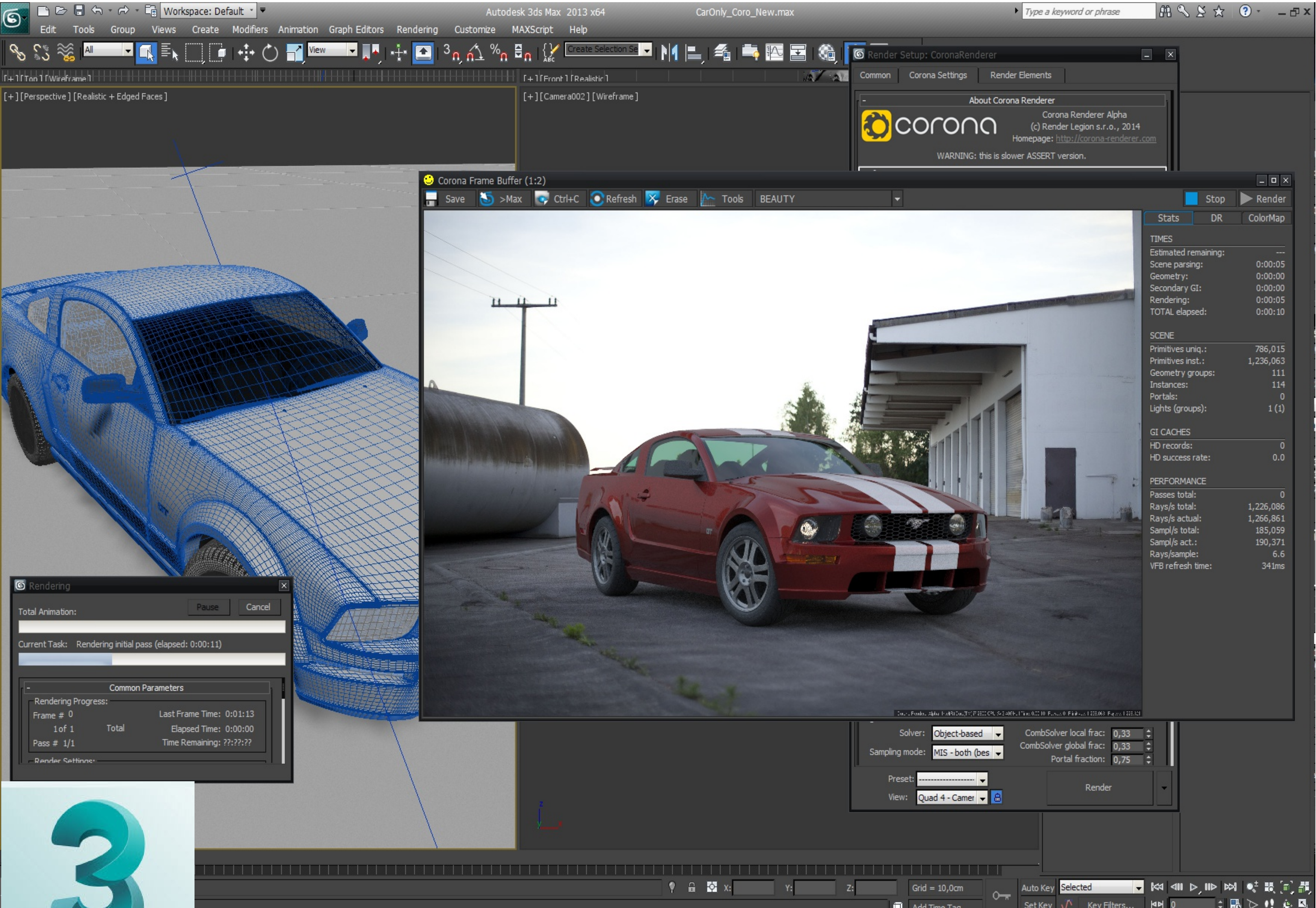
Rendering & Ray Tracing

CGG MFF UK Praha

pepca@cgg.mff.cuni.cz

https://cgg.mff.cuni.cz/~pepca/


C271
Louisville
PRO



Autodesk 3ds Max 2013 x64
CarOnly_Coro_New.max
Workspace: Default
Type a keyword or phrase
Edit
Tools
Group
Views
Create
Modifiers
Animation
Graph Editors
Rendering
Customize
MAXScript
Help
[+] [Perspective] [Realistic + Edged Faces]
[+] [Camera002] [Wireframe]
Render Setup: CoronaRenderer
Common
Corona Settings
Render Elements
About Corona Renderer
corona
Corona Renderer Alpha
(c) Render Legion s.r.o., 2014
Homepage: http://corona-renderer.com
WARNING: this is slower ASSERT version.
Corona Frame Buffer (1:2)
Save
>Max
Ctrl+C
Refresh
Erase
Tools
BEAUTY
Stop
Render
Stats
DR
ColorMap
TIMES
Estimated remaining: ---
Scene parsing: 0:00:05
Geometry: 0:00:00
Secondary GI: 0:00:00
Rendering: 0:00:05
TOTAL elapsed: 0:00:10
SCENE
Primitives uniq.: 786,015
Primitives inst.: 1,236,063
Geometry groups: 111
Instances: 114
Portals: 0
Lights (groups): 1 (1)
GI CACHES
HD records: 0
HD success rate: 0.0
PERFORMANCE
Passes total: 0
Rays/s total: 1,226,086
Rays/s actual: 1,266,861
Sampl/s total: 185,059
Sampl/s act.: 190,371
Rays/sample: 6.6
VFB refresh time: 341ms
Rendering
Total Animation:
Pause
Cancel
Current Task: Rendering initial pass (elapsed: 0:00:11)
Common Parameters
Rendering Progress:
Frame # 0
1 of 1
Total
Pass # 1/1
Last Frame Time: 0:01:13
Elapsed Time: 0:00:00
Time Remaining: ??:??:??
Render Settings:
Solver: Object-based
Sampling mode: MIS - both (bes
CombSolver local frac: 0,33
CombSolver global frac: 0,33
Portal fraction: 0,75
Preset:
View: Quad 4 - Camer
Render
Grid = 10,0cm
Add Time Tag
Auto Key
Selected
Set Key
Key Filters...

Applications

Design, architecture, art

- indoor light propagation

Entertainment

- <u>cinema and TV</u> (IL&M, Pixar, DreamWorks… ”off-line“)
- video games (”real-time“)

Media

- television (virtual studios…)
- <u>advertisement</u>

Advertisement

© Bertrand Benoit

Architectural Visualization

© Pavel Stavila

Possible goals of realistic rendering

Accurately **mimic the world** around us (or a fantasy world)

- a virtual scene in a computer

Accurately **simulate the propagation of light** in a scene?

- "predictive rendering"

Or just **"believable rendering"**…?

- the layperson should not know that the image is artificial

Rendering speed

- **"offline" rendering** (advertising, film – computer farms, etc.)
- **"Real-time"** (min. 25 fps)

What we will cover in this course

Accurately **mimic the world** around us (or a fantasy world)

- a virtual scene in a computer

~~Accurately **simulate the propagation of light** in a scene?~~

- ~~"predictive rendering"~~

Or just **"believable rendering"**…?

- the layperson should not know that the image is artificial

Rendering speed

- **"offline" rendering** (advertising, film – computer farms, etc.)
- ~~**"Real-time"** (min. 25 fps)~~

Not covered!

