Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

Disciplina: Computação Gráfica

2º Trabalho Prático

Transformações Geométricas 3D e Projeções

Luís Filipe Silva Forti 14592348
Luiza Rodrigues Cardoso 14593332
Docente responsável: Profa. Agma Juci Machado Traina

São Carlos
2º semestre / 2025

SUMÁRIO

1 Introdução **1**

2 Métodos e Resultados **1**
2.1 Escolha da IDE 1
2.2 Desenho e extrusão de Polígonos 2D 1
2.3 Câmera Perspectiva e Ortográfica 3
2.4 Transformações Geométricas 3D 5
2.5 Modelos de iluminação 6
2.5.1 Modelos *Flat* e de *Gouraud* 6
2.5.2 Modelo de *Phong* 7

3 Conclusão **9**

1. Introdução

Durante o semestre letivo, foram ensinadas técnicas para processamento de imagens 3D na área de computação gráfica, como: transformações, formas de rendering, tipos de algoritmos para processamento de iluminação, etc. Dentre os modelos de iluminação, destacam-se o modelo *flat*, o *gouraud* e o *phong*.

Este projeto tem como objetivo o desenvolvimento do trabalho anterior, que se limitava à criação de polígonos 2D, avançando para fazer sua extrusão 3D. Também foram desenvolvidos cenários com objetos pré-definidos 3D para a demonstração dos modelos de iluminação, incluindo todo o código para o algoritmo do modelo de *phong*.

O projeto foi feito em C++ com a API OpenGL, responsável por fazer a renderização dos vértices passados para a janela. Todo o código foi desenvolvido e testado inteiramente no sistema operacional Ubuntu. Dentro do diretório principal do programa está o arquivo README.md, o qual descreve como criar e executar o executável.

A divisão do trabalho entre as partes envolvidas foi:

- **Luís:** limpeza e reestruturação do código. Consertar falhas e lógicas incompletas. Produção do relatório.
- **Luiza:** programação dos métodos de extrusão, modelagem da iluminação e interação com o usuário. Toda a estrutura central do código.

2. Métodos e Resultados

2.1. Escolha da IDE

Pela familiaridade dos integrantes, principalmente devido ás aulas práticas da matéria, optou-se por usar a IDE do OpenGL neste projeto. Ela possui métodos prontos para desenhar objetos 3D e o código de preenchimento de polígonos do trabalho anterior também foi, inicialmente, feito para a IDE.

2.2. Desenho e extrusão de Polígonos 2D

O primeiro passo foi aprender como interpretar os inputs do usuário, em especial o clique esquerdo do mouse, para poder reestruturar a funcionalidade do trabalho anterior de construir polígonos 2D quaisquer. Isso feito, descobriu-se como criar menus de inputs, que abrem-se com o clique do botão direito do mouse. Desta forma foi possível criar as configurações de cor de preenchimento e de contorno, além da grossura da linha.

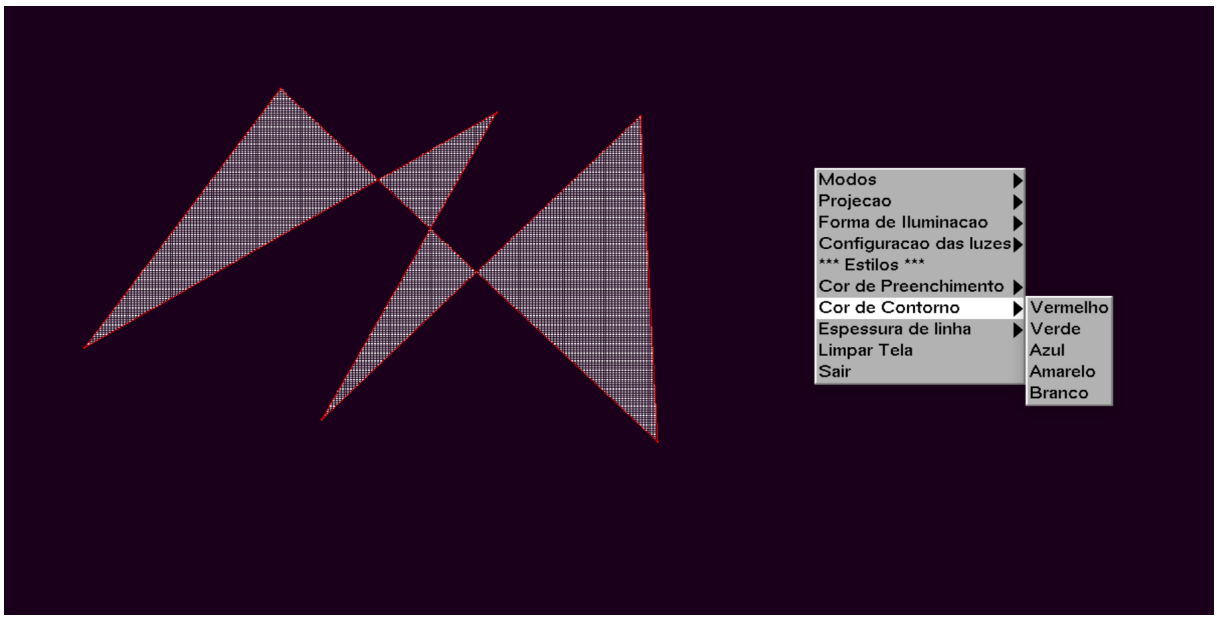

Figura 1: *Desenho complexo 2D e menu de opções*

Com o trabalho 1 adequadamente adaptado, avançou-se para a extrusão dos polígonos. O procedimento escolhido foi:

