VISEDU-CG 5.0 – VISUALIZADOR DE MATERIAL EDUCACIONAL

Peterson Boni Buttenberg, Dalton Solano dos Reis – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

pbuttenberg@furb.br, dalton@furb.br

Resumo: *Este artigo apresenta o processo de desenvolvimento de uma plataforma de ensino e aprendizagem direcionada para alunos de computação gráfica. O aplicativo foi desenvolvido utilizando o motor de jogos Unity e a linguagem de programação C# através da ferramenta Visual Studio. A aplicação visa unir a jogabilidade em um formato de peças encaixáveis no estilo programação visual aos conceitos iniciais da computação gráfica. Foram realizados testes de usabilidade a partir de cenários de utilização da plataforma, desempenho verificando o consumo de memória e responsividade, no qual os resultados foram melhores em resoluções mais horizontais. A aplicação apresentou um alto consumo de memória em alguns navegadores, porém demonstra bom desempenho em ambiente desktop.*

Palavras-chave: *Computação gráfica. Unity. Transformações geométricas. Iluminação.*

1 INTRODUÇÃO

A tecnologia evolui constantemente e a cada dia, uma nova técnica é criada ou aperfeiçoada, contribuindo para o desenvolvimento de muitas áreas. Fuks et al. (2004) diz que dentre as diversas áreas do conhecimento, há um grande avanço para que ferramentas interativas possam ajudar no ensino e aprendizagem estimular a colaboração e interação entre os participantes de um curso baseado na web. Dados estatísticos comprovam que o desempenho dos alunos que fazem uso de várias metodologias de ensino-aprendizagem é melhor que os de alunos que continuam utilizando o método tradicional, baseado em aulas expositivas.

Uma das áreas que pode se beneficiar com o uso de metodologias de ensino é a computação gráfica. A Computação gráfica segundo Manssour e Cohen (2006, p. 1), "é uma área da Ciência da Computação que se dedica ao estudo e desenvolvimento de técnicas e algoritmos para a geração (síntese) de imagens através do computador". Dentro da computação gráfica um conceito muito importante é a modelagem geométrica. Modelos são usados para representar objetos do mundo real no computador. A modelagem permite que uma cena seja desenhada a partir de uma descrição de determinado processo (MANSSOUR; COHEN, 2006, p. 4).

Dessa forma, o VisEdu-CG vêm ao encontro dessas ideias, criando novas metodologias ativas para esse processo. Conforme, Reis (2020, p. 1) "o VisEdu-CG é um projeto para desenvolver uma plataforma Web que permita os alunos da disciplina de Computação Gráfica do curso de Ciências da Computação praticarem os conceitos ministrados nesta disciplina". Essa aplicação contou com o desenvolvimento de vários módulos específicos, dentre eles pode-se citar o motor de jogos, matemática, estatística, processamento de imagens, realidade aumentada e simulação.

Para que o aplicativo evolua e continue a ser aprimorado, é importante utilizar uma plataforma de desenvolvimento tridimensional que seja o mais popular possível. O Unity é a ferramenta de desenvolvimento 3D em tempo real mais utilizada no mundo e fornece ferramentas para criar jogos e publicá-los na mais ampla variedade de dispositivos (UNITY, 2020e). Com o Unity pode-se gerar código para dispositivos móveis, aplicações utilizando realidade aumentada e realidade virtual, gerar compilação para desktop para as plataformas PC, Mac e Linux. Para dispositivos móveis (iOS e Android), bem como para aplicações Web utilizando WebGL. Por esse motivo foi a tecnologia escolhida para ser feita a continuação do projeto.

Dito isto, este trabalho desenvolveu a migração das tecnologias utilizadas pelo VisEdu-CG 4.0 de Koehler (2015) para o motor de jogos Unity, tornando-o um sistema mais acessível e com maior possibilidade de modificação. Também permite futuras extensões e possibilitar a compilação para diversas plataformas. Os objetivos específicos são:

a) converter a ferramenta de visualização gráfica atual para o motor de jogos Unity;
b) apresentar uma interface de ajuda em forma de tutorial informando os passos a serem seguidos;
c) utilizar representação visual usando peças de encaixa para gerar uma cena gráfica;
d) disponibilizar as funções gráficas: câmera, transformações geométricas e iluminação.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão destacados os principais assuntos que fundamentaram o desenvolvimento do aplicativo. Primeiramente serão demonstrados os conceitos utilizados como base para o desenvolvimento como: `RenderTexture`, iluminação e câmera. A seguir será apresentada a versão anterior do sistema, o VisEdu-CG 4.0. Por fim, são apresentados os trabalhos correlatos ao desenvolvido.

2.1 RENDERTEXTURE

O `RenderTexture` é um tipo de textura especial que é criada e atualizada em tempo de execução. Segundo Unity (2020d) para usá-los, deve-se primeiro criar um `RenderTexture` e anexar o recurso a uma câmera fazendo com que a mesma renderize uma textura em vez de renderizar um ponto de vista da cena. Eles podem ser usados para implementar efeitos de renderização baseados em imagem, sombras dinâmicas, projetores, reflexos ou câmeras de vigilância (UNITY, 2020d). A Figura 1 mostra uma forma de utilização do `RenderTexture` no Unity.

Figura 1 – Utilização do RenderTexture no Unity

Fonte: Unity (2020d).

O componente pode ser usado em um material como uma textura comum ou em um `RawImage`. Conforme Unity (2020c) o `RawImage` é um componente que exibe uma imagem não interativa ou não processada. Este componente é utilizado para fins como decoração ou ícones e alterar a imagem de um script para refletir alterações em outros controles.

Como serão usadas câmeras para renderizar as texturas, o fundo poderá ser alterado para uma cor preta sólida em vez da `skybox` que é o fundo padrão do Unity. Além disso, a câmera possui a `Culling Mask`, uma propriedade utilizada para renderizar apenas camadas específicas da textura, em vez de todas elas. Para isso, deve-se selecionar a câmera que deve renderizar objetos de maneira seletiva (UNITY, 2020c).

