Universidade do Estado do Amazonas (UEA)
Escola Superior de Tecnologia (EST)
Curso: Engenharia de Computação
Data: 4 de dezembro de 2019
Disciplina: Computação Gráfica
Professor: Ricardo Barbosa
Alunos:
Carlos Diego Ferreira de Almeida
Jean Phelipe de Oliveira Lima
João Victor Melo de Oliveira
Luiz Carlos Silva de Araújo Filho

Atividade Prática
Reflexão de Luz com OpenGL

Iluminação no mundo real é extremamente complicado e depende de muitos fatores, e calcular todas essas variáveis é um luxo que não podemos nos dar devido ao limitado poder de processamento das máquinas. Iluminação em `OpenGL` é, portanto, baseada em aproximações da realidade usando modelos simplificados que são mais fáceis para processar e parecem muito similares aos reais. Esses modelos de iluminação são baseados na teoria física da luz. Esse trabalho visa documentar a atividade prática realizada em `OpenGL` para implementar reflexão da luz de um Sol virtual.

1 Fundamentação Teórica

Nesta seção são descritos os principais conceitos e técnicas que serviram de base para o desenvolvimento deste trabalho.

1.1 Conceitos de Álgebra Linear

Computação gráfica tem suas bases na álgebra linear. Ela fornece as ferramentas matemáticas necessárias para que se torne possível a apresentação de objetos (e modificações sobre os mesmos) numa tela de computador.

1.1.1 Plano Cartesiano

É um sistema de coordenadas inventando por René Descartes muito utilizado na matemática, pois é uma maneira de se representar graficamente expressões algébricas.

O plano, ilustrado na Figura 6, é composto por duas retas perpendiculares e orientadas, uma horizontal e outra vertical. À reta horizontal, dá-se o nome de **eixo x** ou **eixo das abscissas**; à reta vertical, **eixo y** ou **eixo das ordenadas**. A orientação positiva das retas é representada por uma seta.

No plano cartesiano, pares de coordenadas do tipo **(x, y)** são representados como pontos num plano 2D. Um conjunto de pontos traçados podem representar das mais simples até as mais complexas das funções algébricas existentes.

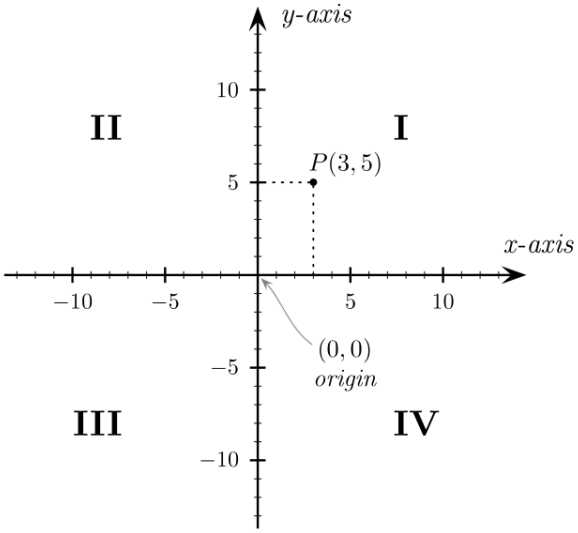

Figura 1: Plano cartesiano

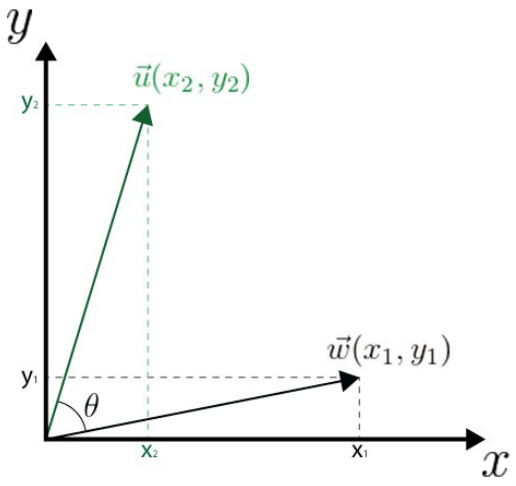

Figura 2: Vetores no plano

1.1.2 Coordenadas Homogêneas

A subseção anterior introduziu a ideia de coordenadas no plano cartesiano 2D. Em computação gráfica, há uma impossibilidade de realizar certas transformações em objetos gráficos a partir de matrizes 2 x 2. Para contornar esse empecilho, coordenadas homogêneas são utilizadas.

