1 2  9 0
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D COIMBRA

dei25 departamento de engenharia informática 1995 – 2020

Computação Gráfica

André Perrotta (avperrotta@dei.uc.pt)

Hugo Amaro (hamaro@dei.uc.pt)

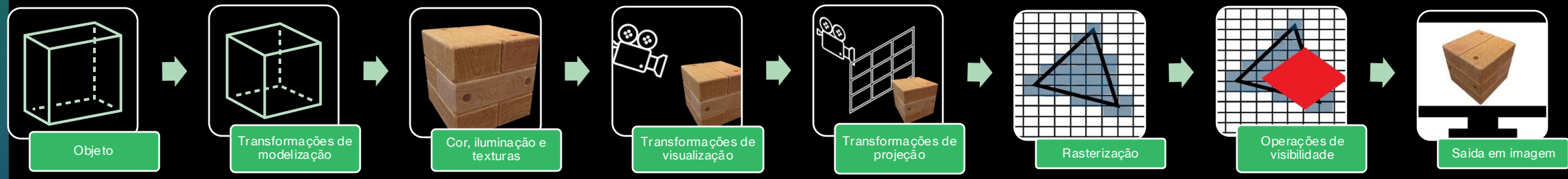
T_06:

projeção, visualização

Objetivos da aula

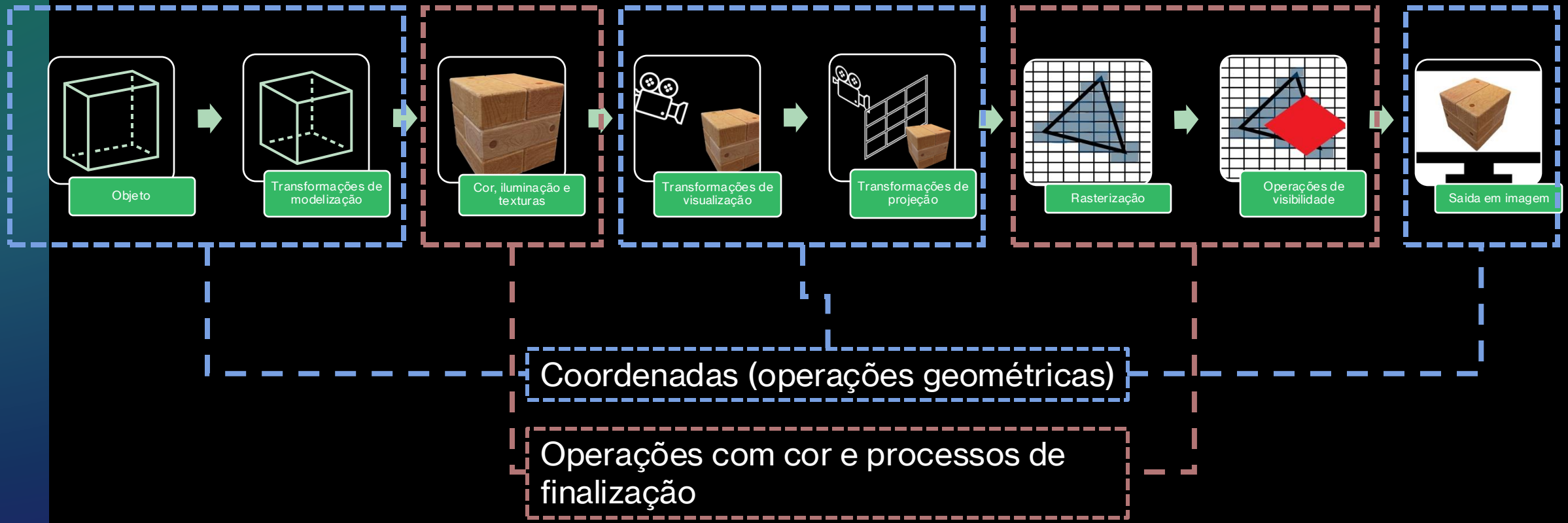
- Entender o conceito de projeção 3D -> 2D.
- Entender as operações matemáticas necessárias para projetar um ponto 3D em um plano 2D (tela).

