High-level real-time 3D graphics development with Coin3D

Karin Kosina (vka kyrah)

Introduction

* Karin Kosina (vka kyrah)
* Computer graphics programmer and lecturer
* Maintainer of Coin3D Mac OS X port
* Feel free to get in touch with me at:
  kyrah@coin3d.org

  PGP fingerprint:
  10EA 9B79 1DDB 7535 1AAB
  4E8D C2B8 C0AC BE32 63FE

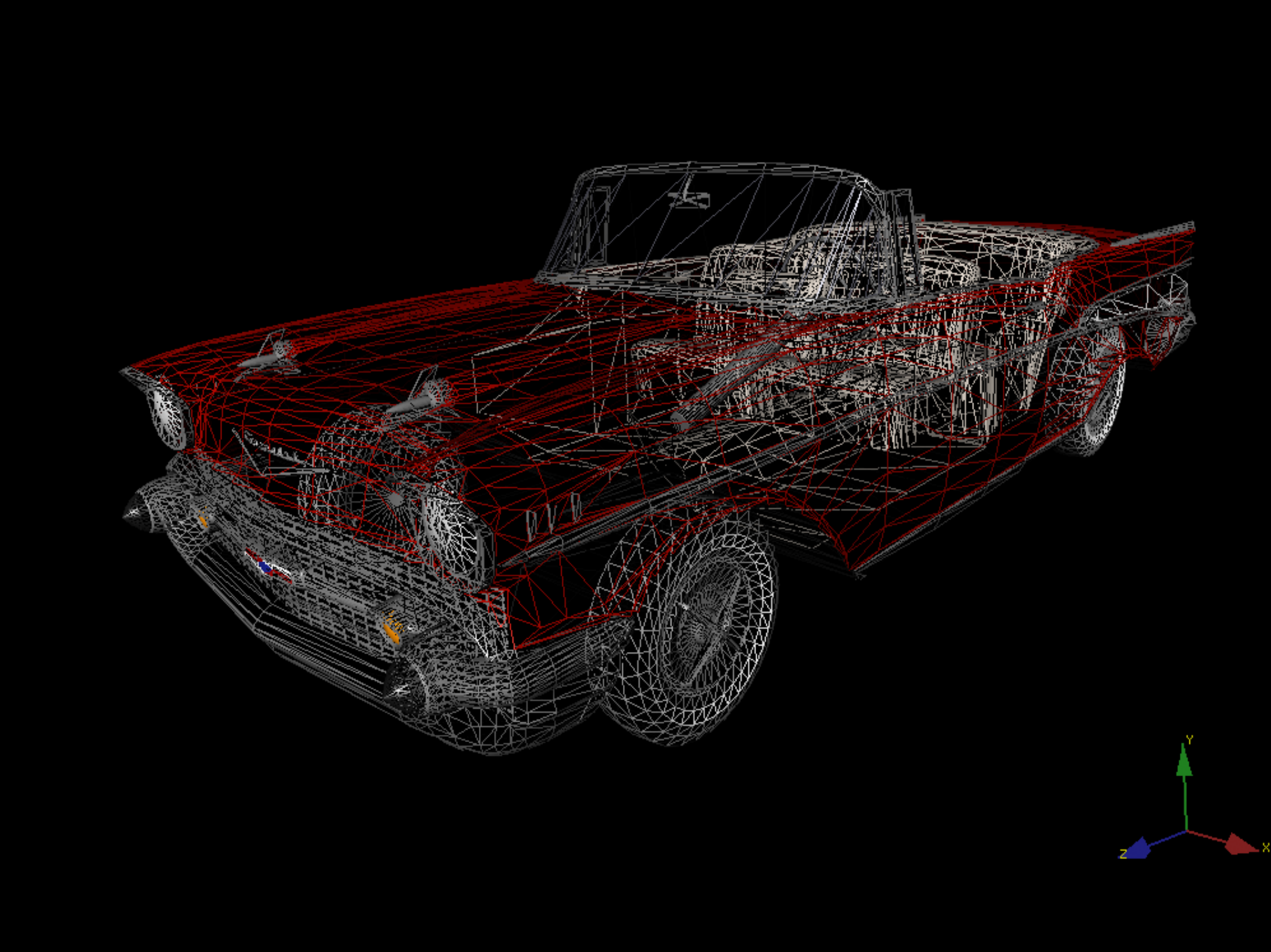
Introduction (your turn)

* Programmers?
  * C++?
  * Python?
* Computer graphics?
  * OpenGL?
  * High-level toolkits?
  * Coin3D/Open Inventor?

What is 3D Graphics?

* Relatively new scientific field (first papers from the 1960s)
* We want to display a virtual (3D) scene on a (2D) screen
  * Concept: taking a picture with a virtual camera
  * This process is often called *rendering*.
* 3D model (mathematical description of an object)
  * e.g. a sphere: defined by its radius
* Relationship between the objects in the scene
* position of the objects in "world space"
* Attributes
  * colour, lighting,...


Y
Z
X


Y
Z
X

So how does this work?

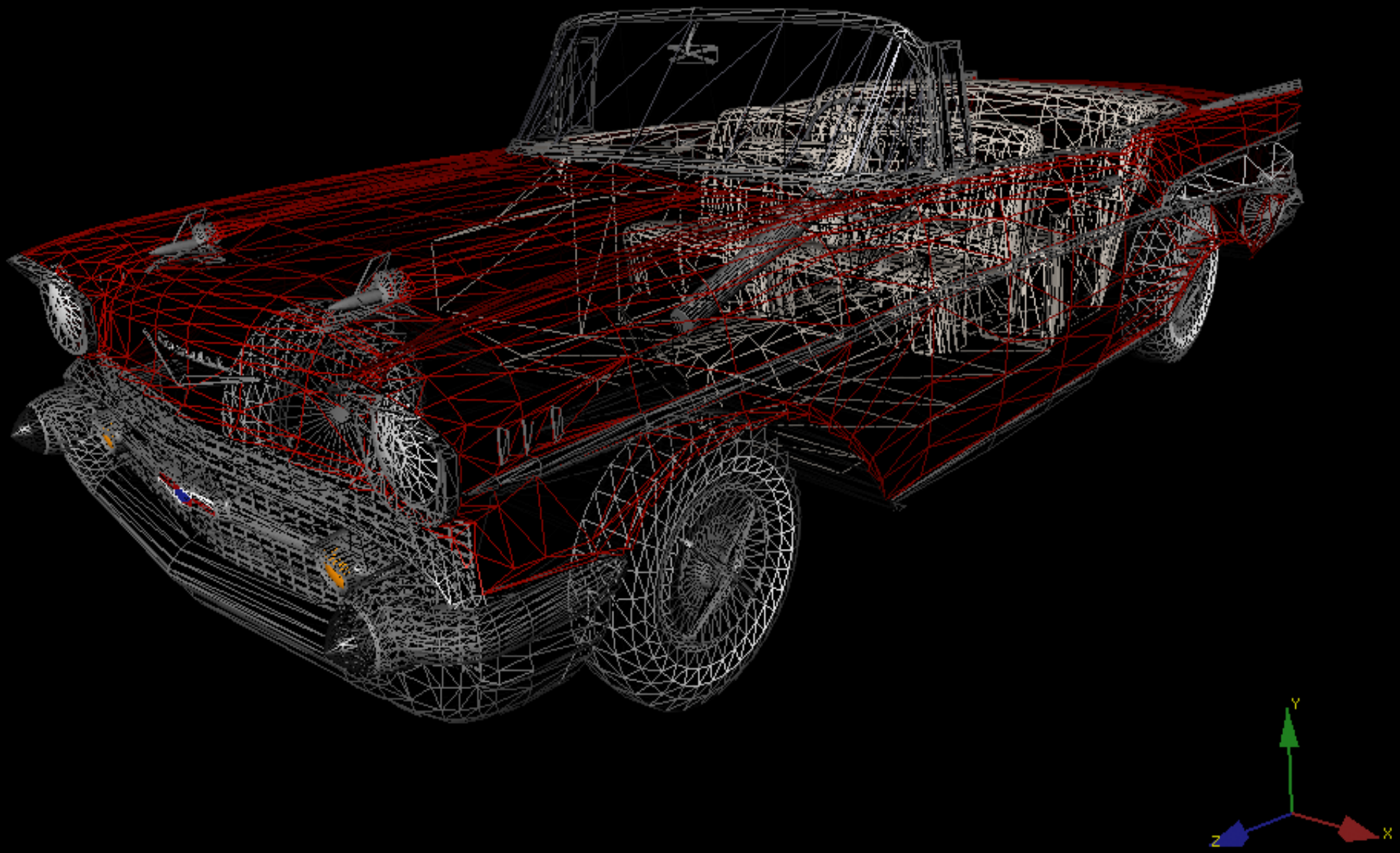
* Most widely used library for 3D graphics is OpenGL.
  * Cross-platform standard, very flexible, very fast.
* Direct3D
  * Microsoft Windows' proprietary 3D library.

