Lecture 5: Thursday, 18 September 1997

Today:
- More about Inventor
- More on Assignment 3A (if necessary)
- Modeling your Own 3D Shapes (H&B S7.3,10.1,11.5)
- Brief Tour of Rendering Pipeline (H&B S6.1,9.1,14.5)

Tuesday
- 3D Coordinate Transformations
- Rendering Pipeline Operation

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 four 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!

Built-In Geometric Primitives

First, trivial example:

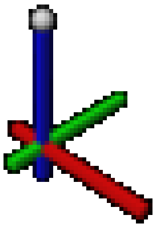
```
#Inventor V2.1 ascii
Cube {}
```

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.

Useful object: 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
```



Example (with gview)

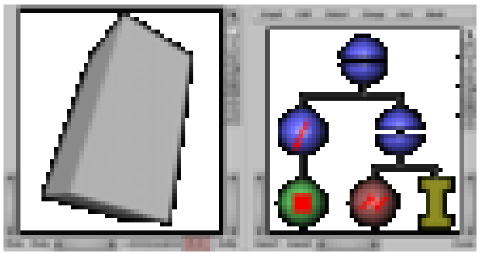
Cube and sphere, with distinct
material properties, transformation

```
#Inventor V2.1 ascii
File { name "coordaxes.iv" }
Separator {
   Material { diffuseColor 0.7 0.2 0.3 # reddish }
   Transform { translation 2 2 0 }
   Cube {}
   }
Separator {
   Material { diffuseColor 0.2 0.7 0.3 # greenish }
   Transform { translation 2 2 2 }
   Sphere {}
   }
```

(Can also alter lighting model, lights...)

Definition & Instantiation

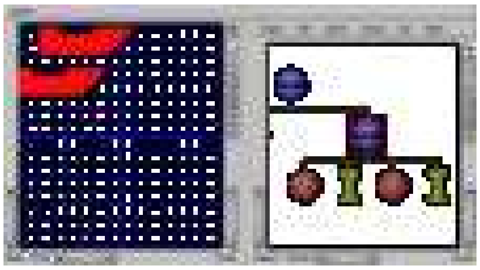
```
#Inventor V2.1 ascii

Switch { DEF unitcube # Switch prevents output!
    Cube {}
} # Switch

Separator {
    Scale { scaleFactor 1 1 2 }
    USE unitcube      # this generates output.  why?
} # Separator
```



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

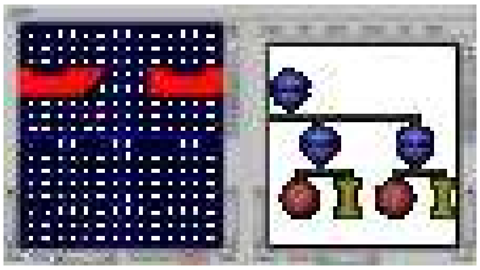
Persistence of Traversal State

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



Separator Nodes

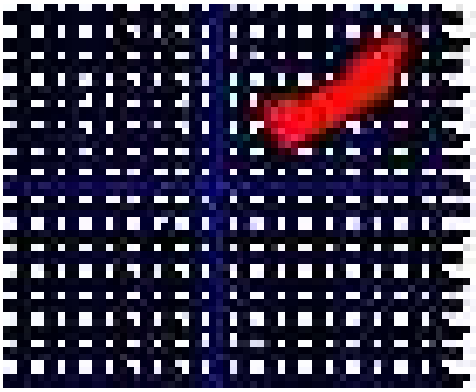
```
# place two boats on surface (expected result)
Separator {
    Translation { translation -8 3 0 }
    USE boat
    }
Separator {
    Translation { translation 2 3 0 }
    USE boat
    }
```



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

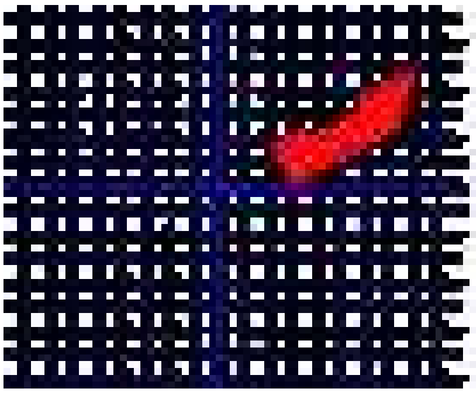
Composing Transformations

