UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RELATÓRIO DA 2ª UNIDADE
IMPLEMENTAÇÃO DE UM BRAÇO ROBÓTICO COM OPENGL

GRUPO Nº 1

FRANKSON SOUZA DOS SANTOS: Nº 20200149069

JULIA COSTA C. DE OLIVEIRA: Nº 20200149318

SAMUEL CAVALCANTI: Nº 20200149318

TIAGO FELIPE DE SOUZA: Nº 20190153105

Natal-RN
2022

FRANKSON SOUZA DOS SANTOS: Nº 20200149069

JULIA COSTA CORREA DE OLIVEIRA: Nº 20200149318

SAMUEL CAVALCANTI: Nº 20200149318

TIAGO FELIPE DE SOUZA: Nº 20190153105

IMPLEMENTAÇÃO DE UM BRAÇO ROBÓTICO COM OPENGL

Relatório de apresentação à disciplina de Computação Gráfica, que corresponde totalmente a avaliação da 2ª unidade do semestre 2021.2 do curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte, sob orientação do **Prof. Dr. Davi Henrique dos Santos.**

Professor: Dr. Davi Henrique dos Santos.

Natal-RN
2022

RESUMO

Este relatório apresenta os detalhes de produção do projeto final, contemplando as notas da segunda e terceira unidade da disciplina de Computação Gráfica (DCA 0114) na Universidade Federal do Rio Grande do Norte (UFRN), ministrada pelo Professor Dr. Davi Henrique dos Santos. Esse projeto consiste na criação de um braço robótico móvel; é possível movê-lo e mover a visão do usuário utilizando apenas algumas teclas do teclado. Para isso, a principal ferramenta utilizada foi o OpenGL através da linguagem de programação Python (pyOpenGL) e suas abstrações. Esse projeto teve como principal objetivo afirmar e firmar os conhecimentos adquiridos durante as aulas teóricas e práticas. No fim é possível afirmar que os objetivos propostos foram alcançados, tendo em vista que durante a criação do braço robótico foi possível utilizar as funções da biblioteca OpenGL e suas extensões, pondo em prática o que foi aprendido na teoria.

Palavras-chave: Computação Gráfica; OpenGL; Python; PyOpenGL; Braço Robótico.

Lista de Figuras

Sumário

1 INTRODUÇÃO

Na indústria do entretenimento, na engenharia, na arquitetura, no design e na comunicação há uma área do conhecimento que está muito presente: a computação gráfica. A computação gráfica está em todo lugar, sendo um grande elo entre a humanidade e os computadores. Essa começou apenas com um computador capaz de processar e projetar imagens tridimensionais em monitores ou televisores comuns e hoje é responsável por desde efeitos especiais em filmes até simulações de campos de batalha militares.

Quando se faz um desenho em um papel ou pinta uma tela, se trata de uma imagem analógica que traduz uma ideia e se assemelha a algo que existe no mundo real. Porém, quando feita no papel essa ideia é difícil ou as vezes impossível de modificar. Já na computação gráfica, é possível manipular as criações a qualquer momento e de qualquer forma. Depois ainda é possível imprimir, enviar para alguém, colocar em vídeos e etc.

Uma ferramenta importante para a computação gráfica, que também fora utilizada neste trabalho, como será melhor discutido posteriormente, é a biblioteca pyOpenGL, que se trata do OpenGL(*Open Graphics Library*) para *Python*. O OpenGL é uma biblioteca com funções específicas disponibilizadas para a criação e desenvolvimento de aplicativos em determinadas linguagens de programação, sendo *Python* a utilizada para este trabalho. Usando as suas funções, é possível definir vários aspectos do que está sendo desenvolvido, tais como a cor, iluminação, textura, sombreamento, transformações e assim por diante. Além disso, afim de facilitar a manipulação de aplicações são utilizadas três abstrações: GL, GLU e GLUT. Onde GLU fornece diversas funções para auxiliar na montagem de matrizes de visualização e projeção, desenho de superfícies e imagens 3D e o GLUT é um *toolkit* que abstrai o sistema de janelas, fornecendo uma interface simples para a criação de uma aplicação OpenGL.

