Lecture 4: Tuesday, 21 September 1999

Administrative:
- Asst 2 due this Friday at 5pm
- Asst 3 out today, due next Friday at 5pm
- 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: T 4pm, 4-370 / NE43-252

Today:
- Object, World Space Modeling (H&B S7.3,10.1,11.5)
- Using Inventor on Athena SGI machines
  - Modeling file format
  - Useful viewers/editors
  - Extensible (C++) tool suite, dev't. env't.
- Modeling Your Own 3D Shapes
- Assignment 3 out (due Friday 1 October)
  - Inventor object modeling; C, C++ programming !

Thursday:
- More of the Graphics Pipeline

Object and World Space Modeling

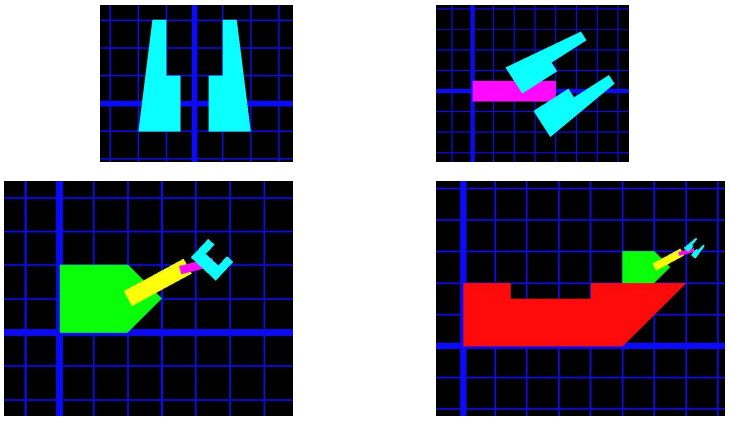
Object space a sort of "workbench" where components are assembled, combined

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

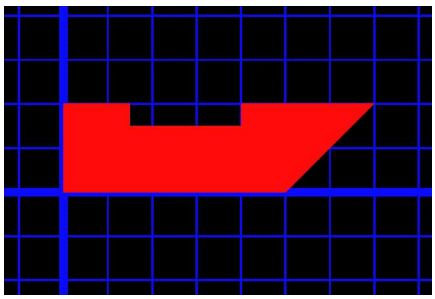
But: often useful to view both ways

Usually, required transformations, ordering obvious

Example: Boat, sub, articulated grasper arm, treasure



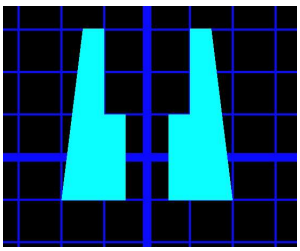
Modeling the Boat



```
# 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

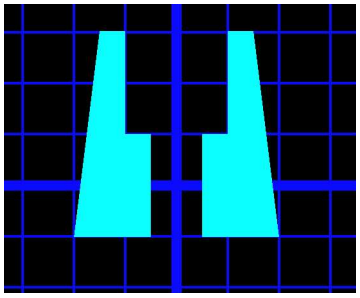
Proceed "bottom up": Grasper, Forearm, Arm, Sub



```
# 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

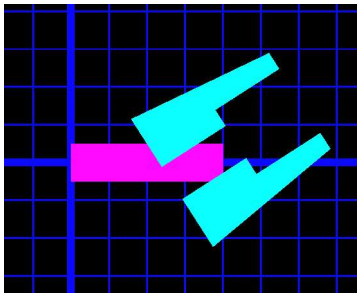
Can exploit symmetry even within grasper DEF:



```
# 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 1 } # why ?
    FaceSet { numVertices [ 6 ] }
} } # Switch
```


Forearm

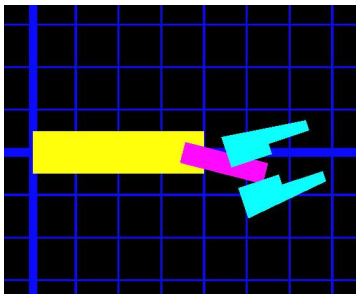
Note **USE** of grasper (rotation, translation)



```
# 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

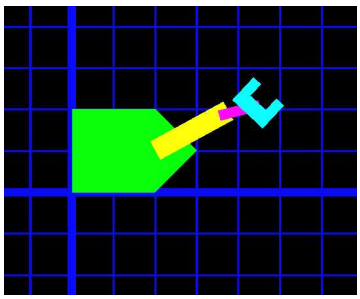
Note **USE** of forearm (which USEs grasper)



