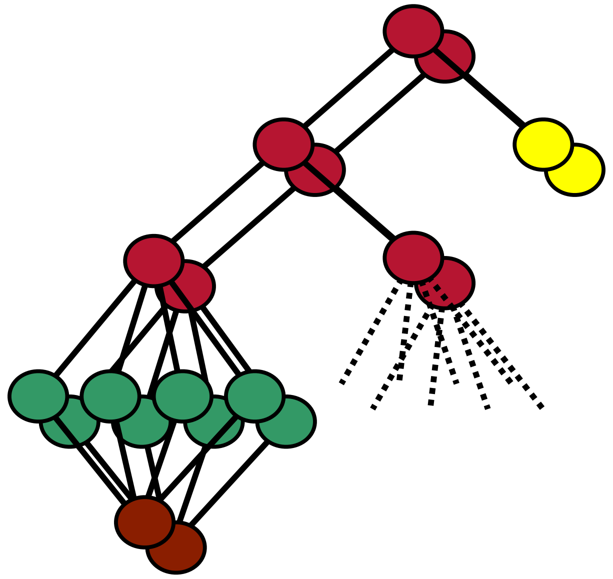
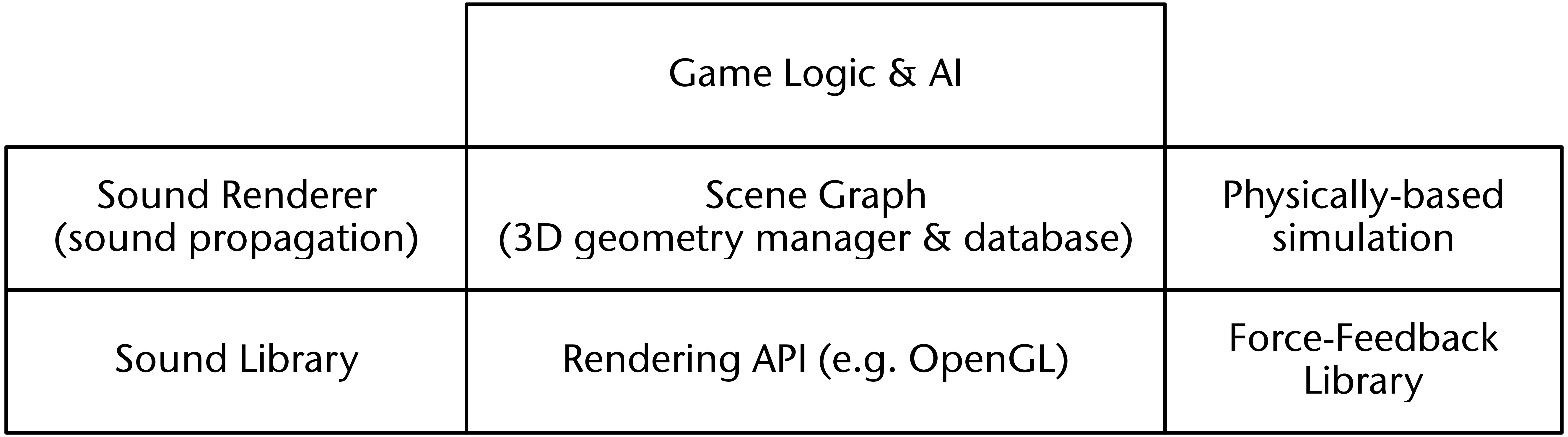
Virtual Reality & Physically-Based Simulation

Scenegraphs & Game Engines



G. Zachmann
University of Bremen, Germany
cgvr.cs.uni-bremen.de

Overall VR System Architecture (Example)



Game Logic & AI

Sound Renderer (sound propagation)

Scene Graph (3D geometry manager & database)

Physically-based simulation

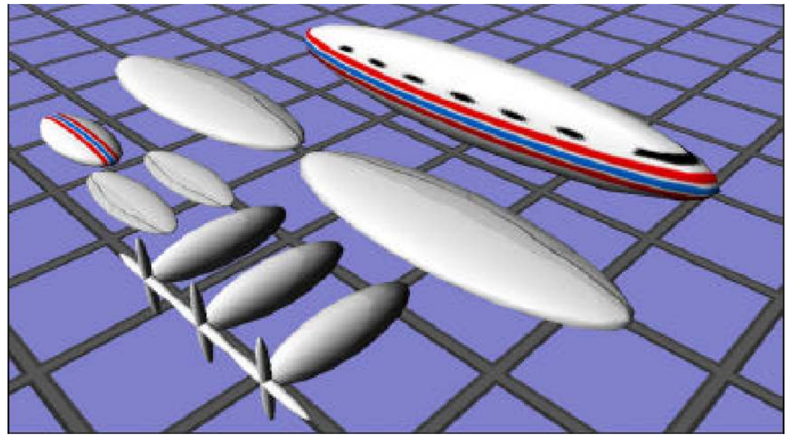
Sound Library

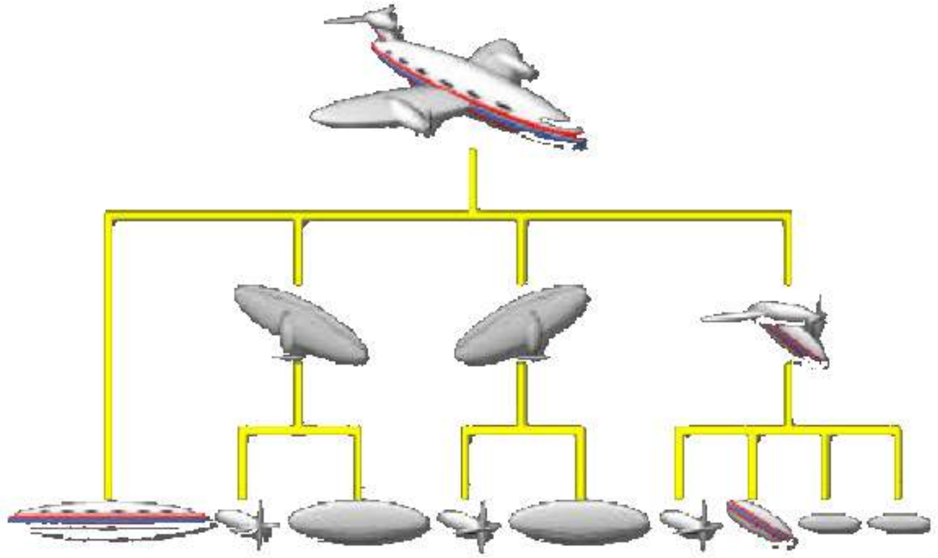
Rendering API (e.g. OpenGL)

Force-Feedback Library

Motivation

- Immediate mode vs. retained mode:
  - Immediate mode = OpenGL / Direc3D = Application sends polygons / state change commands to the GPU → flexibler
  - Retained mode = scenegraph = applications builds pre-defined data structures that store polygons and state changes → easier and (probably) more efficient rendering
- Flat vs. Hierarchical scene descriptions:

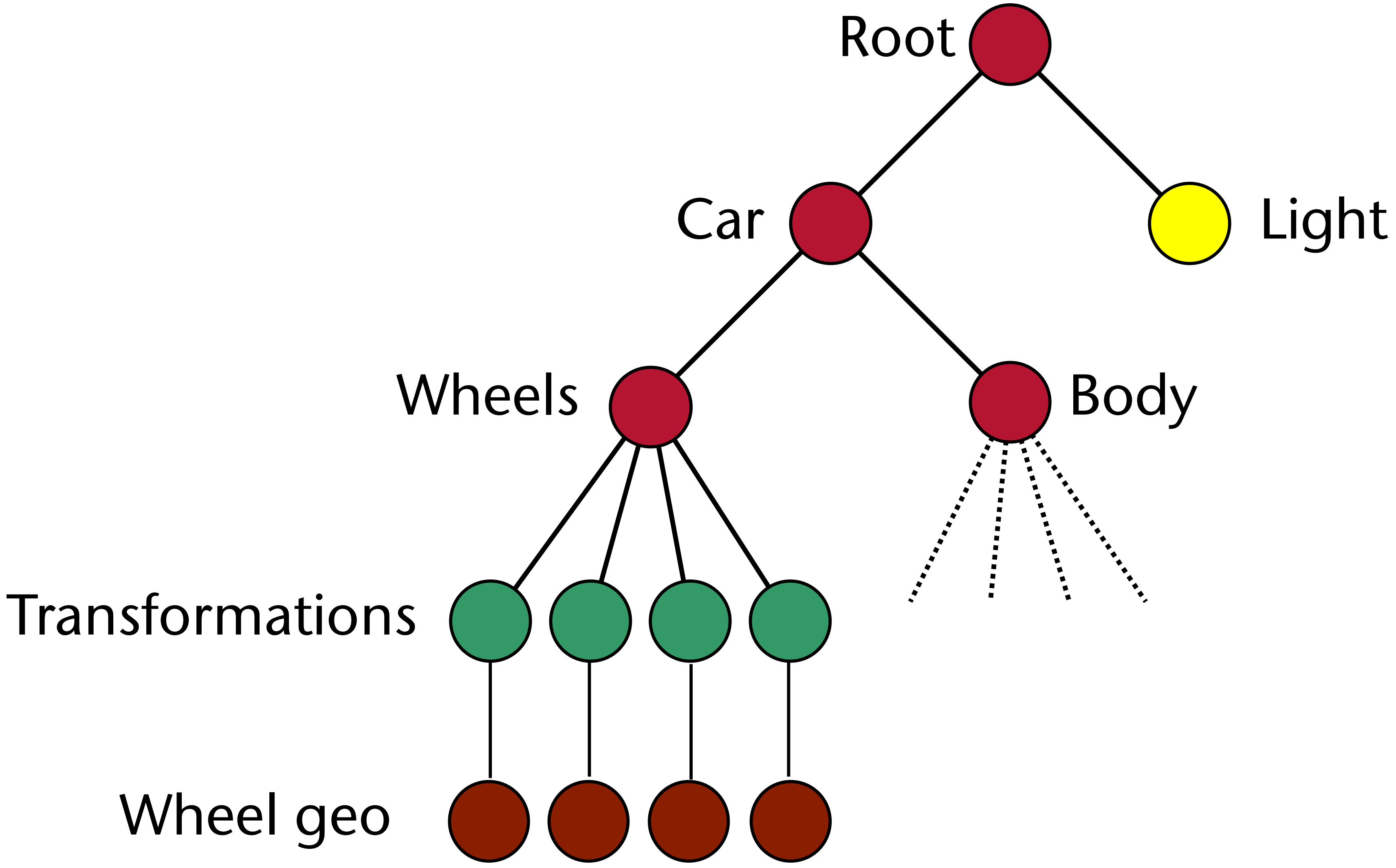




- *Code re-use* and *knowledge re-use*!
- Descriptive vs. imperative (cv. Prolog vs. C)
  - Thinking objects ... not rendering processes

Structure of a Scene Graph

- Directed, acyclic graph (DAG)
  - Often even a proper tree
- Consists of heterogeneous nodes
- Example:



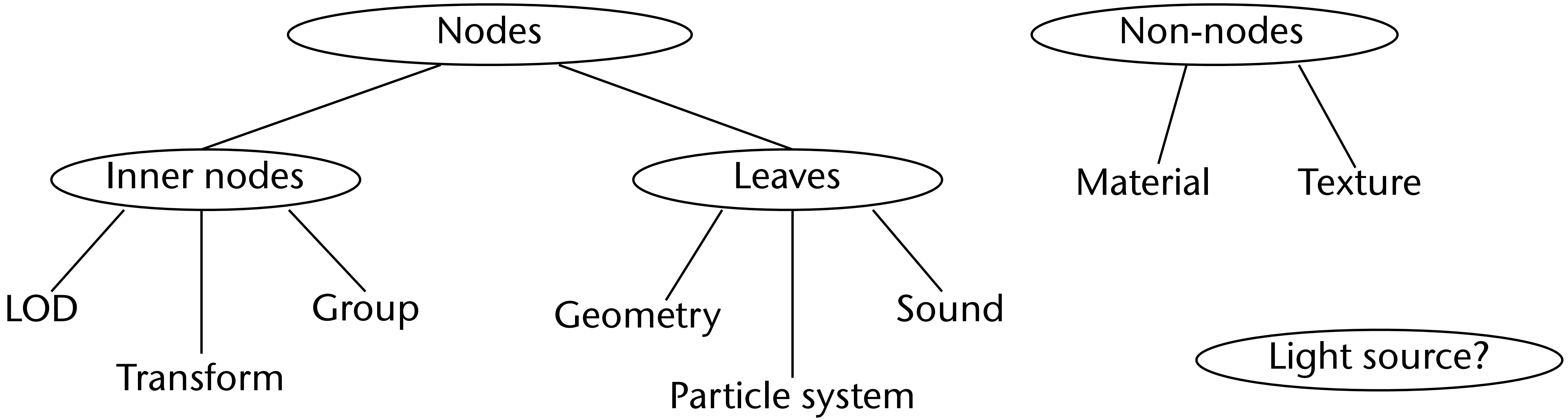
- Most frequent operation on scene graph: rendering
  - Amounts to depth-first traversal
  - Operation per node depends on kind of node

Semantics of the Elements of a Scenegraph

- Semantics of a node:
  - Root ="universe"
  - Leaves = "content" (geometry, sound, ...)
  - Inner nodes = forming groups, store state (changes), and other non-geom. functionality, e.g., transforms
- Grouping: criteria for grouping is left to the application, e.g., by
  - Geometric proximity ⟶ scenegraph induces a nice BVH
  - Material ⟶ good, because state changes cost performance!
  - Meaning of nodes, e.g., all electrical objs in the car under one node → good for quickly switching off/on all electrical parts in the car
- Semantics of edges = inheritance of states
  - Transformation
  - Material
  - Light sources (?)

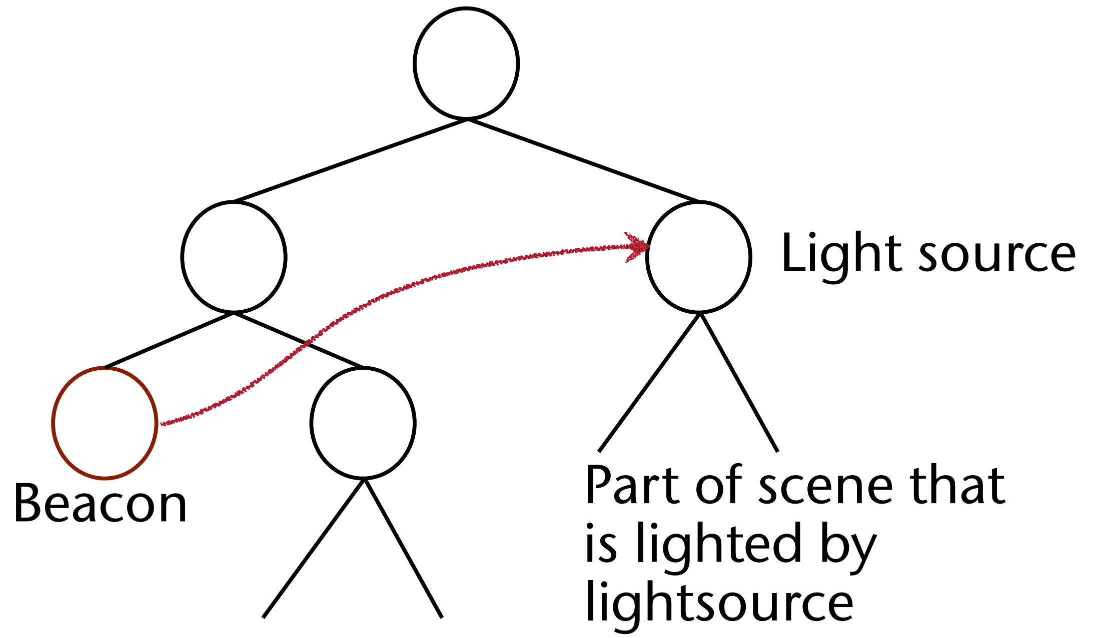
Kinds of Nodes

- There are 2 hierarchies: scenegraph hierarchy + class hierarchy
- The flexibility and the expressiveness of a scenegraph depends heavily on the kinds and number of node classes!

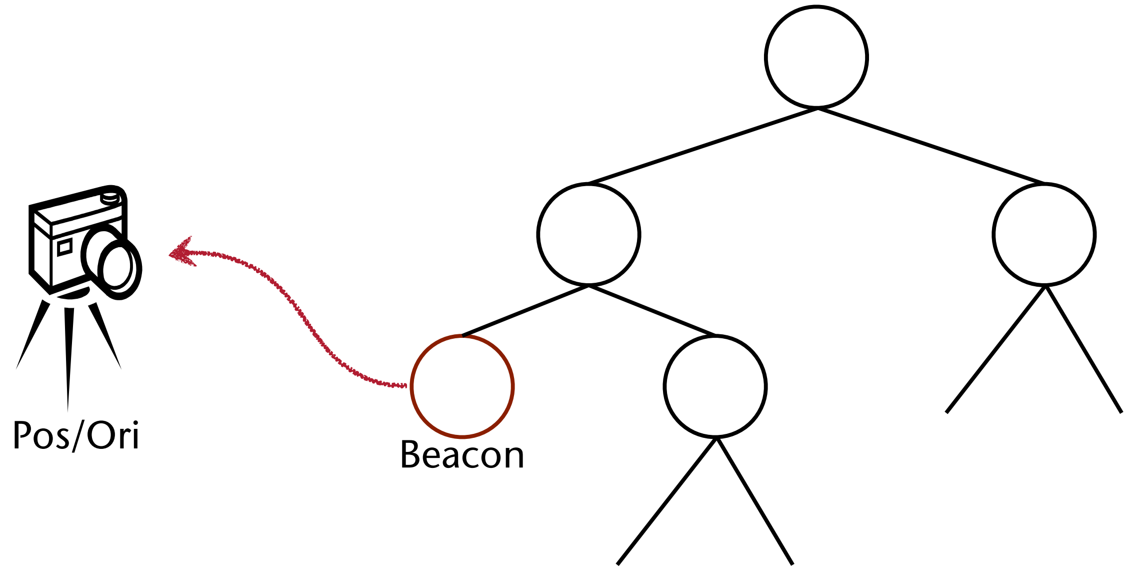


Special Elements of a Scene

- Light sources:
  - Usually part of the scenegraph
  - Problem with naïve semantics: what if light source should move/turn, but not the scene it shines on?
  - Solution: beacons
    - Light source node lights its sub-scene underneath
    - Position/orientation is taken from the beacon



- Camera: to be, or not to be a node in the scenegraph?
  - Both ways have dis-/advantages
  - If not a node: use beacon principle