1. Coleta-se os vértices desenhados pelo usuário
2. Desenha o contorno do polígono no Z = 0
3. Repita o passo anterior em Z = 10
4. Usando o algoritmo de preenchimento de polígonos 2D, preenchem-se ambos os polígonos.
5. Conectam-se os pares de vértices com suas cópias, criando planos que conectam os polígonos através de Z

O OpenGL conta com funções pré-estabelecidas para facilitar o desenho dos contornos e dos planos do polígono 3D. No entanto, ele não consegue montar adequadamente polígonos complexos, adaptando-os para que se tornem côncavos. Isso resultou na escolha descrita na etapa 4: usar o algoritmo de preenchimento de polígonos 2D. Desta forma, o programa é capaz de criar objetos complexos, embora tenha um problema: o polígono parece ”esburacado”.

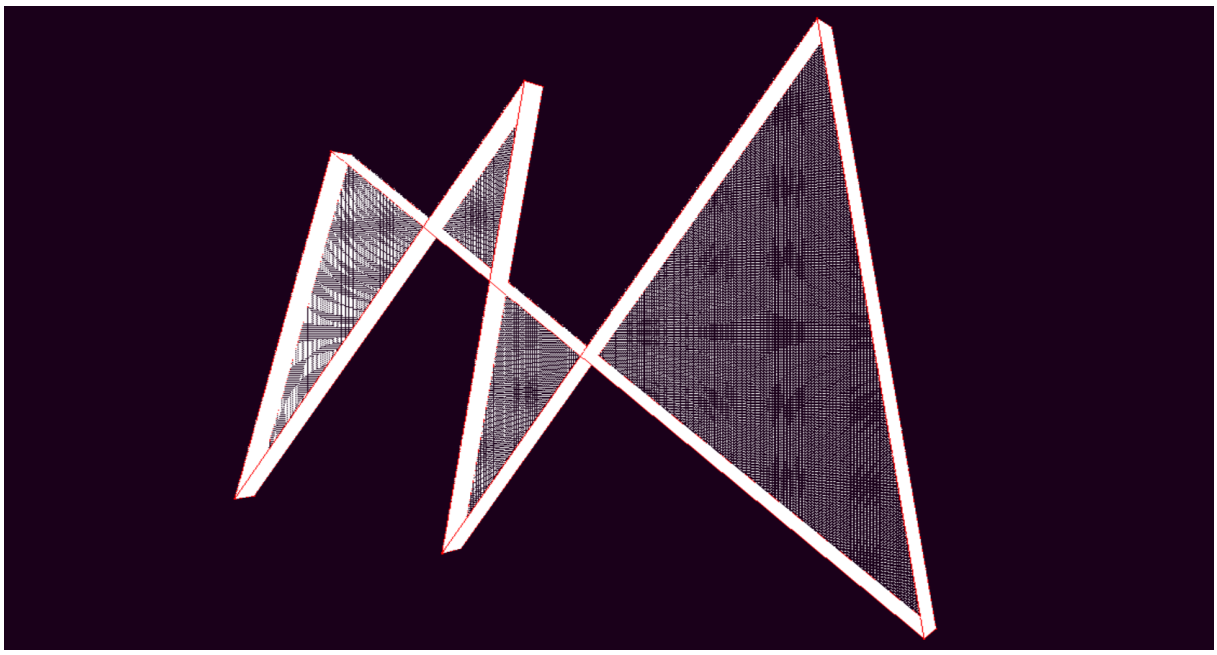

Figura 2: *Desenho complexo 3D levemente rotacionado*

Como pode-se ver nas figuras 1 e 2, o prenchimento do polígono está visualmente ”esburacado”, onde os pixels são extremamente visíveis. Isto é uma consequência da baixa resolução local definida aliada ao funcionamento do algoritmo, onde a tela possui apenas 800 píxels e o algoritmo trabalha apenas com intervalos inteiros. O grupo optou por esta forma, pois serve para mostrar visualmente o comportamento do algoritmo de preenchimento, tornando assim interessante deixá-lo desta forma para o princípio educativo do projeto.

