Portais Renderizáveis em 3D

Omar Mahmoud Abou Ajoue

Orientador:
Marcel Jackowski

1. Introdução

Motivação

Ao longo de nossa graduação, nós compartilhamos um interesse comum pelo desenvolvimento de jogos. Sempre fomos fãs de jogos eletrônicos desde crianças e sempre tivemos a curiosidade sobre como estes jogos funcionavam. Esta curiosidade foi um dos motivos que nos fez optar pelo curso de ciências da computação.

Dentre os jogos que gostamos, um dos quais mais nos fascinou foi um jogo chamado Portal, criado pela produtora Valve, lançado para PC e Xbox 360 em 2007. A jogabilidade presente neste título é única, e os desafios a serem superados em cada fase nos mantinham presos ao jogo por horas. Portal foi aclamado como um dos jogos mais originais de 2007.

Portal é um jogo de puzzle para um único jogador em primeira pessoa. Isto significa que o jogador deve resolver quebra cabeças para passar das fases, enquanto movimenta seu personagem com uma visão em primeira pessoa.

O jogador possui uma arma que pode utilizar para ajudá-lo a resolver os quebra cabeças. A única função desta arma é a de criar portais nas paredes do mundo 3D, que são utilizados pelo jogador para acessar diversas localidades, grande parte delas inacessíveis pela movimentação convencional do personagem (como grandes alturas, por exemplo, ou obstáculos).

A criação dos portais é dada por comandos do mouse, com cliques dos botões esquerdo e direito. Cada botão cria um portal, que no jogo original, possuem borda colorida em azul e laranja para diferenciá-los. Ao acionar novamente os botões, o portal da cor correspondente é trocado pela nova localização. Os portais criados desta forma criam um sistema de entrada e saída. Ao atravessar um dos portais, o jogador é levado à posição do outro portal. Esta transição é feita de modo suave, e o jogador realmente sente como se estivesse atravessando uma janela até o outro lado. Pode-se inclusive ficar parado no meio, entre os dois portais.

Figura 1: Visão de portais no jogo Portal

Além do tele transporte, os portais possuem um efeito visual muito interessante. Ao olhar através do portal, o jogador consegue observar com perfeição o mundo do outro lado, como se estivesse de fato vendo a saída do portal correspondente do par azul-laranja. Se posicionados corretamente, os portais permitem que o jogador se veja através deles. Outro efeito interessante ocorre quando os dois portais são colocados um em frente ao outro. O efeito resultante é o mesmo que ocorre quando ficamos entre dois espelhos, quando há impressão de uma profundidade infinita.

Figura 2: Visão através do portal azul da Fig. 1

Figura 3: Visão através do portal laranja da Fig. 1

Todas estas características deste jogo despertaram grande curiosidade. Inicialmente, era desejado entender como seria feita a ligação entre os portais. Chegamos a imaginar soluções completamente impraticáveis, como clonar o ambiente 3D e colocá-lo atrás de cada portal.

Decidimos explorar esta curiosidade e desenvolver a nossa própria solução para o problema. O trabalho de conclusão de curso é a versão de uma implementação deste tipo de portais. Não é do conhecimento dos integrantes se as soluções encontradas são as mesmas dos desenvolvedores do jogo original, mas acreditamos que conseguimos construir uma ferramenta de criação de portais satisfatória, que pode ser utilizada pra criar um jogo com cenários tão divertidos quanto alguns dos cenários do Portal.

Objetivos

Nosso objetivo é implementar um jogo 3D em primeira pessoa, que permita ao jogador criar portais com propriedades similares às do jogo Portal.

Para a concretização deste objetivo, há diversas tarefas e empecilhos a cumprir. Iniciaremos expondo os problemas que encontramos e daremos a seguir as soluções.

Problemas

A criação do nosso jogo envolve vários problemas, alguns deles não relacionados diretamente à criação dos portais. Estes problemas são:

- Criar um mundo 3D a ser explorado pelo jogador;
- Criar uma câmera que dê ao jogador a ilusão de estar dentro de um personagem 3D, animado e que se movimenta de acordo;
- Prover ao jogador meios de controlar esta câmera satisfatoriamente e atualizar a posição e a rotação do personagem de acordo;
- Criar mecanismos de colisão para este personagem com os objetos do mundo 3D e com os portais;
- Criar uma física básica, que aplique gravidade ao personagem;
- Permitir ao jogador escolher pontos nas paredes do mundo 3D onde ele deseja criar os portais;
- Criar os portais nos pontos escolhidos pelo jogador, orientados corretamente, independentemente das paredes escolhidas;
- Impedir a criação de portais em paredes onde eles não cabem;
- Permitir a visualização através dos portais, incluindo as visualizações recursivas (portais dentro de portais) e uma boa noção de perspectiva;
- Permitir a travessia dos portais.

Além disso, nos propomos a criar pelo menos um quebra cabeça a ser resolvido pelo jogador, utilizando os portais.

2. Conceitos e tecnologias estudadas

A biblioteca Irrlicht

A primeira tarefa necessária para a realização deste trabalho era o entendimento do funcionamento de uma máquina gráfica 3D, ou seja, conceitos de computação gráfica precisavam ser entendidos.

Juntamente à necessidade de entendimento na área de computação gráfica, imediatamente surgiu também a necessidade de relembrar todos os conceitos estudados em álgebra linear, já que estão totalmente relacionados.

Utilizamos, como ferramentas, uma *engine gráfica* chamada Irrlicht, que realizava toda a interface com OpenGL e automatizava o processo para renderização para nós, permitindo assim que nos focássemos na solução dos problemas relacionados a portais e facilitando a parte gráfica.

Dos problemas que citamos acima, a engine sozinha não era capaz de solucionar nenhum, porém, era capaz de prover um arcabouço de ferramentas para esta realização. Assim, vamos explicar o que era exigido de nós e a solução que utilizamos para resolver cada problema.

A engine era capaz de prover:

- Facilidade na manipulação da câmera, além de um grande arcabouço vetorial, extenso e muito bem documentado
- Criação de uma câmera em primeira pessoa sem a existência de um personagem atrelado a ela
- Facilidade na realização de renderizações a texturas, uma técnica que será explicada adiante e indispensável à realização do trabalho
- Possibilidade de carregamento de masmorras pré-montadas em programas específicos, além de modelos 3D, como o personagem e o labirinto em que o jogador será colocado
- Resposta a eventos de mouse e teclado
- Colisão básica com objetos identificados como "paredes", junto de uma gravidade superficial
- Fácil acesso à posição e rotação dos objetos, por intermédio de métodos em classes bem estruturadas, programadas em C++

Álgebra Linear

Para a solução dos problemas, será necessário ter boa noção acerca dos produtos vetoriais e produtos escalares entre vetores, e sua respectiva idéia de resultado. Vamos relembrar sucintamente o que cada um destes produtos indica.

O produto vetorial é uma operação binária entre dois vetores que tem como resultado um outro vetor, perpendicular ao plano formado pelos dois primeiros vetores e com direção dada pela regra da mão direita.

