editar outro objecto qualquer já não é possível voltar a editar a árvore. No início cometeu-se esse erro algumas vezes, mas com o treino foi corrigido. Mas como são poucos os passos para se conseguir uma vegetação realista, o processo não é muito penoso e assim vai-se conhecendo melhor cada opcção do plugin. Depois de concluído o processo todo, pode-se então definir o respectivo shader.

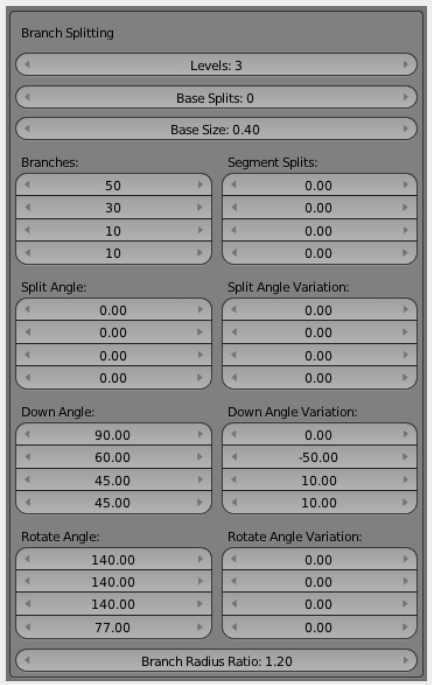

Figura 24 – sapling tree plugin

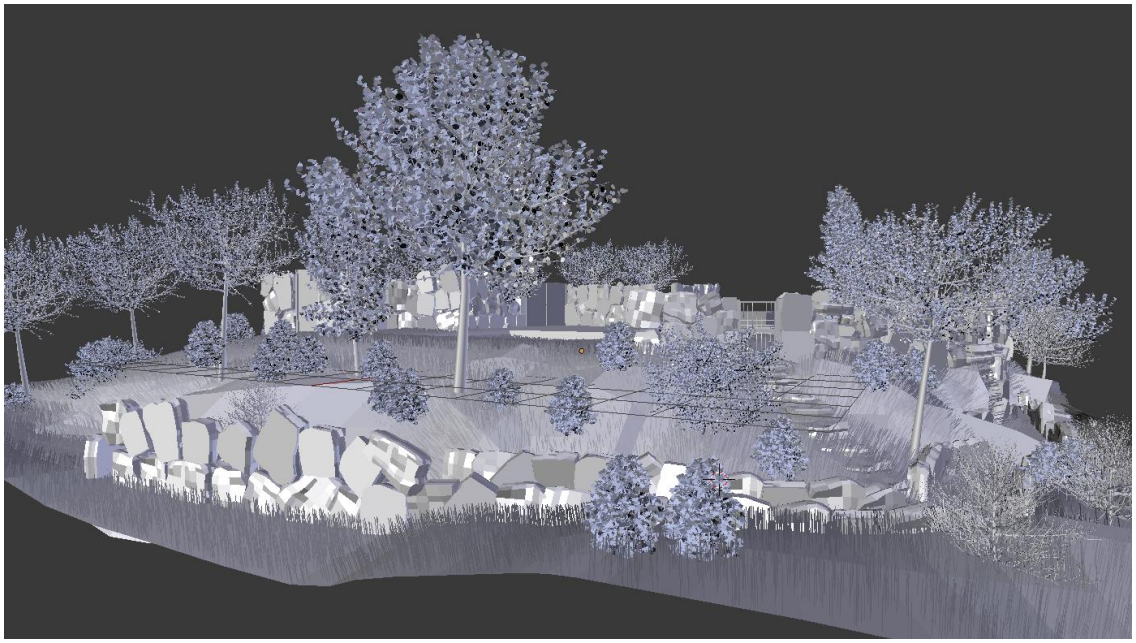

Figura 25 – vista solid do cenário

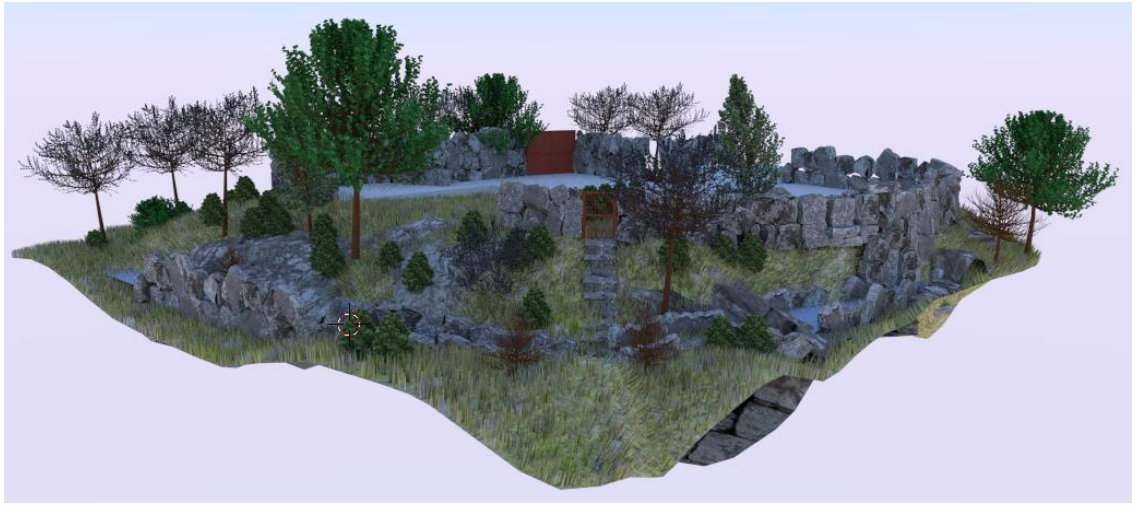

Figura 26 – vista final da vegetação

7.2 Blender/Unreal

7.2.1 Transformação Blender/Unreal

Para que todo o cenário funcione correctamente no Unreal é preciso ter em atenção alguns aspectos. Tanto o Blender como o Unreal tratam de maneira diferente algumas características dos objectos, o que pode levar a que o resultado final não seja o mesmo nos dois programas. É preciso ter em atenção à conversão que vai existir de escala, à modelação do objecto, às texturas, ao pivot point de cada objecto, aos materiais. Estes foram os cuidados necessários para uma correcta exportação dos modelos, mas existem outros cuidados a ter caso se esteja a fazer animação, por exemplo nas colisões.

Na conversão de escala o problema está no facto de uma unidade no Blender corresponde a um centímetro no Unreal, logo isto vai fazer com que o objecto fique muito pequeno no Unreal. Para corrigir este problema existem várias soluções, fazer scale ao objecto no Unreal, ou seja, aumentar o valor de scale para 100 permitindo assim manter o mesmo workflow no Blender. Outra solução passa por definir o valor de scale quando se está a fazer import/export, ou seja, quando se fizer export do Blender para o Unreal definimos o valor de 100, quando se fizer import para o Unreal definimos o valor de 0.01. Esta não é a solução mais indicada, uma vez que o FBX import/export do Blender não é 100% compatível com o Unreal. Para além das soluções já referidas, existe ainda a possibilidade de fazer scale no próprio Blender usando o modifier scale ou ajustando no separador scene as unidades para Metric e definir o valor de 0.01. Usando o modifier scale pode trazer problemas caso o utilizador não faça apply, o que impede que seja feito o export, alterando o valor das unidades provoca a alteração do workflow uma vez que tudo vai ficar mais pequeno. A solução escolhida foi fazer scale no Unreal, ou seja, aumentar o valor para 100.

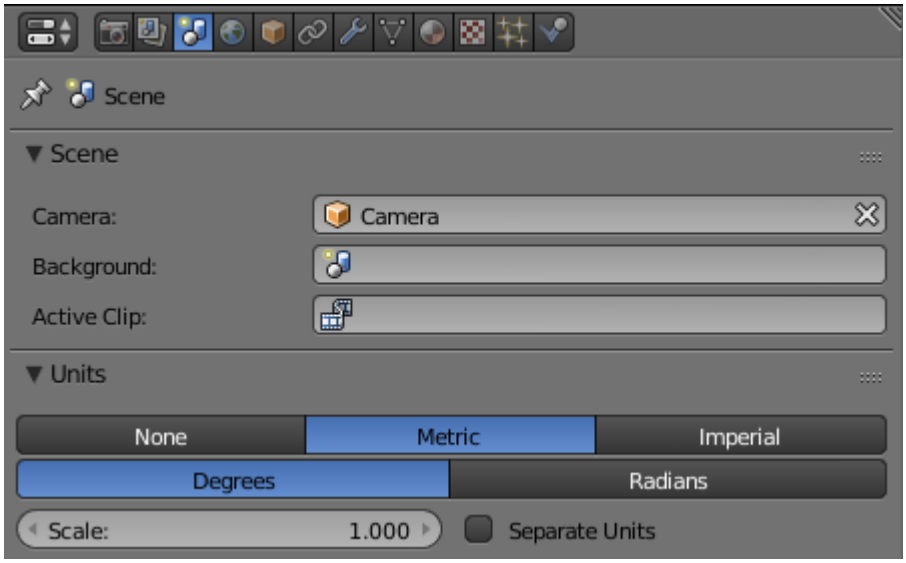

Figura 27 – scale do cenário no blender

Na parte de modelação é preciso ter atenção à técnica usada para a modelação de maneira a não afectar os mapas UV´s. Podem ser usadas nurbs, bezier, modifiers ou outras técnicas que sempre que se faz export de um objecto, o Blender converte para mesh, mas é recomendado que se faça a conversão manualmente para que os mapas UV´s estejam correctos. Outro cuidado a ter na modelação diz respeito à exportação de vários objectos na mesma layer. Cada objecto tem de estar isolado, ou seja, numa layer própria, caso contrario ao ser importado para o Unreal vai existir apenas um pivô para todos os objectos. De maneira a evitar o alargado número de layers necessárias para todos os objectos, usou-se um plugin que permitiu exportar apenas os objectos seleccionados.

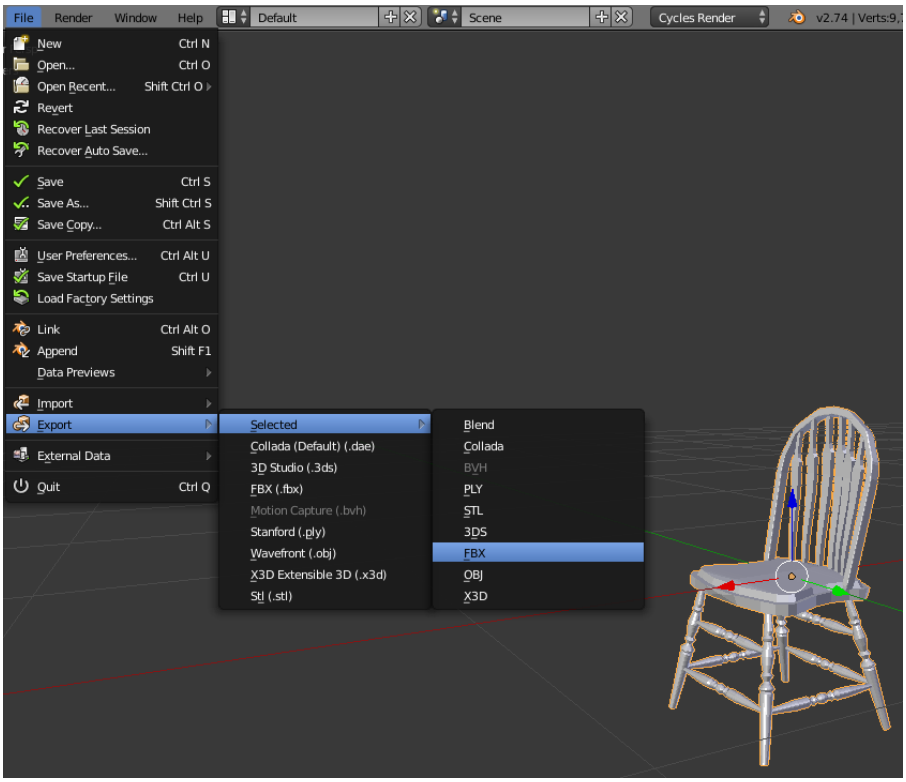

Figura 28 – export selected object no blender

Relacionado com o problema de modelação está o problema dos pivôs de cada objecto, estes têm de ter o pivô point nas coordenadas (0,0,0) caso contrário quando o objecto for importado para o Unreal o ponto vai estar numa posição incorrecta o que pode levar a dificuldades para posicionar o objecto na cena. Com as teclas de atalho, alt+g é possível centrar um objecto nas coordenadas (0,0,0) e depois com as teclas ctrl+alt+shift+c centra o objecto nas mesmas coordenadas.

