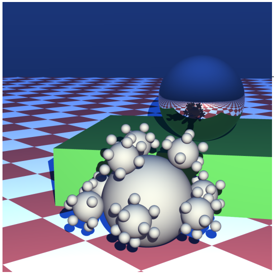
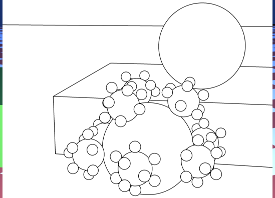
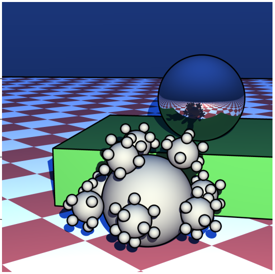
Ray Tracing NPR-Style Feature Lines

A.N.M. Imroz Choudhury   Steven G. Parker

Scientific Computing and Imaging Institute
University of Utah

August 1, 2009

Why Ray Tracing?

- High-quality images (shadows, refraction/reflection, etc. are straightforward)
- Photorealism without hacks
- Interactive now, and still improving
- Easier to use than OpenGL (for some applications)
- On the rise

Why Ray Tracing?

- High-quality images (shadows, refraction/reflection, etc. are straightforward)
- Photorealism without hacks
- Interactive now, and still improving
- Easier to use than OpenGL (for some applications)
- On the rise

Why Ray Tracing?

- High-quality images (shadows, refraction/reflection, etc. are straightforward)
- Photorealism without hacks
- Interactive now, and still improving
- Easier to use than OpenGL (for some applications)
- On the rise

Why Ray Tracing?

- High-quality images (shadows, refraction/reflection, etc. are straightforward)
- Photorealism without hacks
- Interactive now, and still improving
- Easier to use than OpenGL (for some applications)
- On the rise

Why Ray Tracing?

- High-quality images (shadows, refraction/reflection, etc. are straightforward)
- Photorealism without hacks
- Interactive now, and still improving
- Easier to use than OpenGL (for some applications)
- On the rise

Why Ray Tracing?

- High-quality images (shadows, refraction/reflection, etc. are straightforward)
- Photorealism without hacks
- Interactive now, and still improving
- Easier to use than OpenGL (for some applications)
- On the rise

It pays to see how NPR fits in ray tracing.

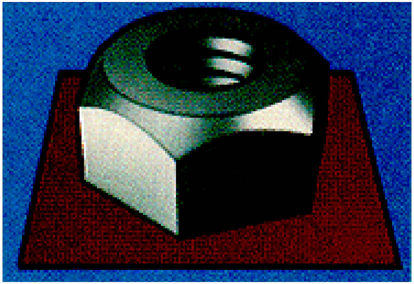
Why Ray Tracing?

High-Quality Images

Why Feature Lines?

Feature lines can

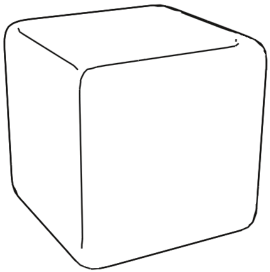
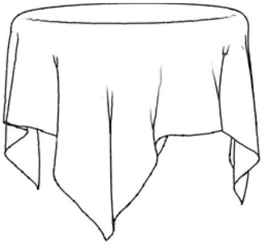
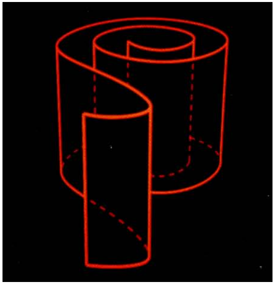
- enhance geometric features (Saito and Takahashi 1990)
- succinctly express shape (Judd et al. 2007, Dooley and Cohen 1990)
- indicate confidence in architectural rendering (Potter et al. 2009)



Why Feature Lines?

Feature lines can

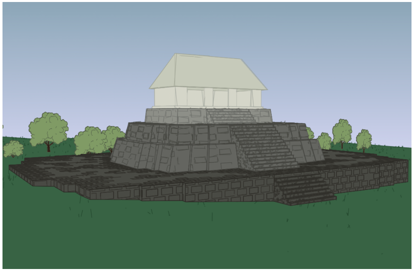
- enhance geometric features (Saito and Takahashi 1990)
- succinctly express shape (Judd et al. 2007, Dooley and Cohen 1990)
- indicate confidence in architectural rendering (Potter et al. 2009)

Why Feature Lines?

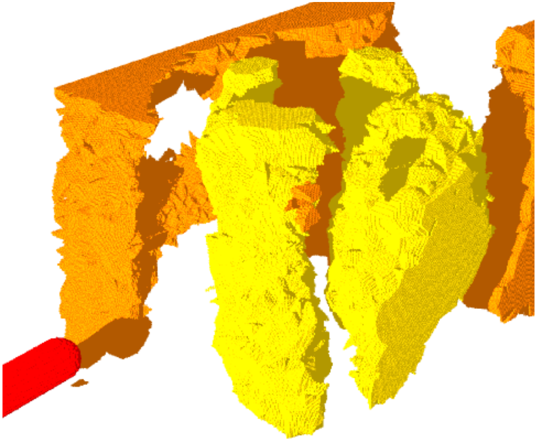
Feature lines can

- enhance geometric features (Saito and Takahashi 1990)
- succinctly express shape (Judd et al. 2007, Dooley and Cohen 1990)
- indicate confidence in architectural rendering (Potter et al. 2009)



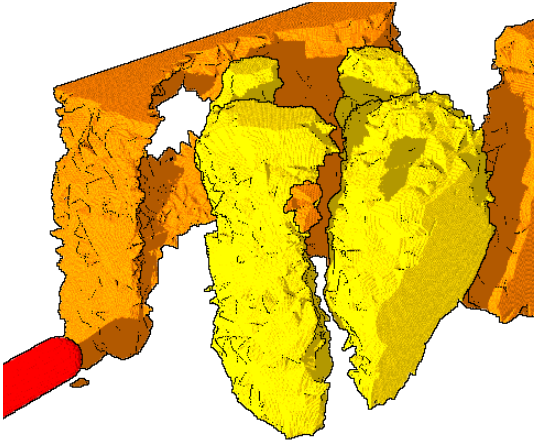
Driving Application—Scientific Visualization

Silhouette edges can indicate particle groupings
(Bigler et al. 2006).



