

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

TOWELJS: ENGINE 3D EM JAVASCRIPT USANDO
ARQUITETURA BASEADA EM COMPONENTES

GABRIEL ZANLUCA

BLUMENAU
2018

GABRIEL ZANLUCA

**TOWELJS: ENGINE 3D EM JAVASCRIPT USANDO
ARQUITETURA BASEADA EM COMPONENTES**

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Dalton Solano dos Reis, M. Sc. - Orientador

**BLUMENAU
2018**

TOWELJS: ENGINE 3D EM JAVASCRIPT USANDO ARQUITETURA BASEADA EM COMPONENTES

Por

GABRIEL ZANLUCA

Trabalho de Conclusão de Curso aprovado
para obtenção dos créditos na disciplina de
Trabalho de Conclusão de Curso II pela banca
examinadora formada por:

Presidente:

Prof. Dalton Solano dos Reis, M. Sc. – Orientador, FURB

Membro:

Prof(a). Luciana Pereira de Araújo, Mestre – FURB

Membro:

Prof(a). Nome do(a) Professor(a), Titulação – FURB

Blumenau, dia de julho de 2018

Dedico este trabalho ... [Geralmente um texto pouco extenso, onde o autor homenageia ou dedica o trabalho a alguém. Colocar a partir do meio da página.]

AGRADECIMENTOS

A Deus...


À minha família...

Aos meus amigos...

Ao meu orientador...


[Colocar menções a quem tenha contribuído, de alguma forma, para a realização do trabalho.]



[Epígrafe: frase que o estudante considera significativa para sua vida ou para  contexto do trabalho. Colocar a partir do meio da página.]

[Autor da Epígrafe]

RESUMO

O resumo é uma apresentação concisa dos pontos relevantes de um texto. Informa suficientemente ao leitor, para que este possa decidir sobre a conveniência da leitura do texto inteiro. Deve conter OBRIGATORIAMENTE  **OBJETIVO, METODOLOGIA, RESULTADOS e CONCLUSÕES**. O resumo deve conter de 150 a 500 palavras e deve ser composto de uma sequência corrente de frases concisas e não de uma enumeração de tópicos. O resumo deve ser escrito em um único texto corrido (sem parágrafos). Deve-se usar a terceira pessoa do singular e verbo na voz ativa (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2003).

Palavras-chave: Ciência da computação. Monografia. Resumo. Formato.

[Palavras-chave são separadas por ponto, com a primeira letra maiúscula. Caso uma palavra-chave seja composta por mais de uma palavra, somente a primeira deve ser escrita com letra maiúscula, sendo que as demais iniciam com letra minúscula, desde que não sejam nomes próprios.]

ABSTRACT




Abstract é o resumo traduzido para o inglês. *Abstract* vem em uma nova folha, logo após o resumo. Escrever com letra normal (sem itálico).

Key-words: Computer science. Monograph. Abstract. Format.

[*Key-words* são separadas por ponto, com a primeira letra maiúscula. Caso uma *key-word* seja composta por mais de uma palavra, somente a primeira deve ser escrita com letra maiúscula, sendo que as demais iniciam com letra minúscula, desde que não sejam nomes próprios.]


LISTA DE FIGURAS

Figura 1 – Exemplo do uso Three.js (geometries).....	17
Figura 2 - Uso do WebGLStudio.js	18
 Figura 3 - Informações na aba Profiling	19
Figura 4 - Arquitetura VisEdu-Engine	20
Figura 5 – Utilização do motor de jogos VisEdu-Engine.....	21

LISTA DE QUADROS

Quadro 1 - Construtor do objeto de jogo	24
Quadro 2 - Código da Classe CubeGameObject.js	25
Quadro 3 - Construtor padrão da luz	25
Quadro 4 - Construtor da luz direcional	25
Quadro 5 - Construtor do ponto de luz	25
Quadro 6 - Construtor do spotlight.....	26
Quadro 7 - Código que adiciona um objeto de jogo a cena.....	26
Quadro 8 - Código do construtor da câmera em perspectiva	27
Quadro 9 - Código do construtor da câmera ortogonal	27
Quadro 10 - Código da classe RenderSystem	28
Quadro 11 - Código da classe do componente padrão	29
Quadro 12 - Código do componente de translação.....	31
Quadro 13 - Código utilizando o motor de jogos	33

LISTA DE TABELAS

Tabela 1 – Trabalhos finais realizados no Curso de Ciência da Computação  **Error! Bookmark not defined.**

LISTA DE ABREVIATURAS E SIGLAS

[Deve conter as abreviaturas e siglas utilizadas mais de uma vez ao longo do texto em ordem alfabética. A seguir estão dois exemplos de forma de apresentação.]

ABNT – Associação Brasileira de Normas Técnicas

API – Application Programming Interface



SUMÁRIO

1 INTRODUÇÃO.....	13
1.1 OBJETIVOS.....	14
1.2 ESTRUTURA.....	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 MOTOR DE JOGOS	15
2.2 ARQUITETURA BASEADA EM COMPONENTES	15
2.3 TRABALHO CORRELATOS	16
2.3.1 Three.JS.....	16
2.3.2 WebGLStudio.js.....	17
2.3.3 VisEdu-Engine	19
3 DESENVOLVIMENTO DO MOTOR DE JOGOS	22
3.1 REQUISITOS.....	22
3.2 ESPECIFICAÇÃO	22
3.2.1 Arquitetura do motor de jogos	22
3.3 IMPLEMENTAÇÃO	23
3.3.1 Técnicas e ferramentas utilizadas.....	23
3.3.2 O motor de jogos	24
3.3.3 Operacionalidade da implementação	32
3.4 ANÁLISE DOS RESULTADOS	33
4 CONCLUSÕES.....	34
4.1 EXTENSÕES	34
REFERÊNCIAS	35
APÊNDICE A – RELAÇÃO DOS FORMATOS DAS APRESENTAÇÕES DOS TRABALHOS	38
ANEXO A – REPRESENTAÇÃO GRÁFICA DE CONTAGEM DE CITAÇÕES DE AUTORES POR SEMESTRE NOS TRABALHOS DE CONCLUSÕES REALIZADOS NO CURSO DE CIÊNCIA DA COMPUTAÇÃO.....	39

1 INTRODUÇÃO

De acordo com a pesquisa realizada pela Newzoo (2017), em 2017 a previsão é que a indústria de jogos eletrônicos movimentará 108,9 bilhões de dólares. Isso representa um ganho de 7,8 bilhões comparado com o ano de 2016. Dessa forma, a indústria de jogos eletrônicos se demonstra como algo atrativo para se investir em desenvolvimento visto que as projeções, segundo a mesma pesquisa, para os próximos anos também indicam aumento.

Para que todo esse crescimento se concretize a busca e utilização de ferramentas adequadas para auxiliar no desenvolvimento se torna essencial, dentre elas pode-se mencionar os motores de jogos. Com a utilização de motores de jogos ocorre-se um ganho de tempo e um aumento no nível de abstração. O ganho de tempo acontece pelo fato de rotinas, comportamentos e algoritmos gerais voltados para o desenvolvimento de jogos já estarem implementados e testados, assim não necessitando refazê-los a cada nova aplicação. O maior grau de abstração ocorre porque vários comportamentos são encapsulados, a exemplo tem-se desenho de objetos gráficos, transformações geométricas, tratamento para entrada de periféricos, entre outros. Em todos os casos citados anteriormente o programador não necessitaria conhecer qual API gráfica está sendo usada para desenhar os objetos, como funciona as matrizes de transformação geométricas ou a forma como o clique do mouse é capturado e tratado. Com essas facilidades citadas anteriormente a tarefa do programador pode focar-se na criação da história e regras da aplicação deixando preocupação gerais e iguais em todas as aplicações para o motor de jogos escolhido resolver.