Mesmo com o objecto nas coordernadas(0,0,0), ainda existe mais um detalhe para mais tarde facilitar o uso do objecto noutro programa, que é definir o pivot na base do objecto. Desta maneira ao mexermos no pivot do objecto temos uma guia o que facilita no correcto posicionamento, evitando assim ter objectos a “voar”. A imagem abaixo mostra um exemplo correcto.

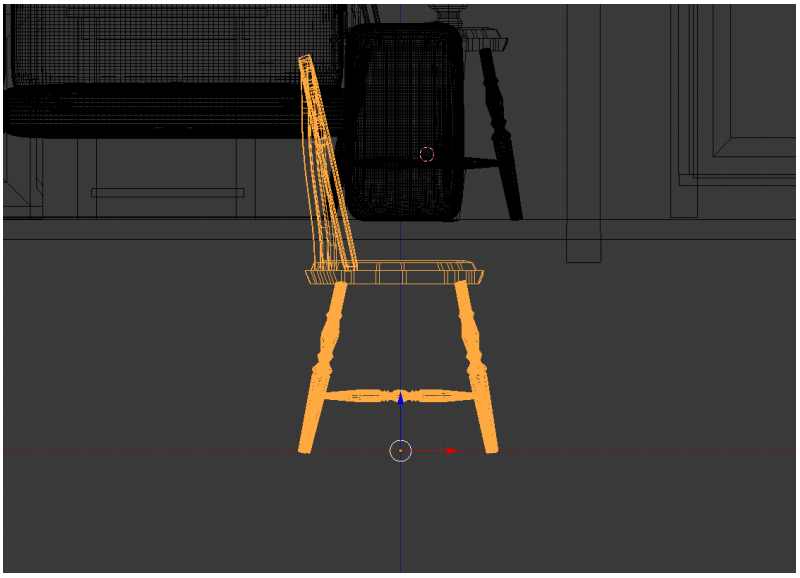

Figura 29 – objecto nas coordenadas (0,0,0) com o pivot na base

As texturas também podem ter problemas quando são importadas para o Unreal, para isso é preciso fazer o unwrap das mesmas. Pode-se usar o método manual, que é o mais aconselhado, pois permite mais controlo sobre o mapa UV, ou então o método automático, Smart UV Project o que nem sempre resulta num bom resultado. É necessário apenas um mapa UV para todas as texturas, base color, normal, etc.

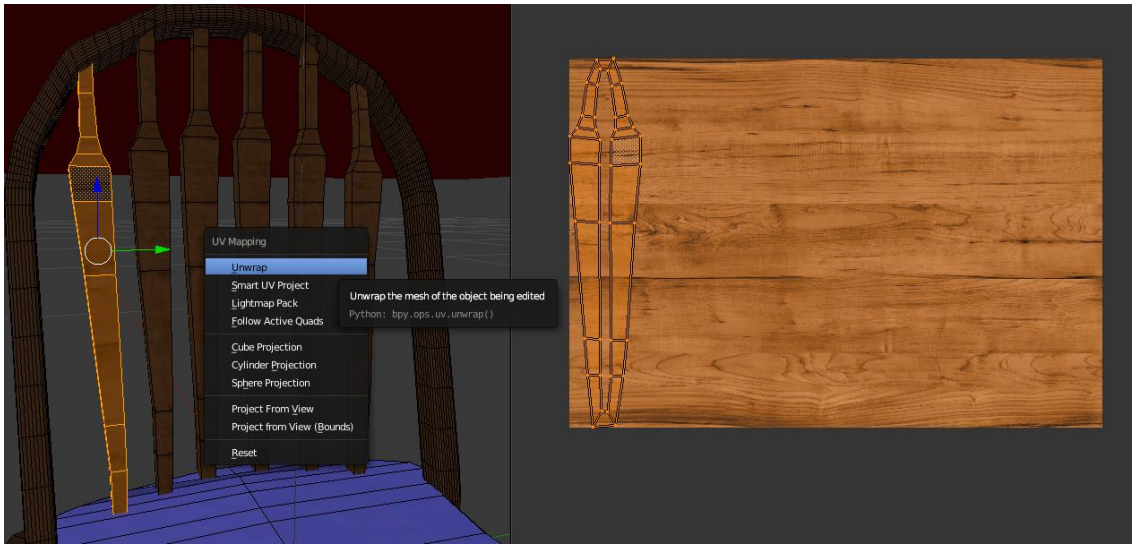

Figura 30 – mapas uv

Os materiais funcionam com algumas diferenças em ambos os programas, portanto é aconselhável que se crie o material para o programa que se está a usar, embora algumas alterações possam funcionar nos dois. Como tal, os materiais criados no Blender, não vão ter o mesmo aspecto no Unreal, embora o processo de criação seja bastante semelhante. O uso de vários materiais no mesmo objecto requer a criação de placeholders no Blender e que vão aparecer no Unreal como elements. Uma maneira de saber que elementos correspondem a cada material consiste em atribuir o material criado no Unreal ao elemento e verificar se o resultado foi o mesmo criado no Blender.

7.2.2 Materiais/Texturas

Depois do processo de modelação estar devidamente preparado, seguindo alguns cuidados a ter passa-se para o processo de atribuir materiais e texturas aos objectos, criando assim os shaders. Esta fase é importante pois pode servir para criar a "ilusão" de um modelo 3D bastante detalhado, para compensar alguma modelação menos detalhada, visto que o número de vértices já era excessivo. Tanto o Blender como o Unreal suportam vários tipos de materiais e texturas, mas o processo de criação é ligeiramente diferente.

