Computer Graphics

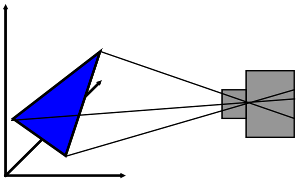
- Ray Tracing I -

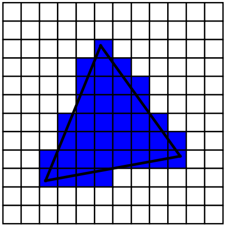
Marcus Magnor
Philipp Slusallek

Overview

- **Last Lecture**
  - Introduction
- **Today**
  - Ray tracing I
    - Background
    - Basic ray tracing
    - What is possible?
    - Recursive ray tracing algorithm
- **Next lecture**
  - Ray tracing II: Spatial indices

Current Graphics: Rasterization



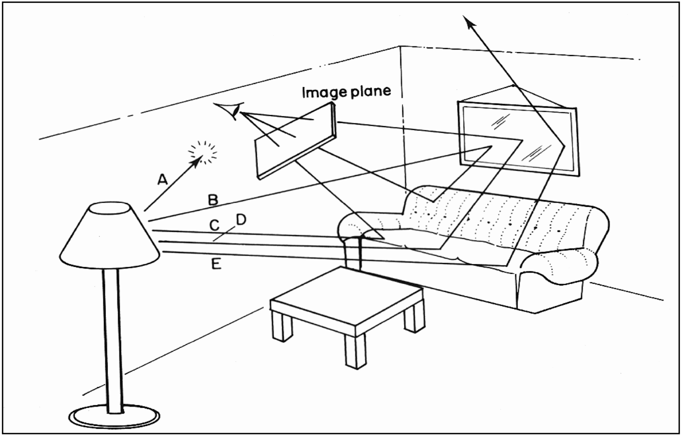


- **Primitive operation of all interactive graphics !!**
  - Scan convert a single triangle at a time
- **Sequentially processes every triangle individually**
  - Can never access more than one triangle

  ➔ But most effects need access to the world: shadows, reflection, global illumination

Tracing the Paths of Light

- **Nature:**
  - Follow the path of ***many*** photons
  - Record those hitting the film in a camera



Ingredients

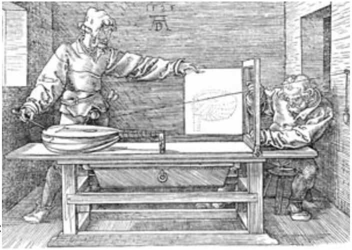
- **Surfaces**
  - 3D geometry of objects in a scene
- **Surface reflectance characteristics**
  - Color, absorption, reflection, refraction, subsurface scattering
  - Local property, may vary over surface
  - Mirror, glass, glossy, diffuse, …
- **Illumination**
  - Position, characteristics of light emitters
  - Repeatedly reflected light → indirect illumination
- **Assumption: air/empty space is totally transparent**
  - Excludes any scattering effects in participating media volumes
  - Would require solving a much more complex problem
  - ➔ Volume rendering, participating media

Light Transport

- **Light Distribution in a Scene**
  - Dynamic equilibrium
  - Newly created, scattered, and absorbed photons
- **Forward Light Transport:**
  - Start at the light sources
  - Shoot photons into scene
  - Reflect at surfaces (according to some reflection model)
  - Wait until they are absorbed or hit the camera (very seldom)
  - ➔ Nature: massive parallel processing at the speed of light
- **Backward Light Transport:**
  - Start at the camera
  - Trace only paths that transport light towards the camera
  - ➔ Ray tracing

Ray Tracing

- **The Ray Tracing Algorithm**
  - One of the two fundamental rendering algorithms
- **Simple and intuitive**
  - Easy to understand and implement
- **Powerful and efficient**
  - Many optical global effects: shadows, ref
  - Efficient real-time implementation in SW and HW
- **Scalability**
  - Can work in parallel and distributed environments
  - Logarithmic scalability with scene size: O(log n) vs. O(n)
  - Output sensitive and demand driven
- **Not new**
  - Light rays: Empedocles (492-432 BC), Renaissance (Dürer, 1525)
  - Uses in lens design, geometric optics, …



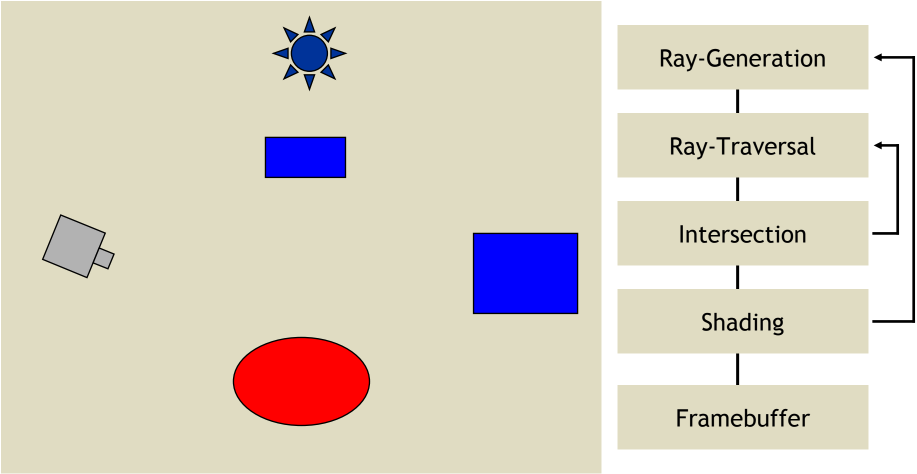
Dürer, 1525

Ray Tracing

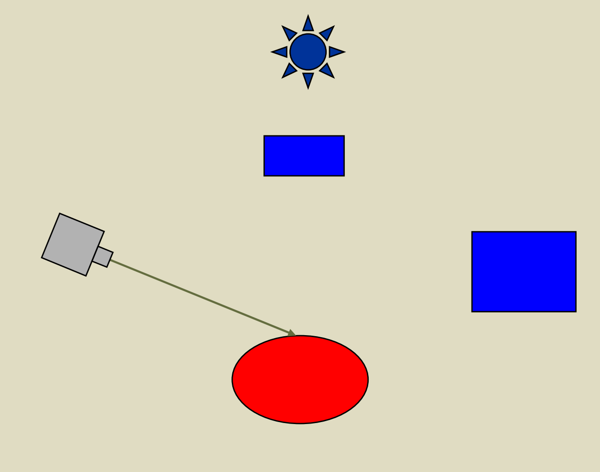
- **Highly Realistic Images**
  - Ray tracing enables *correct* simulation of light transport

