Universidade Federal da Bahia - UFBA
Instituto de Matematica - IM
Sociedade Brasileira de Matematica - SBM
Mestrado Profissional em Matemática em Rede Nacional - PROFMAT
Dissertação de Mestrado

Uso de Programação no ensino das Transformações Geométricas no Plano

José Benício dos Anjos França

Salvador - Bahia
Março de 2016

Uso de Programação no ensino das Transformações Geométricas no Plano

José Benício dos Anjos França

Dissertação de Mestrado apresentada à Comissão Acadêmica Institucional do PROFMAT-UFBA como requisito parcial para obtenção do título de Mestre em Matemática.

Orientador: Prof. Dr. Vinícius Moreira Mello.

SALVADOR - BAHIA

MARÇO DE 2016

SISTEMA DE BIBLIOTECAS DA UFBA

França, José Benício dos Anjos

Uso de programação no ensino das transformações geométricas no plano / José Benício dos Anjos França. – 2016.

179f.: il.

Inclui apêndices.

Orientador: Prof. Dr. Vinícius Moreira Mello.

Dissertação (mestrado) – Universidade Federal da Bahia, Instituto de Matemática, Salvador, 2016.

1. Construções Geométricas. 2. Linguagem de Programação (Computadores). 3. Processing (Linguagem de programação de computadores). I. Mello, Vinícius Moreira. II. Universidade Federal da Bahia. Instituto de Matemática. III. Título.

CDU - 516

CDU - 514

Uso de Programação no Ensino das Transformações Geométricas no Plano

José Benício dos Anjos França

Dissertação de Mestrado apresentada à comissão Acadêmica Institucional do PROFMAT-UFBA como requisito parcial para obtenção do título de Mestre em Matemática, aprovada em 18/03/2016.

Banca Examinadora:

Prof. Dr. Vinícius Moreira Mello (orientador)
UFBA

Prof.ª Dr.ª Mariana Cassol
UFBA

Prof. Dr. Perfilino Eugênio Ferreira Junior
UFBA

À minha família

Agradecimentos

A Deus

Ele que ilumina e rege a minha vida, fortalecendo a cada encalço que a vida oferece, mostrando que mesmo nas dificuldades existe a beleza da vida. Neste dia não quero pedir nada, apenas agradecer por tudo que já pedi e me foi consagrado. Com sua graça e benção continuarei a trilhar os caminhos em busca da evolução plena.

A família

Fonte de vida e energia, onde encontro o paraíso da harmonia e tranquilidade para reestruturar-me cada vez mais, pois com o amor familiar só temos a nos fortalecer mesmo diante das dificuldades. Em especial a minha mãe Maria da Conceição e irmã Patrícia por sempre me apoiar mesmo vendo as dificuldades da profissão e o tempo para conseguir estudar, e a meu irmão Renato por me ajudar com linguagem de programação nos momentos de dificuldade e dúvidas.

À UFBA e aos professores colaboradores

A todos os que integram a Universidade Federal da Bahia, em especial aos professores que colaboraram, e tiveram compreensão da necessidade deste curso, e de quão seria importante para meu crescimento profissional e pessoal. E ao professor Vinícius Moreira Mello pela indicação da linguagem de programação *Processing.JS*, mesmo depois de testarmos a linguagem Blockly da Google sem muito sucesso com os materiais de apoio e instrução.

Ao quarteto fantástico

CAPES, SBM, PROFMAT e a UFBA, pela implantação do curso e estruturação do mesmo. Ao mesmo tempo pela disponibilização de materiais e livros a preços acessíveis com alta qualidade, pois sua aquisição foi fundamental e imprescindível para o sucesso profissional e acadêmico.

A Eliane Santos Alves

Pela nossa relação de cumplicidade, amor, carinho, preocupação e sinceridade. E

por ter me compreendido nos momentos finais do mestrado onde precisei me debruçar sobre o computador para aprender a programar antes de ensinar o que é programar.

Aos colegas do curso

Cada um possui uma importância no decorrer do curso, pois cada ser tem sua especificidade, e compete a todos nós admirarmos na observação e respeitarmos acima de qualquer diferença. Alguns encontravam-se mais afastados, outros mais próximos, mesmo assim todos contribuíram para esta rica convivência em grupo durante o curso. Aos mais próximos espero ter sido presente nas solicitações, assim como foram para mim, e os momentos que vivenciamos juntos seja em grupos de estudos ou para discontração como o jogo de boliche ou os almoços serão sempre inesquecíveis. E quero aqui deixar registrado aos colegas de equipe da disciplina Recursos Computacionais no ensino de Matemática – MA36 – o apoio e o desprendimento pela escolha do tema para minha defesa de mestrado, são eles: Sidnéia, Marconi, Rubens e Vitório.

Aos Amigos e colegas de trabalho

Todos possui uma importância em cada fase de nossas vidas, mas quero deixar aqui registrado o apoio que os colegas de trabalho da Escola Municipal Amélia Rodrigues prestaram juntamente com a direção, Andrea Tavares, no percurso deste mestrado, em especial, minha amiga Nadjara que sempre apoiou minhas peripécias matemáticas com algumas contribuições particulares para aperfeiçoa-la em apresentações e/ou ministrando aulas.

“Nada perece no Universo; tudo
quanto nele acontece não passa de
meras transformações".
Pitágoras

Resumo

Este trabalho apresenta uma síntese histórica do processo de implantação do uso de computadores na educação, explicitando o surgimento das linguagens de programação com sua evolução e contribuição para a mesma, e um conjunto de atividades exploratória para serem trabalhadas na sala de informática como recurso didático para uma aprendizagem significativa a respeito das Transformações Geométricas no Plano através do uso da linguagem de programação *Processing* nas turmas do 9º ano do Ensino Fundamental II da Unidade Escolar Amélia Rodrigues, Monte Gordo-Camaçari/BA. Com o *Processing* é possível propor e realizar atividades lúdicas que promovam uma aprendizagem através do ensino de programação de computadores que consiste basicamente em instruir o computador a realizar uma determinada atividade. Essa linguagem possibilita um primeiro contato com os princípios da computação para a geração de aplicativos e jogos, viabilizando com isso um futuro profissional para os interessados em dar continuidade na aprendizagem que se inicia a partir do curso de programação no site da Khan Academy.

Palavras chave: Linguagem de Programação; Transformação Geométrica; *Processing.JS*.

Abstract

This paper presents a historical overview of the implementation of the use of computers in education process, explaining the emergence of programming languages with their development and contribution to it, and a set of exploratory activities to be worked in the computer room as a resource of teaching for a significant learning about the Geometric Transformations in the plane by using *Processing* programming language in the 9th year of the Secondary School of Unit Amelia Rodrigues, Monte Gordo-Camaçari/BA. With *Processing* is possible to propose and carry out recreational activities that promote learning through computer programming education consisting primarily to instruct the computer to perform a particular activity. That language will allow a first contact with the principles of computing to generate applications and games, allowing thereby a professional future for those interested in learning that starts from the programming course at the Khan Academy website.

Key words: Programming Language; Geometric Transformation; *Processing.JS*.

Lista de Figuras

1 Correntes de ensino-aprendizagem usando o computador . . . . p. 26

2 Segmentos de reta para o algoritmo de Euclides . . . . p. 33

3 MDC de Euclides no retângulo . . . . p. 33

4 Comando para exibição de mensagens e elipse . . . . p. 43

5 IDE e aplicação da elipse animada . . . . p. 46

6 Sistema de Coordenadas Cartesianas e no *Processing* . . . . p. 47

7 Construções no Modo Básico e Modo Contínuo . . . . p. 49

8 Declarações de variáveis . . . . p. 51

9 Diagrama de uma estrutura if-else . . . . p. 52

10 Construção condicional com if-else . . . . p. 52

11 Diagrama e Construção com a estrutura switch(); . . . . p. 53

12 Operadores condicionais . . . . p. 54

13 Operadores Lógicos . . . . p. 55

14 Diagrama de Fluxo com repetição simples . . . . p. 55

15 Desenhando formas com o comando while . . . . p. 56

16 Desenhando formas com o comando do {} while( ) . . . . p. 57

17 Desenhando formas com o comando for . . . . p. 58

18 Animações na estrutura for associado a vetores . . . . p. 62

19 Imagem em tons de cinza usando uma matriz . . . . p. 63

20 Construção do tabuleiro para o Jogo da Velha . . . . p. 64

21 Construção com função de regresso . . . . p. 65

22 Detalhe da DRP de Escher . . . . p. 67

23 Translação do ponto M segundo o vetor $\vec{v}$ . . . . p. 71

24 Translação do ponto M segundo o segmento $\overline{PQ}$ . . . . p. 72

25 Composição de translação do ponto K em K'' . . . . p. 73

26 Reflexão . . . . p. 74

27 Reflexão no plano cartesiano . . . . p. 74

28 Reflexão em relação a um ponto e uma reta r qualquer . . . . p. 75

29 Rotação do ponto X com centro em O e amplitude α . . . . p. 79

30 Rotação do ponto X e amplitude α com centro na origem e no ponto C p. 80

31 Homotetia de redução $(0 < |k| < 1)$ e ampliação $|k| > 1$ . . . . p. 83

32 Dilatação horizontal e vertical . . . . p. 84

33 Fotografia de uma estrada . . . . p. 86

34 Construção de triângulos retângulos com efeito de translação . . . . . . p. 88

35 Código de Translação com a função $pushMatrix$ e $popMatrix$ . . . . . p. 89

36 Construção de triângulos com efeito de simetria . . . . p. 90

37 Construção de triângulos com efeito de rotação . . . . p. 91

38 Fluxograma com esquema de uma rotação no *Processing* . . . . p. 91

39 Construção de quadrados com efeito de ampliação e redução . . . . . . p. 92

40 Construção de círculos com dilatação horizontal e vertical . . . . p. 93

41 Applet 2: Boneco geométrico . . . . p. 98

42 Applet 3: Translação primária . . . . p. 101

43 Applet 4: Translação do café da manhã . . . . p. 104

44 Construção do cenário casa com árvores e céu ensolarado . . . . p. 108

45 Complementação do cenário com bonecos no ambiente . . . . p. 111

46 Applet 5: Cenário com animação da tartaruga no lago . . . . p. 114

47 Medidas de ângulos em graus e em radianos . . . . p. 116

48 Rotação do retângulo de 50 por 70 pixels na malha quadriculada . . . . p. 118

49 Applet 6: Rotação-Translação e Translação-Rotação no retângulo de 50 por 70 pixels . . . . p. 120

50 Applet 7: Giro de quadrados com cores oscilantes . . . . p. 122

51 Applet 7.1: Catavento quadrangular em oscilação de cores . . . . p. 123

52 Applet 8: Construção dos elementos da mandala . . . . p. 126

53 Applet 8.1: Mandala em duas perspectivas centrais . . . . p. 126

54 Operadores lógicos “e”, “ou” e “negação” . . . . p. 129

55 Applet 9: Controle de movimento com rotação . . . . p. 133

56 Escudo para montagem . . . . p. 136

57 Applet 10: Simetria em relação ao eixos e a origem . . . . p. 139

58 Estrutura para o mosaico . . . . p. 141

59 Mosaico completo . . . . p. 143

60 Applet 11: Caleidoscópio do mosaico interativo . . . . p. 146

61 Minions do Filme Meu Malvado Favorito . . . . p. 148

62 Applet 12: Exercíto de 2 Minions . . . . p. 150

63 Triângulo de Sierpinski . . . . p. 152

64 Molde do Triângulo de Sierpinski . . . . p. 154

65 Applet 13: Construção do Triângulo de Sierpinski . . . . p. 155

66 Applet 13.1: Triângulo de Sierpinski no *Processing* . . . . p. 157

67 Applet 14: Dilatação e contração irregular . . . . p. 159

68 Transformação com dois pontos de fuga . . . . p. 172

Lista de Tabelas

1 Tabela com os comandos *textFont* e *ellipse* . . . . p. 43

2 Níveis de complexibilidade da programação *Processing* . . . . p. 48

3 Comandos classificados por modalidade de construção . . . . p. 48

4 Construções Iniciais no Modo Básico e Modo Contínuo . . . . p. 49

5 Tipos de dados para variáveis . . . . p. 51

6 Tabela de exemplos dos operadores lógicos . . . . p. 54

7 Quadro comparativo entre as estruturas de repetição . . . . p. 59

8 Comando for associado a vetores . . . . p. 61

9 Matriz bidimensional . . . . p. 62

Lista de Abreviaturas e Símbolos

API	Application Programming Interface
BASIC	Beginners All-purpose Symbolic Instruction Code
CAI	Computer Aided Instruction
CAPRE	Comissão Coordenadora das Atividades de Processamento Eletrônico
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
COBOL	COmmon Business Oriented Language
CODASYL	Conference on Data Systems Languages
DIGIBRÁS	Empresa Digital Brasileira
DRP	Divisão Regular do Plano
EAO	Enseignemente Assisté par Ordinateur
Fortran	IBM Mathematical FORmula TRANslation System
IBM	International Business Machines
IME	Instituto de Matemática e Estatística
LISP	LISt Processor
MATLAB	MATrix LABoratory
MDC	Máximo Divisor Comum
MEC	Ministério da Educação e Cultura
MIT	Massachussets Institute of Tecnology
PCN+	Orientações Complementares dos Parâmetros Curriculares Nacionais do Ensino Médio

PCN-EF	Parâmetros Curriculares Nacionais do Ensino Fundamental
PCN-EM	Parâmetros Curriculares Nacionais do Ensino Médio
PDE	*Processing* Development Environment
PROINFO	Programa Nacional de Tecnologia Educacional
PRONINFE	Programa Nacional de Informática na Educação
rad	Radiano unidade de medida do ângulo plano no Sistema Internacional
SEI	Secretária Especial de Informática
TGP	Transformações Geométricas no Plano
TIC	Tecnologia de Comunicação e Informação
UCA	Um Computador por Aluno
UFBA	Universidade Federal da Bahia
UFRGS	Universidade Federal do Rio Grande do Sul
UFRJ	Universidade Federal do Rio de Janeiro
UFSCAR	Universidade Federal de São Carlos
UNICAMP	Universidade Estadual de Campinas
USP	Universidade de São Paulo

Sumário

Introdução p. 18

1 História da Programação no Ensino de Matemática p. 23

1.1 História da Computação na Educação . . . . p. 25

1.2 Evolução da Linguagem de Programação . . . . p. 31

1.2.1 Pré-história da Linguagem de Programação . . . . p. 31

1.2.2 Primeiras Linguagens de Programação . . . . p. 36

1.2.2.1 Linguagens de Baixo Nível . . . . p. 39

1.2.2.2 Linguagens Não-Estruturadas . . . . p. 39

1.2.2.3 Linguagens Procedurais . . . . p. 40

1.2.2.4 Linguagens Funcionais . . . . p. 40

1.2.2.5 Linguagens Orientadas a Objeto . . . . p. 41

1.2.2.6 Linguagens Específicas a Aplicações . . . . p. 41

1.2.2.7 Linguagens Visuais . . . . p. 41

1.3 História do *Processing* . . . . p. 42

2 Descrição da Linguagem *Processing* p. 45

2.1 Comandos Básicos e Menu . . . . p. 46

2.2 Variáveis e Tipos de dados . . . . p. 50

2.3 Estrutura condicional e operadores lógicos . . . . p. 51

2.4 Estrutura de Repetição . . . . p. 54

2.5 Variáveis Complexas . . . . p. 59

2.6 Funções ou Métodos . . . . p. 62

3 Transformação Geométrica no Plano p. 66

3.1 Análise matemática das transformações geométricas . . . . p. 69

3.2 Transformações Isométricas . . . . p. 71

3.2.1 Translação . . . . p. 71

3.2.2 Reflexão (ou Simetria) em relação a uma reta . . . . p. 73

3.2.3 Rotação . . . . p. 79

3.3 Transformações Isomórficas . . . . p. 82

3.3.1 Homotetia . . . . p. 82

3.4 Transformações Anamórficas . . . . p. 83

3.4.1 Dilatação . . . . p. 84

4 Transformações Geométricas em *Processing* p. 85

4.1 Transformações isométricas no *Processing* . . . . p. 87

4.2 Transformações isomórficas no *Processing* . . . . p. 92

4.3 Transformações anamórficas no *Processing* . . . . p. 93

5 Atividades Propostas p. 95

5.1 Explorando o aplicativo . . . . p. 96

5.2 Isometria de Translação . . . . p. 99

5.2.1 Conhecendo a translação . . . . p. 100

5.2.2 Aprimorando a translação . . . . p. 101

5.2.3 Macro construção com a translação . . . . p. 104

5.3 Isometria de rotação . . . . p. 115

5.3.1 Conhecendo a rotação . . . . p. 115

5.3.2 Aprimorando a rotação . . . . p. 122

5.3.3 Animação com efeito de rotação . . . . p. 127

5.4 Isometria de reflexão ou simetria . . . . p. 134

5.4.1 Conhecendo a simetria . . . . p. 135

5.4.2 Aprimorando a simetria . . . . p. 139
5.5 Transformação de homotetia . . . . p. 146
5.5.1 Conhecendo a homotetia . . . . p. 147
5.5.2 Aprimorando a homotetia . . . . p. 151
5.6 Transformação de dilatação/contração irregular . . . . p. 157

6 Considerações Finais p. 160

Referências Bibliográficas p. 162

Apêndice A – Transformações Lineares p. 166

Apêndice B – Transformações Projetivas p. 169
B.1 Transformação Projetiva Plana . . . . p. 169

Apêndice C – Mosaico para caleidoscópio p. 174

Introdução

Os livros didáticos de matemática utilizados no Ensino Fundamental, anos finais, e no Ensino Médio limitam-se na sua maioria ao estudo das relações de congruência e semelhança de triângulos sem correlacionar com as transformações geométricas no plano (TGP) e os efeitos que tais transformações geram na análise do comportamento das funções. Essa realidade é reforçada pelos docentes ao ensinarem geometria plana e o comportamento das funções descontextualizado com as transformações geométricas, a saber: *translação, reflexão* e *rotação* chamadas de transformações isométricas e as *homotetias.*

Ambicionamos que, se a abordagem do comportamento gráfico de funções iniciasse por meio da observação de diferentes padrões e movimentos no plano através das transformações geométricas, o aluno poderia chegar às generalizações matemáticas necessárias com maior apropriação do conteúdo estudado, e consequentemente, proporciona ao discente o desenvolvimento de capacidades como preconiza os Parâmetros Curriculares Nacionais do Ensino Fundamental – PCN-EF [1].

> O ensino de Matemática deve garantir o desenvolvimento de capacidades como: observação, estabelecimento de relações, comunicação (diferentes linguagens), argumentação e validação de processos e o estímulo às formas de raciocínio como intuição, indução, dedução, analogia e estimativa [1, p. 56].

Esses padrões existentes no comportamento geométrico e de funções estão presentes em diversos campos de atuação aos quais os alunos posteriormente podem ingressar, como na engenharia, na arquitetura e na arte. Vivemos em um universo repleto de padrões onde a Matemática representa uma das ciências responsáveis pela criação de teorias que revelam os segredos da natureza, mostrando variados padrões que revelam o comportamento de uma função. Diante do exposto, Lima [2] ressalta que o ensino de Matemática deve abranger três componentes essenciais, a conceituação, a manipulação e a aplicação, sendo imprescindível a presença de cada um deles, de forma dosada, para o sucesso de qualquer curso. Esta dosagem é importante para que não volte a acontecer como nas décadas de 60 e 70 onde o excesso de conceituação sem focar nos detalhes usuais direcionaram para um ensino sem objetivos concretos.

Neste sentido, o docente possui o papel de apresentar o mundo dos padrões para

o discente com o intuito de despertar a admiração por uma matemática observável e aparentemente agradável de estudar ao ponto de se aprofundar nos conceitos e reconhecer que a matemática não é uma ciência alheia à realidade. Para Devlin [3] o mundo dos padrões é o real objeto de estudo dos matemáticos contemporâneos.

> O que o matemático faz é examinar "padrões" abstratos, padrões numéricos, padrões de forma, padrões de movimento, padrões de comportamento, etc. Esses padrões tanto podem ser reais como imaginários, visuais ou mentais, estáticos ou assumindo um interesse pouco mais recreativo. Podem surgir a partir do mundo à nossa volta, das profundezas do espaço e do tempo, ou das atividades mais ocultas da mente humana [3, p. 9].

Nos processos de generalização de um certo padrão é comum sermos conduzidos à apresentar resultados equivocados, no entanto, é neste ponto que busca-se através da matemática, aperfeiçoar-se no reconhecimento desses padrões em diferentes formas, como na resolução de problemas que precisam de observação, seleção de dados, representação geométrica, algébrica ou aritmética, interpretação e generalização. Este é o princípio que direciona as pessoas no desenvolvimento do pensar matemático e consequentemente em qualquer outra forma de pensar [4].

O processo de resolução de problemas é uma excepcional estratégia de aprendizagem, tendo em vista que o aluno é oportunizado a observar livremente os dados, organizá-los, representá-los e propor soluções. Esta liberdade de pensamento contribui para o despertar cognitivo como aponta os PCN+ (Orientações Complementares dos Parâmetros Curriculares Nacionais do Ensino Médio):

> A resolução de problemas é peça central para o ensino de matemática, pois o pensar e o fazer se mobilizam e se desenvolvem quando o indivíduo está engajado ativamente no enfrentamento de desafios. Esta competência não se desenvolve quando propomos apenas exercícios de aplicação de conceitos e técnicas matemáticas, pois, neste caso, o que está em ação é uma simples transposição analógica: o aluno busca na memória um exercício semelhante e desenvolve passos análogos aos daquela situação, o que não garante que seja capaz de usar seus conhecimentos em situações diferentes ou mais complexas[5, p. 112].

Entretanto, na sala de aula existe muita repetição de algoritmos sem significação para o aluno acompanhado de conteúdos estanques pré-selecionados pelo professor, quando a matriz curricular não pode ser cumprida na integra. Onde este professor em muitas situações não possui domínio de conteúdo e nem segurança no ato de ensinar, e isso contribui para a extinção do ensino da geometria no Ensino Fundamental além de colaborar com a ampliação do deficit de aprendizagem que é irrecuperável para o discente.

Neste processo de aprendizagem existe um exagero algébrico no Ensino Fundamental que estende-se ao Ensino Médio ao ponto de conjecturar as funções de forma puramente algébrica e isoladas.

A fragmentação do ensino de matemática em função do acúmulo de conteúdos desconexos contribui para uma formação acadêmica de pessoas incapazes de aplicar os mais diversos conceitos matemáticos no dia-a-dia, nem fazer correlações com outros blocos das diversas áreas do conhecimento.

Fundamenta-se ainda a escolha do tema pela importância em que os casos de simetria auxiliam na elaboração do pensamento matemático e possibilidade de correlacionar com outras áreas como evidencia os PCN-EF na escolha de conteúdos:

> Deve destacar-se também nesse trabalho a importância das transformações geométricas (isometricas, homotetias), de modo que permita o desenvolvimento de habilidades de percepção, por exemplo, das condições para que duas figuras sejam congruentes ou semelhantes [1, p. 51].

Salienta-se ainda que, segundo Stormowski [6], as transformações geométricas aparecem nos Parâmetros Curriculares Nacionais do Ensino Médio (PCN-EM) apenas como uma forma de complementar o estudo geométrico e particularmente o estudo dos números complexos. Tanto Stormowski [6] como Cerqueira [7] concordam com a existência de uma diferença na abordagem deste assunto por parte dos documentos que tratam do ensino fundamental e médio. Para Cerqueira [7] a abordagem dos conteúdos é mais específica por não abordar os conceitos de forma geral e aberta como encontramos nos PCN-EF.

Diante do exposto o presente trabalho propõe-se em apresentar o estudo das transformações geométricas através do uso da linguagem de programação como suporte para uma aprendizagem significativa baseada na experimentação e manipulação de dados. A partir do curso de programação oferecido pela Khan Academy, após o curso os discentes realizaram uma série de atividades desenvolvidas diretamente no *Processing.JS* tomando por base o conhecimento adquirido e as orientações constantes no roteiro de estudo. Os alunos piloto do projeto são todos oriundos de uma comunidade de baixa renda da região de Monte Gordo/Camaçari-BA e encontram-se no 9º ano do Ensino Fundamental II da própria Unidade Escolar[1] que disponibilizará o Laboratório de Informática para os encontros orientados e execução das atividades. A princípio o Laboratório será utilizado com o intuito de matricularem os discentes no portal da Khan Academy enquanto iniciam as primeiras aulas de programação direcionada de forma cronológica para fundamentar toda

[1]Escola Municipal Amélia Rodrigues situada na Rua São Bento

a teoria de algoritmo, sintaxe entre outros elementos imprescindíveis para uma aprendizagem formativa e significativa nesse ramo da ciência. Contudo, como o professor pode acompanhar a evolução de seus discentes através de um portal de aprendizagem tão amplo?

A esse respeito o próprio sistema da Khan Academy foi projetado para favorecer uma aprendizagem à distância sem perder de vista o caráter presencial, ou seja, é possível inscrever os alunos em diversos cursos e ainda assim acompanhar a evolução de cada um deles através do código liberado para o cadastro de turmas que funciona basicamente em duas modalidades: Na primeira, o próprio professor convida os alunos a participarem do curso enviando um email onde eles só precisam confirmar a participação e; Na segunda opção, o professor disponibiliza um código que os próprios alunos podem adicionar assim que se inscreverem no portal com uma conta de email ativa da google ou facebook. De posse da assinatura do portal e com o sistema inicializado o curso proporciona aos envolvidos uma aprendizagem de programar desenhos, animações e jogos usando JavaScript e *Processing.JS*. Para os alunos motivados é possível finalizar o curso aperfeiçoando com a criação de páginas Web com linguagem HTML e CSS. O intuito dos alunos participarem desse curso visa as construções de imagens transformadas no plano como proposta de atividade utilizando o programa *Processing*.

No entanto, para o andamento desse curso o aluno deverá ter conhecimentos prévios sobre:

- Geometria Plana – Polígonos regulares e simetrias;
- Relações Binárias;
- Noções de Função;
- Conhecimentos básicos de computação: Word, Excel e Internet.

Além dos requisitos pedagógicos tradicionais, o aluno precisará de um computador com *Processing.JS*[2] instalado e com acesso a Internet para realizar o curso na Khan Academy. Esse *software* é de domínio público e de fácil uso. Trata-se de uma linguagem que possibilitará ao aluno um primeiro contato com a programação, percebendo assim, a importância das transformações geométricas bem como de outros temas da Matemática para criar programas.

Em função disso, o trabalho encontra-se subdividido em 6 capítulos.

[2]A versão mais recente está disponível para download em *https://github.com/processing/processing/releases*

No Capítulo 1, apresentamos o percurso histórico da inserção da computação no ensino da matemática, com ênfase para a evolução das linguagens de programação e as contribuições que tal recurso oferece para a disciplina. Além da história do *Processing* e sua aplicabilidade na matemática.

No Capítulo 2, divulgamos uma descrição da linguagem utilizada no *Processing* com os principais comandos e efeitos gráficos.

No Capítulo 3, abordamos as Transformações Geométricas do Plano Cartesiano, assim como sua representação matricial e sua relação com o produto de matrizes.

No Capítulo 4, expomos uma relação entre as TGP com a linguagem de programação *Processing*, objetivando a compreensão gráfica oferecidas por tais transformações.

No Capítulo 5, difundimos nossas propostas de atividades com roteiro que criem uma relação entre as transformações geométricas descritas no Capítulo 3 com a linguagem de programação descrita no Capítulo 2 e Capítulo 4. Tais atividades são imprescindíveis por possibilitar a visualização e o entendimento das transformações de forma dinâmica e ao mesmo tempo criativa.

No Capítulo 6, faremos as considerações finais do trabalho com as respetivas conclusões de forma a contribuir para o desenvolvimento deste segmento da Matemática.

Para finalizar,, após as referências bibliográficas apresentaremos nos apêndices as Transformações Lineares e Transformações Projetivas, em especial, a Projetiva Plana imprescindível para demonstrar como as transformações geométricas e a programação em computação se interligam, isto é, como a matemática é utilizada nas linguagens de programação no que se refere ao estudo das TGPs.

1 *História da Programação no Ensino de Matemática*

O uso do computador como recurso pedagógico em sala de aula nos últimos anos passou a ser uma necessidade na formação do indivíduo para o mercado de trabalho de tal forma que a discussão sobre as vantagens ou desvantagens da utilização dessa Tecnologia de Comunicação e Informação (TIC) ficou restrito a década de 90 e início dos anos 2000 [8, 9, 10, 11]. Ao invés disso, discuti-se hoje como usar esse recurso para auxiliar didaticamente na promoção da aprendizagem eficiente [12].

De acordo com Valente [10], a inclusão do computador na educação brasileira sempre esteve atrelada historicamente ao fato dessa provocar mudanças pedagógicas, objetivando com isso uma educação centrada no ensino, na transmissão da informação, para resultar numa educação centrada no aluno, onde o mesmo pudesse aprender por intermédio das atividades via computador.

A interação do aluno com o meio em que vive é facilmente constatada com a inserção das tecnologias no ensino de Matemática. Para Sordo Juanena [13] devemos fixar os olhares no estudo das interrelações entre o aspecto tecnológico, educativo e matemático.

Esse autor [13] defende ainda três características primordiais do ponto de vista didático ao se utilizar o computador como ferramenta em sala de aula. São elas:

1. A atenção dos alunos volta-se para o significado dos dados e análise dos resultados.
2. Permite aos alunos prognosticar ordens diversas do tipo (desenhos, cálculos, decisões, etc.) com muito mais rapidez.
3. Interação com os alunos que pode intervir em determinados momentos para prestar informações ou novas tarefas com base em resultados obtidos, tornando-se desta forma uma poderosa ferramenta de exploração e investigação.

Outro ponto importante que consta nos PCN-EM [14] relacionado ao ensino da

matemática enfatiza a necessidade da organização curricular com o intuito de favorecer o desenvolvimento das competências e habilidades desejadas aos alunos do ensino básico.

> Esse impacto da tecnologia, cujo instrumento mais relevante é hoje o computador, exigirá do ensino de Matemática um redirecionamento sob uma perspectiva curricular que favoreça o desenvolvimento de habilidades e procedimentos com os quais o indivíduo possa se reconhecer e se orientar nesse mundo do conhecimento em constante movimento. Para isso, habilidades como selecionar informações, analisar as informações obtidas e, a partir disso, tomar decisões exigirão linguagem, procedimentos e formas de pensar matemáticos que devem ser desenvolvidos ao longo do Ensino Médio, bem como a capacidade de avaliar limites, possibilidades e adequação das tecnologias em diferentes situações [14, p. 41].

Borba [15, p. 285] também evidencia:

> A introdução das novas tecnologias – computadores, calculadoras gráficas e interfaces que se modificam a cada dia – tem levantado diversas questões. Dentre elas destaco as preocupações relativas às mudanças curriculares, às novas dinâmicas da sala de aula, ao "novo" papel do professor e ao papel do computador nesta sala de aula.

Dessa análise dos PCN-EM e Borba evidenciamos a importância do recurso computacional para o desenvolvimento de uma linguagem a ser desenvolvida não somente no Ensino Médio, acreditamos que o incremento dessa linguagem é plausível a partir do 6º ano do Ensino Fundamental com atividades direcionadas com focos distintos daqueles denotados no Ensino Médio. A linguagem de programação é uma dessas linguagens admissíveis no Ensino Fundamental II e consiste basicamente em "Ensinar" o computador a executar um determinado serviço e para isso, o programador precisa utilizar uma sequência lógica a partir da linguagem formal e precisa, onde o papel do professor é fundamental de forma a mediar todo o processo. Com isso, o aluno passa a realizar uma série de atividades que são fundamentais para a aquisição de novos conhecimentos [9].

Nesse sentido, um professor mediador é aquele que trabalhar em constante "zona de risco". E ao sair de sua "zona de conforto" e instigar o aluno à investigação, o professor esta favorecendo uma aprendizagem ativa, crítica opondo-se a aprendizagem passiva [1] [16].

[1]Na aprendizagem ativa o aluno é o principal construtor do processo de ensino. Nesse processo de aprendizagem o professor não ensina diretamente, a sua função consta em criar ferramentas e construir em conjunto com os alunos um ambiente favorável à aprendizagem. Assim, é possível criar estratégias criativas que tenham significado para o aluno e ajudem-no a decidir como e o que vai integrar, diferentemente da aprendizagem passiva, onde a responsabilidade reincide exclusivamente sobre o docente.

Os recursos que a linguagem de programação podem oferecer para o processo de aprendizagem esta entrelaçada com o próprio percurso histórico da computação e as contribuições que diversos matemáticos forneceram ao longo dos séculos desde o surgimento dos primeiros computadores elétricos. Para evidenciar o quanto a linguagem de programação e a matemática estão interligadas vamos subdividir esse capítulo em tópicos que passaram pela história da computação na educação, a evolução da linguagem de programação até chegar à história do *Processing* aplicado principalmente no contexto da Matemática.

1.1 História da Computação na Educação

A história do computador remonta ao período em que o homem começou a sentir necessidade de contar as coisas, daí surgiu o nome computador do latim *computare*. A partir do processo de enumeração das coisas surgiu também a premência dos cálculos para agilizar suas tarefas e, a primeira máquina a executar esse processo foi o ábaco chinês que existe a pelo menos quatro mil anos e até hoje é utilizado em diversos países.

No entanto, somente no meado da década de 30 que surgiu o primeiro computador elétrico em plena Segunda Guerra Mundial. Sua principal função era realizar cálculos complexos num curto espaço de tempo e com a máxima precisão com o intuito de construir poderosas armas ou decifrar os códigos secretos do inimigo.

Após a guerra, tal recurso deixou de ser uma exclusividade do exército e passou a ser utilizado tanto nos negócios, como nas pesquisas industriais e universitárias. Mas, a utilização do computador na educação só iniciou-se juntamente com o início da comercialização dos primeiros computadores com capacidade de programação e armazenamento de informação nos anos 50, mesmo sem a existência de *softwares* apropriados [10, 11, 17].

No entanto, a incorporação dos computadores nas escolas efetivamente ocorreu nas décadas de 70 e 80, tanto nos Estados Unidos como nos demais países, intensificando-se com o surgimento dos microcomputadores na última década mencionada.

Nos Estados Unidos, em meados dos anos 70, existiam duas correntes de pensamento a respeito do uso do computador na educação: Uma foi desenvolvida por Patrick Suppes, o mentor dos programas de exercício CAI (*Computer Aided Instruction*) que consistia basicamente em programas dotados de gráficos dinâmicos onde o computador atuava como máquina de ensinar sob uma perspectiva de instrução conhecida como instrucionismo. O computador fazia perguntas e o aluno apresentava as respostas que julgavam

corretas.

E a outra corrente defendia o uso do computador como uma maneira de provocar mudanças na educação, sob uma perspectiva construcionista. A teoria construcionista desenvolvida por Seymour Papert foi baseada na teoria de Piaget e algumas ideias de Inteligência Artificial que se evidenciou simultaneamente com a linguagem de programação LOGO[2] criada pelo próprio Seymour Papert. Para esse matemático sul-africano o termo construcionismo apresentado em Valente [9, p. 40] serve para "mostrar um outro nível de construção do conhecimento: a construção do conhecimento que acontece quando o aluno elabora um objeto de seu interesse, como uma obra de arte, um relato de experiência ou um programa de computador". Com isso constatamos um diferenciador entre a perspectiva instrucionista e a construcionista.

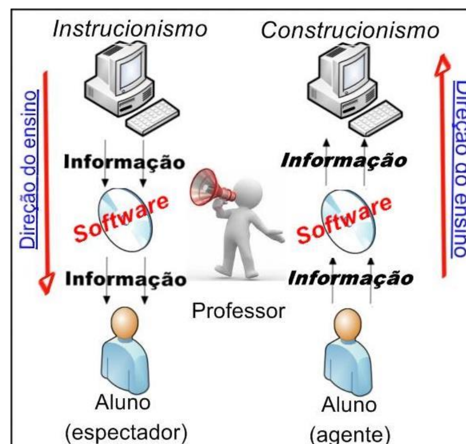


Figura 1: Correntes de ensino-aprendizagem usando o computador

A Figura 1 apresenta um comparativo entre as duas linhas de ensino-aprendizagem – instrucionismo e construcionismo – onde a presença do computador, do aluno, de um professor e de um *software* é constante em ambos os casos. No entanto, a direção do ensino, o tipo de *software* utilizado, a postura a ser adotada pelo docente e a caracterização do discente dentro do processo diferencia essas linhas de ensino-aprendizagem. Para o instrucionismo, o computador já vem pré-programado para ensinar o aluno (o espectador do processo) através de um *software* na modalidade CAI. Já, no construcionismo de Papert existe uma inversão, ou seja, é o discípulo que passa a ensinar ao computador a cumprir uma determinada tarefa a partir de uma postura ativa do mesmo. Isso é alcançado através de um *software*, que em nosso estudo é o *Processing*. Nota-se ainda que em ambos os casos,

[2]Logo é uma linguagem de programação voltada para o ambiente educacional que foi desenvolvido na década de 60 no MIT (Massachussets Institute of Tecnology), em Cambridge, Massachussets, Estados Unidos. Este *software* potencializa os docentes a trabalharem com os conteúdos de todas as áreas do conhecimento e níveis de escolaridade com o objetivo de retirar o aluno da passividade diante de um computador [18, 19].

a figura do professor mediador é fundamental para o processo de aprendizagem.

Constatou-se que os princípios de Seymour Papert foram bastante difundidos e defendidos por diversos países durante os anos 80 e início dos anos 90 como sendo o caminho para uma transformação absoluta na escola a partir da própria linguagem LOGO. Entretanto, tal transformação não ocorreu na concepção de Valente [10] em função dos diversos fatores, tais como a qualificação ineficaz dos professores.

O processo de incorporação dos computadores na educação, nos Estados Unidos, aconteceu de forma descentralizada e sem a interferência das ações governamentais até fazer parte do currículo a partir de 2000. No entanto, em outros países, como a França, Portugal e Brasil, a inserção desse recurso na educação aconteceu com a participação do sistema público que se articulou com as universidades para avaliar de que forma a integração dessa TIC aconteceria em todos os níveis do ensino [17].

A França foi o primeiro país ocidental que preocupou-se imediatamente com a formação de professores na incorporação da informática na educação além de estabelecer o público-alvo, materiais, *software*, meios de distribuição, instalação e manutenção dos equipamentos nas escolas, por considerar imperativo para uma inserção plena desse recurso educacional implantado na década de 70. Até, então, os *softwares* utilizados caracterizavam-se como EAO (Enseignemente Assisté par Ordinateur), equivalente ao CAI americano. Tais *softwares* eram programados com base na teoria comportamentalista e no condicionamento instrumental (estímulo-resposta). Contudo, na década seguinte passaram a utilizar a linguagem de programação e metodologia LOGO nos seus estudos com base nas ideias de Seymour Papert, o que se contrapós às bases conceituais do EAO [10, 11, 20, 19].

Desde 1985, a França preestabeleceu no seu plano nacional *Informatique pour Tous*[3] a disciplina de informática de caráter obrigatório e com orientação para o aluno ser ativo durante o processo de ensino-aprendizagem. E com o intuito de prevalecer esse princípio a sua reformulação em 2006 manteve esse ideal [17].

Apesar dos inúmeros projetos de informática na educação, a França, de acordo com Valente [10] não conseguiu obter êxito e nem provocar mudanças no sentido de romper o hábito milenar da educação do falar/ditar dos professores. Para Lévy [21, p. 5] uma das causas do insucesso esta no seguinte fato: "O governo escolheu material da pior qualidade, perpetuamente defeituoso, fracamente interativo, pouco adequado aos usos pedagógicos".

Já, Portugal iniciou o uso pedagógico do computador através do Projeto para a

[3]"Informática para Todos"

Introdução das Novas Tecnologias no Sistema Educativo mais conhecido como Relatório CARMONA que culminou com a implementação do projeto MINERVA[4], lançado em 1985, com o objetivo de introduzir a informática no ensino não superior de forma racionalizada para valorizar ativamente o sistema educacional em todas as instâncias e que permitisse uma dinâmica constante de avaliações e atualizações de soluções. A principal meta do projeto estava em preparar as escolas ao mesmo tempo que favorecia com a formação continuada de orientadores e professores para o uso da informática no ensino. A partir desse projeto foram desenvolvidos centros de pesquisa e formação de profissionais na área que contribuem para a inserção da computação no ensino das escolas de Portugal [17, 19].

A partir da ampliação do projeto Minerva, em 1996, foi criado o Programa NÓNIO – Século XXI – Programa de Tecnologias da Informação e da Comunicação na Educação, cujo nome foi em homenagem ao próprio inventor desse instrumento de medida de alta precisão, o matemático, geógrafo e pedagogo português Pedro Nunes (1502-1578). Com o intuito de implementar esse Programa foram criados uma rede de Centros de Competência distribuídos pelo país em instituições de ensino superior e em outras organizações educacionais para apoiar, orientar, acompanhar e avaliar os projetos das escolas nas dimensões técnica, teórico-prática e organizacional, onde possuíam indicação das escolas ao propor seus projetos ao Ministério da Educação [17]. Para Almeida [17] e Lemos Junior [19], um dos pontos de atuação dos Centros de Competência (o da Universidade do Minho) que mais se destacaram dentro do Programa NÓNIO foram o empenho e a dedicação dos professores e, o interesse e entusiasmo dos alunos confirmando com isso que a integração das TIC's no ensino assumiu o papel de catalisadora de mudanças na implementação de novas alternativas à educação e à prática pedagógica dos professores perfeitamente envolvidos no projeto e, especificamente, na mudança de atitude dos alunos ao incorporar as TIC no processo de aprendizagem e construção de conhecimentos, tanto na aula formal como nas situações de estudo individual e no desenvolvimento de seus trabalhos.

Portugal promoveu ainda em 2007 o Plano Tecnológico da Educação que representava uma proposta do governo para garantir o acesso ao computador portátil e à Internet banda larga orientada a professores do ensino básico e secundário, atual ensino fundamental e médio, com o objetivo de auxiliar os professores no uso individualizado e profissional das TIC's.

Os primeiros passos para a inserção da tecnologia digital no sistema educacional brasileiro aconteceu durante a década de 70 de forma similar ao ocorrido em Portugal e na França. Com o intuito de construir uma indústria própria, o governo brasileiro

[4]Meios Informáticos na Educação: Racionalizar, Valorizar, Atualizar

criou a Comissão Coordenadora das Atividades de Processamento Eletrônico (CAPRE), a Empresa Digital Brasileira (DIGIBRÁS) e também a Secretária Especial de Informática (SEI), que nasceu como orgão executivo do Conselho de Segurança Nacional da Presidência da República que até então era a ditadura militar [20]. Segundo Almeida [17] e Valente [10], foram realizados diversos seminários nacionais por decisão do governo federal e com a participação da comunidade científica da época. Dessa forma, as políticas de implementação e desenvolvimento da informática na educação deixou de ser produto apenas das decisões governamentais como ocorria na França e nem reflexo imediato do mercado como aconteceu nos Estados Unidos, ou seja, as deliberações e as propostas passaram a serem frutos de discussões e propostas feitas pela comunidade de técnicos e pesquisadores da área baseada nas próprias pesquisas concretizadas entre as universidades e escolas da rede pública. Com isso, a função do MEC passou a ser, desde então, a de acompanhar, viabilizar e implementar tais decisões.

A primeira instituição de ensino superior a utilizar o computador na educação foi a Universidade Federal do Rio de Janeiro (UFRJ), no início da década de 70, com o intuito de realizar simulações de reações químicas em cursos de pós-graduação de Química. Já em 1976 um grupo de pesquisadores da UNICAMP (Universidade Estadual de Campinas) resolveu visitar o Mit Media Lab, cujo retorno viabilizou a criação do grupo interdisciplinar formado por especialistas nas áreas de computação, linguística e psicologia educacional que alavancou as primeiras investigações sobre o uso do computador na educação com a linguagem LOGO impulsionando o movimento [20, 17]. A UNICAMP ainda foi responsável pela introdução da linguagem de programação BASIC no sistema educacional [19].

O MEC e a SEI juntamente com o Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) realizaram o "I Seminário Nacional de Informática na Educação", em Brasília no ano de 1981, com o intuito de discutir a conveniência ou não de se utilizar o computador como instrumento auxiliar no processo ensino-aprendizagem. No ano seguinte, a Universidade Federal da Bahia (UFBA) traçou uma linha política para implementar a informática na educação. Desses seminários surgiu o projeto EDUCOM, em 1983, implantado no âmbito da SEI com uma proposta de trabalho interdisciplinar destinado a criação de centros-piloto nas universidades brasileiras como instrumentos relevantes para a informatização da sociedade, objetivando a capacitação nacional e uma futura política setorial, compete aqui informar que as universidades brasileiras pioneiras nesse processo foram a UFRJ, a UNICAMP e a UFRGS (Universidade Federal do Rio Grande do Sul)[20, 22].

Para viabilizar o projeto EDUCOM foram estabelecidas as seguintes propostas, de acordo com Gonçalves [20, p. 13]:

> √ Sensibilizar e capacitar professores de 1º grau, interessados em uma prática pedagógica através do uso de computadores;
>
> √ Facilitar a divulgação de pesquisas e trabalhos realizados junto às comunidades de ensino de 2º e 3º graus [...];
>
> √ Divulgar técnicas e *software* educacionais necessários ao desenvolvimento de programas de ensino [...] para escolas, universidades e empresas interessadas;
>
> √ Estimular o desenvolvimento de teses, trabalhos e estágios na área;

Valente [10] afirma ainda que o papel do computador na educação é o de provocar as mudanças pedagógicas necessárias, invés de "automatizar o ensino" ou mesmo preparar o aluno para ser capaz de trabalhar com a informática. Dessa forma, o grande desafio estava na mudança da abordagem educacional, ou seja, transformar a educação centrada no aluno, na transformação da informação em uma educação onde o aluno possa realizar atividades por intermédio do computador e, consequentemente, aprender.

Para cumprir as propostas de uso pedagógico do computador nas escolas, o MEC criou o projeto FORMAR, em 1987, coordenado pela UNICAMP. Tal projeto contava com a participação dos pesquisadores e especialistas dos demais centros-piloto integrantes do projeto EDUCOM e o principal objetivo do programa estava no desenvolvimento de cursos de especialização na área de informática educativa para atuarem nos sistemas públicos de educação e assim se tornarem multiplicadores na formação de outros docentes em suas instituições de origem ao finalizar o curso. Contudo, apesar dos cursos contarem com os aplicativos CAI, linguagem de programação LOGO e outros programas básicos, o projeto FORMAR não teve muitos avanços no sentido de incorporar o computador nas atividades de sala de aula.

O MEC resolveu implementar, em 1989, o projeto PRONINFE (Programa Nacional de Informática na Educação) fundamentado nas ideias de Seymour Papert, ou seja, na teoria construcionista e em Paulo Freire com a prática da pedagogia crítico-reflexiva. A principal finalidade desse projeto consistia no desenvolvimento da Informática Educativa no Brasil, através de projetos e atividades amparados em fundamentação pedagógica sólida e atualizada. Tal programa tornou-se o principal referencial das ações atualmente planejadas pelo MEC que perdurou mais de uma década até o surgimento do Programa Nacional de Tecnologia Educacional (PROINFO), em 1997, com maior abrangência a nível nacional. No entanto, tal nomenclatura, PROINFO, só foi adotada através do Decreto nº 6.300, de 12 de dezembro de 2007 para atender a finalidade de incorporar o uso de todas as

mídias tecnológicas de informação e comunicação nas redes públicas de educação básica; melhorar a qualidade do processo de ensino e aprendizagem; proporcionar uma educação voltada para o desenvolvimento científico e tecnológico, e educar para uma cidadania global em uma sociedade tecnologicamente desenvolvida.

Salienta-se que tais mudanças pedagógicas foram o objetivo de todas as ações dos projetos de informática na educação. Todavia, os resultados obtidos não foram suficientes para sensibilizar ou alterar o sistema educacional como um todo até o presente momento. Mesmo fornecendo computadores portáteis aos alunos da rede pública de ensino e o oferecimento de cursos para a formação de professores mediante projetos escolares através do projeto Um Computador por Aluno (UCA), o uso de *tablets*, distribuição dos equipamentos tecnológicos nas escolas, recursos multimídia e digitais como ação do PROINFO.

1.2 Evolução da Linguagem de Programação

É notório que cada civilização ao longo da história possuía um tipo de linguagem diferente, assim como, cada país tem seu idioma que lhes permitem comunicar-se entre si. Com o computador não seria diferente, foi desenvolvido uma linguagem específica que programa os computadores para executarem uma determinada ordem, geralmente através de algoritmos. No entanto, a própria história nos revela que o povo da babilônia já possuía uma linguagem natural para escrever seus algoritmos e estes descreviam situações da vida cotidiana sem recorrer ao uso de variáveis. Desta forma, falar na evolução da linguagem de programação é analisar as contribuições do período da pré-história das linguagens, as primeiras linguagens e os paradigmas da programação com suas principais linguagens.

