Transformações Geométricas em WebGL

Objetivos

- Aprender a efetuar transformações geométricas em WebGL
  - Rotações
  - Translação
  - Mudança de Escala
- Funções e conceitos relevantes oferecidos na biblioteca MV.js

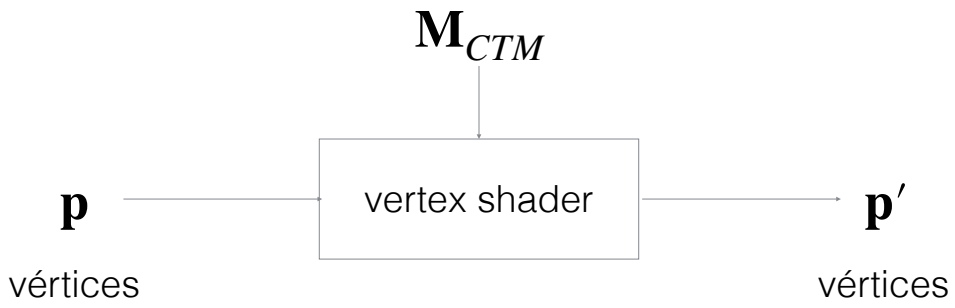
Como modelar uma cena?

- Tipicamente, um programador duma aplicação gráfica 3D tem à sua disposição um conjunto de objetos ditos primitivos, os quais vai manipular para compor a sua cena.
- Cada (instância de um) objeto terá o seu tamanho ajustado, será orientado e posicionado na cena
- Em WebGL (e em qualquer API com pipeline gráfico programável), usamos o vertex shader para transformar os vértices dos objetos

images retrieved from

CTM-Current Transformation Matrix

- Dum ponto de vista conceptual, podemos pensar que todos os vértices que passam pelo pipeline são transformados por uma matriz de transformação corrente ($\mathbf{M}_{CTM}$), a qual pode ir sendo manipulada.
- O valor desta matriz é determinado pela aplicação e colocado à disposição do programa GLSL (no vertex shader).



Operações sobre a M_{CTM}

- A matriz de transformação corrente pode ser alterada por afetação direta dum novo valor:

Afetar com a matriz identidade: $\mathbf{M}_{CTM} \leftarrow \mathbf{I}$
Afetar com uma dada matriz $\mathbf{M}$: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}$
Afetar com uma matriz de Translação: $\mathbf{M}_{CTM} \leftarrow \mathbf{T}$
Afetar com uma matriz de Rotação: $\mathbf{M}_{CTM} \leftarrow \mathbf{R}$
Afetar com uma matriz de Escala: $\mathbf{M}_{CTM} \leftarrow \mathbf{S}$

- ou por concatenação (exemplo com pós-multiplicação):

Pós-multiplicação com uma matriz dada: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{M}$
Pós-multiplicação com uma matriz de translação: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{T}$
Pós-multiplicação com uma matriz de rotação: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{R}$
Pós-multiplicação com uma matriz de mudança de escala: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{S}$

Exercício: Rotação em torno dum eixo arbitrário (desviado da origem)

- Começar com a matriz identidade: $\mathbf{M}_{CTM} \leftarrow \mathbf{I}$
- Trazer um ponto $\mathbf{p}_f$ do eixo para a origem: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{T}(-\mathbf{p}_f)$
- Efetuar a rotação: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{R}(\theta)$
- Mover o ponto $\mathbf{p}_f$ de volta: $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{T}(\mathbf{p}_f)$
- Resultado Acumulado: $\mathbf{T}(-\mathbf{p}_f) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(\mathbf{p}_f)$

Ordem errada!
(com pós-concatenação)

Não esquecer que os pontos são multiplicados do lado direito: $\mathbf{p}' = \mathbf{M}_{CTM} \cdot \mathbf{p}$

Exercício: Rotação em torno dum eixo arbitrário (desviado da origem)

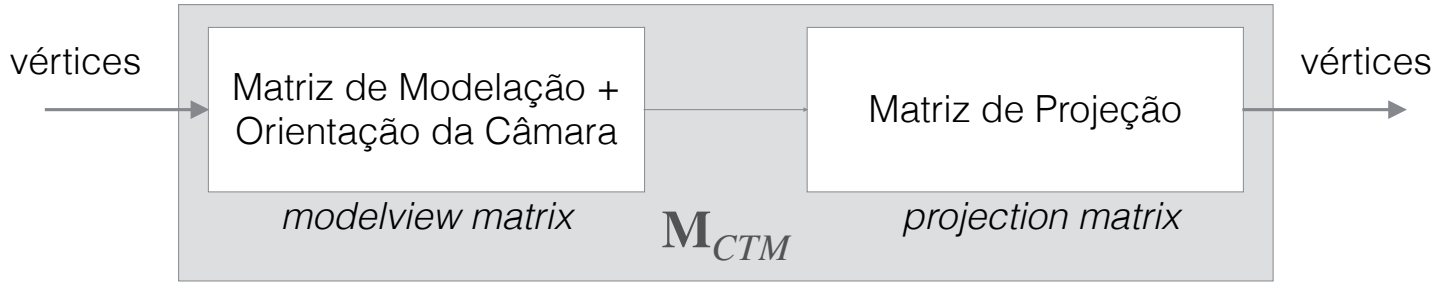
- É necessário inverter a ordem das transformações!

1. $\mathbf{M}_{CTM} \leftarrow \mathbf{I}$
2. $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{T}(\mathbf{p}_f)$
3. $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{R}(\theta)$
4. $\mathbf{M}_{CTM} \leftarrow \mathbf{M}_{CTM} \cdot \mathbf{T}(-\mathbf{p}_f)$

- Resultado Acumulado: $\mathbf{T}(\mathbf{p}_f) \cdot \mathbf{R}(\theta) \cdot \mathbf{T}(-\mathbf{p}_f)$

M_{CTM} em WebGL

- As versões de OpenGL antigas expunham a matriz $\mathbf{M}_{CTM}$ sob a forma do produto de duas matrizes com finalidades diferentes:



- É do nosso interesse emular este processo, mantendo uma matriz dedicada à projeção, separada da matriz usada para instanciar os objetos primitivos e converter as coordenadas dos pontos para o referencial da câmara (posicionar e orientar a câmara).

Modelview+Projection

- A matriz ModelView (M_{MV}) é usada para:
  - Aplicar transformações de modelação/instanciação aos objetos primitivos
    - Passagem do referencial do objeto para o referencial comum da cena - mundo (WC)
  - Posicionar a câmara e orientá-la em relação ao restante da cena
    - Passagem do referencial do mundo (WC) para o referencial da câmara (CC/EC)
    - Pode ser criada por composição de rotações e translações mas a biblioteca MV.js disponibiliza a função lookAt() para este efeito.
- A matriz de projeção (M_{proj}) serve para definir o volume de visão
  - Corresponde à escolha da lente duma máquina fotográfica
- Embora estas matrizes já não façam parte do estado do sistema, é uma boa estratégia usá-las na implementação dos sistemas gráficos (e nas nossas aplicações):

$$\mathbf{p}' = \mathbf{M}_{proj} \cdot \mathbf{M}_{MV} \cdot \mathbf{p}$$

Suporte em MV.js

- Criar uma matriz identidade:

```
m = mat4()
```

- Criar uma matriz de rotação:

```
r = rotateX(angle)
r = rotateY(angle)
r = rotateZ(angle)
r = rotate(angle, vx, vy, vz)
```

- Criar uma matriz de mudança de escala:

```
m = scalem(sx, sy, sz)
```

- Criar uma matriz de translação:

```
m = translate(dx, dy, dz)
```

- Efetuar a pós-multiplicação:

```
m = mult(m, m1)
```

Exemplos

- Rodar 30º em torno dum eixo paralelo a Z, que passa pelo ponto (1,2,3)

```
m = mult(translate(1,2,3), mult(rotateZ(30), translate(-1,-2,-3)))
```

- Aumentar a dimensão x para o dobro e fazer um deslocamento de (2,0,3):

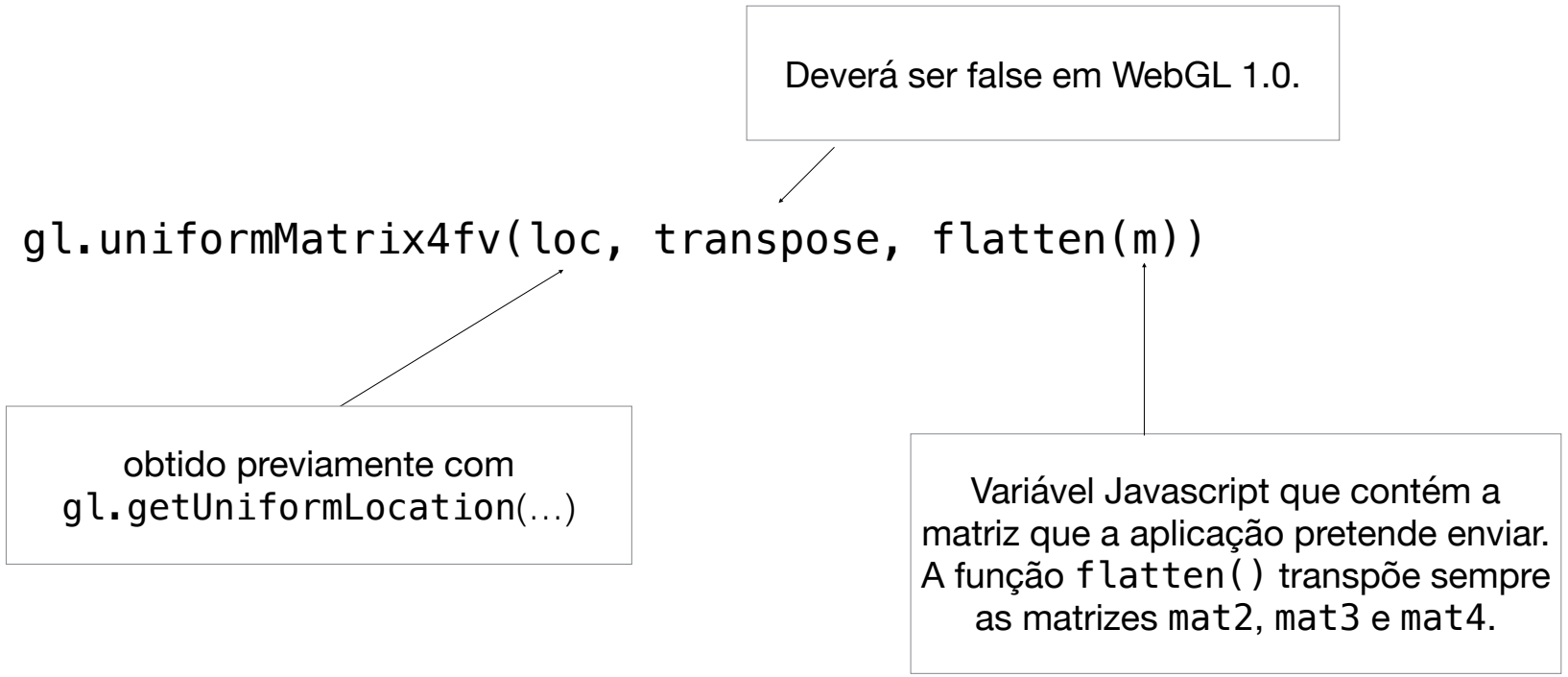
```
m = mult(translate(2,0,3),scalem(2,1,1))
```

Transformações (Matrizes) Arbitrárias

- É possível enviar para o programa GLSL uma matriz 4x4 arbitrária, cujo conteúdo foi previamente definido pela aplicação.
- As matrizes são guardadas como arrays (js) unidimensionais de 16 elementos, mas podem ser acedidas por índices linha, coluna usando o tipo de dados mat4.
- O formato nativo usado pelo WebGL (e pelo OpenGL) requer os elementos dispostos em memória percorrendo primeiro as colunas.
- A função flatten(), da biblioteca MV.js, trata de converter o tipo de dados mat4 (elementos dispostos por linhas) para o formato requerido pelas funções da API WebGL (elementos dispostos por colunas).
- A função gl.uniformMatrix4fv() tem um parâmetro para transpor automaticamente a matriz, mas na versão WebGL 1.0 tem que estar a **false**.

