UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

VISEDU-CG 3.0: APLICAÇÃO DIDÁTICA PARA VISUALIZAR MATERIAL EDUCACIONAL - MÓDULO DE COMPUTAÇÃO GRÁFICA

SAMUEL ANDERSON NUNES

BLUMENAU
2014

2014/1-22

SAMUEL ANDERSON NUNES

VISEDU-CG 3.0: APLICAÇÃO DIDÁTICA PARA VISUALIZAR MATERIAL EDUCACIONAL - MÓDULO DE COMPUTAÇÃO GRÁFICA

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Dalton Solano dos Reis, Mestre - Orientador

BLUMENAU
2014

2014/1-22

VISEDU-CG 3.0: APLICAÇÃO DIDÁTICA PARA VISUALIZAR MATERIAL EDUCACIONAL - MÓDULO DE COMPUTAÇÃO GRÁFICA

Por

SAMUEL ANDERSON NUNES

Trabalho 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, Mestre – Orientador, FURB
Membro:	Prof. Mauricio Capobianco Lopes, Doutor – FURB
Membro:	Prof. Aurélio Faustino Hoppe, Mestre – FURB

Blumenau, 8 de julho de 2014

Dedico este trabalho ao meu Deus por ter me dado a vida e a salvação. A minha família, por todo seu amor e seu apoio.

AGRADECIMENTOS

Ao meu Deus pelo seu amor tão grande e infinito, a ponto de dar seu filho para morrer em meu lugar.

A minha esposa Larissa, por todo seu amor, compreensão e apoio tornando este árduo trabalho em algo mais fácil.

A minha família, especialmente aos meus pais por todo incentivo e apoio que me deram em toda a minha vida.

Ao meu orientador pela confiança e por sua grande e essencial ajuda em todas as etapas deste trabalho.

Aos meus amigos que estiverem presentes durante toda a graduação, especialmente ao James Montibeler, Luiz Hogrefe e Marcelo da Mata.

A empresa Benner Sistemas por todo incentivo, experiência e aprendizado que tem me proporcionado nos últimos anos.

Aos alunos da disciplina de Computação Gráfica (2014-1) pela importante ajuda nos testes de operacionalidade deste trabalho.

Por que Deus amou o mundo tanto, que deu o seu único Filho, para que todo aquele que nele crer não morra, mas tenha a vida Eterna.

Jesus Cristo

RESUMO

Este trabalho tem como objetivo dar continuidade ao desenvolvimento de uma aplicação web para visualização de material educacional utilizando WebGL, disponibilizando uma ferramenta que permite explorar conceitos de computação gráfica 2D e 3D como o uso de iluminação de cena, os novos objetos gráficos Polígono e Spline e também a seleção dos objetos diretamente no espaço gráfico. Além disso, todas as funcionalidades existentes no espaço 3D são disponibilizadas também no espaço gráfico 2D. Utilizou-se a biblioteca Three.js para abstração da WebGL e a mesma se mostrou eficiente e de fácil utilização. A avaliação deste trabalho com relação a operacionalidade se deu com base em um teste de experiência de uso da ferramenta realizado pelos alunos da disciplina de Computação Gráfica. Com isso, foram obtidas sugestões dos alunos com relação a melhorias ao atual trabalho e, a partir da análise dos resultados do teste, é possível concluir que tratando-se do entendimento dos conceitos de computação gráfica e de operacionalidade, o ambiente atendeu as expectativas dos alunos.

Palavras-chave: Computação gráfica. Three.js. WebGL.

ABSTRACT

This work has as objective to give continuity to the development of a web application for viewing educational material using WebGL, providing a tool that allows you to explore concepts of 2D and 3D of computer graphics as the use of scene lighting, new graphical objects Polygon and Spline and also the selection of objects directly in the graph space. In addition, all existing features in 3D space are also available in 2D graph space. It was used the library Three.js for WebGL and abstraction of the same is efficient and easy to use. The evaluation of this work with respect to operability was made based on a test of experience using the tool held by the students of Computer Graphics. Thus, students' suggestions regarding improvements to the current study were obtained, and from the analysis of the test results, we conclude that it comes from the understanding of the concepts of computer graphics and operability, the environment has met the expectations of students.

Key-words: Computer graphics. Three.js. WebGL.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

2D – Duas Dimensões

3D – Três Dimensões

API – *Application Programming Interface*

FPS – Frames Por Segundo

FURB – Fundação Universidade Regional de Blumenau

HTML – *Hyper Text Markup Language*

JVM – *Java Virtual Machine*

MIT - *Massachusetts Institute of Technology*

OpenGL ES – *Open Graphics Library for Embedded Systems*

RF – Requisito Funcional

RNF – Requisito Não Funcional

SVG – *Scalabe Vector Graphics*

STEP – *Scheller Teacher Education Program*

SRO – Sistema de Referência do Objeto

SRU – Sistema de Referência do Universo

UC – *Use Case*

UML – *Unified Modeling Language*

VBO – *Vertex Buffer Object*

WebGL – *Web Graphics Library*

WHATWG – *Web Hypertext Application Technology Working Group*

W3C – *World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

A Internet é atualmente a mídia mais promissora desde a implantação da televisão, sendo a mídia mais aberta e descentralizada. É cada vez maior o número de pessoas ou grupos que criam na Internet suas próprias revistas, emissoras de rádio ou de televisão. A educação através da Internet também vem ocupando um lugar de destaque e, como consequência, pode modificar significativamente a educação presencial. Com isso, as paredes das escolas e das universidades se abrem, as pessoas se intercomunicam, trocam informações, dados e pesquisas. A educação continuada é otimizada pela possibilidade de integração entre várias mídias, acessando-as tanto em tempo real, como no horário favorável a cada indivíduo e também pela facilidade de contato entre os educadores e os educandos (MORAN, 1997).

Com toda a evolução da Internet, os navegadores web talvez sejam o tipo de aplicativo mais utilizado na história. Eles têm evoluído de forma significativa ao longo dos últimos quinze anos e atualmente são executados em diversos tipos de hardware, como celulares e *tablets*. Os recursos podem assumir muitas formas diferentes, incluindo documentos, imagens, clipes de som ou vídeo clipes. Os documentos geralmente são escritos utilizando *Hyper Text Markup Language* (HTML), o que permite incorporar *links* para outros documentos ou para lugares diferentes do mesmo documento (GROSSKURTH; GODFREY, 2006).

Outro recurso atualmente disponível nas aplicações web é a execução das rotinas gráficas dentro dos navegadores. Uma das formas possíveis de se desenvolver aplicações gráficas para web é através da *Web Graphics Library* (WebGL), que é uma biblioteca multiplataforma executada em conjunto com a versão 5 da HTML e é baseada na especificação 2.0 da *Open Graphics Library for Embedded Systems* (OpenGL ES) (KHRONOS, 2013).

Para se trabalhar com representações gráficas, mesmo na web, não basta apenas usar a WebGL: é necessário realizar a representação computacional do objeto, inicialmente definindo-se sua descrição geométrica (forma e posição) e os seus atributos visuais (cor, transparência, material, etc.). A descrição geométrica é relacionada à sua dimensionalidade e para isso é necessário definir se o mesmo será representado no espaço bidimensional (2D) ou tridimensional (3D). No sistema de coordenadas cartesianas, o espaço 2D é formado pelos eixos *x* e *y*, enquanto que o espaço 3D é formado pelos eixos *x*, *y* e *z* (SILVA, 2007, p. 27). Em representações tridimensionais o processo de visualização completa-se com o cálculo da iluminação, pois ela simula a percepção visual do mundo físico (GOMES; VELHO, 2003, p. 419). Já a seleção de objetos gráficos representados no ambiente tridimensional é realizada

utilizando uma técnica chamada *Ray Picking*, que consiste resumidamente em disparar uma linha no eixo *z* até atingir um objeto (SCHABACK, 2011).

Dentro do exposto, este trabalho desenvolveu a continuação do ambiente VisEdu-CG (ver seção 2.4.3) desenvolvido por Montibeler (2014a), acrescentando novas rotinas gráficas como iluminação de cena e seleção de objetos, buscando desta forma aumentar a motivação dos alunos, como também busca alcançar um avanço no processo de ensino de computação gráfica.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é o desenvolvimento e disponibilização de novos recursos gráficos ao trabalho intitulado "VisEdu-CG: Visualizador de Material Educacional, Módulo de Computação Gráfica", desenvolvido por Montibeler (2014a).

Os objetivos específicos do trabalho são:

a) incluir o tipo de objeto polígono e *Spline* 3D;
b) adicionar na representação gráfica o uso de iluminação;
c) disponibilizar a seleção de objetos no espaço 3D;
d) disponibilizar as mesmas funcionalidades gráficas em 3D num espaço em 2D.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos, sendo que o primeiro contém uma introdução ao tema e em seguida são apresentados os objetivos e a estrutura do trabalho.

O segundo capítulo contempla a fundamentação teórica necessária para compreensão dos temas abordados na implementação.

No capítulo três são apresentadas as etapas de desenvolvimento deste trabalho. Primeiramente são apresentados os requisitos da aplicação. Posteriormente é exposta a especificação da aplicação, onde são descritos os casos de uso e os diagramas de classe e de sequência. Em seguida são apresentadas as ferramentas e técnicas utilizadas na implementação e também é apresentada operacionalidade da aplicação. Ainda no terceiro capítulo são discutidos os resultados obtidos com o desenvolvimento deste trabalho.

Por último, o quarto capítulo expõe as considerações finais e conclusões, finalizando com sugestões de extensões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A seção 2.1 apresenta conceitos de computação gráfica. Na seção 2.2 são apresentadas as características do ambiente VisEdu-CG 2.0 desenvolvido por Montibeler (2014a) e que o presente trabalho estendeu. A seção 2.3 apresenta as características do HTML5 e da biblioteca WebGL. Por fim, na seção 2.4 são descritos os trabalhos correlatos.

2.1 CONCEITOS DE COMPUTAÇÃO GRÁFICA

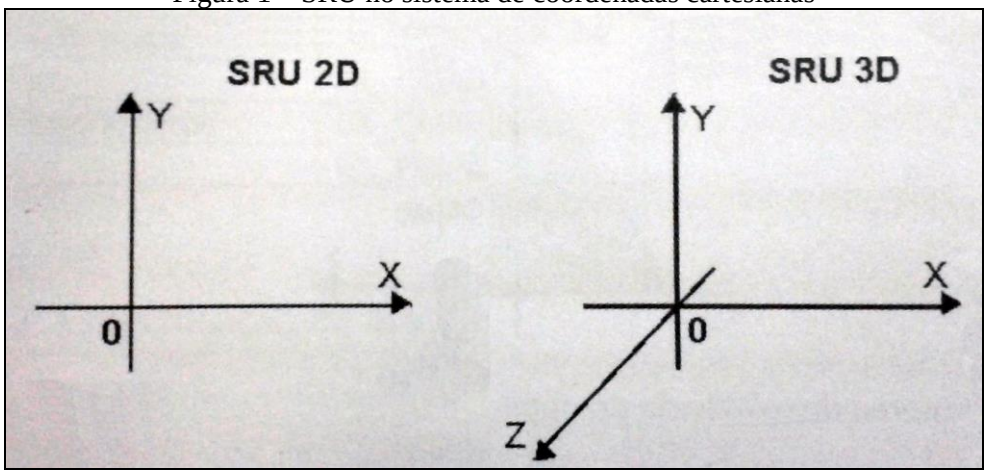
A seguir são demonstrados conceitos de computação gráfica utilizados na representação do espaço gráfico e dos objetos gráficos, tais como: sistemas de coordenadas, representação de objetos gráficos, iluminação, representação de splines e seleção em ambientes 3D.

2.1.1 Sistema de coordenadas

Para realizar a representação computacional de um objeto gráfico, deve-se inicialmente definir a descrição geométrica (forma e posição) e os atributos visuais (cor, transparência, material, entre outros) do mesmo. Esta descrição geométrica deve ser realizada em relação à dimensionalidade do objeto e para isso é necessário definir se a visualização do mesmo dar-se-á no espaço bidimensional (2D) ou tridimensional (3D) (SILVA, 2007, p. 27).

É necessário também definir em que sistema de coordenadas o objeto gráfico será descrito, para, desta forma, obter uma referência para a descrição geométrica do objeto dentro da área de trabalho. Este sistema recebe a denominação de Sistema de Referência do Universo (SRU). O sistema de coordenadas cartesianas (sistema cartesiano) formado no espaço 2D pelos eixos *x* e *y* e no espaço 3D pelos eixos *x*, *y* e *z*, é uma das formas de definições dos objetos (SILVA, 2007, p. 27). A Figura 1 ilustra o SRU no sistema de coordenadas cartesianas.

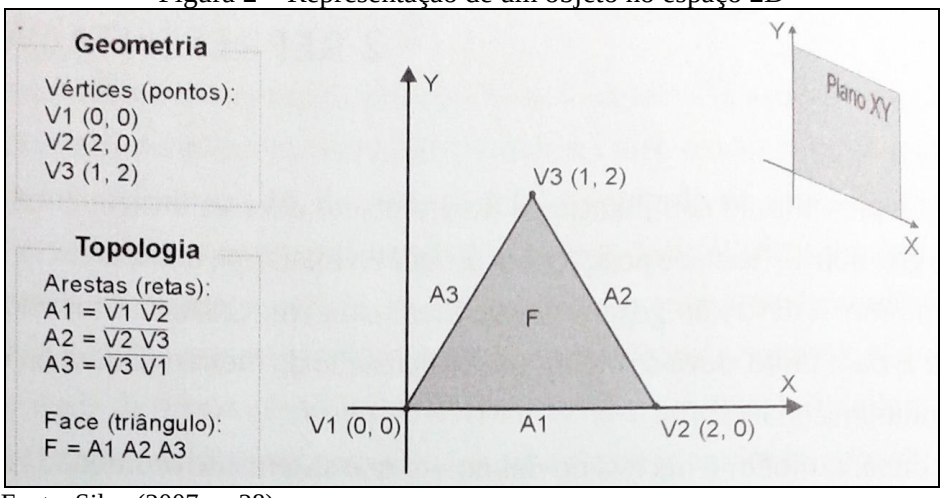
Figura 1 – SRU no sistema de coordenadas cartesianas



Fonte: Silva (2007, p. 27).

No espaço 2D, o objeto gráfico é descrito no plano nas coordenadas *x* e *y* do SRU. Para isso, informações sobre a topologia e geometria devem ser especificadas. Neste caso, entende-se por topologia, a definição de arestas e faces que ligam os vértices e geometria à localização dos elementos topológicos no espaço. A Figura 2 demonstra a representação de um objeto no espaço 2D (SILVA, 2007, p. 27).

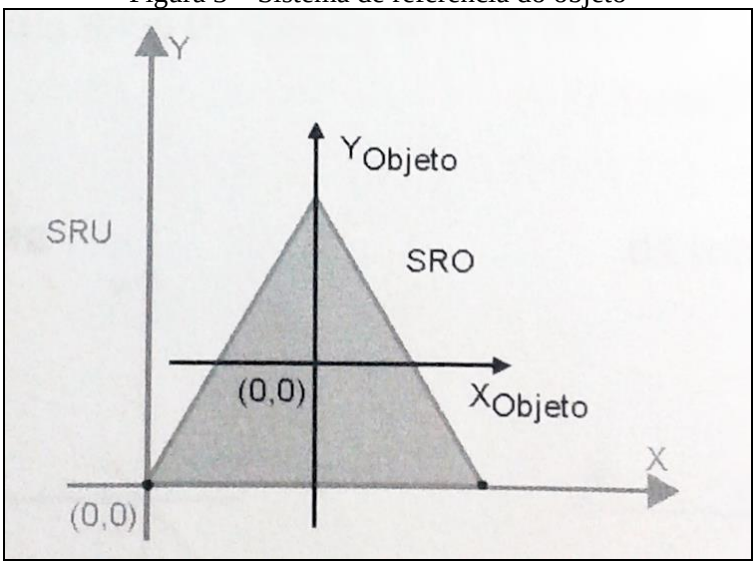
Figura 2 – Representação de um objeto no espaço 2D



Fonte: Silva (2007, p. 28).

Além do SRU, pode-se definir uma entidade em função do seu próprio sistema de referência denominado Sistema de Referência do Objeto (SRO) (SILVA, 2007, p. 28). Este sistema está ilustrado na Figura 3.

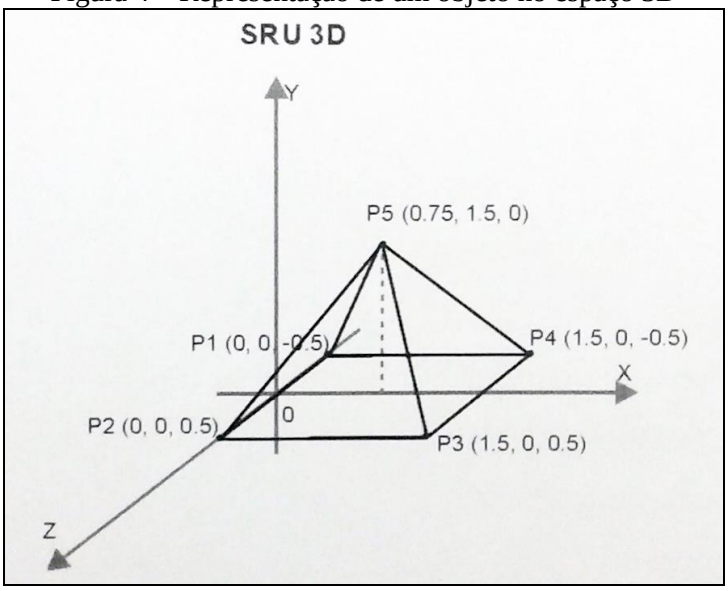
Figura 3 – Sistema de referência do objeto



Fonte: Silva (2007, p. 28).

No sistema de coordenadas 3D, os objetos gráficos são definidos em função de suas coordenadas *x*, *y* e *z*. Para encontrar um ponto no espaço 3D, deve-se localizá-lo primeiramente a partir de duas de suas coordenadas no plano correspondente (*xy*, *xz*, ou *yz*), para então projetar a terceira coordenada no espaço (SILVA, 2007, p. 77). A Figura 4 ilustra a representação de um objeto gráfico no espaço 3D.

Figura 4 – Representação de um objeto no espaço 3D



Fonte: Silva (2007, p. 77).

2.1.2 Representação de objetos gráficos

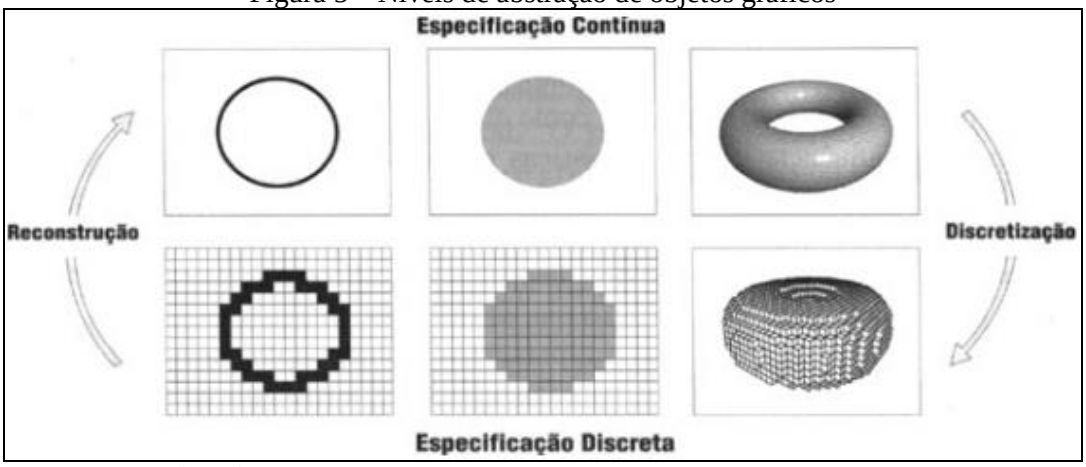
Na representação de objetos gráficos alguns conceitos são importantes, como por exemplo, o que é objeto gráfico, quais são os tipos de primitivas gráficas e como podem ser representados estes objetos utilizando os tipos de desenhos.

Neste caso, segundo Montenegro (2013), o conceito de objeto gráfico é fundamental para a computação gráfica e áreas afins. É através dos objetos gráficos que são representados a geometria (forma) e os atributos (propriedades) de um objeto do mundo real.

Uma definição mais abrangente é apresentada por Catarina (2013), em que os objetos gráficos são definidos através da especificação de seus diversos atributos. Dentre eles destacam-se os atributos geométricos, que estabelecem as propriedades métricas dos objetos gráficos; os atributos topológicos, que tratam da forma dos objetos gráficos e, por fim, os atributos de cor, que definem as informações relacionadas às texturas dos objetos gráficos.

Os objetos gráficos podem ser especificados em dois níveis distintos de abstração: nível contínuo e nível discreto. Estes níveis de abstração são apresentados na Figura 5.

Figura 5 – Níveis de abstração de objetos gráficos



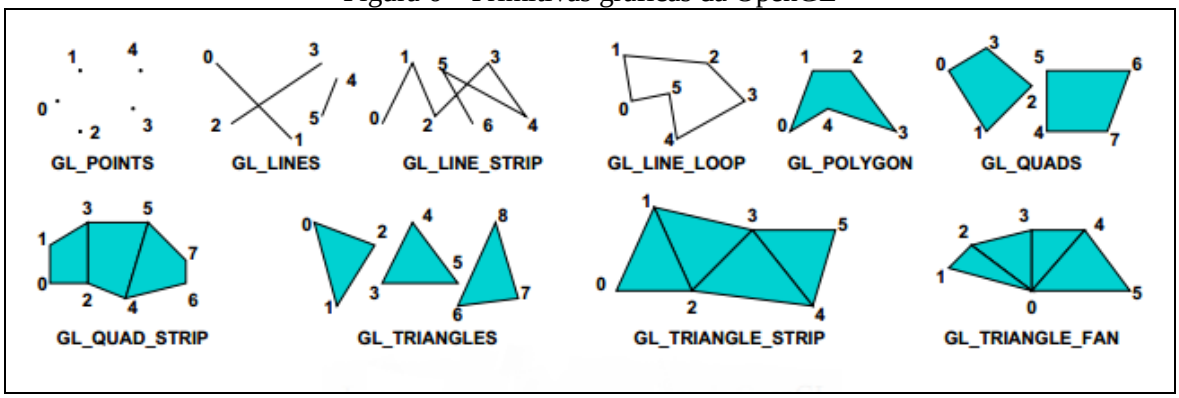
Fonte: Catarina (2013).

As primitivas gráficas são construídas a partir de seus vértices, pois a OpenGL e, consequentemente, a WebGL são interfaces para aplicações gráficas que não possuem rotinas de alto nível de abstração. Um vértice é representado em coordenadas homogêneas (*x*, *y*, *z*, *w*). Se *w* for diferente de zero, estas coordenadas correspondem a um ponto tridimensional euclidiano (*x/w*, *y/w*, *z/w*). Todos os cálculos internos são realizados com pontos definidos no espaço 3D. Caso os pontos especificados pelo usuário sejam uma representação bidimensional, o parâmetro *z* deve ser igual a zero. Os segmentos de reta são representados por seus pontos extremos e os polígonos são áreas definidas por um conjunto de segmentos (BICHO et al., 2002).

Para desenhar um conjunto de pontos, um segmento ou um polígono, os vértices necessários para a definição desta primitiva devem ser agrupados entre as chamadas das funções `glBegin()` e `glEnd()`. A biblioteca OpenGL possui dez primitivas geométricas: um tipo de ponto (`GL_POINTS`), três tipos de linhas (`GL_LINES`, `GL_LINE_STRIP`, `GN_LINE_LOOP`) e seis tipos de polígonos (`GL_POLYGON`, `GL_QUADS`, `GL_QUAD_STRIP`, `GL_TRIANGLES`,

`GL_TRIANGLE_STRIP`, e `GL_TRIANGLE_FAN`) (BICHO et al., 2002). A Figura 6 ilustra as primitivas gráficas existentes na biblioteca OpenGL.

Figura 6 – Primitivas gráficas da OpenGL



Fonte: Bicho et al. (2002).

Os tipos de desenhos podem ser representados pelo uso da *bounding box*, nuvem de pontos, aramado, sólido e textura. Esta representação se dá tanto no espaço gráfico 2D, utilizando duas coordenadas para representar os objetos gráficos, como também no espaço gráfico 3D, com os objetos gráficos utilizando três coordenadas.

A *bounding box* (caixa envolvente) de um objeto em computação gráfica é o paralelepípedo de menor tamanho que contém o objeto e cujas faces são paralelas aos planos cartesianos. Isso é, a *bounding box* de um objeto é a interseção de todos os paralelepípedos que contêm o objeto e cujas faces são paralelas aos planos coordenados (UNIVERSIDADE FEDERAL FLUMINENSE, 2013).

Nuvem de pontos, segundo Tavares (2009), são conjuntos de posições tridimensionais que podem ser associadas às informações do modelo, como por exemplo, cor e normal de superfície.

No tipo de desenho aramado, os objetos são descritos por um conjunto de arestas que define as bordas do objeto. A principal vantagem desta técnica é a sua velocidade na exibição dos modelos, sendo necessário exibir apenas um conjunto de linhas. Um dos inconvenientes do uso da representação aramada é o fato de gerar uma representação ambígua com margem para várias interpretações (AZEVEDO; CONCI, 2003, p. 125).

Na modelagem de sólidos, no ambiente 2D, um conjunto de segmentos de reta ou curva não formam necessariamente uma área fechada. Em 3D, uma coleção de superfícies não envolve necessariamente um volume fechado. A representação deve ser não ambígua e única: uma representação deve corresponder a um único sólido e cada objeto deve ter apenas uma representação possível. A representação única permite comparar dois objetos para determinar a igualdade (SOUSA, 2004).

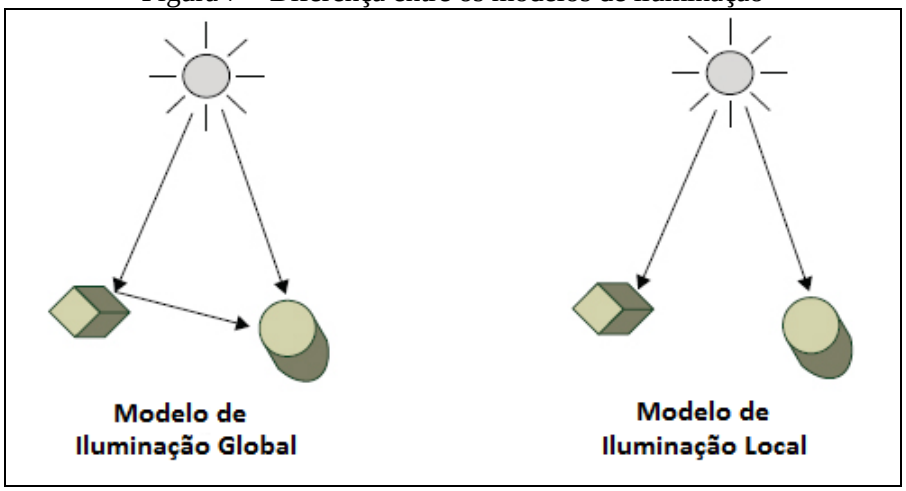
As texturas são um padrão visual que possuem propriedades de homogeneidade que não resultam simplesmente de uma cor ou intensidade. São compostas de um grande número de elementos similares com uma certa ordenação. São descritas por medidas que quantificam suas propriedades de suavidade, rugosidade e regularidade. Possuem características estáticas ou propriedades locais constantes, com pouca variação (UNIVERSIDADE FEDERAL FLUMINENSE, 2009).

2.1.3 Iluminação

Para simular a luz do mundo real em gráficos 3D é preciso utilizar algum tipo de modelo de luz que pode ser simplificado em relação à forma como a luz do mundo real funciona. Dois tipos diferentes de modelos de iluminação podem ser usados ao simular a luz em gráficos 3D: o modelo de iluminação global e o modelo de iluminação local.

