Ray Tracing II

Tomas Akenine-Möller

Modified by Ulf Assarsson

Department of Computer Engineering

Chalmers University of Technology

# Now, we have some basics in place Make ray tracing run faster!

- First, describe shadow cache
    - Simple idea, sometimes gives good results
- Second, spatial data structures
    - Huge improvements
- Third, (and not treated much here), cache-aware coding…
    - You may want to check out Ingo Wald's interactive ray tracing work on the web
    - Shoot rays according to a Hilbert Curve
        - See around middle of this lecture

Overview

- Shadow Cache
- Spatial data structures and ray traversal
  - Bounding volume hierarchies (BVHs)
  - BSP trees
  - Grids
- Materials
  - Fresnell Effect
  - Beer's Law
- Additional ray tracing techniques

The Shadow Cache

image plane

light source

shadow ray A

shadow ray B

ray B

ray A
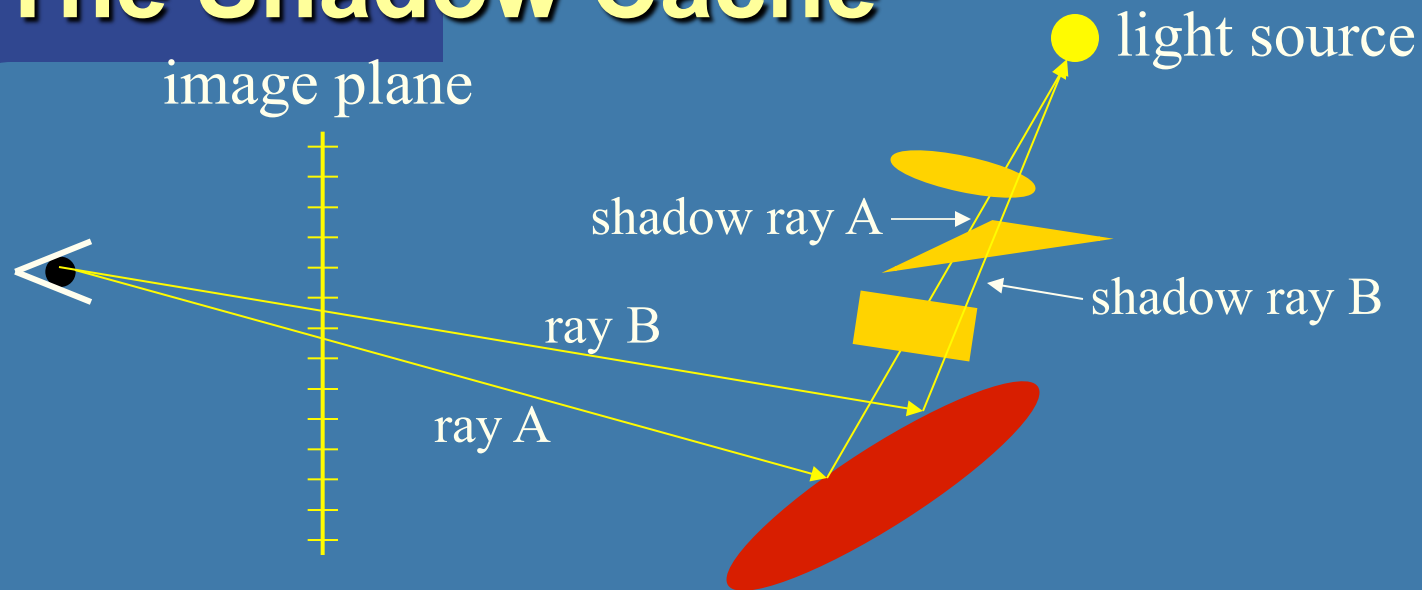
- It does not matter which object between the red ellipse and the light is detected
  - The point is in shadow if we find **one** object between
- Assume we first (A) find that the triangle shadows the red ellipse – store triangle in shadow cache
- For next ray B, start with testing object (triangle) in shadow cache
- If high coherence, then we'll get many hits in cache

# Spatial data structures and Ray Tracing

- Use these to get faster rendering
  - Ray tracing is often slow
  - Not useful for very few primitives (e.g., 2 spheres)
- The point:
  - You don't need to test for intersection between the ray and all objects in the scene
  - Rather, you test a small subset
  - Typically, $O(\log n)$ instead of $O(n)$ for each ray
- We will look at
  - Bounding volume hierarchies (BVHs)
  - BSP trees
  - Grids

Bounding Volume Hierarchy (BVH)

- We'll use axis-aligned bounding boxes (AABBs) here

- The goal: find closest (positive) intersection between ray and all objects in the scene

- Simple: traverse the tree from the root

- If the ray intersects the AABB of the root, then continue to traverse the children

- If ray misses a child, stop traversal in that subtree
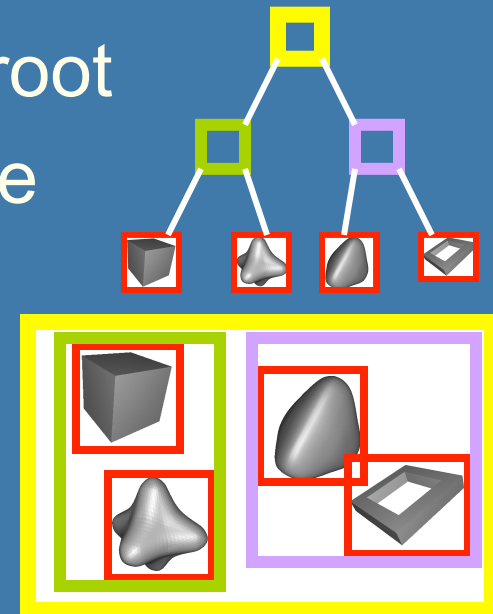
Example: ray against BVH
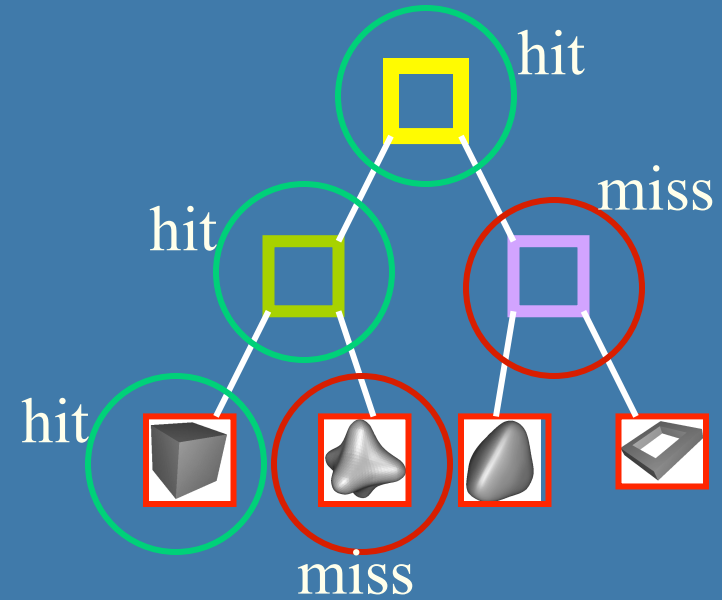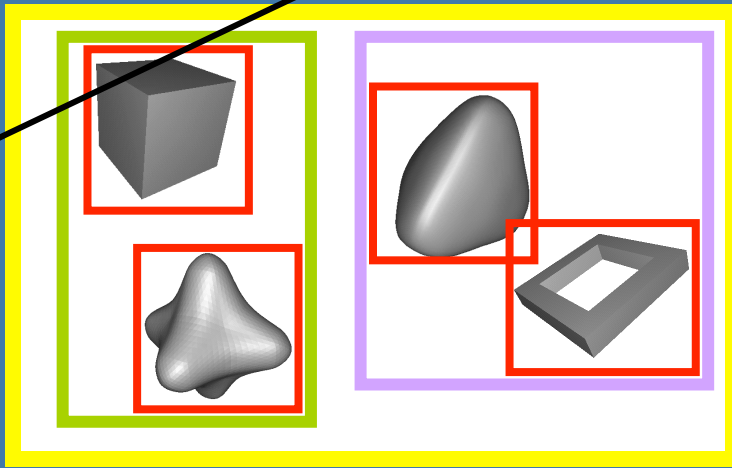
- Without BVH, we would test each triangle of every object against the ray
- With BVH:
  - Only test the triangles of the cube against the ray
  - Plus some AABBs, but these are cheap

Optimizations

- Always make a reference implementation
  - And save it for benchmarking!
- Benchmarking is key here:
  - Not all "optimizations" yield better performance
  - However, this definitely depends on what kind of scene you try to render
- Preprocessing is good
  - Use when possible
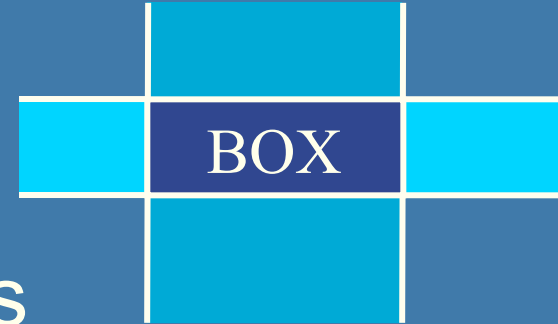
BVH traversal optimizations

1. Use current closest intersection as an upper bound on how far the ray may "travel"

   - Example, if a ray hits a polygon at distance t_{box}, then we don't need to traverse a BV which is located farther than t_{box}.

2. Can also sort the BVs with respect to distance, and only open up if necessary

3. Shadow cache can be used for shadow rays

4. Shading cache: store what object the reflection and refraction ray hit last time, and for next reflection ray, test this object first → if hit, then a good upper bound on distance, and so we can discard everything beyond this distance

A comment on previous slide

- Smits (ref to left) has noted that some of the presented optimizations were not useful any longer

- Reason: architectural changes in computers

- Bottleneck is often in memory accesses, not in floating point operations

- Lesson: use benchmarks to test if your optimizations gives better performance for a couple of scenes

AABB hierarchy optimization

- An AABB is the intersection of three slabs (2 in 2D)

  BOX

- Observation: all boxes' slabs share the same plane normals
- Exploit this for faster AABB/ray intersection!
- AABB/ray needs to compute one division per x,y, and z
 - Precompute these once per ray, and use for entire AABB hierarchy

BVH traversal… optimizations that really work!

- Standard (depth-first) traversal is slow:
  - Involves recursion
  - And memory may be allocated once per node
- Left-child, right-sibling, parent pointers avoids recursion
  - Instead follow pointers

- Store these in a clever way, with skip pointers
  - Store in depth first order
  - A skip pointer, points to the place where traversal shall continue given a miss
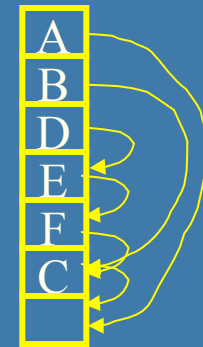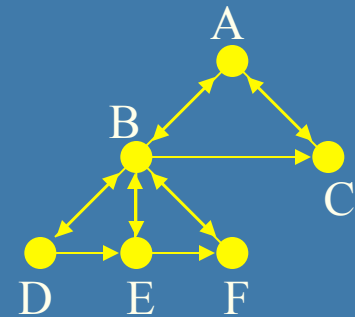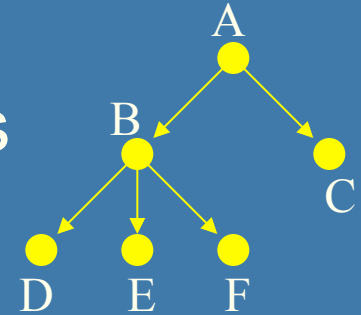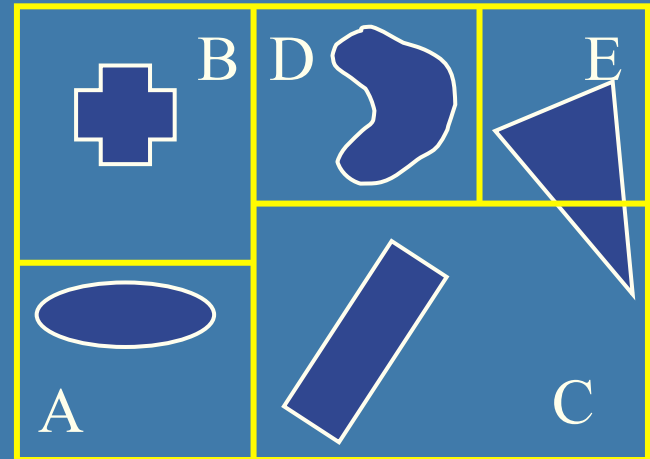
BVH traversal optimization (2)

- If no miss, continue in depth first order
- If nodes are allocated linear in memory, the we can expect many cache hits

- Read the paper (see course website)
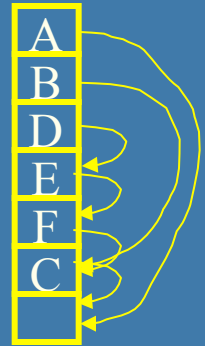
Axis-Aligned BSP trees

- Not that different from BVHs with AABBs
- An advantage here is that we traverse the space in a rough sorted order along the ray
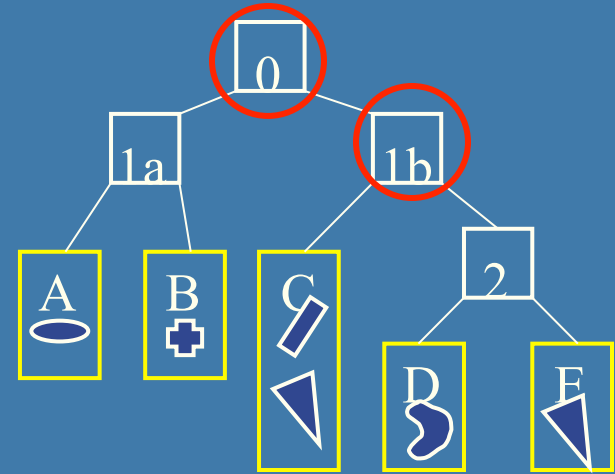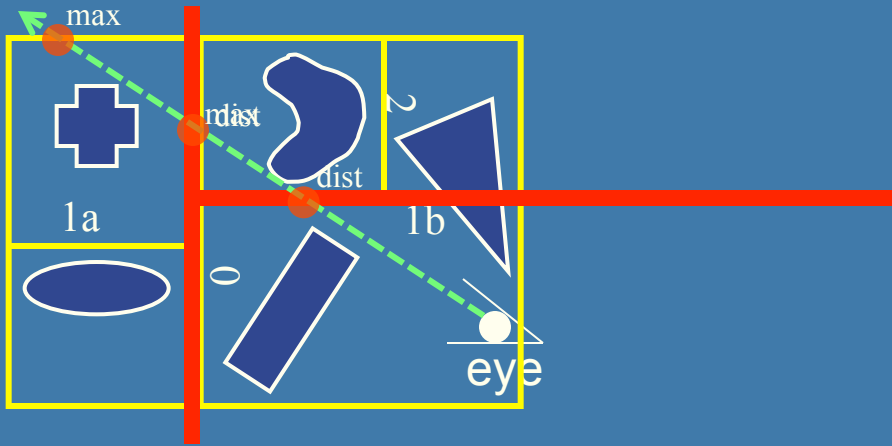- Pretty simple code as we will see

If we have a fixed order for the splitting dimension (e.g. x,y,z,x,y,z… or z,x,y,z,x,y… etc) this is called a kD-tree.

