# Workshop Web3D 2011
## Advanced X3D


Yvonne Jung

Fraunhofer IGD
Fraunhoferstraße 5
64283 Darmstadt
Germany

Tel +49 6151 155 290
yvonne.jung@igd.fraunhofer.de
www.igd.fraunhofer.de/vcst

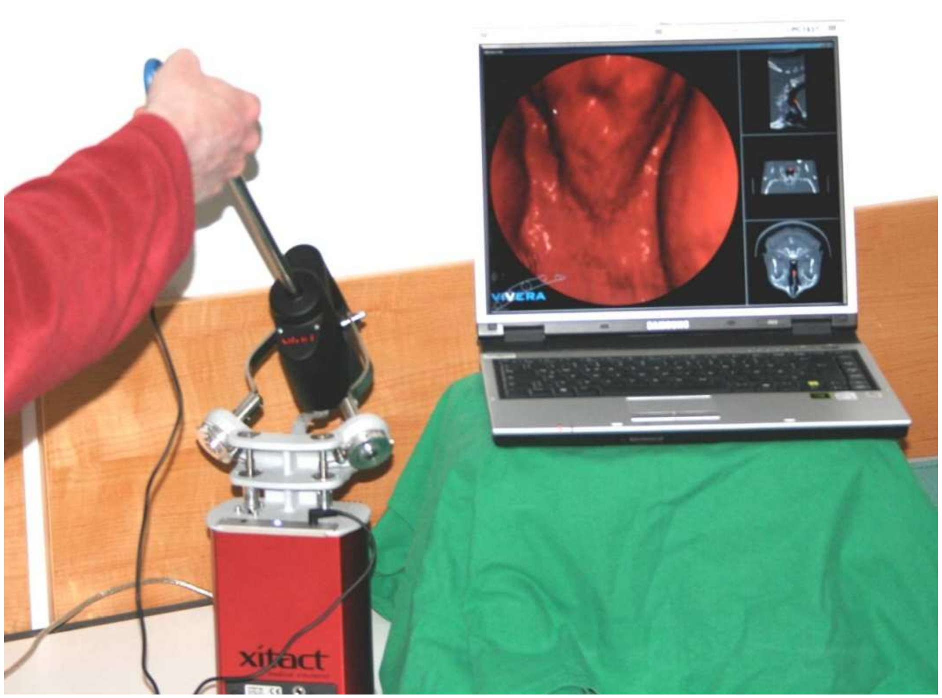

Fraunhofer IGD

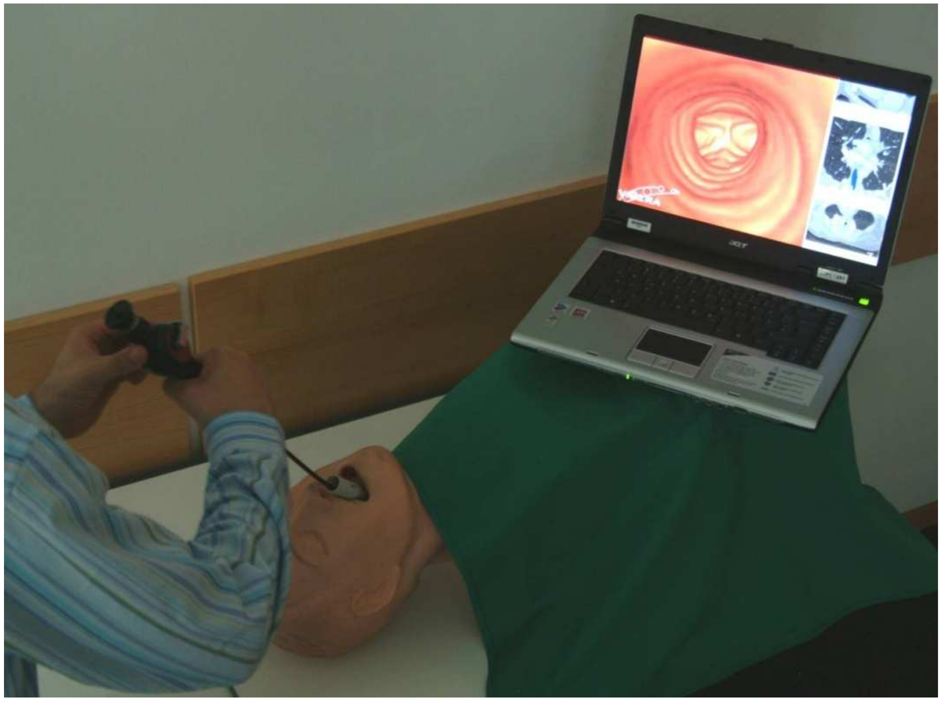
Qualitätsmanagement zertifiziert nach DIN ISO 9001:2000