So how does this work?

* Most widely used library for 3D graphics is OpenGL.
  * Cross-platform standard, very flexible, very fast.
* ~~Direct3D~~
  * ~~Microsoft Windows' proprietary 3D library.~~

So how does this work?

* Most widely used library for 3D graphics is OpenGL.
  * Cross-platform standard, very flexible, very fast.
* OpenGL works great, but it is very low-level.
  * You have to think in triangles, not in objects.

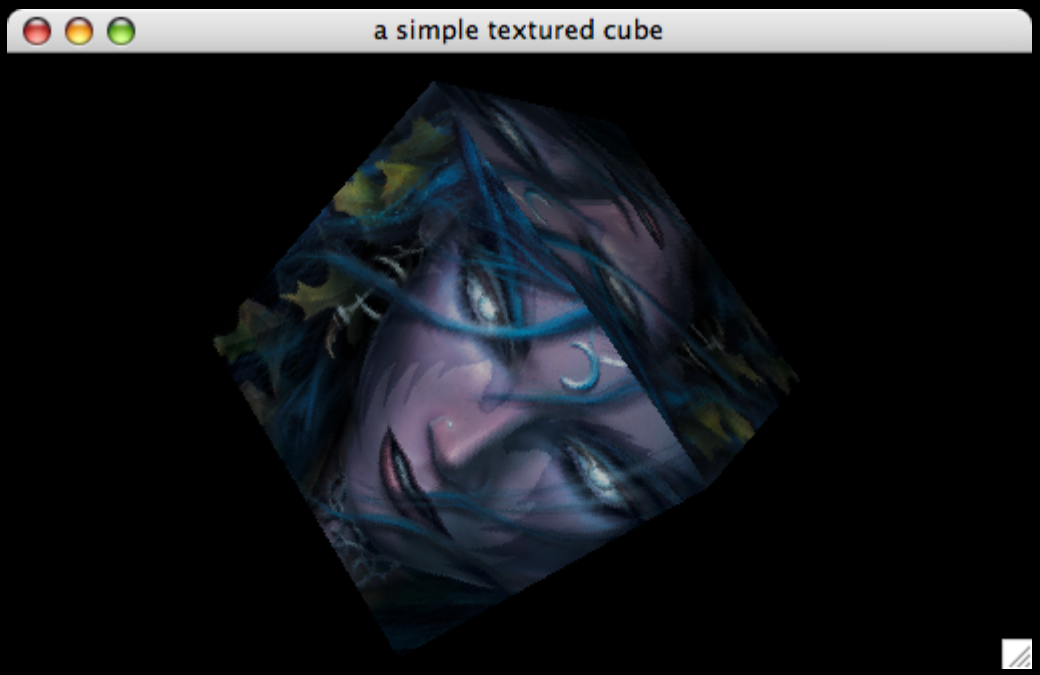

Y
Z
X

So how does this work?

* Most widely used library for 3D graphics is OpenGL.
  * Cross-platform standard, very flexible, very fast.
* OpenGL works great, but it is very low-level.
  * You have to think in triangles, not in objects.
  * You have to organise the scene yourself.
  * You have to optimise for different graphics cards.
* Alternative: High-level 3D graphics
  * Scene arranged in hierarchical structure
  * Higher abstraction level
  * Ease of use, programmer convenience

Let me show you what I mean...

A simple textured cube:



* OpenGL: 560 loc
* Coin: 37 loc

texturedcube/opengl/foo
texturedcube/coin/bar

High-Level 3D Graphics APIs

* All 3D graphics projects end up using *some* kind of scene abstraction...
* Option #1: Re-invent the wheel and write your own...
  * especially popular in games
* Option #2: Use an existing SDK
  * Open Inventor/Coin3D
  * Performer, OpenSG, Java3D,...

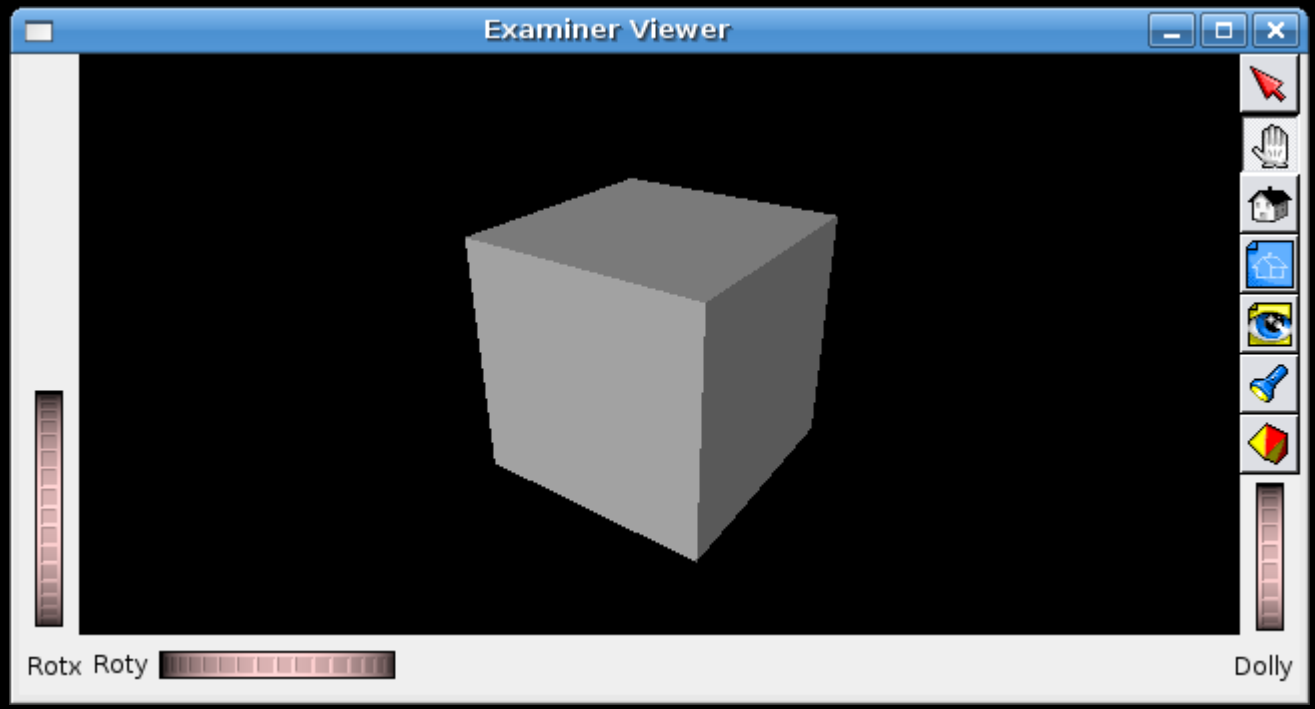
Coin3D Overview

* C++ object oriented high-level 3D graphics API
* Available as Free Software under the GNU GPL
* Portable across a wide range of platforms
  * GNU/Linux, Mac OS X, Windows, SGI Irix,...
* Uses OpenGL for accelerated rendering
* Native file format: Inventor files (.iv)
* GUI bindings available for various toolkits
  * SoQt for cross-platform applications using Qt
  * Others (SoXt, SoWin, Sc21)
* C++ SDK
  * Bindings to other languages available, e.g. *Pivy* for Python

Hello World

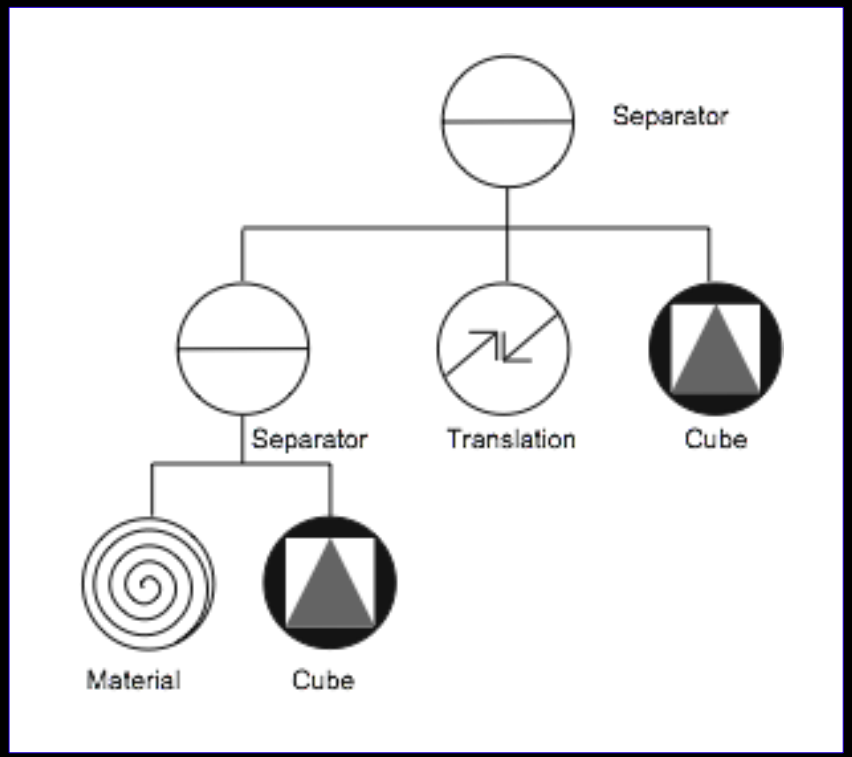
```
#include <Inventor/Qt/SoQt.h>
#include <Inventor/Qt/viewers/SoQtExaminerViewer.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoCube.h>

int main(int argc, char ** argv)
{
  QWidget * mainwin = SoQt::init(argc, argv, argv[0]);

  SoSeparator * root = new SoSeparator;
  root->ref();
  SoCube *cube = new SoCube;
  root->addChild(cube);

  SoQtExaminerViewer * eviewer = new
SoQtExaminerViewer(mainwin);
  eviewer->setSceneGraph(root);
  eviewer->show();
  SoQt::show(mainwin);
  SoQt::mainLoop();

  root->unref();
  delete eviewer;
  return 0;
}
```