Envio duma matriz de transformação para um programa GLSL

- Supondo que temos uma matriz `m`, de tipo `mat4`, o seu envio para que um shader a possa usar é feito com:



Pilhas de Transformações Geométricas

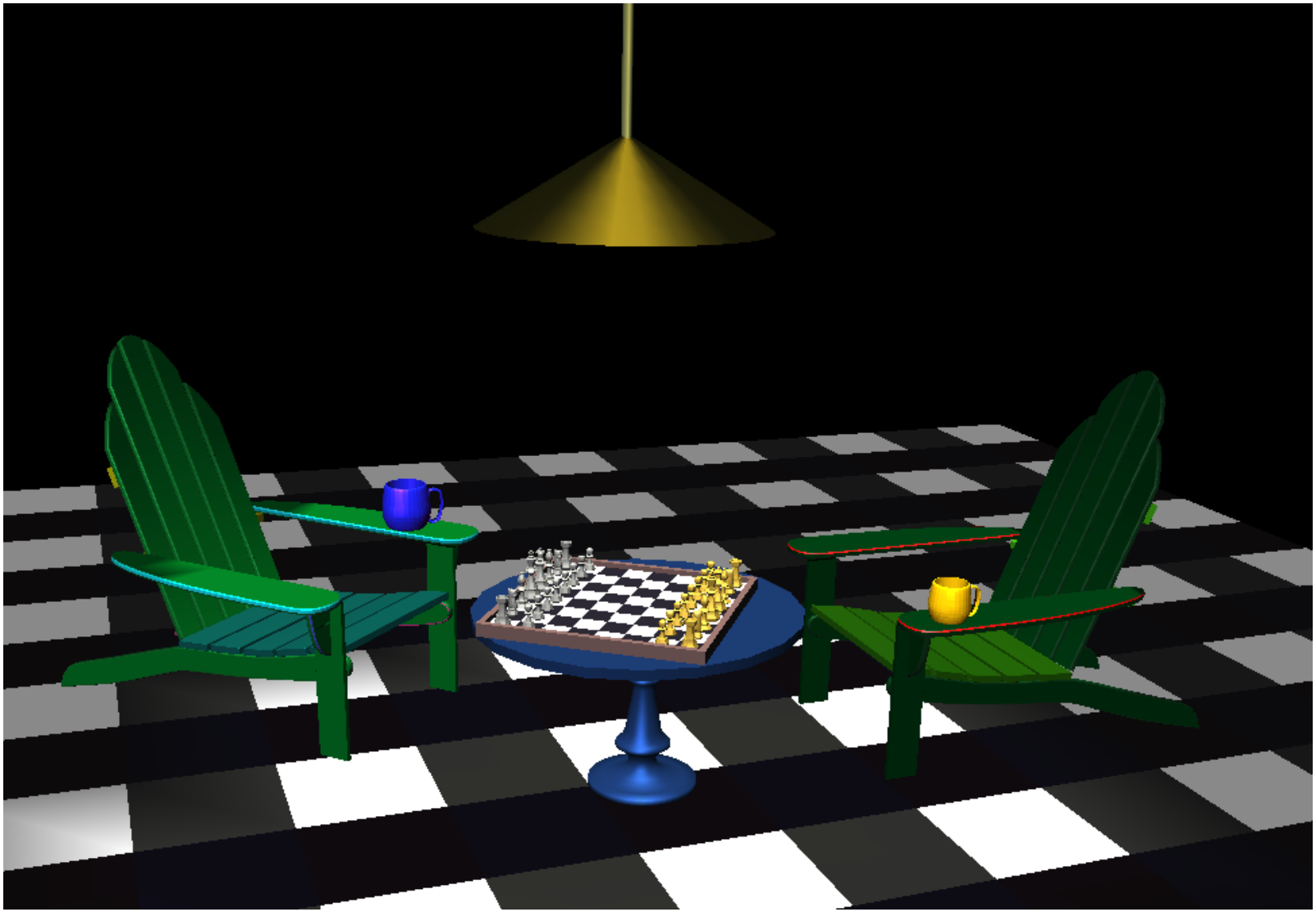
- Em muitas ocasiões (especialmente ao percorrer a base de dados da cena), pretende-se preservar a transformação corrente para uso posterior (Ver capítulo 9).
- Nas versões de OpenGL anteriores à versão 3.1, eram oferecidas várias pilhas de transformações para diferentes usos:
  - ModelView stack
  - Projection stack
  - Color stack
  - Texture stack
- A mesma funcionalidade pode ser facilmente recriada em Javascript usando objetos de tipo Array:

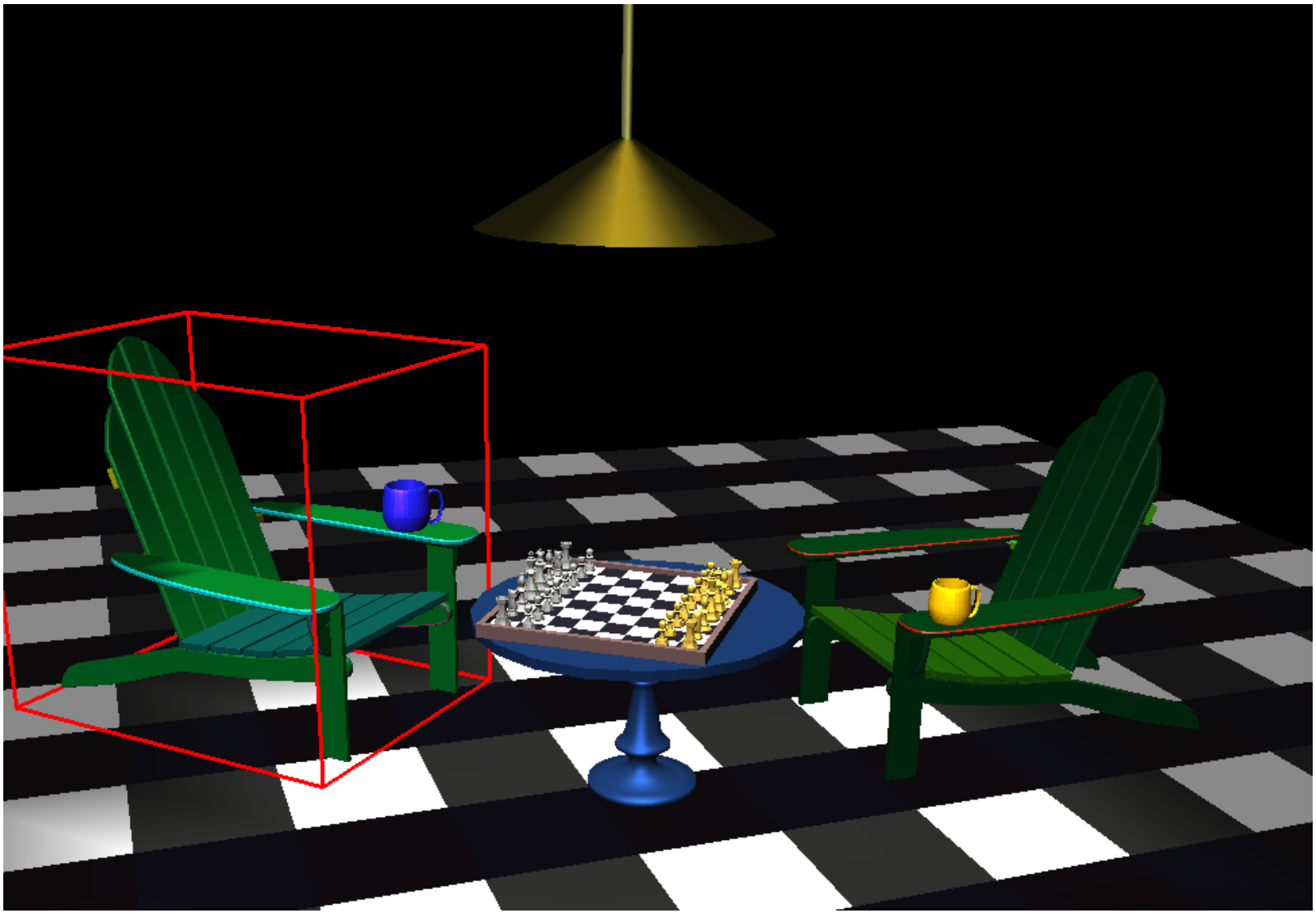
```
var stack = []
stack.push(modelViewMatrix);
...
modelViewMatrix = stack.pop()
```

