VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Computer Graphics I

460-4078

Fall 2023

Last update 14. 12. 2023

Computer Graphics I

- Lecturer
  - Tomas Fabian
  - Office

    room EA408, building of FEECS
  - Office hours

    Tuesday 13:00 – 15:00 (all other office hours are by appointment)
  - Email

    tomas.fabian@vsb.cz
  - Web site with additional materials

    http://mrl.cs.vsb.cz/people/fabian/pg1_course.html

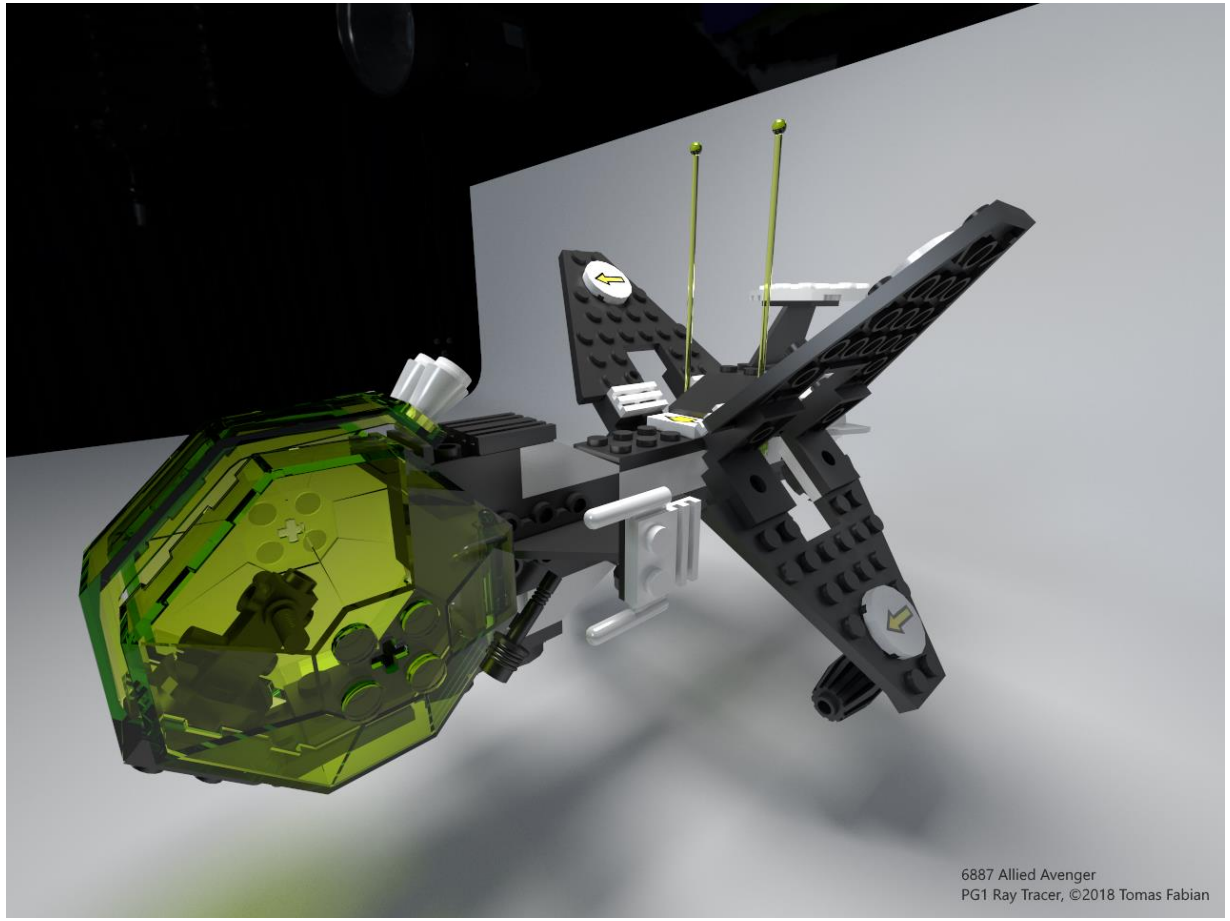
Course Targets and Goals

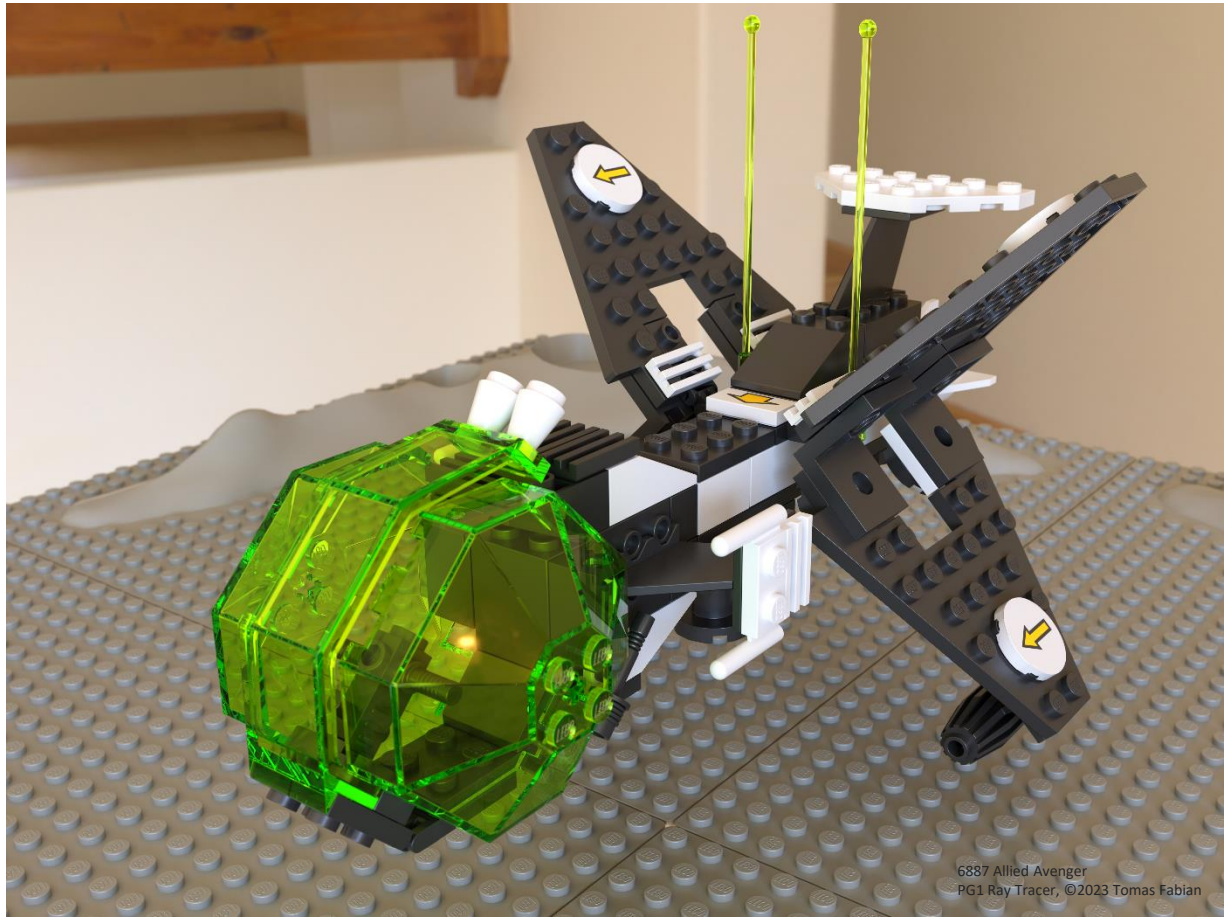
- Introduce basic techniques of photorealistic image synthesis using ray tracing techniques.

- You will have hands-on experience with implementation of the here described methods and algorithms for creating realistic images.

- Mastering selected libraries for (near) real-time ray tracing.

(Perhaps) Motivation





(Perhaps) Motivation

Course Prerequisites

- Basics of programming (C++)
- Previous courses:
  - Fundamentals of Computer Graphics (ZPG)
- To be familiar with basic concepts of mathematical analysis, linear algebra, vector calculus, and statistics

Main Topics

- Physical and mathematical basics of image synthesis (radiometric and photometric quantities, transformations, coordinate)
- Camera model
- Ray tracing method, calculation of ray intersections with geometrical objects
- Basic types of materials, models of light reflection, textures
- Microsurface models (Cook-Torrance, Oren-Nayar), general BRDF
- Sampling and anti-aliasing
- Acceleration methods, acceleration data structures and parallelization
- Rendering equation (Kajiya) and its solution using Monte Carlo methods
- Path tracing, variance reduction techniques (importance sampling, russian roulette, next event estimation, direct lighting)
- Light sources (sampling, image based lighting)
- Bi-directional path tracing, photon mapping
- Spectral tracing, tone mapping
- Other methods of photorealistic rendering of scenes
- Other methods of modeling and displaying solids (boundary models, CSG, distance function)

Organization of Semester and Grading

- Each lecture will discuss one main topic
- Given topic will be practically realized during the following exercises
- The individual tasks from the exercise will be scored (during the last week of the semester)
- You can earn up to 45 points in total
- Final combined (written and oral) test covering topics from the previous slide with subsequent discussion
- You can earn up to 55 points from final exam

Chat GPT/Copilot Policy

- Feel free to use them to prepare your project
- You are good enough to pass the class if you are good enough to verify their outputs. In other words, you need to be able to justify your code
- Beware, their outputs are sometimes completely wrong

Study Materials

- [1] Sojka, E.: *Počítačová grafika II: metody a nástroje pro zobrazování 3D scén*, VŠB-TU Ostrava, 2003, ISBN 80-248-0293-7. (online)
- [2] Sojka, E., Němec, M., Fabián, T.: *Matematické základy počítačové grafiky*, VŠB-TU Ostrava, 2011. (online)
- **[3] Pharr, M., Jakob, W., Humphreys, G.: *Physically Based Rendering*, Fourth Edition: From Theory to Implementation, MIT Press, 2023, 1312 pages, ISBN 978-0262048026. (online)**
- [4] Dutre, P., Bala,K., Bekaert, P. *Advanced global illumination*. AK Peters/CRC Press, 2006.
- [5] Haines, E., Akenine-Möller, T. (ed.): *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*. Apress, 2023, 607 pages, ISBN 978-1484244265. (online)
- [6] Marrs, A., Shirley, P., Wald, I (ed.). *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*. Springer Nature, 2023, 858 pages, ISBN 978-1484271841. (online)
- [7] Shirley, P., Morley, R. K.: *Realistic Ray Tracing*, Second Edition, AK Peters, 2003, 235 pages, ISBN 978-1568814612.
- [8] Akenine-Moller, T., Haines, E., Hoffman, N.: *Real-Time Rendering*, Fourth Edition, AK Peters/CRC Press, 2018, 1178 pages, ISBN 978-1138627000.
- [9] Dutré, P.: *Global Illumination Compendium*, 2003, 68 pages. (online)
- [10] Ryer, A. D.: *The Light Measurement Handbook*, 1997, 64 pages. (online)

Study Materials

- Other free and downloadable materials are at https://www.realtimerendering.com

These are books that are **FREE ONLINE**, ordered by publication date. Do not be fooled by the price; all but one were published as physical books and each has valuable information.


Download for free **Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX**, edited by Adam Marrs, Peter Shirley, and Ingo Wald, Apress, August 4, 2021 (Book's website).


Download for free **Developing Graphics Frameworks with Python and OpenGL**, by Lee Stemkoski and Michael Pascale, CRC Press, July 7, 2021 (Publisher's website).


Download for free **Learn OpenGL - Graphics Programming: Learn modern OpenGL graphics programming in a step-by-step fashion**, by Joey de Vries, Kendall & Welling, June 2020 (Book's website, with fr


Download for free **Ray Tracing Gems**, edited by Eric Haines and Tomas Akenine-Möller, Apress, March 2019 (Book's website, publisher's page, Amazon), ***download for free***.


Read for free **Physically Based Rendering, Third Edition: from Theory to Implementation**, by Matt Pharr, Wenzel Jakob, and Greg Humphreys, Morgan Kaufmann, November 2016 (more information), Pat


Download for free **Ray Tracing in One Weekend**, by Peter Shirley, January 2016 (Code, tweet, blog), ***download for free***, ***read (corrected version) for free***.

