

HCI 558 - lab 9

March 8, 2007

HCI 558 - lab 9 (Mar. 8. 07)

DX Data model (ch. 7)

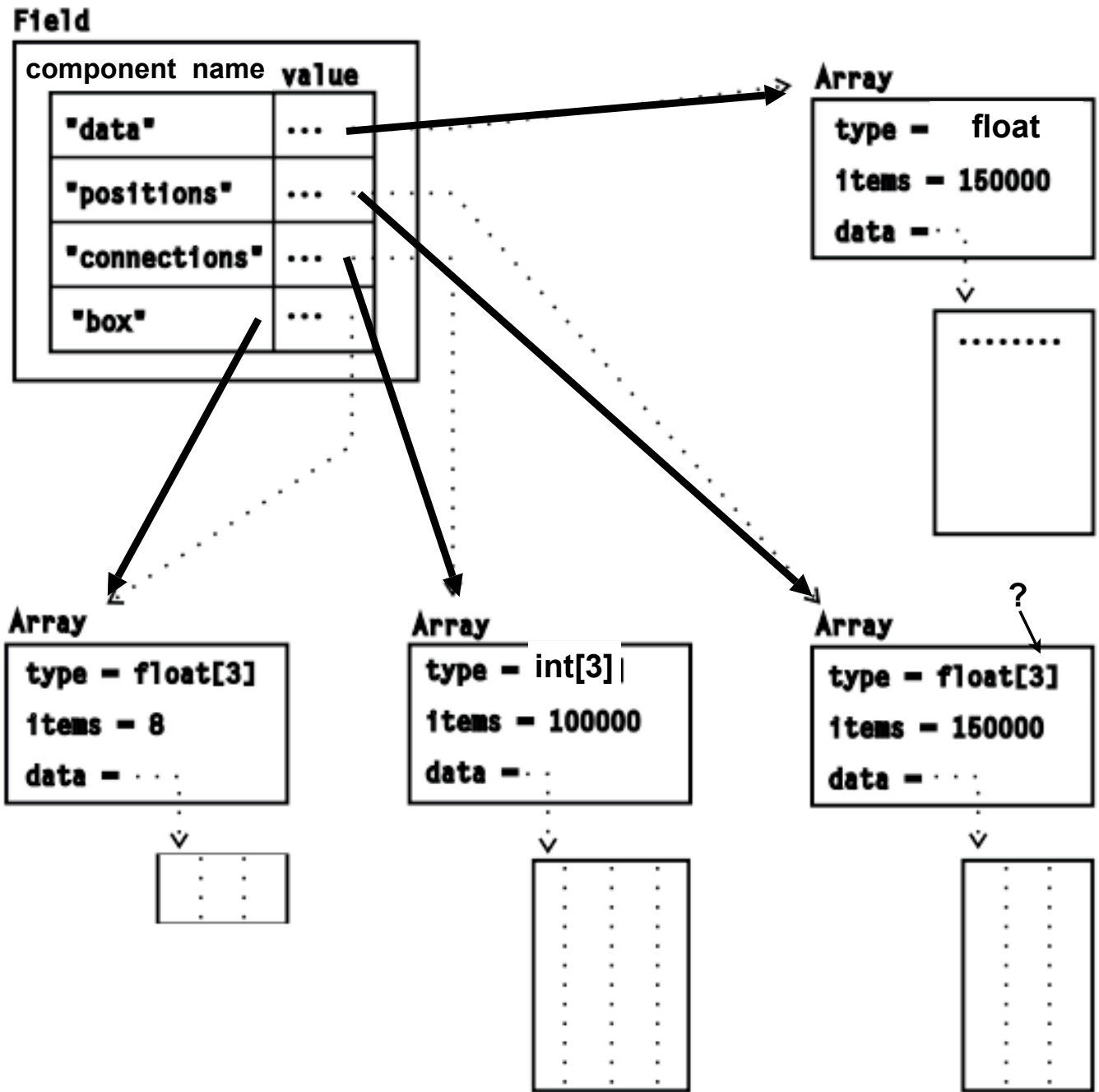
- DX data model: very flexible, very powerful, ... but sometimes very confusing (just a warning :)
- Today: look at major concepts only
- For full details see DX Users guide:
([hci558/DX manuals/DX_userguide.pdf](#))
 - Chapter 3: Understanding the Data Model (pp. 15 - 35)
 - Related: Appendix B (File Formats: pp. 242-279)

Types of objects (p. 105):

- **Groups:** used for “packaging” fields (time series)
- **Fields:** describe spatial data (what is where?)
- **Components:** tie together data values, their positions in space and their connections
- **Attributes:** define relationships between containers (“Container 1 *depends* on Container 3”)
- **Arrays:** container for actual numbers (float, int, index, vector, tensor, ...)

