UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Arthur Carvalho Oliveira

Avaliação de diferentes metodologias para o ensino de computação gráfica em nível superior

Uberlândia, Brasil

2026

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Arthur Carvalho Oliveira

Avaliação de diferentes metodologias para o ensino de computação gráfica em nível superior

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Bruno Augusto Nassif Travençolo

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2026

Arthur Carvalho Oliveira

Avaliação de diferentes metodologias para o ensino de computação gráfica em nível superior

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 08 de maio de 2026:

Prof. Dr. Bruno Augusto Nassif Travençolo
Orientador

Humberto Luiz Razente

Daniel Duarte Abdala

Uberlândia, Brasil
2026

Resumo

A disciplina de Computação Gráfica passou, em 2023, a fazer parte da grade curricular obrigatória do curso de Ciência da Computação da Faculdade de Computação, da Universidade Federal de Uberlândia (UFU). Dessa forma, este estudo realiza uma avaliação comparativa das metodologias de ensino dessa disciplina em cursos superiores considerados referência no exterior: MIT, Utah, British Columbia, e Carnegie Mellon. A pesquisa adotou uma metodologia empírica baseada na implementação prática de 14 projetos propostos por essas instituições. O desenvolvimento abrangeu desde a rasterização via *software* e transformações geométricas até simulações físicas e *Ray Tracing*. Os resultados evidenciaram perfis pedagógicos distintos, contrastando a curva de aprendizado de abstrações *web* (WebGL/Three.js) com a exigência arquitetural de ferramentas de baixo nível (C++/OpenGL). Como desdobramento dessa análise empírica, este estudo delineia as diretrizes de um plano de ensino híbrido para a UFU. A proposta recomenda a adoção de tecnologias de mais baixo nível. A fim de mitigar a dificuldade inicial, sugere-se fornecer uma quantidade gradualmente decrescente de código-base no início de cada projeto, unindo assim a acessibilidade ao rigor técnico.

Palavras-chave: Computação Gráfica. Metodologias de Ensino. Ensino Superior.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CG	Computação Gráfica
PBL	Aprendizagem Baseada em Projetos
CAD	Desenho assistido por computador
CMU	Carnegie Mellon University
MIT	Massachusetts Institute of Technology
UBC	University of British Columbia
UTAH	University of Utah
UFU	Universidade Federal de Uberlândia
FACOM	Faculdade de Computação

Sumário

1 Introdução

O curso de Ciência da Computação da Faculdade de Computação (FACOM) da Universidade Federal de Uberlândia (UFU) foi criado em 1988. Nesses quase 30 anos já passou por várias reformulações, de forma a acompanhar as mudanças tecnológicas e científicas da área da computação. Em sua última reformulação, em 2023 (FACOM, 2023b), a disciplina de Computação Gráfica deixou de ser optativa e tornou-se obrigatória no curso de Ciência da Computação (FACOM, 2012; FACOM, 2023a). Alinhada às diretrizes da Sociedade Brasileira de Computação (SBC) (ZORZO et al., 2017), essa mudança evidencia a relevância cada vez maior da área, tanto no meio acadêmico quanto no mercado de trabalho. Como a disciplina de Computação Gráfica foi ministrada poucas vezes na FACOM, é necessário investigar seu desenvolvimento e as práticas pedagógicas adotadas em outros cursos de computação como guia inicial para a nova disciplina do curso de Computação e da FACOM.

O ensino de Computação Gráfica é fundamental para a formação de profissionais de computação, especialmente diante de sua ampla aplicação em filmes, séries, jogos e outras mídias. Além de aproximar conceitos abstratos de algoritmos e estruturas de dados de aplicações práticas, a disciplina integra álgebra linear com soluções computacionais. Esse contato estimula o aluno a explorar metodologias e algoritmos para representar um mundo virtual, elevando a qualidade do ensino na FACOM e gerando práticas pedagógicas que podem ser replicadas em outras instituições.

O Trabalho de Conclusão de Curso de William Johnson dos Santos Okano (2018) (OKANO, 2018) mostra como bibliotecas gráficas podem tornar o ensino de programação mais visual e facilitar a compreensão de conceitos abstratos. No entanto, ele não discute métodos específicos para ensinar Computação Gráfica, apenas usa recursos visuais para reforçar a aprendizagem de programação. Por isso, é importante analisar cursos renomados, como o **6.837 - Computer Graphics** do MIT (MIT, 2020), **CS4600** da University of Utah (UTAH, 2020), **CMU 15-462/662** da Carnegie Mellon University (CMU, 2020), e o **CPSC 314** da University of British Columbia (COLUMBIA, 2020), que são cursos de referência em Computação Gráfica e não têm apenas uma abordagem de ensino; enquanto (CMU, 2020) e (MIT, 2020) usam C++ e OpenGL com ferramentas auxiliares (como GLUT no MIT), a CMU possui uma biblioteca 3D própria chamada Scotty3D (SCOTTY3D, 2020), e (UTAH, 2020) e (COLUMBIA, 2020) usam WebGL junto com JavaScript. Essa comparação evidencia a necessidade de pesquisa sobre o método de ensino mais eficaz em Computação Gráfica.

O objetivo principal deste trabalho é comparar diferentes metodologias de ensino

de Computação Gráfica no ensino superior, avaliando sua eficácia no contexto da FACOM/UFU. Como objetivos secundários, pretende-se elaborar uma comparação com a ementa do curso de Ciência da Computação da UFU. Adicionalmente, busca-se aprofundar o conhecimento do pesquisador na área de Computação Gráfica.

De forma a orientar o desenvolvimento do trabalho, foi formulada a seguinte pergunta: Quais são os tópicos, ferramentas e abordagens pedagógicas predominantes em cursos de referência de Computação Gráfica? Essa metodologia tem como objetivo analisar cursos de computação gráfica relevantes, historicamente ou por contribuições recentes para a área (SUSELO; WüNSCHE; LUXTON-REILLY, 2017). Um dos critérios para a escolha foi a disponibilidade das aulas dos cursos em versões recentes ou anteriores de forma online. Cursos selecionados para análise: (MIT, 2020); (CMU, 2020); (COLUMBIA, 2020); (UTAH, 2020). Devido a não encontrar referência de calendário/trabalhos para o curso 6.837 (MIT, 2020), será usado o 6.837 (MIT, 2012) como referência para o mesmo, por se assemelharem na divisão de conteúdos e aulas.

2 Fundamentação Teórica

2.1 Computação Gráfica

Computação Gráfica é um ramo da Ciência da Computação que foca no desenvolvimento de técnicas para a representação de um mundo virtual, seja em 2 ou 3 dimensões. A área abrange diversas subáreas que são usadas em conjunto, sendo as três principais: modelagem, renderização e animação. Atualmente, é uma tecnologia central para diversas indústrias, como as de filmes, videogames, arte digital, entre outras (WOLFE; LOWTHER; SHENE, 2000).

Alguns eventos foram particularmente significativos para a evolução dessas tecnologias. Em 1959, o DAC-1 (*Design Augmented by Computers* – Design aprimorado por computadores), desenvolvido pela IBM e pela General Motors, forneceu o primeiro sistema industrial de CAD (*Computer-aided design* – Desenho assistido por computador), que permitia a visualização interativa de desenhos técnicos e, assim, lançou as bases para as ferramentas do gênero (KRULL, 1994). Quatro anos depois, em 1963, Ivan Sutherland apresentou o Sketchpad no MIT, por meio do qual os alunos puderam, pela primeira vez, desenhar diretamente na tela usando uma caneta de luz, consolidando os conceitos de interface gráfica e modelagem interativa (SUTHERLAND, 1963). Finalmente, em 1992, a Silicon Graphics lançou o OpenGL (SEGAL; AKELEY; FRAZER, 1994), uma API de renderização multiplataforma e internacionalizada, até hoje reconhecida como a mais usada para o ensino de computação gráfica (ANGEL; SHREINER, 2024).

O ramo da computação gráfica torna-se cada vez mais vasto com o passar dos anos, com novas APIs de renderização como o Vulkan, que dá mais acesso e controle da GPU e, com a possibilidade de uso em múltiplos sistemas operacionais. A Inteligência Artificial (IA) aplicada a gráficos, com tecnologias como o DLSS (*Deep Learning Super Sampling*) (NVIDIA, 2018), torna possível a renderização a uma fração do custo: renderiza-se apenas uma porção dos pixels para o alvo final e os pixels faltantes são previstos com IA. Além disso, a tecnologia de IA é usada para a geração de quadros inteiros (NVIDIA, 2022); nesse caso, a IA prevê um quadro intermediário a partir dos quadros anteriores. Essas tecnologias têm avançado o campo da computação gráfica, tornando a representação do mundo real virtual cada vez mais fidedigna.

2.2 Metodologias e Abordagens de Ensino em Computação Gráfica

As metodologias e abordagens de ensino em computação gráfica são os métodos usados para a condução da matéria.

As duas principais técnicas de ensino são a *bottom-up* (de baixo para cima) e a *top-down* (de cima para baixo). Na metodologia *bottom-up*, a matéria é conduzida como a construção de uma casa, peça a peça; é preciso construir a base para podermos avançar. Normalmente, inicia-se com algoritmos ou com a base matemática, como rasterização e transformações lineares, e o conhecimento é construído até que se tenha um pipeline gráfico completo. A vantagem dessa abordagem é a profundidade teórica proporcionada ao aluno, que passa a compreender cada etapa do processo (SUSELO; WüNSCHE; LUXTON-REILLY, 2017).

Na estrutura *top-down*, parte-se do pipeline gráfico completo para as partes mais fundamentais, como se, a partir da casa pronta, fôssemos analisar como foi construída. Inicialmente, parte-se de uma abstração generalizada e os tópicos são abordados de modo a explicar cada ponto, de forma que, ao final do curso, estejamos tratando de conceitos mais fundamentais, como curvas. A principal vantagem do modelo *top-down* é seu alto poder de motivação, pois os alunos veem os resultados de seu trabalho rapidamente, o que os mantém engajados (SUNG; SHIRLEY, 2003).

O ensino de computação gráfica vem evoluindo e, com o passar dos anos, técnicas como a aprendizagem baseada em projetos (*Project-Based Learning* – PBL) têm ganhado cada vez mais espaço. Essa abordagem permite ensinar de maneira híbrida. Por exemplo: a primeira semana de aula pode ser dedicada somente às transformações em 3D e, já na segunda semana, podemos propor um projeto prático sobre o mesmo tema. Com o fornecimento de material pré-pronto, é dados ao aluno a tarefa de completar um arquivo responsável pelas transformações. O aluno tem também a liberdade de personalizar a cena com funções de uma biblioteca previamente fornecida para engajá-lo. Assim, desde a segunda semana, o aluno já experiencia o feedback rápido, uma vantagem do *top-down*, enquanto mantém a profundidade teórica (RODRIGUES et al., 2021; SUSELO; WüNSCHE; LUXTON-REILLY, 2017).

3 Trabalhos Relacionados

Nesta seção, são apresentados alguns trabalhos relacionados. Cada trabalho contribui com aspectos específicos relevantes para a presente pesquisa.

O primeiro trabalho relacionado, (SUNG; SHIRLEY, 2003) focam em uma das primeiras implementações da metodologia *top-down* de ensino de Computação Gráfica em universidade. O trabalho detalha como o curso da universidade The University of Washington, Bothell, foi estruturado, focado em estudantes mais velhos, que já trabalham em tempo integral ou meio período, e cujas habilidades matemáticas não são das melhores. O trabalho estrutura o curso dividido em 3 tópicos: *Event and Simulator Driven Programming*, *Graphics API Abstraction* e *Transformation*. *Event and Simulator Driven Programming* foca em ensinar aos alunos como programar aplicações interativas. *Graphics API Abstraction* foca em introduzir o aluno ao uso de APIs gráficas populares, como OpenGL, DirectX-3D, etc., e também comparar as diferenças entre as APIs mais populares, com foco em preparar o estudante para desenvolvimento de software em larga escala. *Transformation* foca em ensinar o uso de transformações (translação, rotação, escala), basicamente apresentando como será feita a conversão e especificação de diversos espaços de coordenadas entre si e explica o pipeline de transformações e a modelagem hierárquica para compor objetos. O referido trabalho determina como seria a aplicação da metodologia *top-down* em universidades.

O trabalho (SUSELO; WüNSCHE; LUXTON-REILLY, 2017) focam em consolidar o conhecimento existente sobre o ensino de Computação Gráfica, identificando problemas comuns e soluções propostas pela comunidade acadêmica. Foram analisados dados de quatro repositórios científicos bem conhecidos – ACM Digital Library, IEEExplore, SpringerLink e ScienceDirect. Após buscas e filtragem, 45 artigos relevantes foram selecionados e, a partir desses, foram categorizados quatro principais problemas de ensino de : Conhecimento insuficiente de matemática (GÁLVEZ; IGLESIAS; CORCUERA, 2008; HUI et al., 2012; ZHOU; YE; HUANG, 2010) e programação básica (LOWTHER; SHENE, 2000; PAPAGIANNAKIS et al., 2014); dificuldades em compreender transformações, projeções e modelagem geométrica 3D (ELYAN, 2012); dificuldades em resolver problemas lógicos (HITCHNER; SOWIZRAL, 2000; TALTON; FITZPATRICK, 2007) e em estabelecer a conexão entre teoria, programação, aplicação e efeitos visuais (STEPHENSON; TAUBE-SCHOCK, 2009); e os estudantes tornaram-se aprendizes passivos, não interagindo muito com colegas nem com o corpo docente (PETERNIER; THALMANN; VEXO, 2006; MARTí; GIL; JULIà, 2006).

O trabalho de (BALREIRA; WALTER; FELLNER, 2017) focam em responder à

pergunta sobre o que está sendo ensinado nos cursos introdutórios de computação gráfica ao redor do mundo. Foram usados como base para pesquisa 20 cursos de graduação de nível superior com base na contribuição de artigos para a SIGGRAPH (*Special Interest Group on Computer Graphics and Interactive Techniques*), Siggraph Asia e Eurographics (*European Association for Computer Graphics*), considerando que o nível de pesquisa conduzido se transferiria para o ensinamento do tópico na universidade. A coleta foi realizada usando como fonte de informações dados públicos online.

A Tabela apresenta um resumo dos trabalhos relacionados.

Tabela 1 – Resumo dos trabalhos relacionados.

Trabalho	**Objetivo**	**Resultado**
(SUNG; SHIRLEY, 2003)	Detalhar a implementação da metodologia *top-down* de ensino de computação gráfica para estudantes maduros e com dificuldades em matemática.	Estruturação de um curso de nível superior dividido em 3 tópicos: Programação Orientada a Eventos e Simuladores, Abstração de API Gráfica e Transformações.
(SUSELO; WüNSCHE; LUXTON-REILLY, 2017)	Consolidar o conhecimento sobre o ensino de computação gráfica, identificando problemas comuns e soluções propostas na literatura.	Identificou, por meio de uma revisão sistemática, quatro problemas chave no ensino de CG e apontou as abordagens mais sugeridas na literatura para solução de cada problema.
(BALREIRA; WALTER; FELLNER, 2017)	Investigou o que está sendo ensinado nos cursos introdutórios de Computação Gráfica ao redor do mundo	Mapeou o conteúdo de 20 cursos de referência e concluiu que a área de Renderização compõe 75% do conteúdo, seguida por Modelagem (14%) e Animação (7%).

4 Desenvolvimento

Este capítulo apresenta a avaliação das metodologias adotadas pelas instituições de referência. A análise está estruturada de forma cronológica, avançando semanalmente pelos conteúdos ministrados em cada curso. Como todas as disciplinas avaliadas possuem um forte foco prático baseado na aprendizagem por projetos (PBL), optou-se por realizar a implementação de parte desses trabalhos. Essa resolução prática foi fundamental para propiciar uma análise mais aprofundada e realista acerca do nível de dificuldade exigido e dos tópicos técnicos priorizados por cada instituição. Todos os projetos podem ser acessados integralmente no repositório: (OLIVEIRA, 2026h).

4.1 Análise geral

Para iniciar a análise, é mostrado Tabela 2 a estrutura geral dos quatro cursos como distribuição de aulas do curso, tecnologias usadas para a realização dos trabalhos práticos e as bibliografias usadas.

Tabela 2 – Analise geral dos cursos analisados neste trabalho.

Curso	**Tecnologias**	**Bibliografias**	**Distribuição de Aulas**	**Trabalhos**
(MIT, 2020) (MIT, 2012)	C++, GLUT e OPENGL.	(BUSS, 2003), (WATT, 1993), (AKENINE-MöLLER; HAINES; HOFFMAN, 2008) e (SHIRLEY; MARSCHNER, 2009)	2 Aulas na semana por um período de 15 semanas.	0 Teóricos e 5 Práticos. Nota: Existe uma referência para pontos de quiz em (MIT, 2012), mas não foi encontrado qualquer quiz no site da disciplina.
(UTAH, 2020)	HTML, WebGL e JavaScript	(MARSCHNER; SHIRLEY, 2016)	2 Aulas na semana por um período de 15 semanas.	0 Teóricos e 7 Práticos
(CMU, 2020)	C++, (SCOTTY3D, 2020) e OPENGL	(SHIRLEY; MARSCHNER, 2009), (HUGHES et al., 2014) e (PHARR; HUMPHREYS, 2010)	2 Aulas na semana por um período de 15 semanas.	2 Teóricos e 4 Práticos
(COLUMBIA, 2020)	HTML, WebGL, JavaScript e Three.js	(MARSCHNER; SHIRLEY, 2016), (AKENINE-MöLLER; HAINES; HOFFMAN, 2018), (DUNN; PARBERRY, 2011), (GORTLER, 2012) e (MATSUDA; LEA, 2013)	3 Aulas na semana por um período de 14 semanas.	4 Teóricos e 4 Práticos

A tabela torna evidente alguns pontos principais de três aspectos: tecnologias, que varia entre duas combinações: C++, OpenGL e alguma biblioteca, ou JavaScript, HTML e WebGL; distribuição de aula, geralmente estrutura de 15 semanas com 2 aulas por semana; e o livro citado por todos como referência, *Fundamentals of Computer Graphics*, apenas a edição varia entre a terceira e a quarta (SHIRLEY; MARSCHNER, 2009; MARSCHNER; SHIRLEY, 2016)

Nas próximas seções serão apresentados os conteúdos abordados pelos cursos para cada semana.

