Xflow

Declarative Data Processing for the Web

Felix Klein, Kristian Sons, Dmitri Rubinstein, Sergiy Byelozyorov, Stefan John, Philipp Slusallek

Motivation

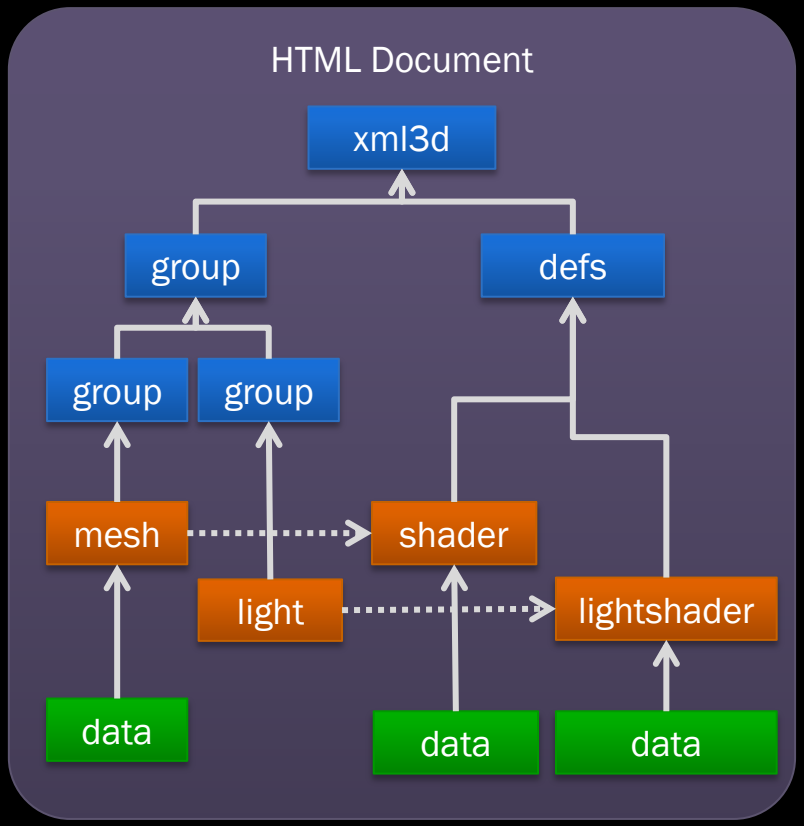
3D applications on the Web

- Web evolved into a full-fledged application platform
- 3D graphics HW support is now everywhere (incl. mobile)
- But still hard to create 3D application for the Web
- WebGL is too low-level for web developers!

We need declarative 3D graphics for the Web

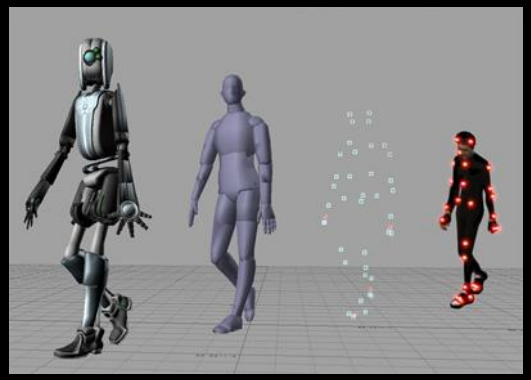
XML3D

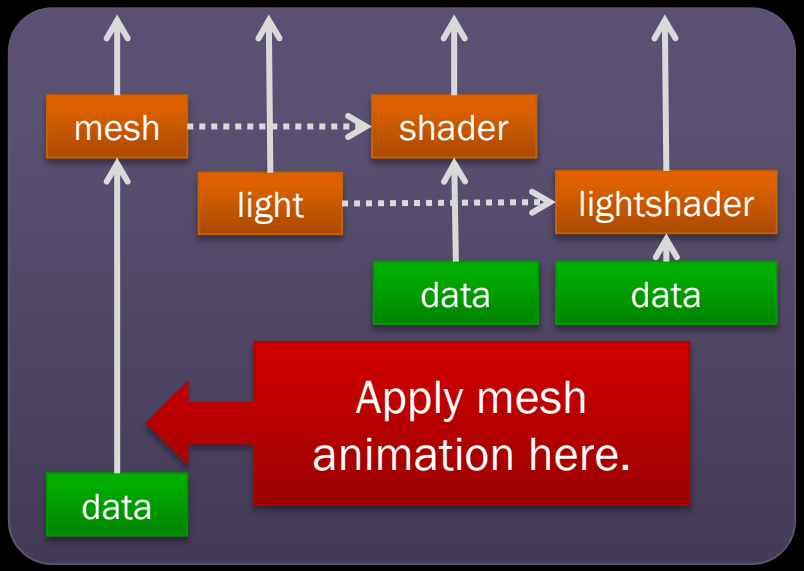
- 3D Scene Graph inside of HTML document
  - Maximize reuse of Web technology (DOM integration)
  - Keep generic to match modern graphics hardware
- Limitation: animations
  - Modify transformations via JS for rigid body animation
- But what about more complex mesh animations?



XML3D + Mesh Animation

- Want to support mesh deformation methods
  - Skinning
  - Morphing
- High-performance data processing tasks
  - Should be applied somewhere between mesh and data source.



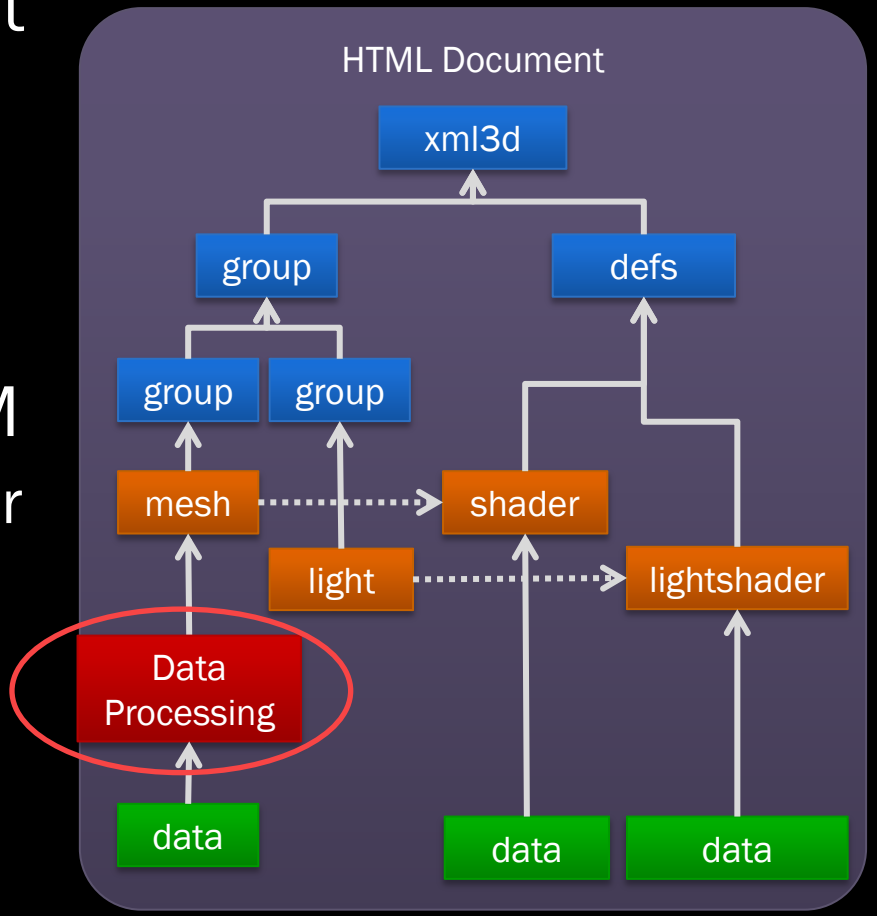


Options for data processing on the Web

- JavaScript
  - To slow for many use-cases
- Low-level API (WebCL, GLSL)
  - Flexible, high entry barrier for Web developers
- Web Workers
  - Thread-Level Parallelism
- River Trail
  - Data-Level Parallelism with JS, internally OpenCL
- All these options are imperative