Real-time graphics in video games

- although realistic rendering is getting here nowadays

GPU programming (one little exception – RTX)

3D scene modeling (Maya, 3DS MAX, Blender…)

Animation

Simulation of physical phenomena

- waves, explosions, sky, clouds…

Image analysis, computer vision, artificial intelligence

- nor the use of machine learning in rendering

History – Classical Rendering

Straßer, Catmull 1974: **Z-buffer**

Faceted model

- most often triangular meshes

Visibility calculation

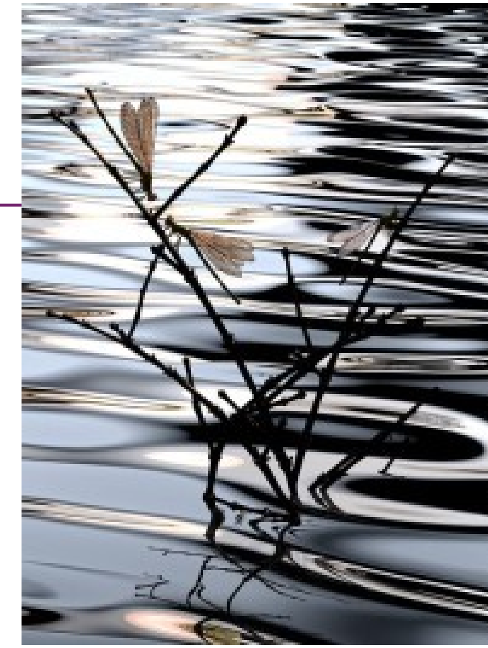
- Z-buffer

Approximate **lighting conditions**

- local lighting model, shadow casting

Contemporary **GPUs: textures, shadows, shaders, multiple passes, deferred lighting…**

History – Ray-tracing



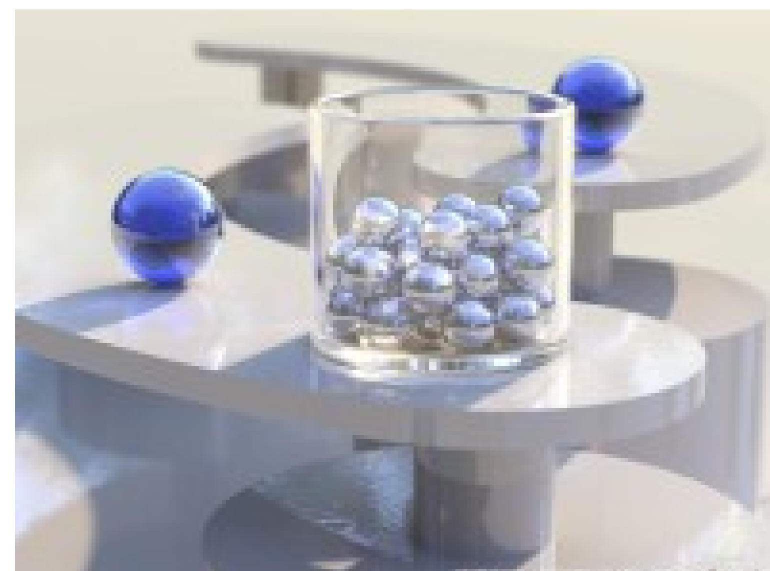
Whitted 1979: **backward Ray-tracing**

Geometric approach

- only the ideally reflected ray is traced

Computationally very demanding **calculation of the intersection** of the ray with the scene