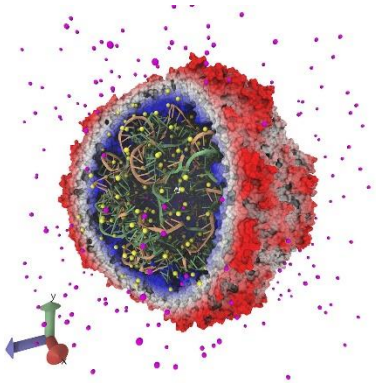

Download for free **Ray Tracing: the Next Week**, by Peter Shirley, March 2016 (Code, tweet, blog), ***download for free***, ***read (corrected version) for free***.

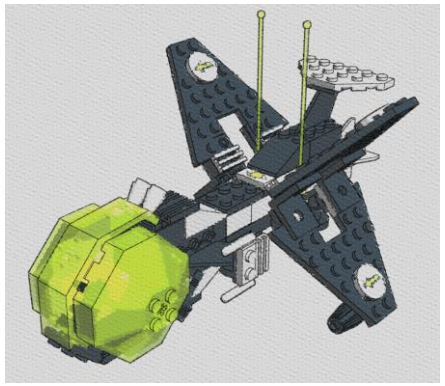

Download for free **Ray Tracing: The Rest Of Your Life**, by Peter Shirley, March 2016 (Code, tweet, blog), ***download for free***, ***read (corrected version) for free***.

- A large list of graphics books is also at https://www.realtimerendering.com/books.html

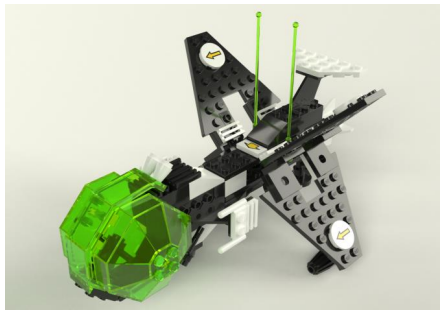
Types of Graphics

- *Non-photorealistic graphics/rendering*
  - Artistic styles
  - Scientific and engineering visualization
  - Data Visualization course





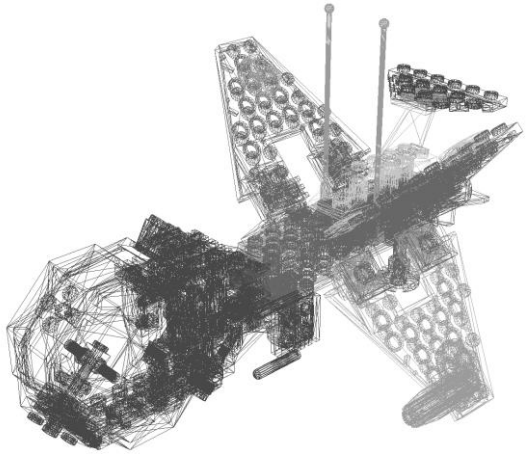
- *Photorealistic graphics/rendering*
  - Simulate the image formation process as precisely as possible
  - Physically plausible light transport through the scene
  - Topic of this course

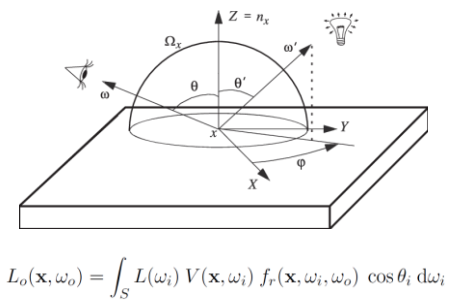


Photorealistic Image Synthesis

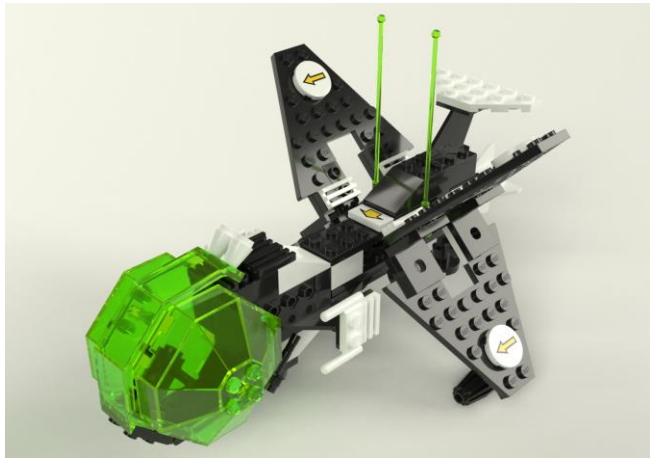
- You will be asked to create an realistically looking image based on a mathematical representation of a real or an artificial world

Real world or your imagination → Mathematical representation of the scene (light sources, geometry, materials, camers) → Mathematical description of light behaviour → Artificial image





$$L_o(\mathbf{x}, \omega_o) = \int_S L(\omega_i)\, V(\mathbf{x}, \omega_i)\, f_r(\mathbf{x}, \omega_i, \omega_o)\, \cos\theta_i\, \mathrm{d}\omega_i$$



Application Areas

- Film industry – special effects or entire scenes/ films
- High quality rendering for commercials, prints, etc. (CG product images)
- Video game industry – ray tracing has recently entered this area (earlier e.g. prebaked lights)
- Architecture and design, virtual prototyping
- VR and AR (remote assistance and collaboration, conferencing)
- Various kind of simulations (lighting, sound propagation, collision detection, creating artificial datasets, etc.)

Knowledge Base

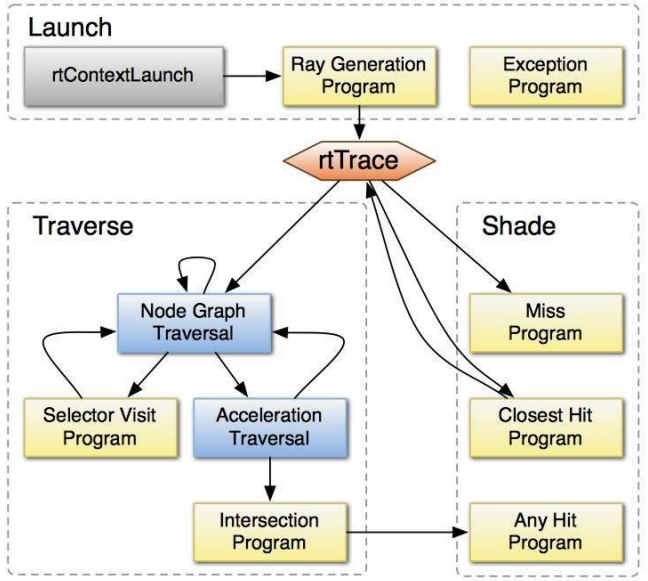
- Physics
  - Radiometry and photometry
  - Models of light interaction with various materials
  - Theory of light transport (mainly laws of geometrical optics)
- Mathematics
  - Integral and equations
  - Monte Carlo methods
- Informatics
  - Software engineering
  - Programming
- Visual perception and Art

Writing a Fast Ray Tracer is Difficult

- Multithreading
  - Rendering is an embarrassingly parallel workload/problem
  - Parallelization of hierarchy structures is not trivial
- Vectorization
  - Effective utilization of SIMD units (e.g. SSE, AVX)
- Domain knowledge
  - Many different data structures (kd-trees, BVHs, octrees) and algorithms (Metropolis-Hastings, Monte Carlo, variance reduction methods, light transport, laws of geometrical optics), …

Ray Tracing Kernel Libraries

- Ray tracing consist of relatively small number of commonly used operations (mainly build and traversal)
- Available (for free) ray tracing kernels
  - Nvidia OptiX (high performance ray tracing on the Nvidia GPUs)
  - AMD Radeon Rays (any OpenCL 1.2 capable device)
  - Intel Embree (highly optimized for Intel CPUs)