Uma coordenada homogênea é um triplo (x, y, t) de números reais equivalente a $(x/t, y/t)$ no plano 2D convencional. Elas são utilizadas em geometria projetiva pois facilitam uma série de operações algébricas e possibilitam representar pontos e direções que tendem ao infinito. Com esse sistema, é possível regularizar as transformações geométricas necessárias em computação gráfica.

1.1.3 Vetores

Um vetor geométrico no plano cartesiano é uma classe de objetos matemáticos (segmentos) com mesma direção, sentido e módulo. Um vetor, ilustrado na Figura 7, é formado a partir de 2 pares ordenados no plano, conhecidos como ponto de origem e de ponto extremidade. As coordenadas do vetor são obtidas pela diferença entre esses pontos.

As características de um vetor são definidas como a seguir:

1. A direção é dada pela reta que define o segmento;
2. O sentido é dado pelo sentido do movimento;
3. O módulo é o comprimento do segmento.

1.1.4 Matrizes

Uma matriz é um arranjo de números, símbolos, letras, etc., dispostos em linhas e colunas. Uma matriz com **m** linhas e **n** colunas é uma matriz de ordem **m x n**. Por

convenção, matrizes são representadas por letras maiúsculas enquanto seus elementos o são por letras minúsculas. Abaixo, X é uma matriz $\mathbf{m \times n}$ genérica e A, uma $\mathbf{3 \times 4}$

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & x_{d3} & \dots & x_{dn} \end{bmatrix} \quad A = \begin{bmatrix} 5 & 2 & 3 & 11 \\ 21 & 10 & 7 & 8 \\ 15 & 9 & 6 & 16 \end{bmatrix}$$

1.1.5 Produto de Matrizes

É possível realizar diversas operações entre matrizes como adição, subtração, produto por escalar dentre outras. Essas operações carregam diversas propriedades consigo.

A operação descrita nessa seção é a multiplicação de matrizes, a mais importante das operações para computação gráfica.

Dadas as matrizes $A = [a_{ik}]_{m\,x\,t}$ e $B = [b_{kj}]_{t\,x\,n}$, o produto das matrizes A e B é uma matriz $C = [c_{ij}]_{m\,x\,n}$ cujos elementos c_{ij} são da forma:

$$c_{ij} = \sum_{k=1}^{t} a_{ik} b_{kj}.$$

O número de colunas da primeira matriz *deve* ser igual ao número de linhas da segunda. Além, o produto de matrizes não é comutativo, isto é, a ordem na multiplicação pode alterar a matriz resultante.

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

Se multiplicarmos as matrizes A e B acima, a matriz C resultante será igual a matriz mostrada abaixo.

$$C = \begin{bmatrix} c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} & c_{13} = a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ c_{21} = a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} & c_{23} = a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ c_{31} = a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} & c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} & c_{33} = a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{bmatrix}$$

1.2 Conceitos Físicos de Luz

Essa seção visa cobrir um pouco da teoria física necessária para o entendimento de como a luz funciona no mundo real.

1.2.1 Intensidade Luminosa

O modelo de reflexão de *Phong* (ver seção 1.3.5) é vagamente baseado no modo como a luz se comporta no mundo real. Portanto, para entender a iluminação no `OpenGL`, precisamos entender o básico sobre a física da luz.

A luz branca contém todas as cores que o olho humano pode ver. Isso pode ser demonstrado ao colocar uma luz branca em um prisma, ilustrado da Figura 4 (a), o que faz a luz se dividir em um arco-íris.

Outra maneira de demonstrar isso é sobrepor três cores diferentes de luz (vermelho, verde e azul) e colocá-las sobre uma superfície branca em uma sala escura. O resultado é mostrado na Figura 4 (b).

Usando apenas três cores de luz, podemos criar oito cores diferentes: vermelho, verde, azul, amarelo, ciano, magenta, preto e branco.

Mas e as outras cores, como laranja? Demais cores, como laranja, são resultados da sobreposição das cores acima com o brilho reduzido como se vê na Figura 4 (c).

