



Introdução à Computação Gráfica

Marcel P. Jackowski
mjack@ime.usp.br

Aula #6: Projeções



Objetivos

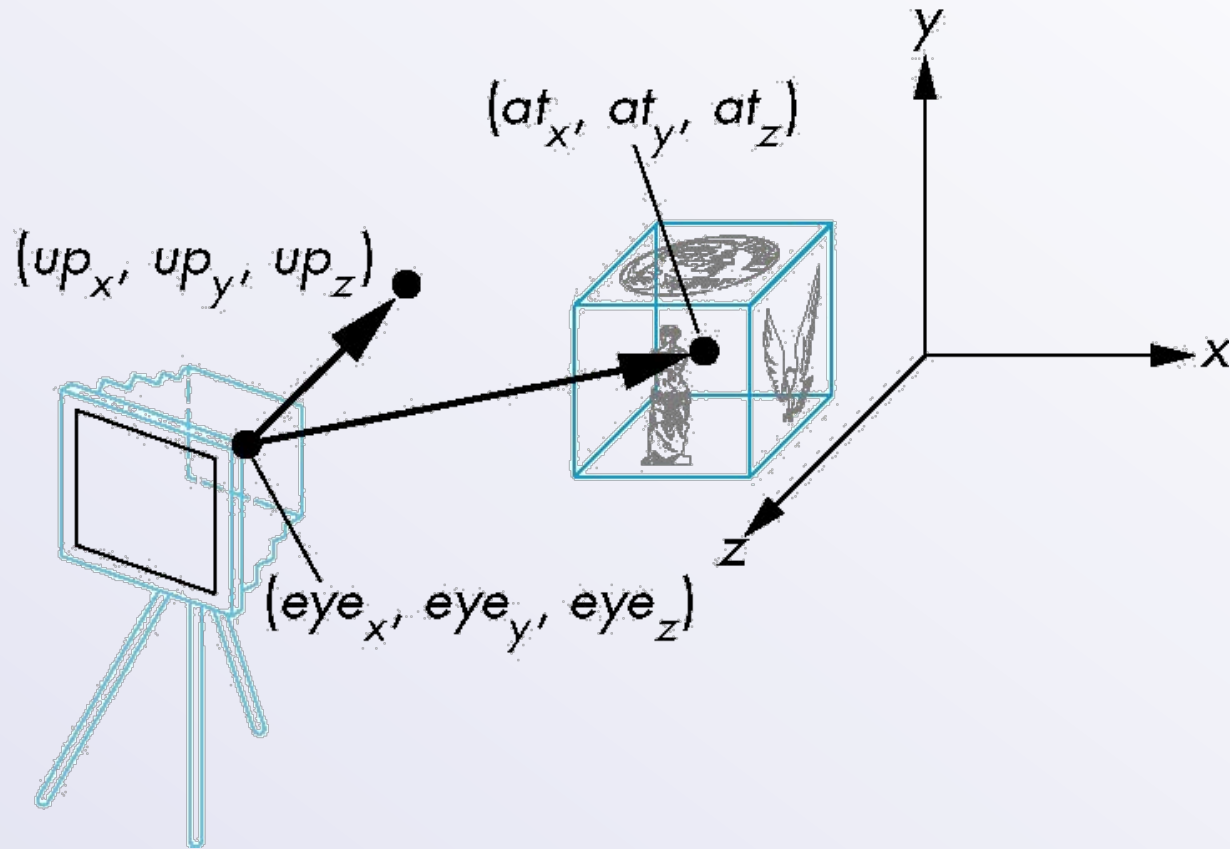
- Especificação da câmera em OpenGL
- Taxonomia das projeções
- Projeções (ortográfica e perspectiva)
- Normalização das projeções
- Introdução à animação
- Sólidos platônicos

A função gluLookAt()

- A biblioteca GLU oferece a função gluLookAt para formar a matriz de modelo através de parâmetros simples.
- Mas precisamos definir o vetor “up” (cima).

```
glMatrixMode(GL_MODELVIEW) ;  
glLoadIdentity() ;  
gluLookAt(1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0,  
1.0, 0.0) ;
```

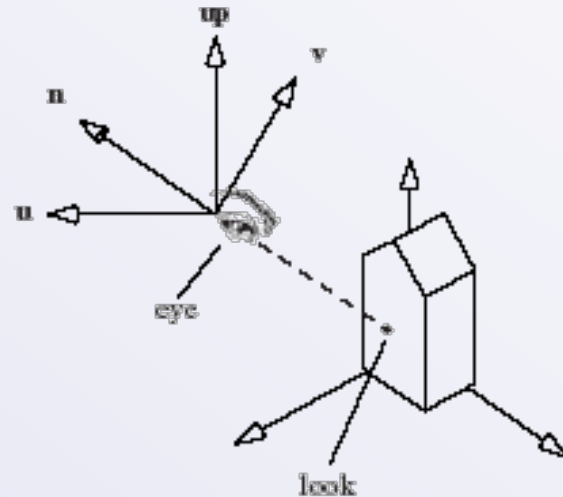
gluLookAt()



`gluLookAt(eyex, eyeey, eyez, atx, aty, atz, upx, upy, upz)`

gluLookAt()

- `gluLookAt` cria um sistema de coordenadas da câmera (SRC) que consiste em de três vetores ortogonais: **u**, **v**, and **n**.
 - $\mathbf{n} = \text{eye} - \text{at}$
 - $\mathbf{u} = \mathbf{up} \times \mathbf{n}$;
 - $\mathbf{v} = \mathbf{n} \times \mathbf{u}$
- Normaliza **n**, **u**, **v**



Matriz de visualização

- Então `gluLookAt()` monta a seguinte matriz de visualização:

$$M = \begin{pmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

`gluLookAt()` é equivalente:

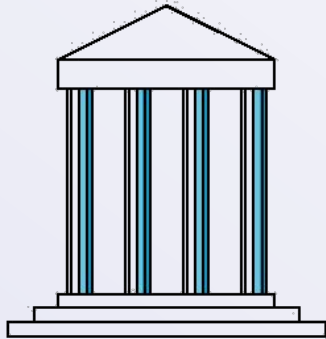
```
glMultMatrix(M);
```

```
glTranslated(-eye[0], -eye[1], -eye[2]);
```

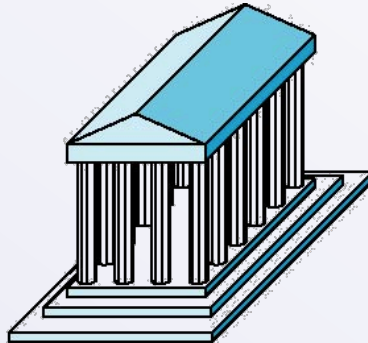
Projeções geométricas planares

- A superfície de projeção é um plano
- Projetores são linhas:
 - Podem convergir em um centro de projeção
 - São paralelas (centro de projeção no infinito)
- Preservam linhas
 - Geralmente não preservam ângulos
- Projeções não-planares são usadas em aplicações como construção de mapas
 - Projeções cartográficas

