Introduction to Hierarchical Modeling using Inventor

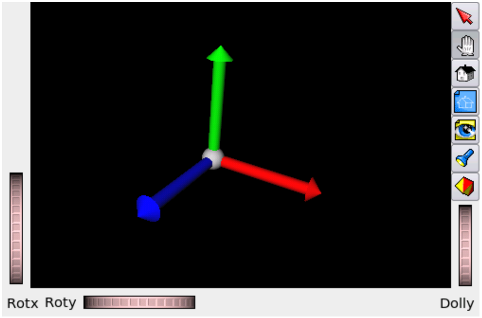
François Faure

EVASION-LJK

A graphical model

Composed of :

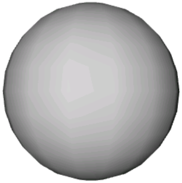
- geometric shapes
- displacements
- colors



A sphere

Text file :

```
#Inventor V2.0 ascii

Sphere {}
```



- File format : `Inventor V2.0 ascii`
- One single object with default values

Adding a cylinder

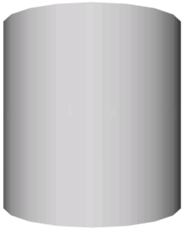
Text file :

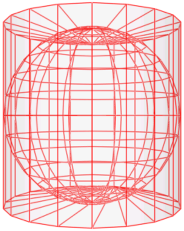
```
#Inventor V2.0 ascii

Sphere {}
Cylinder {}
```

or, equivalently :

```
#Inventor V2.0 ascii

Cylinder {}
Sphere {}
```

- The cylinder has default values too
- The order does not matter here

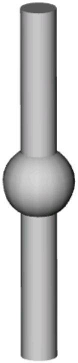




Tuning the parameters

Text file :

```
#Inventor V2.0 ascii

Sphere {
    radius 0.1
}
Cylinder {
    radius 0.05
    height 1
}
```

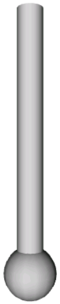


- Objects have attributes
- Position and orientation are not shape attributes

Changing the coordinate system

Text file :

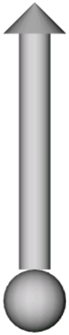
```
#Inventor V2.0 ascii

Sphere {
    radius 0.1
}
Translation {
    translation 0 0.5 0
}
Cylinder {
    radius 0.05
    height 1
}
```



- Objects are drawn in their local coordinate system

Adding a cone

```
#Inventor V2.0 ascii

Sphere {
    radius 0.1
}
Translation {
    translation 0 0.5 0
}
Cylinder {
    radius 0.05
    height 0.8
}
Translation {
    translation 0 0.4 0
}
Cone {
    bottomRadius 0.1
    height 0.1
}
```

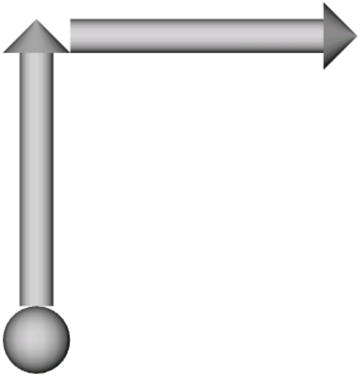


- Geometric transforms are combined
- Basic shapes are centered at the origin of the current coordinate system

Adding the X arrow

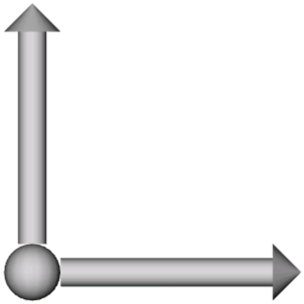
```
#Inventor V2.0 ascii

Sphere { radius 0.1 }

Translation{ translation 0 0.5 0 }
Cylinder {
    radius 0.05
    height 0.8
}
Translation { translation 0 0.4 0 }
Cone {
    bottomRadius 0.1
    height 0.1
}

# Rotate then draw the arrow again
RotationXYZ { axis Z angle —1.5708 }

Translation { translation 0 0.5 0 }
Cylinder {
    radius 0.05
    height 0.8
}
Translation { translation 0 0.4 0 }
Cone {
    bottomRadius 0.1
    height 0.1
}
```

