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)

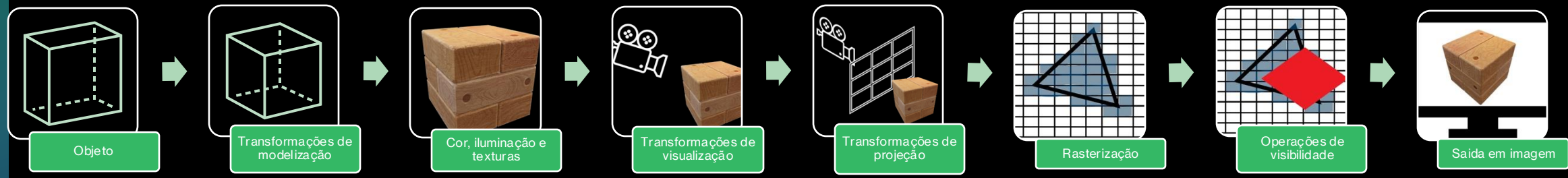
Hugo Amaro (hamaro@dei.uc.pt)

T-04: Modelview Matrix, Hierarquia e combinação de transformações

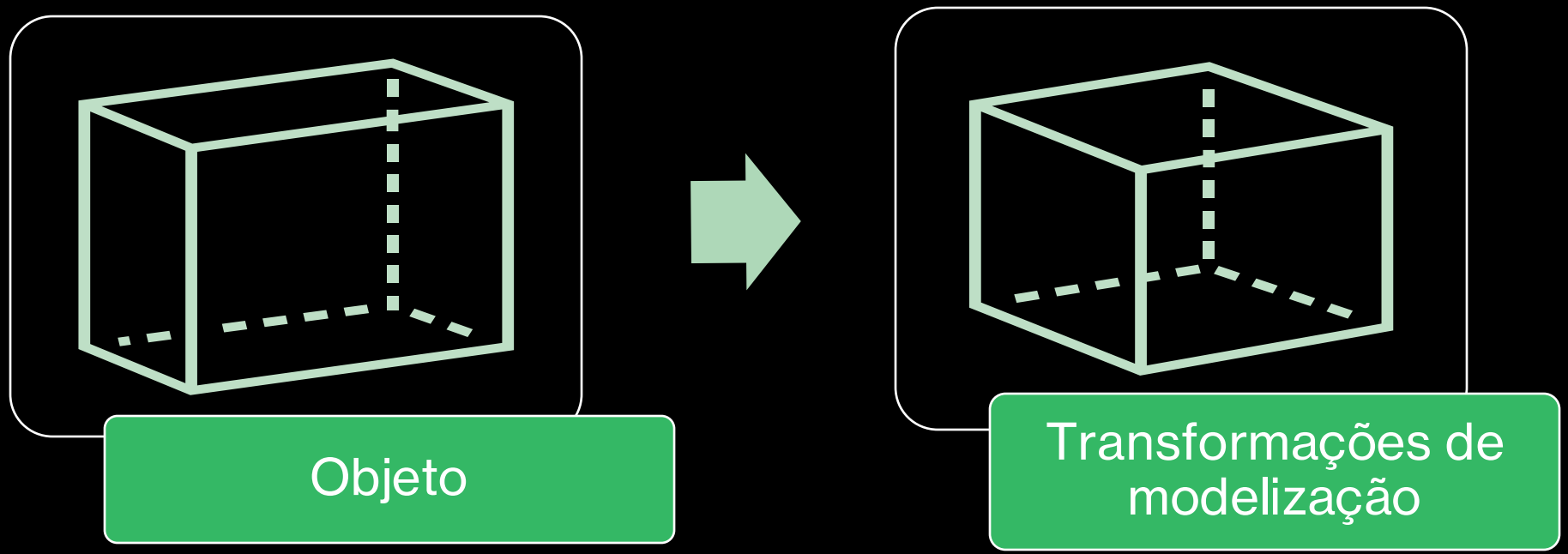
Objetivos da aula

- Entender o Stack de transformações geométricas e suas implicações na ModelviewMatrix
- Entender como combinar e criar relações hierárquicas com transformações

Pipeline de renderização



Pipeline de renderização: etapa de modelização e transformação do modelo



Transformações combinadas

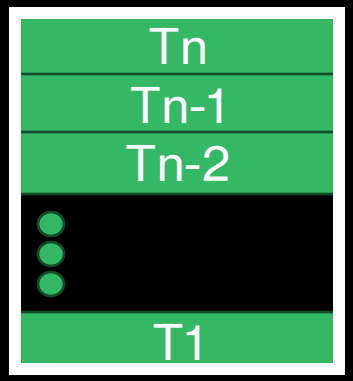
- Na aula T03, vimos como combinar as transformações para dimensionar, rotacionar e posicionar modelos
- Vimos também que a transformação final é a combinação de todas as trasformações em uma única matriz
  - M_final = Tn * Tn-1 * ... * T1
  - Onde Tn é a última transformação pela ordem “intuitiva”, mas a 1ª na ordem do código openGl
- Na aula PL03, vimos que podemos limitar o escopo das transformações utilizando os comandos
  - glPushMatrix()
  - glPopMatrix()

Hierarquia de transformações

- Hoje, vamos tentar responder:
  - Mas o que de fato fazem os comandos push e pop?
  - Quando preciso deles?

