Relatório do Trabalho 2

Computação Gráfica 3D com Iluminação

Disciplina: SCC 250 – Computação Gráfica
Instituto: ICMC – USP
Professora: Agma

Data de Entrega: 26/11/2025

1 Identificação dos Alunos e Participação

Aluno 1: Lucas Greff Meneses - 13671615

As responsabilidades incluíram a implementação dos modelos de geometria 3D (cubo, pirâmide, cone e esfera), o desenvolvimento da estrutura de classes base `Vector3D` e `Geometry3D`, a implementação dos modelos de sombreamento Flat e Gouraud.

Aluno 2: Vitor da Silveira - 10689651

As responsabilidades concentraram-se na implementação do modelo Phong Shading utilizando shaders GLSL, no desenvolvimento da interface gráfica com PyQt6, na implementação do sistema de câmera orbital e na integração de todos os componentes do sistema em uma aplicação coesa e funcional.

2 Objetivo do Trabalho

O objetivo deste trabalho é desenvolver um sistema interativo de visualização 3D que permita manipular objetos tridimensionais por meio de transformações geométricas (rotação, escala e translação), alternar entre projeções perspectiva e ortográfica, aplicar três modelos de iluminação distintos (Flat, Gouraud e Phong), comparar visualmente esses três modelos de iluminação lado a lado e controlar de forma interativa diversos parâmetros da cena, da fonte de luz e dos próprios objetos. O trabalho busca demonstrar na prática os conceitos teóricos de computação gráfica 3D, com ênfase nos diferentes algoritmos de iluminação e sombreamento.

3 Arquitetura do Sistema

3.1 Estrutura Modular

O sistema foi desenvolvido seguindo princípios de Programação Orientada a Objetos e organizado em módulos independentes, agrupados em diretórios lógicos. O arquivo `main.py` atua como ponto de entrada da aplicação. O diretório `gui` contém a interface gráfica, incluindo a janela principal e os controles (`main_window.py`) e o widget responsável

pela renderização em OpenGL (`opengl_widget.py`). No diretório `core` estão concentrados os componentes fundamentais, como a implementação de operações vetoriais em `vector3d.py`, a representação da fonte de luz em `light.py`, as propriedades de material em `material.py`, a câmera orbital em `camera.py` e o gerenciador de cena em `scene.py`. O diretório `shading` agrupa os modelos de iluminação, com uma classe base abstrata em `shading_model.py` e as implementações específicas de Flat Shading, Gouraud Shading e Phong Shading (GLSL) em arquivos separados. Por fim, o diretório `geometry` contém os objetos geométricos, com uma classe base abstrata em `geometry3d.py` e as classes concretas para cubo, pirâmide, cone e esfera.

3.2 Tecnologias Utilizadas

A aplicação foi desenvolvida em Python 3.8 ou superior, que serve como linguagem principal do projeto. A interface gráfica foi construída com o framework PyQt6, que oferece widgets modernos e integração adequada com OpenGL. Para o acesso às funcionalidades gráficas de baixo nível utilizou-se o PyOpenGL, que fornece *bindings* em Python para a API OpenGL. As operações matriciais e vetoriais, especialmente nos cálculos relacionados a transformações e normais, foram implementadas com apoio da biblioteca NumPy. Para o modelo de iluminação Phong foram desenvolvidos shaders em GLSL (OpenGL Shading Language), permitindo o cálculo da iluminação diretamente na GPU.

3.3 Padrões de Projeto Aplicados

Na arquitetura de software, alguns padrões de projeto foram aplicados para tornar o sistema mais flexível e extensível. O padrão *Strategy* foi utilizado na implementação dos modelos de iluminação, permitindo trocar dinamicamente o algoritmo de sombreamento utilizado em tempo de execução. O padrão *Template Method* está presente na classe base `ShadingModel`, que define uma interface comum para os diferentes modelos de iluminação, deixando os detalhes específicos para as subclasses. O padrão *Factory* foi aplicado na criação dos objetos geométricos na classe `Scene3D`, facilitando a instanciação de diferentes formas 3D a partir de uma interface unificada. Além disso, o sistema de sinais e slots do Qt funciona, na prática, como uma forma de *Observer Pattern*, permitindo que alterações na interface gráfica sejam propagadas para o restante da aplicação de forma reativa.

