Silhouette Maps for Improved Texture Magnification

Pradeep Sen
Stanford University

August 30, 2004

SIGGRAPH/Eurographics Workshop on Graphics Hardware 2004
Grenoble, France
A history of texture magnification

Wolfenstein3D by id (1992)
A history of texture magnification

Duke Nukem 3D by 3D Realms (1996)
A history of texture magnification

1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004

Grunt: This could be very difficult, Shen! Akahori, Daenia almost dodged Visor’s rocket

Quake III Arena by id (1999)
A history of texture magnification

A history of texture magnification

1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004

Doom3 by id (2004)
Contributions

- Applied the silhouette map algorithm to address artifacts from magnification of textures.
- Added filtering to silhouette map algorithm to improve magnification of natural textures.
- Implemented entire algorithm on graphics hardware, running in real-time.
Previous work in texture magnification

- SGI GL_SGIS_sharpen_texture extension
- Detail maps
- Procedural textures
Previous work in texture magnification

- SGI GL_SGIS_sharpen_texture extension
- Detail maps
- Procedural textures

Source: http://www.targetware.net/devguide/terrain/3_detail.html
Previous work in texture magnification

- SGI GL_SGIS_sharpen_texture extension
- Detail maps
- Procedural textures

ATI X800 Real-time demo “The Doublecross”

Final splash screen procedurally generated
Concurrent work

Presented at Rendering Workshop ’04:

- Ramanarayan 2004 – Feature-based textures
- Tumblin 2004 - Bixels
Silhouette map texture algorithm

- Standard texture
- Silhouette map
- Deformed texture
The silhouette map

- Introduced in SIGGRAPH ’03 to address aliasing in shadow maps.

- Silhouette map stores the xy-coordinates of points that lie on the discontinuity edges.

- The sample values are on the corners of the silhouette map texels.

- Every cell in the silhouette map has one and only one point.
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:

Silhouette map texture
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:
- Project point into silmap cell
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:
- Project point into silmap cell
- Fetch silmap pt and neighbors
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:
- Project point into silmap cell
- Fetch silmap pt and neighbors
- Fetch corner colors
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:
- Project point into silmap cell
- Fetch silmap pt and neighbors
- Fetch corner colors
- Use sample of correct quadrant
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:
- Project point into silmap cell
- Fetch silmap pt and neighbors
- Fetch corner colors
- Use sample of correct quadrant
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:
- Project point into silmap cell
- Fetch silmap pt and neighbors
- Fetch corner colors
- Use sample of correct quadrant

Silhouette map texture
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:
- Project point into silmap cell
- Fetch silmap pt and neighbors
- Fetch corner colors
- Use sample of correct quadrant

Silhouette map texture
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:
- Project point into silmap cell
- Fetch silmap pt and neighbors
- Fetch corner colors
- Use sample of correct quadrant
Silhouette map texture algorithm

During creation:
- Silmap identifies points on discontinuities
- Color samples are offset from silmap grid

During rendering:
- Project point into silmap cell
- Fetch silmap pt and neighbors
- Fetch corner colors
- Use sample of correct quadrant

Silhouette map texture
Initial results

Converted to 32 x 32 pixel texture
Initial results

128x128 pixel texture
Standard texture bilinearly interpolated
Silhouette map textures
Controlling shaders

Example of using a texture to modulate procedural shader

Hi-res artwork → 64x64 pixel texture

Standard texture
Interpolation and thresholding
Silhouette map textures

(Texture courtesy Anup Lobo)
So far so good…

- Silhouette maps greatly improve the magnification of textures with constant color, like signs and vector artwork.

- However, how does the algorithm fare for more complex textures?
Problem with “natural” images

- Convert to 64x64 textures
- Magnify using silhouette map
- Pixelated appearance!
- Need to filter samples while respecting boundaries
Filtering

In order to avoid this pixelated appearance, we must perform bilinear interpolation only inside of continuous regions in the texture.
To define these “regions” we’re going to need extra info in the silhouette map.

- Imagine a texture with regions in blue and green.
Additional info needed in silmap

To define these “regions” we’re going to need extra info in the silhouette map.

- Imagine a texture with regions in blue and green.
- The standard silmap only stores points at the discontinuities.
To define these “regions” we’re going to need extra info in the silhouette map.

- Imagine a texture with regions in blue and green.
- The standard silmap only stores points at the discontinuities.
To define these “regions” we’re going to need extra info in the silhouette map.

- Imagine a texture with regions in blue and green.
- The standard silmap only stores points at the discontinuities.
- Unfortunately, these points are not enough to reconstruct the discontinuity during rendering.
To define these “regions” we’re going to need extra info in the silhouette map.

- The solution is to embed the edges into the silhouette map.
To define these “regions” we’re going to need extra info in the silhouette map.

- The solution is to embed the edges into the silhouette map.
- This can be done by adding two booleans to each cell to specify whether or not an edge exists.
To define these “regions” we’re going to need extra info in the silhouette map.

