Procedural Modeling Project

Jaime Manuel Trillos Ujueta

Bonn University – March 2018

Motivation

- The goal: Use the concepts and methods learned during the semester.
- Procedural modeling is a term which use the creation of 3D models and textures algorithmically (random) from sets of rules.
- The advantage: Helps developing complex scenes in less time and memory.

Task:

1. Define a procedural grammar suitable for describing a room (including windows, doors, ...).
2. Parse a grammar file of such a procedural description.
3. Define the data structure for grammar and geometry (tree structure).
4. Apply procedural rules, generate room geometry and store it into the data structure.
5. Pass the intermediate result to renderer.

Procedural Grammar for Describing a Room

- Definition of a rule:

```
Head=Rules operators:Probability
```

- Example:

```
main=S(20,20,20) Subdiv(0,20){house}
wall1=Subdiv(1,5,10,5){wallpaper,wallMiddle1,wallpaper}:0.6
wall1=Subdiv(1,20){wallpaper}:0.4
```

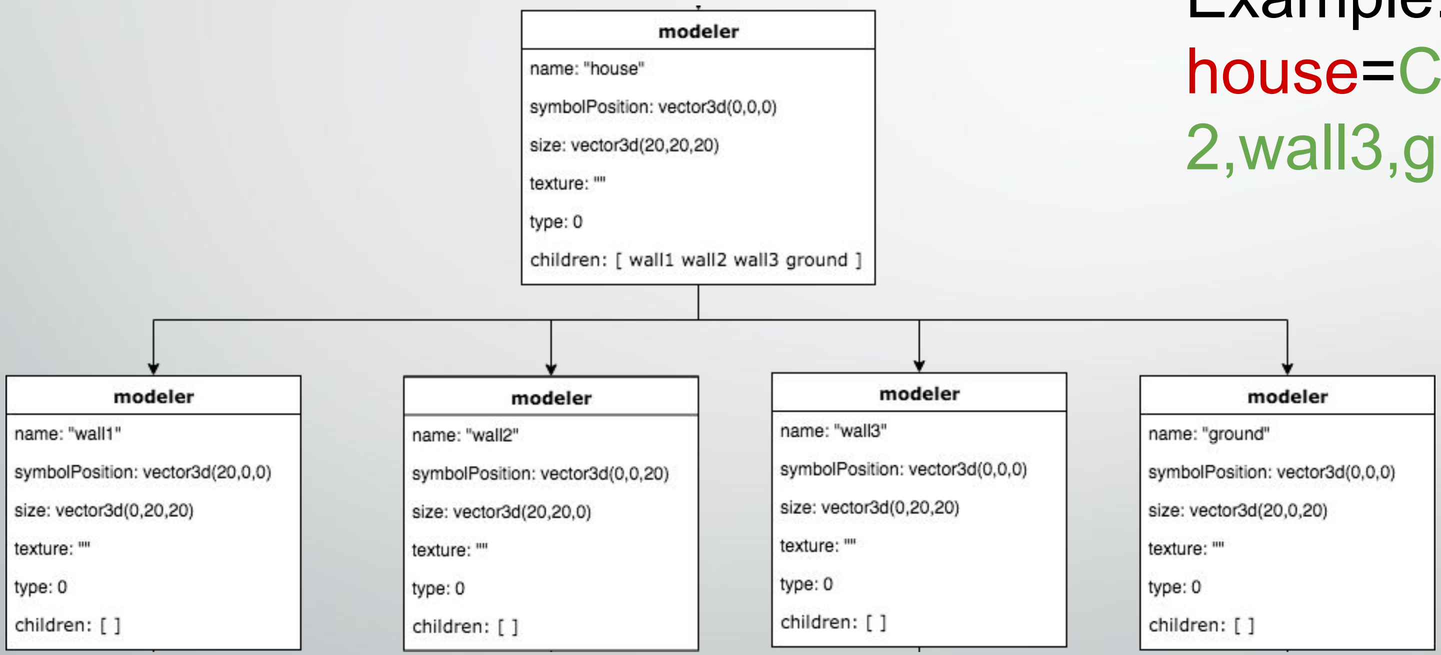
- The file can have comments using at the beginning #

Procedural Grammar for Describing a Room

- **Comp(type){parameters}** : Split scope into planes

The children obtain the position and the size from the father. It change depending of the side. {right, back, left, down, up, front}

Example:
house=Comp(sides){wall1,wall2,wall3,ground}



modeler
name: "house"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,20,20)
texture: ""
type: 0
children: [wall1 wall2 wall3 ground]

modeler
name: "wall1"
symbolPosition: vector3d(20,0,0)
size: vector3d(0,20,20)
texture: ""
type: 0
children: [ ]

modeler
name: "wall2"
symbolPosition: vector3d(0,0,20)
size: vector3d(20,20,0)
texture: ""
type: 0
children: [ ]

