Lecture 4: Tuesday, 16 September 1997

Administrative:
- Asst 1 solutions posted on web
- Asst 2 due this Friday at 5pm
  - Analogous to Asst 1; info posted online
- Athena stuff
  - `setup 6.837` for timely announcements
  - (subscribes to 6.837 `zephyr` instance)
- Theory review: **W (tomorrow) 7-9 in 34-101**
- My office hours: today, starting at 4pm, NE43-208

Today:
- Asst 1 solution: First octant, All octants
- Object, World Space Modeling (H&B S7.3,10.1,11.5)
- Using Inventor on Athena SGI machines
- Assignment 3A out (due Friday 26 September)
  - Inventor object modeling; C,C++ programming !

Thursday:
- Graphics Pipeline; More 3D Xforms; Perspective

Assignment 1 Solution: First Octant

```
// drawLine (int x1, int y1, int x2, int y2);
// handles first octant:  dx,dy > 0; dy <= dx
int x, y, d;
int twody =  ( y1-y2 ) << 1;   // slope:  dy/dx * 2dx (flipped)
int twodx =  ( x2-x1 ) << 1;   // unit increment:  1 * 2dx

d = -twodx >> 1;  // initially, segment is half a pixel below M
y = y1;
for ( x = x1; x <= x2; x++ ) {
  setPixel (x, y);
  d += twody;
  if ( d >= 0 ) {
    d -= twodx;
    y--;  // decrement y due to screen flip
    }
}
```


Assignment 1 Solution: All Octants

```
// drawLine (int x1, int y1, int x2, int y2);
// handles all 8 octants, all special cases
int x, y, dx, dy, npix;
int xinc, yinc, xup, yup; // advance step; error step
int e, einc, emax;        // error; err inc; err limit

dx = x2-x1;  dy = y2-y1;  // x,y deltas and sign bits
xinc = (dx >= 0) ? 1 : -1;  yinc = (dy >= 0) ? 1 : -1;

if ( (xinc * dx) > (yinc * dy) ) { // loop across x
  npix = (dx * xinc) + 1;          // note:  non-neg
  emax = dx * xinc;  einc = (dy * yinc) << 1;
  xup = 0;  yup = yinc;  yinc = 0; // adv, err steps
  }
else {                             // loop across y
  npix = (dy * yinc) + 1;          // note:  non-neg
  emax = dy * yinc;  einc = (dx * xinc) << 1;
  yup = 0;  xup = xinc;  xinc = 0; // adv, err steps
  }

x = x1; y = y1; e = 0;    // error initially zero
while ( npix-- > 0 ) {
  setPixel (x, y);
  x += xinc; y += yinc; e += einc; // advance step
  if ( e > emax ) {
    x += xup; y += yup; e -= emax << 1; // error step
    }
  }
```


Object and World Space Modeling

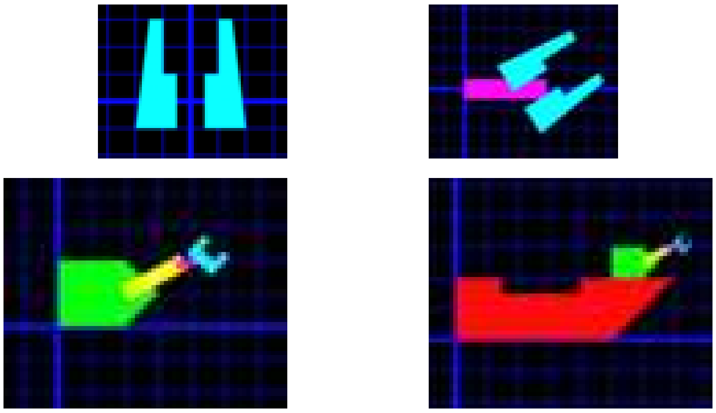
Object space a sort of "workbench" where
components are assembled, combined

