# Transformações de Modelagem, Visualização e Projeção em OpenGL

Prof. Mauro Miazaki

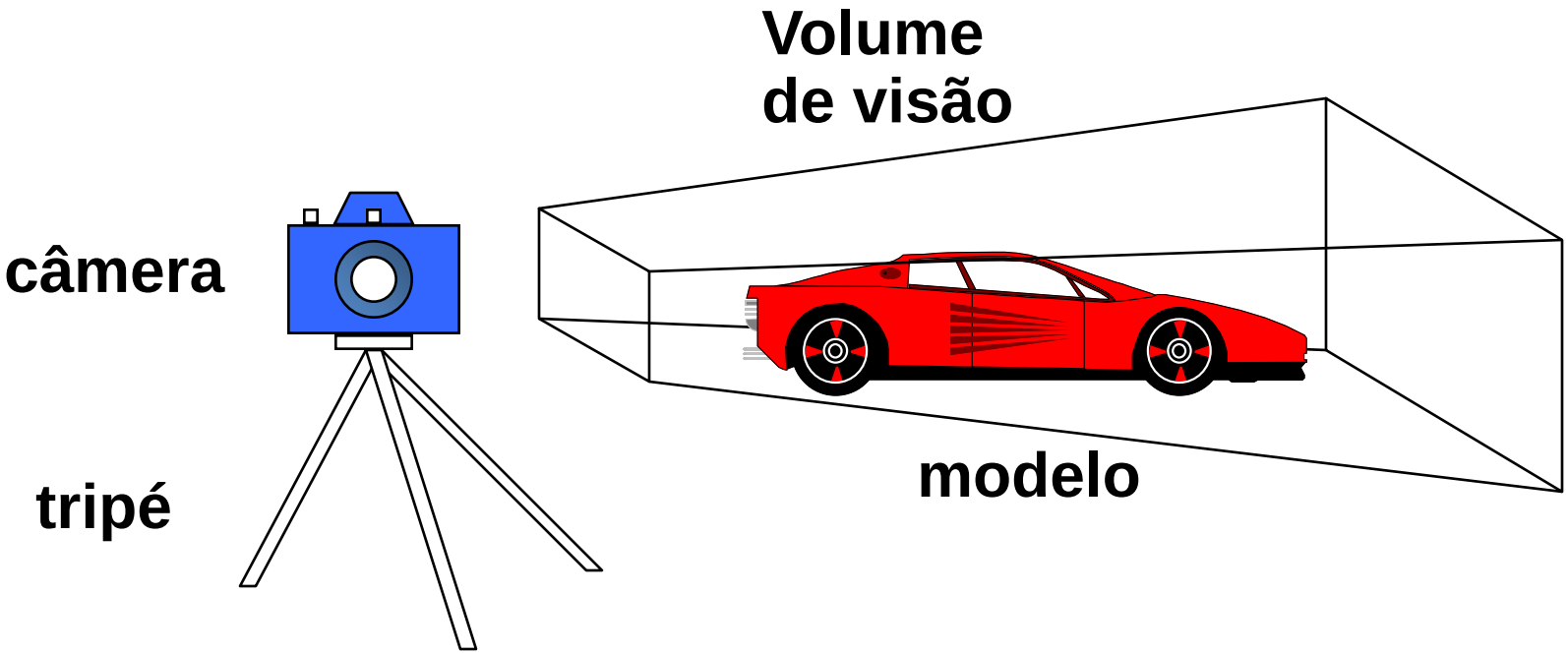
DECOMP - UNICENTRO
Guarapuava/PR, 2025

# Conteúdo

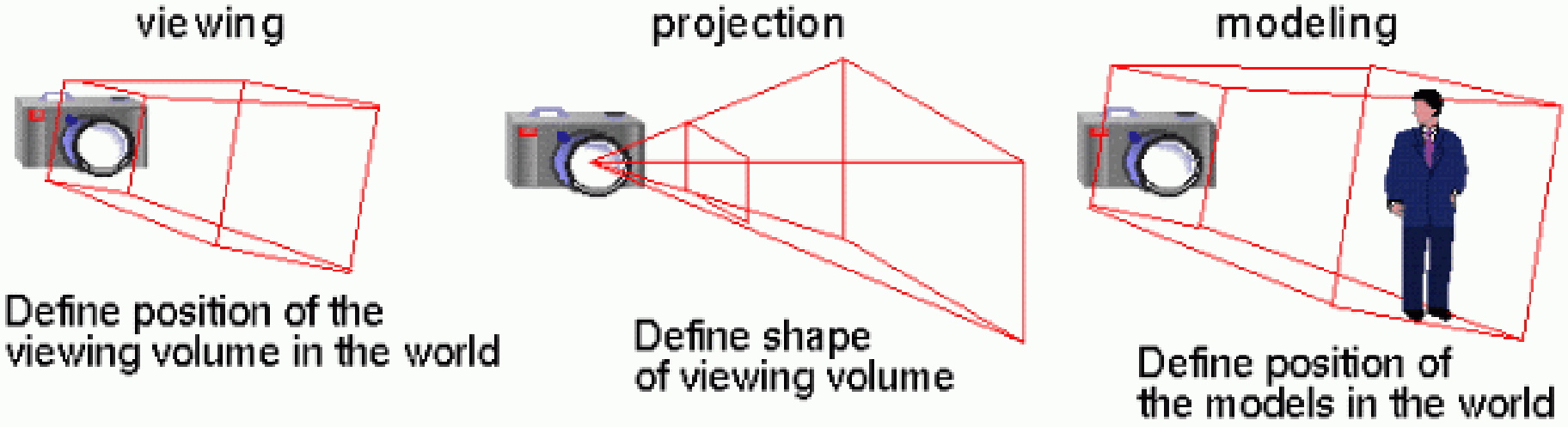
- Analogia com uma câmera.
- Transformações em OpenGL:
  - Modelagem;
  - Visualização;
  - Projeção.
- Pilhas de Matrizes.
- Composição de transformações.

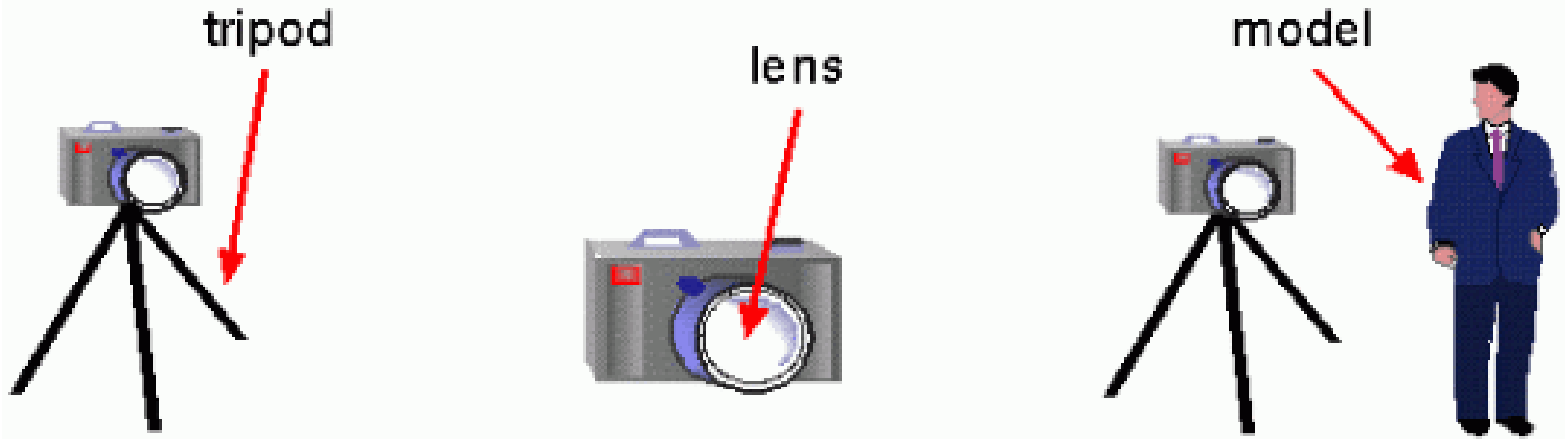
# Analogia com uma câmera

- OpenGL utiliza uma analogia comparando visualização 3D com tirar fotografias com uma câmera.



# Analogia com uma câmera



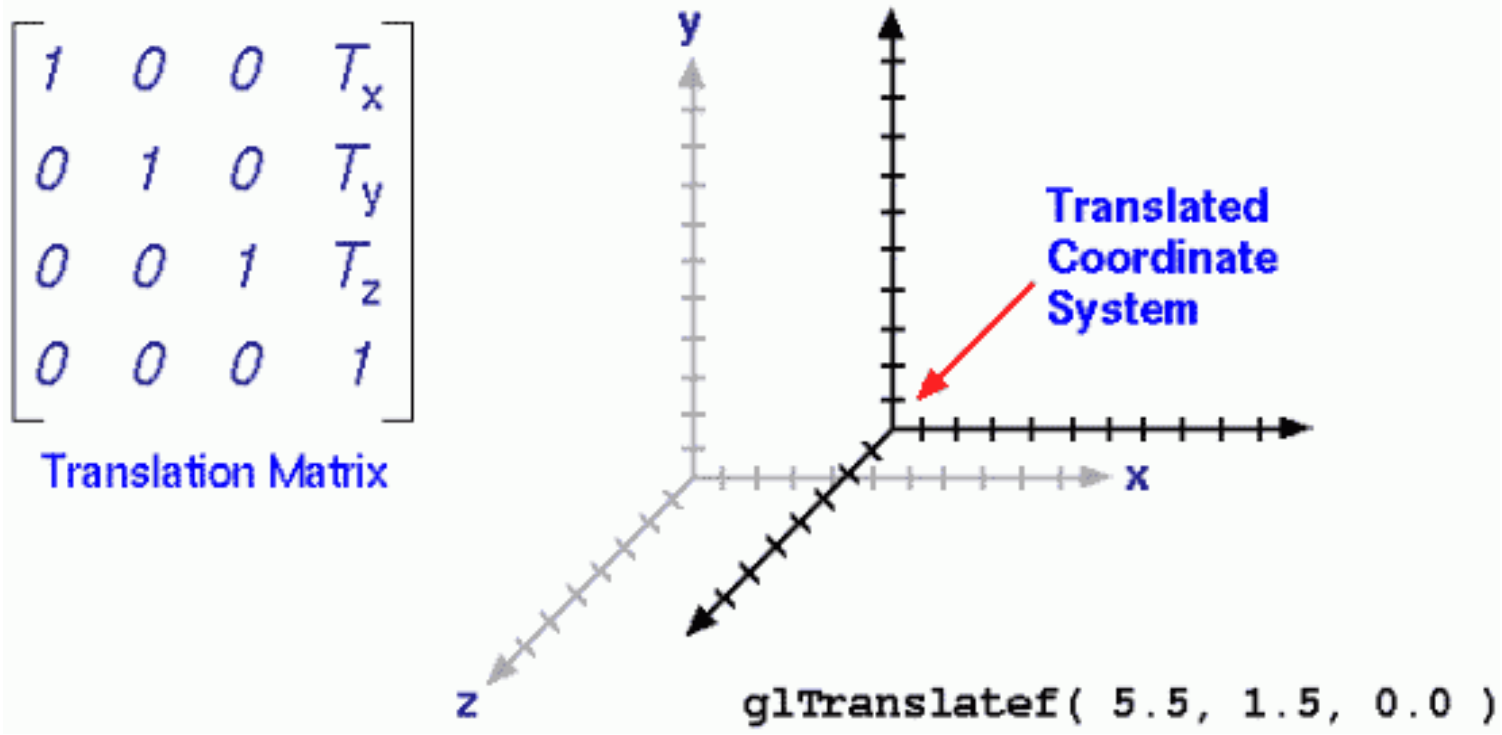
# Transformações em OpenGL

- Modelagem:
  - Mover/deformar os objetos (transformações geométricas).
- Visualização:
  - Mover e orientar a câmera;
  - Viewport: aumentar ou reduzir a fotografia.
- Projeção:
  - Ajuste da lente / objetiva da câmera.