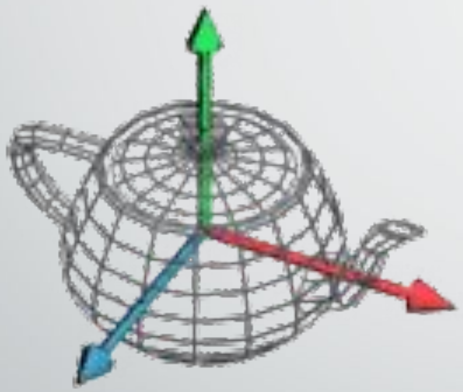
modeler
name: "wall3"
symbolPosition: vector3d(0,0,0)
size: vector3d(0,20,20)
texture: ""
type: 0
children: [ ]

modeler
name: "ground"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,0,20)
texture: ""
type: 0
children: [ ]

Taken:
P. Muller, G. Zeng, P. Wonka, and L. Van Gool. Image-based ¨ procedural modeling of facades. In SIGGRAPH, 2007

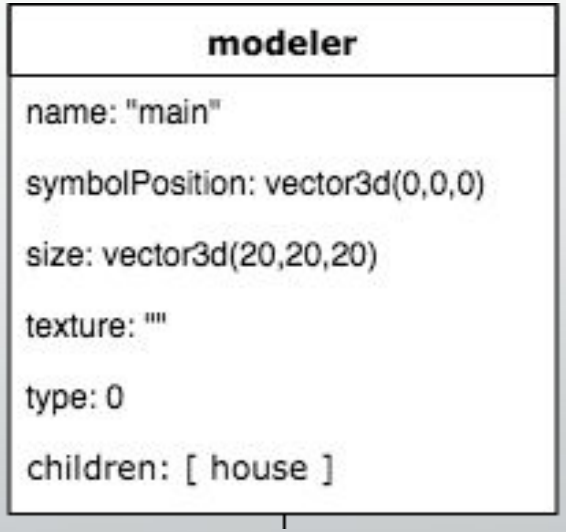
Procedural Grammar for Describing a Room

- **S(X,Y,Z)** : Set new size



Example:

main=**S(20,20,20)** Subdiv(0,20){house}



modeler

name: "main"

symbolPosition: vector3d(0,0,0)

size: vector3d(20,20,20)

texture: ""

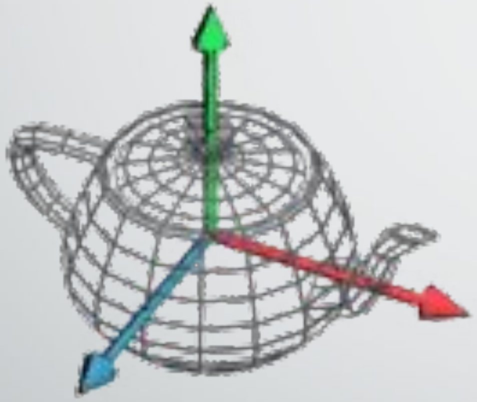
type: 0

children: [house]

Taken:
P. Muller, G. Zeng, P. Wonka, and L. Van Gool. Image-based ¨ procedural modeling of facades. In SIGGRAPH, 2007

Procedural Grammar for Describing a Room

- **S3d(axis,size)** : Set new size for an specific axis



Example:

chair1=**S3d(1,6)** I(cube){rug}:0.4

Axis are X=0, Y=1, Z=2

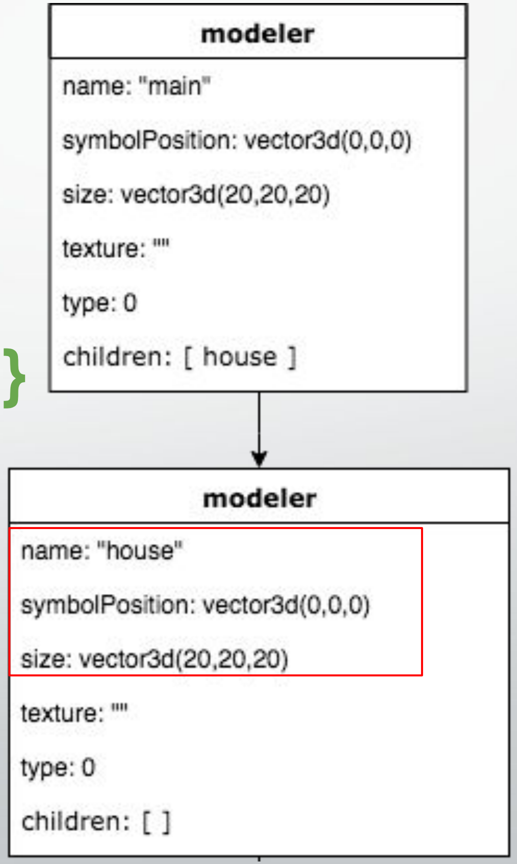
Procedural Grammar for Describing a Room

- **Subdiv(axis,arguments){parameters}** : Divide scope in smaller scopes (children)

The children obtain the position and the size from the father

Example:

main=S(20,20,20) **Subdiv(0,20){house}**



modeler

name: "main"

symbolPosition: vector3d(0,0,0)

size: vector3d(20,20,20)

texture: ""

type: 0

children: [house]

modeler

name: "house"

symbolPosition: vector3d(0,0,0)

size: vector3d(20,20,20)

texture: ""

type: 0

children: [ ]