4.2 Semana 1

O início dos cursos de introdução à Computação Gráfica apresenta um padrão muito semelhante e se divide entre a motivação para o estudo e uma revisão de matemática. Na introdução, (UTAH, 2020), (MIT, 2020; MIT, 2012) e (COLUMBIA, 2020) focam em definir a área e suas aplicações, sendo que (COLUMBIA, 2020) expande a discussão para abranger os tipos de primitivas geométricas. Em contrapartida, (CMU, 2020) adota uma abordagem mais prática logo de início, ilustrando conceitos básicos por meio do processo de modelagem, projeção em perspectiva e renderização de um cubo 3D.

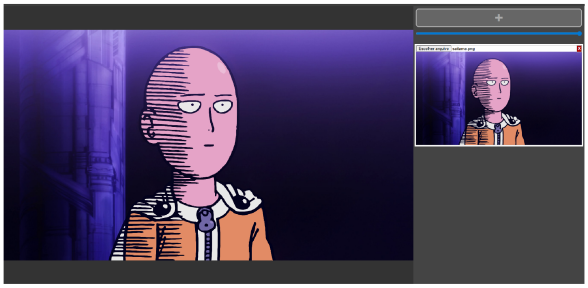
Na revisão de matemática, o foco em álgebra linear é um padrão comum para preparar os alunos, mas a densidade do conteúdo varia bastante entre os cursos. A abordagem de (UTAH, 2020) é a mais direta e concisa, revisando operações vetoriais fundamentais e produto escalar de forma bem objetiva. A referência (COLUMBIA, 2020) explora o tema pela ótica da álgebra geométrica, dando ênfase a conceitos como o *wedge product* e introduzindo a construção de curvas de Bézier. Com maior abrangência, (CMU, 2020) oferece a revisão mais robusta e estruturada em duas partes. A primeira é dedicada a álgebra linear avançada, incluindo ortonormalização, espaços vetoriais e transformações. A segunda foca integralmente em cálculo vetorial, abordando gradientes, campos vetoriais e operadores diferenciais para consolidar a base exigida na Computação Gráfica moderna.

4.3 Semana 2

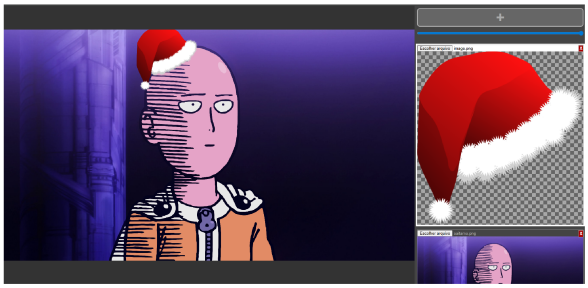
Na segunda semana, se observa uma tendência de introduzir os alunos aos conceitos fundamentais de renderização. A abordagem de (UTAH, 2020) é mais básica, centrada no processamento de imagens. A (CMU, 2020) se concentra nos fundamentos de renderização, destacando seus principais problemas e soluções, enquanto a (COLUMBIA, 2020) oferece um visão geral de cada passo no pipeline do OpenGL. Já o (MIT, 2020) segue um caminho distinto, priorizando o desenvolvimento de curvas de Bézier com denso embasamento teórico. Nessa semana os seguintes projetos foram desenvolvidos:

4.3.1 **UTAH**: *Compositing Images*

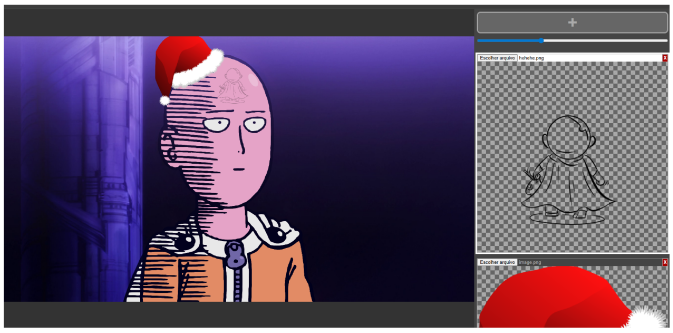
Este projeto tem como foco o ensino e a consolidação do conhecimento do aluno sobre a representação computacional de imagens em forma de dados, adotando a sequência padrão global (RGBA). O trabalho visa também fixar o entendimento sobre alpha blending, processo descrito pela fórmula $f_{color} = f.rgb \cdot f.a + (1 - f.a) \cdot fb.a \cdot fb.rgb$, onde f_{color} representa a cor final resultante, $f.rgb$ e $f.a$ correspondem a cor e a opacidade da imagem frontal, enquanto $fb.rgb$ e $fb.a$ referem-se a cor e a opacidade da imagem no plano de fundo. A Figura 1 ilustra o resultado dos conceitos de composição de imagens (Código desenvolvido disponível em (OLIVEIRA, 2025b)).



(a) Adição de uma imagem.



(b) Composição de duas imagens.

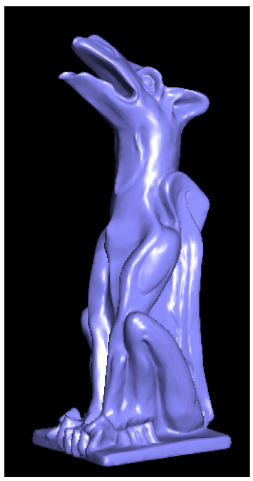


(c) Composição de três imagens com alteração dos valores *alphas*.

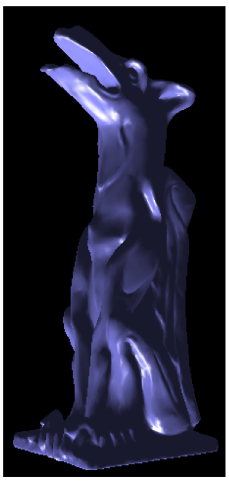
Figura 1 – Processo de composição de imagens.

4.3.2 **MIT**: OpenGL Mesh Viewer

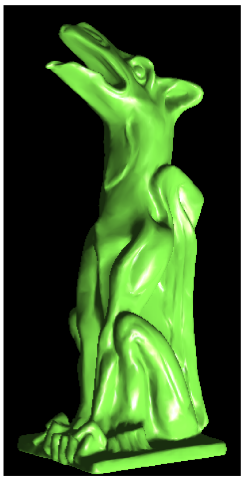
O Projeto 0 do MIT tem como objetivo familiarizar o estudante com o ambiente de desenvolvimento do curso e introduzir conceitos fundamentais de renderização, visando aumentar sua motivação. Para a conclusão do projeto, cujas etapas implementadas podem ser vistas na Figura 2, é necessário desenvolver: um leitor de arquivos .obj, a renderização da malha (utilizando OpenGL legado, para ser mais simples para o aluno) e operações básicas, como a alteração da cor do objeto e o posicionamento da fonte de luz (Código desenvolvido disponível em (OLIVEIRA, 2025c)).



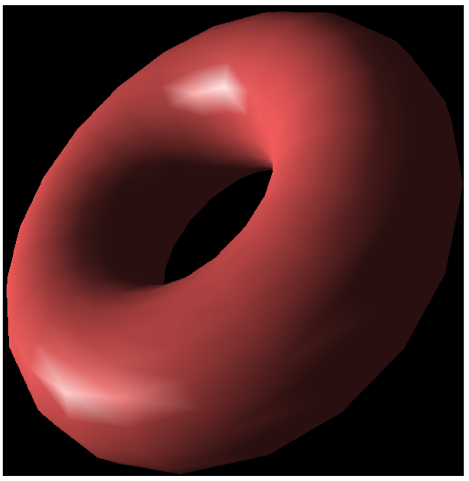
(a) Visualização inicial da malha geométrica carregada.



(b) Simulação de iluminação com fonte de luz móvel.



(c) Manipulação de atributos de cor do objeto.



(d) Validação do renderizador com malha distinta.

Figura 2 – Demonstração das funcionalidades fundamentais implementadas no renderizador.

4.4 Semana 3

Durante a terceira semana, observa-se um padrão comum a todos os cursos analisados: a introdução as transformações geométricas. No entanto, a profundidade e a abordagem variam. A referência (COLUMBIA, 2020) opta por uma explicação mais simplificada, apresentando transformações sem o uso de Coordenadas Homogêneas. Em contrapartida, todas as demais referências introduzem Coordenadas Homogêneas e transformação de perspectivanesta etapa. O curso (UTAH, 2020), por exemplo, busca reduzir a complexidade ao abordar transformações 2D e 3D separadamente. Já (MIT, 2020) se aprofunda no tema, conectando as coordenadas homogêneas e transformações ao estudo de Inverse Kinematics e Hierarchical Modeling. Por fim, (CMU, 2020) cobre o mesmo escopo teórico do MIT, com a exceção de Inverse Kinematics, focando, em seu lugar, na explicação de Quaternions. Nessa semana os seguintes projetos foram desenvolvidos:

4.4.1 **MIT**: Curves and Surfaces

O Projeto 1 tem como foco o estudo e implementação de curvas e superfícies paramétricas. O objetivo principal consiste no desenvolvimento de um avaliador de curvas capaz de gerar cadeias de curvas de Bézier e B-Splines a partir de pontos de controle, cujo comparativo visual pode ser analisado na Figura 3. Além disso, o projeto exige a geração de malhas 3D derivadas dessas curvas. A Figura 4 ilustra as etapas da Superfície de Revolução implementada, gerada por meio da rotação de uma curva perfil ao redor do eixo Y. A etapa seguinte envolve a definição de uma Superfície de Varredura, que utiliza duas curvas distintas para sua formação. Para viabilizar a renderização, foi necessário realizar a tesselação das superfícies, calculando explicitamente vértices, normais e faces. Contudo, não foi possível finalizar a implementação da Superfície de Varredura; a Figura 5 demonstra as falhas da implementação inicial e a correção utilizando a resolução do trabalho de (MORAN, 2012) (Código desenvolvido disponível em (OLIVEIRA, 2025a)).

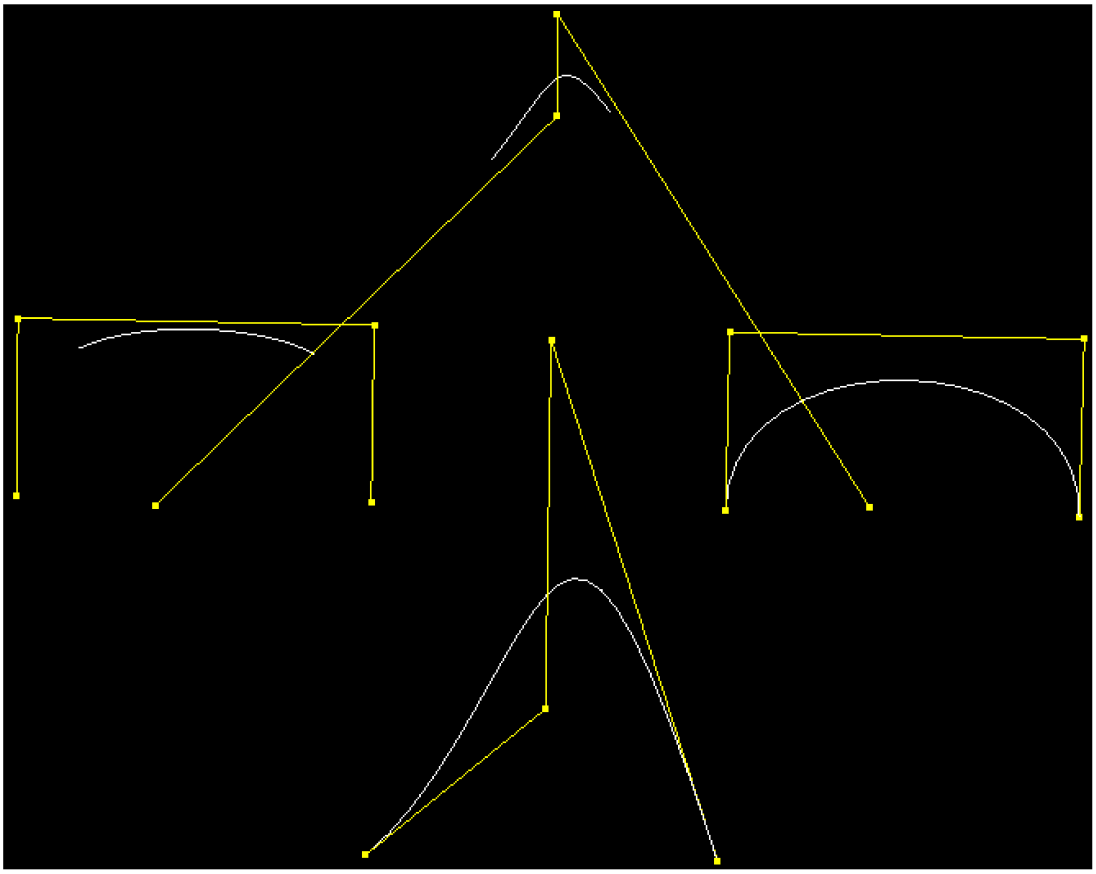
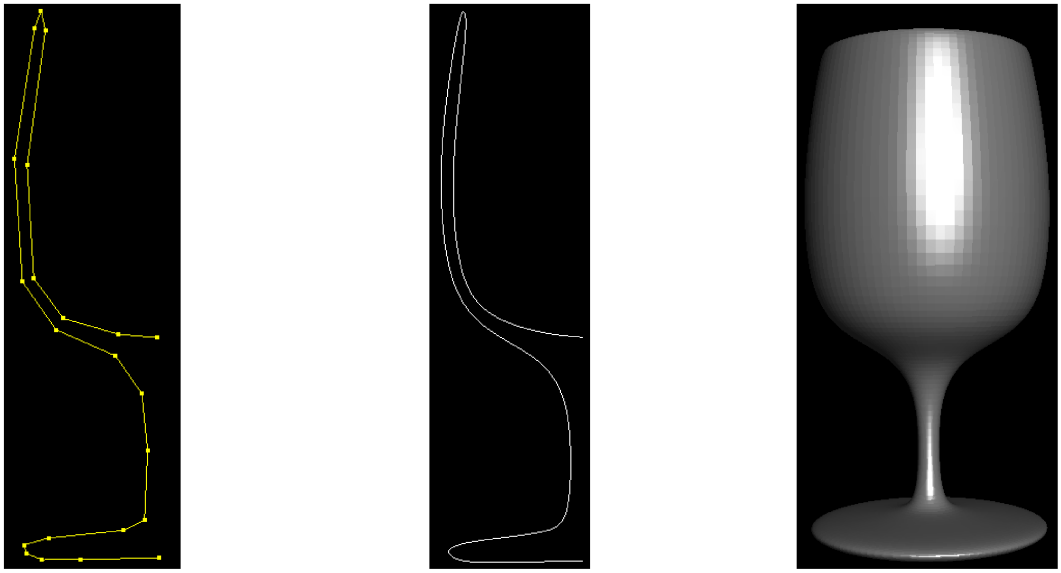
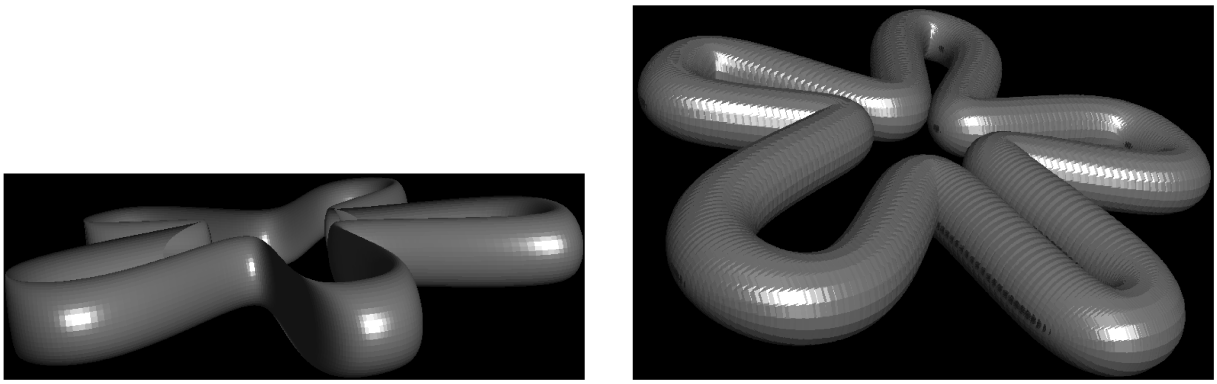


Figura 3 – Comparativo visual entre as implementações de curvas B-Splines (ao topo e à esquerda) e curvas de Bézier (à base e à direita).

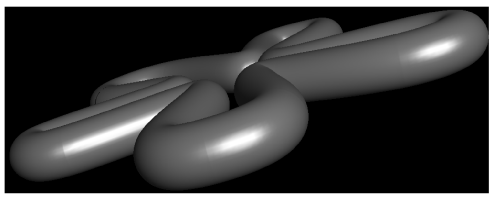


(a) Pontos de controle. (b) Curva B-Spline gerada. (c) Modelo 3D gerado.

Figura 4 – Etapas da geração de uma superfície de revolução: definição dos pontos de controle, interpolação da curva e malha 3D resultante.



