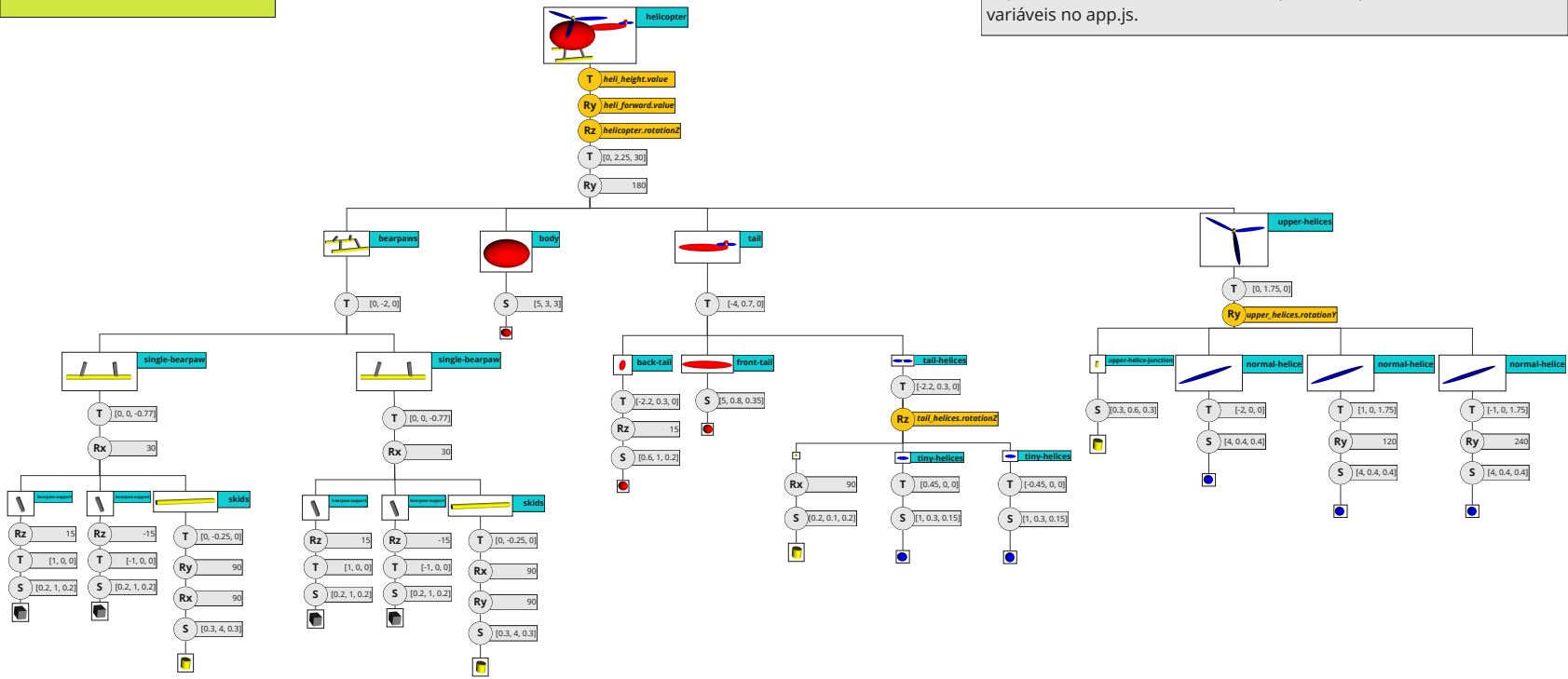
Projeto CG1-2

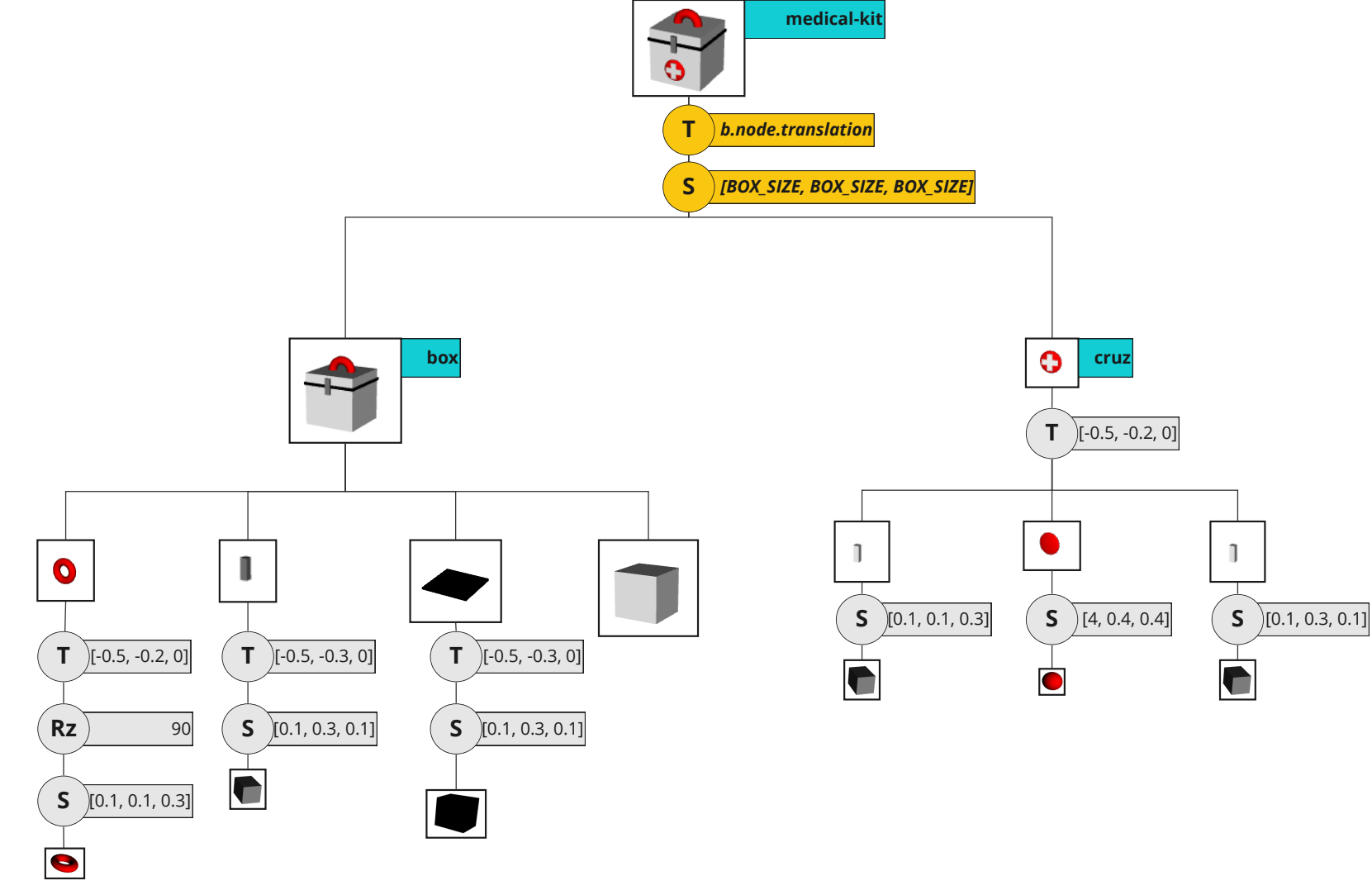
By:
Iago Paulo 60198
James Furtado 61177

Grafo de cena

Informações:

- Transformações em amarelo são os valores das transformações que dependem dos valores de variáveis que correspondem a nome de variáveis no app.js.




