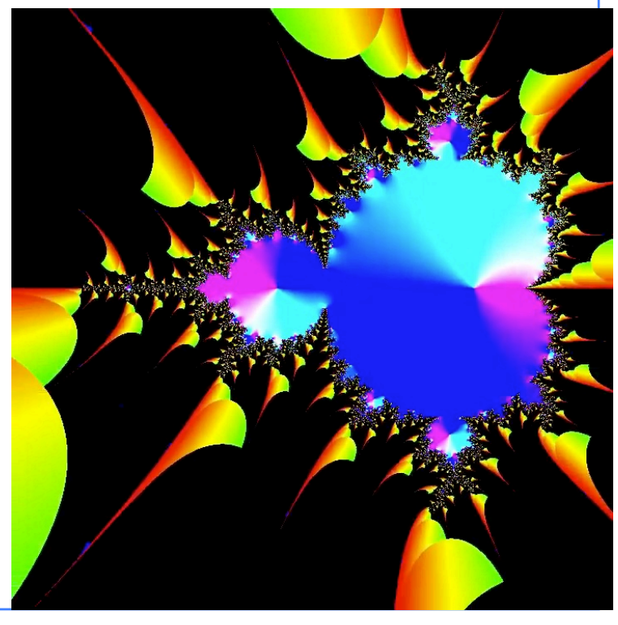
TSBK07/TSBK11

Computer Graphics

Ingemar Ragnemalm, ISY



Lecture 12

Anti-aliasing

Ray-tracing

Radiosity

Path tracing

Project suggestion are rolling in

All that made one should have had a response by now.

If you havn't decided, the last lectures present several possibilities.

Lecture questions

1. How does the frequency contents vary in typical images?

2. What is the highest magnitude of errors for 2x supersampling?

3. How are shadows produced in ray tracing?

4. Why might you want to cast more than one ray per pixel?

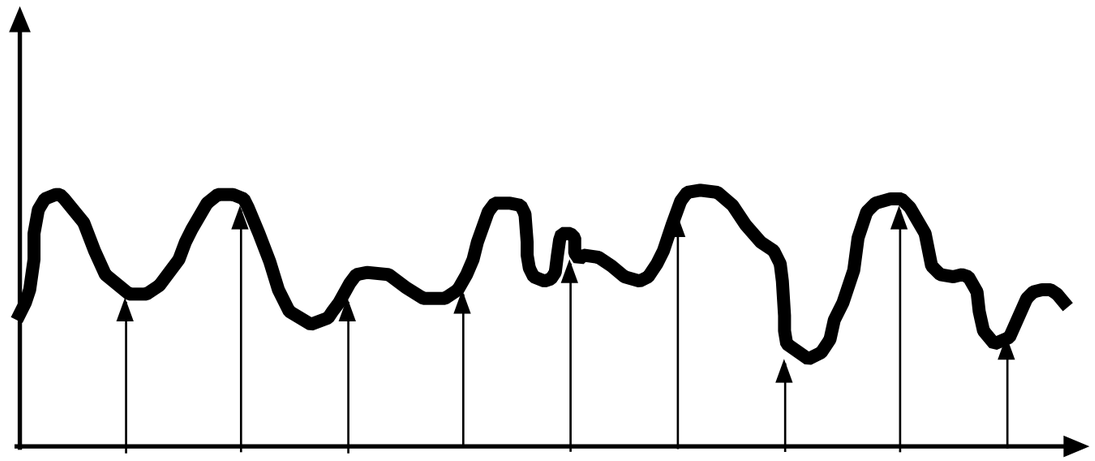
5. How do you anti-alias ray-tracing even more than supersampling?

6. What is the difference between photon mapping and path tracing?

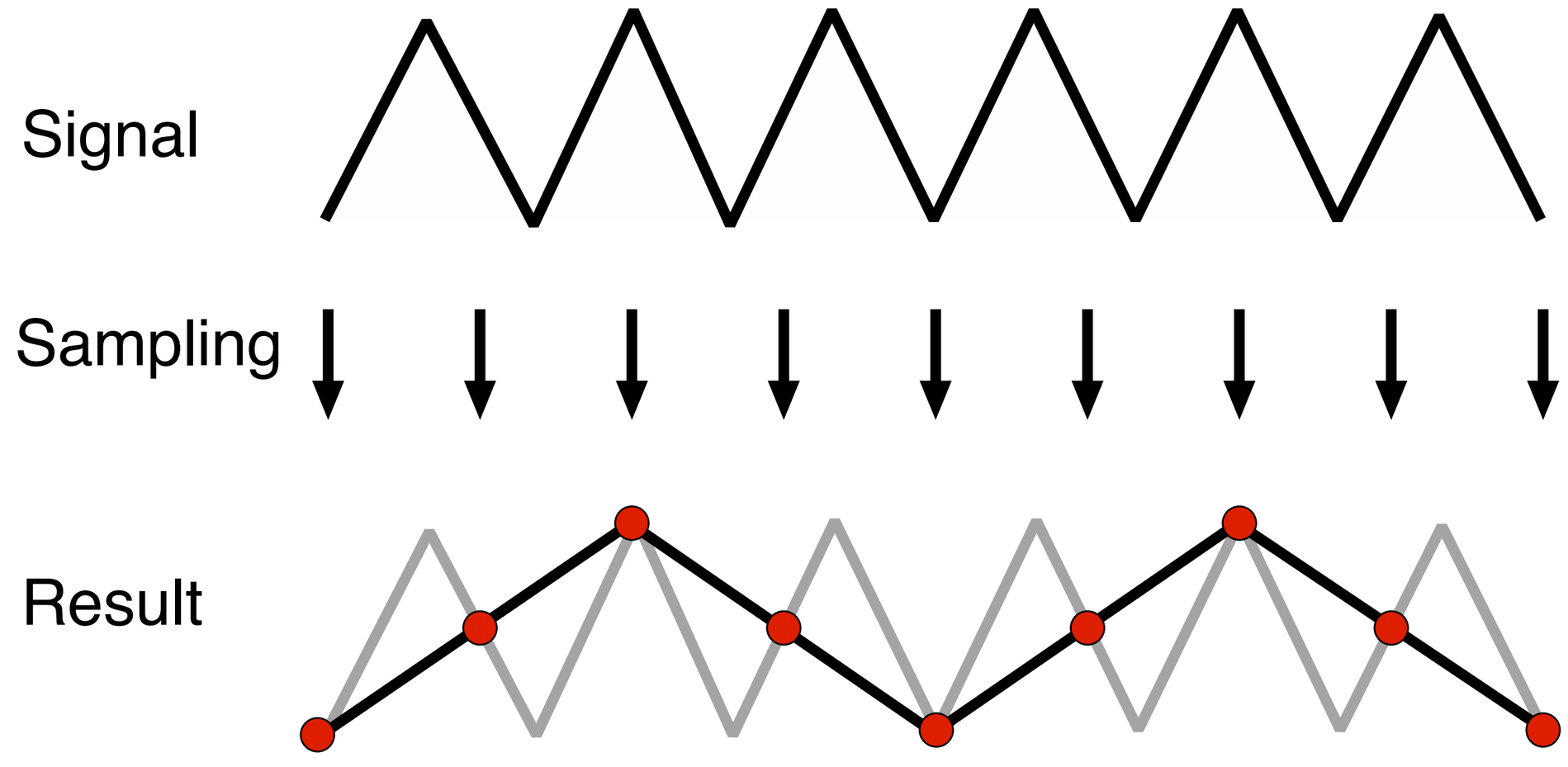
Anti-aliasing

anti-aliasing ... is often an absolute requirement these days (Stefan Gustavson, 2024)

Sampling



A digital image is a sampled version of an underlying analog 2D signal.


Signal
Sampling
Result

Frequency space

Any signal can be decomposed into sinus waves

The amplitudes of all sinus waves form a new space - frequency space

This space exists in any dimension. An image is a 2D signal and has a 2D frequency space.

Sampling + reconstruction

When presenting the digital signal, it is reconstructed to an analog signal.

A signal can be decomposed into different frequency bands.

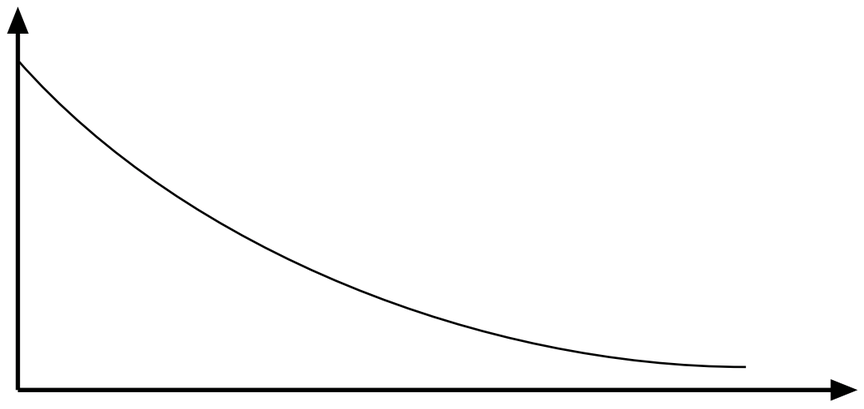
What parts of the original signal that are accurately reconstructed depend on the frequencies.

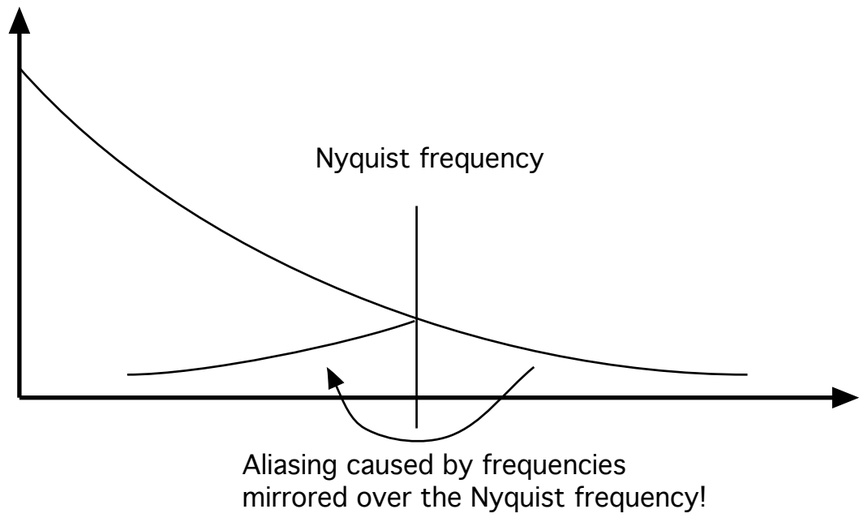
Aliasing in frequency space!

Which frequencies are preserved and which cause problems?

Amplitudes usually lower for higher frequencies!

The 1/f rule!

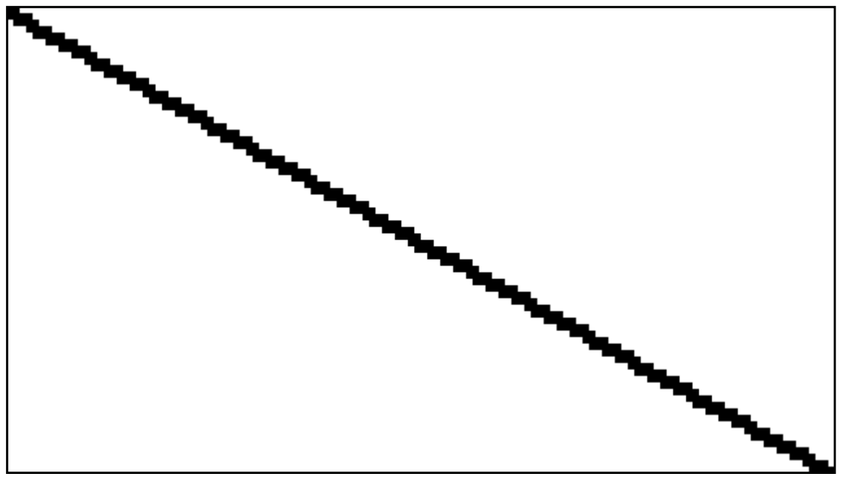




Anti-aliasing methods

- Linear post-filtering. Bad.
- Super-sampling: Split pixels in sub-pixels. Check how many sub-pixels that hit one pixel.
- Edge smoothing: Estimate edges, smooth edges.
- Non-linear post-filtering.

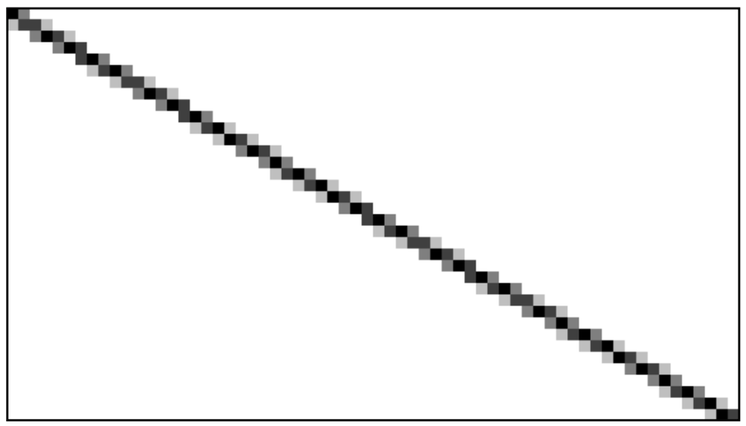
Supersampling



Draw in a high-res image buffer

Simple but slow and memory-demanding

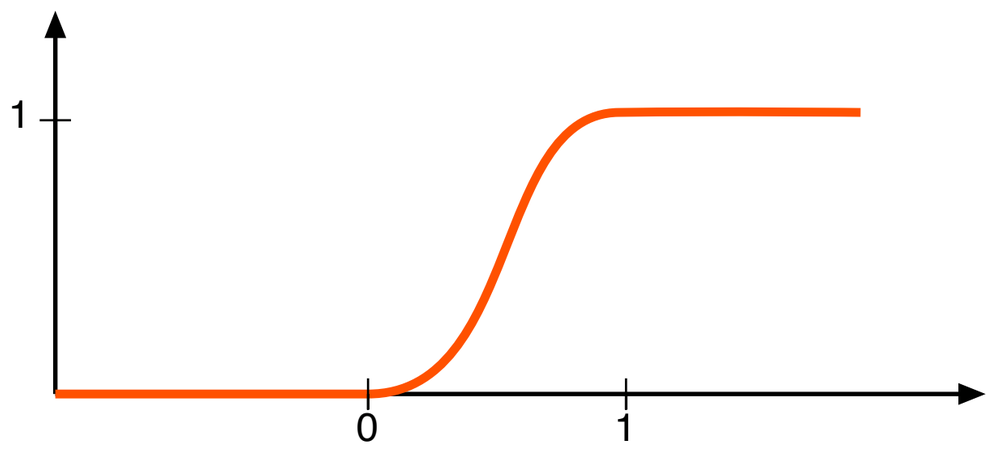
Sample down to destination buffer

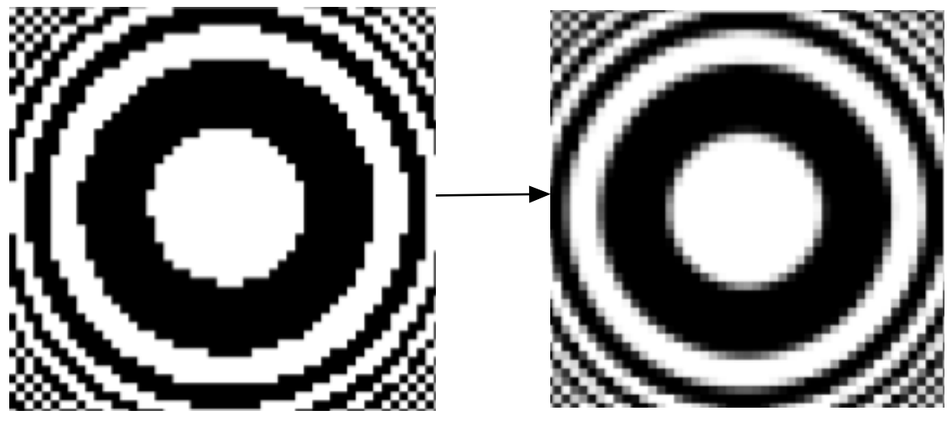


Edge smoothing

For line drawings and similar.