World space is a "canvas" on/in which
finished objects are placed

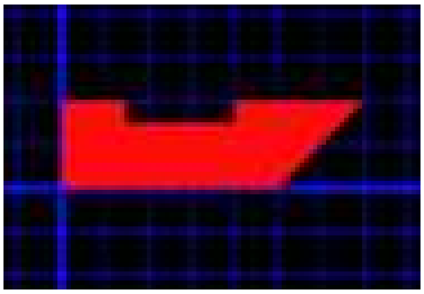
But: often useful to view both ways

Usually, required transformations, ordering obvious

Example: Boat, sub, articulated grasper arm, treasure



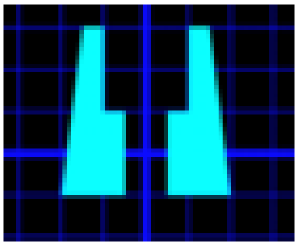
Modeling the Boat



```
# boat, version 1
Switch { DEF boat Separator {
    Material { diffuseColor 1 0 0 } # RGB (red)
    Coordinate3 {
        point   [  0    0    0,
                   5    0    0,
                   7    2    0,
                   4    2    0,
                   4    1.5  0,
                   1.5  1.5  0,
                   1.5  2    0,
                   0    2    0]
    } # Coordinate3
    IndexedFaceSet { coordIndex [ 0, 1, 2, 3, 4, 5, 6, 7, -1 ] }
} } # Switch
```


Modeling Jason, with Robot Arm

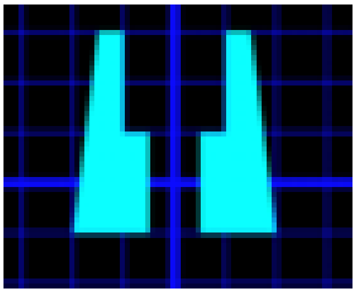
Proceed "bottom up": Grasper, Forearm, Arm, Sub



```
# grasper
Switch { DEF grasper Separator {
    Material { diffuseColor 0 1 1 }
    Coordinate3 {
        point   [ -2    -1    0,
                  -0.5  -1    0,
                   0.5  -1    0,
                   2    -1    0,
                   1.5   3    0,
                   1     3    0,
                   1     1    0,
                   0.5   1    0,
                  -0.5   1    0,
                  -1     1    0,
                  -1     3    0,
                  -1.5   3    0 ]
    } # Coordinate3
    IndexedFaceSet { coordIndex [ 0, 1, 8, 9, 10, 11, -1,
                                  2, 3, 4, 5,  6,  7, -1 ] }
} } # Switch
```


Use of Symmetry

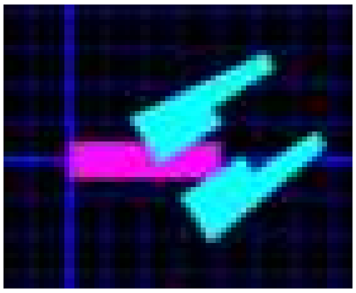
Can exploit symmetry even within grasper DEF:



```
# grasper alternative
Switch { DEF grasper Separator {
    Material { diffuseColor 0 1 1 }
    Coordinate3 {
        point   [ -2    -1    0,
                  -0.5  -1    0,
                  -0.5   1    0,
                  -1     1    0,
                  -1     3    0,
                  -1.5   3    0 ]
    } # Coordinate3
    FaceSet { numVertices [ 6 ] }
    Scale { scaleFactor -1 1 0 } # why ?
    FaceSet { numVertices [ 6 ] }
} } # Switch
```


Forearm

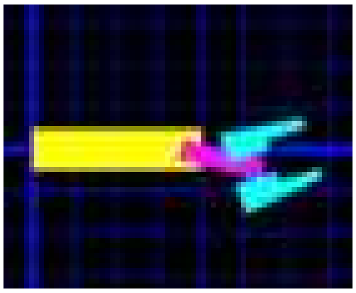
Note `USE` of grasper (rotation, translation)



```
# forearm
Switch { DEF forearm Separator {
    Material { diffuseColor 1 0 1 } # RGB (magenta)
    Coordinate3 {
        point   [  0   -0.5    0,
                   4   -0.5    0,
                   4    0.5    0,
                   0    0.5    0 ]
    } # Coordinate3
    IndexedFaceSet { coordIndex [ 0, 1, 2, 3, -1 ] }

    # instance grasper
    Translation { translation 3.5 0 0 }
    RotationXYZ { axis Z angle -1 }
    USE grasper
} } # Switch
```