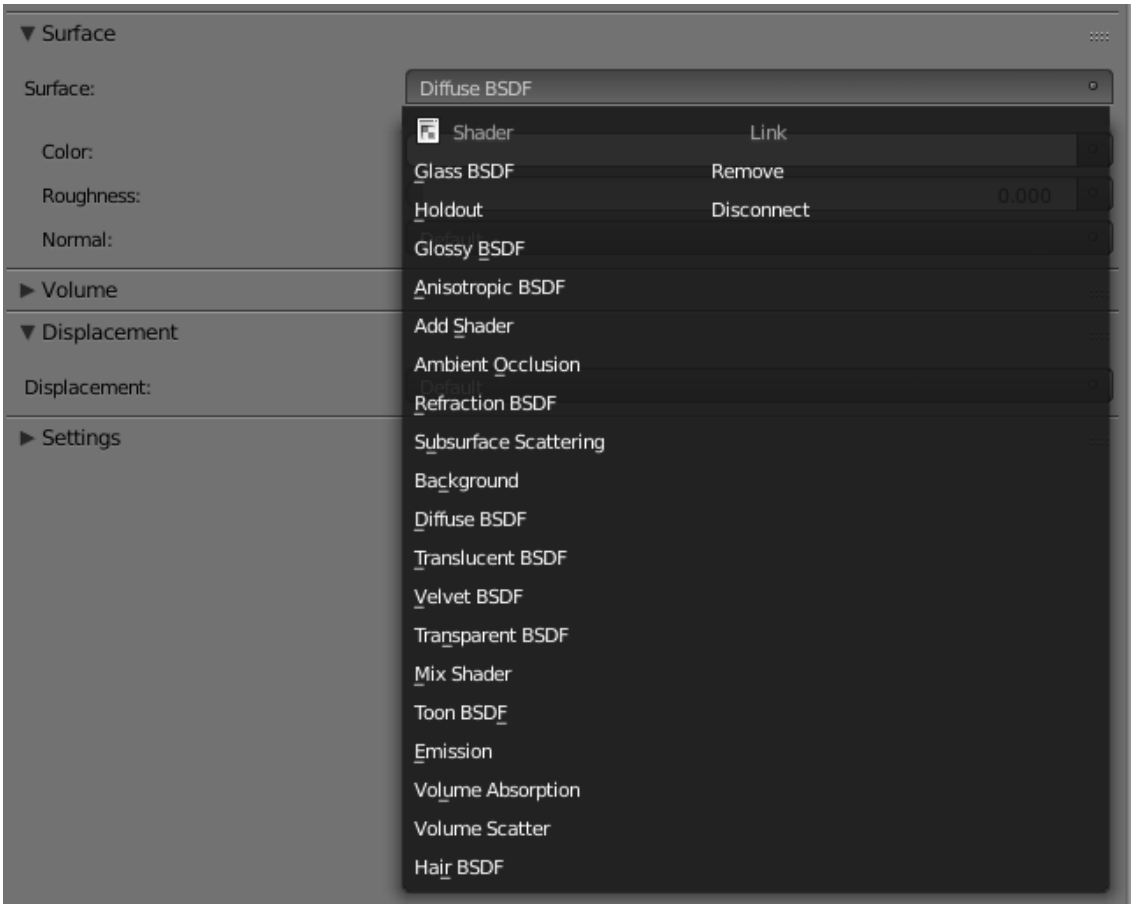

Figura 31 – blender materials

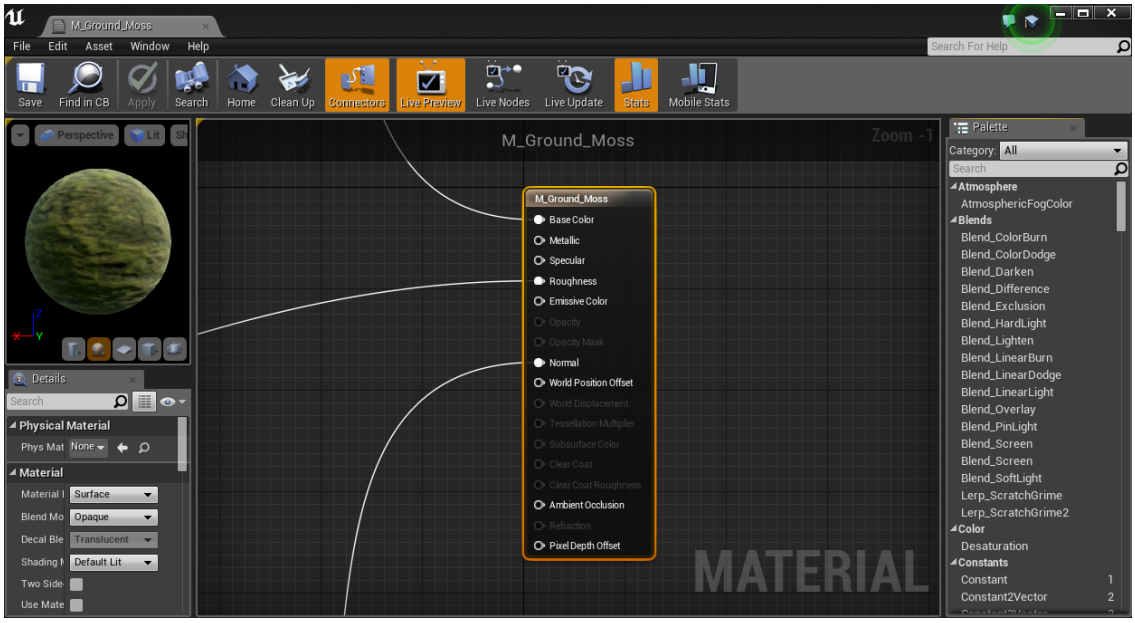

Figura 32 – Material do Unreal

Para criar os diferentes mapas das texturas foram usados dois programas, o Crazy Bump e o Photoshop. Ambos servem perfeitamente para o que se pretende, mas enquanto o Crazy Bump gera automaticamente os vários mapas, com base numa textura difusa, no Photoshop podemos fazer à nossa vontade e tem de ser tudo manual. Esta possibilidade de controlar manualmente detalhes para cada mapa de texturas é um ponto forte, mas em contra partida, no Photoshop não conseguimos gera um tipo de textura, normal map. Tendo este aspecto em conta o Crazy Bump é mais fácil e rápido para gerar as várias texturas e o resultado é muito bom. As texturas base foram obtidas através de vários sites, cgtextures, notextures, chocofur são alguns exemplos. Alguns destes sites já disponibilizam o conjunto das várias texturas por exemplo o chocofur, outros não por exemplo o cgtextures.

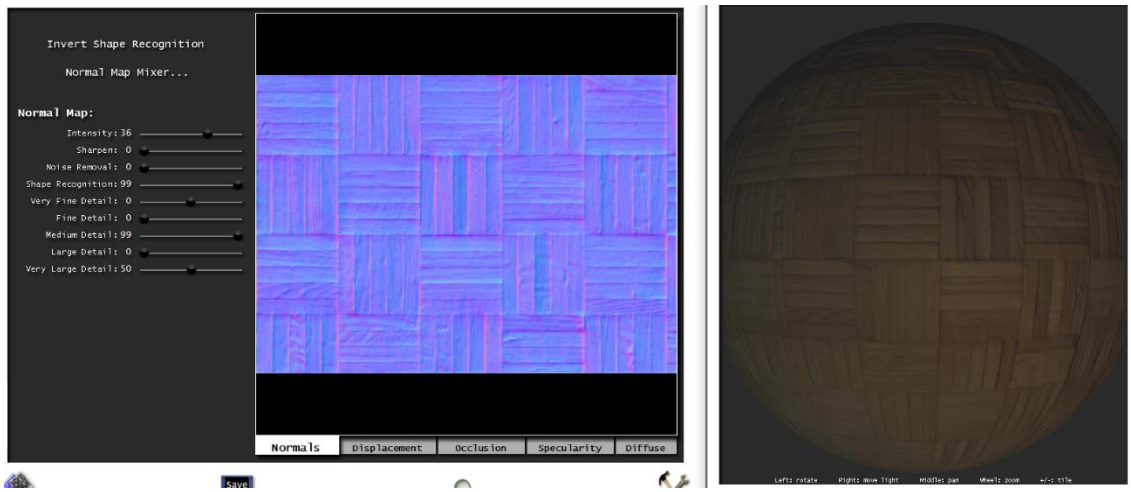

Figura 33 – crazy bump

Para além da textura base foram, como foi referido anteriormente, utilizadas mais texturas que são normal map, specular map, bump texture, displacement map. A textura difusa ou textura de cor é à partida a textura mais importante porque é a partir desta que vão ser geradas as outras. Uma boa resolução, a ausência de iluminação são alguns aspectos a ter em conta na escolha de uma textura difusa.

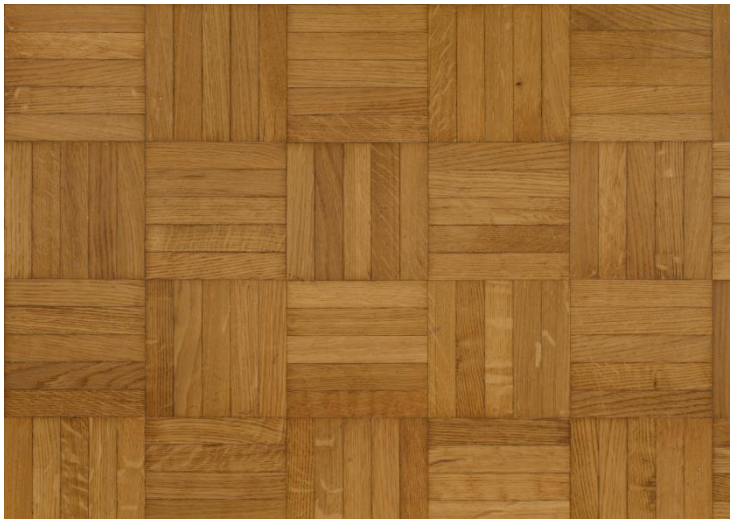

Figura 34 – difuse texture

As bump texture e displacement map são um pouco semelhantes uma vez que têm como objectivo adicionar imperfeições físicas aos shaders. Muitas vezes é preferível e mais fácil modelar estas próprias imperfeições ou até mesmo esculpir. Contudo mapas de texturas displacement correctamente criadas podem criar milagres e transmitir mesmo a ilusão para pequenos detalhes num objecto.

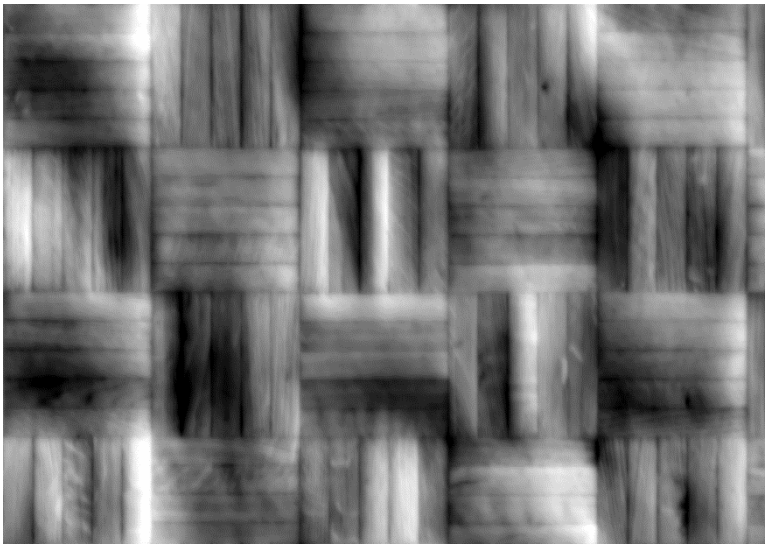

Figura 35 – displacement texture

Para além das texturas referidas acima temos ainda a normal map. Para além de ser necessário um programa específico, ou usar um plugin, o resultado é ligeiramente melhor em relação a usar uma bump map e muito diferente de usar uma displacement map. Outro aspecto a ter em conta é que normal map pode ter diferentes resultados em diferentes rendering engines.

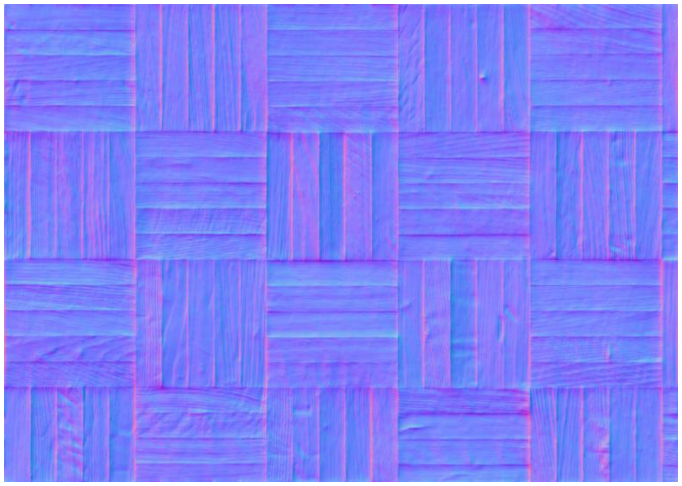

Figura 36 – normal map

Este tipo de textura serve para definir que áreas de uma superfície vão ter brilho, quanto maior o valor (de preto para branco), mais brilho terá a superfície.

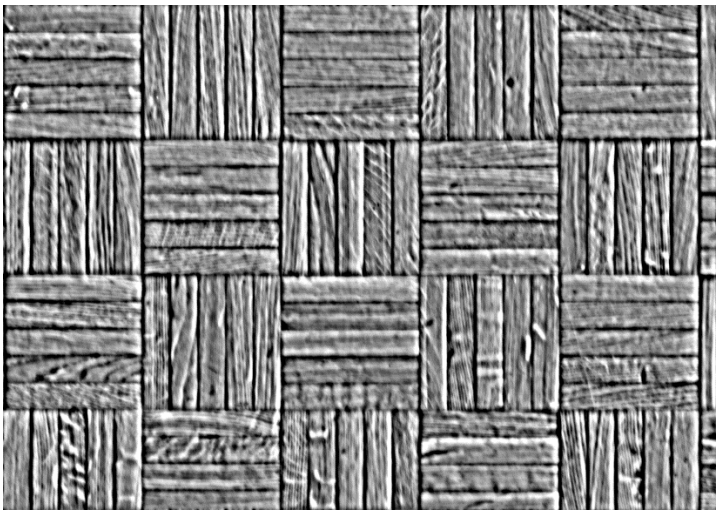

Figura 37 – specular texture

Existem várias maneiras de usar todas estas texturas no Blender, como está na imagem abaixo, ou seja, usando várias imagens e depois usando o respectivo node, para cada tipo de textura, de maneira a produzir um bom shader. O exemplo abaixo é para as paredes exteriores da casa.