Example of a field object (here: triangles)

(Note: don't confuse the "data" component with the data part of the Array!)



Data component

- look at data/pgm/pgm.net for this example
- Data values (here: ints)
- depends (**dep**) on the positions component:
(where in space is the data? look at positions)
- Could also depend on the connections (result?)

Component number 0, name 'data':

Generic Array. 168 items, unsigned byte, real, scalar
data values:

00 .. 00

Attribute. Name 'dep': String. "positions"

- (dx file export format equivalent in pgm.dx, with colors component in pgm_colored.dx - but this is a bit confusing!)

Positions component

- Defines a structure of 3D points in space
- Any dimensionality (1D - 3D) and type (int, float)
- Here a 27 x 7 “product array” of 2-vector floats (“x,y”)
- Result: 2D grid
- Important: each entry has an index (starting at 0)

Component number 1, name 'positions':

Product Array. 2 terms.

Product term 0:Regular Array.24 items,float,real,2-vector
start value [0, 0], delta [1, 0], for 24 repetitions

Product term 1:Regular Array.7 items,float,real, 2-vector
start value [0, 0], delta [0, -1], for 7 repetitions

Attribute. Name 'dep': String. "positions"

Connections component

- Defines structure (topology) to connect positions
- Uses **list(s) of indices** ('ref') to defines lines, faces or 3D structures (why not dep?)
- ref: needs index table setup, dep: direct "pointer"
- Here: a 24 x 7 2D Mesh array of quads

Component number 2, name 'connections':

Mesh Array. 2 terms.

Mesh offset: 0, 0

Mesh term 0: Path Array. connects 24 items

Mesh term 1: Path Array. connects 7 items

Attribute. Name 'element type': String. "quads"

Attribute. Name 'dep': String. "connections"

Attribute. Name 'ref': String. "positions"

Box component

- Bounding box around data
- Here: Coordinates of the 4 corners
- Will recalculate itself when positions change

Component number 3, name 'box':

Generic Array. 4 items, float, real, 2-vector
data values:

0	-6
0	0
23	-6
23	0

Attribute. Name 'der': String. "positions"

DX Attributes

- How to read them:

Attribute. Name 'dep' : type of relationship?

String. Name "connections" : with whom?

- Most common types:
 - **Dep**: depends on a component
 - **Ref**: uses **index** from a component
 - **Der**: derived from a component, may need recalculation when component changes

Paper & pencil exercise

- Lets sketch the main components of a connection (edges) and a position (point) dependent **2 x 2 point grid** (“memory content” only)
- pay special attention to dep vs. ref
- Positions component:



- Connections component



- Data component (rgb, use as color later)



Array Objects

- Example: array **type** float **rank** 1 **shape** 3 **items** 168
- Type:, **float**, **int**, uint, short, ushort, byte, ubyte, and **string**.
- (Category: real or complex)
- Rank: (of each data element)
0 = scalars, 1 = vectors (“list”) 2 to matrices (“grid”),
- Shape (“Dimension”):
Rank 0 -> Shape 0, Rank 1 -> Shape 1 or 2 or 3
- Items: number of data elements (scalars, vectors)
- Examples:
- **type** int **rank** 0 **items** 4: list of scalars (integers): 3 4 5 6
- **type** float **rank** 1 **shape** 3 **items** 12:
list of 3-vectors: (1.2 0.4 2.1) (0.3 1.4 0.1) (x y z) ...

Module “Flow” rules

- each “line” can carry many different components
- each module tab needs a certain type of module as input
- (some module are OK without any input)
- a module operates on 1 or more of the incoming components
- module will:
 - add, delete or change some components
 - let all other components pass through