medical-kit
T
b.node.translation
S
[BOX_SIZE, BOX_SIZE, BOX_SIZE]
box
cruz
T
[-0.5, -0.2, 0]
T
[-0.5, -0.2, 0]
Rz
90
S
[0.1, 0.1, 0.3]
T
[-0.5, -0.3, 0]
S
[0.1, 0.3, 0.1]
T
[-0.5, -0.3, 0]
S
[0.1, 0.3, 0.1]
S
[0.1, 0.1, 0.3]
S
[4, 0.4, 0.4]
S
[0.1, 0.3, 0.1]

Manual do utilizador

- W - Modo com linhas
- S - Modo com faces solidas
- 0 - Modo perspetiva

Dentro do modo perspetiva

  - Botão direito do rato - Free cam em volta do prédio
  - Scroll do rato - Zoom
- 1 - Modo ortogonal
- 2 - Visão de frente
- 3 - Visão de cima
- 4 - Visão pela direita
- 5 - Modo terceira pessoa do helicóptero (em perspetiva)

Sobre o Helicóptero:

- Seta para cima - Helicóptero voa para cima
- Seta para baixo - Helicóptero voa para baixo
- Seta para a esquerda - Helicóptero começa a andar em torno do prédio
- Espaço - Larga a caixa (kit medico)

Nova biblioteca

Com o objetivo de fazer o processo de descrição e desenho do grafo de cena mais fácil e conveniente, nós decidimos fazer uma pequena biblioteca que define algumas classes e uma maneira de descrever o grafo de cena em um ficheiro **JSON**.

A biblioteca encontra-se no ficheiro sg-builder.js (acrónimo para *Scene Graph Builder*). Ela define um grafo em que existem dois tipos de nós: os regulares (ou nós não terminais) e os leafs (os nós terminais). Ambos os nós podem ter transformações (rotações, escalas e translações). A única diferença entre os dois é que os nós regulares, além das transformações, podem ter filhos (que podem ser tanto leafs como regulares) e os nós leafs podem ter uma cor e uma primitiva (que corresponde a um dos oferecidos pelos professores).

Sintaxe do JSON

A biblioteca define ainda uma classe chamada SceneGraph que representa o grafo de cena que nós queremos descrever. Essa classe aceita no construtor o scene_desc que é um objecto javascript (onde a nossa cena esta descrita) e uma array de strings (que representam as primitivas que nós suportamos).

O scene_desc tem que seguir a seguinte sintaxe:

```
{
        "root": ...,
        "base-nodes": ...
}
```

Onde o root é o nó raiz do nosso grafo de cena e o base-nodes é um dicionário de nós onde as chaves desse dicionário podem ser usadas como referencias a esses nós nos lugares do nosso JSON em que se espera um nó.

A sintaxe dos nós é a seguinte:

```
// A sintaxe foi descrita em baixo formato typescript
// pare ser mais facil de entender. Segue-se abaixo uma breve explicação
// de typescript, assumindo que os professores não estão familiarizados com o tal.
// Caso contrario podem saltar a explicação.

// ------------------------------------------------------------------------
//        BREVE EXPLICAÇÃO DE TYPESCRIPT
// ------------------------------------------------------------------------
// O seguinte:
//
// type <name> = {
//  <key-1>: <type-1>
// }
//
// define que o <name> é um tipo objeto com a <key-1> que é do tipo <type-1>.
//
// Para indicar que a <key-1> poderia ser um tipo ou outro por exemplo
// uma string ou um numero usamos a seguinte sintax:
//
// <key-1>: string | number
//
// e pare indicar que algo tem que ser dois tipos ao mesmo tempo
// usamos o & por exemplo
// type BaseType = SuperType & OtherType signica o
// tipo BaseType é ao mesmo tempo SuperType e o OtherType
//
// A ultima a se saber é que em typescript quando se define um tipo
// que é um object pode-se incluir o ? a frente do attributo para
// indicar que ele é opcional. Por exemplo:
//
// type Person = {
//  name: string,
//  age?: number
// }
// O tipo Person define um objeto que tem que ter um atributo
// name e pode ter um atributo age.
// ------------------------------------------------------------------------
//        FIM DA BREVE EXPLICACÃO
// ------------------------------------------------------------------------
```

```
// O atributo "type" é o tipo da transformação
// que é um dos atributos do AbstractNode (excepto o "extra-trans" e o name)
// e o value tem que ser do tipo correspondente assim como descrito em baixo.
// POR EXEMPLO: se o type for "scale" o value tem que ser
// um array de 3 numeros, se for um "rotation-x" o value tem
// que ser um numero.

type transformation = {
        type: string,
        value: number | number[3]
}

type AbstractNode = {
        name?: string,
        "rotatio-x"?: number,
        "rotatio-y"?: number,
        "rotatio-z"?: number,
        scale?: number[3],
        translation?: number[3],

        // array de transformações ou uma transformação
        "extra-trans"?: transformation[] | transforation
}

// a string tem que ser uma key do "base-nodes"
type Node = RegularNode | LeafNode | string

type RegularNode = AbstractNode & {
        "children": Node[] | Node // um array de nós ou só um nó
}

// leaf node
type LeafNode = AbstractNode & {

        // sintexe do nó em cima mais:
        "primitive": string,

        // uma string que corresponde a uma das cores default que
        // biblioteca support ou um array de 3 números (rgb) ou um
        // array de 4 numeros (rgba)
        color?: string | number[3] | number[4]
}
```