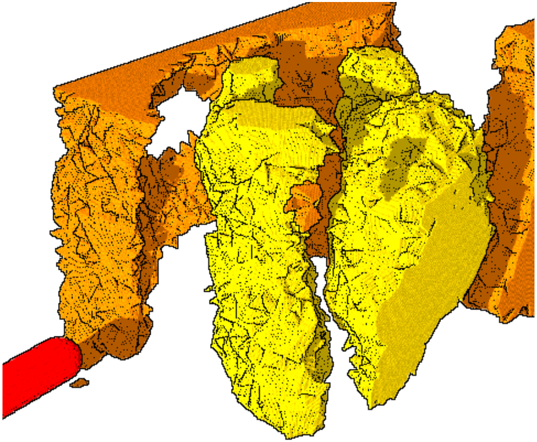
Driving Application—Scientific Visualization

Silhouette edges can indicate particle groupings (Bigler et al. 2006).



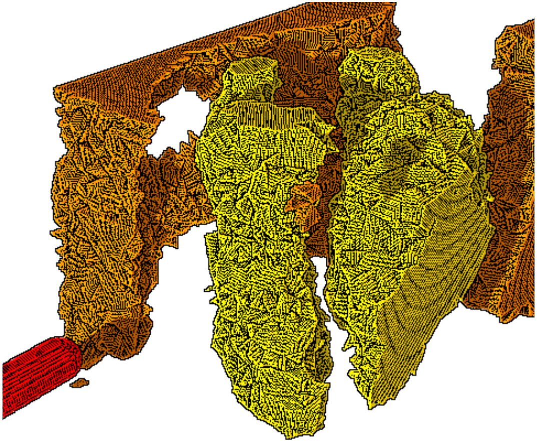
Driving Application—Scientific Visualization

Silhouette edges can indicate particle groupings (Bigler et al. 2006).



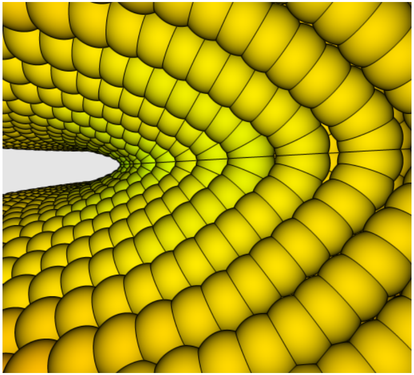
Driving Application—Scientific Visualization

Silhouette edges can indicate particle groupings (Bigler et al. 2006).



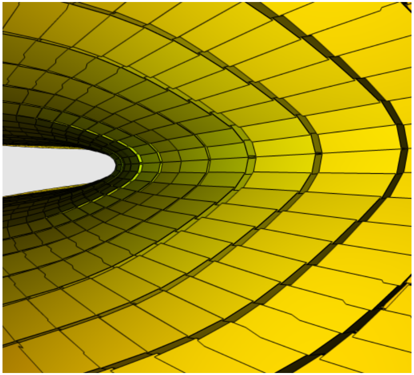
Driving Application—Scientific Visualization

Generalize to *direct rendering* of feature lines from geometry itself (Choudhury et al.)



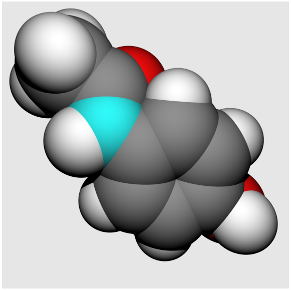
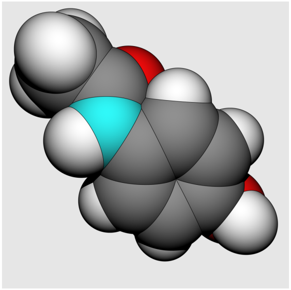
Driving Application—Scientific Visualization

Generalize to *direct rendering* of feature lines from geometry itself (Choudhury et al.)



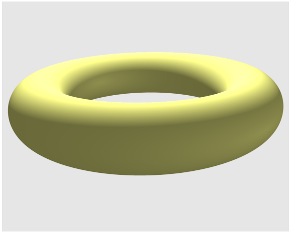
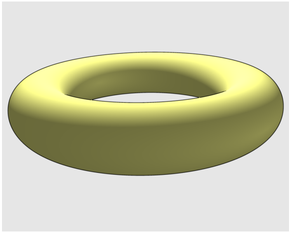
Feature Line Types

- *Intersection lines:* two objects intersect and form a seam
- *Silhouette lines* (or *edges*): the edge of an object lies against the background, a different object, or a further part of itself (i.e. a *self-occluding* silhouette)
- *Crease lines:* an object has a sharp corner (a discontinuity in the gradient of the normal field)

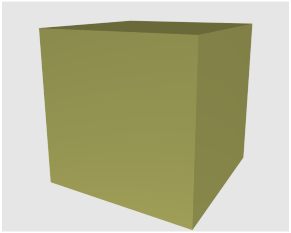
Feature Line Types

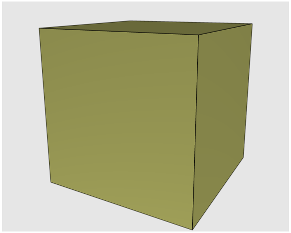
- *Intersection lines:* two objects intersect and form a seam
- *Silhouette lines* (or *edges*): the edge of an object lies against the background, a different object, or a further part of itself (i.e. a *self-occluding* silhouette)
- *Crease lines:* an object has a sharp corner (a discontinuity in the gradient of the normal field)

Feature Line Types

- *Intersection lines:* two objects intersect and form a seam
- *Silhouette lines* (or *edges*): the edge of an object lies against the background, a different object, or a further part of itself (i.e. a *self-occluding* silhouette)
- *Crease lines:* an object has a sharp corner (a discontinuity in the gradient of the normal field)





Lines in Ray Tracing?

- Ray tracing deals in “physical” primitives: sphere, cone, torus, disc, triangle, etc.

Lines in Ray Tracing?

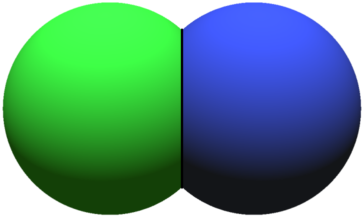
- Ray tracing deals in “physical” primitives: sphere, cone, torus, disc, triangle, etc.
- Lines are *not* physical—they have no breadth

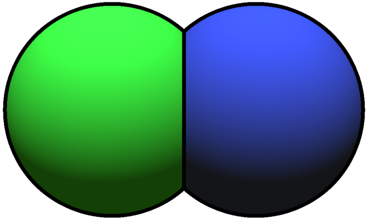
Lines in Ray Tracing?

