INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Gustavo Ceci Guimarães

DESENVOLVIMENTO DE UMA FRAMEWORK EM C++ PARA CRIAÇÃO DE JOGOS 3D PARA A WEB

Trabalho de Graduação
2017

Curso de Engenharia de Computação

CDU 621.38

Gustavo Ceci Guimarães

DESENVOLVIMENTO DE UMA FRAMEWORK EM C++ PARA CRIAÇÃO DE JOGOS 3D PARA A WEB

Orientador

Prof. Dr. Edgar Toshihiro Yano (ITA)

ENGENHARIA DE COMPUTAÇÃO

São José dos Campos
Instituto Tecnológico de Aeronáutica

2017

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão de Informação e Documentação

Guimarães, Gustavo Ceci
Desenvolvimento de uma framework em C++ para criação de jogos 3d para a Web / Gustavo Ceci Guimarães.
São José dos Campos, 2017.
35f.

Trabalho de Graduação – Curso de Engenharia de Computação– Instituto Tecnológico de Aeronáutica, 2017. Orientador: Prof. Dr. Edgar Toshihiro Yano.

1. Jogos. 2. WebGL. 3. OpenGL. 4. WebAssembly. 5. Emscripten. 6. C++. I. Instituto Tecnológico de Aeronáutica. II. Título.

REFERÊNCIA BIBLIOGRÁFICA

GUIMARÃES, Gustavo Ceci. **Desenvolvimento de uma framework em C++ para criação de jogos 3d para a Web**. 2017. 35f. Trabalho de Conclusão de Curso (Graduação) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSÃO DE DIREITOS

NOME DO AUTOR: Gustavo Ceci Guimarães
TITULO DO TRABALHO: Desenvolvimento de uma framework em C++ para criação de jogos 3d para a Web.
TIPO DO TRABALHO/ANO: Trabalho de Conclusão de Curso (Graduação) / 2017

Gustavo Ceci Guimarães
Rua H8B 238
12.228-461 – São José dos Campos–SP

DESENVOLVIMENTO DE UMA FRAMEWORK EM C++ PARA CRIAÇÃO DE JOGOS 3D PARA A WEB

Essa publicação foi aceita como Relatório Final de Trabalho de Graduação

---

Gustavo Ceci Guimarães

Autor

Edgar Toshihiro Yano (ITA)

Orientador

Prof. Dr. Cecilia César
Coordenadora do Curso de Engenharia de Computação

São José dos Campos, 22 de novembro de 2017.

Às ovelhas mais lentas, sem vocês não teria chegado até aqui.

Agradecimentos

Gostaria de agradecer a Sir Isaac Newton, cuja linhagem de orientandos chegou aos professores Vieira Dias e Edgar Yano, fundamentais para o desenvolvimento e aprovação do presente trabalho.

Gostaria de agradecer aos professores medíocres e ruins do ITA, sem vocês eu não teria me dedicado mais a projetos paralelos e aprendido grande parte do que eu sei sobre desenvolvimento de jogos.

Gostaria de agradecer aos poucos bons professores do ITA, obrigado por tentarem tornar esse lugar um ambiente menos desagradável de se estudar.

Por fim quero agradecer a todas as pessoas que gostariam de ser agradecidas na minha seção de agradecimentos do meu TG, o nome de vocês não está aqui por razões pessoais, mas caso vocês queiram, podem se sentir agradecidos por mim. Obrigado.

Não tem como dar errado.

Resumo

Aqui começa o resumo do referido trabalho. Não tenho a menor idéia do que colocar aqui. Sendo assim, vou inventar. Lá vai: Este trabalho apresenta uma metodologia de controle de posição das juntas passivas de um manipulador subatuado de uma maneira subótima. O termo subatuado se refere ao fato de que nem todas as juntas ou graus de liberdade do sistema são equipados com atuadores, o que ocorre na prática devido a falhas ou como resultado de projeto. As juntas passivas de manipuladores desse tipo são indiretamente controladas pelo movimento das juntas ativas usando as características de acoplamento da dinâmica de manipuladores. A utilização de redundância de atuação das juntas ativas permite a minimização de alguns critérios, como consumo de energia, por exemplo. Apesar da estrutura cinemática de manipuladores subatuados ser idêntica a do totalmente atuado, em geral suas caraterísticas dinâmicas diferem devido a presença de juntas passivas. Assim, apresentamos a modelagem dinâmica de um manipulador subatuado e o conceito de índice de acoplamento. Este índice é utilizado na sequência de controle ótimo do manipulador. A hipótese de que o número de juntas ativas seja maior que o número de passivas ($n_a > n_p$) permite o controle ótimo das juntas passivas, uma vez que na etapa de controle destas há mais entradas (torques nos atuadores das juntas ativas), que elementos a controlar (posição das juntas passivas).

Abstract

Well, the book is on the table. This work presents a control methodologie for the position of the passive joints of an underactuated manipulator in a suboptimal way. The term underactuated refers to the fact that not all the joints or degrees of freedom of the system are equipped with actuators, which occurs in practice due to failures or as design result. The passive joints of manipulators like this are indirectly controlled by the motion of the active joints using the dynamic coupling characteristics. The utilization of actuation redundancy of the active joints allows the minimization of some criteria, like energy consumption, for example. Although the kinematic structure of an underactuated manipulator is identical to that of a similar fully actuated one, in general their dynamic characteristics are different due to the presence of passive joints. Thus, we present the dynamic modelling of an underactuated manipulator and the concept of coulpling index. This index is used in the sequence of the optimal control of the manipulator.