Measure edge by gradients in image, use the "smoothstep" function on edges.





"If frequencies are known, add "frequency clamping", smooth high frequencies.

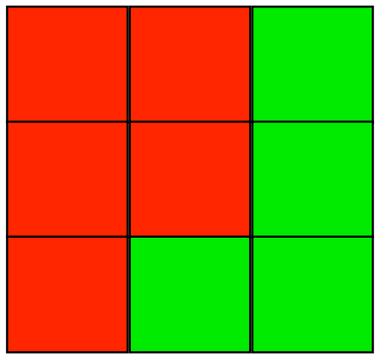
Multisampling

Variant of supersampling

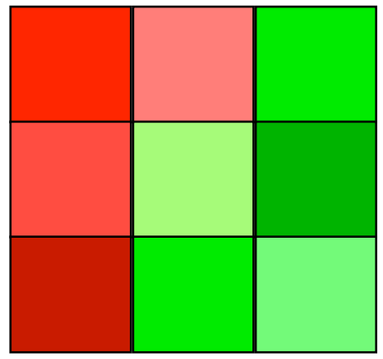
More efficient

Only improves edges

Multisampling



Multisampling: Only one execution of fragment shader for all samples from the same geometry



Supersampling: One execution of fragment shader for each sample

Multisampling

Less fragment processing

Fewer texture accesses

Same number of memory writes and same post-processing

FXAA = Fast approXimative AA

Post-processing

No higher resolution image

Non-linear filter

Don’t filter patterns

Several recent methods of this kind

Anti-aliasing in OpenGL

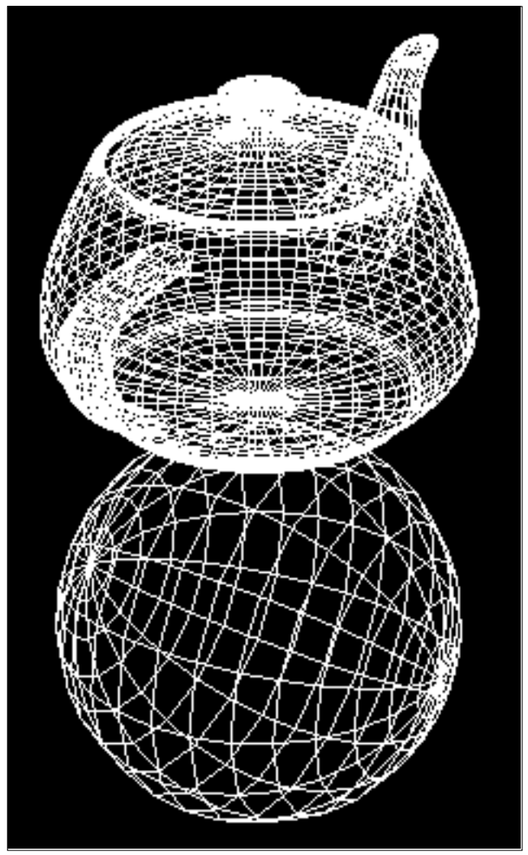
GL_POLYGON_SMOOTH - Old built-in, avoid

Accumulation buffer tricks: Obsolete, avoid

glEnable(GL_MULTISAMPLE); Preferred!

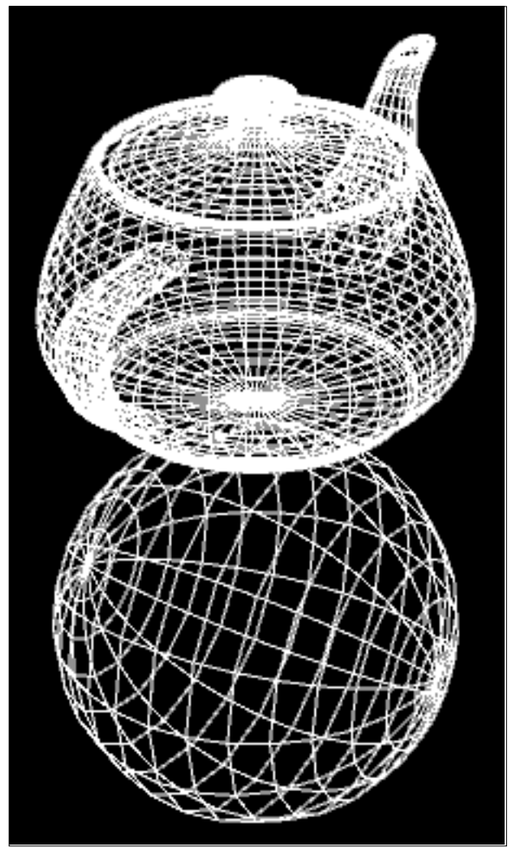
Can also be done by shaders. Usually unnecessary.

FSAA example



No AA

AA



Anti-aliasing

We will return to anti-aliasing in the ray-tracing part (with even more elaborate supersamling)

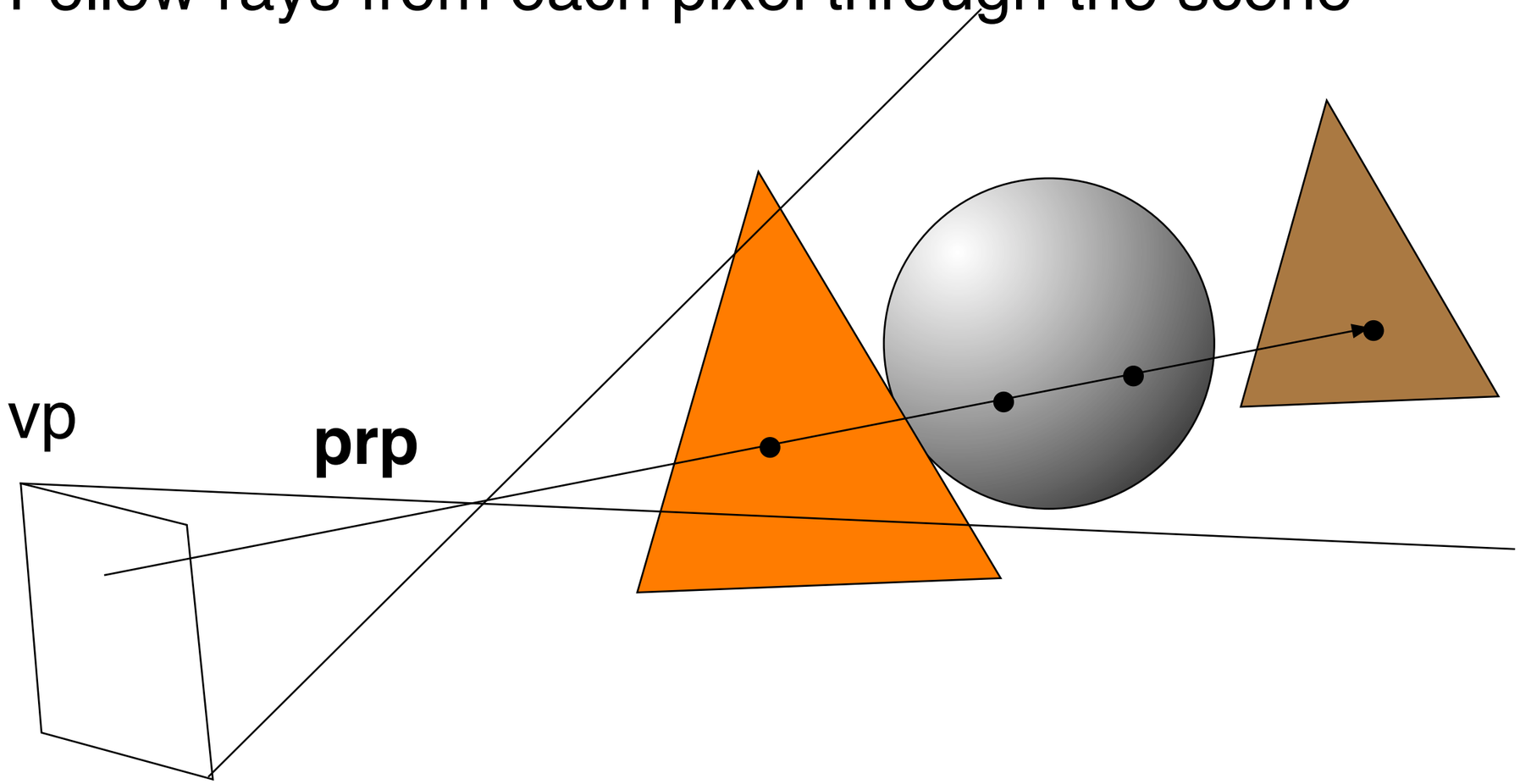
Off-line (?) rendering and global illumination

Performance demanding methods that give better lighting

Ray-casting
Ray-tracing
Radiosity
Photon mapping
Path tracing

Ray-casting

Follow rays from each pixel through the scene



Full 3D raycasting

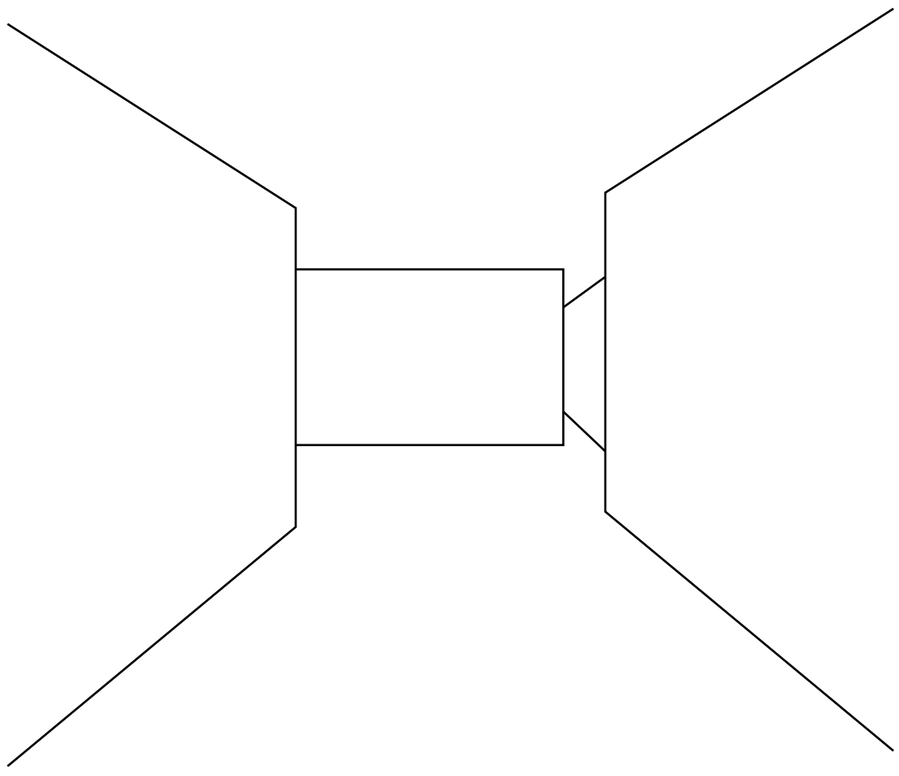
for every pixel (x,y) in the image

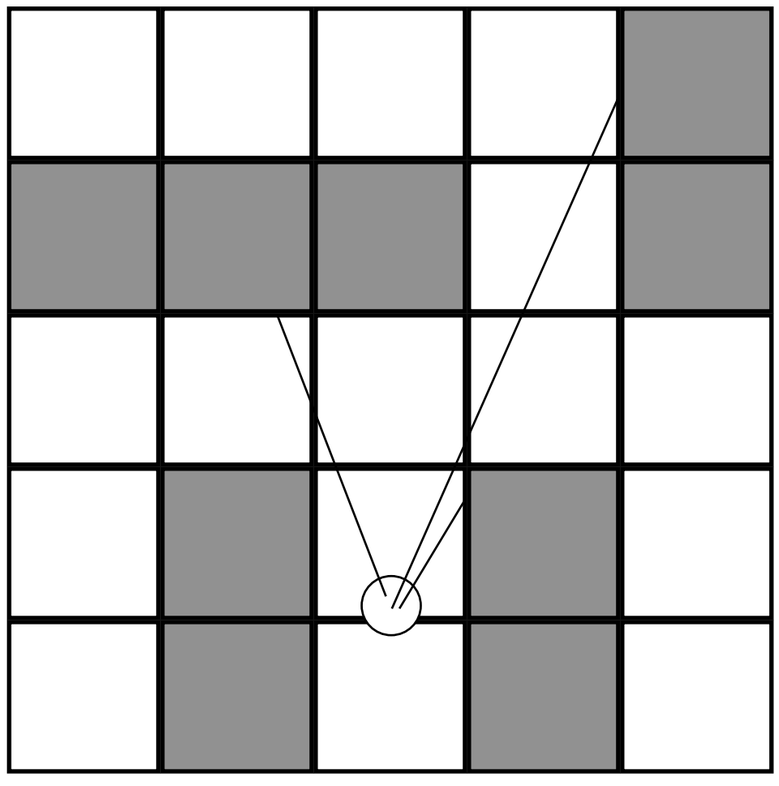
calculate a ray from the pixel through the camera (cop) and through the scene

calculate intersecions with all objects in the scene

the pixel value is calculated from the closest intersection found

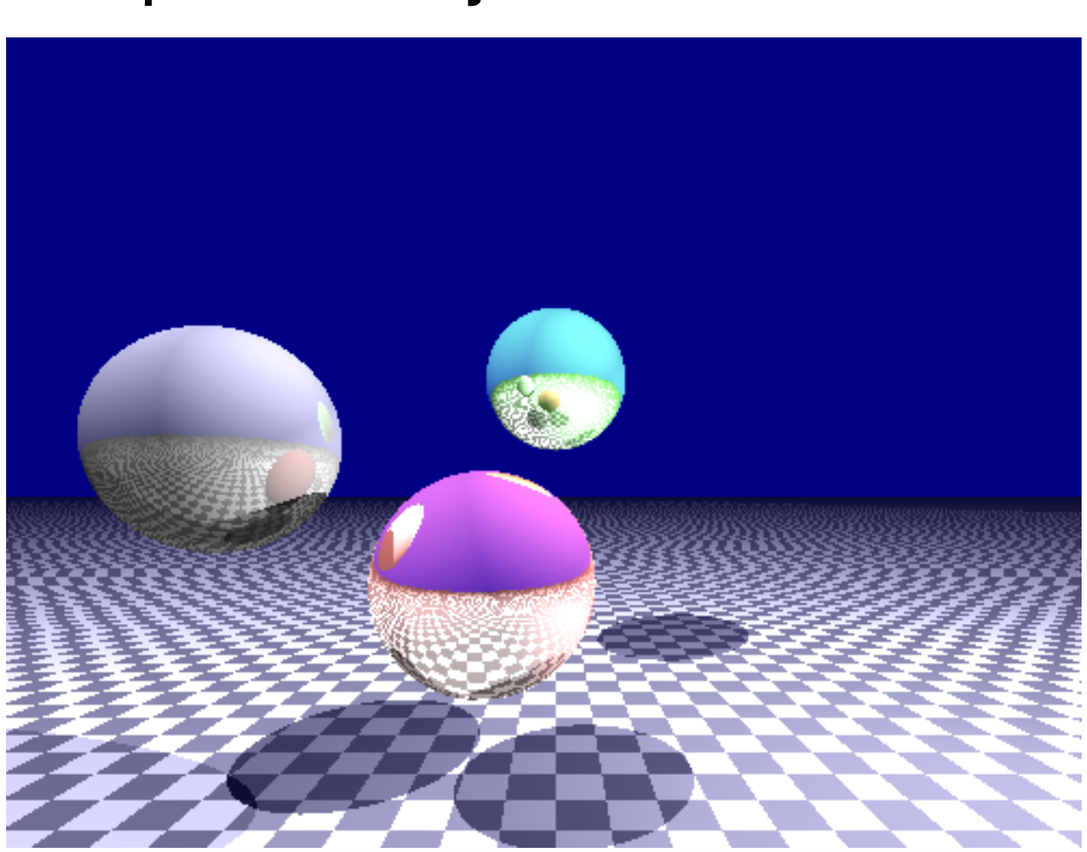
Raycasting in 2D or 3D grid ("Ray-marching")