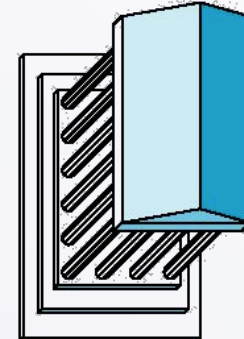
Projeções planares clássicas



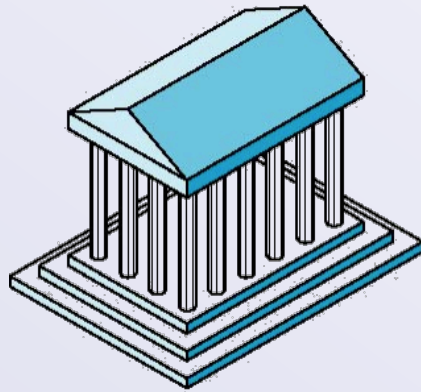
Elevação frontal



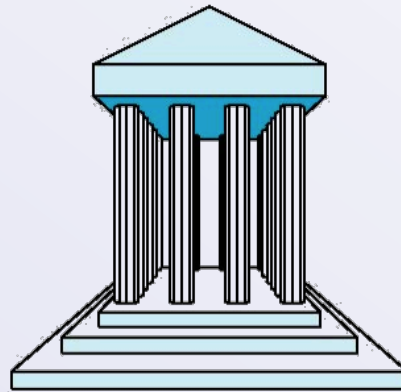
Elevação oblíqua



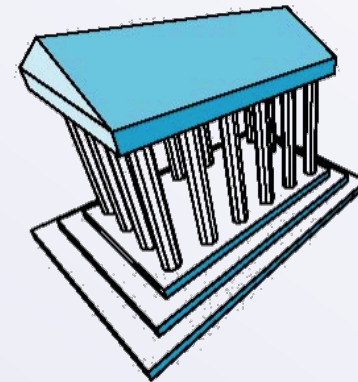
Plano oblíquo



Isométrica



Perspectiva de 1 ponto

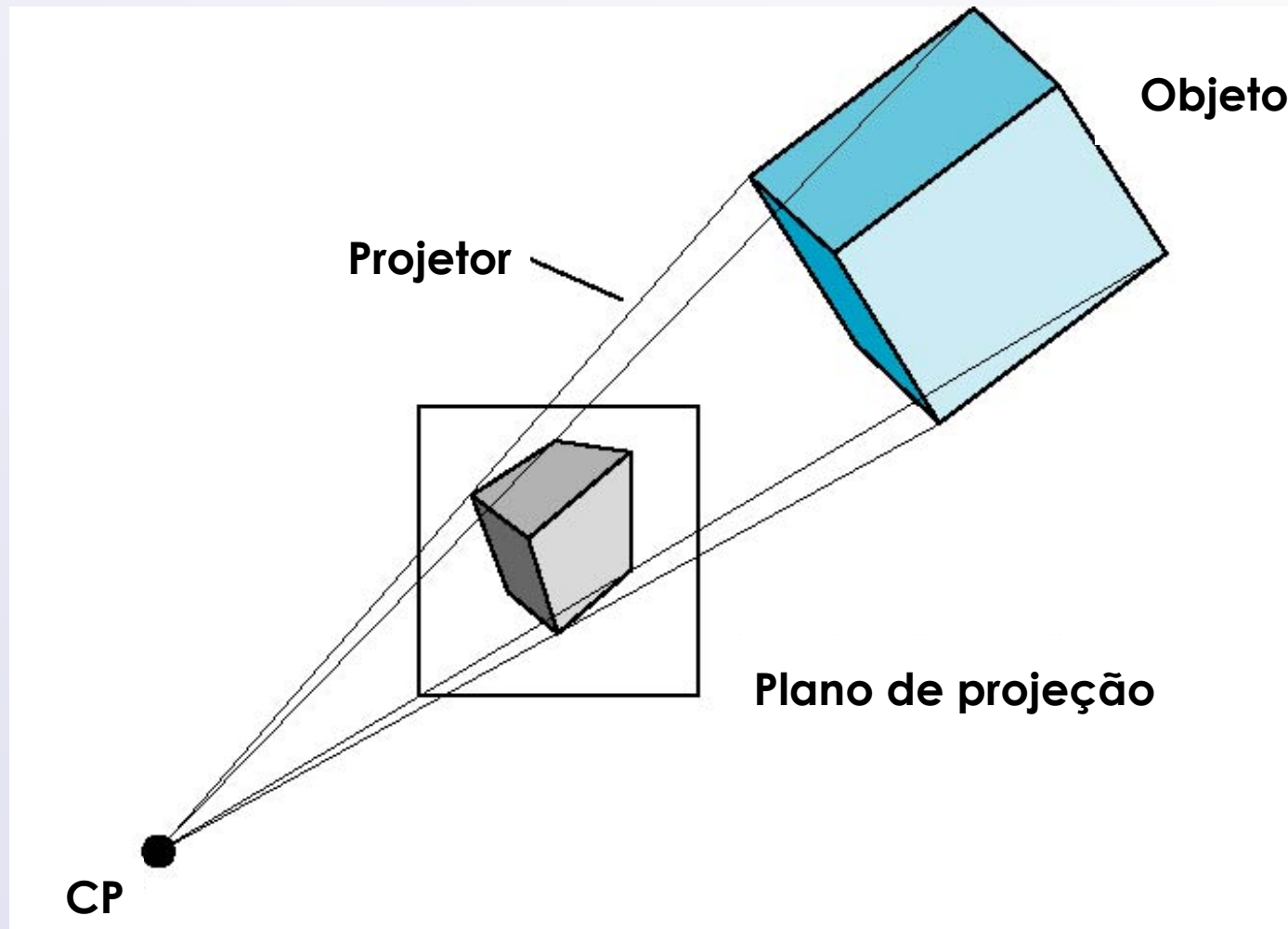


Perspectiva de 3 pontos

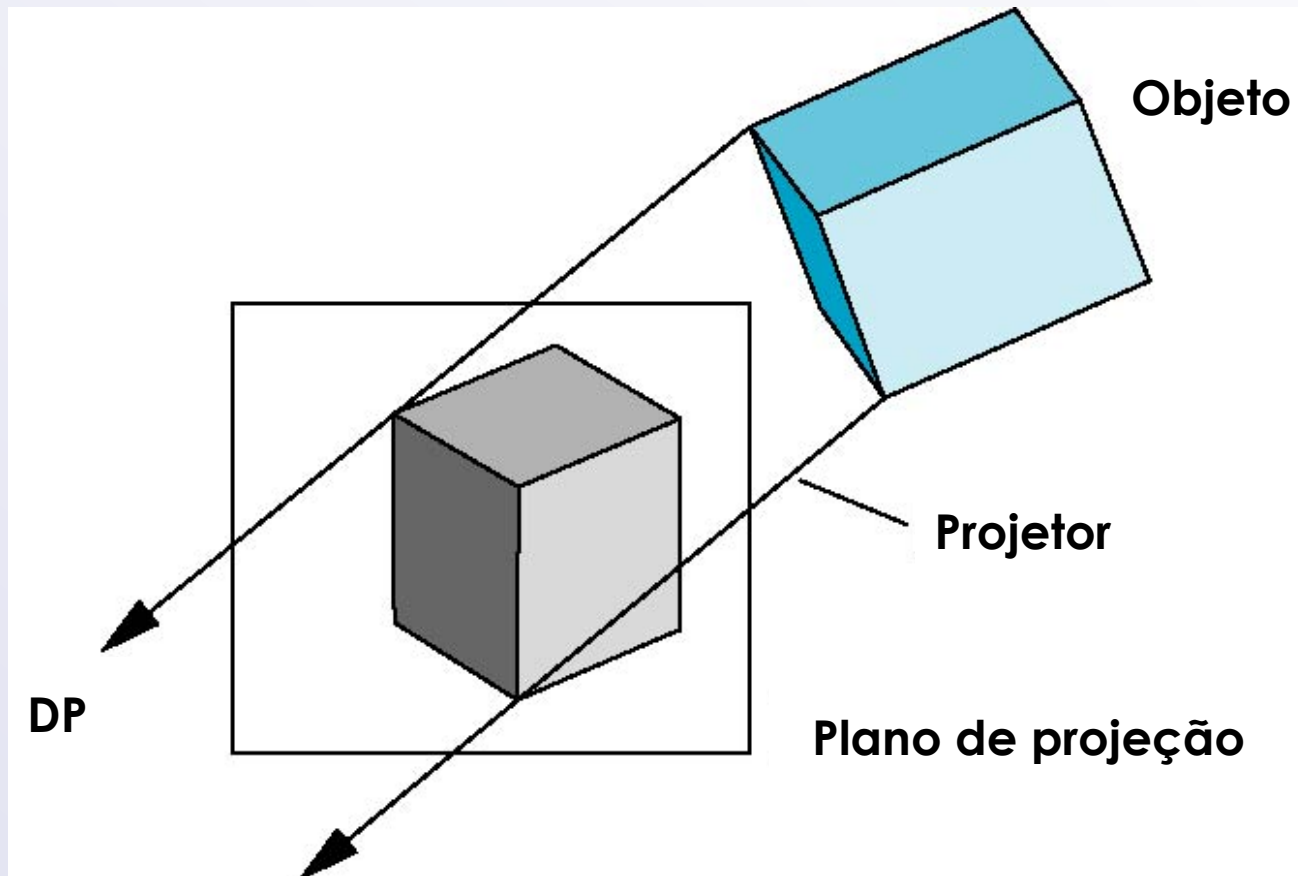
Visualização clássica versus CG

- Visualização clássica
 - Técnicas específicas para cada tipo de projeção
 - Diversidade de projeções
 - Dependência no relacionamento entre objeto, observador e plano de projeção
- Computação gráfica
 - Projeção paralela e perspectiva
 - Um único pipeline de visualização
 - Independência de especificações