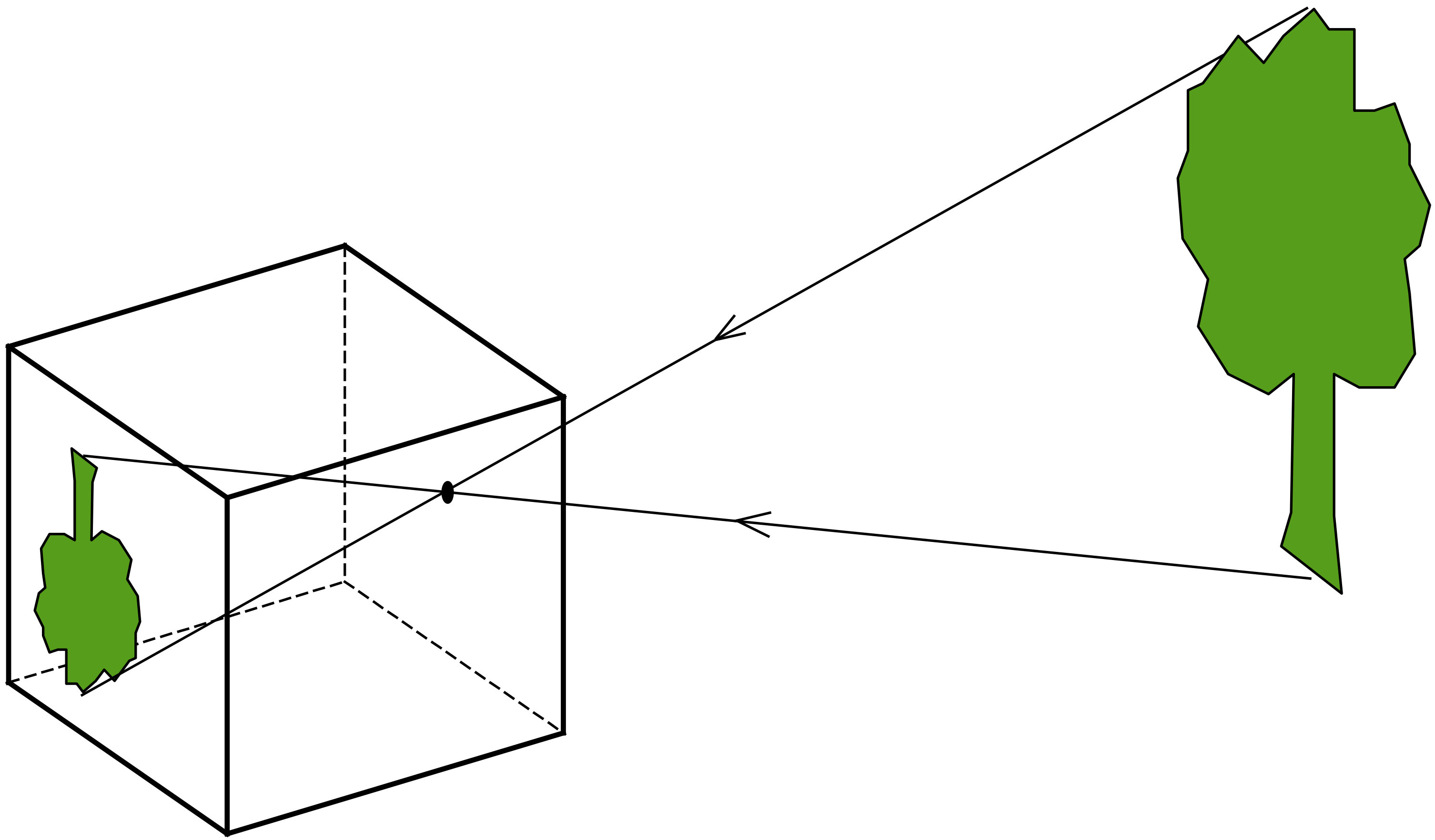
- >90% time $\rightarrow$ acceleration



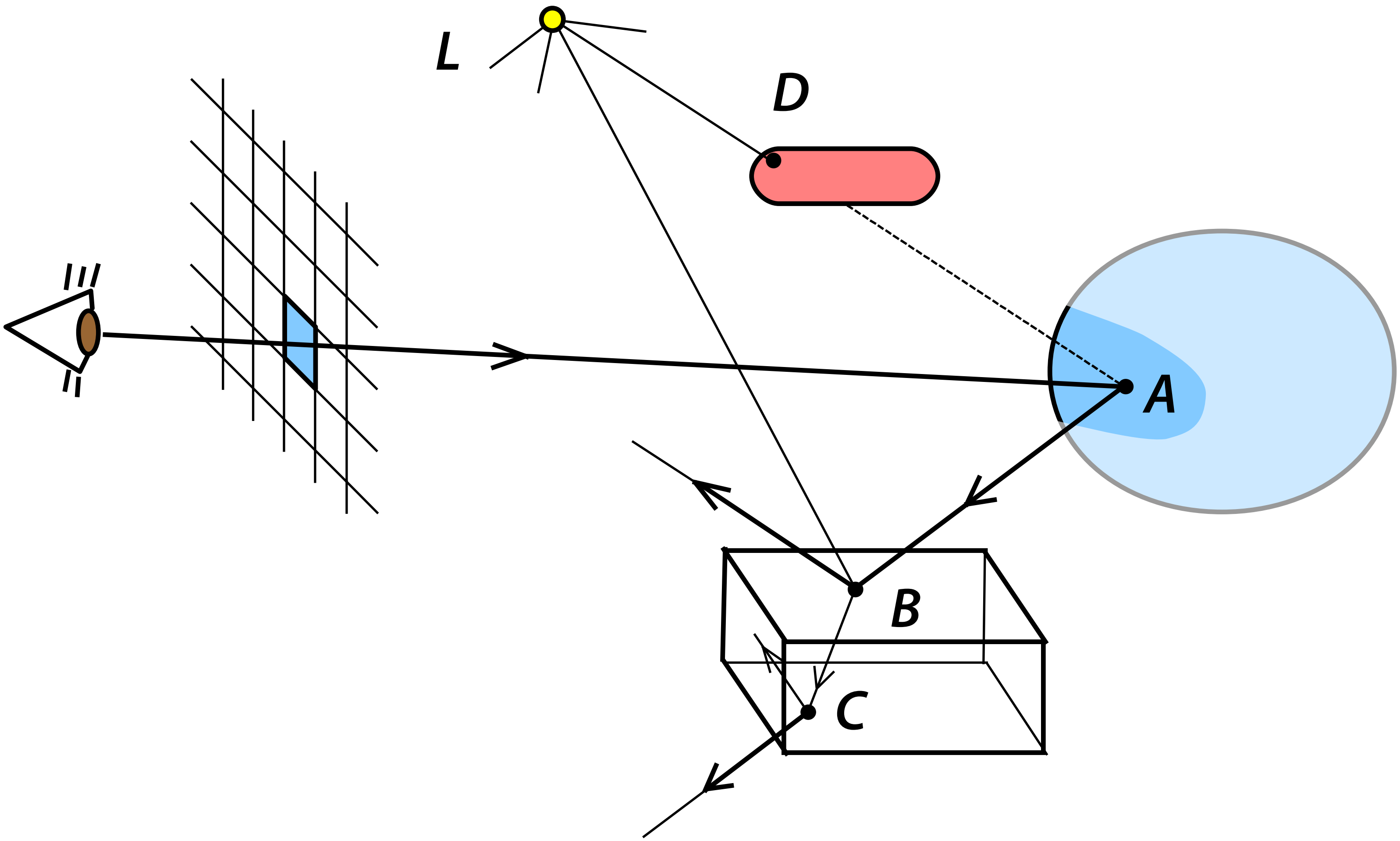
Easy **appearance improvements**

- textures, anti-aliasing, “shaders”
- distributed R-T (Monte-Carlo)

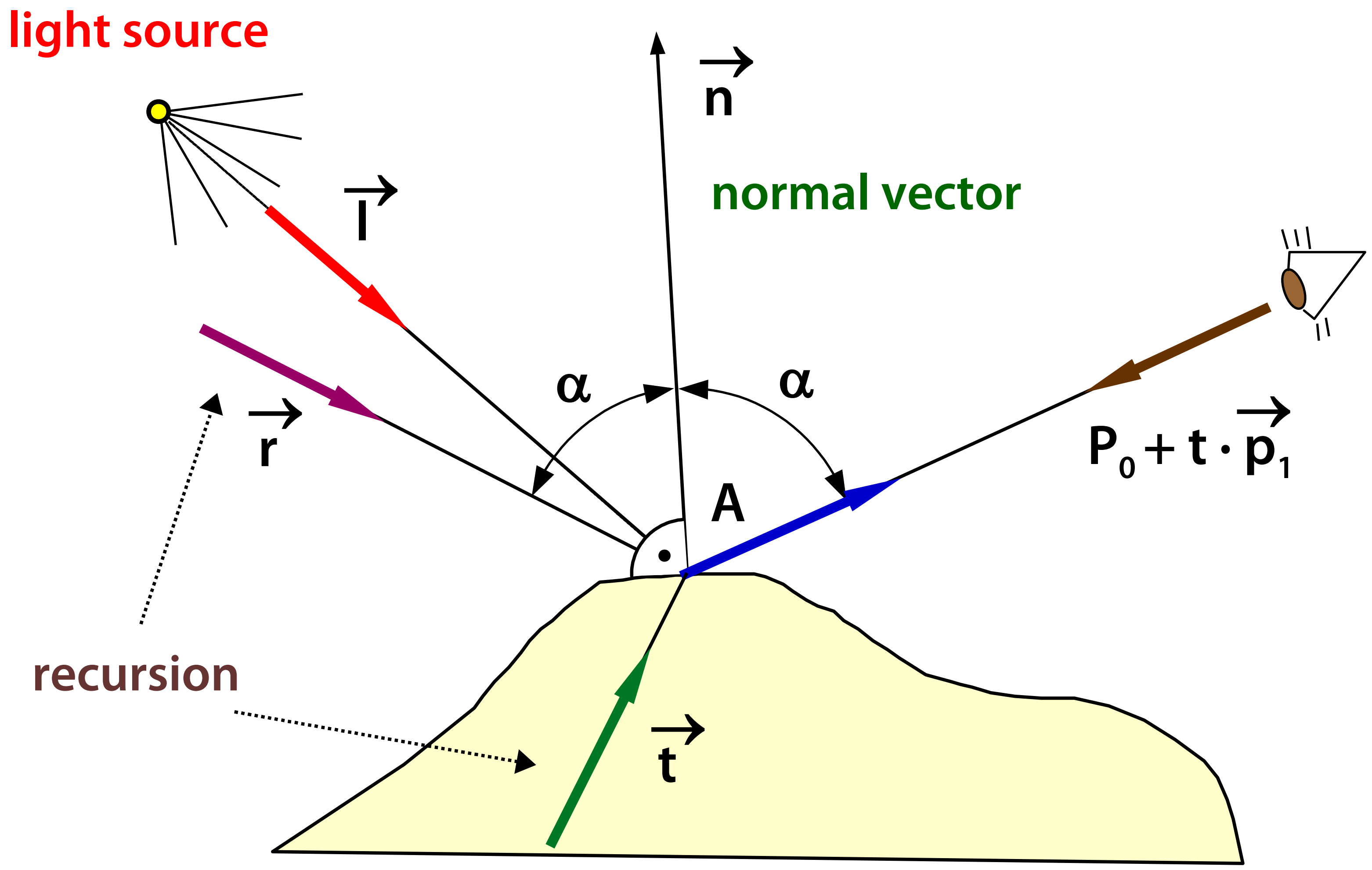
Pinhole Camera Model (Camera Obscura)