2.3. Câmera Perspectiva e Ortográfica

Para a criação e manipulação da câmera, foi criada a classe **Camera**, responsável por armazenar todas as informações da câmera (figura 3), como: posição, vetor frente, vetor up, yaw, etc. Ela também manipula os inputs do usuário pelo teclado, movimentando-a de acordo com as seguintes instruções:

- Setas ($\leftarrow\uparrow\downarrow\rightarrow$) $\rightarrow$ movimentam a câmera
- WASD $\rightarrow$ rotacionam a câmera
- $+-$ $\rightarrow$ controlam o zoom

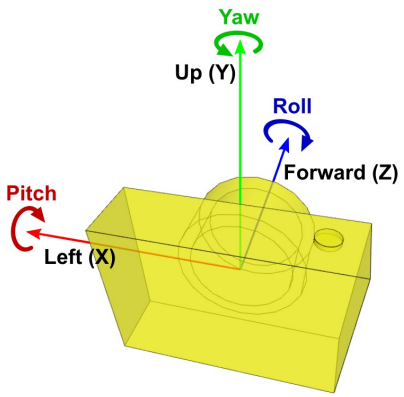

Figura 3: *Informações da câmera (Fonte: songho.ca)*

A câmera também tem duas configurações de projeção: perspectiva e ortográfica. Ambas as configurações são pré-estabelecidas pelo OpenGL, restando ao código apenas informar suas propriedades, como: campo de visão, largura, altura, distância máxima, etc.

