Volume Deformation and Animation

Klaus Engel
Siemens Corporate Research
Princeton

Markus Hadwiger
VR VIS Research Center
Vienna, Austria

Joe M. Kniss
Scientific Computing and
Imaging Insitute,
University of Utah

Aaron E. Lefohn
Institute for Data Analysis
and Visualization.
University of California, Davis

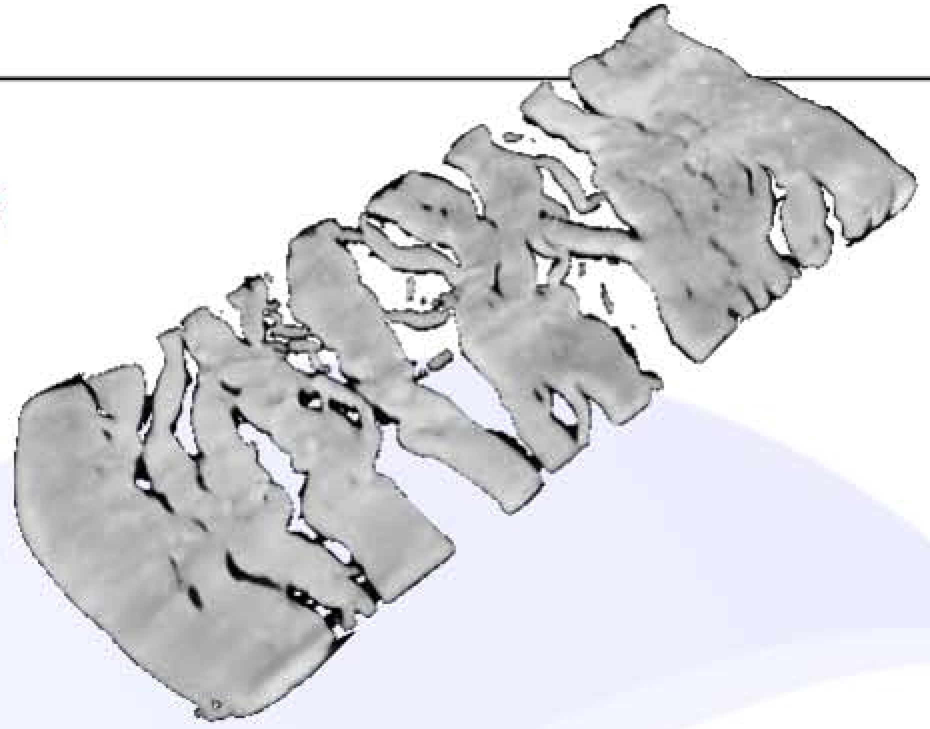
Christof Rezk Salama
Computer Graphics and
Multimedia Group
University of Siegen, Germany

Daniel Weiskopf
Visualization and Interactive
Systems Group,
University of Stuttgart, Germany

Motivation

Deformable Volumetric Objects

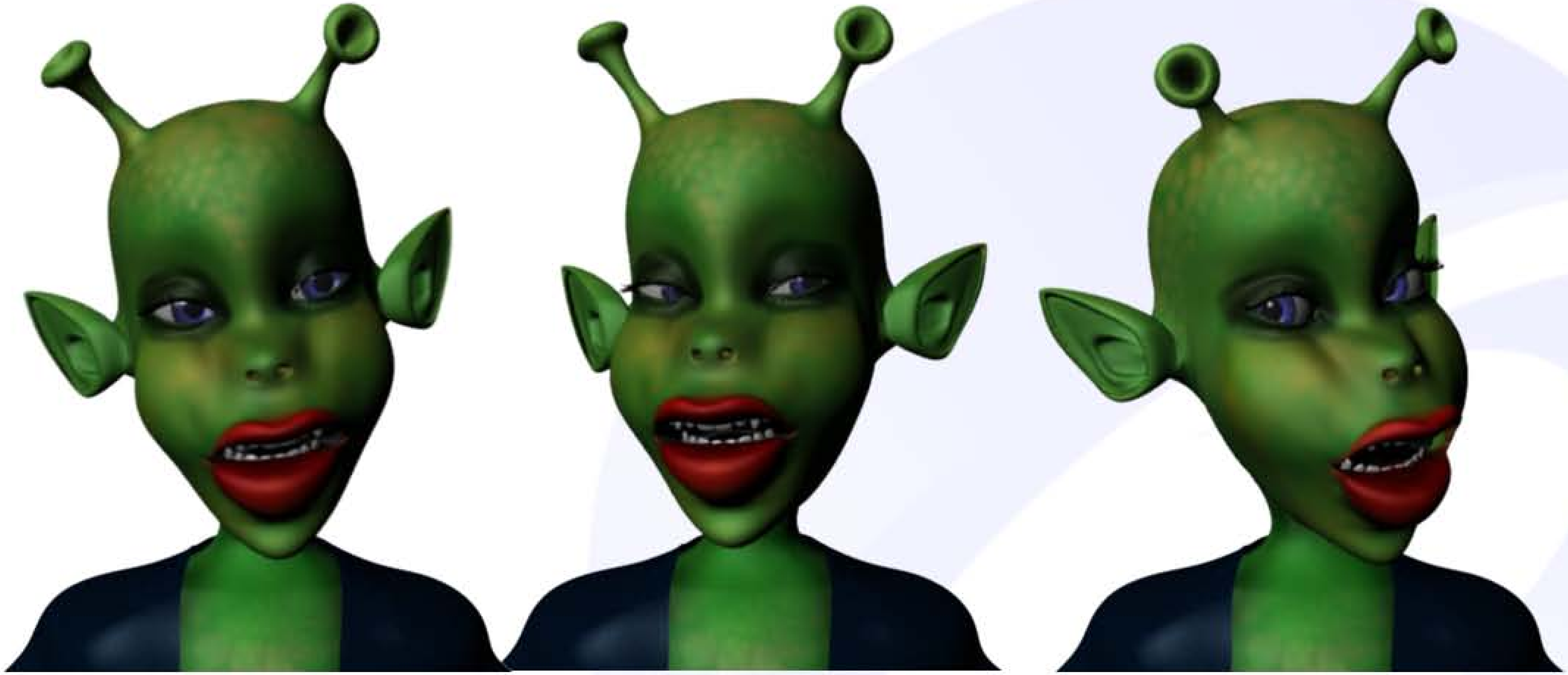
- ***Applications in Science***
  - Medicine
  - Engineering
  - Natural Science



- ***Applications in Arts***
  - Translucent Objects with true volumetric deformation
  - Keyframe Animation
  - Procedural Animation

Modelling

- Traditional Modelling:
  Separation of Shape from Appearance



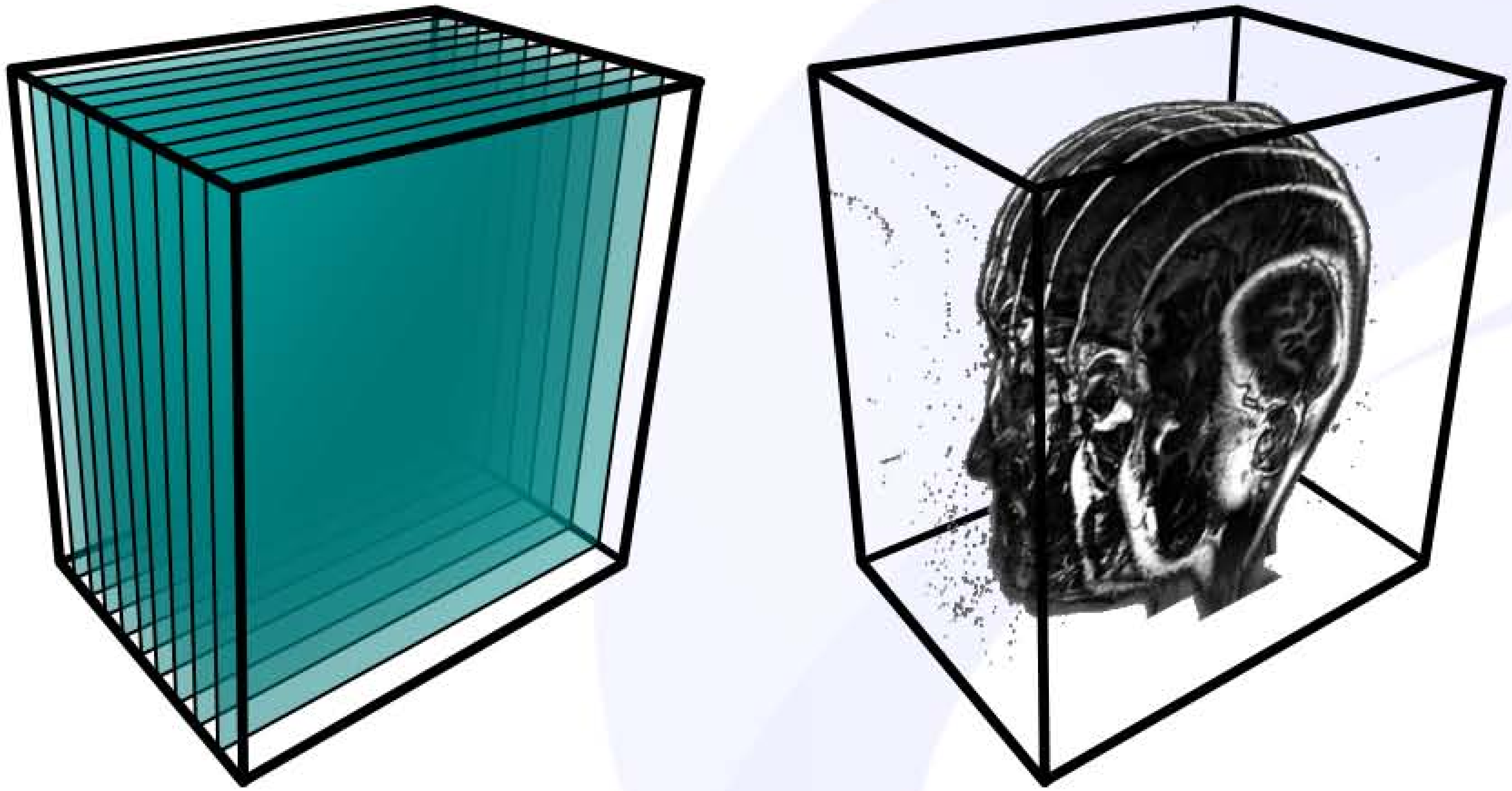
Images from Maya Animation Course, © Rezk-Salama, University of Siegen

- *Deformation of the Shape (Geometry) only*
- *Appearance (Materials, Textures etc.) remains unchanged*

Texture Based VR

Shape and Appearance

- Proxy geometry does not define the shape of object
- Both shape and appearance are defined by 3D textures



Should we deform the proxy geometry or the textures?

