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_05:

Camera, projeção, visualização

Objetivos da aula

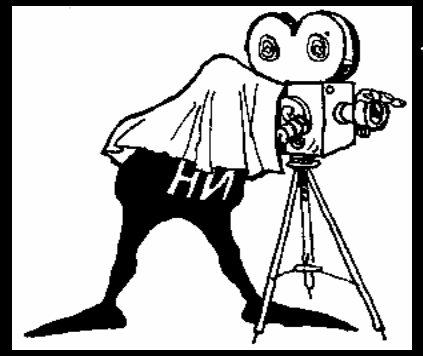
- Entender o conceito de câmera no contexto de CG em 3D.
- Entender as operações matemáticas necessárias para transformar coordenadas mundo em coordenadas da câmera.
- Entender o conceito de projeção 3D -> 2D.
- Entender as operações matemáticas necessárias para projetar um ponto 3D em um plano 2D (tela).

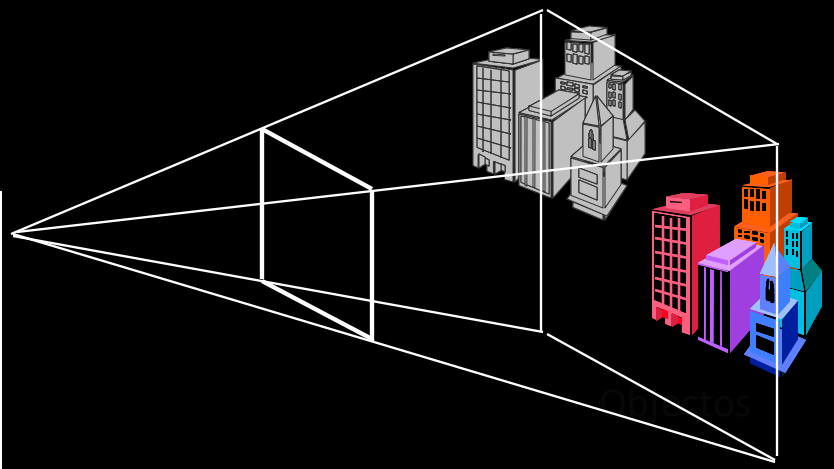
Conceito de câmera em CG

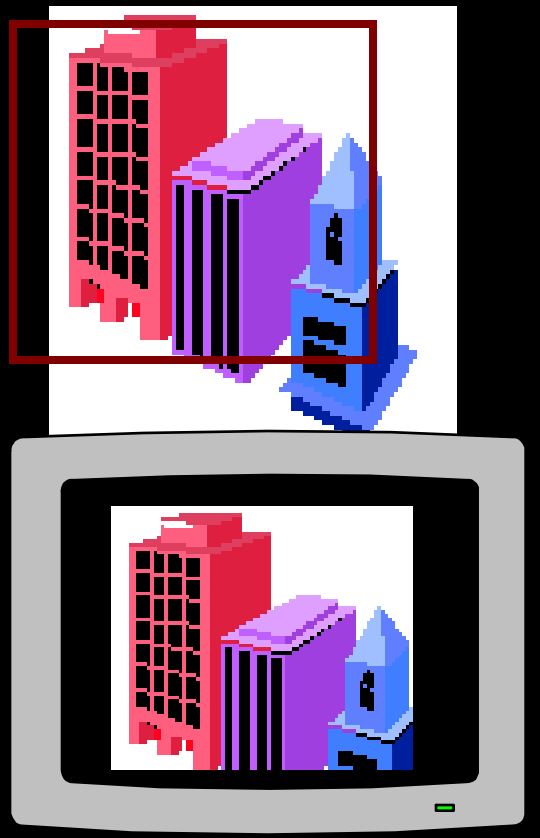
- Dispositivo “abstrato” que permite imaginar uma câmera virtual posicionada em qualquer lugar da cena (mundo 3D), apontada também para um lugar qualquer da cena, orientada de uma forma qualquer, que resulte numa imagem 2D de forma análoga àquilo que podemos fazer no mundo real.
- A câmera faz o “mapeamento” entre coordenadas 3D e 2D

Conceito de câmera em CG

camera



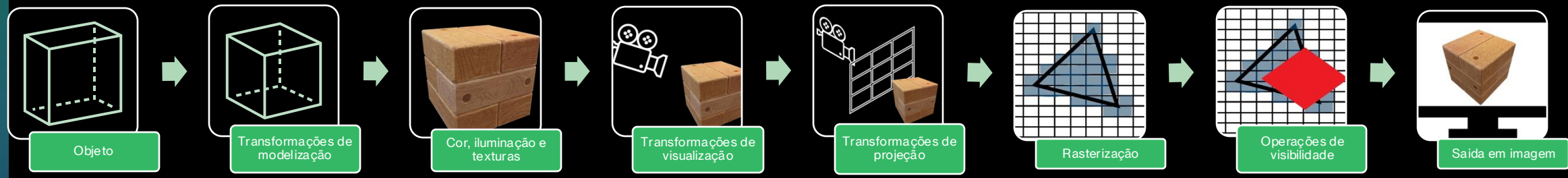




3D

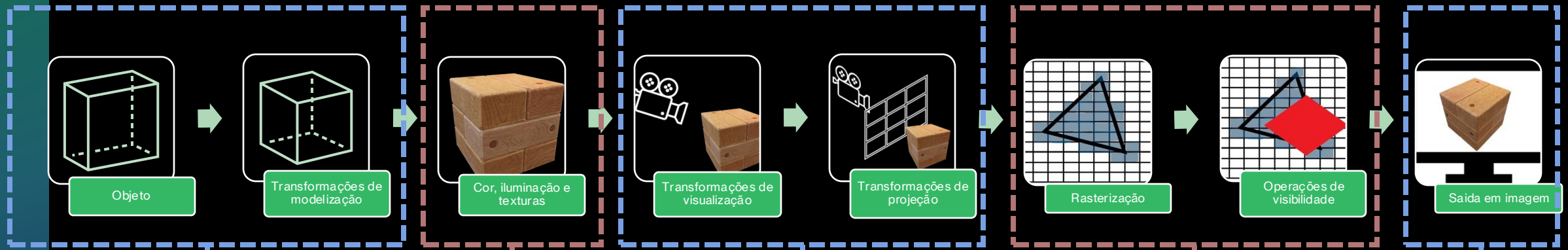
2D

Render pipeline



Pipeline de renderização POLIGONAL

Render pipeline

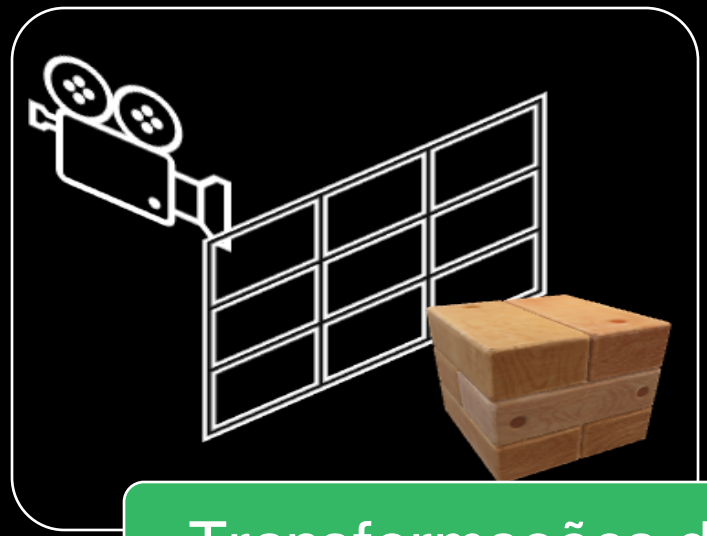


Coordenadas (operações geométricas)

