Creating VRML Worlds

Part1: Basic Look

- Potted history
- Node Types
- Shapes
- Materials
- Texture
- Scene Graph

Anthony Steed

Department of Computer Science
University College London

Introduction to VRML

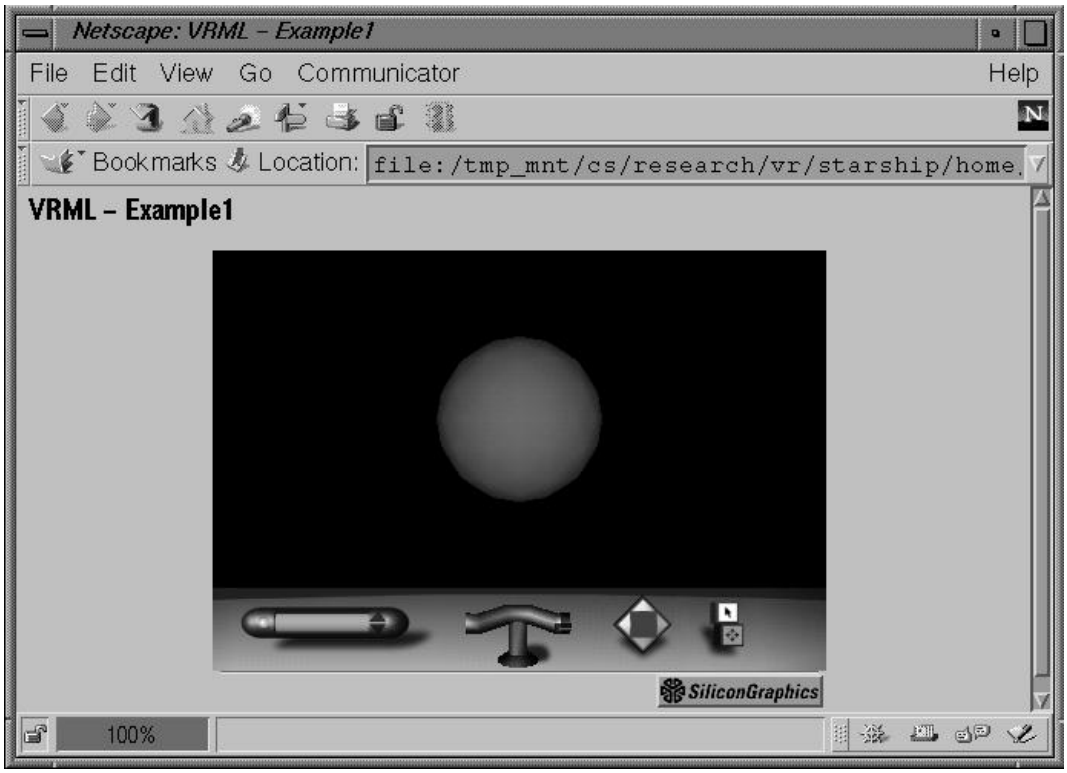
- **Virtual Reality Modelling Language**
  - A file format to describe 3D scenes
    - identified by ".wrl" extension
  - Also a description of a run-time system for animating worlds
  - Designed to be used in network situations
    - Integration with web browsers
    - Can embed VRML in HTML documents
  - Supports interaction and simulation of behaviour of objects
    - 3D hyperlinks
    - Movement about scene
  - Designed to run on desktop PCs through to high-end workstations
  - MIME type is *model/vrml*

VRML History

- **VRML1.0 (May 1995)**
  - Derived from Silicon Graphics Open Inventor
  - Static scenes only
  - Specification was ambiguous, VRML browsers varied greatly
- **VRML2.0 (August 1996)**
  - Derived from Silicon Graphics “Moving Worlds” proposal
  - Response to the need for more dynamic worlds
  - Harder to implement a browser so fewer exist
- **VRML97 (December 1997)**
  - ISO standardisation of VRML2.0
  - No major changes
  - Hopefully most ambiguities removed

Browsing VRML

- **Load file into Netscape (3/4) or Internet Explorer (3/4)**
- **VRML content appears in a "VRML Browser"**



- **Controls (CosmoPlayer)**
  - Navigate around scene
  - Select, pick or drag objects in the 3D scene
  - Select viewpoints to jump to

VRML Basics

- **VRML file contains *Nodes* that describe the scene**
- **A Node is defined with several *Fields***
- **Example Node specification**

```
Cone {
   field SFFloat bottomRadius 1
   field SFFloat height       2
   field SFBool  side         TRUE
   field SFBool bottom        TRUE
}
```

  - Each line give the field, the type of the field, the name and the default value.
  - Example use

```
Cone {
   height 5
   bottom FALSE
}
```

Sample VRML File

- **“example1.wrl”**

#VRML V2.0 utf8

```
Transform {
  children [
    Shape {
    appearance Appearance {
      material Material {
        diffuseColor 0.1 0.7 0.2
      }
    }
    geometry Sphere {
      radius 2
    }
    }
   ]
}
```

- **Every VRML97 file starts**

#VRML V2.0 utf8

Adding VRML to HTML

- **Example HTML File**

```
<html>
  <head>
    <title>VRML - Example1</title>
  </head>

  <body>
    <h1>VRML - Example1 </h1>
    <center>
      <embed src="example1.wrl" border=0
     height="300" width="400">
    </center>
  </body>
</html>
```

Classes of Node

- **Shapes**
  - Geometry
  - Appearance
- **Transformations**
- **Lights**
- **Groups**
- **Interpolators**
- **Sensors**
- **Scripts**
- **Fog**
- **Background**
- **Audio**

Shapes

- **Each Shape has a geometry field that contains a geometry node and an appearance field that contains an Appearance node**

Shape {

appearance <some appearance>

geometry <some geometry>

}

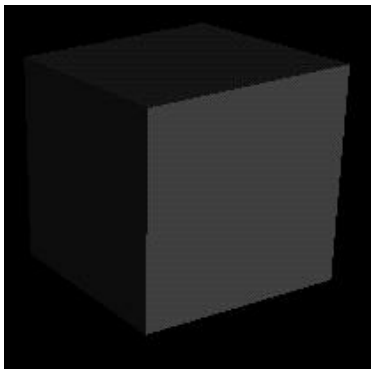
- **Example (from example1.wrl)**

```
Shape {
    appearance Appearance {
      material Material {
        diffuseColor 0.1 0.7 0.2
      }
    }
    geometry Sphere {
      radius 2
    }
```

Geometry Nodes

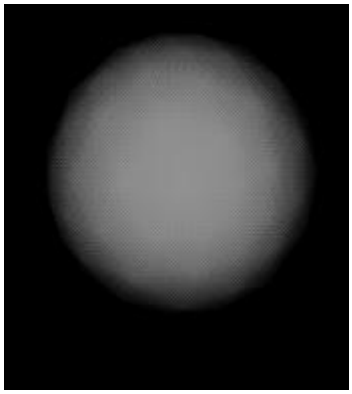
- **Basic types**
  - Box
  - Sphere
  - Cylinder
  - Cone
  - Text
- **Box**
  - defined by its size field

```
Box {
    size 2.0 2.0 2.0
}
```



- **Sphere**
  - defined by its radius field

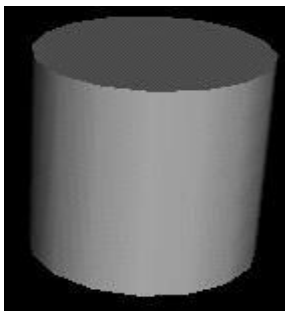
```
Sphere {
    radius 1.5
}
```



Geometry Nodes

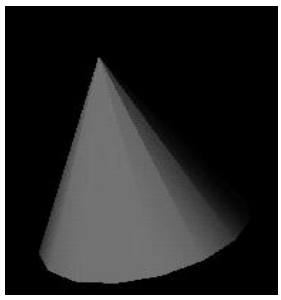
- **Cylinder**
  - defined by its height and radius fields

```
Cylinder {
    height 2.0
    radius 1.0
}
```



- **Cone**
  - defined by its height and radius fields

```
Cone {
      radius 1.3
      height 1.8
}
```



- **Text**
  - defined by the string and the font

```
geometry Text {
  string ["Hi!"]
  fontStyle FontStyle {
    family "TYPEWRITER"
    style "ITALIC"
  }
}
```

Appearance

- **Defines the look of some piece of geometry**
  - Material
    - combination of
      - ambient colour
      - diffuse colour
      - emmisive colour
      - shinines
      - transparency
      - specular colour
  - Texture
    - defines a picture to paste to the object
      - fetch from a URL
      - supports movies
  - TextureTransform
    - defines how the picture is applied to the object