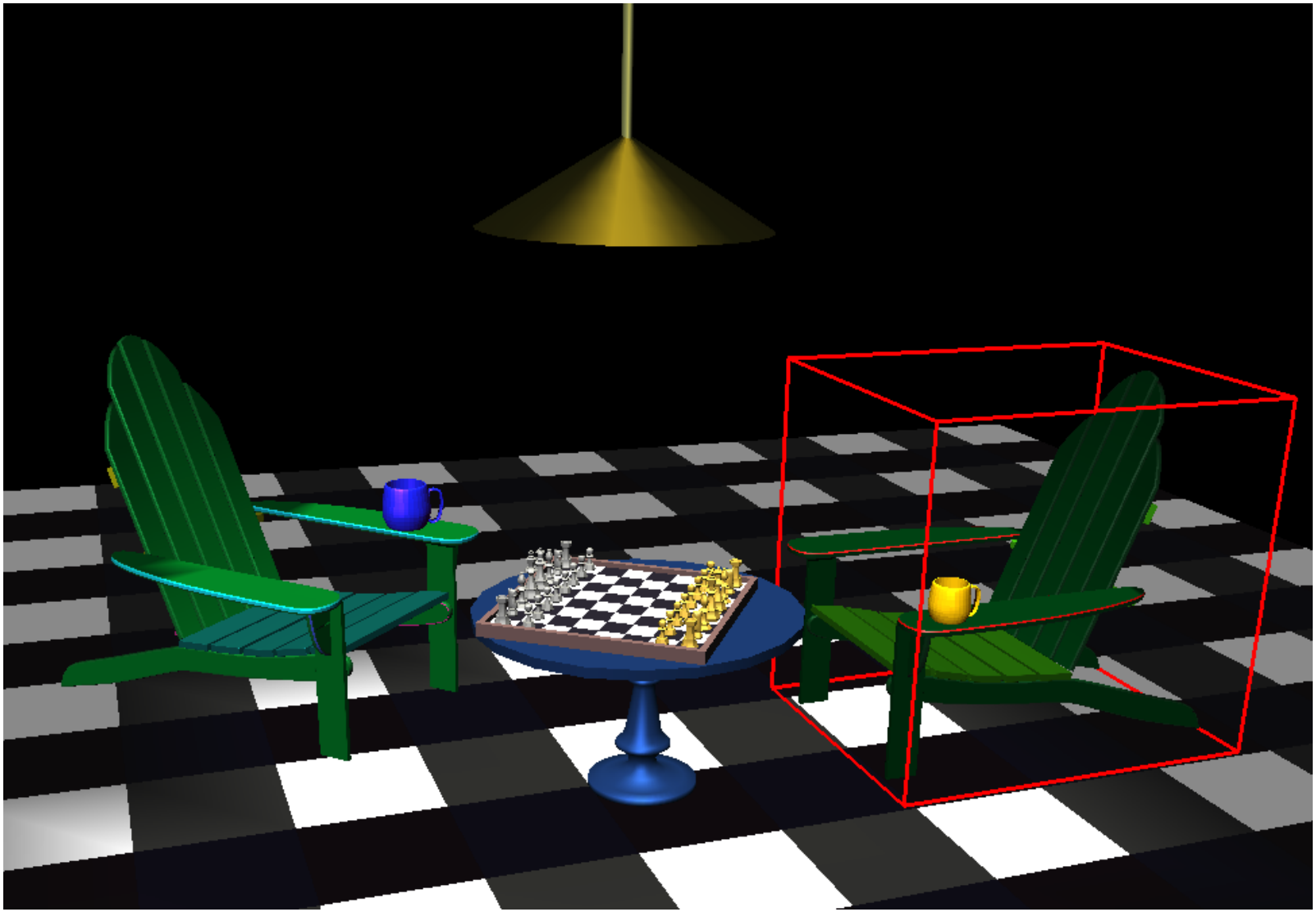
Modelação Hierárquica

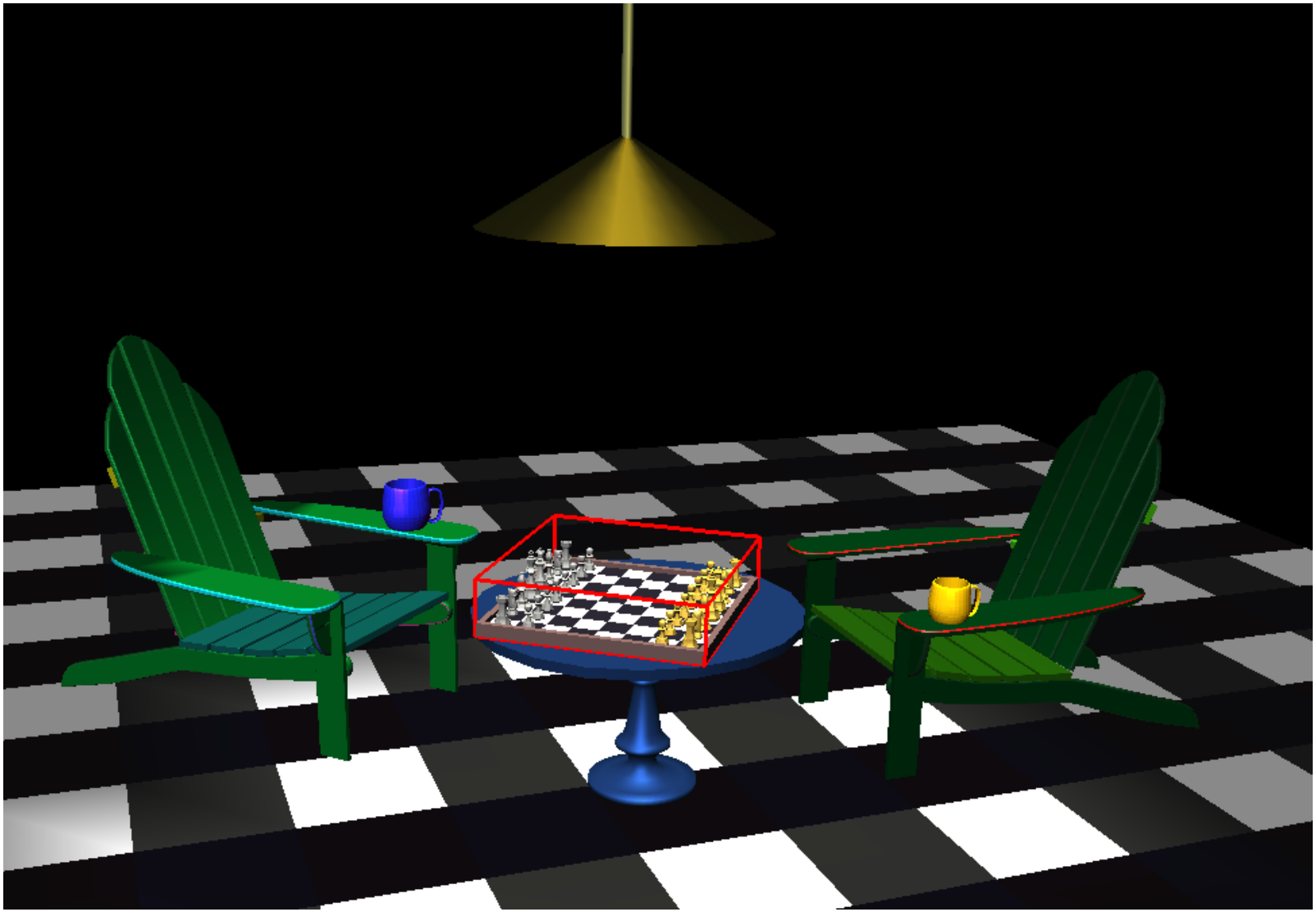
Objetivos

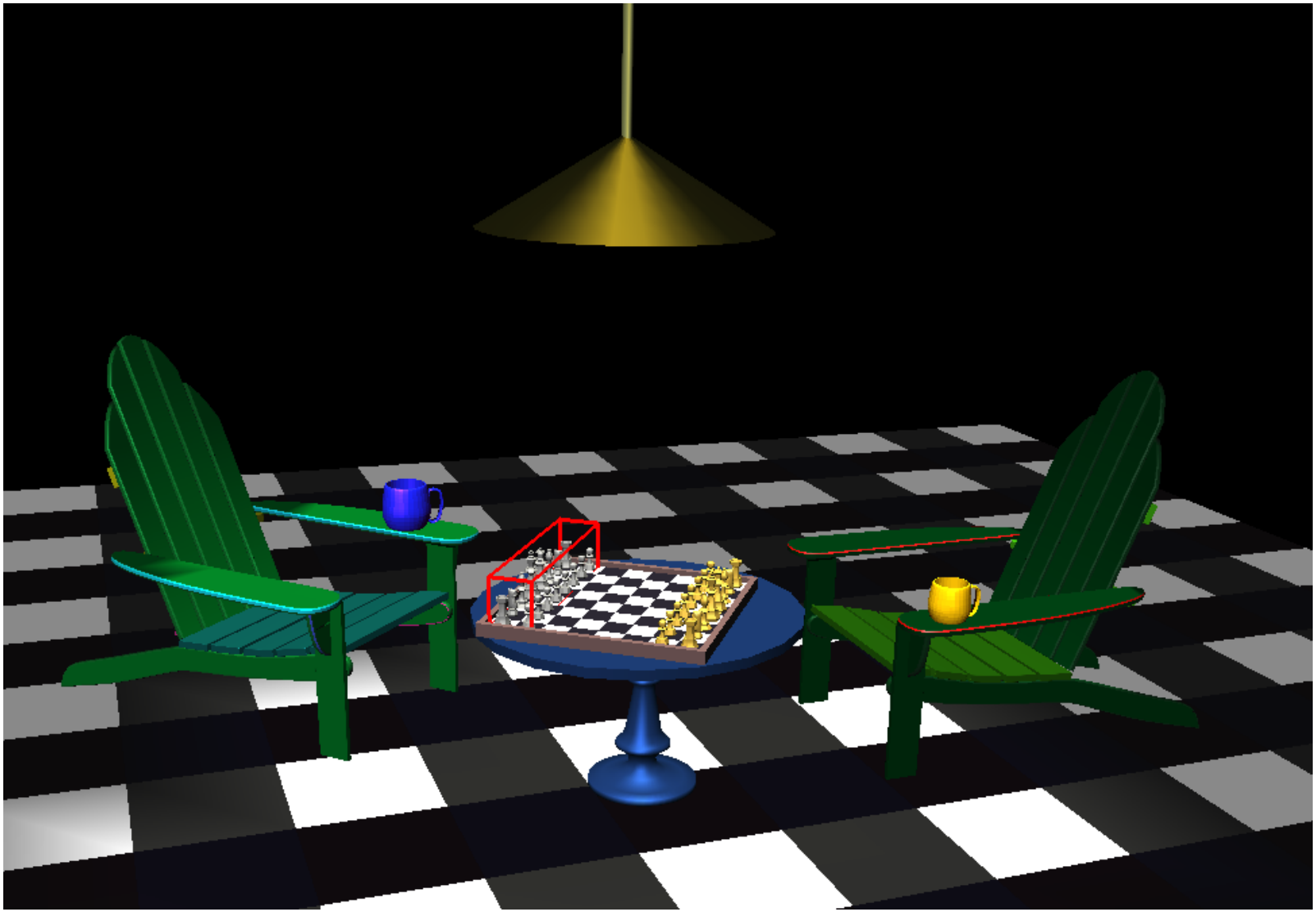
- Perceber as vantagens da modelação hierárquica de cenas
- Perceber a representação em forma de grafo da cena
- Saber escrever o código para a visualização duma cena descrita por um grafo e vice-versa.