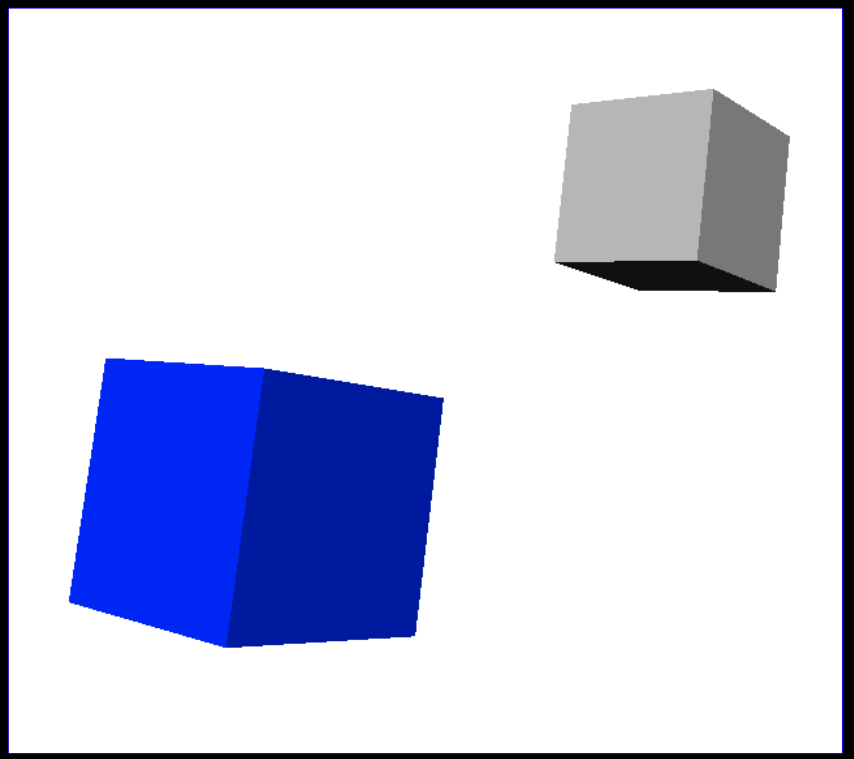
basic/helloworld.cpp


Examiner Viewer
Rotx
Roty
Dolly

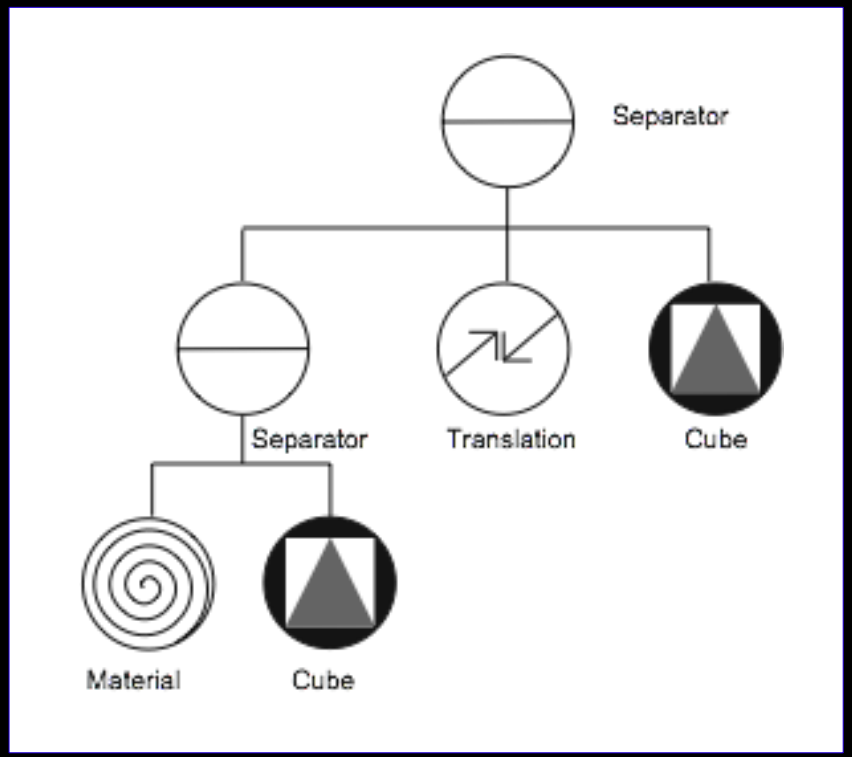
The Scenegraph

A very simple scene





A very simple scene




```
Separator {
  Group {
    Material {
      diffuseColor 0 0 1
    }
    Cube {
    }
  }
  Translation {
    translation 3 2 0
  }
  Cube {
  }
}
```

Inventor files vs. C++ code

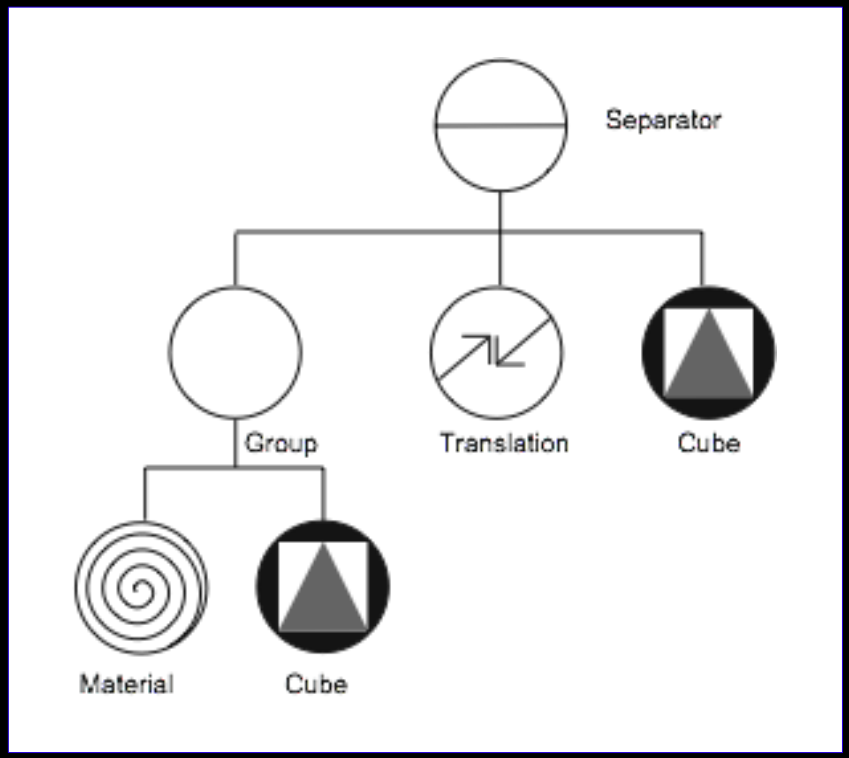
```
Separator {
  Group {
    Material {
      diffuseColor 0 0 1
    }
    Cube {
    }
  }
  Translation {
    translation 3 2 0
  }
  Cube {
  }
}
```

```
SoSeparator *root = new SoSeparator;
root->ref();

SoGroup *group = new SoGroup;
root->addChild(group);
SoMaterial *mat = new SoMaterial;
mat->diffuseColor.setValue(0,0,1);
SoCube *cube = new SoCube;
group->addChild(mat);
group->addChild(cube);

SoTransform *trans = new SoTransform;
trans->translation.setValue(3,2,0);
root->addChild(trans);

SoCube *cube2 = new SoCube;
root->addChild(cube2);
```

