Computer
Graphics
Charles
University

# Recursive Ray Tracing

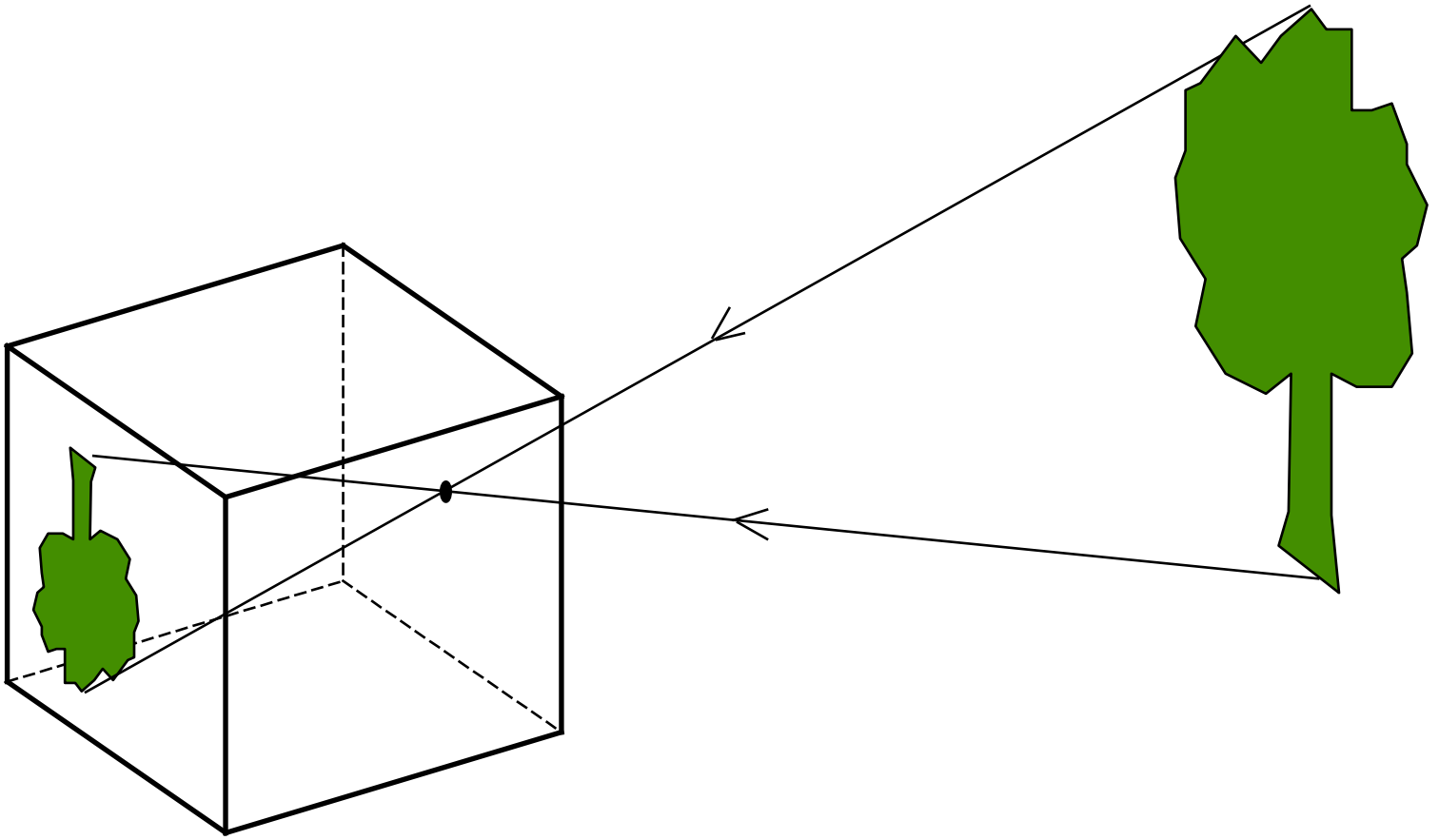


CGG MFF UK Praha

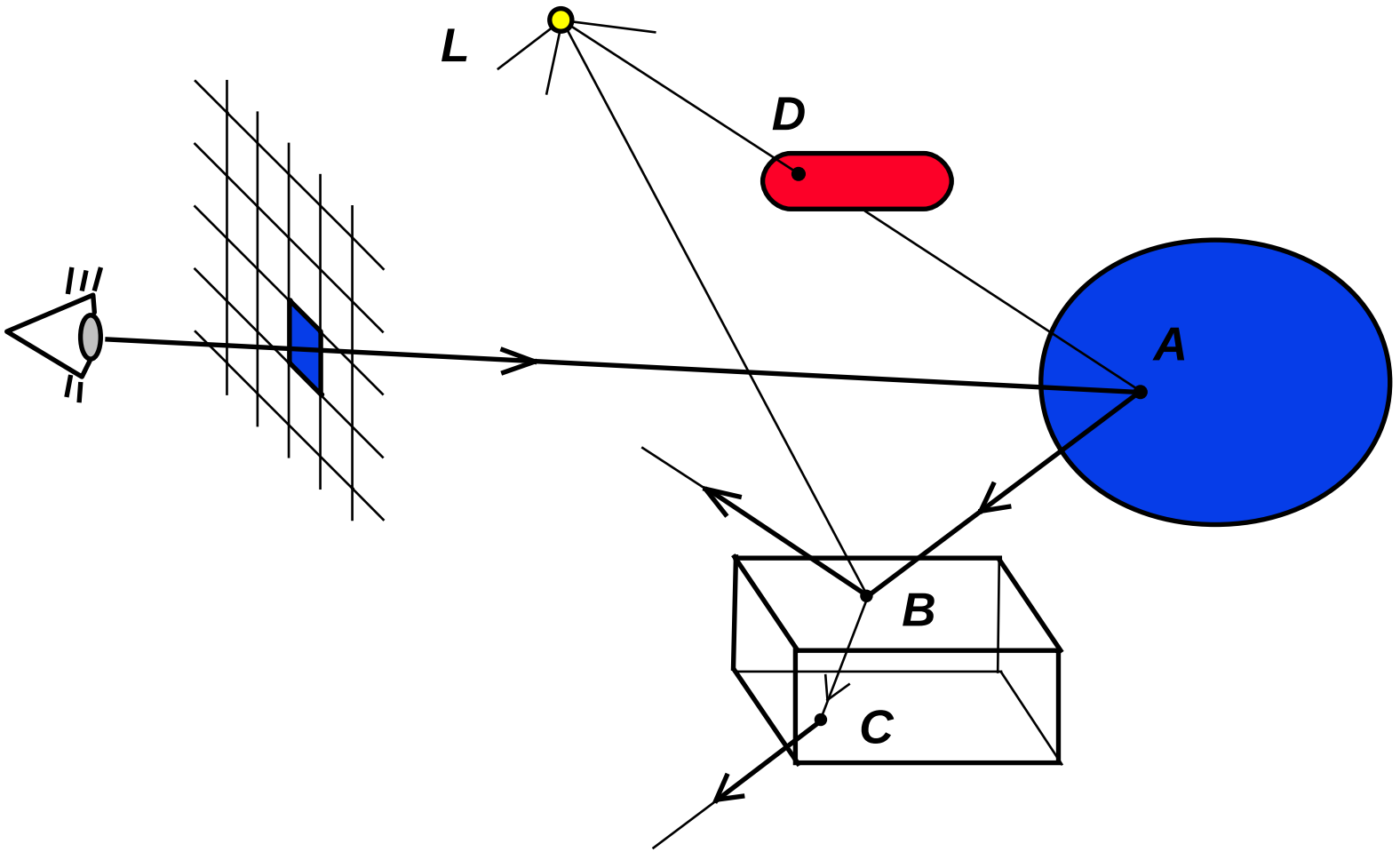
pepca@cgg.mff.cuni.cz

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

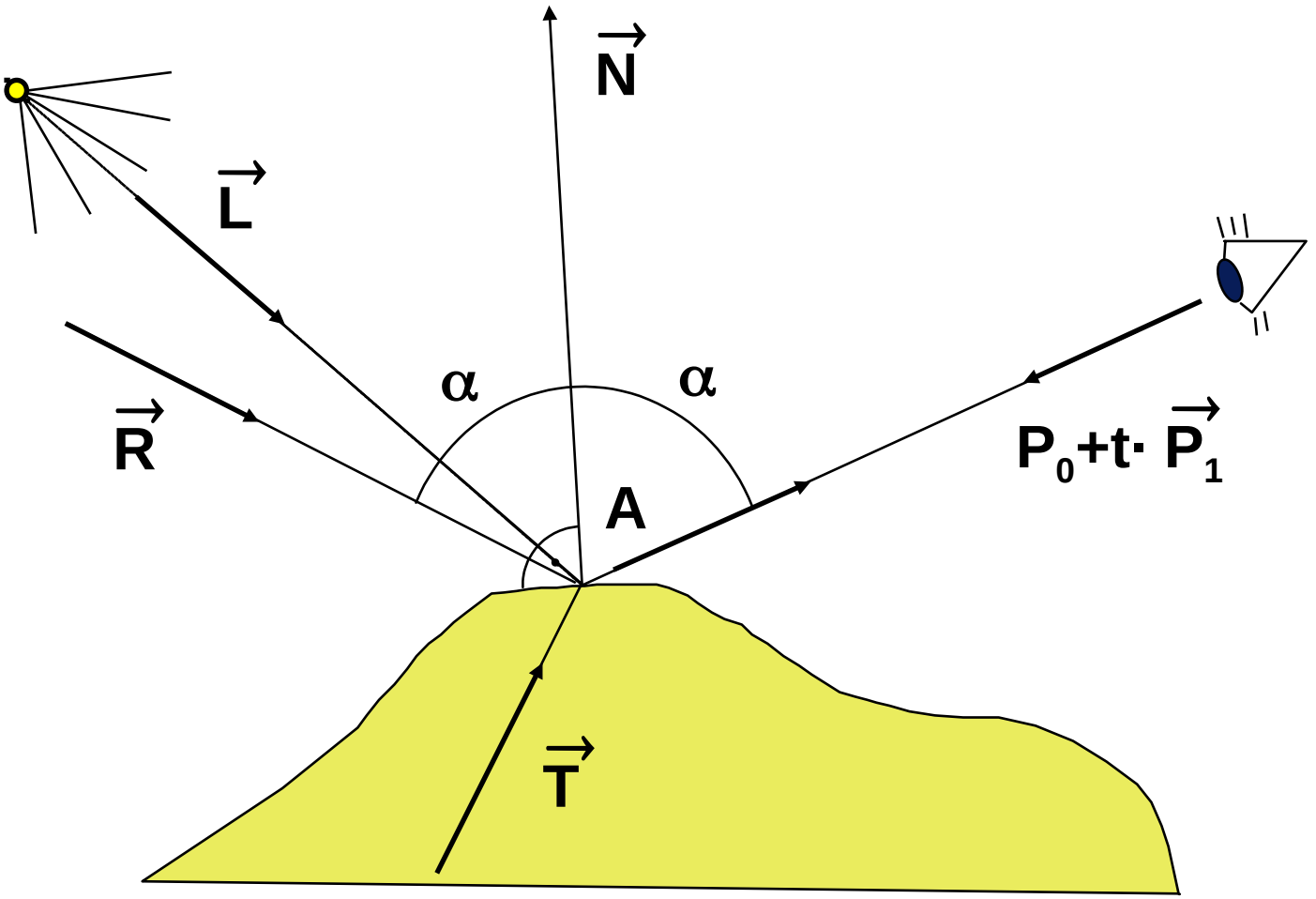
# Pinhole Camera Model



# Backward / Reverse Ray Tracing



# Lighting Computations



# Recursive Implementation

```
function Trace ( P0, P1 : Point3D; depth : integer ) : RGB;
        { P0..ray begining, P1..direction, depth..number of reflections }
var A, R, T : Point3D;              { temporary points and vector }