Um modelo de iluminação global utiliza informações de outros objetos que estão sendo iluminados diretamente. Neste modelo um objeto pode ser iluminado pela luz refletida em outros objetos. Um modelo de iluminação local representa apenas a luz que vem de fontes diretas. Como resultado, os objetos não são iluminados pela luz que é refletida de outros objetos (ANYURU, 2012, p. 249). A Figura 7 ilustra a diferença entre os modelos de iluminação.

Figura 7 – Diferença entre os modelos de iluminação



Fonte: Anyuru (2012, p. 250).

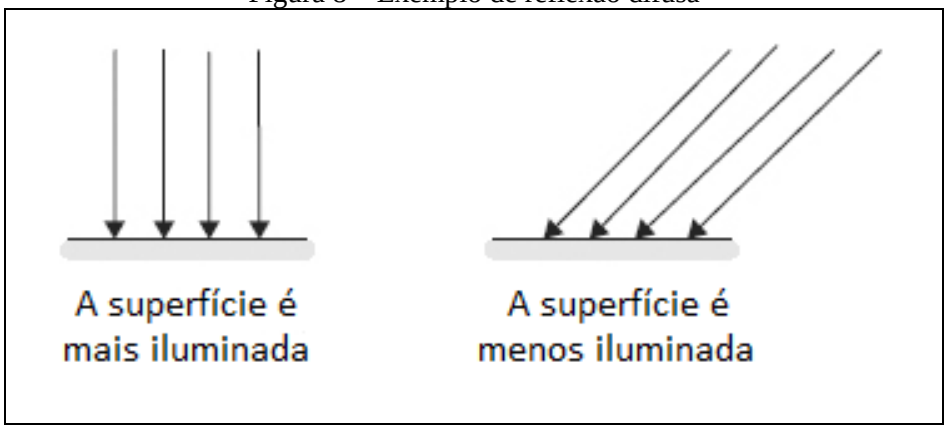
Como geralmente a biblioteca WebGL é utilizada para gráficos 3D, uma forma de iluminação local é frequentemente utilizada. Um modelo comum utilizado para representar o modelo de iluminação local é o modelo de Phong, que recebe este nome em homenagem a um pesquisador de computação gráfica chamado Bui Tuong Phong, que desenvolveu este modelo e publicou em 1973 (ANYURU, 2012, p. 251).

Este modelo baseia-se no fato de que a cor de um objeto do mundo real é indicada pela cor da luz que deixa sua superfície. Sendo assim, um objeto que parece vermelho reflete na maior parte a luz vermelha. Neste modelo, a cor resultante de um ponto é a soma de três componente de reflexão diferentes: ambiente, difusa e especular. Para o cálculo de cada um dos três componentes de reflexão diferentes, o modelo também prevê que todas as fontes de luz na cena têm três componentes diferentes de luz: a luz ambiente, a luz difusa e a luz especular (ANYURU, 2012, p. 251).

A reflexão ambiente é a luz que está espalhada pela cena de tal forma que não aparenta ter uma origem particular. Como resultado todos os lados de um objeto são uniformemente iluminados pela luz ambiente.

A reflexão difusa leva em conta a direção da luz incidente quando a quantidade de luz refletida é calculada. Os raios de luz que atingem a superfície em um ângulo perpendicular refletem mais luz e, portanto, iluminam a superfície mais do que os raios de luz que atingem a superfície com um ângulo que fique entre a superfície e os raios de luz (ANYURU, 2012, p. 251). A Figura 8 ilustra um exemplo de reflexão difusa.

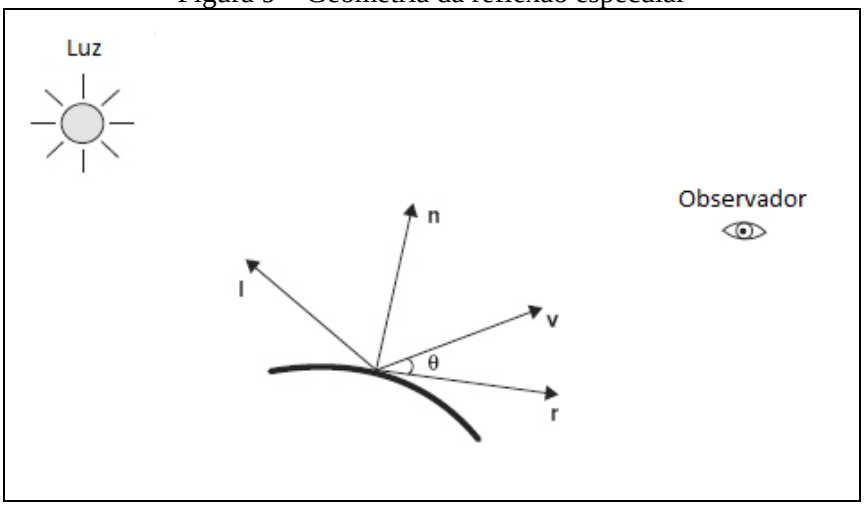
Figura 8 – Exemplo de reflexão difusa



Fonte: Anyuru (2012, p. 253).

A reflexão especular, ao contrário da reflexão ambiente e da reflexão difusa, leva em conta a direção da visão. A luz especular se reflete da mesma forma como a luz é refletida em um espelho. A maior parte da luz é refletida numa direção específica e, portanto, a direção da observação é importante. A Figura 9 demonstra a geometria da reflexão especular.

Figura 9 – Geometria da reflexão especular



Fonte: Anyuru (2012, p. 253).

Na iluminação, existem também diferentes tipos de fontes de luz e formas como elas emitem esta luz. Todo objeto em uma cena é potencialmente uma fonte de luz. A luz pode ser emitida ou refletida de objetos. Geralmente é realizado uma distinção entre emissores de luz e refletores de luz. Os emissores são as fontes de luz (lâmpadas, velas, fogo, sol, estrelas) e os refletores são normalmente os objetos que serão coloridos de maneira realística (*renderizados*). Dentre os tipos de emissores existentes destacam-se a luz ambiente, ponto de luz, luz direcional e luz refletora.

Uma luz é considerada ambiente quando não tem uma direção identificada. Esse modelo faz a simulação da iluminação geral vinda da reflexão da luz em muitas superfícies difusas e é ela que determina o nível de iluminação das superfícies no ambiente. Uma técnica comum para melhorar o realismo em cenas externas é alterar a cor da luz ambiente para complementar a luz principal (AZEVEDO; CONCI, 2003, p. 274).

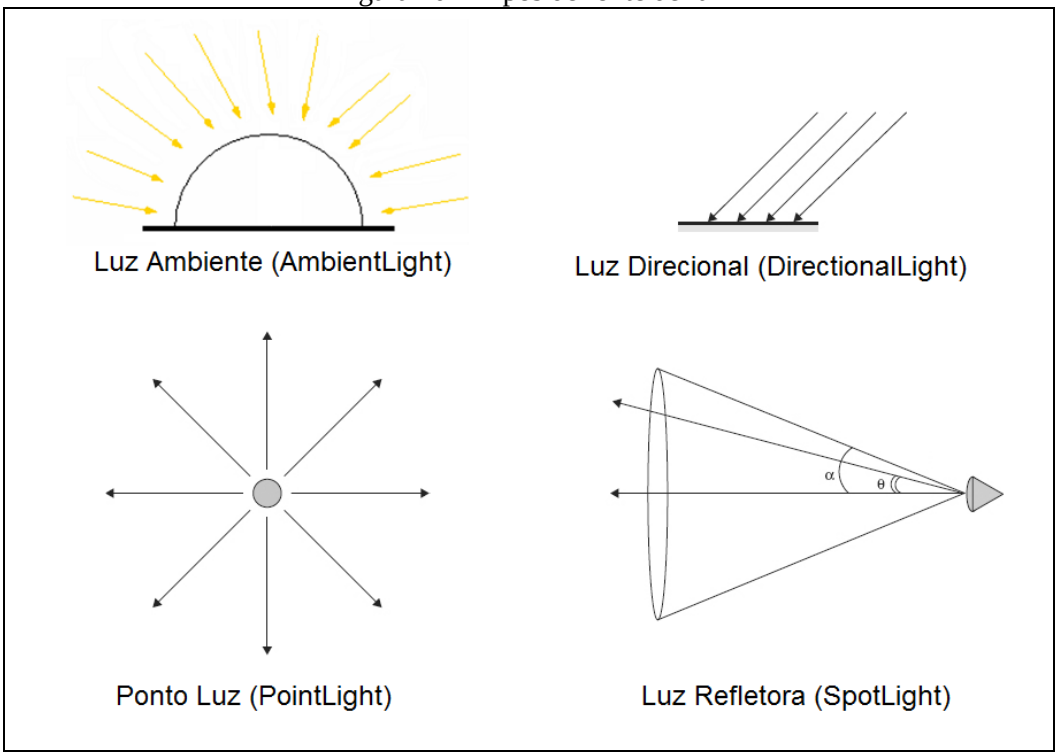
No tipo ponto de luz (*Point Light*) um ponto desta luz está localizado em uma posição específica e emite a luz em todas as direções. Um exemplo de uma luz do mundo real que pode ser modelado com um ponto de luz é uma lâmpada sem um abajur.

O tipo de luz direcional simula uma fonte de luz que está infinitamente distante da cena. Por ser infinitamente distante, os raios de luz que atingem a superfície são paralelos um ao outro e tem a mesma direção. A fonte de luz é definida por uma direção em vez de uma localização (ANYURU, 2012, p. 170). Segundo Azevedo e Conci (2003), os raios de luz iluminam em uma direção única, como o sol faz na superfície da Terra. As luzes direcionais são principalmente usadas para simular luz solar.

Por fim, o tipo de luz refletora (*spot light*) emite a luz em forma de cone em uma determinada direção. Um exemplo de uma fonte de luz no mundo real que pode ser modelado

como uma luz refletora é um farol de um carro (ANYURU, 2012, p. 170). A Figura 10 ilustra os tipos de fonte luz.

Figura 10 – Tipos de fonte de luz

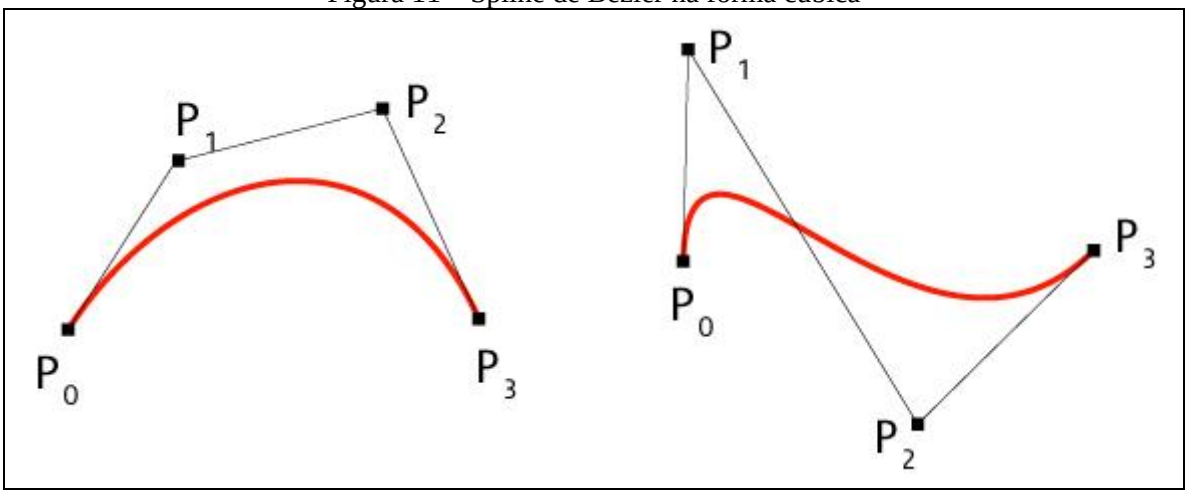


Fonte: adaptado de Anyuru (2012, p. 170).

2.1.4 Splines

O objeto gráfico spline é uma curva definida matematicamente por dois ou mais pontos de controle, podendo ser desenhada no espaço gráfico 2D ou 3D. Elas exercem um papel importante em diversas áreas tanto na criação de objetos sintéticos quanto na visualização de fenômenos científicos. Na modelagem geométrica em computação gráfica, as splines são a base desde a geração de formas simples, como círculos e elipses, como também na criação de projetos mais complexos como modelagem de automóveis, navios ou aeronaves. A spline de Bézier foi desenvolvida por Pierre Bézier durante seus trabalhos em projetos de automóveis para a Renault francesa no início da década de 1960. Bézier utilizou pontos de controle para determinação das tangentes nos pontos de início e fim da spline e não vetores, como descreveu Hermite no uso de polinômios de terceira ordem para ajustes de curvas. Para ajuste por um polinômio de grau *n*, a spline de Bézier pode ser gerada por 3, 4, até n + 1 pontos de controle. Porém, geralmente em computação gráfica a spline de Bézier é utilizada em sua forma cúbica, necessitando apenas de quatro pontos de controle (AZEVEDO; CONCI, 2003, p. 87) A Figura 11 apresenta a spline de Bézier em sua forma cúbica, utilizando os pontos de controle P0, P1, P2 e P3.

Figura 11 – Spline de Bézier na forma cúbica



Fonte: Cartouche (2012).

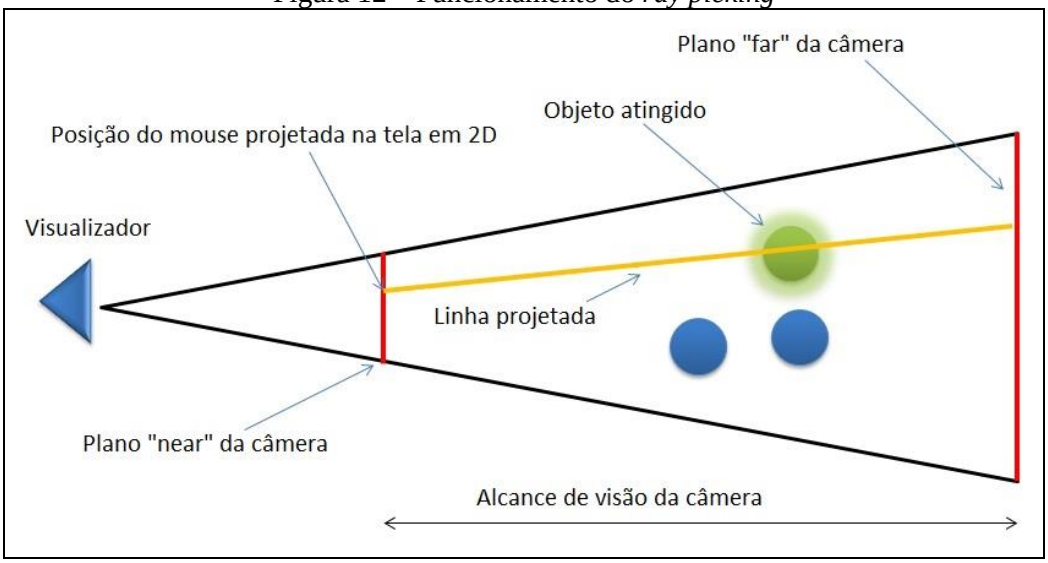
2.1.5 Seleção em ambiente 3D

O processo de encontrar objetos em uma cena no ambiente 3D, com base na entrada do usuário chama-se *picking*. Para determinar em qual objeto gráfico o usuário clicou com o *mouse*, são utilizadas as coordenadas 2D como referência. Uma maneira comum de implementação de *picking* é a abordagem chamada *ray picking*.

Segundo Schaback (2011), o processo *ray picking* consiste em atirar uma linha através da cena até atingir um objeto. Para isso é necessário saber a posição da câmera e o ponto da tela principal, por exemplo, a posição do cursor do *mouse*. Uma das formas de obter os pontos *x* e *y* da tela seria utilizando a matriz inversa, porém pode tornar a rotina com um custo de processamento elevado.

Em vez disso, Schaback (2011) apresenta uma solução calculando a posição do plano da tela principal no espaço global e mapeando o ponto 2D neste plano e de lá para o espaço global. Desta forma, através da intersecção da linha é possível determinar qual objeto está sendo atingido pelo cursor do *mouse* e consequentemente é possível selecionar todos os objetos visíveis da cena. A Figura 12 ilustra o funcionamento do *ray picking*.

Figura 12 – Funcionamento do *ray picking*



Fonte: adaptado de Filipek (2012).

O algoritmo é executado inicialmente projetando uma linha a partir da posição em 2D do *mouse* na tela. Esta linha percorre o espaço gráfico que está dentro do alcance da visão da câmera. Ao final da execução, é retornado uma lista dos objetos atingidos pela linha, ordenada de forma crescente da sequência em que os objetos foram encontrados.

2.2 VISEDU-CG 2.0

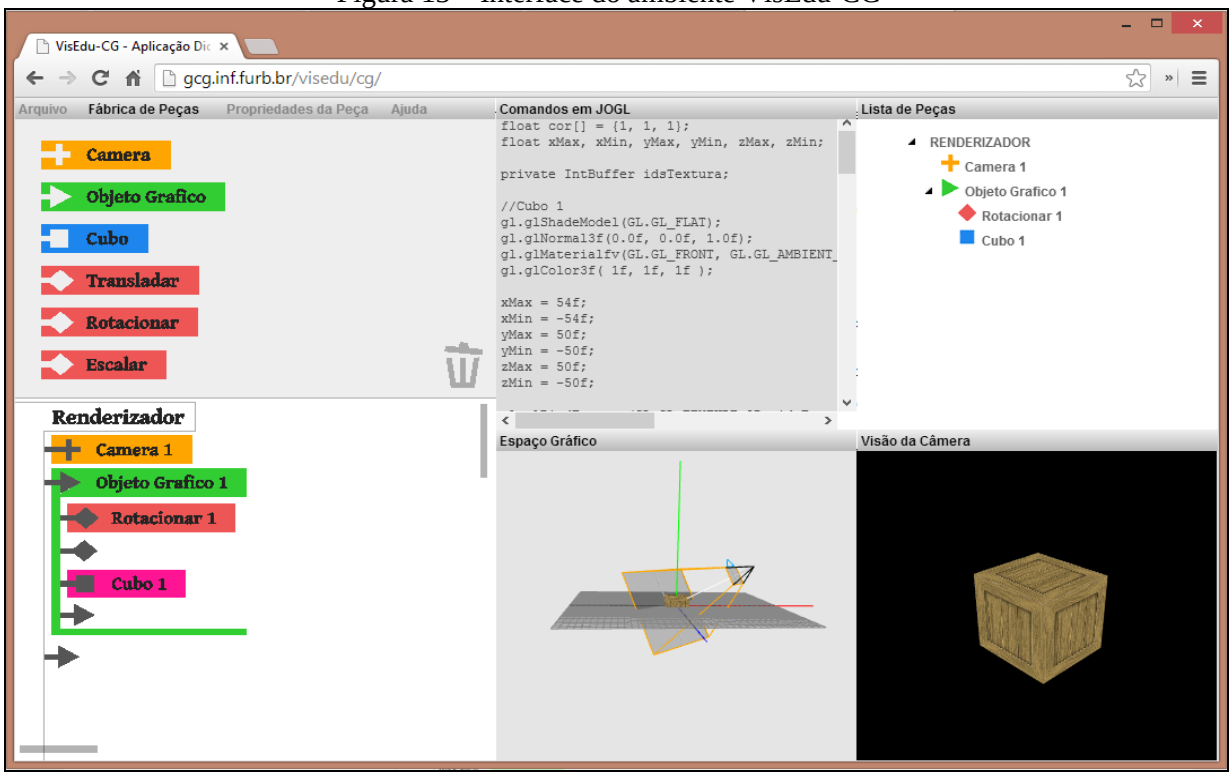
O ambiente VisEdu-CG 2.0 é aplicação educacional voltada ao aprendizado de computação gráfica em ambiente Web, utilizando HTML5 juntamente com a biblioteca multiplataforma WebGL.

Pode ser definido como um *framework* para estudo de computação gráfica com algumas funções prontas da WebGL como translação, rotação e escala. O ambiente utiliza o mesmo conceito de encaixe de peças presentes nos jogos infantis de encaixe de formas geométricas. Contudo, criou-se uma estrutura mais lúdica para o encaixe das peças. Desta forma, para cada tipo de peça existente foi criado um recorte de uma forma geométrica simples para auxiliar o usuário na identificação correta do local de encaixe de cada peça.

Cada uma das peças representa um conceito básico de computação gráfica e realiza uma interação com o resultado 3D que é gerado pelo encaixe e pelas propriedades da peça. A manipulação das propriedades dos objetos gráficos, como também a utilização das funções gráficas, são realizadas através da utilização de peças que se encaixam de forma intuitiva pelo aluno. Os resultados desta manipulação são apresentados em uma das seções do ambiente chamada `Espaço gráfico` (MONTIBELER, 2014a, p.55).

O ambiente possui ainda outras seções como a `Visão da câmera`, `Lista de peças` e uma seção para a exibição do código fonte necessário para reproduzir o resultado gráfico chamada `Comandos JOGL`. Também estão disponíveis no ambiente a utilização de câmeras, aplicação de textura, além da exportação da cena gráfica no formato JSON (MONTIBELER, 2014a, p.27-28). A Figura 13 demonstra a interface do ambiente VisEdu-CG.

Figura 13 – Interface do ambiente VisEdu-CG



2.3 HTML5 E WEBGL

O HTML5 é um trabalho que ainda está em progresso, desenvolvido pela cooperação entre a *World Wide Web Consortium* (W3C) e a *Web Hypertext Application Technology Working Group* (WHATWG). No entanto, os principais navegadores suportam muitos dos seus novos recursos (W3CSCHOOLS, 2013).

A quinta versão do HTML traz diversas funcionalidades como semântica e acessibilidade. Foram incluídos recursos que antes só eram possíveis com a utilização de tecnologias de terceiros, por meio de *plug-ins*, como o suporte à reprodução de áudio e vídeo. Dentre os novos recursos disponibilizados, destaca-se a inclusão do componente `canvas` para desenhos, componentes para reprodução de áudio e vídeo, armazenamento local de arquivos e o WebGL para gráficos 3D (W3C, 2013).

O WebGL é uma *Aplication Programming Interface* (API) projetada para a web. É derivada do OpenGL ES 2.0 e fornece uma funcionalidade de renderização semelhante,

porém em um contexto de HTML. Por padrão, o WebGL vem disponível nos navegadores mais populares (KHRONOS, 2013).

A integração entre o WebGL e o HTML5 acontece através do objeto `WebGLRenderingContext`, que é o responsável pelo acesso aos recursos 3D do WebGL. Este objeto deve ser criado chamando o método `getContext()` de um elemento `canvas` do HTML5.

Neste contexto, o `Three.js` é uma biblioteca gráfica 3D JavaScript de código aberto em HTML5 para navegadores web modernos que pode renderizar seus objetos para o `canvas`, WebGL e *Scalabe Vector Graphics* (SVG). Esta biblioteca dá suporte completo para as funcionalidades do WebGL e permite que seja utilizado o mesmo código para os outros renderizadores (FHTR, 2012). A biblioteca também possui *helpers* para as formas geométricas mais comuns, tais como esferas, cubos e cilindros, além de um sistema de partículas completamente funcional, mapeamento de texturas e suporte a detecção de colisões em nível básico (WILLIAMS, 2011, p.117).

O `Three.js` possui quatro componentes básicos em sua estrutura: o renderizador, a cena, a câmera e os objetos. Entre os recursos disponibilizados na `Three.js` destaca-se o fato de funcionar em todos os *browsers* que suportam WebGL e permitir adicionar e remover objetos da cena em tempo de execução (FHTR, 2012).

Em sua mais recente versão, o `Three.js` dá suporta a seleção de objetos do tipo `lines` no espaço gráfico 3D. Além disso, disponibiliza uma classe específica para trabalhar com esqueletos e ossos. Outro recurso disponibilizado é a criação de uma classe para trabalhar com *caches* de arquivos. Há também diversas melhorias e correções em recursos existentes, como por exemplo, a melhoria no desempenho ao adicionar e/ou remover objetos da cena (MRDOOB, 2014).

2.4 TRABALHOS CORRELATOS

A seguir é apresentado o software educacional StarLogo TNG (STEP, 2013) voltado para o ensino na área da computação. Em seguida, é apresentado também o desenvolvimento de um motor de jogos 3D, utilizando WebGL (PEREIRA, 2012).

2.4.1 StarLogo TNG

O StarLogo TNG é um sistema de modelagem e simulação desenvolvido pelo *Massachusetts Institute of Technology* (MIT) *Scheller Teacher Education Program* (STEP) voltado para o estudo de sistemas descentralizados, ou seja, sistemas que são formados a

partir da interação individual entre vários objetos distintos, em vez de sistemas que são controlados centralmente (STEP, 2013).

O objetivo é motivar as pessoas mais jovens na programação por meio de ferramentas que facilitam o desenvolvimento de sistemas, como jogos 3D com recursos de áudio. Para isso, o software conta com uma interface de programação baseada em blocos. Os elementos da linguagem são representados por estes blocos coloridos que se encaixam como peças de um quebra-cabeça, facilitando aos usuário o entendimento do sistema (STEP, 2013).

O StarLogo TNG executa sobre a plataforma *Java Virtual Machine* (JVM) e por este motivo independe de sistema operacional. O StarLogo TNG não possui uma versão que execute em ambiente web (STEP, 2013). A Figura 14 demonstra a interface de programação do StarLogo TNG.

Figura 14 – Interface de programação do StarLogo TNG

Fonte: Mazurok (2013).

O StarLogo TNG trabalha no espaço gráfico 3D e disponibiliza os painéis de forma dinâmica, podendo habilitar e desabilitar a visualização dos mesmos. Além disso, o ambiente permite a seleção no espaço gráfico e a exportação da cena para outras aplicações. Entre as limitações estão o fato de não disponibilizar iluminação de cena e da aplicação não possuir uma versão para Web (STEP, 2013).

2.4.2 Desenvolvimento de um motor de jogos 3D utilizando WebGL

O motor de jogos, desenvolvido por Pereira (2012), é uma aplicação web que tem como propósito disponibilizar um motor de jogos para as pessoas que não tem muita experiencia com desenvolvimento de jogos. A aplicação disponibiliza funcionalidades como

grafo de cena, gerenciamento de objetos da cena, texturas e iluminação. Foi adotada uma solução para que o desenvolvedor não necessite de conhecimentos em APIs de baixo nível da WebGL e GLSL para programar seu jogo ou aplicativo da área de computação gráfica. A arquitetura adotada foi baseada na arquitetura do V-ART, sendo necessárias algumas adaptações devido a diferenças entre a linguagem JavaScript e C++, porém a estrutura das classes principais foi mantida (PEREIRA, 2012, p. 259).

Com relação à iluminação, o motor de jogos possui duas limitações: foi utilizado o modelo de iluminação de Phong, não disponibilizando outros modelos, limitando as possibilidades do desenvolvedor e não permite que haja múltiplas luzes na cena (PEREIRA, 2012, p. 254).

Segundo Pereira (2012, p. 59), o motor de jogos mostrou um bom desempenho na execução dos testes, conseguindo desenhar mais de mil objetos a mais de 30 Frames Por Segundo (FPS) em dois dos três navegadores testados, sendo este número o suficiente para o usuário não perceber a troca de quadros na cena. Ainda foi obtido o resultado de 60 FPS em cenas com 255 objetos ou menos, o que permite com que o usuário veja a cena de maneira mais natural, especialmente em cenas onde ocorre uma interação muito grande entre a aplicação e o usuário (PEREIRA, 2012, p. 59).

3 DESENVOLVIMENTO

Este capítulo trata de assuntos pertinentes a implementação do projeto. Encontram-se aqui os requisitos, a especificação, os detalhes da implementação e, por fim, os resultados e discussões.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos do ambiente VisEdu-CG 2.0, desenvolvido por Montibeler (2014a), foram mantidos e podem ser encontrados no Anexo A. O ambiente VisEdu-CG em sua versão 3.0 deverá:

a) permitir optar por trabalhar em 2D ou 3D (Requisito Funcional – RF);
a) permitir mudar a primitiva gráfica e o tipo de desenho utilizado (RF);
b) permitir manipular os controles de iluminação da cena (RF);
c) permitir incluir tipo de objeto polígono e *Spline* 3D (RF);
d) permitir selecionar os objetos gráficos no espaço 3D (RF);
e) ser desenvolvido na linguagem HTML5 (Requisito Não Funcional – RNF);
f) utilizar o WebGL para executar as funções de computação gráfica (RNF);
g) ser de código aberto (RNF).