```
# 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
```

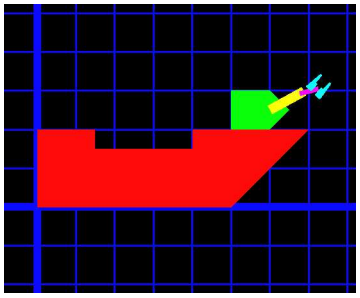

Jason Sub

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



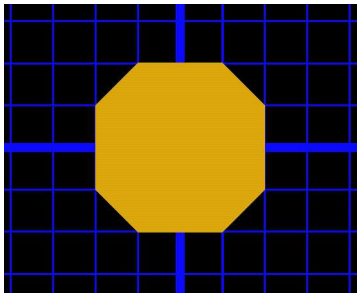
```
# jason sub
Switch { DEF jason Separator {
    Material { diffuseColor 0 1 0 }
    Coordinate3 {
        point   [  0    0    0,
                   2    0    0,
                   3    1    0,
                   2    2    0,
                   0    2    0 ]
    } # Coordinate3
    IndexedFaceSet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



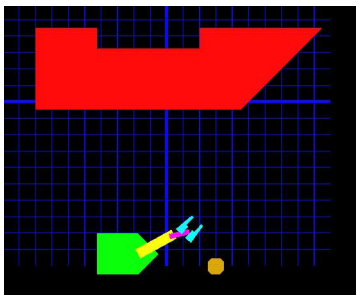
```
# boat, version 2
Switch { DEF boat Separator {
    Material { diffuseColor 1 0 0 }
    Coordinate3 {
        point   [  0    0    0,
                   5    0    0,
                   7    2    0,
                   4    2    0,
                   4    1.5  0,
                   1.5  1.5  0,
                   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



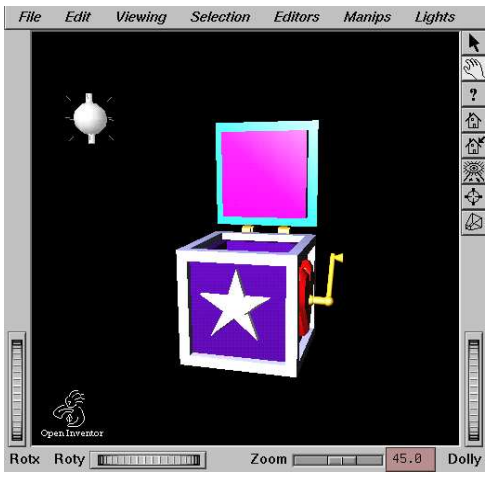
```
# boat, version 3
Switch { DEF boat Separator {
    ...
    # instance jason sub wrt boat
    Translation { translation 1.5 -4 0 }   # diving (down)
    Scale { scaleFactor 0.5 0.5 0 }
    USE jason
} } # Switch
# place boat on sea surface
Separator { Translation { translation -8 -0.5 0 }
      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

Inventor Tools

SceneViewer

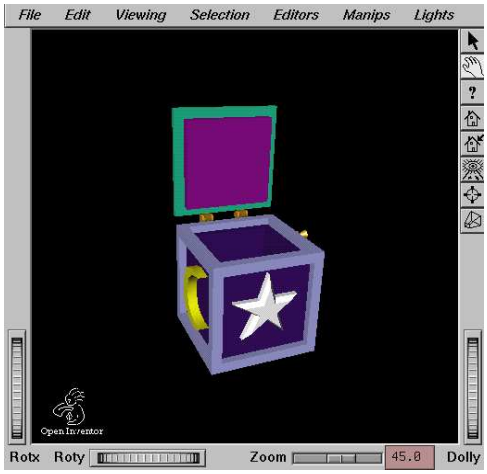
Manipulate, edit, light Inventor models




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

Interaction Modes & Icons

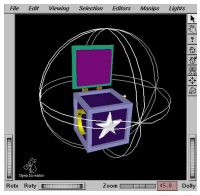
Default: 3D Viewer (hand icon)