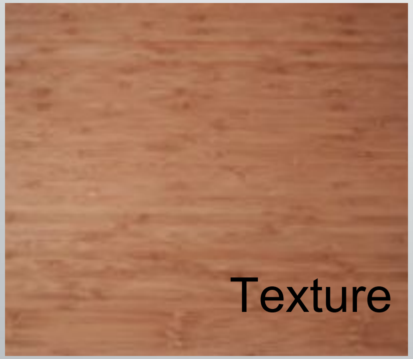
Axis are X=0, Y=1, Z=2

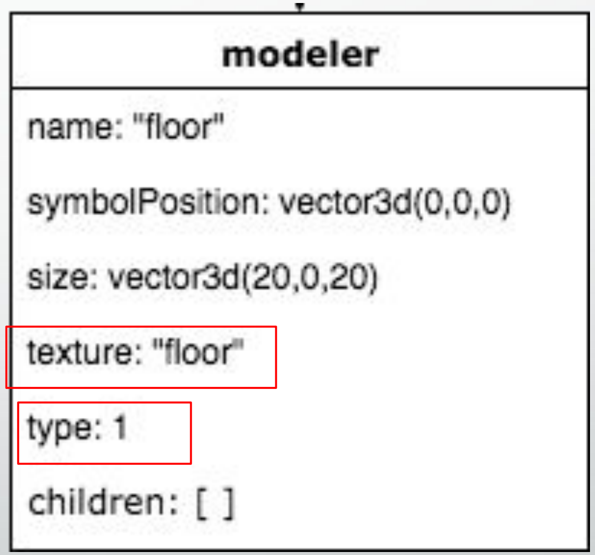
Taken:
P. Muller, G. Zeng, P. Wonka, and L. Van Gool. Image-based ¨ procedural modeling of facades. In SIGGRAPH, 2007

Procedural Grammar for Describing a Room

- **I(typeObject){texture}** : Instance of a geometry and texture

Example:
floor=I(plane){floor}





modeler

name: "floor"

symbolPosition: vector3d(0,0,0)

size: vector3d(20,0,20)

texture: "floor"

type: 1

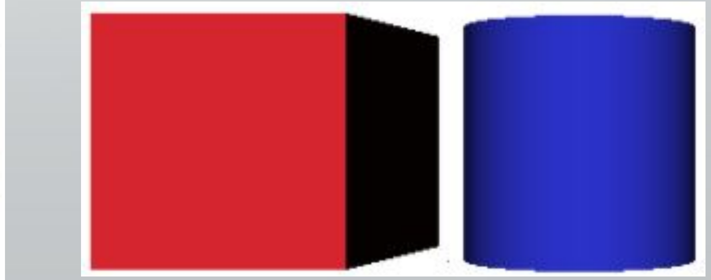
children: [ ]

Type 1 = PLANE

Taken:
P. Muller, G. Zeng, P. Wonka, and L. Van Gool. Image-based ¨ procedural modeling of facades. In SIGGRAPH, 2007

Procedural Grammar for Describing a Room

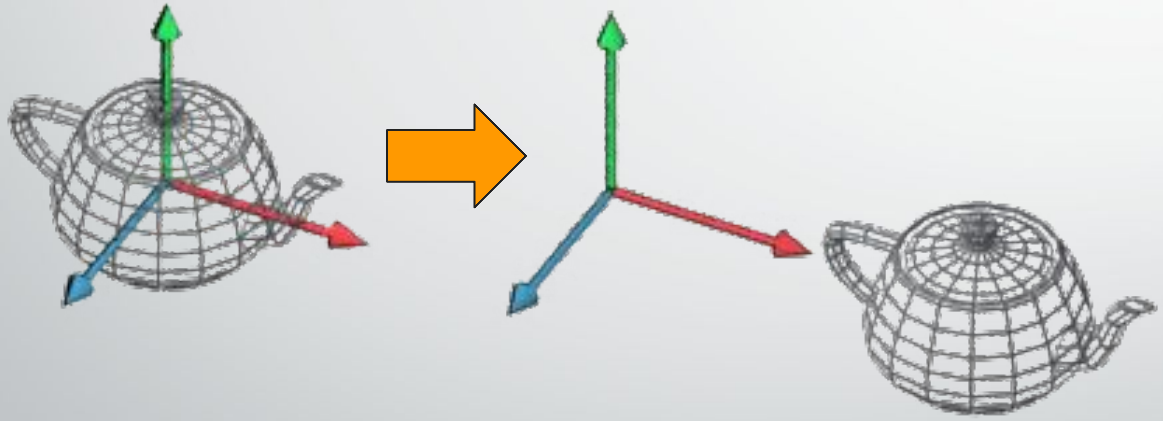
- In order to describe the primitive geometry and furniture objects, I create a global variable (typeObjects):
- enum TypeObject { SCOPE, PLANE, CUBE, CYLINDER, SOFA, TABLE, CABINET, CHAIR, TOY };

