Interactive Cloth Simulation

Matthias Wloka
NVIDIA Corporation
MWloka@nvidia.com
Overview

- Higher-order surfaces
- Vertex-shader deformations
- Lighting modes
  - Per-vertex diffuse
  - Per-pixel diffuse with bump-map
  - Minnaert lighting (velvet)
  - Blinn-bump reflection
  - Reflection and refraction
CPU Work

- CPU computes cloth-equations on a control-net
  - Collision detection and resolution with sphere
  - Vertex shader cannot access control-net points
- Control-net defines Bspline- (or Bezier-) patches
  - GPU generates vertices from the control-net
- Advantages:
  - CPU processes only control-net points
  - Bandwidth savings (cache/main-memory, AGP)
  - Dynamic LODing is simplified
  - Optimal way to draw triangles
Control-Net vs. Rendered Vertices

Control-net:
13 x 10 = 130 vertices

Rendered vertices:
(10*13) x (7*13) = 11,830 vertices
Collision Detection and Resolution

- Control-net collisions are low resolution
  - Possible interpenetration in center of patch
- Vertex shaders resolve interpenetration
  - Compute distance of vertex to sphere-center
  - If greater than radius, do nothing
  - If less than radius, displace vertex along distance-vector by (radius – distance)
Vertex Shader Collision Resolution

Vertex a

|a−c|^2 > r^2

Vertex b

|b−c|^2 < r^2

b’ = c + r * ||b−c||
Vertex Shader Deformation
Per-Pixel Lighting

- Higher-order surfaces module generates per-vertex normal and tangent

- Vertex shader computes
  - Per-vertex bi-normal = normal x tangent
  - Model- to texture-space matrix
    Normal.x Tangent.x Bi-Normal.x
    Normal.y Tangent.y Bi-Normal.y
    Normal.z Tangent.z Bi-Normal.z
  - Transforms light-vector to texture space
  - Send transformed light-vector to pixel shader
Per-Pixel Lighting (cont.)

- Pixel shader fetches
  - Per-pixel normal from bump-map
  - Per-pixel color from texture
  - Per-pixel light-vector from vertex shader
    - Renormalize via normalization cube-map

- And computes
  - Diffuse (N dot L)
  - Modulate with light- and material-color
Per-Pixel Lighting Screenshot
Minnaert Lighting

- Custom per-pixel lighting model that simulates velvet
  - \( F(N \cdot L, N \cdot E) \cdot (N \cdot L) \cdot \text{material\_color} \)
  - \( F(x, y) \) is custom (generated) texture:

- Vertex shader set-up:
  - Compute model-space normal, light, and eye-vector
  - Compute texture space light vector
  - Copy material color and uv-coords (for bump-map)
Minnaert Lighting Screenshot
Blinn-Bump Reflection

- **Vertex shader**
  - Compute and store normal, tangent, and bi-normal
  - And store eye-vector in the w-components

- **Pixel-shader**
  - Looks up normal
  - And lets the hardware do
    - Texture-space to model-space transform
    - Reflection vector computation
    - Environment cube-map look-up
Blinn Bump Reflection Screenshot
Sphere Reflection

- Environment-map does not contain sphere
  - Sphere casts no reflection

- Render sphere into environment-map
  - Environment-map works because environment is conceptually far away
  - Origin of reflection vector is insignificant
  - For close objects, location of reflection does matter (multiple reflections)
Sphere Reflection Problem

- Environment is rendered from some viewpoint V
  - The further away objects are, the more V can jitter without noticeable difference
  - Moving in-between A, B, C, V is very noticeable
Sphere Reflection Solution

- Ray-trace per vertex in vertex shader!
  - Compute eye-reflection vector
  - Test ray-sphere intersection
  - Compute sphere-intersection point and normal
  - Light intersection point

- Store sphere-color blend factor in alpha
  - 0 if no intersection
  - Attenuate by intersection distance
  - Attenuate by proximity to rim
Ray Tracing in the Vertex Shader

1. Compute vector from eye to vertex
2. Reflect that vector
3. Perform ray-sphere intersection; if hit, compute intersection point
4. Compute vector from intersection point to light; light via n’ dot l
Sphere Reflection Close-Up
Blinn-Bump Reflection, 2nd Pass

- First pass
  - Blend between per-pixel reflection and sphere ray-trace

- Second pass
  - Compute specular
  - Modulate texture color with texture alpha
  - Alpha blend sum of above with 1st pass result
Reflection and Refraction

- No per-pixel normal

- Vertex shader computes
  - Reflection vector \( R = -(E - 2*(E \cdot N) \times N) \)
  - Refraction vector
    - Approximate via shortened normal \( N' = s \times N \)
    - \( R' = -(E - s^2 \times 2*(E \cdot N) \times N) \)
  - Specular
Reflection and Refraction (cont.)

- Pixel shader computes
  - Environmental cube-map look-up for reflection and refraction vector
  - Blend reflection-result and sphere ray-trace
  - Blend result with refraction
  - Add specular contribution
Reflection and Refraction Screenshot
Vertex/Pixel Shader Debugger

• It’s a debugger
  • Step
  • Break
  • Examine register values

• [link](http://www.nvidia.com/developer)
NVEffectsBrowser

- Vertex- and pixel-shader framework to:
  - Quickly prototype
  - Experiment
  - Demo DirectX8 effects

- Nvidia’s Vertex and Pixel Shader Contest
  - Submit your shaders to NVidia
  - Fabulous prizes
  - See http://www.nvidia.com/developer for details
Vertex Shader Uses

- Custom texture coordinate generation goes a long way
  - Various materials
  - Reflection and refraction
  - Fogging effects
  - Discretization

- Because texture look-up encodes an arbitrary function $t = f(u, v, w)$

- Pixel shader set-up
Vertex Shader Uses (2)

- Deformations
  - Lens-distortions
  - Distort along normals (halo/silhouette rendering)
  - Local/global deformations
  - Waves/ripples

- Data “generation”
  - Compute normals on the fly (deformed geometry)
  - Compute position/color/etc. from time
Vertex Shader Uses (3)

- Multi-transforms
  - Matrix-palette skinning
  - Morphing
  - Motion blur

- Silhouettes and tangent/bi-normal
  - Art-based rendering (NPR rendering)
  - Volumes (shadow volume)
Pixel Shader Uses

- Per-pixel custom lighting
  - Per-pixel normals, potentially animated
  - Area is wide open

- Access to pixel-neighbors via four texture units
  - Limited image processing
    - Blurred shadows
    - Depth of Field
    - Color-space conversions
    - Edge-detection
  - Procedural textures
Questions…

Matthias Wloka
MWloka@nvidia.com
www.nvidia.com/Developer