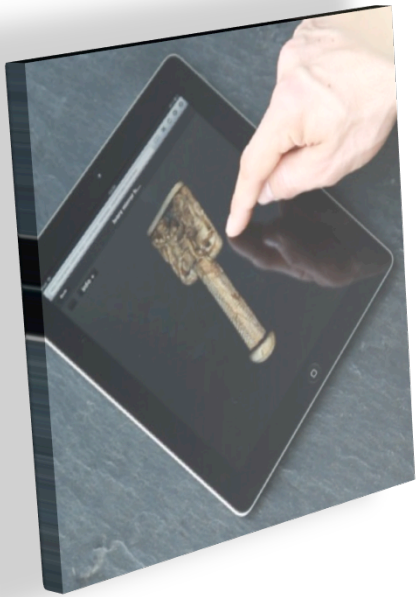
X3DOM – Declarative (X)3D in HTML5

Introduction and Tutorial



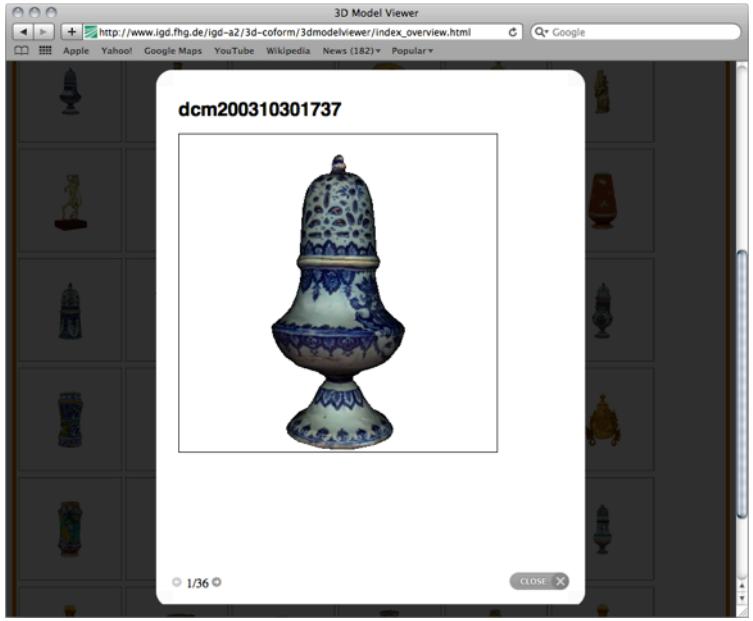
Yvonne Jung
Fraunhofer IGD
Darmstadt, Germany

yvonne.jung@igd.fraunhofer.de
www.igd.fraunhofer.de/vcst

Fraunhofer IGD

3D Information inside the Web

- Websites (have) become Web applications
- Increasing interest in 3D for
  - Product presentation
  - Visualization of abstract information (e.g. time lines)
  - Enriching experience of Cultural Heritage data
- Enhancing user experience with more sophisticated visualizations
  - Today: Adobe Flash-based site with videos
  - Tomorrow: Immersive 3D inside browsers

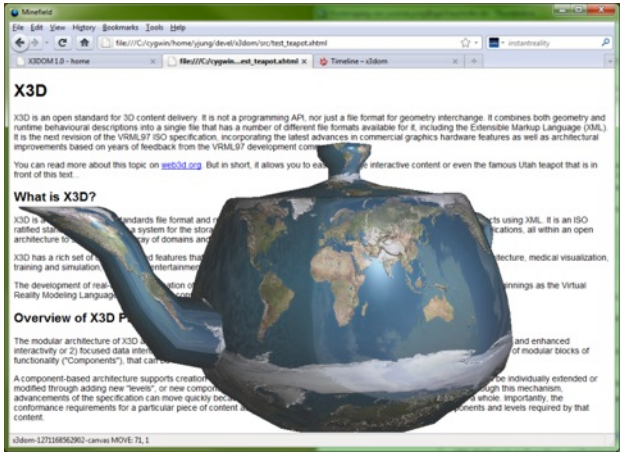


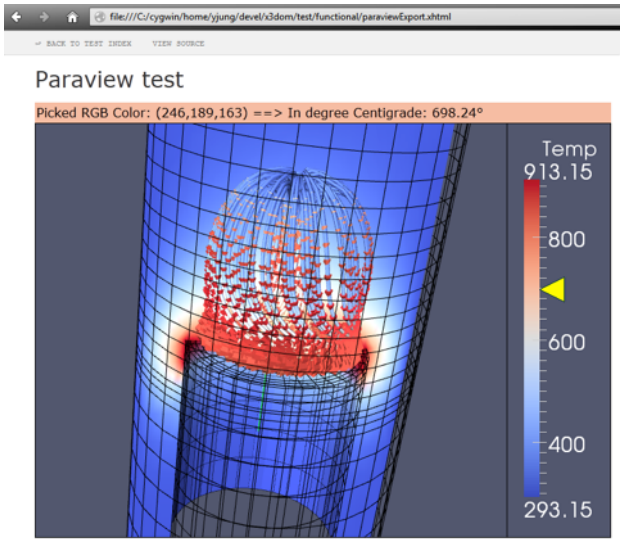
Example Coform3D: line-up of scanned historic 3D objects

Fraunhofer IGD

OpenGL and GLSL in the Web: WebGL

- JavaScript Binding for OpenGL ES 2.0 in Web Browser
  - → Firefox, Chrome, Safari, Opera
- Only GLSL shader based, no fixed function pipeline mehr
  - No variables from GL state
  - No Matrix stack, etc.
- HTML5 <*canvas*> element provides 3D rendering context
  - *gl = canvas.getContext('webgl');*
- API calls via GL object
  - X3D via X3DOM framework
  - http://www.x3dom.org



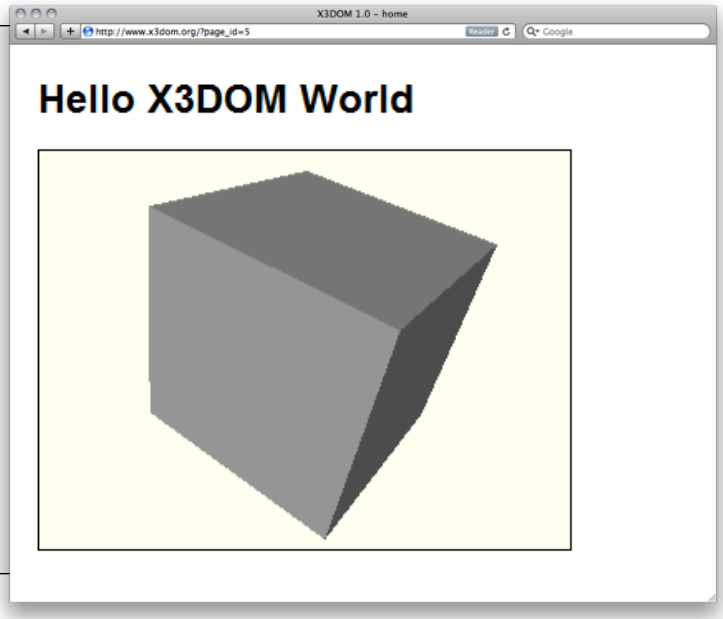


Fraunhofer IGD

X3DOM – Declarative (X)3D in HTML5

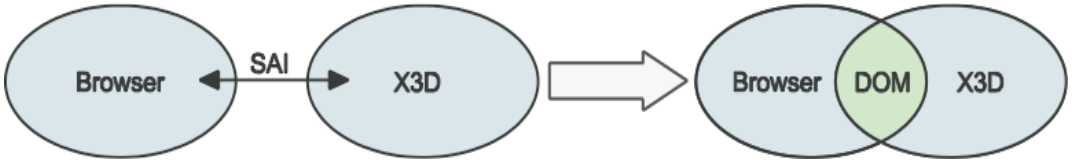
- Allows utilizing well-known JavaScript and DOM infrastructure for 3D
- Brings together both
  - declarative content design as known from web design
  - “old-school” imperative approaches known from game engine development

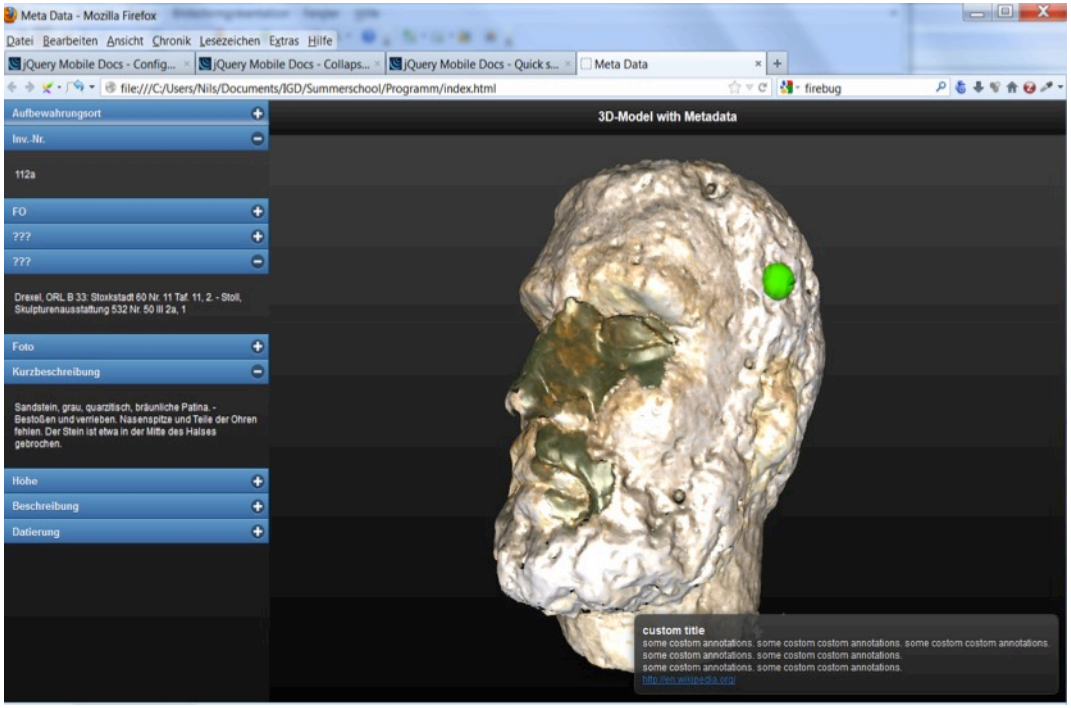
```
<html>
  <body>
    <h1>Hello X3DOM World</h1>
    <x3d>
      <scene>
        <shape>
          <box></box>
        </shape>
      </scene>
    </x3d>
  </body>
</html>
```



Fraunhofer IGD

X3DOM – Declarative (X)3D in HTML5





- X3DOM := X3D + DOM
- DOM-based integration framework for declarative 3D graphics in HTML5
- Seamless integration of 3D contents in Web Browser

X3DOM – Declarative (X)3D in HTML5
Completes todays graphics technologies

	2D (Final HTML5 spec)	3D (No W3C spec yet)
Declarative Scene-graph Part of HTML document DOM Integration CSS / Events		
Imperative Procedural API Drawing context Flexible		

Benefits: Why Declarative 3D in HTML?

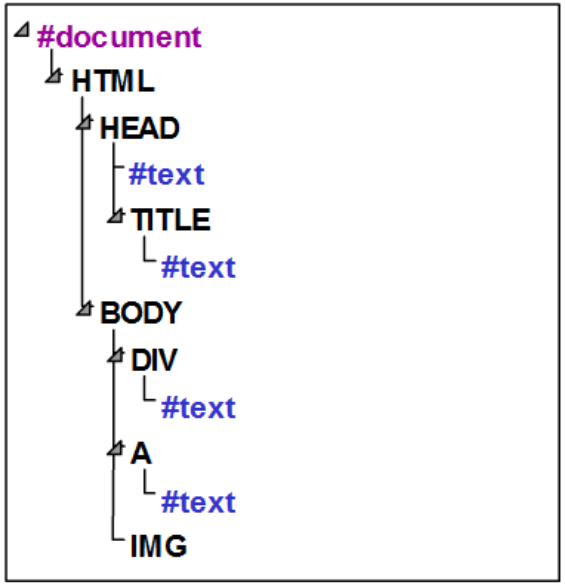
- Native Web Bbrowser integration
  - Plugin/ App free
  - No issues with user permissions, installation, and security
  - OS independent, especially on mobile devices
    - Cluttered: Symbian, Windows Phone, Android, iOS, …
- Web Browsers for most devices available
  - Browser already provides complete deployment structure
  - Eases proliferation of technology and accessibility of content
  - No special APIs (such as in game engines)
  - No expert knowledge required (OpenGL, mathematics, …)
- Integrates with standard Web techniques (e.g. DHTML, Ajax)

Benefits: Why Declarative 3D in HTML?