Ray-tracing

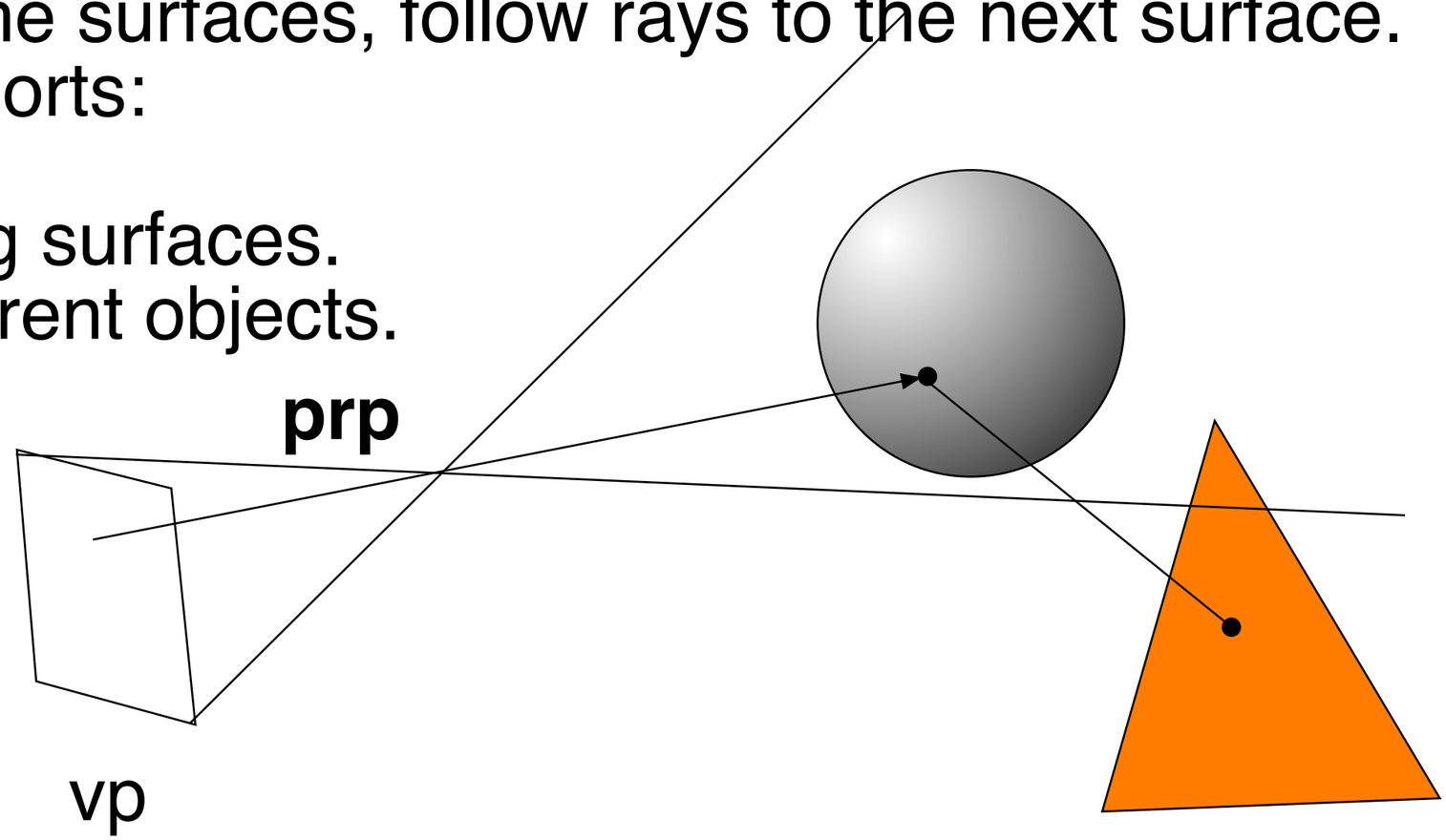
The classic method for rendering realistic images of shiny and transparent objects with shadows.



Ray-tracing basics

Cast rays from the camera as before.
From some surfaces, follow rays to the next surface.
This supports:

- Mirroring surfaces.
- Transparent objects.



At the intersection

Three things can happen when a ray intersects an object

1) Lighting, non-mirroring reflection

2) Reflection

3) Refraction

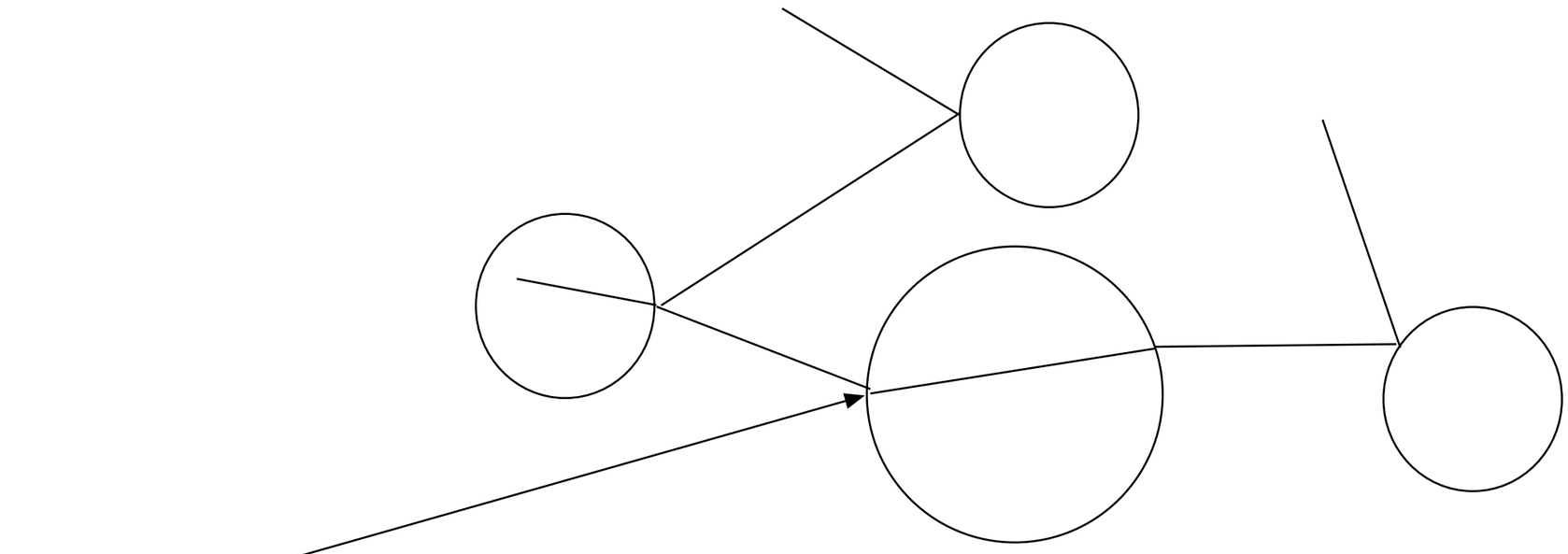
Non-mirroring reflection

Apply e.g. the three-component light model

Ambient light
Diffuse reflection
Specular reflection

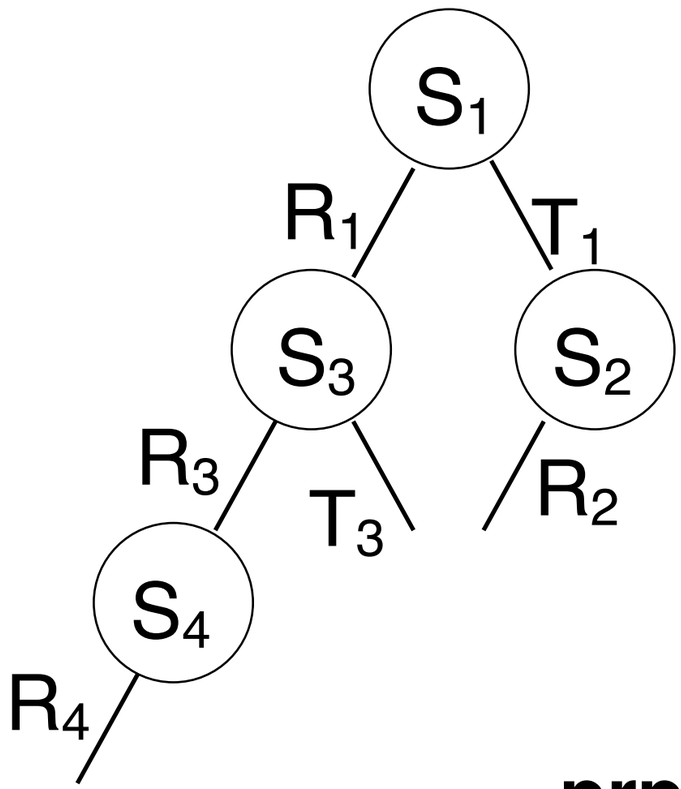
(or other model)

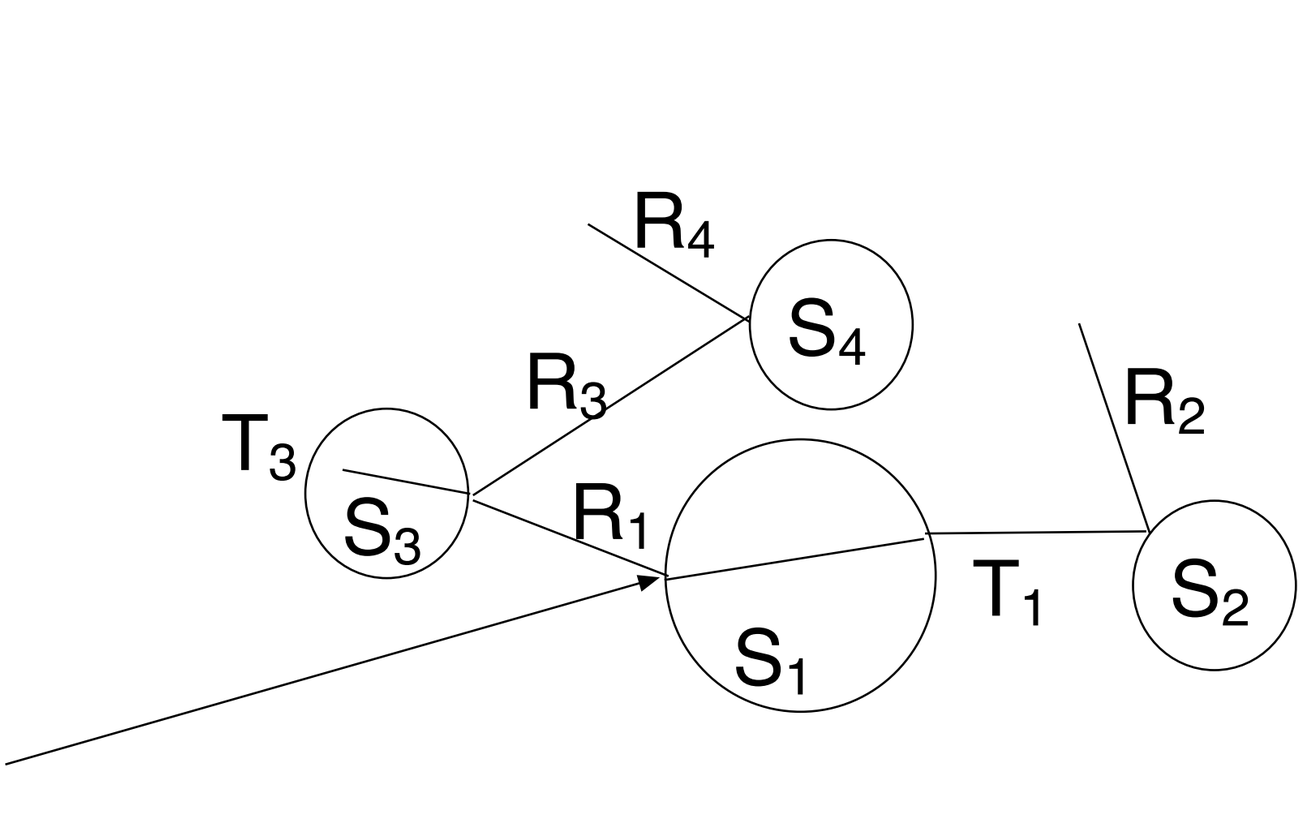
Reflection and refraction



Recursive process!