Mathematical Models

Deformation Models for Texture-Based VR

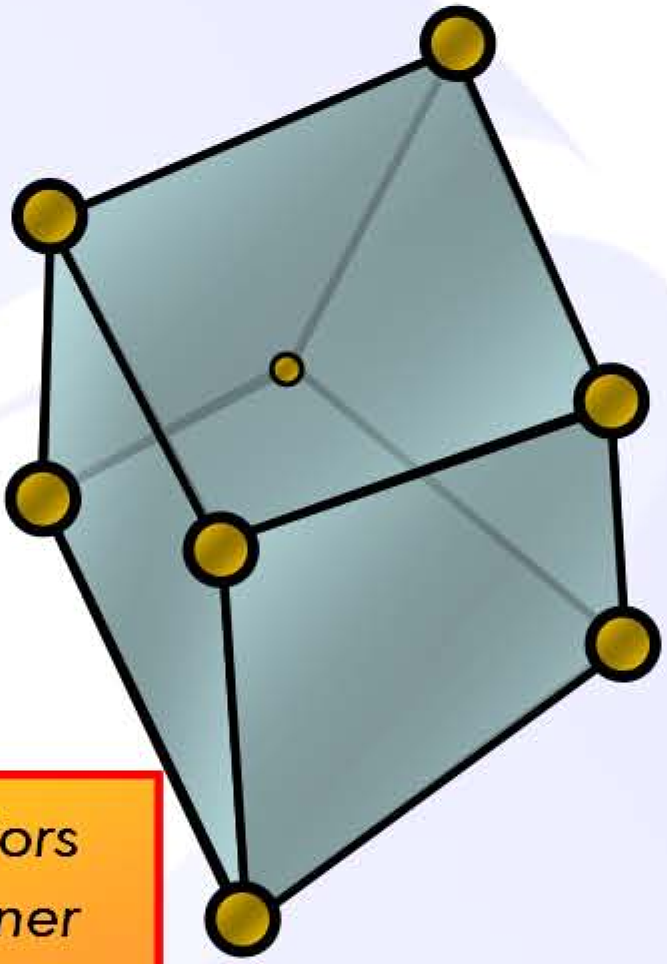
- Deforming the proxy geometry

First Idea:
Simply displace the 8 corner vertices of the bounding box (before slicing it)
Mathematical Description:

$$\Phi(\vec{x}) = \vec{x} + \sum_{i,j,k \in \{0,1\}} a_{ijk} \cdot \vec{t}_{ijk}$$

Trilinear interpolation weights of point x in the undeformed grid

Translation vectors given at the corner vertices



Mathematical Models

Deformation Models for Texture-Based VR

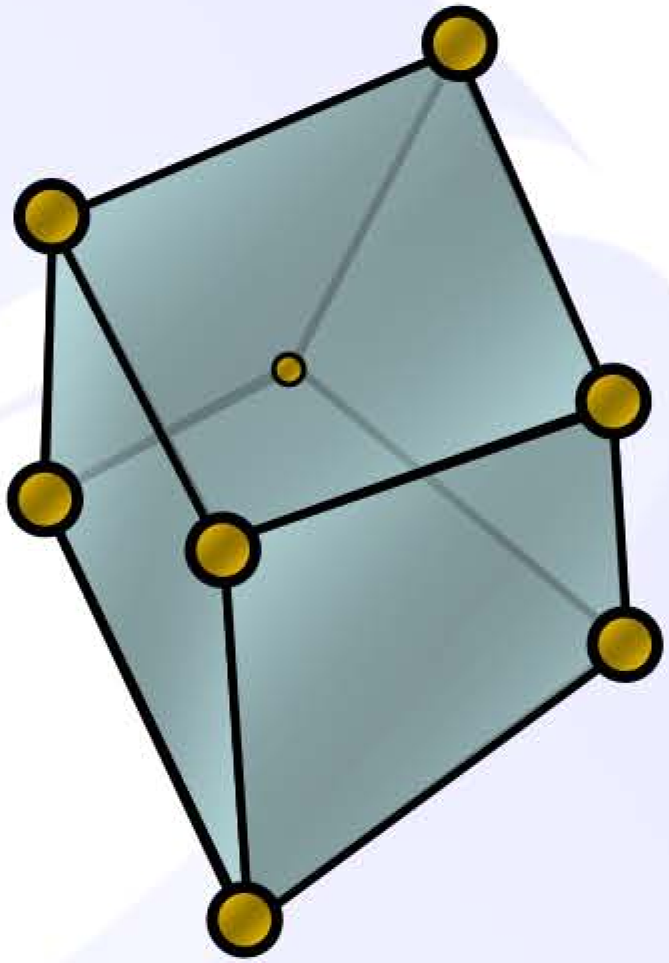
- Deforming the proxy geometry

First Idea:

Simply displace the 8 corner vertices of the bounding box (before slicing it)

Mathematical Description:

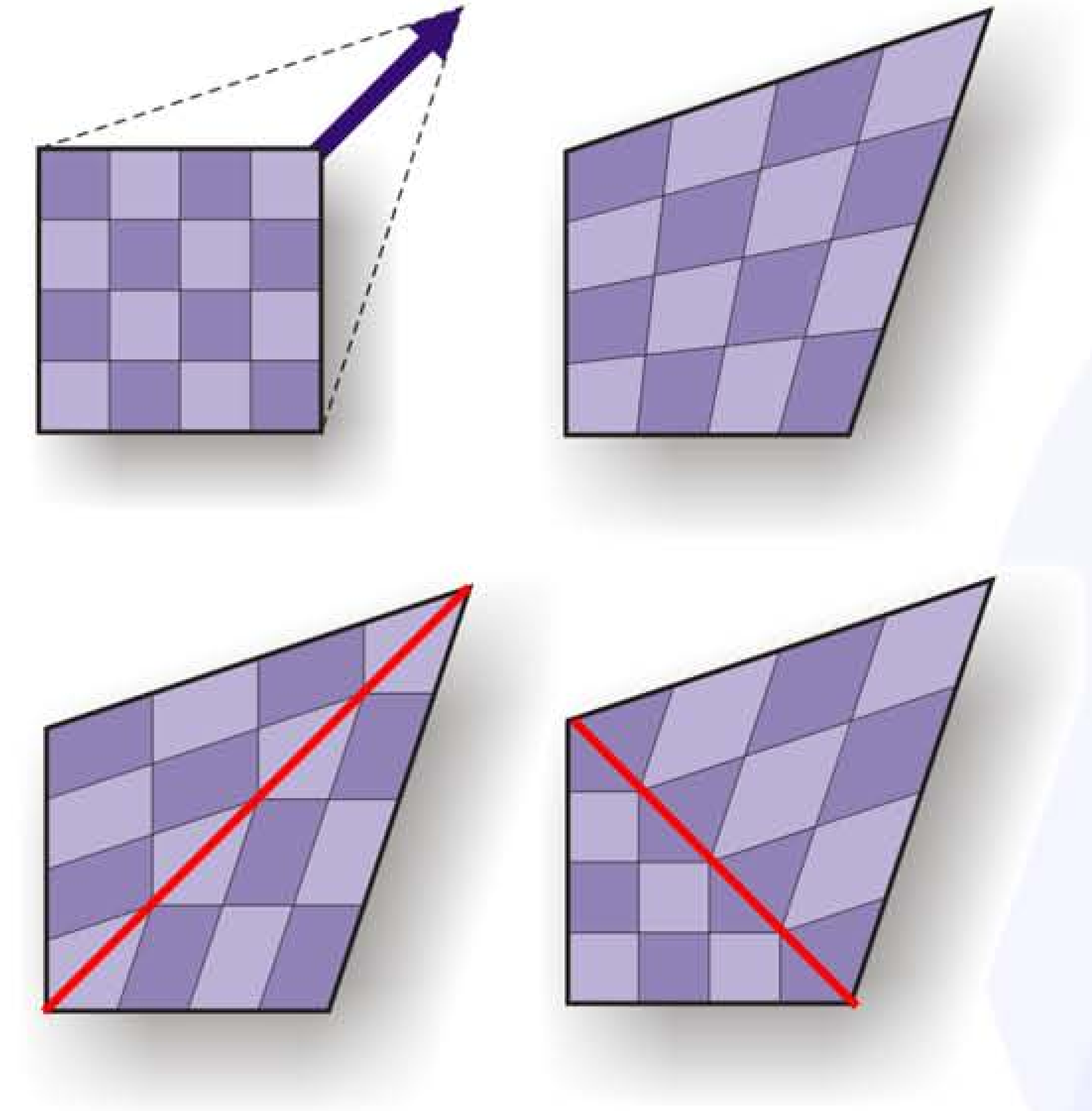
$$\Phi(\vec{x}) = \vec{x} + \sum_{i,j,k \in \{0,1\}} a_{ijk} \cdot \vec{t}_{ijk}$$



Difficulties: The inverse transformation is not again a trilinear function!

Mathematical Models

- *What do we need the inverse for?*



If we displace the vertices, but keep the texture coordinates constant,

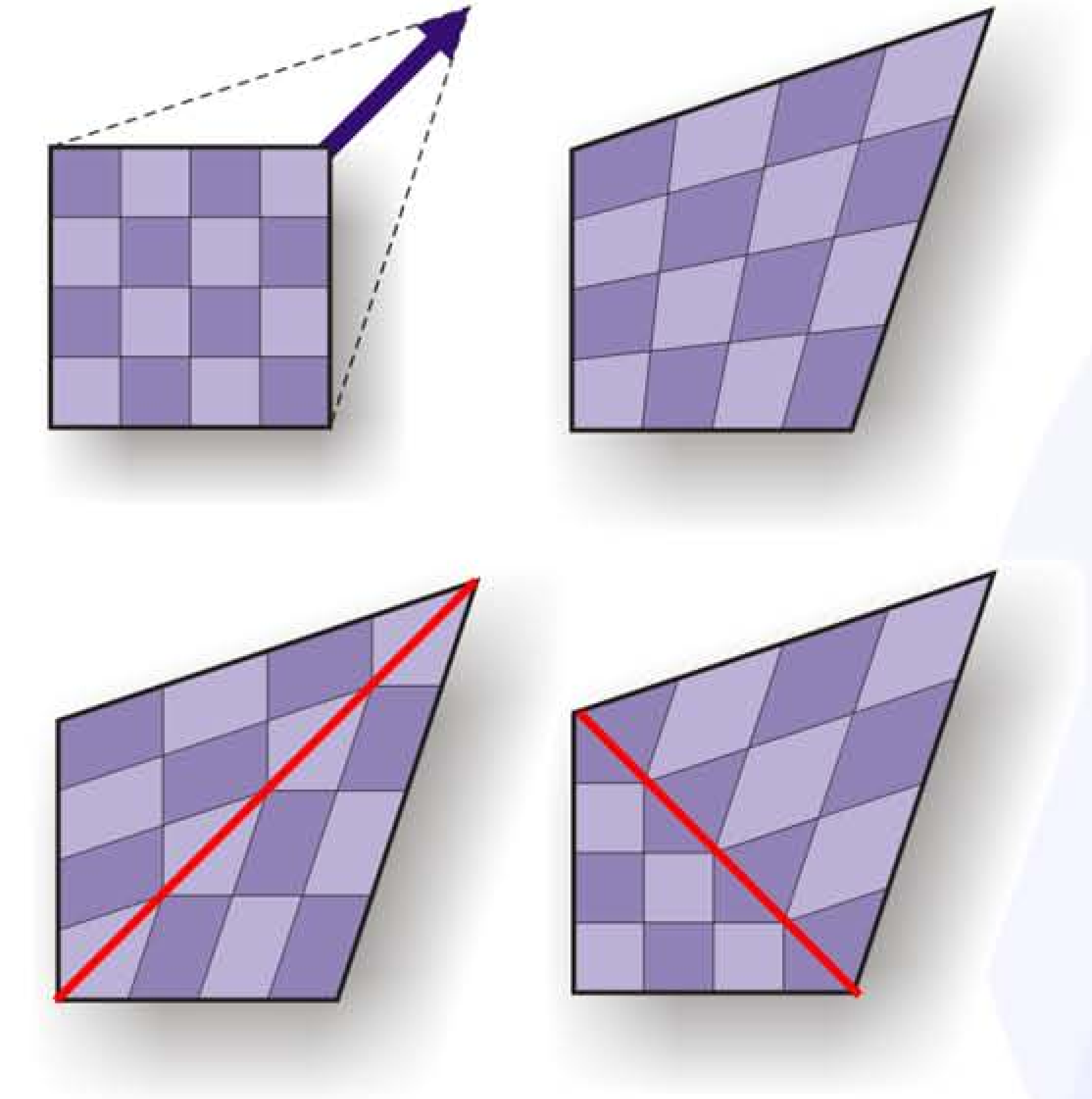
Tessellation into triangles produces undesired results.