Para simular uma visualização 2D em um ambiente 3D na plataforma de desenvolvimento Unity existe um componente chamado `Canvas`, que de acordo com Unity (2020b) todos elementos da interface com o usuário, devem ser filhos do `Canvas`. Este componente é mostrado como um retângulo na cena, tornando a visualização e o posicionamento dos elementos mais fácil. Já a renderização dos filhos do `Canvas` é feita de forma hierárquica começando do primeiro filho para o último, ou seja, se dois elementos estiverem sobrepostos, o último aparecerá em cima do anterior.

2.2 ILUMINAÇÃO

As iluminações no Unity têm o comportamento semelhante ao da luz do mundo real, sendo assim, são capazes de emitir sombras de um objeto em outras partes de si mesmo ou para outros objetos próximos. Além de acrescer um grau de profundidade e realismo das cenas, uma vez que enfatizam a escala e a posição os objetos (UNITY, 2020f). O Unity abrange quatro tipos de iluminação contidos no componente `Light`, sendo elas:

a) *Point*: atua como uma lâmpada no mundo real, ou seja, emitindo luz para todas as direções, sendo assim, a intensidade da luz diminui com a distância. Como na lei do inverso do quadrado da distância de Newton, esse tipo de iluminação tem a intensidade da luz inversamente proporcional ao quadrado da distância da fonte;

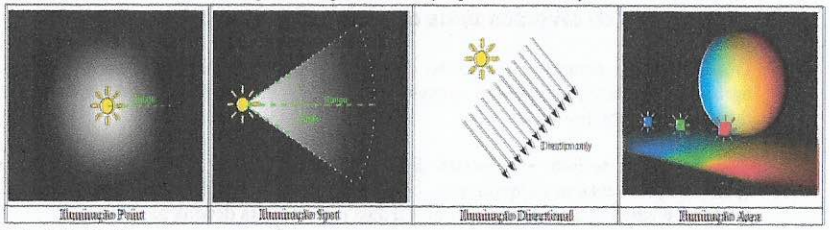
b) *Spot*: a iluminação *Spot* emite luz a partir de um único ponto em forma de cone. Essa luz é limitada por um ângulo que pode ser alterado, dessa forma, aumentando o ângulo aumenta a largura do cone de emissão de luz ocasionando o aumento da penumbra;

c) *Directional*: A iluminação *Directional* simula a luz que está sendo emitida de uma fonte infinitamente distante. É ideal para simular o sol ou até mesmo a lua, pois todas as sombras projetadas por essa luz são paralelas. Essa luz não possui um ponto de início de iluminação, ela ilumina toda a cena, sendo assim, o componente desse tipo de iluminação pode ser colocado em qualquer lugar da cena;

d) *Area*: emite luz na cena a partir de um plano retangular com largura e altura definidas. Pode-se simular qualquer tipo de fonte de luz com áreas retangulares, mas apenas para um dos lados do retângulo. Esse tipo de iluminação se torna mais suave que os outros por iluminar um objeto de várias direções diferentes.

A Figura 2 apresenta todos os tipos de iluminação e o alcance ou direção que cada iluminação possui.

Figura 2 – Tipos de iluminação presentes no Unity



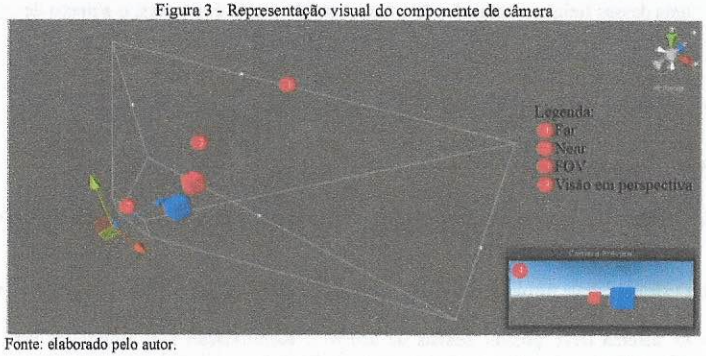
Fonte: Unity (2020f).

2.3 CÂMERA

A câmera representa a visão do espaço de cena do jogador, como o jogador vê o mundo. Uma câmera no mundo real, ou mesmo um olho humano, à medida que os objetos vão ficando mais longe se tem a impressão de que eles estão menores, este efeito chama-se perspectiva, no qual é um termo muito conhecido na computação gráfica. Entretanto, existe outra forma de utilizar a câmera, caso não seja necessária uma visualização em perspectiva, isto é, em situações que a cena a ser visualizada não tenha que alterar seu tamanho de acordo com a distância, nesse caso é utilizado a câmera ortográfica (UNITY, 2020a).

Para garantir que uma câmera em perspectiva tenha a visualização de objetos próximos serem maiores e objetos distantes serem menores, a câmera tem um formato similar a uma pirâmide, porém com as duas bases cortadas, nas quais são representadas pela propriedade `near` (corte no topo da pirâmide) e `far` (corte na base da pirâmide). Essa forma é chamada de *frustum* e o ângulo entre a parte superior e inferior do ápice é conhecido com FOV (*field of view*). A propriedade `FOV` é usada para aumentar e diminuir o campo de visão, sendo representada em graus. A Figura 3 retrata a representação visual do componente de `Camera` no Unity (UNITY, 2020a).

Figura 3 - Representação visual do componente de câmera



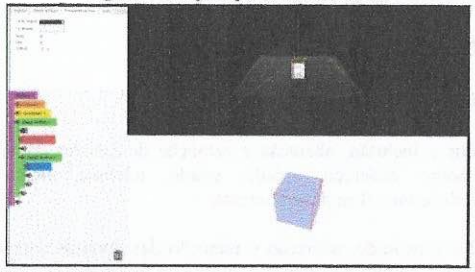
Fonte: elaborado pelo autor.