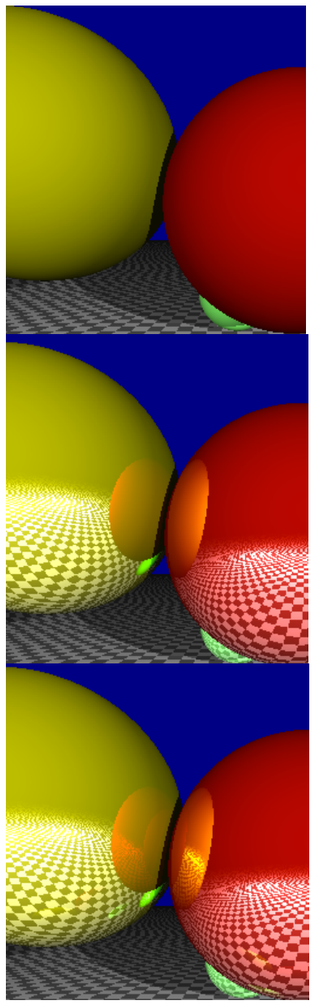
The ray-tracing tree





The maximum depth

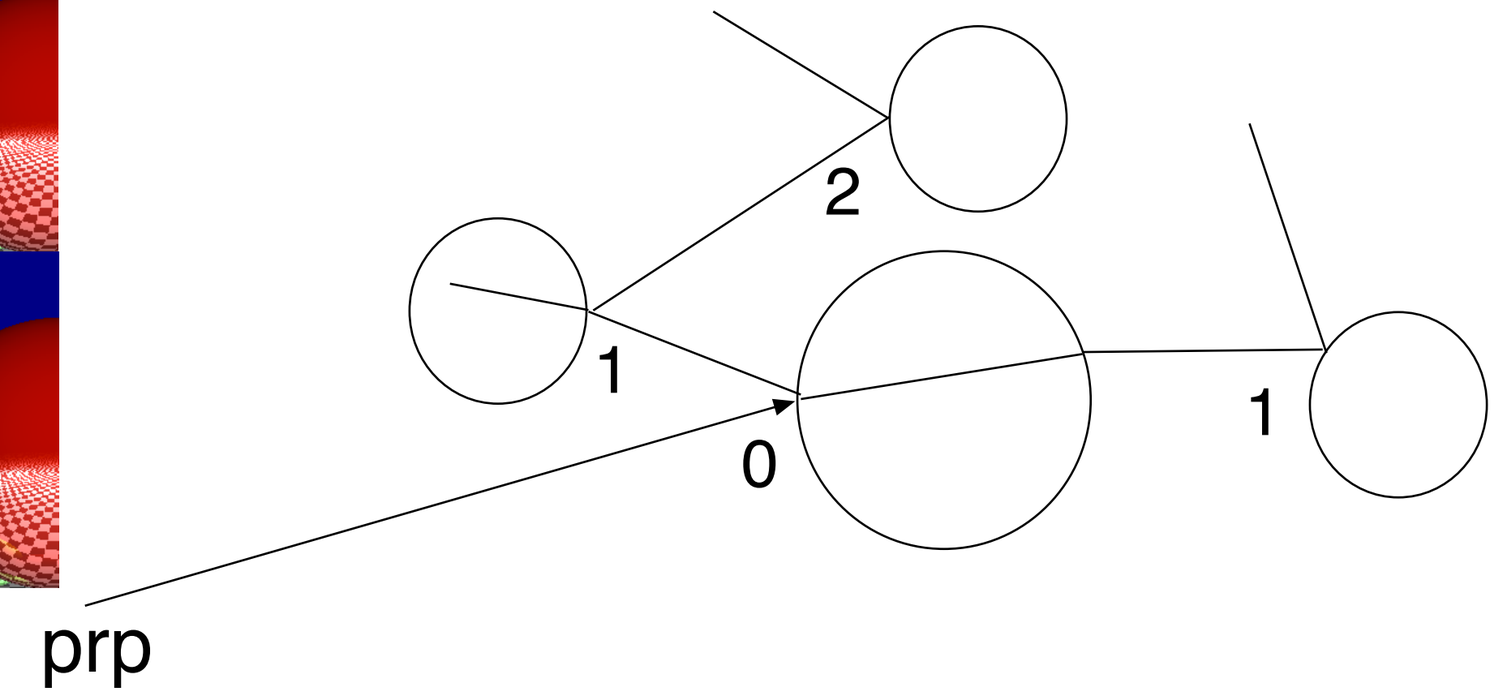
0



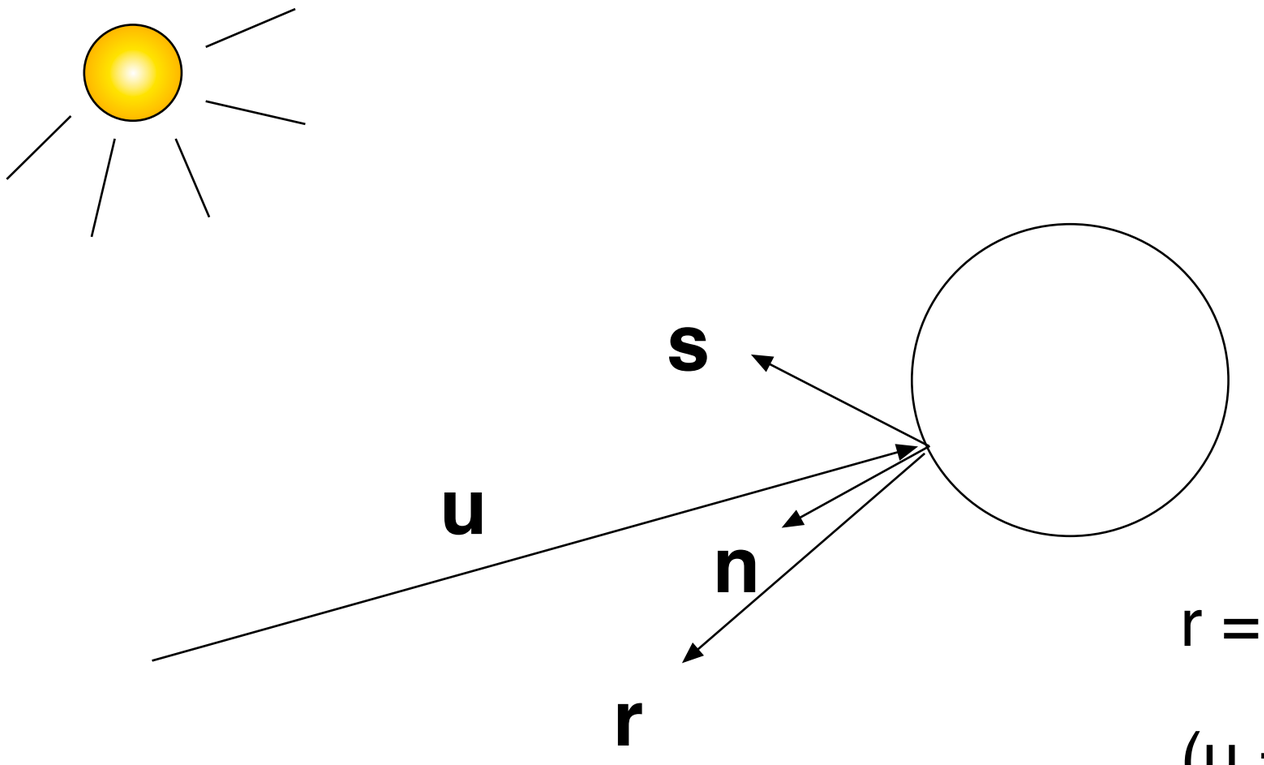
1

2

How deep can the tree get?
How many reflections and refractions are allowed?



Reflections



Look for more contributions here

$r = u - (2u \cdot n)n$

(u = direction vector of the incoming ray)

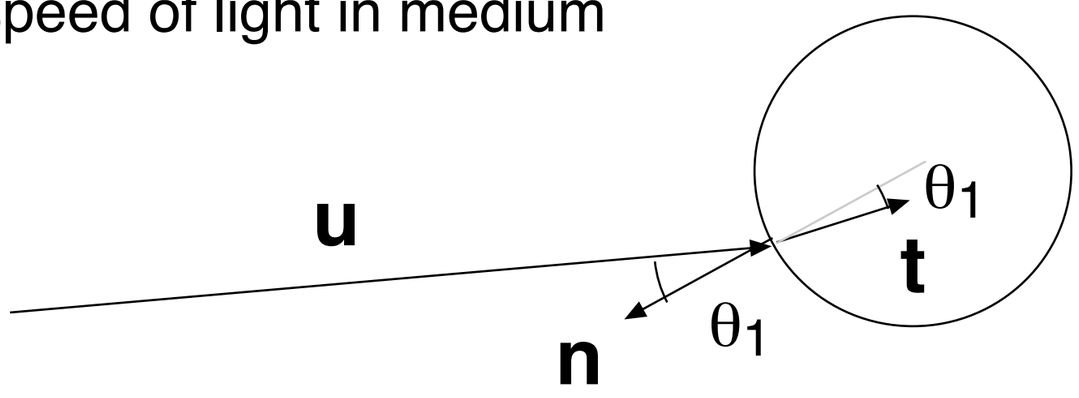
Refractions

Snell's law:

$\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$

Refraction index:
$\eta = c / v$
c: speed of light in vacuum
v: speed of light in medium

Transparent object



Look for more contributions here

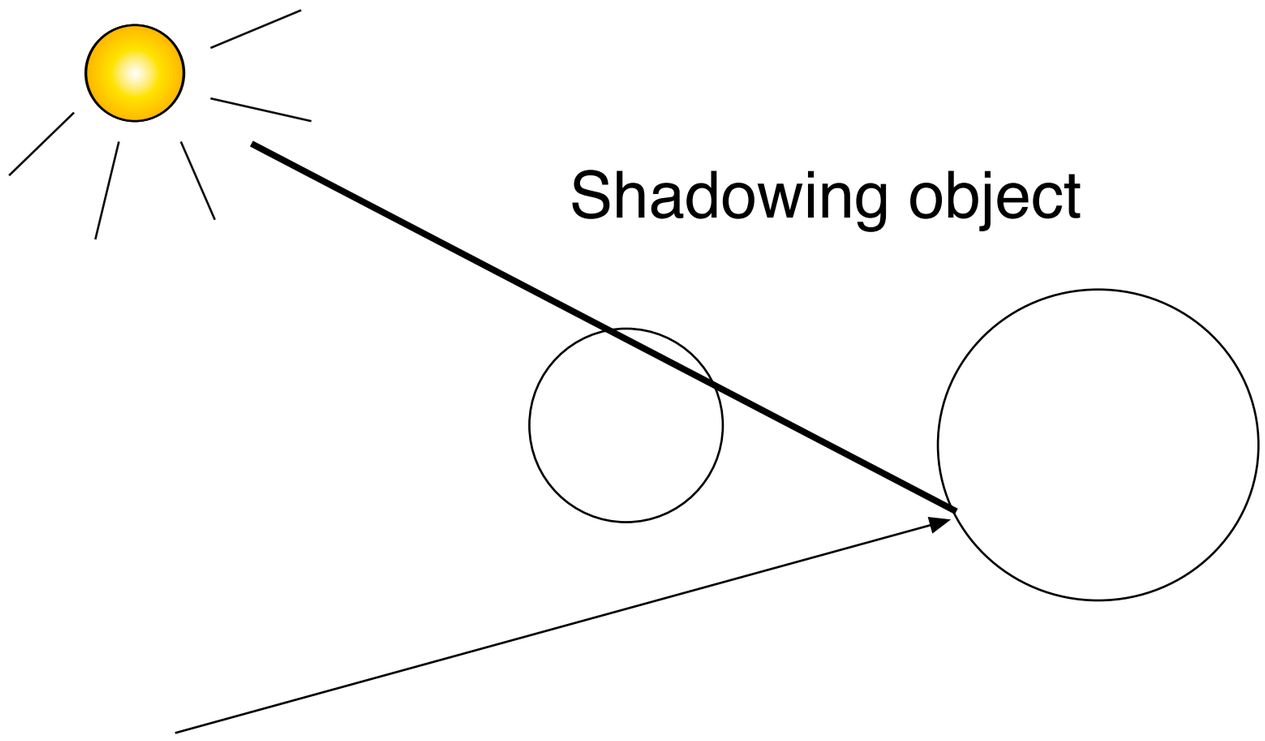
Outgoing angle is given by incoming angles and the density of each material.

Summing up

The total intensity is the sum of

- ambient light
- diffuse reflections from each light source
- specular reflections from each light source
- mirroring reflections
- refractions

The shadow ray



Object that can have illumination (shading) by diffuse or non-perfect specular reflections

The shadow ray tells if a light source can contribute to illumination

Ray-surface intersections

Ray equation:

$\mathbf{p} = \mathbf{p_0} + \mu\mathbf{u}$, $\mu > 0$

Combine with the equation of the surface.

Easiest surface: Sphere!

$x^2 + y^2 + z^2 = r^2$

Not quite as easy as for ray casting, since $\mathbf{p_0}$ can now be any point.

function RayTrace(p_0, u, depth)

if depth > max then return BLACK

μ := FindIntersection(p_0, u) // Returns more data, see below

if μ <= 0 then return BACKGROUND_COLOR

I_{local} :=0
I_R := 0
I_T := 0

if $k_a \neq 0$ and $k_d \neq 0$ and $k_s \neq 0$ then
    I_{local} := ka*Ia + $\sum$ (diffuse shading + specular shading)
// Sum is for all visible light sources

if $k_R \neq 0$ then
    R := CalculateReflection(u, N)
    I_R := RayTrace($p_0 + \mu$*u, R, depth+1)

if kT ≠ 0 then
    T := CalculateRefraction(u, N, h_1, h_2)
    I_T := RayTrace($p_0 + \mu$*u, T, depth+1)

return $I_{local} + I_R + I_T$

Problems in ray-tracing

- Computational load
- Aliasing
- Lack of realism due to extreme sharpness
- No indirect light

Reducing object-intersection calculations

The same large world problems showing up again!

Don’t test everything, use some large world structure!

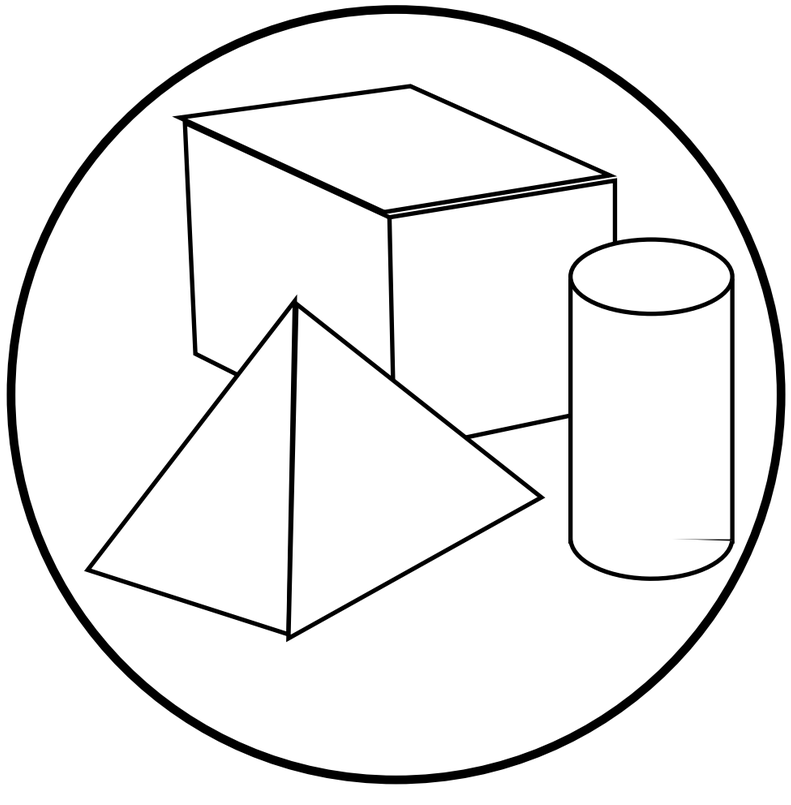
Reducing object-intersection calculations

Make sure that each ray doesn't have to be tested against all objects!

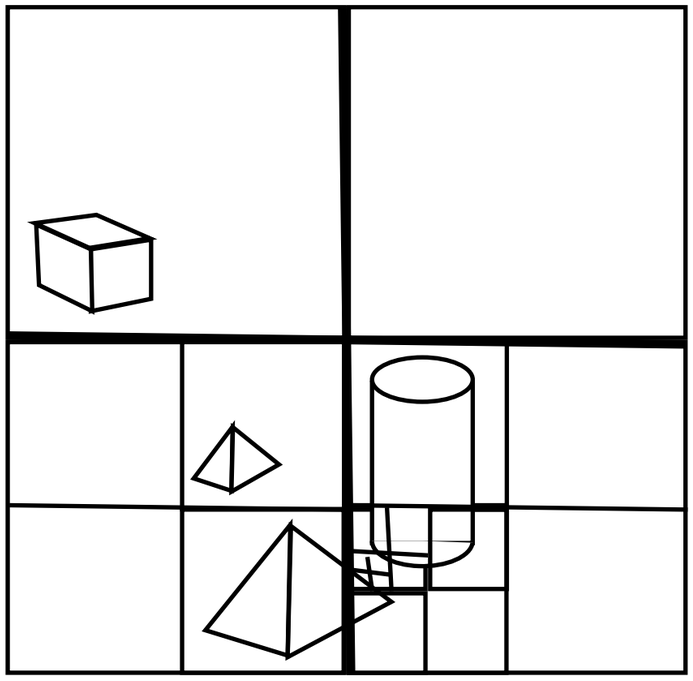
- Group objects within simple grouping surfaces (e.g. spheres or boxes)

- Divide space into cells
  - uniform or adaptive subdivision

Group objects into simple bounding objects



Divide space into cells



(Quad-tree)

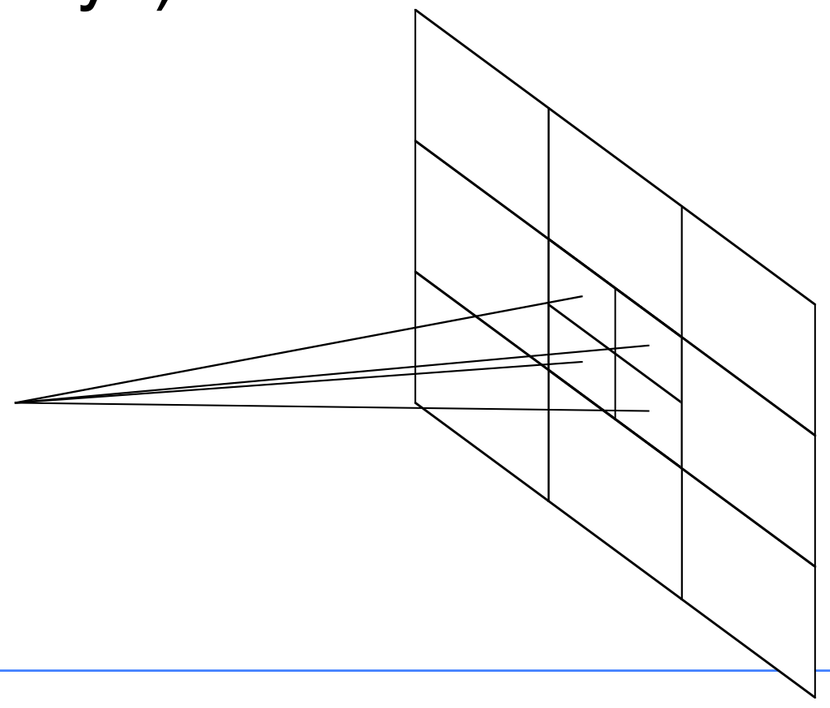
Same principle as in VSD and collision detection

