Fast but Detailed Folds and Wrinkles on Graphics Hardware

Jörn Loviscach
Hochschule Bremen
jlovisca@informatik.hs-bremen.de
www.L7H.cn
Objective

- Plausible wrinkles and folds
- On coarse meshes
- Fast

Demo
Outline

● Short Intro to GPU Programming
● Approaches to Cloth Simulation
● Overview of the Method

● Preparing the Mesh
● Deforming the Mesh
● Determining the Fold Vector Field
● Generating the Height Field
● Rendering

● Results
● Outlook
Short Intro to GPU Programming

Programmability features in current graphics hardware:

• Vertex shader: deformation, pre-computations
• Pixel shader: texturing, bumps, etc.

Demo
Approaches to Cloth Simulation

Full Physical Dynamics
- Stunning results if done right
- High-dimensional stiff PDEs
- Non-robust collision detection

Kinematics
- Shape depends only on deformation
- Fast and plausible if done right
- Large folds difficult
- No temporal variation
Overview of the Method

• Input coarsely tessellated surface
• Deform using standard methods, e.g., matrix palette skinning
• Determine per vertex: strength and direction of local contraction
• Compute oscillating height field
• Render through pixel shader: lighting, texture deformation
Overview of the Method

All done in four rendering passes (= pairs of vertex and pixel shaders)
Preparing the Mesh

• Collect adjacency data in a pseudo-texture

• Unify vertices along texture seams

Demo
Deforming the Mesh

- Standard matrix palette skinning
- *.x file from standard 3D software: mesh, skeleton, bone weights, bone animation
- Matrix palette prepared by CPU
- Vertex Shader evaluates weighted sum
- Positions and normals stored in pseudo-textures for later use
Determining the Fold Vector Field

For every vertex:
- Linear approx. $M$ of local deformation
- Find direction and amount of strongest contraction, eigenanalysis of $M^T M$

Before deformation  after deformation

neighbor vertices

vertex
Determining the Fold Vector Field

How to produce folds for the rest pose?

Bias the computation of the linear approximation $M$:

$M \rightarrow M(E - q \otimes q)$,

where $q$ gives direction and amount.

User interface: 3D painting.

Demo
Determining the Fold Vector Field

Result:
A tangent vector at every vertex, direction = strongest contraction, length = amount of folding

Local plane waves!
Generating the Height Field

Height $h$ of fold depends on width of uncompressed fold.

Real-time control on $W$
Generating the Height Field

Form a plane wave around every vertex:

\[ h(x) = h_A \cos \left( \frac{2\pi}{W} k_A \cdot (x - x_A) + \phi_A \right) \]

Evaluate \( h \) with linear interpolation in post-deform space:

\[ h(x') = \alpha h_A \cos \left( \frac{2\pi}{W} k_A' \cdot (x' - x_A') + \phi_A \right) \]

\[ + \beta h_B \cos \left( \frac{2\pi}{W} k_B' \cdot (x' - x_B') + \phi_B \right) \]

\[ + \gamma h_C \cos \left( \frac{2\pi}{W} k_C' \cdot (x' - x_C') + \phi_C \right) \]
Generating the Height Field

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

Solution: Relax the phases to diminish local misfit.

This is no longer pure kinematics!

Demo
Rendering

Want to render coarse polygons and fake folds simply with coloring

Two issues to address:

• Texture. Deform the texture as though there was curvature.
• Lighting. Adjust the normal vector per pixel.

Demo
Rendering:
Texture Deformation

Principle similar to Parallax Mapping

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

Problem: bad artifacts in particular near the horizon

Solution (partially): Limit the deformation
Rendering:
Illumination

Bump map with dynamic (!) height field

Need the gradient of $h$. 
Rendering: Illumination

Built-in gradient command of HLSL produces blocks of 2x2 pixels.

Thus: Compute the gradient from the wave's equation.
Rendering: Arbitrary Fold Profiles

• height: cosine
  \[ h(x) = h_A \cos \left( \frac{2\pi}{W} k_A \cdot (x - x_A) + \phi_A \right) \]

• gradient: sine

Replace each with a 1D texture lookup: arbitrary profile.

Has to be symmetric, though.
(Fold vector field only determined up to sign!)

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>$\approx 330.000$</td>
<td>328</td>
</tr>
<tr>
<td>Zeppelin</td>
<td>508</td>
<td>$\approx 260.000$</td>
<td>540</td>
</tr>
<tr>
<td>Curtain</td>
<td>92</td>
<td>$\approx 505.000$</td>
<td>537</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage</th>
<th># Shader instructions</th>
<th>Contribution</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Vertex</td>
<td>Pixel</td>
<td>A</td>
<td>B</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>
<tr>
<td>Total time incl. non-shader part</td>
<td></td>
<td></td>
<td>3.45 ms</td>
<td>8.90 ms</td>
</tr>
</tbody>
</table>

$M = \#\text{Bones}; N = \#\text{Neighbors}$

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

• Introduce irregularities, e.g., through additional textures
• Combine with physical dynamics of coarse mesh
• Use geometry shaders (upcoming in DirectX 10) to access neighbor vertices
Thanks for your attention!

Questions?