Neste trabalho foram utilizadas essas funções da computação gráfica afim de implementar um braço robótico. Esse braço é formado por: duas garras e dois corpos de textura metálica, e é alocado em cima de um piso com textura. Além disso, é possível tanto movimentá-lo quanto mudar a visão do usuário utilizando apenas as teclas de um teclado, sendo essas (em maiúsculo e minúsculo): A, S, B, C e as setas.

2 EMBASAMENTO TEÓRICO

2.1 TRANSFORMAÇÕES

Um dos principais conceitos de computação gráfica é as transformações geométricas e sem ela praticamente não existiria a computação gráfica como conhecemos hoje.

Na geometria analítica uma Transformação Geométrica é uma aplicação bijectiva entre duas figuras geométricas, no mesmo plano ou em planos diferentes, de forma que, a partir de uma figura geométrica original, se forma outra geometricamente igual ou semelhante, sem perda das suas propriedades topológicas. Iezzi, G., Dolce, O. (1972)

Já na computação gráfica, para Conci, Azevedo e Leta (2008), as transformações geométricas são operações que levam o tom dos pixels na posição (x0, y0) da imagem origem para outra posição (xd, yd) do espaço em uma imagem destino, ou seja, modificam a posição dos pixels no espaço da imagem. (COSTA, 2018)

As transformações geométricas envolvem operações com vetores e matrizes, do tipo soma e multiplicação, além de conhecimentos básicos de álgebra e geometria. (CARVALHO, 2009)

Para fazer o braço robótico foi utilizado 3 conceitos de transformações geométricas, que são: transformação de escala, translação e rotação.

2.2 TRANSFORMAÇÕES DE ESCALA

Escalonar é uma Transformação Geométrica que significa mudar as dimensões de escala, ou seja, ampliar ou reduzir o tamanho da imagem. (REAL, 2017)

Segundo MANSSOUR e COHEN a operação de transformação escala consiste em multiplicar um valor de escala por todos os vértices do objeto (ou conjunto de objetos) que terá seu tamanho aumentado ou diminuído. Para aumentar o tamanho deve ser aplicado um fator de escala maior que 1.0 nos eixos. Para diminuir basta aplicar um valor entra 0.0 e 1.0. E caso for aplicado um fator de escala negativo irá resultar em um "espalhamento" do objeto no eixo aplicado. Forma matricial, onde as variáveis ex, ey, ez indicam os valores de escala dos eixos.

$$[x'y'z'1] = [xyz1] \begin{bmatrix} ex & 0 & 0 & 0 \\ 0 & ey & 0 & 0 \\ 0 & 0 & ez & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3 TRANSFORMAÇÕES DE TRANSLAÇÃO

A translação é o movimento que um objeto ou figura realiza de um ponto a outro. É o deslocamento paralelo, em linha reta na mesma direção e no mesmo sentido, de um objeto ou figura, em função de um vetor percorrendo a mesma distância. (REAL, 2017)

A operação consiste em adicionar constantes de deslocamento a todos os vértices, assim trocando o objeto ou cena de lugar. Na forma matricial as variáveis tx,ty,tz correspondem aos valores de translação que devem ser aplicados nos eixos. (MANSSOUR; COHEN, 2006)

$$[x'y'z'1] = [xyz1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix}$$

2.4 TRANSFORMAÇÕES DE ROTAÇÃO

Rotação é quando, ao girar um objeto, todos os seus pontos giram conforme um mesmo ângulo. (REAL, 2017)

A operação consiste em aplicar uma composição de cálculos utilizando o seno e cosseno do ângulo de rotação e todas as coordenadas dos vértices que compõem o objeto ou cena. Em 3D existe uma preocupação a mais que é definir qual o eixo se procedera a rotação. Na forma matricial o símbolo alpha representa o ângulo de rotação. (MANSSOUR; COHEN, 2006)

$$r_z = \begin{bmatrix} cos\alpha & sen\alpha & 0 & 0 \\ -sen\alpha & cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} r_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\alpha & sen\alpha & 0 \\ 0 & -sen\alpha & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} r_y = \begin{bmatrix} cos\alpha & 0 & sen\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -sen\alpha & 0 & cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.5 ILUMINAÇÃO

A iluminação na computação gráfica é essencial para a visualização dos objetos e dependendo da forma como é implementada pode fazer uma cena mudar totalmente. Por isso é importante saber qual tipo de iluminação é preciso para os requisitos da sua cena.