Anti-aliasing in ray-tracing

- Supersampling
- Distributed ray tracing

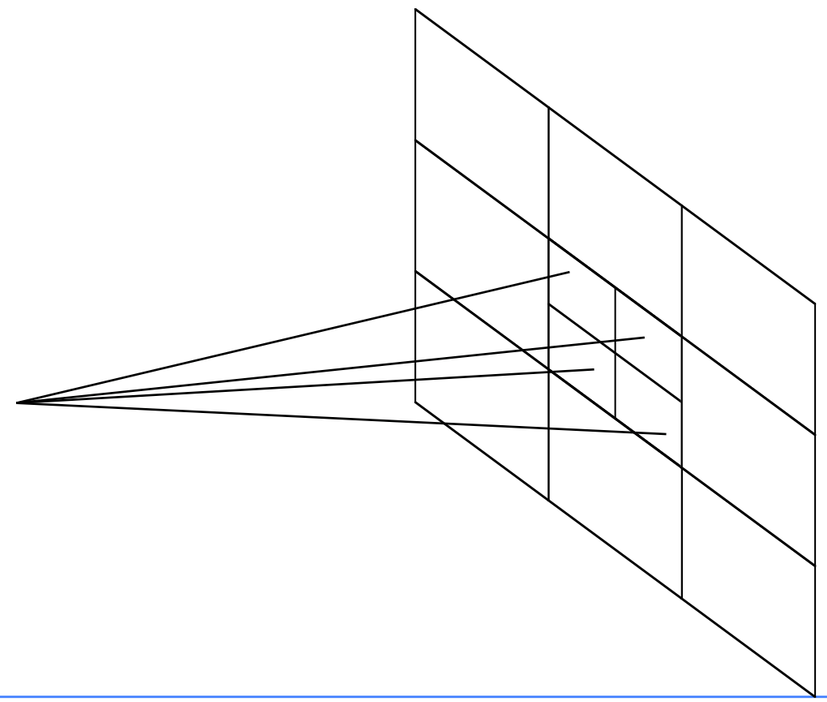
Supersampling
in ray-tracing

Send several rays for each pixel. (Typical for fast rendering: 4 rays)



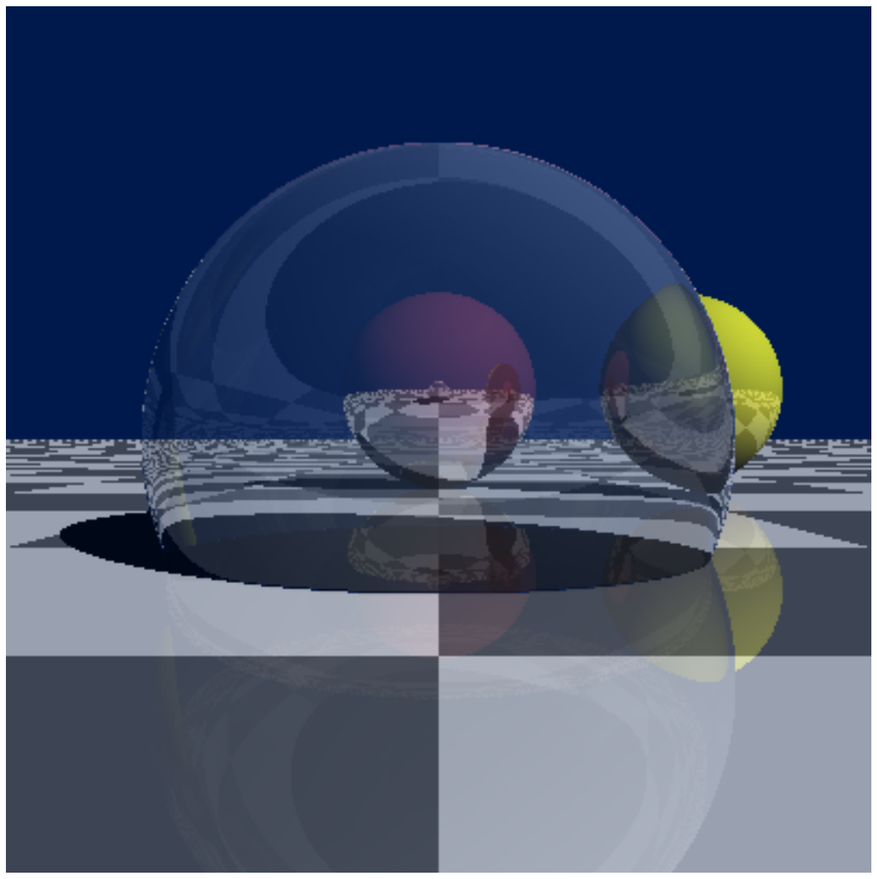
Distributed ray-tracing

Rays (multiple per pixel) are sent in a randomized pattern. Aliasing is replaced by noise!



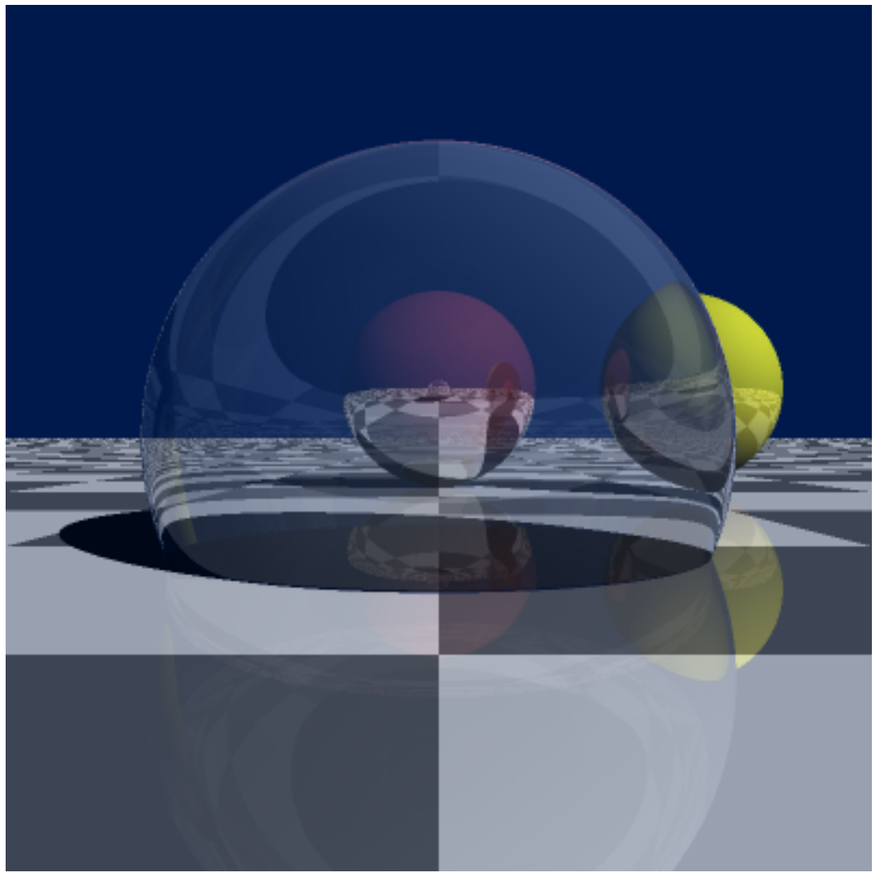
Anti-aliasing in ray-tracing

Example:
Without
anti-aliasing



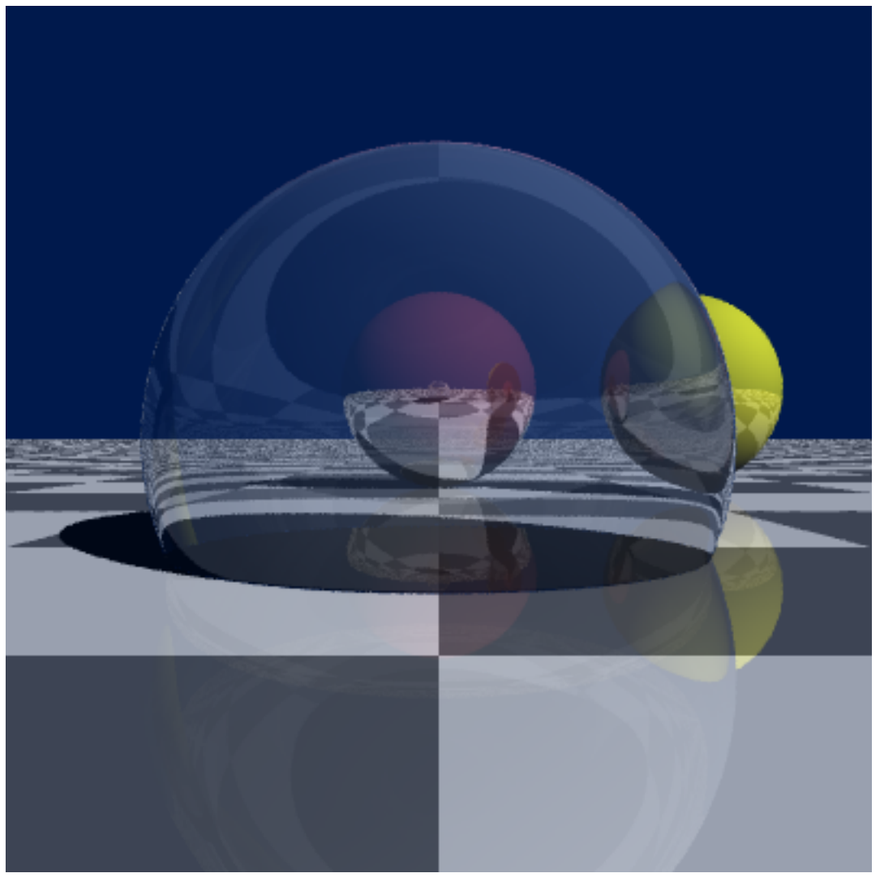
Anti-aliasing in ray-tracing

Example: Super-sampling (2x2)



Anti-aliasing in ray-tracing

Example:
Distributed
raytracing
(2x2)



Anti-aliasing in ray-tracing

Zoomed-in detail

Without anti-alias

Super-sampling

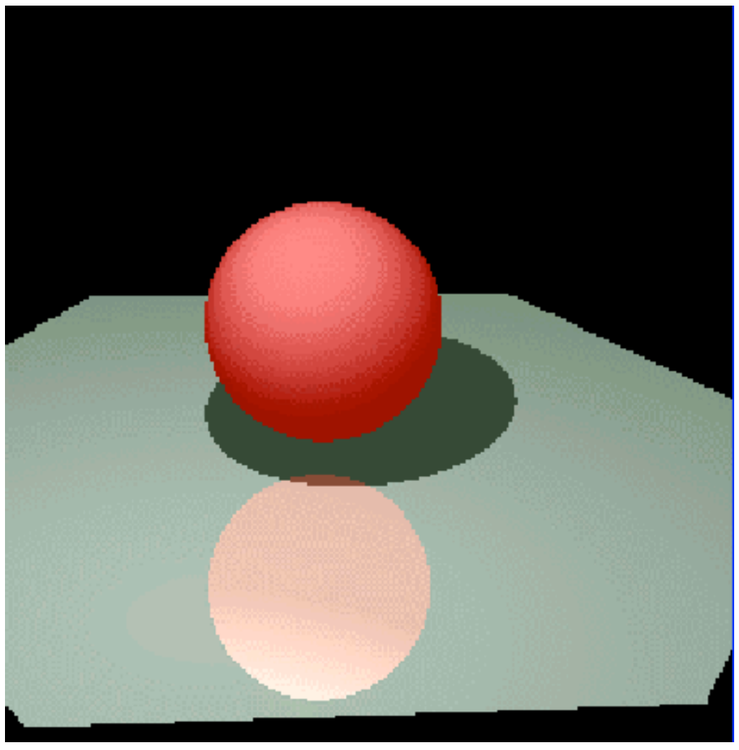
Distributed raytracing

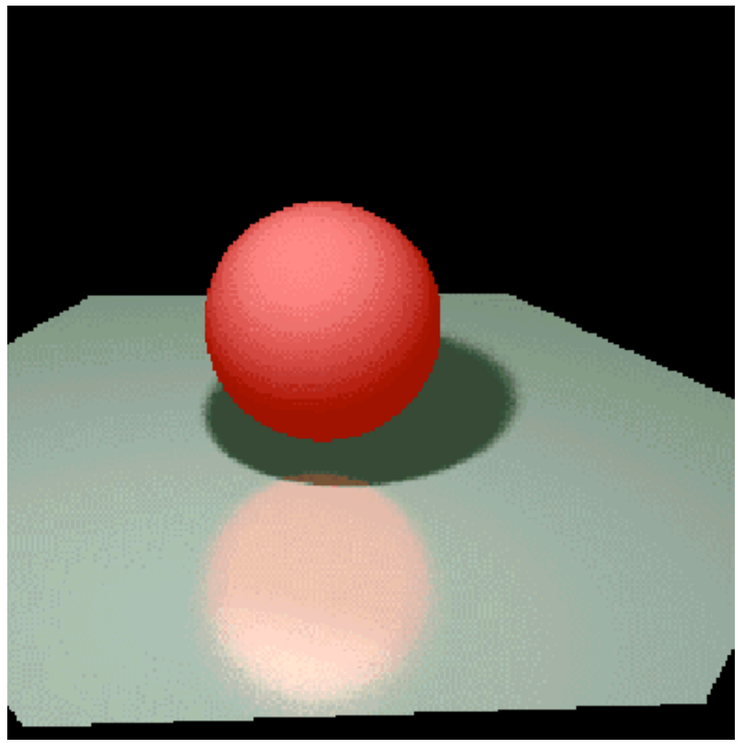
Distributed ray-tracing

Anti-aliasing through stochastic sampling.