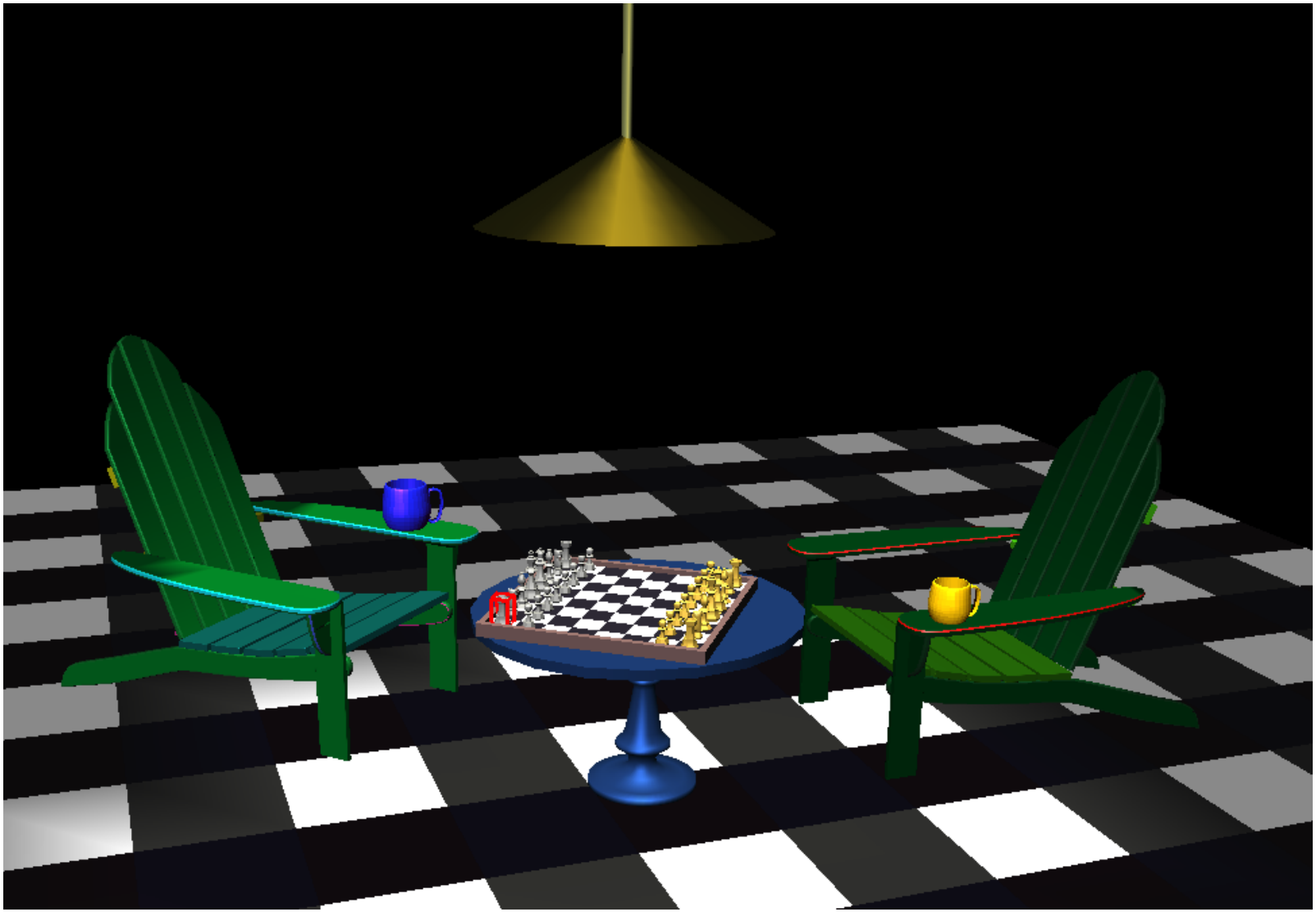






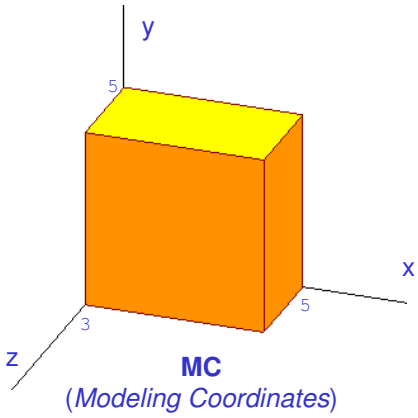


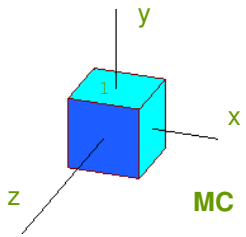


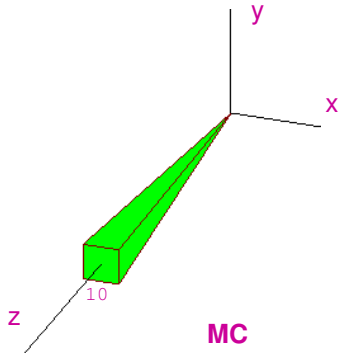


Sistemas de Coordenadas Locais, de Modelação ou do Objeto

- Cada objeto primitivo tem o seu próprio referencial, no qual foi feita a sua modelação.

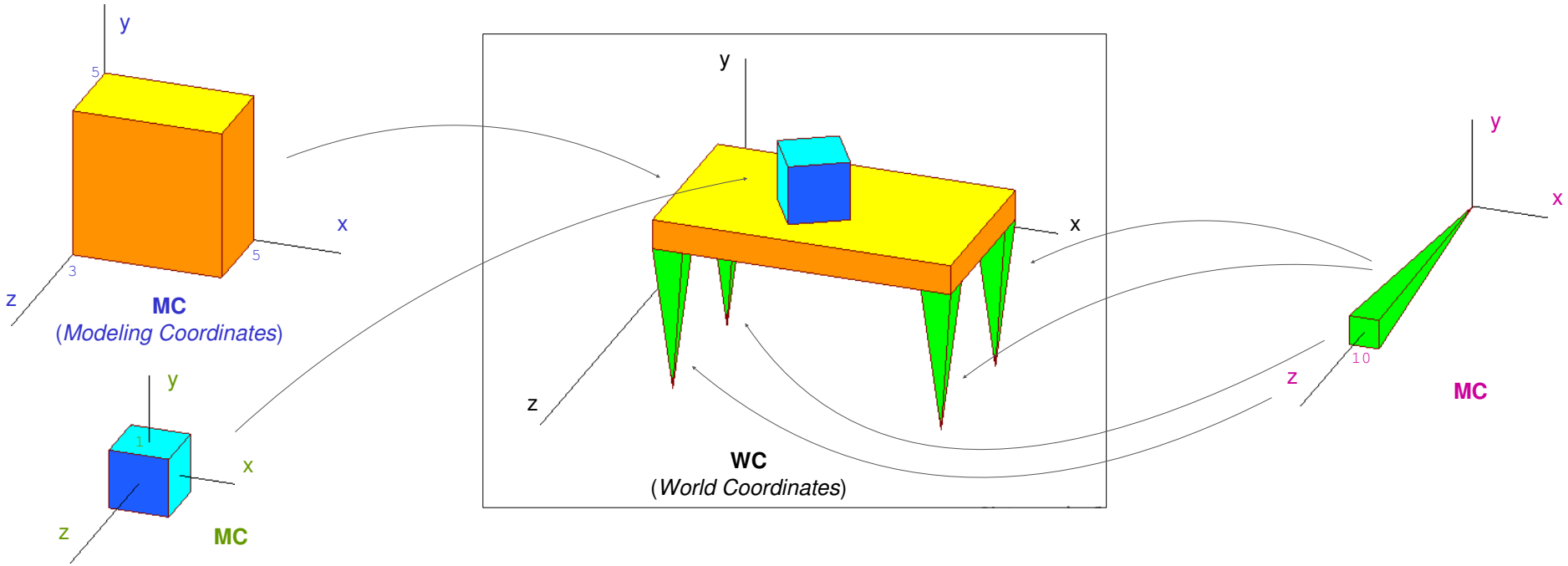






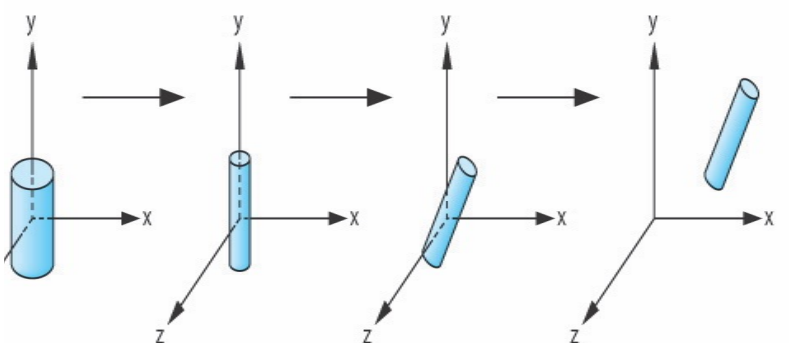
Sistema de Coordenadas do Mundo (Globais)

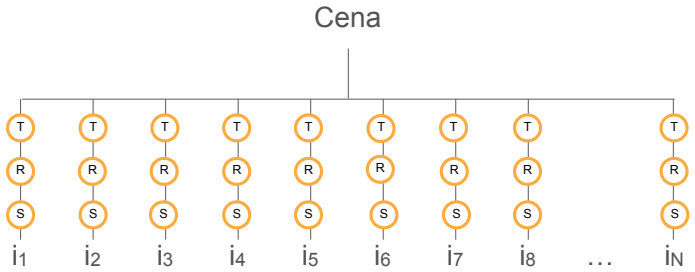
- A modelação duma cena consiste em aplicar aos objetos primitivos um conjunto de transformações geométricas para os instanciar na cena



Composição de Transformações

- Uma solução consiste em pensar isoladamente em cada instância de objeto primitivo e determinar isoladamente, uma composição de transformações do tipo: T.R.S



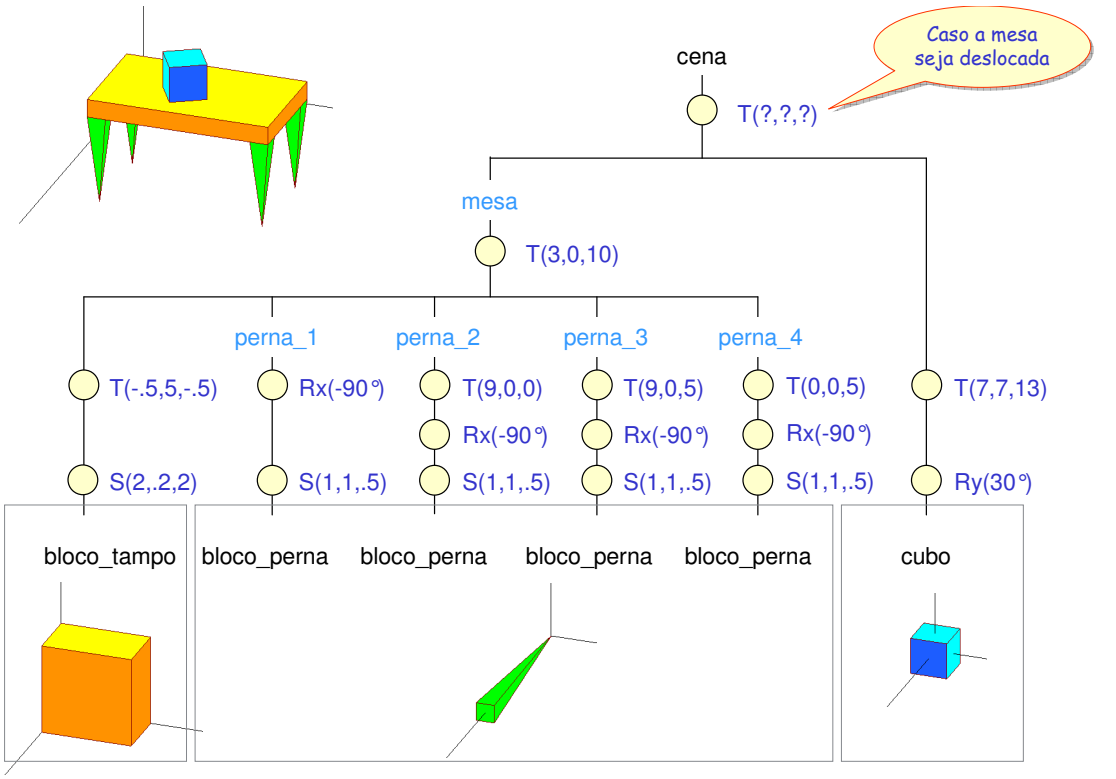


Como cada instância é pensada de forma isolada, não é fácil modificar seletivamente a cena. Exemplo: mudar o tamanho ou a posição da mesa do slide anterior.