Advantages of declarative approach

- Part of the Web document
  - Can take advantage of all Web technology
  - Most intuitive for web developers
- Can be accessed via DOM
  - Analyze data processing for visualization
  - Modify data processing via JS at run time



Xflow
Declarative Data Processing

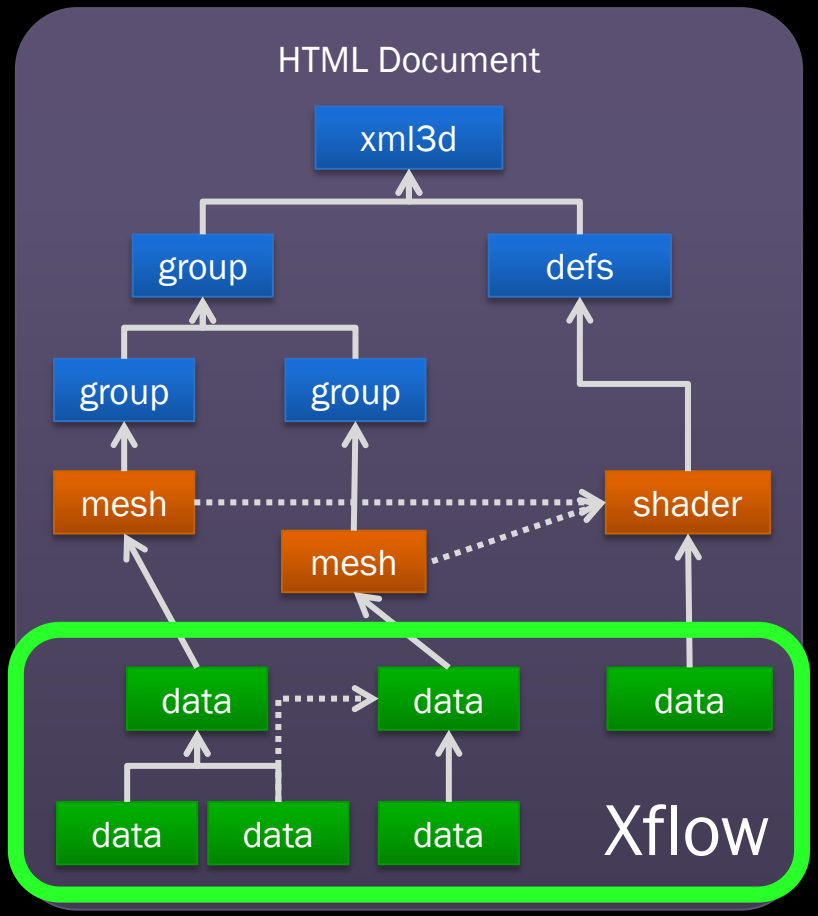
- Declare dataflows to describe data modifications
- Similar to other approaches
  - X3D, SVG Filter Effects
- Improved dataflow design

Efficiency	Match modern hardware, parallelized
Flexibility	No domain-specific functionality
Usability	Ease of use for Web developer

Design

Design Overview

- Xflow as part of XML3D
  - Declared in HTML document
  - Set of nodes to declare dataflow
    - Data input
    - Operations on data
  - Connected to sink nodes
    - <mesh>, <shader>, <lightshader>



Data Input Elements

- Generic data types
  - Each data element has a name
- Declare sequence of input elements
  - Useful for key frame data

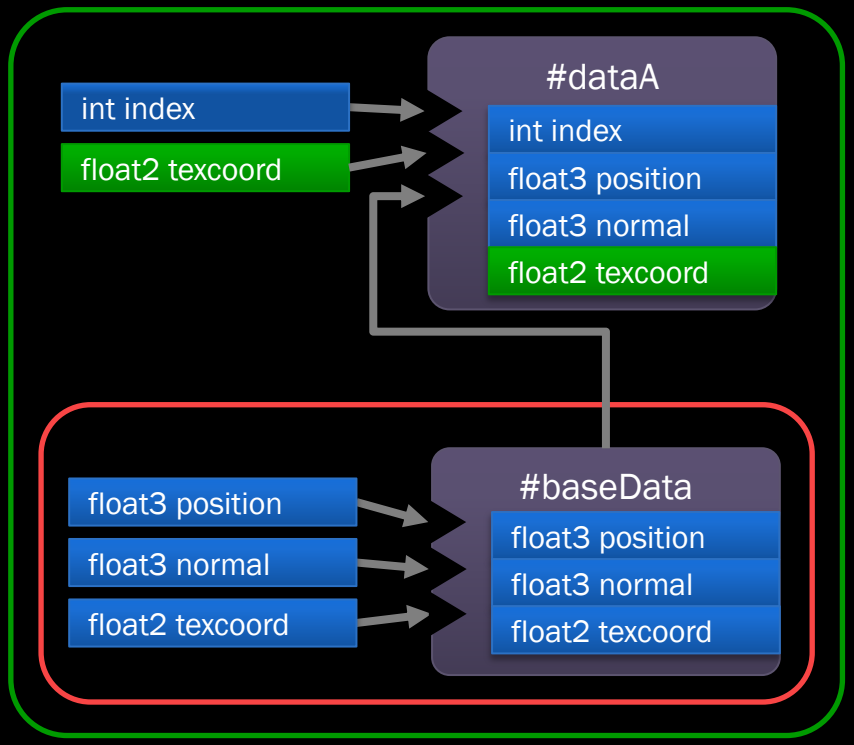
- float
- float2
- float3
- float4
- float3x3
- float4x4
- int
- bool
- texture

Code

```
<data id="baseData">
        <float3 name="position" >1.0 0.04 -0.5 ...</float3>
        <float3 name="normal" >0 1 0 ...</float3>
        <float2 name="texcoord" >0.0 0.0 ...</float2>
</data>


<float3 name="translation" seqnr="0.0">0 0 0 ...</float3>
<float3 name="translation" seqnr="1.2">0 3 0 ...</float3>
<float3 name="translation" seqnr="2.0">0 6 0 ...</float3>
```

Data Compositing

- Combine data from nested elements
- Compositing rules
  - Values with the same name are overridden
  - Values can be renamed to avoid conflict