# NEW X3D VERSION 3.3 NODES & INSTANT REALITY EXTENSIONS

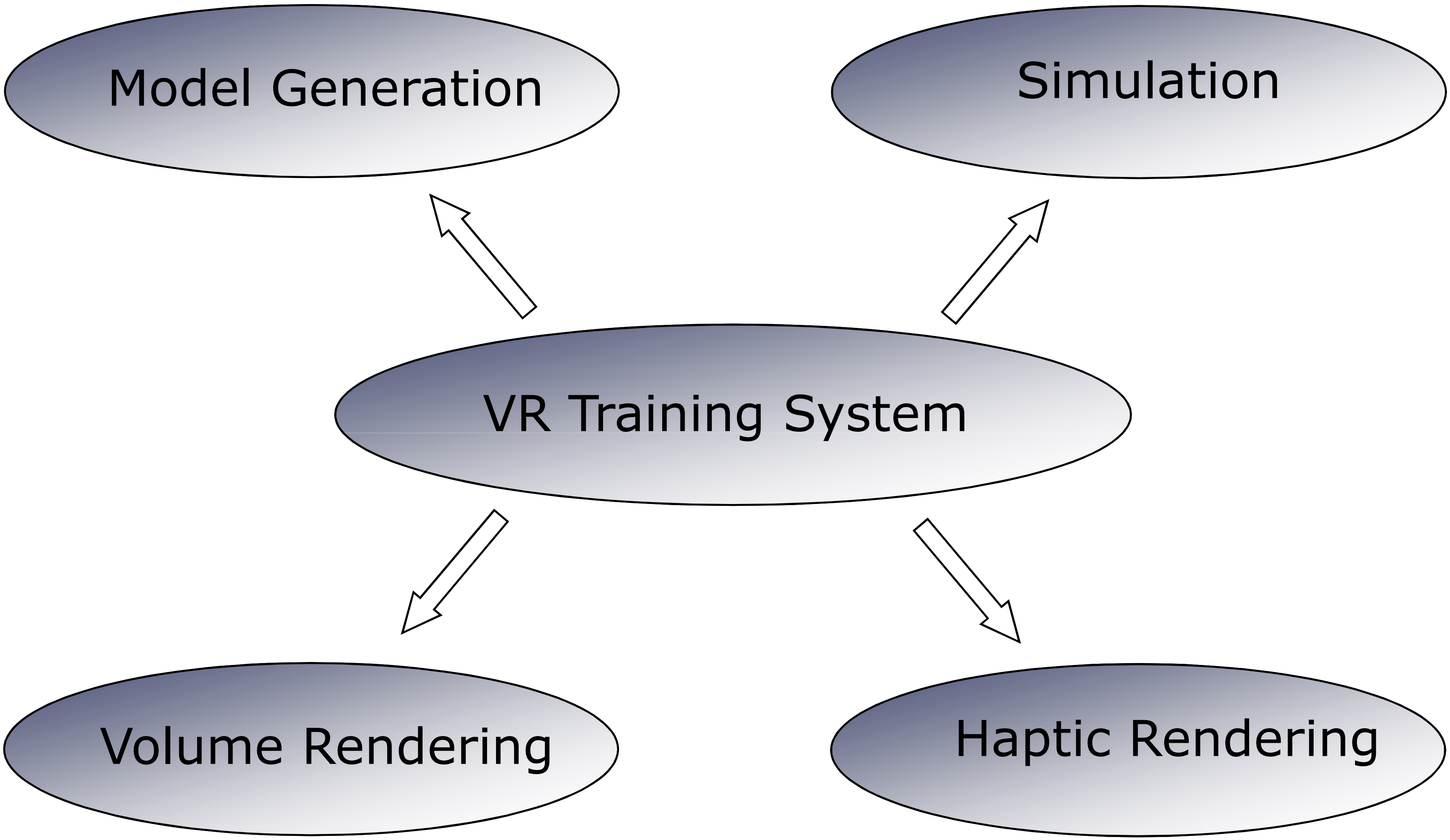
Fraunhofer IGD



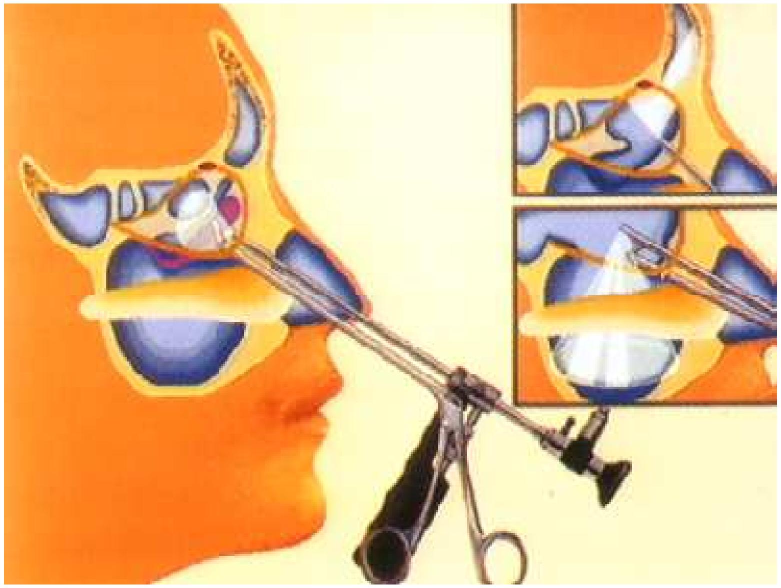



# VOLUME RENDERING COMPONENT

# Motivation: Medical Training Simulations



# Motivation: Medical Training Simulations

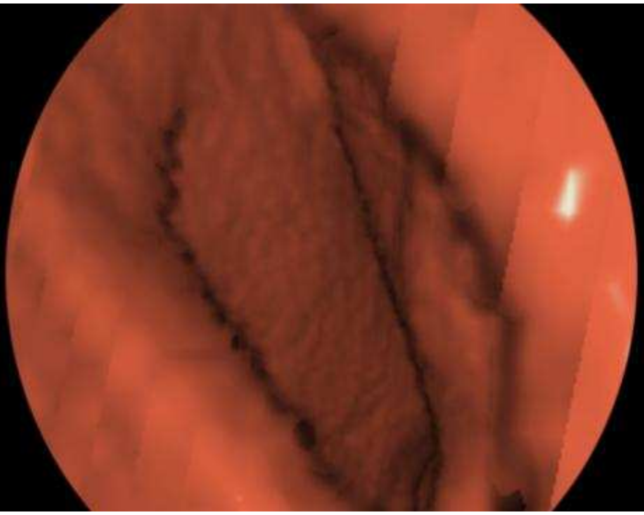
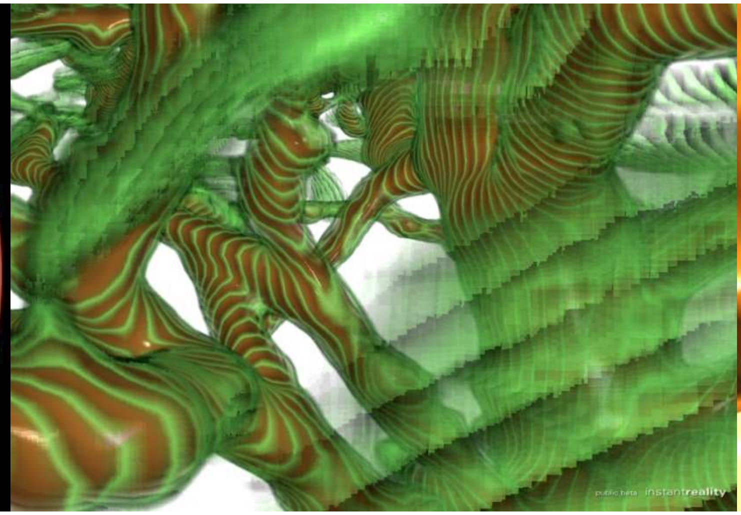
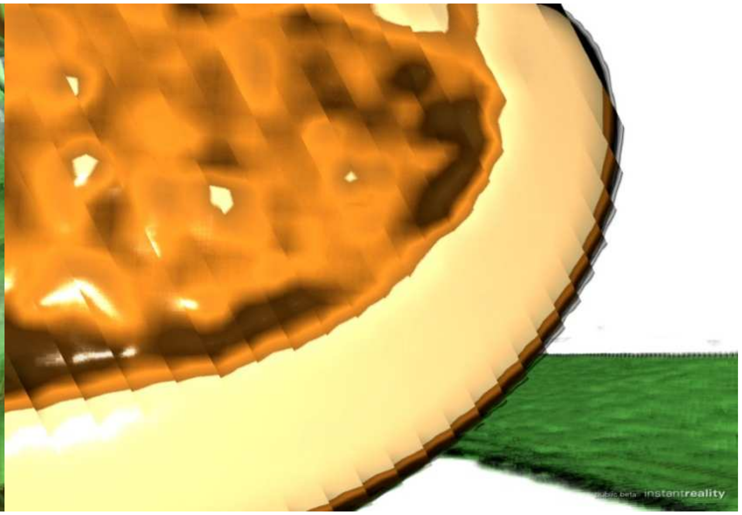


- Medical training simulations are important
    - → No training with living humans
- Training simulations are expensive, because they require knowledge of many domains
- Developers should concentrate on application and special problem cases
- Reducing complexity by using standards

↓

- ISO standard X3D already provides runtime for developing interactive applications
- X3D extension proposal for Volume Rendering Component from Medical Working Group: *http://www.web3d.org/x3d/workgroups/medical/*
- X3D-based interaction framework for device management and haptics part of InstantReality: *http://doc.instantreality.org/apidocs/instantio/*

# Volume Rendering

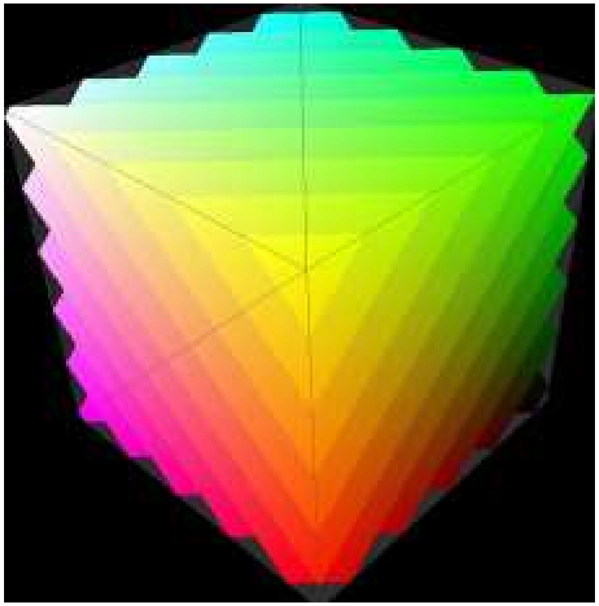




- Volume rendering is alternative form of data representation compared to traditional polygonal form
  - Volume data represents three dimensional block of space that contains some data
  - Indirect volume rendering techniques generate geometry from volume data
    - E.g. via “Marching Cubes”
  - Direct volume rendering is also able to cope with semi-transparent features

```
Shape {
    geometry DEF tri IndexedFaceSet {
        coord DEF coord Coordinate {}
    }
}
DEF iso IsoSurfaceGenerator {
    volumeUrl "Engine.nrrd"
    isoValue 0.2
}
ROUTE iso.coord_changed TO coord.set_point
ROUTE iso.index TO tri.coordIndex
```