Já o produto escalar é a operação binária entre dois vetores cujo resultado é um escalar que indica, de certa forma, a relação angular entre os vetores.

Apresentado o arcabouço que temos e o que nos falta fazer, vamos às atividades realizadas, com as respectivas técnicas envolvidas.

Atividades realizadas

Criação de um mundo básico

O primeiro passo foi conseguir ter um modelo básico de um mundo explorável utilizando a biblioteca da engine Irrlicht. A documentação relacionada a esta biblioteca é bastante extensa e muito bem detalhada, o que facilitou bastante a criação de um início: um mundo 3D já consagrado no título da série Quake estava pronto à nossa disposição, necessitando apenas poucas linhas de comando para ter um simples labirinto e as funcionalidades básicas de movimentação.

A movimentação dos personagens

A movimentação é toda controlada pela biblioteca, pois esta nos fornece uma interface de manipulação de eventos, mas daremos uma breve idéia de como é dada a movimentação da câmera pelo cenário.

A entrada é dada pela "interface wasd", já conhecida nos jogos em primeira pessoa, em que o W move para frente, A para esquerda, S para trás e D para a direita.

O acionamento dos botões W ou S incrementa ou decrementa, respectivamente, a posição atual por um fator do vetor target da câmera, que indica para onde a câmera está olhando.

Já os botões A e D realizam um deslizamento lateral, obtido por uma multiplicação entre o vetor resultante do produto vetorial entre o vetor target e o vetor up da câmera, sendo que este último indica o que é o lado de cima no mundo para a câmera.

Desta forma, conseguimos realizar a movimentação via teclado. A rotação é dada pelo movimento do mouse, em que os movimentos horizontais realizam rotações no eixo X e os movimentos verticais realizam rotações no eixo Y. Para calcular a rotação da câmera pelo mouse, obtemos a posição do cursor na tela no frame atual e a comparamos com a posição do cursor no frame anterior, obtendo o ângulo de rotação que deve ser aplicado à câmera para que ele olhe na direção da posição do novo cursor. Impomos um limite de ângulo vertical de 90 graus, para que a câmera só consiga olhar para cima até certo ponto, simulando o movimento do pescoço do personagem.

Neste ponto, o primeiro objetivo seria entender como funcionaria a idéia de renderização dos portais, ou seja, como era realizado o processo de visualização do mundo existente do outro lado, na saída correspondente ao portal que estávamos visualizando.

Renderização

Para a criação de portais, foi utilizada uma técnica chamada Render to Texture (RTT), que possui suporte nos drivers de vídeo mais recentes e a partir da versão 1.3 do OpenGL, desde que sejam utilizadas algumas extensões. Esta técnica nos permite dar vida aos portais, de forma a permitir que o "outro lado" do mundo seja visível para o jogador, como se fosse uma porta para o "outro lado".

A técnica de RTT consiste numa manipulação de buffers de vídeo, gerando sua saída para uma textura, ao invés de levá-la à saída convencional: a tela. Sem a existência desta ferramenta, seria muito caro criar texturas "on the fly".

Assim, podemos criar um nó da cena (*scene node*, no inglês) como um retângulo, por exemplo, e aplicar uma textura a ele. Após isso, utilizamos a técnica de RTT e aplicamos nesta textura a renderização da visão de uma câmera que nos interesse.

Para criar o efeito de visualização dos portais, colocamos uma câmera no centro de cada um deles, olhando na direção de sua normal. Quando os dois portais estão criados, o processo de renderização pode ser resumido da seguinte forma (onde portal 1 e portal 2 são apenas nomes para diferenciar os dois portais):

- Deixamos o portal 2 invisível;
- Ativamos a câmera do portal 1, desativando assim todas as outras câmeras; Desenhamos na textura do portal 2 a renderização da câmera atual, utilizando a técnica RTT;
- Tornamos o portal 2 visível de novo;
- Deixamos o portal 1 invisível;
- Ativamos a câmera do portal 2, desativando assim todas as outras câmeras;
- Desenhamos na textura do portal 1 a renderização da câmera atual, utilizando a técnica RTT;
- Tornamos o portal 1 visível de novo;

Na Fig. 4 mostramos uma simples idéia de como seria o portal sem nenhuma texturização, ou seja, apenas um retângulo em branco.

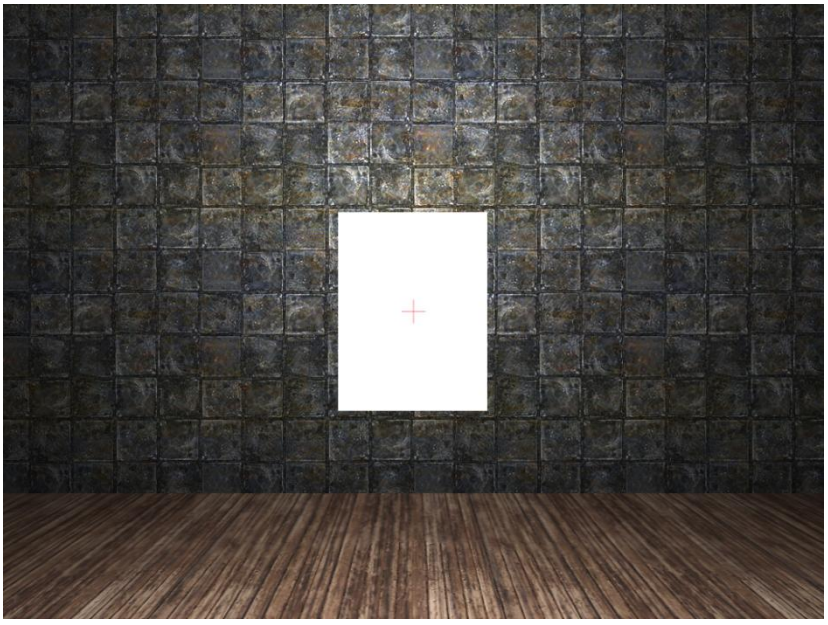

Figura 4: Portal antes da aplicação da textura renderizada

Convencionalmente, texturas são obtidas de imagens ou modelos pré-definidos durante um processo de carga da aplicação e apenas utilizadas em superfícies em tempo de execução. Para dar verossimilhança aos portais, seria necessário criar texturas em tempo de execução, que levariam em conta onde os portais estão localizados.

Utilizando extensões de OpenGL, é possível fazer com que a máquina gráfica renderize diretamente para o alvo desejado: um buffer que será utilizado como textura.

Este buffer especial utilizado para as texturas é chamado pbuffer (ou pixel buffer).

O processo ocorre da seguinte forma:

- Criar uma textura de renderização (chamados pixel buffer ou pbuffer), que não é passado à tela final
- Informar à máquina gráfica que o alvo de renderização será o pbuffer criado
- Renderizar a imagem
- Voltar o alvo de renderização à janela
- Ligar o pbuffer ao objeto de textura que utilizaremos
- Utilizar a textura como qualquer outra
- Liberar o pbuffer da textura

A Fig. 5 mostra o resultado da aplicação RTT no mesmo portal da Fig. 1.

