Fernanda Tostes Marana - no.usp: 4471070
Matheus Aparecido do Carmo Alves - no.usp: 9791114
Sylviane da Silva Vitor - 10889564

Projeto da Disciplina Computação Gráfica: Sistema Interativo de Visualização de objetos 3D

Professora Doutora Agma J. M. Traima
Universidade de São Paulo - USP
Instituto de Ciências Matemáticas e de Computação - ICMC
Bacharelado em Ciências de Computação

São Carlos, SP
15 de Junho de 2018

Sumário

1 Introdução

Após a criação de cenas e objetos tridimensionais o próximo passo é realizar sua apresentação. No entanto, a geração de imagens de cenas 3D sob uma superfície plana, como uma tela de exibição do computador, não é uma tarefa simples. No mundo real, seres humanos são capazes de notar a relação espacial entre os objetos só de olhar para eles, devido a diversas ferramentas de percepção biológicas, como:

- visão estereoscópica: dois olhos fornecem imagens ligeiramente diferentes;
- parallax: ao mexer a cabeça, objetos mais próximos parecem se mover mais do que os distantes;
- alojamento: Quanto a visão humana se concentra em um objeto próximo ou distante, as lentes em seus olhos mudam de forma física, proporcionando uma pista sobre o quão longe o objeto está;
- perspectiva aérea: Como a luz é espalhada aleatoriamente pelo ar, objetos distantes parecem ter menos contraste de objetos próximos.

Em computação gráfica, muitos algoritmos e técnicas são utilizados a fim de imitar tais ferramentas, muitas vezes combinando algumas delas. Assim, uma cena sintética pode ser construída a partir de vários fatores que determinam como será projetada em um plano. Alguns dos parâmetros utilizados pertencem à câmera, como por exemplo, sua posição, orientação, tipo de projeção efetuada e consequentemente dos "planos de recorte" (clipping planes). Outros fatores determinísticos se encontram no rendering dos objetos, isto envolve a iluminação, posição, determinação da superfície visível, tonalização, sombras e transparência.

Além disso, usando as ferramentas de navegação e visualização 3D, é possível navegar através de um desenho, orbitando-o, aplicando zoom e girando ao redor de um modelo 3D. O objetivo deste projeto é apresentar uma interface que auxilie o usuário na interação com objetos tridimensionais num plano bidimensional, aplicando as transformações e os métodos de rendering vistos em classe, de maneira intuitiva e fácil de manusear. Este relatório apresentará todas as técnicas implementadas, as decisões feitas pelo grupo durante o processo de desenvolvimento e os conceitos aplicados durante o trabalho.

O relatório está estruturado como segue: na Seção 2, encontram-se instruções de compilação e observações acerca do ambiente de compilação utilizado nos testes. Na Seção 3, há algumas considerações de implementação. Na Seção 4, está descrita a implementação da interface, correspondente à Etapa 1 de execução do projeto. Na Seção 5, a

implementação das transformações de escala, rotação, translação e mirror. Na Seção 6, estão descritas as técnicas de rendering. Na Seção 7, são apresentados os resultados e na Seção 8 é apresentada uma breve conclusão.

2 Instruções e Ambiente de Compilação

A implementação dos algoritmos foi feita na linguagem C de programação, utilizando como principais suportes a API OpenGL e biblioteca, FreeGLUT uma alternativa de software livre à biblioteca do OpenGL Utility Toolkit (GLUT). A primeira foi utilizada como suporte nos métodos de tonalização e transformações e a segunda foi utilizada nas definições de interface gráfica do trabalho, para criar e gerenciar janelas contendo contextos OpenGL, além de funções para tratamento de mouse e teclado.

Para compilação, é necessário somente a correta instalação das bibliotecas mencionadas. Sua compilação pode ser feita via terminal Linux pelo comando (considerando que o usuário se encontra na pasta local):

```
1        $ make all
```

E em seguida execução de:

```
1        $ make run
```

O ambiente usado para testes possui as seguintes características de arquitetura:

(1) Sistema Operacional: Linux - Ubuntu 16.04 LTS;

(2) Memória 3,8GiB;

(3) Processador: Itel Core i3-3110M CPU @ 2.40GHz x 4

(4) Gráficos: Intel Ivybridge Mobile

(5) Tipo de sistema: 64-bits

(6) Disco: 106,2 GB

Os testes foram rodados com o mínimo de processos paralelos à aplicação possível.

3 Considerações de Implementação

Em vista da quantidade de funcionalidades necessárias ao sistema, optou-se pela modularização de código, a fim de melhorar sua legibilidade. Desta forma, foram construídos os seguintes módulos:

- Button - Contendo métodos da estrutura Button, criada para definir os botões da interface (ver Seção 4);
- Transform - Contendo métodos para realizar as transformações (ver Seção 5)
- Interface - Contendo métodos de inicialização da interface (ver Seção 5)

Desta forma, no programa principal T2 restaram apenas inicializações e as definições de tonalização e iluminação (ver Seção 6), para que a janela de visualização seja criada e atualizada seguindo a estrutura definida pela *API OpenGL*.

3.1 Interação via teclado

Além das transformações, em que o usuário pode rotacionar, escalar, transladar e espelhar os objetos utilizando os botões, também foram implementadas interações via teclado.

Para visualizar os modelos de tonalização, o usuário precisa trocar do modo em modelos de arame para objetos sólidos, o que pode ser feito pressionando a tecla "S". Para visualizar trocar o modelo de tonalização mostrado, o usuário deve pressionar a tecla *"T"*. É possível aplicar as transformações normalmente utilizando os botões. Para finalizar a execução e fechar a janela, basta pressionar a tecla *"ESC"*.

Para alterar a projeção entre ortográfica ou perspectiva, o usuário precisa pressionar a tecla *"P"*.