1.2.1 Pré-história da Linguagem de Programação

A história nos revela que Euclides de Alexandria (330–227 a.C.) se destacou não só por sua obra "os Elementos", esse matemático estruturou de forma sistemática o saber geométrico, implementando as ideias sobre axiomatização, de Aristóteles, para uma ciência exata [23]. Dentre as contribuições encontradas nessa obra encontra-se o algoritmo para calcular o MDC (Máximo Divisor Comum), ou simplesmente, algoritmo de Euclides que apareceu no livro VII com uma interpretação geométrica definida pela determinação da maior medida comum entre dois segmentos de reta [24]. De acordo com esta ideia de Euclides o aluno não precisaria recorrer a decomposição em fatores primos, utilizando

a interpretação geométrica do MDC do algoritmo de Euclides como segue no seu livro, conseguiríamos introduzir inclusive de forma prática este conteúdo nas turmas de 6º ano do ensino fundamental.

Proposição 1.1 (MDC de Euclides)**.** *Sendo dados dois números não primos entre si, achar a maior medida comum deles [24, p. 271].*

De acordo com esta proposição, dados dois segmentos AB e CD representando dois números não primos entre si, respectivamente, e diferentes de zero, existe um terceiro segmento EF que cabe um número inteiro de vezes nos outros dois segmentos, isto é, o segmento EF mensura os segmentos AB e CD, conforme podemos observar na Figura 2.

Um exemplo prático que explícita a ideia do algoritmo de Euclides pode ser obtido determinando o MDC entre 345 e 253. Como 253 não divide 345, então determinamos a diferença entre o maior número e o menor, e se o resultado não dividir o menor, então repetimos o procedimento até obtermos um resto que divida todos os restos antecedentes e os números 253 e 345. Sendo assim, temos:

$$345 - 253 = 92 \Longrightarrow 253 - 92 = 161 \Longrightarrow 161 - 92 = 69 \Longrightarrow 92 - 69 = 23$$

Como o número 23 divide os restos $69, 92$ e 161 e, os números 253 e 345, então o número 23 é o mdc entre 345 e 253. Tal concepção é encontrada segundo Oliveira [25], no livro História da Matemática, de Carl Boyer, na página 84, quando faz referência ao livro Elementos de Euclides e é apresentado da seguinte forma:

> Dados dois números diferentes, subtrai-se o menor a do maior b repetidamente até que se obtenha um resto r_1 menor do que o menor número; então subtrai-se repetidamente esse resto r_1 de a até resultar um resto r_2 menor do que r_1 então subtrai-se repetidamente r_2 de r_1 e assim por diante, finalmente, o processo leva a um resto r_n que mede r_{n-1}, portanto todos os restos precedentes, bem como a e b; esse número r_n será o máximo divisor comum de a e b.

Existe também outra forma de exemplificar geometricamente esta situação que é construindo um retângulo de medidas 345 e 253 (Figura 3). O objetivo é dividir o retângulo em quadrados iguais, com o maior lado possível, conforme a figura mostra. Para isso, dividi-se as medidas do lado maior pelo lado menor, a parte inteira determina o número de quadrados com lado igual ao menor. A medida restante é utilizada em nova divisão do menor lado com o residual anterior para obter novos quadrados e, assim por diante, quando a divisão em quadrados for exata, isto é, não sobrar nenhum retângulo, então a dimensão do menor quadrado será o MDC entre 345 e 253, que ainda é 23.

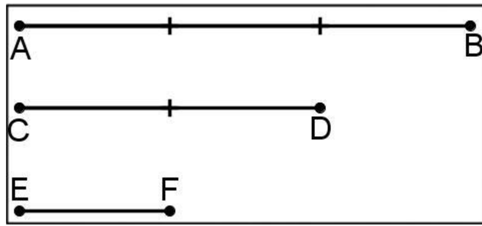


Figura 2: Segmentos de reta para o algoritmo de Euclides

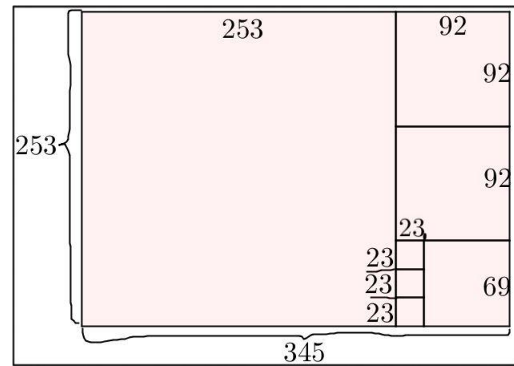


Figura 3: MDC de Euclides no retângulo

Através da programação o aluno poderá desenvolver um aplicativo que execute estes procedimentos ênuplas vezes até obter o MDC. Sem falar que tal concepção impulsionou muitos avanços na própria linguagem de programação.

Salienta-se ainda que Abu Ja'far Muhammad Ibn Musa Al'Khwarizmi (780–850), matemático e astrônomo persa, trouxe grandes avanços para a matemática e para a programação de computadores com a descoberta do sistema de numeração decimal e seus dez símbolos intitulados atualmente como algarismos indo-arábicos que acabou por influenciar fortemente a matemática na Europa Medieval.

Muitos outros matemáticos influenciaram ao longo da história, no entanto, foi através da Lógica Moderna iniciada no século XVII com o filósofo e matemático alemão Gottfried Wilhelm Leibniz (1646–1716) que influenciou, mesmo após 200 anos, vários ramos da Lógica Matemática Moderna e outras áreas relacionadas, como a Cibernética desenvolvida por Norbert Wiener. Com o ambicioso programa para a Lógica, Leibniz influenciou seus contemporâneos e sucessores, que visava criar uma linguagem universal baseada em um alfabeto do pensamento ou *characteristica universalis*, uma espécie de cálculo universal para o raciocínio. Na concepção desse matemático, tal linguagem universal deveria ser como a Álgebra ou como uma versão dos ideogramas chineses[5] [23].

Apesar dos suscetíveis erros que a calculadora de Leibniz possuía, sua invenção foi de fundamental importância para a história da programação por introduzir o conceito de operacionalizar as multiplicações e divisões através das adições e subtrações sucessivas, ou seja, sua máquina era capaz de realizar as 4 operações básicas.

Os ideais de Leibniz a respeito das máquinas para liberar o homem das tarefas repetitivas e de simples execução foi quase implantada pelo matemático e astrônomo inglês

[5]Os ideogramas chineses são uma coleção de sinais básicos que padronizassem noções simples não analíticas.

Charles Babbage (1792–1871), apreciado exclusivamente como um dos grandes desbravadores da era dos computadores. Esse matemático projetou, em 1822, um mecanismo feito de madeira e latão que alteraria a história por elaborar um dispositivo mecânico capaz de executar uma série de cálculos, no entanto, sua construção não foi efetivada por limitações tecnológicas da época.

Ressalta-se que o *Analitical Engine*, a Máquina Analítica – denominação da invenção de Charles Babbage – aproximava conceitualmente do atual computador. E por considerar de fundamental importância a impressão dos resultados, Charles ponderou que os resultados finais e os intermediários fossem impressos com o intuito de evitar erros, sendo necessário para isso dispositivos de entrada e saída. Segundo Fonseca [23], a entrada de dados na máquina seria feito por três tipos de cartões, são eles: “cartões de números” responsável por apresentar os números das constantes de um problema; “cartões diretivos” para controlar a movimentação dos números na máquina; e os “cartões de operação” para comandar a realização das operações tais como adições, subtrações e outras. Contudo, o mais impressionante na sua máquina estava nas duas inovações que trariam um grande impacto, tais inovações contou com a participação de Ada Augusta Byron[6], condessa de Lovelace. A primeira inovação estava no conceito de “transferência de controle” que permitiria à máquina comparar quantidades e, a depender dos resultados da comparação, poderia desviar para outra instrução ou sequência de instruções. A outra característica constava na possibilidade dos resultados calculados alterarem outros números e instruções colocadas na máquina, viabilizando com isso que o “computador” modificasse seu próprio sistema.

A inspiração de Babbage advém do francês Joseph-Mariae Jacquard (1752–1824) que introduziu o conceito de armazenamento de informações em placas perfuradas, para controlar uma máquina de tecelagem com o intuito de substituir o trabalho humano. O funcionamento dessa máquina de tecelagem através dos cartões perfurados determinava como a fiandeira deveria executar seu traçado, passando a linha por cima ou por baixo, quando repetir o processo, entre outros modelos de traçados. Dessa forma, a Máquina Analítica de Babbage poderia ser considerada como uma adaptação do tear de Jacquard, já que a mesma processava padrões algébricos da mesma maneira que o tear processava padrões de desenhos.

[6]Filha do famoso poeta Lord Byron e educada pelo matemático logicista inglês Augustus De Morgan. Ao ser apresentada a Babbage durante a primeira demonstração da Máquina de Diferenças percebeu o potencial alcance das novas invenções e passou a contribuir ativamente em seus trabalhos e por causa disso é considerada a primeira efetiva programadora de computadores, mesmo um século antes da existência do mesmo.

Em 1935, finalmente o computador passou por uma revolução, quando Alan Mathison Turing (1912–1954) tomou conhecimento do *Entscheidungsproblem* de Hilbert durante um curso ministrado pelo então matemático Max Neumann. Naquele momento Turing ainda era estudante do King's College, em Cambridge, e os resultados de Gödel e do problema da decisão motivaram-no inicialmente para a caracterização de quais funções são capazes de serem computadas[7]. A esse conjunto de esforços originou-se a fundamentação teórica da chamada "Ciência da Computação".

Turing definiu que os cálculos mentais consistem em operações para transformar números em uma série de *estados* intermediários que progridem de um para o outro de acordo com um conjunto limitado de regras, até que a resposta seja encontrada. Dessa forma, Turing concluiu que as regras matemáticas exigem definições mais rígidas do que as discussões no campo da metafísica sobre a mente humana e, a partir daí concentrou-se nesses estados objetivando utilizar para comandar as operações da máquina. Por causa de suas ideias, o governo inglês, em 1940, convocou-o para decifrar as mensagens codificadas pelo inimigo, a Alemanha, através de sua atuação na Escola de Cifras e Códigos. Quando a Segunda Guerra Mundial terminou sua contribuição tinha ajudado a construir um computador, o Colossus, uma máquina completamente eletrònica com 1.500 válvulas que moviam-se muito mais rápido do que a máquina Enigma da Alemanha [23].

O interesse de Turing pela programação das operações de um computador – chamada atualmente de *codificação* – contribuiu para a criação de linguagens de programação avançadas para o *hardware* do período. Para Turing, as máquinas poderiam modificar suas próprias operações a partir da elaboração de tabelas de instruções que convertem automaticamente a escrita decimal em dígitos binários para serem lidos pelas máquinas inicialmente construídas nas ideias da álgebra booleana. Na concepção desse matemático, os futuros programadores trabalhariam com linguagens consideradas atualmente como de *alto nível.* Nas palavras do próprio Turing apresentada por Rheingold em Fonseca [23]:

> As tabelas de instruções deverão ser feitas por matemáticos com experiência em computadores e certa habilidade em resolver problemas de solução mais difícil. Haverá bastante trabalho deste tipo a ser feito, se todo os processos conhecidos tiverem de ser convertidos na forma de tabelas de instruções em determinado momento. Esta tarefa seguirá paralelamente à construção da máquina, para evitar demoras entre o término desta e a produção de resultados. Poderão ocorrer atrasos, devido a virtuais obstáculos desconhecidos, até o ponto em que seja melhor deixar

[7]Os matemáticos da época buscavam um novo tipo de cálculo lógico que viabilizasse entre outras coisas, colocar em uma base matemática segura o conceito heurístico do que seja *proceder a um cômputo.* Tal resultado era importante para a matemática por determinar se é possível haver um procedimento efetivo para se solucionar os problemas de uma determinada classe bem definida[23].

> os obstáculos lá do que gastar tempo em projetar algo sem problemas (quantas décadas estas coisas levarão?). Este processo de elaboração de tabelas de instruções será fascinante.

Compete neste trecho validar a importância da álgebra booleana desenvolvida pelo matemático inglês George Boole (1815–1864), o fundador da Lógica Simbólica. Boole conseguiu desenvolver o primeiro sistema formal para o raciocínio lógico aplicando-o inclusive em diversos situações do cálculo formal e operações com regras formais sem considerar as noções primitivas. A sua contribuição foi marcante para a história da computação, tanto que sem ele, o caminho que interligou a Lógica à Matemática demoraria muito a ser construído. No que tange à Computação, a Máquina Analítica de Babbage seria apenas uma tentativa inspiradora, enquanto, sua álgebra booleana foi fundamental para que a tecnologia computacional chegasse com facilidade até a velocidade da eletrônica [23].

Na computação, seu ideal de um sistema matemático fundamentado apenas em duas quantidades, o 'Universo' e o 'Nada', indicados por '1' e '0', respectivamente, conduziu sua invenção do sistema de dois estados para a quantificação lógica que viabilizou a construção do primeiro computador, uma vez que seus construtores entenderam que o sistema com apenas dois valores seriam suficientes para compor os mecanismos para perfazer os cálculos. No entanto, somente após o desenvolvimento de quantificadores, introduzidos por Charles Sanders Peirce (1839–1914), que a lógica booleana parou de se limitar ao raciocínio proposicional e passou a ter uma lógica formal aplicada ao raciocínio matemático geral. A partir da implementação da Lógica à Matemática e dos avanços tecnológicos impulsionados pelas descobertas da época muitas outras linguagens de programação surgiram, juntamente com a evolução do computador.

1.2.2 Primeiras Linguagens de Programação

Os primeiros computadores mecânicos e eletromecânicos que surgiram a partir da década de 1930 estavam fundamentados na estrutura imaginada por Babbage que já poderia ser viabilizada pela tecnologia da época. Além desses computadores, muitos outros projetos de computadores eletrônicos concretizados posteriormente sofreram influências dessas primeiras máquinas. Contudo, foi Konrad Zuse (1910–1995) que desenvolveu a primeira máquina de cálculo controlada automaticamente. Na visão desse engenheiro civil, um dos aspectos mais onerosos ao se realizar longos cálculos com dispositivos mecânicos constava em guardar resultados intermediários para depois utilizá-los nos lugares desejados para os passos seguintes. De acordo com o pensamento de Zuse, em 1934, uma calculadora automática só necessita de três unidades básicas, são elas: uma controladora,

uma memória e um dispositivo de cálculo para a aritmética. A partir desse pensamento desenvolveu o Z1, em 1936, um computador construído inteiramente com peças mecânicas e uma fita de película cinematográfica para as instruções controlarem a máquina.

Depois do computador Z1, Zuse chegou a desenvolver os computadores Z2, Z3 e Z4. No entanto, assim que salvou o Z4 de um bombardeio e mudou-se para a pequena vila Hintesrtein nos Alpes, Zuse percebeu que não existia uma notação formal para a descrição de algoritmos e começou a desenvolver uma linguagem chamada de ***Plankalkül*** (*program calculus*). Essa linguagem consistia numa extensão do cálculo proposional e de predicado de Hilbert. Na visão de Zuse, o *Plankalkül* tinha como missão fornecer uma descrição formal pura de qualquer procedimento computacional. Contudo, tal linguaguem de programação não perdurou muito por não apresentar uma sintaxe amigável para expressar programas em um formato legível e facilmente editável apesar de possuir conceitos fundamentais para a programação: tipos de dados, estrutura de dados hierárquicos, atribuição, iteração, entre outros.

Salienta-se que depois do Teorema de Gödel e do projeto de Hilbert foi possível caracterizar o termo efetivamente computável através da máquina de Turing. Ou seja, foi possível tornar claro o que vem a ser um procedimento efetivo e um problema computável. Para Fonseca [23], um procedimento efetivo é uma sequência finita de instruções que podem ser executadas por um agente computacional, seja ele humano ou não. Tais propriedades apresentadas são:

I. A descrição deve ser finita;

II. Parte de um certo número de dados, pertencente a conjuntos específicos de objetos, e espera-se que produza um certo número de resultados que mantenham relação específica com os dados;

III. Supõe-se que exista um agente computacional (humano, eletrônico, mecânico, entre outros) que execute as instruções do procedimento;

IV. Cada instrução deve ser bem definida;

V. As instruções devem ser tão simples que poderiam ser executadas por alguém usando lápis e papel, em um espaço de tempo finito.

A esse procedimento efetivo dá-se o nome de *algoritmo*. Ou seja, o algoritmo[8] é uma sequência finita de passos para resolver um determinado problema. Cada des-

[8]Instruções no campo de entrada do programa

crição finita do algoritmo é realizada por uma determinada *linguagem*, essa linguagem algorítmica pertence a um subconjunto não ambíguo de uma linguagem natural, como o Francês, o Inglês ou Chinês, ou uma linguagem artificial construída para tal fim, como as linguagens de programação[9]. A forma ou formato desses procedimentos efetivos em uma linguagem algorítmica qualquer é específica de um conjunto de regras conhecido como *regras de sintaxe*, cujas propriedades foram enumeradas anteriormente.

Essa sintaxe corresponde a escrita correta e o relacionamento entre os símbolos e frases apresentadas nesses programas. Para alguns autores, segundo Fonseca [23], essas regras se dividem em concreta e abstrata. Sendo que a concreta abarca: o reconhecimento de textos (sequências de caracteres) coerentemente escritos de acordo com as especificações da linguagem e; a colocação dos textos, de forma não imprecisa, no interior das frases que concebe o programa. Enquanto, a sintaxe abstrata molda as estruturas de frases do programa. Consequentemente, a sintaxe define à *forma* dos programas executarem, de modo que as expressões, os comandos e as declarações possam ser sobrepostos para compor um programa. Dessa forma, a linguagem de programação passa a ter uma notação formal para a descrição de um algoritmo, sem as ambiguidades e as variabilidades de uma linguagem natural, que viabilize o rigor nas definições e demonstrações sobre os procedimentos. Além desses itens, a linguagem de programação precisa ser universal[10] e possuir uma semântica[11].

Diversos autores apresentam a linguagem de programação dividida de diferentes formas para classifica-las ao longo da história conhecidas como paradigmas de programação. Dentre elas evidencia-se a divisão dentro da seguinte taxonomia apresentada por Gudwin [26]:

- ⋆ Linguagens de Baixo Nível
- ⋆ Linguagens Não-Estruturadas
- ⋆ Linguagens Procedurais
- ⋆ Linguagens Funcionais

[9] Assembly, Fortran, Lisp, Algol, Cobol, Basic, Logo, Pascal, Forth, C, Smalltalk, SQL, Ada, C++, Perl, Java, Python, Ruby, JavaScript, PHP, Delphi, C#, VB.NET, entre outros ao longo da história.

[10] A linguagem é universal quando qualquer problema cuja solução possa ser encontrada através de um computador, ou seja, é qualquer linguagem que possa ser definida por uma função recursiva

[11] A semântica depende exclusivamente do efeito que se deseja causar quando o programa for executado por uma agente computacional, eletrônico ou não, isto é, é um conjunto de regras que determinam a ordem na qual as operações do programa irão ser executadas, quais serão executadas primeiro e quando se encerrarão.

⋆ Linguagens Orientadas a Objeto

⋆ Linguagens Específicas a Aplicações

⋆ Linguagens Visuais

As linguagens de programação induzem os alunos a pensarem no que está sendo proposto pelo computador e assim, resulte numa interação entre o computador e o aluno. No entanto, para o sucesso desse processo é preciso basicamente quatro ingredientes apresentado por Valente [9], são eles: o computador, o *software* educativo, o professor capacitado para usar o computador como meio educacional e o aluno. Ao mesmo tempo, Valente [10] afirma:

> As linguagens de programação têm mais recursos, enquanto os outros *software* como os tutoriais, as multimídias já prontas, os processadores de texto, não têm capacidade para executar o que o aprendiz está pensando e, portanto, não fornecem um *feedback* que seja útil para ele compreender o que faz.

Dessa forma, o uso de computadores e das linguagens de programação auxilia o aprendente a construir o conhecimento e a compreender o que faz, constituindo assim uma verdadeira revolução no processo de aprendizagem. Portanto, conhecer os tipos de linguagens e suas características é de fundamental importância para um docente antes de iniciar um programa direcionado à aprendizagem de seus alunos com alguma linguagem de programação, em função disso, descreveremos brevemente cada uma delas com as principais linguagens de programação.

1.2.2.1 Linguagens de Baixo Nível

São linguagens cujas instruções se aproximam diretamente da linguagem de máquina que é enviado ao processador para executar uma tarefa. Nesse sentido, existem diversas linguagens de baixo nível para cada processador diferente e o conjunto de linguagens desse tipo é conhecido de forma geral como "**Linguagem *Assembly***" [26].

1.2.2.2 Linguagens Não-Estruturadas

As linguagens não-estruturadas são mais aprimoradas do que as linguagens de baixo nível, uma vez que seus comandos não encontram-se vinculados ao processador e sistema utilizados, viabilizando com isso sua utilização em diversas plataformas. Com esta evolução, sua semântica passou a ser mais genérica, consentindo ao programador

executar operações mais sofisticadas que são emuladas por sequência de instruções mais simples em linguagem de máquina. No entanto, apesar de representar um grande salto qualitativo quando comparado com as linguagens de baixo nível, tal linguagem tornou-se paulatinamente obsoleta com o surgimento das linguagens procedurais, as funcionais e as orientadas a objetos [26]. Nesse tipo de linguagem destaca-se as linguagens **COBOL** (COmmon Business Oriented Language) e **BASIC** (Beginners All-purpose Symbolic Instruction Code).

1.2.2.3 Linguagens Procedurais

São linguagens de alto nível conhecidas por serem um sub-tipo das linguagens "estruturadas", já que, essas linguagens se opõe as não-estruturadas por apresentar estruturas de controle que permitem, entre outras coisas, o teste de condições (*if-then-else*), controle de repetição de blocos de código (*for, while, do*), fazem a seleção de alternativas (*switch, case*) e dividem o código do programa em módulos intitulados também de funções ou procedimentos [26]. As linguagens procedurais são caracterizadas pela existência de uma sequência de chamada de procedimentos definida através de seus algoritmos. Esse grupo de linguagens é representada pelos programas em **C**, **Pascal**, **Fortran**, **Algol** e **Forth**, entretanto, algumas linguagens mais sofisticadas foram desenvolvidas para corrigir deficiências e explorar novos contextos onde C, Pascal e Fortran apresentaram-se de forma ineficiente, são elas: **Ada**, **Modula-2** e **Modula-3**.

1.2.2.4 Linguagens Funcionais

As linguagens ditas funcionais diferem das linguagens procedurais por evidenciar uma programação funcional onde a ênfase está na avaliação de expressões ao invés da execução de comandos. As expressões nessas linguagens são formadas utilizando-se funções para combinar valores binários. Dentre as linguagens funcionais mais conhecidas encontram-se a **LISP** (LISt Processor) e a **Prolog**, todavia, a linguagem **Scheme** é frequentemente mencionada por ser uma variante simplificada da LISP. Além dessas linguagens funcionais existem na lituratura outras, tais como: Aspect, Caml, Clean, Erlang, FP, Gofer, Haskell, Hope, Hugs, Id, IFP, J, Miranda, ML, NESL, OPAL e Sisal [26].

Embora esses recursos existentes contribuíssem significativamente para o avanço científico e tecnológico da época, ainda existia a necessidade de se obter uma linguagem que explorasse ao máximo o potencial das novas tecnologias recém inventadas, além da própria crise de *software* no final da década de 60 que necessitava de uma linguagem mais

confiável para resolver sistemas complexos e com baixo custo na produção e manutenção caso houvesse necessidade. Diante desse contexto que surgiu às linguagens orientadas a objeto que abordaremos a seguir.

1.2.2.5 Linguagens Orientadas a Objeto

A partir da necessidade de se organizar o processo de programação de uma linguagem que surgiu a programação orientada a objetos que por si só é uma técnica. Antes do surgimento dessa linguagem, os paradigmas da engenharia de *software* derivavam os módulos basicamente da funcionalidade de um sistema, tais módulos correspondiam essencialmente aos módulos procedimentais que eram alimentados por dados e geravam novos dados na operação. Com a chegada do paradigma de orientação a objeto essa concepção foi modificada, idealizando os objetos como módulos que se comunicam por meio de mensagens, encapsulando ao mesmo tempo dados e funções, por meio de um mecanismo conhecido como tipo de dados abstratos [26].

Dentre as linguagens de programação orientada a objetos a primeira que se tem notícia foi o Simula, desenvolvida em 1967. Em seguida, veio o Smalltalk, em 1970. Atualmente, existe uma diversidade de linguagens orientadas a objeto, abrangendo desde linguagens de propósito geral, até linguangens para multimídia e programação em lógica como aponta Gudwin [26], onde destacam-se as seguintes linguagens: **Simula**, **Smalltalk**, **C++**, **Objective-C**, **Java**, **Eiffel**, **Python** e **Ruby**

1.2.2.6 Linguagens Específicas a Aplicações

As linguagens específicas foram desenvolvidas para atenderem algumas necessidades da época, dentre elas destacam-se às linguagens para bancos de dados, Simulação, Scripts e Formatação de textos. Dentre essas linguagens específicas destacamos o **MATLAB** (MATrix LABoratory) que é a linguagem de simulação com maior destaque nas aplicações científicas.

1.2.2.7 Linguagens Visuais

As linguagens de programação visual, segundo Gudwin [26], provém do princípio de que os gráficos são mais inteligíveis do que os textos. Dessa forma, a criação de um programa por meio de diagramas e outros recursos gráficos tendem a tornar a própria programação ainda mais fácil, possibilitando que os usuários sem muitas habilidades em

programação gerem seus próprios programas.

Nessa concepção, uma representação gráfica de um programa tenciona a ser uma descrição de alto nível para o funcionamento do programa e esse tipo de representação normalmente limita bastante a flexibilidade dos programas que podem ser desenvolvidos, no entanto, permite a rápida elaboração de programas com mais facilidade num escopo limitado de opções. Sendo assim, as aplicações principais para a programação visual se limitam a um campo bem específico evitando propósitos gerais.

Dentro dessa ótica, as linguagens de programação visual podem ser subdividas em:

- ⋆ Ferramentas de Programação Visual exemplificada pelos **Toolkits**.
- ⋆ Linguagens de Programação Visual Híbrida representada pelas linguagens **Delphi**, **Visual Basic** e **Visual C++**.
- ⋆ Linguagem de Programação Visual Pura representada pela programação **Khoros**, **Simulink** e ***Processing***.

Sendo que essa última linguagem, é gratuita e de código aberto desenvolvida para artes eletrónicas e projetos visuais cujo foco primordial volta-se para o ensino de noções básicas de programação de computador em contexto visual. Por este motivo, a linguagem de programação *Processing* foi escolhida como nosso objeto de estudo que abordaremos com mais detalhes a partir da próxima secção.

1.3 História do *Processing*

O ambiente ***Processing*** é derivado da linguagem Java e foi projetado inicialmente, em 2001, por Casey Reas e Ben Fry, ambos ex-membros do Grupo de Computação do MIT Media Lab. Por ser uma linguagem de programação utilizada em contexto visual, muitos usuários são primordialmente artistas, estudantes e pesquisadores e, por possuir uma facilidade na instalação, uso e aprendizagem agregada à produtividade de programas que utilizem interações complexas, faz com que *Processing* seja ideal para a prototipação rápida de ideias e como caderno de esboços em *software*.

Outro ponto importante do *Processing* em relação as outras linguagens consta no fato da sua IDE incluir um *sketchbook* como alternativa para organizar projetos sem adotar o mesmo padrão da maioria das IDEs e seus programas chamados de Sketches são uma subclasse do Java Papplet que é uma classe implementada da maioria das funcionalidades

da *Processing*. Além disso, todo o código do *sketch* é traduzido para o Java ao ser compilado. Atualmente, o *Processing* possui uma enorme comunidade de usuários criativos em virtude da gratuitidade e por rodar na maioria dos sistemas operacionais, além da possibilidade de gerar gráficos em 2D, em 3D, de processar mídias digitais audiovisuais, geração de arquivos PDF, exibição de imagens SVG, qualquer biblioteca existente para a linguagem Java, e, ainda integrar com outros ambientes ou interfaces como o Arduino e Kinect.

Com o *Processing*, o programador não precisa possuir domínio da sintaxe de programação para iniciar sua aprendizagem e com poucos comandos é possível escrever um aplicativo que geralmente em outras linguagens são necessárias diversas linhas de instruções que exige um domínio da sintaxe. Para um aluno exibir uma simples mensagem de *"Bom dia, Pessoal!!!"* com o desenho de uma elipse na mesma tela é necessário escrever apenas as seguintes declarações da tabela 1 que resulta na construção da Figura 4.

Tabela 1: Tabela com os comandos *textFont* e *ellipse*

Instruções	Comandos
Tamanho da tela	*size*(300, 250);
Plano de fundo	*background*(0);
Carrega variável fonte	*PFont fonte*;
Escolhe o tipo de fonte	*fonte* = *loadFont*("*MonotypeCorsiva* − 50.*vlw*");
Ativa a fonte e o seu tamanho	*textFont*(*fonte*, 38);
Escreve a mensagem na tela	*text*("*Bom Dia, Pessoal*!!!", 20, 50);
Constrói uma elipse	*ellipse*(150, 150, 160, 120);

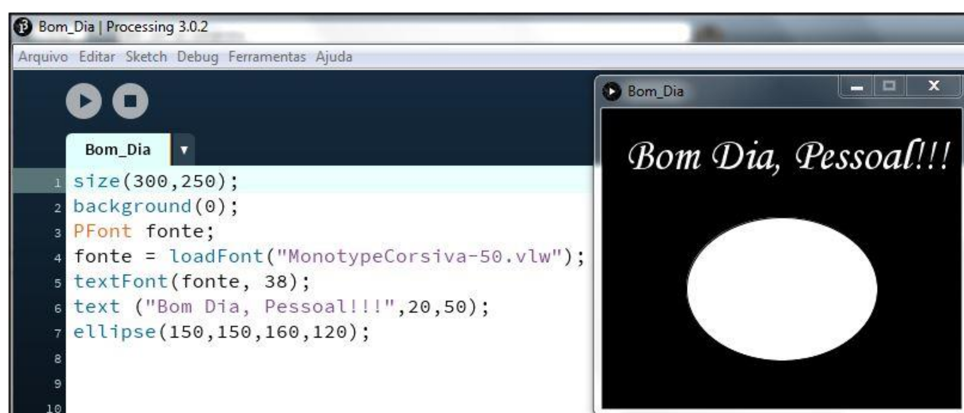


Figura 4: Comando para exibição de mensagens e elipse

Esse aspecto didático do *Processing* não inviabiliza sua aplicação em programas reais que exijam alto desempenho em processamento gráfico e/ou que exijam portabilidade para diversas plataformas de *software*. Em função disso, é considerada uma ótima ferramenta para não-programadores ou iniciantes em programação por apresentar uma

sintaxe simplificada e uma satisfação imediata com um retorno visual como apresentamos nas Figuras 4. Com isso, reafirmamos a importância dessa linguagem para inserir os alunos do Ensino Fundamental e Médio no contexto das linguagens de programação para iniciarmos a descrição da própria janela (IDE) e comandos do *Processing* no próximo capítulo.

Com os avanços tecnológicos ainda foi possível incrementar essa linguagem no sistema de Javascript com o intuito de implementar os trabalhos usando os padrões web e sem nenhum plugin a partir das contribuições de John Resig. Dessa forma, surgiu o *Processing* que é um projeto irmão daquele criado inicialmente por Ben Fry e Casey Reas mantendo a potencialidade do programa original, ou seja, ainda é possível visualizar seus dados, uma arte digital, animações interativas, gráficos educativos, jogos, entre outros. A proposta do *Processing* é elevar o nível de compreensão e visualização das artes eletrónicas com a permissividade do código de processamento ser executado por qualquer navegador compatível com HTML5, incluindo as versões atuais do Firefox, Safari, Chrome, Opera e Internet Explorer.

As versões de *Processing* já conseguem incorporar esta funcionalidade ao instalar um aplicativo em JavaScript pelo próprio programa e assim, viabilizar uma construção tanto em *Processing* como em *Processing.JS*.

Em detrimento disso, as construções em *Processing* ainda será realizada no *Processing* e para os alunos que não possuem acesso ao computador ou notebook em suas residências para exercitar e desenvolver algumas atividades ainda é possível utilizar um aplicativo desenvolvido para Android, o APDE ou o An Editor for *Processing*, gratuitos e com a mesma interatividade daquele encontrado no computador sem os recursos da barra de menus. No caso, dos alunos desejarem divulgar suas construções na *internet*, é possível criar uma conta no *OpenProcessing*[12] para copiar seus códigos ou digita-los diretamente nessa versão *online* do *Processing.JS*.

[12]No endereço eletrônico: *http://www.openprocessing.org*

2 *Descrição da Linguagem Processing*

A utilização do recurso computacional em Matemática para explorar diversos temas matemáticos fundamentando a teoria com a prática tem-se tornado imprescindível para novas descobertas e conjeturas viabilizando uma aprendizagem significativa e contextualizada. Nossa proposta é utilizar a linguagem de programação para fortalecer e desenvolver os conceitos de transformação geométrica no plano como forma de abordagem em problemas que apresentam padrões de deslocamento de figuras planas, tais como a análise gráfica de funções, entre outras aplicações.

Usaremos a linguagem de programação *Processing* compilado para JavaScript que está disponível gratuitamente no endereço *https://github.com/processing/processing/releases* para as plataformas Windows, Linux e Mac OS.

Como suporte de estudo, usaremos como tutorial os seguintes materiais:

- ⋆ Vídeo de Introdução a Linguagem de Programação *Processing* do Professor Cláudio Luís Vieira Oliveira apresentado na Semana de Tecnologia 2014 (FATEC Jundiaí), disponível no youtube https://www.youtube.com/watch?v=pSHRk0B9SwI.
- ⋆ O curso de Programação oferecido no site da Khan Academy.
- ⋆ O material do Workshop *Processing*, desenvolvido pelo Professor Claúdio Esperança, disponível no endereço *http://www.lcg.ufrj.br/Members/esperanc/workshopvis*.
- ⋆ Os vídeos de Introdução a Programação com Jogos em *Processing*, desenvolvido pelo grupo Sputnik Learning Space, disponível no endereço *https://www.youtube.com/channel/UCRltB7VDN-2oLilZBXT8rMg*.
- ⋆ O livro *Processing: a programming handbook for visual designers and artists* do próprio autor, Ben Fry e Casey Reas [27], da linguagem *Processing* com tradução nossa para aplicar nas construções.

Esse capítulo não possui a menção de ser utilizado como um curso de *Processing*, no entanto, objetiva-se apresentar as principais ferramentas básicas para iniciar uma produção nessa linguagem para viabilizar a importância de recorrer a tal recurso para o ensino das TGP. Sua linguagem potencializa a visualização e manipulação dos objetos geométricos ao mesmo tempo que insere os alunos no âmbito da programação, possibilitando uma formação profissional para os interessados em dar continuidade no programa.

2.1 Comandos Básicos e Menu

Essa linguagem de programação possui uma IDE tradicional com menus, uma barra de ferramentas, um editor de textos com abas, painéis para mensagens e para saída textual como segue na Figura 5(a), e na versão atualizada já existe um item específico para JavaScript. A documentação dos arquivos gerados pelo aplicativo é chamado de PDE (*Processing Development Environment*) e os programas construídos nessa linguagem são chamados de *sketches*, como na Figura 5(b), e podem ser compostos de diversos arquivos de código e outros recursos como imagens, fontes, bibliotecas, etc. Cada *sketches* é guardado em um subdiretório de uma pasta intitulada de *sketbook*.

(a) Tela de Comandos

(b) Tela de exibição

Figura 5: IDE e aplicação da elipse animada

Na barra de ferramentas são exibidos seis comandos básicos para o processamento, a depender da versão: Start server, Stop server, New, Open, Save, Export for Web.

⋆ **Start server**: Executa o código (compila o código, abre a janela de exibição e

executa o programa dentro);

* **Stop server**: Encerra um programa em execução, mas não fecha a janela de exibição;
* **New**: Cria um novo desenho (projeto);
* **Open**: Selecionar e carregar um esboço pré-existente;
* **Save**: Salva o desenho atual em seu local atual;
* **Export for Web**: Exporta o desenho atual para o *sketchbook* como um Applet Java incorporado em um arquivo *HTML*. O diretório que contém os arquivos é aberto. Ao clicar no arquivo index.html o programa carrega o *software* no navegador padrão.

Além dessa barra de ferramenta existe na barra do menu seis comandos adicionais: Arquivo, Editar, Sketch, Debug, Ferramentas e Ajuda.

As coordenadas cartesianas na maioria das linguagens de programação incluindo o *Processing*, possuem sua origem no canto superior esquerdo da janela, onde as coordenadas do eixo x aumentam para a direita e as coordenadas do eixo y aumentam a partir do topo. Ressalva-se ainda que cada coordenada mapeia diretamente a posição do objeto em pixels. Por exemplo, se o seu programa possui 300 por 240 pixels de largura e altura, respectivamente, então a coordenada (0, 0) é o pixel superior esquerdo, enquanto a coordenada (300, 240) está no canto inferior direito, onde a última coordenada visível no canto inferior direito da tela está na posição (299, 239), ver Figura 6.

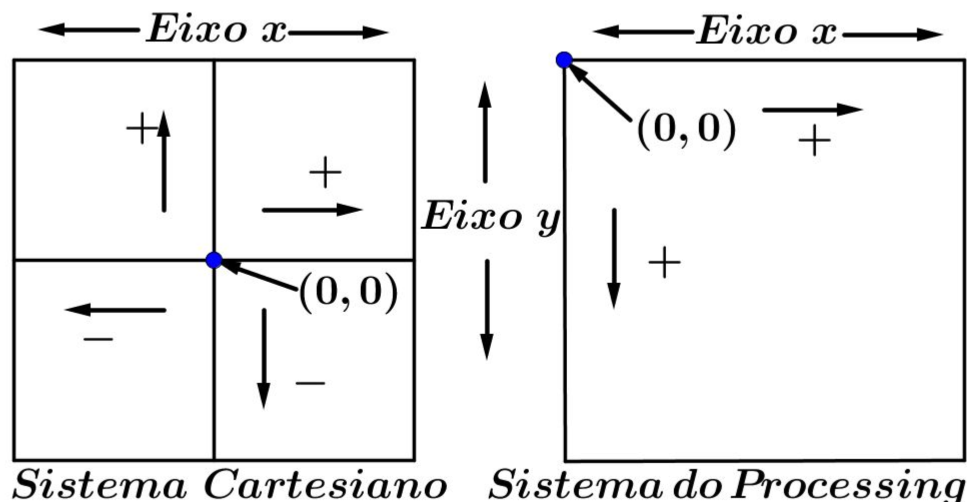


Figura 6: Sistema de Coordenadas Cartesianas e no *Processing*

O *Processing* ainda permite simular desenho em três dimensões que não será abordado nesse estudo. De acordo com Amado [28], essa linguagem permite que o usuário programe em três níveis de complexidade: o modo básico, o modo contínuo e o modo de Java. No entanto, o mesmo aconselha iniciar a programação no modo básico com o intuito de incrementar o conhecimento sobre coordenadas, variáveis e loops antes de se aventurar nos modos contínuos e Java. Diante desse exposto, Amado descreve cada nível conforme mostra a tabela 2.

Tabela 2: Caractéristicas do grau de complexibilidade da programação *Processing*

Modo Básico	Esse modo é utilizado para desenhar objetos simples no ecrã e implementar os fundamentos da programação. As linhas simples de código têm uma representação direta na tela.
Modo Contínuo	Esse modo fornece estrutura ao programa e permite desenhar de forma contínua, atualizar e introduzir interatividade mais complexa.
Java	Esse modo é o mais flexível, permitindo que os programas Java sejam escritos completamente dentro do ambiente *Processing*. Escrevendo no modo Java remove as limitações das bibliotecas de processamento e dá acesso à linguagem de programação Java completo.

Dentro das construções existem alguns comandos que são inerentes ao grupo do Modo Básico e outras do Modo Contínuo como revela a tabela 3 que apresenta o comando com sua descrição.

Tabela 3: Descrição de alguns comandos inerentes aos Modos Básico e Contínuo da programação *Processing*

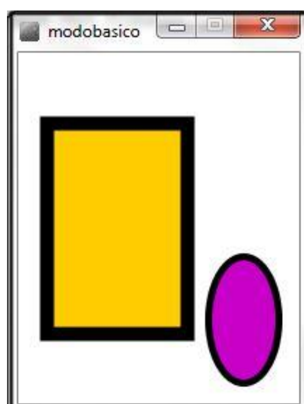
Modos	Comandos	Descrições
Básico	size();	Define a resolução da janela ("Tamanho").
	background();	Define a cor do fundo e limpa a tela.
	fill();	Escolhe a cor usada para preenchimento das formas.
	stroke();	Específica a cor de traçado de linhas.
	strokeWeight();	Determina a largura do traçado em pixels.
	smooth();	Específica a técnica de antiserrilhamento que deve ser utilizada para suavizar o traço.
	point(x, y);	Marca pontos.
	line(x_1, y_1, x_2, y_2);	Desenha segmento de reta.
	rect(x, y, w, h);	Desenha retângulo.
	ellipse(x, y, w, h);	Desenha elipse.
	triangle$(x_1, y_1, x_2, y_2, x_3, y_3)$	Desenha triângulo.
	quad$(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$;	Desenha quadrilátero.
	bezier$(x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4)$;	Curvas Bézier cúbicas.
	println();	Envia texto para área de console do *Processing* e pula uma linha.
Contínuo	setup();	Função chamada quando inicia a execução do programa.
	draw();	Função chamada imediatamente após a execução do setup, loop de repetição para o desenho de forma contínua.
	frameRate();	Valor aproximado de taxa de atualização máxima de frames na execução do programa em modo contínuo.
	frameCount();	No modo contínuo, é um número inteiro que contém o número de frames mostrados desde o início da execução do programa. Cada chamada ao método draw incrementa este valor.
	loop();	Executa a função void draw(); uma única vez.
	noLoop();	Impede o void draw(); de executar em loop;
	redraw();	Redesenha as formas quando chamado em outros eventos fora da draw.

A seguir apresentaremos uma construção considerando o Modo Básico e o Modo

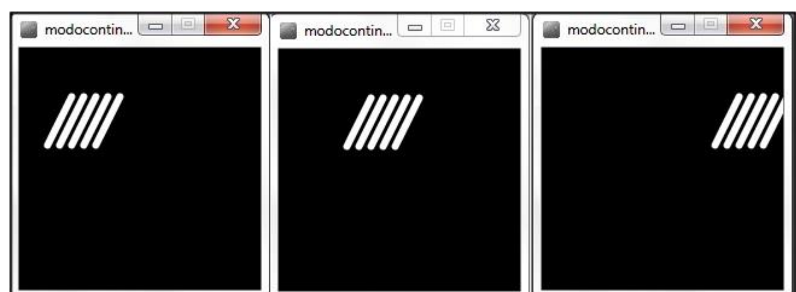
Contínuo de acordo com os códigos de comando constantes na tabela 4. O resultado dessa construção no Modo Básico e no Modo Contínuo encontra-se presente nas imagens das Figuras 7(a) e 7(b), respectivamente.

Tabela 4: Comando das construções iniciais no Modo Básico e no Modo Contínuo

Modo Básico	**Comentários**
size(200,250);	//Tamanho da tela 200 por 250 pixels de largura e altura
background(255);	//Cor de fundo da tela 255 = branca
fill(255,204,0);	//Preenchimento da forma amarelo
stroke(0);	//Cor de traçado 0 = preta
strokeWeight(10);	//Largura do traçado
smooth();	//Arestas mais suaves
rect(20, 50, 100, 150);	//Retângulo de coordenadas (20,50) com largura 100, comprimento 150
strokeWeight(5);	//Largura do traçado
fill(200,0,200);	//Preenchimento da forma rosa
ellipse(160, 190, 50, 90);	//Elipse de centro (160,190) com largura 50, comprimento 90
Modo Contínuo	**Comentários**
int $x0 = 10$; //Chamada da variável $x0$	
int $y0 = 80$; //Chamada da variável $y0$	
void setup(); {	//Função chamada quando inicia a execução do programa.
size(200,200);	// Tamanho da tela
stroke(255);	//Cor de traçado branca
strokeWeight(6);	//Largura do traçado
smooth();	//Arestas mais suaves
}	//Encerrar o comando void setup()
void draw();{	//Inicia o comando de repetição.
background(0);	//Fundo da tela preto
for (int $i = 0; i < 5; i++$){	
int $x = x0 + i * 10$;	
line($x, y0, x + 20, y0 - 40$);	
}	//Encerrar o comando for
$x0 += 1$;	//Move desenho para a direita
if ($x0 >= 200$) $x0 = 0$;	
}	//Encerrar o comando void draw()



(a) Objetos no Modo Básico



(b) Segmentos animados no Modo Contínuo

Figura 7: Construções no Modo Básico e Modo Contínuo

De acordo com Amado [28] o livro de Semiologia e criação Gráfica de Bertin com aplicação à construção de programas apresenta a parametrização do problema bem como a

criação de um conjunto de instruções a serem seguidas por quem irá executar o programa. Deste ponto de vista, o autor supracitado apresenta um conjunto de 7 parâmetros que condicionam ou compõe todo tipo de criações gráficas, são eles:

1 Posição;
2 Tamanho;
3 Valor;
4 Textura;
5 Cor;
6 Orientação;
7 Forma;

No entanto, os meios digitais incrementam pelo menos mais 7 parâmetros que são listados por Amado [28] como:

1 Movimento;
2 Crescimento;
3 Adição(aumento);
4 *Moiré*;
5 Espectro;
6 Rotação;
7 Mutação;

2.2 Variáveis e Tipos de dados

Para Amado [28], a memória é um importante conceito em algoritmia, pois esta permite armazenar resultados intermediários do algoritmo, e a esses elementos de memória utilizados num algoritmo que designamos por variáveis. As variáveis são as "gavetas" ou "contentores" onde serão guardados todos os tipos de elementos para serem reutilizados pelo programa em outra ocasião. Para cada variável existe um identificador que nomeia para ser localizado na memória do computador, tal identificação precisa ser contínua e não pode começar por números. Sendo assim, é preciso informar ao programa qual "gaveta" de um certo tipo de dados será utilizado declarando sua variável que diferentemente do Flash ou PHP, o *Processing* necessita de uma certa indicação do tipo de dados que a variável vai armazenar - números inteiros, reais, texto, entre outros.

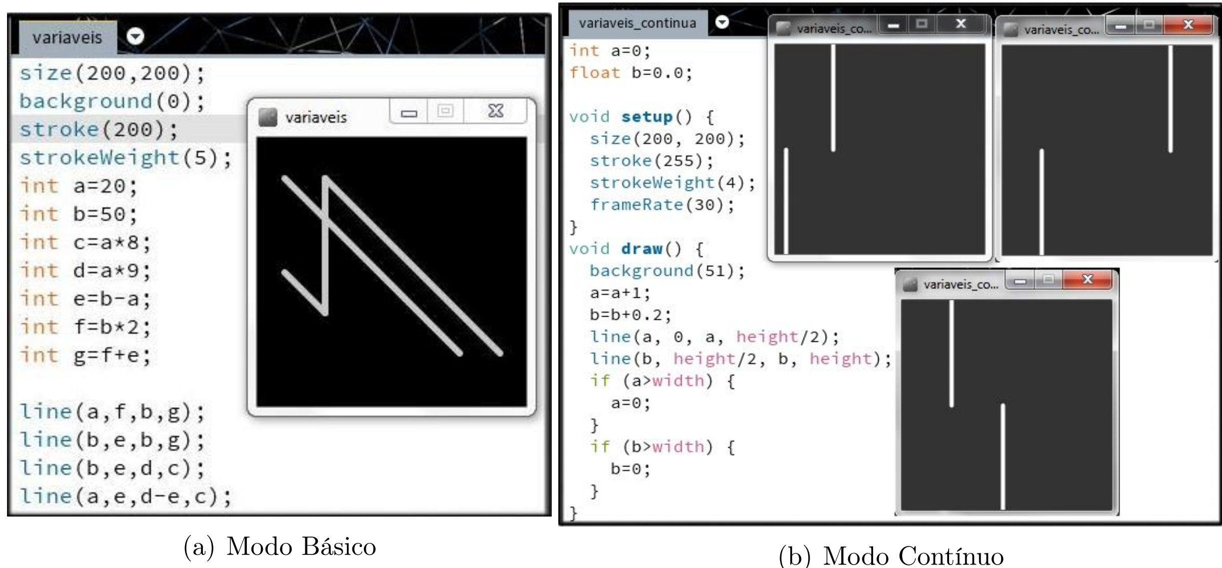
As variáveis são declaradas de acordo com seu tipo, dentre elas destacam-se os tipos primitivos **int**, **float** e **String**. A tabela 5 apresenta a característica básica de algumas variáveis que podem ser visualizadas na construção das Figuras 8(a), no Modo Básico, e 8(b), no Modo Contínuo.