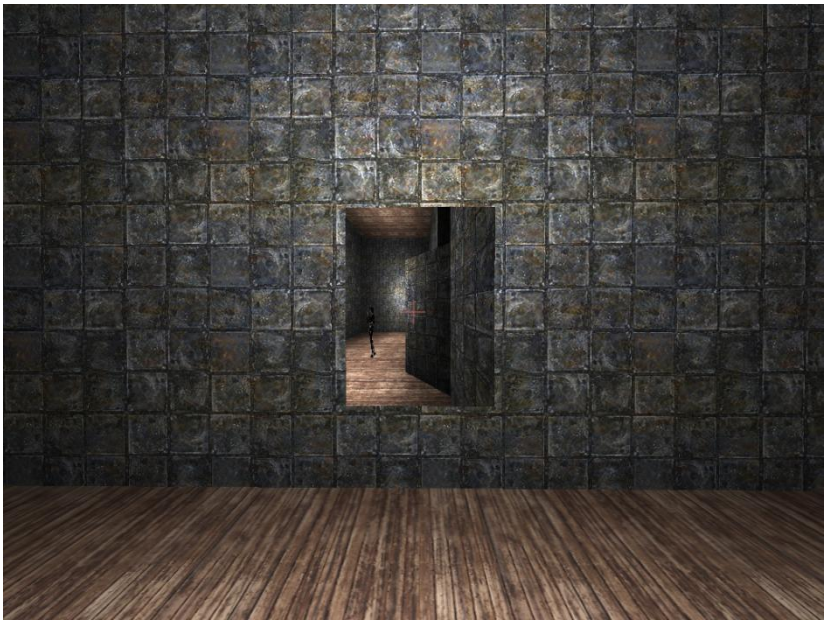

Figura 5: portal depois da aplicação da textura renderizada

Problema na movimentação relacionada à câmera

Porém, um problema ocorreu quando realizamos este tipo de manipulação de câmeras.

A câmera que estávamos utilizando para o jogador, fornecida pela engine, mantem atrelada à ela a movimentação do personagem. Isto acontece, pois na verdade, não há personagem; existe apenas uma câmera se movendo dentro do labirinto de acordo com as entradas do mouse e teclado, ou seja, a posição e a direção para onde a câmera está direcionada são trocados quando são dadas entradas.

Quando realizávamos a troca de câmera ativa, não havia um receptor de eventos ligado a estas câmeras, afinal, para o nosso processo, são necessárias 3 câmeras: o jogador, uma câmera em um portal e outra câmera no outro portal.

Como as câmeras relacionadas aos portais não possuem um receptor manipulador de eventos de entrada e mouse, a movimentação da câmera no cenário ficou bastante estranha, sofrendo diversas pequenas interrupções e travamentos à medida que as câmeras eram trocadas para renderização.

Desta forma, decidimos parar de usar esta câmera e passamos a utilizar uma câmera estática, atualizando sua posição a cada frame utilizando uma função externa. Esta função de atualização foi simplesmente copiada da própria engine, do

código da câmera que utilizávamos anteriormente, e adaptada de acordo com as nossas necessidades.

Modelo de personagem

Corrigida a câmera, precisávamos ter um modelo de personagem para acompanhar a câmera, afinal, não era interessante enxergar através do portal e ver que seu personagem na realidade não existia, era apenas uma câmera invisível no mundo.

Utilizamos um modelo pronto já disponível junto da biblioteca; trata-se da modelo Sydney. Realizamos alguns ajustes até encontrar uma altura ideal e deixá-la de acordo com a câmera existente em nosso mundo. Uma vez colocada no mundo, precisávamos que o modelo respeitasse as mudanças que aconteciam na câmera, tanto de rotação quanto de posicionamento. Seria interessante também que o personagem fosse animado de acordo com a situação: andando ou parado.

O primeiro passo foi ajustar a posição do modelo à posição da câmera a cada frame. Assim, teríamos o modelo seguindo a câmera corretamente. Depois, precisávamos que as rotações realizadas à câmera fossem refletidas no modelo. Para isto, modificamos novamente o nosso manipulador de eventos para que alterasse também as configurações do modelo.

Feitos estes passos, restava apenas um problema: às vezes, partes do modelo ficavam visíveis na câmera, pois parte do rosto ou braço às vezes ficavam na frente da visão e atrapalhavam. A solução foi configurar o modelo como não visível durante a renderização da tela apenas, mas visível durante a renderização dos portais. Assim, quando visualizasse os portais, o jogador conseguiria ver o seu modelo, mas não teria partes do corpo atrapalhando sua visão.

Criação e orientação dos portais nas superfícies

Tendo os portais renderizados, precisamos fornecer ao jogador um meio de colocá-los nas paredes desejadas. Para isso, precisamos estudar os métodos de colisão fornecidos pela engine e a forma como as paredes do mundo 3D são tratadas pela mesma.

Todas as paredes da cena, assim como todos os objetos 3D, são divididos em triângulos pela própria engine, para facilitar a sua renderização pela placa gráfica. Além disso, a engine fornece diversas funções que verificam colisões entre objetos 3D e estes triângulos.

As detecções de colisão com triângulos podem ser feitas por uma classe chamada "TriangleSelector", ou "seletor de triângulos". O seletor de triângulos contém um conjunto de triângulos que é um subconjunto de todos os triângulos da cena 3D e provê diversas funções de colisão, entre elas uma função que recebe uma linha 3D e decide se esta linha colide com algum triângulo do seu conjunto de triângulos, devolvendo este triângulo e o ponto de colisão. Caso a linha colida com mais de um triângulo, esta função devolve o primeiro triângulo onde ocorreu a colisão, partindo do começo da linha até seu fim.

Além disso, dispomos de meios de conseguir um vetor de direção que indica para onde a câmera controlada pelo jogador está olhando. Chamaremos este vetor de "target".

Assim, nós criamos um seletor de triângulos e adicionamos nele todos os triângulos das paredes do mundo 3D que carregamos pela engine e criamos uma linha 3D com início na posição da câmera controlada pelo jogador e fim a 1000 unidades à

frente, na direção do target. Passamos esta linha para o seletor de triângulos e temos em mãos o ponto do triângulo da parede para onde o jogador está olhando.

Queremos, então, que o jogador seja capaz de criar um portal com centro neste ponto. Para que possamos criar o retângulo do portal, precisamos saber as coordenadas de seus quatro vértices.

A primeira solução que tentamos para este problema foi utilizar o triângulo encontrado pelo seletor como base para desenharmos o retângulo. Desta forma, teríamos um retângulo na mesma superfície do triângulo, o que para nós parecia ser uma boa solução inicial. Tivemos esta idéia após encontrar um algoritmo que criava buracos de balas em paredes. Descreveremos agora a implementação desta primeira idéia, e os motivos que nos levaram a abandoná-la.

O triângulo é composto de 3 pontos, que chamaremos de A, B e C. Criamos uma linha, chamada linha 1, que começa em A e termina em B. Após isso, chamamos de D o ponto da linha 1 mais próximo do ponto C e criamos uma nova linha, chamada linha 2, que começa no ponto C e termina no ponto D. A representação destas linhas pode ser vista na Fig. 6, a seguir.