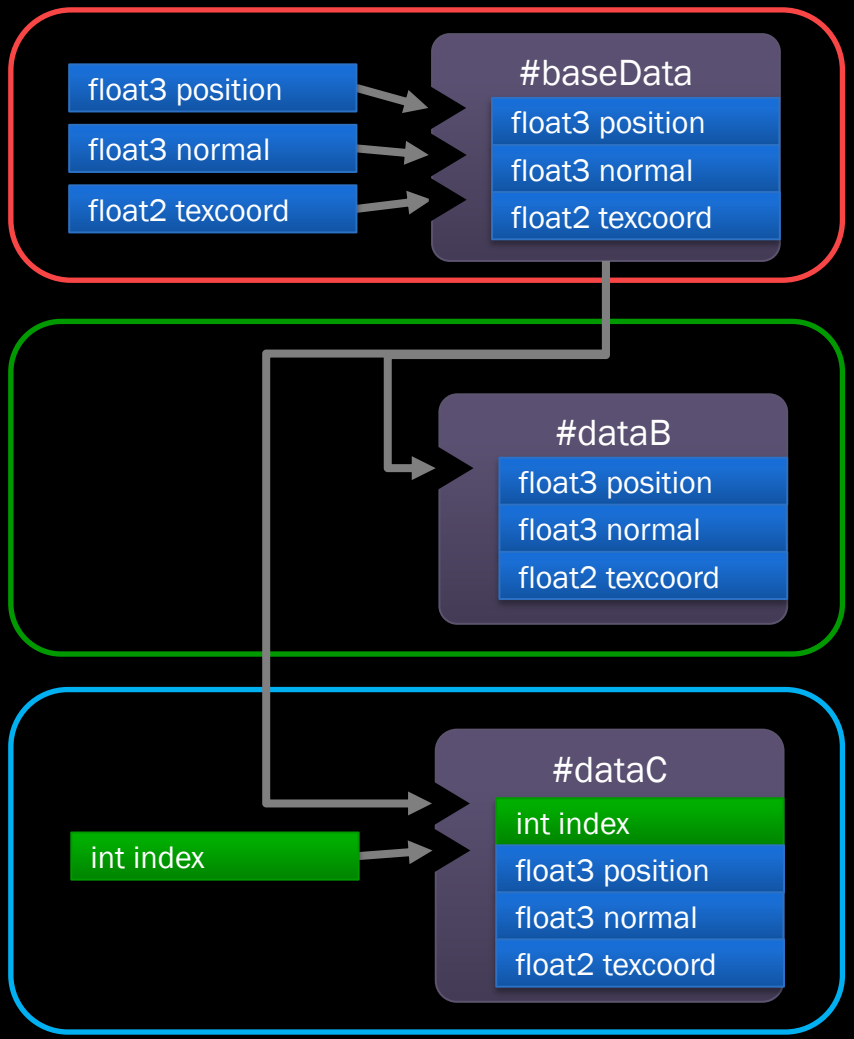


Code

```
<data id="dataA">
        <int name="index" >0 1 2 ...</int>
        <float2 name="texcoord" >1.0 0.5 ..</float2>
        <data id="baseData">
                <float3 name="position" >1.0 0.04 -0.5 ...</float3>
                <float3 name="normal" >0 1 0 ...</float3>
                <float2 name="texcoord" >0.0 0.0 ...</float2>
        </data>
</data>
```

Data References & Reuse

- Reference whole <data> element via src attribute
- Still possible to modify reused data by adding / replacing individual fields



Code

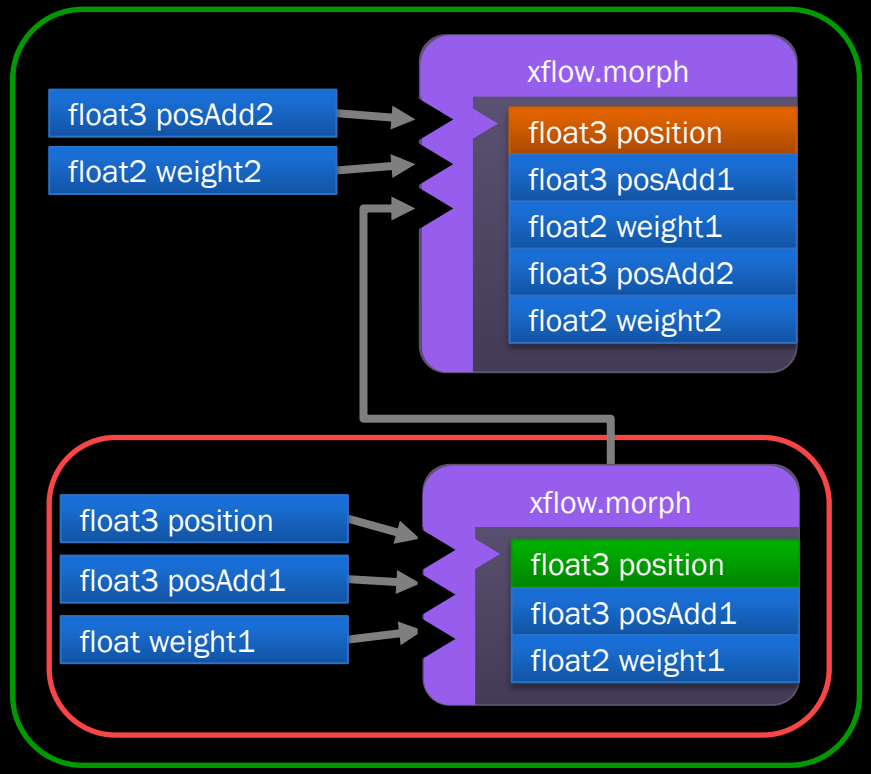
```
<data id="baseData">
        <float3 name="position" >1.0 0.04 -0.5 ...</float3>
        <float3 name="normal" >0 1 0 ...</float3>
        <float2 name="texcoord" >0.0 0.0 ...</float2>
</data>

<data id="dataB" src="#baseData" />

<data id="dataC" >
        <data src="#baseData" />
        <int name="index" >10 11 12 ...</int>
</data>
```

Data Processing

- Apply operator on <data> via compute attribute
- Assign argument via list or name-map
- Result of operators is added to result set of <data>, overriding values by name

Code

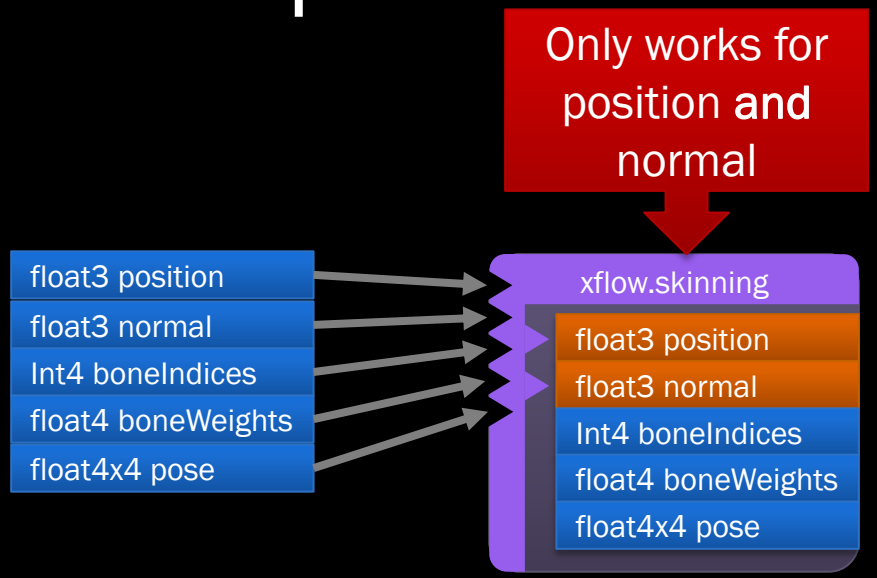
```
<data compute="position = xflow.morph({
            pos: position,
            posAdd: posAdd2,
            weight: weight2})" >

    <float3 name="posAdd2" >1.0 0.0 0.0 ...</float3>
    <float name="weight2" >0.6</float>

    <data compute="position = xflow.morph(
            position, posAdd1, weight1)" >

        <float3 name="position" >1.0 0.04 -0.5 ...</float3>
        <float3 name="posAdd1" >0.0 1.0 2.0 ...</float3>
        <float name="weight1" >0.35</float>

    </data>
</data>
```