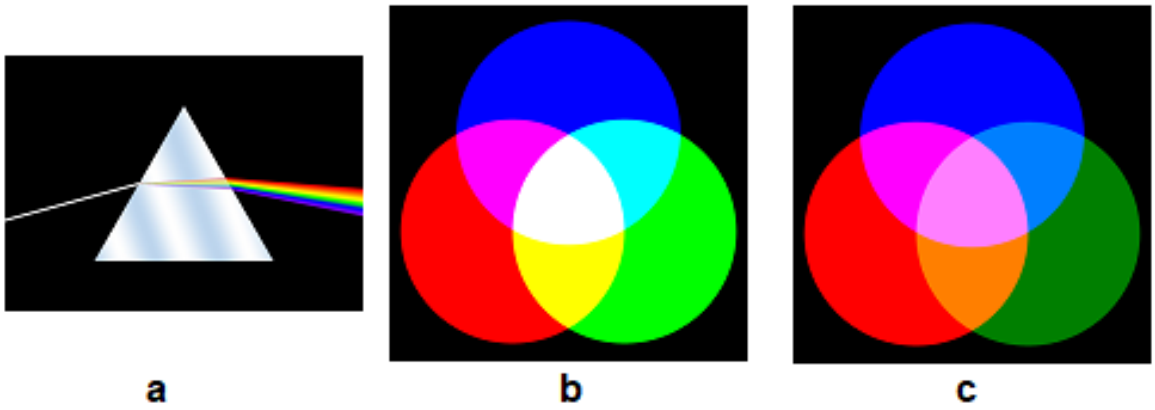

Figura 3: (a) Prisma, (b) RGB comum, (c) RGB metade do brilho

A redução da intensidade (também conhecida como brilho) do verde criou algumas novas cores: verde escuro, azul céu, laranja e rosa.

1.2.2 Absorção e Reflexão de Cor

Imagine que você está olhando para um carro vermelho. O sol emite um raio de luz branca. O raio salta do carro e entra nos seus olhos. Seu olho detecta que o raio contém apenas luz vermelha, e é por isso que você vê um carro vermelho em vez de um carro branco. Sabemos que a luz branca contém todas as cores, então o que aconteceu com o verde e o azul? A luz verde e azul foi absorvida pela superfície e a luz vermelha foi refletida.

E se fôssemos incindirmos uma luz ciana pura (azul + verde) no carro vermelho? Se o carro fosse vermelho puro, ficaria preto, porque absorveria 100% da luz.

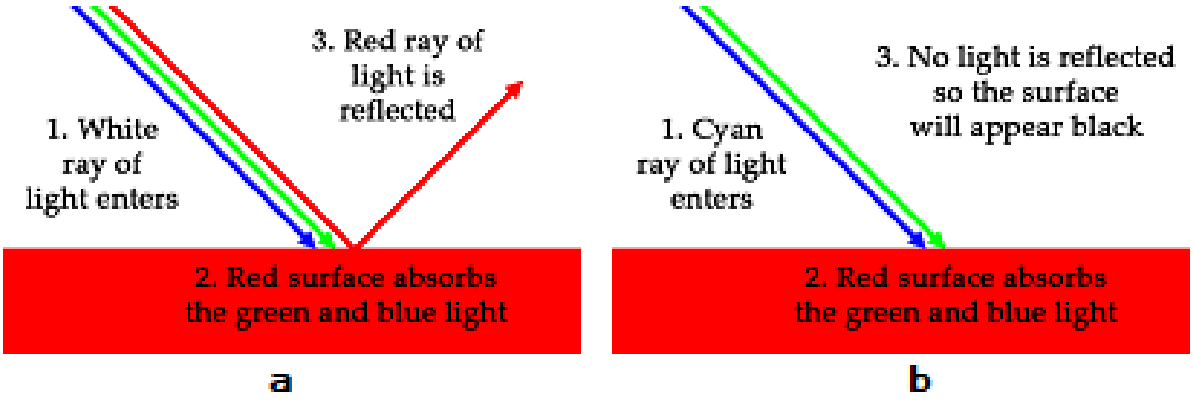

Figura 4: (a) Reflexão de luz vermelha, (b) Absorção total de luz

1.2.3 Ângulos de Incidência

O ângulo em que os raios de luz atingem a superfície é chamado ângulo de incidência (AoI, sigla em inglês). O ângulo de incidência afeta o brilho da superfície.

O brilho máximo ocorre quando a superfície é perpendicular aos raios de luz (AoI = 0^{o}). A escuridão completa ocorre quando a superfície é paralela aos raios de luz (AoI = 90^{o}). Se o AoI for maior que 90^{o}, o raio estará atingindo a parte de trás da superfície. Se a luz está atingindo a parte de trás, definitivamente não está atingindo a frente, portanto o *pixel* também deve estar completamente escuro. A Figura 5 ilustra o fenômeno.

Figura 5: Ângulos de incidência da luz num plano.