Rasterization:
For the desired bilinear/trilinear mapping,

the inverse transformation is required to determine the correct texture coordinates.

Mathematical Models

- *What do we need the inverse for?*



If we displace the vertices, but keep the texture coordinates constant,

Tessellation into triangles produces undesired results.

Rasterization:
For the desired bilinear/trilinear mapping,

the inverse transformation is required to determine the correct texture coordinates.

In 3D: polygons also become non-planar in texture space!

Mathematical Models

Deformation Models for Texture-Based VR

- Deforming the proxy geometry

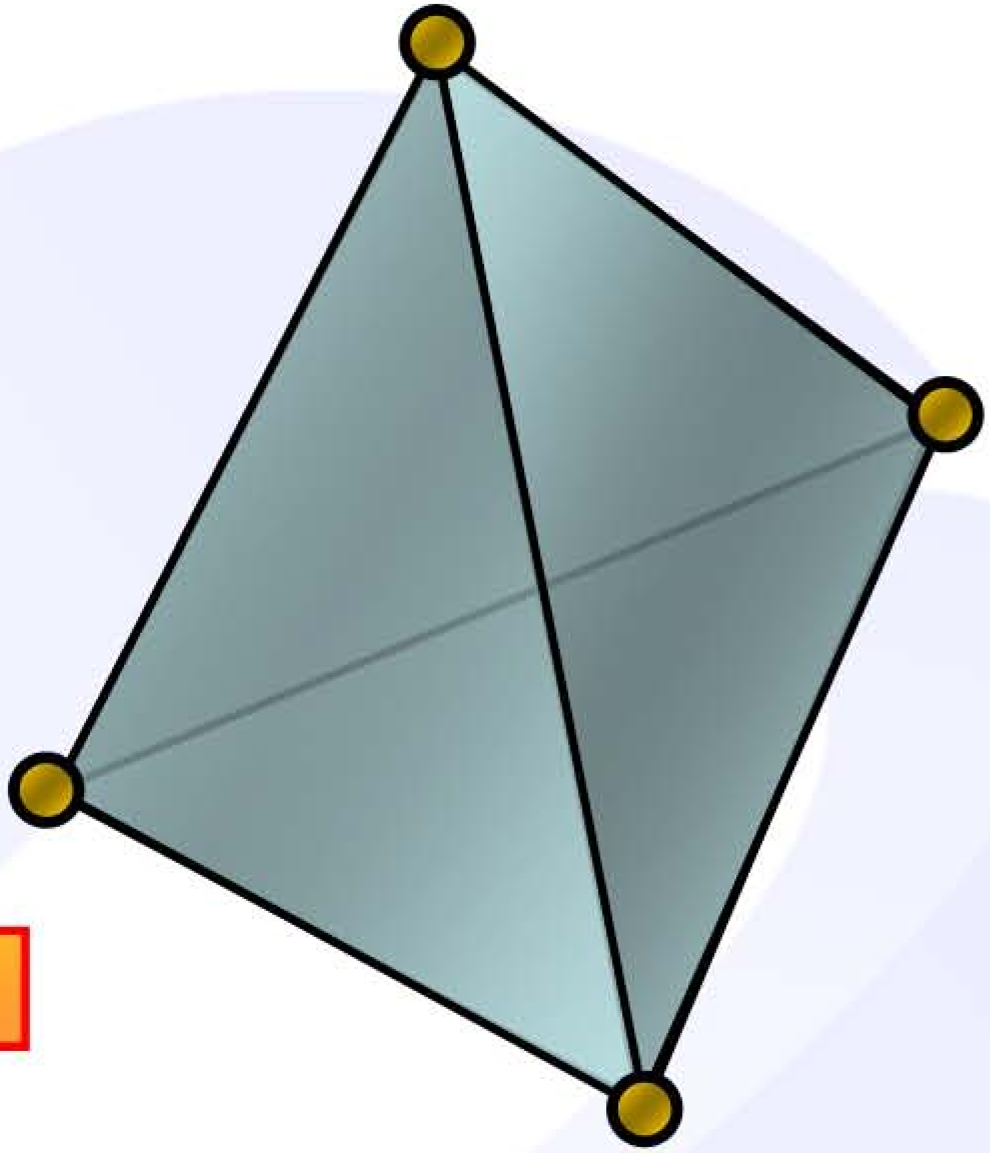
Second Idea:

Use tetrahedra as proxy geometry

Displace the 4 corner vertices.

Mathematical Description:

$$\Phi(\vec{x}) = \underbrace{\mathbf{A}}_{\text{Rotation and Scaling}}\vec{x} + \underbrace{\vec{b}}_{\text{Translation}}$$



Mathematical Models

Deformation Models for Texture-Based VR

- Deforming the proxy geometry

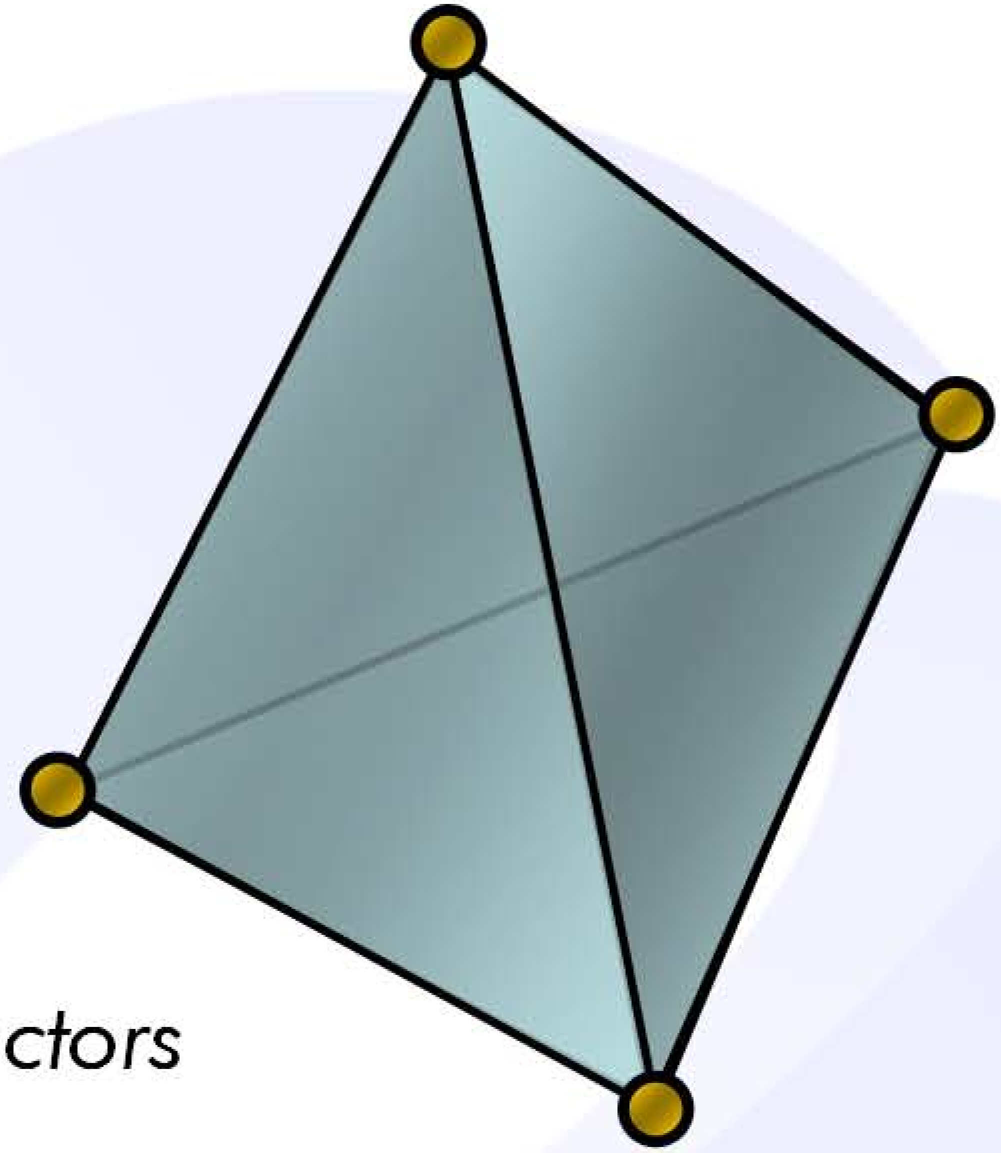
Second Idea:

Use tetrahedra as proxy geometry

Displace the 4 corner vertices.

Mathematical Description:

$$\Phi(\vec{x}) = \mathbf{A}\vec{x} + \vec{b}$$



Fully determined by 4 displacement vectors

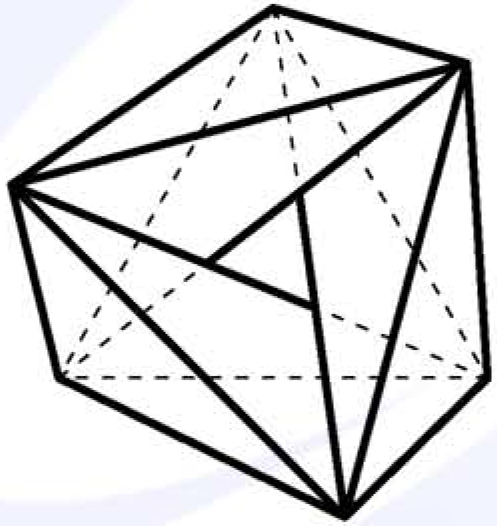
Difficulties: Tessellation, Depth Sorting

Tetrahedra Deformation

- Available in SGI's Volumizer API

Main Difficulties:

- Smooth Deformation requires high tessellation
- Depth sorting arbitrary tetrahedra meshes is a difficult problem
  - Especially true for non-convex tetrahedra meshes
  - Sorting not always possible (Visibility Cycles!)
- Slice Decomposition
  - Mainly performed on CPU



Mathematical Models

Deformation Models for Texture-Based VR

- Deforming the appearance (textures)

Piecewise Linear Transformation:

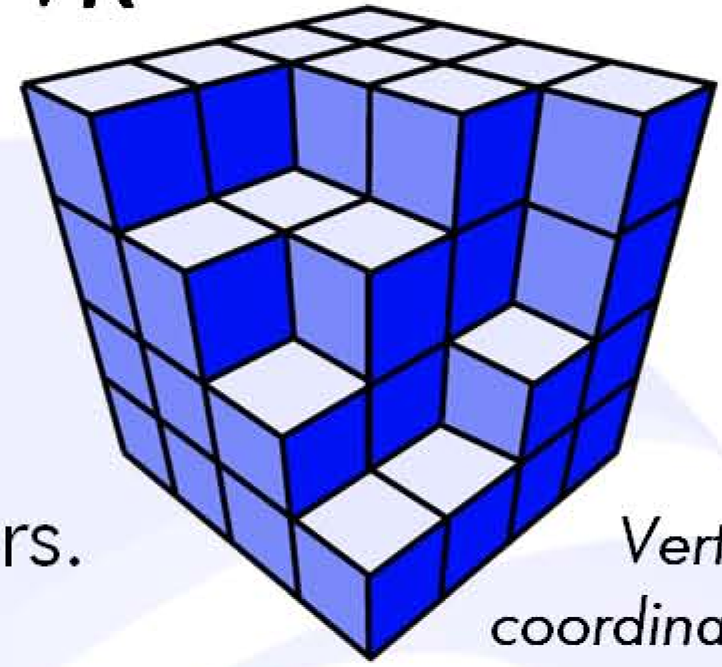
Subdivide into hexahedra cells (3D patches)

Displace the texture coordinates at the corners.