Os dados ainda podem ser classificados como tipos complexos que são tipos compostos formados por vários elementos simples como aponta Amado [28]:

- **Vector** ou *Array*, em inglês, é uma lista de elementos do mesmo tipo que podem ser ativados via um índice;
- **Matriz** é um vector multi-dimensional;

Tabela 5: Declarações dos tipos de variáveis

Keyword	**Descrição**	**Tamanho/Gama**
		Números Inteiros
byte	Byte	8-bit (-128 a 127)
short	Inteiro curto	16-bit (-32768 a 32767)
int	Inteiro	32-bit (-2147483648 a 2147483647)
long	Inteiro longo	64-bit (-9223372036854775808 a ...807)
		Números Reais
float	Vírgula flutuante	32-bit IEEE 754 (1.401298...707e^{45} a 3.402823...860e^{+38}



(a) Modo Básico (b) Modo Contínuo

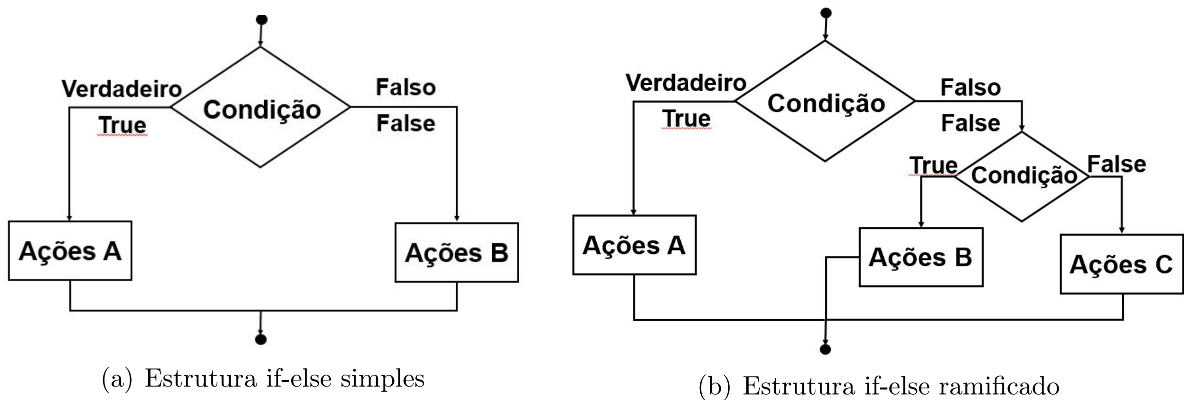
Figura 8: Declarações de variáveis

- **Estrutura** é a agregação de vários tipos de dados;

2.3 Estrutura condicional e operadores lógicos

As ações ou decisões tomadas em função de determinadas condições são uma parte importante nas nossas vidas. Na programação não é diferente, é preciso tomar decisões e executar ações condicionalmente, e a esse processo chamamos de *Controle de Fluxo*. O controle de fluxo do programa é devido essencialmente a duas estruturas: **if** e **switch**.

A forma mais elementar de uma estrutura condicional encontra-se na estrutura if-else. Essa estrutura consiste basicamente se sim - fazemos uma ação A - se não - fazemos uma ação B, conforme ilustra o fluxograma da Figura 9(a). No entanto, essa estrutura pode gerar ramificações bastante complexas a depender do número de condições a serem verificadas ou ainda do número de ramificações de cada condição ou sub-condições conforme o fluxograma da Figura 9(b).



(a) Estrutura if-else simples

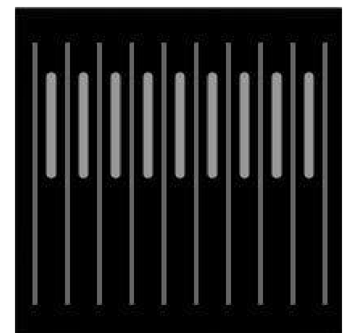
(b) Estrutura if-else ramificado

Figura 9: Diagrama de uma estrutura if-else

Um exemplo de construção utilizando a condicional if-else esta na construção de segmentos obedecendo a seguinte condição: Se "i" divide por 20 sem resto, então desenhar a primeira linha maior, do contrário, desenhar a segunda linha menor e o resultado disso é a imagem da Figura 10(b) a partir do comando 10(a). Entretanto, se houver três ou mais condições a verificar é recomendado usar a estrutura switch conforme mostra o fluxograma da Figura 11(a) constata na construção da Figura 11(b)

```
size(200, 200);
background(0);
for (int i=10; i<width; i+=10) {
  // if: Se 'i' divide por 20 sem resto desenhar a primeira linha.
  // Else: Se não divide então desenhar uma segunda linha.
  if (i%20==0) {
    stroke(153);
    strokeWeight(6);
    line(i, 40, i, height/2);
  } else {
    stroke(102);
    strokeWeight(3);
    line(i, 20, i, 180);
  }
}
```

(a) Comando



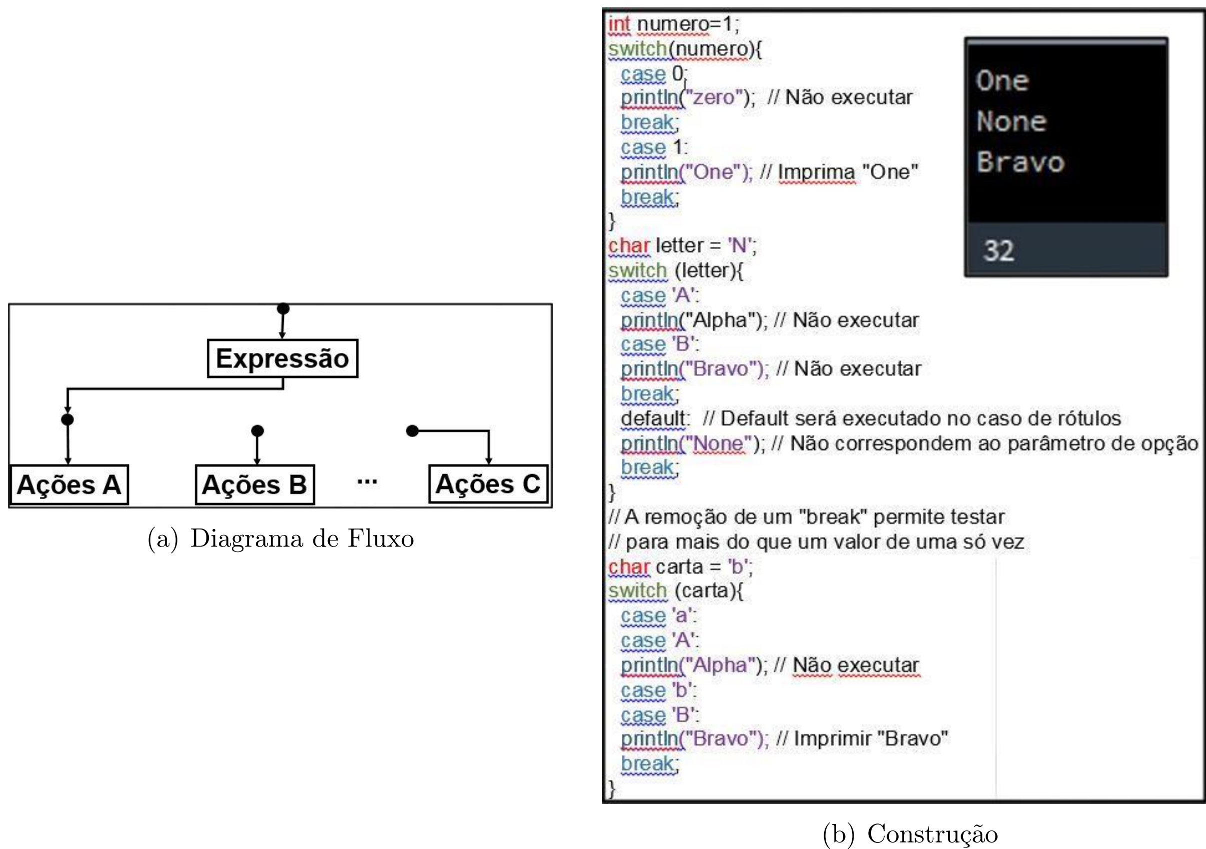
(b) Imagem condicionada

Figura 10: Construção condicional com if-else

Quanto ao requisito operadores, é possível subdividir em três categorias:

- Operadores aritméticos: $+$, $-$, $\star$, $/$;
- Operadores condicionais: $>$, $<$, $>=$, $<=$, $<>$;
- Operadores lógicos: e, ou, negação.

Sendo que os operadores condicionais só podem ser usados nas estruturas condicionais, tendo em vista que o resultado produzido por essa verificação ser sempre uma



(a) Diagrama de Fluxo

(b) Construção

Figura 11: Diagrama e Construção com a estrutura switch();

variável do tipo boolean - verdadeiro ou falso - o que viabiliza ou não a verificação da condição. Ressalta-se ainda que os operadores condicionais divergem dos operadores de atribuição, como na expressão $a = 10$ que é distinto de $a == 10$, uma vez que o primeiro atribui o valor 10 à variável e o segundo verifica se a tem um valor numérico inteiro de 10 (resultando assim em verdadeiro ou falso). A seguir listamos os principais operadores condicionais do programa.

$==$ Igual a;	$>=$ Maior ou igual a;
$>$ Maior que;	$<=$ Menor ou igual a;
$<$ Menor que;	$!=$ Diferente ou negação (*not*).

A Figura 12 foi construída para validar se um número inteiro a é maior ou igual do que um inteiro b. No entanto, o resultado foi apresentado de duas formas nessa figura: A primeira delas aparece no console do programa através do comando println; e a segunda aparece numa janela de exibição com a mesma mensagem de validação ao utilizar o comando *text* associado a escolha de um tipo de fonte e seu tamanho.

```
int a=40;
int b=40;
size(510,100);
background(0);
PFont font;
font = loadFont("PalatinoLinotype-Italic-48.vlw");
textFont(font,25);
if(a>=b){
  println("A variável a é maior ou igual do que a variável b");
  text("A variável a é maior ou igual do que a variável b", 10, 30);
}
else{
  println("A variável a não é maior ou igual do que a variável b");
  text("A variável a não é maior ou igual do que a variável b", 10, 30);
}
```

(a) Comando condicional

(b) Texto condicionado

Figura 12: Operadores condicionais

Com relação aos operadores lógicos que só operam sobre dados lógicos, podemos inferir que são como elementos de conjunção sintática que viabilizam a execução de condições complexas, tipo: Se não chover e fizer calor, então vamos à praia. A verificação dessa condição é concretizada por um operador lógico que serve para interligar os elementos, são eles com seus respectivos símbolos:

&& e; || ou; ! não.

A tabela 6 apresenta um exemplo com os valores lógicos para cada um dos operadores lógicos listados acima.

Tabela 6: Tabela de exemplos dos operadores lógicos

|| - ("ou")

A	B	A || B
Falso	Falso	Falso
Falso	Verdade	Verdade
Verdade	Falso	Verdade
Verdade	Verdade	Verdade

&& - ("e")

A	B	A && B
Falso	Falso	Falso
Falso	Verdade	Falso
Verdade	Falso	Falso
Verdade	Verdade	Verdade

! - ("não")

A	! A
Falso	Verdade
Verdade	Falso

Um exemplo de uma aplicação utilizando os operadores lógicos no *Processing* pode ser percebida pela Figura 13 que mostra o resultado e a linha de comandos nessa linguagem.

2.4 Estrutura de Repetição

Os códigos dependentes de uma condição lógica não são a única forma de estabelecer o critério para que uma determinada operação seja satisfeita. Nesses casos, utiliza-se

```
size(200, 200);
background(200,200,0);
boolean op=false;
for (int i=5; i<=195; i+=5) {
  //Operador lógico "e"
  stroke(10);
  strokeWeight(1.5);
  if ((i>35)&&(i<100)) {
    line(5, i, 95, i);
    op=false;
  }
  //Operador lógico "ou"
  stroke(20);
  strokeWeight(1.5);
  if ((i<=35) || (i>=100)) {
    line(105, i, 195, i);
    op=true;
  }  //Testar se um valor booleano é "true" (
    // A expressão "if(op)" é equivalente a "
  if (op) {
    stroke(1);
    point(width/2, i);
  }   // Teste se um valor booleano é "false"
    // A expressão "if(!op)" é equivalente a
  if (!op) {
    stroke(100);
    point(width/4, i);
  }
}
```

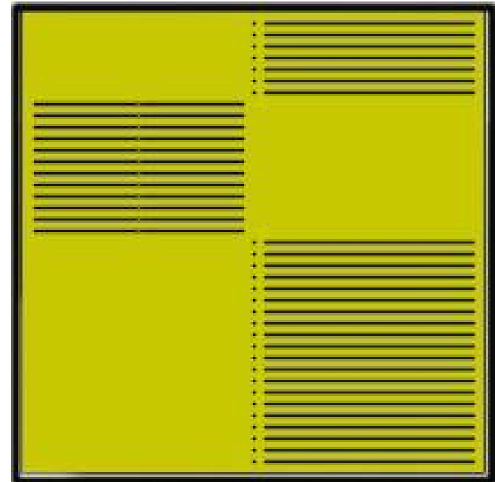


Figura 13: Operadores Lógicos

frequentemente no desenvolvimento de programas de computador uma estrutura de repetição de comandos atuantes em um conjunto de operações sobre vários dados diferentes, ou até a condição inicial a ser concretizada, ou seja, uma programação com repetição simples consiste em estabelecer basicamente uma condição que se verdadeiro executa um comando e retorna ao ponto inicial para ser avaliado se contínua a executar a ação ou se caso o resultado seja falso parar o programa, como na Figura 14. As estruturas de repetição podem ser subdividas de quatro formas, distinguindo-se pelo local no qual a condição é tratada:

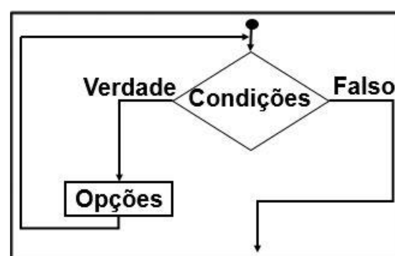


Figura 14: Diagrama de Fluxo com repetição simples

⋆ **Estrutura de Repetição com teste no início**

Essa estrutura repete um fluxo de execução enquanto uma certa condição esteja sendo satisfeita, de acordo com o código:

while (condição)
{ primeiro comando a ser executado;
segundo comando a ser executado;
...
comando n a ser executado }

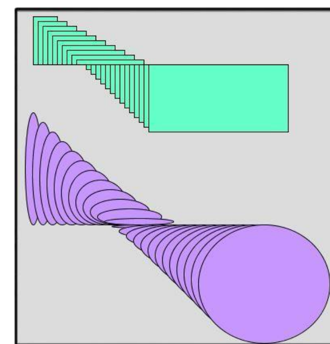
Compete ressaltar que a condição ***while*** (enquanto em português) possui uma expressão lógica do mesmo tipo e forma das utilizadas em estruturas if ... else. E se a expressão tiver valor inicialmente falso, os comandos embutidos dentro do bloco while nunca serão executados, no entanto, se a condição for permanentemente verdadeira, o bloco continuará em execução indefinidamente e programa ficará em execução contínua, levando ao que popularmente se chama de "travamento da máquina". O que não é desejável em um programa, por isso, deve-se prestar atenção na condição lógica do comando *while*.

A Figura 15(a) ilustra um exemplo de utilização da estrutura de repetição while para desenhar um conjunto de formas geométricas como mostra a Figura 15(b).

```
size(400,420);
background(220);

int n=0;
int i=0;
while(n<150 && i<200){
  fill(200,150,255);
  ellipse(2*n+20,200+n,n+20,2*n-140);
  fill(100,255,200);
  rect(i+20,i+10,i+30,60-i);
  n=n+6;
  i=i+6;
}
```

(a) Comando while



(b) Imagem while

Figura 15: Desenhando formas com o comando while

⋆ Estrutura de Repetição com teste no final

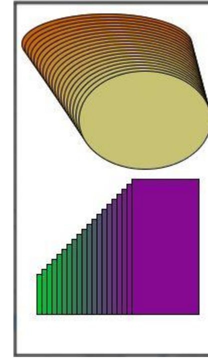
Nessa estrutura, um fluxo de execução será repetido ao menos uma vez e cada vez que terminar a execução de um dos fluxos, o teste condicional é feito para verificar se o fluxo é executado novamente. Sua linha de comando é executado pela descrição:

do
{ primeiro comando a ser executado;
segundo comando a ser executado;
...
comando n a ser executado
} while (condição)

Ilustramos o comando do {} while( ) com os comandos da Figura 16(a) para desenhar um conjunto de formas geométricas como mostra a Figura 16(b).

```
size(200, 350);
background(255);
int i=0;
int n=0;
do {
  fill(200, 100+n, 3*i);
  ellipse(i+90, 2*i+40, 170-i, i+60)
  fill(n+i, 200-2*n, 50+n);
  rect(n+20, 270-n, 0.5*n+20, n+40);
  i=i+2;
  n=n+5;
}
while (i<90 && n<100);
```

(a) Comando do {} while( )



(b) Imagem do {} while( )

Figura 16: Desenhando formas com o comando do {} while( )

Dessa forma, todas as condições são testadas pelo menos uma vez, mesmo que o primeiro comando seja falso.

⋆ Estrutura de Repetição com variável de controle

Nos casos que é difícil determinar quantas vezes um bloco será executado, recomenda-se utilizar os comandos while e do while uma vez que sua operacionalização se repetirá diversas vezes até a condição ser satisfeita. Já, nos casos onde o número de vezes que um bloco é executado é conhecido, o comando mais indicado é o for (para, em português) por permitir a execução de um bloco limitando-se em uma quantidade de vezes conforme a estrutura abaixo:

```
for (inicialização; condição; incremento) {
comando 1
...
comando n
}
```

A estrutura do comando for tem um funcionamento diferenciado em relação aos comandos anteriores. Visto que o comando de inicialização é o primeiro a ser iniciado. Em seguida, o ciclo de repetições se inicia validando a condição lógica e, enquanto a mesma for verdadeira, os comandos dentro do for são executados, seguidos do comando na posição incremento. Salienta-se ainda que as estruturas while e for são equivalentes, quanto à capacidade de expressão em um programa. No entanto, o formato do for permite expressar de maneira mais clara alguns tipos de construções. A equivalência dessa estrutura com a while se dá pela descrição abaixo:

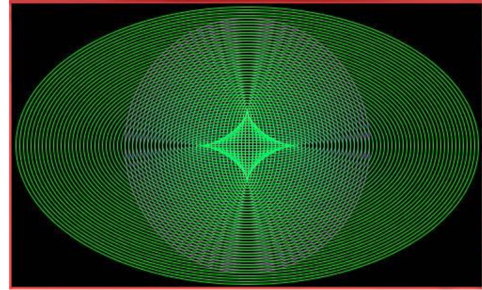
A Figura 17(a) mostra o comando de uma programação na estrutura for que produz como resultado a imagem da Figura 17(b). Se alterarmos o elemento da variável do

```
while (condição) {
comando 1
...
comando n
incremento
}
```

comprimento e da largura da elipse, $4^{\star}i-300, 300-3^{\star}i$ por $4^{\star}i-290, 500-3^{\star}i$, o resultado visual é totalmente diferente.

```
size(500,300);
background(1);
noFill();
smooth();
for(int i=10; i<200; i=i+2){
  stroke(255-2.5*i, 10+3.5*i, 255-1.5*i);
  ellipse (width/2,height/2, 4*i-300,300-3*i);
}
```

(a) Comando for



(b) Imagem for

Figura 17: Desenhando formas com o comando for

⋆ Estrutura de Repetição com teste no meio

Em alguns casos, a estrutura de repetição no início ou no final de um bloco são difíceis de serem executadas com condições complexas. E em função disso, a estrutura de repetição com teste no meio que utiliza-se do comando **break** passar a ser útil para interromper a repetição a qualquer momento da execução. Ao interromper um determinado fluxo, o sistema desvia a execução do programa para imediatamente após aquele fluxo.

Em sua estrutura de repetição, a linguagem segue a descrição:

```
while (true) { // Ou outra condição
primeiro comando a ser executado
segundo comando a ser executado
if (condição)
break
terceiro comando a ser executado
n comando a ser executado
}
```

A interação de um laço é interrompida pelo comando continue, fazendo com que a execução do laço seja desviada para o início deste. Se o comando continue é chamado dentro de um "for", o controle passa a ser executado dentro do comando incremento. A tabela 7 resume a comparação entre as estruturas de repetição apresentadas nesse tópico.

Tabela 7: Quadro comparativo das estruturas de repetição

Estrutura	Quando é testada a condição	Quantidade de execuções	Efeito do comando continue
while	início	0 ou mais	Passa para a próxima iteração, ou seja, testa a condição;
do while	fim	1 ou mais	Passa para a próxima interação, isto é, testa a condição;
for	início	0 ou mais	Faz o incremento e depois testa a condição;
Teste no meio	Qualquer lugar	Depende se utiliza while, do while ou for	Depende se utiliza while, do while ou for.

2.5 Variáveis Complexas

As variáveis ou gavetas que utilizamos até agora permitiram o armazenamento de apenas um valor de cada vez. No entanto, para a construção de aplicativos aprimorados é necessário o conhecimento de uma nova variável que viabilize o armazenamento de diversos valores simultaneamente, a essa variável dá-se o nome de vetores ou *arrays* que são um tipo específico de variável dinâmica, como se fosse uma lista ou gaveta com inúmeras subdivisões.

Esse recurso viabiliza a armazenagem de um conjunto de dados de mesma natureza em uma única variável. Com isso, nossos programas podem executar diversos movimentos utilizando uma única variável a depender de qual função de manipulação de um vetor estejamos ativando. As funções de um vetor podem ser ativadas com as seguintes funções com seu respectivo efeito gráfico.

- **append** Aumenta o tamanho do vetor acrescentando-se um elemento ao final deste;
- **arrayCopy** Copia um vetor ou parte dele em outro;
- **concat** Concatena (junta) dois vetores;
- **expand** Aumenta um vetor para o dobro de seu tamanho ou para um tamanho especificado;
- **reverse** Inverte um vetor;
- **shorten** Elimina o último elemento de um vetor;
- **sort** Ordena um vetor;
- **splice** Coloca um vetor ou elemento dentro de um outro;
- **subset** Retorna um vetor de elementos de dentro de um vetor já existente.

As declarações de uma variável do tipo vetor podem ser elucidadas da mesma maneira que qualquer outro tipo de dado. A diferença consta apenas após o nome do tipo de dado, ou seja, ao final do tipo de variável deve-se colocar um sinal de abertura e de fechamento de colchetes ([e]). Nesse sentido, qualquer dos seguintes códigos declaram vetores:

i. int[] posição;

ii. float[] tamanho;

iii. color[] cor;

iv. string[] nome;

Entretanto, o vetor não pode ser utilizado com apenas a declaração de sua variável, para seu funcionamento é imprescindível informar quantas posições o vetor disponibiliza e a partir disso a memória do computador é separada para cada posição do vetor que é chamado com a palavra chave **new**.

A declaração desse vetor com o número de posições do mesmo pode ser escrito de duas formas como segue:

int[] posicao; posicao = new int[5];	color[] cor = new color[10];

De acordo com os códigos supracitados, o vetor posição é criado com 5 posições em duas linhas, enquanto, o vetor cor é criado com 10 posições.

A atribuição de valores dentro de uma variável vetor, assim como o número de posições que o mesmo ocupa pode ser declarada de duas formas, como segue no código, onde a primeira coluna refere-se a atribuição de valores nomeando o vetor e a posição desejada entre colchetes ([e]) seguida pelo sinal de igualdade (=), ao passo que na segunda coluna, a criação do vetor com o operador new é substituído por uma lista separada por vírgulas e delimitada por chaves.

int[] dado = new int[3];
dado[0] = 5;
dado[1] = 3;
dado[2] = 2;
	int[] dado = {3, 1, 2};

Nas situações onde o programador tenta acessar uma posição fora da lista criada conduz um erro de execução que pára o programa com a exibição da mensagem **ArrayIndexOutOfBoundsException**. Normalmente esta situação acontece com programadores

iniciantes que tentam subscrever uma variável no nível superior, sem prestar atenção que toda variável de um vetor começa sempre na posição zero.

Usualmente a estrutura for é associada para percorrer vetores e sua inicialização se dá com o valor zero, limitando-se até o tamanho do vetor. A interação nesse caso é feita item a item. Com isso, a estrutura for associada a vetores segue como no código da tabela 8 que gera a construção da Figura 18.

Tabela 8: Comando for associado a vetores com construção dinâmica

Linha	Comando	Linha	Comando
1	$float[\ ]\ x, y;$ // Armazena a posição do objeto	20	} else {
2	$color[\ ]\ c;$ // Cor do objeto	21	$x[i] = x[i] + 1;$ //Vá para a direita
3	$void\ setup()${	22	}
4	$size(300, 300);$ // Configuração da tela	23	$if(y[i] < mouseY)${//Se o objeto estiver acima do mouse
5	$x = new\ float[20];$	24	$y[i] = y[i] - 1;$ //Vá para cima
6	$y = new\ float[20];$	25	} $else$ {
7	$c = new\ color[20];$	26	$y[i] = y[i] + 1;$ //Vá para baixo
8	/* O comando new criando 20 posições para armazenar a posição e a cor do objeto */	27	}
9	$for(int\ i = 0; i < x.length; i++)${	28	$if(x[i] < 0\ \vert\vert\ y[i] < 0\ \vert\vert\ x[i] > width\ \vert\vert\ y[i] > height)${ //Se o objeto saiu da tela
10	$x[i] = random(10, width - 10);$	29	$x[i] = random(10, width - 10);$ //Sorteia a posição em x
11	$y[i] = random(10, height - 10);$	30	$y[i] = random(10, height - 10);$ //Sorteia a posição em y
12	//Coloca o objeto na tela em posição aleatória	31	}
13	$c[i] = color(random(256), random(256), random(256));$ //Sorteia a cor do objeto	32	// Desenha o objeto
14	} }	33	$fill(c[i]);$ //Cor do objeto
15	$void\ draw()${	34	$ellipse(x[i], y[i], 10, 10);$ //Desenha elipse
16	$background(255);$ //Limpa a tela	35	$rect(y[i], x[i], 6, 6);$ //Desenha retângulo
17	$for(int\ i = 0; i < x.length; i++)${	36	$arc(x[i] - 5, y[i] - 5, 15, 15, 0, PI + QUARTER_PI, PIE);$ //Desenha arco
18	$if(x[i] < mouseX)${ //Se o objeto estiver à esquerda do mouse	37	}
19	$x[i] = x[i] - 1;$ //Vá para a esquerda	38	}

Nesse código, salienta-se que a variável "i++" disponível nas linhas 9 e 17 da tabela 8 tem o mesmo resultado que o comando "$i = i + 1$", e em função disso, cada iteração acaba acessando uma posição diferente do vetor. Outro recurso incrementado refere-se ao código mouseX, linha 18, que viabiliza uma animação que muda o sentido do movimento a medida que o mouse se aproxima de um determinado ponto da tela.

A aplicação em vetor, notoriamente, é uma extensão de uma variável simples que pode ainda ser estendida para uma estrutura bidimensional chamada de **matrizes**. De

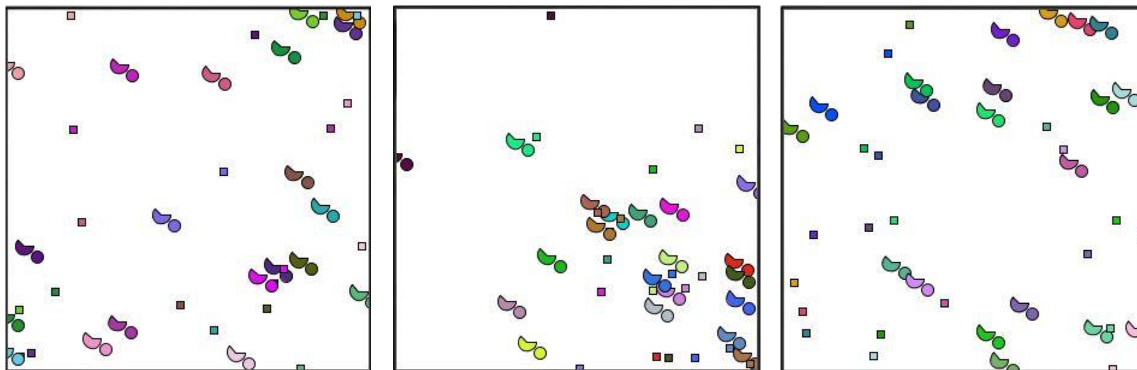


Figura 18: Animações na estrutura for associado a vetores

fato, se um vetor é interpretado como uma linha de uma tabela do Excel, a matriz pode abranger uma tabela inteira. No caso de uma matriz bidimensional, seus valores são acessados através de dois valores de índice e são úteis para armazenar imagens. A imagem da Figura 19(a) foi construída usando uma matriz bidimensional para desenhar uma imagem em tons de cinza de acordo com os códigos incrementados na tabela 9.

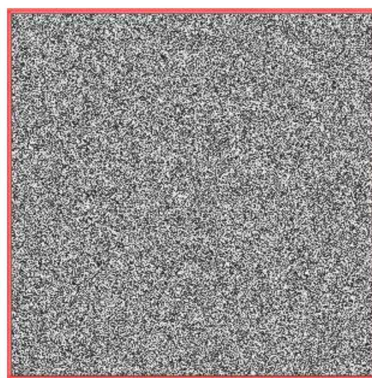
Tabela 9: Código da Imagem em tons de cinza usando uma matriz bidimensional

Linha	**Comando**	**Linha**	**Comando**
1	// Matriz bidimensional	11	$matriz[i][j] \ = \ int(random(255));$
2	$size(300,300);$	12	}
3	background(255);	13	}
4	int colunas = width;	14	//Desenhar pontos
5	int linhas= height;	15	for $(int\ i = 0; i < colunas; i++)$ {
6	//Declaração da matriz bidimensional	16	for $(int\ j = 0; j < linhas; j++)$ {
7	int[][] matriz = new int[colunas][linhas];	17	stroke($matriz[i][j]$);
8	//Inicializar valores da matriz 2D;	18	point(i, j);
9	for $(int\ i = 0; i < colunas; i++)$ {	19	}
10	for $(int\ j = 0; j < linhas; j++)$ {	20	}

Ao substituirmos o comando $point(i,\ j)$; da linha de comando 18, por $point(width/3 + 2^{\star}i/3 - 50, height/3\ + 2^{\star}j/3 - 50)$; a nova imagem construída passar a possuir o dimensionamento da tela de tons cinzas reduzida como podemos constatar na 19(b). Com isso, obtemos uma matriz com menos elementos que só são exibidos na tela do console se incluímos o comando "println ("$matriz[i][j] \ = \ int(random(255));$");" antes de finalizar o programa.

2.6 Funções ou Métodos

Em programação a palavra função e método possuem o mesmo significado como afirma Amado [28]. Onde são mantidos como pequenos programas que tomam determina-


(a) Tela fechada com pontos

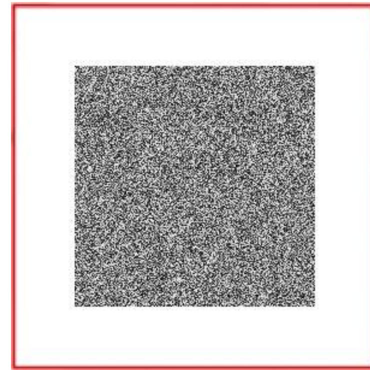

(b) Tela parcialmente fechada com pontos

Figura 19: Imagem em tons de cinza usando uma matriz

dos valores de entrada e produzem um resultado. No entanto, o autor supracitado afirma que a provável distinção encontra-se no fato de usarmos uma função quando estamos nos referindo a uma instrução conhecida pela linguagem (API[1]) tal como ellipse(), ou fill(). Já, se desejarmos construir uma casa estruturada (porta, janela, telhado, entre outros itens) estamos a executar um método ou conjunto de instruções para o efeito. Por este motivo, Amado caracteriza tudo como métodos, visto que o programa só executa uma função, seja ela um rect() ou desenhar uma casa, quando esta tem que estar descrita de alguma forma.

Salienta-se ainda que algumas funções matemáticas já estão embutidas para o âmbito dessa linguagem, são elas:

- sin, cos, tan, atan
- sqrt, log, exp, abs

Para as funções que não estão definidas nessa linguagem como um pacote de ferramentas, é possível incrementar com novas estruturas utilizando parâmetros que são valores passados ao método quando este é chamado mediante a instrução seguinte estabelecida de forma geral, onde seus significados são:

```
tipo nome (tipo1 param1, tipo2 param2, ··· , tipoN paramN) {
comando;
...
comando;
}
```

[1]A API é uma Interface de Programação de Aplicativos, isto é, um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se nos detalhes da implementação do software, mas apenas usar seus serviços.

1. **tipo** é o tipo do valor de retorno (int, float, entre outros);

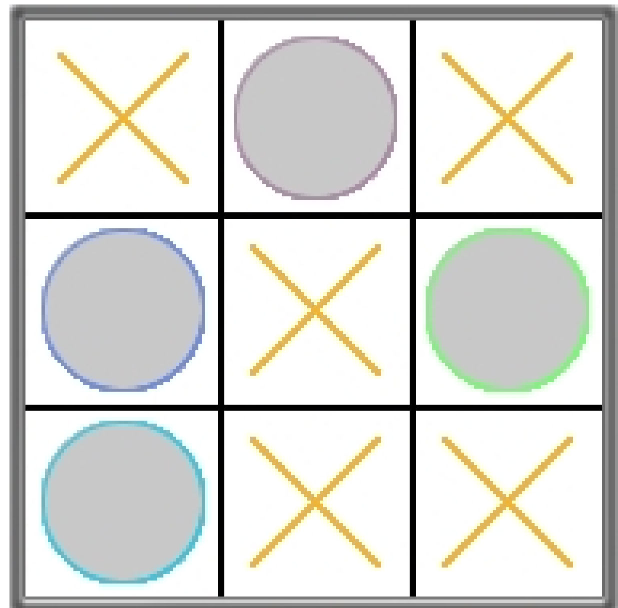
2. **nome** é o nome pelo qual a função será conhecida;

3. **tipo1 param1** são o tipo e o nome do primeiro parâmetro.

Nos casos onde o método computa algo mas não retorna nada, pode-se usar a palavra **void** como **tipo**, como podemos constatar no comando da Figura 20(a) que proporciona a imagem de um tabuleiro estático do jogo da velha, ver imagem 20(b).

```
int i, j, m, n;
void setup() {
  size(180, 180);
  background(255);
}
void x(int x, int y) {//Função X
  stroke(255-m, 200-n, 10+m+n);
  line(x-20, y-20, x+20, y+20);
  line(x-20, y+20, x+20, y-20);
  m=10;
  n=20;
}
//Função Círculo
void circulo(int w, int h) {
  stroke(200-h, 20+w+h, 255-w);
  ellipse(w, h, 50, 50);
}
void draw() {
  x(30, 30);
  x(90, 90);
  x(150, 150);
  x(150, 30);
  x(90, 150);
  fill(200, 200, 200);
  circulo(30, 90);
  circulo(90, 30);
  circulo(150, 90);
  circulo(30, 150);
  stroke(0);
  rect(i+60, 0, 1, 200);
  rect(0, j+60, 200, 1);
  j=60;
  i=60;
}
```

(a) Comando Jogo da Velha



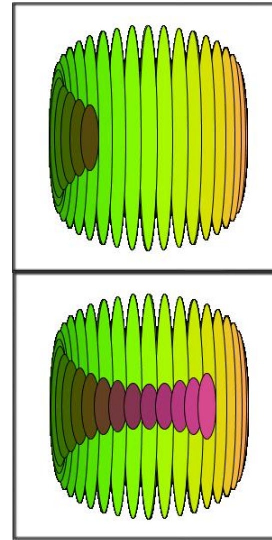
(b) Jogo da Velha estático

Figura 20: Construção do tabuleiro para o Jogo da Velha

Se houver necessidade de retornar algum valor, deve-se prioritariamente incluir em algum ponto da definição um comando da forma **return** valor que viabilizará a reutilização do comando utilizando uma quantidade de códigos de programação reduzido. A exemplo desse recurso temos a aplicação da Figura 21(b) construída com o comando 21(a).

```
float x(float angulo){
  return cos(radians(angulo))*100+150;
}
float y(float angulo){
  return sin(radians(angulo))*100+150;
}
float graus=0;
void draw(){
  fill(x(graus),y(graus),x(graus)-y(graus));
  ellipse (x(graus),150,20,y(graus));
  graus=graus+10;
}
void setup(){
  size(300,300);
  smooth();
  background(255);
}
```

(a) Comando com função *return*



(b) Animação com função *return*

Figura 21: Construção com função de regresso

Além dos recursos apresentados, o *Processing* possui diversas aplicações que viabilizam a construção de jogos com poucos recursos ou domínio da linguagem via às aplicações que podem ser desenvolvidas a partir dos comandos de transformação no plano. Contudo, antes de realizar tal aplicação é de fundamental importância analisar como imagens planas são deslocadas e para isso apresentaremos a transformação geométrica de forma conceitual viabilizando a interpretação da linguagem de programação para o deslocamento de imagens no plano.

3 Transformação Geométrica no Plano

As transformações geométricas estão presentes historicamente na constituição da maioria dos seres vivos através da utilização das simetrias, rotações, translações e homotetias (ampliações e reduções) para copiar os elementos apresentados pela natureza e reproduzidos em cerâmicas, tecidos e cestas em uma prática ancestral.

Nesse sentido, os avanços no estudo da geometria impulsionados pelos egípcios, babilônios, hindus, chineses e principalmente pelos gregos com a publicação do livro "Os Elementos" de Euclides[24] que, em função do desenvolvimento do modelo axiomático, trouxe uma enorme contribuição para o ensino da geometria e o comportamento das funções através das transformações geométricas.

Dentre as obras dos tempos modernos destacam-se as contribuições do artista holandês Maurits Cornelis Escher por utilizar as transformações geométricas em seus trabalhos. No entanto, como afirma Medeiros e Lopes [29, 30], apesar de sua obra estar repleta de conceitos matemáticos e todos os seus trabalhos possuírem um forte conteúdo dessa ciência, o mesmo era absolutamente leigo no assunto.

> Conta-se até que H.M.S. Coxeter, um dos papas da geometria moderna, entusiasmado com os desenhos do artista, convidou-o a participar de uma de suas aulas. Vexame total, Para decepção do catedrático. Escher não sabia do que ele estava falando, mesmo quando discorria sobre teorias que o artista aplicava intuitivamente em suas gravuras [30, p. 5].

Esse artista dedicou seus trabalhos com isometrias decorativas a partir de suas visitas ao palácio mourisco de Allambra, em Granada – Espanha que foi construído pelos árabes no século XIII. Apesar de não possuir o conhecimento sobre o assunto, Escher copiou obsessivamente os ornamentos das paredes do respectivo palácio e, acabou por descobrir os movimentos utilizados nos ornamentos: as isometrias, isto é, a translação, a rotação, a reflexão, a translação refletida e suas combinações. E através da experimentação conseguiu chegar aos dezessete grupos existentes das combinações isométricas que tornam

a figura invariante [29].

Na concepção do Professor de geometria do Instituto de Matemática e Estatística (IME) da Universidade de São Paulo (USP), Sérgio Alves, que utiliza os desenhos desse artista holandês em suas aulas,

> é notável que Escher, sem qualquer conhecimento prévio de matemática, tenha descoberto essas possibilidades. Quanto aos quatro movimentos, são os únicos possíveis de serem aplicados sobre um padrão plano de modo que o resultado obtido seja exatamente a figura original. Em termos matemáticos, são as únicas isometrias do plano. O estudo desses movimentos é chamado de Geometria das Transformações e suas leis governam a construção de desenhos periódicos [30, p. 5].

O holandês Escher chamou seus trabalhos de Divisão Regular do Plano (DRP) como fundamenta Medeiros [29] e algumas de suas artes são mostradas nas figuras[1] 22(a), 22(b).

(a) Butterfly (N°. 70)

(b) Lizard/Fish/Bat (N°. 85)

Figura 22: Detalhe da DRP de Escher

No entanto, voltando a analisar as contribuições de Euclides, encontramos vinte e três definições, cinco postulados e nove noções para demonstrar toda a geometria conhecida como geometria euclidiana ou geometria clássica no Livro I de "Os Elementos". Os cinco postulados estão listados a seguir:

> 1° Fique postulado traçar uma reta a partir de todo ponto até todo ponto.
>
> 2° Também prolongar uma reta limitada, continuamente, sobre uma reta.

[1]Imagens de Escher obtidas no site *http://www.mcescher.com/gallery/ink/*

> 3º E, com todo centro e distância, descrever um círculo.
>
> 4º E serem iguais entre si todos os ângulos retos.
>
> 5º E, caso uma reta, caindo sobre duas retas, faça os ângulos interiores e do mesmo lado menores do que dois retos, sendo prolongadas as duas retas, ilimitadamente encontrarem-se no lado no qual estão os menores do que dois retos [24, p. 98].

A diferenciação entre os postulados e as noções comuns constante no Livro I de Euclides versa sobre a exclusividade dos postulados para à geometria, a medida que as noções comuns são afirmações de validade geral. A exemplo disso, temos a primeira das noções comuns de Euclides [24, p. 99] ao afirmar que "as coisas iguais à mesma coisa são também iguais entre si".

Diversas contribuições foram atreladas ao longo da história, no entanto, somente com Sophus Lie (1842–1899) e Félix Klein (1849–1925) que criaram a noção de grupos de transformações e invariante correspondentes fundamentada na teoria de grupos, que resolveram o problema de identificação da estrutura do conjunto de transformações como sinaliza Medeiros [29].

De acordo com Fainguelernt [31, p. 72], "Klein, em 1870, revolucionou o enfoque da geometria, ao afirmar que devemos entender o seu contexto como o estudo das propriedades invariantes das figuras face às transformações de um grupo". E, em 1872, Klein afirmou que o conceito de transformação desempenha um papel coordenador e simplificador no estudo da Geometria, possibilitando uma abordagem intuitiva e informal da mesma através das transformações.

Félix Klein foi um professor da Universidade de Erlangen (1872–1875) e posteriormente em Leipzig (1880–1886) e em Gottingen (1886–1913). Para Klein [32], a relação entre um grupo de transformações e uma geometria é que a define, chamando esse grupo de transformações de *grupo principal* para essa geometria. Nas palavras do autor supracitado [32, p. 7] as "propriedades geométricas não se alteram pelas transformações do grupo principal. Podemos também afirmar de modo inverso: as propriedades geométricas são caracterizadas pela sua invariância com respeito às transformações do grupo principal".

Outra importância para o ensino das TGP consta no fato da maioria dos aplicativos de computação gráfica permitir a manipulação de uma imagem de várias maneiras, tais como a mudança de suas proporções, rotações ou cilhamentos como aponta Conceição [33]. Além de uma técnica básica da distorção de uma imagem pelo movimento dos vértices de um retângulo que a contém.

Entretanto, Conceição [33] afirma que a deformação consiste em distorcer várias

partes da imagem de maneiras distintas e por isso, é um procedimento mais complicado. Em função disso, a deformação de duas imagens por procedimentos complementares com a fusão das deformações obtidas resulta num morfismo[2] das duas imagens.

Segundo o autor mencionado, a principal aplicação das deformações e morfismos está na produção de efeitos especiais no cinema, na televisão e na propaganda. Além das diversas aplicações científicas e tecnológicas para essas técnicas como na assistência à cirurgia plástica e de reconstrução, na investigação de variações do projeto de um produto e o "envelhecimento" de fotografias de pessoas desaparecidas ou suspeitos da polícia. A seguir, apresentamos uma breve análise matemática das transformações geométricas, evidenciando sua dimensão geométrica e algébrica.

3.1 Análise matemática das transformações geométricas

Uma Transformação Geométrica no Plano é uma função que associa a cada ponto do plano um outro ponto também do plano através de certas regras. Dessa forma, os movimentos que originam a mudança de posição dos pontos podem, ou não, manter a forma e o tamanho da figura a ser transformada. Segundo Wagner [34, p. 70] "Uma transformação T no plano Π é uma função $T : \Pi \to \Pi$ que associa a cada ponto A do plano um outro ponto $A' = T(A)$ do plano chamado imagem de A por T".

Matematicamente, dizemos que uma transformação é bijetiva se, e somente se, uma transformação no plano Π for injetiva, e isso acontece quando possui pontos distintos do plano com imagens distintas, isto é, se $P \neq Q$, então $T(P) \neq T(Q)$. E se a transformação for também sobrejetiva no mesmo plano Π, nesse caso, possui para todo ponto A' pelo menos uma imagem para o ponto A, ou seja, para todo ponto A' existe A tal que $T(A) = A'$. Portanto, essa transformação é bijetiva, isto é, $\forall A' \in \Pi, \exists | A \in \Pi; T(A) = A'$. Nesse contexto, utilizaremos somente as transformações bijetivas.

As transformações que serão tratadas aqui serão sempre bijeções de um conjunto $M \subset \mathbb{R}$ em outro conjunto $N \subset \mathbb{R}$. Onde M e N são reconhecidas como "figuras geométricas" no mundo físico. Como o conceito de Geometria trazido por Klein está intimamente ligado à Teoria de Grupos é recomendável que os professores que não estejam habituados

[2]Um morfismo definido por Conceição [33, p. 49] "pode ser descrito como uma combinação de deformações de duas imagens distintas, associando características correspondentes das duas imagens. Uma das duas imagens é escolhida como imagem inicial e a outra como a imagem final". E é usado para gerar um efeito de animação ou de transição de imagens.

com a noção de grupo, que busquem em bons livros[3] de Álgebra para um aprofundamento mais detalhado.

A definição de um grupo necessita de um conjunto de objetos, como os números inteiros, e de uma operação que possa ser realizada com eles, como a soma, de tal forma que esta operação cumpra certos requisitos de um grupo:

1. A operação precisa ser fechada para o conjunto utilizado. Nesse sentido, a soma de dois números inteiros é sempre um número inteiro;

2. A operação precisa ser associativa. Nesse sentido, temos que a soma é obviamente associativa;

3. A operação precisa possuir elemento neutro. Nesse caso, o elemento zero faz este papel, visto que $0 + t = t + 0, \forall t \in \mathbb{Z}$;

4. A operação precisa possuir um elemento inverso para qualquer elemento do conjunto dado. Para esta situação o inverso de t é o $-t$, uma vez que $-t+t = t+(-t), \forall t \in \mathbb{Z}$.

Em função dos Grupos de Transformações, o nosso "conjunto de objetos" conterá as transformações sobre figuras do plano Π e a "operação" que utilizaremos será a operação de composição. Nesse sentido, faz-se necessário uma breve revisão de Transformações Lineares (ver Apêndice A) para associar com as transformações geométricas operacionalizadas na forma matricial, como associaremos com a linguagem de programação. No entanto, isso não inviabiliza o ensino das TGP nos anos finais do Ensino Fundamental, ao contrário, revela uma possibilidade de se abordar matrizes de forma contextualizada no Ensino Médio.

Como a pesquisa estará sempre abordando transformações (e movimentos), haverá sempre uma figura que é transformada em outra de acordo com o contexto da explicação. Nesse ponto, apresentaremos as transformações geométricas de acordo com as relações estabelecidas entre as figuras iniciais e finais, isto é, quando é isométrica, isomórfica ou anamórfica.

[3]Recomenda-se o livro de Álgebra Moderna de Domingues e Iezzi ([35], capítulo 4)

3.2 Transformações Isométricas

As primeiras TGP chamadas de isométricas são as transformações que preservam distâncias. Nesse sentido, definimos uma transformação isométrica T quando

$$d[T(M), T(N)] = d(M, N)$$

para quaisquer pontos M e N do plano Π. De acordo com Wagner [34, p. 71], qualquer transformação isométrica possui as seguintes regras:

a) a imagem de uma reta por uma isometria é uma reta.
b) uma isometria preserva paralelismo.
c) uma isometria preserva ângulos.

Consequentemente, a isometria constitui um grupo de transformações geométricas que auxiliam a geometria euclidiana, tendo em vista a possibilidade de avaliar quando duas figuras são congruentes ao analisar se o movimento preserva a forma e o tamanho da figura original, mudando apenas sua posição no plano. Por esse motivo, as transformações isométricas são denominadas de movimentos rígidos do plano por modificar apenas a posição da figura inicial e são subdivididas em Translação, Reflexão e Rotação.