Render pipeline



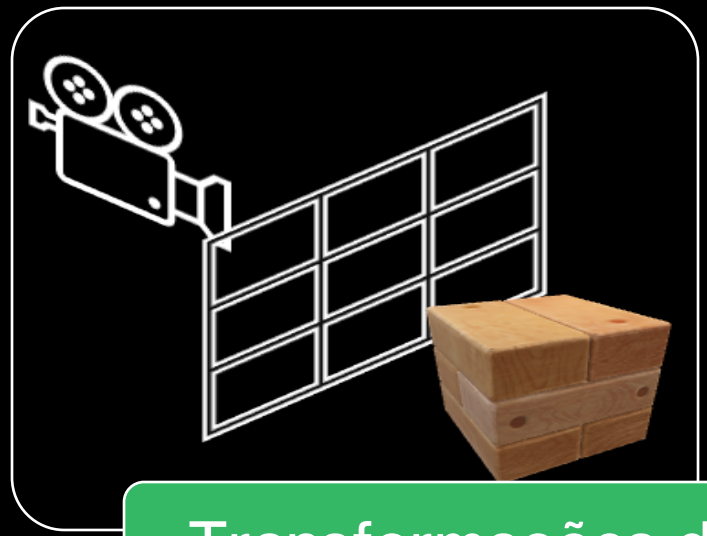
Pipeline de renderização POLIGONAL

Render pipeline



Câmera e projeção

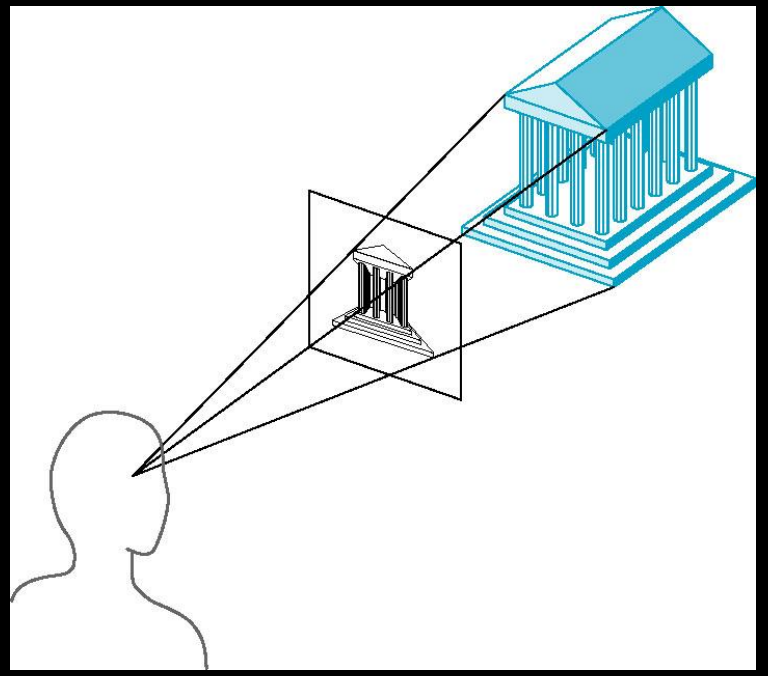
Transformações de visualização



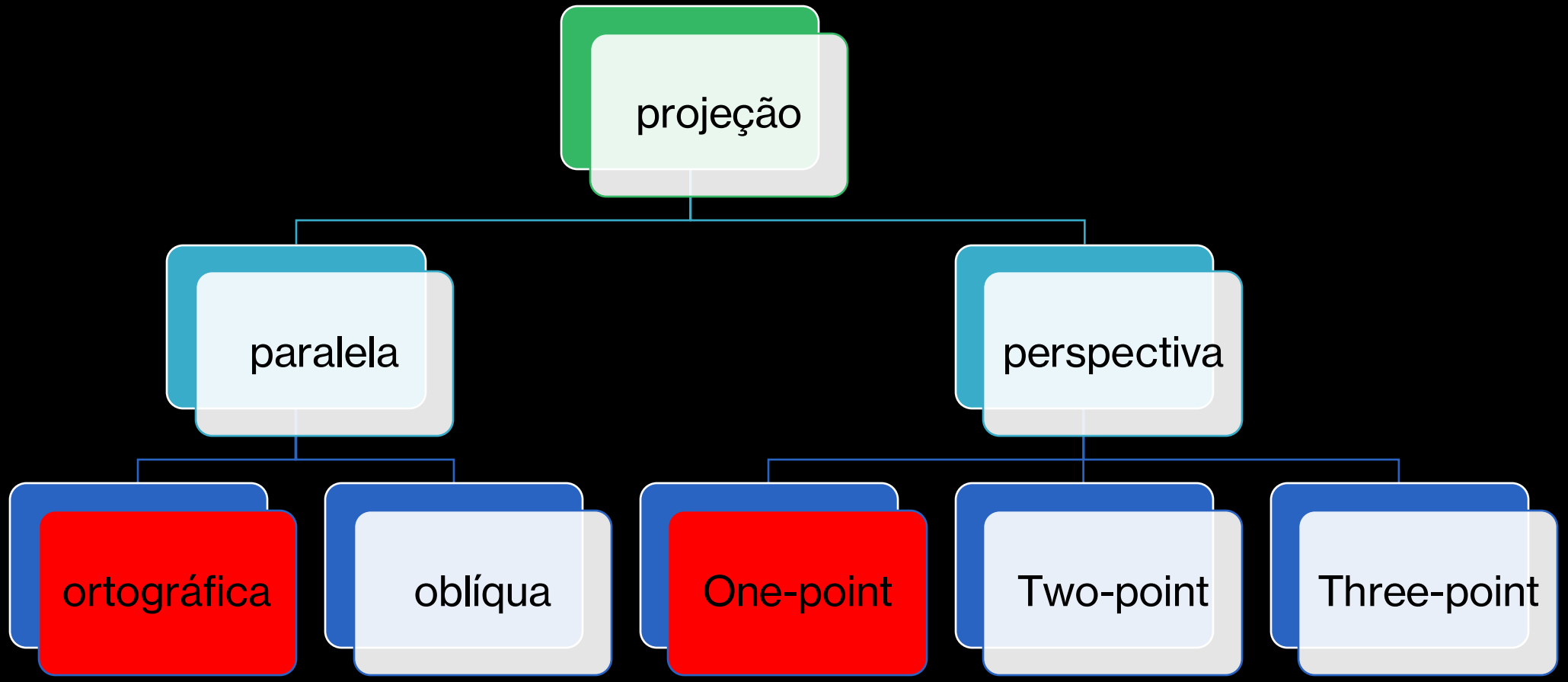
Transformações de projeção

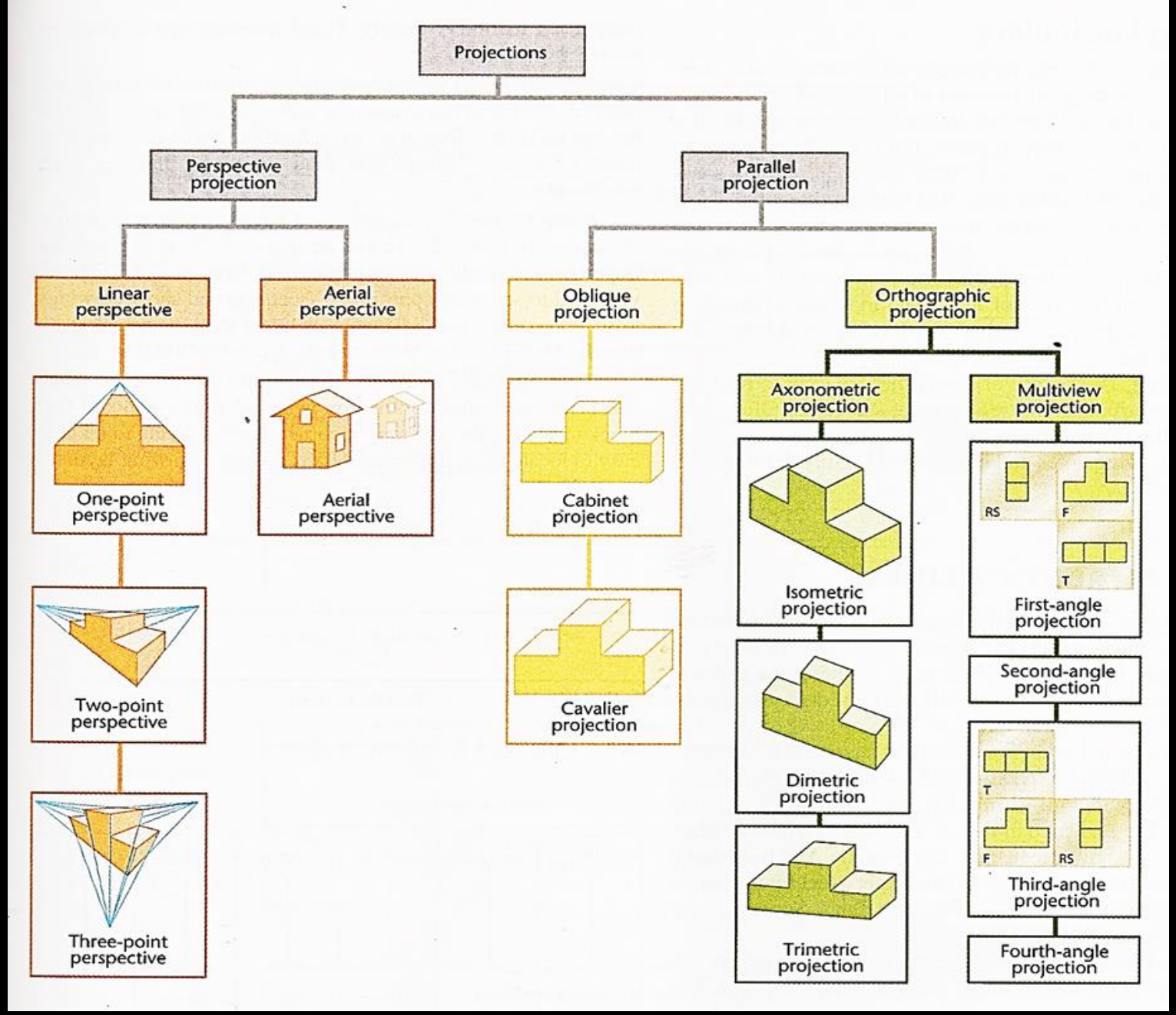
Projeção

Mapear coordenadas 3D num plano 2D.



Projeção: tipos de projeção





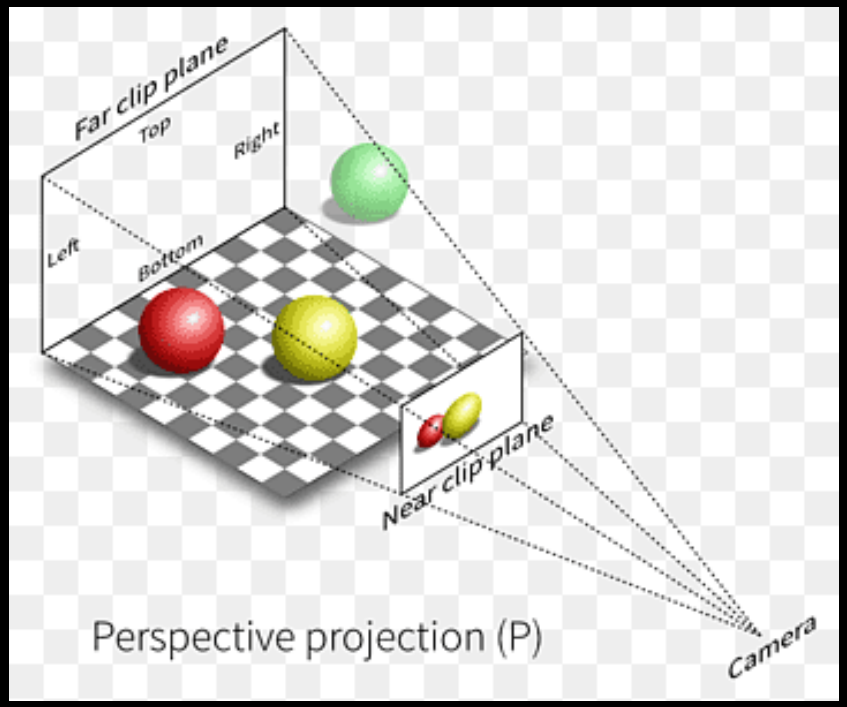
https://www.frankminnella.com/drawing-projection-types.html

Projeção em OpenGL

- Em OpenGl conseguimos trabalhar “diretamente”, com funções pré-definidas as projeções:
  - Perspetiva -> 1 ponto
  - Paralela -> ortográfica

Projeção: perspectiva

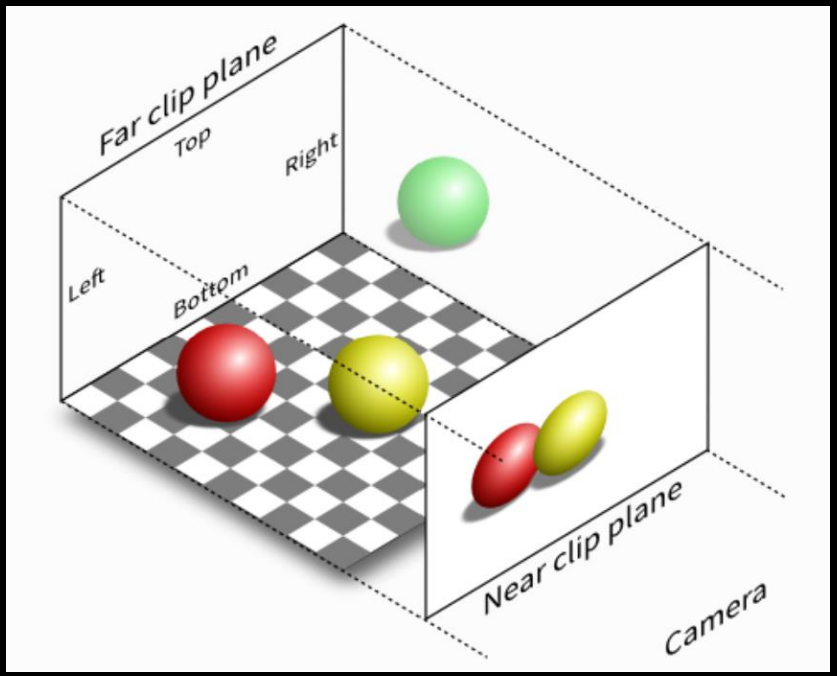
- Centro de projeção converge numa distância finita ao plano de projeção