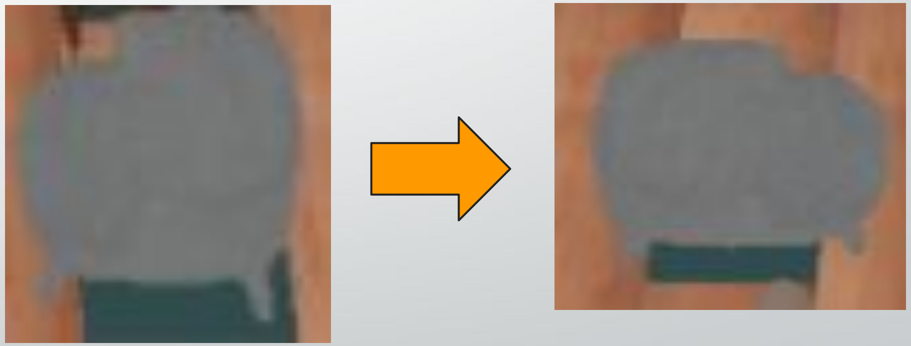


Procedural Grammar for Describing a Room

- **T(X,Y,Z)** : Translation of an object

Example:
chair1=S3d(1,4) **T(4,0,0)**
R(90,0,0) I(chair){armchair}:0.6



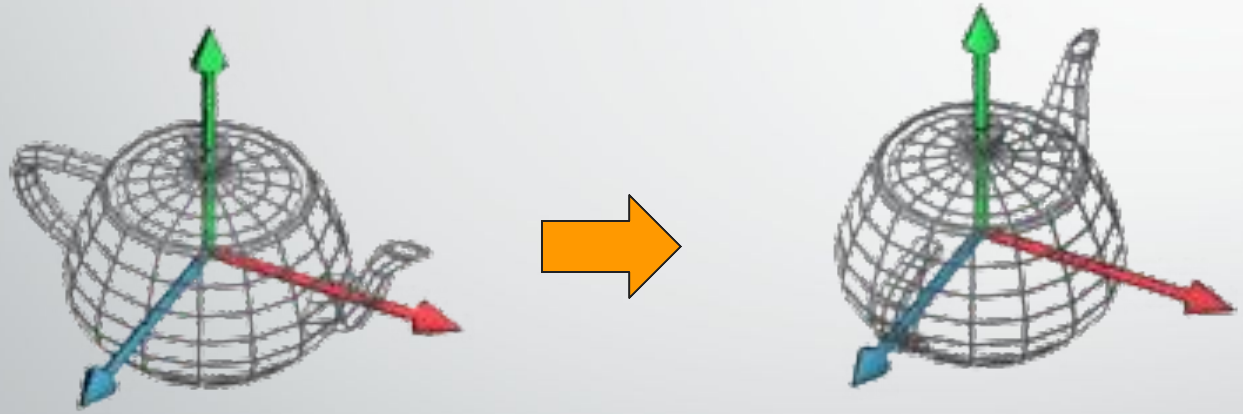


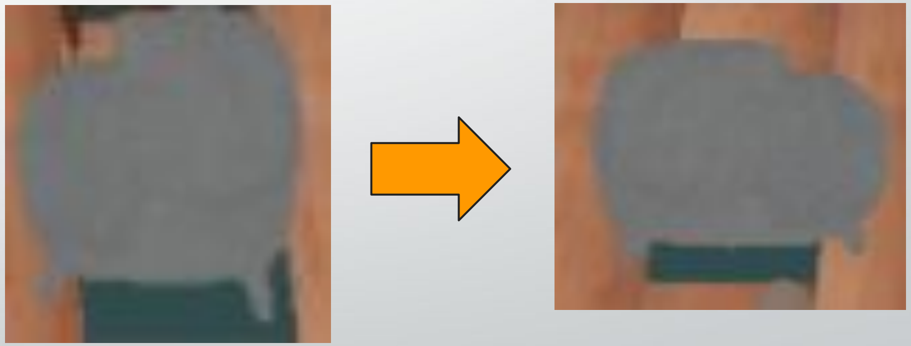
Taken:
P. Muller, G. Zeng, P. Wonka, and L. Van Gool. Image-based ¨ procedural modeling of facades. In SIGGRAPH, 2007

