DESENVOLVIMENTO DE BLOCOS EM OPENGL PARA A FERRAMENTA MOSAICODE

André Lucas Nascimento Gomes, graduando em Ciência da Computação
Flávio Luiz Schiavoni, Departamento de Ciência da Computação

RESUMO

Neste artigo é apresentada o processo de desenvolvimento de uma extensão para a Ferramenta de Programação Visual Mosaicode feita a partir da biblioteca de Síntese de Imagens utilizando a biblioteca OpenGL em C. A metodologia para a elaboração se inicia com o estudo da API seguido da criação de telas com exemplos de funcionalidades disponíveis. Cada unidade atômica foi encapsulada em um bloco, representando uma ação, e interligado a outros a partir de conexões, que realizam alguma conversão de valores ou definem o fluxo de execução da aplicação para gerar um código no final do processo, a partir de um padrão de código.

INTRODUÇÃO

A Computação Gráfica é um domínio tem uma relação estreita com a matemática e as artes. Ela busca simular conceitos artísticos, dando ao usuário poder de gerar imagens sequer imaginadas. A Síntese de imagens é uma das grandes áreas da Computação Gráfica, sendo que seu objetivo é criar representações visuais de objetos utilizando computadores [3]. Uma das bibliotecas que realizam isto é o OpenGL [6], que é open-source e possui implementações disponíveis para as linguagens de programação C e C++. Ela é utilizada para criação de programas interativos gráficos e modelagens 2D e 3D, disponível em diversas plataformas e otimizada de maneira que seja possível sua utilização desde em computadores pessoais a máquinas mais potentes [2].

A Arte digital é uma das áreas onde a síntese de imagens é amplamente utilizada, seja para criação artística visual, seja para a criação de instalações e ambiente imersivos de Realidade Aumentada ou Virtual. Neste contexto, apresentamos o Mosaicode [5], um ambiente de programação visual desenvolvido na UFSJ pelo Laboratório de pesquisa ALICE (Arts Lab in Interfaces, Computers, and Education), que se propõe como uma ferramenta para criação de aplicações de Arte Digital e Realidade Virtual que ao finalizar o processo, gera o código do Software.