Lista de Figuras

Sumário

1 Introdução

A indústria de jogos é um dos setores da indústria de entretenimento que mais gera lucro, gerando em 2016 mais lucro que o setor de musica e de filmes (NASDAQ, 2016). Essa indústria é dividida em diversos outros subsetores como o desenvolvimento de jogos para video-game, jogos mobile e jogos de computador. Dentro do setor dos jogos de computador pode-se tambem fazer algumas subdivisões como jogos que precisam ser instalados e jogos que podem ser jogados no browser.

Apesar de ser a menor parcela do setor no quesito de lucro, os jogos de browser ainda possuem relevância no contexto geral, e muitas vezes servem como o ambiente inicial onde o jogo pode ser jogado (NEWZOO, 2017). Por não necessitarem de software baixado além do navegador e serem tipicamente gratuitos, os jogos de browser são de acessibilidade muito simples para o usuário. Para os desenvolvedores, especialmente aqueles sem investimentos multimilionários voltados para marketing e publicação, a acessibilidade dos jogos de browser torna-se um método de veículação e engajamento de usuário muito eficiente. Diversos jogos independentes bem sucedidos possuem ou possuiram em algum momento alguma versão ou protótipo possivel de ser jogado no browser (JOGO, 2011; JOGO, 2015; JOGO, 2016).

Além do apelo financeiro, jogos podem ser usados como forma de expressão artística e como um hobby. Como para as grandes empresas jogos são um investimento multimilionário, os resultados são em sua maioria jogos menos inovadores que aqueles desenvolvidos por pequenas empresas ou desenvolvedores que o fazem por hobby, uma vez que o risco envolvido para estes costuma ser bem menor (MCGUIRE; JENKINS, 2009).

Inicialmente, grande parte dos jogos de browser eram desenvolvidos utilizando-se a ferramenta Adobe Flash, porém, com o surgimento do HTML5 e do WebGL, o uso de um plugin terceirizado para rodar jogos tem sido cada vez mais mal visto, incentivando o desenvolvimento usando as tecnologias padrão encontradas no browser (UNITY, 2015; FLASH, 2015).

Uma limitação sempre presente quanto à evolução dos jogos de browser é a limitação da velocidade de processamento do browser, especialmente quando os jogos são em ambientes tridimensionais. A linguagem mais utilizada para o desenvolvimento de jogos 3d

é C++, especialmente devido à eficiência, gerenciamento de memória e comunicação com sistemas de baixo nível como a placa de video. Como javascript é a linguagem padrão dos browsers e sua eficiência é consideravelmente inferior à de C++ (BENCHMARKS, 2017), a implementação de jogos 3d com graficos otimizados no browser sempre foi deixada de lado. No entanto com o crescimento cada vez maior das tecnologias dos browsers tornou-se necessário cada vez mais ter códigos mais eficientes e com velocidades mais próximas à nativa, surgindo assim diversas soluções para se executar codigos nativos no browser como PNaCL (PNACL, 2011) e WebAssembly (WASM, 2015).

1.1 Motivação

As ferramentas existentes hoje em dia para desenvolvimento de jogos com suporte para exportar para web são em sua maioria Engines completas, como Unity3d, Unreal Engine e Godot. Estas possuem inúmeras utilidades como simulação de física, criação de interface de usuário (UI), gerenciador de animações para modelos 3d, entre outras. Para ser possível a união de tantas ferramentas em uma unica Engine, o usuário acaba se vinculando demais à arquitetura da mesma e soluções improváveis acabam se tornando mais difíceis de se implementar devido à generalização. Para a maioria dos casos a Engine serve perfeitamente ao desenvolvedor, no entanto, em alguns casos a falta de controle é relevante.

Esse trabalho tem como objetivo criar não uma engine, mas uma framework que dá ao usuário a capacidade de pular as tarefas mais complicadas de se desenvolver uma engine, como um mecanismo simples de input e uma interface mais simples de utilização da placa de vídeo que o OpenGL padrão.

A motivação pessoal para o desenvolvimento desse trabalho é poder ter uma ferramenta propria para desenvolvimento de jogos, com capacidade de exportar para Web. Além disso, esse projeto visa ampliar os conhecimentos sobre os sistemas de baixo nível no desenvolvimento de jogos, especialmente um melhor entendimento de API's de gráfico 3d e da linguagem C++ em um projeto de maior escala.

1.2 Objetivo

O objetivo do trabalho é ter uma framework de facil distribuição e fácil uso no sistema operacional Windows. Para a criação de jogos, a framework terá como objetivo abstrair as camadas de entendimento complexo relacionado ao uso do OpenGL e do STL. A framework deverá ter uma interface simples de posicionamento de formas geométricas no espaço (sendo um bonus a possibilidade de se importar modelos 3d) além da implemen-

tação de um sistema de cameras básico. Além disso, a framework deverá possuir uma interface simples de leitura de input do teclado e do mouse. Por fim, caso se mostre necessário, a framework poderá ter integrada uma biblioteca de UI imediata para casos de debug e desenvolvimento.