# Transformações de Modelagem

# Transformações de Modelagem

- Translação:
  - glTranslatef(Tx, Ty, Tz)

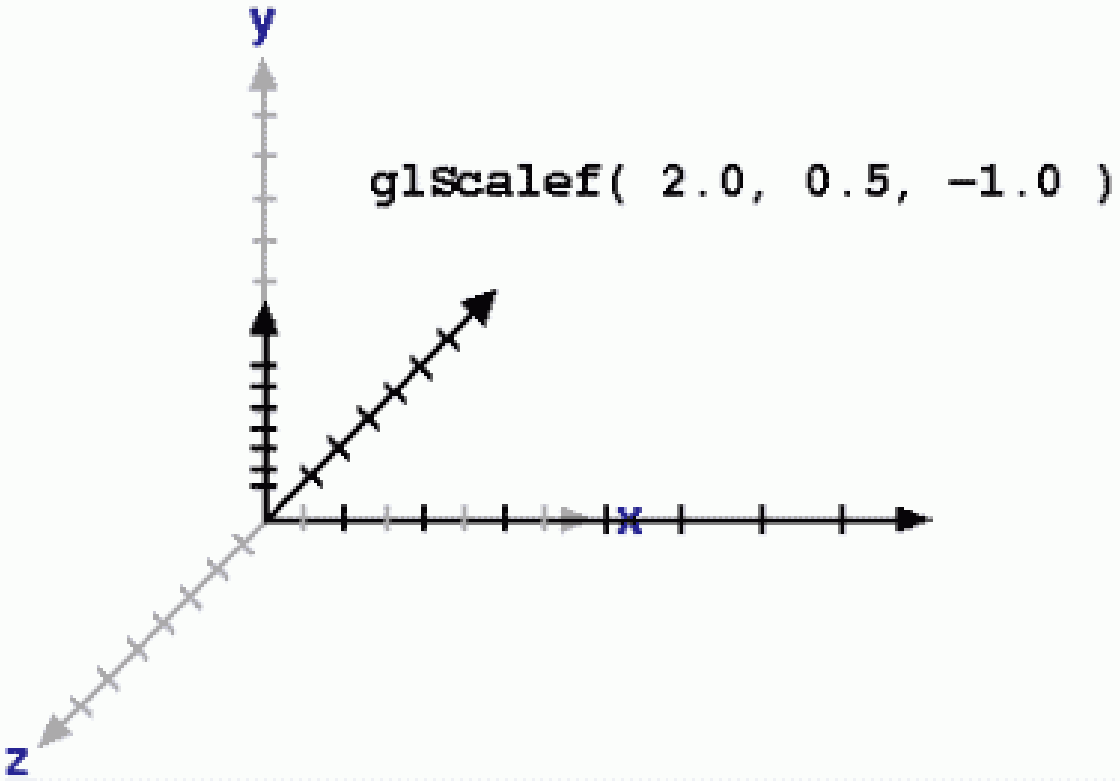


# Transformações de Modelagem

- Escala:
  - glScalef(Sx, Sy, Sz)

$$\begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scale Matrix

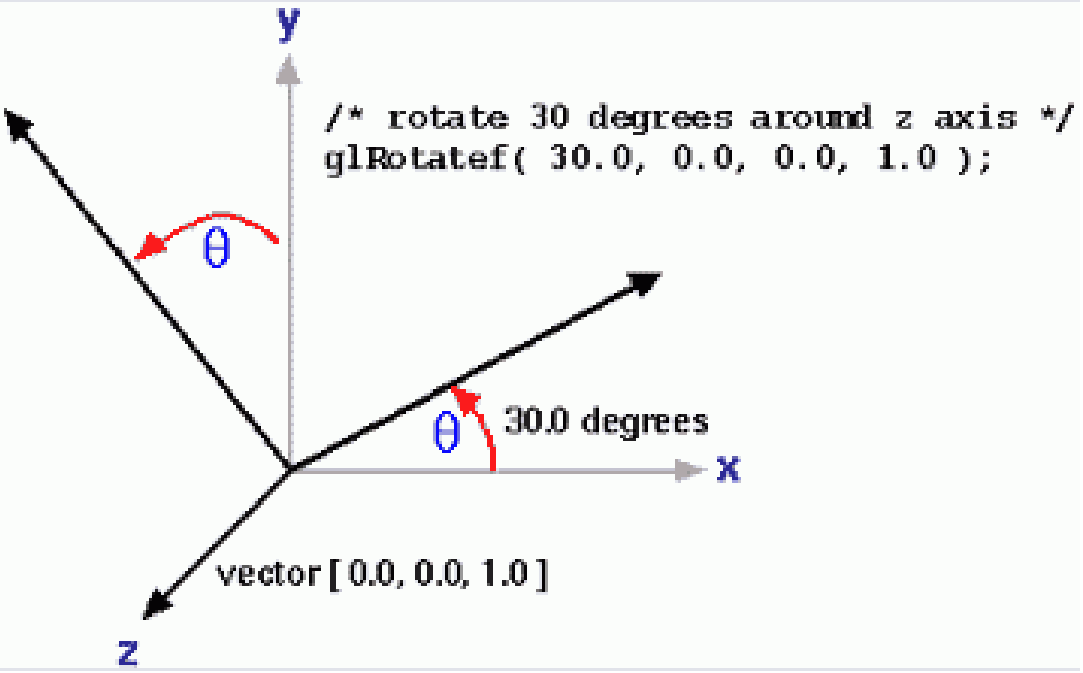


# Transformações de Modelagem

- Rotação:
  - glRotatef(angle, x, y, z)
  - Ângulos positivos representam rotação no sentido anti-horário

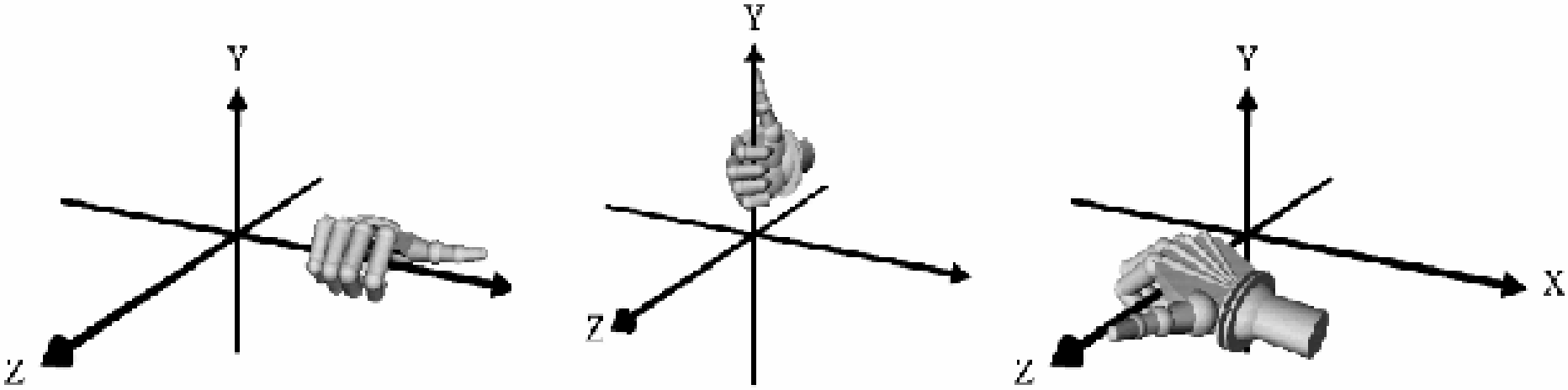
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation Matrix

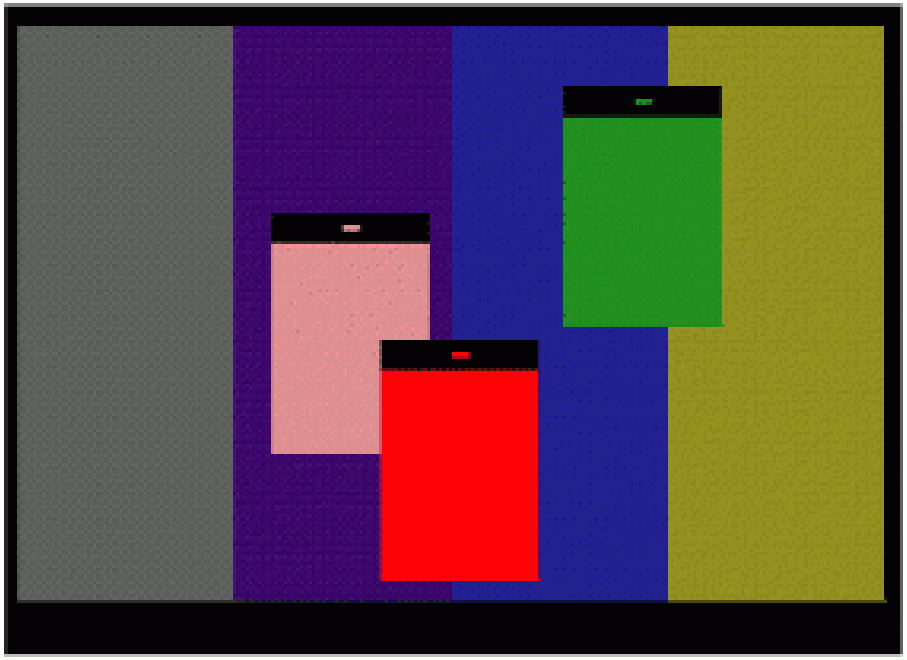


# Transformações de Modelagem

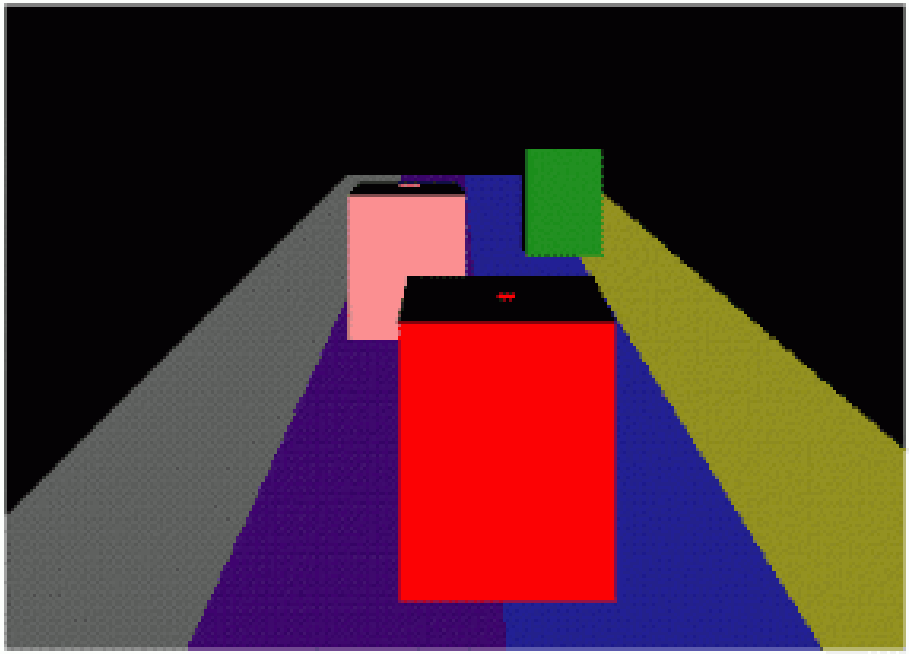
- Regra da mão direita:
  - Esta regra ajuda a lembrar as direções de rotações positivas e negativas.