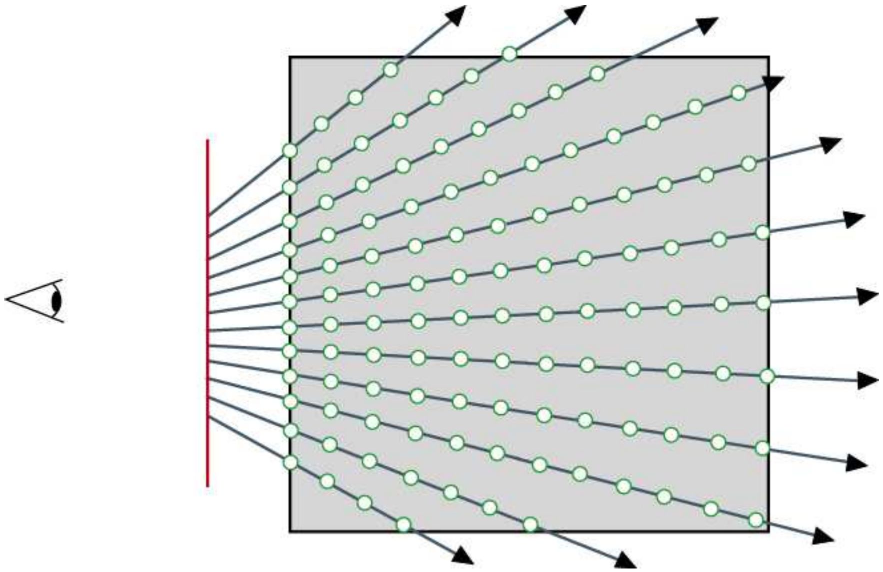
# SliceSet

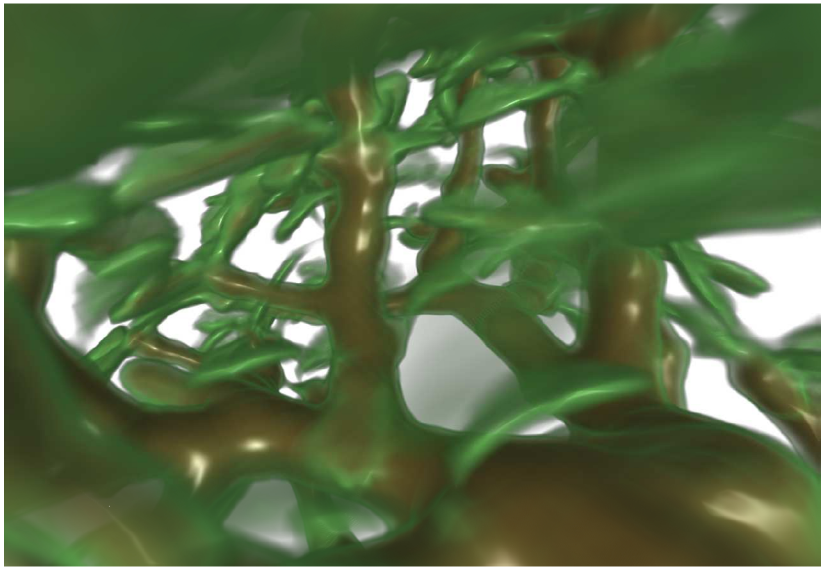
```
SliceSet : X3DViewDependentGeometryNode {
  ...
  SFFloat [in,out] sliceDistance 1
  SFVec3f [in,out] size          1 1 1
}
```



- 3D-texture-based slicing
  - Rendering of data set via viewport-aligned proxy-geometry that slices bounding box
  - Approach is easy to integrate in scene-graph and some node proposals already exist
    - …but it is inflexible and exhibits slicing as well as quantization artifacts
- The SliceSet node is a special geometry node
  - Each slice/polygon is clipped by the bounding box sides and defined by 3 to 6 vertices
  - Each vertex gets a 3D texture coordinate defining a normalized position inside the bbox

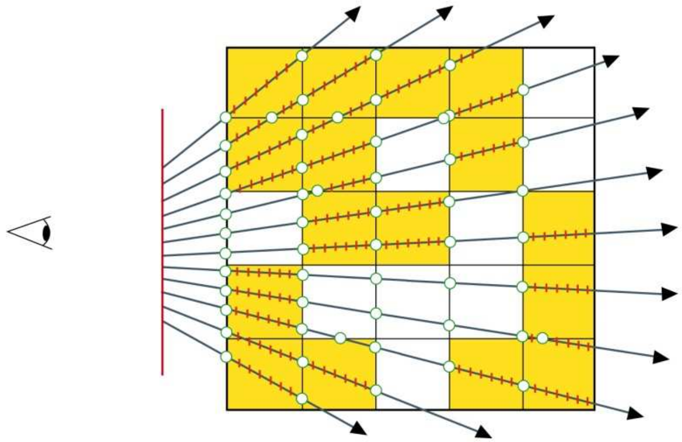
# GPU-based Singlepass-Raycasting

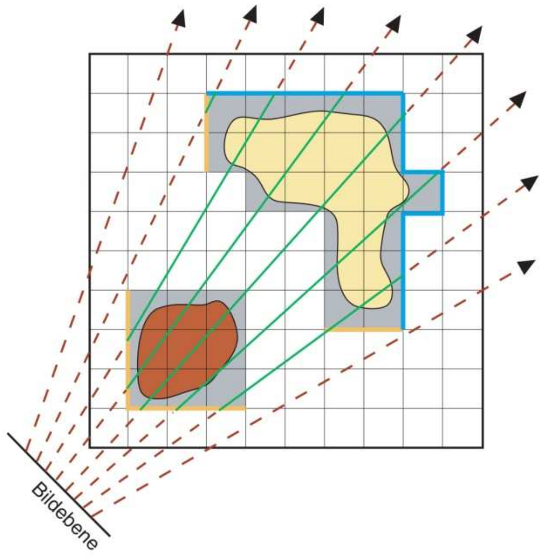


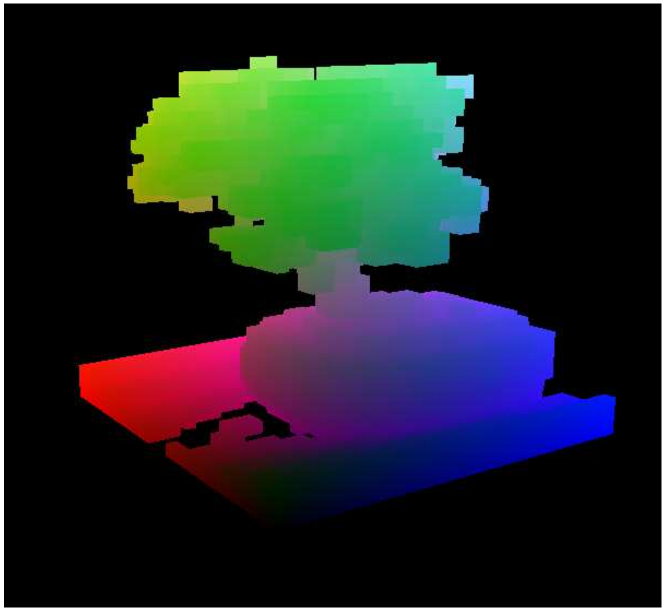


- Early GPU-based implementations used multi-pass methods
- Volume is traversed along a single ray per pixel in one rendering pass (in GLSL Shader)
  - Shader Model 3.0 or better needed
  - Special treatment for handling intersections with standard geometry required
    - Internally handled by rendering standard scene-graph first into additional texture
- Advantages: better quality (no slicing artefacts etc.), more flexibility, …
- Disadvantages: high computational costs in shader programs can lead to low performance
- ➔ Techniques for improving performance needed!