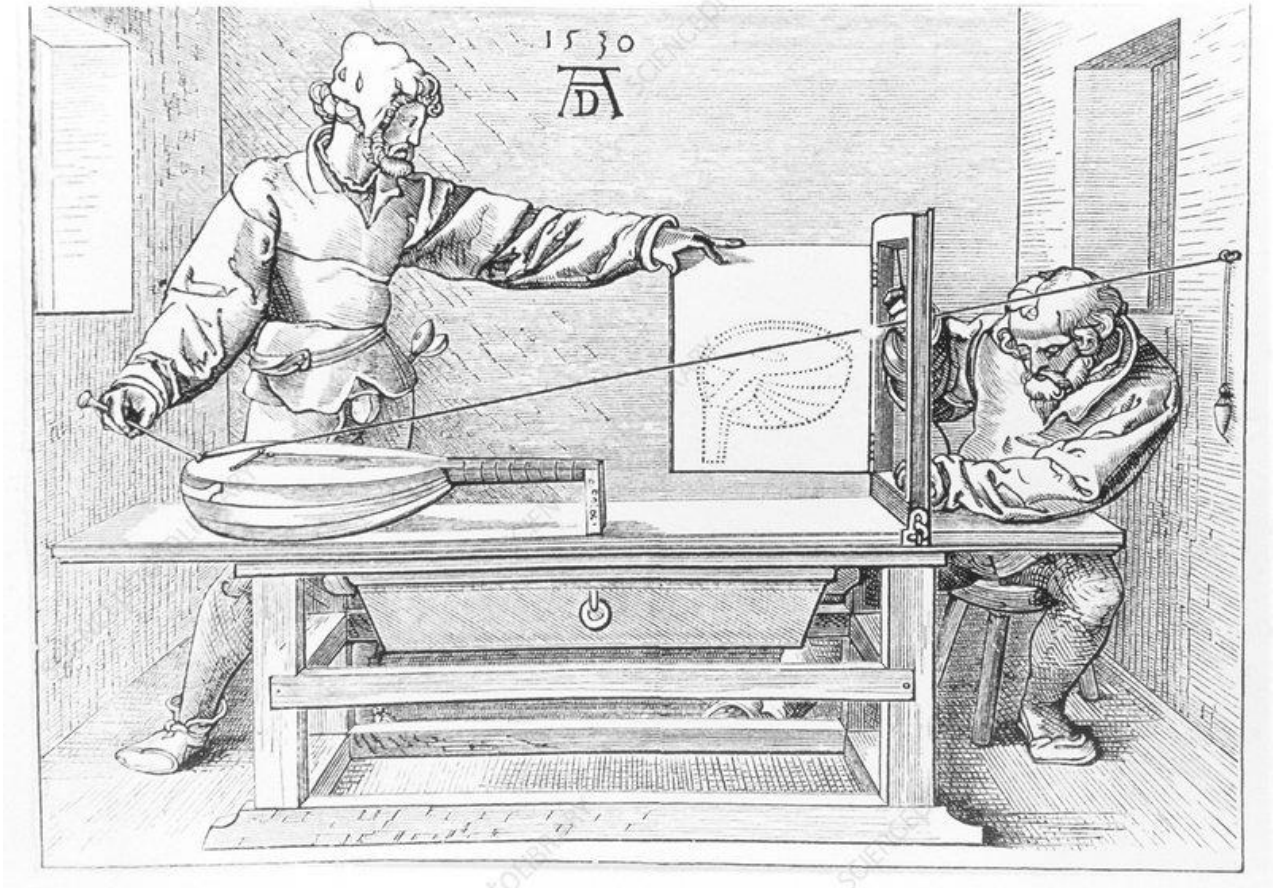


The OptiX API application framework overview

Milestones in the History of Rendering

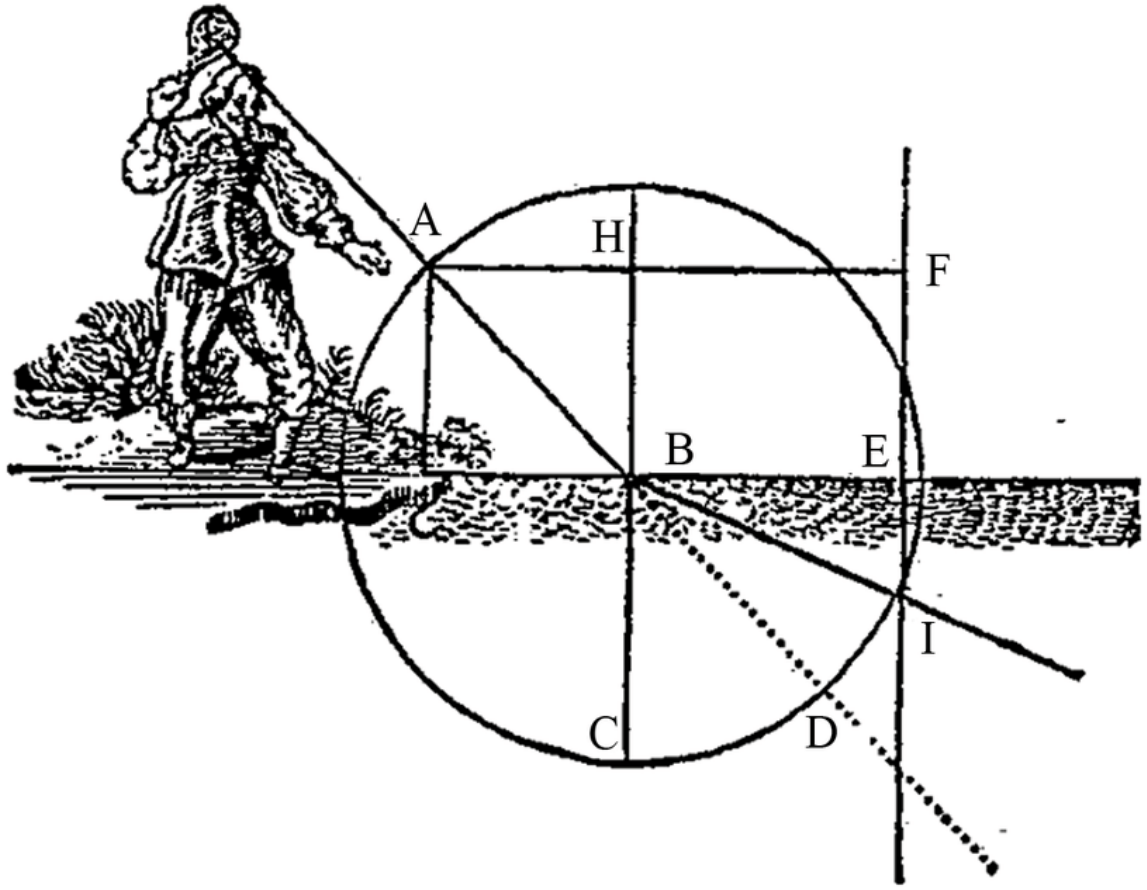
- **Dürer**, Albrecht. **1525**.
  - Perspective projection made by strands of thread
- **Descartes**, Rene. **1637**.
  - Introduced the idea of *tracing light rays*
- **Appel**, Arthur. Some techniques for shading machine renderings of solids. In: Proceedings of the spring joint computer conference. ACM, **1968**. p. 37-45.
  - The first *ray casting* algorithm, surpassed the scan line algorithm from 1967
  - General in terms of surface types, uses traditional shading models
- **Whitted**, Turner. An improved illumination model for shaded display. In: ACM Siggraph Computer Graphics. ACM, **1979**. p. 14.
  - *Ray tracing* introduces three new types of rays: reflected, refracted, and shadow ray
  - Huge improvement but glossy reflection, soft shadows, or caustics can't be simulated
- **Kajiya**, James T. The rendering equation. In: ACM Siggraph Computer Graphics. ACM, **1986**. p. 143-150.
  - Introduces *rendering equation* in the integral form in which the equilibrium radiance leaving a point is given as the sum of emitted plus reflected radiance under a geometric optics approximation

Albrecht Dürer: Measuring the Perspective



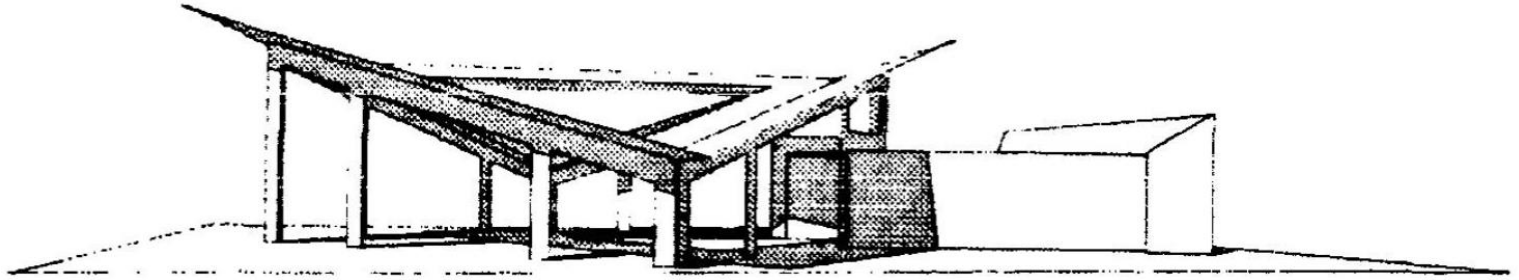
Method of perspective construction, in Underweysung der Messung (1525)

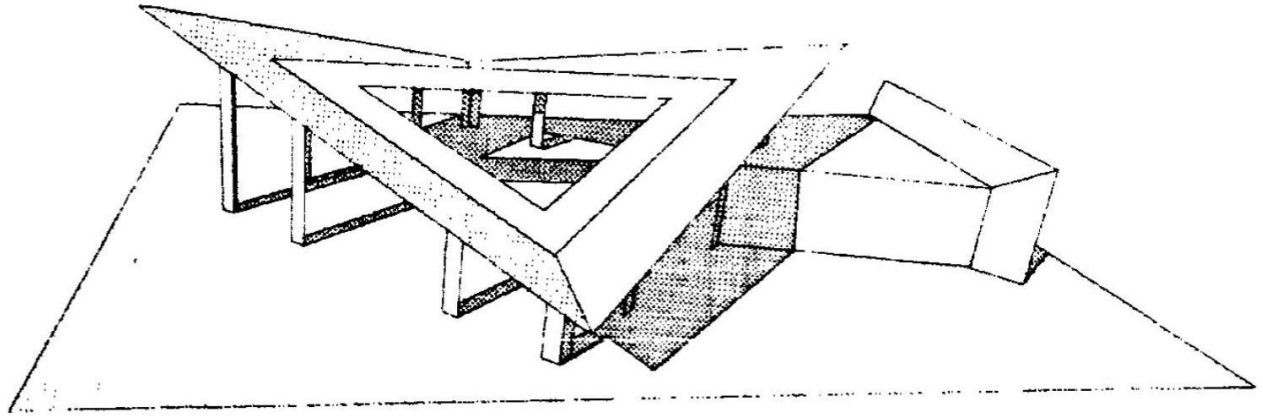
René Descartes: The Reflection of Light



Cartesian theory on light, in Treatise on the Light (1633)

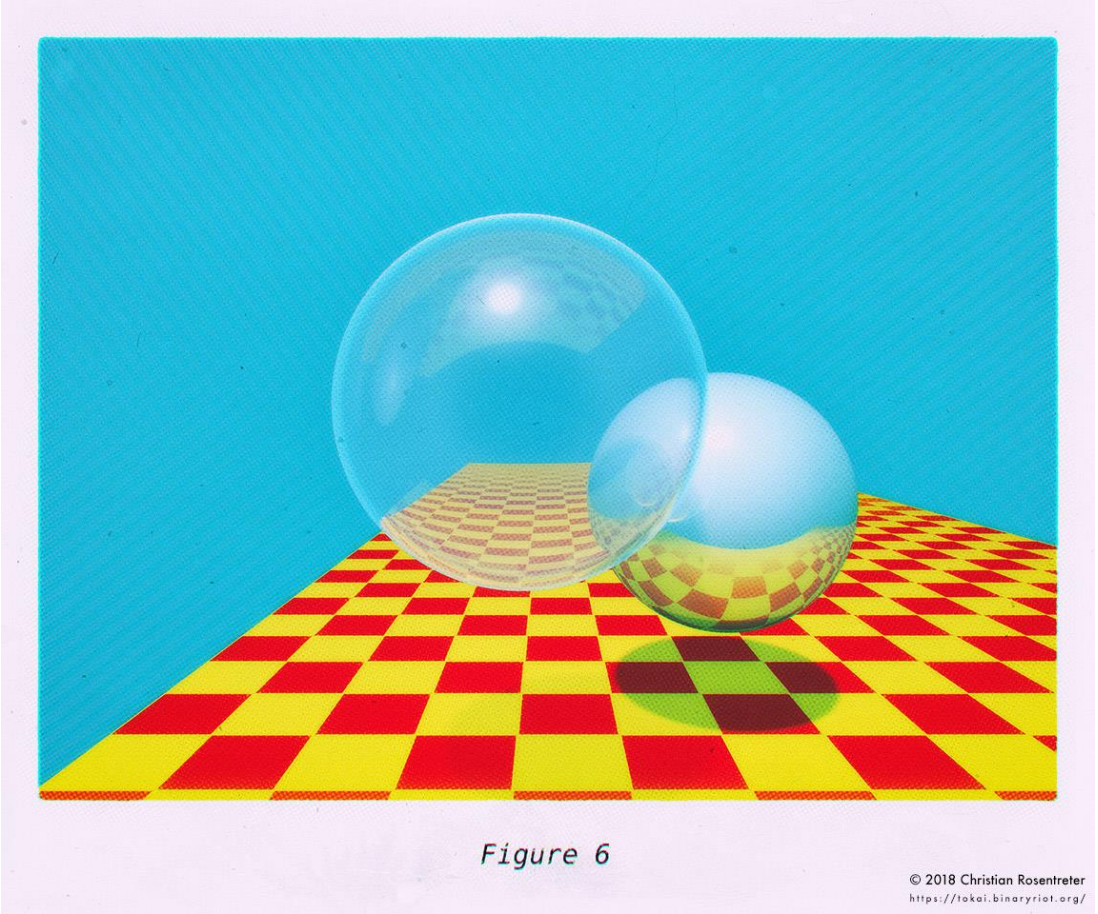
Arthur Appel: On Calculating the Illusion of Reality





First use of a computer for ray tracing to generate shaded pictures, in 1968

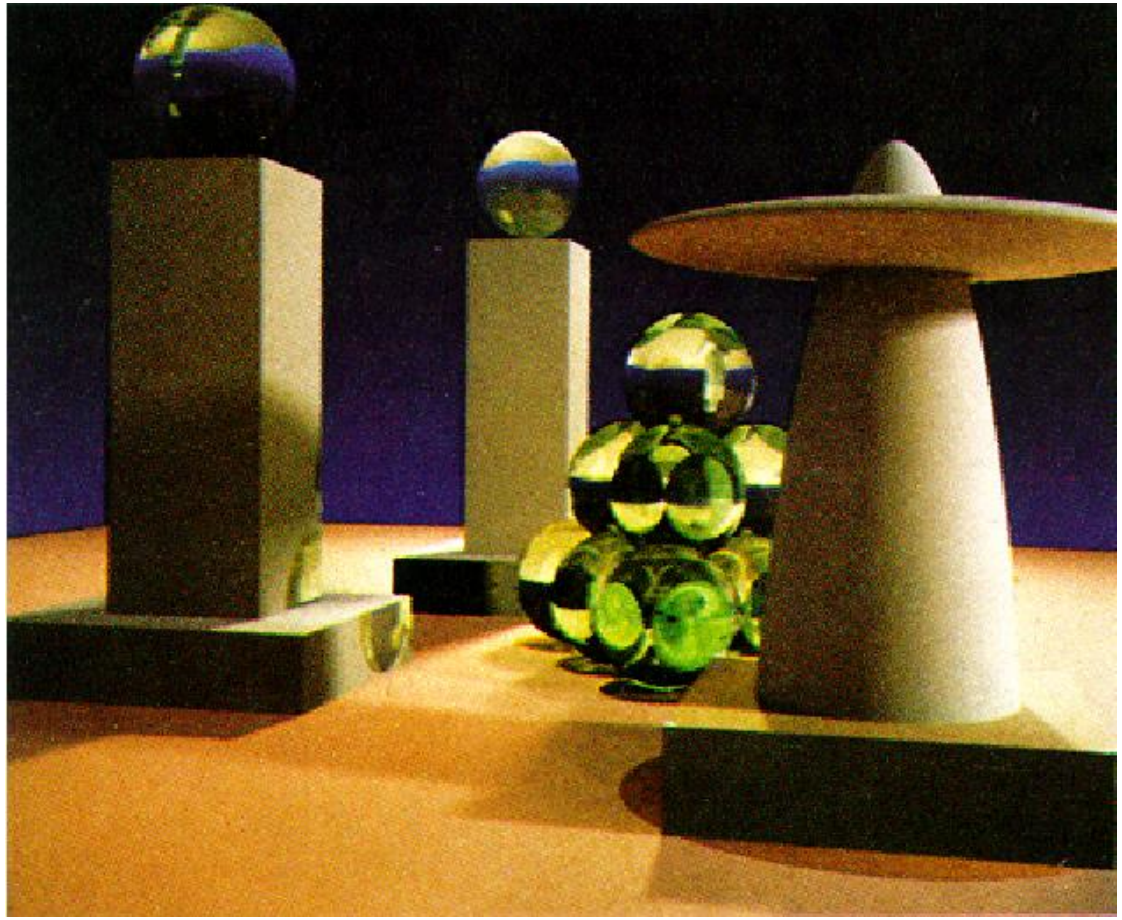
Turner Whitted: Recursive Ray Tracing



Recursive implementation of ray tracing, in An improved illumination model for shaded display (1979)

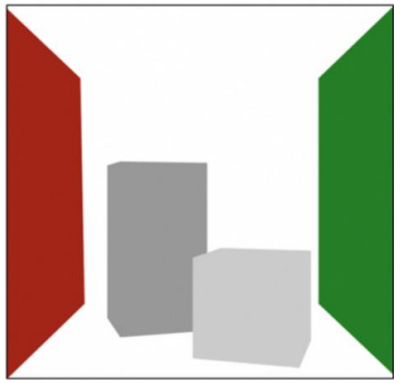
James Kajiya: The Rendering Equation

512×512 image with 40 paths per pixel was computed on an mainframe computer IBM 3081 and consumed 1221 minutes of CPU time.