https://glumpy.readthedocs.io/en/latest/tutorial/cube-ugly.html

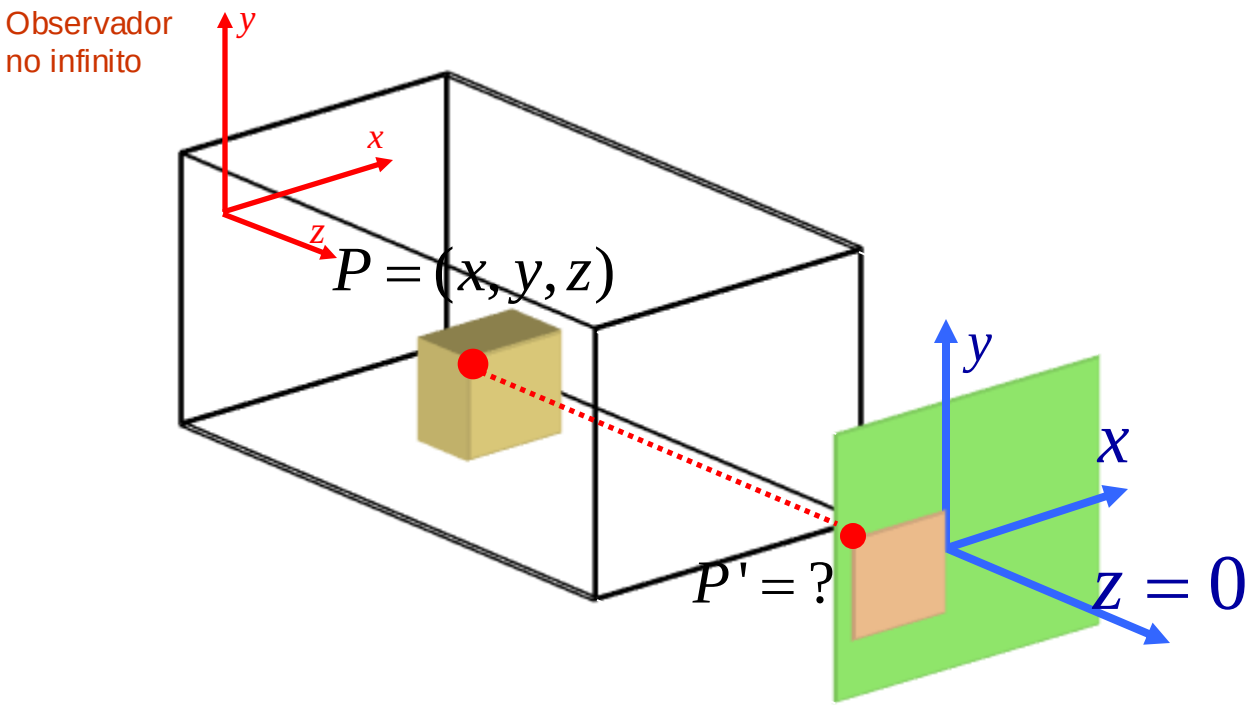
Projeção: paralela ortográfica

- Centro de projeção não converge (ou converge no infinito).
- Há perda da informação de profundidade.
- Linhas de projeção perpendiculares ao plano de projeção.



https://glumpy.readthedocs.io/en/latest/tutorial/cube-ugly.html

Matriz de projeção: ortográfica



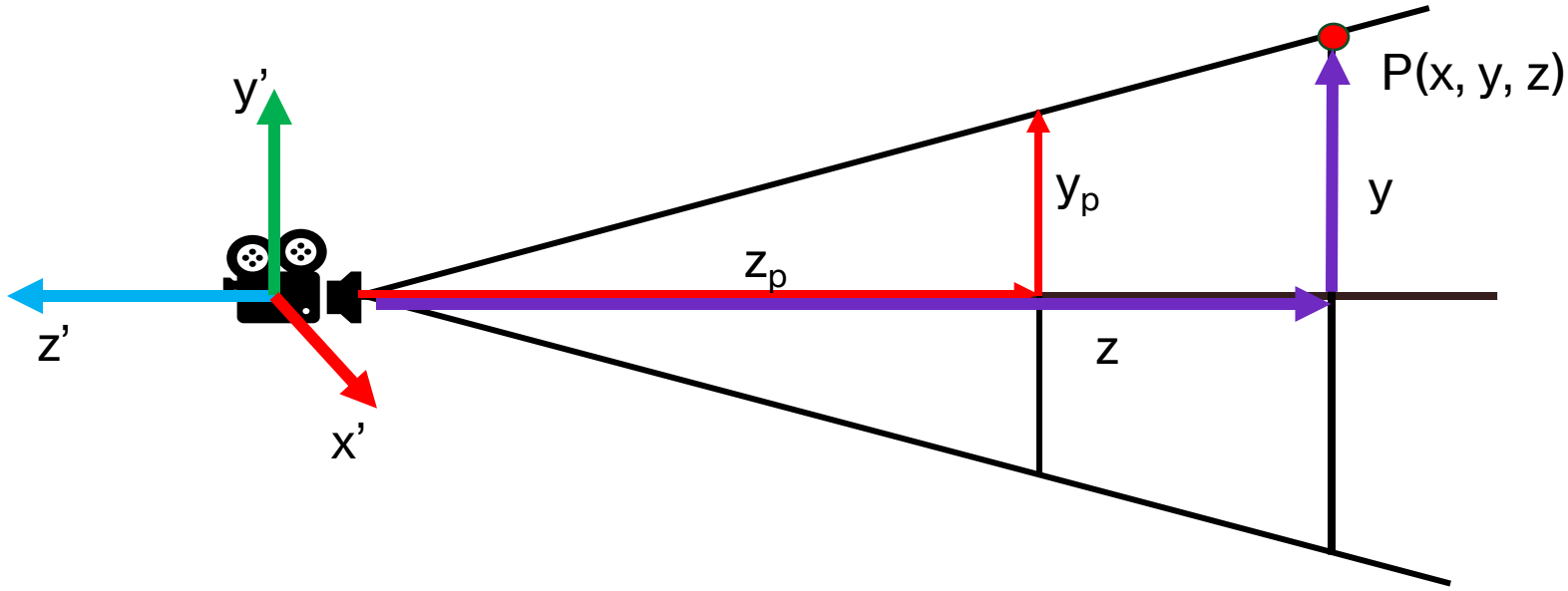
$$x_p = x$$

$$y_p = y$$

$$z_p = 0$$

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Matriz de projeção: perspectiva



Se eu conhecer (x, y, z), observando a semelhança de triângulos posso aferir (x_p, y_p, z_p) :

$\frac{y}{z} = \frac{y_p}{z_p}$, $\frac{x}{z} = \frac{x_p}{z_p}$ z_p = -d (distância até a camera)

Matriz de projeção: perspectiva

- Quero poder construir uma matriz que aplique a relação nas 3 componentes:
  - $x_p = -d*x/z$
  - $y_p = -d*y/z$
  - $z_p = -d$

  - Podemos tentar:

$$\begin{pmatrix} -d/z & 0 & 0 & 0 \\ 0 & -d/z & 0 & 0 \\ 0 & 0 & -d/z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Matriz de projeção: perspectiva

- A solução possível, que seria uma matriz do tipo escala, apresenta problemas:
  - Não é genérica, ou seja, o próprio valor de z de cada vértice entra na matriz.
    - Não representa uma matriz global de vizualização, que pode ser definida apenas 1 vez para cada vista
  - Elimina a possibilidade de guardar o valor de z, o que impossibilita etapas mais avançadas, onde preciso do valor de z para saber a ordem dos desenho

Matriz de projeção: perspectiva

- Se lembrarmos que a coordenada homogênea “w”, pode assumir qualquer valor, podemos pensar na seguinte relação:
  - w = -z/d, então:
  - x_p = x/w
  - y_p = y/w
  - z_p = -d
  - Daí sai que:

$$\begin{cases} x' = x \\ y' = y \\ z' = z \\ w = \dfrac{z}{-d} \end{cases} \longleftrightarrow \begin{bmatrix} x_p / w \\ y_p / w \\ z_p / w \\ 1 / w \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} \longleftrightarrow \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \dfrac{1}{-d} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Matriz de projeção: perspectiva

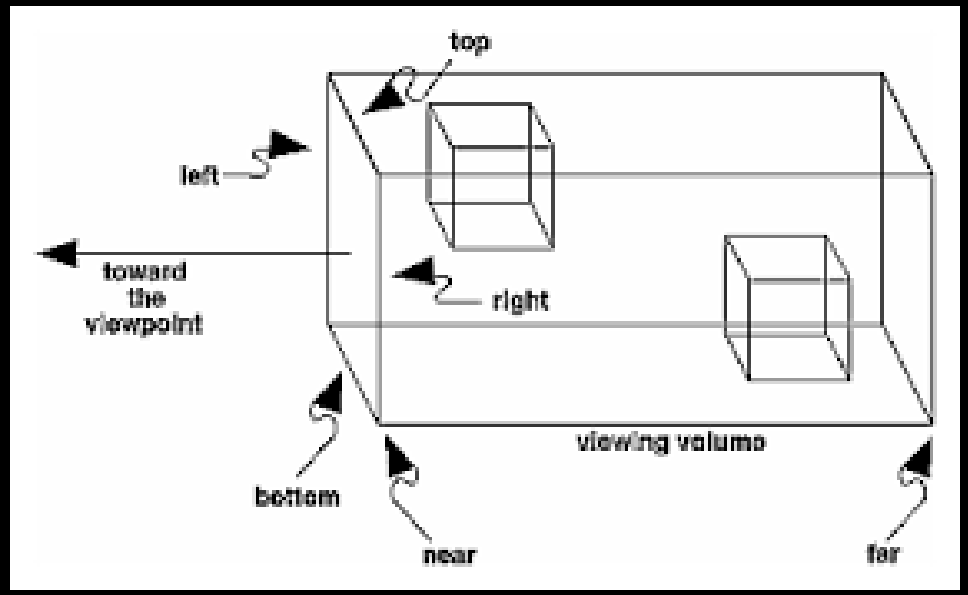
$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{-d} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Volume de projeção