Ainda pensando-se na parte do desenvolvimento, uma das opções a se levar em consideração é o uso de uma arquitetura baseada em componentes, que se mostra interessante pelo fato de ser “[...] caracterizado pela composição de partes já existentes, ou pela composição de partes desenvolvidas independentemente e que são integradas para atingir o objetivo final [...]” (FEIJÓ, 2007, p. 17). Os benefícios de uso de componentes se dão pelo fato de questões relacionadas a desenho de objetos poderem ser criados separadamente, poder criar um comportamento que mais tarde será usado por uma ou mais personagens e em ambos os casos se tem a opção de remover quando necessário. Por exemplo, um comportamento de pulo pode ser implementado como um componente de pulo e sempre adicionando quando as personagens necessitarem dele. Dessa forma o componente pode ser visto como um bloco de montar que pode ser encaixado de diferentes formas.

Visto todos os argumentos citados anteriormente, o trabalho **proposto visa desenvolver** um motor de jogos que auxilie o desenvolvimento de jogos em 3D utilizando a linguagem

JavaScripts e arquitetura baseada em componentes. Com um foco mais didático não tento como requisito o melhor desempenho, mas sim algo que também auxilie alguém a aprender computação gráfica.

1.1 OBJETIVOS

O objetivo é desenvolver um motor de jogos 3D utilizando arquitetura baseada em componentes para facilitar o desenvolvimento de jogos em JavaScript.



Os objetivos específicos são:

- a) desenvolver componentes dedicados para análise da performance;
- b) **Analisar** a performance do motor desenvolvido comparado com outro motor disponível para uso.

1.2 ESTRUTURA



[Referir-se aos tópicos principais do texto, dando o roteiro ou ordem de exposição.]



2 FUNDAMENTAÇÃO TEÓRICA



2.1 MOTOR DE JOGOS

Com o passar do tempo o termo motor de jogos foi mudando o seu significado para abranger mais funcionalidades deixando de ser algo concentrado apenas na parte gráfica de uma aplicação. Segundo Eberly (2006, tradução nossa) “motor do jogo passou a significar uma coleção de motores - para gráficos, física, IA, redes, *scripts* [...]”. Dentre exemplos populares no mercado que atende essa descrição pode-se citar a Unity (UNITY, 2017), Unreal Engine (UNREAL ENGINE, 2017) e Cryengine (CRYENGINE, 2017) entre outras que não apenas fornecem auxílio na parte gráfica de aplicação, mas como já citado ajudam em outras partes da aplicação como simulação de física, criação de IA para as personagens não jogáveis e com tudo mais que se seja necessário atualmente para se construir uma aplicação gráfica. Assim um motor de jogos passa a ser algo bem maior e mais complexo atualmente tentando alcançar todas essas características.

A principal ideia de um motor de jogos “[...] é permitir que os recursos comuns a quase todos os jogos sejam reutilizados para cada novo jogo criado. Neste caso, a cada novo jogo, se implementa apenas seus requisitos particulares” (PESSOA, 2001). Dessa forma ganha-se em tempo, no desenvolvimento, por não necessitar a todo momento construir novamente as estruturas já usadas anteriormente, assim tendo a opção de se manter o foco na jogabilidade.

Logo juntando os fatos que foram apresentados sobre motores de jogos mostra-se que seus recursos vão além do auxílio no desenho de objetos e implementações de algoritmos de computação gráfica, são apresentados como ferramentas poderosas na construção de jogos eletrônicos, pois não ficam restrito no auxílio de uma área só.

2.2 ARQUITETURA BASEADA EM COMPONENTES

A arquitetura baseada em componentes baseia-se no desenvolvimento usando componentes. Segundo Sametinger (1997, p.2, tradução nossa) “componentes são artefatos que nós claramente identificamos em nossos sistemas de software. Eles têm uma interface, encapsulamento interno detalhado e são documentados separadamente.”

Ao adotar-se o uso de componentes combinando os fabricados com os adquiridos ocorre um aumento na qualidade e agiliza o desenvolvimento levando assim a uma entrega mais rápida (SZYPERSKI, 2002). Assim segundo Feijó (2001 apud CHEESMAN, 2007) o desenvolvimento de sistemas baseado em componentes adere ao princípio da divisão e conquista, diminuindo a complexidade, pois um problema é dividido em partes menores,

resolvendo estas partes menores e construindo soluções mais elaboradas a partir de soluções mais simples.

Um ponto que deve ficar bem claro é “[...] a diferença entre o desenvolvimento de componentes e desenvolvimento com componentes. No primeiro caso, os componentes são especificados e implementados, existindo a preocupação em gerar a documentação em projetá-los de forma a serem reusados. No segundo, os componentes já produzidos são utilizados para se conceber um novo sistema” (FEIJÓ, 2007).

2.3 TRABALHO CORRELATOS

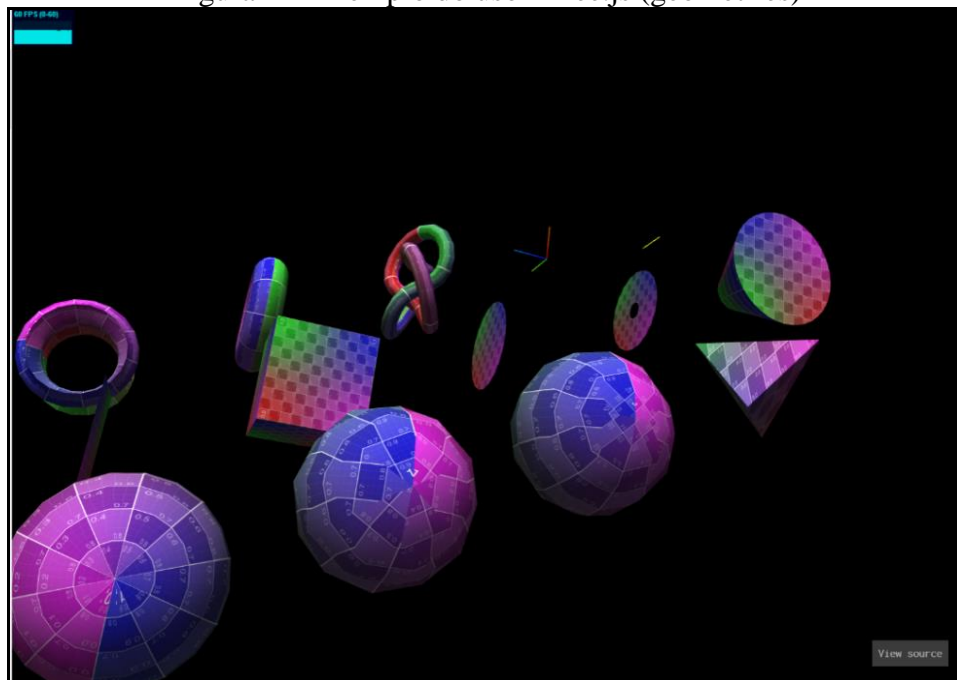
Nesta seção são apresentados trabalhos com características semelhantes aos principais objetivos do estudo proposto. O primeiro é a biblioteca Three.js (THREE.JS, 2017a), segundo é o editor WebGLStudio.js (WEBGLSTUDIOJS, 2017) e o terceiro é o motor de jogos VisEdu-Engine (HARBS, 2013).