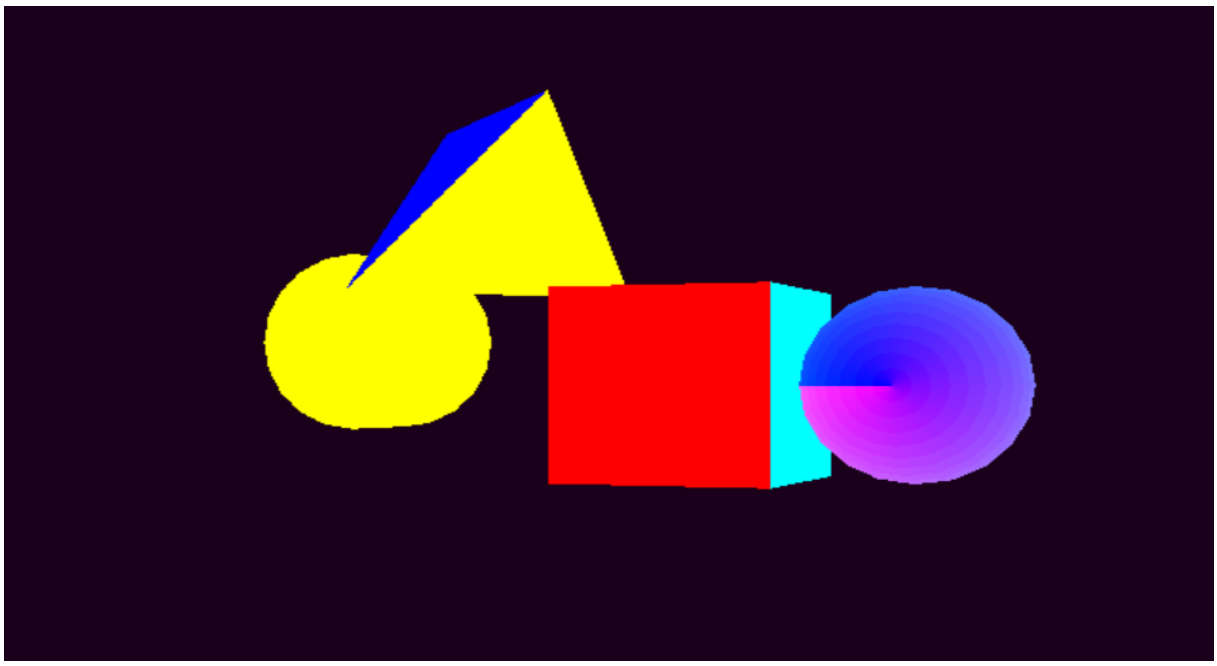

Figura 4: *Visão perspectiva de alguns objetos 3D*

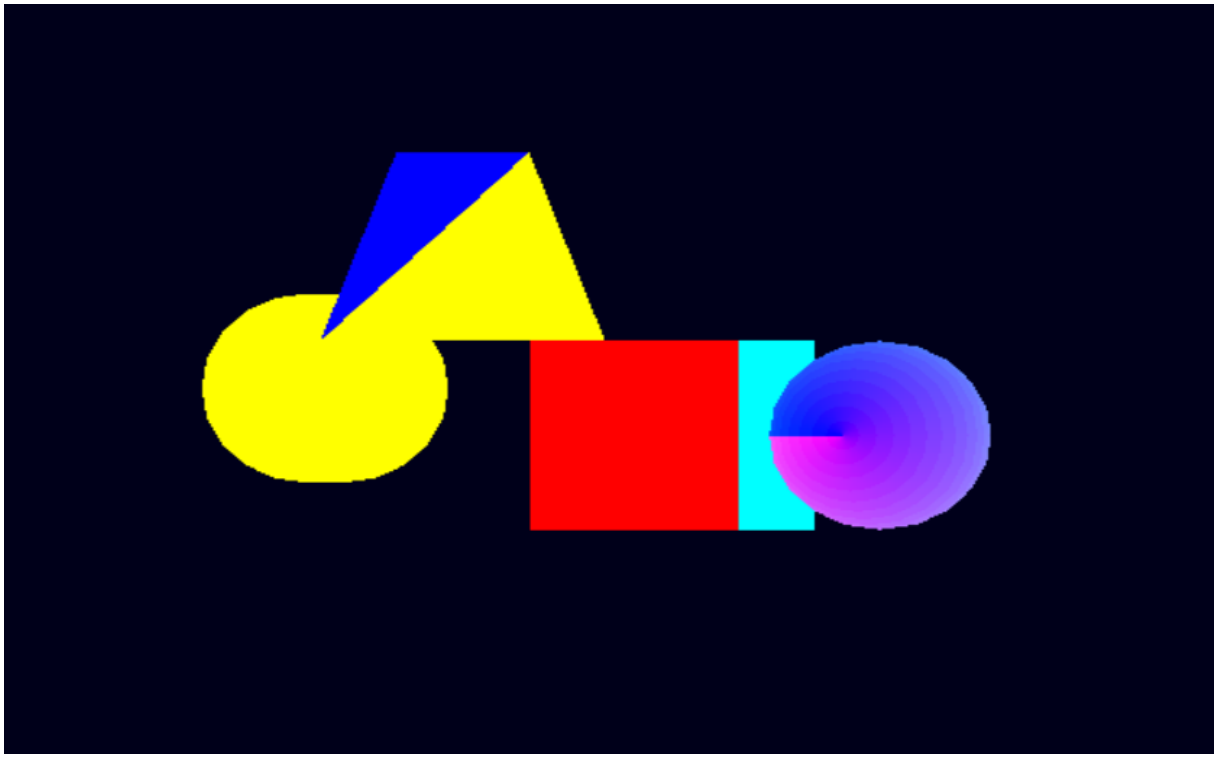

Figura 5: *Visão ortográfica de alguns objetos 3D*

2.4. Transformações Geométricas 3D

Para aplicar as transformações geométricas nos objetos 3D foram, novamente, utilizados inputs do teclado.

- GHJK → movimentam os objetos
- ZXCVFR → rotacionam os objetos
- BN → aumenta/diminuem a escala

Os inputs alteram os valores das variáveis de: translação em X ou Y, rotação em X, Y ou Z, escala dos objetos. Essas variáveis são, então, utilizadas pelas funções de transformação 3D do próprio OpenGL. Desta forma as transformações são tratadas automaticamente e sem grandes dificuldades.

```
glTranslatef(tx, ty, 0);
glRotatef(rxo, 1, 0, 0);
glRotatef(ryo, 0, 1, 0);
glRotatef(rzo, 0, 0, 1);
glScalef(scaleo, scaleo, scaleo);
```