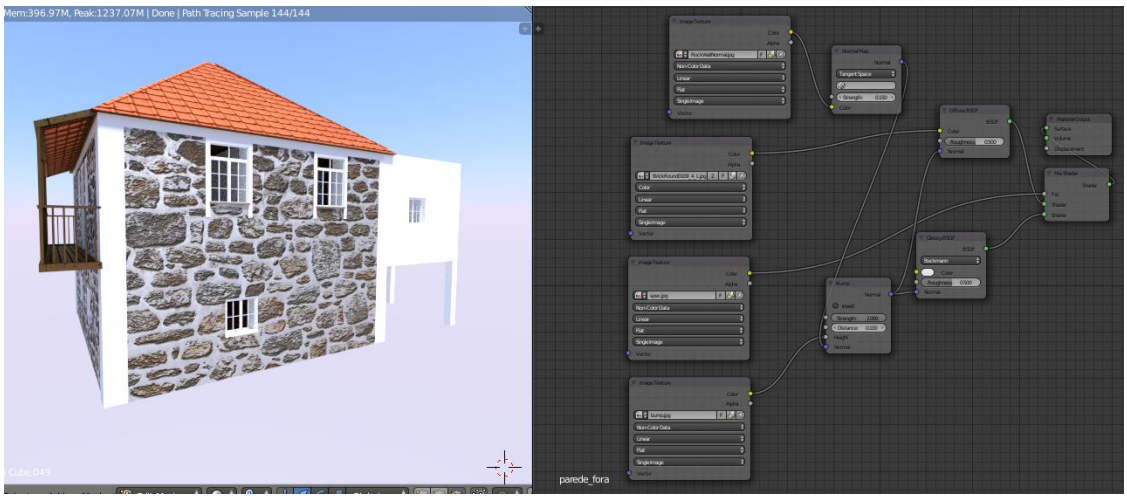

Figura 38 – rock shader blender com texturas

A outra maneira, mais simples e com apenas a textura difusa e que permite resultados, a meu ver melhores é como se verifica na imagem abaixo.

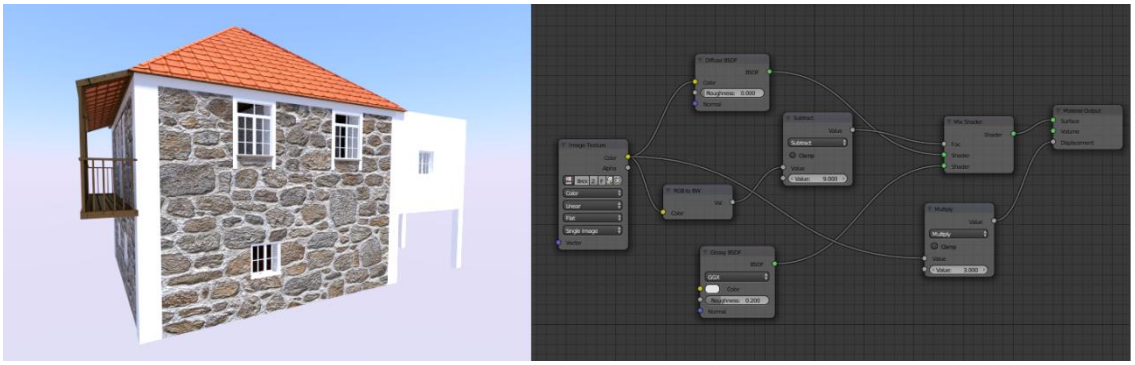

Figura 39 – rock shader com uma textura

As texturas são independentes de cada programa, a mesma textura base vai ser a mesma no Blender e no Unreal. Em relação aos materiais já não existe assim tanta liberdade de utilização, como foi referido no capítulo de transformação do Blender para o Unreal os shaders devem ser criados no próprio programa e não importados de outros.

Como tal, embora ambos os programas usem sistemas semelhantes para criar shaders a partilha entre ambos não funciona. No Blender o sistema é um pouco único, para além de conter materiais prontos a usar com a possibilidade de alterar algumas características, possibilita ainda criar do zero os nossos próprios materiais. Com esta possibilidade é permitido ao utilizador tentar simular physical based materials ou então criar algo que não exista na realidade. No Unreal também podemos criar do zero os nossos materiais, a inclusão de materiais pré-fabricados existe através da opcção *content pack* que é perguntado ao utilizador se quer incluir no seu projecto quando cria um. Embora ambos os programas permitam criar *physical based materials* este não é o único ponto a ter em conta na criação de *render* realista, a iluminação tem de ser trabalhada também de maneira a completar o render.

Ambos usam um sistema do tipo *node* onde o utilizador tem apenas de ligar ponto com ponto de maneira a criar os *shaders*, tem ainda a possibilidade de combinar esses mesmos *shaders.*

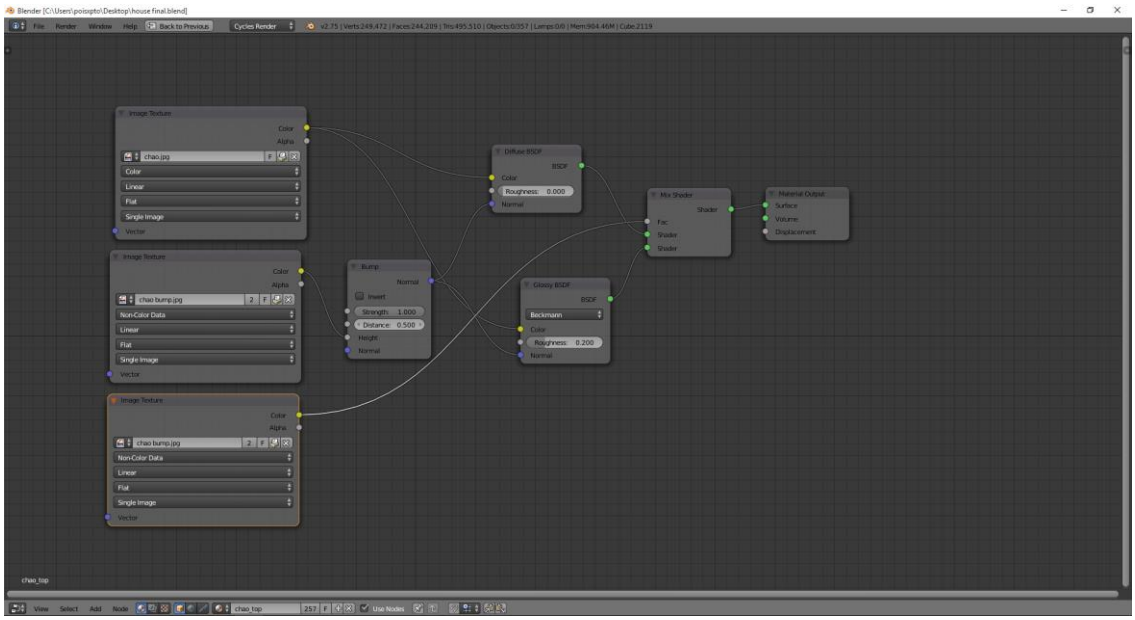

Figura 40 – blender sistema de nodes

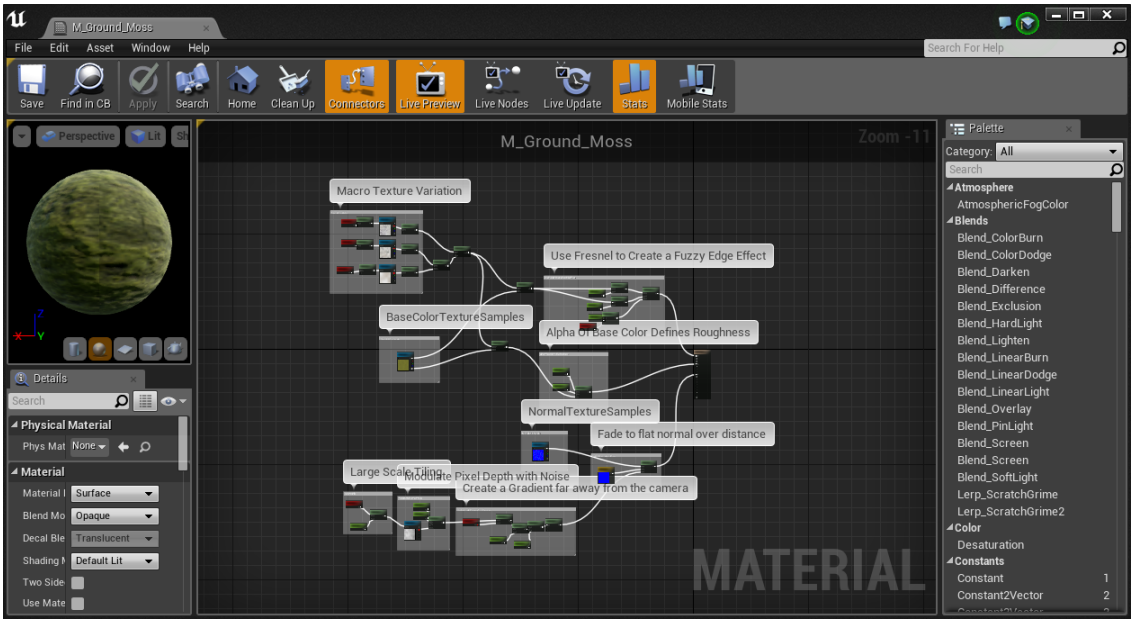

Figura 41 – unreal sistema de nodes

Quando se olha para uma superfície, o nosso cérebro interpreta de forma automática que características estão presentes nessa superfície. Detalhes como assimilar a cor a temperatura ou então associar o metal a algo brilhante ou o cimento a uma superfície rugosa. Existem um conjunto de factores a ter em conta quando se constrói um *shader*, tais como o factor difuso, o de reflexão, o de rugosidade e o de *displacement*. O primeiro diz respeito à cor real do objecto, pode ser uma textura ou mesmo uma cor, é muitas vezes relacionado com materiais que não reflectem uma vez que parecem neutros independentemente dos objectos em redor.

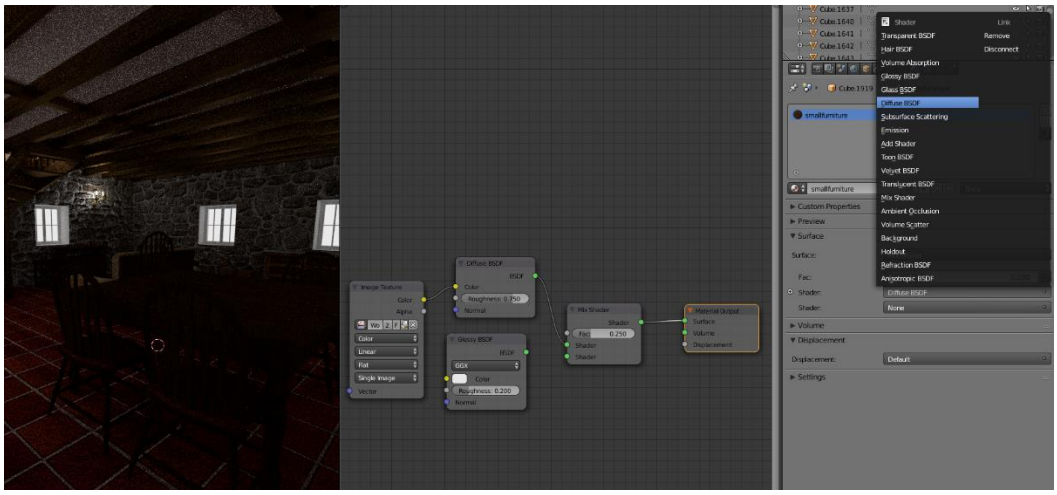

Figura 42 – material difuso blender

Contudo nenhuma superfície é 100% difusa uma vez reflectem sempre algo, por muito reduzido que seja esse valor. Para isso existe o factor de reflexão que é considerado dos mais importantes quando se quer construir algum shader de maneira a terem um aspecto realista. Este valor de reflectividade tem de ser ajustado conforme o shader que se está a criar, em conjunto estes dois materiais são essenciais para começar a criar um shader.