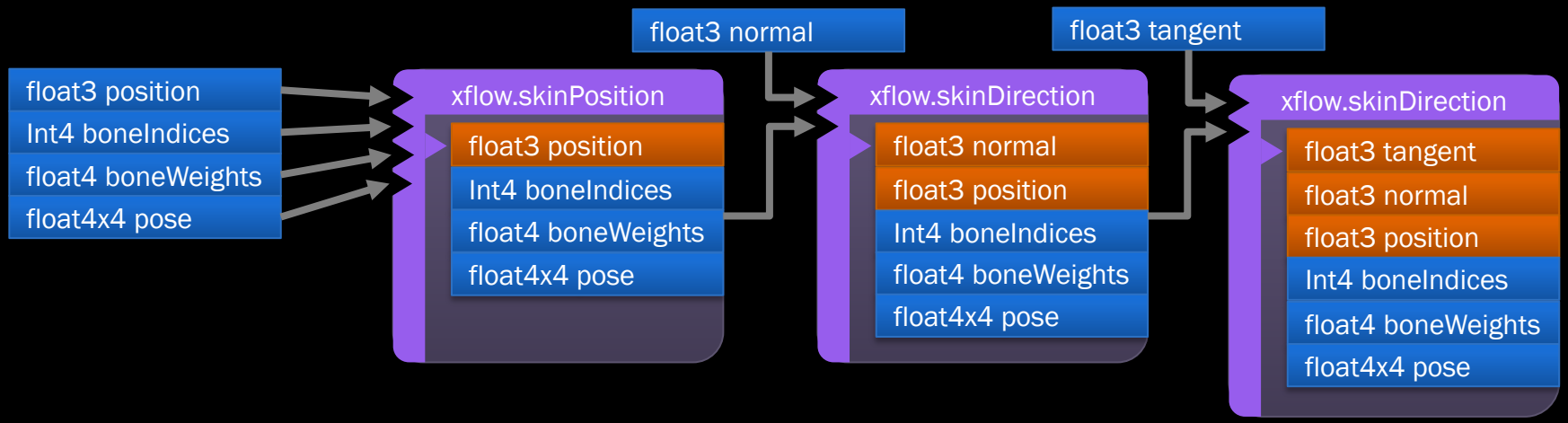
Small & Generic Operators

Flexibility

- Avoid too specific operators:



- Better: small generic operators:

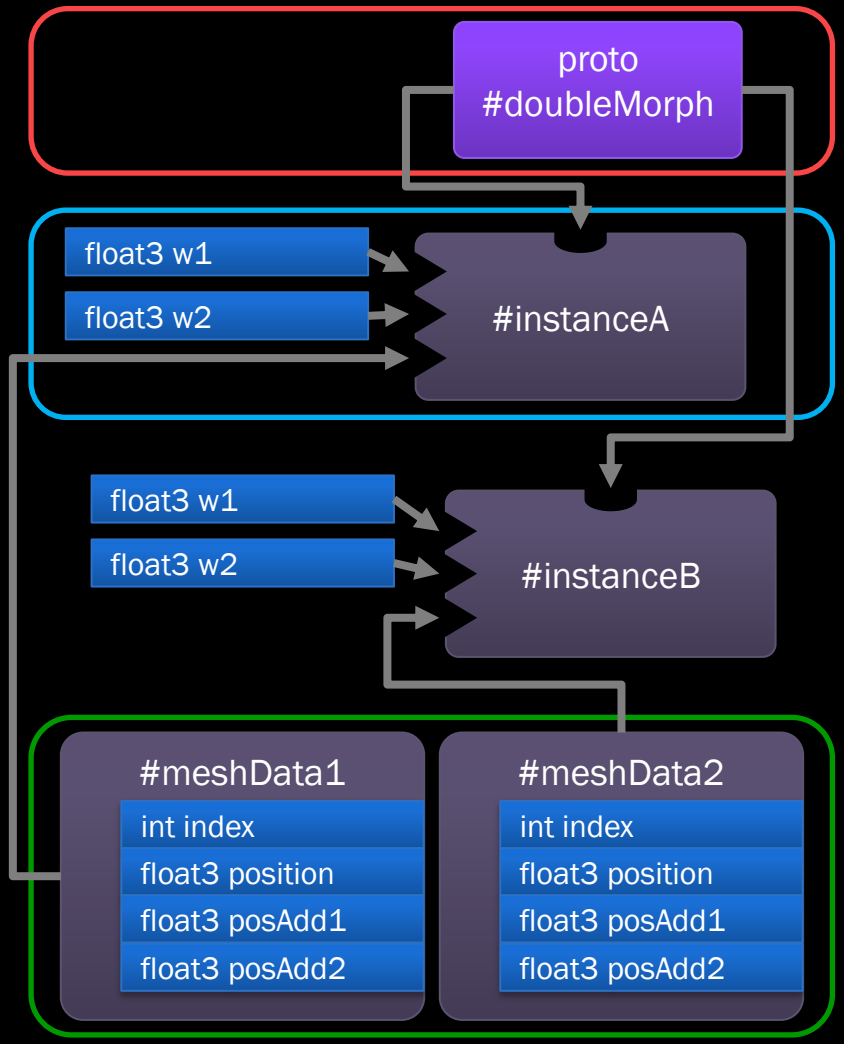


Prototypes

- Instantiate dataflow with different input data sets

Code

```
<proto id="doubleMorph"
        compute="position = xflow.morph(position, posAdd2,w1)" >
         <data compute="position = xflow.morph(pos, posAdd1, w2)" >
                  <float3 param="true" name="position" ></float3>
                  <float3 param="true" name="posAdd1" ></float3>
                  <float3 param="true" name="posAdd2" ></float3>
                  <float param="true" name="w1" ></float>
                  <float param="true" name="w2" ></float>
         </data>
</proto>

<data id="instanceA" proto="doubleMorph" >
         <data src="#meshData1" >
         <float name="w1" >0</float>
         <float name="w2" >0</float>
</data>

<data id="meshData1" >
         <int name="index" >0 1 2 ...</int>
         <float3 name="position" >1.0 0.04 -0.5 ...</float3>
         <float3 name="posAdd1" >0.0 1.0 2.0 ...</float3>
         <float3 name="posAdd1" >0.0 1.0 2.0 ...</float3>
</data>
```



Connection to Scene Graph

- Connect data to *sink elements:* <mesh>, <shader>, <lightshader> etc.
  - *just like <data>, but no further output*