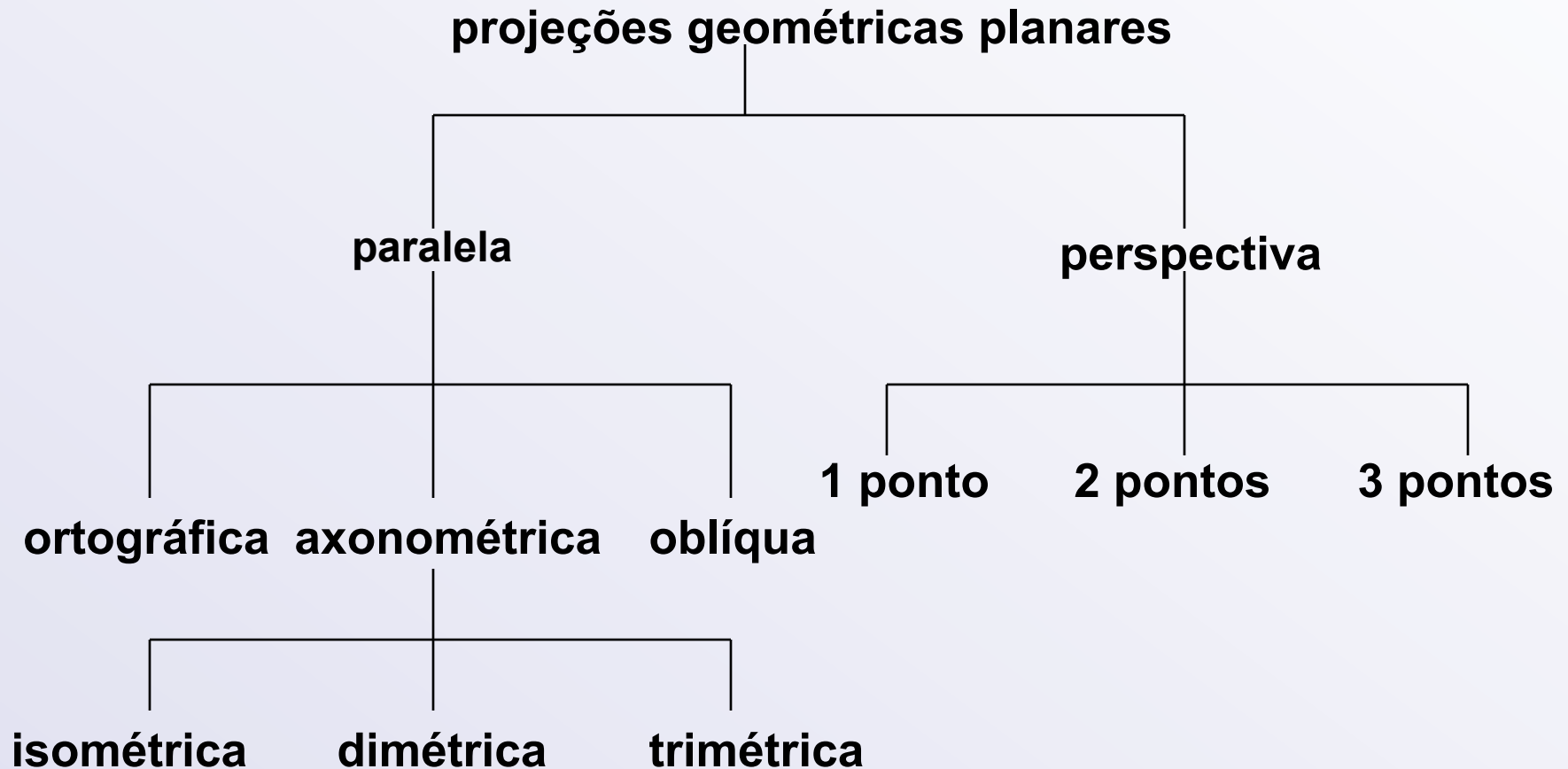
Projeção perspectiva



Projeção paralela

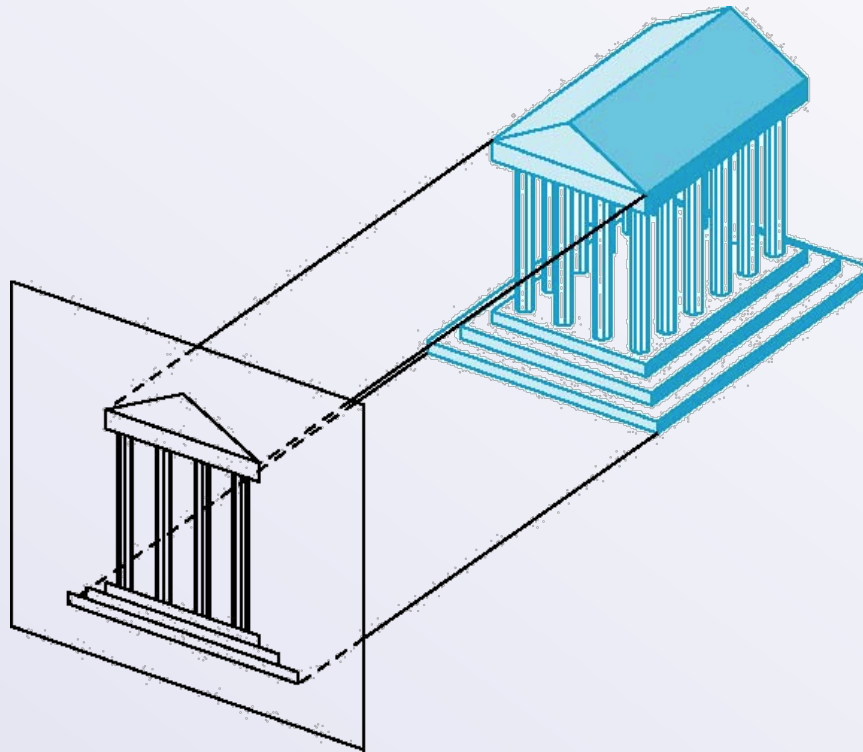


Taxonomia das projeções



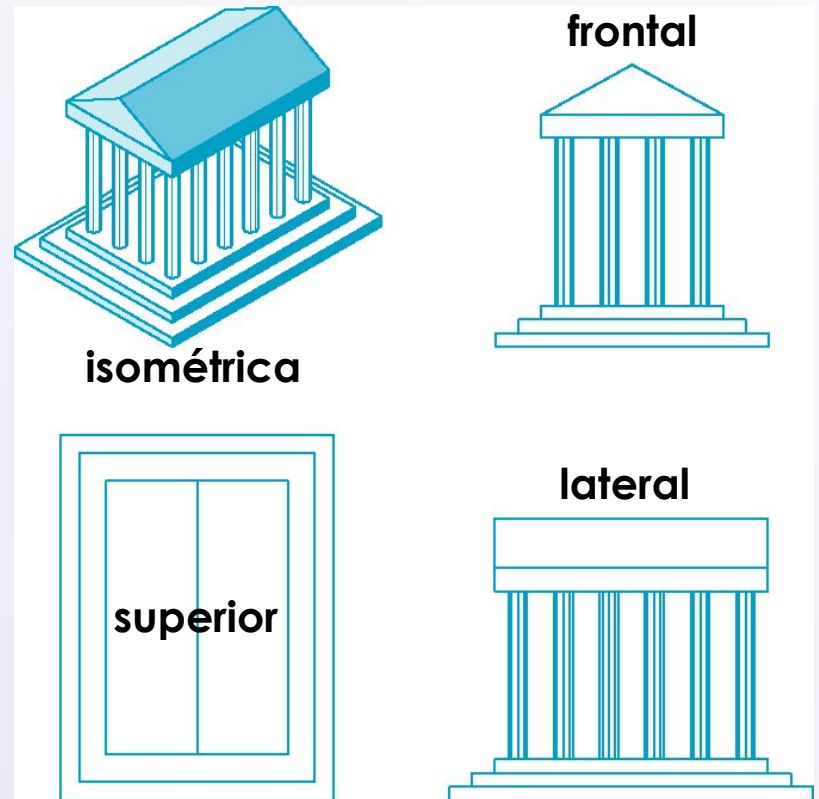
Projeção ortográfica

Projetores são ortogonais à superfície de projeção



Ortográfica multi-visão

- O plano de projeção é paralelo à face principal
- Normalmente formam-se as visões frontal, superior, e lateral
- Usado em CAD, engenharia e arquitetura
- Utilizada em conjunto com a projeção isométrica

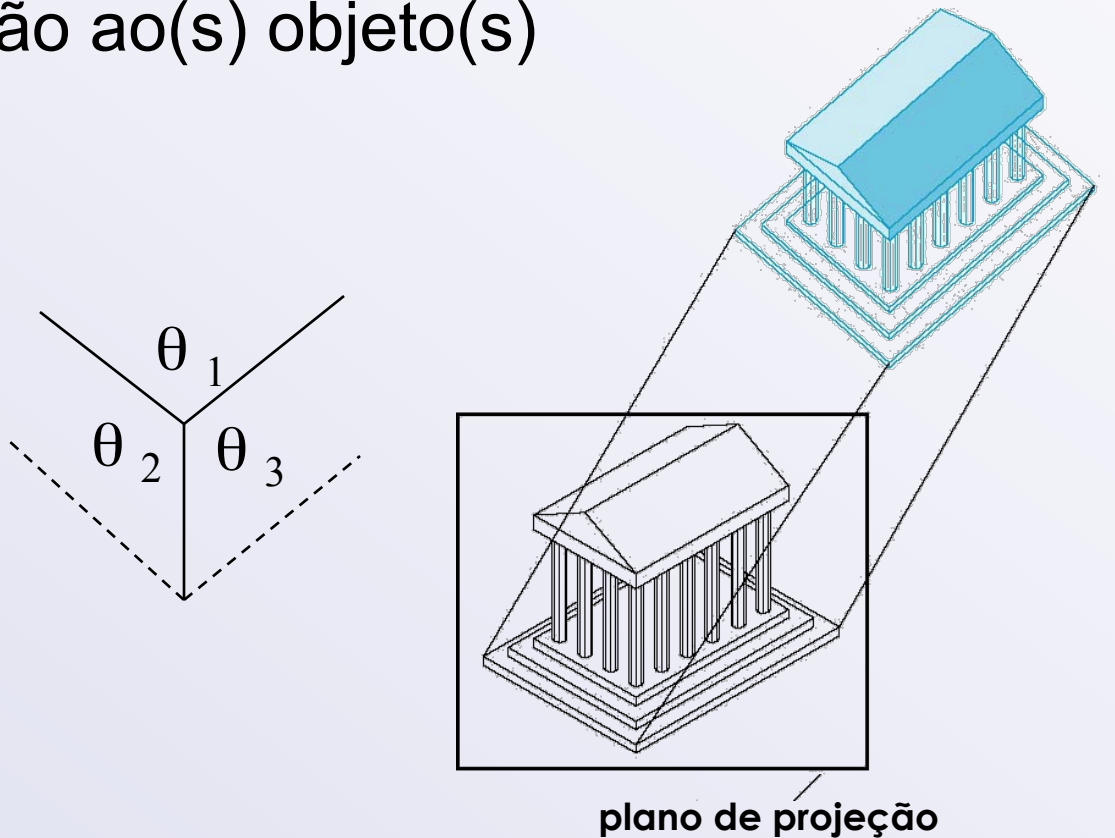


Vantagens e desvantagens

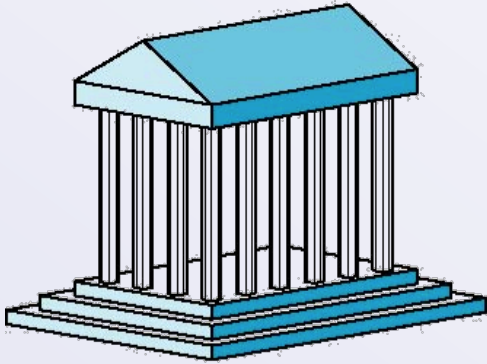
- Preservam distâncias e ângulos
 - Formas são preservadas
 - Podem ser usados para mensuração
 - Plantas (prédios, casas, etc)
 - Manuais
- Impossibilidade de visualizar como o objeto aparece em 3D
 - Quase sempre adicionamos a visão isométrica

