Desenvolvimento de Extensões de Processamento e Síntese de Imagens para a ferramenta Mosaicode

André Gomes
Departamento de Computação
Universidade Federal
de São João del-Rei
São João del-Rei, Minas Gerais
Email: andgomes95@gmail.com

Frederico Ribeiro
Departamento de Computação
Universidade Federal
de São João del-Rei
São João del-Rei, Minas Gerais
Email: fredribeiro97@gmail.com

Flávio Schiavoni
Departamento de Computação
Universidade Federal
de São João del-Rei
São João del-Rei, Minas Gerais
Email: fls@ufsj.edu.br

***Abstract*—The development of specific domain applications can be assisted by programming environments. Among these tools there is Mosaicode, a programming environment which uses visual programming to construct diagrams and in the end, generate code. In this work, you'll propose the process of creating extensions to OpenCV and OpenGL libraries and how are the extensions created for these libraries.**

***Resumo*—O desenvolvimento de aplicações de domínios específicos pode ser auxiliado por ambientes de programação. Dentre estas ferramentas existe o Mosaicode, que utiliza programação visual para construção de diagramas e no final, gerar código. Neste trabalho, será apresentado o processo de criação de extensões para as bibliotecas OpenCV e OpenGL e como estão as extensões criadas.**

I. Introdução

Atualmente muitas ferramentas e tecnologias existem para facilitar a aprendizagem e o desenvolvimento de aplicações específicas. Mais precisamente, estamos abordando sobre áreas de domínio particular, como Visão Computacional, Computação Gráfica ou Processamento Digital de Imagens. Todas estas áreas possuem APIs e bibliotecas para programação facilitada, possibilitando que pessoas com conhecimento da área e de codificação consigam desenvolver suas aplicações com certa facilidade.

Neste ponto podemos perceber que pessoas iniciantes em programação não conseguiriam desenvolver programas para estes domínios sem auxílio ou sem realizar um curso específico para tal. A partir disto, apresentamos o ambiente de programação visual Mosaicode [1], uma ferramenta que permite a criação de aplicações para estes domínios através do ato de arrastar e conectar caixas para geração de código.

O Mosaicode permite que o usuário construa suas aplicações possuindo conhecimento específico sobre o domínio que está a utilizar devido a possibilidade de exploração da intuitividade de programação visual focada em domínios específicos. Para tal, o ambiente separa os diferentes domínios em extensões, que são criadas através de APIs ou bibliotecas particulares da área tratada.

Atualmente, o ambiente Mosaicode possui diversas extensões desenvolvidas ou em desenvolvimento para alguns domínios de aplicação, entretanto, neste trabalho, vamos abordar especificamente duas: a que trata de Computação Gráfica com a biblioteca OpenGL [2] e a de Visão Computacional e Processamento Digital de Imagens com OpenCV [3]. Também será retratado todo o processo para construção de uma extensão.

II. Mosaicode

O Mosaicode é um ambiente de programação Visual desenvolvido na Universidade Federal de São João del-Rei pelo ALICE (Arts Lab in Interface, Computers, and Else). Ele é uma ferramenta open-source que permite a geração e execução de código através do arrastar e soltar e conectar Blocos com pequenas funções para a construção de uma aplicação. O Mosaicode permite uma programação baseada na arquitetura *pipes and filters* [4], onde blocos de processamento são interligados uns nos outros por meio de Conexões, criando um fluxo de dados para os mesmos. Um conjunto de caixas conectadas gera uma aplicação, chamada de diagrama, como podemos observar na Figura 1.

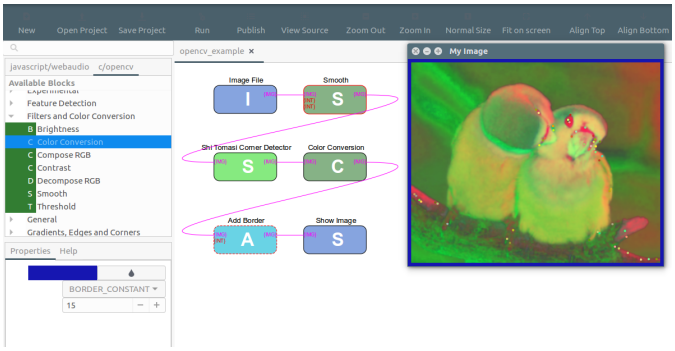


Figura 1. O ambiente Mosaicode, apresentando um diagrama elaborado na extensão OpenCV e sua execução ao lado.

