Computer Graphics
Practical 3

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

INSA Fourth Year - 2025/2026
Maud Marchal, Kelian Baert, Pierre-Antoine Cabaret

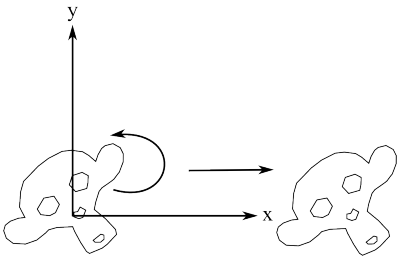
1 Objectives

Hierarchical modeling is a way of composing complex objects out of multiple simpler primitives. Objects are assembled in a hierarchical way. Their position are defined relatively to their parents using a **combination** of transformations such as rotation, translation and scaling. A classical example of hierarchical model is a tree.

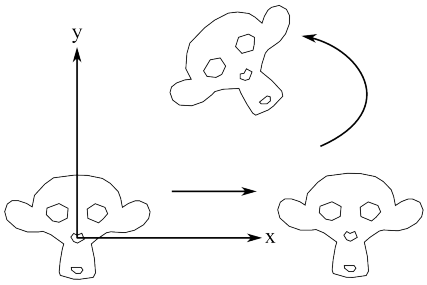
Transformations are combined using matrix products and homogeneous coordinates. For example:

- a rotation R followed by a translation T will be described by the product TR;
- a translation followed by a rotation will be described by the product RT.

Warning: be careful about the order!



Rotation then translation.



Translation then rotation.

2 Exercice 1: HierarchicalRenderable

Recall that `MeshRenderable` is derived from `HierarchicalRenderable`. In this exercice, we will implement the missing parts of the latter.

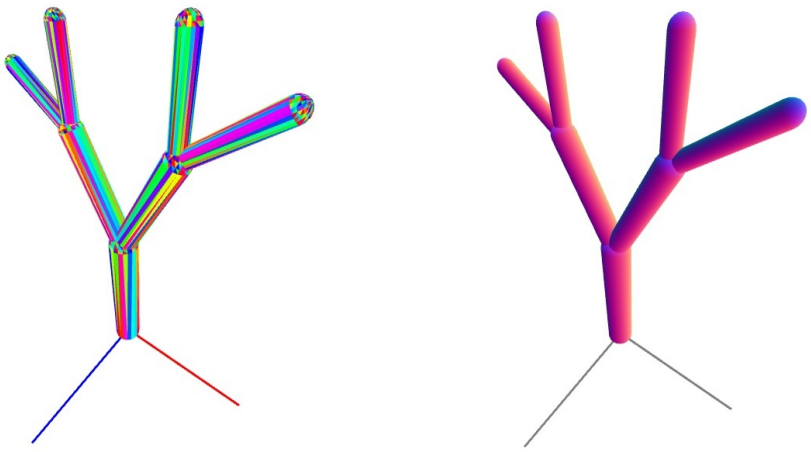
- Read the `HierarchicalRenderable` class to understand how it works.

- In the file `HierarchicalRenderable.cpp`, there are several functions that must be filled to make `HierarchicalRenderable` work. We advise you to do it in the following order:
  - `computeTotalGlobalTransform()`
  - `updateModelMatrix()`
- Build a hierarchical model in the main function using the following pattern:

```
//...
#include "./../include/SomeDerivedHierarchicalRenderable.hpp"
//...

void main()
{
    //...

    // create programs (you can use the same program for both the
        child and parent)
    ShaderProgramPtr parentProg = std::make_shared<ShaderProgram>(
        ... ) ;
    ShaderProgramPtr childProg = std::make_shared<ShaderProgram>( ...
         ) ;
    viewer.addShaderProgram(parentProg);
    viewer.addShaderProgram(childProg);

    // Create renderables
    std::shared_ptr<SomeDerivedHierarchicalRenderable> root = std::
        make_shared<SomeDerivedHierarchicalRenderable>(parentProg);
    std::shared_ptr<SomeDerivedHierarchicalRenderable> child1 = std::
        make_shared<SomeDerivedHierarchicalRenderable>(childProg);

    // For each element of the hierarchy,
    // Set local transform and global transform
    glm::mat4 rootGlobalTransform;
    root->setGlobalTransform(rootGlobalTransform);

    glm::mat4 child1GlobalTransform;
    child1->setGlobalTransform(child1GlobalTransform);
    glm::mat4 child1LocalTransform;
    child1->setLocalTransform(child1LocalTransform);

    // Define parent/children relationships
    HierarchicalRenderable::addChild(root, child1);

    // Add the root of the hierarchy to the viewer
    viewer.addRenderable(root);

    //...
}
```

3 Exercice 2: Grow a tree

Using the hierarchy logic, build a tree of `CylinderMeshRenderables`. You can also add `SphereMeshRenderables` to simulate the joints of your tree.



Your tree might look like this (left: random colors, right: normals as color)

4 Project-related exercise

In the final project you will have to design a turtle and some surrounding pieces of environment. Imagine you would like to model their geometry from a combination of primitives (simple meshes). Start building the basic 3D pieces you will need to create your turtle. In the next practical, we will animate them, so you can already think about rotation axes for example.