O modelo de iluminação é um modelo utilizado para calcular a intensidade de luz observada em um ponto na superfície de um objeto. São baseadas nas leis físicas que descrevem a intensidade luminosa em superfícies. (MONTENEGRO, 2019)

Dentre os modelos iluminação existentes foi utilizando o padrão do openGL que é a luz ambiente do modelo de Phong.

O Modelo de Iluminação de Phong, introduzido em 1975, permite calcular o valor da intensidade de um raio refletido por uma superfície em função da orientação da superfície, da posição da câmara, da posição da fonte de luz e das propriedades da superfície. [...] O modelo de iluminação de Phong considera que os materiais, quanto ao modo como refletem a energia luminosa, são uma combinação linear de um material que reflete toda a energia numa única direção. (GOMES, 2013)

A luz ambiente provê iluminação uniforme na cena. Essa iluminação pode ser alcançada por meio do uso de várias fontes de luz que espalham luminosidade em todas as direções. (GARCIA, 2009)

2.6 SOMBREAMENTO

O sombreamento ou modelo de tonalização utilizado foi o flat ou constante e serve para agiliza o cálculo da iluminação.

O modelo flat é realizado uma única vez por superfície, geralmente tomando-se um ponto no centro desta. Produzindo imagens com rapidez, porém sem muito realismo já que não existe variação de cor ao longo das superfícies. (MANSSOUR; COHEN, 2006)

Como ó braço robótico não possui diferente superfícies então não existia a necessidade de um método mais sofisticado para o sombreamento.

2.7 TEXTURA

Para dá um acabamento melhor para a cena e deixa minimamente mais realística, foi colocado uma textura no braço robótico utilizando uma foto de textura metálica, e para chão uma foto de textura de piso.

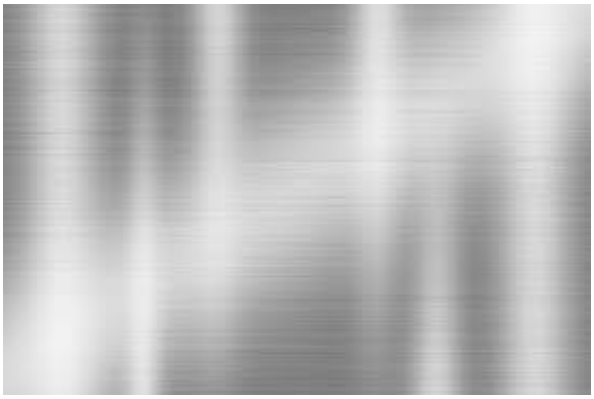

Figura 1: Textura metálica do braço

Figura 2: Textura de piso do chão

A ideia é utilizar uma imagem que contenha a aparência da superfície desejada, então durante o processo de masterização, mapeada sobre a superfície através de coordenadas de textura. A textura funciona como um decalque ou papel de parede sendo “colada” a superfície. (MANSSOUR; COHEN, 2006)

3 IMPLEMENTAÇÃO

3.1 Detalhes do Código

Inicialmente, para implementação do projeto final foi preciso instalar as dependências do Python necessárias para o projeto. Foram instaladas as seguintes bibliotecas: autopep8, pycodestyle, numpy e PyOpenGL. O autopep8 auxilia na formatação do código python de acordo com o guia de estilo do PEP8 (guia de estilo e utilização do código python). Ele utiliza o pycodestyle para determinar quais partes do código precisam ser formatadas. O autopep8 juntamente com o pycodestyle são capazes de corrigir a maioria dos problemas de formatação. Já a dependência numpy, é um pacote fundamental para utilização de arrays em python, com ela é possível ter acesso a um poderoso objeto de matriz N-dimensional, funções sofisticadas, ferramentas de integração com código C/C++, álgebra linear e Transformada de Fourier. Por último, mas não menos importante, temos a biblioteca PyOpenGL, que foi utilizada na versão 3.1.5, com ela podemos ter acesso aos comandos do OpenGL com o python, como o GL, GLU e GLUT.