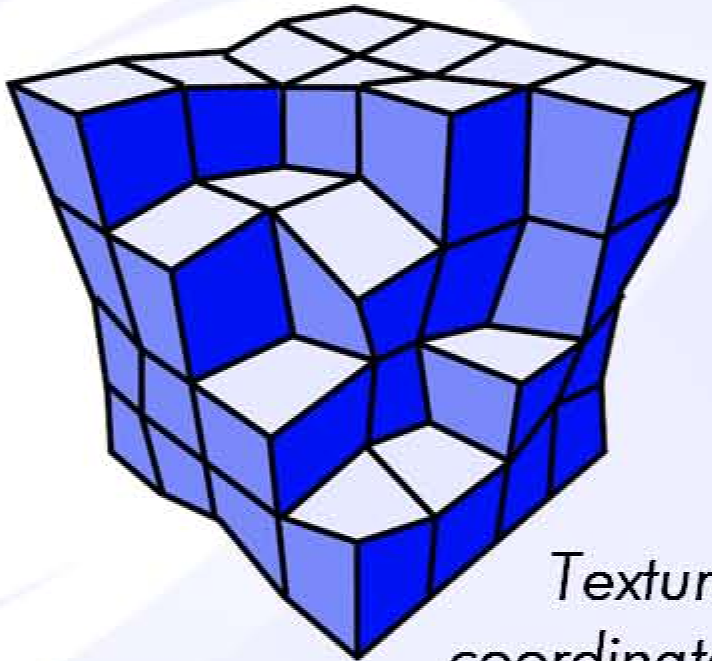
Mathematical Description:

$$\Phi(\vec{x}) = \vec{x} + \sum_{i,j,k\in\{0,1\}} a_{ijk}\cdot\vec{t}_{ijk}$$

*(**x** now refers to the texture coordinate)*



Vertex coordinates



Texture coordinates

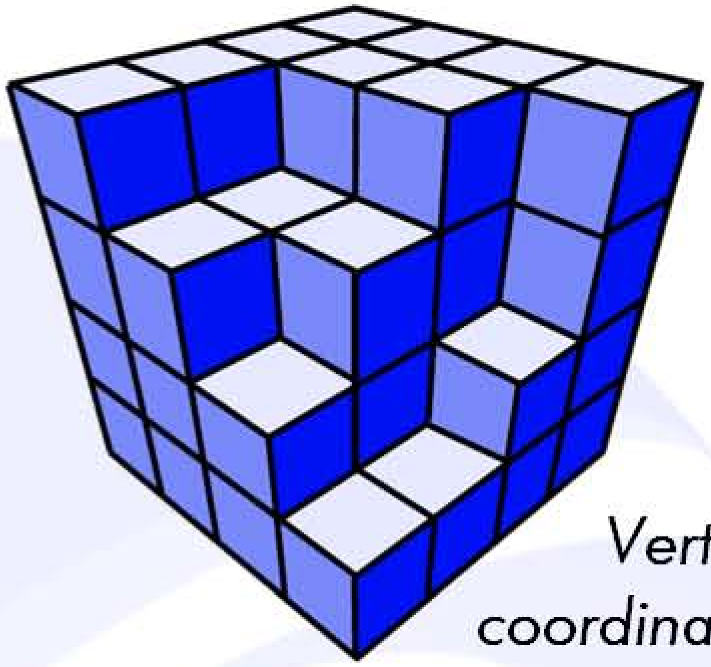
Piecewise Linear Patches

Advantages:

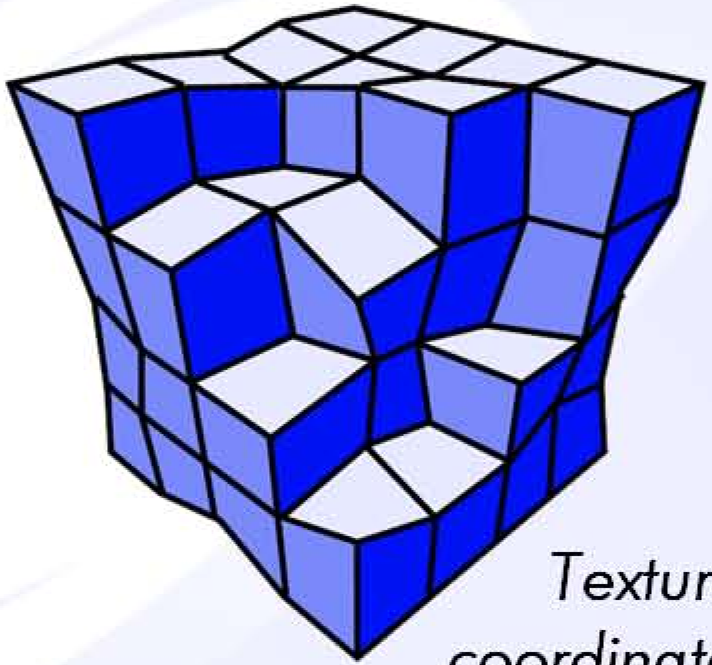
- *Geometry (vertices) is static, only texture coordinates change*
- *Slice decomposition is easy*
  - No expensive recomputation or real-time tessellation necessary
- *No depth sorting required!*
- *Adaptive subdivision possible*

Difficulties:

- How can we circumvent or approximate the *inverse deformation?*



Vertex coordinates

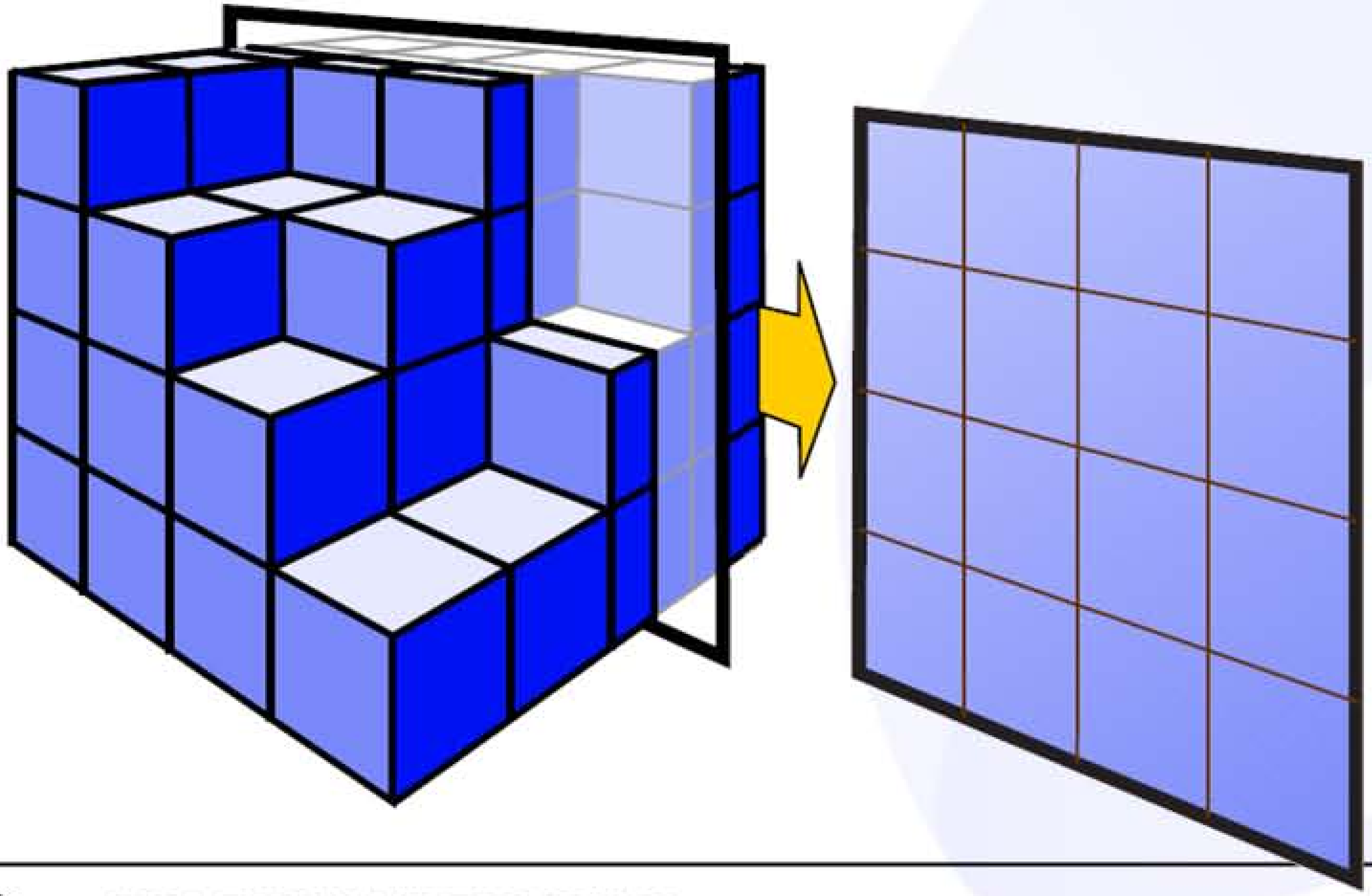


Texture coordinates

Piecewise Linear Patches

Rendering

- Store the volume as *a 3D texture*
- Static Geometry:
  use object aligned slices to preserve this benefit!



3 stacks of slices

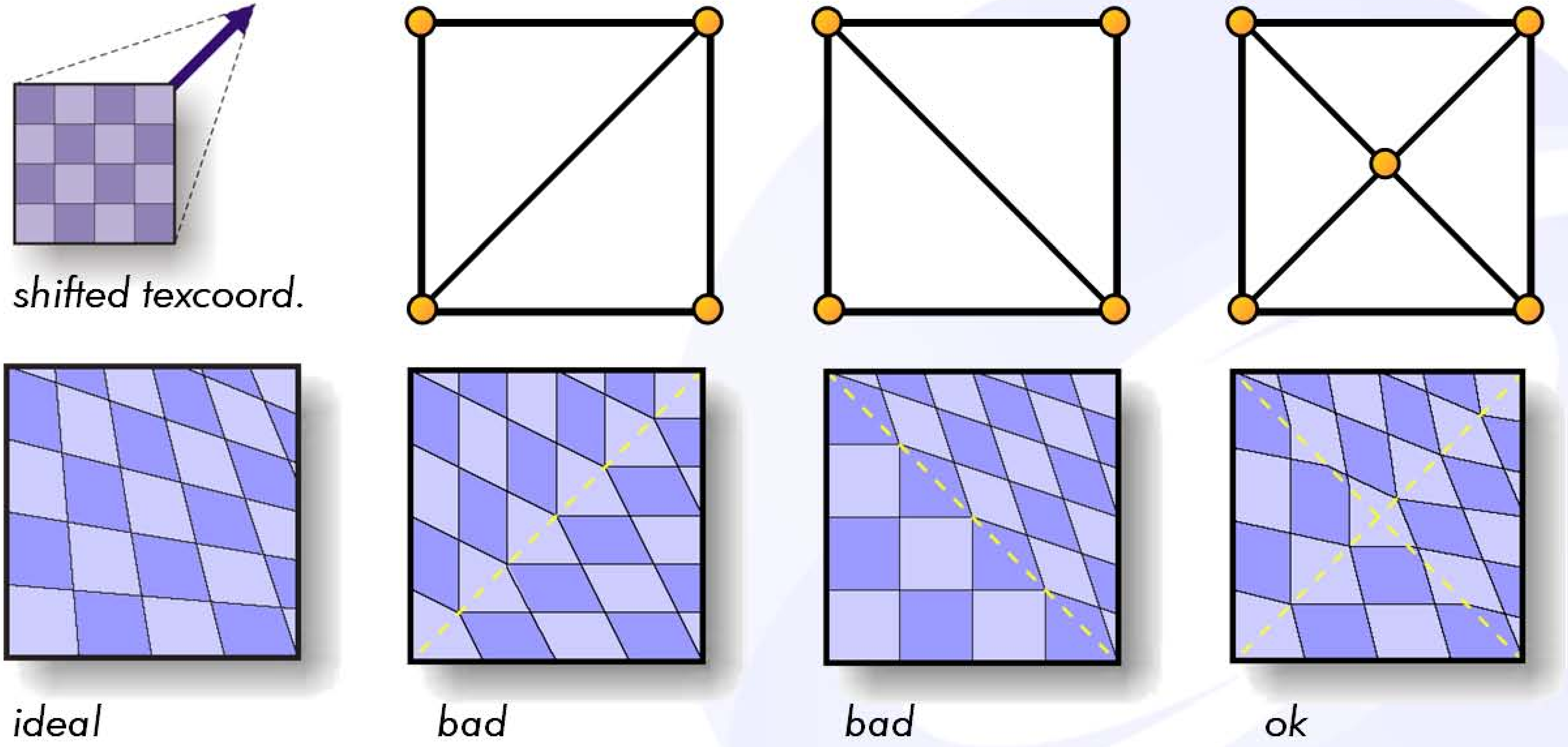
plus a 3D texture

How do we compute texture coordinates?