# Performance Improvements

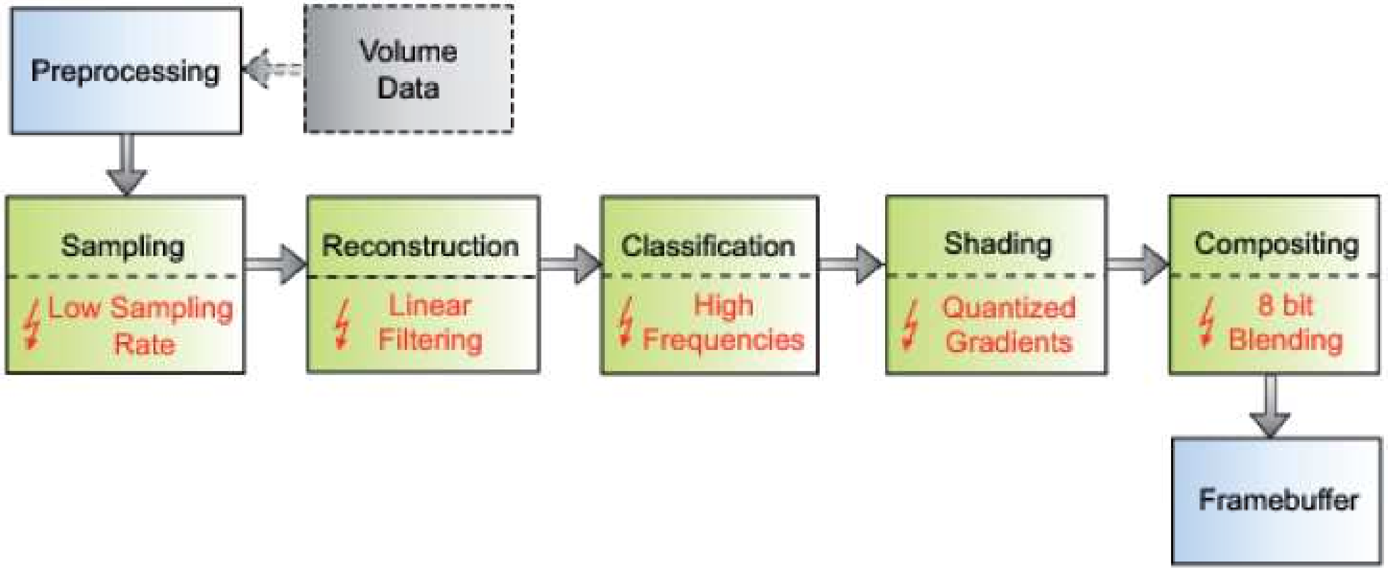


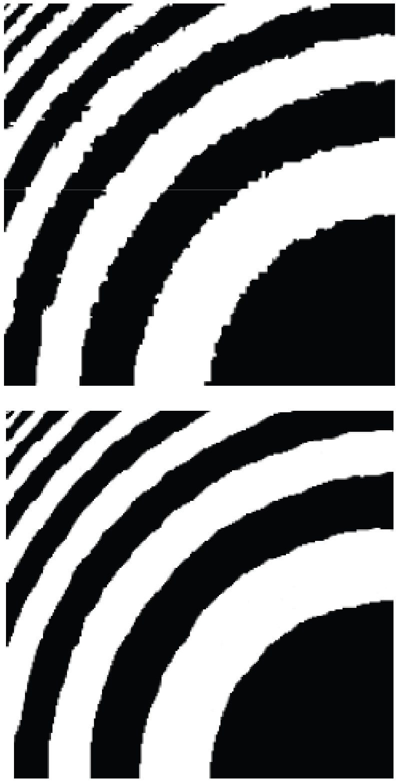


- Image-order empty-space skipping
    - Checks for empty regions inside and outside the volume during its traversal (left)
- Object-order empty-space skipping
    - Skips empty regions before and after traversal of volume (middle)
    - Approximation through bounding geometry (box or better, right image)
    - Calculation of optimal start and end position of ray and encoding in textures
        - 1st texture contains start position image
        - 2nd texture contains ray direction and length

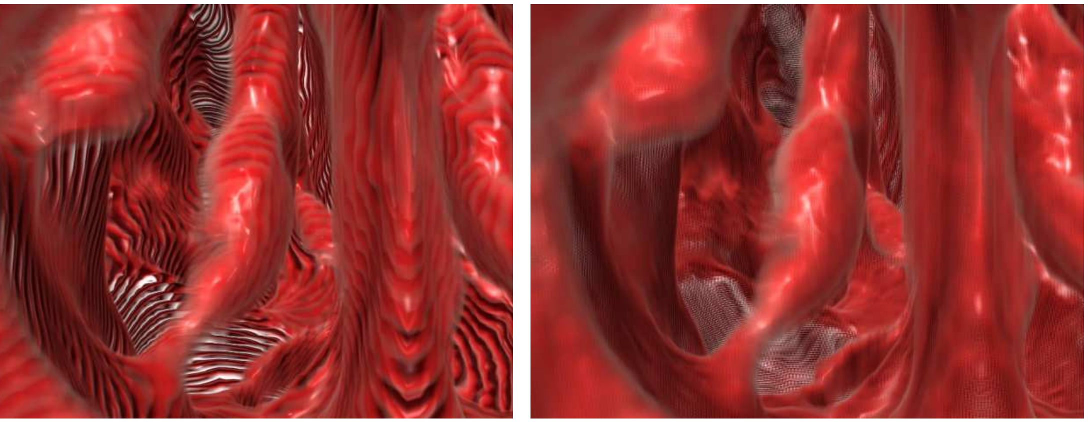
# Quality Improvements

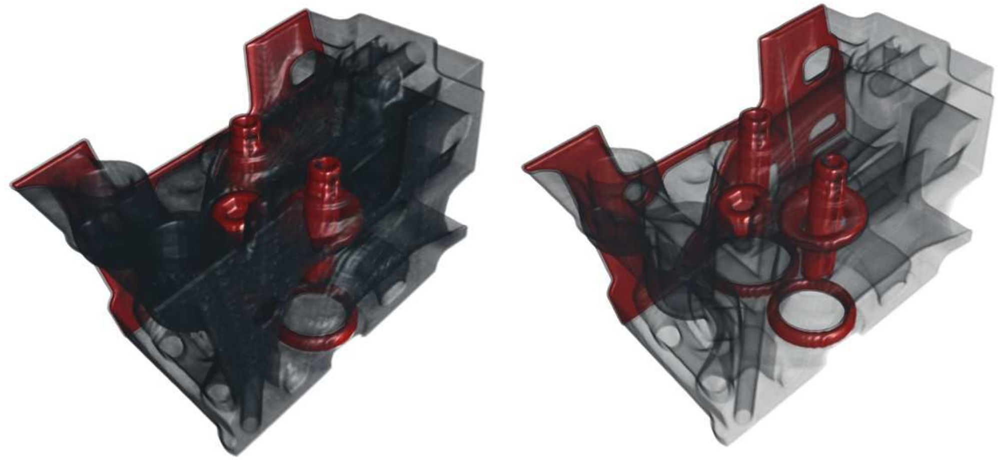


- Artifacts at different stages of rendering pipeline possible
  - Several techniques for improving image quality already exist
- Shading and lighting requires normal information ➔ computation of gradients
  - Pre-computed normals
    - Fast, but more memory and quantization artifacts
  - On-the-fly normals: e.g. central differences, Sobel filter
    - Better quality and lower memory consumption
    - → Good quality–speed trade-off

# Additional Techniques



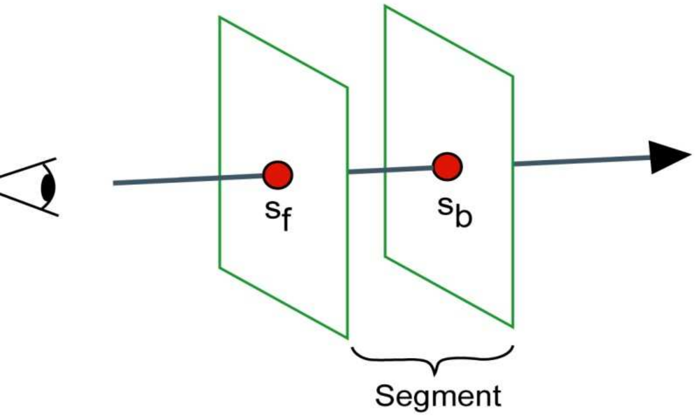


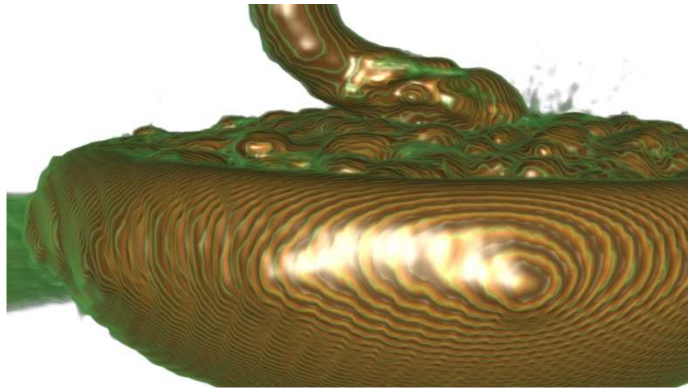
- Interleaved Sampling (left)
  - Avoids oversampling in datasets with high frequencies
  - Adding varying offset on start positions of rays
  - Interleaved pattern reduces wood grain artefacts caused by under sampling
  - Computational cheap, but at low sampling rate dithering pattern visible
- Boundary Enhancement (right)
  - Emphasizes border between different tissues via gradient magnitude modulation:

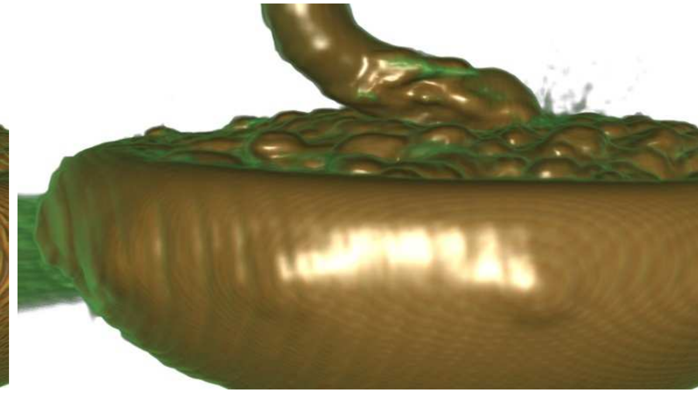
$$\alpha_{srcG} = \alpha_{src}(k_{gc} + k_{gs}(\|\nabla f(s)\|)^{k_{ge}})$$

  - Less noise in nearly homogenous regions (depending on the gradient's quality)