Figura 43 – material do sofá

O factor de rugosidade está relacionado com as outras todas, em conjunto com os outros materiais, este material serve por exemplo para tornar os reflexos nas superfícies mais desfocados ou então possibilitar que as superfícies difusas observam mais luminosidade. Tal como o factor de reflexão, o de rugosidade também é dependente do algo de visão. A lista de materiais é grande como se pode ver:

Diffuse BSDF – este shader recebe luz e uniformiza-a sem reflexões; Pode ser usado para superficies que não reflectam luz; Uma das propriedades deste material chama-se Roughness e permite adicionar algum detalhe ao material, mas detalhe este muito pequeno. Pode ser usado para detalhar tecidos, madeira, etc;

Glossy BSDF – este shader reflecte a luz e o ambiente em seu redor, é usado para adicionar reflexos a qualquer objecto; É usado bastante em combinação com o shader diffuse para criar materiais tais como plástico, metais, ceramicas, etc. Este shader contém também a propriedade Roughness que permite a precisão do reflexo. Com o valor zero este material é parecido com um espelho;

Anisotropic BSDF – este shader funciona da mesma maneira do que o glossy, mas direciona o reflexo em apenas uma direcção. É bastante usado para metais polidos. Este shader tem como propriedades Roughness que permite definir a orientação dos reflexos, valor negativo para orientação horizontal e valor positivo para orientação vertical. A

propriedade rotation serve rodar a direcção dos reflexos, por fim a propriedade tangent que permite escolher o eixo para o qual os reflexos serão contruídos;

Glass BSDF – este shader serve para simular o vidro, é usado para materiais como água, vidro ou joias. A propriedade Roughness permite em valores positivos dar um aspecto de vidro congelado, enquanto o valor de zero ou valores negativos dá o aspecto de vidro normal. Outra propriedade é o IOR(index de refração) que controla o quanto a luz passa por uma superfície;

Refraction BSDF – funciona da mesma maneira que o shader glass mas sem a componente de reflexão. Possui também as mesmas propriedades Roughness e IOR;

Transparent BSDF – shader transparente quando é de cor branca, com outra cor qualquer ligeiramente visivel. Tem como única propriedade a possibilidade de controlar a cor. Pode ser usado como mix de outros shaders por exemplo para usar num shader de vegetação;

Transucent BSDF – este shader permite que a luz atravesse o objecto ao qual o shader está aplicado, usado em materiais finos, materiais que não são vidros mas que permitem a passagem da luz, por exemplo folhas. Tem como propriedade a cor que pode ser à escolha do utilizador;

Velvet BSDF – shader usado para tecidos, deforma a luz em seu redor;

Toon BSDF – shader usado para dar um aspecto cartoon, para quando o resultado final não tem de ser realista. Tem três propriedades, o tipo de componente se é diffuse ou glossy, o diffuse é independete da posição da câmera enquanto que o glossy é dependente. A propriedade size varia dependendo da componente escolhida, a propriedade smooth permite suavizar ou desfocar as reflexões;

Subsurface Scattering – shader usado para simular os desvios da luz dentro de uma superficie, é usado como material para objectos de comida, pele. Tem como propriedades, scale que permite definir a profundidade a que a luz vai debaixo de uma superficie, radius determina a quantidade da superficie que é afectada por este shader, sharpness faz com que a deformação do objecto sejam mais precisas;

Emission – serve como shader para emitir luz a si próprio e os objectos em redor, usado para objecto que precisam de emitir luz, por exemplo, lampadas, fogo, etc. Tem como única propriedade a intensidade da luz desejada;

Background – serve para fornecer iluminação a uma cena, uniforme por todo;

Hair BSDF – usado cabelos e pêlo, absorve e reflecte a luz. Tem como propriedades as componentes de reflection e transmission. Reflection diz respeito à quantidade de luz que é reflectida, enquanto que a transmission tem haver com a quantidade de luz que atravessa os

cabelos ou pêlos. O uso das duas componentes alcança um resultado mais realista. O offset serve como valor para definir a direcção;

Ambient Occlusion – permite adicionar marcas negras a, deformações numa superficie ou intersecções entre objectos. Permite apenas controlar a cor desejada;

Holdout – permite adicionar um espaço em branco, transparente num render, serve para bloquear objectos numa cena;

Volume Absorption – controla a quantidade de luz que é absorvida por um objecto, alterando assim a superfície do objecto, usado por exemplo para dar cor a um vidro. A propriedade density permite controlar a quantidade absorvida;

Volume Scatter – espalha luz pelo volume do objecto e não pela superfície. Usado para simular nuvens. Tem como propriedades a density, que controla quantidade de luz que é espalha pelo volume do objecto e anisotropy que vai definir uma direcção para a density escolhida.

7.2.3 Iluminação/Render

Uma das novas adições do Blender é o seu novo motor de renderização Cycles. Trata-se de um render com funcionalidades actuais, focado na interactividade e facilidade de uso. Baseado na técnica de PathTracing é capaz de obter resultados bastante bons, capazes de competir com os grandes, MentalRay, RenderMan, entre outros. O Cycles possui um workflow bastante único em relação aos seus concorrentes e uma das grandes vantagens é o facto de permitir uma preview da cena em tempo real. Com isto o utilizador pode fazer ajustes e comprovar o resultado no mesmo instante, ao contrário de outros que só permitem verificar alguns pormenores em relação ao render final, como é o caso do V-Ray. Outra grande vantagem do Cycles é o facto de estar principalmente focada em fazer renderização através da placa gráfica, isto permite tempos de render mais rápidos, o único problema está no facto na limitação da memória da placa gráfica. Graças a este render actual, o Blender é capaz de produzir cenas mais realistas graças à iluminação que possui. No render inicial o Blender permitia apenas iluminação directa o que levava a tempos de render reduzidos, mas em contra partida produzia resultados pouco realista. Neste novo render continuam a estar disponíveis as mesmas fontes de iluminação, mas graças à iluminação indirecta agora têm outro impacto.

Software	Single License Costs	Upgrade or Sub-scription	V-Ray	Corona	Octane Renderer	Cycles
3ds Max 2015	3900€	1560€/yr	750€	448€	459€	0€
Maya 2015	3900€	1560€/yr	750€	no info	459€	0€
Cinema 4D	3200€	900€	790€	no info	459€	0€
SketchUp Pro	550€	180€/yr	580€	no info	379€	0€
Blender	0€	0€	250€	no info	416€	0€

The estimation above was based on the prices published on 25th March 2015.

Figura 44 – render Cycles custos

Começou em 2011 com o lançamento da tech demo Samaritan [Andrew Burnes, 2013], que mostrava o que poderiam vir a ser a próxima geração de gráficos. A correr em três placas gráficas topo de gama da NVIDIA, a demo introduziu técnicas como dynamic facial tesselation, point light reflections, BoK depth of field entre outras. Em 2012 a mesma versão foi apresentada [Andrew Burnes, 2013], mas desta vez mais optimizada e corria apenas com uma placa gráfica da geração seguinte. No mesmo ano foi mostrada a primeira tech demo do unreal engine 4, que se chamava Elemental Tech Demo e que mostrava o que poderia vir a ser o próximo "grito" no mundo dos gráficos, graças à iluminação avançada.

Usava uma técnica de render que ainda não é a que está presente na versão final do Unreal 4 de hoje. Essa técnica chamava-se SVOGI (Sparse Voxel Octree Global Illumination) que é baseada na técnica Voxel Cone Tracing [Cyril Crassin, 2012]. Esta, pode ser considerada uma versão "barata" de RayTracing, uma vez que ao contrário de "disparar" vários raios individuais para cada pixel, usa cones e voxels de maneira a poder abranger uma área maior. Ao contrário dos polígonos, cuja unidade básica é o triângulo, um voxel trata-se de um cubo 3D, são representados em formato tridimensional numa grelha.

No entanto esta técnica não veio a ser usada na versão final do Unreal, uma vez que era muito pesada a nível computacional, para tal a Epic criou alternativas, que à primeira vista são muito parecidas ao que anteriormente se esperava. Como o foco do Unreal 4 era a actual geração de consolas e computadores, e graças ao hardware que hoje existe esta técnica não foi então usada.

Em grande parte, o problema deste processamento intensivo que era requerido para fazer uso da técnica de SVOGI, estava no facto de usar uma sparse voxel octree que, consiste numa layer 3D de voxels, ou seja, uma voxel grid. O processamento de uma layer destas é bastante intensivo, devido à quantidade de voxels e ao seu volume que podem ter, como tal e para tornar este processo mais rápido, foram usadas texturas 3D, ou seja, em vez de se usar voxels passou-se a usar um array de texturas em 2D. Com isto surgiu outro problema, este conjunto de texturas, no seu todo, ocupavam muita memória. Para diminuir a memória necessária foi usado uma técnica que se chama, partial resident textures, que consiste em alocar uma textura enorme em pequenas imagens, cada uma com a respectiva entrada, index. As que são visíveis são usadas, as restantes não, isto faz com que seja feita uma gestão da memória mais eficiente.

Para já SVOGI está posta de parte, mas com os avanços a nível de hardware que surgem, que proporcionam mais poder de processamento paralelo, poderemos vir a ter no futuro esta técnica, como padrão e com isso reduzir a barreira ainda mais para o que é apenas visível no grande ecrã.

Em 2013, a segunda tech demo que foi mostrada, cuja nome era Infiltrator Tech Demo [Andrew Burnes, 2013] já mostrou uma técnica de render diferente. Ao contrário da tão desejada iluminação global, usava uma mistura de iluminação estática e dinâmica, milhares de fontes de iluminação com algumas a emitir sombras. A versão mais recente usa, do Unreal, uma técnica que se chama Light Volumes Propagation [Anton Kaplanyan], [Epic], semelhante à que usa o CryEngine da Crytek.

O Unreal possui um motor gráfico actual, que ainda está em constante desenvolvimento, possuidor dos mais variados efeitos, capaz de correr em qualquer plataforma e tem a vantagem, que foi recentemente introduzida, o facto de ser gratuita. A nível de gráficos possui um conjunto de efeitos, post processing tais como anti-aliasing, eye adapation, bloom, color grading, depth of field, lens flare, scene fringe, screen space reflection e por fim vignette.

O anti aliasing usado é o FXAA [Timothy Lottes, 2009], trata-se de uma tecnologia da NVIDIA, que por não requerer uma quantidade grande de processamento o resultado não é o melhor. Ao contrário do anti aliasing normal, esta técnica "alisa" as arestas conforme elas aparecem no ecrã, enquanto pixéis, enquanto o anti aliasing normal tem em conta o modelo 3D. O post processing eye adaptation, serve para recriar o efeito que o ser humano sofre quando passa de espaços escuros para claros, sofre então uma pequena adaptação. O efeito bloom serve para transmitir aquele blur transmitido por objectos que emitem luz, trata-se de um efeito real e que transmite por isso bastante realismo. Color granding é o efeito que permite fazer correcções de cor, alterações de tom de cor num cenário de maneira o ambiente seja mais realista. Depth of field aplica um blur ao cenário com base na distância ao ponto de foco. Este efeito pode ser aplicado antes ou depois do objecto que está a ser focado. O Unreal possui dois tipos de depth of field, bokeh DoF e o gaussian DoF. O primeiro requer mais processamento e o mais parecido com o que usado em filmes e fotos, enquanto o gaussian DoF é bastante usado em jogos, logo não requer grande processamento. Lens flare é o efeito que a luz provoca quando esta atravessa as lentes da câmera devido a imperfeições nas lentes. Este efeito acontece quando se está a ver objectos bastante brilhantes. Um efeito que tem vindo a ser utilizado em vários jogos, visto que parece distorcer toda a imagem na qual é aplicada é o efeito scene fringe. Trata-se de simular de simular as mudanças de cor provocada pelas lentes das câmeras. Screen Space Reflection é principalmente um truque para proporcionar reflexos dinâmicos apenas. Trata-se apenas de fornecer a gráficos que usam a técnica de rasterização a oportunidade de terem reflexos em tempo real. Sombras, transparências, entre outros, não fazem parte deste truque. Uma das desvantagens é o facto de apenas reflectir objectos que estão visíveis no ecrã, ou seja, tudo que não é visível não é reflectido. Por ultimo o efeito vignette que simula o efeito que as lentes das câmeras provocam, ou seja, escurecer os cantos da imagem. Serve para corrigir possíveis falhas da câmera. A mais recente versão do Unreal, a 4, trouxe algumas alterações, que permitem que seja um motor bastante actual e que em alguma maneira esteja preparado para as mudanças que vão surgindo ao longo do tempo. O Unreal ainda não tem todas as funções a 100%, como tal ainda tem alguns bugs, uma alteração nova e que ainda não está completa é o sistema de iluminação global usado.