3.2 ESPECIFICAÇÃO

Para especificar a implementação do projeto, foram utilizados diagramas da *Unified Modeling Language* (UML), sendo eles os diagramas de casos de uso, de classe e de sequência. A modelagem e a geração dos diagramas foram realizadas utilizando a ferramenta Enterprise Architect 9.1.

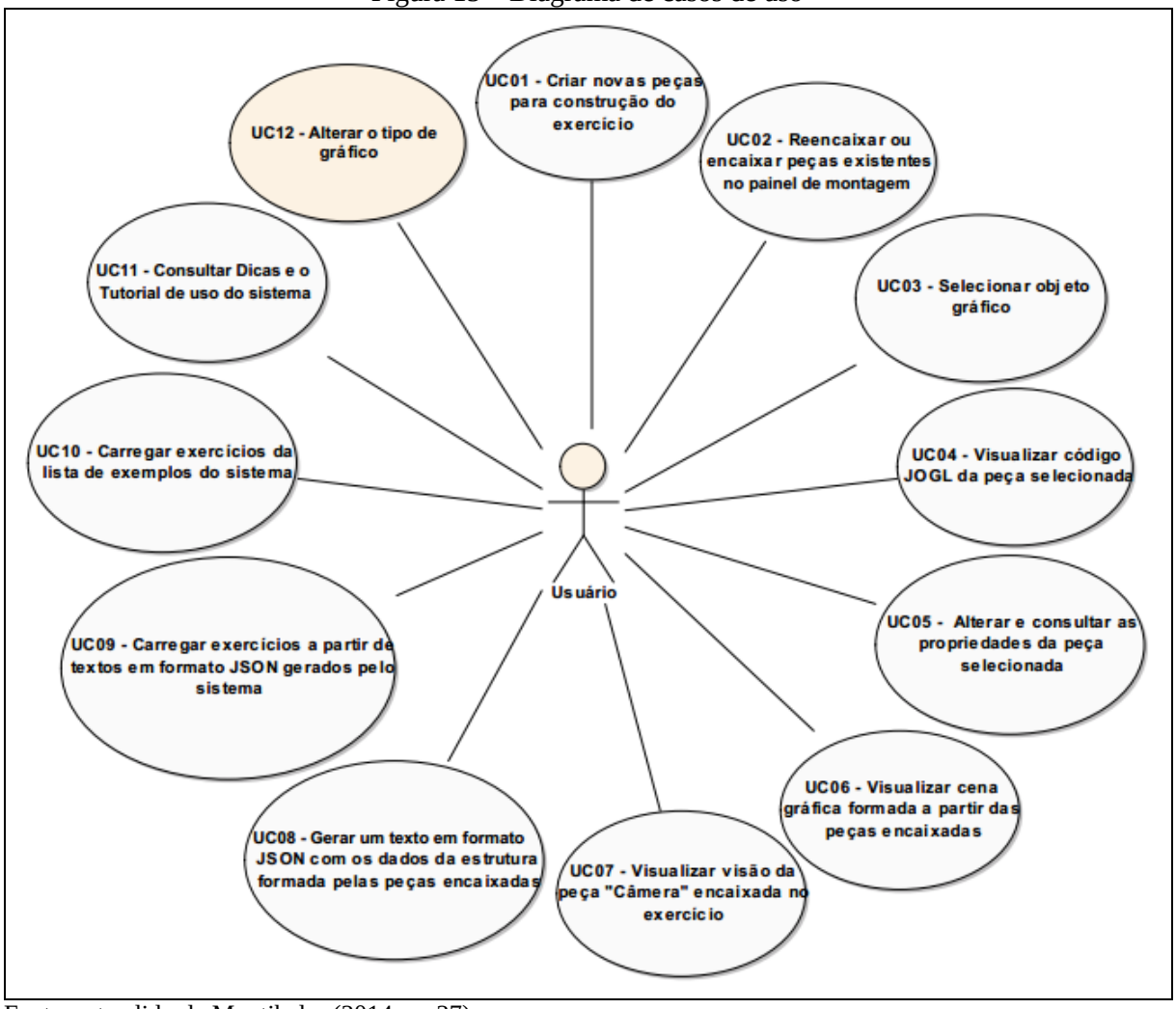
3.2.1 Diagrama de casos de uso

Nesta seção são descritos os principais casos de uso da aplicação. Todos os casos de uso desenvolvidos por Montibeler (2014a) foram mantidos. No entanto, os casos de uso UC01, UC02, UC03, UC05, UC06, UC07 e UC08 foram ajustados de acordo com os novos recursos desenvolvidos. Além destes, foi acrescentado o caso de uso UC12 que contempla a opção de trabalhar com gráficos em 2D ou 3D. Detalhes sobre os casos de uso UC04, UC09 e UC10 desenvolvidos por Montibeler (2014a, p.30-33) e que não sofreram alterações podem ser encontrados no Anexo B.

Segundo Montibeler (2014a, p.27), para compreensão dos casos de uso é necessário ter conhecimento prévio das partes que formam a aplicação. O detalhamento das partes que

formam a aplicação pode ser encontrado no Anexo C. Ainda segundo Montibeler (2014a, p.27), há somente um ator que é o próprio usuário do sistema, podendo ser o aluno ou o professor da disciplina. O detalhamento do novo caso de uso (UC12), juntamente com os casos de uso que sofreram alteração estão disponíveis no Apêndice A. A Figura 15 ilustra o diagrama de casos de uso, destacando o novo caso de uso UC12.

Figura 15 – Diagrama de casos de uso



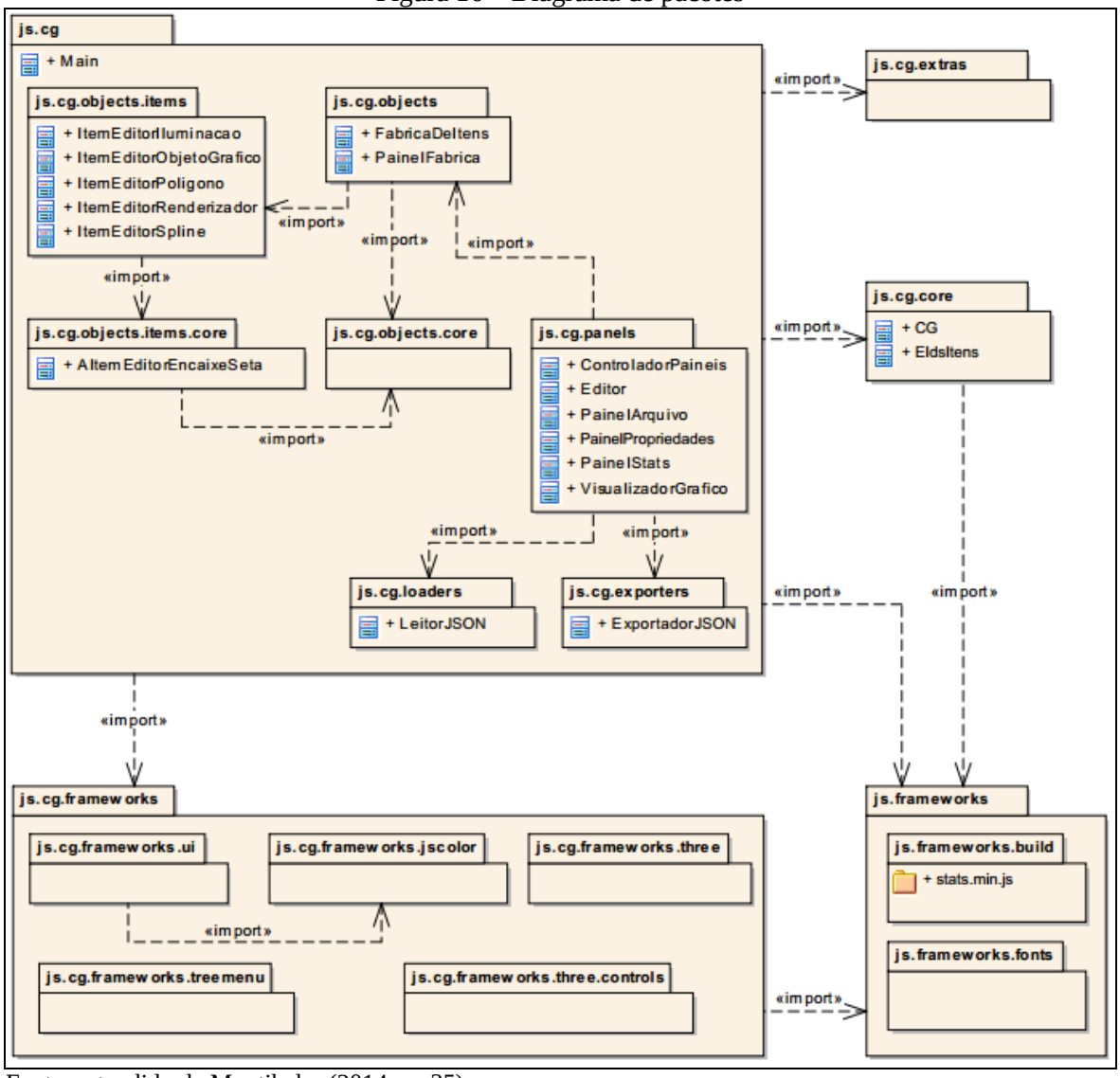
Fonte: estendido de Montibeler (2014a, p.27).

O caso de uso UC01, desenvolvido por Montibeler (2014a, p.28-29), foi alterado para contemplar a criação dos objetos gráficos `Polígono` e `Spline` e também a utilização do objeto `Iluminação`. O caso de uso UC01 está descrito em detalhes no Apêndice A. O caso de uso UC03, desenvolvido por Montibeler (2014a, p.30), foi alterado para contemplar a seleção dos objetos através do duplo clique do *mouse* no `Espaço Gráfico`. O detalhamento deste caso de uso está no Apêndice A. O novo caso de uso UC12, descrito em detalhes no Apêndice A, representa a interação entre o `Usuário` e a funcionalidade que permite trabalhar com gráficos em 2D ou 3D.

3.2.2 Diagrama de classes

Nesta seção são descritas as novas classes desenvolvidas e suas relações. Também são descritas as classes desenvolvidas por Montibeler (2014a) e que foram modificadas para atender os novos recursos. Todas as demais classes desenvolvidas por Montibeler (2014a) foram mantidas e os principais diagramas estão ilustrados no Anexo D. Na Figura 16 são exibidos os pacotes principais da aplicação com as novas classes desenvolvidas.

Figura 16 – Diagrama de pacotes



Fonte: estendido de Montibeler (2014a, p.35).

Além das classes já existentes foram adicionadas as classes `ItemEditorPoligono`, `ItemEditorSpline` e `ItemEditorIluminacao` que tratam dos novos itens disponíveis na `Fábrica de Peças` para montagem. Também foi incluída a classe `PainelStats` juntamente com a biblioteca `stats.min.js` que serve para o monitoramento de performance da

aplicação. Todas as classes ilustradas no diagrama da Figura 16 são abordadas com mais detalhes a seguir.

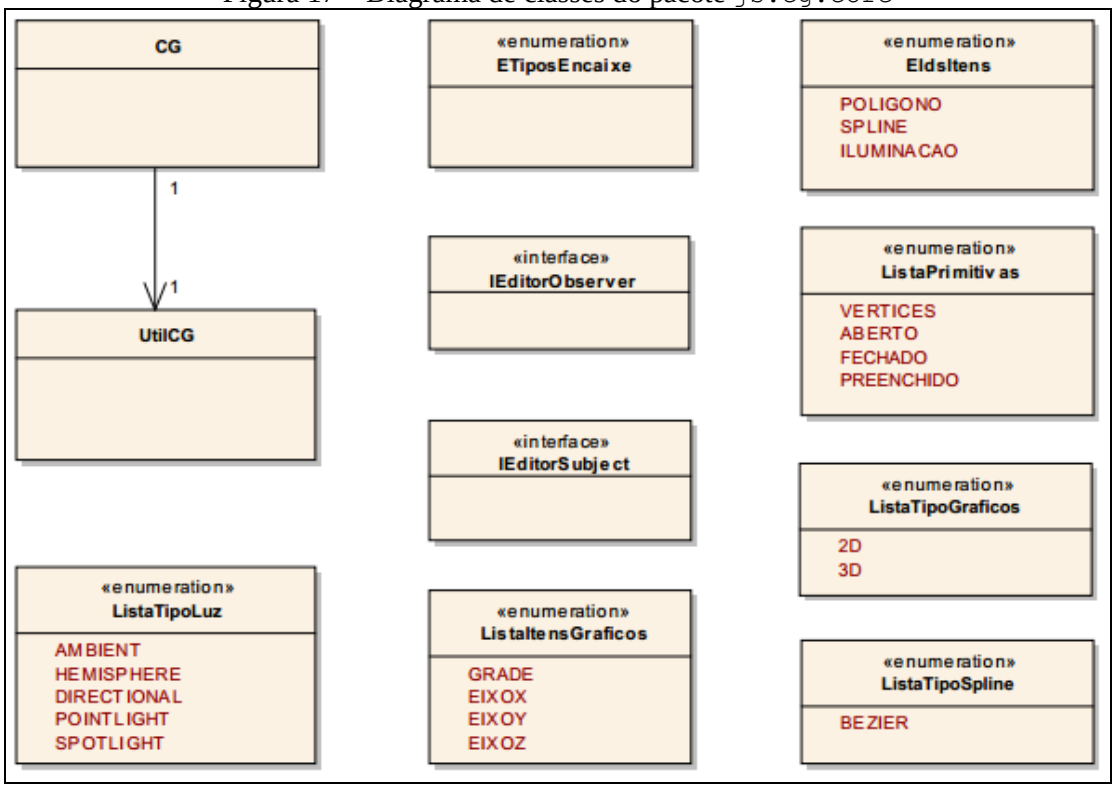
3.2.2.1 Pacote `js.frameworks`

No pacote `js.frameworks` estão as principais bibliotecas de apoio que foram utilizadas no desenvolvimento. Nesse pacote, dentro do subpacote `js.frameworks.build` foi adicionado uma nova biblioteca denominada `stats.min.js`, responsável por realizar o monitoramento da execução dos *frames* da aplicação em FPS.

3.2.2.2 Pacote `js.cg.core`

No pacote `js.cg.core` estão as classes que servem de apoio para as demais classes do sistema. A Figura 17 ilustra o diagrama de classes desse pacote.

Figura 17 – Diagrama de classes do pacote `js.cg.core`



Fonte: estendido de Montibeler (2014a, p.38).

A classe `CG` contém todas as funções que são úteis a toda a aplicação. Nela podem ser encontradas as definições de cores das peças da `Fábrica`, como também as listas de exemplos e de texturas da peça `Cubo`. Neste pacote foram incluídas cinco novas enumerações: `ListaPrimitivas`, que contém as primitivas disponíveis nas propriedades da peça `Polígono`; `ListaTipoSpline`, que contém a lista de tipos de splines disponíveis nas propriedades da peça `Spline`; `ListaTipoLuz`, que contém os tipos de luzes disponíveis nas propriedades da

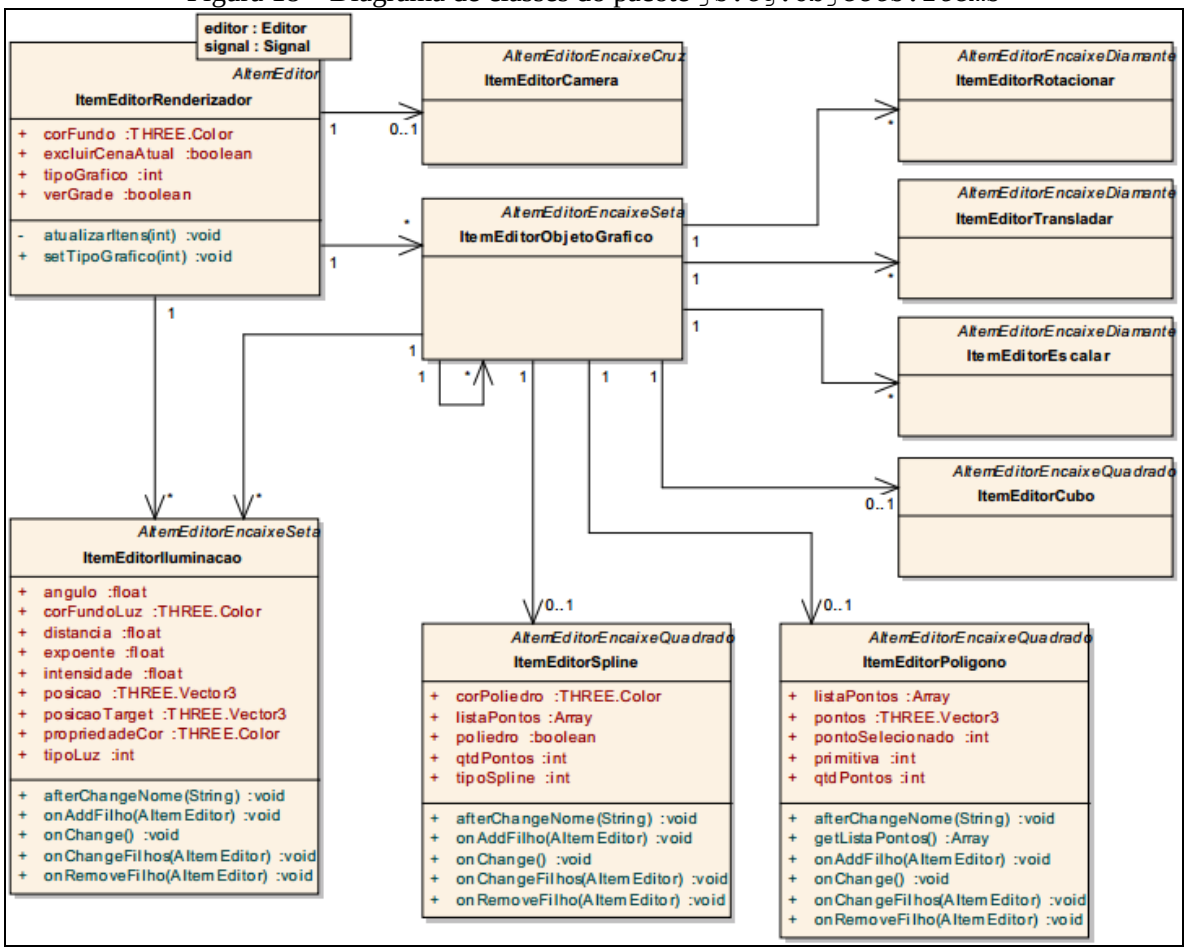
peça `Iluminação`; `ListaItensGraficos` que contém a lista de itens gráficos (`Eixo` e `Grade`) que servem de referência no `Espaço Gráfico` e por fim, `ListaTipoGraficos` que contém os itens referentes aos tipos de gráficos que podem ser utilizados na aplicação (2D ou 3D).

A enumeração `EIdsItens` possui um identificador para cada item disponível na `Fábrica de Peças`. Há também um contador para armazenar a quantidade de cada item criado juntamente com um número sequencial único para cada peça. Nessa enumeração foram incluídos os novos itens da fábrica `Polígono`, `Spline` e `Iluminação`.

3.2.2.3 Pacote `js.cg.objects.items`

No pacote `js.cg.objects.items` estão as classes que representam os itens disponíveis para construção na `Fábrica de Peças`. A Figura 18 apresenta o diagrama de classes desse pacote.

Figura 18 – Diagrama de classes do pacote `js.cg.objects.items`



Fonte: estendido de Montibeler (2014a, p.44).

Segundo Montibeler (2014a, p.45), a classe `ItemEditorRenderizador` é responsável pela peça principal do exercício, ou seja, todas as demais peças serão associadas a partir dela. Além dos métodos e atributos existentes na classe e que foram mantidos, foram adicionados

os atributos `corFundo`, `verGrade`, `verEixos` e `tipoGrafico` que representam as propriedades da peça `Renderizador`. Já o atributo `excluirCenaAtual` foi criado para identificar o momento em que a aplicação deve excluir os itens da `Fábrica` quando o usuário realiza a alteração do tipo de gráfico (2D, 3D). Foram desenvolvidos dois novos métodos: `atualizarItens` que é chamado quando ocorre uma alteração no tamanho da janela ou quando a cena atual é excluída e o método `setTipoGrafico` que atualiza o atributo `tipoGrafico` passando esta informação também para a classe `Editor`.

As classes `ItemEditorIluminacao`, `ItemEditorSpline` e `ItemEditorPoligono` possuem os métodos `afterChangeNome`, `onAddFilho`, `onChange`, `onChangeFilhos` e `onRemoveFilhos` que são herdados da classe abstrata `AObjetoGrafico`.

A classe `ItemEditorIluminacao` representa a peça `Iluminação` e herda os métodos e atributos da classe abstrata `AItemEditorEncaixeSeta`. Ela está associada às classes `ItemEditorRenderizador` e `ItemEditorObjetoGrafico` e todos os seus atributos representam as propriedades da peça `Iluminação`.

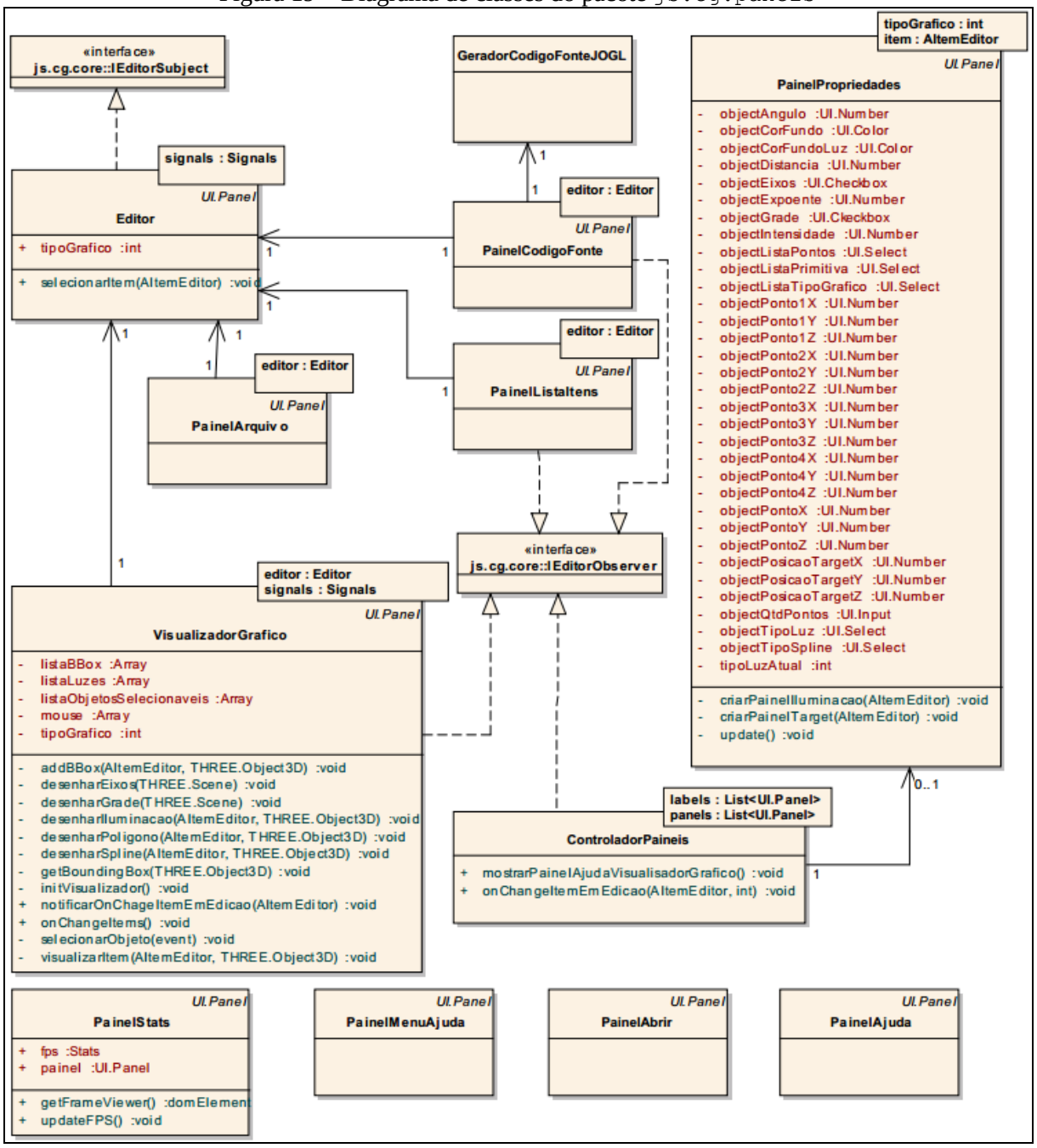
A classe `ItemEditorSpline` representa a peça `Spline` e herda os métodos e atributos da classe abstrata `AItemEditorEncaixeQuadrado`. Ela está associada à classe `ItemEditorObjetoGrafico` e todos os seus atributos representam diretamente as propriedades da peça `Spline`.

Assim como a classe `ItemEditorSpline`, a classe `ItemEditorPoligono` também herda os métodos e atributos da classe abstrata `AItemEditorEncaixeQuadrado` e também está associada à classe `ItemEditorObjetoGrafico`. Essa classe representa a peça `Polígono` e, sendo assim, seus atributos representam diretamente as propriedades dessa peça. Além disso, essa classe possui o método `getListaPontos` que retorna uma lista em formato texto contendo a lista de todos os pontos do `Polígono`.

3.2.2.4 Pacote `js.cg.panels`

Neste pacote estão todas as classes responsáveis pela representação, controle e organização de cada um dos painéis exibidos na aplicação. A Figura 19 ilustra o diagrama de classes desse pacote.

Figura 19 – Diagrama de classes do pacote `js.cg.panels`



Fonte: estendido de Montibeler (2014a, p.48).

Nesse pacote foi incluída a nova classe denominada `PainelStats`. Essa classe é responsável por exibir na interface do sistema um gráfico que é atualizado com o monitoramento de execuções dos FPS da aplicação. A atualização desse gráfico é realizada através do método `updateFPS`, chamado pelo `Renderizador` a cada iteração da renderização. O método `getFrameViewer` retorna o painel que faz a exibição do gráfico. O atributo `fps` é um item da classe `Stats` que pertence a biblioteca `stats.min.js`, que pode ser encontrada no pacote `js.frameworks.build`.

A classe `Editor` recebeu um novo atributo de controle denominado `tipoGrafico` que tem a função de identificar com qual tipo de gráfico o usuário está trabalhando. Esse tipo de gráfico pode ser 2D ou 3D. Quando um item está em edição, a classe `Editor` chama o método `onChangeItemEmEdicao` da classe `ControladorPaineis` passando o item que está em edição e o valor do atributo `tipoGrafico`. A classe `ControladorPaineis` por sua vez, recebe estes parâmetros e instancia a classe `PainelPropriedades` passando esse item que está em edição juntamente com o tipo de gráfico.

A classe `PainelPropriedades` é responsável pela exibição e manipulação das propriedades de todas as peças da `Fábrica de Peças`. Ao ser instanciada, é passado o parâmetro `item`, que é do tipo `AItemEditor`. Através deste parâmetro são identificadas quais propriedades devem ser exibidas para cada peça. Cada atributo da classe corresponde a um campo de propriedade de uma das peças da `Fábrica de Peças`. A exibição desses campos no painel de propriedades está associada ao tipo de peça que o item passado por parâmetro representa, ou seja, o painel propriedades irá exibir as propriedades específicas de cada item, conforme especificado no pacote `js.cg.objects.items` (seção 3.2.2.3).