3.2.1 Translação

Nas palavras de Souza [36, p. 19], “transladar uma figura é o mesmo que movê-la no plano sem rotacioná-la”, isso significa, que a translação é uma TGP que leva cada ponto do plano a um novo ponto segundo uma trajetória representada por um segmento de reta de tamanho fixo e que faz um ângulo fixo em relação ao plano horizontal. Logo, seguindo a ideia de Wagner [34], uma translação é uma transformação no plano Π onde a função bijetora $T : \Pi \to \Pi$ preserva distâncias associando a cada ponto M do plano Π, o ponto $T(M) = M' = M + \overrightarrow{v}$, onde $\overrightarrow{v}$ é um segmento de reta orientado de medida, direção e sentido fixo (Figura 23).

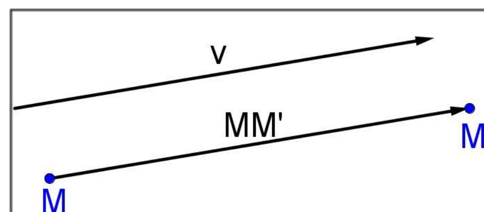


Figura 23: Translação do ponto M segundo o vetor $\vec{v}$

Algebricamente, se considerarmos dois pontos P e Q distintos, de coordenadas (x_1, y_1) e (x_2, y_2), respectivamente, pela definição anterior, conforme a Figura 24 mostra, para que o ponto $M' = (x', y')$ seja o transladado do ponto $M = (x, y)$ segundo o vetor determinado por $\overrightarrow{PQ}$, os triângulos PQR e $MM'O$ devem ser congruentes, daí:

$$m(PR) = x_2 - x_1 = m(MO)$$

e

$$m(RQ) = y_2 - y_1 = m(OM')$$

logo, a expressão do ponto transladado é dada por:

$$(x', y') = T(x, y) = (x + x_2 - x_1, y + y_2 - y_1) \tag{3.1}$$

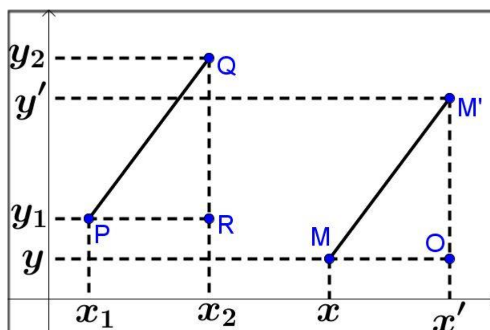


Figura 24: Translação do ponto M segundo o segmento $\overline{PQ}$

Exemplo 3.1. *Se o ponto $P = (2, 5)$ e o ponto $Q = (3, 9)$, a translação segundo o segmento orientado $\overrightarrow{PQ}$, de acordo com a expressão 3.1, será dada por:*

$$T(x, y) = (x + 3 - 2, y + 9 - 5) = (x + 1, y + 4)$$

onde o segmento orientado $\overrightarrow{PQ}$ possui coordenadas vetoriais $(3 - 2, 9 - 5) = (1, 4)$. Dessa forma, a equação de translação de ponto a ponto no plano será dada por:

$$T(x, y) = (x + a, y + b) \tag{3.2}$$

onde $\overrightarrow{PQ} = \overrightarrow{v} = (a, b)$, com a e b pertencente a $\mathbb{R}$.

Por outro lado, pode-se concluir que se $a = 0$, temos uma translação vertical e, se $b = 0$, a translação será horizontal. Nesse sentido ainda, é possível inferir que a transformação desloca os pontos horizontalmente uma distância indicada por a e verticalmente uma distância indicada por b. Por esse motivo, a equação de translação 3.2 pode ser

escrita na forma matricial:

$$T(x,y) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \overrightarrow{v}, onde \quad \overrightarrow{v} = \begin{bmatrix} a \\ b \end{bmatrix}$$

Além disso, as translações possuem as seguintes propriedades.

i A translação é um movimento rígido por preservar distâncias;

ii A composição de duas translações é uma translação. Se todo ponto K de uma figura plana Φ é transladado de um vetor $\overrightarrow{v}_1$ transformando K em K' e, posteriormente, de um vetor $\overrightarrow{v}_2$ que transforma K' em K'', com isso, a transformação resultante levará K em K'' pelo vetor $\overrightarrow{v}_1 + \overrightarrow{v}_2$. Ao prosseguir com esse paralelismo entre translações e vetores validaremos que a composição de translações é comutativa, isto é, se $T_{\overrightarrow{v}_1}$ translada Φ na direção de $\overrightarrow{v}_1$ e $T_{\overrightarrow{v}_2}$ na direção de $\overrightarrow{v}_2$, então $T_{\overrightarrow{v}_1}(T_{\overrightarrow{v}_2}(\Phi)) = T_{\overrightarrow{v}_2}(T_{\overrightarrow{v}_1}(\Phi)) = T_{\overrightarrow{v}_1+\overrightarrow{v}_2}(\Phi)$ (Figura 25).

iii Uma translação admite sempre uma inversão. Para toda translação $T_{\overrightarrow{v}}$ é sempre possível determinar uma translação $T_{\overrightarrow{v}}^{-1} = T_{-\overrightarrow{v}}$ tal que $T_{\overrightarrow{v}}(T_{-\overrightarrow{v}}(\Phi)) = T_{-\overrightarrow{v}}(T_{\overrightarrow{v}}(\Phi)) = \Phi$.

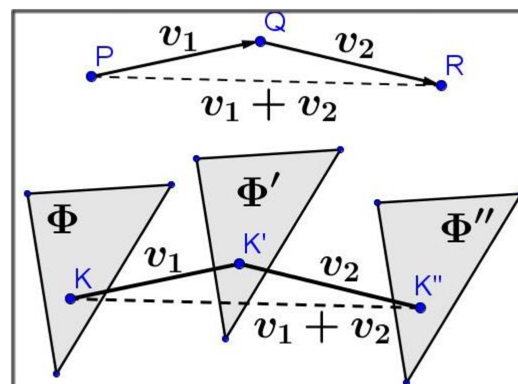


Figura 25: Composição de translação do ponto K em K''

3.2.2 Reflexão (ou Simetria) em relação a uma reta

Dentre os movimentos rígidos, a reflexão ou simetria é a transformação mais difundida no Brasil em detrimento do ensino da óptica geométrica. Sabe-se ainda que as reflexões podem se dar em relação a um ponto ou a uma reta. E por isso, é definida a partir de uma reta r do plano Π, onde a reflexão em torno da reta r é uma função $S_r : \Pi \to \Pi$, tal que $S_r(B) = B$ para todo $B \in r$ e, para $X \notin r$, $S_r(X) = X'$ é tal que a mediatriz do segmento XX' é a reta r (Ver Figura 26).

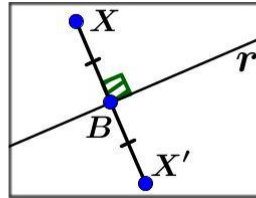
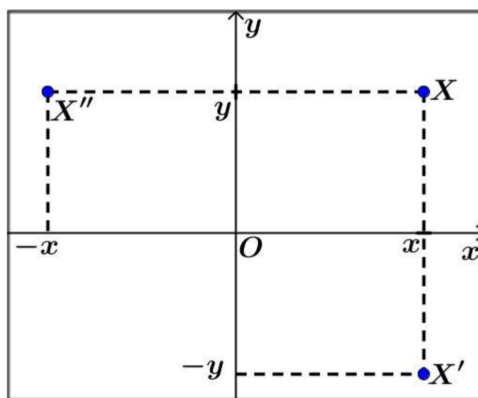


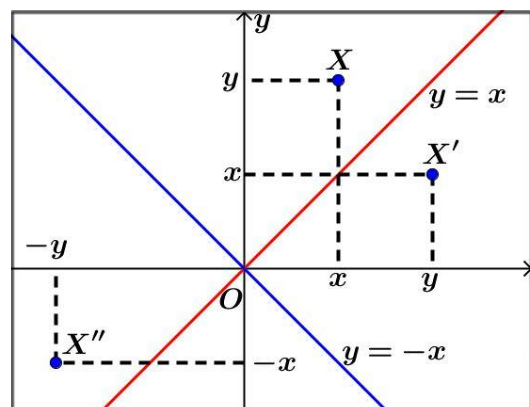
Figura 26: Reflexão

Considerando um sistema de coordenadas cartesianas no plano Π no qual um dos eixos, abscissas ou ordenadas, coincida com a reta r, então a cada ponto $X = (x, y)$, tem-se $S_r(X) = (x', y') = X'$, onde $X' = (x, -y)$ ou $X' = (-x, y)$, respectivamente, ver Figura 27(a).

De modo geral, a relação entre as coordenadas do ponto X com sua imagem refletida, X', na reta r é dada pelo sistema, conforme mostra as Figuras 27(a) e 27(b):



(a) Reflexão em relação aos eixos cartesianos



(b) Reflexão em relação a 1ª e 2ª bissetriz

Figura 27: Reflexão no plano cartesiano

1. Reflexão em relação ao eixo das abscissas $\begin{cases} x' = x \\ y' = -y \end{cases}$

2. Reflexão em relação ao eixo das ordenadas $\begin{cases} x' = -x \\ y' = y \end{cases}$

3. Reflexão em relação a reta $y = x$, primeira bissetriz $\begin{cases} x' = y \\ y' = x \end{cases}$

4. Reflexão em relação a reta $y = -x$, segunda bissetriz $\begin{cases} x' = -y \\ y' = -x \end{cases}$

Outra reflexão importante é em relação a um ponto fixo do plano. Sendo assim, seja $X' = (x', y')$ a imagem refletida do ponto $X = (x, y)$ em relação ao ponto $P = (x_0, y_0)$. Como os pontos X, X' e P são colineares, ver Figura 28(a), tal que

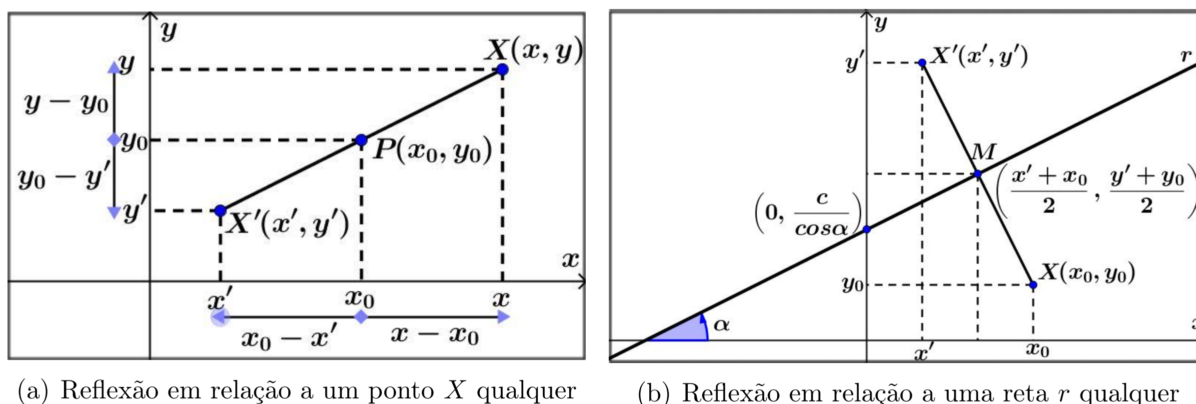
$$d(X', P) = d(X, P) \tag{3.3}$$

a expressão correspondente a essa reflexão obtida a partir de (3.3) e com base na Figura 28(a), resume-se em:

$$(x', y') = (x_0, y_0) + (x_0 - x, y_0 - y) = (2x_0 - x, 2y_0 - y)$$

Logicamente, a lei de transformação $S_r : \Pi \to \Pi$ será dada por:

$$S_r(x, y) = (x', y') = (2x_0 - x, 2y_0 - y) \tag{3.4}$$



(a) Reflexão em relação a um ponto X qualquer (b) Reflexão em relação a uma reta r qualquer

Figura 28: Reflexão em relação a um ponto e uma reta r qualquer

Exemplo 3.2. *A reflexão de um ponto qualquer* $X = (x, y)$ *em relação à origem do sistema cartesiano* $(0, 0)$ *fica assim determinada com base na equação (3.4):*

$$S_r(x, y) = (x', y') = (2 \cdot 0 - x, 2 \cdot 0 - y) = (-x, -y)$$

No entanto, a reflexão de qualquer ponto X do plano Π em relação a uma reta r do próprio plano, determina um ponto X' que se mantém perpendicular a reta r tal que

$$d(X, r) = d(X', r)$$

Nesse sentido, considere as coordenadas do ponto $X = (x_0, y_0)$ e do seu simétrico $X' = (x', y')$ em relação a uma reta r que faz um ângulo α com o eixo das abscissas no

sentido anti-horário, conforme mostra a Figura 28(b). Como a equação cartesiana da reta r é dada por:

$$y = ax + b,$$

onde $a = tg(\alpha)$ e $b = \frac{c}{cos(\alpha)}$, para todo $c \in \mathbb{R}$. Daí,

$$cos(\alpha) \cdot y - sen(\alpha) \cdot x = c \tag{3.5}$$

De modo análogo, a reta que une os pontos X e X' e, é perpendicular a reta r é determinada pela expressão:

$$y = tg(90^o + \alpha) \cdot x + F = -cotg(\alpha)x + F,$$

onde $F = cotg(\alpha)x_0 + y_0$. Consequentemente,

$$cos(\alpha) \cdot x + sen(\alpha) \cdot y = cos(\alpha) \cdot x_0 + sen(\alpha) \cdot y_0 \tag{3.6}$$

Do fato da reta r ser mediatriz do segmento $\overline{XX'}$, tem-se o ponto $M = (m, n)$ (Figura 28(b)) como ponto médio do referido segmento cujas coordenadas:

$$(m, n) = \left(\frac{x_0 + x'}{2}, \frac{y_0 + y'}{2} \right) \tag{3.7}$$

são a solução das equações (3.5) e (3.6). E isso implica no sistema,

$$\begin{cases} cos(\alpha) \cdot n - sen(\alpha) \cdot m = c \\ sen(\alpha) \cdot n + cos(\alpha) \cdot m = cos(\alpha) \cdot x_0 + sen(\alpha) \cdot y_0 \end{cases}$$

Cuja solução é:

$$\begin{aligned} (m, n) &= \big(cos^2(\alpha)x_0 + cos(\alpha)sen(\alpha)y_0 - sen(\alpha)c, sen^2(\alpha)y_0 + cos(\alpha)sen(\alpha)x_0 + cos(\alpha)c\big) \\ &= \left(cos^2(\alpha)x_0 + \frac{sen(2\alpha)}{2}y_0 - sen(\alpha)c, sen^2(\alpha)y_0 + \frac{sen(2\alpha)}{2}x_0 + cos(\alpha)c\right) \end{aligned} \tag{3.8}$$

A partir da comparação das igualdades constantes em (3.7) e (3.8) é possível estabelecer a expressão algébrica do ponto simétrico $X' = (x', y')$, isto é:

$$(x', y') = S_r(x_0, y_0) = (2m - x_0, 2n - y_0)$$

$$= \big(2cos^2(\alpha)x_0 + sen(2\alpha)y_0 - 2csen(\alpha) - x_0, sen(2\alpha)x_0 + 2sen^2(\alpha)y_0 + 2cos(\alpha)c - y_0\big)$$

$$= \big(\left[2cos^2(\alpha) - 1\right] x_0 + sen(2\alpha)y_0 - 2sen(\alpha)c, sen(2\alpha)x_0 + \left[2sen^2(\alpha) - 1\right] y_0 + 2cos(\alpha)c\big)$$

$$= (cos(2\alpha)x_0 + sen(2\alpha)y_0 - 2sen(\alpha)c, sen(2\alpha)x_0 - cos(2\alpha)y_0 + 2cos(\alpha)c) \quad (3.9)$$

Por outro lado, sabe-se que a constante c em (3.9), definida anteriormente na equação da reta r, corresponde a:

$$b = \frac{c}{cos(\alpha)} \Leftrightarrow c = bcos(\alpha) \quad (3.10)$$

Dessa forma, substituímos a expressão (3.10) em (3.9) para determinar as coordenadas algébricas do ponto simétrico de X, isto é, as coordenadas de X':

$$S_r(x, y) = (cos(2\alpha)x_0 + sen(2\alpha)y_0 - sen(2\alpha)b, sen(2\alpha)x_0 - cos(2\alpha)y_0 + 2cos^2(\alpha)b) \quad (3.11)$$

Exemplo 3.3. *Obtenha o ponto simétrico de* $(2, 7)$ *em relação a reta* $r : y = \sqrt{3}x + 2$ *com base na lei de simetria (3.11).*

Solução:

Note na equação da reta r *que seu coeficiente angular e linear são* $a = \sqrt{3}$ *e* $b = 2$*, respectivamente. Como* $a = tg(\alpha) = \sqrt{3}$*, então* $\alpha = \frac{\pi}{3}$*. Com isso, basta aplicar os dados na expressão algébrica (3.11) para obter o ponto simétrico de* $(2, 7)$*, isto é:*

$$\begin{aligned} S_r(2,7) &= \left(2cos\left(2\frac{\pi}{3}\right) + 7sen\left(2\frac{\pi}{3}\right) - 2sen\left(2\frac{\pi}{3}\right), 2sen\left(2\frac{\pi}{3}\right) - 7cos\left(2\frac{\pi}{3}\right) + 4cos^2\left(\frac{\pi}{3}\right)\right) \\ &= \left(-1 + \frac{5\sqrt{3}}{2}, \frac{9}{2} + \sqrt{3}\right) \end{aligned}$$

É notório que sem a demonstração detalhada da lei de simetria para o caso de qualquer reta r do plano que os alunos do nono ano do ensino fundamental conseguirá aplicar o conhecimento, uma vez que o ensino da trigonometria já faz parte da matriz curricular dessa seriação. No entanto, nessa situação, em especial, a abordagem torna-se mais completa quando aplicada no segundo ano do ensino médio, onde o ensino da trigonometria costuma ser mais detalhado.

Desse fato, visualizamos que nem todo o ensino de reflexão se faz sentido dentro do ensino fundamental por necessitar de um conhecimento mais aprofundado da Matemática, seja por meio do ensino da trigonometria ou por meio do ensino de matrizes como apresentaremos a seguir.

As reflexões em relação aos eixos coordenados (Ox e Oy), a origem do sistema e em relação a primeira e a segunda bissetrizes podem ser representadas por uma transformação

com representação matricial como podemos ver no esquema a seguir.

1. Simetria em relação ao eixo Ox e a matriz de transformação $T = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$;

$$T(x, y) = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ -y \end{bmatrix}$$

2. Simetria em relação ao eixo Oy e a matriz de transformação $T = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$;

$$T(x, y) = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -x \\ y \end{bmatrix}$$

3. Simetria em relação a origem e a matriz de transformação $T = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$;

$$T(x, y) = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -x \\ -y \end{bmatrix}$$

4. Simetria em relação a primeira bissetriz e a matriz de transformação $T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$;

$$T(x, y) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} y \\ x \end{bmatrix}$$

5. Simetria em relação a segunda bissetriz e a matriz de transformação $T = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$.

$$T(x, y) = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} -y \\ -x \end{bmatrix} = \begin{bmatrix} y \\ x \end{bmatrix}$$

Uma observação importante ainda em relação as reflexões com base no pensamento de Medeiros [29], refere-se as reflexões em relação aos pontos fora da origem que na visão da autora é uma combinação de uma simetria com translações, isto é, basta transladarmos o centro de simetria para a origem e a partir daí realizarmos a simetria proposta em relação a nova origem e, finalmente transladamos o centro de simetria para sua origem inicial. Já, as reflexões em relação a uma reta qualquer é vista pela autora como uma combinação da simetria em relação ao eixo Ox e uma rotação.

Nesse sentido, é importante destacar o pensamento de Souza [36, p.35] ao afirmar que:

> A reflexão pode ser vista como um movimento rígido realizado no espaço, retirando a figura da folha de papel e fazendo-a rotacionar com relação ao eixo que utilizamos como "espelho" até que ela novamente encontre o papel, no outro semiplano.

Com base nesse pensamento, se faz necessário definir a rotação no plano e como se efetua uma rotação segundo a ótica da Matemática.

3.2.3 Rotação

Considere um ponto, O, fixo no plano Π e um ângulo de rotação α (convencionamos tradicionalmente o sentido anti-horário como positivo) com vértice em O. Dizemos que a rotação de centro O e amplitude α é uma transformação no plano Π, onde a transformação $R_\alpha : \Pi \to \Pi$ preserva distâncias e associa cada ponto X do plano, com $X \neq O$, o ponto $X' = R_\alpha(X)$ de forma que se tenha X' como imagem de X e os segmentos $\overline{OX}$ e $\overline{OX'}$ de mesma medida e $\alpha = X\widehat{O}X'$ com o sentido de X para X' [34] (Ver Figura 29).

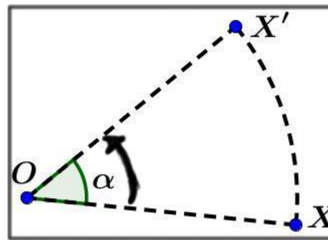


Figura 29: Rotação do ponto X com centro em O e amplitude α

Para todo p inteiro, as rotações de amplitudes $\alpha + p \cdot 360^o$ são idênticas. Nas situações para $0^o \leq \alpha \leq 360^o$, a rotação de amplitude $-\alpha$ é igual à rotação de amplitude $360^o - \alpha$.

Considere o plano cartesiano com $X = (x, y)$ definido, a rotação de centro definido inicialmente em $O = (0, 0)$ e amplitude α que transforma X no ponto $X' = (x', y')$.

No estudo dessa transformação suas coordenadas se apresentam melhor definidas na forma polar, isto é, as coordenadas do ponto X em relação a origem do plano cartesiano é expressa por $(x, y) = (rcos(\beta), rsen(\beta))$ em que r representa a distância da origem ao ponto X e β a amplitude do eixo x ao segmento Ox.

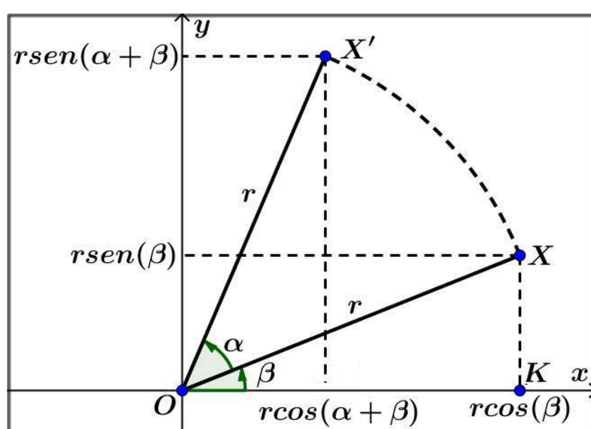
Para os discentes do 9º Ano do Ensino Fundamental, em especial, essa abordagem torna-se viável pelas razões trigonométricas considerando o triângulo retângulo OXK, onde K é o pé da perpendicular baixada do ponto X ao eixo Ox (ver Figura 30(a)).

1. **Rotação em torno da Origem:** Dado o ponto $X = (rcos(\beta), rsen(\beta))$ em que o segmento $\overline{OX} = r$ faz um ângulo β em relação ao eixo Ox, conforme a Figura 30(a).

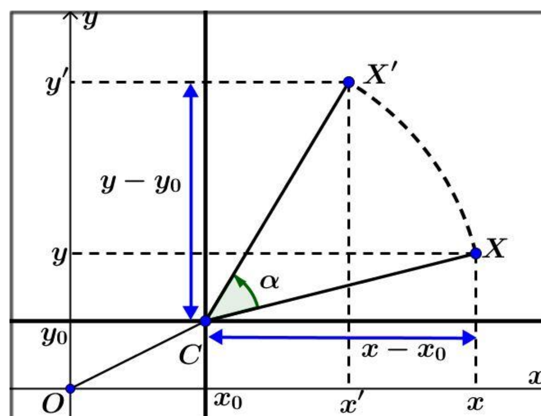
Sendo $X' = (x', y')$ a imagem de X pela rotação de amplitude α em relação ao centro na origem apresentará suas coordenadas dada por:

$$\begin{aligned}(x', y') = R_\alpha(x, y) &= (rcos(\alpha+\beta), rsen(\alpha+\beta)) \\ &= (r(cos\alpha cos\beta - sen\alpha sen\beta), r(sen\alpha cos\beta + sen\beta cos\alpha)) \\ &= (xcos\alpha - ysen\alpha, xsen\alpha + ycos\alpha) \qquad (3.12)\end{aligned}$$

2. **Rotação em torno de um ponto C:** As rotações fora da origem é determinada com uma rotação supondo uma nova origem cartesiana, para depois fazer uma translação. Nesse sentido, vamos considerar o centro de rotação $C = (x_0, y_0)$ como origem de um sistema cartesiano transladado x_0 unidades na abcissa e y_0 unidades na ordenada, ver Figura 30(b).



(a) Rotação em torno da origem O



(b) Rotação em torno do ponto C fora da origem

Figura 30: Rotação do ponto X e amplitude α com centro na origem e no ponto C

Dessa forma, considere que o ponto $X = (x - x_0, y - y_0)$ em relação a nova origem C como mostra a Figura 30(b), com isso a lei de transformação (3.12) assume a forma:

$$(x', y') = R_\alpha(x, y) = ([x - x_0]cos\alpha - [y - y_0]sen\alpha, [x - x_0]sen\alpha + [y - y_0]cos\alpha) \quad (3.13)$$

Contudo, as coordenadas do ponto X' conforme (3.13) considera o ponto C como origem do sistema cartesiano transladado. Para escrever as coordenadas desse ponto em relação à origem O adicionamos as coordenadas do ponto C, obtendo assim a expressão algébrica que representa a rotação do ponto X segundo uma amplitude α com centro em $C =$

(x_0, y_0), conforme mostra a Figura 30(b).

$$R_\alpha(x,y) = (x_0 + [x - x_0]cos\alpha - [y - y_0]sen\alpha, y_0 + [x - x_0]sen\alpha + [y - y_0]cos\alpha) \quad (3.14)$$

Exemplo 3.4. *Considere o centro de rotação $C = (2,3)$ e um ponto qualquer do plano xOy, isto é, $X = (x,y)$ tal que X' seja a imagem rotacionada do ponto X segundo um ângulo de 90°. Com base nessa informação, é possível inferir que a expressão que determinar o lugar geométrico de todas as possíveis imagens do ponto X é dada por:*

Solução:

Com base na lei de rotação (3.14) e nos dados do exemplo, a expressão será dada por:

$$\begin{aligned} R_\alpha(x,y) &= (2 + [x-2]cos(90^o) - [y-3]sen(90^o), 3 + [x-2]sen(90^o) + [y-3]cos(90^o)) \\ &= (2 - y + 3, 3 + x - 2) \\ &= (5 - y, 1 + x) \end{aligned}$$

Portanto, podemos definir que a transformação de rotação

$$R_\alpha(x,y) = (xcos\alpha - ysen\alpha, xsen\alpha + ycos\alpha)$$

é associada na forma matricial por:

$$R_\alpha(x,y) = \begin{bmatrix} cos\alpha & -sen\alpha \\ sen\alpha & cos\alpha \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} xcos\alpha - ysen\alpha \\ ysen\alpha + ycos\alpha \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Salienta-se que a matriz de transformação da rotação possui o determinante unitário, e isso implica que a transformação é isométrica, isto é, preserva distâncias, ângulos e áreas.

$$\begin{vmatrix} cos\alpha & -sen\alpha \\ sen\alpha & cos\alpha \end{vmatrix} = cos^2\alpha - (-sen^2\alpha) = cos^2\alpha + sen^2\alpha = 1$$

Nesse ponto ainda é possível definir que uma reflexão em relação a origem é exatamente igual à uma rotação de 180º centrada em O.

Como podemos confirmar, as transformações isométricas mantém preservadas tanto a distância, como os ângulos e as áreas, e isso, não é mantido nas demais transformações isomórficas e anamórficas.

3.3 Transformações Isomórficas

As transformações isomórficas na concepção de Lira [37] são as transformações de semelhança e homotetia, e possuem a propriedade de modificar o tamanho mantendo a forma da figura inicial com a mesma amplitude entre segmentos correspondentes.

3.3.1 Homotetia

A homotetia com centro O no plano Π e uma razão $k \neq 0$ é a transformação $H_{O,k} : \Pi \rightarrow \Pi$ que a cada ponto $X \in \Pi$ associa o ponto $X' = H_{O,k}(X)$, tal que

$$\overrightarrow{OX'} = k \cdot \overrightarrow{OX}$$

Com base nessa informação, considere $X = (x, y)$ e sua imagem $X' = (x', y')$ no sistema de coordenadas cartesianas do plano Π com uma figura Φ do mesmo plano, dizemos que a homotetia que associa a cada ponto X de Φ é dada por:

$$H_{O,k}(x, y) = (k \cdot x, k \cdot y) = X'$$

ou ainda, pelo sistema:

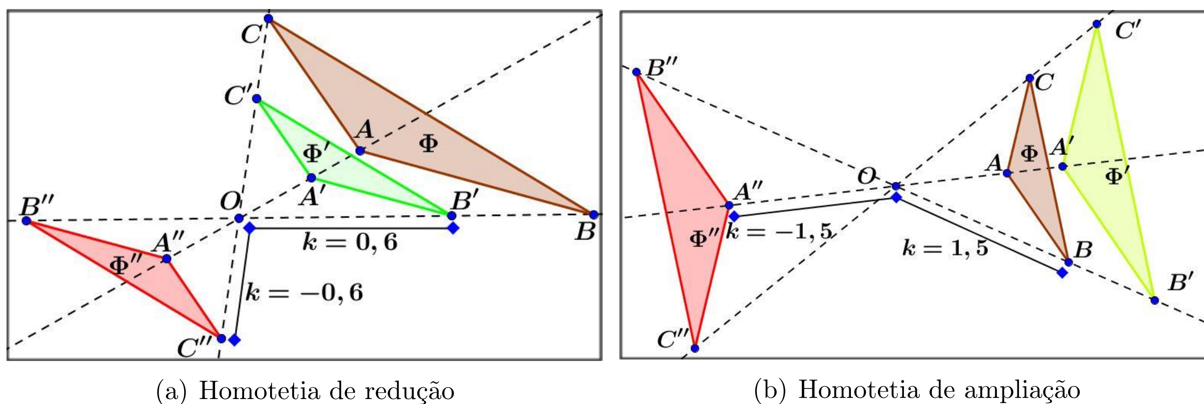
$$\begin{cases} x' = k \cdot x \\ y' = k \cdot y \end{cases}$$

Como k é um número real diferente de zero, as figuras semelhantes a Φ serão uma homotetia *direta* da figura inicial se $k > 0$ e *inversa* nos casos em que $k < 0$. E nas situações em que $k = 1$ ou $k = -1$ dizemos que a homotetia é uma transformação identidade (congruente) ou uma transformação simétrica em relação ao ponto de origem O (ou rotação de 180° em torno de O), respectivamente.

Ressalta-se ainda que:

- Se $\mathbf{0 < |k| < 1}$, as figuras Φ' resultantes dessa transformação serão reduzidas em relação à original (Φ), no mesmo sentido de Φ se $k > 0$ e sentido invertido se $k < 0$, ver Figura 31(a).
- Se $\mathbf{|k| > 1}$, as figuras Φ' serão resultantes de uma ampliação da figura Φ de mesmo sentido para $k > 0$ e sentido invertido para $k < 0$, ver Figura 31(b).

Dessa forma, concluímos que a homotetia é uma transformação que preserva apenas os ângulos construindo imagens semelhantes de razão $k \neq 0$. E por esse motivo a lei de



(a) Homotetia de redução (b) Homotetia de ampliação

Figura 31: Homotetia de redução ($0 < |k| < 1$) e ampliação $|k| > 1$

transformação

$$H_{O,k}(x, y) = (k \cdot x, k \cdot y) = (x', y')$$

é associada a matriz de transformação

$$H_{O,k}(x, y) = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} kx \\ ky \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

É notório que o determinante associado a matriz de transformação é dado por k^2, e isso, justifica o fato de não preservar distâncias e a área como acontece com as transformações isométricas.

3.4 Transformações Anamórficas

As transformações que deformam (ampliam ou reduzem) as figuras no sentido horizontal em proporção diferente do sentido vertical não podem ser consideradas homotetias como aponta Stormowski[6] e alguns autores classificam essa transformação como anamórfica [37].

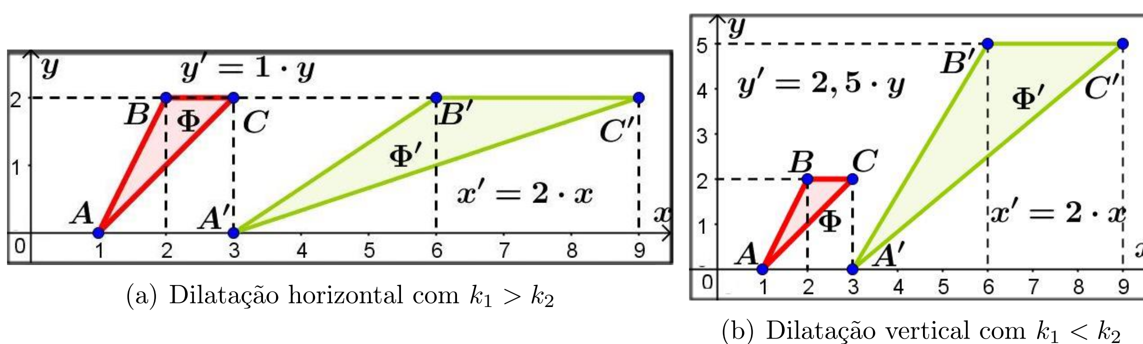
Podemos afirmar então que essa transformação quando aplicada a uma figura Φ do plano Π, modifica sua posição, tamanho e a própria forma original da figura. Por esse motivo, as funções $A : \Pi \to \Pi'$ onde para cada $X \in \Pi$, tem-se $A(X) = X'$ de forma que X' não conserva as características da figura original. Dentre as transformações anamórficas destacaremos a dilatação, sendo que existe ainda a contração vertical e horizontal e o cisalhamento.

3.4.1 Dilatação

A dilatação é uma transformação que gera deformação na direção do eixo Ox de acordo com um fator k_1, e na direção do eixo Oy pelo fator k_2. Dessa forma, as coordenadas do ponto X' obtida a partir de X é expressa por:

$$\begin{cases} x' = k_1 \cdot x \\ y' = k_2 \cdot y \end{cases}$$

Salienta-se que se $k_1 > k_2$, a dilatação será na horizontal, ver Figura 32(a). Do contrário, a dilatação será na vertical, ver Figura 32(b).



(a) Dilatação horizontal com $k_1 > k_2$

(b) Dilatação vertical com $k_1 < k_2$

Figura 32: Dilatação horizontal e vertical

Para essa transformação a matriz de transformação associada apresenta a seguinte lei:

$$A(x, y) = \begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} k_1 x \\ k_1 y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}$$

Dessa análise, concluímos as abordagens das transformações geométricas quanto ao quesito matemático e passaremos a apresentar uma abordagem integradora das transformações com a linguagem de programação *Processing*.

4 Transformações Geométricas em Processing

As transformações geométricas se relacionam com a computação gráfica segundo dois aspectos apontados por Velho [38]:

- **Mudança de Coordenadas** feita através de uma transformação do espaço Π e é utilizado para obter a devida formulação analítica do problema e;
- **Deformação de objetos do espaço Π**, tais deformações se dividem em duas classes:

  As deformações rígidas conhecidas por mudar a posição dos objetos no próprio espaço sem alterar suas relações métricas e são chamadas também de isometrias ou movimentos rígidos (secção 3.2) e;
  As deformações não-rígidas que caracterizam-se por alterar as relações métricas dos objetos e são conhecidas como transformações isomórficas (secção 3.3) e anamórficas (secção 3.4).

No entanto, a abordagem dessas transformações não revelou-se suficiente para compreender como se comporta a geometria na ótica da computação gráfica. Segundo Velho [38, p. 21], definir "o modelo adequado de geometria é importante para colocar e resolver corretamente os problemas tanto do ponto de vista conceitual quanto de implementação". Em função disso, o referido autor discute em seu livro de Fundamentos da Computação Gráfica qual deve ser a geometria adotada na computação gráfica apresentando o conceito de Geometria Axiomático de Euclides, a Geometria Analítica de René Descartes e o conceito de grupo de transformação introduzido por Felix Klein.

Esse autor recorre a alguns conceitos da Álgebra Linear, tais como Transformações Lineares (Apêndice A) e Transformações Ortogonais [1] para definir que a isometria possui uma caracterização euclidiana resultante de uma transformação linear ortogonal.

[1] Para mais detalhes consulte Velho [38, p. 27, 28].

Com isso, foi possível mostrar que a translação não é uma transformação linear, por não preservar as operações do espaço e nem faz parte da álgebra de transformações do espaço euclidiano, visto que a transformação é uma isometria, sem necessariamente ser Linear se, e somente se;

$$T(u) = L(u) + v_o$$

onde L é uma transformação linear ortogonal e v_o um vetor fixo.

Diante do exposto, concluímos que a Geometria Euclidiana não é a mais indicada para a computação gráfica. Entretanto, a implementação da Geometria Afim e da Transformação Afim[2] apresenta-se como uma excelente escolha por incluir nas transformações afins, os movimentos rígidos da Geometria Euclidiana, além de ser representado por matrizes que vislumbram uma estrutura computacional simples.

Contudo, somente com à Geometria Projetiva que foi possível resolver o problema no processo de visualização de uma transformação fotográfica por não preservar retas paralelas em uma estrada quando analisamos uma fotografia.

(a) (b)

Figura 33: Fotografia de uma estrada

A Geometria Projetiva do ponto de vista de Velho [38] é uma extensão natural da Geometria Afim que por sua vez estende o campo de atuação da Geometria Euclidiana. E a partir do conceito de visualização, esse autor define o espaço projetivo como sendo uma *projeção cônica* e isso justifica o fato das retas paralelas da estrada (Figura 33(b)) se encontrarem em um ponto localizado no horizonte para onde convergem os lados paralelos, visto que no espaço projetivo não existem retas paralelas.

A necessidade de definir qual a geometria adotada pela computação gráfica encontra-se no fato de definir uma forma única para representar a translação que é a soma de vetores, a rotação e a escala (simetria) que é a multiplicação de um vetor por uma matriz.

A solução para representa-las de forma única encontra-se nas coordenadas homogêneas das transformações projetivas (Apêndice B) e como cada transformação possui sua

[2]Ver Velho [38, p. 28– 35]

propriedade característica, podemos concluir que as matrizes para as transformações em coordenadas homogêneas são dadas por:

$$T(x,y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}, E(x,y) = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad e \quad R(x,y) = \begin{bmatrix} cos(\alpha) & -sen(\alpha) & 0 \\ sen(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

Onde as transformações T, E e R são respectivamente, translação, escala[3] e rotação.

No caso das simetrias em relação a primeira e a segunda bissetriz, respectivamente, a matriz de transformação E será dada por

$$E(x,y) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad e \quad E(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Observa-se com isso, qual a geometria adotada na Computação Gráfica e como esse sistema construi as TGP na tela. Daí, passamos a apresentar as ferramentas do *Processing*, responsáveis por tais efeitos de design gráfico.

4.1 Transformações isométricas no *Processing*

Conforme apresentamos na secção 3.2, as transformações isométricas são: Translação, Simetria e Rotação. Diante disso, vamos apresentar como o *Processing* executa cada uma dessas transformações e o comando necessário para tal efeito gráfico.

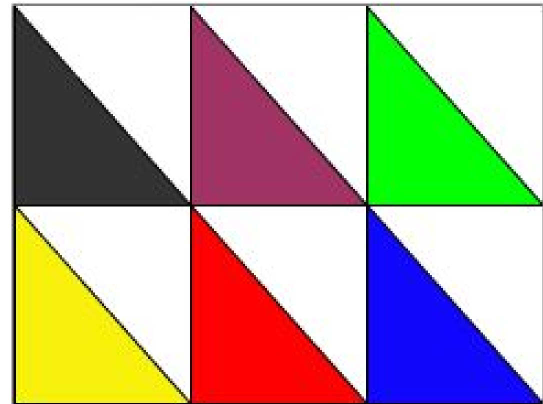
Começaremos pela translação cuja função é ativa pelo termo **translate ()**. Essa função move a origem da figura do canto superior esquerdo da tela para outro ponto. Seu deslocamento gráfico consiste em descolar seus parâmetros x e y, nessa ordem.

Dessa forma, sua função é chamada com a sintaxe $\boldsymbol{translate(x, y)}$ que adiciona os valores dos parâmetros x e y a quaisquer formas desenhadas após a execução da função. A translação viabiliza a construção de figuras no estilo das obras de Escher, como mostra a Figura 34(b) construída com o código da translação de um triângulo retângulo com um dos pontos na origem, ver Figura 34(a).

[3]Escala são as transformações de reflexões, assim como as homotetias, dilatações entre outros efeitos anamórficos para a computação gráfica.

```
size(240,185);
background(255);
fill(50); //preta
//Triângulo na origem (0, 0) cor "preta"
triangle(0,0,0,90,80,90);
translate(80,0);
 fill(160,50,100); // violeta
 //Translada origem para (80,0) cor "violeta"
 triangle(0,0,0,90,80,90);
translate(-80,90);
 fill(247,240,10); //amarelo
 //Translada origem para (0,90) cor "amarelo"
 triangle(0,0,0,90,80,90);
translate(80,0);
 fill(255,0,0); //vermelho
 //Translada origem para (0,90) cor "vermelho"
 triangle(0,0,0,90,80,90);
translate(80,0);
 fill(15,6,252); //azul
 //Translada origem para (0,90) cor "azul"
 triangle(0,0,0,90,80,90);
translate(0,-90);
 fill(0,255,0); // verde
 //Translada origem para (160, 0) cor "verde"
 triangle(0,0,0,90,80,90);
```

(a) Comando com função $translate(x, y)$



(b) Translação de triângulo

Figura 34: Construção de triângulos retângulos com efeito de translação

Nota-se que o código de translação oscila deslocando a última imagem construída para uma nova posição. Nesse sentido, o programa oferece uma ferramenta intitulada de $pushMatrix()$ e $popMatrix$ que são incorporada nas transformações para ampliar seu alcance. A ferramenta $pushMatrix$ funciona basicamente salvando o atual sistema de coordenadas para a pilha matricial enquanto a função $popMatrix$ restaura o sistema antes de deslocar as coordenadas como podemos perceber na Figura 35.

Se analisarmos o código de translação da Figura 35, é possível constatar que o comando $translate(80, 0)$; é o mesmo para todos os triângulos na mesma linha modificando apenas quando transladamos o triângulo inicialmente de preto para a segunda linha da imagem, para voltar ao mesmo comando de translação anterior, conforme mostra a Figura 34(b).

A função de Simetria é ativada no *Processing* pela sintaxe $\boldsymbol{scale(x, y)}$ que reflete a figura em relação aos eixos x e y, quando a origem dos mesmo é deslocada para o meio da tela ao associar os valores 1 ou -1 para os parâmetros dos eixos.

Sendo assim, para construir uma imagem simétrica no *Processing* precisamos inicialmente de uma figura, preferencialmente, relacionada com o centro da tela que pode ser obtida com uma translação para enfim executar a simetria ora em relação ao eixo das abscissas, ora em relação aos eixos das ordenadas, ou ainda em relação a origem do

```
size(240, 185);
background(255);
fill(50);
triangle(0, 0, 0, 90, 80, 90); //Triângulo na origem (0, 0) cor "preta"
pushMatrix();
translate(80, 0); //Translada origem para (80,0) cor "violeta"
pushMatrix();
translate(80, 0); //Translada origem para (160, 0) cor "verde"
fill(160, 50, 100); // violeta
triangle(0, 0, 0, 90, 80, 90);
popMatrix();
fill(0, 255, 0); // verde
triangle(0, 0, 0, 90, 80, 90);
popMatrix();
translate(0, 90); //Translada origem para (0,90) cor "amarelo"
fill(247, 240, 10); // amarelo
triangle(0, 0, 0, 90, 80, 90);
pushMatrix();
translate(80, 0); //Translada origem para (80,90) cor "vermelho"
pushMatrix();
translate(80, 0); //Translada origem para (160,90) cor "azul"
fill(5, 6, 252); // Azul
triangle(0, 0, 0, 90, 80, 90);
popMatrix();
fill(255, 0, 0); // Vermelho
triangle(0, 0, 0, 90, 80, 90);
popMatrix();
```

Figura 35: Código de Translação com a função $pushMatrix$ e $popMatrix$

sistema cartesiano deslocado.

Com base nessa informação, consideremos o eixo $x'O'y'$ com origem deslocada para o centro da tela $(150, 150)$ e um triângulo no quarto quadrante (cor "roxa"). Para transladar esse triângulo em torno dos novos eixos, x' e y', e em torno da nova origem O' utilizamos a linha de comando como mostra na Figura 36(a) para obtermos o efeito de uma imagem similar a um escudo medieval, ver Figura 36(b).

Ao executar os comandos relacionados da Figura 36(a) constatamos que a ordem de lançamento dos comandos interfere diretamente nas imagens simétricas proporcionada e a justificativa é facilmente obtida no próprio programa ao lançar o comando $printMatrix()$; que imprime no console do aplicativo a matriz de transformação associada a simetria transladada. Para cada simetria existe um produto na matriz de transformação associada utilizando o contexto de composição de transformações.

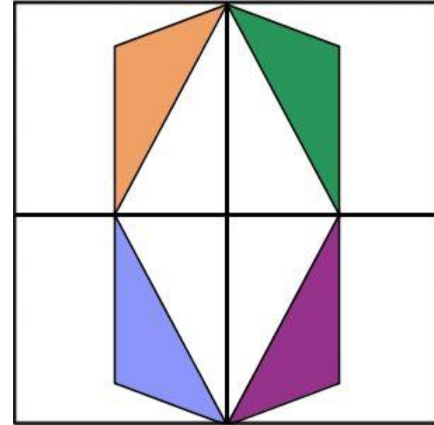
Para evitar esse efeito podemos inserir os comandos de simetria, com o preenchimento e as formas geométricas entre cada $pushMatrix()$ e $popMatrix()$ como aparece abaixo, referenciando a simetria esperada diretamente no comando $scale$.

$pushMatrix()$;
$scale(-1, 1)$; //cor "azul"
$fill(140, 150, 250)$;
$triangle(0, 150, 80, 120, 80, 0)$;
$popMatrix()$;

$pushMatrix()$;
$scale(1, -1)$; //cor "verde"
$fill(40, 150, 90)$;
$triangle(0, 150, 80, 120, 80, 0)$;
$popMatrix()$;

```
size(300,300);
background(255);
strokeWeight(3);
line(0,150,300,150);//Eixo O'x'
line(150,0,150,300);//Eixo O'y'
strokeWeight(1.5);
//Nova Origem do sistema (150,150)
translate(150,150);
fill(150,50,140);
//triângulo inicial cor "roxa"
triangle(0,150,80,120,80,0);
pushMatrix();
scale(-1,1); //cor "azul"
pushMatrix();
scale(-1,-1); //cor "verde"
pushMatrix();
scale(-1,1); // cor "laranja"
printMatrix();
//simétrico em relação à origem cor "laranja"
fill(240,160,100);
triangle(0,150,80,120,80,0);
popMatrix();
//simétrico em relação ao eixo Ox cor "verde"
fill(40,150,90);
triangle(0,150,80,120,80,0);
popMatrix();
//simétrico em relação ao eixo Oy cor "azul"
fill(140,150,250);
triangle(0,150,80,120,80,0);
popMatrix();
```

(a) Comando da função $scale(x, y)$ com $x, y \in \{-1, 1\}$



(b) Simetria de triângulo

Figura 36: Construção de triângulos com efeito de simetria

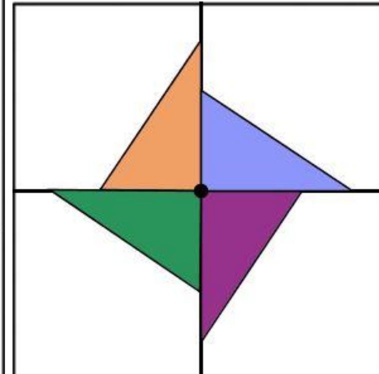
Para as transformações de rotação no plano o programa utiliza a sintaxe **rotate($\boldsymbol{\theta}$)**, onde θ é o ângulo dado em radianos e o sentido positivo da rotação é o horário, já o anti-horário na linguagem de programação é o sentido negativo. Nesse sentido, para o aluno compreender os efeitos da rotação será necessário algumas noções de conversão de ângulo de grau para radianos e vice-versa. Além disso, para visualizar a rotação é necessário que a imagem esteja prioritariamente na proximidade do centro da tela através do recurso de translação para centralizar a construção.

Sendo assim, considere um triângulo deslocado para a nova origem $(150, 150)$, cor roxa, e as rotações de amplitude $\frac{3\pi}{2}$, π, $\frac{\pi}{2}$ radianos. Com os comandos $pushMatrix()$ e $popMatrix()$, o sistema armazena os códigos, ver Figura 37(a), e executa no sentido inverso da composição das transformações, isto é, da última linha do comando $popMatrix()$ até a última entrada no $pushMatrix()$, o resultado destes comandos gera uma imagem similar ao cata-vento de papel, ver Figura 37(b).

