Real-Time Graphics Architecture

Kurt Akeley

Pat Hanrahan

http://www.graphics.stanford.edu/courses/cs448a-01-fall

Advanced Shading and Texturing

Topics

Features

- Bump mapping
- Environment mapping
- Shadow mapping

Mechanisms

- Multipass
- Multitexturing
- Dependent texturing
- Texture addressing
- Texture combiners

Readings

Background

1. T. McReynolds et al., Advanced graphics programming techniques using OpenGL

   Compendium of multipass rendering techniques

   Available online

2. ATI and NVIDIA developer pages

Bumps

Readings: Bump mapping

Required

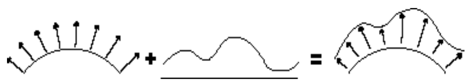
1. M. Peercy, J. Airey, B. Cabral, Efficient Bump mapping hardware, 2000
2. M. Kilgard, A practical and robust bump mapping for today's GPUS, 2001

Recommended

1. J. Blinn, Simulation of wrinkled surfaces, 1978

Bump Mapping [Blinn 1978]

Offset surface position



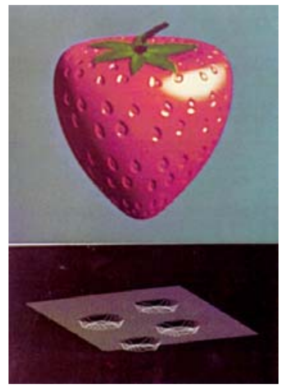
Displacement

$$\mathbf{N}(u,v) = \frac{\mathbf{P}_u(u,v) \times \mathbf{P}_v(u,v)}{\left|\mathbf{P}_u(u,v) \times \mathbf{P}_v(u,v)\right|}$$

$$\mathbf{P}'(u,v) = \mathbf{P}(u,v) + h(u,v)\mathbf{N}(u,v)$$

Perturb normal

$$\mathbf{N}'(u,v) = \frac{\mathbf{P}'_u(u,v) \times \mathbf{P}'_v(u,v)}{\left|\mathbf{P}'_u(u,v) \times \mathbf{P}'_v(u,v)\right|}$$



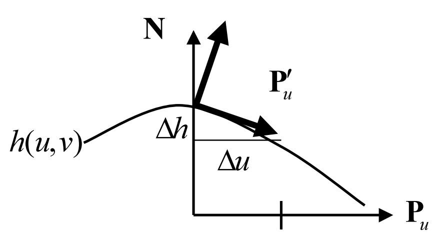
From Blinn 1978

Bumps from Heights

$$\mathbf{P}'_u = \mathbf{P}_u + h_u\mathbf{N} + h\mathbf{N}_u = \mathbf{P}_u + h_u\mathbf{N} + O(\Delta u^2) \approx \mathbf{P}_u + h_u\mathbf{N}$$

$$\mathbf{P}'_v = \mathbf{P}_v + h_v\mathbf{N} + h\mathbf{N}_v = \mathbf{P}_v + h_v\mathbf{N} + O(\Delta v^2) \approx \mathbf{P}_v + h_v\mathbf{N}$$

$$\mathbf{N}' = \mathbf{N} - h_u(\mathbf{P}_v \times \mathbf{N}) - h_v(\mathbf{N} \times \mathbf{P}_u)$$



Tangent Space

Concept from differential geometry

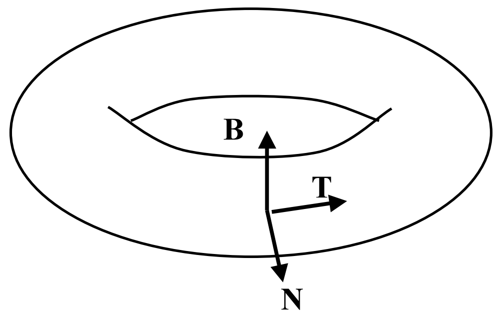
The set of all tangents on a surface

Orthonormal coordinate system or frame at each point

$$\mathbf{N} = \frac{\mathbf{P}_u \times \mathbf{P}_v}{\left|\mathbf{P}_u \times \mathbf{P}_v\right|}$$

$$\mathbf{T} = \frac{\mathbf{P}_u}{\left|\mathbf{P}_u\right|}$$

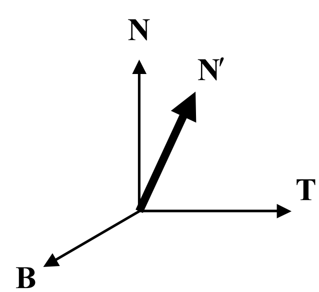
$$\mathbf{B} = \mathbf{N} \times \mathbf{T}$$



