[12] Volume Modeling, Deformation and Animation
Modeling Volume Data

Surface Models + Voxelization

Measured Volumes

Procedural Models

Compositing
Rendering into a 3D texture

```c
#define GL_EXT_framebuffer_object

GLuint framebufferObject;
// create a frame buffer object
glGenFramebuffersEXT(1, &framebufferObject);
// create a 3D texture object
GLuint textureName;
glGenTextures(1, &textureName);
glBindTexture(GL_TEXTURE_3D, textureName);
glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA8,
            size_x, size_y, size_z,
            GL_RGBA, GL_UNSIGNED_BYTE, NULL);
// bind the frame buffer object
glBindFramebufferEXT(
            GL_FRAMEBUFFER_EXT, framebufferObject);

for(int z = 0; z < size_z; ++z) {
    // attach a z-slice to color target
    glFramebufferTexture3DEXT(
```
Rendering into a 3D texture

for(int z = 0; z < size_z; ++z) {
  // attach a z-slice to color target
  glFramebufferTexture3DEXT(
    GL_FRAMEBUFFER_EXT, // bind target
    GL_COLOR_ATTACHMENT0_EXT, // attachment point
    GL_TEXTURE_3D, // texture target
    textureName, // texture object
    0, // render target id
    z); // 3D texture slice

  // now render into the z slice
  renderIntoSlice(z);
} // for()

// unbind the frame buffer object
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
#endif // defined GL_EXT_framebuffer_object
Voxelization

- Creating a binary volume out of a polygonal mesh -
  - Approaches similar to clipping against arbitrary objects

polygonal mesh

$64^3$ voxels

$256^3$ voxels

$512^3$ voxels

Arbitrary Closed Polygonal Meshes with consistent vertex ordering

REAL-TIME VOLUME GRAPHICS
Christof Rezk Salama
Computer Graphics and Multimedia Group, University of Siegen, Germany

Eurographics 2006
Voxelization
Voxelization

Step 1:
Setup a clipping plane with same position and orientation as slice.
Voxelization

Step 1: Setup a clipping plane with same position and orientation as slice.
Voxelization

Step 1:
Setup a clipping plane with same position and orientation as slice.

Step 2:
Clear the slice with the background color.

Render the back faces of the polygonal mesh in foreground color.
Voxelization

Step 3:
Render the front faces of the polygonal mesh in background color.
Voxelization

Step 3:
Render the front faces of the polygonal mesh in background color.

Result:
The slice buffer now contains the correct cross-section in foreground color.

Repeat the whole process for the next slice image.
Procedural Volumes

- Volume data which is defined by a procedure instead of a texture

**Example:** Spectral Synthesis

\[ v(x) = \sum_{i=0}^{N} A_i \sin(f_i x + \varphi_i) \]

- Amplitude
- Frequency
- Phase Shift
Procedural Volumes

- Volume data which is defined by a procedure instead of a texture

**Example:** Spectral Synthesis

\[
v(x) = \sum_{i=0}^{N} A_i \sin(f_i x + \varphi_i)
\]

- Fractal Power Spectrum: Amplitudes of the individual waves are inversely proportional their frequency.
Procedural Volumes

```c
half4 main(half3 uvw : TEXCOORD0,
    uniform half3 phases[5],
    uniform half startStep,
    uniform half endStep) : COLOR
{
    float value = 0.0;
    float frequency = 3.0;
    float amplitude = 0.5;

    for(int i = 0; i < 5; ++i) {
        half3 phase = phases[i];

        value += amplitude *
            sin(frequency*uvw.x + phase.x) *
            sin(frequency*uvw.y + phase.y) *
            sin(frequency*uvw.z + phase.z);

        amplitude /= 2.0;
        frequency *= 2.0;
    }
}```
Procedural Volumes

```c
float frequency = 3.0;
float amplitude = 0.5;

for(int i = 0; i < 5; ++i) {
    half3 phase = phases[i];

    value += amplitude *
              sin(frequency*uvw.x + phase.x) *
              sin(frequency*uvw.y + phase.y) *
              sin(frequency*uvw.z + phase.z);

    amplitude /= 2.0;
    frequency *= 2.0;
}

value = abs(value);
float alpha = smoothstep(startStep,endStep,value);

return half4(value.rrr, alpha*alpha);
```
Procedural Volumes

Good for modeling clouds, fog and smoke

Phase shifts can be animated over time

Alternative: Use random noise instead of sine function

\[ \text{turbulence}(\mathbf{x}) = \sum_{i=0}^{N} A_i \cdot \text{noise}(f_i \cdot \mathbf{x}) \]
Volume Perturbation

Use procedural field to offset texture coordinates:

```cpp
half4 main(half3 uvw : TEXCOORD0,
            uniform half amplitude
            uniform sampler3D noiseTex,
            uniform sampler3D dataTexture) : COLOR
{
    // calculate the turbulence field
    half3 perturb = 0.0;
    perturb += 1.0  * tex3D(noiseTex, 2.0*uvw) - 0.5;
    perturb += 0.5  * tex3D(noiseTex, 4.0*uvw) - 0.25;
    perturb += 0.25 * tex3D(noiseTex, 8.0*uvw) - 0.125;
    perturb += 0.125 * tex3D(noiseTex, 16.0*uvw) - 0.0625;

    uvw += amplitude * perturb;

    return tex3D(dataTexture,uvw);
}
```
Volume Perturbation

Original Volume

Procedural Fur
Deformation and Animation

Deformable Volumetric Objects

Applications in Science

- Medicine
- Engineering
- Natural Science
Deformation and Animation

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

- Deformation of the Shape (Geometry) only
- Appearance (Materials, Textures etc.) remain 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
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 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:

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

Rotation and Scaling
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:

\[ \Phi(\vec{x}) = A\vec{x} + \vec{b} \]

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!)
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)
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?
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*

shifted texcoord.

ideal  bad  bad  ok
Piecewise Linear Patches

What do I need the inverse for?

- **Texture Interpolation**

  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:

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

simply negate the displacement vectors
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.
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
- Runs completely within GPU (except slicing)
- Deformation field can be modified using render-to-3D-texture
Offset Textures

```c
// 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 calculate 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!

```cpp
#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);
}
```