4 Implementação dos Componentes

4.1 Classes Fundamentais

4.1.1 `Vector3D` (`core/vector3d.py`)

A classe `Vector3D` é responsável pela representação e manipulação de vetores em três dimensões. Ela oferece operações fundamentais para computação gráfica, incluindo normalização de vetores, cálculo de produto escalar (*dot product*), cálculo de produto vetorial (*cross product*) e operações de soma e subtração entre vetores. Essa classe é essencial para os cálculos de iluminação, principalmente na determinação de normais de superfície.

Um exemplo típico de uso ocorre no cálculo de normais de uma face, em que dois vetores de aresta são combinados por meio do produto vetorial e o resultado é normalizado:

```
1 normal = v1.cross(v2).normalize()
```

4.1.2 `Light` (`core/light.py`)

A classe `Light` representa uma fonte de luz pontual na cena. Ela armazena a posição da luz em coordenadas 3D e define três componentes principais: a componente ambiente, que modela uma iluminação de base uniforme na cena; a componente difusa, que depende da direção da luz em relação à superfície; e a componente especular, responsável pelos reflexos brilhantes observados em materiais polidos. No sistema implementado, a componente ambiente foi configurada com o vetor $[0.3, 0.3, 0.3]$, representando 30% de intensidade. A componente difusa utiliza o vetor $[0.8, 0.8, 0.8]$, correspondente a 80% de intensidade, enquanto a componente especular é configurada como $[1.0, 1.0, 1.0]$, representando 100% de intensidade.

4.1.3 `Material` (`core/material.py`)

A classe `Material` define como a superfície de um objeto interage com a luz. Ela contém parâmetros para a reflexão ambiente, que determina quanto da luz ambiente é refletida, para a reflexão difusa, que controla a resposta do material à iluminação direcional, e para a reflexão especular, que regula a intensidade dos brilhos especulares. Além disso, a classe possui o parâmetro *shininess*, que controla a concentração do brilho especular; valores mais altos resultam em brilhos mais concentrados. No sistema, o valor padrão de *shininess* foi definido como 50.0, o que produz um brilho considerado de intensidade média.

4.1.4 `Camera` (`core/camera.py`)

A classe `Camera` implementa uma câmera orbital baseada em coordenadas esféricas. A posição da câmera é representada por uma tripla composta pela distância ao centro da cena e por dois ângulos que descrevem a orientação horizontal e vertical. Esses parâmetros são convertidos para coordenadas cartesianas para uso com a função `gluLookAt`. Para evitar problemas de *gimbal lock*, o ângulo vertical é limitado a um intervalo entre aproximadamente -89° e +89°. A interação com o usuário é feita de forma que arrastar o mouse provoque uma rotação ao redor da origem, enquanto o movimento horizontal ajusta a rotação em torno do eixo Y e o movimento vertical ajusta a rotação em torno do eixo X. O scroll do mouse altera a distância da câmera, produzindo um efeito de zoom.

4.2 Objetos Geométricos

4.2.1 Cubo (`geometry/cube.py`)

O cubo implementado é centrado na origem e possui aresta de comprimento igual a duas unidades. Seus oito vértices são definidos manualmente em torno da origem, e a superfície é formada por seis faces quadriláteras, cada uma desenhada com quatro vértices usando `GL_QUADS`. Para cada face, é definida uma normal perpendicular, de modo que o total de vértices enviados para a rasterização chega a 24, considerando que cada face possui sua própria normal.

