Wrinkling Coarse Meshes on the GPU

Jörn Loviscach
Hochschule Bremen
jlovisca@informatik.hs-bremen.de
Objective

- Plausible wrinkles and folds
- Fast: on coarse meshes
- Optional control by artist

Demo
Related Work

Cloth with less than 100% physics:

• Hadap et al., Vis 99: Bump maps invoked according to deformation
• Cutler et al., SCA 05: Wrinkle curves applied according to deformation
• Cordier et al., CGF 05: Learn mesh details from physics

... any many more
Basic Idea

- Per vertex:
  - Determine geometric compression
  - Compute plane wave
  - Iteratively align adjacent waves

- Per pixel: blend plane waves; deform normals and tex coords
Contributions

• Detailed wrinkles from arbitrary deformations; fast, robust, controllable, no precomputation

• Deformation of normals and texture from dynamic height fields

• Optional 3D paint-mode for wrinkle density and direction
Outline

• Overview of the method
• Mesh compression and wrinkle height
• Rendering
• Results
• Outlook
Overview of the Method

Four rendering passes (= pairs of vertex and pixel shaders)
Overview of the Method

- Original Position
- Unified Vertices
- Adjacency
- Color
- Profile
- Triangle Vertices
- Skin
- Position
- Crush
- Crush
- Normal
- Crush
- Relax
- Phase
- Render
- Screen

Apply deformation, e.g., skinning. Store new positions and normals.
Overview of the Method

Determine direction and strength of local compression
Overview of the Method

Align phases of local plane waves
Overview of the Method

Original Position
Unified Vertices
Adjacency
Color
Profile
Triangle Vertices
Skin
Position
Normal
Crush
Crush
Phase
Relax
Render
Screen
Render to screen
Determining the Compression

For every vertex:

- Linear approx. $M$ of local deformation
- Find direction and amount of strongest compression: eigenanalysis of $M^T M$

before deformation  after deformation
Painting Rest-Pose Wrinkles

Demo

Integrate user-defined wrinkles:

Bias the computation of the linear approximation $M$:

$$M \rightarrow M \cdot (1 - \mathbf{q} \otimes \mathbf{q}),$$

$q$ specifies direction and amount, is defined by 3D painting GUI.
Converting Compression to Height

Simple straight-line approximation: compression ratio $r$

$\rightarrow$ wrinkle amplitude $h$

Real-time control on width $W$

Demo
Compression Vector Field (1)

Result at every vertex:
• Tangent unit vector along maximum compression
• Wrinkle amplitude

Local plane waves in rest-pose space
Compression Vector Field (2)

Use $M^{-1}^T$ to convert the direction vectors to wave vectors in post-deformation space.

→ Waves are compressed with the mesh.
Generating the Height Field (1)

Idea:
Blend the linear waves across every triangle.

Ooops: Waves aren’t aligned.
Generating the Height Field (2)

Problem: The phases of the local plane waves are not yet determined.

Solution: Relax the phases gradually to diminish local misfit.
Rendering (1)

Render coarse polygons, fake fine-scale deformation.

Two issues to address:
• Deform texture
• Adjust normal vector

Demo
Rendering (2)

Texture deformation similar to Parallax Mapping

\[ \Delta x' = \left( \frac{v}{v \cdot n'} - n' \right) h(x') \]
Rendering (3)

Illumination: Compute normals of dynamic height field

Need the gradient of $h$. 

\[
\frac{\partial}{\partial x'} h(\pi_p(x'))
\]
Rendering (4)

Wrinkle profile determined by:

- height: cosine
- gradient: −sine

Replace each with a 1D texture lookup: arbitrary profiles

Demo
## Results

<table>
<thead>
<tr>
<th>Name</th>
<th># Vertices</th>
<th># Pixels (average)</th>
<th>fps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shirt</td>
<td>455</td>
<td>≈ 330.000</td>
<td>328</td>
</tr>
<tr>
<td>Zeppelin</td>
<td>508</td>
<td>≈ 260.000</td>
<td>540</td>
</tr>
<tr>
<td>Curtain</td>
<td>92</td>
<td>≈ 505.000</td>
<td>537</td>
</tr>
</tbody>
</table>

One pixel per vertex

### Stage

<table>
<thead>
<tr>
<th>Stage</th>
<th># Shader instructions</th>
<th>Contribution</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vertex</td>
<td>Pixel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skin</td>
<td>$13M + 20$</td>
<td>2</td>
<td>0.06 ms</td>
<td>0.25 ms</td>
</tr>
<tr>
<td>Crush</td>
<td>8</td>
<td>$28N + 102$</td>
<td>0.08 ms</td>
<td>1.83 ms</td>
</tr>
<tr>
<td>Relax</td>
<td>7</td>
<td>$33N + 23$</td>
<td>0.11 ms</td>
<td>1.75 ms</td>
</tr>
<tr>
<td>Render</td>
<td>67</td>
<td>47</td>
<td>3.11 ms</td>
<td>5.01 ms</td>
</tr>
</tbody>
</table>

Total time incl. non-shader part: 3.45 ms A, 8.90 ms B

$M = \#\text{bones used per vertex}; N = \#\text{neighbors}$

A: 1 Mpix, 100 verts; B: 55 kPix, 50 kVerts
Outlook

• Create curvature-aligned hatching with analytical filtering  Demo
• Fewer pseudo-textures with D3D 10: stream VS output to memory; access neighbor vertices in GS
• Diamond buckling (cf. this EG: “Virtual Garments”), tension wrinkles
• Real-time cracks (cf. Iben, SCA 06)
Thanks for your attention.

Questions?