- Declarative, open, human-readable (wraps low-level graphics)
  - Utilizing standard Web APIs for integrating content and user interactions
  - Open architectures (also for authoring) and ease of access
- Integration into HTML document instead of closed systems
  - Metadata: index and search “content” on WebGL apps?
  - Allows “mash-ups” (i.e. recombination of existing contents)
  - Open formats enable automated connection of existing data (e.g., geo-information, Flickr) with 3D content
- Unify 2D and 3D media development
  - Declarative content description
  - Flexible content (cultural heritage, industry,…)
  - Interoperability: Write once, run anywhere (web/ desktop/ mobile)
  - Rapid application development

Excursus: Web-based APIs and DOM

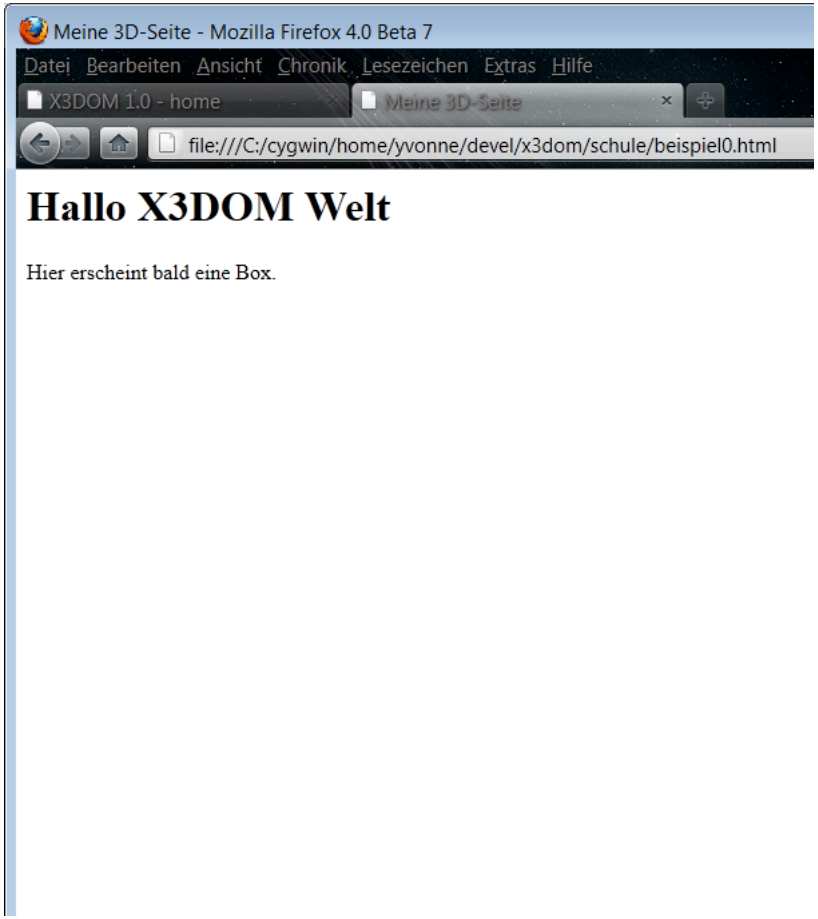
- Browser provides complete deployment structure
- Document Object Model (DOM) is standardized interface that allows manipulating content, structure and style of (X)HTML/ XML documents
- Document is structured as tree with nodes
  - document.getElementById(„myID“);
- Nodes/ tags and attributes can be added, removed and modified (usually with JavaScript)
  - document.createElement(), appendChild(), removeChild()
  - setAttribute(), getAttribute()
- UI events (e.g. ‘mouseover’) can be attached to most elements (e.g. <img>, <a>, , etc.)
- Separation of style and content via CSS



DOM structure (example)

Short introduction of HTML

```
<html>
   <head>
      <title>My 3D page</title>
   </head>
   <body>
      <h1>Hello X3DOM World</h1>
      <p>
         A blue box will soon appear.
      </p>
   </body>
</html>
```

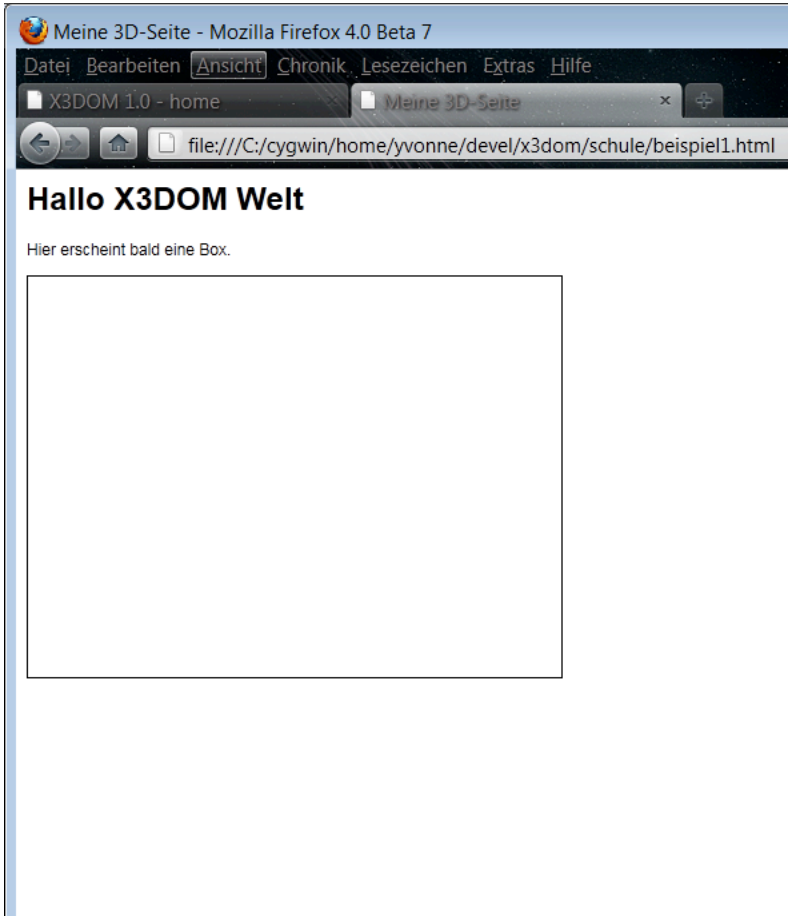


First HTML needs to know about (X)3D

```
<html>
  <head>
    <title>My 3D page</title>
     <link rel="stylesheet" type="text/css"
         href="http://www.x3dom.org/x3dom/release/x3dom.css">
    </link>
    <script type="text/javascript"
          src="http://www.x3dom.org/x3dom/release/x3dom.js">
    </script>
  </head>
  …
```

3D only works inside the <X3D> tag

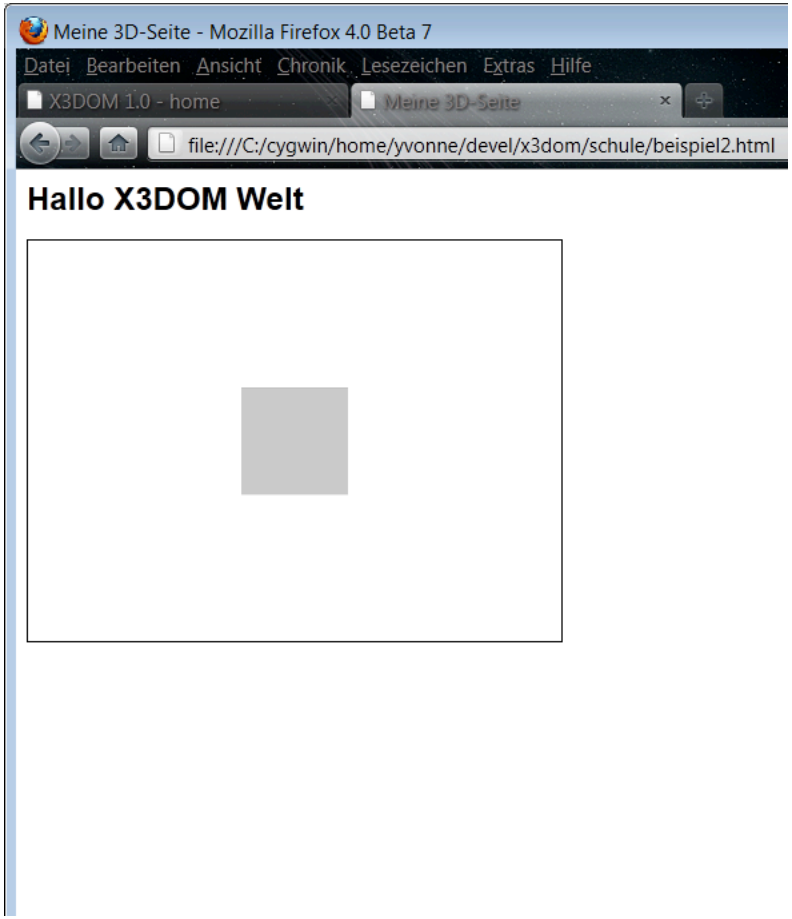
```
…
    <body>
        <h1>Hello X3DOM World</h1>
        <p>
            A blue box will soon appear.
        </p>
        <x3d width="400" height="300">
        </x3d>
    </body>
</html>
```



Fraunhofer IGD

All 3D objects are children of the <scene> element

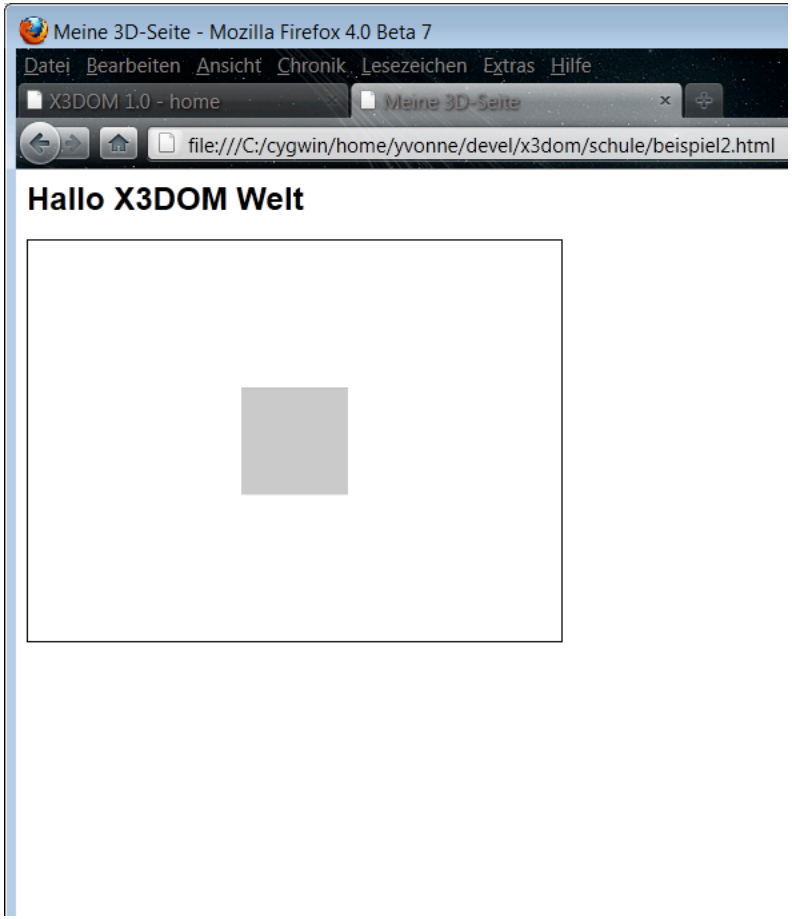
```
…
    <body>
        <h1>Hello X3DOM World</h1>
        <x3d width="400" height="300">
            <scene>
                <shape>
                    <box></box>
                </shape>
            </scene>
        </x3d>
    </body>
</html>
```