2.3.1 Three.JS

O objetivo da Three.js é “[...] criar uma biblioteca 3D leve e fácil de usar. A biblioteca fornece renderizadores <canvas>, <svg>, CSS3D e WebGL” (THREE.JS, 2017a, tradução nossa). Desenvolvida em JavaScript e podendo ser vista como uma camada acima se comparado com uso do WebGL somente, porque permite um nível maior de abstração na questão gráfica do desenvolvimento da aplicação. Referente as questões para utilização ela é gratuita e possui código fonte aberto disponível para alteração.

Dentre as características descritas na documentação da Three.js (2017d) pode-se listar a presença de formas básicas já definidas tais como: cubos, quadrados, círculos, cilindros, esferas, linhas, etc. Também já conta com elementos de luz, diferentes tipos de materiais além de permitir a adição de textura nos objetos e permite o carregamento de objetos externos produzidos em programas de terceiros. Na Figura 1 tem-se um exemplo do uso da biblioteca com algumas das formas geométricas disponíveis e ainda com a utilização de iluminação.

Figura 1 – Exemplo do uso Three.js (geometries)



Fonte: Three.js (2017b).

A biblioteca conta também com recursos para ajudar na criação de animação de objetos. Segundo Three.js (2017c, tradução nossa) “[...] você pode animar várias propriedades de seus modelos: *bones* de um modelo *skinned and rigged*, *morph targets*, propriedades diferentes do material (cores, opacidade, valores lógicos), visibilidade e transformações”. Além dessa facilidade na hora de criar-se animações questões mais simples como transformações geométricas também contam com algum auxílio.

Sobre questões envolvendo simulação de física a Three.js não oferece nada para que possa auxiliar nessa tarefa. Caso seja necessário o desenvolvedor precisará buscar alguma biblioteca de terceiros para auxiliar nessa tarefa, assim poderá adicionar o comportamento como o da gravidade e colisão entre dois corpos.

2.3.2 WebGLStudio.js

O WebGLStudio.js é um editor de gráficos 3D que pode ser acessado pelo navegador, que suporte o mesmo, para se criar os projetos. O editor possui seu código aberto e usa internamente como biblioteca gráfica a LiteScene (LITESCENE, 2017). Tanto o WebGLStudio.js quanto a LiteScene são implementados em JavaScript.

As características a serem ressaltadas do WebGLStudio.js são:

- motor de gráficos 3D completo (LiteScene.js) que suporta múltiplas luzes, shadowmaps, reflexões em tempo real, materiais personalizados, postFX, skinning, animação e muito mais;

- sistema baseado em componentes fácil de expandir (controlando a linha de renderização ou encaixar eventos para de interação);

editor WYSIWYG fácil de usar, com todos os recursos em um só lugar (codificação, composição gráfica e linha de tempo);

editor de gráficos para criar comportamentos interessantes ou efeitos de pós-produção;

sistema virtual de arquivos para armazenar todos os recursos na web LiteFileSystem.js apenas arrastando-os (com cotas, usuários e pastas compartilhadas);

fácil de exportar e compartilhar, apenas enviando um link (WEBGLSTUDIOJS, 2017, tradução nossa).

Na Figura 2 pode se observar o uso do WebGLStudio.js com uma cena já carregada com um objeto 3D modelado e a presença de luz em cena. A direita pode-se visualizar o grafo de cena montado de forma visual. Além disso, também há campos para configurar as propriedades do objeto selecionado.

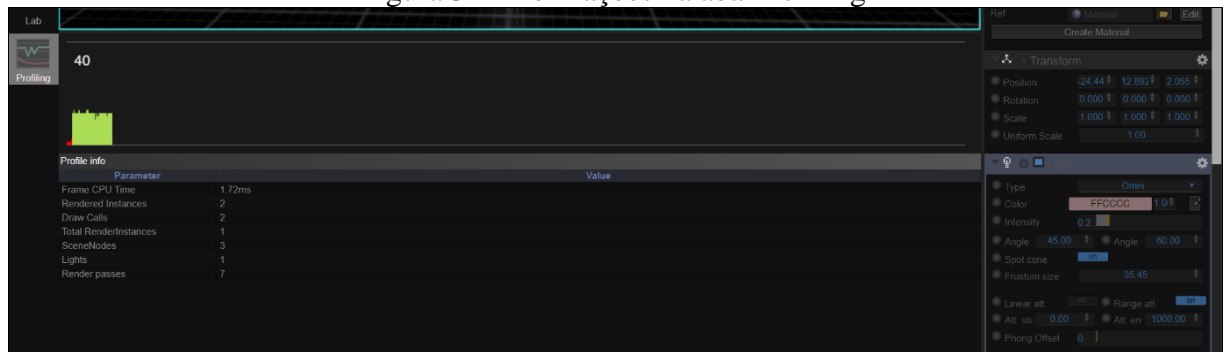
Figura 2 - Uso do WebGLStudio.js



Fonte: WebGLStudio.js (2017).

Semelhante à Three.js, o WebGLStudio.js não conta com recursos para tratar questões relacionadas a física, porém conta com uma aba no editor chamada *Profiling* que pode ser vista na **Figura 3 no qual** são exibidas algumas informações relativas a cena atual. Também conta com questões para criar as animações de personagens, podendo até se fazer uso da interface gráfica para configurar alguns dos parâmetros.

Figura 3 - Informações na aba Profiling



Fonte: WebGLStudio.js (2017).