O stack de matrizes do OpenGl

- No OpenGl, todas as transformações de modelação (e também visualização, que vamos ver na próxima aula T05) ficam guardadas e combinadas numa pilha de transformações que resulta numa matriz, que é aplicada a todos os vértices que aparecem após a transformação
- Esta matrix é chamada MODELVIEW MATRIX



Tn*Tn-1*...*T1

S*r	r	r	Tx
r	S*r	r	Ty
r	r	S*r	Tz
0	0	0	1

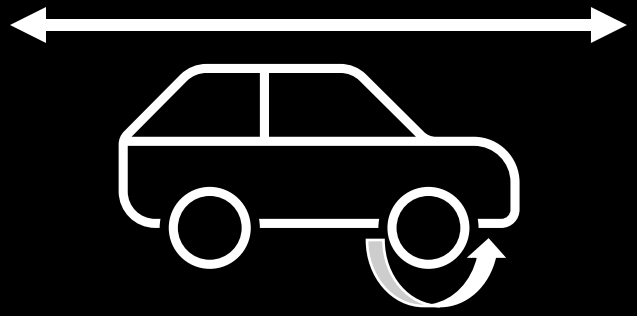
O stack de matrizes do OpenGl

- Em muitas situações, queremos poder “isolar” um conjunto de vértices (modelos) de certas transformações, ou, criar um stack que seja específico para este grupo.
- Nestas situações, utilizamos os comandos:
  - glPushMatrix() -> para iniciar uma pilha
  - glPopMatrix() -> para apagar a pilha que foi inicializada

Hierarquia de transformações

- Em muitas situações, é desejável criar uma espécie de ligação em entre elementos de um modelo, de forma que eles sofram transformações comums a vários elementos e ao mesmo tempo sofram outras que são específicas:
- Exemplos:

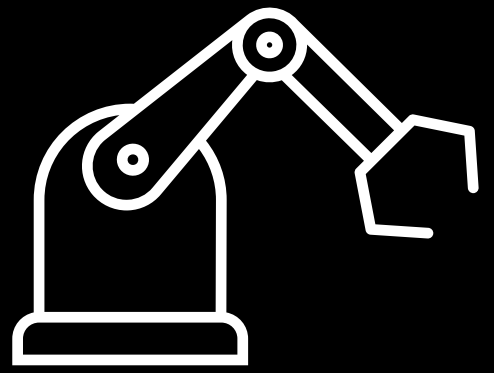
Translação frente/trás
Comum a todos os elementos



A rotação só afeta as rodas

Num painel de GUI,
a posição e tamanho dos elementos deve ser relativa ao painel

Num braço robótico, a posição/rotação dos dedos é resultado da combinação da rotação das outras articulações + a sua movimentação independente