# Pre-Integration

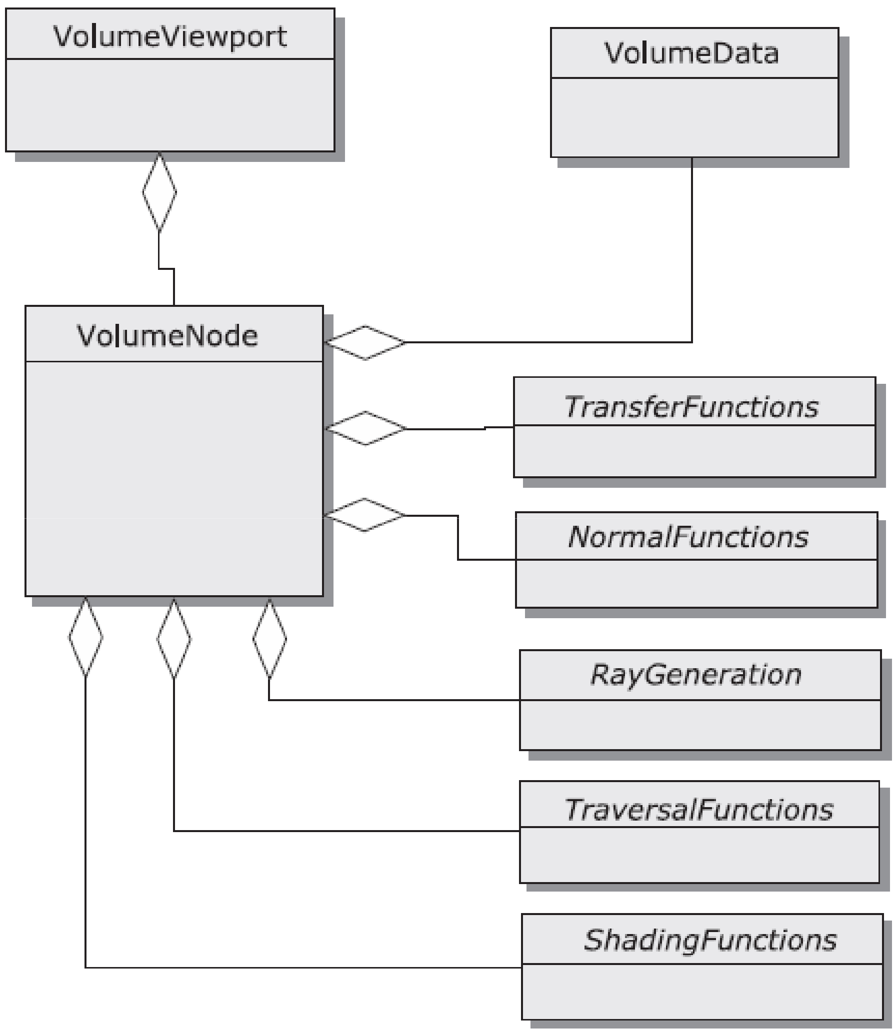




- Volume rendering integral (emission–absorption model): $I(D) = I_0 e^{-\int_{s_0}^{D} \kappa(t)\,dt} + \int_{s_0}^{D} q(s) e^{-\int_{s}^{D} \kappa(t)\,dt}\, ds$
- Besides dataset, transfer function (maps *s* to optical properties) can also contain high frequencies
  - Post-interpolative classification qualitatively better than pre-interpolative classification
- Product of high frequencies avoided by sub-dividing volume rendering integral into two integrations
  - One numerical integration for scalar field *s* and one for transfer functions *q* and *κ*
  - Pre-integration deals with second by computing pre-integration texture for sample pairs
    - Contains colour and opacity per slab (i.e. ray segment)
    - → Higher quality and better performance

# Internal Shader Framework



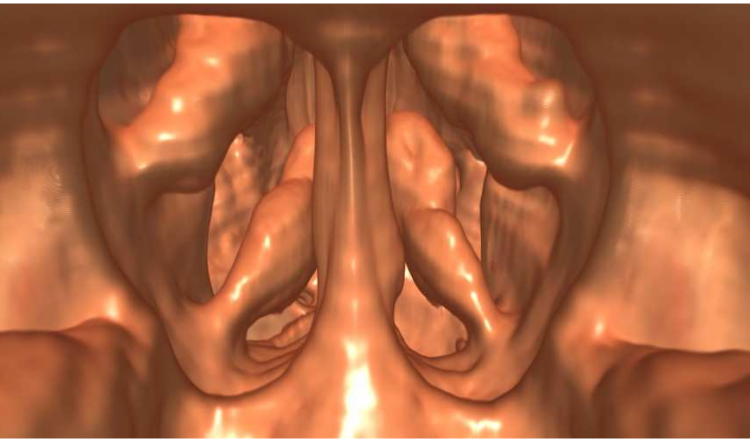

- Volume rendering framework based on OpenSG
  - Shaders implemented in GLSL
- Completely GPU-based for higher quality and better load sharing between GPU and CPU
  - CPU handles haptics and collision
- Core components
  - VolumeNode holds volume and settings
  - VolumeViewport renders volume data + scene-graph for handling intersections with standard polygonal geometry
- Modular structure
  - Different modules (e.g. RayGeneration) for different techniques (e.g. interleaved)
  - → Usage of shader compositing: each module owns special shader fragment
  - Extensible: to integrate new method only new module with shader fragment needed
  - Approach fits with new X3D 3.3 proposal

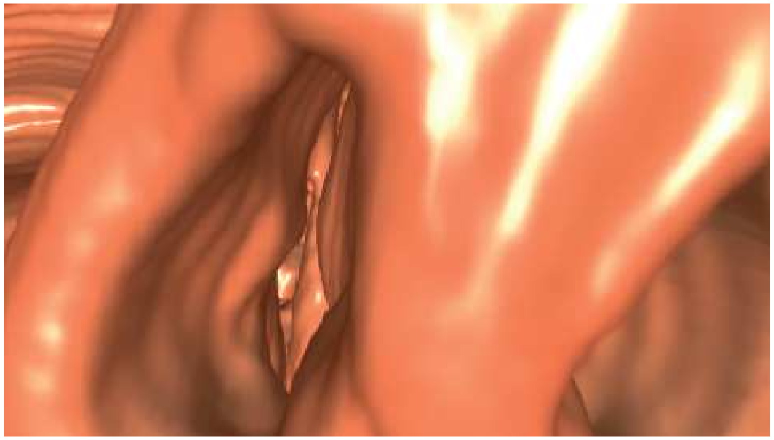
# X3D Integration via Volume Rendering Component

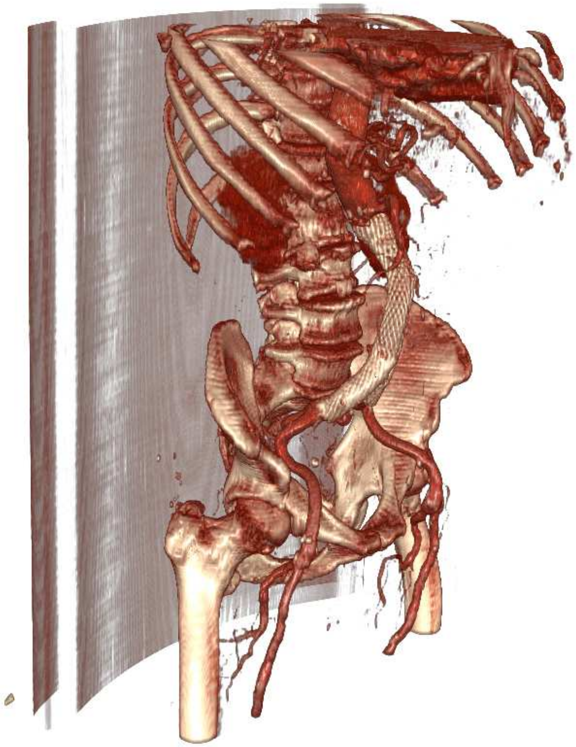
```
DEF volume VolumeData {
    dimensions 256 256 128
    voxels [
        ImageTexture {
            url "engine.nrrd"
        }
    ]
    renderStyle ComposedVolumeStyle {
        renderStyle [
            ShadedVolumeStyle {}
            OpacityMapVolumeStyle {
                transferFunction ImageTexture {
                    url "engineTransfer.png"
                }
                type "preintegrated"
            }
            BoundaryEnhancementVolumeStyle {}
        ]
    }
    renderSettings [
        AccelerationVolumeSettings {
            boundingVolumeType "boundaryGeometry"
            traversalFunction "blockSkipping
        }
        NormalVolumeSettings {
            algorithm "onlineCentralDifferences"
        }
        RayGenerationVolumeSettings {
            type "interleaved"   # "simple"
            stepSize 0.004
        }
    ]
}
```

