VISEDU LIGHT: VISUALIZADOR DE RAY TRACING

Daniel Rossato Martini, 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

drmartini@furb.br, dalton@furb.br

Resumo: *Este artigo descreve o desenvolvimento de um programa para aprendizado de* ray tracing*, bem como os principais testes aplicados e os resultados obtidos. O programa foi desenvolvido em C# com Visual Studio 2019 utilizando a ferramenta Unity. O programa tem o objetivo de ensinar* ray tracing *de forma mais prática, permitindo o estudante fazer alterações em tempo real para ver como elas afetam a renderização da imagem, assim como oferecendo explicações das técnicas que estão sendo demonstradas. De forma a avaliar o desempenho do programa, foram feitos alguns cenários de testes separados pelas áreas do programa. Com os testes realizados, o programa mostrou que cumpriu o esperado, explicando para os usuários como o* ray tracing *é aplicado em uma renderização.*

Palavras-chave*: Aprendizado. Computação gráfica. Ray tracing. Unity.*

1 INTRODUÇÃO

O *ray tracing* não é apenas um algoritmo, mas sim a junção de vários algoritmos, os quais foram desenvolvidos a partir de um trabalho publicado por Appel em 1968. Em 1979 e 1980, Kay e Whitted expandiram a ideia, tendo adicionado cálculos mais corretos da iluminação especular e da refração de luz (LOPES, 2000, p. 1).

De acordo com Pacheco (2008, p. 3, tradução nossa), "*Ray tracing*, de forma geral, não é nada menos do que uma simulação perfeita da luz. [...] A única diferença é que algoritmos de *ray tracing* seguem o caminho da luz num caminho inverso, [...]". Assim, muitas vezes se torna difícil visualizar como o *ray tracing* está sendo utilizado dentro de uma cena, se tornando um algoritmo bem complexo. Muitas variáveis podem afetar renderizações com *ray tracing*, como a cor de um objeto refletir em outro objeto, ou então a textura de um objeto tornar a luz difusa.

Um dos primeiros exemplos de uso de *ray tracing* em animações e filmes foi em Carros, pela Pixar/Disney, aonde os carros eram bem curvos, com superfícies bem reflexivas (PACHECO, 2008, p. 5). Até há pouco tempo, apenas em cenas renderizadas *offline* era possível fazer o uso de *ray tracing*, pois, como nota Batali *et al*. (2006, p. 5), a renderização de algumas cenas pode levar muito tempo, chegando a mais de 100 minutos, o que impossibilita seu uso em qualquer aplicação em tempo real, como jogos.

Em 2018, a NVIDIA lançou sua nova geração de placas de vídeo, a série RTX, que contam com núcleos específicos para calcular *ray tracing* (NVIDIA, 2018). Ainda de acordo com a NVIDIA (2018), esses núcleos de *ray tracing* dão às placas de vídeo poder suficiente para fazer uso de *ray tracing* em tempo real em jogos. Com isso há um novo foco em *ray tracing*, porém por não ter sido muito explorado recentemente, não há muitas formas de se entender como o *ray tracing* funciona e nem como ele afeta as renderizações.

Com base nesses argumentos, se criou uma ferramenta para a visualização de como funciona o *ray tracing*, adicionando explicações para que estudantes da área de computação gráfica possam compreender essa tecnologia. O objetivo deste trabalho é disponibilizar um ambiente para visualização e aprendizado de iluminação sobre a tecnologia de *ray tracing*. Os objetivos específicos são: disponibilizar três cenas para simulação de *ray tracing*; apresentar uma explicação de como está ocorrendo o *ray tracing* na cena; permitir alterar textura dos objetos; permitir alterar cor dos objetos.

2 FUNDAMENTAÇÃO TEÓRICA

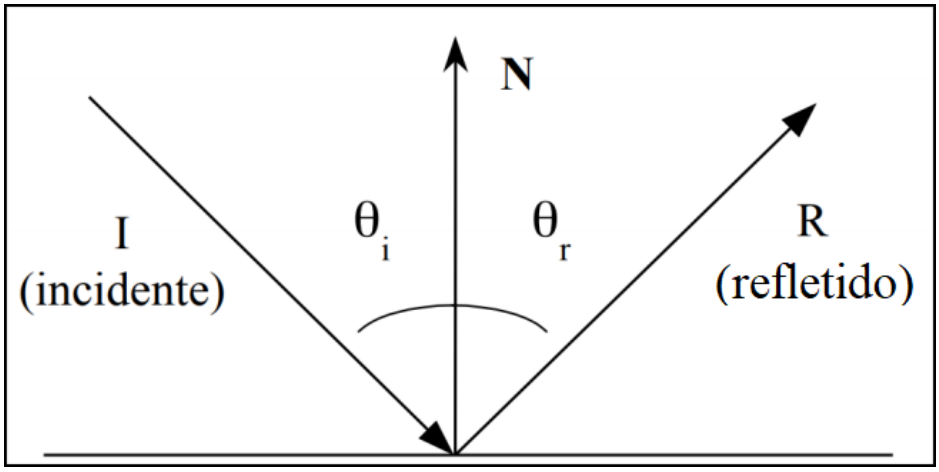
Nesta seção são apresentados os principais assuntos pertinentes à aplicação, sombras criadas com *ray tracing*, reflexos e refrações, *path tracing* e *denoiser*. Também são apresentados os trabalhos correlatos ao programa.

2.1 RAY TRACING

De acordo com Pacheco (2008, p. 1, tradução nossa), "O *ray tracing* pode ser visto como um algoritmo recursivo para calcular a cor de um pixel". Ainda de acordo com Lopes (2000), existem três tipos de raios com funções distintas, sendo eles os raios refletivos, refratados e de iluminação direta ou de sombra. Dentro dos raios de reflexão, existem dois tipos, a reflexão especular perfeita e a reflexão difusa perfeita.

Reflexão especular perfeita ocorre em superfícies polidas e brilhantes, geralmente metálicas, de vidro ou espelhadas. Na realidade não existem superfícies refletivas perfeitas, porém o modelo de reflexão especular perfeita providencia uma aproximação suficientemente parecida com as reflexões que ocorrem em um ambiente real. A Figura 1 demonstra como ocorre a reflexão especular, aonde um raio é refletido no mesmo ângulo de entrada, na imagem indicado como Θ (LOPES, 2000).

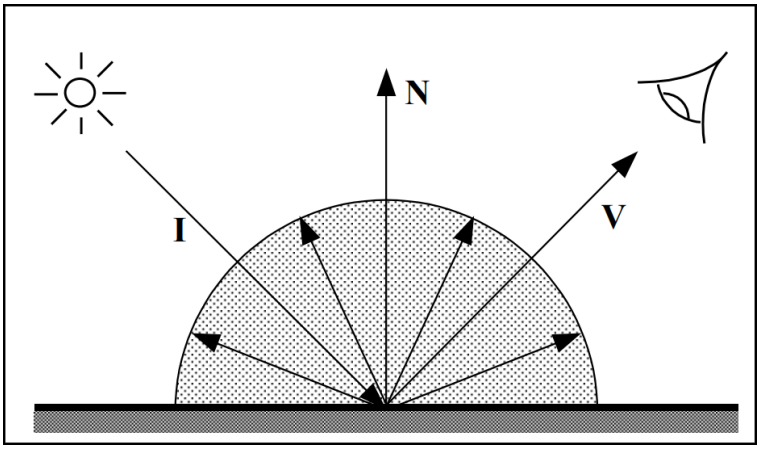
Figura 1 - Reflexão especular perfeita



Fonte: Lopes (2000).

Reflexão difusa perfeita ocorre em superfícies irregulares, com esse tipo de reflexão não existe uma única direção que o raio de luz pode seguir, na verdade a luz se dispersa em igual densidade para todas as direções. A Figura 2 exemplifica como ocorre reflexão difusa perfeita (LOPES, 2000).

Figura 2 - Reflexão difusa perfeita

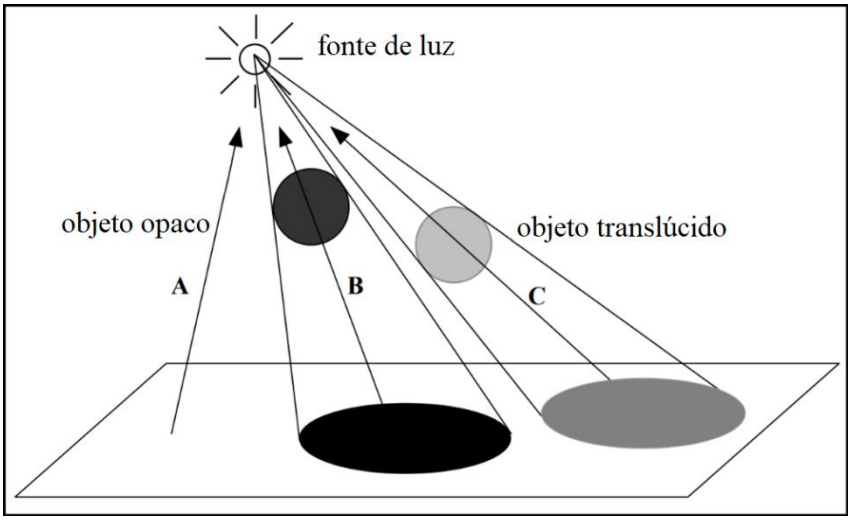


Fonte: Lopes (2000).

A refração é uma outra propriedade que deve ser levada em consideração no *ray tracing.* Segundo Lopes (2000, p. 19), a luz se propaga por objetos transparentes ou translúcidos, por esse motivo é possível ver através de vidros, mas isso também faz com que os objetos aparentam estar mais próximos do que realmente estão caso esteja dentro da água. Isso é causado pela diferença da velocidade de propagação da luz em diferentes meios, ao se propagar de um meio para outro, os raios têm a sua direção alterada. O grau de alteração da direção desse raio depende do índice de refração do meio que o raio acabou de entrar.

A definição de sombras é feita utilizando-se de raios secundários, criando-se um raio do ponto de intersecção do raio principal com um objeto, como o chão, até cada uma das fontes de luz (LOPES, 2000). Na Figura 3, existe o raio A, que por não ter nenhuma intersecção, sabe-se que é um local iluminado. No raio B, há uma intersecção com com um objeto, então não há luz no local abaixo desse objeto, já o raio C intersecciona com um objeto translúcido, então há luz no local, porém reduzida.

Figura 3 - Iluminação direta e sombras



Fonte: Lopes (2000).

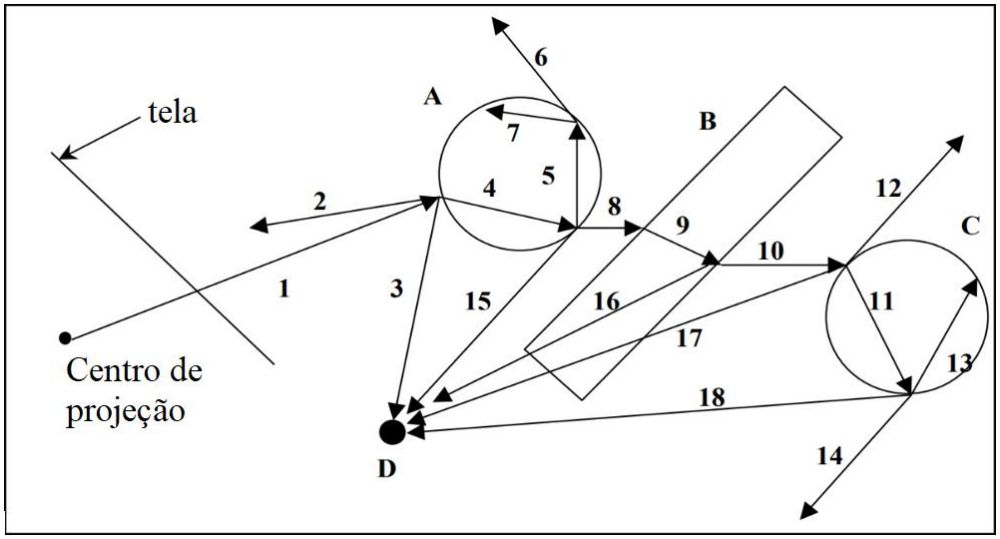
2.2 PATH TRACING

Path tracing foi definido como sendo um único algoritmo para resolver uma grande variedade de algoritmos de renderização. Ele providencia um contexto unificado de aproximações para visualizar a solução de uma única equação (KAJIYA, 1986). Ainda de acordo com Kajiya (1986), isso não é uma surpresa, pois todos os métodos de renderização tentam modelar o fenômeno físico da luz refletindo sobre diversos tipos de superfícies.