4.2.2 Pirâmide (`geometry/pyramid.py`)

A pirâmide possui base quadrada e um ápice posicionado acima do centro da base. A implementação considera um vértice correspondente ao topo (*apex*) e quatro vértices na

base. As quatro faces laterais são triangulares e são renderizadas com `GL_TRIANGLES`, enquanto a base é uma face quadrilátera. As normais das faces laterais são calculadas dinamicamente por meio do produto vetorial entre dois vetores de aresta, seguido de normalização, conforme ilustrado no trecho de código abaixo:

```
v1 = p2 - p1
v2 = p3 - p1
normal = v1.cross(v2).normalize()
```

4.2.3 Cone (`geometry/cone.py`)

O cone é gerado utilizando os recursos de GLU Quadrics. Ele possui raio igual a 1.0 unidade e altura igual a 2.0 unidades, com 32 subdivisões ao longo da circunferência para garantir uma boa aproximação da superfície curva. O corpo do cone é desenhado com a função `gluCone`, e as tampas são criadas com `gluDisk`, tanto na base quanto na parte superior, quando necessário. As normais são geradas automaticamente em modo suave (`GLU_SMOOTH`), o que favorece uma aparência mais contínua sob modelos de iluminação mais sofisticados.

4.2.4 Esfera (`geometry/sphere.py`)

A esfera também é gerada com GLU Quadrics. Ela possui raio de 1.2 unidades e é subdividida em 32 segmentos em latitude e 32 em longitude, resultando em aproximadamente 1024 triângulos na superfície. As normais de cada vértice são geradas automaticamente, apontando radialmente para fora, o que é ideal para modelos de iluminação baseados em normais suaves.

5 Modelos de Iluminação

5.1 Flat Shading (`shading/flat_shading.py`)

No modelo Flat Shading, a iluminação de cada polígono é calculada apenas uma vez, com base na normal da face. Dessa forma, todos os pontos de uma mesma face recebem a mesma cor, produzindo uma aparência claramente facetada. A implementação utiliza o comando `glShadeModel(GL_FLAT)` para configurar o modo de sombreamento no OpenGL. Entre as principais vantagens desse modelo, destacam-se a extrema rapidez e o baixo custo computacional, o que o torna adequado para objetos intencionalmente facetados. Por outro lado, o Flat Shading tende a produzir uma aparência artificial em superfícies curvas, nas quais as descontinuidades entre faces ficam bastante evidentes.

5.2 Gouraud Shading (`shading/gouraud_shading.py`)

No modelo Gouraud Shading, a iluminação é calculada nos vértices e os valores de cor obtidos são interpolados ao longo das arestas e do interior do polígono. A implementação é baseada no uso de `glShadeModel(GL_SMOOTH)`, o que habilita a interpolação de cores entre vértices. O processo consiste em calcular, para cada vértice, a contribuição ambiente, difusa e especular, e depois deixar que o pipeline gráfico faça a interpolação durante a rasterização, muitas vezes utilizando interpolação bilinear em tela. Esse modelo produz transições suaves de iluminação entre faces vizinhas e apresenta bom desempenho, já

que o cálculo de iluminação é feito apenas nos vértices. No entanto, há desvantagens: *highlights* especulares muito concentrados podem ser perdidos quando o brilho máximo ocorre no interior de um polígono, e o fenômeno de *Mach bands* pode causar bandas visuais indesejadas em gradientes de iluminação.

5.3 Phong Shading (`shading/phong_shading.py`)

O modelo Phong Shading realiza uma interpolação das normais ao longo da superfície e calcula a iluminação por pixel, no *fragment shader*. Em vez de interpolar cores, como no Gouraud, o Phong interpola vetores normais e, para cada fragmento, recomputa a contribuição ambiente, difusa e especular. A implementação utiliza shaders GLSL customizados.

O *vertex shader* calcula a posição do vértice no espaço do mundo, armazena a posição e a normal transformada em variáveis *varying* e define a posição final do vértice na tela:

```
varying vec3 frag_position;  // Passa posi   o para fragment
varying vec3 frag_normal;    // Passa normal para fragment

void main() {
    vec4 world_pos = model_matrix * gl_Vertex;
    frag_position = world_pos.xyz;
    frag_normal = normalize(normal_matrix * gl_Normal);
    gl_Position = projection_matrix * view_matrix * world_pos;
}
```