1.3 Ferramentas

1.3.1 Emscripten

Emscripten é um compilador de LLVM para javascript. LLVM é uma infraestrutura de compilador que serve para facilitar a criação de compiladores e transpiladores. Dada qualquer linguagem compilada para a linguagem intermediária do LLVM, a ferramenta Emscripten pode ser utilizada para ser enfim compilada pra javascript (ZAKAI, 2011).

O principal objetivo de compilar LLVM para javascript é a compilação de C++. Além disso, o Emscripten possui já implementadas algumas bibliotecas importantes para o uso no browser, sendo elas a transformação de OpenGL para WebGL e a biblioteca SDL2.

1.3.2 OpenGL

OpenGL, *Open Graphics Library* é uma API multiplataforma padrão para desenvolvimento de graficos 3D. É um padrão de API 3D amplamente aceito com uso significante no mundo real.(MUNSHI DAN GINSBURG, 2008). A versão do OpenGL usada nos browsers, com a API em javascript é conhecida como WebGL, que é baseada no OpenGL ES 2.0, versão essa voltada para dispositivos móveis e sistemas embarcados. Como o emscripten é capaz de transformar um subconjunto de OpenGL ES 2.0 em WebGL sem maiores problemas (EMSCRIPTEN, 2015), esse será o subconjunto utilizado no projeto.

1.3.3 SDL2

SDL2, *Simple DirectMedia Layer* é uma biblioteca de desenvolvimento multiplataforma criada para prover acesso de baixo nível aos hardwares de audio, mouse, teclado, joystick, alem de possuir uma interface com OpenGL (SDL2, 2015). Como é uma biblioteca muito utilizada para desenvolvimento de jogos em C++ e o emscripten já possui uma versão própria da biblioteca já compilada, essa foi a escolha de biblioteca base para a multimídia do trabalho.

2 Desenvolvimento

2.1 Hello World

A primeira funcionalidade que foi implementada no projeto foi utilizar-se da biblioteca OpenGL e seguir os passos iniciais para ser possivel desenhar um triangulo na tela.

Para isso foi implementada uma funcionalidade basica de criação de janela, utilizando-se do SDL2 para que isso fosse multiplataforma e funcionasse com o compilador do Emscripten.

Em seguida foram criadas classes básicas para a criação de shaders, código necessário para que os vertices e pixels da tela sejam coloridos. Após a capacidade de criação de shaders, os pontos de um triangulo foram colocados no loop principal do programa, e utilizando das funções do OpenGL, foi possível renderizar um triangulo conforme a figura 2.1.

2.2 Criação de modelos

A partir do momento que era possivel renderizar um triangulo, a renderização de uma lista de triangulos tornou-se trivial. Assim, o proximo passo do desenvolvimento foi fazer uma função que renderizasse uma lista de vértices quaisquer de um modelo para que fosse possivel renderizar vários ao mesmo tempo.

Para teste da criação de multiplos modelos, foi escrita à mão um array que continha todos os vertices dos triangulos de um cubo, e assim surgiu a primeira versão da função de desenhar cubos.

Em seguida para que multiplos objetos pudessem utilizar do mesmo modelo, mas serem posicionados em lugares diferentes da tela, foi implementada uma matriz de modelo, utilizando-se dos conceitos explicados no apendice Representação Matricial de Pontos, Vetores e Transformações

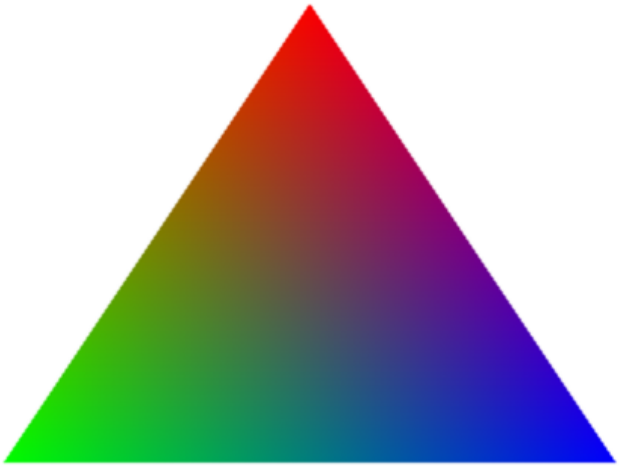


FIGURA 2.1 – Hello World do OpenGL

A cada vértice de um modelo é aplicada a transformação definida pela seguinte matriz:

$$M = T * R * S \tag{2.1}$$

Onde T é a translação do modelo, R é a rotação do modelo e S é a escala do modelo. Assim, para cada modelo novo é enviada a matriz dele para o shader de vértices e nesse shader cada vértice é multiplicado por essa matriz.

2.3 Criação de uma camera com perspectiva

Após a possibilidade de criação e posicionamento de modelos, a proxima coisa importante de ser implementada seria uma camera, para primeiramente poder renderizar a cena vista de diversas posições e também para poder utilizar da perspectiva e os jogos criados parecerem verdadeiramente tridimensionais.