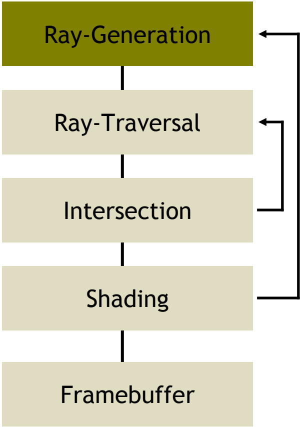
Internet Ray Tracing Competition, June 2002

Ray Tracing Pipeline

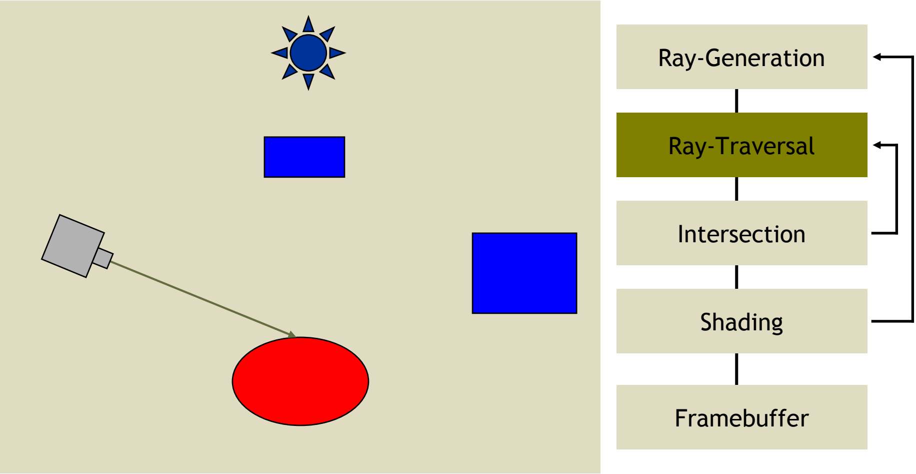


Ray Tracing Pipeline

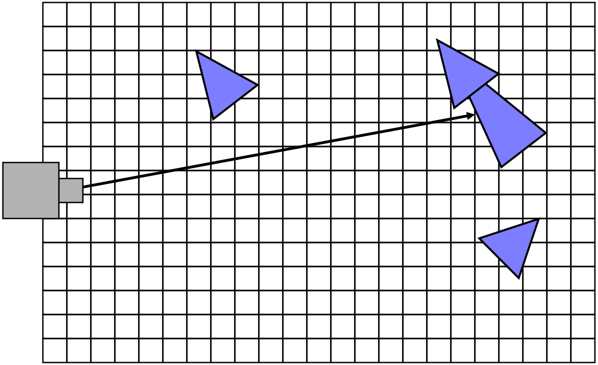


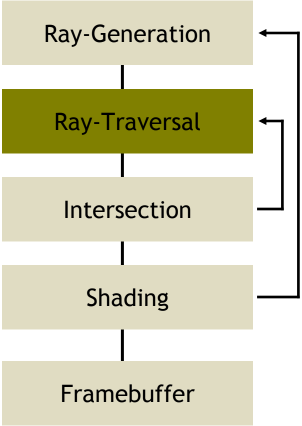


Ray Tracing Pipeline

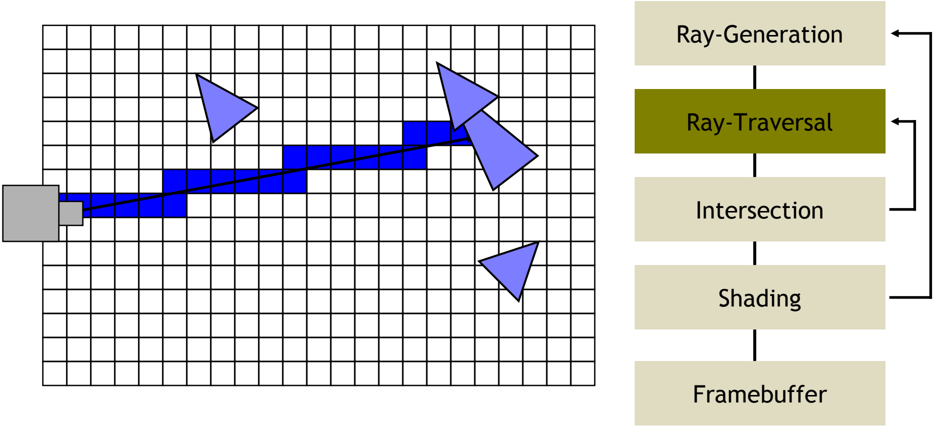


Ray Tracing Pipeline

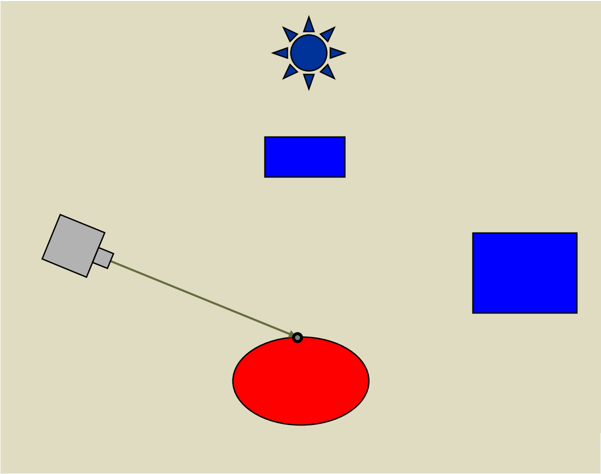


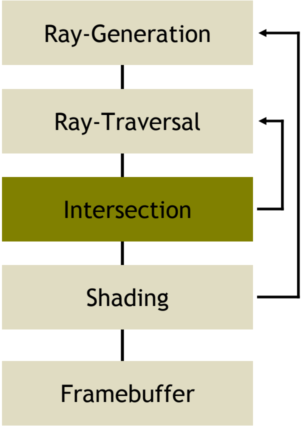