- EM OpenGl, para além de definir a matriz de projeção, devo também definir um volume de projeção que é utilizado para selecionar os vértices que devem ser desenhados ou não.
- Dentro do volume -> desenha
- Fora do volume -> não desenha

Volume de projeção: ortográfico

- O volume de projeção ortográfico é um paralelepípedo definido por:
- Left, right, bottom, top, near, far clipping planes



Volume de projeção: ortográfico

- Por exemplo, se quero ter meu sistema de coordenadas do OF, em projeção ortográfica faço:
  - glMatrixMode(GL_PROJECTION);
  - glLoadIdentity();
  - glOrtho(0, gw(), gh(), 0, nearZ, farZ);
  - glMatrixMode(GL_MODELVIEW);
  - glLoadIdentity();

Volume de projeção: ortográfico

- Em OpenGl, a função glOrtho sempre força uma matriz de transformação canônica, colocando a origem centrada no plano de projeção.
- Apesar de permitir usar valores quaisquer para glOrtho(l, r, b, t, n, f), idealmente, para atingirmos uma utilização intuitiva, devemos escolher glOrtho(-w/2, w/2, -h/2, h/2, near, far). Podendo inverter +- para inverter os eixos de acordo com nossa preferência.

$$\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & t_x \\ 0 & \frac{2}{top-bottom} & 0 & t_y \\ 0 & 0 & \frac{-2}{farVal-nearVal} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

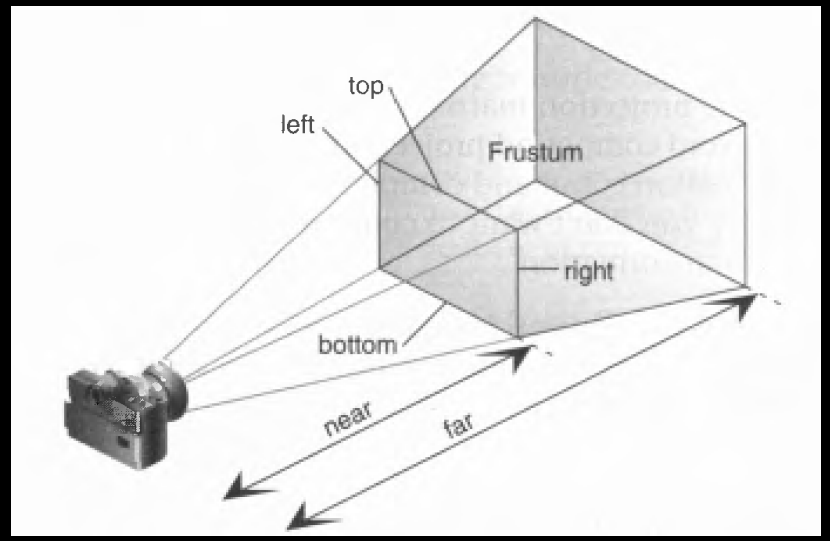
$$t_x = -\frac{right+left}{right-left}$$

$$t_y = -\frac{top+bottom}{top-bottom}$$

$$t_z = -\frac{farVal+nearVal}{farVal-nearVal}$$

Volume de projeção: perspectiva

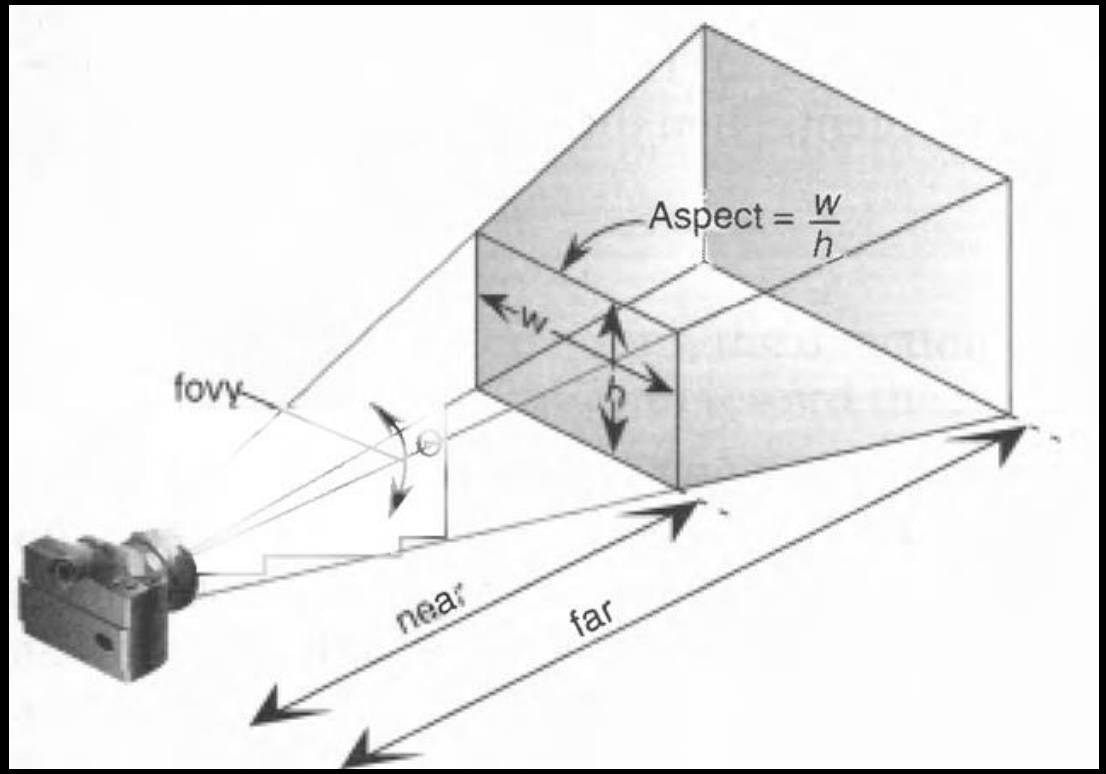
- Em projeção perspectiva, o volume de projeção é um prisma (frustum), que pode ser definido diretamente pela função OpenGl: glFrustum(left, right, bottom, top, near, far).