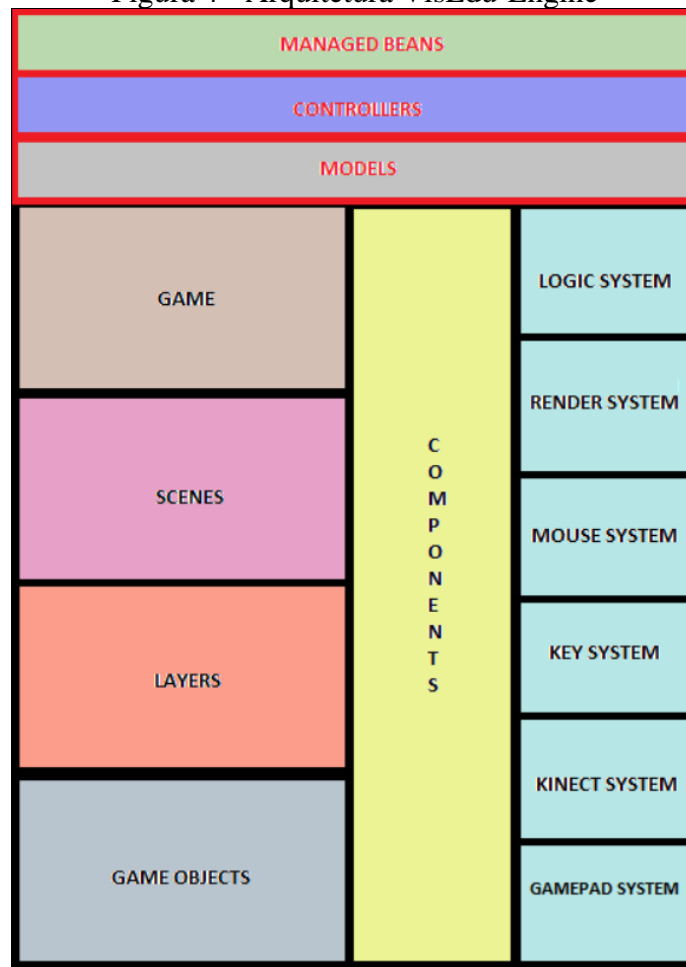
2.3.3 VisEdu-Engine

O VisEdu-Engine é um motor de jogos 2D que possui as seguintes funcionalidades “[...] criação de cena, criação de camadas, gerenciamento de objetos, gerenciamento de recurso de imagens e áudio, detecção de colisão, física de corpos rígidos (utilizando a biblioteca Box2DJS) e uma arquitetura reutilizável orientada a componentes” (HARBS, 2013). Desenvolvido em JavaScript e conta com um **editor que segundo** Harbs (2013) tem como objetivo facilitar o uso do motor de jogos para quem não tivesse um conhecimento tão aprofundado em programação.



Segundo Harbs (2013), a arquitetura do motor de jogos foi especificada de maneira orientada a componentes. Na Figura 4 pode-se observar o diagrama da arquitetura do VisEdu-Engine onde a comunicação das camadas **Game**, **Scenes**, **Layers** e **Game Objects**, ocorre no sentido de cima para baixo, ou seja, o **Game** se comunica com **Scenes** que se comunica com **Layers** que por sua vez se comunica com **Game Objects**. Já todas as camadas com *system* no nome têm como objetivo criar interface com dispositivos externos e essas por sua vez se comunicam com as camadas citadas anteriormente por meio de componentes. Como também pode ser observado além do motor ter suporte a periféricos comuns como *mouse* e teclado, há também suporte para o Kinect e *joystick*.

Figura 4 - Arquitetura VisEdu-Engine



Fonte: Harbs (2013).

Segundo o trabalho desenvolvido por Harbs (2013) os objetos disponíveis no VisEdu-Engine são quadrados, círculos e polígonos, todos possuem seu respectivo corpo rígido também para que possa ocorrer o tratamento de colisões. Em questão de organização, a aplicação consiste em criar uma cena com várias camadas (*layers*), para que assim tenha-se a impressão de algo estar no fundo ou mais à frente conseguindo assim simular o efeito de profundidade em cena. Na Figura 5 pode-se observar a criação de uma aplicação inspirada no jogo Space Invaders utilizando o motor de jogos.

Figura 5 – Utilização do motor de jogos VisEdu-Engine



Fonte: Harbs (2013).

3 DESENVOLVIMENTO DO MOTOR DE JOGOS

Neste capítulo são demonstradas as etapas do desenvolvimento do motor de jogos denominado TowelJS. Na seção 3.1 são apresentados os requisitos funcionais e não funcionais do motor de jogos. A seção 3.2 demonstra a especificação do motor de jogos. A seção 3.3 detalha a implementação do motor de jogos, descrevendo e exibindo os trechos relevantes de códigos e ferramentas utilizadas. Por fim, a seção 3.4 apresenta os resultados dos testes.

3.1 REQUISITOS

O motor de jogos proposto nesse trabalho deverá:

- a) permitir a criação/renderização de objetos gráficos (Requisito Funcional - RF);
- b) possuir objetos gráficos como cubo e esferas já implementados (RF);
- c) permitir a criação ou extensão de objetos gráficos já criados (RF);
- d) permitir a criação de novos componentes (RF);
- e) implementar a estrutura de grafo de cena (RF);
- f) possuir componentes para analisar a performance e consumo de recursos (RF);
- g) contar com ao menos uma opção de câmera sintética (RF);
- h) possuir diferentes tipos de iluminação disponível para colocar-se em cena (RF);
- i) ser implementado utilizando JavaScript (Requisito Não Funcional - RNF);
- j) contar com componentes próprios para fazer análise da performance (RNF);
- k) ser implementado utilizando a arquitetura baseada em componentes (RNF).



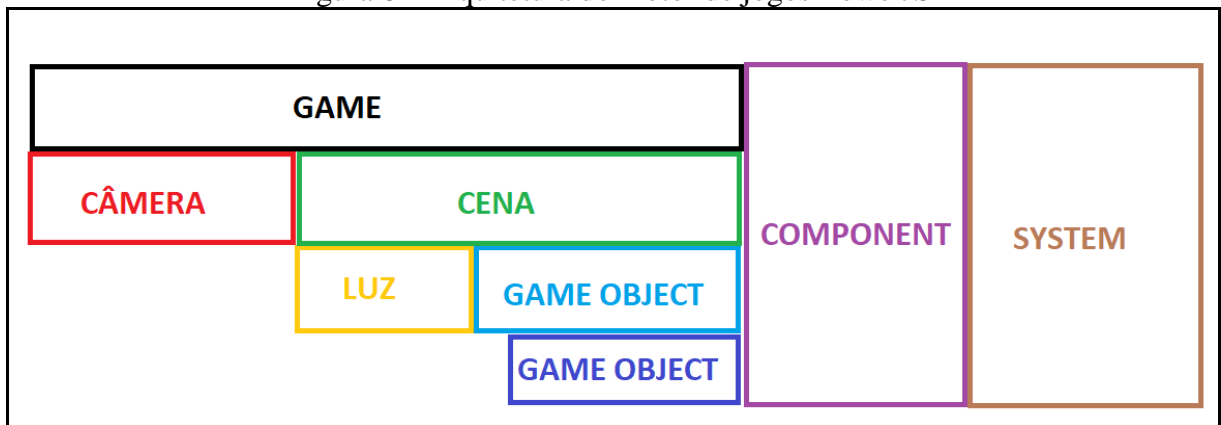
3.2 ESPECIFICAÇÃO

Nesta seção é descrita a especificação do motor de jogos TowelJS. Inicialmente é apresentada a arquitetura do motor de jogos em seguida são apresentados os diagramas ... desenvolvidos com a ferramenta Astah UML.

3.2.1 Arquitetura do motor de jogos

A Figura 6 exibe a arquitetura na qual o motor de jogos foi construído.

Figura 6 - Arquitetura do motor de jogos TowelJS



Fonte: elaborado pelo autor.

A arquitetura do motor de jogos é feita de maneira orientada a componentes e baseada na arquitetura utilizada por Harbs (2013, p39) em seu trabalho. A camada `GAME` é onde todas as outras se unem e por sua vez se comunica diretamente com as camadas `CÂMERA` e `CENA`. A camada `CENA` interage com as camadas `LUZ` e `GAME OBJECT`, que também pode se comunicar com uma outra camada de `GAME OBJECT` para que se possa ter a estrutura de grafo de cena. As camadas com o nome `SYSTEM` são as responsáveis por tratar do recebimento de eventos externos e comunica-los a camada `COMPONENT`, onde recebe as informações para que sejam tratadas.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento do motor de jogos utilizou-se JavaScript e como API gráfica o WebGL. Como ambiente de desenvolvimento optou-se pelo editor de texto Visual Studio Code (REF). Utilizou-se também o gerenciador de pacotes npm (REF) para controlar as dependências do motor de jogos de código externo, como também para poder publicar o motor de jogos para que outros desenvolvedores pudessem fazer uso dele. Para abstrair as operações que são feitas sobre matrizes optou-se pela biblioteca gl-matrix (REF).