2.4 VERSÃO ANTERIOR DO SISTEMA

Como afirma Koehler (2015), o VisEdu-CG 4.0 teve um aumento significativo em sua programação para que fosse concretizada a proposta de reestruturar o visualizador de material educacional e com isso permitiu que a integração entre o motor de jogos fosse mais fácil. A visualização do objeto gráfico e do espaço gráfico foi melhorada com a remoção do painel de Lista de Peças e Comandos JOGL. A cor preta foi atribuída a esse espaço para melhor a visualização do objeto na cena.

A versão 4.0 da plataforma VisEdu-CG passa a ter como foco na tela principal os painéis de Visão da Câmera e Espaço Gráfico. A câmera mostra o ângulo de visão a partir de onde o objeto gráfico será visualizado. Em uma cena 3D, a câmera pode ser movimentada para que seja possível visualizar todos os lados do objeto, já em uma cena 2D a câmera permanece estática. O Espaço Gráfico concebe a visualização do objeto gráfico escolhido que pode ser modificado de acordo com a posição da câmera. Para a criação e manutenção das abas, foram usadas tecnologias como HTML, CSS, Javascript e JQuery (KOEHLER, 2015). Na Figura 4 é possível visualizar a interface atual da aplicação.

Figura 4 - Tela principal do VisEdu-CG 4.0



Fonte: Koehler (2015).

Koehler (2015) realizou uma integração de um visualizador de material educacional com um motor de jogos HTML5, no qual foi proposto como objetivo do trabalho. A biblioteca Three.js foi incluída no projeto e, diante disso, o motor de jogos teve que ser adaptado para suportar a mesma e assim fornecer a capacidade, competência e eficácia necessária para uma visualização tridimensional adequada. A biblioteca Three.js se manteve estável e performática mesmo inserindo um grande número de elementos gráficos.

2.5 TRABALHOS CORRELATOS

A seguir, são apresentados quatro trabalhos correlatos com características semelhantes aos principais objetivos do estudo proposto. O quadro 1 detalha o AduboGL (ARAÚJO, 2012), uma aplicação que colabora no aprendizado de computação gráfica. O quadro 2 descreve o AduboGL 2.0 (SCHRAMM, (2012), sistema voltado para iPad e auxilia no aprendizado de conceitos de computação gráfica. O quadro 3 expõe o VisEdu-CG (MONTIBELER, 2014) ,um projeto desenvolvido em uma plataforma web que possibilita aos alunos de computação gráfica, utilizarem os conceitos aprendidos na disciplina. O quadro 4 aborda o VisEdu-CG 3.0 (NUNES, 2014), uma extensão do VisEdu-CG.

Quadro 1 – AduboGL

Referência	Araújo (2012).
Objetivos	Colaborar no aprendizado de computação gráfica com foco nas transformações geométricas.
Principais funcionalidades	Aplicação didática usando a biblioteca Open GL, é uma aplicação voltada ao aprendizado da computação gráfica com foco nas transformações geométricas, utilizando a biblioteca OpenGL para montar o cenário 2D presente na aplicação e tendo seu resultado em um espaço 3D.
Ferramentas de desenvolvimento	Linguagem de programação C++ no ambiente Microsoft Visual C++ 2010 Express. A biblioteca OpenGL 4.2 utilizada para o desenvolvimento de aplicações gráficas 2D e 3D e a biblioteca V-ART que é um framework utilizado para facilitar a criação de ambientes em 3D.
Resultados e conclusões	Os resultados obtidos na análise de performance, nota-se que conforme aumenta a complexidade da ação e a quantidade de peças a serem comparadas no código, maior o consumo de tempo. Já nos resultados obtidos na análise de usabilidade, a autora comenta que não foi possível aplicar a ferramenta em sala de aula mas foram escolhidos 10 alunos de diversos semestres para testar o sistema. Em geral, acharam simples de entender.

Fonte: elaborado pelo autor.

Nunes (2014), que teve como objetivo estender o VisEdu-CG, incluiu novas funcionalidades à aplicação, deixando-a mais completa e aumentando ainda mais o entendimento dos conceitos de computação gráfica. Ele se propôs a incluir o tipo de objeto polígono e spline 3D, adicionar na representação gráfica o uso de iluminação.

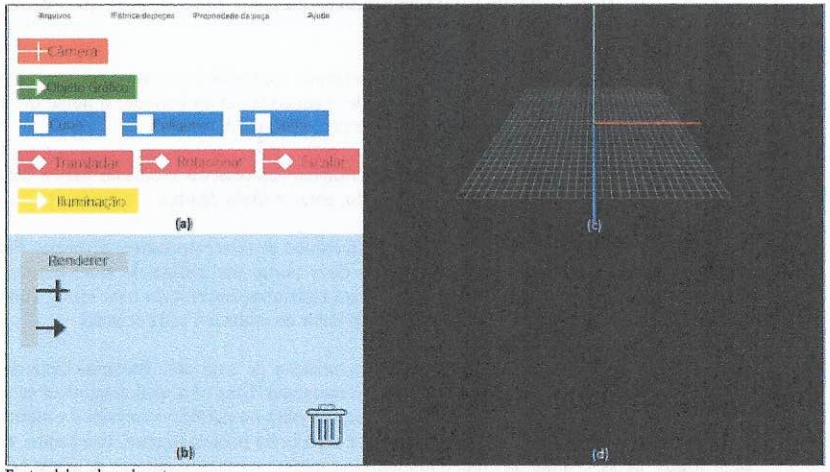
3 DESCRIÇÃO

Nesta seção será detalhado o que foi desenvolvido e uma visão geral da plataforma. Na primeira seção é destacada uma visão geral da plataforma contemplando os painéis e suas formas de uso. Já na segunda seção são apresentadas as principais técnicas utilizadas para o desenvolvimento da aplicação.