Normals Maps

In tangent frame

$$\mathbf{N}' = \begin{pmatrix}\mathbf{T} & \mathbf{B} & \mathbf{N}\end{pmatrix} \begin{pmatrix} -h_T \\ -h_B \\ 1 \end{pmatrix}$$



Normal map

$$\mathbf{N}' = \begin{pmatrix}\mathbf{T} & \mathbf{B} & \mathbf{N}\end{pmatrix} \begin{pmatrix} N_T \\ N_B \\ N_N \end{pmatrix}$$

Directional derivatives

$$h_T = (\mathbf{T} \bullet \nabla) h$$

$$h_B = (\mathbf{B} \bullet \nabla) h$$

Distortions

Classic texture mapping

- Length of parametric derivatives and Jacobian
  - Area changes over surface controls local resolution

Bump mapping

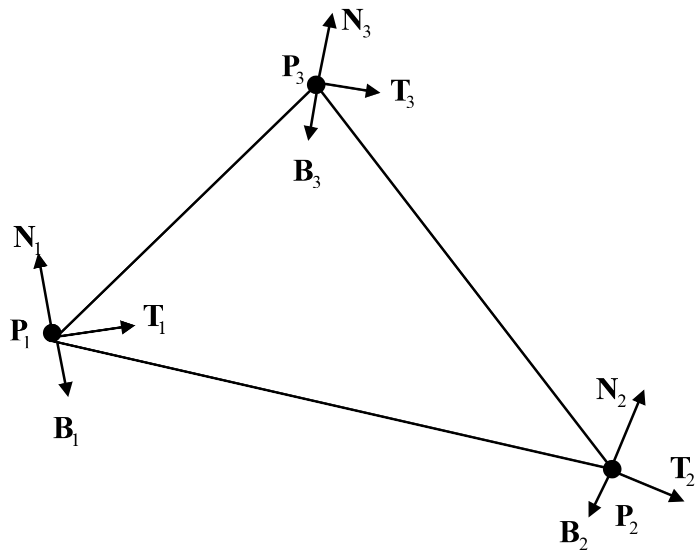
- Scale transformation
  - Controls absolute height of the bumps wrt the surface
- Length of parametric derivatives
  - Controls relative height of bumps across the surface

Modeling vs. Rendering?

Bump Map Representations

1. Height field ~ Embossing
   Single high precision component
   Directional derivative: $\hat{\mathbf{L}} \bullet \hat{\mathbf{N}} = \left(\hat{\mathbf{L}} \bullet \nabla\right) h$
2. Derivatives (HI,LO) $(h_T, h_B) = (h_T, h_B, 1)$
   Scale factor needed to maintain control height
3. Normal maps (R,G,B) $(\mathbf{N}_T, \mathbf{N}_B, \mathbf{N}_N)$
4. Normal index maps (index)
   Index addresses a table of precomputed normals
   16-bit indices more than sufficient

Tangent Space on Triangles



Transformation

Transformation from tangent space to object space

$$\mathbf{R} = \begin{pmatrix} \mathbf{T} & \mathbf{B} & \mathbf{N} \end{pmatrix} = \begin{pmatrix} \mathbf{T}_x & \mathbf{B}_x & \mathbf{N}_x \\ \mathbf{T}_y & \mathbf{B}_y & \mathbf{N}_y \\ \mathbf{T}_z & \mathbf{B}_z & \mathbf{N}_z \end{pmatrix}$$

Transformation from object space to tangent space

$$\mathbf{R}^{-1} = \begin{pmatrix} \mathbf{T} & \mathbf{B} & \mathbf{N} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{T} & \mathbf{B} & \mathbf{N} \end{pmatrix}^{T} = \begin{pmatrix} \mathbf{T}_x & \mathbf{T}_y & \mathbf{T}_z \\ \mathbf{B}_x & \mathbf{B}_y & \mathbf{B}_z \\ \mathbf{N}_x & \mathbf{N}_y & \mathbf{N}_z \end{pmatrix}$$

Remember that normals tranform as $R^{-T}=R$

Basic Algorithm (Eye Space)

For scene (assumes infinite L and E, otherwise per-v or per-f)