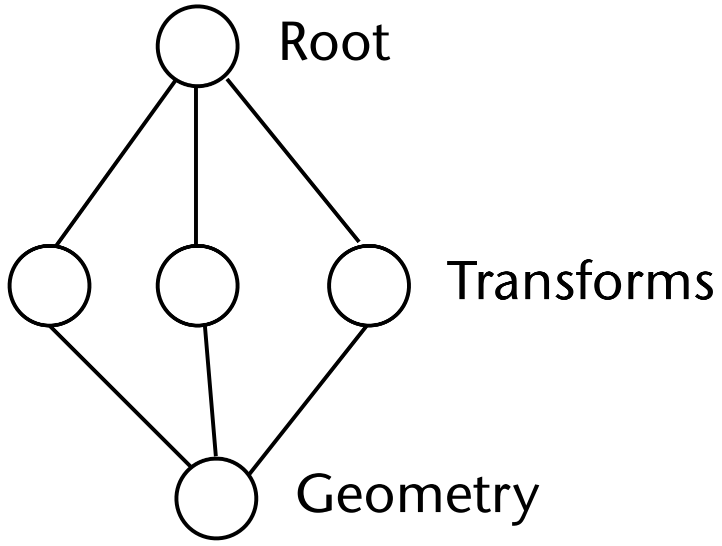


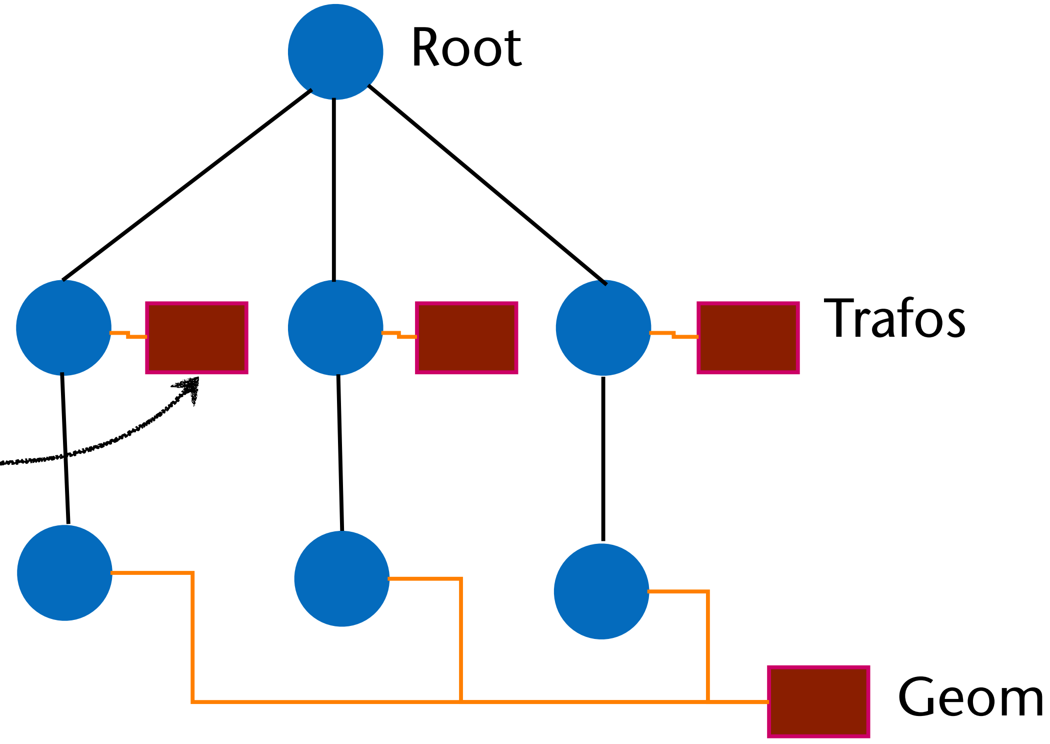
Material

- Material =
  - Color, texture, lighting parameters (see Phong)
  - Is a *property* of a node (not a node in the scenography, usually)
- Semantics of materials stored with inner nodes: top-down inheritance
  - Path from leaf to root should have at least one material
  - Consequence:
    - Each leaf gets rendered with a unique, unambiguously defined material
    - It's easy to determine it
- Bad idea (Inventor): inheritance of material from left to right!

Sharing of Geometry / Instancing

- Problem: large scenes with lots of identical geometry
- Idea: use a DAG (instead of tree)
  - Problem: pointers/names of nodes are no longer *unique/unambiguous*!
- Solution: separate structure from content
  - The tree proper now only consists of *one kind* of nodes
  - Nodes acquire specific properties/content by attachments / properties
- Advantages
  - Everything can be shared now
  - Many scenegraphs can be defined over the same content
  - All nodes can acquire lots of different properties/content





Thread-Safe Scenegraphs for Multi-Threading

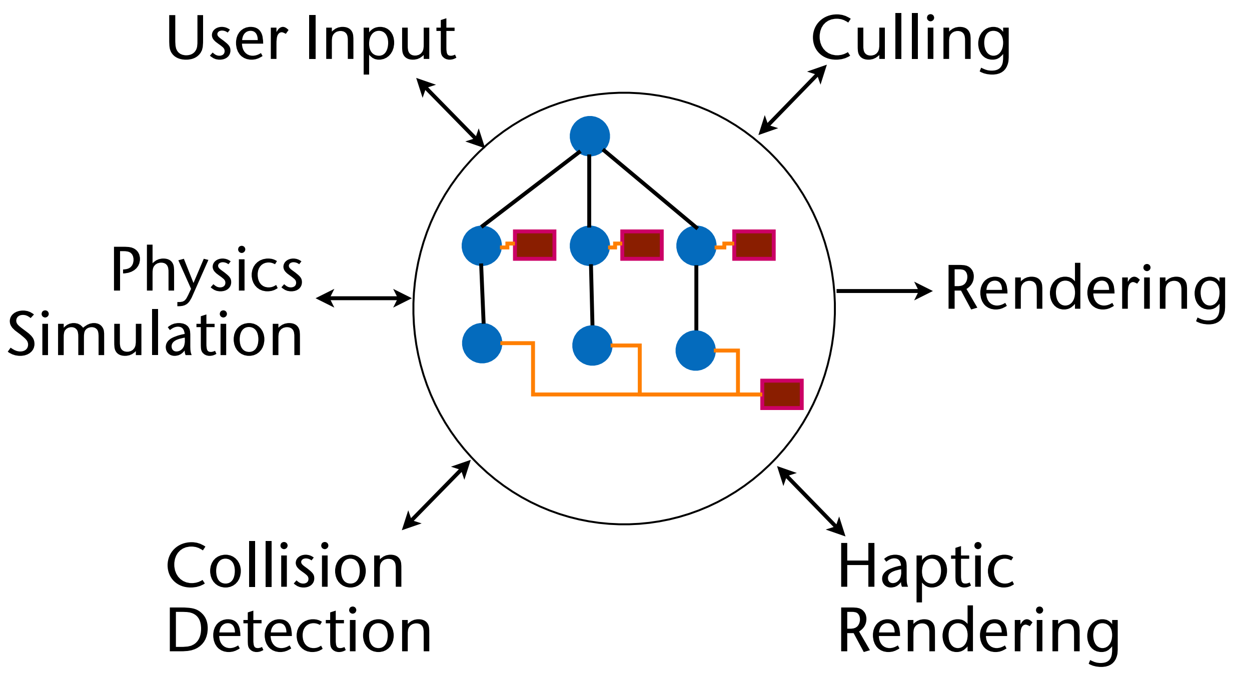
- Idea: several copies of the scenegraph
  - Problem: memory usage & sync!
- Solution:
  - *Copy-on-Write* of the attachments → "Aspects"
  - Each thread "sees" their own aspect
  - Problem: easy access via pointers, like

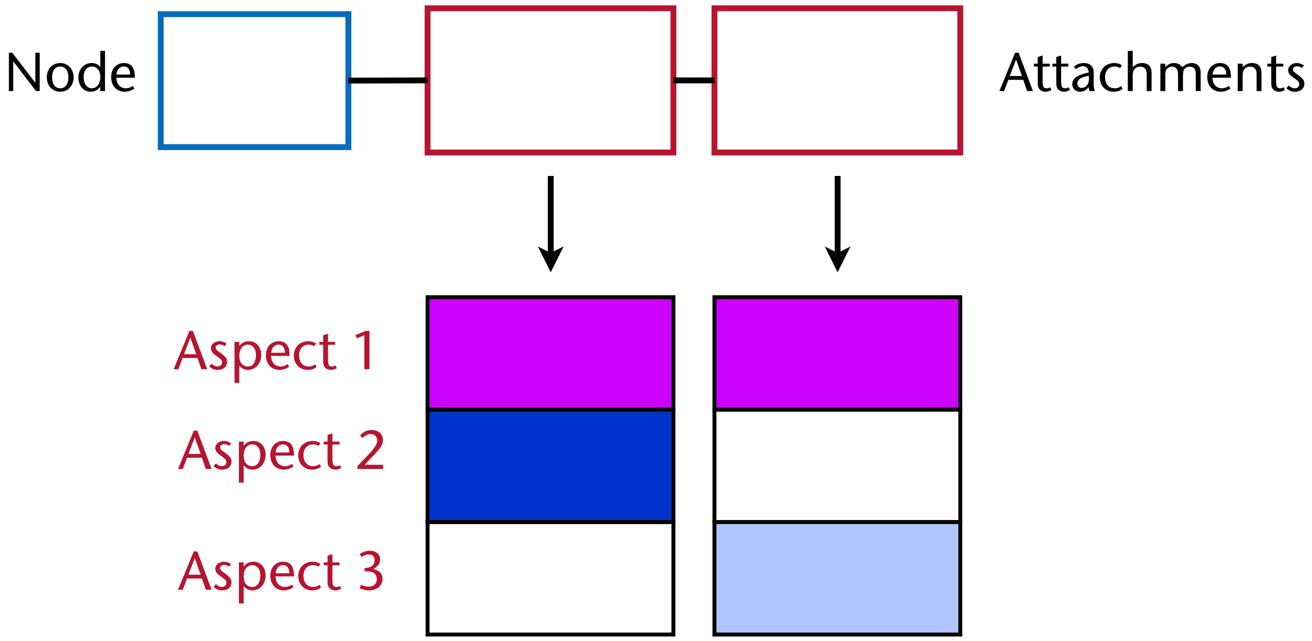
    `node->geom->vertex[0]`

    does not work any more
  - Solution (leveraging C++):
    - "Smart Pointers"
    - Needs one "pointer class" per node. Ex.:

      `geomptr = Geometry::create(…);`

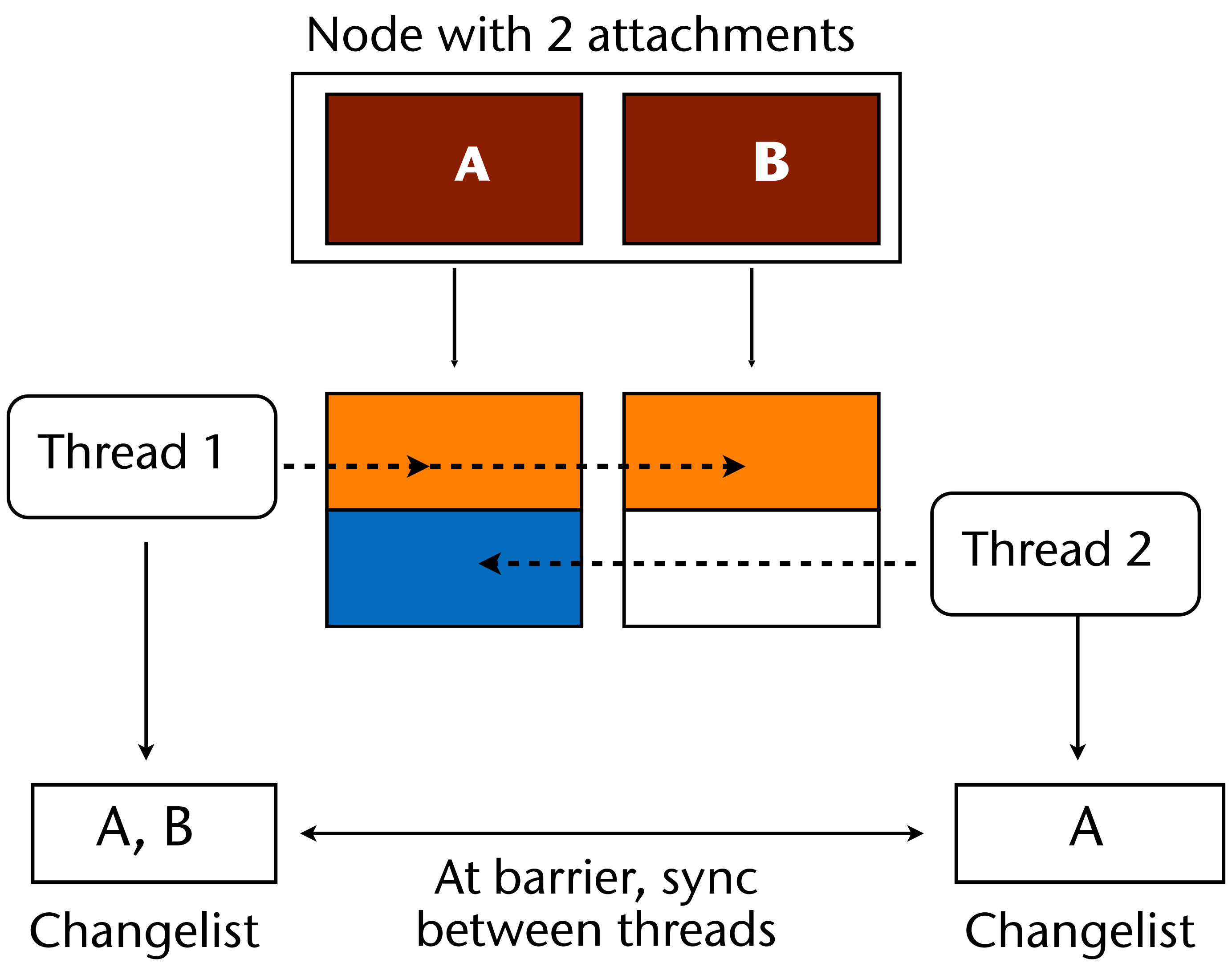
      `geomptr->vertex[0] ...`





Distributed Scenegraphs

- Synchronisation by changelists
  - Make scene graph consistent at one specific point during each cycle of each thread → barrier synchronization



- Distributed rendering:
  - Goal: distributed rendering on a cluster or multiple users
  - Problem: changes in the scenegraph need to be propagated
  - Solution: simply communicate the changelists
    - Items in the changelist = IDs of nodes/attachments to be changed + new data

- One simple (?) method to reduce network traffic: make the physics completely deterministic
  - Example: video game *Rocket League*

Criteria When to Use Scenegraphs

- When is a hierarchical organization of the VE effective:
  - Complex scenes: many hierarchies of transformations, lots of different materials, large environment with lots of geometry of which usually only a part can be seen (culling)
  - Mostly static geometry (opportunities for rendering optimization, e.g., LoD's)
  - Specific features of the scenegraph, e.g., particles, clustering, ...
- When not to use a hierarchical organization / scenegraph:
  - Simple scenes (e.g., one object at the center, e.g., in molecular visualization)
  - Visualization of scientific data (e.g., CT/MRI, or FEM)
  - Highly dynamic geometry (e.g., all objects are deformable)

Fields & Routes Concept by Way of X3D/VRML

- What is X3D/VRML:
  - Specification of nodes, each of which has a specific functionality
  - Scene-graph definition & file format, plus ...
  - Multimedia-Support
  - Hyperlinks
  - Behavior and animation
  - "VRML" = "Virtual Reality Modeling Language"
- X3D = successor & superset of VRML
  - Based on XML
- VRML = different encoding, but same specification
  - Encoding = "way to write nodes (and routes) in a file"

Hello World

- In X3D (strictly speaking: "XML encoding"):

Like the <html> tag in HTML →

Root node →

Definition of nodes →

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D profile='Immersive'>
<Scene>
 <Shape>
   <Text string="Hello" "world!" />
 </Shape>
</Scene>
</X3D>
```

Hello
world!

- In VRML:

No explicit root node in VRML →

```
#X3D V3.1 utf8
Shape {
 geometry Text {
   string [ "Hello" "world!" ]
 }
}
```

Tip: Use an ASCII editor wich identifies *matching brackets* as a text unit, and can jump to the other matching bracket

Nodes and Fields (aka. Entities and Components)

- Nodes are used for describing …
  - … the scenengraph: Geometry, Transform, Group, Lights, LODs, … (the usual suspects)
  - … the behavior graph, which implements all response to user input (later)
- Node := set of fields
  - "Single-valued fields" and "multiple-valued fields"
  - Each field of a node has a unique identifier
  - These are predefined by the X3D/VRML specification
- Field types:
  - field = actual data in the external file
  - eventIn, eventOut = used only for connecting nodes, data that won’t be saved in a file

Types of Fields

FYI

- All field types exist as "single valued" (**`SF`**...) and as "multiple valued" kind (**`MF`**...)
- Example of an SF field:

```
<Material diffuseColor="0.1 0.5 1" />
```

X3D

```
material Material {
  diffuseColor  0.1 0.5 1
}
```

VRML

- MF fields are practically the same as arrays
  - Special notation for signifying an MF field and to separate elements

FYI

- Primitive data types: the usual suspects

Field type	X3D example	VRML example
`SFBool`	`true / false`	`TRUE / FALSE`
`SFInt32`	`12`	`-17`
`SFFloat`	`1.2`	`-1.7`
`SFDouble`	`3.1415926535`	
`SFString`	`"hello"`	`"world"`

Reminder: for each SF-field there exists an MF-field

- Higher data types:

Field type	example
`SFColor`	`0 0.5 1.0`
`SFColorRGBA`	`0 0.5 1.0 0.75`
`SFVec3f`	`1.2 3.4 5.6`
`SFMatrix3f`	`1 0 0   0 1 0   0 0 1`
`SFString`	`"hello"`

FYI

- Special field types:

Field type	X3D example	VRML example
`SFNode`	**`<Shape> ... </Shape>`**	**`Shape { ... }`**
`MFNode`	**`<Shape>`**… , **`<Group>`**… oder **`<Transform>`**…	**`Transform {`** **`children [... ] }`**
`SFRotation`	**`0 1 0 3.1415`**	
`SFTime`	**`0`**	

FYI

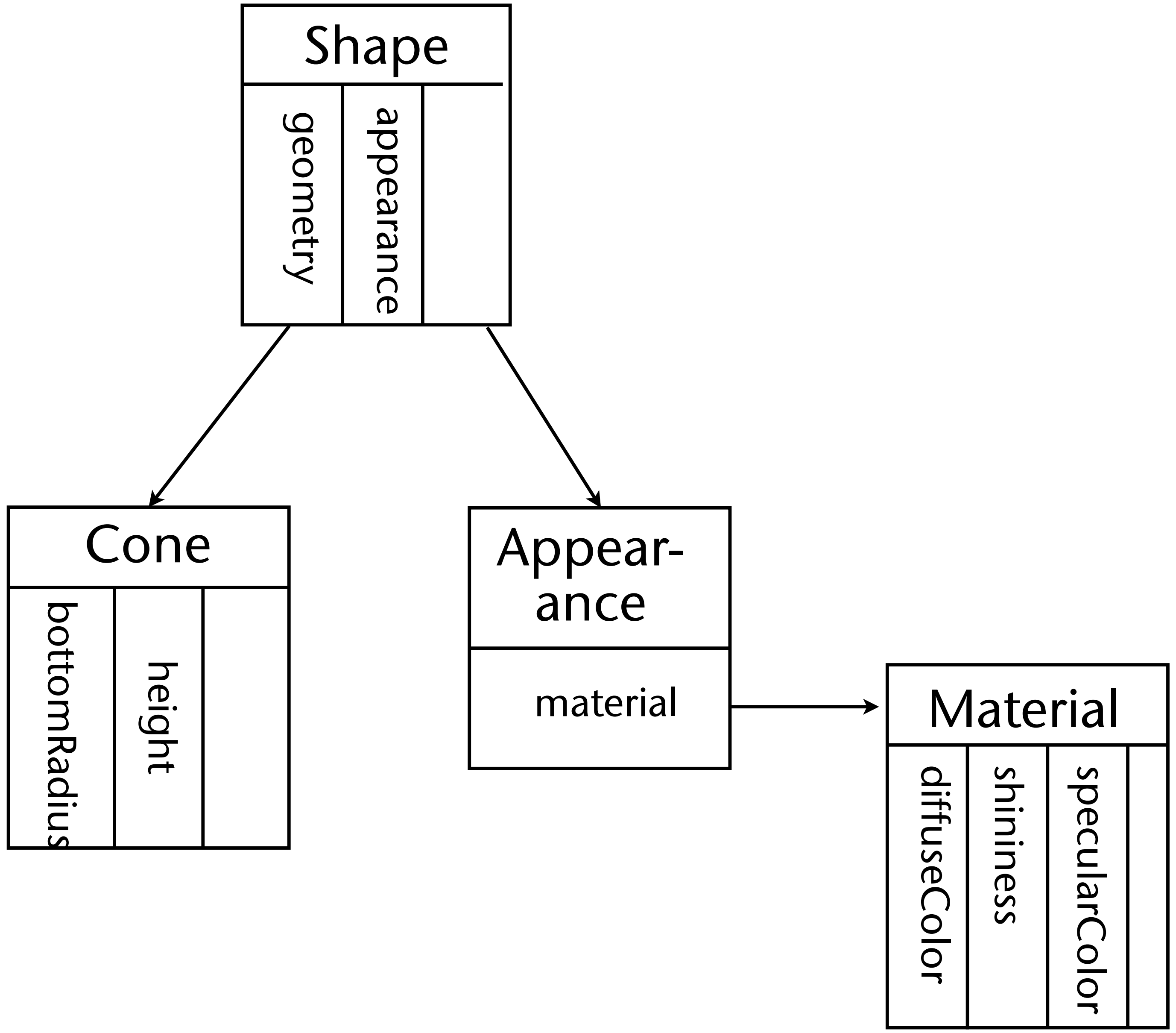
- General remarks on the design of X3D/VRML:
  - The design is orthogonal in that there exists a **`MF`**-type for every **`SF`**-type
  - The design is not orthogonal in that some types are generic (e.g. **`SFBool`**, **`SFVec3f`**) while others have very specific semantics (e.g. **`SFColor`**, **`SFTime`**, etc.)
    - It is not clear whether this is good or bad …

Types of Nodes to Describe the Scenegraph

- Most scenegraphs have a set of different kinds of nodes to define the tree:
  1. Nodes for grouping / hierarchy building
  2. Nodes for storing actual geometry
  3. Nodes for storing appearance, i.e., material def's, textures, etc.
- In X3D/VRML, for instance:
  1. **`Shape, Group, Transform, Switch, Billboard, LOD, ...`**
  2. **`TriangleSet, IndexedTriangleSet, IndexedFaceSet, IndexedTriangleStripSet, Box, Sphere, Cylinder, NurbsPatchSurface, ElevationGrid, ... ...`**
  3. **`Appearance, Material, ImageTexture,`**

A Simple Example