    B : RGB;                        { resulting color }
begin
  A := Intersection(Scene,P0,P1)    { ray intersection with scene }
  if A==0 then Result:= Background { no intersection }
          else
    begin                           { ray intersects an object }
      B := 0;
      for i := 1 to N do            { contributions from N light sources }
        if Intersection(Scene,A,L[i]-A) == 0 then B := B + Light(A,L[i]);
      depth := depth + 1;
      if depth <= maxdepth then { end of recursion }
        begin
          if  "A has nonzero reflection coef kR"  then
            begin
              "calculate reflected intensity" { reflected ray R}
              B := B + kR * Trace(A,R,depth);
            end;
```

# Recursive Implementation

```
            if "A has nonzero transm. coef kT"  then
              begin
                "calculate refracted intensity" { refracted ray T}
                B := B + kT * Trace(A,T,depth);
              end;
          end;
        Result := B;                  { return value }
      end;
end;
```

# Recursion Management

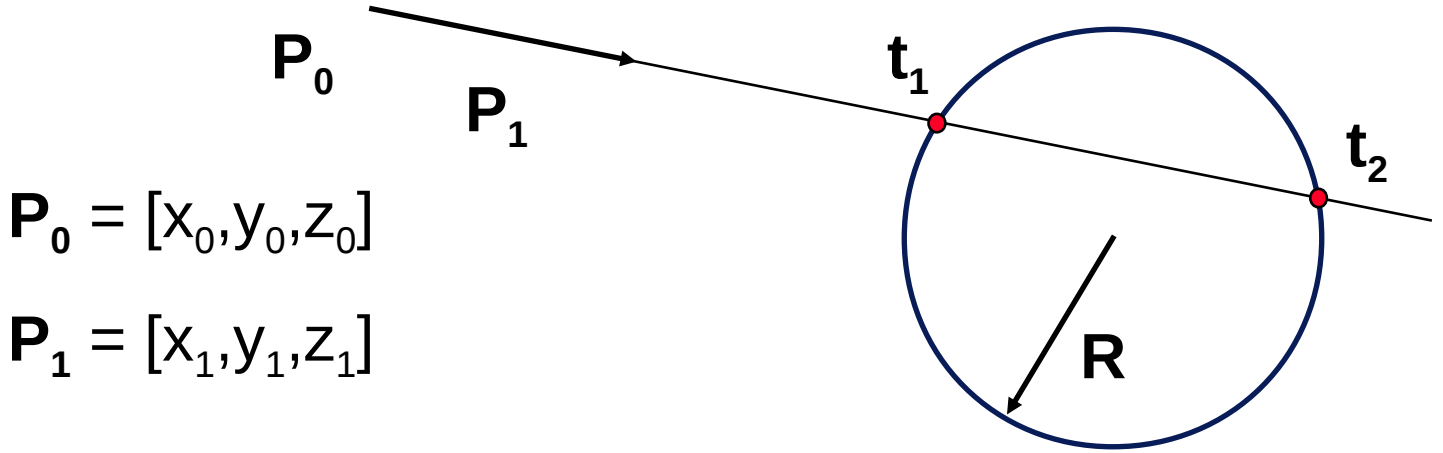
1. **static** – limited by a constant (not suitable for scenes with many mirrors)
2. **dynamic** – according to performance of the ray
   - „**performace**” is the percentage the ray can still contribute to the colour of a given pixel (primary rays: 100%)
   - Limit on the „performance” constant (e.g. 2-10%)
3. **Combined** – limit on the recursion depth, and the „performance” of the ray

# Intersection Computations

1. Intersection coordinates (or „infinity”)
2. ID of the solid (surface) that was hit
3. Normal vector at the intersection

- **Time-consuming operation** (80-90% of render time)
  - Acceleration techniques extremely important
- **Analytical solution** (sphere, cylinder, cube, ..)
- **Numerical solution** (subdivision surfaces, higher order surfaces, rotational surfaces, ..)