Quando o item recebido por parâmetro for do tipo `EIdsItens.ILUMINACAO`, além das propriedades `Tipo de luz` e `Posição` que sempre são visíveis nesse item, as demais propriedades são exibidas de acordo com o tipo de luz selecionada. Desta forma, quando o tipo de luz for `ListaTipoLuz.HEMISPHERE`, as propriedades `Intensidade` e `Cor Fundo` são exibidas. Quando o tipo de luz for `ListaTipoLuz.Directional`, além da propriedade `Intensidade`, a propriedade referente a posição (*x, y z*) do alvo da luz também é exibida. Já no tipo de luz `ListaTipoLuz.PointLight`, além da propriedade `Intensidade` é exibida também a propriedade `Distância`. Por fim, no tipo de luz `ListaTipoLuz.SpotLight` além das propriedades `Intensidade` e `Distância`, são exibidas também as propriedades `Ângulo` e `Expoente`, juntamente com as propriedades referente a posição (*x*, *y*, *z*) do alvo da luz. Os métodos `criarPainelIluminacao` e `criarPainelTarget` são utilizados para realizar este controle de visibilidade das propriedades pelo tipo de luz selecionada.

A classe `VisualizadorGrafico` é responsável por criar e manipular o painel `Espaço Gráfico` e `Visão da Câmera`. O método `addBBox` é chamado quando uma peça ou objeto gráfico é selecionado. Ele adiciona ao objeto gráfico selecionado e a seus objetos filhos as respectivas *bouding box*, que é gerada através do método `getBoudingBox`. Cada *boundig box* adicionada ao `Espaço Gráfico` é também incluída no atributo `listaBBox` que possui uma lista com todas as *bouding box* criadas. Está lista serve para manter a referência desses

objetos para que os mesmos sejam excluídos no momento em que a cena é redesenhada e os objetos não estão mais selecionados.

Os métodos `desenharEixos` e `desenharGrade` servem para realizar o desenho dos eixos e da grade respectivamente, que servem de referência no `Espaço Gráfico`. O método `desenharPoligono` implementa o desenho do objeto gráfico que representa a peça `Polígono`, de acordo com as propriedades definidas no `Painel Propriedades`. Da mesma forma, o método `desenharSpline`, realiza o desenho do objeto gráfico que representa a peça `Spline`, de acordo com as suas propriedades definidas no `Painel Propriedades`.

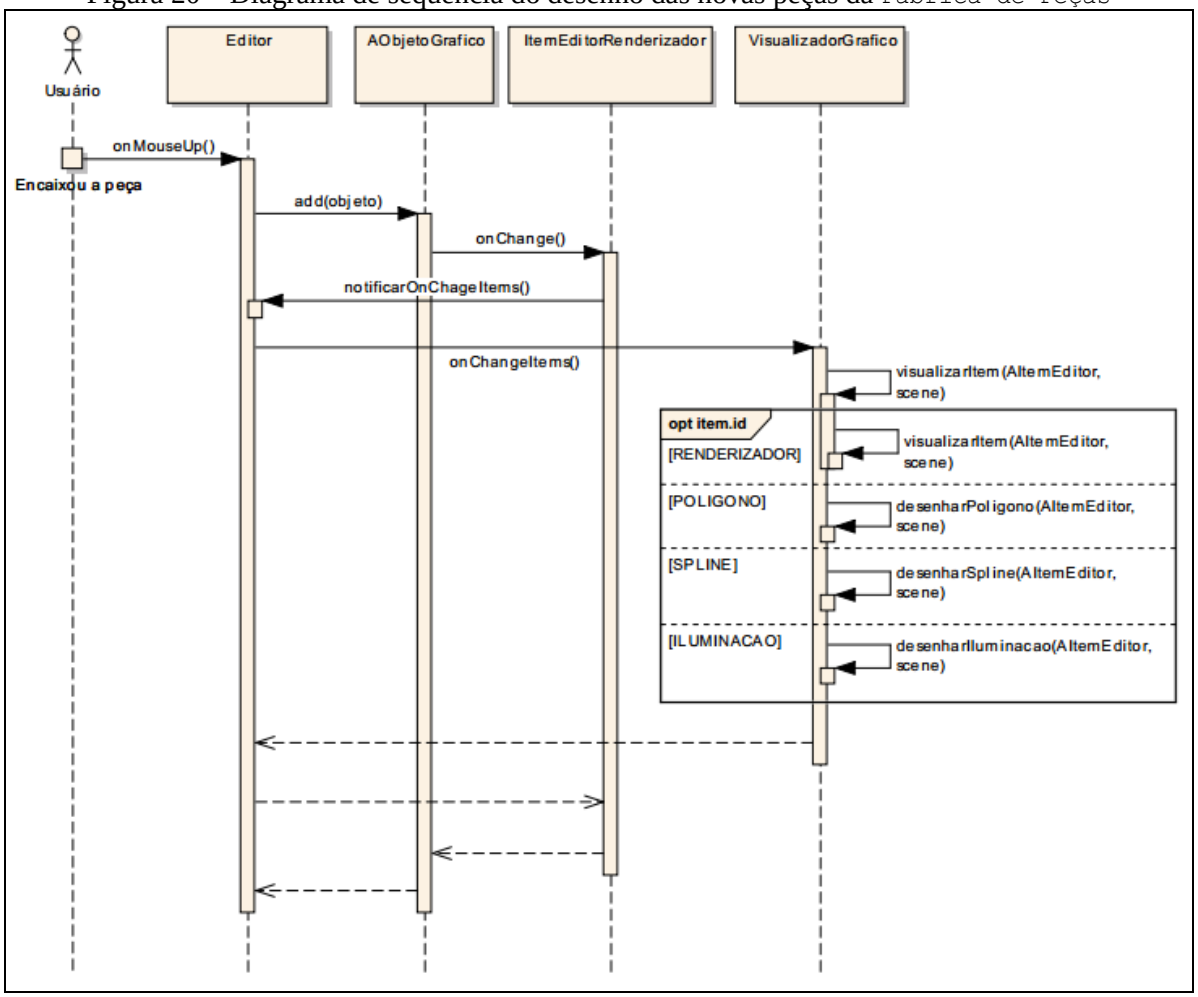
O método `desenharIluminacao` é responsável por desenhar a iluminação da cena. A luz é criada de acordo com a definição do tipo de luz nas propriedades da peça `Iluminação`. Após ser criada, a luz é adicionada ao atributo `listaLuzes` que mantém uma lista de todas as luzes criadas no `Espaço Gráfico`. Está lista serve para manter a referência dessas luzes e quando a visualização da cena é redesenhada, a lista é percorrida e todos os seus itens são excluídos para serem recriados de acordo com as novas definições das propriedades.

O atributo `listaObjetosSelecionaveis` contém uma lista com os objetos criados no `Espaço Gráfico` que podem ser selecionados através do *mouse*, diretamente no `Espaço Gráfico`. Quando os objetos são criados, também são adicionados a esta lista. Essa lista é utilizada no método `selecionarObjeto`, responsável pela seleção dos objetos no `Espaço Gráfico`. Esse método está associado ao evento do *mouse* denominado `ondblclick` e é chamado toda vez que o `usuário` realiza um clique duplo sobre o `Espaço Gráfico`. Nesse método é utilizado o atributo `mouse` para identificar a posição do *mouse* no momento em que é realizado o duplo clique.

3.2.3 Diagrama de sequência

Esta seção apresenta um diagrama de sequência da aplicação VisEdu-CG que ilustra o processo de desenho dos novos objetos gráficos `Polígono`, `Spline` e `Iluminação` no `Espaço Gráfico`. O diagrama de sequência em detalhes está ilustrado na Figura 20.

Figura 20 – Diagrama de sequência do desenho das novas peças da Fábrica de Peças



O fluxo inicia no momento em que o Usuário encaixa a peça (Polígono, Spline ou Iluminação) no Painel de Montagem. Nesse momento, a classe Editor identifica que se trata do encaixe de uma peça e chama o método Add da classe AObjetoGrafico. Esta por sua vez, adiciona o novo objeto em uma lista e notifica as suas classes descendentes que um novo objeto foi adicionado. Esta notificação chega até a classe ItemEditorRenderizador através do método onChange. Para estender a notificação aos demais itens, a classe ItemEditorRenderizador chama o método notificarOnChangeItens da classe Editor. A classe Editor possui uma lista de observadores que são notificados nesse momento. Um destes observadores é a classe VisualizadorGrafico, responsável por desenhar os objetos no Espaço Gráfico. Então a classe Editor chama o método onChangeItems da classe VisualizadorGrafico que, ao executar, chama o seu próprio método visualizarItem passando um objeto do tipo RENDERIZADOR.

No método visualizarItem é verificado o tipo de objeto passado como parâmetro. Quando o tipo do objeto for RENDERIZADOR, é percorrido a lista dos filhos desse objeto,

chamando de forma recursiva o método `visualizarItem` para cada um dos itens filhos. Quando o tipo do objeto for `POLIGONO` o método `desenharPoligono` é chamado e realiza o desenho do `Polígono` de acordo com as suas propriedades. Já se o tipo de objeto for `SPLINE`, é chamado o método `desenharSpline` que realiza o desenho da `Spline`. Por fim, se o tipo de objeto for `ILUMINAÇÃO`, o método `desenharIluminacao` é chamado para realizar o desenho da luz de acordo com as propriedades definidas no painel `Propriedades da Peça`.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentadas ferramentas e técnicas utilizadas na implementação da aplicação e a operacionalidade desta implementação.

3.3.1 Técnicas e ferramentas utilizadas

O desenvolvimento desta aplicação foi realizado utilizando a linguagem de programação JavaScript para implementação das classes e a linguagem HTML5 para implementação da interface web. Também foi utilizada a biblioteca `Three.js` em sua versão 67. O ambiente de programação utilizado foi o Eclipse Java EE IDE for Web Developers, em sua versão Kepler Service Release 1.

Dando início ao desenvolvimento, foi realizado *download* do projeto original (MONTIBELER, 2014b). Em seguida foi criado um novo repositório (NUNES, 2014) hospedado pelo serviço BitBucket. Para manter o repositório atualizado bem como fazer o registro das alterações na codificação, utilizou-se a ferramenta SourceTree na versão 1.5.2.

Os navegadores utilizados nos testes foram Google Chrome versão 35.0.1916.114m, Mozilla Firefox versão 30.0 e Opera 22.0.

3.3.2 O visualizador de material educacional (VisEdu-CG)

Esta seção descreve a implementação do visualizador de material educacional, módulo de computação gráfica, na sua versão 3.0, apresentando as principais rotinas desenvolvidas.

3.3.2.1 Novos itens da `Fábrica de Peças`

Foram disponibilizados três novos itens da `Fábrica de Peças`: `Polígono`, `Spline` e `Iluminação`. Desta forma, foram desenvolvidas as classes `ItemEditorPoligono`, `ItemEditorSpline` e `ItemEditorIluminacao` respectivamente para representação desses itens. As classes `ItemEditorPoligono`, `ItemEditorSpline` foram desenvolvidas seguindo a mesma estrutura da classe `ItemEditorCubo` desenvolvida por Montibeler (2014a), sendo

assim, elas herdam os métodos e atributos da classe `AItemEditorEncaixeQuadrado`. Esta por sua vez, herda o comportamento da classe abstrata `AItemEditor`. Por fim, a classe `AItemEditor` herda o comportamento da classe `AObjetoGrafico`.

O item polígono pode ser desenhado de quatro formas diferentes: somente os vértices, polígono aberto, polígono fechado ou polígono preenchido. Estas formas são denominadas primitivas gráficas e são definidas nas propriedades da peça `Polígono`. No momento em que a classe `VisualizadorGrafico`, responsável por desenhar os objetos no espaço gráfico, desenha o objeto gráfico polígono, a primitiva gráfica é verificada. Para cada primitiva há uma forma diferente de desenho. Sendo assim, quando a primitiva gráfica for `Vértices`, é desenhada uma esfera para representar cada um dos vértices do polígono. Se a primitiva gráfica selecionada for `Aberto`, apenas é desenhada uma linha através do objeto `THREE.Line`, passando por cada um dos pontos do polígono. Para os casos em que a primitiva gráfica selecionada for `Fechado`, o procedimento é o mesmo da primitiva `Aberto`. Apenas é repetido o primeiro vértice na lista de vértices do polígono, fazendo com que o mesmo fique fechado. Nos casos em que a primitiva gráfica selecionada for `Preenchido`, é utilizado o objeto `THREE.Face3` para realizar o preenchimento do polígono. Para isso, é percorrida a lista de vértices, realizando a triangularização do polígono. O Quadro 1 apresenta o trecho do código que desenha o polígono quando a primitiva gráfica `Preenchido` estiver selecionada.

Quadro 1 – Desenho do objeto `Polígono` com a primitiva `Preenchido`

```
...
else if (item.primitiva == CG.listaDePrimitivas.Preenchido) {
	// DESENHA A LINHA ANTES DAS FACES
	var geoLine = new THREE.Geometry();

	geoLine.vertices = points;
	geoLine.vertices.push(item.listaPontos[0]); //REPETE O PRIMEIRO PONTO

	var matLine = new THREE.LineBasicMaterial( { linewidth: 2, color: cor,
transparent: false } );
	var line = new THREE.Line(geoLine, matLine, THREE.LineStrip);
	line.item = item;
	objetoAux.add(line);
	scope.listaObjetosSelecionaveis.push(line);

	var geometria = new THREE.Geometry();
	geometria.vertices = points;

	//ADICIONA AS FACES DOS TRIANGULOS
	for (var i = 0; i < (item.listaPontos.length - 2); i++) {
		geometria.faces.push(new THREE.Face3(0, i + 1, i + 2));
	}

	geometria.computeLineDistances();

	var material = new THREE.MeshPhongMaterial({color:cor, ambient: cor,
overdraw: true, side:THREE.DoubleSide});
	var poligono = new THREE.Mesh(geometria, material);

	poligono.item = item;
	objetoAux.add(poligono);
	scope.listaObjetosSelecionaveis.push(poligono);
}
...
```

Já o item `Spline`, quando selecionado o tipo `ListaTipoSpline.Bezier`, leva em conta a lista de pontos de controle e a quantidade de pontos ao ser desenhado. A lista de pontos de controle, representada pelo atributo `listaPontos` da classe `ItemEditorSpline`, possui quatro pontos. Esses quatro pontos servem de referência para o desenho da curva que é realizado utilizando a quantidade de pontos, representado pelo atributo `qtdPontos` também da classe `ItemEditorSpline`. A partir do ponto inicial da lista de pontos, são definidos os demais pontos com base na quantidade de pontos. Sendo assim, a curva da spline inicia na primeira posição da lista de pontos e termina na última posição da lista de pontos. Os demais pontos são calculados a partir de cada um dos pontos da lista. Após ter os pontos da curva da spline definidos, é desenhada uma linha com o objeto `THREE.Line`, passando por todos esses pontos calculados. O Quadro 2 apresenta o trecho do código que desenha a curva da `Spline`.

Quadro 2 – Desenho do item `Spline` no `Espaço Gráfico`

```
...
var pontosSpline = [];

var t;
var p0, p1, p2, p3;
var x, y, z;

for (var i = 0; i <= item.qtdPontos; i++) {
	t  =  (i / item.qtdPontos);
	p0 = Math.pow((1 - t), 3);
	p1 = (3 * t * Math.pow((1 - t), 2));
	p2 = (3 * Math.pow(t, 2) * (1 - t));
	p3 = Math.pow(t, 3);

	x = (p0 * points[0].x + p1 * points[1].x + p2 * points[2].x + p3 *
points[3].x);
	y = (p0 * points[0].y + p1 * points[1].y + p2 * points[2].y + p3 *
points[3].y);
	z = (p0 * points[0].z + p1 * points[1].z + p2 * points[2].z + p3 *
points[3].z);

	pontosSpline.push(new THREE.Vector3(x, y, z));
}

var geometria = new THREE.Geometry();
geometria.vertices = pontosSpline;
geometria.computeLineDistances();

var material  = new THREE.LineBasicMaterial({ linewidth: 2, color: cor,
transparent: false });
var spline    = new THREE.Line(geometria, material, THREE.LineStrip);

objetoAux.add(spline);
spline.item = item;
scope.listaObjetosSelecionaveis.push(spline);

spline.add(addBBox(item, spline));
...
```

Por fim, o item `Iluminação`, ao ser desenhado verifica qual o tipo de luz selecionado. Caso o tipo de luz selecionado for `Ambient`, é instanciado um objeto do tipo `THREE.AmbientLight` passando a cor da luz como parâmetro. Se o tipo de luz selecionado for `Hemisphere`, é instanciado um objeto do tipo `THREE.HemisphereLight` passando como parâmetro as cores do céu e do fundo, juntamente com a intensidade da luz. Quando o tipo de luz selecionada for `Directional`, é instanciado um objeto do tipo `THREE.DirectionalLight`, informando como parâmetro a cor da luz e a intensidade. No tipo de luz `PointLight`, é instanciado um objeto do tipo `THREE.PointLight`, passando como parâmetro a cor da luz, juntamente com a intensidade e distância da luz. Por fim, quando o tipo de luz selecionada for `SpotLight`, é instanciado um objeto do tipo `THREE.SpotLight`, passando como parâmetro a cor, a intensidade e a distância da luz, juntamente com o ângulo e expoente.

Ainda quando o tipo de luz for `Directional` ou `SpotLight`, é informada a posição (*x*, *y*, *z*) do alvo da luz. Esta informação é atribuída à propriedade `target` da luz. Além disso, para todos os tipos de luz (com exceção da luz `Ambient`), é instanciado um objeto `Helper`, que é a representação gráfica da luz no `Espaço Gráfico` que auxilia o `Usuário` a visualizar algumas propriedades da luz, como a posição e a direção. O Quadro 3 apresenta o trecho do código que desenha a o tipo de luz `SpotLight`.

Quadro 3 – Desenho da luz `SpotLight`

```
...
else if (item.tipoLuz == CG.listaTipoLuz.SpotLight) {
	light = new THREE.SpotLight(cor, item.intensidade, item.distancia,
item.angulo, item.expoente);
	light.position.set( item.posicao.x, item.posicao.y, item.posicao.z );

	light.target.position.set(item.posicaoTarget.x, item.posicaoTarget.y,
item.posicaoTarget.z);
	light.target.matrixWorld.elements[ 12 ] = item.posicaoTarget.x;
	light.target.matrixWorld.elements[ 13 ] = item.posicaoTarget.y;
	light.target.matrixWorld.elements[ 14 ] = item.posicaoTarget.z;

	helper = new THREE.SpotLightHelper(light, 10);
}

if (light !== undefined) {
	objetoAux.add(light);
	listaLuzes.push(light);
}

if (helper !== undefined) {
	sceneHelper.add(helper);
	helper.update();
	listaLuzesHelpers.push(helper);
}
...
```

As classes `ExportadorJSON` e `LeitorJSON` são responsáveis pela exportação e importação da cena gráfica no formato JSON. Nessas classes, foram a exportação e importação respectivamente dos novos itens `Polígono`, `Spline` e `Iluminação`, juntamente com suas propriedades.

3.3.2.2 Seleção de objetos no `Espaço Gráfico`

A seleção dos objetos no `Espaço Gráfico` foi desenvolvida no método `SelecionarObjeto` da classe `VisualizadorGrafico`. Esse método é executado quando ocorre o evento de duplo clique do *mouse* no `Espaço Gráfico`. Ao ser executado, o método `SelecionarObjeto` calcula as coordenadas da posição (*x*, *y*) do *mouse* na tela. No entanto, estas coordenadas referem-se à posição no espaço 2D da tela e, portanto, é necessário realizar a conversão das mesmas para o espaço gráfico 3D. Para isso, é utilizado o método

`unprojectVector` da classe `TREE.Projector`, informando como parâmetro a posição do *mouse* na tela e a câmera responsável pela visualização do `Espaço Gráfico`.

A identificação dos objetos selecionados é realizada através do algoritmo *raycasting*, disponibilizado na classe `THREE.Raycaster`. Após ser instanciado, é chamado o método `set` que atualiza a origem e a direção do raio do *raycasting*. Para executar o algoritmo, é chamado o método `intersectObjects`, informando como parâmetro o atributo `listaObjetosSelecionaveis` que é a lista de objetos da cena que podem ser selecionados. Estes objetos (`Cubo`, `Polígono` e `Spline`) são adicionados a essa lista no momento em que são incluídos no `Espaço Gráfico`. Após a execução, o método `intersectObjects` retorna a lista de objetos atingidos pelo raio, ordenados pela sequência em que foram atingidos.

Por fim, quando a lista retornada pelo método `intersectObjects` não estiver vazia, é chamado o método `selecionarItem` da classe `Editor`, passando o primeiro item da lista. Com isso, além da seleção do objeto no `Espaço gráfico`, é realizada também a seleção da peça correspondente na `Fábrica de Peças`. O Quadro 4 apresenta o código que realiza a seleção dos objetos no `Espaço Gráfico`.

Quadro 4 – Seleção dos objetos no `Espaço Gráfico`

```
1  function selecionarObjeto(event) {
2        mouse.x = ((event.clientX - scope.dom.offsetTop) /
3  (renderer.domElement.width)) * 2 - 1;
4        mouse.y = -((event.clientY - scope.dom.offsetTop) /
5  renderer.domElement.height) * 2 + 1;
6
7        var vetor    = new THREE.Vector3(mouse.x, mouse.y, 0);
8        var projetor = new THREE.Projector();
9        projetor.unprojectVector(vetor, views[0].camera);
10
11       var raycaster = new THREE.Raycaster();
12       raycaster.set(views[0].camera.position,
13 vetor.sub(views[0].camera.position).normalize());
14
15       var intersects =
16 raycaster.intersectObjects(scope.listaObjetosSelecionaveis);
17
18
19       if (intersects.length > 0) {
20             editor.selecionarItem(intersects[0].object.item);
21       }
22       else {
23             editor.selecionarItem(null);
24
25             scope.listaObjetosSelecionaveis = [];
26             visualizarItem(scope.editor.painelMontagem, scene);
27       }
28 };
```

3.3.2.3 Visualização do `Espaço Gráfico` em 2D

A visualização do `Espaço Gráfico` em 2D foi desenvolvida em duas classes: `PainelPropriedades` e `VisualizadorGrafico`. A classe `PainelPropriedades` é responsável por atribuir zero nas propriedades dos objetos que se referem ao eixo `z` do espaço gráfico. Ao realizar o desenho destes objetos no `Espaço Gráfico`, o método `visualizarItem` da classe `VisualizadorGrafico`, realiza o posicionamento fixo da câmera do `Espaço Gráfico` nas posições *x* e *y* igual a zero e *z* igual a 1800. Para impossibilitar que esta posição seja alterada, os controles da cena que são manipulados pelo *mouse* são desabilitados. Com isso, apesar da aplicação continuar a trabalhar no espaço gráfico 3D, simula para o usuário o espaço gráfico 2D. O Quadro 5 apresenta o trecho do código da classe `VisualizadorGrafico`, que altera a câmera quando o tipo de gráfico é 2D.

Quadro 5 – Alteração da posição da câmera quando tipo de gráfico é 2D

```
...
tipoGrafico = item.tipoGrafico;

if (tipoGrafico == 2) {
	views[0].camera.position.x = 0;
	views[0].camera.position.y = 0;
	views[0].camera.position.z = 1800;

	habilitarControls(false);
	item.verGrade = false;
}
else if (tipoGrafico == 3) {
	habilitarControls(true);
}
...
```

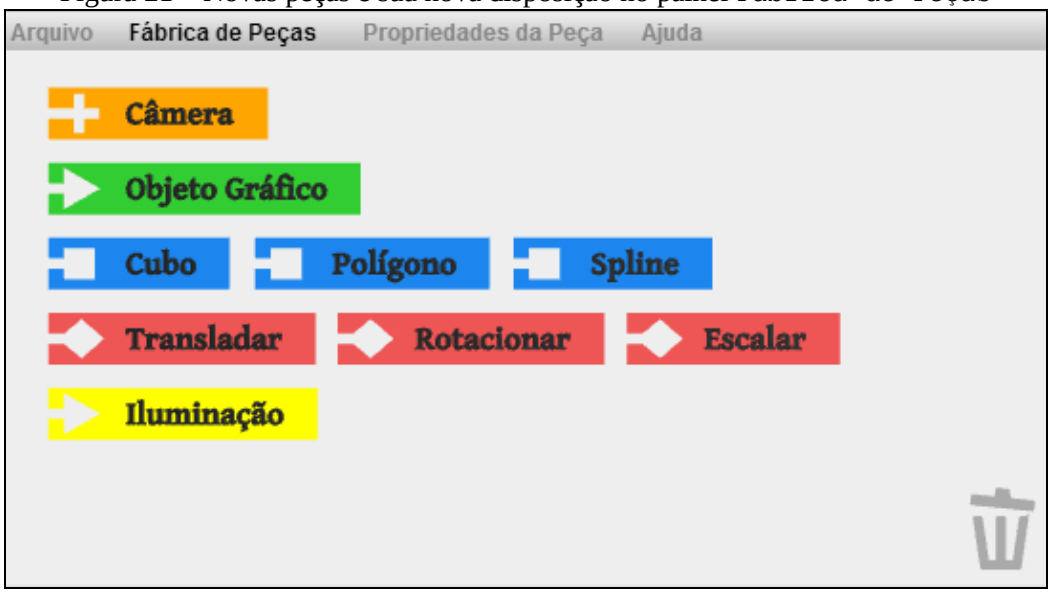
3.3.3 Operacionalidade da implementação

As seções a seguir descrevem os detalhes e funcionalidades da operacionalidade da aplicação VisEdu-CG, focando nas novas funcionalidades implementadas na versão 3.0.

3.3.3.1 Painel `Fábrica de Peças`

Todos os painéis desenvolvidos por Montibeler (2014a) foram mantidos. O painel `Fábrica de Peças` recebeu as novas peças `Polígono`, `Spline` e `Iluminação`. A fim de otimizar o espaço do painel, a forma como essas peças estão agrupadas foi alterada para que fiquem agrupadas por suas cores, que identificam os seus tipos. A Figura 21 ilustra o painel `Fábrica de Peças` com as novas peças e o novo agrupamento das mesmas.

Figura 21 – Novas peças e sua nova disposição no painel `Fábrica de Peças`

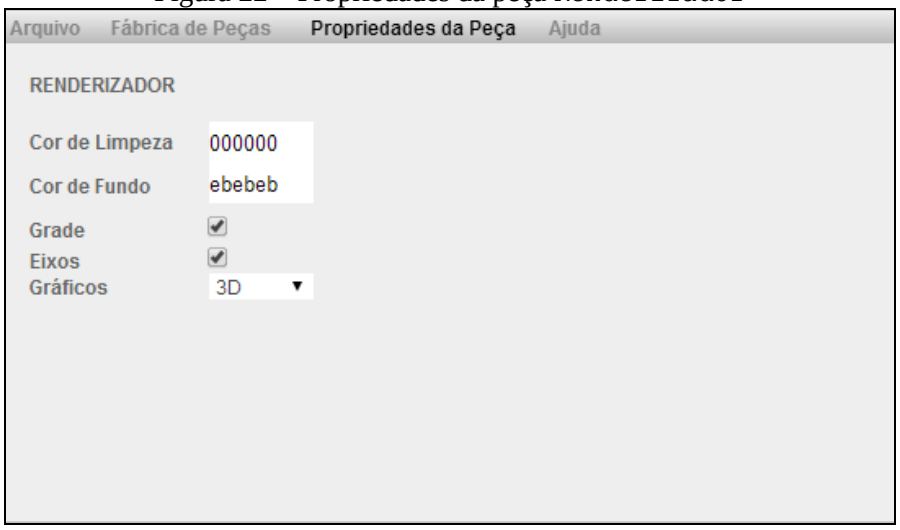


3.3.3.2 `Renderizador`

No item `Renderizador` são encaixadas todas as demais peças da `Fábrica de Peças`. Quando selecionado, o painel `Propriedades da Peça` é habilitado e nele é possível alterar

propriedades que são gerais a toda a aplicação. Estas propriedades estão ilustradas na Figura 22.