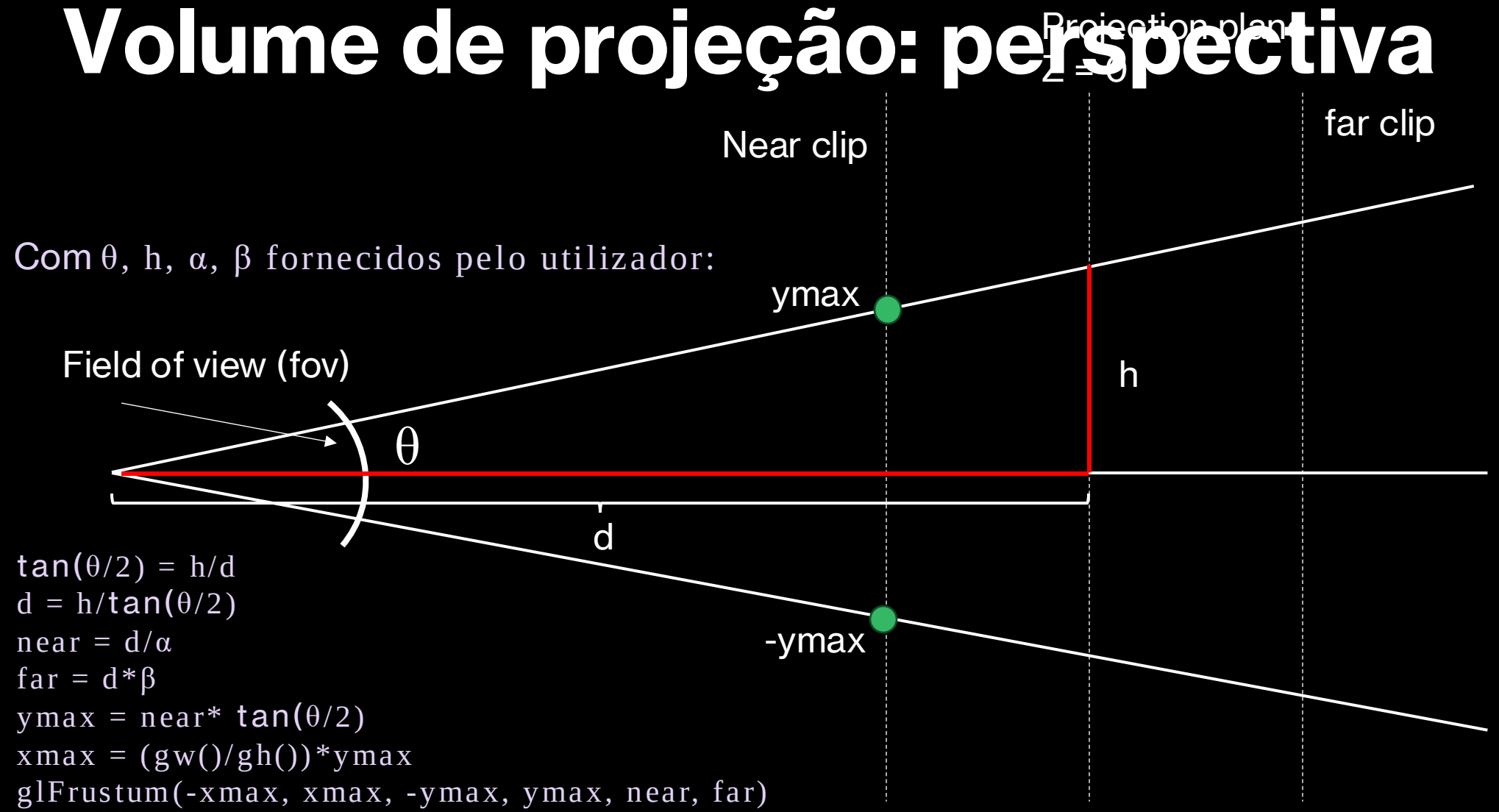

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(left, right, bottom, top, near, far);
```

Volume de projeção: perspectiva

- A função glFrustum() não é muito intuitiva, mas há uma maneira mais elegante de determinar o volume de projeção (prisma) usando como base o tamanho da tela pretendida e o ângulo de abertura.
- Além de ser mais intuitiva, esta forma faz uma ligação direta com as lentes das câmeras no mundo real.



Volume de projeção: perspectiva



tan(θ/2) = h/d
d = h/tan(θ/2)
near = d/α
far = d*β
ymax = near* tan(θ/2)
xmax = (gw()/gh())*ymax
glFrustum(-xmax, xmax, -ymax, ymax, near, far)

Projeção em OpenGL: detalhes tecnicos

- Em OpenGl, os volumes de projeção são sempre centrados no observador localizado em (0, 0, 0) e a matriz GL_PROJECTION, para além de determinar a geometria do volume de projeção ela também é responsável por normalizar os volumes para cubos ”unitários” com a seguinte configuração
  - left = -1 ,right = 1, bottom = -1, top = 1, near = -1, far = 1

Projeção em OpenGL: detalhes tecnicos

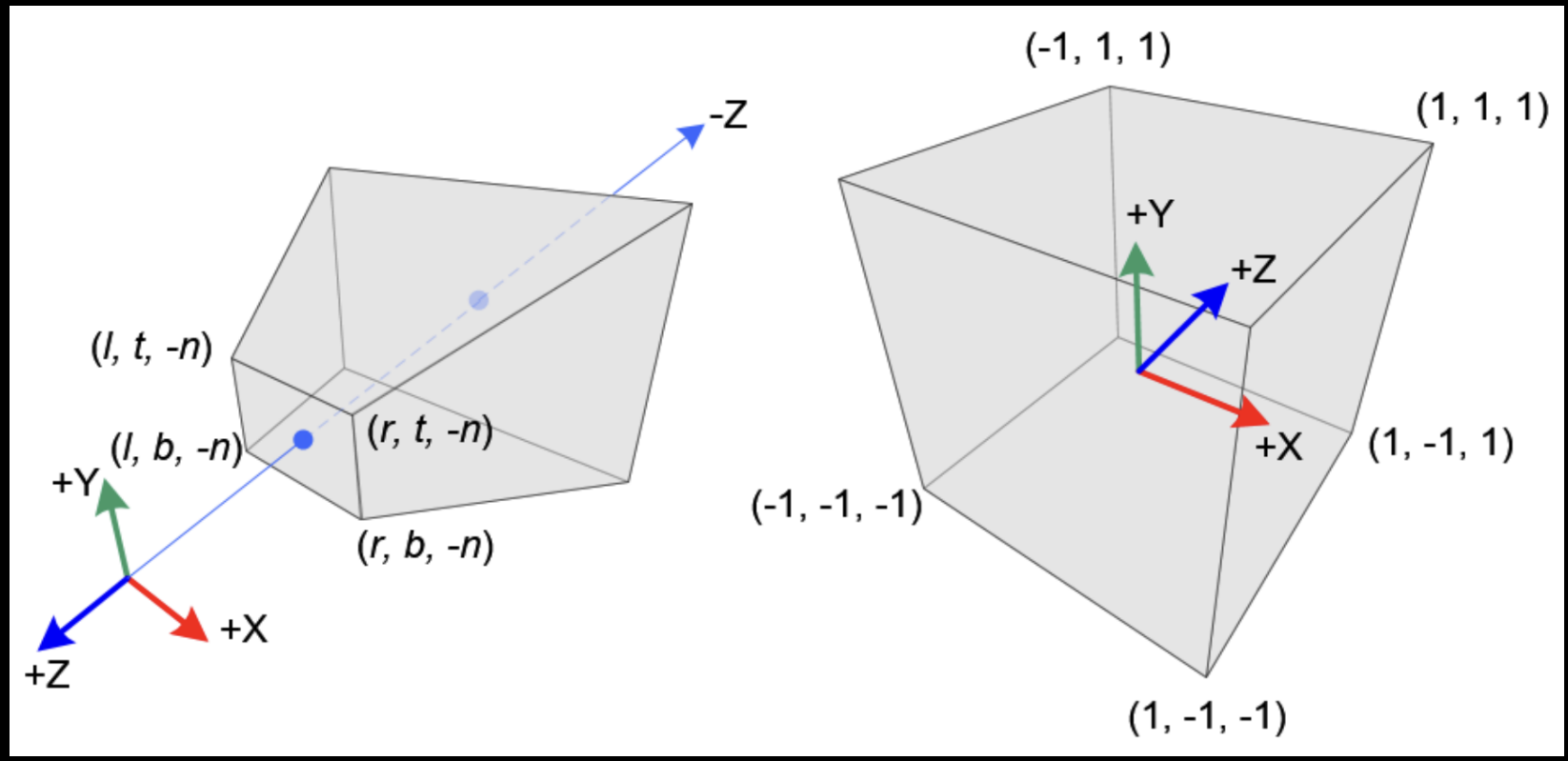


Image from: https://www.songho.ca/opengl/gl_projectionmatrix.html#perspective

Projeção em OpenGL: detalhes tecnicos

- Para poder resolver volumes de projeções quaisquer e normalizá-los, o OpenGl utiliza matrizes de projeção mais complexas:

perspetiva

$$\begin{bmatrix} \frac{2\text{nearVal}}{\text{right}-\text{left}} & 0 & A & 0 \\ 0 & \frac{2\text{nearVal}}{\text{top}-\text{bottom}} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$A = \frac{\text{right}+\text{left}}{\text{right}-\text{left}}$$

$$B = \frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}}$$

$$C = -\frac{\text{farVal}+\text{nearVal}}{\text{farVal}-\text{nearVal}}$$

$$D = -\frac{2\text{farVal}\,\text{nearVal}}{\text{farVal}-\text{nearVal}}$$