Como a versão do JavaScript utilizada para desenvolver o motor de jogos é mais recente do que a implementada nos navegadores precisou-se utilizar um *transpiler* para que o código fosse convertido para uma versão que a maioria dos navegadores do mercado suporte. Visto esse fato o *transpiler* escolhido foi o Webpack (REF). Para os testes foram escolhidos os navegadores Google Chrome, Opera e FireFox. Todos testes foram executados num

computador notebook com processador Intel(R) Core(TM) i7-6700HQ de 2.60GHz e 8 núcleos, memória de 16 GB DDR3 e uma placa de vídeo Nvidia GeForce GTX 960M.

3.3.2 O motor de jogos

Nessa seção serão explicados os principais elementos do motor de jogos. Eles serão explicados na ordem inversa da arquitetura, começando pelo **objeto de jogo** e terminando com o **game**.

3.3.2.1 Objeto de jogo

O **objeto de jogo** é o responsável pela parte lógica dos objetos gráficos do motor de jogos. Nele ficam armazenadas informações essenciais para principalmente ser usada no desenho de cada objeto em **cena**, **dentre** elas pode-se destacar a matriz de transformação, cor, filhos e a lista de componentes. No Quadro 1 pode-se visualizar o código do construtor da classe do objeto de jogo padrão.

Quadro 1 - Construtor do objeto de jogo

```
constructor({ oringin = new Point3D(0, 0, 0), color = new Color(), }) {
  this.__id = JSUtils.generateUUID();
  this.__matrix = mat4.create();
  this.__oringin = oringin;
  this.__color = color;
  this.__listComponents = new ComponentList();
  this.__listComponents.addComponent(new TranslateComponent({ owner:
this }));
  this.__listComponents.addComponent(new RotateComponent({ owner: this
}));
  this.__listComponents.addComponent(new ScaleComponent({ owner: this
}));
  this.translate.translation = [oringin.x, oringin.y, oringin.z];
  this.__children = [];
}
```

Fonte: elaborado pelo autor.



Além dos atributos já citados há também o **id** gerado de forma aleatória buscando assim possuir um para cada **objeto de jogo** criado para que se possa recupera-lo por esse id caso seja necessário. Além disso no construtor já são criados os componentes responsáveis pelas transformações geométricas e assim dessa forma o componente de translação já é usado para colocar o objeto na posição que for passada como parâmetro, pois todos os objetos de jogo são criados por padrão com **x**, **y** e **z** zerado.

Por fim para que um **objeto de jogo** seja completo em cada classe que herde da classe **padrão precisa, que** no seu construtor adicione na lista de componentes o componente responsável por desenhar tal objeto. No Quadro 2 **encontra** o código pertencente a classe `CubeGameObject` **ela** é responsável por representar um cubo dentro do motor de jogos. Para

que tudo funcione de forma correta é necessário instanciar um objeto de `CubeRenderComponent`, logo após isso é necessário chamar o método `onLoad` dele e atribuir a cor a ele, dessa forma cubo será desenhado na cena de forma correta.

Quadro 2 - Código da Classe `CubeGameObject.js`

```
class CubeGameObject extends GameObject{
  constructor({point = new Point3D(0,0,0), color = new Color()}) {
    super({origin : point, color});
    this.__listComponents.addComponent(new CubeRenderComponent({owner
: this}));
    this.render.onLoad();
    this.render.color = color;
  }

  get render() {
    return this.listComponents[CubeRenderComponent.tag];
  }

  get tag(){
    return "CUBE_OBJECT";
  }
}
```

Fonte: elaborado pelo autor.

3.3.2.2 Luz

Semelhante com o objeto de jogo a luz é a representação lógica de iluminação dentro do motor de jogos. Foram implementados três tipos diferentes: luz direcional, ponto de luz e spotlight. No Quadro 3 pode ser observado o código do construtor padrão da luz.

Quadro 3 - Construtor padrão da luz

```
constructor({ color = new Color({ r: 0, b: 0, g: 0 }), position = new
Point3D(0, 0, 0) }) {
  this.__color = color;
  this.__position = position;
}
```

Fonte: elaborado pelo autor.

Para cada tipo diferente de luz seu construtor muda, nos Quadro 4, Quadro 5 e Quadro 6 pode ser visto os construtores da luz direcional, ponto de luz e spotlight respectivamente.

Quadro 4 - Construtor da luz direcional

```
constructor({color = new Color({r:0, b:0, g : 0}), position = new
Point3D(0,0,0) }) {
  super({color : color, position : position});
}
```

Fonte: elaborado pelo autor.

Quadro 5 - Construtor do ponto de luz

```
constructor({ color = new Color({ r: 1, b: 1, g: 1 }), position = new
Point3D(0, 0, 0), shininess = 1, secondColor = new Color({ r: 1, g: 1, b:
1 }) }) {
  super({ color: color, position: position});
  this.__shininess = shininess;
  this.__secondColor = secondColor
}
```

Fonte: elaborado pelo autor.

Quadro 6 - Construtor do spotlight

```

constructor({ position = new Point3D(0, 0, 0), color = new Color({ r: 1,
g: 1, b: 1 }), innerLimit = 0, outerLimit = 0, target = new Point3D(0, 0,
0) }) {
    super({ color: color, position: position });
    this.__outerLimit = (Math.PI / 180) * outerLimit;
    this.__innerLimit = (Math.PI / 180) * innerLimit;
    this.__target = target;
    this.__shininess = 1;
    this.__matrix = mat4.create();
    mat4.lookAt(this.__matrix, [position.x, position.y, position.z],
target.toVector(), [0, 1, 0]);
}

```

Fonte: elaborado pelo autor.

Para a **luz direcional** o construtor padrão já basta, agora para o **ponto de luz** há também atributos relacionados para a segunda cor que a luz emite no seu centro e a intensidade do brilho. Para o **spotlight** além dos atributos da classe padrão há também os atributos para controlar o degrade da luz entre o interior e o exterior e a posição para onde a luz deverá apontar.

3.3.2.3 Cena

A cena funciona como um agregador dos **objetos de jogo** junto com as luzes para que depois essas informações possam ser acessadas pelas funções responsáveis pelo desenho. No Quadro 7 pode-se observar o código responsável por adicionar um **objeto de jogo** na **cena** e nesse momento que todos os componentes adicionados nele executam o seu método `onLoad`.

Quadro 7 - Código que adiciona um objeto de jogo a cena

```

addGameObject(gameObject) {
    this.__gameObjectList.push(gameObject);
    for (let componentKey in gameObject.listComponents) {
        let component = gameObject.listComponents[componentKey];
        component.onLoad();
    }
}

```

Fonte: elaborado pelo autor.

Quando uma luz é adicionada ou removida da cena é necessário alterar o código do *vertex shader* presente nos componentes de desenho da cena, pois as informações da luz adicionada ou removida precisam ser alteradas.