Procedural Grammar for Describing a Room

- **R(angleX,angleY,angleZ)** : Rotating around axis

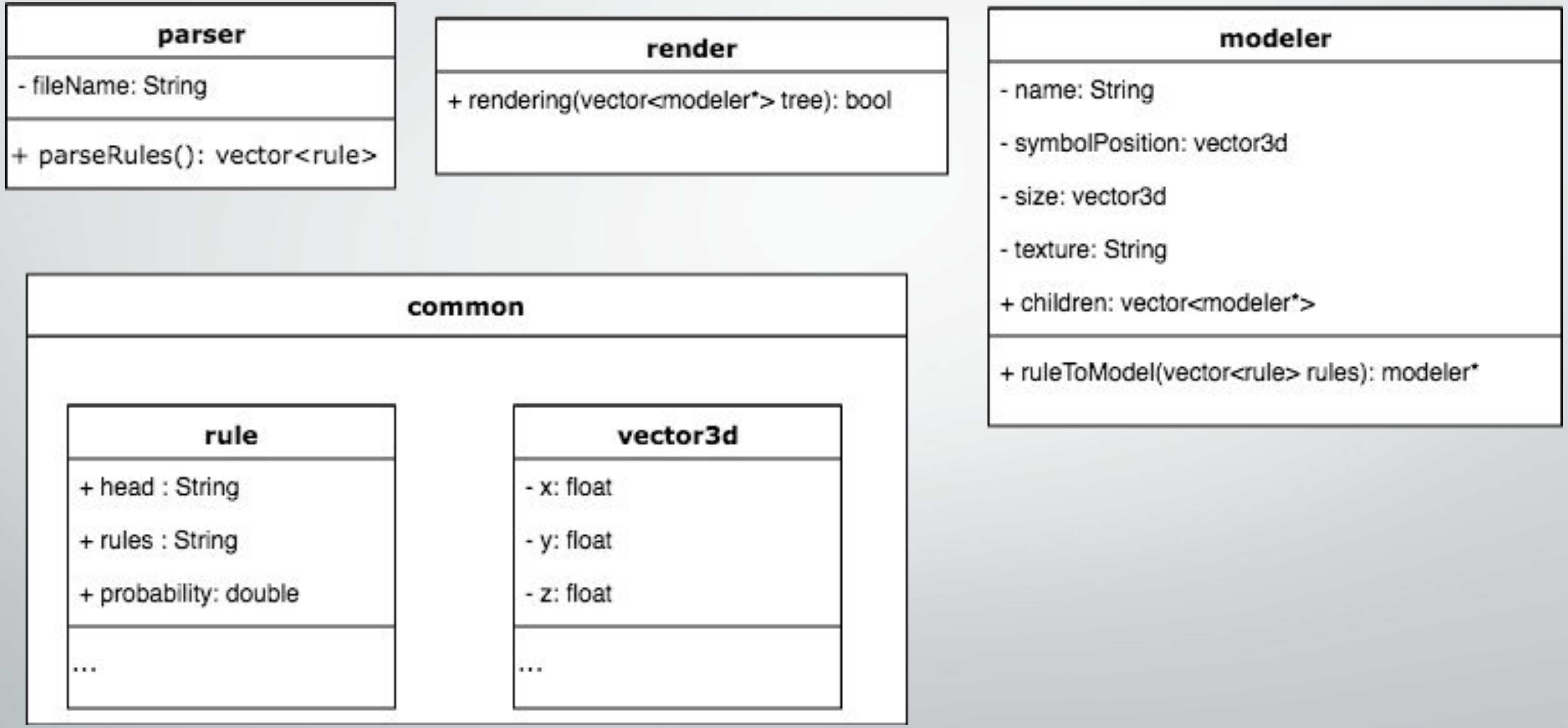
Example:
chair1=S3d(1,4) T(4,0,0)
R(90,0,0) I(chair){armchair}:0.6





Taken:
P. Muller, G. Zeng, P. Wonka, and L. Van Gool. Image-based ¨ procedural modeling of facades. In SIGGRAPH, 2007

Oriented Object Model



parser

- fileName: String

+ parseRules(): vector<rule>

render

+ rendering(vector<modeler*> tree): bool

modeler

- name: String
- symbolPosition: vector3d
- size: vector3d
- texture: String
+ children: vector<modeler*>

+ ruleToModel(vector<rule> rules): modeler*

common

rule

+ head : String
+ rules : String
+ probability: double

...

vector3d

- x: float
- y: float
- z: float

...

Parsing the File

1. Read the file and save it in a vector<string>.
2. Parse the vector<string> into a vector<rule>.
    a. Save the repeat heads in a temporal vector<rule>.
    b. Create a random between 0 and 1.
    c. Find the interval probability that matches the random; save the winner in another vector<rule>.
    d. Erase the losers from the original vector<rule>.
    e. And repeat it until there is no more repeat heads.
3. Return the vector<rule>

