Einführung Computergraphik (SS 2025)

Martin Held

FB Informatik
Universität Salzburg
A-5020 Salzburg, Austria
held@cs.sbg.ac.at

23. Mai 2025

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Personalia

Instructor:	M. Held.
Email:	held@cs.sbg.ac.at
Base-URL:	https://www.cosy.sbg.ac.at/~held.
Office:	Universität Salzburg, FB Informatik, Rm. 1.20, Jakob-Haringer Str. 2, 5020 Salzburg-Itzling.
Phone number (office):	(0662) 8044-6304.
Phone number (secr.):	(0662) 8044-6300.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Formalia

URL of course: Base-URL/teaching/einfuehrung_graphik/cg.html.

Lecture times (VO): Thursday 11^{30}–13^{25}.

Lecture times (PS): Thursday 13^{50}–14^{45}.

Venue: Univ. Salzburg, FB Informatik, T05, Jakob-Haringer Str. 2.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Electronic Slides and Online Material

In addition to these slides, you are encouraged to consult the WWW home-page of this lecture:

https://www.cosy.sbg.ac.at/~held/teaching/einfuehrung_graphik/cg.html.

In particular, this WWW page contains links to online manuals, slides, and code.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

A Few Words of Warning

I hope that these slides will serve as a practice-minded introduction to various aspects of computer graphics. I would like to warn you explicitly not to regard these slides as the sole source of information on the topics of my course. It may and will happen that I'll use the lecture for talking about subtle details that need not be covered in these slides! That is, by making these slides available to you I do not intend to encourage you to attend the lecture on an irregular basis.

Acknowledgements

Several students contributed to the genesis of these slides, by assembling reports on graphics projects, producing electronic transcripts of my own lectures, and by writing LaTeX code and generating computer-based figures:

Richard Bauer, Stephan Czermak, Gerd Dauenhauer, Mohamed Elkattahf, Christian Gasperi, Martin Hargassner, Claudia Horner, Christian Koidl, Florian Krisch, Claudio Landerer, Lothar Mausz, Kathrin Meisl, Oskar Schobesberger, Roland Schorn, Rolf Sint, Alex Stumpfl, Oliver Suter, Florian Treml, Christian Zödl, and Gerhard Zwingenberger; Matthias Ausweger, Günther Gschwendtner, Herwig Höfle, Balthasar Laireiter, Bernhard Salzlechner, and Gerald Wiesbauer; and Markus Amersdorfer, Martin Angerer, Matthias Ausweger, Richard Bauer, Fritz Bischof, Ronald Blaschke, Michael Brachtl, Markus Chalupar, Walter Chalupar, Werner Dietl, Johann Edtmayr, Gregor Haberl, Dorly Harringer, Sandor Herramhof, Martin Hinterseer, Hermann Huber, Gyasi Johnson, Wolfgang Klier, August Mayer, Albert Meixner, Christof Meerwald, Michael Neubacher, Michael Noisternig, Christoph Oberauer, Christoph Obermair, Peter Palfrader, Marc Posch, Christopher Rettenbacher, Herwig Rittsteiger, Gerhard Scharfetter, Josef Schmidbauer, Ingrid Schneider, Harald Schweiger, Stefan Sodar, Gerald Stieglbauer, Marc Strapetz, Johanna Temmel, Christopher Vogl, Werner Weiser, Gerald Wiesbauer, Franz Wilhelmstötter.

I would like to express my thankfulness for their help with these slides. My apologies go to all those who should be on this list and who I omitted by mistake.
This revision and extension was carried out by myself, and I am responsible for all errors.

Salzburg, February 2025 Martin Held

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Legal Fine Print and Disclaimer

To the best of our knowledge, these slides do not violate or infringe upon somebody else's copyrights. If copyrighted material appears in these slides then it was considered to be available in a non-profit manner and as an educational tool for teaching at an academic institution, within the limits of the "fair use" policy. For copyrighted material we strive to give references to the copyright holders (if known). Of course, any trademarks mentioned in these slides are properties of their respective owners.

Please note that these slides are copyrighted. The copyright holder(s) grant you the right to download and print it for your personal use. Any other use, including non-profit instructional use and re-distribution in electronic or printed form of significant portions of it, beyond the limits of "fair use", requires the explicit permission of the copyright holder(s). All rights reserved.

These slides are made available without warrant of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. In no event shall the copyright holder(s) and/or their respective employers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, arising out of or in connection with the use of information provided in these slides.

Recommended Textbooks I


S. Guha.
Computer Graphics Through OpenGL: From Theory to Experiments.
CRC Press, 4th edition, Dec 2022; ISBN 978-1032256986.


J. Kessenich, G. Sellers, and D. Shreiner.
The OpenGL Programming Guide.
Addison-Wesley, 9th edition, 2016; ISBN 978-0134495491.
http://www.opengl-redbook.com/


G. Sellers, R.S. Wright, and N. Haemel.
OpenGL SuperBible.
Addison-Wesley, 7th edition, 2015; ISBN 978-0672337475.
http://www.openglsuperbible.com/


J. de Vries.
Learn OpenGL – Graphics Programming.
Kendall&Welling, June 2020; ISBN 978-9090332567.
https://learnopengl.com/

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Recommended Textbooks II


S. Marschner, P. Shirley.
Fundamentals of Computer Graphics.
CRC Press, 5th edition, Aug 2021; ISBN 978-0367505035.


T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, and S. Hillaire.
Real-Time Rendering.
CRC Press, 4th edition, 2018; ISBN 978-1138627000.
`http://www.realtimerendering.com`.


K. Suffern.
Ray Tracing from the Ground Up.
CRC Press, April 2016; ISBN 978-1-498774703. New edition in Dec 2025!


M. Pharr, W. Jakob, and G. Humphreys.
Physically Based Rendering.
Morgan Kaufmann, 4th edition, March 2023; ISBN 978-0262048026.
`https://www.pbrt.org/`, `http://www.pbr-book.org/`.


H.W. Jensen.
Realistic Image Synthesis Using Photon Mapping.
CRC Press, 2nd edition, Jan 2015; ISBN 978-1568811970.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Table of Content

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

1 Introduction

- Survey of Computer Graphics
- Basics

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

What is Computer Graphics?

The term “computer graphics” was coined by William Fetter in 1960 to describe the work he was pursuing at Boeing.

> *“ . . . a consciously managed and documented technology directed toward communicating information accurately and descriptively.”*
>
> *William A. Fetter, “Computer Graphics” (1960).*

Computer graphics is generally regarded as the creation, storage and manipulation of objects for the purpose of generating images of those objects.

> *“ . . . the use of computers to produce pictorial images. The images produced can be printed documents or animated motion pictures, but the term computer graphics refers particularly to images displayed on a video display screen, or display monitor. ”*
>
> *Encyclopedia Britannica.*

Why Computer Graphics?

- Humans enjoy visual information.
- Visual information is easy to comprehend.
- Visual information is difficult to generate manually.

Photorealism

- What is a realistic image? What does it mean for a picture, whether painted, photographed, or computer-generated, to be “realistic”?
- Answer is subject to much scholarly debate!

Photorealism [Hall&Greenberg (1983)]

“Our goal in realistic image synthesis is to generate an image that evokes from the visual perception system a response indistinguishable from that evoked by the actual environment.”

- The term “photorealism” is normally used to refer to a picture that captures many of the effects of light interacting with real physical objects.
- It is an attempt to synthesize the field of light intensities that would be focused on the film plane of a camera aimed at the objects depicted.
- Physical properties of objects have to be taken into account!

Photorealism and Visualization

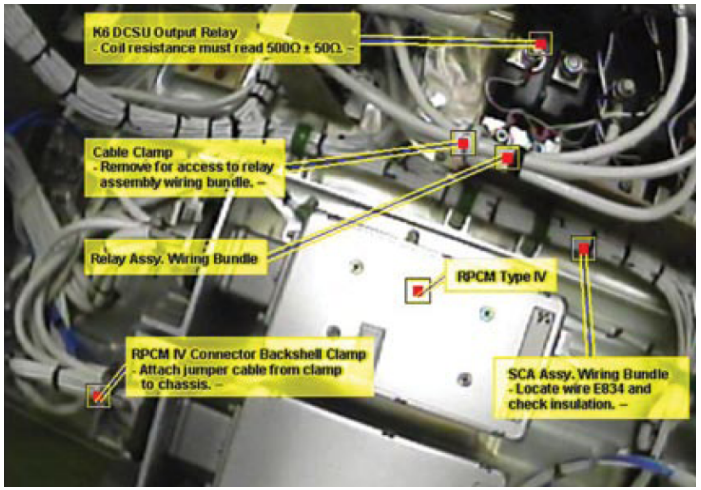
- There exist applications, however, where perfection is not such a mandatory feature. For instance, flight simulators need a fairly believable output, but need not to be perfect in every detail. The dominant challenge here is real-time interactive control.
- A more realistic picture is not necessarily a more desirable or useful one. E.g., when conveying information, a picture that is free of the complications of shadows and reflections may well be more successful than a tour de force of photorealism!
- In some applications reality is intentionally altered for esthetic effect or to fulfill a naïve viewer's expectation.
- Visualization is the art to produce images of objects that could not (or hardly) be seen otherwise. E.g., since they are too small, too abstract, too slow or too fast, or simply invisible for some other reason.
- Typical examples include weather forecast charts in meteorology, hearts, brains and bones of living creatures in medicine, temperature distributions on brakes, the growth of plants over years, geological changes like volcano eruptions or continental movements.

Applications of Computer Graphics

- Computer-Aided Design (CAD, CAM): One of the earliest applications!
  - Car parts, Boeing 777, submarine design.
  - City models, architectural walk-throughs.
  - Control of robots and manufacturing cells.
- Entertainment: Games, commercials, movies (e.g., Star Wars (1977–1983), Tron (1982), Star Trek IV: The Voyage Home (1986), Indiana Jones and the Last Crusade (1989), Terminator 2: Judgement Day (1991), Jurassic Park (1993), Toy Story (1995), Titanic (1997), Ants, A Bug's Life (1998), The Matrix (1999), Gladiator (2000), Lord of the Rings: The Fellowship of the Ring (2002), Troy (2004), Avatar (2009), Rise of the Planet of the Apes (2011), The Life of Pi (2012), Blade Runner 2049 (2017), Avatar: The Way of Water (2022), . . .).
- Education and training:
  - Flight simulation, pilot training.
  - Maintenance and assembly training.
  - Military training (digitized battlefields, mission rehearsal).
  - Telemedicine (3D models, minimal non-invasive surgery).
  - Sports training.

Applications of Computer Graphics: Augmented Reality

- The term “augmented reality” (AR) was coined around 1990 by Caudell and Mizell at Boeing.



[Image credit: The Boeing Company.]

- Sample augmented reality in today’s consumer products: Head-up displays in cars.
- Sample augmented reality in sports: Hawk-Eye.

Applications of Computer Graphics: Visualization

- Scientific visualization and data analysis:
  - Geographic information systems (maps, topographic maps).
  - Turbulence, temperature, stress, etc.
  - Weather models.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Applications of Computer Graphics: VFX vs. CGI vs. SFX

VFX stands for *Visual Effects*, which refers to the process of creating or manipulating imagery outside the context of a live-action shot.
VFX can include things like creating explosions, realistic computer-generated characters (like creatures or superheroes), adding weather effects (like rain or snow), or even altering backgrounds.

CGI stands for *Computer-Generated Imagery*. It refers specifically to any visual content that is created using computers, including 3D models, animations, and environments.
CGI is one tool used in the broader VFX process, but VFX encompasses more than just CGI, including live-action integration and compositing. Essentially, VFX covers all aspects of digitally altering or enhancing visual content, whether it involves CGI or other techniques.

SFX stands for *Special Effects*. Unlike VFX, SFX refers to practical, physical effects that are created on set or during filming. These are real-world effects that happen during production, such as explosions, weather effects (rain, smoke), prosthetics, animatronics, and mechanical effects like moving parts or models.

History of Computer Graphics: 1950s, 1960s, and 1970s

early 1950s US military used an interactive CRT graphics called SAGE.

1959 Computer drawing system DAC-1 by IBM and GM.

1961 Sketchpad developed by Ivan Sutherland at MIT.

1963 Douglas Englebart used a mouse as an input device.

1965 Jack Bresenham introduced his line-drawing algorithm.

1966 First computer-controlled head-mounted display (HMD) designed by Ivan Sutherland.

1971 Henri Gouraud developed Gouraud shading.

1972 Sutherland's students model and render Utah VW Bug.

1972 2D raster display for PC workstations at Xerox.

1973 First SIGGRAPH Conference. Roughly 600 attendees.

1974 Ed Catmull introduced texture mapping (and z-buffering).

1974 Bui-Tong Phong developed Phong shading.

1975 Benoit Mandelbrot published the paper "A Theory of Fractal Sets".

1977 Nintendo entered the graphics market.

History of Computer Graphics: 1980s

1980 "Vol Libre" (by Boeing's Loren Carpenter) shown at SIGGRAPH.

1980 Ray tracing developed by Turner Whitted.

1982 "Tron" produced by Disney; Perlin noise.

1982 Silicon Graphics founded by Jim Clark. Sun Microsystems, Autodesk, and Adobe Systems founded.

1982 AutoCAD developed by John Walker and Dan Drake.

1984 Radiosity method developed at Cornell University by Ben Battaile, Cindy Goral, Don Greenberg and Ken Torrance.

1985 Adobe System introduced Postscript.

1986 Pixar founded.

1988 Pixar Animation's Pat Hanrahan led the design of "RenderMan".

History of Computer Graphics: 1990s

1990 Autodesk introduced 3D Studio.

1991 “Terminator 2” was released.

1991 JPEG and MPEG standards were introduced.

1992 SGI specified OpenGL.

1992 Wavelets used for radiosity.

1993 A team of Pixar Animation won an Academy Scientific and Engineering Award for the development of “RenderMan”.

1993 A team of Industrial Light and Magic won an Academy Award for Best Visual Effects for its ground-breaking work on “Jurassic Park”.

1995 Pixar released “Toy Story”.

1997 SIGGRAPH’97 had 48 700 attendees.

1997 “Titanic” released.

1997 Ken Perlin won an Academy Award for Scientific and Technical Achievement for “Perlin noise”.

1998 “Ants” and “A Bug’s Life” released.

History of Computer Graphics: 2000 and Beyond

2001 Microsoft's "Xbox console" (based on NVIDIA graphics) makes debut.

2003 Graphics cards (NVIDIA, ATI, Matrox, . . .) become widely available.

2004 Graphics cards for mobile phones and PDAs.

2004 OpenGL Shading Language formally included into OpenGL 2.0.

2007 CUDA (Compute Unified Device Architecture) released by NVIDIA.

2008 OpenCL (Open Computing Language) specified by the Khronos Group.

2010 GPUs with native 64bit floating-point precision and support for massively-parallel computing become widely available.

2014 OpenGL 4.5 released.

2015 Vulkan introduced as "next generation OpenGL" at GDC 2015.

2016 Vulkan 1.0 released.

2017 OpenGL 4.6 released.

2020 Hardware-accelerated ray tracing on NVIDIA/AMD GPUs.

20?? Real-time radiosity rendering? Photo-realistic consumer graphics? Realistically rendered humans?

Foundations of Computer Graphics

- Computer Science:
  algorithms, data structures, programming, software engineering, architecture, artificial intelligence.
- Mathematics:
  linear algebra, analytical geometry, complex analysis, numerical analysis, differential geometry, topology, 3D modeling.
- Physics:
  optics, fluid dynamics, energy, kinematics and dynamics.
- Psychology:
  human light and color perception.
- Biology:
  human body, behavioral and cognitive systems, nervous system.
- Art:
  realism, esthetics.

Software Components for Graphics Systems

A system for 3D graphics consists of three major (software) parts:

- the modeler,
- the renderer,
- image handling and display.

Image Handling

- Image handling is often only a device driver to make the computed image visible for the user on a screen or on a hard copy device.
- It can also be an image processing system to improve the quality of images or to alter or transform them before displaying.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Modeling System

- Geometry-based modeling:
  - Lines, polygons, polyhedra,
  - Free-form curves and surfaces,
  - Quadtrees, octrees, bounding volumes,
- Physics-based modeling:
  - Kinematics and dynamics (contact detection, contact resolution, force calculation, natural gait),
  - Fluid dynamics (e.g., for modeling water and waves),
  - Gas, smoke, fire,
  - Deformable objects (e.g., clothes, cords),
  - Haptics (e.g., touch sensors).
- Cognitive-based modeling:
  - Domain knowledge, learning.
  - Interaction with real world.

Wide-spread simple modelers

- CAD systems,
- 3D editors,
- object description languages.

Device-Independent Graphics Primitives

- Since graphics output devices are many and diverse, it is imperative to achieve device independence.
- Thus, it is generally preferred to work in world coordinates rather than device coordinates.
- Typical graphics commands will be similar to the following commands:
  - DrawLine(x_1, y_1, x_2, y_2);
  - DrawCircle(x_1, y_1, r);
  - DrawPolygon(PointArray);
  - DrawText(x_1, y_1, "A Message");

  where x_1, y_1, x_2, y_2, r are specified in world coordinates.
- Graphics primitives have *attributes*, such as style, thickness and color for a line, or font, size and color for a text.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Application Programmer's Interface (API)

- Graphic APIs provide the programmer with procedures for handling menus, windows and, of course, graphics.
- Well-known APIs for 3D graphics:
  - OpenGL,
  - WebGL,
  - DirectX,
  - Java3D,
  - Vulkan.
- Vulkan offers a better CPU/GPU balance (than OpenGL) and supports multi-threaded programming, but it is considerably more low-level than OpenGL.
- Vulkan is expected to replace OpenGL on standard consumer GPUs within the next several years.

- In this course we will use OpenGL for practical work.

A Note on DirectX and Java3D

- DirectX:
  - Advantages:
    - More high-level functionality;
    - Better control of resources.
  - Disadvantages:
    - Only supported by MS Windows machines;
    - Lack of backwards compatibility of newer versions.
  - Fahrenheit was an attempt by Microsoft and SGI to unify OpenGL and Direct3D/DirectX in the 1990s, but it got cancelled.
- Java3D:
  - Advantages:
    - Based on true object-oriented approach (“scene graph”).
    - Ties natively into Java.
    - Open-source code since 2004.
  - Disadvantages:
    - Runs atop of Java OpenGL (JOGL); delay in use of new GPU features.
    - Pause in development during 2003 and 2004.
    - Community project since 2004 with unclear future; JavaFX seems to have taken over.

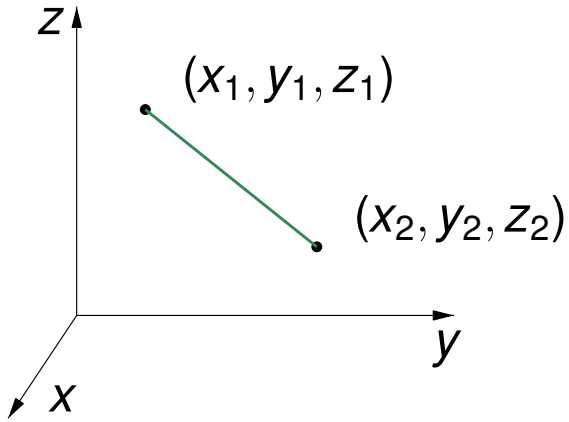
2 Representation and Modeling

- Primitive Objects
- Regions in 2D
- Curved Surfaces in 3D
- Solids in 3D
- Miscellaneous Modeling Schemes

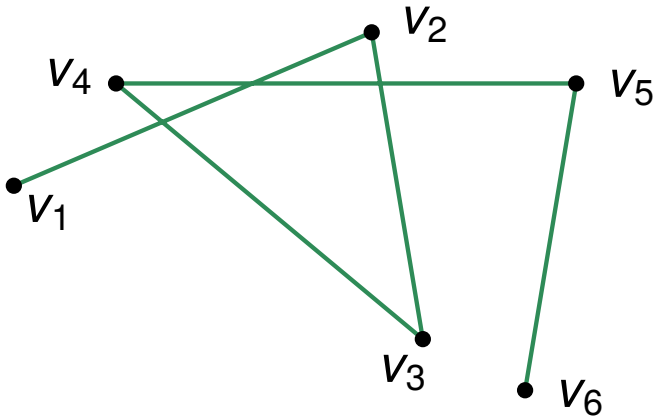
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

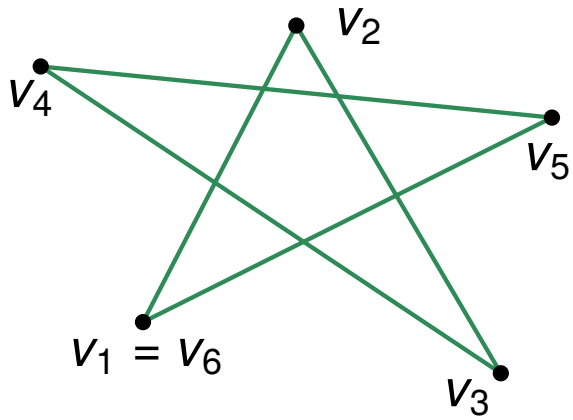
Line Segments

- A *line segment* can be specified by its two endpoints.



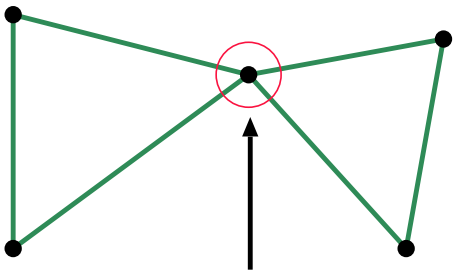
- A *polygonal curve* (or *polygonal chain*), Dt.: Polygonzug, is a sequence of finitely many vertices $v_1, v_2, \ldots, v_n$ connected by line segments such that each segment (except for the first) starts at the end of the previous segment.



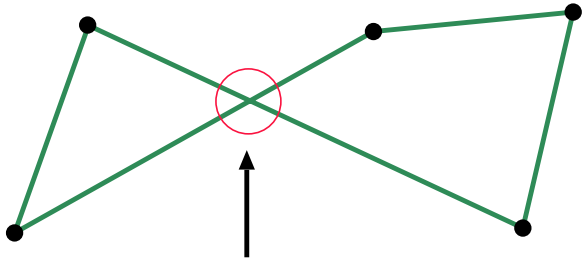


Polygons

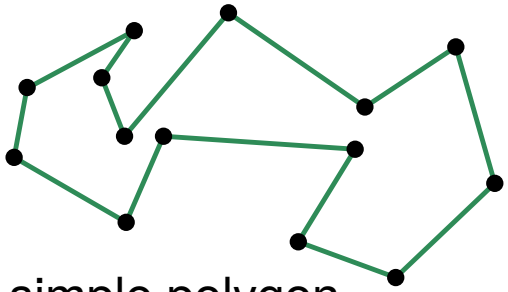
- A *polygon* is a closed polygonal curve where every vertex belongs to exactly two segments. (We will always assume that all vertices of a polygon lie in one plane.)



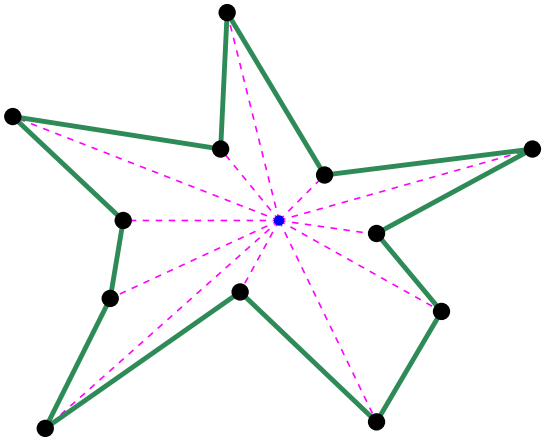
no polygon



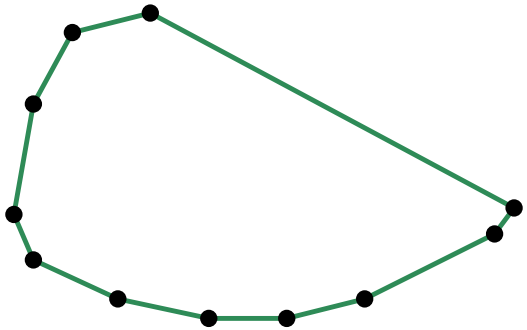
not a simple polygon



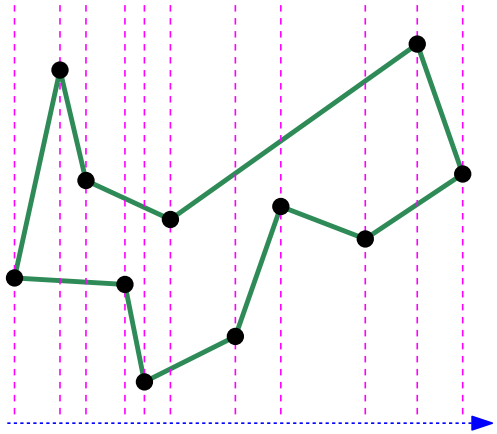
simple polygon



star-shaped polygon



convex polygon

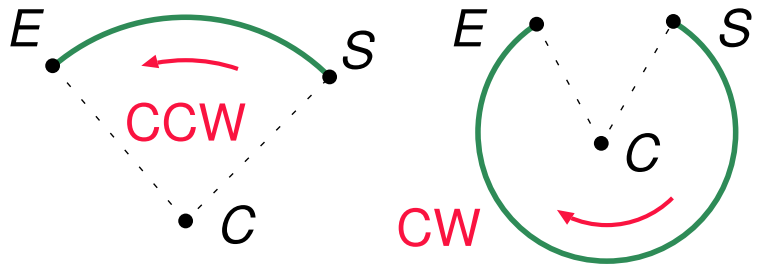


x-monotone polygon

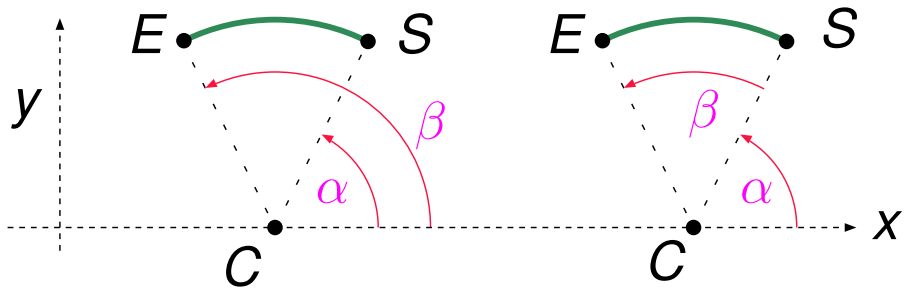
Circular Arcs

Circular arcs can be represented in several ways. (And none of them is universally good!)

- Center C, start point S, end point E, orientation (CW or CCW): redundancy problem!

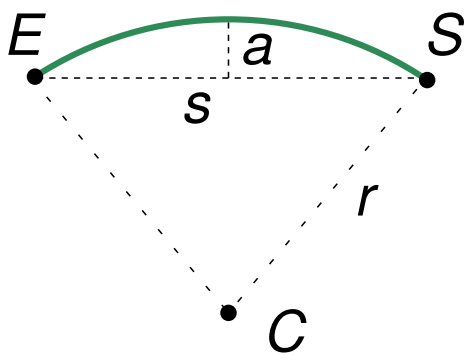


- Center C, radius r, start angle α, end angle β, orientation: start and end unknown, potential numerical problems!



Circular Arcs

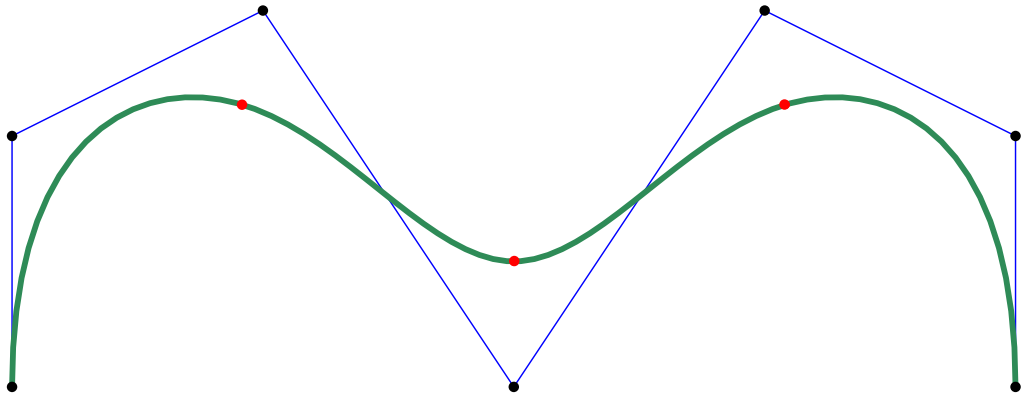
- Start point S, end point E, $\frac{a}{s}$, as suggested by Sabin.
- No redundancy and numerically reliable (unless start point and end point (nearly) coincide).
- But center and radius are unknown (and difficult to compute)!
- In theory, a line segment can be treated as a degenerate arc, but a full circle cannot be represented.
- Known as *bulge factor* in the DXF file format.



$a > 0$: CCW
$a < 0$: CW

General Curves

- In addition to lines, circular arcs, and quadrics, more general types of curves are used in CAD systems.
- Well-known representatives of so-called *free-form curves* include
  - Bézier curves,
  - B-splines,
  - NURBS.



uniform clamped cubic B-spline

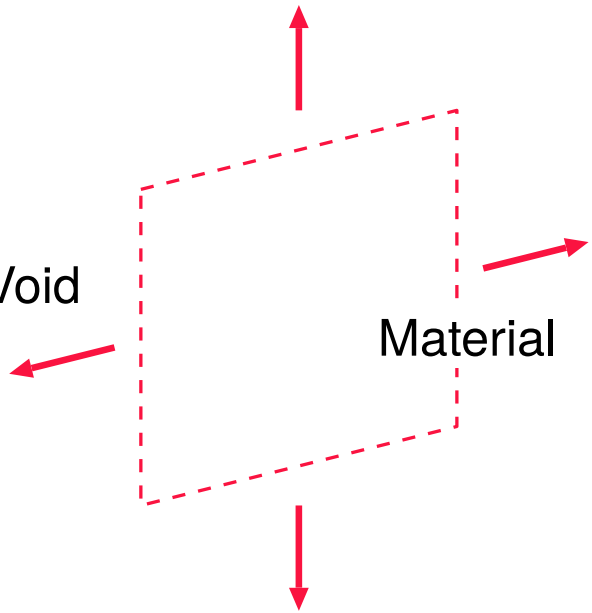
- We will not discuss free-form curves in this lecture — see my lecture on geometric modelling.

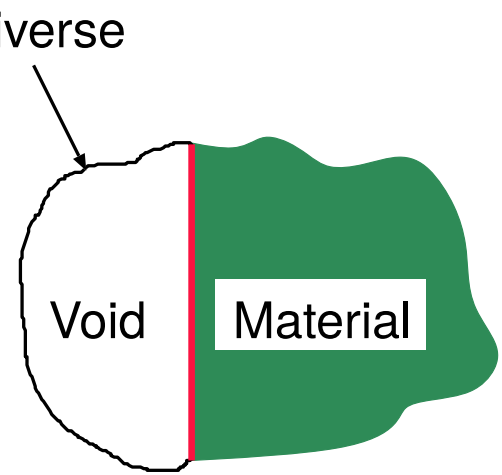
Half-Space

- The *closed half-space* defined by a point p in 3D and a unit vector n is given by

$$H(p,n) := \{u \in \mathbb{R}^3 : \langle n,u\rangle - \langle n,p\rangle \leq 0\}.$$

If $\langle n,u\rangle - \langle n,p\rangle = 0$ then $u \in \varepsilon(p,n) := \{u \in \mathbb{R}^3 : \langle n,u\rangle - \langle n,p\rangle = 0\}$,
$\langle n,u\rangle - \langle n,p\rangle > 0$ then u in the half-space , into which n points,
$\langle n,u\rangle - \langle n,p\rangle < 0$ then u in the half-space , into which n does not point.





Sphere

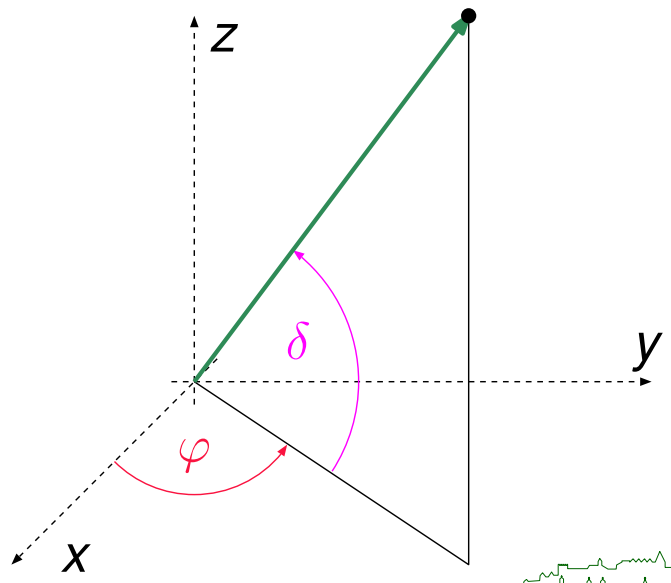
- A sphere is specified by:
  - center (x_c, y_c, z_c),
  - radius r.
- Implicit representation:

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2.$$

- Parametrization:

$$(x_c + r\cos\delta\cos\varphi, y_c + r\cos\delta\sin\varphi, z_c + r\sin\delta),$$

with $\varphi \in [0, 2\pi[$, $\delta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

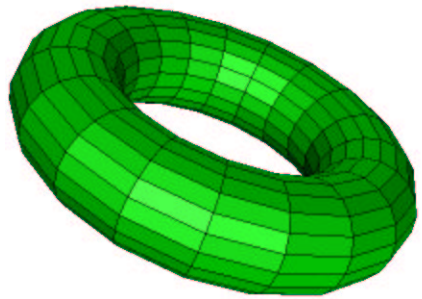


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

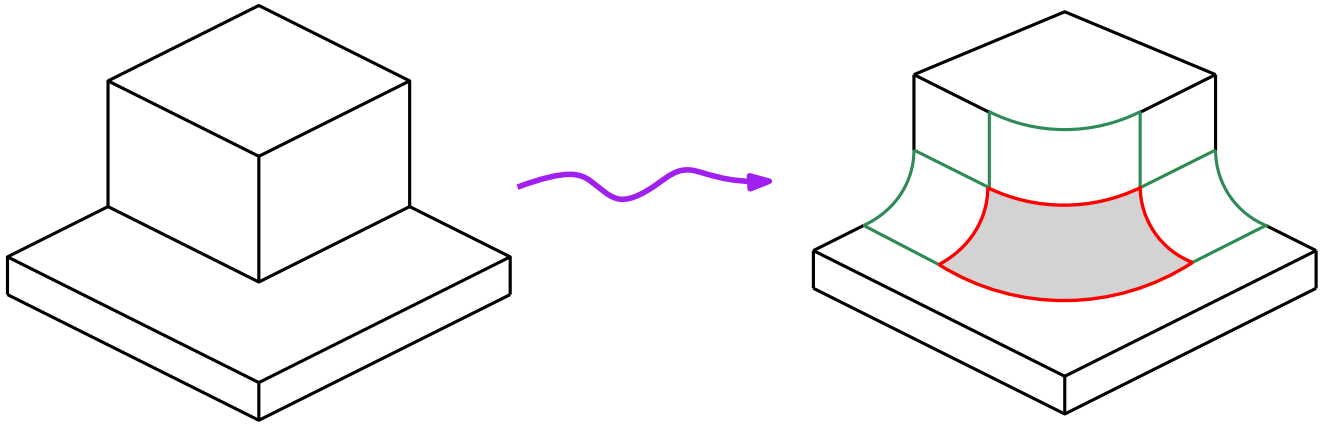
Torus

- A torus is specified by:
  - a center point (x_c, y_c, z_c) on the axis of rotation (parallel to the z-axis).
  - radii R and r.
- Implicit representation:

$$(\sqrt{(x-x_c)^2+(y-y_c)^2}-R)^2+(z-z_c)^2=r^2.$$



- Surface blending may generate portions of the surface of a torus.

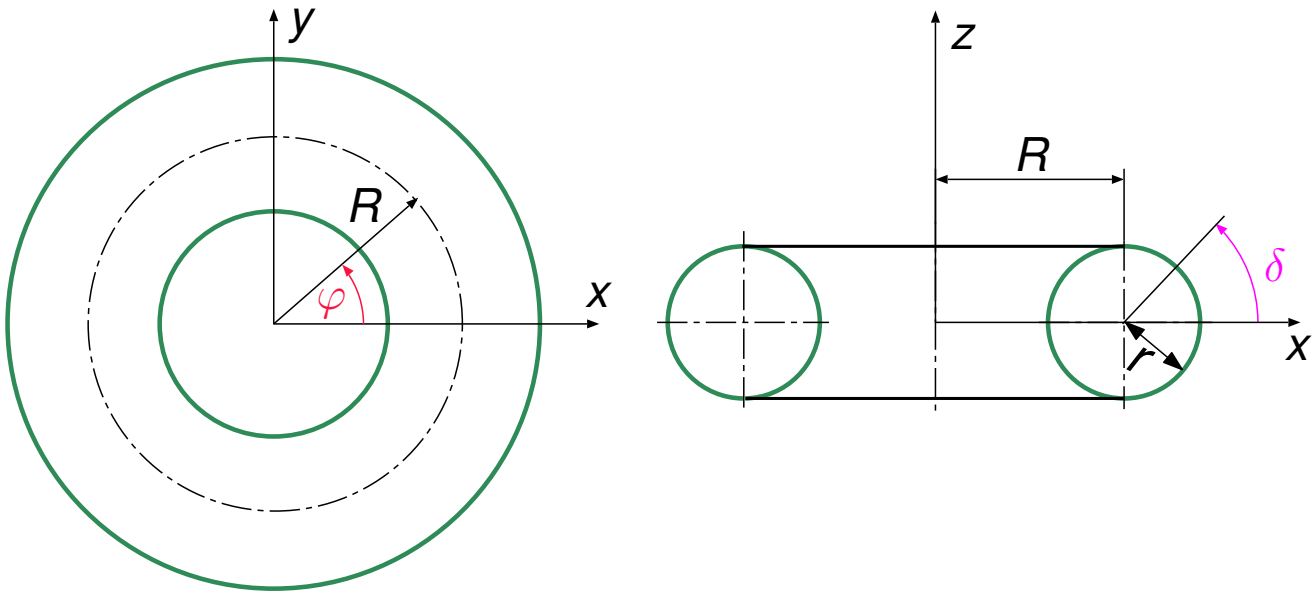


Parametrization of a Torus

- Parametrization:

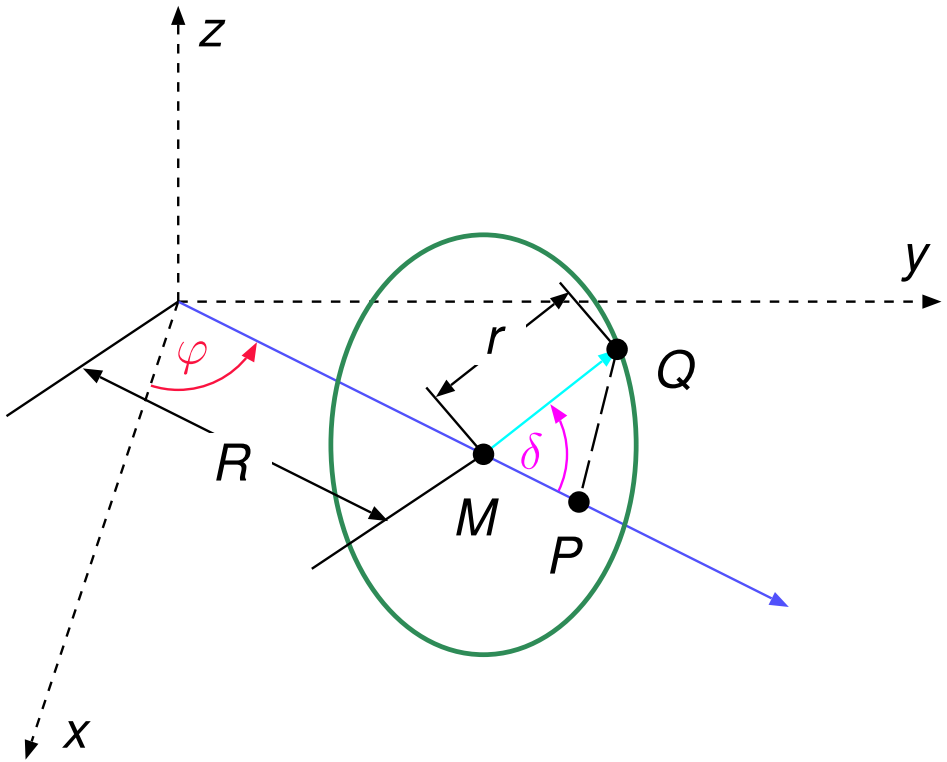
$$((R + r\cos\delta)\cos\varphi, (R + r\cos\delta)\sin\varphi, r\sin\delta),$$

with $\varphi \in [0, 2\pi[$, $\delta \in [0, 2\pi[$.



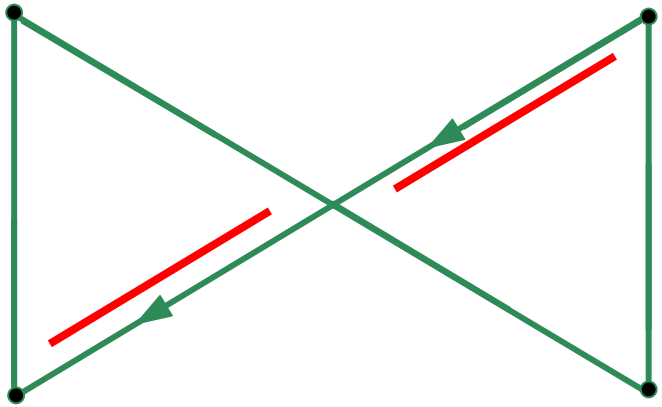
Parametrization of a Torus

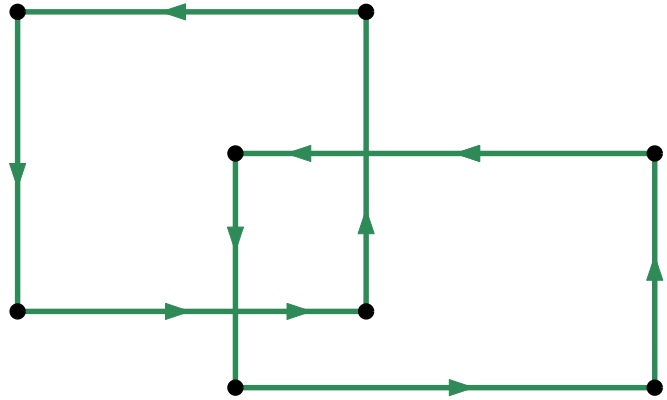
$$
\begin{aligned}
m &= (R\cos\varphi, R\sin\varphi, 0) \\
p &= m + r\cos\delta(\cos\varphi, \sin\varphi, 0) \\
&= (m_x + r\cos\delta\cos\varphi, m_y + r\cos\delta\sin\varphi, 0) \\
q &= (p_x, p_y, r\sin\delta) \\
&= (R\cos\varphi + r\cos\delta\cos\varphi, R\sin\varphi + r\cos\delta\sin\varphi, r\sin\delta) \\
&= ((R + r\cos\delta)\cos\varphi, (R + r\cos\delta)\sin\varphi, r\sin\delta)
\end{aligned}
$$



Exact Representation of the Boundary

- When representing a planar region by its boundary curves the key issue is to be able to extract its interior unambiguously.





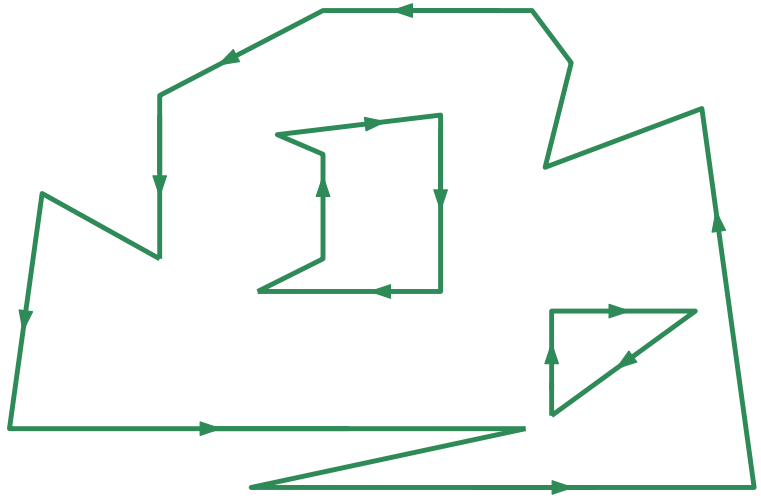
Warning

Different interpretations of “interior” are in practical use for the regions depicted above!

- Even-odd rule.
- Winding number.

Exact Representation of the Boundary

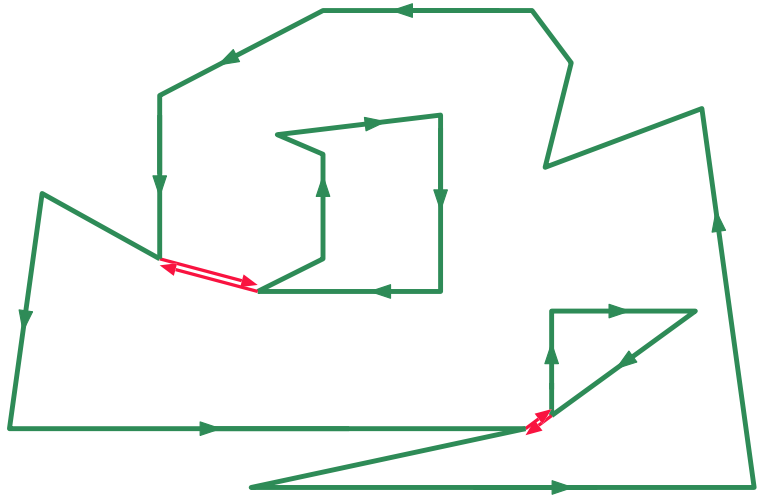
- The boundary curves of a planar region $\mathcal{R}$ should meet the following conditions:
  - all curves are simple and closed,
  - one of them is the “outer” boundary,
  - all other boundary curves (“islands” or “holes”) lie strictly in the interior region of the outer boundary,
  - the island curves (and their interior regions) are pairwise disjoint,
  - all curves are oriented such that $\mathcal{R}$ lies on the same side of every curve.
- In mathematical terms, a collection of curves that meets these conditions bounds a *multiply-connected* region.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Bridge Edges

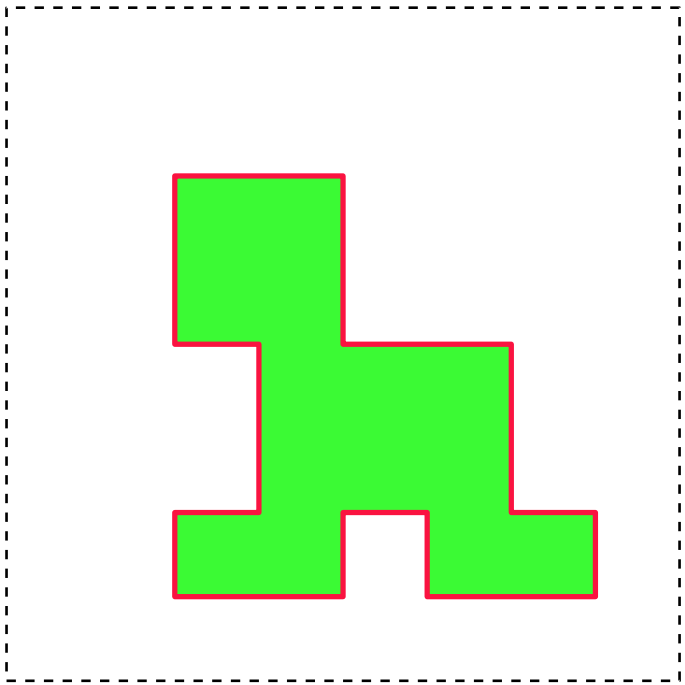
- We may find it convenient to transform a multiply-connected region into a simply-connected region by means of zero-width bridges.
- Note that the resulting curve is not a simple polygon in the strict meaning of our original definition!

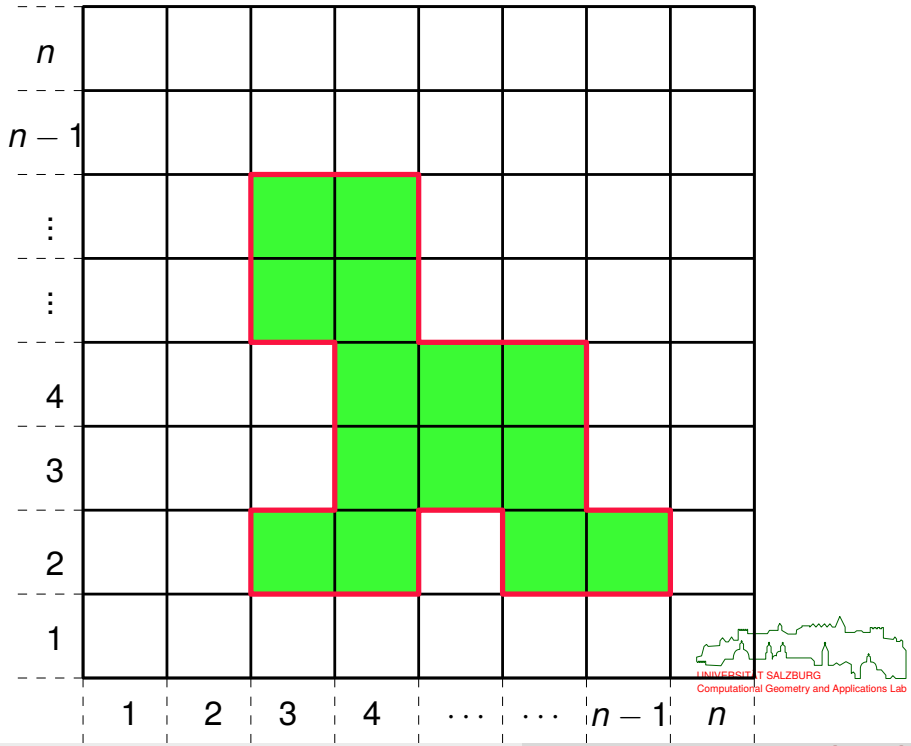


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Cell Decomposition in 2D

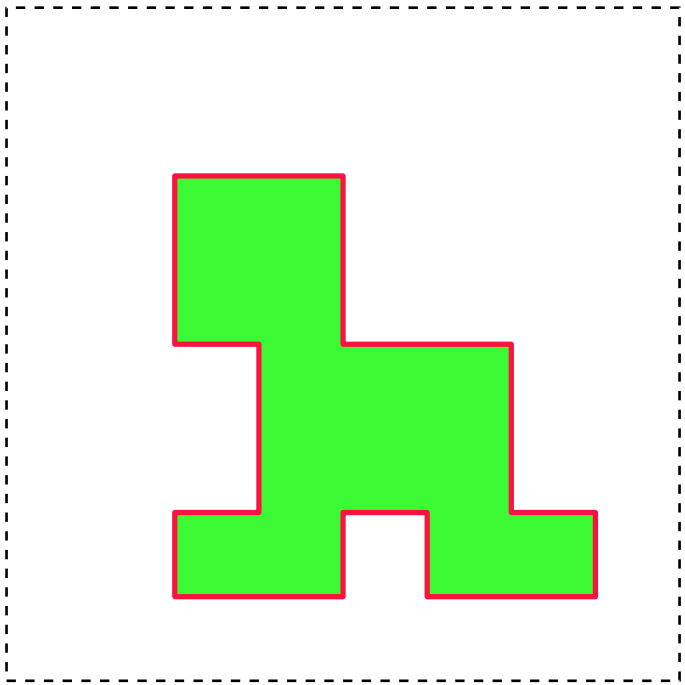
- A *cell decomposition* provides an approximate representation of a region $\mathcal{R}$.
- A user-defined subset of the plane ("workspace") is overlayed with a regular grid.
- Every cell is classified as full, empty, or partially full depending on whether it lies completely in the interior or exterior of $\mathcal{R}$, or whether it intersects the border of $\mathcal{R}$.
- The region is modeled as the union of those cells that are classified as full.
- Whether or not the cells that are partially full are added to the approximate representation depends on the application.

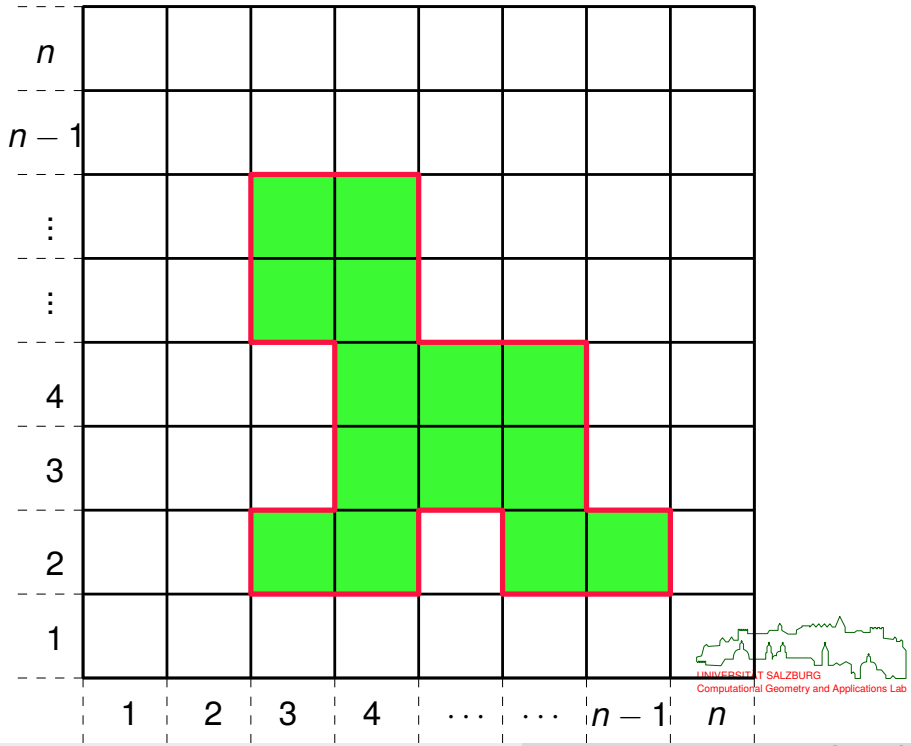




Cell Decomposition in 2D

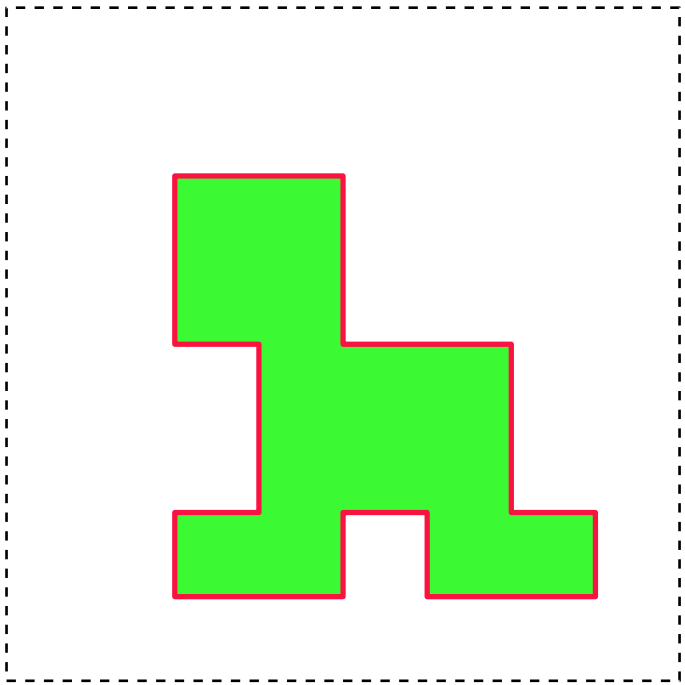
- There is an obvious trade-off between modeling accuracy and memory consumption.
- The high memory consumption tends to be a serious problem unless a very coarse approximation suffices: for an $n \times n$ grid, the number of cells increases by a factor of four if the resolution of the grid is doubled!

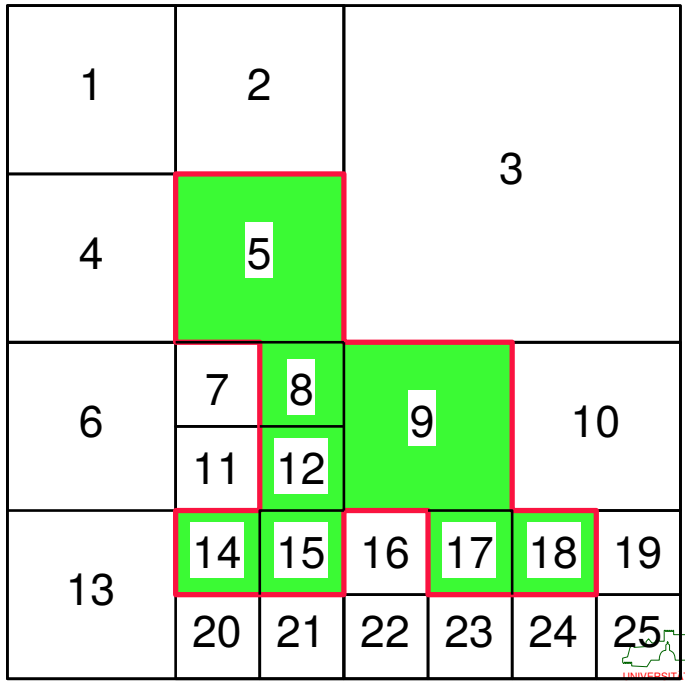




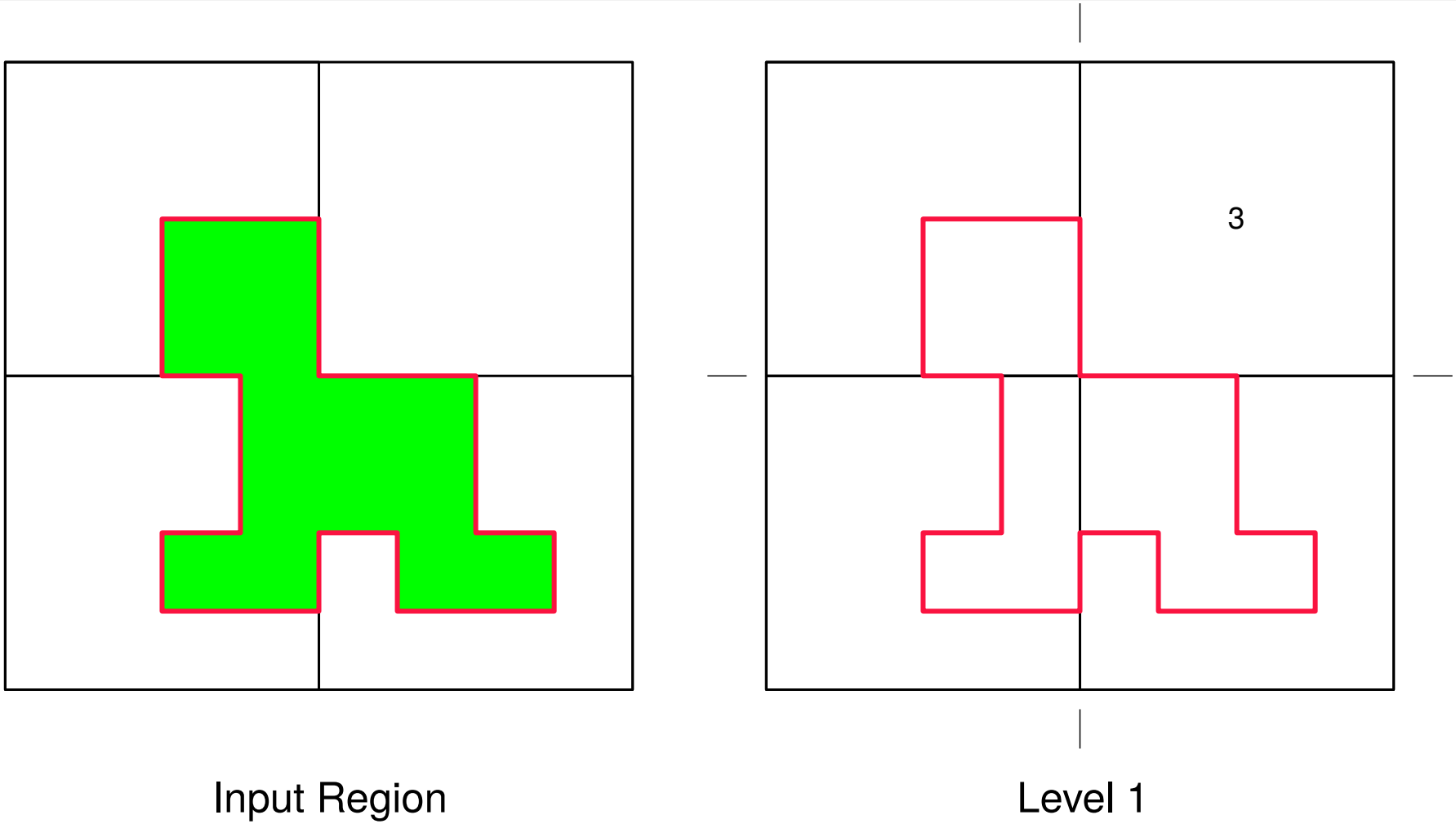
Quadtree

- Goal: Model the boundary of a region $\mathcal{R}$ with sufficient detail, but use larger cells within the interior of $\mathcal{R}$.
- We subdivide the rectangular workspace recursively into four sub-rectangles by bisecting it in both x and y.
- Again, a cell is classified as *full*, *empty*, or *partially full*.
- The recursive subdivision of cells that are partially full continues until a minimum resolution or maximum depth of the quadtree is reached.