Axis-aligned BSP tree against ray

```
RayTreeIntersect(Ray, Node, min, max)
{
    if(node==NULL) return no_intersection;
    if(node is leaf)
    {
        test all primitives in leaf, discard if not between min and max;
        return closest intersection point if any;
    }
    dist = signed distance along Ray to cutting plane of Node;
    near = child of Node that contains ray origin;
    far = child of Node that does not contain ray origin;
    if(dist>0 and dist<max)           // interval intersects plane of Node
    {
        hit=RayTreeIntersect(Ray,near,min,dist);      // test near side
        if(hit) return hit;
        return RayTreeIntersect(Ray,far,dist,max);  // test far side
    }
    else if(dist>max or dist<0)     // whole interval is on near side
        return RayTreeIntersect(Ray,near,min,max);
    else                                 // whole interval is on far side
        return RayTreeIntersect(Ray,far,min,max);
}
```

AA-BSP Tree Traversal

- Test the planes against the ray
- Test recursively from root
- Continue on the "hither" side first

```
RayTreeIntersect(Ray, Node, min, max){
    if(node==NULL) return no_intersection;
    if(node is leaf)
        test all primitives in leaf, discard if not between min and max;
        return closest intersection point if any;
    dist = signed distance along Ray to cutting plane of Node;
    near = child of Node that contains ray origin;
    far = child of Node that does not contain ray origin;
    if(dist>0 and dist<max)           // interval intersects plane of Node
        hit=RayTreeIntersect(Ray,near,min,dist);    // test near side
        if(hit) return hit;
        return RayTreeIntersect(Ray,far,dist,max);  // test far side
    else if(dist>max or dist<0)                      // whole interval is on near side
        return RayTreeIntersect(Ray,near,min,max);
    else    return RayTreeIntersect(Ray,far,min,max);  // whole interval is on far side
}
```

Grids

- A large box is divided into a number of equally-sized boxes

- Each grid box stores pointers to all objects that are inside it

- During traversal, only the boxes that the ray intersect are visited, and objects inside these boxes are tested
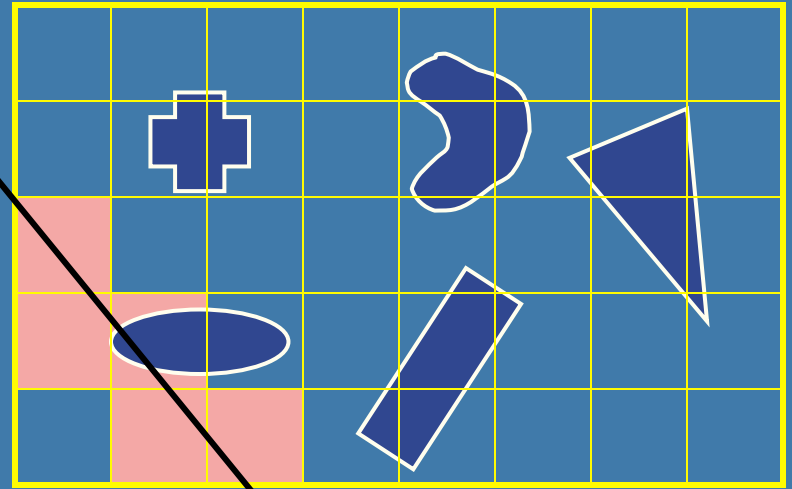
Grid Traversal Algorithm

- A modified line generating algorithm can be used
  - Bresenham or DDA
- But easier to think in geometrical terms
  - Red circles mark where ray goes from one grid box to the next
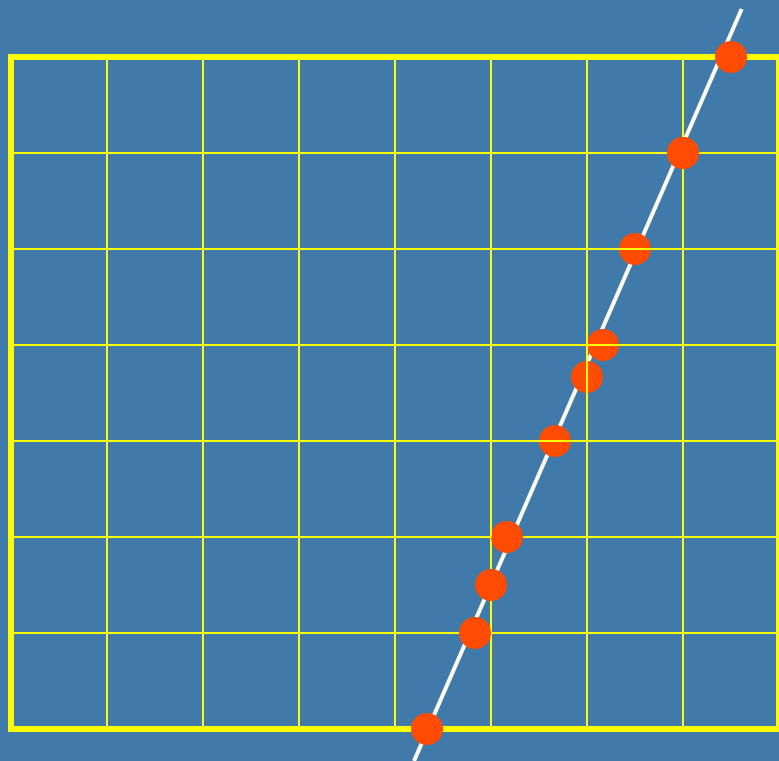
Intersection points appear with irregular spacing ←

But, look first at only intersection with horizontal lines, then vertical →

These are regular spaced! Use that in implementation
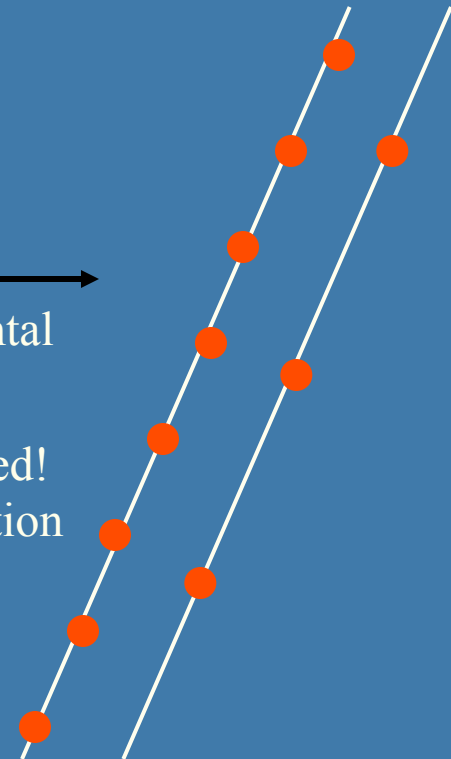
Traversal example

```
loop
    if(tNextX < tNextY)
        tNextX += tDeltaX;
        X= X + stepX;
    else
        tNextY += tDeltaY;
        Y= Y + stepY;
    NextVoxel(X,Y);
```

tNextX

tNextY

ray origin

At start, compute tNextX and tNextY
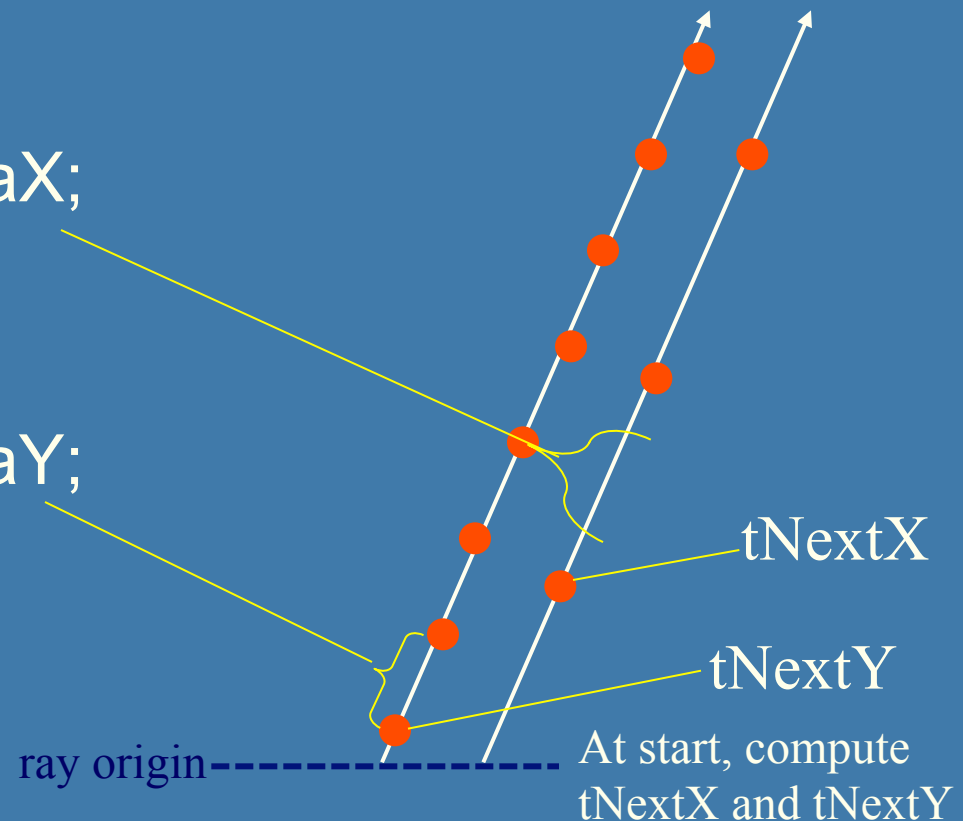
tNextX = t-value at next x increment

tNextY = t-value at next y increment

stepX/Y = ± 1

19

Grid Traversal (2)

- Easy to code up,
- Check out the following paper (for those who want to implement in their project):
    - Amantindes and Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing", Proc. Eurographics '87, Amsterdam, The Netherlands, August 1987, pp 1-10.
- Available on course website

Testing the same object more than once in grids

- If an object intersects more than one grid box, and a ray traverses these, then you can test the same object twice

- Solution: assign a unique rayID to each ray, and for each tested object, store the rayID with the object.

- If rayID of ray and object are the same, then we have already tested the object. So then just fetch the hitpoint, stored with the object
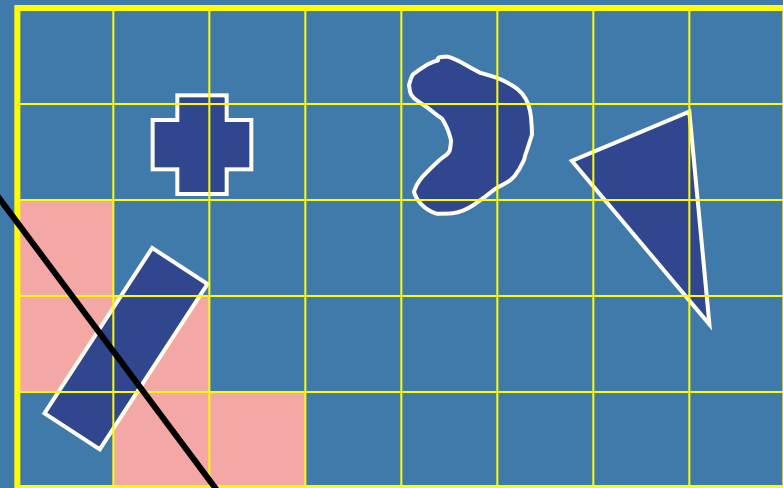
This is called mailboxing

Choose a good grid resolution

- Assume n objects in scene, g is grid resolution

$$g = \sqrt[3]{n}$$

- Only good for cubes!

- Better to have different number of grid boxes per side
- Let the number of grid boxes per side be proportional to the length of the box side
- See Klimaszewkski and Sederberg, in IEEE Computer Graphics & Applications, Jan-Feb, 1997, pp. 42—51.