Code

```
<defs>
        <data id="#meshData" >
                <int name="index">0 1 2 ...</int>
                ...
        </data>
        <shader id="wood" script="urn:xml3d:shader:phong">
                <float3 name="diffuseColor">0.4 0.3 0</float3>
                <float3 name="specularColor">0 0 0</float3>
                ...
        </shader>
</defs>

<group style="transform: url(#t1); shader: url(#wood)" >
        <mesh type="triangles" src="#meshData" />
</group>
<group style="transform: url(#t2); shader: url(#wood)" >
        <mesh type="triangles" >
                <data src="#meshData" />
                <float3 name="color">0.5 0.24 0.6 ...</float3>
        </mesh>
</group>
```

Generic data block

Inline declaration

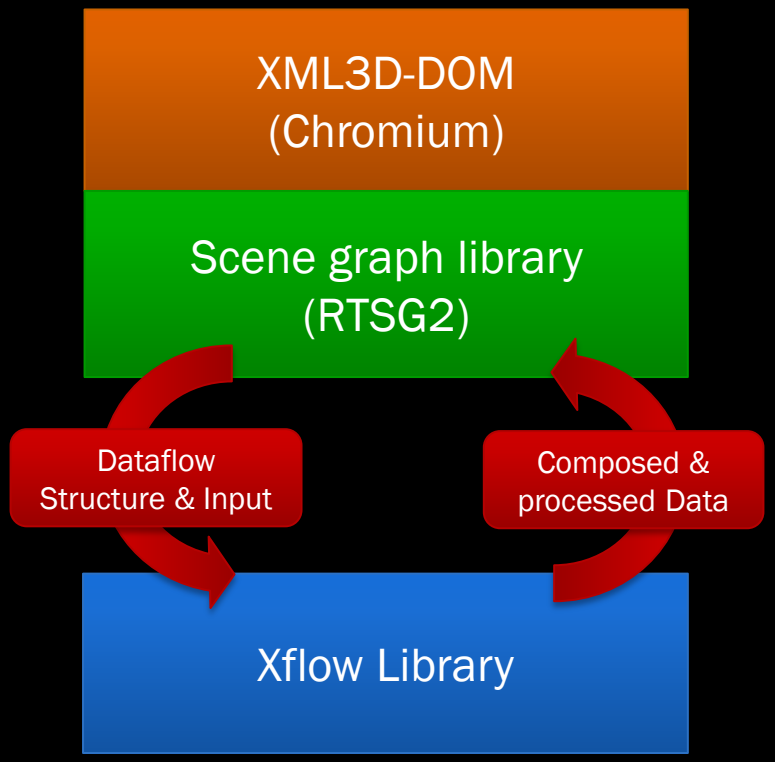
Refer to a data block via doc id

Refer to a data block and extend

Implementation

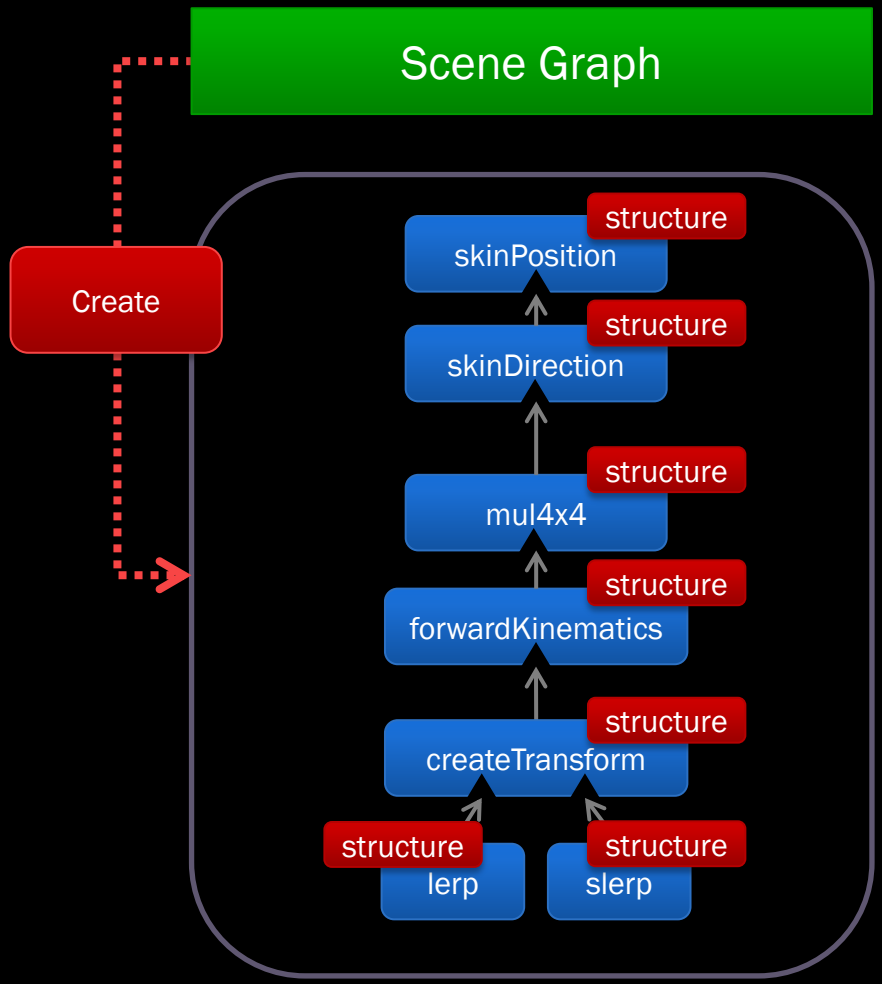
System Overview

- Two implementations:
  - JavaScript + WebGL
  - Native Implementation
- Separate Xflow library
  - Communicates with scene graph library
  - Create dataflow
  - Optimizes computation
  - Request processed data