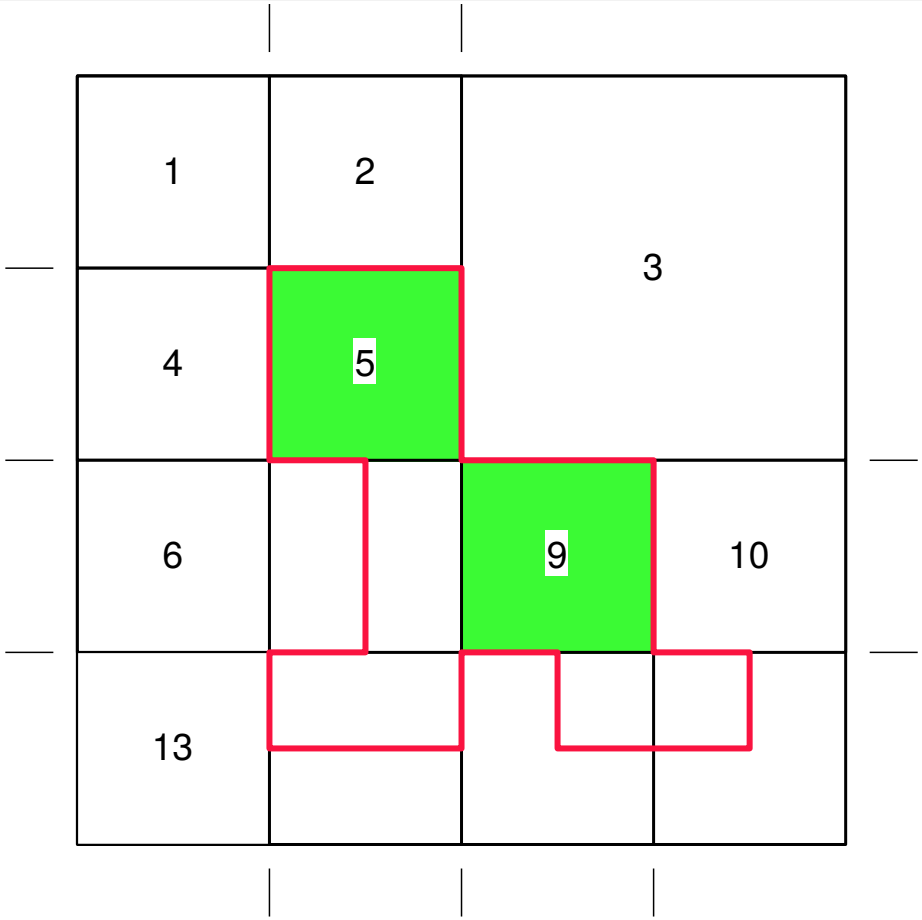




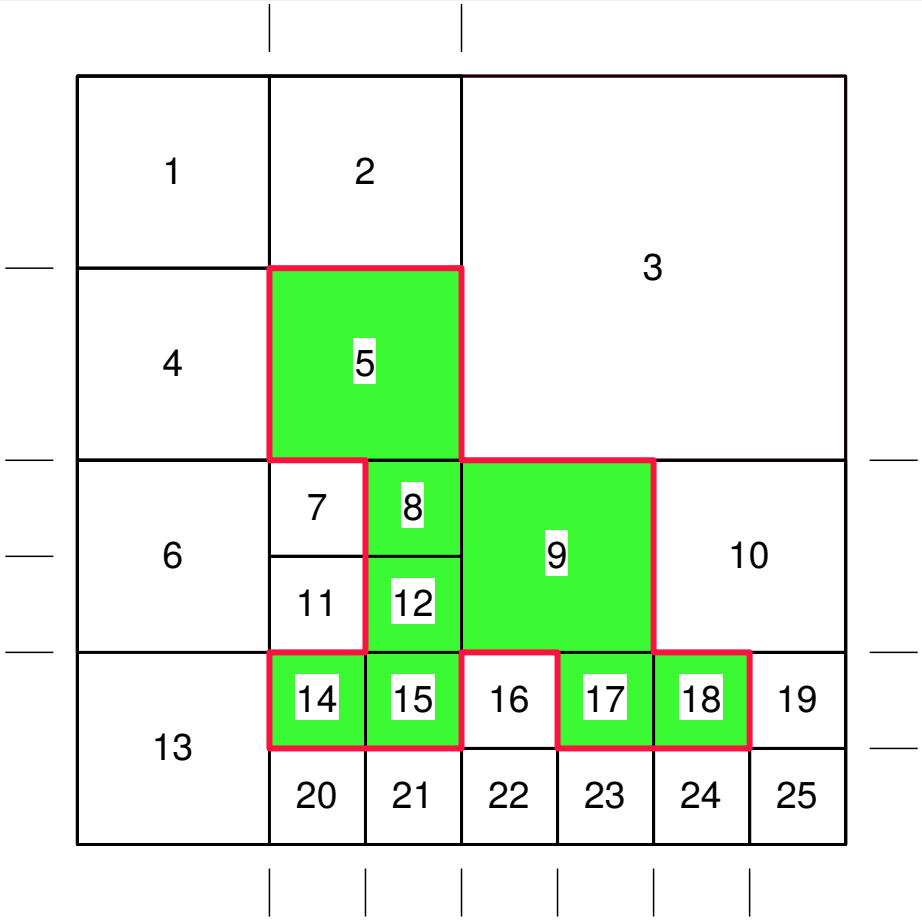
Quadtree: Construction



Quadtree: Construction



Level 1+2



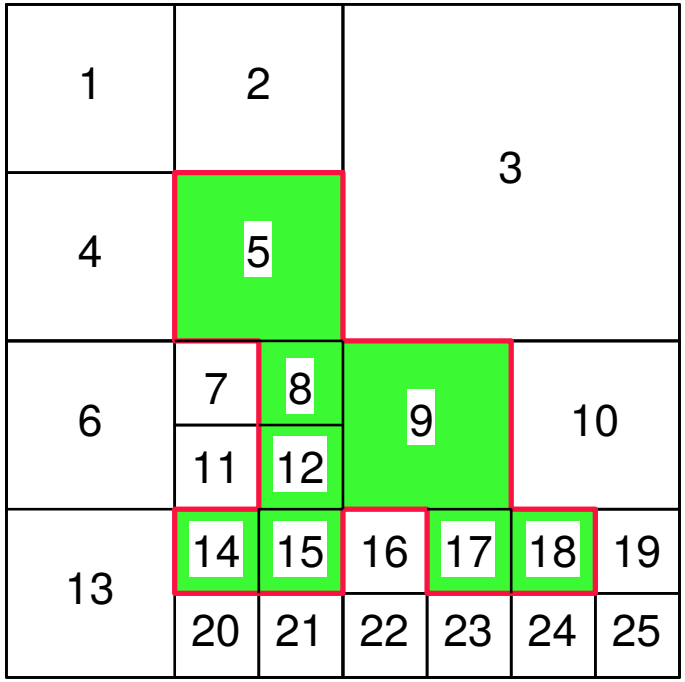
Level 1+2+3

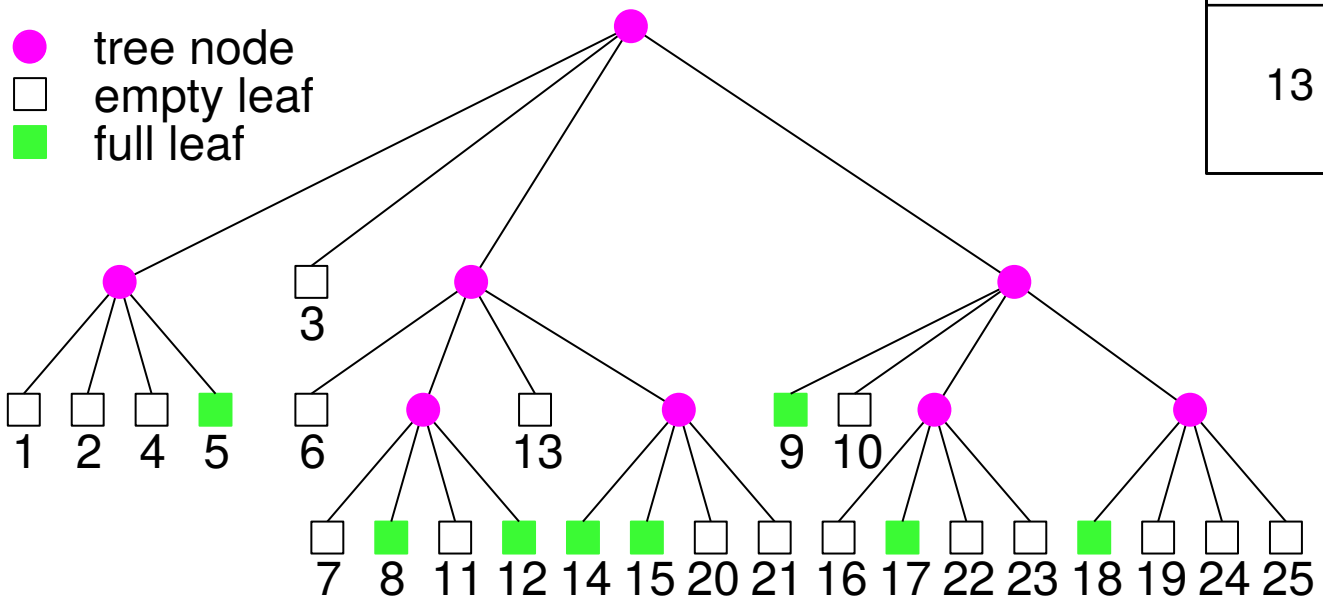
full

empty

Quadtree: Tree Structure

- It is natural to store a quadtree as a tree.

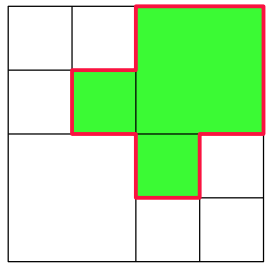




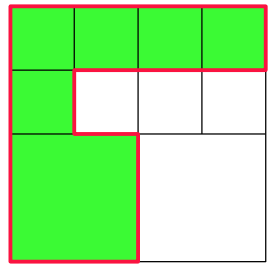
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

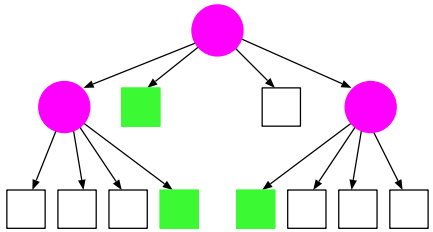
Quadtree: Boolean Operations

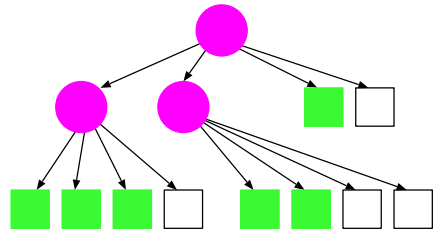
S



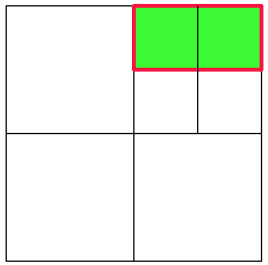
T



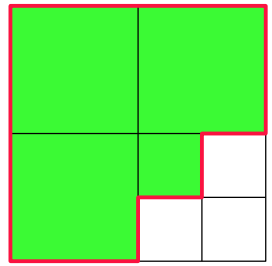


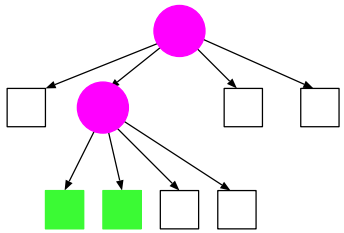


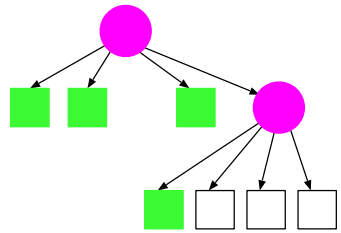
Int(S,T)



Union(S,T)



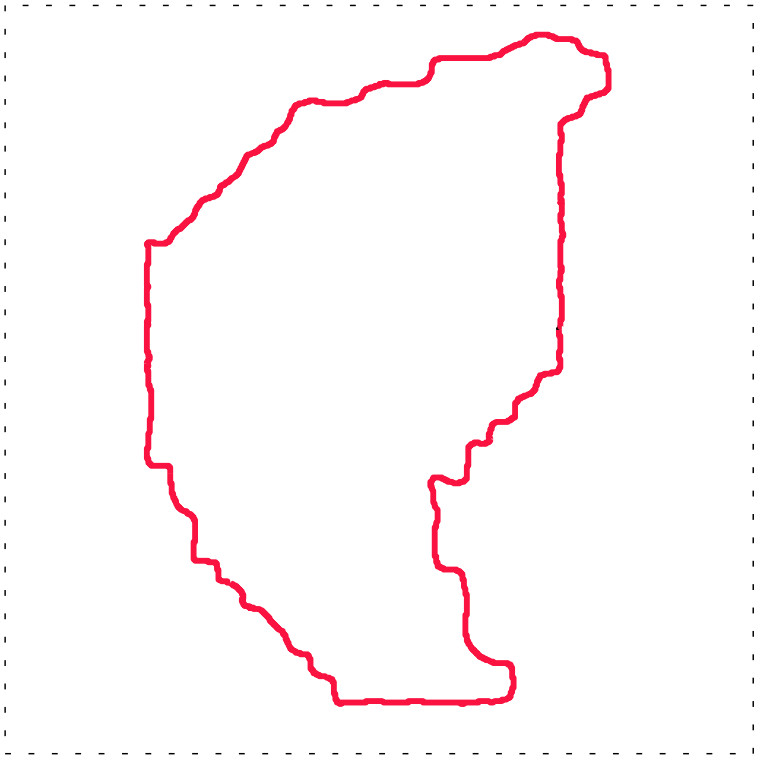


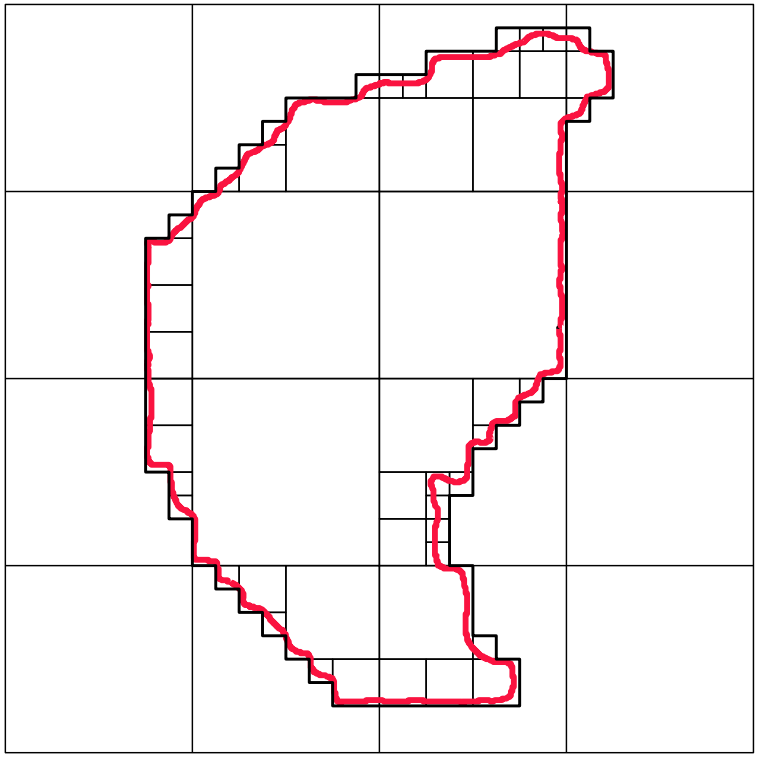


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Quadtree for Curved Data

- It is natural to extend quadtree representations to regions with curved boundaries.





Warning

A recursive quadtree decomposition will, in general, never terminate unless a minimum cell size is specified.

Quadtree: Pros and Cons

Pros:

- Standard advantages of hierarchical modeling, such as a fast test for disjointness.
- Boolean operations are easy to compute (provided that the quadtrees are aligned).
- Point-in-region test is straightforward.

Cons:

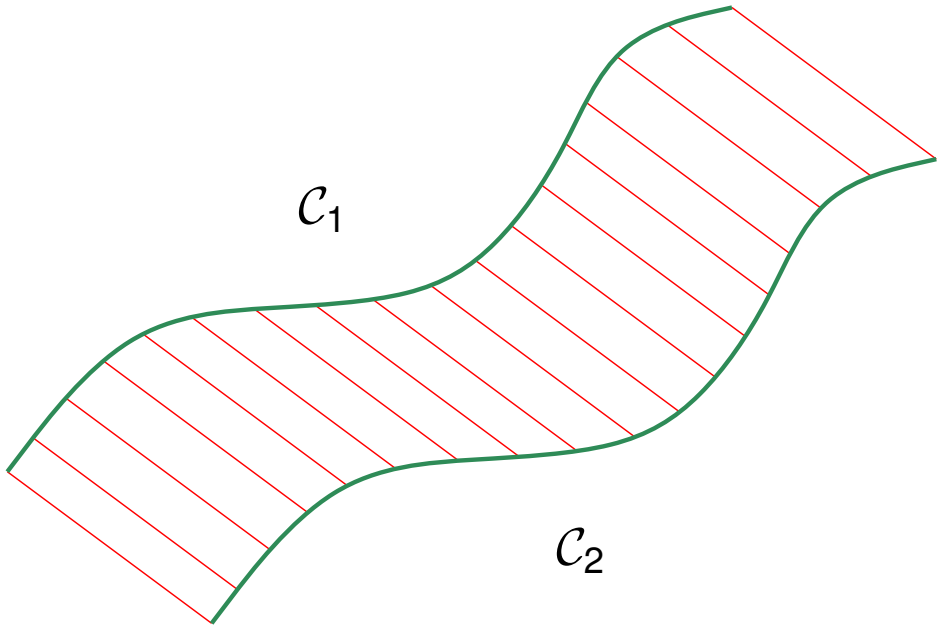
- The representation is coordinate-dependent and not invariant under affine transformations!
- The representation is only approximate, and memory may become an issue.
- A suitable approximation accuracy may be hard to predict.
- Graphical “zooming in” is only supported until the representation accuracy is reached.
- Neighbor finding is tricky.

Ruled Surface

- Consider two curves $\mathcal{C}_1, \mathcal{C}_2 \colon [\alpha, \beta] \to \mathbb{R}^3$, for $\alpha, \beta \in \mathbb{R}$ with $\alpha \leq \beta$.
- The *ruled surface (Dt.: Regelfläche)* $\mathcal{S} \colon [\alpha, \beta] \times [0, 1] \to \mathbb{R}^3$ defined by $\mathcal{C}_1, \mathcal{C}_2$ is given by the linear interpolation of $\mathcal{C}_1$ and $\mathcal{C}_2$:

$$\mathcal{S}(s, t) := (1 - t)\mathcal{C}_1(s) + t\mathcal{C}_2(s) \quad \text{with } s \in [\alpha, \beta], t \in [0, 1].$$

- Note that $\mathcal{S}$ may be curved even if $\mathcal{C}_1, \mathcal{C}_2$ are line segments!
- A ruled surface is an example for procedural modeling.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Ruled Surface

- Example:

$$\mathcal{C}_1(s) := \begin{pmatrix}0\\0\\1\end{pmatrix} + s\begin{pmatrix}1\\0\\0\end{pmatrix},\ \mathcal{C}_2(s) := \begin{pmatrix}0\\1\\1\end{pmatrix} + s\begin{pmatrix}1\\0\\0\end{pmatrix},\quad \text{with } s \in [0,1].$$

- We get the ruled surface $\mathcal{S}\colon [0,1]\times[0,1] \to \mathbb{R}^3$

$$\begin{aligned}\mathcal{S}(s,t) &= (1-t)\left[\begin{pmatrix}0\\0\\1\end{pmatrix} + s\begin{pmatrix}1\\0\\0\end{pmatrix}\right] + t\left[\begin{pmatrix}0\\1\\1\end{pmatrix} + s\begin{pmatrix}1\\0\\0\end{pmatrix}\right]\\ &= \begin{pmatrix}0\\0\\1\end{pmatrix} + s\begin{pmatrix}1\\0\\0\end{pmatrix} + t\begin{pmatrix}0\\1\\0\end{pmatrix},\end{aligned}$$

  i.e., the “top” of the unit cube.

Surface of Revolution

- Consider the 3D curve $\mathcal{C}(s) := \begin{pmatrix} \mathcal{C}_1(s) \\ 0 \\ \mathcal{C}_2(s) \end{pmatrix}$ parameterized by $\mathcal{C}_1, \mathcal{C}_2 : [\alpha, \beta] \to \mathbb{R}$.
- Obviously, $\mathcal{C}$ is constrained to the *xz*-plane of $\mathbb{R}^3$.
- A rotation of $\mathcal{C}$ about the *z*-axis yields the *surface of revolution (Dt.: Rotationsfläche)*

$$\mathcal{S}(s, \varphi) := \begin{pmatrix} \mathcal{C}_1(s) \cdot \cos\varphi \\ \mathcal{C}_1(s) \cdot \sin\varphi \\ \mathcal{C}_2(s) \end{pmatrix}, \qquad \text{where } s \in [\alpha, \beta], \varphi \in [0, 2\pi[.$$

- Properties:
  - Every point of $\mathcal{C}$ which does not lie on the *z*-axis creates a circle in a plane parallel to the *xy*-plane;
  - A line segment which is parallel to the *xy*-plane creates a disk or circular annulus;
  - A line segment which is parallel to the *z*-axis creates a cylinder;
  - Any other line segment creates a cone;
  - A circular arc that is part of $\mathcal{C}$ creates a portion of a torus.

Surface of Revolution

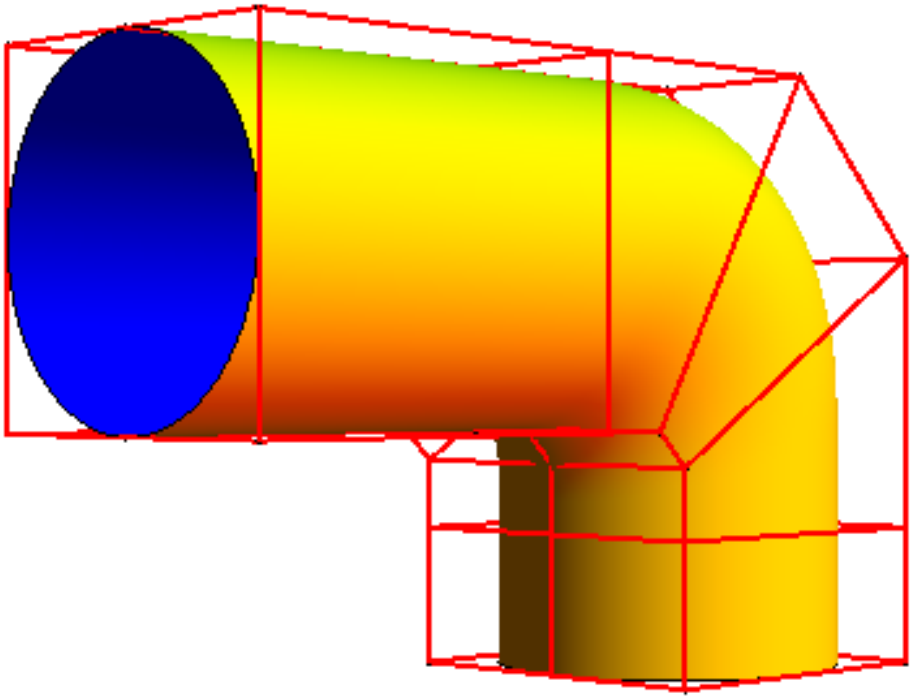
- For $s \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, let $\mathcal{C}_1(s) := \cos s$ and $\mathcal{C}_2(s) := \sin s$.
- This yields

$$\mathcal{S}(s, \varphi) = \begin{pmatrix} \cos s \cdot \cos \varphi \\ \cos s \cdot \sin \varphi \\ \sin s \end{pmatrix} \quad \text{with } \varphi \in [0, 2\pi] \text{ and } s \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$

as surface of revolution, i.e., the surface of the unit sphere.

Free-Form Surfaces

- In addition to quadratic surfaces, ruled surfaces, or surfaces of revolution, more general types of surfaces are used in CAD systems and for CGI.
- Well-known representatives of so-called *free-form surfaces* include
  - Bézier surfaces,
  - B-spline and NURBS surfaces,
  - subdivision surfaces.
- See my lecture on geometric modeling for an introduction to free-form modeling.



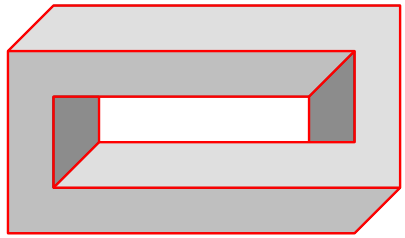
Free-Form Surfaces: Utah Teapot

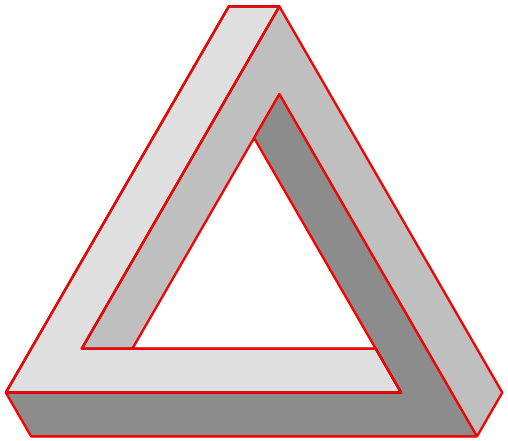
- The Utah teapot was designed in 1974 by Martin Newell at the Univ. of Utah.
- It is a hand-crafted Bézier model of a “Haushaltsteekanne” (“household teapot”) sold by Friesland Porzellan, at that time part of the German Melitta group.
- It has become one of the most iconic models. See, e.g., the ”The Six Platonic Solids” by Arvo&Kirk (1987), showcasing “the newly discovered Teapotahedron”.

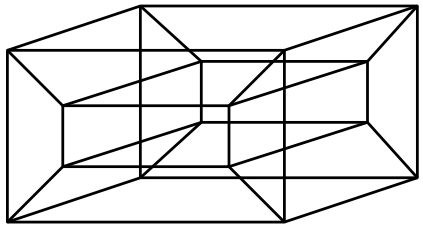
[Image credit: https://en.wikipedia.org/wiki/Utah_teapot]

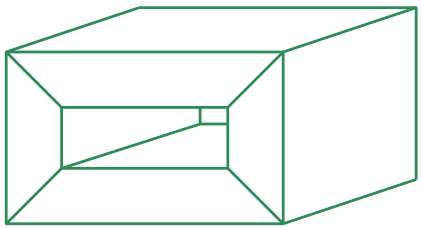
Wire-frame Model

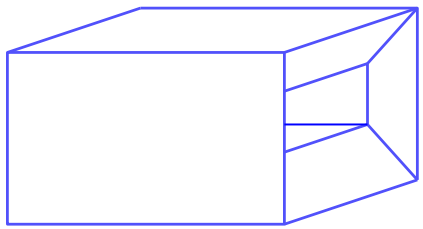
- Wire-frame models “represent” solids by specifying the set of edges of the solid.
- Outdated nowadays — mentioned for historical reasons only!
- It is just too easy to model nonsense objects . . .

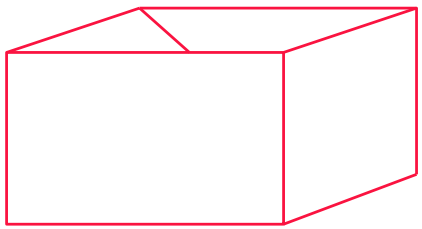










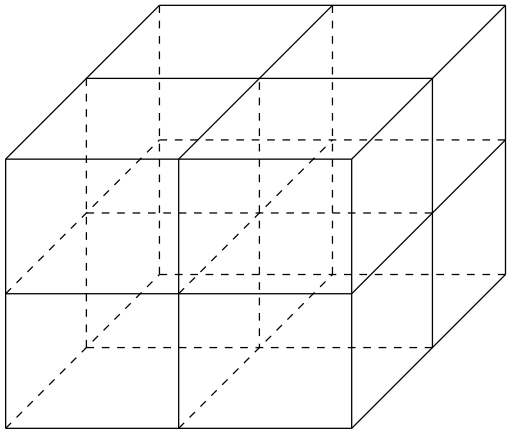


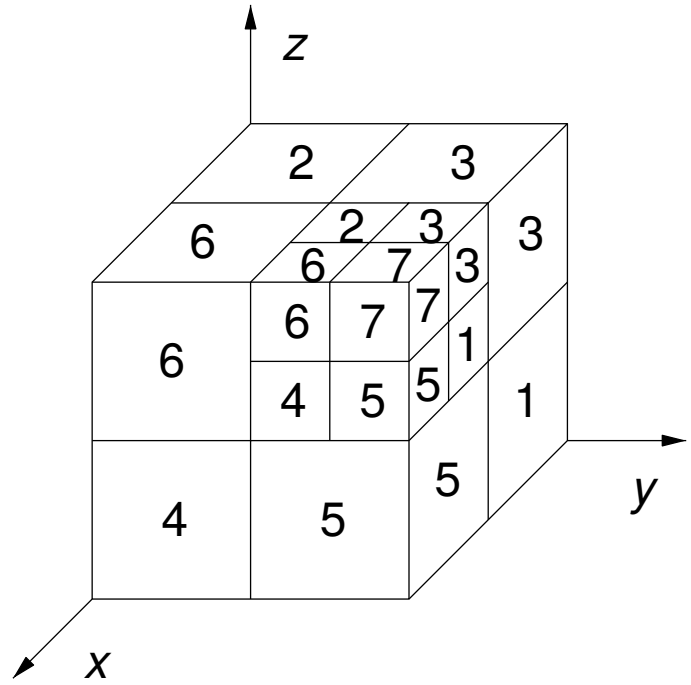
Spatial Decomposition

- Divide the space into *cells*, aka *voxels*.
- Often, the collection of cells forms a regular grid.
- Represent all cells lying in the object.
- Popular representation in volume rendering.
- High storage requirement.
- Similar pros and cons as in 2D.

Octree

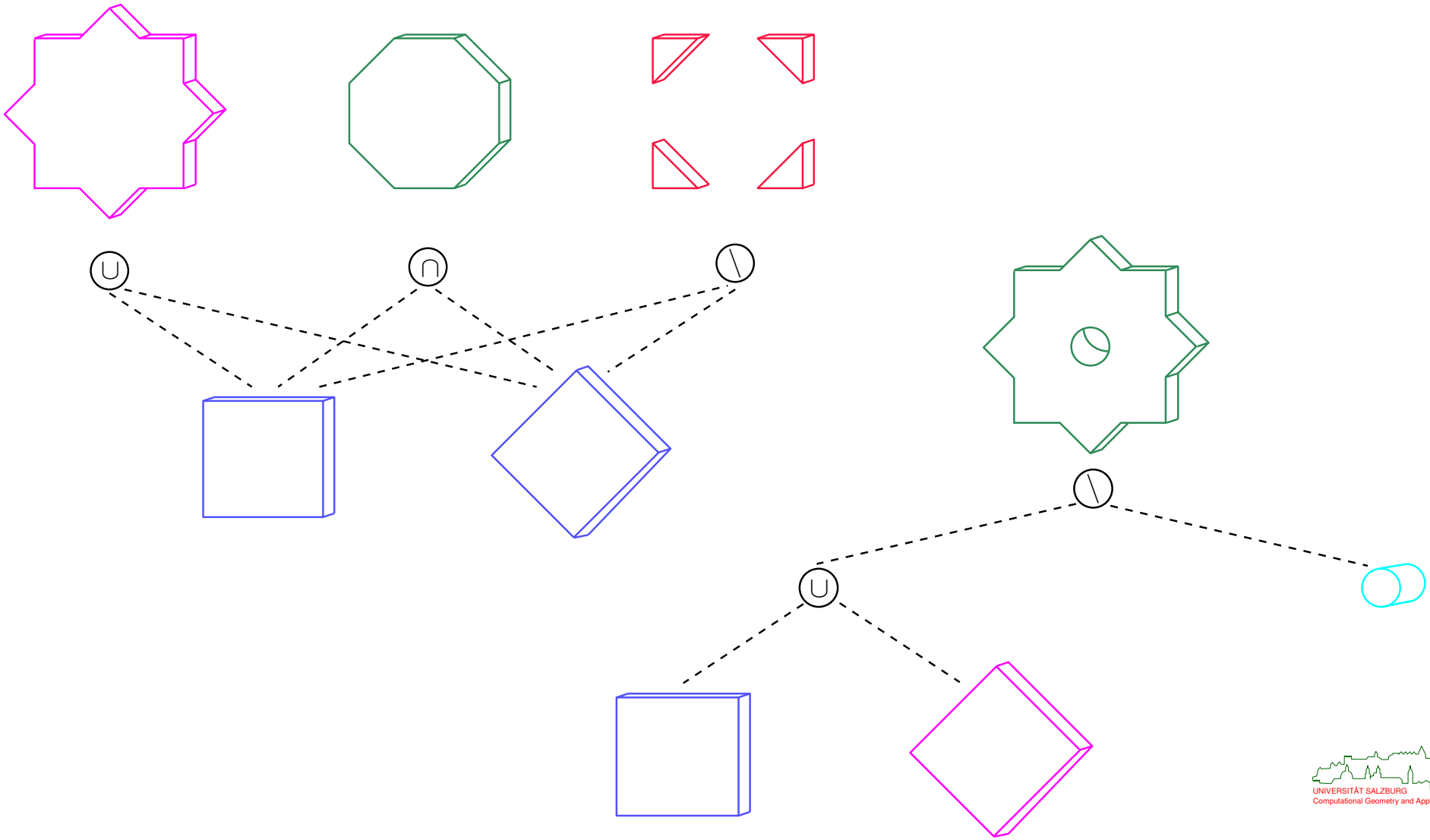
- Hierarchical representation.
- Requires much less space than a standard spatial decomposition.
- Extension of 2D *quadtree*.
- Each cube is divided into eight octants.





- Useful for many operations, e.g., collision detection, ray tracing.
- Similar pros and cons as quadtrees.

- Constructive Solid Geometry (CSG) combines simple solids — so-called primitives — by using Boolean operations.



CSG Tree

- CSG models are commonly used to describe man-made shapes.
- Sample primitives include
  - half-space,
  - spherical ball,
  - cylinder,
  - cone,
  - pyramid,
  - cube,
  - box,
  - ellipsoid.
- A CSG object is stored as a tree with *operators* at interior nodes, and the primitives at the leaves.
- Every interior node stores the position and orientation of its children, and the Boolean operation to be applied to them.
- Edges of the tree are ordered.

CSG and Boolean Set Operations

- CSG combines solid objects by using three or sometimes four different Boolean operations:

  Union: Create a new solid that is the union of two solids; denoted by $\cup$ or $+$.

  Intersection: Create a new solid that is the intersection of two solids; denoted by $\cap$ or $\star$.

  Difference: Create a solid by subtracting one solid from another solid; denoted by $\setminus$ or $-$.

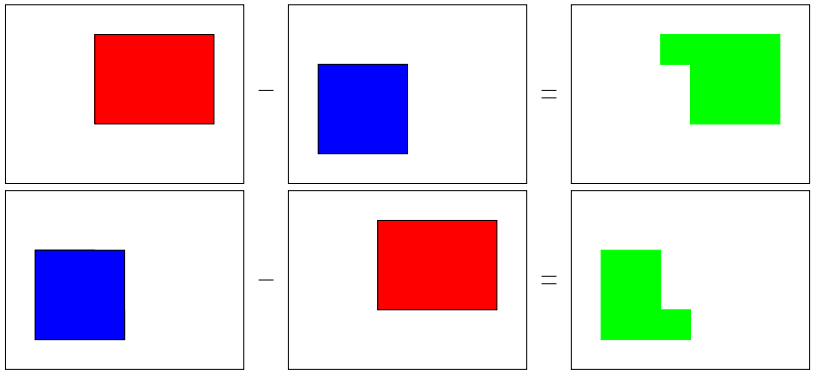
  Complement: Create a new solid by subtracting a solid from the universe.

- In theory, the set-theoretic difference can be replaced by a complement and intersection operation.
- In practice, the difference is often more intuitive as it corresponds to removing a solid volume.

CSG Representation Caveats

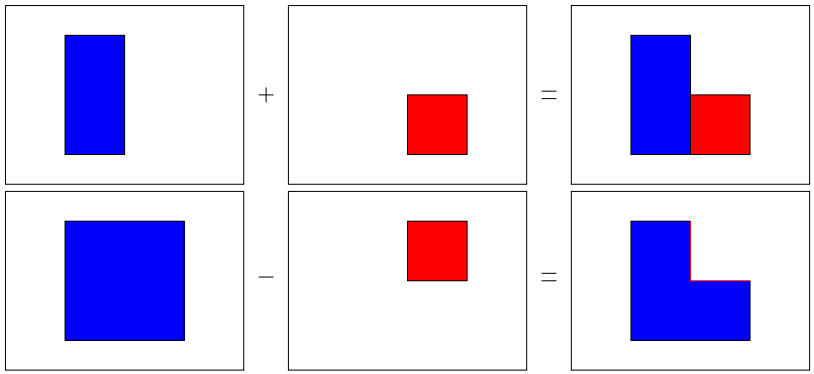
Not commutative

Boolean operations are not commutative!



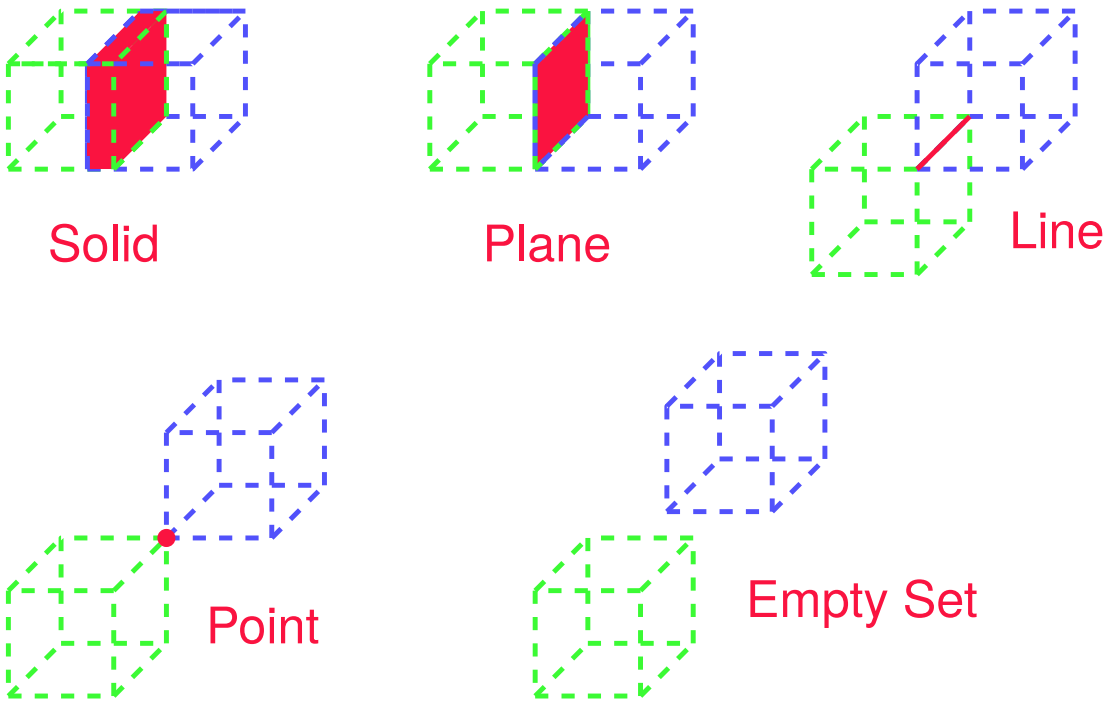
Not unique

A CSG representation is not unique.



Problems of Standard Boolean Operations

- Possible types of intersection of two solids:



- Boolean operations may create dangling faces or edges, or result in lower-dimensional “solids”.

Regularized Boolean Operations

- To eliminate those lower-dimensional sets, the Boolean set operations are regularized:
  1. Compute the interior of the solids. This yields objects without their boundaries.
  2. Apply the standard Boolean set operation.
  3. Compute the closure of the resulting object. This will add back the boundary.
- More formally, let $\cup, \cap, \setminus$ be the standard Boolean operations. We define their regularized counterparts $\cup^*, \cap^*, \setminus^*$ as follows:
  - $A \cup^* B := \overline{\text{int}(A) \cup \text{int}(B)}$,
  - $A \cap^* B := \overline{\text{int}(A) \cap \text{int}(B)}$,
  - $A \setminus^* B := \overline{\text{int}(A) \setminus \text{int}(B)}$.

Pros and Cons of CSG

Pros:

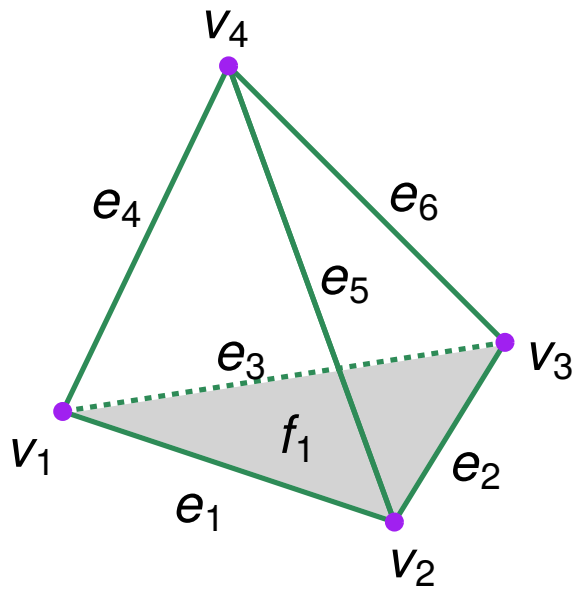
- A CSG tree mimics the design and construction process.
- Boolean operations are trivial for CSG objects.

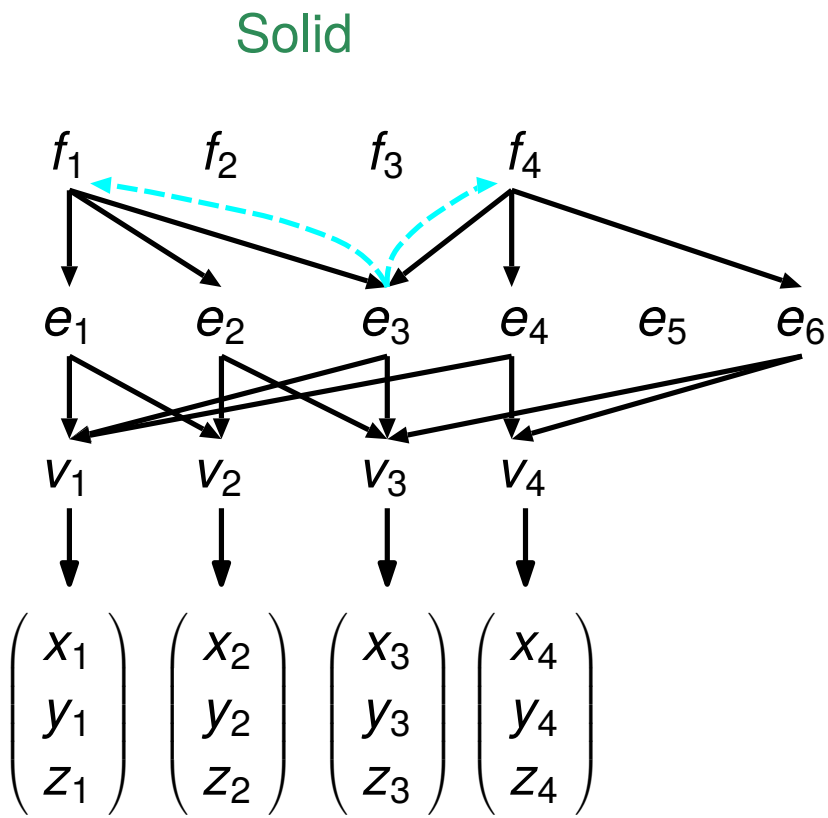
Cons:

- The surface of a CSG object is not readily available.
- Rendering a CSG objects is difficult unless (massively parallel) ray tracing is used.
- CSG trees are not unique: same-object detection and null-object detection are difficult.
- Support for free-form surfaces requires complicated mathematics.

Boundary Representation

- Describes a solid in a graph-like structure in terms of its surface boundaries: *vertices, edges, faces*.
- Common abbreviation: b-rep.
- It is imperative to model the full set of topological and numerical properties.





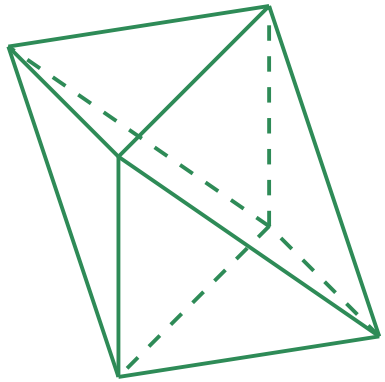
Boundary Representation

- If curved faces are involved then the supporting surfaces of the faces have to be stored, too. Similarly for the edges of a b-rep model.
- Most b-rep modelers support only solids whose boundaries are 2-manifolds.
- So-called Euler operators can be used to guarantee that b-rep modeling produces 2-manifolds.
- Boolean operations require sophisticated mathematical tools in order to represent the resulting object as a (valid) b-rep model.

- In practice, dual and hybrid representation schemes are often used in order to be able to benefit from the advantages of the individual schemes.

Polyhedra

- A polyhedron is bounded by a set of polygonal faces, where every edge is adjacent to an even number of faces.
- In order to guarantee a 2-manifold surface, every edge has to be shared by exactly two faces.
- Faces do not intersect except in common edges.
- All faces are required to be plane.



Deficiencies of polyhedral models

Be warned that (freely available) polyhedral models that are used purely for rendering purposes tend to be of extremely low quality, and may violate our rules drastically!

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

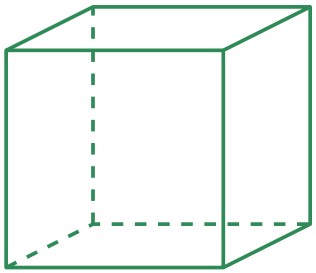
Polyhedra of Genus 0 and Euler's Formula

- Polyhedron of genus 0: Can be deformed to a ball; no holes.
  - Examples: Cube, tetrahedron, pyramid.
  - Torus is not a genus-0 polyhedron.
- *Euler's formula* for genus-0 polyhedra:

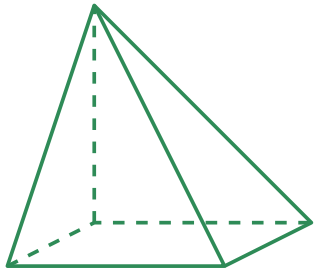
$$V - E + F = 2,$$

where

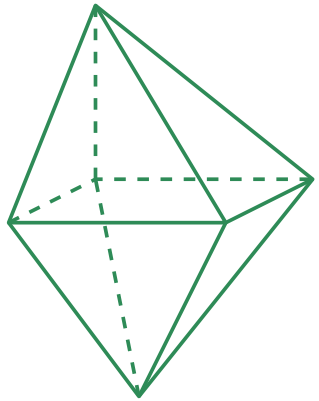
- V: #(vertices),
- E: #(edges),
- F: #(simply-connected 2D faces).



$V = 8$
$E = 12$
$F = 6$



$V = 5$
$E = 8$
$F = 5$



$V = 6$
$E = 12$
$F = 8$

- The validity of Euler's formula is necessary but not sufficient for a polyhedron to be of genus 0.

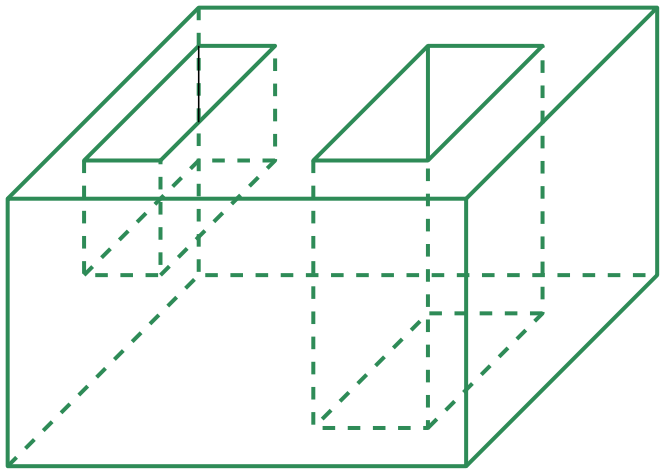
Polyhedra of Higher Genus

- Euler's formula generalizes to polyhedra of higher genus with 2-manifold boundaries:

$$V - E + F - H = 2(C - G),$$

where

- H: #(holes in 2D faces),
- G: #(holes passing through the polyhedron),
- C: #(connected components).



$$V - E + F - H = 2(C - G)$$

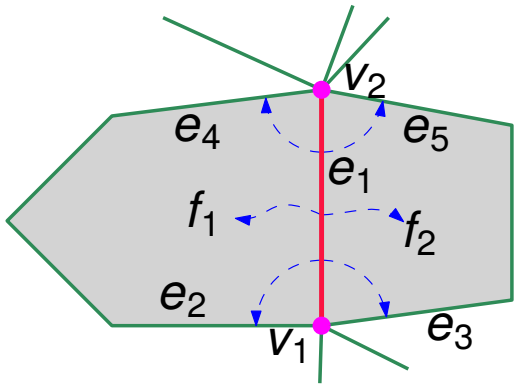
$$24 \quad 36 \quad 15 \quad 3 \qquad 1 \quad 1$$

Practical consequence of Euler's formula

A polyhedron with n vertices can be stored in $O(n)$ memory units.

Winged-Edge Representation

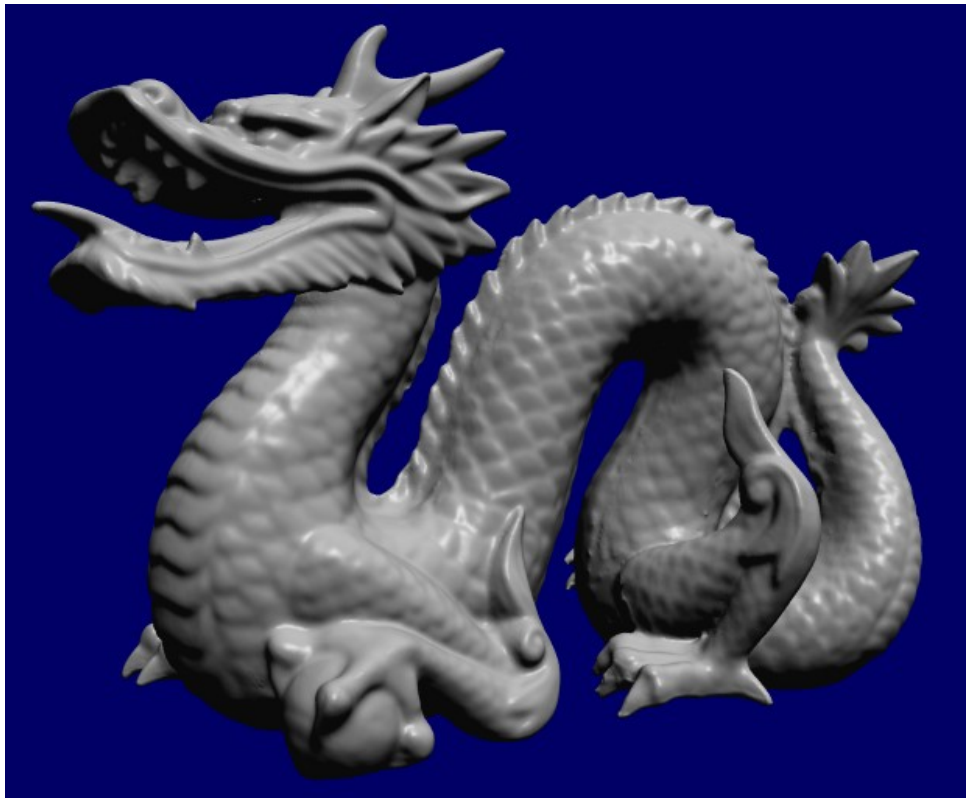
- Common way to represent 2-manifold polyhedra of genus 0.
- Each edge e stores
  - two faces f_1, f_2 adjacent to e,
  - two endpoints v_1, v_2 of e,
  - two edges incident to v_1 immediately before and after e in clockwise direction,
  - two edges incident to v_2 immediately before and after e in clockwise direction.



- Each vertex v stores a pointer to one of the edges incident to v.
- Each face f stores a pointer to one of the edges bounding f.
- Other common alternatives: Doubly-connected edge list (DCEL), half-edge data structure.

Famous Polyhedral Models

Stanford Bunny

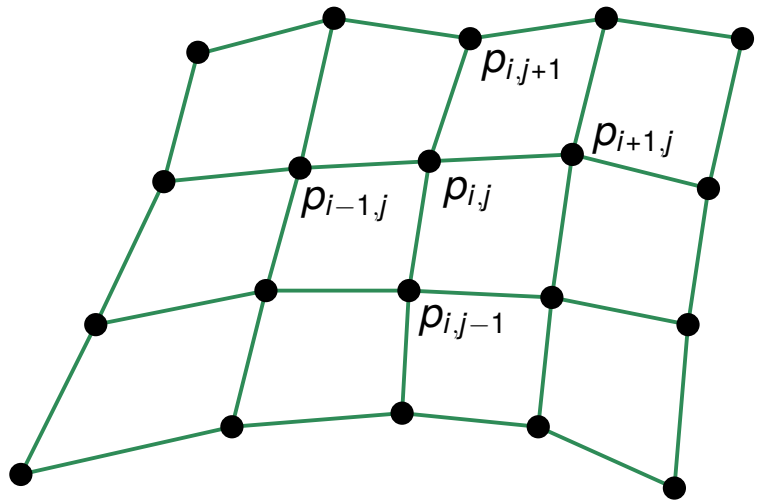


Stanford Dragon

[Image credit: https://graphics.stanford.edu/data/3Dscanrep/]

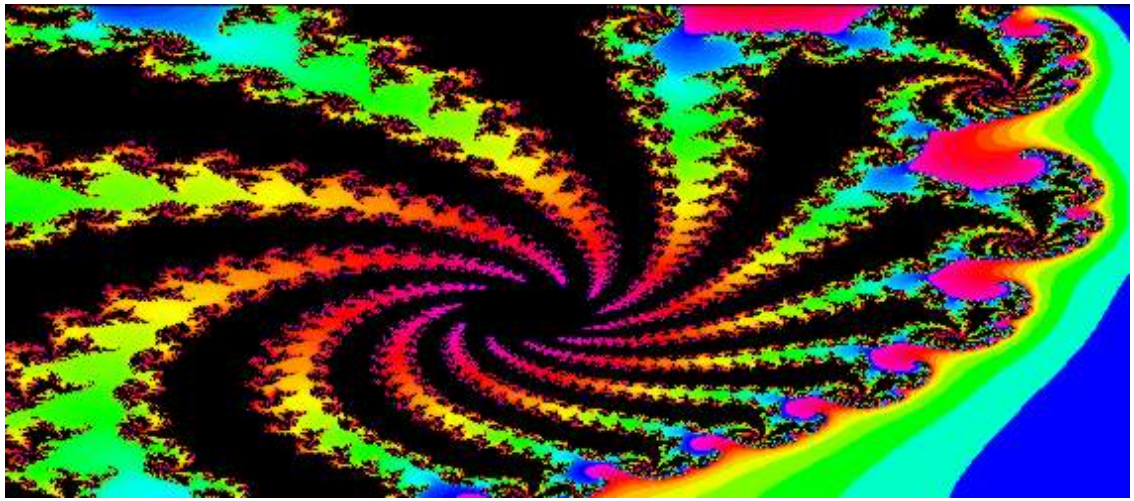
Particle Systems

- Points (aka “particles”) that follow laws of physics are used to model an object.
- Sample phenomena generated by particle systems:
  - Smoke, fire, fog;
  - Deformable objects: clothes, elastic objects, rope;
  - Waves, turbulent air flow, storm.
- *Independent particles*: Position of a particle does not depend on others, e.g., particles under gravity. A time step for an n-particle simulation requires $\Theta(n)$ time.
- *Interactive particles*: Position of a particle depends on the others; particles are “linked”. Each time step requires $\Theta(n^2)$ time.
- In practice the dynamics of a particle do often depend on its neighbors, e.g., clothing simulation, ropes, stars.
- Spring forces can be used to model the interaction of adjacent particles.

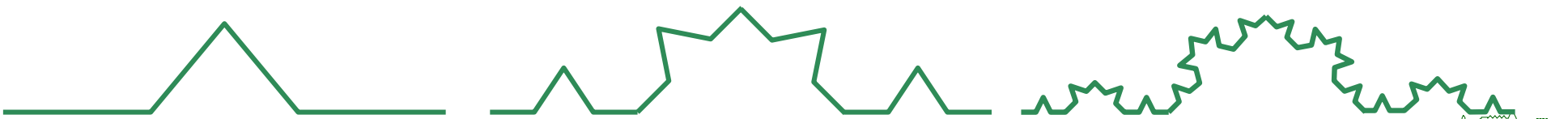


Fractals

- The term “fractal” is used for self-similar objects with fractal dimensions that strictly exceed their topological dimension.
- Sample fractals: Mandelbrot and Julia sets.

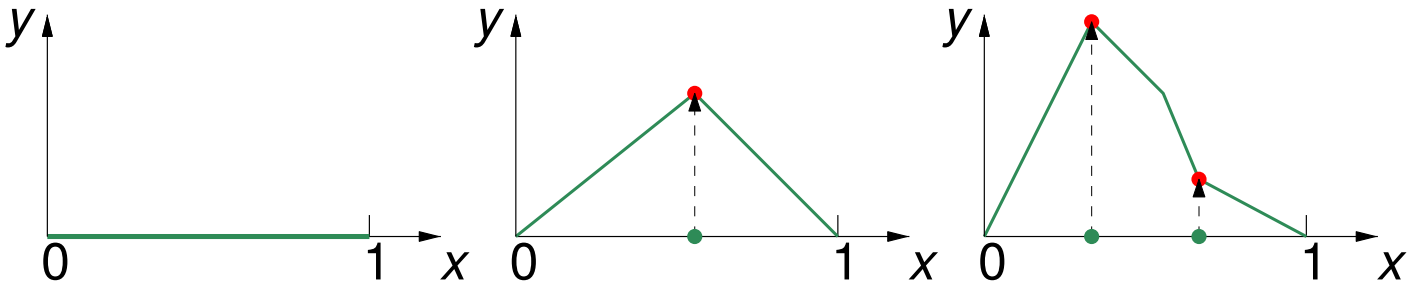


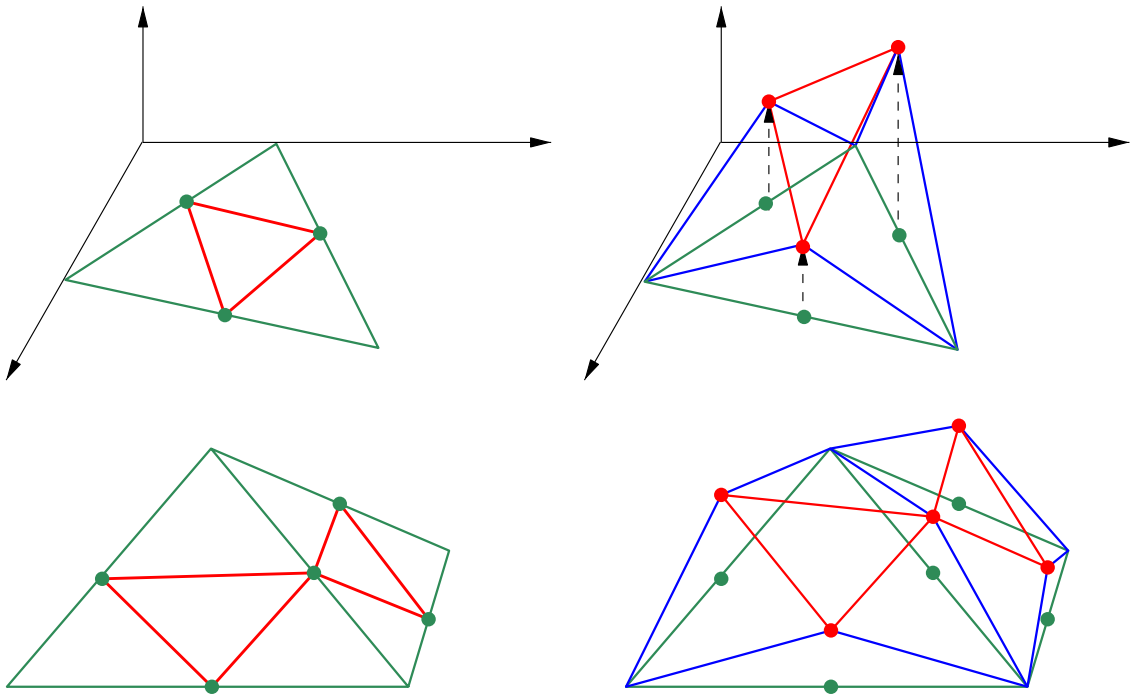
- When constructed by an algorithm, we repeat the same construction scheme recursively: *Iterated function system* (IFS).
- E.g., Koch Curve.



Fractals: Modeling Terrains

- [Carpenter (1980)]: In 1980, in “Vol Libre”, he first used recursive subdivision to model terrains.





Fractals: Barnsley's Fern

- [Barnsley 1988]: "Fractals Everywhere".
- His fern is modeled by four affine transformations that are selected randomly (with four different probabilities).

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

3 Raster Graphics

- Light and Color
- Color Models
- Scan Conversion

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

The Physical Nature of Light

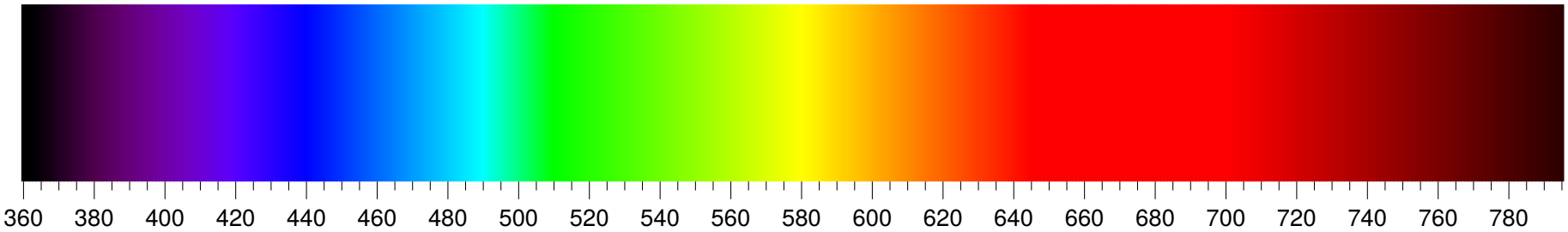
Caveat

The following slides present a *simplified* view of the physical nature of light. Consult a physics textbook for an in-depth coverage of these topics!

- Light is electromagnetic energy that is visible to humans.
- According to physicists, light exhibits a wave-particle duality.
  - The wave model: makes an analogy comparing light to water waves.
  - The particle model: light is made up of many little particles.
- Neither model is really complete or correct.
  - Under some circumstances light behaves like a wave.
  - Under other circumstances light behaves like a particle.
- The particle model alone can go a long way towards understanding and explaining light, though.
- The basic particle of light is called a *photon*: We can regard it as an object that moves along a straight line and vibrates during its move.
- This vibration is a kind of mathematical abstraction.
- It is useful since much of the mathematics that describe vibrations seem to work in describing the behavior of light.

The Physical Nature of Light

- With every photon we can associate a particular **frequency** f of vibration.
  - The frequency is measured in Hertz (Hz).
  - Visible frequencies are within 4.3×10^{14} Hz to 7.5×10^{14} Hz.
- An alternate way to characterize the vibration of a photon is to consider its **wavelength** λ.
  - The wavelength is measured in meters, or nanometers (with $1\,\text{nm} = 10^{-9}\,\text{m}$).
  - Visible wavelengths lie in the 400 nm to 750 nm range, or, at the very best, within 380 nm to 780 nm.
  - Long wavelengths are perceived as reds and short wavelengths are perceived as blues.



The Physical Nature of Light

- Wavelength and frequency are closely related:

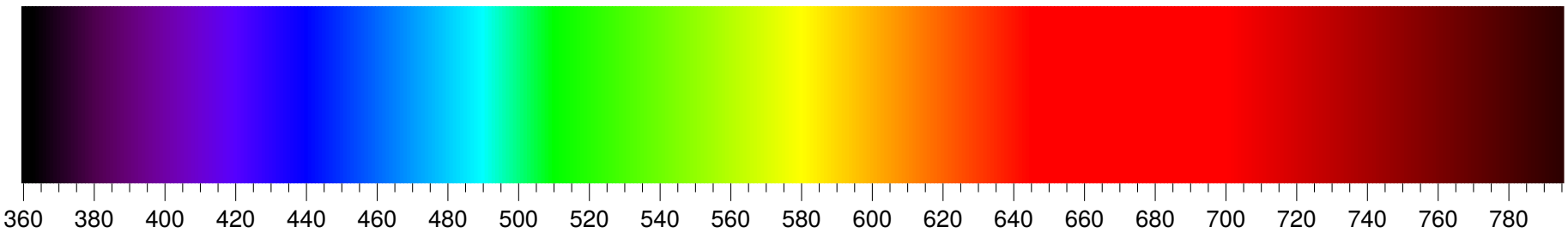
$$\lambda \cdot f = c,$$

where $c := 299\,792\,458\,\mathrm{m\,s^{-1}}$ is the speed of light traveling in vacuum.

- The energy, E, of a photon is directly related to its frequency (Planck-Einstein relation):

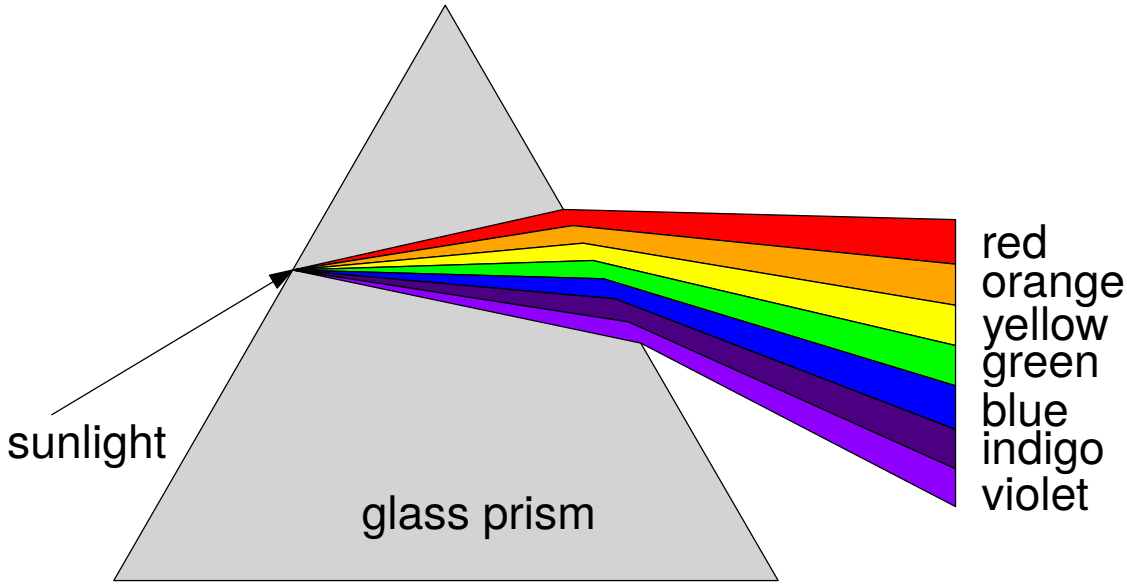
$$E = h \cdot f,$$

where $h := 6.626\,070\,15 \times 10^{-34}\,\mathrm{J\,s}$ is Planck's constant (Dt.: Plancksches Wirkungsquantum); energy is measured in Joule (J).



Color and Spectra

- Have you ever seen a white band in a rainbow?
- Likely not. White is not a pure *spectral color*: No single photon can give us the impression of white light.
- A prism can be used to show that white light is really a mixture of different spectral colors.
- White light:
  - photons of many different spectral colors
  - strike the same region of our eye
  - nearly simultaneously.



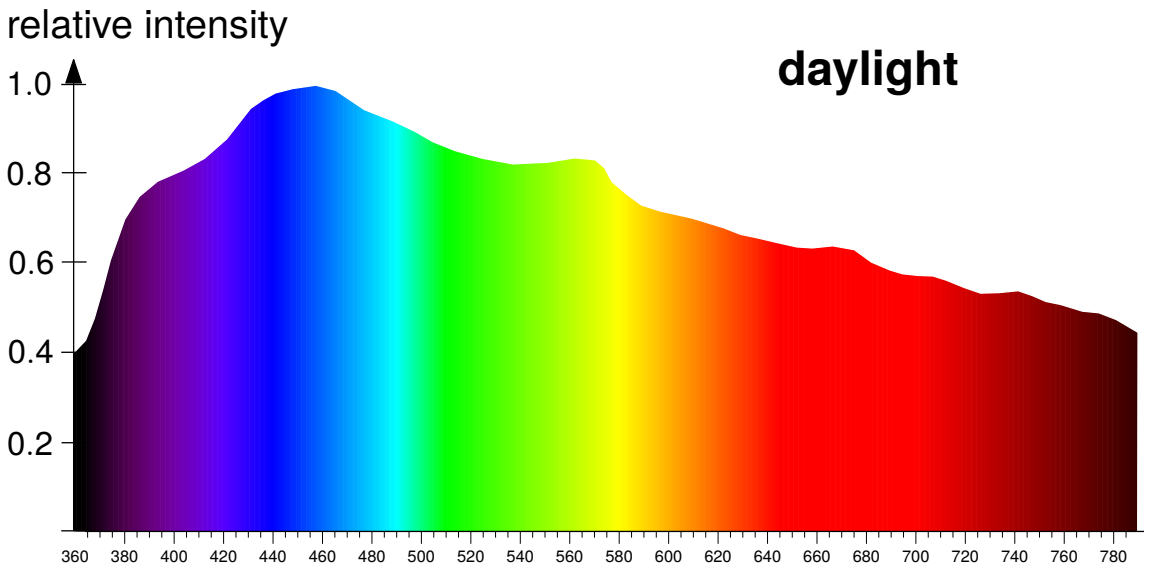
Wavelength is important!