4 Interface

A interface é constituída de uma composição de portas de visão, uma para cada objeto e conjunto de botões. Desta forma, conseguimos manipular de forma independente os objetos e suas transformações implicam que estes não ultrapassem os limites de sua porta de visão.

Também foram colocadas linhas horizontais de forma a criar a melhor visualização de separação dos objetos e seus respectivos botões. Além disso, existem instruções de interações com o teclado dispostas na parte superior. A interface inicial contendo objetos em modelo de arame pode ser vista na Figura 1 e a interface contendo objetos sólidos na Figura 2. As seguintes funções foram usadas para definir a interface:

- setMaterial - Definir propriedades do material para se adaptar aos tipos de reflexão: difusa ou especular e propriedades de luminosidade, utiliza a função glMaterialfv para setar os vetores.
- colorObject - Definir cor dos objetos, caso selecionados, utiliza a função glColor3f.
- drawObject - Definir objetos sólidos ou em modelo de arame e desenhar usando as funções: glutSolidCube, glutSolidSphere, glutSolidTeapot e glutWireCube, glutWireSphere, glutWireTeapot.
- Draw3DWire e Draw3DSolid - Seta porta de visão utilizando a função glViewport, seta posição do observador utilizando a função gluLookAt; definir as funções de transformação (ver Seção 5) e desenhar o objeto utilizando a função drawObject.
- Draw2D - Seta porta de visão utilizando a função glViewport, desenha os botões utilizando a função ButtonDraw e desenha as linhas horizontais definindo sua posição de início e fim utilizando a função glVertex2f.

Para definição dos botões, foi criada a estrutura *Button*, contendo seu nome (a ser mostrado na tela), posição (x, y), altura h e largura w, *flags* de operação e *id* e sua função de *callback* para quando pressionado.

```
struct Button {
    char *label;
    int x, y, w, h;
    int pressed, highlighted, operation, id;
    ButtonCallback callbackFunction;
};
```

Em nossa interface é definida uma lista de botões, inicializada no método principal de T2. As seguintes funções auxiliares foram criadas para o tratamento dos botões:

- ButtonClickTest - Verificar se botão está sendo clicado, utiliza função auxiliar isbetween que apenas verifica se posição do botão e do mouse coincidem;
- ButtonRestore - Retirar highlight do botão, resetar flags e chamar função de callback quando não está mais sendo clicado;
- ButtonSetPress - Setar highlight e flags quando botão é clicado;
- ButtonUnder - Setar highlight quando mouse está sobre botão, verificar se está sendo clicado através da função ButtonClickTest e redesenhar utilizando a função glutPostRedisplay caso necessário;

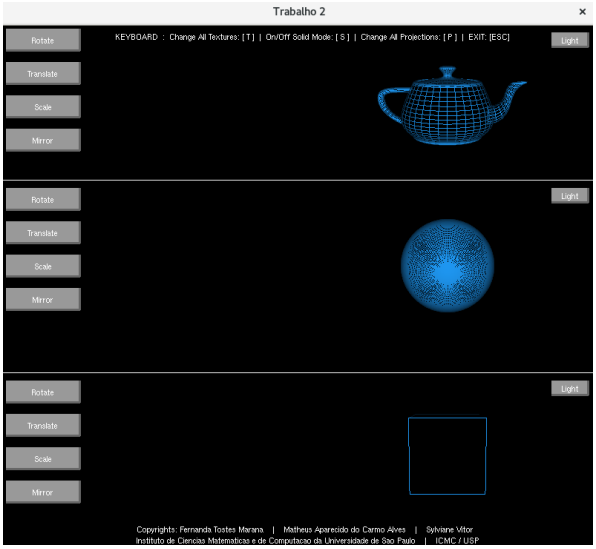

Figura 1 – Interface com objetos em modelo de Arame (*Wire Objects*).

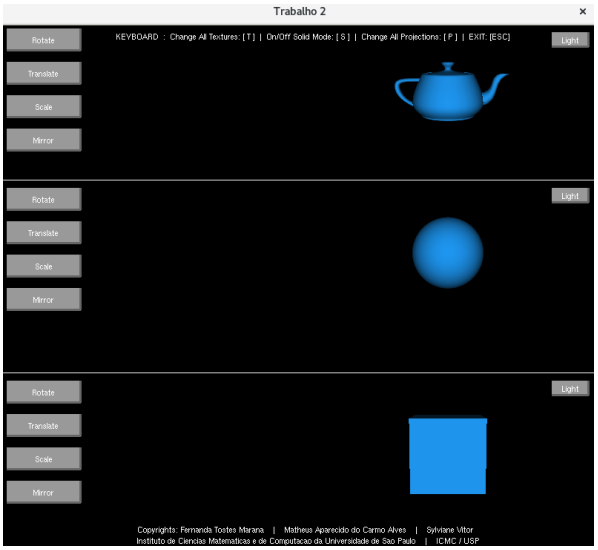

Figura 2 – Interface com objetos em modelo Solidos (*Solid Objects*).

- ButtonDraw - Desenhar botões (retângulos) e centralizar nome, utiliza as funções glColor3f para definir cores, inicializa o retângulo do botão setando seus vértices com a função glVertex2i e inicializa os textos na tela;
- init - inicializar lista de botões ButtonList;
- Font - Utiliza as funções glutBitmapCharacter e glRasterPos2i para definir a posição do texto e desenhar caracteres do nome do botão (label);
- CreateButton - Criar botões.

5 Transformações

Transformações de coordenadas e transformações geométricas em *3D* são de grande utilidade para atividades de manipulação, visualização e construção de imagens gráficas tridimensionais. Essas transformações são formadas pela composição das transformações primárias de translação, de variação de escala e de rotação. Cada uma dessas transformações, tanto em *3D* quanto em *2D*, são representadas por uma matriz de transformação.