Primeiramente a ideia de mostrar a cena sendo vista de uma posição diferente a ideia principal é mover o mundo em relação à camera. Dessa forma, dado que a camera tenha uma translação T_c e uma rotação R_c, podemos definir uma matriz que muda os objetos da sua posição no espaço para sua posição em relação à camera como:

$$V = (T_c * R_c)^{-1} \tag{2.2}$$

A matriz V é conhecida como matriz *View*. Para todos os objetos agora, multiplica-se

todos os vértices do seu modelo pela matriz do modelo, e em seguida pela matriz V, fazendo com que todos se posicionem em relacao à camera.

O proximo passo foi implementar a camera visualizar a cena com uma perspectiva tridimensional, para isso utilizando-se dos conceitos explicados no apendice Representação Matricial de Pontos, Vetores e Transformações, da matriz de perspectiva, tem-se a matriz P e portanto, surge agora uma matriz muito importante, que transforma um modelo em um objeto em perspectiva no espaço em relação à camera.

$$MVP = P * V * M \tag{2.3}$$

Essa matriz é constantemente atualizada, toda vez que a camera se move, e enviada para a GPU para cada objeto a ser renderizado. Com ela toda a renderização 3d se torna possível e para criar as classes relacionadas à renderização 3d (Graphics, Camera, Transform) bastou-se separar as diferentes funcionalidades que deveriam ser publicas e separar as tarefas para cada parte diferente do projeto.

Em seguida, para testar alguns conceitos, a biblioteca gráfica foi expandida para mais próxima de sua versão final, com uma implementação simples de iluminação difusa e especular.

2.4 Input do Usuário

Após a implementação da biblioteca grafica, o proximo passo para tornar a imagem renderizada em um jogo seria a capacidade do código levar os inputs do usuário em consideração. Para a entrega inicial, foi implementado apenas input do mouse e do teclado.

A primeira coisa a ser implementada para o sistema de input era um sistema ainda mais genérico e que é util em diversos contextos de um jogo, um sistema de eventos. Nesse sistema podem ser atribuidas funções relacionadas a um evento e quando esse evento é disparado, todas as funções são chamadas.

Dado um sistema de eventos funcional, utilizando-se de ferramentas proporcionadas pela biblioteca SDL2, a implementação do sistema de input do usuario por eventos foi simples.

Por fim, para ser possíver fazer ações com consequencias depois de um determinado intervalo de tempo, foi implementada uma classe simples de callbacks temporais, na qual é possível dizer um tempo e uma função, e esta será chamada depois do tempo.