# Transformações de Projeção



Ortográfica

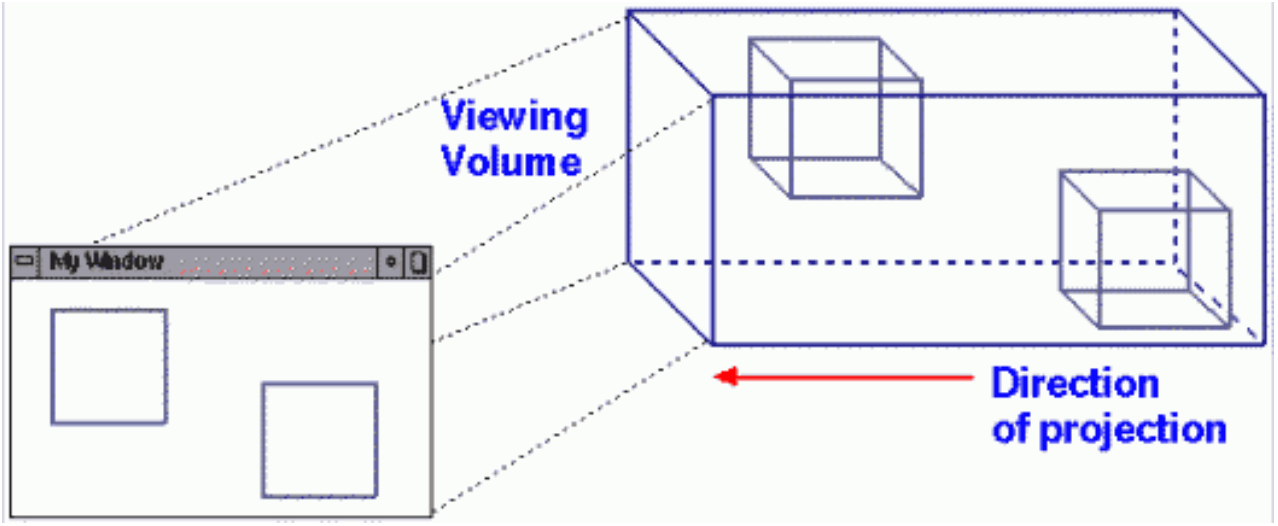


Perspectiva

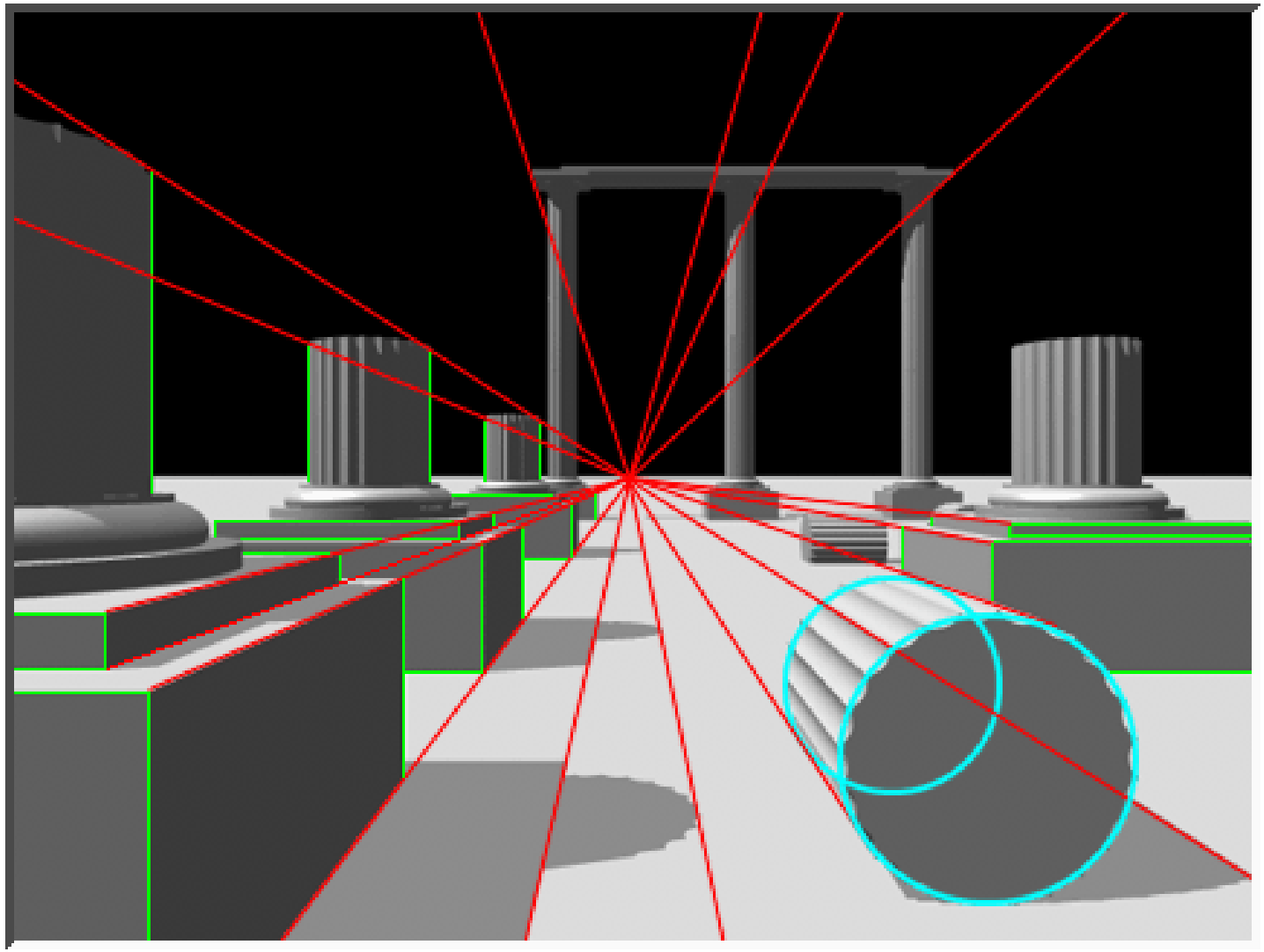
- Projeções geométricas permitem a visualização bidimensional de objetos tridimensionais.

# Projeção ortográfica

- **glOrtho(left, right, bottom, top, nearClip, farClip)**
- Imita raios de luz paralelos captados pelos olhos.
  - Também chamado projeção paralela.
- Objetos são projetados na tela de maneira uniformemente plana.

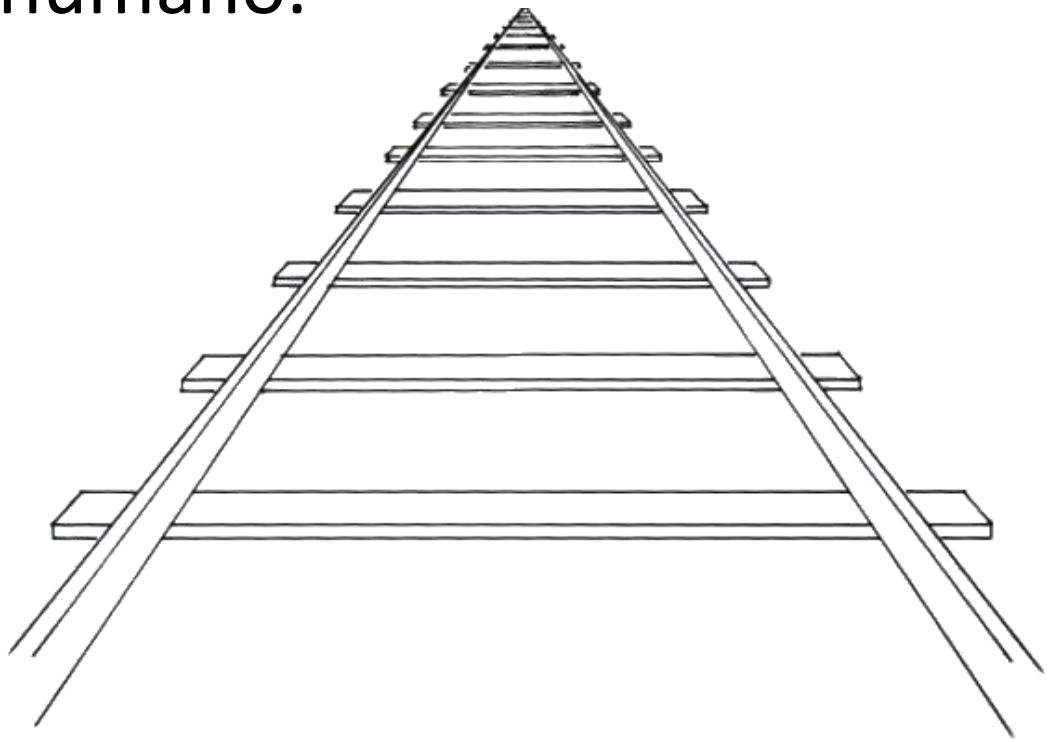


# Projeção perspectiva



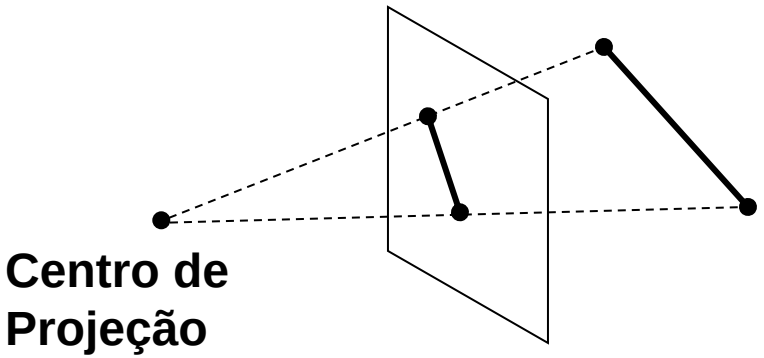
# Projeção perspectiva

- Representação do espaço 3D pela forma vista pelo olho humano.



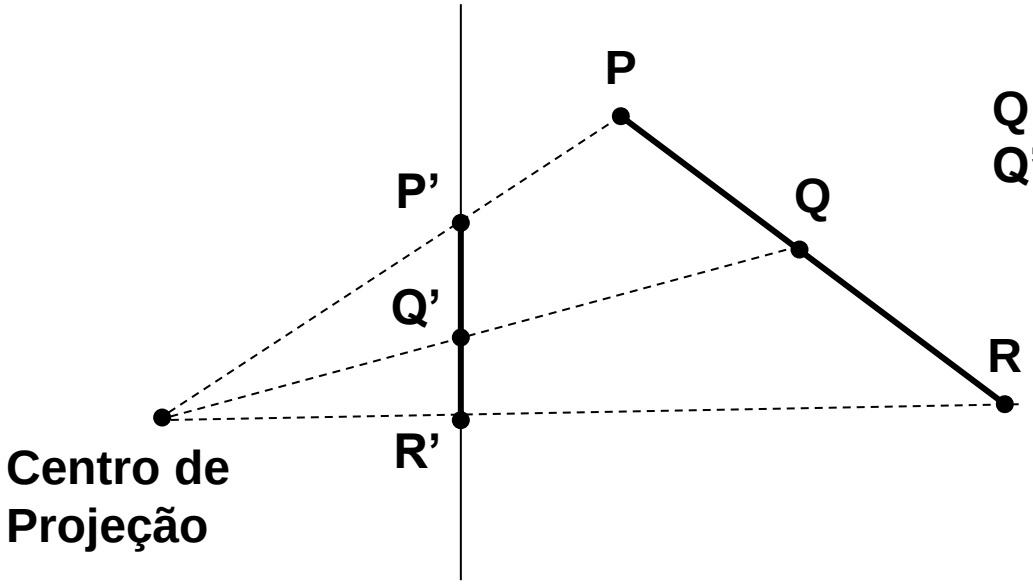
# Perspectiva

- É o estudo de transformações projetivas.
- O problema consiste em projetar pontos do espaço 3D no plano 2D usando um ponto especial chamado de centro de projeção.



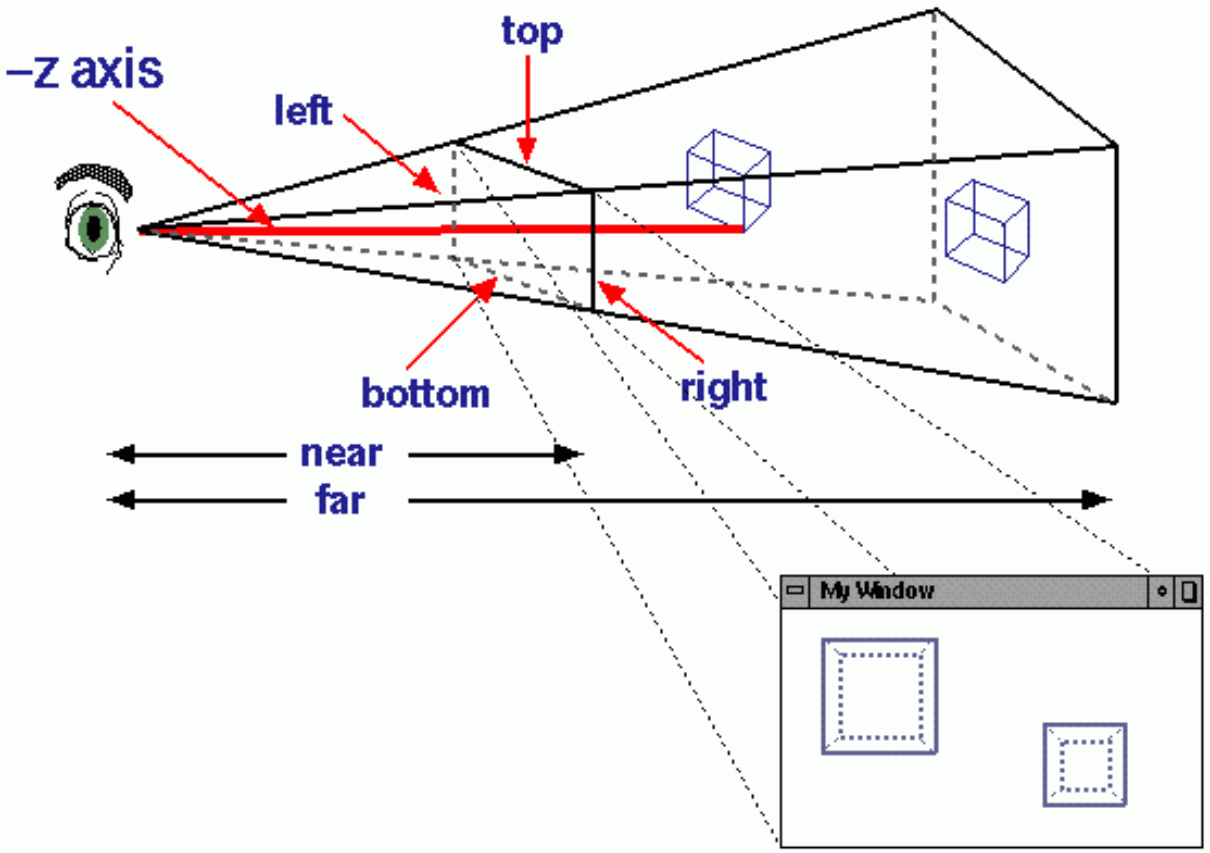
# Transformações Projetivas

- Transformações projetivas transformam retas em retas, mas não preservam proporções.