```
#X3D V3.1 utf8
Shape {
 geometry Cone {
   bottomRadius 1
   height       2
 }
 appearance Appearance {
   material Material {
     ambientIntensity 0.256
     diffuseColor     0.029 0.026 0.027
     shininess        0.061
     specularColor    0.964 0.642 0.980
   }
 }
}
```



Specifying the Material

- A standard model: Phong (somewhat dated)

$$I_{\text{out}} = I_{\text{amb}} + I_{\text{diff}} + I_{\text{spec}}$$

$$I_{\text{diff}} = k_d I_{\text{in}} \cos\phi \qquad I_{\text{spec}} = k_s I_{\text{in}} (\cos\theta)^p$$

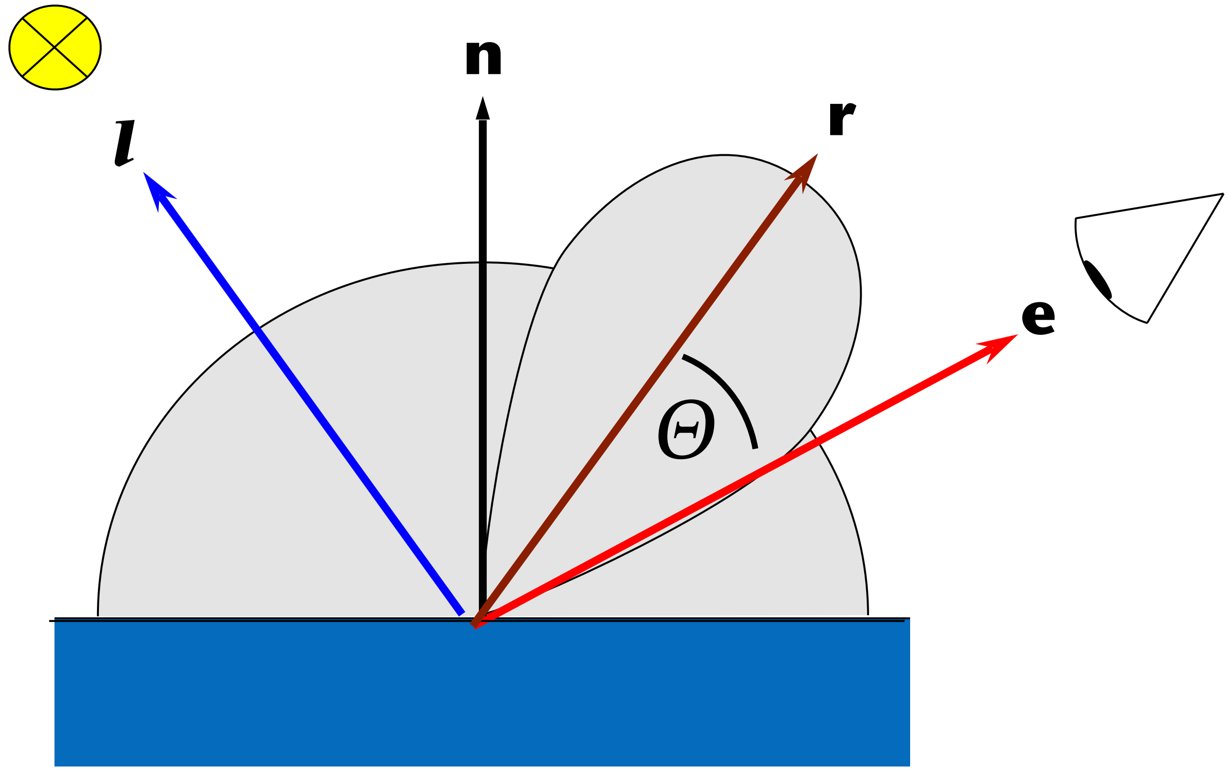
$$I_{\text{out}} = k_d \cdot I_a + \sum_{j=1}^{n} (k_d \cos\phi_j + k_s \cos^p \theta_j) \cdot I_j$$

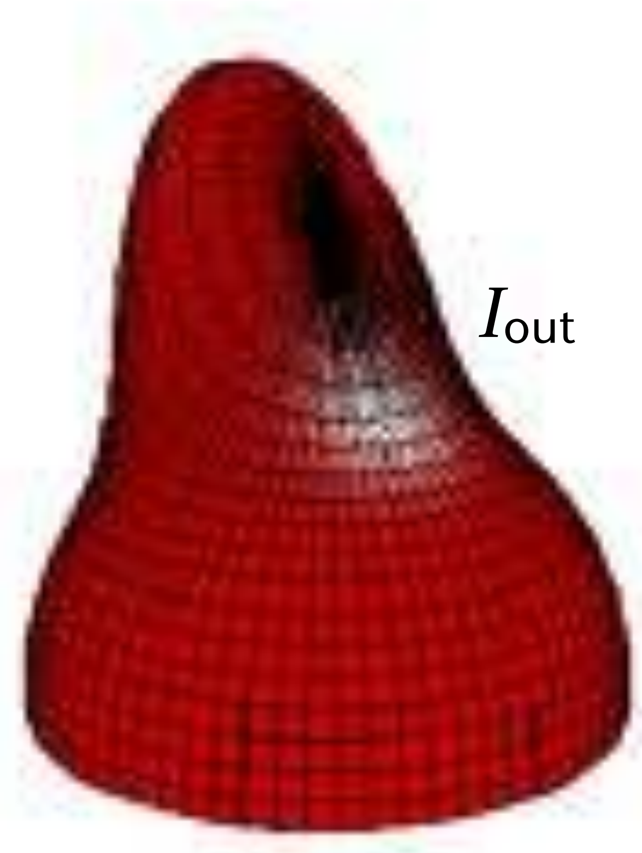
$$= k_d I_a + \sum_{j=1}^{n} \left(k_d(\mathbf{n}\boldsymbol{l}) + k_s(\mathbf{rv})^p \right) \cdot I_j$$

k_d = diffuse reflection coefficient

k_s = specular reflection coefficient

p = *shininess*



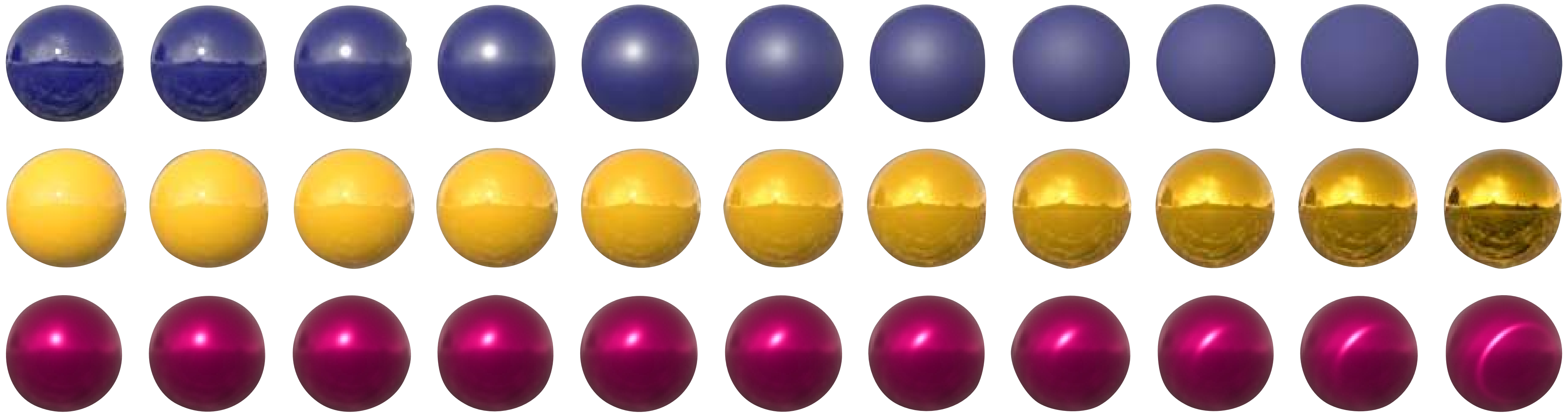


- In VRML/X3D:

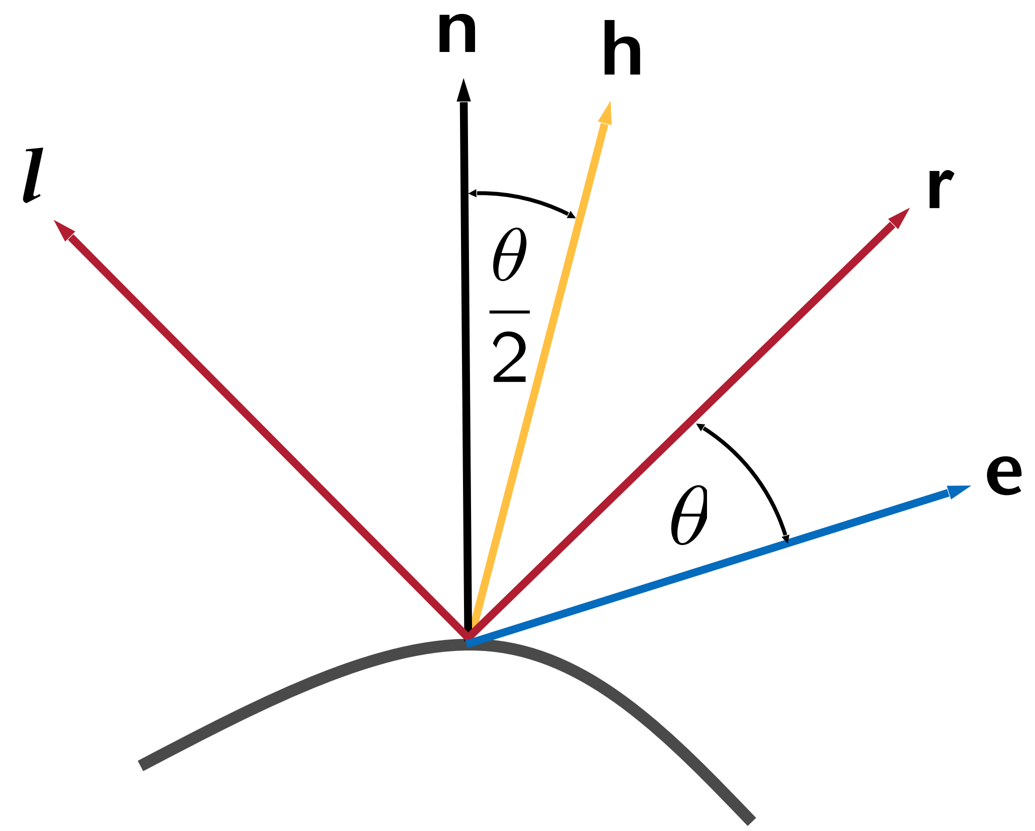
```
Material {
 SFFloat ambientIntensity 0.2
 SFColor diffuseColor     0.8 0.8 0.8
 SFColor specularColor    0  0  0
 SFFloat shininess        0.2
 SFColor emissiveColor    0  0  0
 SFFloat transparency     0
}
```

The Material Model in Unreal

- Based on "Disney's Principled Lighting Model"
- More intuitive (for artists), while still allowing for real-time rendering
- Parameters (all can come from a texture, but could also be constant per obj):
  - Base Color = single color (RGB value)
  - Roughness, in [0,1]
  - Metallic = yes/no (or [0,1])
  - Anisotropic
  - Many more ...



A Bit of Mathematical Background



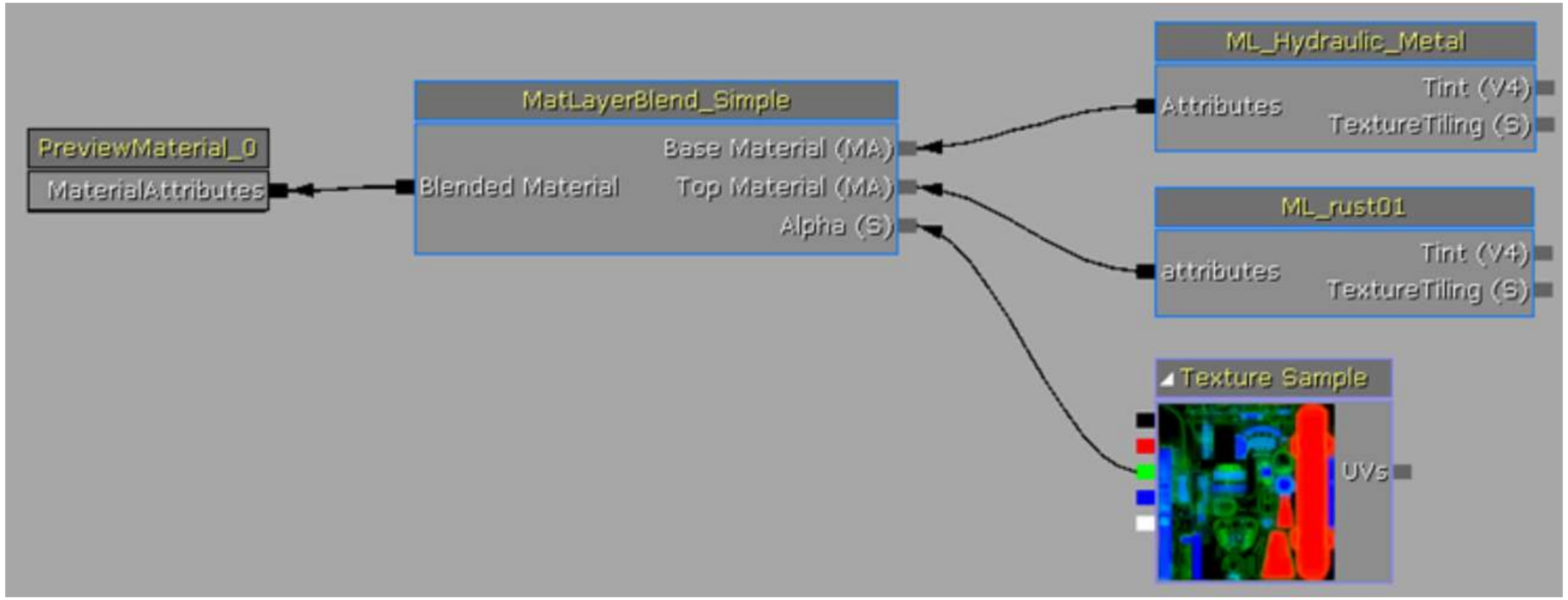
- Uses half-vector $\mathbf{h} = \dfrac{\boldsymbol{l} + \mathbf{e}}{|\boldsymbol{l} + \mathbf{e}|}$
  - Nice property: $\angle(\mathbf{n} \text{ and } \mathbf{h}) = 0 \;\Leftrightarrow\; \angle(\mathbf{e} \text{ and } \mathbf{r}) = 0$
- The BRDF:
  - Function **ρ** describes reflectance = $\dfrac{\text{Outgoing "intensity" in direction } \mathbf{e}}{\text{Incoming "intensity" from direction } \boldsymbol{l}}$
  - Based on Cook-Torrance's microfacet model

$$\rho(\boldsymbol{l}, \mathbf{e}) = \frac{D(\mathbf{h})F(\mathbf{e}, \mathbf{h})G(\boldsymbol{l}, \mathbf{e}, \mathbf{h})}{4(\mathbf{n} \cdot \boldsymbol{l})(\mathbf{n} \cdot \mathbf{e})}$$

with *D* = normal distribution fct, *G* = specular attenuation based on roughness, *F* = Fresnel term

Layered Materials

- Several materials can be applied to the same object using linear interpolation (blending)
-

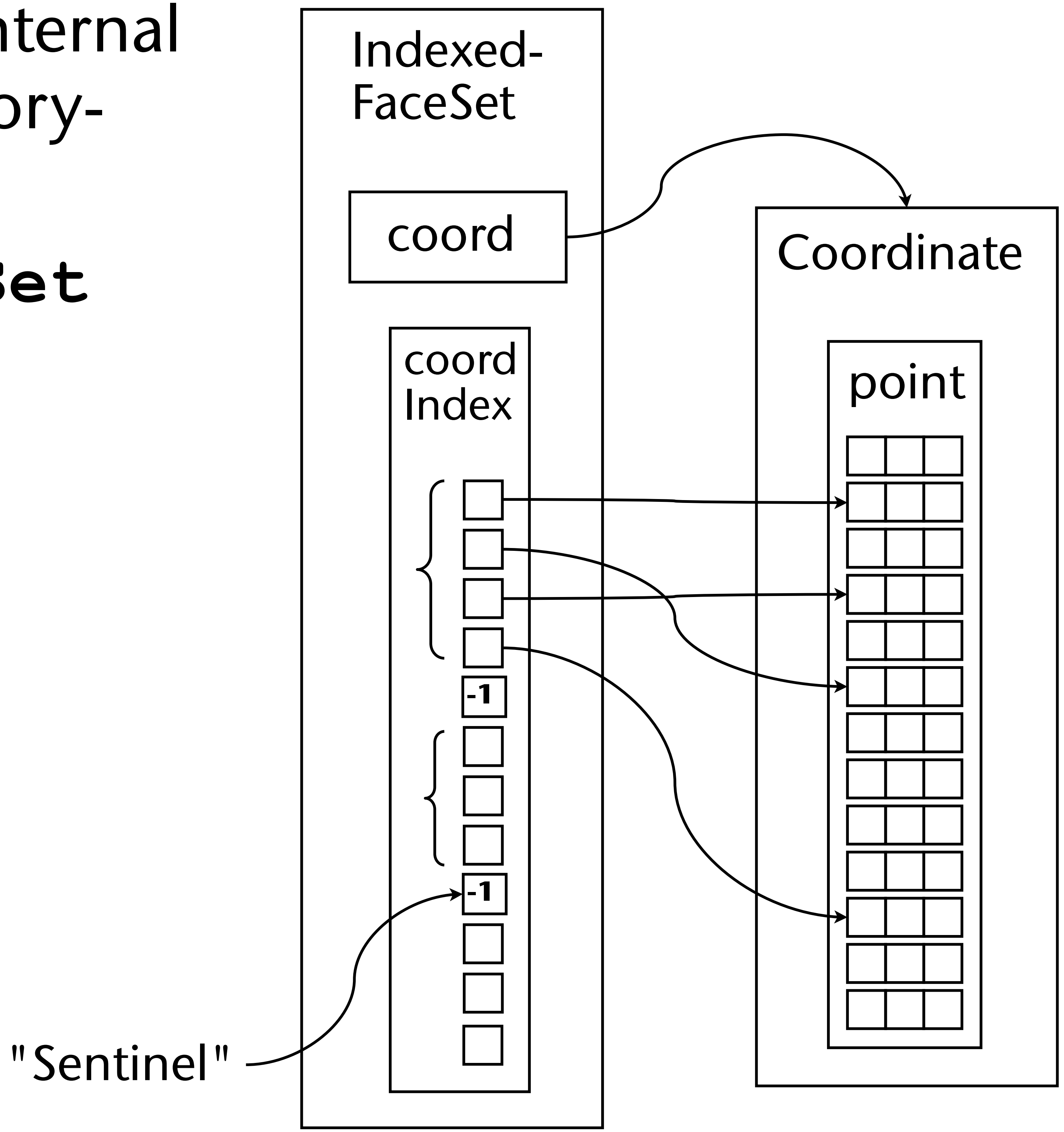


Common Data Structures to Store Geometry

- Most scene graphs / game engines have internal data structures to store geometry in memory-efficient ways
- Prominent data structure: **`IndexedFaceSet`**

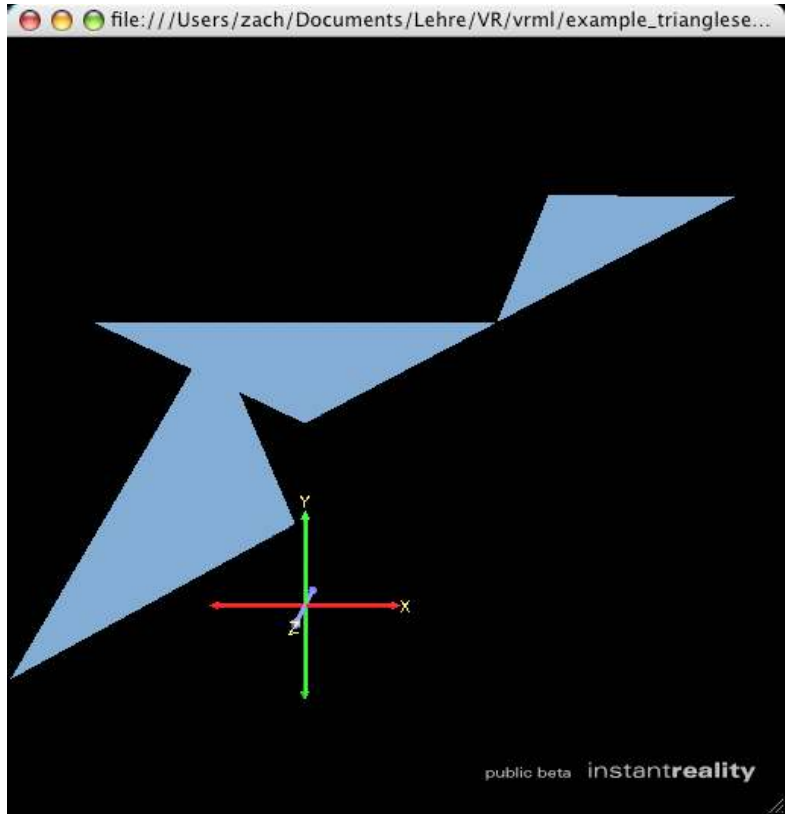
```
IndexedFaceSet {
  SFNode  coord         NULL
  MFInt32 coordIndex    []
  SFBool  ccw           TRUE
  SFBool  normalPerVertex TRUE
  SFBool  solid         TRUE
  SFFloat creaseAngle 0.0
 }
```