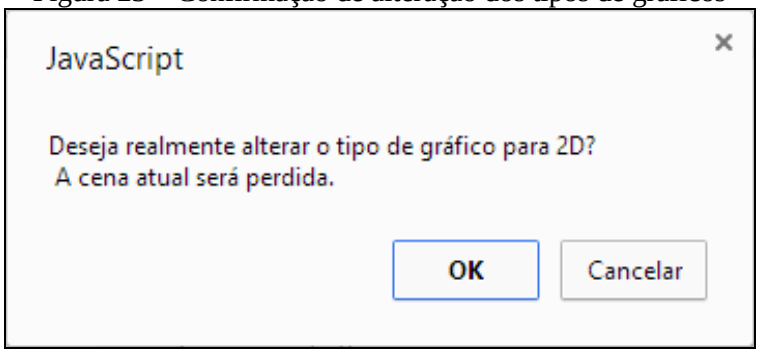
Figura 22 – Propriedades da peça `Renderizador`



Através dessas propriedades é possível habilitar ou desabilitar a visualização da `Grade` e dos `Eixos` do `Espaço Gráfico` e também alterar a cor de fundo. Na propriedade `Gráficos` o usuário pode definir se deseja trabalhar com gráficos em 2D ou 3D.

Quando a propriedade `Gráficos` é alterada, a aplicação emite uma mensagem solicitando a confirmação desta alteração. Caso o usuário confirmar, a cena atual é perdida e os gráficos são alterados conforme selecionado pelo `Usuário`. Caso o `Usuário` cancele a alteração, o tipo de gráfico não é alterado. Esta mensagem de confirmação é ilustrada na Figura 23.

Figura 23 – Confirmação de alteração dos tipos de gráficos

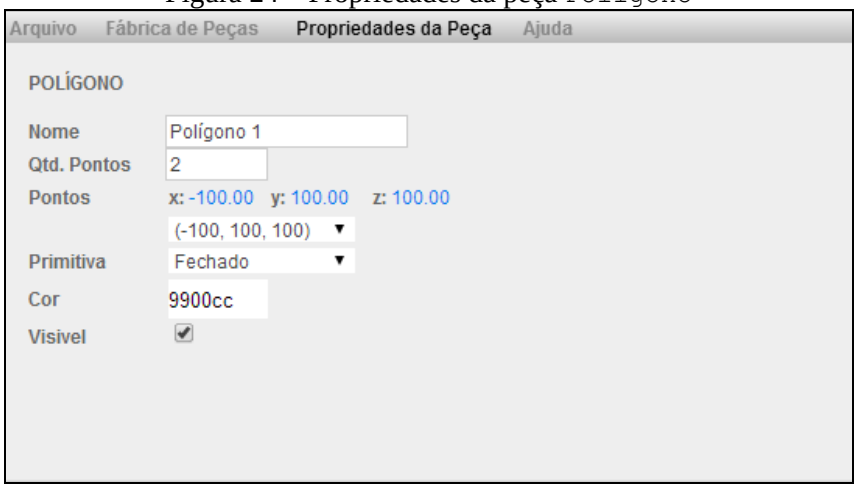


3.3.3.3 Peça `Polígono`

A peça `Polígono` representa um objeto gráfico geométrico no `Espaço Gráfico`. Ela possui o encaixe `Quadrado` e, por este motivo, somente pode ser incluída como uma peça filha da peça `Objeto Gráfico`. Ao incluir esta peça no `Painel de Montagem`, é desenhado no `Espaço Gráfico` por padrão um polígono com dois pontos e com a primitiva gráfica

Vértices. Quando esta peça é selecionada, o painel Propriedades da Peça é habilitado e o Usuário tem acesso às suas propriedades. A Figura 24 ilustra as propriedades da peça Polígono.

Figura 24 – Propriedades da peça Polígono

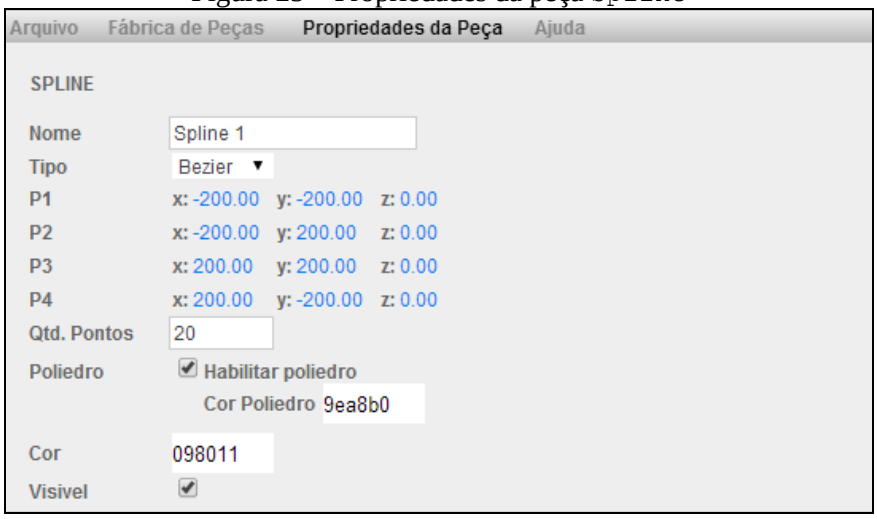


Nas propriedades da peça Polígono é possível definir um nome para peça e quantidade de pontos que o polígono possui. Para incluir novos pontos no polígono, o Usuário deve informar uma quantidade de pontos maior que a existente. Já para excluir os pontos, deve ser informada uma quantidade de pontos menor que a existente no polígono. A inclusão dos novos pontos ocorre sempre na posição 0 (*x*, *y*, *z*) e a exclusão ocorre de acordo com a ordem de inclusão, sendo que o último que foi incluído será o primeiro a ser excluído. Na propriedade Pontos é possível selecionar um determinado ponto do Polígono e alterar a sua posição no Espaço Gráfico e na propriedade Primitiva é possível determinar com qual primitiva gráfica o Polígono será desenhado. Já as propriedades Cor e Visível definem a cor do objeto gráfico e se o mesmo estará visível no Espaço Gráfico.

3.3.3.4 Peça Spline

A peça Spline corresponde a um objeto gráfico no Espaço Gráfico que desenha uma curva de Bézier. Assim como a peça Polígono, esta peça também possui o encaixe Quadrado e somente pode ser incluída como uma peça filha da peça Objeto Gráfico. Quando incluída no Painel de Montagem, a spline é desenhada inicialmente com vinte pontos. Por padrão, juntamente com a spline é desenhado o poliedro de controle. Ao selecionar a spline, o painel Propriedades da Peça é habilitado possibilitando ao usuário manipular as suas propriedades como pode ser visto na Figura 25 que ilustra todas as propriedades da peça Spline.

Figura 25 – Propriedades da peça `Spline`



Nas propriedades da `Spline` é possível definir um nome para a peça, a cor do objeto gráfico e se o mesmo estará visível no `Espaço Gráfico`. Além disso, é possível definir ainda os valores das coordenadas dos pontos de controle, a quantidade de pontos e também alterar a cor do poliedro de controle e habilitar ou desabilitar o mesmo.

3.3.3.5 Peça `Iluminação`

A peça `Iluminação` é responsável pela representação da iluminação da cena no `Espaço Gráfico` e possui uma grande importância na construção da cena gráfica pois, sem ela, não é possível visualizar os objetos no painel `Visão da Câmera`. Diferentemente das peças `Polígono` e `Spline`, essa peça possui o encaixe `Seta`, podendo ser encaixada em dois lugares: diretamente no `Renderizador` ou como uma peça filha da peça `Objeto Gráfico`.

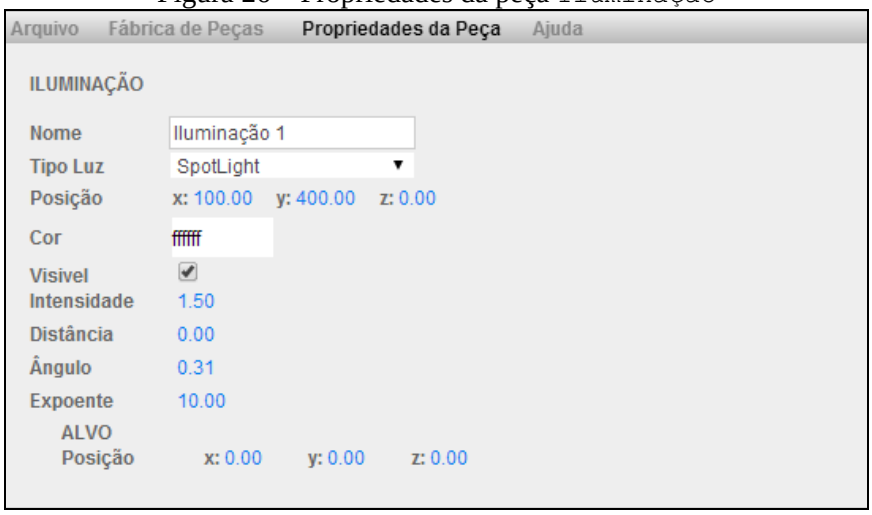
Quando encaixada no `Renderizador`, a iluminação da cena não está associada diretamente a nenhum objeto gráfico específico, mas sim a toda a cena gráfica. Desta forma, quando os objetos gráficos da cena sofrem alterações como translação, escala e/ou rotação, a iluminação da cena não é afetada por estas transformações.

Já quando a peça é encaixada como uma peça filha da peça `Objeto Gráfico`, a iluminação da cena é associada ao respectivo objeto gráfico. Apesar da luz continuar iluminando todos os objetos da cena, se o objeto gráfico associado sofre alguma alteração como translação, escala e/ou rotação, a iluminação também é alterada por esta transformação.

Quando a peça `Iluminação` é adicionada ao `Painel de Montagem` a luz é desenhada na cena gráfica. Esta luz pode ser de cinco diferentes tipos: `Ambient`, `Hemisphere`, `Directional`, `PointLight` ou `SpotLight`. Por padrão, quando a iluminação é criada, o tipo de luz é `Ambient`. Com exceção do tipo de luz `Ambient`, os demais tipos de luzes, quando

criados, desenham juntamente com a luz um objeto gráfico que representa fisicamente as características da luz na cena. Estes objetos são chamados de *Helper* e somente são visíveis no `Espaço Gráfico`, não podendo ser selecionados nem visualizados pela `Visão da Câmera`. Conforme as propriedades da peça `Iluminação` são alteradas, esses objetos, assim como a luz da cena, são atualizados. Além disso, cada tipo de luz possui propriedades particulares que podem ser alteradas quando a peça `Iluminação` é selecionada. As propriedades da luz `SpotLight` são ilustradas na Figura 26.

Figura 26 – Propriedades da peça `Iluminação`



Quando o tipo de luz for `SpotLight`, em suas propriedades é possível definir um nome para a peça, a cor da luz e se a mesma estará visível no `Espaço Gráfico`. Além disso, é possível determinar a intensidade da luz, o ângulo, o expoente e a distância. O ângulo define a abertura que o cone da luz `SpotLight` terá e o expoente define o degradê que acontecerá da sombra para o centro desta luz. Já a distância quando informado zero define que a intensidade da luz continuará constante independentemente da posição da luz. Por último, é possível determinar a posição do alvo da luz.

3.4 RESULTADOS E DISCUSSÃO

Este trabalho apresenta uma aplicação *web* para visualização de material educacional, na qual é possível trabalhar com alguns conceitos de computação gráfica através de um jogo de encaixe de formas geométricas. A fim de apurar com mais precisão os resultados da aplicação foram realizados testes de desempenho e operacionalidade. Os resultados obtidos em cada um dos testes são apresentados nas seções seguintes.

3.4.1 Desempenho

A avaliação do desempenho da aplicação foi realizada adicionando os objetos gráficos no `Espaço Gráfico`. Os testes foram realizados em três navegadores diferentes: Google Chrome, Mozilla Firefox e Opera (descritos na seção 3.3.1). O Internet Explorer não foi utilizado nos testes devido a incompatibilidades encontradas na utilização do `Three.js`.

Os critérios utilizados para testar o desempenho da aplicação foram o consumo de memória durante a criação dos objetos e o processamento de FPS em relação a quantidade de `Peças` adicionadas no `Painel de Montagem`. A ferramenta utilizada para fazer o levantamento das informações de consumo de memória dos três navegadores foi o gerenciador de tarefas do *Windows* e para o monitoramento do processamento de FPS, em todos os navegadores foram utilizados os dados gerados pela biblioteca `Stats`, que é uma biblioteca auxiliar do `Three.js`.

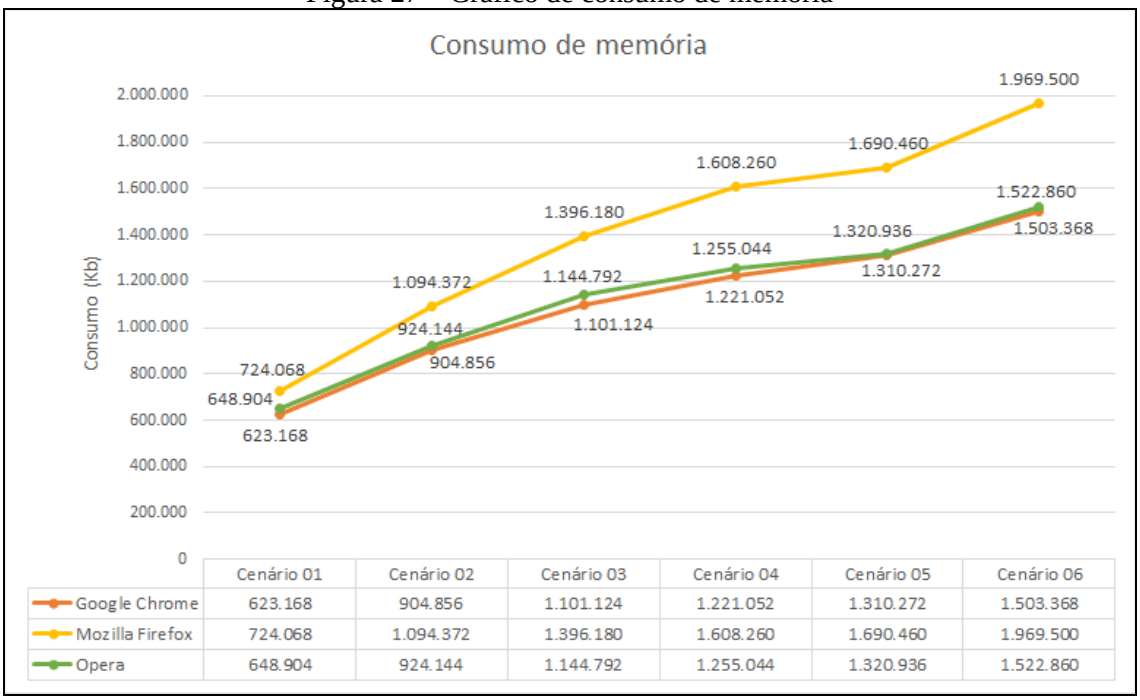
Foram criados ao todo seis cenários de testes, incluindo, além da `Peça Renderizador`, as peças `Câmera`, `Iluminação`, `Objeto Gráfico`, `Transladar`, `Cubo`, `Polígono` e `Spline`, variando a quantidade de cada `Peça` em cada cenário de teste. O Quadro 6 descreve as quantidades de Peças em cada um dos cenários de testes.

Quadro 6 – Cenário de testes

Peça	Cenário 01	Cenário 02	Cenário 03	Cenário 04	Cenário 05	Cenário 06
`Renderizador`	1	1	1	1	1	1
`Câmera`	1	1	1	1	1	1
`Iluminação`	1	1	1	1	1	2
`Objeto Gráfico`	3	6	9	11	12	15
`Transladar`	0	6	9	11	12	15
`Cubo`	1	2	3	3	4	5
`Polígono`	1	2	3	4	4	5
`Spline`	1	2	3	4	4	5
Total	9	21	30	36	39	49

Cada um dos seis cenários de testes foi criado e exportado em formato JSON pela própria aplicação. Após isto, em cada um dos navegadores foi importado cada um dos cenários. Depois de carregado o arquivo, foram realizadas as verificações de consumo de memória e de processamento de FPS. A Figura 27 apresenta o gráfico de consumo de memória dos navegadores nos cenários de testes.

Figura 27 – Gráfico de consumo de memória

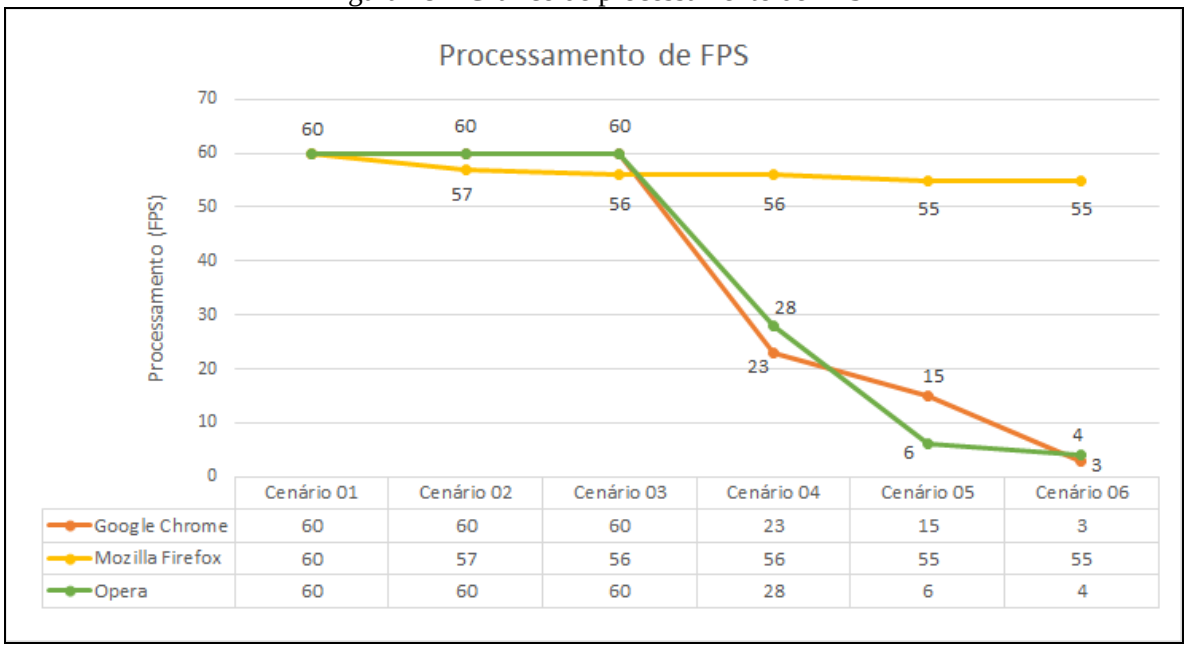


Com relação ao consumo de memória, observou-se que todos os três navegadores tiverem um consumo crescente de memória, conforme foi aumentando a quantidade de peças. O navegador Mozilla Firefox apresentou um consumo de memória acima dos demais navegadores em todos os cenários de testes. Os navegadores Google Chrome e Opera tiveram um desempenho muito similar neste quesito, com o consumo de memória um pouco abaixo do que consumiu o Mozilla Firefox. Dos três navegadores testados, o Goole Chrome foi o que apresentou o melhor desempenho no quesito consumo de memória, sendo o navegador que obteve o menor consumo em todos os cenários de testes, apesar da pequena diferença do navegador Opera.

Em relação ao processamento de FPS, os navegadores Google Chrome e Opera novamente tiveram um resultado similar. Ambos, até o terceiro cenário de teste mantiveram o processamento no máximo (60 FPS). Porém a partir do quarto cenário de testes, tiveram uma grande queda no processamento, chegando no sexto cenário com um processamento tão abaixo que acabam tornando a aplicação quase impossível de ser utilizada. O navegador Mozilla Firefox, demonstrou uma pequena baixa já no segundo cenário de teste. Está queda no processamento continuou em todos os casos de testes, porém a diminuição no processamento foi pequena, chegando no último caso de testes com apenas 5 FPS abaixo do processamento máximo. Baseado nisto, verificou-se que neste critério o navegador Mozilla Firefox apresentou o melhor desempenho, continuando a processar um bom número de FPS,

apesar da quantidade de peças. A Figura 28 apresenta o gráfico de processamento de FPS dos navegadores nos cenários de testes.

Figura 28 – Gráfico de processamento de FPS



Tendo em vista os critérios de consumo de memória e processamento de FPS, observou-se que com relação ao desempenho, o navegador Mozilla Firefox apresentou o melhor resultado, possibilitando a utilização da aplicação mesmo no cenário mais complexo sem nenhum problema de desempenho, apesar do alto consumo de memória verificado no primeiro critério.

3.4.2 Operacionalidade

A fim de avaliar a operacionalidade da aplicação, foi desenvolvida uma atividade com a turma de Computação Gráfica da Fundação Universidade Regional de Blumenau (FURB) em conjunto com o professor da disciplina. Trata-se de uma lista de exercícios contendo 5 questões que os alunos deveriam fazer utilizando a aplicação. A lista de exercícios pode ser encontrada no Apêndice B. Como tratam-se de alunos que estão finalizando a disciplina, todos possuem conhecimento dos conceitos de computação gráfica abordados pela aplicação.

Após a execução das questões do exercício, os alunos foram convidados a responder um questionário com perguntas sobre a experiência de uso da aplicação. O Quadro 7 descreve as questões contidas no questionário de operacionalidade.

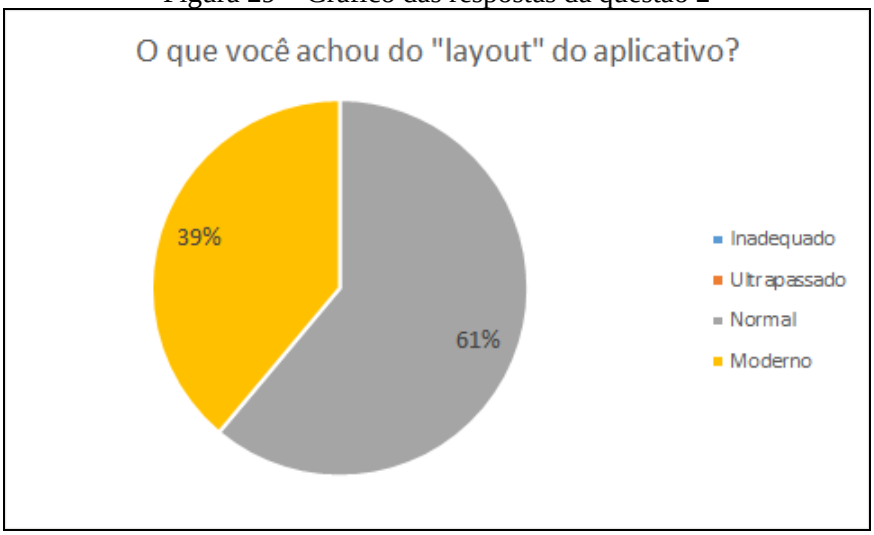
Quadro 7 – Questionário de operacionalidade

Questão	**Opções**			
Qual o seu nível de conhecimento sobre conceitos de "Computação Gráfica"?	Nenhum	Ruim	Bom	Ótimo
O que você achou do "layout" do aplicativo?	Inadequado	Ultrapassado	Normal	Moderno
O que você achou da "operacionalidade" do aplicativo?	Difícil, nada intuitivo	Mediano, mas pouco intuitivo	Mediano, mas intuitivo	Fácil e intuitivo
O que você achou do tamanho e disponibilidade das "janelas"?	Péssimo	Ruim	Bom	Ótimo
O aplicativo ajudou a entender a aplicação de como usar os objetos gráficos "Cubo", "Polígono" e "Spline"?	Não ajudou	Ajudou pouco	Ajudou	Ajudou muito
O aplicativo ajudou a entender a aplicação de "iluminação" nos objetos gráficos?	Não ajudou	Ajudou pouco	Ajudou	Ajudou muito
O aplicativo possibilitou entender em 2D os mesmos conceitos gráficos apresentados no espaço 3D?	Não ajudou	Ajudou pouco	Ajudou	Ajudou muito
Com base nas respostas anteriores, quais seriam os principais "Pontos Positivos" no uso do aplicativo?	Descritiva			
Com base nas respostas anteriores, quais seriam os principais "Pontos Negativos" no uso do aplicativo?	Descritiva			
Com base nas respostas anteriores, quais seriam as suas "Sugestões" de melhoria para o aplicativo?	Descritiva			

Ao todo, foram obtidas as respostas de 18 alunos que responderam ao questionário de operacionalidade e realizaram a atividade na aplicação. Na primeira questão foi solicitado aos alunos qual o seu nível de conhecimento nos conceitos de computação gráfica e 67% dos alunos responderam que possuem um bom nível de conhecimento dos conceitos de computação gráfica. Dos restantes, 28% responderam que possuem um conhecimento ruim e 5% responderam que possuem um conhecimento ótimo.

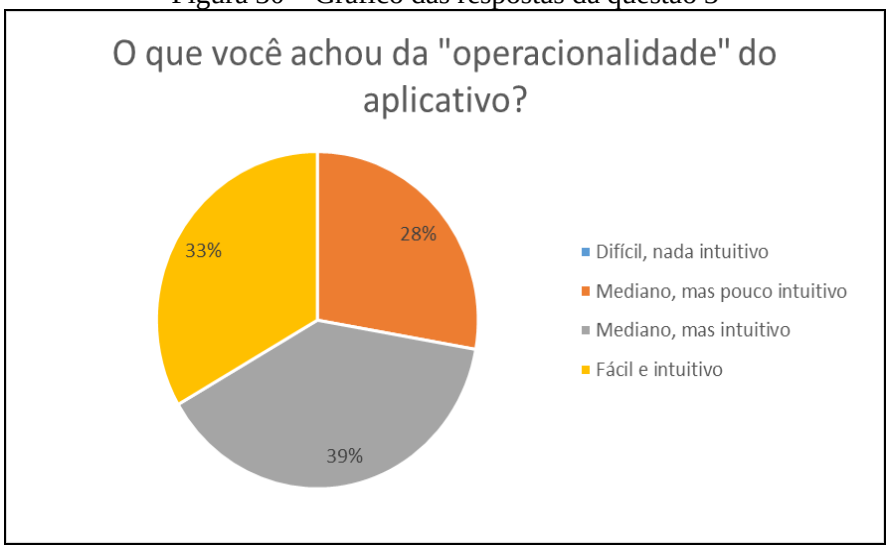
Com relação ao *layout* do aplicativo, a maioria dos alunos respondeu que considera normal e 39% dos alunos responderam que consideram moderno, como pode ser visto no gráfico apresentado na Figura 29.

Figura 29 – Gráfico das respostas da questão 2



Tratando-se da operacionalidade do aplicativo, a maioria dos alunos considerou mediana, porém intuitiva (39%) ou fácil e intuitiva (33%). Apenas 28% dos alunos consideram a operacionalidade mediana e pouco intuitiva, como pode ser verificado no gráfico apresentado na Figura 30.

Figura 30 – Gráfico das respostas da questão 3

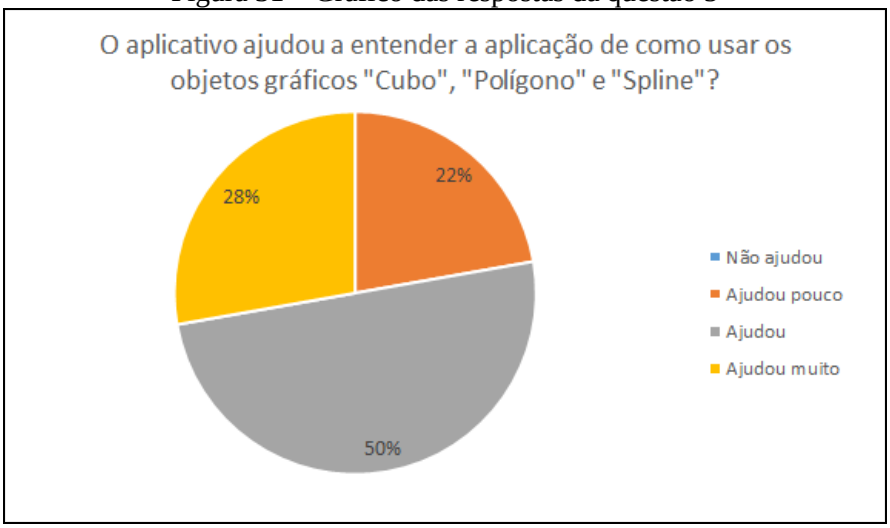


Os alunos classificaram o aplicativo com a operacionalidade mediana e pouco intuitiva por questões relacionadas à interface da aplicação, refletindo também nas respostas descritivas em que os foram apontadas como pontos negativos e sugestão de melhoria, sugerindo na interface painéis dinâmicos e com possibilidade de ocultar painéis menos importantes.