Cada nova rotação executa um produto na matriz homogênea da transformação deslocando a última peça rotacionada. As transformações sucessivas nesta linguagem interagem-se na ordem inversa para a construção como mostra o fluxograma. [38(b)], as construções com o comando *pushMatrix();* e *popMatrix();* executa a última construção

```
size(300,300);
background(255);
strokeWeight(3);
line(0,150,300,150);//Eixo O'x'
line(150,0,150,300);//Eixo O'y'
strokeWeight(1.5);
translate(150,150);//Nova Origem do sistema (150,150)
fill(150,50,140);
triangle(0,0,0,120,80,0);//triângulo inicial cor "roxa"
pushMatrix();
rotate(3*PI/2); //cor "azul"
pushMatrix();
rotate(PI); //cor "verde"
pushMatrix();
rotate(PI/2); // cor "laranja"
printMatrix();
fill(240,160,100); //rotação de ângulo 90º cor "laranja"
triangle(0,0,0,120,80,0);
popMatrix();
fill(40,150,90); //rotação de ângulo 180º cor "verde"
triangle(0,0,0,120,80,0);
popMatrix();
fill(40,150,90); //rotação de ângulo 180º cor "verde"
triangle(0,0,0,120,80,0);
popMatrix();
fill(140,150,250); //rotação de ângulo 270º cor "azul"
triangle(0,0,0,120,80,0);
popMatrix();
fill(0);
ellipse(0,0,10,10);
```

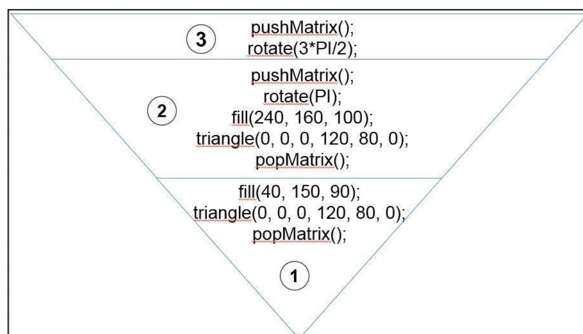
(a) Comando com função $rotate(\theta)$ com $0 \leq \theta \leq 2\pi$



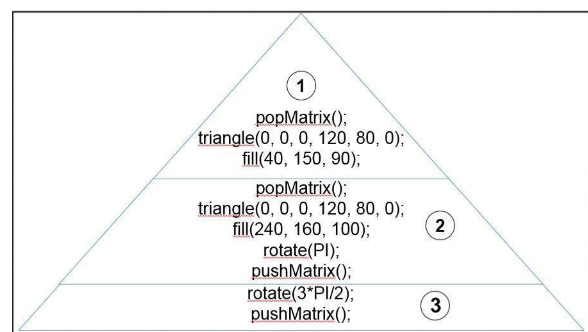
(b) Rotação de triângulo

Figura 37: Construção de triângulos com efeito de rotação

como se estivesse a remover uma peça da torre de hánoi.



(a) Fluxograma dos códigos no *Processing*



(b) Fluxograma na ordem de construção

Figura 38: Fluxograma com esquema de uma rotação no *Processing*

A inclusão dos comandos de preenchimento e formas geométricas entre $pushMatrix()$ e $popMatrix()$ evitará o efeito do produto entre as matrizes como acontece com a simetria anterior, entretanto, será necessário redefinir uma nova ordem para as amplitudes.

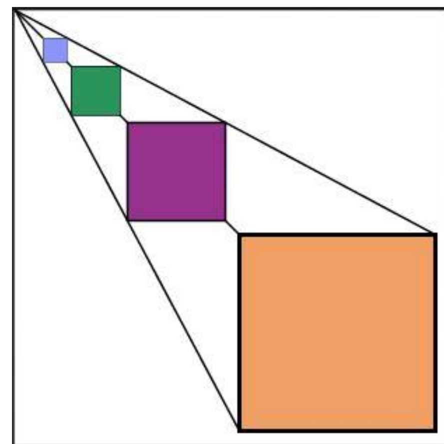
4.2 Transformações isomórficas no *Processing*

Conforme apresentamos na secção 3.3, a homotetia é uma transformação isomórfica. E, vamos apresentar como o *Processing* executa esta transformação com o comando $scale(k, k)$; com $k \in \mathbb{R}_+$.

Em função da limitação visual da linguagem apenas no quadrante positivo do sistema, ou seja, no sistema global da tela, a constante k está definida para todo número real positivo. Neste sentido, toda a construção de ampliação será dada por $k > 1$ e a redução por $0 < k < 1$. Salienta-se ainda que múltiplas homotetias são obtida pelo produto das matrizes de transformação associadas a este efeito visual.

```
size(310,310);
background(255);
strokeWeight(1.5);
//linhas de orientação para homotetia
line(0,0,300,300);
line(0,0,300,160);
line(0,0,160,300);
fill(150,50,140);
//quadrado inicial cor "roxa"
rect(80,80,70,70);
pushMatrix();
scale(0.25,0.25); //cor "azul"
printMatrix();
pushMatrix();
scale(2,2); //cor "verde"
printMatrix();
pushMatrix();
scale(4,4); // cor "laranja"
//ampliação fator "k=2"  cor "laranja"
fill(240,160,100);
rect(80,80,70,70);
popMatrix();
//redução fator "k=0,5" cor "verde"
fill(40,150,90);
rect(80,80,70,70);
popMatrix();
//redução fator "k=0,25" cor "azul"
fill(140,150,250);
rect(80,80,70,70);
popMatrix();
```

(a) Homotetia com fatores $k = \{0, 25; 2; 4\}$



(b) Homotetia no quadrado

Figura 39: Construção de quadrados com efeito de ampliação e redução

Desta forma, considere o código 39(a) da Figura 39(b). A transformação foi construída utilizando constantes $k = \{0, 25; 2; 4\}$, no entanto, suas imagens possuem, respectivamente, razões $\{0, 25; 0, 5; 2\}$ em função do produto das matrizes de transformação associadas a cada etapa da construção. Por este motivo, compete aqui salientar a impor-

tância na ordem dos dados a serem “alimentados” na linguagem para o aplicativo funcionar dentro dos conformes desejados e na forma de implementar as funções $pushMatrix()$ e $popMatrix()$ para evitar o efeito de produtos matriciais.

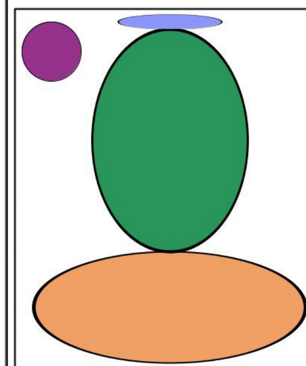
Com a combinação de uma translação na homotetia, o ponto de origem desta transformação seria deslocada para as coordenadas da translação desejada e, consequentemente, toda a construção será transladada mantendo as devidas proporções.

4.3 Transformações anamórficas no *Processing*

Para finalizar a apresentação das ferramentas da linguagem *Processing* para o estudo das transformações planas, iremos utilizar a função $scale(k_1, k_2)$ para construir dilatações no sentido horizontal e vertical considerando sempre $k_1 \neq k_2$. Neste sentido, a dilatação provocará uma distorção na imagem no sentido horizontal, quando $k_1 > k_2$, e no caso, de $k_1 < k_2$, a distorção da dilatação será vertical, como podemos perceber no código 40(a) e na imagem 40(b).

```
size(450,500);
background(255);
strokeWeight(1.1);
fill(150,50,140);
ellipse(60,60,80,80);//circulo inicial cor "roxa"
pushMatrix();
translate(150,10);
scale(1.75,0.25); //cor "azul"
pushMatrix();
translate(-20,80);
scale(1.5,15); //cor "verde"
pushMatrix();
translate(-30,80);
scale(1.75,0.5);// cor "laranja"
fill(240,160,100);//ampliação "k1=4,59 e "k2=1,87  cor "laranja"
ellipse(40,40,80,80);
popMatrix();
fill(40,150,90);//dilatação "k1=2,62 e k2=3,75" cor "verde"
ellipse(40,40,80,80);
popMatrix();
fill(140,150,250);//dilatação "k1=1,75 e k2=0,25" cor "azul"
ellipse(40,40,80,80);
popMatrix();
```

(a) dilatação horizontal e vertical



(b) dilatação do círculo

Figura 40: Construção de círculos com dilatação horizontal e vertical

Com o conhecimento matemático sobre as TGPs e com a linguagem de programação *Processing* sintetiza este conhecimento, iremos abordar um conjunto de atividades para serem executadas pelos discentes do nono Ano da Unidade Escolar Amélia Rodrigues, no referido laboratório de informática Unidade Escolar.

Em função da limitação didática dos alunos para compreenderem com domínio

as principais operações matemáticas envolvendo a TGP, abordaremos a temática utilizando a própria linguagem e os efeitos que tais transformações executam na projeção de imagens. Desta forma, objetivamos uma aprendizagem prática visando a manipulação em detrimento da teoria pela teoria, isto é, os envolvidos no programa reconheceram as transformações através de seus efeitos visuais e assim poderão classificar o efeito gráfico sem utilizar a conceituação matemática, principalmente no que discerne a utilização de matrizes e transformações lineares.

5 *Atividades Propostas*

Neste capítulo, apresentaremos uma sequência de atividades didáticas referentes à construção do conceito das TGPs. Para a elaboração das atividades, utilizaremos a linguagem *processing* com base nas informações constante nos capítulos 2 e 4 suprimindo ao máximo a linguagem matemática do capítulo 3, em função da própria limitação didática dos envolvidos, sem perder de vista sua interpretação geométrica que é o foco desta pesquisa. Sendo assim, as atividades que propomos é viabilizada pela mobilização das formas do pensamento matemático que são, nas palavras de Lage [39, p.6]:

> [...] modos de pensar que contribuem para desenvolver as capacidades de raciocinar, testar, experimentar, procurar relações, descobrir, comunicar e nas atividades investigativas, que incentivam a curiosidade, o interesse e a perserverança dos alunos por meio da cultura da exploração e investigação matemática.

Dentre os hábitos de pensamento matemático, a autora mencionada sinaliza cinco hábitos primordiais fundamentando-se em Cuoco, Goldengerb e Mark (1994), Davis e Hersh (1995), que devemos despertar nos estudantes:

1. *Reconhecer padrões ou invariantes escondidos em uma situação matemática.* Neste sentido, qualquer conteúdo pode ser explorado.

2. *Fazer experiências e explorações.* A busca por padrões e invariantes motiva os alunos e a exploração juntamente com a descoberta assumem um papel fundamental em sua aprendizagem.

3. *Visualização.* Este hábito visa à capacidade de criar, manipular e “ler” imagens mentais de aspectos comuns da realidade.

4. *Pensar, desmontar ideias, ser inventor.* Neste quesito o aluno é capaz de examinar um sistema já existente e fazer as devidas alterações em algum aspecto.

5. *Fazer conjecturas sobre as situações matemáticas propostas.* Neste ponto, o aluno é capaz de supor que algo é verdadeiro com base nas evidências preexistentes e então investigar acerca da sua veracidade.

Com o intuito de proporcionar uma modificação nos hábitos de pensamento matemático, vamos implementar uma seguência didática consistente na exploração das TGPs apresentadas neste material, com o objetivo de gerar imagens estáticas e dinâmicas (animações). Além de construir algumas telas de obra de arte digital a partir de processos iterativos com o uso da linguagem de programação *Processing*.

5.1 Explorando o aplicativo

Neste bloco de atividades figuram aquelas de primeiro contato dos alunos com a linguagem de programação em estudo, e a importância de seguir uma ordem no fluxograma da construção para não modificar a aplicação objetivada. A proposta dessas atividades consiste em oportunizar os alunos a familiarizar-se com as ferramentas disponíveis no *software*.

Objetivo

Levar o aluno a perceber a importância de um fluxograma na criação de aplicações utilizando linguagem de programação e mobilizar algumas formas gerais de pensamento matemático como: aprender a fazer experiências e explorações, incentivar a interpretação e a visualização nas investigações matemáticas.

Conteúdo Programático

Linguagem de Programação; Plano Cartesiano

Subsídio Teórico

O Plano Cartesiano foi criado pelo filósofo, físico e matemático francês René Descartes que durante a Idade Moderna era conhecido por seu nome latino *Renatus Cartesius*. Sua criação foi projetada com o intuito de localizar pontos num determinado espaço e graças a sua façanha matemática, a geometria analítica teve seu marco histórico.

Para localizar pontos no contexto da linguagem de programação, consideremos duas semirretas com origem no ponto O, no canto superior do ecrã, onde os sentidos horizontal para direita e vertical para baixo como representam o eixo das abscissas (eixo $0x$) e o eixo das ordenadas (eixo Oy), respectivamente.

Metodologia

Aula expositiva experimental com participação dos alunos; E exploração da linguagem de programação *Processing* com os estudantes agrupados em duplas ou trios, a depender da relação quantidade de alunos e computadores disponível no laboratório de informática.

Material

Papel, lápis ou caneta e computador com o *Processing* instalado.

Procedimento

Configure a dimensão e o plano de fundo da tela, logo em seguida, construa elementos geométricos, tais como: linhas, quadriláteros, triângulos, círculos, entre outros, adicionando uma espessura ao contorno e o preenchimento das figuras, conforme consta nos códigos da Parte I e organize os dados informados na Parte II dessa atividade para reproduzir a imagem apresentada.

Parte I

a) Abra um arquivo no *Processing* e grave como: **Applet 1 nome da dupla ou trio**.

b) Configuração da área de trabalho do aplicativo.
size(largura, altura); //Defina dois valores para resolução da tela ("Tamanho").
background(k); //Defina um valor entre $[0, 255]$ para a cor do plano de fundo.

c) Construção de formas geométricas básicas.
line(x_1, y_1, x_2, y_2); //Defina os valores das coordenadas iniciais e finais da linha.
triangle($x_1, y_1, x_2, y_2, x_3, y_3$); //Defina os valores dos vértices do triângulo.
point(x, y); //Desenhe pontos variando suas coordenadas.
rect(x, y, comprimento, altura); //Desenhe um retângulo.
ellipse(x, y, comprimento, altura); //Desenhe uma elipse.
Observe que a construção do retângulo e da elipse apresenta a mesma sequência de dados, onde os dois primeiros parâmetros definem a posição nas coordenadas x e y e os dois últimos definem o comprimento e a altura da figura.

d) Configuração das cores e contorno.
fill(R, G, B); //Pinte as formas construídas atribuindo valores entre $[0, 255]$ para determinar os tons de R–vermelho, G–verde e B–azul.
stroke(R, G, B); //Altere a cor dos contornos da mesma forma que atribuiu o

preenchimento das mesmas.
strokeWeight(n); //Escolha uma espessura para os contornos ou cada contorno das formas.

e) Execute o Applet estático a cada passagem de um item para o outro e observe as modificações na aplicação dos comandos. Organize os dados do item “d” entre cada forma geométrica do item “c” modificando seu preenchimento, contorno e espessura de forma a evidenciar graficamente cada elemento.

Parte II

Objetivo

Além dos objetivos anteriores, ressalta-se os seguintes objetivos. Conhecer o sistema de coordenadas no contexto da linguagem de programação e identificar a ordem de construção dos elementos gráficos.

Material

Papel, lápis ou caneta e computador com o *Processing* instalado.

Procedimento

Inicialmente, utilize uma folha quadriculada para representar as construções do item c, dessa atividade, antes de iniciar a construção no programa para determinar qual a melhor forma de organizar os dados com o intuito de gerar a imagem fornecida pelo docente e ao final determinar a espessura e cor do preenchimento constante no item d.

Configure a tela e o plano de fundo conforme solicitado e organize os dados para reproduzir a mesma imagem da Figura 41.

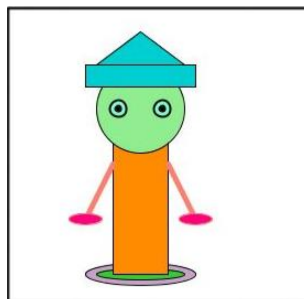


Figura 41: Applet 2: Boneco geométrico

a) Abra um novo arquivo e grave-o como: **Applet 2 nome da dupla ou trio**.

b) Configuração da tela.

```
size(300, 300);
background(255);
```

c) Dados das figuras planas do Applet boneco geométrico.

```
rect(20, 60, 100, 20);                ellipse(70, 250, 100, 20);
triangle(30, 60, 70, 30, 110, 60);    ellipse(70, 100, 80, 80);
ellipse(50, 100, 5, 5);               ellipse(70, 250, 80, 10);
ellipse(50, 100, 15, 15);             ellipse(90, 100, 15, 15);
rect(45, 130, 50, 120);               line(45, 100, 20, 200);
line(95, 150, 120, 200);              ellipse(20, 200, 30, 10);
ellipse(120, 200, 30, 10);
```

d) Dados das cores e espessura do contorno do Applet boneco geométrico.

```
fill(200, 162, 200); – Lilás              fill(255, 140, 0); – Laranja escruo
fill(144, 238, 144); – Verde claro        stroke(250, 128, 114); – Salmão
fill(50, 205, 50); – Verde lima           fill(0); – Preto
strokeWeight(2);                          fill(0, 206, 209); – Turquesa escura
fill(127, 255, 212); – Água-marinha       stroke(10); – Preto
fill(255, 0, 127); – Rosa brilhante       fill(255, 150, 0);
strokeWeight(5);                          strokeWeight(1);
```

Avaliação

Avaliar se todas as equipes conseguiram desenvolver as atividades e, se necessário, retomar as construções para avaliar os possíveis entraves encontrados na construção das imagens. Além de avaliar a transposição dos dados da malha quadriculada para a linguagem de programação.

5.2 Isometria de Translação

Esta secção das atividades propostas concentrará a transformação de isometria cuja característica fundamental encontra-se no deslocamento dos objetos, a chamada isometria de translação. Dessa forma, esperamos que os alunos possam reconhecer e identificar suas características, e peculiaridades com o apoio do professor que poderá trazer exemplos diversos da transformação em questão antes de iniciar a linguagem de programação para os alunos fazerem suas conjecturas iniciais.

Com isso, podemos inferir que o objetivo central dessa secção fundamenta-se em apresentar algumas isometrias de translação aos alunos, de modo que identifiquem peculiaridades, características e regularidades nas construções via linguagem de programação.

5.2.1 Conhecendo a translação

Objetivo

Construir o conceito de translação de uma figura bidimensional e fazer com que os alunos percebam o movimento da imagem na tela do "Applet" como um deslocamento de acordo com uma direção, sentido e distância, conforme a manipulação de um vetor expresso no código da transformação.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Translação

Subsídio Teórico

Transladar é mover o objeto no plano sem rotacionar, ou ainda, transladar consiste em deslocar a figura em a unidades na direção horizontal e b unidades na direção vertical. Em linguagem de programação, este recurso é necessário para mover a origem do canto superior esquerdo da tela para outro local, a depender de seus dois parâmetros mencionados e esses parâmetros são adicionados a qualquer desenhada após a função ser executada.

Metodologia

Aula expositiva com realização do Applet; E exploração do código de translação com os estudantes agrupados em duplas ou trios.

Material

Computador com o *Processing* instalado.

Procedimento

Configure a dimensão e o plano de fundo da tela, logo em seguida, execute os códigos de translação observando os efeitos no deslocamento das figuras.

a) Abra um arquivo no *Processing* e grave como: **Applet 3 nome da dupla ou trio**.

b) Digite os seguintes códigos de configuração do programa.
$size(500, 200);$
$background(241, 250, 0);$ //Plano de fundo no padrão (R, G, B).

c) Construção do triângulo equilátero interno ao quadrado.
$fill(200, 100, 200);$//Definição da cor do quadrado.

$rect(0, 20, 150, 150)$; //Quadrado de lado 150 pixels iniciando do ponto $(0, 20)$.
$fill(10, 250, 255)$; //Definição da cor do triângulo.
$triangle(0, 170, 150, 170, 75, 40)$; //Triângulo equilátero de lado 150 pixels.

d) Translade toda a construção (triângulo e quadrado) para a posição $(160, 20)$ conforme a instrução.
$translate(160, 20)$; //Desloca 160 pixels à direita e 20 para baixo.
$fill(200, 100, 200)$;
$rect(0, 20, 150, 150)$;
$fill(10, 250, 255)$;
$triangle(0, 170, 150, 170, 75, 40)$;

e) Ao acrescentar o deslocamento translate$(160, -20)$, quem será deslizada, a figura original ou a última imagem deslocada? Para descobrir acrescente o comando translate$(160, -20)$ e repita o código de construção do quadrado e do retângulo mantendo a mesma ordem e procedimento.

Avaliação

Avaliar se o aluno compreende como a translação é executada e se consegue distinguir qual imagem foi deslocada a cada acréscimo do código da programação em questão para obter a imagem da Figura 42. E assim, identificar a importância do acréscimo da função de pilhagem de matrizes nas construções envolvendo transformações geométricas que abordaremos na próxima atividade.

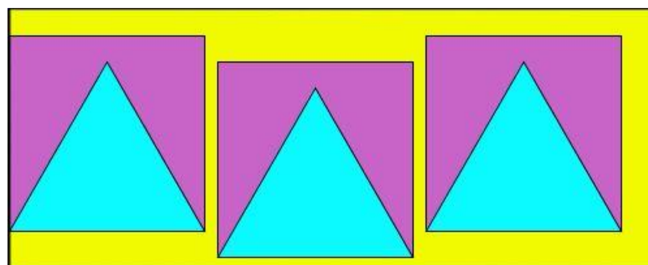


Figura 42: Applet 3: Translação primária

5.2.2 Aprimorando a translação

Objetivo

Aprofundar o conhecimento de translação através da utilização das funções *pushMatrix()* e *popMatrix()* e automatizar processos mais elaborados de uma transformação de translação.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Translação

Subsídio Teórico

No Applet 3 podemos perceber como as aplicações acumulam suas matrizes de transformação e para evitar este efeito "cumulativo" utilizamos os códigos $pushMatrix()$ e $popMatrix()$. Enquanto, a função $pushMatrix()$ registra o estado atual de todas as transformações, a função $popMatrix()$ retorna ao estado anterior à função $pushMatrix()$.

Na prática, essas funções associam cada matriz da transformação como uma folha de papel com a lista atual de transformações (translação, rotação, escala) escrito na superfície. Para salvar a matriz de transformação atual para uso posterior basta adicionar uma nova folha de papel na parte superior da pilha e copiar as informações da folha anterior com as alterações desejadas, preservando assim as informações da folha abaixo. Quando desejar retornar a matriz anterior basta remover o topo da folha para revelar as transformações guardadas.

Nesse sentido, salienta-se que cada *pushMatrix()* é associado a uma *popMatrix()* e uma função não pode ser utilizada sem a presença da outra. E no caso de não necessitar mais da pilha de transformação é possível desfazer utilizando a função *resetMatrix()* que limpa todas as funções de transformações até então definidas restabelecendo a matriz identidade da transformação.

Metodologia

Aula expositiva com realização do Applet; E compete aos estudantes construir o Applet experimentando os efeitos gráficos em sintonia com a teoria da transformação para enriquecer as possibilidades nas construções de novos aplicativos.

Material

Computador com o *Processing* instalado.

Procedimento

Para construir um prato com um omelete e uma torrada para o café da manhã podemos utilizar basicamente as funções *rect* e *ellipse*. No entanto, para servir vários pratos idênticos podemos transladar toda a construção para novas coordenadas facilmente utilizando a função *translate(a,b)* combinada com as funções de pilhagem de matrizes (*pushMatrix()* e *popMatrix()*).

Com o intuito de preparar um café da manhã execute o sistema de códigos da atividade.

a) Abra um novo arquivo e grave-o como: **Applet 4 nome da dupla ou trio**.

b) Configuração da tela.
size(500, 500);
background(144, 255, 244); //Plano de fundo azul claro

c) Dados do Applet café da manhã.

//Prato branco
1 $fill(240, 250, 255)$;
2 $ellipse(120, 120, 220, 220)$;
3 $ellipse(120, 120, 200, 200)$;
//Colher
4 $fill(201, 201, 201)$;
5 $rect(10, 70, 70, 10)$;
6 $rect(60, 60, 20, 30)$;
//Clara do ovo
7 $fill(255)$;
8 $ellipse(120, 70, 100, 50)$;
//Gema do ovo
9 $fill(255, 255, 0)$;
10 $ellipse(120, 70, 30, 30)$;
//Torrada
11 $fill(247, 113, 60)$;
12 $rect(150, 100, 50, 70)$;
13 $rect(160, 110, 30, 50)$;

d) Translação com a função *pusMatrix()* e *popMatrix()* de acordo com os dados do Applet.

1$pushMatrix()$;
2$translate(250, 0)$;
//Prato branco
3 $fill(240, 250, 255)$;
4 $ellipse(120, 120, 220, 220)$;
5 $ellipse(120, 120, 200, 200)$;
//Colher
6 $fill(201, 201, 201)$;
7 $rect(10, 70, 70, 10)$;
8 $rect(60, 60, 20, 30)$;
//Clara do ovo
9 $fill(255)$;
10 $ellipse(120, 70, 100, 50)$;
//Gema do ovo
11 $fill(255, 255, 0)$;
12 $ellipse(120, 70, 30, 30)$;
//Torrada
13 $fill(247, 113, 60)$;
14 $rect(150, 100, 50, 70)$;
15 $rect(160, 110, 30, 50)$;
16$popMatrix()$;

e) Execute uma nova translação com o comando $pushMatrix()$ e $popMatrix()$ deslocando a construção com a função $translate(0, 250)$. Ao invés de digitar os códigos iniciais novamente, selecione, copie e cole.

f) Repita o procedimento anterior usando agora o translate(250, 250) para servir o quarto prato do café da manhã.

Avaliação

Avaliar se todos conseguiram compreender a importância de se utilizar as funções $pushMatrix()$ e $popMatrix()$ para automatizar a mudança de coordenadas dos entes

construídos, através do “Applet” desta atividade. Apesar do árduo trabalho em se digitar o código repetidas vezes até culminar na imagem da Figura 43. Com isso, os alunos constatarão a importância de criar sub-rotinas para evitar a repetição dos códigos enumeras vezes e viabilizar uma construção mais sofisticada, a exemplo das próximas atividades.

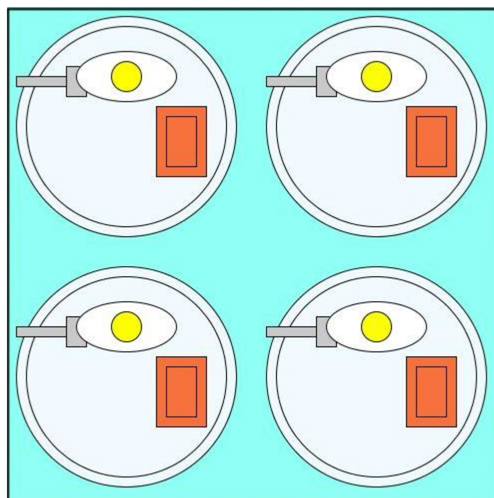


Figura 43: Applet 4: Translação do café da manhã

5.2.3 Macro construção com a translação

Objetivo

Reduzir o número de códigos duplicados no programa para reutilizá-lo sem grandes alterações em outros programas e decompor problemas grandes em pequenas partes para melhorar a interpretação visual do respectivo programa.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Translação

Subsídio Teórico

No contexto da programação, a criação de *sub-rotinas* que são funções, procedimentos ou mesmo subprogramas, consiste em uma porção de código que resolve um problema muito específico, parte de um problema maior para a aplicação maior. Notoriamente, o conceito de função difere da noção de procedimento em algumas linguagens, já que devolve um valor, se bem que esta distinção não é sequer existente nessa linguagem; por exemplo, em *Processing*, a implementação de um procedimento é uma função do tipo *void*.

Dentre as vantagens da utilização de sub-rotinas durante a programação listamos:

- ⋆ A redução de código duplicado num programa;
- ⋆ A possibilidade de reutilizar o mesmo código sem grandes alterações em outros programas;
- ⋆ A decomposição de problemas grandes em pequenas partes;
- ⋆ Melhorar a interpretação visual de um programa;
- ⋆ Esconder ou regular uma parte de um programa, mantendo o restante do código alheio às questões internas resolvidas dentro dessa função.

Dentro dessa ótica, Velho [38, p. 279] pondera:

> A composição de diversos objetos gráficos na formação de modelos mais complexos é de fundamental importância para determinados tipos de aplicação, como por exemplo, a animação de pessoas e animais, e também na área de robótica. Por outro lado, a decomposição de objetos em partes mais simples que se relacionam de forma hierárquica é uma estratégia bastante utilizada na solução de problemas, dentro do espírito de *dividir para conquistar*.

Na concepção de Velho [38], uma hierarquia é um grafo onde os vértices são objetos gráficos. Sendo que um conjunto de pares de objetos constituem as arestas formadoras de uma relação de vínculo hierárquico de dois objetos sobre a mesma aresta. Nesse âmbito, os objetos são classificados de acordo com sua hierarquia em: Objetos compostos quando não existe movimento relativo entre os diversos sub-objetos, ou ainda, os vínculos são rígidos e isso é muito útil para descrever cenários com muitos elementos, como é o caso dessa atividade. E objetos articulados quando os vínculos não são rígidos, ou seja, são objetos constituídos de partes rígidas conectadas por articulações ou juntas que formam um vínculo entre as partes permitindo a movimentação relativa entre elas, como é o caso do corpo humano, dos rôbos, entre outros.

Metodologia

Aula expositiva com realização do Applet; E por ser um Applet bastante iterativo, repleto de implementações para a construção do cenário, foi sugerida a construção de sub-rotinas para atuar de forma lúdica e prática na manipulação e experimentação da linguagem mais complexa.

Material

Papel, lápis ou caneta e computador com o *Processing* instalado.

Procedimento

A construção de um cenário formado por árvores, uma casa, duas pessoas, um lago, uma tartaruga nadando e um céu ensolarado requer um controle aprimorado do uso da linguagem de programação. Em função disso, a computação gráfica desenvolveu uma estratégia para facilitar a confecção de jogos eletrônicos, entre outros. Esse objetivo na programação é alcançado com a criação de funções para controlar cada objeto do cenário e a partir dessa função, preferencialmente com suas coordenadas na origem do sistema, é possível criar vários aplicativos com ou sem animações preservando o objeto.

Com o intuito de construir o cenário mencionado no parágrafo anterior, subdividiremos a atividade em questão em quatro etapas para facilitar a compreensão e domínio dos novos elementos acrescentados na programação.

Parte I

Configuração inicial do cenário, dimensionando a tela e programando o ambiente céu e terra através do comando *void setup* e *void draw*. Esses comandos são muito utilizados nos processos de animação e criação de jogos, por isso, sua importância na execução da atividade para incrementar novas possibilidades de aprendizagem para construções dinâmicas.

a) Abra um novo arquivo e grave-o como: **Applet 5 nome da dupla ou trio**.

b) Configuração da tela com a função *void*.

```
1 void setup(){
2 size(400,400); //Configuração da tela.
3 fill(50,205,50); //Cor do ambiente terra
4 rect(0,250,400,150); //Dimensão da terra.
5 fill(135,206,250); //Cor do ambiente céu
6 rect(0,0,400,250); //Dimensão do céu.
7 }
8 void draw() {
9 }
```

Parte II

Com o mesmo arquivo, vamos incrementar o programa com a construção dos primeiros objetos para o cenário, ou seja, vamos estruturar os objetos árvores e o objeto céu (sol e nuvens), conforme mostra a Figura 44(a). E para finalizar essa parte, vamos programar o ambiente casa particionando seus detalhes (paredes, telhado, janela e porta) em funções distintas para o programa sintetizar como consta na Figura 44(b)

a) Abra uma "Nova Aba" e nomeei o ambiente onde montaremos os objetos de "cenario". Para abrir uma "Nova Aba" clique na seta abaixo da barra de ferramentas e selecione "Nova Aba".

b) Na aba "cenario", construa com a função *void* os objetos mencionados, de acordo com a ordem preestabelecida.

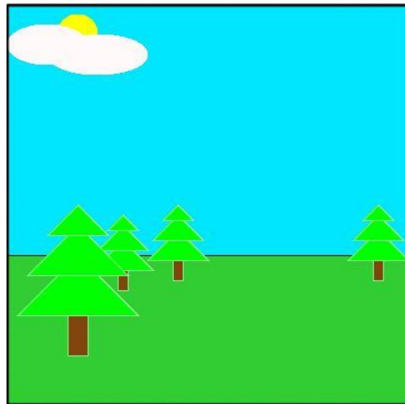
```
1 void arvore1(){//Função árvore 1.
2 fill(0,255,0);//cor verde.
3 stroke(144,238,144);
3 triangle(0,50,-60,110,60,110);//Galhos.
4 triangle(0,20,-50,70,50,70);//Galhos.
5 triangle(0,0,-30,30,30,30);//Galhos.
6 fill(129,69,14);
7 rect(-10,110,20,40);//Tronco.
8 }
9 void arvore2() {//Função árvore 2.
10 fill(0,255,0);//cor verde
11 stroke(144,238,144);
12 triangle(100,25,70,55,130,55);//Galhos.
13 triangle(100,10,75,35,125,35);//Galhos.
14 triangle(100,0,85,15,115,15);//Galhos.
15 fill(129,69,14);
16 rect(95,55,10,20);//Tronco.
17 }
18 void ceu(){ //Função céu(sol e nuvens).
19 fill(255,255,0);//Cor amarela (sol).
20 ellipse(−20,−20,40,40);//sol.
21 fill(255,250,250);//Cor branca (nuvens).
22 ellipse(0,0,100,40); //nuvem.
23 ellipse(−50,−10,80,40);//nuvem.
24 }
```

c) Ao executar o programa não aparecerá os objetos recém criados, para visualiza-los é necessário acrescentar a função arvore1, arvore2 e ceu dentro do comando *void draw* na aba principal. Em outras palavras, abra a aba principal designada de "Applet 5" e no interior do comando *draw* digite os códigos abaixo deslocando os objetos para suas respectivas posições, preservando a ordem dos dados.

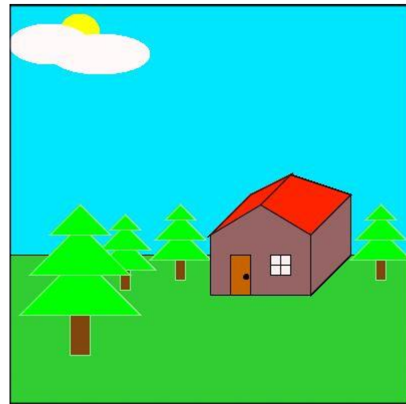
```
1 void draw(){
2 pushMatrix();
3 translate(15,210);
4 arvore2();
5 popMatrix();
6 pushMatrix();
7 translate(70,200);
8 arvore1();
9 arvore2();
10 popMatrix();
11 pushMatrix();
12 translate(270,200);
13 arvore2();
14 popMatrix();
15 pushMatrix();
16 translate(90,50);
17 noStroke();
18 ceu();
19 popMatrix();
20 }
```

Vamos pensar Nessa construção existe alguma relação entre o dimensionamento da árvore 1 para a árvore 2? Se existir, qual a razão entre árvore 1 e a árvore 2?

(*Dica:* Comece analisando as coordenadas dos galhos da árvore 1 para a árvore 2, para depois analisar o tamanho do tronco das árvores, na mesma ordem)



(a) Cenário árvore e céu ensolarado



(b) Cenário com a casa

Figura 44: Construção do cenário casa com árvores e céu ensolarado

Com relação ao questionamento anterior, espera-se que os alunos percebam a relação de proporção existente entre as árvores cujo fator mensurado foi de $\frac{1}{2}$. Nesse sentido, ainda é plausível instigá-los quanto a viabilidade de escrever as coordenadas da árvore 1, menor, da mesma forma que a árvore 2, maior, utilizando para isso uma redução de escala conforme abordaremos no bloco de transformações da secção 5.5.

Após a implementação das árvores, dos elementos sol e nuvem no céu, só faltará programar a construção da casa particionando seus detalhes em funções conforme fora relatado anteriormente. Para isso, voltaremos a aba "cenario" para programar as funções de partição da casa.

d) Na aba "cenario" implementaremos a construção da casa a partir das funções telhado, paredes, porta e janela, conforme a ordem de lançamento dos dados constante na listagem.

```
1 void telhado(){
2 fill(255, 36, 0);
3 quad(-50, 30,-10,-10, 30,-30, 0, 0);
4 quad(0, 0, 30,-30, 90,-10, 50, 30);
5 }
6 void paredes(){
7 //parede da frente
8 fill(150, 100, 100);
9 beginShape();
10 vertex(−50, 30);
11 vertex(−50, 90);
12 vertex(50, 90);
13 vertex(50, 30);
14 vertex(0, 0);
15 endShape(CLOSE);
16//parede lateral
```

```
17 quad(50, 30, 50, 90, 90, 50, 90, −10);
18 }
19 void porta(){
20 fill(200, 100, 0);
21 rect(0, 0, 20, 40);
22 fill(0);
23 ellipse(16, 22, 5, 5);
24 }
25 void janela(){
26 fill(250, 240, 240);
27 rect(0, 0, 10, 10);
28 rect(10, 0, 10, 10);
30 rect(0, 10, 10, 10);
31 rect(10, 10, 10, 10);
32 }
```

e) Para a construção ser visualizada no aplicativo devemos posicionar as funções da partição casa em suas devidas coordenadas dentro do comando *draw* da aba "Applet 5", conforme fizemos no item c da atividade. Entretanto, daremos continuidade a partir de onde finalizamos na programação do ambiente árvores e elementos do céu.

```
void draw(){···
1 //casa
2 pushMatrix();
3 translate(250, 200);
4 stroke(0);
5 telhado();
6 paredes();
7 popMatrix();
8 pushMatrix();
9 translate(220, 250);
10 porta();
11 popMatrix();
12 pushMatrix();
13 translate(260, 250);
14 janela();
15 popMatrix();
}
```

Para essa atividade utilizamos novos códigos, a saber: *quad(···)*, utilizado para construir qualquer quadrilátero dados seus vértices, como fizemos na função telhado, e *beginShape(); vertex(···); ··· ; endShape(CLOSE);* que serve para construir qualquer poligonal, principalmente as que possuem mais de 4 vértices. O que garante o fechamento dessa poligonal é a palavra "CLOSE" encontrada no *endShape*, se remover essa expressão do código que construiu a frente da casa, esta perderá uma linha poligonal que voltará a se fechar somente repetindo o código do primeiro vértice (vertex) antes de encerrar o código *endShape*, ou reescrevendo a expressão "CLOSE" novamente.

Após programar a construção da casa, das árvores e do sol ensolarado, a nossa atividade só precisa agora da imagem de pessoas e um lago com a tartaruga nadando conforme sugerido no início da atividade. Para isso, abordaremos na próxima etapa a construção da figura de dois bonecos com sua vestimenta posicionado atrás da árvore maior e no outro extremo da tela para dar uma noção de profundidade.

Parte III

Nessa atividade construiremos a figura do boneco particionando na mesma lingua-

gem as partes do corpo. No entanto, em animações e construção de jogos costuma-se chamar uma função para cada parte do corpo e assim fazer movimentos independentes de forma mais sofisticada partindo de elementos escritos na origem do sistema. Como nosso boneco ficará estático e já apresentamos muitas funções até o momento, manteremos nossa construção em uma única função com sua vestimenta em outra função.

Dessa forma, vamos iniciar a construção da função boneco e da função roupa seguindo o mesmo princípio das etapas anteriores. Para avaliar o grau de aprendizagem até o momento, omitiremos aonde os códigos devem ser inseridos e como fazer a imagem do boneco aparecer atrás da árvore2, maior, como podemos ver na Figura 45.

a) Construção da função boneco;

```
1 void boneco(){
2 rect(−5, 15, 10, 40); //coluna
3 quad(−5, 55, 0, 55, −10, 85, −15, 85);
//perna direita
4 quad(5, 55, 0, 55, 10, 85, 15, 85);//perna esquerda
5 quad(−5, 20, −5, 25, −20, 45, −25, 45);
//braço direito
6 quad(5, 20, 5, 25, 20, 45, 25, 45); //braço esquerdo
7 //rosto
8 ellipse(0, 0, 35, 35);//cabeça
9 ellipse(0, 0, 3, 8);//nariz
10 fill(135, 206, 250);//cor azul
11 ellipse(−6, −6, 4, 4);//olho direito
12 ellipse(6, −6, 4, 4);//olho esquerdo
13 fill(240, 240, 240);
14 ellipse(0, 7, 20, 5);//boca sorrindo
15 //óculos
16 fill(0, 0, 0, 100);//lente translúcida
17 rect(−10, −10, 8, 8);//direito
18 rect(2, −10, 8, 8);//esquerdo
19 fill(0);//armação preta
20 rect(−2, −7, 4, 2);//central
21 rect(−20, −7, 9, 2);//direita
22 rect(10, −7, 9, 2);//esquerda
23 }
```

b) Construção da função roupa;

```
1 void roupa(){
2 //camisa
3 rect(−5, 20, 10, 40);//coluna
4 quad(−5, 20, −5, 25, −15, 40, −24, 40);
//braço direito
5 quad(5, 20, 5, 25, 15, 40, 24, 40); //braço esquerdo
6 //bermuda
7 quad(−5, 55, 0, 55, −4, 70, −12, 70);
//perna direita
8 quad(5, 55, 0, 55, 4, 70, 12, 70); //perna esquerda
9 }
```

c) Visualização do boneco com sua vestimenta na tela gráfica;

Boneco atrás da árvore

```
1 pushMatrix();
2 translate(40, 245);
3 stroke(0);
4 fill(255, 160, 122);//Salmão claro na pele
5 boneco();
6 popMatrix();
7 pushMatrix();
8 translate(40, 245);
9 stroke(0);
10 fill(100, 50, 255);//azul
11 roupa();
12 popMatrix();
```

Boneco depois do lago

```
1 pushMatrix();
2 translate(350, 300);
3 stroke(0);
4 fill(255, 160, 122);//Salmão claro na pele
5 boneco();
6 popMatrix();
7 pushMatrix();
8 translate(350, 300);
9 stroke(0);
10 fill(100, 50, 255);//azul
11 roupa();
12 popMatrix();
```

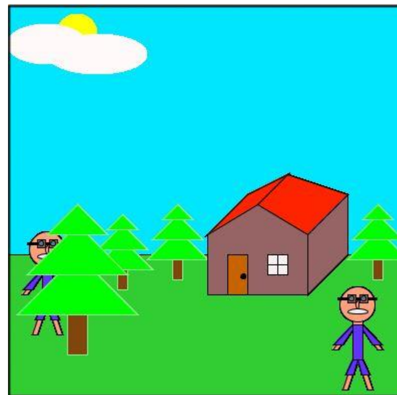


Figura 45: Complementação do cenário com bonecos no ambiente

Com a aprendizagem das atividades anteriores, espera-se que o discente consiga notoriamente descobrir a posição do código do Boneco atrás da árvore sem dificuldades. E essa animação viabiliza ainda alguns questionamentos para os discentes, dentre eles destacamos:

Como é possível visualizar os olhos do boneco mesmo utilizando óculos? Podemos remover parte da armação sem apagar nenhum código? Qual a cor dos olhos do boneco?

Para todos esses questionamentos, os alunos precisam rever a linguagem e idealizar seu objetivo antes de executar algum comando. No caso da visualização dos olhos mesmo utilizando os óculos, a resposta encontra-se na cor da lente, note que o comando *fill* apresenta um quarto valor que determina na escala de $[0, 255]$ o tom de transparência da

cor selecionada, isto é, quanto mais próximo de zero, mais transparente fica essa cor, no sentido contrário, mais escurece a cor preestabelecida.

No segundo questionamento talvez o professor tenha que intervir com algumas dicas para os alunos prestarem atenção nos comentários da própria linguagem, algo do tipo, "na hora que aparece o nome da cor ou nome das partes do boneco alguém visualiza esses nomes na tela ou acontece algum erro durante a execução da linguagem". Em último caso o docente chamará a atenção para as "duas barras" e isso possibilitará a investigação e experimentação por parte dos envolvidos com dificuldade de assimilar algo não usual até então.

Existe ainda a possibilidade de suprimir uma gama de códigos sem utilizar as barras duplas em cada linha a ser omitida, nesse ponto a figura do professor mediador será crucial por apresentar algo novo na linguagem muito utilizado na programação quando não se deseja revelar toda uma construção de imediato ou não se necessita de algum elemento específico sem deletar a informação. Esse recurso é um comentário que pode omitir várias linhas ao mesmo tempo quando colocado os símbolos corretos entre os códigos a serem postergado, assim o docente poderá explicar a diferença entre cada uma das formas de se inserir um comentário como podemos perceber:

```
// Comentário de linha simples.
/* Inicializa o comentário de várias linhas
continua comentário
*/ Finaliza o comentário
```

A partir dessa explanação os alunos poderão responder ao terceiro questionamento com maior presteza otimizando tempo e recursos para alcançar um objetivo, e isso viabiliza seu poder de decisão cotidiana.

Para finalizar essa atividade, construiremos um lago onde uma tartaruga irá nadar constantemente da esquerda para a direita. Por essa a primeira animação dinâmica, deixamos essa etapa para o final por considerar um recurso estimulante para os discentes já pensarem na possibilidade de aperfeiçoarem o conhecimento em programação para a criação de jogos e animações mais sofisticadas.

Parte IV

Para construir um lago será necessário apenas um código da *ellipse*, mas, para construir uma tartaruga necessitaremos de vários comandos da *ellipse* como mostra a sequência de códigos da função tartaruga nadando no lago.

a) Programação do lago e da tartaruga animada;

```
1 void tartaruga(){
2 ellipse(0, 0, 200, 90); //lago
3 fill(183, 115, 51); //cor do casco
4 ellipse(−70 + a, 0, 50, 20); //casco
5 fill(154, 205, 50); //cor do corpo
6 //pernas traseiras
7 ellipse(−87 + a, −13, 10, 10);
8 ellipse(−87 + a, 13, 10, 10);
9 //pernas dianteiras
10 ellipse(−53 + a, 13, 10, 10);
11 ellipse(−53 + a, −13, 10, 10);
12 ellipse(−45 + a, 0, 15, 15);//cabeça
13 //olho
14 fill(255);//parte branca
15 ellipse(−43 + a, −2, 7, 5);
16 fill(0); //parte preta
17 ellipse(−42 + a, −2, 2, 2);
18 }
```

b) Visualização da tartaruga em movimento.

```
1 pushMatrix();
2 translate(200, 350);
3 fill(0, 200, 255);
4 a = (a + 0.5)%139;
5 tartaruga();
6 popMatrix();
```

Quando os alunos digitarem o código para gerarem o movimento da tartaruga em função da variável a, o programa exibirá uma mensagem de erro no seu rodapé com a seguinte expressão em inglês "*The variable 'a' does not exist*[1]", de fato, não definimos para quais valores esse termo a existe. Nesse sentido, a figura do professor volta a ser crucial para apresentar e diferenciar as notações de variável em programação que se restringe a dois tipos nesse trabalho, **int** e **float**.

A variável **int** é utilizada no contexto dos números inteiros, enquanto a variável **float** é empregada no campo dos números reais. Diante desse conhecimento, o docente solicitará que os alunos acrescente o comando *int a = 0* no início da aba "Applet 5" ou fora da função *setup*, antes de iniciar a função *void draw*.

Se os alunos já tiverem escrito o código de visualização da função tartaruga irá aparecer outra mensagem de erro em inglês "*Cannot convert from float to int*[2]". Antes dos discentes alteram conforme o programa sugestiona, questionem aos alunos o porque do programa fazer essa indagação.

Aos atenciosos, talvez este questionamento seja desnecessário. Entretanto, o objetivo das atividades esta além de instrumentalizar-los no uso da linguagem de programação.

[1]A variável 'a' não existe, **tradução nossa**

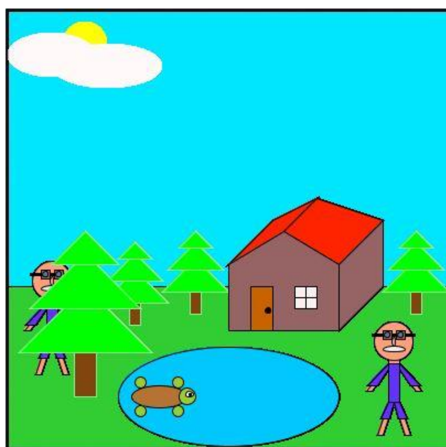
[2]Não pode converter de *float* para *int*, **tradução nossa**

Objetiva-se nos discentes uma atitude investigativa com foco na experimentação e diálogo. Por este motivo, tal questionamento é fundamental para que percebam que apesar do código utilizado refira-se aos números inteiros, na atribuição ao valor de "a" foi estabelecido um incremento não inteiro e por este motivo a própria linguagem nos chama a atenção para substituir a variável lançada por *float*.