Piecewise Linear Patches

What do I need the inverse for?

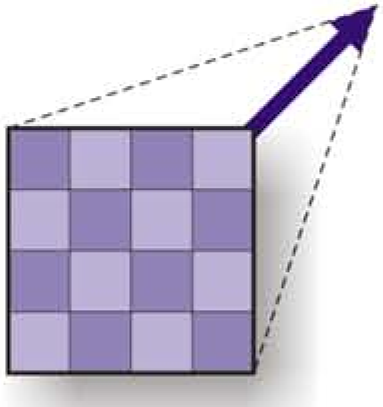
- *Texture Interpolation*



Piecewise Linear Patches

What do I need the inverse for?

- *Texture Interpolation*



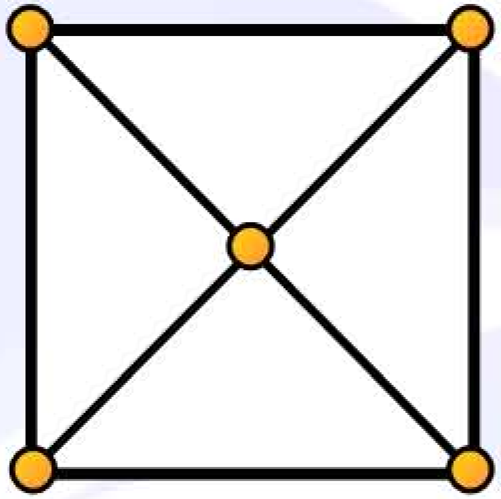
shifted texcoord.

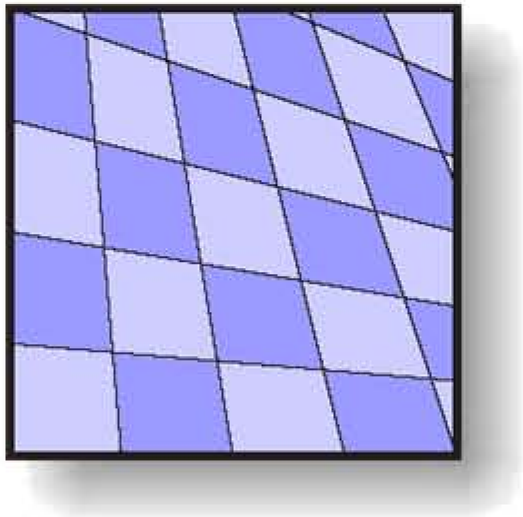
Approximate the correct bilinear interpolation by

4 interpolations in barycentric coordinates

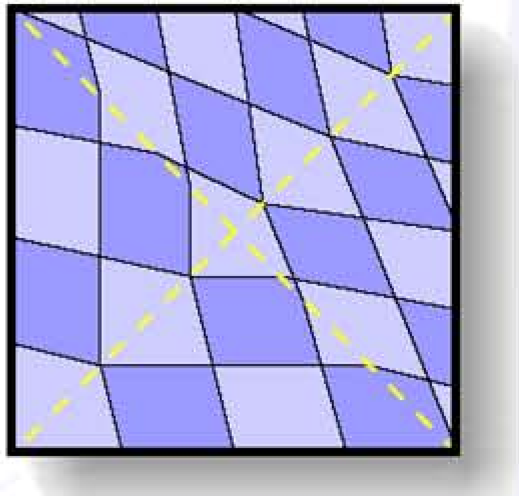
Use higher tessellation if quality is not good enough

Geometry is static!
No depth sorting required!





ideal



ok

Piecewise Linear Patches

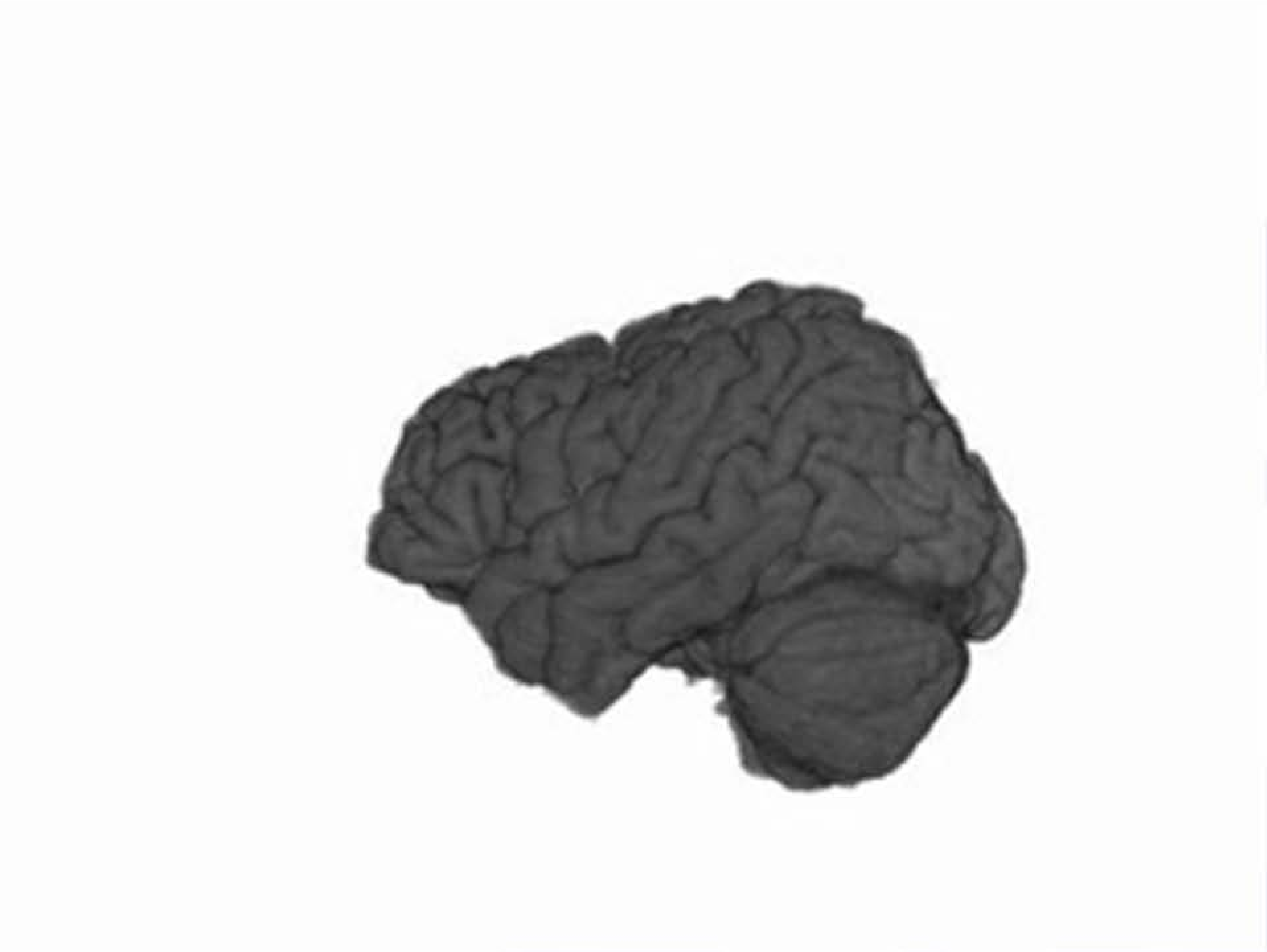
What do I need the inverse for?

- *Texture Interpolation*
- *Intuitive Modelling*
  - *The user does not want to manually specify texture coordinates*
  - *Instead: Picking and dragging of control points*
  - *Only coarse approximation to the correct inverse function is required:*

$$\widetilde{\Phi}^{-1}(\vec{x}) = \vec{x} + \sum_{i,j,k \in \{0,1\}} a_{ijk} \cdot -\vec{t}_{ijk}$$