3.1 VISÃO GERAL DA PLATAFORMA

A plataforma desenvolvida tem a tela principal dividida em quatro painéis, um que armazena as peças a serem escolhidas pelo usuário, chamado de `Fábrica de peças` (Figura 7, a), esse painel também contém um menu onde é possível acessar as `Propriedades das peças` (Figura 8) ou retorna para o menu anterior. Outro painel, denominado como `Renderer` (Figura 7, b), é usado para o encaixe ou exclusão das peças. Já o próximo painel serve para representar graficamente o resultado das peças encaixadas, nomeado de `Ambiente gráfico` (Figura 7, c). E, por fim, o `Visualizador` (Figura 7, d) um painel que demonstra apenas a forma geométrica e as alterações feitas na mesma.

Figura 7 - Tela principal da plataforma



Fonte: elaborador pelo autor.

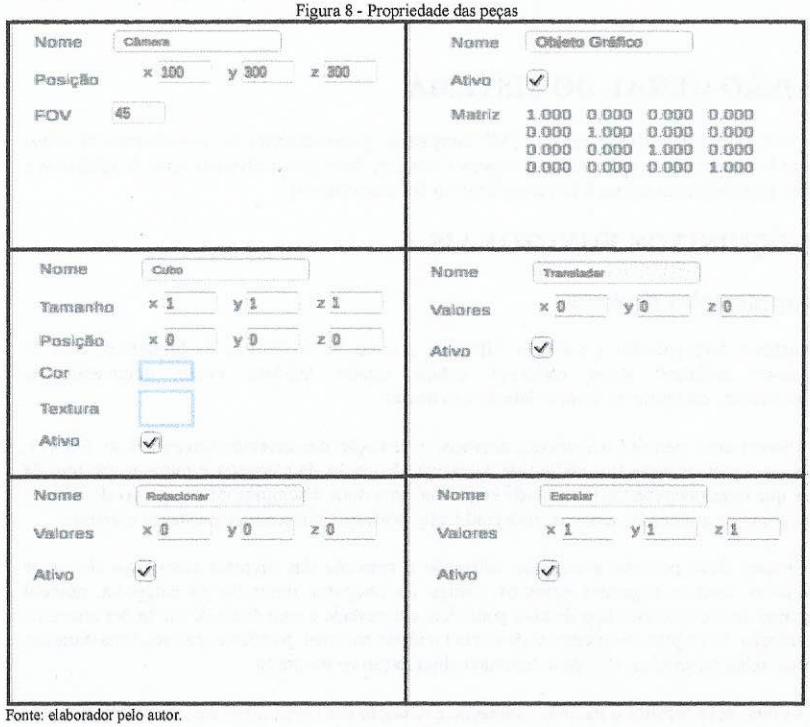
Ao abrir a plataforma é demonstrada a seguinte mensagem: "Deseja realizar o tutorial para aprender os conceitos básicos da plataforma?", que pode ser respondido com os botões `Sim` ou `Pular tutorial`. Caso o botão `Sim` seja pressionado é iniciado um tutorial que simula a execução de um usuário, arrastando algumas peças até as posições adequadas e demonstrando os resultados. São executados sete passos e a cada passo é exibida uma mensagem descrevendo o que será feito. É possível pular o tutorial a cada passo e se o botão `Pular tutorial` for pressionado o sistema pode ser utilizado normalmente.

A `Fábrica de peças` contém as peças que podem ser deslocadas até o `Renderer`. São sete o total de peças que podem ser encaixadas: a primeira é a `Câmera`, a segunda `Objeto Gráfico`, a terceira `Cubo`, a quarta, quinta e sexta são as peças de transformação geométrica, são elas: `Transladar`, `Rotacionar` e `Escalar` respectivamente, e a última `Iluminação`. As peças `Polígono` e `Spline` aparecem na fábrica de peças, mas não podem ser arrastadas para o `Renderer`.

Cada uma das peças possui suas propriedades específicas, como pode ser observado na Figura 9. A Câmera contém o nome (propriedade comum para todas as peças), a posição e FOV (*Field of View*, usado para alterar o campo de visão da câmera). Já a peça `Objeto Gráfico` possui nome e ativo (propriedade que quase todas as peças dispõem, usada para ativar ou desativar a funcionalidade da peça) e uma matriz. O Cubo cujas propriedades são: nome, tamanho

(para dimensionar o objeto), posição (alterado em x, y e z), cor, textura e ativo. As transformações geométricas possuem as mesmas propriedades: nome, valores (x, y, z) e ativo, porém a propriedade valores tem um comportamento específico para cada peça. Para a peça `Transaldar` é alterada a posição do cubo, já a peça `Rotacionar` ajusta a rotação e, por fim, a peça `Escalar` é responsável pela modificação da escala do cubo.

Figura 8 - Propriedade das peças



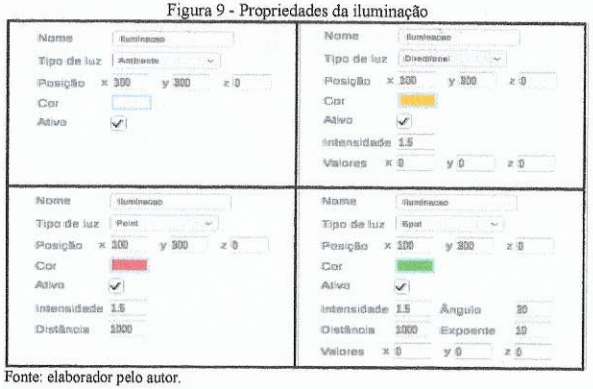
Fonte: elaborador pelo autor.