Material Examples

- **Material**
  - Shiny Material

    ambientIntensity 0.3

    diffuseColor 0.1 0.7 0.2

    specularColor 0.6 0.8 0.6

    shininess 0.6
  - Dull Material

    ambientIntensity 0.1

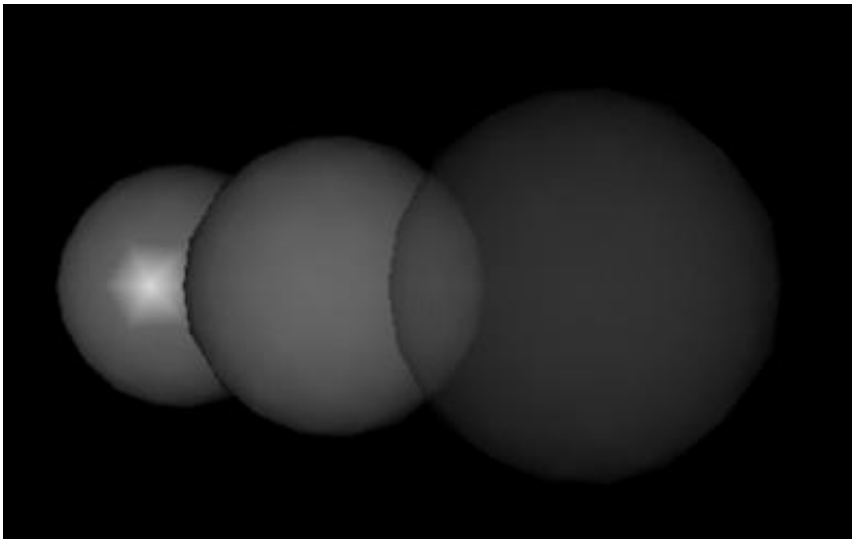
    diffuseColor 0.1 0.7 0.2

    shininess 0.0
  - Transparent Material

    diffuseColor 0.1 0.7 0.2

    transparency 0.5

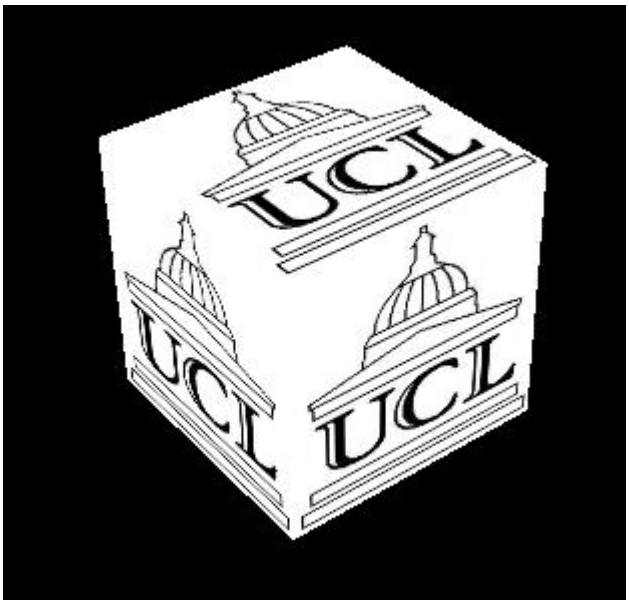
Colour components defined in RGB (red, green, blue triplets)



Texture Mapping

- **Original Image**

- **Applied to a Box**



Manipulating Textures

- **TextureTransform**
  - translation

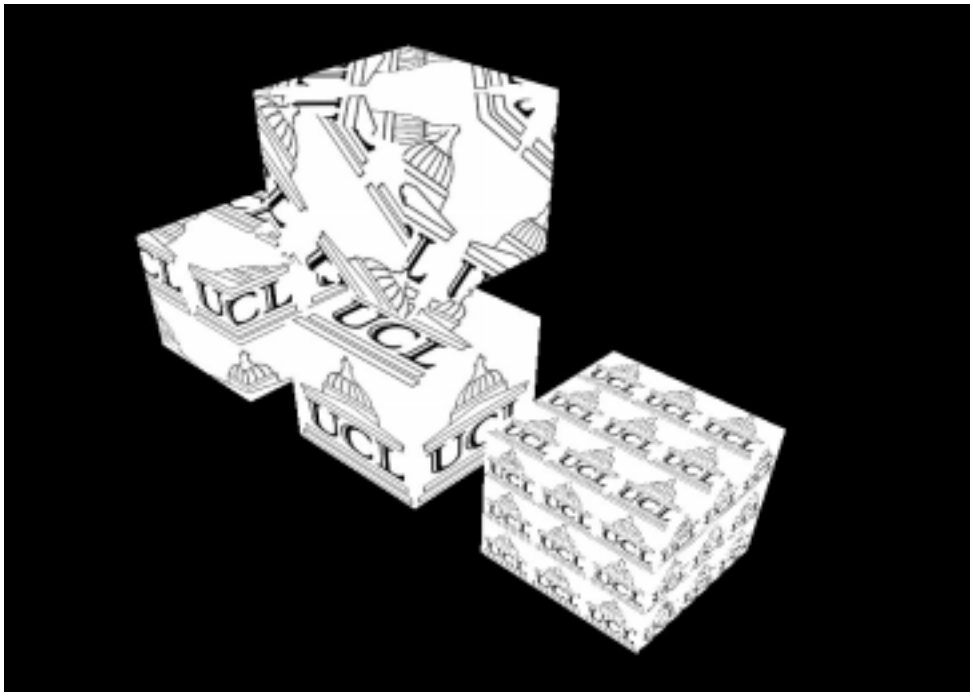
```
textureTransform TextureTransform {
        translation 0.5 0.5  }
```

  - rotation

```
textureTransform TextureTransform {
        rotation 0.785  }
```

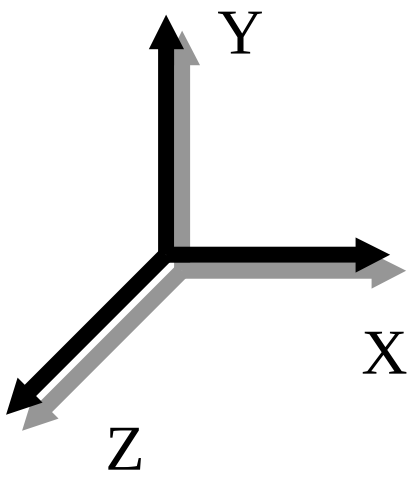
  - scale

```
textureTransform TextureTransform {
        scale 3.0 3.0  }
```



Transformations

- **Define the positions of objects in 3D space**

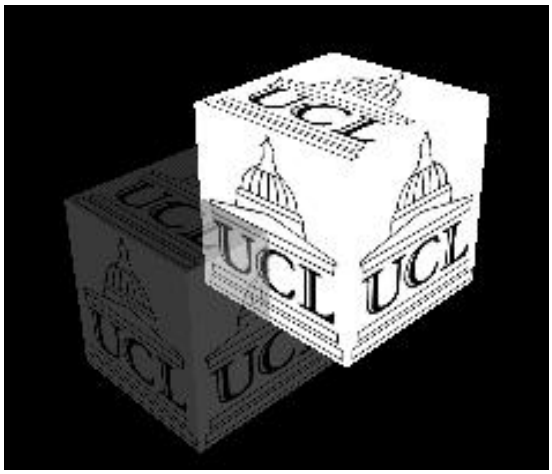


- **XY are the plane of the screen**
- **Z is towards the Viewer**
- **Transformation basically contains**
  - rotation
  - scale
  - translation
  - a set of children nodes
- **Rotations follow “right-hand screw rule”**