Nota: lembrado que o tipo node do children pode ser um objecto javascript do leaf ou do regular node ou uma string que refere a um "base-nodes".

Relativamente as transformações dos nós é de se notar que além das keys (rotation-x, scale, translation) do nó também existem os "extra-trans". A diferença é que as transformações dos nós são aplicados seguindo a ordem TRS e depois são aplicadas as do "extra-trans" pela ordem que aparecem.

Principais Classes da Biblioteca

Em baixo está uma breve descrição das classes mais importantes:

```
// transformações que podem ser
// adicionados aos nós. Vale a pena realçar que
// se mudarmos os atributo value de uma transformação
// que está em um nó da proxima vez que o desenharmos
// notaremos o efeito da mudança
class Transformation{
        get value(){...}
        set value(newValue){...}

         // a matrix de transformação da transformação
        get transMatrix(){...}
}

class RotationX extends Transformation{...}
class RotationY extends Transformation{...}
class RotationZ extends Transformation{...}
```

```
class Scale extends Transformation{...}
class Translation extends Transformation{...}

class Node{

        // usadas para obter e definir
        // valores para as transformações principais
        // aquelas que são aplicadas primeiro e na ordem TRS
        get translation(){...}
        get rotationX(){...}
        get rotationY(){...}
        get rotationZ(){...}

        set translation(newTranslation){...}
        set rotationX(newRotationX){...}
        set rotationY(newRotationY){...}
        set rotationZ(newRotationZ){...}

         // a matrix que resulta na multiplicação das respetivas
         // matrizes de transformações
        get modelMatrix(){...}

        // usadas para adicionar ou remove transformações
        // aquelas definidas em cima. E elas funcionam assim
        // como as "extra-trans"
        addTransformation(trans){...}
        removeTransformation(trans){...}
}

class LeafNode extends Node{
        get primitive(){...}
        get color(){...}
        draw(stack, draw){...}
}

class RegularNode extends Node{
        // usadas para obter os filhos
        // ou remove-los
        get children(){...}
        getChild(name)(){...}
        searchNode(name)(){...}
        removeChild(child)(){...}

        // usadas para desenhar o grafo de cena
        draw(stack, draw){...}
}

class SceneGraph(){
        // usado para obter o root do grafo
        get root(){...}

        // usadas para obter nós do grafo
        getBaseNode(name){...}
        findNode(path){...}

        // usadas para criar um nó apartir de um
        // objeto javascript
        createNode(node_desc){...}
}
```

Para mais informações sobre as classes acima descritas deverão consultar o ficheiro sg-builder.js, lá encontrarão alguns comentários além do proprio codigo que vão ajudar a melhor compreenção da biblioteca.