1.3 Conceitos de OpenGL

OpenGL é uma das APIs gráficas mais utilizadas na atualidade. Ela serve de *middleware*, facilitando cálculos vetoriais e matriciais necessários à computação gráfica. Uma outra API associada ao *OpenGL* é a GLUT, que facilita a criação de janelas e afins.

Algumas características do *OpenGL* são:

(a) É uma máquina de estados. Algumas funções do *OpenGL* guardam flags necessárias às configurações do programa como as características de cor de fundo de tela, configurações de janelas e outras habilitações ao programa;

(b) É orientada a eventos. O programa escrito utilizando a OpenGl permanece em loop até que algum evento aconteça. Na especificação dos eventos está mencionada a função *callback* que será chamada toda a vez que o evento a ela relacionado acontecer. O programa só se encerra se o evento correspondente ao final da execução for ativado.

(c) É independente da plataforma. As funções da *OpenGL* são padronizadas quanto ao formato passado ao usuário e ao programador. As diferenças em relação ao tratamento do sistema de janelas e do *hardware* existem apenas internamente ao código da API. Dessa forma os parâmetros e as funções não mudam e um código escrito para um compilador de uma plataforma UNIX, por exemplo, compila tranquilamente em um do *Windows* e vice-versa.

1.3.1 Sistemas de Coordenadas

O sistema de coordenadas de uma aplicação *OpenGL* tem origem no canto inferior esquerdo da tela (em relação ao observador). Assim os valores de **y** crescem à medida que o ponto se aproxima do topo da tela, e os valores de **x** crescem à medida que o ponto se aproxima da lateral direita da tela.

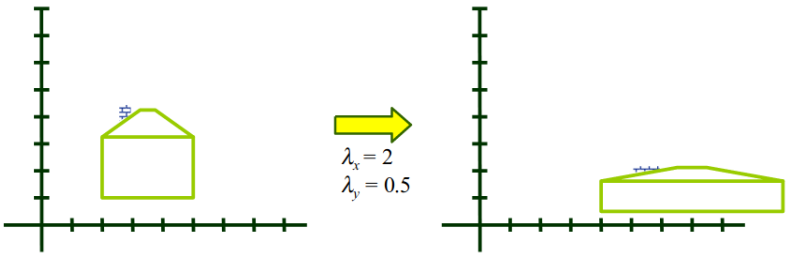

Figura 6: Scaling: representação gráfica.

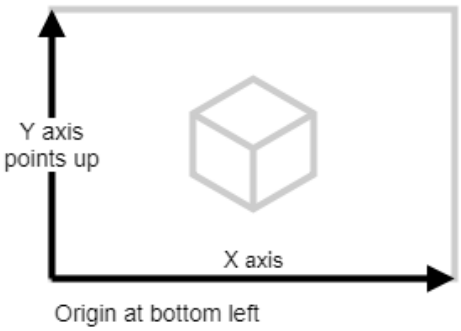

Figura 7: sistema de coordenadas do *OpenGL*.

1.3.2 Inicializando uma janela

Para desenhar utilizando um programa *OpenGL* é necessária uma janela de visualização. As configurações da janela ou das janelas de visualização são feitas na função `main()`.

O código a seguir inicializa uma janela vazia de acordo com as configurações dadas:

```
void main (int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(640, 480);
   glutInitWindowPosition(100, 150);
   glutCreateWindow(argv[0]);
   glutDisplayFunc(desenho);
   glutMainLoop();
}
```

`glutInit(&argc, argv)` Esta função inicializa a GLUT. Os argumentos poderão ser passados como parâmetros para o programa, como foi feito com o parâmetro `argv`, que representa o nome da janela, conforme especificado pela função `glutCreateWindow()`.

`glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)` Essa função especifica como o desenho será inicializado. A constante `GLUT_SINGLE` indica que será utilizado apenas um *buffer* para alocar a imagem – ou seja, o desenho será construído diretamente na tela –, e diz que a imagem será feita utilizando o padrão de cores RGB para colorir.

`glutInitWindowSize(640, 480)` Esta função especifica qual o tamanho inicial da janela. No caso são 640 *pixels* de largura por 480 *pixels* de altura. Quando o programa está rodando, o usuário pode redimensionar a janela, se preferir.

glutInitWindowPosition(100, 150) Esta função especifica que a janela posicionada a uma distância de 100 *pixels* do limite esquerdo da tela e a 150 *pixels* do limite superior da tela.

glutCreateWindow(argv[0]) Esta função abre uma janela de visualização de acordo com as configurações especificadas nas funções acima. Esta janela terá como nome o parâmetro passado para esta função, no caso a entrada de dados na chamada do executável de acordo com o parâmetro argv[0].

glutDisplayFunc(desenho) Esta é uma função *callback* necessária para a execução de qualquer programa que utilize a biblioteca GLUT (a partir da versão 3.0).

glutMainLoop() Esta função é responsável por manter o programa em *loop* até que seja chamado algum evento que determine o fim da aplicação. Essa função caracteriza a especificação de *OpenGL* como uma API orientada a eventos.

