Colored Stochastic Shadow Maps

Morgan McGuire and Eric Enderton
Goal

Photograph
Traditional Shadow Map

[Williams 78]
Stochastic Translucency

Reference Photograph

Williams78 Shadow Map

New Colored Stochastic Shadow Map
Two “Translucent Red” Squares
CSSM Properties

• Unlimited layers of colored, order-independent, non-refractive translucency

• Supports participating media

• Compatible with other shadow map algorithms
  • e.g., dual-paraboloid, cascaded, caching

• Tune for smoothness, performance, or sharpness
  • 0-3 ms time overhead per frame
  • 0x-3x space overhead, depending on precision
Related Work

- **Single surface**: Gosselin et al. 04, Fillion and McNaughton 08: one translucent surface casting onto opaque receivers

- **Deep shadow maps** [Lokovic and Veach 00, Yang et al. 10]: per-pixel linked lists

- **Single scattering**: [Dobashi et al. 02, Mitchell 04, Chen et al. 11]: opaque shadow casters, fog shadow receivers

- **Cloud self-shadowing**: Opacity shadow maps [Kim and Neumann 02], occupancy maps [Sintorn and Assarsson 09], Fourier opacity maps [Jansen and Bavoil 10], Transmittance Function Mapping [Delalandre et al. 11]: soft monochrome self-shadowing within a participating medium

- **Stochastic Transparency** [Enderton et al. 10]: monochrome stochastic shadow maps
ALGORITHM
CSSM1 Algorithm

**generateShadowMap():**

1. For each wavelength $\lambda$:
   (a) Bind and clear depth texture shadow[$\lambda$]
   (b) Set the projection matrix from the light’s viewpoint
   (c) Render all surfaces; discard fragments with $\xi_{\lambda} > \rho_{\lambda}$
2. Return the shadow array

**shadowedLightColor():**

1. Let $\vec{s}_{xyz}$ be the projected shadow map texture coordinate and depth value (as specified by GLSL shadow2D)
2. For each wavelength $\lambda$:
   (a) Let $I_{\lambda} = 0$
   (b) For each sample offset $\Delta$ (of $n$ total):
      i. $\vec{I}_{\lambda} += (\text{texture2D(shadow[\lambda], } \vec{s}_{xy} + \vec{\Delta}).r > \vec{s}_z)$
   (c) $\vec{I}_{\lambda} = \vec{L}_{\lambda} \cdot \vec{I}_{\lambda}/n$
3. Return $\vec{I}$
Traditional shadow map algorithm

**CSSM2 Algorithm**

**generateShadowMap():**
1. Set the projection matrix from the light’s viewpoint
2. Bind and clear the depth buffer and shadow color buffer
3. Disable color write, enable depth write
4. Render all opaque surfaces
5. Enable color write, disable depth write
6. Copy depth to all color channels by rendering a large quad
7. Set MIN blending
   (i.e., \( gl\text{BlendEq}(\text{BLEND\_MIN}); \quad gl\text{BlendFunc}(\text{ONE, ONE}) \))
8. Render all translucent surfaces; let each fragment’s color be
   \[ \max(z, (\xi > \tilde{\rho})) \], where \( z \) is the fragment’s depth value
   (i.e., \( gl\text{FragCoord}.z \))
9. Return the shadow color buffer texture

**shadowedLightColor():**
1. Let \( s_{xyz} \) be the projected shadow map texture coordinate and depth value
2. Let \( \vec{I} = \vec{0} \)
3. For each sample offset \( \vec{\Delta} \) (of \( n \) total):
   (a) \( \vec{I} += (\text{texture2D}(\text{shadow, } s_{xy} + \vec{\Delta}).\text{rgb} > s_z) \)
4. Return \( \frac{\vec{I}}{n} \)
CSSM2 Algorithm

**generateShadowMap():**
1. Set the projection matrix from the light’s viewpoint
2. Bind and clear the depth buffer and shadow color buffer
3. Disable color write, enable depth write
4. Render all opaque surfaces
5. Enable color write, disable depth write
6. Copy depth to all color channels by rendering a large quad
7. Set MIN blending (i.e., `glBlendEq(BLEND_MIN); glBlendFunc(ONE, ONE)`)
8. Render all translucent surfaces; let each fragment’s color be
   \[ \max(z, (\vec{z} > \vec{\rho})) \], where \( z \) is the fragment’s depth value
   (i.e., `glFragCoord.z`)
9. Return the shadow color buffer texture

**shadowedLightColor():**
1. Let \( \vec{s}_{xyz} \) be the projected shadow map texture coordinate and depth value
2. Let \( \vec{I} = \vec{0} \)
3. For each sample offset \( \vec{\Delta} \) (of \( n \) total):
   (a) \( \vec{I} += (\text{texture2D}(\text{shadow, } \vec{s}_{xy} + \vec{\Delta}).\text{rgb} > \vec{s}_z) \)
4. Return \( \vec{I}/n \)
CSSM2 Algorithm