(a) Falha na topologia da malha. (b) Geometria ineficiente com artefatos.



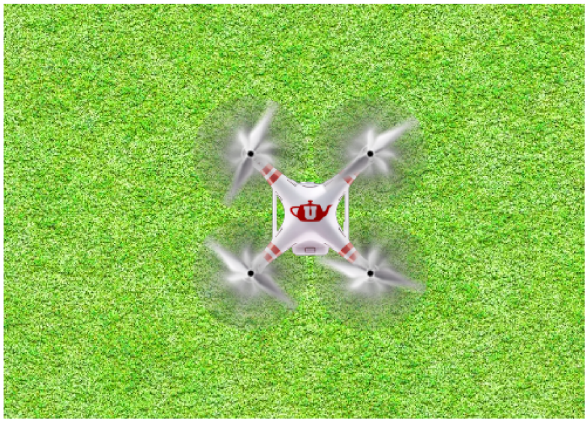
(c) Solução de referência (MORAN, 2012).

Figura 5 – Análise iterativa da implementação da superfície de varredura (*sweep*), evidenciando falhas iniciais e a geometria de referência correta.

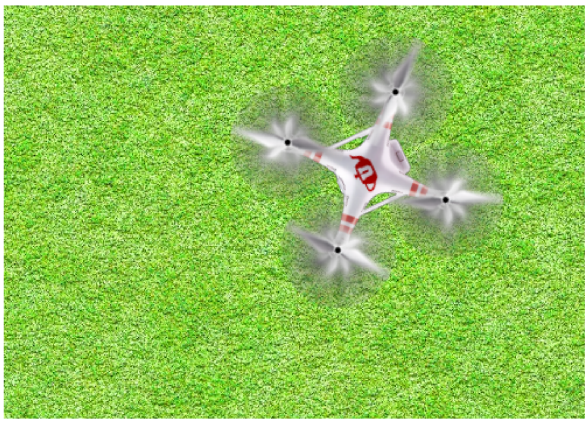
4.4.2 **UTAH**: Transformations

O segundo projeto de (UTAH, 2020) tem como objetivo introduzir a aplicação prática da teoria de transformações, cujos resultados visuais podem ser analisados na Figura 6. A atividade foca na consolidação do conhecimento sobre a ordem de composição das transformações 2D, bem como na implementação de algoritmos de multiplicação de matrizes representadas em memória como 1D *arrays* contíguos em formato *Column-Major*. As matrizes de transformação e a operação final são definidas a seguir (Código desenvolvido disponível em (OLIVEIRA, 2025d)):

$$
\begin{aligned}
M_{scale} &= \begin{bmatrix} scaleX & 0 & 0 \\ 0 & scaleY & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
M_{translation} &= \begin{bmatrix} 1 & 0 & translateX \\ 0 & 1 & translateY \\ 0 & 0 & 1 \end{bmatrix} \\
M_{rotation} &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
M_{final} &= M_{translation} \cdot M_{rotation} \cdot M_{scale}
\end{aligned}
$$



(a) Visualização da cena do projeto finalizado.



(b) Aplicação de rotação ao objeto.

(c) Aplicação de escala (*zoom out*) combinada com rotação.

Figura 6 – Demonstração das interações implementadas no projeto, englobando a cena inicial e as transformações de rotação e escala.

4.5 Semana 4

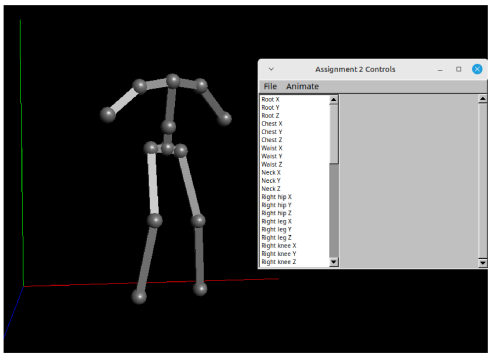
Durante a quarta semana, a (COLUMBIA, 2020) foca no desenvolvimento de transformações, mantendo um ritmo mais ameno que as demais. Nesta etapa, são apresentadas a composição de transformações, *Forward* e *Inverse Kinematics* (sendo este último melhor desenvolvido em material suplementar) e transformações de câmera. Já a (MIT, 2020) concentra-se em simulação de partículas e na resolução de EDOs, abordando os métodos de Euler, *Trapezoid* e RK4. O curso discute a estabilidade e acurácia de tais modelos, além de apresentar conceitos físicos básicos de *drag*, sistema massa-mola e a utilização de sistemas de EDOs para simulação. (CMU, 2020) expande brevemente as transformações com projeção em perspectiva, mas a ênfase recai sobre a aplicação de texturas e interpolação de cores. São desenvolvidos tópicos como coordenadas baricêntricas e de textura (UV), *aliasing* (filtro bilinear, trilinear e *mipmaps*), oclusão via *Z-buffer* e composição de imagens, abordando composição de transparência (*alpha blending*) com e sem pré-multiplicação. Diferentemente das demais, a (UTAH, 2020) adota uma abordagem mais prática, explicando de forma simplificada o funcionamento e os estágios da GPU; no entanto, o foco central recai sobre o WebGL, detalhando a sintaxe da API e sua utilização. Nessa semana os seguintes projetos foram desenvolvidos:

4.5.1 **MIT**: Hierarchical Modeling & SSD

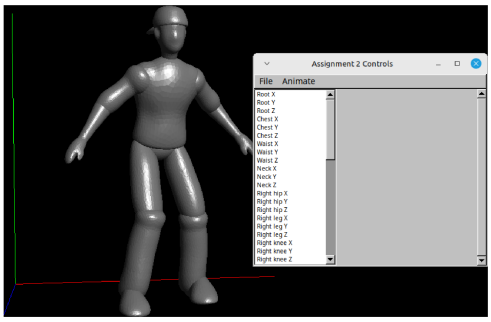
Este projeto tem como foco a consolidação de *Hierarchical Modeling*, *Forward Kinematics* e *Skeletal Subspace Deformation* (SSD). A matriz de transformação para cada objeto é definida por $M_{model} = M_{parent} \cdot M_{local}$. Para a computação de SSD, utiliza-se a seguinte formulação:

$$p'_{ij} = T_j \cdot p_i$$
$$p_{final} = \sum_j (w_{ij} \cdot p'_{ij})$$

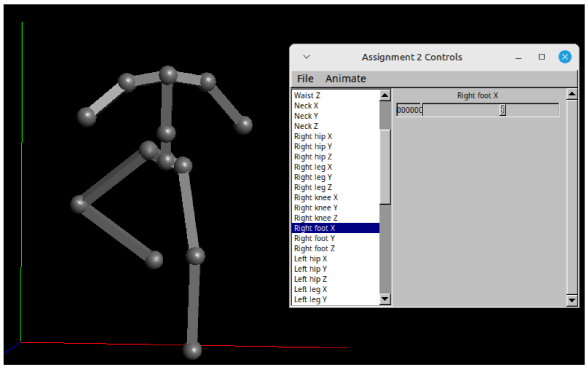
Onde p'_{ij} representa o vértice i transformado pelo osso j, T_j a matriz de transformação do osso j, p_i a posição original do vértice i e w_{ij} o peso de influência da transformação do osso j sobre o vértice i. As funcionalidades implementadas e o impacto visual da deformação SSD podem ser observados na Figura 7 (Código desenvolvido disponível em (OLIVEIRA, 2026c)).



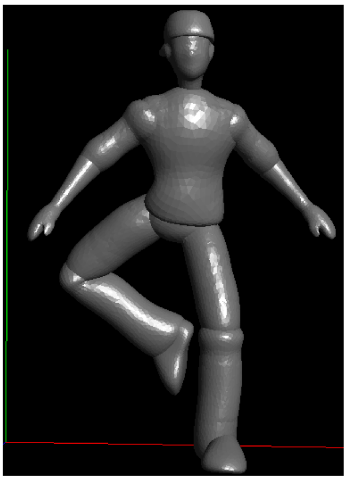
(a) Interface inicial exibindo os controles de manipulação dos ossos.



(b) Visualização da malha do modelo 3D carregado.



(c) Aplicação de transformações geométricas à estrutura esquelética.



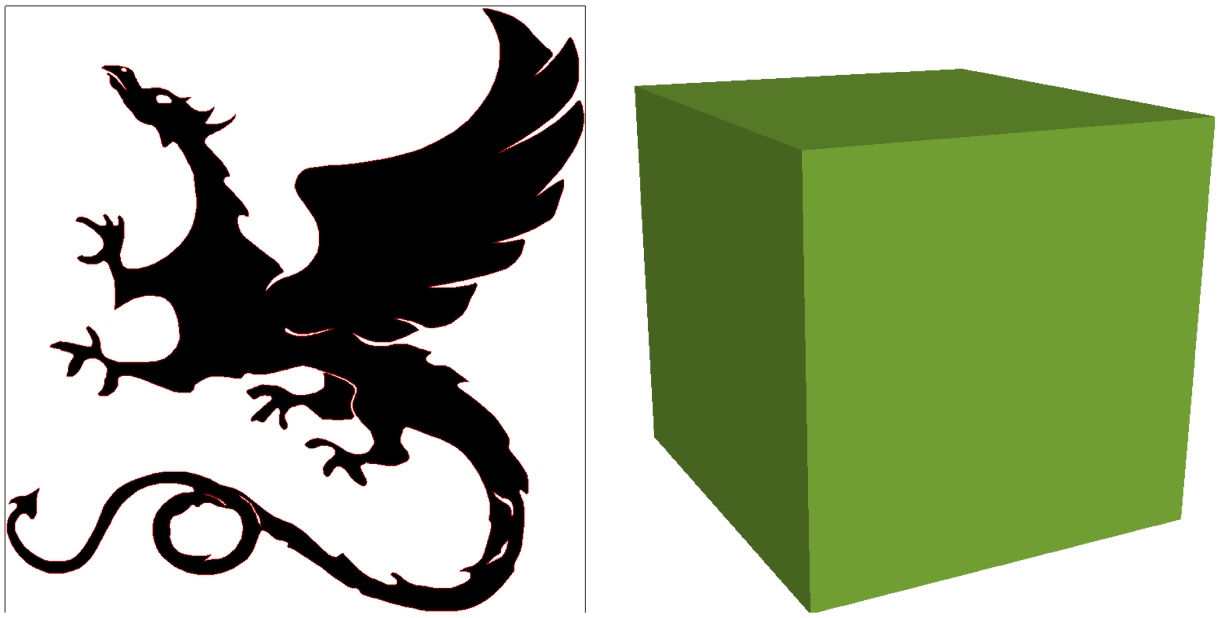
(d) Modelo 3D com deformação SSD aplicada com base nos ossos.

Figura 7 – Visão geral do projeto Modelagem hierárquica e SSD.

4.5.2 **CMU**: DrawSVG 01/02

Este projeto tem como foco a consolidação do entendimento básico de rasterização e aliasing. Ele é estruturado em 8 tarefas e duas entregas; esta seção cobre o desenvolvimento das Tarefas 1 a 4 (Código desenvolvido disponível em (OLIVEIRA, 2026b)).

Tarefa 1: Desenvolvimento de *hardware rendering* utilizando a versão legada do OpenGL (anterior à versão 3.0). Foram implementadas as funções de primitivas definidas no escopo: rasterize_point, rasterize_line e rasterize_triangle, com os resultados processados visualizados na Figura 8.



(a) Renderização acelerada por *hardware*: modelo de dragão com linhas e triângulos.

(b) Renderização acelerada por *hardware*: caixa 3D construída via triângulos.

Figura 8 – Exemplos de processamento de primitivas gráficas utilizando renderização acelerada por *hardware*.

Tarefa 2: Transição para a rasterização via CPU. Implementou-se a rasterização de linhas, abrangendo algoritmos capazes de processar segmentos de reta com qualquer inclinação (*slope*), conforme ilustrado na Figura 9.

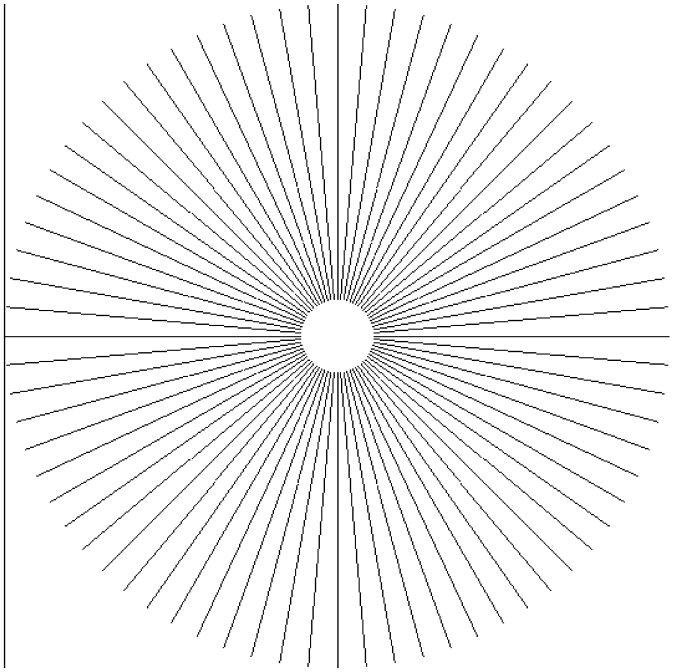


Figura 9 – CPU rendering: linhas.

Tarefa 3: Rasterização de triângulos na CPU, cujo resultado prático pode ser visto na Figura 10. Para otimização, identifica-se uma *bounding box* para restringir a varredura de *pixels*. A verificação se um ponto P pertence ao triângulo definido pelos vértices A, B e C é realizada por meio do cálculo de áreas (princípio das coordenadas baricêntricas). A área de um sub-triângulo, como o formado por A e B em relação a P, é dada por $Area_{ABP} = \frac{||\vec{AB} \times \vec{AP}||}{2}$. A condição de pertinência é validada pela seguinte igualdade:

$$Area_{total} = Area_{ABC}$$
$$Area_{total} = Area_{ABP} + Area_{BCP} + Area_{CAP}$$

Se a soma das áreas dos sub-triângulos for igual à área total, o ponto P encontra-se no interior do triângulo.

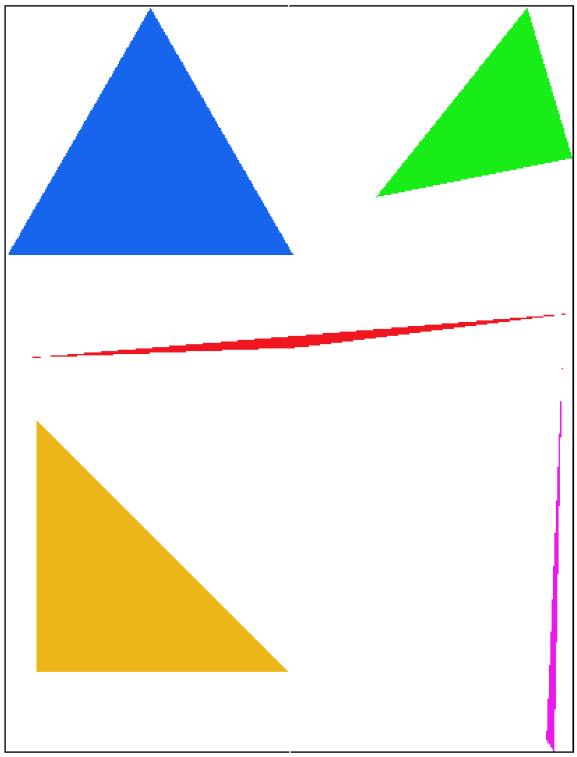
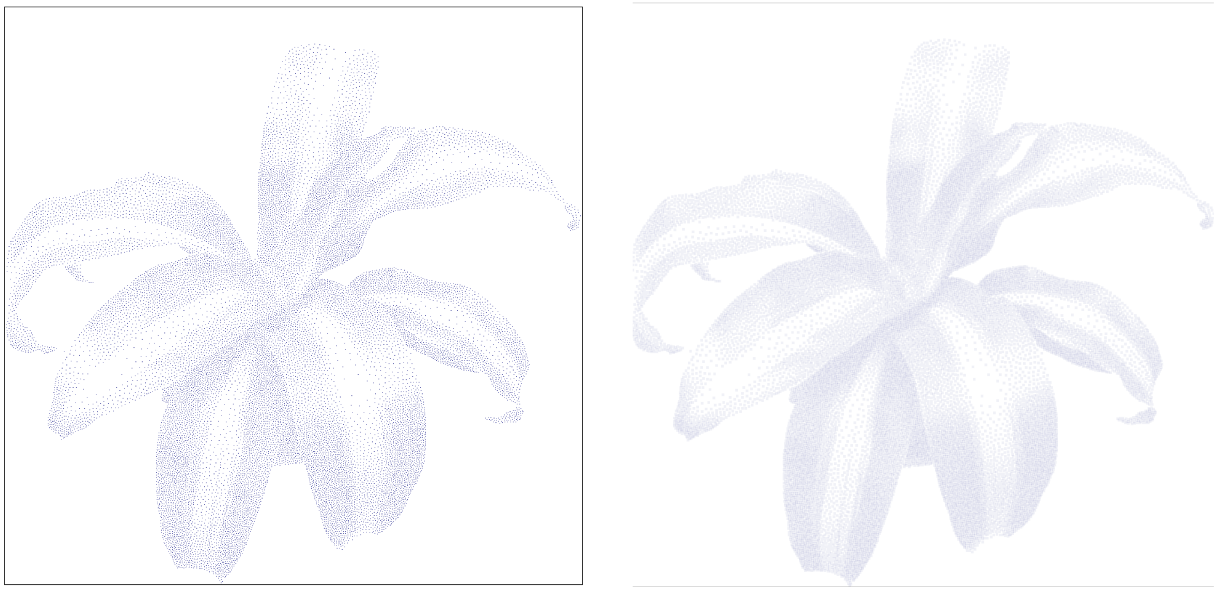


Figura 10 – CPU rendering: triângulos.