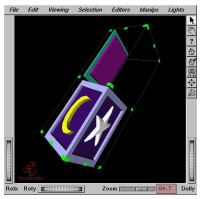
Trackball, Walk, Fly, Plane view
Also Select, Help, Home, Set Home,
Center, Set Center, Ortho/Perspective

Manipulators

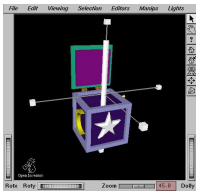
Trackball



Tab Box

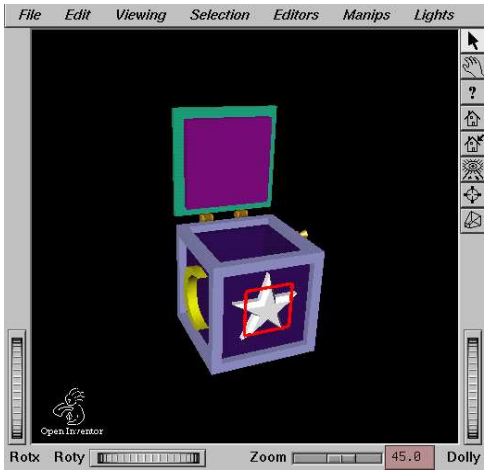


Jack



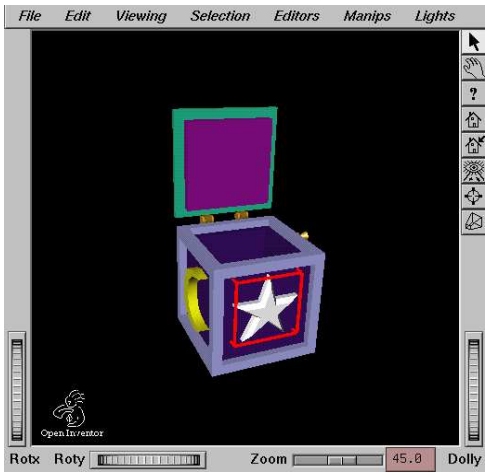
Selection

Pick mode (arrow icon); select



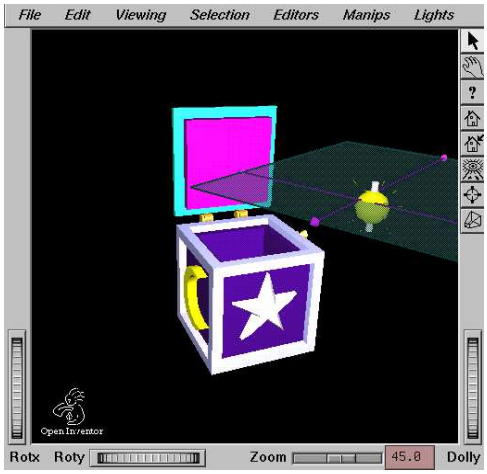
Selection

Edit → Pick Parent



Light Creation/Editing

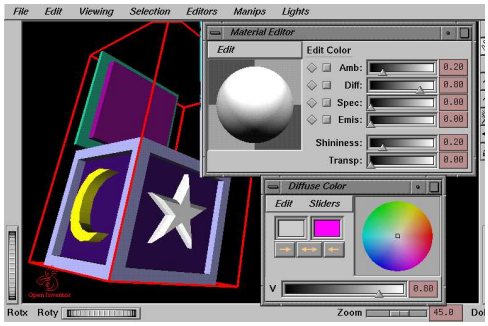
Lights → Create {Point, Dir, Spot}



Material / Color Editing

Editors → {Material, Color}

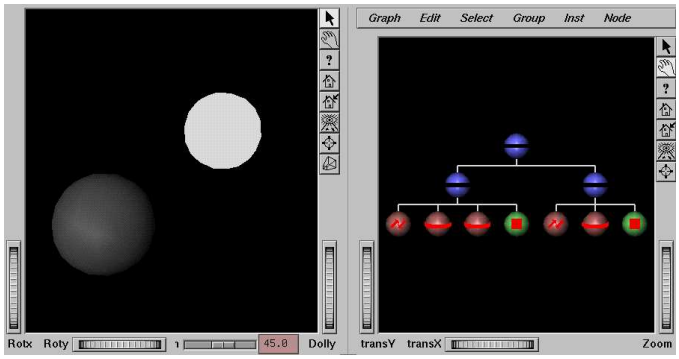
Enabled only when some entity is selected



Inventor Representations

Scene graph (DAG), comprised of several node types:

- Shape: 3D geometric objects
- Transform: Affect current transformation
- Property: Appearance, texture, etc.
- Group: Collection of subgraphs



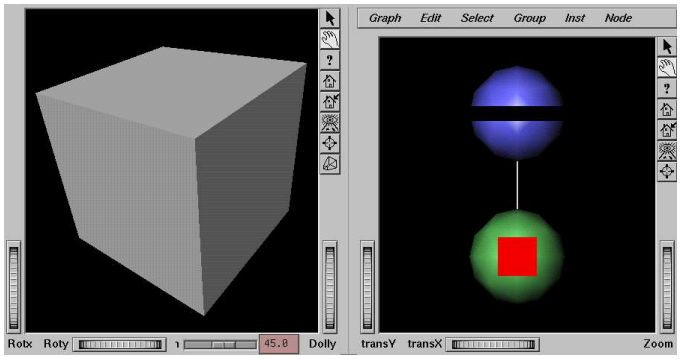
Traversal state:

- 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:

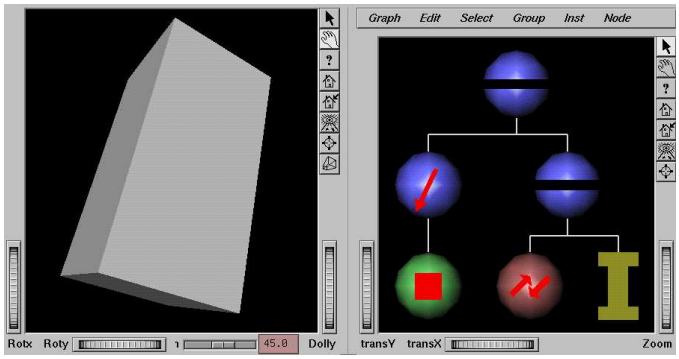
- Default W,H,D (for cubes, spheres, cones, etc.);
- Default Color; Material; &c

Traversal state (all defaulted for now):

- Xform; Lights; Material; &c.

Definition & Instantiation

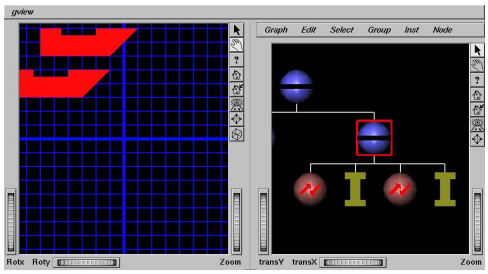
```
Switch { DEF unitcube
    Cube {}
} # Switch

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)

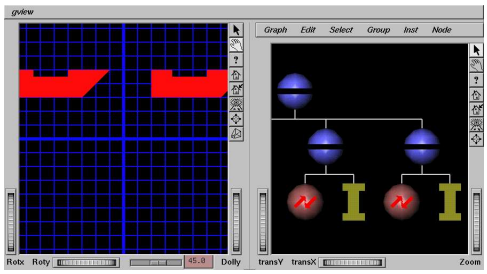
Persistence of Traversal State

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



Separator Nodes

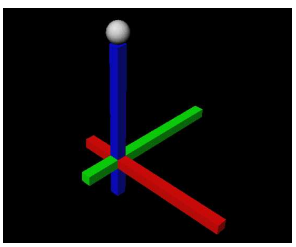
```
# place two boats on surface (expected result)
Separator {
    Translation { translation -8 3 0 }
    USE boat
    }
Separator {
    Translation { translation 2 3 0 }
    USE boat
    }
```



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

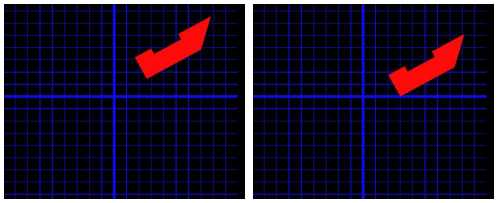
Useful scene graph: Axes (`coordaxes.iv`)