- Ray tracing deals in "physical" primitives: sphere, cone, torus, disc, triangle, etc.
- Lines are *not* physical—they have no breadth
- Can try "line-like" primitives, e.g. thin cylinders and toruses

Lines in Ray Tracing?

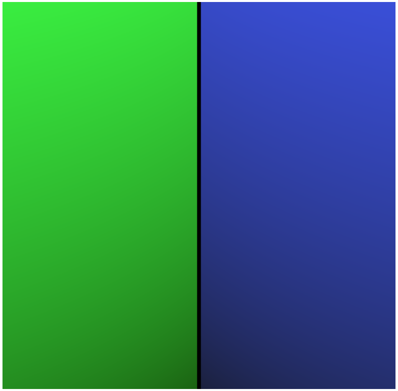
But *geometry* doesn't work as *lines!*





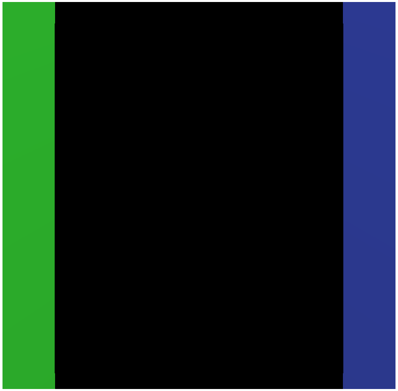
Lines in Ray Tracing?

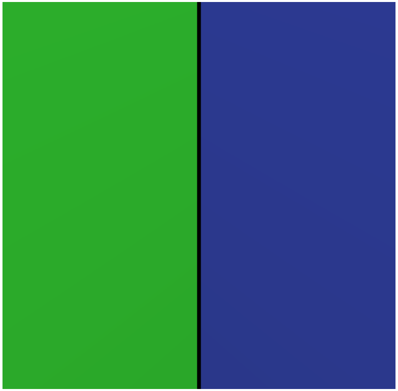
But *geometry* doesn't work as *lines!*



Lines in Ray Tracing?

But *geometry* doesn't work as *lines!*





Lines in Ray Tracing?

We would like to

- draw non-physical lines

Lines in Ray Tracing?

We would like to

- draw non-physical lines
- with constant width in screen space

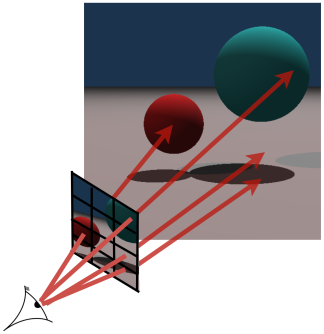
Lines in Ray Tracing?

We would like to

- draw non-physical lines
- with constant width in screen space

i.e. we want to rasterize lines

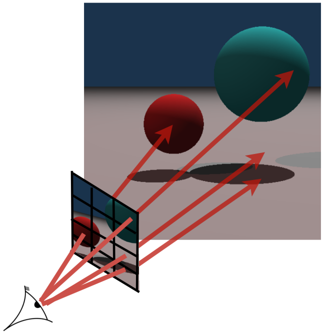
Ray Tracing
Algorithm Overview



(Figure courtesy of Thiago Ize)

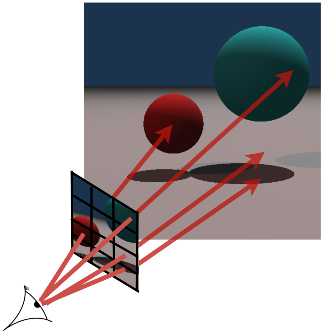
- *Camera rays* cast through the *image plane*, striking the scene at *intersection points*
- *Secondary rays* cast from the intersection points for secondary effects (shadows, reflections, etc.)
- *Sample colors* computed from ray results and *shading model*
- *Final image* assembled from filtered sample colors

Ray Tracing
Algorithm Overview


(Figure courtesy of Thiago Ize)

- *Camera rays* cast through the *image plane*, striking the scene at *intersection points*
- *Secondary rays* cast from the intersection points for secondary effects (shadows, reflections, etc.)
- *Sample colors* computed from ray results and *shading model*
- *Final image* assembled from filtered sample colors

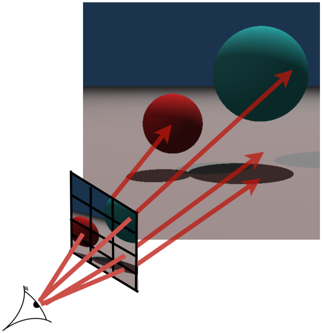
Ray Tracing
Algorithm Overview


(Figure courtesy of Thiago Ize)

- *Camera rays* cast through the *image plane*, striking the scene at *intersection points*
- *Secondary rays* cast from the intersection points for secondary effects (shadows, reflections, etc.)
- *Sample colors* computed from ray results and *shading model*
- *Final image* assembled from filtered sample colors

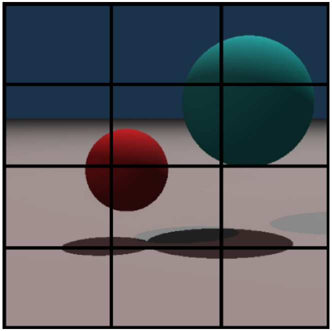
Ray Tracing

Algorithm Overview


(Figure courtesy of Thiago Ize)

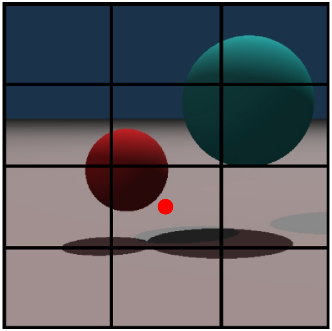
- *Camera rays* cast through the *image plane*, striking the scene at *intersection points*
- *Secondary rays* cast from the intersection points for secondary effects (shadows, reflections, etc.)
- *Sample colors* computed from ray results and *shading model*
- *Final image* assembled from filtered sample colors

Ray Tracing
Navigating Screen Space



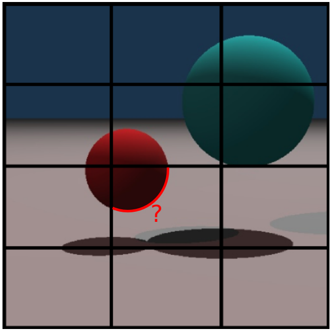
- Camera rays determine visibility

Ray Tracing
Navigating Screen Space



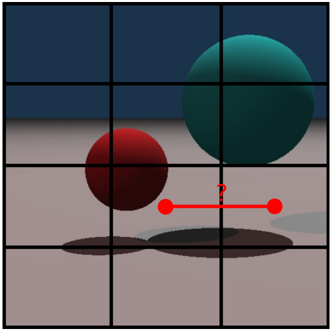
- Camera rays determine visibility
- Parameterized by camera position and pixel position; i.e., they live in screen space