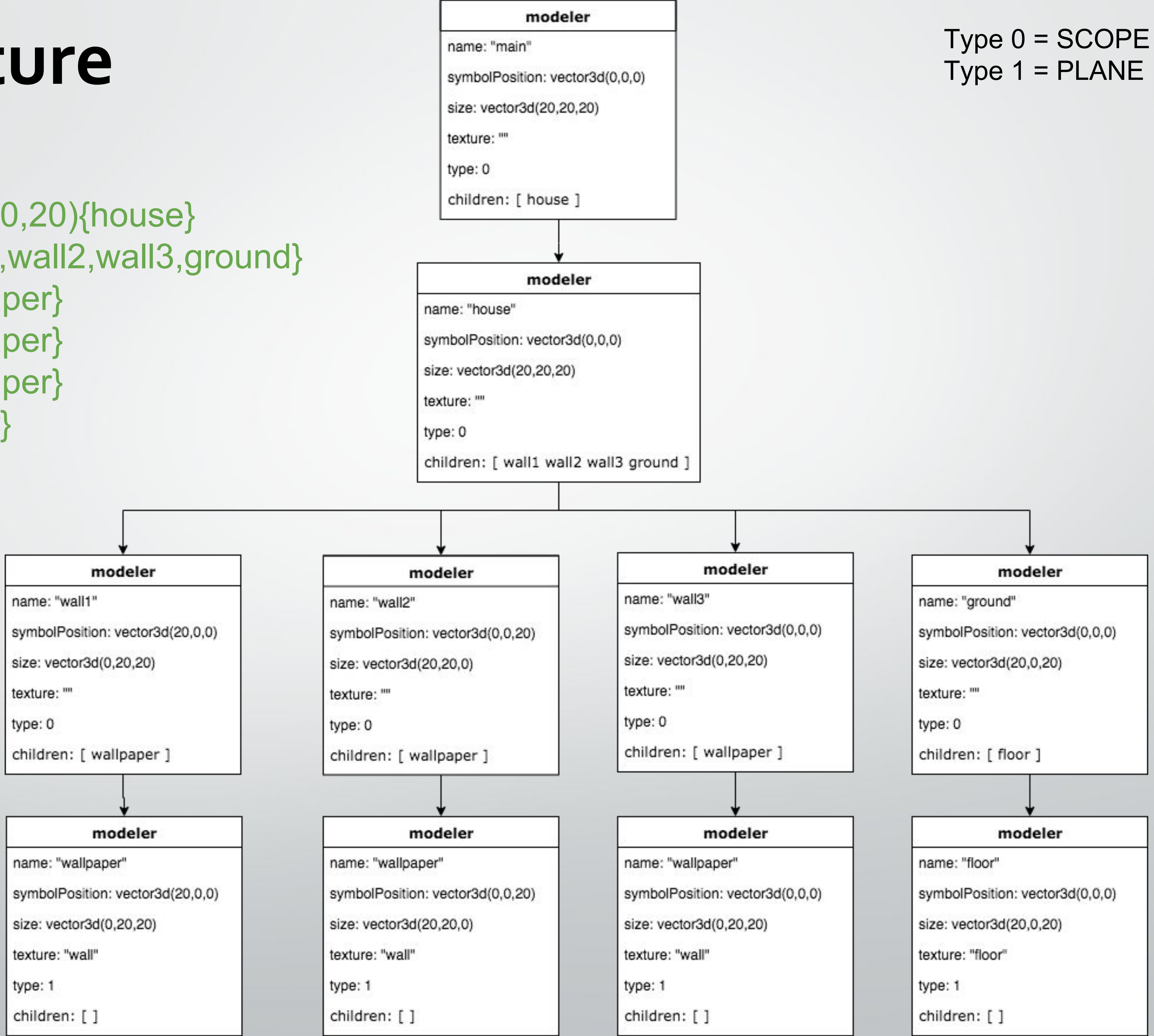
Modeler: Data Structure (Tree)

1. Initialize the model (modeler* tree) and add the model in vector<modeler*> list
2. Copy the first model in the list in a temporal variable (modeler* currentModel)
3. Erase the first element of the list
4. Search if the head exist
    a. Extract the rules and parse to obtain the operator rule (vector<string> keys)
    b. Compare if the keys start with T,S,S3d,Subdiv,Comp,I,…
    c. Transform the rules in modeler*. If it contain children e.g Subdiv then save it in the model but also insert them into the vector<modeler*> list
5. Do the process until the list is empty
6. Return the tree

Tree Structure

Type 0 = SCOPE
Type 1 = PLANE

```
main=S(20,20,20) Subdiv(0,20){house}
house=Comp(sides){wall1,wall2,wall3,ground}
wall1=Subdiv(1,20){wallpaper}
wall2=Subdiv(1,20){wallpaper}
wall3=Subdiv(1,20){wallpaper}
ground=Subdiv(0,20){floor}
wallpaper=I(plane){wall}
floor=I(plane){floor}
```



modeler
name: "main"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,20,20)
texture: ""
type: 0
children: [house]

modeler
name: "house"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,20,20)
texture: ""
type: 0
children: [wall1 wall2 wall3 ground]

modeler
name: "wall1"
symbolPosition: vector3d(20,0,0)
size: vector3d(0,20,20)
texture: ""
type: 0
children: [wallpaper]

modeler
name: "wall2"
symbolPosition: vector3d(0,0,20)
size: vector3d(20,20,0)
texture: ""
type: 0
children: [wallpaper]

modeler
name: "wall3"
symbolPosition: vector3d(0,0,0)
size: vector3d(0,20,20)
texture: ""
type: 0
children: [wallpaper]

modeler
name: "ground"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,0,20)
texture: ""
type: 0
children: [floor]

modeler
name: "wallpaper"
symbolPosition: vector3d(20,0,0)
size: vector3d(0,20,20)
texture: "wall"
type: 1
children: [ ]