Operações com cor e processos de finalização

Câmera e projeção

Transformações de visualização

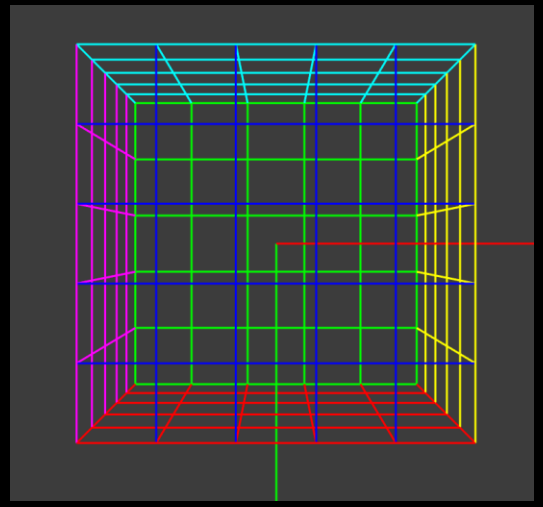


Transformações de projeção

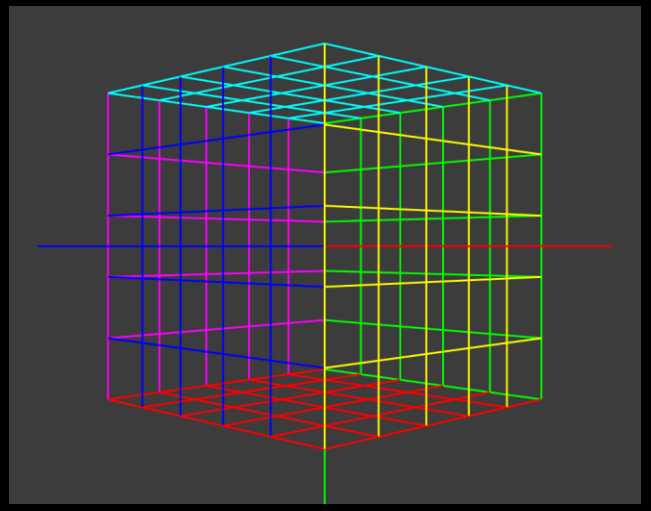
Ponto de partida

- Há diferença entre “posicionar e apontar” uma câmera virtual abstrata vs transformar (rotate, translate, scale) a cena ?

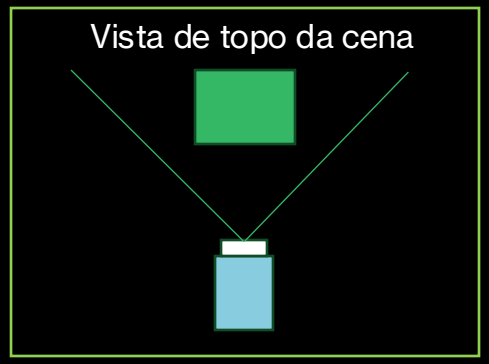
Câmera vs transformações: A

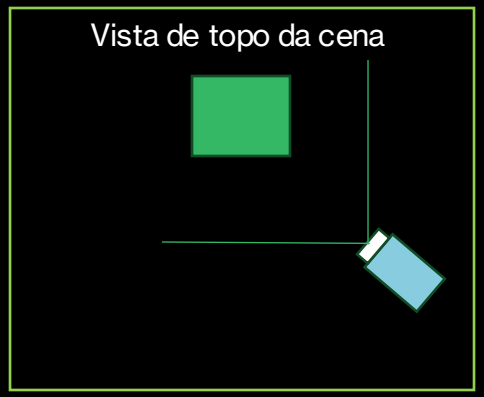


Câmera frontal

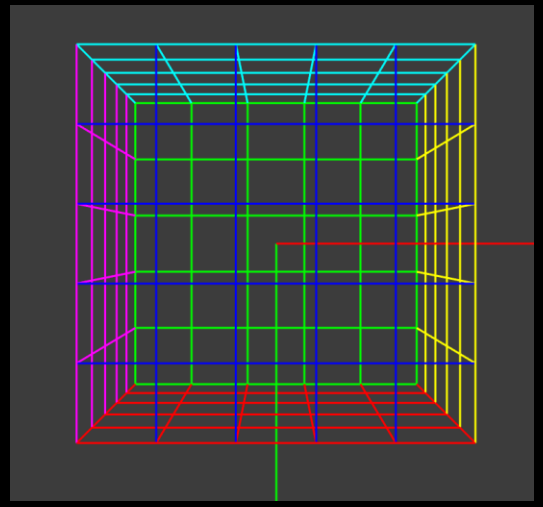


Câmera na “esquina” olhando para a quina do cubo



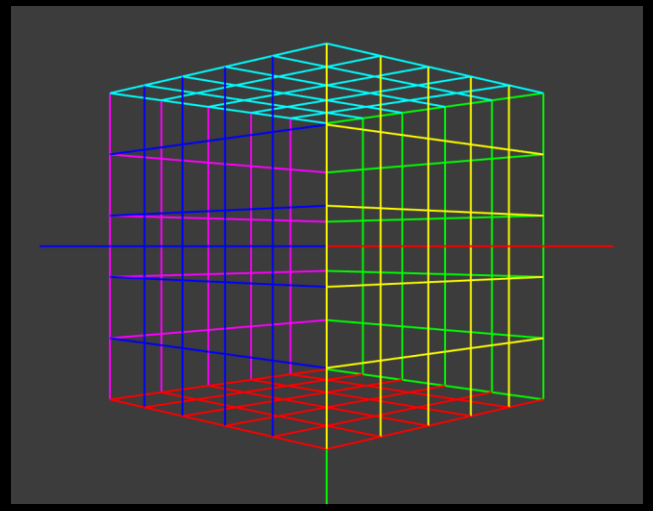


Câmera vs transformações: B



Câmera frontal

glRotatef(-45, 0, 1, 0);



Câmera na “esquina” olhando para a quina do cubo

Câmera vs transformações: A vs B

- Na perspectiva do utilizador, que está olhando para tela:
- A = B?