Figura 6: *Trecho do código de transformação 3D*

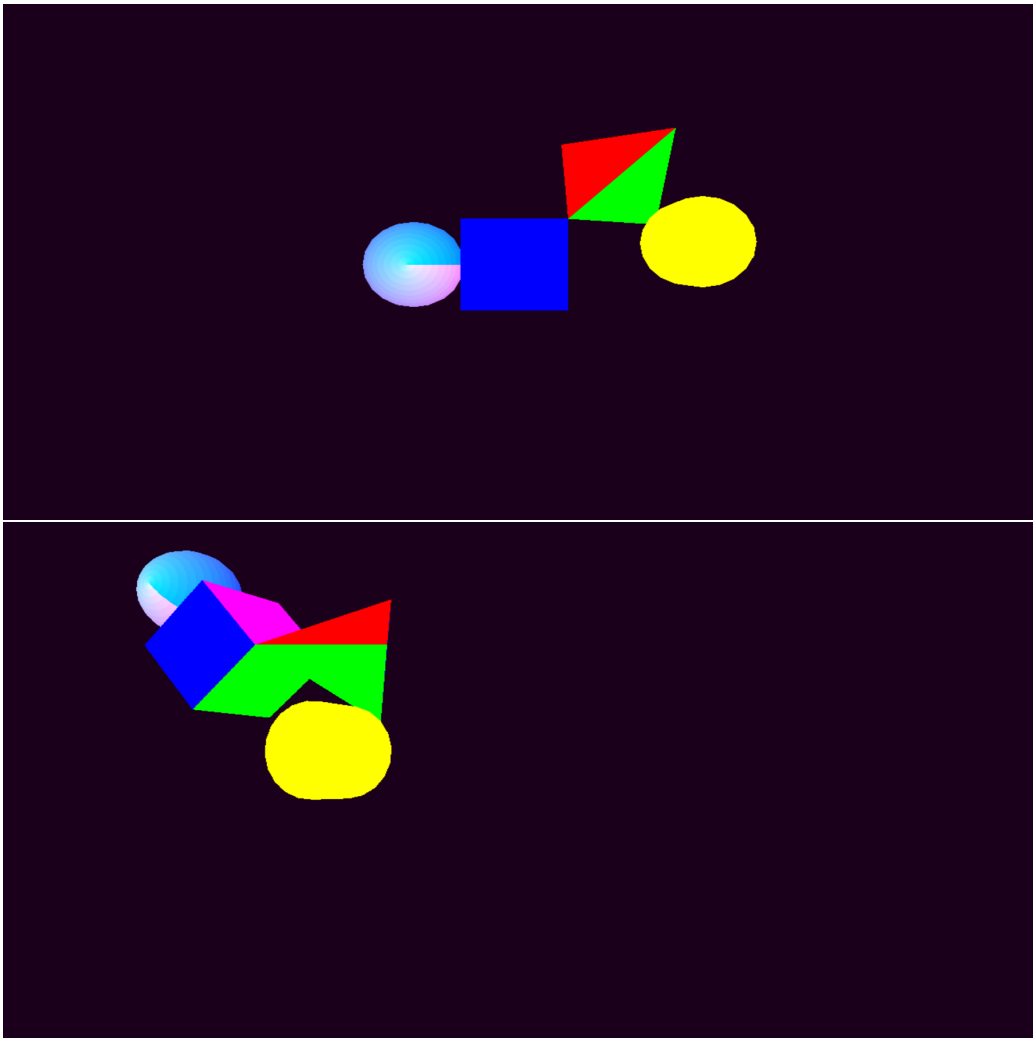

Figura 7: *Objetos 3D antes e depois de transladarem e rotacionarem*

2.5. Modelos de iluminação

2.5.1 Modelos *Flat* e de *Gouraud*

Para a implementação dos modelos *flat* e *gouraud* de iluminação, novamente recorreu-se ao OpenGL. A IDE possui suas próprias configurações de iluminação, as quais incluem os modelos em discussão.

Para o teste e análise deles, foram criados três pontos de luz de cores e comportamentos distintos: uma luz direcional quase branca, semelhante à luz do Sol, um ponto de emissão de luz azul-escuro e um ponto de iluminação em cone (tal qual um holofote) magenta. A luz direcional não possui um ponto de origem, simplesmente se espalha por toda a renderização na direção (x,y,z) = (-1, -1, 1). A luz pontual emite sua luz em todas as direções ao redor do ponto. Por fim, a luz magenta saí do ponto demarcado, propagando apenas numa área em cone na direção (x, y, z) = (-1, 0, 0). Estas combinações de cores de diferentes propriedades e comportamentos permitem visualizar bem o comportamento da luz nos objetos.