Os pontos também conhecidos como âncora (*anchor*) contém a informação digital. Na memória estão armazenadas as coordenadas de cada ponto e as equações matemáticas que representam como eles se ligam aos demais. Assim, as transformações aplicadas sobre eles por meio da matriz resultam em novas informações a serem contabilizadas.

Tendo os pontos são expressos em Coordenadas Homogêneas, todas as 3 transformações principais (translação, rotação e escala) podem ser tratadas como multiplicações. Em coordenadas homogêneas, adicionamos uma terceira coordenada ao ponto. Logo, em vez de um ponto ser representado por um par de valores (x, y), ele é representado por uma tripla (x, y, W), que ao se homogeneizá-la, isto é dividir todas coordenadas por W, obtém-se um ponto da forma $(x, y, 1)$. Neste sistema de visualização foram implementadas as operações de escala, rotação, translação e *mirror*.

5.1 Rotação

A rotação foi implementada utilizando a função *glRotated* que multiplica a matriz da imagem pela matriz de rotação, nos eixos determinados por seus parâmetros. Optamos por rotacionar no eixo y, para melhor visualização. Desta forma, a cada clique do botão *"Rotate"*, a imagem é rotacionada de forma que completa uma volta em sua porta de visão e retorna à posição original. O resultado da rotação dos três objetos em modelo de arame é mostrado na Figura 3 e com objetos sólidos na Figura 4.

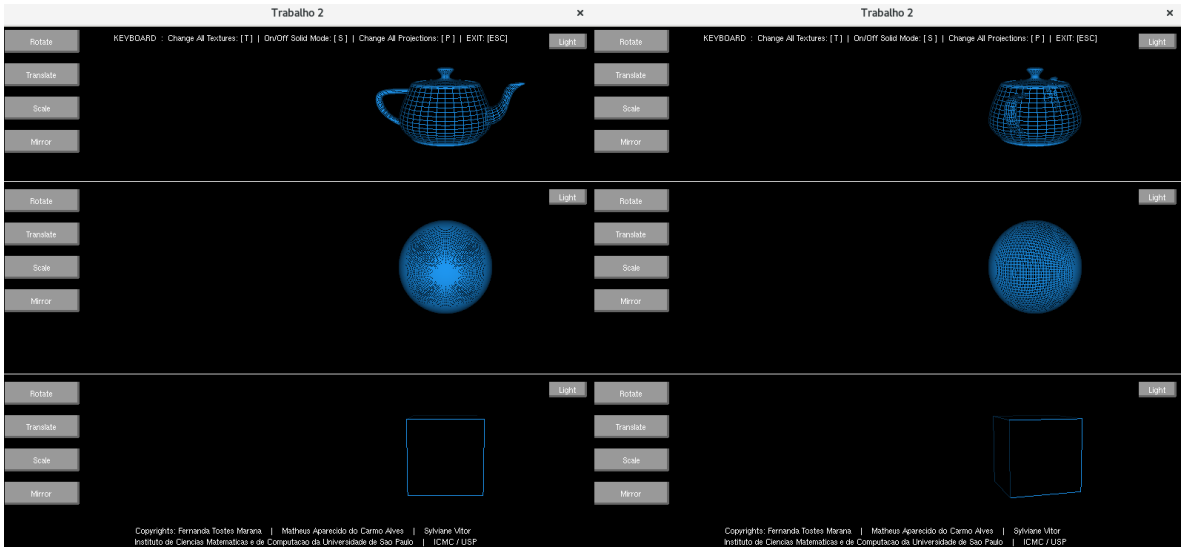

Figura 3 – Modelos de arame antes e após rotação em y.

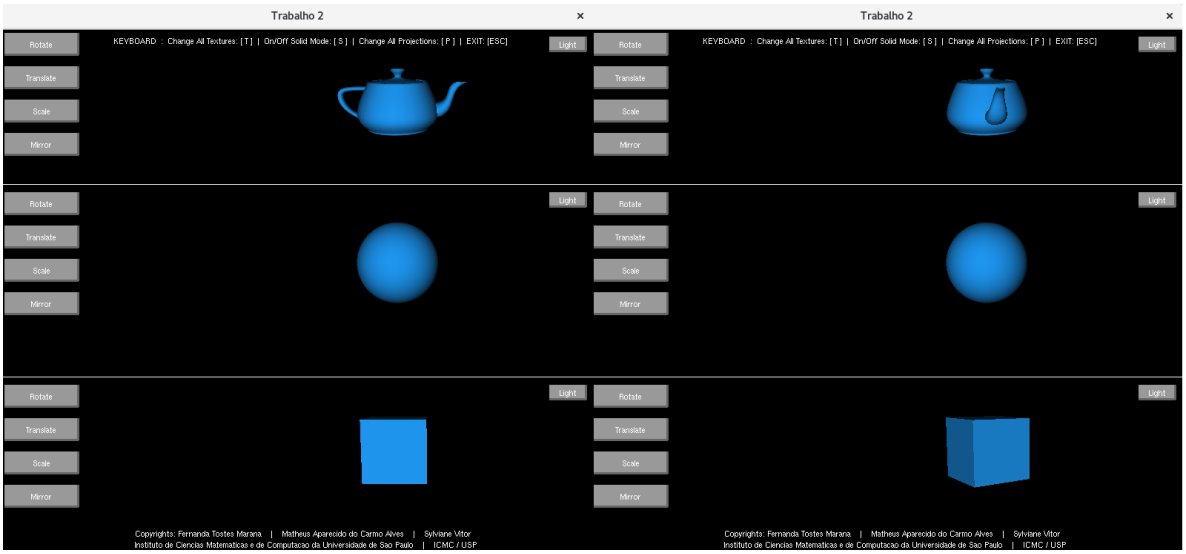

Figura 4 – Modelos sólidos antes e após rotação em y.