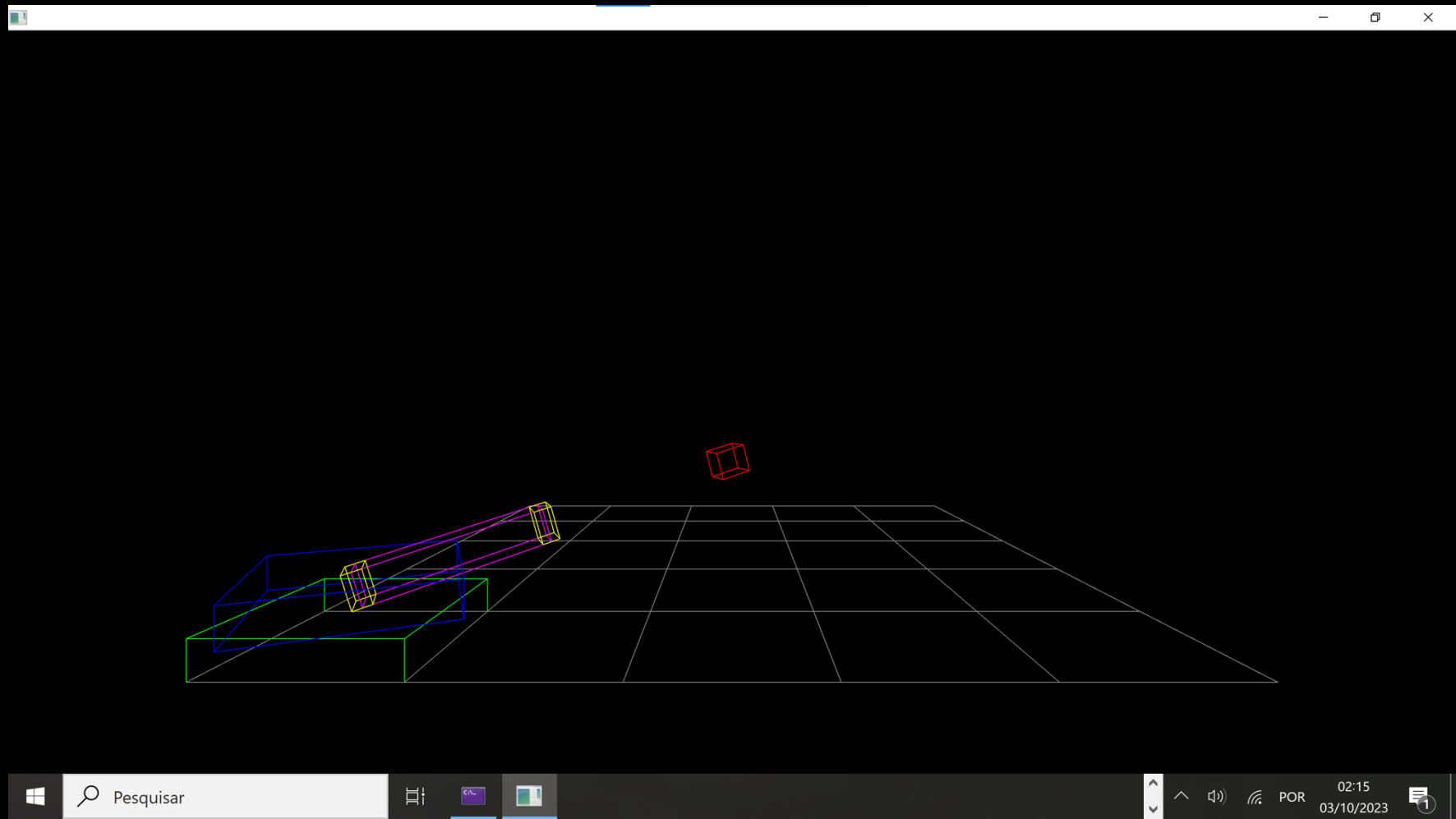


Hierarquia de transformações

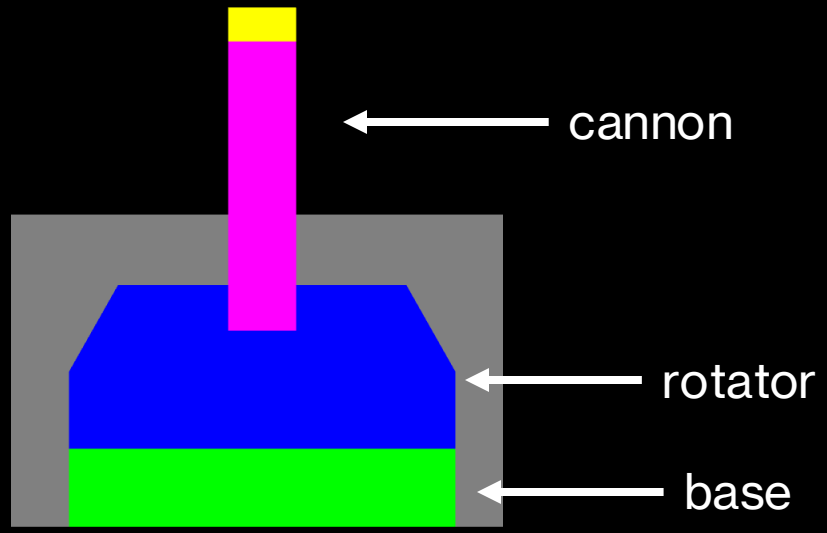
- Nestas situações, podemos utilizar push/pop para criar uma hierarquia de transformações, criando uma relação dinâmica entre as partes do modelo.

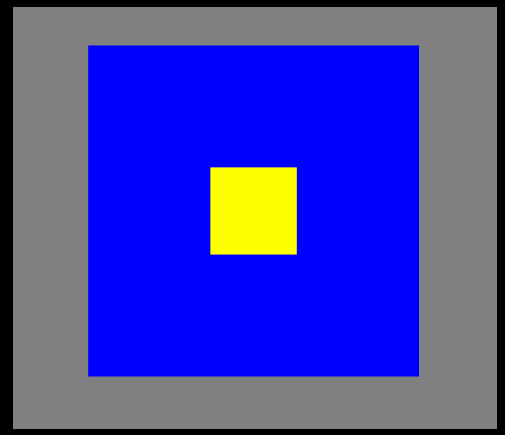
- Vamos ver como isso funciona implementando um exemplo.
- Nosso objetivo final (para ser realizado nas aulas PL04) será o de fazer um tanque de guerra 3D.
- Aqui nesta aula faremos uma simplificação análoga em 2D para entender como funciona uma hierarquia de transformações.
- Utilize o código CG_LEI_2024_T04_transformacoesHierarquia.zip