```
...
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
```

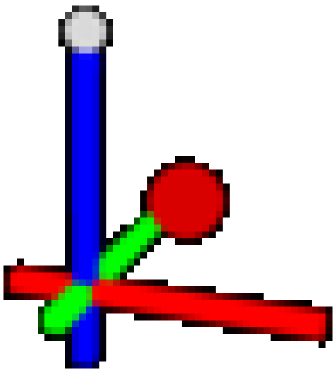


Conclusion?

Intrinsic Material Properties

To color with “intrinsic material”:

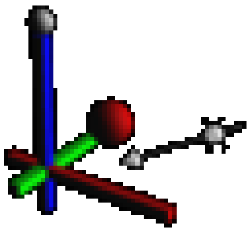
```
...
LightModel { model BASE_COLOR }
File { name "coordaxes.iv" }
Separator {
    Material { diffuseColor 0.8 0.0 0.0 }
    Sphere {}
} # Separator
```



Using a Lighting Model

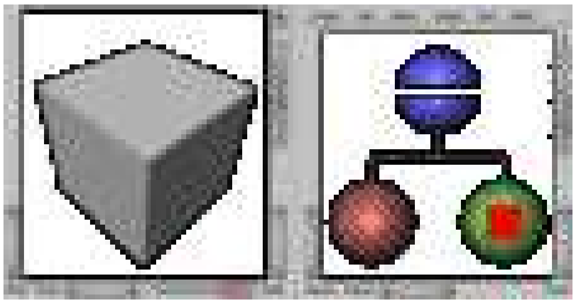
To shade with "lighting model":

```
...
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
```

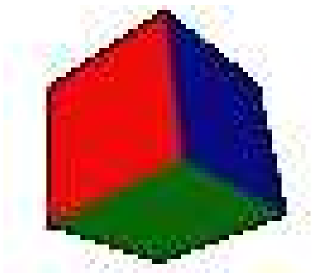


Specifying Face Normals, Colors

```
# 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
    ...
```

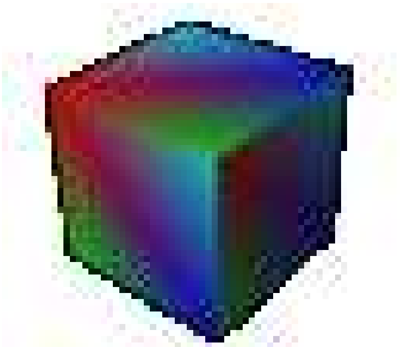

Normals, Colors Per Face

```
    ...
    # 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

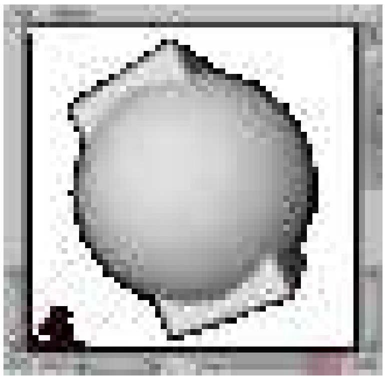
```
...
    MaterialBinding { value PER_VERTEX_INDEXED }
...
       materialIndex    [ # color ids; per face, then per vertex
                          0, 1, 2, 3, -1, # -x
                          0, 1, 2, 3, -1, # +x
                          0, 1, 2, 3, -1, # -y
                          0, 1, 2,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



See `% man SoNormalBinding` for normals

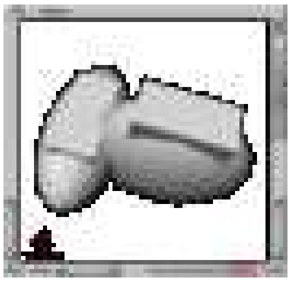
File Inclusion