```
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
```



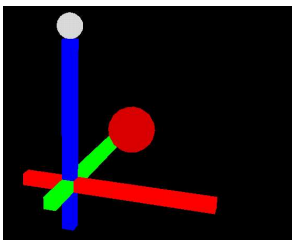
```
...
Separator {
    Translation{ translation 3 0 0 }
    Rotation { rotation 0 0 1 0.5 } # axis, angle
    USE boat
} # Separator
```

Conclusion?

Material Properties

To color with "intrinsic material":

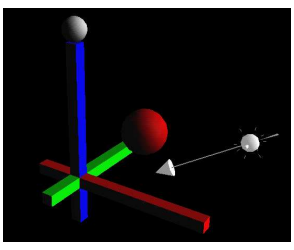
```
...
LightModel { model BASE_COLOR }
File { name "coordaxes.iv" }
Separator {
    Material { diffuseColor 0.8 0.0 0.0 }
    Sphere {}
} # Separator
```



Using a Lighting Model

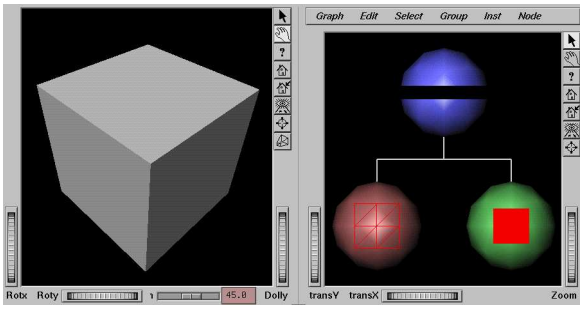
To shade with "lighting model":

```
...
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
```

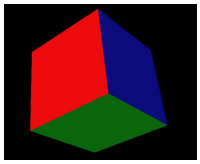


Specifying Face Normals, Colors

```
# 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
    ...
```

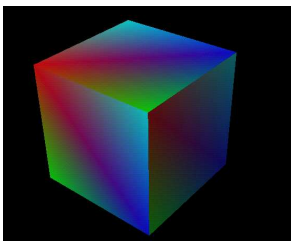


```
    ...
    # assign color ids per face
    MaterialBinding { value PER_FACE_INDEXED }
    IndexedFaceSet {
        coordIndex      [ # define faces with vertex ids
                          4, 5, 1, 0, -1, # -x
                          3, 2, 6, 7, -1, # +x
                          1, 5, 6, 2,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

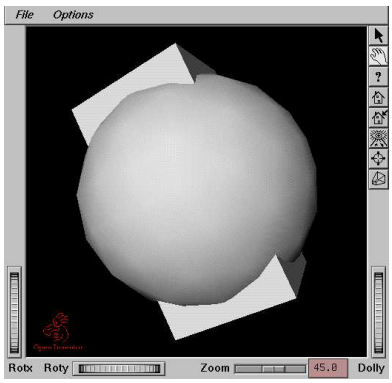
```
...
    MaterialBinding { value PER_VERTEX_INDEXED }
...
       materialIndex    [ # color ids; per face, then per verte
                          0, 1, 2, 3, -1, # -x
                          0, 1, 2, 3, -1, # +x
                          0, 1, 2, 3, -1, # -y
                          0, 1, 2,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



See `% man SoNormalBinding` for normals

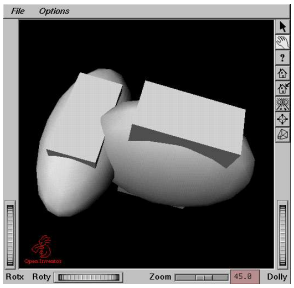
File Inclusion