Composição de Transformações

- A alternativa consiste em modelar a cena de forma hierárquica. Agrupando instâncias de objetos primitivos, eventualmente já transformados, para assim formar objetos mais complexos.
- Estes objetos complexos poderão ser também eles transformados e agregados a outros.
- No final, ter-se-á apenas um conjunto final, representando toda a cena.

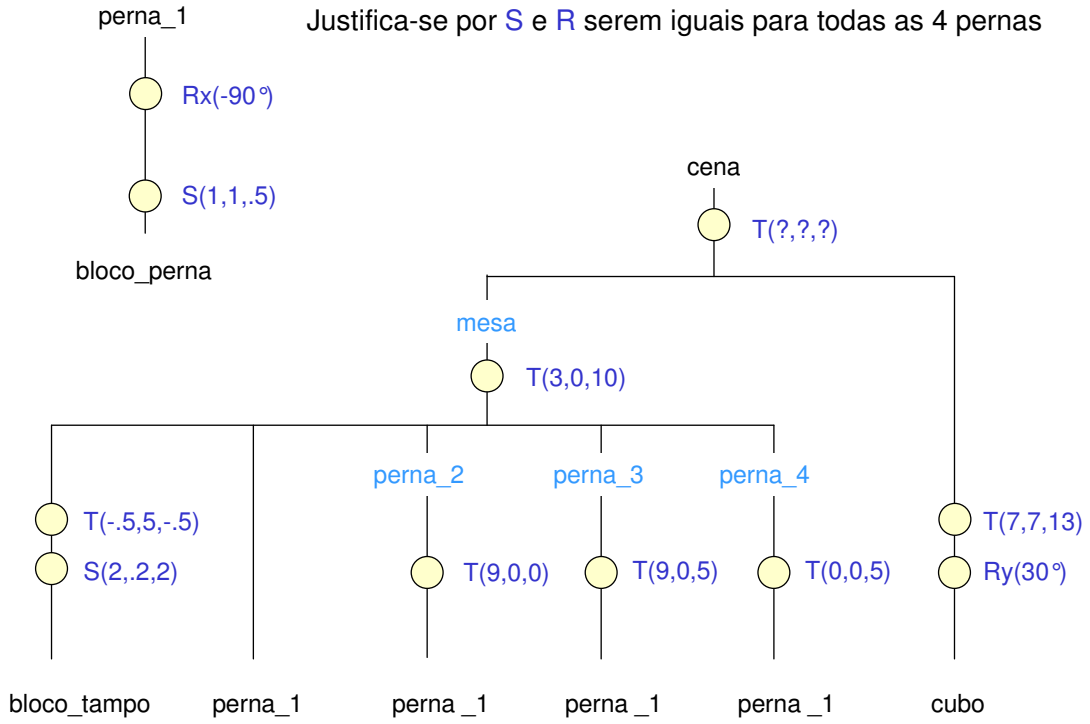
Grafo da Cena



M.Próspero

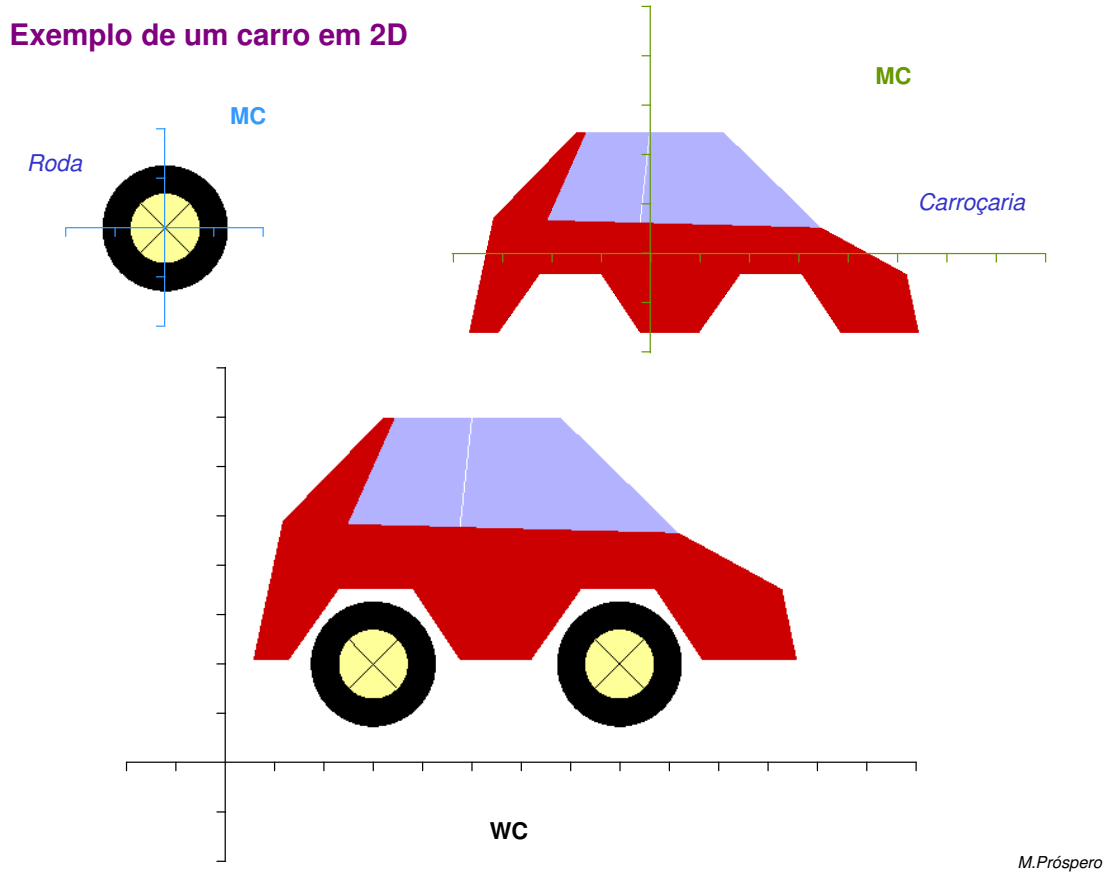
Grafo da Cena

Uma alternativa possível, com base na análise do modelo, usando-se um subgrafo auxiliar:



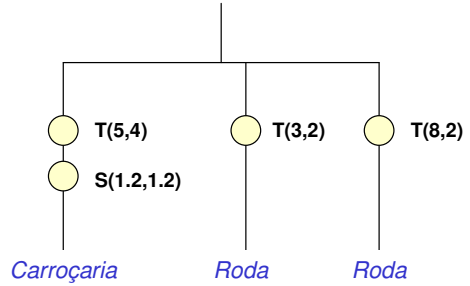
M.Próspero

Exemplo - Carro

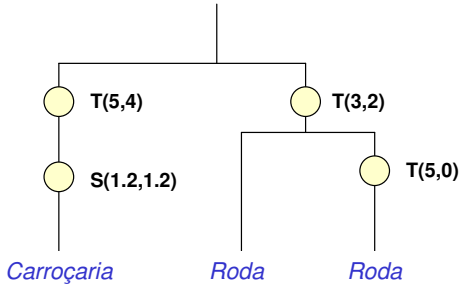


Exemplo - Carro

GRAFO DA CENA



Solução alternativa:

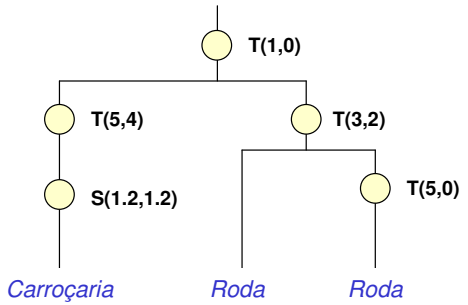


Como fazer avançar o automóvel de 1 unidade?

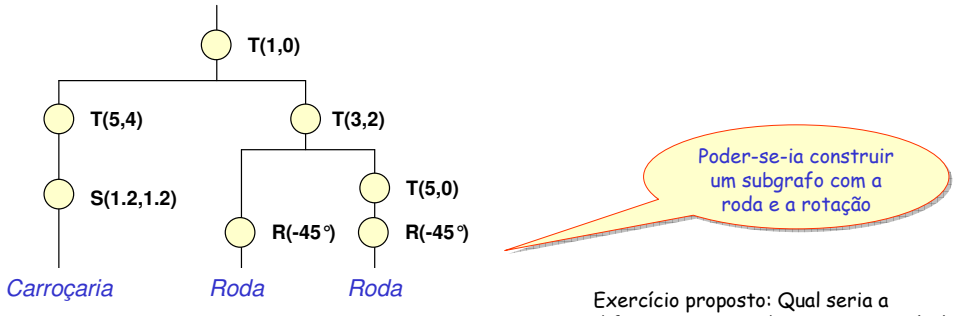
M.Próspero

Exemplo - Carro

GRAFO DA CENA



E como fazer girar as rodas, de acordo com esse movimento?