Fraunhofer IGD

Every object has a <shape>

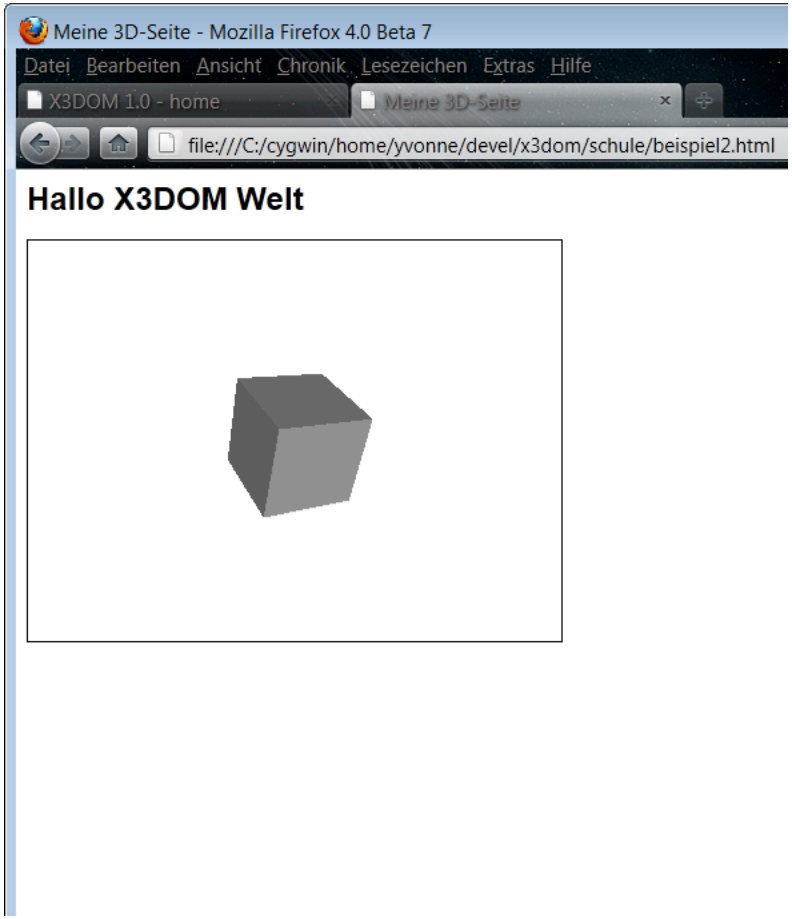
```
…
    <body>
        <h1>Hello X3DOM World</h1>
        <x3d width="400" height="300">
            <scene>
                <shape>
                    <box></box>
                </shape>
            </scene>
        </x3d>
    </body>
</html>
```



Fraunhofer IGD

…and a geometry, like e.g. a <box>

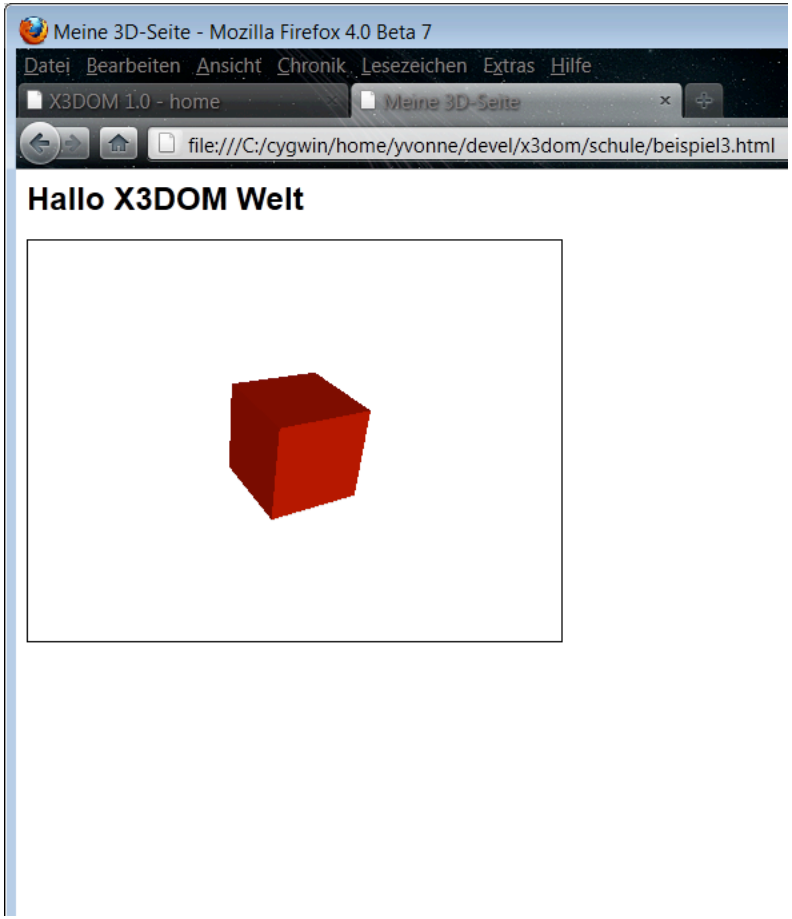
```
…
    <body>
        <h1>Hello X3DOM World</h1>
        <x3d width="400" height="300">
            <scene>
                <shape>
                    <box></box>
                </shape>
            </scene>
        </x3d>
    </body>
</html>
```



Fraunhofer IGD

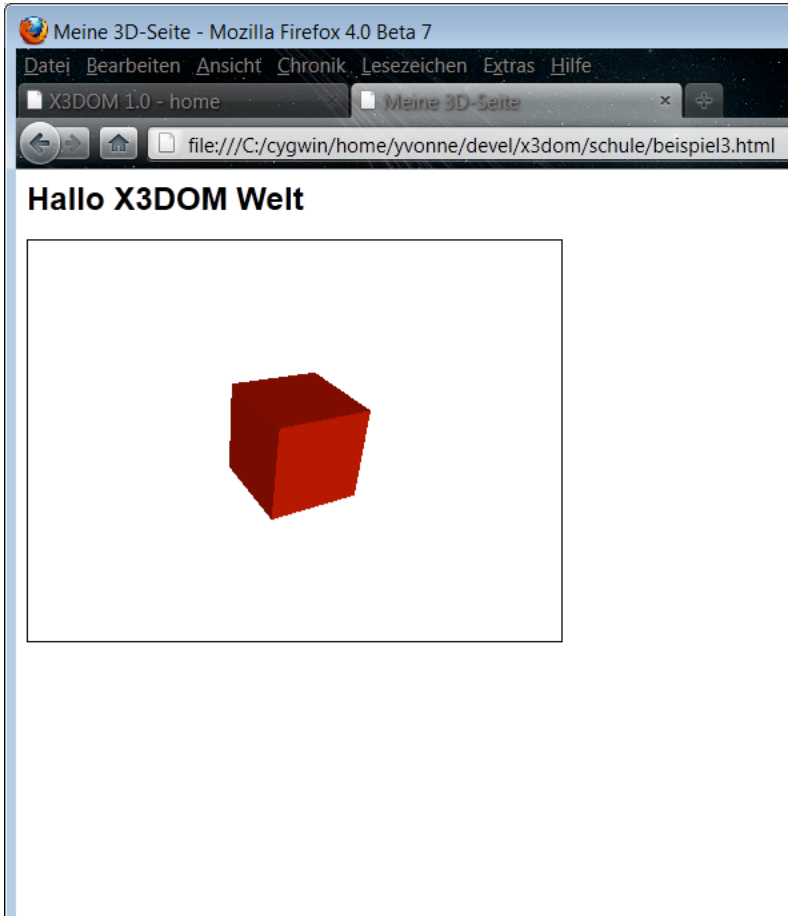
…and an <appearance>

```
<x3d width="400" height="300">
    <scene>
        <shape>
            <appearance>
                <material diffuseColor="red">
                </material>
            </appearance>
            <box></box>
        </shape>
    </scene>
</x3d>
```



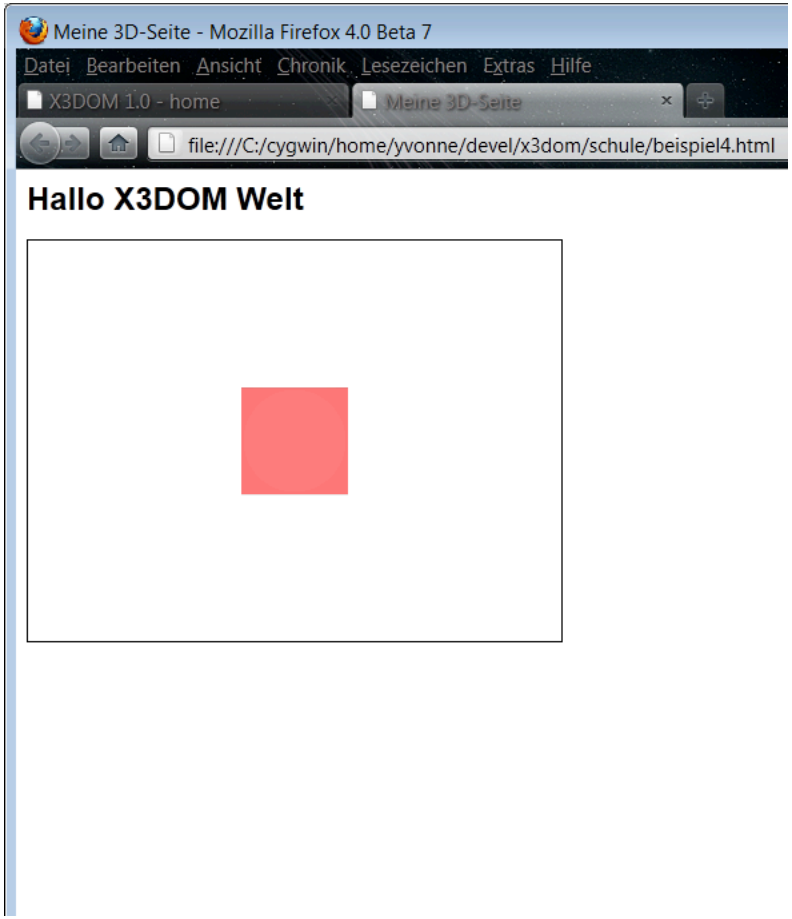
…with a (e.g. red) <material>

```
<x3d width="400" height="300">
    <scene>
        <shape>
            <appearance>
                <material diffuseColor="red">
                </material>
            </appearance>
            <box></box>
        </shape>
    </scene>
</x3d>
```



Materials with specular highlights

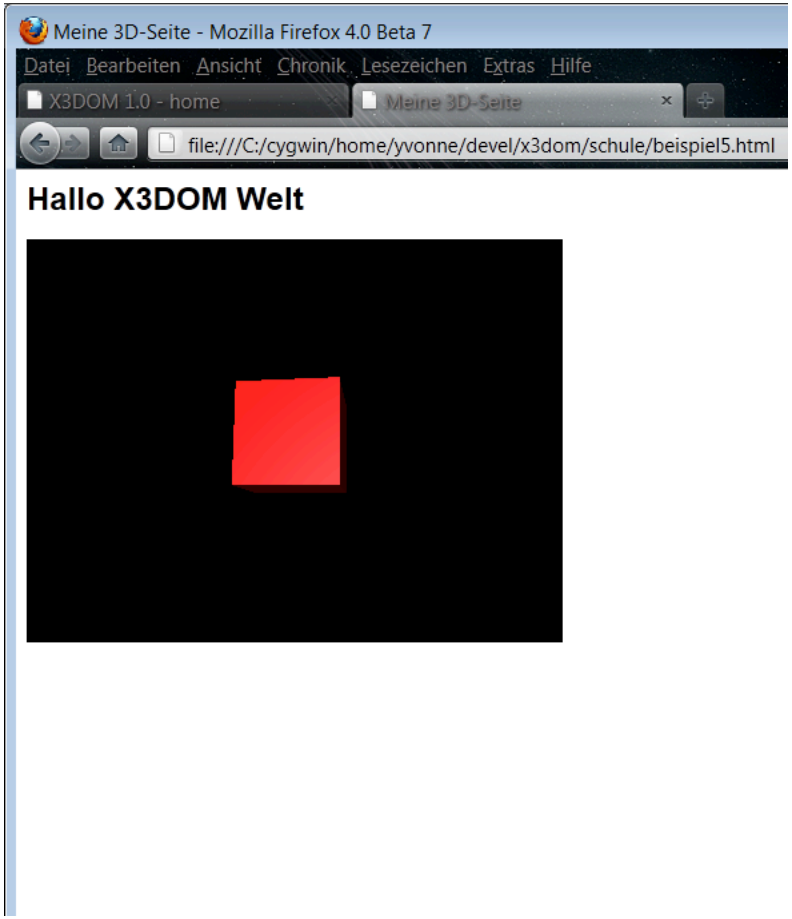
```
<x3d width="400" height="300">
    <scene>
        <shape>
            <appearance>
                <material diffuseColor="red"
                    specularColor="#808080">
                </material>
            </appearance>
            <box></box>
        </shape>
    </scene>
</x3d>
```