Effects of Transformation

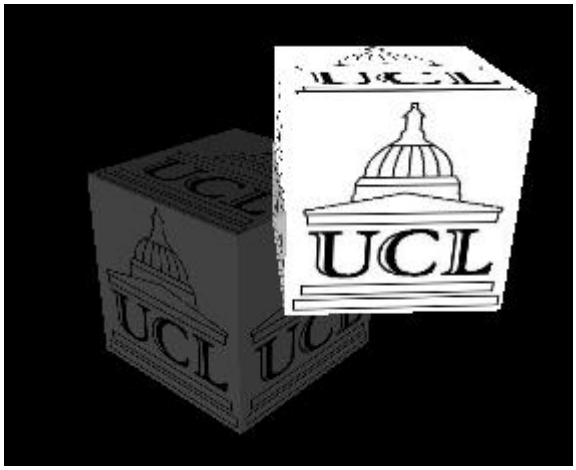
- **translation**

Transform {
translation 1 1 -1
children [
…
]
}



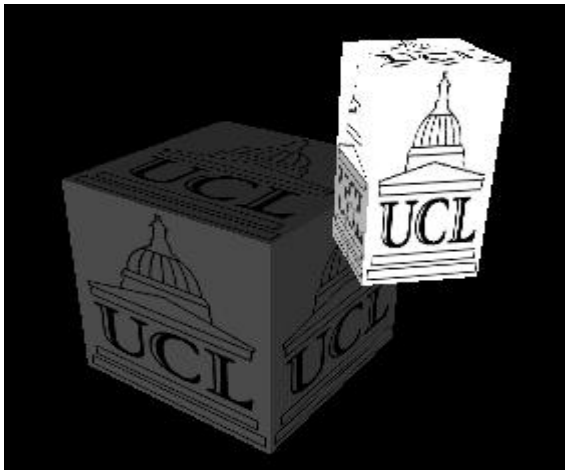
- **rotation**

translation 1 1 -1
rotation 0 1 0 0.785



- **scale**

translation 1 1 -1
rotation 0 1 0 0.785
scale 0.5 1.0 0.5

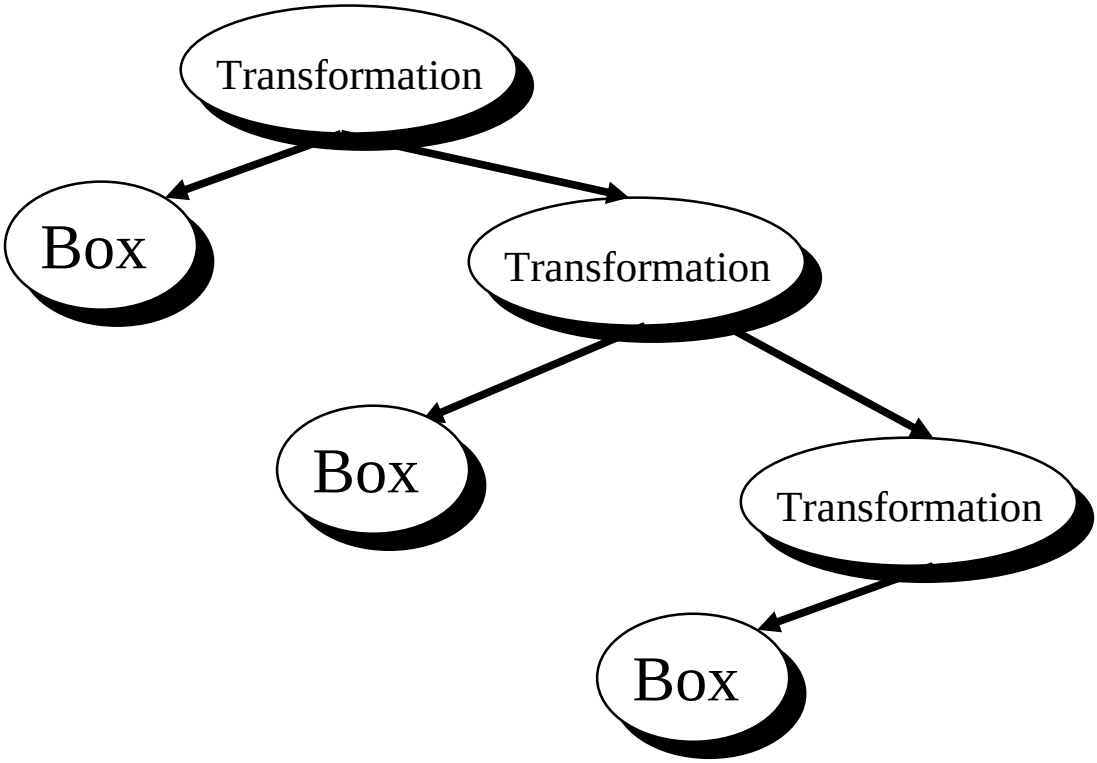


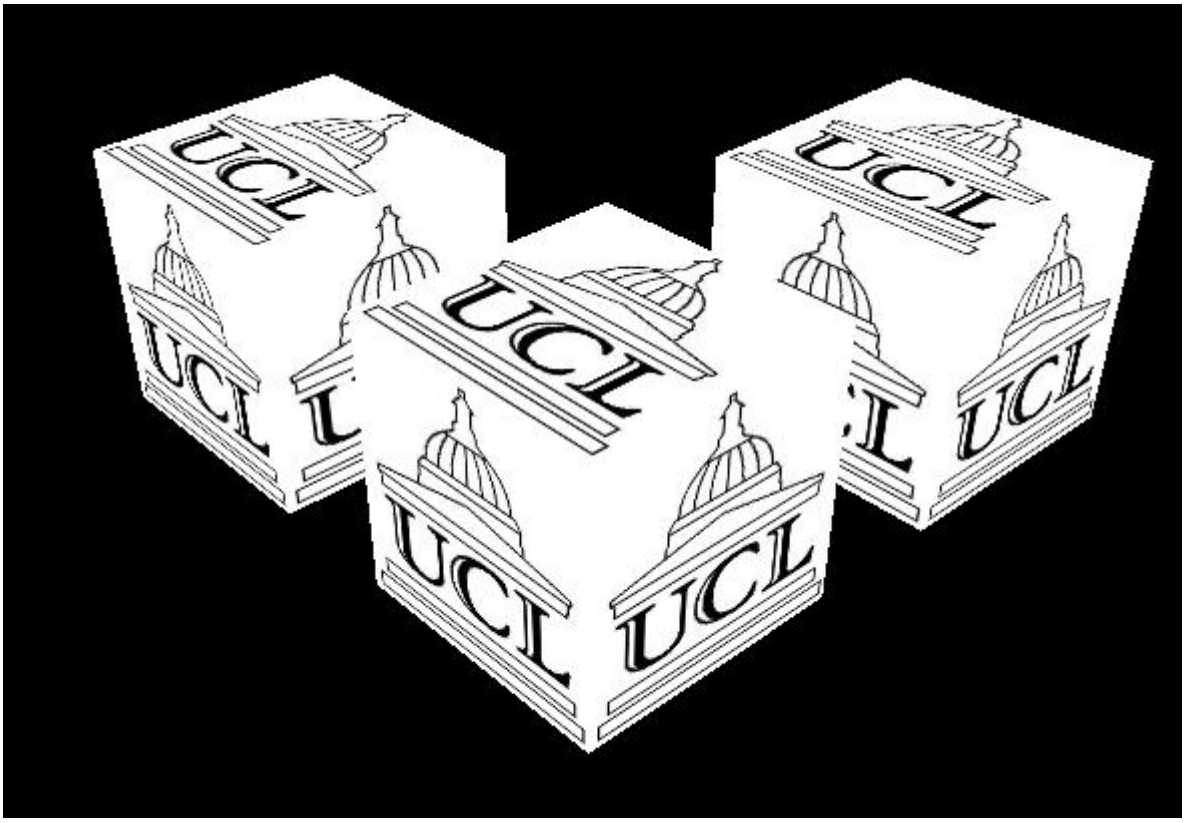
Scene Graph

- **Transforms are hierarchical**
  - forms a directed acyclic graph

```
Transform {
  translation 0 2 0
  children [
    Shape { <OBJECT1> }
    Transform {
      translation 2 0 0
      children [
        Shape { <OBJECT2> }
        Transform {
          translation 0 0 -2
          children [
            Shape { <OBJECT3> }
          ]
        }
      ]
    }
  ]
}
```

Transform Hierarchy Example





Creating VRML Worlds

Part2: Complex Looks

- DEF and USE
- Lights
- Graphical Node
  - Background
  - Fog
  - LOD
  - Billboards
  - Complex Geometry (IndexedFaceSet)

Anthony Steed

Department of Computer Science
University College London

Node Re-Use

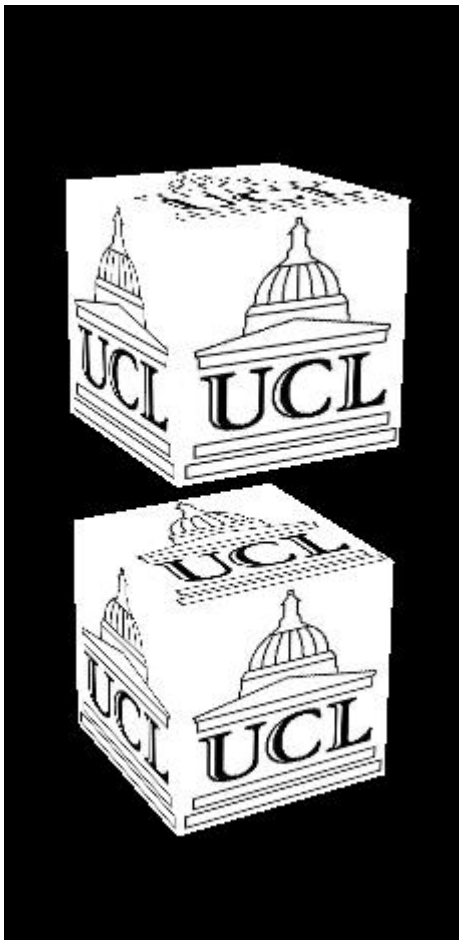
- **Objects can be re-used using DEF/USE pairs**
- **DEF a name for a node when it is first being decsribed**

```
DEF UCL_BOX Shape {
  appearance Appearance {
    texture ImageTexture {
      url ["logo.jpeg"]
    }
  }
  geometry Box {
    size 2 2 2
  }
}
```

- **USE a node whenever a node of that type is required**
  - I.E. if you DEF an Appearance node then USE of that node must be in a place that is valid for an Appearance node to be