Logo em seguida, foi dividido o projeto python em 4 arquivos: arm_gl, camera, *keyboard-controler* e *main*, dessa forma modularizamos o código utilizando os conceitos de *clean code* e **M-V-C**, afim de facilitar a leitura e escrita do projeto já que o mesmo ficará com muitas linhas de código e não seria o ideal deixar em um único arquivo python.

O arquivo **main.py** contém as chamadas de todos os arquivos.py supracitados e as importações das bibliotecas essenciais para o projeto. Primeiramente, foi definida a função reshape que será responsável por redimensionar/remodelar a janela atual de exibição na tela, passando como parâmetros a altura e largura da janela. Utilizamos a função glViewport para a transformação de viewport com parâmetros x e y iguais a 0 e 0, como valores iniciais, simbolizando que a janela inicial de visualização está no canto inferior esquerdo, e depois passamos os valores de dimensão para altura e largura padrão da janela. A função glMatrixMode especifica a pilha de matriz de destino para as operações de matrizes subsequentes, juntamente com o parâmetro GL_PROJECTION, para informar que as operações de matriz são sobre a pilha da matriz de projeção. Para a função glPerspective, define-se a perspectiva de projeção da matriz na tela, com o ângulo de visão, a proporção do campo de visão da largura dividida pela altura, o zNear correspondendo a distância do visualizador ao plano de corte mais próximo e o zFar correspondendo a distância do visualizador ao plano de corte mais distante. Novamente, chama-se a função glMatrixMode, agora com o parâmetro GL_MODELVIEW, para aplicar as operações subsequentes da pilha de matrizes MODELVIEW. Por último, com relação a função reshape temos, a função glutPostRedisplay que avisa ao OpenGL que será necessário ser exibida novamente, a janela atual. Continuando nosso projeto, definimos em seguida, uma função chamada opengl_init para preparar o ambiente para texturização, contendo a função glClearColor que vai servir para especificar as cores vermelho, verde, azul e o alfa (opacidade), e todos valores iniciais

são 0, porém, variam no intervalo de [0,1]. Logo após, temos a função glEnable que vai habilitar regursos do GL que quando utilizado com GL_DEPTH_TEST vai habilitar as comparações de profundidade e constantes atualizações do *buffer* de profundidade. Neste ponto, vamos habilitar a textura com as funções glEnable, com o parâmetro GL_BLEND que vai atualizar os valores nos *buffers* de cores e a função glBlendFunc que vai especificar a forma de cálculo do pixel da janela ao inicializar, passando dois parâmetros, um para especificar como os fatores de mesclagem das cores vão ser calculados nas renderizações e o outro para especificar como os fatores de combinação das cores vão ser calculados. Agora, vamos utilizar a função glShadeModel para escolher um sombreamento, utilizamos com ela o parâmetro GL_FLAT para um sombreamento mais uniforme, por fim, mais uma vez a função gl_Enable, agora passando o parâmetro GL_TEXTURE-2D para habilitar a texturização bidimensional. Agora, definiremos mais uma função, chamada de window_init, para renderização na janela inicial do OpenGL. De início, implementamos a função glutInitWindowPosition para definir a posição inicial da janela. Em seguida, vamos inicializar a biblioteca GLUT com a função glutInit para negociar uma sessão com o sistema de janelas. Com a função glutInitDisplayMode, tivemos como definir o modo de exibição inicial da janela, então, os parâmetros iniciais foram, GLUT_RGBA para colocar uma máscara de bits, para selecionar a janela no modo RGBA. GLUT_DEPTH para adicionar uma máscara de bits para selecionar uma janela com buffer de profundidade. Depois, GLUT_DOUBLE para máscara de bits para selecionar uma janela com *buffer* duplo, e por último, GLUT_ALPHA para colocar uma máscara de bits para selecionar uma janela com um componente alfa para o *buffer* de cor. Tivemos que criar também, uma função para definir o tamanho da janela, que foi a glutInitWindowSize passando o tamanho de largura e altura de 800x600. Foi criado, também, uma janela mais externa com o nome "Braço Garra"com a função glutCreateWindow, e por fim, utilizamos a função glutPositionWindow passando os parâmetros de 400x400 para ajustar a posição inicial da janela. A função main foi implementada inicialmente, carregando as funções window-init, opengl_init e load_textures, nessa última, carrega-se duas imagens para ajudar na simulação da textura. A função display vai renderizar a imagem, criando o ambiente para mostrar o braço robótico na cena em cima de uma mesa quadrada. Para organizar os valores de redistribuição de tamanho de imagem, ou qualquer alteração no braço robótico na cena, inserimos as funções de acionamento de teclas que estão no arquivo keyboard_controler.py, com uma função do programa de gerenciamento de janelas, chamada de glutPostRedisplay junto com as funções de glutReshapeFunc e a função que vai prender tudo isso num loop para que possamos redimensionarmos quantas vezes quisermos, chamada de glutMainLoop. Por fim, para que nós pudéssemos desenhar os objetos na cena juntamente com as texturas, criamos mais 3 arquivos python, **arm_model.py, cube_model.py e floor_model.py**, em que neles se encontram as formar geométricas com texturas do chão e do braço robótico com a garra, utilizando os conceitos e comandos para rasterização, coordenadas para vértices e texturas, translação, angulação, cor e escala.