Hierarchical and Recursive Grids

- We always use hierarchies in CG, so do that now as well

- When a grid box (voxel) contains many primitives, introduce a smaller grid in that grid box
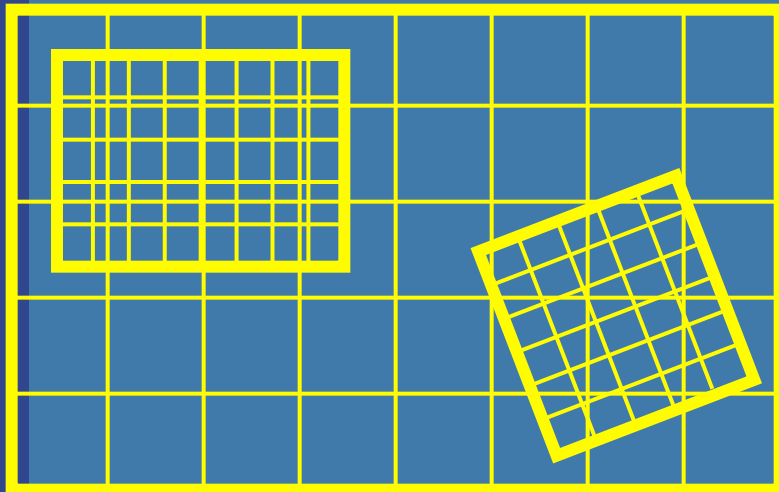
Hierarchical grid
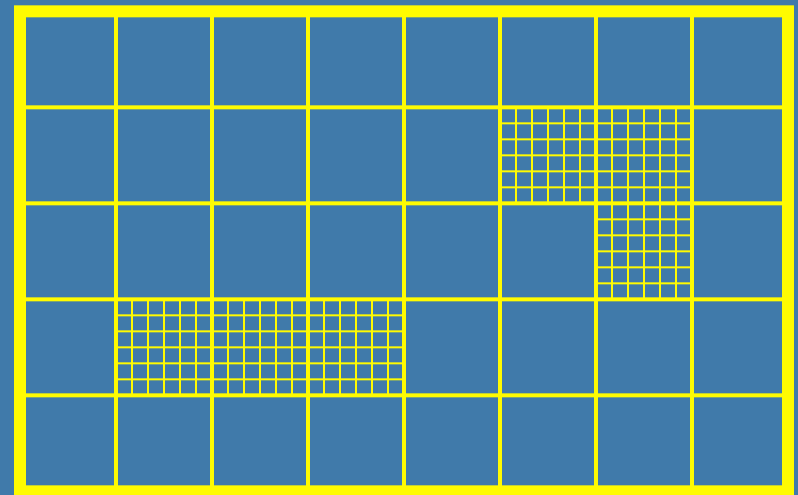
Recursive grid

Which spatial data structure is best?

- Depends on implementation, the type of scene, how complex shading, etc, etc.

- Make timings on several different algorithms and choose what works best for you

- Kd-trees:
    - Fastest to traverse, little memory, slow to build

- AABB-hierachies:
    - Fast to build, slower to traverse (not automatically in order along ray

- Grids
   - Fast to build, middle fast to traverse, typically needs to be hierarchial/recursive

Cache awareness

- To maximize cache locality, you can utilize that the next ray likely will access roughly the same memory locations since it will traverse roughly the same part of the tree and geometrical objects.

  - To maximize spatial locality, shoot the rays according to a Hilbert curve, instead of sequentially scanline by scanline.

Hilbert Curve
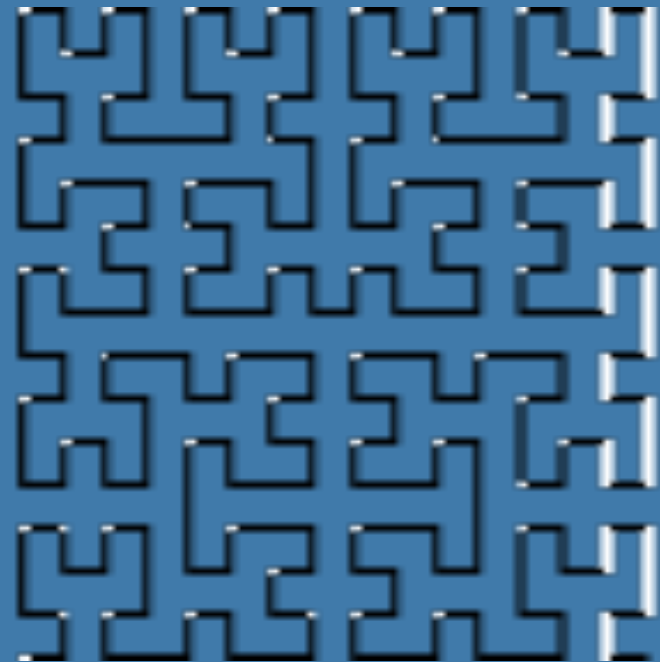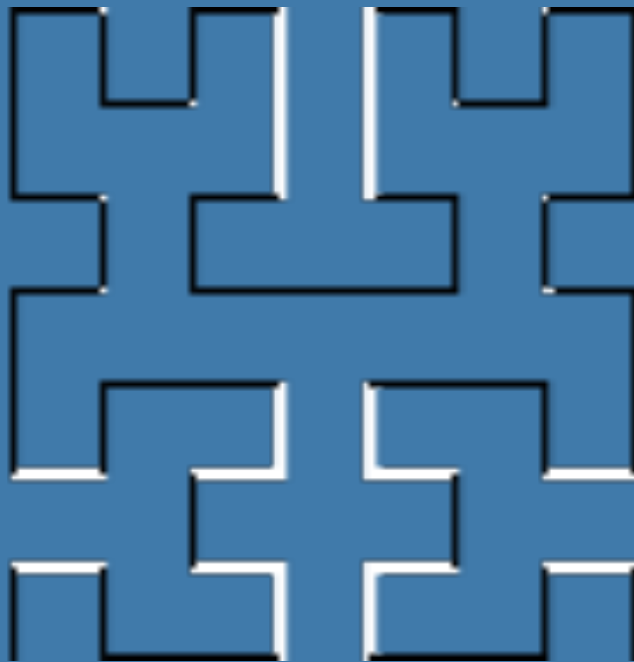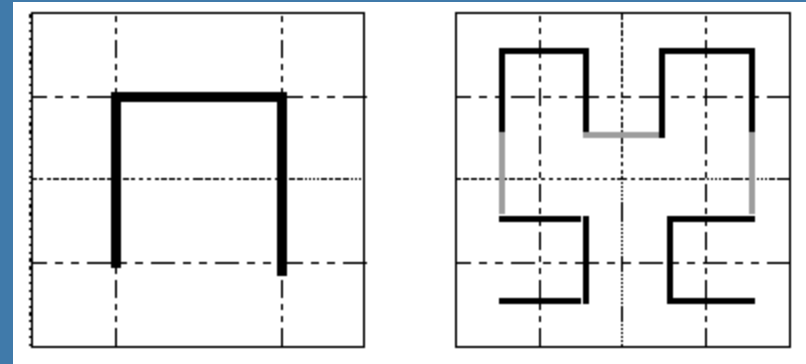
Faster Grid Traversal using Proximity Clouds/Distance Fields
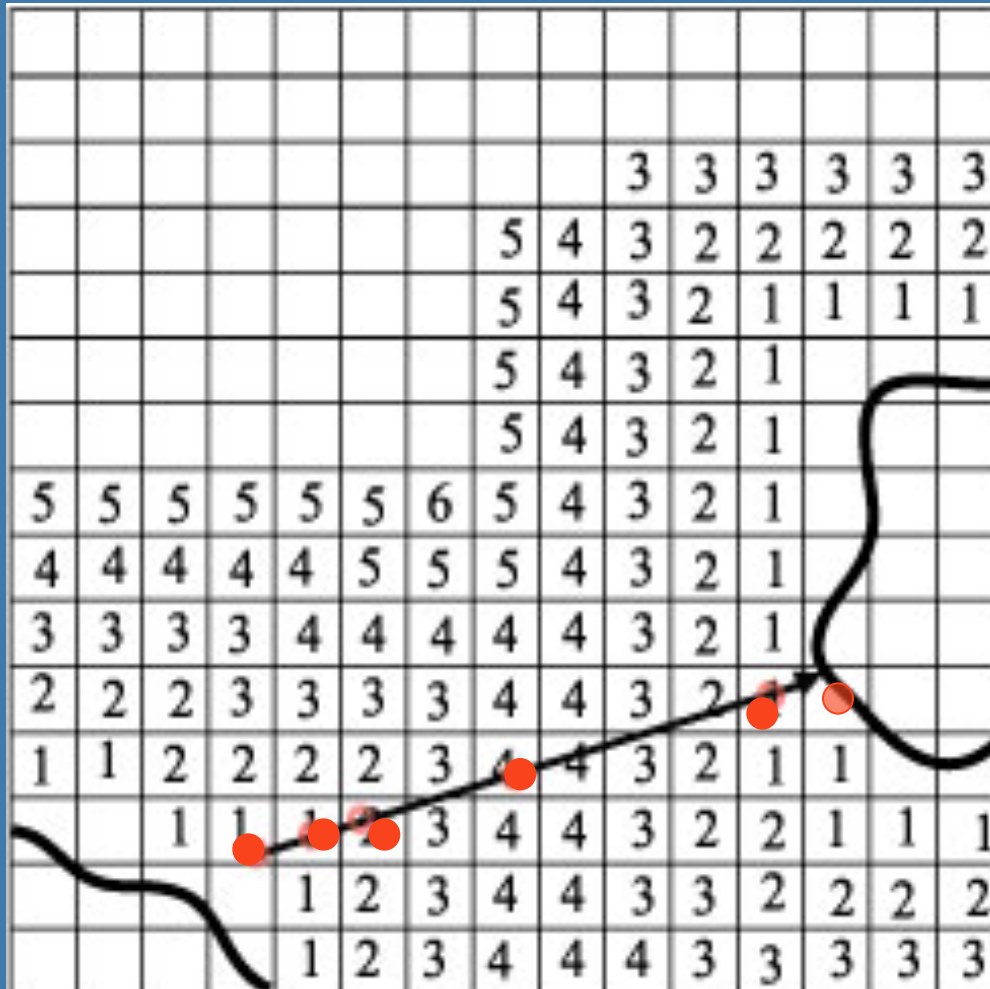
"Proximity Clouds – An Acceleration Technique for 3D Grid Traversal", Daniel Cohen and Zvi Sheffer

Demo
using SSE

MATERIALS

- Types of material, and how light interacts
  - Glass, plastic... (dielectrics)
  - Metal (conductive)

Smooth Metal
(slät metall)

- Often used material, and well-understood in computer graphics

- We'll present a good approximation here

- Metals obey three "laws":
  - The highlight often has the same color as the diffuse
  - Law of reflection
  - The Fresnel equations:
    How much is reflected and how much is absorbed
  - Though, Fresnel effect for metals is subtle
  - Higher for dieletric materials

Smooth metals (2)

plastic   metal

Types of highlights

- Highlight
- The law of reflection
- If the metal is smooth, we can say that it reflects perfectly in the reflection direction
- Fresnel equations depend on
  - Incident angle
  - Index of refraction (chromium oxide: 2.7)

- Can compute polarized, and unpolarized values for the light (in CG, we ignore polarization, often)
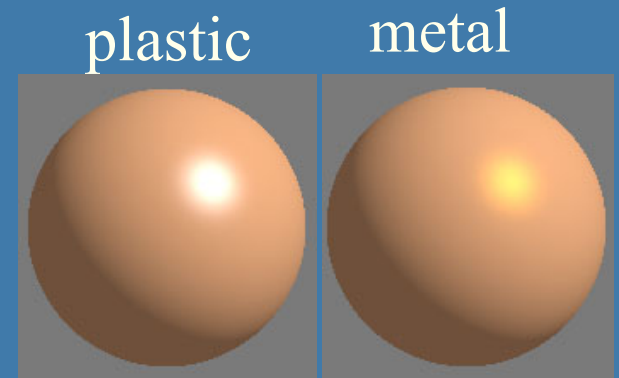
Smooth metals (3) Fresnel

- *F* is describes the reflectance at a surface at various angles (*n*=index of refraction)

$$F = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left(1 + \frac{[c(g+c)-1]^2}{[c(g-c)+1]^2}\right)$$

$$g = \sqrt{n^2 + c^2 - 1}$$

$$c = \mathbf{v} \bullet \mathbf{n}$$

- Set refraction index=1.5, then you get:

Metals
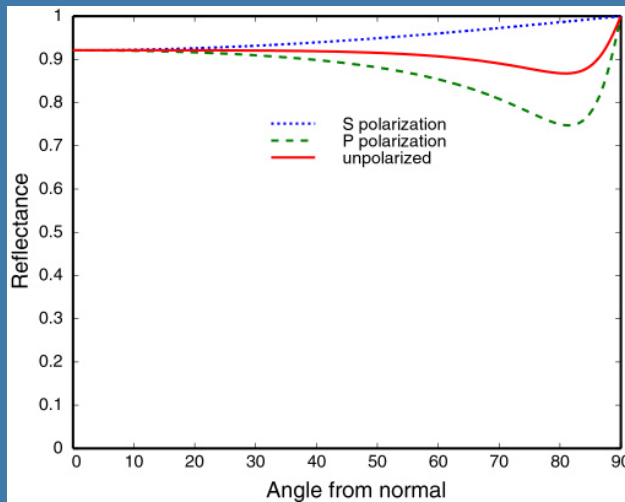
Images courtesy of Steve Westin
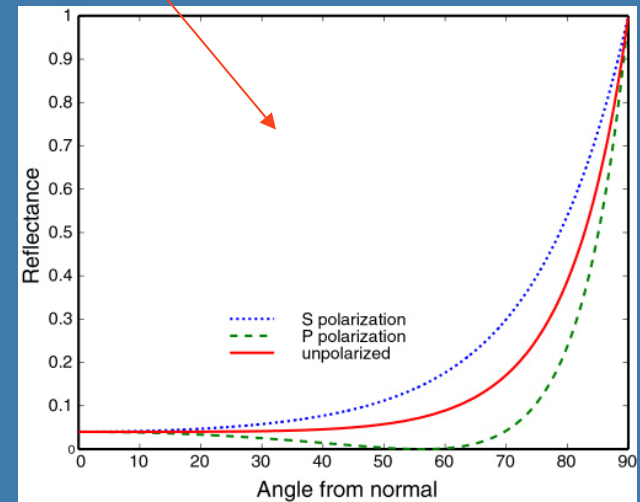
Dielectrics