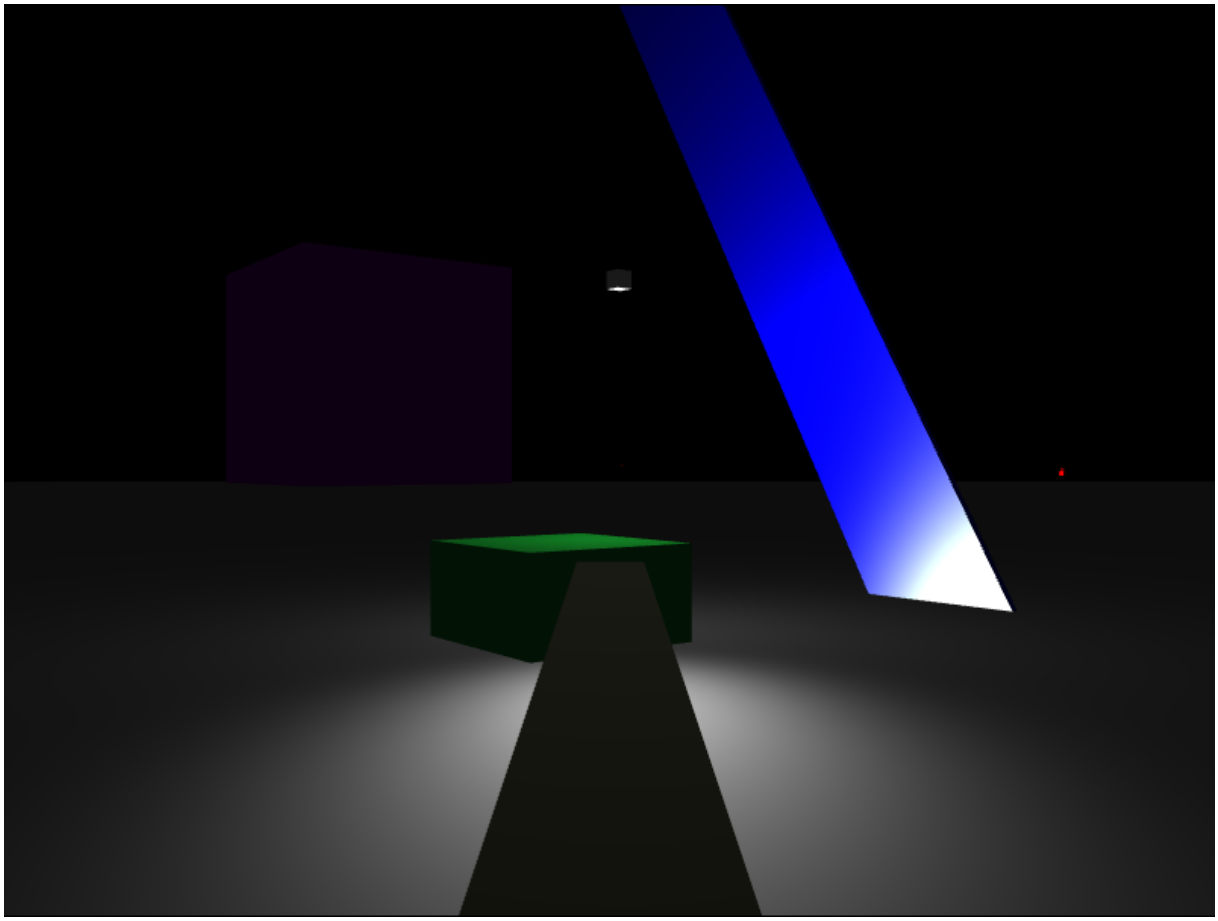


FIGURA 2.2 – Jogo implementado utilizando-se a framework

2.5 Provas de Conceito

Paralelo ao desenvolvimento da framework, alguns demos de uso da mesma foram sendo desenvolvidos. O mais elaborado deles foi um jogo em primeira pessoa no qual existem alguns cubos se movendo no espaço, um paralelepipedo sempre no centro da camera. Nesse jogo WASD é utilizado para se movimentar, o mouse serve para rodar a camera em relação ao eixo y e o clique do mouse atira pequenos cubos vermelhos. A figura 2.2 mostra uma imagem do jogo.

O código desenvolvido pode ser encontrado em `https://github.com/Ghust1995/my-cpp-engine/tree/love-like`

3 Documentação

Nesse capítulo as principais interfaces publicas da aplicação serão explicadas. Como explicado no Desenvolvimento, foram implementados duas bibliotecas: Namespace Graphics, Namespace Time, e 6 classes: Classe Game, Classe Transform, Classe Camera, Classe EventSystem, Classe Scheduler, Classe KeyboardInput, Classe MouseInput,

3.1 Classe Game

A classe game é a classe principal executada pela framework, ela é composta de 3 métodos que devem ser implementados:

3.1.1 void Load()

Chamada uma única vez no código antes do jogo inicializar qualquer coisa. Essa função tem como intenção alocar as variaveis e coisas que não serão executadas a todo frame.

3.1.2 void Update()

Chamada a todo frame e tem como intenção atualizar as variaveis do jogo.

3.1.3 void Draw()

Chamada todo frame com a intenção de desenhar coisas na tela. Essa função é otimizada pra utilizar a biblioteca Graphics.

3.2 Classe Transform

A classe transform é um container de translação, rotação e escala, tendo também algumas funções auxilares muito usadas com essa informação.

3.2.1 Transform()

O construtor da Classe transform pode receber um vetor de translação, um quaternião de rotação e um vetor de escala.

3.2.2 glm::vec3 forward()

Esse método retorna o vetor (0, 0, -1) rotacionado da rotação da Transform.

3.2.3 glm::vec3 up()

Esse método retorna o vetor (0, 1, 0) rotacionado da rotação da Transform.

3.2.4 glm::vec3 right()

Esse método retorna o vetor (1, 0, 0) rotacionado da rotação da Transform.

3.2.5 void Translate(glm::vec3 translation)

Esse método muda a posição da Transform, fazendo uma translação definida pelo parametro *translation*.

3.2.6 void Rotate(float angle, glm::vec3 axis)

Esse rotaciona a Transform, fazendo uma rotação definida pelo eixo *axis* e pelo angulo *angle*.

3.2.7 glm::vec3 position

A posição da Transform

3.2.8 glm::vec3 rotation

A rotação da Transform

3.2.9 glm::vec3 scale

A escala da Transform

3.3 Classe Camera

A camera é um objeto utilizado pela biblioteca Graphics para determinar como a cena será renderizada, toda cena deverá possuir pelo menos uma camera.

3.3.1 Camera(const float fov, const float aspect_ratio, const float near, const float far)

O construtor da camera recebe basicamente as informaçoes sobre o *frustrum*. Cria uma camera na posição (0, 0, 0) e sem rotação (olhando para o eixo z negativo).

3.3.2 void Translate(glm::vec3 translation)

Esse método muda a posição da Transform da camera, fazendo uma translação definida pelo parametro *translation*.

3.3.3 void Rotate(float angle, glm::vec3 axis)

Esse rotaciona a Transform da camera, fazendo uma rotação definida pelo eixo *axis* e pelo angulo *angle*.

3.3.4 void SetPosition(glm::vec3 position)

Esse método seta a posição da Transform da camera.

3.3.5 void SetRotation(glm::quat rotation)

Esse método seta a rotação da Transform da camera.

3.3.6 glm::vec3 position()

Esse método pega a posição da Transform da camera.

3.3.7 glm::quat rotation()

Esse método pega a rotação da Transform da camera.

3.4 Classe EventSystem

3.4.1 EventSystem()

Cria um novo event system.

3.4.2 void Add(Event e, Callback callback)

Adiciona um novo callback para quando o evento e for disparado.

3.4.3 void Fire(Event e)

Dispara o evento e.

3.5 Classe KeyboardInput

Essa classe serve para transformar inputs do teclado do usuário em funções do jogo.

3.5.1 KeyboardInput(SDL_KeyMapping *keymapping, const int mappingsize)

O construtor recebe uma lista de mapeamento de teclas para um nome de ação e o tamanho dessa lista.

3.5.2 void BindAction(const char *name, const InputType type, const std::function<void()> callback)

Essa função registra a função callback para ser chamada quando a ação *name* do tipo *type* for chamada.