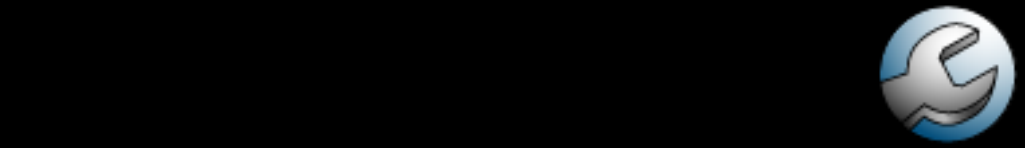
cpp-vs-iv/cpp.cpp
cpp-vs-iv/iv.iv

Scenegraph components

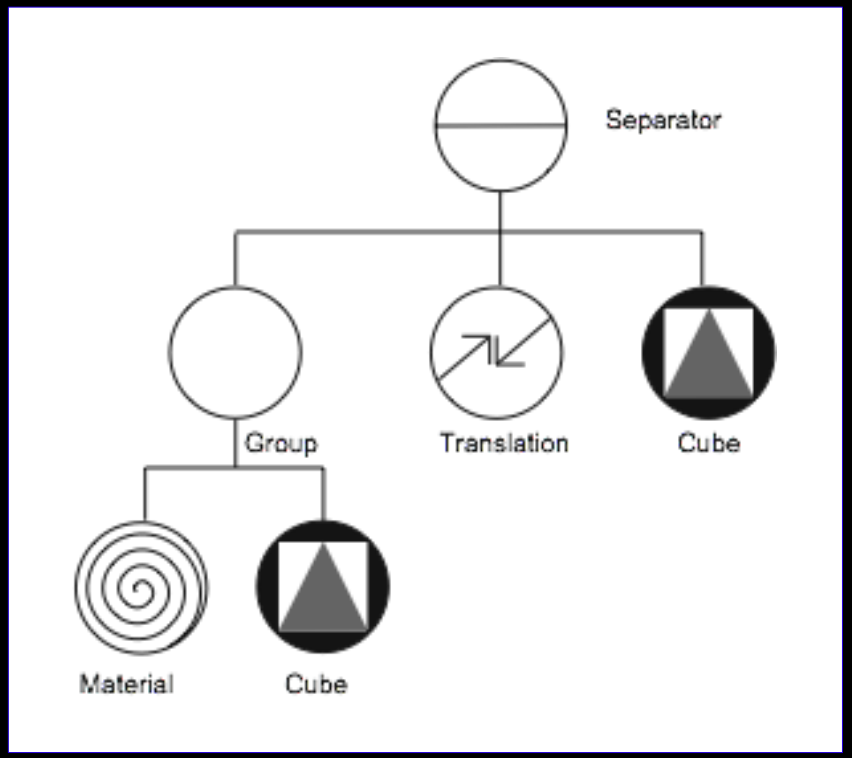

```
Separator {
  Group {
    Material {
      diffuseColor 0 0 1
    }
    Cube {
    }
  }
  Translation {
    translation 3 2 0
  }
  Cube {
  }
}
```