1.3.3 Orientação a Eventos

Como foi dito, a *OpenGL* é orientada a eventos e trata eventos de Mouse, de Teclado ou de Alterações da Janela de Visualização. A chamada dos eventos e seus *callbacks* ficam no `main()` do programa. O cabeçalho, declaração e a definição das funções são por conta do desenvolvedor.

A seguir, um trecho de código que manipula eventos a partir de funções *callback*:

```
glutDisplayFunc(desenho);
glutReshapeFunc(reshape);
glutMouseFunc(mouse);
glutKeyboard(teclado);
glutMainLoop();
```

glutDisplayFunc(desenho) Sempre que o programa determinar que uma janela deve ser redesenhada na tela, isso significa um evento de atualização de desenho. Neste exemplo a função `desenho()` é registrada como a função de atualização de desenho e chamada sempre que é preciso redesenhar uma cena.

glutReshapeFunc(reshape) O tamanho da janela de visualização pode ser alterado com o mouse por exemplo. Neste exemplo a função `reshape()` é registrada como um evento de atualização de tamanho de janela. A função `reshape`,passa parâmetros automaticamente para que sejam atualizados a largura e a altura da janela.

glutMouseFunc(mouse) Quando um botão do mouse é pressionado ou solto, um evento de mouse é acionado. Neste exemplo a função `mouse()` é registrada como uma função para ser chamada quando ocorrer eventos de *mouse*. A função `mouse()` recebe argumentos que identificam posição do cursor e a natureza da ação iniciadas na função.

glutKeyboardFunc(teclado) Este comando chama a função `teclado` quando uma tecla do teclado é pressionada ou solta. Os argumentos da função `teclado()` possuem argumentos que indicam que tecla foi pressionada além da posição do cursor naquele instante.

1.3.4 Desenhando Primitivas

Todos os desenhos produzidos pela *OpenGL* são gerados por primitivas geométricas básicas. Pontos, retas, sequência de retas, polígonos e outras são combinadas de forma que geram uma ampla gama de figuras que vemos em projetos que utilizam a API.

É preciso passar como parâmetro para a função de estado `glBegin()` a primitiva desejada com que se trace o desenho. Tudo o que estiver entre o `glBegin()` e o `glEnd()` será desenhado utilizando a primitiva passada como argumento.

No trecho de código abaixo, a função `desenho` desenha tudo aquilo por ela especificado utilizando pontos, que é a primitiva passada como parâmetro para a `glBegin()`.

```
glBegin(GL_POINTS);
desenho():
glEnd();
```

As funções da *OpenGL* seguem o seguinte padrão de nomeação:

$$\underbrace{\texttt{\{biblioteca\}}}_{1}\underbrace{\texttt{\{comando básico\}}}_{2}\underbrace{\texttt{\{número argumentos\}}}_{3}\underbrace{\texttt{\{tipo argumentos\}}}_{4}$$

1. indica a biblioteca à qual a função pertence;
2. é o radical da função, a função em si;
3. é a quantidade de parâmetros passados para a função;
4. é o tipo dos argumentos passados para a função.

O exemplo a seguir esclarece como o padrão funciona:

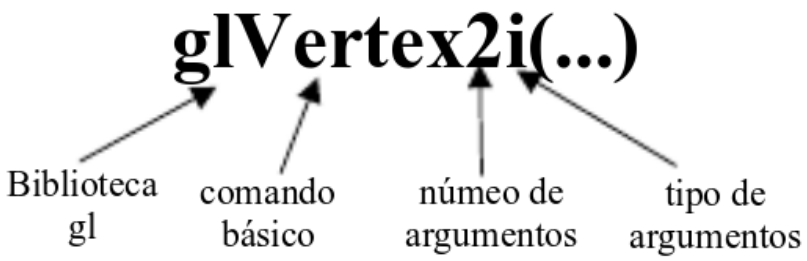

Figura 8: Padrão de nomeação: exemplo.

A API *OpenGL* possui seus próprios tipos de dados para evitar problemas de compatibilidade entre plataformas.