```
#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
```



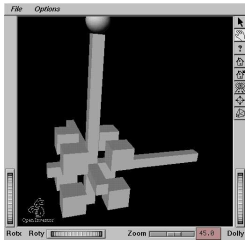
File Inclusion & Instancing

```
#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
    }
```



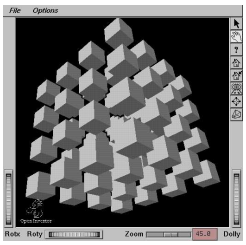
Hierarchical Instancing

```
#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

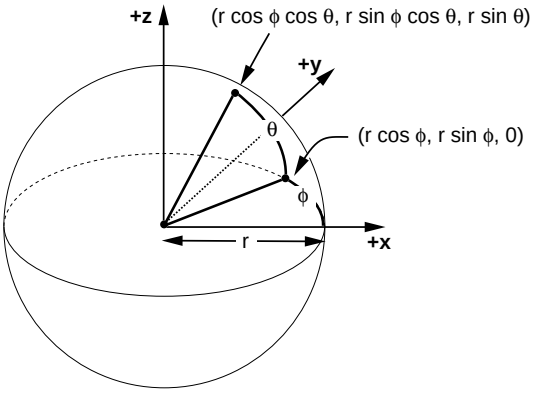
Use simple hierarchy when:
- natural "ownership" relation exists
- repetitive structure occurs
- simple shapes reoccur

Some entities require generalized hierarchy
- graphs, networks, etc.

Some entities have no hierarchical structure
- induce one with *spatial data structures*

Representing Smooth Surfaces with Polygons

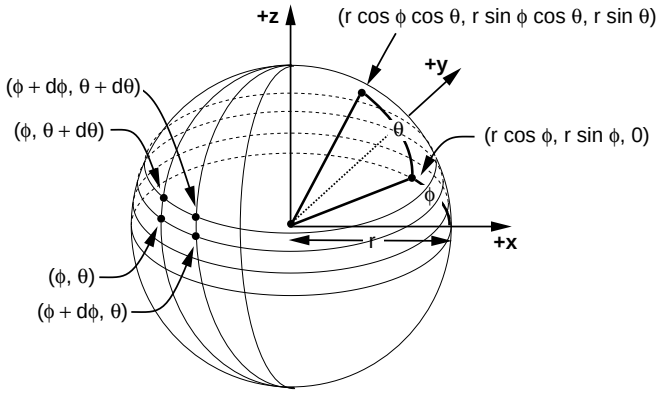
Example: Sphere, using Spherical Coordinates



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

Example: Faceted Sphere

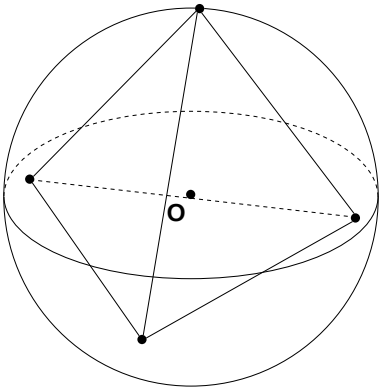
Method 1: Explicit Parametrization



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

Projective Methods

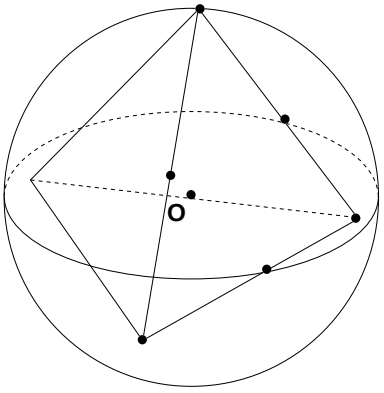
Start with a **regular polyhedron**, with cospherical vertices (by definition)