ortográfica

$$\begin{pmatrix} \frac{2}{\text{right}-\text{left}} & 0 & 0 & t_x \\ 0 & \frac{2}{\text{top}-\text{bottom}} & 0 & t_y \\ 0 & 0 & \frac{-2}{\text{farVal}-\text{nearVal}} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where

$$t_x = -\frac{\text{right}+\text{left}}{\text{right}-\text{left}}$$

$$t_y = -\frac{\text{top}+\text{bottom}}{\text{top}-\text{bottom}}$$

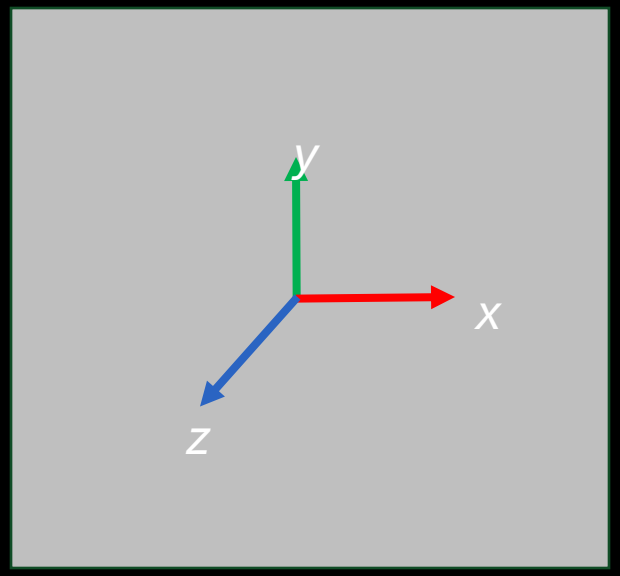
$$t_z = -\frac{\text{farVal}+\text{nearVal}}{\text{farVal}-\text{nearVal}}$$

Projeção perspectiva: OF

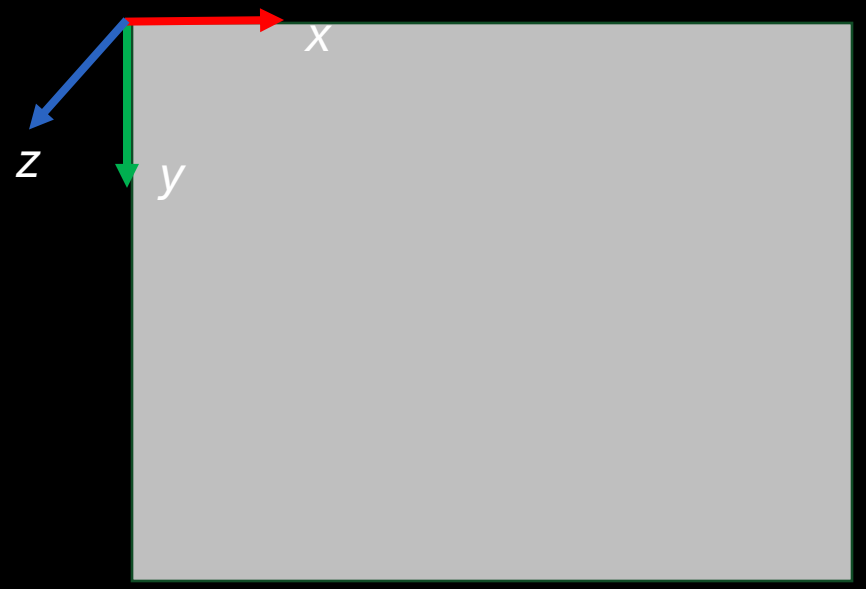
- Direção dos eixos:
  - As funções glFrustum(left, right, bottom, top, near, far) e glOrtho(left, right, bottom, top, near, far) não assumem que valores são utilizados para determina-las.
  - A escolha dos valores afeta a direção final dos eixos. Se por exemplo, left for maior do que right, o eixo horizontal será positivo para a esquerda e não para a direita.
  - Essa escolha é crítica, pois vai determinar a orientação final de visualização dos eixos de coordenadas.
  - Por defeito, em OpenGl, os valores para glFrustum(r, l, b, t, n, f) são (-1, 1, -1, 1, 1, 100).
  - Em OF, por defeito, há uma inversão dos valores bottom e top, que faz com que o eixo y esteja orientado para baixo e não para cima.

Projeção perspectiva: OF

OpenGL default



OF default

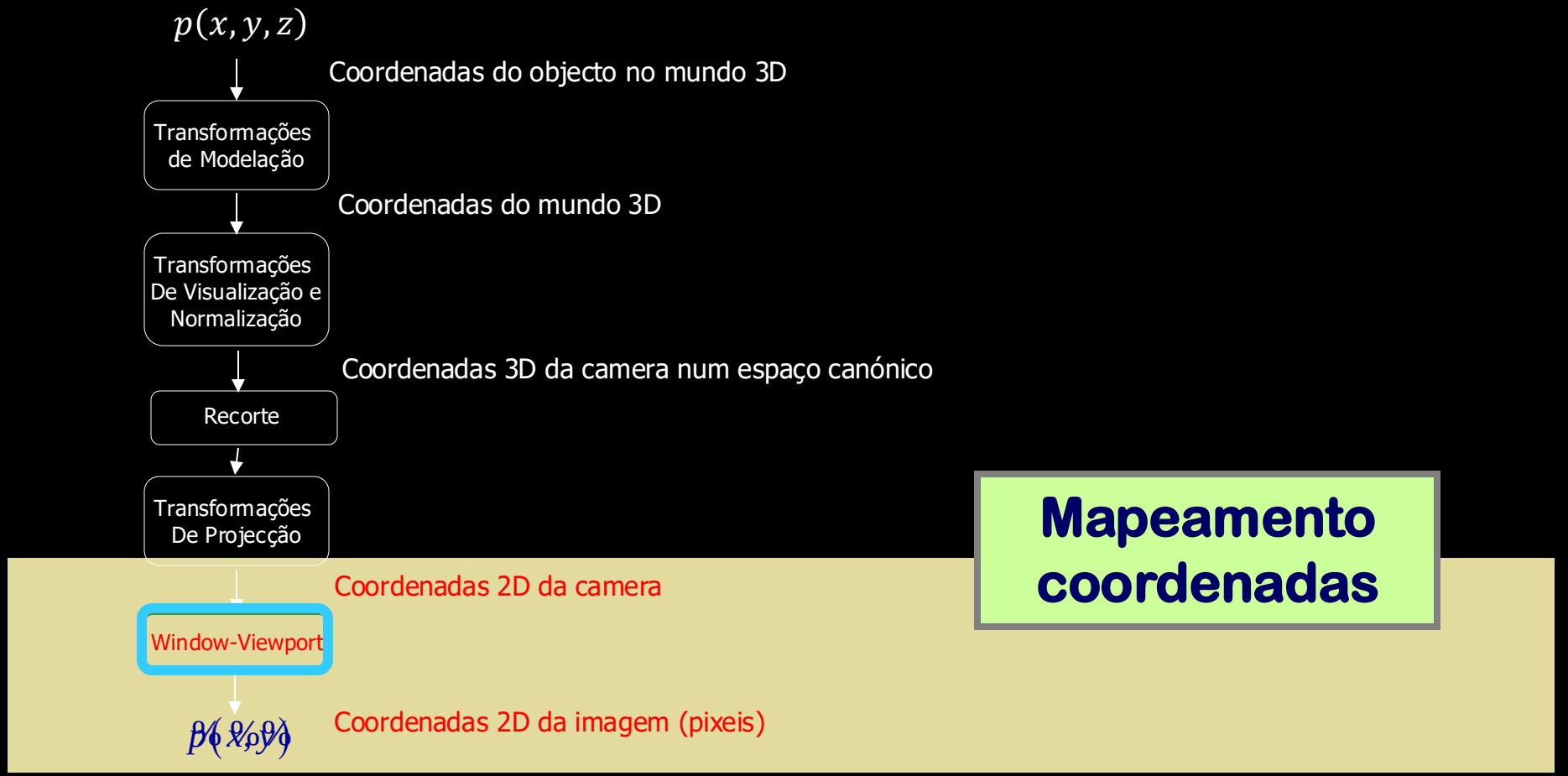


Openframeworks vs OpenGL

- Como a função glFrustum sempre força o observador para o centro da projeção, o eixo definido pelo Openframeworks para uma projeção perspectiva não é possível de ser obtido diretamente pela função glFrustum(...)
- Para se obter um eixo descentralizado, mas mantendo a intuição e algoritmos usuais para as transformações de modelação e cãmera (por ex. A função lookat), é sempre necessário modificar a matriz de projeção resultante.