simply negate the displacement vectors

Examples

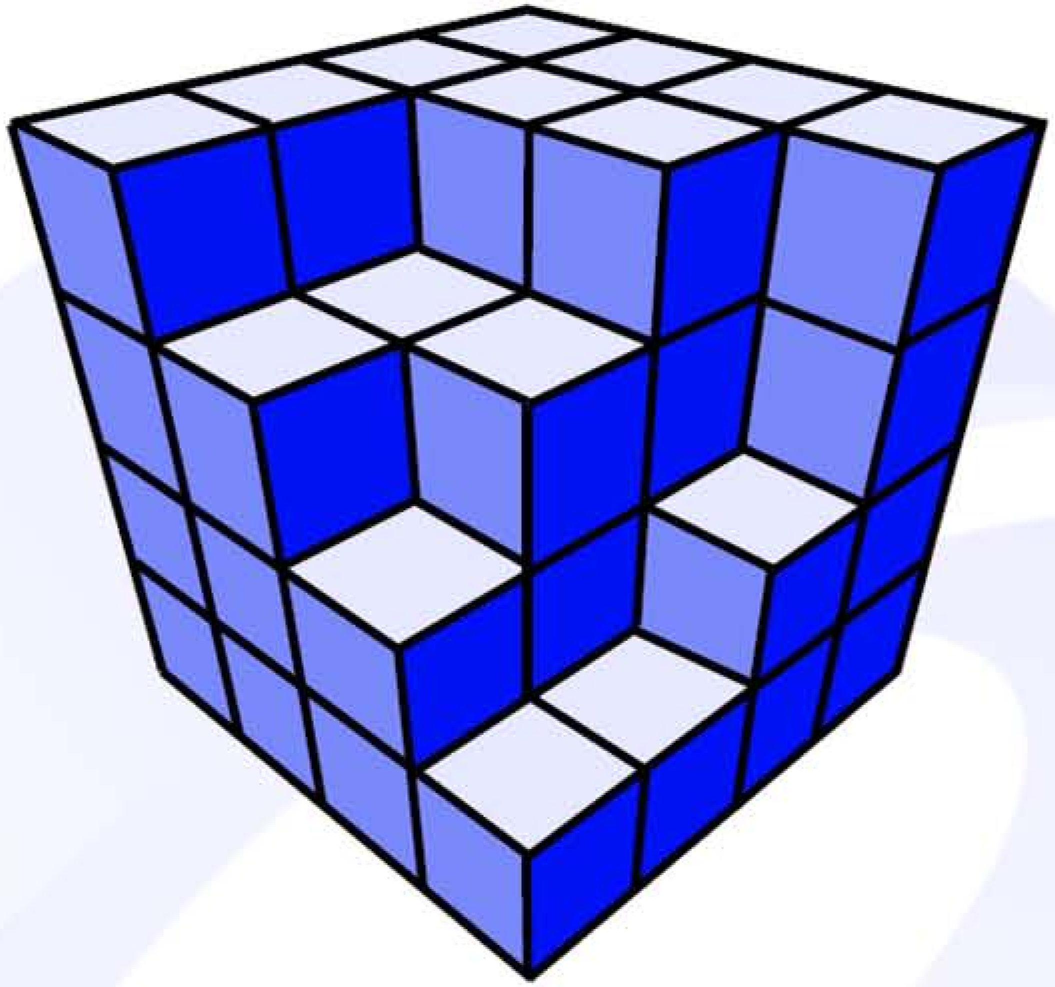


Piecewise Linear Patches

High Flexibility

- requires high subdivision level

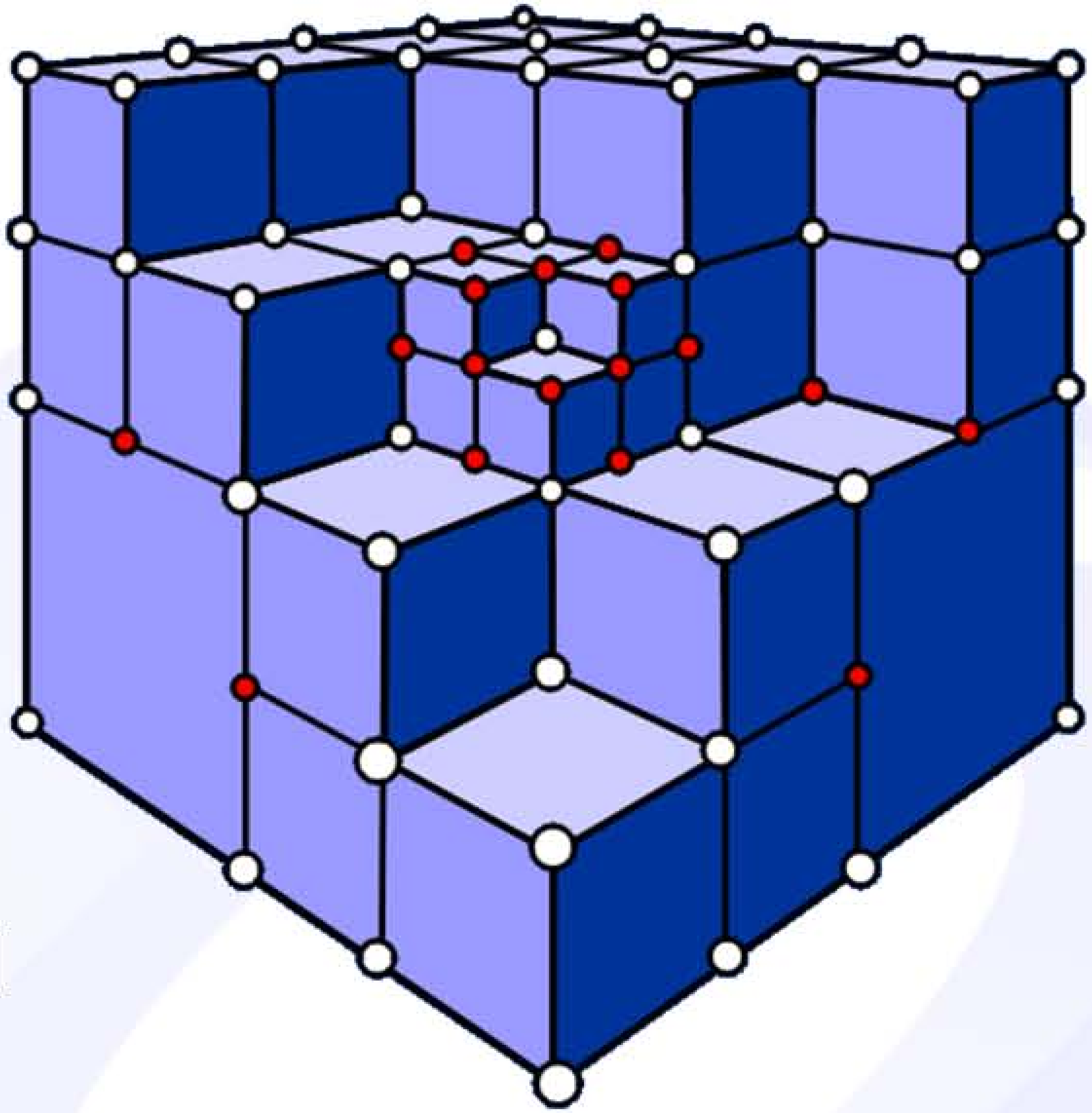
⇒ high number of free vertices



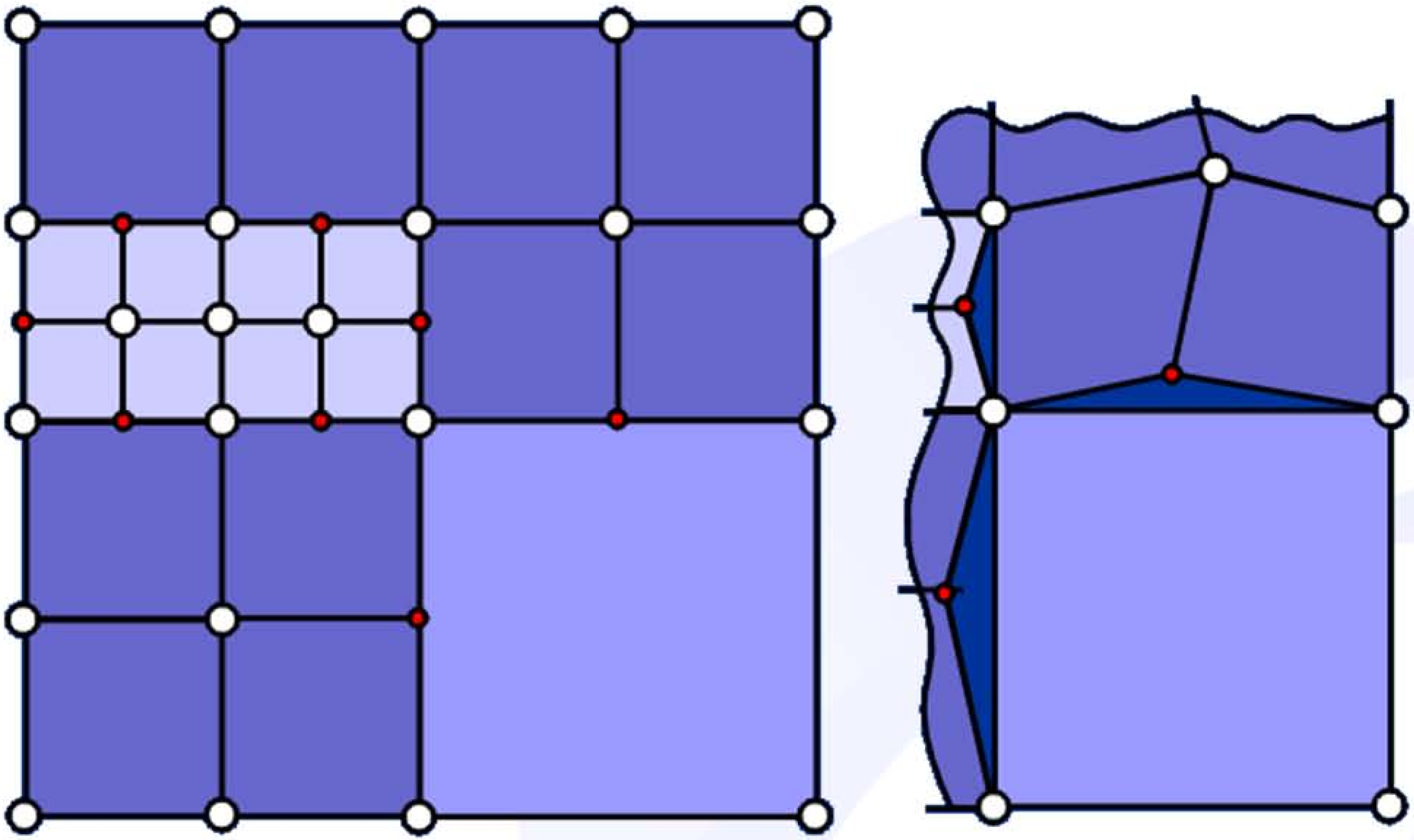
Piecewise Linear Patches

High Flexibility

- requires high subdivision level
- ⇨ high number of free vertices
- can be reduced by *Adaptive Subdivision*
  - Refine only where higher flexibility is required
  - *Octree structure*
- *Constraints required to prevent gaps in texture space!*