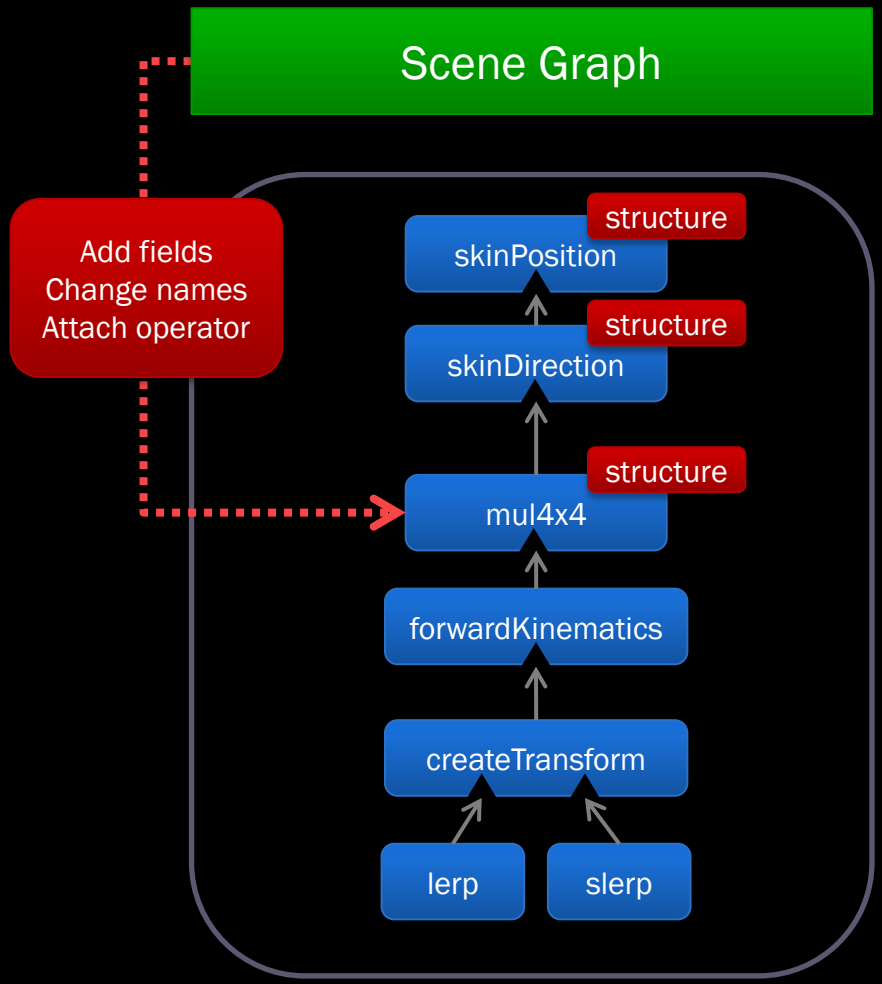
Dataflow Execution Model

- Push-based update phase
  - Nodes are only marked as out-of-sync when modified / created
  - No processing is triggered
  - Two states
    - Structure out-of-sync
    - Data out-of-sync



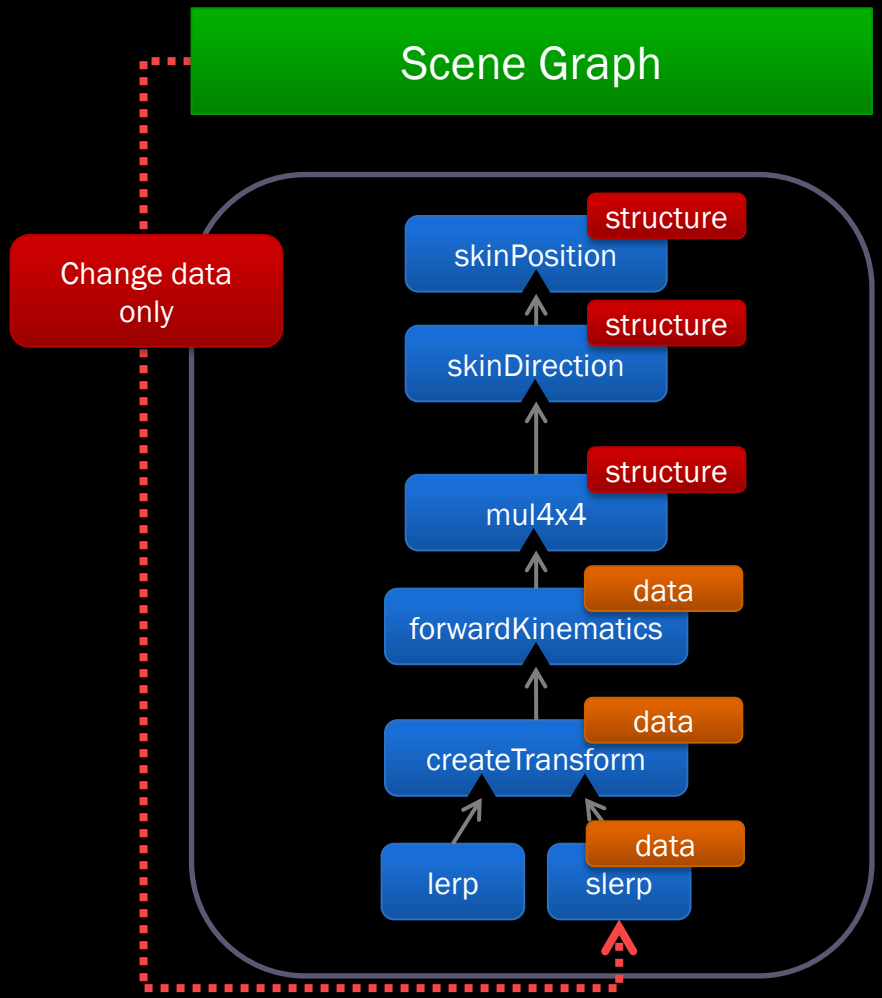
Dataflow Execution Model

- Push-based update phase
  - Nodes are only marked as out-of-sync when modified / created
  - No processing is triggered
  - Two states
    - Structure out-of-sync
    - Data out-of-sync



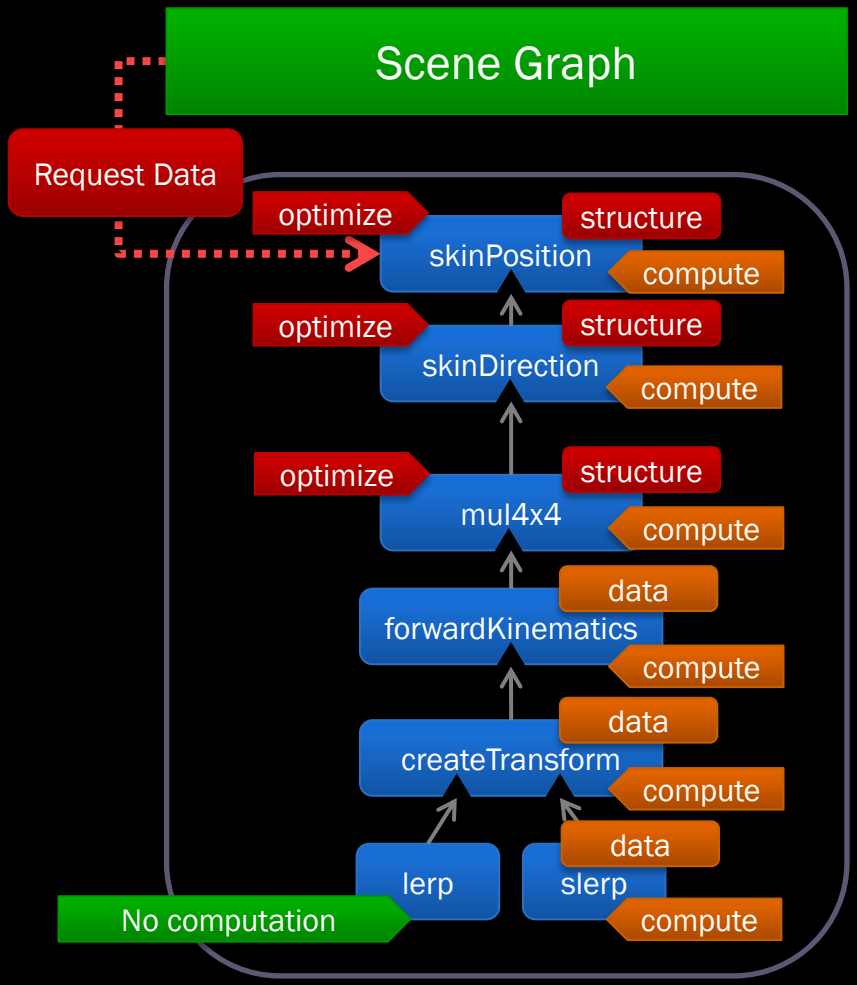
Dataflow Execution Model

- Push-based update phase
  - Nodes are only marked as out-of-sync when modified / created
  - No processing is triggered
  - Two states
    - Structure out-of-sync
    - Data out-of-sync