O *fragment shader* normaliza a normal interpolada, calcula os vetores direção da luz, direção do observador e direção do reflexo, e em seguida computa as componentes ambiente, difusa e especular:

```
void main() {
    vec3 normal = normalize(frag_normal);
    vec3 light_dir = normalize(light_position - frag_position);
    vec3 view_dir = normalize(view_position - frag_position);
    vec3 reflect_dir = reflect(-light_dir, normal);

    // Componente Ambiente
    vec3 ambient = light_ambient * material_ambient * 2.0;

    // Componente Difusa (Lei de Lambert)
    float diff = max(dot(normal, light_dir), 0.0);
    vec3 diffuse = light_diffuse * (diff * material_diffuse) * 1.2;

    // Componente Especular (Modelo de Phong)
    float spec = pow(max(dot(view_dir, reflect_dir), 0.0),
       shininess);
    vec3 specular = light_specular * (spec * material_specular) *
       0.8;

    // Cor final
    vec3 result = ambient + diffuse + specular;
    gl_FragColor = vec4(clamp(result, 0.0, 1.0), 1.0);
}
```

O Phong Shading oferece *highlights* especulares muito precisos, produz iluminação mais realista e posiciona corretamente o brilho em superfícies curvas. Em contrapartida, é mais custoso computacionalmente e requer o uso de shaders customizados, não estando disponível no antigo pipeline fixo. Em termos de desempenho relativo, considera-se o Flat como referência de 100%, o Gouraud com cerca de 105% do custo e o Phong entre 150% e 200% do custo, dependendo da resolução e complexidade da cena.

6 Transformações Geométricas

A rotação dos objetos foi implementada utilizando as funções de transformação do OpenGL. Para cada eixo, chama-se `glRotatef` com o ângulo e o eixo apropriado, por exemplo `glRotatef(angle_x, 1, 0, 0)` para rotação em X, `glRotatef(angle_y, 0, 1, 0)` para rotação em Y e `glRotatef(angle_z, 0, 0, 1)` para rotação em Z. Os ângulos variam de 0° a 360° em cada eixo. A escala é aplicada por meio da função `glScalef`, usando um fator uniforme em todos os eixos, com valores típicos entre 0.1 e 2.0, o que corresponde a escalas entre 10% e 200%. Já a translação é feita com `glTranslatef(tx, ty, tz)` e é usada principalmente para posicionar os objetos quando o modo de comparação está ativado.

7 Projeções

A projeção perspectiva é utilizada para simular a visão humana, na qual objetos mais distantes parecem menores. Ela foi configurada com a função `gluPerspective`, utilizando um campo de visão de 45 graus, a razão de aspecto da janela, um plano de corte próximo em 0.1 e um plano de corte distante em 100.0. Nessa projeção, linhas paralelas convergem, a sensação de profundidade é reforçada e o resultado visual tende a ser mais realista.

A projeção ortográfica, por sua vez, é uma projeção paralela que preserva as proporções dos objetos independentemente da profundidade. Ela foi configurada com a função `glOrtho`, definindo limites esquerdo e direito proporcionais ao tamanho e à razão de aspecto, limites inferior e superior baseados em um parâmetro de escala e planos de corte próximos e distantes em 0.1 e 100.0. Nesse modo, linhas paralelas permanecem paralelas e não existe distorção perspectiva, o que é particularmente útil para visualizações técnicas e vistas isométricas.

8 Interface Gráfica

A interface da aplicação foi dividida em duas áreas principais. A maior parte da janela, aproximadamente 75% da largura, é ocupada pelo widget de visualização 3D baseado em OpenGL, que exibe a cena renderizada em tempo real. Na lateral, em cerca de 25% da janela, encontra-se um painel de controle que reúne os elementos de interação com o usuário.

No painel, o usuário pode escolher qual objeto 3D será exibido por meio de uma caixa de combinação com quatro opções: cubo, pirâmide, cone e esfera. Em outra caixa de combinação, é possível selecionar o modelo de iluminação desejado entre Flat, Gouraud e Phong. Um botão do tipo *toggle* permite alternar entre o modo normal, no qual apenas um objeto é exibido com o modelo selecionado, e o modo de comparação, em que três

cópias do mesmo objeto são exibidas lado a lado, cada uma com um modelo de iluminação diferente.