```
Coordinate {
 MFVec3f point []
}
```



- Example:

```
Shape {
 geometry IndexedFaceSet {
   coord Coordinate {
     point [ -2 0 3, -0 1 1, -1 3 0,
              0 2 0,  2 3 1, -2 3 1,
              3 5 -2, 4 4 2 ]
   }
   coordIndex [ 0 1 2 -1  3 4 5 -1  6 4 7 -1 ]
   solid FALSE
   ccw TRUE
 }
 appearance Appearance { … }
}
```



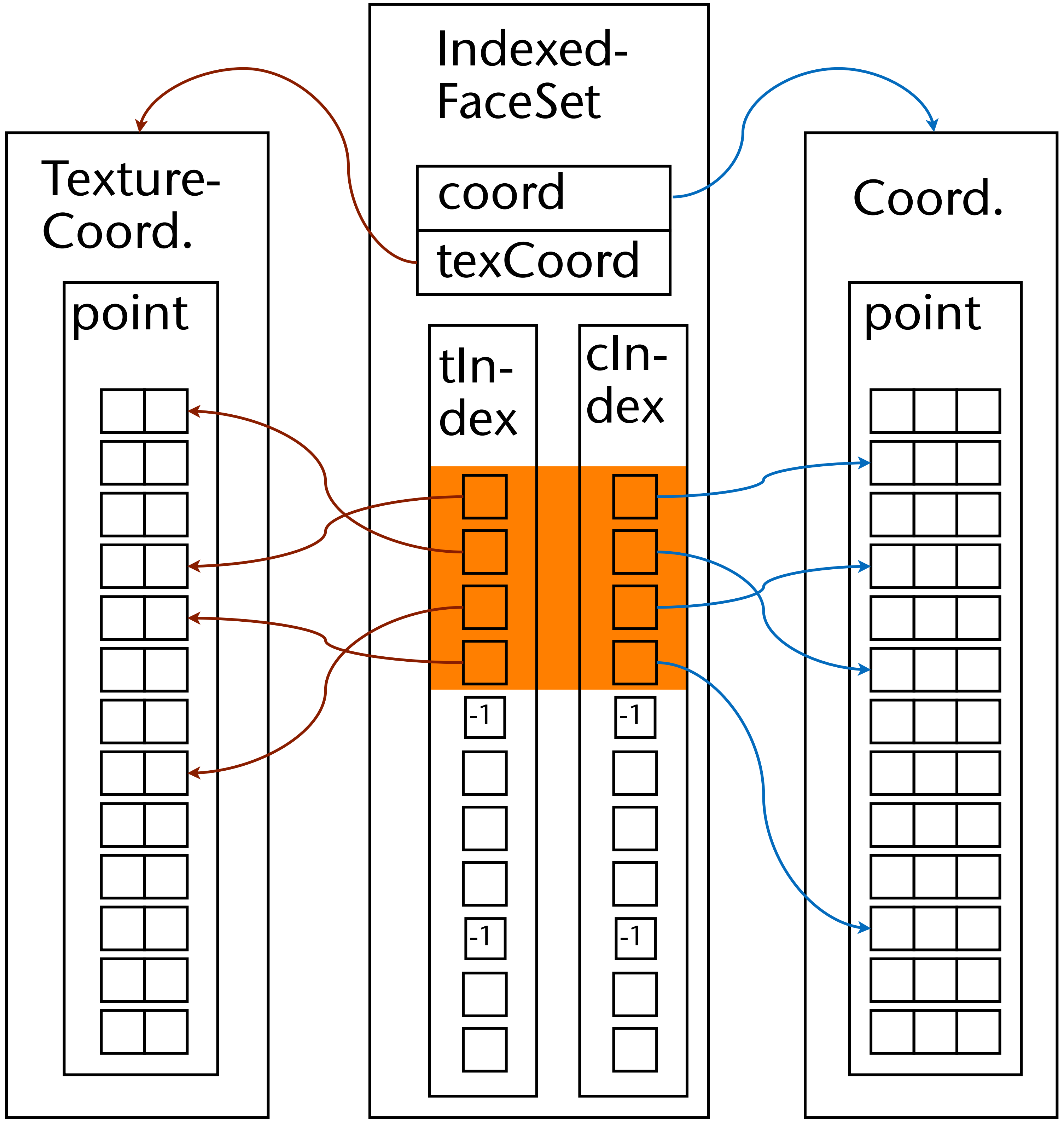
example_indexedtriangleset.wrl

- Geometry stored this way is called a mesh

Specification of Additional Attributes per Vertex

- In meshes, you can always specify additional vertex attributes, eg., normals or texture coordinates per vertex
- Texture coords are stored in an indexed face set as follows:

```
IndexedFaceSet {
  SFNode   coord
  MFInt32  coordIndex
  SFNode   texCoord
  MFInt32  texCoordIndex
  SFBool   ccw
  SFBool   normalPerVertex
  SFBool   solid
}
```



The OBJ File Format

- Only geometry and textures
  - Usually only used for polygonal geometry
  - Can store NURBS, too
- Only in ASCII (very good)
  - Very easy to read and parse as a human
  - Extremely easy to write a loader (takes just an afternoon)
    - Line-based, i.e., one line = one piece of information (e.g., vertex, polygon)
- No hierarchy

Example

Letter(s) at beginning of the line tells what information the line contains:

v = vertex,

vt = texture coords,

vn = vertex normal,

f = face

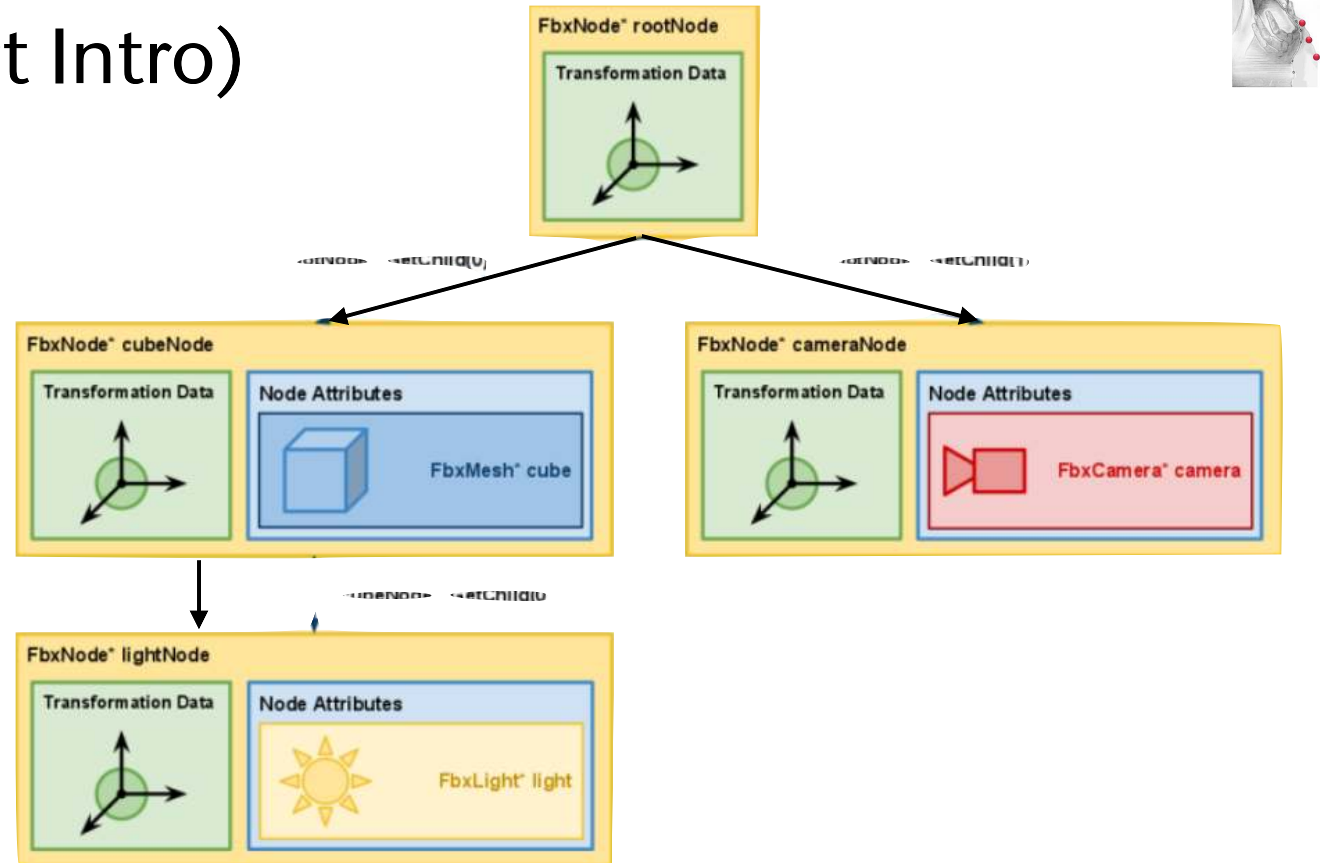
```
# A cube
mtllib cube.mtl
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
vt 0.748573 0.750412
vt 0.749279 0.501284
vt 0.999110 0.501077
vt 0.999455 0.750380
vt 0.250471 0.500702
vt 0.249682 0.749677
vt 0.001085 0.750380
vt 0.001517 0.499994
vt 0.499422 0.500239
vt 0.500149 0.750166
vt 0.748355 0.998230
vt 0.500193 0.998728
vt 0.498993 0.250415
vt 0.748953 0.250920
```

```
vn 0.000000 0.000000 -1.000000
vn -1.000000 -0.000000 -0.000000
vn -0.000000 -0.000000 1.000000
vn -0.000001 0.000000 1.000000
vn 1.000000 -0.000000 0.000000
vn 1.000000 0.000000 0.000001
vn 0.000000 1.000000 -0.000000
vn -0.000000 -1.000000 0.000000
usemtl Material_ray.png
f 5/1/1 1/2/1 4/3/1
f 5/1/1 4/3/1 8/4/1
f 3/5/2 7/6/2 8/7/2
f 3/5/2 8/7/2 4/8/2
f 2/9/3 6/10/3 3/5/3
f 6/10/4 7/6/4 3/5/4
f 1/2/5 5/1/5 2/9/5
f 5/1/6 6/10/6 2/9/6
f 5/1/7 8/11/7 6/10/7
f 8/11/7 7/12/7 6/10/7
f 1/2/8 2/9/8 3/13/8
f 1/2/8 3/13/8 4/14/8
```

Indices defining one vertex of a face (ID's for v/vt/vn)

The FBX File Format (Only a Very Short Intro)

- Geometry and textures
- Can store hierarchies
- Animations
- Instancing
- ASCII (pretty well readable by humans), and binary
- Proprietary (Autodesk), but a de-facto standard
  - Still changes over time



```
node_name: possible_properties {
  Node_Property_1: value
  Node_Property_2: value
  Subnode1  :  {
    Subnode_Property_1: value
    [...]
  }
  Node_Property_3: value
  [...]
}
```

beginning of node

beginning of sub-node

end of node

end of node

Storing Geometry

Node Objects contains the geometry

Beginning of one of the objects

Vertex coords, *n* floats follow, 3 values = 1 vertex

Vertex indices, *m* integers follow,
negative index = last one of the face (*)

Sub-node of Model

Property containing *k* normals, 3 values = 1 normal
Mapping to vertices is determined by
`MappingInformationType`:
`ByPolygonVertex` = one normal per polygon vertex
`ByVertex` = one normal per vertex in Vertices

Dito for uv-coordinates

```
Objects:  {
  Model: "model name", "Mesh" {
    [...]
    Vertices: *n {
      a: [...]
    }
    PolygonVertexIndex : *m {
      a: [...]
    }
    LayerElementNormal: 0 {
      Normals: *k {
        a: [...]
      }
    }
    LayerElementUV: 0 {
      UV: *n {
        a: [...]
      }
    }
  }
}
```

*) How to convert negative indices *i* : $i' = -i - 1$
(in C: **`posIndex = ~negIndex;`**)

Example

```
; a cube
Objects: {
  Geometry: "Geometry::", "Mesh" {
    Vertices: *24 {
      a: -1,-1,1,1,-1,1,-1,1,1,1,1,1,-1,1,-1,1,1,-1,-1,-1,-1,1,-1,-1
    }
    PolygonVertexIndex: *24 {
      a: 0,1,3,-3,2,3,5,-5,4,5,7,-7,6,7,1,-1,1,7,5,-4,6,0,2,-5
    }
    Edges: *12 {
      a: 0,2,6,10,3,1,7,5,11,9,15,13
    }
    LayerElementNormal: 0 {
      Normals: *72 {
        a: 0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,0,1,0,0,1,0,0,1,0,0,0,-1,0,0,-1,0,0,-1,
           0,0,-1,0,-1,0,0,-1,0,0,-1,0,0,-1,0,1,0,0,1,0,0,1,0,0,1,0,0,-1,0,0,
           -1,0,0,-1,0,0,-1,0,0
      }
    }
    LayerElementUV: 0 {
      UV: *28 {
        a: 0.375,0,0.625,0,0.375,0.25,0.625,0.25,0.375,0.5,0.625,0.5,0.375,0.75,
           0.625,0.75,0.375,1,0.625,1,0.875,0,0.875,0.25,0.125,0,0.125,0.25
      }
      UVIndex: *24 {
        a: 0,1,3,2,2,3,5,4,4,5,7,6,6,7,9,8,1,10,11,3,12,0,2,13
      }
    }
  }
}
```

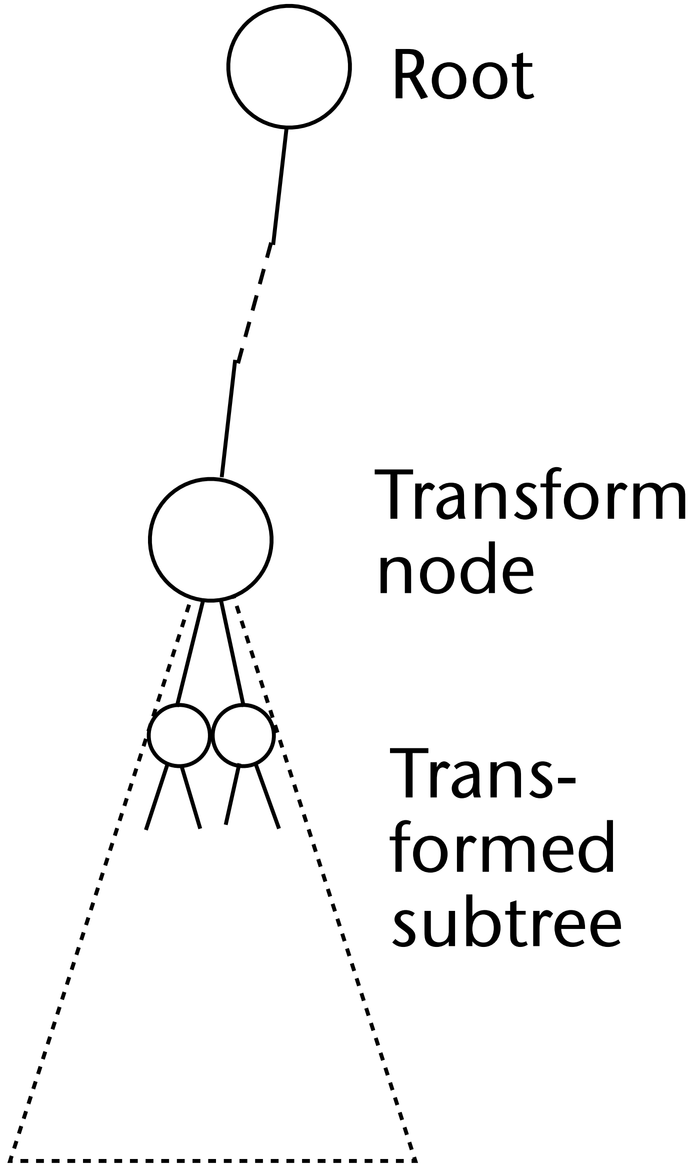
← Indices into **PolygonVertexIndex** array; edge = that vertex and next one

← 14 pairs of (u,v) coordinates

← Indices into the **UV** array; one index per vertex in **PolygonVertexIndex**

Specification of Transformations

- Transformations are stored in a specific type of nodes, or by properties / attachments to nodes
  - All children in subtree will get transformed by it
    - Warning: FBX (3ds Max) allows you to specify inherited and non-inherited transformations!
- There are three ways how to store transformations in a scenegraph in principle:
  1. A transform node stores just *one kind* of elementary transformation, e.g., rotation
  2. A transform node stores *one transform of each kind* (only the common ones), in a *pre-defined order*
  3. A transform node stores a single *4x4 matrix*
     - It is up to the application programmer to convert elementary transformations (e.g., rotation + translation) to 4x4 matrix



Example for the Second Way: Transform Nodes in VRML

- The transformation node in VRML:

```
Transform {
 MFNode     children         []
 SFVec3f    center           0 0 0
 SFRotation scaleOrientation 0 0 1 0
 SFVec3f    scale            1 1 1
 SFRotation rotation         0 0 1 0
 SFVec3f    translation      0 0 0
}
```

translation	C
rotation	R_1
scaling	S
rotation	R_2
translation	T

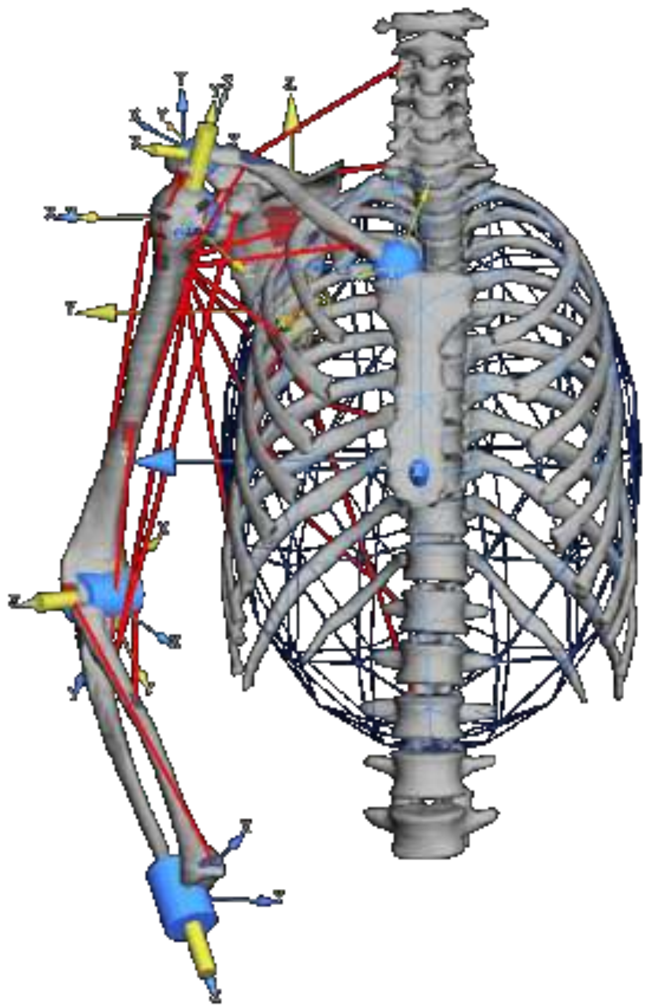
- Meaning:

$$M = T \cdot C \cdot R_2 \cdot R_1 \cdot S \cdot R_1^{-1} \cdot C^{-1}$$

with the usage/assumptions

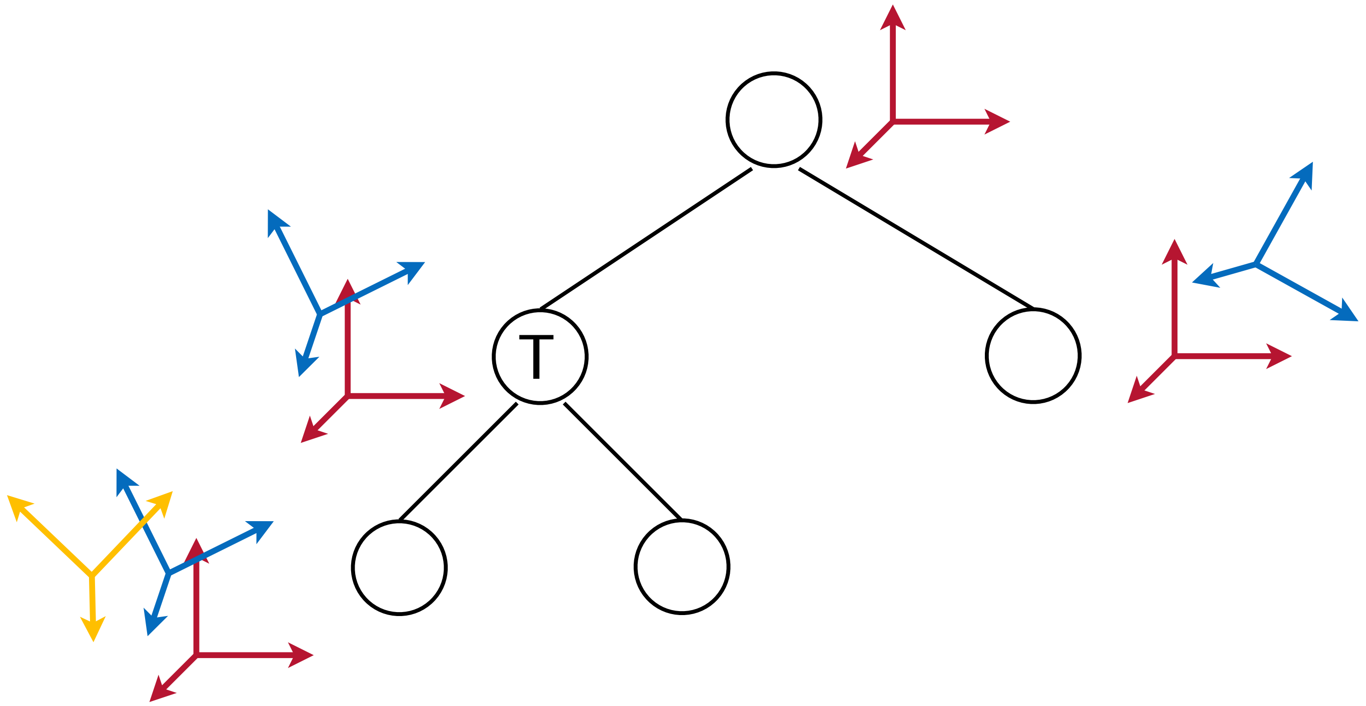
$$\mathbf{p}_{\text{world}} = M \cdot \mathbf{p}_{\text{model}}$$

Hierarchical Transformations

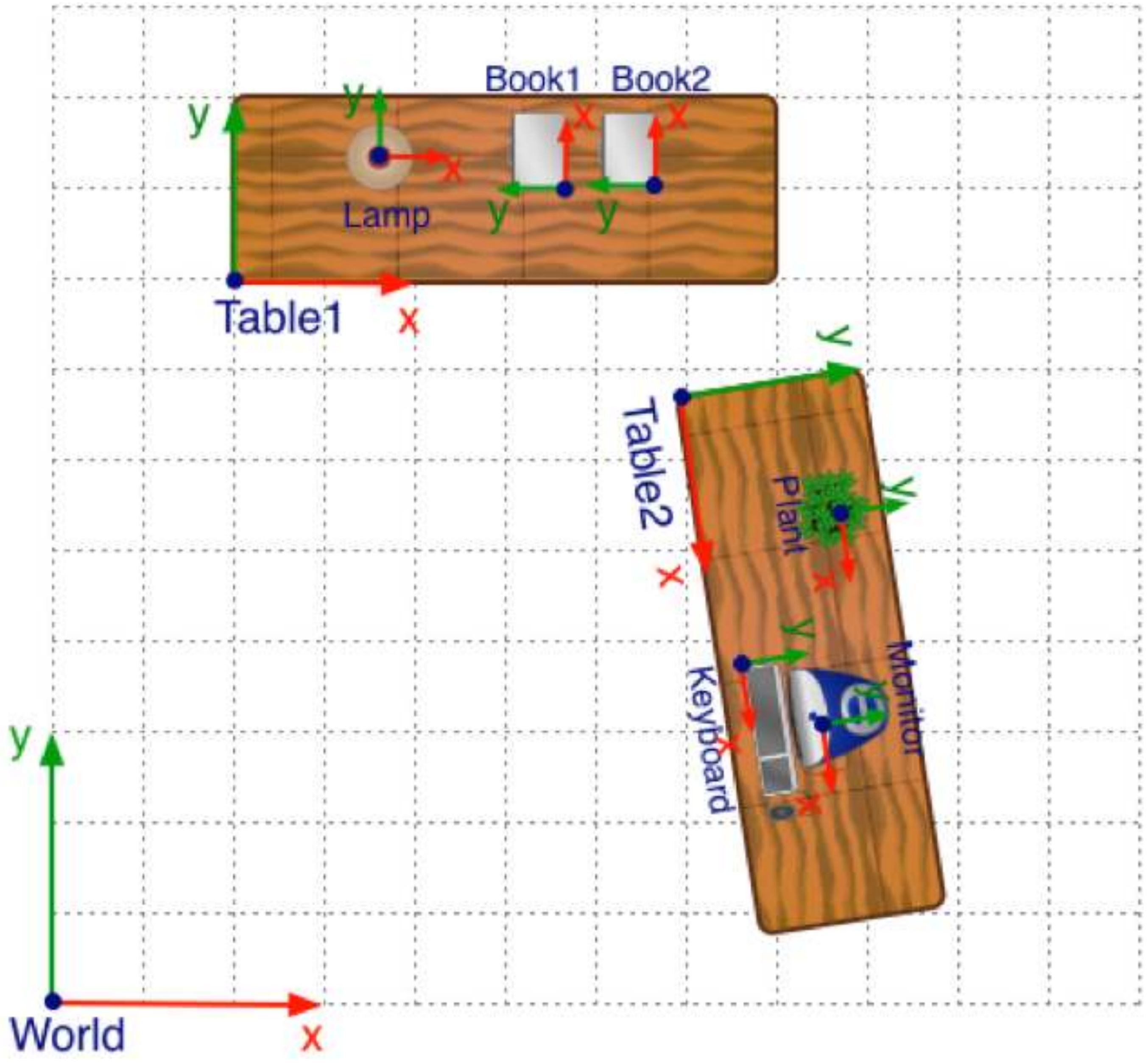


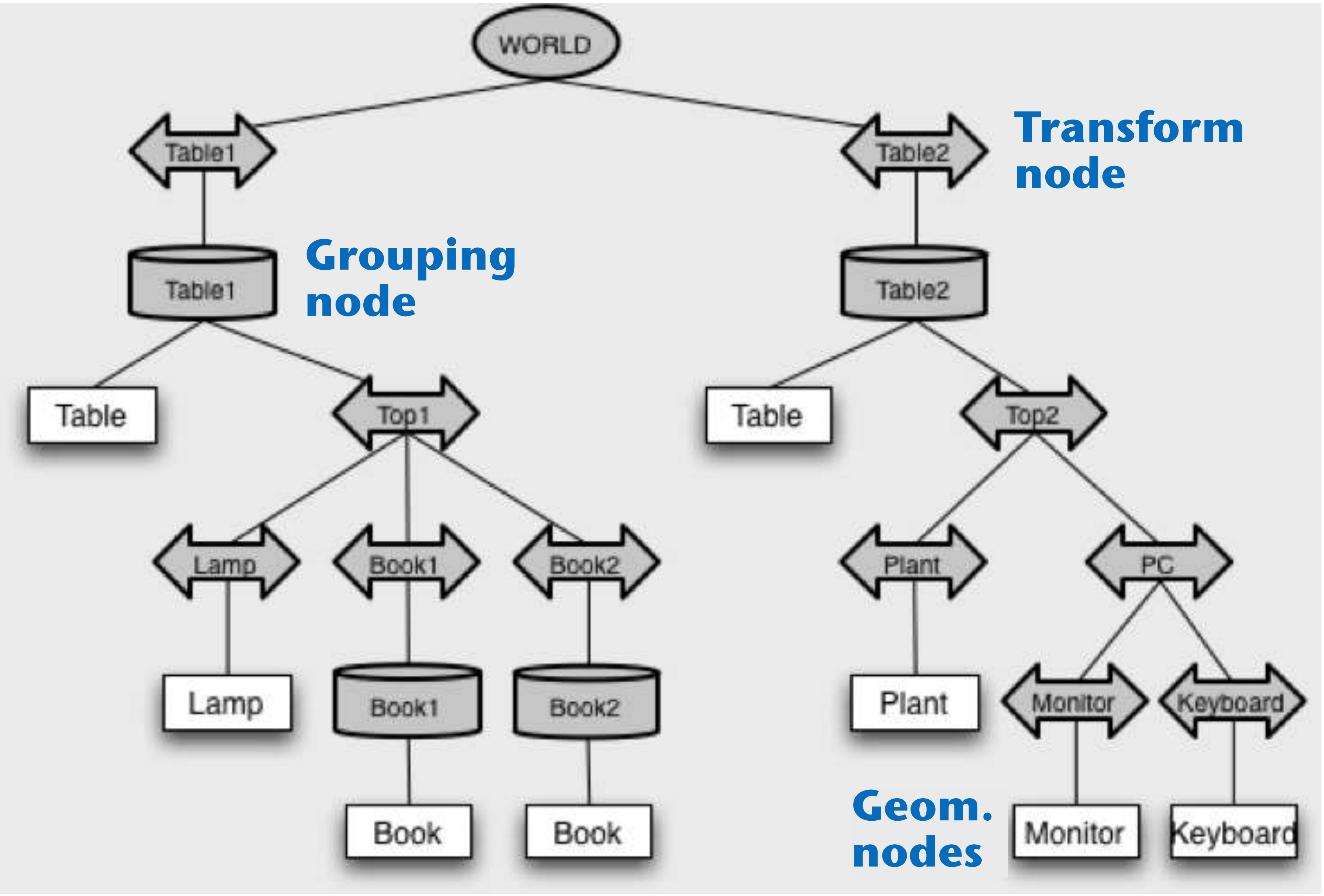
- One of the core concepts provided by scenegraphs
- Transformation node ⟶ new local coordinate system (reference frame)
  - Consequence: transformations are always specified **relative** to the parent coord frame
- Job of the renderer during scenegraph traversal: maintain a stack of transformation matrices

```
traverse( node N ):
  if N has transform T:
    let M = top of matrix stack
    M' = M * T
    push( M' )
    traverse sub-trees
    pop()          // restores M
```



Another Example





- Transform in node Table1 makes table + objs on top of it move
- Change of transformation in Top1 makes all the objs on the table top move, but not the table

- Very convenient for articulated objects
  - E.g., robots, skeletons, ..
- Remark: 2D drawing programs (Photoshop et al.) create a similar hierarchy when you group objects

The Behavior Graph

- Animations, simulations, and interactions eventually cause changes in the scene graph; e.g.:
  - Changes of transformations, i.e. the position of objects, e.g. of a robot arm
  - Modification of the materials, e.g. color or texture of an object
  - Deformation of an object, i.e. changes in the vertex coords
- All these changes are equivalent to the change of a field of a node at runtime

Events and Routes

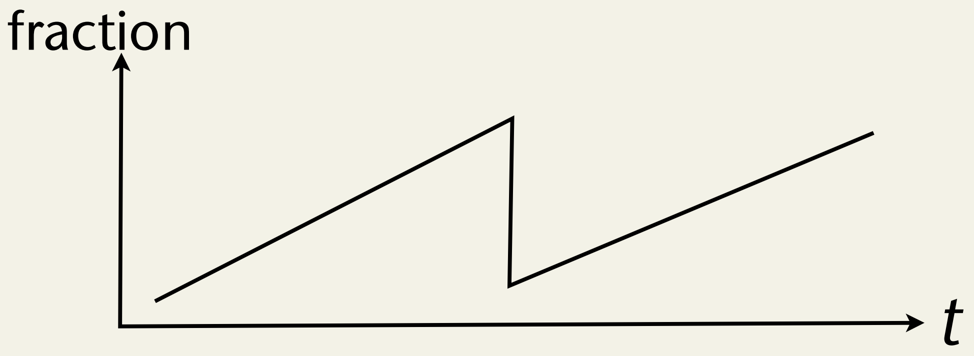
- The mechanism for changing the X3D/VRML scene graph:
  - Fields are connected to each other by routes
  - A *change* of a field produces an event
  - When an event occurs, the *content* of the field from the route-source is *copied* to the field of the route-destination ("the event is propagated")
- Other terminology: *data flow paradigm / data flow graph*
  - Used in most game engines today (and in scientific visualization tools for a long time)
- Syntax of routes in VRML:

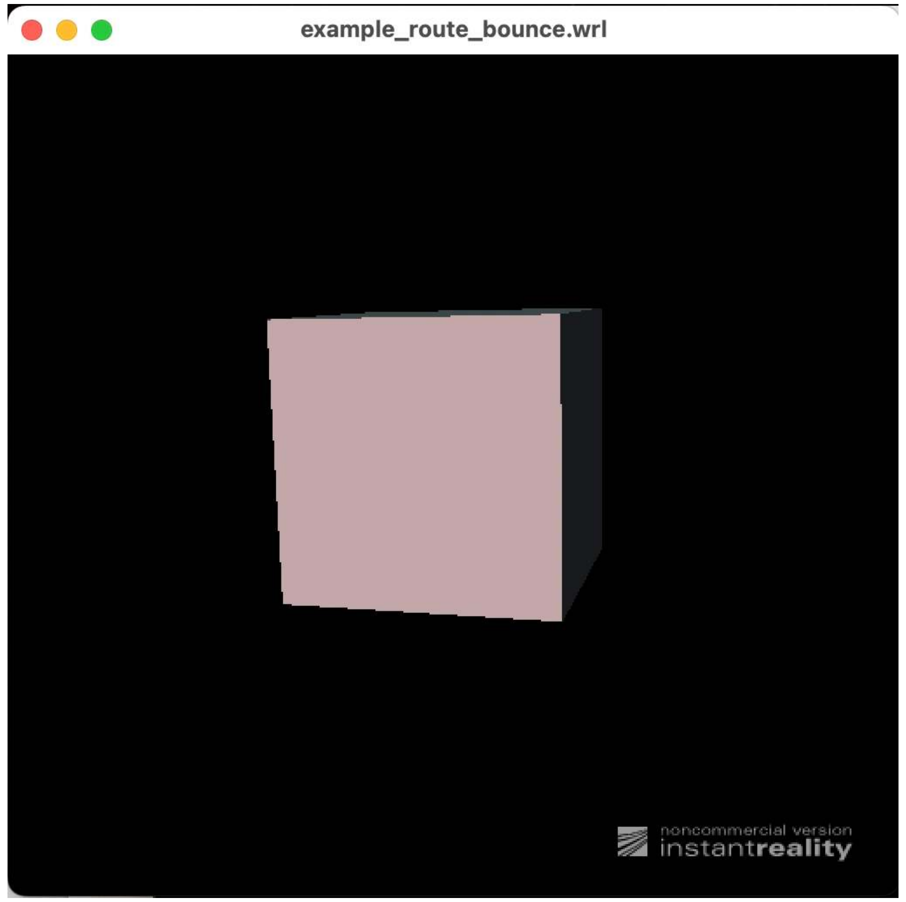
```
ROUTE Node1Name.outputFieldName TO Node2Name.inputFieldName
```

A Simple Example

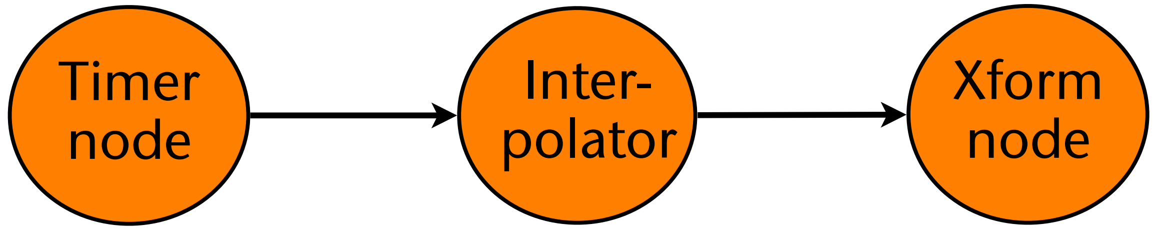
```
DEF trf Transform {
 translation 0 0 0
 children [
   Shape { geometry Box { } }
 ]
}
DEF timer TimeSensor {
 loop TRUE
 cycleInterval 5
 fraction 0.0       // out
}