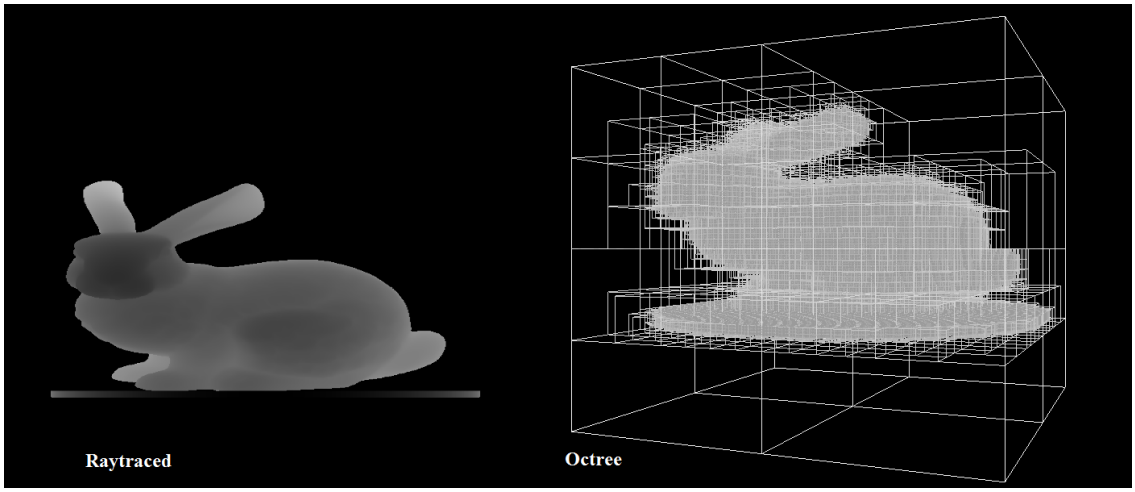

Figura 45 – svogi

A iluminação é um dos componentes mais importantes de qualquer rendering. A modelação e as texturas por mais precisas que sejam, não têm um resultado realista sem que a cena onde se encontram esteja devidamente iluminada. No Blender existe um conjunto de elementos, dos quais temos controlo, e que podem afectar a iluminação. A cor que é dada ao mundo, ou seja, a cor que damos de fundo à cena que pode ser uma cor ou até mesmo uma textura HDR.

Outros elementos que afectam a iluminação são o uso de Ambiente Occlusion como maneira de determinar o quanto exposto está um objecto à iluminação ambiente, o número de luzes na cena e o tipo de render usado.

No Blender uma fonte de iluminação pode ser do tipo, sun, spot, lamp, hemi ou area. Pode ter qualquer cor à escolha, pode emitir em qualquer direcção de qualquer posição e deve ter uma energia atribuída. Esta energia é que vai definir a intensidade da luz projectada. No Blender para mudarmos o tipo de iluminação, é preciso primeiro alterar o render para Cycles. O integrator é o algoritmo de render usado para processar a iluminação, actualmente o render Cycles suporta o integrator PathTracing, que na grande parte das situações funciona bem tirando algumas situações onde existe pouca iluminação ou caustics. Com o integrator PathTracing temos presentes dois modos: PathTracing e Branched PathTracing. O PathTracing usado no Cycles trata-se de um PathTracer puro portanto requer uma elevada quantidade de samples por pixel de maneira a obter um resultado mais preciso e real. É possível ainda alterar um conjunto de campos de maneira a obter melhores resultados, tais como o número de samples diffuse ou glossy ou AO de maneira a tirar melhor partido de cada superfície. No grupo Bounces podemos alterar o número de samples para alterar a qualidade final do render. O max bounces serve para definir quantas vezes a luz vai saltar de superfície em superfície, quanto maior for o numero melhor a qualidade de render mas em contrario o tempo de render aumenta substancialmente.

O valor de zero faz com que seja usado apenas iluminação directa visto que a luz ao interagir com uma superfície não salta mais. O min bounces serve para determinar quando o integrator usa Russian Roulette de maneira a terminar com paths que não contribuem muito para o render final.

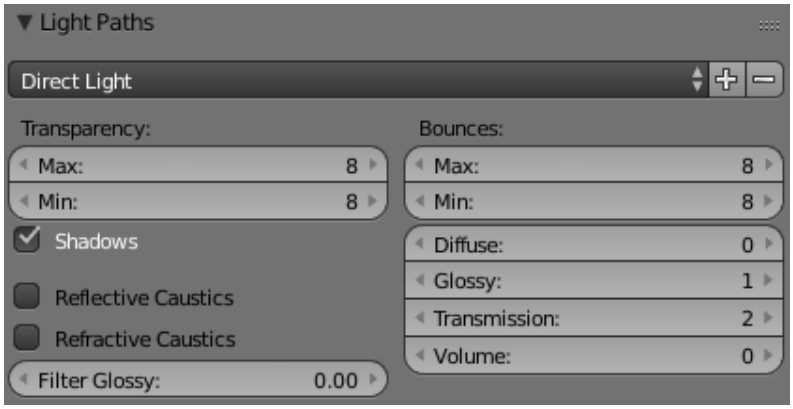

Figura 46 – características da iluminação directa no Blender

Figura 47 – exemplo de render usando apenas iluminação directa no Blender

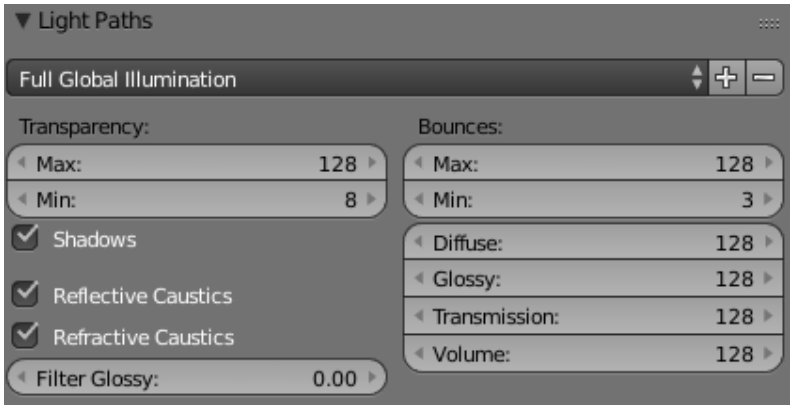

Figura 48 – características da iluminação global no Blender

Figura 49 – exemplo de render com iluminação global no Blender

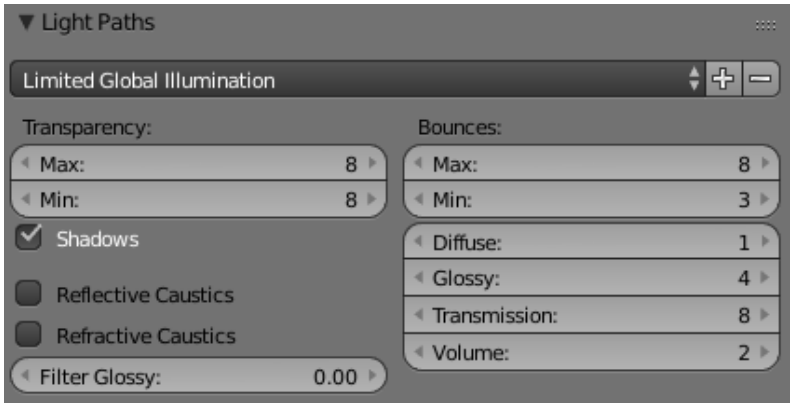

Figura 50 – características da iluminação global limitada no Blender

Figura 51 – exemplo de iluminação global limitada no Blender

Embora o Unreal tenha um conjunto avançado de post process, a iluminação sofre um bocado tendo em conta que é para correr em situações diferentes da do Blender. A iluminação é bastante realista, embora não seja a prometida SVOGI, o motor possui um sistema de iluminação melhorado que permite resultados semelhantes ao bastante conhecido, V-Ray, usando a fonte de iluminação skylight para simular a luz que vem do céu e as directional lights para simular a luz que vem do sol. No Unreal existem directional lights, point lights, spot lights e sky lights. Estas fontes de iluminação podem ter um tipo de mobilidade, static, stationary ou moveable. A primeira é usada para obter resultados rápidos e permite fazer bake à iluminação. Stationary light é em algo semelhante à static mas permite em tempo real mudar a cor ou a intensidade, possibilita ainda sombras e light bounces de geometria estática. Moveable light, significa que a luz é totalmente dinâmica bem como as suas sombras, é a mais lenta das três mas permite uma maior flexibilidade.

No Unreal existem dois algoritmos de iluminação possíveis, o LightMass e o Light Volume Propagation. Ambas funcionam de maneira diferente, mas a principal diferença está no facto de uma ser dinâmica e a outra não. Light Volume Propagation é dinâmica mas ainda se encontra em fase de desenvolvimento, LightMass é estática e está implementada. LightMass consiste em criar mapas de iluminação com interacções complexas de luz, tais como sombras ou reflexos difusos, por exemplo. O LightMass é usado para processar pequenas quantidades de luz provenientes das fontes de iluminação estáticas ou stationary. Todo este processo de entre o editor do Unreal e o LightMass é comunicado através de um SwarmAgent que controla a construção da iluminação local ou então distribui esse processo por máquinas remotas. O SwarmAgent é uma aplicação em C#, que arranca automaticamente com o Unreal e para além da tarefa de processar a iluminação possibilita ainda verificar o processo de construção todo, bem como que máquinas estão a ser usadas, quantas threads, entre outros valores.

Para se obter uma qualidade de render superior, usando LightMass, é preciso alterar um conjunto de opcções. As que vêm por defeito servem bem para jogos, mas para se obter uma visualização superior é preciso alterar estas quatro opcções:

Static Lighting Level Scale que diz respeito ao número de samples que são emitidos. Quanto menor for este valor mais samples são emitidas, em contrapartida se esse valor for demasiado pequeno, pode provocar artefactos na iluminação. O valor recomendado é 1. Number Indirect Lighting Bounces que controla o número de bounces da luz, quanto maior for este valor mais samples são iluminados, melhor qualidade é obtida no render. Para este projecto foi definido o valor de 100, muito similar ao usado no Blender com a iluminação global activada. Indirect Lighting Quality é o slider mais importante pois controla a qualidade da iluminação global. Este slider está apenas limitado ao valor de 4, mas pode ser superior, para alterar basta carregar nele e digitar o número desejado. Para este projecto foi escolhido o valor de 10. Como se trata do valor mais importante para a qualidade final do render, o impacto no tempo necessário para o LightMass processar a iluminação é muito grande. Por último o campo Indirect Lighting Smoothness controla o quanto suave são as sombras, valores pequenos permitem sombras certas, detalhadas mas podem produzir artefactos.