- Em OF, há uma modificação da GL_PROJECTION alterando os valores de A e B para 0 após a definição do glFrustum(...)
- Isto é resolvido em “back office” antes do método ofApp::draw() ser chamado.
- Ao carregar a matriz identidade na GL_PROJECTION explicitamente no começo do draw, ou através da função perspective(), apagamos esta definição do OF

Dúvidas?

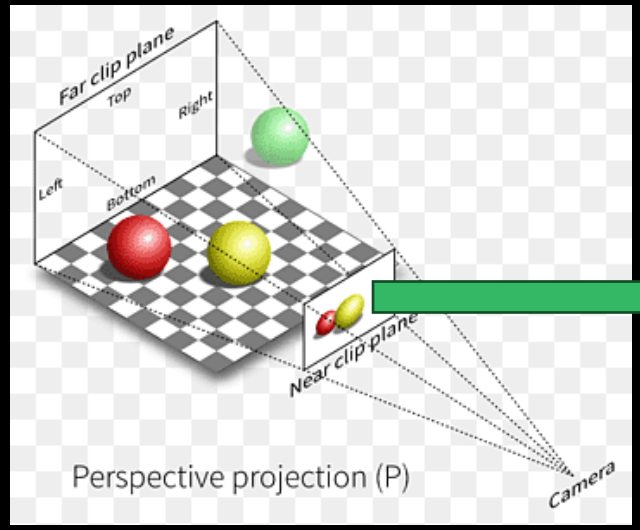
Janela de visualização: viewport



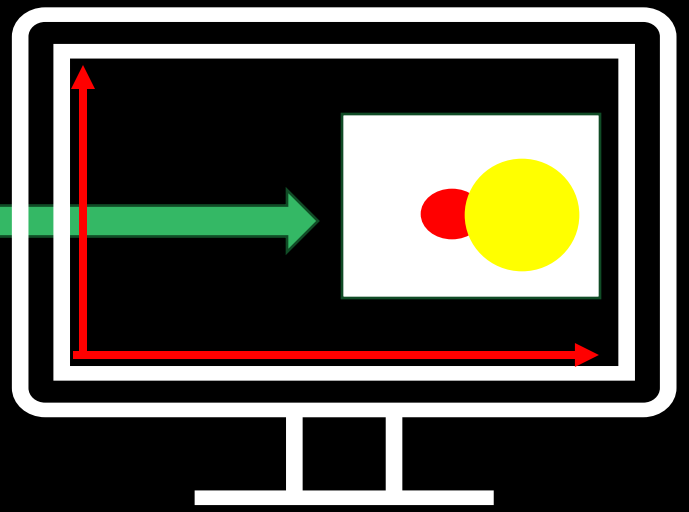
Viewport

- No pipeline de transformações geométricas, a última etapa é a escolha da posição na janela do programa onde quero colocar o meu desenho.

Coordenadas do mundo * Lookat * projeção



Coordenadas da janela do aplicativo



https://glumpy.readthedocs.io/en/latest/tutorial/cube-ugly.html

Viewport

- Window-to-Viewport

- Window : área rectangular visivel no sistema de coordenadas mundo
  - Representado no sistema de coordenadas cartesiano.
  - Definido por **Xmin, Ymin, Xmax, Ymax**
  - **Definido por glFrustum ou glOrtho**

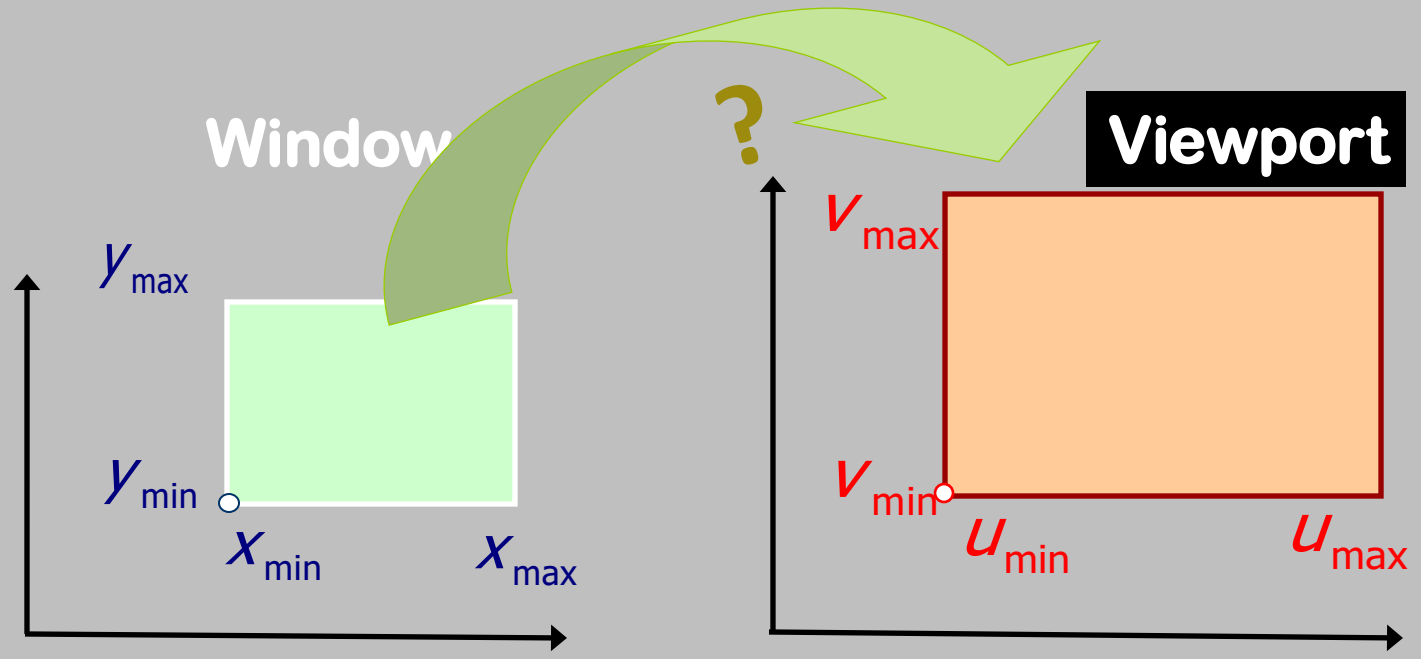
- Viewport: área ecrã onde é desenhado o que se deseja mostrar
  - Geralmente de tamanho fixo (ou não controlado pela aplicação)
  - Definido por: **Umin, Vmin, Umax, Vmax**

Viewport: matriz de transformação

- Como nas outras etapas, o posicionamento e escala resultante da viewport é realizado com uma matriz:

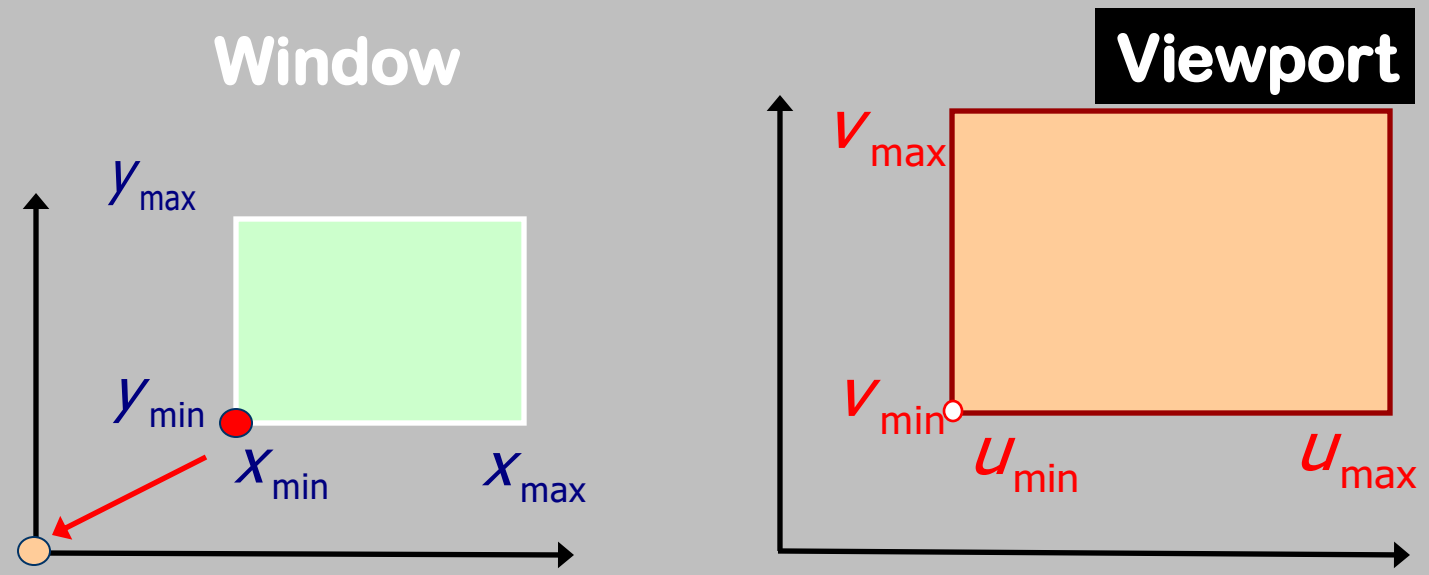
$$M_T = \begin{bmatrix} Sx & 0 & Tx \\ 0 & Sy & Ty \\ 0 & 0 & 1 \end{bmatrix}$$

Viewport: matriz de transformação



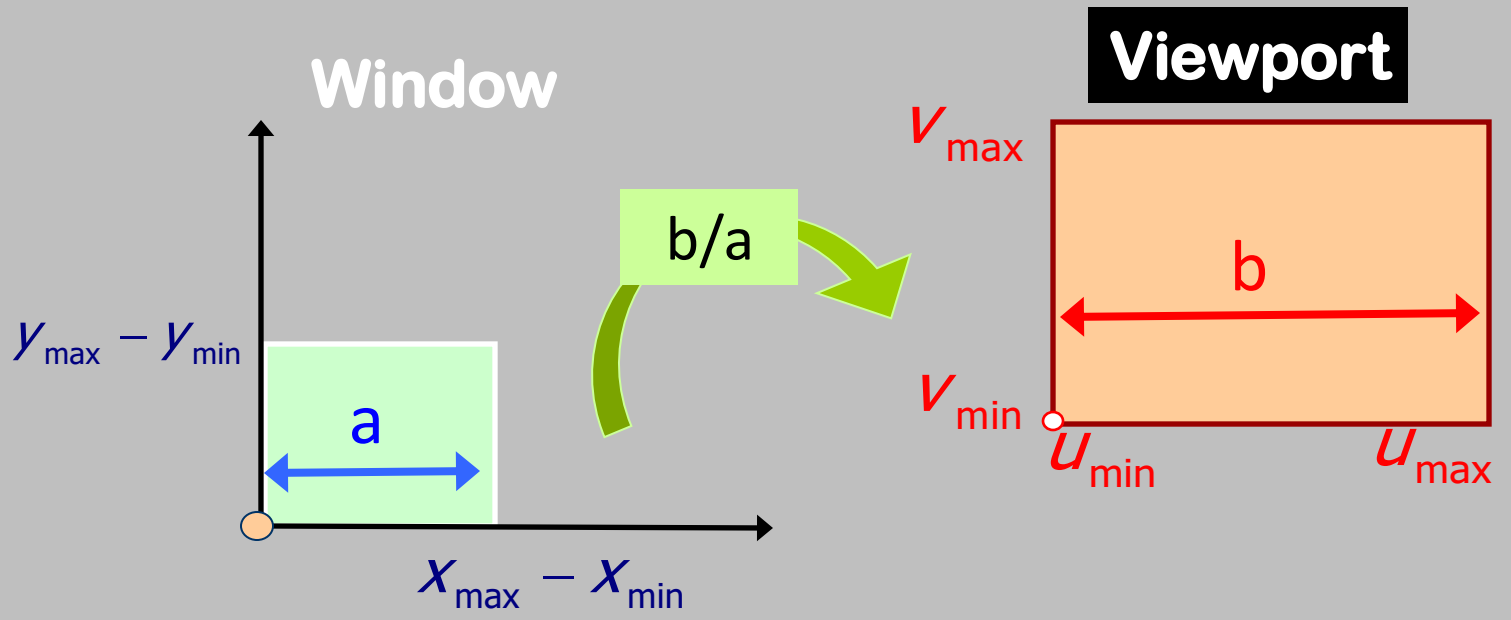
Viewport: matriz de transformação

- 1. **Translação** da janela para a origem



$$T(-x_{min}, -y_{min})$$

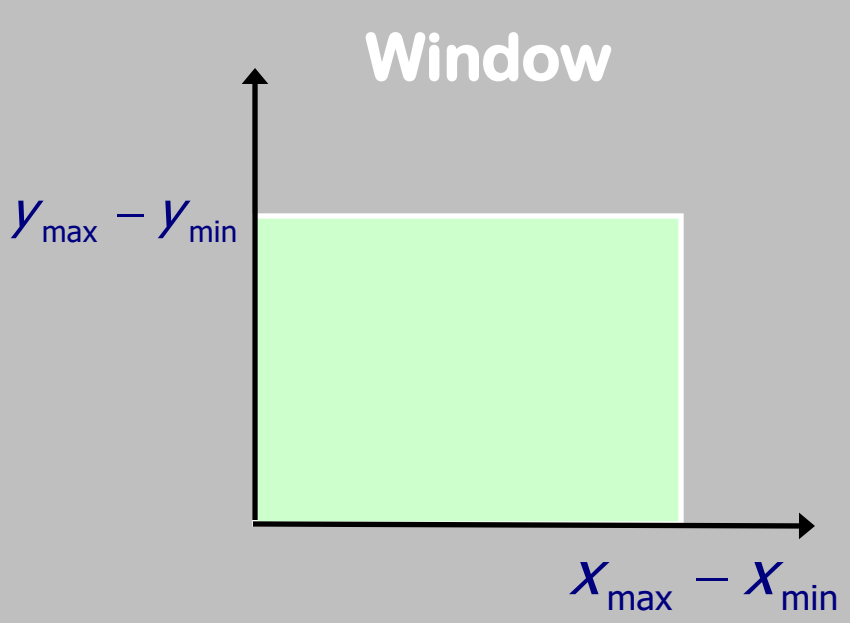
Viewport: matriz de transformação

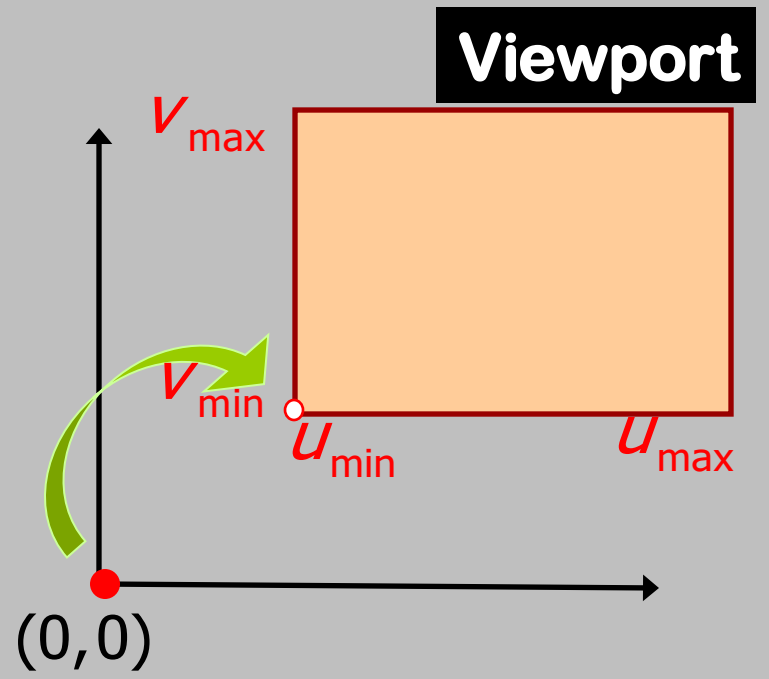