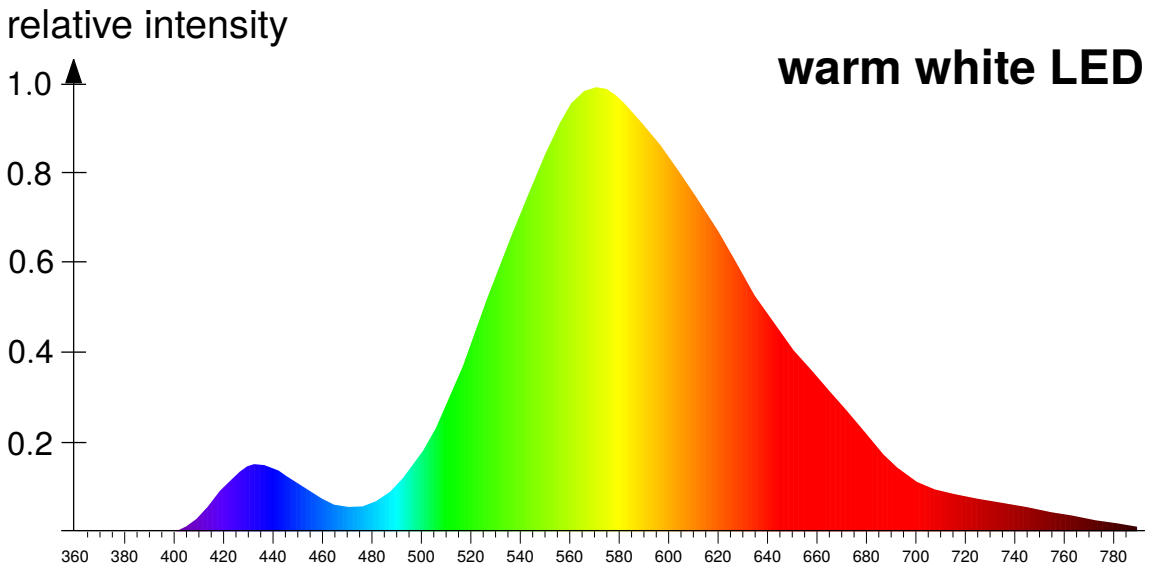
Reflection and refraction of light depend on the wavelength!

Color and Spectra

- How can we characterize different amounts of photons at different wavelengths?
- We could
  - set up a measuring instrument,
  - count the average number of photons,
  - at each visible wavelength,
  - over some period of time,
  - and then plot the results.
- Such an intensity versus wavelength plot is called a *frequency spectrum plot*, which is often abbreviated simply as *spectrum*.
- Hence, one way to describe color is to attach a spectrum with each light ray, describing the light traveling along that ray.
- Except for a few cases, such as fluorescent light that has spikes, the spectrum (regarded as a function of wavelength) tends to be rather smooth.

Sample Spectra





UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Color and the Eye: Rods and Cones

- Human vision relies on light sensitive cells (“sensors”) in the retina of the eye: rods and cones.

  Rods (Dt.: Stäbchen) are cells which can work at low intensity, but cannot handle color.

  Cones (Dt.: Zäpfchen) are cells which can handle color, but require brighter light to function.

- Cones are concentrated near the *fovea (Dt.: Sehgrube)* of the retina (Dt.: Netzhaut).
- Many more rods ($\approx$ 120M) than cones ($\approx$ 6M).

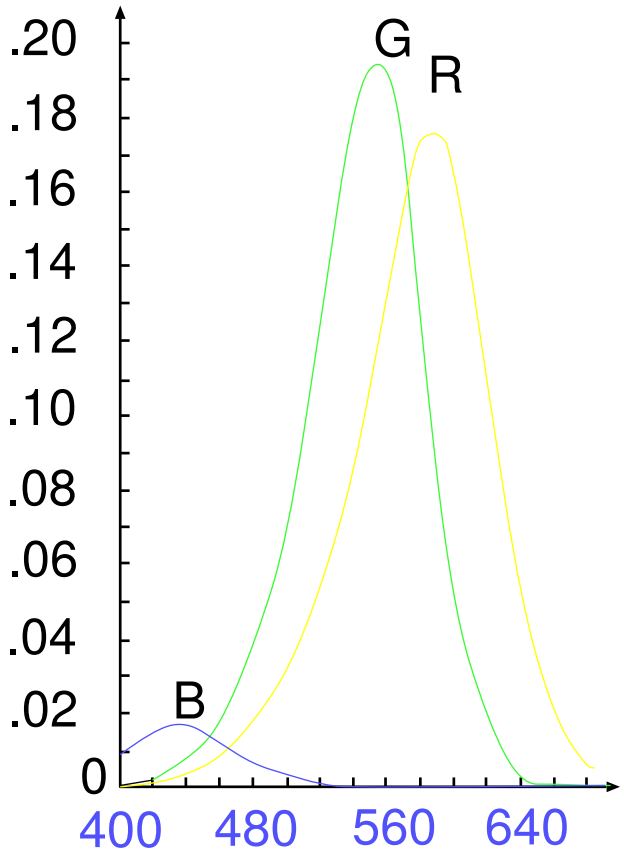
Color and the Eye: Trichromatic Theory

- Proposed by Thomas Young (1801), refined by Hermann von Helmholtz (1861).
- The standard assumption is that the retina has three different types of cones, with peak sensitivity to
  - yellow: the peak response is around 580 nm, most commonly but not correctly also referenced as red;
  - green: the peak response is around 545 nm; and
  - blue: the peak response is around 440 nm.
- Every single wavelength triggers all three kinds of cones by different amounts.
- The trichromatic theory helps to explain color blindness:
  - Protanope (red blindness, top-right),
  - Deuteranope (green blindness, bottom-left),
  - Tritanope (blue blindness, very rare, bottom-right).

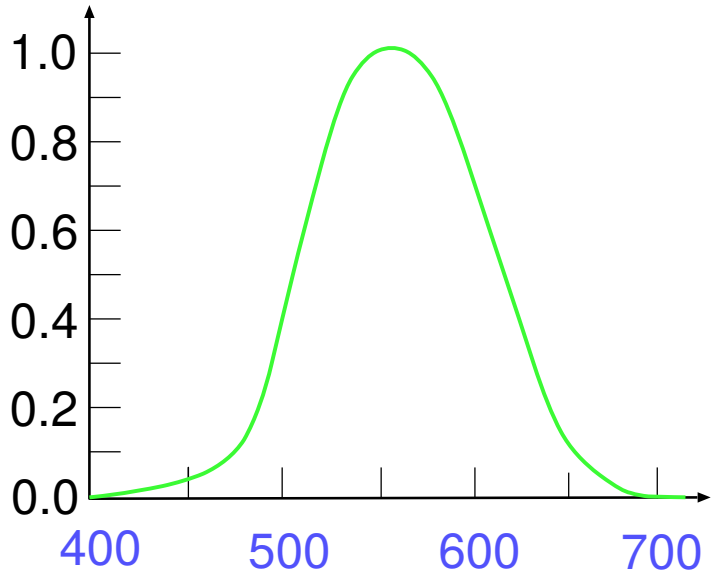
[Image credit: https://www.graphics.cornell.edu/online/tutorial/color/]

Color and the Eye

- Spectral-response functions of each of the three types of cones on the human retina, describing the fraction of light absorbed by each cone with respect to wavelength.



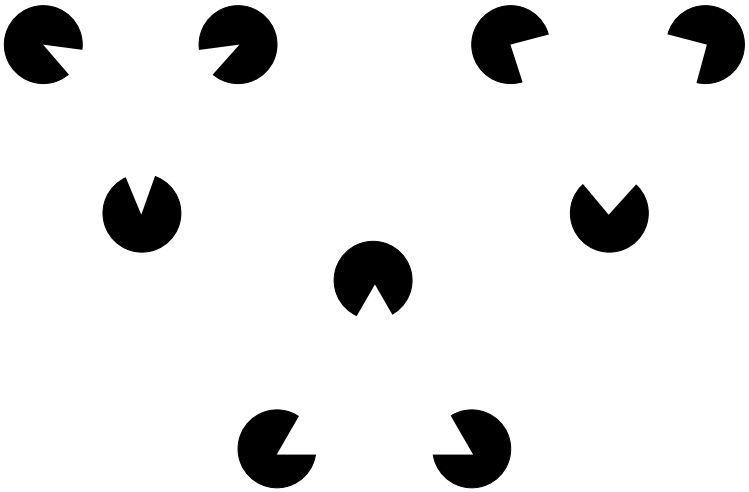
- Luminous-efficiency function for the human eye.



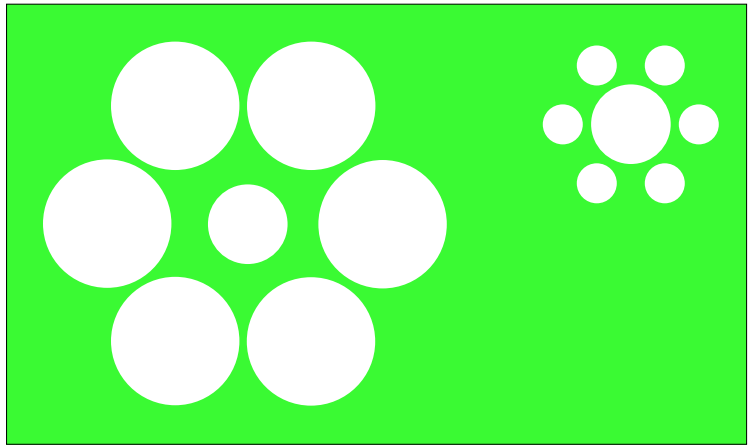
- We have a peak sensitivity to wavelengths around 550 nm:
  - About two thirds of the cones are sensitive to yellow.
  - Almost one third is sensitive to green.

Optical Illusions

- The human visual system is very susceptible to optical illusions!
- Humans are not particularly good at judging absolute quantities, such as angles or areas. E.g., we tend to overestimate small angles and underestimate large angles.
- *Kanizsa triangles*: Our visual system fills in the missing portions of the edges in order to allow us to see triangles (“subjective contours”).



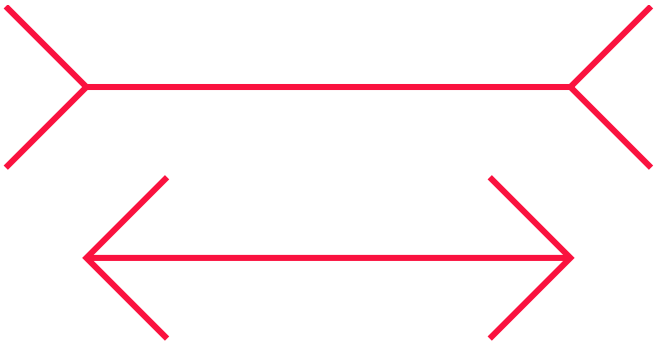
- *Ebbinghaus illusion*: The two inner circles are of the same size! Colors, shapes, and relative sizes can fool humans easily . . .



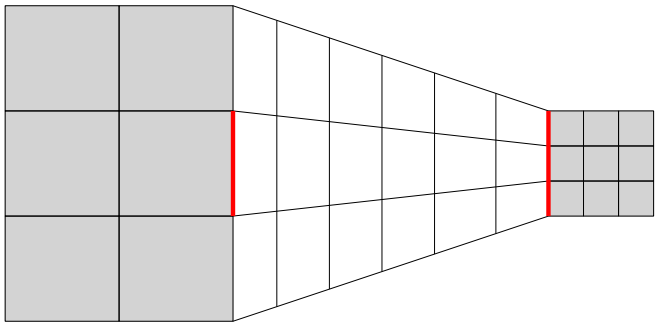
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Optical Illusions

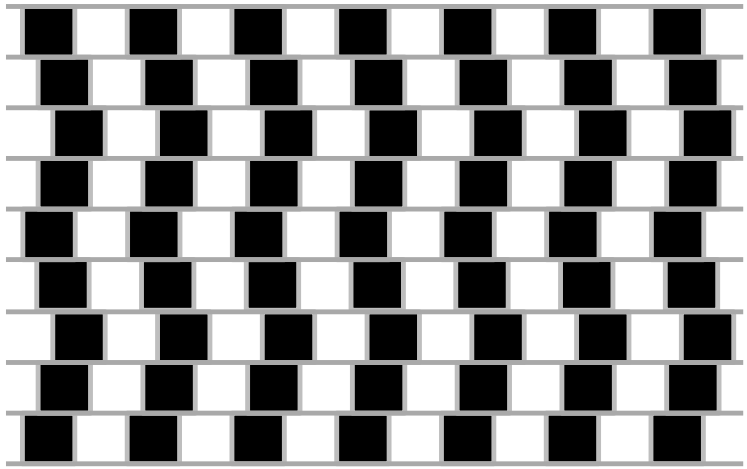
- Geometrical-optical illusions are characterized by an incorrect perception of size, length or curvature.
- *Müller-Lyer illusion*: The horizontal line segments are of the same length!



- *Ponzo illusion*: The red vertical line segments are of the same length and width!



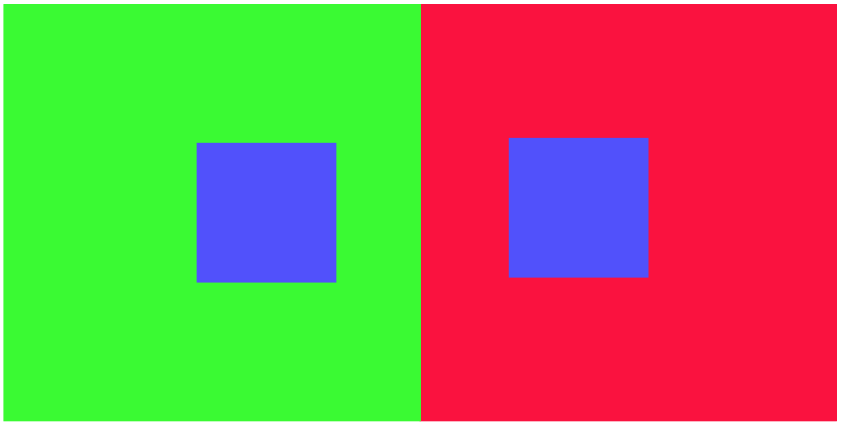
- *Café wall illusion*: The gray horizontal lines between staggered rows of alternating black and white squares are parallel.



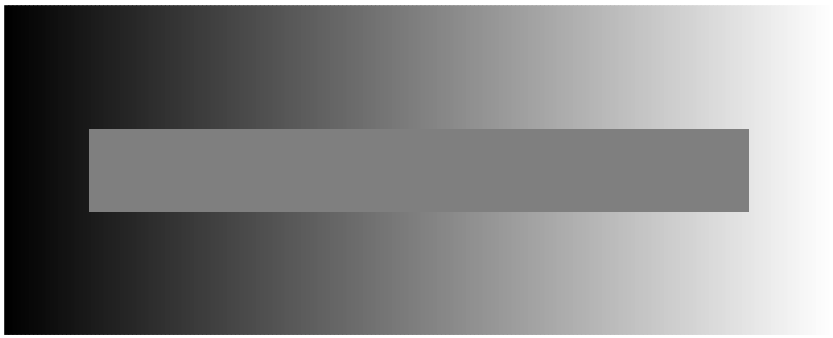
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Optical Illusions

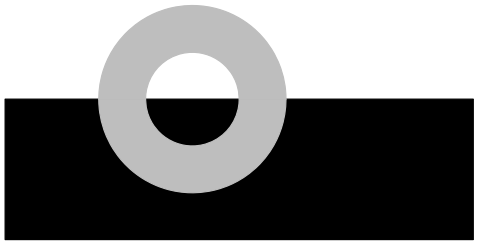
- *Simultaneous contrast illusion*: The luminocity of an object perceived by a human depends on its surrounding background.
- The two inner squares are colored equally!

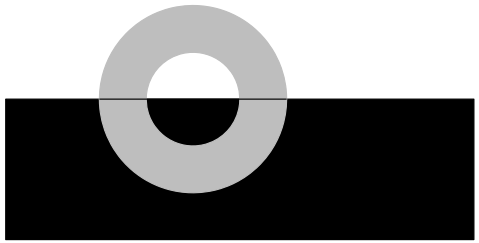


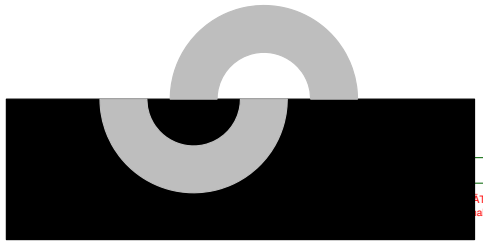
- The background is a color gradient that progresses from black to white. The horizontal bar is shaded uniformly!



- *Koffka ring*: Brightness differences between adjacent areas are enhanced in human perception. The more pronounced the separation of (equally-colored) areas is, the stronger the contrast illusion.

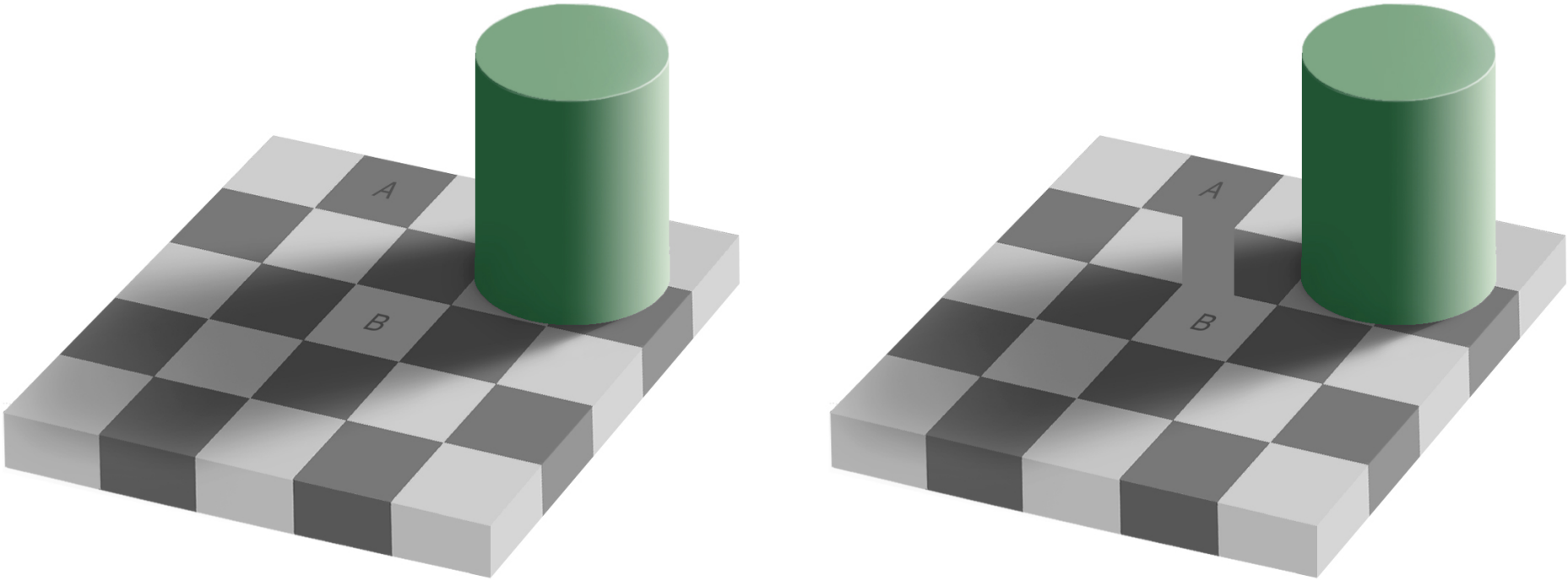






Optical Illusions

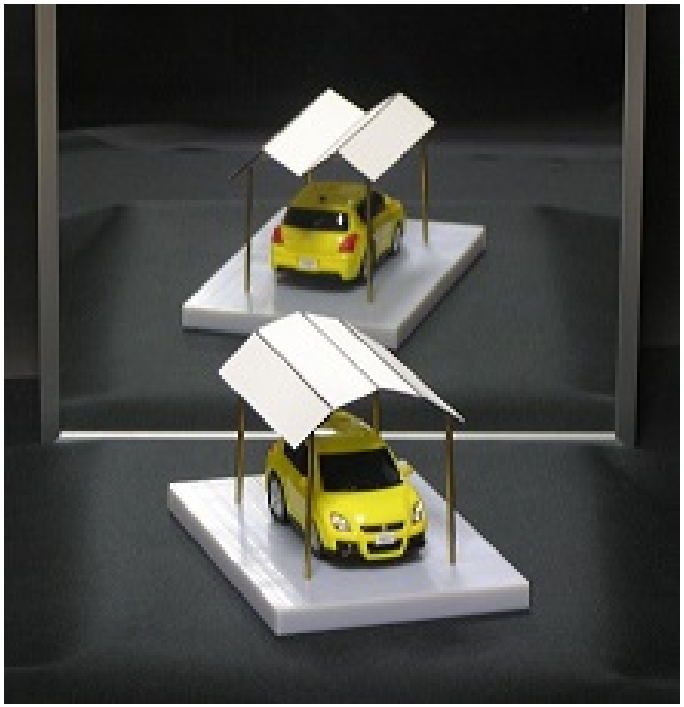
- *Checker shadow illusion* (Adelson 1995): The two cells labelled *A* and *B* are of exactly the same color. (E.g., a color picker returns the hex value 787878, or 47% each of red, green and blue.) See
  `https://www.youtube.com/watch?v=z9Sen1HTu5o` for a video.



[Image credit: Wikipedia]

Optical Illusions

- *Ambiguous garage roof illusion* (Sugihara 2015): The garage roof is neither round nor corrugate. The illusion exploits the fact that a single image does not convey depth information, and that the human brain prefers to take the silhouette curve of the roof as the intersection of the roof with a plane normal to the obvious axis of the roof. See https://www.youtube.com/watch?v=KtA6u1HIqbg for a video.



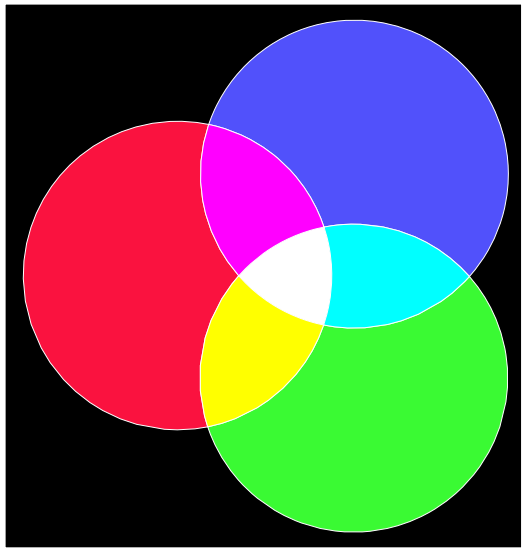
[Image credit: K. Sugihara]

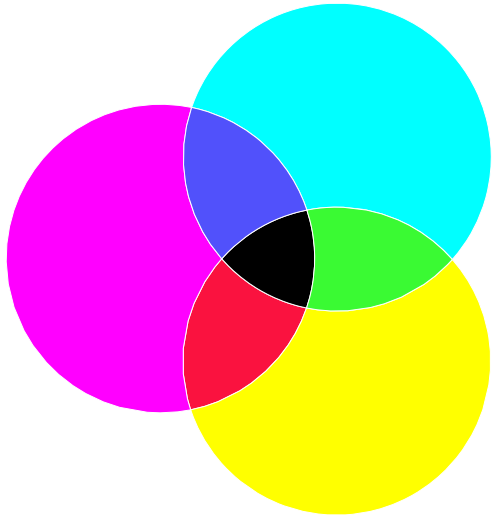
- *Schröder staircase illusion* (Sugihara 2020): https://www.youtube.com/watch?v=5DYeAkx2IBo.
- More information (plus several videos) on https://www.isc.meiji.ac.jp/~kokichis/Welcomee.html.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

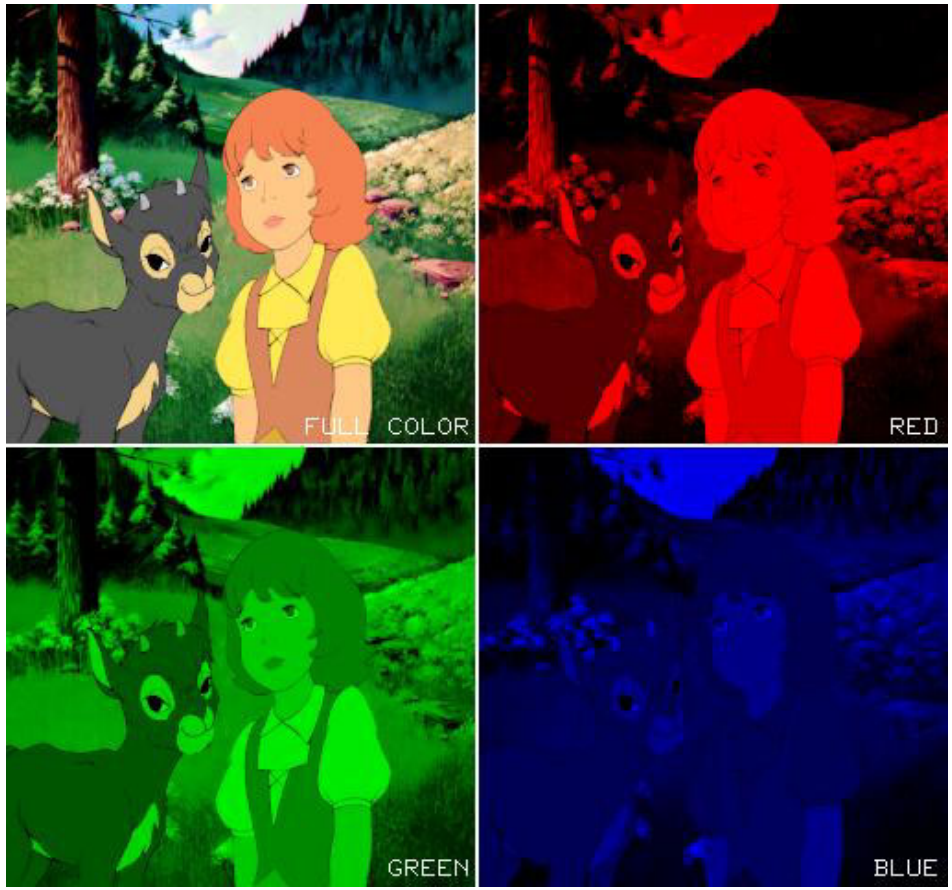
Primary Colors

- It is natural to attempt to model colors as a “mixture” of a small number of *primary colors*.
- There are two basic ways of mixing color: one is additive, by combining emitted light of different colors, while the other is subtractive, by preventing certain portions of white light from being reflected.
  - Additive representation: Starting from black, create colors by adding different amounts of the primaries. E.g., adding red and blue generates magenta.
  - Subtractive representation: Starting from white, create colors by subtracting portions of white. E.g., magenta dye blocks green from being reflected.





Sample Additive Representation



[Image credit: https://www.graphics.cornell.edu/online/tutorial/color/]

Perceptual Color Matching: CIE-*XYZ*

- In 1931, CIE (Commission Internationale de l'Eclairage) defined a "standard observer":
  - Roughly, a standard observer is a small group of 15–20 individuals. It is supposed to be representative of normal human color vision.
  - The observer viewed a split screen with (close to) 100% reflectance.
  - On one half, a test lamp casts a pure spectral color on the screen.
  - On the other half, three lamps emitting varying amounts of red, green, and blue light attempted to match the spectral light of the test lamp.
  - The observer determined when the two halves of the split screen were identical, thus defining the tristimulus values for each distinct spectral color.
- It was realized that a linear combination (with non-negative coefficients) of the red, green and blue primary lamps could not reproduce all spectral light.
- Since negative coefficients were considered inadequate, CIE defined three (artificial) additive primaries and a corresponding color model, CIE-*XYZ*, in 1931.
- The CIE color model was developed to be completely independent of any means of emission or reproduction and is based as closely as possible on how humans perceive color.
- In 1960, the CIE-*XYZ* model was modified, and again revised in 1976 to become CIE-1976-L*a*b and CIE-1976-L*u*v, in an attempt to linearize the perceptibility of color differences. (The basic principles remained the same, though.)

CIE Chromaticity Diagram

- Every visible color (and some invisible ones, too) can be expressed as a combination of the CIE primaries: $\alpha \cdot X + \beta \cdot Y + \gamma \cdot Z$.
- This defines a 3D linear color space with respect to X, Y and Z.
- It is common to project this space onto the plane $X + Y + Z = 1$.
- The coordinates of this projected 2D plane are usually called x and y, where

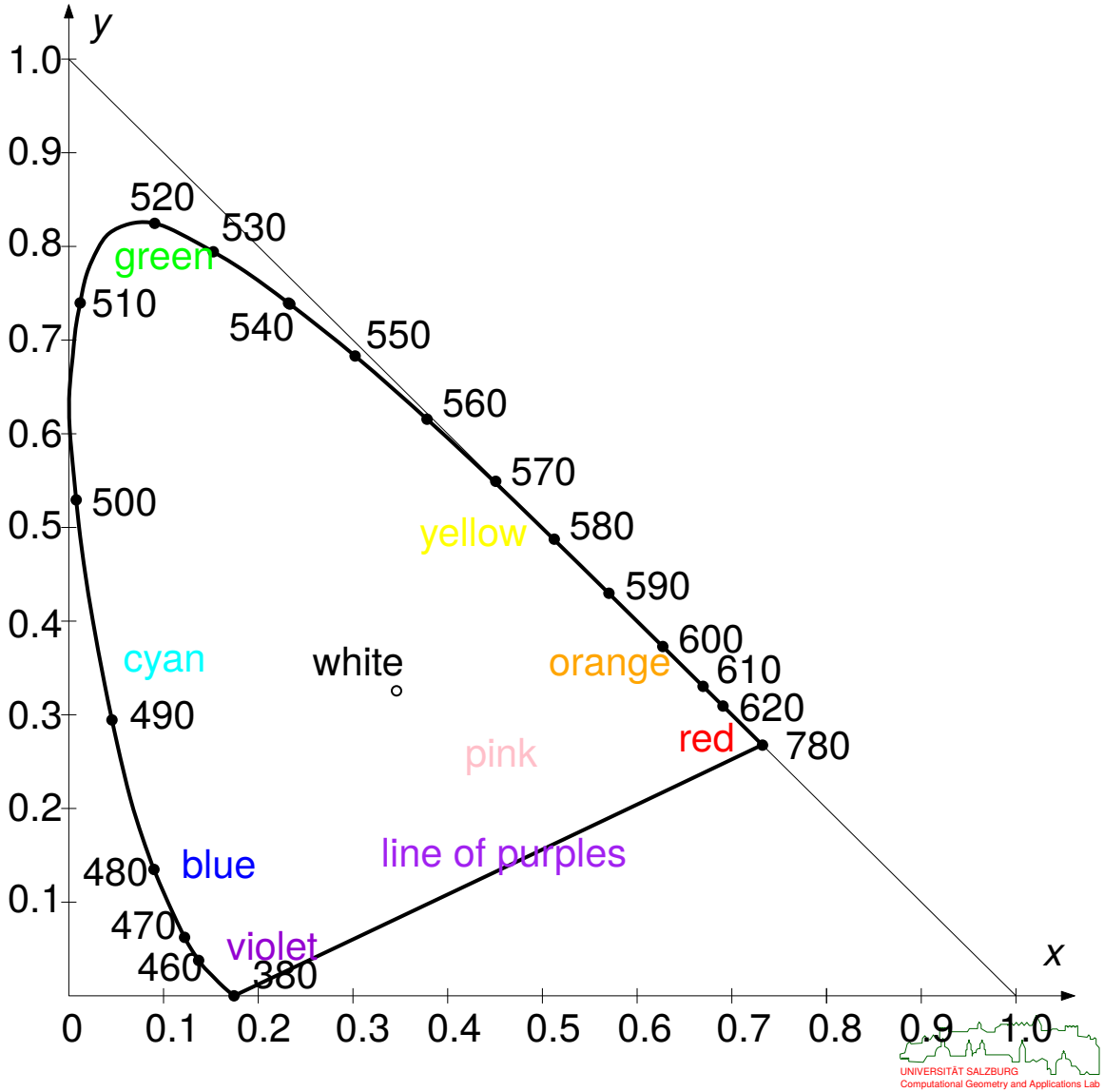
$$x = \frac{X}{X+Y+Z} \quad \text{and} \quad y = \frac{Y}{X+Y+Z} \quad \text{and} \quad z = 1 - x - y = \frac{Z}{X+Y+Z}.$$

- The resulting diagram is known as *chromaticity diagram* (Dt.: Chromatizitätsdiagramm, Farbwertdiagramm).
- Note that this diagram captures only color but not luminance of the light source.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

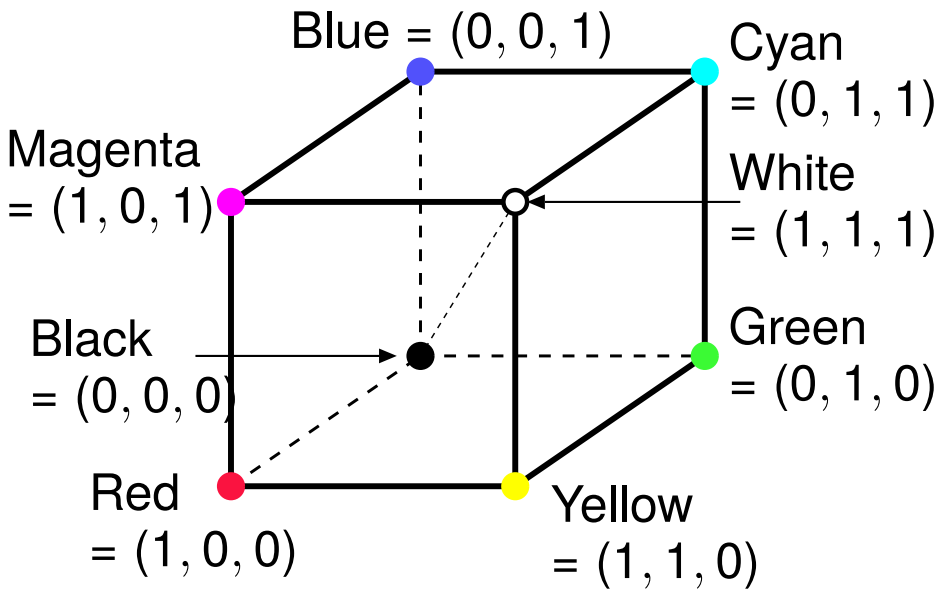
CIE Chromaticity Diagram

- Spectral colors are on the curved boundary of the “horseshoe”.
- Colors on the line joining violet and red, the *line of purples*, are non-spectral; they are additive mixtures of red and violet.
- White is near the middle. (CIE D6500 is at position (0.313,0.329).)
- Any color that results from additive mixing of three colors will lie in the triangle connecting these three colors.



RGB Color Model

- The RGB color space is given by the unit cube, where the primaries red (R), green (G), and blue (B) correspond to the coordinate axes.
- In this system, $(0, 0, 0)$ corresponds to black and $(1, 1, 1)$ is white.



RGB Color Model

- The RGB model is the most widely used color model for specifying the color of a pixel on a monitor.
- Its practical importance is derived from the fact that triads of three LCD/LED cells – with colors red, green, and blue – are used to produce a color in an additive way on a standard monitor.
- Although the arithmetic interpolation between two RGB triples is geometrically linear, such an interpolation need not be linear perceptually: An incremental change of an RGB triple may produce no perceivable difference in one part of the RGB cube, while it may create visually different colors in some other part of the cube.
- Always bear in your mind that the class of all colors that can be displayed on a monitor is a subset of the colors perceivable by humans.
- Recall that an RGB image may look different on different monitors.

CMYK for Color Printing

- When a white surface is coated with cyan ink, no red light is reflected: Cyan subtracts red from the reflected white light.
- CMY color model: The inks used in color printing are cyan (light blue), magenta (purple), and yellow.
- To maintain black color purity, and to speed-up the drying process, a separate black ink is used rather than to rely on cyan, magenta, and yellow to generate black: CMYK.

Dye Color	Absorbs Color	Reflects Colors
Cyan (C)	Red	Blue and Green
Magenta (M)	Green	Blue and Red
Yellow (Y)	Blue	Green and Red
Black (K)	All	None

- As the RGB model, the CMY model can be regarded as a unit cube, where $(0, 0, 0)$ corresponds to white and $(1, 1, 1)$ is black.

RGB-to-CMYK Conversion

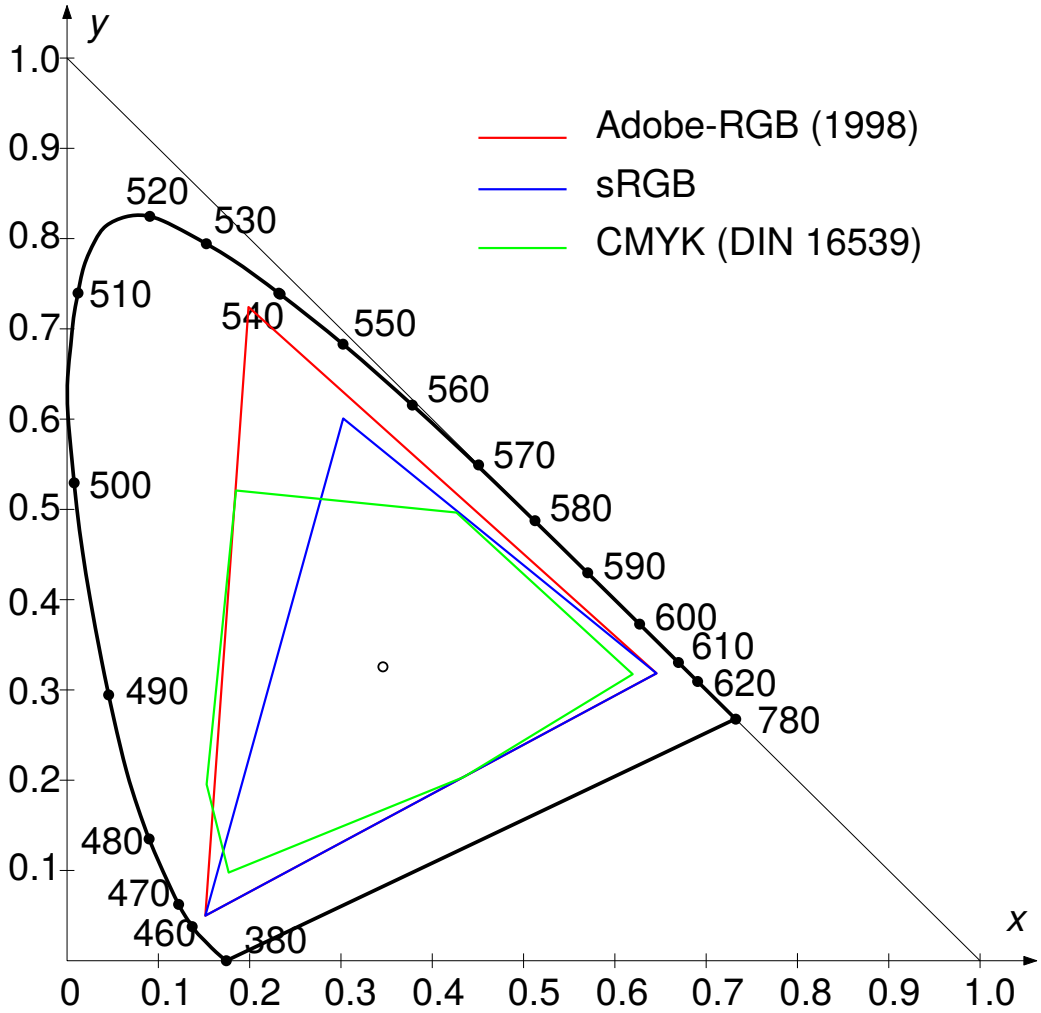
- Given intensity values R, G, B, where each value is between 0 and 1, we can convert to CMY using the following *masking equations*:

$$C = 1 - R \quad \text{and} \quad M = 1 - G \quad \text{and} \quad Y = 1 - B.$$

- This is approximate: It assumes that the printed cyan is equal to white minus the red of the monitor, and this is rarely the case.
- Adding black (K) as an additional color further complicates the matter.
- Typically, a color printer cannot print all colors a computer monitor can display, and a computer monitor cannot display all colors a color printer can print!
- E.g., pure green or pure blue is outside of the gamut of printers.
- Consequently, the same image displayed on a computer monitor need not match the image printed in a publication.
- Color shifts may occur when the RGB-to-CMYK conversion takes place.
- Nevertheless, this “four-color process” or “full-color” printing generates the vast majority of magazines and marketing publications.
- High-fidelity conversions from RGB to CMYK currently require careful tweaking to compress and stretch the RGB gamut of a particular image so that it fits into the available CMYK gamut.
- This is an area of active research!

Color Gamut

- The color gamuts of films, monitors and color printers form (fairly small) subsets of the chromaticity diagram: *gamut mapping* may be required! (Note that hardware vendors sometimes prefer to claim larger/different gamuts!)

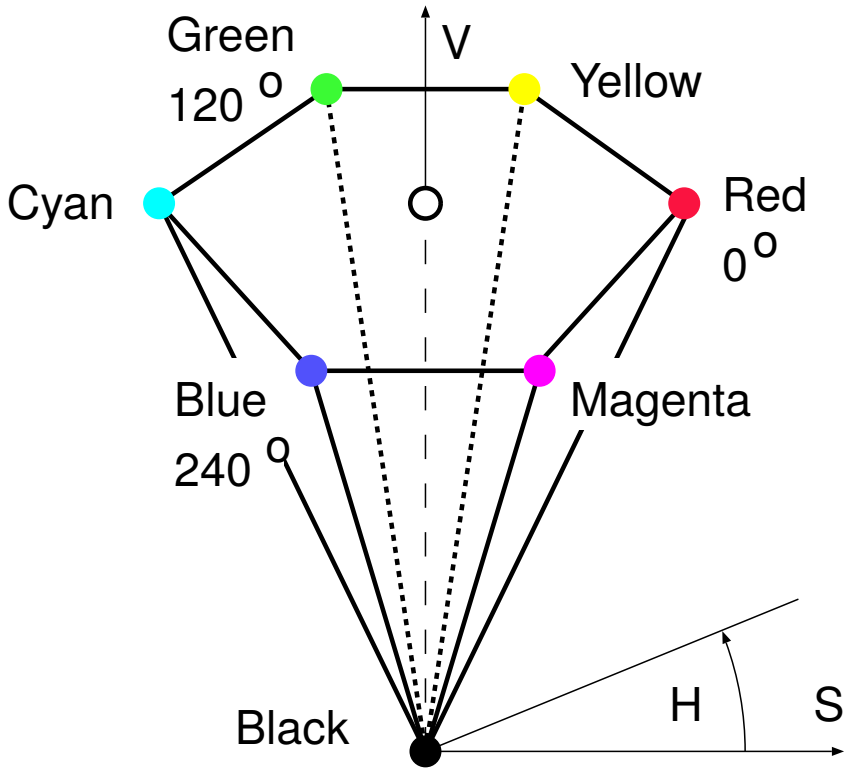


Other Color Models

- The CIE color diagram is a scientific formalism, but it does not provide a natural user interface for specifying colors.
- RGB and CMY(K) are great from a technical point of view, but both are equally bad from an artist's perspective.
- Several other color models have been developed:
  - HSV: Hue, Saturation, Value.
  - HLS: Hue, Lightness, Saturation.

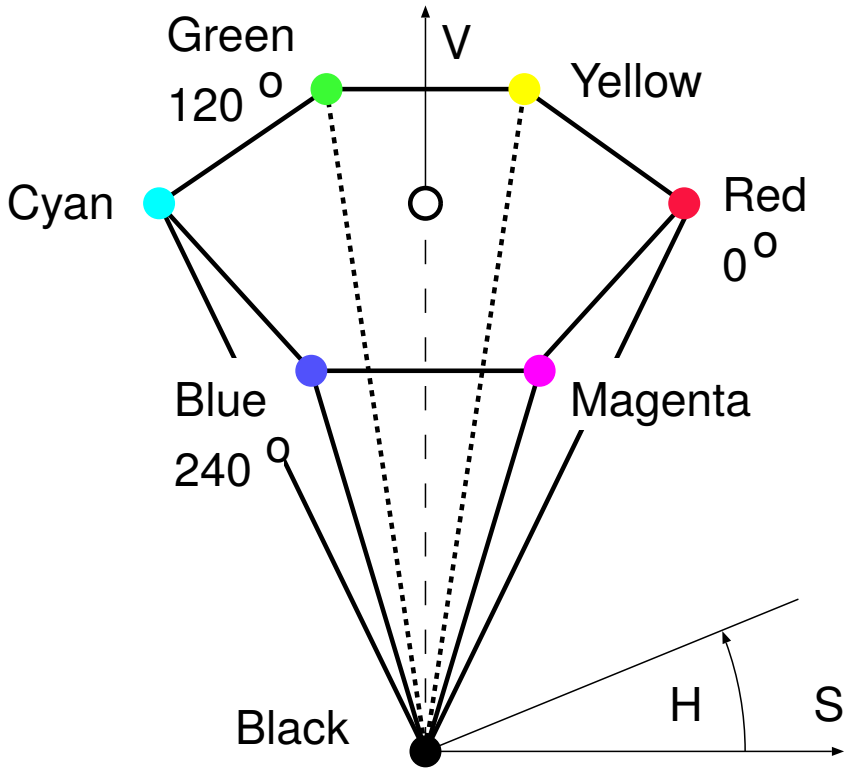
Hue, Saturation, Value

- The HSV dates back to a color notation proposed by Munsell in 1905:
  - *Hue:* “It is that quality by which we distinguish one color from another, as red from yellow”. It is given by the dominant wavelength of the light in that color.
  - *Saturation:* “The degree of departure of a color sensation from that of white or gray”. It models the purity of the color.
  - *Value:* “It is that quality by which we distinguish a light color from a dark one”. It models the brightness (i.e., amount of energy) of the light.



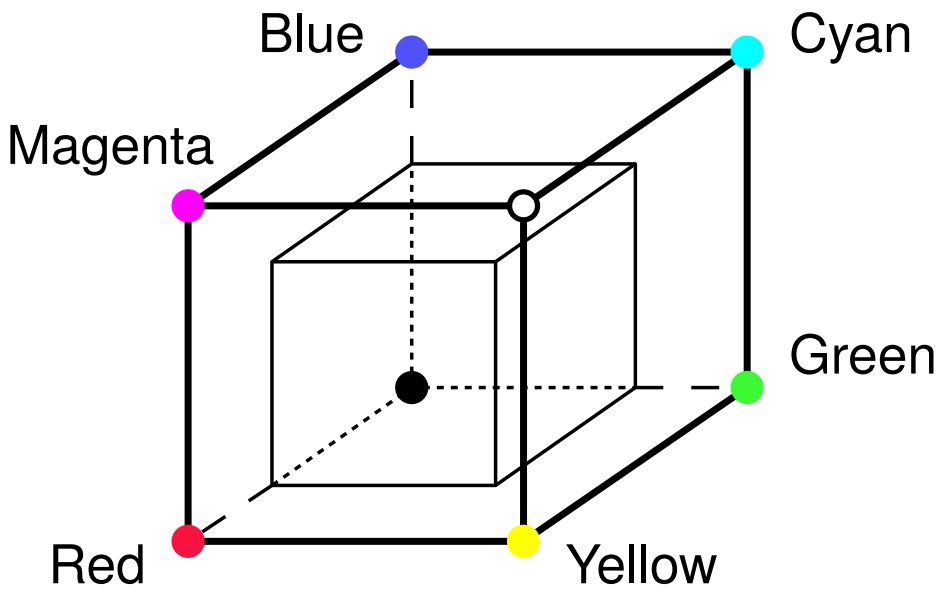
HSV Pyramid

- Hue is measured from 0 to 360 degrees counter-clockwise, with red at 0.
- Saturation is the distance away from the center line; decreasing S corresponds to adding white.
- Value is the vertical distance above black; decreasing V corresponds to adding black.
- The spectral colors are given by $V = S = 1$ and arbitrary H.



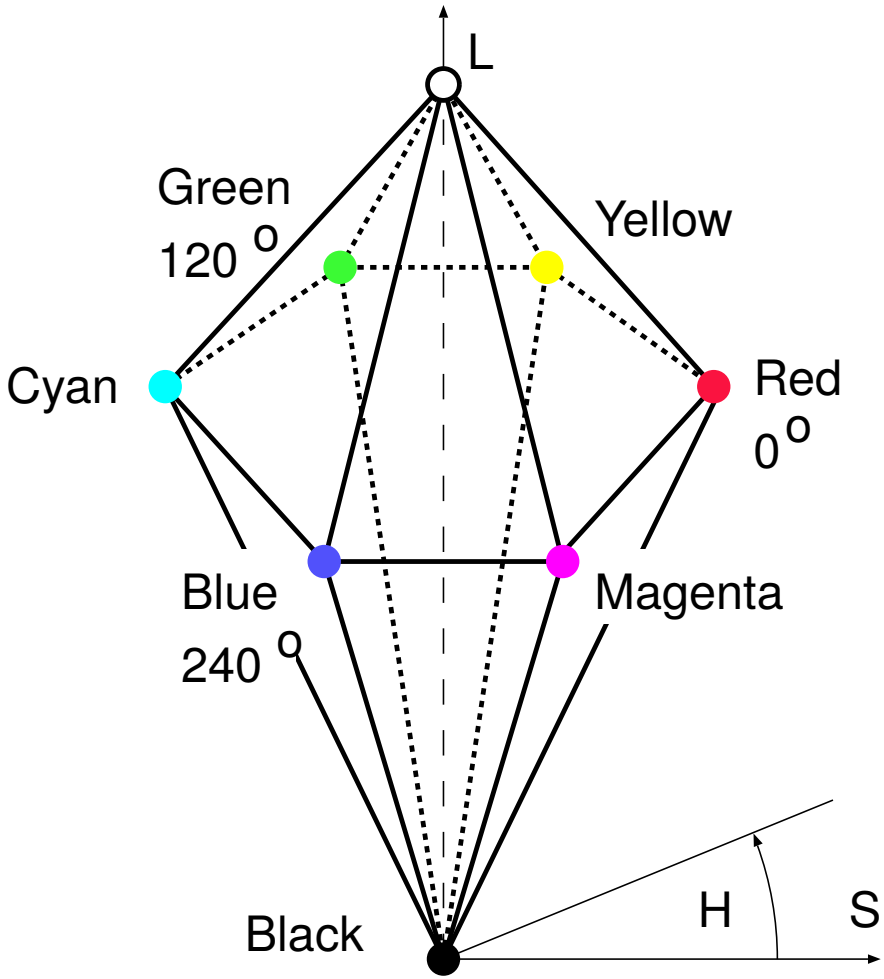
HSV-to-RGB Conversion

- A hexagonal cross-section of the HSV pyramid can be regarded as a sub-cube of the RGB cube projected onto a plane that is normal to its main diagonal.
- This establishes a one-to-one mapping between RGB and HSV.
- Thus, the arithmetic interpolation between two HSV triples is neither geometrically linear nor perceptually linear.



Hue, Lightness, Saturation

- Developed at Tektronix, the HLS model is very similar to HSV.
- It accounts for the fact that as light gets too bright or too dark, the range of perceivable colors narrows to only white or only black.



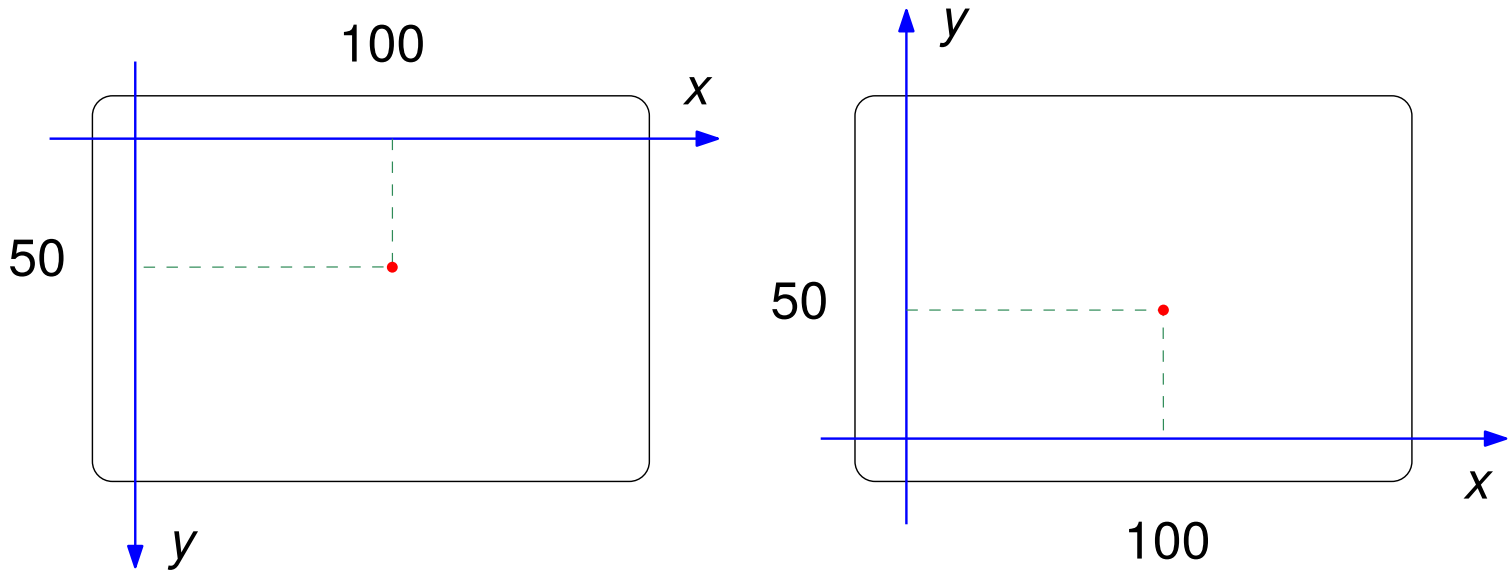
Raster Devices

- The most common graphics output device is the raster display.
- An image is generated by a 2D array of small dots or squares: *pixels* (shorthand for “picture elements”).
- Every pixel can be set individually; a typical (API) command might be
  - SetPixel(x,y,*color*)

  where x and y are pixel coordinates.
- Depending on the number of different color and intensity values of every pixel we distinguish among the following displays.
  - Monochrome display:
    Each pixel can either be on or off. Can only display one color.
    Typical device: b/w laser printer.
  - Grey-scale display:
    Each pixel can have one of n possible brightness values (“intensities”).
  - Color display:
    Each pixel can have one of 2^k possible colors (if k bits per pixel are used).
    Each pixel is composed of a cluster of single-color pixels that fool the eye.
    Typical device: color monitor.

Different Device Coordinate Systems

- Unfortunately not all systems adopt the same pixel addressing conventions: Some systems have the origin at the upper-left corner, some have it at the lower-left corner.

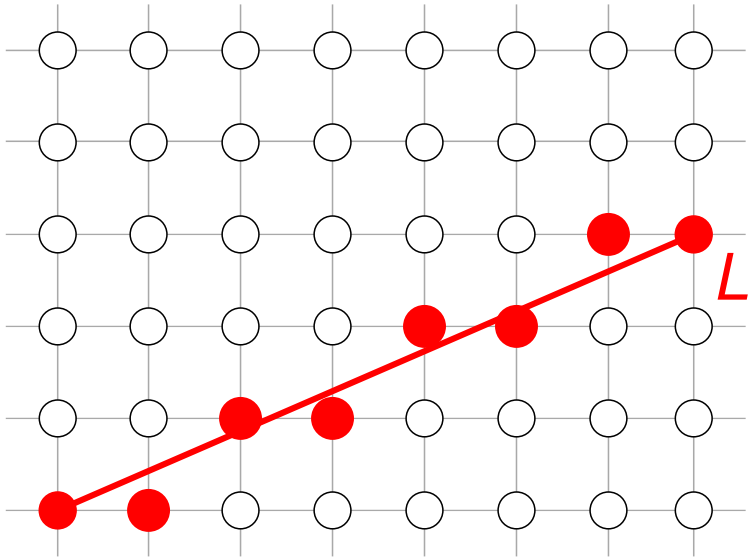


Warning

X11 has its coordinate origin in the upper-left corner, while OpenGL coordinates have their origin in the lower-left corner!

Drawing a Line

- We will always assume that the end-points of a straight-line segment L are given in integer coordinates relative to the resolution of the output device.
- Which pixels should be turned on?
- Pixels should be as close to L as possible!



Brute-Force Scan Conversion

- Consider a line segment L between $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$ and $\begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$, where $x_1, y_1, x_2, y_2 \in \mathbb{N}$, and $x_1 \leq x_2$ and $y_1 \leq y_2$.
- The equation of the supporting line is given by

$$y = s \cdot x + c,$$

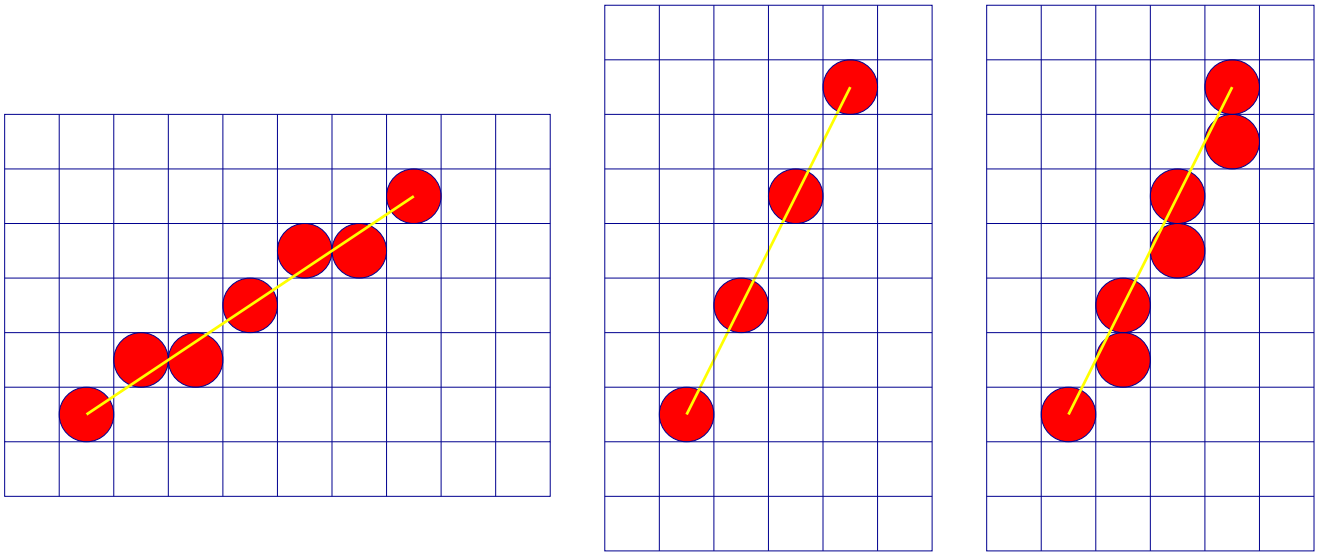
where

$$s = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{and} \quad c = \frac{y_1 \cdot x_2 - y_2 \cdot x_1}{x_2 - x_1}.$$

- We get a simple scan-conversion algorithm by incrementing x, computing the corresponding y, and rounding it to the nearest integer value.
- The algorithm involves floating-point arithmetic, and the rounding operation is expensive.
- In any case, this simple algorithm will not work particularly well ...

Handling Different Inclinations

- This simple algorithm will not work particularly well . . .
- We need a different algorithm depending on whether the change in y is bigger than the change in x.

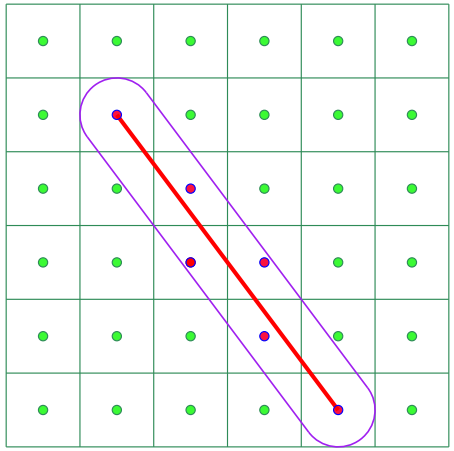


Specifying the Desired Output

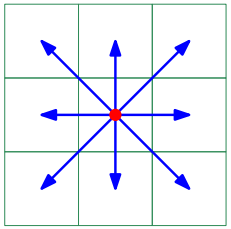
- The general goal is
  - to minimize the stair-case effect (“jaggies”) due to the replacement of a continuum of width zero by a discrete set of pixels of non-zero area,
  - to have a uniform line density,
  - to make sure that a line drawn is independent of whether we draw it from P_1 to P_2 or from P_2 to P_1,
  - to cast the conversion into an algorithm that is fast.
- The simple fact that a continuum is replaced by a discrete set of pixels is the source of many serious problems in graphics that are known as *aliasing problems*!
- The following two specifications are widely used:
  - The set of pixels whose centers are covered by a parallelogram of width 1 centered on the line.
  - The shortest sequence of eight-connected pixels that most closely approximate the line.

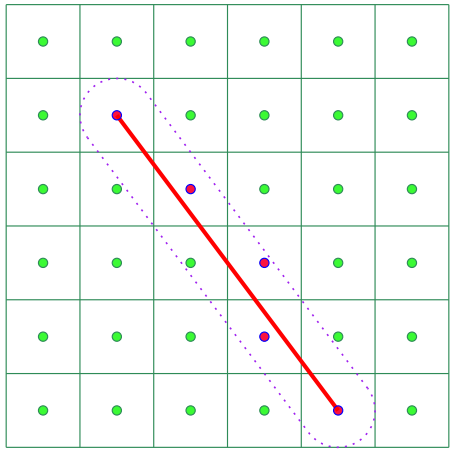
Specifying the Desired Output

Parallelogram Coverage: Select pixels within strip of width 1.



Eight-Connectedness: Used by Bresenham's algorithm.





UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Bresenham's Algorithm for Drawing a Line

- Developed in the early 1960s to control the pen movements of plotters.

Watch the details!

Be warned that several improvements to Bresenham's original algorithm have been proposed since its invention. So, by now, dozens of slightly different scan-conversion algorithms are denoted as "Bresenham's Algorithm".

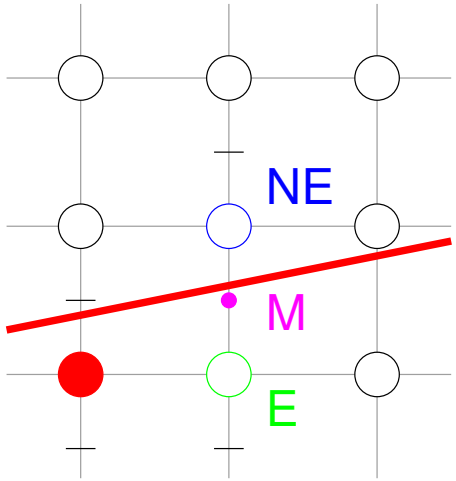
- The following description is limited to line segments that lie in the first octant, i.e.

$$y = s \cdot x + c \quad \text{where} \quad 0 \leq s \leq 1.$$

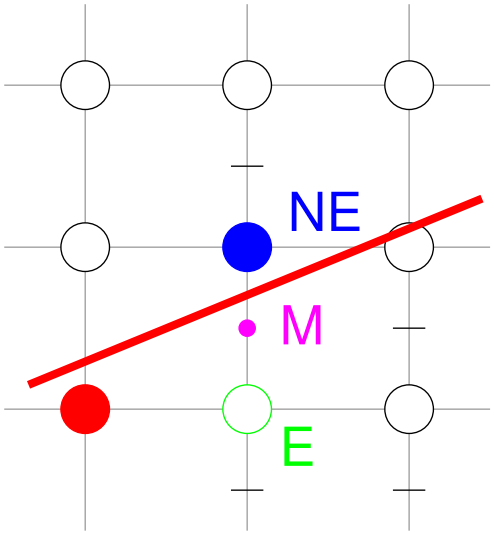
- Let $\Delta_x := x_2 - x_1$ and $\Delta_y := y_2 - y_1$, where $x_1, x_2, y_1, y_2 \in \mathbb{N}$, and $x_1 \leq x_2$ and $y_1 \leq y_2$. Furthermore, $\Delta_y \leq \Delta_x$.
- We will draw the segment from left to right.
- Assume that pixel (x_i, y_i) has been set. Which pixel is next? The pixel $(x_i + 1, y_i)$ or the pixel $(x_i + 1, y_i + 1)$?
- Remember that we seek an eight-connected set of pixels. Thus, $(x_i, y_i + 1)$ is no option once (x_i, y_i) was drawn.