Ray Tracing
Navigating Screen Space



- Camera rays determine visibility
- Parameterized by camera position and pixel position; i.e., they live in screen space
- With a way to
    1. detect feature lines, and

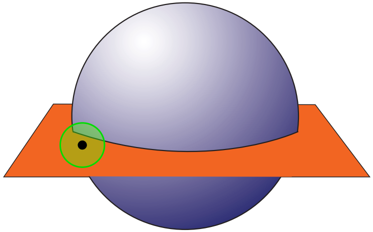
Ray Tracing
Navigating Screen Space

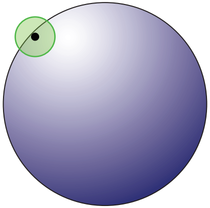


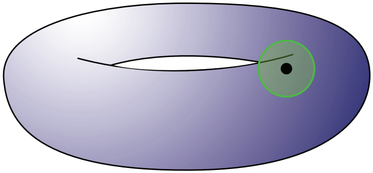
- Camera rays determine visibility
- Parameterized by camera position and pixel position; i.e., they live in screen space
- With a way to
  1. detect feature lines, and
  2. measure distances in screen space

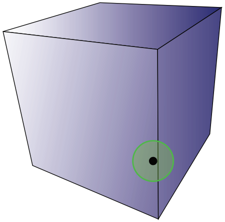
  we can incorporate feature line rendering into a ray tracer.

Detecting Feature Lines


Intersection

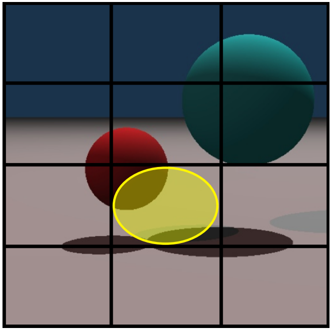

Silhouette


Self-occluding silhouette


Crease

Measuring Distances

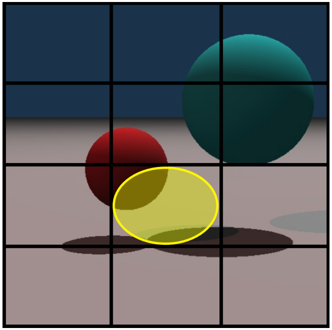
Cone Tracing (Amanatides 1984)



- Trace a *cone* instead of a ray; footprint is circle instead of point

Measuring Distances

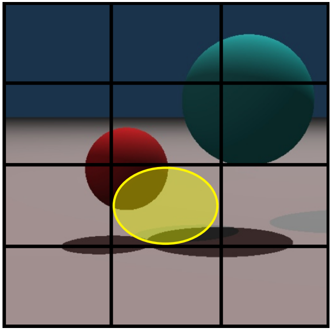
Cone Tracing (Amanatides 1984)



- Trace a *cone* instead of a ray; footprint is circle instead of point
- Used for non-singular scene coverage: anti-aliasing, glossy reflections, etc.

Measuring Distances

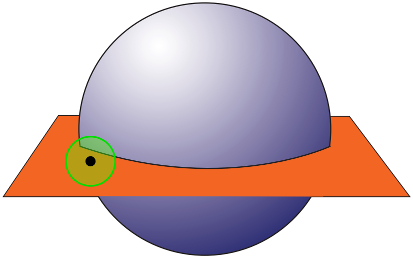
Cone Tracing (Amanatides 1984)



- Trace a *cone* instead of a ray; footprint is circle instead of point
- Used for non-singular scene coverage: anti-aliasing, glossy reflections, etc.
- We borrow the idea of a ray having a radius; our notion of non-physical feature lines exists over some area of the image.

Drawing Feature Lines

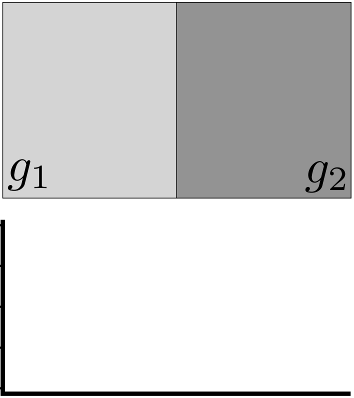
Continuous Case



- Estimate *foreign geometry area* (*FGA*)

Drawing Feature Lines

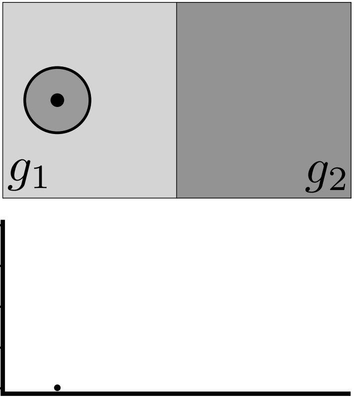
Continuous Case



- Estimate *foreign geometry area* (*FGA*)

Drawing Feature Lines

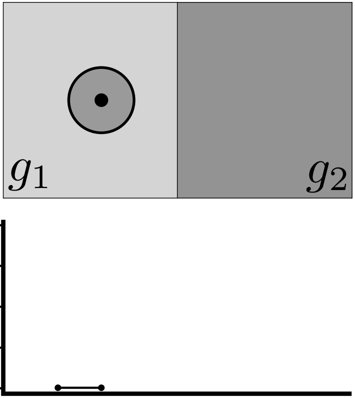
Continuous Case



- Estimate *foreign geometry area* (*FGA*)

Drawing Feature Lines

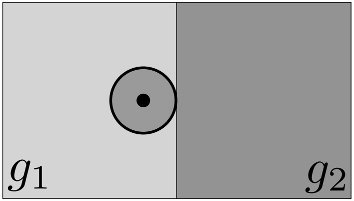
Continuous Case

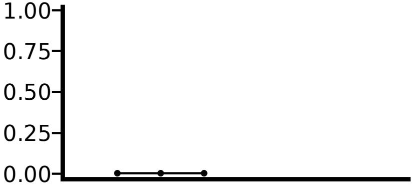


- Estimate *foreign geometry area* (*FGA*)