5.2 Translação

A translação foi implementada utilizando a função glTranslated que multiplica a matriz da imagem pela matriz de translação. Desta forma, a cada clique do botão *"Translate"*, a imagem é transladada de forma que completa uma volta em sua porta de visão e retorna à posição original. O resultado da translação dos três objetos em modelo de arame é mostrado na Figura 5 e com objetos sólidos na Figura 6.

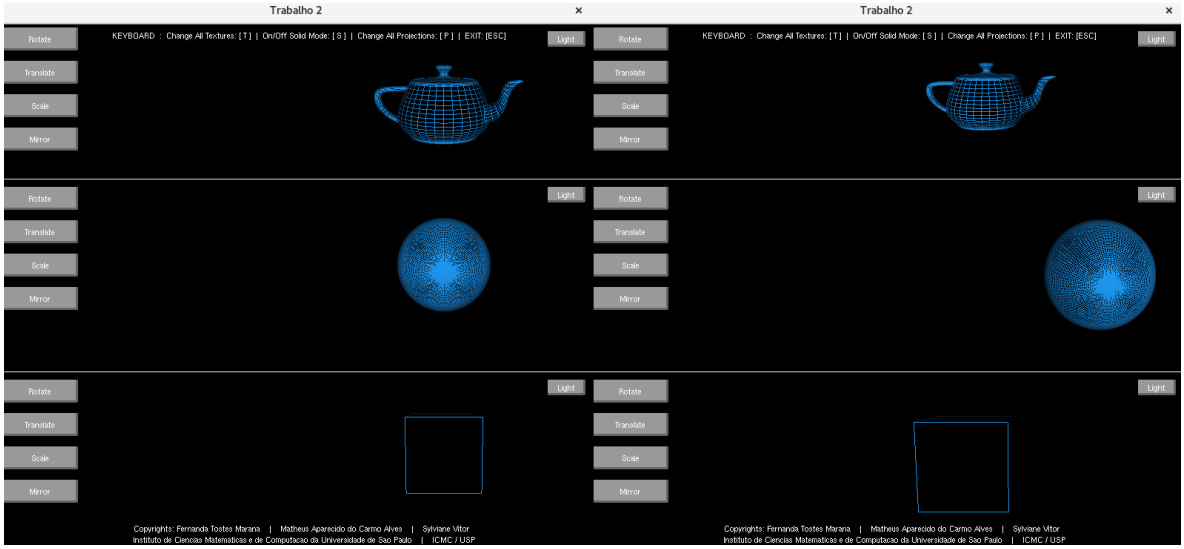

Figura 5 – Modelos de arame antes e após translação.

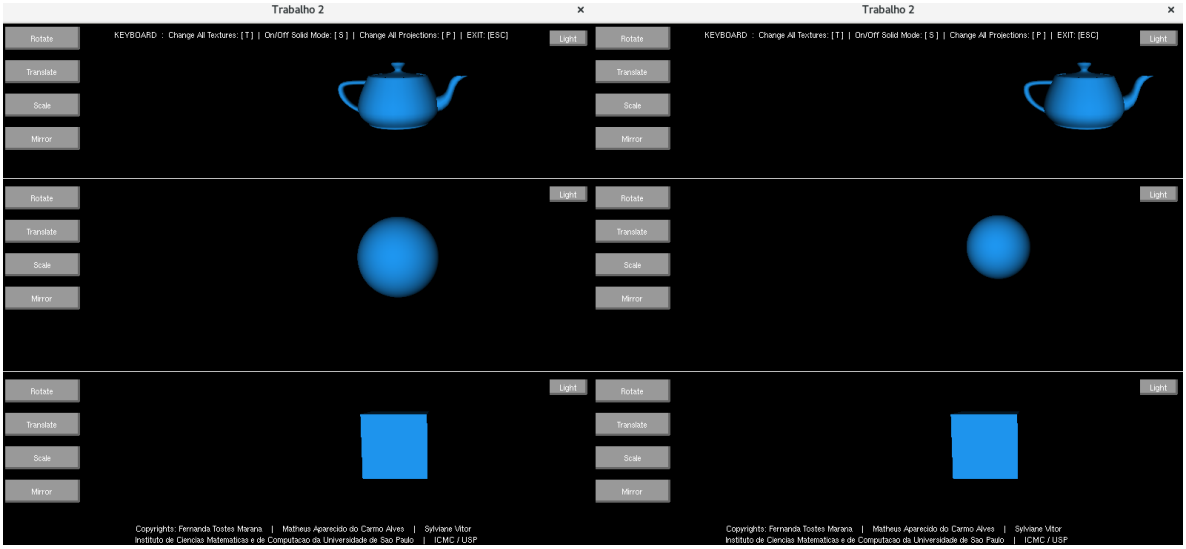

Figura 6 – Modelos sólidos antes e após translação.

5.3 Escala

A escala foi implementada utilizando a função *glScalef* que multiplica a matriz da imagem pela matriz de escala, nos eixos determinados por seus parâmetros. Optamos por escalar uniformemente a imagem, para melhor visualização. Desta forma, a cada clique do botão *"Scale"*, a imagem é escalada e tem seu tamanho reduzido até um limiar e então retorna ao tamanho original. O resultado da escala dos três objetos em modelo de arame é mostrado na Figura 7 e com objetos sólidos na Figura 8.