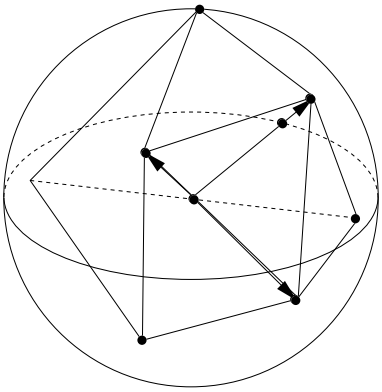
Subdivide each (planar) face
Project new vertices onto sphere
Recurse

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):

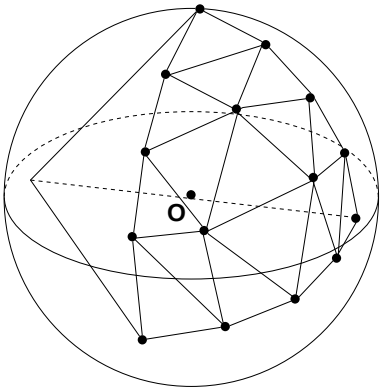
$$(0, 0, 0) + t(v_x, v_y, v_z)$$

Plug in:

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

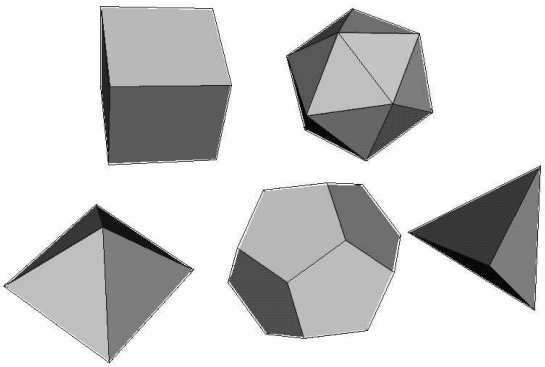
Solve for t, plug in to *explicit* expression

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

Quadrics

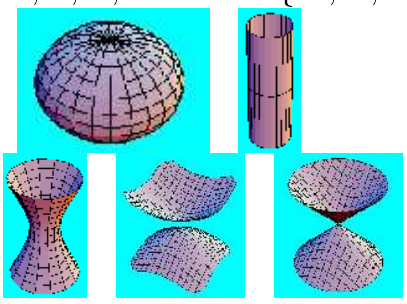
Implicit surfaces of form

$$\begin{pmatrix} X & Y & Z & 1 \end{pmatrix} \begin{pmatrix} A & B & C & D \\ B & E & F & G \\ C & F & H & I \\ D & G & I & J \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = 0$$

Or, can be expressed as

$$Ax^2 + 2Bxy + 2Cxz + 2Dx + Ey^2$$
$$+2Fyz + 2Gy + Hz^2 + 2Iz + J = 0$$

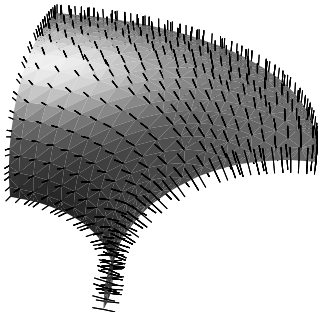
"Normalized" form:
A, E, H, and one of { D, G, I } $\in 0, \pm 1$



`http://amath.colorado.edu/appm/staff/fast/java/qg.html`

Parametrizing More Complex Objects

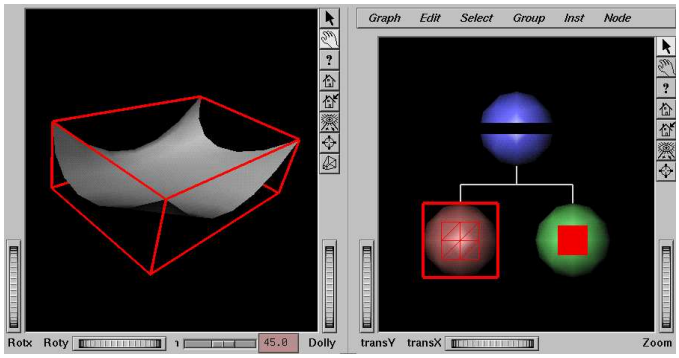
Example: Torus



How do you parametrize a torus?