Arm

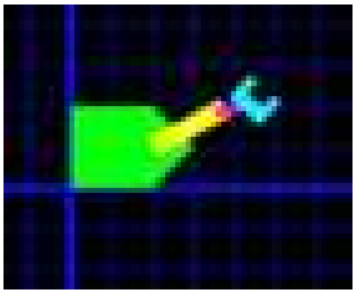
Note USE of forearm (which USEs grasper)



```
# arm
Switch { DEF arm Separator {
    Material { diffuseColor 1 0.5 0 }
    Coordinate3 {
        point   [  0   -0.5    0,
                   4   -0.5    0,
                   4    0.5    0,
                   0    0.5    0 ]
    } # Coordinate3
    IndexedFaceSet { coordIndex [ 0, 1, 2, 3, -1 ] }

    # instance forearm
    Translation { translation 3.5 0 0 }
    Scale { scaleFactor 0.5 0.5 0 }
    RotationXYZ { axis Z angle -0.25 } # joint angle wrt. mount
    USE forearm
} } # Switch
```

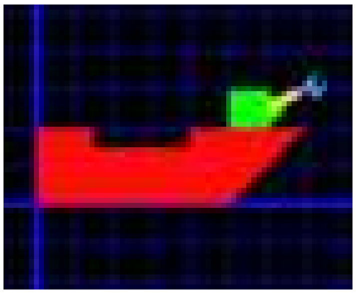

Jason Sub

Note USE of arm (which USEs forearm, which USEs...)



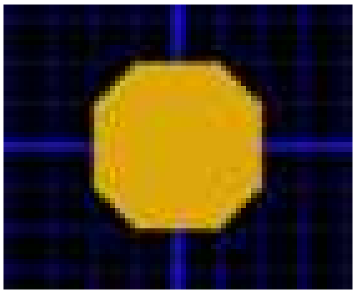
```
# jason sub
Switch { DEF jason Separator {
    Material { diffuseColor 0 1 0 }
    Coordinate3 {
        point   [  0    0    0,
                   2    0    0,
                   3    1    0,
                   2    2    0,
                   0    2    0 ]
    } # Coordinate3
    IndexedFaceSet { coordIndex [ 0, 1, 2, 3, 4, -1 ] }

    # instance entire arm
    Translation { translation 2 1 0 }
    Scale { scaleFactor 0.5 0.5 0 }
    RotationXYZ { axis Z angle 0.5 } # joint angle wrt. mount
    USE arm
} } # Switch
```


Placing Sub on Boat



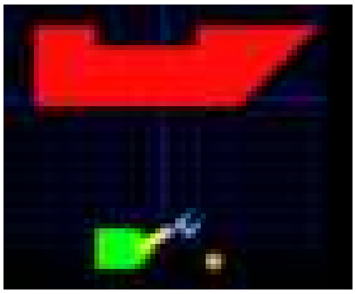
```
# boat, version 2
Switch { DEF boat Separator {
    Material { diffuseColor 1 0 0 }
    Coordinate3 {
        point   [  0    0    0,
                   5    0    0,
                   7    2    0,
                   4    2    0,
                   4    1.5  0,
                   1.5  1.5  0,
                   1.5  2    0,
                   0    2    0]
    } # Coordinate3
    IndexedFaceSet { coordIndex [ 0, 1, 2, 3, 4, 5, 6, 7, -1 ] }

    # instance jason sub wrt boat
    Translation { translation 5 2 0 }  # on board (up)
    Scale { scaleFactor 0.5 0.5 0 }
    USE jason
} } # Switch
```


Treasure: Gold Coin(s)



```
# coin
Switch { DEF coin Separator {
    Material { diffuseColor 0.8 0.6 0 } # gold
    Coordinate3 {
        point   [ -1    -2    0,
                   1    -2    0,
                   2    -1    0,
                   2     1    0,
                   1     2    0,
                  -1     2    0,
                  -2     1    0,
                  -2    -1    0 ]
    } # Coordinate3
    FaceSet { numVertices [ 8 ] }
} } # Switch
```

Placing Boat, Coin in Ocean



```
# boat, version 3
Switch { DEF boat Separator {
    ...
    # instance jason sub wrt boat
    Translation { translation 1.5 -4 0 }   # diving (down)
    Scale { scaleFactor 0.5 0.5 0 }
    USE jason
} } # Switch
# place boat on sea surface
Separator { Translation { translation -8 -0.5 0 }
            Scale { scaleFactor 2.5 2.5 0 }
            USE boat }
# place coin on sea bottom
Separator { Translation { translation 3 -10 0 }
            Scale { scaleFactor 0.25 0.25 0 }
            USE coin }
```