Drawing Feature Lines

Continuous Case

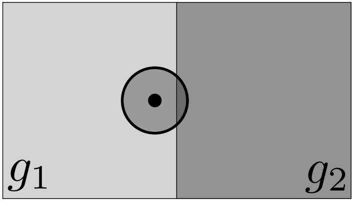


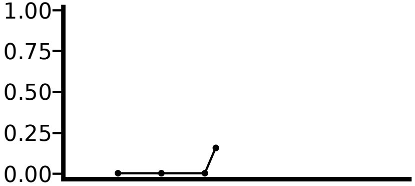


- Estimate *foreign geometry area* (*FGA*)

Drawing Feature Lines

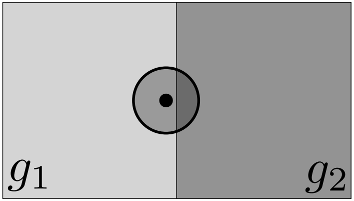
Continuous Case

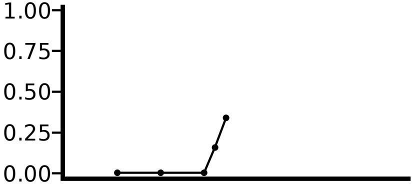




- Estimate *foreign geometry area* (*FGA*)

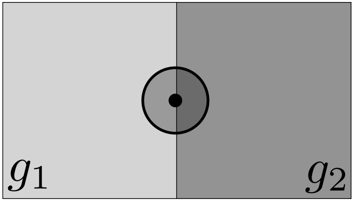
Drawing Feature Lines
Continuous Case

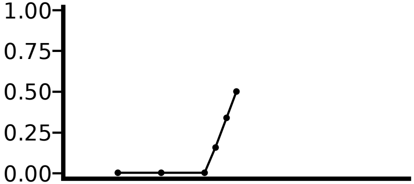




- Estimate *foreign geometry area* (*FGA*)

Drawing Feature Lines
Continuous Case

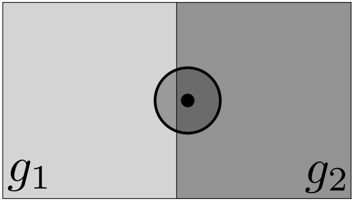


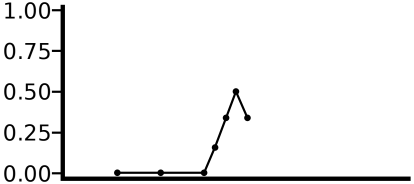


- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%

Drawing Feature Lines

Continuous Case

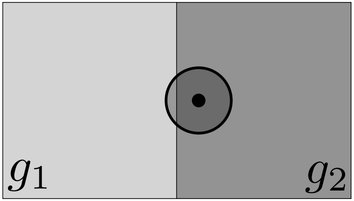


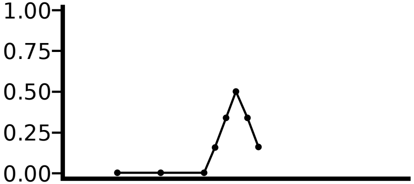


- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%

Drawing Feature Lines

Continuous Case

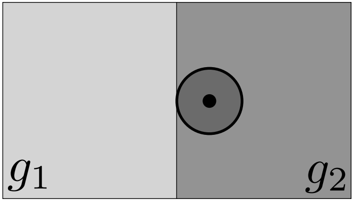


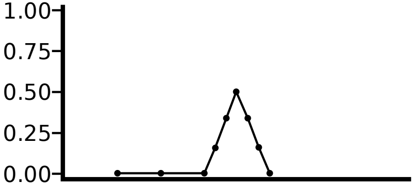


- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%

Drawing Feature Lines

Continuous Case

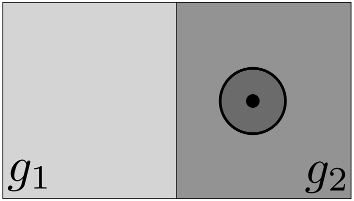


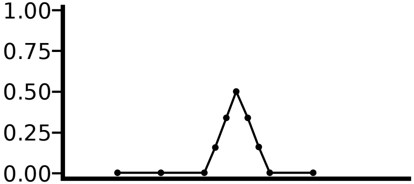


- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%

Drawing Feature Lines

Continuous Case

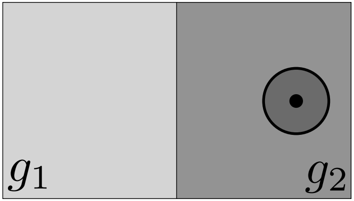


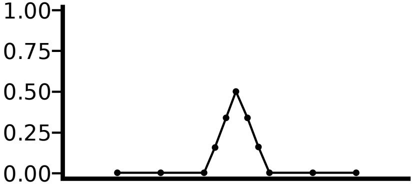


- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%

Drawing Feature Lines

Continuous Case

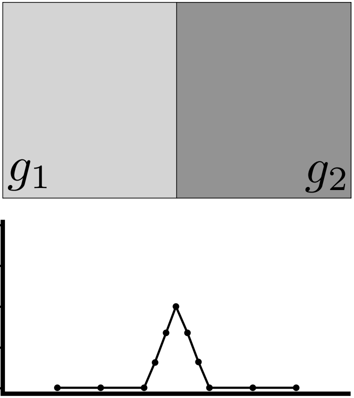




- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%

Drawing Feature Lines

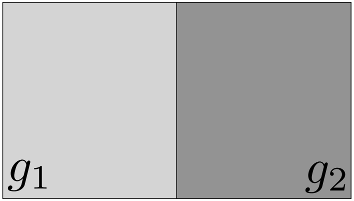
Continuous Case

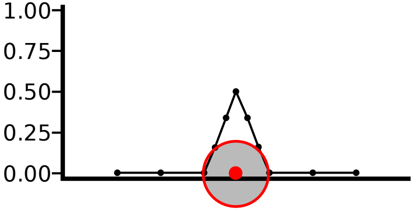


- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%

Drawing Feature Lines

Continuous Case

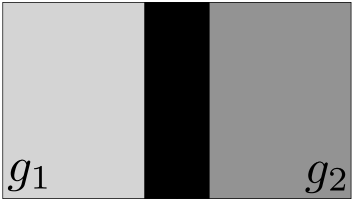


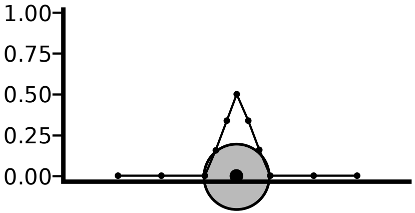


- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%
- Note: filter diameter equals width of peak