InputType pode ser INPUT_DOWN (quando a tecla é pressionada), INPUT_UP (quando a tecla é solta) e INPUT_HOLD (quando ela está sendo segurada);

3.6 Classe MouseInput

Essa classe serve para transformar inputs do mouse do usuário em funções do jogo.

3.6.1 MouseInput(SDL_MouseKeyMapping *keymapping, const int mappingsize)

O construtor recebe uma lista de mapeamento de botões do mouse para um nome de ação e o tamanho dessa lista.

3.6.2 void BindAction(const char *name, const InputType type, const std::function<void()> callback)

Análoga à função BindAction da KeyboardInput.

3.6.3 void BindMovement(const std::function<void(const MouseMovementData *d)> callback)

Registra uma função que recebe informações sobre a movimentação do mouse para ser chamada toda vez que o mouse se movimentar.

3.7 Classe Scheduler

Essa classe serve para fazer chamadas de funções após um tempo em segundos.

3.7.1 Scheduler()

Cria um novo scheduler.

3.7.2 void Update(SchedulerTime delta_time)

Atualiza o scheduler do tempo que passou. Geralmente o uso disso é chamar ela no Update do Game com delta_time = Time::GetDelta().

3.7.3 void Add(SchedulerTime dt, SchedulerTask task)

Adiciona a função task para ser chamada após dt segundos.

3.7.4 void Repeat(SchedulerTime delay, SchedulerTime dt, SchedulerTask task)

Adiciona a função task para ser repetida a cada dt segundos, depois de um delay.

3.7.5 void Repeat(SchedulerTime dt, SchedulerTask task)

Adiciona a função task para ser repetida a cada dt segundos (com delay inicial de dt segundos).

3.8 Namespace Time

Namespace com funções globais relacionadas a tempo.

3.8.1 float GetDelta()

Retorna o tempo que passou entre o ultimo frame e o frame atual.

3.8.2 float GetTotal()

Retorna o tempo total de jogo desde sua inicialização.

3.9 Namespace Graphics

3.9.1 void Cube()

Existem duas versoes, a Cube(glm::vec3 position, glm::quat rotation = glm::quat(), glm::vec3 scale = glm::vec3(1.0f)), e a Cube(Transform transform), a primeira recebe uma posição, uma rotação e uma escala, e posiciona um cubo, a outra recebe uma Transform e posiciona um cubo baseado nas mesmas informações. Essa função é uma prova de conceito da futura função que importaria um objeto genérico, que nao foi implementada por motivos de simplicidade.

3.9.2 void SetMaterial(glm::vec3 diffuse$_color$, $glm :: vec3 specular_color$)

Essa função seta o materia que vai ser usado a partir desse momento, dependendo do shader atual, o material pode ou não afetar em propriedades como como a luz interage

com o objeto.

3.9.3 void PointLight(glm::vec3 position, glm::vec3 color, float intensity)

Cria uma luz pontual na posicao definida, com cor e intensidade dada pelos parametros. Envia essas informações que podem ou não ser utilizadas pelo atual shader.

3.9.4 void SetAmbientLight(glm::vec3 color)

Seta a luz ambiente para uma determinada cor. Envia essas informações que podem ou não ser utilizadas pelo atual shader.

3.9.5 void SetClearColor(glm::vec3 color)

Seta a cor que é o plano de fundo do jogo (padrão preto).

3.9.6 void SetCamera(const Camera *camera)

Faz com que a atual camera que está renderizando a cena seja a setada por essa função.

3.9.7 void Clear()

Limpa todos os poligonos criados até então.

4 Considerações Finais

O objetivo geral proposto foi cumprido com sucesso. Foi feita uma implementação de uma framework simples, nos moldes da framework Love2D e XNA, com simplificações. Alem disso foram implementadas algumas funcionalidades não propostas no objetivo que facilitam profundamente a implementação de um jogo, em especial o sistema de eventos e o de chamadas temporais.

O projeto pode ser continuado através da implementação de mais funcionalidades nele para expandir ainda mais o leque de possibilidades de implementação de jogos com a framework.

Algumas funcionalidades já planejadas para a framework são:

4.0.1 Alta prioridade

- Permitir o uso de uma biblioteca de UI imediata para permitir alterações em parametros do jogo em tempo real e criação de uma interface de desenvolvimento durante o jogo.
- Implementação de um importador de modelos 3d e criação de mais algumas formas geométricas como: triangulo, quadrado, esfera e piramide.
- Implementar a utilização de texturas, para que seja possível que os objetos tenham não só uma unica cor, mas desenhos mais complexos.

4.0.2 Media prioridade

- Fazer com que a camera possa renderizar para lugares diferentes da tela (fazer 2 camera que mostram coisas diferentes em cada metade da tela),
- Melhorar a performance para suportar ainda mais objetos em cena.

4.0.3 Baixa prioridade

- Extensão da interface input_handler para receber inputs de joysticks, e tambem criação de uma classe KeyboardMouseInput, que junta teclado e mouse em uma única funcionalidade.

- Expandindo as melhorias da camera e da textura, porém outra tarefa de igual dificuldade, seria poder colocar a imagem da camera em uma textura, abrindo margem para diversas possibilidades graficas, como espelhos.