Kajiya (1986) utilizou como base a equação de radiosidade, balanceando o fluxo de energia de um ponto da superfície para outro. Kajiya (1986, p. 1, tradução nossa) define, "[…] o transporte da intensidade da luz de um ponto das superfícies para outro é simplesmente a soma da luz emitida e a intensidade total da luz que foi espalhada em direção ao ponto de vista vindo de todos os outros pontos de superfície".

De forma simples, o algoritmo de *path tracing* funciona lançando um ou mais raios de cada pixel da tela, então para cada raio verifica-se as intersecções com os objetos. Quando o raio se intersecciona com algum objeto, verifica-se se esse objeto gera reflexão difusa ou especular, gerando o raio secundário apropriado. De forma recursiva então são gerados novos raios, até que eles atinjam uma fonte de luz ou a quantidade máxima de novos saltos for atingida. A Figura 4 demonstra o algoritmo em um ambiente simples, contendo alguns objetos translúcidos. O número em cada um dos raios é a sequência em que eles foram criados e o item D é a fonte de luz.

Figura 4 - Algoritmo de *path tracing*



Fonte: Lopes (2000).

2.3 DENOISER

O *denoiser* é parte integral do processo de renderização do *ray tracing* (MUNKBERG, 2019, p. 287). Munkberg (2019) ainda diz que em uma aplicação em tempo real o *ray tracing* apenas consegue realizar a renderização com poucos raios, dessa forma gerando ruído, porém combinando esses poucos raios com filtros e um *denoiser* pode gerar um resultado muito melhor. Munkberg (2019) define o *denoiser* de maneira ampla como sendo um algoritmo que limpa uma imagem ou parte dela, removendo seus ruídos.

Chaitanya (2017) descreve que existem diversos tipos de *denoisers* que podem ser utilizados, um dos mais simples apenas borrando a imagem. Ainda de acordo com Chaitanya (2017), é possível aplicar em renderizações *offline* alguns *denoisers*, como filtros de imagem não lineares e filtro cross-bilateral baseado em informação mútua, ou até métodos que utilizam características do processo de renderização. Há também os métodos que são aplicados à renderizações interativas, como jogos, que são geralmente mais difíceis de implementar, pois costumam precisar partir de imagens com muito ruído. Alguns exemplos de *denoisers* para renderizações interativas de acordo com Chaitanya (2017) são: *manifolds* adaptivas, filtros de imagens guiadas e onduleta *À-trous.*

2.4 TRABALHOS CORRELATOS

São apresentados três trabalhos semelhantes ao trabalho proposto. O primeiro é apresentado no Quadro 1, se trata do trabalho de conclusão de curso de Koehler (2015), que desenvolveu um visualizador de material educacional voltado à renderização. O segundo é o POV-Ray (PERSISTENCE OF VISION RAYTRACER PTY, 2013), uma ferramenta para criar imagens tridimensionais com *ray tracing*. Por último, é apresentado o trabalho feito por Peternier, Thalmann e Vexo (2006), o Mental Vision, um programa para ensino de computação gráfica.

Quadro 1 – VisEdu-CG 4.0

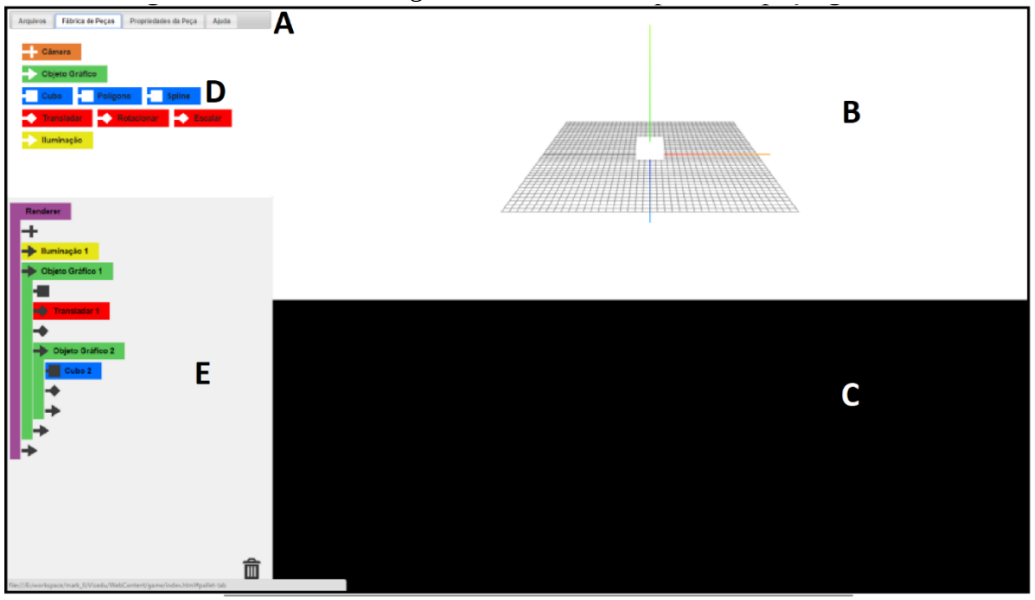
Referência	Koehler (2015)
Objetivos	É uma aplicação *web* voltada para facilitar a educação da disciplina de computação gráfica, permitindo que o usuário interaja com conceitos básicos da disciplina.
Principais funcionalidades	Adicionar e remover objetos do espaço gráfico, rotacionar, deslocar elementos, aplicar textura e iluminação.
Ferramentas de desenvolvimento	A aplicação foi desenvolvida com a IDE Eclipse em conjunto com as linguagens HTML5, Javascript. O JQuery foi utilizado para realizar os comportamentos de arrastar e soltar pelas, bem como fazer o controle das abas e diálogos do visualizado. Para criar e manipular os elementos gráficos foi utilizada a biblioteca Three.js que serve para abstrair o WebGL, que utiliza a Graphics Processing Unit (GPU) para realizar a renderização (KHRONOS GROUP, 2019).
Resultados e conclusões	Usuários informaram que a aplicação é fácil de usar e que consegue cumprir seu papel de apresentar os conceitos aprendidos em sala. O autor conclui dizendo que a performance se manteve boa em alguns navegadores, dessa forma mostrando que há espaço para aplicações gráficas cada vez mais exigentes.

Fonte: elaborado pelo autor.

Koehler (2015) integrou dois outros trabalhos, sendo eles o VisEdu-CG 3.0 criado por Nunes (2014) e o projeto de Harbs (2013). A aplicação serve para facilitar o ensino de computação gráfica para alunos, permitindo criar cenas que servem para a visualização das diversas propriedades.

A Figura 5 apresenta a interface da aplicação que contém 4 abas (item A). A primeira é `Arquivos`, ela serve apenas para importar e exportar cenas criadas. A segunda aba, `Fábrica de Peças`, é dividida em 4 partes. Já a área B representa a cena que está sendo criada. A área C mostra o que a câmera está captando, que é o que está sendo renderizado. O item D são as peças, como objetos e propriedades que podem ser adicionados à cena. No item E é onde são adicionadas as peças à cena. Dentre os objetos que podem ser adicionados estão as fontes de luz, que funcionam de forma bem simples, apenas iluminando aquilo que atingem, sem reflexões. A terceira aba é a `Propriedades da Peça`, nela são definidas as propriedades, como posição e textura da peça que está selecionada. Por último há a aba `Ajuda`, nessa aba há apenas algumas informações sobre como o programa funciona.

Figura 5 - ViseduCG 4.0



Fonte: Koehler (2015).

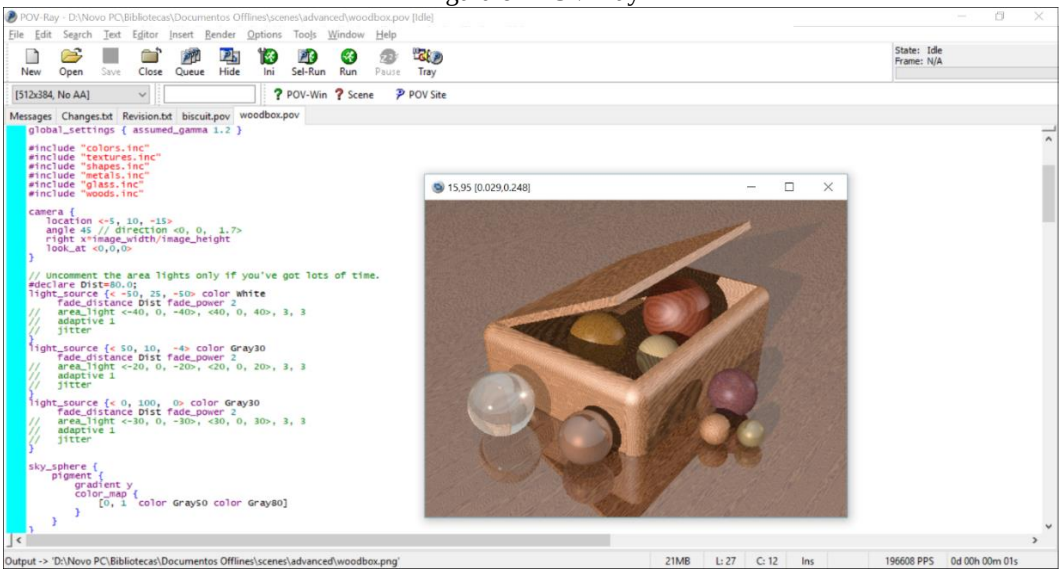
Quadro 2 – POV-Ray

Referência	PERSISTENCE OF VISION RAYTRACER PTY (2013)
Objetivos	É um editor livre que busca permitir criar e renderizar cenas utilizando *ray tracing*.
Principais funcionalidades	Disponibiliza uma tela para criar cenas (Figura 6). Para criar as cenas ele suporta alguns objetos, como esferas e retângulos, bem como diversos tipos de fontes de luz. As cenas são definidas com uma linguagem própria. Permite inserir objetos de forma fácil a partir de uma extensa lista de opções.
Ferramentas de desenvolvimento	Não encontrado.
Resultados e conclusões	É um programa de alta complexidade e difícil de aprender, porém gera renderizações de alta qualidade e cumpre bem o que se dispõe a fazer.

Fonte: elaborado pelo autor.

A renderização é feita utilizando-se de *ray tracing* em toda a cena, utilizando as fontes de luz definidas no projeto e levando como base as propriedades definidas para os objetos, como reflexibilidade e cor. A renderização pode por vezes demorar bastante para terminar, pois não é utilizada aceleração por GPU, sendo apenas utilizada a Central Processing Unit (CPU). A alta complexidade do programa permite criar imagens bem realistas com o uso do *ray tracing*. O programa tem uma área de ajuda, porém não faz nenhuma explicação do que se está passando na hora da renderização.