Ray Tracing Pipeline

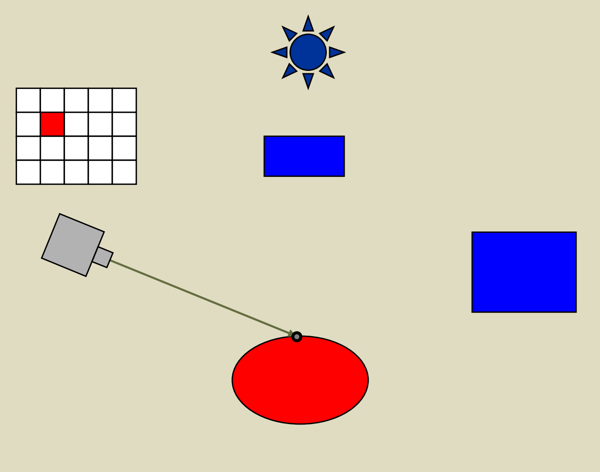


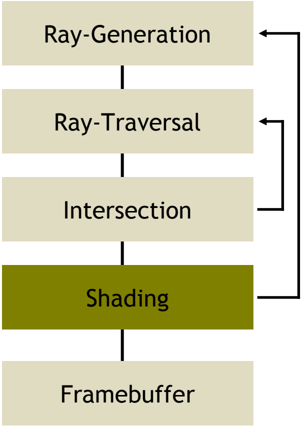
Ray Tracing Pipeline



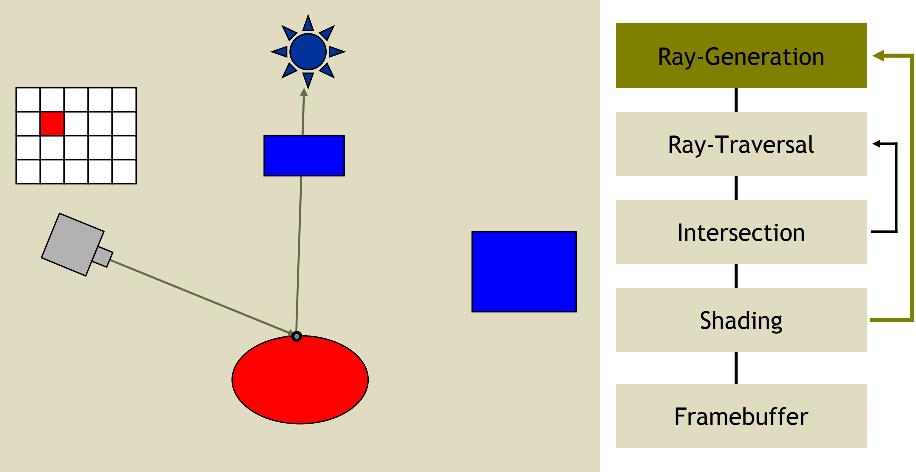


Ray Tracing Pipeline

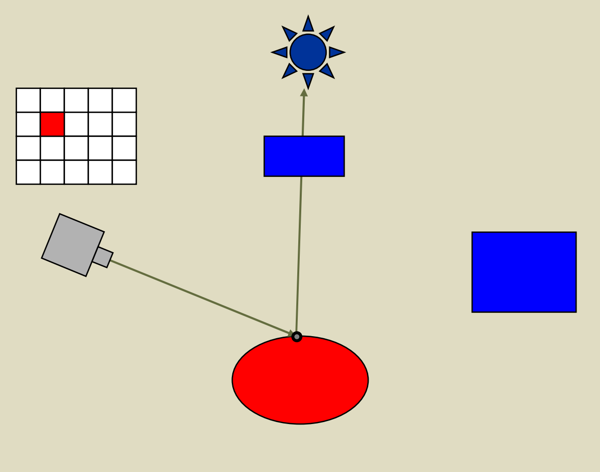


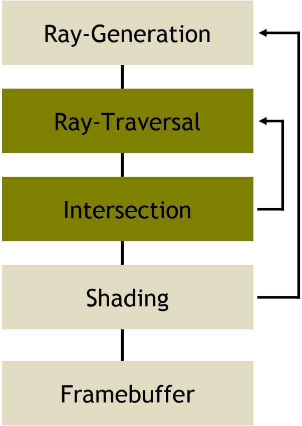


Ray Tracing Pipeline

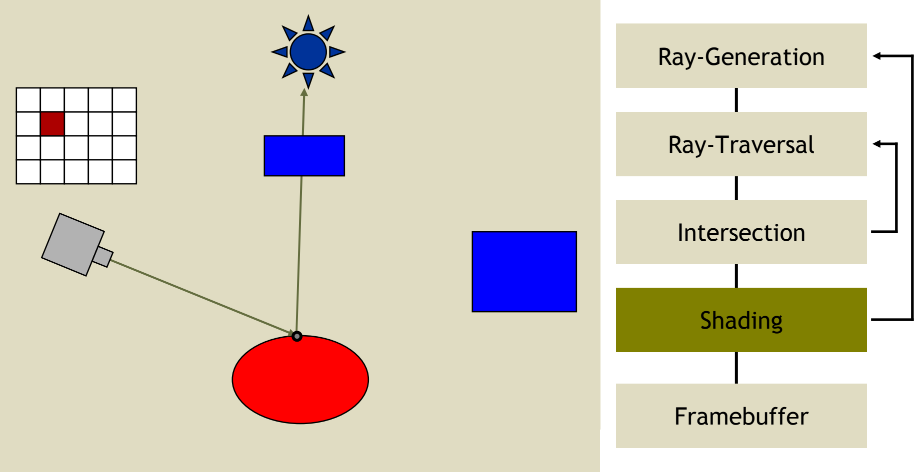


Ray Tracing Pipeline

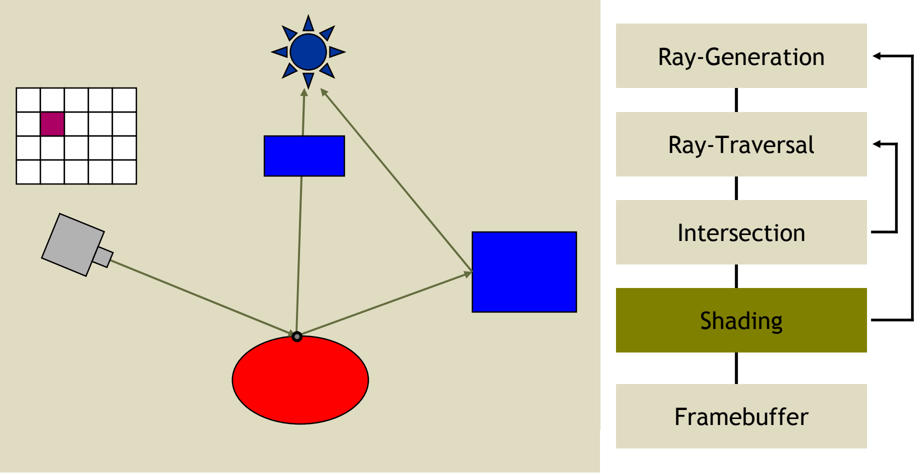