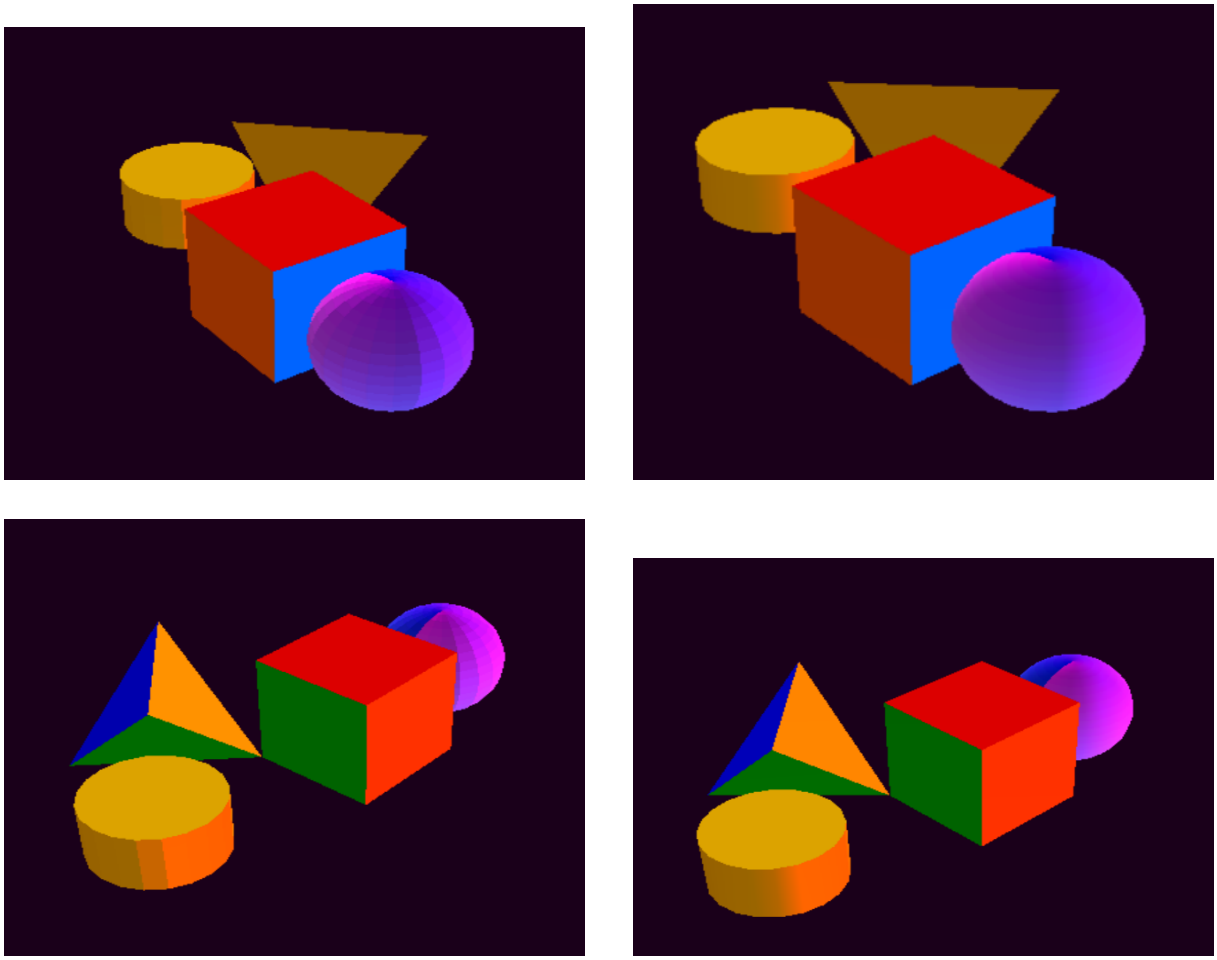

Figura 8: *Diferença entre flat (coluna da esquerda) e gouraud (coluna da direita)*

À direita de todas as imagens está a fonte de luz magenta, resultando na cor das esferas e na mudança de tonalidade das faces laranja/amarelas dos demais polígonos.

O polígono que torna mais perceptível a diferença entre os modelos é a esfera, principalmente na segunda linha. Enquanto no modelo *flat* é possível ver linhas que separam os tons da cor, no modelo *gouraud* elas desaparecem, mostrando o seu benefício: *gouraud* é muito mais sutil com a diferença de tonalidade entre pixels. Isso, no entanto, tem custo de eficiência, com o *gouraud* sendo consideravelmente mais pesado para processar.

2.5.2 Modelo de *Phong*

Por fim, o último tópico do projeto. O modelo de *Phong*, diferentemente dos demais, não possui uma forma nativa no OpenGL. Desta forma, restou ao grupo criar um algoritmo para fazer os cálculos e montar os objetos manualmente.

O primeiro passo foi montar a estrutura dos objetos que iriam ser processados. Foram criadas as classes **Cubo**, **Esfera** e **Piramide**, as quais seriam responsáveis por armazenar e calcular as informações pertinentes. A esfera, como é o único objeto sem vértices, foi adaptada para ser composta por cortes latitudinais e longitudinais, assim criando seus ”vértices”.

Para a criação das malhas, simplesmente montam-se tabelas definindo cada face do polígono. Desta forma, para o cubo, por exemplo, a malha seria composta por todas as combinações de 4 vértices que compõem suas faces. Como a esfera foi adaptada para ter vértices, sua malha é composta assim como no cubo: faces de 4 vértices.

O modelo de *Phong* requer, fora a malha do objeto, os vetores normais aos seus vértices. Embora contraintuitivo, todo vértice possui um vetor normal, que pode ser calculado pela média normalizada (comprimento final 1) da soma das normais das faces que possuem o vértice. Novamente voltando para o exemplo do cubo, a normal de qualquer um de seus vértices seria a soma das normais dos três vértices adjacentes. Essa soma é, então, dividida pelo seu módulo, assim a normalizando.

Tendo os dados dos objetos, resta apenas definir como é feito o cálculo do modelo. A imagem abaixo (figura 9) mostra o comportamento da luz ao incindir sobre um plano: a luz chega no plano a um ângulo β em relação à normal, o mesmo ãngulo da sua reflexão; a visão do observador possui um vetor principal, que chega no plano a um ângulo α da normal. Esses ângulos são fundamentais para compreender o comportamento da luz.