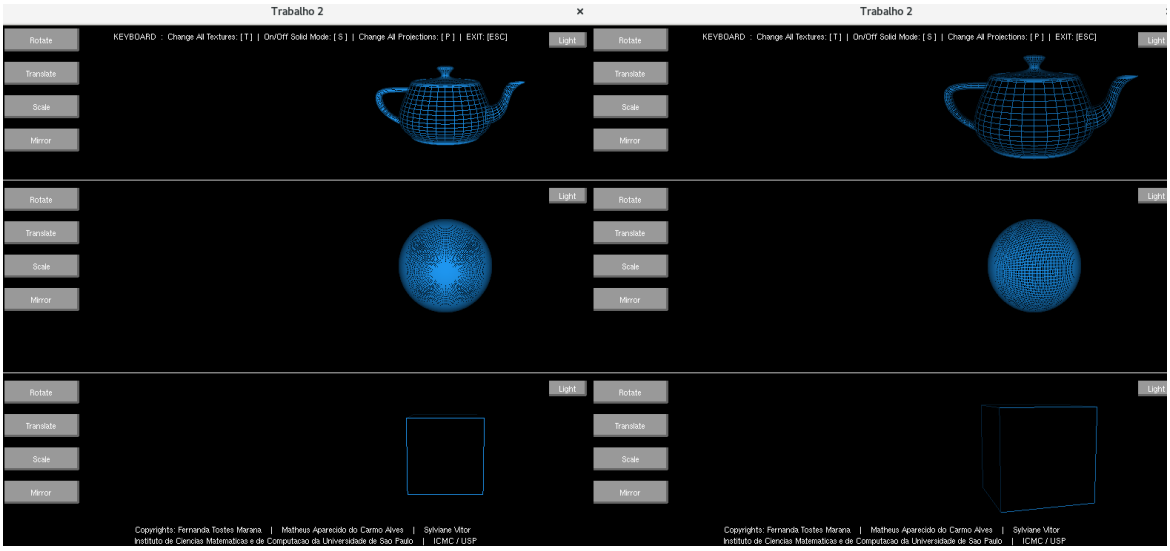

Figura 7 – Modelos de arame antes e após escala uniforme.

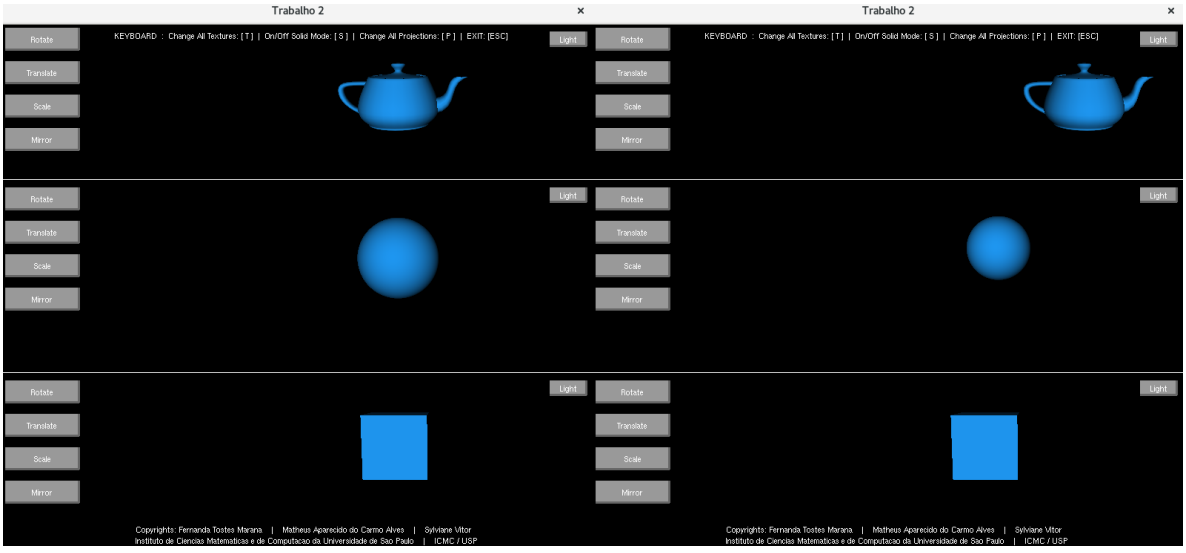

Figura 8 – Modelos sólidos antes e após escala uniforme.

5.4 Espelhamento

O espelhamento trata-se de uma rotação nos eixos principais. Desta forma, foi implementado utilizando a função *glRotated* que multiplica a matriz da imagem pela matriz de rotação, nos eixos determinados por seus parâmetros. Optamos por rotacionar no eixo y, para melhor visualização. Desta forma, a cada clique do botão "*Mirror*", a imagem é espelhada. O resultado do espelhamento dos três objetos em modelo de arame é mostrado na Figura 9 e com objetos sólidos na Figura 10.