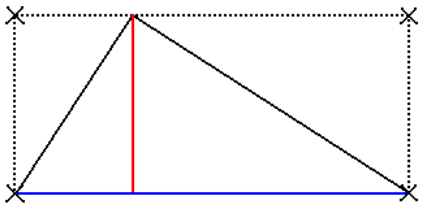

Figura 6: Vetores diretores a partir de um triângulo qualquer

Note que a linha 2 (vermelha) sempre será perpendicular à linha 1 (azul), pois se existe um ponto de uma reta mais próximo de um ponto fora dela do que qualquer outro ponto desta reta, os dois sempre estarão em uma reta perpendicular à primeira.

Assim, utilizando vetores diretores a partir dos pontos A e C, que apontam na direção destas linhas, conseguimos posicionar os vértices do retângulo nas posições indicadas na figura pelos X.

Esta solução realmente garante que nosso retângulo ficará no mesmo plano que o triângulo da parede, mas acarreta em diversos problemas. Como nós não sabemos qual é o formato do triângulo com o qual estamos lidando, não podemos ter

garantia nenhuma sobre a orientação, o tamanho e o formato de nosso retângulo. Em diversos casos, por exemplo, podemos ter como resultado um losango, ao invés de um retângulo. Além disso, essa solução seria inviável para resolver problemas futuros, como decidir se um portal cabe em uma determinada parede, por exemplo.

Observando estes resultados, nós ainda insistimos um pouco nesta solução, pensando em meios de rotacionar o polígono criado para que se assemelhasse a um retângulo. Após um tempo, decidimos que seria necessária uma solução melhor.

Apesar de errada, a primeira solução nos ensinou que precisaríamos de 2 vetores para desenhar nosso portal: um para desenhar sua largura, e outro para desenhar sua altura. Estes vetores devem garantir não só que o portal esteja no mesmo plano que o triângulo da parede, mas também que esteja orientado de forma correta, da mesma forma que um quadro deve ficar alinhado em uma parede.

Para tal, uma linha deve ser traçada, a partir do personagem, para encontrar o ponto de colisão com a superfície mais próxima à sua frente e utilizá-la como base, conforme descrito no processo acima. Nomearemos esta linha de L, e a normal do triângulo na superfície à frente será chamada de N.

A forma que utilizamos para orientar os portais funciona bem para qualquer superfície que não seja paralela ao chão (eixo XZ), mas não é difícil fazer um método genérico que permite orientar os portais corretamente para qualquer superfície. O método utilizado para o posicionamento dos portais é o seguinte:

Criamos um vetor diretor D, com coordenadas (0, 1, 0), ou seja, um vetor que aponta para cima (lembrando que na máquina 3D, o eixo X representa a largura, o eixo Y representa a altura e o eixo Z representa a profundidade).

Temos à disposição a normal do triângulo da parede, que chamaremos de N.

O produto vetorial do vetor D com o vetor N nos gera um vetor perpendicular aos dois, que pode ser utilizado para desenhar a largura do portal, pois ele sempre ficará na direção "horizontal" da parede. Fazendo um novo produto vetorial, deste vetor horizontal com a normal da superfície, obtemos um vetor que vai estar sempre "para cima" com relação à superfície. Este algoritmo, bastante simples, funciona muito bem para quaisquer superfícies que não sejam chão, teto ou paralelas.

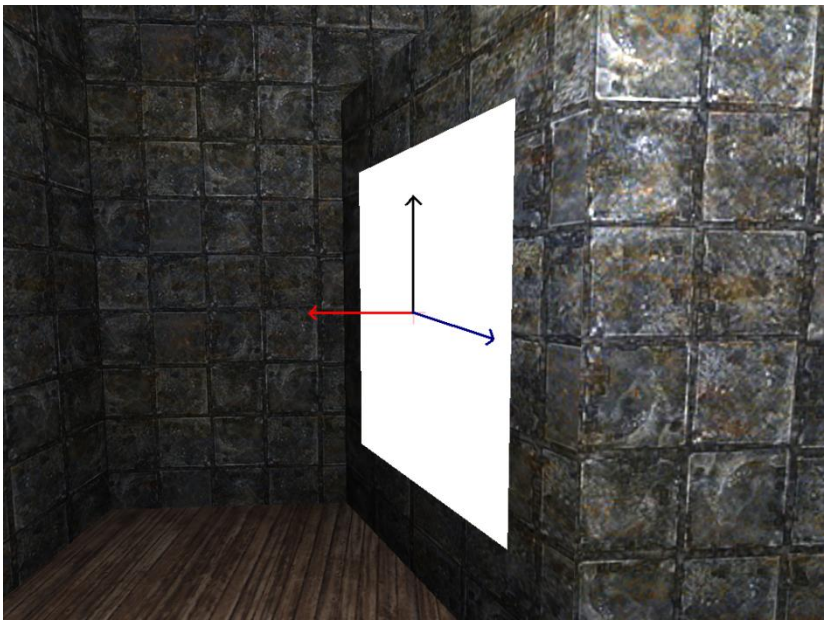

Figura 7: A multiplicação do vetor diretor (preto) com a normal da superfície (vermelho) sempre resultará em um vetor "horizontal" (azul) que pode ser usado para desenharmos o portal

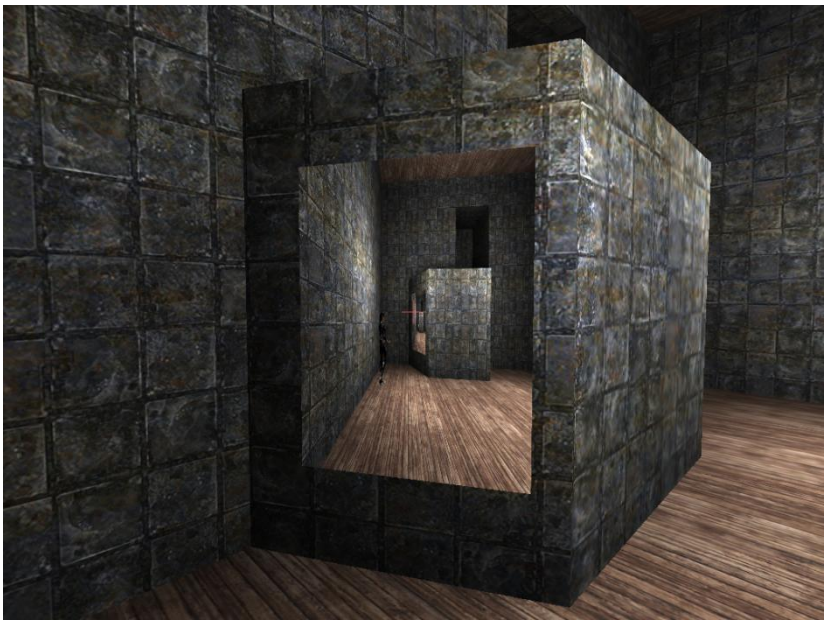

Figura 8: O algoritmo da foto anterior é aplicado à superfície e proporciona um posicionamento confiável