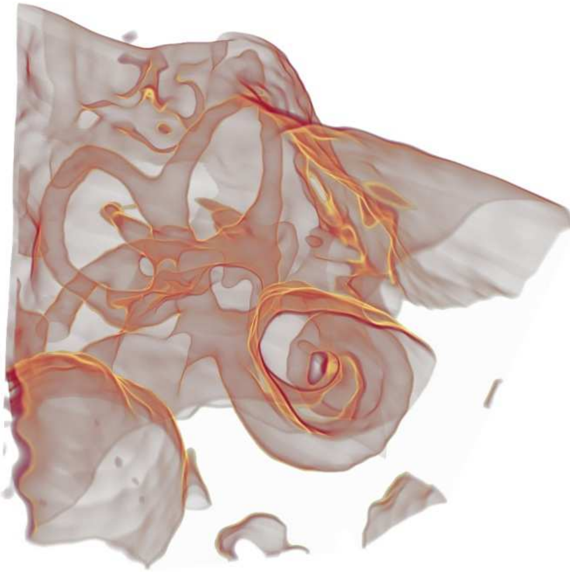
- *RenderSettings* extend X3D 3.3 Volume Rendering proposal for controlling the quality-speed trade-off
- Introducing node for managing normals

# Some Images

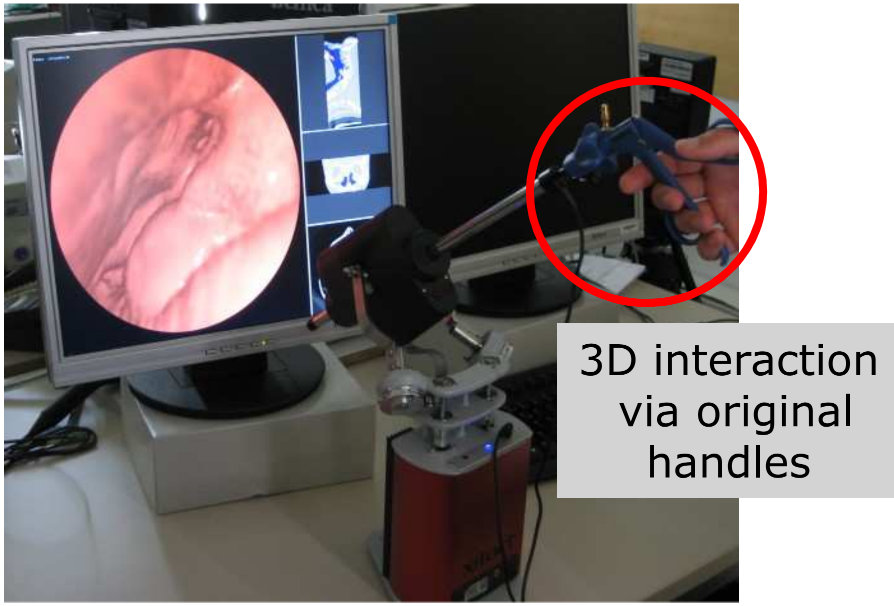


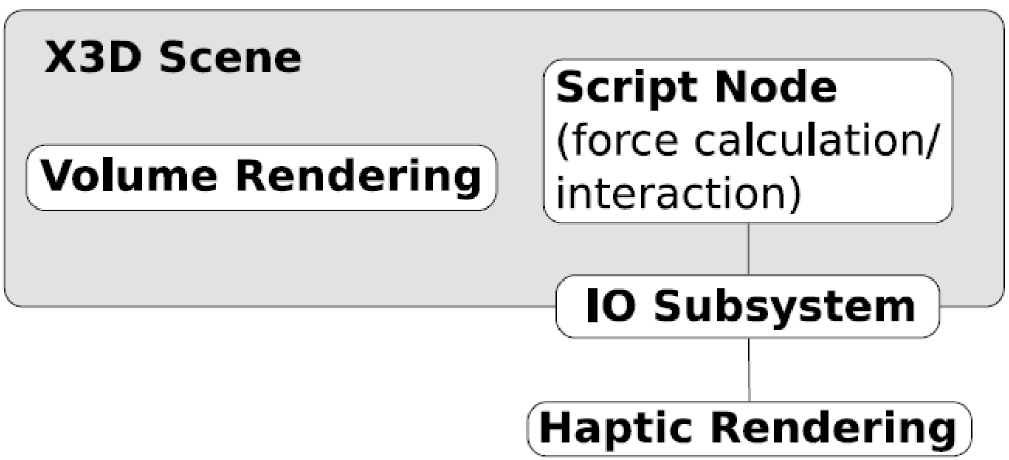






# Excursus: Haptic Rendering




Y. Jung et al. Using X3D for medical training simulations. In Proc. Web3D 2008, ACM.

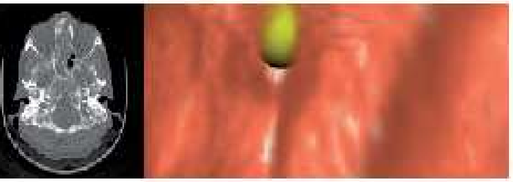
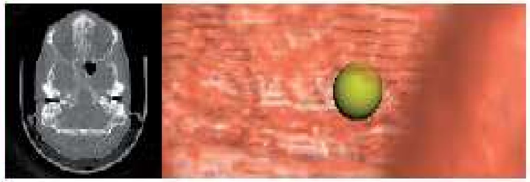
- System Design
  - X3D Volume Rendering component
  - X3D Script node for interaction logic
  - InstantReality extension “InstantIO”
    - Connecting haptic devices
- Interaction modes
  - Camera and tool navigation, milling
- Application logic
  - Implemented in Java embedded in Script
  - Camera + tool position for visualization
  - Tool position + state for manipulation
  - Responsible for haptics calculations
- Haptics requires higher frame-rate of 1 kHz
  - Calculations in extra Java thread
  - → Connected directly to IO subsystem
  - Direct volume haptics for unified approach

Fraunhofer IGD

# MULTI-PASS NODE EXTENSIONS

# Tissue Manipulation: Milling

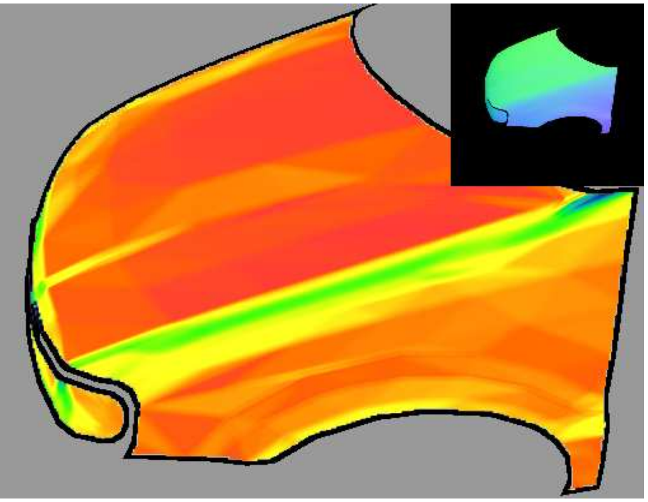



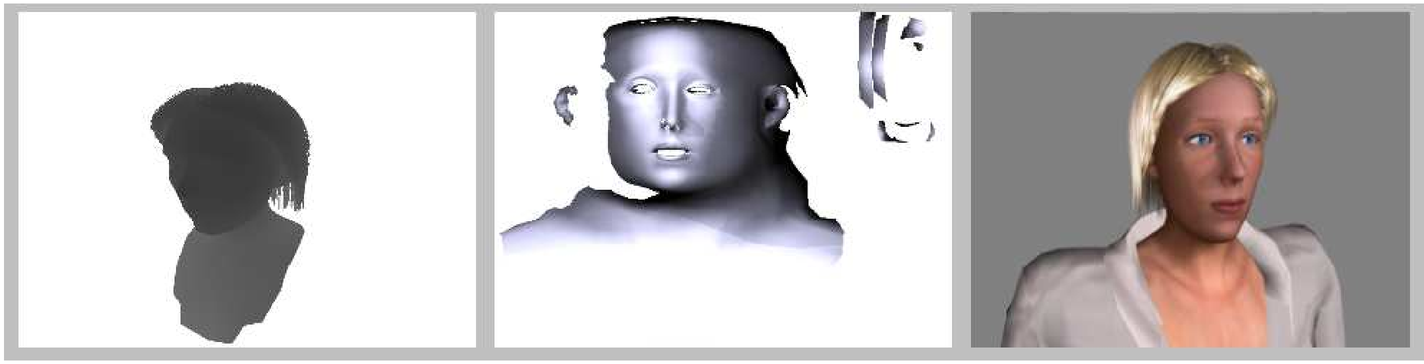
```
RenderedTexture : X3DEnvironmentTextureNode {
 SFNode      []         textureProperties NULL
 MFNode      []         excludeNodes []
 SFString    [in,out]  update        "NONE"
 SFNode      [in,out]  viewpoint     NULL
 SFNode      [in,out]  background    NULL
 SFNode      [in,out]  fog           NULL
 SFNode      [in,out]  scene         NULL
 SFNode      [in,out]  foreground    NULL
 MFInt32     [in,out]  zOffset       []
 MFNode      [in,out]  targets       []
 MFInt32     [in,out]  dimensions    [128 128 4]
 MFBool      [in,out]  depthMap      []
 SFBool      [in,out]  readBuffer    FALSE
 SFMatrix4f  [out]     projection    identity
 SFMatrix4f  [out]     viewing       identity
}
```