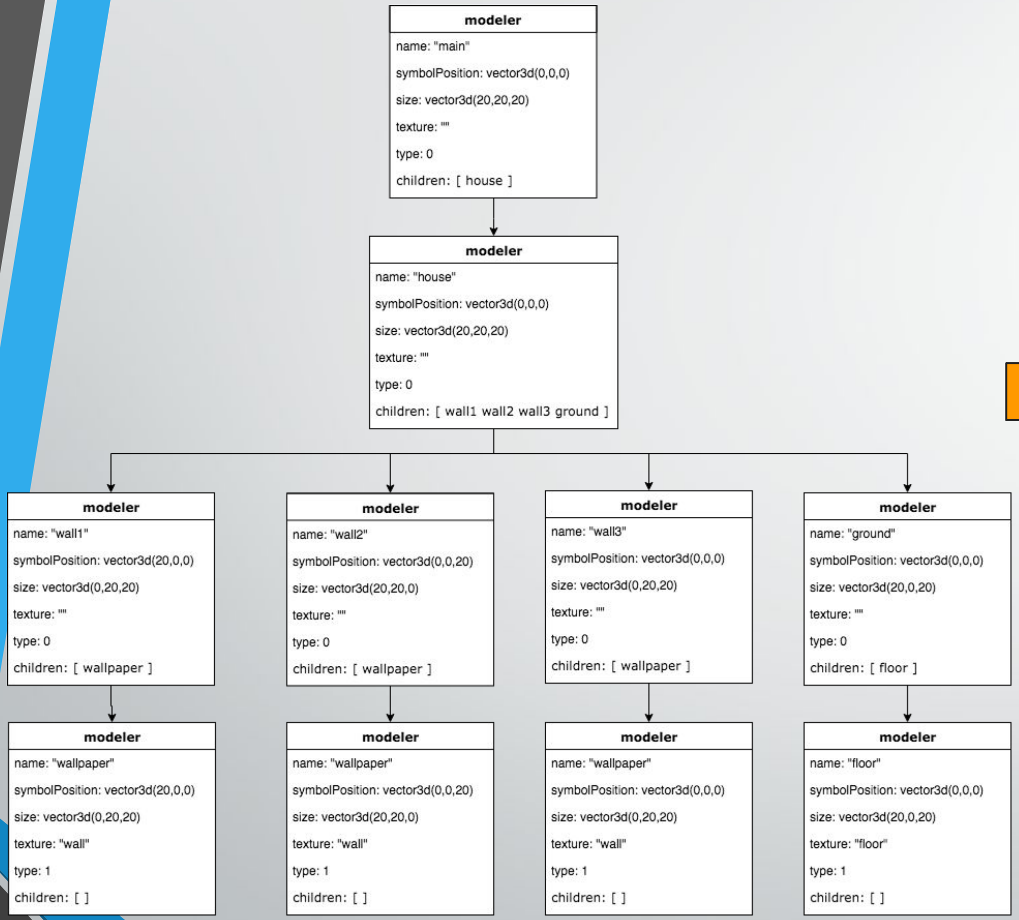
modeler
name: "wallpaper"
symbolPosition: vector3d(0,0,20)
size: vector3d(20,20,0)
texture: "wall"
type: 1
children: [ ]

modeler
name: "wallpaper"
symbolPosition: vector3d(0,0,0)
size: vector3d(0,20,20)
texture: "wall"
type: 1
children: [ ]

modeler
name: "floor"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,0,20)
texture: "floor"
type: 1
children: [ ]

Tree Structure

Type 0 = SCOPE
Type 1 = PLANE



modeler
name: "main"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,20,20)
texture: ""
type: 0
children: [house]

modeler
name: "house"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,20,20)
texture: ""
type: 0
children: [wall1 wall2 wall3 ground]

modeler
name: "wall1"
symbolPosition: vector3d(20,0,0)
size: vector3d(0,20,20)
texture: ""
type: 0
children: [wallpaper]

modeler
name: "wall2"
symbolPosition: vector3d(0,0,20)
size: vector3d(20,20,0)
texture: ""
type: 0
children: [wallpaper]

modeler
name: "wall3"
symbolPosition: vector3d(0,0,0)
size: vector3d(0,20,20)
texture: ""
type: 0
children: [wallpaper]

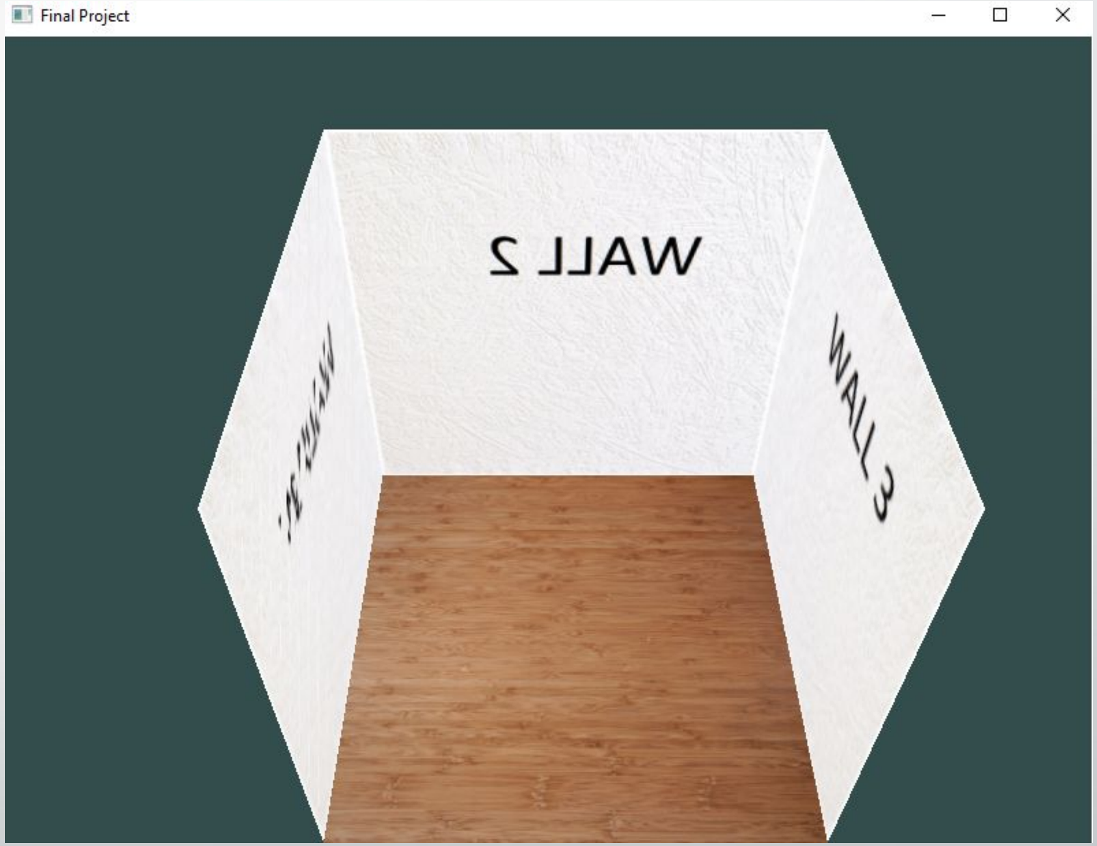
modeler
name: "ground"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,0,20)
texture: ""
type: 0
children: [floor]