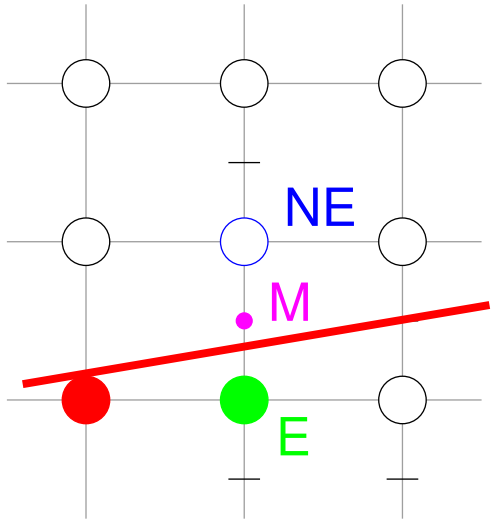
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Basic Idea of a Midpoint Algorithm



previous pixel | choices for current pixel | choices for next pixel





UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

The Mathematics of Bresenham's Line Algorithm

- In implicit form we get

$$F(x,y) := x\,\Delta_y - y\,\Delta_x + c\,\Delta_x$$

for the equation of the line through $\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}$ and $\begin{pmatrix} x_2 \\ y_2 \end{pmatrix}$, where

$$F(x,y) \left\{ \begin{array}{c} < \\ = \\ > \end{array} \right\} 0 \;\Leftrightarrow\; \left\{ \begin{array}{ll} (x,y) & \text{above line,} \\ (x,y) & \text{on line,} \\ (x,y) & \text{below line.} \end{array} \right.$$

- Bresenham's Algorithm always increments x. Whether or not y is incremented depends on the position of the next midpoint relative to the line.

$$e_i := F(x_i + 1, y_i + \frac{1}{2}) \left\{ \begin{array}{lllll} < 0: & x_{i+1} := x_i + 1, & y_{i+1} := y_i & & \text{(E)}, \\ \geq 0: & x_{i+1} := x_i + 1, & y_{i+1} := y_i + 1 & & \text{(NE)}. \end{array} \right.$$

The Mathematics of Bresenham's Line Algorithm

- Goal: derive the *error variable* e_{i+1} directly from the last error variable e_i.
- Case (E):

$$\begin{aligned} e_{i+1} = F(\underbrace{x_i+2}_{x_{i+1}+1}, \underbrace{y_i+\frac{1}{2}}_{y_{i+1}+\frac{1}{2}}) &= x_i\,\Delta_y + \Delta_y + \Delta_y - y_i\,\Delta_x - \tfrac{1}{2}\Delta_x + c\,\Delta_x \\ &= F(x_i+1, y_i+\tfrac{1}{2}) + \Delta_y = e_i + \Delta_y. \end{aligned}$$

- Case (NE):

$$\begin{aligned} e_{i+1} = F(\underbrace{x_i+2}_{x_{i+1}+1}, \underbrace{y_i+\frac{3}{2}}_{y_{i+1}+\frac{1}{2}}) &= x_i\,\Delta_y + \Delta_y + \Delta_y - y_i\,\Delta_x - \tfrac{1}{2}\Delta_x - \Delta_x + c\,\Delta_x \\ &= F(x_i+1, y_i+\tfrac{1}{2}) + \Delta_y - \Delta_x = e_i + \Delta_y - \Delta_x. \end{aligned}$$

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

The Mathematics of Bresenham's Line Algorithm

- The first error variable e_1 is initialized as

$$\begin{aligned} e_1 := F(x_1 + 1, y_1 + \tfrac{1}{2}) &= x_1\,\Delta_y + \Delta_y - y_1\,\Delta_x - \tfrac{1}{2}\Delta_x + c\,\Delta_x \\ &= \underbrace{F(x_1, y_1)}_{=0} + \Delta_y - \tfrac{1}{2}\Delta_x = \Delta_y - \tfrac{1}{2}\Delta_x. \end{aligned}$$

- For the purposes of Bresenham's algorithm we may replace $F(x, y)$ by $2F(x, y)$, thus eliminating the division by 2 in e_1:

$$e_1 = 2\Delta_y - \Delta_x,$$

$$e_{i+1} := \begin{cases} e_i + 2\Delta_y & \text{if (E)}, \\ e_i + 2\Delta_y - 2\Delta_x & \text{if (NE)}. \end{cases}$$

Bresenham's Line Algorithm

Algorithm *Bresenham*
Input: P_1, P_2: point
$(* \; P_1 = (x_1, y_1), P_2 = (x_2, y_2) \; *)$
1. var $x, y, \Delta_x, \Delta_y, error, c_1, c_2$:integer;
2. $\Delta_x \leftarrow x_2 - x_1$; $\Delta_y \leftarrow y_2 - y_1$;
3. $x \leftarrow x_1$; $y \leftarrow y_1$;
4. $c_1 \leftarrow 2 \cdot \Delta_y$; $error \leftarrow c_1 - \Delta_x$; $c_2 \leftarrow error - \Delta_x$;
5. **repeat**
6. SetPixel(x, y);
7. inc(x);
8. **if** $error < 0$ **then** $(*$ (E) $*)$
9. $error \leftarrow error + c_1$;
10. **else** $(*$ (NE) $*)$
11. inc(y);
12. $error \leftarrow error + c_2$;
13. **until** $x > x_2$;

Drawing a Circle

- The standard parameterization of a circle with radius r centered at the origin is given by

$$x(\varphi) := r\cos\varphi \quad \text{and} \quad y(\varphi) := r\sin\varphi,$$

for $0 \leq \varphi < 2\pi$.

- Discretization based on $\varphi_i := i \cdot \frac{2\pi}{n}$ for $0 \leq i \leq n-1$ yields

$$x_{i+1} := x(\varphi_{i+1}) = x(\varphi_i + \delta) = r(\cos\varphi_i \cos\delta - \sin\varphi_i \sin\delta) = x_i \cos\delta - y_i \sin\delta,$$

$$y_{i+1} := y(\varphi_{i+1}) = y(\varphi_i + \delta) = r(\sin\varphi_i \cos\delta + \cos\varphi_i \sin\delta) = x_i \sin\delta + y_i \cos\delta,$$

where

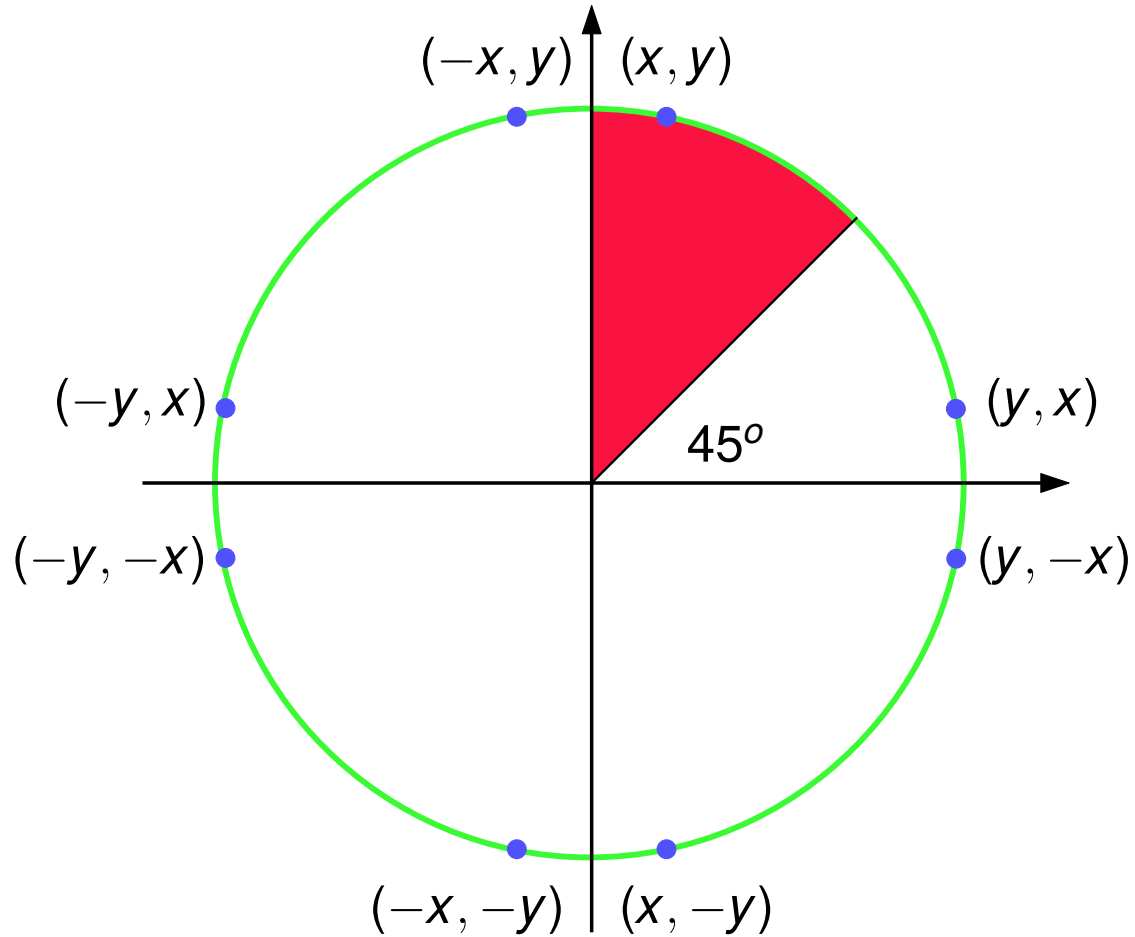
$$\delta := \frac{2\pi}{n} \quad \text{and} \quad x_0 := r \quad \text{and} \quad y_0 := 0.$$

- A brute-force scan-conversion algorithm for circular arcs and ellipses is easily derived from these equations.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Bresenham's Algorithm for Drawing a Circle

- We consider the second octant of circles with integer radius centered at the origin.
- The circular arc is drawn from 90^o to 45^o!
- Use symmetry to draw the other portions of the circle.

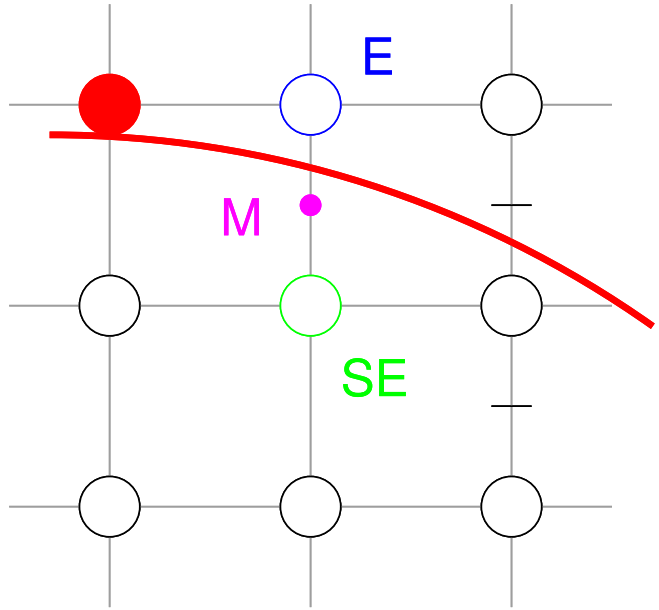


The Mathematics of Bresenham's Circle Algorithm

- We have

$$F(x,y) := x^2 + y^2 - r^2 \left\{ \begin{array}{l} > 0 \\ = 0 \\ < 0 \end{array} \right\} \Leftrightarrow \left\{ \begin{array}{ll} (x,y) & \text{outside the circle,} \\ (x,y) & \text{on the circle,} \\ (x,y) & \text{inside the circle.} \end{array} \right.$$

- Once again, we use the idea of a midpoint algorithm.
- If the midpoint lies inside the circle then (E) else (SE).



previous pixel | current choice for pixel | next pixel

The Mathematics of Bresenham's Circle Algorithm

- The error variable is defined as

$$e_i := F(x_i + 1, y_i - \frac{1}{2}) = (x_i + 1)^2 + (y_i - \frac{1}{2})^2 - r^2.$$

- Furthermore,

$$e_i \begin{cases} < 0: & x_{i+1} := x_i + 1, & y_{i+1} := y_i & (E), \\ \geq 0: & x_{i+1} := x_i + 1, & y_{i+1} := y_i - 1 & (SE). \end{cases}$$

- We get

$$\begin{aligned} \text{(E):} \quad e_{i+1} &= F(x_i + 2, y_i - \tfrac{1}{2}) = (x_i + 2)^2 + (y_i - \tfrac{1}{2})^2 - r^2 \\ &= F(x_i + 1, y_i - \tfrac{1}{2}) + 3 + 2x_i = e_i + 2x_i + 3 \\ &= e_i + 2x_{i+1} + 1. \end{aligned}$$

$$\begin{aligned} \text{(SE):} \quad e_{i+1} &= F(x_i + 2, y_i - \tfrac{3}{2}) = \ldots = e_i + 2x_i - 2y_i + 5 \\ &= e_i + 2x_{i+1} - 2y_{i+1} + 1. \end{aligned}$$

The Mathematics of Bresenham's Circle Algorithm

- The first error variable is initialized as $e_1 := F(1, r - \frac{1}{2}) = \frac{5}{4} - r$.
- Once again, we substitute $F(x, y)$ by $2F(x, y)$. Also, we add $\frac{1}{2}$ to e_1. Thus, we end up with

$$e_{i+1} = \begin{cases} e_i + 4x_{i+1} + 2 & \text{(E)}, \\ e_i + 4x_{i+1} - 4y_{i+1} + 2 & \text{(SE)}. \end{cases}$$

and

$$e_1 := 3 - 2r.$$

Bresenham's Circle Algorithm

Algorithm *Bresenham*
Input: *rad* : integer

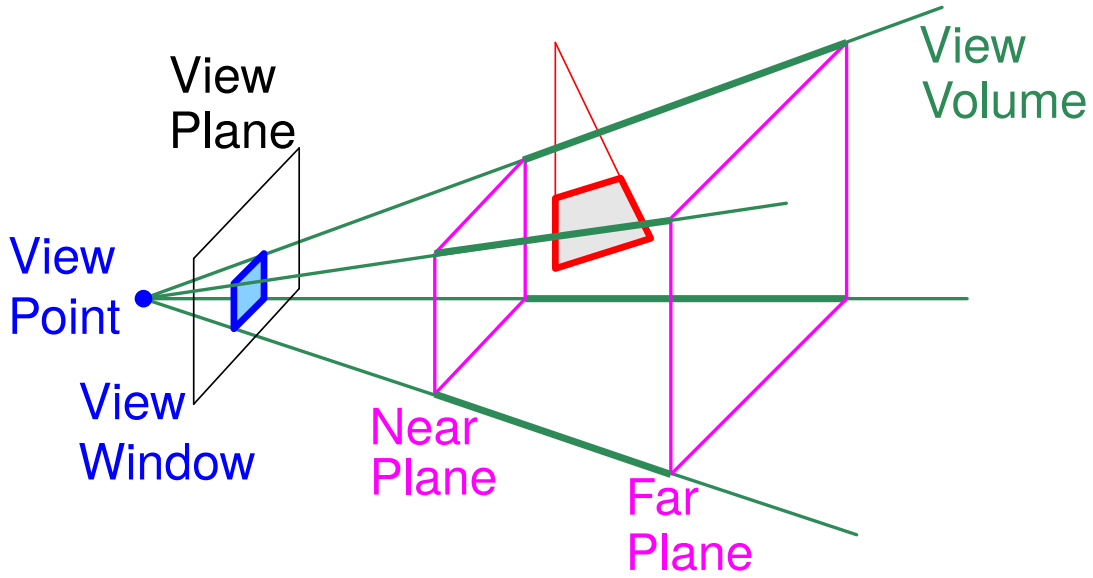
```
1.   var x, y, error : integer;
2.   x ← 0; y ← rad;
3.   error ← 3 − 2rad;
4.   while x ≤ y do
5.     SetPixel(x, y);
6.     inc(x);
7.     if error ≥ 0 then
8.       dec(y);
9.       error ← error − 4y;
10.    error ← error + 4x + 2;
```

4 Basic Rendering Techniques

- Clipping
- Hidden-Surface Removal
- Illumination Model
- Incremental Shading Techniques
- Aliasing and Anti-Aliasing
- Texture Mapping

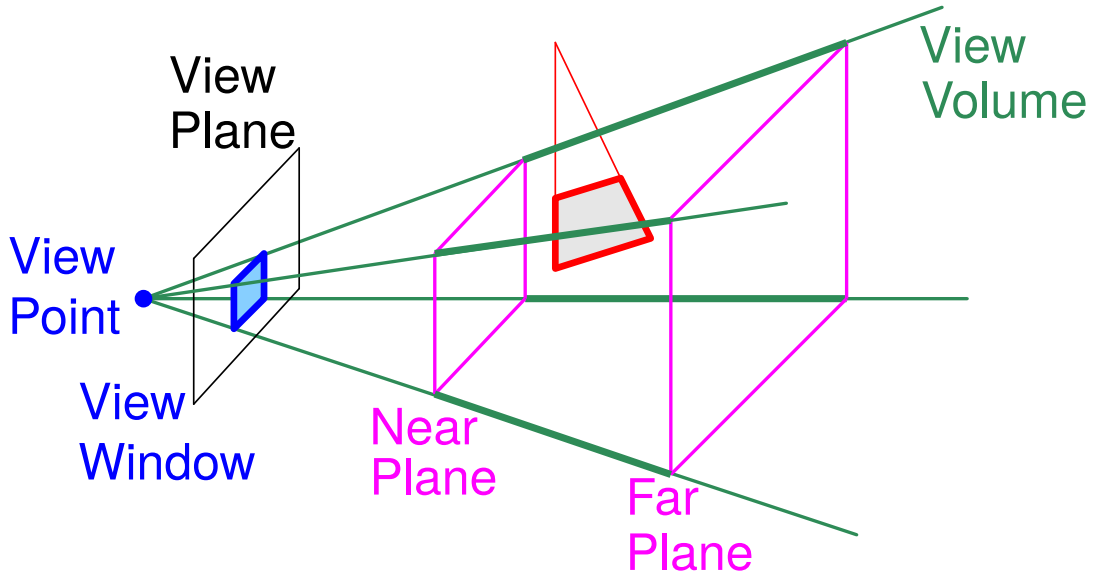
View Volume and Clipping

- The portion of the *view plane* (or *image plane*) that we are interested in is defined by the *view window*.
- Together with the *view point* the view window defines a pyramid-shaped portion of the space: the so-called *view volume* (or *view frustum*).
- Typically, a *near plane* (or *front plane*) and a *far plane* (or *back plane*) are added in order to exclude objects from being projected that are very far from or very close to the viewer.
- The process of restricting an object to the view volume/window is called *clipping*.



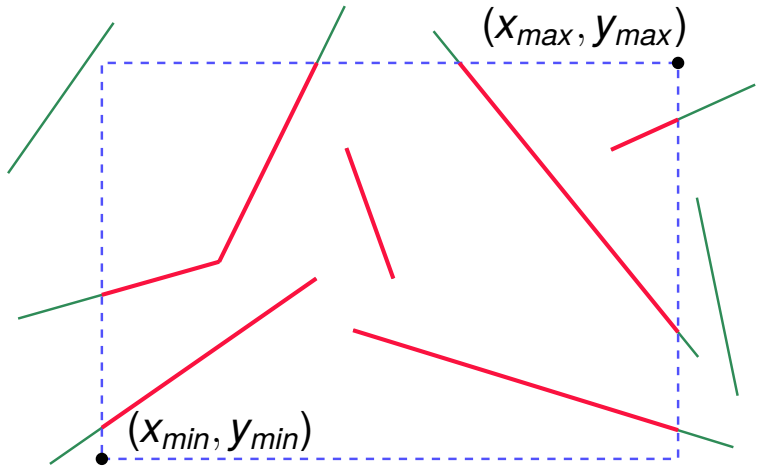
View Volume and Clipping

- The view volume is a frustum of a pyramid in the case of perspective projection, and a parallelepiped in the case of parallel projection.
- Since clipping objects to a box is much simpler than clipping to a genuine pyramid, it is common to apply a perspective normalization in order to convert a perspective projection into an orthographic projection.
- Clipping can be performed in 3D prior to projecting the objects onto the view plane, or in 2D after projecting the objects onto the view plane.



Clipping in 2D

- The task of clipping is to replace line segments with shorter segments that fit neatly into the view/clip window.



- Clipping can be performed in object space or in image space:

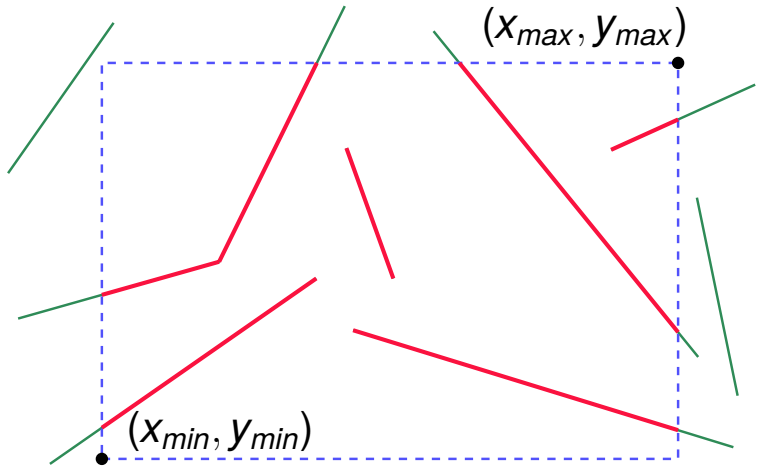
  Object-space clipping: Compute intersections analytically, and scan-convert only clipped segments.

  Image-space clipping: Scan-convert full segments and perform point clipping afterwards.

- We assume that the window is given by the axis-parallel rectangle $W := [x_{min}, x_{max}] \times [y_{min}, y_{max}]$.
- Point clipping:
  - $x_{min} \leq x \leq x_{max}$?
  - $y_{min} \leq y \leq y_{max}$?

Line Clipping

- Consider the clipping of a line segment $\ell := \overline{AB}$.

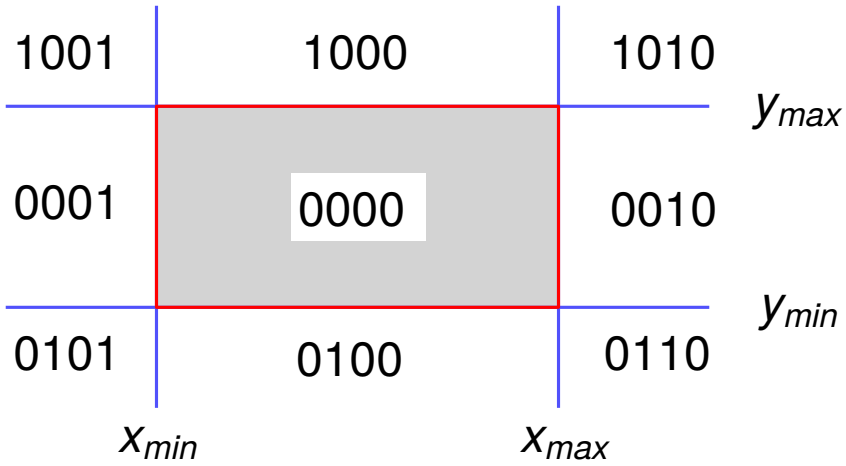


- Cases:
  - $A \in W, B \in W$: Accept ℓ.
  - $A \in W, B \notin W$: Compute $P := \ell \cap \partial W$, accept $\overline{AP}$.
  - $A \notin W, B \in W$: Compute $P := \ell \cap \partial W$, accept $\overline{BP}$.
  - $A \notin W, B \notin W$:
    - If A, B lie outside the same boundary line of W then reject ℓ.
    - Otherwise, a more complicated test is needed ....

Cohen-Sutherland Algorithm

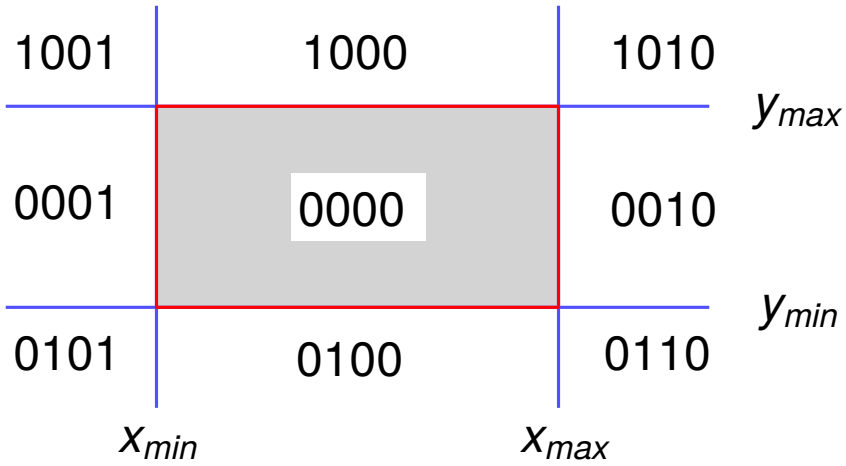
- We classify the position of every point P with respect to the supporting lines of W by assigning a 4-bit *out code*, $OC(P)$, as follows:

  0001 : P to the left of $x = x_{min}$.
  0010 : P to the right of $x = x_{max}$.
  0100 : P below of $y = y_{min}$.
  1000 : P above of $y = y_{max}$.



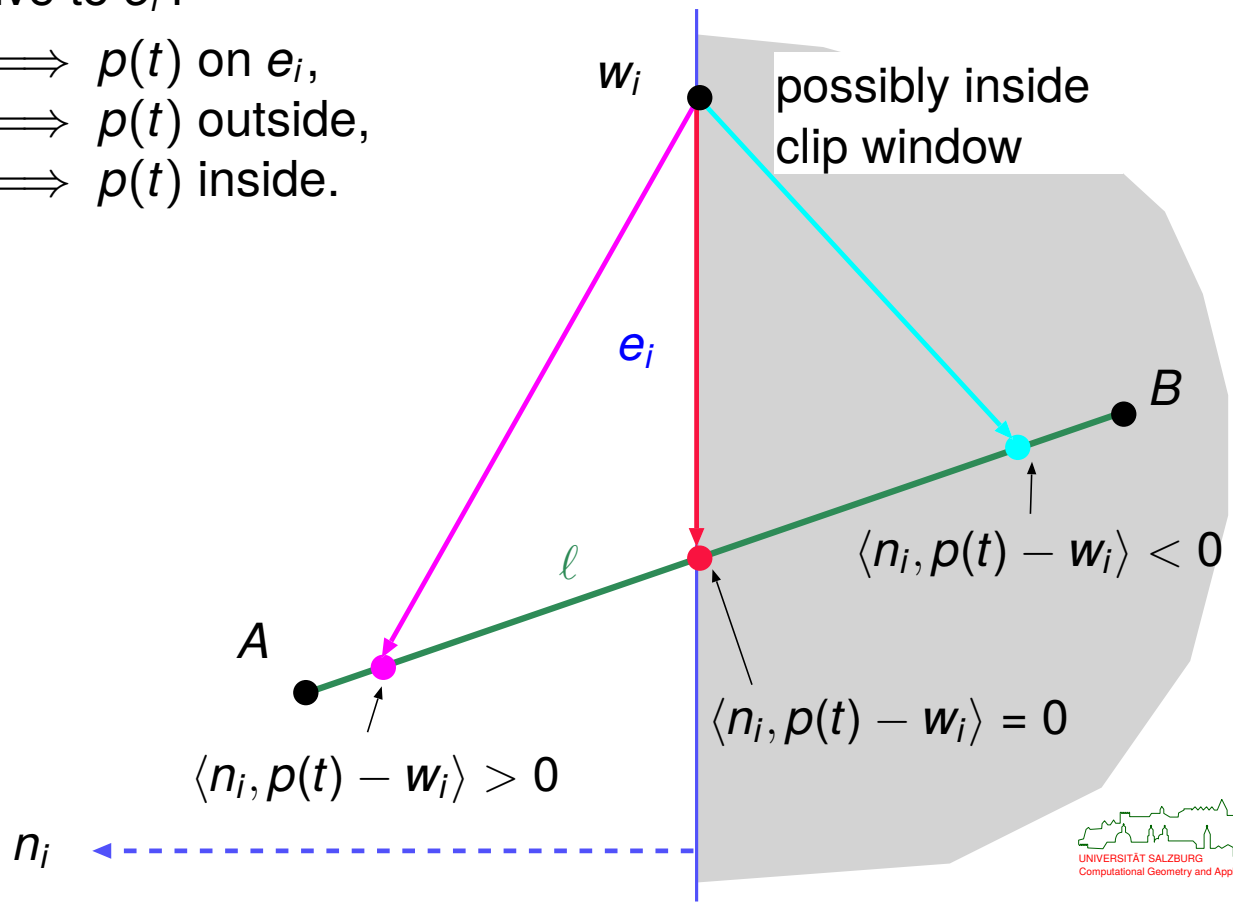
Cohen-Sutherland Algorithm

- If $OC(A) \mid OC(B) = 0000$ then accept ℓ, where $\mid$ is the bitwise OR operator.
- If $OC(A) \,\&\, OC(B) \neq 0000$ then reject ℓ, where & is the bitwise AND operator.
- Otherwise:
  - If $OC(A) = 0000$ then swap A and B.
  - Find the rightmost bit i such that $OC_i(A) = 1$.
  - Compute intersection P of ℓ with the supporting line of W which defines bit i.
  - Let $A := P$, and update $OC(A)$.
- Repeat the above steps until ℓ is accepted or rejected.
- Easily generalized to more general convex clip windows.



Cyrus-Beck-Liang-Barsky Algorithm

- We consider the standard parameterization $p(t) := (1-t)A + tB$, with $t \in [0,1]$.
- Goal: Compute $t_E, t_L \in [0,1]$ such that $W \cap \ell = \overline{p(t_E)p(t_L)}$.
- We pick an arbitrary point w_i on clip edge e_i, and denote the outwards normal vector of e_i by n_i.
- Where does $p(t)$ lie relative to e_i?
  - $\langle n_i, p(t) - w_i \rangle = 0 \iff p(t)$ on e_i,
  - $\langle n_i, p(t) - w_i \rangle > 0 \iff p(t)$ outside,
  - $\langle n_i, p(t) - w_i \rangle < 0 \iff p(t)$ inside.



Cyrus-Beck-Liang-Barsky Algorithm

- Thus, the equation for the intersection of ℓ with e_i is

$$\langle n_i, p(t_i) - w_i \rangle = 0.$$

- Suppose that $d := b - a \neq 0$. We get

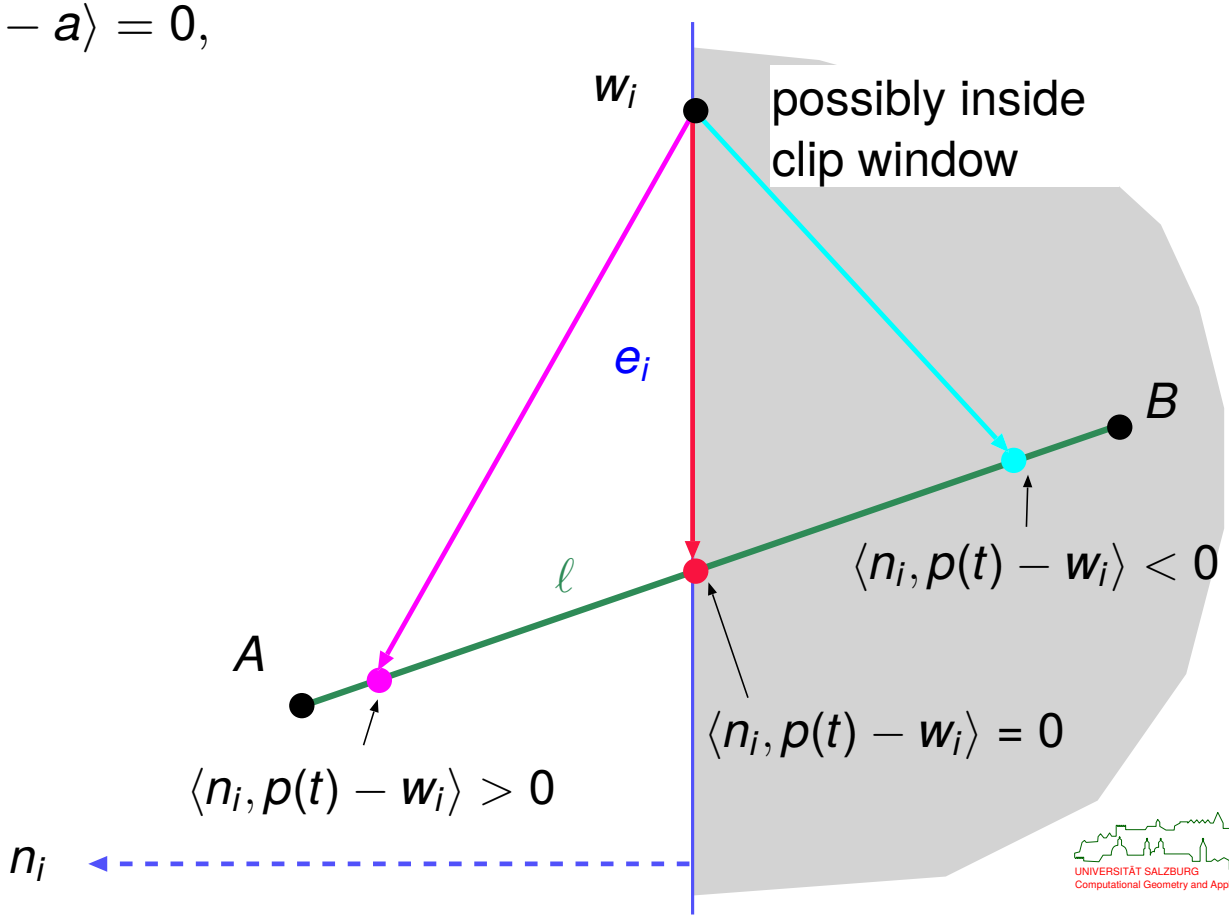
$$\langle n_i, a - w_i \rangle + t_i \langle n_i, b - a \rangle = 0,$$

that is

$$t_i = \frac{\langle n_i, a - w_i \rangle}{-\langle n_i, d \rangle}$$

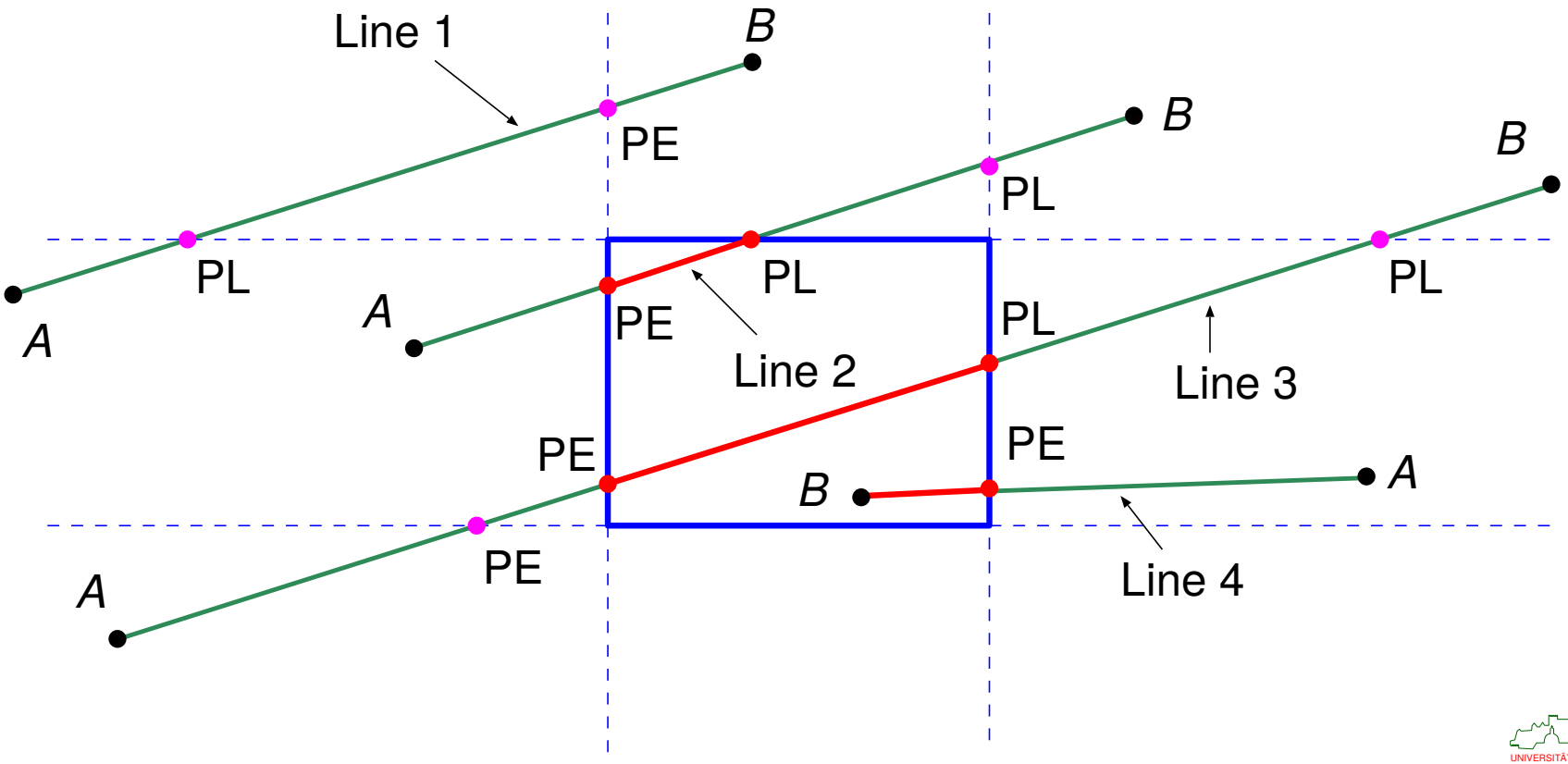
if $\langle n_i, d \rangle \neq 0$.

- If $\langle n_i, d \rangle = 0$ then $e_i \parallel \ell$.



Cyrus-Beck-Liang-Barsky Algorithm

- We define the predicates "'potentially entering"' (PE) and "'potentially leaving"' (PL) for each intersection:
  - $(PL)_i \iff \langle n_i, d \rangle > 0$,
  - $(PE)_i \iff \langle n_i, d \rangle < 0$.



Cyrus-Beck-Liang-Barsky Algorithm

- This yields the parameters t_E, t_L as

$$t_E := \max\left(\{t_i : (PE)_i\} \cup \{0\}\right),$$

$$t_L := \min\left(\{t_i : (PL)_i\} \cup \{1\}\right),$$

where

$$t_i := \frac{\langle n_i, a - w_i \rangle}{-\langle n_i, d \rangle}.$$

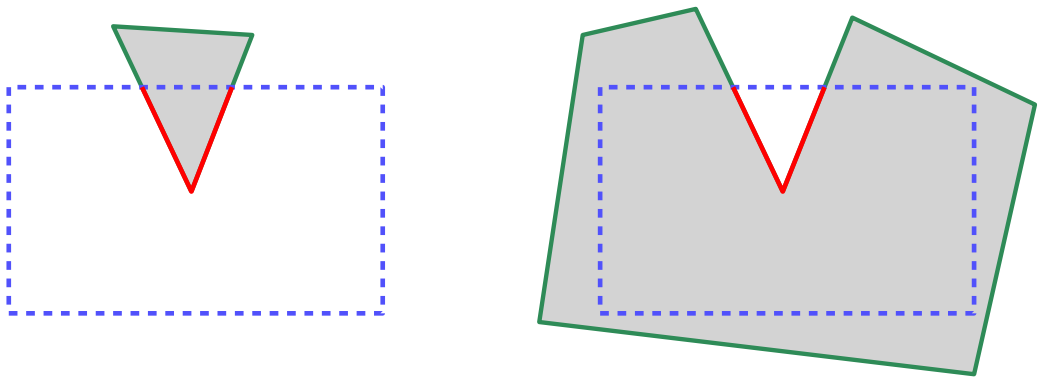
- If the clip edge e_i is parallel to a coordinate axis then the term for t_i is much simpler. E.g., for $e_i \equiv x = x_{min}$ we get

$$\langle n_i, d \rangle = a_x - b_x \quad \text{and} \quad t_i = \frac{a_x - x_{min}}{a_x - b_x}.$$

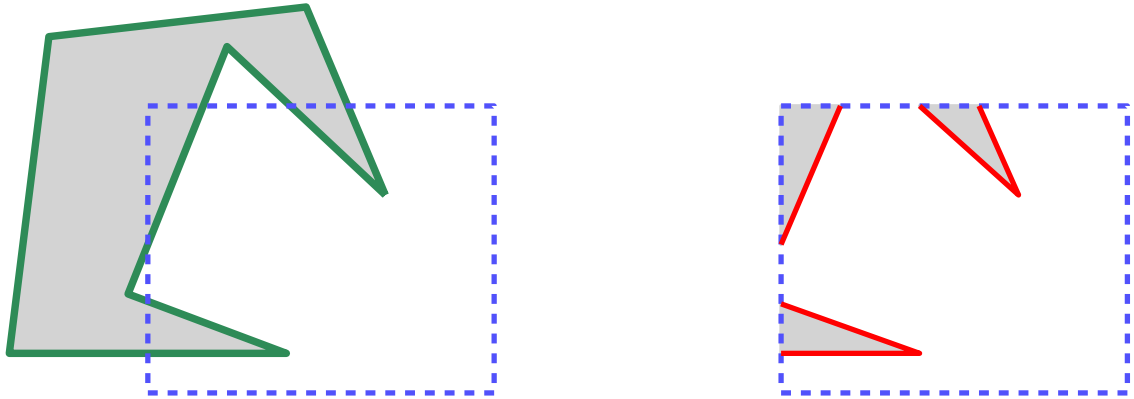
- Similar to the Cohen-Sutherland algorithm, also the CBLB algorithm can easily be generalized to general convex clip windows.

Polygon Clipping

- Clipping the edges of a polygon individually is not good enough.

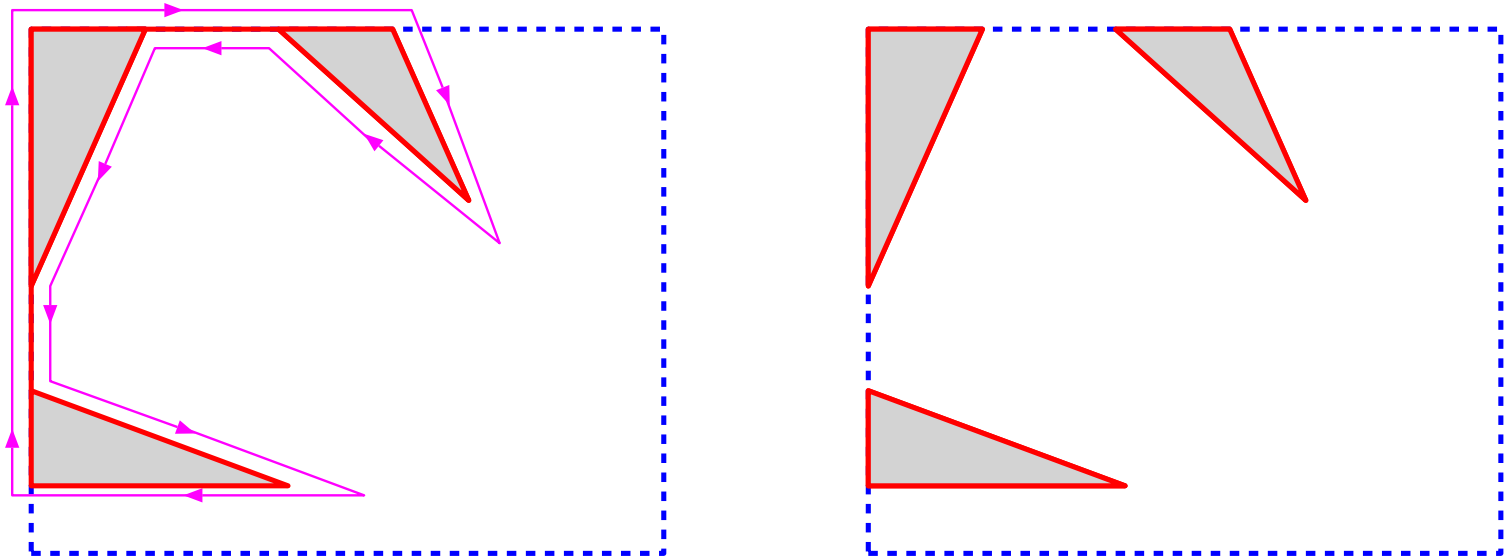


- Note that the polygon clipped may be disconnected!



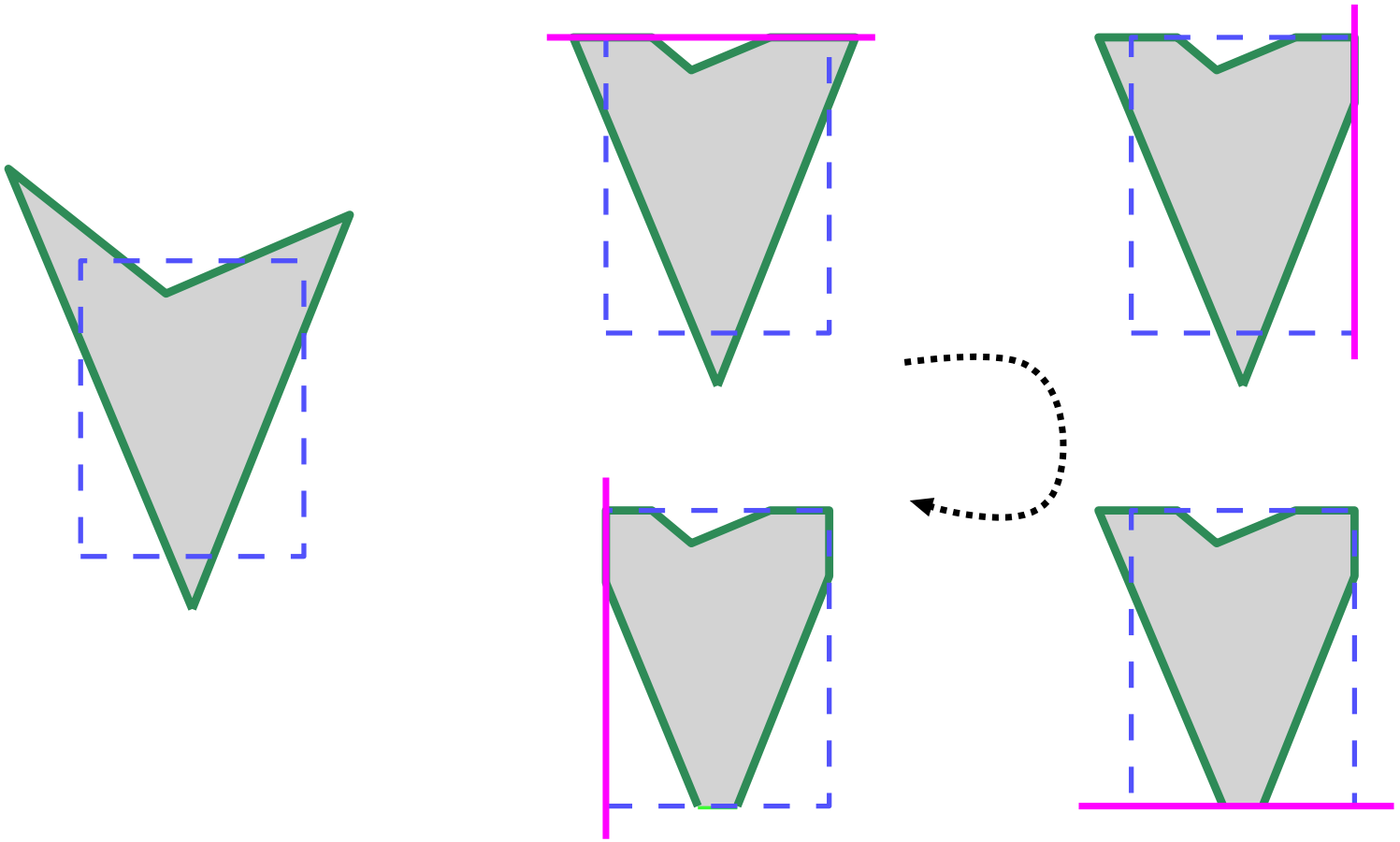
Polygon Clipping

- Two main options, and none of them is ideal . . .



Sutherland-Hodgeman Algorithm

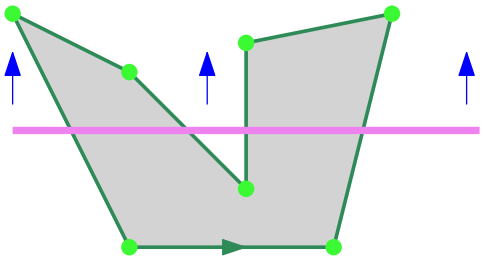
- A conventional line-clipping algorithms would clip one edge of the polygon with respect to the entire clip window, and would loop over all edges of the polygon.
- The Sutherland-Hodgeman Algorithm clips the entire polygon with respect to one clip edge of the clip window, and loops over all clip edges.

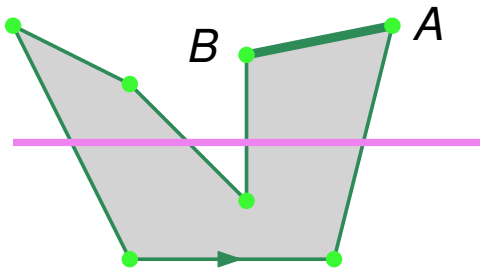


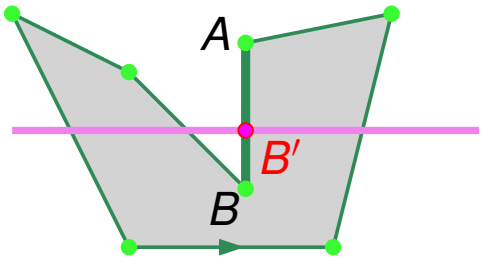
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

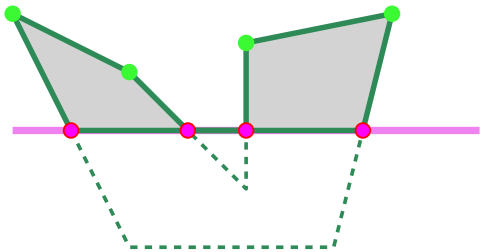
Sutherland-Hodgeman Algorithm

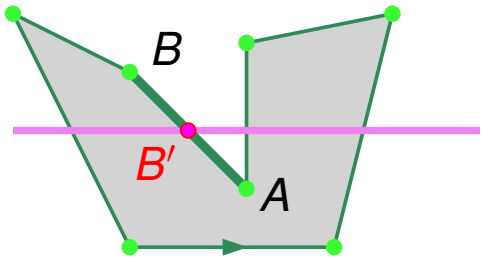
- We distinguish four cases, depending on how the start-point A and end-point B of an edge E are located relative to the clip edge.

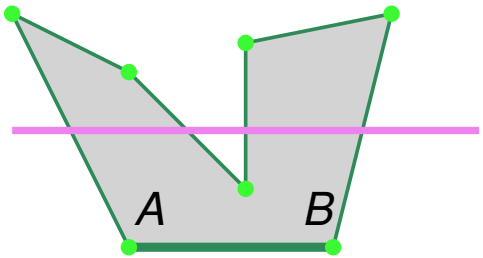












Clipping in 3D

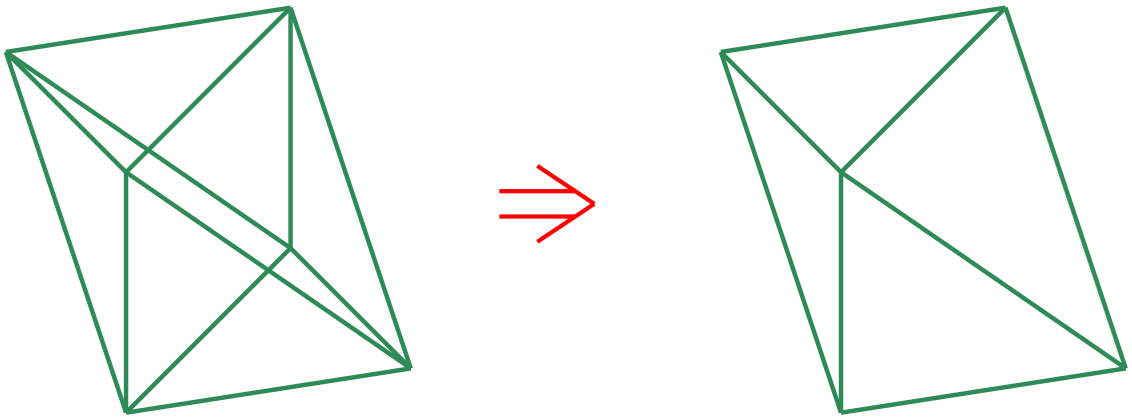
- The Cohen-Sutherland Algorithm and the Cyrus-Beck-Liang-Barsky Algorithm are straightforward to extend to 3D.
- E.g., for the Cohen-Sutherland Algorithm it suffices to maintain a six-bit out code, where Bit 5 is set if $p_z < z_{min}$, and Bit 6 is set if $p_z > z_{max}$.

Invisible Polygons

- A point P_1 is *occluded* by a point P_2 if
  - P_1 and P_2 project onto the same point in the view plane.
  - P_2 is closer to the viewer than P_1.
- Why can a polygon be invisible?
  - The polygon is occluded by some other polygon(s) that are closer to the viewer.
  - The polygon is outside of the view frustum.
  - The polygon is on the back side (with respect to the viewer) of an opaque object.
- For the sake of efficiency, we want to know whether a polygon is outside of the view frustum: *view frustum culling*!
- Also for the sake of efficiency, we want to know whether a polygon is occluded by some other polygon(s): *occlusion culling*!
- For correctness reasons, we need to know when a polygon is invisible — *hidden-surface removal* (HSR) and *visible-surface determination*!

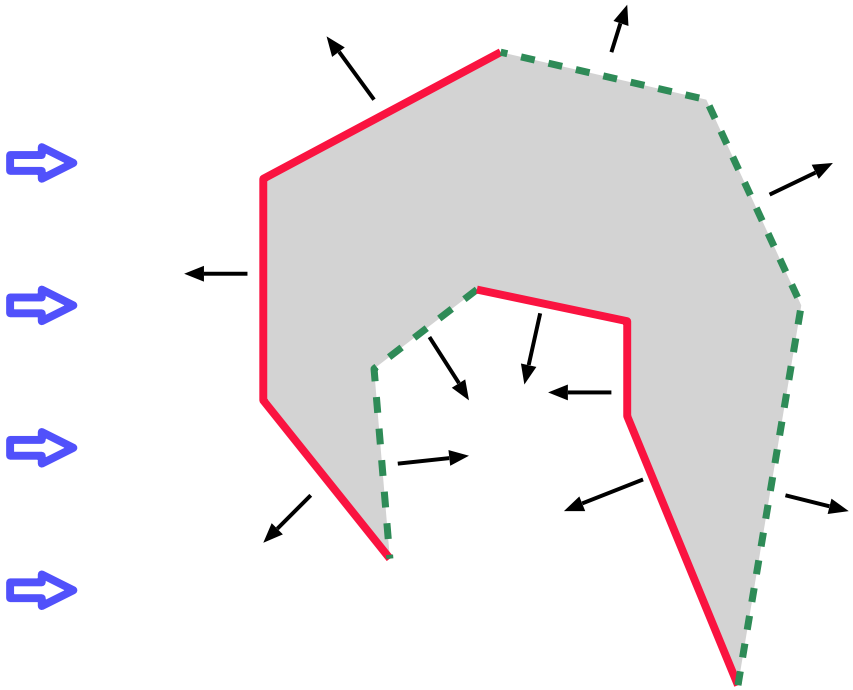
Invisible Polygons

- In the subsequent slides on hidden-surface removal, we will always assume that the canonical orthographic projection from $z = -\infty$ onto the xy-plane is used.
- Recall that a coordinate transformation and perspective normalization suffices to transform any projection to this canonical set-up.
- Thus, P_1 occludes P_2 exactly if $x_1 = x_2$ and $y_1 = y_2$ and $z_1 < z_2$.
- For simple rendering of complex scenes, hidden-surface removal accounts for a substantial portion of the total rendering time. Thus, efficiency is a key issue!
- A major step on the path to efficiency is to avoid sending many polygons to the HSR algorithm that are “obviously” not visible — and this is true even for GPU-based HSR!



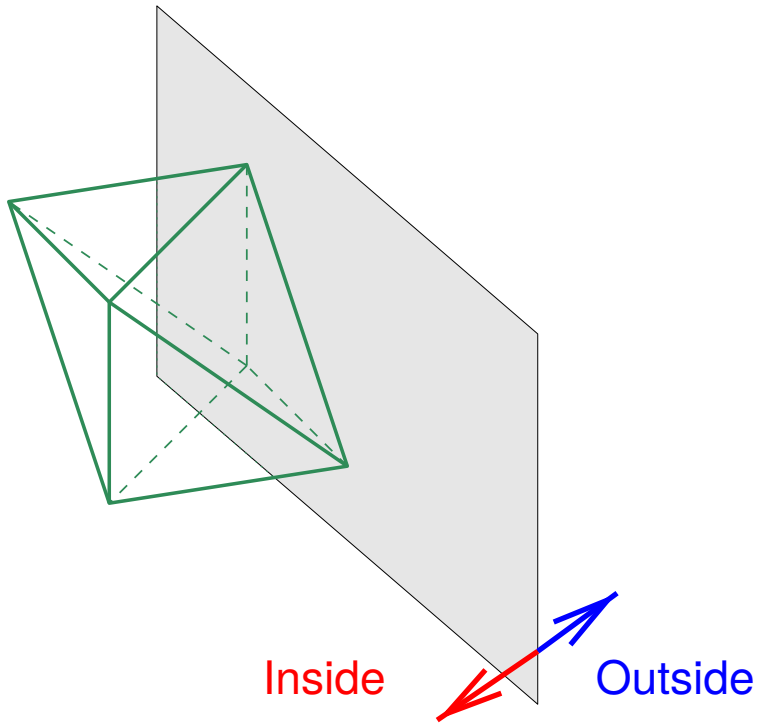
Back-Face Culling

- On a closed manifold surface, polygons whose exterior normal vectors point away from the viewer are always invisible.
- The process of removing all back-facing polygons is called *back-face culling* or *back-face removal*.
- Note that back-face culling does, in general, not solve the HSR problem for a non-convex object!
- Also, note that back-face culling may not be applied if the surface is not a closed manifold!



Back-Face Culling

- Consider the supporting plane of a polygon. To check whether the polygon is back-facing under a parallel projection, it suffices to check whether the view point is in the “inside half-space”.
- For a general parallel projection, this test is quickly performed by computing the sign of the dot product between the normal vector n and the viewing direction.
- For the canonical orthographic projection, this test boils down to checking whether $n_z > 0$.



Object Space versus Image Space

Object-Space HSR Algorithm:

- For each polygon σ in the scene:
  1. Compute the visible portion σ' of σ analytically.
  2. Project σ' onto the image plane and scan-convert the projection.
  3. Use σ to assign the appropriate color to each pixel corresponding to σ'.

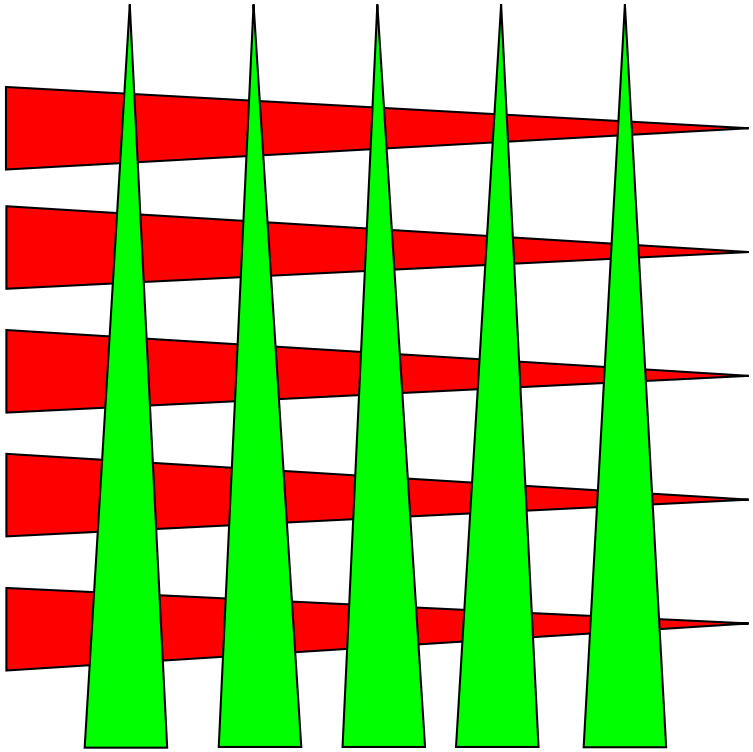
Image-Space HSR Algorithm:

- For each pixel π in the image plane:
  1. Find the polygon σ closest to the view point that is pierced by the projector through π.
  2. Use σ to assign the appropriate color to this pixel.

- Several algorithms employ a hybrid strategy: the polygons are re-arranged and subdivided in object space, until drawing them in proper order suffices to solve the visibility problem in image space. E.g., Binary Space Partitioning.
- Even in the presence of hardware support (mostly for image-space algorithms), there still is a need for object-space algorithms!

Object Space Versus Image Space

- A lower bound on the worst-case complexity of object-space algorithms is $\Omega(n^2)$ for a scene consisting of n polygons.

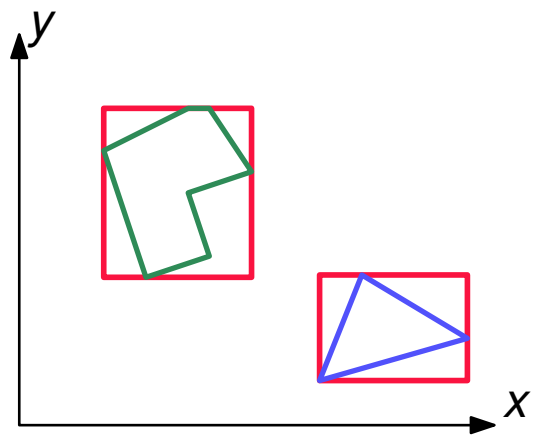


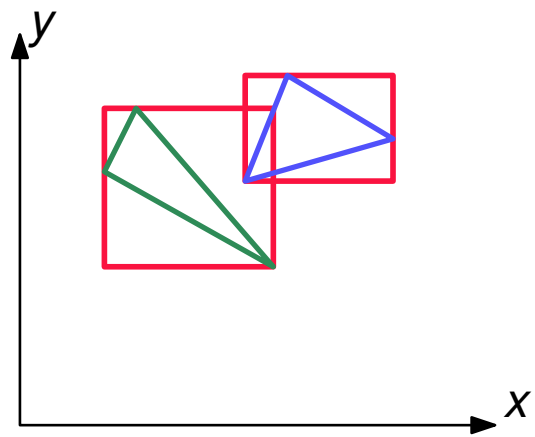
- A lower bound on the worst-case complexity of image-space algorithms is $\Omega(n \cdot p)$, where p denotes the resolution of the image.
- Note that both Ω-terms are not very realistic for practical applications, and that they hide multiplicative constants!

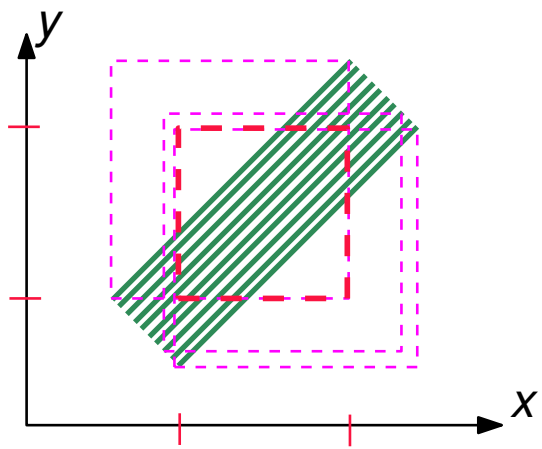
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Bounding-Box Test

- The *(axis-aligned) bounding box* (AABB) of an object is the smallest axis-aligned rectangle (in 2D) or box (in 3D) that encloses the object.
- If the bounding boxes do not intersect then the objects do not intersect.
- If two objects intersect then their bounding boxes intersect.
- No conclusion is possible if the bounding boxes intersect each other.
- Obvious problem: A lot of bounding boxes might intersect even if their objects do not intersect.
- This idea can be generalized to other bounding volumes, like oriented bounding boxes (OBB), discrete-orientation polytopes (k-dop), spheres, etc.