Axonométricas

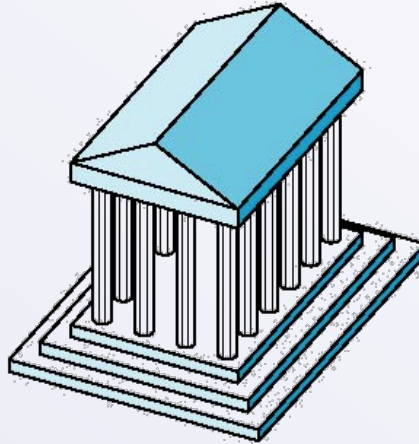
O plano de projeção pode ter qualquer orientação em relação ao(s) objeto(s)



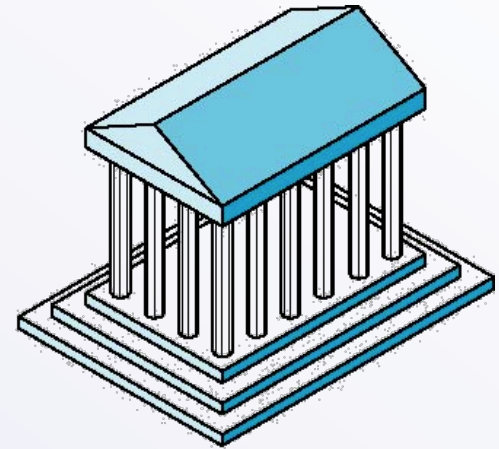
Axonométricas



dimétrica



trimétrica



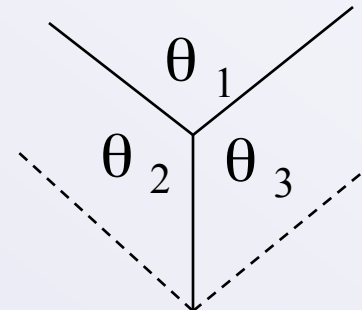
isométrica

Quantos ângulos de um canto de um cubóide projetado são iguais ?

nenhum: trimétrica

dois: dimétrica

três: isométrica

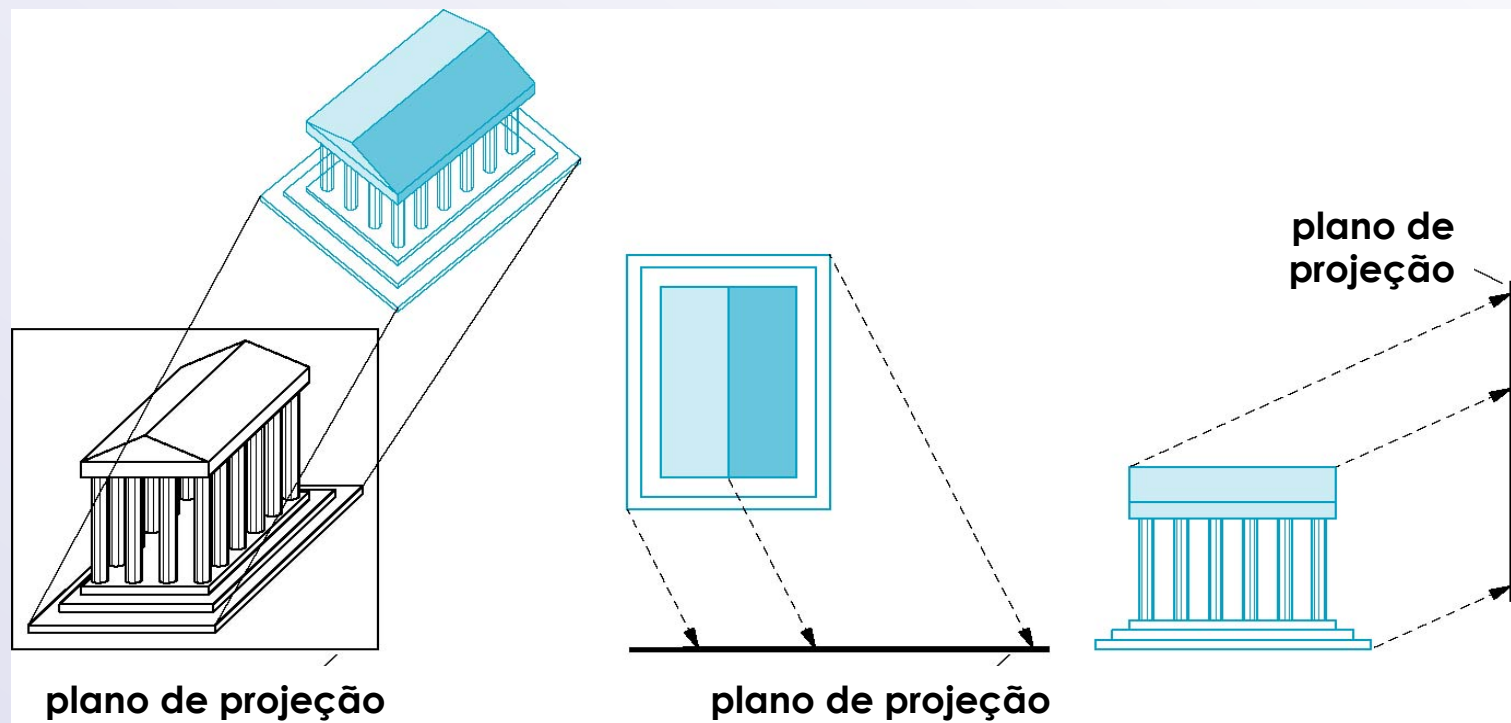


Vantagens e desvantagens

- Linhas são preservadas mas são escaladas
- Ângulos não são preservados
- A projeção de um círculo em um plano não paralelo ao plano de projeção se torna uma elipse
- Permite visualizar os eixos principais de objetos
- Não parece real, pois a escala é a mesma para objetos próximos ou distantes do observador
- Usados em arquitetura ou desenho mecânico

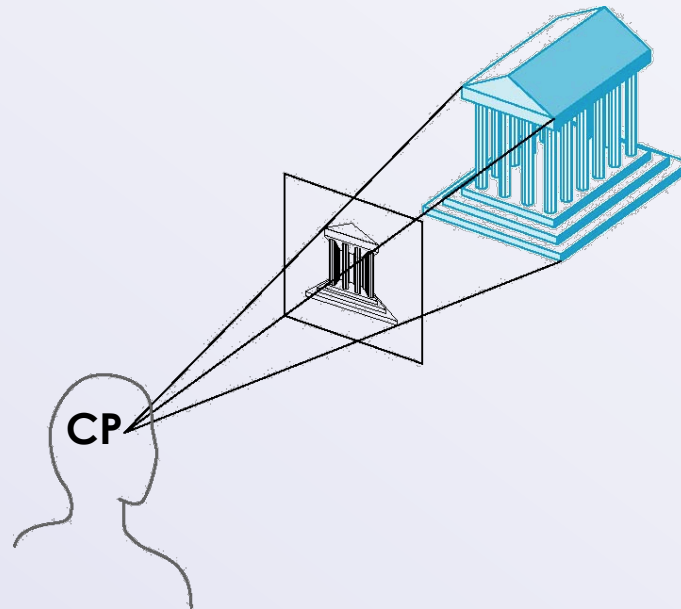
Projeção oblíqua

Relacionamento arbitrário entre os projetores e plano de projeção



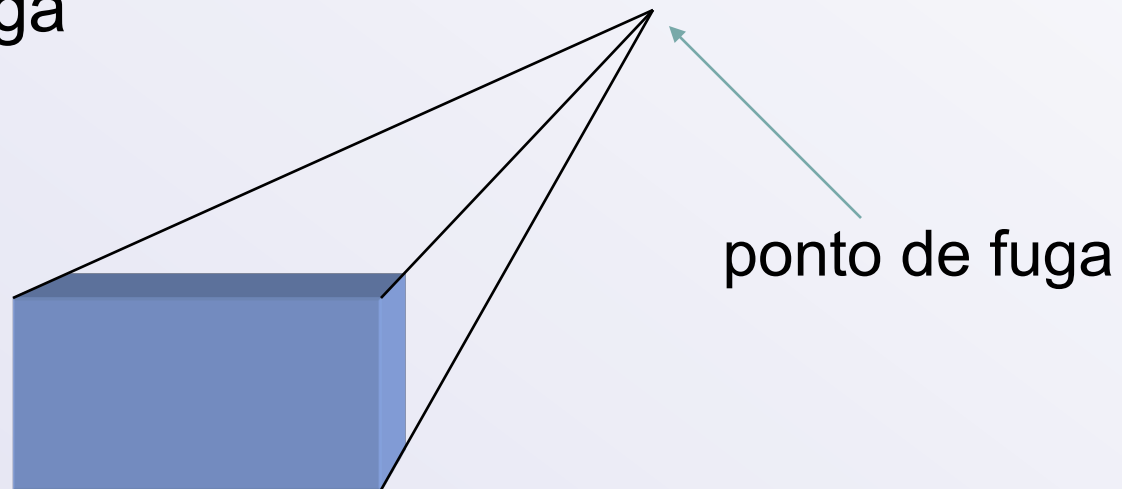
Projeção perspectiva

- Projetores convergem no centro de projeção
- Caracterizados pela diminuição de tamanho