DEF/USE Example

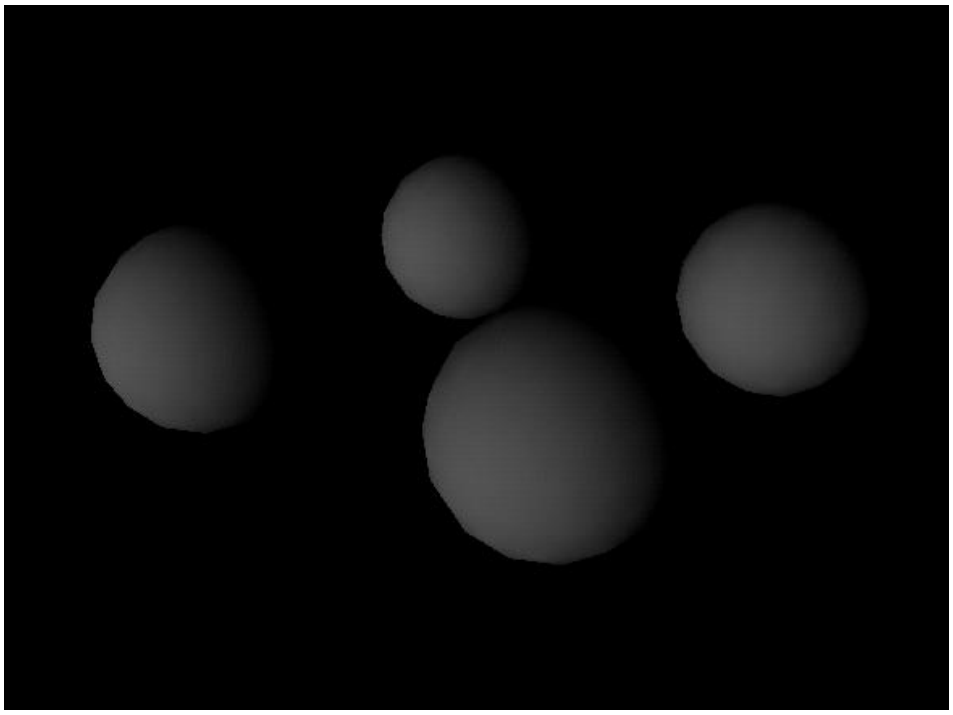
```
#VRML V2.0 utf8

Transform {
  translation 0 0 0
  children [
    DEF UCL_BOX Shape {
     appearance Appearance {
       texture ImageTexture {
         url ["logo.jpeg"]
       }
     }
     geometry Box {
       size 2 2 2
     }
    }
  ]
}

Transform {
  translation 0 3 0
  children [
    USE UCL_BOX
   ]
}
```



Lights

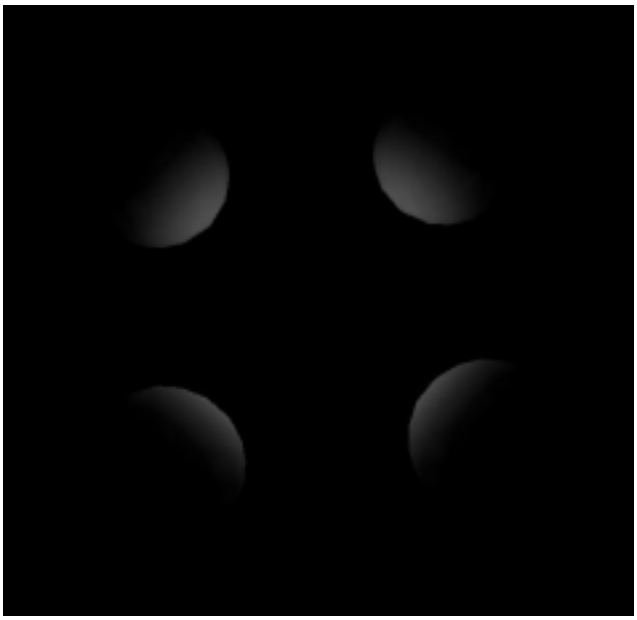
- **Provide illumination in the scene**
  - DirectionalLight
  - PointLight
  - SpotLight
- **DirectionalLight**
  - light rays travelling in parallel lines e.g. sunlight

```
DirectionalLight {
        direction -1 -1 -1
        intensity 0.8
}
```



Lights

- **PointLight**
  - radiate in all directions



- **SpotLight**
  - radiate only in certain directions, and with volumes of different intensity
- **SpotLight and PointLight “attenuate” - their effect can decrease over distance**

Graphical Nodes

- **Background**
  - specifies texture and colours to apply behind all other geometry
- **Fog**
  - (both fog and background nodes are *bindable* i.e. only one of them can be active at any one time)
- **Complex Geometry**
  - IndexedFaceSets
  - IndexedLineSets
  - PointSet
  - ElevationGrid
  - Extrusion
- **LOD**
  - multiple versions of an object
- **Billboard**
  - geometry that always faces the user

Background

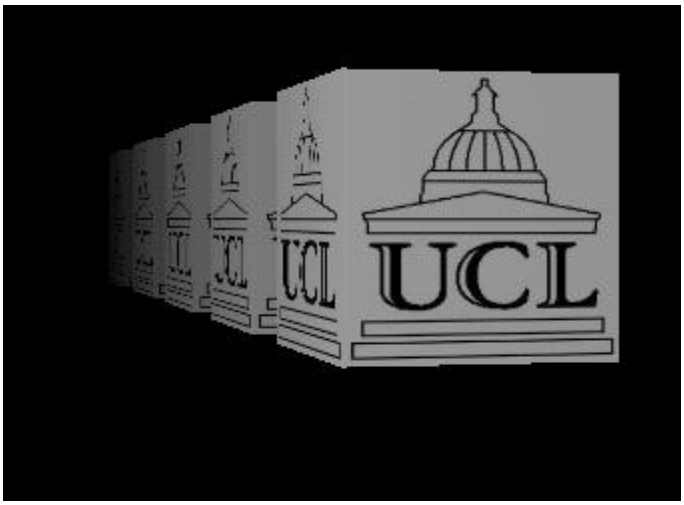
- **Can specify colour ranges for sky and ground**
- **Can specify 6 images to paste onto box "outside" all geometry (backUrl [] etc…)**

```
Background {
  groundColor [0 0 0,
               0 0 0]
  groundAngle [1.57]
  skyColor [0.4 0.4 0.4,
            0.4 0.4 0.4,
            0.9 0.9 0.9]
  skyAngle [1.1, 1.45]
}
```

Fog

- **Can specify colour and “type” of fog (LINEAR/EXPONENTIAL)**
- **Specify maximum visible distance**

```
Fog {
  color 0 0 0
  visibilityRange 25
}
Transform {
  translation 0 0 -3
 children [
    USE UCL_BOX
  ]
} etc ….
```

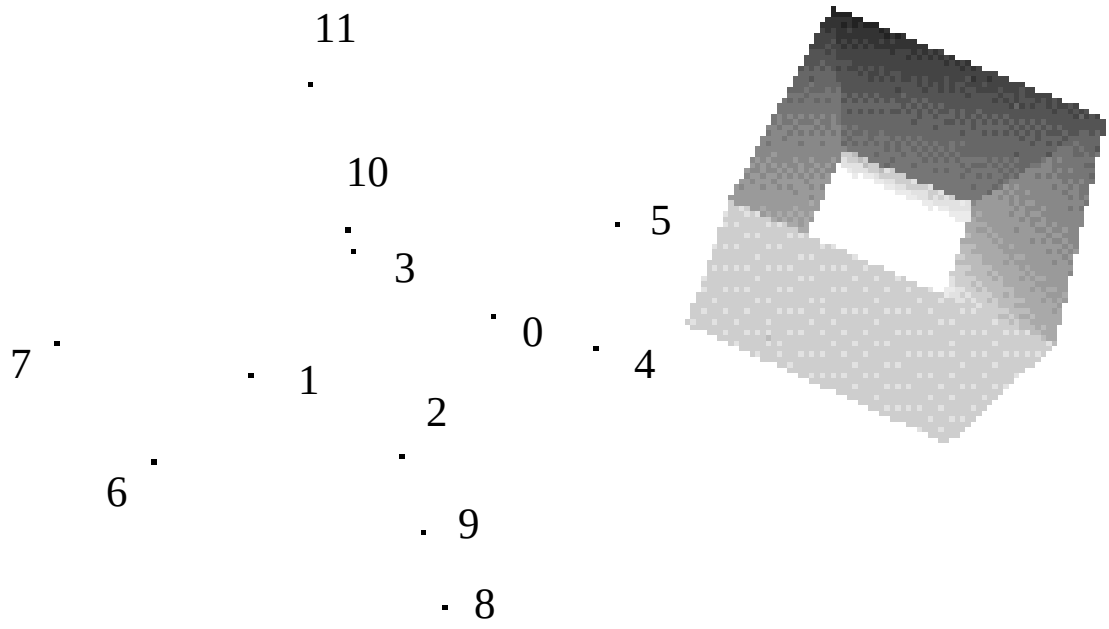


Complex Geometry

- **Many objects can not be described with standard solids**
- **General geometry described with**
  - IndexedFaceSet
  - IndexedLineSet
  - PointSet
  - ElevationGrid
  - Extrusion
- **Each uses some of the following nodes in definition**
  - Coordinate
  - Normal
  - Color
  - TextureCoordinate

PointSet

- **Define a set of points and the colour of each point**




```
geometry PointSet {
    color Color { color [1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1,
                         1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1, 1 1 1] }
    coord DEF COORDS Coordinate {
     point [
            1  0  0, -1  0  0,  0  0  1,  0  0 -1,
            2 -1  0,  2  1  0, -2 -1  0, -2  1  0,
            0 -1  2,  0  1  2,  0 -1 -2,  0  1 -2 ]
    } }
```

IndexedFaceSet

- **Most commonly used geometry node**
- **Complex definition**

```
IndexedFaceSet {
        eventIn MFInt32 set_colorIndex
        eventIn MFInt32 set_coordIndex
        eventIn MFInt32 set_normalIndex
        eventIn MFInt32 set_texCoordIndex
        exposedField SFNode color NULL
        exposedField SFNode coord NULL
        exposedField SFNode normal NULL
        exposedField SFNode texCoord NULL
        field SFBool   ccw TRUE
        field MFInt32  colorIndex []
        field SFBool colorPerVertex TRUE
        field SFBool  convex TRUE
        field MFInt32  coordIndex []
        field SFFloat creaseAngle    0
        field MFInt32  normalIndex []
        field SFBool normalPerVertex TRUE
        field SFBool  solid TRUE
        field MFInt32  texCoordIndex []
}
```