SIM !

- Se a rotação for aplicada na matriz ModelView (que é o que fizemos até então), o resultado é sim o mesmo.
- Inclusive, a implementação é realizada através de transformações geométricas.
  - Calculamos a matriz de mudança de base (mundo->câmera), e multiplicamos os vértices por esta matriz. Tal qual as transformações tradicionais.

Surge nova pergunta:

- Mas então, qual é a vantagem de pensar numa câmera virtual?

Câmera virtual

- Permite trabalhar a visualização da cena de forma intuitiva, análoga ao que fazemos no mundo real.
- Permite criar efeitos de projeção similares aos das lentes das câmeras reais.

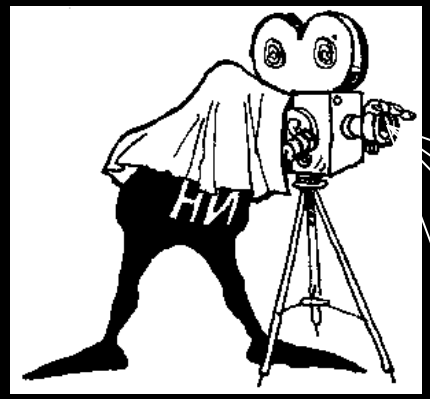
Guia para entender e utilizar a câmera virtual: sumário

1. Entender os sistemas de coordenadas envolvidos.
2. Entender como calcular a matriz de mudança de base (mudar de um sistema para o outro).
3. Entender o que são projeções e quais os tipos que podemos utilizar.
4. Entender como calcular a matriz de projeção.
5. Entender o que é volume de visualização e aplica-lo na matriz de projeção.
6. Implementar tudo isso no OpenGl.

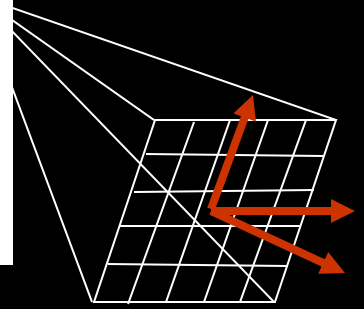
1. Entender os sistemas de coordenadas

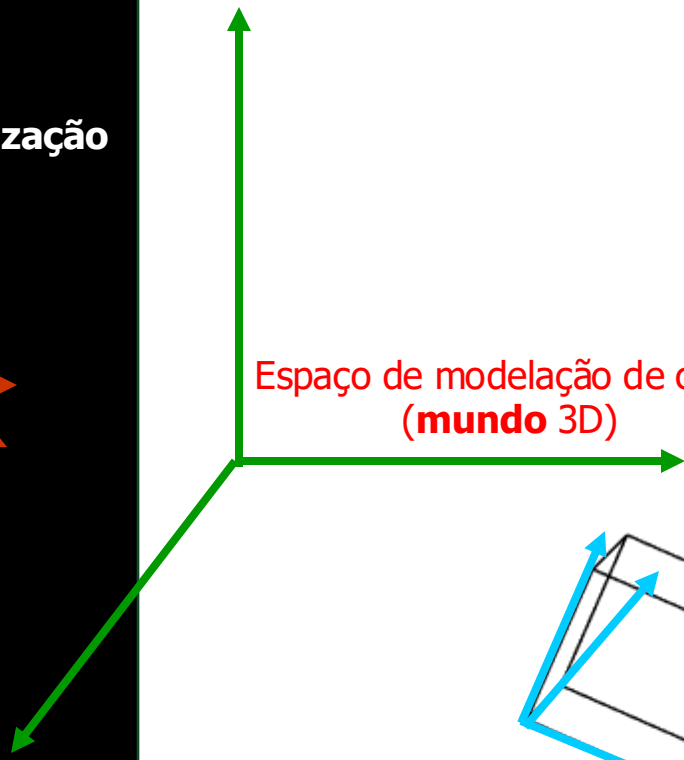
- Quando criamos uma cena e objetos, temos 4 sistemas de coordenadas:
  - Coordenadas do objeto.
    - Definido com vértices com base numa origem que definimos.
  - Coordenadas da cena.
    - Posicionamos os objetos com base numa origem da cena que definimos.
      - No OF, essa origem está definida como o topo superior esquerdo. (como isso é feito???)
  - Coordenadas do espaço de visualização
    - Câmera + projeção + volume de visualização
  - Coordenadas 2D da janela de visualização
    - Operações de recorte, posicionamento e escala da janela do aplicativo