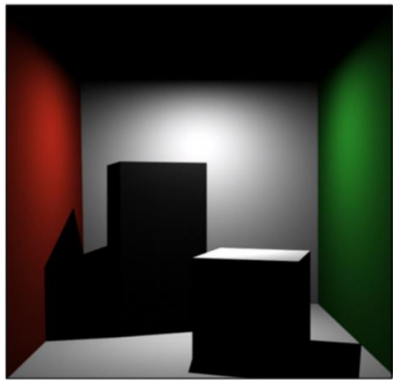


An integral equation which generalizes a variety of known rendering algorithms and simulates a range of optical phenomena , in The rendering equation (1986)

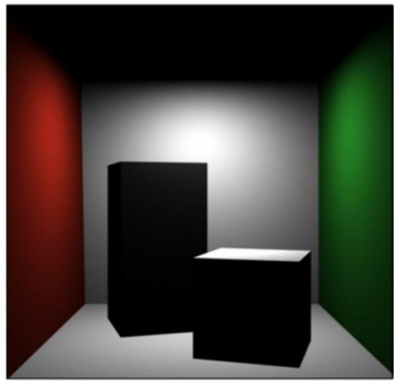
Levels of Realism



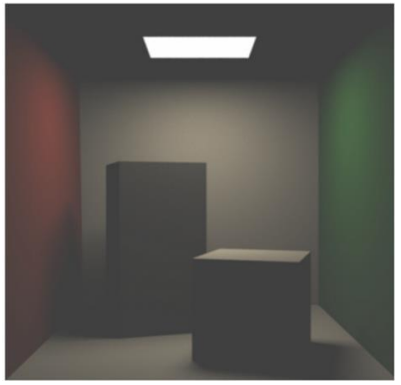
Surface color



Diffuse shading, point light, hard shadows



Diffuse shading, point light, no shadows



Diffuse shading, area light source, soft shadows

Diffuse inter-reflections, area light source

Direct vs. Global Illumination

- Direct illumination
  - A surface point illumination is computed directly from all light sources by the direct illumination model
- Global illumination
  - A surface point illumination is given by the complex light rays interaction with the entire scene

Main (GI) Rendering Methods

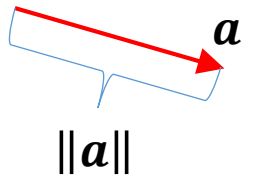
- **Radiosity**
  - fast, finite element
  - only useful for diffuse and semi-glossy scenes
  - performance deteriorates quickly in glossy scenes
  - many artefacts due to light bleeding through, singularity effects
- **Unidirectional path tracing (PT)**
  - best for exteriors (mostly direct lighting)
  - not so good for interiors with much indirect lighting and small light sources
  - very slow for caustics
- **Bidirectional path tracing (BDPT)**
  - best for interiors (indirect lighting, small light sources)
  - fast caustics
  - very slow for reflected caustics
- **Metropolis light transport (MLT) + BDPT**
  - best for interiors (indirect lighting, small light sources)
  - especially useful for scenes with very difficult lighting
  - faster for reflected caustics
- **Energy redistribution path tracing**
  - mix of Monte Carlo PT and MLT
  - best for interiors (indirect lighting, small light sources)
  - much faster than PT for scenes with very difficult lighting
  - fast caustics
  - not so fast for glossy materials
  - problems with detailed geometry
- **Photon mapping**
  - best for indoor scenes
  - biased, artefacts, splotchy, low frequency noise
  - fast, but not progressive
  - large memory footprint
  - very useful for caustics + reflected caustics
- **Stochastic progressive photon mapping**
  - best for indoor
  - fast and progressive
  - very small memory footprint
  - handles all kinds of caustics robustly

Source: http://raytracey.blogspot.com/2011/01/which-algorithm-is-best-choice-for-real.html

Basic Math Operations Review

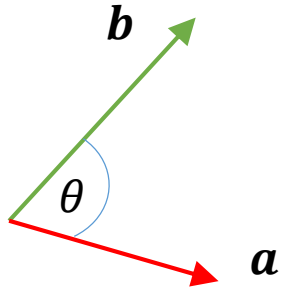
- L2 norm $\|\boldsymbol{a}\| = \sqrt{\boldsymbol{a}_x^2 + \boldsymbol{a}_y^2 + \boldsymbol{a}_z^2} = \sqrt{\boldsymbol{a} \cdot \boldsymbol{a}}$

Unit vector $\hat{\boldsymbol{a}} = \frac{\boldsymbol{a}}{\|\boldsymbol{a}\|}$ and it holds that $\|\hat{\boldsymbol{a}}\| = 1$



- Dot product $\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}^T \boldsymbol{b} = \sum_i a_i b_i = \|\boldsymbol{a}\| \|\boldsymbol{b}\| \cos \alpha$

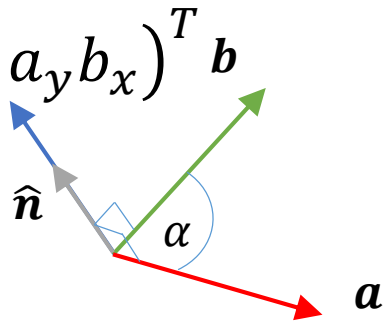
where θ is an angle clamped between both vectors



- Cross product $\boldsymbol{a} \times \boldsymbol{b} = \boldsymbol{c} = \left(a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x\right)^T$

Not commutative, follows the right hand rule, it also holds that

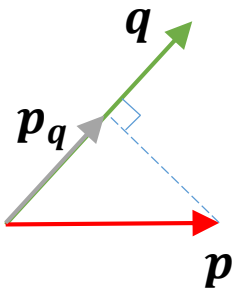
$\boldsymbol{a} \times \boldsymbol{b} = \|\boldsymbol{a}\| \|\boldsymbol{b}\| \hat{\boldsymbol{n}} \sin \alpha$ and $\|\boldsymbol{a} \times \boldsymbol{b}\| = \|\boldsymbol{a}\| \|\boldsymbol{b}\| |\sin \alpha|$



Basic Math Operations Review

- Projection of $\boldsymbol{p}$ on $\boldsymbol{q}$

$$\boldsymbol{p_q} = \frac{\boldsymbol{p}\cdot\boldsymbol{q}}{\|\boldsymbol{q}\|}\,\boldsymbol{q} = (\boldsymbol{p}\cdot\hat{\boldsymbol{q}})\hat{\boldsymbol{q}}$$



Basic Math Operations Review

- Other useful formulas

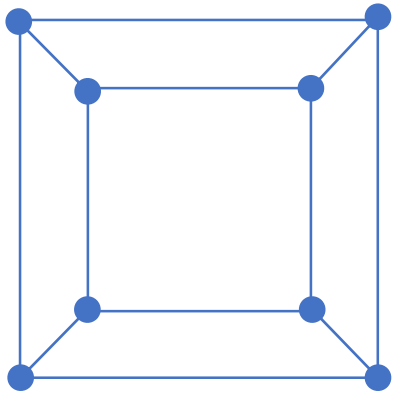
- $\|\boldsymbol{a}-\boldsymbol{b}\|^2 = \boldsymbol{a}\cdot\boldsymbol{a} - \boldsymbol{2}(\boldsymbol{a}\cdot\boldsymbol{b}) + \boldsymbol{b}\cdot\boldsymbol{b}$

- Lagrange's identity $\|\boldsymbol{a}\times\boldsymbol{b}\|^2 = \|\boldsymbol{a}\|^2\|\boldsymbol{b}\|^2 - (\boldsymbol{a}\cdot\boldsymbol{b})^2$

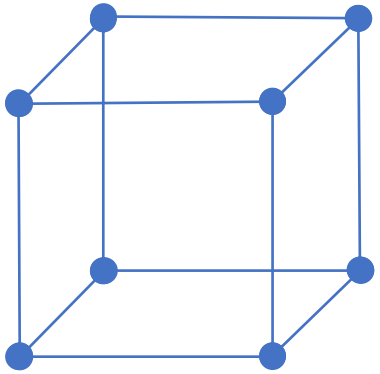
Vertices

Topology is the mathematical study of the properties that are preserved through deformations, twistings, and stretchings of objects. Tearing, however, is not allowed.

- A point (corner or node) where two edges (lines) meet
- The connectedness between the vertices defines a mesh's topology

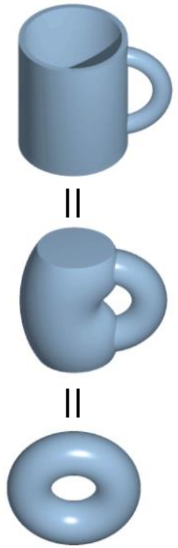


Graph embedding



Topology
Connectivity/neighborhood

Geometry
Position of vertices

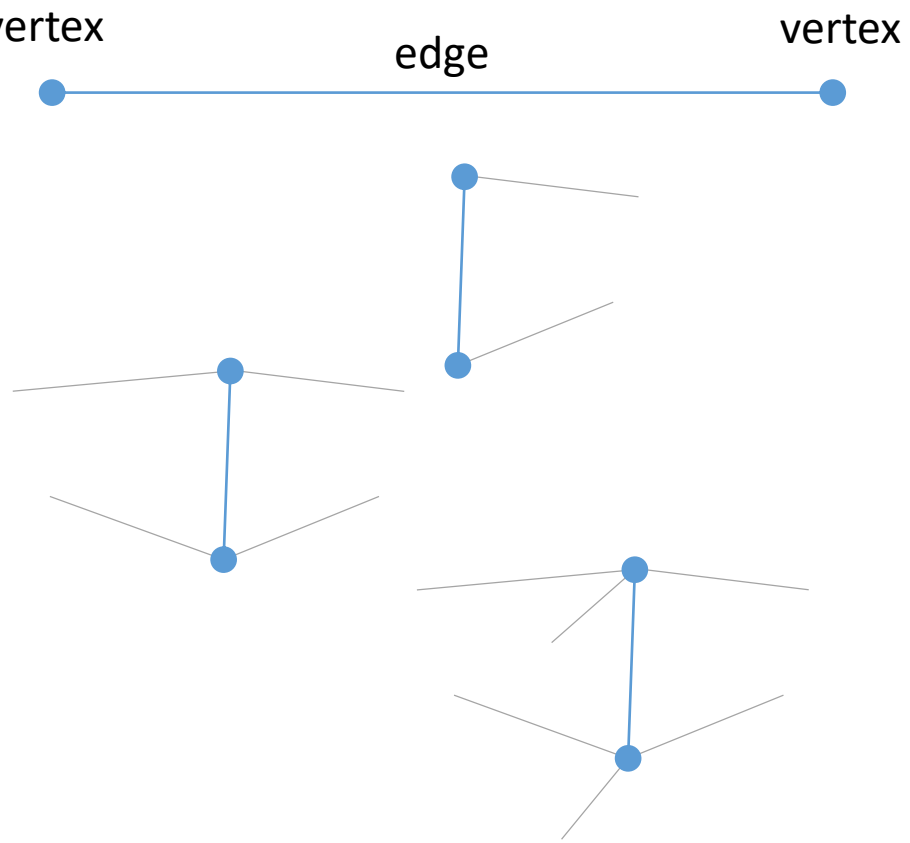


Topological equivalence
(homeomorphism, homotopy)

- Beside the position, vertices may have other attributes: color, normal (tangent, bitangent), texture coordinates, etc.