Scenegraph components



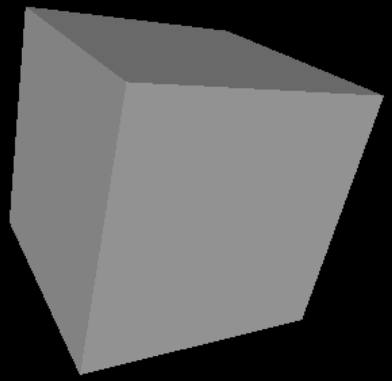
* Shape nodes
  * Geometry in the scene

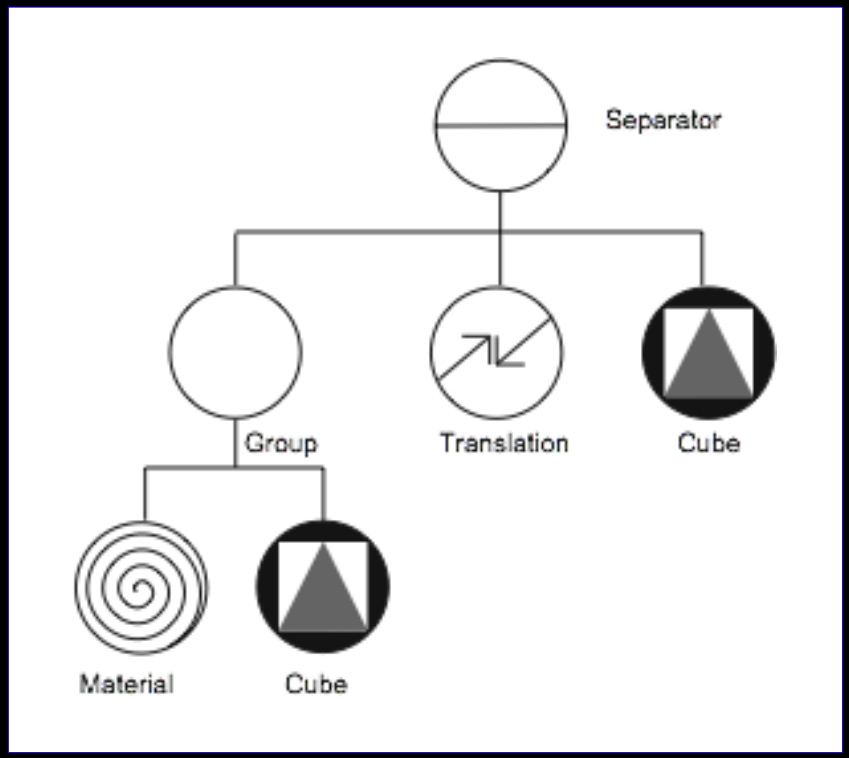


components/test.iv

Scenegraph components

* Shape nodes
* Geometry in the scene



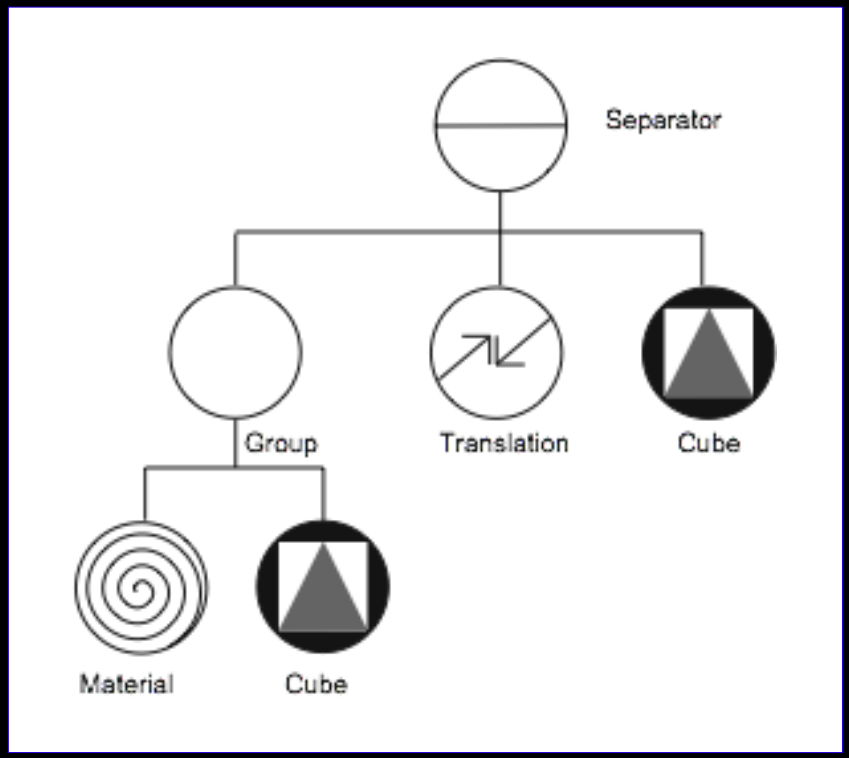


components/test.iv

Scenegraph components

* Shape nodes
* Geometry in the scene

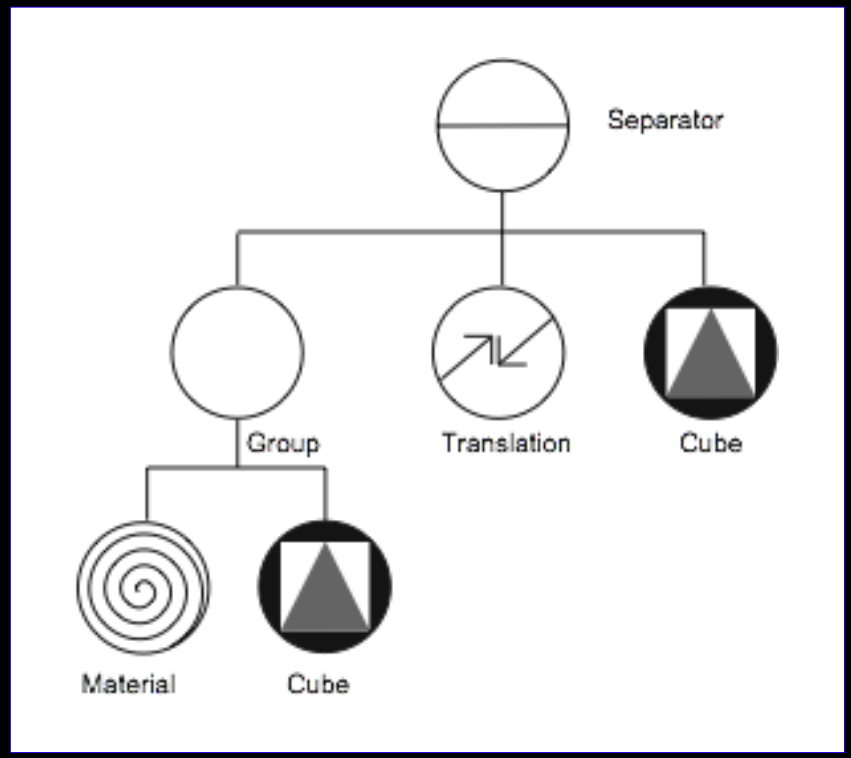
Hello World!



components/test.iv

Scenegraph components

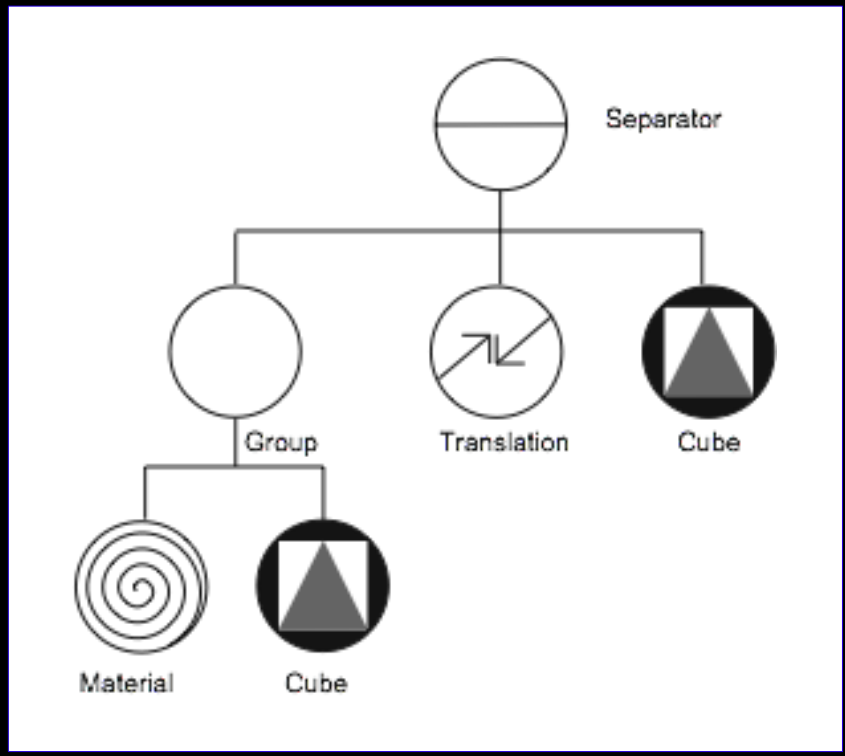
* Shape nodes
  * Geometry in the scene



components/test.iv

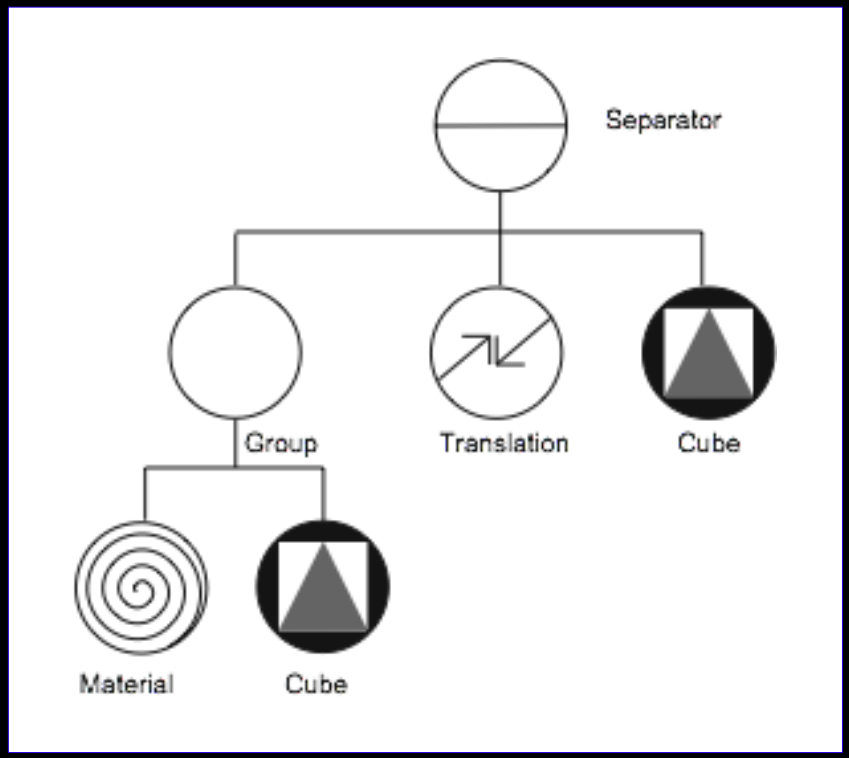
Scenegraph components

* Shape nodes
  * Geometry in the scene
* Property nodes
  * OpenGL state
  * Inventor state

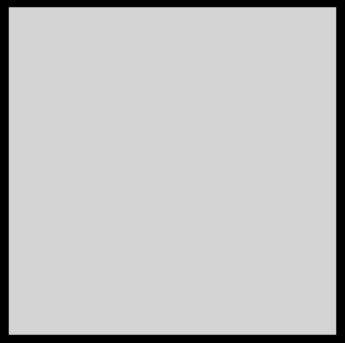


components/test.iv

Scenegraph components



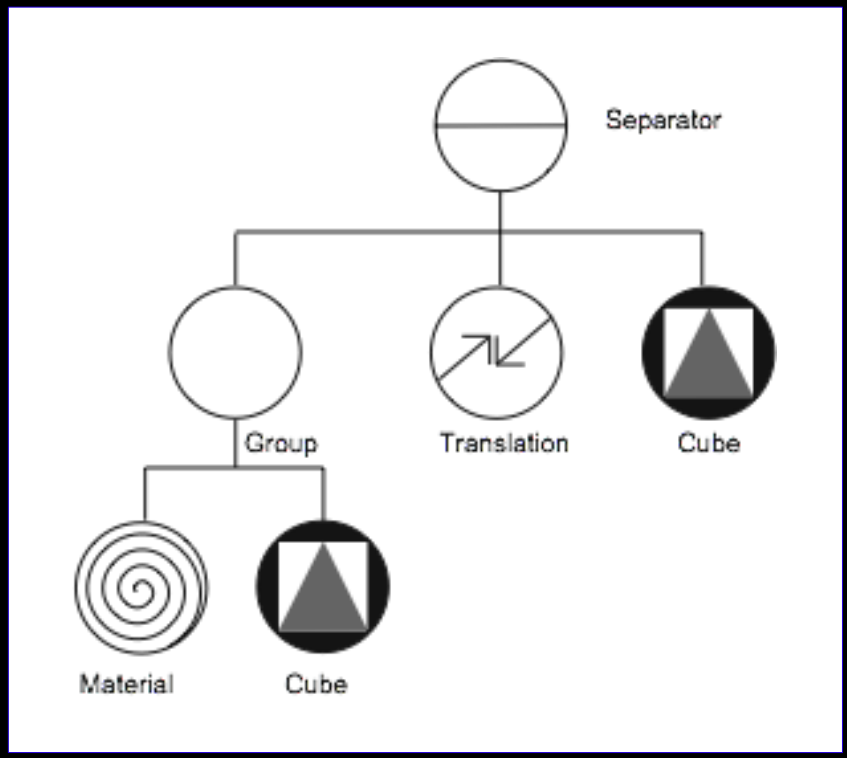
* Shape nodes
  * Geometry in the scene
* Property nodes
  * OpenGL state
  * Inventor state