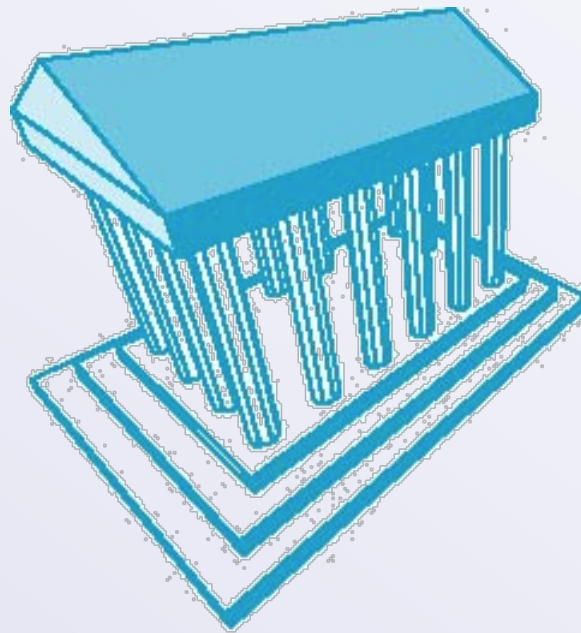
Pontos de fuga

- Linhas não paralelas ao plano de projeção partem do objeto e convergem em um único ponto (*ponto de fuga*)
- O desenho manual de perspectivas simples usam pontos de fuga



Perspectiva de 3 pontos

- Nenhuma direção principal é paralela ao plano de projeção
- Três pontos de fuga para um cubóide



Perspectiva de 2 pontos

- Uma direção principal paralela ao plano de projeção
- Dois pontos de fuga para um cubóide



Perspectiva de 1 ponto

- Uma face principal paralela ao plano de projeção
- Um único ponto de fuga para um cubóide



Vantagens e desvantagens

- Objetos distantes do observador são projetados em escala menor que objetos perto do observador
 - Realismo
- As projeções de duas distâncias iguais em uma linha quando projetadas não serão mais iguais
- Ângulos somente são preservados em planos paralelos ao plano de projeção
- Manualmente, mais difícil de construir que as projeções ortogonais

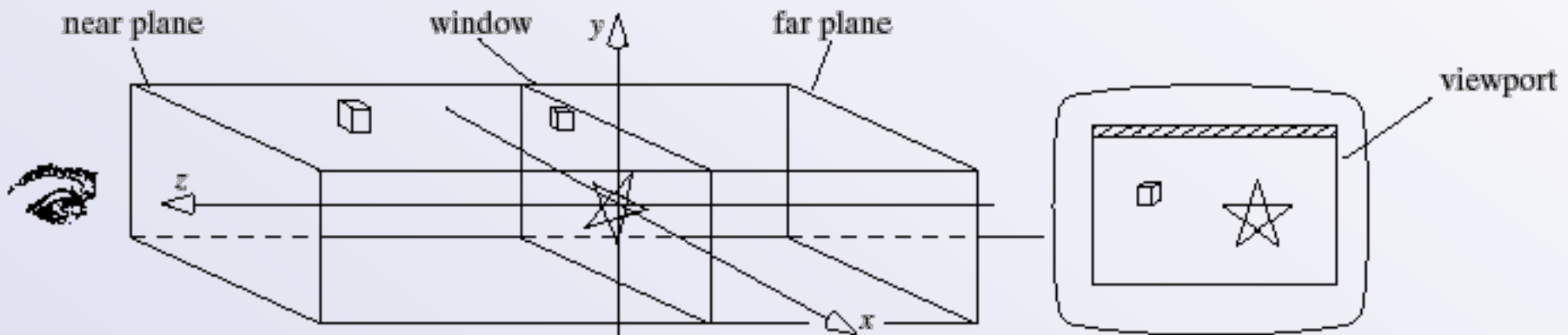
Projeções e normalização

- A projeção default da câmera é a ortogonal.
- Para pontos no volume de visualização default

$$x_p = x$$

$$y_p = y$$

$$z_p = 0$$



Matriz de projeção ortogonal

projeção ortogonal default

$$\begin{aligned}x_p &= x \\y_p &= y \\z_p &= 0 \\w_p &= 1\end{aligned}$$

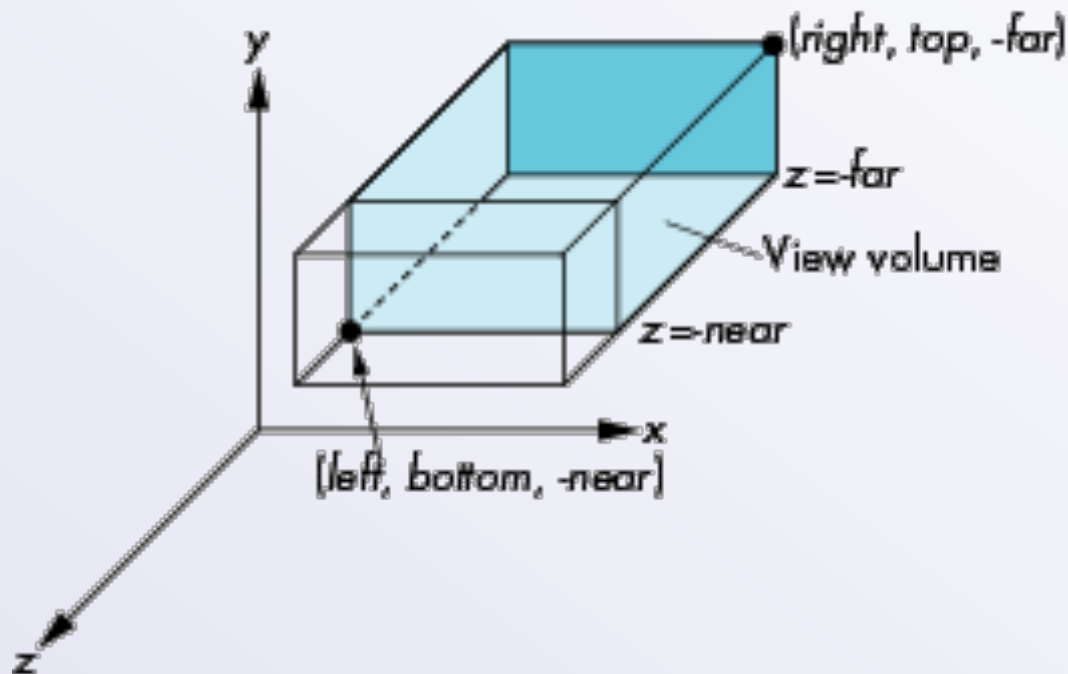
$$\mathbf{p}_p = \mathbf{M}\mathbf{p}$$

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Na prática, podemos fazer $\mathbf{M} = \mathbf{I}$ e zerar o termo z depois

Projeção ortogonal

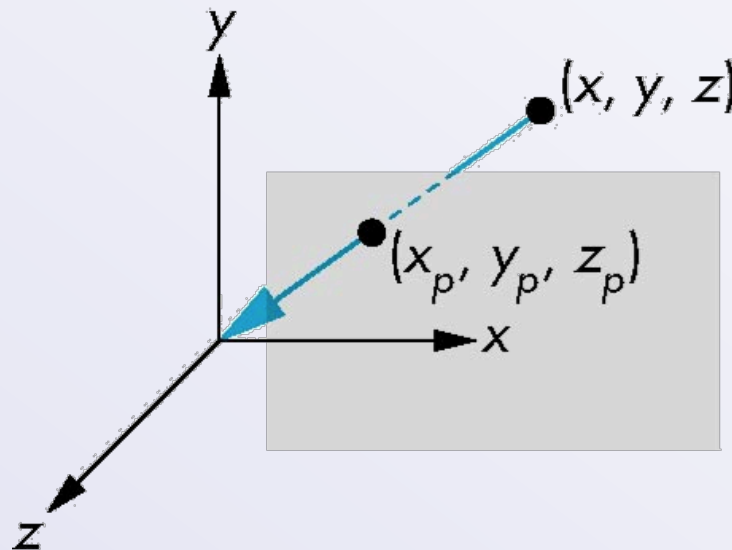
`glOrtho(left, right, bottom, top, near, far)`