Also used for:
- gloss (fuzzy reflections)
- fuzzy translucency
- soft shadows
- depth of field (out-of-focus effect)
- motion blur

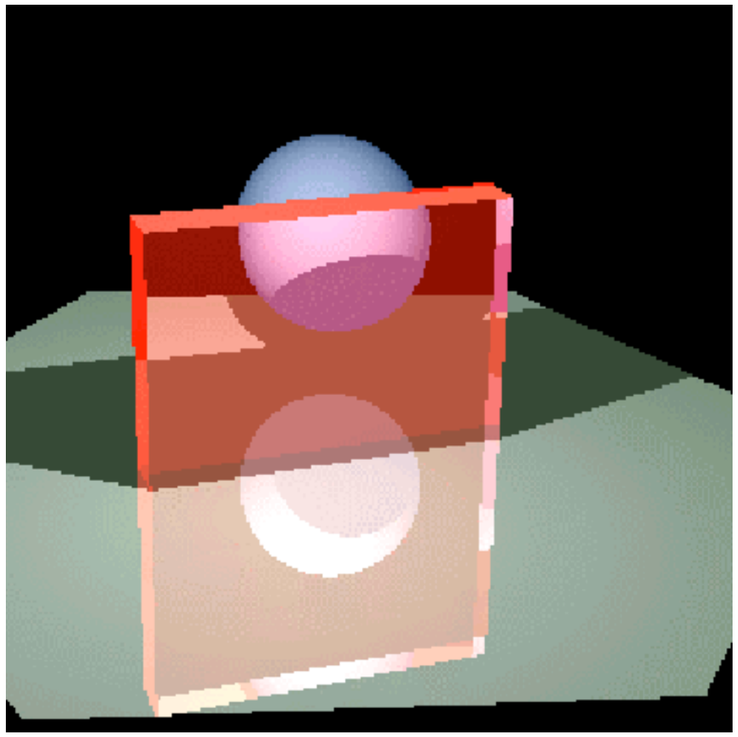
Gloss (fuzzy reflections)

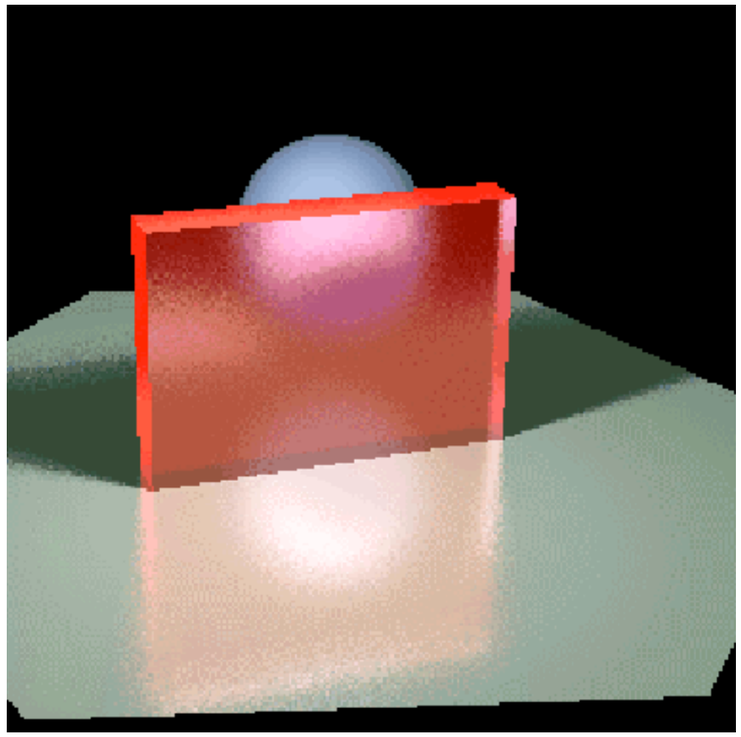




Jittering reflection ray

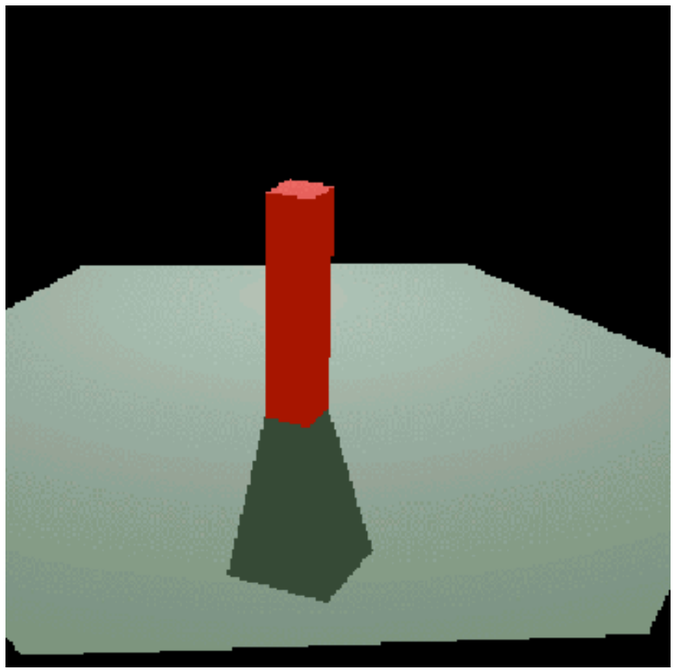
Fuzzy translucency

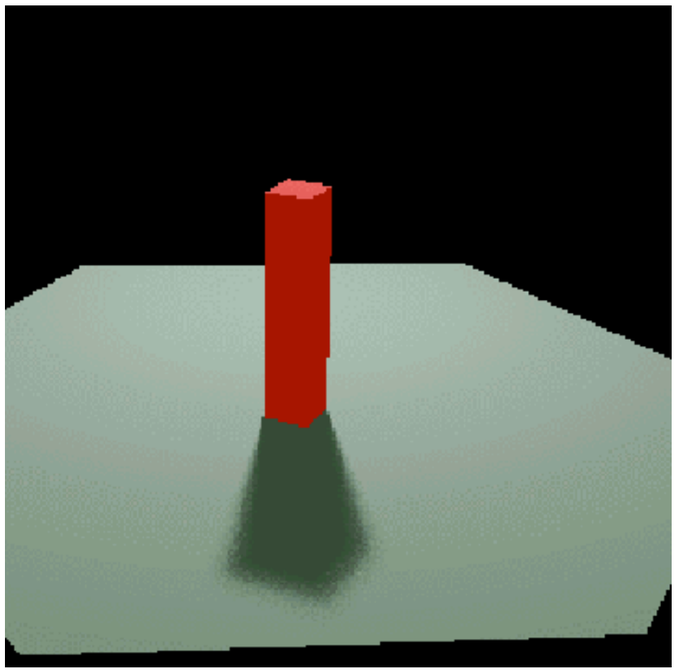




Jittering refraction ray

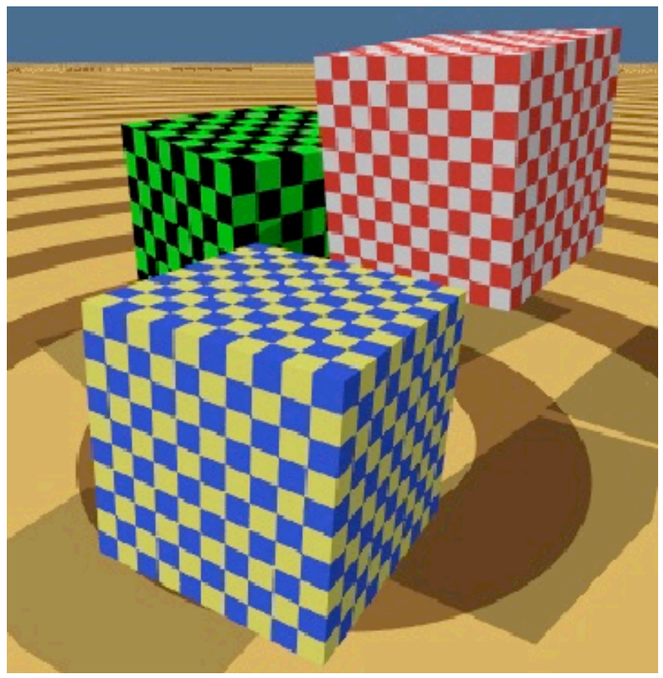
Soft shadows

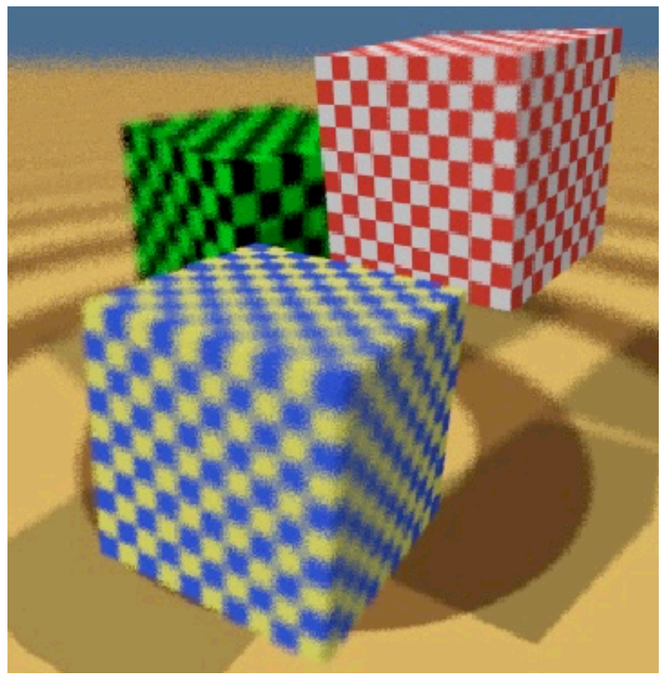




Jittering shadow ray

Depth of field (out-of-focus effect)





Jittering main ray (simulates the size of the lens)

Motion blur

Jittering time

Ray-tracing in the fragment shader

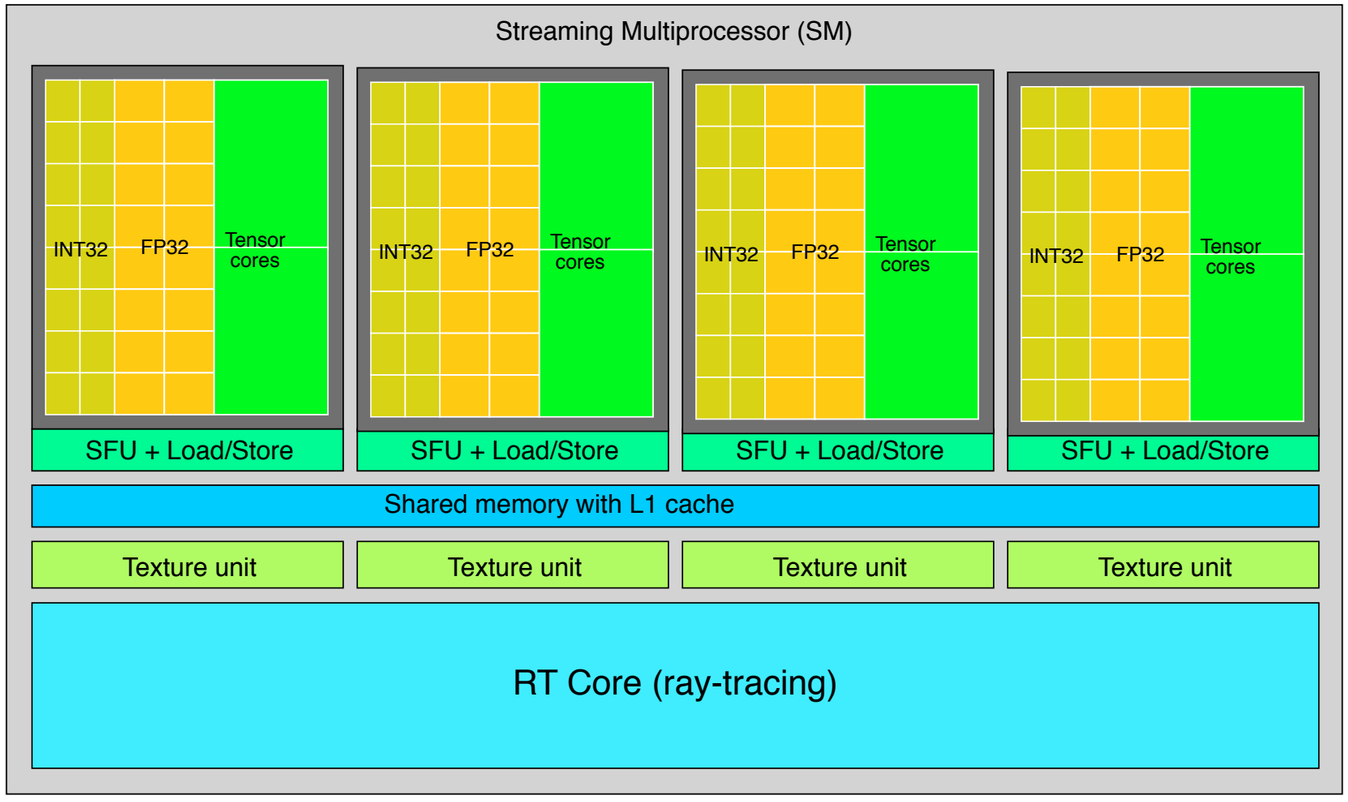
Does not fit the GPU rendering model!

- Recursion not allowed! Use lists instead!

- Limited amount of data available to a shader - needs to use textures or other buffers for input

Not impossible but we may need to use tricks, multi-pass rendering etc.

Ray-tracing on GPU often done in ”GPU computing languages” (CUDA, OpenCL) rather than shaders.



The Turing GPU and later

RTX boards

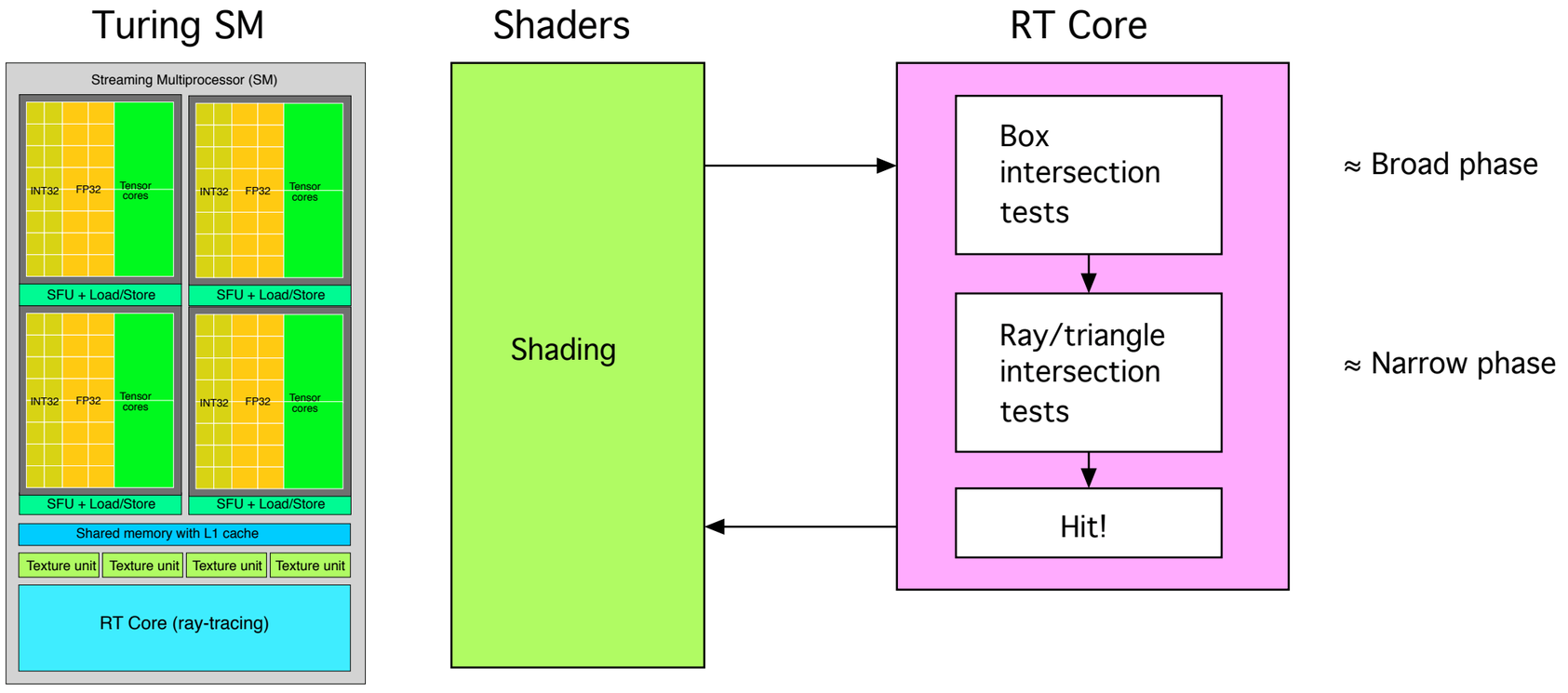
RT cores = Raytracing cores

Special-purpose hardware in RTX GPUs

Available through Vulkan, CUDA etc.

RT cores

Accelerates ray-box and ray-triangle calculations



Conclusions on ray-tracing

No longer an offline method!

Moving into real-time now!

Variations of ray-tracing a hot topic.