Fraunhofer IGD

Change Background
Colors in (R,G,B) with red/green/blue ∈ [0,1]

```
<scene>
    <shape>
        <appearance>
            <material diffuseColor="red"
                specularColor="#808080">
            </material>
        </appearance>
        <box></box>
    </shape>
    <background skyColor="0 0 0">
    </background>
</scene>
```



Fraunhofer IGD

Change Background (now using CSS)

```
<x3d style="background-color: #00F;">
    <scene>
      …
    </scene>
</x3d>
```

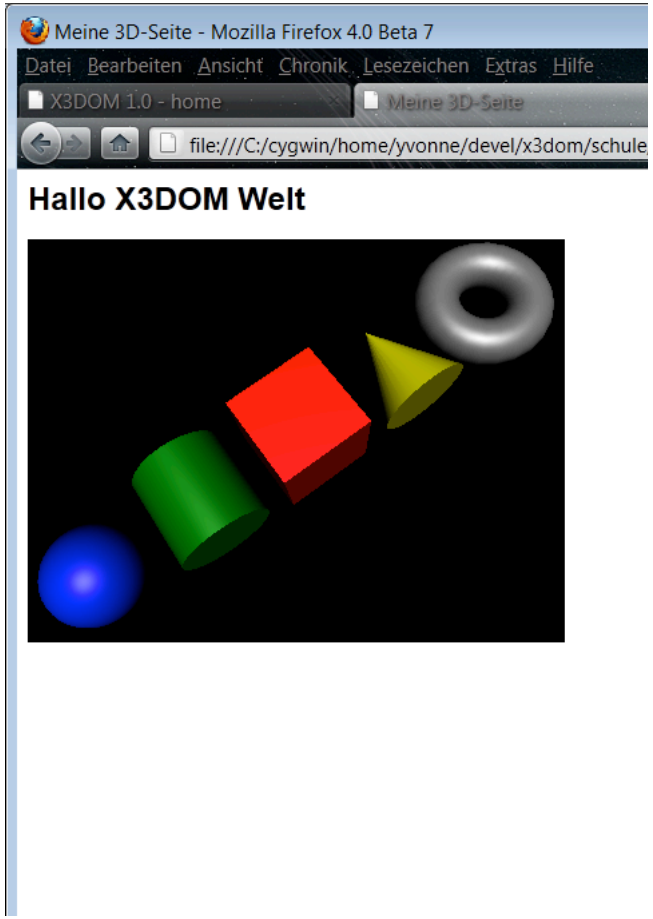
Change size of <x3d> element to full size

```
<x3d style="left:0px; top:0px; width:100%; height:100%; border:none;">
    …
</x3d>
```

Geometric base objects

See screenshot – from left to right:

- <sphere radius="1.0">
- <cylinder radius="1.0" height="2.0">
- <box size="2.0 2.0 2.0">
- <cone bottomRadius="1.0" height="2.0">
- <torus innerRadius="0.5" outerRadius="1.0">



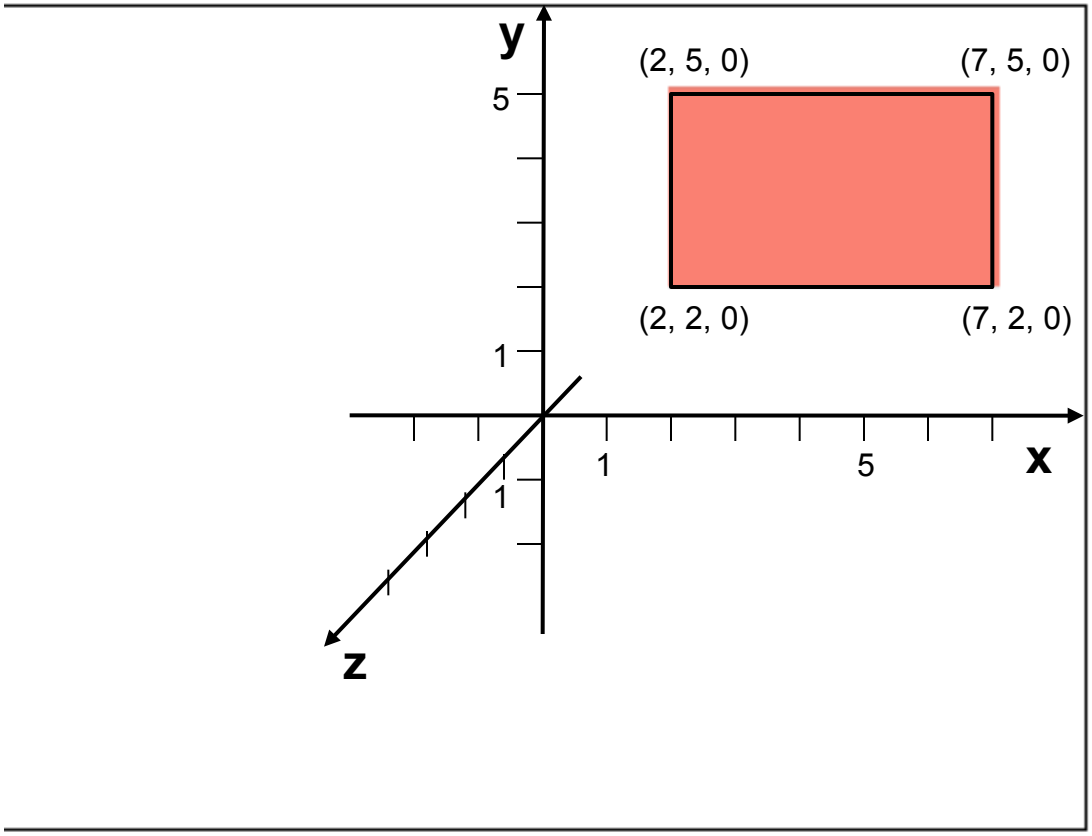
Fraunhofer IGD

Defining own geometries
Example: simple rectangle with an <indexedFaceSet>

```
<scene>
   <shape>
      <appearance>
         <material diffuseColor="salmon">
         </material>
      </appearance>
      <indexedFaceSet coordIndex="0 1 2 3 -1">
         <coordinate point="2 2 0, 7 2 0, 7 5 0, 2 5 0">
         </coordinate>
      </indexedFaceSet>
   </shape>
   <viewpoint position="0 0 15"></viewpoint>
</scene>
```

Defining own geometries
Example: simple rectangle with an <indexedFaceSet>

```
<indexedFaceSet
    coordIndex="0 1 2 3 -1">
   <coordinate point=
    "2 2 0, 7 2 0, 7 5 0, 2 5 0">
    </coordinate>
</indexedFaceSet>
```



Important building blocks

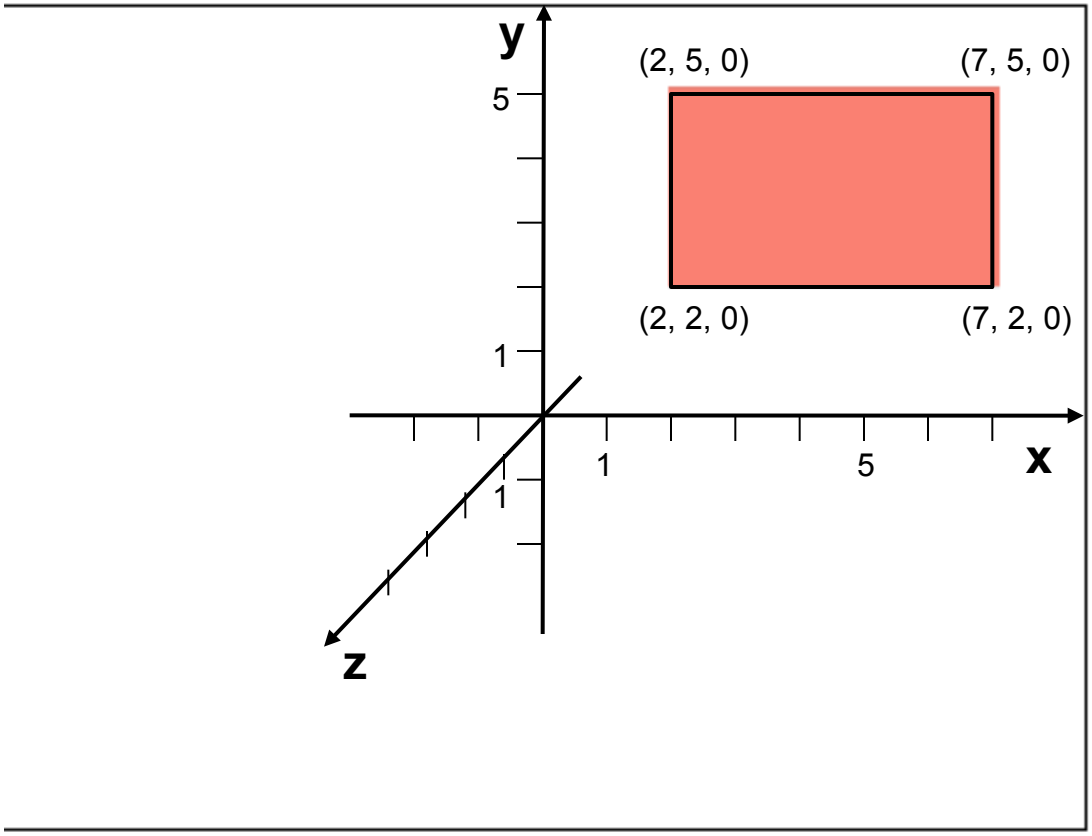
- The vertices of a Polygon (here “face”), given as <coordinate>
- The index to a vertex, given as list: “coordIndex”

Defining own geometries
Example: simple rectangle with an <indexedFaceSet>

```
<indexedFaceSet
    coordIndex="0 1 2 3 -1">
  <coordinate point=
    "2 2 0, 7 2 0, 7 5 0, 2 5 0">
    </coordinate>
</indexedFaceSet>
```

The end of one polygon and the begin of a new one is marked as “-1” in the index array

This way arbitrarily complex 3D objects can be created



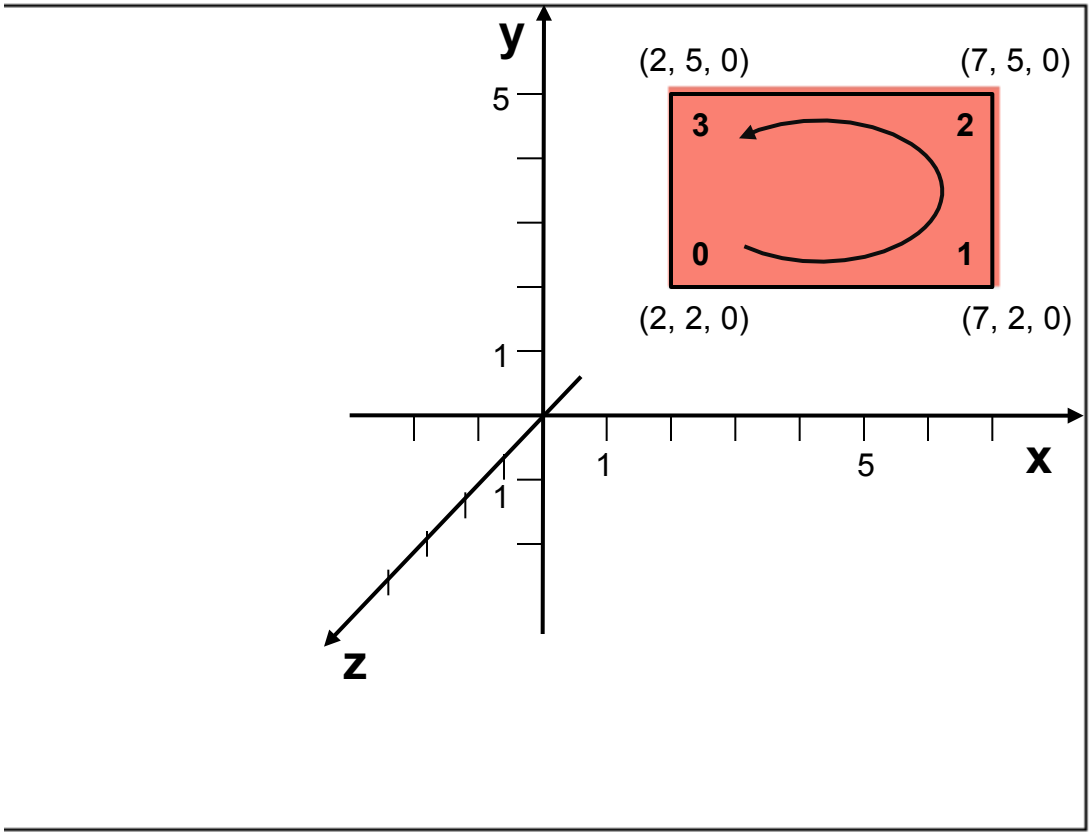
Fraunhofer IGD

Defining own geometries
Example: simple rectangle with an <indexedFaceSet>

```
<indexedFaceSet
    coordIndex="0 1 2 3 -1">
  <coordinate point=
    "2 2 0, 7 2 0, 7 5 0, 2 5 0">
    </coordinate>
</indexedFaceSet>
```