**generateShadowMap()**:  
1. Set the projection matrix from the light’s viewpoint  
2. Bind and clear the depth buffer and shadow color color buffer  
3. Disable color write, enable depth write  
4. Render all opaque surfaces  
5. Enable color write, disable depth write  
6. Copy depth to all color channels by rendering a large quad  
7. Set MIN blending  
   (i.e., `glBlendEq(BLEND_MIN); glBlendFunc(ONE, ONE)`)  
8. Render all translucent surfaces; let each fragment’s color be  
   \( \max(z, (\xi > \rho)) \), where \( z \) is the fragment’s depth value  
   (i.e., `glFragCoord.z`)  
9. Return the shadow color buffer texture

**shadowedLightColor()**:  
1. Let \( \vec{s}_{xyz} \) be the projected shadow map texture coordinate and depth value  
2. Let \( \vec{I} = \vec{0} \)  
3. For each sample offset \( \vec{\Delta} \) (of \( n \) total):  
   (a) \( \vec{I} += (\text{texture2D}(\text{shadow}, \vec{s}_{xy} + \vec{\Delta}).\text{rgb} > \vec{s}_z) \)  
4. Return \( \vec{I}/n \)
Reconstruction Filter

1 sample

25 regular samples

13 irregular samples
RESULTS
Phenomena

Photograph

CSSM
Phenomena

Photograph

CSSM
## Performance

<table>
<thead>
<tr>
<th></th>
<th>Opaque</th>
<th>Translucent</th>
<th>CSSM2 - Wil78</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Game (typical)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate</td>
<td>7.6</td>
<td>7.0</td>
<td>7.4</td>
</tr>
<tr>
<td>1096 kTri Apply</td>
<td>8.4</td>
<td>8.6</td>
<td>7.1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>16.0</strong></td>
<td><strong>15.6</strong></td>
<td><strong>14.5</strong></td>
</tr>
<tr>
<td><strong>Game (worst)</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate</td>
<td>6.6</td>
<td>6.8</td>
<td>6.5</td>
</tr>
<tr>
<td>1096 kTri Apply</td>
<td>7.6</td>
<td>7.5</td>
<td>6.0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>14.2</strong></td>
<td><strong>14.3</strong></td>
<td><strong>12.5</strong></td>
</tr>
<tr>
<td><strong>Sibenik</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate</td>
<td>2.6</td>
<td>2.3</td>
<td>3.1</td>
</tr>
<tr>
<td>80 kTri Apply</td>
<td>4.6</td>
<td>4.0</td>
<td>4.7</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>7.2</strong></td>
<td><strong>6.3</strong></td>
<td><strong>7.8</strong></td>
</tr>
<tr>
<td><strong>Sponza</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate</td>
<td>3.3</td>
<td>4.1</td>
<td>3.3</td>
</tr>
<tr>
<td>66 kTri Apply</td>
<td>4.3</td>
<td>3.2</td>
<td>3.1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>7.6</strong></td>
<td><strong>7.3</strong></td>
<td><strong>6.4</strong></td>
</tr>
<tr>
<td><strong>Postsparkasse</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generate</td>
<td>2.6</td>
<td>4.8</td>
<td>3.4</td>
</tr>
<tr>
<td>267 kTri Apply</td>
<td>6.8</td>
<td>7.1</td>
<td>6.8</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>9.4</strong></td>
<td><strong>11.9</strong></td>
<td><strong>10.2</strong></td>
</tr>
</tbody>
</table>

Per-frame cost in milliseconds for adding translucency

2048^2 shadow map, GeForce 280

*environment diffuse + reflection + refraction + 2 nonshadowed lights + 1 shadowed light*
## Performance

<table>
<thead>
<tr>
<th>Game</th>
<th>Opaque Wil78</th>
<th>Translucent Gos04*</th>
<th>End10*</th>
<th>CSSM1</th>
<th>CSSM2</th>
<th>CSSM2 - Wil78</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate</td>
<td>7.6</td>
<td>7.0</td>
<td>7.4</td>
<td>27.4</td>
<td>7.7</td>
<td></td>
</tr>
<tr>
<td>Apply</td>
<td>8.4</td>
<td>8.6</td>
<td>7.1</td>
<td>9.8</td>
<td>8.4</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>16.0</strong></td>
<td><strong>15.6</strong></td>
<td><strong>14.5</strong></td>
<td><strong>37.2</strong></td>
<td><strong>16.1</strong></td>
<td><strong>0.1</strong></td>
</tr>
</tbody>
</table>

### GeForce 280
- (published)

### GeForce 280*
- (new)

### GeForce 480*

- 2.5x faster than published results

### GeForce 480*
- 1024² shadow map, single unified lighting pass

* Preliminary results
Participating Medium
CSSM + Mitchell 04
Participating Medium

Smoke casts and receives shadows.

Orange pipes are translucent.
Glass in a Light Shaft

Enderton et al. 10, Fillion and McNaughton 09: No color

Gosselin et al. 04: Wrong color

New CSSM: Correct color
Postsparkasse
Conclusion

• Robust shadowing algorithm for transmission and partial coverage
• Easy to integrate with existing renderers
• Modest incremental time and space costs
• Recipe for robust and fast graphics:

  “Very Good Samples” +
  “Good Enough Filters”
Thank you

Thanks to Marc Blackstein at NVIDIA for optimization advice