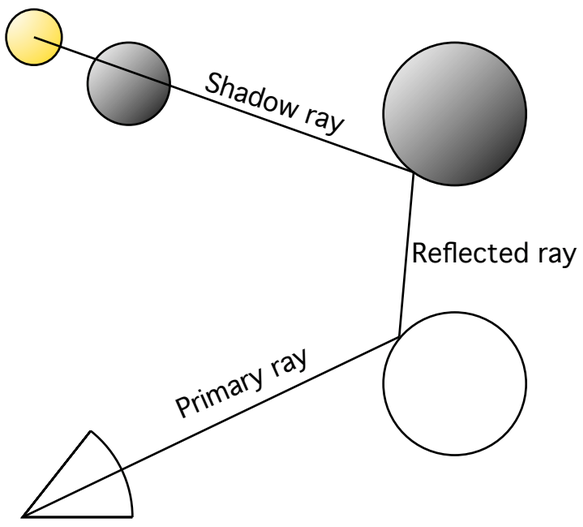
But... how about global illumination?

- Radiosity
- Photon mapping
- Path tracing

Ray-tracing limitations

Trace rays from the camera. Reflect, refract.

Shadow rays check visibility from surface to light source.



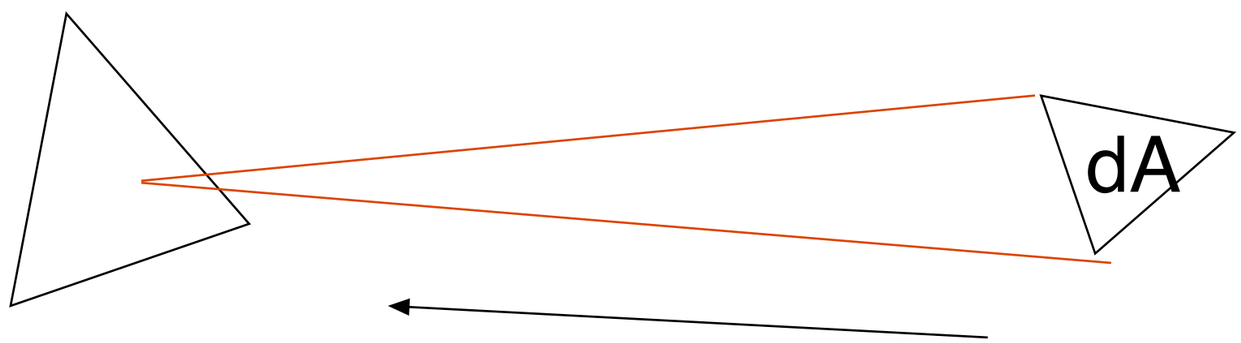
Only direct light can be handled this way!

Radiosity method

Models indirect light from surface to surface. Only diffuse surfaces.

Incoming light is summed to outgoing. Amount of light based on the solid angle, the form factor.

Accumulate, repeat to approach correct light.



Emission * form factor

The model

A surface emits energy that is a sum of reflected and emitted energy:

Energy * area = emitted + reflected

$$B_k * dA_k = E_k * dA_k + R_k * \int B_j * F_{kj} * dA_k$$

Emitted light (true light sources only)

Reflectivity

Outgoing energy from the surface element j

Form factor between j and k

Simplified to discrete patches

$$B_k = E_k + R_k * \sum B_j * F_{jk}$$

=> Equation system!

1) Solve equation system! Can be done incrementally.

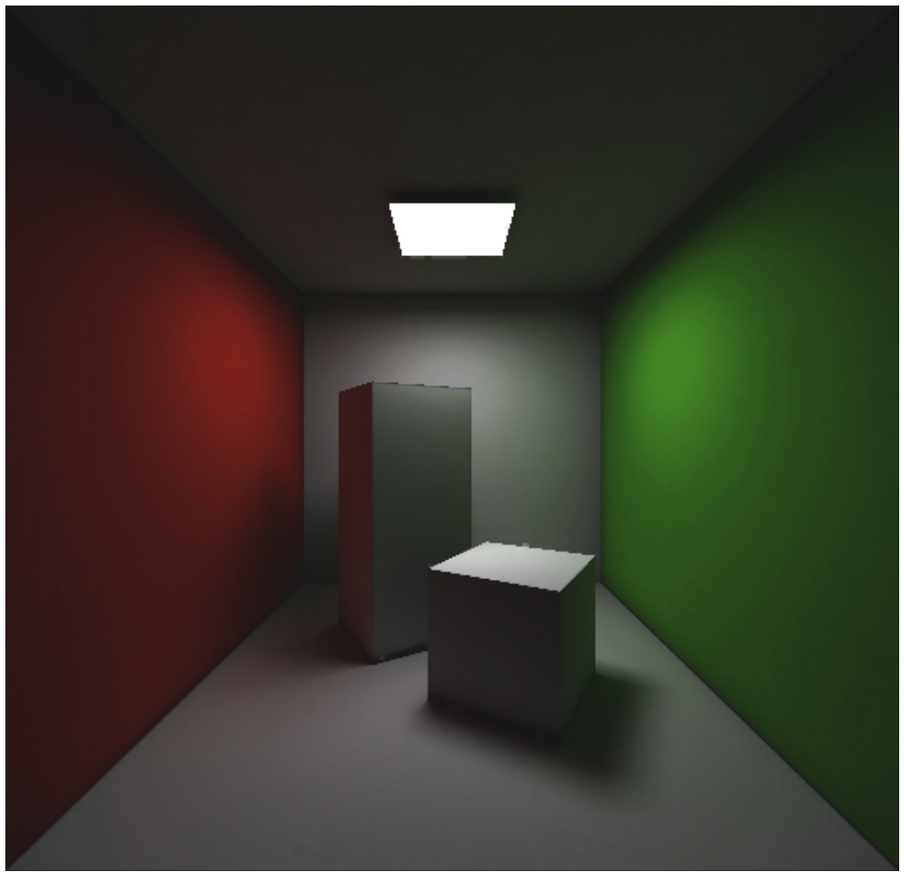
2) We must interpolate between patches (e.g. Gouraud shading)

3) The form factors F_{jk} must be calculated

Example

TSBK03 project on progressive refinement

(Variant of Cornell Box, standard scene for illumination)



More global illumination models

Very important problem, much research!

- Photon mapping
- Path tracing
- Approximation by proximity measurements
- Hybrid models
- Many variants and parallel implementations

Photon mapping

"Backwards ray-tracing"

Applies ”backwards” light (from light sources) to measure how light is scattered over a scene

Gives a good measure of indirect light

Photon Mapping

Follow rays from light source with ray-tracing methods

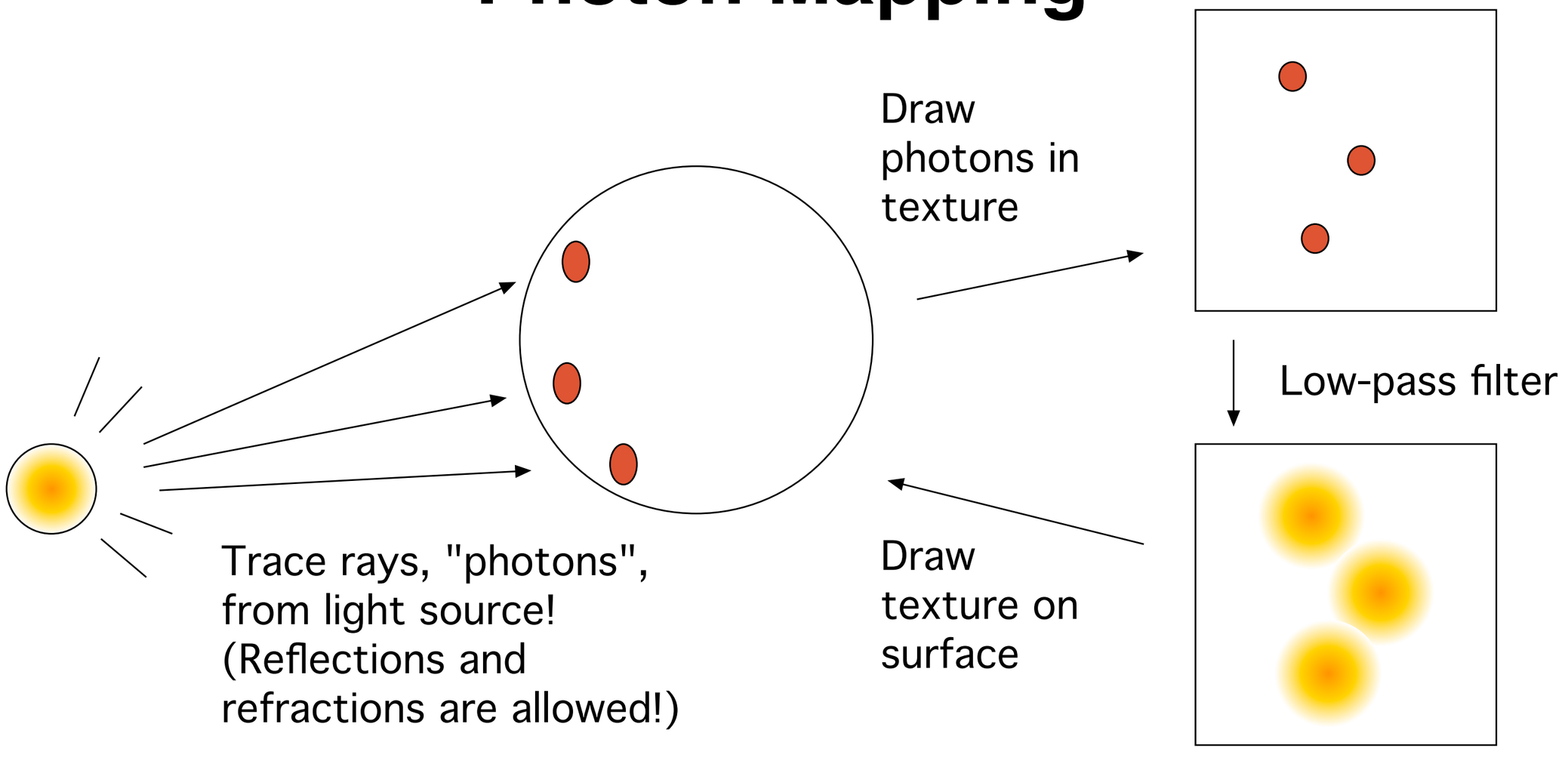
Accumulate light in ”photon maps”

Saves information about every photon - allows specular surfaces!

Low-pass filter

Then render scene using these maps as surfaces.
Handles both diffuse and specular reflections!

Photon Mapping

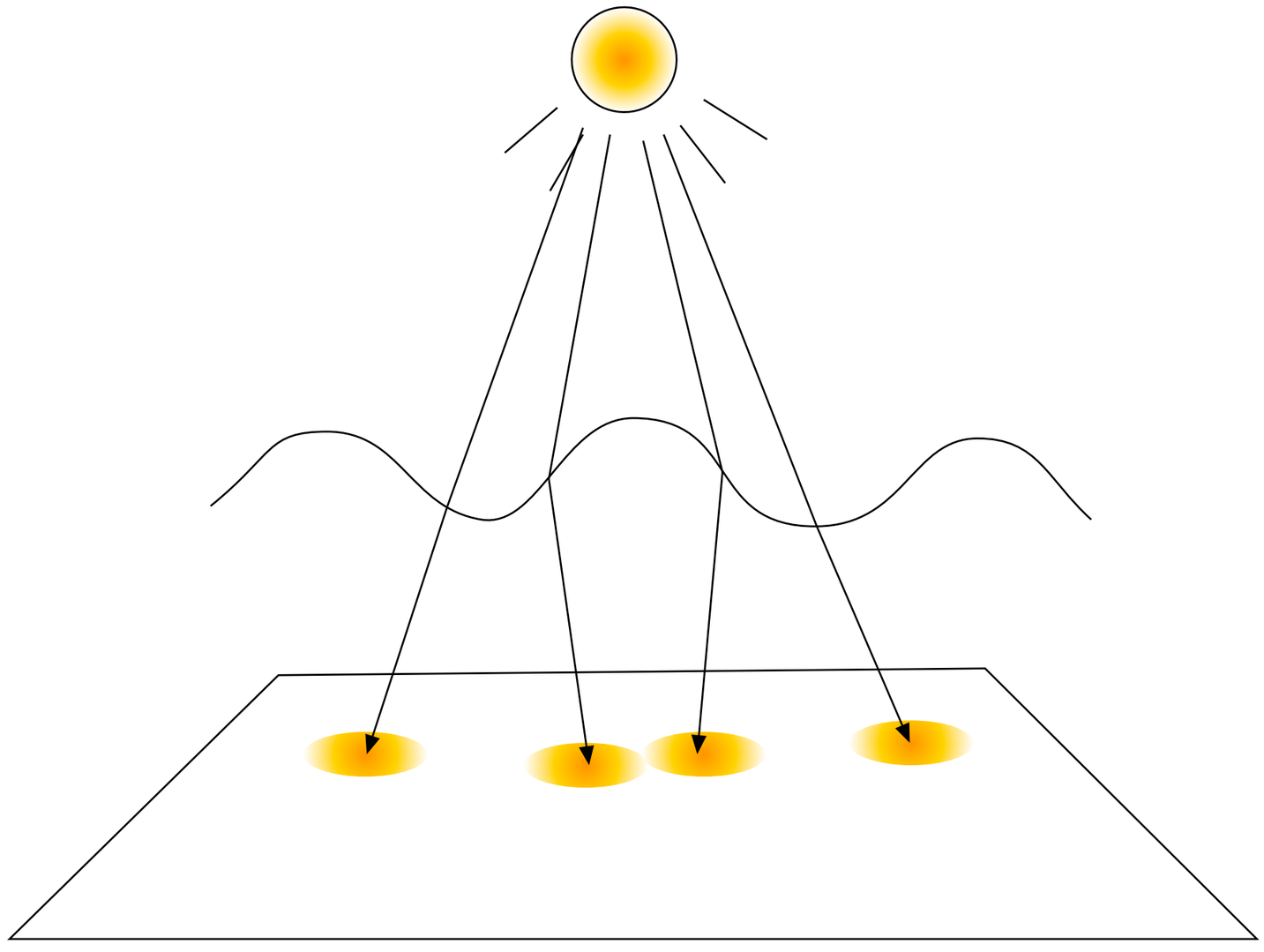


Caustics

Nice case for photon mapping

Bottom of lake only surface to illuminate

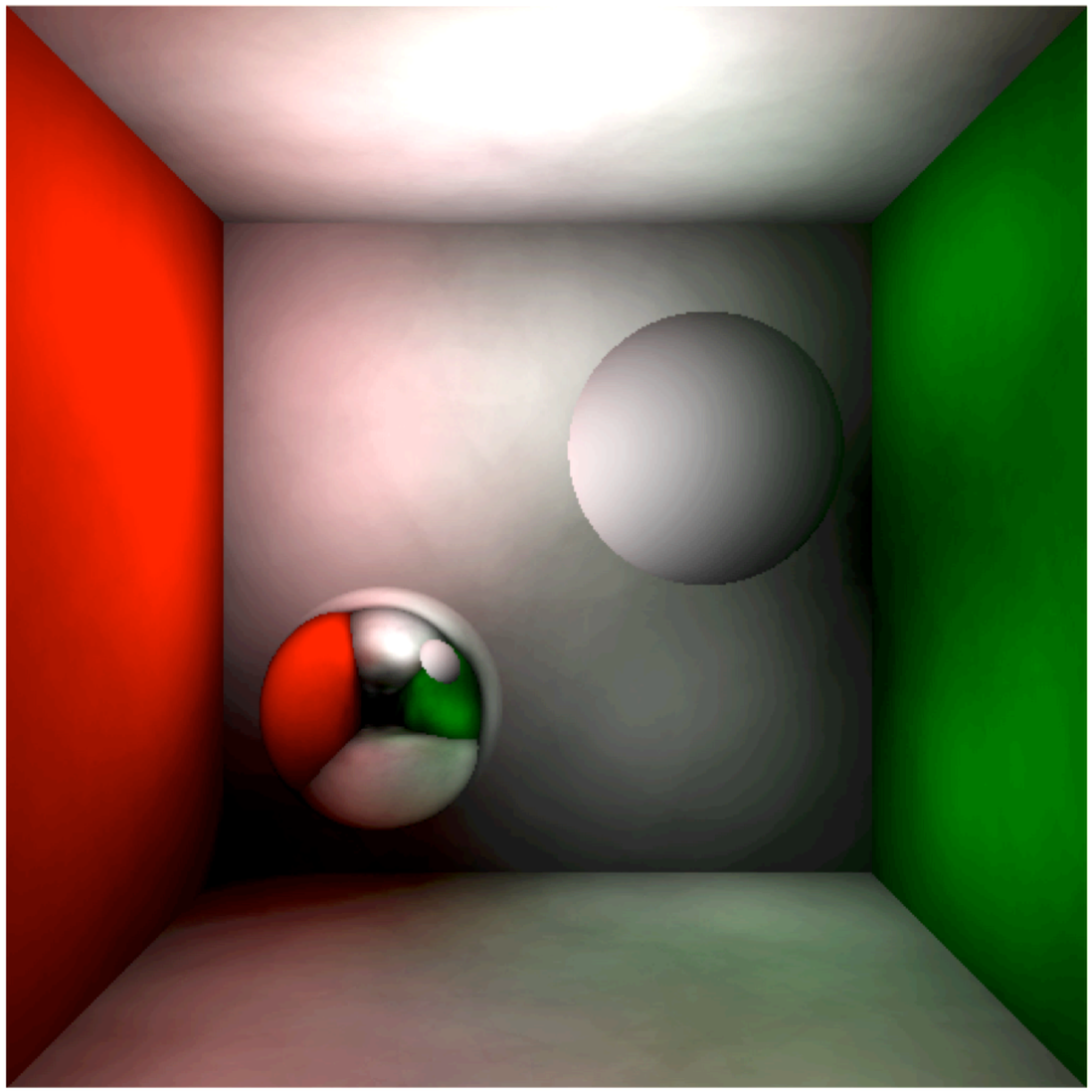
(Harder if the lake bottom is not flat or if there are several objects to illuminate.)