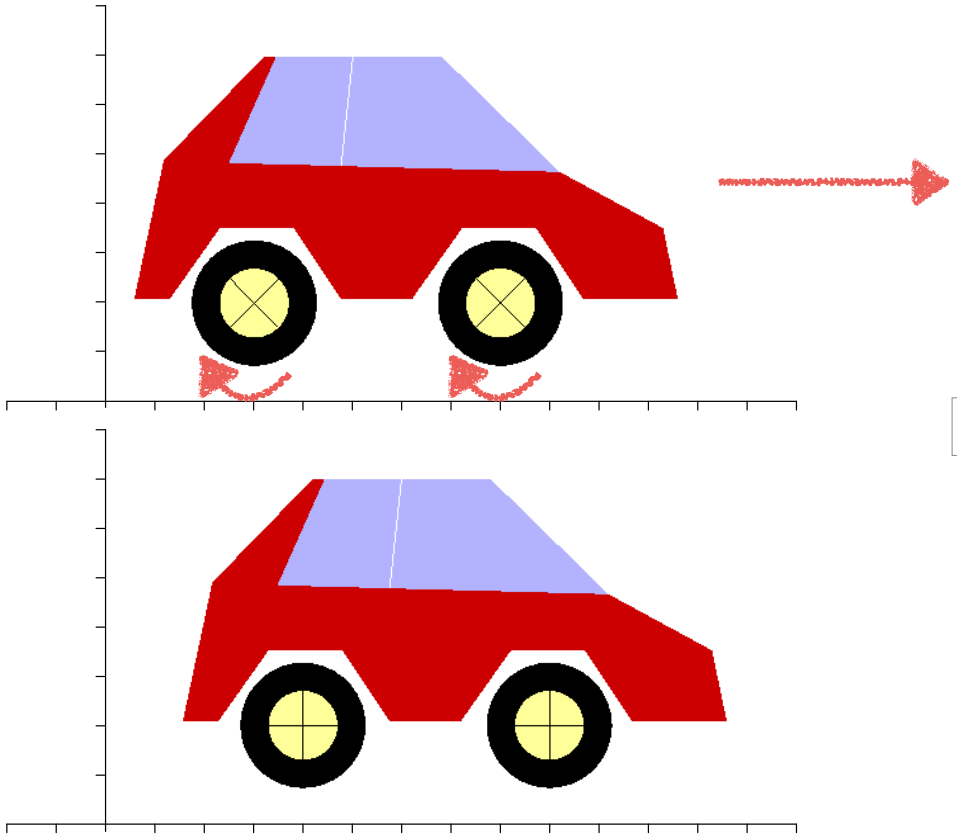


Poder-se-ia construir um subgrafo com a roda e a rotação

Exercício proposto: Qual seria a diferença no caso de um trator agrícola?

M.Próspero

Exemplo - Carro

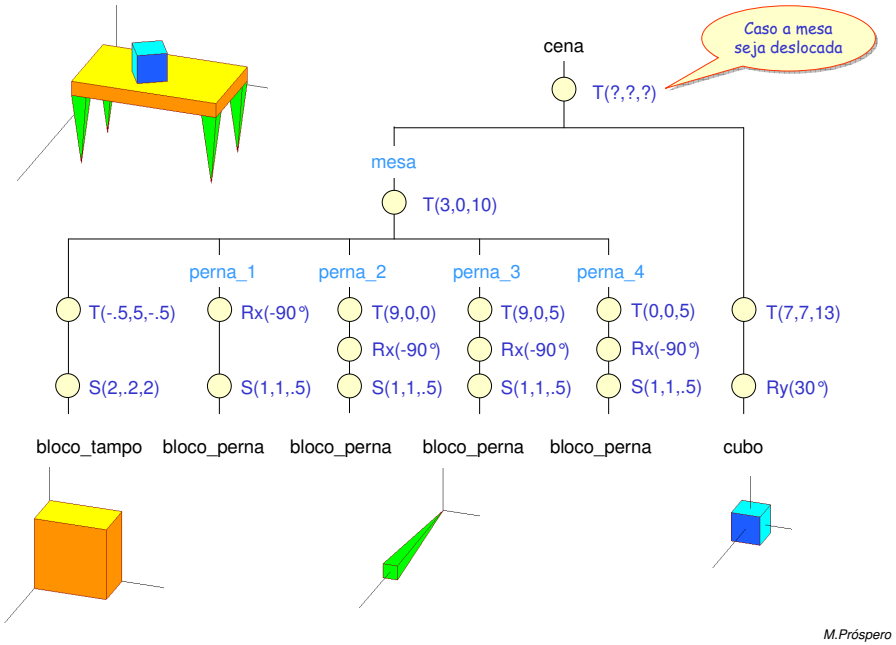


com rotação das rodas

M.Próspero

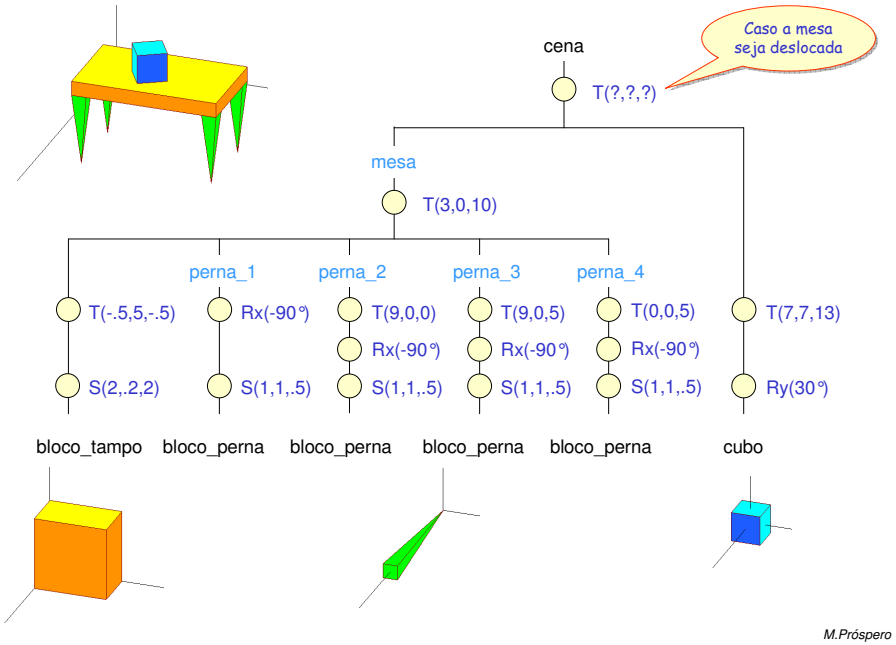
Pilha de transformações

- As transformações que afetam um determinado ramo do grafo (uma primitiva) são aplicadas à respetiva primitiva pela ordem obtida lendo as transformações de baixo para cima
- Em cada ramificação, todas as transformações desde a raiz têm efeito sobre todos os ramos descendentes desse ponto.
- Seja M a composição de transformações M_1, M_2, …, M_n, encontradas num percurso descendente, desde a raiz até uma primitiva, então $M=M_1.M_2…M_n$ e, para um ponto P, dessa primitiva, o ponto transformado será $M.P = M_1.M_2…M_n.P$



Pilha de transformações

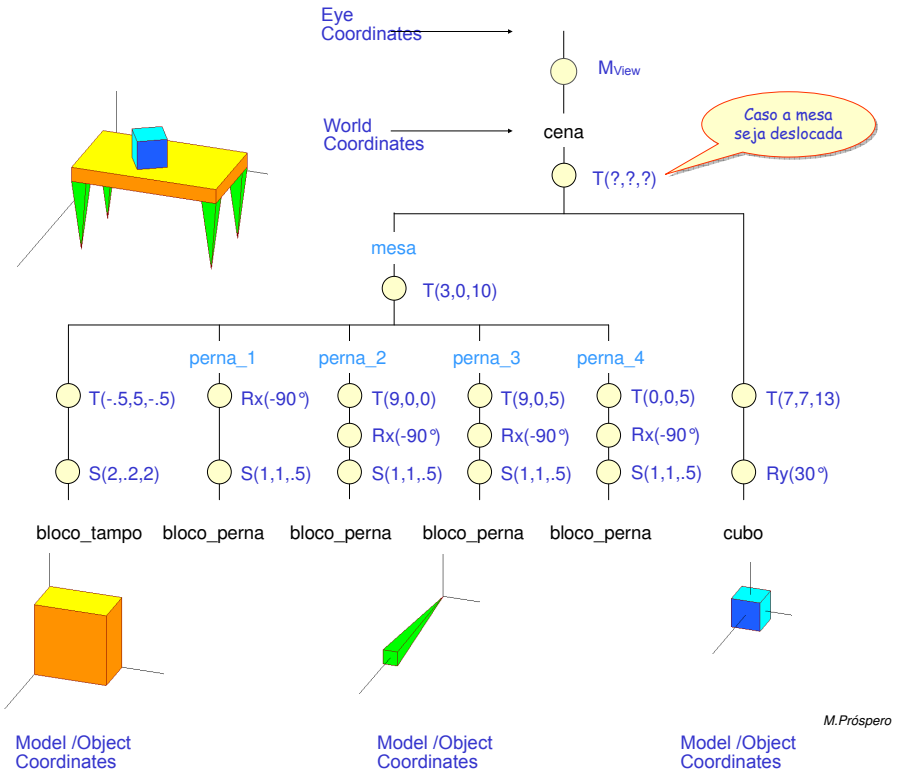
- A geração do código poderá ser feita através dum percurso prefixado no grafo, acumulando na matriz **Model***, por multiplicação à direita, cada uma das transformações que se vão encontrando.
- Nos pontos de bifurcação, poderemos preservar o valor atual da matriz **Model** empilhando-a numa pilha de transformações
- Num determinado nível do grafo, ao regressar-se dum ramo, recupera-se o valor anterior da matriz **Model** desempilhando-o da pilha, antes de prosseguir para um ramo irmão.



* Matriz com a transformação de modelação da instância corrente

Pilha de transformações

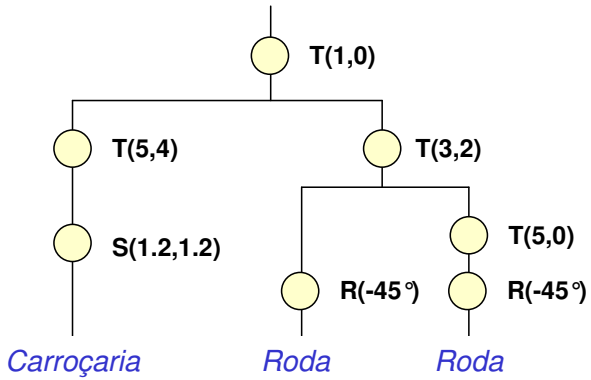
- Na prática, em vez de manipularmos a matriz **Model** (de modelação da instância corrente), manipula-se a matriz **ModelView** (**View** * **Model**).
- A Matriz **View** transforma de coordenadas do Mundo (World Coordinates) para coordenadas da câmara ou do olho (Eye Coordinates)



* Matriz com a transformação de modelação da instância corrente

Tradução para pseudo código (direta)

MV representa a matriz **ModelView** corrente. Presume-se que cada invocação duma primitiva usa a matriz MV para transformar os vértices.