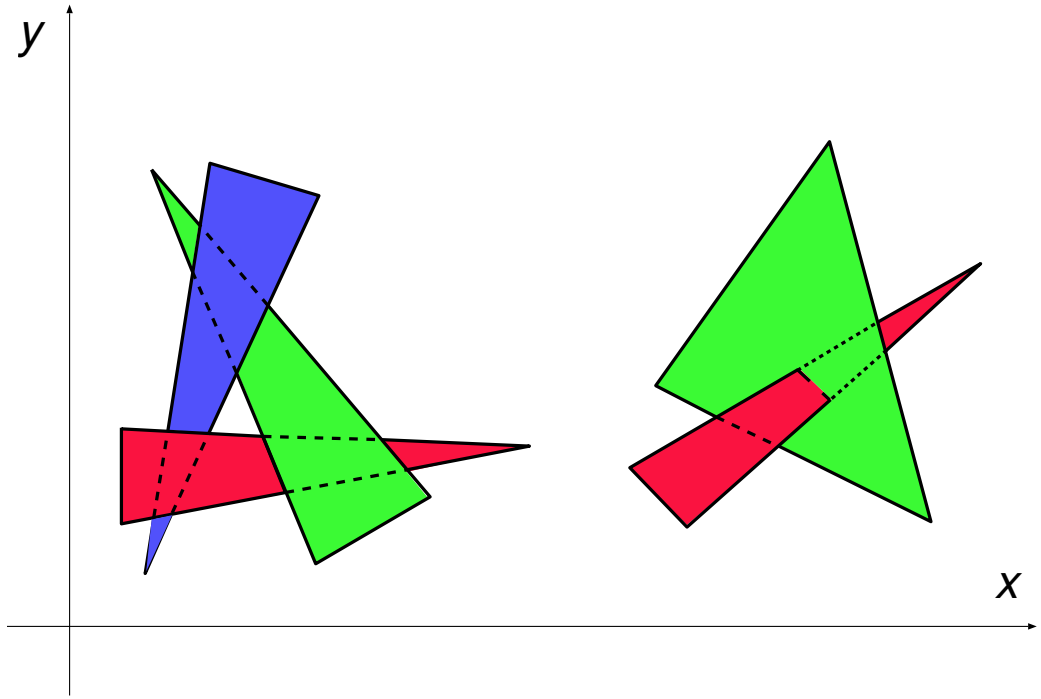






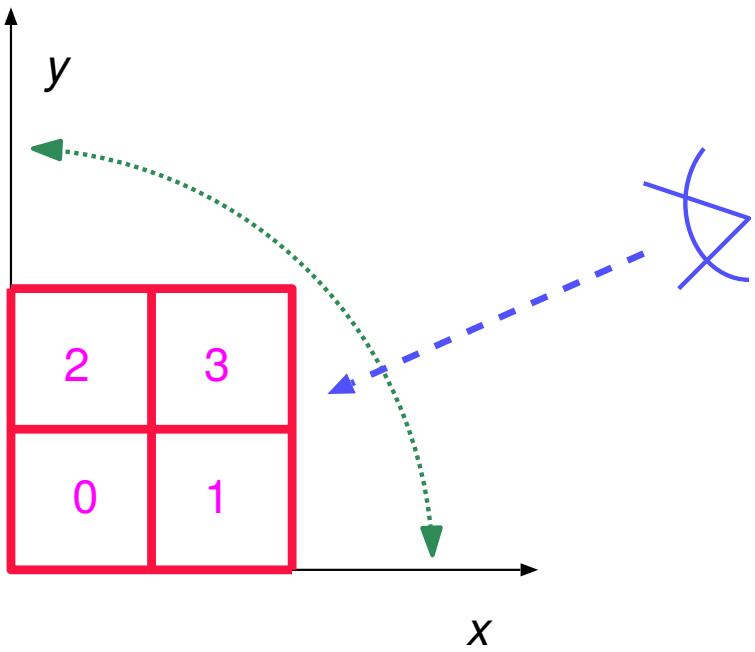
Painter's Algorithm

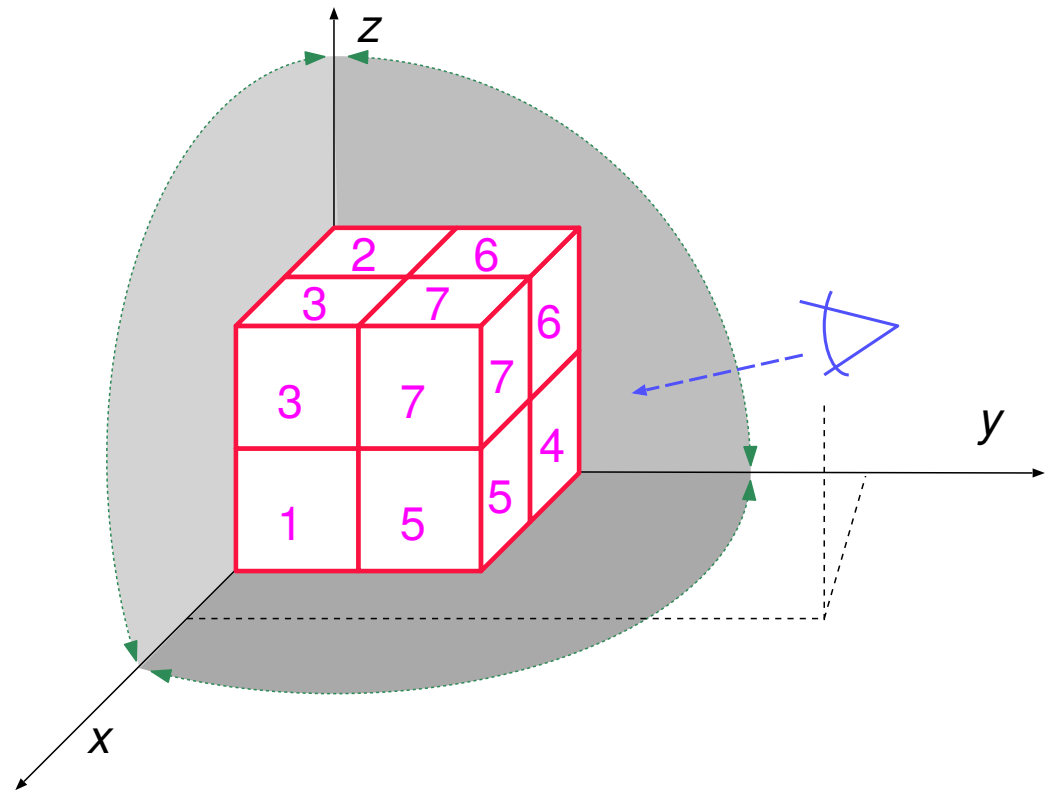
- For objects that can be placed in an *occlusion-compatible order*, the *painter's algorithm* combined with *depth sorting* is sufficient:
  1. Sort all of the potentially visible polygons by decreasing z-coordinates.
  2. Draw them in this order.
  3. Polygons in front of other polygons will be drawn later, so they will be visible, and they will occlude the polygons behind them.
- The painter's algorithm fails for polygons that interpenetrate each other, or in the case of cyclic overlaps.



Hidden-Surface Removal for Octrees

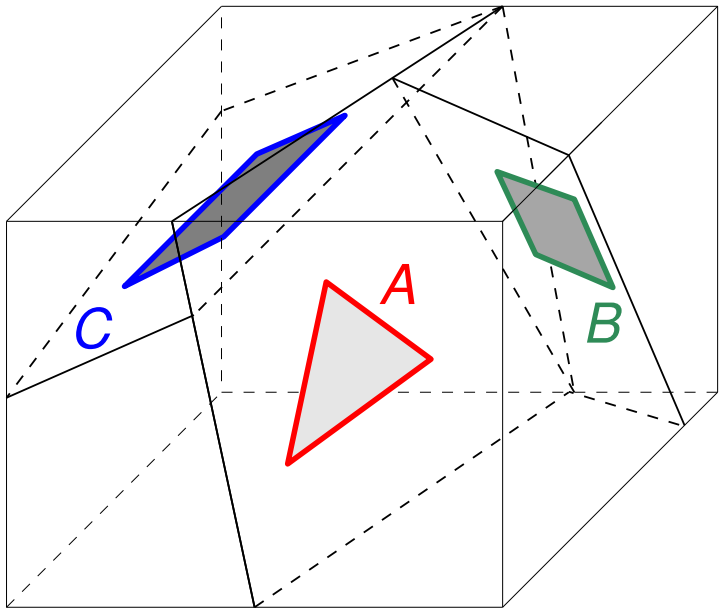
- Recursively visit and draw the cells of an octree in the proper order.

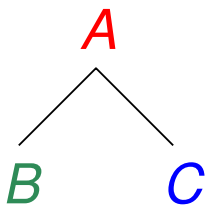




Binary Space Partitioning

- A *binary space partition tree* (BSP tree) subdivides 3D space along the supporting planes of particular polygons [Fuchs&Kedem&Naylor 1980].

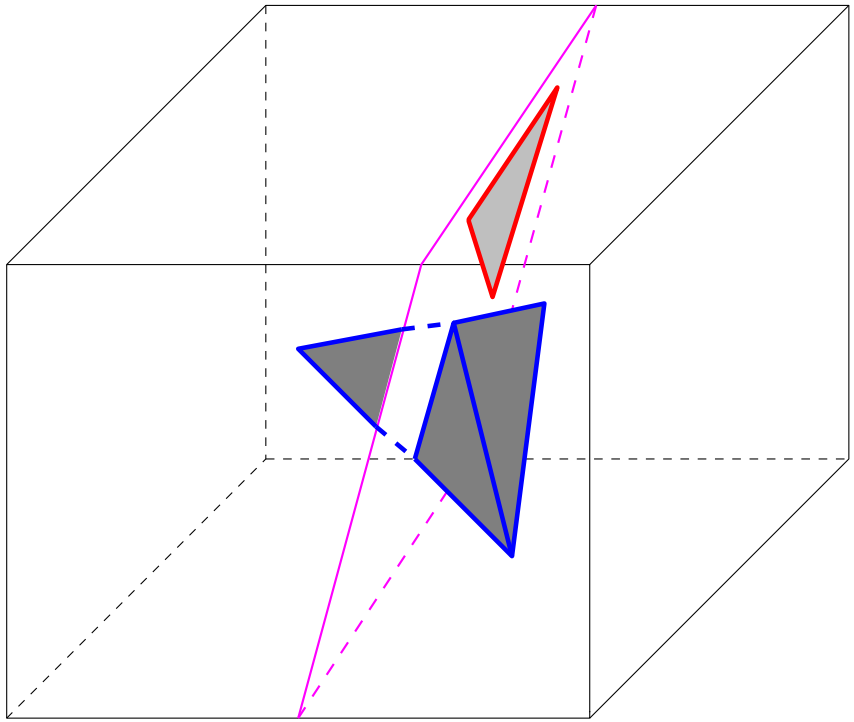




- Appropriately traversing this tree enumerates the polygons from back to front.
- Analogously for 2D and edges of polygons.

Constructing BSP Trees

- BSP trees are constructed much like Quicksort works.
- Suppose that all polygons are triangular.
- We use the supporting plane of a (randomly selected) triangle to split the space into triangles on one side and triangles on the other side.
- Triangles must be split (and re-triangulated) if they cross the plane.



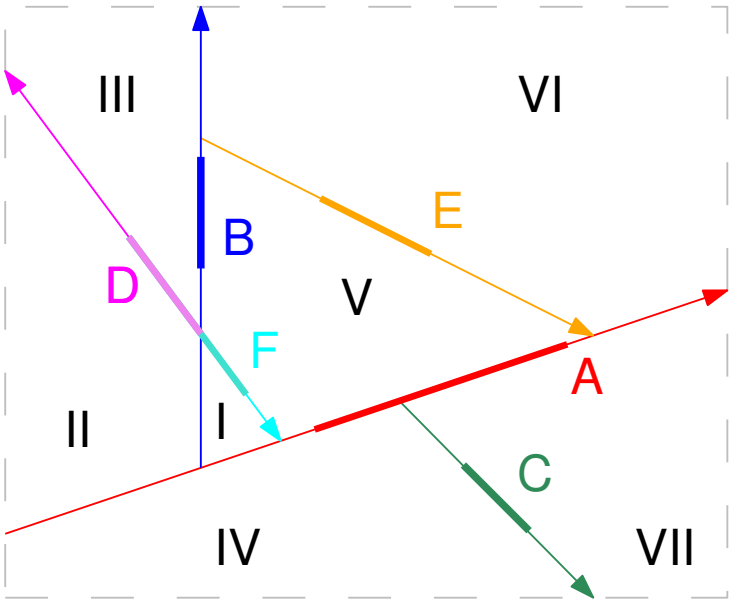
Constructing BSP Trees

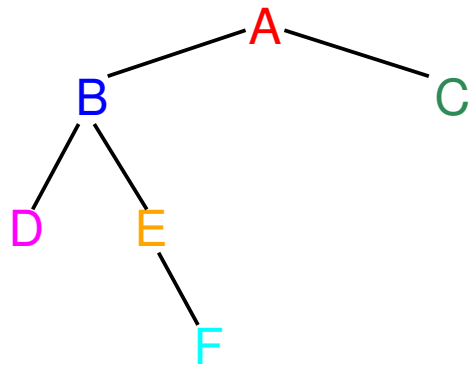
- This process continues recursively until every cell contains at most one triangle, or until the depth of the tree exceeds a threshold.
- Care has to be taken to keep the vertices of new triangles in consistent order.
- Ideally, a BSP tree should be small in size and balanced!
- Note that naive splitting of n input triangles (that do not intersect) may cause $O(n^3)$ output cells in the worst case.
- Paterson&Yao (1990): $O(n^2)$ size can be achieved.
- Practical experience: In 2D and in 3D, the space complexity tends to be in $o(n \log n)$, based on a randomized approach.

- HSR removal based on BSP trees is far too slow when compared to GPU-based approaches.
- But BSP trees are a very versatile structure with their merits!

Sample BSP Tree in 2D

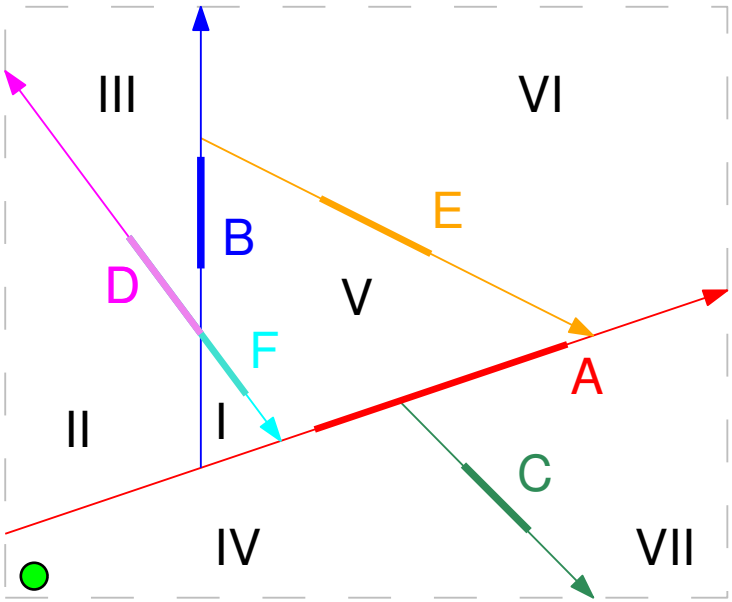
- Left cells/children are “left of” the splitting line segment; right cells/children are “right of” the splitting line segment.
- All resulting cells I, II, . . ., VII are convex.
- BSP tree: Each node stores all line segments that are collinear with the splitting line segment.

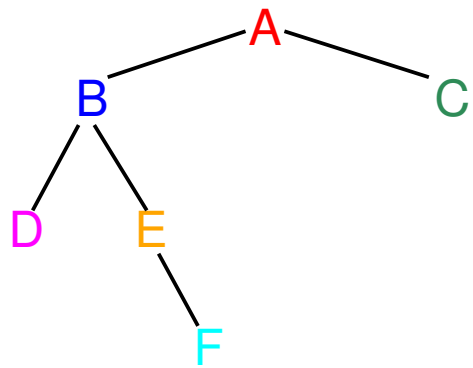




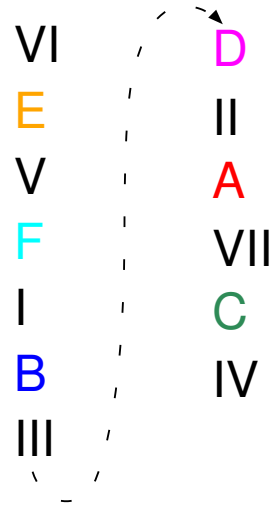
Sample BSP Tree in 2D: Tree Traversal

- Locate the viewpoint in the BSP subdivision.
- At each splitting plane, first draw the stuff on the farther side, then the polygon that defines the splitting plane, and finally the stuff on the nearer side.
- Moving the viewpoint does not render the BSP tree invalid, but merely changes the traversal order.





BSP Tree Traversal



Depth-Buffer Algorithm

- First described by Strasser and, independently, by Catmull in 1974.
- Idea: Determine visibility independently for each pixel.
- The depth-buffer algorithm, aka z-buffer algorithm, makes use of two buffers:
  - Frame/color buffer: $F[i,j]$ contains the color of pixel (i,j).
  - Z buffer: $Z[i,j]$ contains the z-coordinate of the object visible at pixel (i,j).

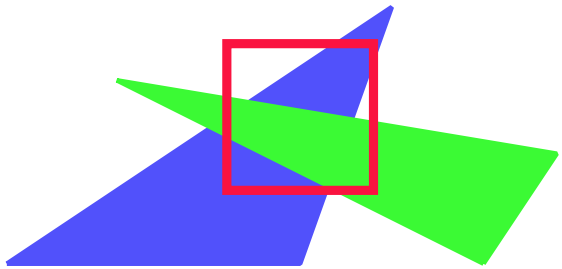
Algorithm *Depth-Buffer*

1. $\forall i,j: \quad Z[i,j] \leftarrow +\infty.$ ($*$ initialization of z buffer $*$)
2. $\forall i,j: \quad F[i,j] \leftarrow$ background color. ($*$ initialization of frame buffer $*$)
3. **for** each polygon π **do**
4. **for** each pixel (i,j) in projection of π **do**
5. $z \leftarrow z$-coordinate of point P of π that projects onto pixel (i,j).
6. **if** $z \leq Z[i,j]$ **then**
7. $Z[i,j] \leftarrow z$.
8. compute color at P and assign it to $F[i,j]$.

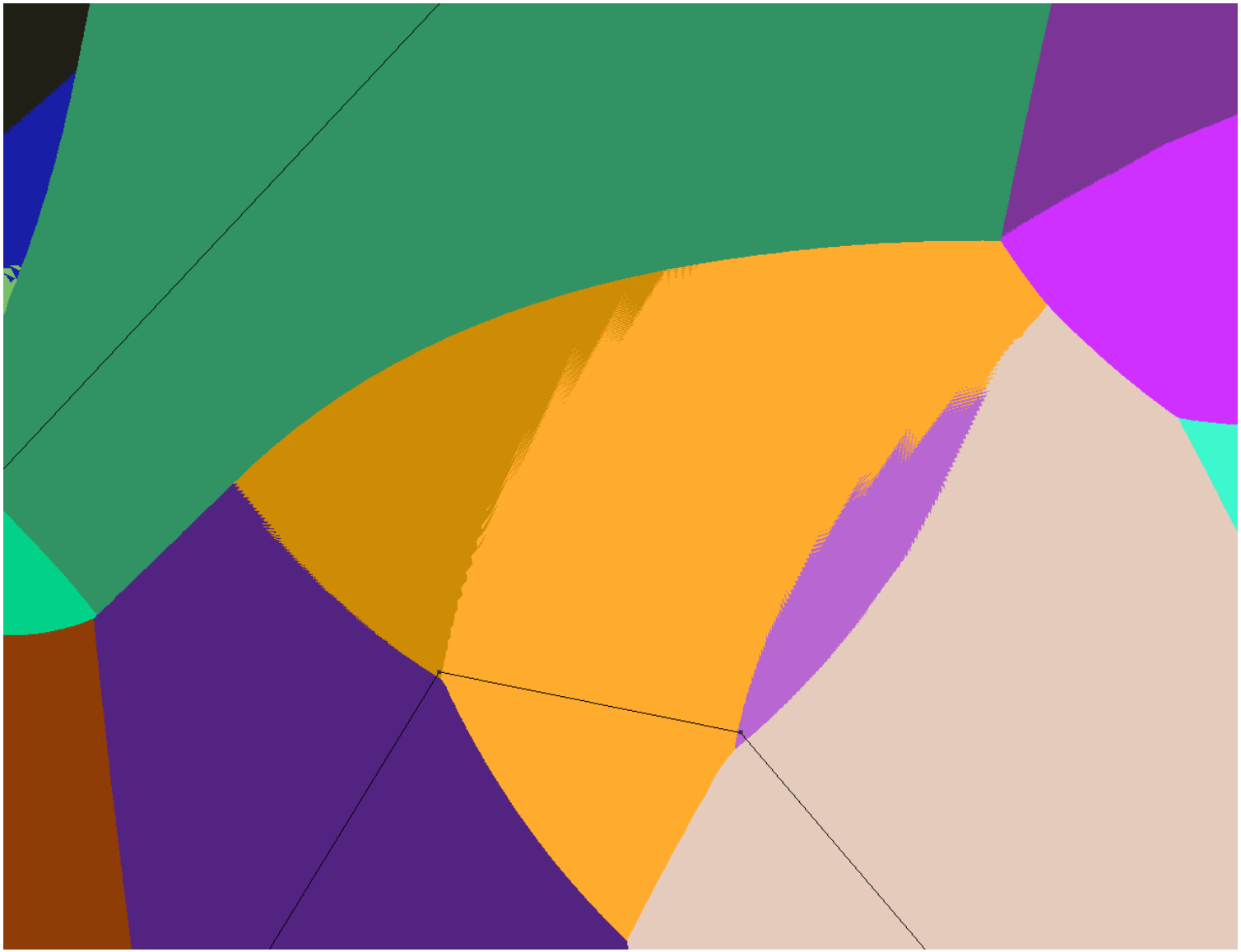
- Interpolation can be used to speed-up the computation of the z-values.

Depth-Buffer Algorithm: Pros and Cons

- Pros:
  - Simple, easy to implement, and ideally suited for hardware implementation.
  - Primitives can be processed in arbitrary order.
  - Interpenetration of primitives poses no problem.
- Cons:
  - The basic z-buffer algorithm does not handle translucency.
  - Most GPUs offer only 32-bit floating-point precision for z-buffer computation; 64-bit precision is not yet mandatory!
  - A perspective-to-orthogonal transformation tends to reduce z precision.
  - Aliasing problems if different polygons share the same pixel.
  - Co-planar primitives are handled unpredictably ("z-buffer fighting").



Depth-Buffer Algorithm: z-Buffer Fighting

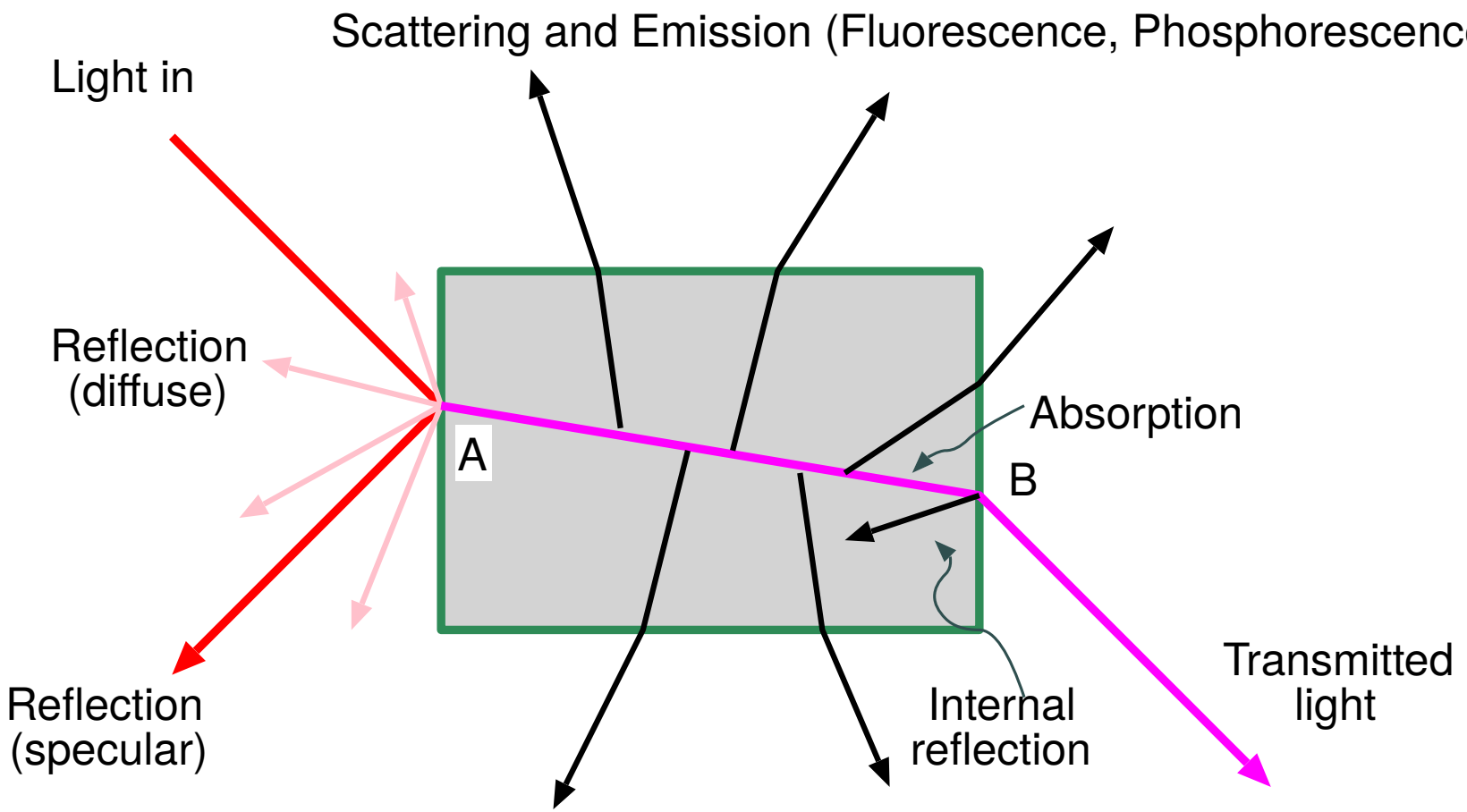


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

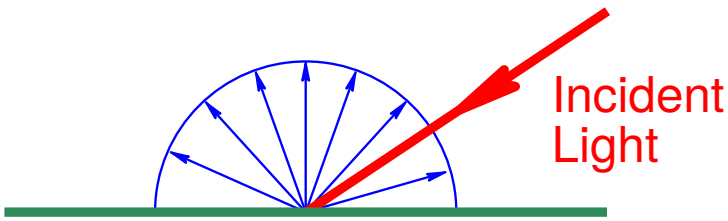
Depth-Buffer Algorithm: Handling Translucent Objects

- Translucent objects require a modification of the standard z-buffer:
  1. Draw all the opaque objects first, using the standard z-buffer.
  2. Draw translucent objects with blending:
     - Translucent objects behind an opaque object do not have any effect.
     - Translucent objects in front of all opaque objects do not change the z-value.
     - Blend colors appropriately.
- E.g., “alpha blending” as utilized by OpenGL:
  - $(R, G, B) \rightsquigarrow (R, G, B, A)$, where smaller values of A denote higher translucency.
  - $A := 0$ means transparent and $A := 1$ means opaque.
- Modern GPUs use several other specialized buffers to simulate other special effects, e.g., an accumulation buffer for simulating multiple exposures, motion blur or depth of field.

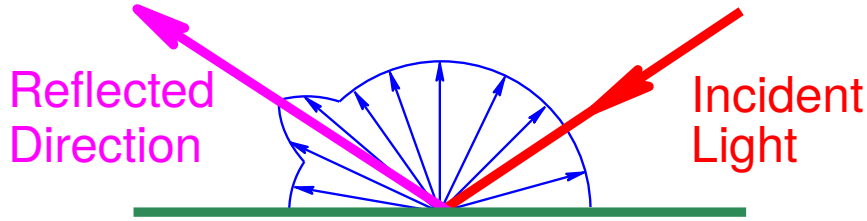
Light Interacting with an Object



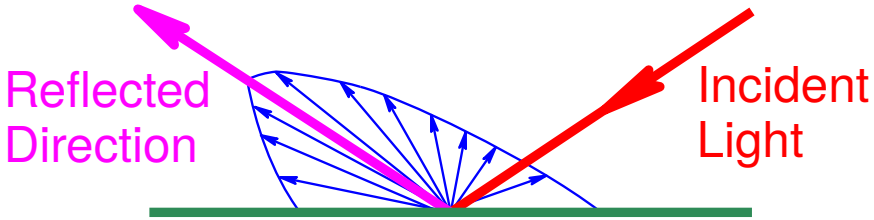
Surface Characteristics and Types of Reflection



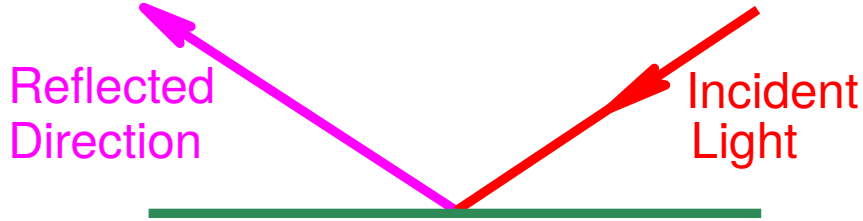
Perfectly matt surface: diffuse reflection



Slightly specular (shiny) surface



Highly specular (shiny) surface

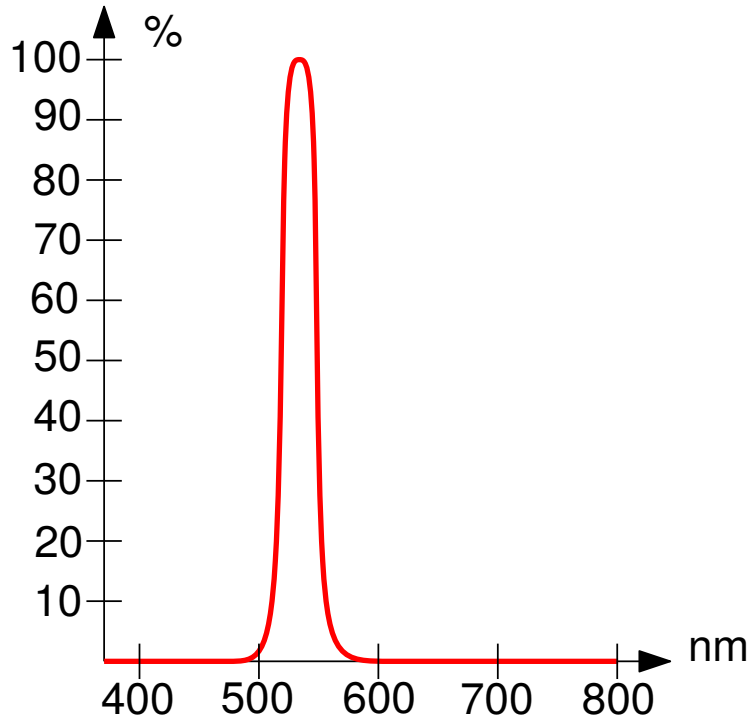


Perfect mirror: specular reflection

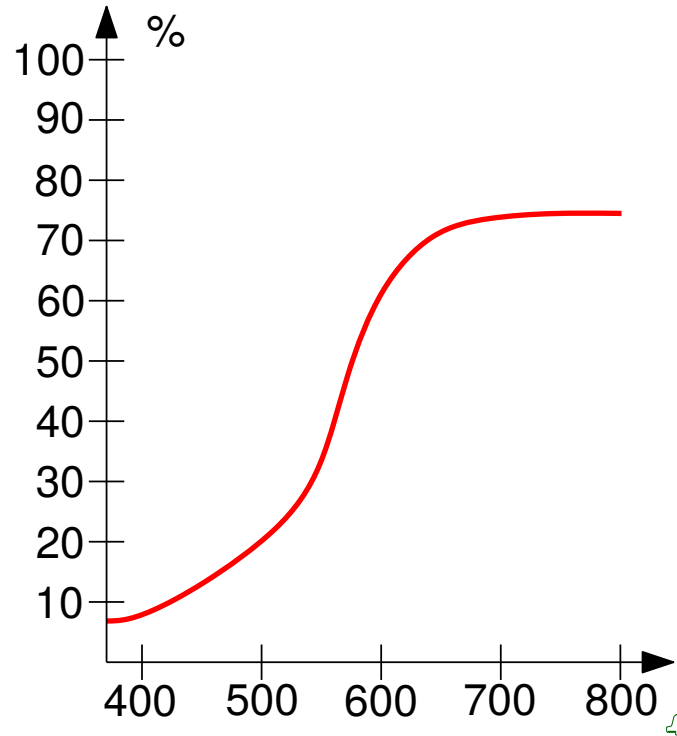
- Perfectly diffuse or perfectly specular surfaces hardly occur in the real world.

Modeling Reflection

- A *reflectance spectrum* indicates
  - for a particular angle of incidence
  - the percentage of the incoming light
  - reflected at each wavelength by the surface.
- To find the color of the light leaving the surface, we multiply the amount of incoming light by the percentage reflectance of the surface at each wavelength.
- Generalization: Bidirectional reflectance distribution function [Nicodemus 1965].



reflectance of a “greenish” surface



reflectance of copper at normal incidence

Basics of Illumination Models

- An illumination model defines the nature of the light emanating from a point, e.g., the geometry of its intensity distribution, etc.
- An illumination model can be expressed by an illumination equation in variables associated with the point on the object being shaded.
- An illumination equation can be interpreted as an equation for intensities – e.g., in a grey-scale image – as well as an equation for colors, for example RGB.
- This makes the illumination equation a vector equation, which must be evaluated for the red, green, and blue component separately.
- Obvious trade-off between the accuracy and complexity of a physics-based model and the convenience and speed of a purely heuristic model.

Phong's Illumination Model

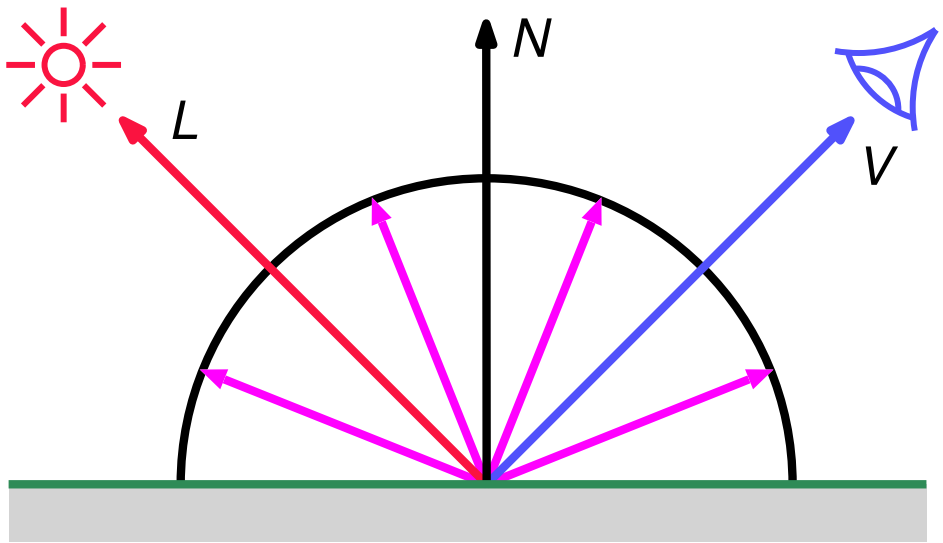
- The standard illumination model in computer graphics that compromises between acceptable results and processing cost is the Phong model [Phong 1975].
- This model handles reflected light in terms of a diffuse and specular component together with an ambient term:

$$I = I_a + I_d + I_s$$

I	...	intensity at a point,
I_a	...	ambient part of I,
I_d	...	diffuse part of I,
I_s	...	specular part of I.

Diffuse Reflection

- Diffuse reflection: Light is scattered uniformly in all directions from a point on the surface of the object.
- The amount of reflected light seen by the viewer does not depend on the viewer's position. Such surfaces are dull.

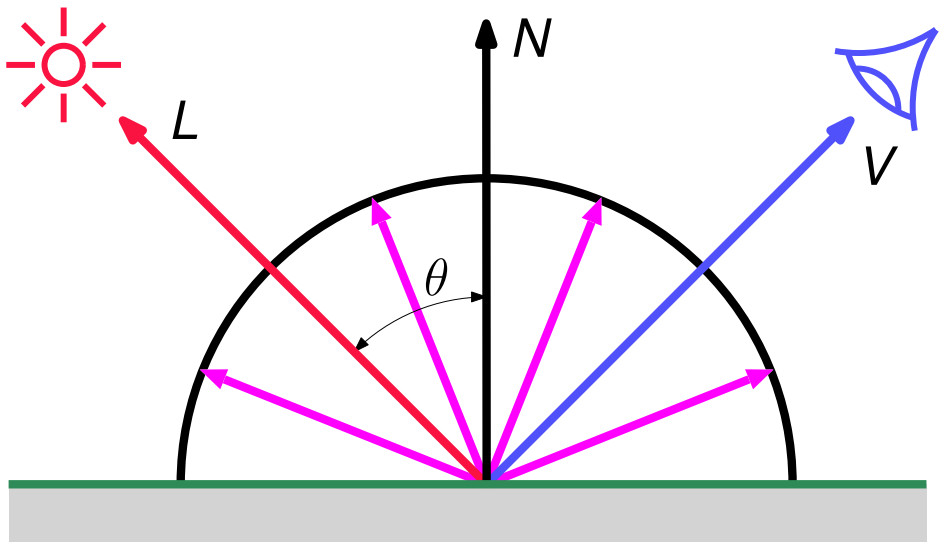


Diffuse Reflection

- The intensity of diffusely reflected light is given by Lambert's Cosine Law:

$$I_d = I_l \cdot k_d \cdot \cos\theta \qquad (0 \leq \theta \leq \frac{\pi}{2})$$

  where
  - I_l ... intensity of the light source,
  - θ ... angle between surface normal N and vector L (pointing to the light),
  - k_d ... diffuse-reflection coefficient, ranging between 0 and 1.
- The value k_d depends on the material *and* the wavelength of the incident light.



Diffuse Reflection

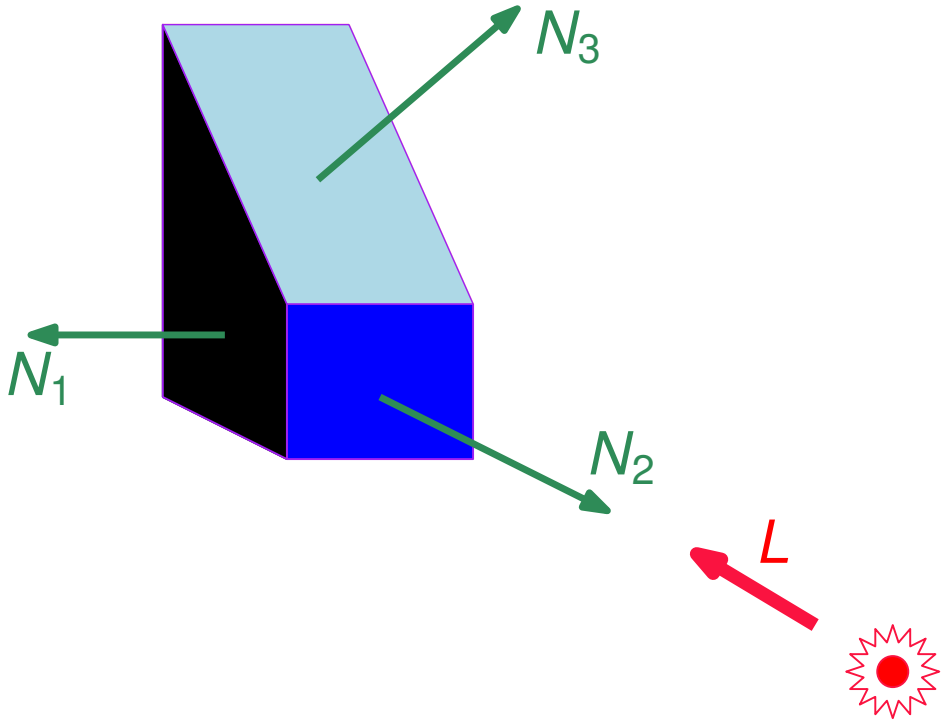
- For simplicity, all vectors are normalized!
- Since $\cos\theta = \langle L, N \rangle$, the illumination equation $I_d = I_l \cdot k_d \cdot \cos\theta$ can be rewritten using the dot product:

$$I_d = I_l \cdot k_d \cdot \langle L, N \rangle$$

- If a point light is sufficiently distant from the objects:
  - It makes essentially the same angle with all surfaces sharing the same surface normal.
  - In this case, L is a constant for the light source.
- If L does not vary then two parallel surfaces with identical surface normal will be shaded the same, no matter how different their distances from the light source or viewer are.
- This effect can be mitigated by using a light-source attenuation factor. (Problematic in practice, though.)
- Perfect Lambertian diffusers do not exist in nature.

Ambient Light

- Using the diffuse illumination model, any surface visible by the viewer but not directly lit by the light (since $\langle N, L \rangle = 0$) is ……
……black!



- This does not match reality!

Ambient Light

- Ambient light is the result of multiple reflections from walls and objects, and it is incident on a surface from all directions.
- It is modeled as a constant term I_{al} for a particular object using a constant ambient-reflection coefficient k_a ranging between 0 and 1:

$$I_a = I_{al} \cdot k_a$$

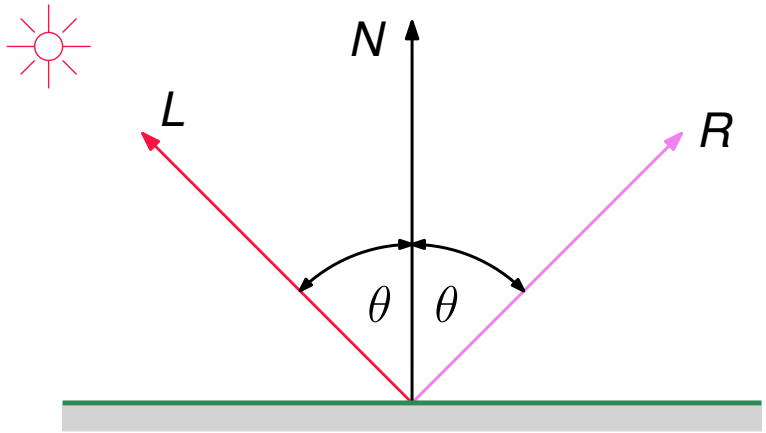
- The ambient-reflection coefficient is a material property.

Caveat

The ambient-light term is an empirical convenience and does not correspond directly to any physical property of real materials.

Specular Reflection

- The *law of reflection* tells us that
  - the reflected ray remains within the plane of incidence, and the angle of reflection equals the angle of incidence. The plane of incidence includes the incident ray and the normal to the point of incidence.
  - the reflected ray leaves a glossy surface at angle θ, where θ is the angle of incidence.

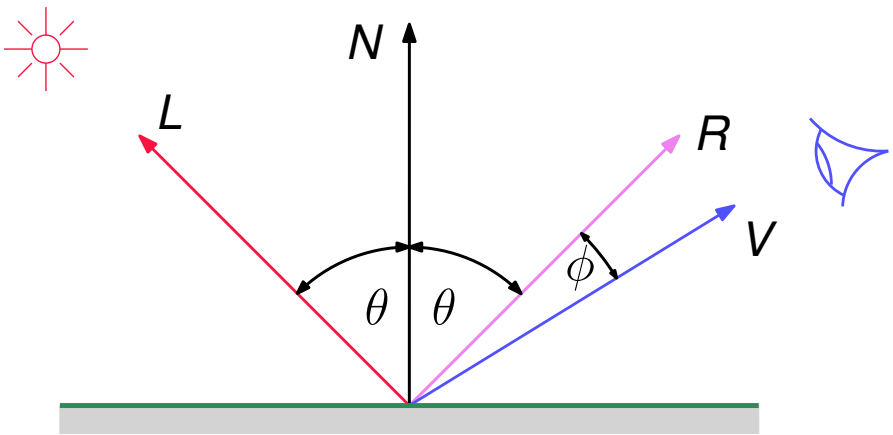


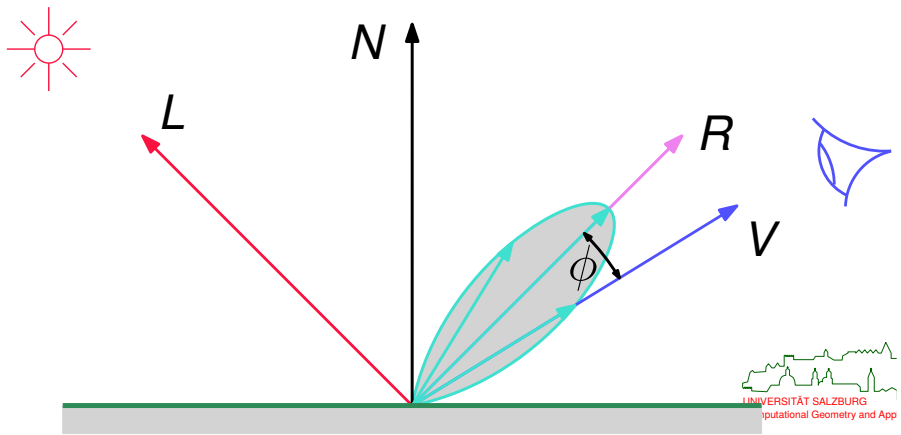
Specular Reflection

- However, most surfaces – and their reflectance properties – are somewhere in between a perfect diffuser and perfect glossiness, i.e., a perfect mirror.
- In practice, specular reflection is not perfect and reflected light can be seen for viewing directions close to the direction of the reflected beam.
- Thus, the degree of specular reflection seen by a viewer depends on the view point.
- The area over which specular reflection is seen for a given view point is commonly referred to as *highlight*.
- The color of the specularly reflected light is different from the color of the diffusely reflected light.
- In simple models the specular component is assumed to be the color of the light source.

Specular Reflection

- For a perfect mirror, a highlight is only visible if ϕ – the angle between the viewing direction V and the reflection vector R – is zero.
- In practice, however, specular reflection is seen over a range of ϕ that depends on the glossiness of the surface.



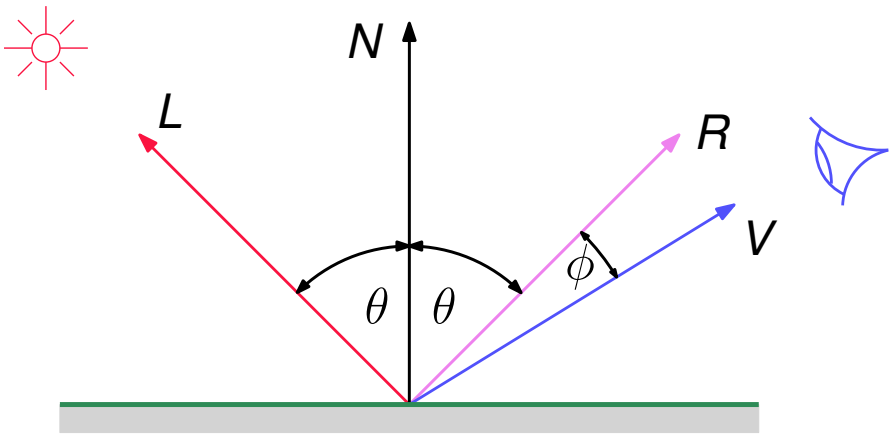


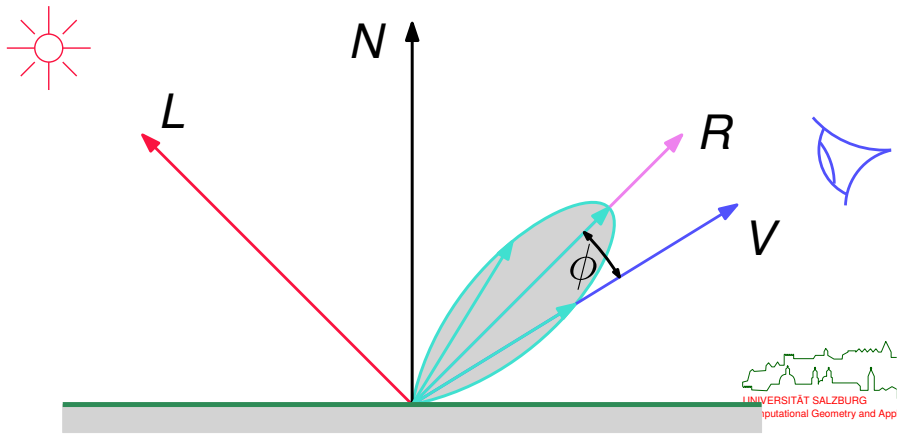
Coefficient of Glossiness

- Phong modeled this behavior empirically by a term $cos^n \phi$:

$$I_s = I_l \cdot k_s \cdot cos^n \phi \qquad (0 \leq k_s \leq 1; \quad 0 \leq n \leq \infty)$$

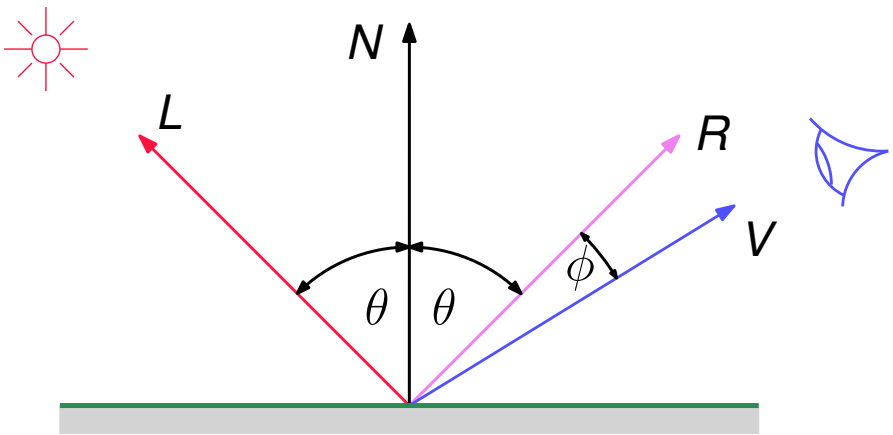
  where k_s is the specular reflection coefficient, usually taken to be a material-dependent constant.
- Note that large values of n are required for a tight highlight to be obtained: For metals values between 100 and 200 are common, while values between 5 and 10 will result in a plastic appearance.
- For a perfect mirror surface, this coefficient of glossiness is infinite.

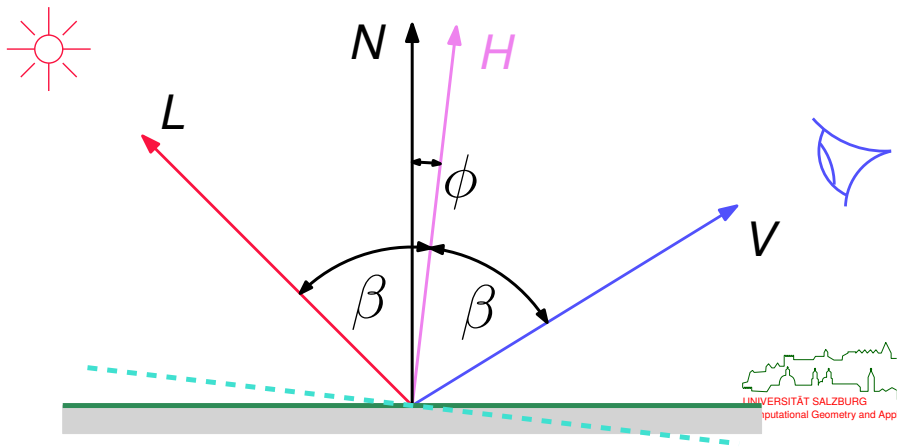




Specular Reflection

- The expense of the specular illumination equation can be reduced considerably by making some geometric approximations.
- Since the vector R is expensive to compute, in 1977 Blinn suggested to use the vector H instead: Blinn-Phong reflection model.
- H is the mean vector of L and V. The specular term then becomes a function of $\langle N, H \rangle$ rather than $\langle R, V \rangle$.



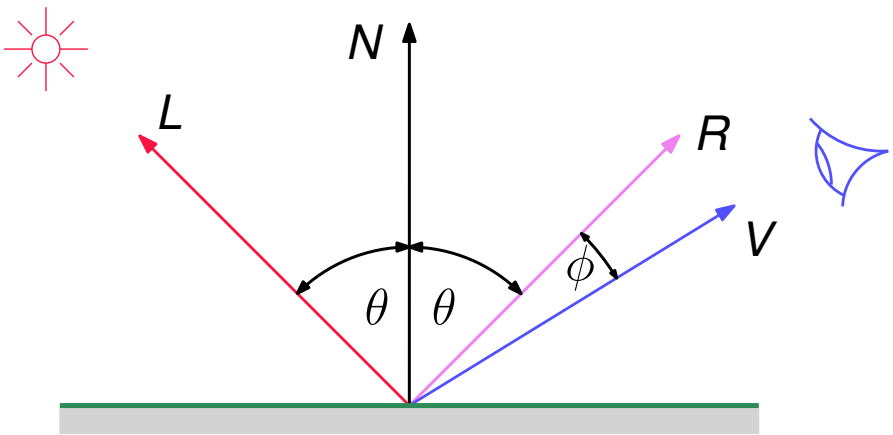


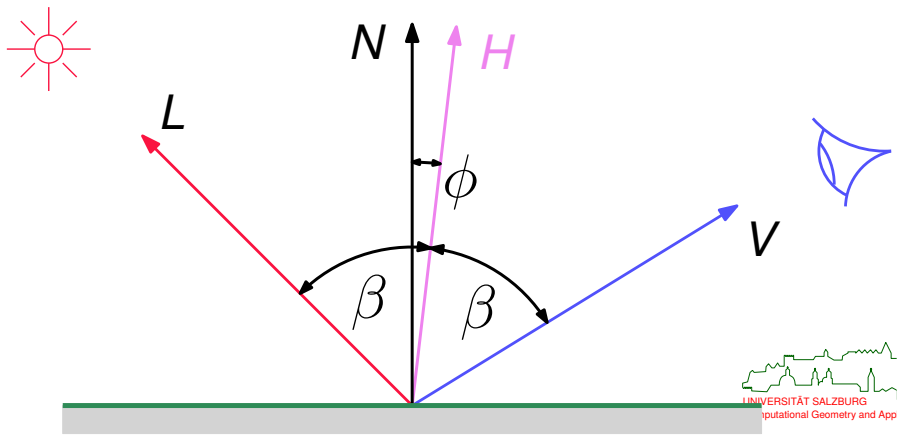
Specular Reflection

- Thus,

$$I_s = I_l \cdot k_s \cdot (\langle N, H \rangle)^n.$$

- As the angle between R and V is twice the angle between N and H, the use of N and H spreads the highlight over a greater area.
- Like the diffuse term this simple model of specular reflection is a local model.
- Light reflected onto the surface that originates from specular reflections in other objects is not considered!





Specular Reflection

- Summarizing, according to Phong, for colored objects the easiest way to model the specular highlights is to use the color of the light source, and to control the color of the objects by setting the diffuse reflection coefficients appropriately:

$$I_r = I_a \cdot k_{ar} + I_i[k_{dr}\langle L, N\rangle + k_s(\langle N, H\rangle)^n]$$
$$I_g = I_a \cdot k_{ag} + I_i[k_{dg}\langle L, N\rangle + k_s(\langle N, H\rangle)^n]$$
$$I_b = I_a \cdot k_{ab} + I_i[k_{db}\langle L, N\rangle + k_s(\langle N, H\rangle)^n]$$

- Combining these three equations in a single vector expression yields:

$$I_{r,g,b} = I_a \cdot k_a(r,g,b) + I_i[k_d(r,g,b)\langle L, N\rangle + k_s(\langle N, H\rangle)^n]$$

Discussion of Phong's Illumination Model

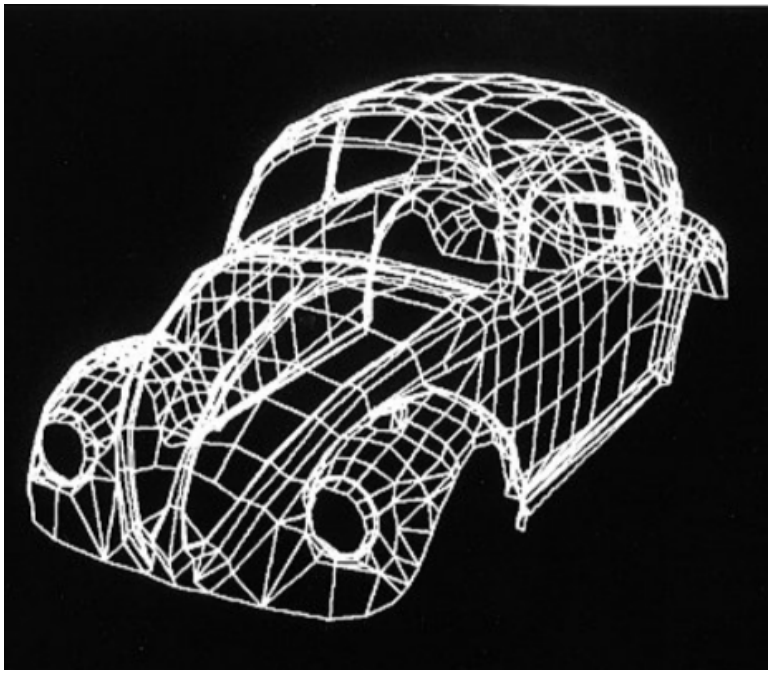
- Light sources are assumed to be point sources. Any intensity distribution of the light source is ignored.
- All geometry except the surface normal is ignored and no distance information is considered.
- The diffuse and specular terms are modeled as local components.
  - No reflections of other objects in the surface of the object being rendered are considered.
  - Phong's model lacks shadows!
  - Lack of shadows means not only that objects do not cast a shadow on other objects, but self-shadowing within an object is omitted.
  - Concavities in an object that are hidden from the light source are erroneously shaded simply on the basis of their surface normal.
- An empirical model is used to simulate the decrease of the specular term around the reflection vector modeling the glossiness of the surface.
- The color of the specular reflection is assumed to be that of the light source. That is, for white light highlights are rendered white regardless of the material.
- Phong's illumination model (or some variant thereof) is in wide-spread use due its apparent main advantage: Its simplicity is unruled, and it produces acceptable results for many applications.

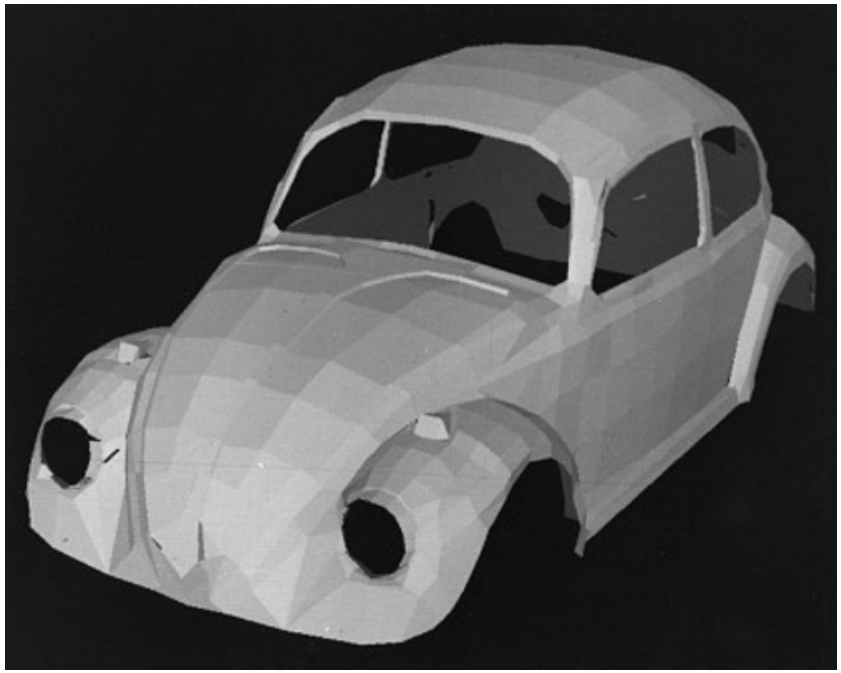
Flat Shading

- The simplest shading model for a polygon is flat shading, also known as faceted shading or constant shading.
- This approach evaluates an illumination model once for a polygon to determine a single intensity value (or color value) that is then used to shade an entire polygon.
- Basically, the illumination equation is sampled once for each polygon, and this value is used across the polygon to reconstruct the polygon's shade.
- This approach is only valid if the following assumptions are true:
  - The light source is at infinity, so $\langle N, L \rangle$ is constant across the polygon's face.
  - The viewer is at infinity, so $\langle N, V \rangle$ is constant across the polygon's face.
  - The polygon represents the actual surface being modeled, and is not an approximation to a curved surface.
- Otherwise, constant shading produces a "faceted" appearance.

Flat Shading: Utah VW Bug

- In 1972, Sutherland's students manually digitized, modeled and rendered his wife's car.

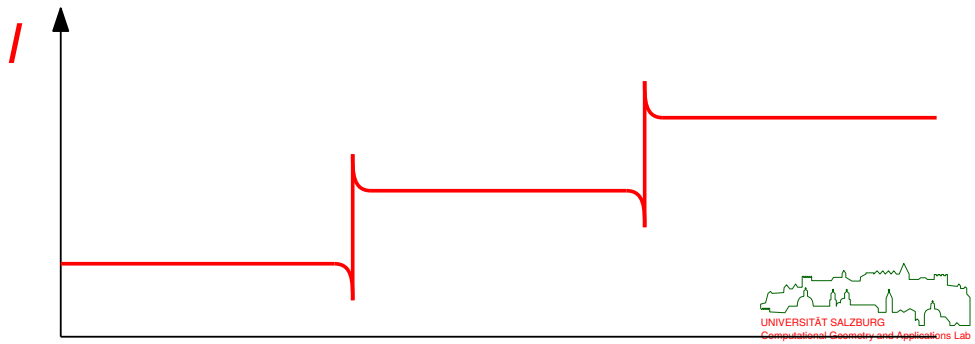




[Image credit: CS Dept. at Univ. of Utah]

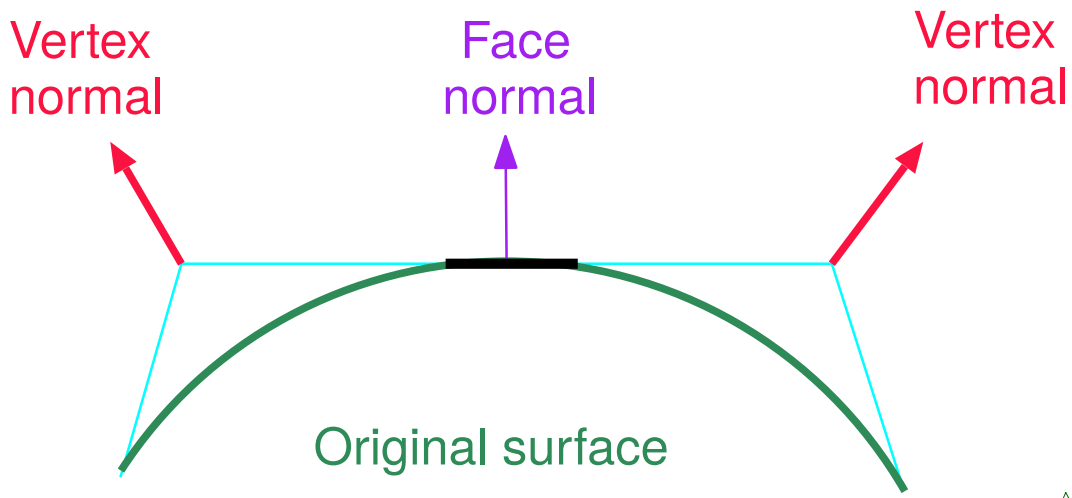
Flat Shading: Mach Band Effect

- The simple solution of using a finer surface mesh turns out to be surprisingly ineffective: The perceived difference in shading between adjacent facets is accentuated by the Mach band effect [Mach, $\approx$ 1860].
- Mach banding is caused by lateral inhibition of the receptors in our eyes: The more light a receptor receives, the more that receptor blocks the response of the receptors adjacent to it.
- That is, the human visual system performs some form of edge enhancement by exaggerating the intensity change at any edge where there is a discontinuity of intensity.
- As a result, at the border between two facets the dark facet appears even darker and the light facet appears even lighter.



Gouraud Shading

- Gouraud shading [Gouraud 1971], also called intensity interpolation shading, extends the concept of interpolated shading applied to individual polygons by interpolating polygon vertex illumination values that take into account the surface being approximated.
- The technique first calculates the intensity at each vertex of the polygon by applying an illumination model.
- These vertex intensities are afterwards interpolated over the polygon.
- The normal vector N used in these equations is the so-called vertex normal: It is calculated as the average of the normals of the polygons that share the vertex.
- This is an important feature of the method since the vertex normal is an approximation to the true normal of the surface (which the polygon represents) at that point.
- Gouraud shading reduces intensity discontinuities.



Gouraud Shading

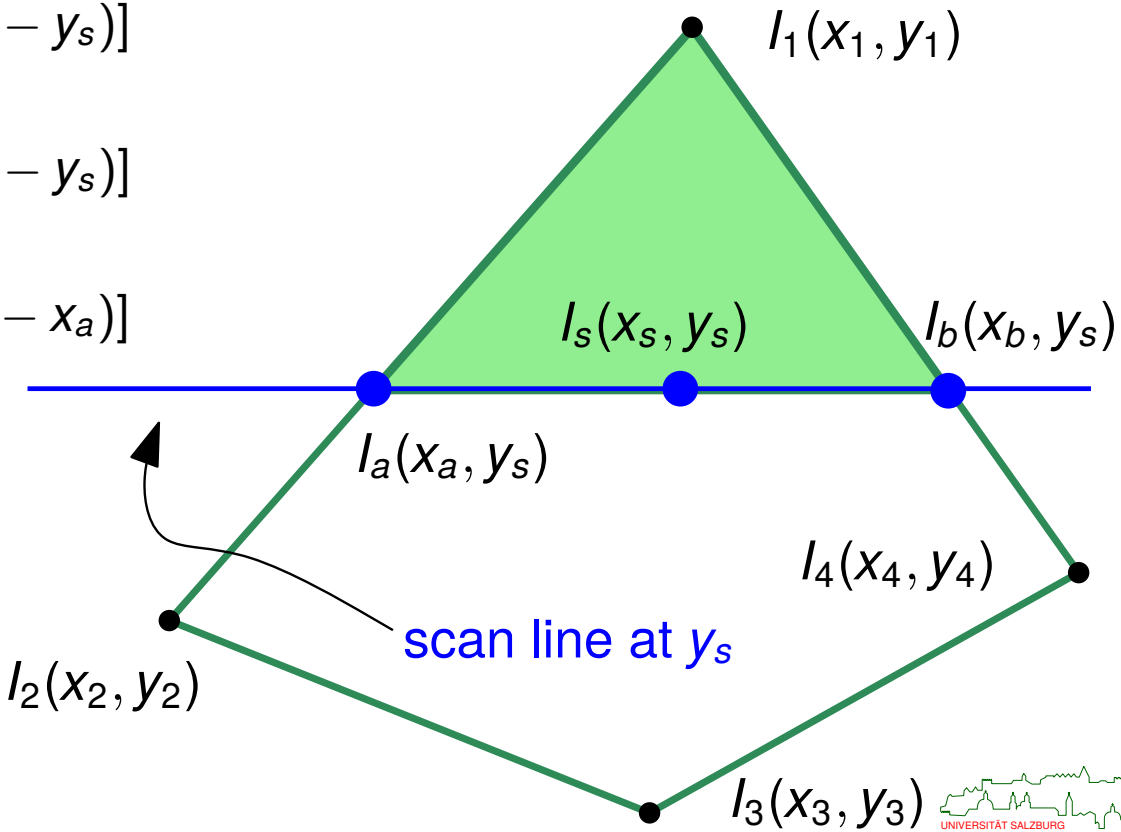
- The interpolation process that calculates the intensity over a polygonal surface can then be integrated with a scan-conversion algorithm that evaluates the screen position of the edges of a polygon from the vertex intensities, and the intensities along a scan line from these.
- This yields the following bilinear interpolation scheme:

$$I_a = \frac{1}{y_1 - y_2}[I_1(y_s - y_2) + I_2(y_1 - y_s)]$$

$$I_b = \frac{1}{y_1 - y_4}[I_1(y_s - y_4) + I_4(y_1 - y_s)]$$

$$I_s = \frac{1}{x_b - x_a}[I_a(x_b - x_s) + I_b(x_s - x_a)]$$

- These equations can be implemented as incremental calculations.
- Gouraud shading handles specular reflection correctly only if the highlight occurs in one of the vertices.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

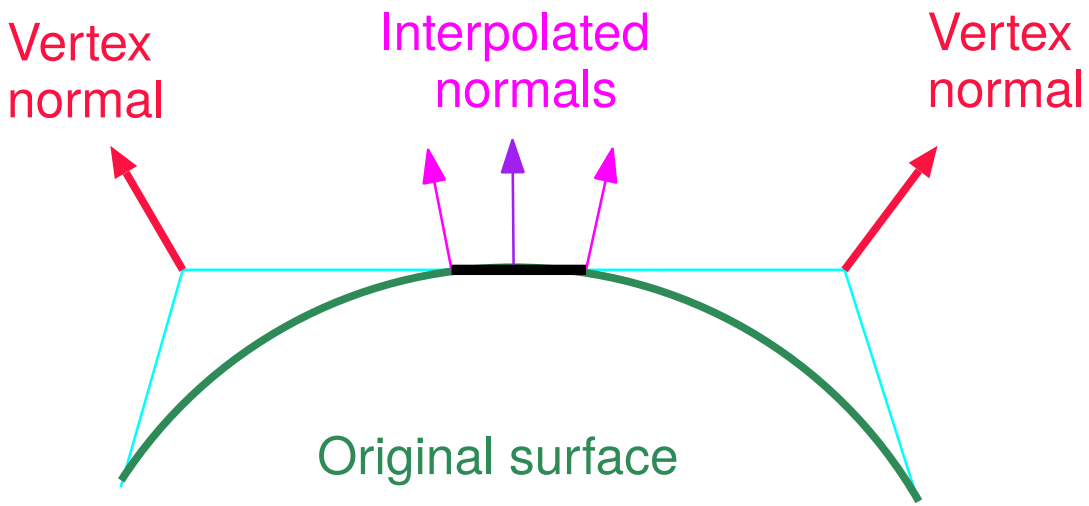
Gouraud Shading: Sophie Gouraud

- Tough luck if you are married to someone who needs a model for trying out his theory . . .