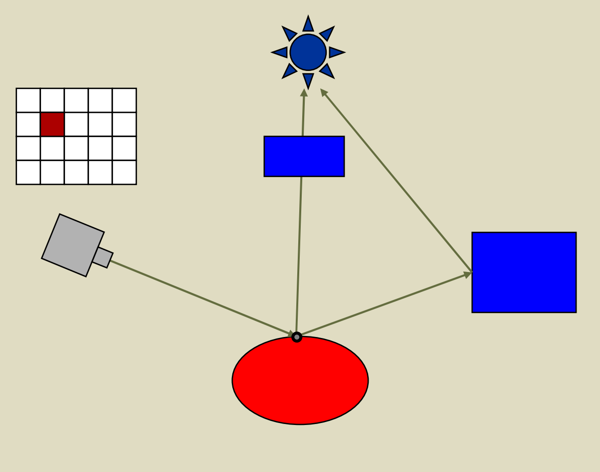
Ray Tracing Pipeline

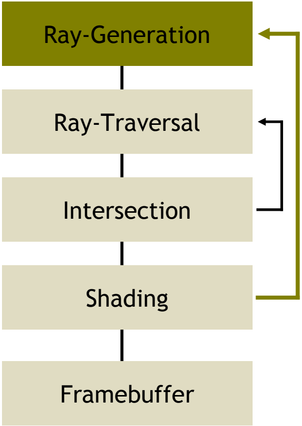


Ray Tracing Pipeline

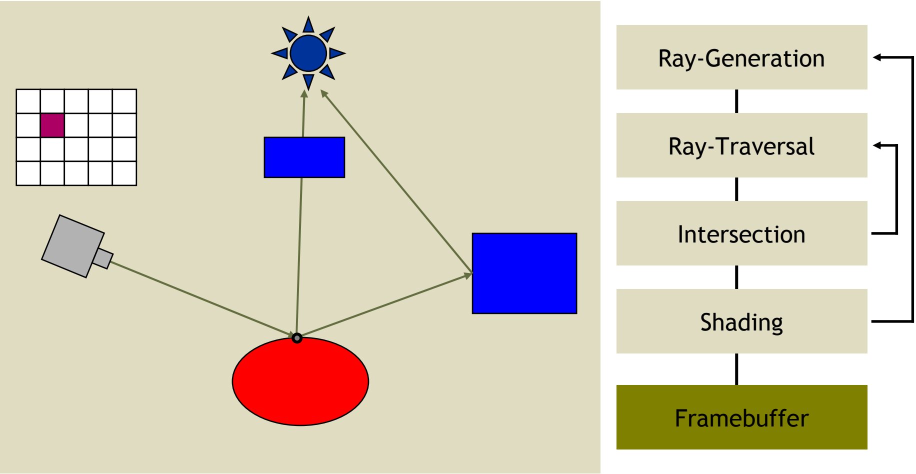


Ray Tracing Pipeline

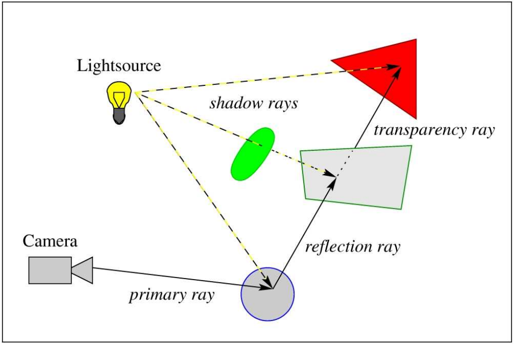




Ray Tracing Pipeline



Ray Tracing



- Global effects
- Parallel (as nature)
- Fully automatic
- Demand driven
- Per pixel operations
- Highly efficient