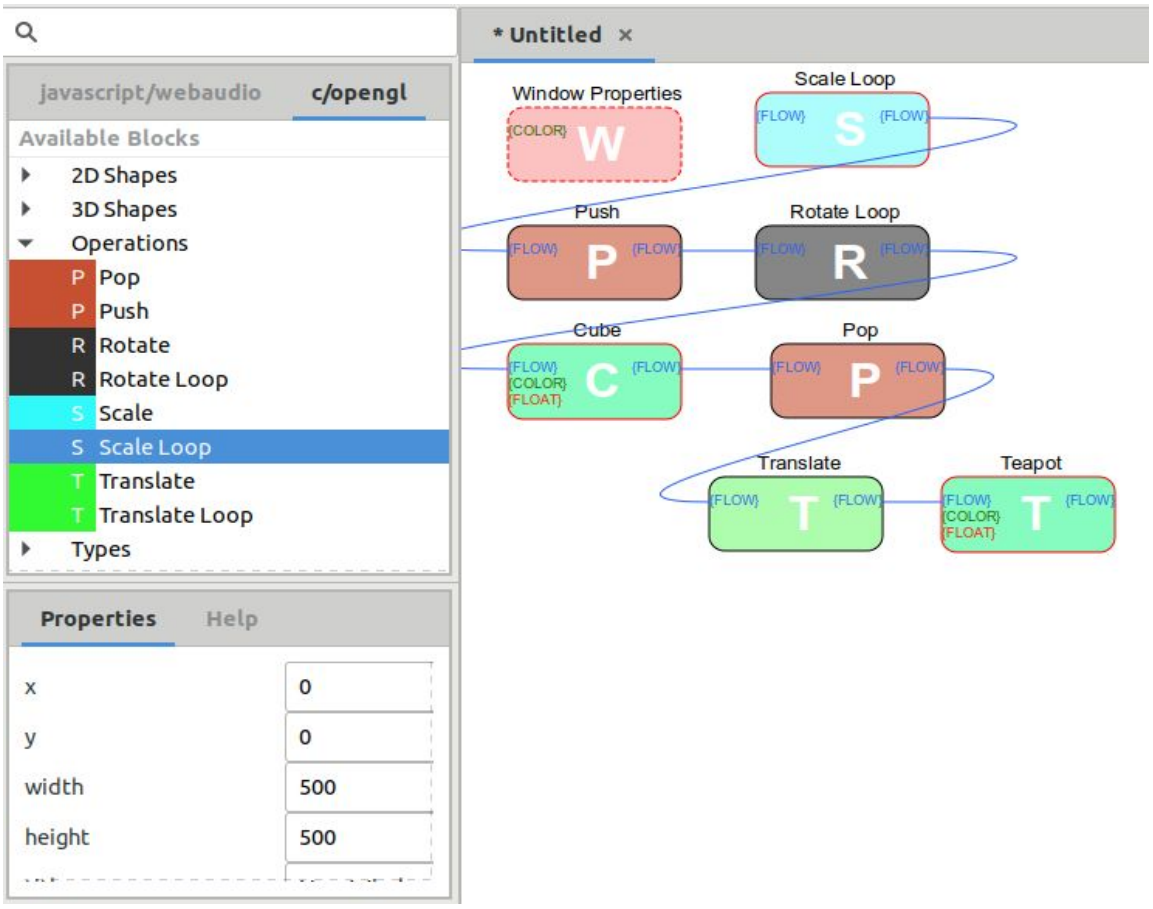

Figura 1 - Ambiente de Programação Visual Mosaicode.

A geração de código no Mosaicode se dá por meio da criação de diagramas, que são compostos por blocos e suas propriedades. Blocos são unidades atômicas, cuja função é executar uma funcionalidade básica no código. Para modificar características dos blocos, são utilizadas propriedades, que possuem dois tipos: dinâmicas e estáticas. As propriedades estáticas são integradas aos blocos e podem ser alteradas somente antes da execução do código, no momento da escrita. As propriedades dinâmicas são chamadas também de conexões, e interligam um bloco a outro, assim, podendo ou não repassar informações entre eles e definir seus valores no momento da execução. As conexões de saída o repassam dados de um bloco para conexões de entrada de outro. Estruturalmente, os blocos são compostos de trechos de código, que são inseridas em locais especificados no padrão de código. Um padrão consiste de partes de algoritmos que costumam ser sempre utilizados em cada um dos programas que utilizam a API selecionada, além de pedaços referentes ao bloco inserido no diagrama. Um conjunto de blocos, conexões, padrão de código e propriedades são chamados extensões, que são uma abstração de um domínio, biblioteca ou API.

Neste trabalho, utilizando a biblioteca OpenGL na linguagem C, foi criada uma extensão da ferramenta Mosaicode para o domínio da síntese de imagens. Na seção **Desenvolvimento**, tem a explicação de como foi o processo para o aprendizado da biblioteca. Na seção **Resultados**, será documentado como foi realizada a implementação da extensão, descrevendo o padrão de código, os blocos e as conexões criados. Por fim, na seção **Conclusão**, será exposto possibilidades de utilização da extensão gerada além de propor melhorias nela.

DESENVOLVIMENTO

O projeto de Iniciação Científica tinha como etapas para criação da extensão o Estudo das funcionalidades do OpenGL, criação de blocos, integração deles no Mosaicode, criação de exemplos e documentação.

