Interactive Summed-Area Table Generation for Glossy Environmental Reflections

Justin Hensley*
Thorsten Scheuermann†
Montek Singh*
Anselmo Lastra*

*University of North Carolina at Chapel Hill
†ATI Research
Overview

• Summed-area tables
  – Useful for averaging pixels
  – Efficient creation on GPU

• Rendering dynamic reflections with per-pixel glossiness using dual-paraboloid maps and summed-area tables
Summed-Area Tables (SATs)

- Each element $S_{mn}$ of a summed-area table $S$ contains the sum of all elements above and to the left of the original table/texture $T$ (for a left handed coordinate system) [Crow84]

$$S_{mn} = \sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij}$$
Summed-Area Tables (SATs)

- Each element $S_{mn}$ of a summed-area table $S$ contains the sum of all elements above and to the left of the original table/texture $T$ (for a left handed coordinate system) [Crow84]
Summed-Area Tables (SATs)

- Each element $S_{mn}$ of a summed-area table $S$ contains the sum of all elements above and to the left of the original table/texture $T$ (for a left handed coordinate system) [Crow84]

```
1  2  3  4
4  0  7  2  4
3  1  4  1  2
2  6  1  2  0
1  0  3  5  2
```

```
1  2  3  4
4  0  7  9  13
3  1  12  15  21
2  7  19  24  30
1  7  22  32  40
```

`input texture`
Summed-Area Tables (SATs)

- Each element $S_{mn}$ of a summed-area table $S$ contains the sum of all elements above and to the left of the original table/texture $T$ (for a left handed coordinate system) [Crow84]
Using a Summed-Area Table

- Summed-area tables enable the averaging rectangular regions of pixels with a constant number of reads.

\[ \text{average} = \frac{LR - LL - UR + UL}{\text{width} \times \text{height}} \]
Efficient Summed-Area Table Creation

- Borrow technique from high performance computing - recursive doubling
- Summed-area table construction can be decomposed into horizontal and vertical phase each with $\log_2(\text{texture size})$ passes
- Each pass adds two elements from previous pass.

Horizontal Phase:

$$P_i(x, y) = P_{i-1}(x, y) + P_{i-1}(x - 2^{\text{passindex}}, y)$$
Horizontal Phase

Sampling off texture returns 0 and does not affect sum
Boundary Conditions

- To make sure sampling off the texture does not affect the results we need to set up the correct texture clamping behavior
- Two possibilities:
  - Clamp to border color with a color of (0, 0, 0, 0)
  - Render a black border around the texture to be converted into SAT and set Clamp to Edge mode
Saving unnecessary texture reads

• Reads off of the texture are wasteful
  – Texel cache *should* catch these reads

• Optimization:
  – Do not perform computation for *finished* texels
  – Reduce the size of the rendered quad each pass to only cover texels have not finished computation
Saving Render Passes

- Two samples per pass requires 16 passes for a 256x256 texture $\rightarrow 2^{\log_2(256)}$

- Reduce number of passes by adding more samples per pass
  - passes $= 2^{\log \#samples}(texture\ size)$

- Only need 4 passes to convert a 256x256 texture into a summed-area table if 16 samples / pass used
  - Trade extra work versus reducing context switches
Precision Requirements

• For proper reconstruction:
  
  \[
  \text{table precision} = \log_2(w) + \log_2(h) + b
  \]

• A 256x256 8-bit input texture requires 24-bits of precision per component

• Use 32-bit floats to compute and store summed-area tables

• Precision errors average out as you use larger box filter kernels
Effects of Precision Loss

Input texture

16-bit float

1x1 box filter

3x3 box filter

32-bit float
Effects of Precision Loss
24-bit floats

Input texture

1x1 box filter reconstruction
Improving Precision Requirements (1/2)

- Summed-area tables store a positively increasing monotonic function
  - Construction requires the addition of a value that is at least zero

- **Construct table using offsets instead of absolute values**
  - Function no longer monotonic
  - Removes DC component of signal
Improving Precision Requirements (2/2)

- Bias input texture by -0.5 before generating table
- Reconstruct samples from table by adding 0.5 to final result
  - For best results, use actual image mean
- Particularly useful on hardware with limited pixel pipeline precision
Offset Summed-Area Tables

<table>
<thead>
<tr>
<th>input texture</th>
<th>original summed-area table</th>
<th>this work</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Input Texture" /></td>
<td><img src="image2.png" alt="Original Summed-Area Table" /></td>
<td><img src="image3.png" alt="This Work" /></td>
</tr>
</tbody>
</table>

SIGGRAPH 2005
Dynamic Glossy Reflections

Outline

• Render dynamic cubemap
• Convert to dual-paraboloid map
• Convert dual-paraboloid map faces to summed-area tables
• Apply summed-area table Dual-paraboloid map to glossy object
• Sounds like a lot of work, but is actually quite fast on modern hardware
  – Real-time demo later
Dual-Paraboloid Maps

- A set of two textures that store an environment as reflected by a pair of parabolic mirrors.
Cubemap to DP Map Conversion

- Convert uv position on DP map face to 3D vector using: (from [Blythe99])

\[
R = \begin{pmatrix}
\frac{2u}{u^2+v^2+1} \\
\frac{2v}{u^2+v^2+1} \\
\frac{-1+u^2+v^2}{u^2+v^2+1}
\end{pmatrix}
\]

Front face: \(R = \begin{pmatrix}
\frac{-2u}{u^2+v^2+1} \\
\frac{-2v}{u^2+v^2+1} \\
\frac{1-u^2-v^2}{u^2+v^2+1}
\end{pmatrix}\)

Back face:

- Do the math on the fly or precompute lookup textures:
Why Bother With DP Mapping?

- Summed-area table concept does not map well to cubemaps
  - Filtering across face boundaries is problematic
  - Potentially forced to read from all six of the cubemaps faces for large kernels
- Filtering in image space with a dual-paraboloid map incurs less error than cubemaps and spherical maps (ref [Kautz00])
Putting it All Together

1. Render cubemap
2. Render dual-paraboloid map
3. Generate summed-area tables
4. Render scene with per-pixel glossy reflections
Direct DP Face Rendering

- Alternative to rendering cubemap, then converting to DP map:
  - Transform environment using parabolic projection function and render directly into DP faces
- Unfortunately parabolic projection is non-linear and maps lines to curves
  - Might be acceptable if your geometry is tessellated highly enough
- See [Coombe04] for details
Other Possibilities

- Average several box-filtered environment map samples to approximate smoother blur filter kernels
- Approximate a Phong BRDF by combining samples from the normal direction and the reflection direction
Real-time Demo
Disadvantages of technique

- Precision requirements for summed-area tables

- Automatic bilinear filtering not supported for float32 textures
  - Not so much of an issue for larger filter kernels
  - Can perform bilinear filtering manually
Conclusion

• Summed-area tables for constant time filtering of textures
• Efficient summed-area table generation scheme using the GPU
  – Does not require reading from and writing to the same texture
• Use summed-area tables and dual-paraboloid mapping together to achieve dynamic glossy environment reflections
Additional Information

• Upcoming Eurographics’05 paper
  – Covers additional applications for fast summed-area table generation

• In depth implementation information in upcoming ShaderX4
Acknowledgments

• ATI Research
• National Science Foundation
  – CCF-0306478, CCF-0205425, CNS-0303590
• Eli Turner for the demo artwork
Questions?