Já o arquivo **arm_gl.py** é responsável por armazenar todas as classes e funções relativas as chamadas do braço robótico com a garra. Inicialmente, definimos uma função make_arm que é responsável por construir o braço, em seguida utilizamos as funções glGenlist passando como parâmetro o número 1 como range da quantidade de listas a serem geradas para exibição e a glNewList para criar

essa lista de exibição, passando como parâmetros o nome especificado para a lista de exibição criada e o mode de compilação dessa lista, nesse caso foi usado o GL_COMPILE, e dessa forma iniciamos a composição para a geração inicial do braço. Para o próximo passo, utilizamos a função glRotatef para gerar a origem posicionada do braço, e já iniciamos a função passando as posições de origem da base e do ombro, e para a angulação, passamos a chamada a classe arm_angles. Para a origem posicionada do braço no centro da imagem, utilizamos as funções glTranslatef para multiplicar a matriz atual da imagem em uma determinada translação e já utilizamos, juntamente, a função PushMatrix para colocar a matriz definida no topo da pilha, logo após, utilizamos a glscalef para multiplicar a matriz atual na pilha por uma determinada sequência de parâmetros formando a escala da matriz, em sequência colocamos uma cor na matriz com a função glColor3f, definimos um formato geométrico de cubo para representá-lo com a função glutSolidCube e para finalizar o braço utilizamos a função glPopMatrix para pegar e utilizar a matriz que está no topo da pilha. Continuando o projeto, temos as partes do cotovelo, dedo1 e dedo2, representando as garras. Criamos o cotovelo já posicionado no braço como origem, assim como as garras (dedos 1 e 2) - a maioria das funções já foram descritas e explicadas com os parâmetros, mostrando a criação e posicionamento - para isso foi definido mais um conjunto de funções, com as funções glTranslatef, glRotatef (para as partes do braço robótico que giram ou fazem alguma angulação como a garra, há a dependência da classe arm_angles), em seguida, as funções glPushMatrix, depois, glScalef, glColor3f, a função de criação da geometria do cubo que é a glutSolidCube e para colocar em execução na pilha de procedimentos a função glPopMatrix. E por fim, para terminar a composição do braço, glEndList.

O arquivo **camera.py** contém, a classe Camera, com os atributos: obs, que representa a posição inicial do observador (câmera) e o look, que é para identificar o centro de para onde a câmera vai olhar, ou seja, o centro da cena. Para calcular a posição do observador (câmera), criamos uma função chamada look_at, o observador, sendo a câmera terá o movimento circular em torno do objeto, no caso o braço robótico com a garra. A função gluLookAt vai retornar uma matriz de visualização derivada de um ponto do observador (câmera), um ponto de referência indicando o centro da cena e um vetor apontando para cima da janela de visualização. Essa matriz mapeia o ponto de referência pra o eixo z e ponto do observador (câmera) olhando para a origem. Quando uma matriz de projeção é usada, o centro da cena é mapeado para o centro da viewport. Da mesma forma, a direção descrita pelo vetor cima projetado no plano de visualização é mapeada para o eixo y positivo para que aponte para cima na janela de visualização. O vetor UP não deve ser paralelo à linha de visão do ponto do olho até o ponto de referência.