IndexedFaceSet

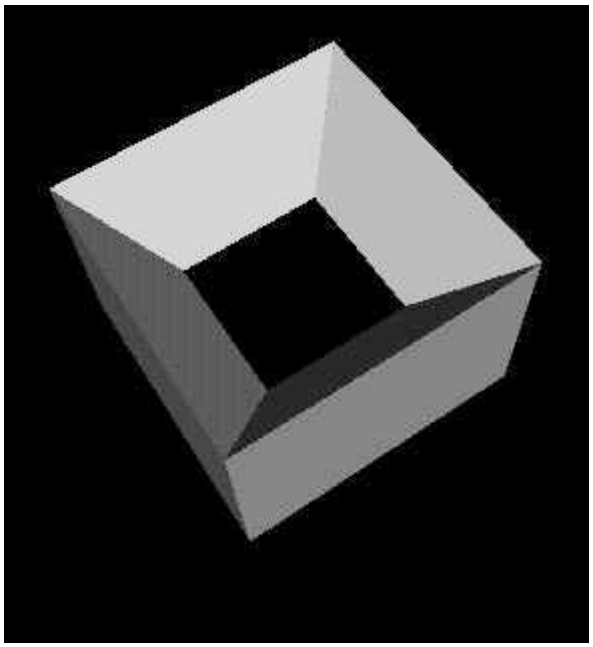
- **Components**
  - the points to construct the face from
  - the normals to give at the points
  - the colours of the points
  - the texture coordinates of the points
- **Can define normals and colours**
  - for each vertex or each face
- **Texture coordinate defined at each vertex (since they are usually different!)**

- **Hints**
  - each face is “convex”
  - the faces are defined in counter-clockwise order
  - the object is solid (i.e. each triangle is one-sided)

IndexedFaceSet

- **Coordinate**
- **Define faces using list of vertices each terminated by -1**

```
geometry IndexedFaceSet {
        coord USE COORDS
        coordIndex [ 0, 2, 9, 5, 0, -1, 3, 0, 5, 11, 3, -1,
                     1, 3, 11, 7, 1, -1, 2, 1, 7, 9, 2, -1
                     0, 3, 10 ,4, 0, -1, 3, 1, 6, 10, 3, -1,
                     2, 8, 6, 1, 2, -1, 2, 0, 4, 8, 2, -1,
                     9, 8, 4, 5, 9, -1, 5, 4, 10, 11, 5, -1,
                     11, 10, 6, 7, 11, -1, 7, 6, 8, 9, 7, -1
                    ]
    }
```



IndexedFaceSet

- **Colours**
  - colour per face

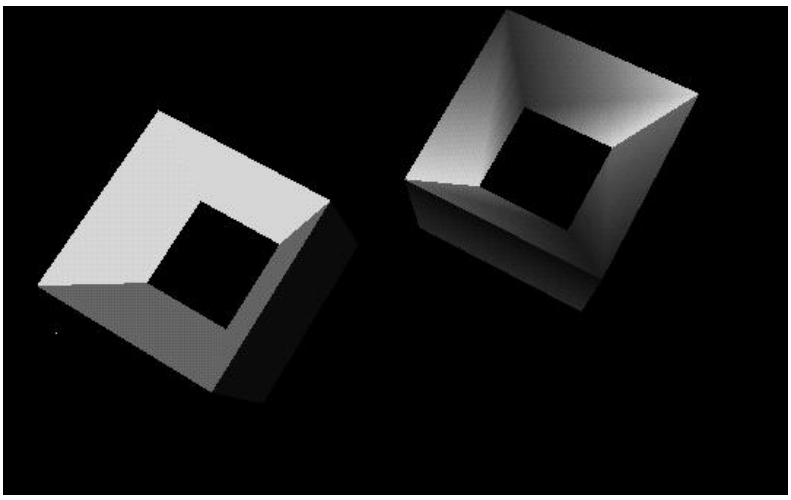
```
color Color { color [ 1 1 1, 0.3 0.3 0.3]}
colorPerVertex FALSE
colorIndex [ 0, 0, 0, 0, 0, 0,
             0, 0, 1, 1, 1, 1]
```

12 colours = 1 per face

  - OR colour per vertex

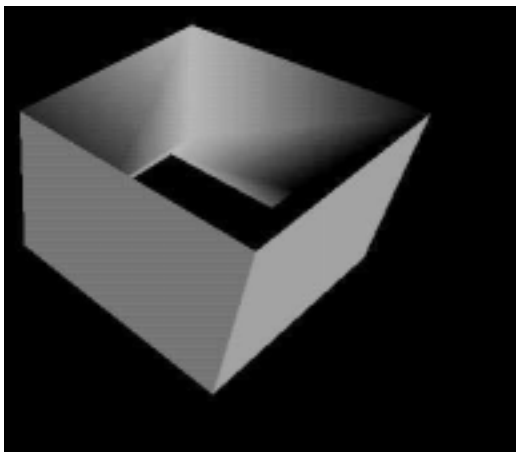
```
color Color { color [ 1 1 1, 0.3 0.3 0.3]}
colorPerVertex TRUE
colorIndex [ 0, 1, 1, 0, 0, -1, 1, 0, 0, 1, 1, -1,
             0, 1, 1, 0, 0, -1, 1, 0, 0, 1, 1, -1
             0, 1, 1 ,0, 0, -1, 1, 0, 0, 1, 1, -1,
             0, 1, 1 ,0, 0, -1, 1, 0, 0, 1, 1, -1,
             0, 1, 1 ,0, 0, -1, 1, 0, 0, 1, 1, -1,
             0, 1, 1 ,0, 0, -1, 1, 0, 0, 1, 1, -1
             ]
```

72 indices = 1 for each vertex in the face set definition



IndexedFaceSet

- **Normal**
  - defines the direction of each point and the way it reflects light
  - default normal is perpendicular to surface of polygon
  - Polygon is flat if all normals point same way
  - Each smooth shading

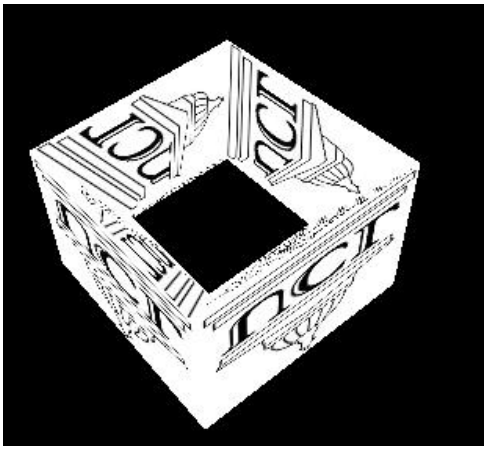


```
normal Normal {
  vector [
          -1 0 -1, 1 0 1, 1 0 -1, -1 0 1,
          1 0 0, -1 0 0, 0 0 1, 0 0 -1
          ]
}
normalPerVertex TRUE
normalIndex [
          5, 7, 7, 5, 5, -1, 6, 5, 5, 6, 6, -1,
          4, 6, 6, 4, 4, -1, 7, 4, 4, 7, 7, -1
          5, 6, 6 ,5, 5, -1, 6, 4, 4, 6, 6, -1,
          7, 7, 4, 4, 7, -1, 7, 5, 5, 7, 7, -1,
          1, 1, 1, 1, 1, -1, 2, 2, 2 ,2, 2, -1,
          0, 0, 0, 0, 0, -1, 3, 3, 3, 3, 3, -1
         ]
```

IndexedFaceSet

- **Texture Coordinates**
  - specify the position on the texture map of each point on the shape
  - fractional points are allowed
  - measure from bottom left of image
- **Ugly example:**

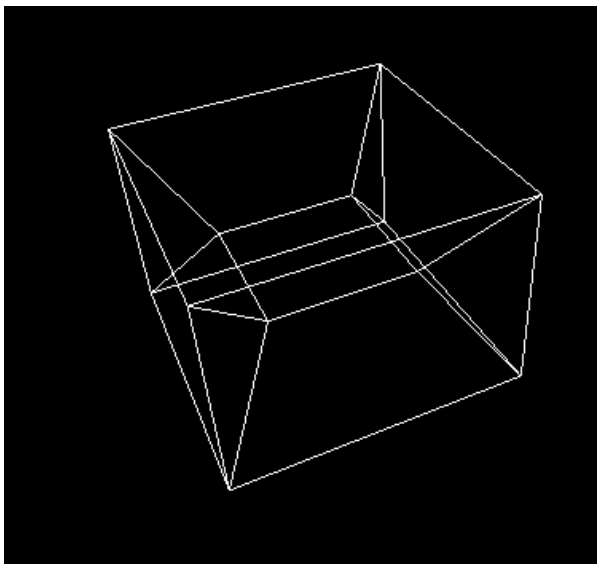
```
texCoord TextureCoordinate  {
   point [ 0 0, 0 1, 1 1 , 1 0 ]
}
texCoordIndex [
                    0, 1, 2, 3, 0, -1, 0, 1, 2, 3, 0, -1,
                    0, 1, 2, 3, 0, -1, 0, 1, 2, 3, 0, -1,
                    0, 1, 2, 3, 0, -1, 0, 1, 2, 3, 0, -1,
                    0, 1, 2, 3, 0, -1, 0, 1, 2, 3, 0, -1,
                    0, 1, 2, 3, 0, -1, 0, 1, 2, 3, 0, -1,
                    0, 1, 2, 3, 0, -1, 0, 1, 2, 3, 0, -1
                ]
```