Figura 6 - POV-Ray



Fonte: Persistence of Vision Raytracer PTY (2013).

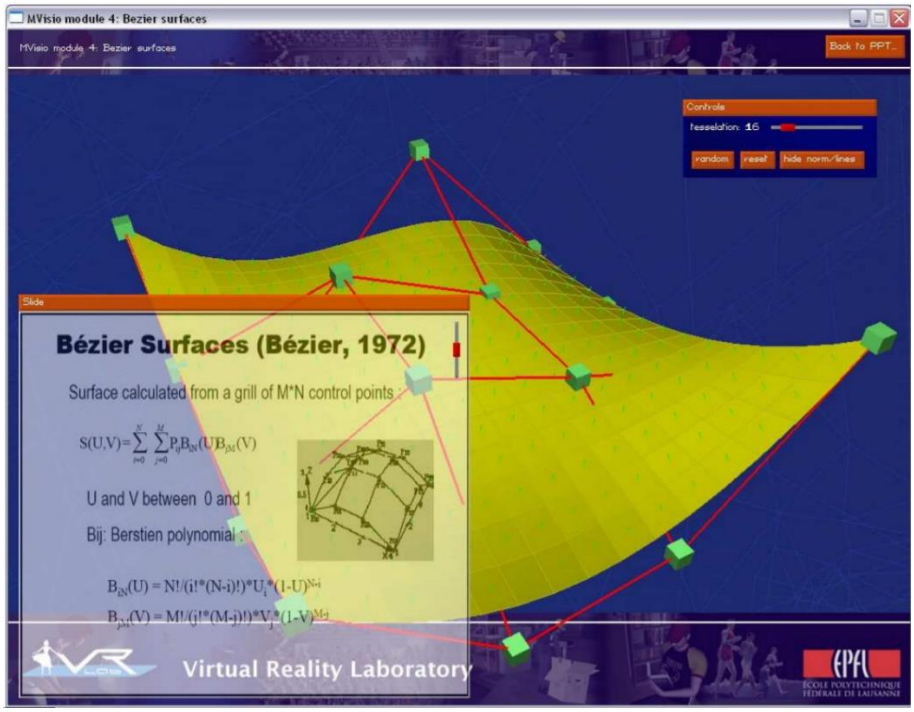
Quadro 3 – Mental Vision

Referência	Peternier, Thalmann e Vexo (2006)
Objetivos	É um programa para simplificar e melhorar o ensino e prática de computação gráfica. Ele proporciona explicações de técnicas e algoritmos de computação gráfica e disponibilizar um motor gráfico orientado à pedagogia para ser usado por alunos para projetos e trabalhos.
Principais funcionalidades	Disponibiliza duas partes. A primeira é são módulos pedagógicos que ilustram de maneira simples alguns temas complexos da computação gráfica, a Figura 7 demonstra um módulo sobre a superfície de Bézier. A segunda parte é um motor gráfico multiplataforma, que abstrai operações de baixo nível, permitindo que os usuários foquem em criar aplicações em alto nível.
Ferramentas de desenvolvimento	Foi construído em C++ e utiliza o OpenGL.
Resultados e conclusões	De acordo com os autores a utilização do programa se mostrou um sucesso, com os alunos que usaram o programa dando uma avaliação extremamente positiva. Os autores destacam ainda que o suporte à multiplataformas do programa estimulou os alunos a criarem aplicações que funcionam em várias plataformas ao mesmo tempo.

Fonte: elaborado pelo autor.

Peternier, Thalmann e Vexo (2006) perceberam que os seus alunos tinham dificuldades em utilizar motores gráficos de mercado, então decidiram criar o Mental Vision para auxiliar seus alunos. O foco foi criar um programa intuitivo e de fácil acesso à alunos que estão começando a aprender sobre computação gráfica. Os módulos pedagógicos são bem customizáveis, permitindo mudar diversas variáveis para ficar claro como elas funcionam e afetam a representação visual da cena. Para cada tópico há uma explicação detalhada sobre eles, mostrando fórmulas e desenhos sobre o tema. Diversas funções necessárias para o motor gráfico, como carregamento de imagens foram construídas diretamente dentro dele.

Figura 7 – MVisio, módulo de Bézier



Fonte: Peternier, Thalmann e Vexo (2006).

3 DESCRIÇÃO DO PROGRAMA

Este capítulo visa apresentar os detalhes de especificação e desenvolvimento do programa. O capítulo apresenta quatro seções, sendo a primeira responsável por exibir algumas características do desenvolvimento. Como o programa foi criado com três salas, as próximas seções detalham cada uma delas. A primeira é destinada à implementação do programa, a segunda é destinada à uma visão geral do menu inicial e da sala `Sombras`. A terceira seção explica o que foi feito na sala `Técnicas` e a quarta seção apresenta explicações sobre a sala `PathTracing`. Para realizar o desenvolvimento do programa foram utilizados os requisitos apresentados no Apêndice A.

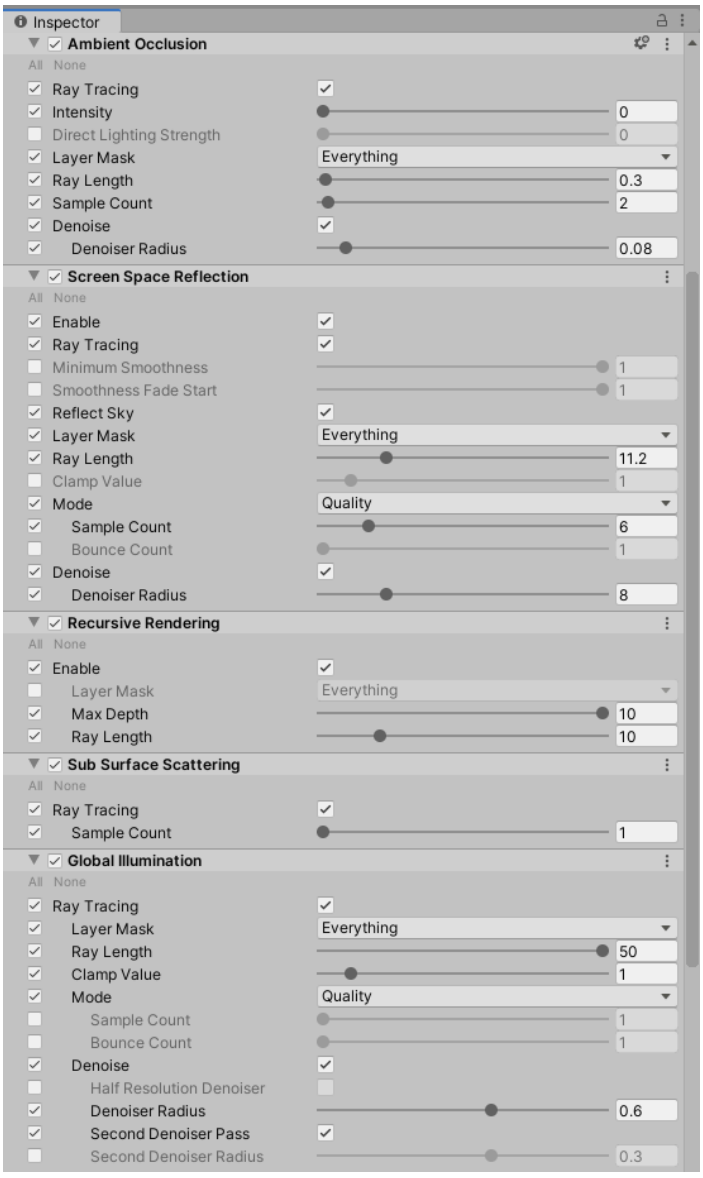
3.1 IMPLEMENTAÇÃO

Para realizar a implementação do programa se utilizou o Unity com scripts em C#. Para fazer uso do *ray tracing* se utilizou uma versão beta do Unity, a versão 2019.3b, que permite acesso à API de *ray tracing* do DirectX12. O próximo passo foi criar um projeto de High Definition Render Pipeline (HDRP), que é o tipo de projeto no Unity que permite fazer uso das APIs de *ray tracing* do DirectX12. Dentro do Unity foram utilizados os componentes `Volume`, no qual são disponibilizadas sobrecargas das opções padrões de renderização, e com essas opções é possível fazer o uso do *ray tracing*.

As sobrecargas utilizadas no programa foram de *Ambient Oclusion* (AO), *Screen Space Reflection* (SSR), *Recursive Rendering* (RR), *Global Illumination* e *Path Tracing*. O AO é responsável de verificar o quão escuro as pequenas áreas da cena devem ser renderizadas, com *ray tracing* isso pode ser feito de forma mais precisa. O SSR é responsável por calcular os reflexos da cena, definindo o que será refletido ou não, mas com o *ray tracing* ele se torna mais preciso. O RR é utilizado para fazer reflexos e refrações nas salas, e ele é utilizado apenas com *ray tracing*. O *Global Illumination* é utilizado para verificar o quão iluminada a cena como um todo deve ser. Dessa forma o *ray tracing* permite uma aproximação muito mais próxima à real. O *Path Tracing* sobrescreve a renderização de todos os outros componentes, e dessa forma sempre é utilizado sozinho. O *Path Tracing* é o mais custoso de processamento dos tipos que são utilizados nesse programa. O *Path Tracing* permite renderizar uma cena completa e de forma mais realista, apenas utilizando *ray tracing*.

Cada uma das salas utilizou uma combinação diferente dessas sobrecargas para poder mostrar da melhor forma como cada uma delas afetam uma cena. Na Figura 8 há um exemplo do uso dessas sobrecargas na sala `Técnicas`. Também se utilizou alguns `assets` disponíveis na Unity Store (Quadro 4).

Figura 8 - Configuração de `Volume` da Sala `Técnicas`



Fonte: elaborado pelo autor.

Quadro 4 - `Assets` utilizados

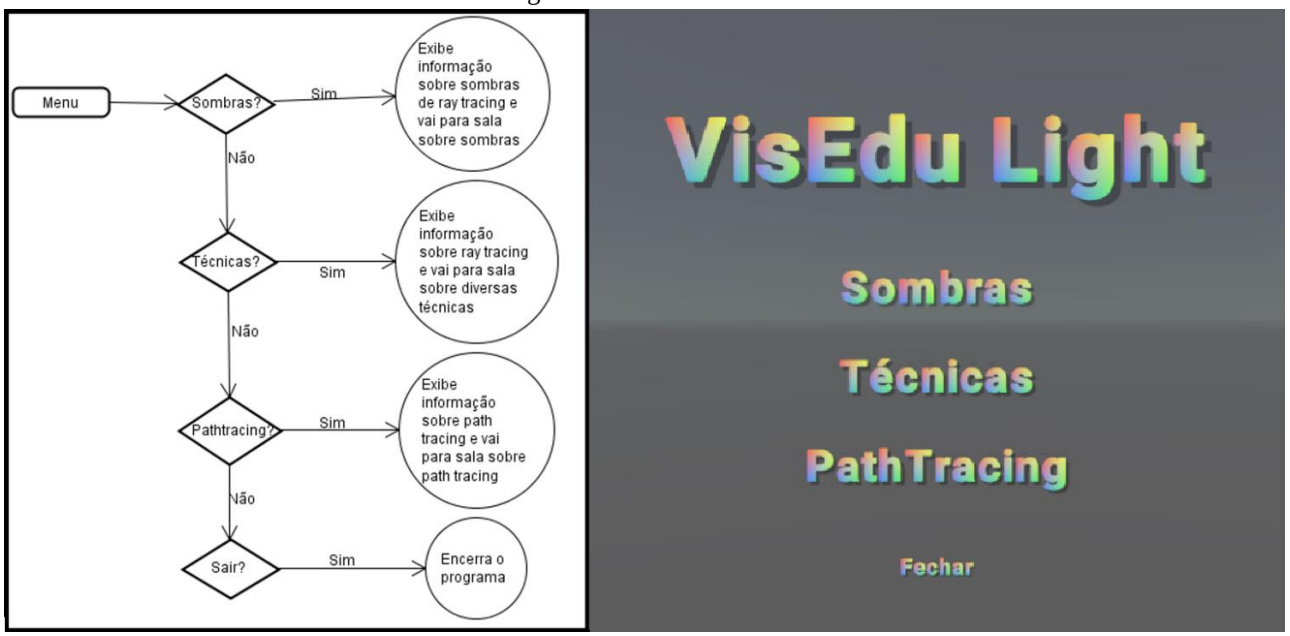
Asset	Uso
MS Sports Car	Foi utilizado o modelo gráfico de um carro em duas das salas.
Snaps Prototype | Asian Residencies	Utilizado para adicionar os modelos gráficos das casas visíveis na sala `PathTracing`.
Snaps Prototype | Construction Site	`Assets` leves utilizados para ver onde cada objeto deve ficar.
Snaps Art HD | Construction Site	`Assets` de alta qualidade utilizados na sala `PathTracing` para ter uma cena realista.
Asset Swap Tool	Ferramenta utilizada para fazer a troca entre os `assets` leves para os `assets` de alta qualidade

Fonte: elaborado pelo autor.