- Cutting requires persistent changes of volume
  - Texture updates on GPU for rendering
  - On CPU for collision detection/ response
- Transferring modified 3D texture to GPU every frame to slow → split up into 2 separate updates
  - CPU-part easy (simple array operation)
  - GPU-part requires multi-pass rendering and fine grained render state control
- Extending “RenderedTexture” to 3rd dimension
- Algorithm (called after moving cutter)
  1. Pass 1: clear 2D texture (A)
  2. Render mask to (x,y) and incr. Stencil
  3. Render volume slice with zOffset := z
  4. Pass 2: targets field refers to volume
  5. Render quad textured with A at zOffset
- On-the-fly gradient computation essential for correct shading

Fraunhofer IGD

# Multi pass rendering (1)





- Term "multi-pass" is twofold, it means both, the ability to…
  - Dynamically render a partial scene graph to an offscreen texture
  - Render in an ordered sequence with different drawing operations
- *RenderedTexture* can be seen as FBO/ PBuffer abstraction
  - First proposed in *http://www.xj3d.org/extensions/render_texture.html*
  - Floating point textures can be forced → higher precision + HDR rendering

# Multi pass rendering (2)

- *RenderedTexture* node
  - Image space rendering operations (e.g. rendering to texture space or NPR rendering)
  - Accessing e.g. neighboring information in shader programs
  - Field "depthMap“ allows generation of depth maps
    → only useful in combination with appropriate transforms
  - projection (modelview projection matrix of camera space)
  - viewing (model matrix of parent)
- *TextureGrabOverlay* node
  - Child of new Foreground bindable
    → Useful for special effects
  - Contains grabbed frame buffer (depending on its ordering)
  - Field “texture” can be re-USE-d

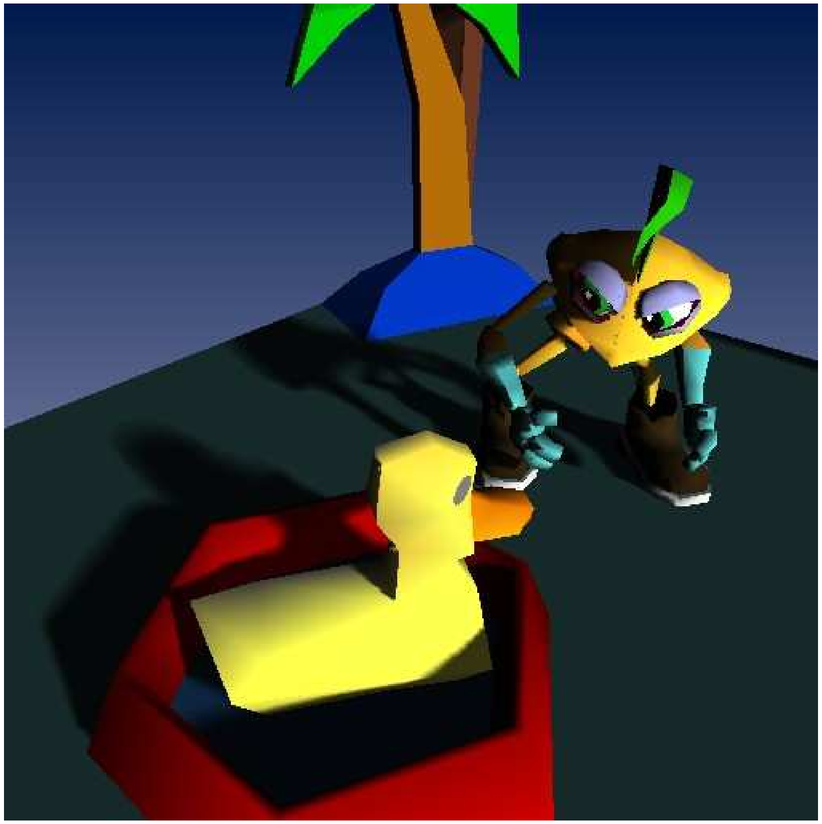
```
TextureGrabOverlay  : X3DOverlayNode {
  SFBool [in,out] enabled   TRUE
  SFNode [in,out] texture   NULL
}
```

- X3D 3.2: Layering/ layout for interaction and screen-space-text
  - Only sub-trees, rendering order and 2d-positions are defined (→ HUD)
  - Need for screen space compositing effects (e.g. blur, glow → IBR)
- Post processing step in image space to create visual effects
  - Render window-sized, view-aligned quads with some Appearance
  - Additionally provide some way to control the composition methods

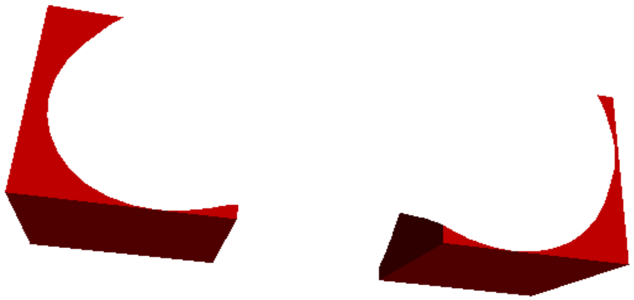
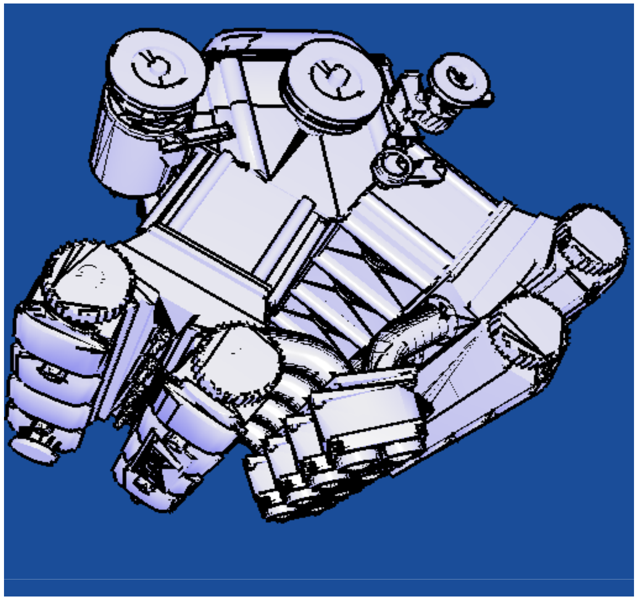
# Real-time Shadows in X3D

- Requirements
  - Robust and intuitive usage
  - Applicable for every type of scene
  - No special treatment for shaders
- Solution
  - No special shadow nodes, but extension of existing light nodes for regulating light and shadow
  - Generic parameter/ abstraction level for supporting different types of implementations
  - Example (values in [0,1])

```
SpotLight {
     shadowIntensity 0.7
     direction  0 -1 0
     location  -2 14 2
}
```



# Render State Control

- Access to color masking and arbitrary masking (i.e. stencil) in combination with defined rendering order for compositing
- Special materials for front/ back faces beyond *TwoSidedMaterial*
- Possibility to disable depth writing or using different depth functions
- Compositing of objects or foregrounds via blending, discarding, etc.