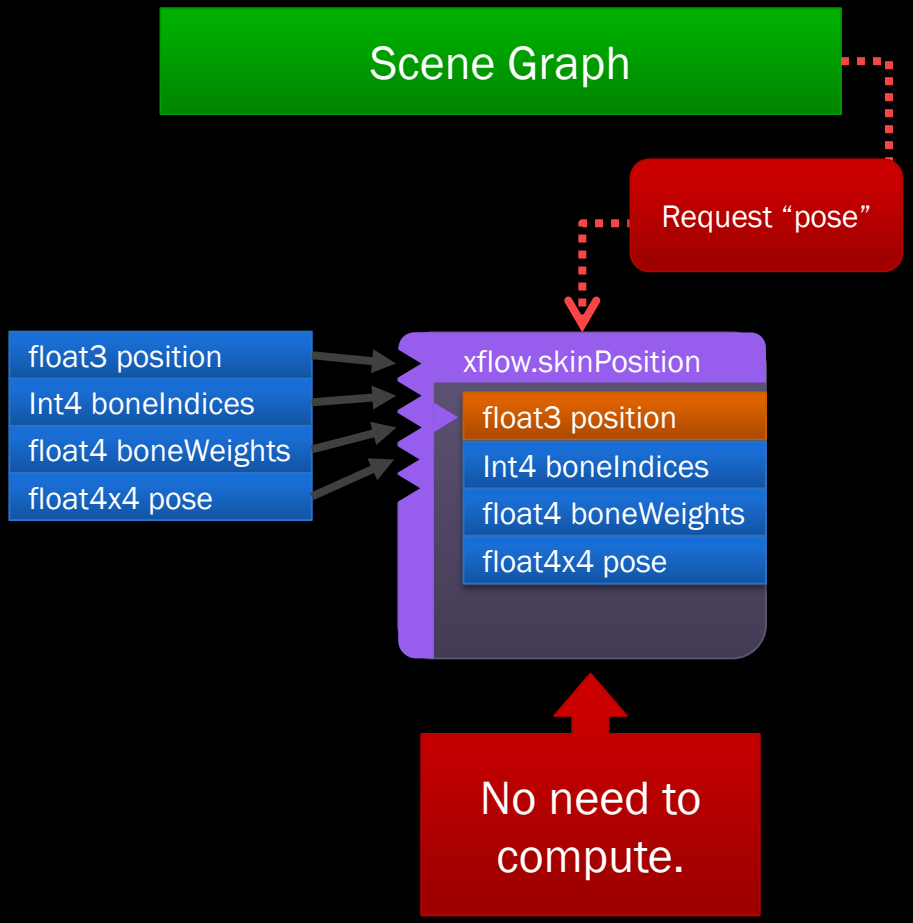
Dataflow Execution Model

- Pull-based execution phase
  - Request data from a node
  - Trigger computation of whole sub-graph
    - Optimize internal structures beforehand if structure-out-of-sync



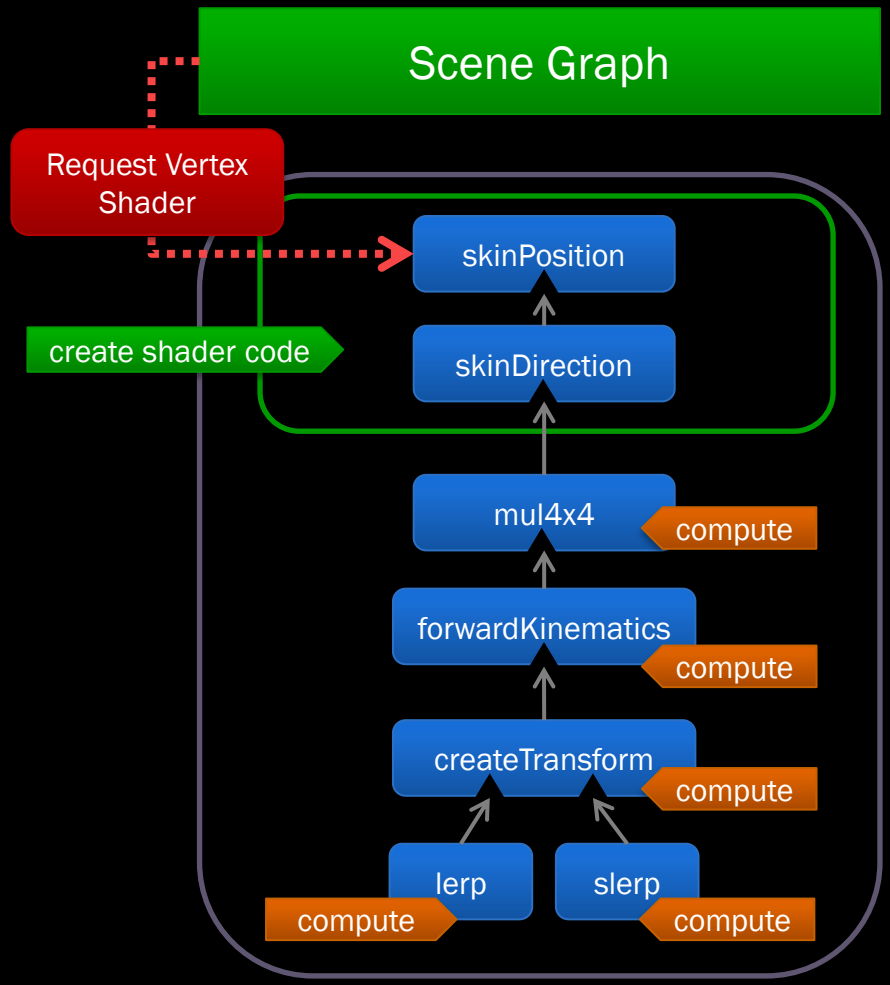
Dataflow Optimizations

- Cache already computed results
- Skip operators that are not required to be executed
  - Request subset of data by providing name set



