Operações com Dados Geométricos e Posicionamento do Observador

Rafael Bonfim
rafael.queiroz@ufop.edu.br

DECOM – ICEB – UFOP – 2025.2

Conteúdo

Objetivos da Aula

- Dominar a manipulação de vértices e arestas através do uso de pilhas de matrizes (`Matrix Stack`).
- Entender como o **sistema de coordenadas** é transformado para criar modelos hierárquicos complexos.
- Aprender a configurar a **Matriz de View** para posicionar o observador (câmera) na cena 3D.
- Preparar a base para as próximas aulas sobre Projeção e Iluminação.

O Contexto: Seleção da Matriz Ativa

Onde as Transformações Acontecem

No OpenGL Fixed Pipeline, as transformações não agem diretamente nos vértices, mas sim em uma **matriz ativa**. A função `glMatrixMode` define qual matriz será modificada.

- `glMatrixMode(GL_MODELVIEW)`:
  - Matriz Ativa: Combina Modelagem (`Model`) e Visualização (`View` / Câmera).
  - Uso: Transformar o objeto e posicionar a câmera. **É a matriz que mais usamos**.
- `glMatrixMode(GL_PROJECTION)`:
  - Matriz Ativa: Matriz de Projeção.
  - Uso: Definir o tipo de lente da câmera (Perspectiva ou Ortográfica).
- **Atenção: Limpeza da Matriz**
  glMatrixMode(GL_MODELVIEW); // 1. Seleciona a Matriz
  glLoadIdentity(); // 2. Limpa (Matriz Identidade)
  gluLookAt(...); // 3. Aplica View Matrix
  glRotatef(...); // 4. Aplica Model Matrix

Funções de Transformação (*OpenGL Fixed Pipeline*)

Chamadas Simples, Poder Complexo

Essas funções **multiplicam** a Matriz Atual pela matriz de transformação correspondente (Translação, Rotação ou Escala).

- **Translação (Multiplica por Matriz T):**
  - `void glTranslated(GLdouble x, GLdouble y, GLdouble z);`
  - `void glTranslatef(GLfloat x, GLfloat y, GLfloat z);`
  - Move o objeto por (x, y, z).
- **Rotação (Multiplica por Matriz R):**
  - `void glRotated(GLdouble angle, GLdouble x, GLdouble y, GLdouble z);`
  - `void glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);`
  - Roda o objeto em torno de um vetor (x, y, z) por um ângulo em graus.

Funções de Transformação (*OpenGL Fixed Pipeline*)

- **Escala (Multiplica por Matriz S):**
  - `void glScaled(GLdouble x, GLdouble y, GLdouble z);`
  - `void glScalef(GLfloat x, GLfloat y, GLfloat z);`
  - Escala o objeto por fatores (x, y, z) em relação à origem do sistema de coordenadas atual.

Composição: Ordem de Aplicação das Transformações

Transformação (M) × Vetor (v)

No OpenGL, o vetor v é pré-multiplicado pela matriz M: $v' = M \cdot v$. A ordem de aplicação é **inversa** à ordem de chamada no código.

Ordem Global (T → R)

- Objetivo: Rotacionar o objeto em torno da origem do Mundo.
- Código: Rotação é chamada **antes** da Translação (Primeiro R, depois T).
- `glRotatef(...);`
  `glTranslatef(...);`
- Resultado: $M = T \cdot R$. O ponto é primeiro rotacionado (R) e depois transladado (T).

Ordem Local (R → T)

- Objetivo: Rotacionar o objeto em torno do seu próprio centro local.
- Código: Translação é chamada **antes** da Rotação (Primeiro T, depois R).
- `glTranslatef(...);`
  `glRotatef(...);`
- Resultado: $M = R \cdot T$. O ponto é primeiro transladado (T) e depois rotacionado (R).

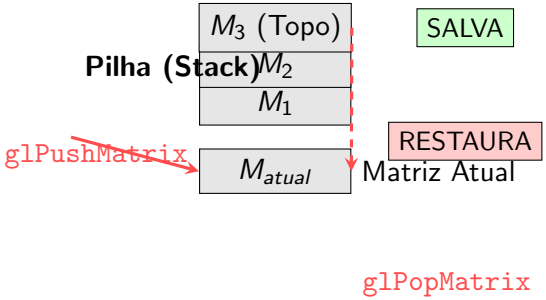
O Conceito de Matrix Stack

Transformações Independentes

A Pilha de Matrizes (`Matrix Stack`) permite que você aplique transformações complexas a um objeto **sem afetar** as transformações de outros objetos na cena.