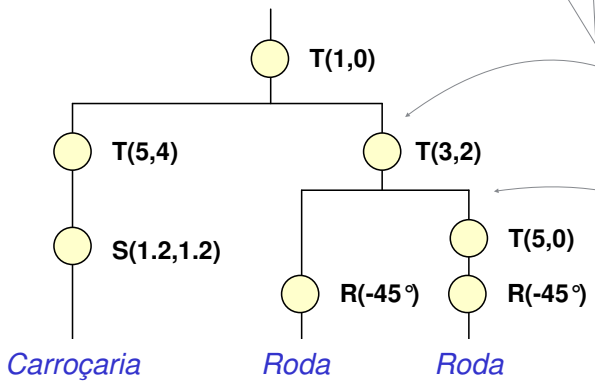
(Travessia prefixada do grafo)

A inicialização de MV é geralmente efetuada chamando a função **lookAt** que posiciona e orienta a câmara no mundo (Devolve a matriz **View**)

```
MV = I
MV = MV.T(1,0)
Push(MV)
    MV = MV.T(5,4)
    MV = MV.S(1.2, 1.2)
    Carroçaria
MV = Pop()
Push(MV)
    MV = MV.T(3,2)
    Push(MV)
        MV = MV.R(-45)
        Roda
    MV = Pop()
    Push(MV)
        MV = MV.T(5,0)
        MV = MV.R(-45)
        Roda
    MV = Pop()
MV = Pop()
```

Tradução para pseudo código (otimizada)

Não há necessidade de preservar a composição de transformações atual, contida na matriz **ModelView**, sempre que se vai descer pelo ramo mais à direita!

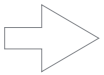


(Travessia prefixada do grafo)

```
MV = I
MV = MV.T(1,0)
Push(MV)
    MV = MV.T(5,4)
    MV = MV.S(1.2, 1.2)
    Carroçaria
MV = Pop()
Push(MV)
    MV = MV.T(3,2)
    Push(MV)
        MV = MV.R(-45)
        Roda
    MV = Pop()
    Push(MV)
        MV = MV.T(5,0)
        MV = MV.R(-45)
        Roda
    MV = Pop()
MV = Pop()
```

Tradução para código WebGL (com stack.js)

```
MV = I
MV = MV.T(1,0)
Push(MV)
    MV = MV.T(5,4)
    MV = MV.S(1.2, 1.2)
    Carroçaria
MV = Pop()
Push(MV)
    MV = MV.T(3,2)
    Push(MV)
        MV = MV.R(-45)
        Roda
    MV = Pop()
    Push(MV)
        MV = MV.T(5,0)
        MV = MV.R(-45)
        Roda
    MV = Pop()
MV = Pop()
```



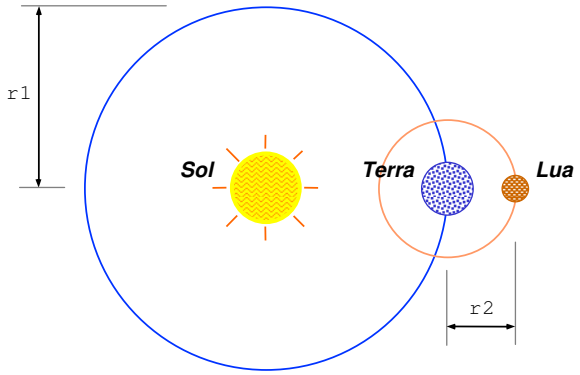
```
loadIdentity();
multTranslation([1,0,0]);
pushMatrix();
  multTranslation([5,4,0]);
  multScale([1.2,1.2,1]);
  Carroçaria();
popMatrix();

multTranslation([3,2,0]));
pushMatrix();
  multRotationZ(-45);
  Roda();
popMatrix();

multTranslation([5,0,0]);
multRotationZ(-45));
Roda();
```

Nota: As primitivas, aqui representadas por funções, terão que enviar ao vertex shader o valor atual da matriz modelView, para que os respetivos pontos que as compõem possam ser transformados corretamente.

Código da biblioteca stack.js

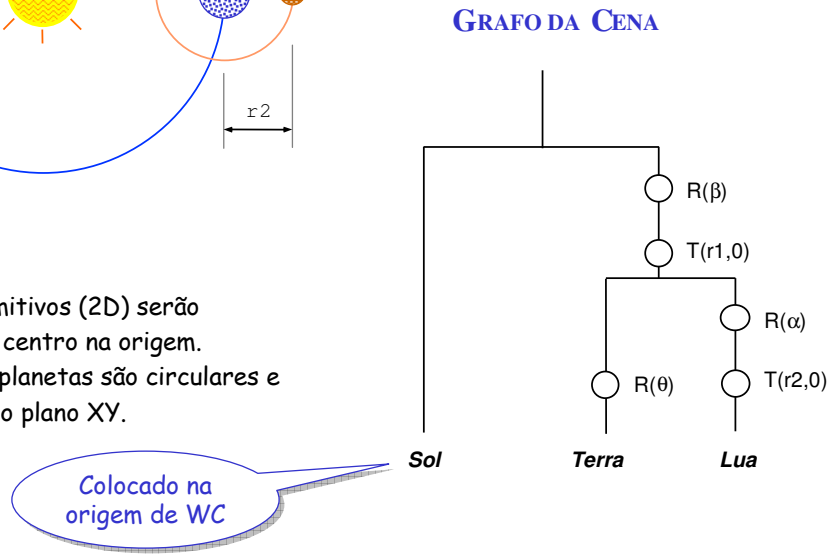
```
/**
 * This mimics the stack implementations of
OpenGL.
 * The model-view matrix is the matrix on
the top of the stack
 */
const stack = [ mat4() ];

// Replaces the current modelView (at the
top of the stack) with an identity matrix
function loadIdentity()
{
    stack[stack.length-1] = mat4();
}

// Replaces the current modelView (at the
top of the stack) with a given matrix
function loadMatrix(m)
{
    stack[stack.length-1] = mat4(m[0],
m[1], m[2], m[3]);
}

// The modelView matrix is the matrix on
the top of the stack.
// This function returns a copy of it.
function modelView()
{
    let m = stack[stack.length-1];
    return mat4(m[0], m[1], m[2], m[3]);
}

// Stack related operations

// Push a copy of the current model view
matrix
function pushMatrix() {
    const mv = stack[stack.length-1];
    var m = mat4(mv[0], mv[1], mv[2],
mv[3]);
    stack.push(m);
}

// Removes the matrix at the top of the
stack
function popMatrix() {
    stack.pop();
}

// Append transformations to modelView
function multMatrix(m) {
    stack[stack.length-1] =
mult(stack[stack.length-1], m);
}

// Append a translation to the modelView
function multTranslation(t) {
    stack[stack.length-1] =
mult(stack[stack.length-1], translate(t));
}

// Append a scale to the modelView
function multScale(s) {
    stack[stack.length-1] =
mult(stack[stack.length-1], scalem(s));
}

// Appends a rotation around X to the
modelView
function multRotationX(angle) {
    stack[stack.length-1] =
mult(stack[stack.length-1],
rotateX(angle));
}

// Appends a rotation around Y to the
modelView
function multRotationY(angle) {
    stack[stack.length-1] =
mult(stack[stack.length-1],
rotateY(angle));
}

// Appends a rotation around Z to the
modelView
function multRotationZ(angle) {
    stack[stack.length-1] =
mult(stack[stack.length-1],
rotateZ(angle));
}
```

Exemplo - Sistema Planetário Simplificado



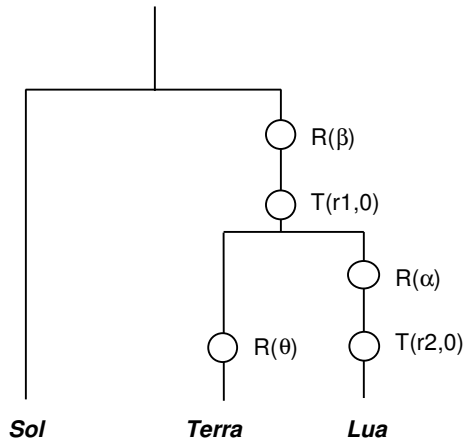
Admite-se que:

- Os objetos primitivos (2D) serão modelados com centro na origem.
- As órbitas dos planetas são circulares e encontram-se no plano XY.



M.Próspero

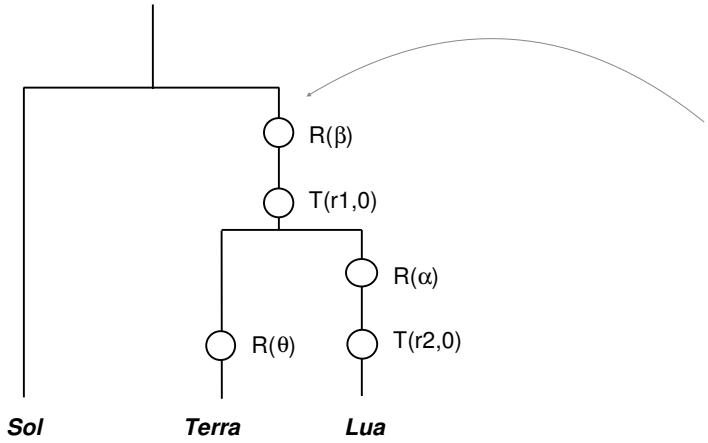
Tradução direta



Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
      popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
    popMatrix();
popMatrix();
```

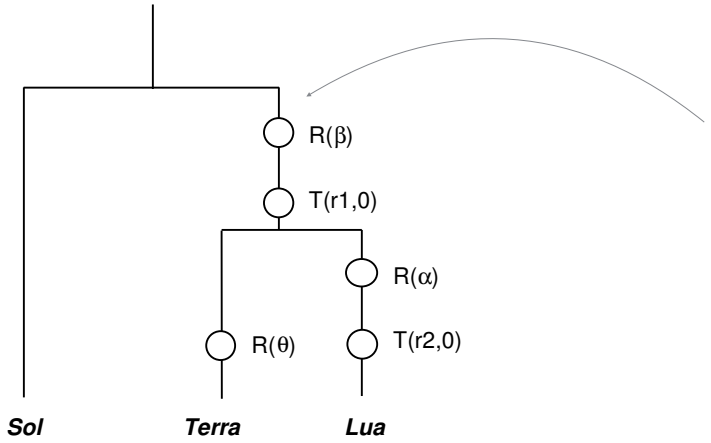
Tradução otimizada (I)




```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
      popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
    popMatrix();
popMatrix();
```

Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

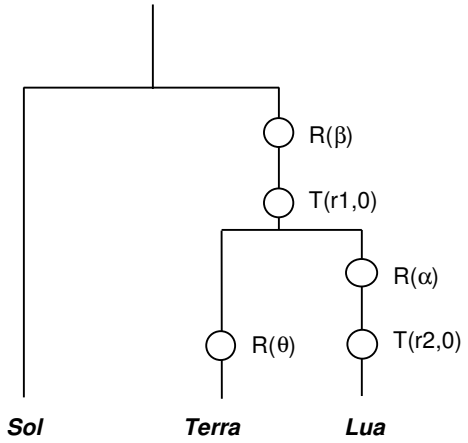
Tradução otimizada (I)