Tarefa 4: Implementação de *Anti-aliasing* utilizando *Supersampling*. A técnica consiste em aumentar a resolução do *render target* e calcular a cor final do *pixel* por meio da média dos valores amostrados na resolução superior (*downsampling*). O efeito dessa técnica na atenuação do serrilhado é demonstrado no comparativo da Figura 11.



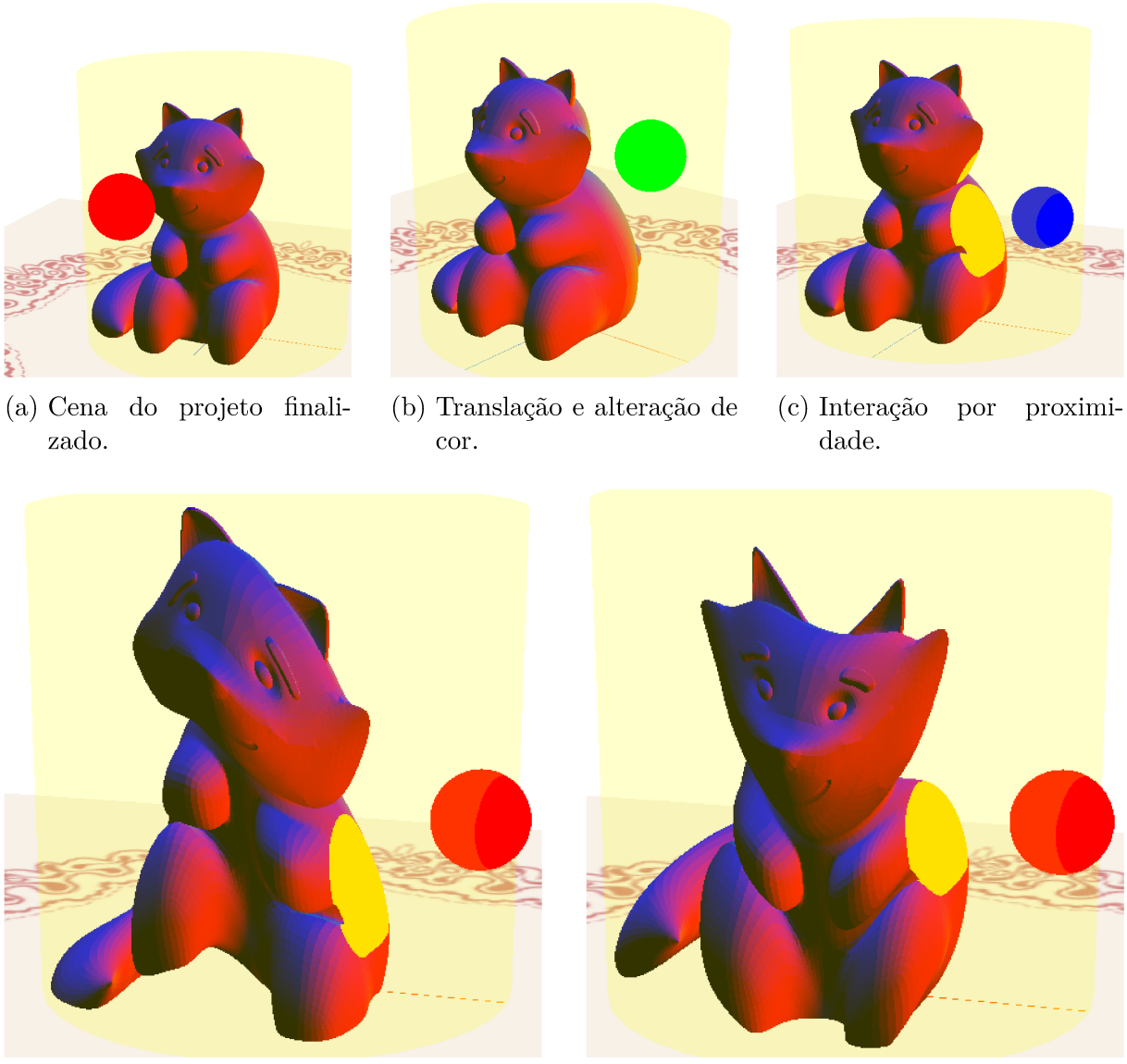
(a) Renderização sem *supersampling*.

(b) Renderização com *supersampling*.

Figura 11 – Comparativo de renderização evidenciando o efeito da técnica de *supersampling* na atenuação do serrilhado (*aliasing*).

4.5.3 **UBC**: Intro

Este projeto tem como objetivo principal introduzir os alunos ao ambiente de desenvolvimento utilizando a API Three.js. A ênfase recai sobre a passagem de informações para os *shaders*, a aplicação de transformações geométricas básicas e a representação de interações entre os objetos. Além disso, propõe-se uma etapa de escopo aberto, estimulando a criatividade do aluno para explorar e expandir o que foi aprendido. A Figura 12 ilustra a cena construída, demonstrando as transformações e interações implementadas (Código desenvolvido disponível em (OLIVEIRA, 2026d)).



(a) Cena do projeto finalizado.

(b) Translação e alteração de cor.

(c) Interação por proximidade.

(d) Deformação por função seno no *vertex shader*.

(e) Variação temporal da geometria.

Figura 12 – Visão geral do projeto introdutório da UBC.

4.6 Semana 5

O curso (UTAH, 2020) inicia a semana introduzindo curvas e suas representações com um denso embasamento teórico, de forma semelhante ao que foi abordado na semana 2 por (MIT, 2020). O foco desta semana em (CMU, 2020) é a representação geométrica implícita e explícita. São abordados diversos modelos de representação, tais como CSG (*Constructive Solid Geometry*), curvas de Bézier, fractais, nuvens de pontos e malhas triangulares. Também é apresentada a estrutura de dados *half-edge* para a representação de malhas e como realizar operações sobre elas para alterar a topologia e a geometria.

Por sua vez, (COLUMBIA, 2020) concentra-se em desenvolver a intuição por trás das projeções de câmera, abrangendo tanto a projeção ortográfica quanto a perspectiva. Em contrapartida, (MIT, 2020) foca na introdução à renderização. Diferente de (UTAH, 2020) e (COLUMBIA, 2020), inicia o conteúdo de renderização dando ênfase *ray casting*, apresentando como calcular a interseção de raios com geometrias implícitas e explícitas. Nessa semana os seguintes projetos foram desenvolvidos:

4.6.1 **MIT**: Physically-based simulation

Este projeto foca em consolidar o conhecimento do aluno em simulação física, baseando-se na resolução de sistemas de Equações Diferenciais Ordinárias (EDOs). Para isso, foram desenvolvidos solvers utilizando os métodos de Euler e do Trapézio, definidos pelas seguintes fórmulas:

Método de Euler:

$$X(t+h) = X(t) + hf(X,t)$$

Método do Trapézio:

$$\begin{aligned} f_0 &= f(X,t) \\ f_1 &= f(X + hf_0, t+h) \\ X(t+h) &= X(t) + \frac{h}{2}(f_0 + f_1) \end{aligned}$$

Para validar a implementação, utiliza-se um sistema simples definido por:

$$X(t) = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad f(X,t) = \begin{bmatrix} -y \\ x \\ 0 \end{bmatrix}$$

Sendo a sua aproximação dada por:

$$X(t) = \begin{bmatrix} r\cos(t+k) \\ r\sin(t+k) \end{bmatrix}$$

As forças aplicadas no sistema físico são descritas pelas seguintes equações:

- Força gravitacional: $F(x_i, x_i', m_i) = m_i g$
- Força de arrasto: $F(x_i, x_i', m_i) = -kx_i'$
- Força envolvendo partículas conectadas (molas): $F(x_i, x_i', m_i) = -k(||d|| - r)\left(\frac{d}{||d||}\right)$, onde $d = x_i - xj$.

O movimento para cada partícula pode ser modelado pela seguinte EDO de segunda ordem:

$$x'' = F(x, x')$$

Como os solvers implementados resolvem apenas sistemas de primeira ordem, transforma-se a EDO de segunda ordem em um sistema equivalente com um par de equações de primeira ordem:

$$\begin{bmatrix} x' \\ v' \end{bmatrix} = \begin{bmatrix} v \\ F(x,v) \end{bmatrix}$$

Dessa forma, o solver resolve cada variável separadamente e o sistema de EDOs final assume a seguinte forma:

$$X = \begin{bmatrix} x \\ v \end{bmatrix}, \quad \frac{d}{dt}X = f(X,t) = \begin{bmatrix} v \\ F(x,v) \end{bmatrix}$$

Tanto para o pêndulo quanto para a simulação de tecido, a base física é a mesma; a principal diferença reside em quais partículas exercem força umas sobre as outras. Como exemplo, para uma partícula interna i de um pêndulo, as forças elásticas atuantes provêm de seus vizinhos imediatos $i-1$ e $i+1$. Portanto, a força elástica resultante é a soma dessas duas conexões (Código desenvolvido disponível em (OLIVEIRA, 2026e)):

$$F_{total} = \left[-k(||d_1|| - r)\left(\frac{d_1}{||d_1||}\right)\right] + \left[-k(||d_2|| - r)\left(\frac{d_2}{||d_2||}\right)\right]$$

onde $d_1 = x_i - x_{i-1}$ e $d_2 = x_i - x_{i+1}$.

Para uma visualização clara dos resultados, o vídeo de demonstração das simulações físicas implementadas pode ser acessado ao clicar na Figura 13.

Figura 13 – Clique na imagem para assistir ao vídeo de demonstração do projeto.

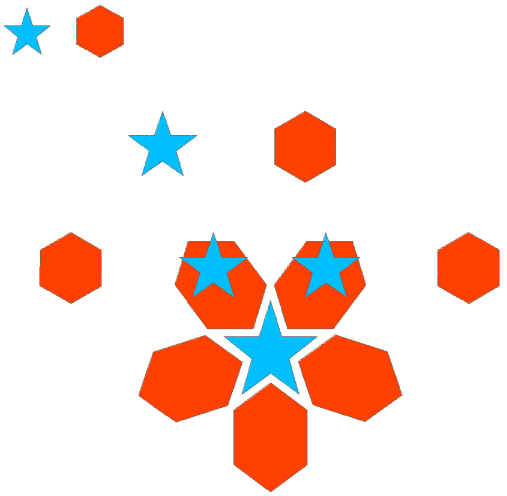
4.6.2 **CMU**: DrawSVG 02/02

A segunda parte desse projeto foca na consolidação dos conhecimentos de transformações hierárquicas e *viewing*, *aliasing* em imagens e composição de transparência. Esta seção cobre o desenvolvimento das Tarefas 5 a 8 (Código desenvolvido disponível em (OLIVEIRA, 2026b)).

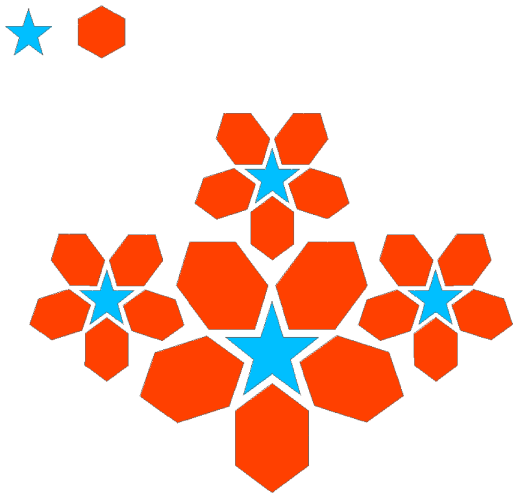
Tarefa 5: A Tarefa 5 aborda a renderização de cenas com transformações hierárquicas (definidas por $M_{model} = M_{parent} \cdot M_{local}$) e a aplicação de transformações de visualização (*viewing transformations*). O mapeamento para o espaço da câmera utiliza a seguinte matriz:

$$M_{view} = \begin{bmatrix} \frac{1}{2 \cdot vspan} & 0 & \frac{-(centerX - vspan)}{2 \cdot vspan} \\ 0 & \frac{1}{2 \cdot vspan} & \frac{-(centerY - vspan)}{2 \cdot vspan} \\ 0 & 0 & 1 \end{bmatrix}$$

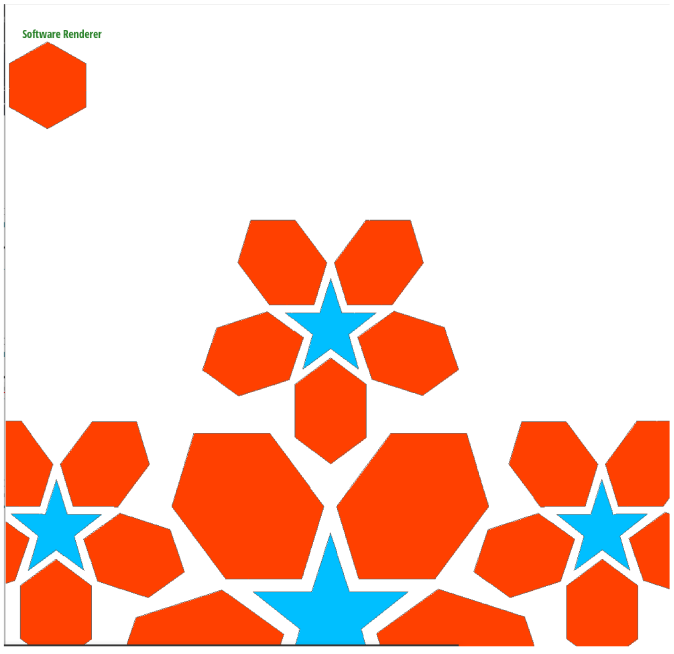
onde *vspan* é a meia-extensão da janela da *viewbox*. O impacto da aplicação dessas matrizes de transformação na cena pode ser visualizado na Figura 14.



(a) Exemplo de renderização sem a aplicação de transformações hierárquicas.



(b) Exemplo de renderização com a correta aplicação de transformações hierárquicas.



(c) Renderização após transformações de visualização (translação e escala).

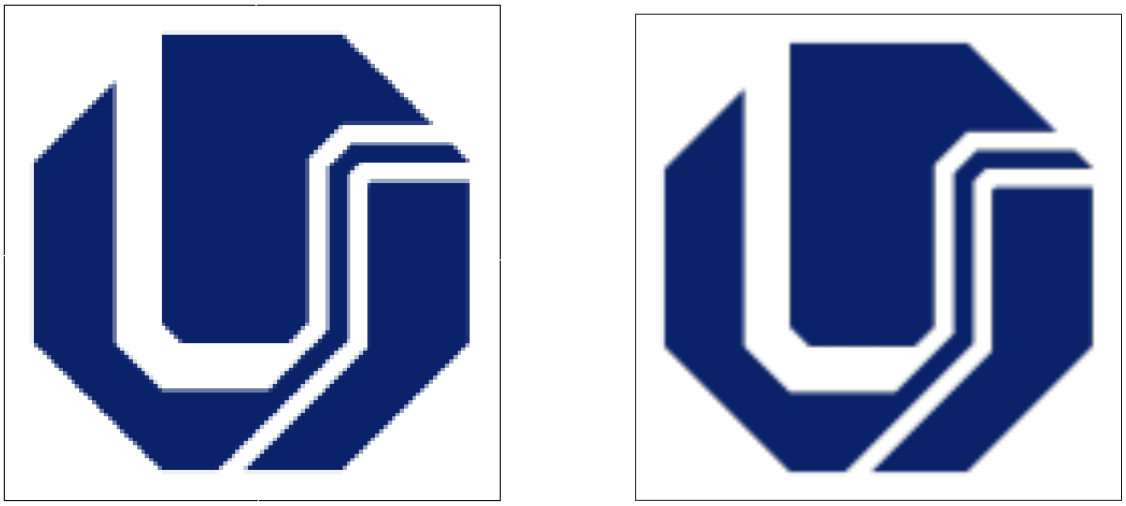
Figura 14 – Comparativo estrutural evidenciando o impacto das transformações hierárquicas e demonstração das transformações de visualização aplicadas à cena.

Tarefa 6: Rasterização de imagens com redimensionamento e mapeamento de texturas. A função *rasterize_image* processa a *bounding box* da imagem, mapeando os *pixels* para as coordenadas normalizadas da textura $(u, v) \in [0, 1]^2$. Para o cálculo de cor, implementou-se o filtro bilinear, utilizando os 4 *texels* (*texture elements*) mais próximos

e fazendo interpolação:

$$
\begin{aligned}
t &= (u \cdot \text{width} - 0.5) - x_{idx} \\
b &= (v \cdot \text{height} - 0.5) - y_{idx} \\
t_f &= t_1 \cdot t + t_0 \cdot (1 - t) \\
b_f &= b_1 \cdot t + b_0 \cdot (1 - t) \\
f &= t_f \cdot (1 - b) + b_f \cdot b
\end{aligned}
$$

Onde u e v são as coordenadas normalizadas da textura; *width* e *height* representam as dimensões da textura (largura e altura) no nível de *mipmap*; x_{idx} e y_{idx} são os índices do *texel*; t e b são as distâncias, utilizadas como pesos para a interpolação nos eixos horizontal e vertical, respectivamente; t_f e b_f são as cores intermediárias resultantes das interpolações horizontais na parte superior e inferior. Isso resulta em imagens redimensionadas mais suaves e com menos serrilhadas, conforme o comparativo apresentado na Figura 15.



(a) Rasterização de textura com *nearest neighbor*.

(b) Rasterização de textura aplicando filtragem bilinear.

Figura 15 – Comparativo de escalonamento de textura entre os métodos *nearest neighbor* e bilinear.