Funções do Stack:

- `glPushMatrix()`:
  Salva (Copia) a matriz atual e a coloca no topo da pilha.
- `glPopMatrix()`:
  Restaura a matriz do topo da pilha para a matriz atual, desfazendo todas as transformações intermediárias.



Exemplo Prático: Desenhando Cenas Complexas

A Base da Modelagem Hierárquica

A pilha é essencial para o desenho de objetos compostos (como um Braço Robótico ou o Sistema Solar), onde a transformação de um componente depende do seu "pai".

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(...); // Configura a camera (Matriz Base)

// --- Desenha o SOL (Raiz da Hierarquia) ---
glPushMatrix();
glRotatef(rot_sol, 0, 1, 0);
glutSolidSphere(1.0);
glPopMatrix();

// --- Desenha o PLANETA (Depende do Sol) ---
glPushMatrix();
glRotatef(orb_angle, 0, 1, 0); // Orbita o Sol
glTranslatef(5.0, 0.0, 0.0);    // Posiciona em relacao ao Sol

glPushMatrix();
glRotatef(rot_planeta, 0, 1, 0); // Rotaciona em torno de si mesmo
glutSolidSphere(0.5);
glPopMatrix();

glPopMatrix(); // Restaura a matriz após o Planeta
```

Hierarquia: O Papel de `glPopMatrix()` (Cleanup)

Objetivo do `glPopMatrix()`: Isolar Objetos

A chamada `glPopMatrix()` é essencial para garantir que as transformações aplicadas a um objeto (o Planeta) não afetem o próximo objeto a ser desenhado.

Sequência Crítica (Planeta):

1. `glPushMatrix()`: Salva a Matriz do Sol/Órbita.
2. `glRotatef(orb_angle)`: Transforma o SC para a **Órbita**.
3. `glTranslatef(5.0)`: Transforma o SC para a **Posição Final** do Planeta.
4. `glPopMatrix()`: Restaura o SC para o estado **anterior ao Passo 1** (Matriz do Sol).

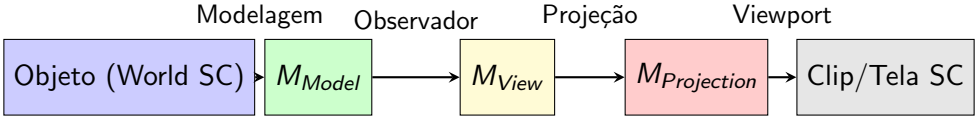
Consequência para o Desenho:

- O `glPopMatrix()` final desfaz a **Translação** e a **Rotação de Órbita**.
- A matriz atual retorna a ser apenas a matriz do Sol (ou Matriz Base da Câmera), limpa de qualquer transformação.
- Sem esse `glPopMatrix()`, a Lua seria desenhada transladada e rotacionada junto com o Planeta, perdendo a hierarquia.

A Cadeia de Transformações (Pipeline MVP)

Do Espaço do Objeto ao Espaço da Tela

O processo de renderização transforma um ponto do espaço do objeto para o espaço da tela através de três matrizes principais.



- **Model Matrix** (M_{Model}): Translação, Rotação e Escala aplicadas ao objeto (Matriz da Aula 7 e Stack).
- **View Matrix** (M_{View}): Posiciona e orienta a câmera (**Foco desta Seção**).
- **Projection Matrix** ($M_{Projection}$): Define a perspectiva (Lente da câmera - **Foco da Próxima Aula**).

A Matriz Unificada: GL_MODELVIEW

Fusão: Modelagem e Visualização

No OpenGL Fixed Pipeline, as transformações M_{Model} (do objeto) e M_{View} (da câmera) são combinadas em uma única matriz ativa: a GL_MODELVIEW.

$$M_{Modelview} = M_{View} \times M_{Model}$$

- **Aplicações:**
  1. **Movimentação da Câmera (View):** A Matriz de View (primeira a ser aplicada ao M_{Model}) move o **Mundo** para que o observador fique na origem.
  2. **Posicionamento do Objeto (Model):** As transformações subsequentes (glTranslate, glRotate) posicionam o objeto em relação à câmera (que está fixa na origem do Sistema de Coordenadas da Câmera).
- **Função Essencial da M_{View}:**
  - Transforma todos os pontos e vetores do Mundo de forma que o observador se encontre em $(0, 0, 0)$, olhando na direção do eixo $-Z$.

Definição da Janela de Visualização (Viewport)

Mapeamento para a Janela

O `Viewport` é a etapa final da transformação que mapeia as coordenadas normalizadas (após a projeção) para o sistema de coordenadas da tela (pixels).