A peça `Iluminação` compreende um conjunto de propriedades diferente para cada tipo de iluminação, sendo eles: `Ambiente`, `Directional`, `Point` e `Spot`. Na iluminação ambiente existem propriedades que são padrões para todos os tipos de iluminação, sendo elas o nome, tipo de luz, posição (eixos x, y e z), a cor da iluminação e uma caixa de seleção chamada de ativo (habilita ou desabilita a luz). A iluminação `Directional`, além das propriedades já citadas, também possui a intensidade e valores, no qual a intensidade tem a finalidade de aumentar ou diminuir a força da luz emitida e valores nos eixos x, y e z com o intuito de rotacionar a iluminação para a direção desejada. No que se refere, a iluminação `Point`, nota-se uma propriedade que a diferencia das demais, chamada de distância, que possibilita alterar a distância da iluminação em relação à forma gráfica selecionada. E por fim, a iluminação `Spot`, que tem como diferencial a propriedade ângulo, com a finalidade de alterar o raio atingido pela iluminação. As propriedades da iluminação podem ser vistas na Figura 9.

O painel `Renderer` integra os *slots* para encaixe das peças, a lixeira para a exclusão e a peça `Renderer` que é pai de toda a árvore de peças e tem sua posição fixa (não pode ser movida). Ao clicar na peça `Renderer`, pode-se modificar a forma de visualização no `Ambiente gráfico` para 2D no qual a padrão é 3D. Quando alterada a forma de visualização para 2D, o eixo z das propriedades das peças é desabilitado permitindo apenas incluir valores nos eixos x e y. Em relação ao `Ambiente gráfico`, nele é possível observar o `Gizmo` que representa os eixos x, y e z, juntamente com uma grade para dar uma noção de posicionamento e dimensão aos objetos a serem visualizados. Esses objetos

podem ser o cubo, a câmera ou as iluminações. E finalmente, o `Visualizador` que tem o propósito mostrar a cena gráfica definida no Render.

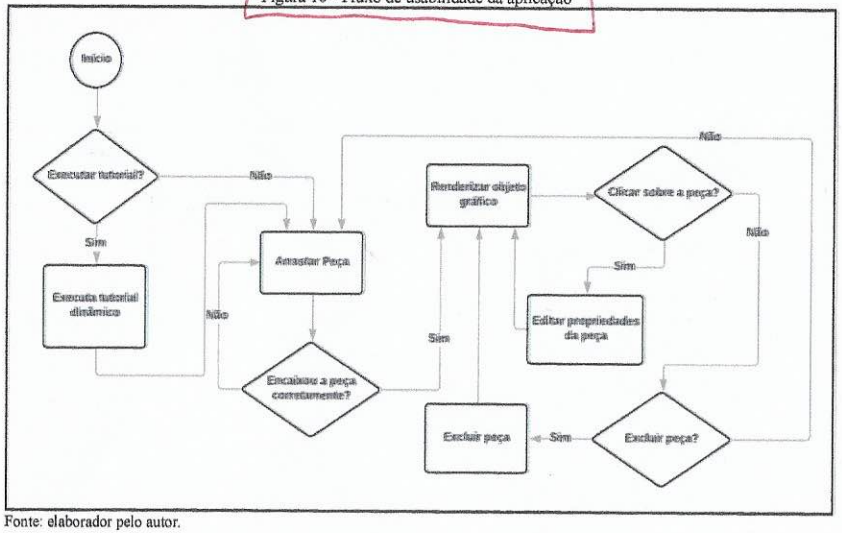
Figura 9 - Propriedades da iluminação



Fonte: elaborador pelo autor.

Ao selecionar uma peça e arrastá-la para o `Renderer`, se a peça for encaixada no local correto, então ela ficará presa ao *slot*, caso contrário ela voltará para sua posição inicial. Além disso, depois que o usuário tiver uma árvore de peças já encaixadas e alguma delas não se fizer mais útil, é possível excluir ela arrastando-a sobre a lixeira e soltando-a. Quanto ao `Ambiente gráfico`, para visualizar melhor o que está sendo demonstrado pode-se rotacionar a imagem para qualquer direção, pressionando o botão esquerdo do mouse, segurando e arrastando para a posição desejada. Assim como, o *scroll* do mouse possibilita aproximar e distanciar a visibilidade do painel. Na Figura 10 pode ser visto o fluxo de usabilidade da aplicação.

Figura 10 - Fluxo de usabilidade da aplicação



Fonte: elaborador pelo autor.

3.2 IMPLEMENTAÇÃO

Para o desenvolvimento do aplicativo, na parte gráfica foi utilizado o motor de jogos Unity e a linguagem de programação C# no ambiente de desenvolvimento Visual Studio 2017 para a implementação dos scripts. Grande parte dos objetos 3D utilizados na aplicação foram criados em uma plataforma externa e importadas para Unity. Objetos esse como as peças presentes na `Fábrica de peças`, as peças de encaixe presentes no `Renderer`, o `Gizmo` e todas as representações de iluminação.

O tutorial dinâmico tem como principal característica a movimentação das peças dinamicamente, sem nenhuma interação do usuário. Isso é possível com a utilização do método `Lerp` que contem três parâmetros, sendo o primeiro a posição inicial do objeto, o segundo a posição final e por fim, um parâmetro que serve como interpolador com valores de zero a um, sendo zero mais próximo da posição inicial e um mais próximo da posição final. O método é chamado repetidamente até que a peça atinja a posição esperada. Para verificar se a peça chegou a posição final é utilizado o método `Distance` que retorna a distância entre duas posições passadas por parâmetro, caso essa distância seja zero então a peça chegou a posição desejada. A partir disso, são feitas as chamadas dos métodos simulando o encaixe das peças e a criação das representações gráficas.