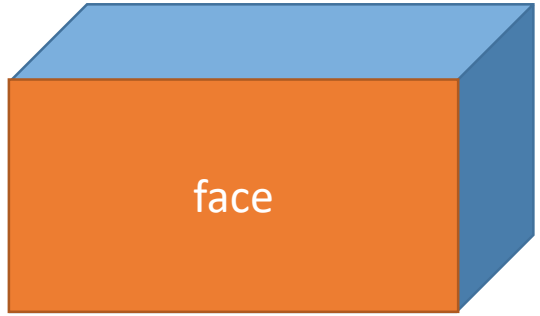
Edges

- Connection of two vertices
- Boundary (1 incident face)
- Regular (2 incident faces)
- Singular (3 or more incident faces)

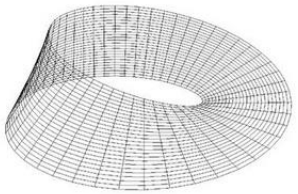


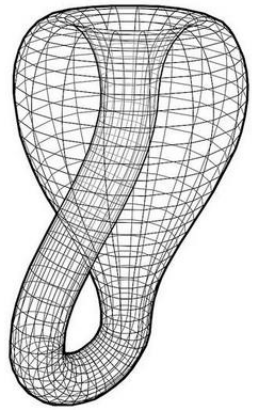
Faces

- The flat surface on a shape or a solid is known as its face



Manifold

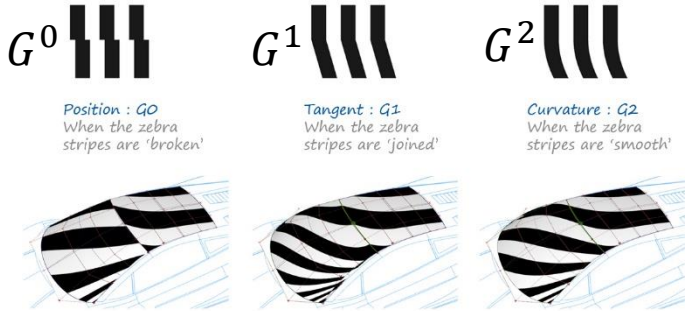




- A n-dimensional manifold is a topological space that locally resembles (homeomorphic) n-dimensional Euclidean space near each point
- 1-manifolds include lines and circles (not a lemniscate or ∞)
- 2-manifold (surface) include plane, sphere (genus 0), torus (genus 1), Klein bottle (not orientable), Möbius strip (closed and not orientable), real projective plane (closed, non-orientable), triangle (smooth manifold with boundary)
- Theorem:
  - Every orientable and closed surface is homeomorphic to a connected sum of tori
  - Every surface is homeomorphic to a connected sum of tori and/or projective planes

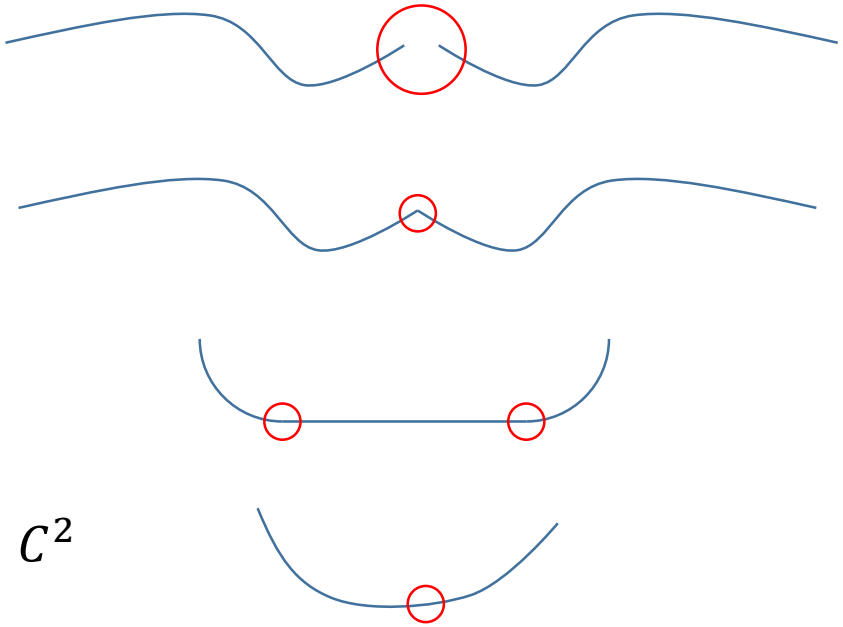
Smoothness

Geometric continuity requires that the parametric derivatives of the two segments be proportional to each other, not equal



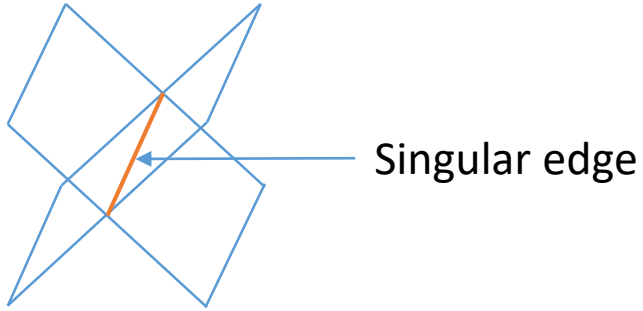
Geometric continuity examples, source: Autodesk Alias Workbench

- Reminder: a function is called C^n continuous if it's n-th order derivative is continuous **everywhere**
- Parametric continuity examples:
  - Not continuous
  - Position continuity $= C^0$
  - Position and tangent continuity $= C^1$
  - Position, tangent, and curvature continuity $= C^2$



Manifold Meshes (with Boundaries)

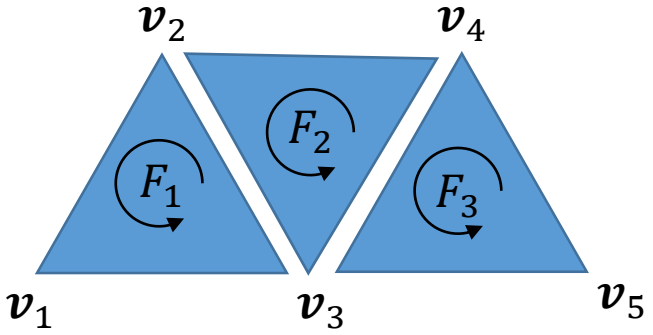
- No singular edges
- No singular vertices



- Simples data structure
  - Polygon soup – list of faces formed by n-tuple of vertices
  - No (explicit) information about adjacency (can be stored in adjacency list)

F_1	$\boldsymbol{v}_1, \boldsymbol{v}_3, \boldsymbol{v}_2$	F_2
F_2	$\boldsymbol{v}_2, \boldsymbol{v}_3, \boldsymbol{v}_4$	F_1, F_2
F_3	$\boldsymbol{v}_3, \boldsymbol{v}_5, \boldsymbol{v}_4$	F_2

Face orientation (CW or CCW) is given by the order in which the vertices are listed



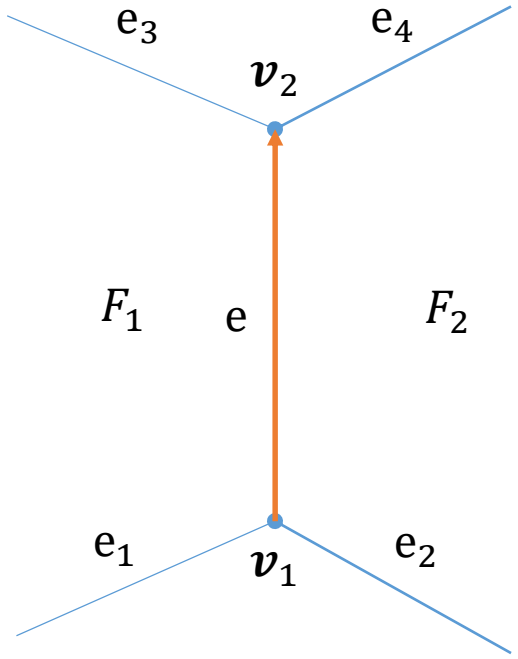
  - Possible extension with indexing (lower memoty req.) (note that shared vertices must have same atributes)

OBJ Format

```
v  14.363998 -19.782551 78.445435
v  14.106861 -19.928997 78.699089
v  13.828163 -19.789648 78.457726
v  14.051285 -19.662575 78.237625
v  14.953951 -19.069986 77.211243
v  14.877973 -19.261126 77.542297

vn -0.111118 0.910786 0.397570
vn 0.132542 0.566302 -0.813439
vn 0.192175 0.622958 -0.758245

f 1//1 2//2 3//3
f 4//1 5//2 6//3
```

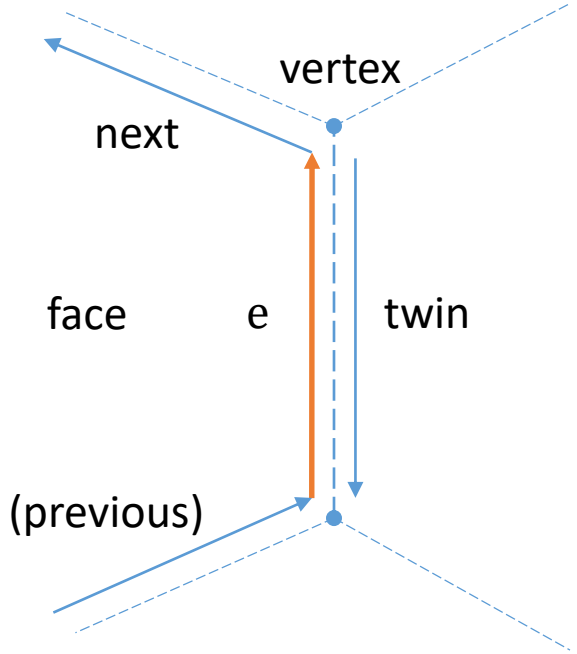
Winged-Edge

- Structure is also valid for non-orientable manifolds



Half-edge Data Structure (DCEL)

- Structure is valid only for orientable manifolds

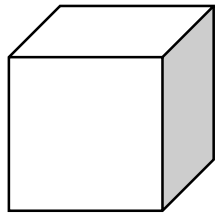


Euler Characteristic χ

The Euler-Poincaré formula describes the relationship of the number of vertices, the number of edges and the number of faces of a manifold. It has been generalized to include potholes and holes that penetrate the solid.

- Simplified Euler's formula for any convex polyhedron's surface

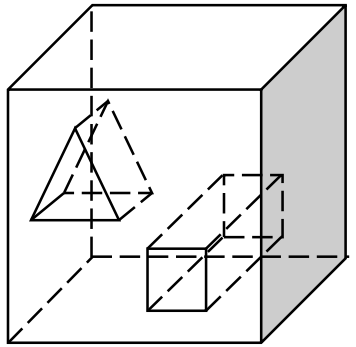
$$V - E + F = 2(S) = \chi$$



$$8 - 12 + 6 = 2(1)$$

- Generalized formula allowing holes

$$V - E + F - L = 2(S - H)$$



$$22 - 33 + 14 - 3 = 2(1 - 1)$$

L … number of holes in faces (inner loops)

H… number of holes passing through the whole solid (genus)

S … number of disjont components (shells, solids)

A homeomorphism is a bijection that is continuous and its inverse is also continuous.

Ray

- A ray r is defined as a parametric line going from an origin O and heading in some unit direction $\widehat{\boldsymbol{d}}$.

$$r(t) = O + \widehat{\boldsymbol{d}}t, \text{ where } t > 0$$

Note that the directional vector is a unit vector and the real parameter t is non negative value representing the length of the ray.