Por conseguinte, o arquivo **keyboard_controle.py** é responsável pela ação do toque nas teclas correspondendo alguma ação, tanto com relação ao observador, nesse caso a câmera, ou o braço robótico com a garra. Criamos a função key_press com retorno booleano para fazer os cálculos relativos a ação que deverá ser tomada. Nesta função, ao acionar determinadas teclas, é possível fazer movimentos, sendo que os dedos 1 e 2 (garra) fazem os movimentos contrários, o que o dedo 1 faz para cima, o dedo 2 faz para baixo, para simular a ação de uma pinça. Esses movimentos são:

- Caso "A": rotacionar o ombro em +5º.

- Caso "a": rotacionar o ombro em -5º.
- Caso "S": rotacionar o cotovelo em +5º.
- Caso "s": rotacionar o cotovelo em -5º.
- Caso "B": rotacionar a base do braço em +5º.
- Caso "b": rotacionar a base do braço em -5º.
- Caso "c": fechar garra em -5º.
- Caso "C": abrir garra em +5º.

Também implementamos a função python chamada de special_key_press que assim, como a key_press que comanda as rotações do observador (câmera) e retorna um booleano quando acionada alguma tecla da função GLUT_KEY, função nativa para controle de teclado (setas), podendo ser, GLUT_KEY_UP para cima, GLUT_KEY_DOWN para baixo, GLUT_KEY_LEFT para esquerda e GLUT_KEY_RIGHT para direita.