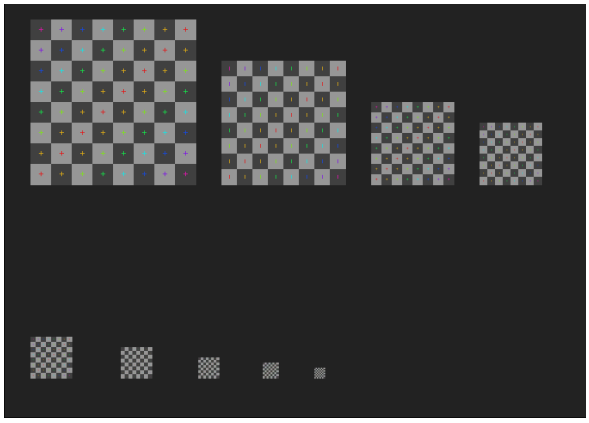
Tarefa 7: Implementação de *antialiasing* em texturas utilizando filtragem trilinear. A função *generate_mips* foi implementada para pré-computar níveis de *mipmaps* onde cada nível tem a metade do tamanho do anterior. Para determinarmos qual nível de *mipmaps* devemos usar, utilizamos a seguinte fórmula:

$$L = \max(0, \log_2(\max(L_x, L_y)))$$

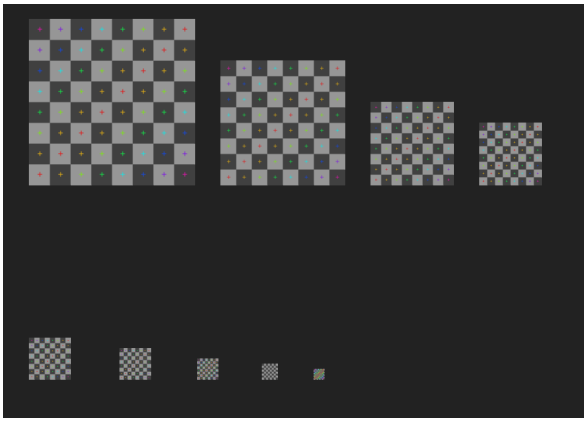
A função *sample_trilinear* então interpola linearmente as cores dos dois níveis de *mipmap* mais próximos, utilizando a seguinte formulação:

$$\begin{aligned} \text{level}_0 &= \lfloor L \rfloor \\ \text{level}_1 &= \lceil L \rceil \\ C_0 &= \text{sample\_bilinear}(u, v, \text{level}_0) \\ C_1 &= \text{sample\_bilinear}(u, v, \text{level}_1) \\ C_f &= C_1 \cdot \text{blend} + C_0 \cdot (1 - \text{blend}) \end{aligned}$$

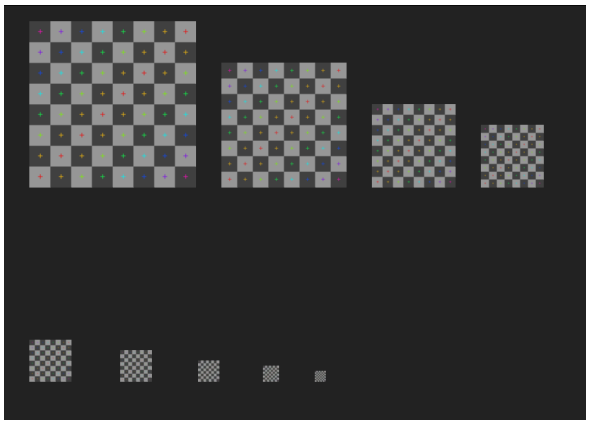
Os resultados visuais dos diferentes métodos de filtragem implementados estão expostos na Figura 16.



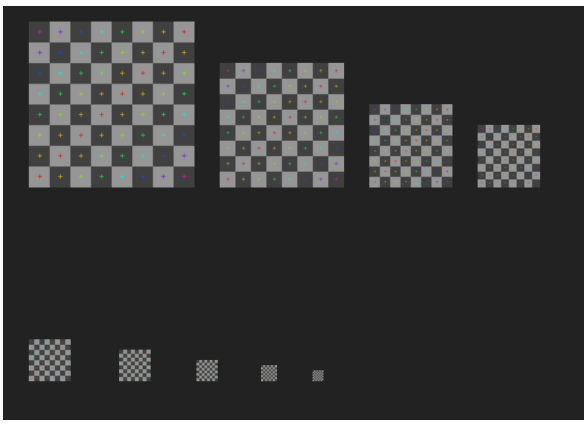
(a) Filtragem *Nearest Neighbor*.



(b) Filtragem Bilinear.



(c) Filtragem Bilinear com *Mipmaps*.



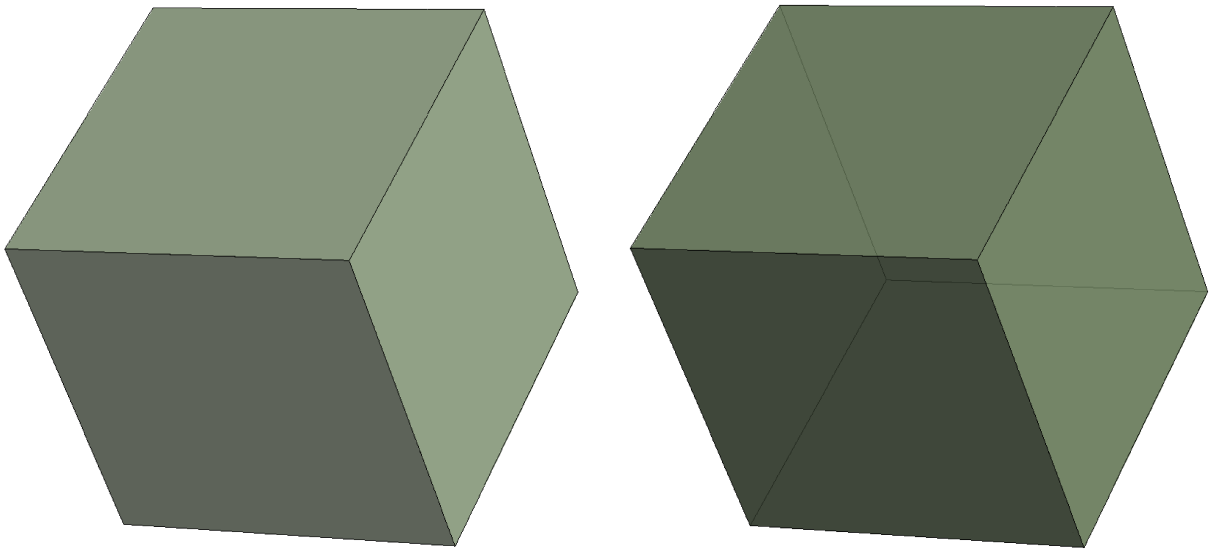
(d) Filtragem Trilinear.

Figura 16 – Comparativo visual dos diferentes métodos de filtragem de textura implementados.

Tarefa 8: Implementação da composição de transparência (*alpha blending*). O objetivo desta tarefa é realizar a composição de cores da cena utilizando o conceito de *alpha* pré-multiplicado. A operação matemática para combinar a nova cor com a cor já existente no *buffer* foi implementada da seguinte forma:

$$\alpha_{final} = \alpha_c + (1 - \alpha_c) \cdot \alpha_0$$
$$C_f = (C \cdot \alpha_c + C_0 \cdot (1 - \alpha_c)) \cdot \alpha_{final}$$

Onde C_0 é a cor atual no *frame buffer*. A Figura 17 evidencia a diferença na renderização com e sem a aplicação dessa composição.



(a) Renderização padrão sem a composição.

(b) Renderização aplicando o *alpha* pré-multiplicado.

Figura 17 – Comparativo de renderização com e sem a aplicação de *alpha blending*.

4.7 Semana 6

Na sexta semana, os cursos aprofundam as técnicas de modelagem geométrica e de renderização. A referência (UTAH, 2020) foca no desenvolvimento de superfícies implícitas e explícitas, dando ênfase às malhas triangulares e como utilizar essas estruturas no *pipeline* da GPU, especificamente no contexto de WebGL. (CMU, 2020) continua exatamente de onde parou na semana anterior, explorando operações sobre a estrutura de dados *half edge* para alterações sistemáticas na topologia. O curso detalha algoritmos de subdivisão como *Catmull Clark* e *Loop*, simplificação via *edge collapse* e apresenta a minimização da métrica de erro quadrático para obter uma melhor aproximação da malha original com menos polígonos. Além disso, desenvolve a base matemática para calcular a interseção de raios com planos e superfícies implícitas.

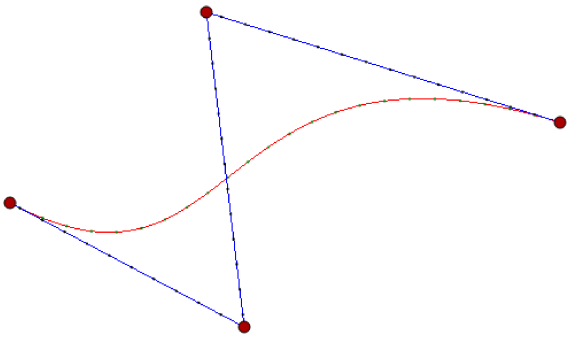
(MIT, 2020) avança nos conceitos de *ray tracing* introduzidos na semana passada. O foco recai sobre a geração de efeitos secundários como sombras, reflexos e refração, e em otimização por meio de estruturas de dados espaciais como BVH (*Bounding Volume Hierarchy*), *kd trees* e *ray marching*. Em contrapartida, a (COLUMBIA, 2020) tem como foco o desenvolvimento de algoritmos básicos para rasterização. São abordados o *clipping*, que é o descarte de triângulos e geometria fora do *view frustum*, coordenadas baricêntricas e algoritmos para melhorar o desempenho da rasterização *pixel* a *pixel*, incluindo *Bounding Volumes*, *Scan Line* e *Hierarchical Rasterization* especificamente para execução na GPU. Nessa semana os seguintes projetos foram desenvolvidos:

4.7.1 **UTAH**: Curves

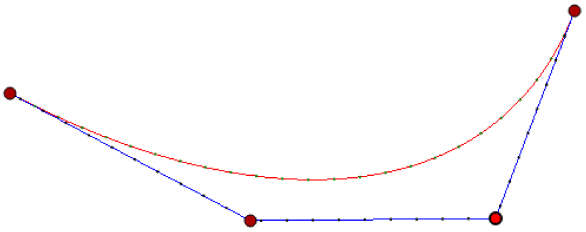
Este projeto tem como foco consolidar o conhecimento prático sobre curvas e o desenvolvimento na API gráfica WebGL. A implementação foi baseada na formulação cúbica da curva de Bézier para a construção da geometria. A interpolação espacial para encontrar a posição na curva é definida pela seguinte equação:

$$f(t) = (1-t)^3 \cdot p_0 + 3(1-t)^2 \cdot t \cdot p_1 + 3(1-t) \cdot t^2 \cdot p_2 + t^3 \cdot p_3$$

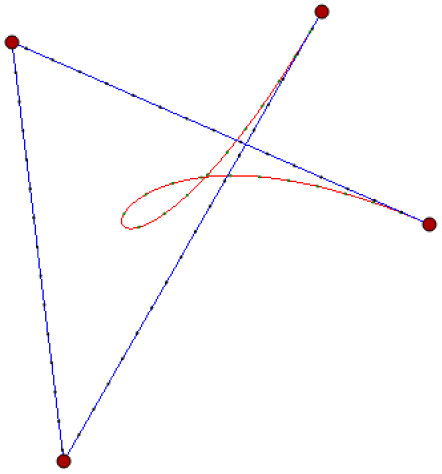
Onde p_0, p_1, p_2 e p_3 representam os pontos de controle da curva e t é o parâmetro de interpolação que varia de 0 a 1. Todo o algoritmo de cálculo das posições foi executado diretamente no *vertex shader*, o que permite aproveitar o poder de paralelização da GPU para otimizar o desempenho da renderização. A Figura 18 demonstra a curva sob diferentes configurações de seus pontos de controle(Código desenvolvido disponível em (OLIVEIRA, 2026a)).



(a) Visualização inicial da curva com seus pontos de controle.



(b) Alteração da geometria pela movimentação dos pontos.



(c) Configuração gerando uma autointerseção na curva.

Figura 18 – Análise da curva de Bézier cúbica sob diferentes configurações de pontos de controle.

4.7.2 **UBC**: Transformations

Este projeto tem como foco as transformações lineares, abordando especificamente as transformações hierárquicas e a implementação de forward kinematics. A transformação cada objeto dentro da hierarquia é definida pela seguinte relação (Código desenvolvido disponível em (OLIVEIRA, 2026j)):

$$M_{model} = M_{parent} \cdot M_{local}$$

A demonstração visual do projeto em funcionamento pode ser conferido através do vídeo referenciado na Figura 19.

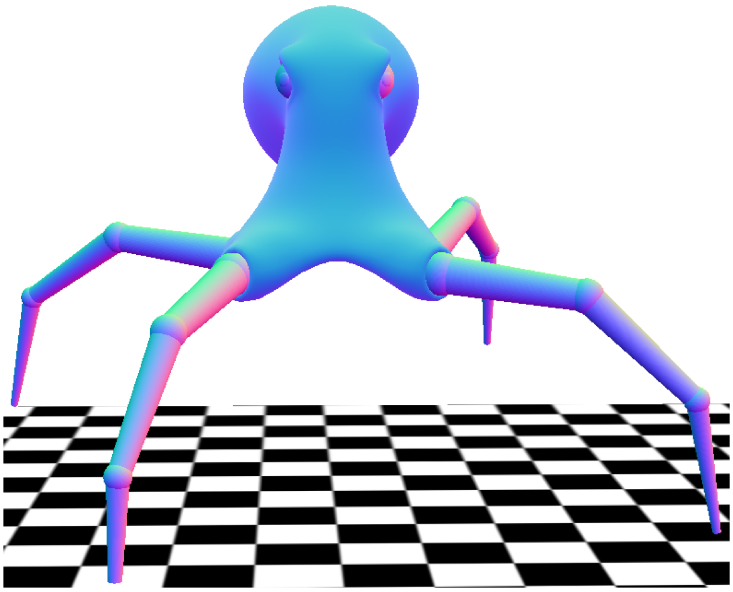


Figura 19 – Clique na imagem para assistir ao vídeo de demonstração do projeto.

4.8 Semana 7

A referência (UTAH, 2020) avança para o estudo de texturas. O foco recai sobre o *texture mapping*, detalhando o processo de mapear uma imagem 2D em uma malha triangular 3D. Em seguida, são abordadas técnicas de *sampling*, com destaque para os filtros *nearest*, bilinear e trilinear. Para dar suporte à filtragem trilinear, o material desenvolve o conceito de *mipmaps* e os métodos para a sua criação. Por fim, o curso analisa o comportamento de todo esse processo na GPU, contextualizando com uma aplicação prática na API gráfica WebGL.

Por sua vez, utilizando os conceitos construídos nas últimas duas semanas, a referência (CMU, 2020) continua o desenvolvimento matemático abordando o cálculo de interseção de raios com triângulos. Após consolidar essa base, o curso direciona a atenção para a renderização por meio de *ray casting*, apresentando a lógica do procedimento em formato de pseudocódigo. Além disso, introduz as estruturas de dados espaciais necessárias para acelerar esse processo, como BVH e *kd trees*. Por fim, o material explora a teoria da cor, com foco principal nos seus métodos de representação.

De maneira complementar, a referência (MIT, 2020) avança nos conceitos necessários para a construção de um renderizador completo com base em *ray casting*. O material detalha BRDF (*Bidirectional Reflectance Distribution Function*), que consiste na função matemática utilizada para medir a porção de luz proveniente de uma direção que é refletida em outra. Dentro desse tópico, há uma ênfase no modelo de reflexão de Phong e na variação proposta por Blinn, conhecida como modelo Blinn Phong. O curso também explora o tema de texturas, abordando *texture mapping* e conceitos básicos de *mipmaps*, além de apresentar técnicas como *normal mapping* e a criação de *shaders* combinados com *Perlin noise*.

Assim como a referência (MIT, 2020), o curso (COLUMBIA, 2020) foca no cálculo de iluminação e na definição da BRDF. Na primeira aula, o conteúdo aborda o cálculo de interpolação, com ênfase na interpolação bilinear e nas coordenadas baricêntricas, além de introduzir a base teórica da BRDF. Na segunda aula, a atenção se volta exclusivamente para os modelos de *shading*. O material explica o conceito de *flat shading*, demonstra como o *gouraud shading* interpola as normais durante o cálculo de luz e define o *phong shading*, incluindo a sua variação conhecida como modelo *blinn phong*. Por fim, o curso detalha os diferentes tipos de fontes de luz, como *directional light*, *point light* e *spot light*. Nessa semana os seguintes projetos foram desenvolvidos:

4.8.1 **MIT**: Ray casting

Este projeto desenvolve um renderizador baseado em *ray casting*, servindo como base para o próximo projeto de *ray tracer*. A estrutura gera um raio para cada *pixel* e testa a interseção com os objetos da cena. A direção do raio no espaço é calculada pela inversa das matrizes de perspectiva e visualização:

$$d = \text{normalize}((M_{perspective} \cdot M_{lookAt})^{-1} \cdot P_{ndc})$$

Onde P_{ndc} é a coordenada com $z = -1$ e o raio é definido por $R = O + t \cdot d$.