Função: `void glViewport(GLint x, GLint y, GLsizei width, GLsizei height);`

- `(x, y)`: Coordenadas do canto inferior esquerdo da janela na tela (em pixels).
- `width`: Largura da viewport em pixels.
- `height`: Altura da viewport em pixels.

Uso Comum:

- Geralmente, cobre toda a janela da aplicação:
  `glViewport(0, 0, window_width, window_height);`
- Permite a divisão da tela

Viewport: A Distinção Crucial

Viewport vs. Projeção

É fundamental diferenciar as responsabilidades da **Matriz de Projeção** (próxima aula) e do **Viewport**.

Matriz de Projeção (GL_PROJECTION)

- **O Quê** se vê (Field of View - Campo de Visão).
- Determina a lente da câmera (ex: grande angular, teleobjetiva).
- Define a relação de aspecto da cena.
- `glOrtho` ou `gluPerspective`.

Viewport (`glViewport`)

- **Onde** se desenha na tela (Coordenadas de Pixel).
- Determina o tamanho e a posição da imagem final na janela.
- Não afeta a distorção da perspectiva, apenas o mapeamento 2D.
- `glViewport`.

Exemplo prático

Se a Projeção for uma foto, o Viewport é a moldura e a posição dessa foto dentro da sua janela de aplicação. Você pode mudar a moldura (`glViewport`) sem tirar uma nova foto (Projeção).

Posicionamento da Câmera (`View Matrix`)

Definindo o Observador no Mundo

A View Matrix é responsável por posicionar e orientar o observador (a câmera) no Espaço do Mundo. No OpenGL, usamos a biblioteca auxiliar GLU.

Função Principal:

- `void gluLookAt(GLdouble eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);`

Três Vetores Chave:

1. **Eye (Posição da Câmera):** Onde a câmera está no espaço 3D.
2. **Center (Ponto de Interesse):** O ponto para o qual a câmera está olhando.
3. **Up (Orientação Vertical):** Define qual direção é "para cima" para o observador (normalmente $(0, 1, 0)$).

Ação Imediata do `gluLookAt`

Regra da Matriz de Visualização

O `gluLookAt` não é uma função de modelagem; ele é uma ferramenta de conveniência que **calcula e aplica** a Matriz de View (M_{View}).

Fluxo de Execução:

1. **Cálculo:** O GLU calcula a matriz M_{View} necessária para mover o mundo para o Sistema de Coordenadas da Câmera (SCC).
2. **Multiplicação:** O M_{View} calculado é **multiplicado** pela matriz ativa atual (que deve ser M_{Model} ou a identidade).
3. **Resultado:**
   - A Matriz Ativa passa a ser: $M'_{atual} = M_{\text{View}} \times M_{\text{atual}}$.
   - Se você chamou `glLoadIdentity()` antes, a matriz ativa é M_{View}.

Detalhes: O que o `gluLookAt` Realmente Faz

A Matriz de View é a Inversa da Transformação da Câmera

O `gluLookAt` não move a câmera, ele move o **Mundo** em relação à câmera. Ele calcula uma matriz de View M_{View} que, quando aplicada, coloca a câmera na origem $(0, 0, 0)$ e a faz olhar para a direção $-Z$.

Passos Conceituais (Cálculo Interno):

1. **Cria um Sistema de Coordenadas da Câmera (SCC):**
   - **Eixo Z:** Vetor direção (Center - Eye).
   - **Eixo X:** Produto vetorial (Z $\times$ Up).
   - **Eixo Y:** Produto vetorial (X $\times$ Z).
2. **Calcula a Matriz de Rotação (Mundo $\rightarrow$ SCC):**
   - Uma matriz R que rotaciona os eixos do Mundo para alinhar com os eixos do SCC.
3. **Calcula a Matriz de Translação:**
   - Uma matriz T que translada a origem do Mundo para a posição da Câmera (Eye).
4. **Resultado:** A Matriz de View é dada por: $\mathbf{M_{View}} = \mathbf{R} \cdot \mathbf{T}$.

Exemplo Prático com gluLookAt

Câmera em Posição Estática

Para renderizar uma cena a partir de um ponto fixo, a View Matrix deve ser configurada logo após carregar a identidade na GL_MODELVIEW.