Backward Ray Tracing



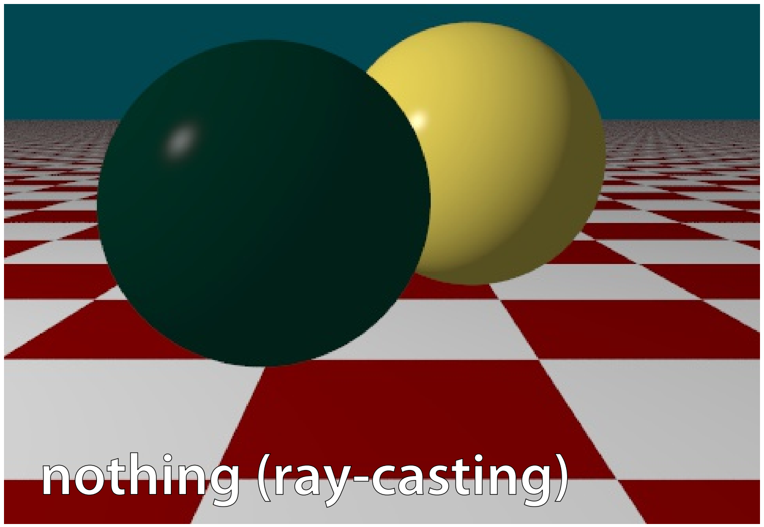
Lighting Computations

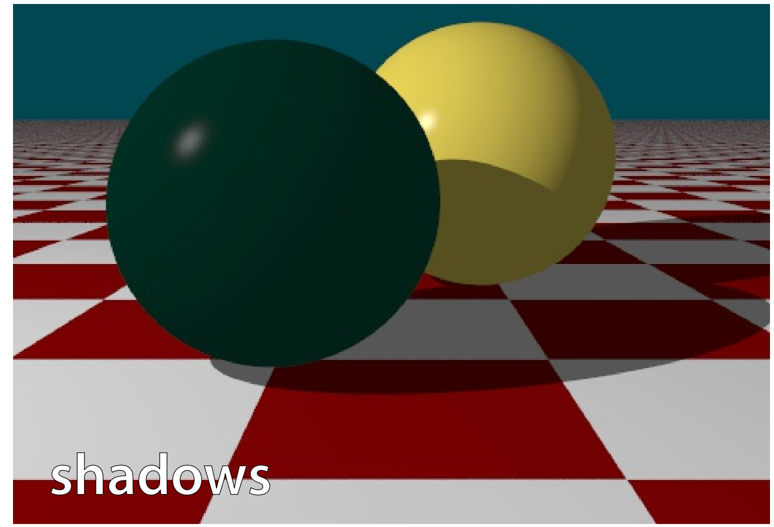


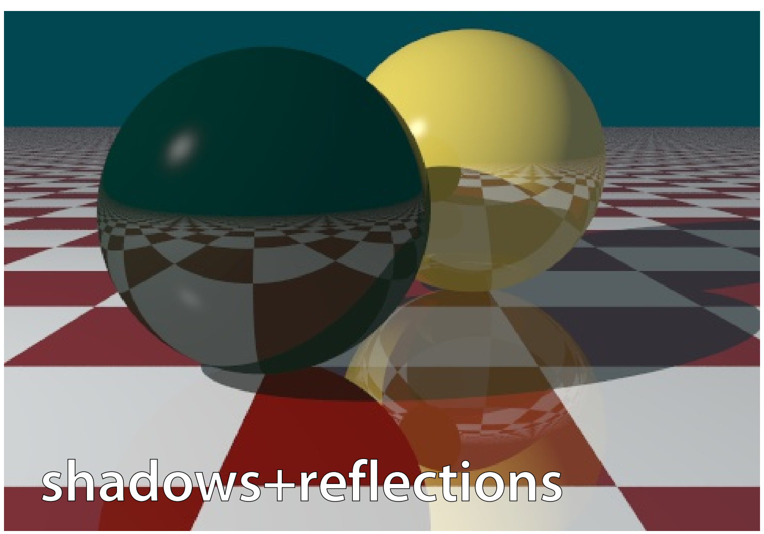
Recursive implementation

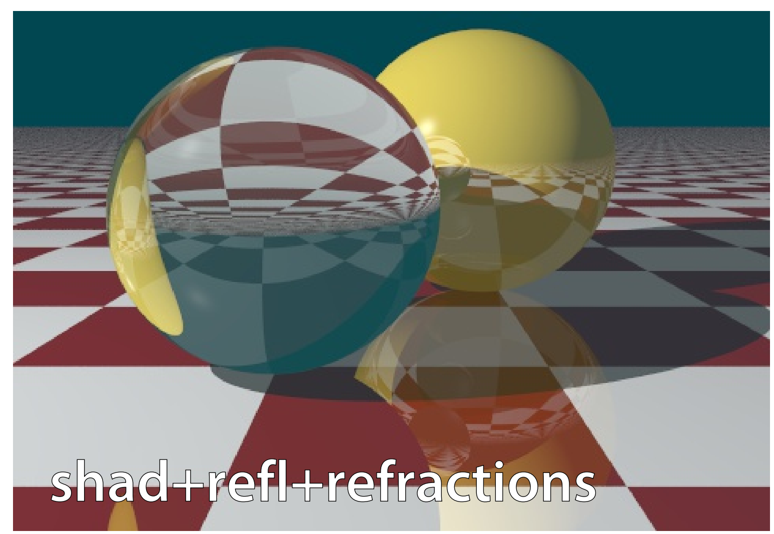
```
int maxDepth = 10;              // Maximum recursion level
RayScene scene;                 // Global scene object (geometry, materials, light sources…)

RGB shade (Vector3d P0, Vector3d p1, int depth)
  // P0 – ray origin, p1 – ray direction, depth – interactions so far
{
  Vector3d A = intersection(scene, P0, p1);
  if (!isValid(A)) return scene.background; // No intersection at all

  RGB color{0};                 // Result color
  for (const auto& light : scene.lightSources)
    if (!isValid(intersection(scene, A, light.point - A)))
      color += scene.kL(A) * light.contribution(A, -p1, scene.material(A), scene.normal(A));

  if (++depth >= maxDepth) return color;

  if (scene.isGlossy(A))      // Recursion - reflection
  {
    Point3d r = reflection(p1, scene.normal(A));
    color += scene.kR(A) * shade(A, r, depth);
  }

  if (scene.isTransparent(A)) // Recursion – refraction
  {
    Point3d t = refraction(p1, scene.normal(A), scene.index(A));
    color += scene.kT(A) * shade(A, t, depth);
  }

  return color;
}
```

Individual Components









Recursion Management

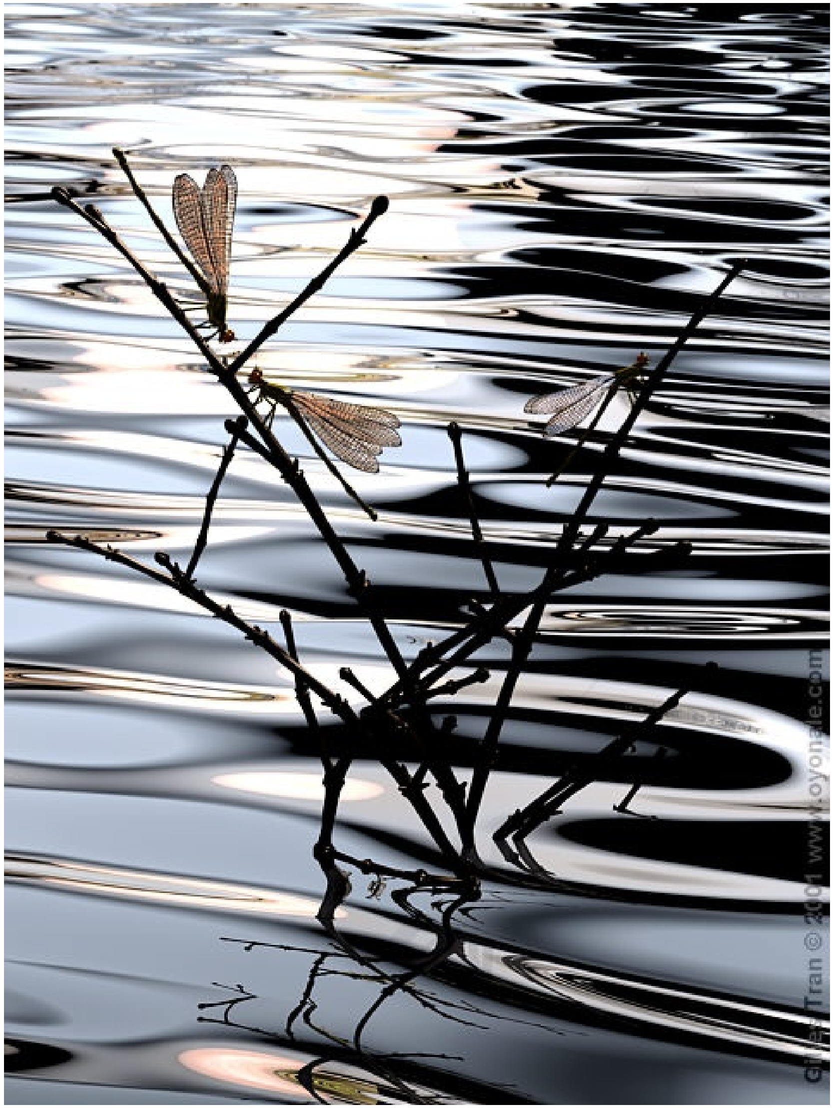
Static – limited by a constant (not suitable for scenes with many mirrors and less glossy surfaces simultaneously)

Dynamic – according to performance/importance of the ray

- "**performance**" is the percentage the ray can still contribute to the color of a given pixel (primary rays: 100%)
- limit on the "performance" constant (e.g. 1-2%)

Combined – limit on the recursion depth, and the "performance" of the ray

More Examples



More Examples

More Examples

More Examples

Intersection Computations

Geometric computation, result is composed of

- intersection coordinates (1D is enough, special value “infinity”)
- normal vector of the surface
- 2D texture coordinates
- “id“ of the solid (surface) that was hit $\rightarrow$ color, material…