Para a interseção com triângulos, o algoritmo encontra a distância t até o plano do triângulo e valida o ponto calculando as coordenadas baricêntricas α, β e γ pelas áreas dos subtriângulos:

$$\begin{aligned}
t &= \frac{N \cdot (C - O)}{N \cdot d} \\
P &= O + t \cdot d \\
\alpha &= \frac{||(V_1 - P) \times (V_2 - P)||}{2 \cdot A_{total}} \\
\beta &= \frac{||(V_2 - P) \times (V_0 - P)||}{2 \cdot A_{total}} \\
\gamma &= \frac{||(V_0 - P) \times (V_1 - P)||}{2 \cdot A_{total}}
\end{aligned}$$

Se a soma das coordenadas for aproximadamente 1, a interseção é confirmada. As normais e coordenadas de textura são então interpoladas usando as coordenadas baricêntricas.

A interseção com esferas é resolvida extraindo as raízes da equação do segundo grau formada pela projeção do raio:

$$\begin{aligned}
v &= C_{esfera} - O \\
b &= v \cdot d \\
\Delta &= b^2 - ||v||^2 + R^2_{esfera}
\end{aligned}$$

Se $\Delta \geq 0$, as interseções ocorrem nas distâncias $t_{1,2} = b \pm \sqrt{\Delta}$, sendo selecionada a menor raiz positiva válida.

Para o plano, a interseção calcula a distância onde o raio cruza a superfície infinita definida por sua normal N:

$$t = \frac{N \cdot (2D_{plano}N - O)}{N \cdot d}$$

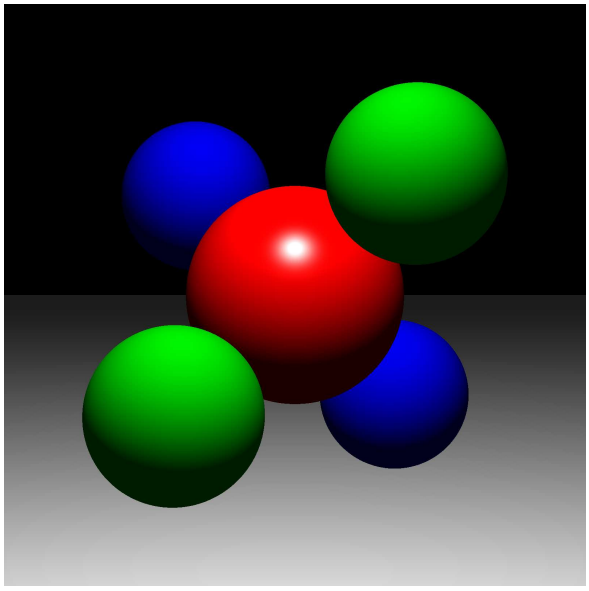
O cálculo de iluminação utiliza o modelo de reflexão de Phong, combinando as luzes da cena e uma cor ambiente global. Para cada fonte de luz, computa se a reflexão difusa e especular.

$$\begin{aligned} L &= \text{normalize}(\text{dirToLight}) \\ V &= \text{normalize}(-d) \\ R &= \text{normalize}(-L + 2(L \cdot N)N) \\ I_{difusa} &= \max(N \cdot L, 0) \cdot k_d \\ I_{especular} &= \max(V \cdot R, 0)^{\alpha} \cdot k_s \\ C_{final} &= I_a \cdot k_a + \sum_{i=1}^{N_{luzes}} (I_{difusa,i} + I_{especular,i}) \end{aligned}$$

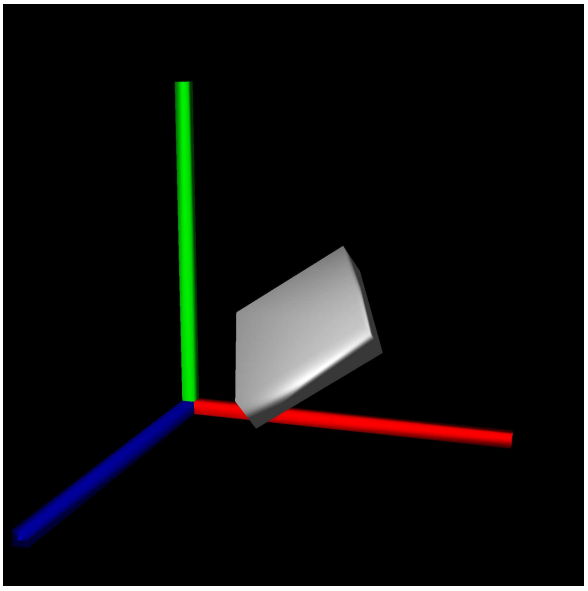
Onde:

- k_d representa a cor difusa do modelo. Quando tem textura, esse valor é substituído diretamente pela cor da textura;
- k_s é a cor especular, responsável pela cor da reflexão direta da luz na superfície;
- k_a é a cor ambiente da cena (no projeto $k_a = k_d$);
- α representa o expoente de brilho da superfície;
- I_a representa a constantes de intensidade da luz ambiente.

O resultado combinado da resolução de interseções geométricas e do modelo de iluminação na renderização da cena é ilustrado na Figura 20. (Código desenvolvido disponível em (OLIVEIRA, 2026f)).

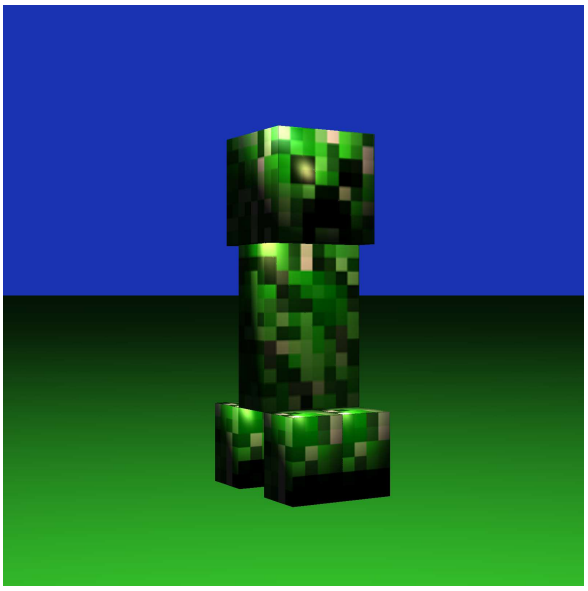


(a) Renderização de esferas com *Phong shading*.



(b) Renderização de objetos baseados em triângulos com transformações.

(c) Renderização de malha triangular complexa (coelho).



(d) Renderização com aplicação de texturas.

Figura 20 – Demonstração das capacidades do *ray caster* implementado.

4.9 Semana 8

Na oitava semana, a referência (UTAH, 2020) direciona o seu foco para os modelos de *shading*. Semelhante ao que foi abordado por outras referências em semanas anteriores, o curso introduz o conceito de BRDF, com foco principal na explicação do modelo Blinn Phong. Há uma forte ênfase no desenvolvimento da intuição do aluno, priorizando uma abordagem mais visual para a explicação do conteúdo. Além disso, o material detalha as diferenças conceituais entre *flat shading*, *smooth shading* e *phong shading*, dedicando também uma grande parte da segunda aula à explicação matemática sobre a correta aplicação de transformações geométricas em vetores normais.

A referência (MIT, 2020) dedica esta semana aos estudos de *antialiasing*, amostragem e iluminação global. A primeira aula oferece uma visão geral sobre a teoria de amostragem, estabelecendo a distinção técnica entre os processos de *sampling* e quantização. O material explora métodos para determinar a densidade ideal de amostragem e a utilização de transformadas de Fourier para a reconstrução de sinais e aproximação da imagem original. Além disso, são abordadas técnicas de *supersampling* e *jittered supersampling*, que consiste na adição de pequenas perturbações aleatórias na imagem para minimizar artefatos visuais.

A segunda aula do MIT avança para a iluminação global por meio da definição da Equação de Renderização (*The Rendering Equation*). Essa equação descreve matematicamente a aparência de uma cena ao contabilizar de forma simultânea a luz direta e a luz indireta refletida pelos objetos. Visto que a cálculo dessa equação é impraticável, o curso desenvolve metodologias de aproximação. O material demonstra passo a passo a evolução de um *ray tracer* tradicional para um algoritmo de *path tracing*, culminando na apresentação do pseudocódigo estrutural que define o funcionamento desse sistema.

Nessa semana os seguintes projetos foram desenvolvidos:

4.9.1 **UTAH**: Triangular Meshes

Este projeto tem como foco consolidar o conhecimento prático sobre superfícies formadas por malhas triangulares, aplicação de texturas e projeção geométrica. A implementação aborda diretamente como esses conceitos são estruturados e processados na GPU por meio da API gráfica WebGL.

A organização espacial da cena exige a construção matemática das transformações aplicadas aos vértices. Para isso, são definidas as matrizes individuais de translação (T) e de rotação nos eixos X (R_x) e Y (R_y):

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) & 0 \\ 0 & \sin(\theta_x) & \cos(\theta_x) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Onde t_x, t_y, t_z compõem o vetor de deslocamento e θ_x, θ_y são os ângulos de rotação fornecidos. A matriz de visualização (M_{view}) é obtida por meio da composição dessas transformações lineares:

$$M_{view} = T \cdot R_x \cdot R_y$$

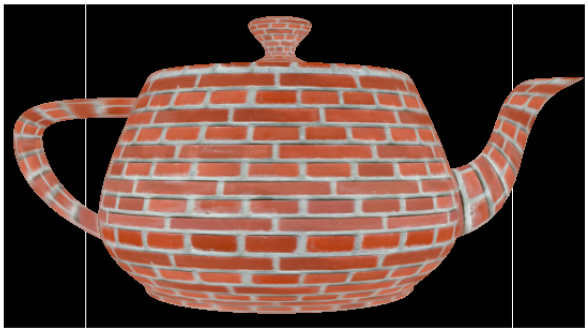
Como o modelo 3D da cena é estático, a matriz de modelo assume o valor da matriz identidade e é omitida do cálculo. Portanto, a transformação final que leva os vértices do espaço local para o espaço de projeção (MVP) é calculada exclusivamente pelo produto da matriz de projeção pela matriz de visualização estruturada acima:

$$M_{mvp} = M_{projection} \cdot M_{view}$$

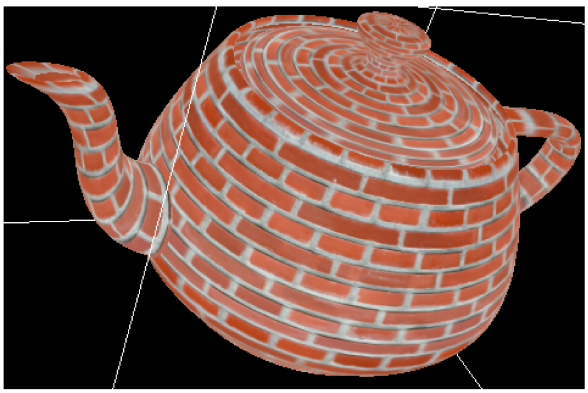
No que tange à utilização da API gráfica, o desenvolvimento exige o entendimento prático da alocação de recursos na memória da GPU. Isso abrange o envio dos vértices do modelo triangular, o processo de carregamento e amostragem de texturas e a passagem

correta das variáveis globais, conhecidas como *uniforms*, para a execução nos *shaders*. Os resultados visuais da aplicação dessas técnicas podem ser observados na Figura 21 (Código desenvolvido disponível em (OLIVEIRA, 2026k)).

(a) Renderização do modelo clássico do *teapot*.



(b) Renderização do *teapot* com a aplicação de texturas.



(c) Demonstração da movimentação da câmera na cena do *teapot* com textura.

(d) Renderização de malha triangular de personagem complexo com textura.

Figura 21 – Progressão da renderização de malhas: do *teapot* básico a modelos complexos texturizados.

4.9.2 **UTAH**: Shading

Este trabalho é uma continuação direta do projeto anterior, o utilizando como base (OLIVEIRA, 2026k) e tem como foco consolidar o conhecimento prático sobre BRDF, explorando especificamente o modelo de reflexão Blinn Phong. A prática consiste na modificação do projeto base, com as modificações concentradas no *vertex shader* e no *fragment shader*.

A direção da fonte de luz é definida de forma dinâmica na cena, assumindo que as coordenadas já estão no espaço de visualização (P_{light_view}). No *vertex shader*, as matrizes de transformação são aplicadas por vértice para calcular as posições e as normais relativas à câmera:

$$\begin{aligned} M_{modelView} &= M_{view} \cdot M_{model} \\ M_{normal} &= (M^{-1}_{modelView})^T \\ P_{view} &= M_{modelView} \cdot P_{vertex} \\ N_{view} &= M_{normal} \cdot N_{vertex} \end{aligned}$$

Esses vetores resultantes são interpolados pela GPU e repassados para o *fragment shader*. Como os cálculos ocorrem no espaço da câmera, a origem do sistema de coordenadas é a própria câmera, o que simplifica a obtenção do vetor de visualização. O cálculo final de iluminação por fragmento é definido pelas seguintes equações:

$$\begin{aligned} V &= \text{normalize}(-P_{view}) \\ L &= \text{normalize}(P_{light\_view} - P_{view}) \\ H &= \text{normalize}(L + V) \\ \cos(\theta) &= L \cdot N_{view} \\ \cos(\phi) &= N_{view} \cdot H \\ C_{frag} &= I \cdot (k_d \cdot \cos(\theta) + k_s \cdot \cos(\phi)^{\alpha}) + k_a \cdot I_{ambient} \end{aligned}$$

Onde:

k_d representa a cor difusa do modelo. Quando uma textura é carregada, esse valor é substituído diretamente pela cor da textura;

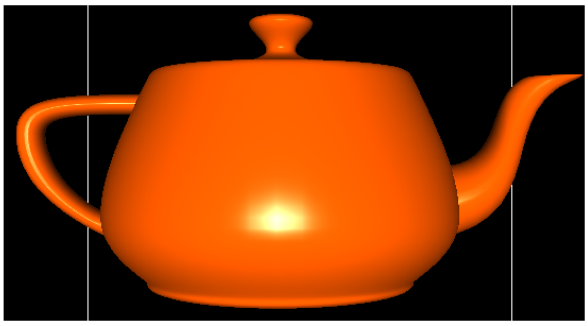
k_s é a cor especular, responsável pela cor da reflexão direta da luz na superfície;

k_a é a cor ambiente da cena (no projeto $k_a = k_d$);

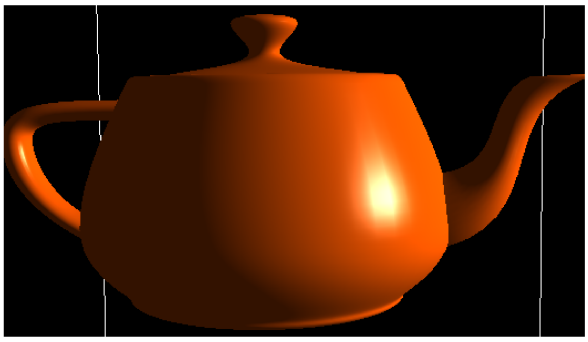
α representa o expoente de brilho da superfície;

I e $I_{ambient}$ representam as constantes de intensidade da luz direcional e da luz ambiente, assumindo os valores de $1,0$ e $0,2$, respectivamente.

Os resultados visuais da aplicação do modelo Blinn-Phong em diferentes malhas e sob várias perspectivas podem ser observados na Figura 22 (Código desenvolvido disponível em (OLIVEIRA, 2026i)).



(a) Renderização do *teapot* com o modelo Blinn-Phong.



(b) Movimentação da câmera evidenciando a reflexão especular.

(c) Renderização de malha complexa com aplicação de textura.

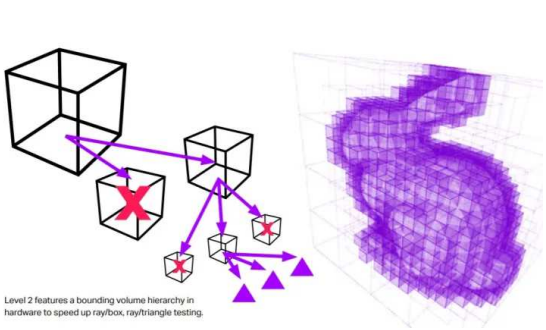
(d) Variação da perspectiva da câmera e interação da iluminação.

Figura 22 – Aplicação do modelo Blinn-Phong: do *teapot* básico a malhas texturizadas complexas.

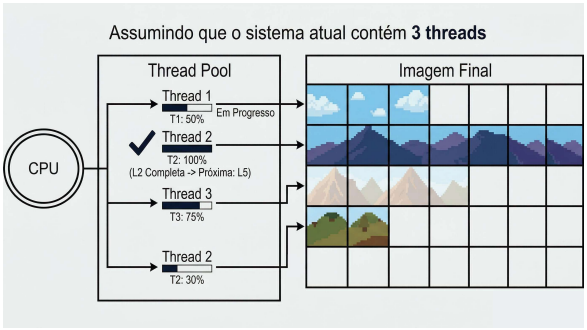
4.9.3 **MIT**: Ray tracing

Este projeto é uma continuação direta do projeto (OLIVEIRA, 2026f), evoluindo o desenvolvimento do renderizador para um *ray tracer*. A estrutura geral se mantém, ocorrendo uma melhora na otimização e a adição de efeitos secundários à renderização, como reflexão, refração e sombra.

Para melhorar a performance do renderizador, o foco recai sobre duas frentes: melhorar a eficiência do código e paralelizar a execução. Para tornar o código mais eficiente, é usada a estrutura de dados BVH (*Bounding Volume Hierarchy*), que permite que o traçado de raios caia de uma complexidade, em geometrias triangulares, de $O(n)$ para $O(\log n)$. Ela funciona como uma árvore tridimensional para otimizar os testes de busca de interseção. Para paralelizar a execução, o sistema identifica quantas *threads* a máquina executora do código possui e atribui uma linha da imagem para cada *thread* renderizar. A Figura 23 detalha essas duas abordagens, ilustrando a estrutura BVH e a divisão do processamento paralelo.