IndexedLineSet

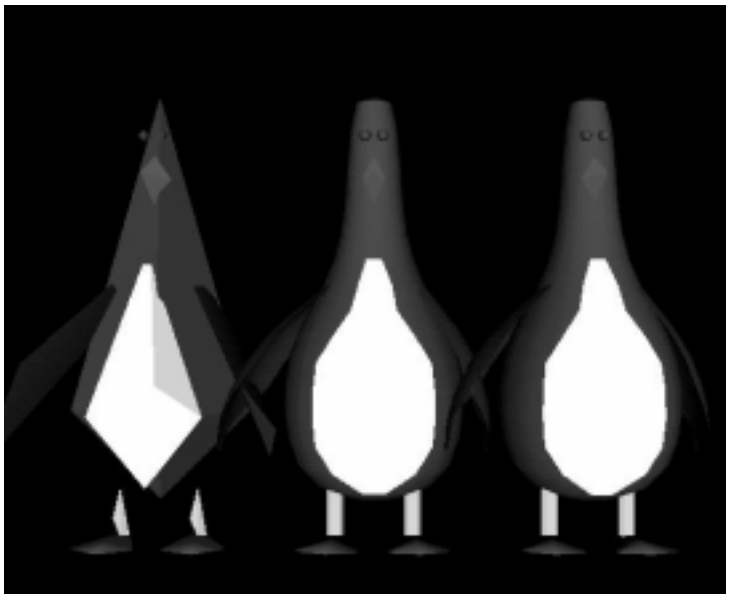
- **Defines a line set**
- **Colour per line or per vertex**

```
geometry IndexedLineSet {
  coord USE COORDS
  coordIndex [ 0, 4, 5, 0, -1, 3, 10 , 11, 3, -1,
                    1, 6, 7, 1, -1, 2, 8, 9, 2, -1,
                    0, 2, 1, 3, 0, -1,
                    4, 8, 6, 10, 4, -1,
                    5, 9, 7, 11, 5, -1]
  color Color { color 1 1 1}
  colorPerVertex FALSE
  colorIndex [ 0, 0, 0, 0, 0, 0, 0, 0,
                  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
}
```

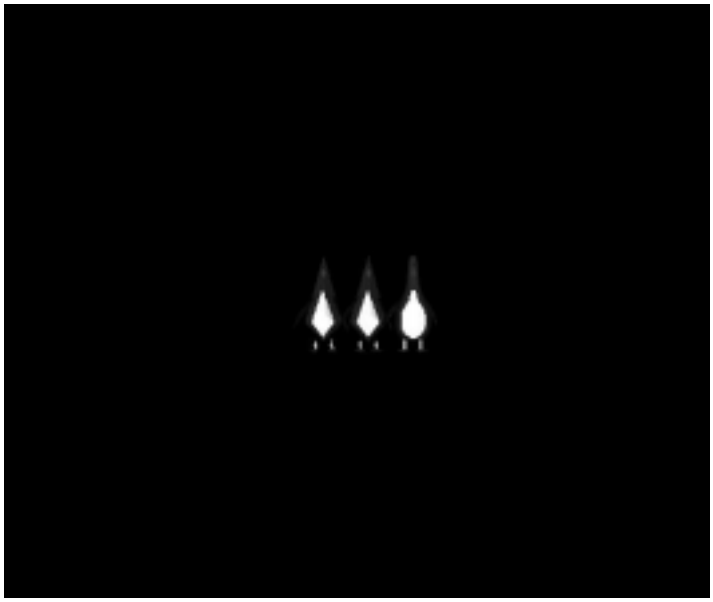


LOD

- **Nearby View**



- **Distant View**



- **Penguin1 = 1000 polygons**
- **Penguin2 = 88 polygons**

Billboards

- **Billboards**
  - object always faces the viewer

```
Transform {
  translation 0 3 0
  children [
    Billboard {
      children [
        USE UCL_BOX
        ]
    }
   ]
}
```

Creating VRML Worlds

Part3: Behaviour

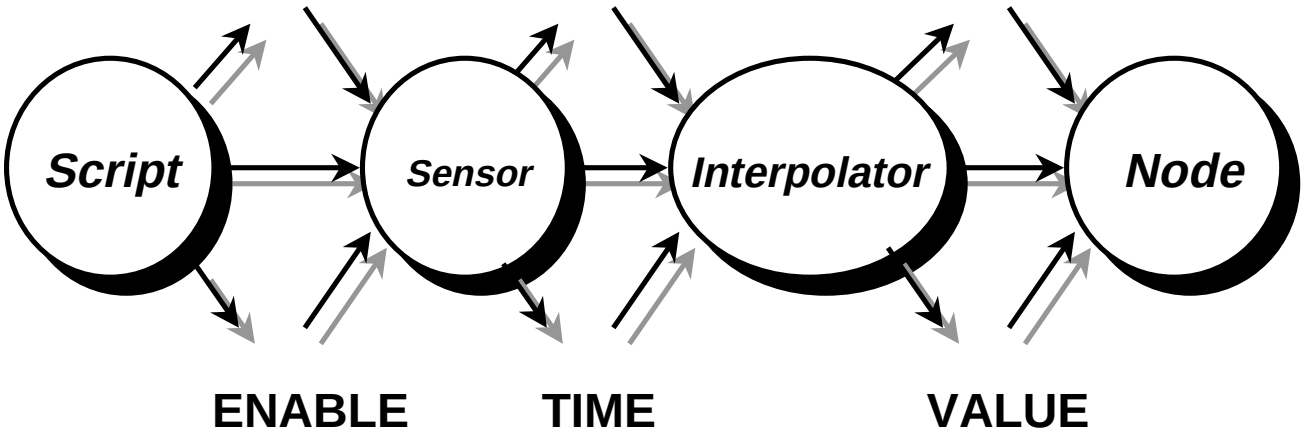
- Data Flow
- Field Revision
- Interpolators
- Sensors
- Scripts
- In-Depth Example

Anthony Steed

Department of Computer Science
University College London

Data Flow

- **Complex behaviours controlled by data flow between nodes**



Typical Use of Data Flow Mechanism

- **Components**
  - Sensors - react to the user
  - Interpolators - generate field values
  - Scripts - general value processing
  - Nodes - destination of new field values

Field Revision

- **Recall Nodes are made up of fields (which might be other nodes)**
- **Fields come in four classes**
  - field
  - exposedField
  - eventOut
  - eventIn
- **a plain field may only be set at initialisation**
- **eventIn are fields that can be set by the data flow**
- **eventOut are fields that can not be set, only read**
- **exposedFields can be set at initialisation and set and read at run-time**

Field Types

- **The definition of nodes and data flow use the following types**

  SFBool
  SFVec2f
  SFVec3f
  SFRotation
  SFFloat
  SFNode
  SFString
  SFColor
  SFime
  SFImage
  SFInt32

- **There are similar types for multiple values of some type**

  MFColor
  MFFloat
  MFInt32
  MFNode
  MFRotation
  MFString
  MFTime
  MFVec2f
  MFVec3f

Field Examples

- **Box**

  Box {

  field SFVec3f size 2 2 2

  }

  – At run time the size can not be changed

- **Shape**

  Shape {

  exposedField SFNode appearance NULL
  exposedField SFNode geometry NULL

  }

  – At run time the appearance or geometry nodes can be changed for other similar nodes

- **OrientationInterpolator**

  OrientationInterpolator {

  eventIn SFFloat set_fraction

  eventOut SFRotation value_changed

  exposedField MFFloat key []

  exposedField MFRotation keyValue [] }

  – the current fraction can not be read

  – the current rotation can not be set

  – neither of these has an initial value

Outline Data Flow

- **Data flows from eventOut or exposedField to eventIn or exposedField**
- **ROUTE statements identify the node and field pairs to be connected**
- **Names are those declared by DEF**

- **Notes**
  - Fan in and Fan out of routes is allowed
  - Events are processed in time order
  - Circular route graph are avoided