Quanto ao tamanho e disponibilidade das janelas, a grande maioria dos alunos (72%) considerou boa, seguido por 17% dos alunos que avaliaram como ótima. Ao todo, apenas 11% dos alunos julgaram ruim o tamanho e disponibilidade das janelas.

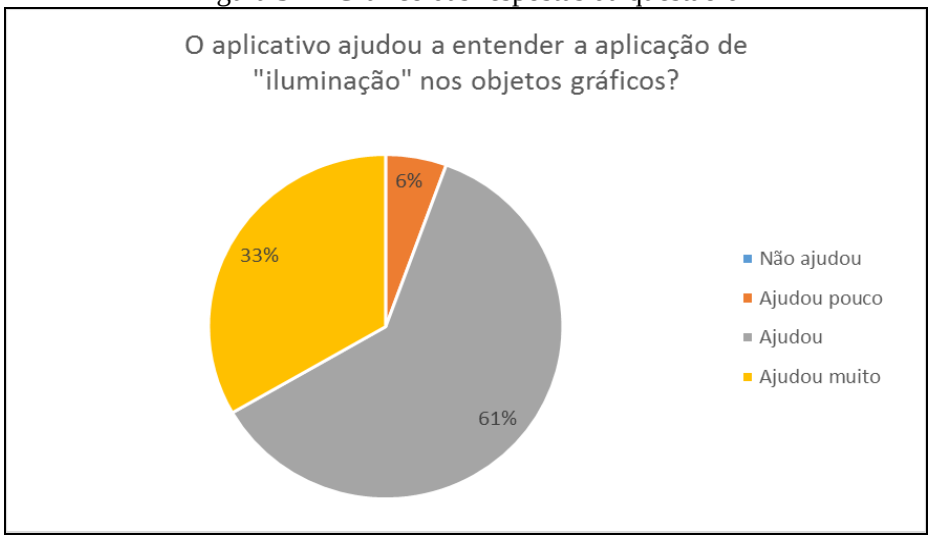
Todos os alunos responderam que o aplicativo ajudou a entender a aplicação de como usar os objetos gráficos `Cubo`, `Polígono` e `Spline`, sendo que 28% dos alunos avaliaram que ajudou muito e apenas 22% dos alunos acreditam que ajudou pouco. O gráfico das respostas desta questão é apresentado na Figura 31.

Figura 31 – Gráfico das respostas da questão 5



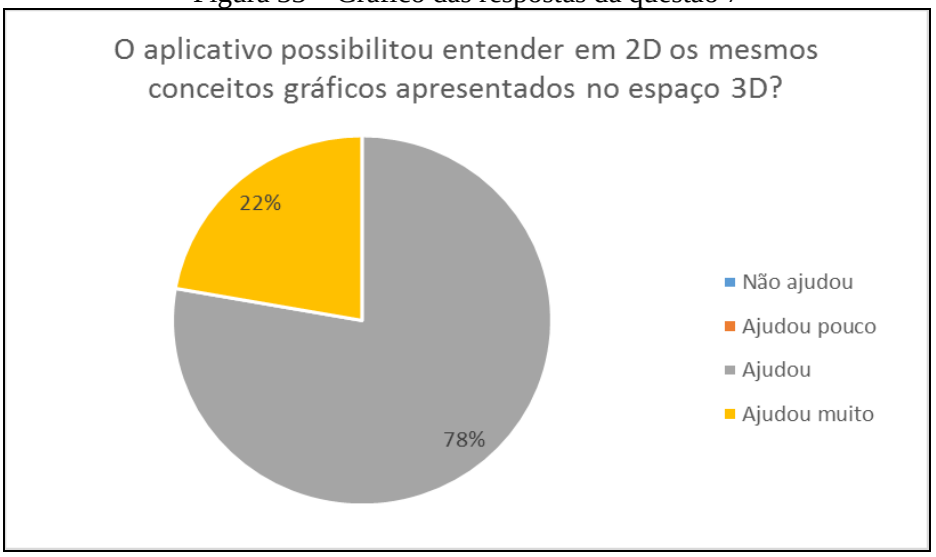
Com relação a aplicação da `Iluminação` nos objetos gráficos, novamente todos os alunos concordaram que a aplicação ajudou, sendo que 33% dos alunos avaliaram que ajudou muito e apenas 6% dos alunos julgaram que a aplicação ajudou pouco, conforme pode ser visto no gráfico apresentado na Figura 32.

Figura 32 – Gráfico das respostas da questão 6



Por fim, tratando-se do entendimento em 2D dos conceitos apresentados em 3D, mais uma vez todos os alunos concordaram que a aplicação ajudou, sendo que 22% dos alunos acreditam que ajudou muito. O gráfico das respostas desta questão é apresentado na Figura 33.

Figura 33 – Gráfico das respostas da questão 7



As respostas das questões descritivas estão disponibilizadas no Apêndice C. Na questão referente aos pontos positivos da utilização do aplicativo, a maior parte dos alunos destacou a simplicidade e facilidade do uso da ferramenta, como também a grande ajuda que proporcionou no entendimento dos conceitos de computação gráfica.

Com relação aos pontos negativos, 28% dos alunos responderam que não encontraram nenhum. Alguns alunos destacaram a dificuldade de modificar os valores dos pontos do `Polígono` e `Spline` pois acharam difícil identificar qual ponto estava sendo alterado. Foi adicionada uma extensão (seção 4.1) para solucionar este problema.

Por fim, com relação à questão que trata das sugestões de melhorias, alguns alunos sugeriram melhorar a disposição dos painéis, ocultando os que são menos importantes como a lista de peças. Outro aluno sugeriu a disponibilização de um tutorial inicial para o `Usuário` e alguns alunos ainda sugeriram a correção de alguns erros que ocorreram enquanto estavam utilizando, como é o caso da exportação que não estava funcionando no navegador Mozilla Firefox. Esta última sugestão foi corrigida testando-se vários cenários simulados para tentar reproduzir o mesmo erro. No caso foi corrigida a exportação no navegador Mozilla Firefox que não estava realizando o *download* automático do arquivo exportado. Para solucionar o problema, o conteúdo do arquivo exportado passa a ser disponibilizando em uma nova aba, possibilitando ao usuário salvar o arquivo.

3.4.3 Relação dos trabalhos correlatos com o presente trabalho

A comparação entre as principais características do presente trabalho com as características dos trabalhos correlatos é apresentada no Quadro 8.

Quadro 8 – Comparação com os trabalhos correlatos

Característica	StarLogo TNG (STEP, 2013)	Motor de jogos 3D (PEREIRA, 2012)	VisEdu-CG 3.0
Diversos tipos de objetos gráficos	X	X	X
Iluminação de cena	-	X	X
Animação da cena	X	-	-
Seleção de objetos no espaço gráfico	X	-	X
Exportação/Importação em formato de arquivo conhecido	X	-	X
Disponibilidade na web	-	X	X
Janelas e painéis dinâmicos	X	-	-
Tipo de cenário	3D	3D	2D/3D

Pode-se observar que a aplicação desenvolvida não se enquadra em apenas dois itens: janelas e painéis dinâmicos e no item animação. Ambos estão presentes apenas no StarLogo TNG pois trata-se de uma ferramenta mais robusta que as demais. Estes itens também foram incluídos na lista de sugestões de extensões do presente trabalho (seção 4.1). Destaca-se o presente trabalho na disponibilização de diversos tipos de objetos gráficos e iluminação, tanto no cenário 2D como também no 3D.

3.4.4 Análise dos Resultados

Com relação ao objeto gráfico `Polígono` foi encontrada uma limitação no desenho de polígonos côncavos quando a primitiva gráfica selecionada for `Preenchido`. O preenchimento das faces na triangularização do polígono côncavo se dá nas posições erradas. Para polígonos convexos o problema não acontece. Foi adicionada uma extensão (seção 4.1) para solucionar este problema.

A seleção dos objetos gráficos no espaço 3D não apresentou um resultado satisfatório. Conforme descrito na seção 3.3, optou-se por utilizar o algoritmo da seleção que faz parte da biblioteca `Three.js`. Para objetos que possuem uma área plana como o `Cubo` e o `Polígono` preenchido, o resultado foi razoável, sendo possível selecionar os objetos na maioria dos casos. Já quando tratam-se de linhas, como no caso da `Spline` e nos polígonos `Aberto` e `Fechado`, a seleção não acontece na maior parte dos casos, sendo possível somente após aproximar-se bastante do objeto no `Espaço Gráfico`.

Após o desenvolvimento, alguns aspectos foram analisados e identificados como possíveis soluções para aumentar o desempenho do código desenvolvido. Um destes aspectos é fazer com as faces do polígono `Preenchido` sejam pré-calculadas para diminuir o custo no

momento do desenho. Atualmente as faces são calculadas toda vez que o polígono é desenhado pois elas são desenhadas de acordo com as propriedades informadas pelo `Usuário`. Foi adicionada uma extensão (seção 4.1) para verificar a possibilidade de fazer com as faces do polígono `Preenchido` sejam pré-calculadas antes da execução da rotina de desenho do objeto gráfico.

Outro aspecto analisado foi a utilização *display-list*, que é um recurso presente na biblioteca OpenGL que serve para armazenar os objetos em memória sem ter que recriá-los várias vezes. Porém, foi verificado que a biblioteca OpenGL ES 2.0 não possui este recurso e como a WebGL é baseada nesta versão da OpenGL, consequentemente também não possui este recurso. Uma alternativa ao uso de *display-list* é a utilização de *Vertex Buffer Object* (VBO), que está presente na biblioteca WebGL. Foi adicionada uma extensão (seção 4.1) para verificar a possibilidade de utilização de VBOs na aplicação.

Por fim, outro aspecto analisado foi a utilização da primitiva `GL_POINTS` presente na OpenGL para representação dos vértices do polígono, quando a primitiva selecionada for `Vértices`. Atualmente, a representação está sendo feita utilizando o desenho de uma esfera para cada um dos vértices. Porém foi verificado que a biblioteca `Three.js` na sua versão atual não disponibiliza nenhum objeto especifico que simule o `GL_POINTS` e verificou-se ainda que, dentre os recursos que existem atualmente na `Three.js`, as esferas são as que proporcionam o resultado visual mais próximo do esperado na representação dos vértices.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de novas funcionalidades de uma aplicação *web* para visualização de material educacional. Além dos conceitos básicos existentes como transformações geométricas, foram incluídos alguns conceitos de complexidade maior, como é o caso da iluminação de cena. Em consequência disto, um dos requisitos que refere-se ao uso da iluminação foi atendido disponibilizando cinco tipos diferentes de luzes que podem ser usadas simultaneamente, associadas a um objeto gráfico específico ou associadas a toda cena. Para que este objetivo fosse atendido, foi necessário obter conhecimento a respeito dos conceitos de iluminação de cena em computação gráfica e dos tipos de luz disponibilizadas na biblioteca `Three.js`.

Atendendo ao objetivo referente a disponibilização de novos objetos gráficos, adicionou-se dois novos itens: `Spline` e `Polígono`. No caso da `Spline`, foi disponibilizada a curva de Bézier. Já o `Polígono` foi disponibilizado podendo ser desenhado com quatro primitivas gráficas diferentes, porém há uma limitação no desenho da primitiva `Preenchido` quando se trata de polígonos côncavos, desenhando as faces de forma incorreta.

A disponibilização da seleção de objetos gráficos no espaço 3D, presente no segundo objetivo proposto foi desenvolvida. Porém por limitação da biblioteca `Three.js`, responsável pelo algoritmo de seleção, apresentou pouca operacionalidade principalmente em objetos gráficos que utilizam linhas, como é o caso da `Spline` e `Polígono`.

Por fim, atendendo ao objetivo que refere-se a disponibilização do espaço gráfico 2D, foram disponibilizadas todas as funcionalidades que existem no ambiente 3D, também no ambiente 2D.

Com relação aos resultados do teste de operacionalidade realizado com os alunos de Computação Gráfica da FURB, é possível concluir que, tratando-se do entendimento dos conceitos de computação gráfica e de operacionalidade, o ambiente atendeu as expectativas dos alunos. Nos testes de desempenho foi identificado que, com relação ao consumo de memória, o navegador Google Chrome apresentou o menor consumo se comparado com os navegadores Mozila Firefox e Opera. Já com relação ao processamento de FPS, o navegador Mozila Firefox apresentou o melhor processamento, quando comparado com os demais navegadores. Sendo assim, tratando-se do desempenho, é possível concluir que o Mozila Firefox é o navegador mais indicado para utilização do sistema pois apresentou o melhor processamento de FPS, apesar do alto consumo de memória.

4.1 EXTENSÕES

Durante as etapas finais do desenvolvimento deste trabalho, foram identificados alguns pontos que podem ser melhorados e algumas sugestões de extensões, que são:

a) verificar a possibilidade de utilização de VBO na aplicação para diminuir o custo de processamento;
b) disponibilizar os painéis de forma dinâmica;
c) disponibilizar animação de cena;
d) disponibilizar toda a cena gráfica em um espaço normalizado;
e) alterar o uso do clique duplo no acesso a ajuda da aplicação para não conflitar com a seleção de objetos no `Espaço Gráfico`;
f) ajustar o desenho do `Polígono` com a primitiva `Preenchido` para desenhar corretamente os polígonos côncavos;
g) verificar a possibilidade de fazer com as faces do polígono `Preenchido` sejam pré-calculadas antes da execução da rotina responsável pelo desenho do objeto gráfico;
h) criar uma tabela de pesos pré-calculada para o cálculo dos pontos da `Spline` em vez de realizar o cálculo toda a vez que o objeto gráfico é desenhado;
i) disponibilizar a geração do código JOGL para as peças `Polígono` e `Spline`.
j) disponibilizar uma forma do `Usuário` identificar no `Espaço Gráfico` qual o ponto está sendo alterado nas propriedades do `Polígono` e da `Spline`;
k) possibilitar a utilização dos controles do `Espaço Gráfico` quando o tipo de gráfico selecionado for 2D;
l) possibilitar a manipulação das propriedades `Posição` e `Look At` da peça `Câmera` quando o tipo de gráfico selecionado for 2D.

REFERÊNCIAS

ANYURU, Andreas. **Professional WebGL programming**: developing 3D graphics for the web. Chichester: Wrox, 2012.

AZEVEDO, Eduardo; CONCI, Aura. **Computação gráfica**: teoria e prática. Rio de Janeiro: Elsevier, 2003.

BICHO, Alessandro L. et al. **Programação gráfica 3D com OpenGL, Open Inventor e Java 3D**. [Rio de Janeiro], 2002. Disponível em: <http://www.joinville.udesc.br/portal/professores/rogerio/materiais/Programacao_Visual.pdf>. Acesso em: 02 nov. 2013.

CARTOUCHE. **Quadratic and Cubic Bézier Curves**. [Zurich], 2012. Disponível em: <http://www.e-cartouche.ch/content_reg/cartouche/graphics/en/html/Curves_learningObject2.html>. Acesso em: 13 jul. 2014.

CATARINA, Adair S. **Computação gráfica**. [Cascavel], 2013. Disponível em: <http://www.inf.unioeste.br/~adair/CG/Notas%20Aula/Aulas%20de%20CG%202013.pdf>. Acesso em: 13 set. 2013.

FHTR. **Introduction to Three.js**. [S.l.], 2012. Disponível em: <http://fhtr.org/BasicsOfThreeJS/>. Acesso em: 9 set. 2013.

FILIPEK, Bartlomiej. **Select + Mouse + OpenGL**. [Cracóvia], 2012. Disponível em: <http://www.bfilipek.com/2012/06/select-mouse-opengl.html>. Acesso em: 13 jul. 2014.

GOMES, Jonas; VELHO, Luiz. **Fundamentos da computação gráfica**. Rio de Janeiro: IMPA, 2003.

GROSSKURTH, Alan; GODFREY, Michael W. **Architecture and evolution of the modern web browser**. Waterloo, 2006. Disponível em: <http://grosskurth.ca/papers/browserarchevol-20060619.pdf>. Acesso em: 15 set. 2013.

KHRONOS. **WebGL specification**. [S.l.], 2013. Disponível em: <https://www.khronos.org/registry/webgl/specs/1.0/>. Acesso em: 9 set. 2013.

MAZUROK, Igor E. **Aprender a programar**. [Odessa], 2013. Disponível em: <http://mazurok.com/wordpress/?p=294>. Acesso em: 9 set. 2013.

MONTENEGRO, Anselmo. **Objetos gráficos 2D.** [S.l.], 2013. Disponível em: <http://www2.ic.uff.br/~anselmo/cursos/CGI/slides/CG_aula3(objetosgraficos2D).ppt>. Acesso em: 13 set. 2013.

MONTIBELER, James P. **VISEDU-CG**: visualizador de material educacional, módulo de computação gráfica. 2014a. 103 f. Trabalho de conclusão de curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MONTIBELER, James P. **Repositório JamesPMontibeler.** Blumenau, 2014b. Disponível em: <https://bitbucket.org/VisEdu-Educacao/samuelanunes>. Acesso em: 6 mar. 2014.

MORAN, José M. **Como utilizar a Internet na educação.** [S.l.], 1997. Disponível em: <http://www.scielo.br/pdf/ci/v26n2/v26n2-5.pdf>. Acesso em: 15 set. 2013.

MRDOOB. **Releases**. [S.l.], 2014. Disponível em: <https://github.com/mrdoob/three.js/releases>. Acesso em: 9 maio 2014.

NUNES, Samuel A. **Repositório SamuelANunes.** Blumenau, 2014. Disponível em: < https://bitbucket.org/VisEdu-Educacao/samuelanunes>. Acesso em: 13 nov. 2013.

PEREIRA, Daniel. **Desenvolvimento de um motor de jogos 3D, utilizando WebGL**. 2012. 65 f. Trabalho de conclusão de curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SCHABACK, Johanens. **OpenGL picking in 3D**. [S.l.], 2011. Disponível em: <http://schabby.de/picking-opengl-ray-tracing/>. Acesso em: 17 set. 2013.

SILVA, Isabel C. S. **Aprendendo computação gráfica com OpenGL e Blender**. Porto Alegre: UniRitter, 2007.

SOUSA, Augusto. **Modelação de sólidos**. [S.l.], 2004. Disponível em: <http://paginas.fe.up.pt/~aas/pub/Aulas/CG/Slides/11_ModelSolid.pdf>. Acesso em: 17 set. 2013.

STEP. **StarLogo TNG**. [S.l.], 2013. Disponível em: <http://education.mit.edu/projects/starlogo-tng>. Acesso em: 10 set. 2013.

TAVARES, Denison L. M. **Aproximação eficiente de visibilidade para nuvem de pontos utilizando a GPU**. Porto Alegre, 2009. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/25502/000751162.pdf>. Acesso em: 17 set. 2013.

UNIVERSIDADE FEDERAL FLUMINENSE. **Computação gráfica**: processamento e análise de imagens digitais. [S.l.], 2009. Disponível em: <http://computacaografica.ic.uff.br/transparenciasvol2cap7.pdf>. Acesso em: 17 set. 2013.

UNIVERSIDADE FEDERAL FLUMINENSE. **Projeções em perspectiva**. [S.l.], 2013. Disponível em: <http://www.uff.br/cdme/v3d/v3d-html/v3d-segment-05-br.html>. Acesso em: 17 set. 2013.

W3C. **HTML5**. [S.l.], 2013. Disponível em: <http://www.w3.org/TR/html5/>. Acesso em: 10 set. 2013.

W3SCHOOLS. **HTML5 introduction**. [S.l.], 2013. Disponível em: <http://www.w3schools.com/html/html5_intro.asp>. Acesso em: 10 set. 2013.

WILLIAMS, James L. **Learning HTML5 game programming**: a hands-on guide to building online games using Canvas, SVG, and WebGL. Chichester: Wrox, 2011.

APÊNDICE A – Detalhamento dos casos de uso

Este apêndice apresenta o detalhamento do novo caso de uso desenvolvido (UC12) juntamente com os casos de uso (UC01, UC02, UC03, UC05, UC06, UC07, UC08 e UC11) desenvolvidos por Montibeler (2014a, p.28-34) e que sofreram alterações na versão 3.0 do VisEdu-CG.

Quadro 9 - Detalhamento do caso de uso UC01

UC01 – Criar novas peças para construção do exercício	
Descrição	As peças do exercício são aquelas que são encaixadas no `painel de montagem` para possibilitar a geração da cena 3D e do código fonte. Cada peça representa um comando ou conceito de computação gráfica, como por exemplo: o desenho de um cubo, Polígono ou Spline, um objeto gráfico (empilhamento e desempilhamento de matrizes de transformação), as transformações de escala, rotação e translação, ou ainda a Iluminação de cena e a visão de uma câmera em um cenário 3D ou 2D. O painel `Fábrica de Peças` é composto por dois subpainéis: a `fábrica` e o `painel de montagem`. O painel `fábrica` possui todas as peças que podem ser fabricadas pelo usuário e o `painel de montagem` apresenta um ambiente para realizar a montagem do exercício através do encaixe das peças fabricadas. Ele possui uma peça principal (`Renderizador`) que disponibiliza encaixes para as peças `Câmera` e `Objeto Gráfico`, permitindo a inicialização do exercício. A peça `Objeto Gráfico` irá disponibilizar encaixes internos para peças filhas, com encaixes para as peças `Cubo`, `Polígono`, `Spline`, `Transladar`, `Rotacionar`, `Escalar`, `Iluminação` e `Objeto Gráfico`.
Cenário Principal	1. O `Usuário` clica com o botão direito do mouse em uma peça da `fábrica` e a arrasta para um dos encaixes livres compatíveis, dentro do `painel de montagem`, mantendo o botão do mouse pressionado.
Cenário Alternativo 1	1. Se no passo 1 do cenário principal o `Usuário` soltar a peça dentro da `fábrica`, a peça sendo arrastada será excluída.
Cenário Alternativo 2	1. Se no passo 1 do cenário principal o `Usuário` soltar a peça dentro do `painel de montagem`, fora de um encaixe compatível, a peça sendo arrastada ficará na posição que foi solta.
Pós-Condição	O `Usuário` deve poder visualizar a nova peça encaixada no `painel de montagem`. A peça deverá ter um nome automático, conforme um contador interno de inclusão de peças do mesmo tipo. Peças encaixadas alteram o resultado gráfico exibido na cena 2D/3D do exercício. Peças encaixadas são incluídas na `Lista de Peças`. O `Usuário` poderá visualizar a peça conforme o cursor do mouse é movimentado e a peça deve acompanhar o cursor.

Quadro 10 - Detalhamento do caso de uso UC02

UC02 - Reencaixar ou encaixar peças existentes no painel de montagem	
Descrição	As peças já encaixadas no `painel de montagem` podem ser movidas para outros encaixes compatíveis com o encaixe da peça, alterando assim o resultado obtido pelo exercício. Também é possível encaixar peças que foram soltas no `Painel de Montagem` e que não estão encaixadas em nenhum encaixe. Essas peças não encaixadas não afetam o resultado do exercício até serem encaixadas em um encaixe.
Pré-Condição	UC01
Cenário Principal	1. O `Usuário` clica com o botão direito do mouse em uma peça do `Painel de Montagem` e a arrasta para um dos encaixes livres compatíveis, mantendo o botão do mouse pressionado.
Cenário Alternativo 1	1. Se no passo 1 do cenário principal o `Usuário` soltar a peça dentro do painel `Fábrica`, a peça sendo arrastada será excluída.
Cenário Alternativo 2	1. Se no passo 1 do cenário principal o `Usuário` soltar a peça dentro do `painel de montagem`, fora de um encaixe compatível, e ela não estiver encaixada em outra peça (flutuando no painel), a peça sendo arrastada ficará na posição que foi solta.
Cenário Alternativo 3	1. Se no passo 1 do cenário principal o `Usuário` soltar a peça dentro do `Painel de Montagem`, fora de um encaixe compatível, e ela estiver encaixada em outra peça, a peça sendo arrastada continuará encaixada no encaixe de origem e o movimento da peça será cancelado.
Pós-Condição	O `Usuário` deve poder visualizar a peça posicionada no seu novo encaixe no `Painel de Montagem`. Peças encaixadas alteram o resultado gráfico exibido na cena 2D ou 3D do exercício. A `Lista de Peças` será atualizada com o novo encaixe da peça, se existir. O usuário poderá visualizar a peça conforme o cursor do mouse é movimentado e a peça deve acompanhar o cursor.

Quadro 11 - Detalhamento do caso de uso UC03

UC03 - Selecionar objeto gráfico	
Descrição	Através da seleção dos objetos gráficos o `Usuário` tem acesso as propriedades do mesmo no painel `Propriedades da Peça`.
Pré-Condição	UC01
Cenário Principal	1. O `Usuário` clica com o botão esquerdo do mouse em uma peça do `Painel de Montagem`.
Cenário Alternativo 1	1. O `Usuário` clica com o botão esquerdo do mouse sobre o nome da peça na `Lista de Peças`.
Cenário Alternativo 2	1. O `Usuário` realiza duplo clique com o botão esquerdo do mouse no objeto gráfico diretamente no `Espaço Gráfico`.
Pós-Condição	No `Painel de Montagem` a peça terá uma coloração diferente das demais peças. O painel `Propriedades da Peça` será exibido sobre o painel `fábrica`, exibindo a(s) propriedade(s) da peça selecionada. O painel `Comandos em JOGL` será atualizado com o código fonte da peça. No `Espaço Gráfico` é desenhado a *BBox* do objeto selecionado, evidenciando a seleção.

Quadro 12 - Detalhamento do caso de uso UC05

UC05 - Alterar e consultar as propriedades da peça selecionada	
Descrição	É possível consultar e alterar as propriedades das peças existentes no exercício, alterando assim o resultado no cenário 2D ou 3D e no código fonte. Com isto é possível visualizar o impacto de cada peça e de cada propriedade existente. A peça `Renderizador` permite modificar a cor de limpeza, que pode ser visualizada pelo painel `Visão da Câmera`. A peça `Câmera` permite a definir um nome para a peça, sua posição (`x`, `y`, `z`), o *look at* (x, y, z), o *near*, o *far* e o *field of view* (FOV). A peça `Objeto Gráfico` permite definir um nome para a peça e se ela será visível na cena. A peça também permite visualizar a matriz resultante das peças de transformações geométricas encaixadas nela. A peça `Cubo` permite a definir um nome para a peça, o seu tamanho (x, y, z), a sua posição (x, y, z), a sua cor ou a sua textura e se ela será visível na cena. A peça `Polígono` permite a definir um nome para a peça, a sua quantidate de pontos, o ponto que está selecionado (x, y, z), a sua primitiva gráfica, além da sua cor e se ela será visível na cena. A peça `Spline` permite a definir um nome para a peça, o seu tipo e a quantidade de pontos. Além disso é possível definir a posição (x, y, z) de cada um dos quatro pontos de controle, além da sua cor ou se ela será visível na cena. É possivel também habilitar o desenho do seu `Poliedro`, definindo também uma cor especifica para ele. A peça `Iluminação` permite a definir um nome para a peça, a sua posição (x, y, z), a sua cor e se ela será visível na cena. Além disso é possivel definir a intensidade da luz. Quanto o tipo de luz for *Directional* ou *SpotLight* é possível definir a posição (x, y, z) do alvo da luz. As peças `Transladar`, `Rotacionar` e `Escalar` permitem definir um nome para as peças, o valor da transformação (x, y, z) e se a transformação será visível na cena.
Pré-Condição	UC03
Cenário Principal	1. O `Usuário` seleciona uma peça. 2. O `Usuário` altera a propriedade desejada.
Pós-Condição	Os valores definidos para as propriedades deverão ser atualizados no código fonte e nas cenas do resultado do exercício.