3.2 VISÃO GERAL DO MENU E SALA SOMBRAS

Este programa é composto pelo menu inicial e três salas, o menu tem três botões, cada qual quando clicados apresentam informações sobre a técnica escolhida e então leva o usuário à uma sala diferente. As salas apresentam explicações e exemplos sobre diferentes técnicas de *ray tracing* que podem ser aplicadas durante a renderização de uma imagem. Na Figura 9 há um fluxo do menu e uma imagem da tela do menu.

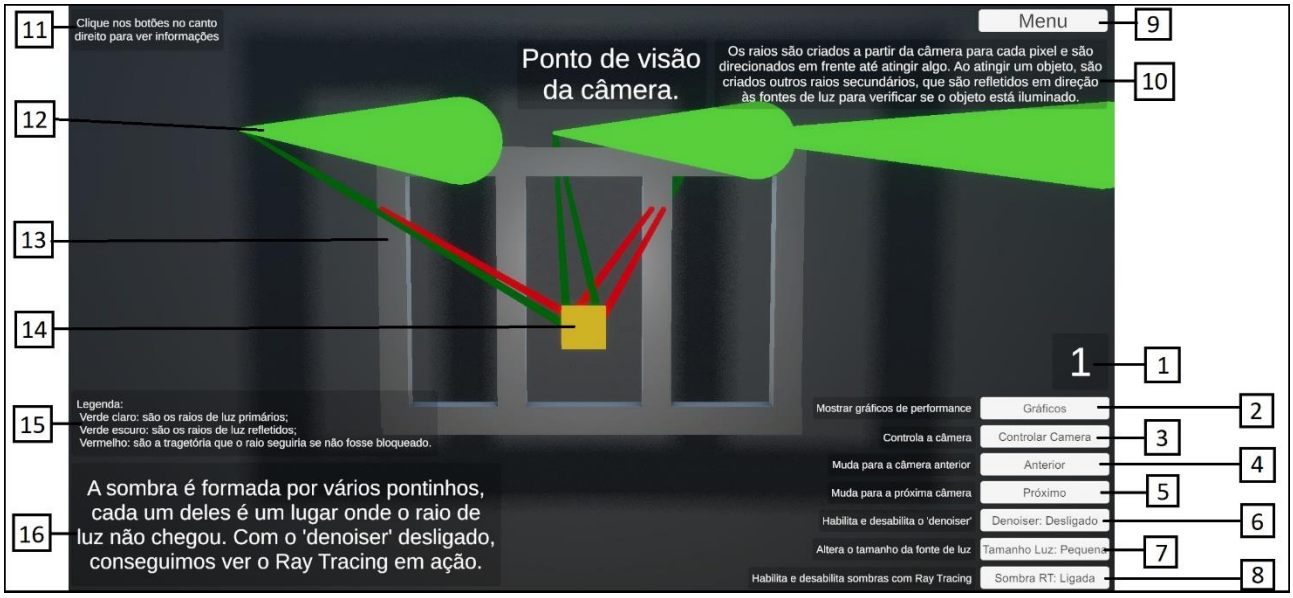
Figura 9 - Fluxo do menu



Fonte: elaborado pelo autor.

Ao clicar no primeiro botão, o usuário é levado à sala sobre sombras de *ray tracing.* Nesta sala o usuário é apresentado com uma cena exemplificando o uso de *ray tracing* para gerar sombras. O usuário pode fazer algumas alterações, como ligar e desligar as sombras de *ray tracing*, alterar o tamanho da fonte de luz, ligar e desligar o *denoiser*, alterar qual o ponto de visão e exibir gráficos de performance do sistema. A seguir a Figura 10 mostra essa sala com seus componentes identificados por números e o Quadro 5 apresenta a descrição de cada um dos componentes enumerados.

Figura 10 - Sala `Sombras`



Fonte: elaborado pelo autor.

Quadro 5 - Descrição da tela inicial da sala `Sombras`

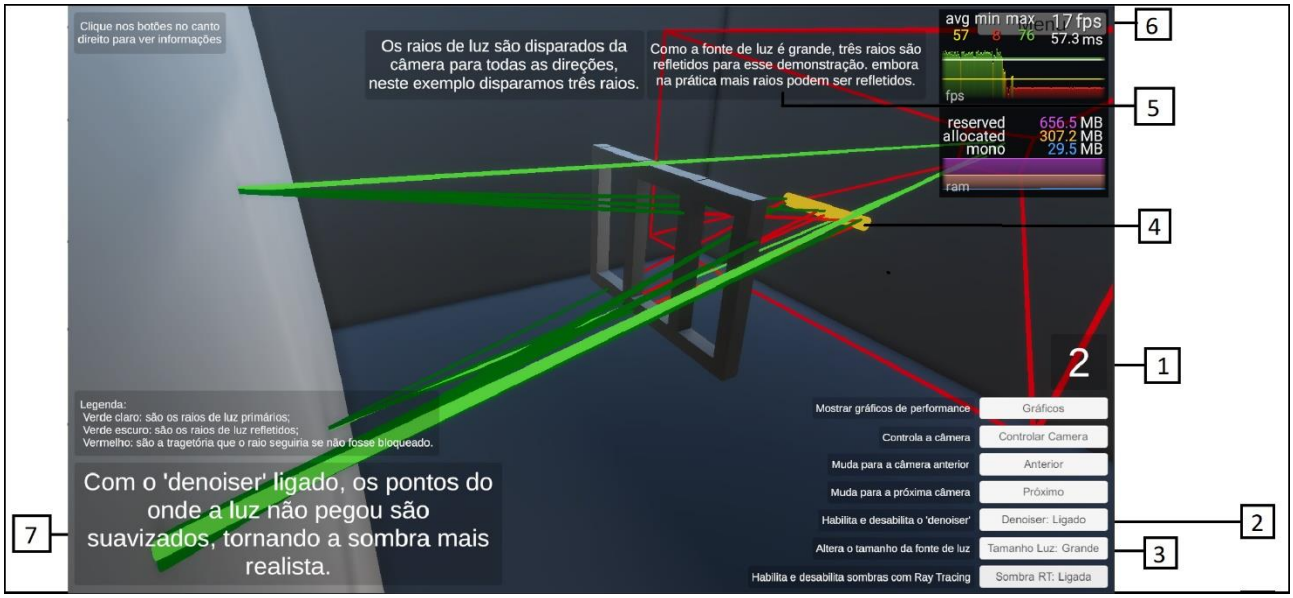
Identificador	Descrição
1	Indica qual o ponto de visão atual.
2	Botão para exibir os gráficos sobre performance.
3	Botão para controlar a câmera.
4	Botão para ir ao ponto de visão anterior.
5	Botão para ir ao próximo ponto de visão.
6	Botão para ligar e desligar o *denoiser*.
7	Botão para alterar o tamanho da fonte de luz.
8	Botão para ligar e desligar as sombras de *ray tracing*.
9	Botão para voltar ao menu.
10	Caixas de texto explicando o ponto de visão atual.
11	Caixa de texto dando instrução sobre os botões.
12	Guia para demonstrar um raio de luz.
13	Objeto que gera a sombra.
14	Objeto para representar a fonte de luz.
15	Caixa de texto com a legenda das cores dos raios de luz.
16	Caixa de texto com explicação do *ray tracing*.

Fonte: elaborado pelo autor.

Conforme o usuário passa de um ponto de visão para o próximo, as explicações vão mudando para refletir aquilo que está sendo dado ênfase no momento (Figura 11-5). Ao alterar o tamanho da fonte de luz, a cena muda para se adaptar ao novo tamanho da luz, criando novos raios para a demonstração e alterando a explicação exibida. Na Figura 11-4 se pode ver a nova fonte de luz maior, em amarelo, junto com novos raios. Já no item 3 a descrição do botão também é alterada. Caso o usuário clique no botão para exibir os gráficos (Figura 11-6) aparece na tela um gráfico dos Frames Por Segundo (FPS) junto com uma informação da memória que está sendo alocada. Com o FPS é possível analisar o impacto na performance com cada alteração feita na cena. Ao clicar no botão para ligar o *denoiser* a explicação no canto direito inferior é alterada (Figura 11-7) e o botão de *denoiser* também muda sua descrição (Figura 11-2).

O usuário também pode alterar o ponto de vista (na Figura 11 é possível ver outro ponto de visão), bem como no item 1 há um outro número indicando que é outro ponto de visão. No ponto de visão 6 o usuário pode controlar a câmera, dessa forma permitindo visualizar algo que o tenha interessado mais de perto.

Figura 11 - Alterações na sala de Sombras



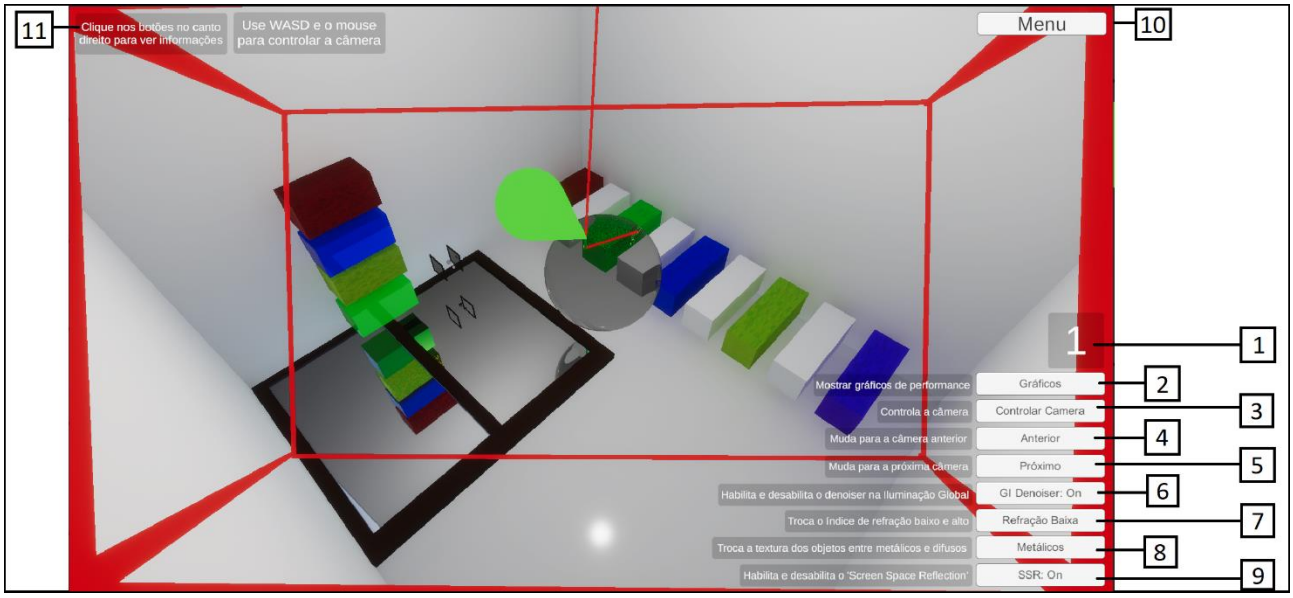
Fonte: elaborado pelo autor.