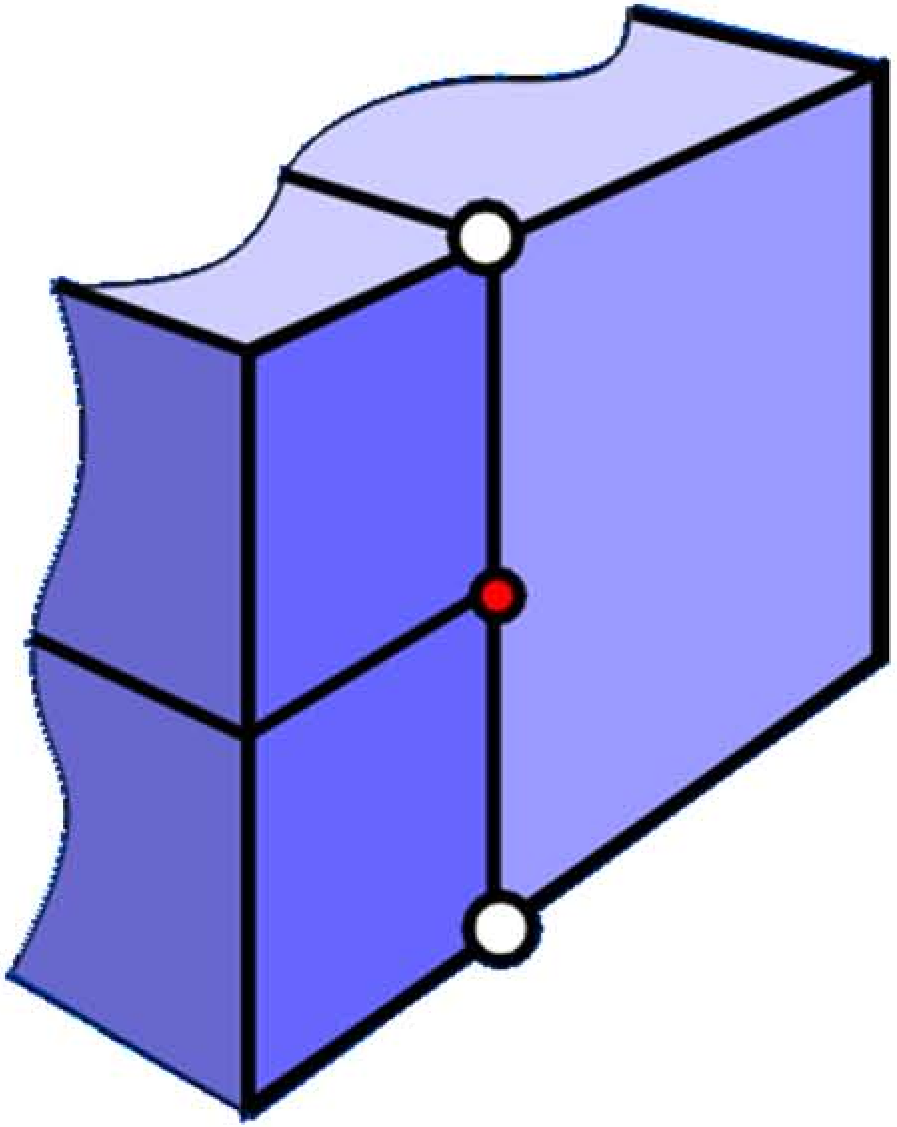


Adaptive Subdivision



- ○ Free Vertices
- ● Vertices with constraints

Edge Constraints



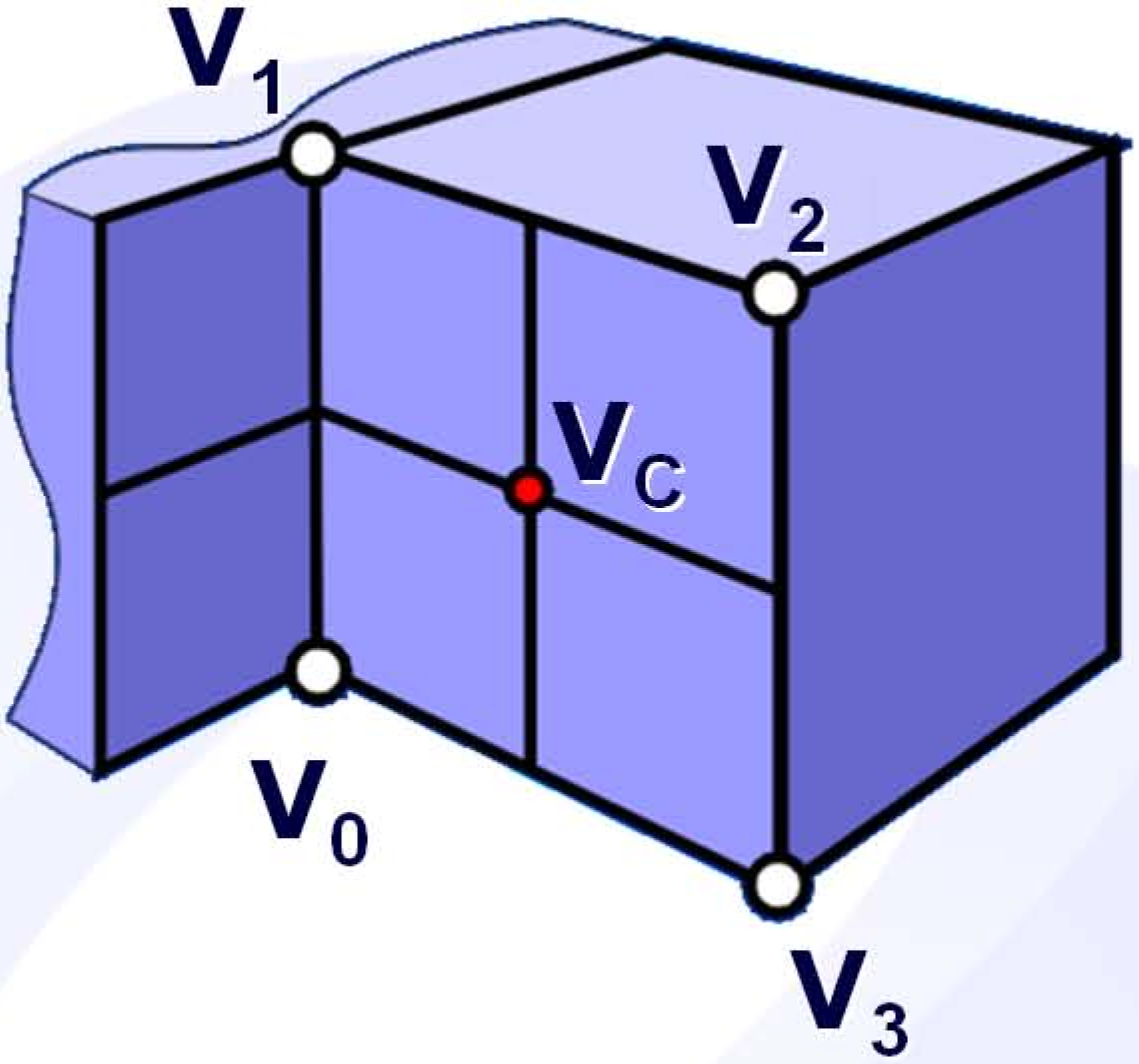
- Vertices on edges between different subdivision levels must stay ***collinear***

$$\vec{V}_C = \frac{1}{2}\vec{V}_0 + \frac{1}{2}\vec{V}_1$$

Face Constraints

- Vertices on faces with different subdivision levels must stay coplanar

$$\vec{V}_C = \frac{1}{4} \sum_{i=0...3} \vec{V}_i$$



Local Illumination

Local Illumination

- Pre-computed gradient vectors become invalid after the deformation
  - *Use on-the-fly gradient estimation techniques OR*
  - *Adapt pre-computed gradient vectors to the deformation*

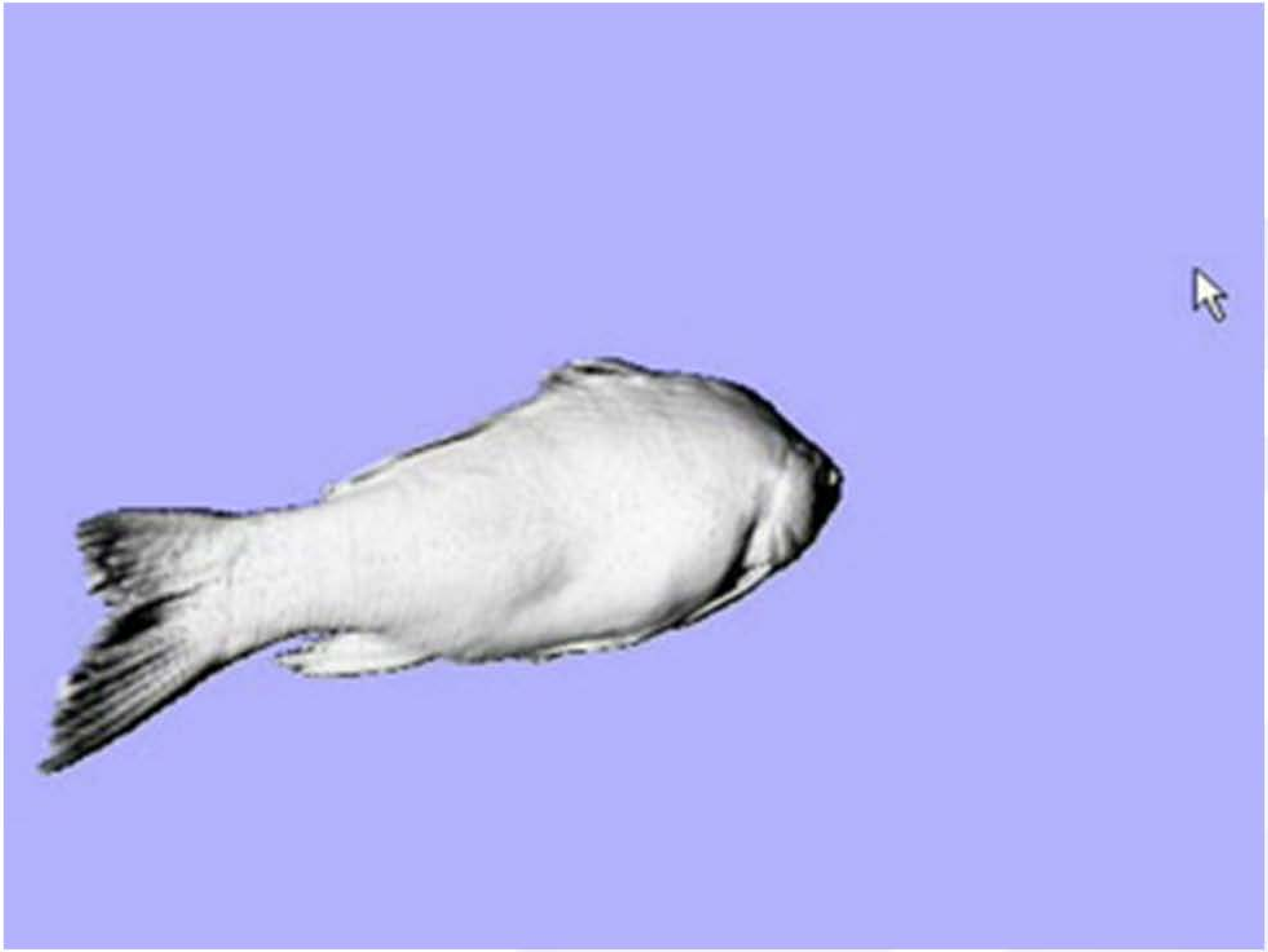
  Idea: Approximate trilinear mapping by affine matrix

$$\Phi(\vec{x}) = \mathbf{A}\vec{x}$$

(in homogenous coordinates)

Use this equation to approximate the inverse mapping and to adapt pre-computed gradient vectors

Examples

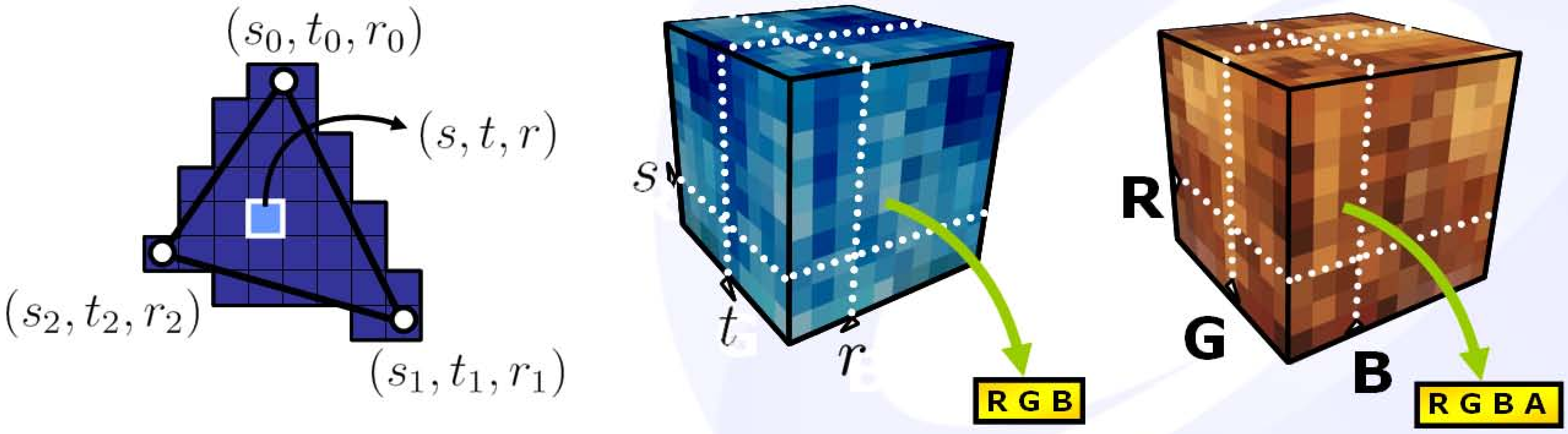


Volumetric Deformation

Deformation Models for Texture-Based VR

- Deforming the appearance (textures)

Dependent Textures /Offset Textures
Specify a deformation field as an additional 3D texture.