[Image credit: CS Dept. at Univ. of Utah]

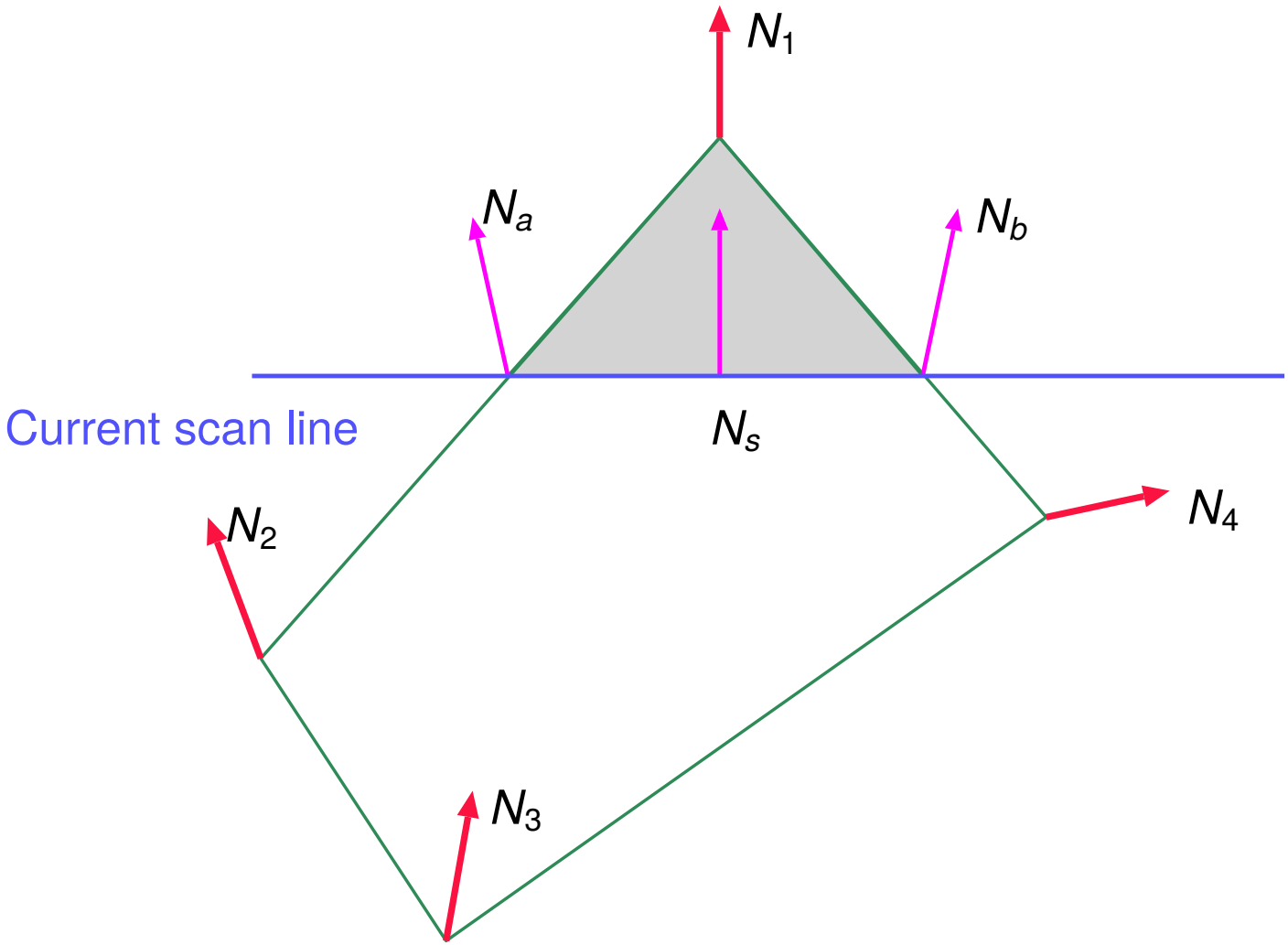
Phong Shading

- Phong shading [Phong 1975], which is also known as normal-vector interpolation shading, makes use of the vertex normal vectors for bilinear interpolation in the following steps:
  1. Interpolation of the normal vectors along the edges between the vertices.
  2. By sliding a horizontal scan line from, say, bottom to top the normal vectors of the surface inclosed by the edges are interpolated.
  3. So there exists a normal vector for each point of the polygon surface. This normal vector can then in turn be used for evaluating the illumination equation.
- The interpolation of the normal vectors tends to “restore” curvature.



Bilinear Interpolation of the Vertex Normals

- With Phong shading, normal vectors are interpolated rather than the vertex intensities: vector interpolation replaces intensity interpolation.

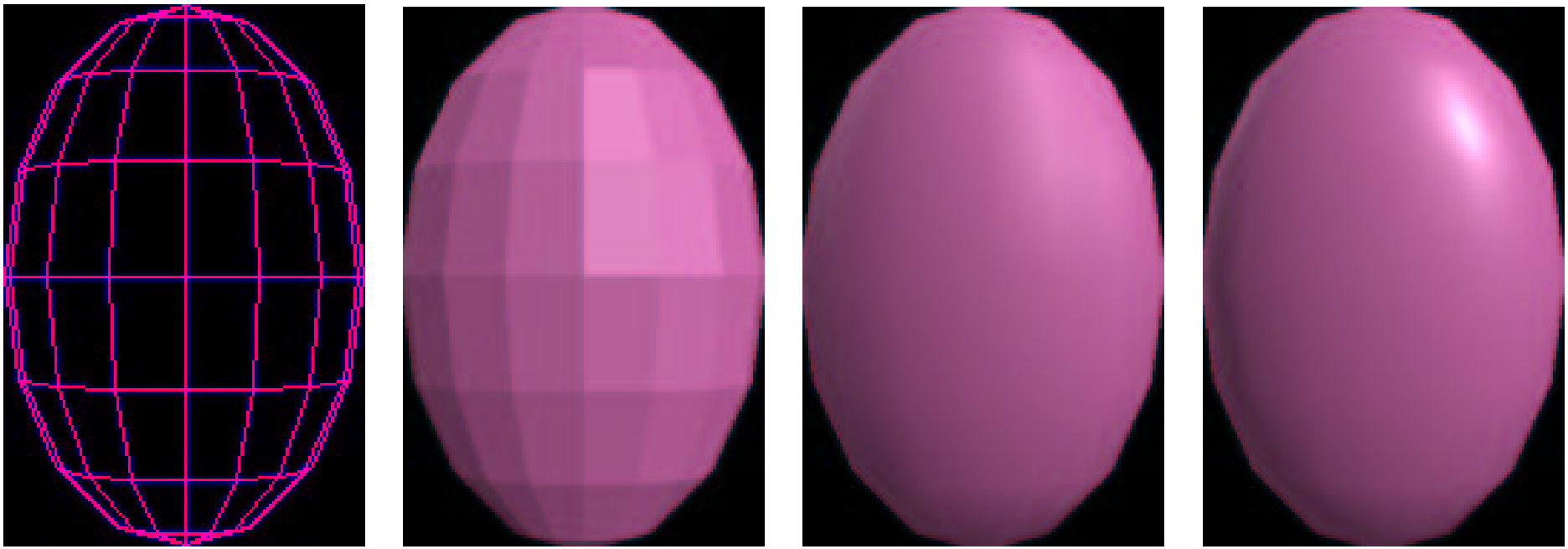


Characteristics of Phong Shading

- Results of normal vector interpolation are in general superior to intensity interpolation because an approximation to the normal is used at each point. (This is true even without taking into account specular reflection.)
- Specular reflection is handled by Phong shading.
- Phong shading tends to be much more expensive than Gouraud shading: The illumination equation has to be evaluated at every pixel. In particular, this requires unit normal vectors.
- To avoid the costs of normalizing the interpolated normals, approximation techniques (such as Taylor series expansion [Bishop&Weimer 1986]) may be used.

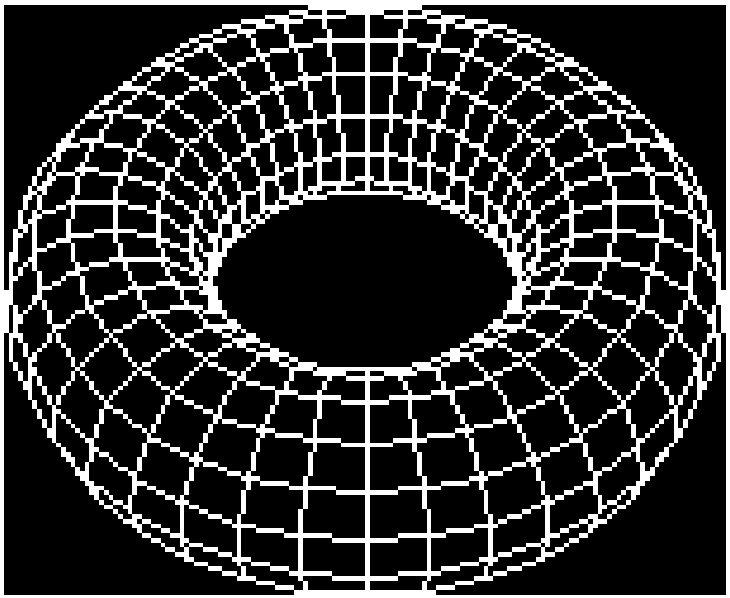
Sample Shading: Ellipsoid

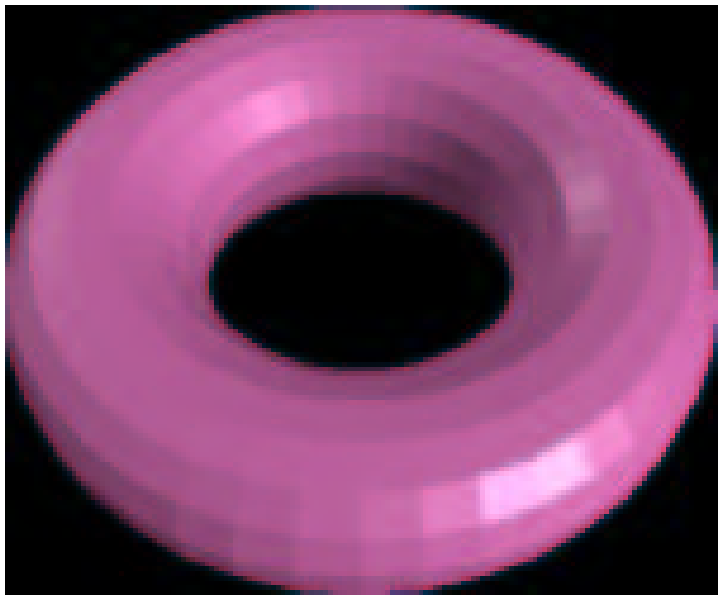
- The images show, from left to right, line drawing, flat shading, Gouraud shading, and Phong shading.

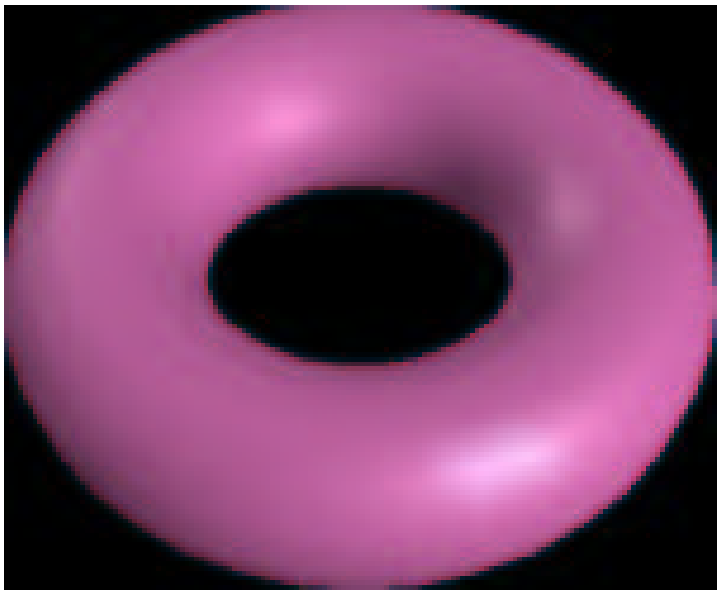


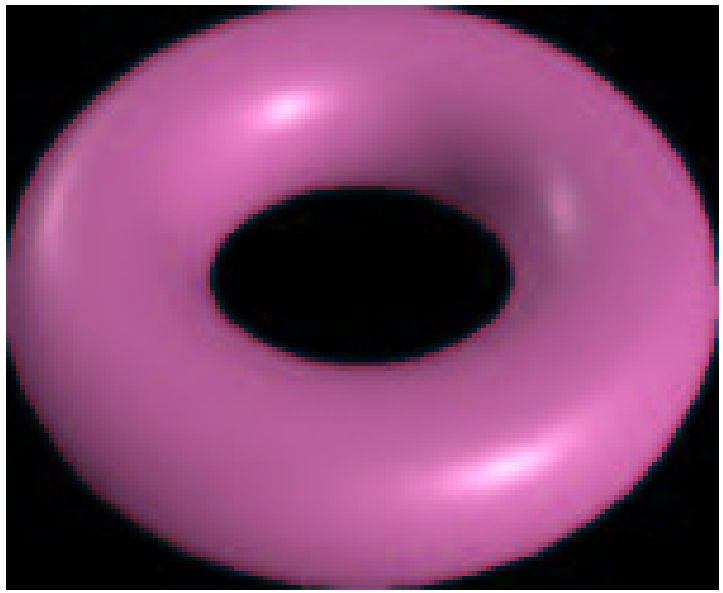
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sample Shading: Torus





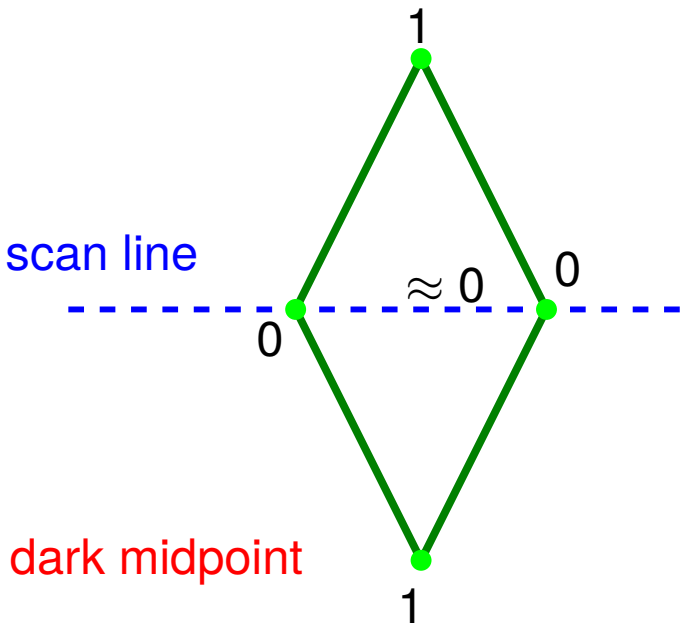


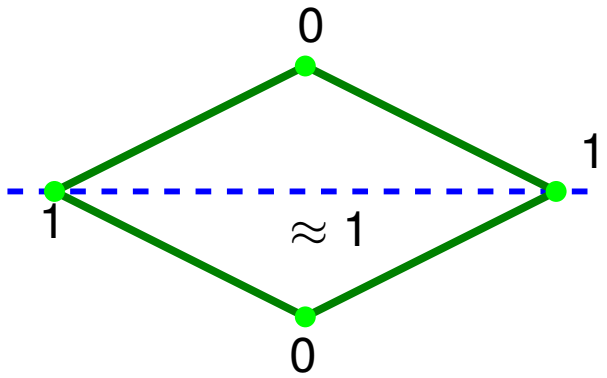


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Problems of Interpolated Shading

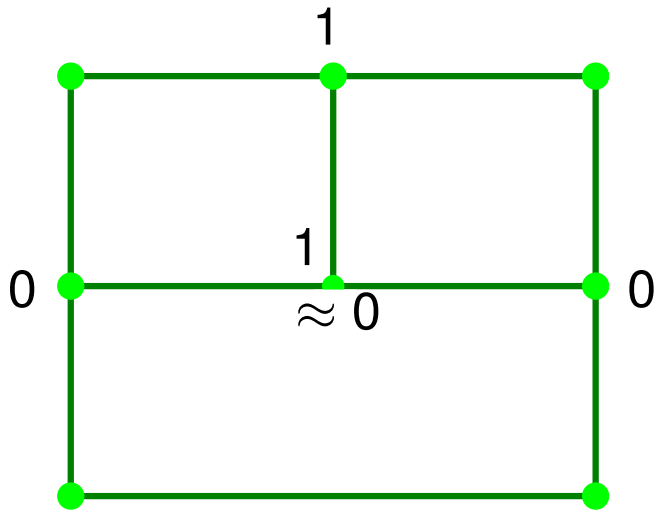
- *Silhouette edges*: No matter how good a polygonal approximation an interpolated shading model offers to the actual shading of a curved surface, the *silhouette edges* of the mesh are still clearly polygonal.
- *Orientation Dependencies:* The results of interpolated shading are not independent of the projected polygon's orientation.
- This problem can be mitigated by using a larger number of smaller polygons, or by decomposing the polygon into triangles and using barycentric coordinates for the interpolation.

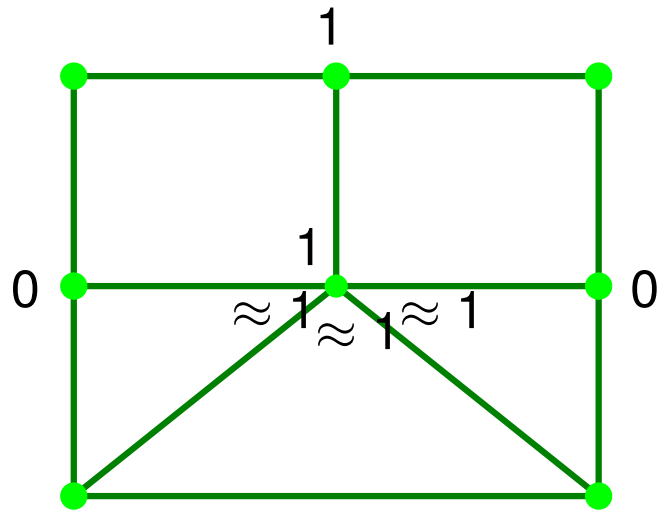




Problems with Interpolated Shading

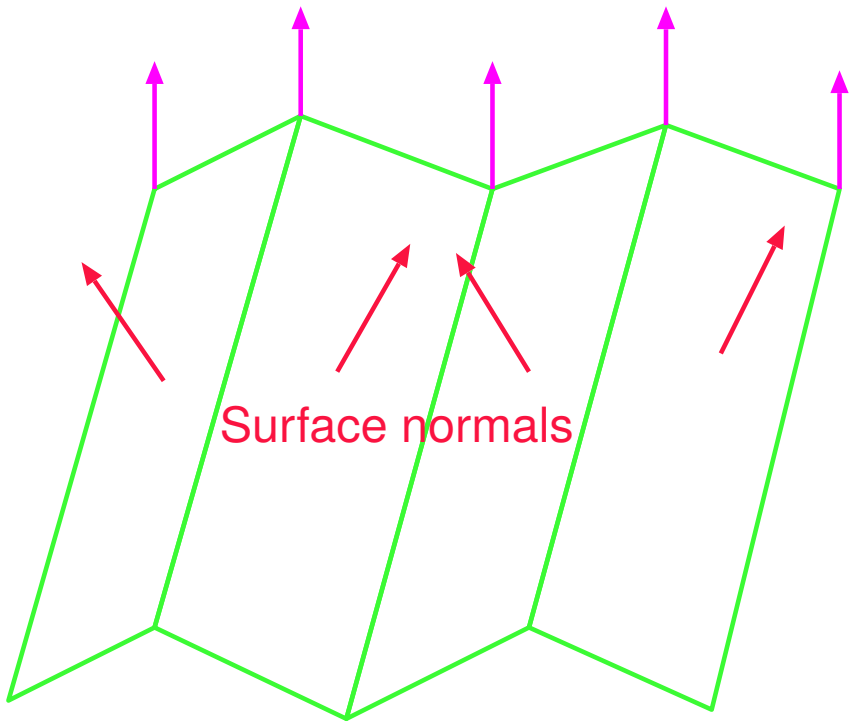
- *Perspective distortion:* Anomalies can appear in animated sequences because the intensity/normal-vector interpolation is carried out in screen coordinates from vertex normals as calculated in world coordinates. This is not invariant with respect to transformations, and may cause frame-to-frame disturbances in animations.
- *Problems with shared vertices*: Shading discontinuities can occur when two adjacent polygons fail to share a vertex that lies along their common edge.
- Thus, a vertex has to be shared by all adjacent areas. As further improvement, such a vertex is connected to other vertices of the adjacent polygon.





Problems with Interpolated Shading

- *Unrepresentative surface normals*: The process of averaging surface normals to provide vertex normals for the intensity calculation can cause errors which result in corrugations being smoothed out. This would result in a visually flat surface.



Aliasing and Anti-Aliasing

- *Aliasing* is the collective term for any form of visual artifact caused by mapping a continuum to a discrete set of samples.
- The aliasing problem thoroughly permeates computer graphics. Its most familiar manifestations are
  - spatial aliasing,
  - temporal aliasing.
- Sample spatial aliasing: In the figure, the first letter suffers from aliasing and looks coarse compared to the second letter.

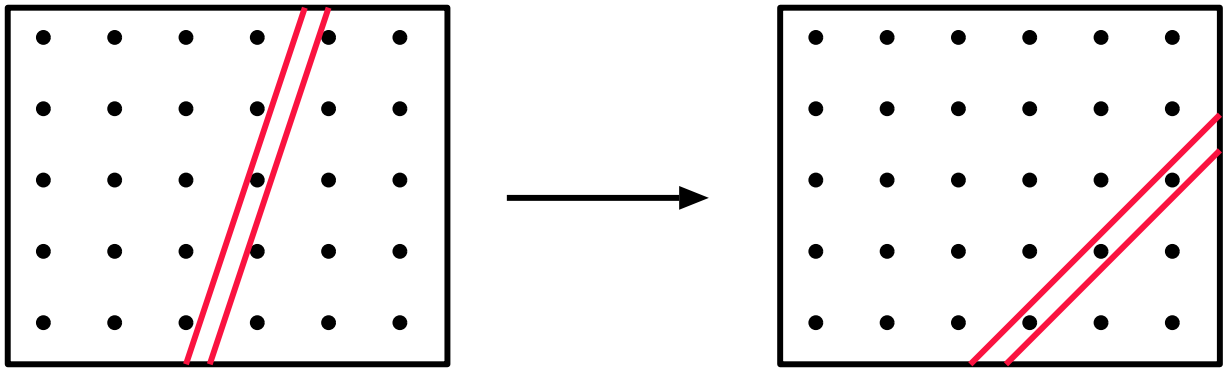
- *Anti-aliasing* is one of the most important classes of techniques for making graphics visually pleasing and text easy to read.
- It is a way of fooling the eye into seeing straight lines, smooth curves, and smooth motions where there are none.

PDF viewer and anti-aliasing

Note that a PDF viewer may also cause (anti-)aliasing artifacts!

Spatial Aliasing

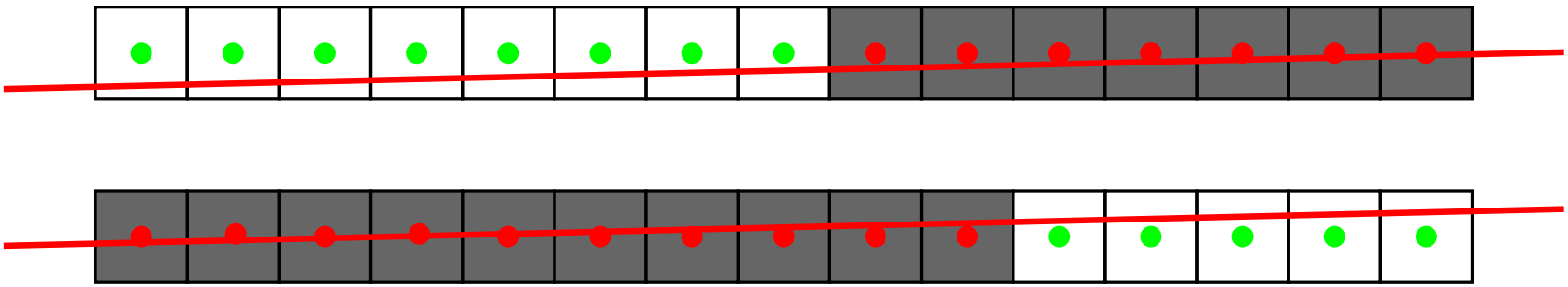
- A *silhouette edge* is the boundary of a polygon, or of any surface unit that exhibits a high contrast over its background. (In general, contrast means light and dark areas of the same color.)
- Spatial aliasing is due to the discrete nature of pixels on a monitor, and results in silhouette edges that do not look smooth: "*jaggies*" and "*stair-casing*".
- A long thin object may break up depending on its position with respect to the pixel array.



- Another aliasing artifact occurs when small objects, whose spatial extent is less than the area of a pixel, are rendered or not depending on whether they are intersected by a sample point.

Temporal Aliasing

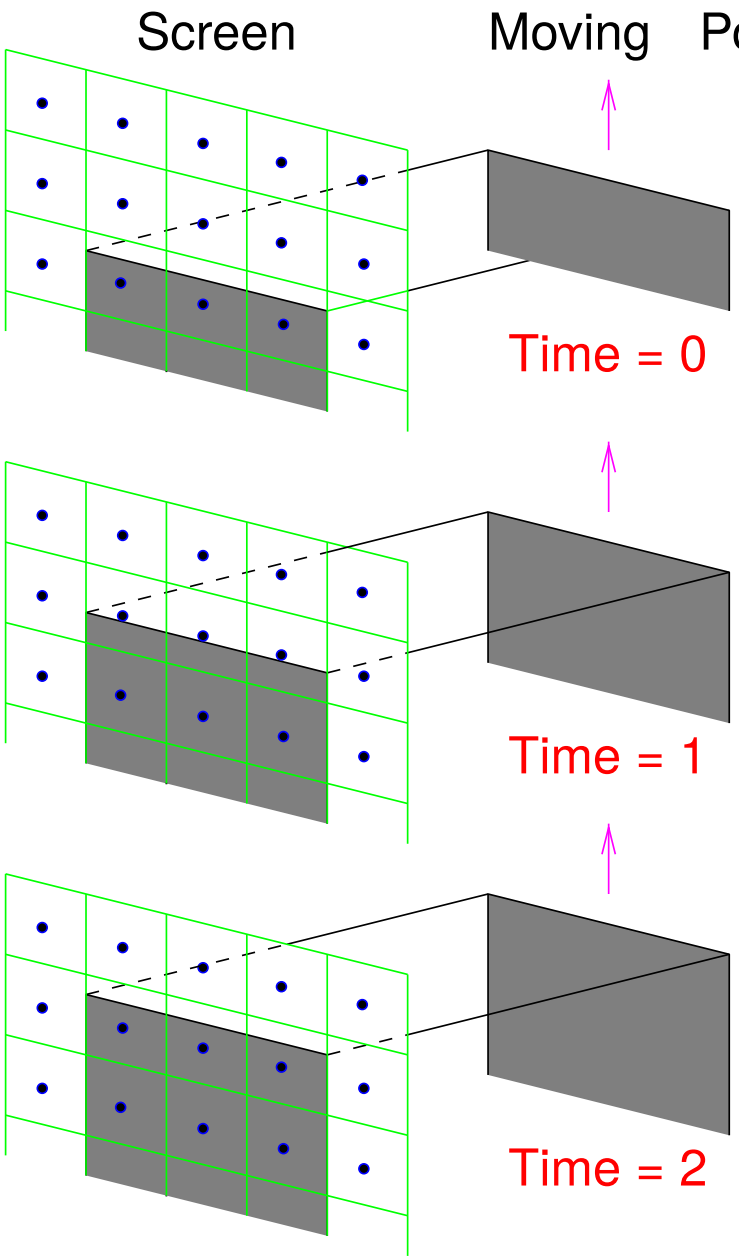
- New problems may occur when still images are shown in an animated sequence:
  - “Crawling” edges (i.e., moving jaggies);
  - “Scintillating” objects (i.e., objects (dis-)appearing during a move).
- A slight change in the position of a line in world coordinates can cause a huge “jump” in the position of the digitized line in screen coordinates, i.e., a “crawling” of the pixels that represent the line.



- Such changes can be very distracting and are intolerable in some applications (e.g., flight simulators).

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Temporal Aliasing: Scintillating and Jumping

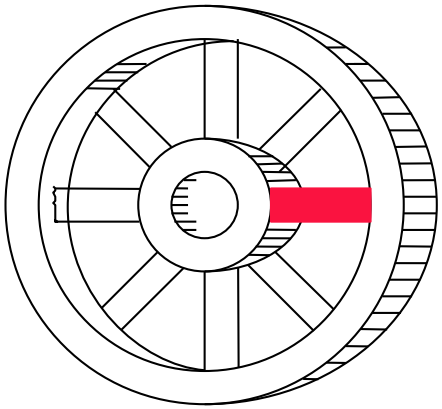


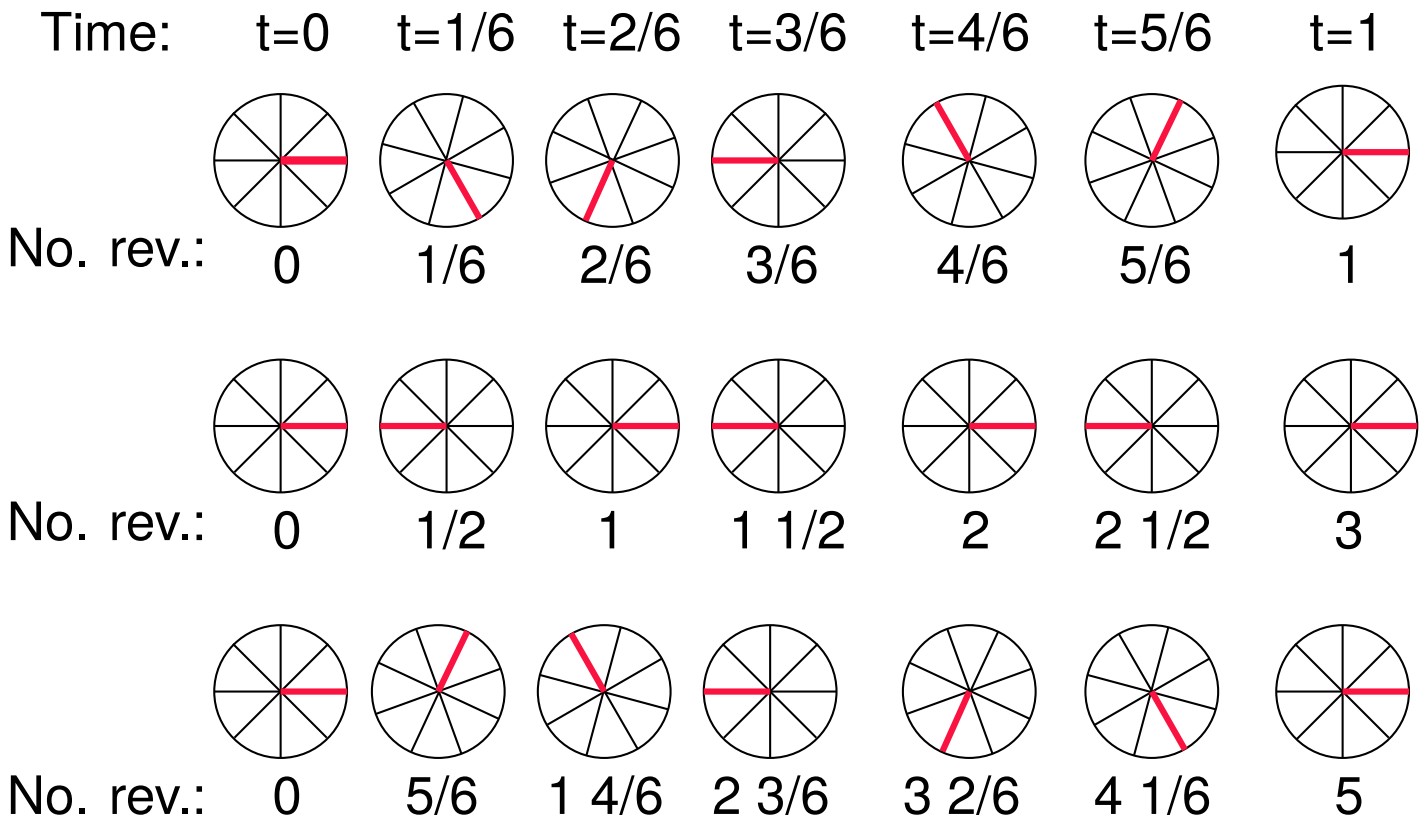
Flow of time (successive frames)

This row suddenly 'pops' on when the moving edge covers the pixel centers

Temporal Aliasing: Spinning Wheel

- In the third row, the wheel is spinning at 5 revolutions per second, but appears to be spinning backwards at 1 revolution per second. Thus, the fast speed is aliasing as a slower speed after sampling.

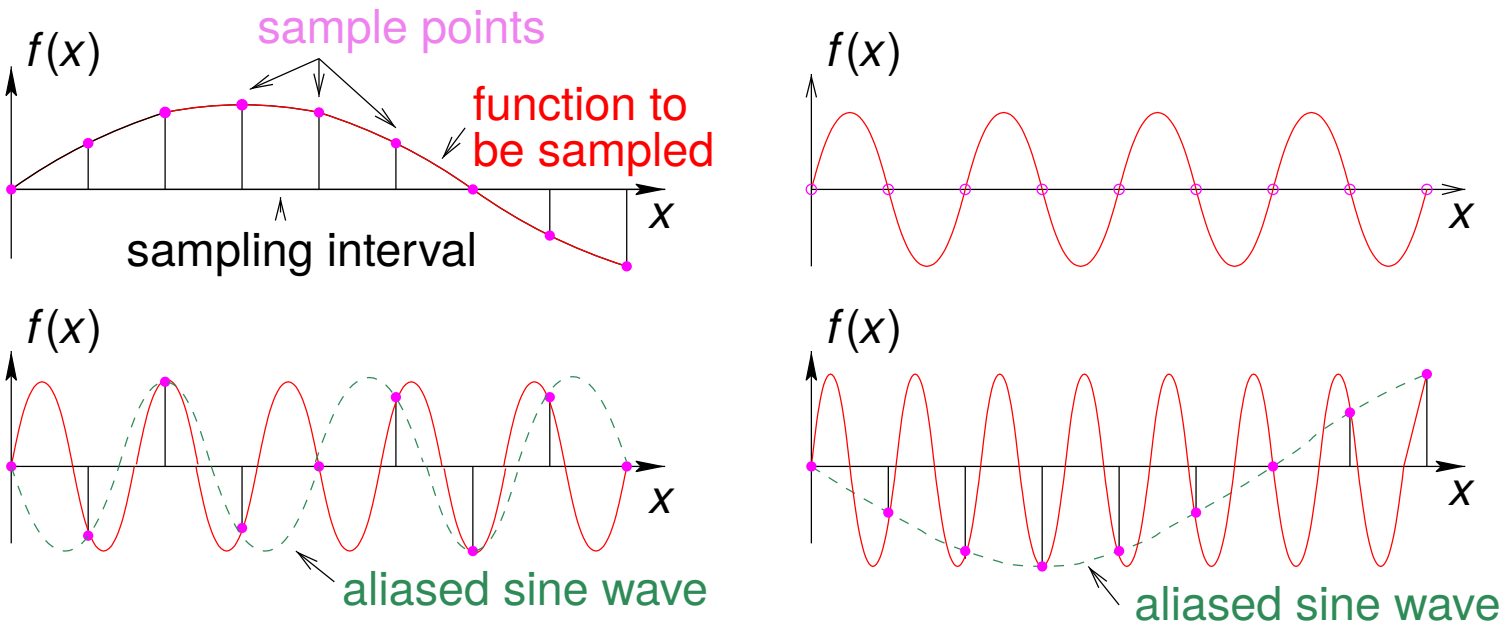




Nyquist-Shannon Sampling Theorem

Nyquist-Shannon sampling theorem (Dt.: Abtasttheorem)

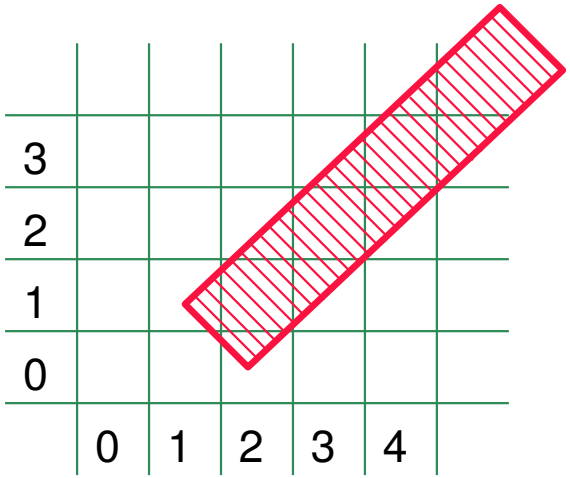
The Nyquist-Shannon sampling theorem tells us that a periodic signal can be properly reconstructed from equally-spaced samples if the the sampling rate is greater than twice the frequency of the highest-frequency component in its spectrum. This lower bound on the sampling rate is known as the *Nyquist rate*.



- The phenomenon of high frequencies masquerading as low frequencies in the reconstructed signal is aliasing.

Anti-Aliasing: Area Sampling

- *Area sampling* (aka *prefiltering*), which is one of the more prominent anti-aliasing methods, attempts to assign an intensity to a pixel that depends on the percentages of the areas that are covered by the objects.
- The actual pixel intensity is obtained as a weighted average of the object intensities, by using the overlap areas as weights.

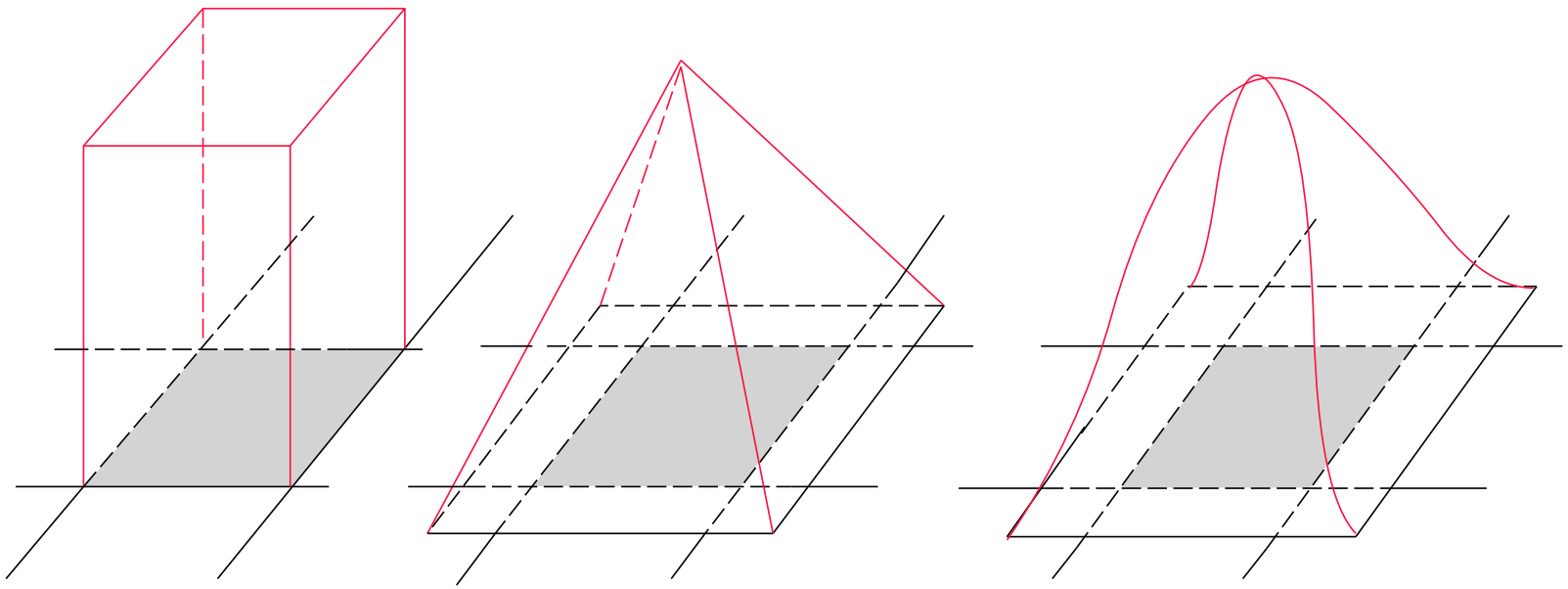


pixel	%(intensity)
(1,0)	5%
(1,1)	30%
(2,1)	95%
(3,1)	50%

- Despite several advances, prefiltering is rarely used since it tends to be computationally expensive.

Anti-Aliasing: Supersampling

- In *supersampling* (aka *postfiltering*) more than one sample per pixel is evaluated.
- In practice, supersampled images are computed by applying standard image-generation techniques to an n-times increased resolution, and by obtaining the value of an actual pixel as the (weighted) average of its corresponding n^2 supersampled pixels.
- Rather than combining samples with an unweighted average one might use a weighted filter: (a) *Box filter* (unweighted average), (b) *Bartlett filter* (linearly weighted average), (c) *Gaussian filter*.



(a) (b) (c)

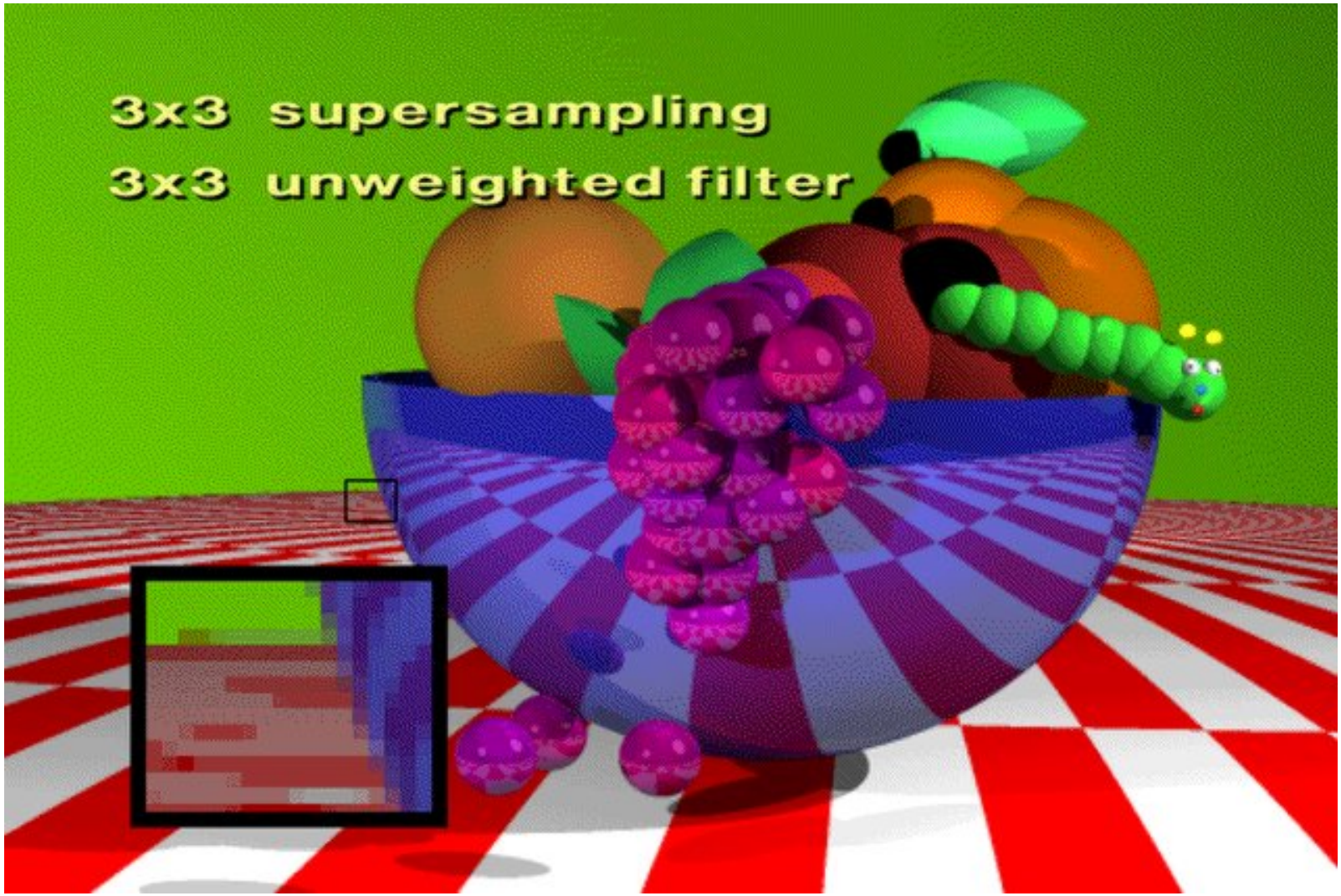
Anti-Aliasing: Supersampling

- Note that $n \times n$ supersampling increases the number of samples and the image-generation time by a factor of n^2!
- Supersampling works well with most computer graphics images and is easily integrated into a depth-buffer algorithm.
- Main draw-backs:
  - Non-adaptive supersampling does not work with images whose spectral energy does not fall off with increasing frequency. (Texture rendered in perspective is a common example of an image that does not exhibit a falling spectrum with increasing spatial frequency.)
  - Blurring effects occur because information is integrated from a number of neighboring samples.
  - The fixed, regular grid used for supersampling may create a variety of new aliasing artifacts in an image: Humans tend to recognize regular patterns!
- Variants: Adaptive supersampling and stochastic/jittered supersampling.

Anti-Aliasing: Sample Images

[Image credit: SIGGRAPH Educator's Slide Sets.]

Anti-Aliasing: Sample Images



[Image credit: SIGGRAPH Educator's Slide Sets.]

Anti-Aliasing: Sample Images

[Image credit: SIGGRAPH Educator's Slide Sets.]

Adding Surface Details

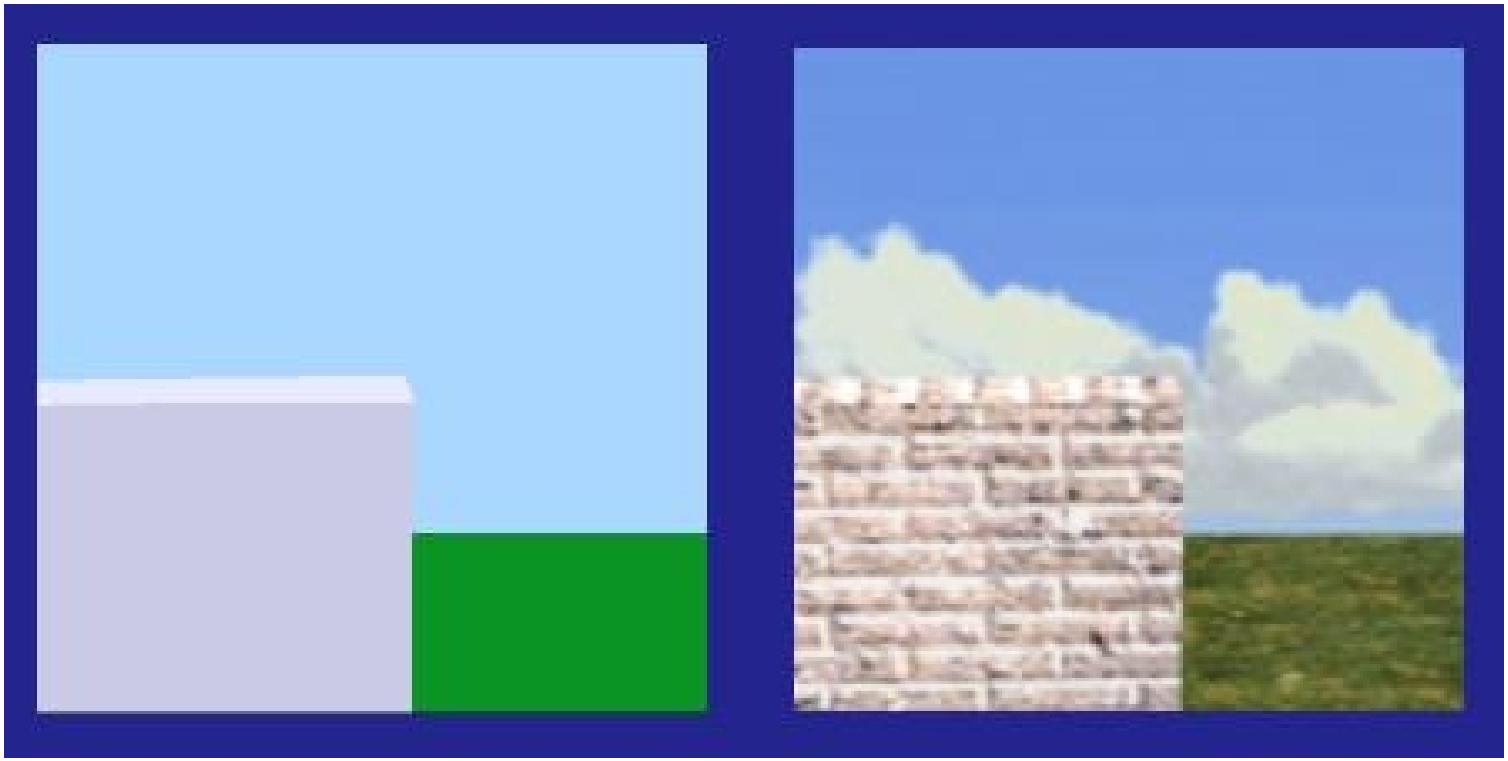
- All the shading techniques dealt with so far produce uniform and smooth surfaces — in sharp contrast to real-world surfaces!
- The simplest approach to add gross detail is to use so-called *surface-detail polygons*.
- Every surface-detail polygon is coplanar with its base polygon, and is marked in order to exclude it from hidden-surface removal.
- As details become finer and more intricate, explicit modeling with polygons or other geometric primitives becomes less feasible . . .
- [Catmull (1974):] Suggested as an alternative to map an image, either digitized or synthesized, onto a surface.
- This approach is known as *texture mapping* (or *pattern mapping*).

Ed Catmull

Catmull is the recipient of four Academy Awards (1993, 1996, 2001, 2008); he is a former president of Pixar and Walt Disney Animation Studios.

Adding Surface Details

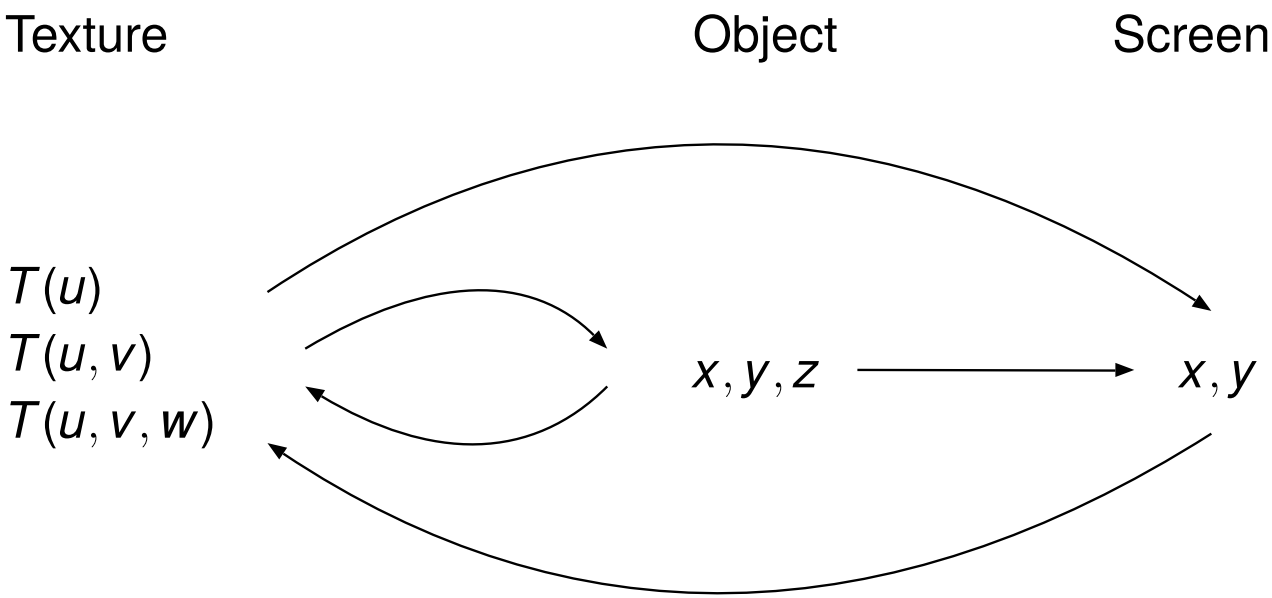
- Even rudimentary textures make an image much more pleasing and convey additional information! (Both images shown below are based on the same number of polygons.)



[Image credit: SIGGRAPH Educator's Slide Sets.]

Dimensionality of Texture Space

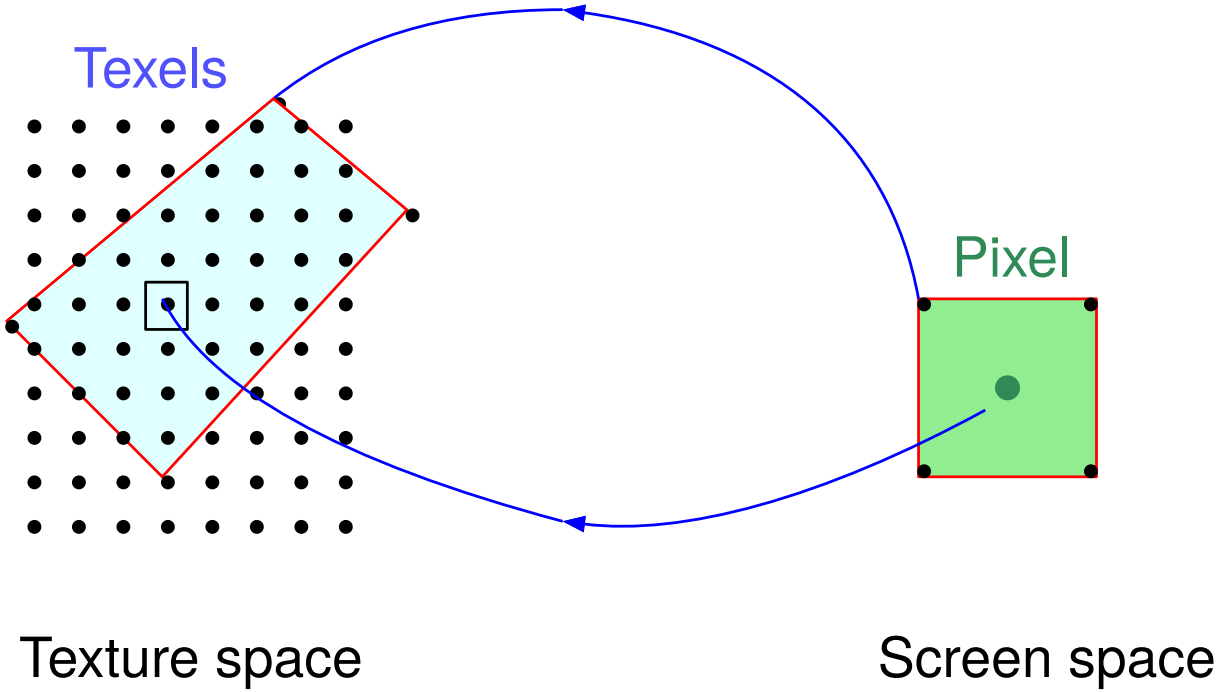
- The process of mapping a texture onto an object is called *texture mapping*.
- The texture space can be one-dimensional, two-dimensional, or three-dimensional.



- For the sequel, except for the slides on “solid texturing”, we will focus on a 2D texture space.

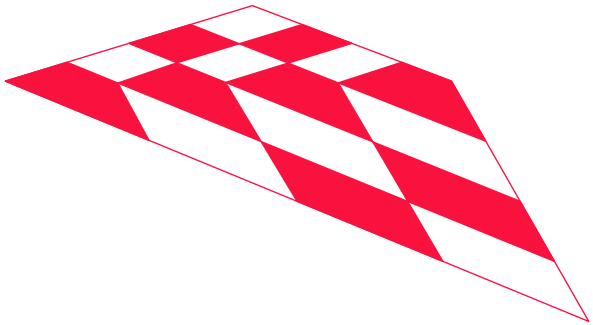
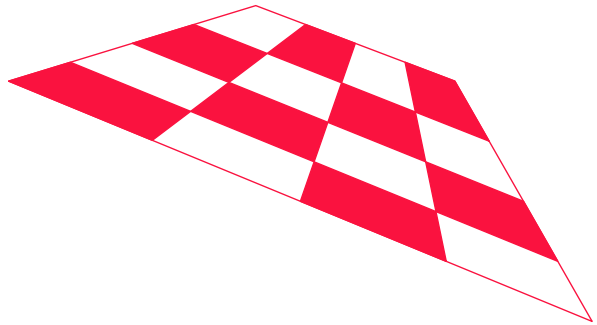
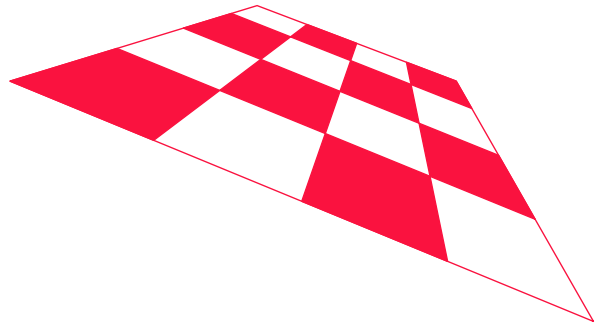
Texture Map

- The 2D image that is mapped onto an object is called a *texture*, and its individual elements are often referred to as *texels*.
- A one-to-one mapping between pixels and texels need not exist!



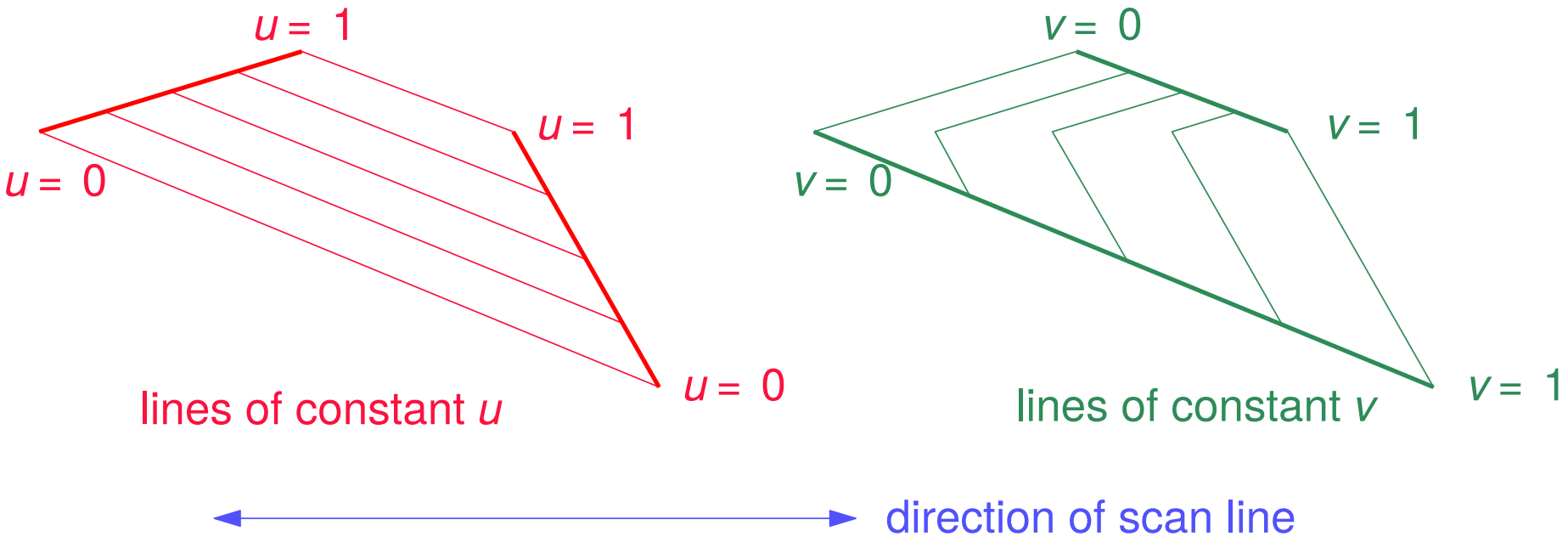
Texture Mapping Caveats

- A simple-minded approach to texture mapping assigns texture coordinates to vertex coordinates and uses a sweep line (scan line) for the interpolation of the texture coordinates
- This approach may result in horrible anomalies, e.g., the “bent” checkerboard.
- An improved approach performs the interpolation in texture and screen space in parallel.
- This approach avoids bent lines, but still gives way to incorrectly spaced lines, i.e., to incorrect perspective views.

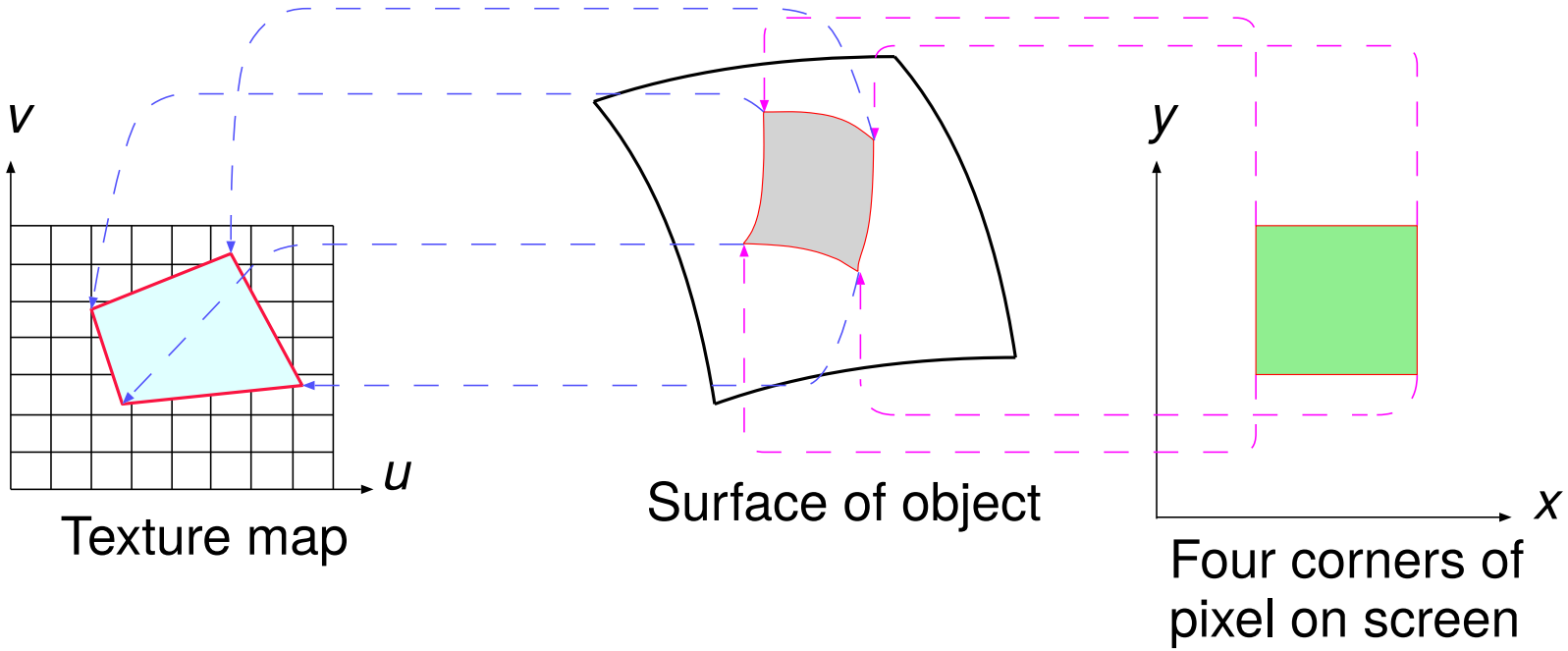
Texture Mapping Caveats

- The bent checkerboard is caused by the scan-line interpolation (in image space) of the texture coordinates assigned to the vertices of the quadrilateral (in object space).