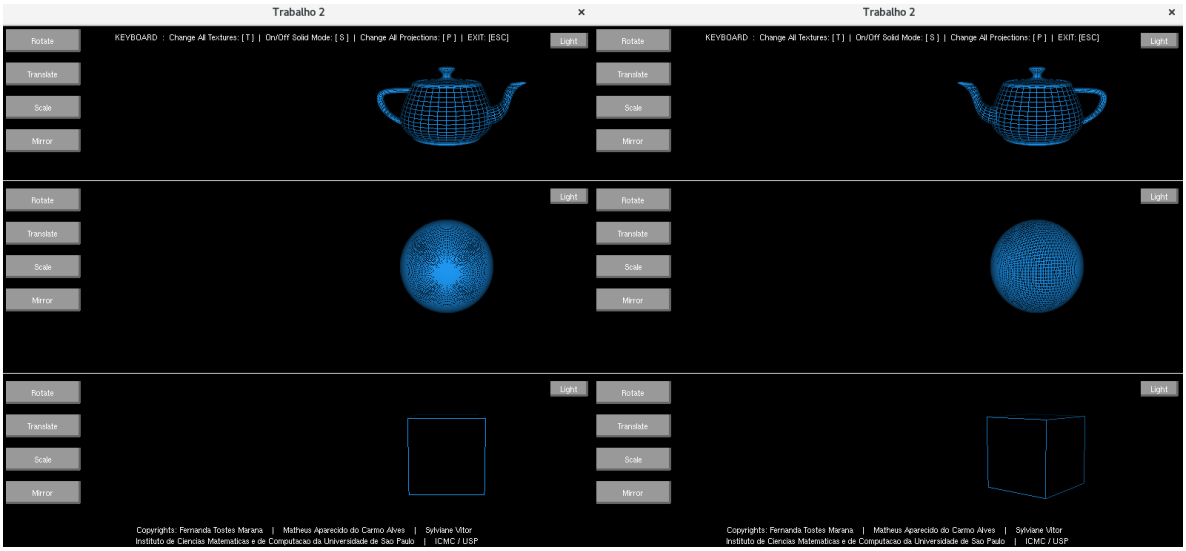

Figura 9 – Modelos de arame antes e após espelhamento.

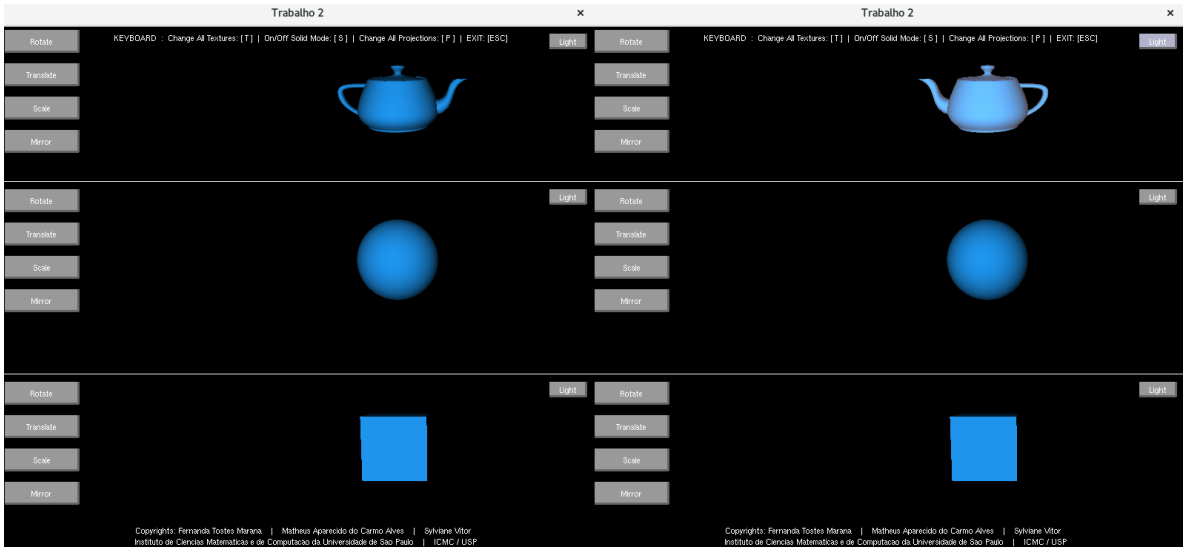

Figura 10 – Modelos sólidos antes e após espelhamento.

6 Rendering

Rendering é o processo de adicionar realismo em uma imagem a partir de uma descrição de cena. Pode-se adicionar efeitos de textura, luz, posição relativa dos objetos, entre outros. No Sistema Interativo, tratamos a iluminação e tonalização dos objetos, descritos a seguir.

6.1 Iluminação

Os modelos de luz são usados para simular a interação dos componentes da cena com os raios de luz. Em nosso sistema a iluminação é definida por uma fonte de luz, utilizando a função *glLightfv* em que uma luz é especificada e sua posição relativa é definida.

Para visualizar as diferentes iluminações, definidos a partir da função *setMaterial* que seta as propriedades do material para que ocorra a iluminação, o usuário deve clicar no botão *"Light"*. Os modos disponíveis são:

1. Reflexão apenas difusa;
2. Reflexão difusa e especular com baixo brilho;
3. Reflexão difusa e especular com alto brilho;
4. Emissão difusa.

Ao clicar no botão "Light" o usuário vai alternando entre os modos de reflexão; como mostram as Figuras 11 e 12 a seguir.

6.2 Tonalização

O primeiro modelo de tonalização implementado foi *Flat*, que considera somente os termos de luz ambiente e reflexão difusa do modelo de iluminação; uma única fonte de luz colocada no infinito na direção de visão do observador. Além disso, ele calcula uma normal para cada face e aplica o modelo de iluminação, de forma que toda face é associada a uma cor única. Foi utilizada a função *glShadeModel(GL_FLAT)* que implementa o modelo. Seu resultado nos objetos sólidos é mostrado na Figura 13, que mostra seu aspecto facetado, que acentua as arestas.

O segundo modelo de tonalização implementado foi *Gouraud*, que considera as componentes ambiente, difusa e especular. Nele o modelo de iluminação é aplicado para calcular as intensidades nos vértices do polígono e os valores obtidos são interpolados para

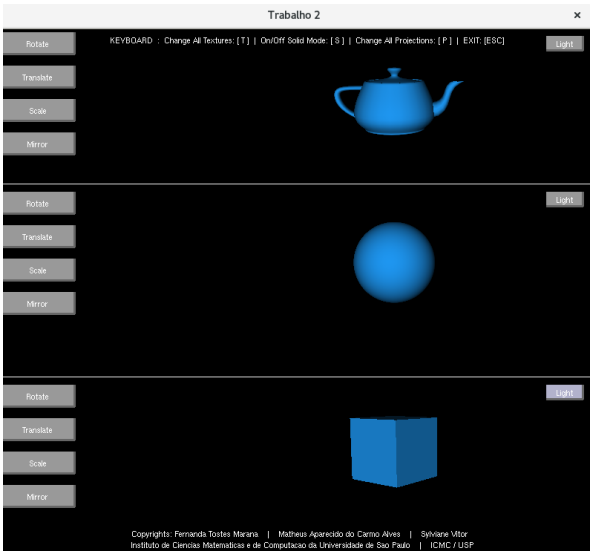

Figura 11 – Modelos sólidos com reflexão difusa.