Dependent Textures

- Basically the same mathematical model as for piecewise linear patches
- Inverse mapping is avoided by 3D texture lookup
- Works both with object- and viewport-aligned slices
- Resolution of offset texture is independent of volume texture
- No adaptive subdivision
- Gradient adaptation difficult
- Runs completely within GPU (except slicing)
- Deformation field can be modified using render-to-3D-texture ("über-buffers")

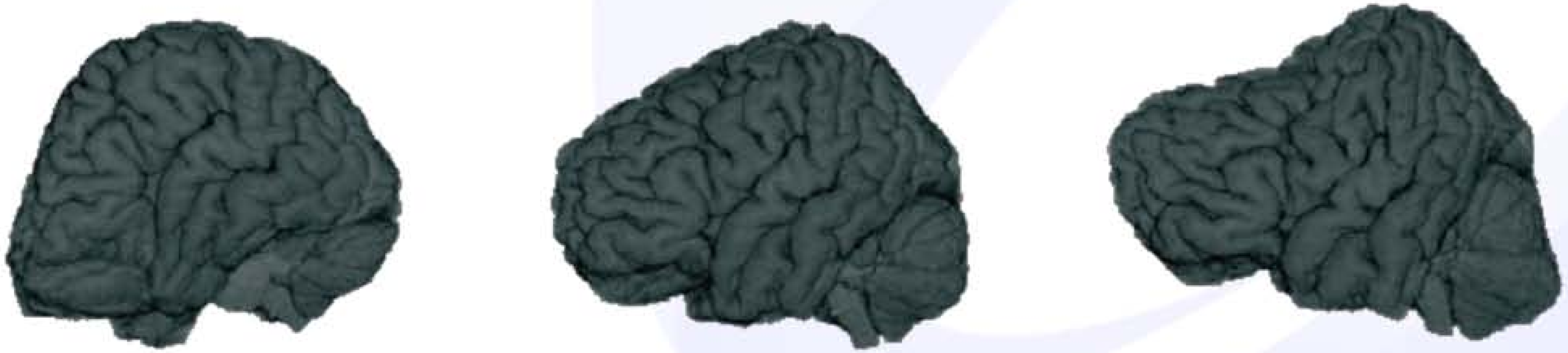
Offset Textures

```
// Cg fragment shader for
// texture-space volume deformation

  half4 main (float3 texcoords :  TEXCOORD0,
              uniform sampler3D offsetTexture,
              uniform sampler3D volumeTexture) : COLOR0
  {
     float3 offset = tex3D(offsetTexture, uvw);

     uvw = uvw + offset;

     return tex3D(volumeTexture, uvw);
  }
```



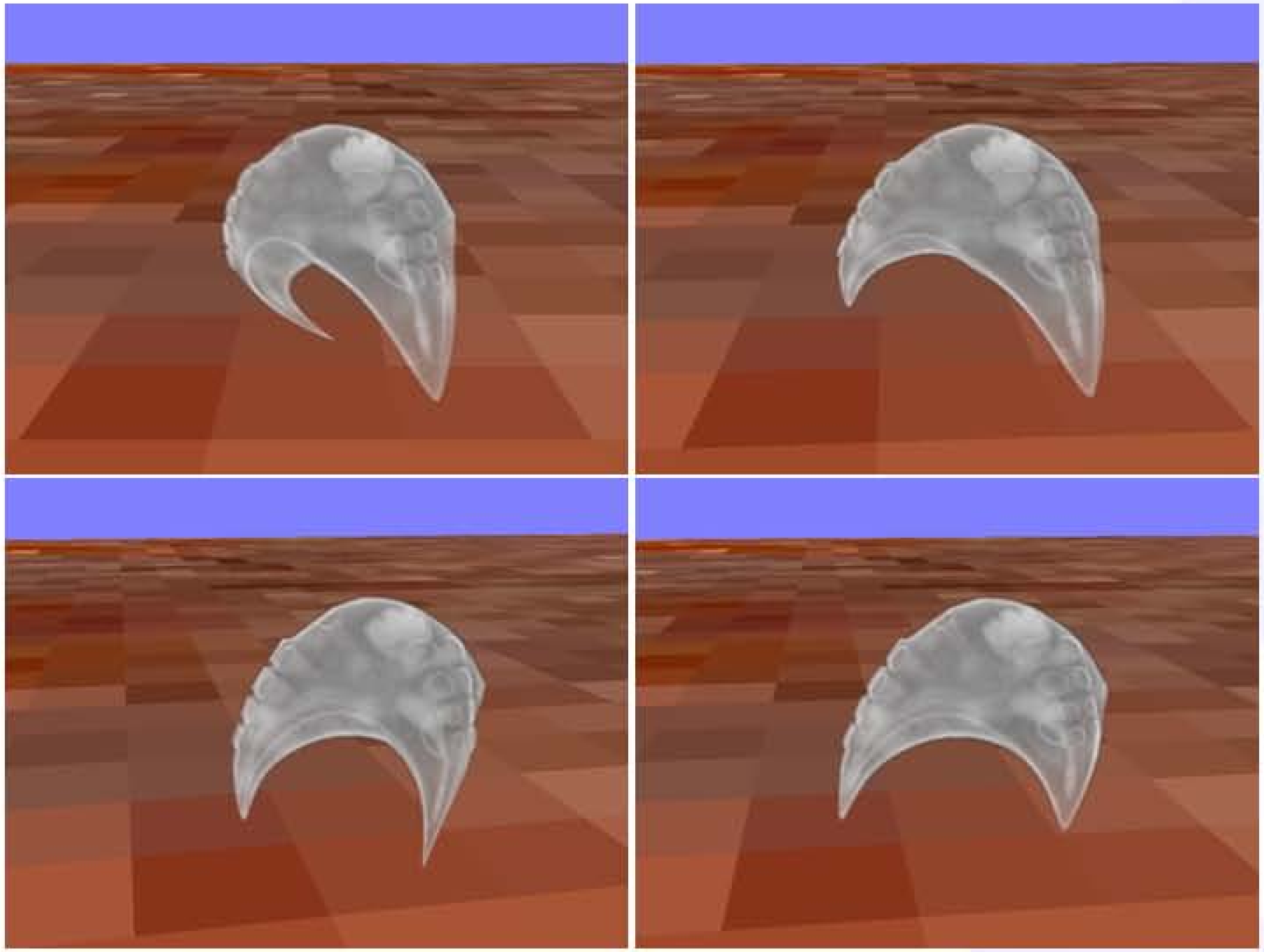
Volume Animation

- *Keyframe Animation/Blend Shapes:*
  - Easy with piecewise linear patches (simple vertex shader)
  - Offset textures: interpolate between different offset textures in fragment shader
- *Skeleton Animation:*
  - Use piecewise linear patches with matrix skinning in the vertex shader.
  - Dependent textures: Read the skin weights from 3D texture and caluclate offset in fragment shader.
- *Procedural Animation:*

  Calculate 3D offsets on-the-fly in the fragment shader

Texture Deformation

- Deformation field does not need to be stored in a texture
- Use procedural animation instead!



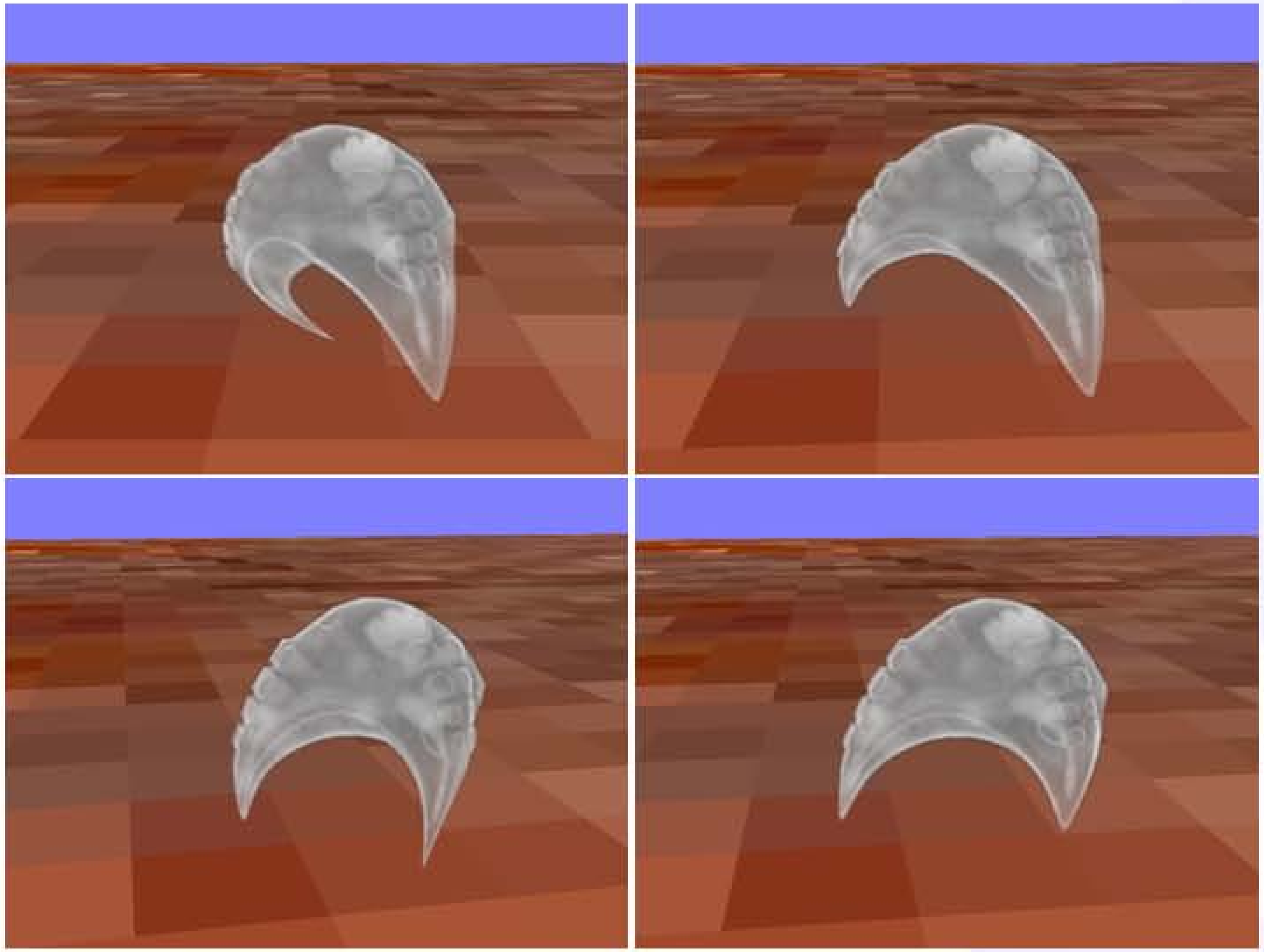
Example: Tripod Creature

Texture offsets parameterized in cylinder coordinates

Animation procedure moves 3 legs independently

Texture Deformation

- Deformation field does not need to be stored in a texture
- Use procedural animation instead!



```
#define PI (3.1415)

half modulo(half a, half b) {

    a -= floor(a/b)*b;
        if (a < 0) a+=b;
        return a;
}

half4 main( half3  uvw     : TEXCOORD0,
            uniform sampler3D volumeTexture,
            uniform half3 move1,
            uniform half3 move2,
            uniform half3 move3) : COLOR
{

        half3 P = uvw - half3(0.32,0.5,0.5);

        const half starangle = 2.0*PI/3.0;
        half angle = PI + atan2(P.z,P.x);
        half whichLeg = floor(angle/starangle);
        half A = modulo(angle, starangle)*3.0/2.0;|
        half weight = sin(A);

        half moveY = 1.2-uvw.y;
        moveY *= moveY;
        moveY *= moveY;

        weight *= moveY;

        if (whichLeg  < 1) {
                uvw -= move1 * weight;
        } else if (whichLeg  < 2) {
                uvw -= move2 * weight;
        } else {
                uvw -= move3 * weight;
        }

        half4 color = tex3D(volumeTexture,uvw);

        return half4(color);
}
```

Thanks

Thanks to
Mark Kilgard and *Nick Triantos*
from *NVidia* for providing the
GeForce 6800 demo machine

Thanks to my students:

Christoph Bastuck

Timo Hambürger