DEF pi PositionInterpolator {
 fraction 0.0       // in
 key      [ 0       0.5    1      ]
 keyValue [ 0 -1 0, 0 1 0, 0 -1 0 ]
 value 0.0          // out
}
ROUTE timer.fraction_changed TO pi.set_fraction
ROUTE pi.value_changed TO trf.set_translation
```

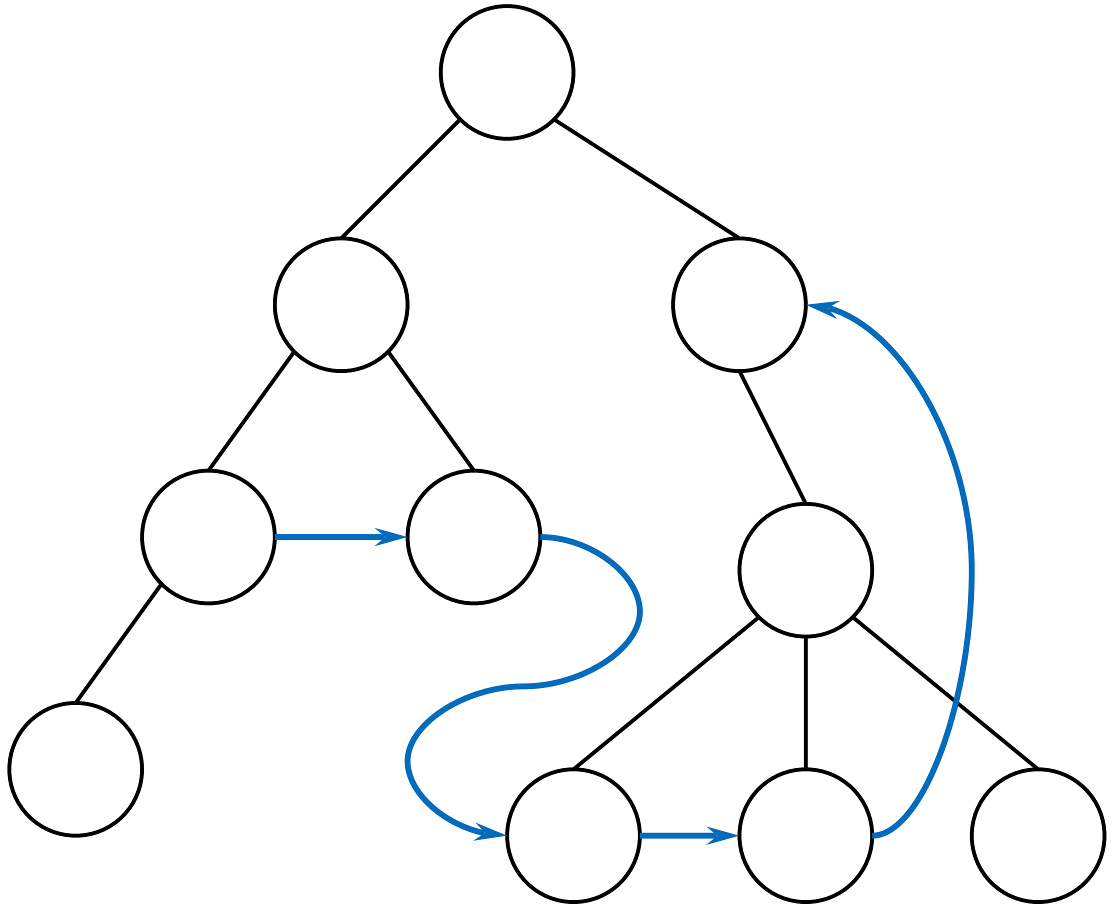




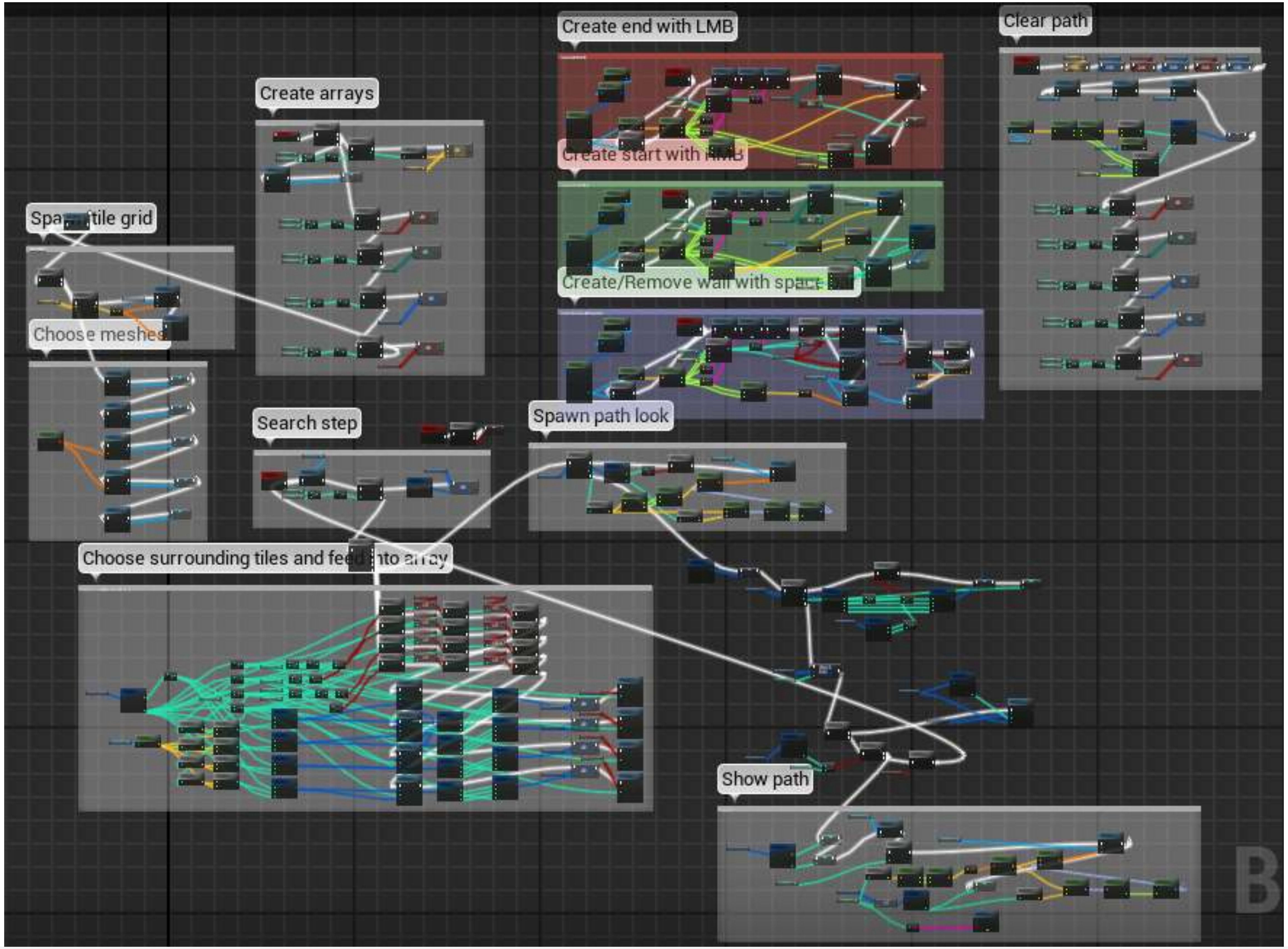
example_route_bounce.wrl



- Routes connect nodes ⟶ behavior graph:
  - Is given by the set of all routes
  - A.k.a. route graph, or event graph (blueprint in Unreal engine)
  - Is a second graph, superimposed on the scenengraph

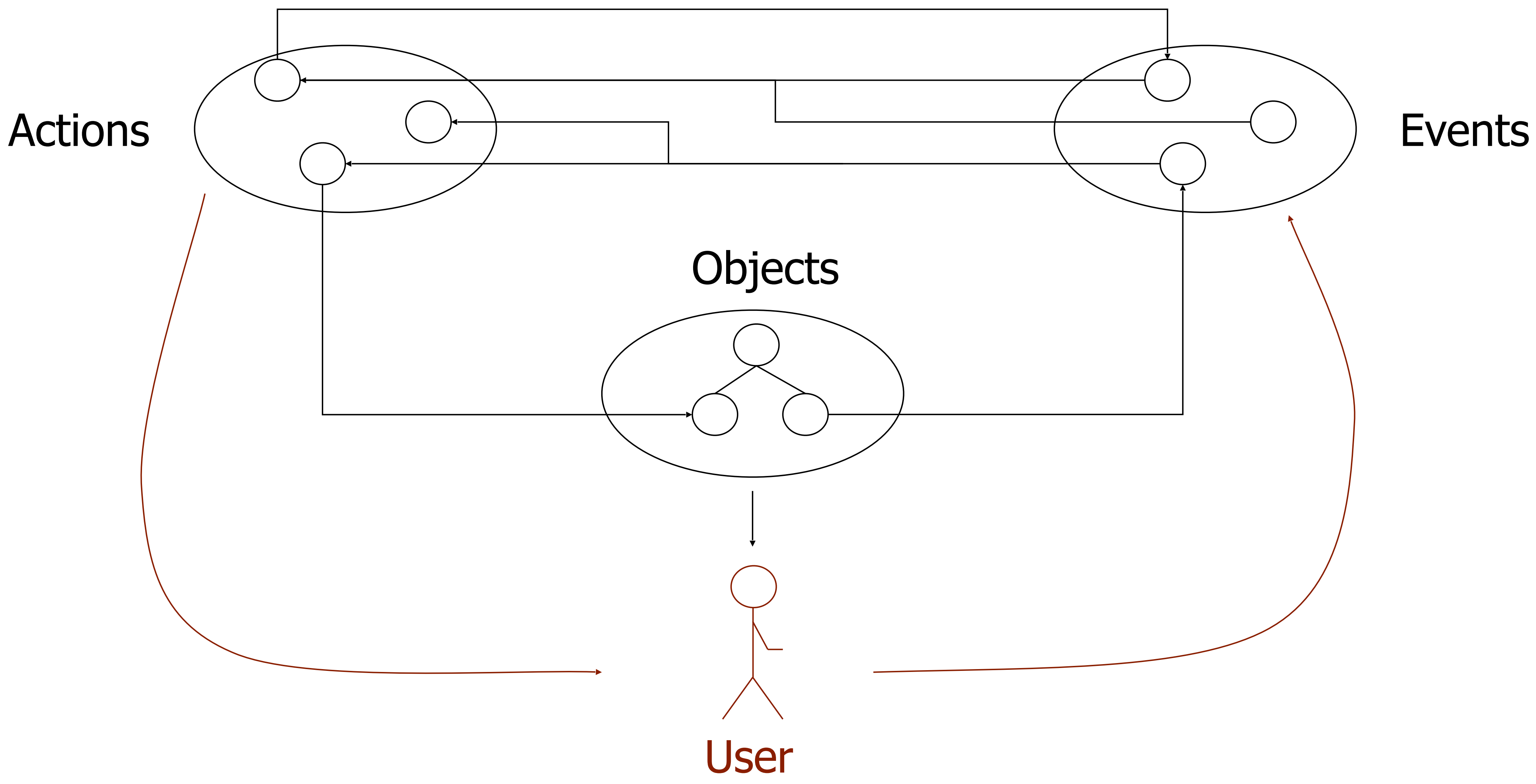


Example from Unreal



A* path finding behavior graph

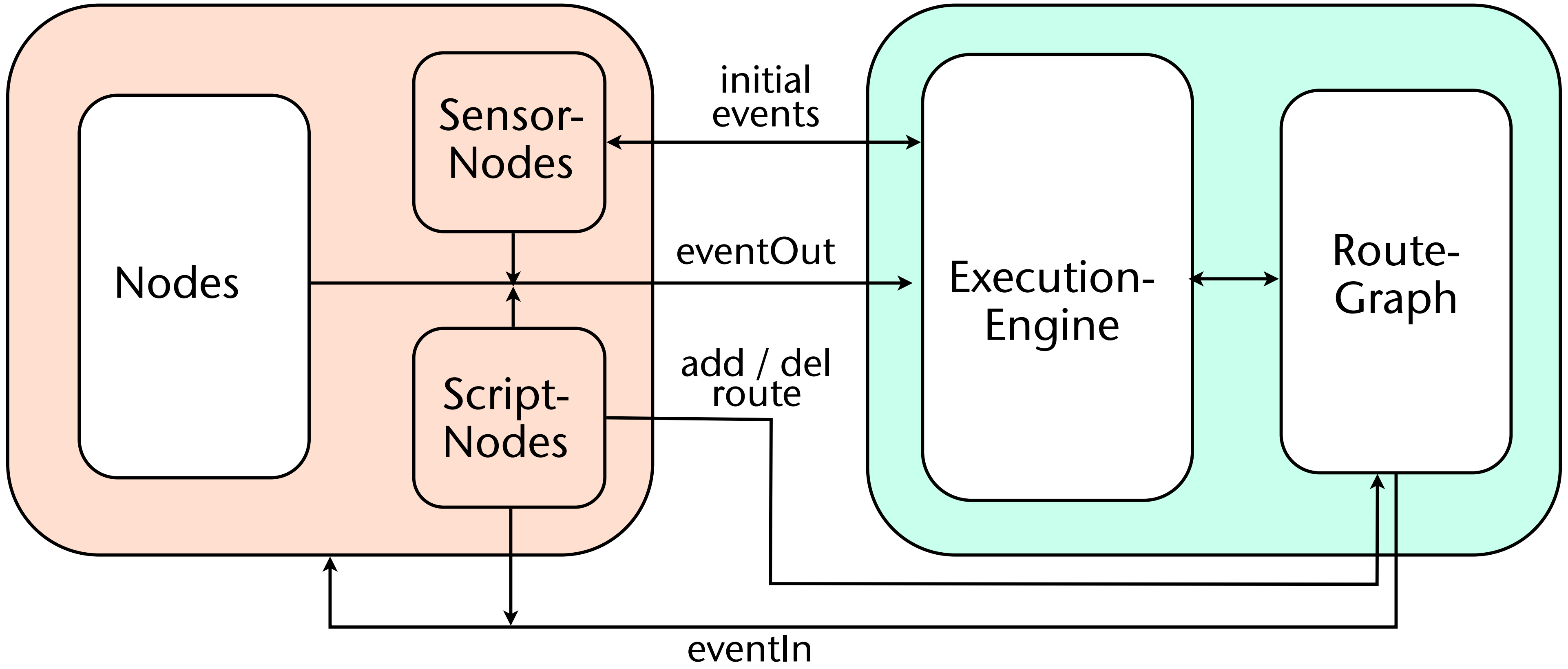
The AEO Concept



- In X3D/VRML:
    - Actions & objects are all nodes in the same scene graph
    - Events are volatile and have no "tangible" representation

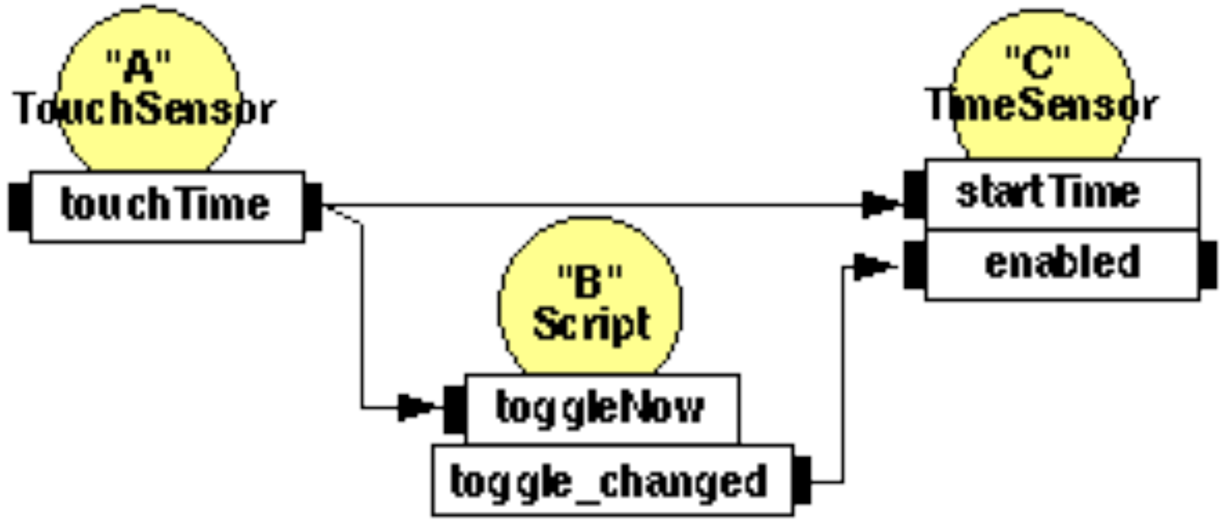
The Execution Model

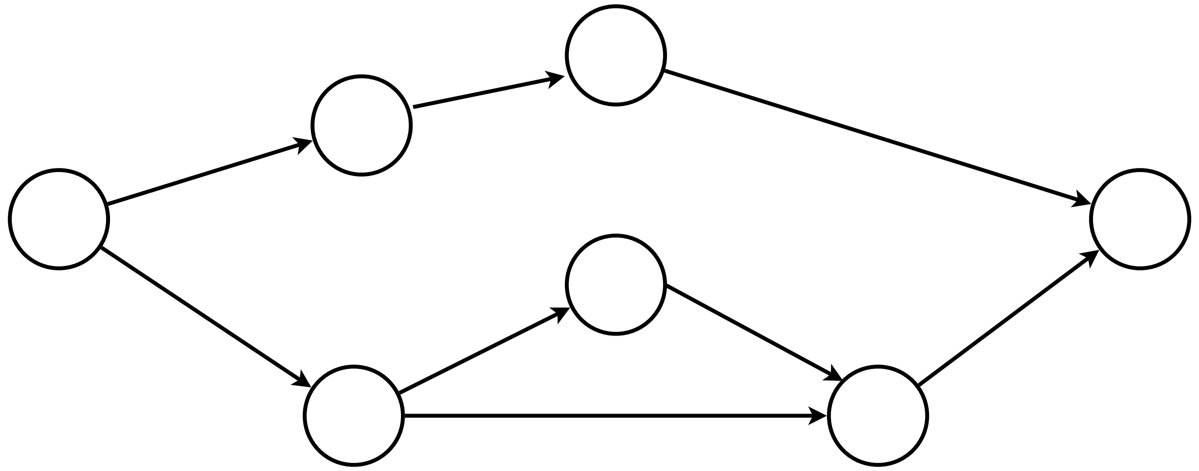
- The Event Cascade:
  - Event := change of a field



  - Initial event (of scripts, sensor, or timer)
  - Propagate to all connected `eventIn`'s
  - Nodes (e.g. interpolator) can generate other events over `eventOut`'s
  - All of these events are part of the same cascade
  - Propagating until the cascade is empty
  - Several cascades can occur per frame (caused by various initial events)

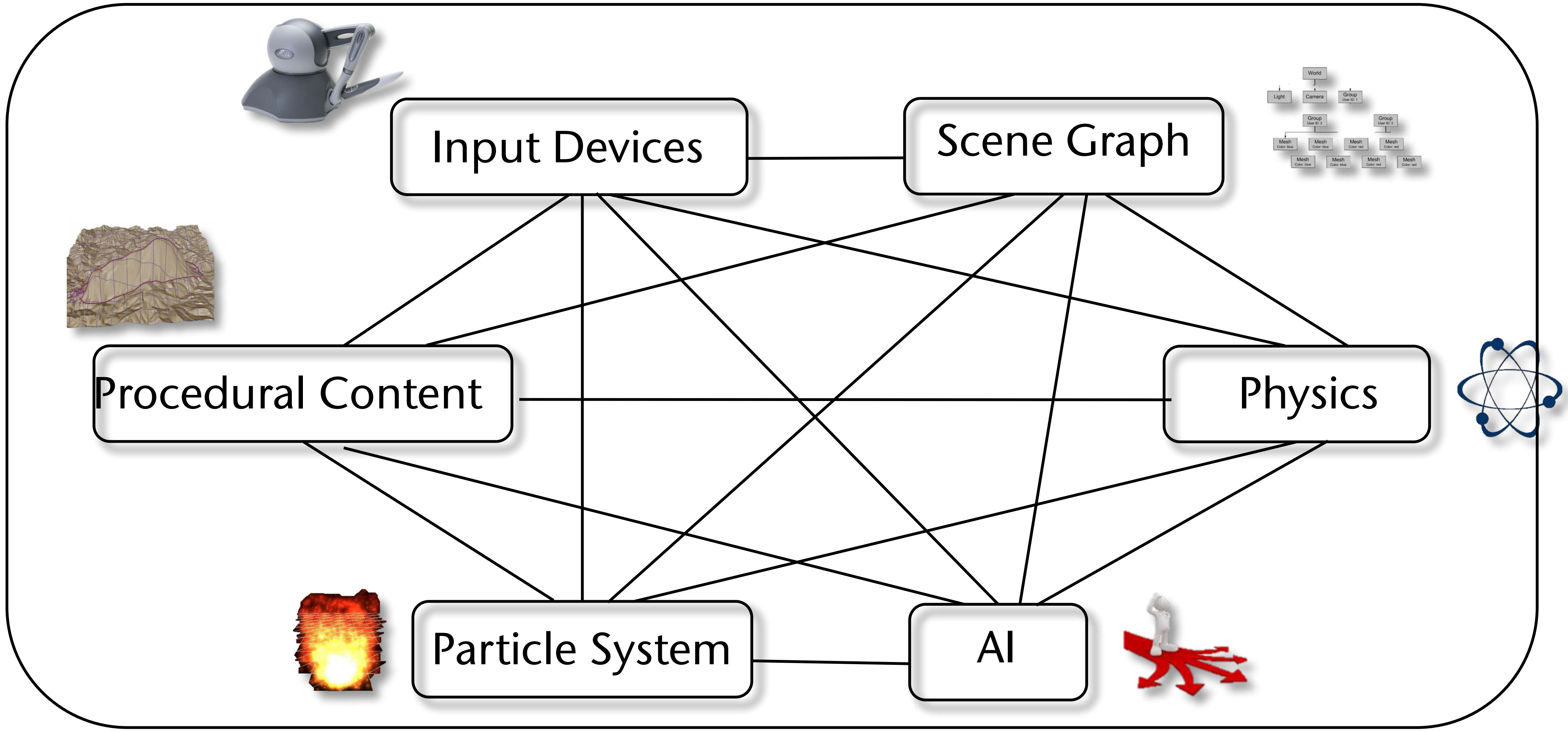
- Routes induce dependence between nodes:
  - Propagating in the "right" order
  - Algorithm:
    - Breadth-first traversal through graph
    - Sort according to current dependencies among the nodes in the "moving front"
- Cycles:
  - Are allowed (in VRML!, sometimes even useful)
  - *Loop breaking rule*: Each field may "fire" only 1x per event-cascade; i.e., every route is "served" only 1x per event-cascade