Example

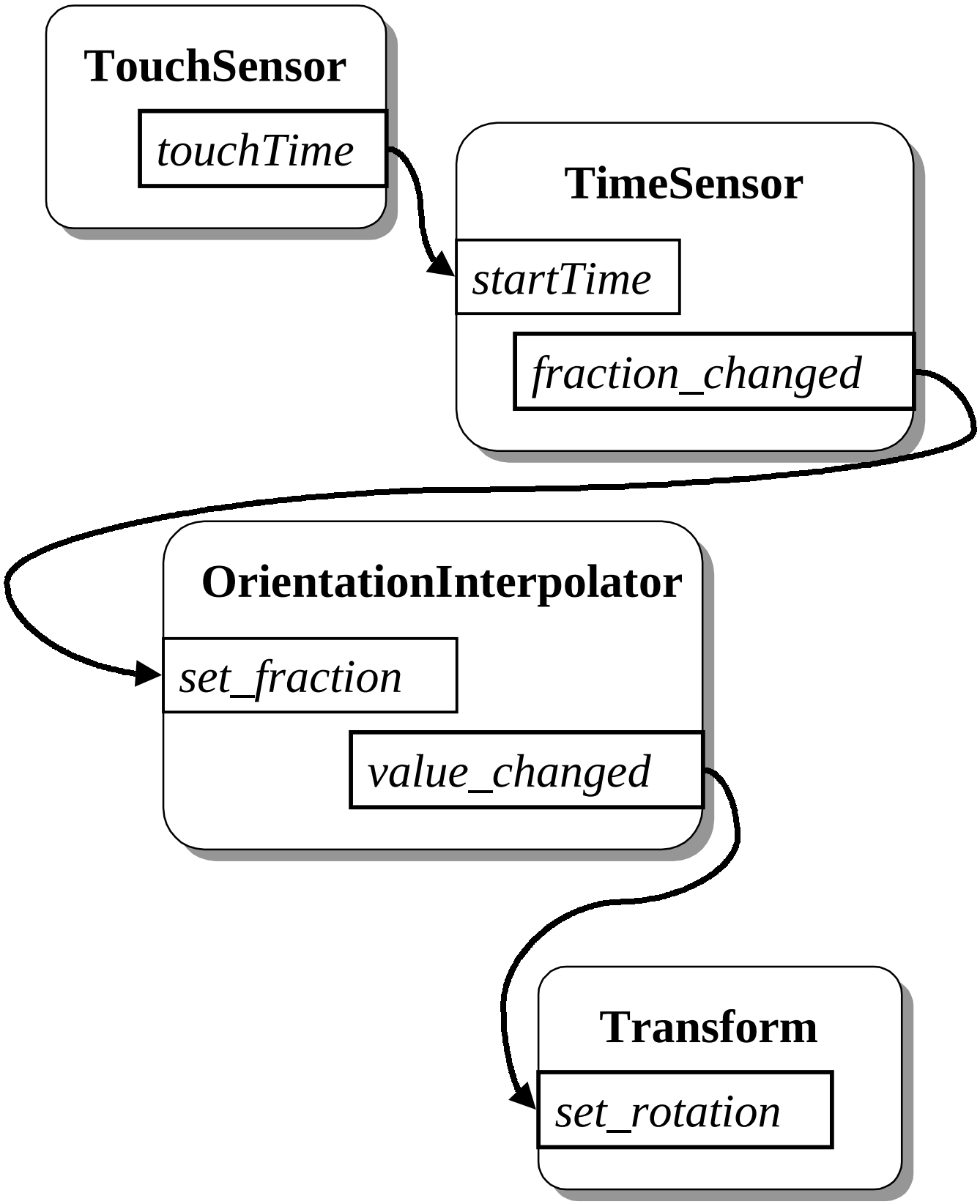
```
DEF TRANS Transform {
  children [
    DEF TOUCH TouchSensor {}
    Group {
     children [  USE UCL_BOX ]
    }
   ]
}
DEF TIMER TimeSensor {
  loop FALSE
  cycleInterval 2.0
}
DEF ROTATOR OrientationInterpolator {
  key [0, 0.5]
  keyValue [0 1 0 0, 0 1 0 3.141]
}
ROUTE TOUCH.touchTime TO TIMER.startTime
ROUTE TIMER.fraction_changed TO ROTATOR.set_fraction
ROUTE ROTATOR.value_changed TO TRANS.rotation
```

- **When the user click on the object it rotates once**

Components of Example

- **TouchSensor**
  - generates a time event when the user click
- **TimeSensor**
  - we set the start time for the timer
  - it runs for 2 seconds
  - loop is FALSE so it runs once
  - it generates a fraction between 0 and 1
- **OrientationInterpolator**
  - takes a value between 0 and 1
  - generates an orientation value from a key value set
- **Transform**
  - has a rotation field that we set

Example Data Flow Graph



Interpolators

- **ColorInterpolator**
  - generate sets of colours to apply to objects
- **CoordinateInterpolator**
  - generate points with which to build objects (morphing)
- **NormalInterpolator**
  - generate normals for polygons
- **OrientationInterpolator**
  - generate an orientation value
- **PositionInterpolator**
  - generate a position value
- **ScalarInterpolator**
  - generate a scalar value (value)

ScalarInterpolator

ScalarInterpolator {

eventIn SFFloat set_fraction

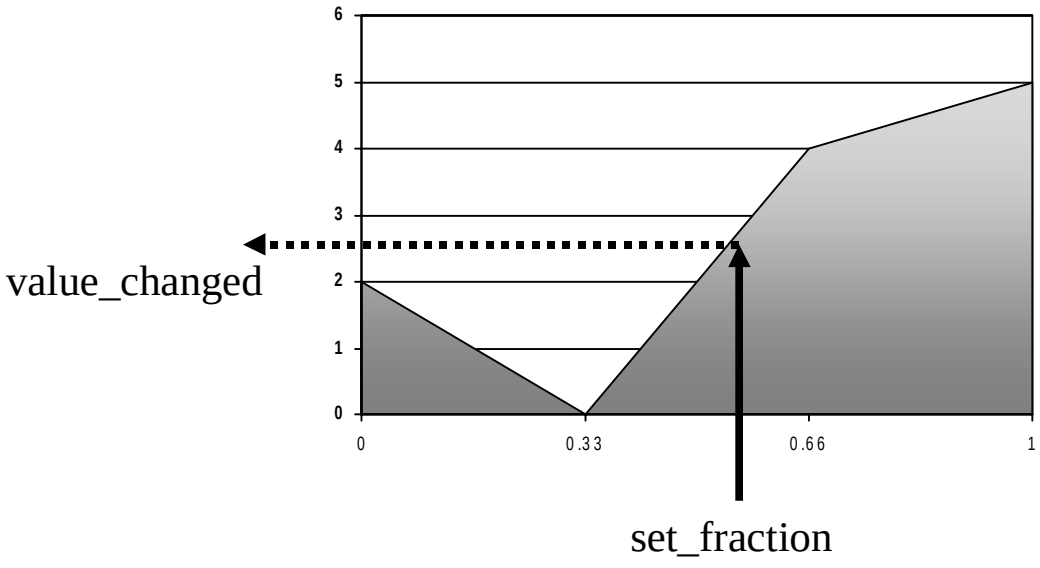
exposedField MFFloat key []

exposedField MFFloat keyValue []

eventOut SFFloat value_changed

}

- **key and keyValue defined a graph**
  - number of keys = number keyValues
- **Example**
  - key [0, 0.33, 0.66, 1]
  - keyValue [2.0, 0.0, 4.0, 5.0]



Position & Orientation Interpolators

- **All interpolators work in a similar fashion to ScalarInterpolator**
- **PositionInterpolator interpolates SFVec3f, that is spatial positions e.g.**

```
PositionInterpolator {
    key [0, 0.25, 0.5, 0.75, 1]
    keyValue [ 0 0 0, 0 5 0, 0 0 0, 0 -5 0, 0 0 0]
}
```

  - Object oscillates about the origin
- **Orientation Interpolator interpolates SFRotation**

```
OrienationInterpolator {
    key [0, 0.25, 0.5, 0.75, 1]
    keyValue [ 0 1 0 0, 0 1 0 1.57, 0 1 0 0,
                0 1 0 -1.57, 0 1 0 0]
}
```

  - Object turns to the right, then to the left, then back to the front.

Sensors

- **TimeSensor**
  - continual output of time values during a defined active period
- **“drag sensors”**

  react to user input when clicked upon
  - TouchSensor
  - PlaneSensor
  - SphereSensor
  - Cylinder Sensor
  - Anchor
- **“scene sensors”**
  - VisibilitySensor
    - generates events when user sees the object
  - ProximitySensor
    - generates events when user approaches
  - Collision
    - generates events upon user collision

TimeSensor

TimeSensor {
exposedField SFTime cycleInterval 1
exposedField SFBool enabled TRUE
exposedField SFBool loop FALSE
exposedField SFTime startTime 0
exposedField SFTime stopTime 0
eventOut SFTime cycleTime
eventOut SFFloat fraction_changed
eventOut SFBool isActive
eventOut SFTime time
}

- **Has quite complex behaviour!**
  - it starts looping at startTime
    - measured in seconds since date in 1970
  - it stops looping at stopTime
    - doesn't stop if stopTime < startTime
  - it stops after one loop if loop=FALSE
  - during a loop it generates fraction_changed events from 0 to 1
  - at the start of each cycle it generates cycleTime
  - the current time is always output
  - it can be turned on or off with the enabled field
  - isActive sends TRUE when the TimeSensor starts and FALSE when it stops

TimeSensor

- **Many uses**
  - drive animations using Interpolators
    - such as human gait pattern
  - animation can constantly loop
    - rotating signs
  - or run just once
    - door opening
  - can be used to schedule events in the future
- **See the previous example**
  - How would we get the box to
    - Speed up?
    - Turn twice?
    - (V. HARD) Turn then wait 10 seconds before turning again when clicked?