An approximation to Fresnel
(by Schlick)

$$F \approx R_0 + (1 - R_0)(1 - \mathbf{v} \cdot \mathbf{n})^5$$

- $\mathbf{v}$ is the vector from the eye to the point on the surface
- $\mathbf{n}$ is the surface (micro-) normal
- R_0 is the reflectance when $\mathbf{v} \cdot \mathbf{n}$=1
- Works well in practice

- Use F for your reflection rays in shading:
  - F*`trace`(reflection_vector)

Fresnel, example

- What does it look like
- A black dielectric sphere (glass or plastic)
  - in computer graphics, glass can be black
- Which is which?
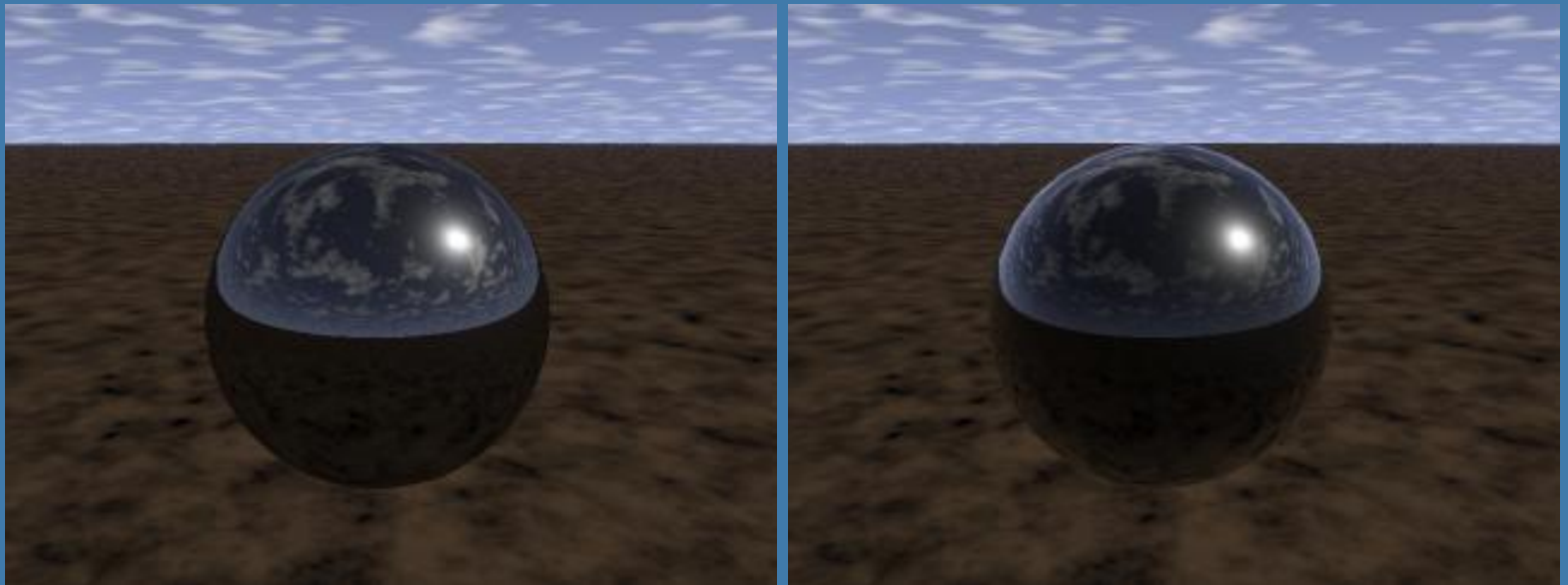
Images courtesy of Steve Westin, Cornell University

Smooth dielectric materials

- A dielectric is a transparent material
- Refracts light
- Filters light (due to impurities in material)
- Examples:
  - Glass = 1.5
  - Diamond = 2.4
  - Water = 1.33
  - Air = 1.0

The following 17 slides are from an excellent presentation at Microsoft Meltdown 2004,

"Advanced Real-Time Reflectance",

by Dan Baker, Naty Hoffman and Peter-Pike Sloan

The corresponding paper can be found here:

www.gdconf.com/archives/2004/hoffman_naty.pdf

Smooth Dielectric

- **Low reflectance (water, glass, plastic, etc. ~5%)**

- **Refracted light continues inside the material, being scattered by impurities until it is absorbed or re-exits the surface**
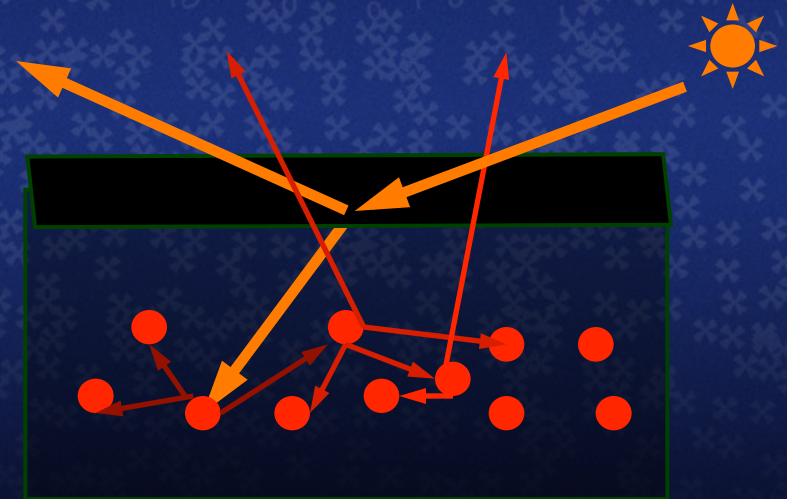
From Advanced Real-Time Reflectance, by Dan Baker, Naty Hoffman & Peter-Pike Sloan

Diffuse / Specular Tradeoff

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough (Glossy)

- **Most surfaces are not flat at all scales**

Semi-Rough (Glossy)

- **Most surfaces are not flat at all scales**
  - **Many surfaces which appear flat at visible scales have complex *microscale* structure**
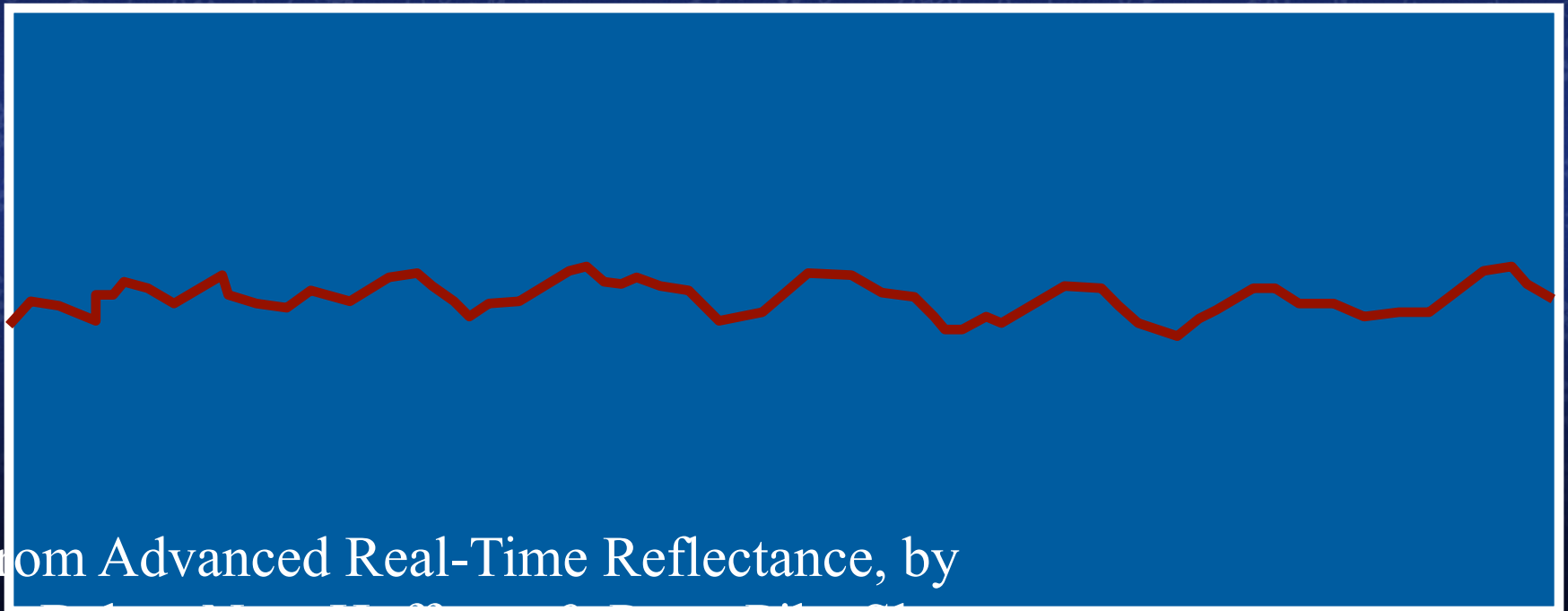
From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough (Glossy)

- **Most surfaces are not flat at all scales**
  - **Many surfaces which appear flat at visible scales have complex *microscale* structure**
  - **At the smallest scale we can often treat the surface as flat again**

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough (Glossy)

- A surface patch contains micro-facets with continuously distributed normals

- Light reflects off facets, 'spreads out'

- In 'semi-rough' surfaces distribution of micro-normals biased to macro-normal

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan
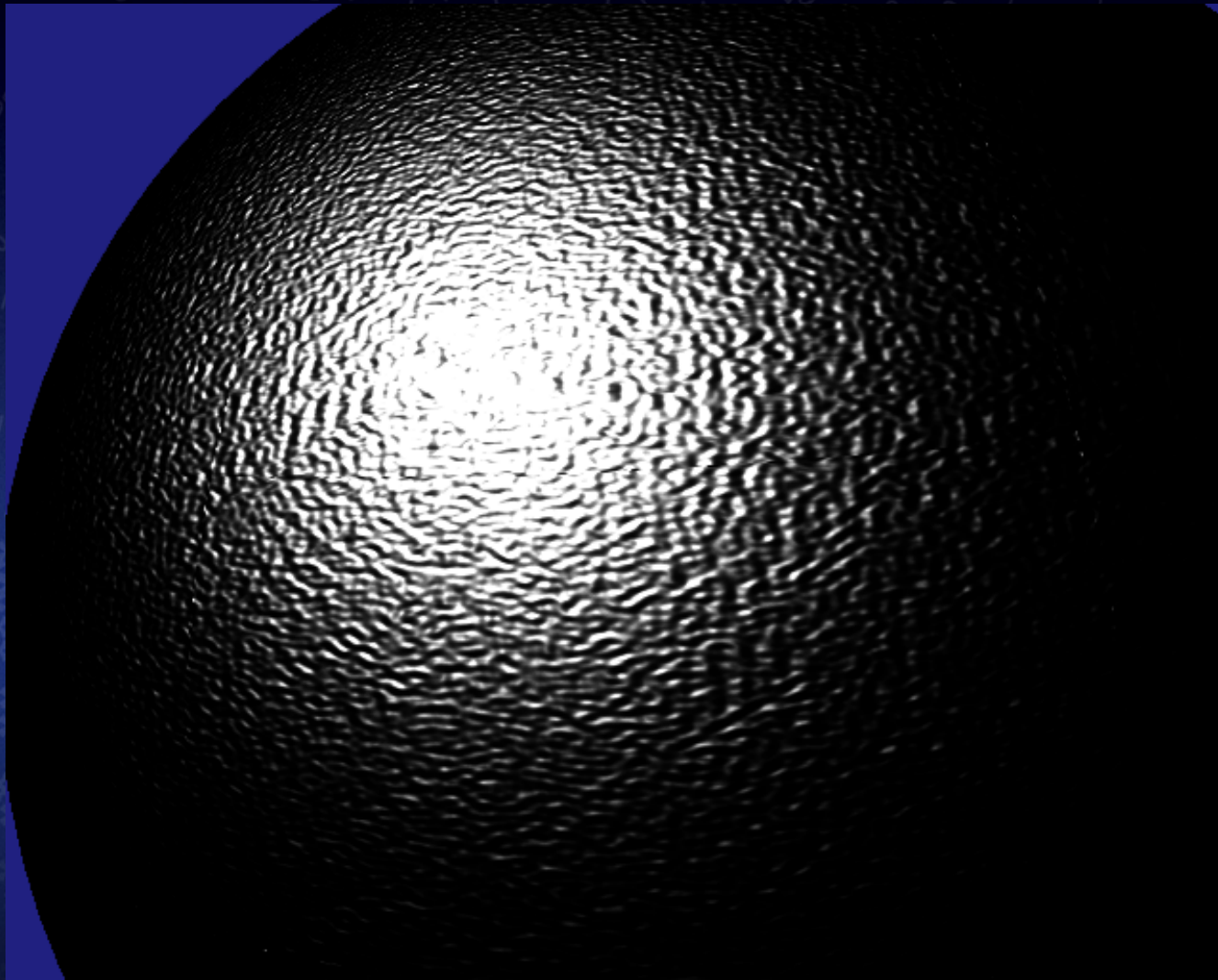
Semi-Rough (Glossy)