A projeção utilizada na visualização também pode ser escolhida por meio de uma caixa de combinação que alterna entre projeção perspectiva e projeção ortográfica. A rotação do objeto é controlada por três *sliders* independentes, um para cada eixo (X, Y e Z), variando de 0° a 360°, com atualização em tempo real da cena conforme o usuário move os controles. A escala é ajustada por um *slider* único que varia de 10% a 200%, e o valor atual é exibido de forma dinâmica.

A posição da fonte de luz é configurada por três *sliders* que controlam as coordenadas X, Y e Z em um intervalo típico entre -5.0 e 5.0. A luz é representada visualmente na cena por uma pequena esfera amarela, facilitando a compreensão do usuário sobre sua localização. Além desses controles, há botões para ativar uma animação de rotação automática contínua em torno do eixo Y e para restaurar a vista para as configurações iniciais, o que facilita o recomeço da exploração visual.

Na interação com o mouse, arrastar com o botão esquerdo provoca a rotação da câmera ao redor do objeto, com movimentos horizontais correspondendo a rotações em torno do eixo Y e movimentos verticais correspondendo a rotações em torno do eixo X. O uso do scroll permite aproximar ou afastar a câmera, simulando um zoom, com a distância limitada a um intervalo entre 2 e 20 unidades para evitar que o objeto seja atravessado ou fique excessivamente distante.

9 Modo Comparação

O modo comparação foi concebido como um recurso didático para permitir a observação simultânea dos três modelos de iluminação aplicados ao mesmo objeto. Quando esse modo é ativado, a cena passa a exibir três instâncias do objeto escolhido, alinhadas horizontalmente e separadas por cerca de 3,5 unidades. A instância situada à esquerda é renderizada com Flat Shading e colorida em vermelho, a instância central utiliza Gouraud Shading e é colorida em verde, enquanto a instância à direita adota Phong Shading e é colorida em azul. Para acomodar melhor as três cópias na mesma vista, cada objeto é desenhado com escala de aproximadamente 80% do tamanho normal.

Sobre a imagem renderizada, são desenhadas legendas utilizando `QPainter`, com um fundo semitransparente de cantos arredondados para melhorar a legibilidade. Cada legenda indica o modelo de iluminação correspondente e utiliza uma cor consistente com a do objeto exibido. Do ponto de vista educacional, esse modo facilita a comparação direta entre os modelos: o Flat enfatiza a natureza facetada das faces e destaca descontinuidades entre polígonos; o Gouraud produz superfícies mais suaves, mas pode perder *highlights* especulares muito concentrados; e o Phong gera superfícies visualmente contínuas, com brilhos especulares bem definidos e colocados no local correto.

10 Desafios e Soluções

Durante o desenvolvimento, alguns problemas técnicos chamaram a atenção e exigiram soluções específicas. Em um primeiro momento, o modelo Phong passou a renderizar os objetos completamente pretos. A análise revelou que os *uniforms* dos shaders não estavam configurados corretamente, que a componente ambiente da luz estava muito baixa e que a cor do material não estava sendo passada do OpenGL para o shader. A solução consistiu

em capturar a cor atual por meio de `glGetFloatv(GL_CURRENT_COLOR)`, utilizá-la para inicializar os parâmetros de material no shader e ajustar as intensidades ambiente, difusa e especular, aumentando a componente ambiente de aproximadamente 30% para valores efetivos maiores e aplicando fatores multiplicativos adequados.

Outro problema recorrente foi o estado inconsistente do OpenGL após a renderização de objetos com shaders. Em alguns casos, o programa continuava utilizando o último *program* ativo, o que afetava a renderização de outros elementos da cena. Verificou-se que `glUseProgram(0)` não estava sendo chamado de forma consistente ao final da renderização de cada objeto. A correção envolveu garantir que, após o uso de cada shader, o estado fosse restaurado para o pipeline fixo com `glUseProgram(0)`, além de reconfigurar explicitamente a iluminação no início da função `paintGL()`.