TouchSensor

TimeSensor {
exposedField SFBool enabled TRUE
eventOut SFVec3f hitNormal_changed
eventOut SFVec3f hitPoint_changed
eventOut SFVec2f hitTextureCoord_changed
eventOut SFBool isActive
eventOut SFBool isOver
eventOut SFTime touchTime
}

- **Generates events when the user touches or clicks on some geometry**
  - the geometry that is used is everything below or to the right of the TouchSensor in the scene graph
- **Events generated are the**
  - position of the click
  - the normal of the object at the click
    (see discussion of normals in lecture 3)
  - the texture coordinate of the texture at that point
    (see lecture 3)
  - the time of the click
  - a boolean event when the click occurs
  - a boolean event when the “cursor” first goes over the object

Grab Sensors

- **Grab sensors**
  - when the user clicks on the object, further movement of the mouse generates events that map mouse movements into 3D movements in the scene
  - PlaneSensor and CylinderSensor have limits of movement
- **Plane**
  - movements of the mouse are mapped onto a plane (linear movement in two dimensions)
- **Sphere**
  - movements of the mouse are mapped onto a sphere (free rotation)
- **Cylinder**
  - mouse movements are mapped into rotation in one dimension
  - useful for levers and switches

Scene Sensors

- **VisibilitySensor**
- **ProximitySensor**
- **Collision**

- **Visibility Sensor**
  - this and ProximitySensor are mainly used to start and stop complex behaviour that can not be seen or heard by the user
  - generates events when the user is first able to see, or is obscured from a part of the scene
  - used liked other sensors such as TouchSensor
  - (not widely implemented in browser yet?)

Scene Sensors

- **Collision**
  - prevention of avatar-scene intersection
  - depends on some details of NavigationInfo
  - proxy geometry (e.g. low-complexity) nodes can be substituted

```
Collision {
  children [
    UclBox { translation  1 0 -1 }
    UclBox { translation -1 0 -1 }
  ]
}
```

Far boxes are solid

```
Collision {
  collide FALSE
  children [
    UclBox { translation -1 0 1 }
    UclBox { translation  1 0 1 }
   ]
}
```

Near boxes are not

Scene Sensors

- **ProximitySensor**
  - based on distance of user from object
  - when “proximate” generates the position and orientation of the user

```
DEF TRANS Transform {
 children [
   DEF PROX ProximitySensor {
     size 10 10 10
   } ,
   Group {
     children [
          UclBox {  boxSize 0.8 0.5 0.8 }
         ]
   }
  ]
}

DEF TIMER TimeSensor { cycleInterval 2.0 }
DEF ROTATOR OrientationInterpolator {
 key [0, 0.5]
 keyValue [0 1 0 0, 0 1 0 3.141] }

ROUTE PROX.enterTime TO TIMER.startTime
ROUTE TIMER.fraction_changed TO ROTATOR.set_fraction
ROUTE ROTATOR.value_changed TO TRANS.rotation
```

Scripts

- **Scripts provide general data flow processing nodes**
- **Part of their declaration states what eventIn and eventOut fields they contain**
- **The actual script is written in Java or JavaScript**
- **The script get notified when events are received at the node, and can post events to be propagated through the data flow**
- **For each scripting language there is a programming interface that allows the script to access the VRML model in the browser**
- **For each field type there are procedures and accessers to manipulate the values**
  - e.g. if value is a SFVec3f
    - l = value.length();
    - y = value.y;

Script API Capabilities

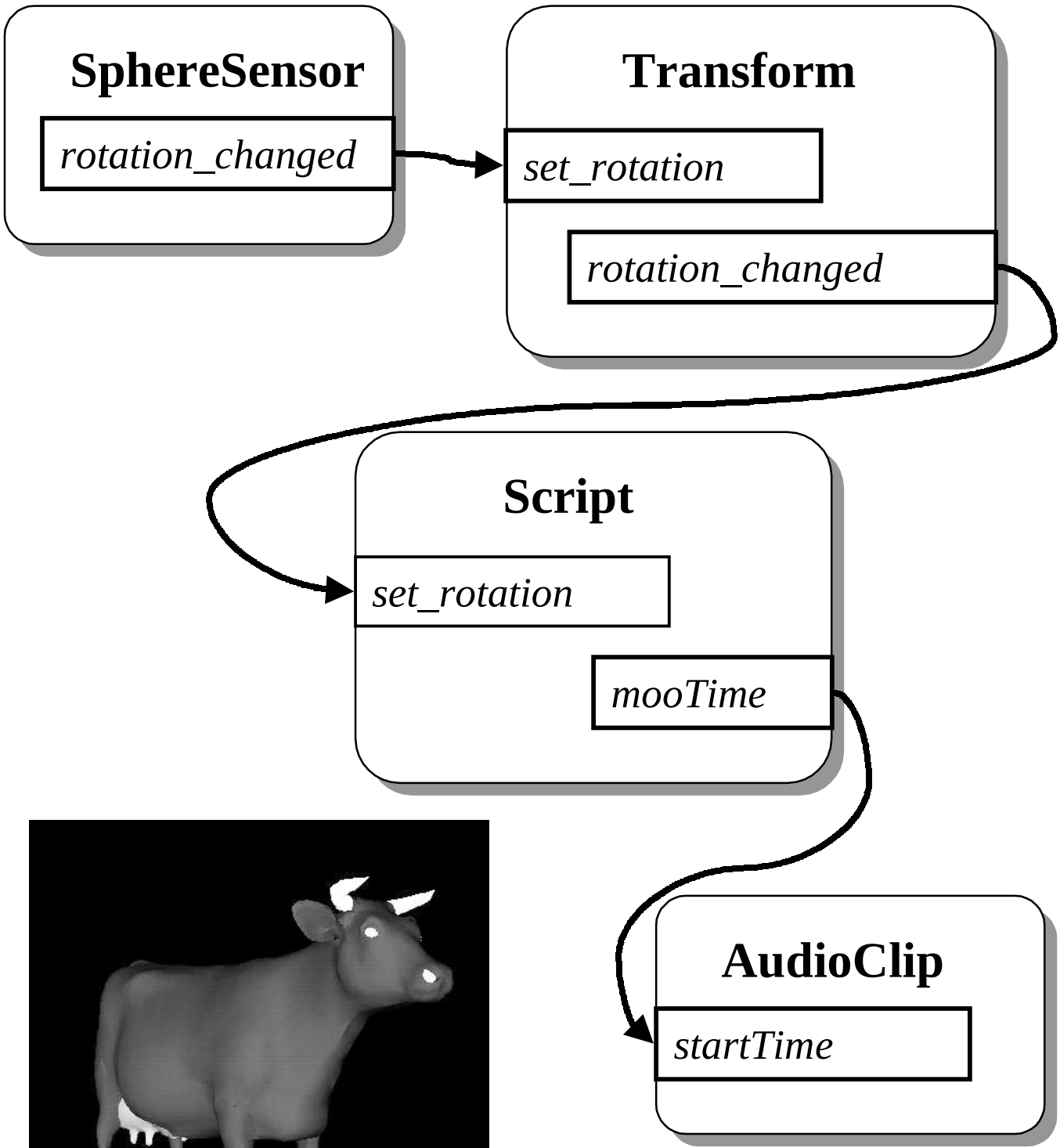
- **If a field it receives is a Node then the script may access the readable parts of that Node and alter then directly**
- **May add and delete routes between Nodes**
- **May query the Browser to get frame rate, speed of navigation**
- **Can create new VRML**
  - createVRMLFromString
  - createVRMLFromURL
- **Can replace the whole world**

Script and Sensor Example

```
#VRML V2.0 utf8
Group {
  children [
    DEF ROT SphereSensor {
    }
    ,
    DEF TRANS Transform {
      translation 0 0 0
      children [
        Inline {
          url "cow-model.wrl"
        },
        Sound {
         source DEF
                    MOO_SOUND
              AudioClip {
                 url "moo.wav"
              }
        }
       ]
    }
   ]
}

ROUTE ROT.rotation_changed
      TO TRANS.set_rotation
```

```
DEF MOOER Script  {
  eventIn SFRotation set_rotation
  eventOut SFTime mooTime

  field SFBool last_updir TRUE
  field SFBool first TRUE
  field SFBool lastup TRUE
  url ”javascript:

function set_rotation(value,
      timestamp) {
  updir = value.multVec(new
     SFVec3f(0,1,0));
  up = (updir.y > 0.0);
  if (first) {
    lastup = up;
    first = false;
  }  else {
    if ((!lastup) && up) {
      mooTime = timestamp;
    }
    lastup = up;
  }
}
"
}

ROUTE TRANS.rotation_changed
      TO MOOER.set_rotation
ROUTE MOOER.mooTime TO
      MOO_SOUND.startTime
```

Script and Sensor Example



Other VRML Components

- **Sound**
  - used in example
- **AudioClip**
  - used in examples
- **Viewpoint**
  - define a set of camera positions from which to view the scene
- **NavigationInfo**
  - specify a desktop navigation metaphor
- **MovieTexture**
  - movie applied as texture to an object
  - can also act as a sound source
- **PixelTexture**
  - small textures within the VRML file
- **WorldInfo**
  - miscellaneous creator info (including a title)

Conclusions

- **Complex example for interested programmers**
  - VRML Boids (see web page below)
- **Resources**
  - Examples from these lectures

    http://www.cs.ucl.ac.uk/staff/A.Steed/vrml
- **Internet Resources**

  http://cosmosoftware.com

  http://vrml.sdsc.edu

  http://vrml.miningco.com