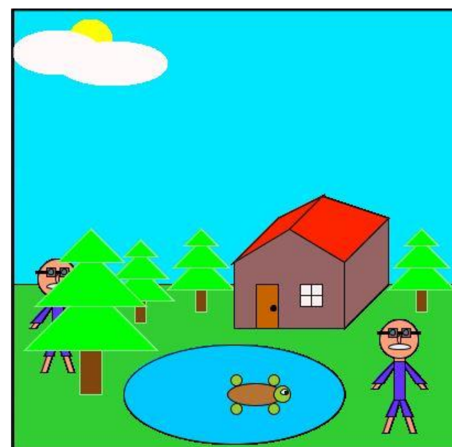
Apesar de chegarem a essa conclusão, antes que os alunos mudem o contexto da variável, de inteiro para real, solicitem aos discentes uma alteração no incremento para um número inteiro, tipo: $a = (a + 1)\%139$, depois continuem a aumentar esse acrescimento. A grande pergunta a ser fazer com esta oscilação no incremento é: Qual o efeito gráfico que o acréscimo inteiro provoca no movimento da tartaruga? A provável resposta será na velocidade do movimento da tartaruga. Diante disso, vem o questionamento quanto ao acréscimo mais ideal para o movimento de uma tartaruga, se na escala dos números inteiros ou dos números reais.

Por este motivo, vamos mudar o contexto da variável "a" para os números reais, isto é, trocaremos o comando *int a = 0* por *float a = 0.* E para o acréscimo podemos solicitar aos alunos que testem valores menores do que 1 e mantenham aquele que julgarem mais fidedigno ao movimento da tartaruga.

Com isso, finalizamos essa atividade com uma animação para o movimento da tartaruga, conforme mostra o jogo de imagens das Figuras 46(a) e 46(b). Salienta-se ainda que tais imagens são apenas representações do deslocamento da tartaruga, uma vez que o recurso impresso limita a visualização de imagens em movimento, por esse motivo, utilizar recursos tecnológicos no processo de ensino aprendizagem viabiliza um conjunto de possibilidades ilimitadas quando comparado ao material didático.



(a) Tartaruga no início do movimento



(b) Tartaruga em movimentação

Figura 46: Applet 5: Cenário com animação da tartaruga no lago

Avaliação

Com essa atividade esperamos que os alunos desenvolvam a habilidade de construir funções para controlar os elementos gráficos e assim manipulem muitas partes do cenário com a menor quantidade de códigos.

5.3 Isometria de rotação

A rotação é uma transformação geométrica que preserva as características de uma isometria de tal forma que o objeto sofre apenas um giro em torno de ponto fixo no plano. Dessa forma, esperamos que os alunos caracterizem os objetos como isométricos segundo um determinado ângulo de rotação em relação ao objeto original e para atingir esse feito, contaremos com o papel mediador do professor durante o processo de programação e exemplificação de construções existentes realizadas utilizando esse princípio geométrico.

Para fundamentar o processo de aprendizagem dessa isometria utilizaremos construções básicas e elaboradas via linguagem de programação, no entanto, utilizaremos material concreto como ponto de partida no intuito de viabilizar a combinação da translação com a rotação no efeitos gráficos que serão produzidos em parceria com os discentes. Tal parceria e ponto de partida é crucial para as atividades elaboradas e o entendimento das propriedades características da rotação.

5.3.1 Conhecendo a rotação

Objetivo

Construir o conceito de rotação de uma figura bidimensional e fazer com que os alunos percebam a rotação da imagem na tela do “Applet” como um giro segundo um ângulo em torno de um ponto fixo. Além de identificar a importância da translação no processo de uma rotação em computação gráfica, principalmente, de sua ordem de execução.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Rotação e Translação; Plano Cartesiano.

Subsídio Teórico

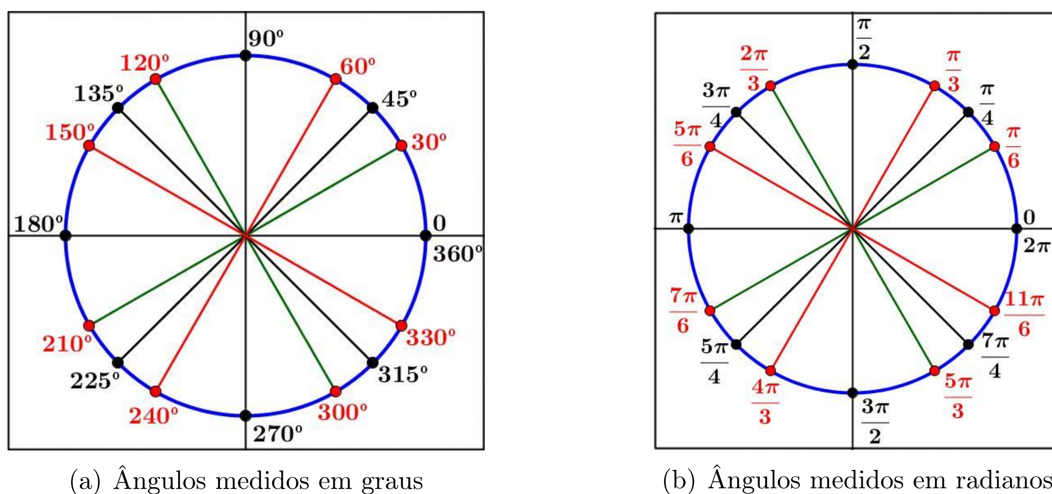
Rotacionar é rodar o objeto no plano, em outras palavras, rotacionar uma figura

consiste em girar o objeto em torno de um ponto fixo segundo uma amplitude medida em radianos ou graus, onde a forma e o tamanho do objeto é preservada.

Na computação gráfica, em especial no *Processing*, a rotação é uma função que faz girar todo o sistema de coordenadas em torno de sua origem quando o comando *rotate(rad)* é acionado na construção. Onde o parâmetro *rad* é o ângulo medido em radianos quando giramos o objeto no sentido horário.

Na Matemática, a unidade padrão para medir ângulos é o radiano. Entretanto, costumamos ensinar trigonometria com os ângulos medidos em graus tanto a nível de Ensino Fundamental como no Ensino Médio. Em função disso, vamos escrever a rotação segundo essa unidade de medida, o grau, acionando o código *rotate (radians(grau))*; Onde a expressão *radians(grau)* será a conversão automática da medida do ângulo em graus para radianos.

Mesmo não sendo necessário o domínio das unidades do ângulo em radiano, é importante que os alunos conheçam pelo menos o sistema de conversão de uma unidade para a outra e para isso podemos partir de uma circunferência de raio r qualquer com centro na origem do plano cartesiano. Como o círculo é dividido em quatro quadrantes nesse plano e uma volta completa mede 360º, então conforme mostra a Figura 47, os ângulos em graus e radianos são denotados por sua correspondência na regra de três simples:



(a) Ângulos medidos em graus (b) Ângulos medidos em radianos

Figura 47: Medidas de ângulos em graus e em radianos

Graus	Radianos
360º	2π
180º	π
α	β

Pela regra de três simples, os alunos podem converter a unidade de grau para radianos ou radianos para graus no momento que julgarem oportuno aplicando a conversão direta como segue:

$\alpha = \frac{180^{o}\beta}{\pi}$ $\beta = \frac{\alpha\pi}{180^{o}}$

Combinação das transformações de rotação e translação

A função de rotação gera um efeito visual interessante. No entanto, utilizando somente esse recurso na programação acometeremos um erro no quesito visual, visto que, a rotação gira todo o sistema de coordenadas em torno da origem, não importando qual o objeto desenhado na tela. Para evitar esse problema utilizaremos a translação para alterar a origem antes de rotacionar a construção e assim, toda e qualquer construção permanecerá visualmente na tela.

Metodologia

Aula expositiva com construção do "Applet" e manipulação do plano cartesiano; Exploração do código de rotação com os estudantes agrupados em duplas ou trio; E compete aos estudantes construir o Applet experimentando os efeitos gráficos em sintonia com a teoria da transformação para enriquecer as possibilidades nas construções de novos aplicativos.

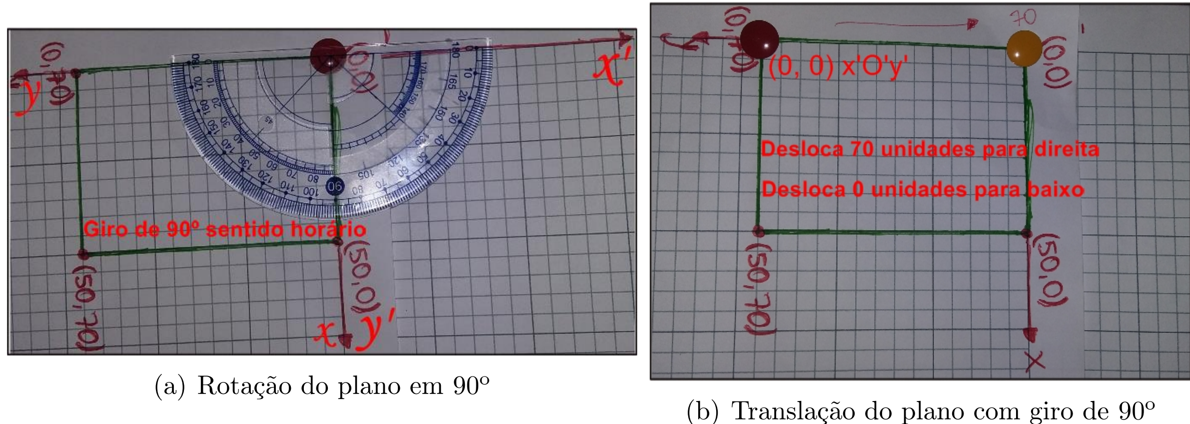
Material

Malha quadriculada, lápis ou caneta e computador com o *Processing* instalado.

Procedimento

Considere um retângulo de dimensões 50 por 70 *pixels* iniciada na coordenada $(0, 0)$. Fixando-se um alfinete na origem do sistema e girando toda a malha quadriculada no sentido horário, com o auxílio do transferidor, faça o retângulo girar 90° em relação as coordenadas iniciais, como mostra a Figura 48(a).

Mantendo-se uma malha quadriculada em branco, fixa abaixo da inicial, os alunos constataram que a rotação do retângulo saiu totalmente do sistema da malha, em computação isso significa que a construção não esta visível na tela. Para evitar esse efeito visual utilizamos uma translação depois da rotação, deslocando o novo retângulo em 70 unidades para a direita e 0 unidades para baixo, em relação ao "novo" sistema, e com isso, a construção volta a aparecer totalmente sob a malha quadriculada fixada ressurgindo na tela como aparece na Figura 48(b).



(a) Rotação do plano em 90º

(b) Translação do plano com giro de 90º

Figura 48: Rotação do retângulo de 50 por 70 pixels na malha quadriculada

Contudo, se executarmos o mesmo deslocamento no *Processing* não obteremos a mesma imagem obtida de forma manual. O motivo encontra-se no fato do novo sistema obtido após o giro de 90º ser equivalente ao eixo $-yOx$, em outras palavras, o novo eixo horizontal é o $-y$ e o eixo vertical é o x. Com isso, na programação devemos lançar a seguinte informação em relação ao sistema de eixos 'original", xOy, deslocamos 0 unidades para a direita e -70 unidades para baixo. Para exemplificar isso, construiremos essa mesma atividade no *Processsing* salientando que essa situação seria totalmente diferente se o ângulo de giro fosse distinto do ângulo reto, ou ainda, para qualquer outro ângulo diferente dos múltiplos de 90º a relação entre os "novos" eixos e os "originais" não seria facilmente obtida e isso dificulta a visibilidade do objeto programado.

Essa atividade mostrará na prática a importância no deslocamento da origem do sistema para um ponto no interior da tela, em alguns casos costumamos utilizar o centro da tela como "nova" origem, e só assim executarmos o giro do ângulo estipulado. O intuito de manter essa ordem tanto em programação como "manualmente" versa sobre a facilidade de realizarmos a rotação sem nos preocuparmos com a coordenada de origem. Isso, é muito empregado no estudo da Cônicas em Geometria Analítica, para ser mais preciso, a ordem de execução é a mesma diferindo apenas no fato de deslocarmos as cônicas com centro fora da origem para um "novo" eixo cuja origem coincida com o centro da cônica para obtermos a rotação da mesma como aparece nos livros de Geometria Analítica de nível superior, a exemplo, o livro de Geometria Analítica da Coleção PROFMAT [40].

Nesse intuito, além da referida construção, montaremos um jogo de imagens do quadrado de lado 50 pixels girando em torno do mesmo ponto fixo, no centro da tela, através da função void para não precisarmos repetir sempre o mesmo código modificando apenas a cor de cada quadrado.

Para alcançar nosso objetivo, vamos lançar os códigos do retângulo de 50 por 70 pixels e do quadrado de 50 pixels de lado em "Applets" distintos e avaliar o efeito da rotação no *Processing* seguindo ordens diversas na execução da translação.

Parte I

Visualização do retângulo de 50 por 70 pixels na origem seguida da rotação de 90º em torno dessa origem, e por fim, um deslocamento de 0 unidades para direita e -70 unidades para baixo.

a) Abra um arquivo no *Processing* e salve como: **Applet 6 nome da dupla ou trio**.

b) Construa o retângulo de 50 por 70 pixels centrado na origem do "Applet" de tamanho 150 por 150 pixels com plano de fundo branco. Não esqueça de definir uma cor (*fill*) para o retângulo;

c) Aplique o comando *rotate(radians(90));* e, em seguida, repita o código do retângulo modificando apenas sua cor. Ao final, execute o programa e observe as imagens obtidas. O novo retângulo com giro de 90º apareceu na tela?

d) Como a rotação não desenha o retângulo dentro da tela, vamos transladar, agora, o eixo em 0 unidades para a direita e -70 unidades para baixo. Para isso, utilize a função *translate* seguida do código do retângulo com uma nova cor. Execute novamente o programa e visualize como o retângulo aparece na tela. O retângulo continua na mesma posição do inicial? O efeito visual proporcionado pode se igualar ao da atividade realizada em malha quadriculada?

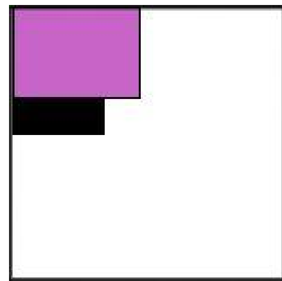
e) Como desafio, coloque toda a construção a partir do comando *rotate(radians(90));* em comentário utilizando o artifício /* $\cdots$ */ e;

f) Refaça a construção iniciando com a mesma translação adaptando suas coordenadas de forma que o retângulo deslocado apareça na tela com uma cor distinta;

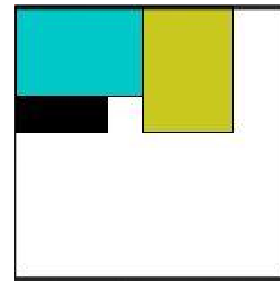
g) Depois de transladar o retângulo, vamos girar o retângulo em 90º da mesma forma que fizemos antes, repetindo para isso as coordenadas do retângulo com uma nova cor logo em seguida ao comando *rotate(radians(90));*

h) Comparando o primeiro resultado com esse, existe alguma diferença visual? E quanto aos códigos, existe alguma distinção? Se houver, qual foi a principal modificação necessária?

Com essa atividade os alunos viabilização a importância de executar uma translação antes de iniciar a própria rotação como meio facilitador, ao ponto de alcançar o mesmo objetivo com um grau menor de dificuldade como consta nas Figuras 49(a) e 49(b). Salienta-se ainda que se a translação fosse para o centro da tela a construção exigiria um grau de dificuldade ainda menor. Dessa forma, o docente pode sugerir que os alunos mudem a rotação para o centro da tela e avaliem o quanto esse recurso torna a atividade versátil.



(a) Rotação seguida da translação do retângulo



(b) Translação seguida da rotação do retângulo

Figura 49: Applet 6: Rotação-Translação e Translação-Rotação no retângulo de 50 por 70 pixels

Parte II

De posse do conhecimento de rotação e a importância em executar uma translação antes de ativar o giro do plano, vamos construir diversos quadrados de 50 pixels de lado com giros constantes de 30º ao redor do mesmo ponto de origem deslocada. Para deixar a construção mais dinâmica, utilizaremos o recurso de mudança de cor constante para dar um efeito visual diversificado em conjunto com a função *void*.

a) Abra um novo arquivo e salve como: **Applet 7 nome da dupla ou trio**.

b) Vamos construir o cenário para a função *void* com aplicação para números inteiros na mudança de cores dos quadrados.

```
1 int a=0; //Variável inteira para as cores.
2 void setup() {
3 size (200, 200);//Tamanho da tela.
4 smooth();//Suaviza o traçado.
5 background(255);//Fundo branco.
6 }
7 void draw() {
8 }
```

c) Construa o quadrado vermelho de 50 pixels de lado com deslocamento de 100 por 100 unidades a partir da origem que servirá de base para nossa construção, como mostra a Figura 50(a). Para alcançar esse objetivo utilize os comandos *rect*$(\cdots)$;

$translate(\cdots);$ e $fill(\cdots);$ em qualquer tom de vermelho na função *void draw ()*. Não esqueça de organizar os dados para manter o quadrado a partir do ponto de coordenadas $(100, 100)$.

d) Abra uma aba e nomeia de "quadrado". Nessa aba vamos construir duas funções quadrados e cores programadas com um giro de 30º, onde a função cor dependerá do parâmetro "a". Siga os comandos para construir esses quadrados e cores animadas.

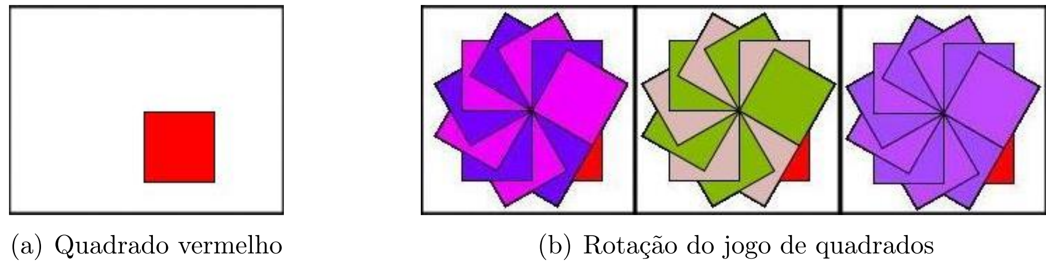
```
1 void quadrado1() {
2 rotate(radians(30)); //Giro de 30º.
3 cor(1); //Cor animada.
4 rect(0, 0, 50, 50); //Quadrado 1.
5 }
6 void quadrado2() {
7 rotate(radians(30)); //Giro de 30º.
8 cor2 (); //Cor animada.
9 rect(0, 0, 50, 50); //Quadrado 2.
10 }
11 void cor(1) {
12 fill(255−a, 255−2 ⋆ a, 5 ⋆ a+1);
13 }
14 void cor(2) {
15 fill(100+a, 255−2 ⋆ a, 5 ⋆ a−200);
16 }
```

Observe que a função *void* quando utilizado para controlar a cor de preenchimento cria a possibilidade de manuseia-la em outras funções. Diante disso, qual o benefício prático viabilizado por esse procedimento? Ao executar uma separação do objeto de seu preenchimento podemos manipular as cores de forma independente e assim diferenciar durante a programação ao mesmo tempo que reaproveitar em objetos diversos.

e) Para reproduzir a Figura 50(b), volte a aba "applet 7" e digite o nome da função quadrado1(); e quadrado2(); alternadamente depois do comando do quadrado vermelho na função *void draw* de modo que a construção complete uma volta encaixando o primeiro quadrado girado com o último.

f) Note que a construção apresenta uma cor constante e nosso objetivo é deixa-la oscilante a partir do parâmetro "a". Para manter esse intuito, acrescente o comando $a = (a+1)\%width;$ ao final do jogo de dados das funções quadrados antes de encerrar a função *void draw*.

Com isso, nossa atividade se encerrar. No entanto, é possível incrementar a atividade adicionando um círculo e dois triângulos simétricos no centro da construção de forma que o triângulo gire 360º, continuamente. Para isso, é só construir os elementos necessários na aba "quadrado" e depois ampliar os códigos da aba "applet 7" com os respectivos dados na ordem.



(a) Quadrado vermelho (b) Rotação do jogo de quadrados

Figura 50: Applet 7: Giro de quadrados com cores oscilantes

Na aba "quadrado"

```
1 void centro() {
2 fill(255,200,0); //Cor amarela
3 ellipse(0, 0, 30, 30); //Círculo no centro
4 strokeWeight(2);
5 rotate(b); //Para rotação de 360º
```

Na aba "quadrado"

```
6 cor1();
7 triangle(0, 0, 15, 0, 0, 15);
8 rotate(radians(180));
9 cor2();
10 triangle(0, 0, 15, 0, 0, 15); }
```

Na aba "applet 7"

```
1 int a = 0;
2 float b = 0; //Para rotação dos triângulos
3 void setup () {
4 ··· }
```

Na aba "applet 7"

```
5 void draw() {
6 ···
7 centro ( );
8 b += 0.01; }
```

Aos alunos que complementarem a atividade é viável utilizar uma malha quadriculada para construir o círculo e explorar a possibilidade de construir dois triângulos com um dos pontos na borda do círculo e os demais no interior do mesmo de forma a se assemelhar com um losango. Com isso, os alunos explorarão a linguagem em conjunto com uma estratégia para solucionar um problema eminente ao nível de seu conhecimento. O resultado do incremento da atividade e essa última variação desafiadora produz como resultado as Figuras 51(a) e 51(b), sendo que a última é considerada um possível resultado tendo em vista a liberdade de escolha das coordenadas dos triângulos.

5.3.2 Aprimorando a rotação

Objetivo

Aprimorar a função de rotação através do condicionador *if-else* para teste de valor lógico, verdade e falsidade no processo de construção de "mandalas". E Automatizar as

(a) Triângulo em giro (b) Losângulo em giro

Figura 51: Applet 7.1: Catavento quadrangular em oscilação de cores

aplicações com elementos repetidos em coordenadas distintas para melhorar a linguagem durante a programação.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Rotação; Noções de lógica.

Subsídio Teórico

O desenvolvimento dessa atividade está atrelada a uma noção rápida de lógica, principalmente, no que discerne a operação de verdade e falsidade. Na programação existe alguns comandos utilizados para analisar os operadores lógicos através dos conectivos "e", "ou" e "negação". No entanto, tal análise não faz parte do enfoque dessa atividade, mas na próxima atividade abordaremos com mais detalhes, no momento foquemos nossa atenção ao fato da informação verdadeira executar uma operação e o contrário, a informação falsa executará outra operação.

Para esse tipo de análise vamos utilizar o recurso da função *if-else*. Essa estrutura atua em conjunto com o comando *for* que executa uma repetição no comando até que uma condição seja satisfeita, isto é, o comando *for* é uma estrutura de repetição empregado quando se conhece o número de vezes que o bloco da construção é executado. Sua estrutura fundamenta-se é um valor inteiro ou real associado a uma condição com incremento na função e, a parti daí, iniciamos os comandos para o argumento verdadeiro seguida da informação falsa, como segue no esquema abaixo:

for(valor inicial; condição; incremento na função){
if (afirmação) {
comando se for verdadeiro;
} *else*{
comando se for falso;
}

A esse tipo de argumentação é possível inserir outras condições caso a primeira informação não seja verdadeira, com isso é possível criar ramificações no comando *if-else* até atingir todos os objetivos. No entendimento de Ben Fry e Casey Reas [27, p. 53], the conditionals allow a program to behave differently depending on the values of their variables[3]. Para cada afirmação deve haver uma expressão que resolve para verdade ou falso e quando a expressão é verdadeira o código entre as chaves é executado, do contrário passa para a próxima afirmação. Os autores supracitados [27, p. 66] enumeram ainda como o comando *for* é executado a cada nova interação como segue na sequência com tradução nossa:

1. A declaração do valor inicial é executado;
2. A afirmação é avaliada como verdade (true) ou falso (false);
3. Se a afirmação for verdadeira, vá para a etapa 4. Se a afirmação for falsa, pule para a etapa 6;
4. Execute as instruções dentro do comando se verdadeiro;
5. Execute a instrução de incremento da função e passe para a etapa 2;
6. Saia da estrutura e continue a execução do programa.

Durante a aplicação será possível testar as informações anteriores e seus efeitos visuais na construção de mandalas.

Metodologia

Aula expositiva com construção do "Applet"; Exploração do código de repetição *for* com teste lógico de verdade ou falso com os estudantes agrupados em duplas ou trio; E explorar os efeitos visuais na retirada do preenchimento dos objetos para construir mandalas.

Material

Computador com o *Processing* instalado.

Procedimento

Para construir imagens no estilo de mandalas facilmente vamos utilizar o recurso de repetição de comandos seguindo uma lógica para verdade e outra para falsidade. Como

[3]as estruturas condicionais permitem que um programa se comporte de forma diferente, dependendo dos valores de suas variáveis (**Tradução nossa**).

essa é nossa primeira atividade no seguimento, iremos mostrar como utilizar esse recurso com detalhes para futuras construções ou remodelação das já realizadas com o intuito de enxugar o código ao máximo e otimizar a construção.

Vamos começar preparando o cenário para a construção das mandalas e depois lançar os dados dos objetos para gerar o visual gráfico necessário do nosso objetivo.

a) Abra um arquivo e salve como: **Applet 8 nome da dupla ou trio**.

b) Configuração inicial do cenário da mandala.

```
1 size(300, 300);
2 background(255); //Fundo branco.
3 translate(150, 150); //Nova origem da construção.
```

c) Estrutura *for* com comando *rotate*.

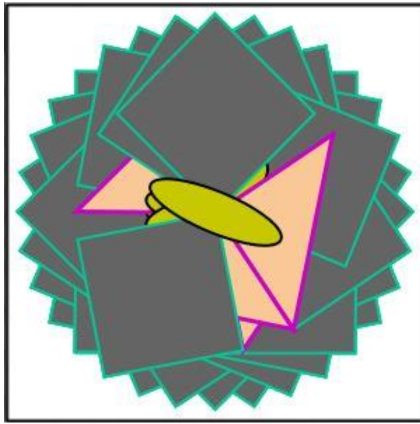
```
1 for (int i= 1; i < width; i ++) {
2 rotate(radians(11.25)⋆i);
3 //Construção inicia para i=1 e aumenta uma unidade a cada novo giro até atingir o valor da largura (width) da tela.
4 ···}
```

d) Incrementando o teste lógico no comando *for* para avaliar o efeito visual.

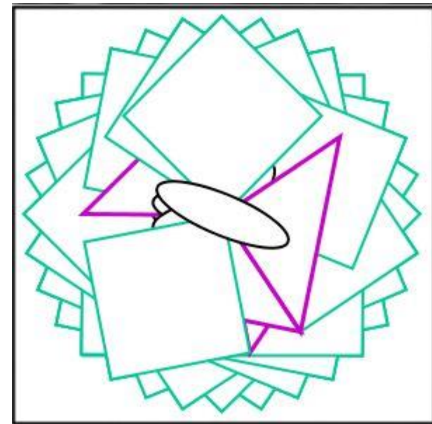
```
4 if (i%3==0) {
5 fill(250, 200, 150);
6 stroke(200, 0, 200);
7 strokeWeight(3);
8 triangle(0, 0, 100, 0, 0, 100);
9 } else if (i%2==0) {
10 fill(100, 100, 100);
11 stroke(0, 200, 150);
12 strokeWeight(2);
13 rect(0, 0, 100, 100);
14 } else {
15 fill(200, 200, 0);
16 stroke(0, 0, 0);
17 strokeWeight(2);
18 ellipse(0, 0, 100, 30);
19 }
```

e) Execute a construção e observe como fica o empilhamento dos objetos conforme mostra a Figura 52(a). Se omitirmos o preenchimento dos objetos colocando uma barra dupla nos comandos *fill* obteremos a Figura 52(b).

A função *if* desenha o triângulo se i dividir por 3 sem resto. Se não dividir executa os códigos depois de *else*, para desenhar o quadrado de 100 pixels de lado, o valor de i dividirá exatamente por 2 e, se houver resto ainda desenha uma elipse como comando final.



(a) Sobreposição de objetos coloridos

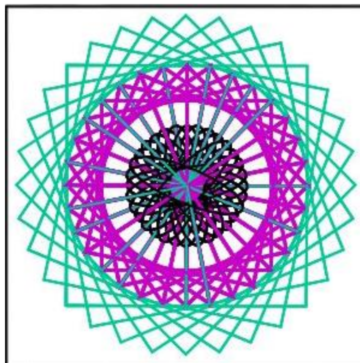


(b) Objetos sobrepostos

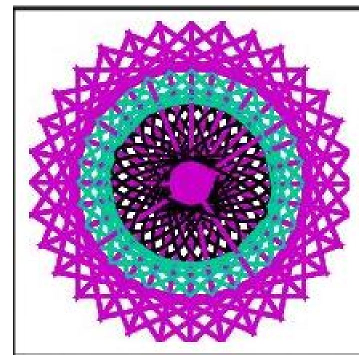
Figura 52: Applet 8: Construção dos elementos da mandala

f) No entanto, nossa construção ainda não apresenta uma característica de uma mandala, para isso acrescente na configuração inicial antes da função *translate* o comando *noFill();*. Esse comando remove o preenchimento das construções permitindo sobrepor as linhas visualmente como podemos ver através da Figura 53(a).

g) O que acontece com a mandala se acrescentarmos logo após a cor do plano de função o comando *rectMode(CENTER);*. Teste esse comando e observe seu efeito.



(a) Mandala expandida



(b) Mandala compactada

Figura 53: Applet 8.1: Mandala em duas perspectivas centrais

O comando *rectMode* combinado com o modo CENTER afeta como os retângulos são desenhados conforme mostra a Figura 53(b), esse modo faz com que o primeiro e o segundo parâmetro do retângulo seja o centro da construção para uma largura e altura estabelecida, para os alunos perceberem a diferença solicitem que ocultem o comando *noFill* e execute o programa. Depois oculte o comando *rectMode(CENTER)* e questionem qual a diferença visual da construção. Assim poderá explicar como esse comando funciona na integra.

Avaliação

Avaliar se todos conseguiram construir suas mandalas nos dois formatos solicitados. Com essa atividade o mediador poderá solicitar uma nova construção mudando a posição dos objetos ou inserindo novos objetos geométricos. Com isso, vai ser possível reavaliar o domínio da nova ferramenta e sua potencialidade de ensino preparando-os para aplicações mais complexas.

5.3.3 Animação com efeito de rotação

Objetivo

Ampliar o domínio do recurso da função de rotação com a construção de uma animação dinâmica e interativa nas suas cores.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Rotação.

Subsídio Teórico

Para a construção dessa animação os alunos utilizaram todos os conhecimentos já apresentados nas aplicações anteriores com o acréscimo de uma nova função chamada *keyPressed()*, onde cada vez que uma tecla é pressionada o código dentro da função *KeyPressed* é executado uma vez. Essa função é associada a variável *boolean* que possui apenas dois valores lógicos, verdade ou falso, além disso, existe a possibilidade de combinar com a condicional *if* ativando seus códigos somente quando pressionar uma tecla ou mesmo parando seus códigos ao pressionar alguma tecla a depender da atribuição na variável *boolean*.

O roteiro da atividade e execução da animação visa uma manipulação prática de um sistema de rotação a partir do esquema na malha quadriculada. O sucesso do processo depende da estruturação do fluxograma antes da programação para viabilizar uma otimização na linguagem utilizada.

Além dessa nova variável o aluno precisará da noção de lógica com relação ao operador lógico “ou”, entretanto, abordaremos os outros valores lógicos com o intuito de diferenciar-los primordialmente no contexto da programação. Nesse intuito, o professor mediador poderá utilizar atividades textuais utilizando esses conectivos para exemplificar a veracidade da informação. Como o nosso público se resume aos alunos do nono ano do Ensino Fundamental, vamos elucidar essas operações lógicas explanando quando a relação

entre duas sentenças assume valor lógico verdade e do contrário assume valor lógico falso.

Como as operações lógicas são visualizadas geralmente no primeiro ano do Ensino Médio, não vamos entrar em detalhes com a tabela verdade. Em suma, os discentes necessitam apenas reconhecer quando duas sentenças assumem valor lógico verdade ou falso a depender do conectivo lógico empregado. Por esse motivo, a presença do professor mediador em conjunto com um banco de afirmações textuais e matemáticas podem ajudar os alunos analisarem quando uma informação é verídica ou não.

De acordo com o emprego da lógica em consonância com a linguagem de programação podemos resumir cada operador lógico e seu emprego computacional como segue:

1. Operador lógico "**e**" só assume o valor lógico verdade quando todas as informações são verdadeiras. Em programação, esse operador lógico é representado pelo símbolo "&&".

Exemplo 5.1. *Dado o retângulo nas coordenadas* $rect(x+10, x+20, 60, 40)$; *Construa todos os retângulos para* $x > 20$ *e* $x < 50$, *onde* x *é um número inteiro não negativo.*

2. Já o operador lógico "**ou**" só assume valor lógico verdade quando pelo menos uma das afirmações é verdadeira. Na programação, o símbolo representativo é o "||".

Exemplo 5.2. *Dada a elipse nas coordenadas* $ellipse(a+50, 30+a, a+30, a+30)$; *Construa todos as elipses quando* $a < 40$ *ou* $a > 60$ *para a pertencente ao conjunto dos números inteiros não negativos.*

3. O operador lógico da "*negação*", nega todas as afirmações anteriores. Em programação, essa negação é ativada pelo símbolo "!".

Exemplo 5.3. *Para todo c não maior que* 50, *construa os triângulos nas coordenadas* $triangle(c+20, 50, c+60, 10, 150, 200)$ *Onde c pertence ao conjunto dos números inteiros não negativos.*

Todos esses exemplos podem ser explorados em sala de aula através de uma malha quadriculada para visualizar se os alunos compreendem o significado prático de cada conectivo no contexto da matemática com ênfase para a construção de objetos geométricos. Uma vez compreendido no papel o processo de sua construção passa-se a planejar como inserir essa construção através de uma linguagem de programação de forma a se obter o mesmo efeito visual. Nesse ponto, sugerimos a aplicação dos exemplos na programação através dos códigos que sugestionaremos a seguir.

1. Configure a tela no tamanho necessário para a construção com um plano de fundo branco, sem preenchimento (*noFill*) e uma cor para a linha do contorno (*stroke*) com a espessura de 1.5 (*strokeWeight*);

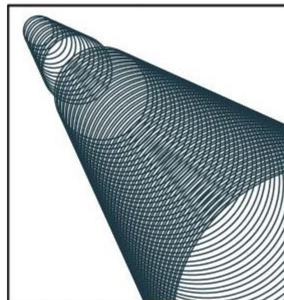
2. Utilize o comando *for* para inserir o tipo de variável, a condição pode ser o valor da tela e seu incremento unitário;

3. Para cada construção lógica utilize no código *if* a restrição para enfim escrever as funções solicitadas.

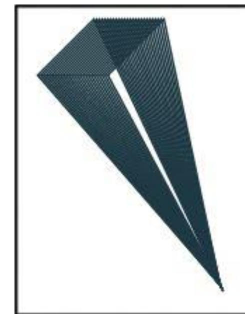
Para demonstrar como proceder nessa exemplificação com os alunos via linguagem de programação, vamos lançar os códigos referente aos exemplos 5.1, 5.2 e 5.3 com as respectivas Figuras 54(a), 54(b) e 54(c), nessa ordem. Salienta-se que a visualização isolada de cada uma delas se dar perante a inclusão de parte do código como comentário.

```
1 //Configuração básica do programa
2 setup(300, 300);
3 background(255);
4 noFill(); //sem preenchimento.
5 stroke(30, 57, 67);
6 strokeWeight(1.5);
7 //Teste do valor lógico “e”.
8 for (int x = 0; x < 300; x = x + 2){
9 if (x > 20 && x < 50){
10 rect(x + 10, x + 20, 60, 40);
11 }
12 }
13 //Teste do valor lógico “ou”.
14 for (int a = 0; a < 300; a = a + 3){
15 if(a < 40 || a > 60){
16 ellipse(a + 50, 30 + a, a + 30, a + 30);
17 }
18 }
19 //Teste do valor lógico “negação”.
20 for(int c = 0; c < 300; c = c + 2){
21 boolean d = c > 50; //d = verdade
22 if(!d){ //!d = falso
23 triangle(c + 20, 50, c + 60, 10, 150, 200);
24 }
25 }
```

(a) Exemplo 1



(b) Exemplo 2



(c) Exemplo 3

Figura 54: Operadores lógicos “e”, “ou” e “negação”

Note no código das aplicações lógicas que o incremento amplia a construção em duas ou três unidades e a esse efeito dar-se o espaçamento entre as linhas das figuras, no caso do incremento unitário não haveria espaçamento suficiente para visualizar a interrupção na construção do exemplo 2, e ao construir a negação foi necessário construir uma variável booleana com a declaração indesejada para só assim negar essa nova variável através da lógica computacional, isto é, utilizando o símbolo "!".

Através de todas essas ferramentas podemos construir nosso "Applet 9" com a funcionalidade em futuras programações elaboradas utilizando o mesmo princípio básico.

Metodologia

Aula expositiva com construção do "Applet" e manipulação do plano cartesiano; Exploração do código dos operadores lógicos em conjunto com os estudantes agrupados em duplas ou trio; E compete aos estudantes construir o esquema estrutural na malha quadriculada para converter em linguagem de programação e assim, aprimorar o applet através da rotação deslocada para o centro da tela.

Material

Malha quadriculada, lápis ou caneta, compasso e computador com o *Processing* instalado.

Procedimento

Vamos construir duas barras verticais com uma bola em cada uma, se movendo, e com essas barras apoiadas a partir do seu centro na terceira barra faz todo o conjunto girar completamente até uma tecla ser acionada para ativar e desativar o movimento. Para elucidar esse objetivo é necessário intercalar a programação com o planejamento na malha quadriculada de forma a garantir o sucesso do "Applet". A sensação de movimento será proporcionado pelas variáveis reais controlando os ângulos dos giros e uma variável inteira para gerenciar o movimento da bola, sendo que essa atividade é uma adaptação do vídeo tutorial do Professor Doutor Michael Kipp[4] que objetivava ensinar o gerenciamento das funções *pushMatrix* e *popMatrix*.

Com isso, vamos lançar mão dos dados da construção no *Processing* alternando com a simulação na malha quadriculada a medida que julgarmos necessário.

a) Abra um arquivo e salve como: **"Applet 9 nome da dupla ou trio**.

[4]Para maiores informações sobre o curso do *Processing* desse professor, alemão, da Universidade de Augsburg acesse o site http://processing.michaelkipp.de/.

b) Configure a função *void setup* da seguinte forma;

```
1 void setup() {
2 size(300, 300);
3 noStroke(); //Sem serrilhamento no traço
4 fill(0);
5 rectMode(CENTER); //Para centralizar
6 }
```

c) Os primeiros itens da função *void draw* será configurar o plano de fundo e construir o retângulo que servirá de suporte para os verticais.

Para isso, use uma malha quadriculada e projete o retângulo com as dimensões de 200 por 15 pixels na origem. Como nossa construção terá o efeito de giro, então desloque o retângulo para o centro da malha quadriculada. Diante disso, podemos incrementar os códigos da etapa no programa e visualizar o primeiro giro de 360º.

```
1 void draw() {
2 background(200); //Fundo cinza.
3 translate(150, 150); //Centro do Applet.
4 rotate(a); //Para o giro de 360º.
5 rect(0, 0, 200, 15);
6 }
```

Note que o programa informa que a variável não existe, solicite que os alunos criem essa variável no campo dos números reais (*float*) antes da função *setup* e execute o programa. Nessa hora a barra não faz nenhum movimento por não especificar como será o incremento dessa variável a a cada execução do código, daí, adicione o comando $a+=0.01$ antes de fechar a chave da função *draw* e volte a executar o primeiro giro.

d) Na malha quadriculada desenhe um novo retângulo na origem com dimensões 10 por 80 pixels, observe que essa será nossas barras verticais e devemos posicionar-las nas extremidades na barra de suporte.

Para isso, solicitem aos discentes que determinem inicialmente a posição da barra à direita (barra 1), ou seja, qual o deslocamento necessário dessa vez para alcançar o objetivo, deixe os alunos testarem seus dados no programa dentro da função *draw* adicionando os códigos entre *pushMatrix* e *popMatrix* para visualizar se estão corretos.

Provavelmente, essa barra sairá deslocada da barra suporte em função do primeiro deslocamento já ter mudado a origem em 150 pixels. Nesse ponto, a interferência do professor viabilizará essa observação para que os mesmos corrijam os dados sendo necessário, a depender do nível de cada aluno, apenas chamar a atenção para a primeira construção. Uma vez corrigido os dados lançados se equipararam aos seguintes implementado na função *draw*.

```
void draw() {
...
translate(100, 0); //Centro barra1 (250, 150).
rotate(b); //Novo giro de 360º.
rect(0, 0, 10, 80);
... }
```

Solicite que repita o procedimento da variável a para criar essa nova variável b com o mesmo aumento no incremento, sendo este último anotado após o incremento a.

e) Descubra qual o deslocamento para a barra vertical à esquerda (barra 2) a partir da conclusão da barra anterior e adapte os códigos anteriores para essa nova barra que ficará com os seguintes códigos adicionado logo após a última barra.

```
void draw() {
...
translate(−100,0); //Centro barra2 (50,150)
rotate(c); //Novo giro de 360º.
rect(0, 0, 10, 80);
... }
```

f) Para finalizar a atividade falta colocar uma bola em movimento em cada barra vertical e só conquistaremos essa etapa depois de planejar na malha quadriculada as coordenadas da referida bola.

Solicitem aos alunos para construir um círculo, nossa bola, de raio 10 pixels apoiado na barra 1 com seu centro sobre o segmento que parte do centro dessa barra paralelo ao lado menor. Ao deslocar a barra 1, deslocamos a origem do sistema mantendo seu centro na origem, em função disso, considere o centro da barra 1 como o centro do sistema para determinar as coordenadas da bola. A provável coordenada encontrada se seguida a orientação terá a seguinte característica na programação $ellipse(15, 0, 20, 20);$.

Para movimentar a bola na vertical, vamos trocar a coordenada do eixo y por uma variável "*bola*" e lançar esse código juntamente com a criação dessa variável inteira no programa. Como essa bola refere-se ao movimento da barra vertical podemos lançar seu código tanto na barra 1 como na barra 2. Assim, o código do programa ficará com esse acréscimo logo após cada barra, mas só terá movimentação se incluirmos o incremento $bola += 1;$.

```
void draw() {
...
translate(100, 0); //Centro barra 1 (250, 150).
rotate(b); //Novo giro de 360º.
rect(0, 0, 10, 80);
ellipse(15, bola, 20, 20);//bola 1
translate(−100, 0); //Centro barra 2 (50,150).
rotate(c); //Novo giro de 360º.
rect(0, 0, 10, 80);
ellipse(15, bola, 20, 20);//bola 2
```

$\cdots$

$a+=0.01;$

$b+=0.01;$

$c+=0.01;$

$bola+=1;$

$\}$

Note que a bola não executa o seu movimento somente sobre as barras e para corrigir isso precisamos de uma nova variável para controlar o movimento da bola condicionando-a a mover-se somente sobre as barras. Nesse quesito recorremos aos operadores lógicos, especificamente, ao operador lógico "ou" para restringir o movimento além da barra.

Daí, incluiremos os seguintes códigos no seu lugar específico.

$float\, a = 0;$

$\cdots$

$int\, bola = 0;$

$int\, moverbola = 1;$

$\cdots$

$c+=0.01;$

$bola+=moverbola;$

$if(bola > 40\,||\,bola < -40)\{$

$moverbola = -moverbola;$

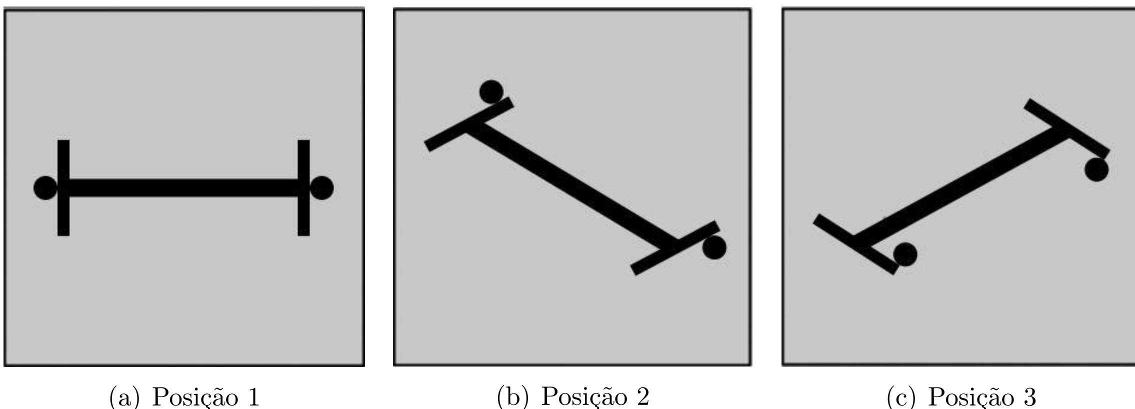
$\}$

$\}$

Com isso, as bolas se moverão ininterruptamente dentro do intervalo da altura do retângulo, ou seja, quando a bola atingir os limites do retângulo o código $moverbola = -moverbola$; fará suas coordenadas retrocederem o movimento da bola. Mas, a bola 2 encontra-se na região interna em relação a barra suporte, para mudar isso, basta mudar a coordenada do eixo x de 15 para -15 resultando na Figura 55. Para instalar um controle que pare o movimento ao pressionar qualquer tecla é só acrescentar uma nova função chamada *void keyPressed* ao final da linguagem com sua variável de controle.

$void\, KeyPressed()\{$

$jogar =!jogar;\}$



(a) Posição 1 (b) Posição 2 (c) Posição 3

Figura 55: Applet 9: Controle de movimento com rotação

No entanto, é preciso atribuir um valor lógico na variável booleana, isto é, acrescentar no início da programação o código $boolean\, jogar = false;$. Mesmo assim, não é possível parar o movimento com nenhuma tecla sem antes configurar os valores da função $jogar$, para isso, precisamos criar uma condição, if, controladora de todos os incrementos. Daí, basta incluir os incrementos no comando if como segue:

```
if(jogar){
a+ = 0.01;
b+ = 0.01;
c+ = 0.01;
bola+ = moverbola;
if(bola > 40 || bola < -40){
moverbola = -moverbola;
}
}
}
```

Não podemos esquecer de fechar todas as chaves, do contrário o programa irá acusar um erro na linguagem e inviabilizará sua aplicação.

Avaliação

Com essa atividade espera-se que os alunos desenvolvam uma interação com os operadores lógicos juntamente com a função de controle de movimento além de ampliar seu domínio no planejamento dos fluxogramas importante para o desenvolvimento de aplicativos básicos e avançados.

5.4 Isometria de reflexão ou simetria

A reflexão ou simetria é uma transformação geométrica que preserva as características de uma isometria de tal forma que o objeto sofre uma "rotação em relação ao eixo espelhado" como se fosse retirado da folha de papel e transportado para o outro semiplano da folha como aponta o pensamento de Souza [36]. Dessa forma, esperamos que os alunos caracterizem os objetos como isométricos segundo uma reflexão em relação ao espelhamento dos eixos coordenados. Nesse sentido, a figura do professor mediador durante a etapa de execução é de fundamental importância, ainda mais sabendo que o mesmo recurso computacional é utilizado para ampliar ou reduzir os objetos possibilitando ainda a dilatação ou compressão de apenas um dos eixos.

Para fundamentar o processo de aprendizagem dessa isometria vamos combinar com outras transformações em construções básicas e avançadas via programação diversificando com material concreto para elaborar os projetos na companhia dos discentes.

5.4.1 Conhecendo a simetria

Objetivo

Construir o conceito de simetria ou reflexão de uma figura bidimensional e fazer com que os alunos notem a simetria como um "espelhamento" do objeto em outro semiplano da tela, isto é, em torno dos eixos coordenados deslocados e da nova origem.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Simetria e Translação.

Subsídio Teórico

A simetria é uma transformação que necessariamente precisa estar combinada com a translação dos eixos coordenados pelo mesmo motivo que demonstramos na rotação. Sendo assim, todas simetrias implementadas na atividade encontra-se com seus eixos deslocados para o centro da tela, como referencial visual.

Para essa atividade vamos utilizar os elementos de outras atividades implementando o recurso de inserir texto na tela para diferenciar a figura original das refletidas em relação aos eixos e a própria origem deslocada (centro da tela).

Uma fonte deve ser convertida inicialmente para o formato VLW antes do *Processing* exibir o texto no *display*. Para converter uma fonte, selecione a opção "Criar Fonte" no menu Ferramentas e na janela que se abrir com os nomes das fontes instaladas no computador plausível de conversão, selecione uma dessas fontes e clique em "OK". A fonte é gerada e copiada para a pasta de dados do Sketch atual. Para confirmar se a fonte foi instalada, clique no menu Sketch e selecione "Ver Pasta de Sketch".