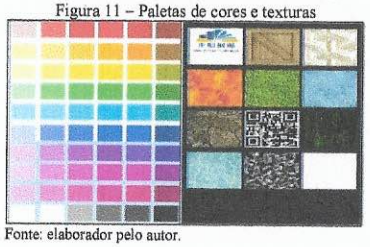
Ao clicar em uma peça para arrastá-la, o método `OnMouseDown`, padrão do Unity, é solicitado, guardando a posição inicial da peça, além disso, dentro deste mesmo método, são gravadas em uma lista, todas as peças que foram encaixadas, juntamente com suas posições atualizadas. Após isso, assim que a peça sai da sua posição inicial é criada uma nova peça, idêntica a peça original na mesma posição e com todas suas propriedades, para isso foi utilizado o método `Instantiate`. Enquanto a peça é arrastada, o método `OnMouseDrag` é chamado, e em seu escopo é atualizada a posição atual da peça em relação à cena, através de uma conversão feita pela função `ScreenToWorldPoint` a qual, transforma a posição do espaço da tela (definida por pixels) no espaço da cena (relativa ao sistema de coordenadas do mundo). Quando a peça é posicionada no *slot* e o botão do mouse é solto, o evento `OnMouseUp` é chamado e em seu escopo é verificado se a peça está próxima do *slot* correto, por meio do método `PodeEncaixar`.

Cada peça possui uma representação no `Ambiente gráfico` e a cada `GameObject`, que representa a peça na cena do Unity, é adicionado o script `Controller` que, por sua vez, é o script responsável pela maior parte das interações relacionadas às peças. Para implementar a visualização da câmera, foi incluído um `GameObject` do tipo `Camera`, na mesma posição do objeto de representação gráfica da câmera. Já para demonstrar apenas a forma gráfica selecionada no `Visualizador` foi acoplado o `Layer` da forma selecionada à propriedade `Culling Mask` do objeto `Camera`, possibilitando demonstrar apenas objetos com o `Layer` escolhido.

A peça `Objeto Gráfico` é pai dos objetos de formas (`Cubo`), transformações geométricas (`Transladar`, `Rotacionar`, `Escalar`) e iluminações. Sendo assim, cada vez que um objeto desse tipo é selecionado, ele gera toda uma hierarquia de objetos consigo, e quando é criada uma cópia do pai, utilizando o método `Instantiate`, também são copiados todos os filhos com suas propriedades e estados salvos.

Para implementação do cubo que é visualizado no `Ambiente gráfico`, foi utilizado um objeto `Cube` do menu `3D Object` nativo do Unity. Ele é criado antes mesmo da peça ser encaixada, mas com o `MeshRenderer`, que é o componente que processa os dados geométricos no Unity, desabilitado. O `MeshRenderer` só é habilitado quando a peça for encaixada. Acrescenta-se também, a implementação das propriedades do cubo, na qual uma delas é a cor. Foi feita uma paleta de cores, com 60 cores diferentes, cada umas delas é um `GameObject` contendo uma cor atribuída ao mesmo. Todos esses `GameObjects` que representam as cores detêm o script `SelecionaCor` vinculado a eles. Ao clicar sobre uma cor, o método `OnMouseDown` é invocado, e em sua implementação é feita a atribuição da cor selecionada ao `Cubo`. Há também a propriedade de textura, que segue a forma de implementação da propriedade de cor, mas ao invés de atribuir a cor, é atribuída a textura selecionada ao cubo. A Figura 11 mostra a paleta de cores e texturas.

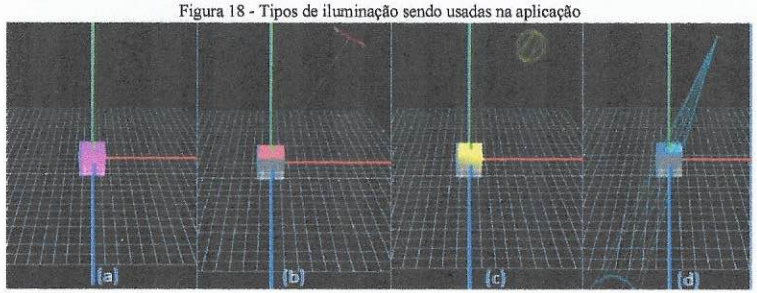
Figura 11 – Paletas de cores e texturas



Fonte: elaborador pelo autor.

Na Figura 18 são apresentados os tipos de iluminação e suas funcionalidades. As capturas de imagem foram feitas do mesmo ângulo para demonstrar as diferencias entre cada tipo de iluminação. Também são usadas cores de luz diferentes para visualizar o efeito de cada uma sobre o cubo.

Figura 18 - Tipos de iluminação sendo usadas na aplicação



Fonte: elaborado pelo autor.

O posicionamento das peças na `Fábrica de peças`, assim como a geração de cópia das peças ao serem arrastadas foi bem sucedido, como pode ser visto nas Figura16 e Figura 17. A captura da imagem foi obtida no momento em que se está deslocando a peça `Rotacionar`, demonstrando a cópia da mesma sendo gerada na posição original.

A primeira implementação dos encaixes das peças no painel `Renderer` foi desenvolvida utilizando `Colliders`, quando uma peça era deslocada do `Ambiente gráfico` até o *slot* adequado, verificava-se se o `Collider` que recebeu a colisão era o correto e então a peça era encaixada. Mas ocorreram muitos conflitos entre os colisores, por estarem muito próximos uns aos outros, por esse motivo essa opção foi descartada. Portanto, a maneira utilizada para fazer o encaixe das peças nos *slots* foi mapeando a posição das peças na cena. Toda vez que uma peça é solta na cena, é atualizada uma lista que guarda a posição (no eixo y do `GameObject`) atual de todos *slots*, após isso é feita uma verificação se o *slot* é compatível com a peça e se a peça está próxima, só então é feito o encaixe. Entretanto, os resultados obtidos nos encaixes das peças no painel `Renderer` foram bem sucedidos, assim como a exclusão da peça na lixeira, como pode ser observado na Figura 17.