The indices (except “-1”) refer to the array position of a 3D coordinate in <coordinate>

The coordinates of a certain polygon are listed **counterclockwise**



Fraunhofer IGD

DOM holds structure and data
More than 95% are usually unstructured data

```
<!DOCTYPE html>
<html>
  <head>
    <link rel='stylesheet' type='text/css' href='http://www.x3dom.org/x3dom/release/x3dom.css'></link>
    <script type='text/javascript' src='http://www.x3dom.org/x3dom/release/x3dom.js'></script>
  </head>
  <body>
    <x3d id='3dstuff' width='400px' height='400px'>
      <scene DEF='scene'>
        <shape>
          <appearance>
            <material diffuseColor='#FF0000'></material>
          </appearance>
          <indexedTriangleSet solid='false' index='0 1 2 1 3 2 1 4 3 5 4 1 0 5 1 0 6 5 6 7 5 5 7 4 7 8 4 7 9 8 7 6 9 6 10 9 10 11
9 10 2 11 10 0 2 6 0 10 11 2 3 8 11 3 4 8 3 11 8 9'>
            <coordinate point='0.447214 0 -0.894427 0.447214 0.850651 -0.276393 1 0 -0 0.447214 0.525731 0.723607 -0.447214
0.850651 0.276393 -0.447214 0.525731 -0.723607 -0.447214 -0.525731 -0.723607 -1 0 0 -0.447214 0 0.894427 -0.447214 -0.850651
0.276393 0.447214 -0.850651 -0.276393 0.447214 -0.525731 0.723607'></coordinate>
            <normal vector='0.447214 0 -0.894427 0.447214 0.850651 -0.276393 1 0 -0 0.447214 0.525731 0.723607 -0.447214 0.850651
0.276393 -0.447214 0.525731 -0.723607 -0.447214 -0.525731 -0.723607 -1 0 0 -0.447214 0 0.894427 -0.447214 -0.850651 0.276393
0.447214 -0.850651 -0.276393 0.447214 -0.525731 0.723607'></normal>
          </indexedTriangleSet>
        </shape>
      </scene>
    </x3d>
  </body>
</html>
```

New Geometry node types

```
<binaryGeometry vertexCount='1153083' primType='"TRIANGLES"'
        position='19.811892 -57.892578 -1.699294'
        size='92.804482 159.783081 26.479685'
        coord='binGeo/BG0_interleaveBinary.bin#0+24' coordType='Int16'
        normal='binGeo/BG0_interleaveBinary.bin#8+24' normalType='Int16‘
        color='binGeo/BG0_interleaveBinary.bin#16+24‘ colorType='Int16' >
</binaryGeometry>
```

Data transcoding (example with input file „model.ply“)

Without mesh optimization

aopt -i model.ply -G binGeo/:sal -x model-bg.x3d -N model-bg.html

With mesh optimization (cleanup, patching, and binary creation)

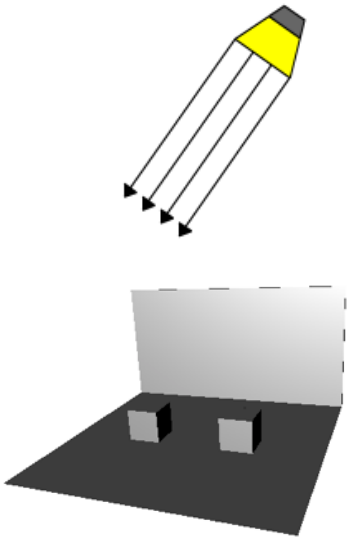
aopt -i model.ply -u -b model-clean.x3db

aopt -i model-clean.x3db -F Scene -b model-opt.x3db

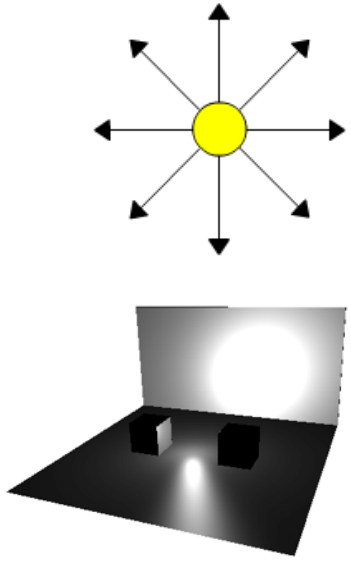
aopt -i model-opt.x3db -G binGeo/:sal -N model-bg.html

Light sources in X3DOM
…are part of the <scene>

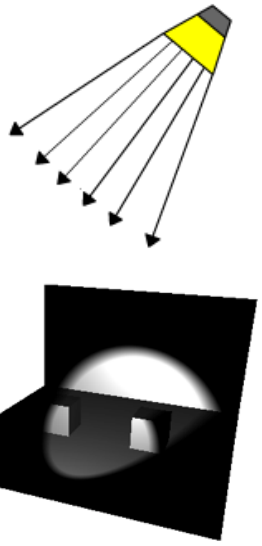
Directional light



Point light



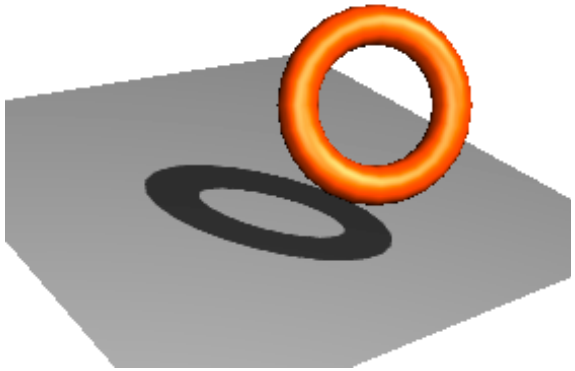
Spot light




```
<directionalLight direction='0 0 -1' intensity='1'> </directionalLight >
<pointLight location='0 0 0' intensity='1'> </pointLight >
<spotLight direction='0 0 -1' location='0 0 0' intensity='1'> </spotLight >
```

Other rendering effects

shadows



fog

textures

<directionalLight direction='0 0 -1' intensity='1' shadowIntensity='0.7'>

</directionalLight >

- Note: only implemented for the first <directionalLight> in the sene

<fog visibilityRange='1000'></fog>

<imageTexture url="myTextureMap.jpg"**></ imageTexture>**

- Note: like <material> only as child node of <appearance> possible!

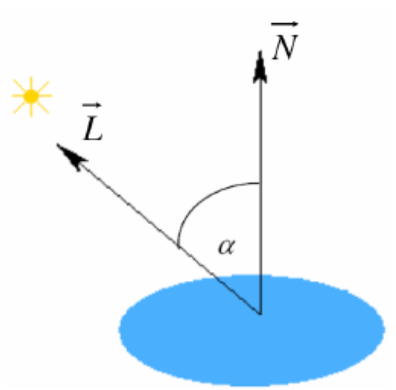
Appearance example: a textured box

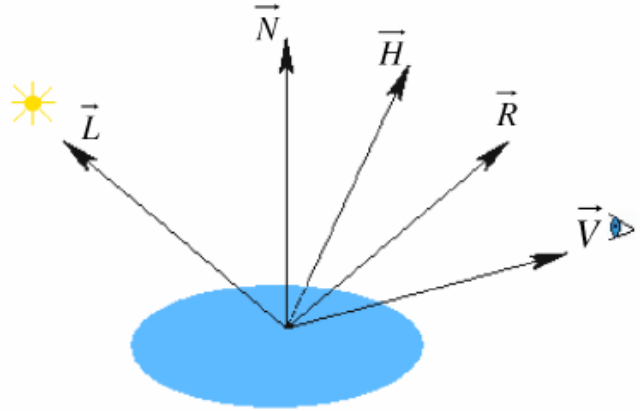
```
<x3d width="500px" height="400px">
  <scene>
    <shape>
      <appearance>
        <imageTexture url="logo.png"></imageTexture>
      </appearance>
      <box></box>
    </shape>
</scene>
</x3d>
```

Interesting alternative – using a video as texture:

<movieTexture url=”"foo.mp4",”foo.ogv”“></movieTexture>

Excursus: the lighting model (diffuse and specular reflection)





$\vec{L}$ = Lichtvektor
$\vec{N}$ = Normale
$\vec{H}$ = Halbvektor
(Winkel zw. $\vec{V}$ und $\vec{L}$)
$\vec{R}$ = Reflexionsvektor
$\vec{V}$ = Betrachtervektor (View)

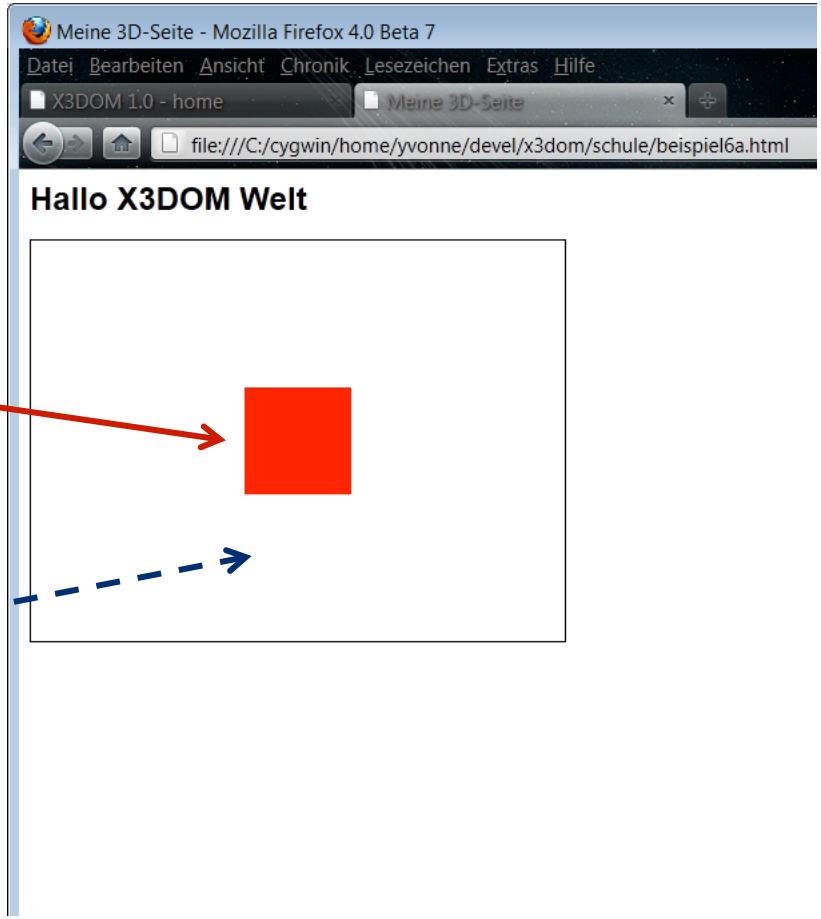
$$I_{diff} = \max(0, \vec{N} \bullet \vec{L}) = \max(0, \cos\alpha) \qquad I_{spec} = (\vec{N} \bullet \vec{H})^s = \cos^s \beta \qquad \vec{H} = (\vec{L} + \vec{V}) / |\vec{L} + \vec{V}|$$

Final color *I* := ambient material + diffuse material * *(N·L)* + specular material * *(N·H)*

For more light sources: $$I_{ges} = a_{glob} \otimes m_{amb} + m_{em} + \sum_k c_{spot}^k (I_{amb}^k + d^k (I_{diff}^k + I_{spec}^k))$$

Two objects in one scene (?!)