Quadro 13 - Detalhamento do caso de uso UC06

UC06 - Visualizar cena gráfica formada a partir das peças encaixadas	
Descrição	No painel `Espaço Gráfico`, conforme o usuário encaixa as peças ou altera as propriedades das mesmas, é possível visualizar o resultado gráfico da peça ou alteração efetuada. Ele exibe um espaço gráfico que possui 4 componentes principais: a) indicador dos eixos x, y e z: são três retas que indicam a direção dos eixos a partir do ponto [0,0,0] que são identificadas pelas cores vermelho (x), verde (y) e azul (z). Quando o tipo de gráfico for 2D, o eixo z não é exibido; b) grade: um plano quadrado em forma de grade, centralizado no ponto [0,0,0], que se estende pelos planos x e z, auxiliando na percepção da profundidade do ambiente. Quando o tipo de gráfico for 2D, a grade não é exibida; c) pirâmide da câmera: uma pirâmide tridimensional de 4 faces que indica a amplitude da visão da peça `Câmera` dentro do ambiente gráfico; d) cubos, polígonos e curvas: representam as peças `Cubo`, `Polígono` e `Spline` respectivamente, encaixadas no editor, exibindo as transformações e propriedade aplicadas sobre os mesmos. e) iluminação: quanto o tipo de luz definido nas propriedades da peça Iluminação for Hemisphere, Directional, PointLight ou SpotLight, é exibido a representação da posição da luz no Espaço Gráfico. Quando o tipo de gráfico for 3D, é possível também ajustar o ponto de vista do `Espaço Gráfico` usando os botões do mouse: a) botão esquerdo: ao pressionar o botão esquerdo e arrastar o mouse, o ponto de vista é rotacionado; b) botão direito: ao pressionar o botão esquerdo e arrastar o mouse, o ponto de vista é transladado; c) botão de rolagem: ao rolar o botão de rolagem do mouse, é possível aumentar ou diminuir o *zoom* do ponto de vista.
Pré-Condição	UC01, UC02 ou UC05
Cenário Principal	1. Encaixar uma peça no `Painel de Montagem` ou editar a propriedade de alguma peça.
Pós-Condição	O conceito gráfico representado pela peça, ou propriedade, deverá ser visualizado ou modificar o ambiente gráfico do painel `Espaço Gráfico`.

Quadro 14 - Detalhamento do caso de uso UC07

UC07 - Visualizar visão da peça `Câmera` encaixada no exercício	
Descrição	Através do painel `Visão da Câmera` é possível visualizar a cena 2D ou 3D formada pelo exercício a partir de um segundo ponto de vista, que é determinado pelas propriedades da peça `Câmera` encaixada no `painel de montagem`. Os componentes de orientação dentro do painel `Espaço Gráfico` (os indicadores dos eixos x, y e z, a grade e a pirâmide da câmera) não serão enxergados neste painel. Assim, os únicos objetos que serão reproduzidos na cena serão o `Cubo`, `Polígono` ou `Spline`, assim como a influência do objeto `Iluminação` nestes objetos. A cor de limpeza da cena exibida no painel `Visão da Câmera` usa a cor de limpeza definida nas propriedades da peça `Renderizador` do `painel de montagem`.
Pré-Condição	UC01, UC02 ou UC05
Cenário Principal	1. Encaixar na peça `Renderizador` a peça `Câmera` ou editar as propriedades da peça `Câmera` já encaixada.
Pós-Condição	O painel `Visão da Câmera` deverá exibir a visão da cena 2D ou 3D do exercício (também visualizada no painel `Espaço Gráfico`) a partir do ponto de vista definido pela peça `Câmera` encaixada na peça `Renderizador`.

Quadro 15 - Detalhamento do caso de uso UC08

UC08 - Gerar um texto em formato JSON com os dados da estrutura formada pelas peças encaixadas	
Descrição	É possível criar um texto contento os dados do exercício criado no aplicativo. Através do painel `Arquivo` é possível gerar um texto em formato JSON com toda a estrutura de peças encaixadas no exercício.
Cenário Principal	1. Selecionar o painel `Arquivo`; 2. Selecionar a opção `Exportar`.
Pós-Condição	A aplicação irá iniciar automaticamente o *download* de um arquivo texto contendo o texto com os dados do exercício.

Quadro 16 - Detalhamento do caso de uso UC12

UC12 – Alterar o tipo de gráfico	
Descrição	Permite ao `Usuário` escolher se deseja trabalhar com gráficos em 2D ou 3D.
Pré-Condição	-
Cenário Principal	1. O `Usuário` seleciona a peça `Renderizador`. 2. Nas propriedades da peça, o `Usuário` altera a opção `Gráficos` para o tipo de gráfico desejado. 3. A aplicação exibe uma mensagem de confirmação informando que a cena atual será perdida. 4. O `Usuário` confirma a alteração do tipo de gráfico. 5. O `Espaço Gráfico` é modificado para trabalhar com a opção escolhida.
Cenário Alternativo	2. Caso no passo 3 do cenário principal o `Usuário` cancelar a alteração do tipo de gráfico, a aplicação não altera o tipo de gráfico.
Pós-Condição	O `Usuário` passa a ser capaz de trabalhar com a opção de gráficos em 2D ou 3D, conforme selecionada.

APÊNDICE B – Lista de exercícios desenvolvida pelos alunos no teste de operacionalidade

A atividade que foi aplicada no teste de operacionalidade da aplicação está ilustrada na Figura 34, Figura 35 e Figura 36.

Figura 34 – Primeira página da atividade aplicada no teste de operacionalidade

DSC – Departamento de Sistemas e Computação (FURB)
Grupo de Pesquisa em Computação Gráfica, Processamento de Imagens e Entretenimento Digital
Disciplina: Computação Gráfica - prof. Dalton Solano dos Reis

Unidade 04 - Conceitos básicos de 3D

Informações utilizadas no exercício:
- aonde encontrar o VisEdu-CG: http://www.inf.furb.br/gcg/visedu/cg/
- os arquivos das respostas devem ser zipados e salvos no AVA em VisEdu-CG
- cada arquivo de resposta deve ser nomeado seguindo este padrão cg4r_n.txt, aonde "n" corresponde ao número do exercício
- uma copia dos arquivos utilizados como "entrada" nestes exercícios (se for o caso) estão em: http://www.inf.furb.br/gcg/visedu/cg/exercicios

- após terminar o exercício FAVOR RESPONDER o questionário: https://docs.google.com/forms/d/133mxQCax35kmpdXNdoH9SZ7zzMrmIPE7Y5hidDIX7a8/viewform?pli=1

1) Crie uma cena utilizando o VisEdu-CG que **somente** contenha as peças "Camera", "Objeto Gráfico", "Cubo" (com textura "Logo Grupo CG") e "Iluminação". Após altere as "Propriedades" da peça "Camera 1" no "Renderizador" para mudar o ponto de vista do observador na cena. A visualização do objeto "Cubo 1" deve ser a mais próxima possível ao exibido na imagem abaixo. Neste caso utilize os valores de "Near", "Far" e "FOV" também descrito na figura abaixo, mudando os valores de "Posicao" e "Look At".

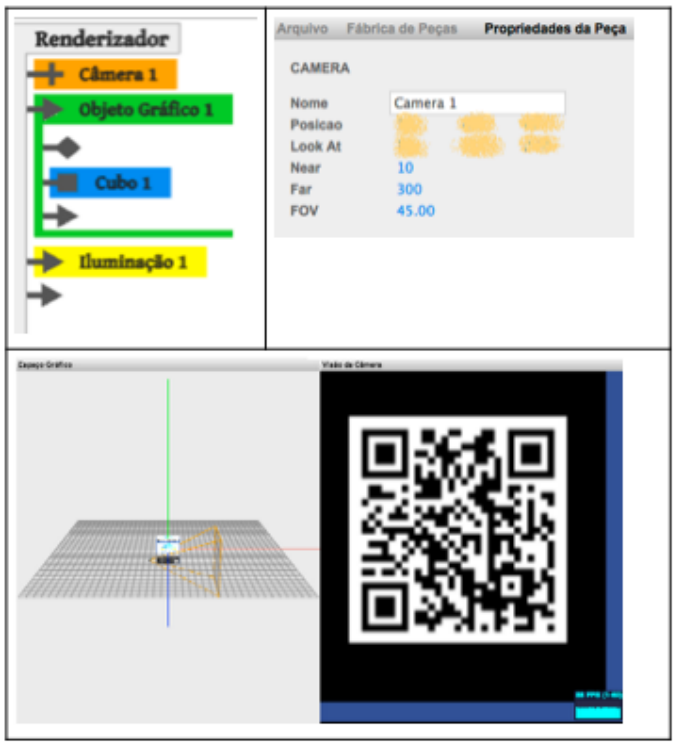
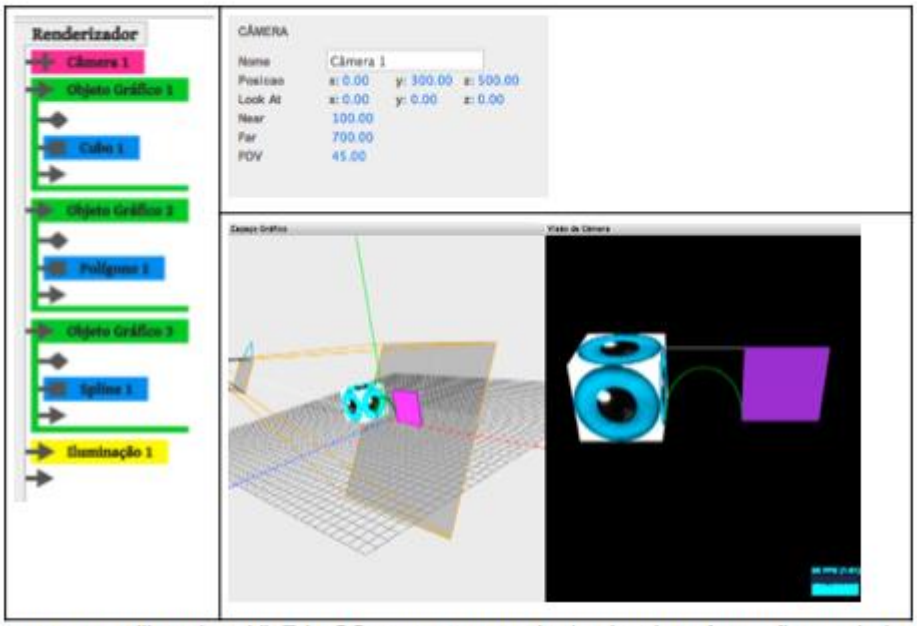


Figura 35 – Segunda página da atividade aplicada no teste de operacionalidade

DSC – Departamento de Sistemas e Computação (FURB)
Grupo de Pesquisa em Computação Gráfica, Processamento de Imagens e Entretenimento Digital
Disciplina: Computação Gráfica - prof. Dalton Solano dos Reis

2) Crie uma cena utilizando o VisEdu-CG que **somente** contenha as peças descritas na figura abaixo para que tem as mesmas texturas e representação visual. No caso o tamanho do objeto "cubo" foi mantido com 100 unidades e esta posicionado com a face de baixo com valor no plano "zero" do eixo Z. Os Objetos "Polígono" e "Spline" tem a sua BBox com o mesmo tamanho do "cubo".



3) Crie uma cena utilizando o VisEdu-CG para ter o resultado visual conforme figura abaixo. Não é permitido mudar os parâmetros dos objetos "Cubo", somente o valor da sua textura. O objeto "Cubo 1" tem o seu centro com coordenadas (0,0,0) coincidindo com a origem, os outros dois "Cubos" sofreram transformação geométrica e se encontram adjacentes ao "Cubo 1". Todos os objetos "Cubo" são ampliados somente na altura em duas vezes ao seu tamanho original usando uma transformação geométrica. Neste caso esta cena **somente** pode conter as peças: uma "Câmera", uma "Iluminação", três "Objetos Gráficos", três "Cubo", uma "Escalar" e dois "Transladar".

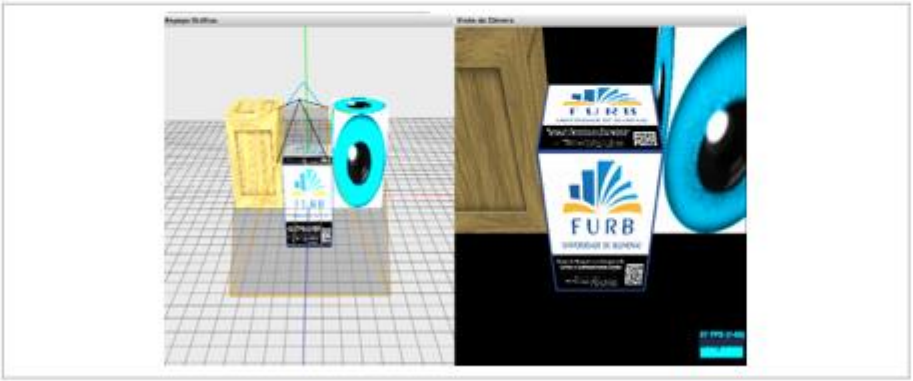
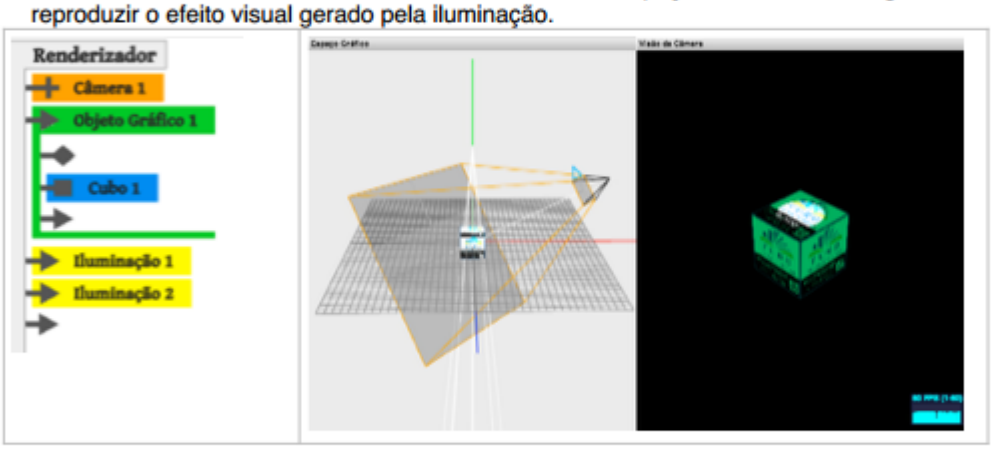


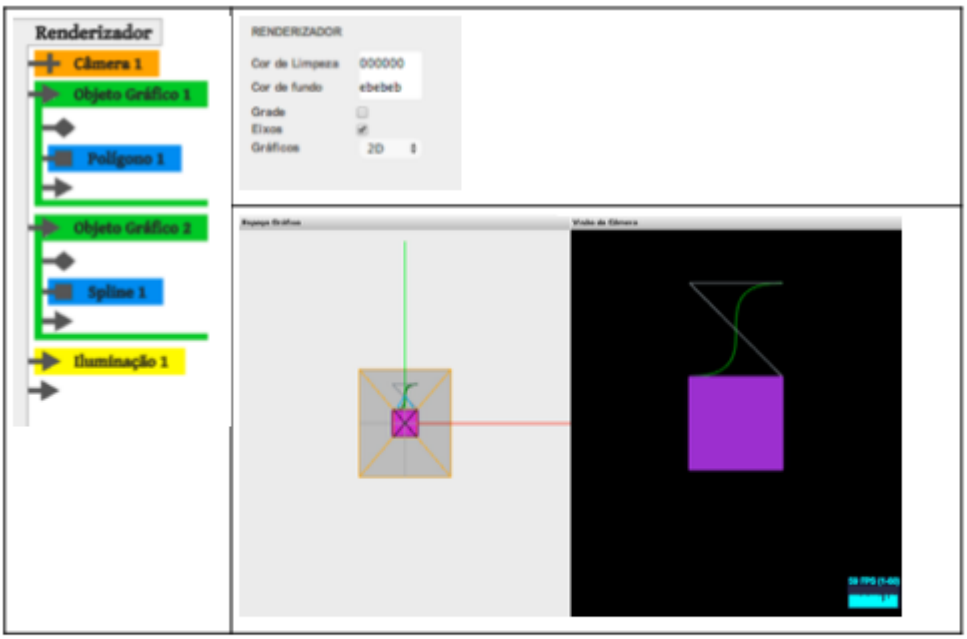
Figura 36 – Terceira página da atividade aplicada no teste de operacionalidade

DSC – Departamento de Sistemas e Computação (FURB)
Grupo de Pesquisa em Computação Gráfica, Processamento de Imagens e Entretenimento Digital
Disciplina: Computação Gráfica - prof. Dalton Solano dos Reis

4) Use o arquivo "CG-04_exer_04.txt" (entrada) disponível em "http://gcg.inf.furb.br/visedu/cg/exercicios/CG-04_exer_04.txt" e usando **somente** as peças descritas na figura abaixo para reproduzir o efeito visual gerado pela iluminação.



5) Crie uma cena utilizando o VisEdu-CG usando **somente** as peças descritas na figura abaixo. O resultado visual deve ser idêntico ao exibido na figura. Mude opção do "Renderizador"/"Gráficos" para usar **Espaço Gráfico 2D.** O Objeto "Polígono" tem uma largura e altura de 100 unidades e o seu centro está posicionado na origem.



LEMBRE de responder o questionário: https://docs.google.com/forms/d/133mxQCax35kmpdXNdoH9SZ7zzMrmIPE7Y5hidDIX7a8/viewform?pli=1

Boa prática!

APÊNDICE C – Respostas das questões descritivas do teste de operacionalidade

As respostas da questão referente aos pontos positivos do uso da aplicação estão descritas no Quadro 17.

Quadro 17 – Respostas da questão referente aos pontos positivos

Com base nas respostas anteriores, quais seriam os principais "Pontos Positivos" no uso do aplicativo?
Este aplicativo facilita a aplicação de conceitos de computação gráfica de forma simples e rápida, possui uma boa interface simples e intuitiva.
É simples de usar, permite obter a intuição necessária para os primeiros contatos com computação gráfica.
Intuitivo e agradável de utilizar.
Web, e com conceitos básicos e medianos de computação gráfica.
Rápido e prático no desenvolvimento de exemplos, atividades.
Simples e fácil de utilizar.
Facilidade do uso.
Usabilidade intuitiva. Código gerado fácil de entender.
Bem tranquilo de entender, mesmo nunca tendo trabalhado com ele não precisei da janela de ajuda.
A visualização do espaço gráfico e a forma visual de criação dos objetos da cena, que permite um melhor entendimento do papel de cada item na cena. Também vale comentar a possibilidade de visualizar o código JOGL que pode ajudar quando há dúvida em implementações (ocorreu comigo).
Facilidade de uso
O aplicativo demonstra bem uma estrutura de grafo de cena, bem como permite a visualização em tempo real do que está sendo feito.
Ajuda a entender os conceitos de Computação Gráfica mais rápido e de forma simples.
O aplicativo consegue, com uma interface intuitiva e prática, mostrar diversos elementos que podem ser usados para compor uma cena.
Facilidade no uso. Grande ajuda no entendimento do que é e para que serve cada objeto gráfico. Ótima ajuda no uso de JOGL.
Um usuário que não é da área de computação tem capacidade para manipular o aplicativo, apenas conhecendo o plano cartesiano e conceitos matemáticos.
Facilidade nos testes referente aos conceitos de computação gráfica de forma visual (interagindo com as "peças") e ao mesmo tempo podemos analisar o código gerado em JOGL.

As respostas da questão referente aos pontos negativos do uso da aplicação estão descritas no Quadro 18.

Quadro 18 – Respostas da questão referente aos pontos negativos

Com base nas respostas anteriores, quais seriam os principais "Pontos Negativos" no uso do aplicativo?
Apenas o tamanho da janela em que os comandos JOGL ficam.
Usabilidade pode ser melhorada.
Alguns problemas com resolução e tamanho dos componentes.
Na parte de scroll das telas não funciona com o mouse. A rotação da scena é muito precária.
Apenas o click no espaço gráfico, é muito sensível.
O comportamento da aplicação é um tanto estranho, as vezes alterando entre a Fábrica e as Propriedades sem querer. Além de duplo click abrir a ajuda? Seria mais interessante o duplo click faz a alternância entre Fábrica e Propriedades.
Não encontrei uma função para remover os objetos, lentidão.
Não encontrado.
Achei a definição dos vértices um pouco ruim. Não conseguia entender qual o ponto que eu estava modificando. Poderia ter um destaque no ponto que está sendo editado.
Talvez por ainda ser uma versão em desenvolvimento, ocorrem alguns erros de utilização que comprometem um pouco da experiência. Em geral, quem é da área de computação consegue desviar destes "problemas", mas podem existir complicações com pessoas mais leigas.
Não encontrei pontos negativos.
Encontrei algumas dificuldades para trabalhar em determinadas peças, principalmente em propriedades do polígono e spline.
Pouco intuitivo em algumas partes.
Para o que se propõe, não vi pontos negativos na ferramenta.
Não há pontos negativos.
As vezes ocorrem erros inesperados, de acordo com as versões do Chrome.
Não identifiquei nenhum.

As respostas da questão referente às sugestões de melhorias da aplicação estão descritas no Quadro 19.

Quadro 19 – Respostas da questão referente as sugestões de melhorias

Com base nas respostas anteriores, quais seriam as suas "Sugestões" de melhoria para o aplicativo?
Talvez diminuir o espaço da janela lista de peças e aumentar o espaço da janela comando JOGL.
O botão exportar não funcionou no Firefox. Na janela onde se arrastam os componentes para formar a cena, o scroll do mouse não funciona. É necessário clicar na barra lateral e arrastar.
Melhorar os pontos citados acima.
Apenas o espaço gráfico.
JOGL: Acho que utilizar de cores (talvez cinza e preto) para destacar o que é da biblioteca OpenGL do código em Java. Além disso por questão de didática talvez seja interessante os códigos que aparecem em Java estarem em português. Não entendi a ideia da Janela "Lista de peças", pois já temos a listagem no renderizador. Talvez seja interessante passar o JOGL para a "Lista de peças" e fixar uma janela com as propriedades, e o local onde hoje é alternado entre propriedades e fábrica ficar só para fabrica. As outras opções fazer um menu como um context menu no canto superior esquerdo.
Melhorar a performance do aplicativo.
Ao criar um polígono é necessário selecionar no combo os pontos, o combo poderia estar acima dos valores.
Acabei, sem querer, selecionando e arrastando o objeto renderizador e ao tentar colocar no lugar novamente invadi a janela superior que era a fábrica de objetos. Ele foi selecionado para a exclusão (a lixeira), mas ele não se exclui e após isso o objeto avançava para a janela de cima toda a vez que rolava a barra para ver todos os objetos que coloquei. Sugiro também permitir o scroll na janela de objetos que estou usando na cena.
Poder omitir janelas que sejam relevantes para o que a pessoa está fazendo. Adição de um tutorial guiado no estilo do Facebook quando são adicionadas novas funcionalidades. Evitar disponibilizar a versão desenvolvimento, deixando sempre a versão mais estável para uso.
Adição do recurso de dicas (hints) nas propriedades dos objetos gráficos.
Apenas melhorar a interação do usuário com os componentes, principalmente no momento da edição das propriedades.
Enfrentei alguns bugs envolvendo o 'drag and drop' dos componentes. O evento deveria ser revisado.
Visualização do código JOGL em pop-up, ou outra forma em que seja possível ver mais código sem ter que ficar mexendo nas barras de rolagem.
Corrigir os bugs que ocorrem com diferentes versões do Chrome, como por exemplo a câmera que as vezes não é possível selecionar as propriedades.
Dificuldade na manipulação do ângulo do nosso ponto de vista do ambiente gráfico 3d (com o mouse).

ANEXO A – Requisitos da aplicação VisEdu-CG 2.0

Neste anexo encontram-se os principais requisitos do ambiente VisEdu-2.0, desenvolvido por Montibeler (2014a). Quanto aos requisitos funcionais, o VisEdu-CG deverá:

a) disponibilizar as funcionalidades existentes no AduboGL;
b) possuir funções que executem comandos do WebGL;
c) disponibilizar peças para representar comandos gráficos básicos de CG;
d) permitir o encaixe das peças em outras peças que disponibilizem o tipo de encaixe correspondente;
e) permitir que o usuário resolva exercícios de computação gráfica que utilizem o conceito de câmera;
f) permitir que o usuário resolva exercícios de computação gráfica que utilizem um cubo e os conceitos de transformações de translação, rotação e escala;
g) permitir o agrupamento de transformações sobre um objeto gráfico e seus filhos;
h) permitir que o usuário resolva exercícios de computação gráfica que utilizem o cubo e o conceito de textura;
i) permitir que o usuário da aplicação visualize um cenário 3D montado a partir da estrutura das peças encaixadas;
j) permitir que o usuário visualize um cenário 3D a partir do ponto de vista definido para a peça que representa uma câmera;
k) permitir alterar a cor de limpeza do cenário 3D que representa o ponto de vista definido para a peça que representa uma câmera;
l) permitir definir um nome para cada peça inserida no editor;
m) permitir alterar a posição, o *look at*, o *near*, o *far* e o *Field Of View* (FOV) da peça que representa uma câmera;
n) permitir habilitar ou desabilitar a visualização de um objeto gráfico (cubo) no cenário 3D formado a partir das peças encaixadas;
o) permitir visualizar a matriz resultante das transformações aplicadas sobre um determinado objeto gráfico;
p) permitir que o usuário altere os valores das transformações existentes sobre um determinado objeto gráfico;
q) permitir que o usuário habilite ou desabilite as transformações existentes;
r) permitir que o usuário altere o tamanho, a posição, a cor e a textura dos cubos

existentes no editor;

s) disponibilizar uma lista de peças em forma de árvore com todas as peças encaixadas;
t) permitir que o usuário selecione uma peça a partir da lista de peças;
u) permitir que o usuário selecione uma peça do editor com o mouse;
v) permitir a visualização de código em Java OpenGL (JOGL) dos comandos gráficos representados pela peça selecionada;
w) permitir que o usuário salve o exercício realizado;
x) permitir que o usuário carregue o exercício salvo para continuá-lo;
y) disponibilizar um tutorial de ajuda com informações a respeito do funcionamento da ferramenta.

Quanto aos requisitos não funcionais, o VisEdu-CG deverá:

a) ser desenvolvido na linguagem HTML5;
b) utilizar o WebGL para executar as funções de computação gráfica;
c) utilizar o framework Three.js;
d) ser de código aberto;
e) executar em qualquer navegador que tenha suporte ao WebGL.

ANEXO B – Casos de uso da aplicação VisEdu-CG 2.0

Neste anexo encontram-se descrições detalhadas dos casos de uso UC04, UC09 e UC10, desenvolvidos por Montibeler (2014a, p.30-33) e que não sofreram alterações na versão 3.0.

O UC04 mostra como o ator `Usuário` visualiza o código-fonte JOGL de uma peça ou do exercício. O Quadro 20 apresenta os detalhes do caso de uso.

Quadro 20 - Detalhamento do caso de uso UC04

UC04 - Visualizar código JOGL da peça selecionada	
Descrição	A aplicação permite que o `Usuário` visualize o código fonte de: a) uma peça: selecionando a mesma; b) de um conjunto de peças: selecionando uma peça que tenha outras peças encaixadas a ela (peças filho); c) uma classe Java pronta para compilar o exercício em um projeto configurado com a biblioteca JOGL: selecionando a peça `Renderizador` do `painel de montagem`. Na classe será necessário ajustar o caminho das texturas, caso o exercício faça uso de texturas.
Pré-Condição	UC03
Cenário Principal	1. O `Usuário` seleciona uma peça.
Pós-Condição	O painel `Comandos em JOGL` será atualizado com o código fonte correspondente da peça.

O UC09 mostra como o ator `Usuário` consegue abrir exercícios a partir de textos em formado JSON com a mesma estrutura dos textos gerados pela ferramenta. O Quadro 21 apresenta os detalhes do caso de uso.

Quadro 21 - Detalhamento do caso de uso UC09

UC09 - Carregar exercícios a partir de textos em formato JSON gerados pelo sistema	
Descrição	É possível carregar exercícios criados anteriormente no aplicativo. Através do painel `Arquivo` é possível abrir um texto com o mesmo a mesma estrutura no formado JSON dos textos gerados pelo aplicativo.
Cenário Principal	1. Selecionar o painel `Arquivo`; 2. Selecionar a opção `Abrir`; 3. Selecionar um arquivo contendo os dados de um exercício usando o campo `Arquivo` ou colar um texto com os dados do exercício no campo `Texto Exportado`; 4. Selecionar o botão `Abrir`.
Cenário Alternativo 1	1. Se no passo 3 do cenário principal for informado um texto inválido será exibida uma mensagem de erro.
Pós-Condição	O aplicativo será focado no painel `Fábrica de Peças` com o exercício informado carregado.