Referências

BENCHMARKS: The computer language benchmarks game. 2017. Disponível em: <http://benchmarksgame.alioth.debian.org/>.

EMSCRIPTEN: Opengl support in emscripten. 2015. Disponível em: <https://kripken.github.io/emscripten-site/docs/porting/multimedia_and_graphics-/OpenGL-support.html>.

FLASH: The death of adobe flash. 2015. Disponível em: <http://kotaku.com/the-death-of-flash-is-coming-and-not-everyones-happy-1717824387>.

JOGO: The binding of isaac. 2011. Disponível em: <https://en.wikipedia.org/wiki/The_Binding_of_Isaac_(video_game)>.

JOGO: Clicker heroes. 2015. Disponível em: <https://en.wikipedia.org/wiki/Superhot>.

JOGO: Superhot. 2016. Disponível em: <https://en.wikipedia.org/wiki/Superhot>.

MCGUIRE, M.; JENKINS, O. C. **Creating games: mechanics, content, and technology**. 3rd. ed. Wellesley, Mass: AK Peters, Ltd, 2009. 33–34 p.

MUNSHI DAN GINSBURG, D. S. A. **OpenGL ES 2.0 Programming Guide**. [S.l.]: Pearson Education, 2008.

NASDAQ: Investing in video games industry. 2016. Disponível em: <http://www.nasdaq.com/g00/article/investing-in-video-games-this-industry-pulls-in-more-revenue-than-movies-music-cm634585/>.

NEWZOO: Global games market report. 2017. Disponível em: <https://newzoo.com/insights/articles/the-global-games-market-will-reach-108-9-billion-in-2017-with-mobile-taking-42/>.

PNACL: Google native client. 2011. Disponível em: <https://developer.chrome.com/native-client>.

SDL2: Simple directmedia layer homepage. 2015. Disponível em: <https://www.libsdl.org/index.php>.

UNITY: The computer language benchmarks game. 2015. Disponível em: <https://blogs.unity3d.com/2015/10/08/unity-web-player-roadmap/>.

WASM: The computer language benchmarks game. 2015. Disponível em: <http://webassembly.org/>.

ZAKAI, A. Emscripten: an llvm-to-javascript compiler. **OOPSLA '11 Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion**, p. 301–312, out. 2011.

Apêndice A - Representação Matricial de Pontos, Vetores e Transformações

A.1 Pontos e Vetores

Model Matrix

Quando se tratando de pontos no espaço tridimensional que serão transformados de diversas formas, é comum utilizar a notação matricial para pontos e vetores de forma que toda transformação se resume a uma multiplicação de matrizes.

Assim, um ponto $p = (p_x, p_y, p_z)$, é representado por:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{A.1}$$

Analogamente um vetor $v = (v_x, v_y, v_z)$, é representado por:

$$\begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} \tag{A.2}$$

O quarto elemento da matriz têm algumas propriedades interessantes, que serão demonstradas adiante, mas um primeiro fato relevante é que a subtração de dois pontos resulta num vetor.

Com essa representação matricial de pontos e vetores surgem tambem representações matriciais para diversos tipos de transformações:

A.2 Transformações

A.2.1 Translação

A matriz que define uma translação de t_x no eixo x, t_y no eixo y e t_z no eixo z é definida da seguinte forma:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.3}$$

Assim, para aplicar uma transformaçao em um ponto basta multiplicar a matriz pelo ponto:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{bmatrix} \tag{A.4}$$

Da mesma forma, pode-se aplicar uma matriz de translação a um vetor, tendo o seguinte resultado:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} \tag{A.5}$$

Percebe-se que pela característica de seu ultimo elemento ser 0, a translação não afeta o vetor, sendo esse o comportamento esperado.

A.2.2 Escala

Outra transformação muito comum é a de escalar vetores e pontos. Escala é uma multiplicação termo a termo dos elementos de uma tripla ordenada por outra. A matrix de escala é definida da seguinte forma:

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.6}$$

No contexto de pontos, a escala afasta o ponto da origem por um fator s_x no eixo x, s_y no eixo y e s_z no eixo z. A aplicação da matrix no ponto é dada pela equação a seguir:

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} p_x * s_x \\ p_y * s_y \\ p_z * s_z \\ 1 \end{bmatrix} \tag{A.7}$$

De forma análoga, a matriz de escala muda o tamanho do vetor em cada um de seus eixos, sendo sua aplicação dada pelo valor a seguir.

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} v_x * s_x \\ v_y * s_y \\ v_z * s_z \\ 0 \end{bmatrix} \tag{A.8}$$