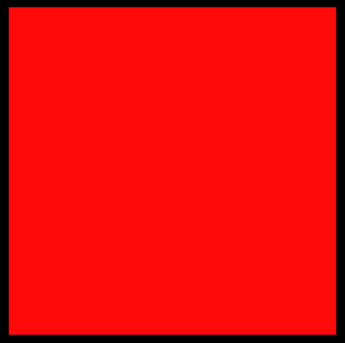


components/test.iv

Scenegraph components

* Shape nodes
  * Geometry in the scene
* Property nodes
  * OpenGL state
  * Inventor state

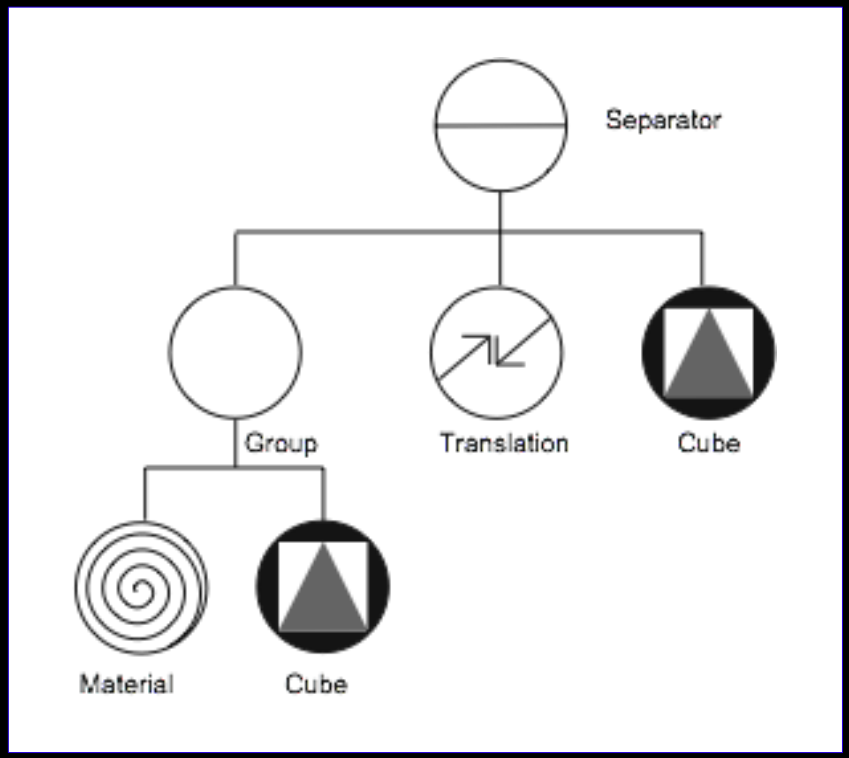


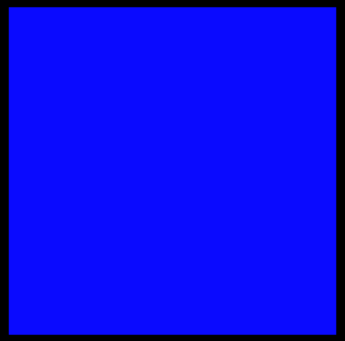


components/test.iv

Scenegraph components

* Shape nodes
  * Geometry in the scene
* Property nodes
  * OpenGL state
  * Inventor state

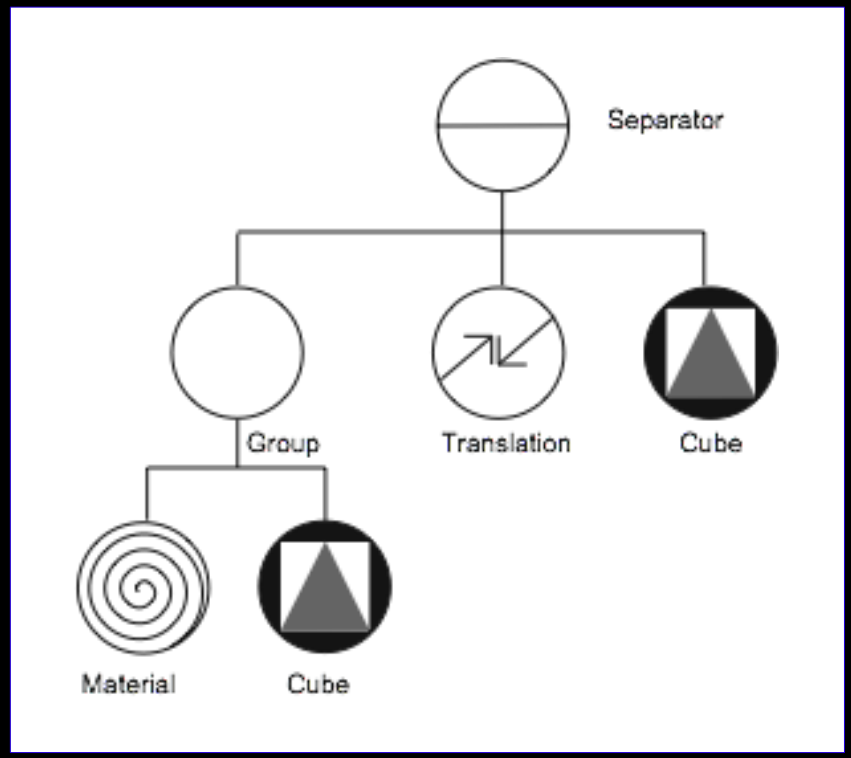




components/test.iv

Scenegraph components

* Shape nodes
  * Geometry in the scene
* Property nodes
  * OpenGL state
  * Inventor state
* Group nodes
  * Collect groups of nodes into a subtree to build a hierarchy

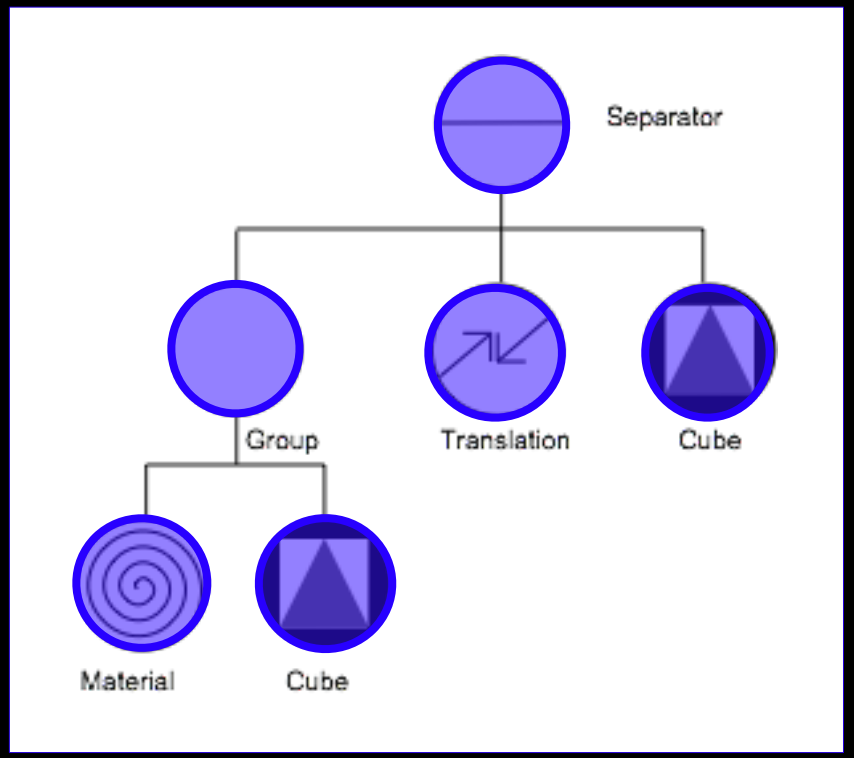


components/test.iv

Scenegraph rendering

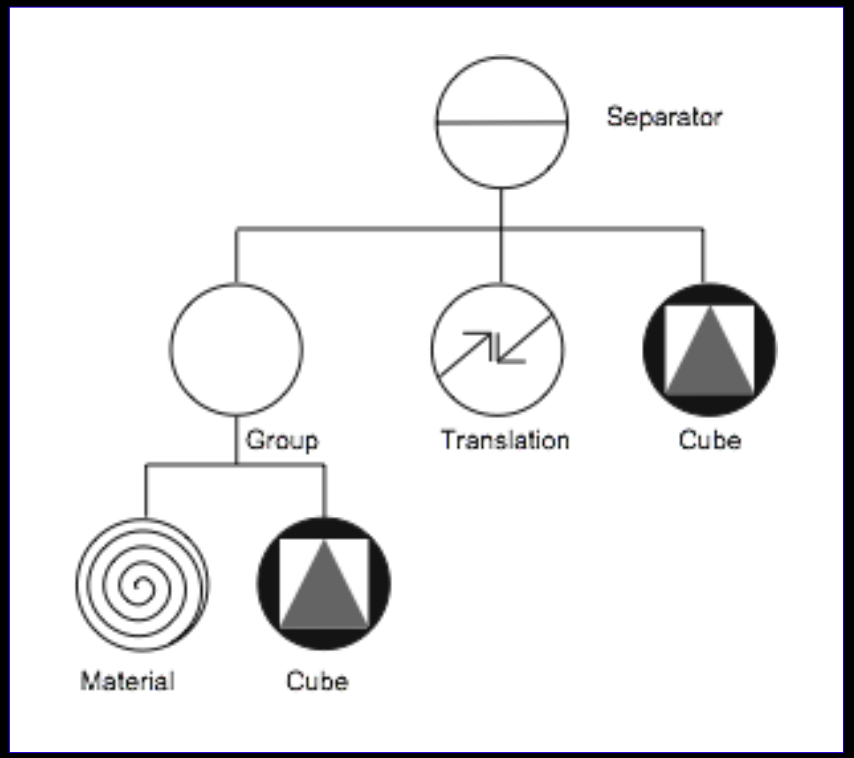
Actions

* Scenegraph is traversed from top to bottom and from left to right
* Each node can react to the action (behaviour depending on node type)
* Nodes inherit state from nodes visited earlier
* Rendering the scene is an action