near e **far** medidos a partir da posição da câmera

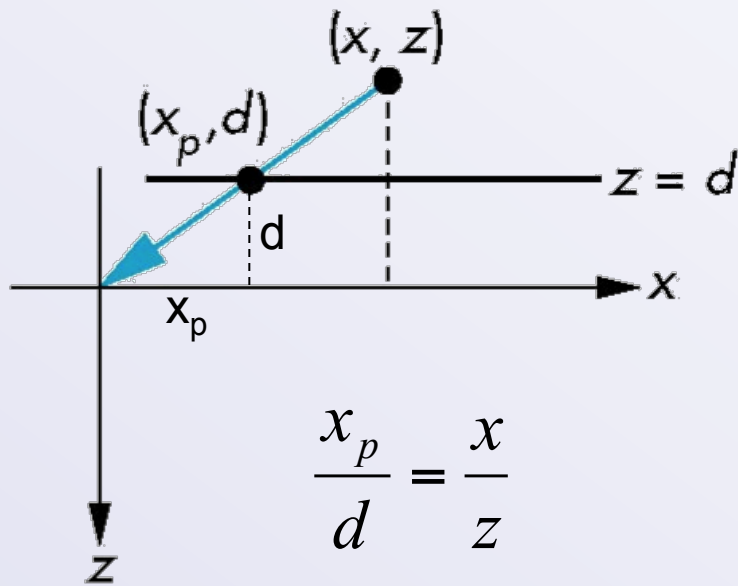
Projeção perspectiva

- Centro da projeção na origem
- Plano de projeção $z = d, d < 0$



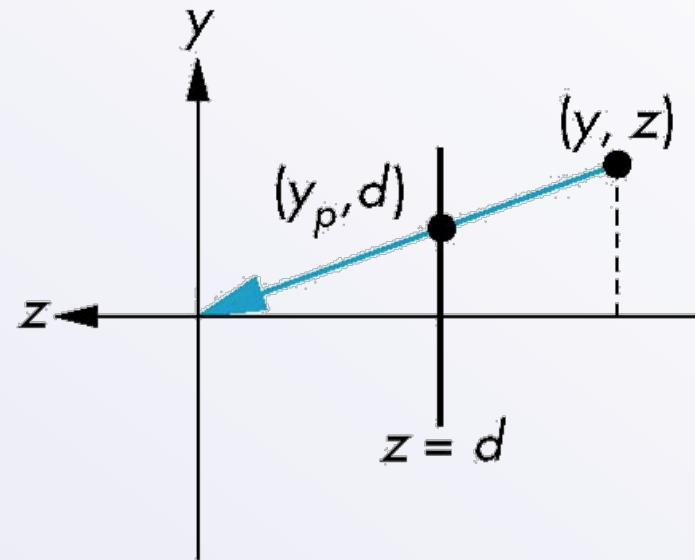
Equações de perspectiva

Visões superior e lateral



$$\frac{x_p}{d} = \frac{x}{z}$$

$$x_p = \frac{x}{z/d} \quad y_p = \frac{y}{z/d} \quad z_p = d$$



Matriz de projeção perspectiva

considere $\mathbf{q} = \mathbf{M}\mathbf{p}$ onde

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \mathbf{q} = \begin{bmatrix} x \\ y \\ z \\ z/d \end{bmatrix}$$

Normalização da perspectiva

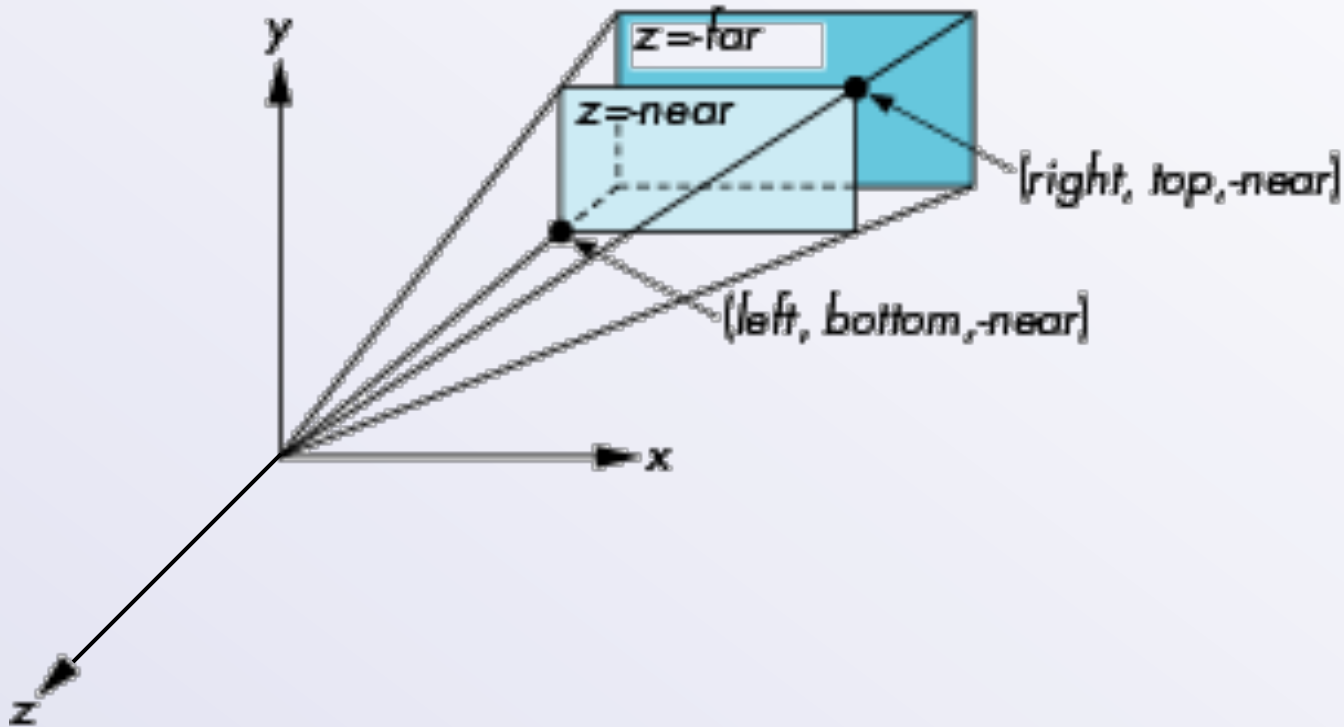
- Note que $w = 1$, então precisamos dividir por w para retornar às coordenadas homogêneas
- Essa divisão gera:

$$x_p = \frac{x}{z/d} \quad y_p = \frac{y}{z/d} \quad z_p = d$$

que são as equações desejadas.

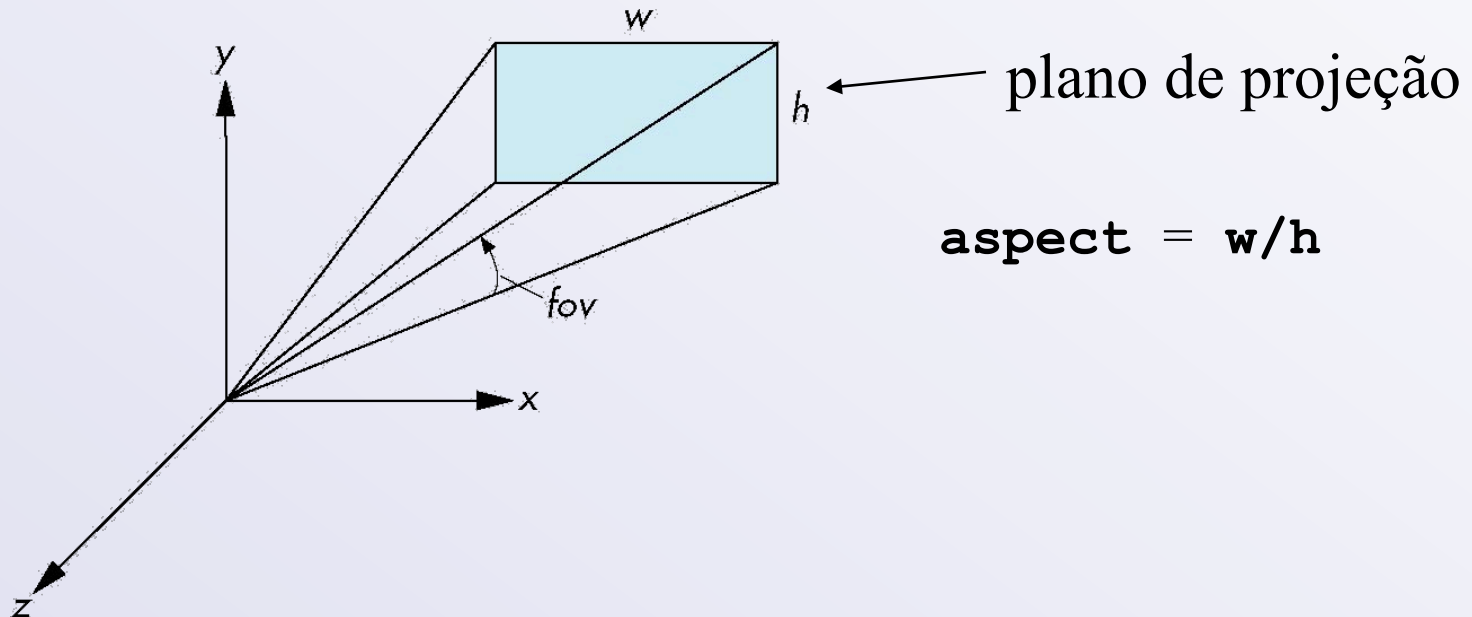
Projeção perspectiva

`glFrustum(left, right, bottom, top, near, far)`



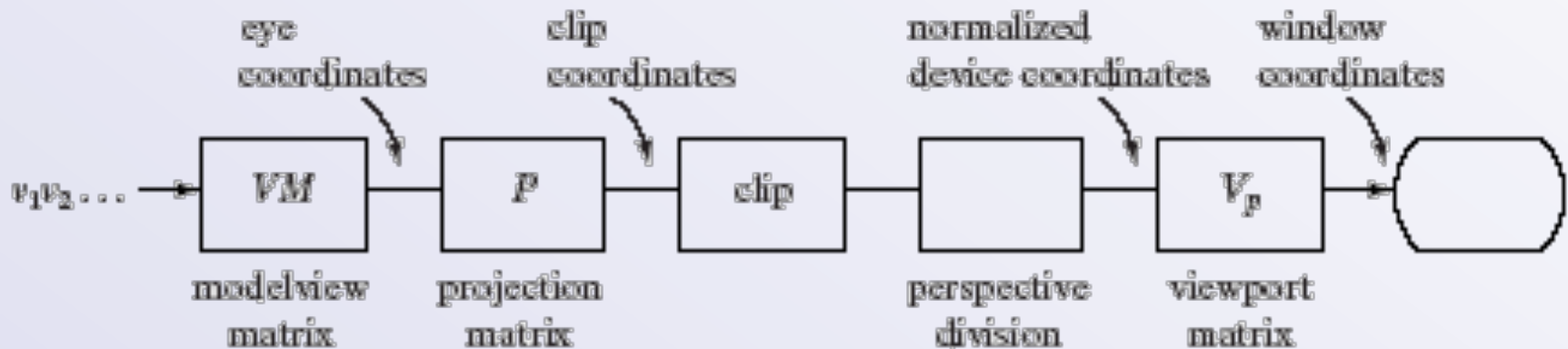
Usando campo de visão (fov)

- Com `glFrustum` é um pouco complicado obter a visualização desejada
- `gluPerspective(fovy, aspect, near, far)` já oferece uma opção mais simples.