- Transform L and E to eye space and normalize
- Compute normalized H

For each vertex

- Transform N from object space to eye space

For each fragment

- Interpolate and renormalize N
- Compute Pu and Pv in eye space (may require transform)
- Fetch (hu, hv) = texture(s,t,q,r)
- Compute N’ = N + hu (N x Pu) + hv (Pv x N)
- Normalize N’
- Compute max(L.N’,0) and max(H.N’,0)^s
- Combine using the standard diffuse+specular equation

Fast Algorithm (Tangent Space)

For each vertex

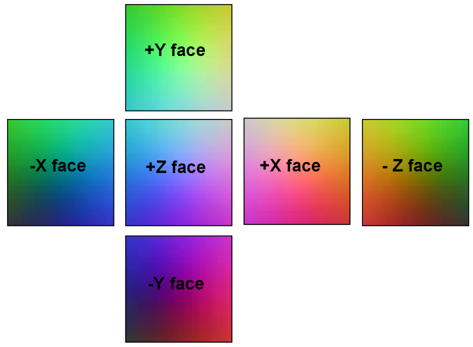
- Transform L and E to tangent space and normalize
- Compute normalized H

For each fragment

- Interpolate L and H (no need to interpolate N)
- Renormalize L and H
- Fetch N’ = texture(s,t,q,r)
- Compute max(L.N’,0) and max(H.N,0)^s
- Combine using the standard diffuse+specular lighting equation

Normalizing Vectors

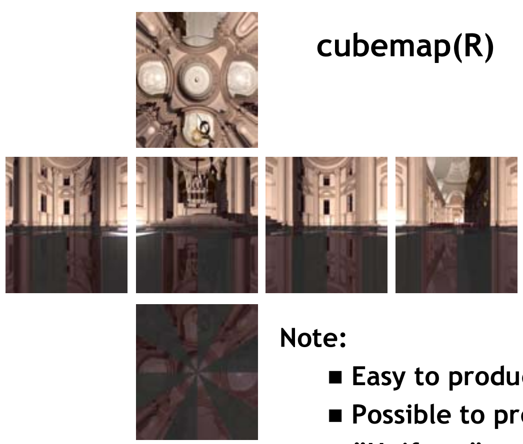
Cubemap(R) = R/|R|, [Voorhies and Foran, 1994]



Alternatively, use one iteration of a Newton-Raphson reciprocal square root algorithm

Cubical Environment Map

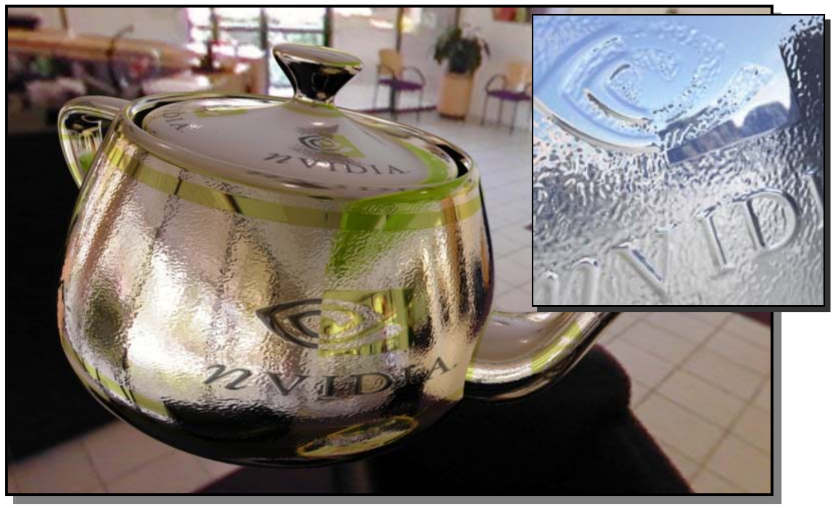
cubemap(R)



Note:

- Easy to produce with rendering system
- Possible to produce from photographs
- "Uniform" resolution
- Simple texture coordinates calculation

Reflective Bump Mapping



From Reflective Bump Mapping presentation, C. Everett

Reflective Bump Mapping Algorithm

For each vertex

- Transform E to world space
- Compute tangent space to world space transform (T,B,N)

For each fragment

- Interpolate and renormalize E
- Interpolate frame (T,B,N)
- Lookup N = texture(s,t,q,r)
- Transform N from tangent space to world space
- Compute reflection vector R (in world space)

$$\mathbf{R} = -\mathbf{E} + 2\mathbf{N}(\mathbf{N} \bullet \mathbf{E})$$

- Lookup C = cubemap(R)

Note: This is an example of *dependent texturing*

Shading Space

How to avoid transformations?