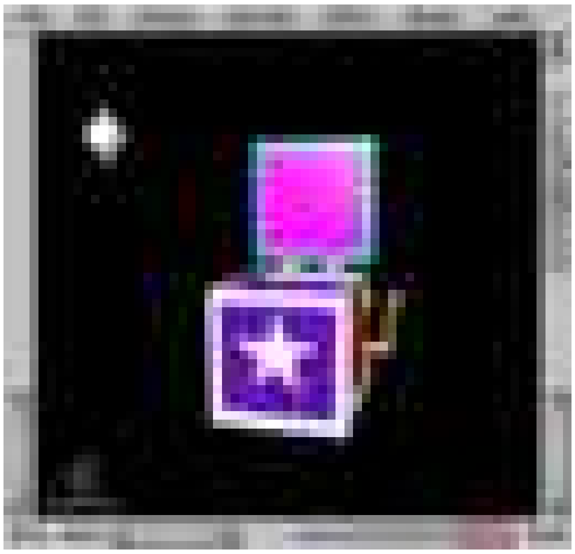
How to get grasper to pick up coin?

SGI Inventor

Modeling file format
Several useful viewers/editors
Extensible (C++) tool suite & dev. env.

Getting started I

SceneViewer
  Manipulate, edit, light Inventor models



`/usr/demos/Inventor/SceneViewer foo.iv`
  `/usr/share/data/models/*.iv`
  `/usr/share/data/models/simple/*.iv`

Icons

Default: 3D Viewer (hand icon)

Trackball, Walk, Fly, Plane view
Also Select, Help, Home, Set Home,
  Center, Set Center, Ortho/Perspective

Manipulators

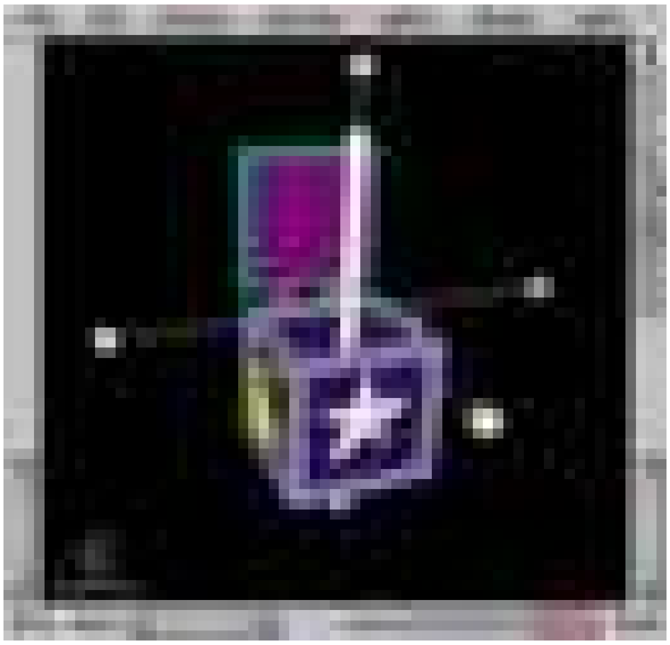
Example Manipulator: Trackball

Manipulators

Example Manipulator: Tab Box

Manipulators

Example Manipulator: Jack



Selection

Pick mode (arrow icon); select

Selection

Edit → Pick Parent

Selection

Edit → Pick Parent (Edit → Pick All)