Drawing Feature Lines

Continuous Case

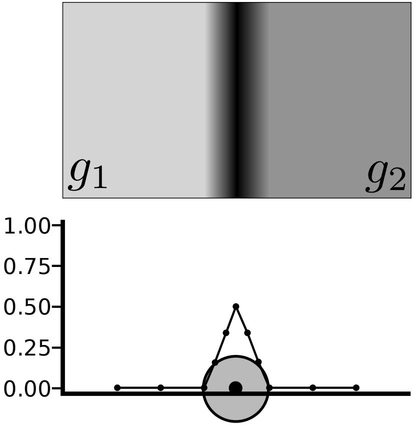




- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%
- Note: filter diameter equals width of peak
- Easiest way to create a line: black where $FGA > 0$; sample color where $FGA = 0$

Drawing Feature Lines

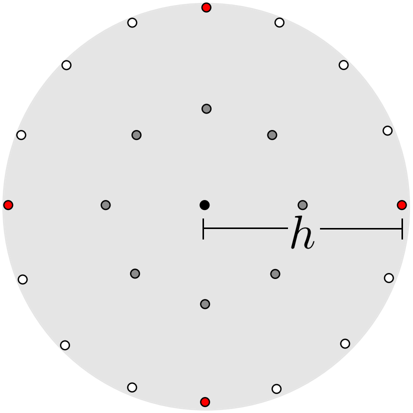
Continuous Case



- Estimate *foreign geometry area* (*FGA*)
- Intuition: edge must be strong where *FGA* is 50%
- Note: filter diameter equals width of peak
- Easiest way to create a line: black where $FGA > 0$; sample color where $FGA = 0$
- More generally: determine darkness of line *as a function of FGA*; i.e. use an *edge strength metric*

Drawing Feature Lines

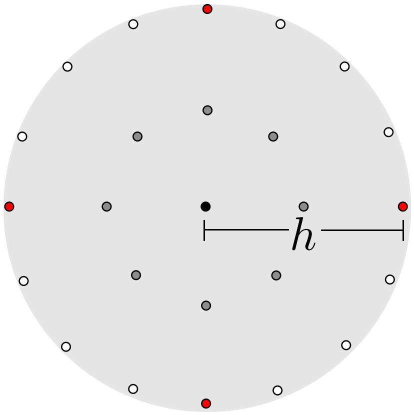
Discretizing to Ray Stencils



- Approximate filter by sampling the disc
- h is a distance in *screen space*
- Increase sampling density by packing more rings of samples
- Estimate *FGA* by *counting* which rays hit what
- Red samples form *finite difference stencil* for computing creases

Drawing Feature Lines

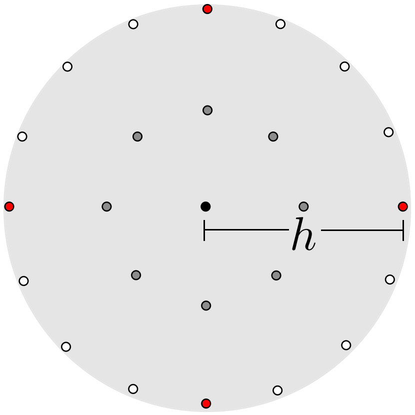
Discretizing to Ray Stencils



- Approximate filter by sampling the disc
- *h* is a distance in *screen space*
- Increase sampling density by packing more rings of samples
- Estimate *FGA* by *counting* which rays hit what
- Red samples form *finite difference stencil* for computing creases

Drawing Feature Lines

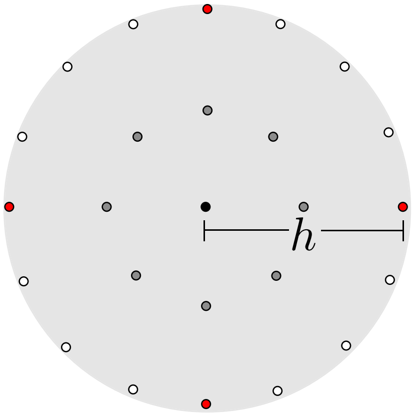
Discretizing to Ray Stencils



- Approximate filter by sampling the disc
- h is a distance in *screen space*
- Increase sampling density by packing more rings of samples
- Estimate *FGA* by *counting* which rays hit what
- Red samples form *finite difference stencil* for computing creases

Drawing Feature Lines

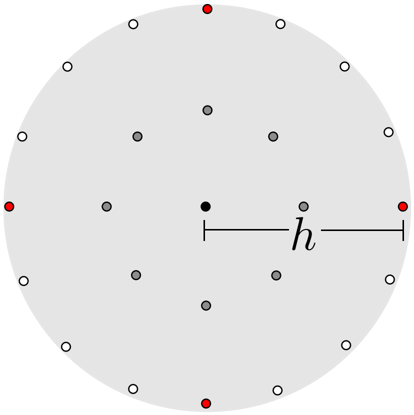
Discretizing to Ray Stencils



- Approximate filter by sampling the disc
- *h* is a distance in *screen space*
- Increase sampling density by packing more rings of samples
- Estimate *FGA* by *counting* which rays hit what
- Red samples form *finite difference stencil* for computing creases

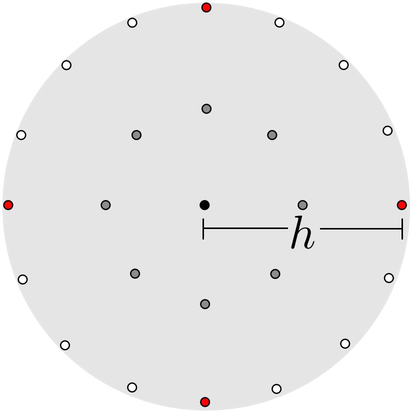
Drawing Feature Lines

Discretizing to Ray Stencils



- Approximate filter by sampling the disc
- h is a distance in *screen space*
- Increase sampling density by packing more rings of samples
- Estimate *FGA* by *counting* which rays hit what
- Red samples form *finite difference stencil* for computing creases

Drawing Feature Lines
Computing Edge Strength

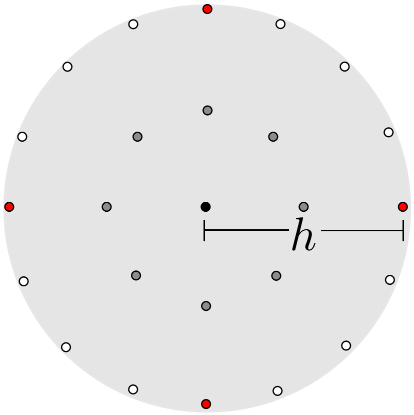


sample ray s (black), striking object g_s, surrounded by M stencil rays (gray, white, red)

- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla \vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays “far” from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