# Ray-Sphere Intersections




$\mathbf{P_0} = [x_0, y_0, z_0]$

$\mathbf{P_1} = [x_1, y_1, z_1]$

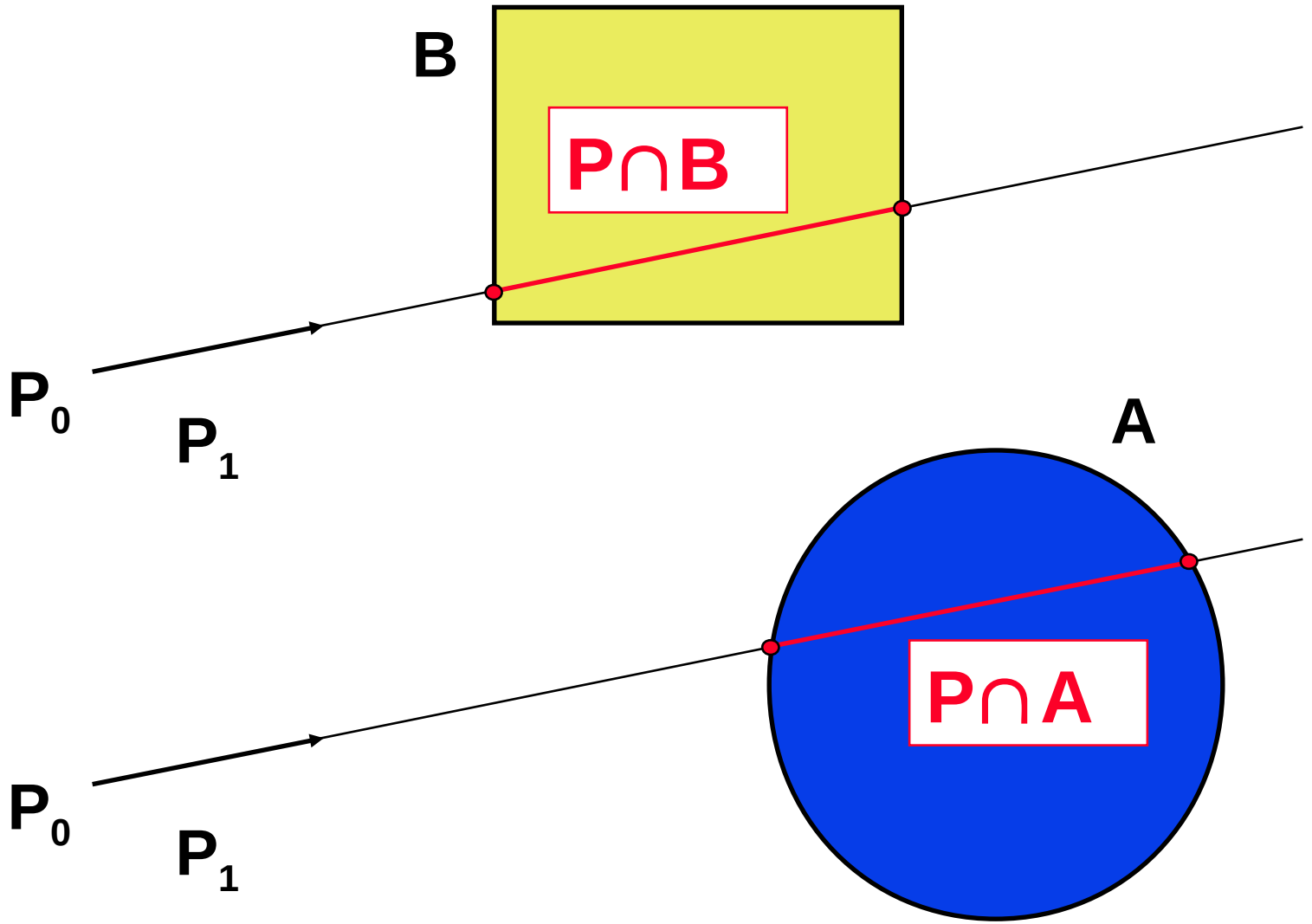
- Ray: $\mathbf{P_0} + t\,\mathbf{P_1}, \quad t > 0$ (1)
- Sphere (at origin): $\mathbf{x^2 + y^2 + z^2 - R^2 = 0}$ (2)
- When substituting (1) into (2) we obtain a **quadratic equation** for ***t***:

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

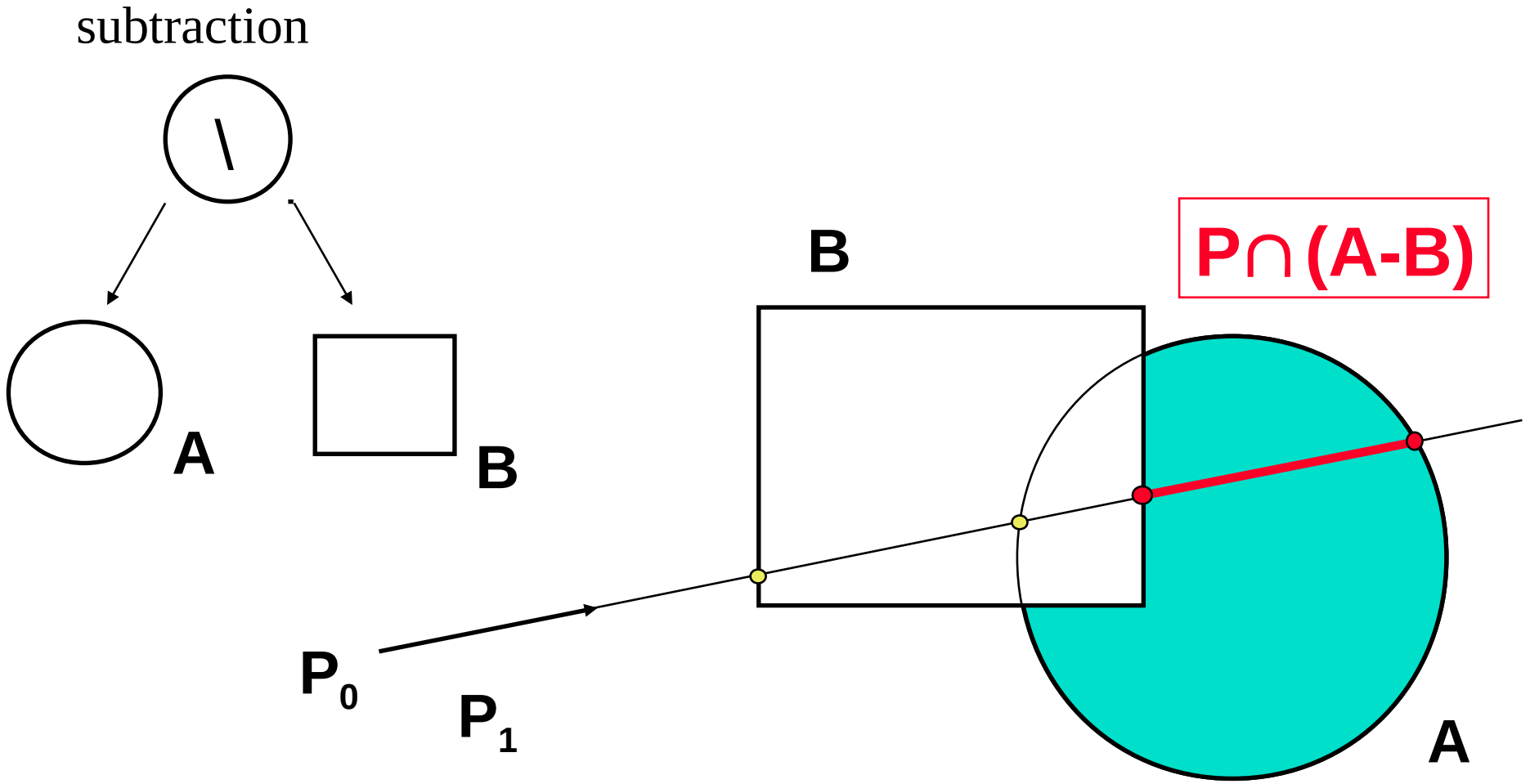
# Ray Intersections with CSG

- For **elementary solids**, intersections can be calculated
  - Start and end of ray traversal through a 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 an interval
- **Geometric transformations**:
  - The inverse transformation is applied to the ray

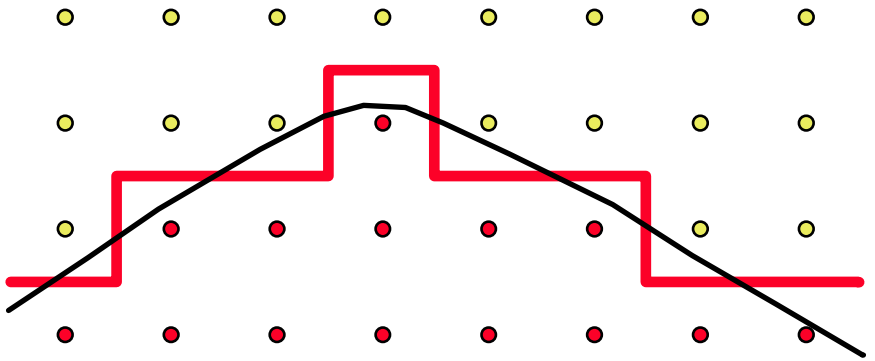
# Intersection P∩A, P∩B



# Intersection: P∩(A-B)

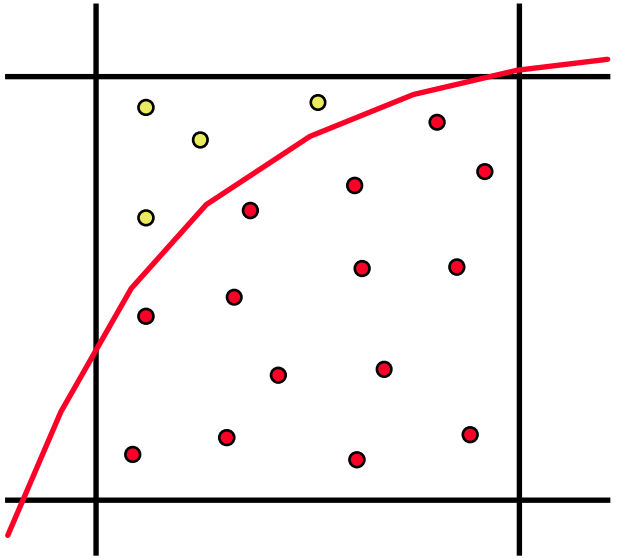


# Anti-aliasing



- Only one ray per pixel leads to „**aliasing**”
  - Jagged edges
  - Interference
- **Increased resolution** only solves the problem partially (and at great expense)

# Multiple Sampling



- Multiple rays are shot into each pixel
- The resulting colour is obtained via arithmetic mean
- Transitions are softer (no jaggies)
- The rays should cover the pixel area evenly, but not regularly!

# End

**Further information:**

- **A. Glassner**: ***An Introduction to Ray Tracing***, Academic Press, London 1989, 1-31
- **Jiří Žára a kol.**: ***Počítačová grafika***, principy a algoritmy, 374-378