Light Creation/Editing

Lights → Create {Point, Dir, Spot}

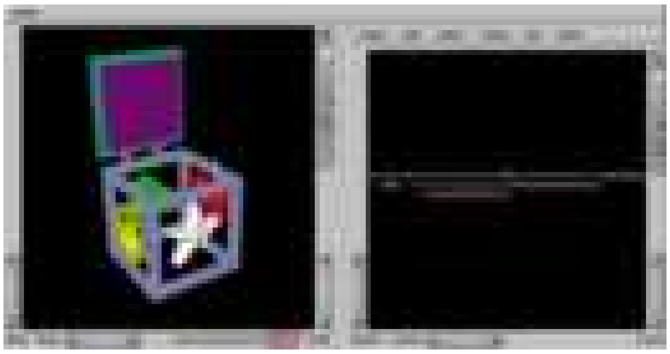
Material / Color Editing

Editors → {Material, Color}

Enabled only when some entity is selected

Getting started II

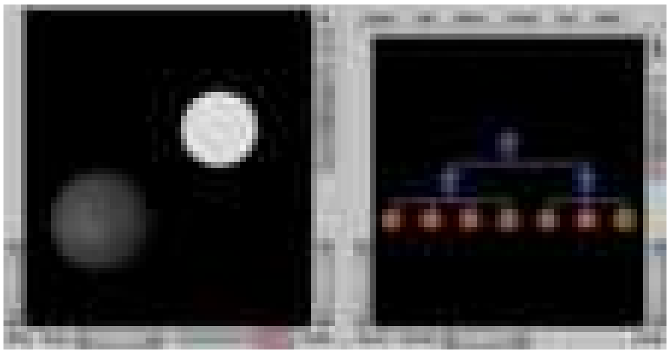
Inventor Scene Graphs
  Hierarchical model descriptions
  Primitives, transforms, lights, materials...
`/usr/demos/Inventor/gview foo.iv`



  `/usr/share/data/models/simple/*.iv`

Inventor Introduction

Scene graph (DAG), comprised of three node types:
  Shape: 3D geometric objects
  Transform: Affect current transformation
  Property: Appearance, texture, etc.
  Group: Collection of subgraphs



Traversal state:
  Maintained during (modified) pre-order traversal
    current node, children (left to right), recursively
    some nodes have effects in both directions!

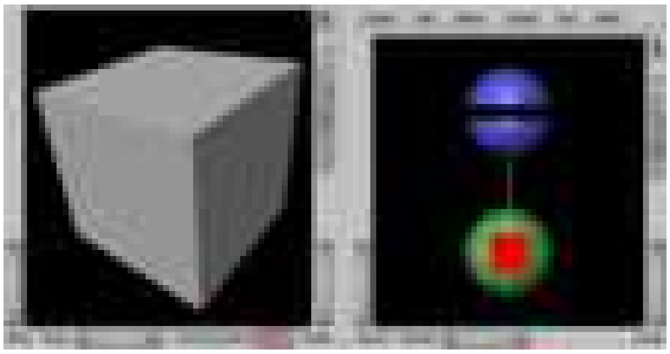
Inventor Resources

Example Files & Documentation
  `athena:/usr/share/data/models/`
  `athena:/usr/share/src/Inventor/`
  `athena% man SoCube`, `man SoTranslation`, etc.
    (should have `/usr/share/catman` in `$MANPATH`)
  On reserve at LCS reading room NE43-113
    Open Inventor C++ Reference Manual
      (The “green book”)
    Inventor Mentor
      (The “orange book”)
  On Web: Inventor Quick Reference
    Top-level link from course page

Built-In Geometric Primitives

First, trivial example:

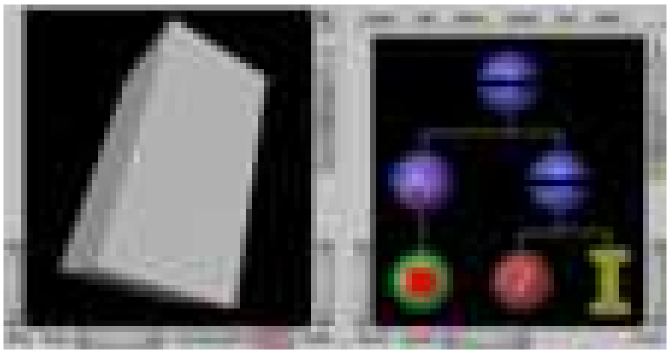
```
#Inventor V2.1 ascii
Cube {}
```



Primitive attributes:
  Default W,H,D (for cubes, spheres, cones, etc.);
  Default Color; Material; &c
Traversal state (all defaulted for now):
  Xform; Lights; Material; &c.

Definition & Instantiation

```
Switch { DEF unitcube
    Cube {}
} # Switch

Separator {
    Scale { scaleFactor 1 1 2 }
    USE unitcube
} # Separator
```



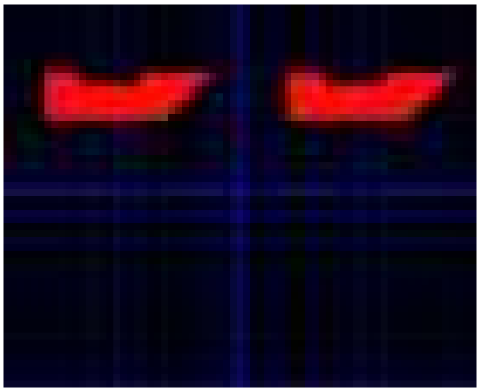
Note: `gview` comes up with nodes "closed"
To open, mouse-select, then use Alt-O(pen)