- A surface patch contains micro-facets with continuously distributed normals

- Light reflects off facets, 'spreads out'

- In 'semi-rough' surfaces distribution of micro-normals biased to macro-normal

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Microsoft
DIRECTX®

Semi-Rough (Glossy)

From Advanced Real-Time Reflectance, by
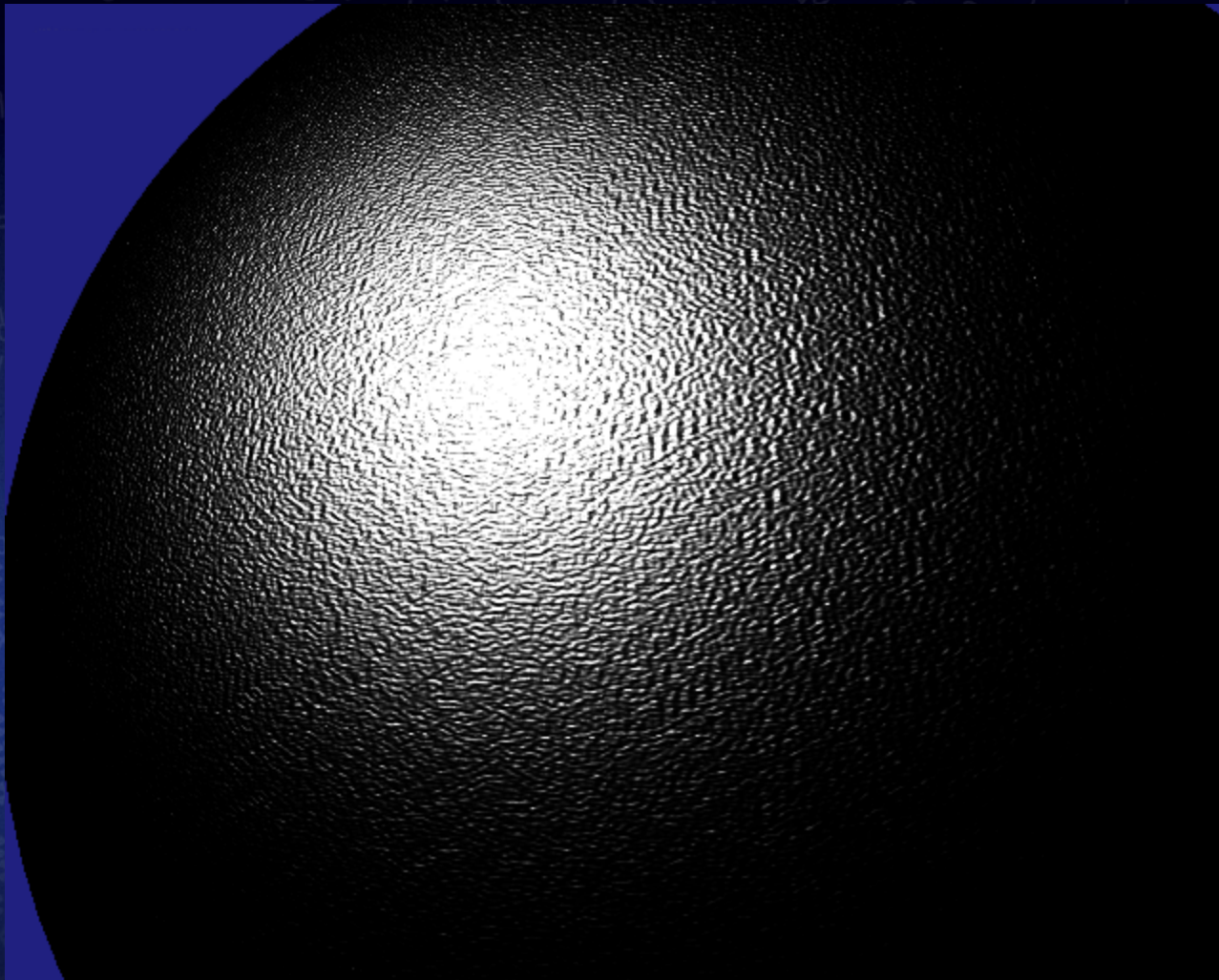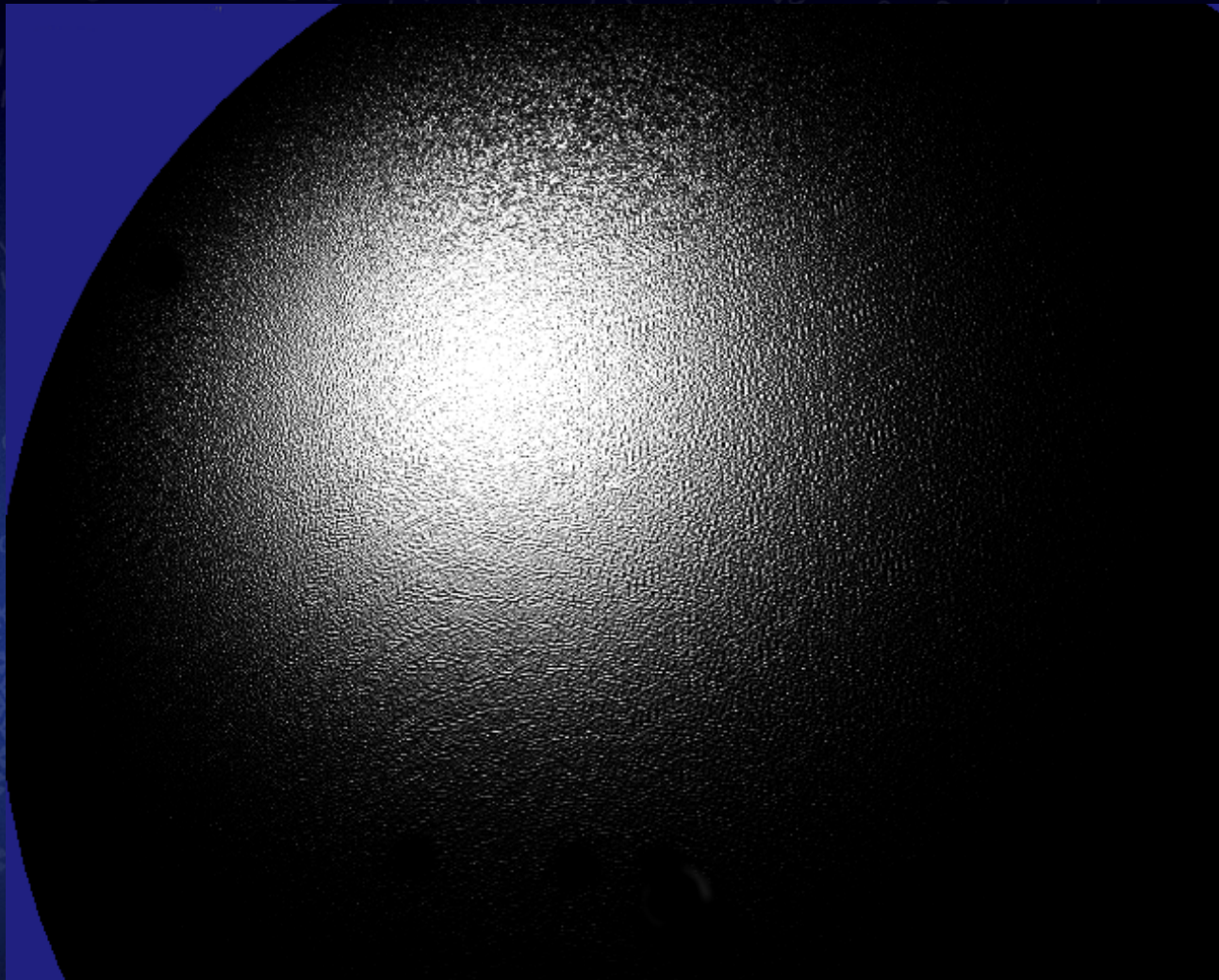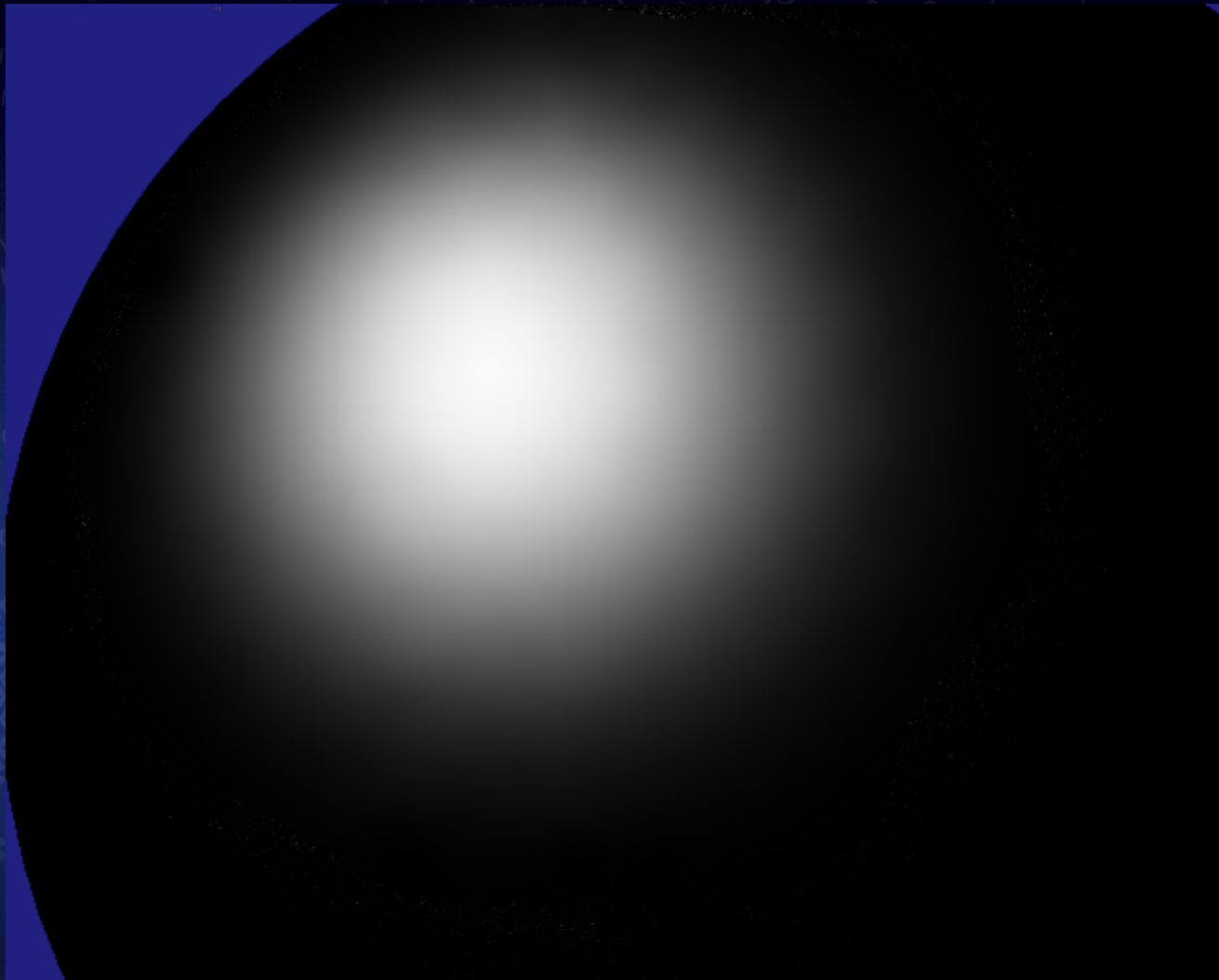Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough (Glossy)

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough (Glossy)

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough (Glossy)

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough (Glossy)

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough (Glossy)

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough (Glossy)

- Lights reflect as highlights
- Blurry reflection of environment

From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough Metal

- **The reflectance of a metallic rough surface is dominated by the highlight**
- **Multiple-bounce reflections create an additional, more diffuse reflection**
  - **More strongly colored since each bounce increases the colors saturation**

From Advanced Real-Time Reflectance, by Dan Baker, Naty Hoffman & Peter-Pike Sloan

Semi-Rough Dielectric

- **Highlight tend to be weaker in dielectric surfaces due to lower reflectance**
  - **For the same reason, multiple-bounce reflections are less noticeable**
  - **Diffuse mostly due to subsurface scattering**

Microsoft®
DIRECTX®

Rough Dielectric

- **Normal distribution is extremely random**

- **Almost uniformly diffuse with some retroreflection**
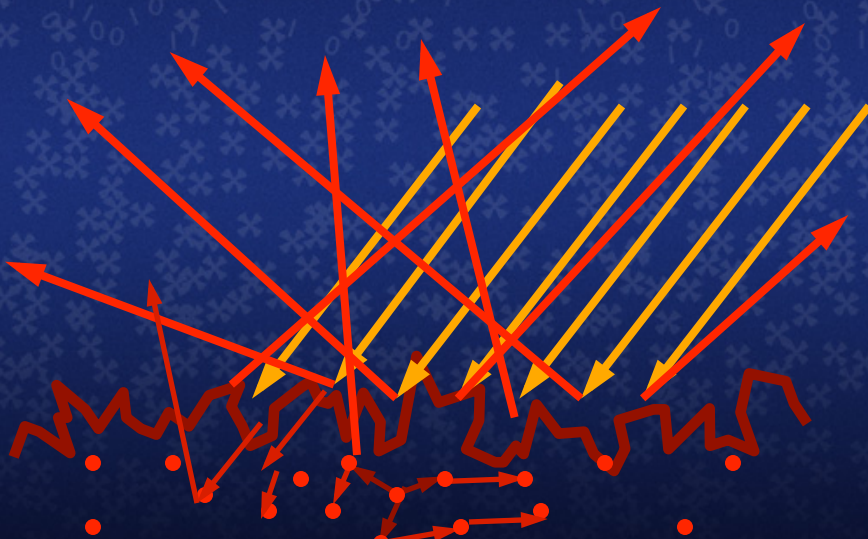
From Advanced Real-Time Reflectance, by
Dan Baker, Naty Hoffman & Peter-Pike Sloan