Houve também um problema com a matriz de normais. Quando transformações de escala não uniforme eram aplicadas, as normais deixavam de ser unitárias e não preservavam a direção correta, o que gerava iluminação incorreta. A solução adotada foi calcular a matriz de normais como a inversa transposta da submatriz 3×3 da matriz modelo, conforme o padrão de computação gráfica:

```
1 normal_matrix = np.linalg.inv(model_matrix[:3, :3]).T
```

Por fim, observou-se que o grid não aparecia quando o modo de comparação estava ativado. A causa foi identificada na ausência de chamadas para os métodos `draw_grid()` e `draw_axes()` na função responsável pelo desenho da visão de comparação. A correção foi simples: incluir essas chamadas no início do método, garantindo que o grid e os eixos fossem desenhados em todos os modos de visualização.

11 Resultados Obtidos

O sistema desenvolvido implementa um conjunto robusto de funcionalidades. Foram incluídos quatro tipos de objetos tridimensionais (cubo, pirâmide, cone e esfera), três modelos de iluminação (Flat, Gouraud e Phong com shaders), dois tipos de projeção (perspectiva e ortográfica) e todas as transformações geométricas básicas, incluindo rotação, escala e translação. Além disso, foi implementado controle interativo da posição da luz, uma câmera orbital controlada pelo mouse, um modo de comparação visual entre os modelos de iluminação, uma animação automática de rotação e uma interface gráfica intuitiva que integra todos esses recursos.

Do ponto de vista visual, o Flat Shading apresenta uma aparência marcadamente facetada, com descontinuidades nítidas entre as faces, sendo particularmente útil para destacar a estrutura poligonal de objetos angulares. O Gouraud Shading produz superfícies mais suaves, com transições graduais de iluminação, sendo adequado para superfícies curvas sob iluminação difusa, embora possa falhar na representação precisa de *highlights* especulares muito concentrados. Já o Phong Shading oferece a melhor qualidade visual, com superfícies suaves e brilhos especulares precisos, tornando-se a opção mais apropriada para materiais brilhantes, como metais e plásticos, ainda que a um custo computacional superior. Em todos os casos, o desempenho permaneceu adequado para aplicações interativas.

Em termos de qualidade de código, o projeto acumula aproximadamente 2000 linhas, distribuídas em 21 classes. A documentação cobre a totalidade ou a grande maioria das classes e métodos, e a organização modular foi considerada bastante satisfatória, facilitando a manutenção e a expansão futura do sistema.

12 Exemplos de Execução

No primeiro exemplo analisado, foi utilizado um cubo sob iluminação Phong e projeção perspectiva. Os ângulos de rotação foram configurados como 30° em X, 45° em Y e 0° em Z, e a fonte de luz foi posicionada em $(3.0, 3.0, 3.0)$. A imagem resultante, ilustrada na Figura 1, apresenta *highlights* especulares nítidos nas arestas superiores e transições suaves entre as faces, produzindo um aspecto de material plástico realista.

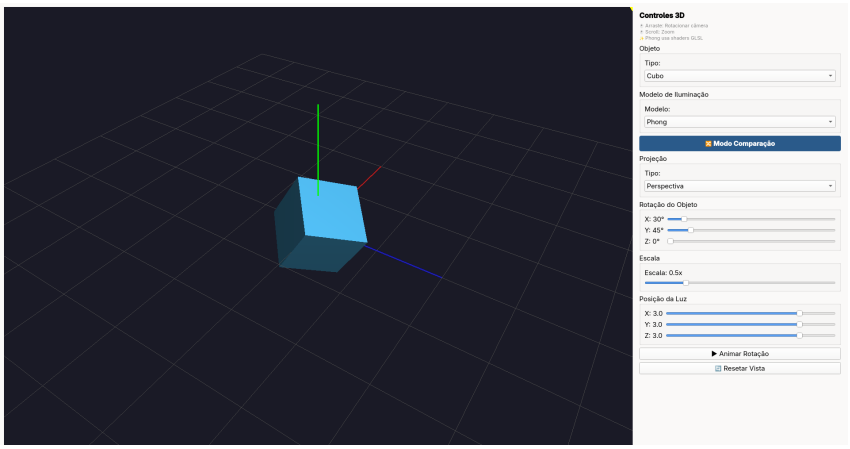

Figura 1: Cubo renderizado com iluminação Phong e projeção perspectiva.