1. Entender os sistemas de coordenadas



Espaço de **visualização**

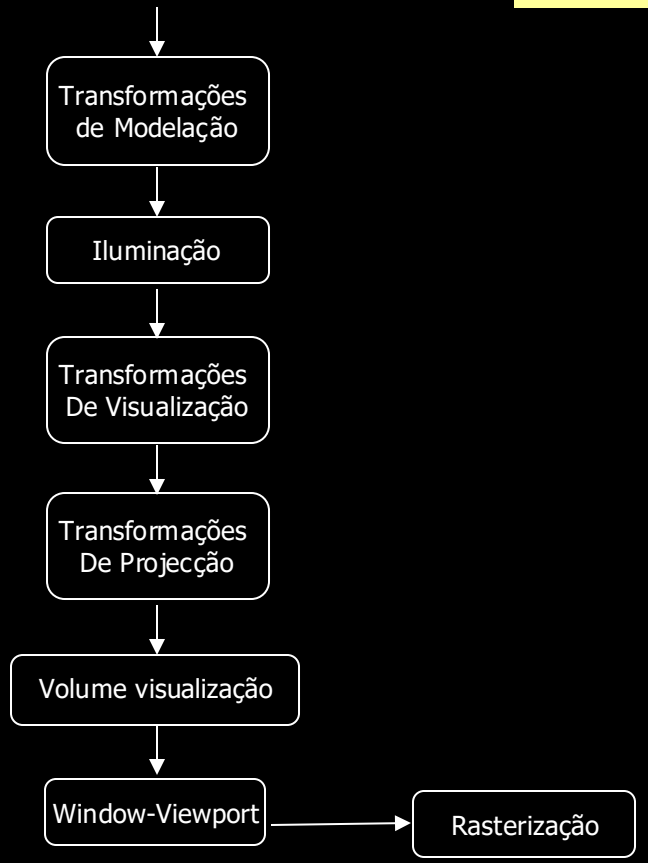


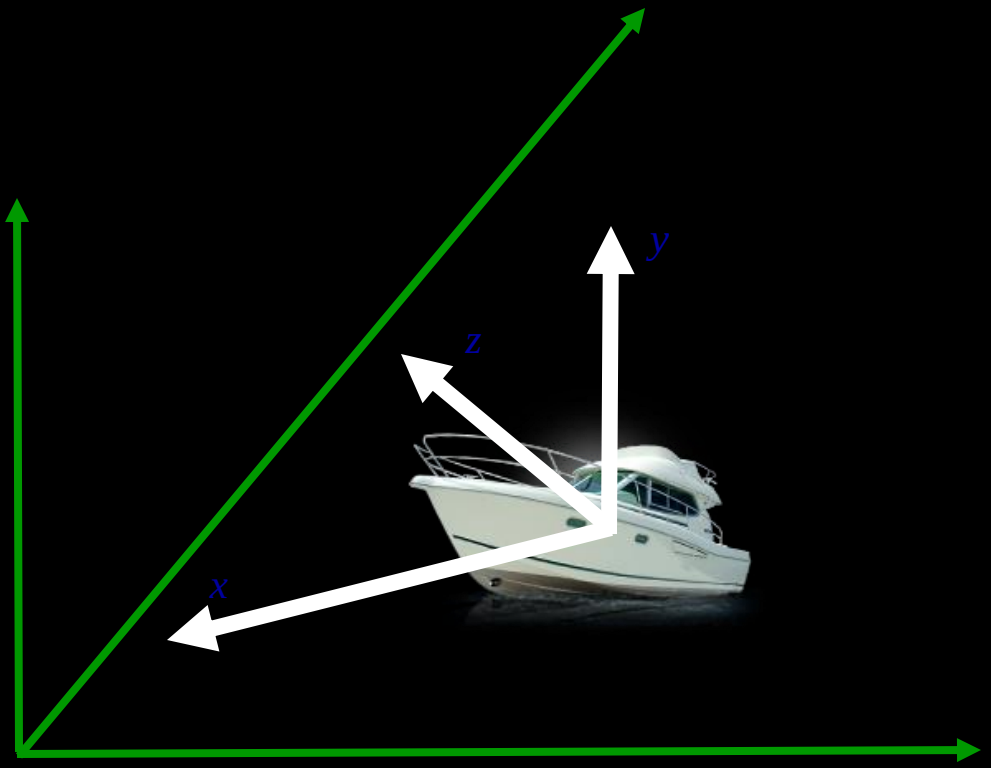


1. Entender os sistemas de coordenadas

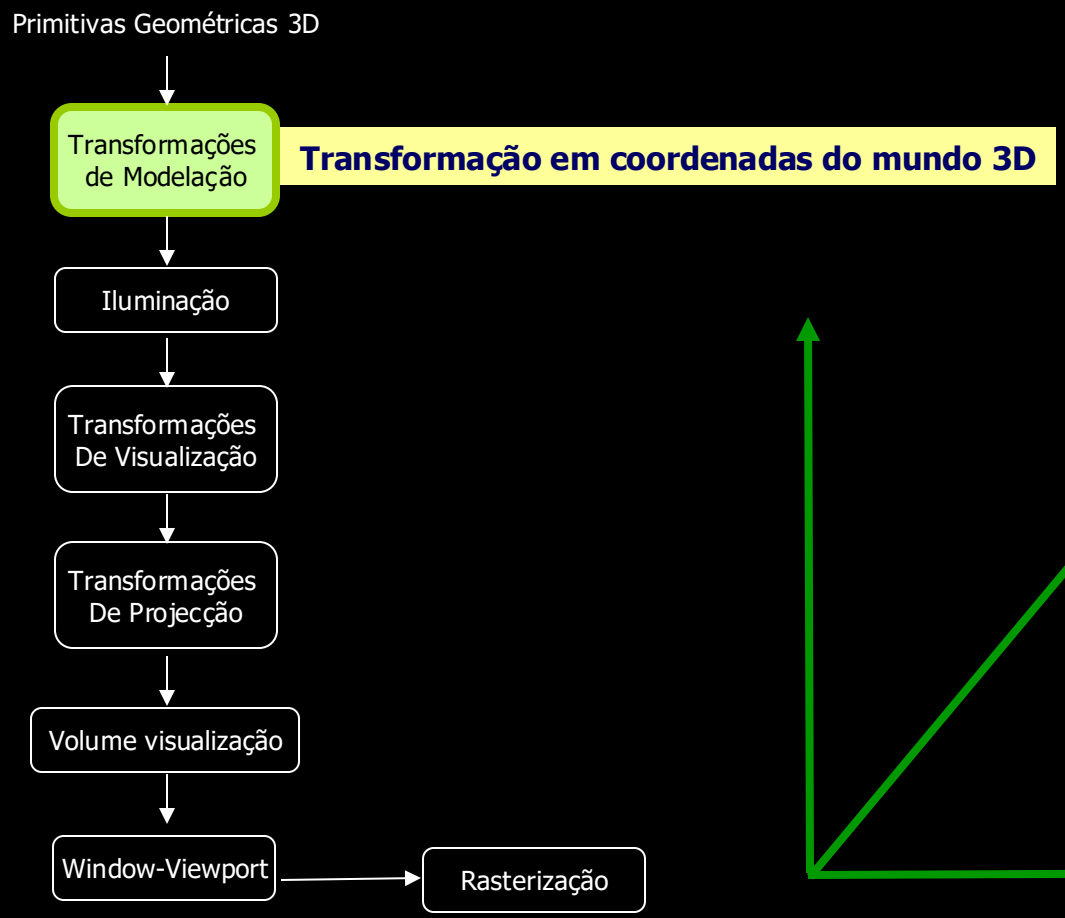
Primitivas Geométricas 3D