3.3 SALA DE TÉCNICAS

Quando o usuário clica no menu na sala `Técnicas` ele é direcionado a uma sala onde vão ser explicadas as técnicas de reflexos, refrações, iluminação global e dispersão de luz, (todas utilizando *ray tracing*). Esta é a sala com mais explicações e pontos de visão do programa, porém sua estrutura é parecida com a sala anterior, com os botões de ações no canto direito inferior e com as caixas de texto aparecendo na tela do usuário conforme ele vai passando ponto de visão em ponto de visão. As opções de `Gráficos`, `Controlar Câmera`, `Anterior` e `Próximo` funcionam da mesma forma que já explicado na sala `Sombras`. A Figura 12 enumera cada um dos pontos importantes da tela e o

Quadro 6 descreve esses pontos.

Figura 12 – Primeiro ponto de visão da Sala `Técnicas`



Fonte: elaborado pelo autor.

Quadro 6 - Descrição da tela inicial da sala `Técnicas`

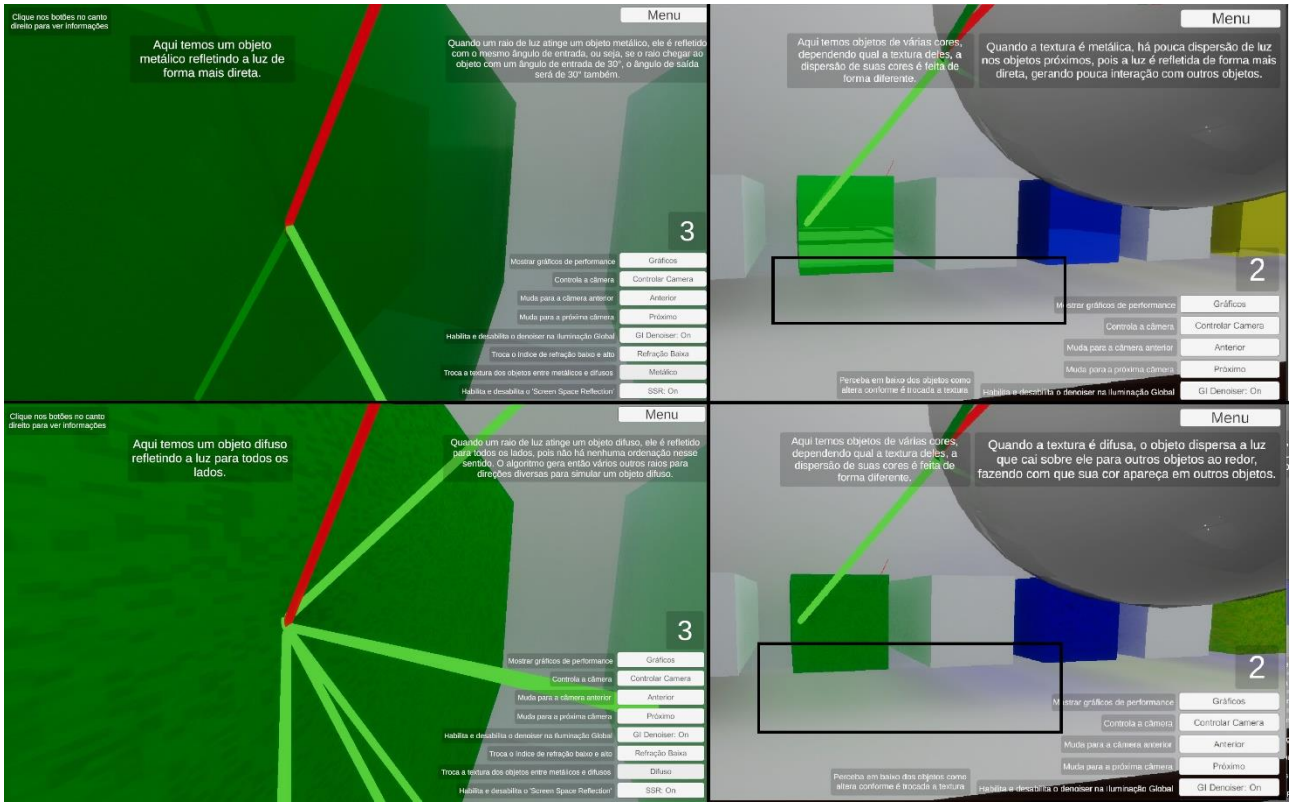
Identificador	Descrição
1	Indica qual o ponto de visão atual.
2	Botão para exibir os gráficos sobre performance.
3	Botão para controlar a câmera.
4	Botão para ir ao ponto de visão anterior.
5	Botão para ir ao próximo ponto de visão.
6	Botão para ligar e desligar o *denoiser* da Iluminação Global.
7	Botão para alterar o índice de refração dos vidros.
8	Botão para alterar o tipo dos objetos na cena entre metálicos e difusos.
9	Habilita e desabilita o *Screen Space Reflection.*
10	Botão para voltar ao menu.
11	Caixa de texto dando instrução sobre os botões.

Fonte: elaborado pelo autor.

As explicações da sala são feitas por etapa. Conforme o usuário vai avançando clicando no botão `Próximo` o ponto de vista vai mudando, e para cada novo ponto de vista há uma explicação nova. A primeira etapa é sobre a difusão da cor de um objeto em outros objetos próximos e como objetos metálicos e difusos afetam a renderização de uma cena de formas diferentes. Na Figura 13 há uma demonstração do que o usuário vê e como a cena é alterada quando o usuário muda os objetos de metálicos para difusos. Após essa explicação, são apresentadas ao usuário algumas explicações sobre os reflexos com *ray tracing*. Na tela são mostrados ao usuário dois espelhos, um usando o SSR e outro utilizando apenas *ray tracing*, junto com explicações sobre cada um deles e qual a diferença entre eles. O SSR pode também fazer o uso de *ray tracing*, e com o botão `SSR` o usuário pode habilitar e desabilitar para ver a diferença. Um exemplo dessa tela se encontra no Apêndice B - Figura 16. Se o usuário seguir para o próximo ponto de vista é explicado mais a fundo sobre os reflexos utilizando apenas o *RR*. Esta técnica faz uso apenas de *ray tracing*, utilizando dois espelhos um de frente para o

outro, o usuário pode ver como eles são refletidos um no outro diversas vezes, tela dessa parte se encontra no Apêndice B - Figura 17.

Figura 13 – Diferença entre metálico e difuso



Fonte: elaborado pelo autor.

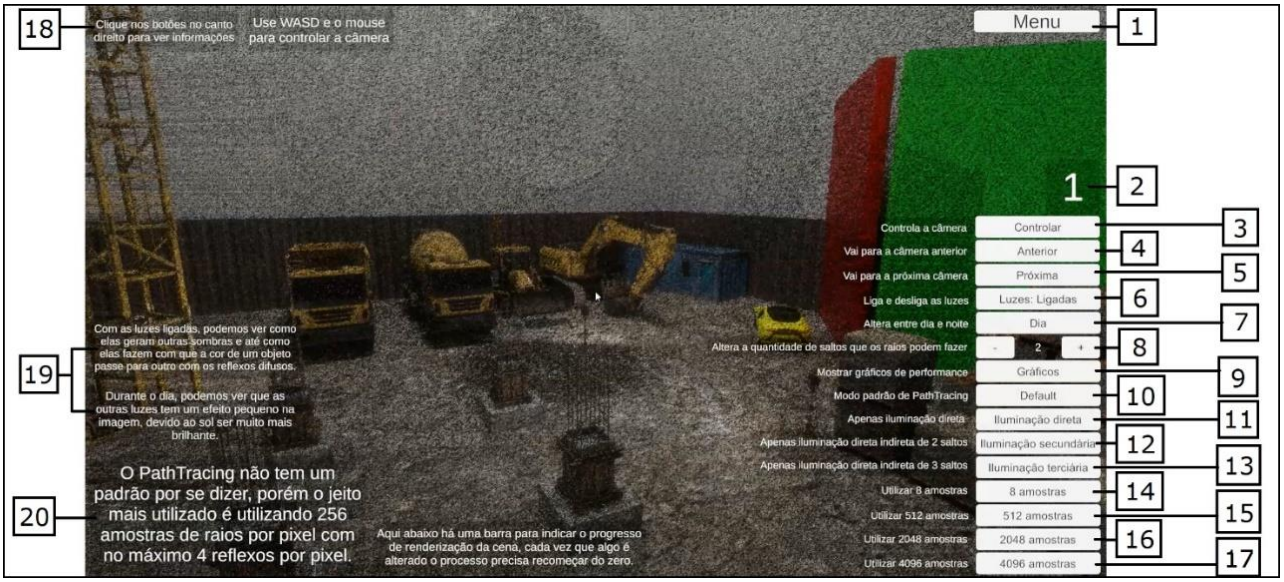
A terceira explicação é sobre a iluminação global com *ray tracing*, aonde é dada ao usuário uma explicação de como é aplicada a iluminação global na cena e como o *ray tracing* pode melhorar bastante a sua qualidade, mesmo utilizando poucas amostras por pixel. O usuário tem a opção de desligar o *denoiser* da iluminação global, podendo dessa forma ver como o *ray tracing* está sendo aplicado na cena inteira.

A última explicação da sala `Técnicas` é sobre a refração. Para essa explicação é mostrado ao usuário, utilizando-se de gizmos, como o raio é afetado ao passar por um objeto que refrata a luz. Junto com a explicação visual há mais alguns textos explicando o que está acontecendo. O usuário tem a opção de alterar o índice de refração do objeto e ao fazer isso os textos que são exibidos para ele mudam novamente, no Apêndice B - Figura 18 há um exemplo dessa tela. Durante todo o percurso o usuário tem a opção de controlar a câmera, permitindo então que ele veja algo que o tenha interessado mais de perto.

3.4 SALA DE PATHTRACING

Ao clicar no botão `PathTracing` no menu, o usuário é direcionado à sala que irá lhe mostrar diversas explicações sobre como *path tracing* é utilizado para realizar a renderização de uma imagem. Essa é a sala que contém a maior quantidade e qualidade de `assets`, bem como a técnica que mais demora para ser processada. Então ela requer um computador com uma performance maior para garantir uma taxa de renderização mínima. A estrutura dos botões na tela é parecida com as duas outras salas, com os botões `Gráficos`, `Controlar Câmera`, `Anterior` e `Próximo`. E o funcionamento destes botões também são iguais aos das outras salas. A Figura 14 enumera os principais pontos e o Quadro 7 explica cada um dos pontos enumerados.

Figura 14 – Sala `PathTracing`



Fonte: elaborado pelo autor.