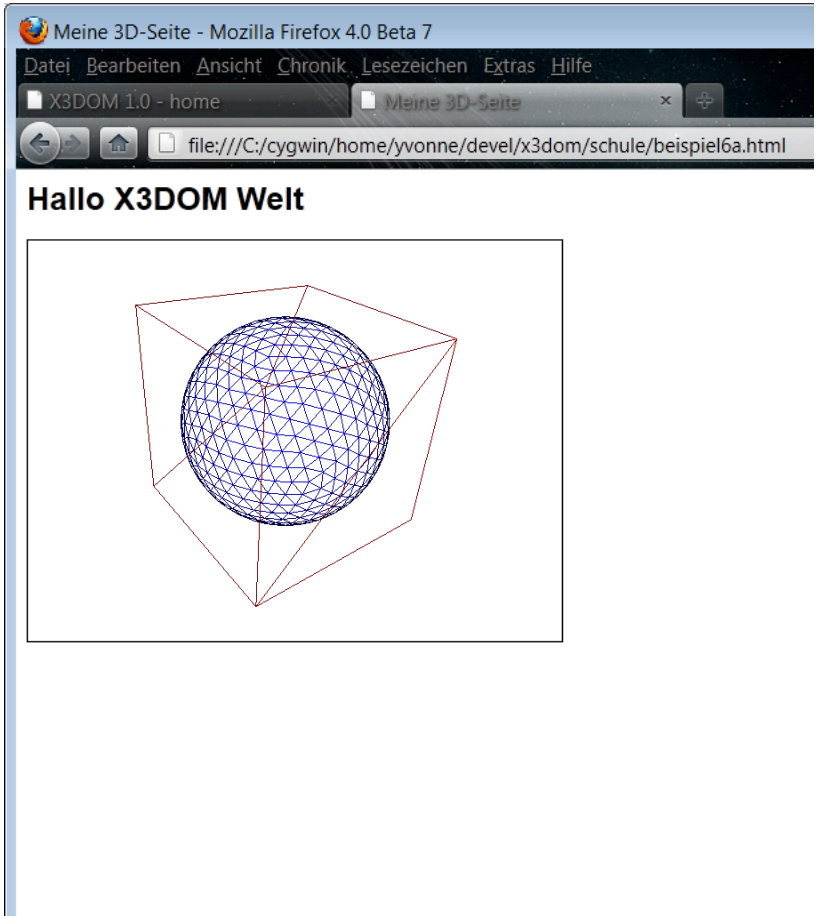
```
<scene>
    <shape>
      <appearance>
        <material diffuseColor='red'></material>
      </appearance>
      <box></box>
    </shape>
    <shape>
      <appearance>
        <material diffuseColor='blue'></material>
      </appearance>
      <sphere></sphere>
    </shape>
</scene>
…
```



Fraunhofer IGD

Two objects in one scene
Problem: both appear at same position

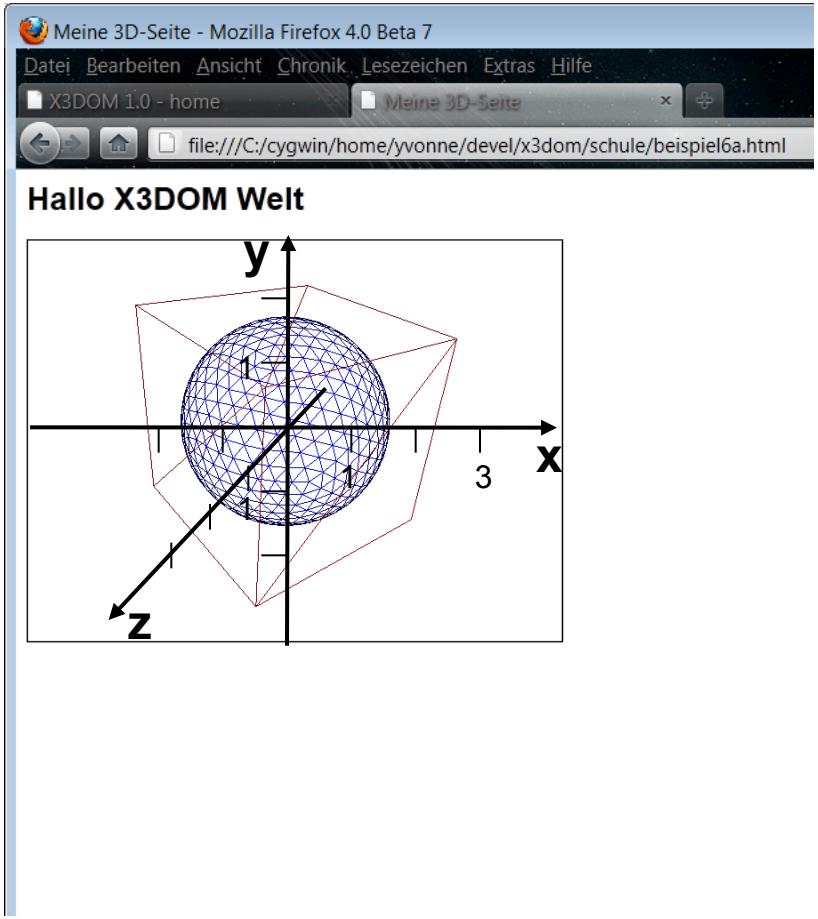
```
<scene>
    <shape>
      <appearance></appearance>
      <box></box>
    </shape>
    <shape>
      <appearance></appearance>
      <sphere></sphere>
    </shape>
</scene>
```



Fraunhofer IGD

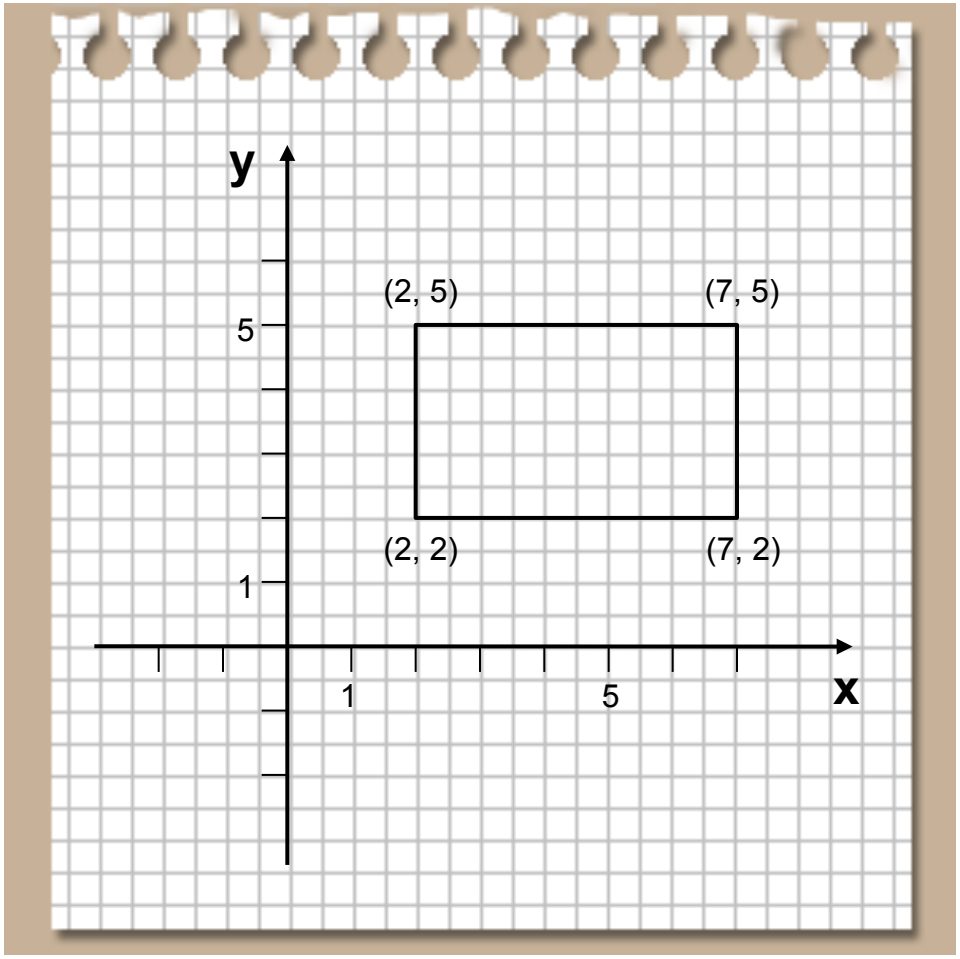
Two objects in one scene
Problem: both appear at same position

```
<scene>
    <shape>
      <appearance></appearance>
      <box></box>
    </shape>
    <shape>
      <appearance></appearance>
      <sphere></sphere>
    </shape>
</scene>
```

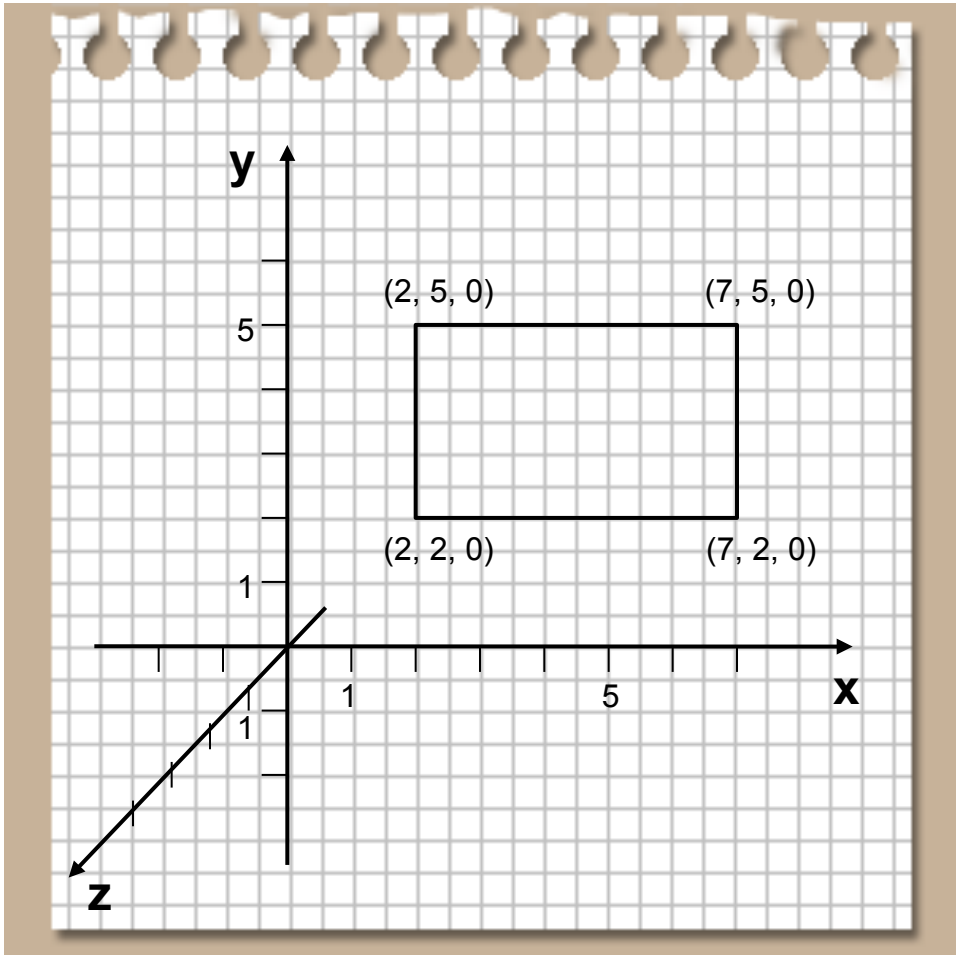


Reason: 3D objects are usually created in coordinate origin and need to be repositioned afterwards

Excursus: (2D) coordinate systems
Object coordinates in image plane (given by x & y)