(a) Estrutura de árvore tridimensional *Bounding Volume Hierarchy* (BVH). (TECHNOLOGIES,)



(b) Diagrama de divisão do processamento paralelo.

Figura 23 – Técnicas de otimização implementadas para acelerar a renderização.

Em relação aos efeitos secundários, a implementação das sombras ocorre por meio do lançamento de raios de teste adicionais. Após o algoritmo computar a interseção primária entre o raio da câmera e a geometria, um novo raio é emitido a partir do ponto de impacto em direção a cada fonte de luz da cena. Caso esse raio colida com algum outro objeto antes de alcançar a fonte de iluminação, é validado que o ponto analisado está em uma região de sombra em relação a esta luz específica.

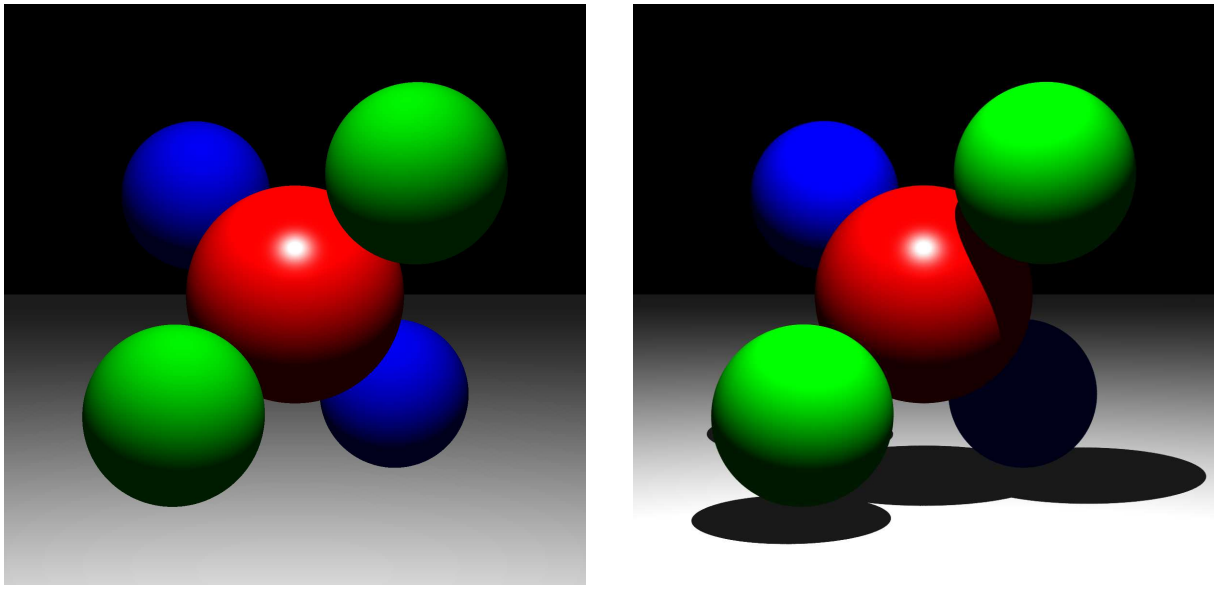
Assim, há uma leve alteração na formulação da iluminação do projeto anterior:

$$R_{sombra} = P + \epsilon \cdot L$$

$$V_i = \begin{cases} 1, & \text{se não houver interseção ou } t_{intersecao} > D_{luz} \\ 0, & \text{caso contrário} \end{cases}$$

$$C_{final} = k_a \cdot I_{ambient} + \sum_{i=1}^{N_{luzes}} V_i \cdot \left[I_i \cdot (k_d \cdot \cos(\theta_i) + k_s \cdot \cos(\phi_i)^{\alpha}) \right]$$

Onde P é o ponto de origem na superfície, L é a direção à luz normalizada, ϵ é um valor escalar mínimo para evitar autointerseção e D_{luz} é a distância total até a fonte. O comparativo de renderização evidenciando o impacto visual da aplicação desses raios de sombra (*shadow rays*) pode ser observado na Figura 24.



(a) Renderização de esferas com *Phong shading* sem raios de sombra.

(b) Renderização da mesma cena com a adição de raios de sombra.

Figura 24 – Comparativo de renderização evidenciando o impacto visual da aplicação de raios de sombra (*shadow rays*).

Para a implementação do efeito de reflexão, a lógica de renderização foi encapsulada em uma função recursiva, viabilizando a sua reutilização para o cálculo dos raios secundários. Essa função recebe como argumentos um raio (que inicialmente corresponde ao raio emitido pela câmera), um ponteiro para a estrutura da cena, a profundidade máxima de recursão e um controle booleano para a emissão de sombras. O retorno da função é a cor calculada para o raio processado.

Dessa forma, para gerar a cena inicial, a chamada da função é estruturada passando o raio primário:

```
RayTracer::traceRay(sceneParser.getCamera()->generateRay(Vector2f(x, -y)),
scene, BOUNCES, EMIT_SHADOWS);
```

Para adicionar os reflexos, é necessário traçar um novo raio a partir do ponto de impacto na direção da reflexão. O vetor de reflexão R é calculado com base na direção do raio incidente D e na normal da superfície N:

$$R = \text{normalize}(D - 2 \cdot (D \cdot N) \cdot N)$$
$$R_{reflexao} = \text{Ray}(P, R)$$

Com o novo raio criado a partir do ponto P, o algoritmo realiza uma chamada recursiva da função traceRay. A cor resultante dessa chamada secundária é então ponderada pela componente especular do material do objeto atingido:

$$C_{reflexao} = \text{traceRay}(R_{reflexao}, \text{scene}, \text{depth} - 1, \text{castShadows}) \cdot C_{especular}$$

A Figura 25 demonstra isoladamente o resultado visual desse efeito de reflexão aplicado à cena do vaso.

Figura 25 – Demonstração do efeito de reflexão na cena do vaso.

Semelhante à reflexão, para adicionarmos a refração utilizamos a recursividade da função traceRay para calcular o comportamento da luz ao atravessar geometrias translúcidas. Primeiro, identificamos se o raio de luz está entrando ou saindo do objeto. Para

isso, avaliamos se $N \cdot D > 0$. Nesse caso, o raio está no interior do objeto; portanto, invertemos o vetor normal e ajustamos o índice de refração do meio de destino, n_t, para 1.0, que assumimos ser o ar.

Com os índices de refração do meio de origem n e de destino n_t definidos, calculamos o radicando da Lei de Snell para verificar se ocorre o fenômeno de reflexão interna total:

$$\Delta = 1.0 - \frac{n^2 \cdot (1.0 - (N \cdot D)^2)}{n_t^2}$$

Caso $\Delta \geq 0$, a refração é válida. Computamos o vetor de direção da refração T, o que nos permite criar recursivamente o novo raio refratado:

$$T = \frac{n \cdot (D - N \cdot (N \cdot D))}{n_t} - N \cdot \sqrt{\Delta}$$
$$R_{refracao} = \text{Ray}(P + T \cdot \epsilon, T)$$

A cor obtida pela recursão do raio refratado é ponderada pela componente especular do material:

$$C_{refracao} = \text{traceRay}(R_{refracao}, \text{scene}, \text{depth} - 1, \text{castShadows}) \cdot C_{especular}$$

Por fim, para balancear a proporção de luz refletida e refratada, aplicamos a aproximação de Schlick para o efeito Fresnel.

$$R_0 = \left(\frac{n_t - n}{n_t + n}\right)^2$$
$$R_{contrib} = R_0 + (1.0 - R_0) \cdot (1.0 - c)^5$$

Onde c assume o valor de $|N \cdot D|$ se a luz passa para um meio mais denso ($n \leq n_t$), ou $|T \cdot N|$ caso contrário. Ponderando as cores de refração e reflexão, temos:

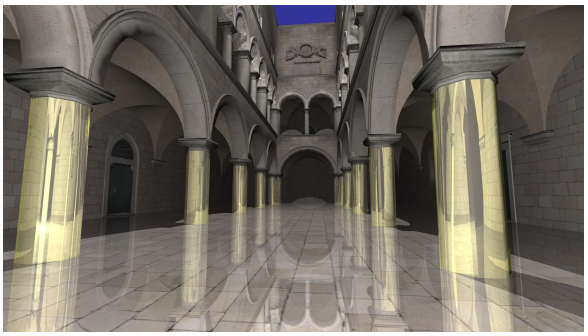
$$C_{refracao} = \text{traceRay}(R_{refracao}, \text{scene}, \text{depth} - 1, \text{castShadows}) \cdot (1.0 - R_{contrib}) \cdot C_{especular}$$

$$C_{reflexao} = \text{traceRay}(R_{reflexao}, \text{scene}, \text{depth} - 1, \text{castShadows}) \cdot R_{contrib} \cdot C_{especular}$$

Os resultados finais do *ray tracer*, integrando todos os efeitos geométricos implementados de sombras, reflexão e refração, estão consolidados na Figura 26 (Código desenvolvido disponível em (OLIVEIRA, 2026g)).

(a) Cena final integrando efeitos de sombras, reflexão e refração.

(b) Renderização da cena *Sibenik*.



(c) Renderização da cena *Sponza*.

Figura 26 – Resultados finais do *ray tracer*.

4.10 Considerações finais

Nesta seção é apresentado um resumo dos projetos descritos neste capítulo. Para cada um é feita uma avaliação do nível de dificuldade e um comparativo com a ementa de computação gráfica da UFU.

A Tabela 3 mostra um resumo dos projetos desenvolvidos. É feita uma análise da dificuldade de execução do projeto.

Tabela 3 – Comparativo dos projetos desenvolvidos.

Projeto	**Instituição**	**Conteúdos**	**Descrição**	**Dificuldade**
Compositing Images	UTAH	*Alpha blending*, manipulação de *buffers*, hierarquia 2D.	Implementação de manipulação de *buffers* de cor e composição de camadas com *alpha blending*. Foca no entendimento de transparência e ordem de renderização.	Fácil
OpenGL Mesh Viewer	MIT	*Parsing* de OBJ, *pipeline* de renderização com OpenGL legado e cálculo de normais.	Desenvolvimento *parser* de arquivos OBJ e processamento de normais baseado em cada face de triângulos. Foca na transição de dados para a GPU e iluminação básica.	Fácil
Curves and Surfaces	MIT	Curvas de Bézier, *B-Splines* e superfícies de revolução.	Implementação algorítmica de avaliação de curvas paramétricas (Bézier e *Splines*). Envolve o estudo de continuidade geométrica e interpolação de superfícies.	Muito difícil
Transformations	UTAH	Tansformações, matrizes homogêneas, ecoordenadas 2D.	Aplicação prática de álgebra linear para manipulação de cena. Foca no domínio da composição de matrizes homogêneas e transformações de coordenadas.	Fácil
Hierarchical Modeling & SSD	MIT	*Forward kinematics*, *skinning* (SSD) e árvore de cena.	Implementação de animação baseada em esqueleto (*skinning*). Foca no cálculo de matrizes relativas e na deformação ponderada de malhas para personagens.	Difícil
DrawSVG	CMU	Rasterização, coordenadas baricêntricas, *anti-aliasing* e *mipmapping*.	Construção de um *pipeline* de rasterização em *software*. Foca em *anti-aliasing* via *supersampling* e em filtros de textura.	Médio
Intro	UBC	*Shaders* GLSL (*vertex*/*fragment*) e *pipeline* básico utilizando Threejs.	Desenvolvimento de *shaders* iniciais em GLSL e integração com Three.js. Foca no entendimento básico dos estágios de renderização na GPU.	Fácil
Physically-based simulation	MIT	Integração numérica (Euler/*Trapezoidal*), EDOs e malhas deformáveis.	Implementação de simuladores dinâmicos baseados em física. Foca na modelagem de forças físicas (molas e gravidade).	Médio

Tabela 3 – Continuação da página anterior

Projeto	**Instituição**	**Conteúdos**	**Descrição**	**Dificuldade**
Curves	UTAH	Curvas de Bézier e avaliação paramétrica de Bézier na GPU.	Migração do cálculo geométrico de curvas para a GPU. Foca na avaliação de equações paramétricas diretamente no *vertex shader* para ganho de *performance*.	Médio
Transformations	UBC	*Forward kinematics* e transformações em hierarquias articuladas.	Desenvolvimento de animações procedurais. Foca no controle de movimentos articulados e utilizando os conceitos de *foward kinematics*.	Fácil
Ray casting	MIT	Interseção de primitivas, coordenadas baricêntricas e geração de raios.	Implementação matemática das interseções raio com geometria básicas. Foca na renderização por *pixel* de uma cena através do traçado de raios.	Difícil
Triangular Meshes	UTAH	Transformações MVP, texturização de malhas e uso de WebGL.	Renderização em tempo real de malhas texturizadas. Foca no processamento do sistema *Model-View-Projection* (MVP) e no gerenciamento de memória na GPU.	Médio
Shading	UTAH	Modelo Blinn-Phong e BRDF.	Implementação do modelo de iluminação Blinn-Phong. Foca no cálculo das componentes difusa e especular calculado no espaço da câmera via *shaders*.	Médio
Ray tracing	MIT	Sombras, reflexão/refração, BVH e paralelização *multi-thread*.	Expansão do renderizador para suportar efeitos de iluminação global. Foca na simulação física da luz e na otimização arquitetural via particionamento espacial (BVH) e *threads*.	Muito difícil

A Tabela 4 apresenta uma análise comparativa dos conteúdos abordados nas referências em comparação com a UFU, considerando os conteúdos presentes na Ficha de Componente Curricular (FACOM, 2023a) para a matéria GBC204 - Computação Gráfica.

Tabela 4 – Comparativo dos conteúdos desenvolvidos.

Matéria abordada na UFU	**Referências que também abordam**
Conceito de Computação Gráfica	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)

Tabela 4 – Continuação da página anterior

Matéria abordada na UFU	**Referências que também abordam**
Histórico e Aplicações	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Equipamentos para Computação Gráfica	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Equipamentos de entrada e saída	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Geração de linhas	(CMU, 2020)
Geração de circunferências	(OLIVEIRA, 2026b; MIT, 2020; UTAH, 2020)
Preenchimento de polígonos	(OLIVEIRA, 2026b; MIT, 2020; COLUMBIA, 2020)
Algoritmo de Cohen-Sutherland	
Algoritmo de ponto médio	
Recorte de polígonos	(COLUMBIA, 2020; CMU, 2020; MIT, 2020)
Transformações em 2D	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Coordenadas homogêneas	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Escala, translação e rotação	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Matriz de transformação geométrica	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Transformação em 3D	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Câmera sintética e passos na visualização 3D	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Projeções perspectivas e paralelas	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Modelos poliedrais e malhas de polígonos.	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Modelagem de sólidos	(CMU, 2020)
Sólidos R-set	
Esquemas de representação de sólidos	(MIT, 2020; UTAH, 2020; CMU, 2020)
Diagrama cromático CIE	(CMU, 2020; MIT, 2020).

Tabela 4 – Continuação da página anterior

Matéria abordada na UFU	**Referências que também abordam**
Modelos de cor: RGB, CMY, HSV	(MIT, 2020; UTAH, 2020; CMU, 2020)
Algoritmo de Depth-Buffer	(MIT, 2020; UTAH, 2020; CMU, 2020)
Algoritmo Z-Buffer	(MIT, 2020; UTAH, 2020; CMU, 2020)
Algoritmo Scan-Line	(COLUMBIA, 2020)
Reflexão difusa e luz ambiente	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Reflexão especular	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Modelo de Phong	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020)
Múltiplas fontes de luz	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Constante (Flat Shading)	(MIT, 2020; COLUMBIA, 2020)
Interpolado	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Gouraud	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020)
Phong	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020)
Aplicação de Texturas e sombras	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)
Ray tracing, radiosity e modelos híbridos.	(MIT, 2020; UTAH, 2020; COLUMBIA, 2020; CMU, 2020)

5 Conclusão

O objetivo central deste trabalho foi avaliar metodologias de ensino de Computação Gráfica no ensino superior, identificando os tópicos, ferramentas e abordagens predominantes em cursos de referência mundial.

Foram analisados quatro cursos: (MIT, 2020), (UTAH, 2020), (CMU, 2020) e (COLUMBIA, 2020). A metodologia incluiu a implementação prática de 14 projetos distribuídos ao longo de oito semanas, viabilizando uma avaliação direta da dificuldade técnica e da progressão de complexidade de cada instituição.

Tecnologicamente, os cursos se dividem em dois grupos. O primeiro adota C++ e OpenGL (MIT e CMU), com foco em controle de mais baixo nível. O segundo utiliza JavaScript com WebGL ou Three.js (UTAH e UBC). A primeira abordagem aprofunda o entendimento da arquitetura do pipeline gráfico, enquanto a segunda minimiza a barreira técnica inicial, permitindo que o aluno foque na matemática e nos conceitos.

Quanto ao conteúdo, os cursos convergem para um núcleo comum: transformações geométricas, rasterização, malhas triangulares, shading e ray tracing. Esse resultado se alinha ao mapeamento detalhado em (BALREIRA; WALTER; FELLNER, 2017), que aponta a renderização como tema dominante. A principal diferença entre as instituições reside na ordem, na profundidade e no rigor matemático aplicados a esses tópicos.

A análise de dificuldade revelou perfis distintos. Os projetos do MIT são os mais exigentes matematicamente, cobrindo superfícies de revolução, simulação física com EDOs e ray tracing com estrutura BVH. UTAH se destaca por uma progressão mais gradual, aumentando progressivamente a complexidade e a dificuldade de cada projeto; CMU aprofunda os fundamentos algorítmicos da rasterização por meio do projeto DrawSVG, capacitando o aluno a implementar um pipeline gráfico completo operando estritamente em software (CPU); e UBC oferece a entrada mais acessível via Three.js, apostando em projetos lúdicos e de alto apelo visual que engajam o aluno pelo resultado imediato, servindo como uma excelente porta de entrada para a área. Conclui-se que não existe uma metodologia superior absoluta, mas sim diferentes balanceamentos entre rigor técnico, escopo e acessibilidade.