➔ Fundamental Technology for Next Generation Graphics

Ray Tracing

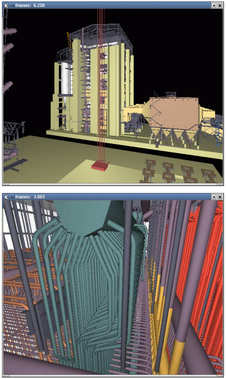
- **In the Past**
  - Only used as an off-line technique
  - Was computationally far too demanding
  - Rendering times of minutes and hours
- **Recently**
  - Interactive ray tracing on supercomputers [Parker, U. Utah‘98]
  - Interactive ray tracing on PCs [Wald‘01]
  - Distributed ray tracing on PC clusters [Wald’01]
- **OpenRT-System (www.openrt.de)**
  - Demo later today

What is Possible?

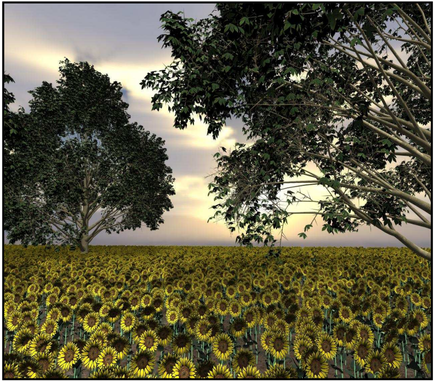
- **Models Physics of Global Light Transport**
  - Dependable, physically-correct visualization

What is Possible?

- **Huge Models**
  - Logarithmic scaling in scene size



12.5 Million Triangles

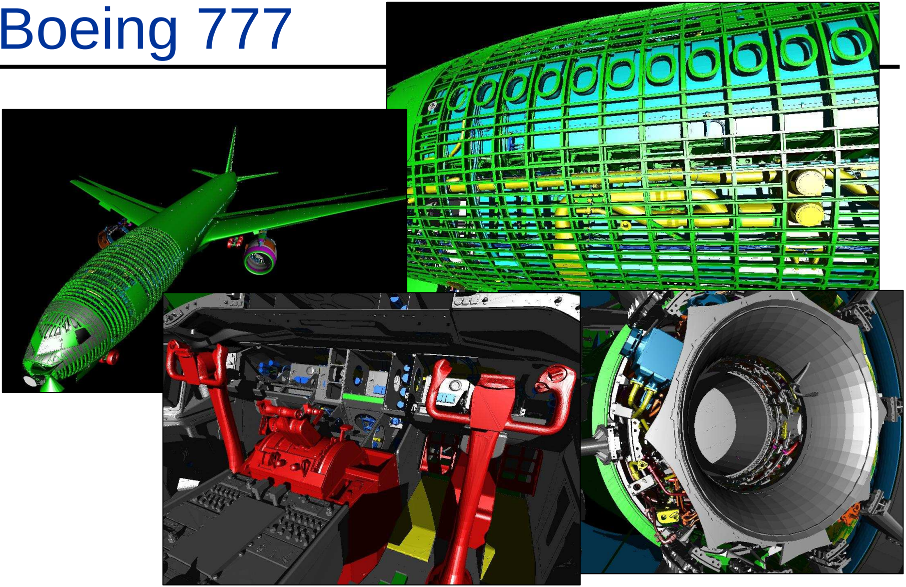


~1 Billion Triangles

Huge & Realistic 3D Models

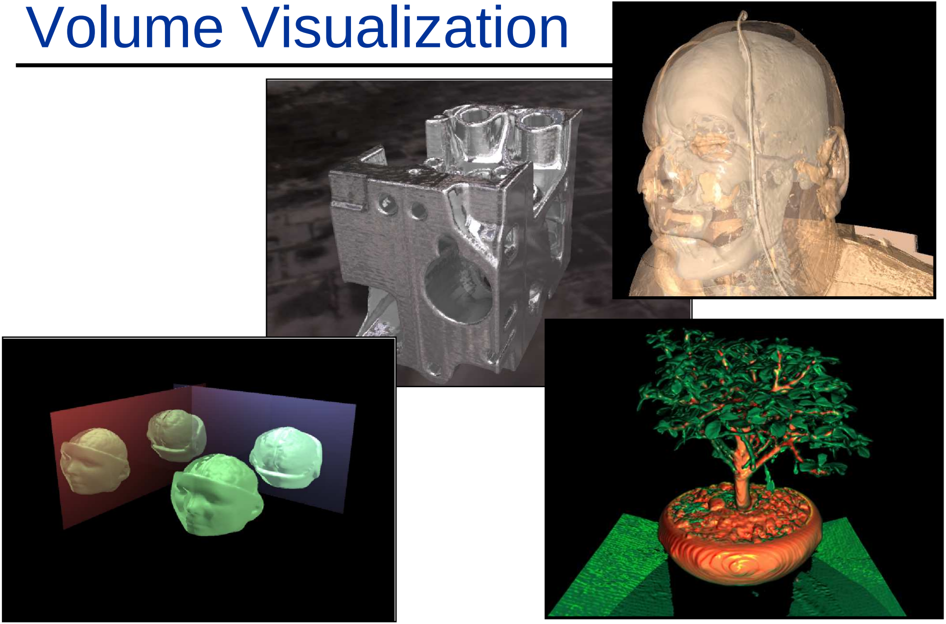
Outdoor environment: ~365,000 plants, ~1.5 billion triangles
Rendered in realtime with skylight illumination on PC cluster