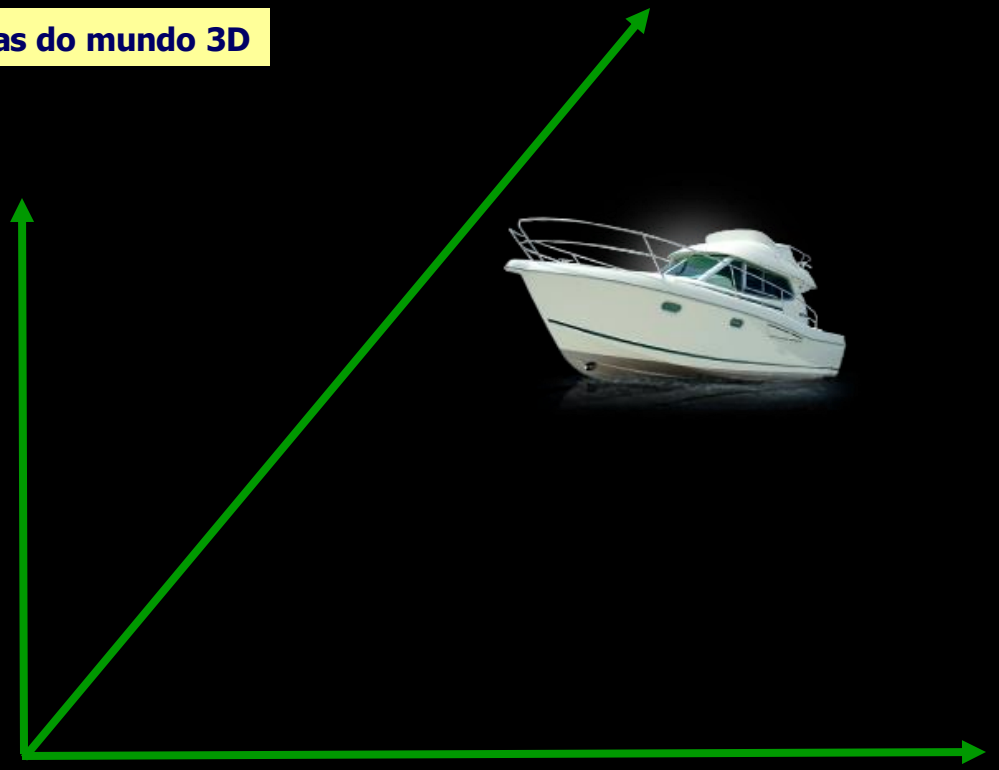
Coordenadas do objecto



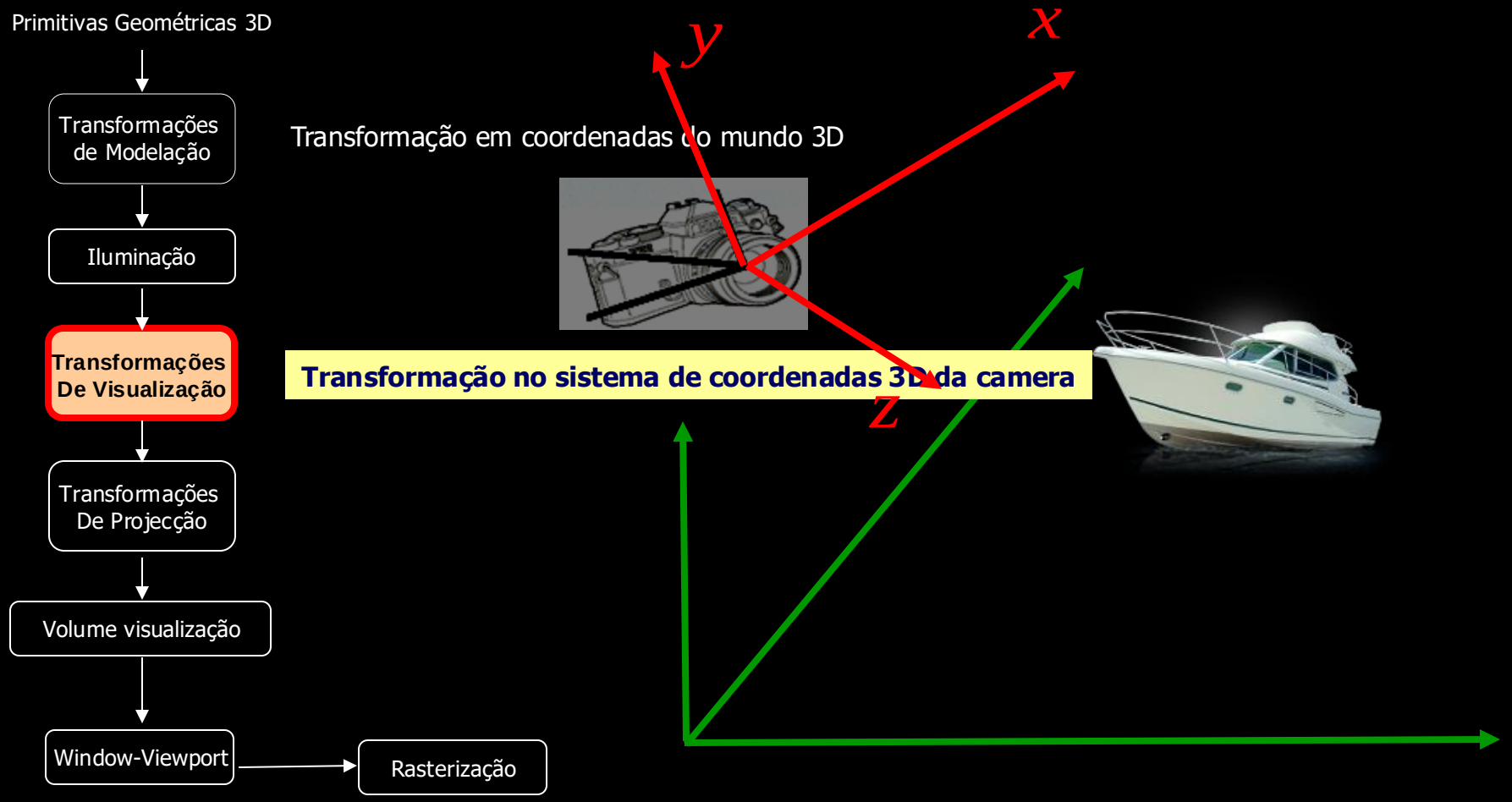


1. Entender os sistemas de coordenadas

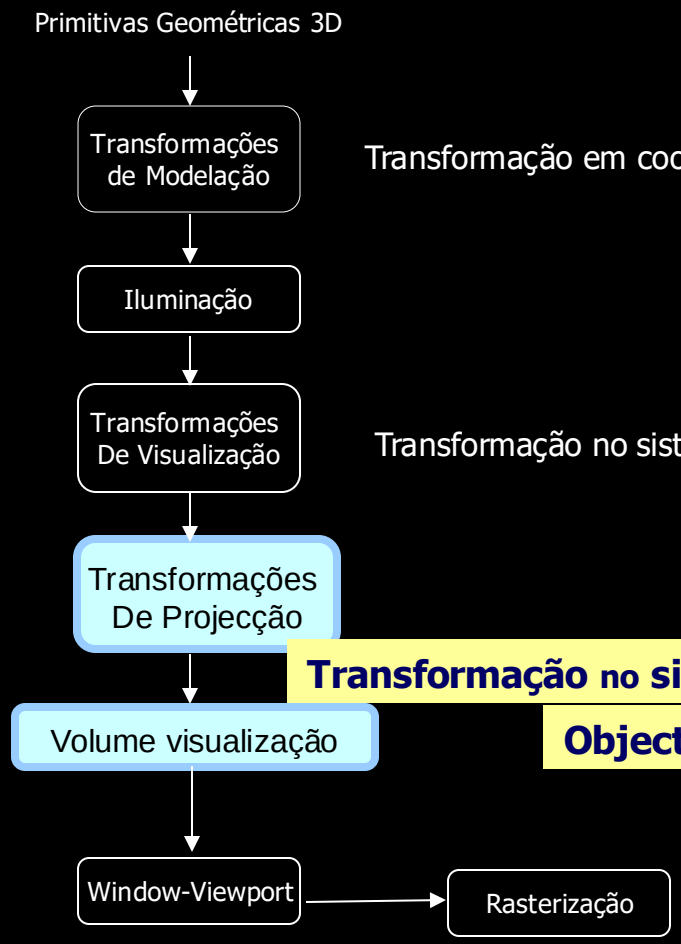




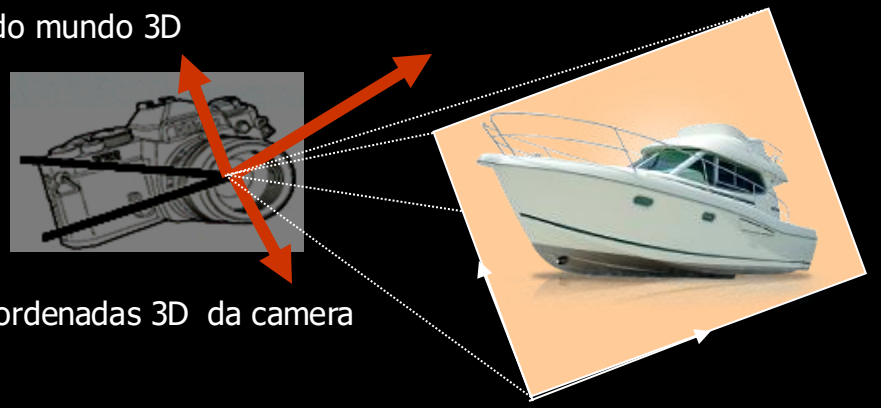
1. Entender os sistemas de coordenadas