No segundo exemplo, foi selecionada a esfera no modo de comparação, mantendo-se a projeção perspectiva. Nesse cenário, a esfera renderizada com Flat Shading passa a parecer um poliedro, com faces bem delineadas. Com Gouraud Shading, a esfera torna-se visualmente suave, mas o brilho especular tende a se espalhar demais, perdendo precisão. Já com Phong Shading, a esfera apresenta superfície perfeitamente suave e um *highlight* especular concentrado, posicionado de acordo com a localização da fonte de luz. A Figura 2 ilustra a visualização simultânea dos três modelos de iluminação aplicados à esfera.

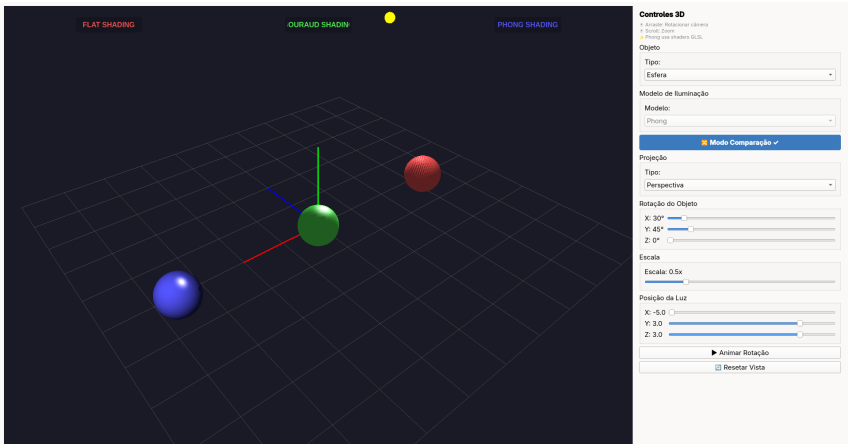

Figura 2: Esfera no modo de comparação, com Flat, Gouraud e Phong Shading.

13 Conclusões

13.1 Objetivos Alcançados

O trabalho atingiu plenamente os objetivos propostos. Foi desenvolvido um sistema completo de visualização 3D interativa, capaz de manipular objetos por meio de transformações geométricas, alternar entre diferentes projeções, aplicar diversos modelos de iluminação e permitir a comparação visual entre eles. Os três modelos de iluminação (Flat, Gouraud e Phong) foram implementados de forma funcional e comparável. A interface mostrou-se intuitiva e responsiva, e o código é modular, bem documentado e preparado para extensões futuras. Além disso, foi implementada uma versão real do algoritmo de Phong utilizando shaders GLSL, o que trouxe o sistema para um patamar mais próximo de aplicações gráficas modernas.

13.2 Aprendizados

Do ponto de vista conceitual, o trabalho permitiu consolidar o entendimento das diferenças práticas entre modelos de iluminação, da importância das normais na renderização e do funcionamento do pipeline de renderização do OpenGL. Também reforçou os conceitos de transformações geométricas em 3D e de sistemas de projeção perspectiva e ortográfica. Do ponto de vista técnico, o desenvolvimento do projeto proporcionou experiência em programação de shaders GLSL, uso combinado de PyOpenGL e PyQt6, arquitetura de software orientada a objetos, *debug* de aplicações gráficas 3D e integração de múltiplos subsistemas em uma única aplicação.

13.3 Comparação dos Modelos de Iluminação

Aspecto	Flat	Gouraud	Phong
Qualidade visual	Baixa	Média	Alta
Performance	Excelente	Muito boa	Boa
Uso de memória	Mínimo	Baixo	Médio
Complexidade	Simples	Média	Alta
Highlights	Imprecisos	Aproximados	Precisos
Melhor para	Objetos facetados	Geral	Materiais brilhantes

Tabela 1: Comparação dos modelos de iluminação.

A Instruções de Instalação

```
# Criar ambiente virtual
python -m venv venv

# Ativar ambiente
# Windows:
venv\Scripts\activate
# Linux/Mac:
```

```
source venv/bin/activate

# Instalar dependências
pip install PyQt6 PyOpenGL PyOpenGL_accelerate numpy
```

B Instruções de Execução

```
cd trabalho_2
python main.py
```