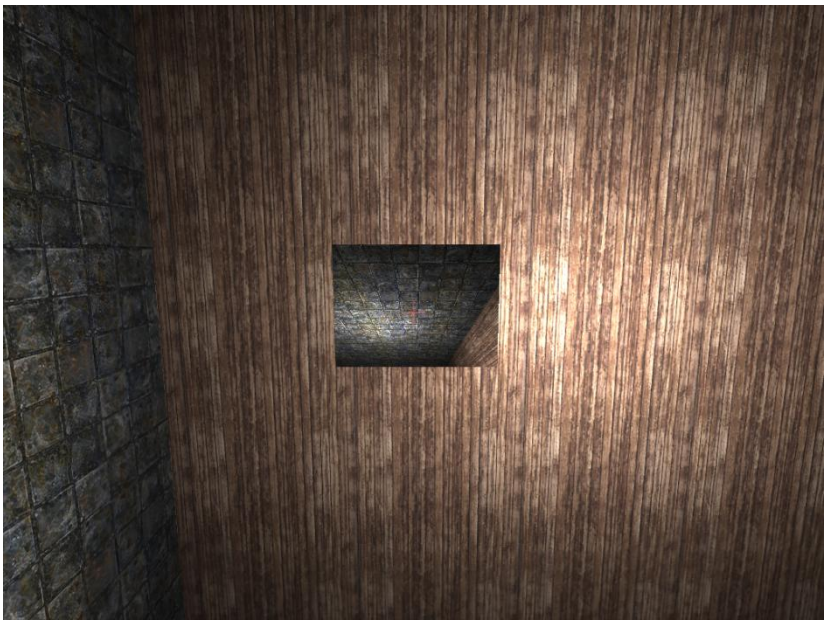

Figura 9: Portais no teto terão sempre a mesma orientação de baixo; problema e solução descritos a seguir.

A razão deste acontecimento é que, não importa qual vetor diretor D seja utilizado, os portais serão sempre orientados com a mesma referência de baixo e cima. É fácil visualizar o acontecimento imaginando a seguinte cena: escolha um vetor diretor D com coordenadas (0, 0, -1), ou seja, um vetor que está apontando, por padrão, para o "fundo" do mundo. Em qualquer ponto do chão ou teto, a normal será sempre (0, +/-1, 0). Este produto vetorial resulta sempre no mesmo vetor: (+/-1, 0, 0), ou seja, todos os portais terão como orientação horizontal o eixo X e como orientação vertical o eixo Z.

Para resolver o problema do posicionamento dos portais em superfícies paralelas ao eixo XZ, são utilizadas projeções de vetores para obtenção dos vetores horizontal e vertical da seguinte forma:

Se traçarmos uma linha que vai do observador até o ponto de intersecção com o chão ou teto, temos um vetor, que chamaremos de O (observador). Projetando este vetor na normal, conseguimos obter sua componente na vertical do mundo. Subtraindo-a do vetor O, seria como se "deitássemos" este vetor no chão ou teto. Utilizando-o como diretor vertical, e utilizando um produto vetorial com a normal, é possível obter um diretor horizontal.

Nível de profundidade dos portais

Com as técnicas até agora descritas, é possível criar um portal, renderizá-lo com o mundo visto através do outro portal e posicioná-lo com orientação correta. Porém, é possível que ocorra uma situação em que um portal consegue enxergar o outro, tornando necessário que o mundo seja redesenhado dentro do mundo visto pelo portal.

Uma vez feito o RTT de um portal, é como se tivéssemos tirado uma foto do mundo, com a visão do "outro lado". Se aplicarmos esta textura à parede e realizarmos o processo novamente, teremos uma "foto da foto", ou seja, conseguimos criar uma profundidade necessária para dar vida aos portais.

Apesar da explicação da técnica ser bastante simples e sua implementação mais ainda, chegar a esta conclusão foi bastante demorado.

Abaixo segue o processo de renderização modificado que suporta o caso em que um portal enxerga o outro:

- Deixamos o portal 2 invisível;
- Ativamos a câmera do portal 1, desativando assim todas as outras câmeras; Desenhamos na textura do portal 2 a renderização da câmera atual, utilizando a técnica RTT;
- Tornamos o portal 2 visível de novo;
- Desenhamos novamente na textura do portal 2 a renderização da câmera atual, utilizando a técnica RTT. Este passo é repetido n vezes, onde n é a profundidade que se deseja que o portal tenha;
- Deixamos o portal 1 invisível;
- Ativamos a câmera do portal 2, desativando assim todas as outras câmeras;
- Desenhamos na textura do portal 1 a renderização da câmera atual, utilizando a técnica RTT;
- Tornamos o portal 1 visível de novo;
- Desenhamos novamente na textura do portal 1 a renderização da câmera atual, utilizando a técnica RTT. Este passo é repetido n vezes, onde n é a profundidade que se deseja que o portal tenha;

Figura 10: Uma situação em que o alcance da visão de um portal contem o outro e vice-versa

Posicionamento dos portais nas superfícies

Durante o processo de criação de um portal, precisamos garantir que há espaço para alocação do mesmo na parede escolhida pelo jogador. Devemos então verificar se toda a extensão do portal será devidamente posicionada na parede.

Para isto, poderíamos simplesmente criar mundos 3D em que as paredes são construídas de forma a sempre suportarem os portais. Como não haveriam paredes tortas ou complicadas, todos os portais caberiam facilmente nelas. Ainda com estas limitações, conseguiríamos criar fases interessantes para nossos jogos.

Figura 11: Exemplo de posicionamento de portal que não deve ser permitido

Entretanto, optamos por criar um algoritmo que decide se o portal cabe em uma determinada parede, que funciona em qualquer mundo 3D. Assim, temos a liberdade de usar qualquer cena para utilizarmos nossos portais. Este algoritmo se baseia no fato de que a máquina gráfica divide todas as paredes da cena 3D em triângulos, para que a placa gráfica possa desenhá-las. O processo executado por este algoritmo é descrito a seguir:

- Utilizamos a linha L descrita anteriormente, que encontra o ponto de colisão com a parede mais próxima à frente do jogador e nomeamos este ponto de C.
- Obtemos também o triângulo da parede onde este ponto está contido. Chamaremos este triângulo de T.
- O ponto C será o centro do retângulo que forma o portal.
- A partir do ponto C, criamos quatro linhas, cada uma ligando o ponto C aos pontos onde ficarão os vértices do retângulo.
- Para cada linha, verificamos se a mesma está contida em T.
- Caso uma linha esteja contida em T, isto significa que o portal cabe na parede na direção daquela linha. Caso contrário, achamos o próximo triângulo T' da parede na direção da linha.
- Caso não haja nenhum, ou T' não esteja no mesmo plano de T, o algoritmo termina e descobrimos que o portal não cabe naquela parede.
- Caso contrário, continuamos o algoritmo a partir do novo triângulo.
- Decidimos que o portal cabe na parede quando as 4 linhas tiverem sido exploradas até o final.