O UC10 mostra como o ator `Usuário` consegue abrir exercícios existentes na lista de exemplos do aplicativo. O Quadro 22 apresenta os detalhes do caso de uso.

Quadro 22 - Detalhamento do caso de uso UC10

UC10 - Carregar exercícios da lista de exemplos do sistema	
Descrição	É possível carregar exercícios de exemplo do próprio aplicativo. Através do painel `Arquivo` é possível abrir textos de exemplo com o mesmo formado JSON dos textos gerados pelo aplicativo. Esses exemplos contêm exercícios criados para demonstrar o uso das diversas peças existentes no aplicativo.
Cenário Principal	1. Selecionar o painel `Arquivo`; 2. Selecionar a opção `Abrir`; 3. Selecionar um dos itens do campo `Lista de Exemplos`; 4. Selecionar o botão `Abrir`.
Cenário Alternativo 1	1. Se antes de se executar o passo 4 do cenário principal, o texto carregado no campo `Texto Exportado` pelo passo 3 do cenário principal for alterado de forma inválida, será exibida uma mensagem de erro.
Pós-Condição	O aplicativo será focado no painel `Fábrica de Peças` com o exercício selecionado carregado.

ANEXO C – Detalhamento das partes que formam o ambiente VisEdu-CG

Neste anexo encontram-se as descrições detalhadas das partes que formam o ambiente VisEdu-CG. Segundo Montibeler (2014a, p.27-28), o ambiente VisEdu-CG é composto por cinco painéis principais, sendo eles:

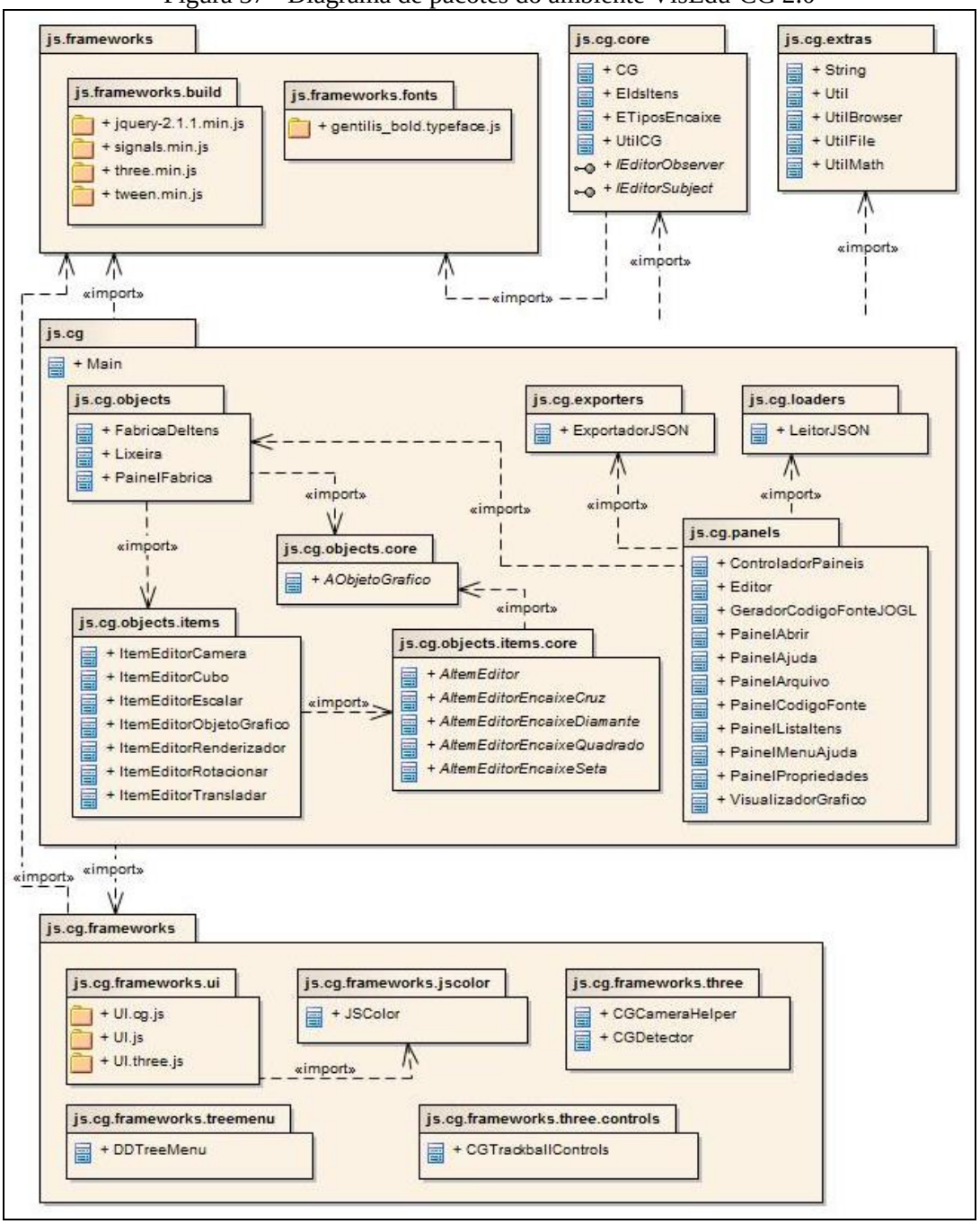
a) `Fábrica de Peças`: local onde as peças são disponibilizadas a peças para construção do exercício. Ele é composto de dois subpainéis: o `painel de montagem` e a `fábrica`;
b) `Comandos em JOGL`: painel onde o usuário visualiza o código fonte em Java que é necessário para reproduzir o resultado gráfico da peça selecionada usando a biblioteca JOGL;
c) `Lista de Peças`: painel com uma lista das peças encaixadas no exercício que permite a visualização e seleção rápida das peças existentes;
d) `Espaço Gráfico`: painel com a visualização da cena 3D formada pelo exercício;
e) `Visão da Câmera`: é a visão da cena 3D exibida no painel `Espaço Gráfico` a partir da visão da peça `Câmera`. Esse painel permite visualizar a cena 3D formada pelo exercício de um ângulo diferente, utilizando as propriedades definidas para a peça `Câmera`.

O VisEdu-CG ainda disponibiliza outros três painéis que concorrem com a visualização do painel `Fábrica de Peças`: o painel `Arquivo`, que exibe opções para exportar ou abrir exercícios; o painel `Propriedades da Peça`, que exibe as propriedades pertinentes a peça selecionada e o painel `Ajuda`, que permite consultar informações e dicas a respeito do aplicativo e seu funcionamento.

ANEXO D – Diagramas das principais classes do ambiente VisEdu-CG 2.0

A Figura 37 apresenta o diagrama contendo o conjunto de pacotes que compõem o sistema e o relacionamento existente entre eles. Nela é possível observar os *frameworks* e classes do sistema, dentro de seus respectivos pacotes.

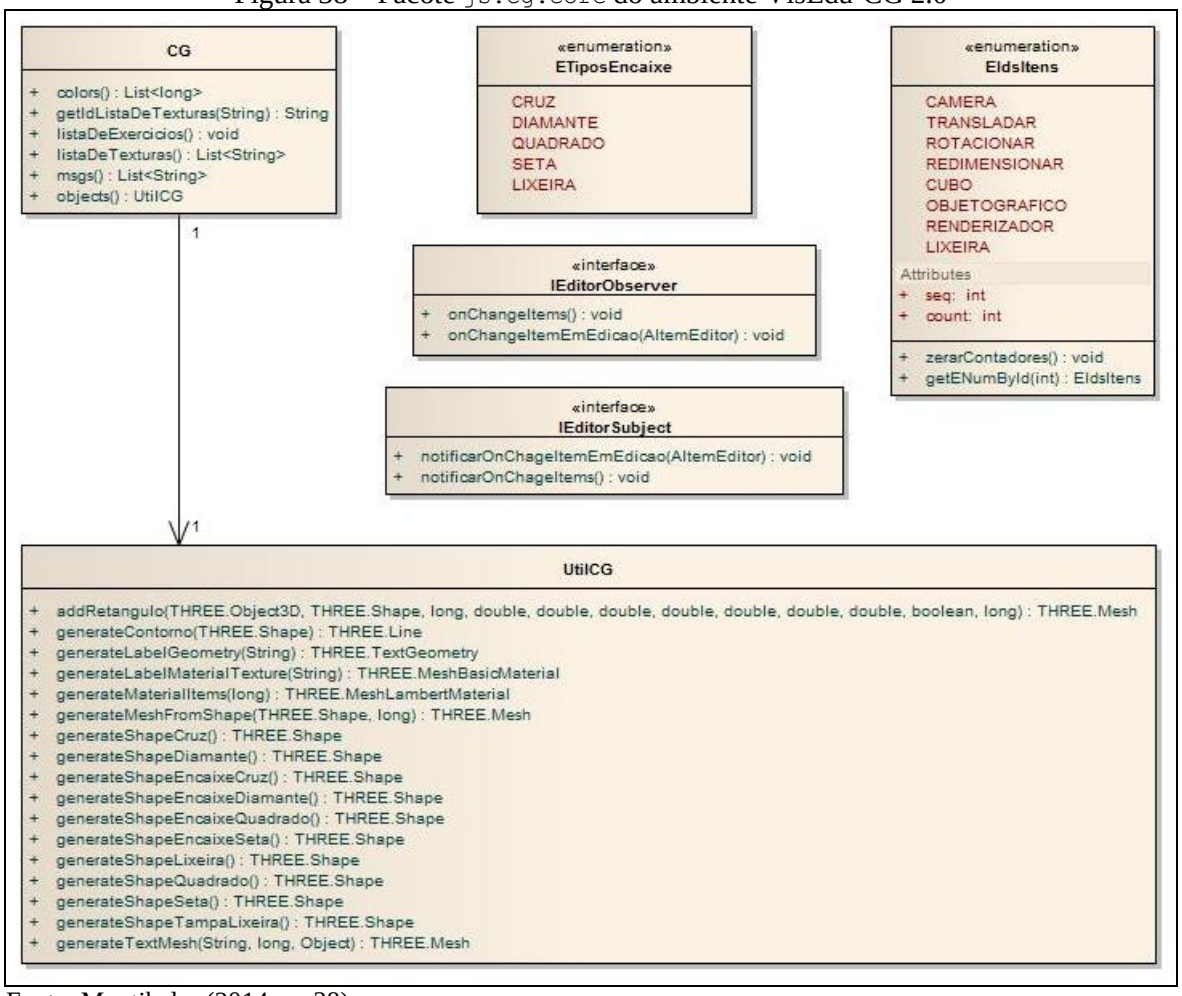
Figura 37 - Diagrama de pacotes do ambiente VisEdu-CG 2.0



Fonte: Montibeler (2014a, p.35).

Na Figura 38 são exibidas as classes do pacote `js.cg.core`. Este pacote contém as classes bases utilizadas pelas demais classes do sistema.

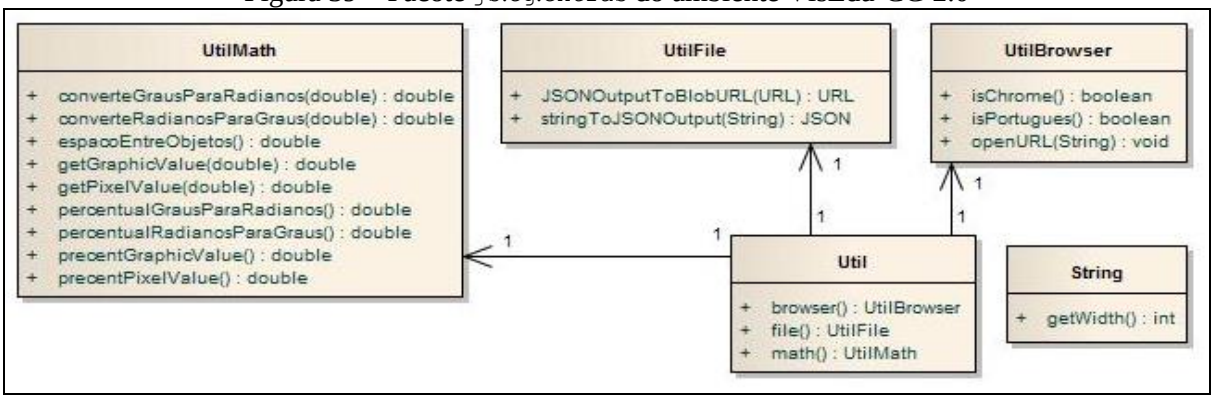
Figura 38 – Pacote `js.cg.core` do ambiente VisEdu-CG 2.0



Fonte: Montibeler (2014a, p.38).

Na Figura 39 são apresentadas as classes do pacote `js.cg.extras`. Neste pacote estão contidas classes utilitárias que não são aplicáveis somente ao sistema, ou seja, que poderiam ser reutilizadas por outros sistemas.

Figura 39 – Pacote `js.cg.extras` do ambiente VisEdu-CG 2.0



Fonte: Montibeler (2014a, p.37).

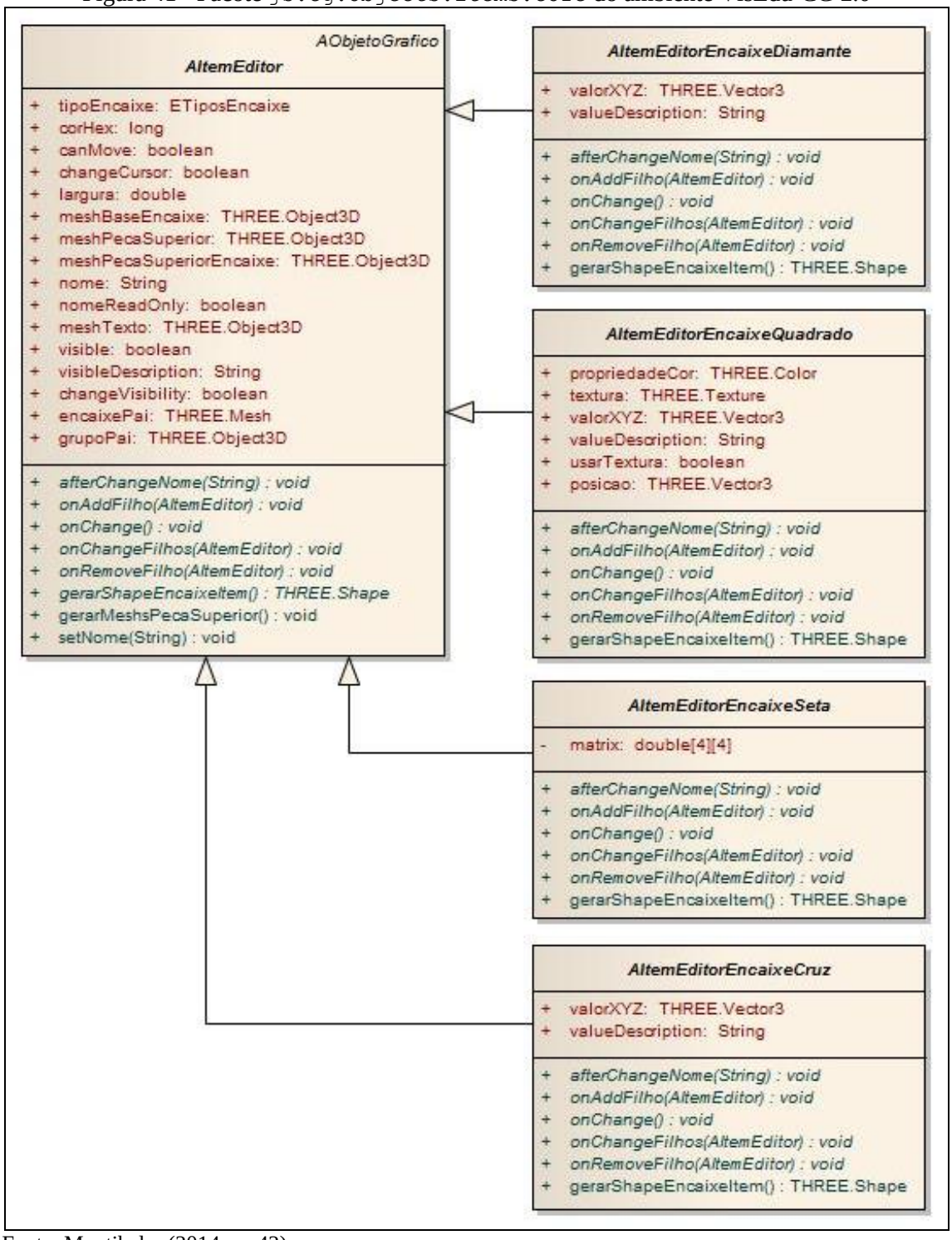
A Figura 40 apresenta as classes do pacote `js.cg.objects.core`. O pacote contém as classes bases para o controle dos objetos gráficos manipuláveis do sistema: as peças e a lixeira.

Figura 40 – Classe `AObjetoGrafico` do ambiente VisEdu-CG 2.0

Fonte: Montibeler (2014a, p.41).

Na Figura 41 pode-se visualizar as classes do pacote `js.cg.objects.items.core`. Este pacote contém as classes bases para a manipulação e renderização das peças do painel `Fábrica de Peças`.

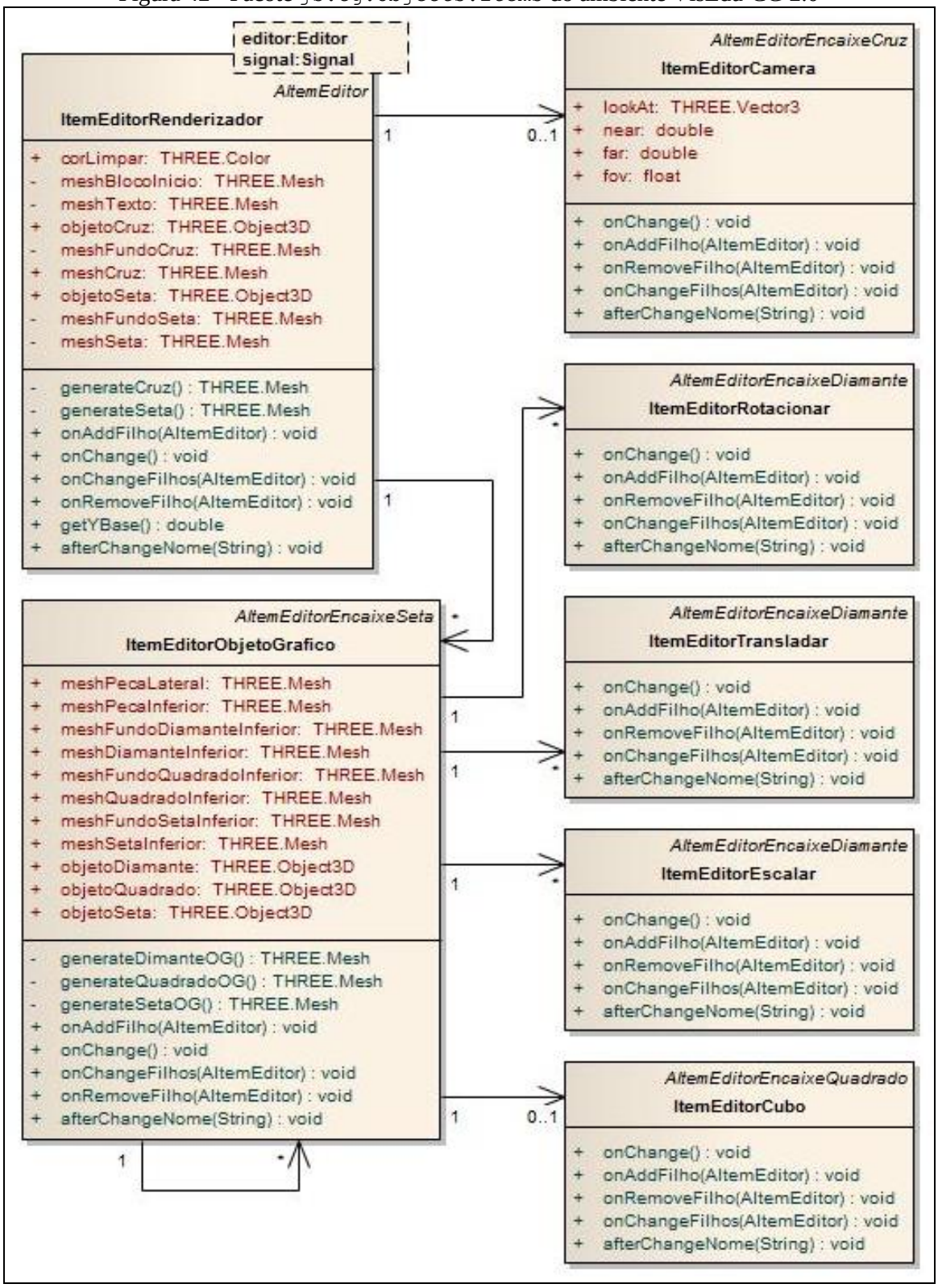
Figura 41 - Pacote `js.cg.objects.items.core` do ambiente VisEdu-CG 2.0



Fonte: Montibeler (2014a, p.42).

Na Figura 42 são apresentadas as classes do pacote `js.cg.objects.items`. O pacote contém as classes que representam cada item existente no painel `Fábrica de Peças`.

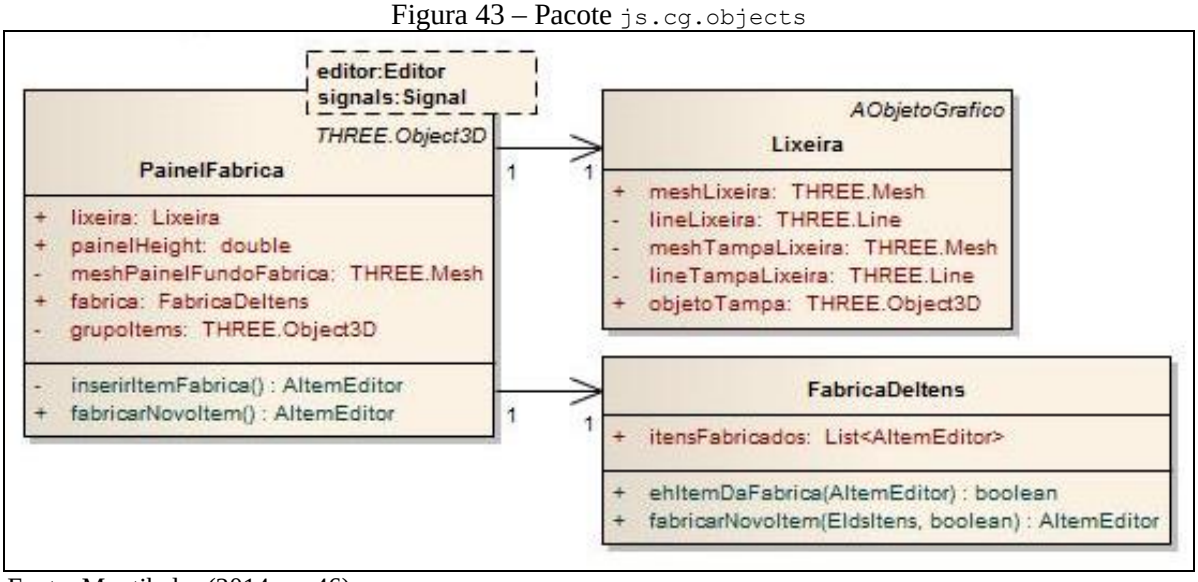
Figura 42 - Pacote `js.cg.objects.items` do ambiente VisEdu-CG 2.0



Fonte: Montibeler (2014a, p.44).

Na Figura 43 são apresentadas as classes do pacote `js.cg.objects`. O pacote contém as classes que renderizam e auxiliam no controle do painel `Fábrica`, sub-painel do painel `Fábrica de Peças`.

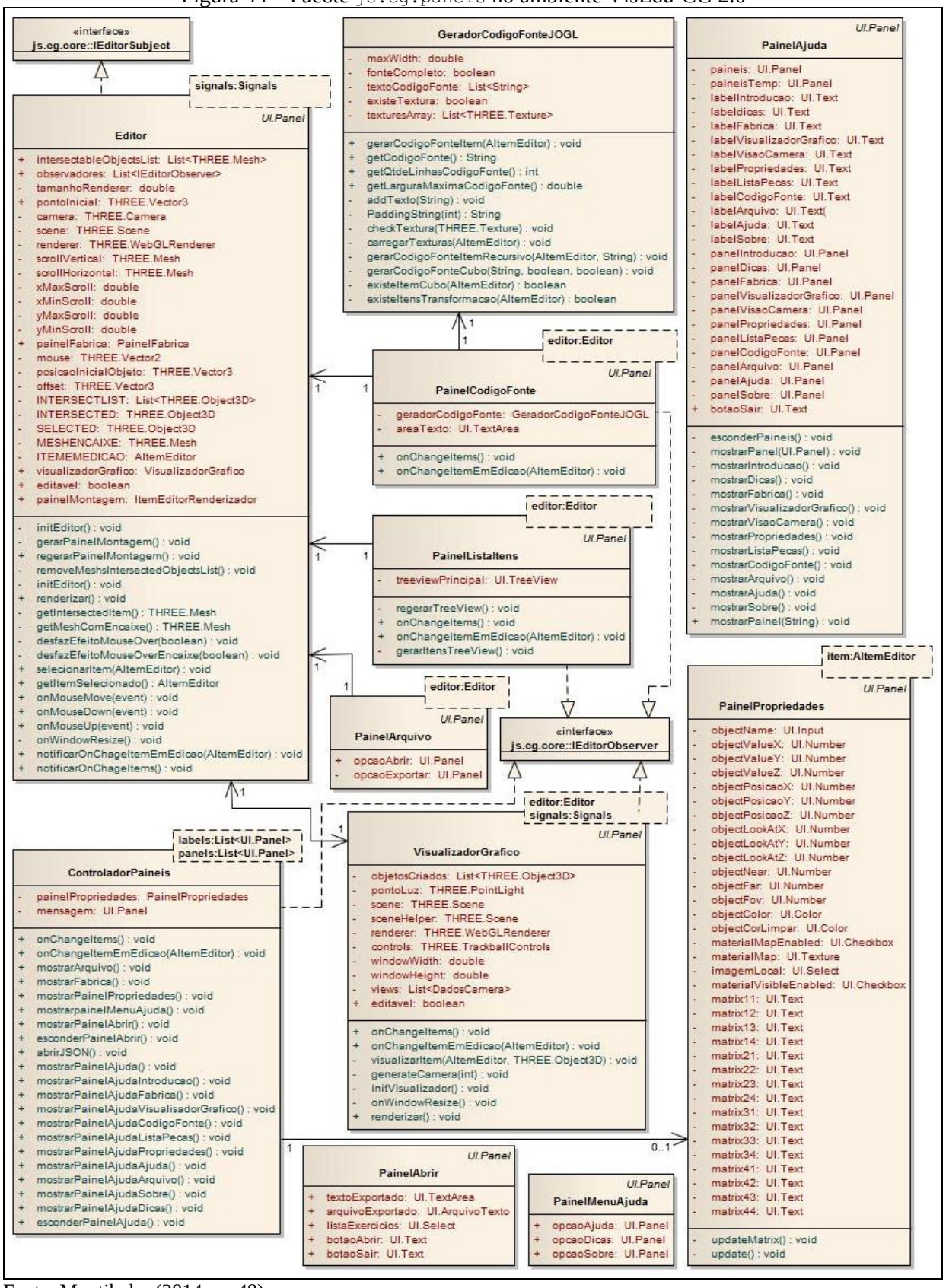
Figura 43 – Pacote `js.cg.objects`



Fonte: Montibeler (2014a, p.46).

Na Figura 44 são apresentadas as classes do pacote `js.cg.panels`. Este pacote contém as classes que representam, controlam e organizam cada painel exibido no sistema.

Figura 44 – Pacote `js.cg.panels` no ambiente VisEdu-CG 2.0



Fonte: Montibeler (2014a, p.48).