3.2 Pipeline e o que cada etapa faz

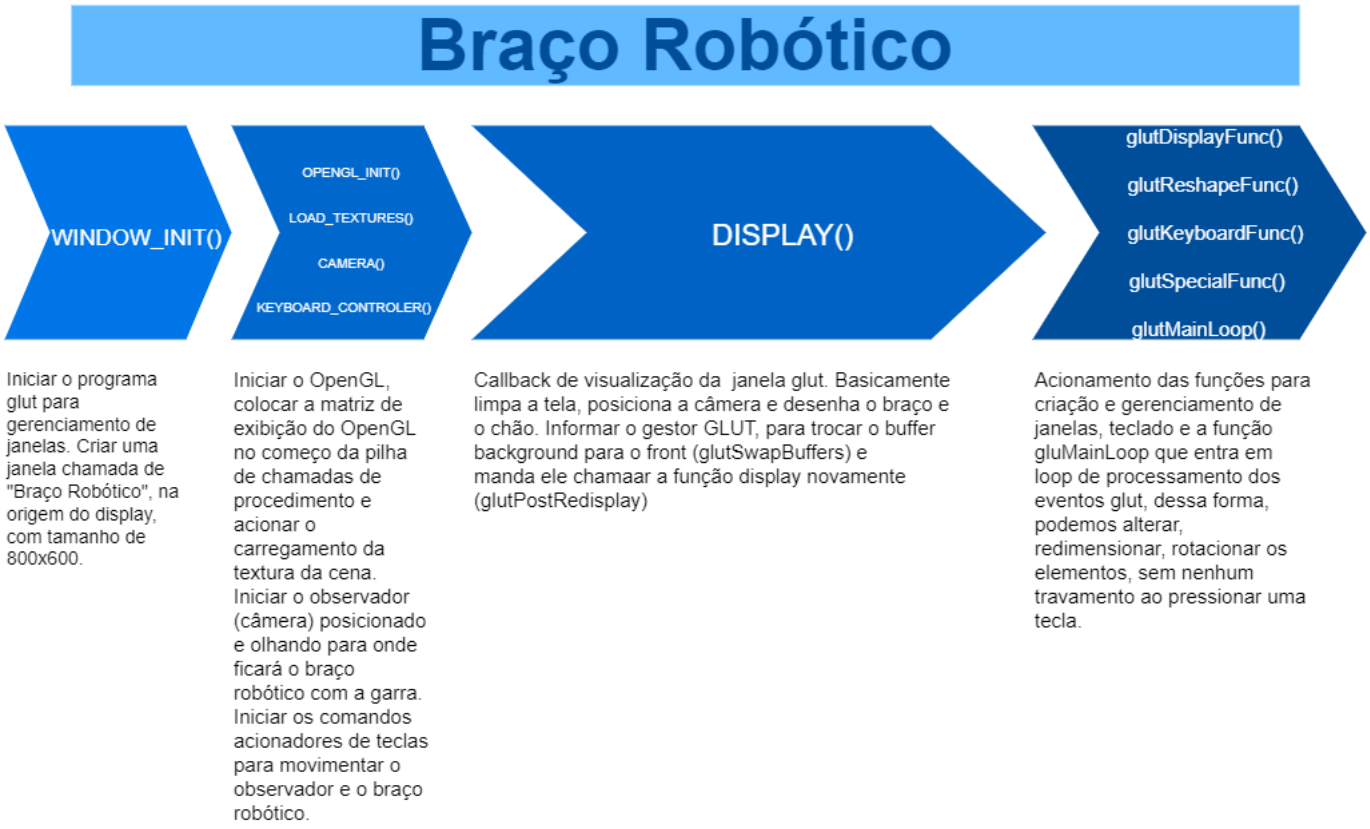

Figura 3: Pipeline de criação do braço robótico e cena

4 CONSIDERAÇÕES FINAIS

Tendo em vista os aspectos observados conclui-se que os objetivos do trabalho foram alcançados com pequenas ressalvas, a fim de tornar o projeto mais interativo com o ambiente, porém nada que comprometa o desempenho do trabalho apresentado.

4.1 Principais problemas encontrados

Durante o projeto houveram problemas com a textura do cubo, que representa as partes do braço e da garra. A situação era que a textura se movia conforme a câmera era rotacionada, assim tendo um comportamento inesperado. A solução encontrada foi modificar a biblioteca que fazia o cubo e implementar a textura na hora da sua criação.

Com o decorrer do projeto, conseguimos perceber as diversas etapas práticas para construção do algoritmo em Python com o OpenGL. Coletamos algumas informações de forma geral, para agregar valor e conhecimento para as próximas turmas no tocante a utilização dos materiais de forma mais prática, porém, cremos que isso deve-se pelo fato do formato de aulas em virtude da pandemia do COVID-19. Listamos algumas considerações que devem ser levadas de forma construtiva, foram elas:

- Nem todos os colegas tinham familiaridade com linux para facilitar a utilização do OpenGL em C e por conta disso fizemos em Python, os nomes das funções são as mesmas que em C então uma sugestão seria adotar Python ao invés de C, apesar de C ser uma linguagem de alto nível na prática ela não possui um gerenciador de pacotes padrão e agnóstico de sistema Operacional.
- O OpenGL foi um linguagem nova, apesar de ser programação, a forma de trabalhar com imagens e matrizes tem sutilezas, e para a primeira vez vendo tornou o trabalho um pouco pesado nesse sentido. Uma sugestão seria ter dividido o projeto em etapas e ir realizando a medida que a disciplina ia ocorrendo.

4.2 O que pode ser melhorado (e como melhorar)

Muitos dos principais problemas encontrados supracitados serão resolvidos quando houver a possibilidade das aulas presenciais, pois torna o contato e a resposta com o professor mais rápido sobre dúvidas, pois todos os alunos estarão fazendo os exercícios práticos ao mesmo tempo e isso trará equalização nas informações.

Outro tópico bem importante é fazer mais exercícios para se ambientar melhor ao OpenGL e as tratativas com as particularidades com os parâmetros, pois deu bastante trabalho.

4.3 Link do repositório

Disponível em: `https://github.com/samuel-cavalcanti/projeto_final_computacao_grafica_ufrn`

5 CONTRIBUIÇÕES

5.1 Contribuição de cada integrante do grupo

- Frankson: Arquivos arm_angles.py, camera.py e keyboard_controler.py do projeto. Embasamento teórico do relatório.
- Júlia: Arquivo main.py (fuções: main, display, normal_keys_handler e special_keys_handler) do projeto. Resumo e Introdução do relatório.
- Samuel: Arquivo main.py (funções: reshape, opengl_init, window_init e load _texture) do projeto. Resumo e considerações finais do relatório.
- Tiago: Arquivos arm_model.py, cube_model.py e floor_mode.py do projeto. Implementação do relatório.

