Displacement Mapping

Tom Forsyth
Muckyfoot Productions

Michael Doggett
ATI Research, Inc.
Outline

- DirectX9
  - Pre Sampled
  - Filtered
- OpenGL
  - überBuffers
- Real Time Algorithms
  - Summed area tables
  - Interval analysis
Displacement Mapping

\[ P'(u,v) = P(u,v) + d(u,v)N(u,v) \]
Uniform and Adaptive
Advantages

- Geometry Compression
- Fixes errors with Bump Mapping
  - Silhouettes and Occlusion
- Level of detail control
  - Tessellation level has associated displacement map/mip level
- VPU generated surfaces
DirectX 9

- Add texture maps to the vertex shader
  - New sampler stage before vertex shader
  - D3DDMAPSAMPLER
DirectX 9

- Base surface always NPatches
  - Can set the order
- Multiple formats supported
  - 1-4 channels
- No anisotropic filter
- Two types
  - Pre Sampled
  - Filtered
DirectX 9 – Pre Sampled

- Set up array of displacements at different tessellation levels
- Displacement map texture set with
  - `SetTexture(D3DDMAPSAMPLER, dmap_levels[curTessLevel]);`
- NPatch tessellation level
  - `SetNPatchMode(curTessLevel);`
DirectX 9 – Pre Sampled

- Displacement map is 1D array of filtered samples
  - Point sampled
  - Filtering settings ignored
    - \( U = \frac{\text{Index} \mod \text{Texture Width InPixels}}{\text{Texture Width InPixels}} \)
    - \( V = \frac{\text{Index}}{\text{Texture Width InPixels}} / \frac{\text{Texture Height InPixels}}{\text{Texture Height InPixels}} \)

- Sequential index of generated vertices
  - Output one float
    - range -1.0 to 1.0
  - Match tessellator order
DirectX 9 - Presampled

- Order of generated vertices for NPatches
DirectX 9 – Pre Sampled Vertex Shader

vs.1.1

// c0-c3 - View+Projection matrix
dcl_position v0
dcl_normal v1
dcl_texcoord0 v2
dcl_sample v3

// Decompress position into r0
.
.
.
mov r3.x, v3.x // Displacement value
mul r1.xyz, v1, r3.x // (nx, ny, nz) * DisplacementValue
add r0.xyz, r1.xyz, r0 // V + N*d

// Transform position into oPos
dp4 oPos.x, r0, c0
.
.

mov oD0, v1 // Output Normal
mov oT0, v2 // Output Tex Coord
DirectX 9 - Filtered

- Mip map chains of displacement map
- Trilinear filtering applied between levels of detail
OpenGL Displacement Mapping

- überBuffer extension
  - "generalized method for allowing memory buffers to be used for many purposes"
- Render to Vertex Arrays
- OpenGL SuperBuffer working group started
- überBuffers available soon
- RADEON 9500/9700 class hardware
  - Fragment shader
OpenGL Displacement Mapping

- 2 Pass algorithm
  - 1st pass
    - Displacement fragment shader
    - Output to Vertex, Normal, TexCoord Buffer
  - 2nd pass
    - Vertex shader uses displaced Vertex Buffer
    - Fragment shader does bump mapped lighting
Real Time Algorithms

- CPU based
  - Possible VPU implementations
- Rasterization algorithms
  - Gumhold, Hüttner Graphics Hardware 1999
  - Doggett, Kugler, Strasser Computer Graphics Forum 2001
Recursive Adaptive Tessellation

- Edge based tessellation
  - Use only edge local information
  - Insert midpoint
  - Ensures no cracks
Displacement Mapping Using Summed Area Tables