```
#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
```



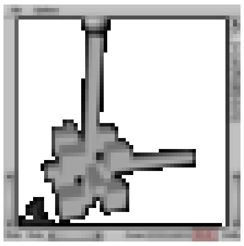
File Inclusion & Instancing

```
#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
    }
```



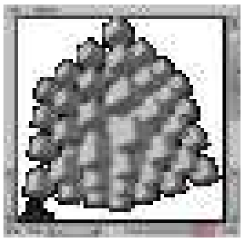
Hierarchical Instancing

```
#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*

Inventor Resources

Executables:
  `/usr/sbin/ivview foo.iv` (Viewer only)
  `/usr/demos/Inventor/`
    `gview foo.iv` (Edit SceneGraph)
    `SceneViewer foo.iv` (Edit Objects)
Example Files & Documentation
  `athena:/usr/share/data/models/`
  `athena:/usr/share/src/Inventor/`
  `athena% man SoCube`, `man SoTranslation`, etc.
    (if `/usr/share/catman` is in your `$MANPATH`)
  On reserve at LCS reading room NE43-113
    Open Inventor C++ Reference Manual (“Green book”)
    Inventor Mentor (“Orange book”)
  On Web: Inventor Quick Reference
    Top-level link from course page

Review: Useful Inventor Facts

Every file must start with `#Inventor V2.1 ascii`
Quick reference for all Inventor node types:
  `http://www.sgi.com/Technology/`
`Inventor/PostScript/quickRef.ps`
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 }`
  (scoped) to specify non-convex polygons

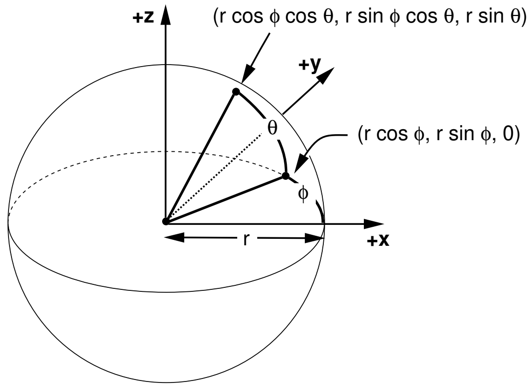
Assignment 3A – Object Modeling

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 of 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 !

Note: Asst 3A uses Athena SGIs, `make`, C, C++
- Class use of Fusion **ends** with Asst 2.

Representing Smooth Surfaces with Polygons

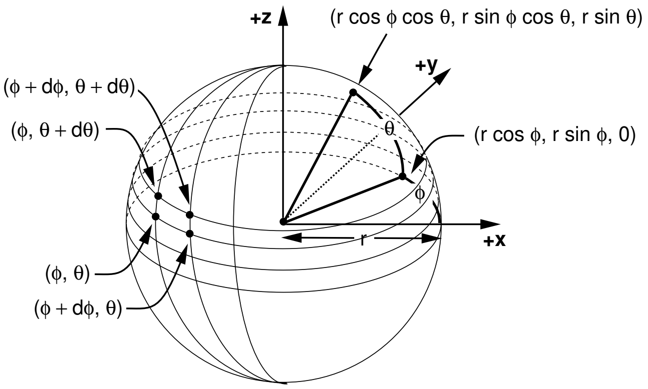
Example: Sphere, using Spherical Coordinates



$$x = r\cos(\phi)\cos(\theta)$$
$$y = r\sin(\phi)\cos(\theta)$$
$$z = r\sin(\theta)$$

Example: Faceted Sphere

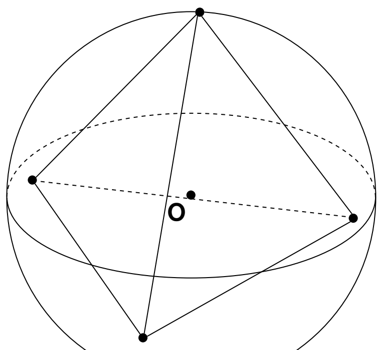
Method 1: Explicit Parametrization



What range should ϕ, θ take for sphere?
- Advantages?
- Disadvantages?