```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
      popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
    popMatrix();
popMatrix();
```

Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

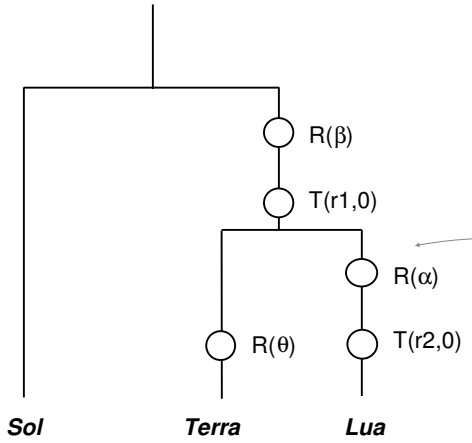
Tradução otimizada (I)



Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
      popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
    popMatrix();
popMatrix();
```

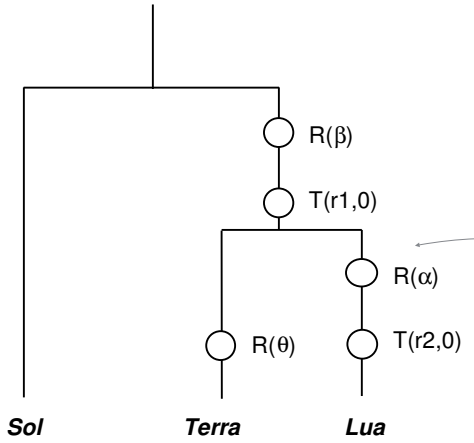
Tradução otimizada (II)




```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
      popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
    popMatrix();
popMatrix();
```

Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

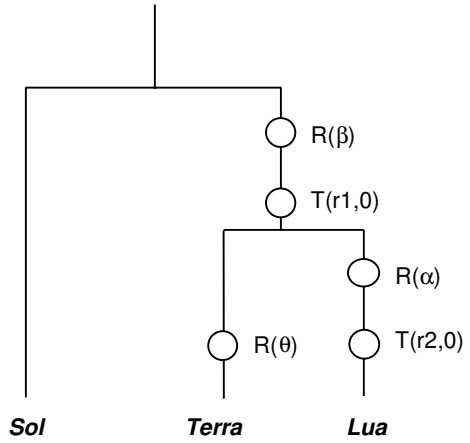
Tradução otimizada (II)




```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
      popMatrix();
    popMatrix();
   pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
   popMatrix();
popMatrix();
```

Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

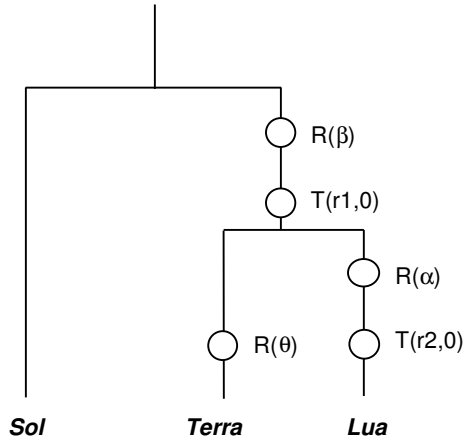
Tradução otimizada (II)



Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
      popMatrix();
    popMatrix();
   pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
   popMatrix();
popMatrix();
```

Tradução otimizada (III)

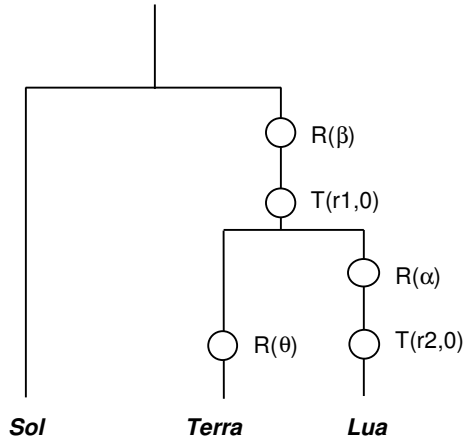


Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
      popMatrix();
    popMatrix();
   pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
   popMatrix();
popMatrix();
```

Inútil, devido ao PopMatrix() imediatamente a seguir!

Tradução otimizada (III)

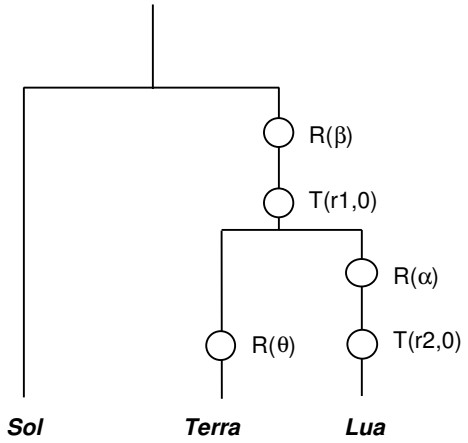


Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
      Terra();
    popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
    popMatrix();
popMatrix();
```

Inútil, devido ao PopMatrix() imediatamente a seguir!

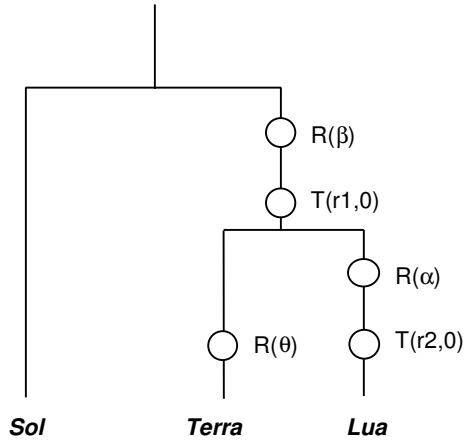
Tradução otimizada (III)



Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
    popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
    popMatrix();
popMatrix();
```

Tradução otimizada (IV)

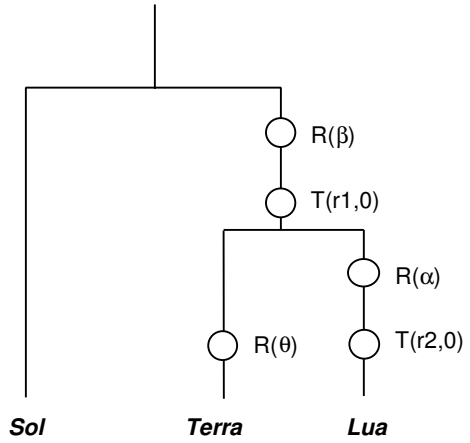


Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
      pushMatrix();
        Lua();
      popMatrix();
    popMatrix();
popMatrix();
```

Inútil, pois nada mais há para desenhar!

Tradução otimizada (IV)

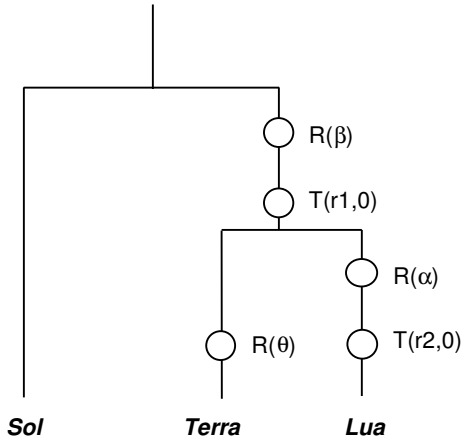


Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
    popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
    pushMatrix();
       Lua();
    popMatrix();
    popMatrix();
popMatrix();
```

Inútil, pois nada mais há para desenhar!

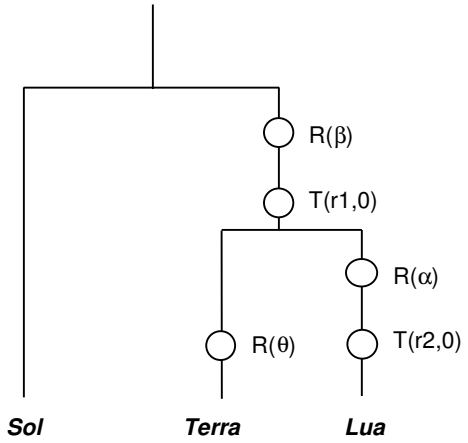
Tradução otimizada (IV)



Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
    popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
    pushMatrix();
       Lua();
    popMatrix();
    popMatrix();
popMatrix();
```

Tradução otimizada (V)

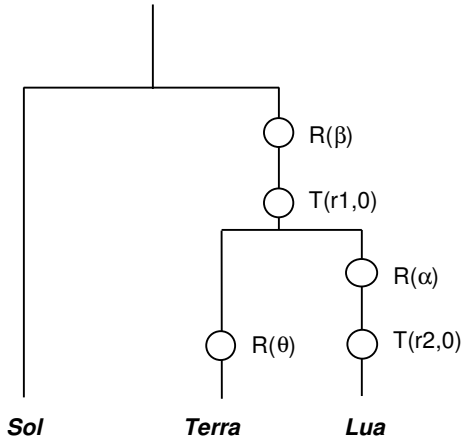


Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
       Terra();
    popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
    pushMatrix();
      Lua();
    popMatrix();
    popMatrix();
popMatrix();
```

Inútil, devido ao PopMatrix() imediatamente a seguir!

Tradução otimizada (V)

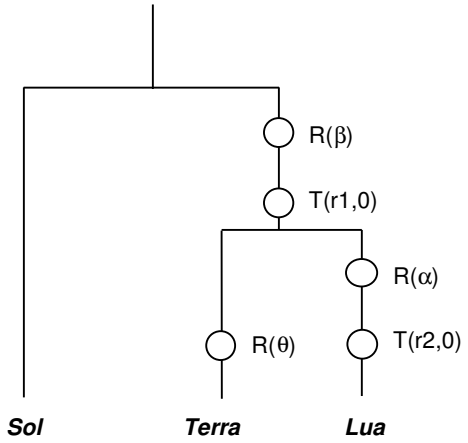


Esta solução foi escrita supondo que **nada** se conhece sobre a programação dos objetos primitivos, não sendo de excluir que aí possam ter sido utilizadas **transformações geométricas** (ver as instruções Push e Pop como parênteses de cada um desses objetos).

```
pushMatrix();
  pushMatrix();
    Sol();
  popMatrix();
popMatrix();
pushMatrix()
  multRotationZ(beta);
  multTranslation([r1,0,0]);
  pushMatrix();
    multRotationZ(theta);
    pushMatrix();
      Terra();
    popMatrix();
    popMatrix();
    pushMatrix();
      multRotationZ(alpha);
      multTranslation([r2,0,0]);
    pushMatrix();
      Lua();
    popMatrix();
  popMatrix();
popMatrix();
```

Inútil, devido ao PopMatrix() imediatamente a seguir!

Tradução otimizada




```
pushMatrix();
  Sol();
popMatrix();
multRotationZ(beta);
multTranslation([r1,0,0]);
pushMatrix();
  multRotationZ(theta);
    Terra();
popMatrix();
multRotationZ(alpha);
multTranslation([r2,0,0]);
Lua();
```