The SoGLRenderAction – behaviours

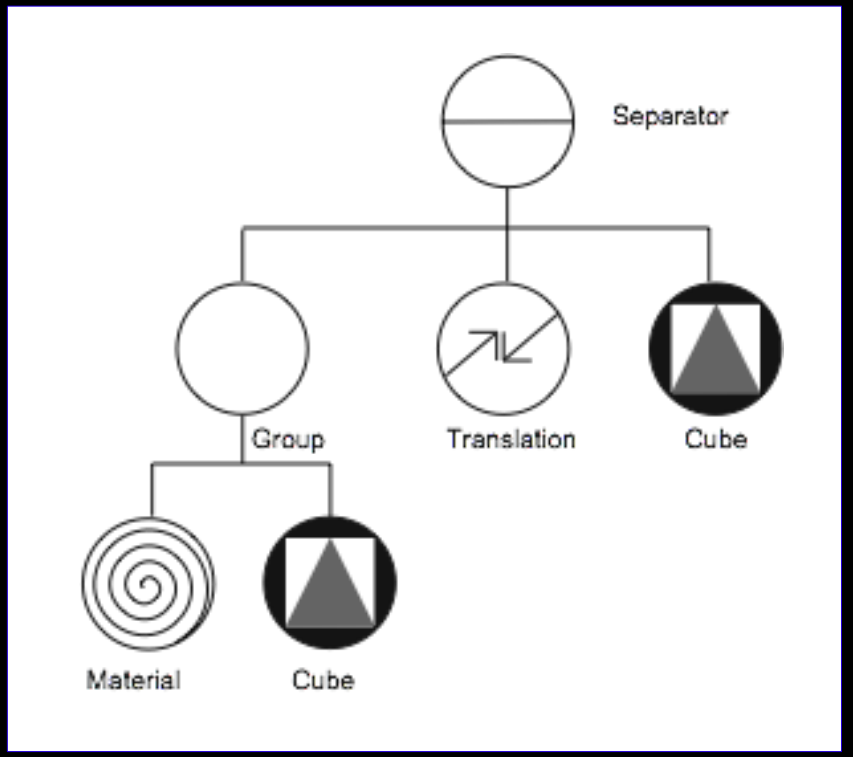
* Group nodes traverse their children
* Shape nodes draw their geometry
* Property nodes set the OpenGL state
  * usually replacing the previous state
  * except transformation (concatenated) and light sources (added)



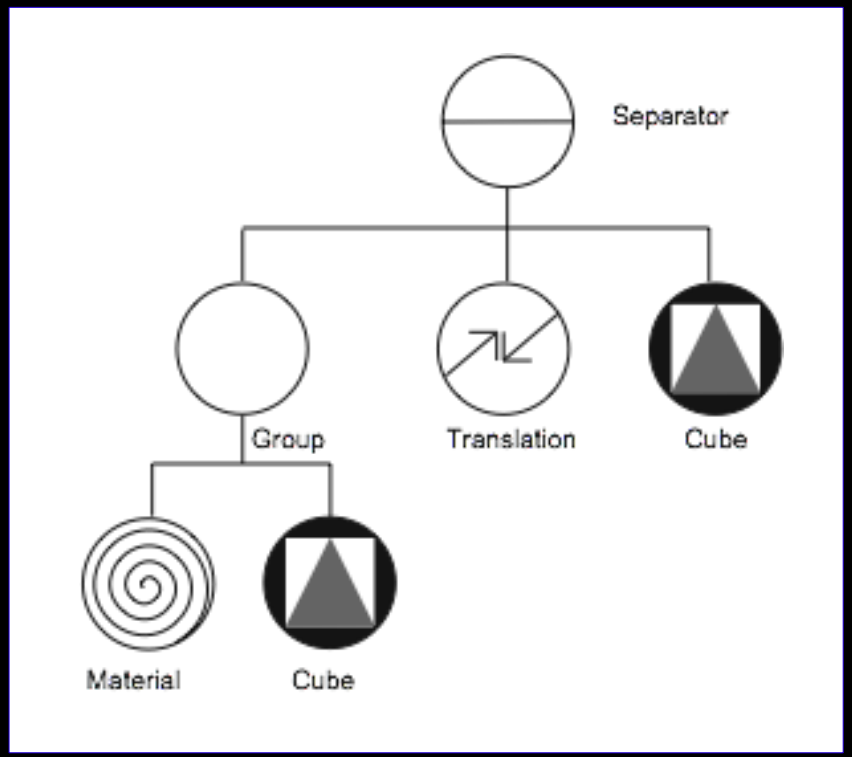
SoGroup vs. SoSeparator

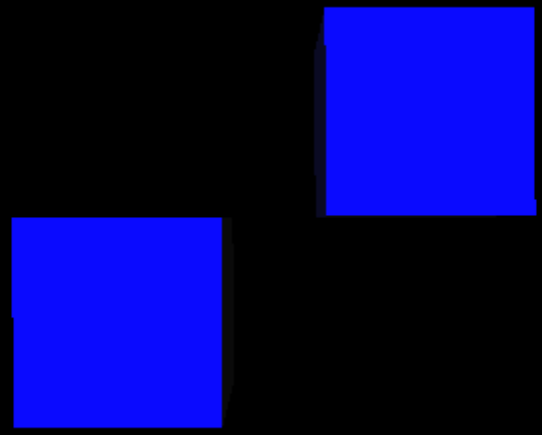
Group Nodes Revisited

- SoGroup: Nodes inherit state from to the left and above
- SoSeparator saves the state before traversing its children, and restores it when done.
- cf. OpenGL push and pop