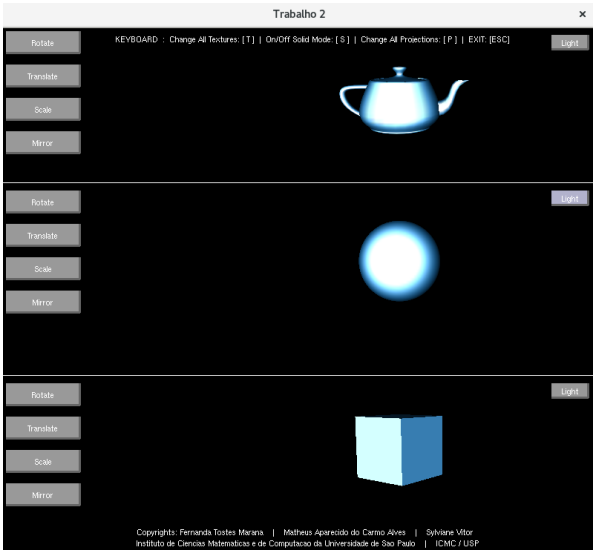

Figura 12 – Modelos sólidos com reflexão difusa e especular com alto brilho.

determinar o valor de intensidade de cada pixel nas arestas e no interior do polígono. Foi utilizada a função *glShadeModel(GL_SMOOTH)* implementa o modelo. Seu resultado nos objetos sólidos é mostrado na Figura 14, que mostra sua suavização em relação ao *Flat*, porém não captura bem os *highlights*.

O terceiro modelo de tonalização implementado foi o modelo de *Phong*, que, assim como *Gouraud*, considera as componentes ambiente, difusa e especular. Ele consiste em calcular as normais nos vértices do polígono interpolando os valores obtidos para determinar a normal em cada pixel do polígono, incluindo as arestas e o interior. Com isso é possível se obter o modelo de iluminação de cada ponto pela descrição da superfície como uma combinação da reflexão difusa e reflexão especular. Adicionalmente é aplicado também a luz ambiente espalhada por toda cena que o objeto está posicionado. Assim

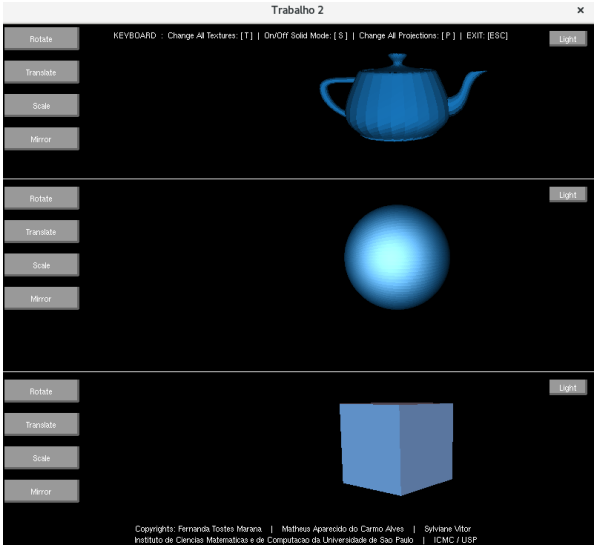

Figura 13 – Modelos sólidos com reflexão difusa.

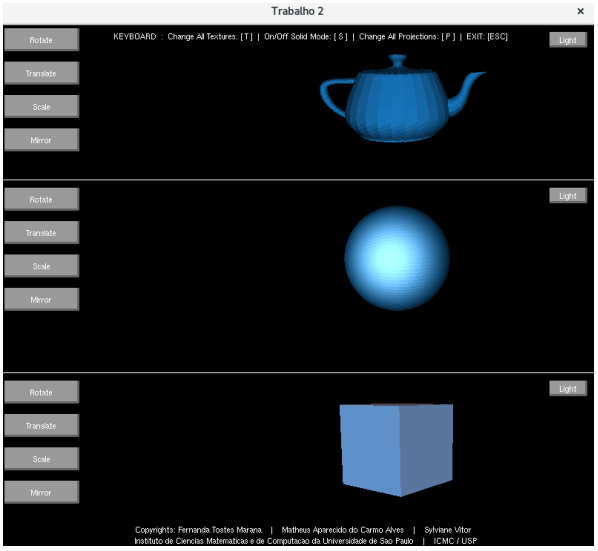

Figura 14 – Modelos sólidos com reflexão difusa.

obtém-se a seguinte fórmula para encontrar a iluminação de *Phong*.

$$I_{phong} = I_a k_a + I_1 k_d cos(\theta) + I_1 k_s (R * S)^n \tag{6.1}$$

, sendo I_a a Intensidade da luz ambiente, k_a o Coeficiente de reflexão do ambiente, I_1 a Intensidade de luz emitida pela fonte de luz e incidente na superfície, k_d o Coeficiente de reflexão difusa, θ o ângulo entre a direção da luz L e a normal N da superfície, k_s o Coeficiente de reflexão especular, R a Direção de reflexão que depende do vetor diretor do ponto da superfície até a fonte de luz e da normal da superfície, S a Linha de Visão e n o Brilho.

O *Phong* também pode ser aplicado a diversas fontes de luz tendo cada uma componentes especular e de reflexão próprias. Esse tipo de iluminação também apresenta

resultados melhores que o *Gouraud* para polígonos complexos, uma vez que ao sombrear um vetor normal, utiliza interpolação linear, método conhecido por fazer ajuste de curvas para construir novos pontos de dados dentro do intervalo de pontos de dados conhecidos. O resultado é uma aparência mais natural e marcada pela discreta transição das cores.