Computing Edge Strength

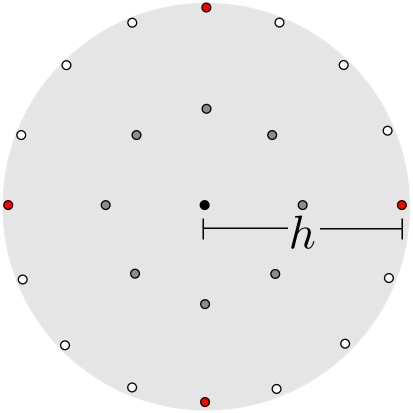


sample ray s (black), striking object g_s, surrounded by M stencil rays (gray, white, red)

- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla\vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays "far" from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

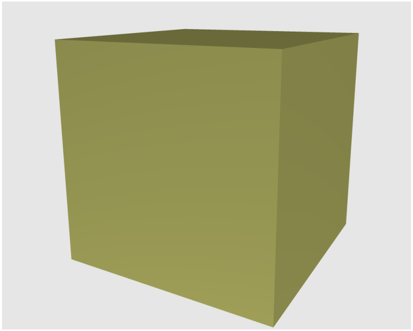
Computing Edge Strength



- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla \vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays “far” from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

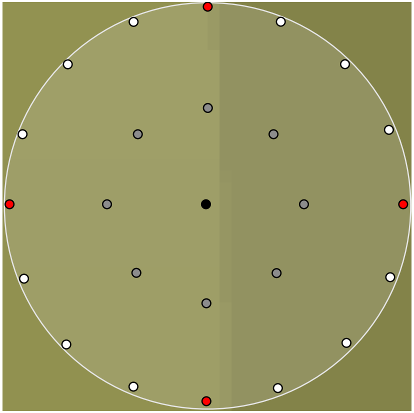
Computing Edge Strength



- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla\vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays "far" from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

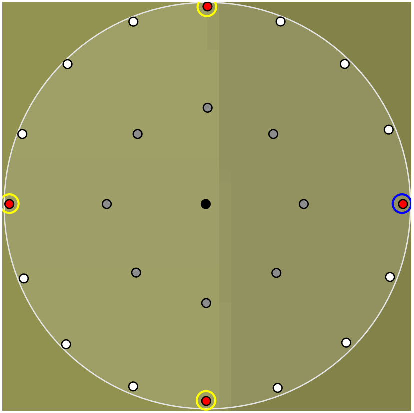
Computing Edge Strength



- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla\vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays "far" from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

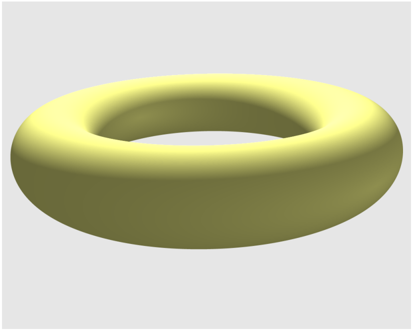
Computing Edge Strength



- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla\vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays "far" from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

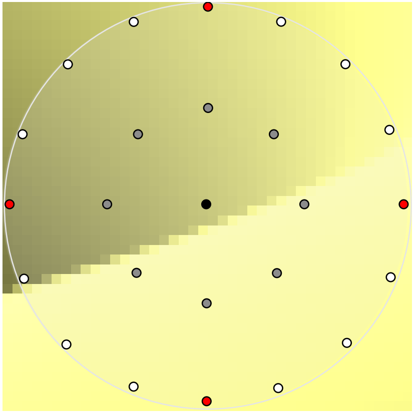
Computing Edge Strength



- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla\vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays "far" from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

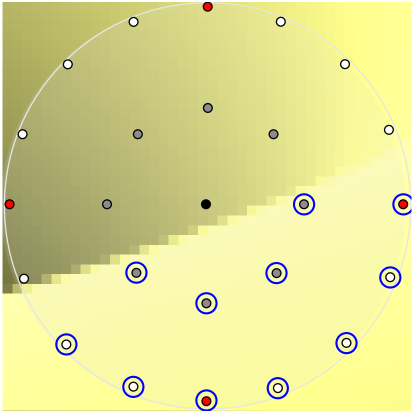
Computing Edge Strength



- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla\vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays "far" from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

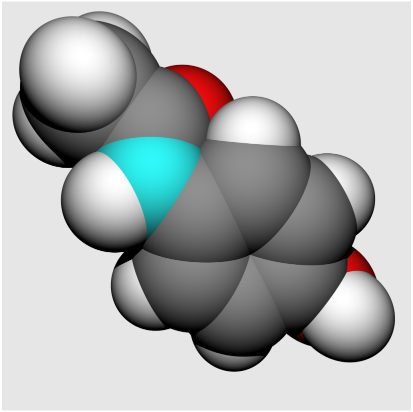
Computing Edge Strength



- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla\vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays "far" from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

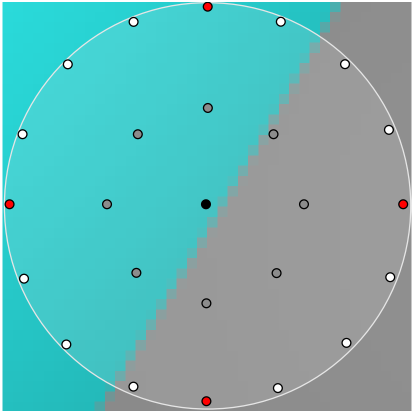
Computing Edge Strength



- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla\vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays "far" from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

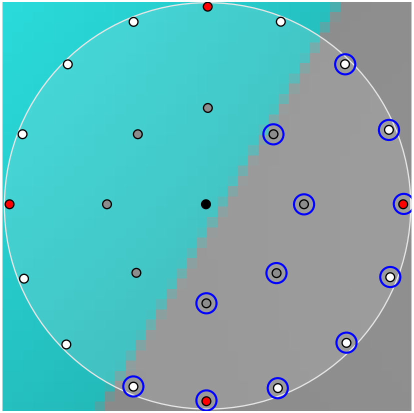
Computing Edge Strength



- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla \vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays “far” from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Drawing Feature Lines

Computing Edge Strength

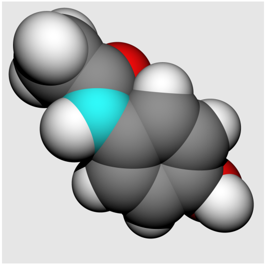


- Select *edge strength metric* E (e.g. $E(m) = \frac{m}{\frac{1}{2}M}$)
- if *entire stencil* hits object g_s,
  1. compute $\nabla\vec{n}$; if above threshold, edge strength $e_s = 1$, otherwise,
  2. d is the number of stencil rays "far" from the sample ray: $e_s = E(d)$
- otherwise, set $e_s = E(m)$, where m is the number of rays *not* striking g_s