Quadro 7 - Descrição da tela inicial da sala `PathTracing`

Identificador	Descrição
1	Botão para voltar ao menu.
2	Indica qual o ponto de visão atual.
3	Botão para controlar a câmera.
4	Botão para ir ao ponto de visão anterior.
5	Botão para ir ao próximo ponto de visão.
6	Botão para ligar e desligar as luzes espalhadas pela cena.
7	Botão para alterar entre dia e noite.
8	Botões para alterar a quantidade de saltos que os raios de luz podem fazer.
9	Botão para exibir os gráficos sobre performance.
10	Botão alterar a configuração de *ray tracing* para a padrão.
11	Botão para que apenas iluminação direta seja mostrada.
12	Botão para que apenas iluminação secundário seja mostrada.
13	Botão para que apenas iluminação terciária seja mostrada.
14	Botão para que sejam utilizadas 8 amostras por pixel.
15	Botão para que sejam utilizadas 512 amostras por pixel.
16	Botão para que sejam utilizadas 2048 amostras por pixel.
17	Botão para que sejam utilizadas 4096 amostras por pixel.
18	Caixa de texto dando instrução sobre os botões.
19	Caixas de texto explicando sobre as luzes na cena.
20	Caixa de texto que altera sempre que o usuário muda qual a configuração ele está usando.

Fonte: elaborado pelo autor.

Nessa sala há dois tipos de luzes, o Sol e algumas outras luzes pequenas espalhadas pela cena. O usuário tem a opção de deixar a cena de noite ou apagar as luzes pequenas, podendo assim ver como as luzes e o Sol afetam a cena. Conforme o usuário apaga ou acende as luzes a descrição que aparece nas caixas de texto muda para refletir estado atual das luzes. O Apêndice B - Figura 19 mostra como essa sala fica de noite, porém com as luzes pequenas acesas.

O usuário também pode alterar várias propriedades do *path tracing*, entre elas a quantidade de amostras e a quantidade máximas e mínimas de saltos que os raios podem realizar. Conforme cada botão que o usuário clica a forma que a cena é renderizada é alterada, permitindo assim o usuário ver as diversas formas que o *path tracing* pode ser utilizado em uma cena.

Umas das configurações que o usuário pode realizar é mudar entre iluminação direta, secundária e terciária. Caso o usuário clique na opção de iluminação direta, os raios de luz serão limitados à no máximo 1 salto, dessa forma apenas ficarão iluminados os objetos que estão diretamente na linha de alguma fonte de luz. Outra opção que o usuário tem é ver a iluminação secundária, nesse modo os raios são obrigados a realizar apenas dois saltos, não podendo realizar menos ou mais. Esse tipo de iluminação permite que o usuário veja para onde os raios estão sendo refletidos, e o quanto esses raios refletidos interferem na iluminação de áreas que não estão diretamente na linha de alguma fonte de luz. Há também a

opção de iluminação terciária, que limita a três a quantidade máxima e mínima de saltos dos raios, com essa opção o usuário pode ver todos os locais que são iluminados por raios que fizeram três saltos, a partir desse ponto a iluminação fica bem mais fraca. Para cada uma das opções o texto de explicação muda, para permitir que o usuário entenda quais as diferenças. Essa quantidade de saltos está representada no fluxo do Apêndice C.

Outra configuração que o usuário pode alterar é a quantidade de amostras que são coletadas para cada pixel da tela. Cada amostra é um raio disparado, aonde por padrão essa sala utiliza 256 amostras por pixel, porém o usuário pode alterar para utilizar 8, 512, 2048 e 4096 amostras. Quando o usuário seleciona 8 amostras, são disparados apenas 8 raios por pixel, fazendo dessa forma que a cena seja renderizada muito mais rapidamente, porém com uma qualidade muito ruim. Para essa opção o usuário é informado sobre as suas características de ter uma melhor performance, porém com um resultado muito ruim e uma imagem difícil de distinguir. O usuário pode selecionar 512 amostras, o que resulta em uma imagem melhor, porém ao custo de um tempo maior para ser concluída. Há também as opções de 2048 amostras e 4096 amostras, aonde essas opções trazem a melhor qualidade no resultado final, porém podem demorar muitos minutos para serem concluídas, mesmo com hardware específico para isso. O usuário é informado nessas duas opções sobre suas características de serem melhores para renderização de animações do que para aplicações em tempo real. Para essa configuração da quantidade de amostras que o *path tracing* utiliza, há o Apêndice C que é possível ver aonde ela se encaixa.

4 RESULTADOS

Durante a fase de implementação foram sendo realizados diversos testes para garantir que o que estava sendo implementado estava correto. O computador utilizado para os testes foi um I7 4790k, 16GB de memória RAM e uma RTX 2070. Os testes foram divididos em três partes, uma para cada sala implementada, e foram aplicados separadamente conforme a implementação de cada sala foi terminada. O Quadro 8 mostra quais testes foram realizados para a sala `Sombras`, bem como o resultado esperado e se o teste foi bem-sucedido.

Quadro 8 - Testes realizados para validar Sala Sombras

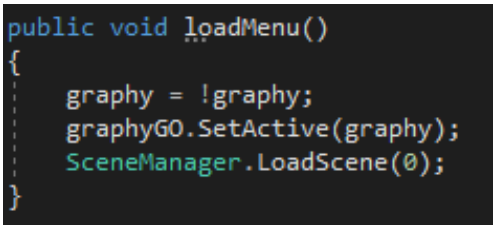
Teste	Esperado	Obtido
Clicar no botão `Menu`	Voltar ao menu principal.	OK
Clicar nos botões `Anterior` e `Próxima`	Alterar o ponto de vista que está sendo exibido na tela.	OK
Clicar no botão `Gráficos`	Caso os gráficos de performance não estejam ativos, então eles devem ser exibidos, caso contrário eles devem ser removidos.	NOK
Clicar no botão `Denoiser`	Caso *denoiser* esteja ativo, desativar ele, caso contrário ativar ele, alterar a descrição do botão para refletir o estado dele e alterar uma das caixas de texto para descrever o que está ocorrendo com ele.	OK
Clicar no botão `Tamanho Luz`	Caso a fonte de luz esteja pequena, alterar o tamanho para grande, caso contrário, alterar para pequeno. Mudar explicação nas caixas de texto e alterar os gizmos dos raios de luz para refletirem o novo tamanho da luz.	OK
Clicar no botão `Sombra RT`	Caso as sombras estejam sendo formadas com *ray tracing*, desativar elas, caso contrário, ativar elas. Alterar caixas de texto para explicar a diferença entra a sombra padrão do Unity com a sombra *ray tracing*.	OK
Controlar câmera	Permitir controlar a câmera com os botões WASD e com o mouse.	NOK

Fonte: elaborado pelo autor.

Utilizando o Quadro 8 é possível identificar dois pontos de falha, um em relação ao controle da câmera e outro em relação aos gráficos de performance. O controle da câmera estava sendo feito por um código criado pelo autor, porém isso não estava funcionando corretamente, pois ao tentar usar um dos botões de controle junto com o mouse ele acabava ignorando um dos *inputs*. Foi alterado para utilizar um código padrão do Unity, chamado `SimpleCameraControl.cs`. Com esse novo código o controle da câmera funcionou corretamente, mesmo se o usuário utilizar vários *inputs* ao mesmo tempo.

O outro problema encontrado foi com os gráficos de performance. Caso eles fossem ativados e depois o usuário voltasse ao menu principal, os gráficos continuavam presentes e não eram desabilitados conforme esperado. Para corrigir o erro, os gráficos foram adicionados dentro de um novo objeto e foi criado um novo trecho de código no momento de voltar ao menu. Nesse trecho o novo objeto está sendo desabilitado conforme pode ser observado na Figura 15. Após corrigidos os erros encontrados na sala `Sombras` foi realizado o desenvolvimento e subsequente teste da sala `Técnicas`. O Quadro 9 mostra os testes e resultados da sala `Técnicas`.

Figura 15 – Trecho de código para carregar Menu



Fonte: elaborado pelo autor.

Quadro 9 - Testes realizados para validar Sala `Técnicas`

Teste	Esperado	Obtido
Clicar no botão `Menu`	Voltar ao menu principal.	OK
Clicar nos botões `Anterior` e `Próxima`	Alterar o ponto de vista que está sendo exibido na tela.	OK
Clicar no botão `Gráficos`	Caso os gráficos de performance não estejam ativos, mostrar eles, caso contrário esconder ele.	OK
Clicar no botão `SSR`	Caso *ray tracing* no SSR esteja ativo, desativar ele, caso contrário ativar ele. Alterar a descrição do botão para refletir o estado dele e alterar uma das caixas de texto para descrever o que está ocorrendo com ele.	OK
Clicar no botão `Refração`	Caso a refração esteja baixa, alterar para ficar alta, caso contrário tornar a refração baixa. Alterar a descrição do botão e alterar a caixa de texto para explicar a diferença entre refração baixa e alta.	NOK
Clicar no botão `Metálico/Difuso`	Caso os objetos em tela sejam metálicos, alterar eles para ficarem difusos, caso contrário alterar para que fiquem metálicos. Alterar a descrição do botão, alterar as caixas de texto para que façam a explicação pertinente ao tipo de objeto que está sendo exibido e alterar os gizmos de raio de luz que demonstram a interação dos raios de luz com os objetos.	OK
Clicar no botão `GI Denoiser`	Caso o *denoiser* da iluminação global esteja ativo, ele deve ser desativado, caso contrário, deve ser ativado. Alterar a descrição do botão e alterar as caixas de texto que explicam sobre a Iluminação Global por *ray tracing.*	OK
Controlar câmera	Permitir controlar a câmera com os botões WASD e com o mouse.	OK

Fonte: elaborado pelo autor.

Com a execução dos testes foi possível perceber que apenas havia um problema com a refração de objetos. Como o índice de refração é definido por objeto e não algo global, não estava sendo possível alterar a refração do objeto em questão em tempo de execução. Neste caso não acontecia nada, possivelmente devido à algum problema com a versão do Unity. Então foi alterado para que existam objetos com índice de refração diferentes, e sempre apenas um deles está visível de cada vez, alterando qual está visível com base no que o usuário escolheu.

Não houve mais problemas com a sala pois algumas das correções que foram aplicadas na sala anterior já foram aplicadas a essa sala já durante o desenvolvimento, evitando assim a geração de outros erros. Após a correção do erro encontrado, foi realizado um novo teste para validar as correções e não foi encontrado mais nenhum erro. Foi realizado então o teste da terceira sala `PathTracing` (Quadro 10).

Quadro 10 - Testes realizados para validar Sala `PathTracing`

Teste	Esperado	Obtido
Clicar no botão `Menu`	Voltar ao menu principal.	OK
Clicar nos botões `Anterior` e `Próxima`	Alterar o ponto de vista que está sendo exibido na tela.	OK
Clicar no botão `Gráficos`	Caso os gráficos de performance não estejam ativos, mostrar eles, caso contrário esconder ele.	OK
Clicar no botão `Dia`	Trocar entre dia e noite na cena e alterar alguns textos de explicações.	OK
Clicar no botão `Luzes`	Caso as luzes espelhadas pela cena estejam ligadas, desligar elas, caso contrário ligar elas. Alterar a descrição de um dos textos em tela para refletir o estado atual das luzes.	OK

Clicar nos botões + e –	Alterar a quantidade de saltos que os raios podem realizar. Alterar a *label* entre os botões para mostrar quantos saltos os raios estão realizando atualmente.	OK
Clicar no botão `Default`	Alterar a configuração do *ray tracing* para ficar com os valores padrões.	OK
Clicar no botão `Iluminação Direta`	Alterar a configuração do *ray tracing* para utilizar apenas iluminação direta. Alterar as caixas de texto para explicar sobre iluminação direta.	OK
Clicar no botão `Iluminação Secundária`	Alterar a configuração do *ray tracing* para utilizar apenas iluminação secundária. Alterar as caixas de texto para explicar sobre iluminação secundária.	OK
Clicar no botão `Iluminação Terciária`	Alterar a configuração do *ray tracing* para utilizar apenas iluminação terciária. Alterar as caixas de texto para explicar sobre iluminação terciária.	OK
Clicar nos botões de `amostras`	Alterar a quantidade de amostras que o *ray tracing* está utilizando conforme o botão clicado. Alterar as caixas de texto para refletir qual opção foi utilizada.	OK
Controlar câmera	Permitir controlar a câmera com os botões WASD e com o mouse.	OK

Fonte: elaborado pelo autor.