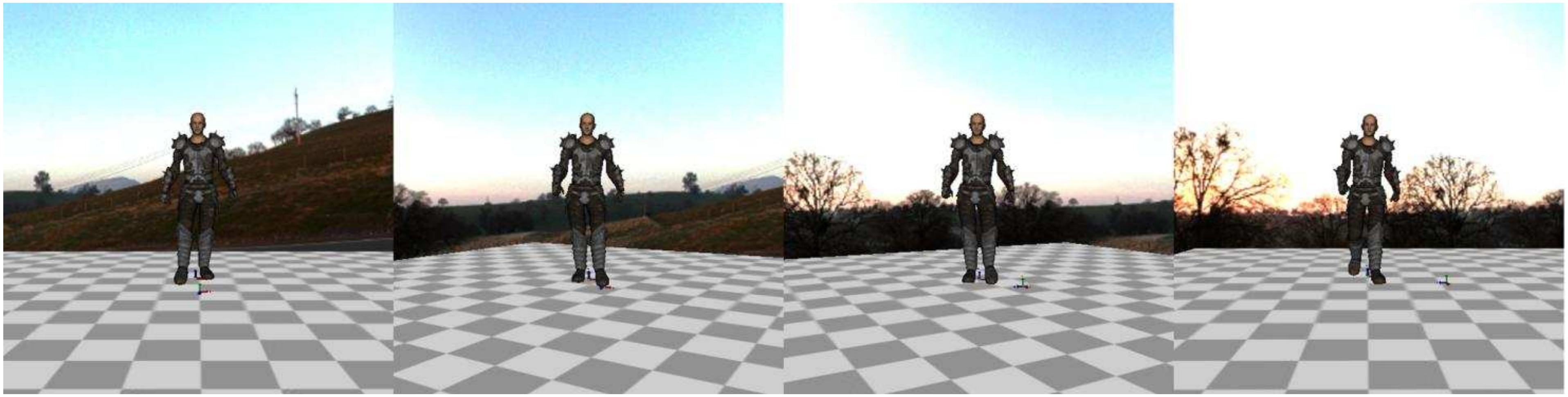
# X3D Node Extensions – Appearance

```
Appearance : X3DAppearanceNode {
  SFInt32 [in,out] sortKey          0
  SFNode  [in,out] fillProperties   NULL
  SFNode  [in,out] lineProperties   NULL
  SFNode  [in,out] material         NULL
  MFNode  [in,out] shaders          []
  SFNode  [in,out] texture          NULL
  SFNode  [in,out] textureTransform NULL
  SFNode  [in,out] blendMode        NULL
  SFNode  [in,out] stencilMode      NULL
  SFNode  [in,out] colorMaskMode    NULL
  SFNode  [in,out] depthMode        NULL
  SFNode  [in,out] faceMode         NULL
}

AppearanceGroup : X3DGroupingNode {
  SFBool [in,out]   render        TRUE
  MFNode [in,out]   children      []
  SFNode [in,out]   appearance    NULL
}
```

- *Appearance* reveals how *Shape* node looks like → extend shape component with some new nodes for setting different render states
  - …and *Appearance* with suiting fields
  - Maps to GPU, no PROTOs possible!
- Need to control the color-/ stencil-/ depth-buffer writing and merging → Requirement: control over rendering order
  - Introduce "sortKey" field (default is 0)
  - More robust and intuitive than e.g. a special ordering group for rendering
- Nodes for fine grained render state control
  - If corresponding fields in *Appearance* not set, standard settings are used
- New *AppearanceGroup* node useful, if whole group of nodes share same material
  - Field “render” (shared by all grouping nodes) simplifies setting of visibility

# X3D Node Extensions – Render States

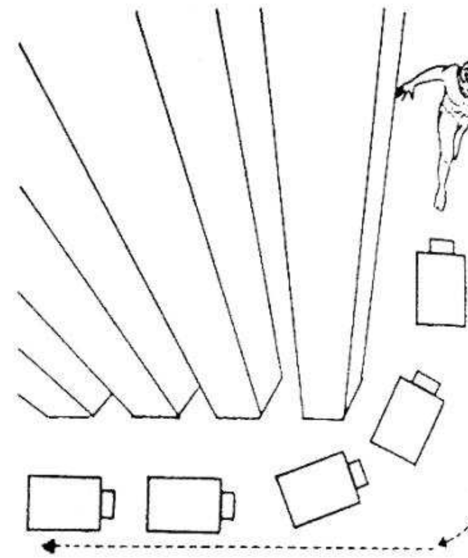
```
StencilMode : X3DAppearanceChildNode {
  SFString [in,out] stencilFunc "none"
  SFInt32  [in,out] stencilValue 0
  SFInt32  [in,out] stencilMask  0
  SFString [in,out] stencilOpFail  "keep"
  SFString [in,out] stencilOpZFail "keep"
  SFString [in,out] stencilOpZPass "keep"
  SFInt32  [in,out] bitMask  -1
}

ColorMaskMode : X3DAppearanceChildNode {
  SFBool [in,out] maskR TRUE
  SFBool [in,out] maskG TRUE
  SFBool [in,out] maskB TRUE
  SFBool [in,out] maskA TRUE
}

FaceMode : X3DAppearanceChildNode {
  SFString [in,out] cullFace     "auto"
  SFString [in,out] frontFace    "auto"
  SFString [in,out] frontMode    "auto"
  SFString [in,out] backMode     "auto"
  SFFloat  [in,out] offsetFactor 0
  SFFloat  [in,out] offsetBias   0
}

BlendMode : X3DAppearanceChildNode {
  SFString [in,out] srcFactor  "one"
  SFString [in,out] destFactor "zero"
  SFColor  [in,out] color 1 1 1
  SFFloat  [in,out] colorTransparency 0
  SFString [in,out] alphaFunc "none"
  SFFloat  [in,out] alphaFuncValue 0
}

DepthMode : X3DAppearanceChildNode {
  SFBool   [in,out] enableDepthTest TRUE
  SFString [in,out] depthFunc "none"
  SFBool   [in,out] readOnly FALSE
  SFFloat  [in,out] zNearRange -1
  SFFloat  [in,out] zFarRange  -1
}
```



# CAMERA CONTROL AND VFX

# Extending the Camera: Cinematographic Camera Placement

- X3D describes content declaratively
  - …but placing a *Viewpoint* properly requires tools or trial-and-error
    - Especially camera animations with moving targets
  - Describe camera pose and moves/ framing on screen declaratively
    - Standard interactive navigation modes not adequate
- Creative people tend to think in images
  - …but not how to achieve them by defining 3D position/orientation
  - For storyboarding/ film scenes camera is defined relative to objects
- Cinematographic Viewpoint: declarative approach to camera placement
  - Idea based on well-established techniques from the film area
    - Allows specifying what objects shall appear where on screen
  - Intuitive framing of objects useful for pre-vis or dialog systems
    - Supports camera moves that are bound to interactive content
- Camera/ Viewpoint additionally includes special visual effects as child nodes
  - DepthOfFieldFX, BlurFX, SketchFX, etc.



Jung, Y., and Behr, J. Towards a new camera model for X3D. Web3D 20

# The CinematographicViewpoint Node

```
X3DViewpointNode : X3DBindableNode {
  MFNode     [in,out] effects        []
  [...]
}
CinematographicViewpoint : X3DViewpointNode {
  [...]
  MFNode     [in,out] objectsFull    []
  MFNode     [in,out] objectsCloseUp []
  SFVec3f    [in,out] facingDir      0 0 1
  SFVec3f    [in,out] upVector       0 1 0
  MFVec2f    [in,out] minScreenPos   []
  MFVec2f    [in,out] maxScreenPos   []
  SFString   [in,out] shotSize       "auto"
  SFFloat    [in,out] shotAngle      0
  SFFloat    [in,out] shotPitch      0
  SFFloat    [in,out] shotRoll       0
  SFString   [in,out] follow         "none“
}
```

- Perspective camera node
  - Has effects field for VFX
- objectsCloseUp field
  - Refers to scene object, e.g. head
  - For given shot sizes, starting with ‘close’
- objectsFull: other shots (refers to human)
- shotSize: common shot sizes for actors
- min-/maxScreenPos
  - Bbox in normalized screen coordinates
  - Refers to objectsFull/ objectsCloseUp field
  - Modified by shotSize
- shotAngle/-Pitch/-Roll
  - Offset from line of action/ floor/ up vector
- follow: target nodes continuously or not

# Special Visual Effects (VFX)

- Specific to camera and lens system, usually implemented as post-processing step on GPU (DoF, blur,…)

```
X3DVisualEffects : X3DNode {
 SFBool [in,out] enabled TRUE
}

SketchFX : X3DVisualEffects {
 SFBool  [in,out] enabled TRUE
 SFInt32 [in,out] thickness 1
}

DepthOfFieldFX : X3DVisualEffects {
 SFFloat [in,out] focalDepth 10.0
 SFFloat [in,out] blurCutoff 0.7
}

HDRRenderingFX : X3DVisualEffects {
 SFFloat [in,out] exposure 1.64
 SFFloat [in,out] brightnessThreshold 1.0
}

ScreenSpaceAmbientOcclusionFX :
  X3DVisualEffects {
 SFFloat [in,out] scale 0.001
 SFFloat [in,out] attenuation 0.001
}

MotionBlurFX : X3DVisualEffects {
 SFString [in,out] type "auto"
 SFFloat  [in,out] strength 0.02
}

BlurFX : X3DVisualEffects {
 SFBool   [in,out] enabled TRUE
 SFString [in,out] kernelType "auto"
 SFInt32  [in,out] kernelSize 5
 SFBool   [in,out] grain FALSE
 SFBool   [in,out] blackAndWhite FALSE
}
```

# Enhancing Quality with VFX

No **DepthOfFieldFX**; DoF and character nearby; different focal depth; character further away

**BlurFX** effects node for old-fashioned look of chapel scene: standard rendering, only blur with grain, black-and-white rendering, all effects combined (black-and-white, slight blur, grain)

Fraunhofer
IGD

# Thank you!

# Questions?