Na primeira parte, foi realizado um estudo da API OpenGL baseando-se no livro "OpenGL - Uma abordagem prática e objetiva" [2] e também no conteúdo das aulas de Computação Gráfica, disciplina obrigatória do curso de Ciência da Computação na UFSJ, que utilizou o OpenGL como ferramenta de apoio ao ensino neste semestre. Nos primeiros protótipos gerados, foram criados elementos básicos que são utilizados em cada programa que utiliza a API, como gerenciamento de janelas, funções que iniciam a biblioteca e tratamento de eventos externos, como mouse e teclado.

Logo depois foi realizado o aprendizado de como fazer os primeiros desenhos utilizando a API. Inicialmente, utilizando ferramentas 2D, foi praticado como esboçar pontos e linhas, assim sendo possível esboçar qualquer polígono, além dos triângulos e quadrados, modelos que são nativos da API. Após isto, foi estudado modelos 3D, utilizando primitivas gráficas da biblioteca, como esfera, cone, torus e teapot.

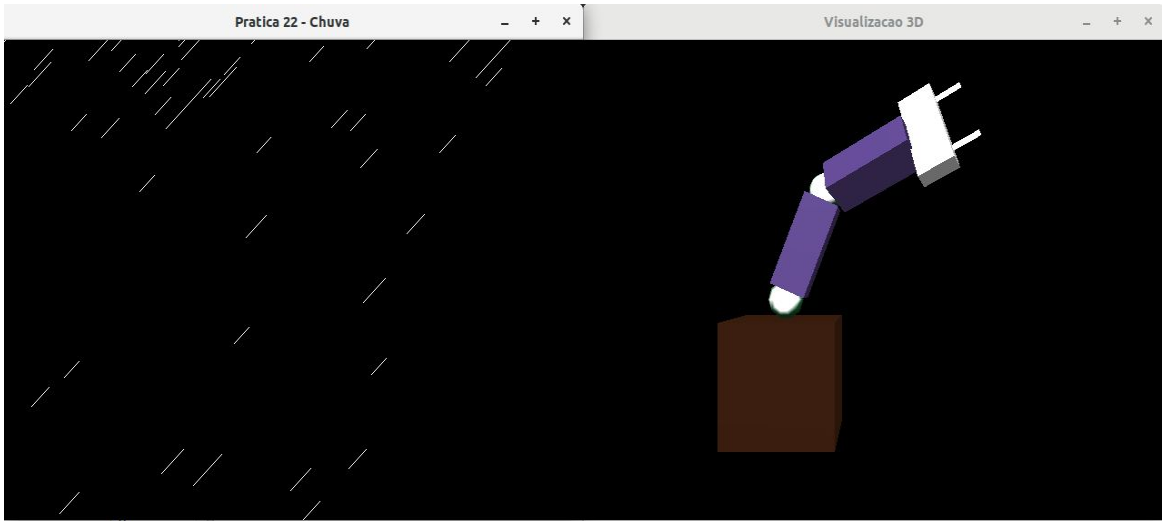

Figura 2 - Dois exemplos gerados: O da esquerda é uma simulação de chuva e o da direita é movimentação de um braço mecânico iluminado

Com a criação destes desenhos, o próximo passo a ser efetuado foi realizar transformações geométricas nestes modelos. Utilizando a translação, escala e rotação, foi feito protótipos misturando elementos gráficos e colocando hierarquia nas transformações, o que possibilitou a alteração de um objeto sem ter que modificar a cena inteira. Por fim, foi estudado o acréscimo de algumas funcionalidades, como iluminação, mudança de perspectivas e uma introdução no estudo de partículas utilizando OpenGL. Os exemplos citados acima estão disponíveis em: **https://github.com/andgomes95/Exemplos-OpenGL.**

RESULTADOS

Com estes estudos preliminares, foi criada uma base para iniciar a construção da extensão e todos seus elementos, como blocos, conexões e padrão de código.

Os blocos são divididos em 5 categorias: Types, Window, 2D Shapes, 3D Shapes e Operations. Em **Types**, estão todos os tipos que podem ser utilizados, como Color e Float. Com eles, é possível inserir valores dos tipos específicos dos tipos selecionados. Em **Window**, possui blocos em que seja possível editar propriedades da cena, como tamanho, posição inicial dela na tela e título.