Após a realização dos testes foi identificado que todas as ações estavam funcionando corretamente. Cada um dos botões estava realizando suas ações conforme esperado, porém foi percebido que caso fossem desligadas as luzes e trocado para ficar de noite, não era renderizado nada por falta de alguma fonte de luz. Então foi adicionado um texto quando está de noite e as luzes estão desligadas explicando que não é possível ver nada devido ao fato de que não há nenhuma fonte de luz na cena. A Quadro 11 demonstra o trecho de código que foi adicionado para mostrar esse texto.

Quadro 11 - Código para mostrar texto

```
public void toggleAllLights()
{
    bAllLights = !bAllLights;
    AllLights.SetActive(bAllLights);

    LightsInstructions.SetActive(bAllLights || bdayNight);
    SunInstructions.SetActive(bAllLights || bdayNight);

    if (!bAllLights && !bdayNight)
    {
        allLightsInstructions.text = "De noite e sem outras luzes, não é possível ver " +
            "nada na cena pela falta de alguma fonte de luz.";
        return;
    }

    if (bAllLights)
    {
        AllLightsTextText.text = "Luzes: Ligadas";
        allLightsInstructions.text = "Com as luzes ligadas, podemos ver como elas geram ou
    }
    else
    {
        AllLightsTextText.text = "Luzes: Desligadas";
        allLightsInstructions.text = "Com as luzes desligadas, apenas o Sol está iluminand
    }
}
```

Fonte: elaborado pelo autor.

5 CONCLUSÕES

O uso de *ray tracing* em aplicações de tempo real ainda está em seus estágios iniciais, com desempenho limitado mesmo com hardware feito para esse uso. O suporte de *ray tracing* no software também ainda está sendo melhorado. No início desse projeto o Unity para *ray tracing* ainda estava em versão alpha, e no momento de encerrar o projeto ainda se encontra na versão beta. Durante o desenvolvimento houve diversos problemas de instabilidade do software, com o Unity dando *crash* com frequência e até certo ponto parando de funcionar completamente, resultando em precisar de uma reinstalação do Windows para que ele voltasse a funcionar. Com o Unity saindo do alpha para o beta a estabilidade foi aumentando gradualmente até ela chegar no ponto atual que está aceitável. Acredita-se que com as próximas versões e com ele saindo do beta ficará muito mais fácil utilizar o *ray tracing* no Unity.

A maioria dos objetivos para a implementação foram cumpridos como especificados, porém as cenas descritas nos objetivos a, b e c foram modificadas de modo a ficarem com melhores exemplos de *ray tracing*. Sendo que a cena descrita no objetivo c foi completamente refeita utilizando-se de `assets` complexos e de alta qualidade para que fosse mostrado um exemplo mais realista. O programa se mostrou como uma boa forma de exemplificar o *ray tracing* para alunos que estão iniciando seu aprendizado, mas seria necessário fazer testes com alunas da disciplina de Computação Gráfica para analisar o seu ganho.

Embora o programa tenha cumprido os seus objetivos, é possível aprimorar sua eficiência no ensino do *ray tracing*. As possíveis extensões encontradas para o aprimoramento do programa são permitir ao usuário alterar e adicionar objetos nas cenas; permitir o uso do programa em computadores com menor capacidade de processamento; adicionar explicações mais complexas, com fórmulas; melhorar o controle da câmera; adicionar as explicações nos objetos ao invés de diretamente na tela, dessa forma apresentar as explicações conforme o usuário se aproxima dos objetos.

REFERÊNCIAS

BATALI, Dana et al. **Ray Tracing for the Movie 'Cars'.** 2006. Disponível em: <https://graphics.pixar.com/library/RayTracingCars/paper.pdf>. Acesso em 06 abr. 2019.

CHAITANYA, Chakravarty R. Interactive Reconstruction of Monte Carlo Image Sequences using a Recurrent Denoising Autoencoder**. ACM Transactions on Graphics**, v. 36, n. 4, p. 98:1-98:12, jul. 2017. Disponível em: <https://research.nvidia.com/sites/default/files/publications/dnn_denoise_author.pdf>. Acesso em: 14 jun. 2020;

HARBS, Marcos. **Motor para Jogos 2D Utilizando HTML5**. 2013. 78 f. Trabalho de Conclusão de curso (Bacharelado em Ciência da Computação) – Centro de Ciências e Exatas Naturas, Universidade Regional de Blumenau, Blumenau.

KAJIYA, James T. **The Rendering Equation.** In: ACM SIGGRAPH 86, 9., 1986, California. **Anais eletrônicos**… California: California Institute of Technology, 1986. Disponível em: < http://www.cse.chalmers.se/edu/year/2016/course/TDA361/rend_eq.pdf >. Acesso em: 20 jun. 2020.

KHRONOS. **WebGL 2.0 Specification.** 2019. Disponível em: <https://www.khronos.org/registry/webgl/specs/latest/2.0/>. Acesso em: 24 mar. 2019.

KOEHLER, William F. **VISEDU-CG 4.0:** visualizador de material educacional. 2015. 89 f. Trabalho de Conclusão de curso (Bacharelado em Ciência da Computação) – Centro de Ciências e Exatas Naturas, Universidade Regional de Blumenau, Blumenau.

LOPES, João M B. **Ray Tracing.** Lisboa, 2000. Apostila do Curso de Licenciatura em Engenharia Informática e de Computadores da Universidade Técnica de Lisboa. Disponível em: <http://disciplinas.ist.utl.pt/leic-cg/textos/livro/Ray%20Tracing.pdf>. Acesso em: 13 abr. 2019.

MUNKBERG, Jacob et al. **Ray Tracing Gems.** 2019. 607p. Disponível em: <http://raytracinggems.com>. Acesso em: 29 mar. 2020.

NUNES, S. A. **VisEdu-CG 3.0**: Aplicação didática para visualizar material educacional – Módulo de Computação Gráfica. 2014. 89 f. Trabalho de Conclusão de curso (Bacharelado em Ciência da Computação) – Centro de Ciências e Exatas Naturas, Universidade Regional de Blumenau, Blumenau.

NVIDIA. **NVIDIA RTX™ platform.** 2018. Disponível em: <https://developer.nvidia.com/rtx>. Acesso em: 13 abr. 2019.

PACHECO, Hugo. **Ray Tracing in Industry:** An up-to-date review of industrial ray tracing applications and academic contributions. 2008. Disponível em: < https://pdfs.semanticscholar.org/ded2/6e60f3a6c175604c303cd7dba0b4e621412f.pdf >. Acesso em: 10 abr. 2019.

PERSISTENCE OF VISION RAYTRACER PTY. **POV-Ray.** 2013. Disponível em: <http://www.povray.org/>. Acesso em: 24 mar. 2019.

PETERNIER, Achille; THALMANN, Daniel; VEXO, Frederic. **Mental Vision**: A Computer Graphics Teaching Platform. 2006. Disponível em: <https://www.researchgate.net/publication/227186762_Mental_Vision_A_Computer_Graphics_Teaching_Platform>. Acesso em: 23 mar. 2019.

APÊNDICE A – REQUISITOS DA APLICAÇÃO

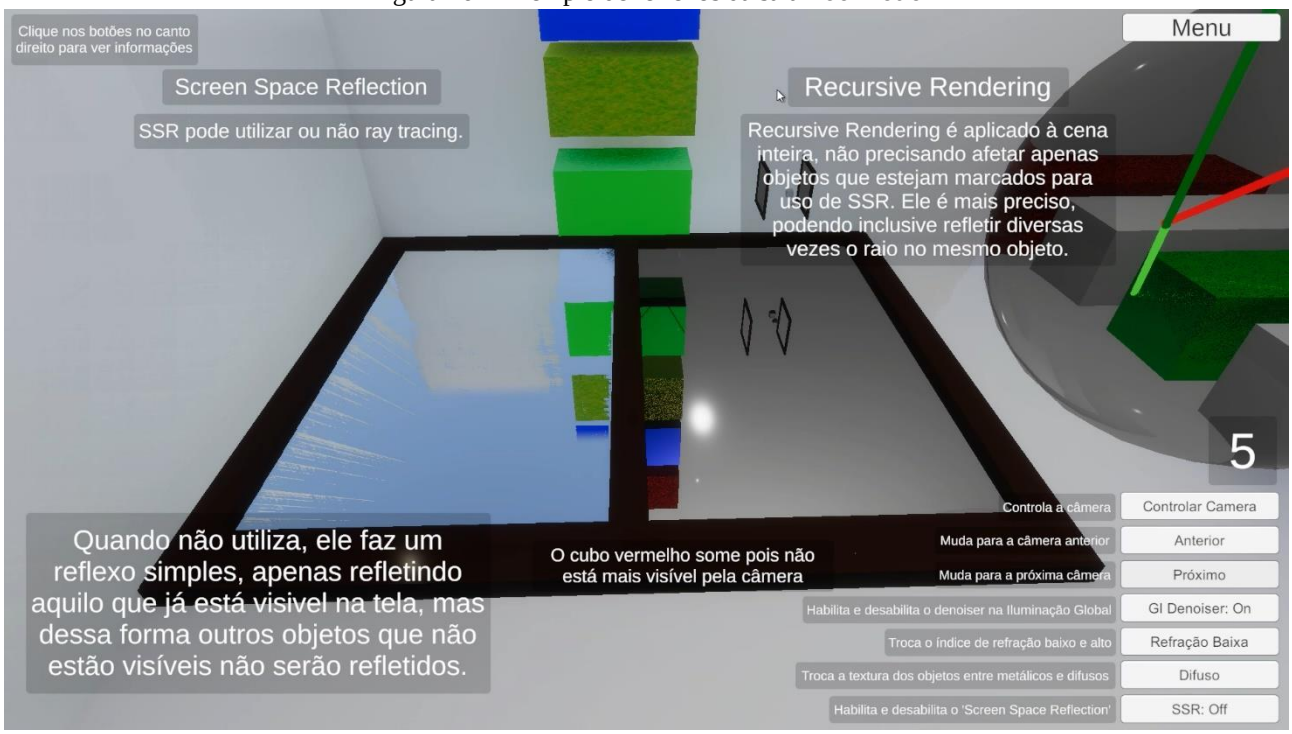
Neste Apêndice serão exibidos os principais requisitos da aplicação utilizados no desenvolvimento. Os requisitos estabelecidos foram:

a) possuir uma sala com um quarto branco, com dois cubos dentro e um holofote como fonte de luz apontando para um dos cubos (Requisito Funcional - RF);
b) possuir uma sala com um chão branco e uma parede branca ao fundo, um objeto cúbico e outro objeto esférico, um holofote apontando para a esfera e uma luz dispersa logo acima dos objetos (RF);
c) possuir uma sala com dois quartos ligados por uma porta, com uma fonte de luz dispersa num dos quartos, com dois objetos esféricos no quarto com a luz, e um objeto cúbico no quarto sem a luz (RF);
d) permitir o usuário alterar a textura e a cor dos objetos (RF);
e) permitir o usuário ligar e desligar o *ray tracing* (RF).
f) adicionar caixas de texto com explicações sobre como o *ray tracing* está sendo utilizado na cena (RF);
g) permitir escolher três cores, vermelho, verde e azul (RF);
h) possuir duas telas de visualização, uma com a visão da câmera e outra em terceira pessoa mostrando a cena como um todo (RF);
i) criar traços na visualização em terceira pessoa mostrando o caminho dos raios de luz (RF);
j) criar três tipos de texturas, reflexiva, opaca e transparente (Requisito Não Funcional - RNF);
k) menu que direcione para cada uma das salas (RNF);
l) utilizar aceleração de GPU quando disponível para ter um melhor desempenho (RNF).