- Rotate along Z
- Need a translation to start back from the origin



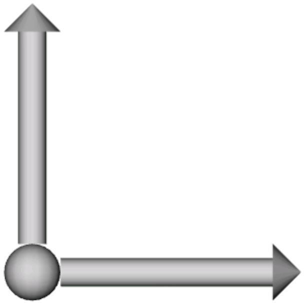
Better

```
#Inventor V2.0 ascii

Sphere { radius 0.1 }
Translation { translation 0 0.5 0 }
Cylinder {
    radius 0.05
    height 0.8
}
Translation { translation 0 0.4 0 }
Cone {
    bottomRadius 0.1
    height 0.1
}

# back to origin then draw again
Translation { translation 0 −0.9 0 }
RotationXYZ { axis Z angle −1.5708 }

Translation { translation 0 0.5 0 }
Cylinder {
    radius 0.05
    height 0.8
}
Translation { translation 0 0.4 0 }
Cone {
    bottomRadius 0.1
    height 0.1
}
```



Groups

```
#Inventor V2.0 ascii

Sphere { radius 0.1 }

DEF arrow Group {
    Translation { translation 0 0.5 0 }
    Cylinder {
        radius 0.05
        height 0.8
    }
    Translation { translation 0 0.4 0 }
    Cone {
        bottomRadius 0.1
        height 0.1
    }
}

# back to the origin
Translation { translation 0 −0.9 0 }

# then rotate and redraw
RotationXYZ { axis Z angle −1.5708 }
USE arrow
```



- Groups define hierarchical structures
- Naming a group allows us to re-use it

Separators

- Separators push/pop state values
- Separators leave the state unchanged

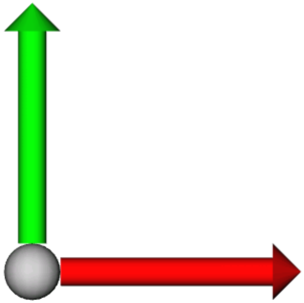
```
#Inventor V2.0 ascii

Sphere { radius 0.1 }

DEF arrow Separator {
    Translation { translation 0 0.5 0 }
    Cylinder {
        radius 0.05
        height 0.8
    }
    Translation { translation 0 0.4 0 }
    Cone {
        bottomRadius 0.1
        height 0.1
    }
}
# automatically back to previous state

# then rotate and redraw
RotationXYZ { axis Z angle −1.5708 }
USE arrow
```

- Groups “contaminate” the rest of the scene and may induce data dependency

```
#Inventor V2.0 ascii

Sphere { radius 0.1 }

DEF arrow Group {
    Translation { translation 0 0.5 0 }
    Cylinder {
        radius 0.05
        height 0.8
    }
    Translation { translation 0 0.4 0 }
    Cone {
        bottomRadius 0.1
        height 0.1
    }
}

# back to the origin
Translation { translation 0 −0.9 0 }

# then rotate and redraw
RotationXYZ { axis Z angle −1.5708 }
USE arrow
```

Colors

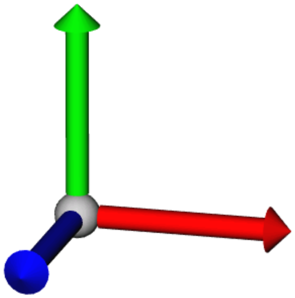
```
#Inventor V2.0 ascii

Sphere { radius 0.1 }

Material { diffuseColor 0 1 0 }
DEF arrow Separator
{
    Translation { translation 0 0.5 0 }
    Cylinder { radius 0.05
               height 0.8}
    Translation { translation 0 0.4 0 }
    Cone { bottomRadius 0.1
           height 0.1 }
}

RotationXYZ { axis Z angle −1.5708 }
Material { diffuseColor 1 0 0 }

USE arrow
```



- Separators leave the state unchanged

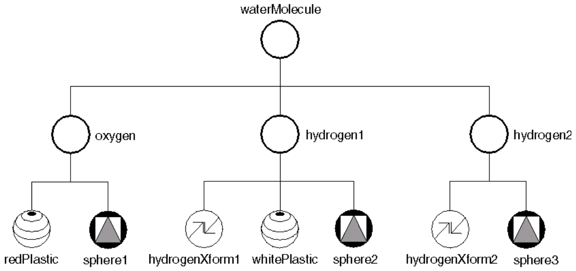
Adding the Z axis

```
#Inventor V2.0 ascii

Sphere { radius 0.1 }

Material { diffuseColor 0 1 0 }
DEF arrow Separator
{
    Translation { translation 0 0.5 0 }
    Cylinder { radius 0.05
               height 0.8}
    Translation { translation 0 0.4 0 }
    Cone { bottomRadius 0.1
           height 0.1 }
}

RotationXYZ { axis Z angle −1.5708 }
Material { diffuseColor 1 0 0 }

USE arrow

RotationXYZ { axis X angle 1.5708 }
Material { diffuseColor 0 0 1 }

USE arrow
```