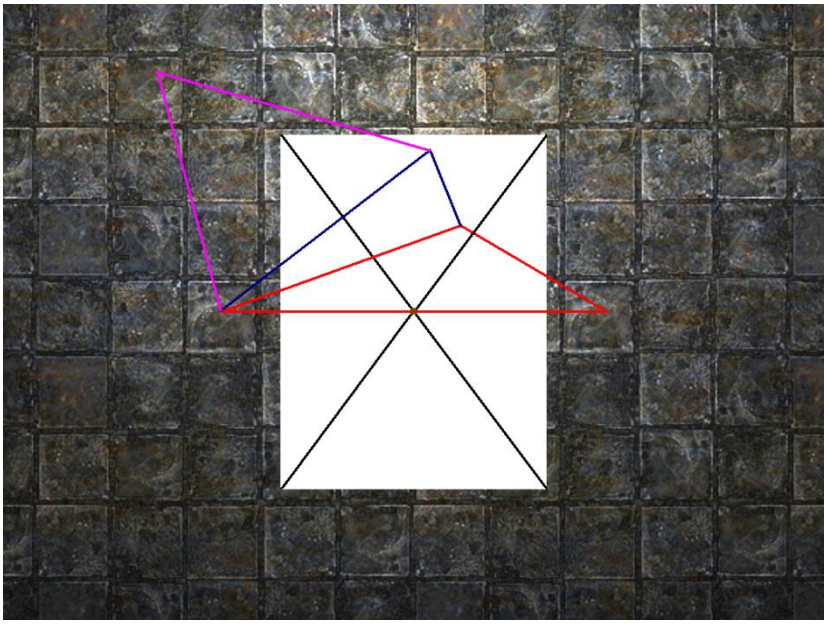

Figura 12: Simulação da busca feita pelo algoritmo para a linha superior esquerda do portal. Os triângulos da figura representam os triângulos da parede que foram encontrados. No caso, o algoritmo decide que o portal cabe na parede.

Nós tínhamos a idéia para este algoritmo pronta antes de tentarmos implementá-lo. Durante a sua implementação, esbarramos em dois problemas. Primeiramente, não há na engine um método para se decidir se duas linhas 3D se intersectam, pois é raro duas linhas 3D se intersectarem em um ponto (e às vezes impossível, devido à imprecisão do float). A cada iteração do algoritmo acima, nós precisamos decidir se as 4 linhas que criamos intersectam com algum lado do triângulo, para decidir se elas estão dentro do triângulo ou não.

O segundo problema foi descobrir um método de achar um triângulo adjacente a outro, na direção de uma das 4 linhas.

Para resolver o primeiro problema, decidimos utilizar um algoritmo que calcula a menor distância entre duas linhas 3D e comparar este valor com um valor limite. Assim, podemos decidir com uma boa aproximação se duas linhas 3D se intersectam.

Abaixo segue uma descrição deste algoritmo:

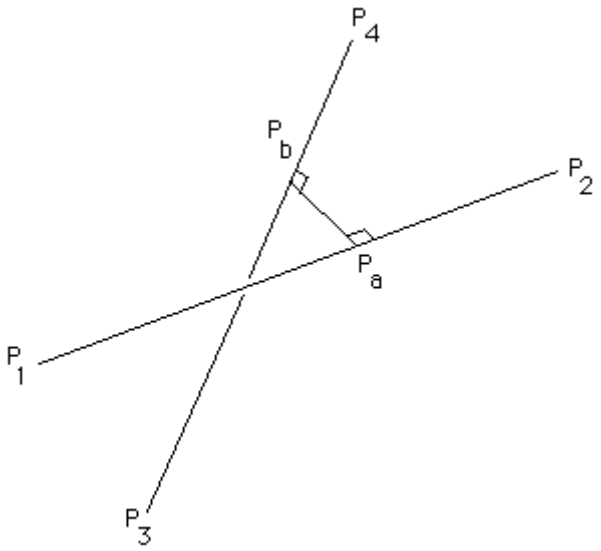

Figura 13: Menor distância entre 2 retas

Uma linha pode ser definida por 2 pontos que pertencem a ela. Assim, um ponto a em uma linha definida pelos pontos P1 e P2 tem a equação:

Pa = P1 + mua(P2 - P1)

E similarmente, um ponto b em uma segunda linha definida pelos pontos P1 e P2 tem a equação:

Pb = P3 + mub(P4 - P3)

Os valores mua e mub variam de menos infinito a mais infinito e são estes os valores que devemos encontrar. Para isso, basta percebermos que a reta que liga Pa a Pb é perpendicular às 2 linhas. Assim, podemos escrever:

(Pa - Pb) . (P2 - P1) = 0
(Pa - Pb) . (P4 - P3) = 0

onde "." é a operação de produto interno de vetores.

Expandindo estas equações utilizando as equações das linhas, temos:

(P1 - P3 + mua (P2 - P1) - mub (P4 - P3)) . (P2 - P1) = 0
(P1 - P3 + mua (P2 - P1) - mub (P4 - P3)) . (P4 - P3) = 0

A expansão destas equações utilizando as coordenadas x, y e z resulta nas seguintes equações:

d1321 + mua d2121 - mub d4321 = 0
d1343 + mua d4321 - mub d4343 = 0

onde dmnop = (xm - xn)(xo - xp) + (ym - yn)(yo - yp) + (zm - zn)(zo - zp)

obs: note que dmnop = dopmn

Finalmente, resolvendo para mua, temos:

mua = (d1343 d4321 - d1321 d4343) / (d2121 d4343 - d4321 d4321)

E reseolvendo para mub, temos:

mub = (d1343 + mua d4321) / d4343

Substituindo estes valores nas equações originais de Pa e Pb, obtemos os pontos Pa e Pb. Caso a distância entre eles seja menor do que um certo limite, decidimos que as duas retas se intersectam e tanto Pa quanto Pb podem ser usados como o ponto de intersecção.

O segundo problema, como dito anteriormente, consiste em achar o próximo triângulo da parede na direção de uma reta. Porém, não sabemos como o seletor de triângulos ordena seu conjunto de triângulos, se é que ele os ordena de alguma forma.

A solução que encontramos foi passar para o seletor de triângulos uma reta que passa pelo ponto de intersecção encontrado no algoritmo acima, e tem início a uma unidade de distância na direção da normal do triângulo atual e fim a uma unidade de distância na direção negativa da normal do triângulo atual. Transladamos esta reta uma unidade na direção da linha que estamos percorrendo atualmente e perguntamos ao seletor se ela intersecta com algum triângulo do seu conjunto de triângulos.

Para decidir se um triângulo está no mesmo plano que outro, comparamos suas normais. Ambos estarão no mesmo plano somente se suas normais forem iguais.

Perspectiva

Chegamos a um ponto em que conseguimos garantir que o portal está devidamente posicionado, preenchido, com profundidade, mas ainda falta algo: uma noção de perspectiva.

É importante que, quando o jogador se move, o portal de fato demonstre o que acontece do outro lado, para dar mais realidade à cena, ou seja, de acordo com a posição do personagem, um ajuste à câmera que representa o outro lado deve ser feito. O portal deve funcionar como se o jogador estivesse olhando através de uma janela. Quando uma pessoa olha para uma janela e se move para a esquerda, ela consegue enxergar coisas que estavam mais para a direita, no mundo exterior. Queremos reproduzir isto nos portais.