A ferramenta permite que o código resultante possa ser conferido, editado e executado, possibilitando várias experimentações durante o processo de desenvolvimento. O algoritmo gerado é criado em cima de um **padrão de código**, onde são especificados partes onde o código costuma ser repetitivo nas diversas aplicações desenvolvidas sobre uma mesma API, como inicializações das funções, e seções com

rótulos, onde códigos referentes podem ser eventualmente inseridos.

A unidade mais básica da programação no Mosaicode são os **Blocos**, que representam uma funcionalidade específica e mínima dentro de um domínio específico. Uma extensão possui um conjunto de Blocos de diversas funções específicas dentro do domínio em questão. O comportamento da funcionalidade de um Bloco pode ser modificada a partir do conjunto de **propriedades estáticas e dinâmicas**, se for cabível ao mesmo possuí-las. As propriedades estáticas podem ser definidas em tempo de programação, quando um determinado bloco for selecionado no diagrama, abrindo uma janela com todos os seus atributos e permitindo ao usuário alterar estes dados através da interface gráfica do ambiente. Os blocos podem possuir também um conjunto de portas, de entrada ou saída, que permitem enviar ou receber dados de processamento de outros Blocos em que estejam conectados em tempo de execução. Estas portas definem as propriedades dinâmicas, que interligam portas de saída de um bloco com portas de entrada de outro, transmitindo dados em tempo de execução. A relação entre portas de entrada e de saída são chamadas Conexões. Conexões podem propagar diferentes tipos de informações, entretanto, deverá haver sempre coerência por parte do usuário ao conectar portas de acordo com os diferentes tipos de dados, pois os dados não são híbridos e nem possibilitam conversões automáticas entre tipos.

III. Metodologia

Para a criação de uma extensão devemos, primeiramente, entender um domínio e também uma biblioteca ou API referente a este domínio selecionado. Depois, é necessário encontrar padrões de código que possam ser encapsulados e reusados como base para a geração de código no Mosaicode [5]. Para encontrar este padrão, é necessário a criação de vários códigos exemplos, sejam eles básicos, avançados ou experimentais, tentando fazer com que os algoritmos sigam um modelo de escrita. Se houver um padrão de código, e partes em comum entre estes exemplos, está detectado um padrão no código em questão.

A criação de um nova extensão depende de:

a) Padrão de Código: Refere-se ao modelo ou template para geração e construção da aplicação. Aqui, elencamos as partes em repetição entre códigos similares e criamos um padrão que servirá de base para o processo de agrupar todas as partes comuns no trecho correto de código. Este modelo realizará a construção da aplicação disposta no Diagrama, gerando o código com as partes padrões fixadas e adicionando o que for necessário de acordo com os Blocos demandados pelo usuário. Para isto, é preciso especificar a linha de comando que o terminal irá receber para a compilação da aplicação gerada. Um exemplo de padrão de código é mostrado na figura 2.

Um template para geração de código de uma extensão para o Mosaicode deve possuir todos os trechos que são habituais em qualquer exemplo de uso da API em questão. É também necessário ter uma visualização do código gerado, para que a construção do modelo seja mais simples. Separam-se os trechos de código em partes distintas, como declaração de variáveis, funções externas necessárias, execução e desalocação de variáveis. Estas partes não necessariamente possuem um padrão, variam de extensão para extensão; é nelas que o código dos Blocos serão inseridos, mas não necessariamente todos os trechos deverão ser preenchidos por todos os Blocos. Na Figura 2 contém um exemplo de padrão de código em OpenGL.

```
class CFile(CodeTemplate):
    """
    This class contains methods related the JavascriptTemplate class.
    """
    # --------------------------------------------------------------------

    def __init__(self):
        CodeTemplate.__init__(self)
        self.name = "opengl"
        self.language = "c"
        self.description = "A full template to generate opengl code"
        self.extension = ".c"
        self.command = "gcc -Wall -g $dir_name$$filename$$extension$ -o $dir_name$$filename$ -lGL -lGLU -lglut -lm\n"
        self.command += "$dir_name$./$filename$"
        self.code_parts = ["global", "function", "call","idle","declaration", "execution"]
        self.code = """
#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{
}mosaicgraph_window_t;
mosaicgraph_window_t * mosaicgraph_create_window(float width, float height,float x, float y){
}
int mosaicgraph_draw_window(mosaicgraph_window_t * window){
}

//Function
$single_code[function]$
void display(){
}
void idle(){
}
int main (int argc, char** argv){
}
"""
```