Persistence of Traversal State

```
# place two boats on surface (surprising result)
Separator {
    Translation { translation -8 3 0 }
    USE boat
    Translation { translation 2 3 0 }
    USE boat
    }
```

Separator Nodes

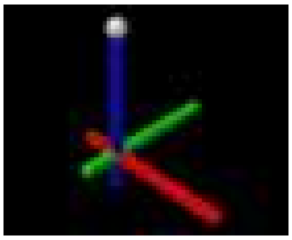
```
# place two boats on surface (expected result)
Separator {
    Translation { translation -8 3 0 }
    USE boat
    }
Separator {
    Translation { translation 2 3 0 }
    USE boat
    }
```



Pushes (saves) state when encountered ↓
Pops (restores) state when encountered ↑

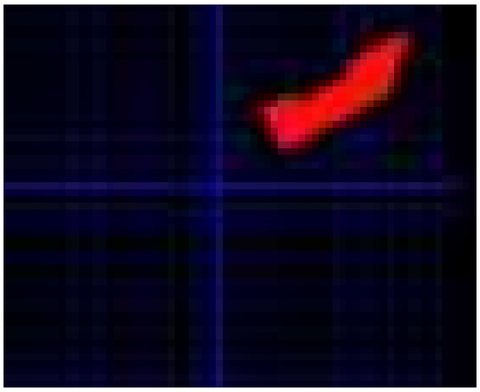
Useful scene graph: Axes (`coordaxes.iv`)

```
Separator {
    Translation { translation 2 0 0 }
    Scale { scaleFactor 4.0 0.25 0.25 }
    Material { diffuseColor 1 0 0 }
    Cube {}
} # Separator
Separator {
...
    Material { diffuseColor 0 1 0 }
    Cube {}
} # Separator
...
Separator {
    ... Sphere {}
} # Separator
```



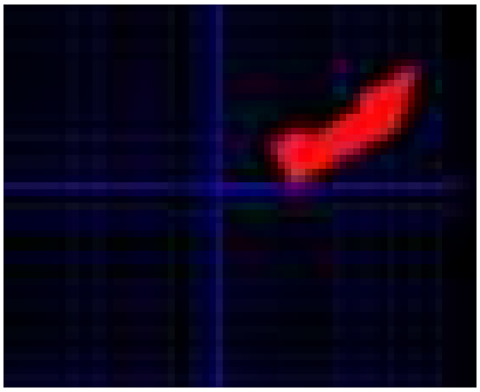
Composing Transformations

```
...
Separator {
    Rotation { rotation 0 0 1 0.5 } # axis, angle
    Translation { translation 3 0 0 }
    USE boat
} # Separator
```



Composing Transformations

```
...
Separator {
    Translation{ translation 3 0 0 }
    Rotation { rotation 0 0 1 0.5 } # axis, angle
    USE boat
} # Separator
```

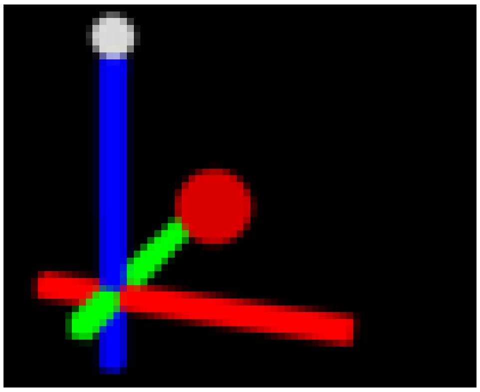


Conclusion?