modeler
name: "wallpaper"
symbolPosition: vector3d(20,0,0)
size: vector3d(0,20,20)
texture: "wall"
type: 1
children: [ ]

modeler
name: "wallpaper"
symbolPosition: vector3d(0,0,20)
size: vector3d(20,20,0)
texture: "wall"
type: 1
children: [ ]

modeler
name: "wallpaper"
symbolPosition: vector3d(0,0,0)
size: vector3d(0,20,20)
texture: "wall"
type: 1
children: [ ]

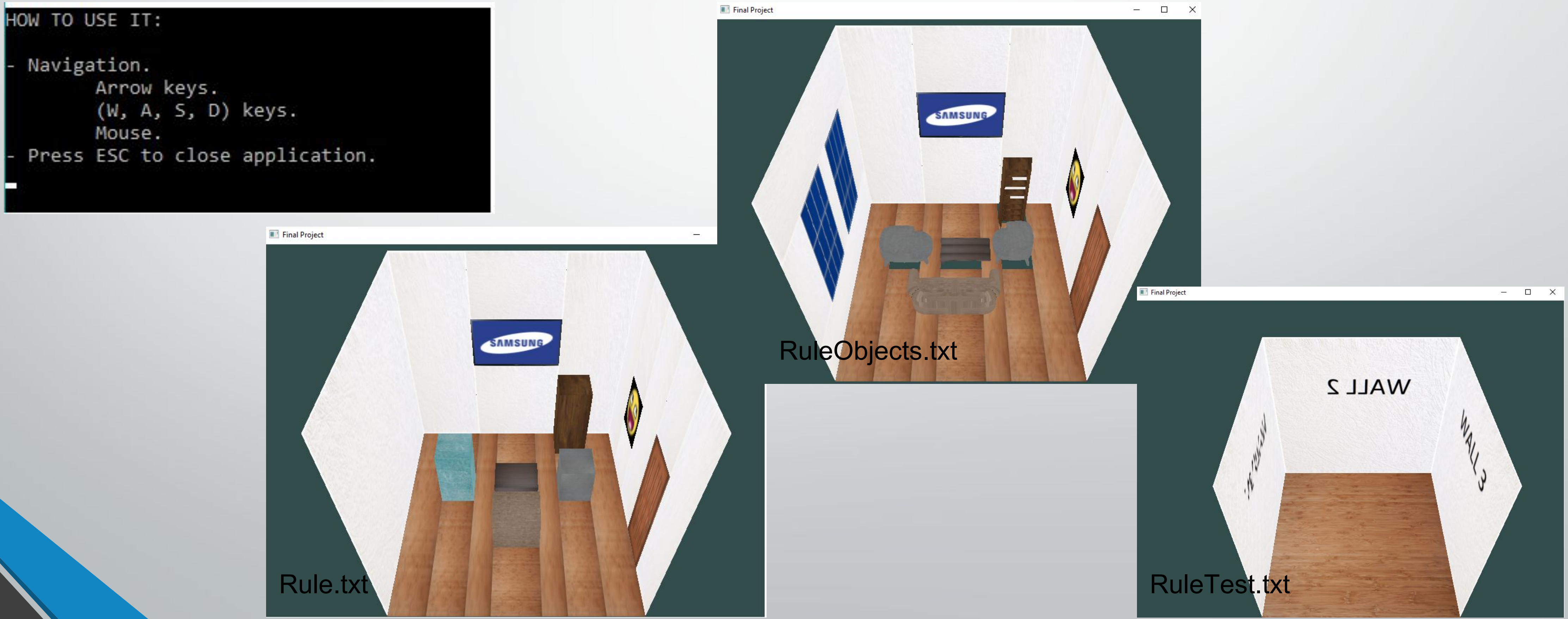
modeler
name: "floor"
symbolPosition: vector3d(0,0,0)
size: vector3d(20,0,20)
texture: "floor"
type: 1
children: [ ]



Result to Rendering

1. Transform the modeler* tree in a vector<modeler*> model
2. Save the model in a global variable
3. Save the respective model in the ourShader variable and render it.
4. The program permits zoom, movement in the scene with the WASD/arrow keys and mouse.

Screenshots



```
HOW TO USE IT:

- Navigation.
        Arrow keys.
        (W, A, S, D) keys.
        Mouse.
- Press ESC to close application.
```

Thank you.

Questions?