Novamente, utilizamos projeções de vetores para resolver o problema. A solução é bastante simples:

- Obtemos a norma de V, que é o vetor que vai do personagem até o centro do portal
- Normalizamos os seguintes vetores: o vetor V, o vetor DH que é o diretor horizontal do portal e o vetor DV que é o diretor vertical do portal utilizado pela biblioteca para a criação do portal na parede, no passo de orientação dos portais
- Calculamos a projeção do vetor V em DH e DV. Desta forma, temos a componente vertical e horizontal (com relação ao portal) do vetor V. Chamaremos estas projeções de pDH e pDV.
- Subtraímos de N, que é a normal da parede onde o vetor se encontra (também normalizada) os valores pDH e pDV. Assim sendo, temos agora as 3 componentes que formam o vetor V, todas perpendiculares
- Realizamos a mudança de forma correspondente na câmera que está do outro lado, utilizando a norma de pDH, pDV e V - pDH - pDV

O último passo da lista acima é feito da seguinte forma:

Como pDH e pDV são obtidos como múltiplos de DH e DV respectivamente, então, sua norma representa diretamente o quanto "para direita" ou "para cima" o personagem está do portal.
Se seguirem o mesmo sentido de DH e DV, significa que o personagem está à direita ou acima do portal, então, utilizaremos sua normal invertida como multiplicador dos vetores diretores da câmera presente no outro lado.

Atravessar

Para concluir, é necessário permitir que o personagem atravesse o portal de um lado para o outro. Isto é feito de forma simples: quando o jogador se aproxima de um portal, mudamos a sua posição para a frente do outro portal. A visão da câmera é modificada de modo a fazer com que o jogador esteja olhando na direção da normal do portal de saída. É importante notar que diversos objetos podem atravessar os portais, além do próprio jogador. Estes objetos podem ser utilizados para adicionar desafios em jogos que utilizam nossos portais.

Criação da fase

Tendo a mecânica dos portais pronta, nós decidimos criar uma fase jogável que os utilizasse. A idéia desta fase é criar um objetivo a ser atingido pelo jogador, que só pode ser cumprido por meio do uso dos portais.

Para a criação da fase, utilizamos a ferramenta GTK Radiant, que permite a criação de fases para o jogo Quake, entre outros.

Este programa possui uma interface simples. Basicamente, existem quatro janelas. Uma delas permite a visualização do mapa a partir de uma câmera controlável, para que o usuário possa verificar o que está criando. As outras três janelas possuem um plano cartesiano cada (uma para o plano XY, outra para o plano XZ e outra para o plano YZ). Nestas três janelas, podemos desenhar retângulos que representam as paredes, chãos e tetos do mundo 3D. Estes retângulos podem ter qualquer largura, altura e profundidade que se deseje. A única restrição existente é que o poliedro resultante da união de todos os poliedros criados deve ser completamente fechado.

Os paralelepípedos criados podem também ser cortados. Assim, podemos criar formas como prismas, pirâmides, etc.

Após a criação do mundo 3D, nós o importamos na engine e partimos para a criação dos objetivos e obstáculos. Em nossa fase, o jogador começa em uma sala simples, que possui uma plataforma elevada a qual ele não consegue acessar com seus movimentos normais. Nesta plataforma encontra-se uma porta e atrás dela o final da fase, que deve ser alcançado pelo jogador. Ao lado da porta existe um botão que o jogador não consegue alcançar. Quando algo atinge este botão, a porta se abre.

A porta e o botão são simples paralelepípedos, criados com uma chamada da engine que pede o seu centro e suas dimensões.

No nível mais baixo da sala há uma partícula se movendo e rebatendo nas paredes utilizando um simples cálculo de reflexão. Para passar da fase, o jogador deve fazer a partícula acertar o botão e abrir a porta, utilizando portais. Após isso, o jogador deve se tele portar para a plataforma e sair pela abertura.

A colisão da partícula com os portais é detectada da mesma forma que a colisão do personagem com os portais, utilizando o seletor de triângulos.

Para calcular a reflexão da partícula nas paredes, calculamos a projeção do vetor incidente (trajetória da partícula) no vetor normal da parede. Criamos um vetor r, resultado da diferença entre o vetor incidente e esta projeção. Assim, o vetor de reflexão é:

reflexão = -1 * incidente + 2 * r

Quando a partícula colide com uma parede, sua trajetória passa a ser este vetor de reflexão.

Resultados e produtos obtidos

As atividades acima realizadas nos permitiram criar um pequeno e simples exemplo de uma fase para o jogo. Adicionamos uma partícula cuja movimentação é controlada automaticamente: possui uma velocidade inicial que se manterá constante e, ao deparar-se com um obstáculo, é refletida de acordo com o ângulo de colisão.

Com isto, conseguimos analisar o funcionamento das nossas técnicas e avaliar a qualidade do que fizemos. A movimentação foi satisfatória, a criação de portais funcionou de acordo com o esperado, a renderização adicionou a realidade esperada à cena inclusive com sua profundidade.

Porém, a perspectiva não ficou da forma como esperávamos, ou seja, a movimentação do personagem diante do portal realizava uma mudança no que era visto, porém, não estava sendo realizada de forma satisfatória. Este é um trabalho pendente que pretendemos consertar em nosso trabalho futuro.

A travessia entre os portais também ficou insatisfatória; no jogo que nos inspirou, a travessia era realizada de forma bastante suave, sem trancos ou tele portes, além de permitir que o personagem entrasse no portal de costas e, ao sair, continuasse com esta orientação. Em nossa implementação, o personagem é forçado para olhar para frente assim que chega do outro lado, num tele porte brusco. Nossas tentativas de suavizar este efeito não obtiveram sucesso.

Além disto, é desejável adicionar uma física mais realista e que seja capaz de enfrentar situações como ter o personagem saindo de um portal no chão. Este tipo de situação, no cenário atual que temos, faria com que o personagem fosse automaticamente jogado de volta, enquanto que no jogo Portal, o personagem seria suavemente passado para o lado de fora e movimentado para uma lateral do portal.

Conclusões

Cada tarefa realizada no trabalho foi importante para tornar o projeto cada vez mais completo e funcional.

A criação de um labirinto 3D foi um passo final que nos permitiu ter certeza de que nossas técnicas estavam corretas.

A movimentação do personagem no mundo estava satisfatória, o modelo funciona de acordo com a movimentação do jogador, posicionando-se e rotacionando-se corretamente, além de ter animação correta, o posicionamento dos portais estava correto e restringia posicionamentos ruins, além de dar a orientação desejada.

Por fim, a visualização através dos portais funcionou conforme o esperado, apenas a noção de perspectiva que ainda deve ser melhorada para se tornar mais suave, e a transição do jogador de um portal a outro também não ficou como desejado. Estas tarefas serão melhoradas na continuação do projeto que será realizada por nós.