Ray in Embree

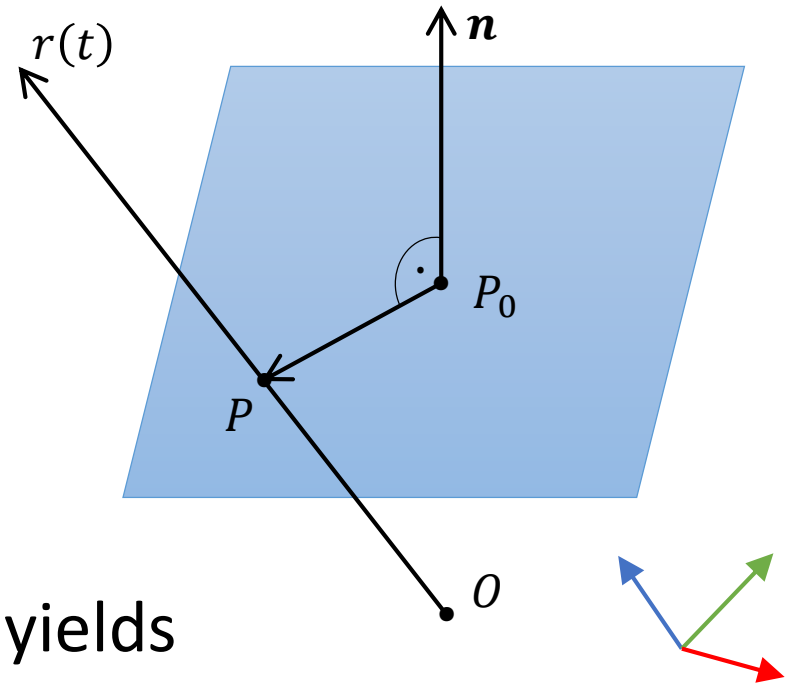
```
#include <embree3/rtcore_ray.h>

struct RTC_ALIGN(16) RTCRay
{
  float org_x;        // x coordinate of ray origin
  float org_y;        // y coordinate of ray origin
  float org_z;        // z coordinate of ray origin
  float tnear;        // start of ray segment

  float dir_x;        // x coordinate of ray direction
  float dir_y;        // y coordinate of ray direction
  float dir_z;        // z coordinate of ray direction
  float time;         // time of this ray for motion blur

  float tfar;         // end of ray segment (set to hit distance)
  unsigned int mask;  // ray mask
  unsigned int id;    // ray ID
  unsigned int flags; // ray flags
};
```

Hit in Embree

```
#include <embree3/rtcore.h>

struct RTCHit
{
  float Ng_x;          // x coordinate of geometry normal
  float Ng_y;          // y coordinate of geometry normal
  float Ng_z;          // z coordinate of geometry normal

  float u;             // barycentric u coordinate of hit
  float v;             // barycentric v coordinate of hit

  unsigned int primID; // geometry ID
  unsigned int geomID; // primitive ID
  unsigned int instID[RTC_MAX_INSTANCE_LEVEL_COUNT]; // instance ID
};
```

Ray-Plane Intersection

- Plane eq. $(P - P_0) \cdot \boldsymbol{n} = 0$
- Ray eq. $r(t) = O + \hat{\boldsymbol{d}}t$

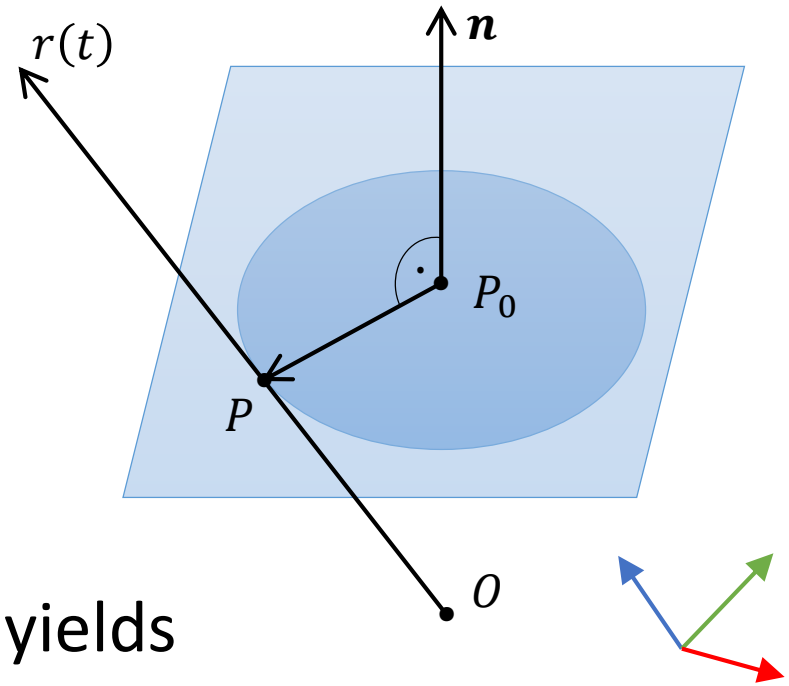


If $P = r(t_{hit})$ then $\left(O + \hat{\boldsymbol{d}}t_{hit} - P_0\right) \cdot \boldsymbol{n} = 0$ yields

$t_{hit} = \frac{(P_0 - O) \cdot \boldsymbol{n}}{\hat{\boldsymbol{d}} \cdot \boldsymbol{n}}$ providing that $\hat{\boldsymbol{d}} \cdot \boldsymbol{n} \neq \boldsymbol{0}$.

Ray-Disc Intersection

- Disc eq. $(P - P_0) \cdot \boldsymbol{n} = 0, \|P - P_0\| \leq R$
- Ray eq. $r(t) = O + \hat{\boldsymbol{d}}t$



If $P = r(t_{hit})$ then $\left(O + \hat{\boldsymbol{d}}t_{hit} - P_0\right) \cdot \boldsymbol{n} = 0$ yields

$t_{hit} = \frac{(P_0 - O) \cdot \boldsymbol{n}}{\hat{\boldsymbol{d}} \cdot \boldsymbol{n}}$ providing that $\hat{\boldsymbol{d}} \cdot \boldsymbol{n} \neq \boldsymbol{0}$ and $\|P - P_0\| \leq R$.

Ray-Triangle Intersection 1

- Step 1: Ray intersects triangle plane (same as Ray-Plane)
- Step 2: Triangle is in front of us, i.e. $t_{hit} \geq 0$
- Step 3: Intersection point $P = r(t_{hit})$ is inside the triangle (there are a number of ways)

For each edge of the triangle holds that the intersecting point is on the *left side* of this edge, i.e.

$$C_0 = (v_1 - v_0) \times (P - v_0), C_0 \cdot N > 0$$

and

$$C_1 = (v_2 - v_1) \times (P - v_1), C_1 \cdot N > 0$$

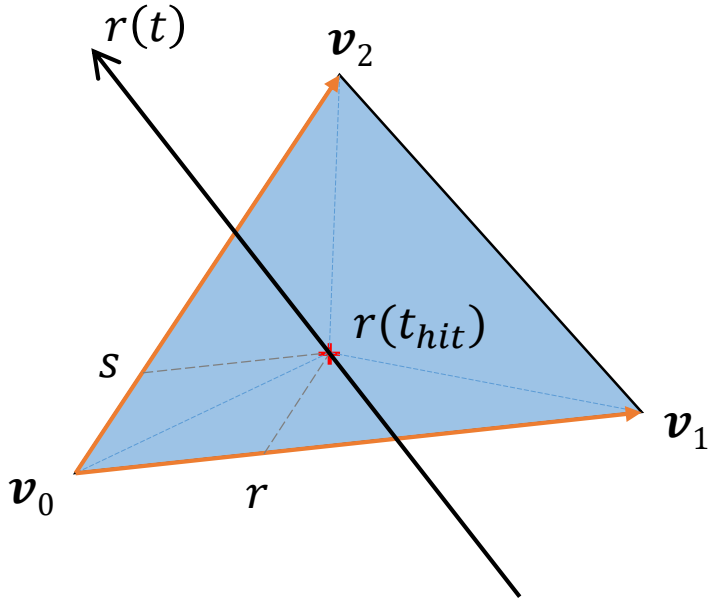
and

$$C_2 = (v_0 - v_2) \times (P - v_2), C_2 \cdot N > 0$$

Ray-Triangle Intersection 2

- Triangle eq. $T(r,s) = (1-r-s)\boldsymbol{v}_0 + r\boldsymbol{v}_1 + s\boldsymbol{v}_2 =$

  $= \boldsymbol{v}_0 + r(\boldsymbol{v}_1 - \boldsymbol{v}_0) + s(\boldsymbol{v}_2 - \boldsymbol{v}_0), r \geq 0, s \geq 0, r + s \leq 1$
- Ray eq. $r(t) = O + \hat{\boldsymbol{d}}t$
- Set $T(r,s) = r(t_{hit})$ and we get 3 equations and 3 unknowns (r, s, and t_{hit})



Ray-Triangle Intersection 3

- $\boldsymbol{v}_{0_x} + r(\boldsymbol{v}_{1_x} - \boldsymbol{v}_{0_x}) + s(\boldsymbol{v}_{2_x} - \boldsymbol{v}_{0_x}) = O_x + \hat{\boldsymbol{d}}_x t_{hit}$

  $\dots y$

  $\dots z$

- $r(\boldsymbol{v}_{1_x} - \boldsymbol{v}_{0_x}) + s(\boldsymbol{v}_{2_x} - \boldsymbol{v}_{0_x}) - t_{hit}\hat{\boldsymbol{d}}_x = O_x - \boldsymbol{v}_{0_x}$

  $\dots y$

  $\dots z$

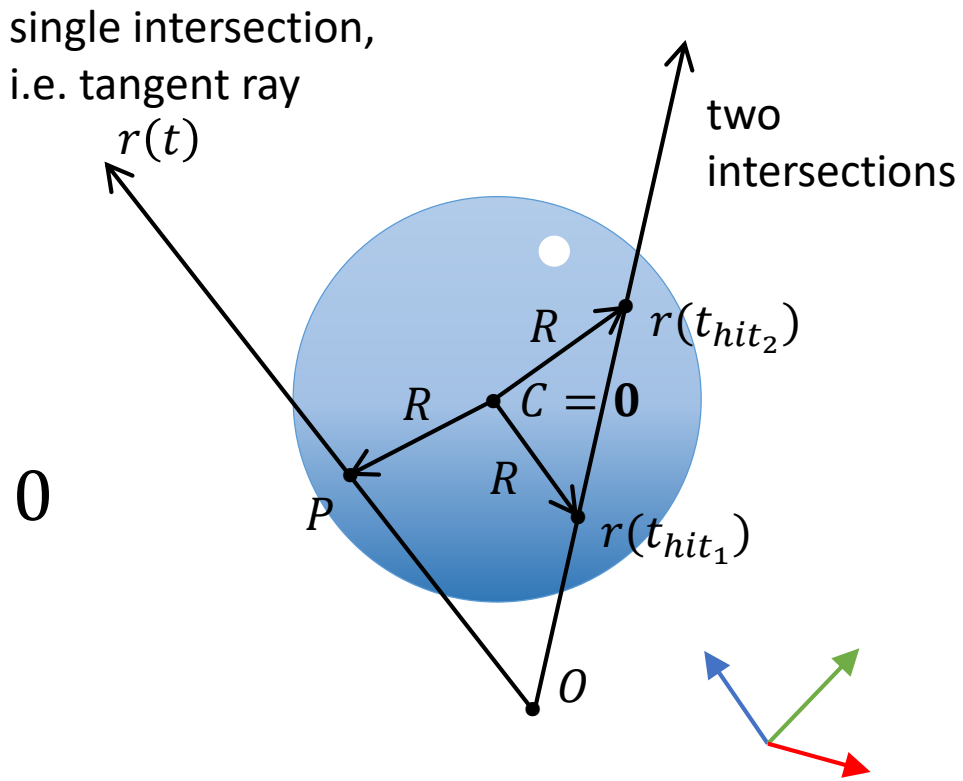
- $$\begin{bmatrix} \boldsymbol{v}_{1_x} - \boldsymbol{v}_{0_x} & \boldsymbol{v}_{2_x} - \boldsymbol{v}_{0_x} & -\hat{\boldsymbol{d}}_x \\ & \dots y & \\ & \dots z & \end{bmatrix} \begin{bmatrix} r \\ s \\ t_{hit} \end{bmatrix} = \begin{bmatrix} O_x - \boldsymbol{v}_{0_x} \\ \dots y \\ \dots z \end{bmatrix}$$

$$Ax = b$$
$$x = A^{-1}b$$

Fast Ray-Triangle Intersection

- The standard algorithm for computing ray-triangle intersections in ray tracing: MÖLLER, Tomas; TRUMBORE, Ben. Fast, Minimum Storage Ray-Triangle Intersection, *Journal of Graphics Tools* 1997.
- Given a ray and a triangle, this algorithm transforms the ray from the global coordinate system to a triangle-specific barycentric one and then tests for intersection in that coordinate system. The algorithm is fast because it requires only a modest number of calculations and supports early detection of many cases in which the ray cannot intersect the triangle. No information is precomputed, and thus there is no memory overhead.
- Precomputing the coordinate transformation to be faster than MT: BALDWIN, Doug; WEBER, Michael. Fast Ray-triangle Intersections by Coordinate Transformation. *Journal of Computer Graphics Techniques* 2016.