- The solution is to embed the edges into the silhouette map.
- This can be done by adding two booleans to each cell to specify whether or not an edge exists.
- The other edges can be determined from neighboring cells.
Goals for filtering

The filtering we implement must be fast and execute on graphics hardware:

- Use only a small, localized reconstruction kernel that can be accessed quickly.
- Assume a bilinear interpolation as the filter basis because it is available in hardware.
Six different interpolation cases
Bilinear interpolation as filter basis

- Four corner case is trivial

\( (x, y) \)
Bilinear interpolation as filter basis

- Four corner case is trivial
- One corner case
Bilinear interpolation as filter basis

- Four corner case is trivial
- One corner case
- Two corner case

\[(x,y)\]
Bilinear interpolation as filter basis

- Four corner case is trivial
- One corner case
- Two corner case

Three corner case tricky…
3 corner interpolation

\[ f(x,y) = (1 - x - y)C + xB + yD \]

We note:

\[ g(x,0) = (1 - x)C + xB \]
\[ g(0,y) = (1 - y)C + yD \]

\[ g(x,0) + g(0,y) = 2C - xC - yC + xB + yD \]

So:

\[ g(x,0) + g(0,y) - g(0,0) = (1 - x - y)C + xB + yD \]
Implementation of Filtering

Thus, we can represent any case using the bilinear interpolation function available on hardware.

Because we have so many cases, implementation is tricky. Refer to paper...
Results of filtering

Original texture

Nearest neighbor silmap

Filtered silmap
Minifying silhouette map textures

- Silhouette maps can introduce high-frequency elements to textures and must be properly filtered when minified.

- Fortunately, mipmaps are good at doing this already!

- Our solution is to mipmap a pre-filtered bitmap and then use the mipmap hardware to blend between the silhouette map and the mipmapped version seamlessly.
Demos
Creation of silmap textures

Two ways of creating silhouette map textures:

- Using an editor to manually embed the edges
- Using edge-detection algorithms to find silhouette points automatically
Silmap samples from hi-res texture

When generating color samples from hi-res textures, do not filter across discontinuities!

Hi-res 1024x1024  →  Silmap 64x64  →  Properly filtered
Silmap samples from hi-res texture

When generating color samples from hi-res textures, do not filter across discontinuities!

Hi-res 1024x1024 → Silmap 64x64

Incorrectly filtered
Cost of silhouette maps

- In the current implementation we use a float4 texture to store each point plus edge data. This is overkill!

- If bit-ops are available, we can pack it into a single byte:
  - Three bits for the x and y values of the silhouette point, addressing 8x8 positions per cell.
  - Two bits for the extra edge information

- An extra byte per cell adds 33% to a 24-bit RGB texture
Benefits of silhouette map textures

- Space-efficient representation
- Constant-time lookup and bounded complexity
- Piecewise linear approximation by deforming the underlying mesh
- Easy to create content
Silhouette map hardware

- The silhouette map algorithm might be implemented in hardware, exposed to the user as a texture fetch command.

- This would allow these examples to be implemented without a single fragment program being written!
Comparison: Feature-based textures

- FBT’s encode texture features by dividing cells into different regions using spline curves.
- Like silhouette map textures, they address the artifacts that occur in the magnification of textures.
- Complicated spline intersection algorithm would be difficult to implement in real-time on graphics hardware.

Ramanarayanan, Bala, Walter – “Feature-Based Textures”
Comparison: Bixels

- Bixels encode discontinuities by identifying points in cells, and are essentially equivalent to silhouette maps but with a different name.

- Bixels use a more complex reconstruction kernel which is less localized than the algorithm proposed.

- Bixels are not implemented on graphics hardware and do not run in real-time.

- Application targets image compression – no texture mapping examples shown.

  Tumblin, Choudhury – “Bixels: Picture samples with sharp embedded boundaries”
Why does it look like a cartoon?
Future work

- Shadow silhouette maps store scalar depth values at the corner samples, our silhouette map textures store RGB values.

- One can imagine storing other coefficients to yield better texture reconstruction and allow the “mid-range” frequencies to be captured more faithfully.

- This might be compared with more general image compression algorithms.
Conclusions

- We have extended the shadow silhouette map algorithm to handle more general textures.
- We apply a discontinuity-preserving filter kernel to complex textures.
- This technique is fast and can be used anywhere textures are normally used: texture mapping, alpha billboarding, shader control.
Acknowledgements

- Mike Cammarano and Pat Hanrahan for valuable discussion during the development of the idea
- Tim Foley, Mike Cammarano, Jeff Klingner, Kayvon Fatahalian for talk prep
- NVIDIA and ATI for hardware/software support
- md2 demo engine written by Mark Kilgurd, NVIDIA
- Work funded by research grant from ATI, NVIDIA, and SONY
Questions?