New Concepts for Data Flow in VR/Game Engines

- Modern systems usually consist of many different components
- Classic approach: fields-and-routes-based data flow
  - Good for "visual programming" (up to some complexity)
- Problem:
  many-to-many
  connectivity



Input Devices

Scene Graph

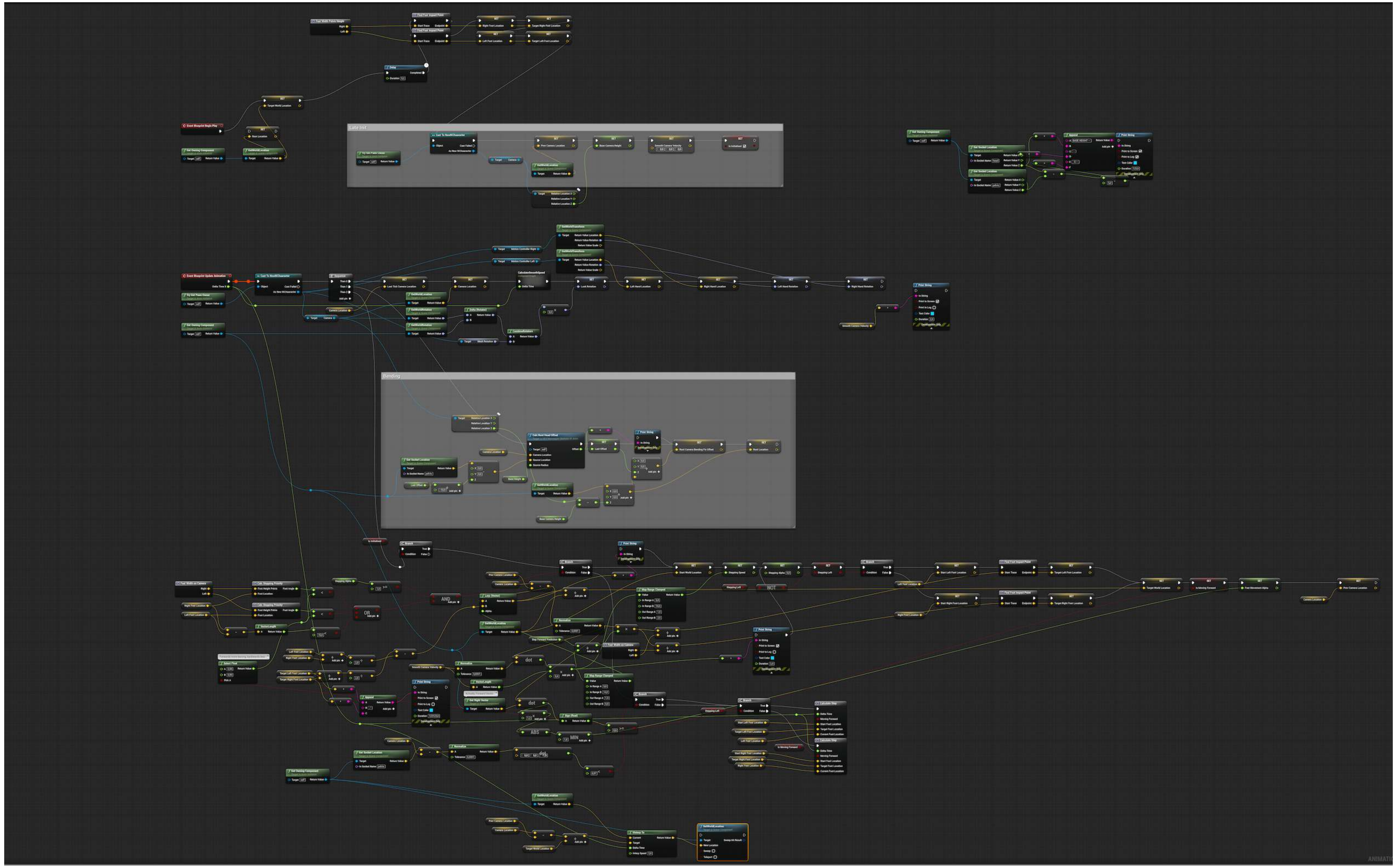
Procedural Content

Physics

Particle System

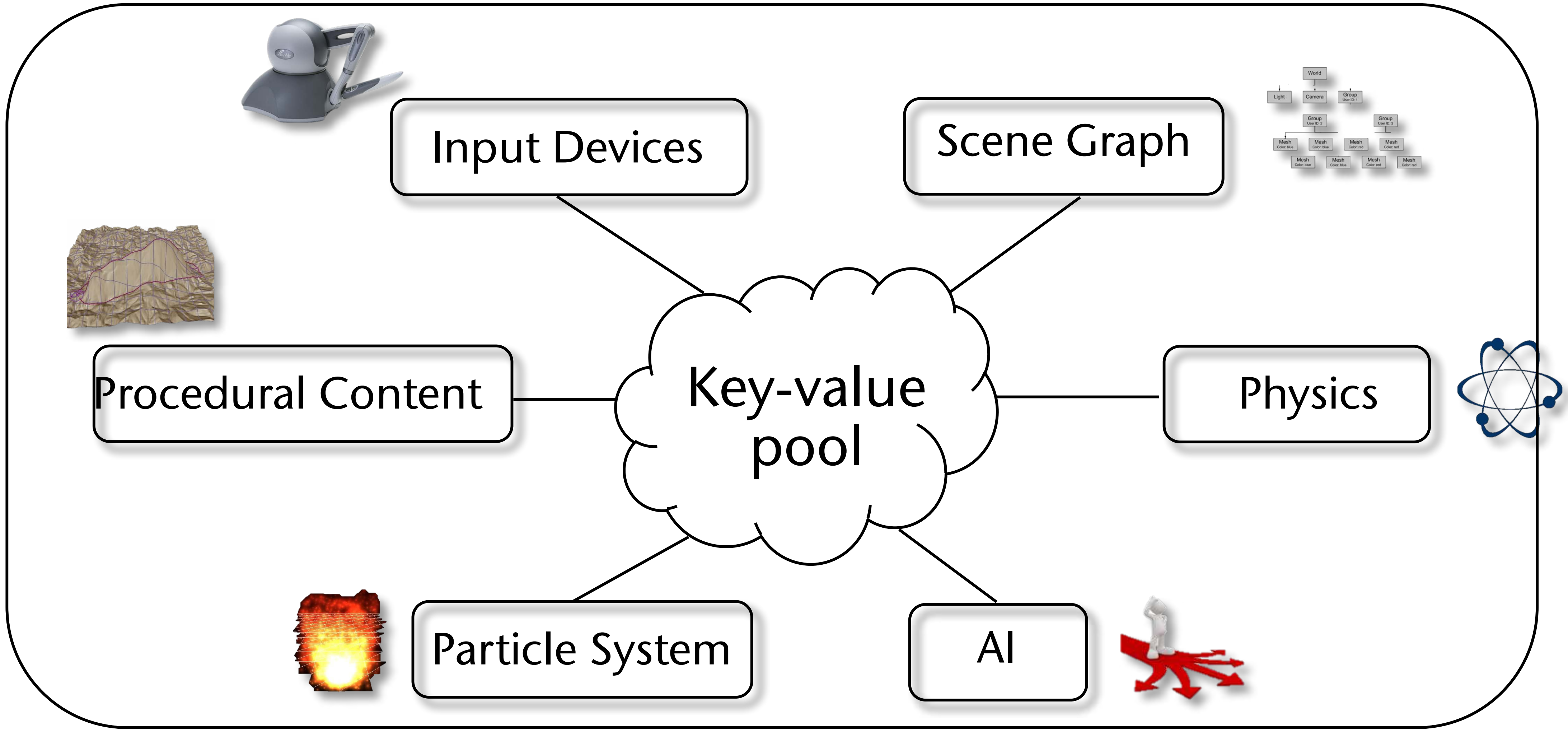
AI

This Becomes Unviable Pretty Quickly



Our Proposed Approach: the Key-Value Pool

- Assign a unique key to each outgoing field
- Producer stores value with key in KV pool ⟶ KV pair
  - Corresponds to generating an event in the data flow paradigm
- Consumer reads value from KV pair every time in its loop
- Set of all KV pairs
  ⟶ KV pool



Input Devices

Scene Graph

Procedural Content

Key-value pool

Physics

Particle System

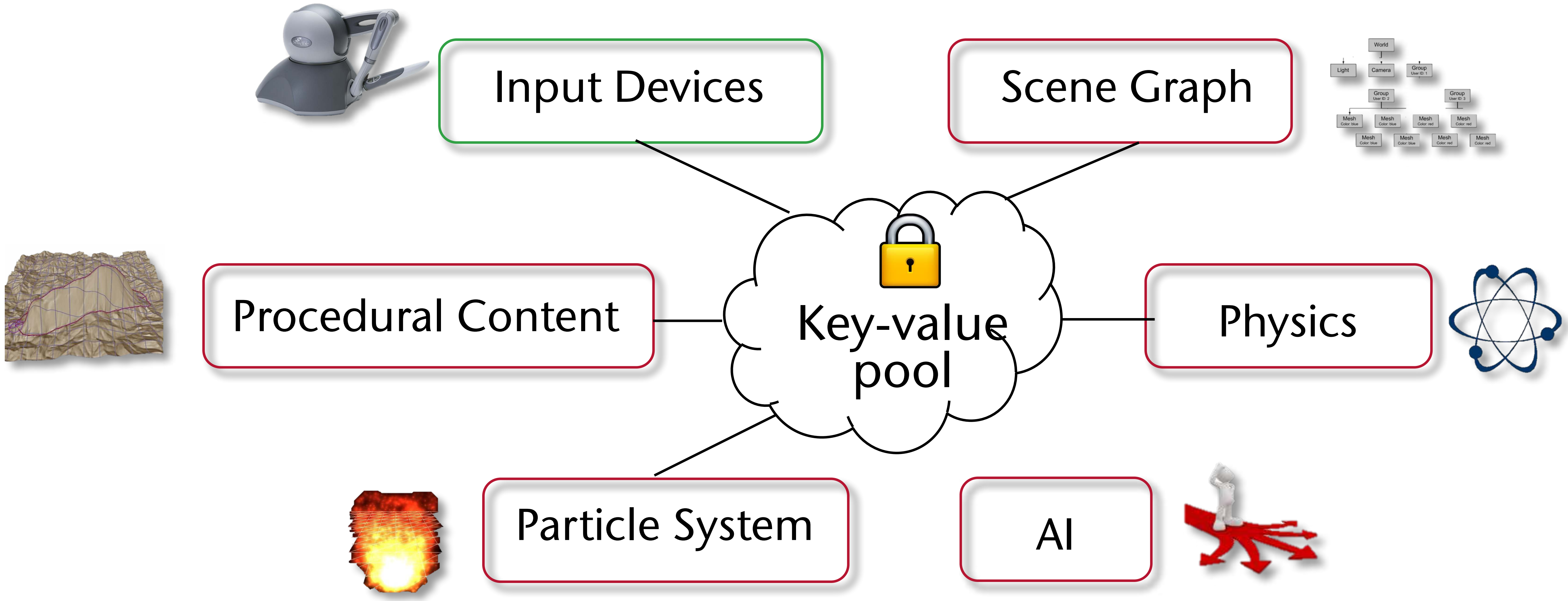
AI

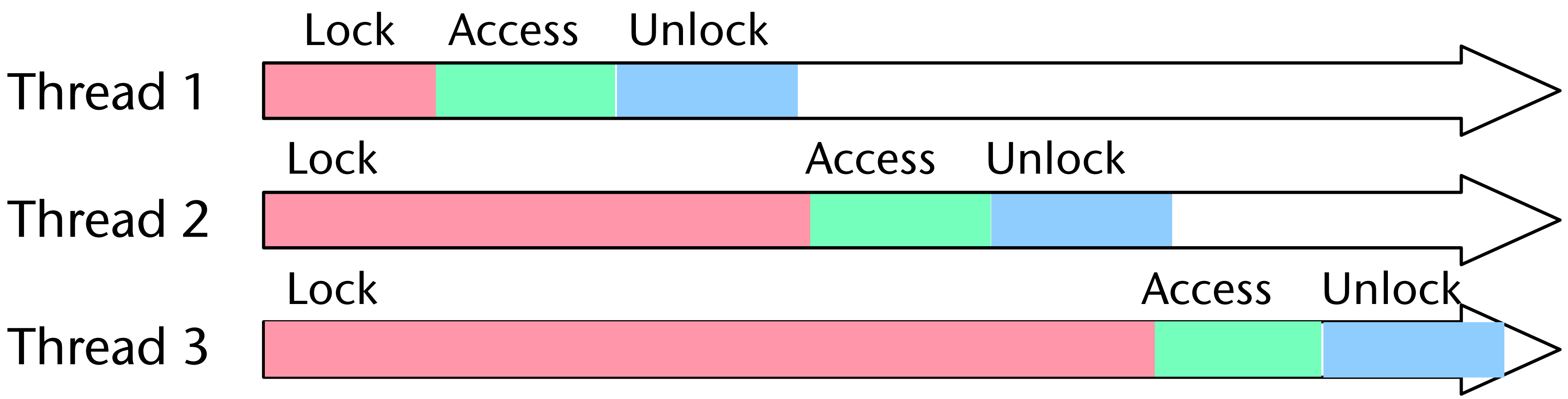
Advantages of the Approach

- The KV pool holds complete state of the virtual environment
- Can save/load state, or unwind to earlier state
- One-to-many connections are trivial

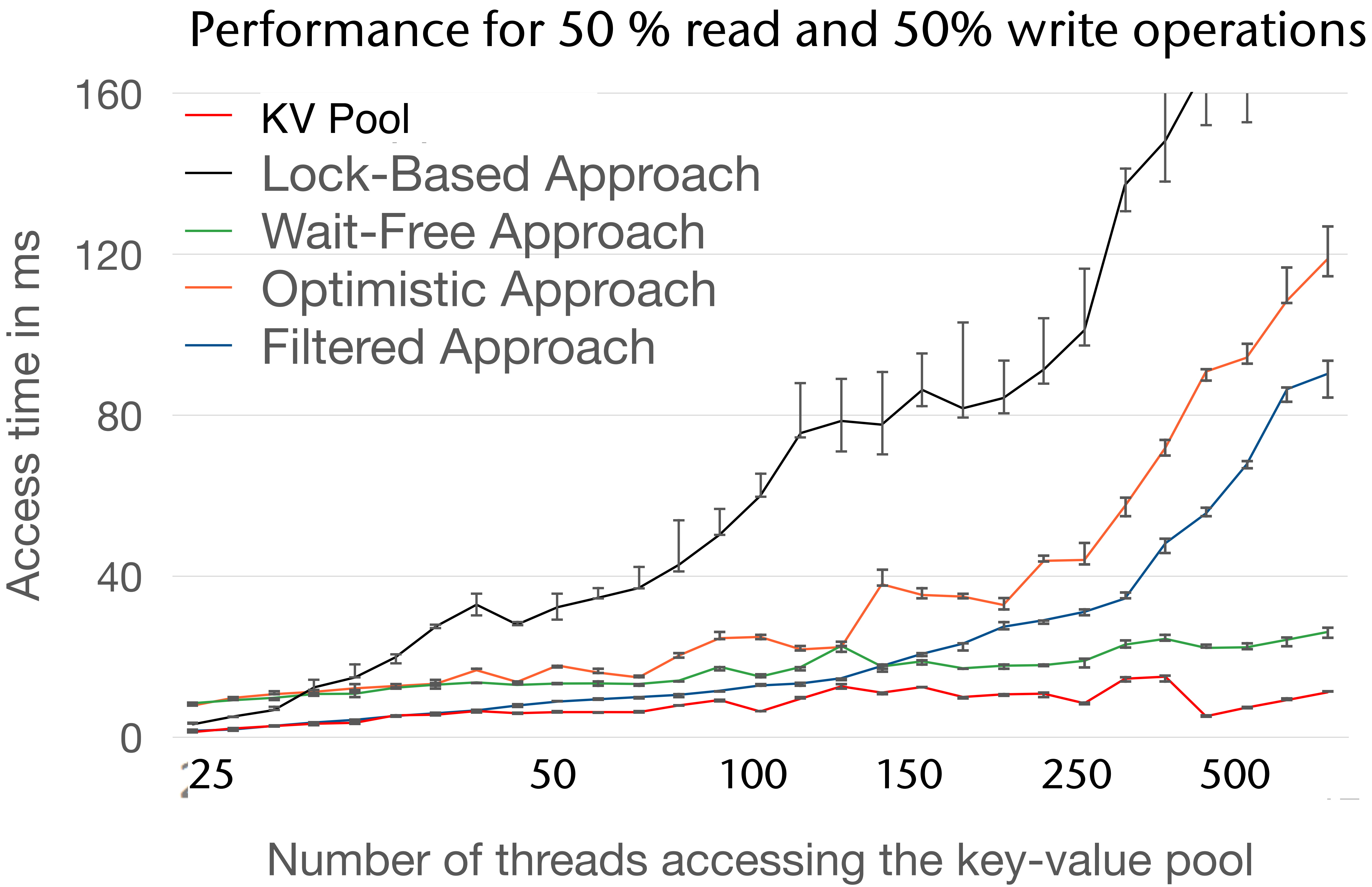
Problem: Thread-Safety

- Naive apporach: one lock per KV pair, or one lock for the whole KV pool
- In any case: lots of waiting





Our Wait-Free Hash-Map

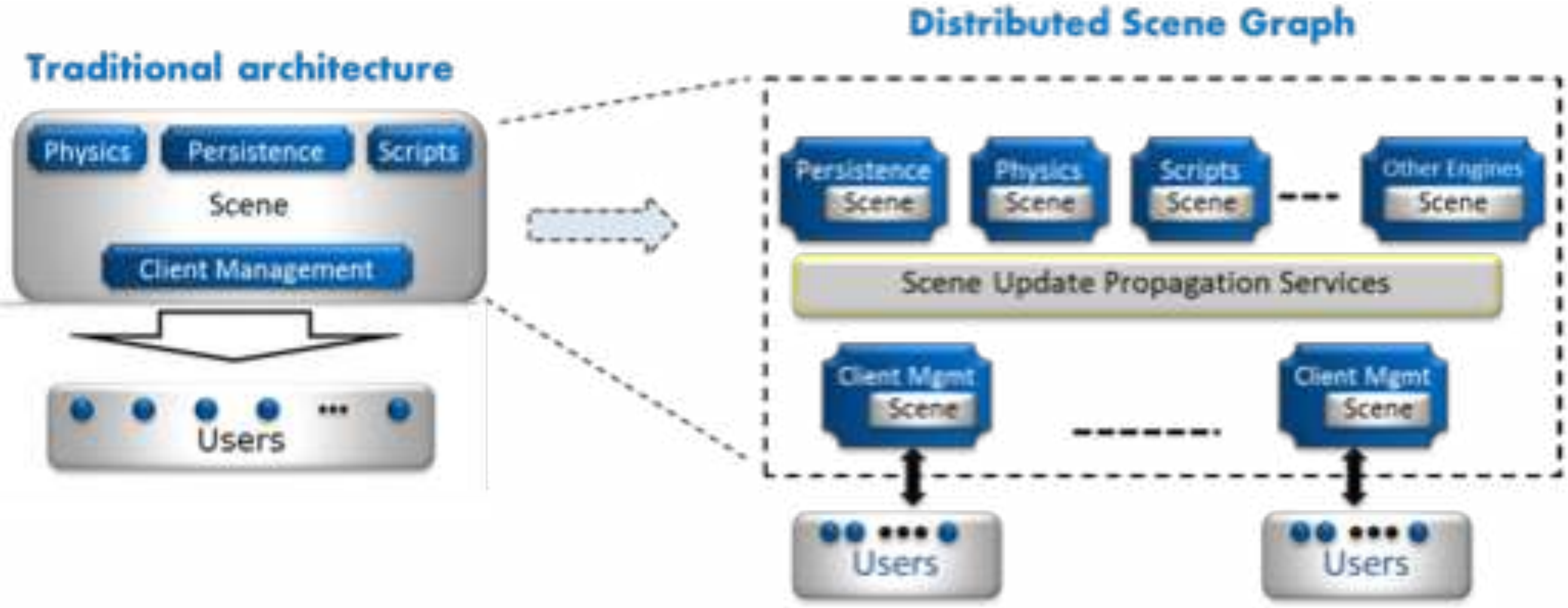


Distributed Scene Graphs (Again)

- For Massively Multiuser Virtual Environments (MMVEs)
- Two types of state manipulations:
  - Transactional state operations (TSO):
    - Modification of shared entities
    - Examples: passing ownership (trading in games), creating/destroying objects
    - Less frequent
    - Require ACID properties: atomicity, consistency, isolation, and durability
  - Self-state updates (SSU):
    - Very frequent (5-30 Hz)