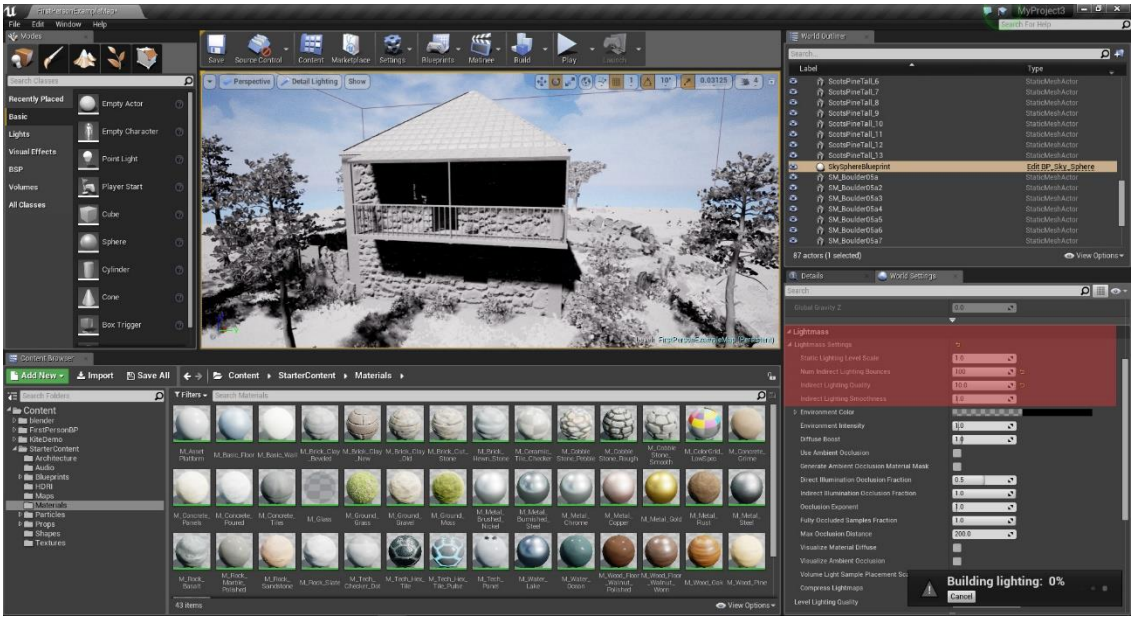

Figura 52 – lightmass settings

Mesmo que o valor introduzido, neste último campo, não produza resultados desejados é possível aumentar o tamanho do lightmap de cada objecto, individualmente ou no geral. É preciso ter em conta que também este valor tem um impacto enorme no tempo de baking, um valor entre 128 e 256 é bom para fazer preview, mas para render final tem de ser maior, 512 por exemplo. É preciso também ter em atenção ao limite do buffer, quanto maior o tamanho do lightmap mais tamanho é ocupado, caso este valor atinga o limite o resultado não é nada bom. Para contornar este problema é possível desligar o streaming dos lightmaps, streaming que é criado pelo motor de maneira a contornar o tamanho reduzido do buffer, mas para isso é preciso um computador muito potente para correr o projecto.

Outra maneira para de obter um render com mais qualidade é alterar o valor dos reflexos. No Unreal existem duas maneiras possíveis, Reflection Capture Spheres ou PostProcess Screen Space Reflection. A segunda é a melhor escolha visto que a primeira graças ao DirectX 11 está limitado no tamanho.

O passo final é aumentar o valor do AA para um valor superior, para isso executa-se o seguinte comando na consola: r.PostProcessAAQuality 6; pode-se ainda aumentar a resolução final do render para um tamanho superior ao nativo.

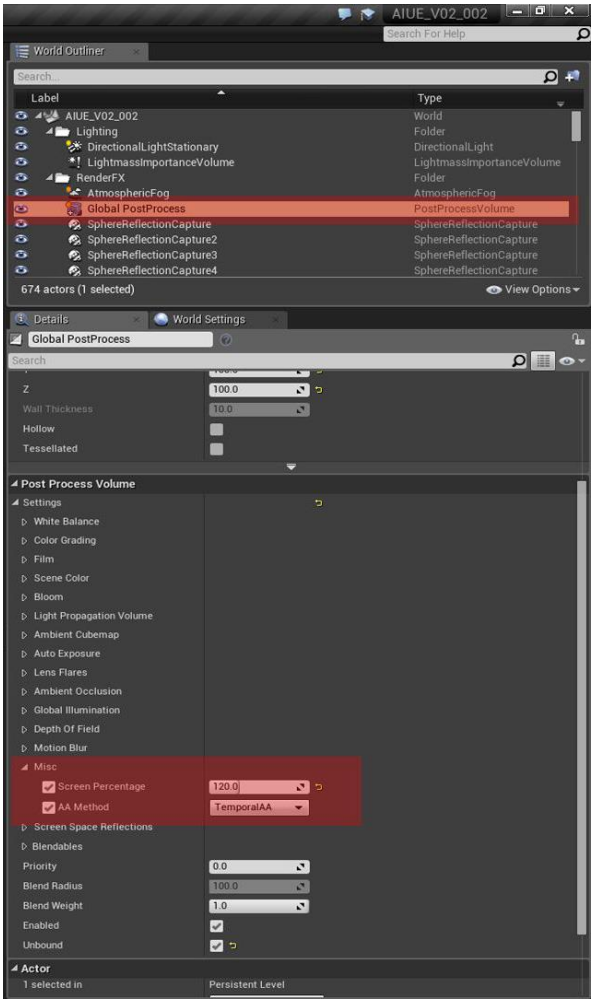

Figura 53 – aa e resolution no Unreal

Mesmo com estas alterações é preciso ter alguns cuidados, como foi referido no capítulo de transformação do cenário, caso contrário o resultado pode ser desagradável.

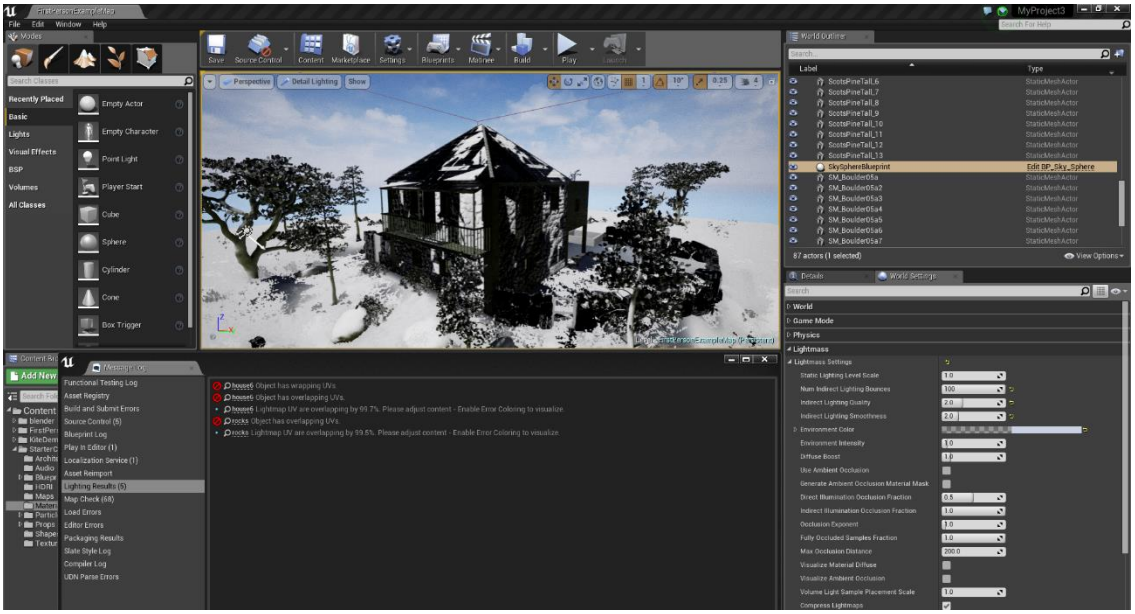

Figura 54 – erro no lightmass

LightMass é a iluminação activada por defeito no Unreal, para mudar para Light Volume Propagation é preciso acrescentar o campo r.LightPropagationVolume = 1 ao ficheiro de configuração ConsoleVariables.ini. Todo este processo realiza-se apenas uma vez e é necessário porque este tipo de iluminação ainda está em desenvolvimento.

No entanto houve alguns problemas, por exemplo a imagem acima mostra exemplos de erros que para serem corrigidos, requerem voltar ao Blender. São problema simples de resolver mas são demorosos. Estão relacionados com os mapas UV, ou seja, no objecto todo existem mapas UV que estão mal, isso significa que existem mapas UV que estão sobrepostos ou não existem. Com estes problemas o render do LightMass dá erro.

Para que o cenário suporte Light Volume Propagation é preciso ainda fazer mais algumas alterações às luzes e às configurações no separador world. Como se pode ver na figura seguinte é preciso alterar a mobility da fonte de iluminação para movable, activar o campo dynamic indirect lighting. Foi ainda desactivado o cast static shadows e activado o cast dynamic shadows. A intensidade da iluminação indirecta pode ser alterada no respectivo campo, como se vê abaixo onde. O valor 1 é o mínimo, para cima deste valor o efeito começa a ser muito mau.

Figura 55 – características da fonte de iluminação usando LPV no Unreal

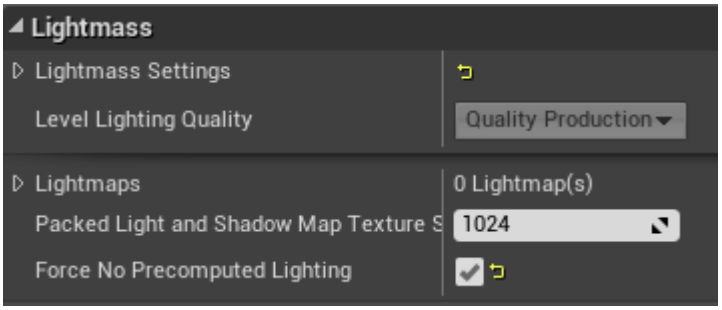

Figura 56 – desactivar LightMass no Unreal

É preciso ainda no separador world settings activar a opcção force no precomputed lighting para que toda a iluminação seja construída em tempo real caso contrário sempre que se fizesse uma alteração na iluminação era necessário construir a iluminação. Depois de feitas estas alterações é possível fazer algumas modificações à iluminação como se pode ver na figura abaixo.