Figura 2. Exemplo de Padrão de Código em OpenGL.

b) Portas e Conexões: Os tipos de Portas são definidos a parte e uma porta é associada a uma linguagem de programação. A definição da porta conta com a escrita do código que garantirá que o dado da porta de entrada irá interagir com a de saída. Se não houver nenhuma transmissão de valores, a conexão pode ser utilizada para definir a ordem dos Blocos na geração de código.

c) Blocos: Os Blocos possuem um padrão bastante preciso como um conjunto de portas e um conjunto de propriedades estáticas. Para construção dos mesmos, é preciso especificar a linguagem de programação (que normalmente segue um padrão para todos os Blocos de uma determinada extensão), o framework ou biblioteca que está sendo utilizado, e sua representação no ambiente como cor e categoria na qual ele pertencerá dentro da extensão. As categorias estão relacionadas à função em que determinado conjunto de blocos reproduzem, onde Blocos de um mesmo agrupamento possuem paradigmas ou comportamentos semelhantes.

Também são definidas propriedades estáticas do Bloco, sendo que cada propriedade deve possuir um nome, tipo, faixa de valores, valor inicial e conjunto de opções possíveis, caso se adeque ao comportamento da propriedade. Para garantir a troca de dados, caso necessário, são definidas as portas de entrada e de saída dos Blocos utilizando as Portas já definidas anteriormente. Após isto, são inseridos trechos de código, sendo que em cada trecho colocado, deve ser especificar o rótulo onde se deseja colocar o código, ao inserir o bloco. Nestes trechos, é possível resgatar variáveis referentes a portas e propriedades e

```
from mosaicode.GUI.fieldtypes import *
from mosaicode.model.blockmodel import BlockModel

class Icosahedron(BlockModel):

    # ----------------------------------------------------------------------
    def __init__(self):
        BlockModel.__init__(self)

        self.language = "c"
        self.framework = "opengl"
        self.help = "Not to declare"
        self.label = "Icosahedron"
        self.color = "150:150:250:150"
        self.group = "3D Shapes"
        self.ports = [{"type":"mosaicode_lib_c_opengl.extensions.ports.flow",
                "label":"Flow",
                "conn_type":"Input",
                "name":"flow"},
                {"type":"mosaicode_lib_c_opengl.extensions.ports.flow",
                "label":"Flow",
                "conn_type":"Output",
                "name":"flow"},
                {"type":"mosaicode_lib_c_opengl.extensions.ports.color",
                "label":"Color",
                "conn_type":"Input",
                "name":"color"}
            ]

        self.codes["function"] = """
        void mosaicgraph_draw_icosahedron(){
            glColor3f(0.8f,0.2f,0.0);
            glutSolidIcosahedron();
        }

"""
        self.codes["call"] = """
        mosaicgraph_draw_icosahedron();
"""
```

Figura 3. Exemplo do Bloco Icosahedron em OpenGL.

utilizar isto como elementos a serem manipulados. É possível também explicitar que um rótulo de variável declarada no pedaço do código receberá valor de identificação, criado no momento em que o código for gerado.

IV. Extensão C/OpenGL

Para Síntese de Imagens no ambiente Mosaicode, foi criada uma extensão baseada na biblioteca OpenGL. Esta biblioteca foi escolhida por ser uma conhecida API com rotinas gráficas para modelagem bidimensional e tridimensional, open-source e multiplataforma, escrita em C/C++. Um diagrama de exemplo é mostrado na Figura 4a.

A. Blocos

Foram criadas 5 categorias de Blocos para esta extensão: **2D Shapes**, **3D Shapes**, **Operations**, **Types** e **Window**. Na categoria **2D Shapes** estão localizados os objetos bidimensionais pré-criados, como Quadrilátero, Triângulo, Círculo e Elipse. Em **3D Shapes**, possuem todos modelos 3D nativos da biblioteca, como Cubo, Cone, Esfera e Teapot. Um exemplo de construção deste tipo de bloco é mostrado na figura 3.