Pipeline de visualização

- Vértices iniciam em coordenadas do universo;
- Após MV, em coordenadas do observador,
- Após P, em coordenadas de recorte;
- Após a normalização de perspectiva, em coordenadas normalizadas do dispositivo;
- E, finalmente após V_p , em coordenadas da tela.



$$W = V_p * D_{div} * C_{clip} * P * V * M$$

Construindo cenas em 3D

- Desejamos transformar objetos para orientá-los e posicioná-los em uma cena.
- A biblioteca OpenGL provém as funções necessárias para construir e aplicar as matrizes de transformações necessárias.
- As pilhas de matrizes mantidas pelo OpenGL tornam mais fácil a especificação de transformações para diferentes objetos:

glMatrixMode(GL_MODELVIEW)

glPushMatrix()

Define transformação para objeto #1

Desenha objeto #1

glPopMatrix()

glPushMatrix()

Define transformação para objeto #2

Desenha objeto #2

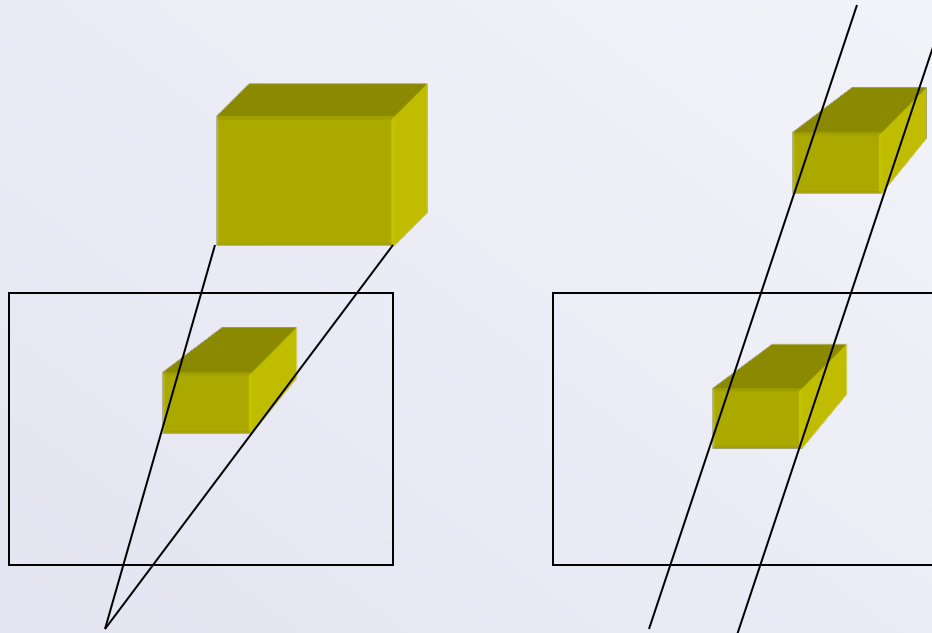
glPopMatrix()

Projeções em OpenGL

- Ao invés de derivar diferentes matrizes para cada tipo de projeção (ortogonal e perspectiva), podemos converter todas elas em projeções ortogonais usando o volume de visualização default
- Esta estratégia nos permite usar transformações padrões no pipeline e torna o processo de recorte mais eficiente.

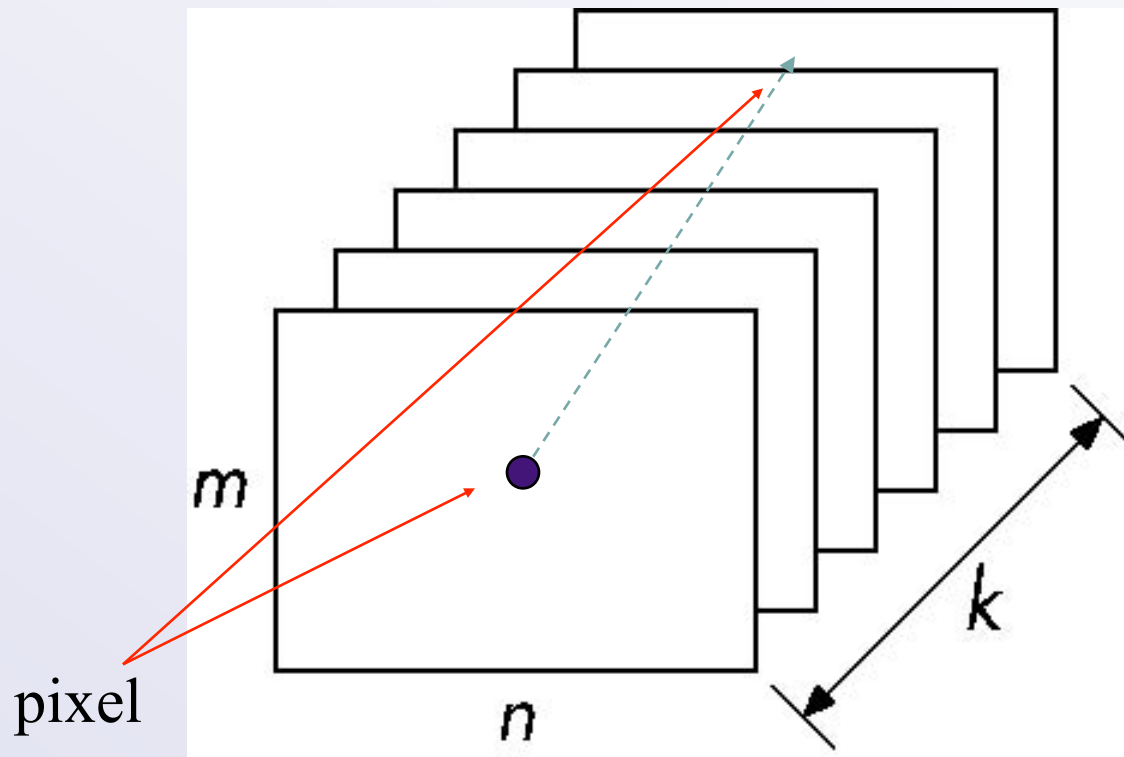
Normalização de projeção

- Primeiro, distorce a cena através de uma transformação afim
- A projeção ortográfica da cena distorcida é a mesma que a projeção original

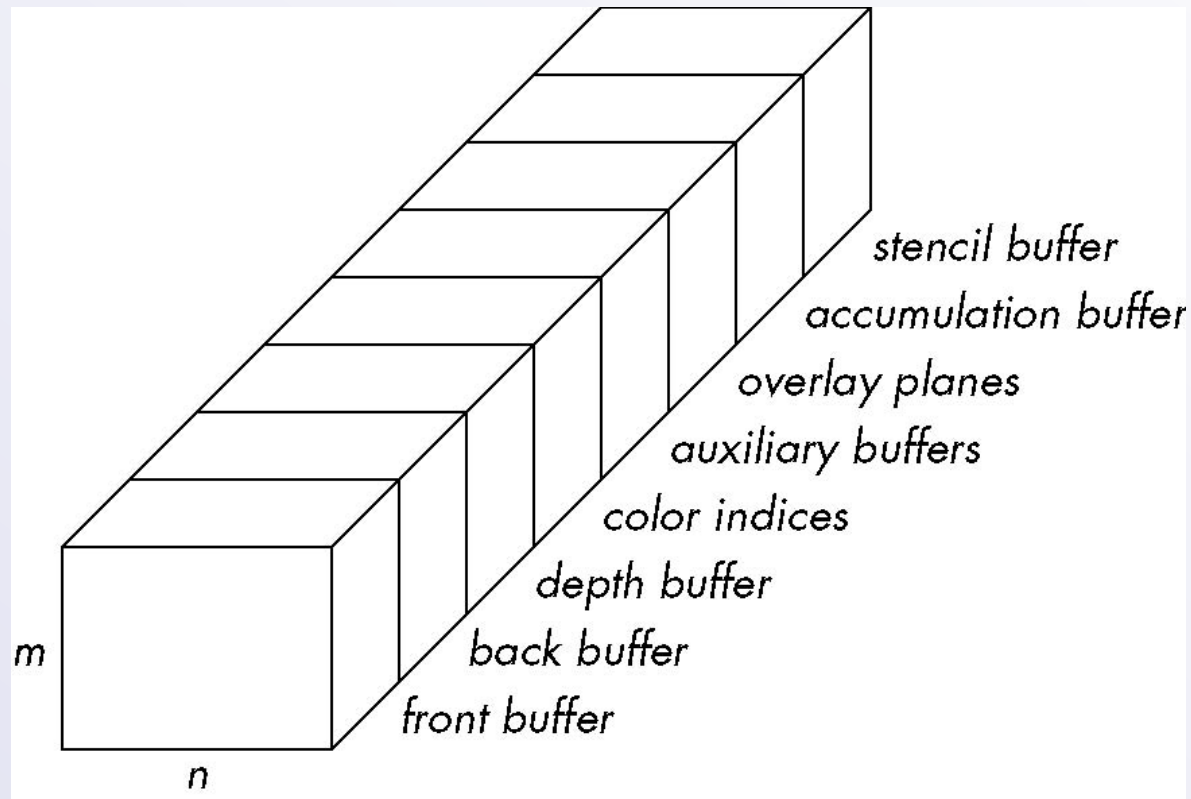


Buffer

- Um buffer é caracterizado por sua resolução espacial ($n \times m$) e sua profundidade (ou precisão) k , o número de bits/pixel