Intrinsic Material

To color with "intrinsic material":

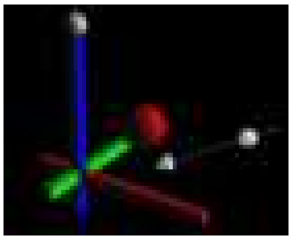
```
...
LightModel { model BASE_COLOR }
File { name "coordaxes.iv" }
Separator {
    Material { diffuseColor 0.8 0.0 0.0 }
    Sphere {}
} # Separator
```



Using a Lighting Model

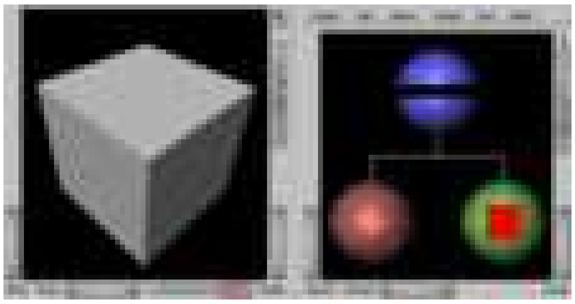
To shade with "lighting model":

```
...
LightModel { model PHONG }
DirectionalLight {
    intensity   1
    color       1 1 1
    direction  -1 -1 -1 # "shining" this dir
    }
Separator {
    Material {
        diffuseColor 0.25 0.25 0.25
        specularColor 0.8 0.8 0.8
        shininess 0.9
    } # Material
    Sphere {}
} # Separator
```



Polyhedral Objects

```
Separator {
    Coordinate3 {
        point   [ -1  1  1,   -1 -1  1,    1 -1  1,    1  1  1,
                  -1  1 -1,   -1 -1 -1,    1 -1 -1,    1  1 -1 ]
    } # Coordinate3
    IndexedFaceSet { # multiple faces
        coordIndex [ 0, 1, 2, 3, -1, # vertex ids, -1 ends face
                     3, 2, 6, 7, -1,
                     7, 6, 5, 4, -1,
                     4, 5, 1, 0, -1,
                     0, 3, 7, 4, -1,
                     1, 5, 6, 2, -1 ]
    } # IndexedFaceSet
} # Separator
```

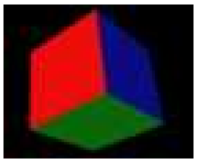


Specifying Face Normals, Colors

```
# primarycube.iv
Separator {
    # cube vertices
    ...
    # face normals, per-face
    Normal {
        vector  [
                 -1  0  0,
                  1  0  0,
                  0 -1  0,
                  0  1  0,
                  0  0 -1,
                  0  0  1, ]
    } # Normal
    # assign normals per face
    NormalBinding { value PER_PART }
    # face colors, per-face
    BaseColor { # set diffuse colors
        rgb     [ 1 0 0,  # 0, red
                  0 1 0,  # 1, green
                  0 0 1,  # 2, blue
                  0 1 1,  # 3, ~red
                  1 0 1,  # 4, ~green
                  1 1 0, ]# 5, ~blue
    } # BaseColor
    ...
```

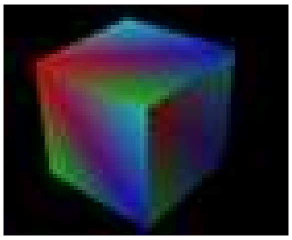


```
    ...
    # assign color ids per face
    MaterialBinding { value PER_FACE_INDEXED }
    IndexedFaceSet {
        coordIndex      [ # define faces with vertex ids
                          4, 5, 1, 0, -1, # -x
                          3, 2, 6, 7, -1, # +x
                          1, 5, 6, 2, -1, # -y
                          0, 3, 7, 4, -1, # +y
                          7, 6, 5, 4, -1, # -z
                          0, 1, 2, 3, -1,]# +z
        materialIndex   [ # material ids per face
                          3, 0, 4, 1, 5, 2, -1 ]
    } # IndexedFaceSet
} # Separator
```



Normals, Colors Per Vertex

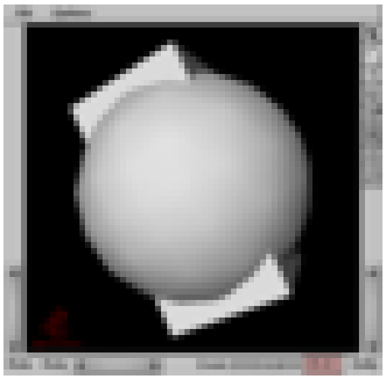
```
...
    MaterialBinding { value PER_VERTEX_INDEXED }
...
       materialIndex    [ # color ids; per face, then per vertex
                          0, 1, 2, 3, -1, # -x
                          0, 1, 2, 3, -1, # +x
                          0, 1, 2, 3, -1, # -y
                          0, 1, 2, 3, -1, # +y
                          0, 1, 2, 3, -1, # -z
                          0, 1, 2, 3, -1  # +z
                        ]
```