Referências

[1] Repositório do projeto final da disciplina de Computação Gráfica.

[2] Autopep8. Disponível em: <https://pypi.org/project/autopep8/>

[3] PEP8. Disponível em: <https://www.python.org/dev/peps/pep-0008/#introduction>

[4] Numpy. Disponível em: <https://pypi.org/project/numpy/>

[5] PyOpenGL. Disponível em: <https://pypi.org/project/PyOpenGL/>

[6] GLUT and OpenGL. Disponível em: <https://www.opengl.org/resources/libraries/>

[7] Fundamentals of Computer Graphics. Disponível em: <https://math.hws.edu/bridgeman/courses/324/s06/doc/opengl.html>

[8] OpenGL 2.1, GLX, and GLU Reference Pages. Disponível em: <https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/>

[9] Iezzi, G., Dolce, O., Geometria Analítica, Editora Moderna SP, 1972.

[10] Melo, Helena S. (2010). "Isometrias noPlano: Uma abordagem segundo a Geometria Analítica", Influir, Ponta Delgada, Açores, 121 pp. ISBN 978-989-97107-0-2. Disponível em: <https://repositorio.uac.pt/bitstream/10400.3/2513/1/Isometrias%20no%20plano_%20uma%20abordagem%20segundo%20a%20geometria%20analitica.pdf>

[11] COSTA, Jackson. TRANSFORMAÇÕES GEOMÉTRICAS BIDIMENSIONAIS COM APLICAÇÕES. 2018. 83 f. TCC (Graduação) - Curso de Licenciatura em Matemática, Universidade Federal de São Carlos, Sorocaba, 2018. Disponível em: <https://www.matematicasorocaba.ufscar.br/arquivos/tcc-jackson-costa.pdf>

[12] CONCI, A.; AZEVEDO, E.; LETA, F.R. Computação Gráfica: Teoria e Prática. 2. ed. Rio de Janeiro: Elsevier, 2008. 404 p. v. 2.

[13] CARVALHO, Marco Antonio Garcia de. COMPUTAÇÃO GRÁFICA: transformações geométricas. São Paulo, 2009. Color. Disponível em: <https://www.ft.unicamp.br/~magic/ST765/CG2009_Transformacoes.pdf>

[14] REAL, Luana Pereira Villa. TRANSFORMAÇÕES GEOMÉTRICAS: APLICAÇÃO DE MATRIZES NA COMPUTAÇÃO GRÁFICA. 2017. 244 f. Dissertação (Mestrado) - Curso de Pós- Graduação em Ensino de Ciências e Matemática, Centro Universitário Franciscano, Santa Maria/Rs, 2017. Disponível em: <http://www.tede.universidadefranciscana.edu.br:8080/bitstream/UFN-BDTD/599/5/Dissertacao_LuanaPereiraVillaReal.pdf>

[15] MANSSOUR, Isabel Harb; COHEN, Marcelo. Introdução à Computação Gráfica. Porto Alegre, 2006. Disponível em: <https://www.inf.pucrs.br/manssour/Publicacoes/TutorialSib2006.pdf>

[16] MONTENEGRO, Anselmo. Computação Gráfica: iluminação. Rio de Janeiro: Uff, 2019. Color. Disponível em: <http://www.ic.uff.br/~anselmo/cursos/CGI/slidesNovos/iluminacao.pdF>

[17] GOMES, Mário Rui. Iluminação e Sombreamento. Lisboa: Instituto Superior Técnico, 2013. Disponível em: <http://disciplinas.ist.utl.pt/leic-cg/textos/livro/Shading.pdf>

[18] GARCIA, Marco Antonio. COMPUTAÇÃO GRÁFICA: iluminação. São Paulo: Unicamp, 2009. Color. Disponível em: <https://www.ft.unicamp.br/~magic/ST765/CG2009_iluminacao.pdf>