2D Shapes é a categoria onde os objetos 2D se encontram. Além do triângulo e o quadrilátero, formas nativas da biblioteca, foram adicionados o círculo e a elipse, por serem formas padrões e conhecidas. Na classe **3D Shapes**, possui todas as primitivas gráficas 3D disponíveis, como Cubo, Esfera e Cone. Por fim, em **Operations**, possuem as operações de hierarquia Push e Pop, além de possuir blocos lineares e em laço para as transformações geométricas Translação, Escala e Rotação.

Além dos blocos, foram criadas também conexões que relacionam com as possíveis entradas ou saídas que cada um destes elementos. As conexões disponíveis na extensão possuem 3 tipos: Float, Color e Flow. O **Float** transfere um valor em ponto flutuante de uma porta a outra. A conexão **Color** realiza a mesma operação, porém utiliza valores em RGBA e são utilizadas para alterações em cores de fundo ou de objetos, por exemplo.

Já a categoria **Flow** foi criada pela necessidade de inserir uma ordem de precedência, para definir qual comando deve executar primeiro, e onde e até quando as transformações hierárquicas irão atuar durante a execução da aplicação.

O padrão de código é iniciado com a inserção da biblioteca básica do opengl (GL/gl.h), da que possui elementos para auxiliar na criação de interfaces gráficas (GL/glut.h) e que encapsula em funções operações de baixo nível (GL/glu.h) [2], além das padrões da linguagem C, "stdio.h", "string.h" e "math.h". Logo após é inserido, se existente, o trecho de código com o rótulo **global**, que acrescenta bibliotecas extras

necessárias para o funcionamento do bloco, ou variáveis globais para inicializar a aplicação.

Na sequência, é declarada uma *struct* com parâmetros de inicialização da janela principal do software gerado, como posição, título e tamanho da tela e cor sólida do plano de fundo. Seguido disso, é explícito uma função que inicializa a janela com os dados inseridos salvos na estrutura anterior.

```
#include <GL/glut.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <math.h>
#define ESCAPE 27
int window;
//Global
$code[global]$
typedef struct mosaicgraph_window{
    float x;
    float y;
    float width;
    float height;
    float red;
    float green;
    float blue;
    float alpha;
    float fullscreen;
    char title[128];
    int id;
    void (*process)(void *self);
}mosaicgraph_window_t;
mosaicgraph_window_t * mosaicgraph_create_window(float width, float height,float x, float y){
    mosaicgraph_window_t * window = (mosaicgraph_window_t *) malloc(sizeof(mosaicgraph_window_t));
    window->fullscreen = 0;
    window->x = x;
    window->y = y;
    window->width = width;
    window->height = height;
    return window;
}
int mosaicgraph_draw_window(mosaicgraph_window_t * window){
```

Figura 3 – Padrão de Código: Chamadas de biblioteca e struct referente a janela.

Funções específicas de cada bloco são incluídos na continuidade, por um pedaço de código com rótulo **function**. Vale ressaltar que este pedaço só é adicionado uma única vez, independente da quantidade blocos iguais existentes no diagrama. Posteriormente, é inserido a função **display**, onde estão reunidos todos elementos que serão desenhados

na janela. Dentro dela possui o trecho de código etiquetado como **call**, onde são chamadas as funções inseridas anteriormente na etiqueta function, ordenados de acordo com o fluxo especificado pelas conexões. A seguir vem o procedimento **idle**, em que chama um pedaço de código com rótulo **idle**, e após isto, ele executa a função display novamente.

```
    glutInitWindowPosition(window->x, window->y);
    glutInitWindowSize(window->width, window->height);
    glClearColor(window->red, window->green, window->blue, window->alpha);
    glClear(GL_COLOR_BUFFER_BIT);
    window->id = glutCreateWindow(window->title);
    if (window->fullscreen){
        glutFullScreen();
    }
    return window->id;
}
//Function
$single_code[function]$

void display(){
    glMatrixMode(GL_MODELVIEW);
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    //Call
    $code[call]$
    glutSwapBuffers();
}
void idle(){
    //Idle
    $code[idle]$
    display();
}
```

Figura 4 - Padrão de Código: Funções Display e Idle.