3.3.2.4 Câmera

O motor de jogos oferece duas opções de câmera sintética, ortogonal e perspectiva, para criar-se a matriz de cada câmera fez uso da biblioteca `glMatrix`. Nos Quadro 8 e Quadro 9 apresenta-se os códigos dos construtores da câmera em perspectiva e ortogonal respectivamente.

Quadro 8 - Código do construtor da câmera em perspectiva

```

constructor({ aspect = 1, near = 0, far = 0, fovy = 0, position = new
Point3D(0, 0, 0) }) {
    this.__near = near;
    this.__far = far;
    this.__aspect = aspect;
    this.__fovy = fovy;
    this.__position = position;
    this.__projection = mat4.create();
    this.__matrix = mat4.create();
    mat4.perspective(this.__projection, fovy, aspect, near, far);
    mat4.lookAt(this.__matrix, [position.x, position.y, position.z], [0,
0, 0], [0, 1, 0]);
}

```

Fonte: elaborado pelo autor.

Quadro 9 - Código do construtor da câmera ortogonal

```

constructor({ left, right, bottom, top, near, far }) {
    this.__projection = mat4.create();
    mat4.ortho(this.__projection, left, right, bottom, top, near, far);
}

```

Fonte: elaborado pelo autor.

3.3.2.5 Classes System

As classes *system* são responsáveis por receber e propagar os eventos externos, como por exemplo o clique ou movimento do mouse, o pressionar de uma tecla, a chamada do sistema para desenhar a cena, entre outros. Para demonstração no Quadro 10 tem-se o código do *RenderSystem* responsável por propagar o evento de desenho de cada *frame*.

Cada propagação de evento executa uma função específica do componente e sempre segue essa ordem: começa pelo **game**, seguido pela **cena** e por fim os **objetos de jogo**. As classes **system** presentes atualmente no motor de jogos são: *RenderSystem*, *LogicSystem*, *MouseSystem* e *KeyboardSystem*. Responsáveis por propagar os eventos de desenhar a cada *frame*, atualização de cada *frame*, clique ou movimento do mouse e pressionar de uma tecla respectivamente.

Quadro 10 - Código da classe RenderSystem

```

class RenderSystem {
    static fireRenderListener(){
        let game = new Game();
        game.context.clearColor(1.0, 1.0, 1.0, 1.0);
        game.context.enable(game.context.CULL_FACE);
        game.context.clear( game.context.COLOR_BUFFER_BIT |
game.context.DEPTH_BUFFER_BIT);
        game.context.clearDepth(1.0);
        game.context.enable(game.context.DEPTH_TEST);
        game.context.depthFunc(game.context.LEQUAL);

        if (game.scene) {
            for (let gameObject of game.scene.gameObjectList) {
                if (gameObject instanceof GameObject){
                    for (let index in gameObject.listComponents) {
                        let component = gameObject.listComponents[index];
                        if (component instanceof Component){
                            component.onRender(game.context,
game.projection);
                        }
                    }
                }
            }
        }

        static get tag () {
            return "RENDER_SYSTEM";
        }
    }
}

```

Fonte: elaborado pelo autor.

3.3.2.6 Componentes

Os componentes são onde a arquitetura baseada em componentes realmente se faz presente. Cada componente herda da classe padrão `Component` e nela estão presentes os métodos que são responsáveis por tratar os eventos recebidos por classe `System`. Atualmente o **jogo**, a **cena** e os **objetos de jogo** podem possuir componentes. No Quadro 11 pode-se observar o código da classe `Component` padrão.

Quadro 11 - Código da classe do componente padrão

```

class Component {
    constructor({ owner }) {
        this.__id = JSUtils.generateUUID();
        this.__enabled = true;
        this.__owner = owner;
    }

    get id() {
        return this.__id;
    }

    get enabled() {
        return this.__enabled;
    }

    set enabled(enabled) {
        this.__enabled = enabled;
    }

    get owner() {
        return this.__owner;
    }

    set owner(owner) {
        this.__owner = owner;
    }

    onKeyDown(keyCode) { }

    onKeyUp(keyCode) { }

    onKeyPress(keyCode) { }

    onClick(x, y, wich) { }

    onMouseDown(x, y, wich) { }

    onMouseMove(x, y) { }

    onBeforeRender(context) { }

    onRender(context, projectionMareix) { }

    onUpdate(delta) { }

    onLoad() { }

    onDestroy() { }

    get tag() {
        return "COMPONENT";
    }
}

```



Fonte: elaborado pelo autor.

Dentro do motor de jogos dois tipos de componentes já são especificados e precisam estar presente para que ocorra o funcionamento correto, podendo ou não ser os implementados nesse trabalho. São eles os componentes responsáveis por desenhar os objetos

de jogo e os responsáveis por fazer as transformações geométricas. Para que o grau de abstração seja melhor criou-se uma camada a mais entre a classe `Component` e as classes específicas de desenho de cada `objeto de jogo`, para garantir que certos métodos sejam implementados e para evitar repetição desnecessária de código. Essa camada é a classe `RenderComponent` é de onde todas as outras classes responsáveis pelo desenho herdam.

As transformações geométricas também são efetuadas por meio de `componente` e igualmente o desenho dos `objetos de jogo` há uma opção já implementada. As transformações geométricas por usarem matrizes foram feitas com auxílio da biblioteca `gl-matrix`, porém se pensou em fazer da forma mais desacoplada possível para que fosse fácil realizar a troca da biblioteca ou do próprio componente em si. No Quadro 12 observa-se o código do componente de `translação os outros, rotação e escala`, seguem o mesmo princípio só mudando a chamada do método da biblioteca.

Por fazer-se uso da biblioteca `gl-matrix` as transformações ocorrem exatamente como foram implementadas nela que é de forma relativa. Dessa forma, toda transformação é a acumulação do estado anterior com o novo valor. Exemplo ao se fazer a translação de um objeto em `x` acrescentando 2, com `x` igual a 5, o resultado final será `x` igual a 7.

Quadro 12 - Código do componente de translação

```

class TranslateComponent extends Component{

    constructor({owner}) {
        super({owner : owner});
        this.__translation = vec3.create();
    }

    set translation(translation){
        vec3.set(this.__translation, translation[0], translation[1],
translation[2]);
        this.translate(this.owner.matrix);
    }

    get translation() {
        return this.__translation;
    }

    get x() {
        return this.__translation[0];
    }

    get y() {
        return this.__translation[1];
    }

    get z() {
        return this.__translation[2];
    }

    set x(x) {
        this.__translation[0] = x;
        mat4.translate(this.owner.matrix, this.owner.matrix, [x, 0, 0]);
    }

    set y(y) {
        this.__translation[1] = y;
        mat4.translate(this.owner.matrix, this.owner.matrix, [0, y, 0]);
    }

    set z(z) {
        this.__translation[2] = z;
        mat4.translate(this.owner.matrix, this.owner.matrix, [0, 0, z]);
    }

    translate(matrix){
        mat4.translate(matrix, matrix, this.__translation);
    }

    get tag(){
        return TranslateComponent.tag;
    }

    static get tag(){
        return "TRANSLATE_COMPONENT";
    }
}

```

Fonte: elaborado pelo autor.

3.3.2.7 Game

O `Game` é a estrutura principal do motor de jogos é nele que as classes `system` são cadastradas para tratar os eventos externo. Também é nele que a `câmera` e a `cena` juntam-se para haver o `desenho dos objetos`. Além disso o `Game` é responsável por dar início a função que irá desenhar a cena.