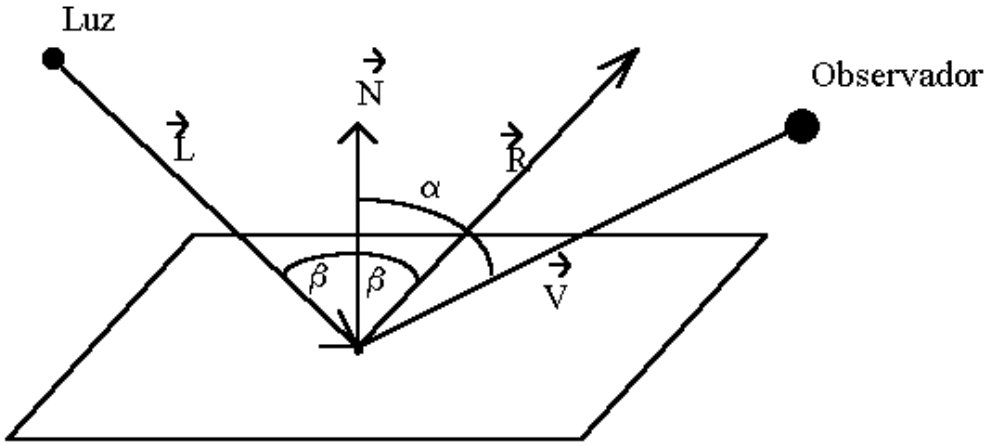

Figura 9: *Reflexão da luz para a análise do modelo de Phong (Fonte: PUCRS)*

Todo cenário possui uma ”luz ambiente”. Esta é a luz resultante da reflexão de todos os objetos do cenário, representada como uma ”fonte de luz global”, que afeta todos os pontos de todos os objetos igualmente. A luz de uma fonte, quando bate num ponto, pode ser distribuída para todas as direções, gerando uma luz difusa. Por fim, quando a luz é refletida de forma concentrada, se caracteriza como luz especular.

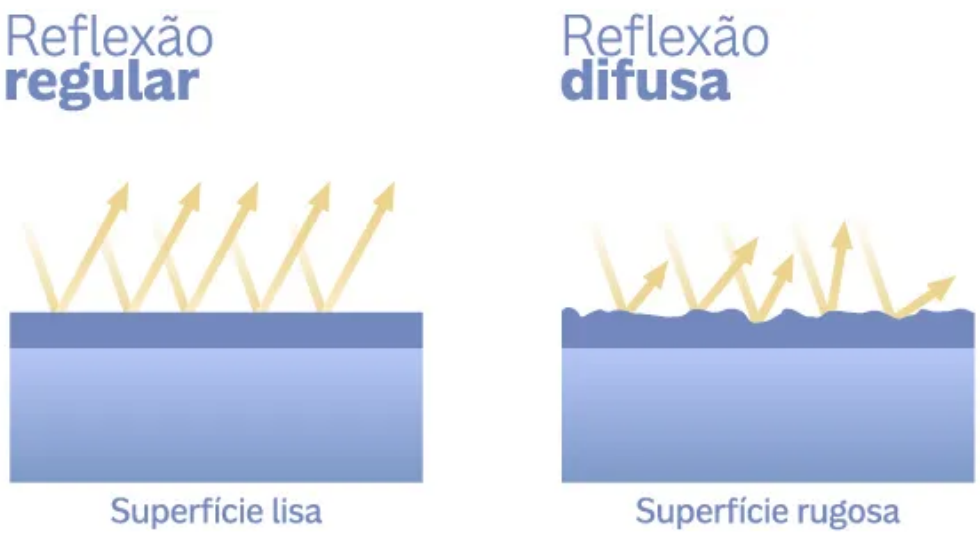

Figura 10: *Reflexão da luz para difusa e especular (Fonte: Brasil Escola)*

A combinação dessas luzes forma a equação do modelo de *Phong*, como mostra a equação 2.1. Todos os componentes (luz ambiente, difusa e especular), possuem constantes de atenuação (k_a, k_d, k_s), que dependem do material. O produto escalar entre a luz incidente e a normal ($\hat{L} \cdot \hat{N}$) representa que a intensidade da reflexão difusa depende do ãngulo de incidência α, diminuindo o valor quanto mais perpendicular à normal for. Algo semelhante ocorre à reflexão especular ($\hat{R} \cdot \hat{V}$), onde a intensidade da luz especular depende do ângulo entre o vetor da luz refletida e o vetor da visão do observador, novamente diminuindo quanto mais perpendicular estiverem. Por fim, o exponencial $(\hat{R} \cdot \hat{V})^{\alpha}$ representa o expoente de brilho. Quanto maior for α, mais concentrado será o brilho especular.

$$I = k_a i_a + k_d i_d (\hat{L} \cdot \hat{N}) + k_s i_s (\hat{R} \cdot \hat{V})^{\alpha} \tag{2.1}$$

Agora, resta apenas juntar todos os dados. As normais dos vértices calculadas anteriormente, serão as normais utilizadas pelos cálculos das intensidades de luz apresentados. Assim define-se as intensidades e cores da luz nos vértices, bastando apenas interpolá-los para cobrir todas as faces.

No projeto, isso foi feito três objetos azuis e duas fontes de luz: uma luz ambiente cinza e uma luz branca, ligada à câmera, que atua como fonte de luz difusa e especular. Assim, a luz estará sempre concentrada no ponto de visão do usuário, criando uma mancha especular fortíssima em objetos com superfícies perpendiculares à visão do usuário.