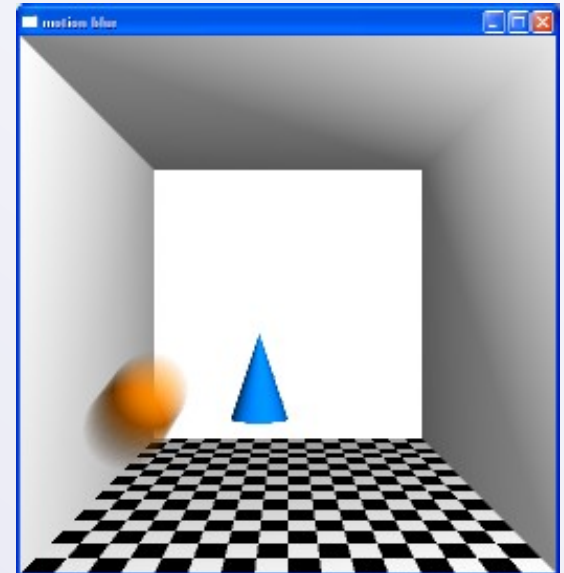


O frame buffer em OpenGL



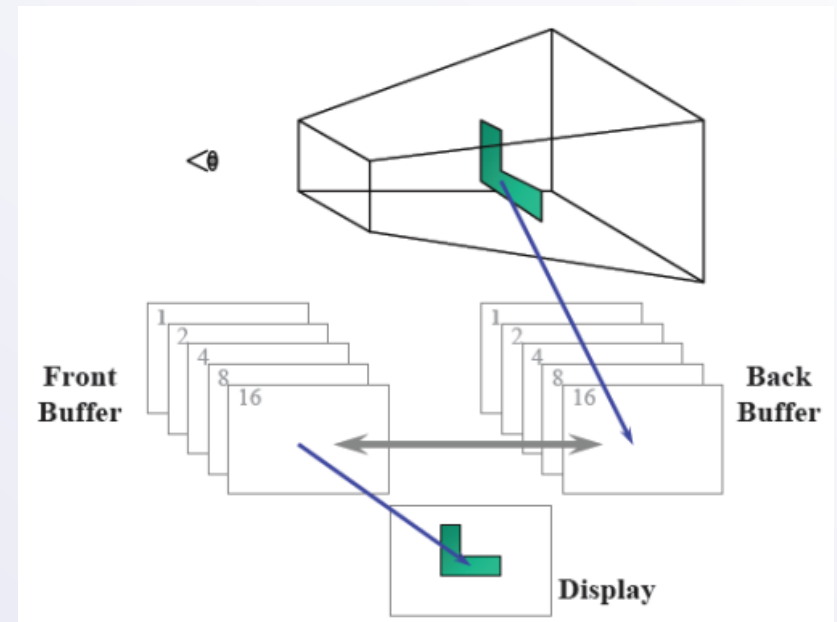
Buffers em OpenGL

- Os buffers coloridos podem ser visualizados
 - Front
 - Back
 - Auxiliary
 - Overlay
- Depth (buffer de profundidade)
- Accumulation (acumulação)
 - High resolution buffer
- Stencil (estêncil)
 - Armazena máscaras



Double buffering

- Técnica para remoção de artefatos visíveis durante o processo de desenho
- Devido à alta taxa de “refresh” dos displays, a renderização de objetos complexos podem não completar em um único ciclo
- Utiliza-se dois buffers, “front” e “back” (visível e escondido).
- A renderização é feita no buffer escondido
- Quando finalizado, troca-se os buffers, de modo a visualizar o que foi desenhado offscreen.



Double buffering em OpenGL

- GLUT possui comandos para habilitar o uso de buffers duplos.
- Ao invés de usar GLUT_SINGLE, utilize o flag GLUT_DOUBLE na chamada
`glutInitDisplayMode`
`(GLUT_RGB | GLUT_DOUBLE)`
- Na função display, troca-se os buffers utilizando-se a função `glutSwapBuffers()`
 - Deve ser a última função chamada na função de callback de display.

Animação em GLUT

- GLUT é orientado por eventos
- Para mostrar o próximo quadro (frame), precisamos de uma função que seja chamada quando o GLUT estiver em “idle” (sem eventos para processar).
- Duas funções callback para este fim:
 - `glutIdleFunc()`: Toda vez que ela é chamada podemos determinar se está na hora de desenhar um novo quadro.
 - `glutTimerFunc()`: Dado um delay (milisegundos), esta função é chamada depois de transcorrerem <delay> milisegundos. Precisa de um novo registro a cada vez.

Animação em GLUT

- Inicialização:
 - `glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GL_DEPTH);`
- Callback de display:
 - `glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);`
 - Constrói a cena
 - `glutSwapBuffers();`
- Callback de idle ou timer:
 - Decide se vai gerar um novo quadro:
 - SIM: `glutPostRedisplay();`
 - Se callback de timer, registra timer para próximo delay

Demo single_double.c

Demo particula.c

Toolkits e Widgets

- A maioria dos sistemas de janelas possuem uma biblioteca para a construção de interfaces através de controles chamados de “*widgets*”
- Widgets podem tomar a forma de
 - Menus
 - Slidebars
 - Dials
 - Input boxes (caixas de entrada)
- O problema que estas toolkits são normalmente dependentes da plataforma
- O GLUT provém alguns widgets que incluem a construção de menus

Outras funções do GLUT

- Criação de menus
- Janelas dinâmicas
 - Podemos criar e destruir durante a execução
- Subjanelas
- Janelas múltiplas
- Troca de callbacks durante a execução
- Timers
- Fontes
 - `glutBitmapCharacter`
 - `glutStrokeCharacter`

Menus

- GLUT suporta somente menus do tipo “pop-up”
 - Um menu pode ter submenus
- Três passos para o seu uso:
 - Definir as entradas para cada menu
 - Definir a ação a ser executada para cada item
 - Atrelar o menu a um botão do mouse

Criando um menu simples

```
menu_id = glutCreateMenu(menu);  
glutAddmenuEntry("limpa tela", 1);  
gluAddMenuEntry("Sair", 2);  
glutAttachMenu(GLUT_RIGHT_BUTTON);
```



estas entradas aparecerão
quando o botão direito
for pressionado

identificadores

Ações

- Callback de menu

```
void menu(int id)
{
    if(id == 1) glClear();
    if(id == 2) exit(0);
}
```

- Note que cada menu possui um identificador que é retornado ele é criado
- Podemos adicionar submenus com a chamada
`glutAddSubMenu(char *submenu_name, submenu id)`

← entrada no menu pai






Demo single_double-2.c

Desenhando objetos com GLU

- A GLU disponibiliza uma série de objetos 3D: esfera, cone, toro, 5 sólidos platônicos, e o bule de chá (“teapot”).
- Cada um deles é disponível em modelo wireframe e modelo sólido.
- Todos eles são desenhados por default na origem.
- Para usar a versão sólida de cada, troque **Wire** por **Solid** nas funções.

Sólidos Platônicos

- Todo poliedro convexo onde:
 - Todas as suas faces são polígonos congruentes.
 - Em cada vértice encontram-se o mesmo número de faces.

Nome	Imagem	Faces	Arestas	Vértices	Vértices por face	Encontros de faces em cada vértice	Configuração vértices
tetraedro		4	6	4	3	3	3.3.3
cubo (hexaedro)		6	12	8	4	3	4.4.4
octaedro		8	12	6	3	4	3.3.3.3
dodecaedro		12	30	20	5	3	5.5.5
icosaedro		20	30	12	3	5	3.3.3.3.3

Desenho de objetos em 3D

- **cubo:** `glutWireCube (GLdouble size);`
 - Cada lado tem comprimento `size`.
- **esfera:** `glutWireSphere (GLdouble radius, GLint nSlices, GLint nStacks);`
 - `nSlices` é o número de cortes;
 - `nStacks` é o número de discos;
 - De outra forma, `nSlices` representam número de linhas longitudinais e `nStacks` representam o número de linhas latitudinais.

Desenho de objetos em 3D

- **toro:** `glutWireTorus (GLdouble inRad, GLdouble outRad, GLint nSlices, GLint nStacks);`
- **teapot:** `glutWireTeapot (GLdouble size);`

Desenho de objetos em 3D

- **tetraedro:** `glutWireTetrahedron ();`
- **octaedro:** `glutWireOctahedron ();`
- **dodecaedro:** `glutWireDodecahedron ();`
- **icosaedron:** `glutWireIcosahedron ();`
- **cone:** `glutWireCone (GLdouble baseRad, GLdouble height, GLint nSlices, GLint nStacks);`

Desenho de objetos em 3D

- **cilindro:** `gluCylinder (GLUquadricObj *qobj, GLdouble baseRad, GLdouble topRad, GLdouble height, GLint nSlices, GLint nStacks);`
- A função **gluCylinder** desenha uma *família* de objetos, dependendo do valor de `topRad`.
 - Quando `topRad` é 1, ela desenha um cilindro.
 - Quando `topRad` é 0, ela desenha um cone.

Desenho de objetos em 3D

// cria um objeto quádrico

```
GLUquadricObj *qobj =  
    gluNewQuadric();
```

// muda estilo para wireframe

```
gluQuadricDrawStyle(qobj, GLU_LINE |  
    GLU_FILL);
```

// desenha cilindro

```
gluCylinder(qobj, baseRad, topRad,  
    nSlices, nStacks);
```

Demo teapot.c