Boeing 777



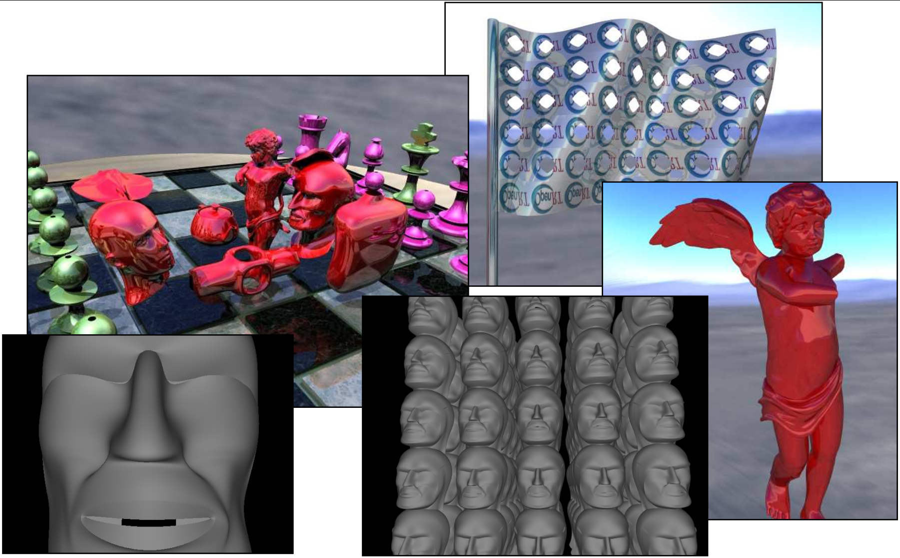
Boeing 777: ~350 million individual polygons, ~30 GB on disk

Volume Visualization



Iso-Surface Volume Rendering

Higher Order Surfaces

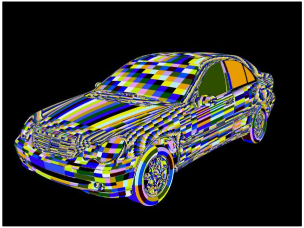


Splines & Subdivision Surfaces: little memory, constant fps

Measured Materials

Data Courtesy R. Klein, Uni Bonn

Realistic Visualization: CAD

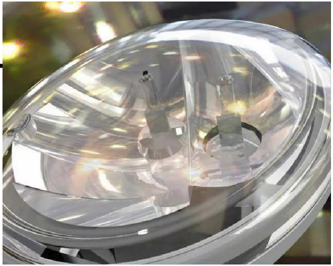


Games?

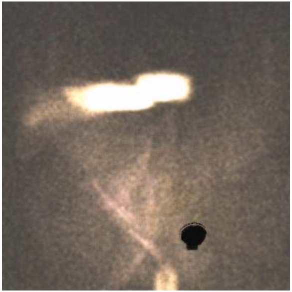
Realtime Lighting Simulation

Lighting Simulation

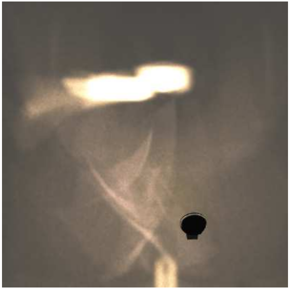
- **Complex Scattering**
- **Highly accurate Results**



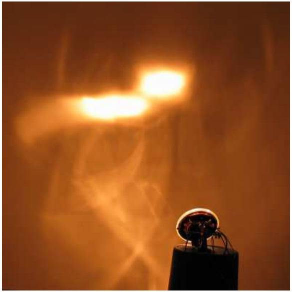
250k / 3 fps



25M / 11 fps



Photograph



Fundamental Ray Tracing Steps

- **Generation of primary rays**
  - Rays from viewpoint along viewing directions into 3D scene
  - (At least) one ray per picture element (pixel)
- **Ray tracing**
  - Traversal of spatial index structures
  - Intersection of ray with scene geometry
- **Shading**
  - From intersection, determine “light color” sent along primary ray
  - Determines “pixel color”
  - Needed
    - Local material color and reflection properties
      - Object texture
    - Local illumination of intersection point
      - Can be hard to determine correctly

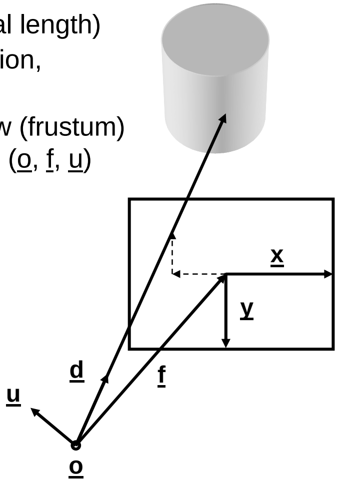
Ray and Object Representations

- **Ray in space: $\underline{\mathbf{r}}(t)=\underline{\mathbf{o}}+t\,\underline{\mathbf{d}}$**
  - $\underline{\mathbf{o}}=(o_{\underline{x}}, o_{\underline{y}}, o_{\underline{z}})$
  - $\underline{\mathbf{d}}=(d_{\underline{x}}, d_{\underline{y}}, d_{\underline{z}})$
- **Scene geometry**
  - Sphere: $(\underline{p}-\underline{c})\cdot(\underline{p}-\underline{c})-r^2=0$
    - $\underline{\mathbf{c}}$: sphere center
    - ***r*** : sphere radius
    - $\underline{\mathbf{p}}$: any surface point
  - Plane: $(\underline{p}-\underline{a})\cdot\underline{n}=0$
    - Implicit definition
    - $\underline{\mathbf{n}}$: surface normal
    - $\underline{\mathbf{a}}$: one given surface point
    - $\underline{\mathbf{p}}$: any surface point
  - Triangles: Plane intersection plus barycentric coordinates