`Pode existir` apenas um objeto da classe `Game` por aplicação e para isso usou-se o padrão singleton.

3.3.3 Operacionalidade da implementação

Para apresentar o uso do motor de jogos escolheu-se um cenário bem simples, ele consiste de uma cena com um cubo e uma esfera sendo essa esfera filha do cubo. Nesse cenário há também a presença de uma luz e optou-se por usar a câmera em perspectiva.

A primeira coisa a se fazer para usar o motor de jogos é ter um elemento de `canvas` declarado na página HTML. Esse elemento é responsável por exibir a cena e usando ele é possível ter acesso ao uso da GPU. Com esse elemento criado deve-se acessa-lo pelo código e em seguida criar o contexto, como o motor de jogos desenvolvido trabalha em 3D os parâmetros possíveis para a função `getContext` são: `"webgl"`, `"experimental-webgl"` e `"webgl2"`.

Seguindo a explicação cria-se uma cena e após isso uma câmera, nesse caso em perspectiva, com isso pode-se criar uma `instancia` da classe `Game`. Em seguida cria-se três `instancia` da classe `Cor`, uma para cor vermelha, azul e branca, para serem usadas como cor dos objetos e luz. Para os objetos de jogo criou-se uma `instancia` da classe `CubeGameObject` e outra de `SphereGameObject` usando as cores criadas anteriormente e como não se passou nenhum parâmetro para indicar a posição ambos serão criados na posição `x`, `y` e `z` igual zero. Cria-se em seguida uma instancia da classe `DirectionalLight` para que haja incidência de luz em cena, mesmo que não seja necessário ter luz para que o objeto apareça.

Nesse exemplo aproveitou-se do próprio componente de rotação para que a cada `frame` seja atualizado a rotação do objeto. Por fim adiciona-se os objetos de jogos na cena usando o método `addGameObject` e para adicionar a esfera como filho do cubo no grafo de cena usa-se o mesmo método, porém chame-se ele através da `instancia` do cubo.

Quadro 13 - Código utilizando o motor de jogos

```

import { Scene } from "../game/Scene";
import { Point3D } from "../geometric/Point3D";
import { PerspectiveCamera } from "../game/PerspectiveCamera";
import { Game } from "../game/Game";
import { Color } from "../geometric/Color";
import { CubeGameObject } from "../gameObject/CubeGameObject";
import { PointLight } from "../Light/PointLight";
import { SphereGameObject } from "../gameObject/SphereGameObject";
import { DirectionalLight } from "../Light/DirectionalLight";

let canvas = document.getElementById("glCanvas");

let context = canvas.getContext("webgl2");

let scene = new Scene();
let camera = new PerspectiveCamera({near: 0.1, far : 500, aspect : 1, fovy
: 45 * Math.PI / 180, position : new Point3D(0, 0, 15)});

let game = new Game(context, scene, camera);

let red = new Color({r : 1});
let blue = new Color({b : 1});
let white = new Color({r : 1, g : 1, b : 1});

let cube = new CubeGameObject({color : red});
let sphere = new SphereGameObject({color : blue});

let directLight = new DirectionalLight({color : white, position : new
Point3D(2, 8, 5)});

scene.addLight(directLight);

cube.rotation.onUpdate = (deltaTime) => {
    cube.rotation.y = 0.9 * deltaTime;
}

cube.translate.z = -5;
sphere.translate.x = 3;

scene.addGameObject(cube);
cube.addGameOdbject(sphere);

```

Fonte: elaborado pelo autor.

3.4 ANÁLISE DOS RESULTADOS

[Apresentar os casos de testes do software, destacando objetivo do teste, como foi realizada a coleta de dados e a apresentação dos resultados obtidos, preferencialmente em forma de gráficos ou tabelas, fazendo comentários sobre os mesmos.

Confrontar com os trabalhos correlatos  apresentados na fundamentação teórica.]

4 CONCLUSÕES

[As conclusões devem refletir os principais resultados alcançados, realizando uma avaliação em relação aos objetivos previamente formulados. Deve-se deixar claro se os objetivos foram atendidos, se as ferramentas utilizadas foram adequadas e quais as principais contribuições do trabalho para o seu grupo de usuários ou para o desenvolvimento científico/tecnológico.]

[Deve-se também incluir aqui as principais vantagens do seu trabalho e limitações.]

4.1 EXTENSÕES

- a) Permitir a troca de componentes de transformações geométricas e de desenho de objetos em tempo de execução.
- b) Adicionar novos tipos de objetos.
- c) Possuir uma forma de selecionar objetos.
- d) Importar objetos feitos em software de terceiros como Blender ou May.
- e) Permitir aplicar texturas.
- f) Construir mais teste para análise se performance.

REFERÊNCIAS

[As referências deverão ser apresentadas em ordem alfabética. Só podem ser inseridas nas referências os documentos citados ao longo da monografia. Todos os documentos citados obrigatoriamente têm que estar inseridos nas referências.]

[No formato do nome do autor, após a chamada (sobrenome com todas as letras em caixa alta), o primeiro nome deverá ser apresentado por extenso com a primeira letra em maiúscula e demais em minúscula e os outros nomes abreviados (letra em maiúscula seguida de ponto).]

[Abaixo são mostrados alguns exemplos de referências bibliográficas.]

[livro em meio eletrônico:]

ALVES, Castro. **Navio negreiro**. [S.l.]: Virtual Books, 2000. Disponível em: <<http://www.terra.com.br/vistualbooks/freebook/port/Lport2/navionegreiro.htm>>. Acesso em: 10 jan. 2002.

[parte de um documento:]

AMADO, Gilles. Coesão organizacional e ilusão coletiva. In: MOTTA, Fernando C. P.; FREITAS, Maria E. (Org.). **Vida psíquica e organização**. Rio de Janeiro: FGV, 2000. p. 103-115.

[trabalho acadêmico ou monografia (TCC/Estágio, especialização, dissertação, tese):]

AMBONI, Narcisa F. **Estratégias organizacionais**: um estudo de multicasos em sistemas universitários federais das capitais da região sul do país. 1995. 143 f. Dissertação (Mestrado em Administração) - Curso de Pós-Graduação em Administração, Universidade Federal de Santa Catarina, Florianópolis.

[norma técnica:]

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**: informação e documentação: referências - elaboração. Rio de Janeiro, 2002a. 24 p.

_____. **NBR 6024**: informação e documentação: numeração progressiva das seções de um documento escrito - apresentação. Rio de Janeiro, 2012. 4 p.

_____. **NBR 6027**: informação e documentação: sumário - apresentação. Rio de Janeiro, 2013. 2 p.

_____. **NBR 6028**: resumos. Rio de Janeiro, 2003. 2 p.

_____. **NBR 10520**: informação e documentação: citações em documentos: apresentação. Rio de Janeiro, 2002b. 7 p.

_____. **NBR 14724**: informação e documentação: trabalhos acadêmicos: apresentação. Rio de Janeiro, 2011. 11 p.

[livro:]

BARRASS, Robert. **Os cientistas precisam escrever**: guia de redação para cientistas, engenheiros e estudantes. São Paulo: Ed. da Universidade de São Paulo, 1979.