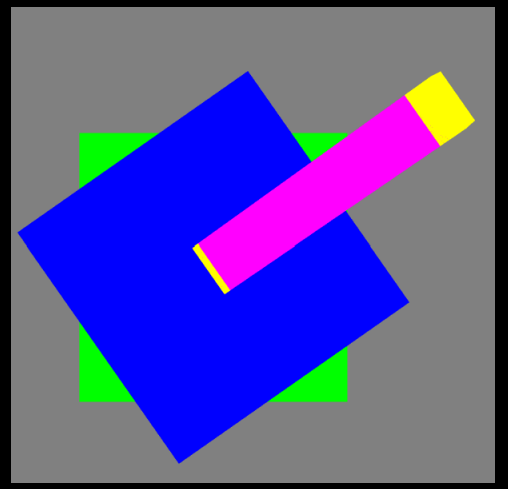
Tank shooter

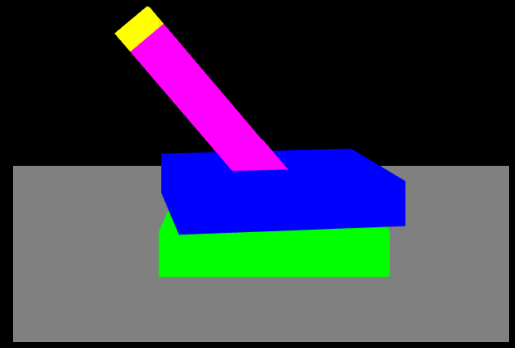


O tanque





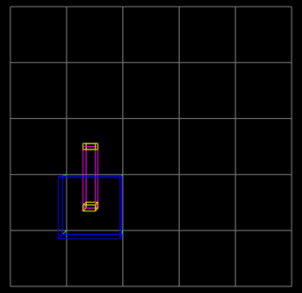


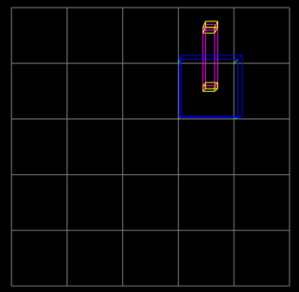


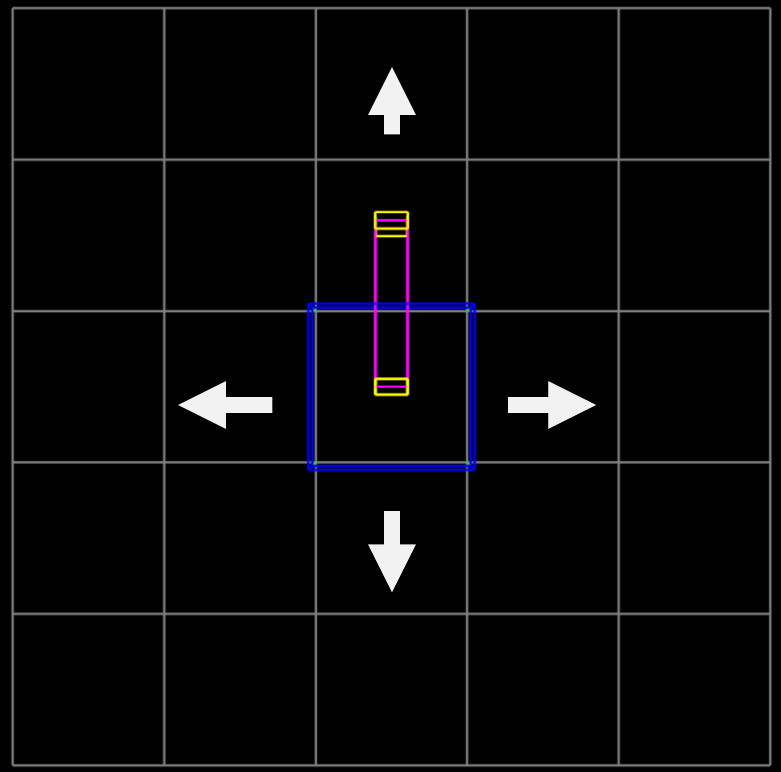
movimentos do tanque: base

A base pode movimentar em 4 direções: cima, baixo, esquerda, direita.

Os passos são quantizados de forma que o tanque está sempre em perfeitamente alinhado a um dos quadrados da malha.