Ray-Sphere Intersection



- Sphere eq. $x^2 + y^2 + z^2 = R^2, R > 0$

or alternatively $\|P\|^2 - R^2 = 0$

For $C \neq \mathbf{0}$ we can write $\|P - C\|^2 - R^2 = 0$

- Ray eq. $\quad r(t) = O + \hat{\boldsymbol{d}}t$

If $P = r(t_{hit})$ then $\left\|O + \hat{\boldsymbol{d}}t_{hit} - C\right\|^2 - R^2 = 0$ yields quadratic eq.

$$\left\|\hat{\boldsymbol{d}}\right\|^2 {t_{hit}}^2 + [2((O - C) \cdot \hat{\boldsymbol{d}})]t_{hit} + [\|O - C\|^2 - R^2] = 0.$$

Ray-X Intersections

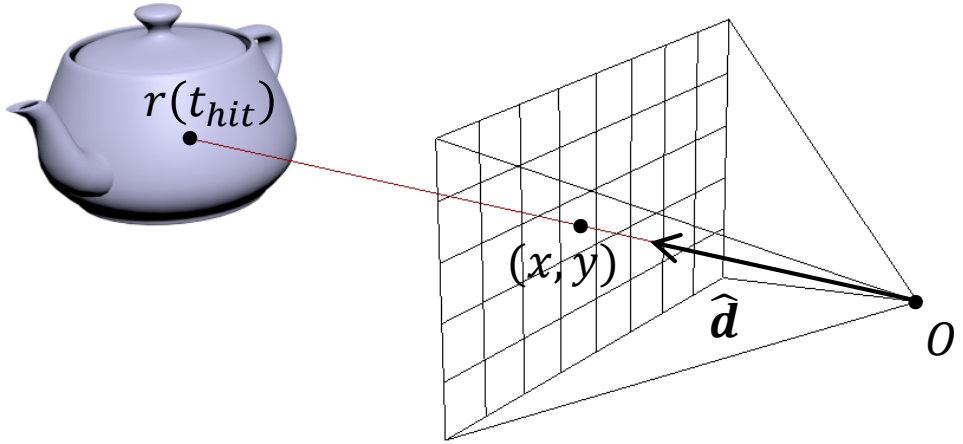
- More intersection routines are available, e.g. Ray-AABB, Ray-OOBB, Ray-Cylinder, Ray-Cone, Ray-NURBS etc.

- *Further reference:*

http://www.realtimerendering.com/intersections.html

Ray Casting

- At the begining of each ray tracer there is a primary/camera ray
- Primary ray originates in the eye – the focus point of the camera projection (viewing frustum)
- We need to find out the right O and $\hat{\boldsymbol{d}}$ for each ray passing through the sensor of our virtual camera (pinhole camera or camera obscura)



How to set O is clear, but how to find $\hat{\boldsymbol{d}}$ for given pixel (x, y)?

Ray Casting

- Pseudocode of a simple ray caster:

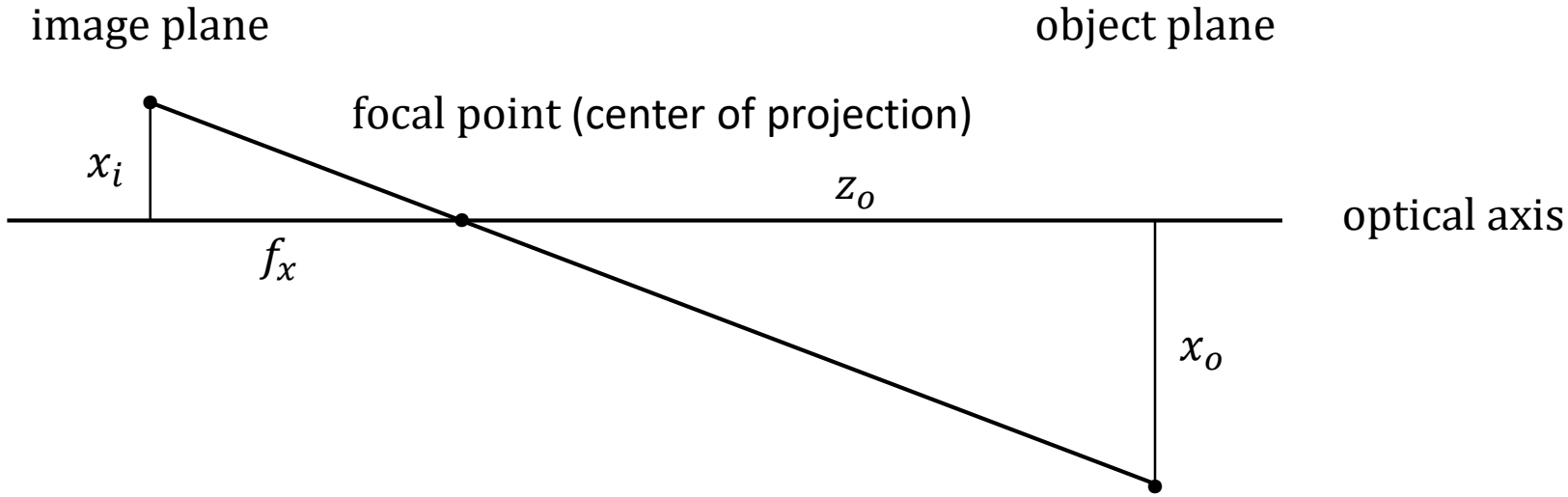
```
for each pixel:
	compute ray from eye through pixel
	for each primitive:
		find closes intersection
	if intersection exists:
		shade pixel using material parameters at intersection
	else:
		set pixel to background color
```

- What is the time complexity of this algorithm?

Pinhole Camera Model

Similar triangles: All the corresponding sides have lengths in the same ratio.
$\frac{a}{a'} = \frac{b}{b'} = \frac{c}{c'}$

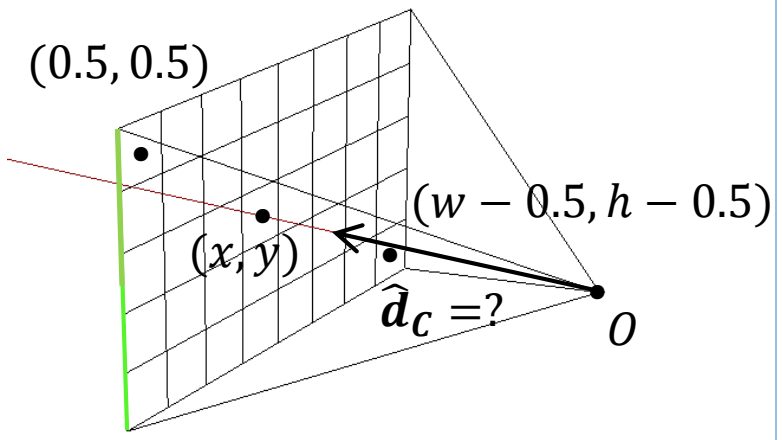
- Idealized model for the optics of a camera defining the geometry of perspective projection

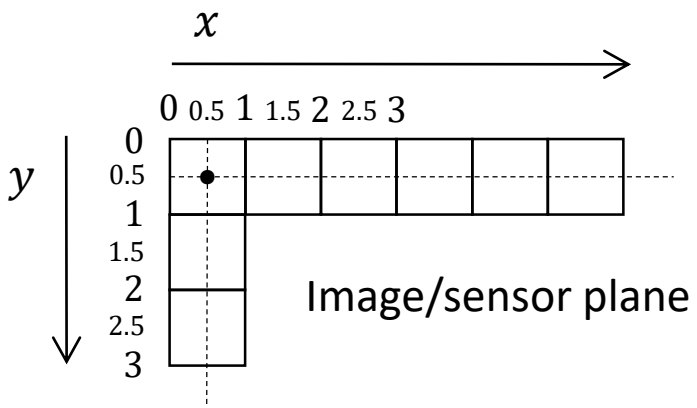


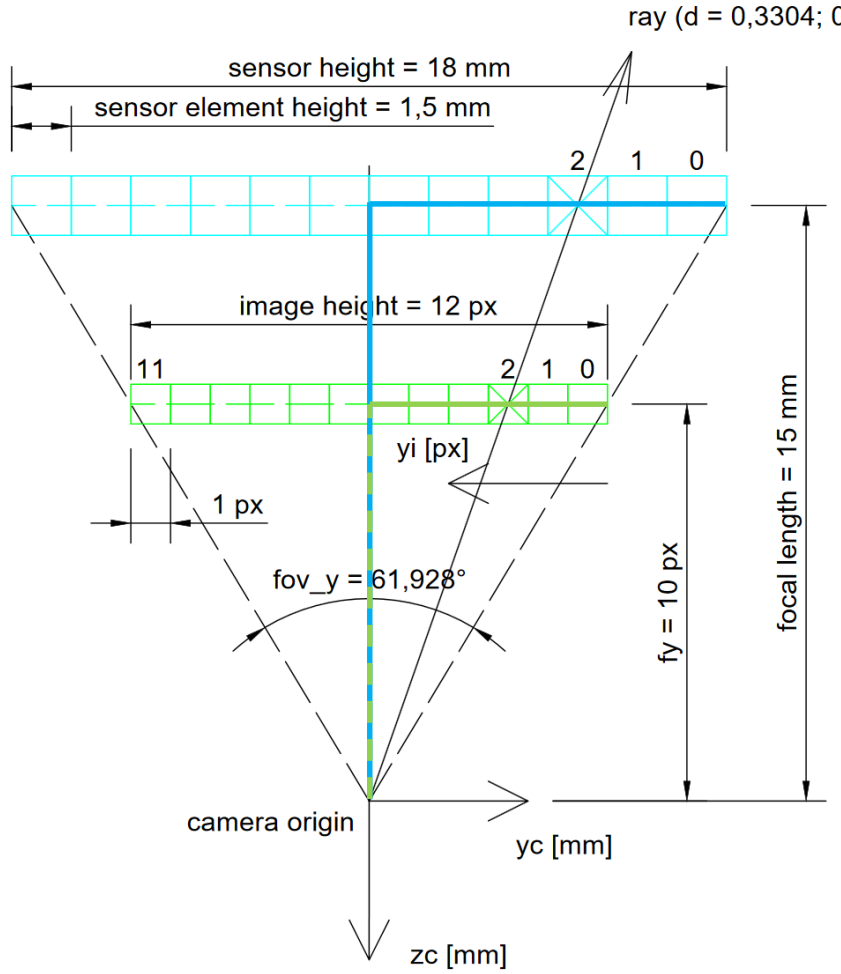
$$\frac{x_i}{f_x} = \frac{x_o}{z_o}, \quad \frac{y_i}{f_x} = \frac{y_o}{z_o}$$

i … image point coordinates
o … object point coordinates

Primary Ray Generation in Camera Space







sensor plane — What would we deal with in the case of a real camera

image plane — What will we deal with in the case of an imaginary camera

Known camera parameters:
O, T, width w, height h, fov_y

$f_y = f_x = ?$ (px)