Como consequência desse resultado agradável, a operação possui um alto custo computacional.

Para o cálculo da normal de uma superfície, utiliza-se três dos vértices que à compõem (vértice A, vértice B, vértice C). Realiza-se duas diferenças de dois vértices diferentes e obtém-se dois vetores distintos V_1 e V_2. Com isso, pode-se obter as posições i, j, k do vetor normal aplicando-se as seguintes equações com os vetores V_1 e V_2:

$$N_i = V_1 j * V_2 k - V_1 k * V_2 j \tag{6.2}$$

$$N_j = V1i * V2k - V1k * V2i \tag{6.3}$$

$$N_k = V1i * V2j - V1j * V2i \tag{6.4}$$

Em aplicação, foram implementados esse cálculo de normais sobre um cubo criado utilizando os vértices. Na figura 15, apresenta-se o resultado obtido.

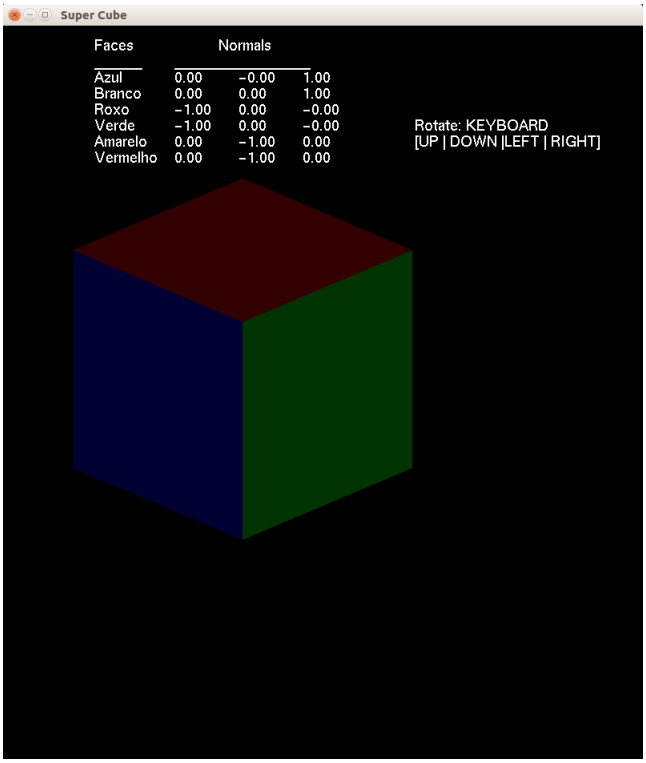

Figura 15 – Modelos sólidos com a aplicação do Phong.

6.3 Projeções

A projeção dos objetos tridimensionais no plano de observação em nosso sistema é definida por um plano de projeção utilizando as funções *glOrtho*, que define o modo de projeção ortográfica e *glFrustum* que define o modo de projeção perspectiva. Para visualizar as diferentes projeções, definidos a partir da função *switchProjection* que alterna o modo de projeção, o usuário deve pressionar a tecla *"p"*.

7 Resultados

Os resultados obtidos neste trabalho estão fortemente relacionados à vantagens encontradas na utilização de API's Gráficas para modelagem de cenas e aprendizado de Computação Gráfica.

A utilização do *OPEN GL* para a criação das cenas e para a realização das operações solicitadas criou uma abordagem mais aplicativa e lógica para o desenvolvimento do projeto. Após claro entendimento semântico das características ligadas ao trabalho e de todo o assunto necessário para a aplicação que compõe a base da disciplina disciplina (operação sobre objetos, parâmetros importantes para iluminação e posicionamento no cenários, etc), por meio de estudo individual e/ou das aulas os alunos conseguiram facilmente aplicar o conhecimento adquirido de forma natural e direta. Adicionalmente, está facilidade criada pela utilização desta plataforma foi o que possibilitou a habituação dos alunos a problemas e implementações reais da computação gráfica, assim como o desenvolvimento e maturação de um raciocínio lógico para assuntos deste âmbito (diferente do raciocínio desenvolvidos em outros cursos/matérias).

Desta forma, os alunos, após finalização do projeto, possuem maior capacitação, conhecimento e maturidade sobre o assunto, algo de extrema relevância para a formação profissional e acadêmica, atingindo o objetivo principal do curso.

8 Discussão

O Sistema Interativo de Visualização de Objetos *3D* proposto possibilita interações tanto com teclado, quanto com botões, melhorando a experiência do usuário durante sua navegação. Os principais desafios encontrados durante seu desenvolvimento envolveram, principalmente, a melhoria da apresentação e garantia de que todas as operações nos objetos ocorressem dentro dos limites esperados.

Seu objetivo é aplicar os conceitos vistos em sala para objetos *3D* como transformações, *rendering* e projeções. Além disso, é possível verificar que o pipeline de visualização foi reproduzido, desde a etapa de definição das posições de câmera, observador, plano de projeção, até a fase de *rendering*.

A *API* multiplataforma e multi linguagem *OPENGL* permitiu a criação de um ambiente simples, porém com diversas funcionalidades de forma prática. A definição das funções utilizadas descrevia os modelos aprendidos ao longo da disciplina, possibilitando seu uso de forma consciente para fazer as escolhas adequadas para a melhor apresentação.

A utilização de diferentes portas de visão possibilitou maior independência na manipulação dos objetos, garantindo seu comportamento dentro do esperado diante das operações de transformação.

A aplicação de diferentes texturas para iluminação possibilitou melhor visualização das diferenças entre os modelos *Flat* e *Gouraud*, possibilitando avaliar seu desempenho no que diz respeito à aspecto e realismo.