1. Entender os sistemas de coordenadas



Transformação em coordenadas do mundo 3D

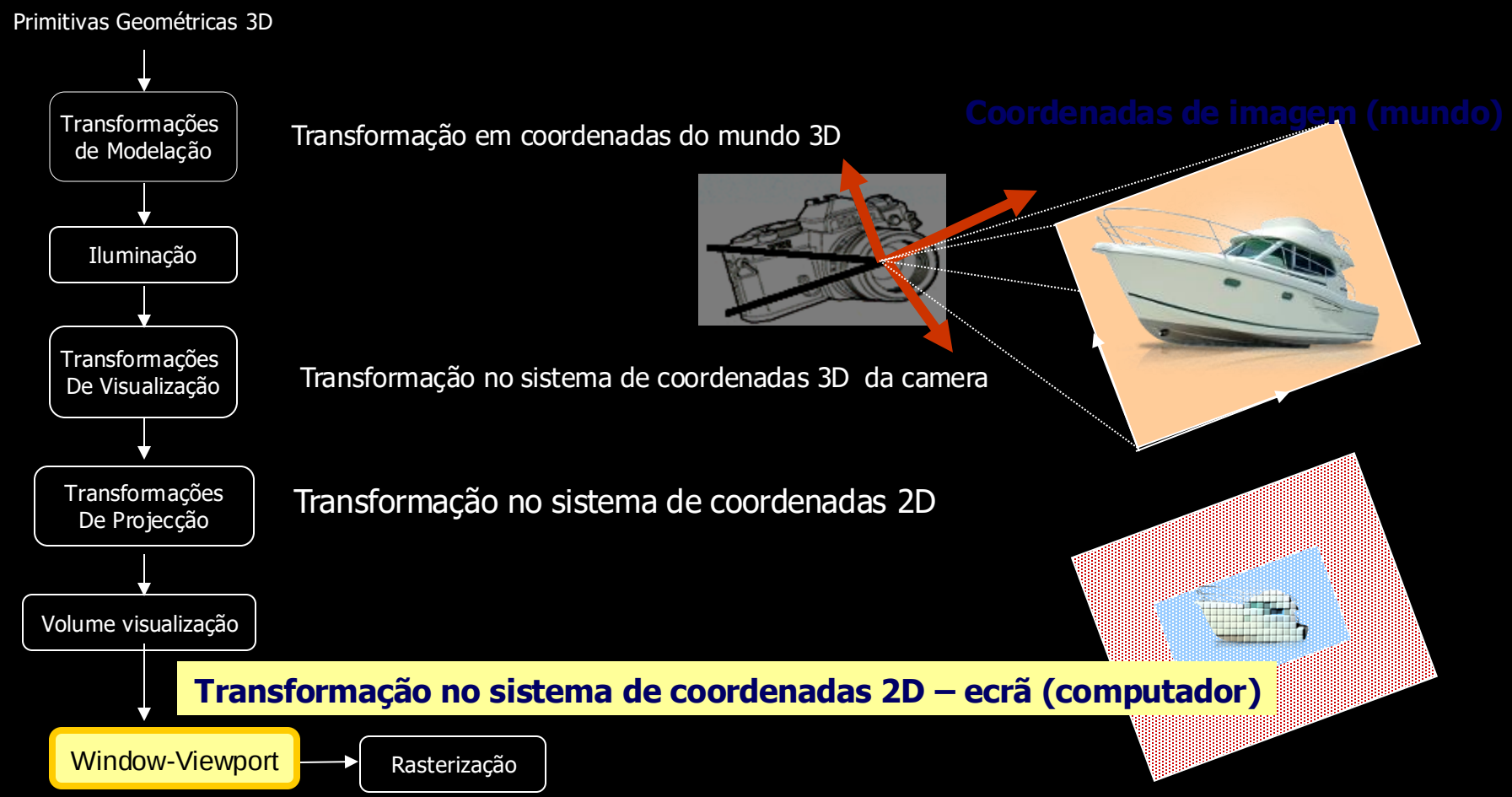


Transformação no sistema de coordenadas 3D da camera

Transformação no sistema de coordenadas 2D (mundo/camera/imagem)