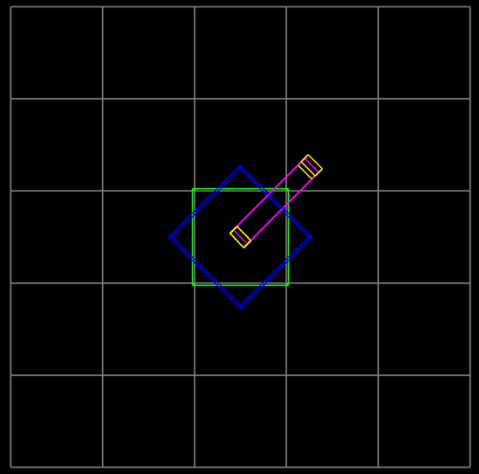


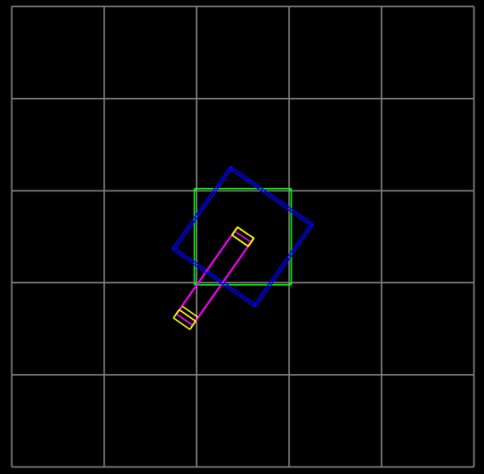


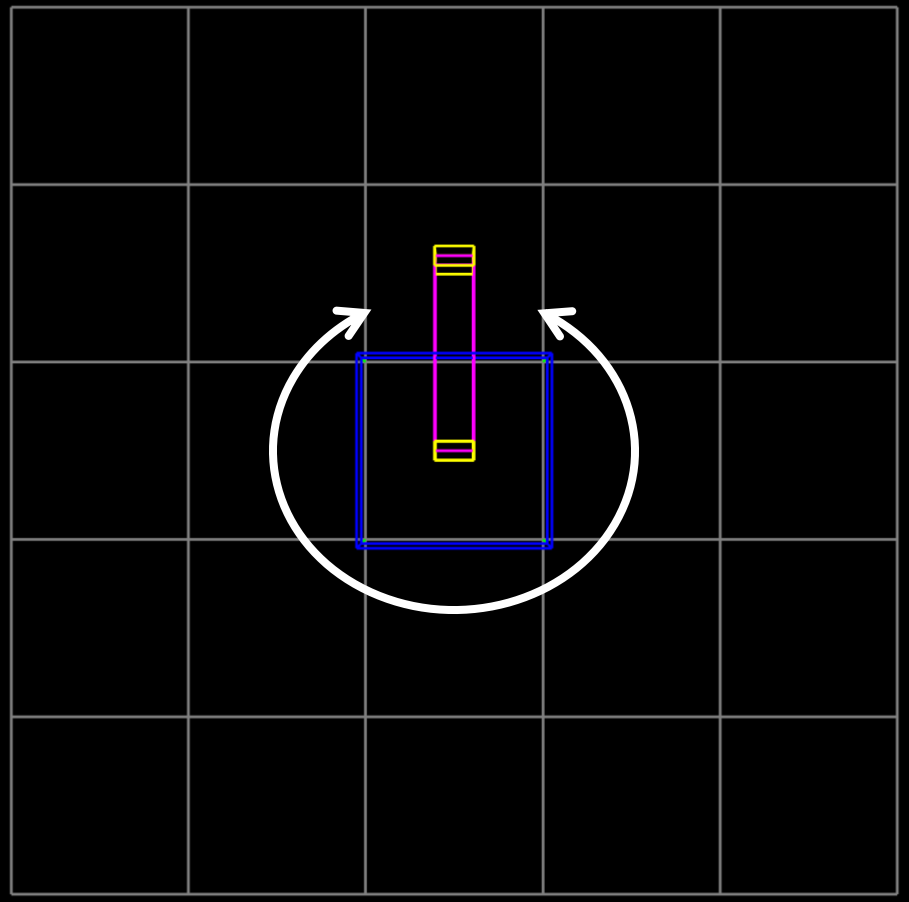


Movimentos do tanque: rotator

O rotator pode girar livremente em torno de seu próprio eixo (perpendicular ao plano horizontal), nos dois sentidos (clockwise, counter-clockwise)

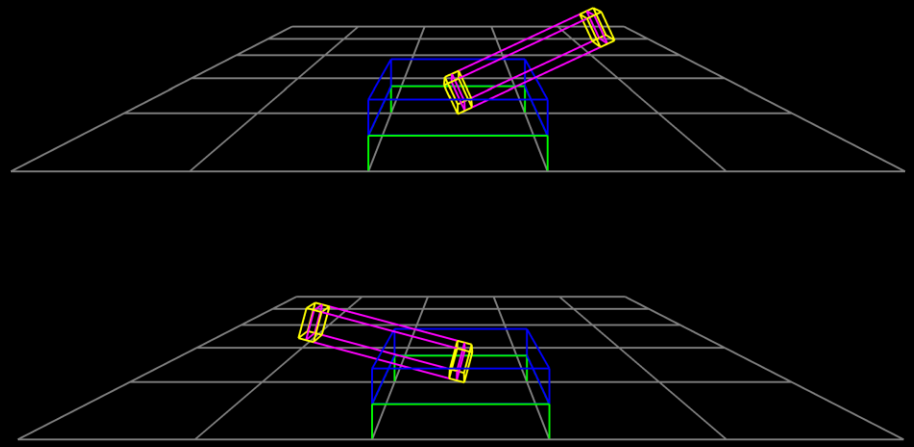


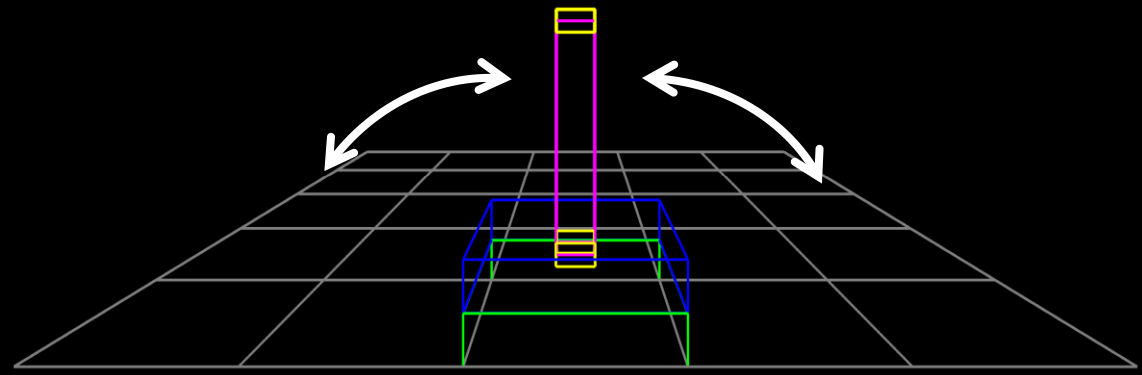




Movimentos do tanque: canhão

O canhão pode inclinar até um máximo de 90° nos dois sentidos, com eixo de rotação em sua base.

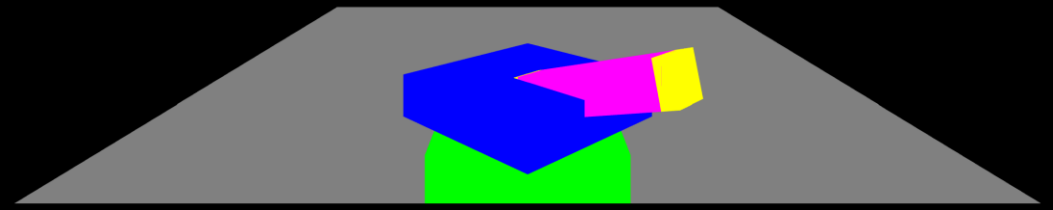


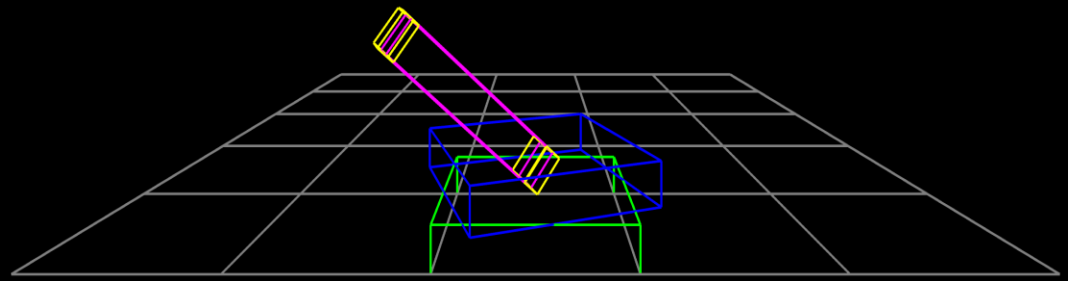


Movimentos do tanque: combinação

O tanque está construído de forma que há uma “Irelação” entre os elementos:

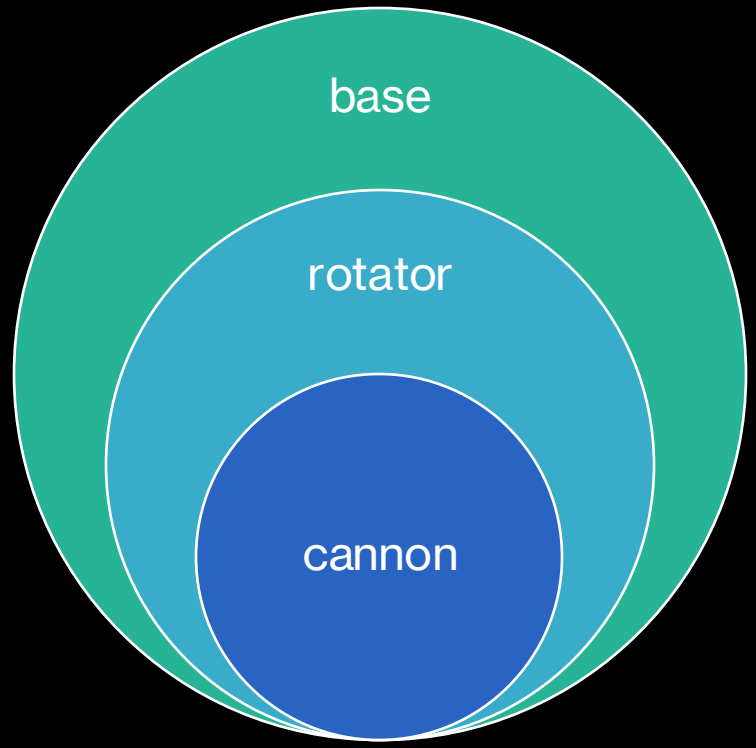
- O movimento da base, move todas as parte.
- A rotação do rotator, move-o e também o canhão
- A inclinação do canhão afeta apenas a ele próprio





Relação entre as partes

- Há uma “hierarquia” entre as partes, que é garantida pelo encadeamento das transformações geométricas:
  - A translação da base afeta as 3 partes
  - A rotação do rotator afeta o rotator e o canhão
  - A rotação (inclinação) do canhão afeta apenas a ele próprio



Relação entre as partes

- Há uma “hierarquia” entre as partes, que é garantida pelo encadeamento das transformações geométricas:
  - A translação da base afeta as 3 partes
  - A rotação do rotator afeta o rotator e o canhão
  - A rotação (inclinação) do canhão afeta apenas a ele próprio
- Isto significa que no código, teremos também que ter uma hierarquia de transformações:

glTranslate

glRotate

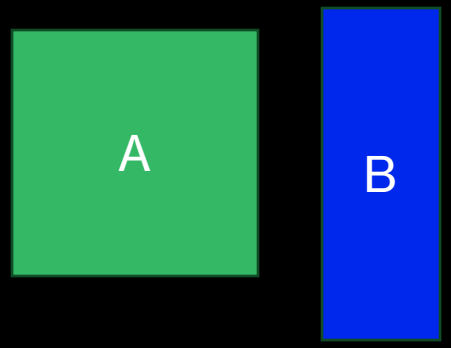
glRotate

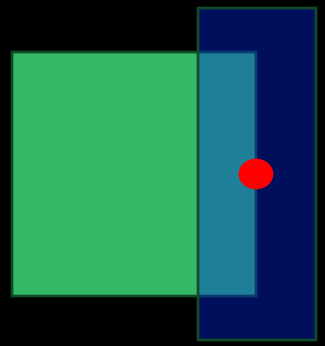
Hierarquia e ordem entre transformações

- Para entender como fazer esse tipo de implementação, vamos “quebrar” o problema em partes e analisar cada uma delas:
  1. Transladar objetos de forma conjunta
  2. Rotacionar objetos em eixo resultante de translação
  3. Rotação combinada após transformações

1 – translação de objetos acoplados

- Problema:
  - Tenho 2 retângulos A e B; B está acoplado ao A na posição indicada na figura; quero implementar a hierarquia de transformações de tal forma que quando A for transladado, B se mova junto de A.