Projective Methods

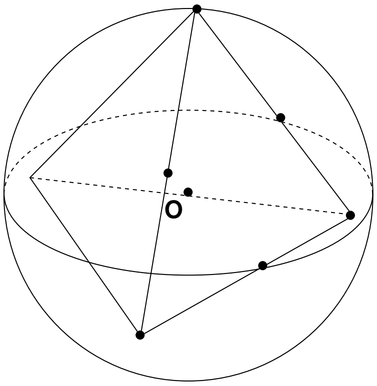
Start with a **regular polyhedron**, with cospherical vertices (by definition)



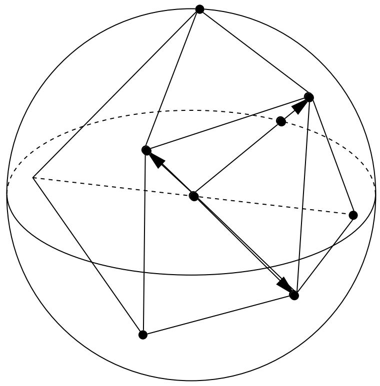
Subdivide each (planar) face
Project new vertices onto sphere
Recurse

Subdivision

Center-based, edge-based



Projection



Sphere equation (implicit):

$$x^2 + y^2 + z^2 = r^2$$

Ray from origin to vertex v_x, v_y, v_z (explicit):

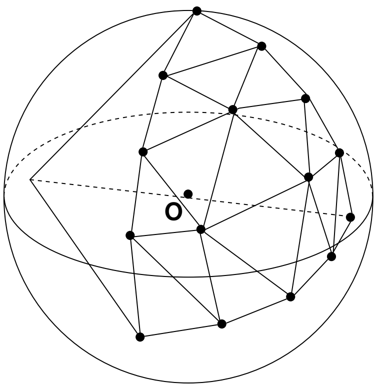
$$(0, 0, 0) + t(v_x, v_y, v_z)$$

Plug in:

$$t^2({v_x}^2 + {v_y}^2 + {v_z}^2) = r^2$$

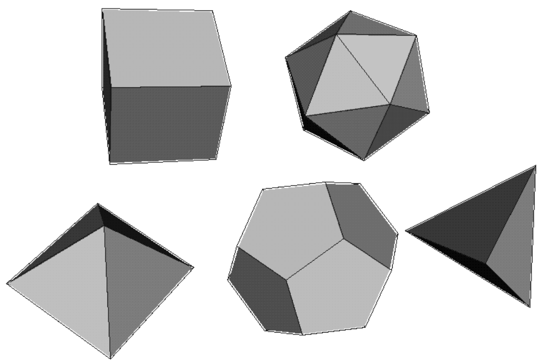
Solve for t, plug in to *explicit* expression

Recursion



How many triangles after n recursion steps?
(consider initial tetrahedron as $n = 1$)
Does this scheme exhibit any degeneracies?
What is surface normal at projected vertex $({v_x}', {v_y}', {v_z}')$?

Regular polyhedra



Tetrahedron (four equilateral triangles, π ster)
Cube (six squares, $2\pi/3$ ster)
Octahedron (eight equilateral triangles, $\pi/2$ ster)
Dodecahedron (twelve regular pentagons, $\pi/3$ ster)
Icosahedron (twenty equilateral triangles, $\pi/5$ ster)
These and many more sited at graphics group page