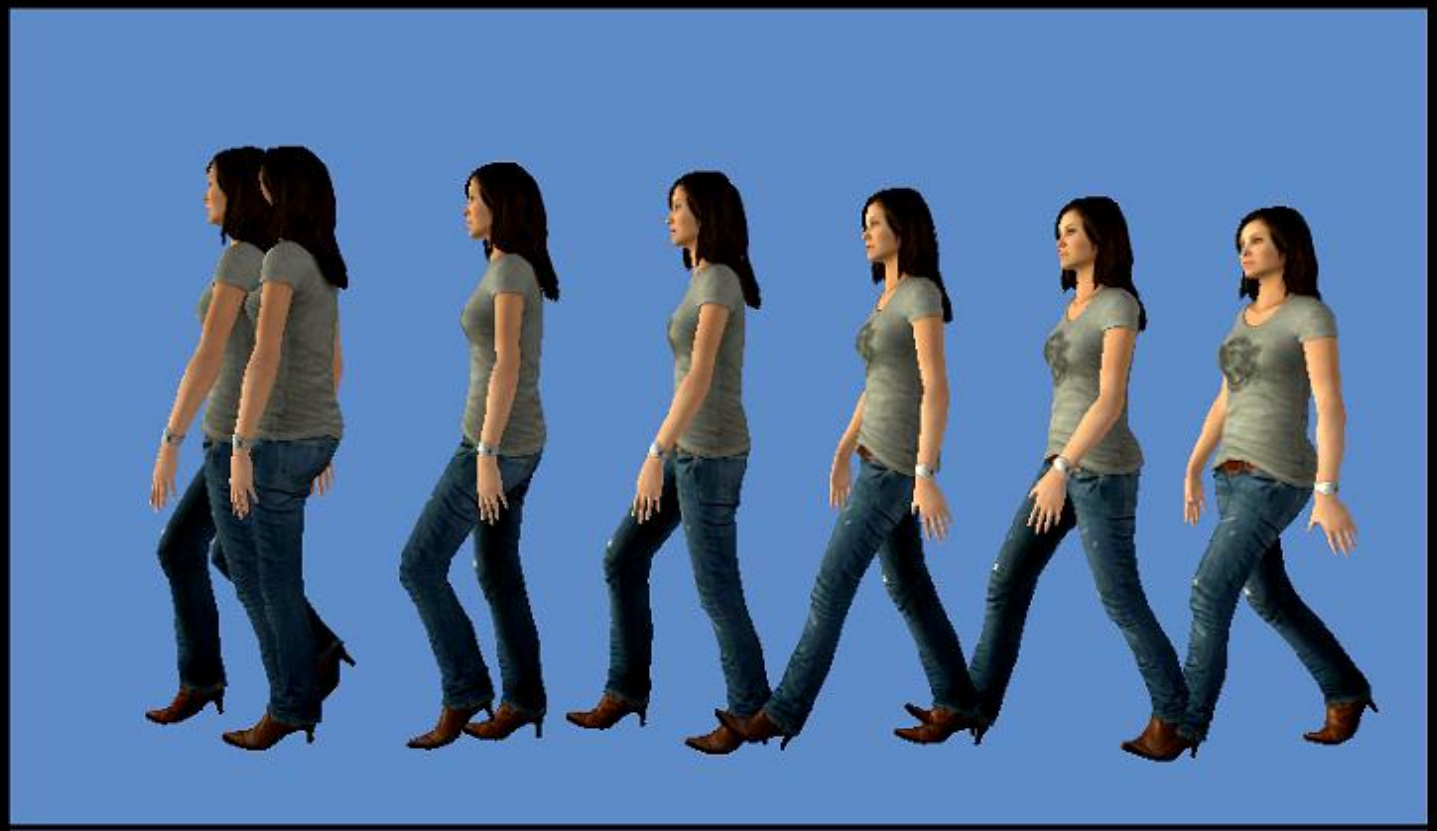
Mapping on the GPU

- Integrate dataflow into render pipeline
  - Create vertex shader from subgraph
    - Only includes per-vertex operations
  - Scene graph provides information for vertex attribute transformation
    - Required for connection with fragment shader



Results

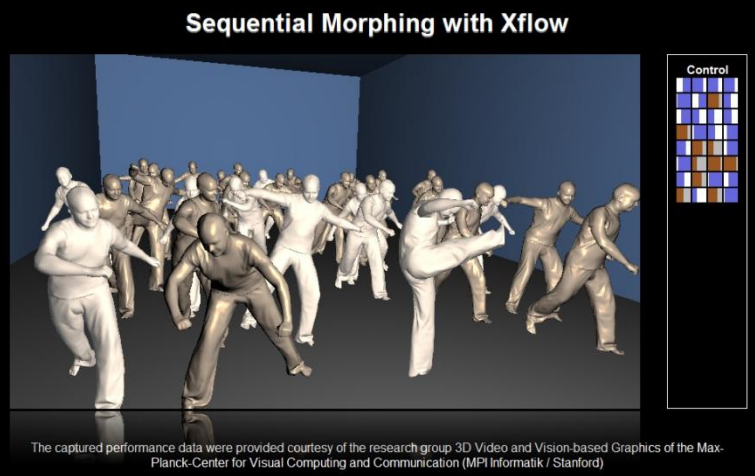
Skeletal Animation with Xflow



Sequential Morphing with Xflow

The captured performance data were provided courtesy of the research group 3D Video and Vision-based Graphics of the Max-Planck-Center for Visual Computing and Communication (MPI Informatik / Stanford)

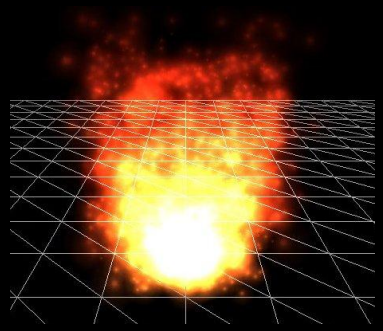
Morphing Performance



Scene	#triangles	FPS (CPU)	FPS (GPU)
16x cloned movement	639,552	92	74
16x individual movement	639,552	32	68
32x individual movement	1,279,104	19	41

Future Work

- Xflow everywhere
  - Surface Shading (Shading graph)
  - Post-Processing (Tone Mapping, SSAO)
  - Image Processing
- Cyclic connections
  - Important for particles, physics, image processing
- Improve River Trail implementation
- Image Geometry with Xflow?



Acknowledgements

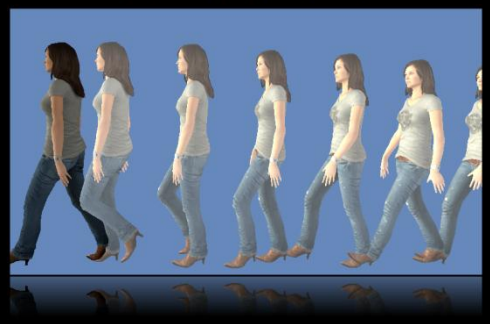
- Intel
  - Funding via Intel Visual Computing Institute
  - Intel Labs, River Trail Team
- EU Project: Future Internet PPP – FI-Content
  - Usage Area: Games & Virtual Environment
  - Collaboration with Disney Research, BBC …
- EU Project: VERVE
  - VR-based patient treatment
  - Realistically animated characters on the Web

Thank you!

Questions?

Performance Native vs. JS

Skeletal Animation with Xflow



Processing Time on CPU in milliseconds

Instances	JavaScript	Nativ w/o VS	Nativ with VS*
16	45	19	1,4
8	23	9,5	0,7
1	3,3	1,2	0,1

*only includes operators executed on CPU (Transform computations)

Xflow Filter Syntax

Code

```
<data filter="rename({pos: position, norm: normal})">
        <float3 name="position" >1.0 0.04 -0.5 ...</float3>
        <float3 name="normal" >0 1 0 ...</float3>
        <float2 name="texcoord" >0.0 0.0 ...</float2>
</data>

<data filter="keep(position, normal)">
        <float3 name="position" >1.0 0.04 -0.5 ...</float3>
        <float3 name="normal" >0 1 0 ...</float3>
        <float2 name="texcoord" >0.0 0.0 ...</float2>
</data>


<data filter="remove(texcoord)">
        <float3 name="position" >1.0 0.04 -0.5 ...</float3>
        <float3 name="normal" >0 1 0 ...</float3>
        <float2 name="texcoord" >0.0 0.0 ...</float2>
</data>
```

Imperative options hard to integrate into Scene Graph

- DOM modifications only straight-forward option
  - Slow for large data-sets
- We also want to extend shaders for data processing
  - Shaders already managed by scene graph
- Extend scene graph to support proper integration