See `% man SoNormalBinding` for normals

File Inclusion

```
#Inventor V2.1 ascii
# file cubesphere.iv, a cube and sphere.
Switch {
    DEF mycube Separator { Cube { height 4 } }
    DEF mysphere Separator { Sphere { radius 2. } }
}

# top-level Separator is optional
USE mycube
USE mysphere

# end of cubesphere.iv
```

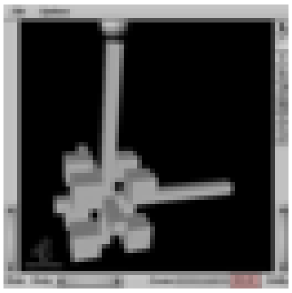


File Inclusion & Instancing

```
#Inventor V2.1 ascii
# instance 2 cubespheres.
Switch {
    DEF cubesphere # call it what you wish
    File { name "cubesphere.iv" }
}
Separator {
    Translation { translation 2 0 0 }
    Scale { scaleFactor 2 2 1 }
    USE cubesphere
    }
Separator {
    Translation { translation -2 0 0 }
    Scale { scaleFactor 1 1 2 }
    USE cubesphere
    }
```

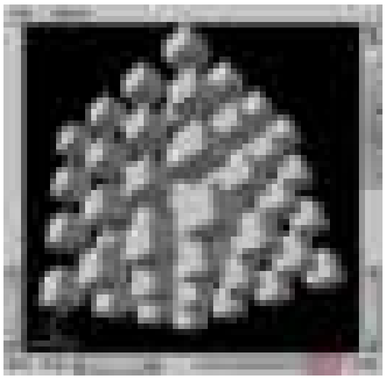
Hierarchical Instancing

```
#Inventor V2.1 ascii
File { name "coordaxes.iv" }
Switch { DEF fourcubes Separator {
  Separator {
    Translation { translation -1 -1 -1 }
    Scale { scaleFactor 0.5 0.5 0.5 }
    Cube {}
  } # Separator
  ...
  Separator {
    Translation { translation  1 -1 -1 }
    Scale { scaleFactor 0.5 0.5 0.5 }
    Cube {}
  } # Separator
} }
USE fourcubes
Translation { translation 0 0 2 }
USE fourcubes
```

Hierarchical Instancing

```
#Inventor V2.1 ascii
Switch {
  DEF eightcubes
  File { name "eightcubes.iv" }
}
Switch { DEF fourgroups Separator {
  Separator {
   Translation { translation -1 -1 -1 }
   Scale { scaleFactor 0.5 0.5 0.5 }
   USE eightcubes
  } # Separator
  ...
  Separator {
   Translation { translation  1 -1 -1 }
   Scale { scaleFactor 0.5 0.5 0.5 }
   USE eightcubes
  } # Separator
} }
USE fourgroups
Translation { translation 0 0 2 }
USE fourgroups
```

Modeling Complex Structures

Use simple hierarchy when:
  natural "ownership" relation exists
  repetitive structure occurs
  simple shapes reoccur
Some entities require generalized hierarchy
  graphs, networks, etc.
Some entities have no hierarchical structure
  induce one with *spatial data structures*

Review: Useful Inventor Facts

Quick reference for all Inventor node types:
 `http://www.sgi.com/Technology/Inventor/PostScript/quickRef.ps`
Every file must start with `#Inventor V2.1 ascii`
Comments begin with `#`, continue to end of line
File inclusion via `File { name "file.iv" }`
  Define name in **including**, not included, file
Separator nodes save and restore state
All rotations in axis-angle form (angles in radians)
Use gview:Node→Create to add primitives
SceneViewer:File→Save Environment saves view, lights
Note: Use (shorthand) `Transform` carefully:

```
Transform {
    translation 0 0 6.5
    rotation 1 1 1 1.5707963
    scaleFactor 0.5 0.5 0.5 # fixed order
} # Transform
```

Use `ShapeHints { faceType UNKNOWN_FACE_TYPE }`
if you specify non-convex polygons (note scope)

Assignment 3A – Object Modeling

At least four "things going on" today:
- Defining a single object in Inventor (`.iv` files)
- Viewing such objects with Inventor viewing tools
- Inventor's internal representation for SceneGraphs
- Writing a C program to output **one** such file
  - Parametrizing the program to output one object for each set of command-line parameters
- Only the first and last are part of Assignment 3A !