BASTOS, Lília R.; PAIXÃO, Lyra; FERNANDES, Lúcia M. **Manual para a elaboração de projetos e relatórios de pesquisa, teses e dissertações**. Rio de Janeiro: Zahar, 1979.

[guias do usuário:]

BORLAND INTERNATIONAL INC. **Delphi user's guide**. Scotts Valley: Borland, 1995.

[help:]

BORLAND SOFTWARE CORPORATION. **Delphi enterprise**: help. Version 3.0. [S.l.], 1997. Documento eletrônico disponibilizado com o Ambiente Delphi 3.0.

[trabalho acadêmico ou monografia (TCC/Estágio, especialização, dissertação, tese):]

BRUXEL, Jorge L. **Definição de um interpretador para a linguagem Portugol, utilizando gramática de atributos**. 1996. 77 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

[verbete de enciclopédia em meio eletrônico:]

EDITORES gráficos. In: WIKIPEDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2006. Disponível em: <http://pt.wikipedia.org/wiki/Editores_graficos>. Acesso em: 13 maio 2006.

[artigo em evento:]

FRALEIGH, Arnold. The Algerian of independence. In: ANNUAL MEETING OF THE AMERICAN SOCIETY OF INTERNATIONAL LAW, 61, 1967, Washington. **Proceedings...** Washington: Society of International Law, 1967. p. 6-12.

[artigo em evento em meio eletrônico:]

GUNCHO, Mário R. A educação à distância e a biblioteca universitária. In: SEMINÁRIO DE BIBLIOTECAS UNIVERSITÁRIAS, 10, 1998, Fortaleza. **Anais...** Fortaleza: Tec Treina, 1998. 1 CD-ROM.

[norma técnica:]

IBGE. **Normas para apresentação tabular**. 3. ed. Rio de Janeiro, 1993. 61 p. Disponível em: <<http://biblioteca.ibge.gov.br/visualizacao/monografias/GEBIS%20-%20RJ/normastabular.pdf>>. Acesso em: 27 ago. 2013.

[artigo de periódico:]

KNUTH, Donald E. Semantic of context-free languages. **Mathematical Systems Theory**, New York, v. 2, n. 2, p. 33-50, Jan./Mar. 1968.

[parte de um documento:]

LAKATOS, Eva M. Cultura e poder organizacional e novas formas de gestão empresarial. In: _____. **Sociologia da administração**. São Paulo: Atlas, 1997. cap. 5, p. 122-143.

[artigo em periódico em meio eletrônico:]

MALOFF, Joel. A internet e o valor da "internetização". **Ciência da Informação**, Brasília, v. 26, n. 3, 1997. Disponível em: <<http://www.ibict.br/cionline/>>. Acesso em: 18 maio 1998.

[trabalho acadêmico ou monografia (TCC/Estágio, especialização, dissertação, tese):]

SCHIMT, Hédio. **Implementação de produto cartesiano e métodos de passagem de parâmetros no ambiente FURBOL**. 1999. 86 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SCHUBERT, Lucas A. **Aplicativo para controle de ferrovia utilizando processamento em tempo real e redes de Petri**. 2003. 76 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

[página da internet: se a página não for livro, artigo ou parte de documento em meio eletrônico, deve-se fazer a referência conforme o exemplo abaixo. (O ano da página abaixo descrita não existe explicitamente descrito. Ele foi obtido a partir de informações fornecidas pelo *browse* Mozilla, através da opção “*Page Info*” alcançado através da opção do menu “*View*”. Foi pego a data da última alteração (*modified*). Quando a data for indefinida, colocar uma provável, sendo que neste caso vai entre colchetes e logo após o ano existe o símbolo de interrogação “?” (ex.: ..., [2003?] . Disponível em: ...). Quando a data estiver explícita na página, colocar esta sem colchetes. Se o mês também estiver explícito, colocá-lo (ex.: ..., out. 2003. Disponível em: ...));]

SCHULER, João P. S. **Tutorial de Delphi**. Porto Alegre, [2002]. Disponível em: <<http://www.schulers.com/jpss/pascal/dtut/>>. Acesso em: 27 ago. 2013.

[artigo em evento:]

SILVA, José R. V. et al. Execução controlada de programas. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 1, 1987, Petrópolis. **Anais...** Petrópolis: UFRJ, 1987. p. 12-19.

[artigo em evento em meio eletrônico:]

SILVA, Roseane N.; OLIVEIRA, Ramon. Os limites pedagógicos do paradigma da qualidade total em educação. In: CONGRESSO DE INICIAÇÃO CIENTÍFICA DA UFPe, 4, 1996, Recife. **Anais eletrônicos...** Recife: UFPe, 1996. Disponível em: <<http://www.propesq.ufpe.br/anais/anais/educ/ce04..htm>>. Acesso em: 21 jan. 1997.

[livro:]

SEBESTA, Robert W. **Conceitos de linguagens de programação**. 4. ed. Porto Alegre: Bookman, 2000.

[parte de um documento em meio eletrônico:]

TEODOROWITSCH, Roland. **Manual de ética, estilo e português para a elaboração de trabalhos acadêmicos**. [Gravataí], 2003. Disponível em: <<http://www.ulbra.tche.br/~roland/pub/etica-est-port-2003-2.pdf>> . Acesso em: 28 mar. 2006.

[relatório de pesquisa:]

VARGAS, Douglas N. **Editor dirigido por sintaxe**. 1992. Relatório de pesquisa n. 240 arquivado na Pró-Reitoria de Pesquisa, Universidade Regional de Blumenau, Blumenau.


[artigo em periódico em meio eletrônico:]

VIEIRA, Cassio L.; LOPES, Marcelo. A queda do cometa. **Neo Interativa**, Rio de Janeiro, n. 2, inverno 1994. 1 CD-ROM.


WINDOWS 98: o melhor caminho para atualização. **PC World**, São Paulo, n. 75, set. 1998. Disponível em: <<http://www.idg.com.br/abre.html>>. Acesso em: 10 set. 1998.

APÊNDICE A – Relação dos formatos das apresentações dos trabalhos

[Elemento opcional. **Apêndices são textos elaborados pelo autor** a fim de complementar sua argumentação. Os apêndices são identificados por letras maiúsculas consecutivas, seguidas de um travessão e pelos respectivos títulos. Deverá haver no mínimo uma referência no texto anterior para cada apêndice.]

[Colocar sempre um preâmbulo no apêndice. Não colocar tabelas e ou ilustrações sem identificação no apêndice. Caso existirem entifique-as através da legenda, seguindo a numeração normal do volume final (para as legendas). Caso existirem tabelas e ou ilustrações, sempre referenciá-las antes.]

ANEXO A – Representação gráfica de contagem de citações de autores por semestre nos trabalhos de conclusões realizados no Curso de Ciência da Computação

[Elemento opcional. **Anexos são documentos não elaborados pelo autor**, que servem de fundamentação, comprovação ou ilustração, como mapas, leis, estatutos, entre outros. Os anexos são identificados por letras maiúsculas  consecutivas, seguidas de um travessão e pelos respectivos títulos. Deverá haver no mínimo uma referência no texto anterior para cada anexo.]

[Colocar sempre um preâmbulo no anexo. Não colocar tabelas e ou ilustrações sem identificação no anexo. Caso existirem, identifique-as através da legenda, seguindo a numeração normal do volume final (para as legendas). Caso existirem tabelas e ou ilustrações, sempre referenciá-las antes.]