Scene graph

- The scene is modeled as a tree-like structure (Directed Acyclic Graph)
- Three families of rendering nodes :
  - Shape nodes, which represent 3D geometric objects
  - Property nodes, which represent appearance and other qualitative characteristics of the scene
  - Group nodes, which are containers that collect nodes into graphs



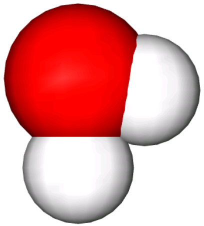


Image from "The Inventor Mentor", Josie Wernecke, Addison-Wesley

The Traversal state

- Scene processing is performed by depth-first graph traversals
- The drawings depend of the traversal state :
  - Current geometric transformation
  - Current material components
  - Current lighting model
  - Current drawing style
  - Current text font
  - Current coordinates
  - Current normals
  - Current lights
  - Current viewing specification
- Separators push/pop the traversal state
- See chapter 3 of *The Inventor Mentor*

Actions

Different actions (graph traversals) can be applied to the scene graph :

- Draw, or *render*, the scene graph
- Compute a 3D bounding box for objects in the scene graph
- Compute a cumulative transformation matrix (and its inverse)
- Write the scene graph to a file
- Search for nodes
- Allow objects in the scene graph to handle an event
- Pick objects in the scene graph along a ray
- Traverse the scene graph and accumulate traversal state, then perform your own action using callback functions
- See chapter 9 of *The Inventor Mentor*

Modeling the Water Molecule in C++

```
// Code from ''The Inventor mentor'', Josie Wernecke, Addison-Wesley
// Construct all parts
SoGroup *waterMolecule = new SoGroup;        // water molecule

SoGroup *oxygen = new SoGroup;               // oxygen atom
SoMaterial *redPlastic = new SoMaterial;
SoSphere *sphere1 = new SoSphere;

SoGroup *hydrogen1 = new SoGroup;            // hydrogen atoms
SoGroup *hydrogen2 = new SoGroup;
SoTransform *hydrogenXform1 = new SoTransform;
SoTransform *hydrogenXform2 = new SoTransform;
SoMaterial *whitePlastic = new SoMaterial;
SoSphere *sphere2 = new SoSphere;
SoSphere *sphere3 = new SoSphere;

// Set all field values for the oxygen atom
redPlastic->ambientColor.setValue(1.0, 0.0, 0.0);
redPlastic->diffuseColor.setValue(1.0, 0.0, 0.0);
redPlastic->specularColor.setValue(0.5, 0.5, 0.5);
redPlastic->shininess = 0.5;
```

Modeling the Water Molecule in C++ (continued)

```
// Code from ''The Inventor mentor'', Josie Wernecke, Addison-Wesley
// Set all field values for the hydrogen atoms
hydrogenXform1->scaleFactor.setValue(0.75, 0.75, 0.75);
hydrogenXform1->translation.setValue(0.0, -1.2, 0.0);
hydrogenXform2->translation.setValue(1.1852, 1.3877, 0.0);
whitePlastic->ambientColor.setValue(1.0, 1.0, 1.0);
whitePlastic->diffuseColor.setValue(1.0, 1.0, 1.0);
whitePlastic->specularColor.setValue(0.5, 0.5, 0.5);
whitePlastic->shininess = 0.5;

// Create a hierarchy
waterMolecule->addChild(oxygen);
waterMolecule->addChild(hydrogen1);
waterMolecule->addChild(hydrogen2);

oxygen->addChild(redPlastic);
oxygen->addChild(sphere1);
hydrogen1->addChild(hydrogenXform1);
hydrogen1->addChild(whitePlastic);
hydrogen1->addChild(sphere2);
hydrogen2->addChild(hydrogenXform2);
hydrogen2->addChild(sphere3);
```