Por fim, a função principal é declarada, começando por funções de inicialização nativas das bibliotecas OpenGL, seguidos do trecho de código com etiqueta **declaration**, onde são adicionadas declarações que devem ser executadas antes da declaração da tela, que é o próximo passo do fluxo de execução. Em seguida é incluído o trecho de código com rótulo **execution**, onde define as ações a serem executadas antes da geração da cena, e também é onde são inseridos códigos referentes às conexões. Enfim é desenhada a janela com o conteúdo do procedimento display, que entra em laço alterando valores especificados na função idle.

Imagem padrão de código

A extensão gerada com todos os elementos especificados acima se encontra disponível em: **https://github.com/andgomes95/mosaicode-c-opengl.**

CONCLUSÃO

Este artigo apresentou o processo de aprendizado e desenvolvimento da extensão em C/OpenGL para o Mosaicode. Este processo permitiu com que se percebesse padrões de repetição nos códigos que utilizam a API OpenGL, localizando onde deve ser inserido trechos e simplificou a criação dos primeiros blocos. Com os blocos disponíveis na ferramenta, é possível introduzir o ensino da biblioteca, bem como criar protótipos de forma rápida a modelagem de objetos utilizando formas primitivas disponíveis.

Futuramente, será necessário expandir a quantidade de blocos, adicionando elementos como posicionamento da câmera e iluminação como alternativa. Além disto, há o desejo de desenvolver blocos que envolvam conceitos matemáticos, como Diagrama de Voronoi [1] e Fractais [4], para expandir o leque de possibilidade de telas complexas que possam ser construídas utilizando apenas o Mosaicode.

Outra possibilidade seria testar a usabilidade da extensão com alunos da disciplina de Computação Gráfica, que são usuários que sabem programar porém não possuem conhecimento na biblioteca, ou com o Grupo de Estudos de Arte Digital, para avaliar se há a possibilidade de criação artística com estes elementos disponíveis atualmente na ferramenta.

AGRADECIMENTOS

O autor agradece a equipe de desenvolvimento da ferramenta Mosaicode, ao Grupo de Pesquisa ALICE e o Grupo de Estudos de Arte Digital da Universidade Federal de São João Del Rei pelo apoio e suporte criativo para o desenvolvimento desta biblioteca. Também agradece a FAPEMIG, CNPq e da PROAE/PROPE/UFSJ pelo apoio financeiro institucional.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] L. P. Chew and R. L. S. Dyrsdale III. Voronoi diagrams based on convex distance functions. In Proceedings of the first annual symposium on Computational geometry pages 235–244. ACM, 1985.

[2] M. Cohen and I. H. Manssour. OpenGL: uma abordagem prática e objetiva. Novatec editora, 2006.

[3] A. Conci and E. Azevedo. Computação gráfica: geração de imagens. Editora Campus, 2003.

[4] C. Redies, J. Hasenstein, and J. Denzler. Fractal-like image statistics in visual art: similarity to natural scenes. Spatial Vision, 21(1):137–148, 2007.

[5] F. L. Schiavoni and L. L. Gonçalves. From virtual reality to digital arts with mosaicode. In 2017 19th Symposium on Virtual and Augmented Reality (SVR), pages 200–206, Curitiba - PR - Brazil, Nov 2017.

[6] M. Woo, J. Neider, T. Davis, and D. Shreiner. OpenGL programming guide: the official guide to learning OpenGL, version 1.2. Addison-Wesley Longman Publishing Co., Inc., 1999.