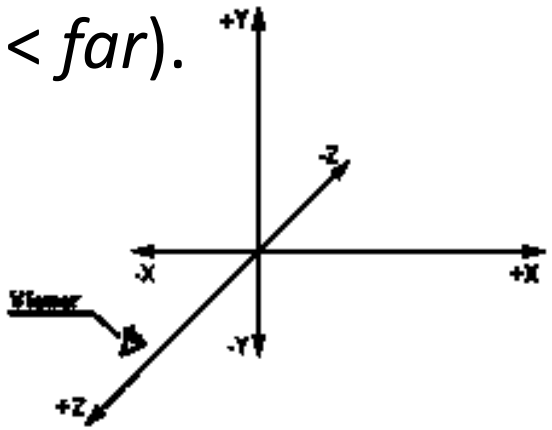

**Q é o ponto médio de PR, mas Q' não é o ponto médio de P'R'**

- **glFrustum(left, right, bottom, top, nearClip, farClip)**
  - Objetos mais próximos do ponto de visão aparentam ser maiores.
  - nearClip e farClip devem ser valores positivos.

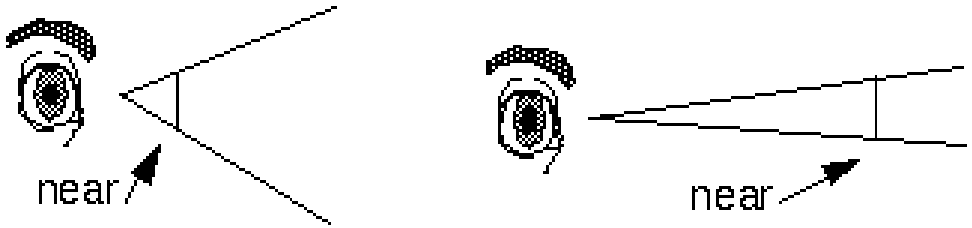


**glFrustum(left, right, bottom, top, nearClip, farClip)**

- O olho localiza-se na origem, olhando na direção –z.
- *Near* e *Far* especificam a distância do olho aos planos de corte ao longo do eixo z (0.0 < *near* < *far*).

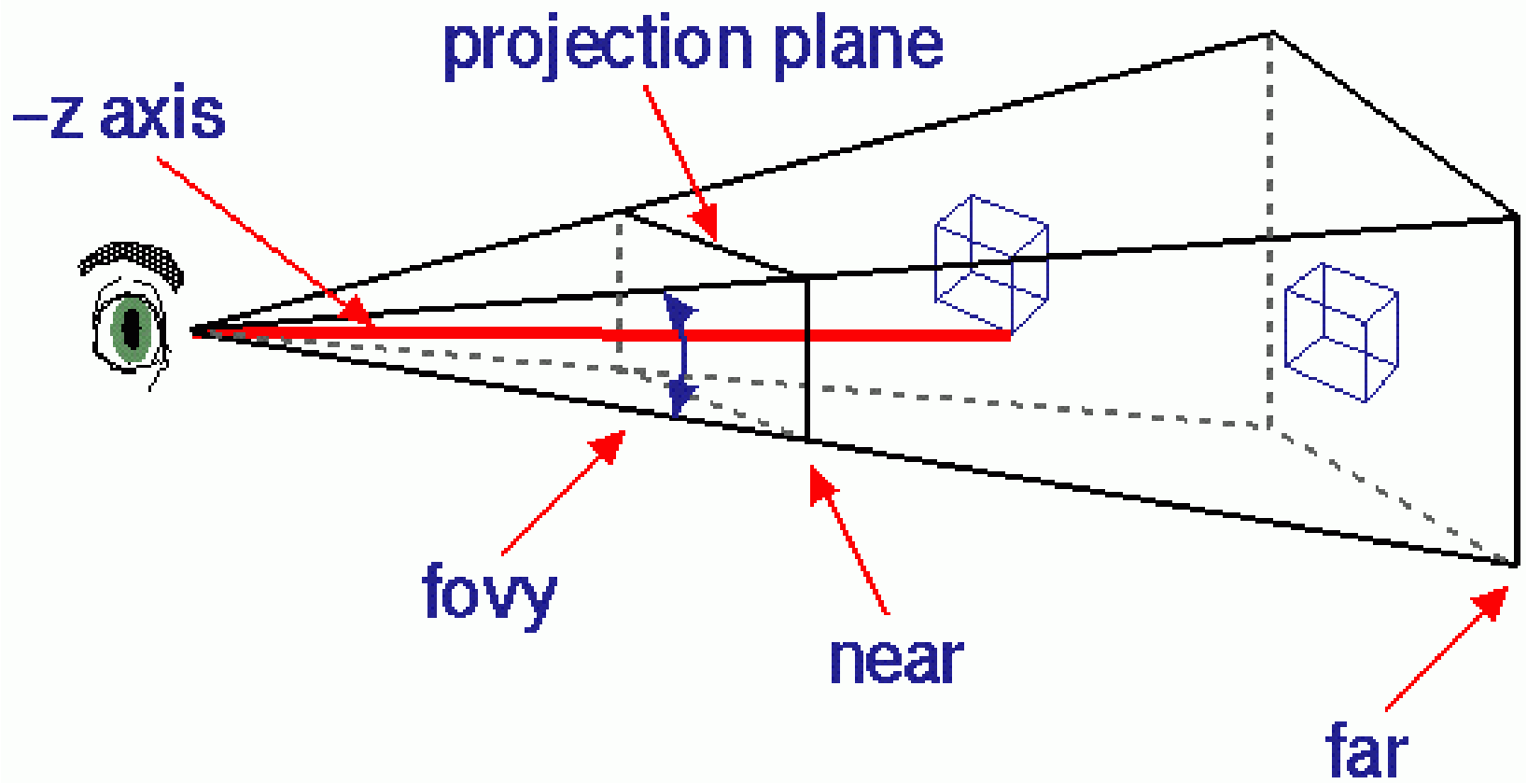



- Modificar *Near* altera a forma do volume de visualização.

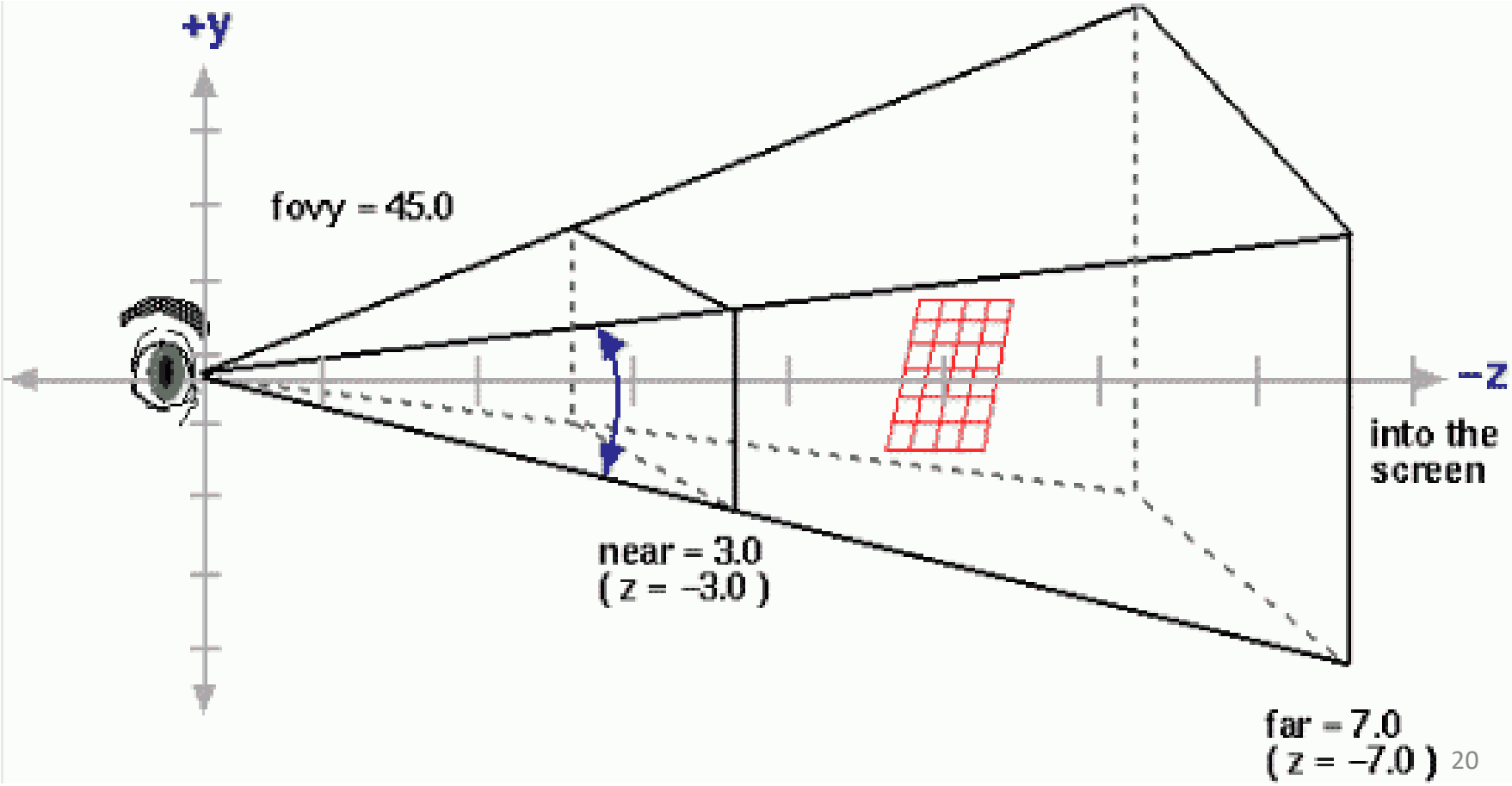



- Como glFrustrum() não é intuitivo para se usar, uma função relacionada foi adicionada à biblioteca GLU.

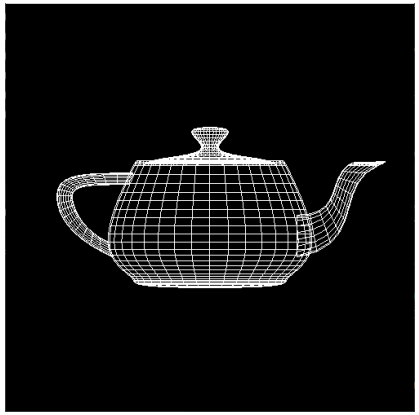
- **gluPerspective**(**fovy**, **aspect**, **nearClip**, **farClip** )
  - Visão simétrica em x e y ao longo da linha de visão.
  - Fovy: ***f****ield* ***o****f* ***v****iew angle in the* ***y*** *direction* (em graus).



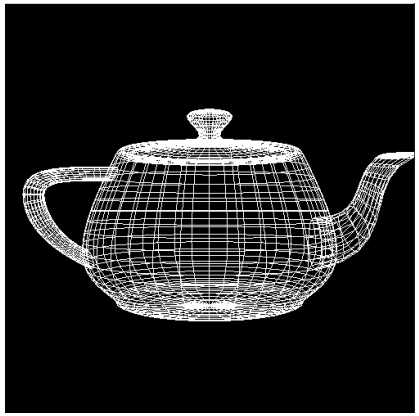
- Para que os objetos sejam vistos, deve-se assegurar que estejam no campo de visão.
- Ex.: gluPerspective( 45.0, 1.0, 3.0, 7.0 )
  glTranslatef( 0, 0, -5.0 )
  desenhaGrade()



# Exemplo



- Desenhe: glutWireTeapot(0.5)
- Visualize com:
  glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0)

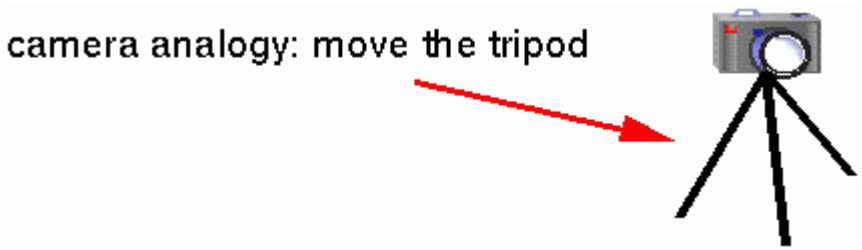


- Agora, visualize com:
  gluPerspective( 45.0, 1.0, 1.0, 5.0 )
- Será necessário:
  - Mover o objeto: glTranslatef(0, 0, -2)
  - Ou mover a câmera: gluLookAt(0, 0, 2, 0, 0, 0, 0, 1, 0)

# Transformações de Visualização

# Movendo o Olho

- Orientação padrão do olho:
  - Na origem;
  - Olhando ao longo do eixo z (do sentido positivo para o negativo).
- Suponha que se queira olhar para algum outro lugar:
  - Pode-se usar uma transformação de visualização para trocar a posição e orientação do olho.