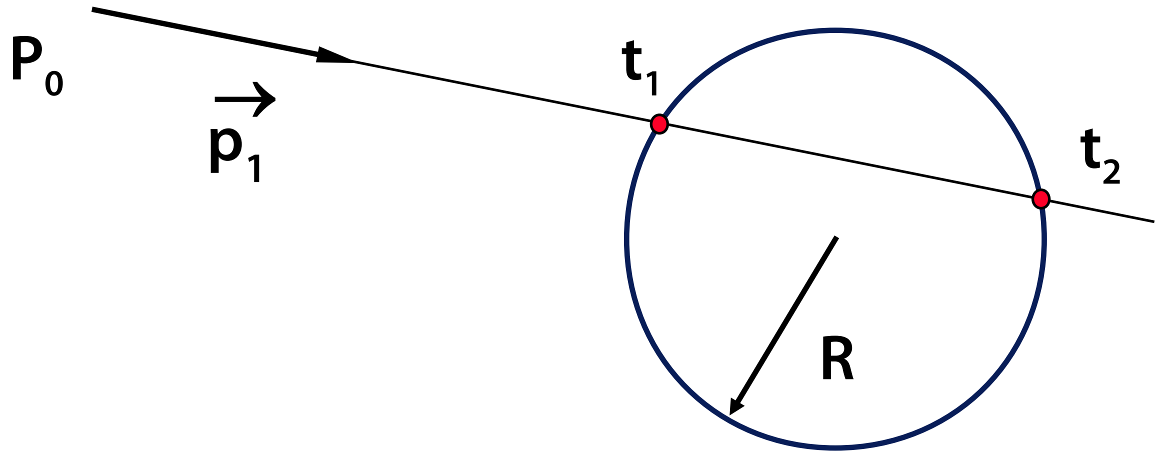
Time-consuming operation (90-95% of render time)

- acceleration techniques extremely important

Analytical solution (sphere, cylinder, cube, triangle…)

Numerical solution (subdivision surfaces, higher order surfaces, rotational surfaces, implicit surfaces…)

Ray-Sphere Intersections



$P_0 = [x_0, y_0, z_0]$

$\vec{p}_1 = [x_1, y_1, z_1]$

Ray: $\mathbf{P(t) = P_0 + \mathit{t}\,\vec{p}_1}, \quad \mathit{t} > 0$ (1)

Sphere (at origin): $\mathbf{x^2 + y^2 + z^2 - R^2 = 0}$ (2)

After substituting (1) into (2) we obtain a **quadratic equation (*t*)**

$$\mathbf{t^2\,(x_1^2 + y_1^2 + z_1^2) + 2t\,(x_0x_1 + y_0y_1 + z_0z_1) + x_1^2 + y_1^2 + z_1^2 - R^2 = 0}$$

Ray Intersections with CSG

For **elementary solids**, intersectionscan be calculated

- start and end of ray traversal through a [convex] solid body

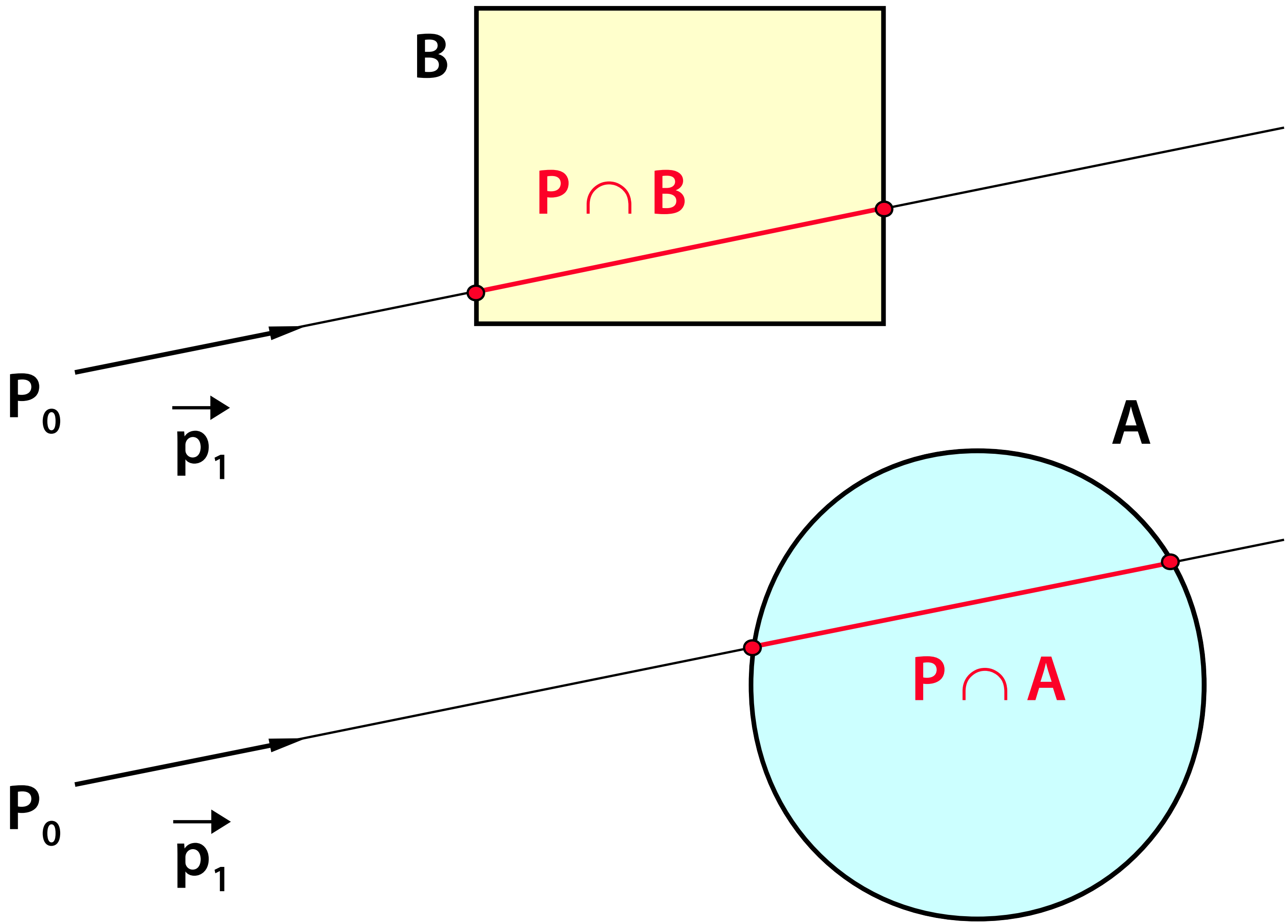
Set theoretic operations on all intersections along the ray

- distributive: $\mathbf{P \cap (A - B) = (P \cap A) - (P \cap B)}$
- the usual ray-object intersection is a sequence of intervals