Objectos fora do volume visualização

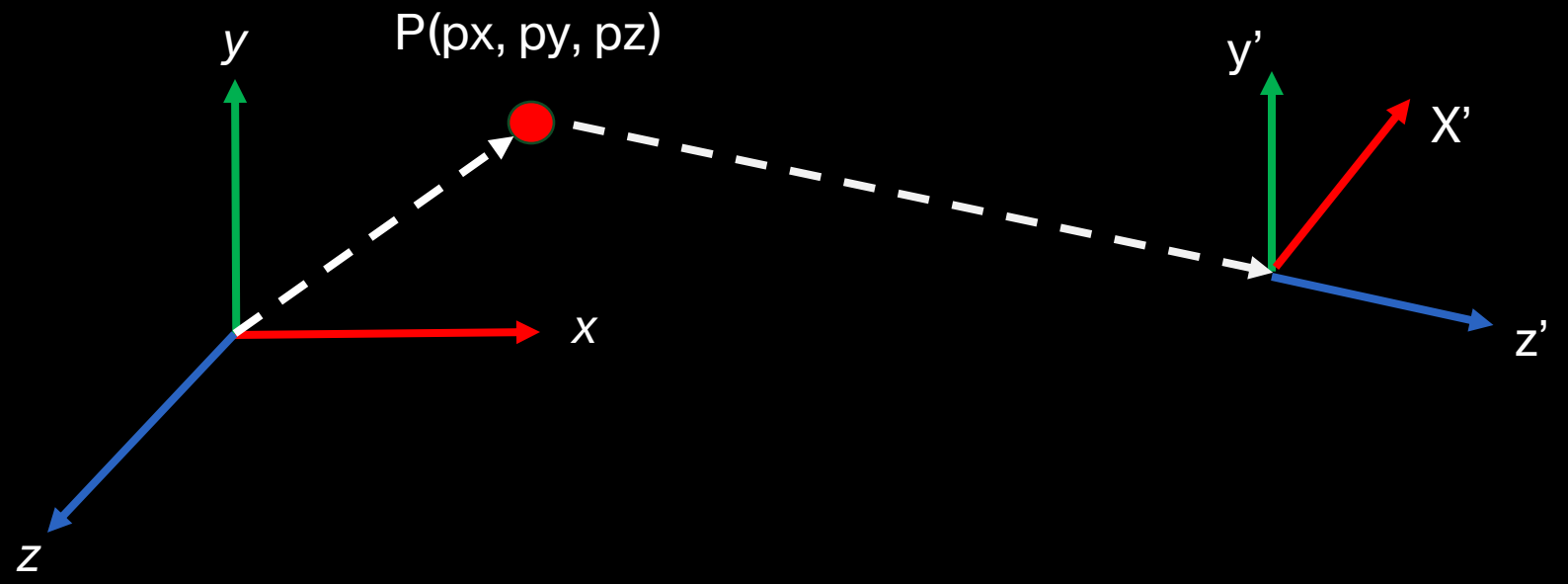
1. Entender os sistemas de coordenadas



2. Mudança de base

- Base = conjunto de vetores linearmente independentes (por ex: ortogonais).
- O 1º passo para implementar uma câmera virtual na cena, e modificar a visualização de acordo com a posição e orientação da câmera é, determinar a matriz de transformação das coordenadas do mudno para as da câmera.
- Ou seja:
  - Dado um ponto P=(px, py, pz) no sistema de coordenadas do mundo (x, y, z), quero saber quais são suas coordenadas num sistema que tem a posição da câmera como origem.

2. Mudança de base



P'(px', py', pz') = ?

2. Mudança de base

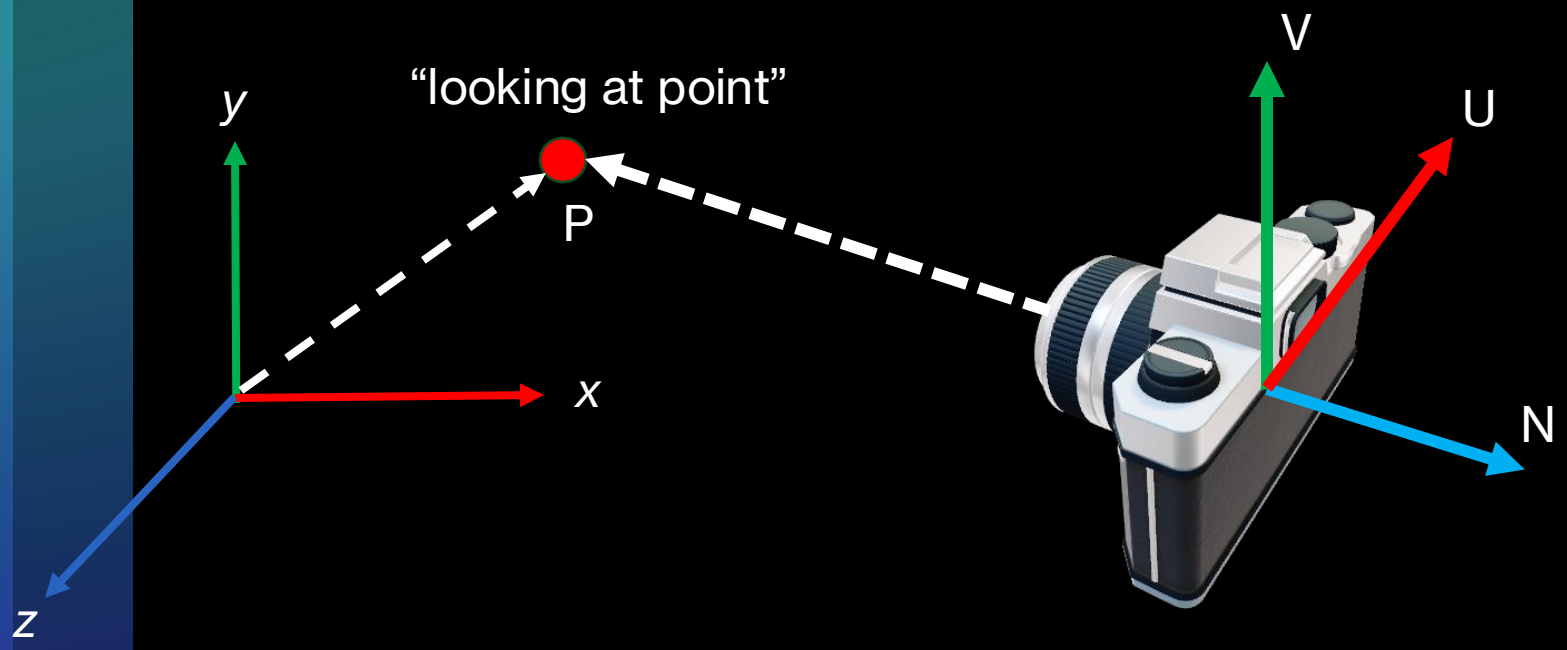
- Ver CG_LEI_2024_T05_mudaBase.pdf

2. Mudança de Base: algoritmo UVN - lookAt

- É fácil conseguir enxergar a matriz de transformações necessárias para configurar uma vista de uma cãmera se esta está a “olhar” numa perspetiva alinhada com os eixos. Mas para obtermos efeitos de cãmera mais complexos, temos de pensar numa estratégia que generalize o problema.
- Em computação gráfica, esta estratégia é implementada pelo algoritmo UVN, mais conhecido como “Lookat”

2. Mudança de base - lookat

- O algoritmo “Lookat” parte da seguinte definição para os eixos que formam a base da cãmera:



U = camera “right vector
V = camera “up” vector
N = vetor que aponta na direção negativa do “lookat”

2. Mudança de base - lookat

- O algoritmo “lookat” é usalmente implementado com uma função que recebe 3 parâmetros:
  - Ponto (x, y, z) – posição da camera na base de coordenadas do mundo
  - Ponto (x, y, z) – posição para onde a camera está a olhar (lookat position)
  - Vetor (vx, vy, vz) – direção e sentido do vetor “up” (V), que determina a rotação da camera com relação ao eixo N

- Lookat((x, y, z), (x, y, z), (vx, vy, vz))