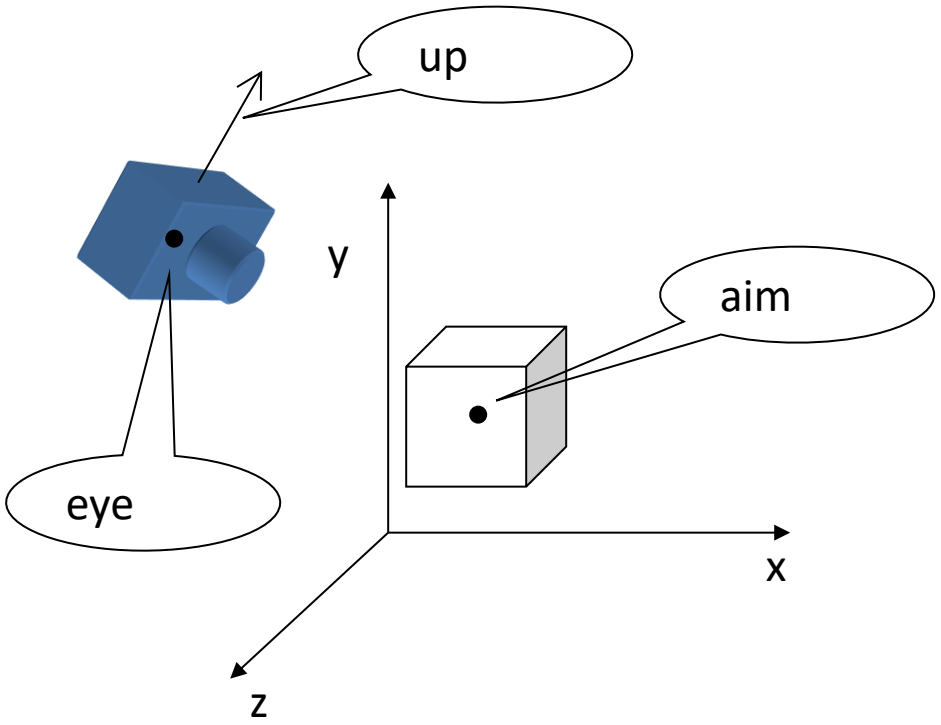


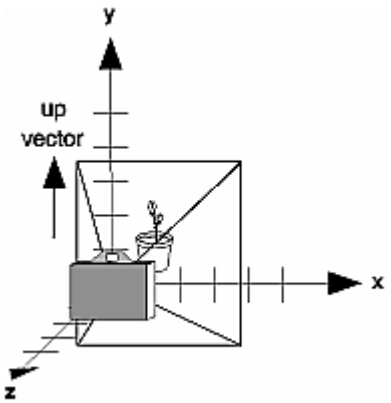
# Transformações de Visualização

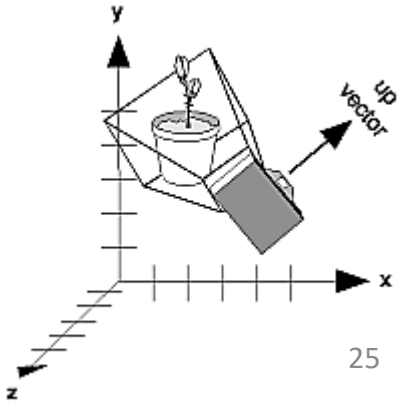
- Geralmente composta de rotações e transformações.
- Invocado antes das transformações de modelagem.
- Diversas maneiras de ser realizado:
  - Usar **glTranslatef()** e/ou **glRotatef()** diretamente;
  - Usar **gluLookAt();**
  - Combinar diversas transformações de modelagem para criar sua própria matriz de visualização.

# gluLookAt()

- Duas interpretações:
  - Levam a câmera até a cena que se quer visualizar
  - Levam os objetos da cena até uma câmera estacionária
- `gluLookAt(`
  $eye_x$, $eye_y$, $eye_z$,
  $aim_x$, $aim_y$, $aim_z$,
  $up_x$, $up_y$, $up_z$)
  - eye = ponto onde a câmera será posicionada
  - aim = ponto para onde a câmera será apontada
  - up = vetor que dá a direção "para cima" da câmera

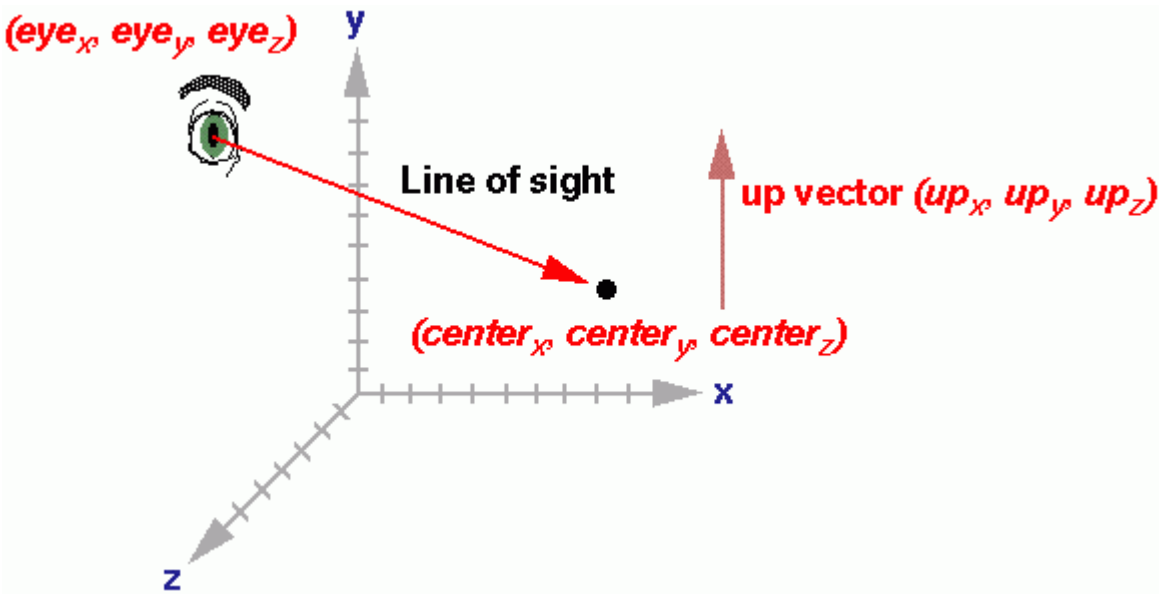


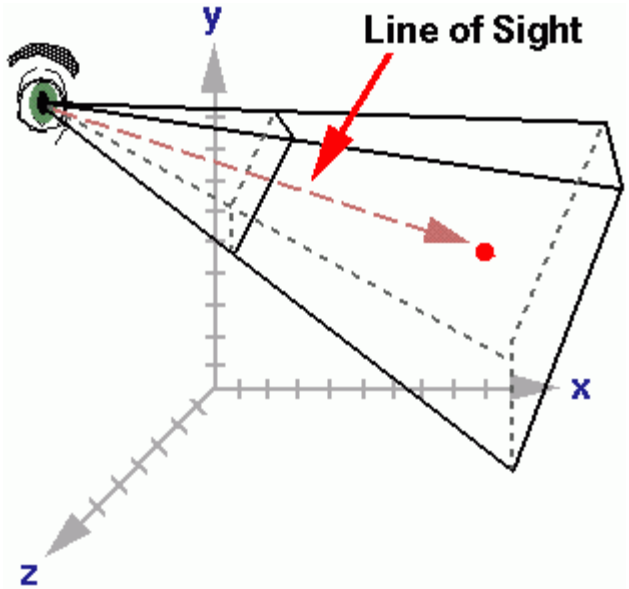



# gluLookAt()

- **gluLookAt** (
  $eye_x$, $eye_y$, $eye_z$,
  $center_x$, $center_y$, $center_z$,
  $up_x$, $up_y$, $up_z$ )



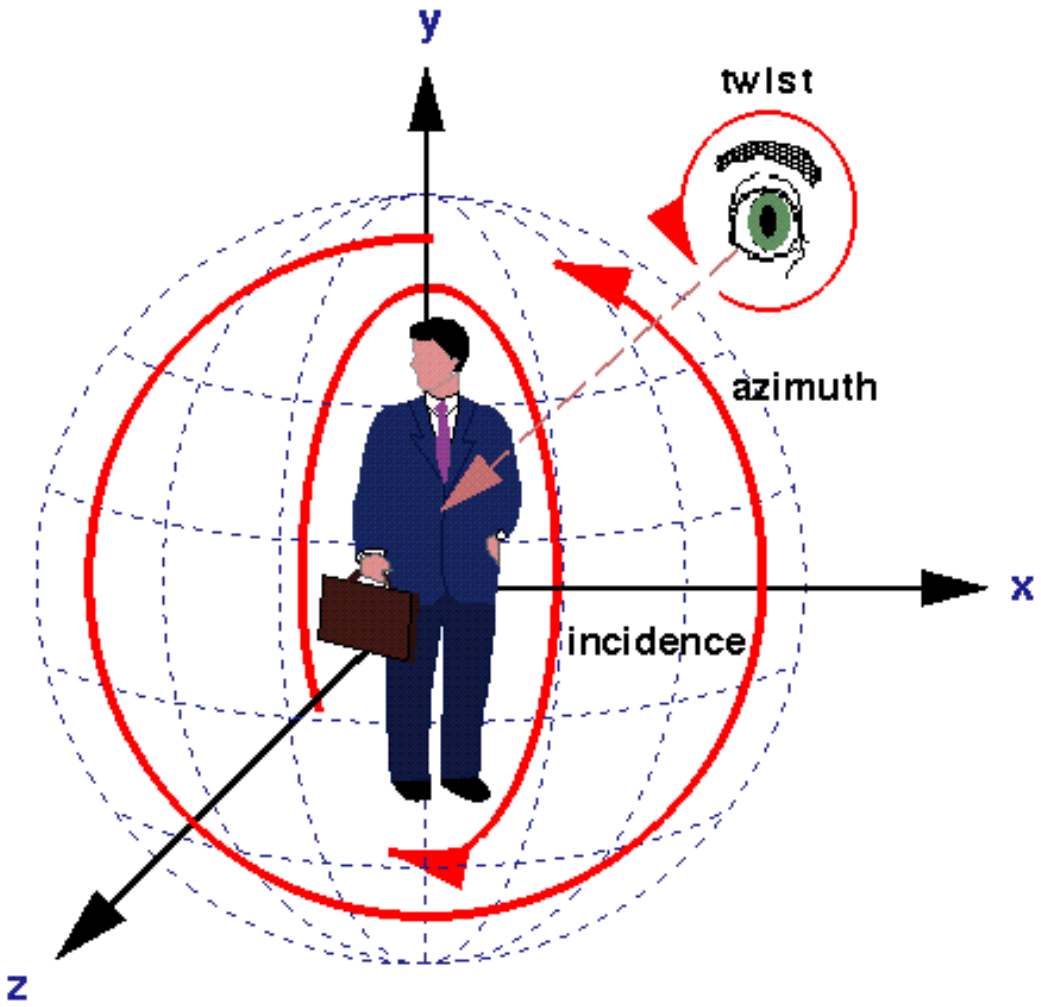



# Outras Formas de Olhar

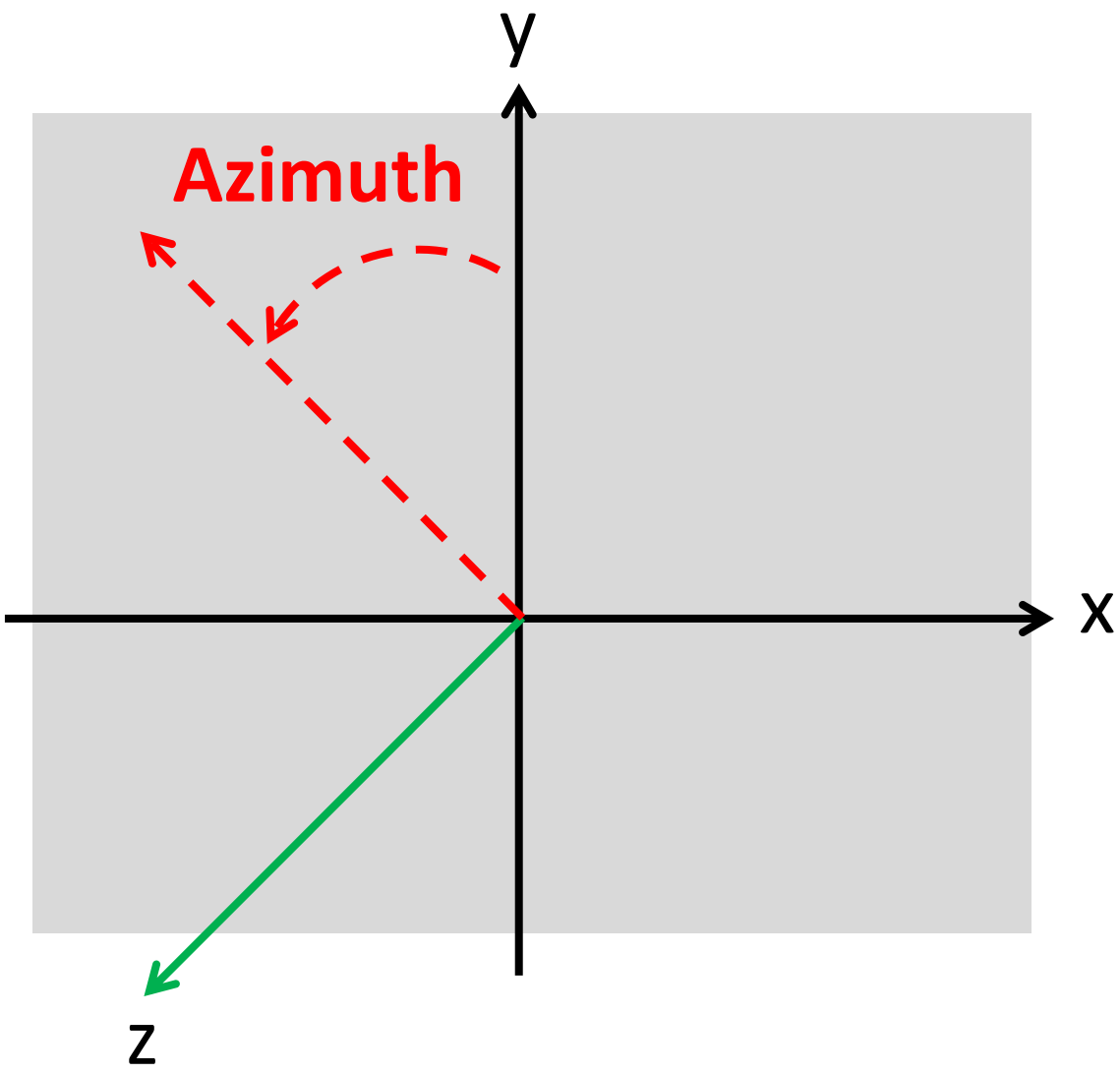
- gluLookAt() é muito bom para se movimentar pela cena:
  - Realidade virtual;
  - Projeto arquitetural;
  - Simulação visual.
- Para outras áreas seria melhor ver o modelo inteiro de vários ângulos:
  - Visualização científica;
  - Modelagem molecular;
  - CAD/CAM.