Embora as evidências confirmem a inexistência de uma metodologia universalmente superior, a vivência prática ao longo desta pesquisa permite delinear um percurso de aprendizado considerado ideal sob a ótica de um discente no oitavo período de Sistemas de Informação. Para esse perfil de aluno, que já possui base algorítmica, mas pode se beneficiar de uma curva de aprendizado mais engajadora, um modelo pedagógico híbrido apresenta-se como o mais eficaz.

Observa-se que projetos iniciais de alto apelo estético, a exemplo de (OLIVEIRA, 2026d), são altamente benéficos para capturar rapidamente o interesse pela disciplina. Em contrapartida, desenvolvimentos como (OLIVEIRA, 2026b), embora visualmente mais rudimentares, revelam-se indispensáveis para a consolidação dos fundamentos de renderização. O equilíbrio ideal materializa-se em implementações como (OLIVEIRA, 2026g), que conseguem conciliar um engajamento visual de alto nível com uma densa carga de conteúdos essenciais.

Nesse contexto, nota-se o grande valor de adotar, desde o primeiro projeto, um ambiente de programação de alto controle e baixo nível. Essa base seria formada por C++ e OpenGL, preferencialmente com o apoio de bibliotecas como o GLFW para gerenciar o loop de renderização. Para reduzir a difi-

culdade inicial com essas ferramentas, a disciplina utilizaria a entrega de um código inicial estruturado. Assim, nas primeiras etapas, a complexidade tecnológica fica isolada pela infraestrutura já fornecida pelo curso, permitindo que o aluno foque exclusivamente na matemática e nos conceitos. Na progressão ideal da matéria, a quantidade de código entregue pronta diminui a cada novo projeto. Conforme o aluno consolida o seu aprendizado, a responsabilidade de programar a arquitetura do sistema passa gradualmente para ele. É nesse momento que o rigor e o controle direto sobre o hardware, oferecidos pelo C++ e OpenGL, tornam-se essenciais e são compreendidos pelo estudante.

Como limitação, ressalta-se que a análise utilizou versões específicas dos cursos (de 2020) e que as instituições selecionadas não representam a totalidade do ensino de Computação Gráfica. Trabalhos futuros podem expandir o escopo para outras universidades, coletar métricas de alunos e professores, e avaliar a aplicação direta dessas abordagens no currículo local da FACOM/UFU.

Em resumo, o trabalho fornece um mapeamento técnico e prático das metodologias de ensino adotadas por instituições de referência na área. Os resultados geram uma base documentada para orientar a reestruturação curricular de disciplinas de Computação Gráfica, servindo como material de apoio prático para coordenadores e docentes da área.

Referências

AKENINE-MöLLER, T.; HAINES, E.; HOFFMAN, N. **Real-Time Rendering**. 3rd. ed. USA: A. K. Peters, Ltd., 2008. ISBN 1568814240. Citado na página 16.

_____. **Real-Time Rendering, Fourth Edition**. 4th. ed. USA: A. K. Peters, Ltd., 2018. ISBN 0134997832. Citado na página 16.

ANGEL, E.; SHREINER, D. The future of teaching computer graphics. In: **ACM SIGGRAPH 2024 Educator's Forum**. New York, NY, USA: Association for Computing Machinery, 2024. (SIGGRAPH '24). ISBN 9798400705175. Disponível em: <https://doi.org/10.1145/3641235.3664433>. Citado na página 11.

BALREIRA, D. G.; WALTER, M.; FELLNER, D. W. What we are teaching in introduction to computer graphics. In: **Proceedings of the European Association for Computer Graphics: Education Papers**. Goslar, DEU: Eurographics Association, 2017. (EG '17), p. 1–7. Disponível em: <https://doi.org/10.2312/eged.20171019>. Citado 3 vezes nas páginas 13, 14 e 61.

BUSS, S. R. **3D Computer Graphics: A Mathematical Introduction with OpenGL**. USA: Cambridge University Press, 2003. ISBN 0521821037. Citado na página 16.

CMU. **15-462/662 Computer Graphics, Fall 2020**. 2020. Disponível em: <https://15462.courses.cs.cmu.edu/fall2020/>. Citado 14 vezes nas páginas 9, 10, 16, 17, 18, 21, 26, 32, 40, 43, 58, 59, 60 e 61.

COLUMBIA, U. of B. **CS 314 - Course Materials, Sep 2020**. 2020. Disponível em: <https://www.students.cs.ubc.ca/~cs-314/Vsep2020/>. Citado 14 vezes nas páginas 9, 10, 16, 17, 18, 21, 26, 32, 40, 43, 58, 59, 60 e 61.

DUNN, F.; PARBERRY, I. **3D math primer for graphics and game development, 2nd edition**. [S.l.: s.n.], 2011. 1-830 p. Citado na página 16.

ELYAN, E. Enhanced interactivity and engagement: Learning by doing to simplify mathematical concepts in computer graphics and animation. In: **Proceedings of the 2012 IEEE Global Engineering Education Conference (EDUCON)**. [S.l.: s.n.], 2012. p. 1–8. Citado na página 13.

FACOM. **Ficha de disciplina Computação Gráfica**. 2012. Disponível em: <https://facom.ufu.br/system/files/conteudo/gbc204-computacao-grafica.pdf>. Citado na página 9.

_____. **Ficha de disciplina Computação Gráfica**. 2023. Disponível em: <https://facom.ufu.br/system/files/conteudo/sei_5129012_ficha_de_componente_curricular_gbc204.pdf>. Citado 2 vezes nas páginas 9 e 58.

_____. **Projeto Pedagógico do Curso de Graduação em Ciência da Computação**. 2023. Disponível em: <https://facom.ufu.br/system/files/conteudo/ppc_2023-1_final_correcao_horas_-_tabela_2.pdf>. Citado na página 9.

GÁLVEZ, A.; IGLESIAS, A.; CORCUERA, P. An introductory computer graphics course in the context of the european space of higher education: A curricular approach. In: BUBAK, M.; ALBADA, G. D. van; DONGARRA, J.; SLOOT, P. M. A. (Ed.). **Computational Science – ICCS 2008**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 715–724. ISBN 978-3-540-69387-1. Citado na página 13.

GORTLER, S. J. **Foundations of 3D Computer Graphics**. [S.l.]: The MIT Press, 2012. ISBN 0262017350. Citado na página 16.

HITCHNER, L.; SOWIZRAL, H. Adapting computer graphics curricula to changes in graphics. **Computers & Graphics**, v. 24, p. 283–288, 04 2000. Citado na página 13.

HUGHES, J. F.; DAM, A. van; MCGUIRE, M.; SKLAR, D. F.; FOLEY, J. D.; FEINER, S. K.; AKELEY, K. **Computer Graphics - Principles and Practice, 3rd Edition**. [S.l.]: Addison-Wesley, 2014. ISBN 978-0-321-39952-6. Citado na página 16.

HUI, P.; LIU, S.; XIU, J.; ZHAI, R. Ability-oriented teaching reform for computer graphics. In: **Proceedings of the 7th International Conference on Computer Science & Education (ICCSE)**. Melbourne, Australia: [s.n.], 2012. p. 1908–1911. Citado na página 13.

KRULL, F. The origin of computer graphics within general motors. **IEEE Annals of the History of Computing**, v. 16, n. 3, p. 40–, 1994. Citado na página 11.

LOWTHER, J.; SHENE, C.-K. Rendering + modeling + animation + postprocessing = computer graphics. **Journal of Computing Sciences in Colleges**, v. 16, p. 20–28, 11 2000. Citado na página 13.

MARSCHNER, S.; SHIRLEY, P. **Fundamentals of Computer Graphics, Fourth Edition**. 4th. ed. USA: A. K. Peters, Ltd., 2016. ISBN 1482229390. Citado na página 16.

MARTí, E.; GIL, D.; JULIà, C. A pbl experience in the teaching of computer graphics. **Computer Graphics Forum**, v. 25, n. 1, p. 95–103, 2006. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2006.00920.x>. Citado na página 13.

MATSUDA, K.; LEA, R. **WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL**. Addison-Wesley, 2013. (Always learning). ISBN 9780321902924. Disponível em: <https://books.google.com.br/books?id=6V5sAQAAQBAJ>. Citado na página 16.

MIT. **6.837 Computer Graphics, Fall 2012**. 2012. Disponível em: <https://ocw.mit.edu/courses/6-837-computer-graphics-fall-2012/>. Citado 3 vezes nas páginas 10, 16 e 17.

_____. **6.837 Computer Graphics, Fall 2020 (YouTube playlist)**. 2020. Disponível em: <https://youtu.be/-LqUu61oRdk?list=PLQ3UicqQtfNuBjzJ-KEWmG1yjiRMXYKhh>. Citado 15 vezes nas páginas 9, 10, 16, 17, 18, 21, 26, 32, 40, 43, 47, 58, 59, 60 e 61.

MORAN, A. **Curves and Surfaces**. 2012. <https://github.com/andrewmo2014/Curves-and-Surfaces/blob/master/surf.cpp>. Citado 2 vezes nas páginas 22 e 23.

NVIDIA. **Graphics Reinvented: New Technologies in RTX Graphics Cards**. 2018. Disponível em: <https://www.nvidia.com/en-us/geforce/news/graphics-reinvented-new-technologies-in-rtx-graphics-cards/#dlss>. Citado na página 11.

_____. **Introducing NVIDIA DLSS 3**. 2022. Disponível em: <https://nvidianews.nvidia.com/news/nvidia-introduces-dlss-3-with-breakthrough-ai-powered-frame-generation-for-up-to-4x-performance>. Citado na página 11.

OKANO, W. J. d. S. **Implementação de uma biblioteca gráfica multiplataforma utilizando OpenGL e GLFW**. 2018. Trabalho de Conclusão de Curso (Graduação em Sistemas de Informação) – Universidade Federal de Uberlândia, Uberlândia. 39 f. Citado na página 9.

OLIVEIRA, A. C. **Curves and Surface: Project 1**. 2025. <https://github.com/Arthur020104/MIT6.837/tree/main/one>. Citado na página 22.

_____. **Image Processing: Project 0**. 2025. <https://github.com/Arthur020104/UtahCs4600/tree/master/p1>. Citado na página 19.

_____. **Mesh Viewer: Project 0**. 2025. <https://github.com/Arthur020104/MIT6.837/tree/main/zero>. Citado na página 20.

_____. **Transformations: Project 1**. 2025. <https://github.com/Arthur020104/UtahCs4600/tree/master/p2>. Citado na página 24.

_____. **Curves: Project 2**. 2026. <https://github.com/Arthur020104/UtahCs4600/tree/master/p3>. Citado na página 41.

_____. **DrawSVG: Project 0**. 2026. <https://github.com/Arthur020104/DrawSVG>. Citado 4 vezes nas páginas 28, 35, 59 e 61.

_____. **Hierarchical Modeling & SSD: Project 2**. 2026. <https://github.com/Arthur020104/MIT6.837/tree/main/two>. Citado na página 27.

_____. **Intro: Project 0**. 2026. <https://github.com/Arthur020104/UbcCG/tree/master/Intro>. Citado 2 vezes nas páginas 31 e 61.

_____. **Physically-based simulation: Project 3**. 2026. <https://github.com/Arthur020104/MIT6.837/tree/main/three>. Citado na página 34.

_____. **Ray casting: Project 4**. 2026. <https://github.com/Arthur020104/MIT6.837/tree/main/four>. Citado 2 vezes nas páginas 45 e 52.

_____. **Ray tracing: Project 5**. 2026. <https://github.com/Arthur020104/MIT6.837/tree/main/five>. Citado 2 vezes nas páginas 55 e 61.

_____. **Repositório Geral dos Projetos do Trabalho de Conclusão de Curso**. 2026. <https://github.com/Arthur020104/Tcc>. Citado na página 15.

_____. **Shading: Project 4**. 2026. <https://github.com/Arthur020104/UtahCs4600/tree/master/p5>. Citado na página 51.

_____. **Transformations: Project 1**. 2026. <https://github.com/Arthur020104/UbcCG/tree/master/Transformations>. Citado na página 42.

_____. **Triangular Meshes: Project 3**. 2026. <https://github.com/Arthur020104/UtahCs4600/tree/master/p4>. Citado 2 vezes nas páginas 49 e 50.

PAPAGIANNAKIS, G.; PAPANIKOLAOU, P.; GREASSIDOU, E.; TRAHANIAS, P. glGA: an OpenGL Geometric Application Framework for a Modern, Shader-based Computer Graphics Curriculum. In: BOURDIN, J.-J.; JORGE, J.; ANDERSON, E. (Ed.). **Eurographics 2014 - Education Papers**. [S.l.]: The Eurographics Association, 2014. ISSN 1017-4656. Citado na página 13.

PETERNIER, A.; THALMANN, D.; VEXO, F. Mental vision: A computer graphics teaching platform. In: PAN, Z.; AYLETT, R.; DIENER, H.; JIN, X.; GÖBEL, S.; LI, L. (Ed.). **Technologies for E-Learning and Digital Entertainment**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 223–232. ISBN 978-3-540-33424-8. Citado na página 13.

PHARR, M.; HUMPHREYS, G. **Physically Based Rendering, Second Edition: From Theory To Implementation**. 2nd. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN 0123750792. Citado na página 16.

RODRIGUES, R.; MATOS, T.; Valle de Carvalho, A.; BARBOSA, J. G.; ASSAF, R.; NóBREGA, R.; COELHO, A.; de Sousa, A. A. Computer graphics teaching challenges: Guidelines for balancing depth, complexity and mentoring in a confinement context. **Graphics and Visual Computing**, v. 4, p. 200021, 2021. ISSN 2666-6294. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2666629421000048>. Citado na página 12.

SCOTTY3D. 2020. <https://github.com/CMU-Graphics/Scotty3D>. CMU Graphics. Citado 2 vezes nas páginas 9 e 16.

SEGAL, M.; AKELEY, K.; FRAZER, C. The opengl™ graphics system: A specification. version 1.0 – 1 july 1994. **Silicon Graphics, Inc. Technical Documentation**, Jul 1994. Copyright © 1992–1994 Silicon Graphics, Inc. Disponível em: <https://registry.khronos.org/OpenGL/specs/gl/glspec10.pdf>. Citado na página 11.

SHIRLEY, P.; MARSCHNER, S. **Fundamentals of Computer Graphics**. 3rd. ed. USA: A. K. Peters, Ltd., 2009. ISBN 1568814690. Citado na página 16.

STEPHENSON, B.; TAUBE-SCHOCK, C. Quickdraw: bringing graphics into first year. In: **Proceedings of the 40th ACM Technical Symposium on Computer Science Education**. New York, NY, USA: Association for Computing Machinery, 2009. (SIGCSE '09), p. 211–215. ISBN 9781605581835. Disponível em: <https://doi.org/10.1145/1508865.1508946>. Citado na página 13.

SUNG, K.; SHIRLEY, P. A top-down approach to teaching introductory computer graphics. **Computers & Graphics**, v. 28, 2003. Citado 3 vezes nas páginas 12, 13 e 14.

SUSELO, T.; WüNSCHE, B.; LUXTON-REILLY, A. The journey to improve teaching computer graphics: A systematic review. In: . [S.l.: s.n.], 2017. Citado 4 vezes nas páginas 10, 12, 13 e 14.

SUTHERLAND, I. E. **Sketchpad: A man-machine graphical communication system**. Tese (Doutorado) — Massachusetts Institute of Technology, 1963. Disponível em: <https://dspace.mit.edu/handle/1721.1/14979>. Citado na página 11.

TALTON, J. O.; FITZPATRICK, D. Teaching graphics with the opengl shading language. In: **Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education**. New York, NY, USA: Association for Computing Machinery, 2007. (SIGCSE '07), p. 259–263. ISBN 1595933611. Disponível em: <https://doi.org/10.1145/1227310.1227402>. Citado na página 13.

TECHNOLOGIES, I. <https://www.embedded.com/wp-content/uploads/sites/2/2020/09/Imagination-Ray-Tracing-level-2.jpg?resize=768,393>. Accessed: 2026-04-05. Citado na página 52.

UTAH, U. of. **CS4600 Computer Graphics, Fall 2020**. 2020. Disponível em: <https://graphics.cs.utah.edu/courses/cs4600/fall2020/>. Citado 16 vezes nas páginas 9, 10, 16, 17, 18, 21, 24, 26, 32, 40, 43, 47, 58, 59, 60 e 61.

WATT, A. **3d Computer Graphics**. 2nd. ed. USA: Addison-Wesley Longman Publishing Co., Inc., 1993. ISBN 0201631865. Citado na página 16.

WOLFE, R.; LOWTHER, J. L.; SHENE, C.-K. Rendering + modeling + animation + postprocessing = computer graphics. **SIGGRAPH Comput. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 34, n. 4, p. 15–18, nov. 2000. ISSN 0097-8930. Disponível em: <https://doi.org/10.1145/369215.369224>. Citado na página 11.

ZHOU, J.; YE, M.; HUANG, C.-L. Reform of computer graphics teaching method. In: **2010 International Forum on Information Technology and Applications**. [S.l.: s.n.], 2010. v. 3, p. 222–225. Citado na página 13.

ZORZO, A. F.; NUNES, D.; MATOS, E.; STEINMACHER, I.; LEITE, J.; ARAUJO, R. M.; CORREIA, R.; MARTINS, S. **Referenciais de Formação para os Cursos de Graduação em Computação**. [S.l.]: Sociedade Brasileira de Computação (SBC), 2017. 153 p. Citado na página 9.