Definition space:

- E in eye space
- L in light space
- H in eye space (transform L to eye space)
- Shadow maps in light space
- Environment maps in world space
- Normal maps in tangent space

Important now, important in the future?

See Miller, Halstead and Clifton for algorithms

Filtering Bump Maps

Ideally

Compute reflection at different positions on the surface with different perturbed normals a high resolution

Average reflected values

Practically

Average perturbed normals

Compute reflection using the average normal

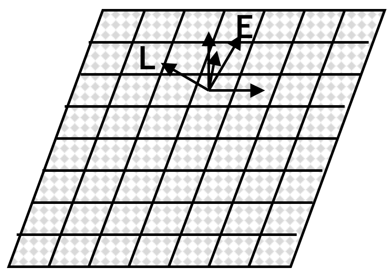
Still an unsolved problem

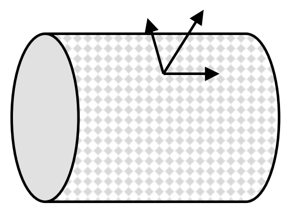
Shade-First

Uniformly or adaptively tesselate the surface

Shade the surface (in object- or tangent-space)

Warp shaded surface to image-space





Catmull and Smith, 1980

Shadows

Readings: Shadows

Required

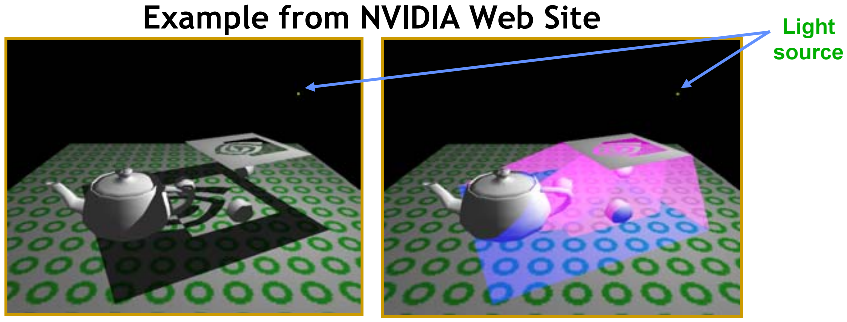
1. F. Crow, Shadow algorithms for computer graphics, SIGGRAPH 77
2. L. Williams, Casting curved shadows on curved surface, SIGGRAPH 78

Recommended

1. W. Reeves, D. Salesin, and R. Cook (Pixar), Rendering antialiased shadows with depth maps, SIGGRAPH 87
2. M. Segal, et al. (SGI), Fast shadows and lighting effects using texture mapping," SIGGRAPH 92

Shadow Volumes [Crow 1977]

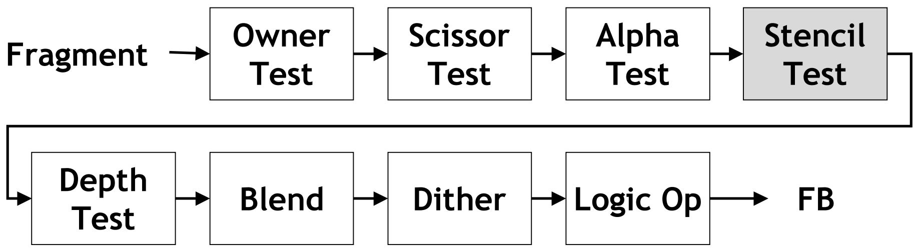
Example from NVIDIA Web Site



Given a point light source:

1. Each back-facing triangle is culled
2. Each silhouette edge generates an infinite quadrilateral
3. Each front-facing triangle is kept

Stencil Stage



Stencil buffer (0-32 bits)

Stencil test

- Tests against value from stencil buffer; rejects fragment if stencil test fails.
- Operations on stencil buffer depending on the results of the stencil and depth tests.