APÊNDICE B – EXEMPLO DE TELAS DA SALA TÉCNICAS

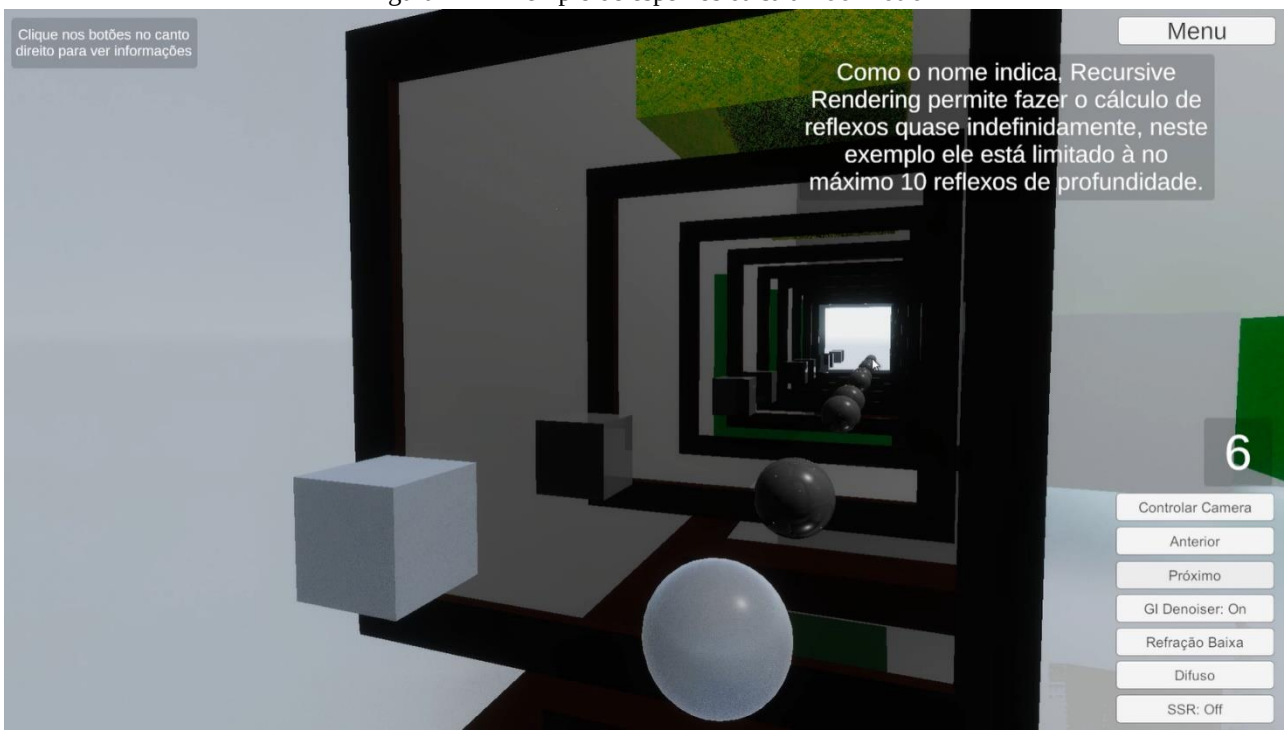
Neste Apêndice são exibidas algumas telas das salas `Técnica` e `PathTracing`. Da sala `Técnica`, há a Figura 16 que exibe um exemplo da parte de reflexos da sala `Técnica`, a Figura 17 que exibe um exemplo da etapa de espelhos e por último a Figura 18 que exibe a etapa de refração. A Figura 19 exibe um exemplo de noite da sala `PathTracing`.

Figura 16 – Exemplo de reflexos da sala `Técnicas`



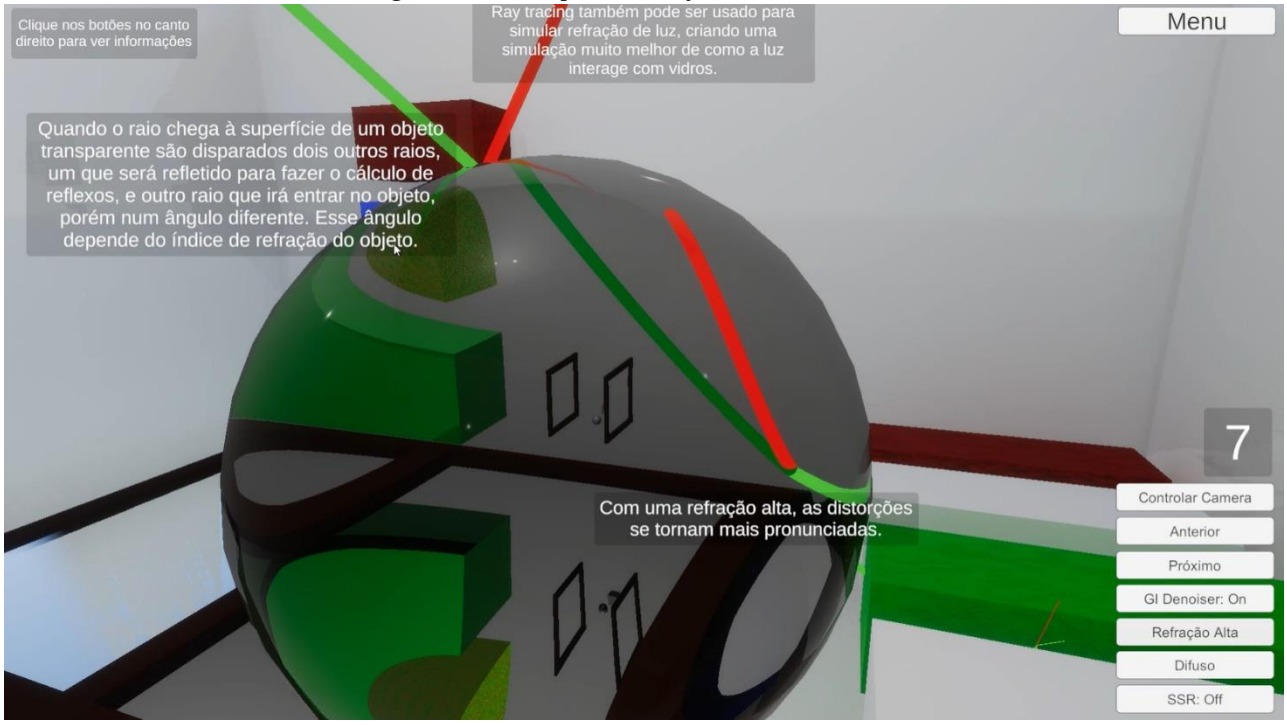
Fonte: elaborado pelo autor.

Figura 17 – Exemplo de espelhos da sala `Técnicas`



Fonte: elaborado pelo autor.

Figura 18 – Exemplo de refração da sala `Técnicas`



Fonte: elaborado pelo autor.

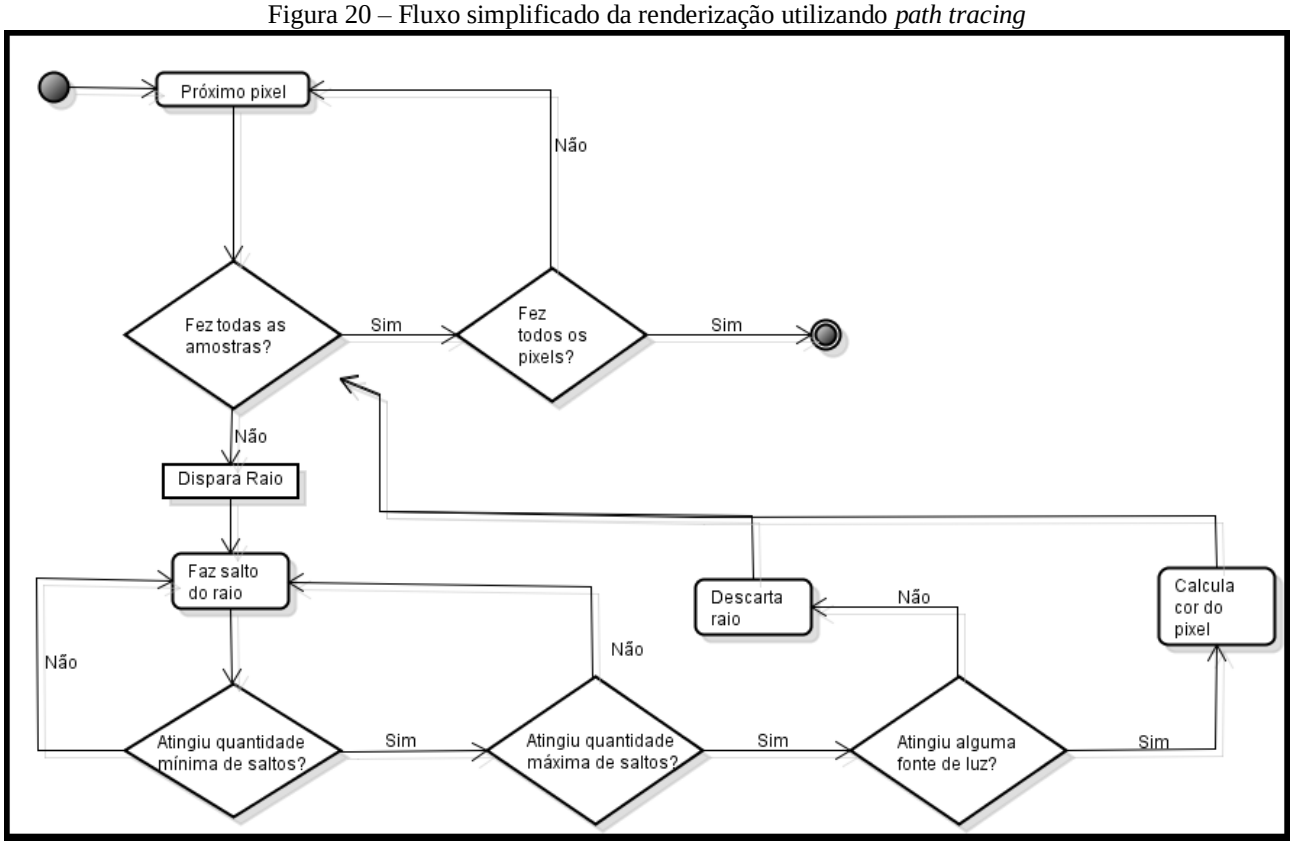
Figura 19 – Exemplo de noite da sala `PathTracing`

Fonte: elaborado pelo autor.

APÊNDICE F – FLUXOGRAMA DA RENDERIZAÇÃO COM *PATH TRACING*

Neste Apêndice é exibido um fluxo simplificado de como é feita a renderização de uma cena utilizando o *path tracing.* A Figura 20 exibe o fluxo.

Figura 20 – Fluxo simplificado da renderização utilizando *path tracing*



Fonte: elaborado pelo autor.