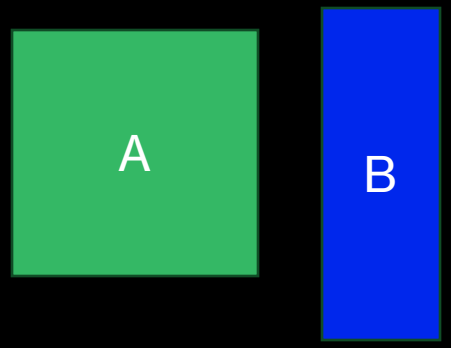


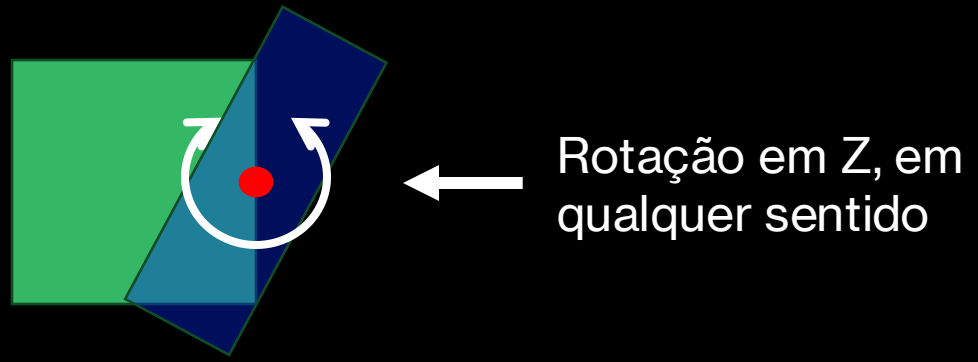


O centro de B está perfeitamente alinhado com o ponto central do segmento de reta que forma o lado esquerdo de A.

2 – rotação de objetos acoplados

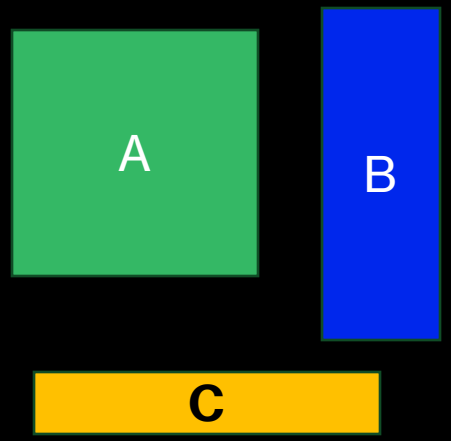
- Problema:
  - Partindo do resultado obtido em 1 (slide anterior), queremos agora que o retângulo B possa girar em torno de si próprio (eixo z).

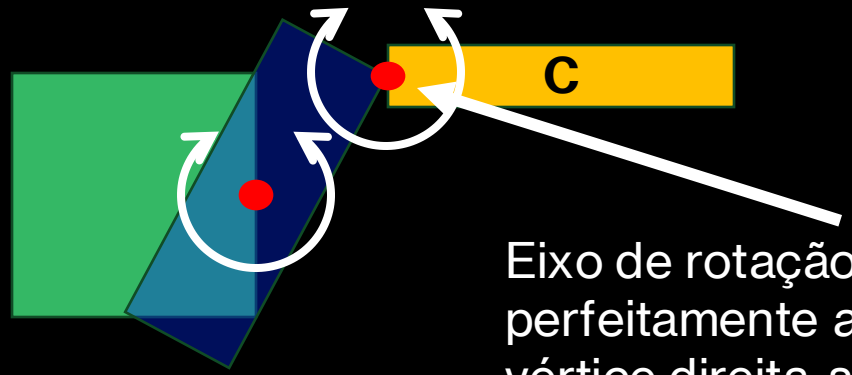




3 – rotação combinada

- Problema:
  - Partindo do resultado obtido em 2 (slide anterior), queremos agora adicionar um 3° retângulo C, que está acoplado em B e pode girar livremente conforme apresentado na figura.

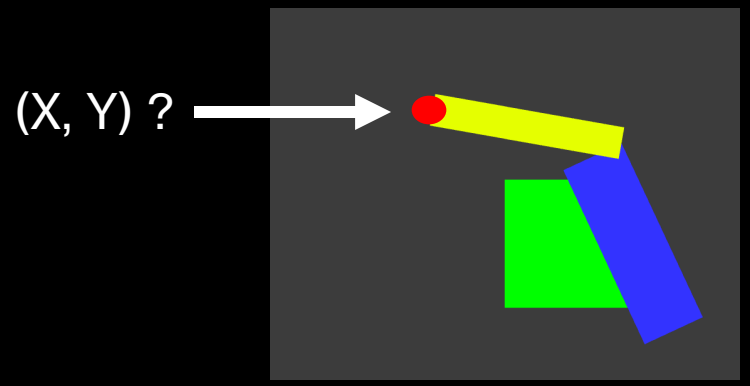




Eixo de rotação de C está perfeitamente alinhado com o vértice direita-superior de B

Modelview matrix

- Após as translações e rotações, qual a posição do centro do retângulo C?
- Se eu quiser saber se a extremidade do retângulo C colide com alguma coisa, como posso saber onde estão seus vértices?
- Se o retângulo C fosse um “canhão”, qual a direção dos projéteis disparados?



Modelview matrix

- Toda vez que adiciono uma transformação à pilha, a Modelview Matrix, reflete essa transformação.
- Se “perguntar” o estado da matriz no lugar adequado, posso saber as translações (x, y) daquele lugar.