Perspective Camera Model

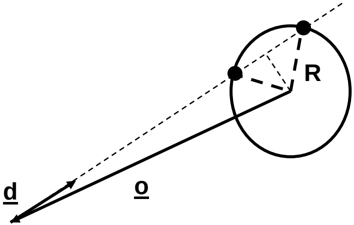
- **Definition of the pinhole camera**
  - o: Origin (point of view)
  - f: Vector to center of view (focal length)
  - u: Up-vector of camera orientation, in one plane with y vector
  - x, y: Span half the viewing window (frustum) relative to coordinate system (o, f, u)
  - xres, yres: Image resolution

```
for (x= 0; x < xres; x++)
  for (y= 0; y < yres; y++)
  {
    d= f + 2(x/xres - 0.5)·x
         + 2(y/yres - 0.5)·y;
    d= d/|d|; // Normalize
    col= trace(o, d);
    write_pixel(x,y,col);
  }
```



Intersection Ray – Sphere

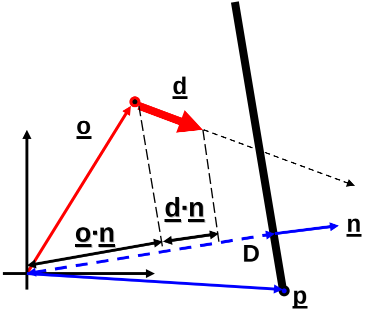
- **Sphere**
  - Given a sphere at the origin
    $x^2 + y^2 + z^2 - 1 = 0$
  - Given a ray
    $\underline{r} = \underline{o} + t\underline{d}$ ($r_x = o_x + td_x$ and so on)
    $\underline{o}$: origin, $\underline{d}$: direction
  - Substituting the ray into the equation for the sphere gives
    $t^2(d_x^2 + d_y^2 + d_z^2) + 2t\,(d_x o_x + d_y o_y + d_z o_z) + (o_x^2 + o_y^2 + o_z^2) - 1 = 0$
    - Easily solvable with standard techniques
    - But beware of numerical imprecision
  - Alternative: Geometric construction
    - Ray and center span a plane
    - Simple 2D construction



Intersection Ray – Plane

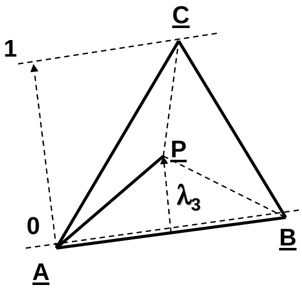
- **Plane: Implicit representation (Hesse form)**
  - Plane equation: $\underline{p} \cdot \underline{n} - D = 0$, $|\underline{n}| = 1$
    - $\underline{n}$: Normal vector:
    - D: Normal distance of plane from (0, 0, 0):
- **Two possible approaches**
  - Geometric
  - Mathematic
    - Substitute $\underline{o} + t\underline{d}$ for p
    - $(\underline{o} + t\underline{d}) \cdot \underline{n} - D = 0$
    - Solving for t gives

$$t = \frac{D - \underline{o} \cdot \underline{n}}{\underline{d} \cdot \underline{n}}$$



Intersection Ray – Triangle

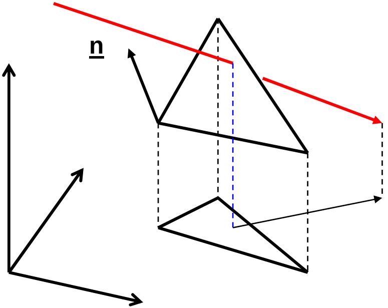
- **Barycentric coordinates**
  - Non-degenerate triangle ABC
  - Every point P in the plane can be described using $\underline{P} = \lambda_1 \underline{A} + \lambda_2 \underline{B} + \lambda_3 \underline{C}$
  - $\lambda_1 + \lambda_2 + \lambda_3 = 1$
    - Interpretation of barycentric coordinates $\lambda_3 = \angle(APB) / \angle(ACB)$ etc
  - For fixed λ_3, $\underline{P}$ may move parallel to AB
  - For $\lambda_1 + \lambda_2 = 1$ $\underline{P} = (1-\lambda_3)(\lambda_1 \underline{A} + \lambda_2 \underline{B}) + \lambda_3 \underline{C}$ $(0 < \lambda_3 < 1)$
    - $\underline{P}$ moves between $\underline{C}$ and AB



- **Point is in triangle, iff all λ_i greater or equal than zero**

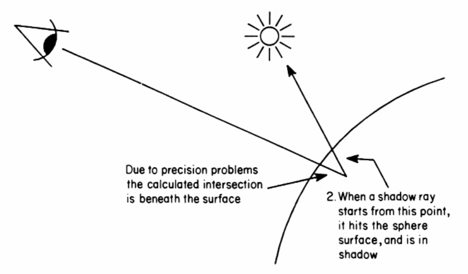
Intersection Ray – Triangle (2)

- **Compute intersection with triangle plane**
- **Given the 3D intersection point**
  - Project point into xy, xz, yz coordinate plane
  - Use coordinate plane that is most aligned
    - xy: if n_z is maximal, etc.
  - Coordinate plane and 2D vertices can be pre-computed
- **Compute barycentric coordinates**
- **Test for positive BCs**



Precision Problems

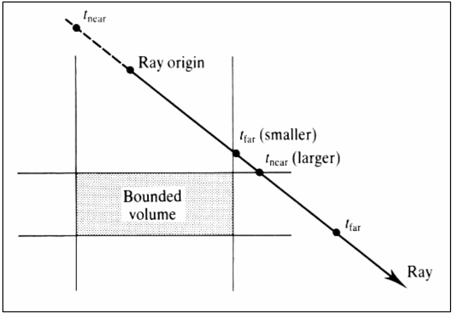
Inaccuracies of the intersection points computations due to floating-point arithmetic can result in incorrect shadow rays (self-shadowing) or infinite loops for secondary rays which have origins at a previously found intersection point. A simple solution is to check if the value of parameter *t* (used for intersection point calculations) is within some tolerance. For example, if *abs(t)* < *0.00001*, then that *t* describes the origin of some ray as being on the object. The tolerance should be scaled to the size of the environment.