Na categoria **Operations**, são realizadas transformações que influenciam na hierarquia do que sofrerá mudanças na tela, como Push e Pop e alterações que modificaram diretamente o conjunto de objetos, como Translate, Rotate e Scale. Na categoria **Types**, são criadas variáveis dos tipos de conexões a serem utilizados para inserir valores fixos em alguma porta. A categoria **Window** possui um bloco que altera os dados da janela como tamanho, cor de fundo e titulo.

B. Portas e Conexões

Foram criados, até o momento, três tipos de dados para as portas e conexões entre Blocos: **Color**, **Float** e **Flow**. Portas do tipo **Color** e **Float** servem para trocar valores entre Blocos enquanto que Portas do tipo **Flow** não transmitem dados e servem apenas para definir o fluxo de geração de código.

C. Padrão de Código

Foi criado um padrão de código para a geração de código que define as bibliotecas básicas para utilizar o OpenGL. Logo após há um trecho de código onde são declaradas variáveis utilizadas em todo o programa chamado **Global**. Depois, uma *struct* é criada, com parâmetros como tamanho, posição e título que compõe uma janela, uma função armazenando valores dentro dessa estrutura e outra que inicializa a tela. Em seguida, é definido um trecho de código para permitir a definição de funções específicas, chamado **Function**.

No momento seguinte, um procedimento chamado *display* é inicializado, onde são adicionados o que será traçado de fato no *canvas*. Dentro dela há um rótulo chamado **Call**, onde são inseridas as chamadas de função declaradas em function.Outra função é criado na sequência, onde é colocado o rótulo **Idle**, onde são inseridos valores a serem modificados a cada frame desenhado da cena. Para finalizar a função, é chamada a função *display* novamente, que será repetida mudando os parâmetros modificados no rótulo *idle*.

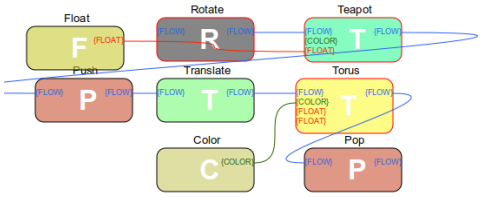
Para a função principal, são chamadas funções de inicialização da biblioteca. Após isto, é declarado o rótulo **Declaration**, onde são adicionadas e declaradas variáveis necessárias para a criação da janela com a tela. Em seguida é criada uma janela com as funções citadas acima, e chamado o rótulo **Execution**, em que são inseridos valores antes de ser desenhado no canvas. Por fim, é iniciado um *loop*, sendo que a função *display* é inicializada na primeira iteração, e a *idle* chamada a partir da segunda. A estrutura do padrão de código é mostrada na figura 2.

V. Extensão C/OpenCV

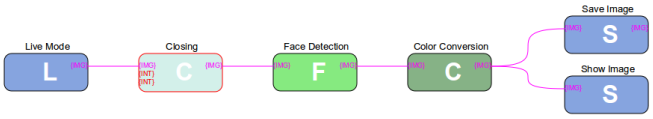
Para o Processamento de Imagens e Visão Computacional, o Mosaicode conta com uma extensão baseada em OpenCV, uma biblioteca open-source e multiplataforma criada pela Intel, na qual utilizamos atualmente em sua versão 3 para C++. O OpenCV possui o objetivo de facilitar a criação de aplicações com processamento em tempo real que exijam alta performance computacional [6]. Um exemplo de diagrama na extensão está na figura 4b.

A. Blocos

Os Blocos da extensão de Visão Computacional foram divididos em 12 Categorias: Operações Aritméticas, Lógicas, Matemáticas, Morfológicas, Filtros, Conversão de Cores e Detecção de Padrões. Especificamente, apenas a última citada possui Blocos responsáveis pela parte de Visão Computacional, incluindo funções como detecção de faces, objetos e vértices mais acentuados. As demais categorias possuem diversas funções básicas para o processamento, tratamento e manipulação de imagens.



(a) Diagrama utilizando a extensão OpenGL que rotaciona um **Teapot** com um tamanho inicializado pelo bloco **Float** e um **Torus** que rotaciona e translada na cena, com uma cor inicializada pelo bloco **Color**.