Durante o desenvolvimento do trabalho foi identificado com o orientador, que para o melhor aprendizado dos alunos sobre os conceitos básicos de computação gráfica, alguns tipos de iluminação deveriam ser implementados. A Figura 18 mostra todos os tipos de iluminação da aplicação em uso, sendo eles: `Ambiente` (a), `Directional` (b), `Point` (c) e `Spot` (d). Todos os tipos de iluminação tiveram um melhor comportamento quando utilizadas individualmente. Em especial, a iluminação ambiente, quando utilizada com outros tipos de iluminação acaba tendo a emissão de luz muito mais forte, mesmo com uma intensidade baixa, faz com que as demais iluminações usadas simultaneamente a ela sejam ofuscadas, não representando adequadamente suas funcionalidades. Já a iluminação `Spot` demonstrou uma falha na sua representação gráfica ao ser alterado o ângulo da iluminação. Como pode ser visto na Figura 18 (d), foi diminuído o ângulo da iluminação e as arestas que compõem a representação gráfica tiveram sua espessura diminuída. Assim como, ao aumentar o ângulo, a espessura das arestas também aumenta e faz com que a visualização no `Ambiente gráfico` fique muito poluída e confusa, como é demonstrado na Figura 19.

Figura 19 – Aumento das arestas da iluminação Spot

Fonte: elaborador pelo autor.

4.2 TESTE DE DESEMPENHO

Para validar a responsividade e operacionalidade da plataforma, foram feitos testes gerando executáveis para web, desktop e mobile. O critério de análise para mensurar o desempenho da plataforma, foi o consumo de memória, verificando o gerenciador de tarefas do Windows em tempo real. Os testes web foram feitos localmente, utilizando os navegadores Google Chrome, Mozilla Firefox e Microsoft Edge. Já os testes desktop foram realizados em um PC (Windows).

Os cenários de testes são semelhantes aos usados no desenvolvimento do VisEdu 4.0 por Koehler (2015). Foram criados quatro cenários de teste utilizando as peças `Câmera`, `Objeto Gráfico`, `Cubo`, `Transladar`, `Rotacionar`, `Escalar` e `Iluminação`, cada cenário com uma quantidade de peças diferente, como é demonstrado na Tabela 1.

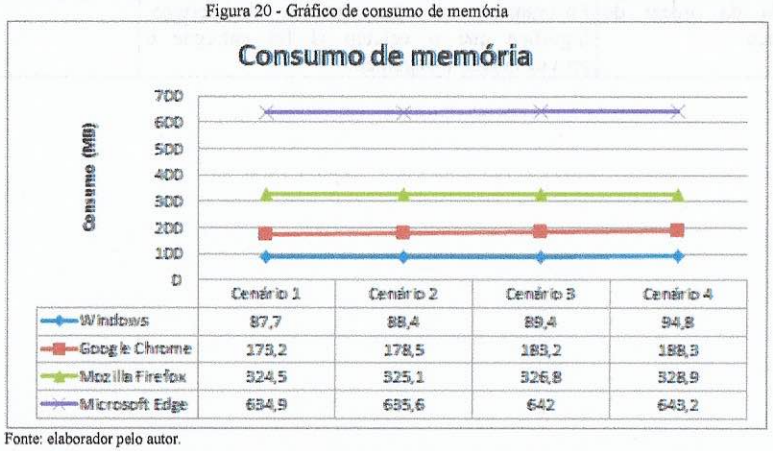
Tabela 1 – Cenários de teste

Peça	Cenário 1	Cenário 2	Cenário 3	Cenário 4
Câmera	1	1	1	1
Objeto Gráfico	1	2	4	6
Cubo	1	2	4	6
Transladar	1	2	4	6
Rotacionar	1	2	4	6
Escalar	1	2	4	6
Iluminação	1	2	4	6
	7	13	25	37

Fonte: elaborado pelo autor.

No que se refere ao consumo de memória, todos os testes realizados mantiveram o uso da memória constante, tendo um aumento de memória não muito significativo entre os cenários. Foi identificado que a aplicação desktop teve melhor desempenho que todos os navegadores. Dos navegadores testados, o Google Chrome teve o menor consumo de memória. Já o Mozilla Firefox teve quase o dobro de consumo se comparado com o Google Chrome, porém muito melhor que o Microsoft Edge, que teve o pior consumo de memória. A Figura 20 mostra o gráfico de consumo de memória feito a partir dos cenários testados.

Figura 20 - Gráfico de consumo de memória



Fonte: elaborador pelo autor.

No que se refere a responsividade, foi identificado que para algumas resoluções a plataforma não fica corretamente emoldurada e nesses casos, permite ao usuário acessar algumas funcionalidades que podem ocasionar erros ou falhas no sistema. Em resoluções mais verticais como a 1280 x 720 a aplicação apresenta o enquadramento correto.

5 CONCLUSÕES

O principal objetivo deste trabalho era converter a ferramenta VisEdu-CG 4.0 de Koehler (2015), para o motor de jogos Unity, e o mesmo cumpre o estabelecido. Apesar de apresentar problemas de visualização em alguns objetos, como por exemplo, a iluminação spot, a plataforma teve resultado satisfatório na construção de cenários contendo conceitos básicos de computação gráfica, como as transformações geométricas e também em conceitos com maior complexidade como é o caso das iluminações.

Outro objetivo específico teve como proposta apresentar uma interface de ajuda em forma de tutorial informando os passos a serem seguidos. Este objetivo foi alcançado, pois o tutorial monta uma cena demonstrando os principais conceitos da ferramenta dinamicamente. Um ponto que pode ser melhorado é forma que as mensagens demonstradas a cada passo são exibidas.