Ray Tracing Feature Lines

For each sample:

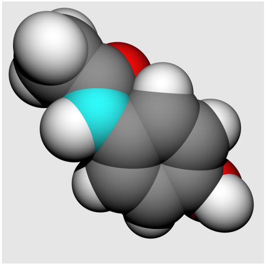
- Compute and shade *sample ray*

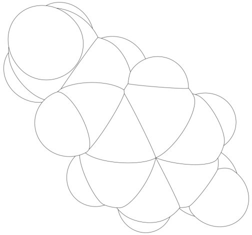


Ray Tracing Feature Lines

For each sample:

- Compute and shade *sample ray*
- Compute *edge strength*

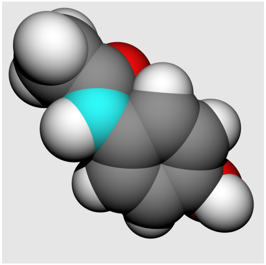
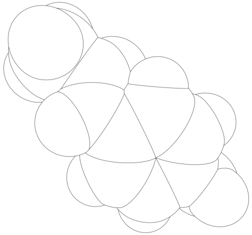
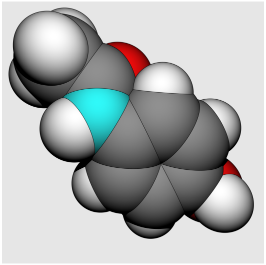




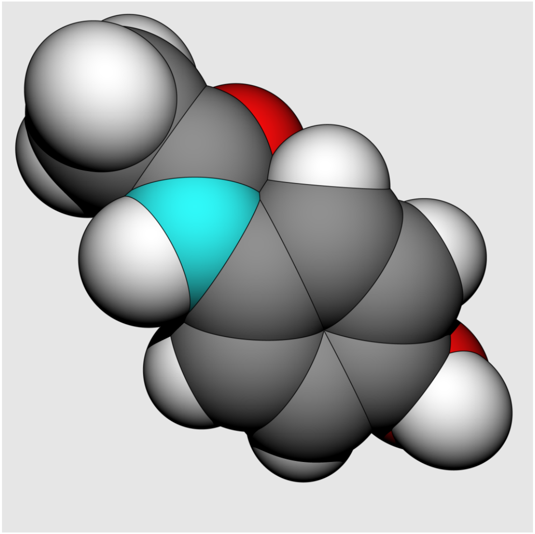
Ray Tracing Feature Lines

For each sample:

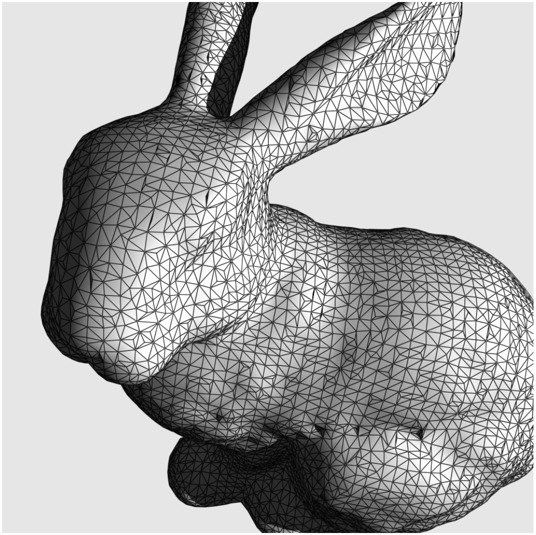
- Compute and shade *sample ray*
- Compute *edge strength*
- *Darken* shaded sample color according to edge strength ($e_s = 0$ results in sample color itself, $e_s = 1$ results in black)

Primitive Joints

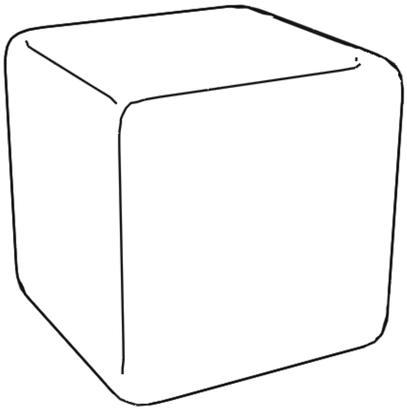


Mesh Visualization



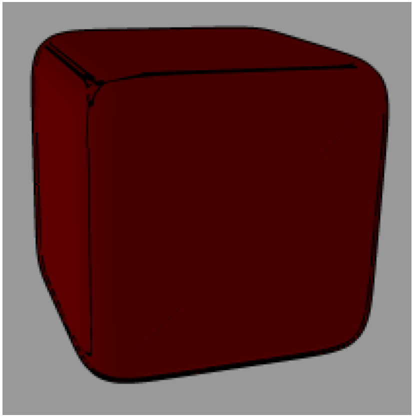
Application to NPR Techniques

Apparent Ridges (Judd et al. 2007)



- *Apparent ridges*: lines along which the view-dependent curvature attains a local maximum
- Followed algorithm in paper, adapting to ray stencils framework
- Reproduced "hooks" on corners
- Very faint ridge on front face, corresponding to slight bulge in model

Application to NPR Techniques
Apparent Ridges (Judd et al. 2007)



- *Apparent ridges*: lines along which the view-dependent curvature attains a local maximum
- Followed algorithm in paper, adapting to ray stencils framework
- Reproduced “hooks” on corners
- Very faint ridge on front face, corresponding to slight bulge in model

Thank You!

Amanatides, J.
1984.
Ray tracing with cones.
Computer Graphics 18, 3 (July), 129–135.

Dooley, D., and Cohen, M. F.
1990.
Automatic illustration of 3d geometric models: lines.
In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, Blah, 77–82.

Judd, T., Durand, F., and Adelson, E. H.
2007.
Apparent ridges for line drawing.
ACM Trans. Graph. 26, 3, 19.

Potter, K., Gooch, A., Gooch, B., Willemsen, P., Kniss, J., Riesenfeld, R., and Shirley, P.

2009.
Resolution independent npr-style 3d line textures.
Computer Graphics Forum 28, 1, 52–62.

Saito, T., and Takahashi, T.
1990.
Comprehensible rendering of 3-d shapes.
SIGGRAPH Comput. Graph. 24, 4, 197–206.