Na caixa de diálogo "Criar Fonte" é possível alterar o tamanho da fonte e selecionar se será *smooth* (suave), além da possibilidade de exportar "All Characters" (todos os caracteres) da fonte. Outra vantagem desse mecanismo está no fato da possibilidade de alterar o nome da fonte antes de criar a mesma.

Uma vez que a fonte já exista, a exibição de letras ou textos na tela dependerá de algumas etapas. Antes dessa fonte ser usada, precisamos carregar-la no programa e definir com nossa fonte atual. Para isso, utilizamos um tipo de dados único do *Processing* chamado "PFont" para armazenar seus dados. Daí, para construir uma mensagem textual lançamos os seguintes códigos:

$$PFont\, fonte; \; //\text{Declaração da variável.}$$

$$fonte = loadFont(\text{“}nome\,da\,fonte - tamanho.vlw\text{”}); \text{ //Carrega a fonte.}$$
$$textFont(fonte); \text{ Define a fonte do texto atual.}$$
$$text(\text{“}mensagem\text{”}, x, y); \text{ //Escreva a mensagem na coordenada (x,y).}$$

É possível ainda alterar o tamanho da fonte sem criar uma nova fonte incluindo o tamanho desejado dentro do código $textFont(fonte, novotamanho)$.

Metodologia

Aula expositiva com construção do “Applet” a partir da montagem da estrutura na malha quadriculada para montagem do fluxograma; Exploração do código de simetria com os estudantes agrupados em duplas ou trio; E compete aos estudantes construir o Applet experimentando os efeitos gráficos em sintonia com a teoria da transformação para enriquecer as possibilidades nas construções de novos aplicativos.

Material

Malha quadriculada, lápis ou caneta, compasso e computador com o *Processing* instalado.

Procedimento

Nessa atividade vamos construir um escudo composto de vários círculos, quadrados e um triângulo conforme mostra a Figura 56 e projeta-los em semiplanos distintos tanto em relação aos eixos coordenados como em relação a origem do deslocamento. Para facilitar a visualização vamos construir usando linhas dois sistemas de eixos passando pelo centro da tela para ser nossos eixos coordenados com centro em $(150, 150)$.

Figura 56: Escudo para montagem

Dessa forma, o primeiro passo da atividade consiste basicamente em programar a tela e construir o escudo como aparece na Figura 56. Diante disso, lançamos os seguintes códigos no programa:

a) Configuração do Applet;

```
1 void setup( ) {
2 size(300, 300);
3 background(255);
4 for(int i = 0; i <= 300; i = i + 150){
5 strokeWeight(2);
6 line(0, 150, i + 150, 150);
7 line(150, 0, 150, i + 150);
8 }
9 PFont fonte;
10 fonte = loadFont("Benicio − 40.vlw");
11 textFont(fonte, 38);
12 }
13 void draw(){
14 translate(150, 150);
15 strokeWeight(1);
16 ···}
```

b) Para construir o escudo, abra uma nova "Aba" e nomeia de escudo. Nessa aba configure o escudo particionando-a em subfunções;

```
1 void brasao( ){
2 translate(50, 50);
//Desloca da origem 50 por 50 pixels
3 fill(255, 0, 0);//cor vermelha
4 ellipse(0, 0, 60, 60);
5 translate(−30, −30);
//Volta 30 pixels para a origem
6 fill(237, 145, 33);//Cor salmão
7 quad(30, 0, 60, 30, 30, 60, 0, 30);
8 translate(30, 30);
//Desloca 30 pixels da origem
9 fill(0, 0, 255);//Cor azul
10 ellipse(0, 0, 40, 40);
11 }
12 void simbolo(){
13 translate(30, 30);
//Desloca 30 pixels da origem
14 fill(255, 215, 0);//Cor amarela
15 triangle(20, 0, 40, 30, 0, 30);
16 translate(20, 20);
//Desloca mais 20 pixels
17 fill(252, 15, 192);//Cor rosa choque
18 ellipse(0, 0, 20, 20);
19 translate(−10, −10);
//Volta 10 pixels
20 fill(124, 252, 0);//Cor verde lima
21 quad(10, 0, 20, 10, 10, 20, 0, 10);
22 }23 void escudo(){24 pushMatrix();
25 brasao();
26 popMatrix();
27 simbolo();
28 }
```

c) Uma vez programado a função escudo podemos executar toda a simetria em relação a cada um dos eixos cartesianos e a origem do plano transladado, em 150 pixels, na aba "Applet";

```
13 void draw(){
15 ···
16 pushMatrix();
17 escudo();//Escudo original.
18 fill(0);
19 text("Escudo", −30, 65);
20 text("Original", −30, 90);
21 popMatrix();
```

22 $pushMatrix();$ 23 $scale(-1,1);$

//Simetria em relação ao eixo x deslocado.

24 $escudo();$

25 $popMatrix();$

26 $pushMatrix();$

27 $scale(1,-1);$

//Simetria em relação ao eixo y deslocado.

28 $escudo();$

29 $popMatrix();$

30 $pushMatrix();$

31 $scale(-1,-1);$

//Simetria em relação a origem deslocada.

32 $escudo();$

33 $popMatrix();$

34 }

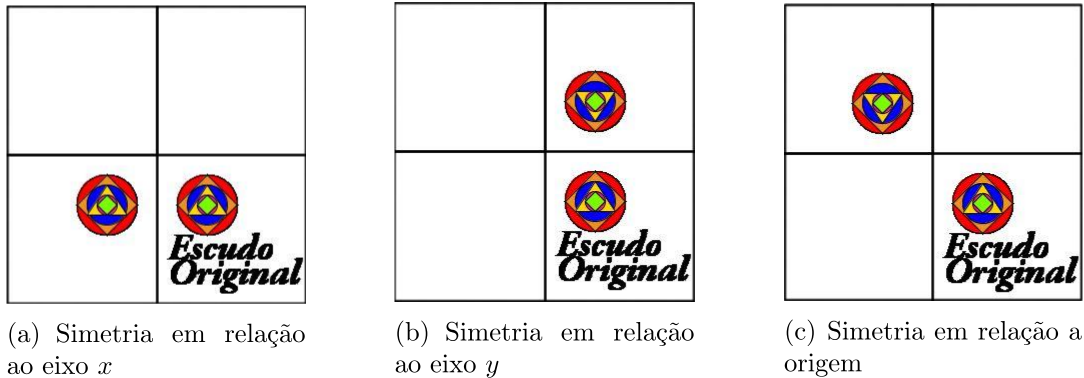
O professor mediador pode intercalar essa construção com o uso da malha quadriculada solicitando para os alunos construírem cada um dos elementos do escudo isoladamente na malha quadriculada e depois montar cada uma das funções, brasao e simbolo, com três objetos apenas, encaixando o menor sobre o maior, sendo um quadrado entre dois círculos para o brasao, e um círculo entre um triângulo e um quadrado para a outra função. Nesse ponto, os alunos devem determinar quanto cada um deslocará para a primeira construção, o círculo maior, aparecer centrada na coordenada $(50, 50)$. E por fim, como montar a função escudo combinando estas duas funções.

Quando o escudo já estiver montado o mediador já pode sugestionar a reflexão em relação aos eixos coordenados e a origem, de forma intercalada para não sobrecarregar os discentes ao mesmo tempo. Nessa fase, é possível ensinar o "espelhamento" utilizando o artifício da dobradura, isto é, dobrando a malha em relação a um dos eixos de forma a sobrepor a construção no outro semiplano da malha, como se houvesse uma "rotação" do objeto em 180º com relação ao eixo deslocado. Após essa sobreposição e marcação da nova posição de cada objeto refletido seria fácil determinar o tipo de comando para a simetria observando qual dos eixos foi refletido, isso é, teve sua coordenada invertida.

De posse dessa informação visual é possível planejar como as coordenadas 1 e -1 será empregada na função $scale(x, y)$ de acordo com a simetria, ou valor invertido do eixo como podemos perceber em cada reflexão da figura 57.

Avaliação

Com essa atividade espera-se que os discentes tenham se familiarizados com a função *scale* para obter simetrias e na composição das equipes tenham compreendido qual o efeito visual de uma reflexão muito encontrado em faixas de ambulância em cima do capo do carro, entre outros meios que exploram esse recurso visual.



(a) Simetria em relação ao eixo x

(b) Simetria em relação ao eixo y

(c) Simetria em relação a origem

Figura 57: Applet 10: Simetria em relação ao eixos e a origem

5.4.2 Aprimorando a simetria

Objetivo

Aprimorar o conceito de simetria na construção de um caleidoscópio a partir de uma peça de dimensão 150 por 150 pixels construído a partir dos códigos fornecidos pelo docente com utilização das TGP.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Simetria , Translação e Rotação.

Subsídio Teórico

A simetria é uma transformação facilmente encontrada na natureza, por exemplo, nos flocos de neve, nas estrelas do mar, nos ouriços, nas borboletas, em alguns pássaros, entre tantos outros meios. Sua ideia remete-se ao equilíbrio proporcional, ao padrão com regularidade, a harmonia com beleza e a uma ordem na sua perfeição.

Dessa forma, como definir a simetria no contexto das formas geométricas planas e nas formas da natureza? Sabe-se que uma figura geométrica plana é dita simétrica se for possível dividi-la por uma reta, de forma que as duas partes obtidas possam se sobrepor por dobragem. Além disso, seu conceito está associado às operações de reflexão, reflexão deslizante, rotação e translação.

Em nossa atividade anterior basicamente utilizamos o conceito de simetria de reflexão ou axial, obtido quando o objeto é refletido em relação a um eixo como se fosse um espelho, e simetria de rotação, quando o objeto gira em torno de um ponto. No entanto, a simetria não se resume apenas no espelhamento ou na rotação, a exemplo da simetria de translação que é dada em função do deslizamento do objeto sobre uma reta mantendo

o mesmo inalterado com apenas dois elementos caracterizadores, são eles: o comprimento da translação ou período e a repetição da forma. Salienta-se ainda que essa propriedade é necessária para compor mosaicos, faixas de ornamentos e estão muito presentes nas obras de Escher.

Outra forma de simetria é a reflexão deslizante que consiste numa operação combinada da reflexão com a translação paralela ao plano de reflexão, ou seja, o objeto é refletido no outro semiplano seguido de um deslocamento paralelo ao eixo de simetria. Com isso, podemos potencializar o ensino dessa transformação geométrica construindo diversas aplicações explorando cada um dessas formas de simetria. Contudo, nosso objetivo é construir um caleidoscópio a partir do fragmento de uma figura fornecida nos códigos divulgados pelo docente que será completada com as transformações de simetria e, em seguida, girada continuamente de acordo com um ângulo a de crescimento constante.

Para a construção dessa atividade precisamos definir o caleidoscópio com o intuito de compreender as passagens em cada trecho da atividade objetivando o efeito visual que tal aparelho proporciona. A palavra caleidoscópio origina de três palavras gregas: *kalos* = belo/beleza + *eidos* = figura/forma + *escópios* = vejo, e segundo a revista Univerciência[5], o caleidoscópio "é um instrumento ótico com pequenos espelhos na extremidade que refletem, múltipla e simetricamente, pequenos objetos, como vidros, formando imagens artísticas muito belas e divertidas" e ao movimentar o tubo, os pequenos vidros se movem modificando a imagem dinamicamente mantendo a simetria regida pelo posicionamento dos espelhos e das leis da reflexão na física. Sabe-se ainda que existe diversas versões de caleidoscópio que preserva o princípio básico regido pela simetria de reflexão que faz parte do intuito dessa atividade.

Nesse tipo de atividade é possível produzir sua própria imagem ou utilizar imagens obtidas na *Internet* para mostrar a importância da simetria de reflexão, a exemplo da borboleta que pode ser obtida a partir da metade de seu corpo, entre outras formas. No entanto, para inserir uma figura no *Processing* é necessário importar-las para o "*Sketch*" antes de iniciar a própria aplicação com a imagem. Esse tipo de atividade será explorado nas transformações de homotetia com viabilidade para ser explorado nas transformações simétricas.

Na construção do caleidoscópio vamos inserir um comando para deixar o preenchimento das formas oscilante para dar a impressão do olho piscando entre outras formas construídas. O comando $random(\text{“}valor'')$ sorteia um número no intervalo do "valor" de-

[5]Revista Univerciência da UFSCAR com publicação em jan/abr de 2002 disponível para consulta no endereço http://www.ufscar.br/ univerci/n_1_a1/assunto_editorial.pdf

terminado de forma aleatória para deixar a animação com efeito de movimento de forma simplificada.

Metodologia

Aula expositiva com construção do "Applet" a partir de uma imagem na malha quadriculada fornecida pelo docente para construir as peças e o sistema de cores para confecção de um mosaico através das simetrias de reflexão e reflexão deslizante para montar um caleidoscópio com a rotação do mosaico. Compete aos estudantes traduzir as imagens em códigos para executá-los no "Applet" observando seus dados para posteriormente construírem seu próprio mosaico.

Material

Malha quadriculada, lápis ou caneta e computador com o *Processing* instalado.

Procedimento

Nessa atividade vamos fornecer um pequeno mosaico, Figura 58, na malha quadriculada para servir de base para os alunos buscarem os códigos de cada figura plana considerando sempre um ponto como origem para visualizar o domínio dos alunos em transladar cada origem para formar o mosaico padrão além de utilizar a simetria de alguns elementos que se repetem em pontos distintos da malha.

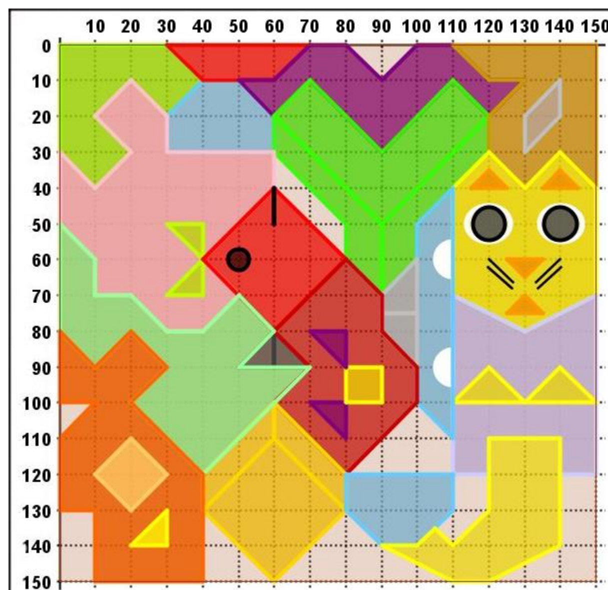


Figura 58: Estrutura para o mosaico

Em função de alguns elementos apresentarem coordenadas com um grau de complexibilidade maior, em especial as formas circulares, vamos determinar suas coordenadas locais para o aluno especificar apenas suas coordenadas globais na construção. Além

disso, todos os pontos estão distantes na proporção de 10 pixels de um para o outro, os que não constam nessa distancia encontram-se a 5 pixels para facilitar suas localizações locais para formar a coordenada global.

Para orientar o docente vamos apresentar a construção do mosaico inteiro com os códigos seguindo uma orientação que pode coincidir com a dos discentes. Para facilitar a escolha das cores, vamos divulgar uma listagem de códigos para os alunos preencherem as figuras planas constante na malha quadriculada, na forma de função.

Diante disso, vamos iniciar a aplicação configurando a tela principal e uma aba para as funções cores disponíveis.

a) Configuração do Applet;

```
void setup(){
size(700, 700);
strokeWeight (1.5);
}
```

```
void draw(){
background(255);
translate(width/2, height/2);
}
```

b) Construção das funções cores na aba "cor":

```
void prata() {
fill(192, 192, 192); }
void azceu() {
fill(135, 206, 250); }
void rosabril() {
fill(255, 0, 127); }
void salmao() {
fill(255, 160, 122); }
void ouro() {
fill(255, 215, 0); }
void jambo() {
fill(255, 69, 0); }
```

```
void castanho() {
fill(165, 42, 42); }
void carmesim() {
fill(220, 20, 60); }
void azvio() {
fill(138, 43, 226); }
void turquesa() {
fill(64, 224, 208); }
void verde() {
fill(0, 255, 127); }
void lima() {
fill(191, 255, 0); }
```

```
void peru() {
fill(205, 133, 63); }
void amarelo() {
fill(255, 255, 0); }
void vermelho() {
fill(255, 0, 0); }
void verclaro() {
fill(135, 206, 250); }
void esmeralda() {
fill(80, 200, 120); }
void cinzafosco() {
fill(105, 105, 105, 200); }
```

Coordenadas dos elementos considerados complexos para se obter de forma individual analisando somente a malha quadriculada.

c) Códigos das formas circulares e das linhas do bigode do gato, com suas cores e linhas animadas com a função *random*(256) que atribui valores aleatórios do intervalo de

[0, 255] para pigmentar a construção com cores oscilando de entre os tons de preto a branco:

```
//Olho do peixe
fill(random(256));
scale(0,0,5,5);
//Olho do gato
ellipse(0,0,10,5);
fill(random(256));
ellipse(0,0,5,5);
//Bigode direito
stroke(random(256));
line(0,0,8,8);
line(0,5,8,13);
noStroke();
```

Nessa atividade vamos favorecer a criatividade dos discentes na construção dos códigos das peças desse mosaico, mantendo as cores que desejarem de acordo com a tabela de cores que disponibilizaremos para a implementação. No entanto, o mediador precisará de um possível resultado para consulta no caso de dúvidas que encontra-se no Apêndice C.

Como nosso objetivo final é construir o mosaico da Figura 59 para criarmos a ideia de movimento com a função de rotação e assim dar mobilidade a nosso caleidoscópio, os discentes vão precisar construir uma função que gerencie todas as peças de forma única para facilitar no "espelhamento". Nesse ponto, os alunos vão precisar combinar as translações de forma a encaixar as peças como consta no modelo da malha quadriculada (Figura 58), isto é, em uma nova aba chamada "mosaico" vamos construir a função *void mosaico*() para recriar a imagem da malha quadriculada no *Processing*.

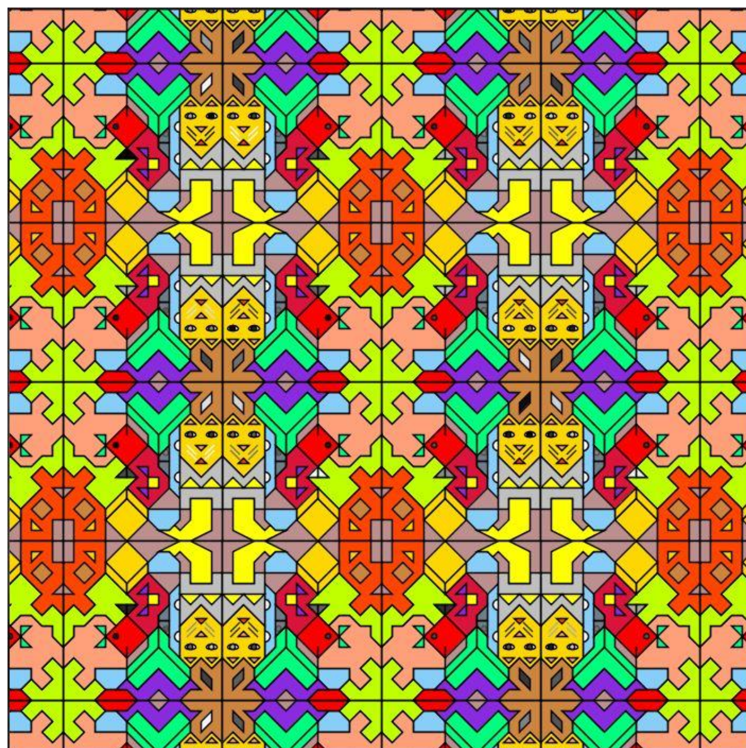


Figura 59: Mosaico completo

A partir da própria malha quadriculada é possível determinar qual a translação necessária para deslocar cada figura com o intuito de encaixar as peças e por esse motivo vamos deixar os códigos no Apêndice C juntamente com os códigos das peças, sugerimos que o docente intermediei o início dessa função sugestionando a chamada de algumas funções da aba "peças" como segue abaixo, não esqueça de mencionar que as funções $pushMatrix()$ e $popMatrix()$ auxiliam toda essa construção.

d) Princípio da construção da função "mosaico":

```
void mosaico(){
fill(189,143,143);
rect(0,0,150,150);
trevo();
pushMatrix();
translate(30,20);
pent();
translate(0,−20);
pm();
translate(10,10);
y();
translate(−30,30);
forma2();
popMatrix();
···}
```

Note que no início da construção surgiu um retângulo com as dimensões da malha quadriculada cuja função é preencher os espaços ausentes de figura plana com a mesma cor para reduzir um pouco o número de códigos utilizados no processo. Com a estrutura do mosaico pronta é só ativar a função $mosaico()$; na função $void\, draw()$ para visualizar se todas as figuras planas encontram-se nas suas respectivas posições e se será necessário algum ajuste no código antes de avançarmos no nosso objetivo.

De posse desses ajustes vamos passar para o "espelhamento" do mosaico nos eixos e na origem para construir nossa peça principal. Para isso, vamos construir uma última aba, "tela", para controlar a reprodução dos mosaicos simétricos fundamentais para nossa aplicação final. Em função da atividade anterior, esperamos que os alunos consigam de forma independente programar essas simetrias combinando as translações com as respectivas simetrias em relação aos eixos. Para orientar-los disponibilizaremos os códigos dessa simetria juntamente com os códigos anteriores no Apêndice C, e assim, o docente poderá auxiliar em qualquer dúvida pertinente as passagem com dificuldade.

Nesta aba "tela" construa a função $void\, telamosaico()$ para implementar uma construção de simetria que funcionará como peça central do nosso mosaico 59. Uma vez com a função construída é possível visualiza-la substituindo a função $mosaico()$ pela função $telamosaico$ e verificar se completou um quadrado com quatro figuras simétricas entre si.

Para os alunos vislumbrarem nosso produto final que é um caleidoscópio em movimento solicite aos mesmos para adicionar o comando $rotate(a)$; no início da função $void\, telamosaico$ e no final da função o incremento $a = a + 0.001$; Salienta-se que o

tipo de variável tem que ser $float$ para números reais e programado antes da função $void\,setup()$ com a inscrição $float\,a = 0$.

Finalmente chegamos na última parte da programação e antes de iniciar-la solicite aos discentes para ocultarem o comando $rotate(a)$ para deixar a construção estática e assim analisar os efeitos dos passos seguintes.

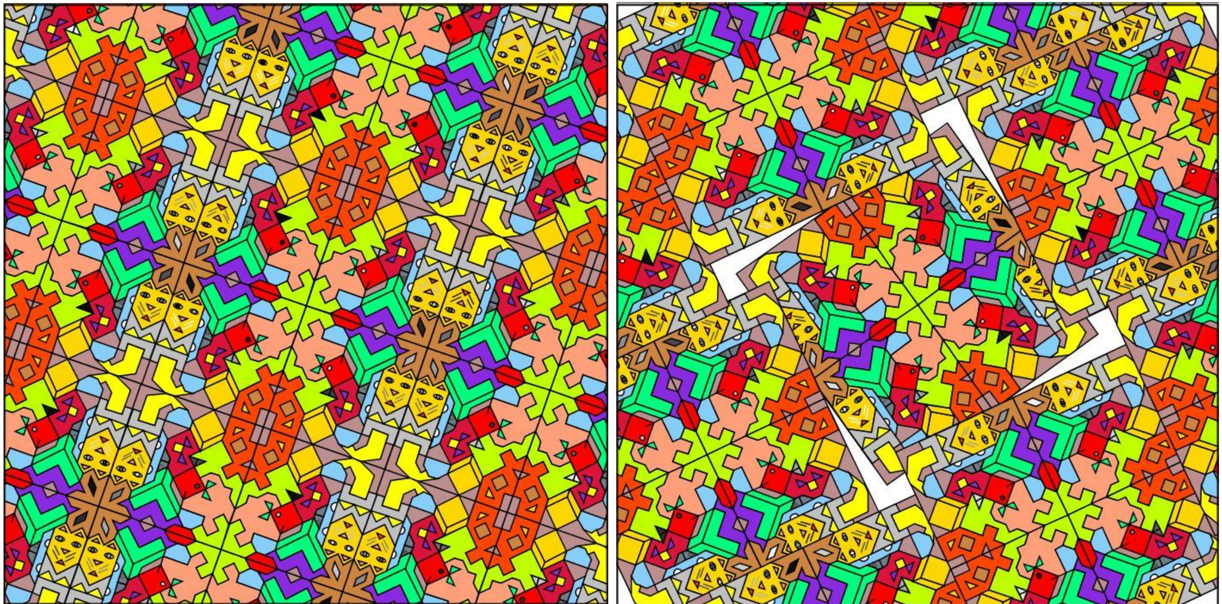
Nessa parte, vamos refletir a função $telamosaico()$ no entorno da parte central. Salienta-se que poderíamos obter o mesmo efeito refletindo a função $mosaico()$, no entanto, será necessário muito mais códigos para alcançar o mesmo objetivo e utilizar a função *scale* para refletir a função *telamosaico* em combinação com a translação não causará nenhum efeito visual diferente daquele em que a função *scale* não for embutido. O motivo para tal ausência de alteração de imagem encontra-se no fato dessa *telamosaico* funcionar como o efeito visual das asas das borboletas, isto é, essa função foi obtida de uma reflexão e a reflexão da reflexão é a própria figura deslocada para uma nova posição.

Por este motivo para economizar no número de códigos na construção do nosso mosaico, como aparece na Figura 59, vamos implementar a função $void\,draw$ com a função *telamosaico* deslocando sua origem para as origens externas da parte central considerando sempre um distanciamento de 300 pixels entre cada nova origem em função da *telamosaico* possuir o dobro do tamanho da Figura 58. Apesar do número de deslocamentos para completar o mosaico seja relativamente pequeno (8 translações) vamos abordar seus códigos no Apêndice C por apresentar em média 45 linhas de comandos. Entretanto, o docente poderá intermediar o início da construção estimulando a descoberta de cada nova origem para os deslocamentos considerando a origem inicial no centro da tela.

Para proporcionar o movimento do nosso caleidoscópio vamos implementar com a função $rotate(b)$ logo após a função *translate* dentro da aplicação *draw* e no final da mesma seu incremento $b = b + 0.004$. Não esqueçam de criar essa variável no campo dos números reais. Para ilustrar o resultado final de nosso caleidoscópio observe o jogo de imagens da Figura 60.

Avaliação

A produção de um caleidoscópio proporciona um jogo de imagens em movimento com variações que permitem uma exploração visual diferente a cada novo jogo de movimento ou visualizar regularidades quando a imagem iniciar um movimento de repetição na sequência de imagens. Além disso, esperamos ainda que os alunos consigam manipular os dados para obter o produto final e assim fazer suas conjecturas quanto ao nível de aprendizagem da linguagem até o presente momento.



(a) Rotação horária da tela (b) Rotação horária da tela e do mosaico

Figura 60: Applet 11: Caleidoscópio do mosaico interativo

A exploração da linguagem de forma quase independente por parte dos alunos viabiliza um desenvolvimento cognitivo favorável para o desenvolvimento de suas próprias aplicações caracterizando a figura do professor como um mediador para impulsionar nos entraves e formalizar suas próprias conquistas.

5.5 Transformação de homotetia

A homotetia é uma transformação geométrica que preserva a forma e as amplitudes dos segmentos, ou seja, é uma transformação que modifica apenas de forma proporcional as medidas de seus segmentos. Em função disso, podemos dizer que a homotetia é uma transformação de ampliação ou redução da imagem segundo uma constante $k \in \mathbb{R}$, tal que $|k| > 1$ ou $0 < |k| < 1$. Com isso, esperamos que os alunos caracterizem os objetos segundo uma relação de semelhança por uma homotetia preestabelecida no princípio da construção.

Para fundamentar o processo de aprendizagem da homotetia, vamos combiná-la com outras transformações em construções básicas e avançadas via programação diversificando com material concreto para elaborar os projetos na companhia dos discentes.

5.5.1 Conhecendo a homotetia

Objetivo

Construir o conceito de homotetia, ou seja, o conceito de ampliação e redução de figuras bidimensionais, e fazer com que os alunos notem a homotetia como uma relação de semelhança entre as figuras planas e suas potencialidades na programação.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Homotetia e Translação.

Subsídio Teórico

A homotetia é uma transformação que preserva as características principais, como a forma e os ângulos, alterando apenas o tamanho da figura e consequentemente sua área de forma proporcional. Essa relação é facilmente explicada através da derivação grega da palavra homotetia, onde *homós* = igual e *thetós* = colocado, isto é, as figuras homotéticas são colocadas a uma distância igual a "algo", geralmente um ponto. Umas das máquinas que integram esse princípio em seu funcionamento são as copiadoras que fazem ampliações ou reduções.

Diferentemente das isometrias, as homotetias positivas não necessitam de uma translação da origem para serem executadas. No entanto, no caso das homotetias inversas, negativas, a translação é imprescindível para visualizarmos na tela da aplicação.

Para implementar a primeira atividade de homotetia, vamos importar uma figura extraída da internet e construir diversas imagens homotéticas a essa. A incorporação da imagem é feita através das seguintes linhas de comando.

$$PImage\ t;\ //\text{Variável para chamada da imagem.}$$
$$t = loadImage(\text{“}nomedaimagem.extenso\text{”});\ //\text{Arquivo com a imagem em } .jpg \text{ e } .png.$$
$$image(t, x, y);\ //\text{Chamada da imagem na coordenada } (x, y).$$

No entanto, é necessário adicionar esta imagem no ficheiro do "*sketch*", mais precisamente numa pasta chamada "data" onde consta o arquivo salvo. Para fazer isso, inicialmente com um arquivo já salvo é só abrir o menu "*sketch*" e selecionar a opção "adicionar ficheiro". Quando uma nova janela se abrir é só procurar a pasta onde deixou a imagem a ser incorporada no aplicativo e selecionar para adiciona-la e assim ser reconhecido pelo programa ao executar as linhas de comando anteriores.

Com as ferramentas necessárias é só iniciar a implementação e produzir uma tela

rica de imagens em diversas proporções para produzir obras de artes visuais utilizando somente as transformações de homotetia e translação.

Metodologia

Aula expositiva com construção do "Applet" a partir da imagem fornecida em arquivo pelo docente mediador para montagem da tela com figuras homotéticas; Exploração do código de homotetia com os estudantes agrupados em duplas ou trio para culminar com a construção do Applet e seus efeitos gráficos em concordância com a teoria da transformação para ampliar as possibilidades das construções e dos aplicativos.

Material

Lápis ou caneta e computador com o *Processing* instalado.

Procedimento

Nessa atividade vamos construir um jogo de imagens dos Minions, mais especificamente, dos Minios Dave e Stuart conforme mostra a Figura 61 em diversas escalas de proporcionalidade. Para facilitar o preenchimento da tela de 850 por 650 pixels vamos utilizar uma dilatação de 50% no Minios Dave seguida de reduções de 50% ou mais para dar a impressão de encolhimento. Ao mesmo tempo que incorporamos o Minios Stuart na aplicação utilizando uma homotetia de redução invertida em diversas posições conforme mostra a Figura 62.

(a) Minion Dave

(b) Minion Stuart

Figura 61: Minions do Filme Meu Malvado Favorito

Entretanto, para alcançar o objetivo dessa atividade inicialmente devemos carregar as configurações básicas do programa para importar as imagens dos Minions supracitados diretamente para o applet, conforme consta nos comandos:

a) Configuração do Applet para carregamento de imagens;

```
//Variável das imagens
PImagek, t;
size(850, 650);
//Imagem do Minion Dave
k = loadImage(”MinionDave.jpg”);
//Imagem do Minion Stuart
t = loadImage(”MinionStuart.jpg”);
background(255);
```

Para o applet reconhecer as figuras dos Minions devemos importar estas imagens para o seu “*sketch*”, mas, antes devemos salvar a aplicação com o nome: **Applet 12 nome da dupla ou trio**. Com o arquivo salvo é só localizar a aba “Adicionar Ficheiro” localizada na barra de ferramentas *sketch* e na nova janela é só buscar onde o professor mediador salvou o arquivo dos Minions para mandar abrir, importar, diretamente no nosso applet.

Com o arquivo anexado em nossa aplicação é só lançar os códigos dispondo cada um dos personagens na tela segundo uma escala de medida determinada no próprio código.

b) Posicionamento do Minion Dave com ampliação de 50% e redução de mesma proporção, além de outras reduções desse personagem;

```
pushMatrix();
translate(400, 0);//Posição (400, 0)
scale(1.5); //Dave ampliado 50%
image(k, 0, 0);
popMatrix();
pushMatrix();
translate(105, 0); //Posição (105, 0)
image(k, 0, 0); //Dave tamanho normal
popMatrix();
pushMatrix(); //Posição (0, 0)
scale(0.5); //Dave reduzido 50%
image(k, 0, 0);
popMatrix();
pushMatrix();
translate(385, 0); //Posição (385, 0)
pushMatrix();
scale(0.35); //Dave em redução 65%
image(k, 0, 0);
popMatrix();
pushMatrix();
translate(10, 200); //Posição (395, 200)
scale(0.27); //Dave em redução 75%
image(k, 0, 0);
popMatrix();
popMatrix();
```

c) Disposição do Minion Stuart em redução invertida a partir de 60% e do Minion Dave em redução invertida de 75%;

```
pushMatrix();
translate(115, 315); //Posição (115, 315)
scale(−0.3); //Stuart em redução 70%
image(t, 0, 0);
```

$popMatrix();$

$pushMatrix();$

$translate(195, 525);$ //Posição (195, 525)

$scale(-0.5);$ //Stuart em redução 50%

$image(t, 0, 0);$

$popMatrix();$

$pushMatrix();$

$translate(415, 635);$ //Posição (415, 635)

$scale(-0.6);$ //Stuart em redução 40%

$image(t, 0, 0);$

$popMatrix();$

$translate(150, 650);$ //Posição (150, 650)

$pushMatrix();$

$scale(-0.33);$ //Stuart em redução 67%

$image(t, 0, 0); popMatrix();$

$pushMatrix();$

$translate(70, -5);$ //Posição (220, 645)

$scale(-0.2);$ //Stuart em redução 80%

$image(t, 0, 0);$

$popMatrix();$

$pushMatrix();$

$translate(330, 0);$ //Posição (480, 650)

$scale(-0.3);$ //Stuart em redução 70%

$image(t, 0, 0);$

$popMatrix();$

$translate(345, -120);$ //Posição (495, 530)

$scale(-0.25);$ //Dave em redução 75%

$image(k, 0, 0);$

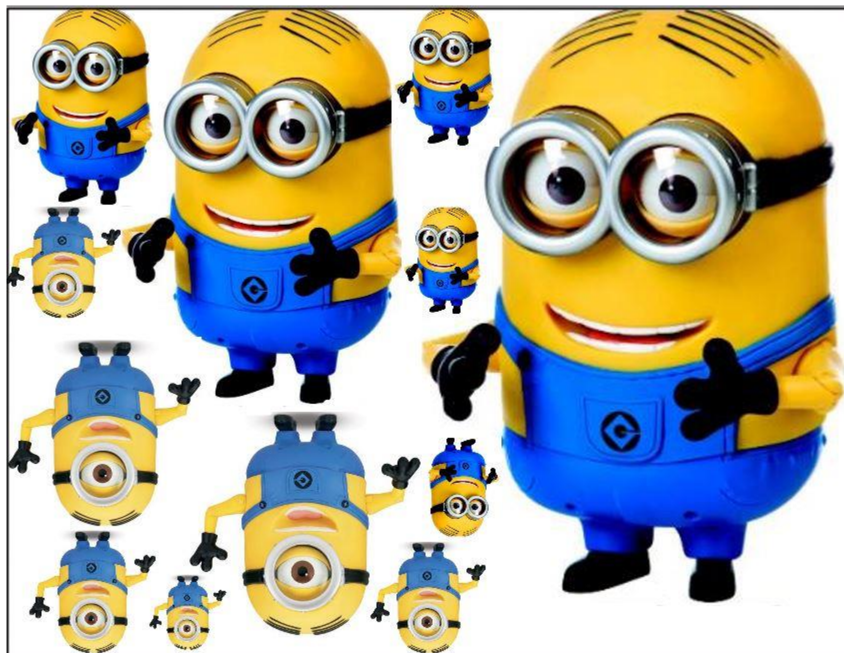


Figura 62: Applet 12: Exercíto de 2 Minions

A figura do professor mediador nessa atividade visa explorar o recurso da homotetia, no entanto, os alunos podem ficar livres para executar a atividade e produzir telas distintas uma da outra. Nesse caso, o mediador poderá intervir para as construções não se sobreporem ao mesmo tempo estabelecer novos desafios com o intuito de preencher ao máximo os espaços vazios, sugestionando inclusive a utilização de rotação de eixos para aumentar o número de Minions nos pontos inexistentes.

Avaliação

Ao concluir o processo de construção da tela repleta de Minions, espera-se que os alunos notem como esta transformação geométrica contribui para a redução de códigos, principalmente, quando desejamos construir a mesma imagem em diversos escalas. Além disso, a atividade proporciona uma nova habilidade que inclui a inclusão de figuras extraídas de qualquer meio digital nos formatos *jpeg* e *png*.

5.5.2 Aprimorando a homotetia

Objetivo

Aprimorar o domínio da transformação de homotetia no processo de construção do Triângulo de Sierpinski e automatizar tal construção com a implementação do código preservando o princípio de implementação do mesmo.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Homotetia.

Subsídio Teórico

O matemático polonês, Waclav Sierpinski (1882-1969), descobriu um padrão recursivo na divisão de um triângulo equilátero em quatro triângulos semelhantes, onde um destes encontra-se invertido em relação ao original e é retirado do triângulo inicial sobrando apenas os outros três. A recursividade da construção esta no fato de repetir o mesmo procedimento em cada um dos três novos triângulos com a orientação original de forma sucessiva.

Esse triângulo intitulado de Triângulo de Sierpinski (ver Figura 63) é considerado uma das formas elementares da geometria fractal por apresentar algumas propriedades como apresenta Santos, Silva e Rossy [41], tais como: possuir tantos pontos como o conjunto dos números reais; possuir área igual a zero; ser auto-semelhante, ou seja, as partes das figuras são cópias reduzidas de toda a figura; e não perder sua definição inicial à medida que é ampliado.

No entanto, antes de visualizarmos como proceder para a construção desse fractal, faz-se necessário definir o que é um fractal que teve sua noção inicial estabelecida pelo matemático francês, Benoit Mandelbrot, e para defini-lo utilizamos as palavras de Sallum [42, p.1]:

> Um fractal é uma figura que pode ser quebrada em pequenos pedaços,

> sendo cada um desses pedaços uma reprodução do todo. Não podemos ver um fractal porque é uma figura limite, mas as etapas de sua construção podem dar uma ideia da figura toda. Seu nome se deve ao fato de que a dimensão de um fractal não é um número inteiro.

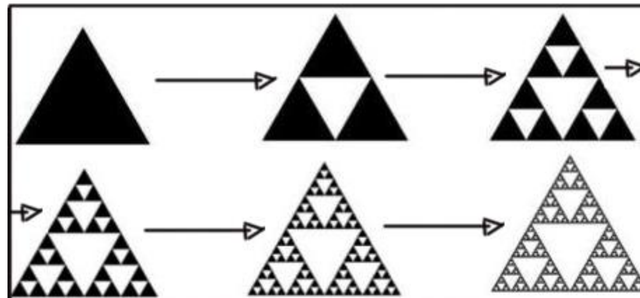


Figura 63: Triângulo de Sierpinski

Com base na definição podemos apresentar a noção de como é feito o Triângulo de Sierpinski e uma das maneiras de se obter esse triângulo é através do seguinte algoritmo divulgada por Santos, Silva e Rossy [41, p.4]:

1. Comece com um triângulo equilátero em um plano. O triângulo de Sierpinski canônico utilizava um triângulo equilátero com a base paralela ao eixo horizontal, mas qualquer triângulo pode ser usado;

2. Encolha o triângulo pela metade (cada lado deve ter metade do tamanho original), faça três cópias e posicione cada triângulo de maneira que encoste nos outros dois em um canto;

3. Repita o passo 2 para cada figura obtida, indefinidamente (ver a partir da terceira figura);

Pereira [43, p. 38] também apresenta em sua dissertação sobre Fractais Circulares como se obter esse triângulo e nas palavras do próprio autor:

> Para a construção do triângulo de Sierpinski construímos um triângulo equilátero e inscrevemos outro triângulo com vértices no ponto médio de cada um de seus lados. Retirando-se esse triângulo, temos 3 novos triângulos de mesmo tamanho. Em cada novo triângulo inscreve-se outro triângulo com vértice no ponto médio de cada uma de seus lados. Retirando-se o triângulo inscrito, temos 9 novos triângulos de emsmo tamanho. Repetindo se esse processo indefinidamente nos novos triângulos formados obter-se-á o fractal denominado Triângulo de Sierpinski, cuja área cada vez se torna menor.

O fractal propriamente dito só é obtido ao repetir o algoritmo infinitas vezes, no entanto, à medida que o número de interações aumenta, a imagem obtida tende a se tornar cada vez mais parecida com o fractal de Sierpinski.

Para implementar esse fractal de forma automatizada vamos construir uma nova função *void* com controle das coordenadas do triângulo equilátero para obter sempre novos triângulos pela metade do anterior e assim obtermos todo o fractal do Triângulo de Sierpinski.

Metodologia

Aula expositiva com construção do Triângulo de Sierpinski na malha quadriculada para obter um fragmento do fractal e construir no "Applet". Após a construção inicial, buscar automatizar a construção de todo o fractal com o mínimo de códigos no processo.

Material

Lápis, Caneta, Compasso, Malha quadriculada e Computador com o *Processing* instalado.

Procedimento

Para construir a noção do Triângulo de Sierpinski depois de fazer o molde na malha quadriculada utilizaremos a redução de escala combinada com uma simetria em relação ao eixo dos y e a translação para posicionar os triângulos em suas devidas coordenadas. Ao final apresentaremos o código obtido na página do OpenProcessing[6] para construção do fractal do Triângulo de Sierpinski.

Vamos começar preparando o applet para o lançamento dos códigos.

a) Abra um arquivo e salve como: **Applet 13 nome da dupla ou trio**.

b) Configuração inicial para o triângulo equilátero.

```
void setup(){
size(350,300);
background(255);
}
```

```
void draw(){
translate(30,290);
//Posição do triângulo inicial
molde(); }
```

A função "*molde*()" no referido código acima servirá de guia para construção mais imediata das miniaturas considerando o primeiro passo da aplicação já satisfeita (Reduzir

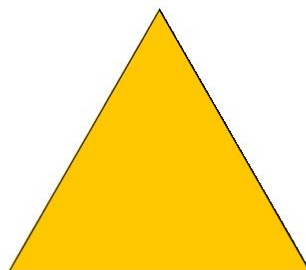
[6]Disponível para consulta no endereço *http://www.openprocessing.org/sketch/17026*.

o triângulo equilátero a sua metade e distribuir três cópias idênticas dessa redução nos cantos do triângulo original). Para construir essa função "*molde*()" basta aplicarmos os seguintes códigos para produzir a Figura 64.

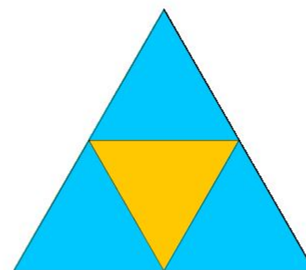
c) Implantação do código da função "*molde*()".

```
void molde(){
pushMatrix();
fill(255, 200, 0); //Cor amarela
triangulo(); //Triângulo Inicial
popMatrix();
pushMatrix();
scale(0.5); //Redução 50%
fill(0, 200, 255); //Cor azul
triangulo(); //Triângulo direito
popMatrix();
pushMatrix();
translate(150, 0); //Triângulo esquerdo
scale(0.5);
triangulo();
popMatrix();
translate(75, −130); //Triângulo superior
scale(0.5);
triangulo();
resetMatrix(); //Limpa as Transformações
}
void triangulo(){ //Triângulo inicial
scale(1, −1);
triangle(0, 0, 300, 0, 150, 150 ∗ sqrt(3));
}
```

O resultado dessa codificação servirá para agilizar o processo de construção do Triângulo de Sierpinski na tentativa de automatizar todo o processo. A próxima etapa é reproduzir esse "molde" de forma reduzida seguindo a proporção de $\frac{50\%}{2^{n-1}}$, onde n representa o número de reduções da figura inicial distribuídas nos três cantos onde aparece o triângulo azul da Figura 64.



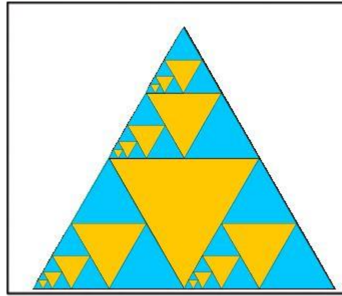
(a) Triângulo equilátero



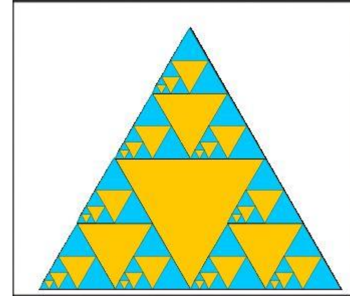
(b) Molde Sierpinski

Figura 64: Molde do Triângulo de Sierpinski

O próximo passo é replicar esse molde para construir as demais partes do Triângulo de Sierpinski aplicando os devidos deslocamentos com sua respectiva redução conforme podemos notar nos próximos códigos até resultar na Figura 65.



(a) Montagem do Triângulo



(b) Triângulo em construção

Figura 65: Applet 13: Construção do Triângulo de Sierpinski

d) Códigos para ampliação da quantidade de moldes do Triângulo de Sierpinski inseridos na função $voiddraw()$:

```
void draw(){
molde();
1 for(int j = 30; j <= 300; j = j + 150){
translate(j, 290);
scale(0.5/i);
molde();
translate(j, 290);
scale(0.25/i);
molde(); } }
2 for(int a = 1; a < 5; a = 2 * a){
for(int b = 30; b <= 300; b = b + 150){
translate(75 + b, 290);
scale(0.25/a);
molde();
translate(b, 290);
scale(0.25/a);
molde(); } }
3 for(int c = 1; c < 5; c = 2 * c){
for(int d = 30; d <= 300; d = d + 150){
translate(75/2 + d, 224);
scale(0.25/c);
molde();
translate(d, 290);
scale(0.25/c);
molde(); } }
4 for(int e = 1; e < 5; e = 2 * e){
translate(105, 160);
scale(0.5/e);
molde();
translate(105, 160);
scale(0.25/e);
molde();
translate(142, 95);
scale(0.25/e);
molde(); } }
5 for(int g = 1; g < 5; g = 2 * g){
for(int h = 30; h < 100; h = h + 150){
translate(150 + h, 160);
scale(0.25/g);
molde();
translate(h + 75, 160);
scale(0.25/g);
molde(); } }
}
```

Notem que mesmo utilizando expressões no processo de construção do Triângulo de Sierpinski, não conseguimos construir toda a imagem desse triângulo. Para isso, preci-

saríamos melhorar nossa expressão ou continuar o processo de construção já iniciado, isto é, sintetizamos essa expressão para construir o máximo de moldes ou condicionaremos de forma ilimitada a extensão do código já utilizado.

Em função disso, vamos construir um outro aplicativo para esse Triângulo utilizando a linguagem retirada do site da OpenProcessing.org como foi mencionado no procedimento da atividade.

a) Abra um novo arquivo e salve-o como **Applet 13.1 nome da dupla ou trio**.

b) Adicione os seguintes códigos para visualizar o resultado do Triângulo de Sierpinski de forma integral.

```
void setup(){
size(800,800);
background(255);
smooth();
noStroke();
colorMode(RGB,5,100,100);
triangleSier(0,700,400,0,800,700,7);
}

//Função do Triângulo de Sierpinski.

void triangleSier(float x1, float y1, float x2, float y2, float x3, float y3, int n){

//'n' é o número da interação.
if(n > 0){
fill(0,128/n,128);
triangle(x1,y1,x2,y2,x3,y3);
//Calcule os pontos médios de todos segmentos.
float h1 = (x1 + x2)/2;
float w1 = (y1 + y2)/2;
float h2 = (x2 + x3)/2;
float w2 = (y2 + y3)/2;
float h3 = (x3 + x1)/2;
float w3 = (y3 + y1)/2;
//Trace os triângulos com as novas coordenadas.
triangleSier(x1,y1,h1,w1,h3,w3,n − 1);
triangleSier(h1,w1,x2,y2,h2,w2,n − 1);
triangleSier(h3,w3,h2,w2,x3,y3,n − 1);
}
}
```

Como podemos constatar nos códigos para o Triângulo de Sierpinski, o algoritmo de programação utilizado estão bem reduzidos e nessa aplicação não foi utilizado nenhuma aplicação direta das transformações geométricas. Por este motivo, não iniciamos a atividade aplicando tais códigos que resultam no Triângulo completo de Sierpinski como mostra a Figura 66.