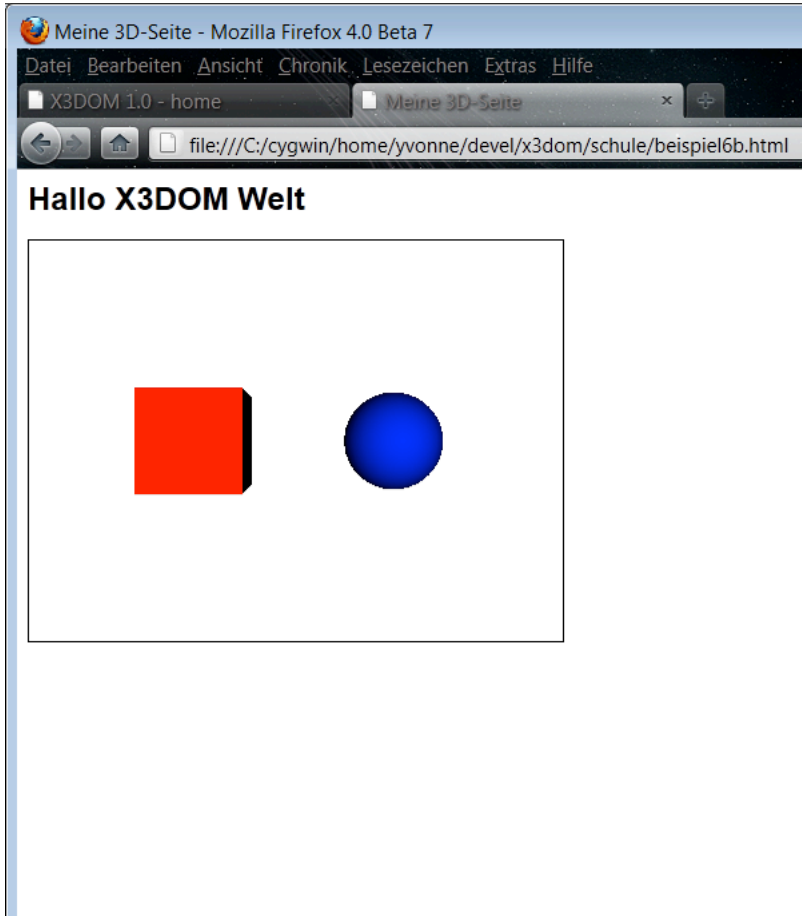
Excursus: (3D) coordinate systems
Object coordinates in 3D space (z orthogonal on x & y)



Fraunhofer IGD

Two objects in one scene
Now with translation

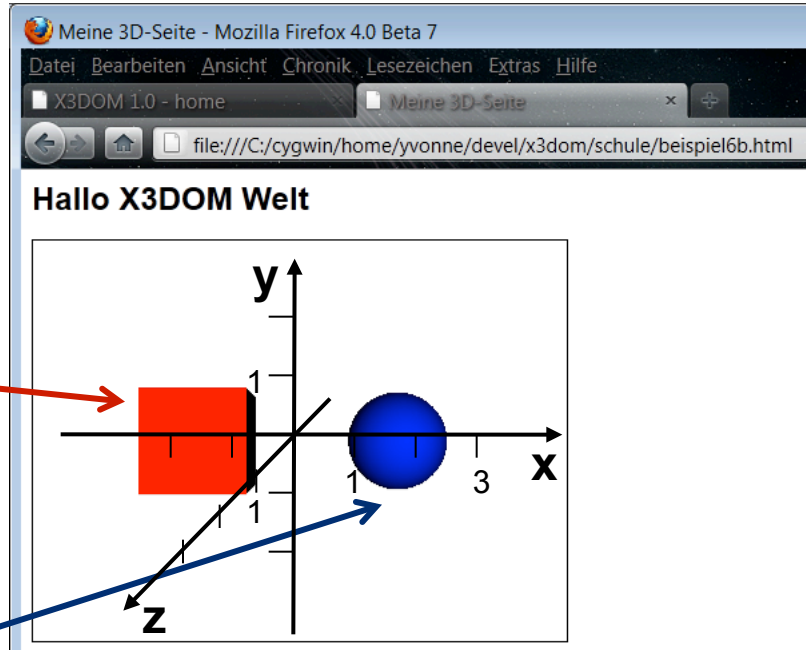
```
<transform translation="-2 0 0">
  <shape>
    <appearance>
      <material diffuseColor="red"></material>
    </appearance>
    <box></box>
  </shape>
</transform>
<transform translation="2 0 0">
  <shape>
    <appearance>
      <material diffuseColor="blue"></material>
    </appearance>
    <sphere></sphere>
  </shape>
</transform>
```



Fraunhofer IGD

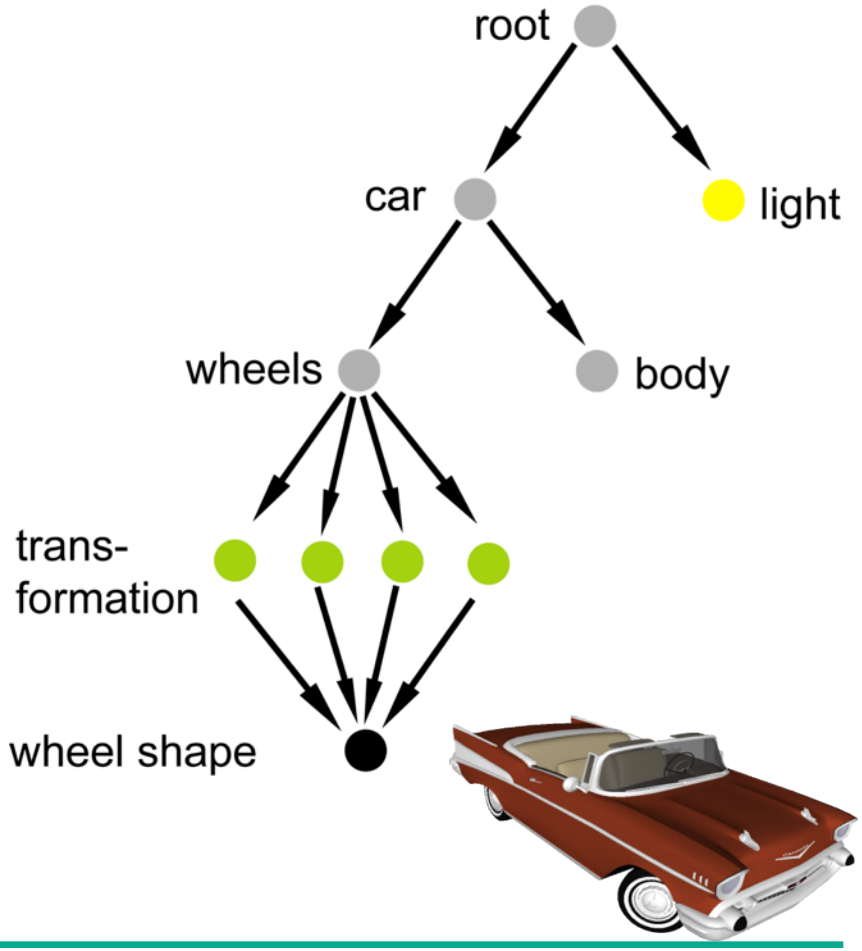
Two objects in one scene
Now with translation

```
<transform translation="-2 0 0">
  <shape>
    <appearance>
      <material diffuseColor="red"></material>
    </appearance>
    <box></box>
  </shape>
</transform>
<transform translation="2 0 0">
  <shape>
    <appearance>
      <material diffuseColor="blue"></material>
    </appearance>
    <sphere></sphere>
  </shape>
</transform>
```



The scene graph: Grouping and transformations

- 3D elements are usually organized hierarchically
- Starting from the root node (i.e. from <scene> element) all 3D elements (e.g. <shape>, <box> etc.) are inserted into the “tree” (scene graph) as child or sibling elements
  - Note: *tree ≠ graph*
- <group> and <transform> elements help to group and reposition objects:
  - <transform translation="0 0 0" rotation="0 1 0 0" scale="1 1 1">
    ...
    </transform>



Fraunhofer IGD

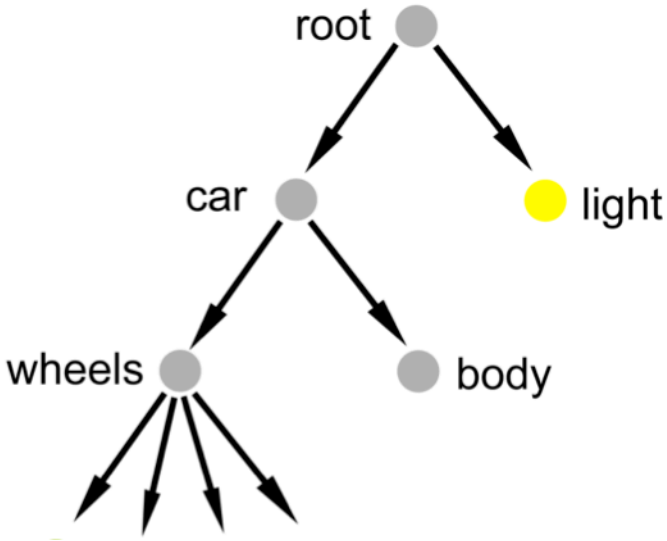
DOM Manipulation: Node appending / removal

HTML/X3D code:

```
<group id='root'></group>
…
```

JS script to add nodes:

```
root = document.getElementByid('root');
trans = document.createElement('Transform');
trans.setAttribute('translation', '1 2 3' );
root.appendChild(trans);
```



JS script to remove nodes:

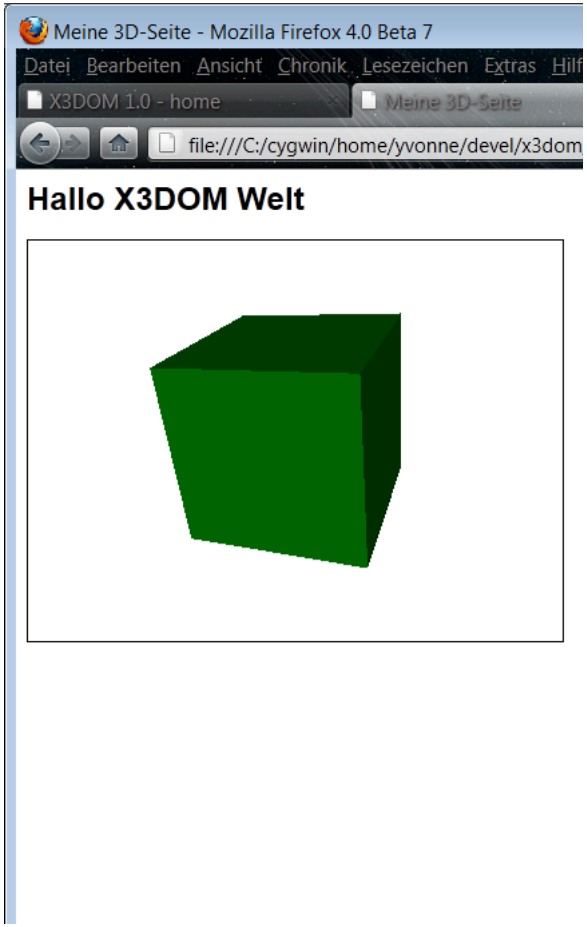
```
root.removeChild(trans);
```

JS script with setAttribute() (also useful for libs like jQuery):

```
document.getElementById('mat').setAttribute('diffuseColor','red');
```

HTML Events: user interaction through DOM Events

```
<shape>
    <appearance>
        <material id="mat" diffuseColor="red">
        </material>
    </appearance>
    <box onclick="
          document.getElementById('mat').
          setAttribute('diffuseColor', 'green');" >
    </box>
</shape>
…
```



HTML Events: user interaction through DOM Events - V2

```
<shape>
    <appearance>
        <material id="mat" diffuseColor="red"></material>
    </appearance>
    <box id="box"></box>
</shape>

<script type="text/javascript">
 document.onload = function() {
        document.getElementById('box').addEventListener('click', function() {
          document.getElementById('mat').setAttribute('diffuseColor', 'olive');
        }, false) };
</script>
```

HTML Events: 3DPickEvent extends DOM MouseEvent

```
interface 3DPickEvent : MouseEvent {
  readonly attribute float  worldX;          // 3d world coordinates at pick position
  readonly attribute float  worldY;
  readonly attribute float  worldZ;
  readonly attribute float  normalX;         // picked surface normal
  readonly attribute float  normalY;
  readonly attribute float  normalZ;
  …
}
```

```
<group onmousemove="updateTrafo(event); "> … </group>
<transform id="trafo"><shape isPickable="false"> … </shape></transform>
```

```
function updateTrafo(event) {
    var t = document.getElementById(‘trafo‘);
    var norm = new x3dom.fields.SFVec3f(event.normalX, event.normalY, event.normalZ);
    var qDir = x3dom.fields.Quaternion.rotateFromTo(new x3dom.fields.SFVec3f(0, 1, 0), norm);
    var rot = qDir.toAxisAngle();
    t.setAttribute('rotation', rot[0].x+' '+rot[0].y+' '+rot[0].z+' '+rot[1]);
    t.setAttribute('translation', event.worldX+' '+event.worldY+' '+event.worldZ);
}
```