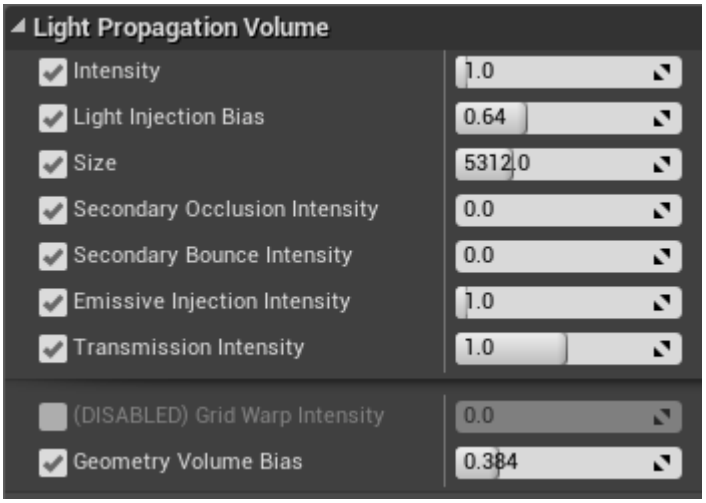

Figura 57 – características do algoritmo de iluminação LPV no Unreal

Depois de todas as alterações realizadas, o resultado é o seguinte, como se vê na imagem abaixo.

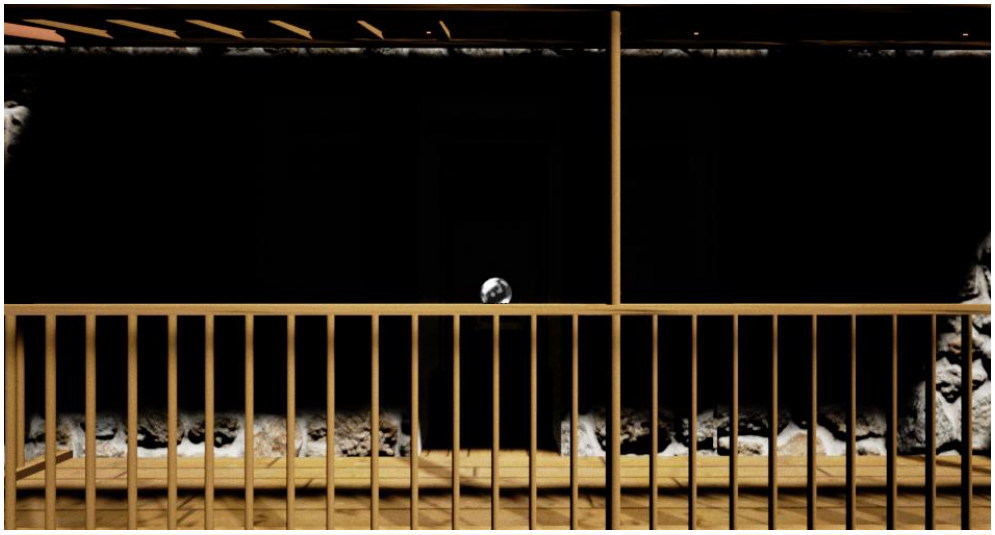

Figura 58 – iluminação directa Unreal

Figura 59 – iluminação LPV no Unreal

Figura 60 – render final Blender

Figura 61 – render final Unreal

8 Conclusão

No geral, esta tese mostra que tanto o Blender como o Unreal são soluções viáveis para a construção de cenários e ambientes 3D. Partindo do Blender para o Unreal, tudo foi modelado de raiz, as texturas foram ajustadas de maneira a servirem e a iluminação foi um conjunto de fontes que no final obtiveram um bom resultado. Um aspecto importante de realçar é o facto de cada vez mais é possível correr ambientes 3D, em tempo real com iluminação global, graças as melhorias de hardware. A qualidade obtida apenas em filmes pode vir a ser uma realidade daqui a uns anos nos nossos computadores.

A nível de programas a escolha, como é provada nesta tese, é muita e variada, cada um com os respectivos preços e características mas todos possibilitam ao utilizador criar ambientes realistas. Estes dois programas mostram também que a nível de programas free e open source a qualidade tem sido cada vez maior.

Todo o trabalho foi conseguido utilizando a secção de ajuda dos programas que foram utilizados, entre outros sites, sendo todos os outros problemas resolvidos através de várias tentativas.

Como trabalhos futuros, com base neste, poderiam ser criados ambientes mais envolventes, ou numa área mais avançada talvez até melhorara as técnicas de render dos dois programas.

Concluído a tese, cumpridos os objectivos propostos fica a experiência para o futuro destas ferramentas como alternativas credíveis para utilizar num futuro próximo na arte de domínio do 3D.

Referências

[Anton Kaplanyan, 2009]	Advances in Real-Time Rendering in 3D Graphics and Games Course – SIGGRAPH 2009.
[Anton Kaplanyan]	Cascaded Light Propagation Volumes for Real-Time Indirect Illumination.
[Epic]	Light Propagation Volumes GI, https://wiki.unrealengine.com/Light_Propagation_Volumes_GI
[Timothy Lottes, 2009]	FXAA, http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf.
[Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, Elmar Eisemann 2011]	Interactive Indirect Illumination Using Voxel Cone Tracing, https://research.nvidia.com/sites/default/files/publications/GIVoxels-pg2011-authors.pdf
[Ian Lilley, 2012]	Introduction to GigaVoxels and Sparse Textures, http://realtimevoxels.blogspot.pt/2012/09/introduction-to-gigavoxels-and-sparse.html
[Andrew Burnes. 2011]	Epic Talks Samaritan & The Future Of Graphics, http://www.geforce.com/whats-new/articles/epic-tim-sweeney-and-nvidia-talk-samaritan-and-the-future-of-graphics
[Andrew Burnes, 2013]	Infiltrator: GTX 680-Powered Unreal Engine 4 Tech Demo Unveiled, http://www.geforce.com/whats-new/articles/infiltrator-gtx-680-powered-unreal-engine-4-tech-demo-unveiled
[Cyril Crassin, 2012]	Voxel Cone Tracing and Sparse Voxel Octree for Real-time Global Illumination, http://on-demand.gputechconf.com/gtc/2012/presentations/SB134-Voxel-Cone-Tracing-Octree-Real-Time-Illumination.pdf
[Ivo Boyadzhiev and Kevin Matzen, 1997]	INSTANT RADIOSITY, http://www.cs.cornell.edu/Courses/cs6630/2012sp/slides/Boyadzhiev-Matzen-InstantRadiosity.pdf
[M.C. Escher, 1898–1972]	Irradiance Caching and Derived Methods, http://www.cs.dartmouth.edu/~wjarosz/publications/dissertation/chapter3.pdf

[Ben Woodhouse, 2012]	Dynamic Global Illumination in Fable Legends, http://www.lionhead.com/blog/2014/april/17/dynamic-global-illumination-in-fable-legends/
[H. Dammertz, A. Keller, d H. P. A. Lensch, 2010]	Progressive Point-Light-Based Global Illumination, http://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.100/institut/mitarbeiter/holger/paper_ProgressivePointLightGI.pdf
[Jacco Bikker, Jeroen van Schijndel, 2012]	The Brigade Renderer: A Path Tracer for Real-Time Games, http://lib.westfield.ma.edu/webapa.htm [ultimo acesso: Dez 2011]
[Cyril Crassin, 2012]	Octree-Based Sparse Voxelization for Real-Time Global Illumination, http://www.icare3d.org/research/GTC2012_Voxelization_public.pdf
[Fredo Durand and Barb Cutler]	Global Illumination with Radiosity, https://www.cs.drexel.edu/~david/Classes/CS431/Lectures/Radiosity.pdf
[Zeff O. 2010]	Radiosity - Ray Tracing, https://www.cg.tuwien.ac.at/research/rendering/rays-radio/
[Vilém Otte, 2013]	Path tracing on GPU, http://is.muni.cz/th/396530/fi_b/Bachelor.pdf
[Mark Wilson, 2014]	Ikea's Catalog Is Computer Generated Imagery, http://www.fastcodesign.com/3034975/75-of-ikeas-catalog-is-computer-generated-imagery
[Jim Thacker, 2014]	See an R&D preview of V-Ray RT for MotionBuilder, http://www.cgchannel.com/2014/04/see-an-rd-preview-of-v-ray-rt-for-motionbuilder/
[Jim Thacker, 2014]	Watch real-time path tracing in Otoy's Brigade engine, http://www.cgchannel.com/2013/04/eye-candy-otoys-brigade-real-time-path-tracing-engine/
[Pixar Animation Studios]	Ray Tracing for the Movie 'Cars' http://www.cs.cmu.edu/afs/cs/academic/class/15869-f11/www/readings/christensen07_cars.pdf
[Walt Disney Animation Studios, 2013]	Sorted Deferred Shading for Production Path Tracing https://disney-animation.s3.amazonaws.com/uploads/production/publication_asset/70/asset/Sorted_Deferred_Shading_For_Production_Path_Tracing.pdf
[TXAA NVIDIA, 2015]	http://www.geforce.com/hardware/technology/txaa/technology

[SMAA, Crytek 2012]	SMAA: Enhanced Subpixel Morphological Antialiasing, http://www.iryoku.com/smaa/
[CSAA, Peter Young, NVIDIA]	CSAA (Coverage Sampling Antialiasing), http://www.nvidia.com/object/coverage-sampled-aa.html
[Modeling Global Illumination With radiosity, 3DS MAX 2015]	http://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/3DSMax-Archive/files/GUID-C5A3C77B-794B-4444-9783-7F2EA11C16BD-htm.html
[Disney render, 2015]	http://www.disneyanimation.com/technology/innovations/hyperion
[Disney Hyperion, 2015]	http://www.engadget.com/2014/10/18/disney-big-hero-6/
[Disney CODA, 2015]	http://www.fxguide.com/featured/disneys-new-production-renderer-hyperion-yes-disney/
[Rasterization overview, 2015]	http://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation
[Mental Ray, NVIDIA]	http://www.nvidia-arc.com/products/mentalray/mental-ray-313.html
[RenderMan overview, 2015]	http://www.fxguide.com/featured/renderman-under-the-new-varnish/
[OTOY render, 2015]	https://home.otoy.com/otoy-unveils-octanerender-3-worlds-best-gpu-renderer/
[Monte Carlo methods, 2015]	http://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-in-practice
[PathTracing methods, 2015]	http://www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing
[Bi-directional PathTracing, 1993]	http://www.cs.princeton.edu/courses/archive/fall03/cs526/papers/lafortune93.pdf
[Rendering equation, 1986]	http://graphics.cs.williams.edu/courses/cs371/f08/reading/kajiya86.pdf
[Fundamentas of Texture Mapping and Image Warping, Paul S.Heckbert, 1989]	http://digitalassets.lib.berkeley.edu/techreports/ucb/text/CSD-89-516.pdf
[RayTracing for the movie “Cars”, Pixar]	https://www.cs.cmu.edu/afs/cs/academic/class/15869-f11/www/readings/christensen07_cars.pdf
[Unreal rendering]	[https://docs.unrealengine.com/latest/INT/Programming/Rendering/ThreadedRendering/index.html],

[Stéphane Marchesin and Catherine Mongenet and Jean-Michel Dischler] Dynamic Load Balancing for Parallel Volume Rendering, https://dpt-info.u-strasbg.fr/~dischler/papers/egpgv06.pdf

[Construct, NVIDIA QUADRO 2015] How a Render Farm Squeezed Into an Apartment Brought to Life a World Ruled by Robots - See more at: http://blogs.nvidia.com/blog/2015/03/17/construct/#sthash.gQ8tpOXw.dpuf, http://blogs.nvidia.com/blog/2015/03/17/construct/