O terceiro objetivo, utilizar representação visual usando peças de encaixa para gerar uma cena gráfica, foi atingido. As peças importadas de uma ferramenta de criação de modelos 3D se comportaram adequadamente no Unity e os encaixes das peças nos *slots* foi bem sucedido. Quase todas peças tiveram suas representações gráficas efetuadas, com exceção das peças *spline* e polígono, o que leva ao quarto objetivo de disponibilizar as funções gráficas: câmera, transformações geométricas e iluminação. O objetivo foi atingido, porém faltaram algumas propriedades da câmera que são citadas como sugestões de trabalhos futuros. O objeto gráfico da câmera não permite movimentar certas partes necessárias para desenvolver tais propriedades.

Por fim, conclui-se que esse trabalho alcançou, ainda que parcialmente, os resultados esperados, faltando apenas o objetivo específico de apresentar uma interface de ajuda em forma de tutorial informando os passos a serem seguidos. Os resultados foram satisfatórios, mesmo contendo um nível de consumo de memória significativo em alguns navegadores. Porém o browser Google Chrome se mostrou bem estável e com bom consumo de memória. O melhor desempenho visto nos testes aplicados foi no ambiente desktop no sistema operacional Windows. Por este motivo, o maior diferencial da utilização do Unity para esse projeto, é a possibilidade de serem gerados executáveis, não só para web, mas também para desktop, mobile e vários consoles de jogos. Como sugestões de trabalhos futuros indica-se a: (i) desenvolver a função de *look at*, *near* e *far* da câmera; (ii) implementar as funcionalidades das peças polígono e *spline*; (iii) ajustar a hierarquia do objeto gráfico para que possa ser encaixado um objeto gráfico filho; (iv) criar a exportação do cenário construído; (v) desenvolver um painel de ajuda.

REFERÊNCIAS

ARAÚJO, Luciana Pereira. **AduboGL – Aplicação didática usando a biblioteca OpenGL**. 2012. 76 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Fundação Universidade Regional de Blumenau, Blumenau, 2012. Disponível em: https://bu.furb.br//docs/MO/2012/350348_1_1.pdf. Acesso em: 3 fev. 2020.

FUKS, Hugo et al. O modelo de colaboração 3C no ambiente AulaNet. **Informática na Educação: Teoria e Prática**, Porto Alegre, v. 7, n. 1, p. 25-48, 2004. Disponível em: https://www.seer.ufrgs.br/InfEducTeoriaPratica/article/view/4938/2941. Acesso em: 19 fev. 2020.

KOEHLER, William Fernandes. **VisEdu-CG 4.0:** visualizador de material educacional. 2015. 90 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Fundação Universidade Regional de Blumenau, Blumenau, 2015. Disponível em: < http://dsc.inf.furb.br/arquivos/tccs/monografias/2015_1_william-fernandes-koehler_monografia.pdf>. Acesso em: 20 mar. 2020.

MANSSOUR, Isabel H.; COHEN, Marcelo. **Introdução à computação gráfica.** Revista de Informática Teórica e Aplicada, Rio Grande do Sul, v. 13, n. 2, p. 1 - 25, 2006. Disponível em: https://www.inf.pucrs.br/manssour/Publicacoes/TutorialSib2006.pdf. Acesso em: 22 fev. 2020.

MONTIBELER, James Perkison. **VisEdu-CG**: Aplicação didática para visualizar material educacional, módulo de computação gráfica. 2014. 106 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Fundação Universidade Regional de Blumenau, Blumenau, 2014. Disponível em: <http://dsc.inf.furb.br/arquivos/tccs/monografias/2014_1_james-perkison-montibeler_monografia.pdf>. Acesso em: 20 abr. 2020.

NUNES, Samuel Anderson. **VisEdu-CG 3.0**: Aplicação didática para visualizar material educacional, módulo de computação gráfica. 2014. 89 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Fundação Universidade Regional de Blumenau, Blumenau, 2014. Disponível em: < http://dsc.inf.furb.br/arquivos/tccs/monografias/2014_1_samuel-anderson-nunes_monografia.pdf>. Acesso em: 20 abr. 2020.

REIS, Dalton Solano. **Grupo de Pesquisa em Computação Gráfica, Processamento de Imagens e Entretenimento Digital**. [201-], Disponível em: http://gcg.inf.furb.br/?page_id=1147. Acesso em: 16 fev. 2020.

SCHRAMM, Elizandro José. **AduboGL ES 2.0**: Aplicação didática usando a biblioteca OpenGL ES 2.0 IOS. 2012. 78 p. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Fundação Universidade Regional de Blumenau, Blumenau, 2012. Disponível em: https://bu.furb.br//docs/MO/2012/350319_1_1.pdf. Acesso em: 13 fev. 2020.

UNITY. **Unity – Manual: Cameras**. [S.l], 2020a. Disponível em: https://docs.unity3d.com/Manual/CamerasOverview.html. Acesso em: 03 jun. 2020.

UNITY. **Unity – Manual: Canvas**. [S.l], 2020b. Disponível em: https://docs.unity3d.com/Manual/UICanvas.html. Acesso em: 16 jun. 2020.

UNITY. **Unity – Manual: Raw Image**. [S.l], 2020c. Disponível em: https://docs.unity3d.com/Manual/script-RawImage.html. Acesso em: 26 mar. 2020.

UNITY. **Unity – Manual: RenderTexture**. [S.l], 2020d. Disponível em: https://docs.unity3d.com/ScriptReference/RenderTexture.html. Acesso em: 03 jun. 2020.

UNITY. **Unity – Solutions: Game**. [S.l], 2020e. Disponível em: https://unity.com/pt/solutions/game. Acesso em: 03 jun. 2020.

UNITY. **Unity – Manual: Types of light**. [S.l], 2020f. Disponível em: https://docs.unity3d.com/Manual/Lighting.html. Acesso em: 03 jun. 2020.