# Visualização Polar

- Ver o modelo como se estivesse dentro de uma esfera de vidro centrada na origem.

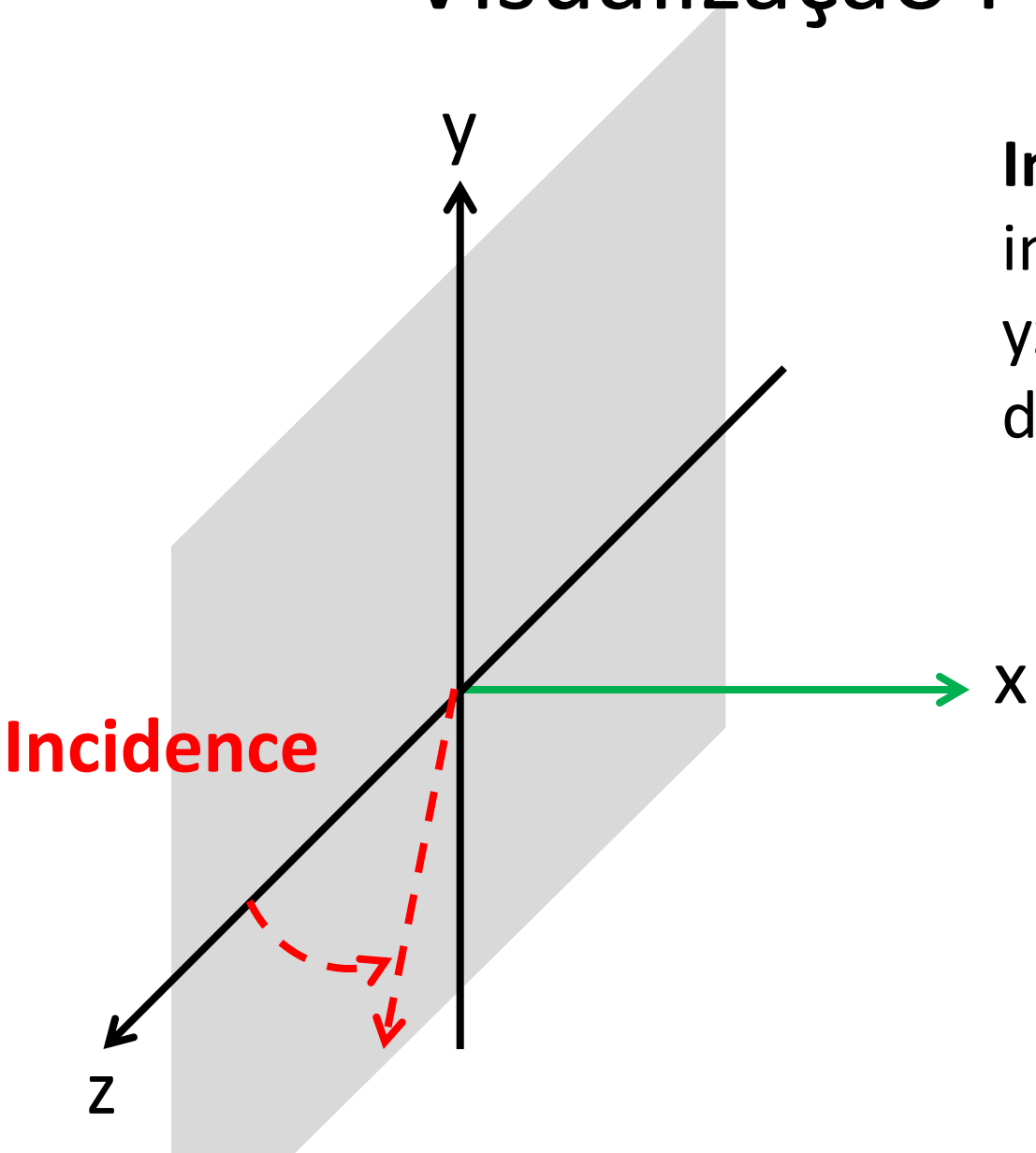


# Visualização Polar




**Azimuth**: ângulo azimute no plano xy medido a partir do eixo y.

# Visualização Polar




**Incidence**: ângulo de incidência no plano yz medido a partir do eixo z.

# Visualização Polar

- **Distance**: distância do olho à origem do sistema de coordenadas do mundo (raio).
- **Twist**: rotação do ponto de visão ao redor da linha de visão.

# Visualização Polar

- polarView() emula a função polarview() presente na biblioteca IRIS GL.

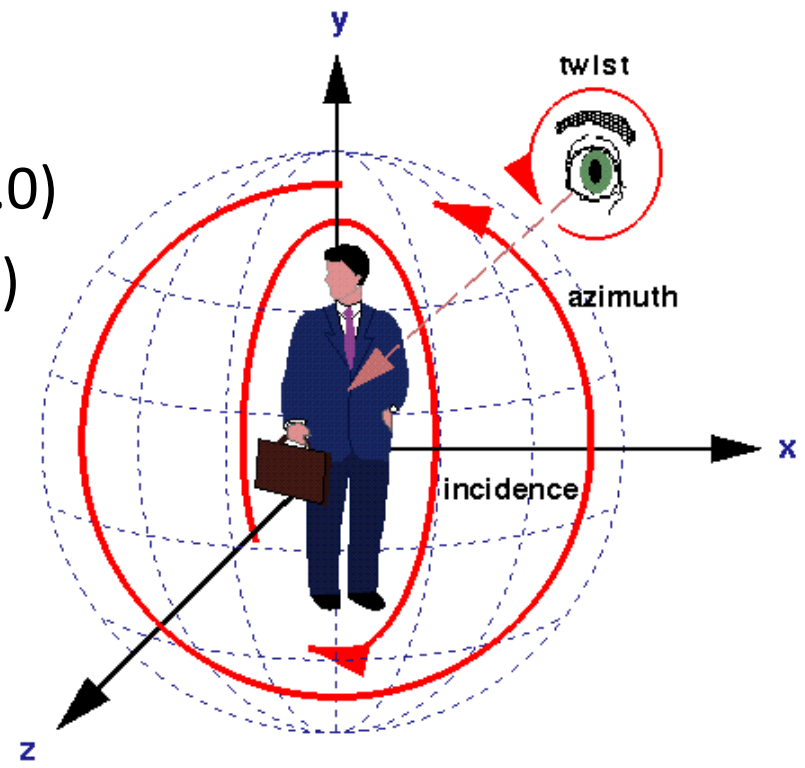
def **polarView**(**distance**, **azimuth**, **incidence**, **twist**):

glTranslatef(0.0, 0.0, -distance)

glRotatef(-twist, 0.0, 0.0, 1.0)

glRotatef(-incidence, 1.0, 0.0, 0.0)

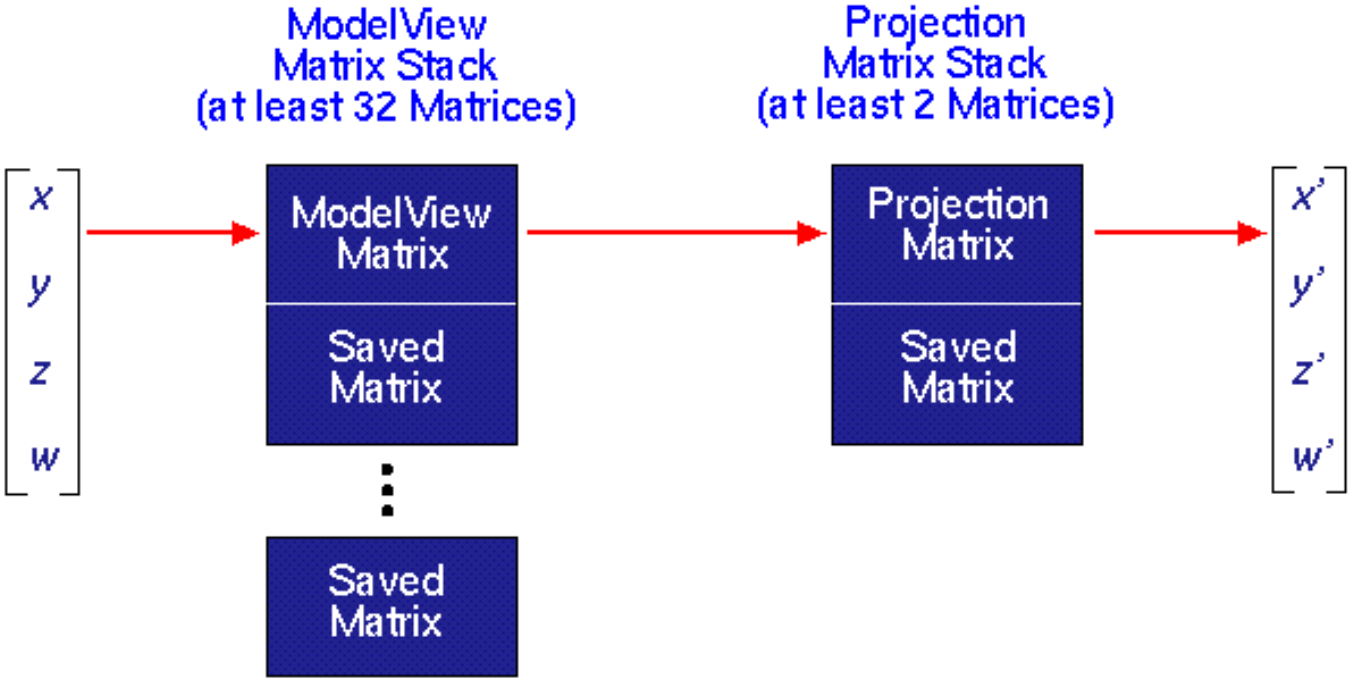
glRotatef(-azimuth, 0.0, 0.0, 1.0)



# Pilhas de Matrizes

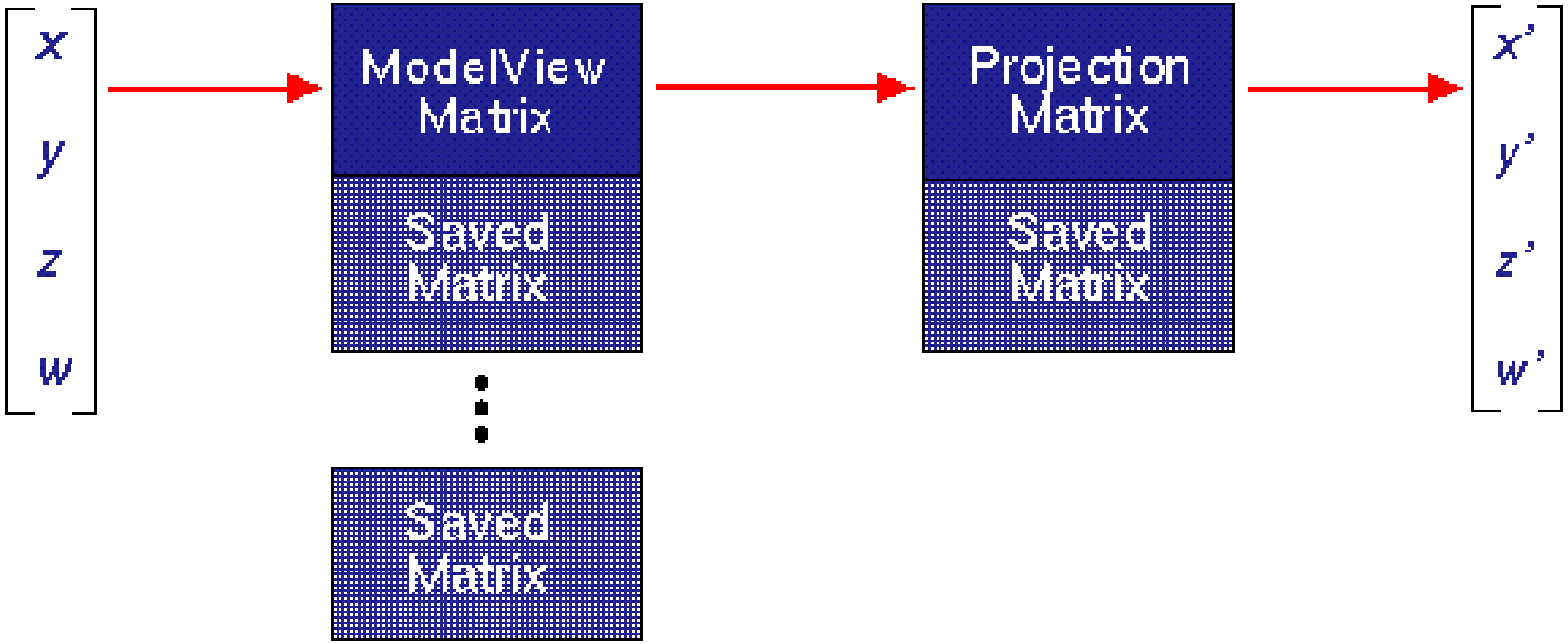
# Pilhas de Matrizes

- Há duas pilhas de matrizes usadas para salvar as transformações de modelagem/visualização e de projeção.
  - Todas as transformações são realizadas na matriz no topo da pilha corrente.