Photon Mapping Example

(The Schindler demo)

Typical features: Reflections, caustics and diffuse shadows



Path tracing

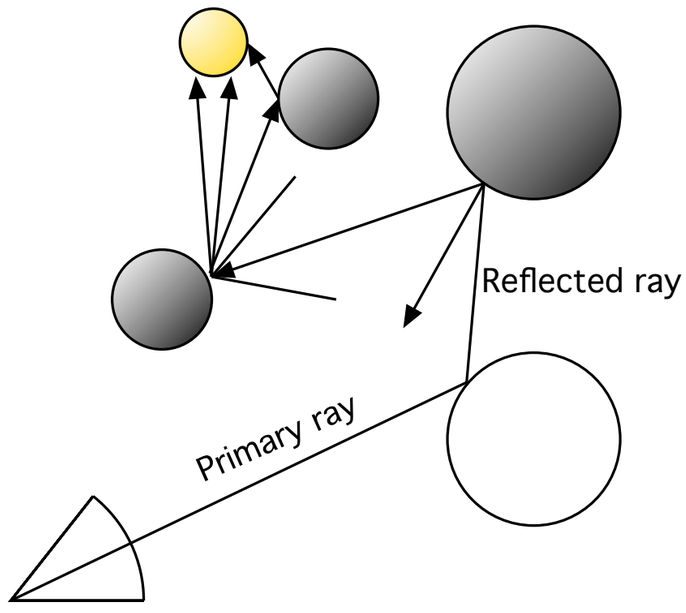
Similar to ray-tracing, but more rays.

Cast rays to search for "light paths".

Essentially cast multiple "shadow rays" to find indirect light.

"Backwards" path tracing

When hitting that opaque surface, cast many "shadow rays" recursively and see if they hit a light source! Search for light paths!



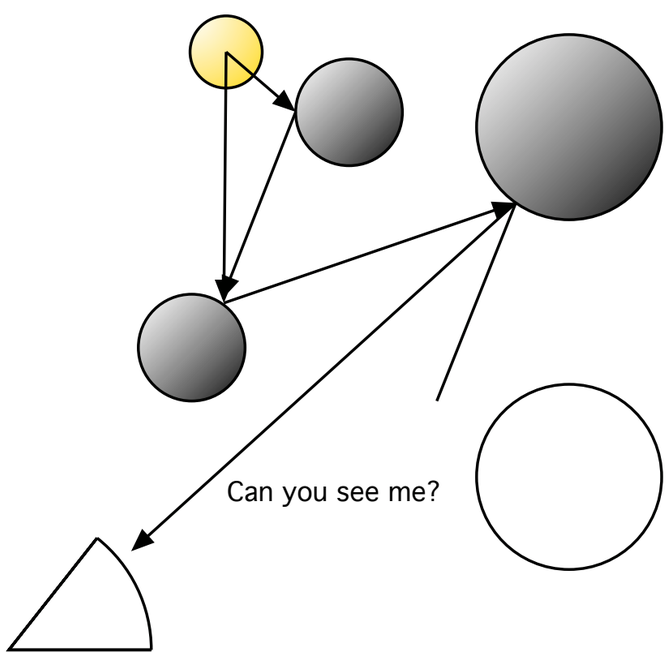
"Forwards" path tracing

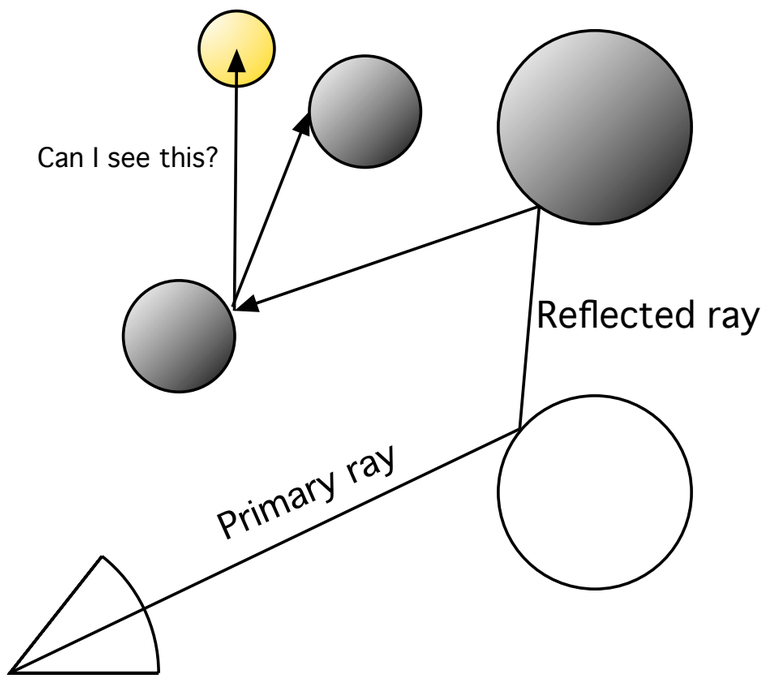
Instead of going from the camera, go from the light source, trace until you hit the camera.

Both cases can be accelerated by traditional "shadow rays" to detect that a surface has direct light or a direct path to the camera.

Shadow rays still relevant

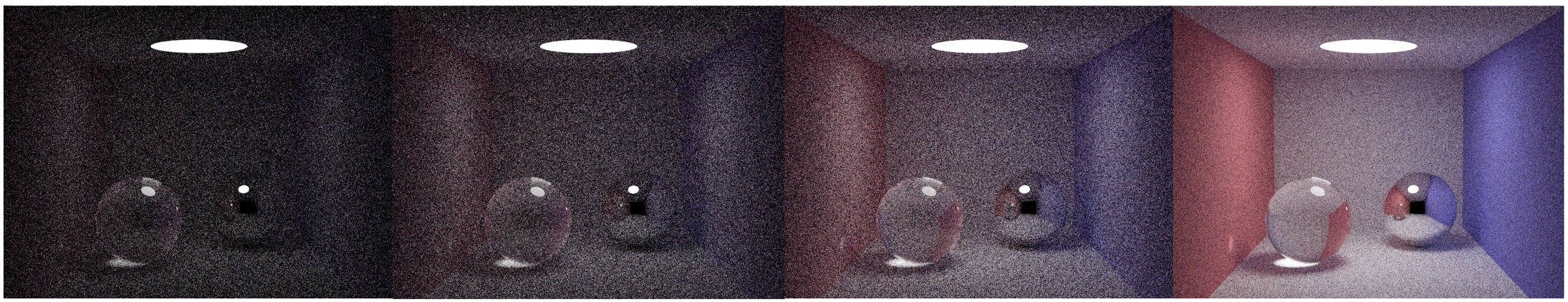
Both cases use traditional "shadow rays" to detect that a surface has direct light or a direct path to the camera.





Results (using Kevin Beason's "smallpt")

Small path tracer available on-line.



4 spp (21s) | 8 spp (42s) | 16 spp | 64 spp

We need *many* rays to get a good result!
Beason reports thousands!
Parallel computing vital!

Beason's path tracer

Casts multiple primary rays. Secondary rays make random bounces but never multiply!

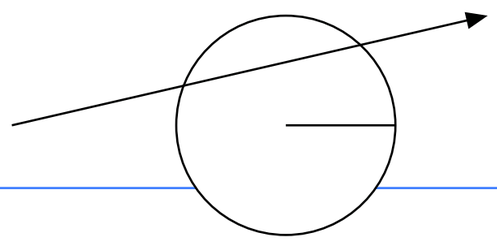
Uses shadow rays.

Uses light models for more realistic light.

Beason likes spheres

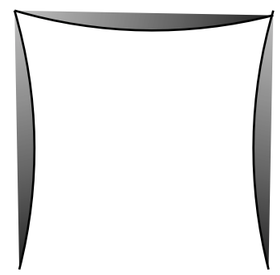
All parts of the scene are spheres! The walls are large spheres!

```
double intersect(const Ray &r)
const
{
    // returns distance, 0 if nohit
    Vec op = p-r.o;
    // Solve t^2*d.d + 2*t*(o-p).d + (o-p).(o-p)-R^2 = 0
    double t, eps=1e-4, b=op.dot(r.d), det=b*b-op.dot(op)+rad*rad;
    if (det<0)
        return 0;
    else
        det=sqrt(det);
    return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
}
```



As in the book but differently expressed.

```
// Scene description. Even the walls are spheres!
Sphere spheres[] =
{
      //Scene: radius, position, emission, color, material
      Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(),Vec(.75,.25,.25),DIFF),//Left
      Sphere(1e5, Vec(-1e5+99,40.8,81.6),Vec(),Vec(.25,.25,.75),DIFF),//Rght
      Sphere(1e5, Vec(50,40.8, 1e5),     Vec(),Vec(.75,.75,.75),DIFF),//Back
      Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(),Vec(),           DIFF),//Frnt
      Sphere(1e5, Vec(50, 1e5, 81.6),    Vec(),Vec(.75,.75,.75),DIFF),//Botm
      Sphere(1e5, Vec(50,-1e5+81.6,81.6),Vec(),Vec(.75,.75,.75),DIFF),//Top
      Sphere(16.5,Vec(27,16.5,47),       Vec(),Vec(1,1,1)*.999, SPEC),//Mirr
      Sphere(16.5,Vec(73,16.5,78),       Vec(),Vec(1,1,1)*.999, REFR),//Glas
      Sphere(600, Vec(50,681.6-.27,81.6),Vec(12,12,12),  Vec(), DIFF)
      //Lite
};
```

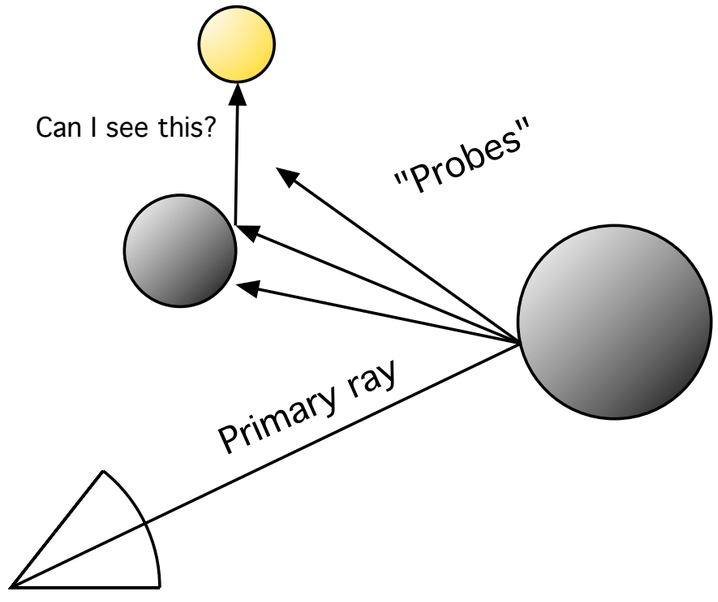


Path tracing in fragment shader

Hours to produce a noisy image? A bit better with openmp...?

But how about doing it on the GPU?

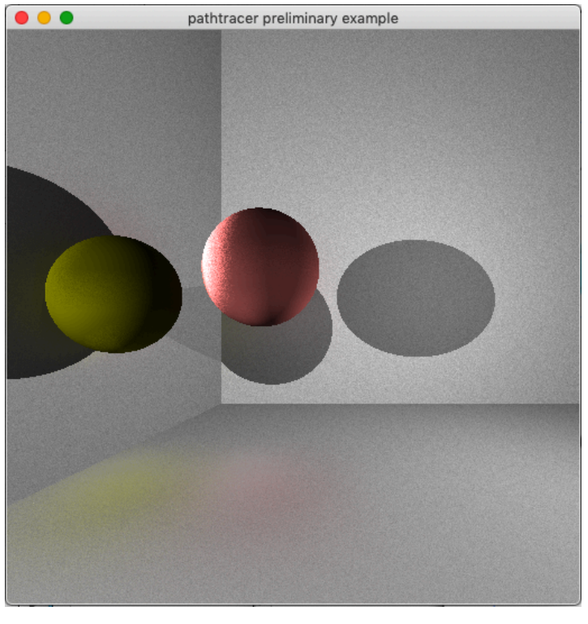
My first attempts: Simple path tracer:

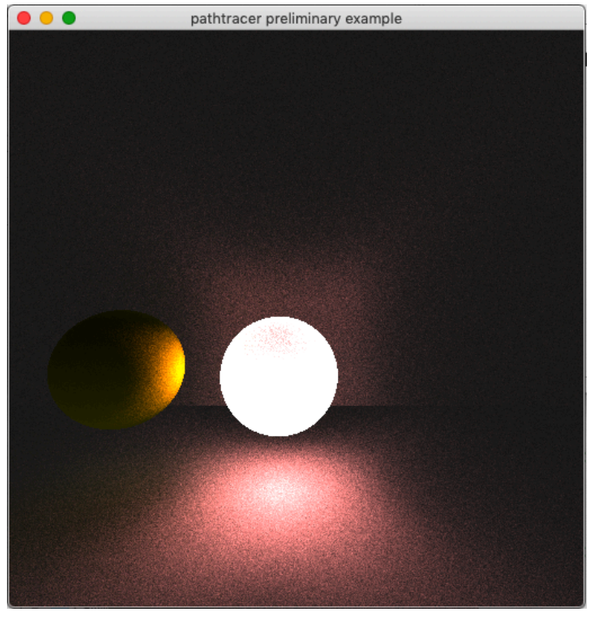


My simplified path tracer

Two variants: One with point lights, one with only emissive objects.

Pretty OK in real time (or almost) on my 650M. Cheating?





Summary

Ray-tracing:

Searches for surfaces and direct light. Good for shiny surfaces, transparency etc. “Hard” images.

Radiosity:

Models light patch to patch. Good for realistic images of diffuse surfaces. Can not handle specular reflections!

Photon Mapping: accumulates light in surfaces

Path tracing: searches in multiple directions for light paths