(b) Diagrama utilizando a extensão OpenCV que aplica uma operação de fechamento, detecta faces e converte o sistema de cores em uma câmera em tempo real.

Figura 4. Diagramas construídos utilizando o ambiente Mosaicode.

B. Portas e Conexões

Para a conexão e troca de dados entre os Blocos foram criados cinco tipos de Portas: **Image**, **Int**, **Double**, **Rect** e **Point**.

C. Padrão de Código

Nesta extensão em específico, a geração de código é realizada através da conjunção de cinco partes de código distintas. Inicia-se com a **Include**, onde encontram-se as chamadas de todas as bibliotecas e arquivos necessários para a aplicação final. Após esta seção temos a **Function** que traz alguma função externa necessária para o funcionamento de alguma parte específica de código comum a diversos BLocos. Em sequência, a **Declaration** inclui a nomeação de todas as variáveis e tipos a serem usados na aplicação e a **Execution** realiza o algoritmo construído. Por fim, a **Deallocation** efetua as desalocações das variáveis alocadas.

VI. Conclusão

Este trabalho apresentou a geração de código para os domínios de Visão computacional, computação gráfico e processamento Digital de Imagens utilizando, para isto, o Mosaicode. Esta geração depende de alguns fatores como a construção de Blocos de programação para estes domínios específicos. A construção dos blocos para este ambiente de programação visual continua em andamento, buscando cobrir mais funcionalidades das bibliotecas aqui apresentada e, com isto, gerar aplicações mais complexas. Para extensão em OpenGL para Síntese de Imagens, inicialmente há a intenção de adicionar blocos que mudem a perspectiva de visão da câmera, por exemplo. Blocos referentes à iluminação seriam também bastante relevantes.

Para a extensão do OpenCV também existem planos futuros. Como já mencionado, a mesma possui variados blocos em relação à Processamento Digital de Imagens, considerando todas as funções básicas e intermediárias. Em relação à isto, poderão surgir novos blocos que desempenhem funções mais robustas; para a seção de Visão Computacional a demanda de blocos deverá ser maior e abordar variadas técnicas de identificação de padrões até recursos para geração dos dados identificados para reaproveitamento.

Ao mesmo tempo, para ambas as extensões, seria interessante criar blocos que trouxessem resultados rápidos para seus usuários especificamente no contexto de Arte Digital. Um caminho para isto seria a criação de blocos que tracem desenhos ou manipulem imagens utilizando formas matemáticas, como a sequência de Fibonacci [7] ou a teoria dos fractais [8].

Outro passo seria a validação destes blocos em sala de aula, no contexto de disciplinas como Computação Gráfica, Processamento Digital de Imagens e Visão Computacional, dando suporte aos alunos para a prototipação de aplicações e auxiliando-os no ensino e aprendizado do funcionamento da estrutura de um programa utilizando bibliotecas como OpenGL ou OpenCV.

Uma trabalho futuro é permitir a conexão do código destas duas extensões, openGL e openCV, citadas neste artigo de maneira a integrar a programação visual de síntese de imagens com visão computacional.

VII. Agradecimentos

Os autores gostariam de agradecer a o apoio institucional da FAPEMIG, CNPq e da PROAE/PROPE/UFSJ. Também gostariam de agradecer os integrantes do Arts Lab in Interfaces, Computers, and Else (ALICE.) e do Grupo de Estudo em Arte Digital do Departamento de Computação da Universidade Federal de São João del-Rei.

Referências

[1] 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)*, Curitiba - PR - Brazil, Nov 2017, pp. 200–206.

[2] 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.

[3] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. "O'Reilly Media, Inc.", 2008.

[4] R. S. Pressman, "Engenharia de software. tradução: Rosângela delloso penteado," 2006.

[5] J. Herrington, *Code generation in action*. Manning Publications Co., 2003.

[6] K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov, "Real-time computer vision with opencv," *Communications of the ACM*, vol. 55, no. 6, pp. 61–69, 2012.

[7] T. H. Garland, *Fascinating Fibonaccis: Mystery and Magic in Numbers*. ERIC, 1987.

[8] C. Redies, J. Hasenstein, and J. Denzler, "Fractal-like image statistics in visual art: similarity to natural scenes," *Spatial Vision*, vol. 21, no. 1, pp. 137–148, 2007.