Example 1: Interactive Car Configurator

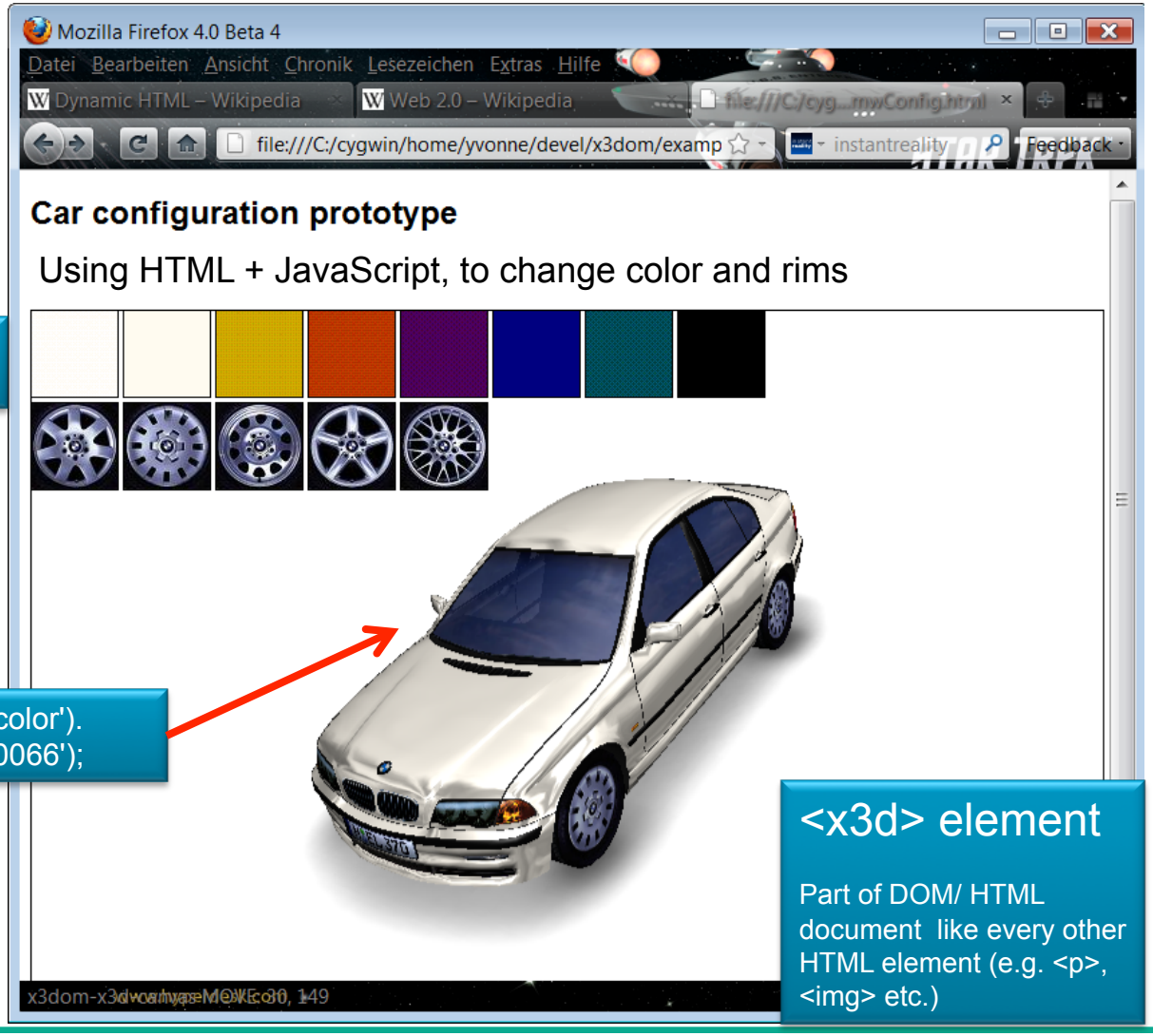
Interaction via standard Web technologies (e.g. JavaScript Events etc.)

```
<img src="felge1_64.jpg“
   onclick="..." style="…">
```

Click on <img> element…

```
document.getElementById('body_color').
   setAttribute("diffuseColor", '#000066');
```

…causes attribute change of <texture> url (i.e., other wheel rims appear)

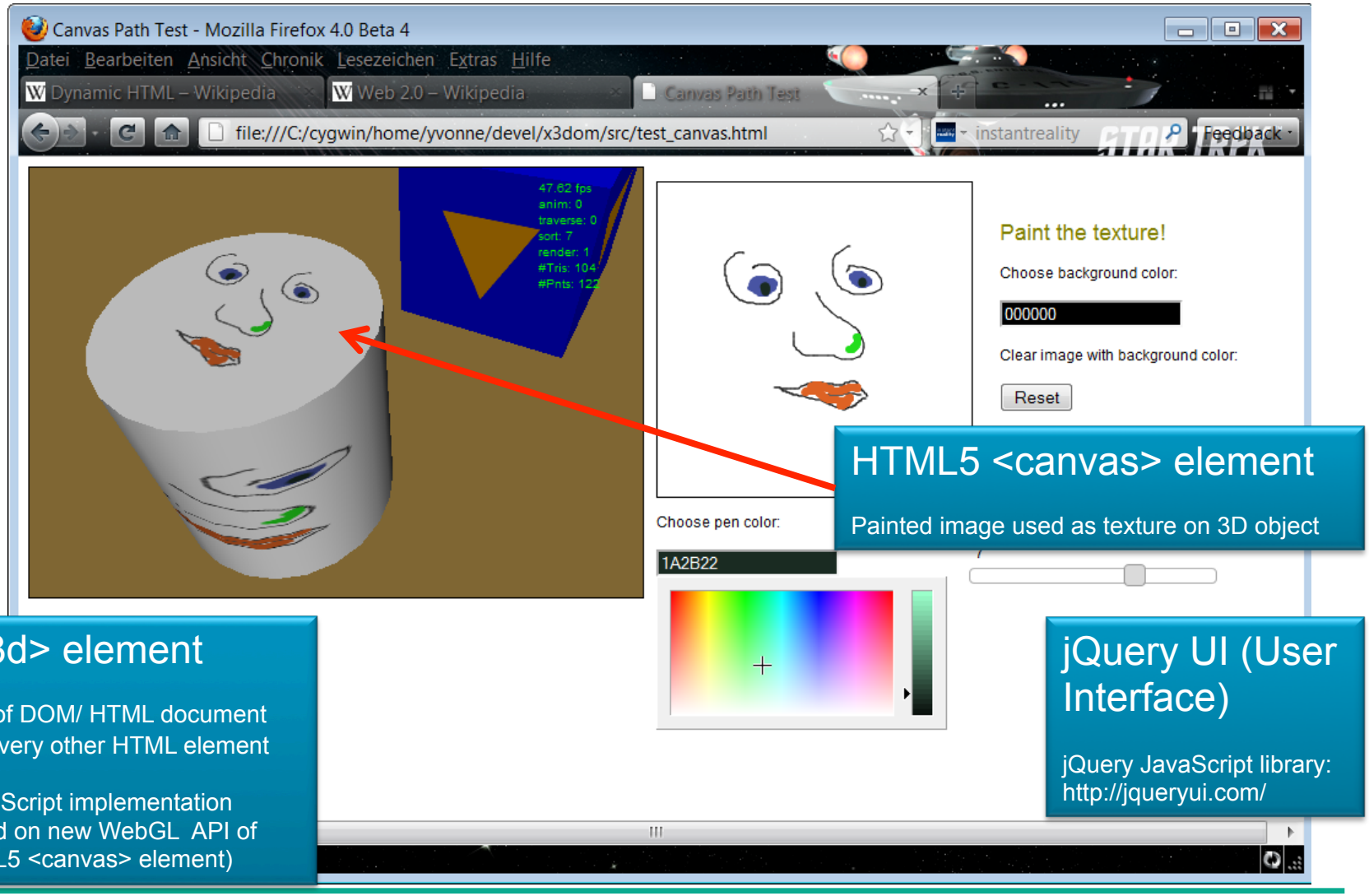


<x3d> element

Part of DOM/ HTML document like every other HTML element (e.g. <p>, <img> etc.)

Fraunhofer IGD

Example 2: Painting Textures of 3D Objects



HTML5 <canvas> element

Painted image used as texture on 3D object

<x3d> element

Part of DOM/ HTML document like every other HTML element

(JavaScript implementation based on new WebGL API of HTML5 <canvas> element)

jQuery UI (User Interface)

jQuery JavaScript library: http://jqueryui.com/

Navigation: moving the virtual camera interactively

- Built-in navigation modes
  - Examine, walk, fly, lookat, game and none
    <navigationInfo type="any"></navigationInfo>
  - Abstract behavior dynamically maps to various user inputs: mouse, keys, multi-touch

- Application-specific navigation
  - Use 'none' mode
  - Move camera by updating position and orientation of <viewpoint>

Animations
CSS 3D Transforms & CSS Animation

Utilized to transform and update <transform> nodes (only in WebKit)

```
<style type="text/css">
    #trans {
        -webkit-animation: spin 8s infinite linear;
    }
    @-webkit-keyframes spin {
        from { -webkit-transform: rotateY(0); }
        to { -webkit-transform: rotateY(-360deg); }
    }
</style>
    …
<transform id="trans">
    <transform style="-webkit-transform: rotateY(45deg);">
```

Animations
X3D TimeSensor and Interpolator nodes

```
<scene>
 <transform id="trafo" rotation="0 1 0 0">
  <shape>
   <appearance>
    <material diffuseColor="red">
    </material>
   </appearance>
   <box></box>
  </shape>
 </transform>
 <timeSensor id="ts" loop="true" cycleInterval="2">
 </timeSensor>
 <orientationInterpolator id="oi" key="0.0 0.5 1.0"
   keyValue="0 1 0 0, 0 1 0 3.14, 0 1 0 6.28">
 </orientationInterpolator>
 <ROUTE fromNode='ts' fromField='fraction_changed'
   toNode='oi' toField='set_fraction'></ROUTE>
 <ROUTE fromNode='oi' fromField='value_changed'
   toNode='trafo' toField='set_rotation'></ROUTE>
</scene>
```

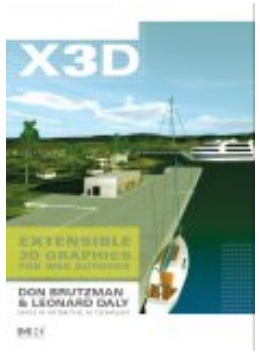
- The <timeSensor> „ts“ triggers via the first ROUTE the <orientationInterpolator> „oi“, which provides the values for the rotation around the y-axis (0,1,0)
- The resulting value is then ROUTE ‘d to the field ‘rotation’ of the <transform> node “trafo”, which results in an animation

Entry points for getting started

Some books

“X3D: Extensible 3D Graphics for Web Authors”

“The Annotated VRML 97 Reference” (explains concepts)



X3DOM online documentation and code examples

http://x3dom.org/docs/dev/ (tutorials and docs)

http://www.x3dom.org/school/ (12 simple examples)

http://www.x3dom.org/iX/ (7 examples with animation)

http://www.x3dom.org/x3dom/test/functional/ (lots of feature tests)

More docs and tools

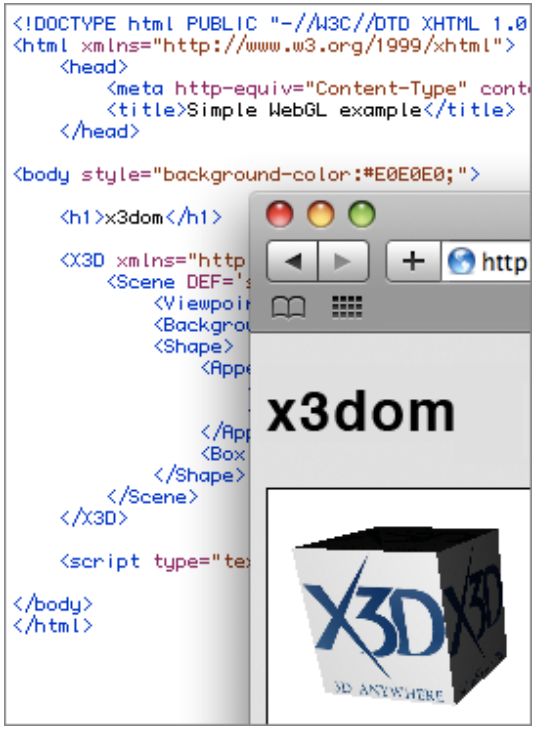
http://www.instantreality.org/downloads/ (InstantPlayer and aopt converter)

http://doc.instantreality.org/documentation/getting-started/ (links to X3D)

Declarative 3D Graphics in the Web Browser

Introduction and Tutorial

x3dom



http://www.x3dom.org/

Fraunhofer IGD