O desenvolvimento de todas as técnicas acima nos permitiu criar uma estrutura propícia para até criar um jogo ou inventarmos novas aplicações, de forma bastante satisfatória. A aplicação que obtivemos está bastante semelhante à que nos motivou, nos permitindo aprender todas as técnicas e descobrir como funcionava esta aplicação tão fascinante.

Apesar dos pontos que faltaram na implementação, fomos capazes de produzir um jogo jogável e funcional.

O trabalho será continuado com melhorias e refatoração do código, que ainda não está devidamente fatorado, além da remoção do nosso código específico de dentro do corpo do programa, permitindo que nosso trabalho possa ser utilizado como biblioteca ou ferramenta externa a um projeto já existente.

Parte Subjetiva

Motivação do trabalho

A idéia de realizar este projeto surgiu no começo do ano, em diversas conversas entre eu e o Otavio. A idéia dele era fazer algo relacionado a um jogo, e nosso gosto comum nos fez chegar até um ponto em que queríamos obter um trabalho tanto divertido quanto de conteúdo, onde pudéssemos ter muito o que aprender.

Eu já havia cursado a matéria de Computação Gráfica e estava fascinado pelo assunto, porém, em tudo que havia aprendido, não fazia idéia de como poderia fazer algo semelhante ao jogo que nos inspirou. Ao questionar o Otavio a respeito, chegamos á conclusão de queríamos não só criar algo, mas criar algo que não sabíamos como seria, na intenção de aprender.

Foi a partir deste ponto que conversamos com nossos futuros orientadores, professores doutores Marcel Jackowski e Flavio Soares a respeito do assunto e se este poderia ser um trabalho válido. Concordamos então em trabalhar neste assunto.

Desafios e Frustrações

Apesar de já ter cursado a disciplina de Computação Gráfica no IME, ainda havia muito que aprender. Fizemos tarefas teorias e práticas em sala, porém, em sua totalidade, lidando apenas e diretamente com OpenGL. Para a realização do nosso trabalho, seria impossível atuar sem um framework ou biblioteca que facilitasse o nosso trabalho, foi quando iniciamos a pesquisa quanto a engines para uso.

Nosso primeiro desafio foi entender o funcionamento da biblioteca Irrlicht, que possui uma documentação extremamente vasta e bem estruturada. Seguimos diversos tutoriais e víamos a capacidade da engine.

Logo de início também percebemos que precisaríamos muito dos conceitos de álgebra linear que haviam sido ensinados e já parcialmente, esquecidos, necessitando retomar os conhecimentos via consulta aos materiais guardados.

A partir do momento que começamos a desenvolver o trabalho, a cada passo que conseguíamos avançar nos sentíamos mais motivados a continuar.

Durante o desenvolvimento do trabalho, sempre atuamos juntos discutindo as decisões e soluções que encontramos, tanto as boas quanto as ruins. Isto facilitou a realização do projeto e programamos juntos todas as fases. Cada etapa do projeto era discutida.

Feliz ou infelizmente, durante a realização do nosso projeto, consultamos duas vezes o orientador Marcel, o que o surpreendeu na última visita que fizemos, ocorrida uma semana antes da apresentação, deixando-o surpreso com os resultados. Acredito que esta liberdade e independência que tomamos nos ajudaram a nos organizarmos e nos policiarmos quanto às tarefas que tínhamos que realizar.

Também por esta razão, ao conversarmos com o professor Flavio por e-mail, que estava ausente em um congresso, explicamos o andamento e perguntamos se ele desejaria continuar como nosso orientador, devido à mínima consulta realizada, apenas na proposta e por fim antes da apresentação. O professor optou por deixar o professor Marcel como único orientador, devido à maior consulta.

As orientações do orientador Marcel foram decisivas na criação de um pôster elegante e com conteúdo atraente, além do auxílio na estrutura da apresentação.

Relação com as disciplinas cursadas

- **MAC 0110**: Introdução à computação
  MAC 0122: Princípios de desenvolvimento de algoritmos
  Apesar de já ter contato com programação mesmo antes de ingressar no IME, estas matérias ensinaram boas noções a respeito da estrutura de uma linguagem, seus tipos, rigidez ou flexibilidade, além de noções importantes do funcionamento da memória de um programa.
- **MAT0139**: Álgebra Linear
  Os conhecimentos aprendidos em álgebra linear foram vastamente utilizados e aplicados. Durante o curso, pude entender um pouco da idéia de como funcionam espaços 3D e como seriam dadas as operações, e finalmente pude entendê-los na prática em Computação Gráfica
- **MAC0211**: Laboratório de Programação I
  MAC0242: Laboratório de Programação II
  Estas disciplinas nos ensinaram a trabalhar em equipe, estipular metas e separar o trabalho em *checkpoints* para realização. Além disto, nas disciplinas, foram desenvolvidos jogos, área diretamente ligada ao nosso interesse.
- **MAC0420**: Computação Gráfica
  Disciplina essencial para entender os conceitos relacionados à computação gráfica, como funcionamento da câmera, posicionamento de objetos, operações vetoriais e matriciais. A aplicação destes conceitos foi também vastamente utilizada.
- **MAC0332**: Engenharia de software
  Os princípios de trabalho em equipe, importância de documentação e boas práticas de programação, ainda que não aplicados em sua extensão, foram de extrema utilidade para tentarmos manter um código organizado e bem estruturado.
- **FAP0126**: Física I
  Os conceitos de mecânica foram utilizados para regular diversos valores do programa de forma a torná-lo mais realista.

Trabalho futuro

Prendemos continuar trabalhando no projeto, tornando a nossa ferramenta uma biblioteca e documentá-la para uso. Ainda há ajustes a serem feitos, conforme citado anteriormente, e estas tarefas ainda serão concluídas.

Há interesse por nossa parte em trabalhar na área de jogos, tentando nos aperfeiçoar na área.

Referências bibliográficas

[1] IRRLICHT. Irrlicht Engine 1.6 API Documentation [online]. Disponível na internet via WWW. URL: http://irrlicht.sourceforge.net/docu/index.html

[2] WIKIPEDIA. A enciclopédia livre [online]. Disponível na internet via WWW. URL: http://pt.wikipedia.org/wiki/Produto_vetorial

[3] WIKIPEDIA. The free encyclopedia [online]. Disponível na internet via WWW. URL: http://en.wikipedia.org/wiki/Vector_projection

[4] KAYILI, Ihsan. Projection of a Vector onto another vector [vídeo online]. Disponível na internet via WWW. URL: http://www.youtube.com/watch?v=-6i8bBCil-4

[5] VALVE, Software. Portal [online]. Disponível na internet via WWW. URL: http://www.whatistheorangebox.com/portal.html

[6]BOURKE, Paul. The Shortest Line Between Two Lines In 3D [online]. Disponível na internet via WWW. URL: http://local.wasp.uwa.edu.au/~pbourke/geometry/lineline3d/

[7]GTK RADIANT. Integrated SCM & Project Management [online]. Disponível na internet via WWW. URL: http://www.qeradiant.com/cgi-bin/trac.cgi

[8] WYNN, Chris. Nvidia OpenGL Render-To-Texture [online]. Disponível na internet via WWW. URL: http://www.nvidia.com.br/attach/6725