Stencil Test

Compares reference value to stencil buffer value

```
glStencilFunc(op, ref, mask);
```

Same comparison functions as alpha and depth tests

- NEVER, ALWAYS
- LESS, LEQUAL
- GREATER, GEQUAL
- EQUAL, NOTEQUAL

Bit mask controls comparison

```
test = ((ref & mask) op (svalue & mask))
```

Stencil Operations

Three possible stencil side effects

```
glStencilOp(fail, zfail, zpass);
```

Possible operations

- Increment, Decrement (saturates)
- Increment, Decrement (wrap, DX6 option)
- Keep, Replace
- Zero, Invert

Stencil mask controls write-back

```
glStencilMask(mask);
```


Stencil Shadow Algorithm

1. Render scene to create depth buffer

   Don't shade when rendering

2. Render the shadow volume to create stencil buffer

   Invert stencil bits when depth test passes

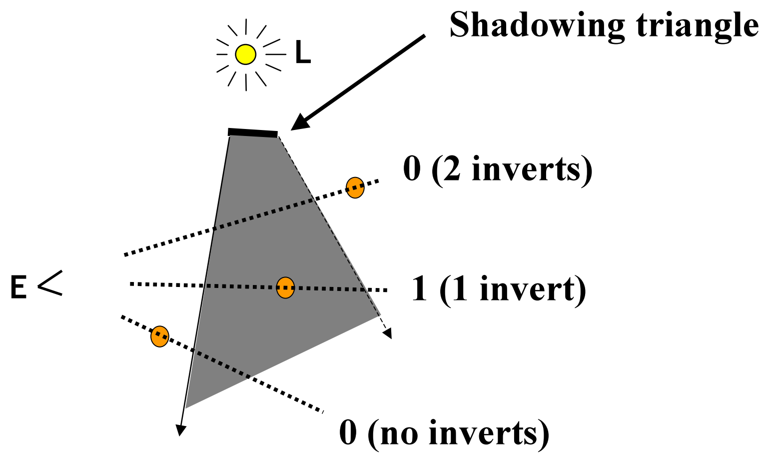
   ```
   glStencilOp(GL_KEEP,GL_KEEP,GL_INVERT);
   ```

Result:

- Pixels outside the shadow volume are inverted an even number of times
- Pixels inside the shadow volume are inverted an odd number of times

Example

From Kilgard



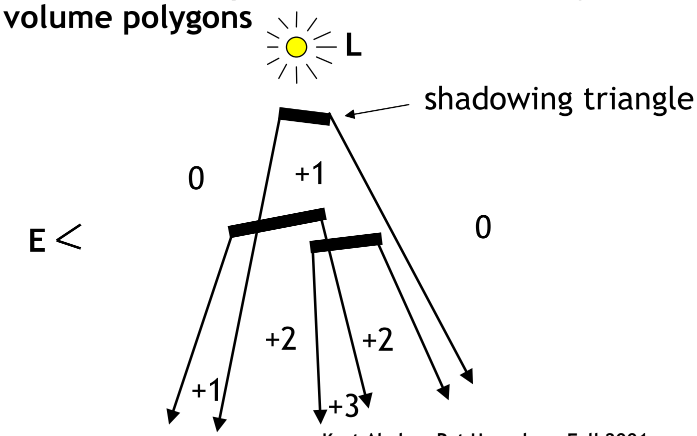
Render In and Out of Shadow

3. Render the scene with the light disabled, update only pixels with an odd stencil bit setting
4. Render the scene with the light enabled, update only pixels with an even stencil bit setting

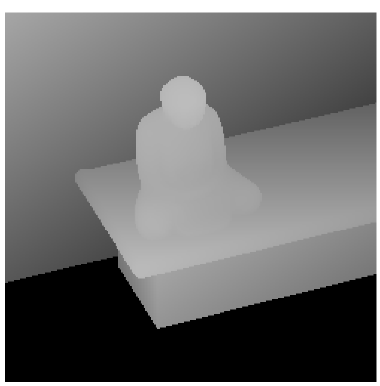
Multiple Shadow Volumes

Use the GL_INCR stencil operation for front facing shadow volume polygons

Use the GL_DECR stencil operation for back facing shadow volume polygons



Shadow Maps



Depth buffer

William's Shadow Map Algorithm

1. Render the scene from the point of view of the light source to create a depth map

   Note: No need to shade

2. Render the scene from the point of view of the eye
   1. Transform fragment positions to light space
   2. Compare light z with shadow map z

      Alpha = (zl < shadow[xl][yl].z + bias)

   3. Modulate color by shadow matte

Problem:

Attenuates after reflection, not before!

Shadow Map Algorithm

1. Render the scene from the point of view of the light source to create a depth map
2. Copy texture to shadow texture map
3. Render the scene from the point of view of the eye
   1. Transform vertex eye positions to light space positions
   2. Perspectively-correct interpolate light space positions
   3. Compare light z with shadow map z
   4. Set stencil

Projected Textures [Segal et al.]