# Transformações em vértices

- Cada vértice é primeiro multiplicado pela matriz ModelView, e depois pela matriz Projection.
  - Inicialmente, cada pilha contém a matriz identidade.



# Modos de Matriz

- glMatrixMode( *mode* )
- Determina qual é a pilha corrente.
  - A matriz corrente é a do topo da pilha corrente.
  - Transformações subsequentes afetam esta matriz.
  - Devem ser aplicados os tipos de transformações apropriados para cada um.
- *mode* indica qual pilha usar.

| Modo | Onde as transformações ocorrem |
|---|---|
| GL_PROJECTION | Projection Matrix |
| GL_MODELVIEW | ModelView Matrix |

# Transformações

- As transformações são definidas em funções que multiplicam o topo da pilha de matrizes pelas transformações especificadas:
  - GL_PROJECTION:
    - glOrtho()
    - gluOrtho2D()
    - glFrustum()
    - gluPerspective()
  - GL_MODELVIEW:
    - glTranslate*()
    - glRotate*()
    - glScale*()

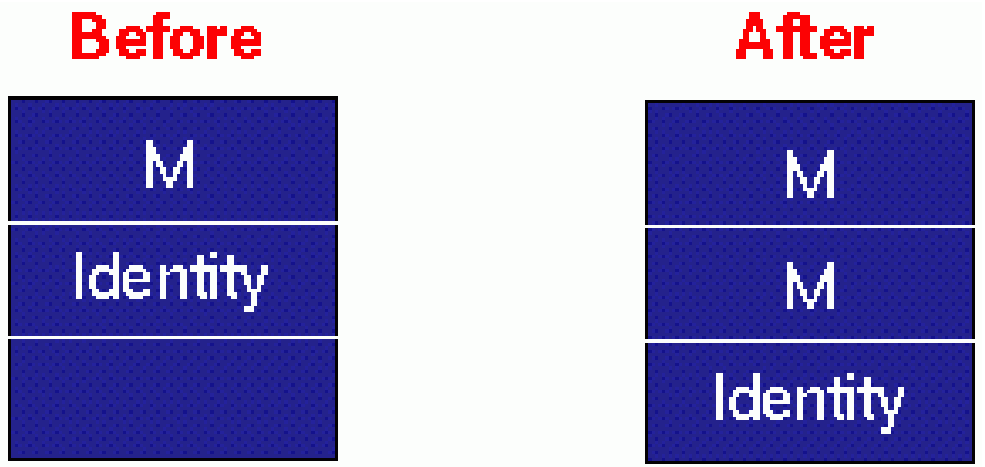
# Modos de Matriz

- A maior parte das operações é realizada no modo ModelView.
  - O sistema inicia em GL_MODELVIEW.
  - Mantenha o sistema em GL_MODELVIEW na maior parte do tempo.
  - Recomenda-se chamar as transformações de projeção como no exemplo:

**glMatrixMode( GL_PROJECTION )**
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0)
**glMatrixMode( GL_MODELVIEW )**

# Operações na Pilha de Matrizes

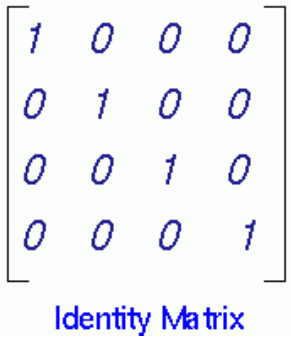
- glPushMatrix()
  - Duplica a matriz corrente e a coloca no topo da pilha.




- glPopMatrix()
  - Retira a matriz corrente do topo da pilha.

# Carregando a Matriz Identidade

- glLoadIdentity()
  - Substitui o topo da pilha corrente por uma matriz identidade.




- Outras possíveis operações com matrizes:

| Operação | Descrição |
|---|---|
| glLoadMatrix*() | Carrega a matriz especificada na pilha |
| glGetFloatv() | Retorna a matriz no topo da pilha |
| glMultMatrix*() | Multiplica a matriz corrente por uma matriz especificada |

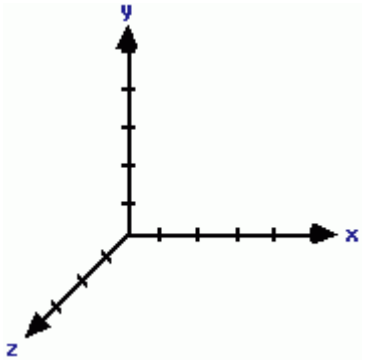
# Composição de Transformações

# Composição de transformações

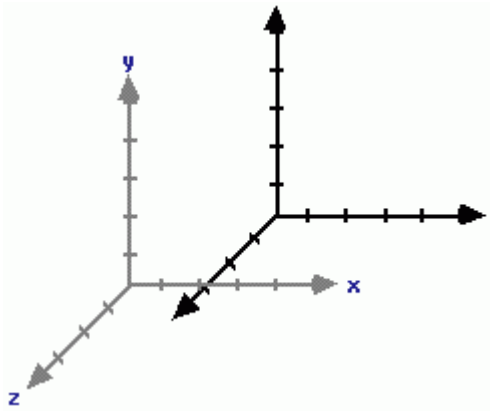
- Pense em um sistema de coordenadas móvel.
- As transformações afetam o sistema de coordenadas, não os objetos.
- Todos os modelos são desenhados relativos ao sistema de coordenadas corrente.
- O sistema de coordenadas corrente é representado pela matriz corrente.

# Sistema de coordenadas móvel

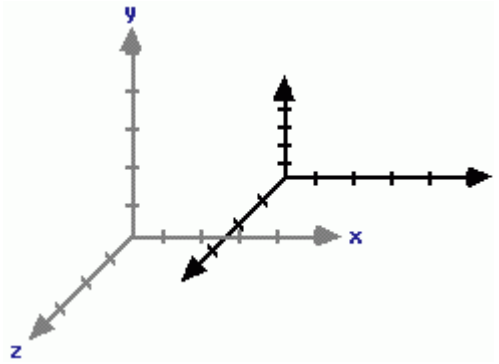
- glPushMatrix()




- glTranslatef( 4.0, 2.0, 0.0 )

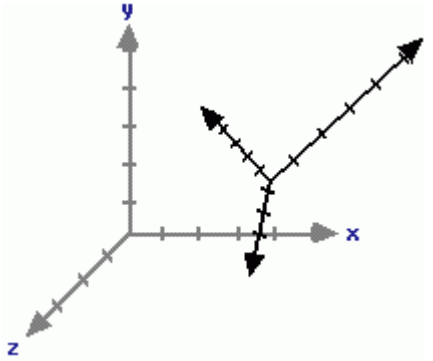



- glScalef( 1.0, 0.5, 1.0 )

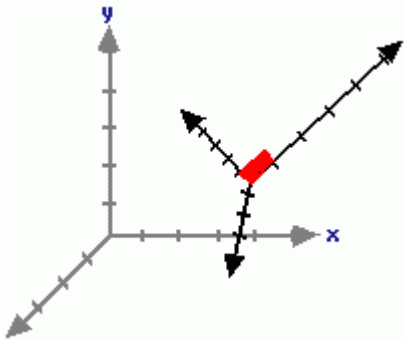


# Sistema de coordenadas móvel

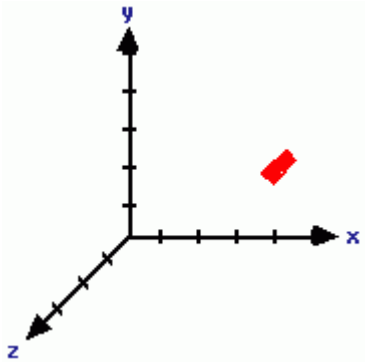
- glRotatef( 45.0, 0.0, 0.0, 1.0 )




- draw_unit_square_box()




- glPopMatrix()



# Ordem das transformações

- A ordem de execução das transformações é importante:
  - Ex.:

```
glTranslatef( 1.0, 0.0, 0.0 )
glRotatef( 60.0, 0.0, 0.0, 1.0 )
```

é diferente de:

```
glRotatef( 60.0, 0.0, 0.0, 1.0 )
glTranslatef( 1.0, 0.0, 0.0 )
```

# Ordem das transformações

- glRotatef( 90.0, 0.0, 0.0, 1.0 )
  glTranslatef( 4.0, 2.0, 0.0 )
  glVertex3f( 1.0, 2.0, 3.0 )

$$\begin{bmatrix} 1 & 0 & 0 & 4.0 \\ 0 & 1 & 0 & 2.0 \\ 0 & 0 & 1 & 0.0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 90 & -\sin 90 & 0 & 0 \\ \sin 90 & \cos 90 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1.0 \\ 2.0 \\ 3.0 \\ 1.0 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix}$$

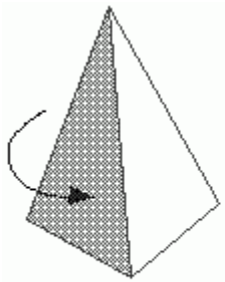
Translate Rotate Vertex Transformed vertex

# Efeitos da ordem de transformação

| Second Transformation / First Transformation | Rotate | Translate | Scale |
|---|---|---|---|
| Rotate | Spin around the origin one way, then another | Spin around the origin, then move the origin to a new point in the rotated world | Spin around the origin, then stretch/ compress the world |
| Translate | Move the origin to a new position, then spin around the origin | Move the origin to a new position, then move the origin to another new position | Move the origin to a new position, then stretch/compress the world |
| Scale | Stretch/compress the world, then spin around the origin | Stretch/compress the world, then move the origin using the scaled world coordinates | Stretch/compress the world, then stretch/ compress it again using the scaled world coordinates |

# Modelos independentes

- Modelos independentes passam por transformações independentes:
  - Push para salvar o sistema de coordenadas corrente;
  - Pop para retornar ao sistema de coordenadas salvo anteriormente.
- Push/pop é usado para isolar objetos dos efeitos de um conjunto de transformações aplicados em outros objetos.