$\boldsymbol{d_C} = (x - w/2\,, h/2 - y\,, -f_y)$

Primary Ray Generation in World Space

- Now we have to put $\widehat{\boldsymbol{d}}_C$ in the world space where all our objects live
- We define a camera coordinate system by providing three orthogonal basis as follows (T is camera target and vector $\boldsymbol{up} = (0 \quad 0 \quad 1)^T$)

$$\boldsymbol{z}_c = O - T,\ \boldsymbol{x}_c = \boldsymbol{up} \times \boldsymbol{z}_c, \text{ and } \boldsymbol{y}_c = \boldsymbol{z}_c \times \boldsymbol{x}_c$$

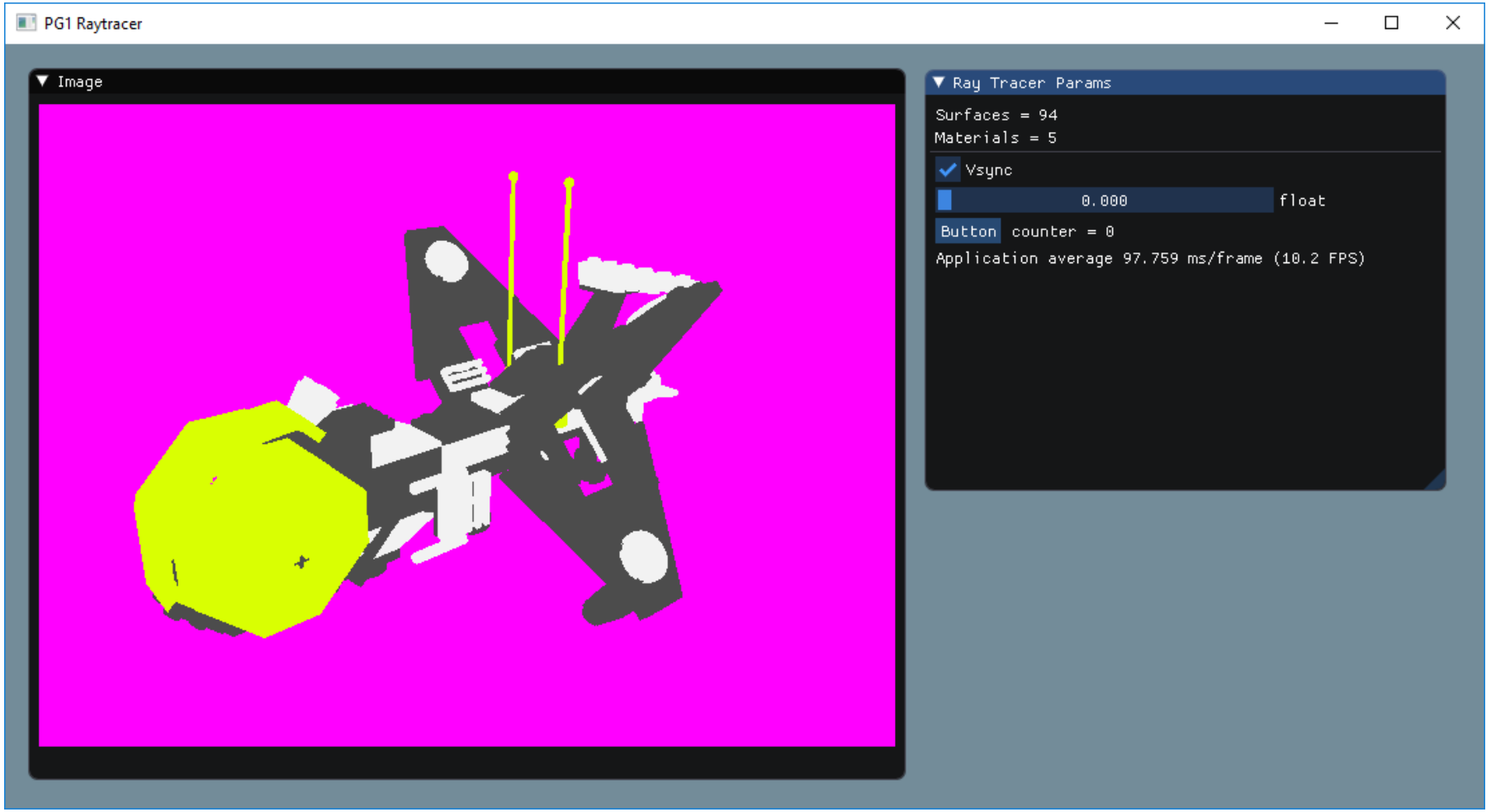
Note that both O and T are in world space.

- Basis form transformation matrix $M_{C\to W} = \begin{pmatrix} \vdots & \vdots & \vdots \\ \widehat{\boldsymbol{x}}_c & \widehat{\boldsymbol{y}}_c & \widehat{\boldsymbol{z}}_c \\ \vdots & \vdots & \vdots \end{pmatrix}$
- Prim. ray direction vector for pixel (x, y) in WS $\widehat{\boldsymbol{d}}_W = M_{C\to W}\widehat{\boldsymbol{d}}_C$
- Finally, we get our primary ray as $r(t) = O + \widehat{\boldsymbol{d}}_W t$

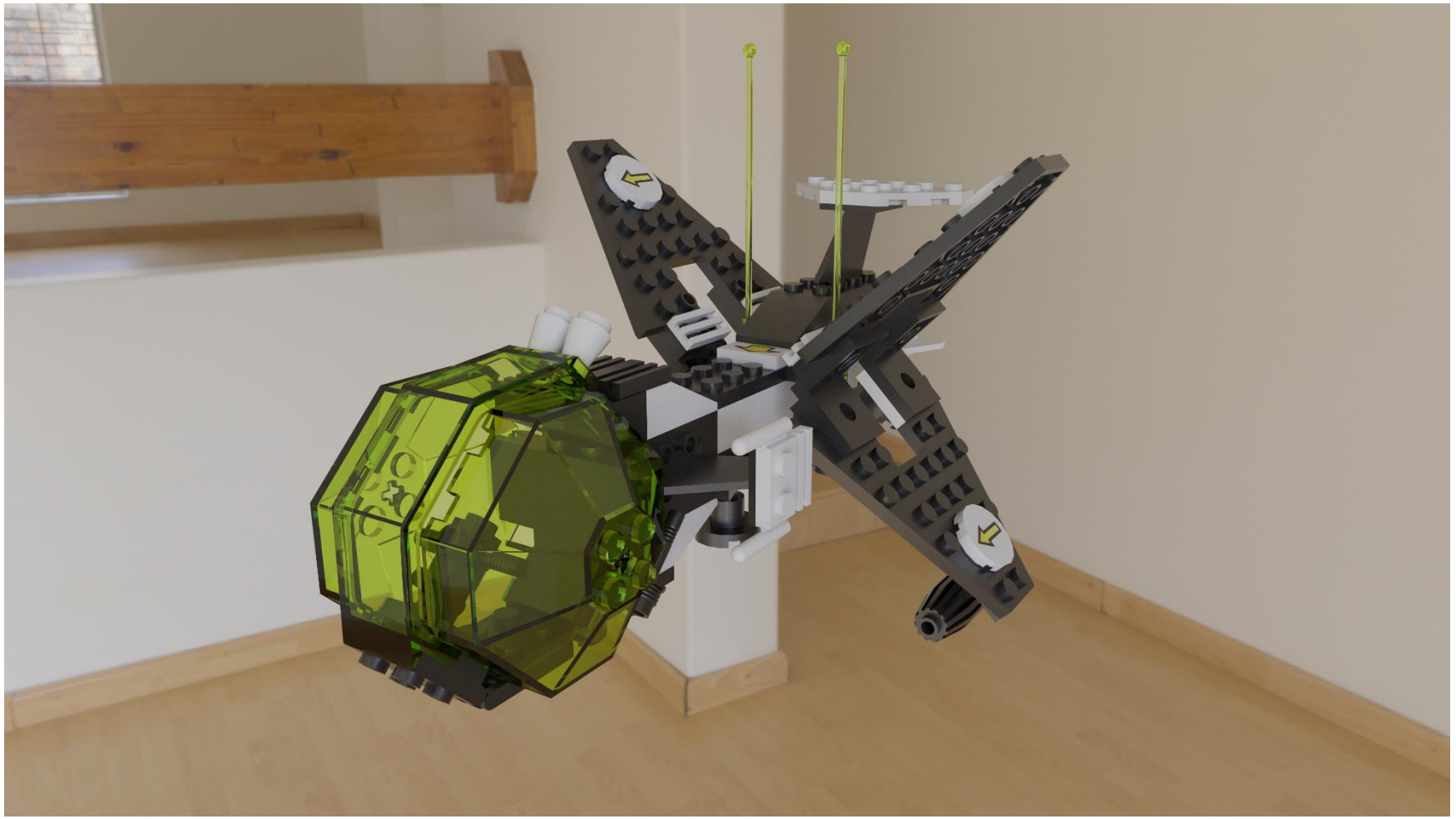
Answer from the previous slide:

$$f_y = \frac{h}{2\tan\left(\frac{fov_y}{2}\right)}$$

Result of First Exercise



Motivation



Quiz

- What are the common representations of polygonal meshes in computer memory? [2]
- What are the topological properties of solids/manifolds? What the Euler's characteristic denotes? [2, 3]
- What are the typical components of a triangular face? [3]
- Describe the meaning of basic mathematical operations used in computer graphics. [3]
- How can we change the base of the coordinate system? [2, 3]
- Can you describe the idealized model of a pinhole camera? [3, 4]
- What are the main parameters of a pinhole camera? [3, 4]

Quiz

- How a single ray is represented? [3]
- Outline the ray casting algorithm. What is its time complexity? [1]
- How to determine the intersection of a ray with selected geometric objects? [3]
- What is the fundamental difference between local and global illumination? [8]

Appendix

- Following slides are optional and describe the organization of a frame buffer including alpha channel as used in the reference implementation of the PG1 Ray Tracer

Framebuffer

- #s(0) … number of mis hits (every sample is valid)
- #s(1) … number of samples from depth 1 (every sample is valid)
- #s(end) … number of samples from depth >1 (only valid samples are included)
- #s = #s(0) + #s(1) = #s(0) + #s(end) + #invalid samples
- Normal … #s(1) surface normals at hit(1)
- Albedo … #s(1) surface diffuse colors at hit(1)
- Position … #s(1) WS positions of hit(1)
- Foreground color … #s(end) radiances comming from hit(end)
- Background color … #s(0) radiances comming from hit(0), i.e. background

Framebuffer (AoS)

```
struct RayPayload
{
Color3f foreground;
Color3f background;
Color3f albedo;
Normal3f normal;
Vector3 position;

int s_0{ 0 }; // +1 when mis hit (background samples)
int s_1{ 0 }; // +1 when primary ray hits the surface
int s_end{ 0 }; // +1 when valid radiance is returned
after one or more hits

// samples normalization, call it only once !
float normalize()
{
  if ( s_0 + s_1 > 0 ) {
    const float alpha = s_end / float( s_0 + s_1 );

    if ( s_0 > 0 )
      background *= 1.0f / s_0;

    if ( s_1 > 0 ) {
      albedo *= 1.0f / s_1;
      normal.normalize();
      position *= 1.0f / s_1;
    }

    if ( s_end > 0 )
      foreground *= 1.0f / s_end;

    s_0 = s_1 = s_end = 0;

    return alpha;
  }

  return 0.0f; // fully transparent pixel
} };
```

Image Composition – Alpha Blending

- alpha … opacity value (0 – transparent fg, 1 – opaque fg)
- rgb … **linear** color (radiance)

- Associated (premultiplied) alpha

$$c.rgb = fg.rgb + bg.rgb \cdot (1 - fg.alpha)$$

  - In case of black background, ray tracers naturally produce alpha premultiplied rgb values and alpha is a ratio of #hits/#samples

- Unassociated (straight) alpha

$$c.rgb = fg.rgb \cdot fg.alpha + bg.rgb \cdot (1 - fg.alpha) \quad \text{(over op.)}$$

or (in faster form)

$$c.rgb = (fg.rgb - bg.rgb) \cdot fg.alpha + bg.rgb$$

PORTER, Thomas; DUFF, Tom. Compositing digital images. In: *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. 1984. p. 253-259.

Premultiplication

- Results of fg and bg composition in GIMP

Unassociated alpha

Associated alpha

Premultiplication

- Results of denoised fg and bg composition in GIMP

Unassociated alpha

Associated alpha