Implementado em cg_extras.h

```
inline ofVec3f getModelViewMatrixPos() {
    GLfloat Matriz[4][4];
    glGetFloatv(GL_MODELVIEW_MATRIX, &Matriz[0][0]);
    ofVec3f aux;
    aux.x = Matriz[3][0];
    aux.y = Matriz[3][1];
    aux.z = Matriz[3][2];
    //cout << endl << aux.x<<" "<<aux.y<<" "<<aux.z;
    return aux;
}
```

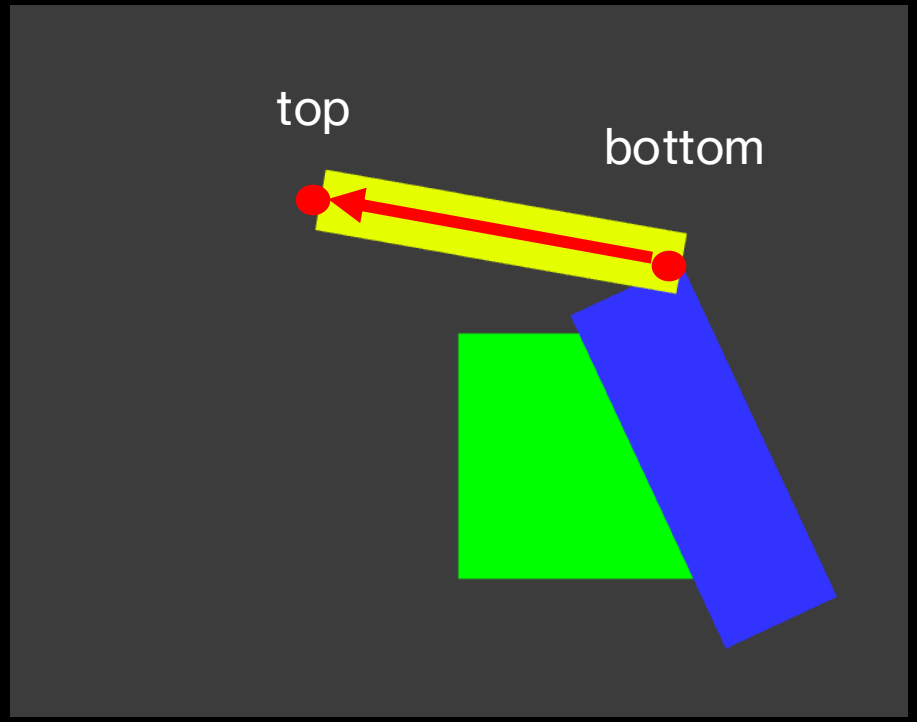
Modelview matrix

- Uma vez que eu tenho a matriz modelview, posso desenhar objetos utilizando as transformações nela contida. Para isso faço:
  1. Faço reset na matriz (matriz identidade)
  2. Aplico a matriz modelview que guardei numa variável
  3. Desenho os objetos

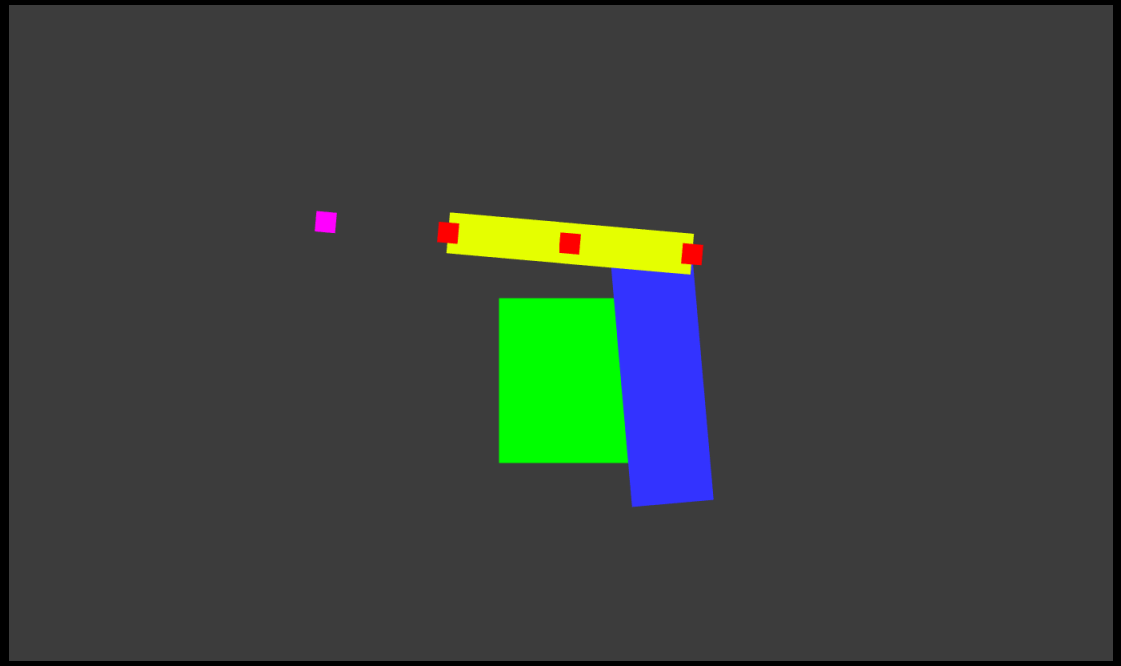
```
//desenha ponto no C_center
glColor3f(1, 0, 0);
glPushMatrix();
glMatrixMode(GL_MODELVIEW);//ativo a modelview matrix
glLoadIdentity();//reset da modelview matrix
glMultMatrixf(&C_center[0][0]);//aplico a transformação
glScalef(C_w * 0.5, C_w * 0.5, 1.);
rectFill_unit();
glPopMatrix();
```

Modelview matrix

- Se quiser saber a direção do retângulo C para usá-lo como um canhão, posso fazer:
  - dirVec = Top(x,y) – Bottom(x, y)
- Posso usar o dirVec como vetor de velocidade do projétil:
  - Projetil(X, Y) += dirVec x intensidade



Resultado final



Questões ou dúvidas?