Smooth dielectric materials (2) Homegeneous impurities

- Light is attenuated with Beer's law
- Loses intensity with: $dI = -C\ I\ ds$
- $I(s) = I(0)e^{-Cs}$
- Compute once for each RGB
- Also, use the Fresnel equations for these materials

Beer's Law

Constant intensity decrease at greater distance due to absorption.

$$dI = -CIds$$

$$I(s) = I(0)e^{-C*s}$$

$I(0)$

ds

$I(s)$
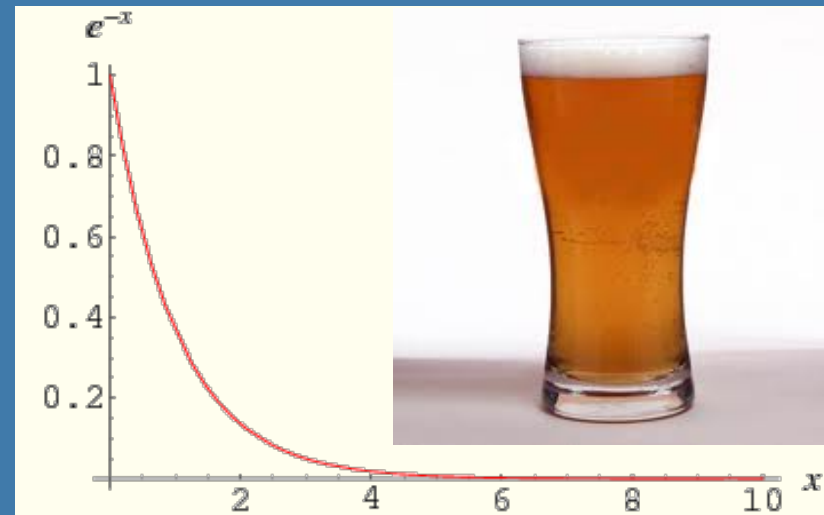
s
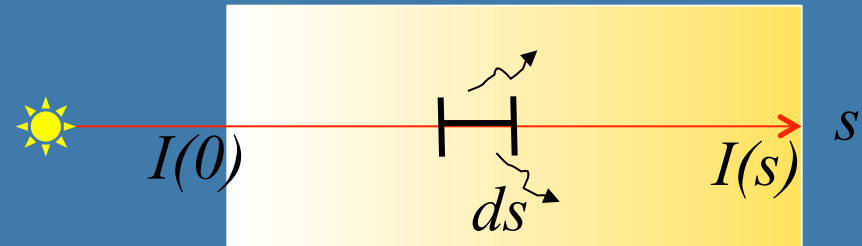
Beer's law

The taller the glass, the darker the brew,
The less the amount of light that comes through

RAY TRACING ADDITIONALS

What more can we do with ray tracing?

- A million things…
- Geometrical objects, etc.
- Optical effects
- Speed-up techniques
- Animation

Geometry

- Perfect for object-oriented programming
  - Makes it simple to add new geometrical objects
- Add a transform to each object
- The standard trick is not to apply the transform matrix to the object, but instead inverse-transform the ray
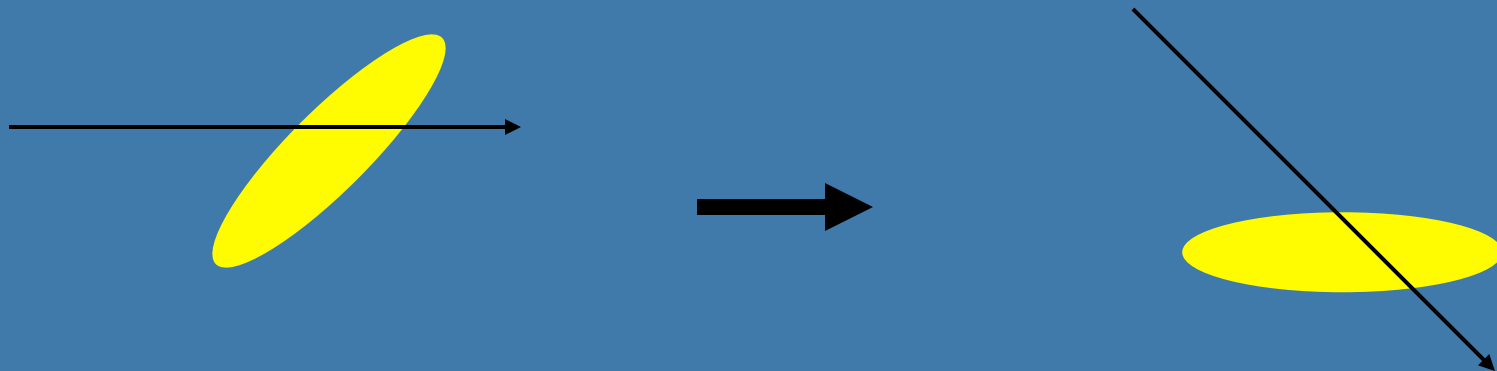
Geometry: Constructive Solid Geometry (CSG)

- Boolean operations on objects
  - Union
  - Subtraction
  - Xor
  - And
- Simple to implement
- Must find *all* intersections with a ray and an object
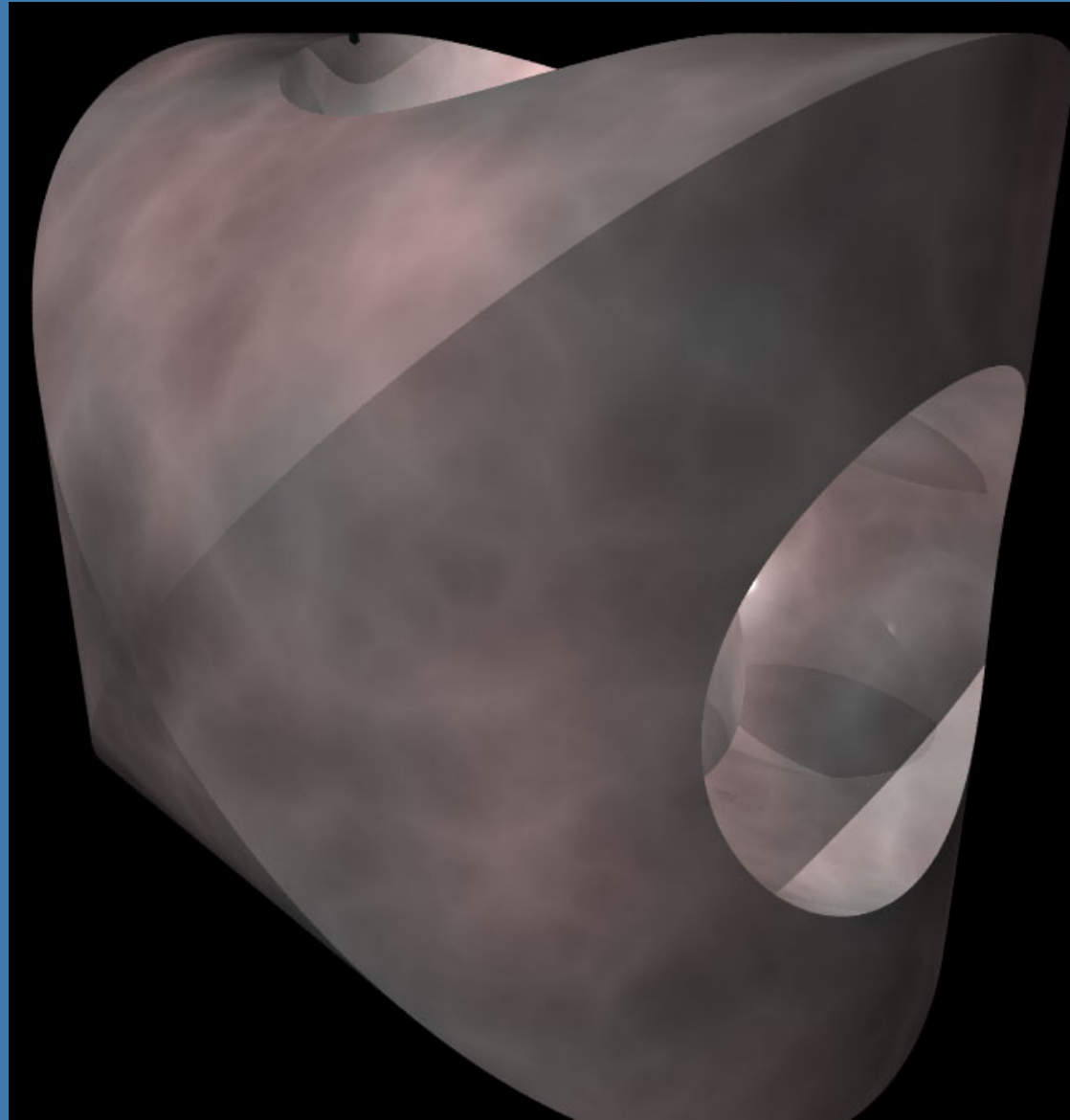- Then do this for involved objects, and apply operators to found interval

58

Geometry: Constructive Solid Geometry (CSG)

- Examples, operators:
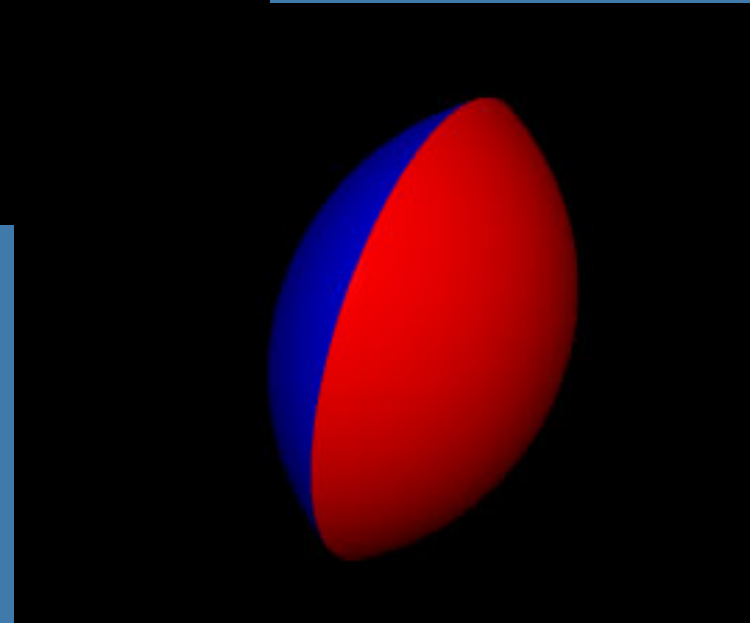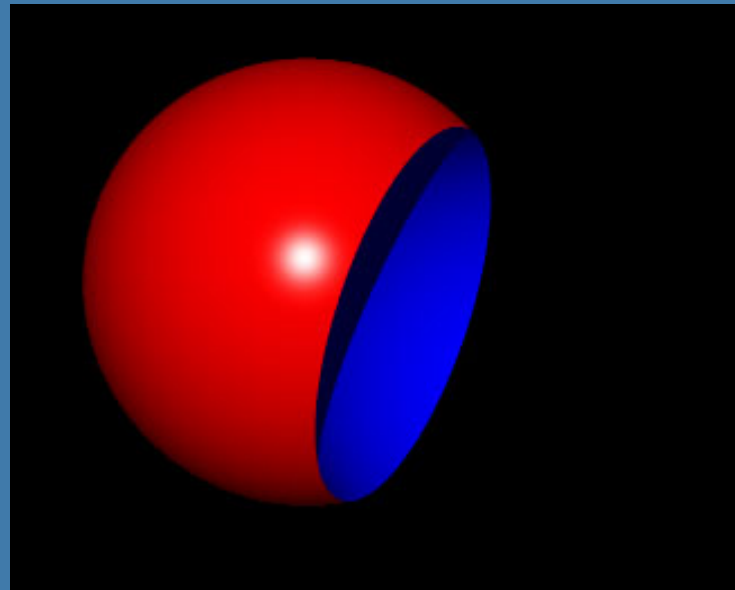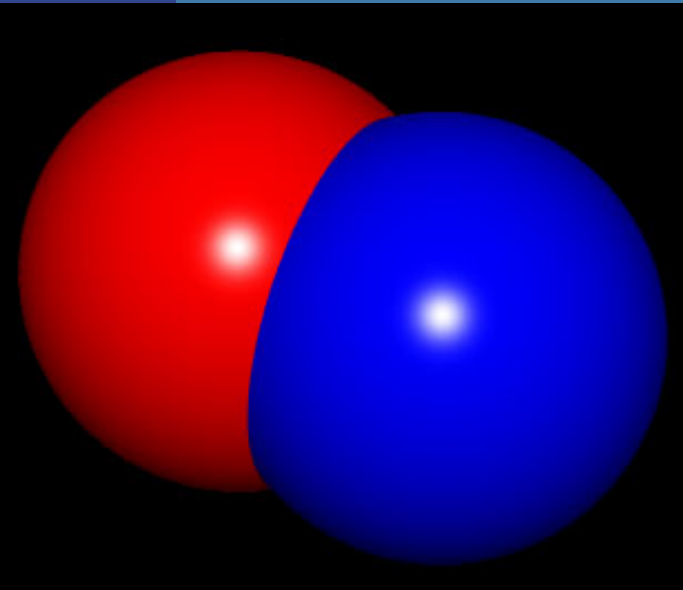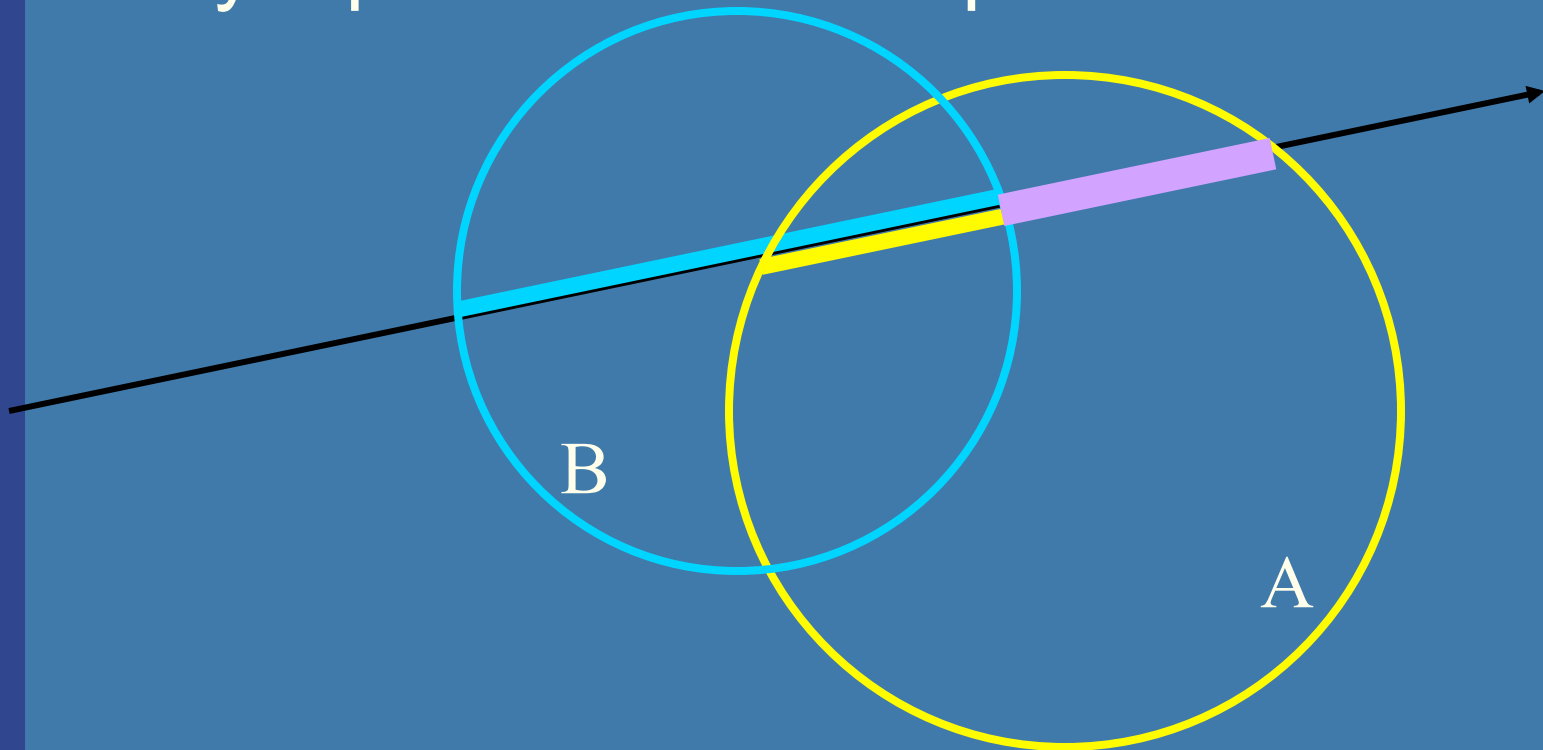
Geometry: Constructive Solid Geometry (CSG)