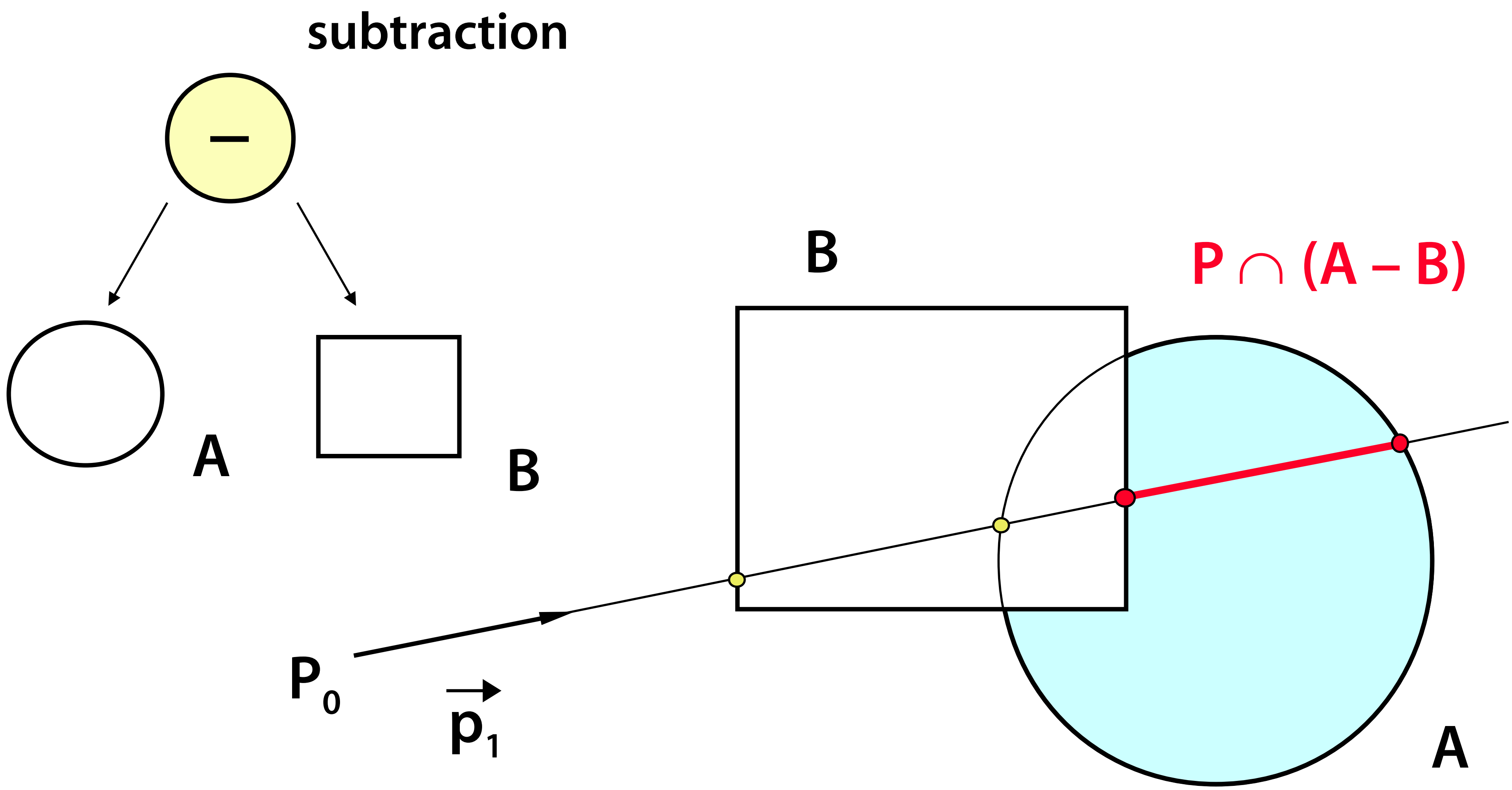
Geometric transformations

- the inverse transformation is applied to the ray

Intersections $P \cap A$, $P \cap B$



Intersection P ∩ (A – B)



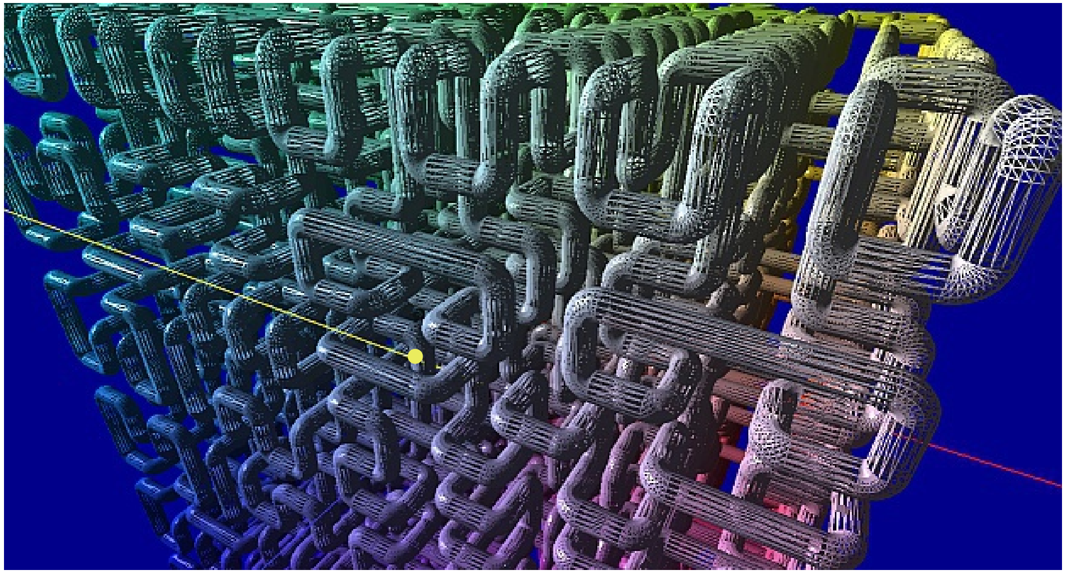
Intersections with Traingle Mesh

Scene = **triangle mesh**

- simple idea (simple API, even for GPU)
- any useful scene geometry can be approximated in this way

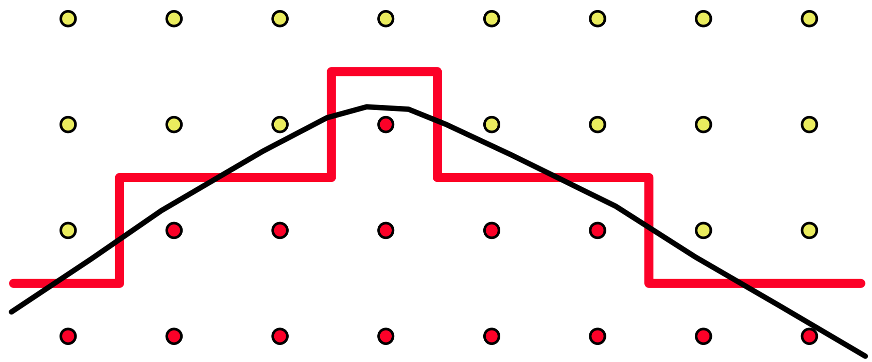
Simple test **“Ray – triangle“**

- huge amount of triangles (10^6 to 10^{10}), ***O(N)*** is too slow
- acceleration methods try to get better complexity: ***O(log N)***