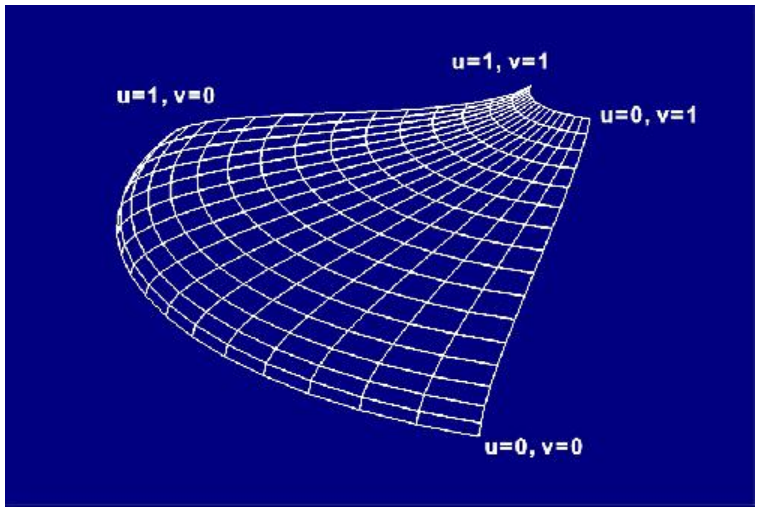
Correct Texture Mapping

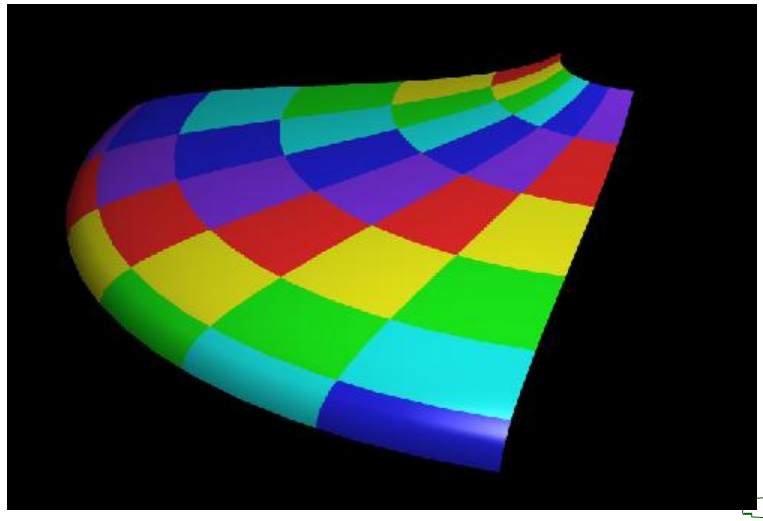
- The only remedy applicable is to determine the texture coordinates explicitly for every pixel via transformations from screen space to object space, and from object space to texture space.



Texture Mapping Problems: Inverse Mapping

- In general, this inverse mapping from screen to the object surface and to the texture space is highly non-trivial. This is particularly true for curved objects.
- Two approaches are commonly used:
  - Unfolding the polygon mesh: The dimensionality is reduced from 3D to 2D by “unfolding” adjacent polygons, thus generating a flat polygonal mesh which is easier to project into the texture space.
  - Two-part mapping: The 3D object surface is mapped to an intermediate surface (such as a cylinder), and then into the texture space.
- For texturing a parametric surface its parametric representations can be employed. Note, though, that parametric mapping does not take care of perspective foreshortening!

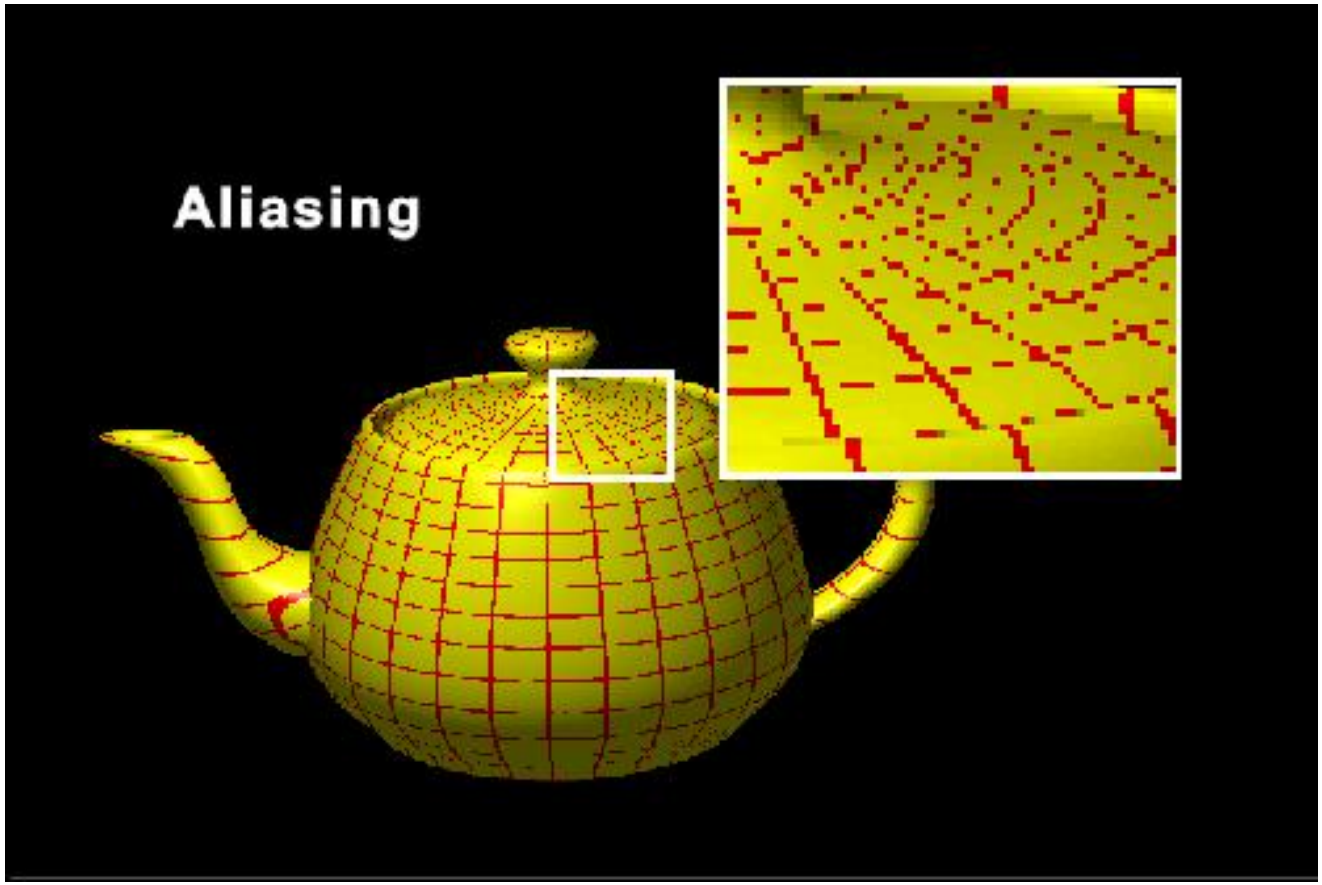




[Image credit: SIGGRAPH Educator’s Slide Sets.]

Texture Mapping Problems: Aliasing

- Even if the inverse mapping is performed accurately, serious aliasing errors may occur, and, in the worst case, may ruin the visual appearance of a textured object.



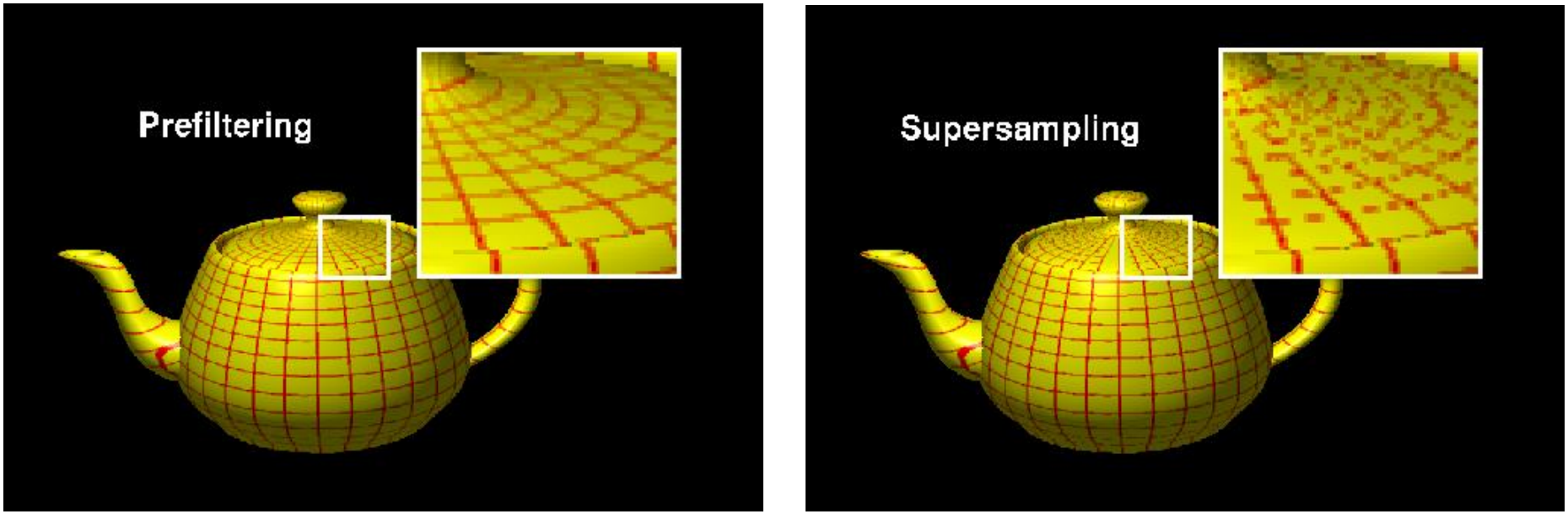
[Image credit: SIGGRAPH Educator's Slide Sets.]

Texture Mapping Problems: Aliasing

- Aliasing is due to the point sampling problem in texel space and due to perspective foreshortening.
- Typically, all texels that correspond to the area of a pixel are summed by applying a weighing process (such as a box filter).
- However, neighboring pixels need not map to neighoring texels, and information of the texture map may be lost.
- This is particularly problematic if the texture contains thin curves or small details.
- Also, simple box filtering is not sufficient in the case of perspective foreshortening because texels in the back have the same weight as texels in the front.
- Such aliasing artifacts are particularly noticeable for textures that exhibit coherence or regularity (e.g., a checkerboard).
- Correct filtering of non-linearly mapped areas would require space-variant filters, i.e., filters whose shape and area change as they move across the texture domain.
- However, this is time-consuming. And prefiltering, supersampling or mipmapping do go a long way to reduce aliasing artifacts.

Texture Mapping Problems: Aliasing

- Prefiltering and supersampling to cope with the point sampling problem in texture space.



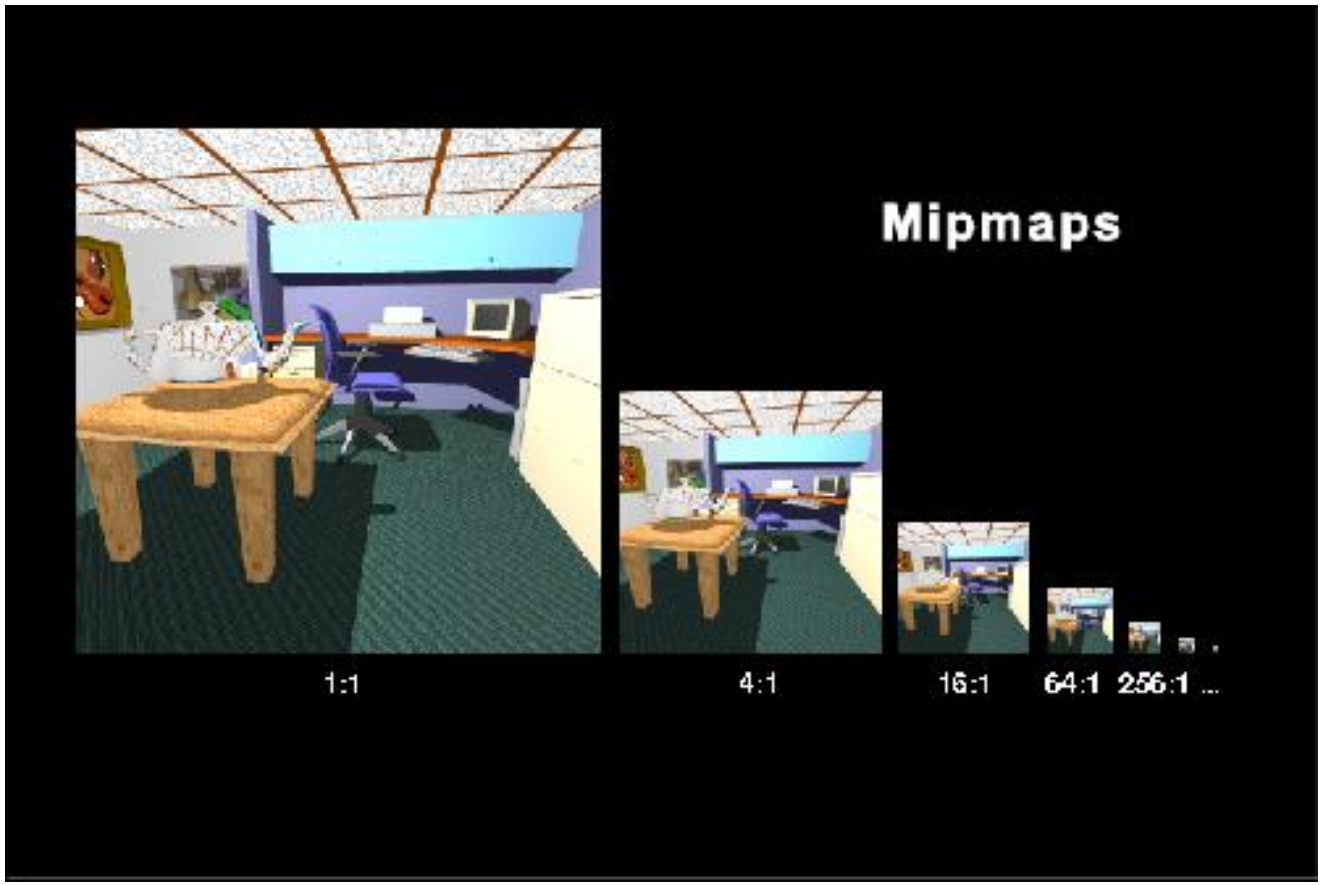
[Image credit: SIGGRAPH Educator's Slide Sets.]

Texture Mapping Problems: Fixed Resolution

- A major problem of texturing is that the texture space has a fixed resolution!
- Too small a resolution causes zooming to result in poor-quality images.
- Too high a resolution causes blurring and other aliasing problems, such as Moiré patterns, and is a source for computational inefficiency.
- What is a good resolution??
- Even if the resolution of the texture space has been chosen judiciously, an extremely large number of texels may have to be weighted and summed just to texture a single pixel.
- This phenomenon may arise when a large number of texels maps onto a surface, but the projection of the surface in screen space is small, either because of its depth or because of its orientation with respect to the viewing direction.

Mipmapping

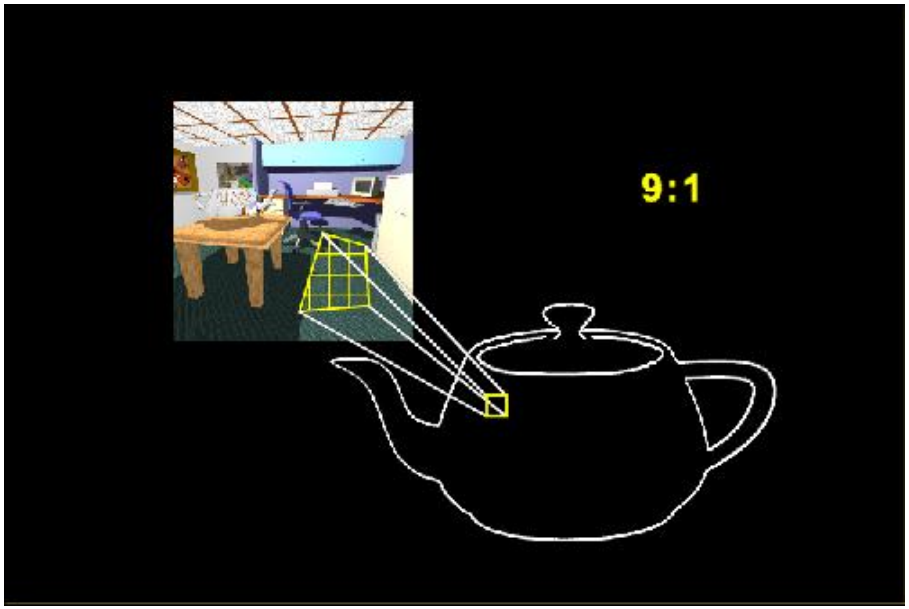
- Mipmapping (aka MIP mapping) uses textures at diverse resolutions.
- MIP: “multum in parvo”, i.e., “much in little”.
- Generation of mipmap: Succesive averaging or Fourier transform.
- Memory consumption goes up by at most 33%.

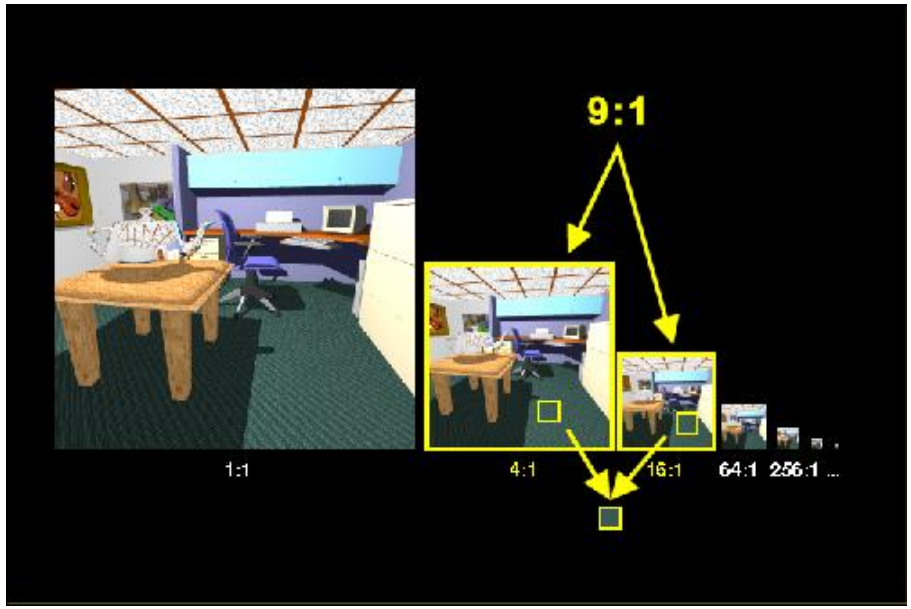


[Image credit: SIGGRAPH Educator’s Slide Sets.]

Mipmapping

- To compute the color of a pixel, we determine the number of texels that correspond to the pixel in the original texture map, find the two texture maps closest in size, and average the pixel colors obtained from those two texture maps.
- Alternatively, and simpler, find the texture resolution such that one pixel is fully covered by one texel.

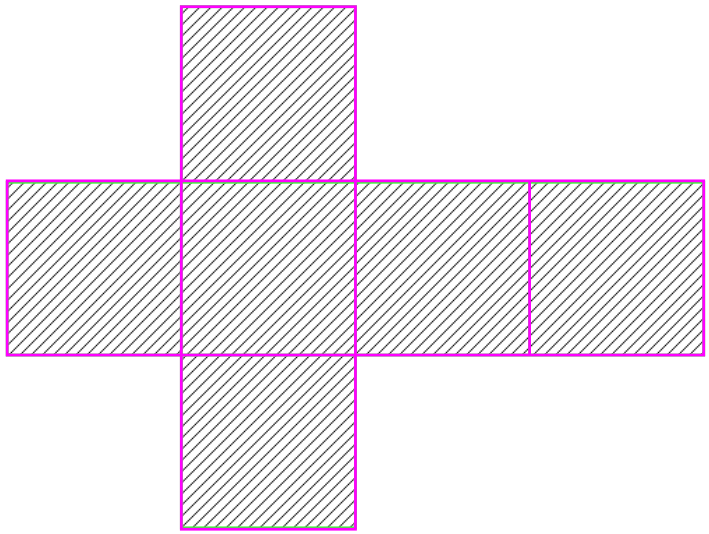


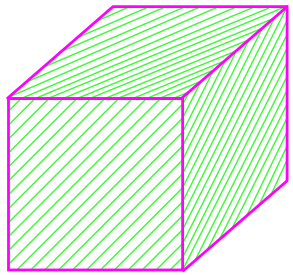


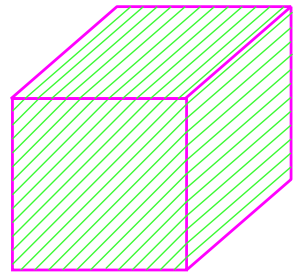
[Image credit: SIGGRAPH Educator's Slide Sets.]

How to Make a Texture Seamless?

- For wood, granite, marble, and other natural materials, simply pasting a 2D texture onto the exterior of the object does not give the desired result: the texture does not appear to be seamless!



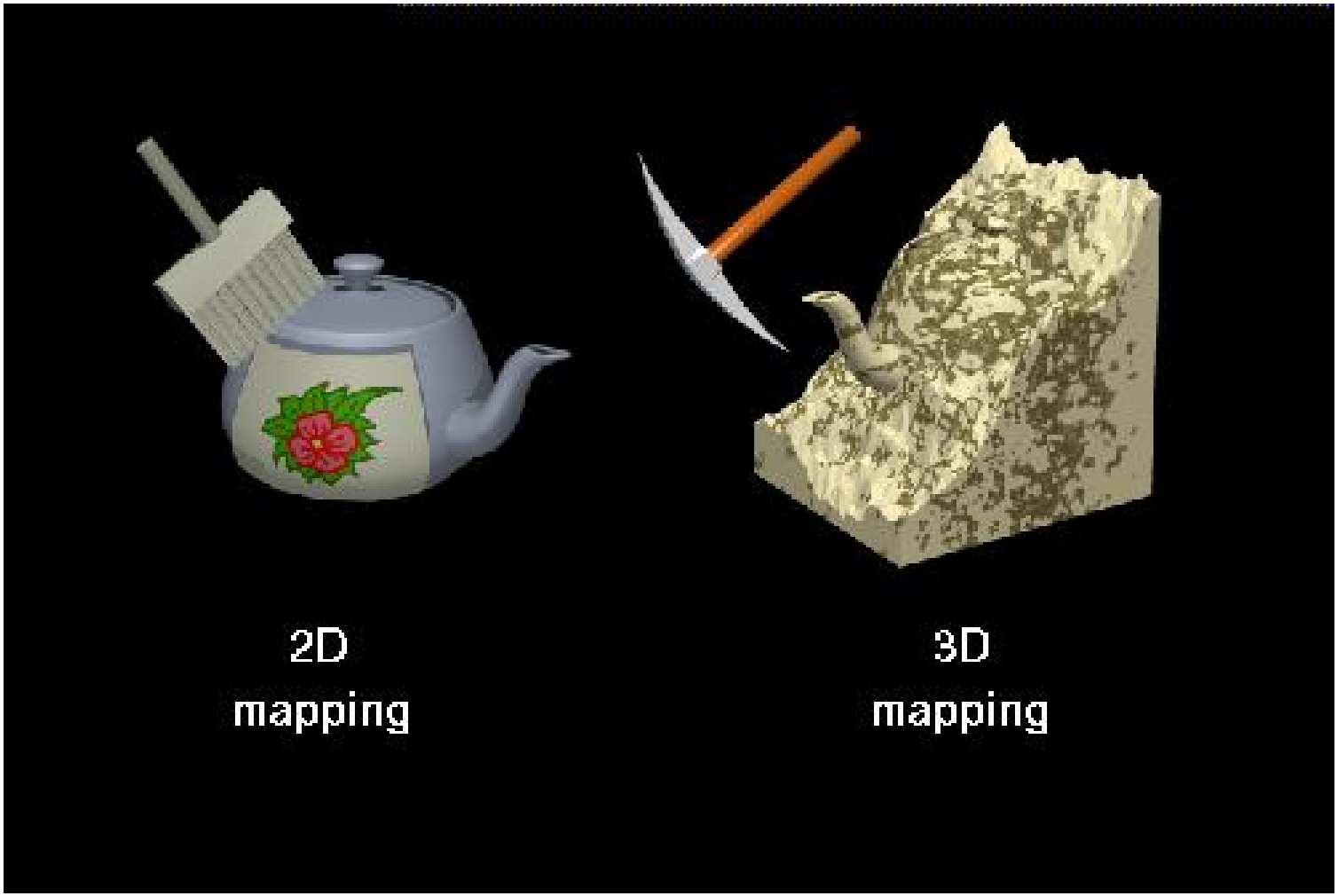




UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

2D Textures Versus Solid Textures

- 3D (“solid”) textures allow the user to “carve” an object out of a solid block.



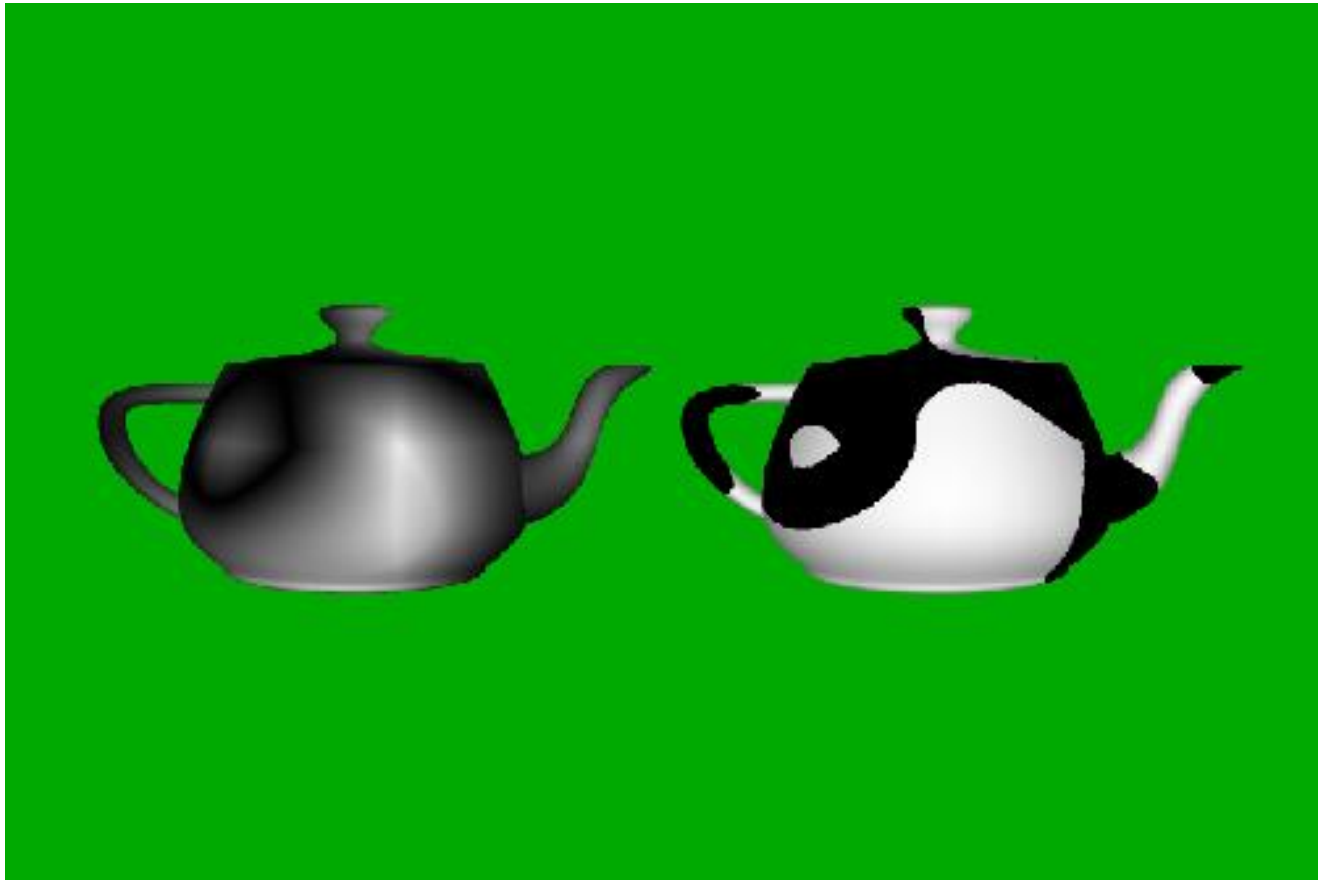
[Image credit: SIGGRAPH Educator’s Slide Sets.]

Solid Texturing

- In order to avoid gaps and inconsistencies, and also to circumvent the mapping problem, a 3D texture space can be employed.
- Imagine that a texture value exists for every point in 3D object space.
- We may then assume that the texture coordinates of a point on a 3D surface are given by the identity mapping.
- The color of the object is determined by the intersection of its surface with the predefined 3D texture field of the block.
- This is equivalent to sculpting or carving an object out of a solid block of material.
- A major advantage of the elimination of the mapping problem is that objects of arbitrary complexity can receive a texture on their surface in a “coherent” fashion.
- In an animated sequence, the texture space would have to be transformed in the same way as the object space. (The “incorrect” approach of moving the object through the texture space can produce unique visual effects, though.)
- A digitizing approach is not applicable to generate solid textures due to memory constraints.
- Typically, 3D textures are generated procedurally.
- A 3D texture can also be generated by sweeping a 2D texture through 3D space.

Sample Solid Texture: Black&White

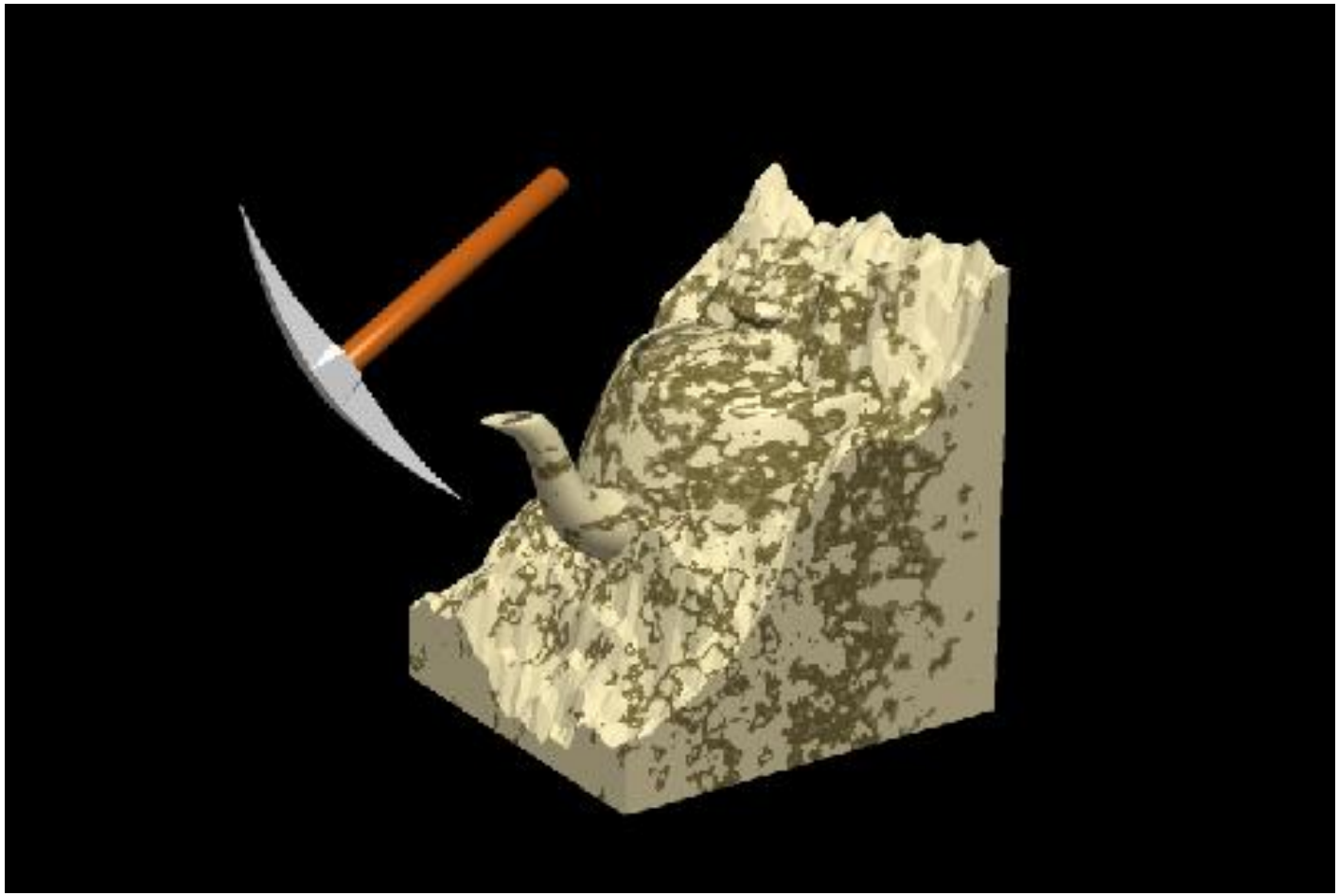
- Again assign random numbers to the vertices of a regular (coarse) grid.
- Obtain intermediate texture values by (bilinear) interpolation. If a texture value is above a threshold then paint the pixel white, else black.



[Image credit: SIGGRAPH Educator's Slide Sets.]

Sample Solid Texture: Marble

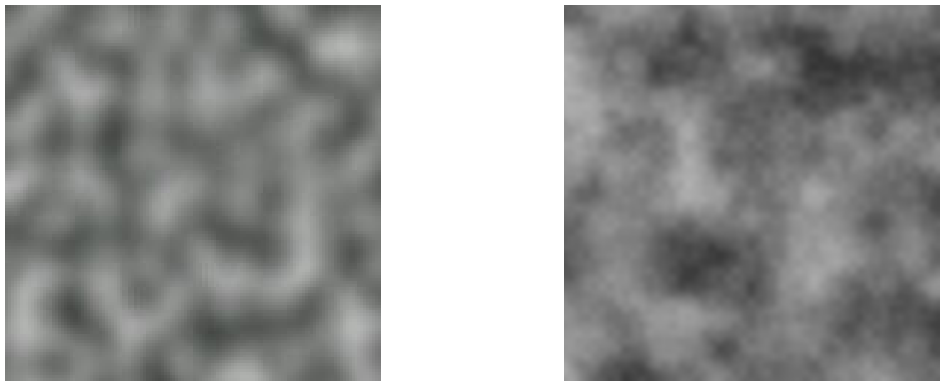
- Marble textures are typically generated as procedural textures, based on noise functions, e.g., Perlin noise [1982] or Perlin simplex noise [2001].



[Image credit: SIGGRAPH Educator's Slide Sets.]

Sample Solid Texture: Perlin Noise

- Developed by Ken Perlin for “TRON”, with work started in 1981.
- Published in 1985 as a SIGGRAPH paper on “An Image Synthesizer”.
- Computational costs: $O(2^d)$ for the interpolation of the 2^d corners of a cell in $\mathbb{R}^d$.
- Perlin noise is coherent, i.e., the noise function changes smoothly as one moves across the texture space.

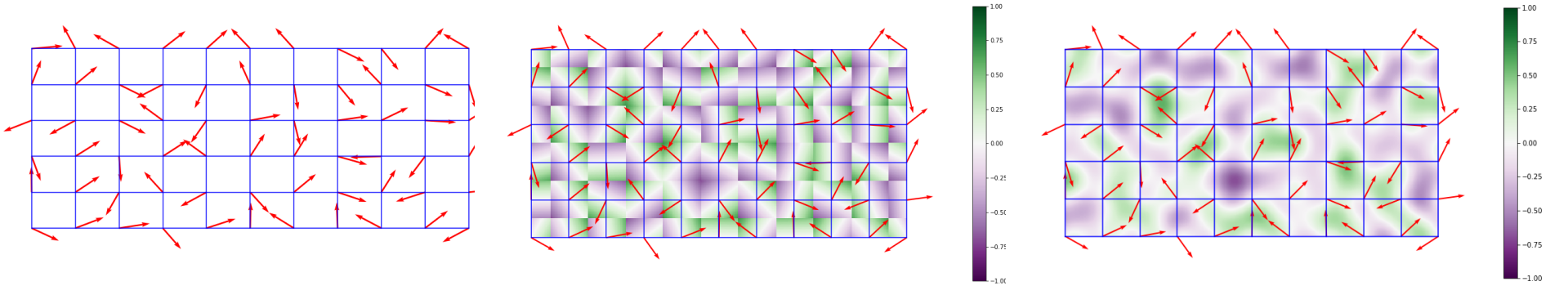


Standard and fractional (aka hierarchical) Perlin noise. [Image credit: Wikipedia.]

- Simplex noise:
  - Replaces a d-dimensional cube by a simplex, i.e., a d-dimensional “triangle”.
  - The complexity can be brought down to $O(d^2)$.
  - It has no directional artifacts. (At least none that are easily visible.)
- Perlin was awarded an Academy Award for Technical Achievement in 1997, and received a patent for the use of implementations of simplex noise.

Sample Solid Texture: Perlin Noise

- Initialization:
  - Allocate a regular 3D grid within $[0, 1]^3$ or within $[-1, 1]^3$.
  - Assign a random unit vector ("gradient vector") to each node of the grid.
- Noise function:
  - Determine the (cubic) grid cell that contains a texture point $p := (u, v, w)$.
  - For each node of that cell:
    1. Compute a direction vector from the node to p (aka "distance vector").
    2. Compute the dot product between this vector and the corresponding gradient vector.
  - Compute an interpolation of the values obtained; the interpolation function has zero first derivative at the nodes of the grid.



[Image credit: https://en.wikipedia.org/wiki/Perlin_noise]

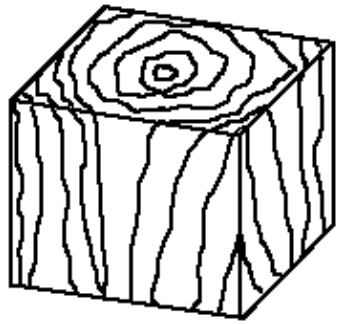
Sample Solid Texture: Perlin Noise for Terrain Generation

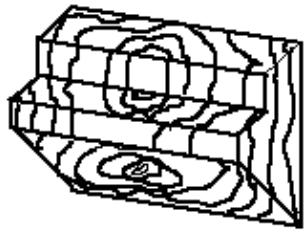
[Image credit: Wikipedia]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Sample Solid Texture: Wood Grain

- Wood grain can be simulated by a set of concentric cylinders, whose reference axis is, in general, tilted with respect to a reference axis of the object.





- The texture field is given by a modular function of the radius, returning a color for texture space coordinates (u, v, w).

$$r := \sqrt{u^2 + v^2}$$

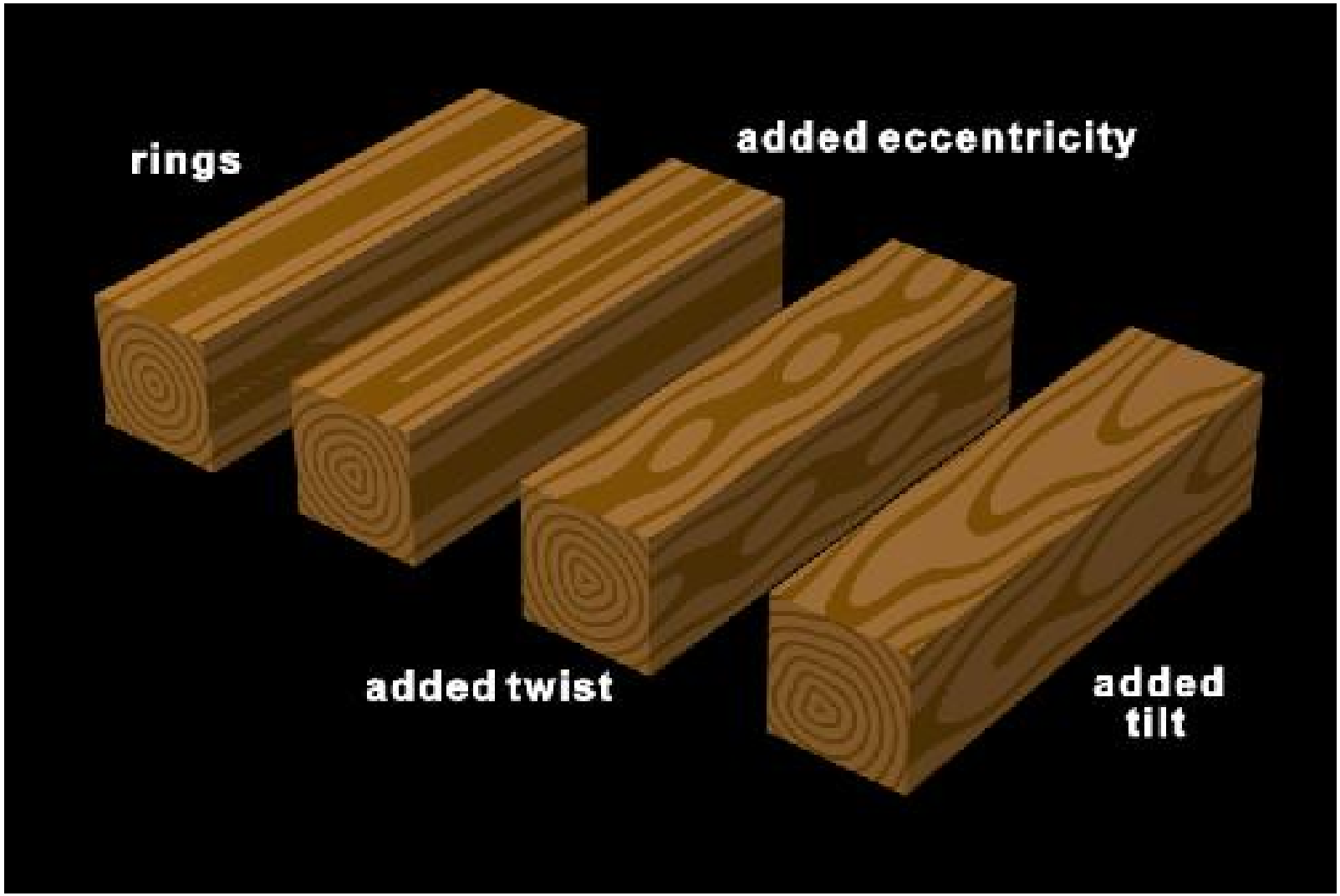
$$\alpha := \arctan \frac{u}{v}$$

$$r := r + 2 \sin(20\alpha + \frac{w}{150})$$

$$c := \lfloor r \rfloor \bmod 60$$

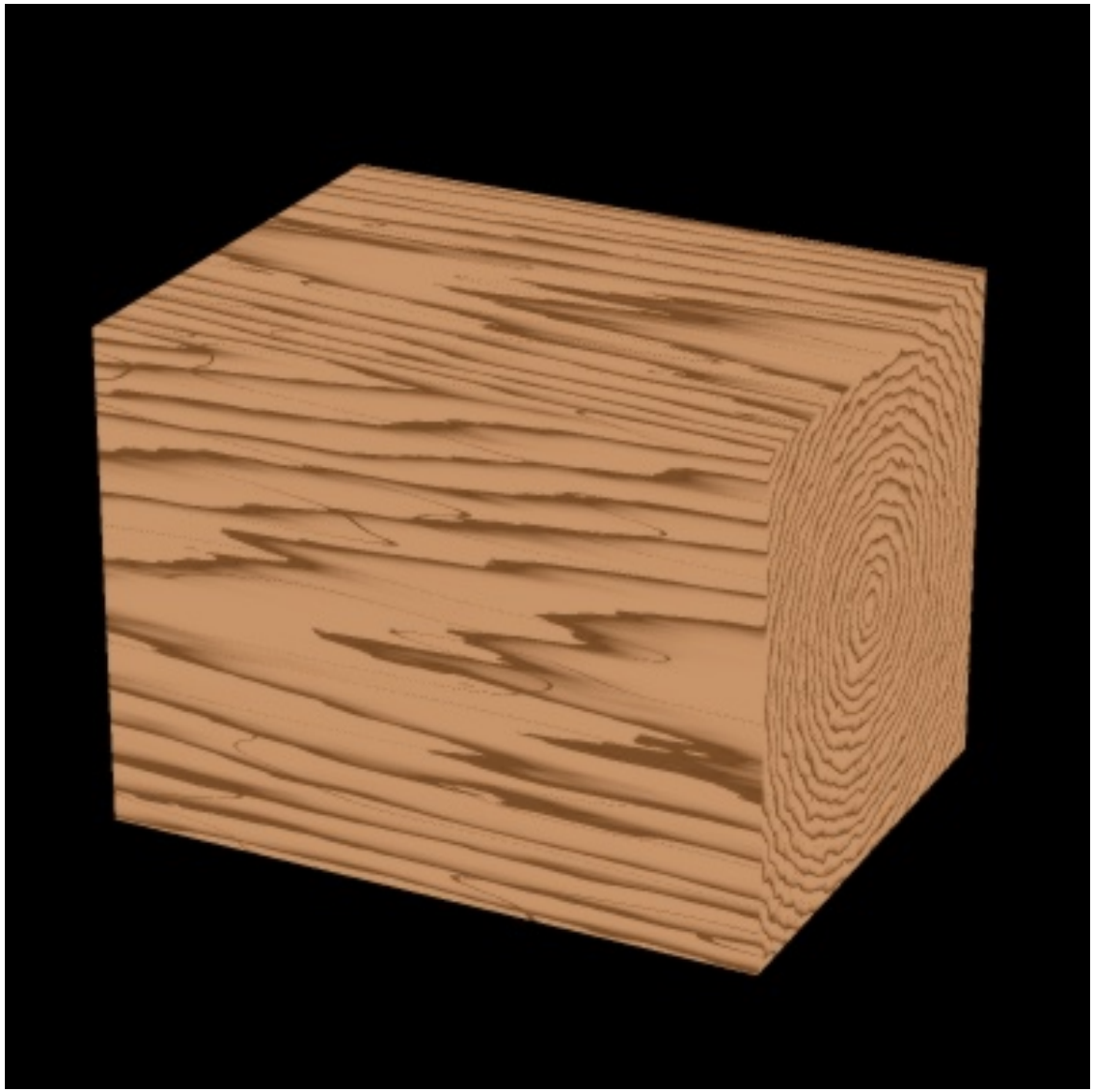
- Assign a dark brownish color if $c > 40$, and a light color, otherwise.

Example: Wood Grain



[Image credit: SIGGRAPH Educator's Slide Sets.]

Example: Wood Grain

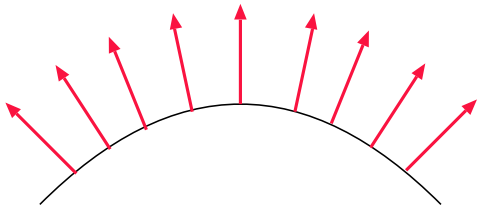


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

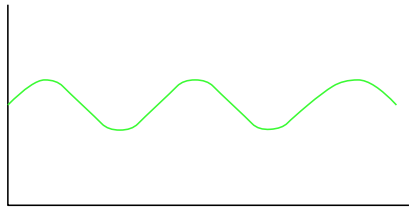
Bump Mapping

- Texture mapping affects the shading of a surface, but the surface will still look smooth: It changes the color, diffuse, and specular reflection properties, but it does not change the surface normal.
- How can we make a surface look rough?
- Use a photograph of a rough surface?
- If a photograph of a rough surface is used as a texture map then the shaded surface will not look quite right because the direction to the light source used to create the texture map is likely different from the direction to the light source illuminating the surface.
- Blinn's *bump mapping* (1978) is a way to provoke the appearance of a rough surface geometry that avoids explicit geometric modeling.
- It involves perturbing a surface normal before it is used in the illumination model, just as a slight roughness in a surface would perturb the surface normal in the real world.
- A *bump map* is an array of displacements, each of which can be used to simulate displacing a point on a surface a little above or below that point's actual position.

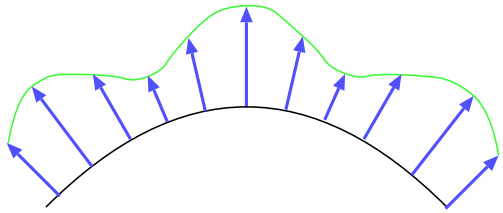
Bump Mapping



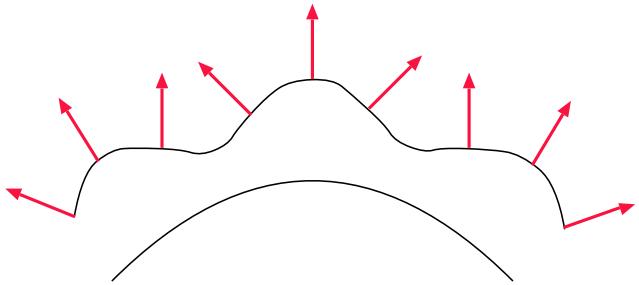
Original surface



A bump map



Simulate the displacement of the surface

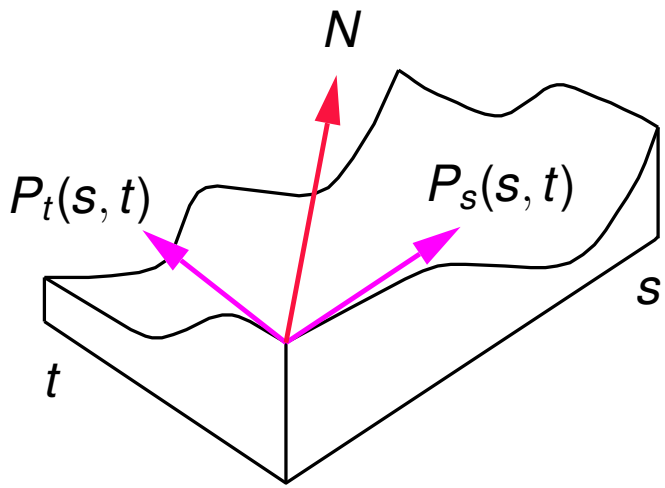


Normal vectors to the 'new' surface

Bump Mapping for a Parametric Surface

- Let $P(s,t)$ be a parametric surface.
- To obtain its unit surface normal $n(s,t)$ at point $P(s,t)$, we compute

$$P_s(s,t) := \frac{\partial P(s,t)}{\partial s},$$
$$P_t(s,t) := \frac{\partial P(s,t)}{\partial t},$$
$$N(s,t) := P_t(s,t) \times P_s(s,t),$$
$$n(s,t) := \frac{N(s,t)}{\|N(s,t)\|}.$$



- Let $B(s,t)$ be the bump map value that will be applied at $P(s,t)$. (For simplicity, we assume that the bump map is also parameterized over s,t.)
- We add this amount in the direction normal to $P(s,t)$, thus obtaining a new point $P^*(s,t)$:

$$P^*(s,t) := P(s,t) + B(s,t) \cdot n(s,t).$$

Bump Mapping for a Parametric Surface

- Compute partial derivatives of $P^*(s,t) := P(s,t) + B(s,t) \cdot n(s,t)$:

$$\begin{aligned} P_s^*(s,t) &= \frac{\partial P^*(s,t)}{\partial s} = P_s(s,t) + B_s(s,t) \cdot n(s,t) + B(s,t) \cdot n_s(s,t) \\ P_t^*(s,t) &= \frac{\partial P^*(s,t)}{\partial t} = P_t(s,t) + B_t(s,t) \cdot n(s,t) + B(s,t) \cdot n_t(s,t). \end{aligned}$$

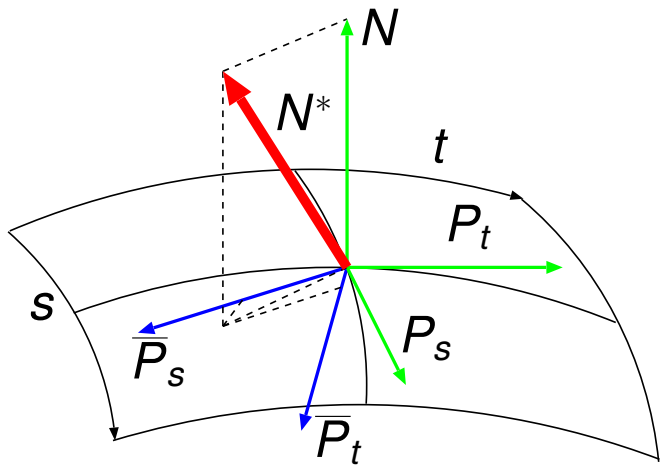
- Blinn showed that a good approximation to the new (unnormalized) normal N^* is obtained by ignoring the last term in each partial derivative and by taking their cross-product. (Recall that $(A+B) \times (C+D) = A \times C + A \times D + B \times C + B \times D$.)

$$\begin{aligned} N^*(s,t) &= P_t^*(s,t) \times P_s^*(s,t) \\ &\approx [P_t(s,t) + B_t(s,t) \cdot n(s,t)] \times [P_s(s,t) + B_s(s,t) \cdot n(s,t)] \\ &= P_t(s,t) \times P_s(s,t) + P_t(s,t) \times (B_s(s,t) \cdot n(s,t)) + \\ &\quad (B_t(s,t) \cdot n(s,t)) \times P_s(s,t) + (B_s(s,t) \cdot B_t(s,t) \cdot (n(s,t) \times n(s,t)) \\ &= N(s,t) + B_s(s,t) \cdot (P_t(s,t) \times n(s,t)) + B_t(s,t) \cdot (P_s(s,t) \times n(s,t)) \\ &= N(s,t) + \frac{B_s(s,t) \cdot (P_t(s,t) \times N(s,t))}{\|N(s,t)\|} + \frac{B_t(s,t) \cdot (P_s(s,t) \times n(s,t))}{\|N(s,t)\|} \end{aligned}$$

Computing the Altered Surface Normal

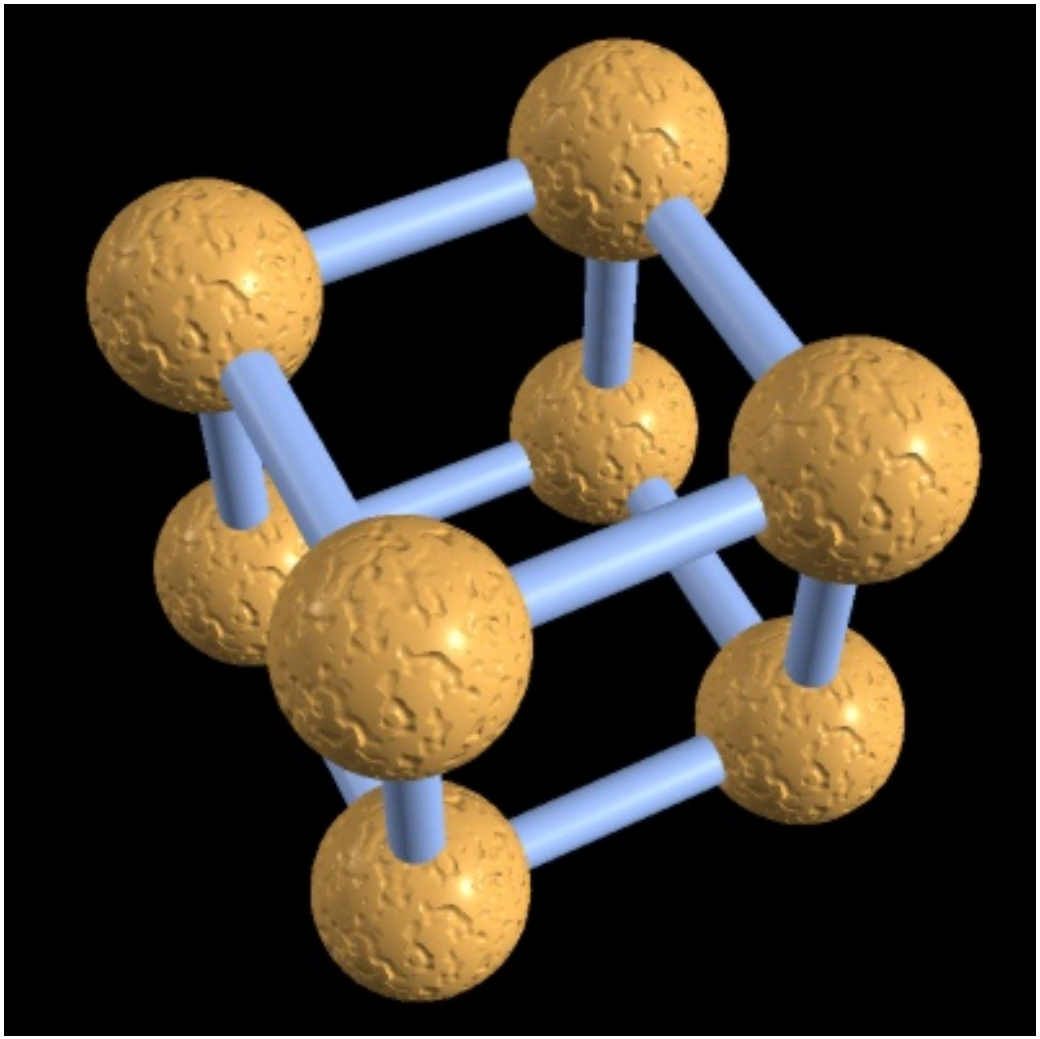
- By dropping the parameters, and with $\overline{P}_t := P_t \times N$ and $\overline{P}_s := P_s \times N$, we obtain the following more concise formula:

$$N^* = N + \frac{B_s(P_t \times N) - B_t(P_s \times N)}{||N||} = N + \frac{B_s\overline{P}_t - B_t\overline{P}_s}{||N||}$$



- $N^*(s,t)$ is then normalized and substituted for the true surface normal in the illumination equation at $P(s,t)$.
- Note that bump mapping does not actually compute the altered surface — it suffices to compute only (an approximation of) the altered normal!

Sample Bump Map



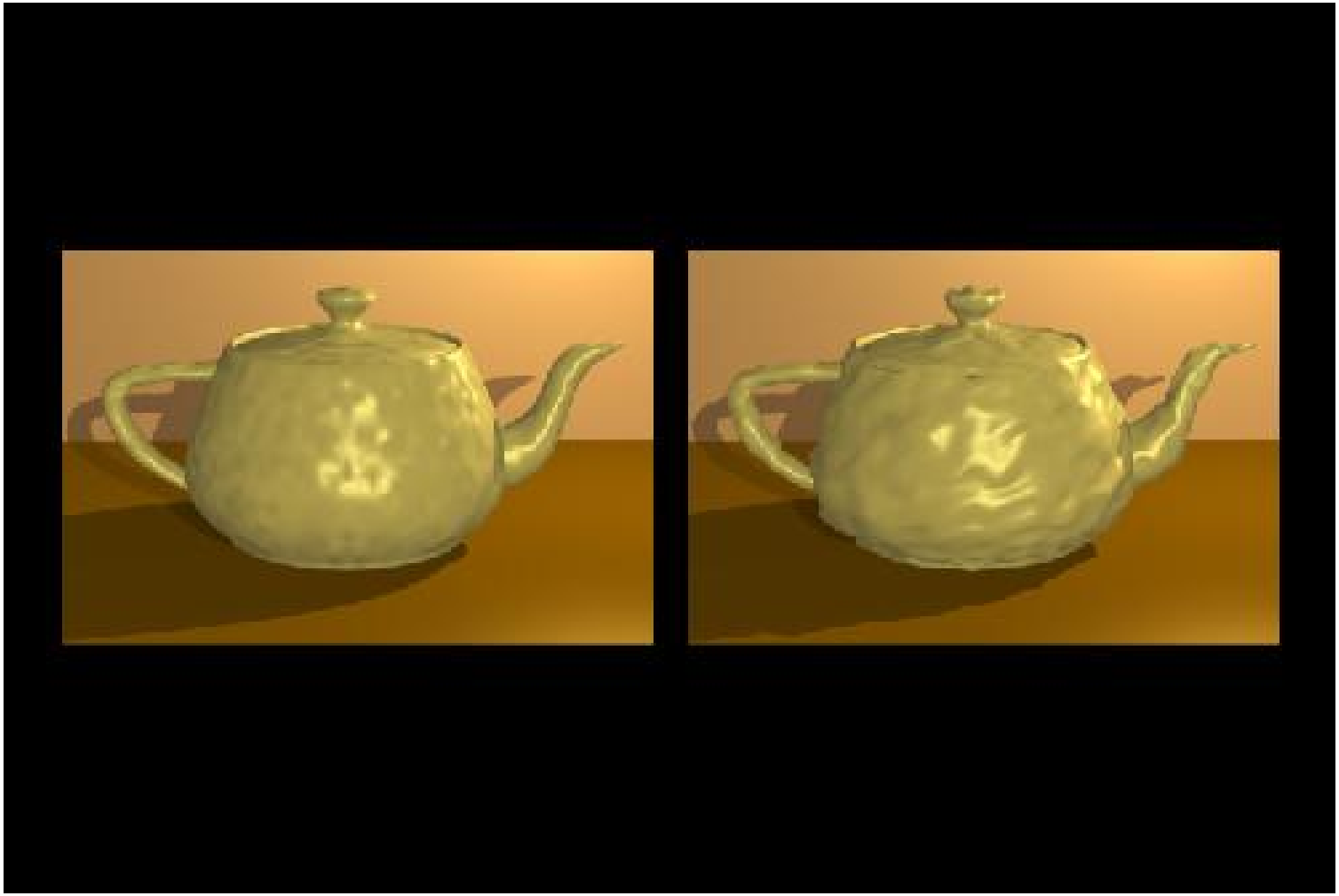
UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Displacement Mapping

- A method similar to bump mapping for adding wrinkles to a surface is displacement mapping.
- Displacement mapping is applied to a surface by first dividing the surface up into a mesh of coplanar polygons.
- The vertices of these polygons are then perturbed according to the displacement map.
- The resulting model is then rendered with any standard polygon renderer.
- Displacement mapping can be used to convert the visual appearance of a cylinder into a screw.
- However, to achieve a fine resolution in the texture of the wrinkles, the additional polygons would get ever smaller and more numerous, placing a tremendous burden on the renderer.

Displacement Mapping

- Displacement mapping does alter the object's geometry!

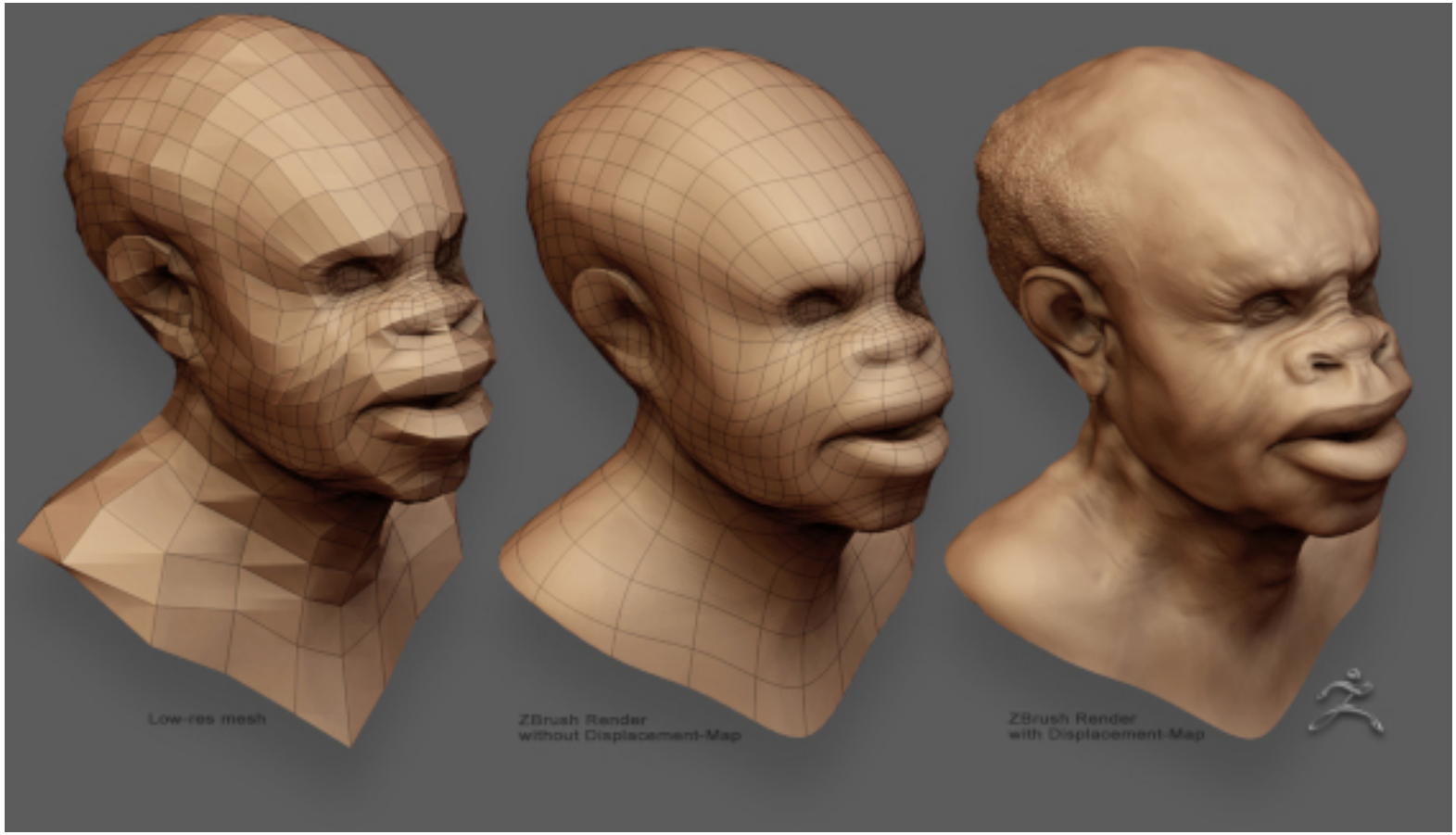


[Image credit: SIGGRAPH Educator's Slide Sets.]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Displacement Mapping in Conjunction with Subdivision Surfaces

- Bump mapping or displacement mapping in conjunction with (adaptive) subdivision surfaces is widely used in CGI for organic modeling.
- Recent GPUs provide hardware support!