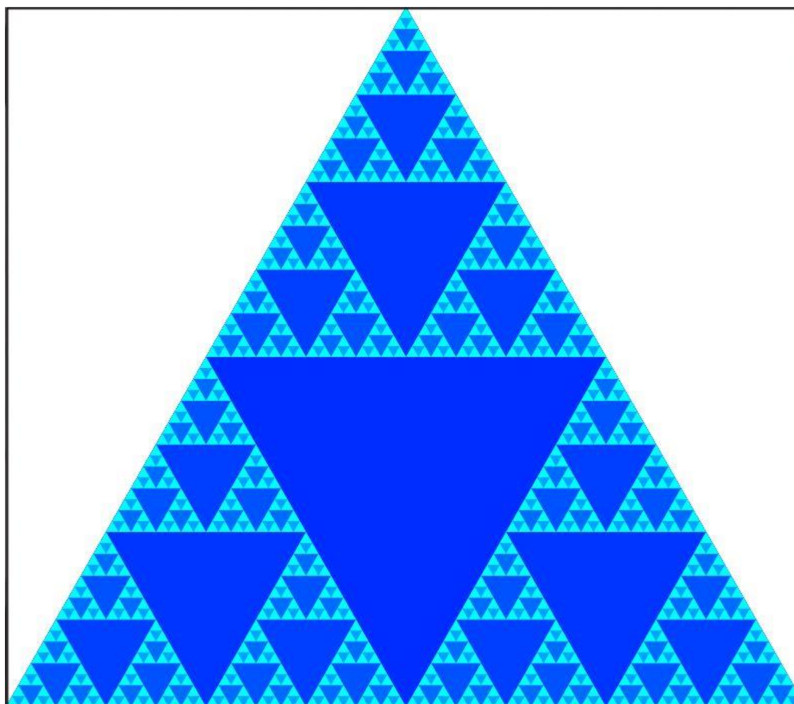


Figura 66: Applet 13.1: Triângulo de Sierpinski no *Processing*

Avaliação

Avaliar o domínio do conceito de homotetia e manuseio de expressões para otimizar o processo da construção de figuras entre outros itens. Com essa atividade ainda é possível avaliar o fluxograma da construção para otimizar o processo ao mesmo tempo que potencializa a descoberta dos fractais.

5.6 Transformação de dilatação/contração irregular

Quando ampliamos ou encolhemos uma figura em ambos sentidos ao mesmo tempo (vertical e horizontal) estamos empregando o conceito de homotetia de fator k. No entanto é possível dilatar ou comprimir a mesma figura em apenas um dos sentidos ou com fatores distintos, nesses casos dizemos que a figura transformada sofreu uma dilatação/contração irregular.

O processo de dilatação ou contração de figuras quando aplicado por razões distintas para o sentido horizontal e vertical não são classificadas como homotetias [6], existem ainda autores [37] que denominam essa transformação de anamórfica.

Para apresentar estes conceitos vamos construir um ambiente utilizando partes dos elementos de outros Applet's com o intuito de dilata-los e contraí-los.

Objetivo

Construir o conceito de dilatação/contração irregular e as potencialidades dessa transformação para geração de movimentos e animações gráficas.

Conteúdo Programático

Linguagem de Programação; Transformação Geométrica - Dilatação/Contração irregular, Translação.

Subsídio Teórico

Dilatar ou contrair uma figura quando realizado em ambos os sentidos gera uma imagem proporcional a original, no entanto, a dilatação/contração pode ser empregada em relação a um dos eixos ou em ambos com fatores distintos. As imagens de dilatação no sentido vertical esticam a figura dando uma impressão de emagrecimento da mesma, já no sentido horizontal proporcionam uma sensação de alargamento da figura. No caso da contração, o efeito encurta na horizontal e achata na vertical.

Metodologia

Aula expositiva com construção do "Applet" a partir dos códigos de outras aplicações (Applet's 2, 5 e 12) para proporcionar a sensação de dilatação e contração de forma desproporcional nos sentidos horizontais e verticais.

Material

computador com o *Processing* instalado.

Procedimento

A atividade propõe reaproveitar os códigos de funções construídas em outros applet's para otimizar a atividade e proporcionar uma análise dos efeitos de dilatação e contração desproporcional. O efeito gerado nessa atividade pode ser comparado com um labirinto de espelhos onde temos variados tipos de lentes esféricas.

Ao finalizar o lançamento dos dados, o resultado será a Figura 67 composta por diversas dilatações e contrações desproporcionais construídas com os seguintes códigos:

Inicialmente capturamos os códigos do boneco geométrico (Applet 2) e construímos uma aba "boneco geometrico" para a função *void boneco*() na aplicação do novo Applet 14.

a) Configuração do Applet para carregamento de imagens;

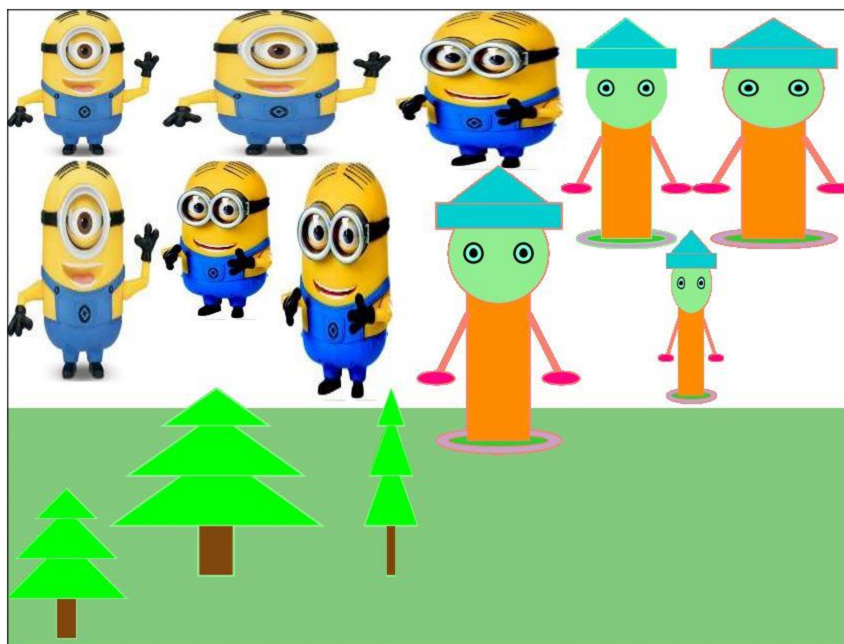


Figura 67: Applet 14: Dilatação e contração irregular

```
//Variável das imagens
PImagek, t;
size(850, 650);
//Imagem do Minion Dave
k = loadImage("MinionDave.jpg");
//Imagem do Minion Stuart
t = loadImage("MinionStuart.jpg");
background(255);
```

A figura do professor mediador nessa atividade visa explorar o recurso da homotetia, no entanto, os alunos podem ficar livres para executar a atividade e produzir telas distintas uma da outra. Nesse caso, o mediador poderá intervir para as construções não se sobreporem ao mesmo tempo estabelecer novos desafios com o intuito de preencher ao máximo os espaços vazios, sugestionando inclusive a utilização de rotação de eixos para aumentar o número de Minions nos pontos inexistentes.

Avaliação

Ao concluir o processo de construção da tela repleta de Minions, espera-se que os alunos notem como esta transformação geométrica contribui para a redução de códigos, principalmente, quando desejamos construir a mesma imagem em diversos escalas. Além disso, a atividade proporciona uma nova habilidade que inclui a inclusão de figuras extraídas de qualquer meio digital nos formatos *jpeg* e *png*.

6 Considerações Finais

Constatou-se neste trabalho que a implementação de uma linguagem de programação não é só interessante como viável para inserir os alunos no processo de construção da própria aprendizagem tendo em vista que os mesmos serão protagonistas na construção do seu conhecimento segundo a teoria construcionista. Salienta-se neste sentido a importância em definir qual a melhor linguagem para favorecer um domínio mínimo em programação com o intuito de otimizar o processo de construção da própria Matemática objetiva no emprego deste recurso tecnológico.

Após muitas pesquisas e análise das linguagens de programação, constatamos que a linguagem de programação *Processing* é a que melhor satisfaz nosso objetivo para inserção no processo pedagógico. Dentre os motivos desta escolha podemos listar o fato de ser uma linguagem gratuita, com versões para *Windows*, *Linux*, *Mac OS* e o próprio sistema *Android* para *Smartphones*, além de ser uma linguagem de fácil programação pensada para Designers Visuais, Artes Eletrônicas e não-programadores com a intenção de se inserir no universo da Programação.

O trabalho apresentou um percurso histórico da computação na educação visando analisar a evolução das linguagens e sua contribuição matemática. Após conhecer como surgiu os variados tipos de linguagens focamos em nossa linguagem de estudo, *Processing*, para enfim apresentarmos as TGPs e como podemos interagir via Programação. As possibilidades de aprendizagem ultrapassam as barreiras da própria tecnologia, uma vez que a maioria das construções fora pensada para ser trabalhada na malha quadriculada com o intuito de planejar e otimizar o algoritmo empregado na linguagem.

Além disso, as atividades seguiram uma orientação pedagógica abordando seus objetivos, materiais empregados, conteúdos visualizados, um suporte teórico e ainda um procedimento metodológico com a sequência de construção evolutiva, isto é, a medida que os alunos são convidados a participar do programa, os mesmos são estimulados a novas descobertas ao tempo que demonstram o domínio/conhecimento adquirido a cada nova etapa. Em função disso, algumas de nossas atividades discutem com os discentes a cerca de

algumas passagens sugestionando inclusive o motivo da escolha de determinado elemento em detrimento de outrem, já que a proposta visa o trabalho em equipe, a valorização de ideias e sua organização, e o incentivo a construção tendo como base o desenho em uma malha quadriculada.

Dentre as dificuldades para participação no Curso de Programação no Khan Academy e aprendizagem da linguagem para posterior implementação das atividades propostas na Unidade Escolar, podemos elucidar os problemas de configuração do sistema de rede *Wifi* e uma rede de cabeametno insuficiente para todos os 12 computadores em funcionamento em função da versão do Linux Educacional e suporte técnico por parte da Prefeitura para solucionar estes empecilhos.

O processo de construção das atividades buscou conceituar e interpretar as Transformações Geométricas através da manipulação de dados e deslocamento dinâmico dos construtos de forma a contribuir para uma aprendizagem lúdica com potencialidade para a descoberta de novas competências e habilidades ao mesmo tempo que revela aos envolvidos uma Matemática aplicável com viabilidade de avançarem tanto nos conceitos matemáticos como na programação para construção de jogos eletrônicos quando atingir um nível de domínio das ferramentas computacionais.

Por isso esperamos que o trabalho sirva como fonte de consulta para os professores de Matemática em ambos os níveis de ensino, Fundamental e Médio, de forma a contribuir para a minimização das dificuldades dos alunos em compreender como se comportam os objetos em movimentação preservando algumas de suas características, e isso, potencializa também o estudo de funções quando referenciamos o posicionamento de seus gráficos como um dos objetos geométricos em estudo. Salienta-se ainda que não objetivamos substituir nenhum conteúdo matemático, ao contrário, a implementação dessa tecnologia visa reforçar a aprendizagem matemática com manipulação de dados em conjunto com o recurso visual e investigação matemática para atingir nossos objetivos.

Referências Bibliográficas

[1] BRASIL. Parâmetros Curriculares Nacionais: Matemática. Brasília: Ministério da Educação/Secretária de Educação Fundamental, 1998.

[2] LIMA, E. L. Conceituação, manipulação e aplicação: os três componentes do ensino de matemática. Revista do Professor de Matemática – SBM, n. 41, p. 1–6, 1999.

[3] DEVLIN, K. Matemática - a ciência dos padrões. Porto: Porto Editora, 2002.

[4] CANGUSSÚ, E. S. O Ensino de Sequências de Recorrências na Educação Básica com o auxílio de Linguaguem de Programação. Disserta (Mestrado Profissional em Matemática) — Universidade Federal do Maranhão - UFMA, São Luís, 2013.

[5] BRASIL. PCN+ Ensino Médio: Orientações Educacionais Complementares aos Parâmetros Curriculares Nacionais. Ciências da Natureza, Matemática e suas Tecnologias. Brasília: Ministério da Educação/Secretária de Educação Média e Tecnológica, 2002.

[6] STORMOWSKI, V. Estudando matrizes a partir de transformações geométricas. Disserta (Mestrado Profissional em Ensino de Matemática) — Universidade Federal do Rio Grande do Sul - UFRGS, Porto Alegre, 2008.

[7] CERQUEIRA, A. P. F. de. Isometrias: análise de documentos curriculares e uma proposta de situação de aprendizagem para o ensino médio. Disserta (Mestrado Profissional em Ensino de Matemática) — Pontifícia Universidade Católica de São Paulo - PUC-SP, São Paulo, 2008.

[8] VALENTE, J. A. O uso inteligente do computador na educação. Pátio - Revista Pedagógica, Editora Artes Médicas Sul, n. 1, p. 19–21, 1 1997. Disponível em: *http://www.educacaopublica.rj.gov.br/biblioteca/educacao/0024.html*. Acesso em 02 ago 2014.

[9] VALENTE, J. A. Computadores e conhecimento: Repensando a educação. In: ______. 2ª. ed. Campinas, SP: UNICAMP/NIED, 1998. cap. 1 e 2, p. 1 a 53. Disponível em: *http://www.nied.unicamp.br/?q=system/files/Computadores%20e%20Conhecimento.pdf*. Acesso em 02 ago 2014.

[10] VALENTE, J. A. O computador na sociedade do conhecimento. In: ______. Campinas, SP: OEA-NIED/UNICAMP, 1999. cap. 1. Disponível em: *http://www.nied.unicamp.br/oea/pub/livro1/cap1.zip*. Acesso em 02 ago 2014.

[11] FRANÇA, J. B. dos A. Novas tecnologias no ensino da matemática: Formação inicial de professores. Artigo de Pós-Graduação em Docência do Ensino Superior pela Associação Bahiana de Educação e Cultura – ABEC. Disponível em:

https://drive.google.com/file/d/0ByHgpHWSevqEOUpBOWM5Qmo0TDg/view?pli=1. 2009.

[12] GIRALDO, V.; CAETANO, P. A. S.; MATTOS, F. R. P. Recursos Computacionais no Ensino de Matemática. 1ª. ed. Rio de Janeiro: SBM, 2012. (Coleção PROFMAT).

[13] SORDO JUANENA, J. M. Estudio de una estrategia didáctica basada en las nuevas tecnologías para la enseñanza de la geometría dinámica. Tese (Facultad de Educación) — Universidad Complutense de Madrid, Madrid, 2005. Disponível em *http://biblioteca.ucm.es/tesis/edu/ucm-t28911.pdf*. Acesso em 10 dez 2014.

[14] BRASIL. Parâmetros Curriculares Nacionais: Ensino Médio. Brasília: Ministério da Educação/Secretária de Educação Média e Tecnológica, 1999.

[15] BORBA, M. de C. Tecnologias informáticas na educação matemática e reorganização de pensamento. In: BICUDO, M. A. V. (Ed.). Pesquisa em Educação Matemática: concepções e perspectivas. São Paulo: UNESP, 1999. p. 285–295.

[16] REIS, L. R. M. Estratégias de aprendizagem ativa para reduzir o fracasso escolar: Papel do psicopedagogo. Disponível em *http://proerdpmdf.files.wordpress.com/2010/07/aprendizagem-ativa1.pdf*. Acesso em 10 jan 2015.

[17] ALMEIDA, M. E. B. de. Tecnologias na educação: dos caminhos trilhados aos atuais desafios. In: ______. Rio Claro: UNESP, 2008. v. 21, n. 29, cap. 4, p. 99 a 129. BOLEMA. Disponível em *http://www.periodicos.rc.biblioteca.unesp.br/index.php/bolema/article/viewFile/1723/1497*. Acesso em 10 dez 2014.

[18] POCRIFKA, D. H.; SANTOS, T. W. Linguagem logo e a construção do conhecimento. In: PUCPR. IX Congresso Nacional de educação e III Encontro Sul Brasileiros de Psicopedagogia. Curitiba: PUCPR, 2009. p. 2469 a 2479. Disponível em *http://www.pucpr.br/eventos/educere/educere2009/anais/pdf/2980_1303.pdf*. Acesso em 12 dez 2014.

[19] LEMOS JUNIOR, J. A. S. Estudo de Funções Afins e Quadráticas com o auxílio do Computador. Disserta (Mestrado Profissional em Matemática em Rede Nacional - PROFMAT) — Universidade Federal de Campina Grande - UFCG, Campina Grande, Jul 2013.

[20] GONÇALVES, M. B.; LENTZ, C. R.; PEREIRA, R. Matemática e Informática. Florianópolis: Laboratório de Ensino à Distância, 2001. (Programa de Formação Continuada à Distância).

[21] LÉVY, P. As Tecnologias da Inteligência: o futuro do pensamento na era da informática. 13ª. ed. São Paulo: Editora 34, 2004.

[22] MORAES, M. C. Informática educativa no brasil: Uma história vivida, algumas lições aprendidas. Revista de Informática na Educação, v. 1, n. 1, p. 19–44, Abril 1997. Disponível em: *http://www.br-ie.org/pub/index.php/rbie/article/view/2320/2082*. Acesso em 12 dez 2014.

[23] FONSECA FILHO, C. História da computação: O Caminho do Pensamento e da Tecnologia. Porto Alegre: EDIPUCRS, 2007. Disponível em: *http://www.pucrs.br/edipucrs/online/historiadacomputacao.pdf*. Acesso em 05 ago 2014.

[24] EUCLIDES. Os elementos/ Euclides. São Paulo: UNESP, 2009. Tradução e introdução de Ireneu Bicudo.

[25] OLIVEIRA, Z. C. de. Uma interpretação geométrica do MDC. Revista do Professor de Matemática – SBM, n. 29, p. 24–26, 1995.

[26] GUDWIN, R. R. Linguagens de programação – mini e microcomputadores: *Software*. UNICAMP, 1997. Notas de aula para disciplina EA877. Disponível em *ftp://ftp.dca.fee.unicamp.br/pub/docs/ea877/lingpro.pdf*. Acessado em 10 jan 2015.

[27] REAS, C.; FRY, B. Processing: a programming handbook for visual designers and artists. Foreword by John Maeda. Cambridge, Massachusetts EUA: MIT Press books, 2007.

[28] AMADO, P. Introdução à Programação Gráfica (Usando Processing. Porto, Portugal: FBAUP - Faculdade de Belas Artes, 2007. Disponível em *http://repositorio-aberto.up.pt/handle/10216/1848*. Acesso em 10 dez 2014.

[29] MEDEIROS, L. G. F. de. DANDO MOVIMENTO À FORMA: as transformações geométricas no plano na formação continuada a distância de professores de matemática. Disserta (Dissertação de Mestrado Profissional em Educação Matemática) — Universidade Severino Sombra, USS, Brasil, Vassouras, Abr 2012. Disponível em: *http://www.uss.br/arquivos/posgraduacao/strictosensu/educacaoMatematica/dissertacoes/2012/Dissertacao_LICIA_GIESTA_FERREIRA_DE_MEDEIROS.pdf*. Acesso em 05 dez 2014.

[30] LOPES, C. F. Escher: O gênio da arte matemática. Jornal dá Licença, Ano XIV, n. 40, p. 8, mai 2009. Disponível em: *http://www.uff.br/dalicenca/images/stories/jornais/jornal40novo.pdf*. Acesso em 15 dez 2014.

[31] FAINGUELERNT, E. K. Educação Matemática: Representação e Construção em Geometria. Porto Alegre: Artes Médicas Sul, 1999.

[32] KLEIN, F. O Programa Erlangen de Félix Klein. São Paulo: Intituto de Física/USP, 1984. Disponível em: *http://publica-sbi.if.usp.br/PDFs/pd499*. Acesso em: 05 jan 2015.

[33] CONCEIÇÃO, M. R. F. Transformações no Plano: Uma aplicação do Estudo de Matrizes com o Uso de Planilhas Eletrônicas. Disserta (Mestrado Profissional em Matemática em Rede Nacional - PROFMAT) — Universidade Federal do Rio Grande – FURG, Rio Grande, mar 2013.

[34] WAGNER, E. Construções geométricas. In: ______. 6ª. ed. Rio de Janeiro: Sociedade Brasileira de Matemática-SBM, 1994. v. 1, cap. Transformações Geométricas, p. 70–90.

[35] DOMINGUES, H. H.; IEZZI, G. Álgebra Moderna. 4. ed. São Paulo: Atual, 2003.

[36] SOUZA, D. M. de. Uso de Transformações Geométricas na Revigoração do Ensino de Geometria Plana. Disserta (Mestrado Profissional em Matemática em Rede Nacional - PROFMAT) — Instituto Nacional de Matemática Pura e Aplicada – IMPA, Rio de Janeiro, 2014.

[37] LIRA, A. C. de B. A matemática dos espelhos: Proposta para o ensino-aprendizagem de matrizes utilizando transformações geométricas. Disserta (Especialização em Educação Matemática para Professores do Ensino Médio) — Universidade Estadual da Paraíba - UEPB, Campina Grande, 2011.

[38] GOMES, J.; VELHO, L. Fundamentos da Computação Gráfica. 1ª. ed. Rio de Janeiro: IMPA, 2008. (Série de Computação e Matemática).

[39] LAGE, M. A. Mobilização das formas de pensamento matemático no estudo de transformações geométricas no plano. Disserta (Mestrado em Ensino de Matemática) — Pontifícia Universidade Católica de Minas Gerais - PUC MINAS, Belo Horizonte, 2008.

[40] DELGADO, J.; FRENSEL, K.; CRISSAFF, L. Geometria Analítica. Rio de Janeiro: SBM, 2013. (Coleção PROFMAT).

[41] SANTOS, E. R. M. dos; SILVA, J. do Socorro Costa da; ROSSY, N. da cunha. Utilizando a geometria fractal na introdução do conceito de potência para uma turma do projovem urbano. In: SBEM. X Encontro Nacional de Educação Matemática: Educação Matemática, Cultura e Diversidade. Salvador-BA, 2010. Disponível em: *http://www.lematec.net/CDS/ENEM10/artigos/RE/T17_RE1694.pdf*. Acesso em: 10 jan 2016.

[42] SALLUM Élvia M. Fractais no ensino médio. Revista do Professor de Matemática, n. 57, p. 1–8, 2º quadrimestre 2005.

[43] PEREIRA, A. S. Fractais circulares: algumas considerações e atividades. Disserta (Mestrado Profissional em Matemática - PROFMAT) — Universidade Estadual de Londrina, Londrina, 2013.

[44] FRANÇA, J. B. dos A. et al. (Ed.). Transformações de Funções: função afim, função quadrática e função exponecial. Salvador: UFBA, 2014. Trabalho da Disciplina MA36 do PROFMAT.

[45] VALE, I. et al. Os padrões no ensino e aprendizagem Álgebra. In: ISABEL VALE, TERESA PIMENTEL, ANA BARBOSA AND L. FONSECA AND L. SANTOS AND P. CANAVARRO. Números e Álgebra. Lisboa: SEM-SPCE, 2007. p. 193 a 211. Disponível em: *http://www.rdpc.uevora.pt/bitstream/10174/1416/1/PadrAcesso em: 20 out 2015.*

[46] LIMA, E. L. Álgebra Linear. 8. ed. Rio de Janeiro: IMPA, 2011.

[47] BOLDRINI, J. L. Álgebra Linear. 3. ed. São Paulo: Harbra Ltda, 1986.

[48] HEFEZ, A.; FERNANDEZ, C. de S. Introdução à Álgebra Linear. Rio de Janeiro: SBM, 2012. (Coleção PROFMAT).

APÊNDICE A – Transformações Lineares

As funções cujos domínios e contradomínios são espaços vetoriais e que preservam as operações de adição de vetores e de multiplicação de um vetor por um escalar são as funções objetivadas pela Álgebra Linear. Essa função é definida da seguinte forma:

Definição A.1 (Transformação Linear)**.** *Sejam V e W espaços vetoriais*[1]*. Uma transformação linear de V em W é uma aplicação (função) $T : V \to W$ que possui as seguintes propriedades:*

(i) $T(v_1 + v_2) = T(v_1) + T(v_2)$, para quaisquer v_1 e v_2 em V;

(ii) $T(\alpha v) = \alpha T(v)$, para quaisquer v em V e α em $\mathbb{R}$.

As propriedades (i) e (ii) são equivalentes à seguinte propriedade:

$$T(v_1 + \alpha v_2) = T(v_1) + \alpha T(v_2),$$

para quaisquer v_1 e v_2 em V e para qualquer α em $\mathbb{R}$.

Exemplo A.1. *Sejam $V = \mathbb{R}$ e $W = \mathbb{R}$ e $T : \mathbb{R} \to \mathbb{R}$ definida por $T(m) = km$, $k \in \mathbb{R}$. Dados $u, v \in \mathbb{R}$, $\alpha \in \mathbb{R}$, temos:*

$$T(u + \alpha v) = k(u + \alpha v) = ku + k\alpha v = ku + \alpha(kv) = T(u) + \alpha T(v),$$

segue que T é uma transformação linear.

De fato, toda a transformação linear de $\mathbb{R}$ em $\mathbb{R}$ só pode ser deste tipo.

[1] O leitor interessado poderá encontrar as definições de espaço vetorial, subespaços vetoriais e bases em livros de Álgebra Linear dos quais recomendo [46], [47] e [48]

Exemplo A.2. *As transformações lineares $T : \mathbb{R}^2 \to \mathbb{R}^2$, definidas por:*

1 $T(x, y) = (2x, 2y)$ são "esticamentos" na imagem da aplicação;

2 $T(x, y) = (-y, x)$ é uma rotação de 90º no sentido anti-horário;

3 $T(x, y) = (y, x)$ é uma reflexão (espelhamento) com relação à reta $y = x$;

4 $T(x, y) = (x, 0)$ é uma projeção ortogonal no eixo Ox;

Teorema A.1. *A toda matriz $m \times n$ está associada uma transformação linear de $\mathbb{R}^n$ em $\mathbb{R}^m$ e a toda transformação linear de $\mathbb{R}^n$ em $\mathbb{R}^m$ está associada uma matriz $m \times n$.*
***Demonstração**:*
Sejam $V = \mathbb{R}^n$ e $W = \mathbb{R}^m$ e A a matriz $m \times n$. Definimos

$$M_A : \mathbb{R}^n \to \mathbb{R}^m \text{ por } v \mapsto A \cdot v$$

onde $v = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$, *assim:* $M_A(v) = A \cdot \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$
das propriedades de operações com matrizes:
$M_A(u + \alpha v) = A(u + \alpha v) = Au + A\alpha v = Au + \alpha Av = M_A(u) + \alpha M_A(v)$, *e portanto M_A é uma transformação linear.*

Por outro lado, dada uma transformação linear, associamos uma matriz da seguinte forma:

Seja $T : V \to W$ uma transformação linear, em que $dim(V) = n$ e $dim(W) = m$. Sejam $\alpha = (v_1, \cdots, v_n)$ e $\beta = (w_1, \cdots, w_m)$ bases de V e W, respectivamente. Como β é uma base de W, podemos determinar de modo único números reais a_{ij}, com $1 \leq i \leq n$ e $1 \leq j \leq m$, tais que

$$T(v_i) = a_{1i}w_1 + \cdots + a_{ji}w_j + \cdots + a_{mi}w_m. \tag{A.1}$$

Tomemos agora v em V. Temos que $v = k_1v_1 + \cdots + k_nv_n$, em que $k_i \in \mathbb{R}$ para $1 \leq i \leq n$. Pela linearidade de T e por (A.1), segue que

$$\begin{aligned} T(v) &= k_1T(v_1) + \cdots + k_nT(v_n) \\ &= k_1(a_{11}w_1 + \cdots + a_{m1}w_m) + \cdots + k_n(a_{1n}w_1 + \cdots + a_{mn}w_m) \\ &= (a_{11}k_1 + \cdots + a_{1n}k_n)w_1 + \cdots + (a_{m1}k_1 + \cdots + a_{mn}kn)w_m. \end{aligned}$$

Logo,

$$[T(v)]_\beta = \begin{bmatrix} a_{11}k_1 + \cdots + a_{1n}k_n \\ \vdots \\ a_{m1}k_1 + \cdots + a_{mn}kn \end{bmatrix}$$

$$= \begin{bmatrix} a_{11} + \cdots + a_{1n} \\ \vdots \\ a_{m1} + \cdots + a_{mn} \end{bmatrix} \begin{bmatrix} k_1 \\ \vdots \\ kn \end{bmatrix} = [T(v)]^\alpha_\beta \cdot [v]_\alpha \qquad \text{(A.2)}$$

Onde definimos

$$[T]^\alpha_\beta = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

. A matriz $[T]^\alpha_\beta$, que representa T em relação às bases α e β, é chamada *matriz de T nas bases α e beta*. Por (A.2), obtemos a expressão

$$[T]_\beta = [T]^\alpha_\beta \cdot [v]_\alpha \text{ para todo } v \text{ em } V.$$

Fica assim estabelecida a bijeção entre transformações lineares e matrizes.

Considere $[T]^\alpha_\beta = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ como uma aplicação $M_A : \mathbb{R}^2 \to \mathbb{R}^2$ em relação a base canônica do $\mathbb{R}^2$. Sendo assim, aplicando a equação (A.2) para um vetor $v = (x, y) \in \mathbb{R}^2$, temos:

$$[T]_\beta = [T]^\alpha_\beta \cdot [v]_\alpha = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \end{bmatrix}$$

Note que esta é a aplicação linear que representa o "esticamento" listada no exemplo (A.2).

APÊNDICE B – Transformações Projetivas

Considere o modelo afim do espaço projetivo n-dimensional, $\mathbb{RP}^n$, onde os pontos projetivos são retas passando pela origem de $\mathbb{R}^{n+1} - \{0\}$. Uma transformação projetiva transforma pontos do espaço projetivo, logo, T deve transformar uma reta pela origem de $\mathbb{R}^{n+1}$ noutra reta que também passa pela origem, segundo a ótica euclidiana. Portanto, T é uma transformação linear invertível $T : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$, do espaço euclidiano $\mathbb{R}^{n+1}$.

Dessa forma, concluímos que uma transformação projetiva de $\mathbb{RP}^n$ preserva os elementos lineares do espaço projetivo e além disso é representado por uma matriz de ordem $n+1$.

É notório que se $T : \mathbb{RP}^n \to \mathbb{RP}^n$ é uma transformação projetiva e $\lambda \in \mathbb{R}, \lambda \neq 0$, então pela linearidade de T, temos:

$$(\lambda T)P = T(\lambda P) = T(P)$$

Daí, uma transformação projetiva fica definida a menos de um produto por um escalar não nulo [38].

B.1 Transformação Projetiva Plana

A transformação projetiva $T : \mathbb{RP}^2 \to \mathbb{RP}^2$ do plano projetivo é dada por uma transformação linear invertível $T : \mathbb{R}^3 \to \mathbb{R}^3$. Por isso, representamos na forma de uma matriz M de ordem 3 invertível. Para compreender como essa matriz atua no plano projetivo, vamos dividi-la em 4 blocos.

Seja

$$M = \left(\begin{array}{cc|c} a & b & t_x \\ c & d & t_y \\ \hline f_1 & f_2 & k \end{array}\right) = \begin{pmatrix} A & T \\ F & H \end{pmatrix},$$

onde

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad F = \begin{pmatrix} f_1 & f_2 \end{pmatrix}, \quad T = \begin{pmatrix} t_x \\ t_y \end{pmatrix} \quad e \quad H = \begin{pmatrix} k \end{pmatrix} \tag{B.1}$$

Sabe-se que a matriz

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

em (B.1) é uma matriz de transformação geométrica em cada caso apresentado nas transformações isométricas (secção 3.2), isomórfica (secção 3.3) e anamórfica (secção 3.4). Com isso, vamos analisar o comportamento dos blocos F, T e H constante em (B.1).

Supondo que

$$F = \begin{pmatrix} 0 & 0 \end{pmatrix}, \quad T = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad e \quad H = \begin{pmatrix} 1 \end{pmatrix},$$

a matriz da transformação projetiva (B.1) será dada por:

$$\left(\begin{array}{cc|c} a & b & 0 \\ c & d & 0 \\ \hline 0 & 0 & 1 \end{array}\right)$$

Nesse caso, aplicando a transformação a um ponto do infinito $(x, y, 0)$ ou a um ponto afim $(x, y, 1)$ do plano projetivo, suas imagens serão dadas, respectivamente, por:

$$\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \\ 0 \end{pmatrix} \qquad e \qquad \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \\ 1 \end{pmatrix}$$

Onde o ponto resultante é um ponto do infinito e um ponto afim, nesta ordem.

Salienta-se que em ambos os casos as coordenadas afins dos pontos transformados são dadas por:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} ax + by \\ cx + dy \end{pmatrix}.$$

Fundamentando que o bloco A é a matriz de uma transformação linear no espaço euclidiano aplicado ao espaço projetivo. Portanto, o grupo das transformações projetivas do plano contém, naturalmente, o grupo das transformações lineares do plano euclidiano, em especial, o grupo dos movimentos rígidos da Geometria Euclidiana Plana.

Da análise das transformações afins é notório que a matriz

$$\begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

representa uma transformação linear do plano seguida de uma translação, facilmente confirmada na seguinte situação onde:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad F = \begin{pmatrix} 0 & 0 \end{pmatrix}, \quad e \quad H = \begin{pmatrix} 1 \end{pmatrix}$$

Daí;

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ 1 \end{pmatrix}$$

Portanto, essa aplicação resulta de uma translação de um vetor $\overrightarrow{v} = (t_x, t_y)$. Por outro lado, o elemento k, que constitui o bloco H da matriz, corresponde a uma homotetia do plano afim de fator $\frac{1}{k}$, $k \neq 0$. Logicamente comprovado, visto que:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & k \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ k \end{pmatrix} = \begin{pmatrix} \frac{x}{k} \\ \frac{y}{k} \\ 1 \end{pmatrix}$$

Com base nos dados apresentados, as transformações projetivas preservam os pontos afins e os pontos ideais. Ou seja, segundo as palavras de Velho [38, p.42]: "O grupo das transformações projetivas contém o grupo das transformações afins (e portanto os movimentos rígidos da geometria Euclidiana)".

Nota-se que falta apenas o bloco F da matriz M em (B.1) a ser identificado. Para isso, tomemos o bloco A como sendo a matriz identidade, o bloco T nulo e $k = 1$. Aplicando a transformação no ponto afim $(x, y, 1)$, obtemos:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ f_1 & f_2 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ f_1 x + f_2 y + 1 \end{pmatrix}$$

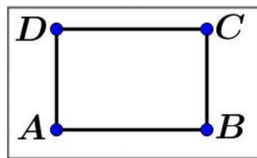
Se $f_1 \neq 0$ ou $f_2 \neq 0$, a equação $f_1 x + f_2 y + 1 = 0$ possuirá uma infinidade de soluções. Isso revela que os pontos afins $(x, y, 1)$ serão transformados em pontos do infinito $(x, y, 0)$ do plano projetivo.

No entanto, se a aplicação da transformação for em um ponto ideal $(x, y, 0)$, teremos:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ f_1 & f_2 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \\ f_1 x + f_2 y \end{pmatrix}$$

Com isso, concluímos que se $f_1 x + f_2 y \neq 0$, então os pontos do infinito do plano projetivo são transformados em pontos do plano afim. Nessas situações dizemos que um ponto ideal foi transformado em um ponto P_0 do plano afim e a família de retas paralelas que interceptam nesse ponto ideal, são transformados em uma família de retas incidentes no ponto P_0 e esse ponto P_0 é chamado de ponto de fuga da transformação.

A existência dos pontos de fuga é controlada pelos elementos f_1 e f_2, na matriz M da transformação projetiva. Onde $f_1 \neq 0$ e $f_2 = 0$ é o ponto de fuga em relação ao eixo Ox, do contrário, se $f_1 = 0$ e $f_2 \neq 0$, o ponto de fuga será em relação ao eixo Oy.



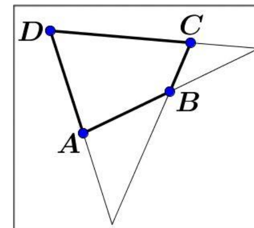


Figura 68: Transformação com dois pontos de fuga

Nota-se que uma transformação projetiva $T : \mathbb{RP}^2 \to \mathbb{RP}^2$ definida em coordenadas homogêneas pela matriz

$$\begin{pmatrix} a & b & t_x \\ c & d & t_y \\ f_1 & f_2 & k \end{pmatrix}$$

Associa um ponto euclidiano (x, y) do plano a uma imagem $z = (x, y, 1)$ para escrever a transformação na forma de produto, isto é:

$$T(x, y, 1) = \begin{pmatrix} a & b & t_x \\ c & d & t_y \\ f_1 & f_2 & k \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Salienta-se nesse ponto que as transformações geométricas abordadas no trabalho não enfatiza a existência de pontos de fugas, adotaremos o bloco F em (B.1)sempre na forma nula. Além disso, o bloco A será apresentado de acordo com o tipo de transformação geométrica, enquanto o bloco T só possuirá valor diferente de zero nas situações em que

haja translação somente ou em composição com outra transformação. E nos casos em que haja uma homotetia a constante k terá valor diferente de 1. Nesse sentido, a transformação anterior pode ser escrita da seguinte forma:

$$T(x, y, 1) = \begin{pmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & k \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = (ax + by + t_x, cx + dy + t_y, k)$$

Se $k \neq 0$, é possível dividir $(ax + by + t_x, cx + dy + t_y, k)$ por k, e assim obter o mesmo ponto escrito na forma

$$T(x, y) = T(x, y, 1) = \left(\frac{ax + by + t_x}{k}, \frac{cx + dy + t_y}{k} \right) \tag{B.2}$$

A equação (B.2) expressa uma transformação projetiva em coordenadas euclidianas de $\mathbb{R}^2$.

APÊNDICE C – Mosaico para caleidoscópio

O Mosaico da atividade da secção 5.4.2 foi pensada de forma a oportunizar os alunos na construção dos códigos de cada figura plana. Para auxiliar o docente vamos disponibilizar o código de cada figura para construir o mosaico básico para nosso caleidoscópio com movimento contínuo.

a) Construção das figuras como funções na aba "peças":

```
1 void trevo(){
lima();
beginShape();
vertex(0, 0);
vertex(0, 30);
vertex(10, 40);
vertex(20, 30);
vertex(10, 20);
vertex(20, 10);
vertex(30, 20);
vertex(40, 10);
vertex(30, 0);
endShape(CLOSE);
}
2 void pent(){
azceu();
beginShape();
vertex(0, 0);
vertex(10, −10);
vertex(20, −10);
vertex(30, 0);
vertex(30, 10);
vertex(0, 10);
endShape(CLOSE);
}
3 void pm(){
vermelho();
quad(0, 0, 10, 10, 30, 10, 40, 0);
translate(20, 10);
azvio();
beginShape();
vertex(0, 0);
vertex(10, 0);
vertex(20, −10);
vertex(30, −10);
vertex(40, 0);
vertex(50, −10);
vertex(60, −10);
vertex(70, 0);
vertex(80, 0);
vertex(70, 10);
vertex(60, 0);
vertex(40, 20);
vertex(20, 0);
vertex(10, 10);
endShape(CLOSE);
}
4 void y(){
verde();
beginShape();
vertex(0, 0);
vertex(10, −10);
vertex(30, 10);
vertex(50, −10);
vertex(60, 0);
vertex(30, 30);
endShape(CLOSE);
beginShape();
vertex(0, 0);
vertex(0, 10);
```

```
vertex(20,30);
vertex(20,40);
vertex(30,50);
vertex(30,30);
endShape(CLOSE);
translate(30,30);
beginShape();
vertex(0,0);
vertex(0,20);
vertex(10,10);
vertex(10,0);
vertex(30,−20);
vertex(30,−30);
endShape(CLOSE);
}
5 void a(){
peru();
beginShape();
vertex(0,0);
vertex(10,10);
vertex(20,10);
vertex(10,20);
vertex(10,30);
vertex(20,40);
vertex(30,30);
vertex(40,40);
vertex(40,0);
endShape(CLOSE);
translate(20,20);
fill(random(256));
quad(0,0,0,10,10,0,10,−10);
}
6 void forma1(){
salmao();
beginShape();
vertex(0,0);
vertex(0,20);
vertex(10,30);
vertex(10,40);
vertex(20,40);
vertex(30,50);
vertex(40,50);
vertex(50,40);
vertex(40,30);
vertex(60,10);
vertex(60,0);
vertex(30,0);
vertex(30,−10);
vertex(20,−20);
vertex(10,−10);
vertex(20,0);
vertex(10,10);
endShape(CLOSE);
verde();
translate(30,20);
triangle(0,0,10,0,10,10);
translate(0,20);
scale(1,−1);
triangle(0,0,10,0,10,10);
}
7 void peixe(){
vermelho();
quad(0,0,20,−20,40,0,20,20)
translate(10,0);
fill(random(0,256));
ellipse(0,0,5,5); //olho
line(10,−20,10,−10);
translate(0,30);
scale(1,−1);
//caudas
fill(random(256));
triangle(0,0,10,0,10,10);
translate(20,0);
scale(−1,1);
triangle(0,0,10,0,10,10);
}
8 void gato(){
//rosto
ouro();
beginShape();
vertex(0,0);
vertex(0,30);
vertex(20,50);
vertex(40,30);
vertex(40,0);
vertex(30,−10);
vertex(20,0);
vertex(10,−10);
endShape(CLOSE);
salmao();
pushMatrix();
//orelha esquerda
translate(5,0);
triangle(0,0,10,0,5,−5);
//orelha direita
translate(20,0);
triangle(0,0,10,0,5,−5);
popMatrix();
pushMatrix();
//olho esquerdo
translate(10,10);
fill(255);
ellipse(0,0,10,5);
fill(random(256));
ellipse(0,0,5,5);
//olho direito
translate(20,0);
fill(255);
```

```
ellipse(0, 0, 10, 5);
fill(random(256));
ellipse(0, 0, 5, 5);
popMatrix();
pushMatrix();
//nariz
translate(15, 20);
salmao();
triangle(0, 0, 10, 0, 5, 5);
//boca
translate(0, 17);
scale(1, −1);
jambo();
triangle(0, 0, 10, 0, 5, 5);
popMatrix();
pushMatrix();
//pelo esquerdo
translate(7, 20);
stroke(random(256));
line(0, 0, 8, 8);
line(0, 5, 8, 13);
//pelo direito
translate(25, 0);
scale(−1, 1);
line(0, 0, 8, 8);
line(0, 5, 8, 13);
noStroke();
popMatrix();
stroke(0);
//corpo do gato
translate(0, 30);
prata();
beginShape();
vertex(0, 0);
vertex(0, 50);
vertex(10, 50);
vertex(10, 40);
vertex(30, 40);
vertex(30, 50);
vertex(40, 50);
vertex(40, 0);
vertex(20, 20);
endShape(CLOSE);
//patas
amarelo();
translate(0, 30);
triangle(0, 0, 10, −10, 20, 0);
translate(20, 0);
triangle(0, 0, 10, −10, 20, 0);
}
void carro(){
azceu();
rotate(PI/2);
beginShape();
vertex(0, 0);
vertex(10, 10);
vertex(20, 10);
vertex(30, 20);
vertex(40, 20);
vertex(50, 10);
vertex(60, 10);
vertex(70, 0);
endShape(CLOSE);
translate(20, 10);
fill(random(256));
ellipse(7, 3, 5, 5);
cinzafosco();
quad(0, 0, 15, 0, 15, 10, 10, 10);
translate(30, 0);
scale(−1, 1);
quad(0, 0, 15, 0, 15, 10, 10, 10);
//rodas
fill(255);
translate(0, −10);
arc(0, 0, 10, 10, 0, PI);
translate(30, 0);
arc(0, 0, 10, 10, 0, PI);
}
9 void forma2(){
carmesim();
beginShape();
vertex(0, 0);
vertex(10, 10);
vertex(0, 20);
vertex(20, 40);
vertex(40, 20);
vertex(40, 10);
vertex(30, 0);
vertex(30, −10);
vertex(20, −20);
endShape(CLOSE);
azvio();
translate(10, 0);
triangle(0, 0, 10, 0, 10, 10);
translate(10, 10);
amarelo();
rect(0, 0, 10, 10);
translate(−10, 10);
azvio();
triangle(0, 0, 10, 0, 10, 10);
}
10 void forma3(){
lima();
scale(1, −1);
beginShape();
vertex(0, 0);
vertex(10, 10);
vertex(10, 20);
```

```
vertex(20,20);
vertex(30,30);
vertex(40,30);
vertex(50,20);
vertex(60,30);
vertex(50,40);
vertex(70,40);
vertex(40,70);
vertex(20,50);
vertex(30,40);
vertex(20,30);
vertex(10,40);
vertex(0,30);
endShape(CLOSE);
}
11 void forma4(){
jambo();
beginShape();
vertex(0,0);
vertex(10,10);
vertex(20,0);
vertex(30,10);
vertex(20,20);
vertex(40,40);
vertex(40,70);
vertex(10,70);
vertex(10,50);
vertex(0,50);
vertex(0,30);
vertex(10,20);
vertex(0,20);
endShape(CLOSE);
translate(10,40);
peru();
quad(0,0,10,−10,20,0,10,10);
translate(10,20);
scale(1,−1);
ouro();
triangle(0,0,10,0,10,10);
}
12 void cubo(){
quad(0,0,20,−20,40,0,20,20);
quad(0,0,0,10,20,30,20,20);
translate(40,0);
scale(−1,1);
quad(0,0,0,10,20,30,20,20);
}
13 void dent(){
beginShape();
vertex(0,0);
vertex(10,0);
vertex(15,−5);
vertex(20,0);
vertex(30,−10);
vertex(30,−30);
vertex(50,−30);
vertex(50,0);
vertex(30,10);
vertex(20,10);
endShape(CLOSE);
}
```

b) Construção da função “mosaico” na respectiva aba:

```
1 void mosaico(){
fill(189,143,143);
rect(0,0,150,150);
trevo();
pushMatrix();
translate(30,20);
pent();
translate(0,−20);
pm();
translate(10,10);
y();
translate(−30,30);
forma2();
popMatrix();
pushMatrix();
translate(110,0);
a();
popMatrix();
pushMatrix();
translate(0,30);
forma1();
translate(−30,20);
forma3();
translate(0,30);
forma4();
translate(20,10);
cubo();
translate(−30,0);
pent();
popMatrix();
pushMatrix();
translate(40,60);
peixe();
popMatrix();
pushMatrix();
translate(110,40);
```

```
carro();
popMatrix();
pushMatrix();
translate(110, 40);
gato();
translate(−40, 40);
dent();
popMatrix();
}
```

c) Construção da função *void telamosaico* na aba “tela”:

```
void telamosaico(){
rotate(a);
//Mosaico principal
mosaico();
pushMatrix();
//Simetria no eixo 0x;
scale(−1, 1);
mosaico();
popMatrix();
pushMatrix();
//Simetria no eixo 0y;
scale(1, −1);
mosaico();
popMatrix();
pushMatrix();
//Simetria em relação à origem (350,350);
scale(−1);
mosaico();
popMatrix();
//Incremento para rotação do conjunto.
a = a + 0.001;
}
```

d) Construção do caleidoscópio com implementação da função *telamosaico* no interior da função *void draw*:

```
void draw(){
background(255);
//Origem no centro da tela (350, 350);
translate(width/2, height/2);
rotate(b);
telamosaico();
pushMatrix();
//Origem no ponto (350 + 300, 350);
translate(300, 0);
telamosaico();
popMatrix();
pushMatrix();
//Origem no ponto (350 − 300, 350);
translate(−300, 0);
telamosaico();
popMatrix();
pushMatrix();
//Origem no ponto (350 + 350, 350 + 350);
translate(300, 300);
telamosaico();
popMatrix();
pushMatrix();
//Origem no ponto (350 − 300, 350 + 300);
translate(−300, 300);
telamosaico();
popMatrix();
pushMatrix();
//Origem no ponto (350 + 300, 350 − 300);
translate(300, −300);
telamosaico();
popMatrix();
pushMatrix();
//Origem no ponto (350, 350 + 300);
translate(0, 300);
telamosaico();
popMatrix();
pushMatrix();
//Origem no ponto (350, 350 − 300);
```

$translate(0, -300);$

$telamosaico();$

$popMatrix();$

$pushMatrix();$

//Origem no ponto $(350 - 300, 350 - 300);$

$translate(-300, -300);$

$telamosaico();$

$popMatrix();$

//Incremento para rotação da tela do mosaico;

$b = b + 0.004;$

}

Com isso, o mosaico que servirá de suporte para nosso caleidoscópio está completo e pronto para iniciar seus movimentos.