Sufixo	Tamanho de dados	Tipo de dado no C++	Nome na OpenGL
b	inteiro 8-bits	signed char	Glbyte
s	inteiro 16-bits	short	Glshort
i	inteiro 32-bits	int or long	Glint, Glsizei
f	floating points 32 bits	float	Glfloat, Glclampf
d	floating points 64 bits	double	Gldouble, Glclampd
ub	unsigned 8-bits	unsigned char	Glubyte, Glboolean
us	unsigned 16-bits	unsigned short	Glushert
ui	unsigned 32 bits	unsigned int or unsigned long	Gluint, enum, bitfield

Figura 9: Tipos de dados *OpenGL*

A seguir, são apresentadas as principais primitivas da *OpenGL*:
Exemplos renderizados das primitivas apresentadas são mostrados abaixo:

Primitiva	Descrição
GL_POINTS	Desenha pontos independentes
GL_LINE	Desenha uma linha entre dois pontos passados
GL_LINE_STRIP	Desenha uma sequência de linhas
GL_LINE_LOOP	Desenha uma sequência de linhas que retornam à origem
GL_QUADS	Desenha quadriláteros independentes a cada quatro pontos
GL_QUAD_STRIP	Desenha uma sequência de quadriláteros ligados
GL_TRIANGLES	Desenha triângulos independentes a cada 3 pontos
GL_TRIANGLE_STRIPS	Desenha triângulos conectados com origens diferentes
GL_TRIANGLE_FAN	Desenha triângulos conectados com origem comum
GL_POLYGON	Desenha um polígono convexo

Tabela 1: Primitivas da *OpenGL*

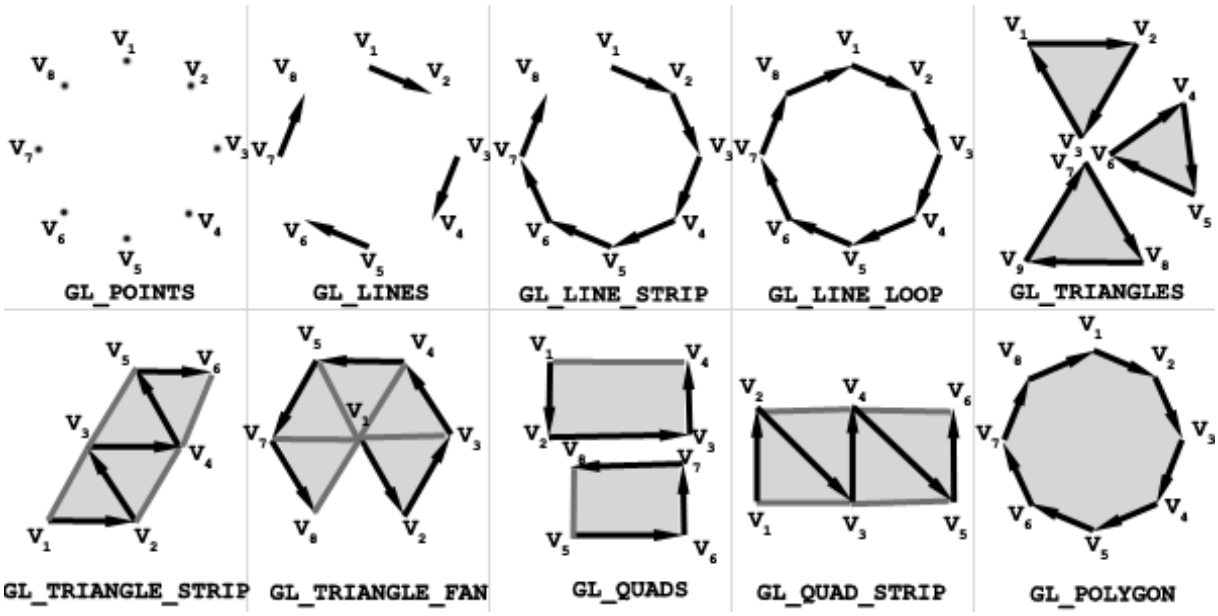

Figura 10: Exemplos de primitivas renderizadas

1.3.5 Iluminação Básica

Dos modelos de iluminação disponibilizados pela API, um dos mais famosos é o *Phong Lightning Model*.O modelo de reflexão *Phong* fornece um método para calcular a cor de um *pixel*, com base nos parâmetros de uma fonte de luz e superfície. Os parâmetros da fonte de luz incluem a posição, direção da luz, a cor e a intensidade da luz. Os parâmetros da superfície incluem a cor da superfície, a direção em que a superfície está voltada (também conhecida como normal) e o brilho da superfície.