$N \approx 10^6$

Anti-aliasing

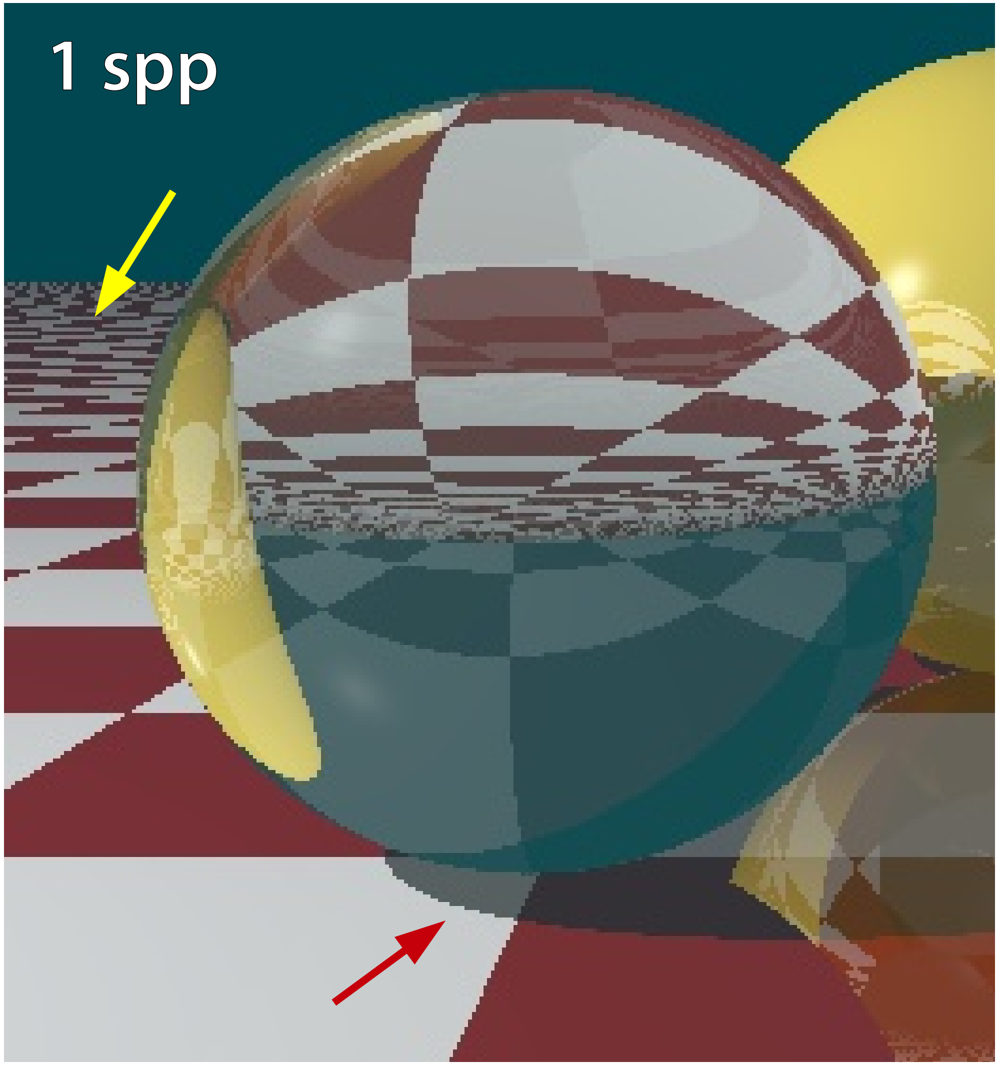


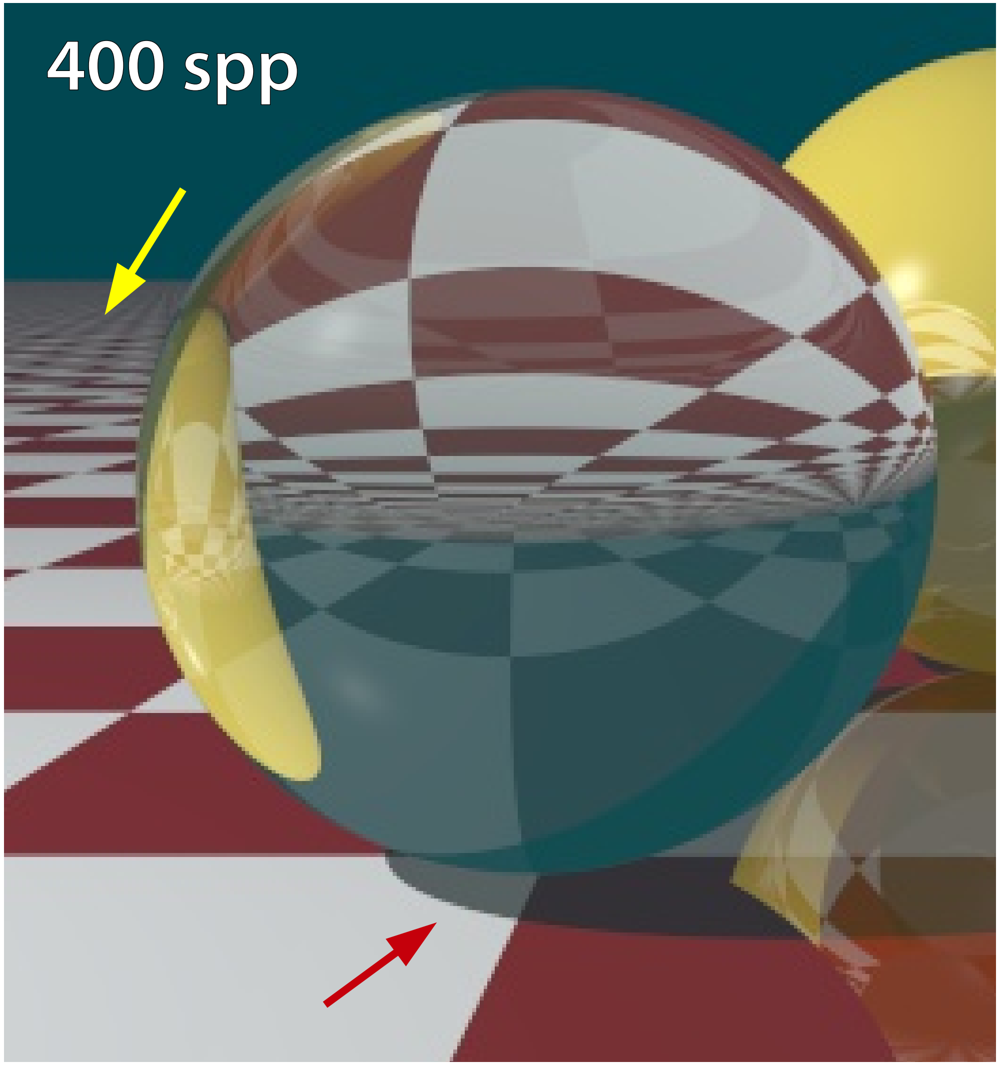
Only one ray per pixel leads to "**aliasing**"

- jagged edges
- interference

Increased resolution only partially solves the problem (and at great cost)

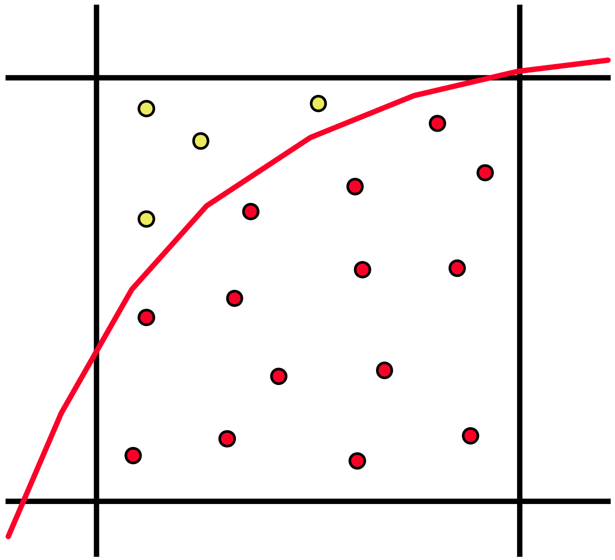
Anti-aliasing (Super-sampling)

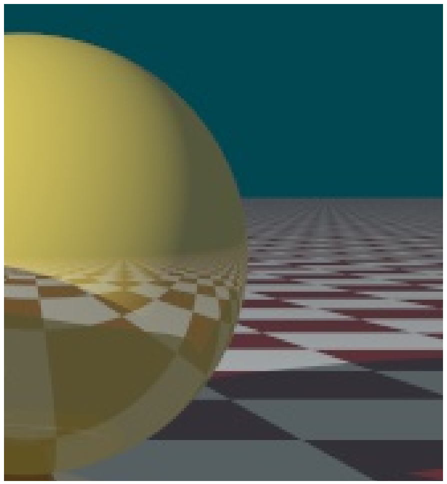




spp = samples per pixel

Multiple Sampling





Multiple rays are fired into each pixel

Resulting color is an **arithmetic average**

Transitions are **smoother** (no jaggies)

The rays should cover the pixel area **evenly**, but not regularly!