- Doggett, Hirche, Graphics Hardware 2000

4 edge tests
1. Local area average height test
   - Summed Area Table (SAT)
   - Each (X,Y) entry is summation of entries contained in the rectangle with corners (0,0) and (x,y)
2. Surface normal variance test
3. View dependent test
4. Refinement limit test
for (int nEdge=0; nEdge<3; nEdge++)
{
    edge.endpoints( tri.vertex[nEdge], tri.vertex[(nEdge+1) % 3] );
    // Local area average height test
    endP0avH = dmap.SATavHeight(edge.endpoint0.tex, edge.height/2, edge.length/2);
    endP1avH = dmap.SATavHeight(edge.endpoint1.tex, edge.height/2, edge.length/2);
    midPavH = dmap.SATavHeight(edge.midpoint.tex, edge.height/2, edge.length/2);
    SummedHeightTest = (midPavH/2 - (endP0avH + endP1avH)) < SummedHeightThres;
    // Surface normal variance test
    NormalTest = ((edge.endpoint0.normal - edge.midpoint.normal) < NormalThres) |
                ((edge.endpoint1.normal - edge.midpoint.normal) < NormalThres);
    // View dependent test
    ViewTest = edge.lengthInScreenSpace() < ViewThres;
    // Refinement limit test
    CoordTest = (int) edge.endpoint0.tex == (int) edge.midpoint.tex |
                (int) edge.endpoint1.tex == (int) edge.midpoint.tex ;
    // insert midpoint
    insertMidpoint[nEdge] = (SummedHeightTest | NormalTest) & ViewTest & CoordTest;
}
SAT DMap – Combined Effect

Effective Area Test
Ineffective Normal Test

Effective Area Test
Ineffective Normal Test

Area boundaries
SAT DMap – View dependent results
Interval Analysis

- Moule, McCool GI02
- Generate mipmap chains
- Create Interval hierarchy
  - Upper bound
  - Lower bound
  - 8 bits instead of 16
Interval Analysis

- Insertion decision based on min and max displacement along edge
- Take interval union that covers edge
  - Four possible edge cases
Interval Analysis

- Union of the interval that covers the three edges covers the triangle
  - Interior of the triangle is covered
Interval Analysis

- Edge split if interval greater than user-defined threshold
- Optimize by testing multiple levels
- View dependent tessellation control similar to Summed Area Tables DMap
Looking Forward

- Generic texture fetch in vertex shader
- DirectX 9 Vertex Shader 3.0
  - `tex1dl dst, src0, src1`
- Primitive Shader
  - Generates vertex stream
  - Per-surface operations
  - Primitive computation frequency
    - Stanford shading language [Proudfoot et. al. SIGGRAPH01]
Summary

- Displacement mapping techniques
  - DirectX 9 Pre Sampled and Filtered
  - überBuffers OpenGL extension
  - Software based recursive adaptive tessellation
    - Hardware Friendly

- Displacement mapping is the next step beyond bump mapping towards visual realism
Questions

- Acknowledgements
  - ATI: Arcot Preetham, Rob Mace, Jeff Royle, Evan Hart, Jason Mitchell
  - Kevin Moule, Michael McCool
Practical Displacement Mapping

Tom Forsyth
Muckyfoot Productions
TomF@Muckyfoot.com
Displacement Maps

• Lower memory cost and bandwidth
• Closely related to normal maps
  • Often get one “free” with the other
• Mesh data becomes mainly 2D, not 3D
• Scalable – author once, use many
• Plenty of existing hardware support
• Advanced hardware covered by Doggett
Required Data

- Low-poly “base” mesh
- Unique UV mapping over base mesh
- Heightfield disp.map
- Normal (dot3) map
- Can be authored, or generated using tools
- Most tools apply to both normal & disp.maps
How To Get: Low-Poly Mesh

- Hand-authored
- Generate by simplifying high-poly mesh
  - Edge-collapse methods (mainly)
  - Generates Progressive Mesh sequence as well
- Lots of existing tools such as:
  - D3DXPMesh interface
  - nVidia’s “Melody”
  - Charles Bloom’s “Galaxy”
  - Mine! (Game Programming Gems 2 VIPM article)
How To Get: Unique UVs

- Ensures each texel used no more than once
- Allows object-space normal maps
- Used for other things e.g. lightmaps
- Hand-authored
  - With automated packing
- Generate from base mesh
  - Most art packages (Max, Maya, XSI)
  - Melody, Galaxy
How To Get: Disp & Normal Map