For each vertex

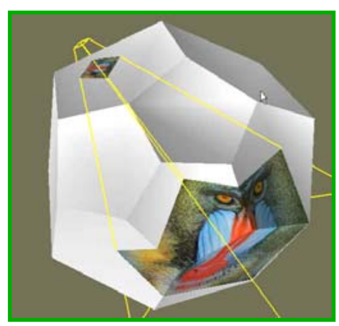
- Transform P to light space
- Set texture coordinates to light space positions

For each fragment

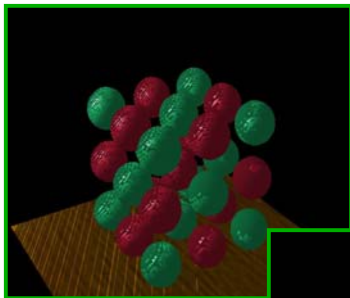
- Interpolate light space projective texture coords.
  - (sr,tr,qr,r)
- Compute projective texture coordinates
  - (sr/r,tr/r,qr/r,1)
- Lookup color in textures
  - v = texture(sr/r,tr/r)

Slide Projector

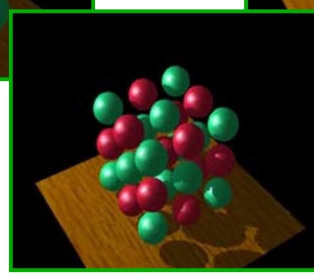
Source: Heidrich [99]



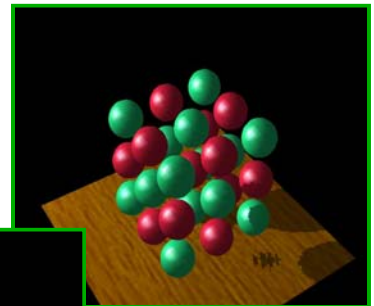
Bias



Too little bias, everything begins to shadow



Just right

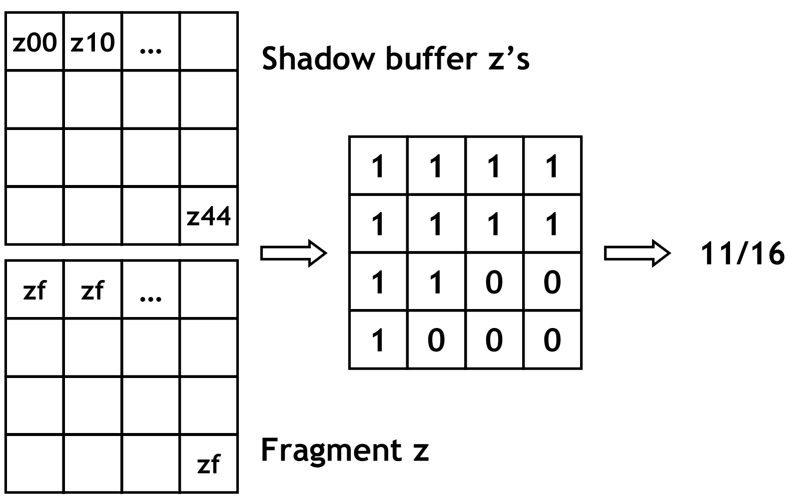


Too much bias, shadow starts too far back

From NVIDIA

Percentage-closer Filtering

Reeves, Salesin and Cook, 1991



Summary: Shadows

Shadow volumes

- Drawing shadow volumes consumes lots of fill rate
- Relatively accurate

Shadow maps

- Relatively efficient: one additional drawing only (no shading) pass per light source
- Precision and biasing issues lead to cruftiness and hand tuning per scene; difficult for dynamic scenes
- Maybe extended to linear and area light sources to create soft shadows with penumbras

Shadows are very important

But still remain difficult to implement and are quite costly

Trends and Observations

Reduced features to orthogonal mechanisms

- Texture types (normals, shadows, environments)
- Multitexturing
- Dependent texturing
- Texture addressing
- Texture combining and multipass

Quickly becomes very general: programmable

Defining moments filled with tricks and approximations

These tricks often become fundamental and are built upon

Trends and Observations

Shading and texturing costs dominate

99.99% of the rendering time in movie production

Where in the pipeline?

- Vertex shading
  - Lights and texture maps in object space; floating point
  - Low shading rate (per-vertex), no texturing
- Fragment shading
  - Access to limited data; low precision fixed point
  - High shading rate (per-fragment), texturing

Evolve towards …

- Reyes
- Ray tracing