- Another example
- Done with 4 cylinders

Constructive Solid Geometry (CSG) How to implement

- Try: sphere A minus sphere B

- In summary: find both entry and exit points on both spheres. Such two points on a sphere is an interval (1D). Apply the operator on these intervals

CSG

- Works on any geometrical object, as long as you can find all intersection point along a line

- Be careful with optimizations…

- And objects should be closed
 - Example: put caps on cylinder.

Geometry

- Quadrics (2:a-gradsytor)
  - Cone, cylinder, paraboloids hyperboloids, etc.

- Higher order polynomial surfaces
  - Example: torus (badring), 4th degree

- Fractal landscapes
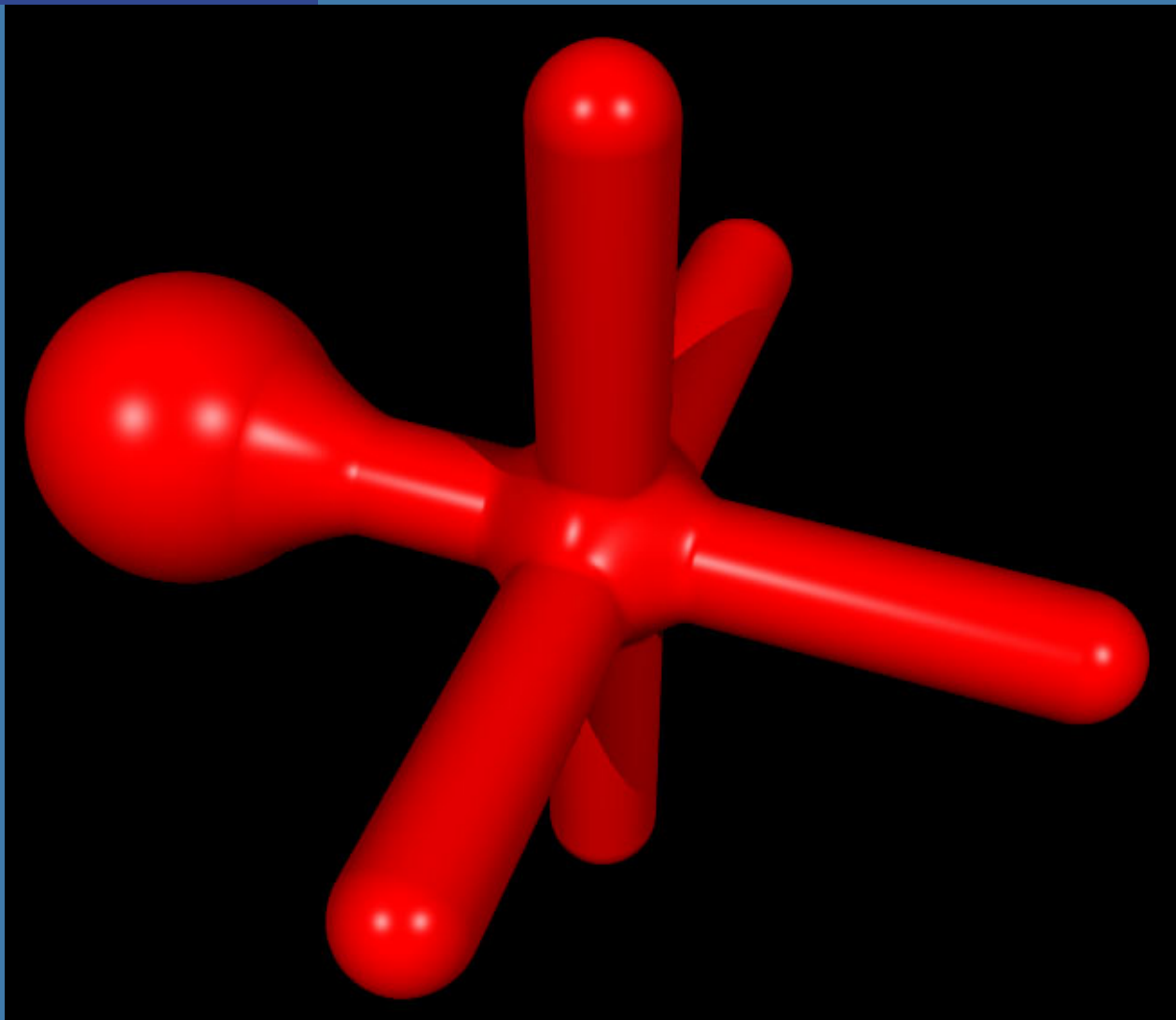 - Pretty simple, fast algorithm exist
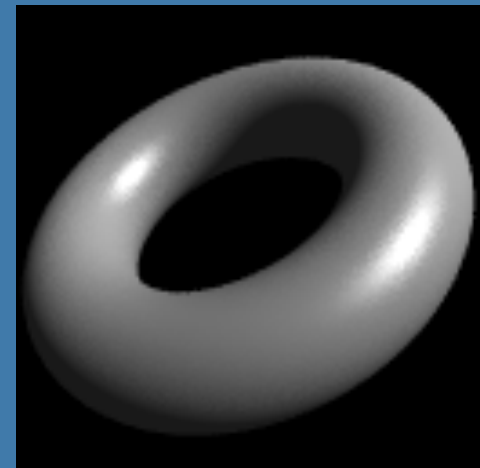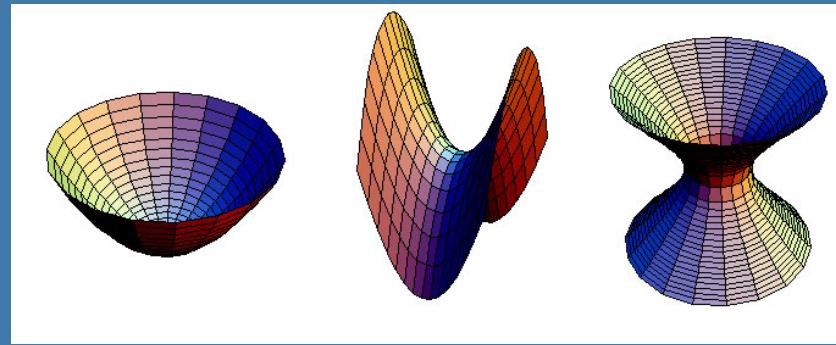
Geometry: Blobs

- A method for blending implicit surfaces (e.g., spheres, $x^2+y^2+z^2=1$)
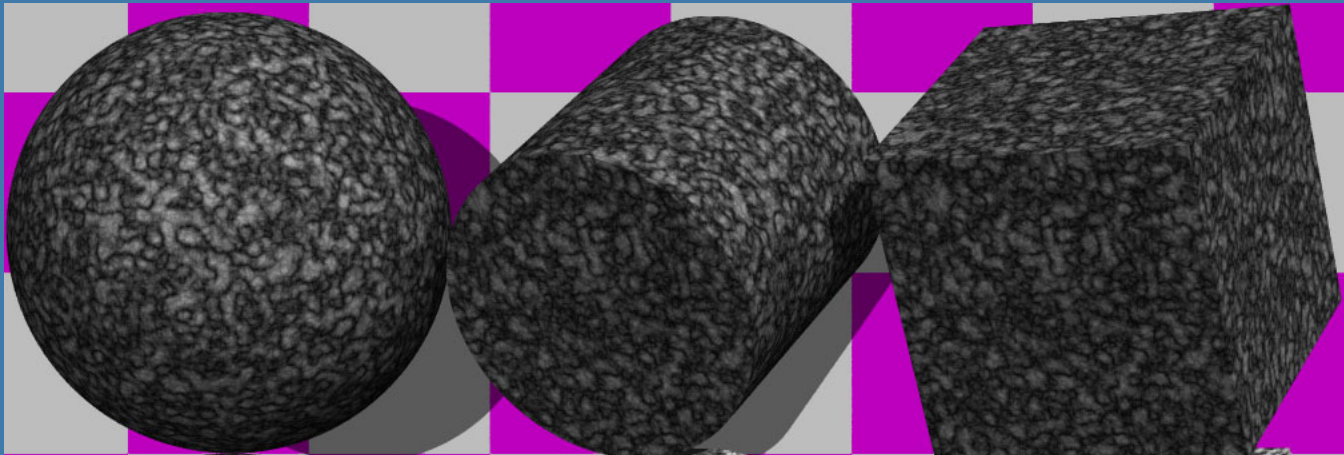- After blend, you get a higher order surface
- Need a numerical root finder
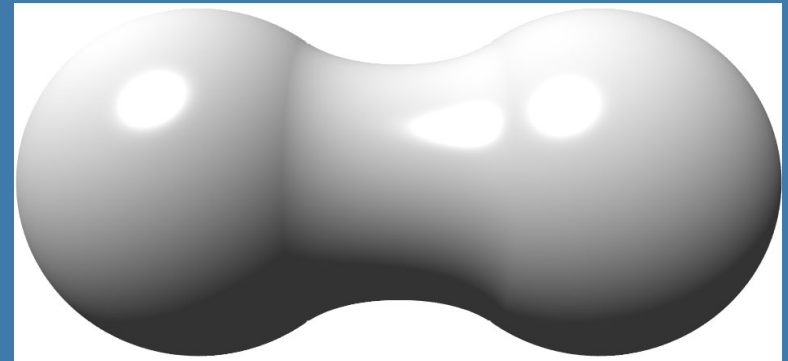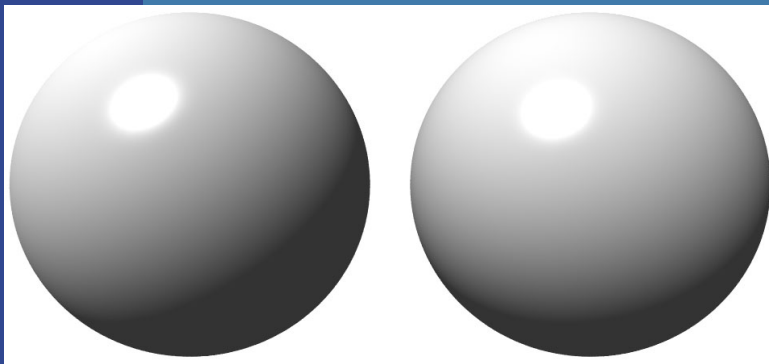
Blob example