Free-Form Surfaces: Bézier Patches

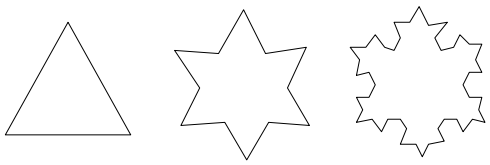
```
Separator {
    Coordinate3 {
        point   [ -1.5 -1.5  0,
                  -0.5 -1.5  0,
                   ...
                   0.5  1.5  0,
                   0.5  1.5  0, ]
    } # Coordinate3
    IndexedNurbsSurface {
        numUControlPoints 4
        numVControlPoints 4
        uKnotVector [ 0, 0, 0, 0, 1, 1, 1, 1 ]
        vKnotVector [ 0, 0, 0, 0, 1, 1, 1, 1 ]
        coordIndex [ 0, 1, 2, 3,
                     4, 5, 6, 7,
                     8, 9, 10, 11,
                     12, 13, 14, 15 ]
    } # IndexedNurbsSurface
}
```

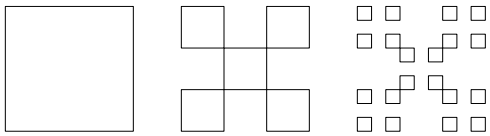
Use very sparingly (slow to render...)

Fractals: Self-similar objects (FvDFHP §9.5)

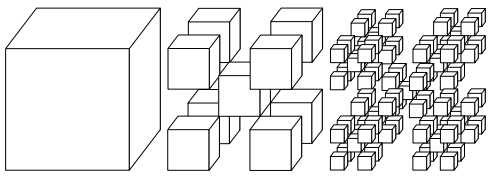
Recursive *construction rules* yield regular fractals



von Koch snowflake
(strictly speaking, convergent or limit curve)
Sub-objects need not be connected:

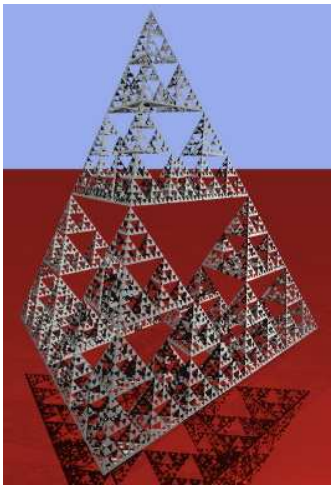


Three-D construction rules abound:



Fractal Objects

Sierpinski's gadget (area, volume in limit?)

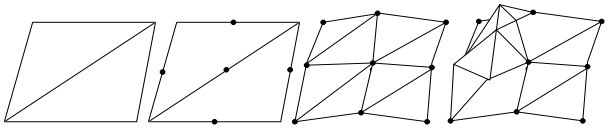


(Credit: Unreal image, rendered on Amiga)

Other versions possible

Terrain: irregular fractal generation

Usually height fields, i.e., $z = f(x, y)$



Displacements can be *random*:

```
// returns pseudo-random numbers in [ 0 .. (2^15)-1 ]
extern int rand ( void );

// return random float in [low..high]
float randr ( float low, float high )
{
    return low + ( rand() / ((1<<15)-1.0) ) * (high-low);
}
```

Can also use randomness for (e.g.)

- geometric perturbations
- parameter settings
- material properties

Assignment 3 – 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
  - Parametrize the program to output one object for each configuration of command-line parameters
- Only the first and last are part of Assignment 3 !

We provide the "template" program `uid_object.c`

- generalized command-line handling
- valid Inventor file (by default)
- embedded informational comments

Parametric Modeling

You should support at least four parameter types:

- Existence – a binary attribute
- Continuous – some arbitrary-valued attribute
- Discrete – some integer-valued attribute
- Material – a modifier for the surface properties

Extensions:

- Animation (Rotor, Shuttle, Engines, etc.)

Constraint: object must render at ≥ 5 Hz on an O2

- (can use `ivview -q` to optimize)

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 if at all:

```
Transform { # fixed order S, R, T!
    translation 0 0 6.5
    rotation 1 1 1 1.5707963
    scaleFactor 0.5 0.5 0.5
} # Transform
```

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

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