    - Examples: updates of player's character, head and hand tracking, ...
    - Only most recent updates are relevant, i.e., message loss is OK
- Common problem with peer-to-peer: $O(n^2)$ messages

1. Approach: static space partitioning
   - Partition the VE into (geographic) regions
   - Each region is handled by a server
   - Each client (player) can connect to only one server
     - Sees / sends only the updates handled by that server
   - Assumption: clients are distributed across the VE uniformly
2. Approach: distributed data base / distributed hash table
   - Objects of the VE are identified by keys
   - Keys can be mapped to a hash table slot locally by clients
   - Hash table is partitioned among the servers

Overview of System Architecture



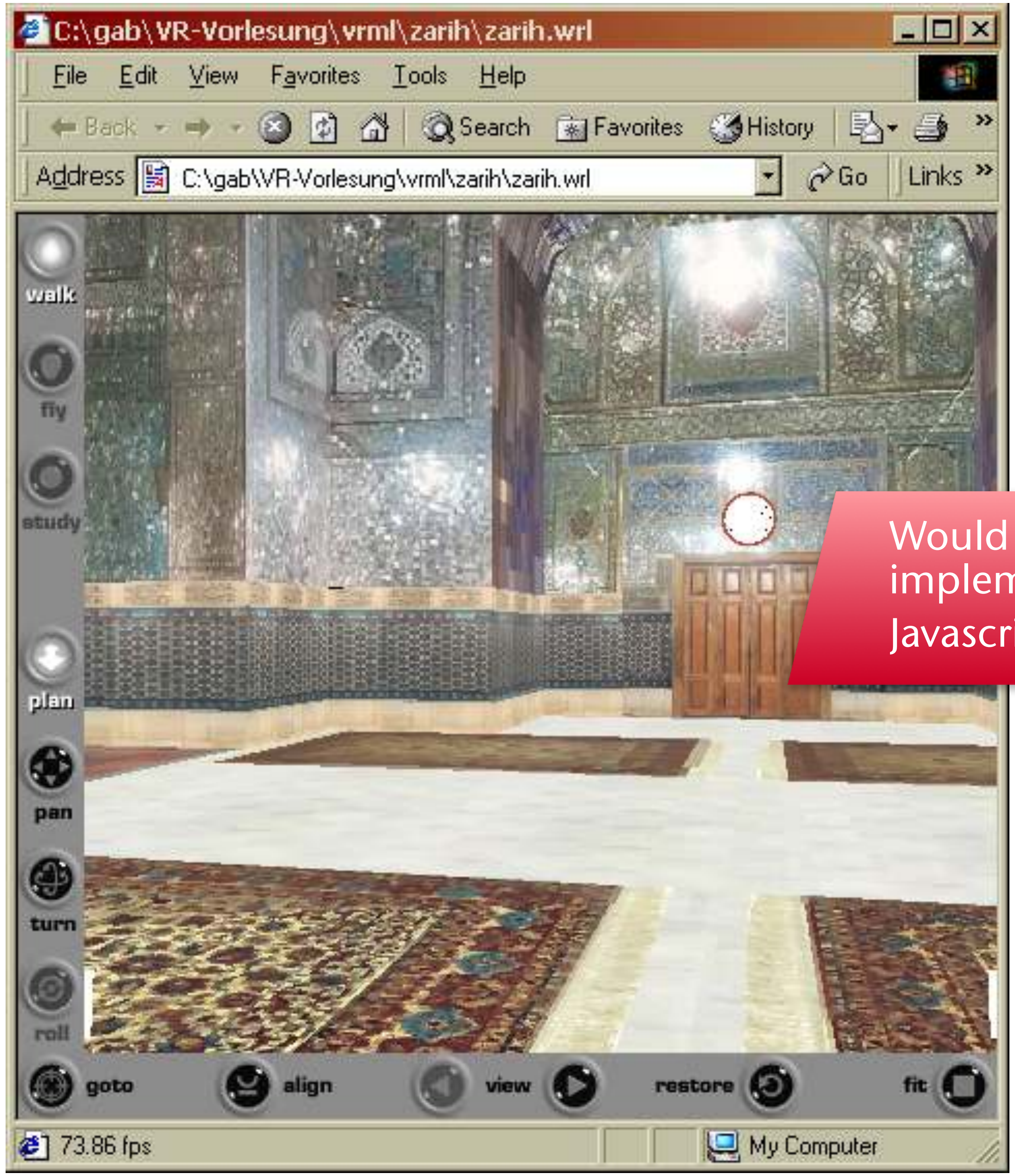
Distrib. KV Pool

Space partitioning or Distributed hash table

Some VRML Demos: Sphere Eversion Video

http://www.youtube.com/watch?v=BVVfs4zKrgk

Demos (Some Applications of WebVR)



Cultural heritage
(Quelle: www.aqrazavi.org)

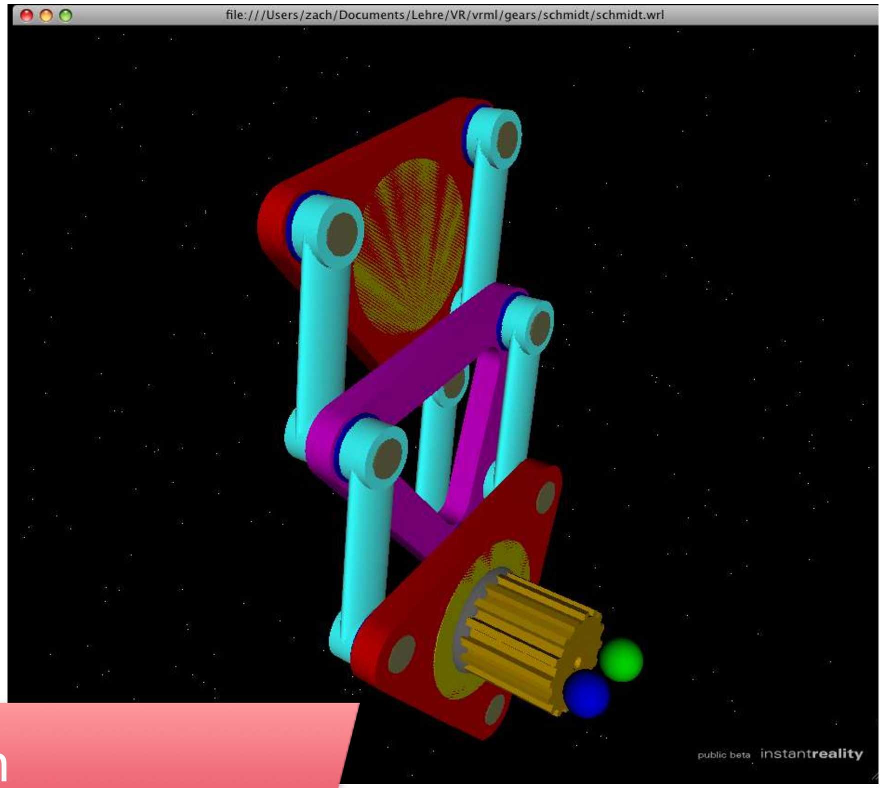
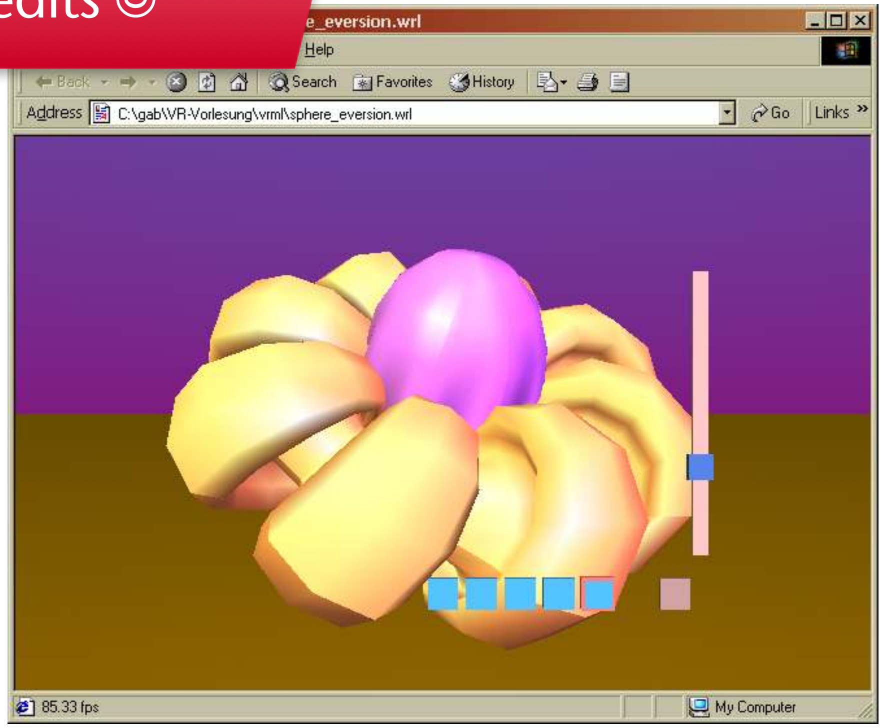
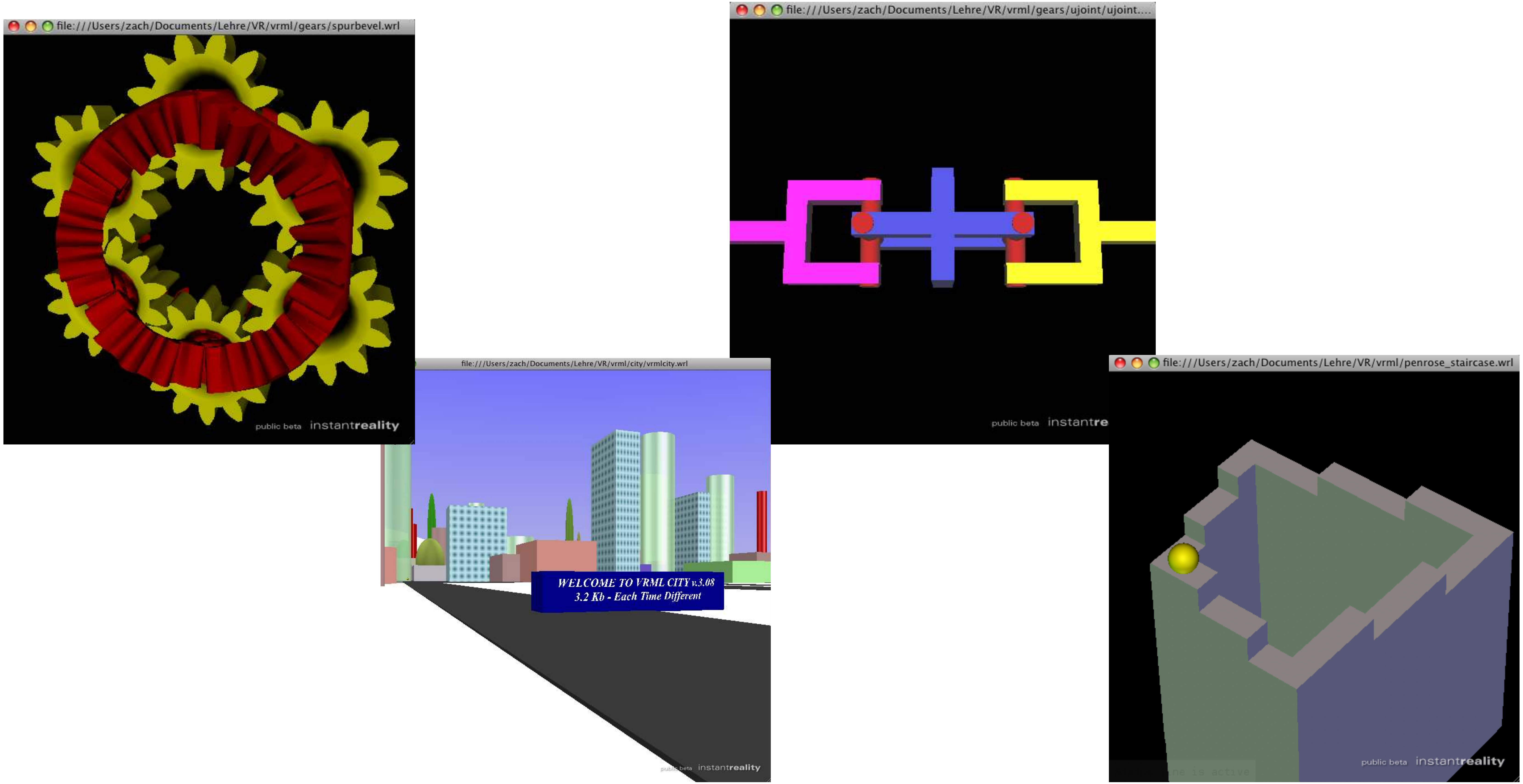


Illustration of complicated mechanics (hier: *Schmidt Offset Coupling*)

Would somebody be interested in implementing them in Unreal (with VR) or Javascript? (for Mac) Credits, credits ☺



Education
Bsp.: *sphere eversion*


file:///Users/zach/Documents/Lehre/VR/vrml/gears/spurbevel.wrl
public beta instantreality
file:///Users/zach/Documents/Lehre/VR/vrml/gears/ujoint/ujoint....
public beta instantre
file:///Users/zach/Documents/Lehre/VR/vrml/city/vrmlcity.wrl
WELCOME TO VRML CITY v.3.08
3.2 Kb - Each Time Different
public beta instantreality
file:///Users/zach/Documents/Lehre/VR/vrml/penrose_staircase.wrl
public beta instantreality