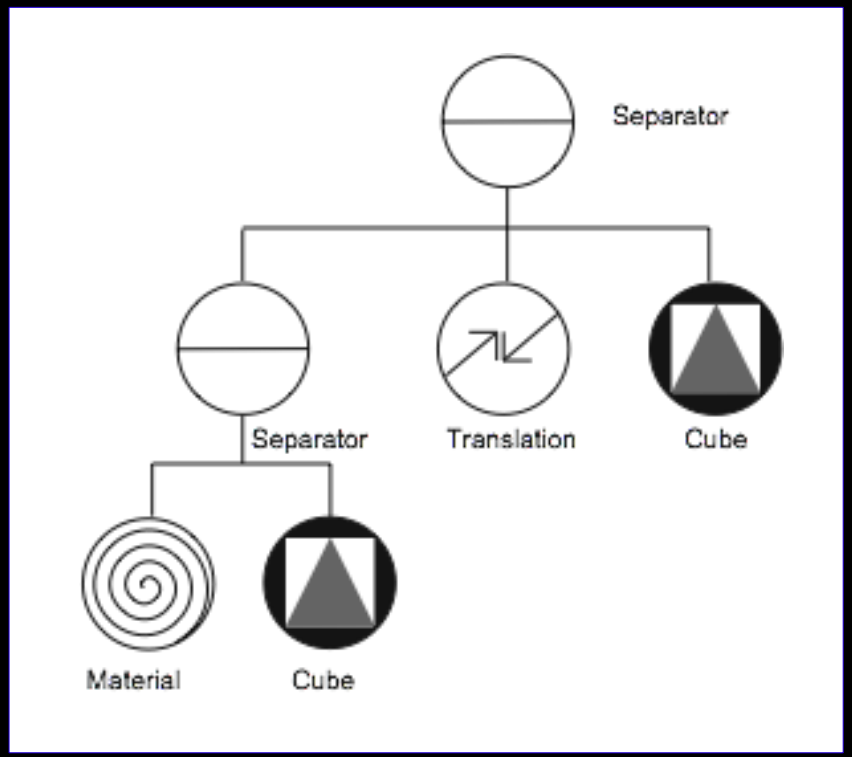
Group nodes revisited

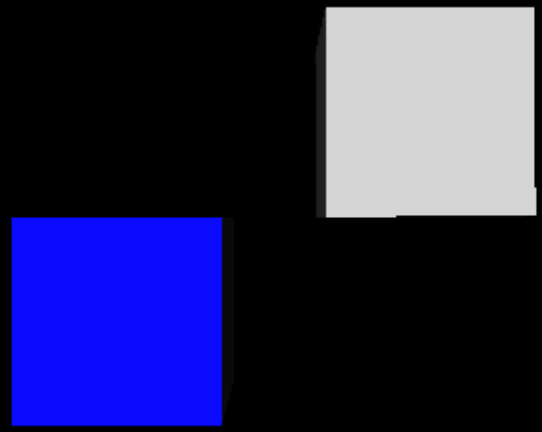




group-vs-separator/group.iv

Group nodes revisited





group-vs-separator/separator.iv

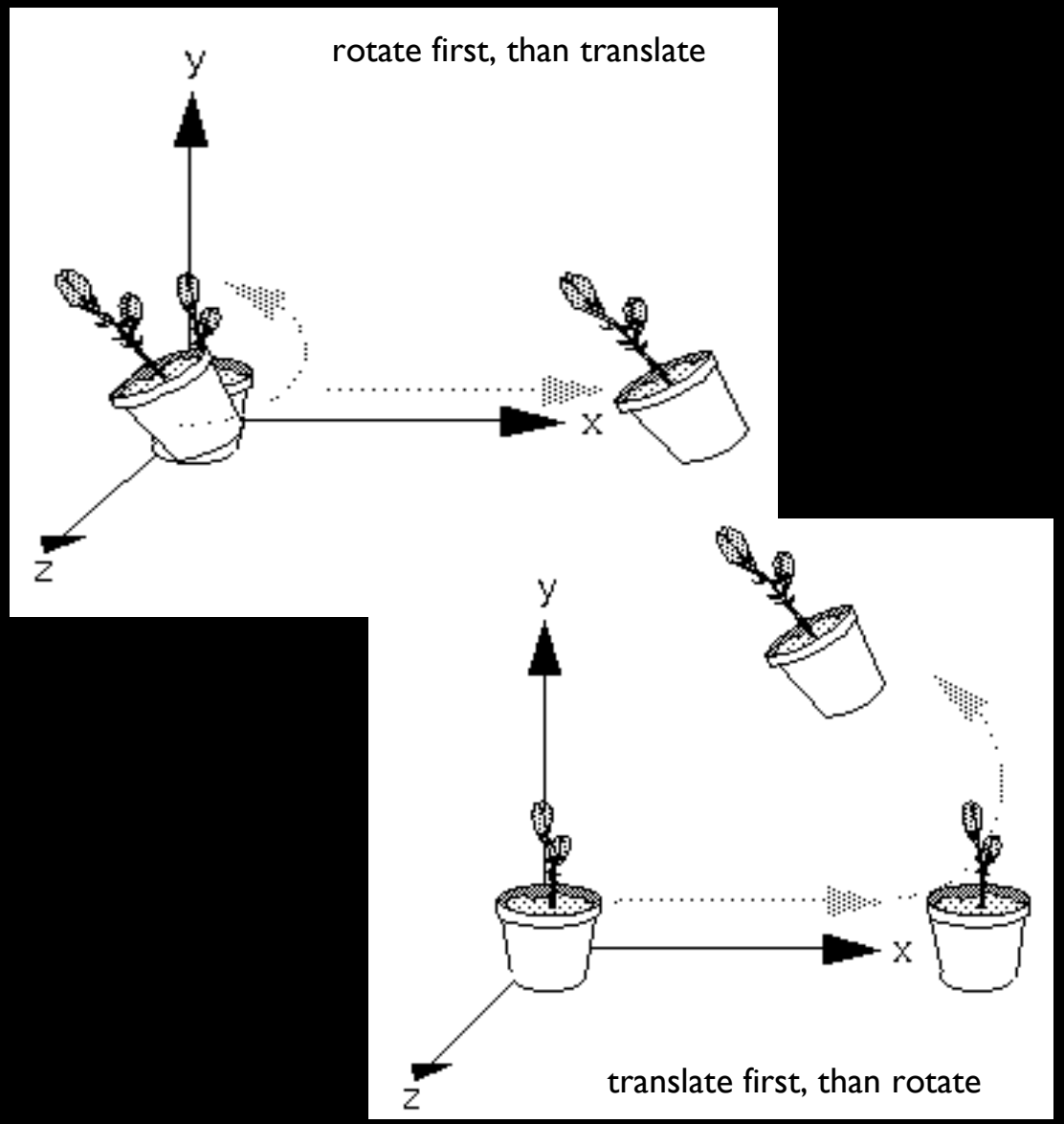
Transformations revisited

* Transformations are property nodes in the scenegraph
* Convenience nodes for basic transformations:
  * SoRotation
  * SoTranslation
  * SoScale
* It is also possible to specify the transformation matrix directly
  * SoMatrixTransform

Achtung! Here lies dragons...

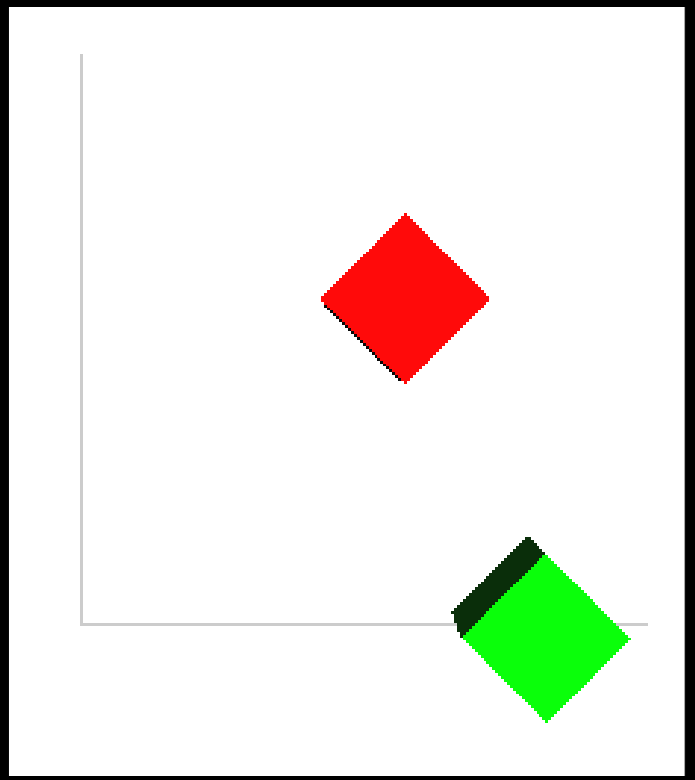
Transformations revisited

* Matrix multiplication is not commutative
  * The order of operations is important
  * Example: combined rotation and translation



Transformations revisited

```
#Inventor V2.1 ascii

Separator {
  Separator {
    Rotation { rotation 0 0 1 0.785 }
    Translation { translation 8 0 0 }
    Material { diffuseColor 1 0 0 }
    Cube { }
  }
  Separator {
    Translation { translation 8 0 0 }
    Rotation { rotation 0 0 1 0.785 }
    Material { diffuseColor 0 1 0 }
    Cube { }
  }
}
```



basic/03_transorder.iv

Summary:

* Hierarchical scene description in a *scenegraph*.
  * Group nodes, shape nodes, property nodes.
  * Created programmatically or via Inventor files.
* To render the scene, the scenegraph is traversed top-down and left-right by the *render action*.
  * State is inherited from nodes visited earlier.
  * The order of operations is important.

Qt integration

* Qt is a cross-platform GUI toolkit developed by Trolltech
  * Free Software edition available under the GPL
  * excellent solution for cross-platform development
* The SoQt library provides integration with the Qt toolkit
  * OpenGL setup
  * Event translation
* SoQt also provides a set of convenient viewer components
  * e.g. the SoQtExaminerViewer

Getting started

First steps

* Download Coin3D from http://www.coin3d.org.
* Build and install the Coin, SoQt, and simage libraries.
* Play around with the sample code.

* Documentation:
  * API documentation: http://doc.coin3d.org
  * The “Inventor Mentor” and “Inventor Toolmaker” books
  * coin-discuss open mailing list hosted by SIM
    https://www.coin3d.org/mailman/listinfo/coin-discuss/

Other Coin3D features...

* Easy way to read and write files
  * Including animations

io/readfile.cpp
io/writefile.cpp

* Field connections, notification mechanism
  * Automatic update based on scenegraph changes
* Interactive components
  * Draggers and manipulators

drama/dragger.cpp

* Picking
  * Select and identify objects in the scene
* Event handling
* Sensors

Questions?

Legal