Os blocos principais de construção desse modelo consistem em 3 componentes: luz ambiente, luz de difusão e luz especular. A figura 11 ilustra o modelo.

- Ambient lightning: simula um ambiente de baixa luz constante que sempre proporciona a aparição de um objeto. Sempre porque, por mais que o mundo se torne escuro, a incidência luminosa nunca é zero, devido à Lua.

- Diffuse lightning: simula o impacto direcional da luz no objeto. Quando maior a parte da face que recebe iluminação, mais brilhoso ele se torna.

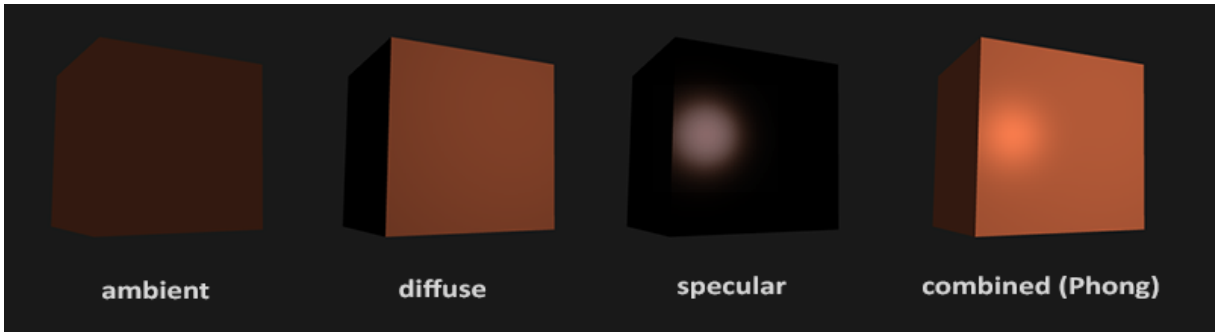

Figura 11: Modelo Phong.

- Specular lightning: simula o ponto da luz que aparece em objetos brilhosos. Depende mais da cor da luz que da cor do objeto.

A `OpenGL` nos fornece rotinas poderosas para realização dos efeitos luminosos citados e são aplicados na atividade prática.

2 Metodologia

Primeiramente, cada atividade foi dividia por integrante. Os programas foram desenvolvidos em máquinas relativamente modernas dos próprios alunos, em ambiente *Windows* ou *MacOS*, e após instalação de pacotes necessários e solução de problemas do processo de instalação. Em seguida, cada um estudou individualmente uma ou mais maneiras de resolver o problema. Majoritariamente, o procedimento de desenvolvimento consistiu em leitura de artigos pela *web* e visualização de vídeo-aulas em plataformas *online*. Os códigos da equipe foram desenvolvidos na linguagem de programação C utilizando a API `OpenGL` como suporte para renderização e criação dos efeitos necessários na atividade. Após solucionar o problema, a equipe se reuniu e discutiu os códigos-fontes.

3 Resultados

A atividade foi concluída e os resultados podem ser vistos na Figura xxx. Apesar das dificuldades, que foram responsáveis pela falta de fidelidade entre a teoria e a prática, o resultado do programa escrito é razoável.

Primeiramente o `GLUT` é iniciado com alguns parâmetros para permitir *double buffering* e o uso de cores são definidos. O programa tem quatro principais funções, `initGL()` onde todas as configurações de *display* e iluminação são definidas, a função `draw()` é chamada toda vez que o *display* precisa ser redesenhado, a função `reshapeHandler()` é chamada quando o tamanho do *display* é alterado e a função `hotKeys()` é requisitada todas vez que alguma tecla é pressionada, as setas no teclado são usadas para rotacionar o desenho. O resultado dessa sequência de operções é visto a seguir.