```
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity(); // Começa com a matriz limpa (Identidade)

    // Configura a Câmera (View Matrix)
    gluLookAt(
    10.0, 5.0, 10.0, // Eye: Câmera em (10, 5, 10)
    0.0, 0.0, 0.0,  // Center: Olhando para a origem do Mundo
    0.0, 1.0, 0.0   // Up: Eixo Y é a direção 'para cima'
    );

    // --- Início das Transformações de Modelagem ---
    glRotatef(angle, 0.0, 1.0, 0.0);
    drawScene(); // Desenha todos os objetos da cena
```

Movimentação Interativa da Câmera

Câmera Livre (*Free-Look*)

A movimentação dinâmica envolve alterar as variáveis `Eye` (posição) e `Center` (direção) em cada frame, geralmente usando a entrada do usuário (teclado/mouse).

Variáveis de Câmera (Globais):

- `vec3 cameraPos = (0.0, 0.0, 3.0);`
- `vec3 cameraFront = (0.0, 0.0, -1.0);`
- `vec3 cameraUp = (0.0, 1.0, 0.0);`

Lógica de Atualização (Frame a Frame):

1. **Entrada do Usuário:** Detectar teclas pressionadas (e.g., 'W' para avançar).
2. **Cálculo do Movimento:** `cameraPos += speed * cameraFront;`
   gluLookAt(cameraPos.x, cameraPos.y, cameraPos.z, cameraPos.x + cameraFront.x, cameraPos.y + cameraFront.y, cameraPos.z + cameraFront.z, cameraUp.x, cameraUp.y, cameraUp.z);

Movimentação: Detalhes Essenciais (Strafe e Normalização)

Cálculo do Vetor de Movimento

Para um movimento real em 3D (avançar, recuar, lateral), o vetor de direção precisa ser preciso e unitário.

1. Normalização (Velocidade Consistente) - Um vetor normalizado possui magnitude (comprimento) igual a 1.

- **Por Quê:** Garante que a velocidade de movimento (`speed`) seja constante em todas as direções, independentemente do comprimento atual do vetor `cameraFront`.
- **Cálculo:** `cameraFront.normalize();`

2. Movimento Lateral (Strafe) - Mover a câmera para a direita ou esquerda em relação à sua direção atual.

- **Cálculo:** O vetor lateral (`cameraRight`) é obtido através do **Produto Vetorial** (*Cross Product*) entre a direção atual e o vetor "para cima" do mundo.
- `vec3 cameraRight = cross(cameraFront, cameraUp);`
- **Aplicação:** `cameraPos += speed * cameraRight;`

Considerações sobre a Movimentação da Câmera

- A movimentação suave da câmera melhora a experiência do usuário
- Considere implementar aceleração e desaceleração para transições mais naturais
- Evite movimentos abruptos que possam desorientar o usuário
- Teste diferentes configurações de velocidade e sensibilidade para encontrar o equilíbrio ideal

Rotação da Câmera: Pitch e Yaw (Mouse)

O Segredo da Rotação Livre (Free-Look)

Para rotacionar a câmera com o mouse, a entrada de movimento 2D (`dx`, `dy`) é convertida em alterações angulares: **Yaw** (Horizontal) e **Pitch** (Vertical).

Ângulos e Movimento:

- **Yaw (Guinada):** Rotação em torno do eixo Y (vira a câmera para a esquerda/direita).
- **Pitch (Arfagem):** Rotação em torno do eixo X (vira a câmera para cima/baixo).

Trigonometria no Vetor Frontal: Após atualizar os ângulos 'Yaw' e 'Pitch', o vetor de direção da câmera (`cameraFront`) é recalculado usando seno e cosseno para projetar a direção no espaço 3D:

```
cameraFront.x = cos(yaw) * cos(pitch);
cameraFront.y = sin(pitch);
cameraFront.z = sin(yaw) * cos(pitch);
// Depois, normalizar (garantir que o vetor tenha comprimento 1)
cameraFront.normalize();
```

Integração Final: Modelagem Hierárquica e Câmera

Exercício Conceitual de Fixação

Combine os conceitos de Modelagem Hierárquica e Posicionamento do Observador.

Cenário: Queremos modelar um Braço Robótico articulado (Base, Braço, Antebraço) enquanto a câmera o observa de cima, girando lentamente ao redor dele.

Tarefa (Discussão):

1. **Câmera:** Quais seriam os valores ideais de `Eye, Center, Up` para o `gluLookAt` fixo (sem movimento interativo)?
2. **Base (Pai):** Para desenhar a base, qual sequência de `glPushMatrix/glPopMatrix` e transformações é necessária?
3. **Braço (Filho):** O braço deve ser desenhado transladado a partir da Base, mas rotacionado em torno de sua própria junta. Qual a ordem correta (T $\rightarrow$ R ou R $\rightarrow$ T) no código?

Referências

- Angel, E., & Shreiner, D. Interactive Computer Graphics: A Top-Down Approach with WebGL, 7th Edition, Pearson, 2014.
- Shreiner, D., Sellers, G., Kessenich, J., & Licea-Kane, B. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3, 8th Edition, Addison-Wesley Professional, 2013.
- **Sites**:
  - **Learn OpenGL**: `https://learnopengl.com/`
  - **MDN WebGL Documentation**: `https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API`