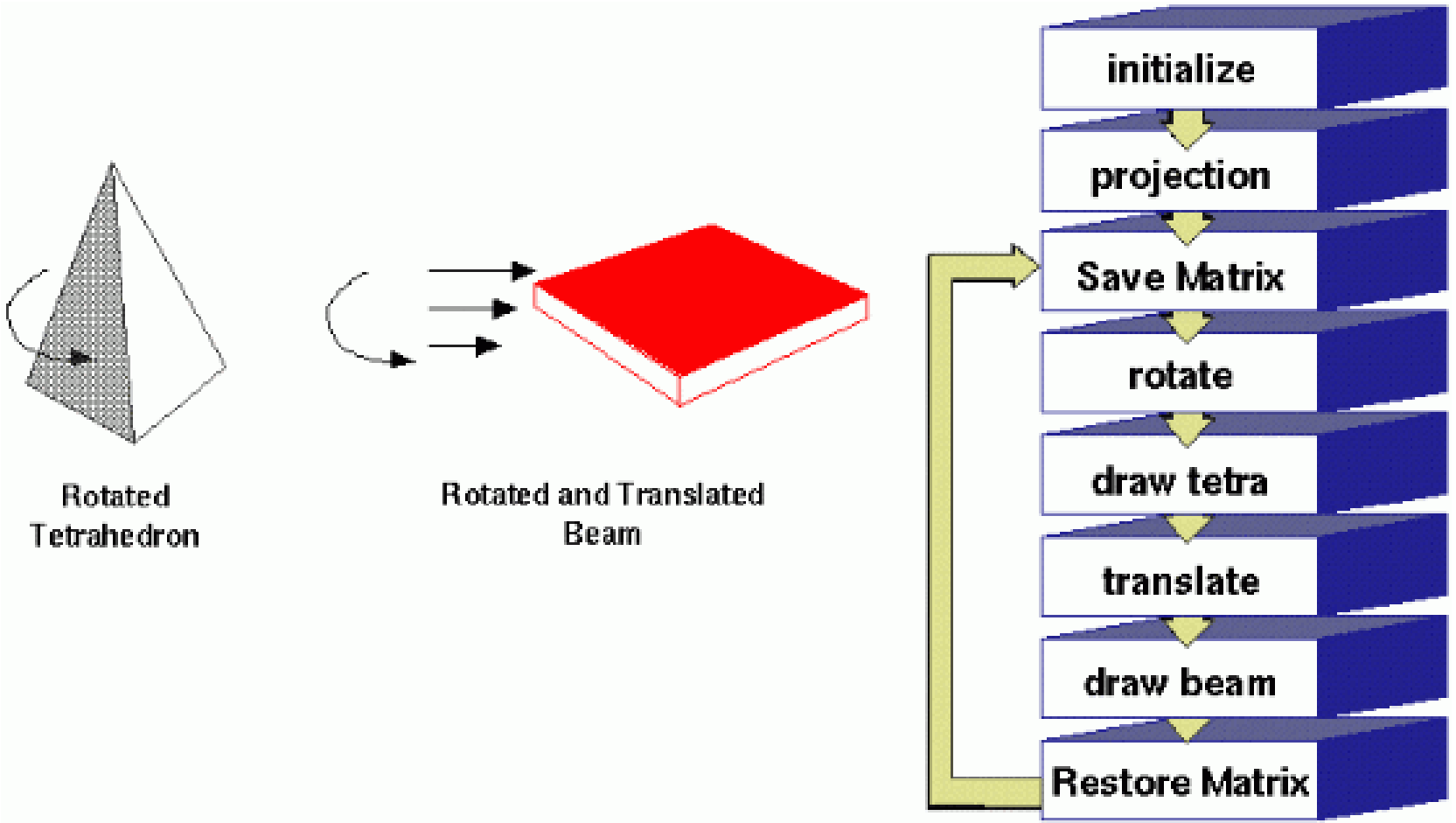


Rotated Tetrahedron

Translated Beam

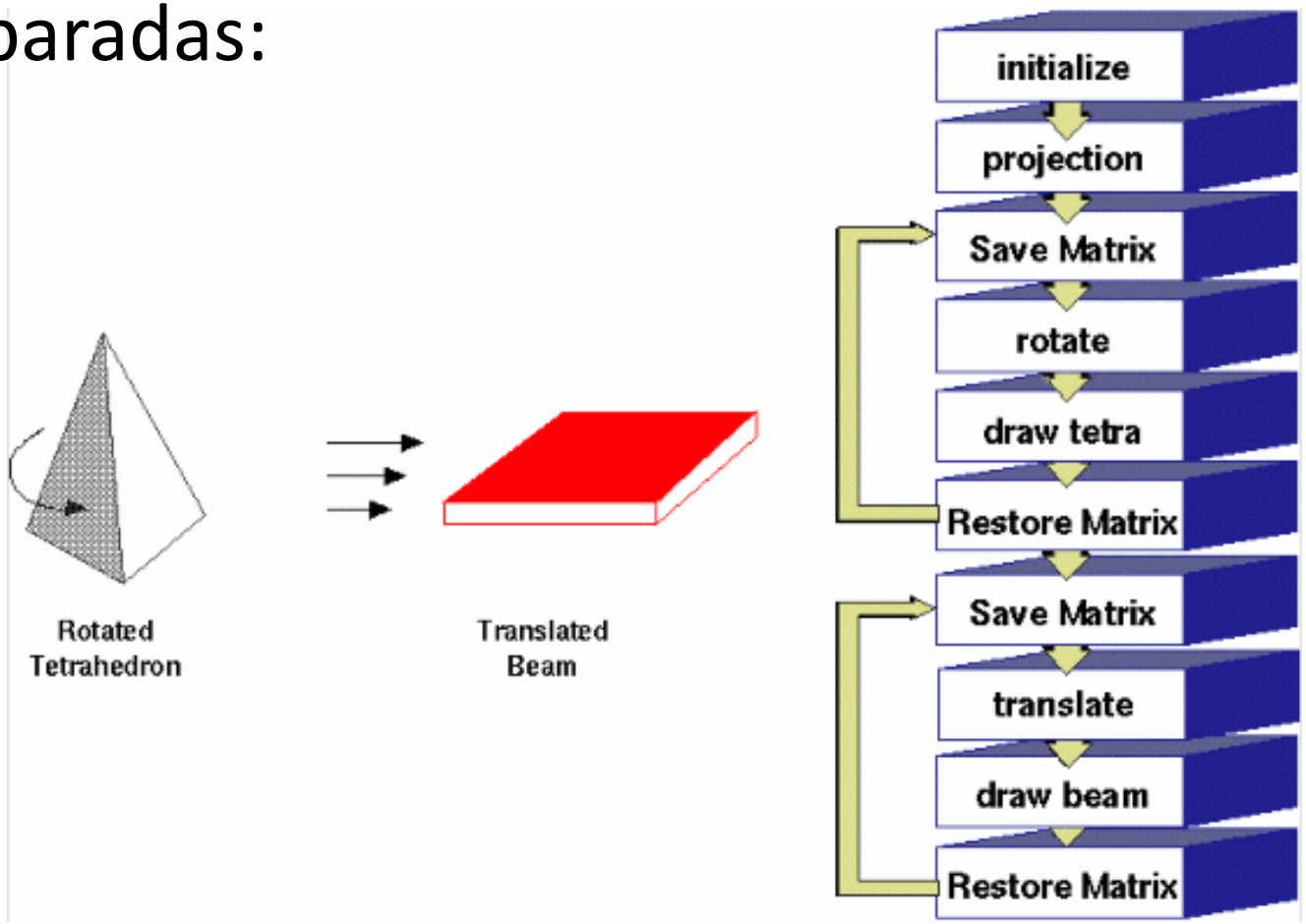
# Exemplo de modelos que *não* são independentes

- As transformações se acumulam:



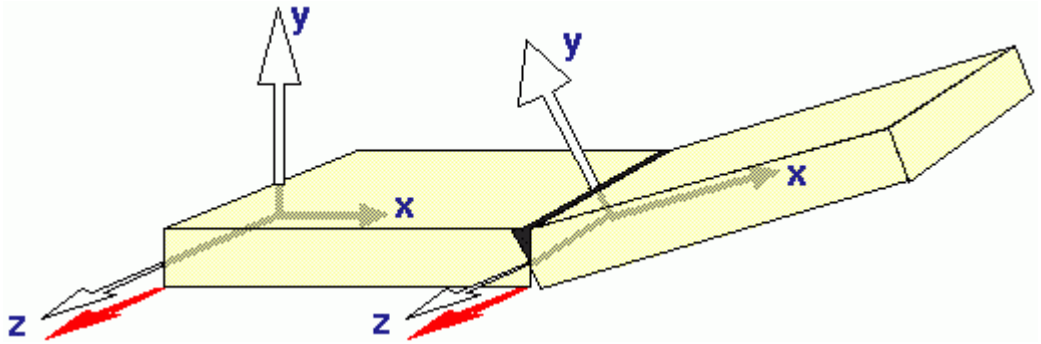
# Exemplo de modelos que são independentes

- Cada modelo é afetado por transformações separadas:



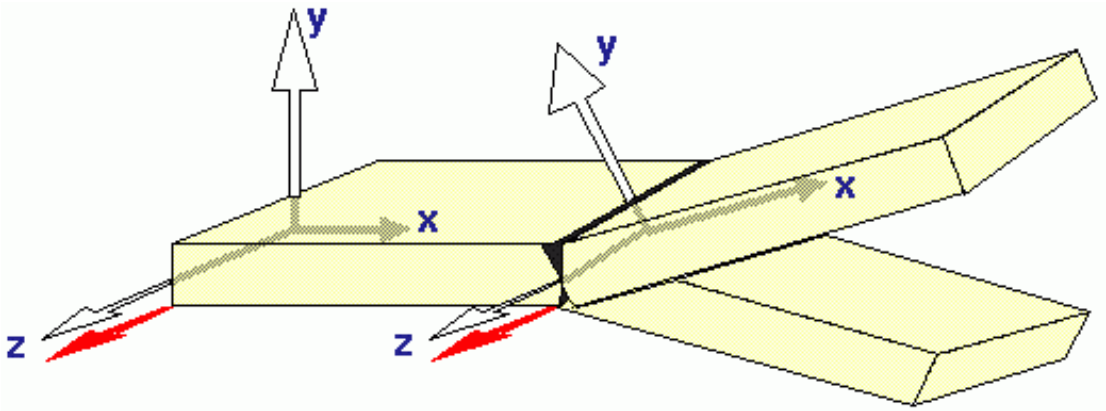
# Exemplo de composição: Garra do robô

1. Desenhe uma caixa;
2. Mova para o cotovelo;
3. Rotacione 45° em torno do cotovelo;
4. Desenhe uma caixa.



# Exemplo de composição: Garra do robô

Como posicionar a terceira parte?



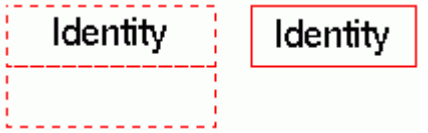
# WireBox

- Como desenhar uma caixa?
  - Desenhe um cubo “achatado”.

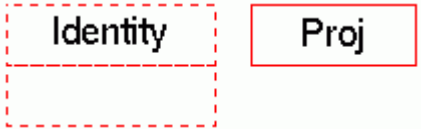
```
def wireBox(dx, dy, dz):
      glPushMatrix()
      glTranslatef(dx/2, 0, 0)
      glScalef(dx, dy, dz)
      glutWireCube(1)
      glPopMatrix()
```

# Exemplo de composição: Garra do robô

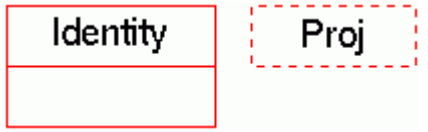
- glMatrixMode( GL_PROJECTION )



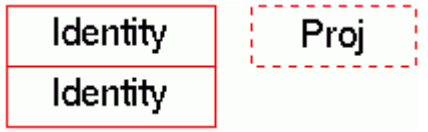

- gluPerspective(60.0,1.0,1.0,20.0)



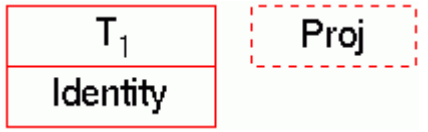

- glMatrixMode( GL_MODELVIEW )




- glPushMatrix()



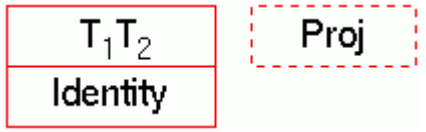

- glTranslatef( 0.0, 0.0, -8.0 )



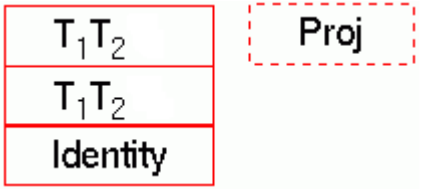

- wireBox( 2.0, 0.4, 1.0 )

# Exemplo de composição: Garra do robô

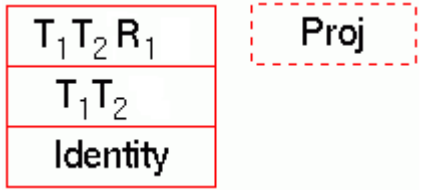
- glTranslatef( 2.0, 0.0, 0.0 )




- glPushMatrix()

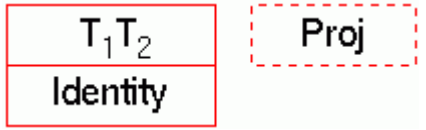



- glRotatef( 45.0, 0.0, 0.0, 1.0 )



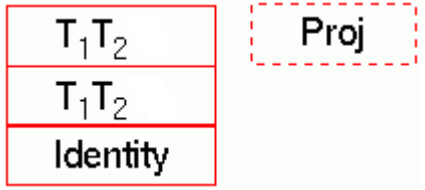

- wireBox( 2.0, 0.4, 1.0 )

- glPopMatrix()

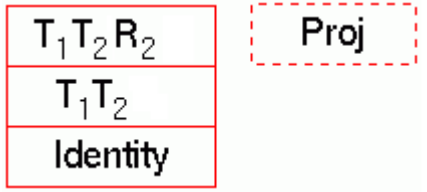


# Exemplo de composição: Garra do robô

- glPushMatrix()

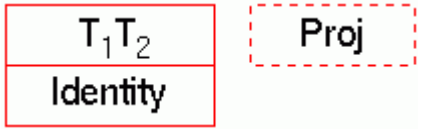



- glRotatef( -45.0, 0.0, 0.0, 1.0 )




- wireBox( 2.0, 0.4, 1.0 )

- glPopMatrix()




- glPopMatrix()

# Referências

- Transformações de visualização, projeção e modelagem: http://profs.sci.univr.it/~colombar/html_openGL_tutorial/en/03basic_transforms_000.html