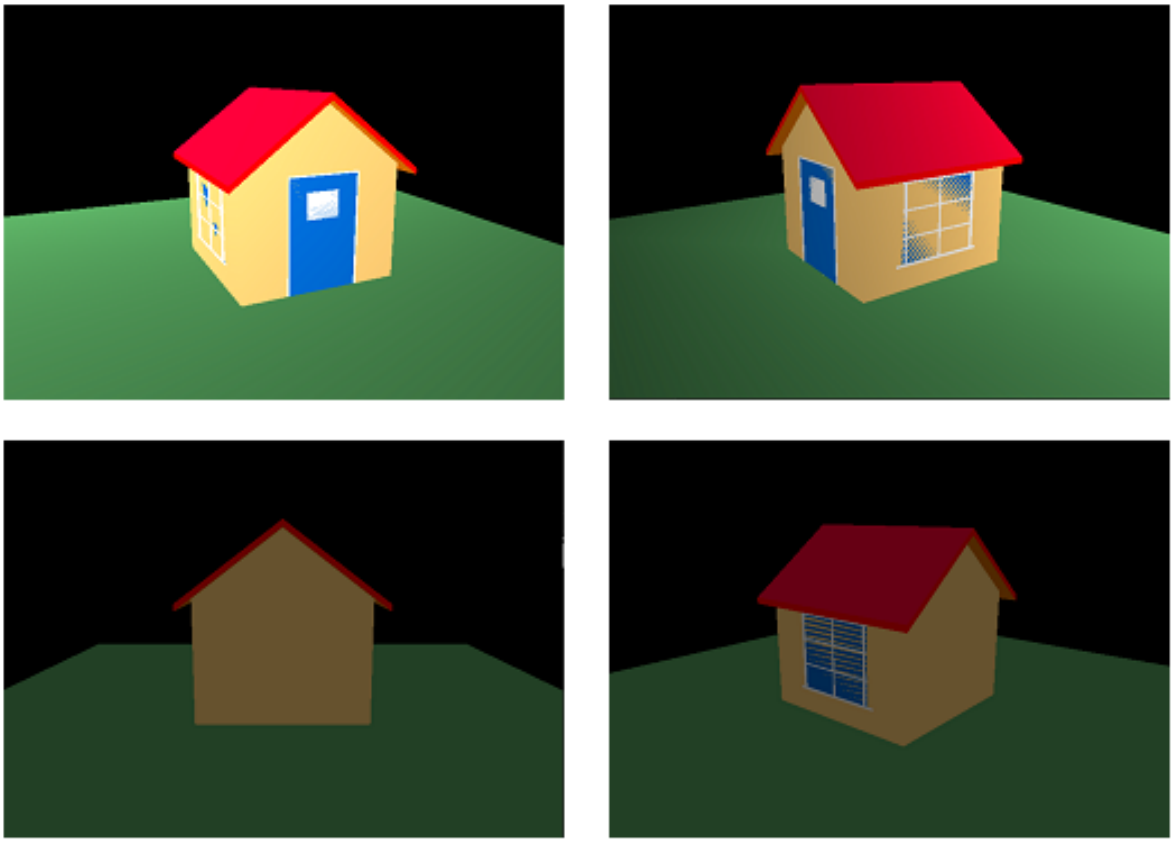

Figura 12: Iluminação com reflexão: capturas de tela.

4 Códigos-fontes

O código-fonte está disponível no link abaixo.
`https://github.com/DiegoFr75/Luz-e-sombra/blob/master/main.cpp`

Referências

[1] BEAN, S. E. P. C.; KOZAKEVICH, D. N. *Álgebra Linear I*. 2. ed. Florianópolis - SC, Brasil: FSC/EAD/CED/CFM, 2011.

[2] Fábio Franco. *Resumo sistema de coordenadas*. 2018. Disponível em `http://docente.ifrn.edu.br/fabiofranco/disciplinas/elementos-de-fisica/1a-unidade/2o-momento/resumo-sistema-de-coordenadas`. Acesso em 4 de dezembro de 2019.

[3] Ulysses Sodré. *Geometria plana e espacial - Vetores no plano cartesiano*. 2008. Disponível em `http://www.uel.br/projetos/matessencial/geometria/vetor2d/vetor2d.htm`. Acesso em 4 de dezembro de 2019.

[4] Daniel Duarte Abdala. *Sistema de coordenadas homogêneas*. 2019. Disponível em `http://www.facom.ufu.br/~abdala/GBC204/05_sistemasDeCoordenadas.pdf`. Acesso em 4 de dezembro de 2019.

[5] Charles Novaes de Santana, José Garcia Vivas Miranda. *OpenGL na Computação Gráfica*. 2014. Disponível em `https://www.passeidireto.com/arquivo/24346621/tutorial-1-3-open-gl`. Acesso em 4 de dezembro de 2019.

[6] Joey DeVries. *OpenGL – Basic Lightning*. 2015. Disponível em `https://learnopengl.com/Lighting/Basic-Lighting`. Acesso em 4 de dezembro de 2019.

[7] Tom Dalling. *Modern OpenGL 06 – Diffuse Point Lighting*. 2013. Disponível em `https://www.tomdalling.com/blog/modern-opengl/06-diffuse-point-lighting/`. Acesso em 4 de dezembro de 2019.