2. Mudança de base - lookat

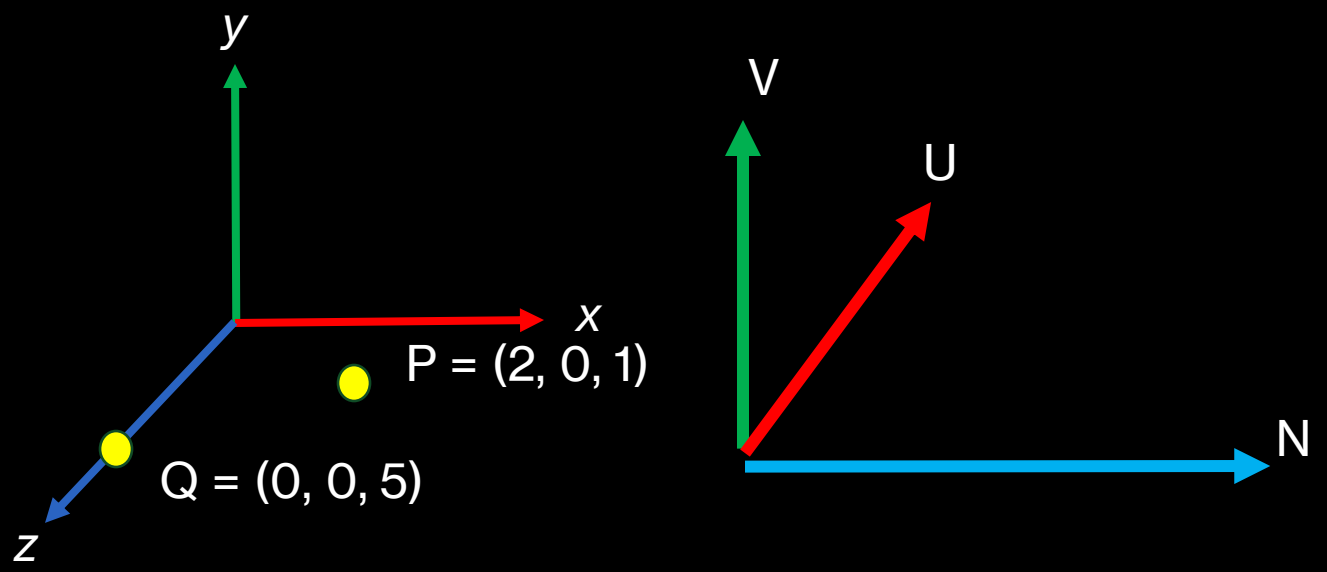
- O algoritmo consegue então calcular a matriz de transformação para a simular a vista camera da seguinte forma:
  1. Cálculo do vetor N:
     - N = (Cam – P) normalizado (Cam é a posição x, y, z da câmera nas coordenadas do mundo, P é a posição x, y, z do ponto para onde a camera está a olhar)
  2. U = (Up x N) normalizado (x é o produto vetorial, V é o vetor Up fornecido pelo utilizador)
  3. V = (N x U) normalizado (V é uma segunda iteração do vetor Up, assim, ao desenvolver, basta utilizar uma direção aproximada para a orientação da camera e o algoritmo se encarrega de modificar para a orientação correta)
  4. Determinar a matriz de mudança de base e aplicar à ModelviewMatrix:

$$M = \begin{pmatrix} Ux & Uy & Uz & -dot(Cam, U) \\ Vx & Vy & Vz & -dot(Cam, V) \\ Nx & Ny & Nz & -dot(Cam, N) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2. Exemplo

Qual a posição do ponto P na base UVN da camera, para uma função lookat com os seguintes parâmetros:

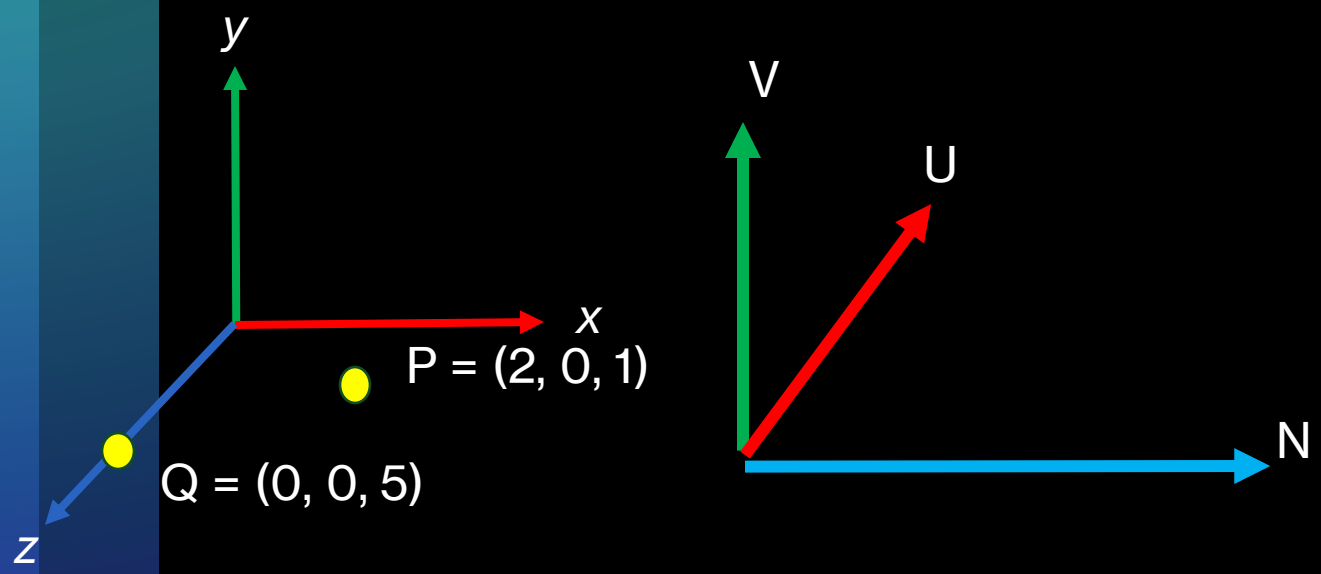
Lookat((5, 0, 5), (0, 0, 5), (0, 1, 0))



2. Exemplo

Qual a posição do ponto P na base UVN da camera, para uma função lookat com os seguintes parâmetros:

Lookat((5, 0, 5), (0, 0, 5), (0, 1, 0))



1. N = (5, 0, 5) – (0, 0, 5) = (5, 0, 0)
   - N = norma(N) = (1, 0, 0)
2. U = Up x N = (0 1, 0) x (1, 0, 0) = (0, 0, -1)
3. V = N x U = (1 0, 0) x (0, 0, -1) = (0, 1, 0)
4. M =

$$\begin{pmatrix} 0 & 0 & -1 & 5 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -5 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

P’ = MP = (4, 0, -3)