2. Mudança de **escala** para que coincida com as dimensões do viewport

$$S\left(\frac{u_{max}-u_{min}}{x_{max}-x_{min}}, \frac{v_{max}-v_{min}}{y_{max}-y_{min}}\right)$$

Viewport: matriz de transformação

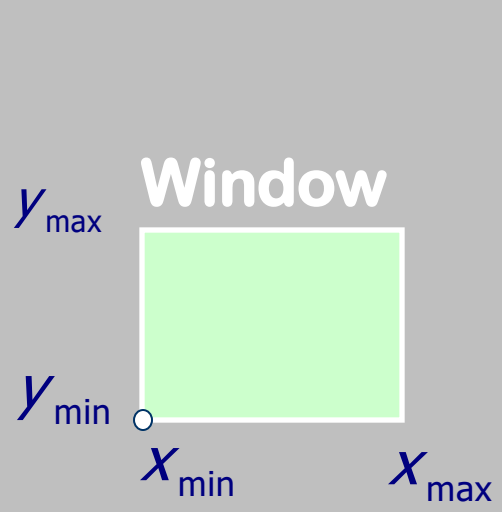


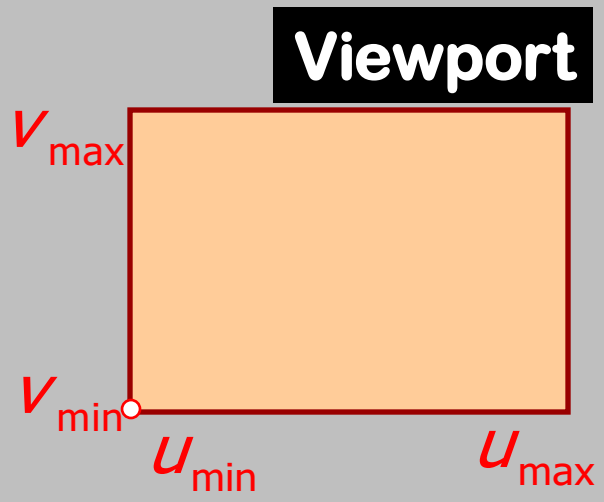


3. <u>Translação</u> para a região de viewport

$$T(u_{min}, v_{min})$$

Viewport: matriz de transformação





$$M_T = T(u_{min}, v_{min}).S(\frac{u_{max} - u_{min}}{x_{max} - x_{min}}, \frac{v_{max} - v_{min}}{y_{max} - y_{min}}).T(-x_{min}, -y_{min})$$

Viewport: matriz de transformação

- **Matricialmente**

$$M_T = T(u_{\min}, v_{\min}).S(\frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}}, \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}).T(-x_{\min}, -y_{\min})$$

$$= \begin{pmatrix} 1 & 0 & u_{\min} \\ 0 & 1 & v_{\min} \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{pmatrix}$$

Viewport: matriz de transformação

$$M_T = \begin{bmatrix} Sx & 0 & Tx \\ 0 & Sy & Ty \\ 0 & 0 & 1 \end{bmatrix}$$

Matriz (3,3) – 2D

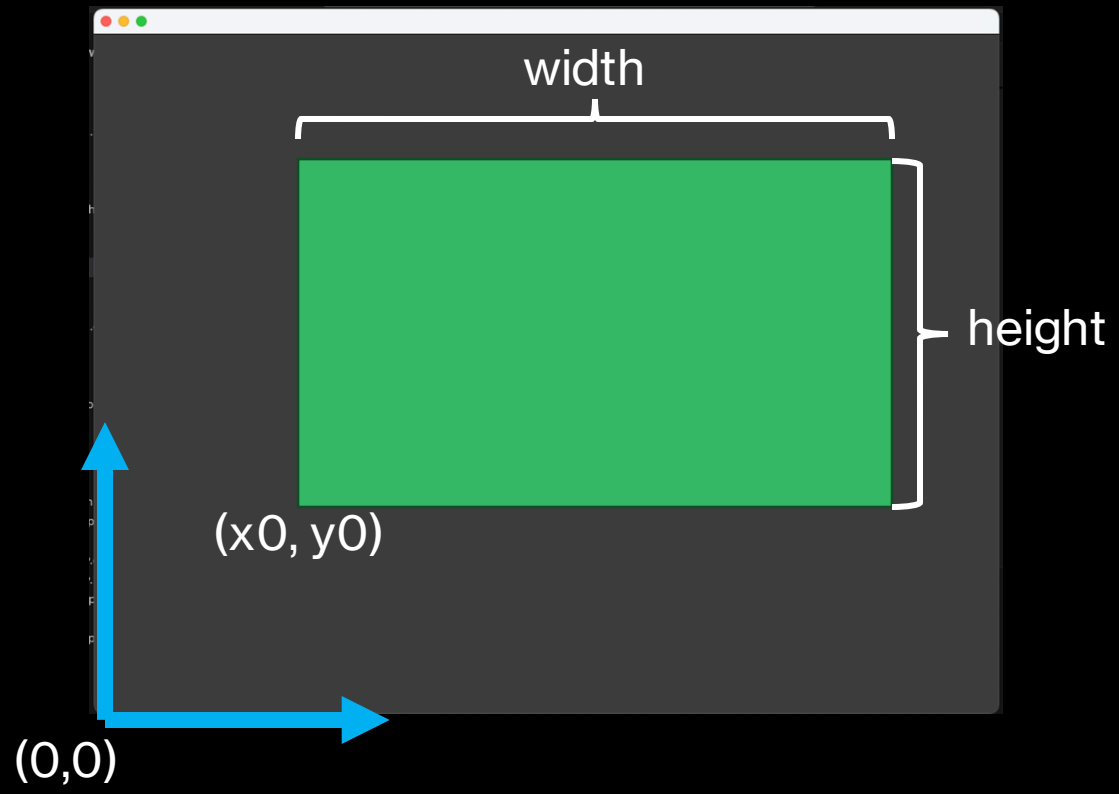
- **Matricialmente**

$$M_T = \begin{pmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & -x_{min} \cdot \frac{u_{max} - u_{min}}{x_{max} - x_{min}} + u_{min} \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & -y_{min} \frac{v_{max} - v_{min}}{y_{max} - y_{min}} + v_{min} \\ 0 & 0 & 1 \end{pmatrix}$$

Viewport em OpenGl + OF

- No Openframeworks/OpenGl, a definição do viewport é bastante simples
- glViewport(x0, y0, width, height)

Por exemplo um “viewport” que ocupe a janela toda:
glViewport(0, 0, gw(), gh());



OpenGl coordinate pipeline done!

- Agora sabemos calcular a posição final de um vértice na tela!!!

- V(u, v) = Viewport*Projeção*Modelview*V(x, y, z, 1)

Créditos

- Muitos slides e imagens foram “emprestados” do ótimo material do prof. Jorge Henriques ☺