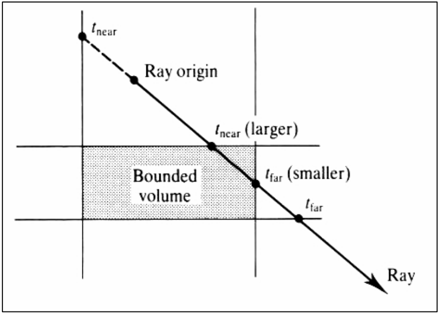


Problem in surface intersection.

Intersection Ray- Box

Ray/box intersections are important because boxes are used as bounding volumes, especially in hierarchical schemes. To check if a ray intersects a box, we treat each pair of parallel planes in turn, calculating the distance along the ray to the first plane t_{near} and the second plane t_{far}. If the value of t_{near} for one pair of planes is greater than t_{far} for another pair of planes, the ray cannot intersect the box.





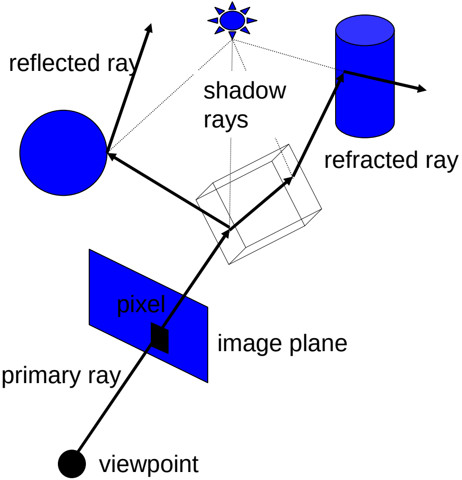
History of Intersection Algorithms

- **Ray-geometry intersection algorithms**
  - Polygons: [Appel '68]
  - Quadrics, CSG: [Goldstein & Nagel '71]
  - Recursive Ray Tracing: [Whitted '79]
  - Tori: [Roth '82]
  - Bicubic patches: [Whitted '80, Kajiya '82]
  - Algebraic surfaces: [Hanrahan '82]
  - Swept surfaces: [Kajiya '83, van Wijk '84]
  - Fractals: [Kajiya '83]
  - Deformations: [Barr '86]
  - NURBS: [Stürzlinger '98]
  - Subdivision surfaces: [Kobbelt et al '98]

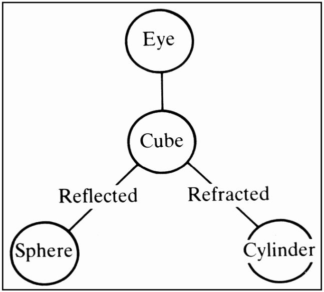
Shading

- **Intersection point determines primary ray's "color"**
- **Diffuse object: color at intersection point**
  - No variation with viewing angle: diffuse (Lambertian)
  - Must still be illuminated
    - Point light source: shadow ray
    - Scales linearly with received light (Irradiance)
    - No illumination: in shadow = black
- **Non-Lambertian Reflectance**
  - Appearance depends on illumination *and* viewing direction
    - Local Bi-directional Reflectance Distribution Function (BRDF)
  - Simple cases
    - Mirror, glass: secondary rays
- **Area light sources, indirect illumination can be difficult**

Recursive Ray Tracing



- **Searching recursively for paths to light sources**
  - Interaction of light & material at intersection points
  - Recursively trace new rays in reflection, refraction, and light direction



Ray Tracing Algorithm

- **Trace(ray)**
  - Search the next intersection point ➔ (hit, material)
  - Return Shade(ray, hit, material)
- **Shade(ray, hit, material)**
  - For each light source
    - if ShadowTrace(ray to light source, distance to light)
      - Calculate reflected radiance (i.e. Phong)
      - Adding to the reflected radiance
  - If mirroring material
    - Calculate radiance in reflected direction: Trace(R(ray, hit))
    - Adding mirroring part to the reflected radiance
  - Same for transmission
  - Return reflected radiance
- **ShadowTrace(ray, dist)**
  - Return false, if intersection point with distance < dist has been found

Ray Tracing

- **Incorporates into a single framework**
  - Hidden surface removal
    - Front to back traversal
    - Early termination once first hit point is found
  - Shadow computation
    - Shadow rays/ shadow feelers are traced between a point on a surface and a light sources
  - Exact simulation of some light paths
    - Reflection (reflected rays at a mirror surface)
    - Refraction (refracted rays at a transparent surface, Snell's law)
- **Limitations**
  - Easily gets inefficient for full global illumination computations
    - Many reflections (exponential increase in number of rays)
    - Indirect illumination requires many rays to sample all incoming directions

Ray Tracing: Approximations

- **Usually RGB color model instead of full spectrum**
- **Finite number of point lights instead of full indirect light**
- **Approximate material reflectance properties**
  - Ambient: constant, non-directional background light
  - Diffuse: light reflected uniformly in all directions,
  - Specular: perfect reflection, refraction
- **All are based on purely empirical foundation**

Wrap-Up

- **Background**
  - Forward light transport vs. backward search in RT
- **Ray tracer**
  - Ray generation, ray-object intersection, shading
- **Ray-geometry intersection calculation**
  - Sphere, plane, triangle, box
- **Recursive ray tracing algorithm**
  - Primary, secondary, shadow rays

- **Next lecture**
  - Advanced acceleration techniques