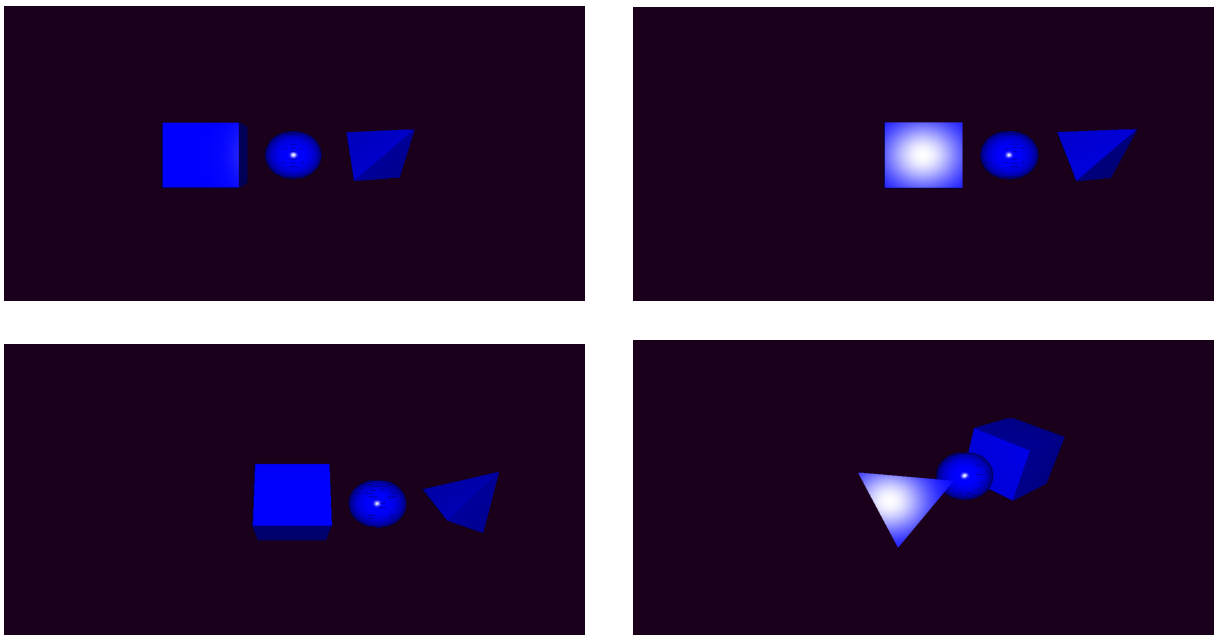

Figura 11: *Algumas configurações dos objetos com o modelo de Phong*

Infelizmente, não foi possível fazer com que a direção da fonte de luz alinhasse com a direção de visão da câmera (o vetor forwards da figura 3). Assim a fonte de luz só gerava o comportamento especular se a face do objeto estivesse na altura da câmera e alinhada ao eixo X ou Z da mesma.

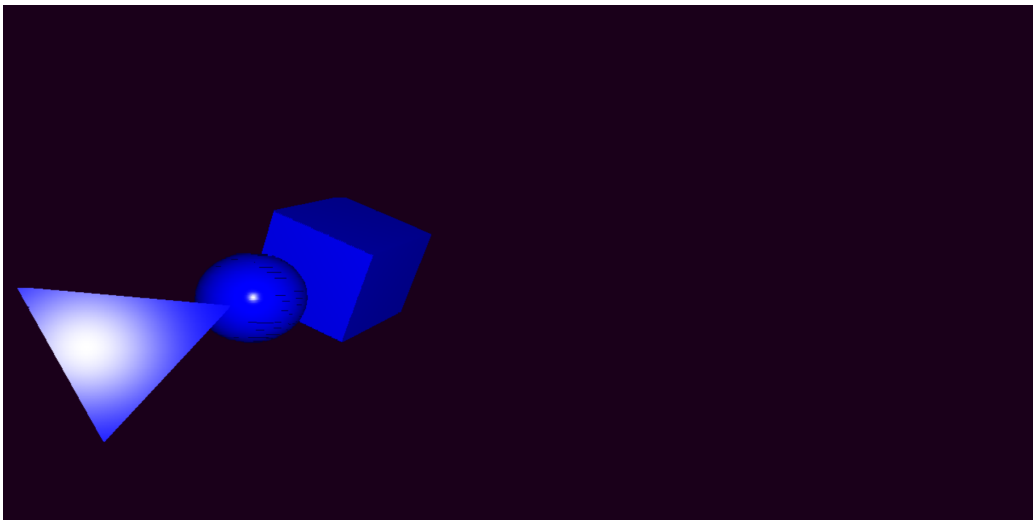

Figura 12: *Objetos na mesma posição da última imagem, apenas a câmera rotacionou para a direita. O comportamento especular se manteve, embora a câmera não esteja mais concentrada na face*

3. Conclusão

Os conhecimentos adquiridos ao longo do semestre, que iniciaram em conceitos básicos, como obtenção de imagens e desenhos de primitivas, até em conceitos mais abstratos, como métodos de rendering, modelos de projeção de câmera, entre outros, foram cruciais para realizar este trabalho. Logo que a partir deles foi construída uma plataforma de manipulação de polígonos em espaços 2D e 3D, além de exemplicar os modelos de iluminação que foram explicados em sala de aula.

Consideramos que nosso trabalho conseguiu cumprir os requisitos solicitados e se mostrou como um oportunidade para explorarmos o vasto universo da Computação Gráfica.