- Hard to create disp.maps manually
  - 8bpp not accurate enough
  - 16bpp tools still primitive
  - Fine control over shape difficult
- Generate by raycasting from base mesh
  - ATI Normal Mapper
- Normal map extracted at the same time
- Modified by surface-detail bumpmaps
Animation

- Base mesh is animated, then displaced
- Base mesh is simple – animate on CPU!
- Allows more animation flexibility
  - Physics simulation (e.g. cloth)
  - Morph targets
  - Vertex code doesn’t care about details
- Use curved basis (N-patch) for animation
  - Or just use more base verts!
Lighting

- Use dot3 normal maps
  - Object-space or tangent-space
- Light vector interpolated over base mesh
- Tangent-space transform done on base mesh
  - Same idea as animation – CPU does the work
  - Tangent vectors never sent to hardware
  - Flexible - whatever sort of light you like
- PS2 is a bit tricky…
Shadows

- Real geometry
- Shadow buffers “Just Work”
- Stencil volume shadows
  - Send to infinity if \( L \cdot N < 0 \)
  - By definition, mesh is well-tessellated
    - No need for degenerates
  - Requires normal per tessellated vertex
    - But you don’t need texture coords
Rendering: Current Hardware

- Displacement Compression
  - Uses VS1.1, VU1, SSE, etc – ubiquitous!
- Base mesh uploaded to vertex unit
- Progressive mesh - vertices only 8 bytes:
  - 3 indicies to base mesh vertices
  - I,J barycentric coordinates lerp between them
  - Displacement along interpolated normal
  - UV corrections – usually small
Rendering: Current Hardware

Linear interpolation

\[ B = I \cdot P_1 + J \cdot P_2 + K \cdot P_3 \]

\[ I + J + K = 1 \]
Rendering: Current Hardware

Linear interpolation
+displacement

\[ P = B + \text{disp} \times N \]
Rendering: Current Hardware

• Disp.comp. demo
  • VS1.1 shader – 30 lines
  • 260 base vertices – more than needed
    • In two chunks – head and body
  • No cheating or caching of data
  • 13k tris in 100kb
    • Over half of that is index data!
  • Animation is tiny amount of extra load
Rendering: New Hardware

- Adaptive tessellation – “Just Works”
- Presampled displacement maps
  - Sample disp.map according to rules + mipmap
  - Each mipmap level has two channels
    - Second one is next-smaller mipmap, but up-sampled
  - Lerp between two displacements in vertex unit
- Allows continuous smooth detail changes
  - Change tessellation levels when fully lerped
Muckyfoot Choices 1

- Artists make medium-poly models
  - ~3k polys for a human
  - Allows good shape control
- VIPM to get low-poly mesh (~200 verts)
- Bumpmaps for fine details
  - More efficient than fiddly polys
  - Friendlier to raycaster
Muckyfoot Choices 2

• Unique UVs
  • People, cars, small objects – hand generated
  • Buildings – automated (also for lightmaps)
• Raycast to product disp & normal maps
• Scale UV parts by texel frequency
• Pack & re-raycast to use texels efficiently
• Tessellate higher for VIPM+disp.comp
Conclusions

• Disp.maps are fast & low memory
• Same art pipeline as for normal maps
  • Disp.maps are no extra work
• Existing hardware support everywhere
  • Including consoles and old PCs
• New hardware even faster
• Easily scalable – use on all platforms
Questions?

• Do you want more details on:
  • Disp.comp
  • VIPM
  • Lighting
  • Tools
  • MF’s choices
The Last Slide

- Tools to investigate:
  - nVidia’s “Melody” – VIPM, UVs, raycaster
  - Charles Bloom’s “Galaxy” – ditto
  - ATi’s “Normal Mapper” – raycaster
  - Crytek – raycaster
  - My GPG2 article – VIPM generator
  - D3DXPMesh lib – VIPM generator
  - Max/Maya/XSI “unfold” unique UV mappers
- TomF @ Muckyfoot.com – happy to chat