A.2.3 Rotação

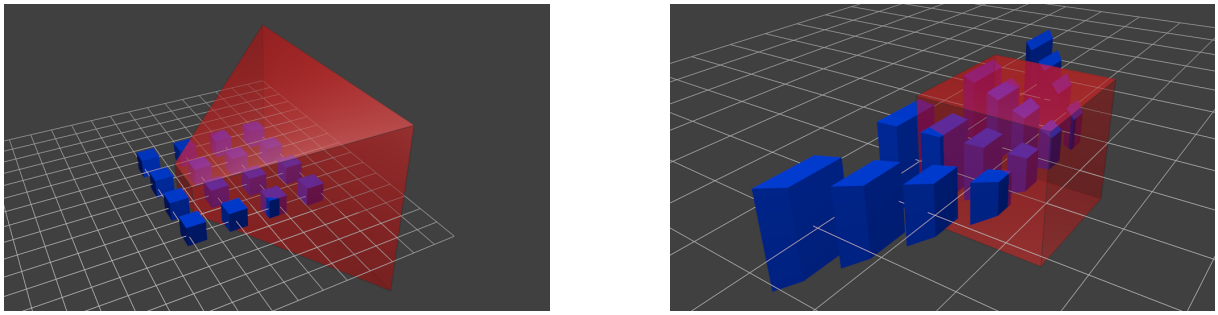
Rotações são operações complexas, e portanto uma outra matemática foi desenvolvida para ela, a dos quaterniões, que não será explicada em detalhes no presente relatório. No entanto mesmo que o uso de quaterniões para composição e interpolações seja mais simples, os mesmos são transformados em matrizes de rotação e utilizados da mesma forma que as transformações anteriores. Um exemplo da matriz de rotação mais utilizada que é a rotação de um ângulo θ por um eixo $u = (u_x, u_y, u_z)$, representada a seguir:

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z \sin\theta & u_x u_z(1-\cos\theta) - u_y \sin\theta & 0 \\ u_y u_x(1-\cos\theta) - u_z \sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x \sin\theta & 0 \\ u_z u_x(1-\cos\theta) - u_y \sin\theta & u_z u_y(1-\cos\theta) - u_x \sin\theta & \cos\theta + u_z^2(1-\cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.9}$$

A.2.4 Projeção

Outra transformação muito útil na renderização 3d é a de projeção, sendo a mais utilizada delas a de projeção em perspectiva. A matrix de projeção em perspectiva transforma os pontos de forma que distancias em x e y são afetadas pela posição z. Isso torna-se muito mais claro observando as seguintes imagens:

A matriz de perspectiva geralmente é definida por uma forma conhecida como *frustrum*, um tronco de pirâmide no qual todos os objetos internos são projetados em relação à parte menor. O formato do frustrum pode ser observado na figura A.1a. Para tanto são necessárias alguns parâmetros, a distancia no eixo z do plano proximo (parte menor),



(a) Antes (b) Depois

FIGURA A.1 – Aplicação da matriz de perspectiva a pontos no espaço

representada por n_z, a distância no eixo z do plano distante (base), representada por f_z, o ângulo de abertura da pirâmide, *fov*, e a razão entre a largura e a altura da base da piramide, representada por *ar*.

Com esses parâmetros, é possível definir a matriz de projeção em perspectiva da seguinte forma:

$$S = \begin{bmatrix} \frac{1}{ar*\tan(\frac{fov}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{fov}{2})} & 0 & 0 \\ 0 & 0 & \frac{-n_z-f_z}{n_z-f_z} & 0 \\ 0 & 0 & 1 & \frac{2*n_z*f_z}{n_z-f_z} \end{bmatrix} \tag{A.10}$$

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
TC	22 de março de 2017	DCTA/ITA/TC-018/2015	35

5. TÍTULO E SUBTÍTULO:

Desenvolvimento de uma framework em C++ para criação de jogos 3d para a Web

6. AUTOR(ES):

Gustavo Ceci Guimarães

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica – Divisão de Engenharia Mecânica – ITA/IEM

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Cupim; Cimento; Estruturas

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Cupim; Dilema; Construção

10. APRESENTAÇÃO: (**X**) **Nacional** ( ) **Internacional**

ITA, São José dos Campos. Curso de Mestrado. Programa de Pós-Graduação em Engenharia Aeronáutica e Mecânica. Área de Sistemas Aeroespaciais e Mecatrônica. Orientador: Prof. Dr. Adalberto Santos Dupont. Coorientadora: Profª. Drª. Doralice Serra. Defesa em 05/03/2015. Publicada em 25/03/2015.

11. RESUMO:

Aqui começa o resumo do referido trabalho. Não tenho a menor idéia do que colocar aqui. Sendo assim, vou inventar. Lá vai: Este trabalho apresenta uma metodologia de controle de posição das juntas passivas de um manipulador subatuado de uma maneira subótima. O termo subatuado se refere ao fato de que nem todas as juntas ou graus de liberdade do sistema são equipados com atuadores, o que ocorre na prática devido a falhas ou como resultado de projeto. As juntas passivas de manipuladores desse tipo são indiretamente controladas pelo movimento das juntas ativas usando as características de acoplamento da dinâmica de manipuladores. A utilização de redundância de atuação das juntas ativas permite a minimização de alguns critérios, como consumo de energia, por exemplo. Apesar da estrutura cinemática de manipuladores subatuados ser idêntica a do totalmente atuado, em geral suas caraterísticas dinâmicas diferem devido a presença de juntas passivas. Assim, apresentamos a modelagem dinâmica de um manipulador subatuado e o conceito de índice de acoplamento. Este índice é utilizado na sequência de controle ótimo do manipulador. A hipótese de que o número de juntas ativas seja maior que o número de passivas ($n_a > n_p$) permite o controle ótimo das juntas passivas, uma vez que na etapa de controle destas há mais entradas (torques nos atuadores das juntas ativas), que elementos a controlar (posição das juntas passivas).

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **CONFIDENCIAL** ( ) **SECRETO**