Optics and more…

- You can add
  - Fog
  - Light fall off : $1/d^2$
  - Fresnel equations
  - Texture mapping
  - Prodedural texturing

Optics

- Depth-of-field
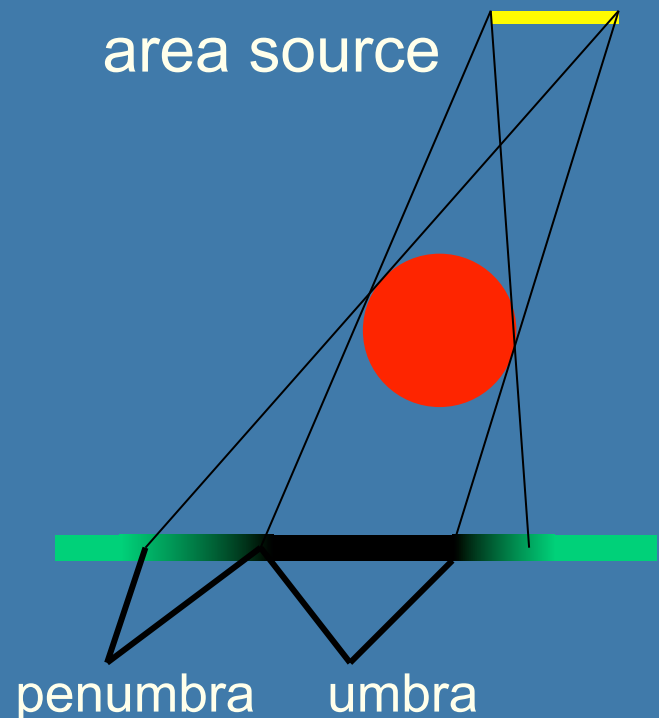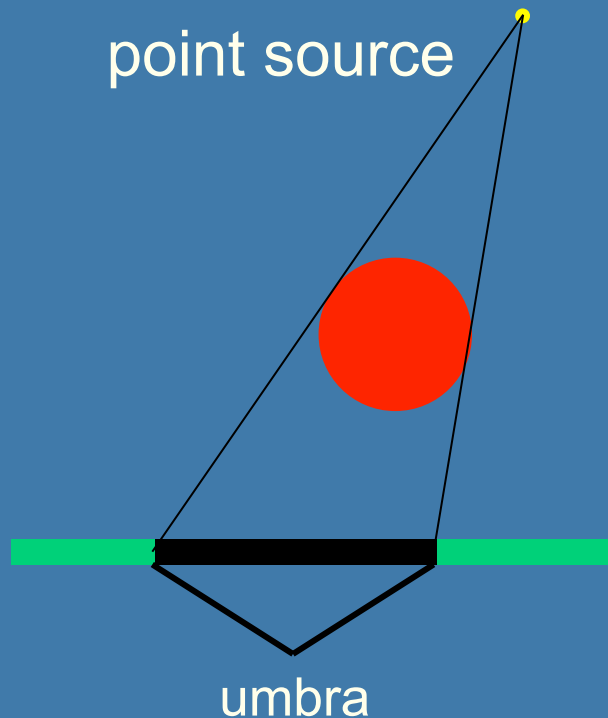 - Add more samples on a virtual camera lens
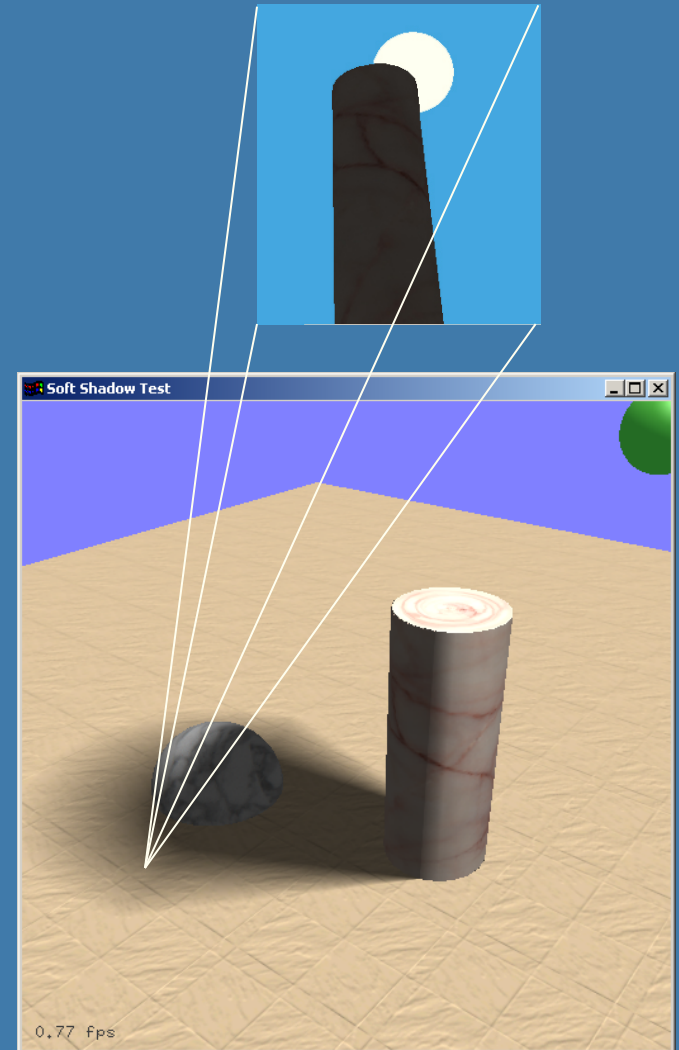
Soft shadows
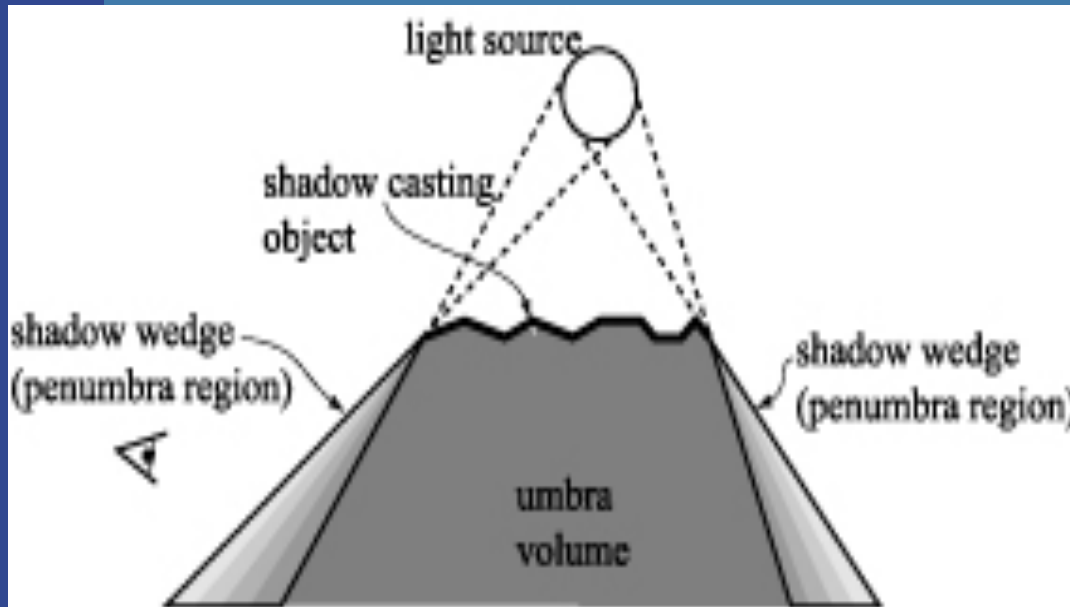
- Higher level of realism
- Examples

Soft shadows

- Why do they appear?
- Because light sources have an area or volume (seldom point lights)

point source

area source

umbra

penumbra   umbra

Example

light source

shadow casting object

shadow wedge (penumbra region)
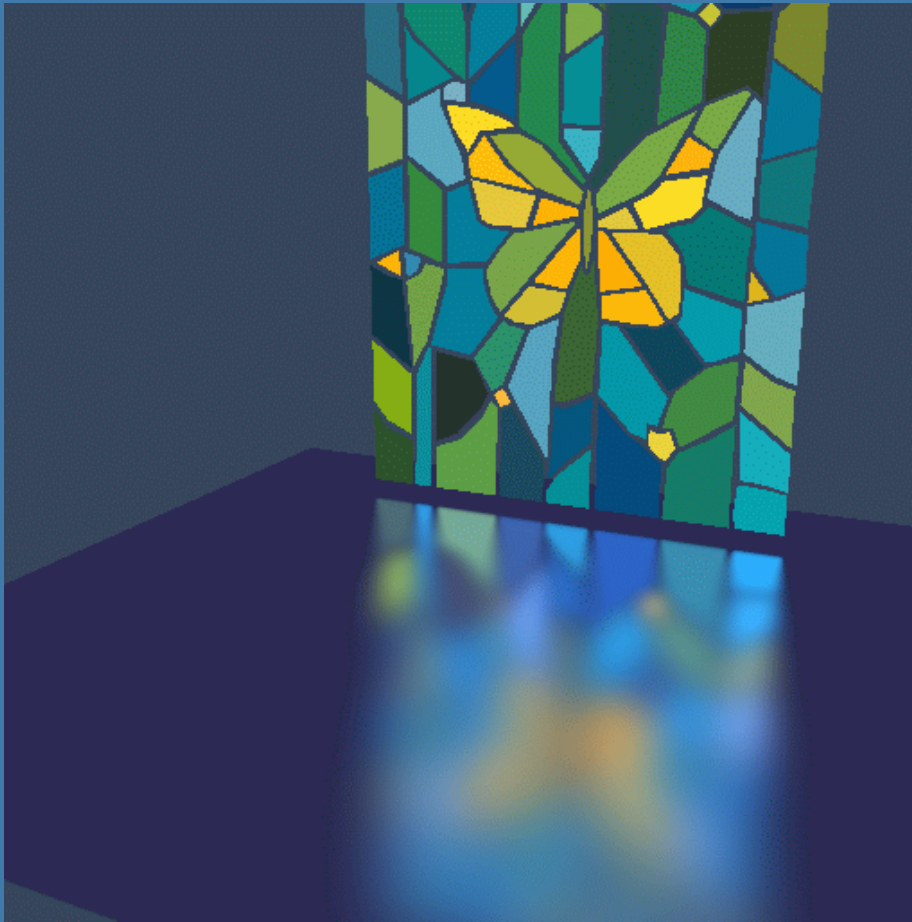
shadow wedge (penumbra region)

umbra volume

Glossy (blurry) reflections

- Trace many reflection directions
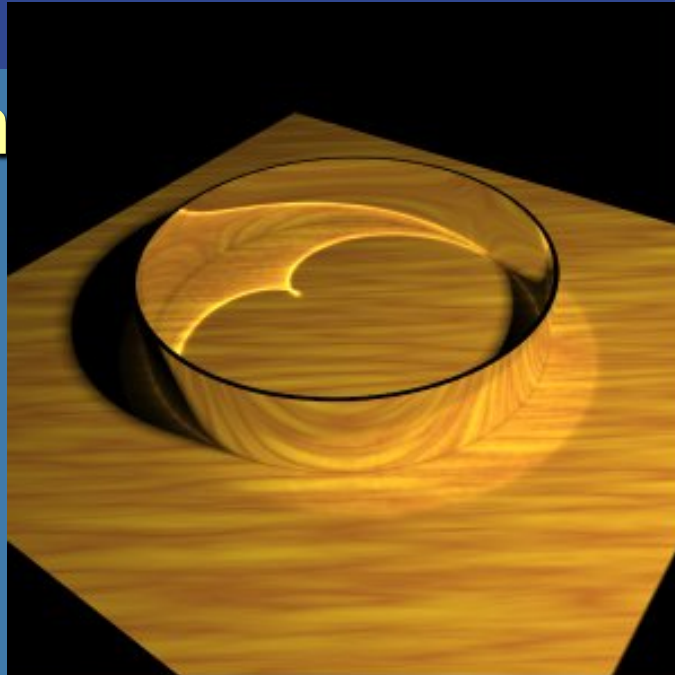 - Each perturbed slightly

Do the same
with transmission

Caustics

Photorealism

etc

Animation

- You can develop a little animation system, where you can save images
- Transforms are dealt with as treated previously
- Make transform a function of time: $\mathbf{M}(t)$
- Add motion blur (exposure time)

Image courtesy Brostow and Essa

Speed-up techniques

- Adaptive supersampling
- Try to exploit coherence
    - Frame-to-frame coherence (animation)
    - Pixel-by-pixel coherence (shadow cache)
    - Object-to-object (use spatial data structures)
    - Shading coherence?
    - And more…

Speed-up techniques

- For eye rays:
  - Render scene with OpenGL
  - Let each triangle or object have a unique color
  - Then read back color buffer
  - For each pixel, the color identifies the object
  - You need fast rendering in order to gain from this

- For eye rays (another technique):
  - Create a plane through a scanline of pixels
  - Objects that does not intersect plane can be skipped for this scanline

Typical Exam Questions

- Draw grid (plain/hierarchical/recursive)
- What's a
  - skip-pointer tree?
  - Shadow cache?
- Descibe ray/BVH intersection test
- The Fresnel-effect: metall, dielectrics)
  - How does glass/water/air behave?
  - How does metal behave?
- Describe how to implement ray/object intersection for Constructive Solid Geometry

THE END