Textures

Changing the **color** on object surfaces

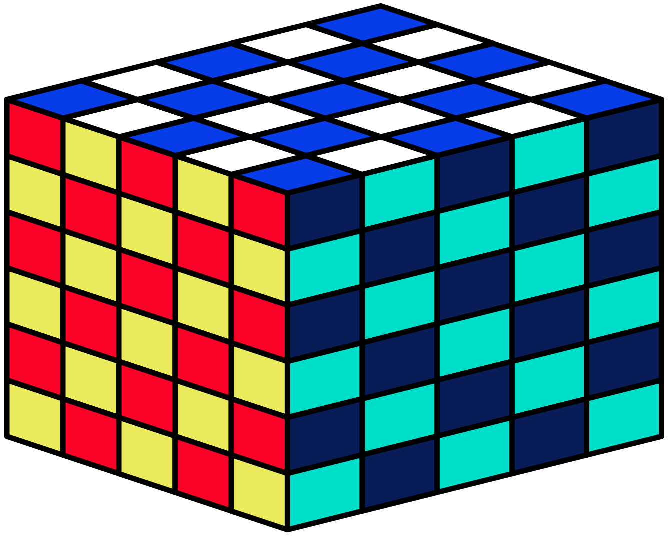
They can also affect **reflectivity** (**k_D** and **k_S**), **normal vectors** ("normal map")…

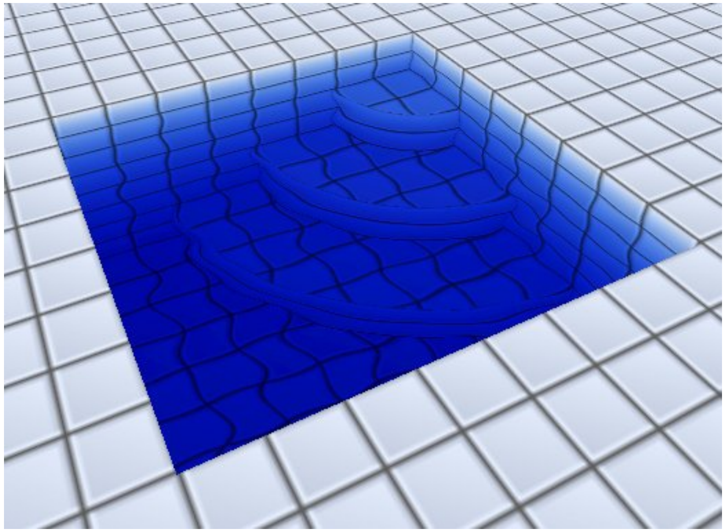
Realistic capture of **material properties** (color pattern, micro- and macro-structure)

- wood, orange/lemon peel, polished metal, plaster…

Replaces complex **geometry** (water waves…)

2D Textures





Covers the **object surface** (like wallpaper or decal)

Texure mapping: $[x, y, z] \rightarrow [u, v]$

Custom texture: $[u, v] \rightarrow$ **color** (normal, material…)

3D Textures

Mimics the **internal structure of materials**

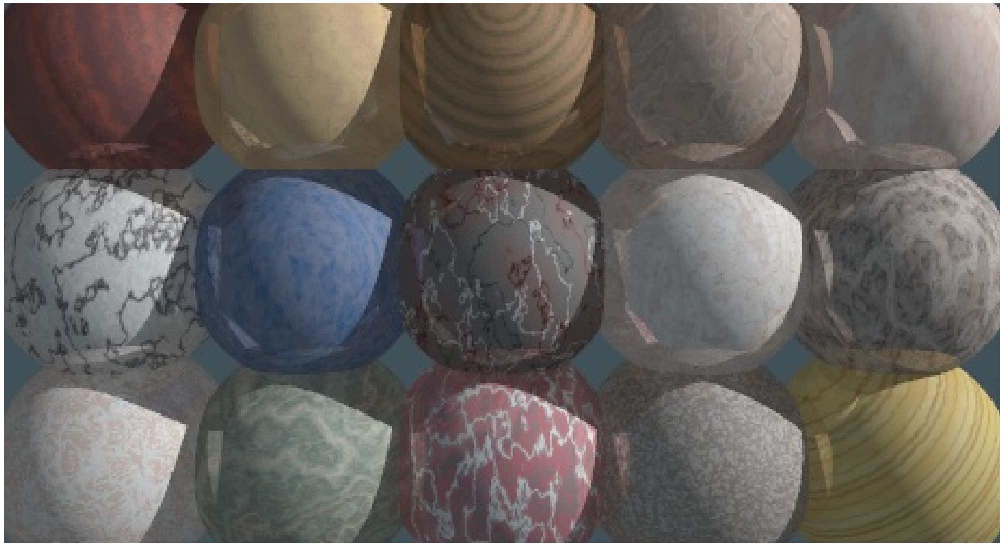
- wood, marble…

Mapping is **not necessary**

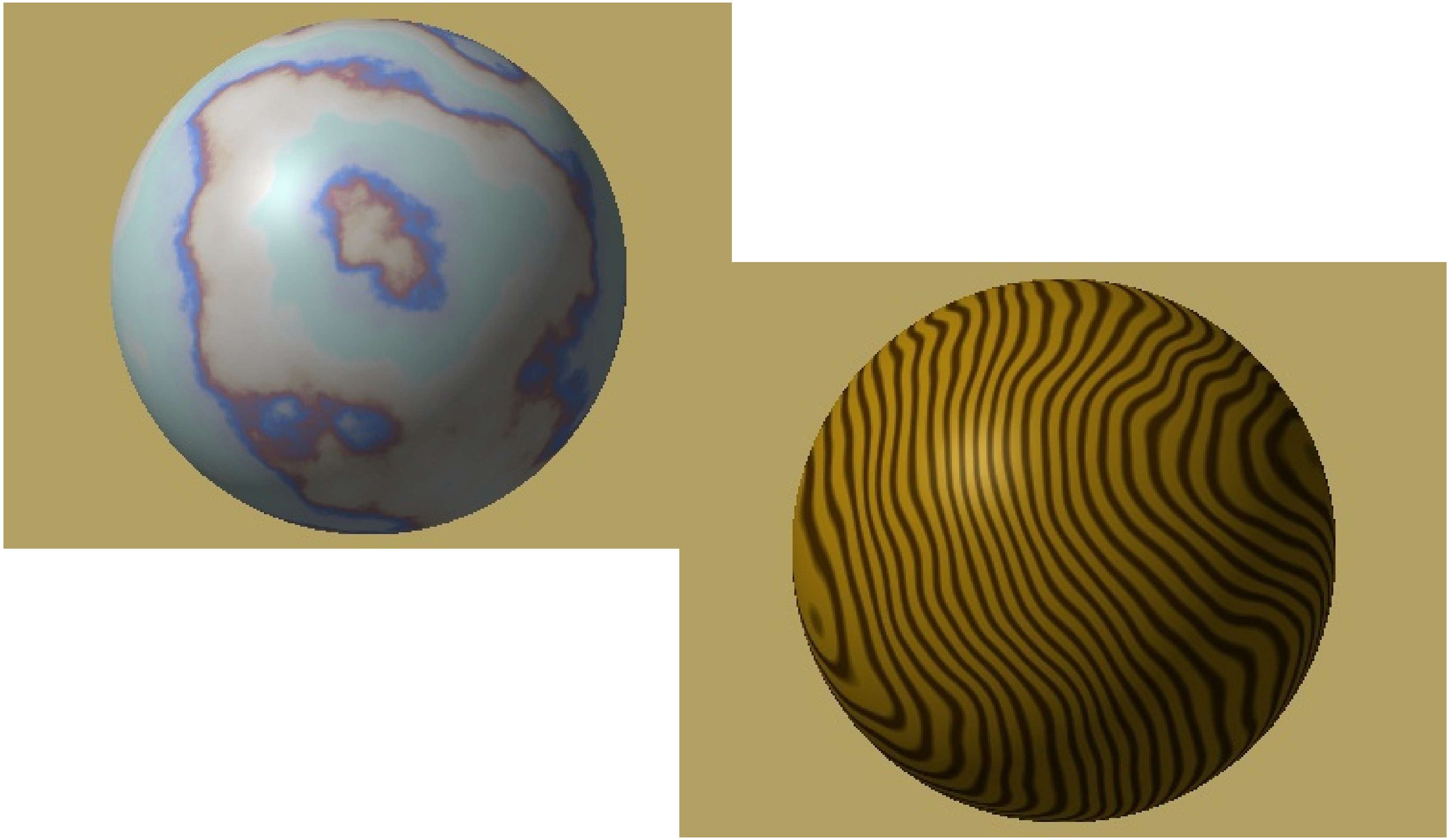
3D texture: [x, y, z] $\rightarrow$ **color** (reflectance…)

3D noise functions are often used

- simulation of natural phenomena … turbulence, wrinkling



More 3D Texture Examples



References

A. Glassner: ***An Introduction to Ray Tracing***, Academic Press, London 1989, 1-31

A. Glassner: ***Principles of Digital Image Synthesis***, Morgan Kaufmann, 1995

Jiří Žára a kol.: ***Počítačová grafika, principy a algoritmy***, 374-378