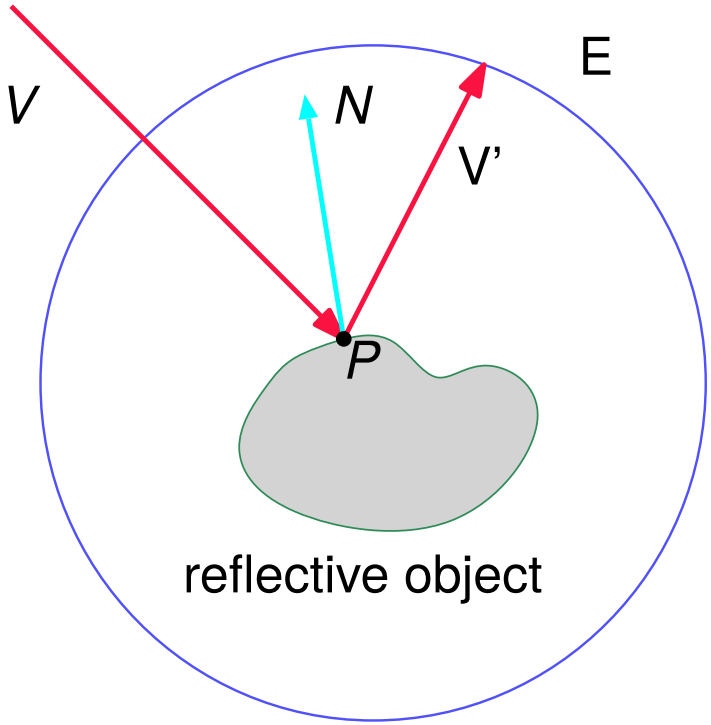


[Image credit: https://www.zbrushcentral.com]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

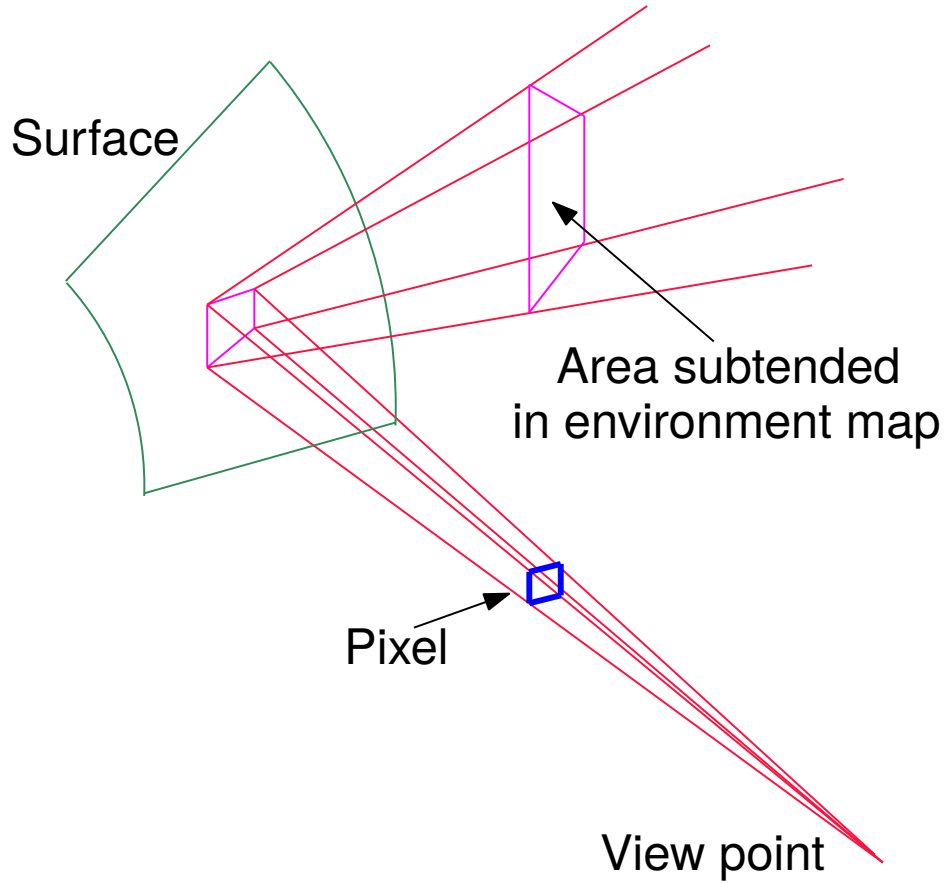
Reflection Mapping

- Reflection mapping (aka “environment mapping”) refers to the process of reflecting the surrounding environment on a shiny, reflective object without resorting to ray tracing (or similar means).
- Let V' be the reflection vector of a viewer direction V for a particular point on the surface of the reflective object.
- The intersection of V' with a surface, such as the interior of a sphere that contains an image of the environment to be reflected in the object, gives the shading attributes for the point P on the object surface.



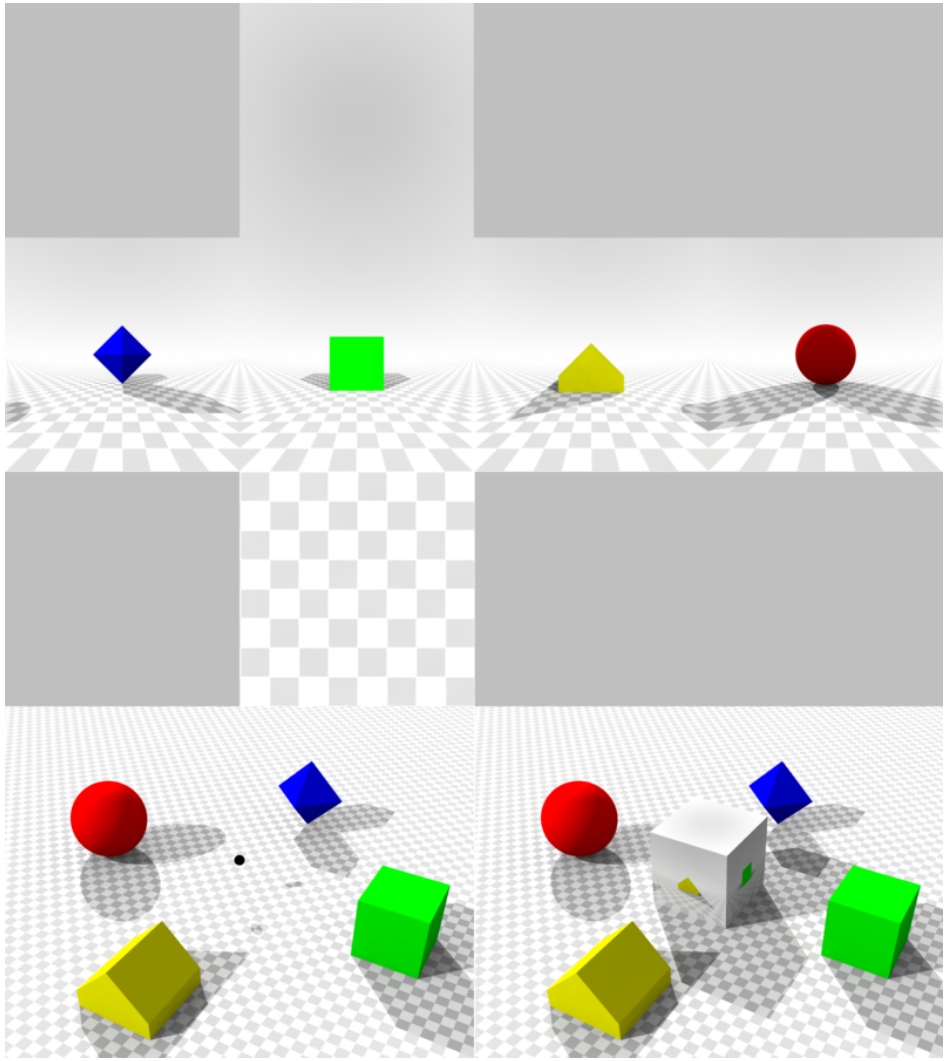
Reflection Mapping

- In practice, four rays through the pixel point define a reflection “cone” with a quadrilateral cross-section. The region that subtends the environment map is then filtered to give a single shading attribute for the pixel.



Reflection Mapping: Cube Mapping

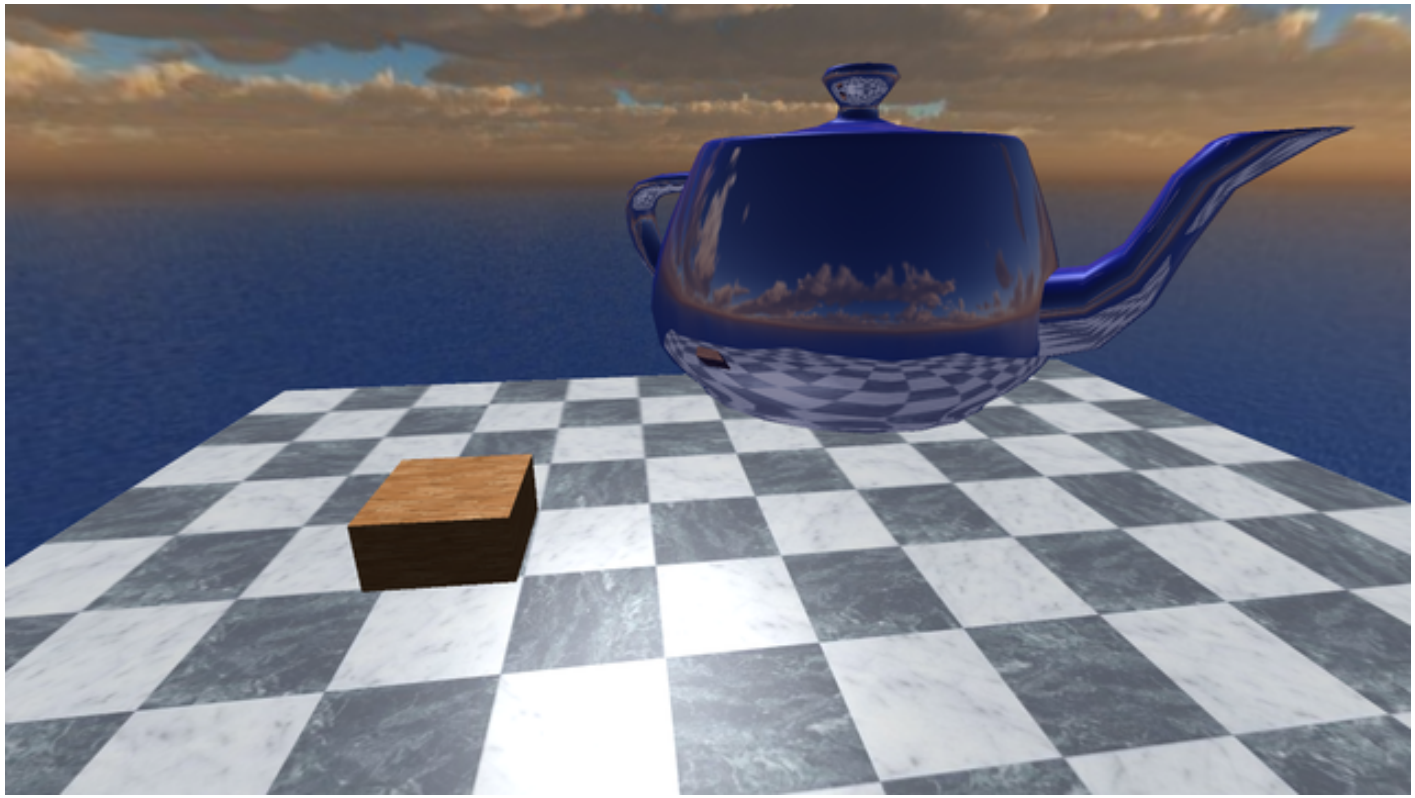
- Nowadays, the environment is mostly mapped into a cube (“cube mapping”) or some other polyhedral object.
- This allows to store the six reflection maps as textures.
- E.g., “sky box”.



[Image credit: Wikipedia.]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Reflection Mapping: Sample Cube Map



[Image credit: Wikipedia.]

Problems with Reflection Mapping

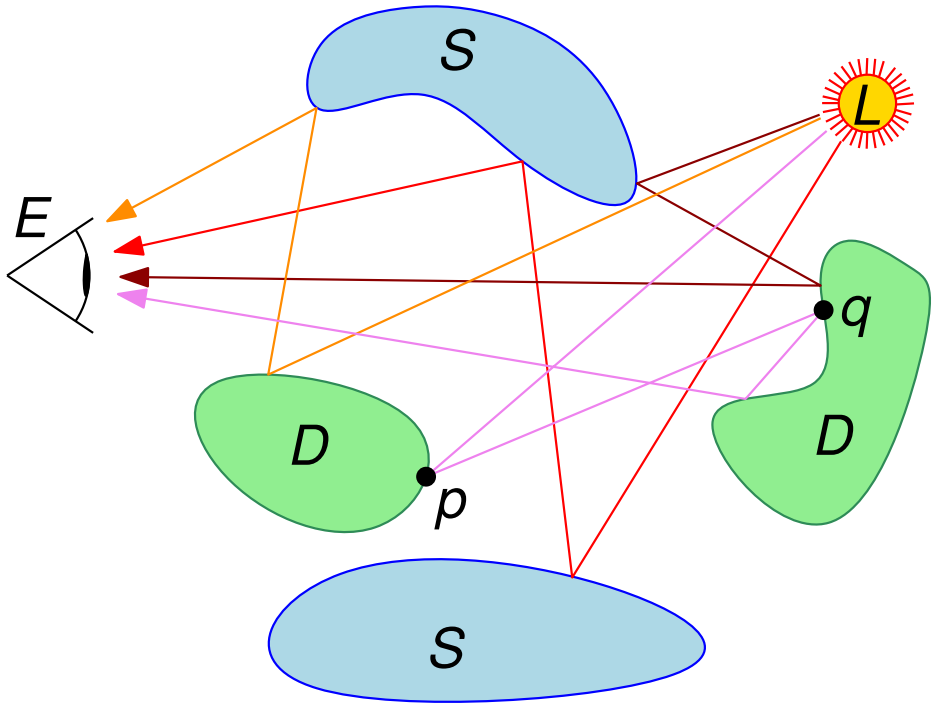
- A practical difficulty is in the production of the environment map. Predistorting an (photographed) environment to fit the interior surface of a sphere is difficult, and storing a reflection on the six sides of a cube requires tricks to produce seamless reflections.
- A general disadvantage is that reflection mapping is geometrically accurate only for (rather) small reflective objects located at the center of the surrounding environment sphere/cube.
- As the object size becomes large with respect to the environment sphere/cube, reflections tend to appear in the wrong place on the reflected object.
- If the reflective object is positioned away from its center then the geometric distortion increases.
- Also, reflection mapping works well only if the reflective object is mostly convex. (A non-convex object does not appear as self-reflection in the reflection!)
- Reflection mapping does not scale well when the number of reflective objects is increased.

5 Photorealistic Rendering

- Mathematical Model of Illumination
- Ray Tracing
- Beyond Conventional (Whitted-style) Ray Tracing
- Towards Accurate Global Illumination

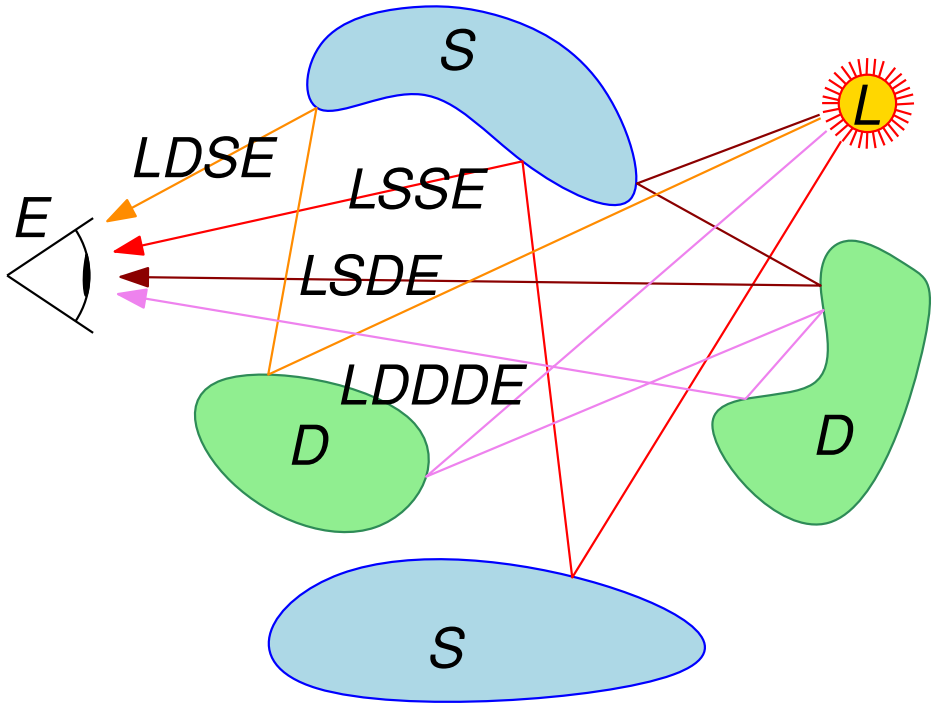
Light Transport

- Real images are formed when photons exit a light source (L), bounce off various specular (S) or diffuse (D) objects in the scene, and eventually reach the eye (E).
- Similarly for transparency and translucency.
- We may have *direct (local) illumination* (e.g., at p) and *indirect (global) illumination* (e.g., at q).



Light Transport: Heckbert's Notation

- Real images are formed when photons exit a light source (L), bounce off various specular (S) or diffuse (D) objects in the scene, and eventually reach the eye (E).
- [Heckbert (1990)]: Each light path can be labeled with a string given by the regular expression $L(D|S)^*E$.
- The more accurately we simulate the physics of all these paths of light transport, the more realistic our images will be.
- Helmholtz reciprocity: The physics is invariant under path reversal!



Rendering Equation

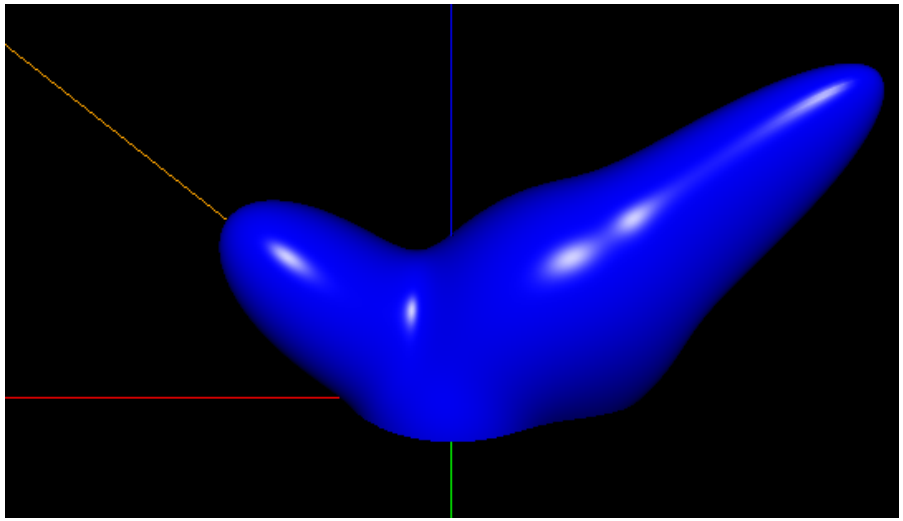
- [Kajiya (1986), Immel et al. (1986)]: Outgoing light $L_o(x, \omega)$ at a specific point x, with direction ω, is equal to light emitted from that point in that direction, $L_e(x, \omega)$, plus an integral $\int_\Omega$ of a reflectance function, $f_r(x, \omega, \omega')$, times the incoming light, $L_i(x, \omega')$, over all directions $\omega' \in \Omega$ determined by the hemisphere at x:

$$L_o(x, \omega) = L_e(x, \omega) + \int_\Omega f_r(x, \omega, \omega') L_i(x, \omega') \, d\omega'$$

- Since $L_i(x, \omega')$ equals $L_o(y, \omega'')$ times some attenuation factor $a(x, y, \omega'')$, for some other point y and direction ω'', the unknown solution function L is on the right-hand side inside the integral and on the left-hand side: “Fredholm equation of the 2nd kind”.
- The attenuation factor depends on distance, visibility and, of course, the participating media.
- Although the rendering equation is fairly general, it does not model all forms of light transport even when a sophisticated reflectance function is used.
- E.g., subsurface scattering (where light enters and exits at different places) and fluorescence (where the wavelengths of absorbed and emitted light differ) are difficult to capture appropriately.

Rendering Equation: Reflectance Function

- The task of the reflectance function is to describe how incoming incident light is reflected.
- It depends on the properties of the surface.
- The reflectance function might be as simple as the functions used by Phong shading, modeling diffuse, glossy and specular reflection.
- To get a better approximation of physics in the real world it is common to use a *Bidirectional Reflectance Distribution Function (BRDF)*.

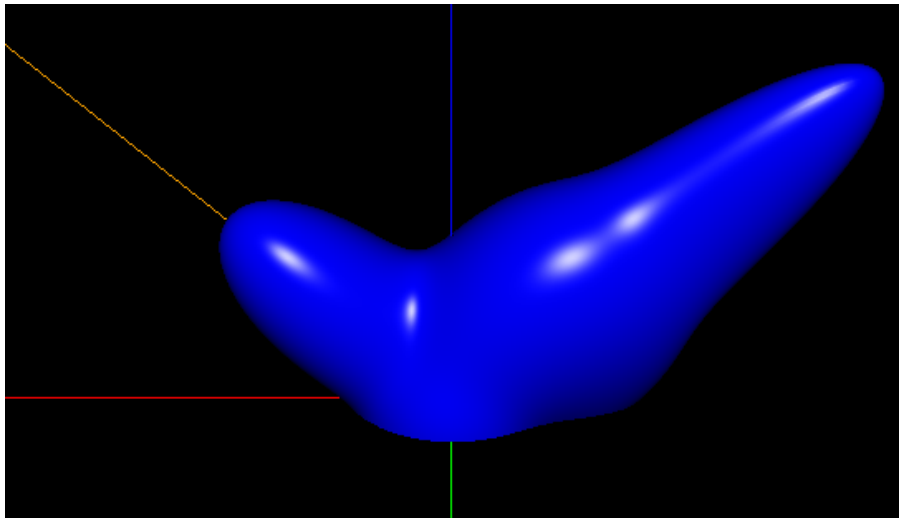


[Image credit: https://brdflab.sourceforge.net/]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Rendering Equation: BRDF

- A BRDF is a distribution function that tells us which amount of light that arrives at point x from one direction ω' is reflected in some other direction ω.
- With some efforts every physically realistic material can be modeled by an appropriate BRDF.
- Transmission can be included in the rendering equation by adding a second integral and a *Bidirectional Transmittance Distribution Function (BTDF)*.
- Some authors prefer to combine BRDF, BTDF (and similar functions) into a *BSDF*, a *Bidirectional Scattering Distribution Function*.



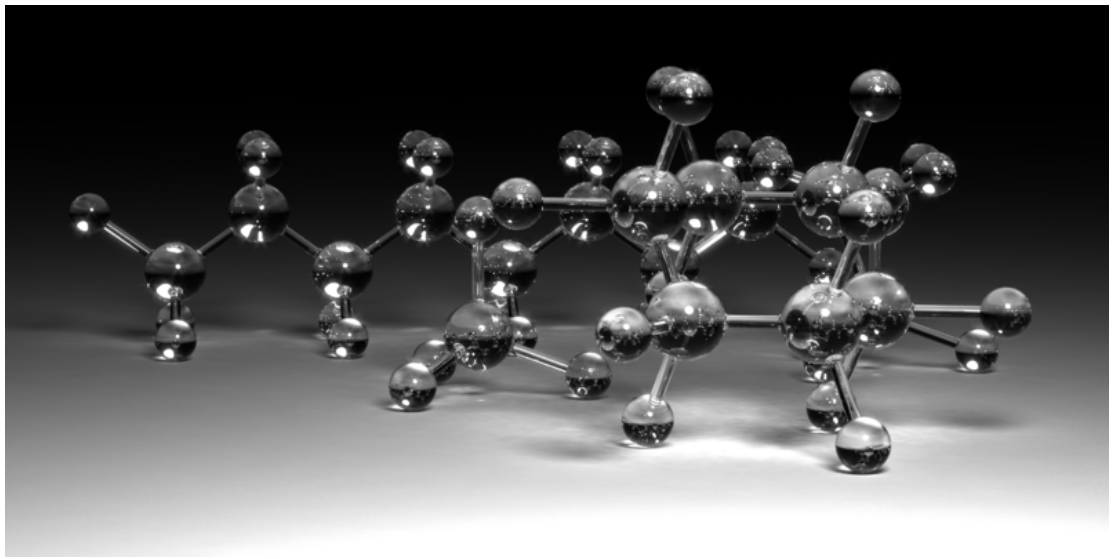
[Image credit: https://brdflab.sourceforge.net/]

Solving the Rendering Equation

- [Whitted (1979)]: Simplify the reflectance function drastically, and focus on specular reflection and light transport of the form $L(D)S^*E$: $\longrightarrow$ *ray tracing*.
- [Goral et al. (1984)]: FEM-based approach to handle LD^*E light transport for Lambertian diffusers: $\longrightarrow$ *radiosity*.
  The solution to the radiosity equation is view-independent.
- Monte-Carlo based techniques can handle $L(D|S)^*E$ light transport.
  - Path tracing,
  - Bidirectional path tracing,
  - Photon mapping,
  - Metropolis light transport,
  - ...

Ray Tracing for Image Synthesis

- Ray tracing can correctly model shadows and multiple specular reflections.
- It can handle refraction and transparent objects.
- It can deal with CSG objects, i.e., with Boolean combinations of objects.
- It can handle non-polygonal objects provided that normal vectors can be computed.
- Recent GPUs provide impressive HW-support for ray tracing!

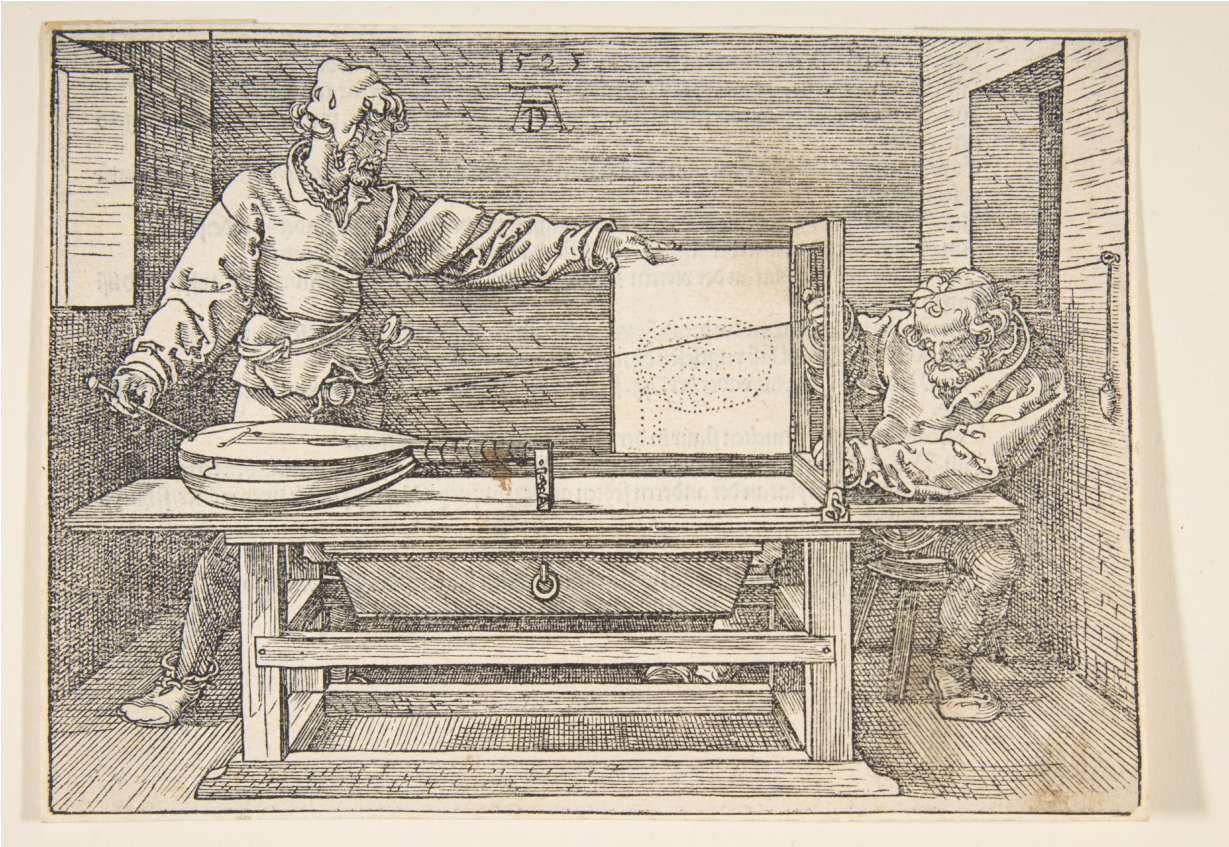


[Image credit: Wikipedia.]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

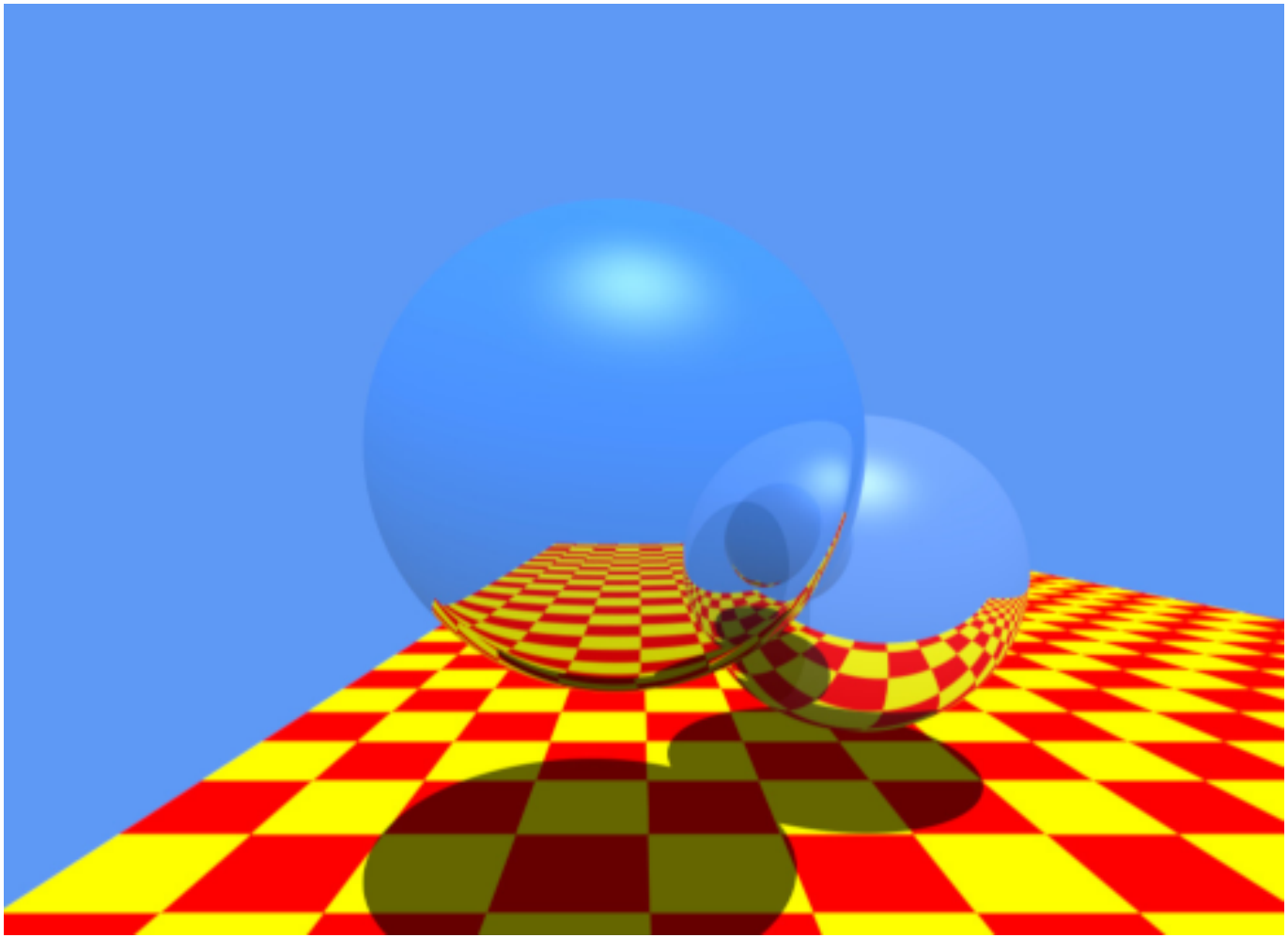
Early Years of Ray Tracing

- [Dürer 1525]: “Underweysung der messung mit dem zirckel und richtscheyt” contains the first description of “Dürer’s door”.



Early Years of Ray Tracing

- [Appel 1968]: Ray casting used to create shaded images.
- [Whitted 1979]: Recursive ray tracing simulates (perfect specular) reflections, refractions and (hard) shadows: $L(D)S^*E$ paths. No “real” use for $\approx$ 20 years.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Photorealistic Images via Ray Tracing

- [Cook&Porter&Carpenter (1984)] in “Distributed Ray-Tracing”:

  Ray-tracing is one of the most elegant techniques in computer graphics. Many phenomena that are difficult or impossible with other techniques are simple with ray tracing, including shadows, reflections, and refracted light.

[Image credit: Wikipedia.]

Light-Based and Eye-Based Ray Tracing

- Consider a particular pixel in the image plane.
- Photons in a three-dimensional scene originate at light sources.
- Photons leave a light source and bounce around the scene.
- Usually, light gets a little dimmer on every bounce.
- Only photons that eventually hit the screen and then pass into the eye (when they are still bright enough) actually contribute to the image.
- Light-based ray tracing (aka “forward ray tracing”) means tracing the path of photons from the light sources via reflections at objects to the eye.

- Efficiency problem: Most of the photons emitted by a light source will never pass into our eye!
- This problem has been partially overcome by clever algorithms, e.g., GPU-based implementations of bidirectional path tracing and Metropolis light transport. Still . . .

Light-Based and Eye-Based Ray Tracing

- The key insight for computational efficiency is to reverse the light transport by tracing photons backward instead of forward: We would like to trace only those photons which certainly contribute to the image.
- The relevant photons are the photons that actually strike the image plane and then pass into the eye.
- Finding the path taken by a photon is easy:
  - We follow rays from the eye to objects to the light sources.
  - If we extend the ray taken by the photon into the world, we can look for the nearest object along the path of the ray.
  - The photon must have come from this object.
- Thus, we follow a ray not forward, from the light source to the eye, but backward, from the eye to the objects and onwards to the light sources.

Ambiguous Terminology!

Note that there is some controversy about the terminology and, in particular, the meaning of backward ray tracing, since early ray tracing was always done from the eye. Hence, it seems best to talk about eye-based versus light-based ray tracing.

Light and Ray Combination

- There are three components that contribute to the color of light at a point p on the surface of an object:
  1. *Local contribution:* Light resulting from direct exposure to light sources.
  2. *Reflected contribution:* Light originating at some (light) source that is reflected towards the eye, based on the physical laws of specular reflection.
  3. *Transmitted contribution:* Light originating at some (light) source that is refracted through the object towards the eye.
- Hence, we divide the rays into four classes:
  1. *Pixel rays* (or eye rays), which carry light directly to the eye through a pixel on the screen.
  2. *Illumination rays* (or *shadow rays*), which carry light from a light source directly to an object surface;
  3. *Reflection rays*, which carry light reflected by an object surface; and
  4. *Transparency rays*, which carry light passing through an object;

Illumination Rays and Shadow Rays

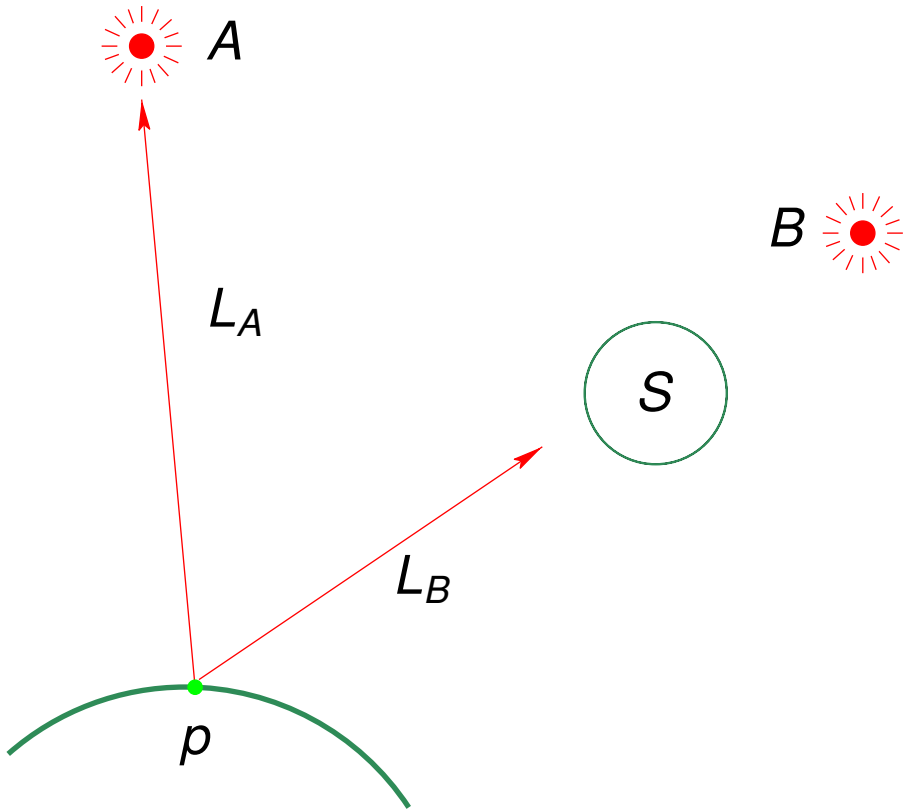
- Imagine yourself at the point p on the surface of an object. *Question:* Is any light coming to you from the light sources?
- To determine the illumination at p, we ask whether photons could possibly travel from each light source to p.
- *Shadow rays* are like any other ray, except that we use them to "feel around" for light. That is why they are often called *shadow feelers*.
- *Illumination ray:* When a shadow ray is able to reach a light source (on a straight path), then we stop thinking of it as a "shadow feeler" and prefer to think of it as an illumination ray which carries light to us from the light source.

Point lights

Note that standard ray tracing deals only with *point light sources*!

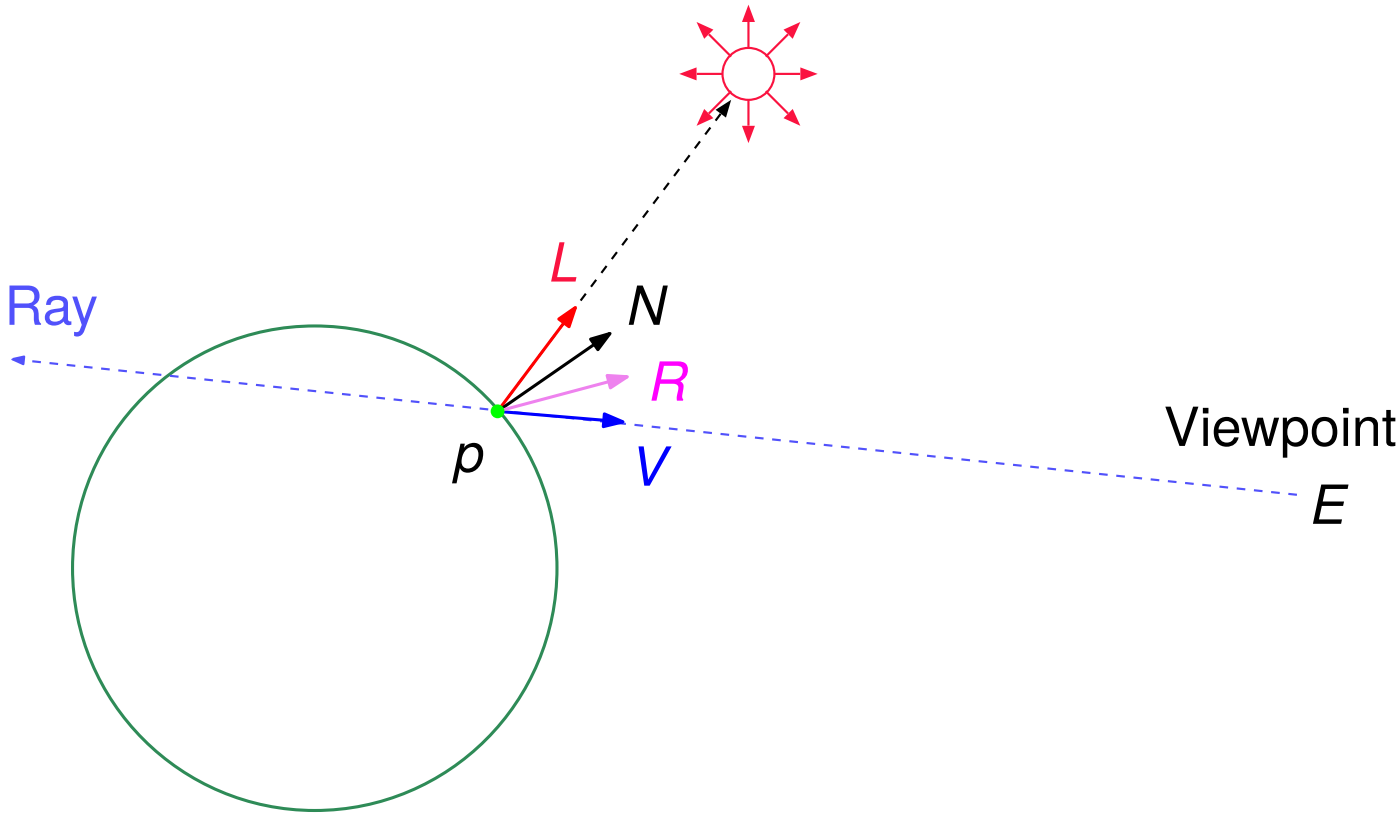
Illumination Rays and Shadow Feelers

- Which light sources are visible from p in the example setting below?
- We answer this by sending the shadow ray L_A towards light source A. It arrives at A, so L_A is actually an illumination ray from p to A.
- On the other hand, ray L_B is blocked from light source B by sphere S. Thus, no light arrives at p from B.



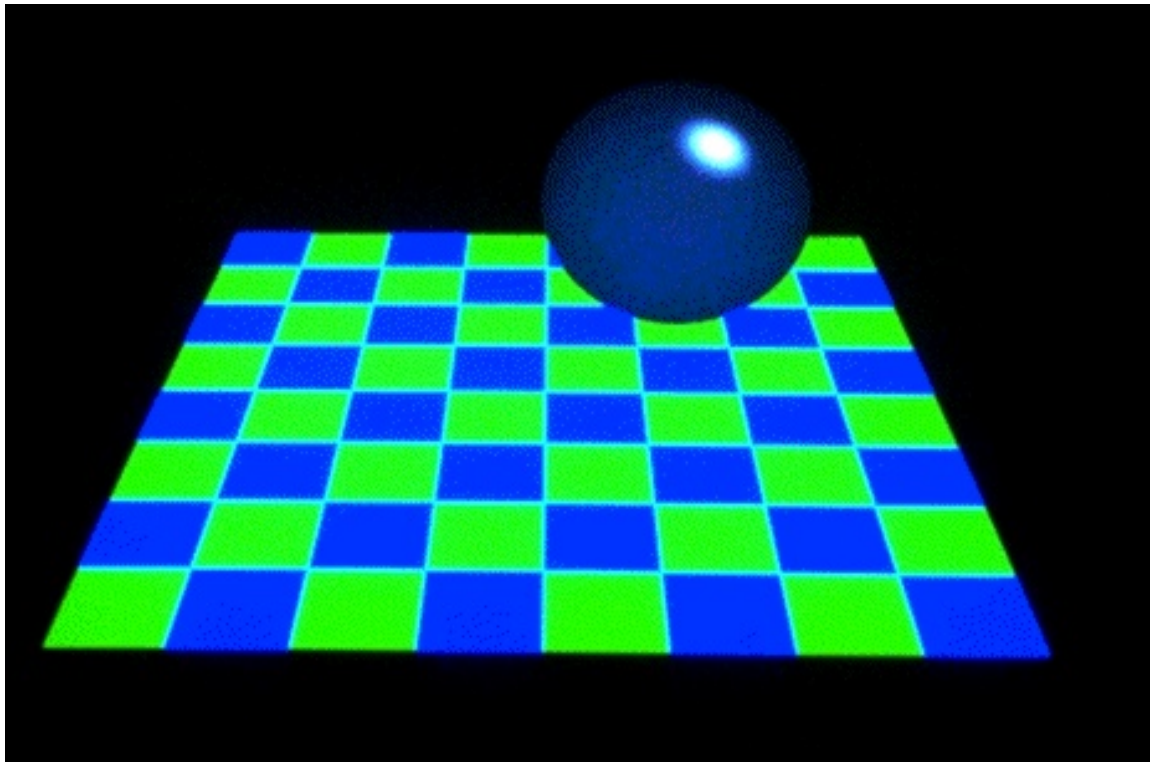
Illumination Rays and Shadow Feelers

- If the ray reaches the light source, then we apply a suitable illumination model for computing the contribution of the light source to the light reflected in the direction of the eye.
- E.g., we may use Phong's rule to compute a (partial) specular highlight for specular surfaces, or use a diffuse reflection model for mostly dull surfaces.



Ray Casting

- *Ray casting*: Every ray is stopped at the first object intersected. (The scene consists of a checkerboard and of a reflective sphere.)
- No shadow feelers.
- Ray casting solves the hidden-surface problem: The object first encountered by a ray is the visible object.

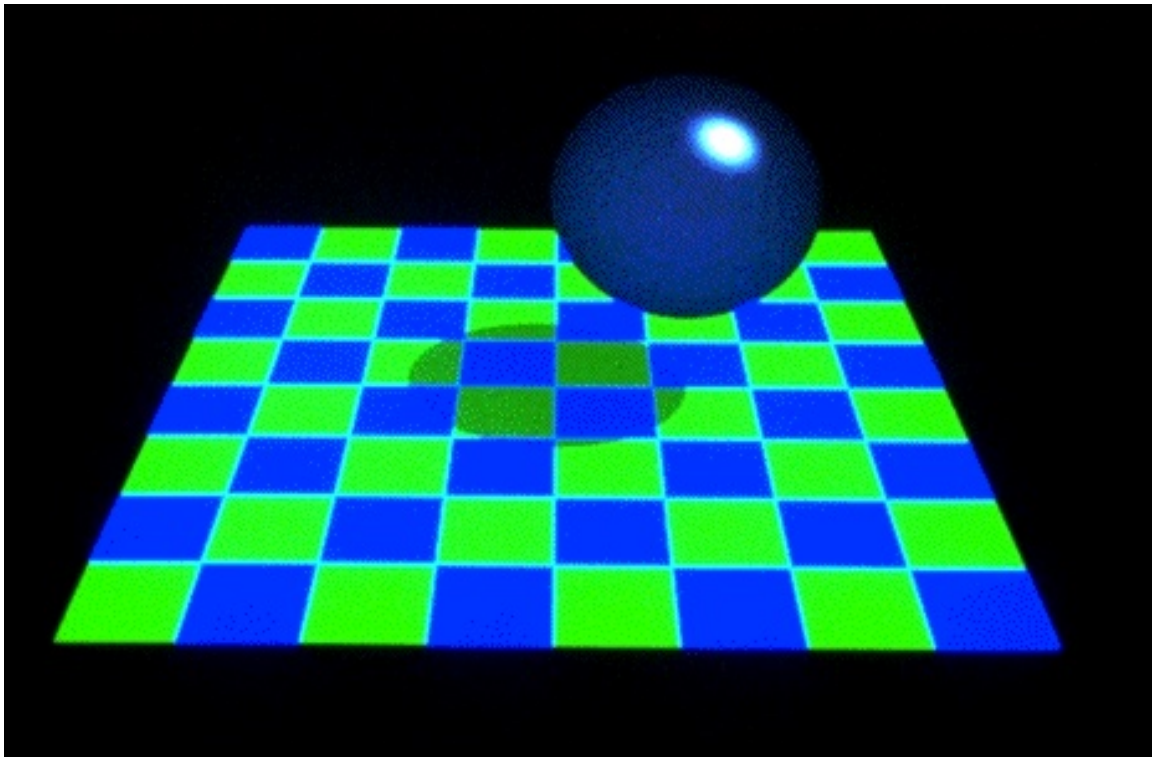


[Image credit: SIGGRAPH Educator's Slide Set (Slide #10).]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Ray Casting with Shadow/Illumination Rays

- All eye rays were stopped at their first intersection with the scene, and illumination rays were considered.

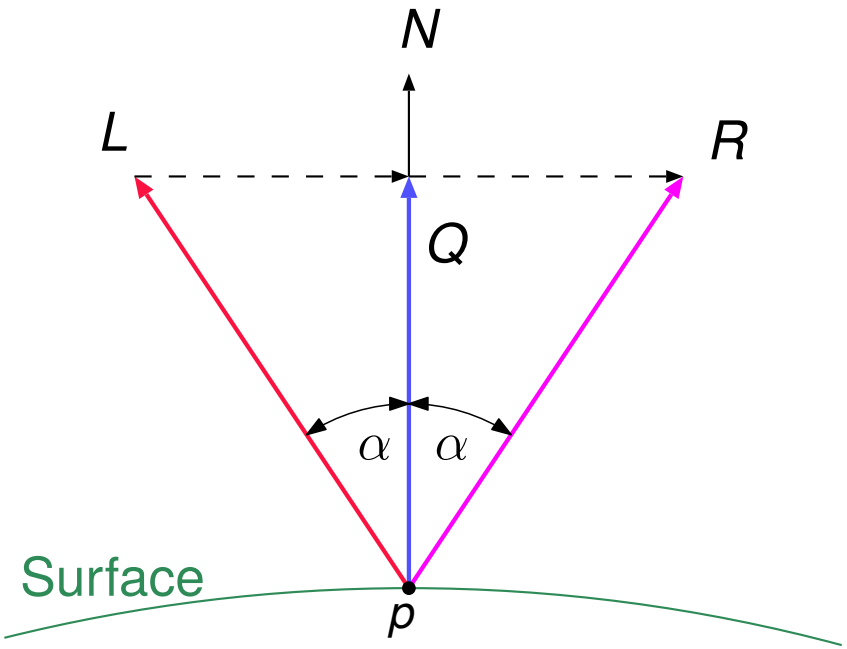


[Image credit: SIGGRAPH Educator's Slide Set (Slide #16).]

Reflection Rays

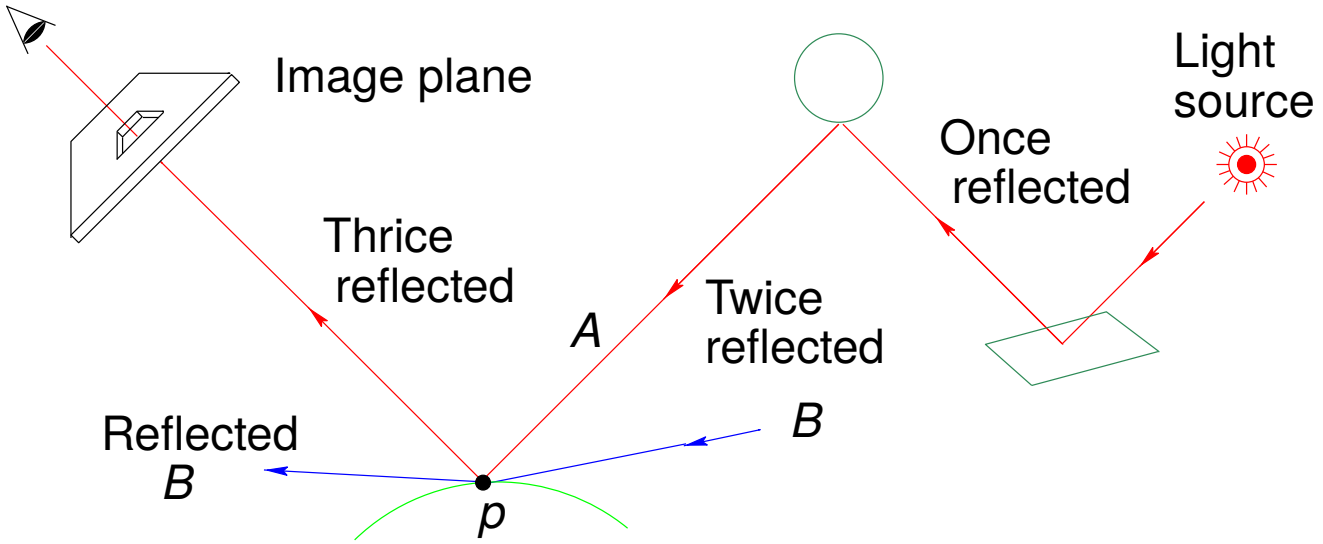
- When determining the illumination at a point p, recall that we originally found that point by following a ray to the object: *incident ray* with direction vector $-L$.
- Our goal is to find the color of the light leaving the object in the direction opposite to the incident ray: *reflection ray* with direction vector R.
- Note that *perfect specular reflection* is assumed.
- In this case the angle of reflection is equal to the angle of incidence, and L, N and R lie in a plane.

$$
\begin{aligned}
Q &= (L \cdot N) \cdot N \qquad \text{where } ||N|| = 1 \\
R &= L + 2(Q - L) \\
&= L + 2((L \cdot N) \cdot N - L) \\
&= 2(L \cdot N) \cdot N - L
\end{aligned}
$$



Reflection Rays

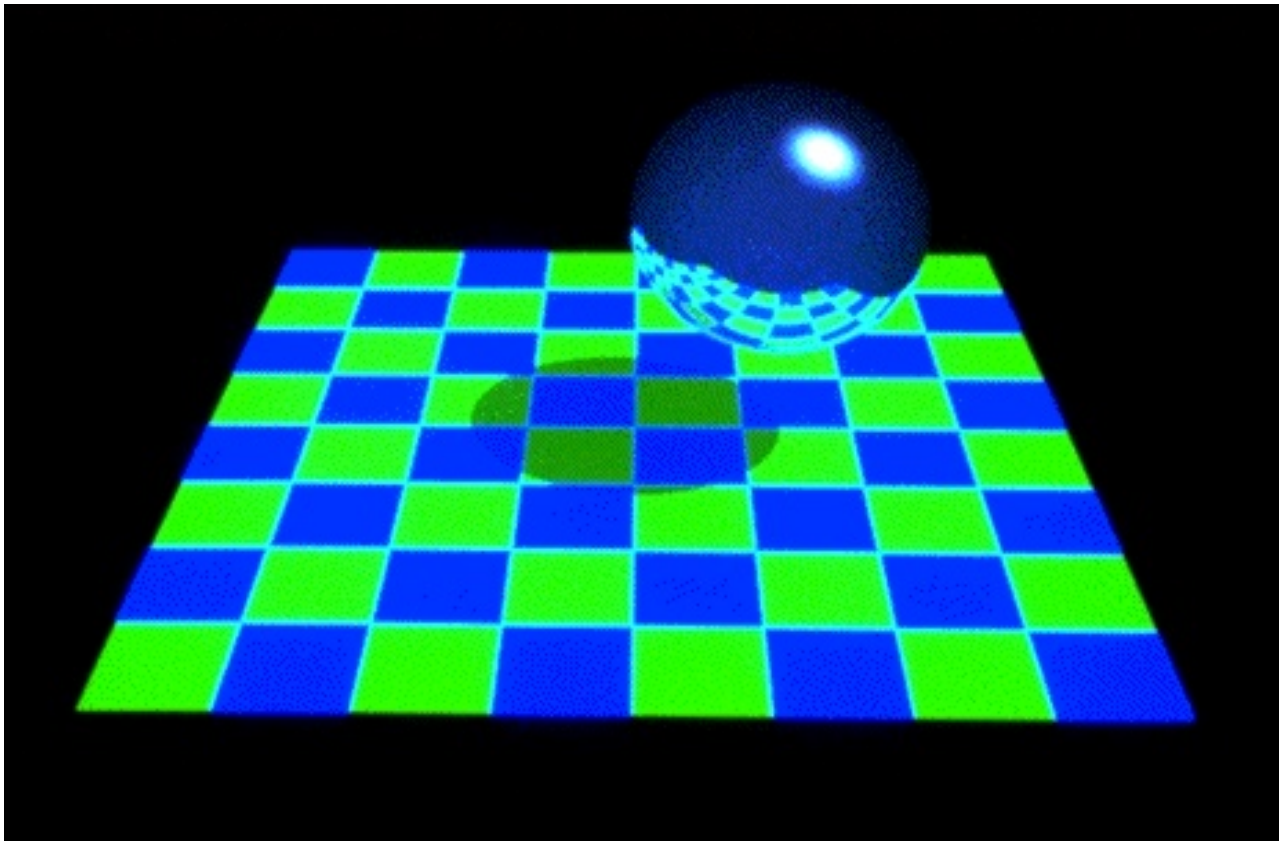
- At point p (of a reflective surface) we want to know the color of the light coming in on ray A, since that light is then bounced into the eye. Other rays passing through P, such as B, do not have any impact on what is reflected towards the eye.



- For finding the color of a reflection ray, we follow it backwards to determine the object where it originated.
- The color of the light leaving that object along the line of the reflected ray is the color of the reflected ray.
- When we know the reflected ray's color, we add it to any other light leaving the original surface struck by the incident ray.

Reflection Rays: Sample

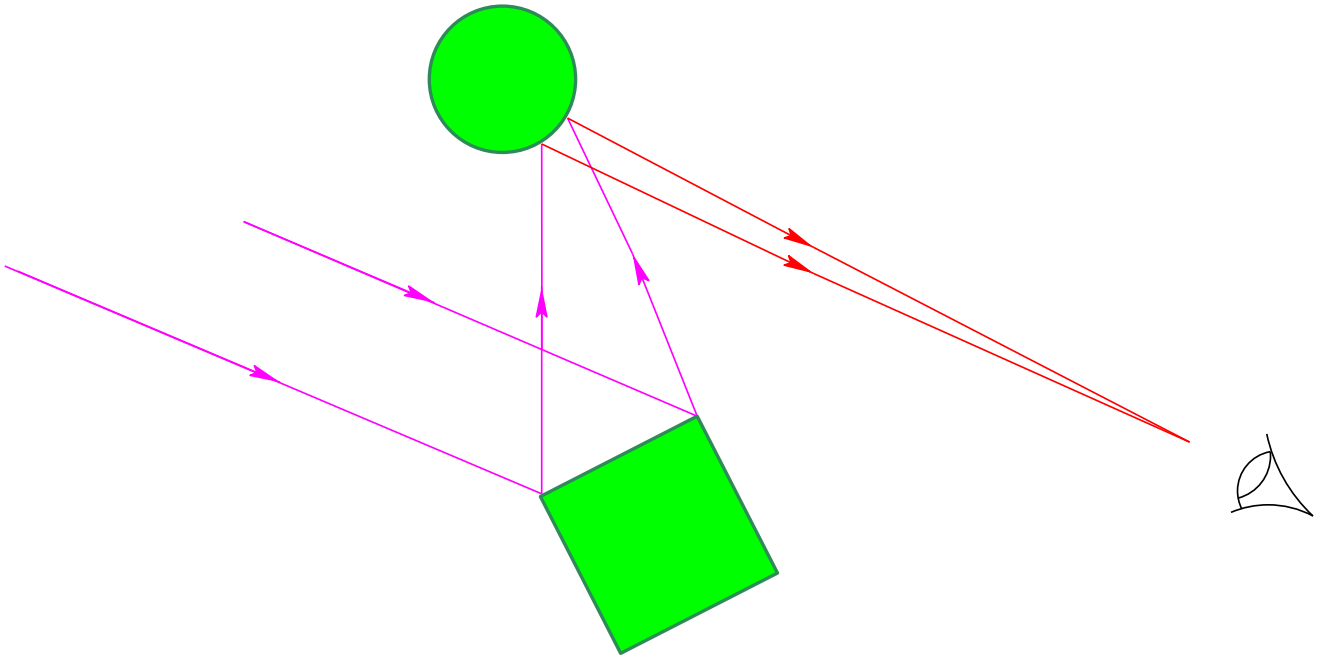
- The scene consists of a dull checkerboard and of a reflective sphere. Illumination rays and first-level reflection rays were considered.



[Image credit: SIGGRAPH Educator's Slide Set (Slide #19).]

Reflection Rays and Back-Face Culling

- It is important to observe that ”back” surfaces of an object may be visible in a ray-traced scene.
- Thus, back-face culling is not applicable when ray tracing is applied!

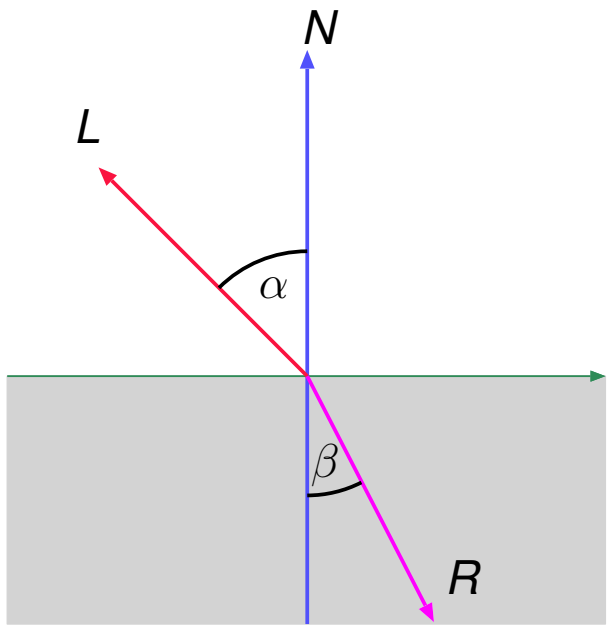


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

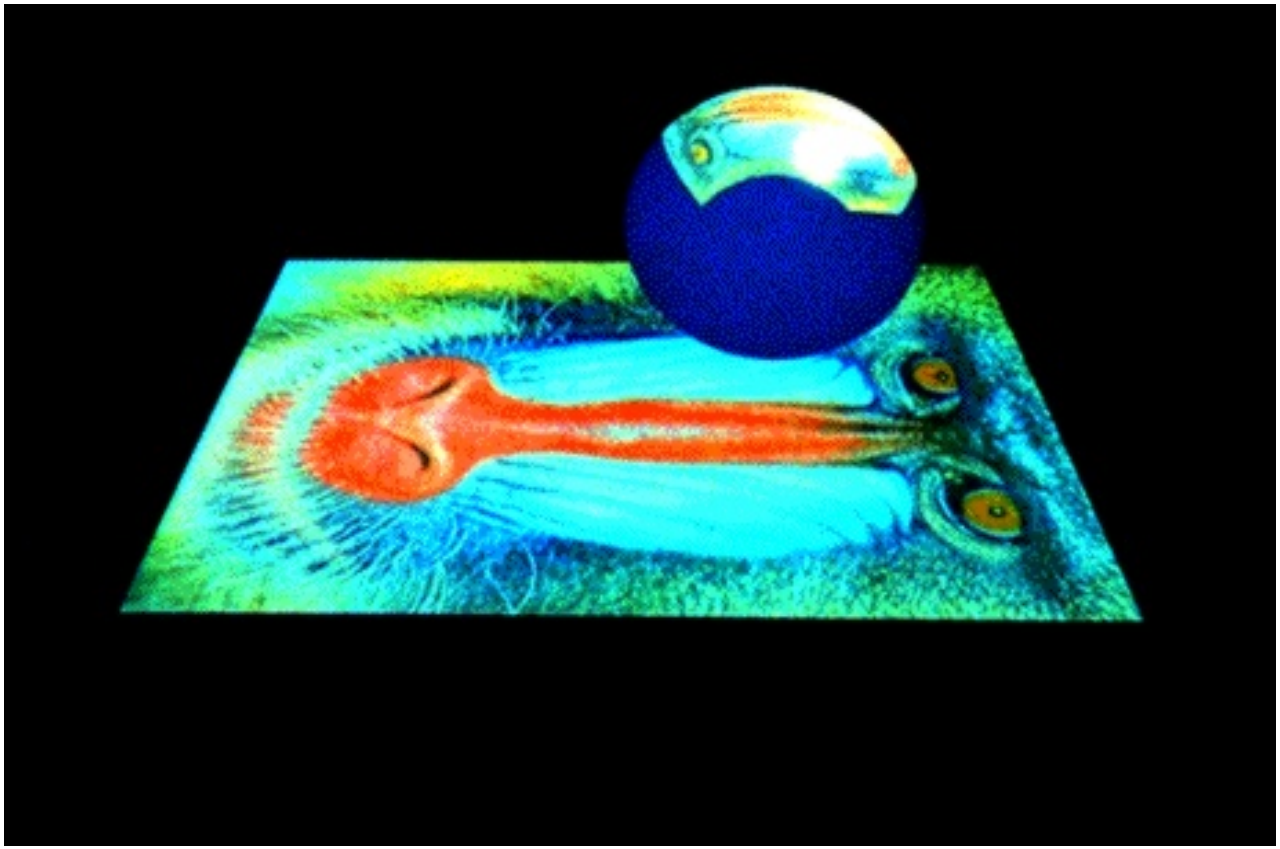
Transparency Rays

- There is a single direction from which light can be perfectly transmitted into the direction of the incident ray.
- The ray created to determine the color of this light is called the *transmitted ray*, or *transparency ray*.
- One has to pay attention to the *refraction* of light as it passes from one medium to another: Each material has an *index of refraction*, η, given by the ratio of the speed of light in vacuum and the speed of light in the material.
- In general, the index of refraction depends on the wavelength of the light — this causes *dispersion* in a prism!
- Snell's law tells us that the transmitted ray remains within the plane of incidence, and that the sine of the angle of refraction is directly proportional to the sine of the angle of incidence:

$$\frac{\sin\alpha}{\sin\beta} = \frac{\eta_b}{\eta_a}.$$

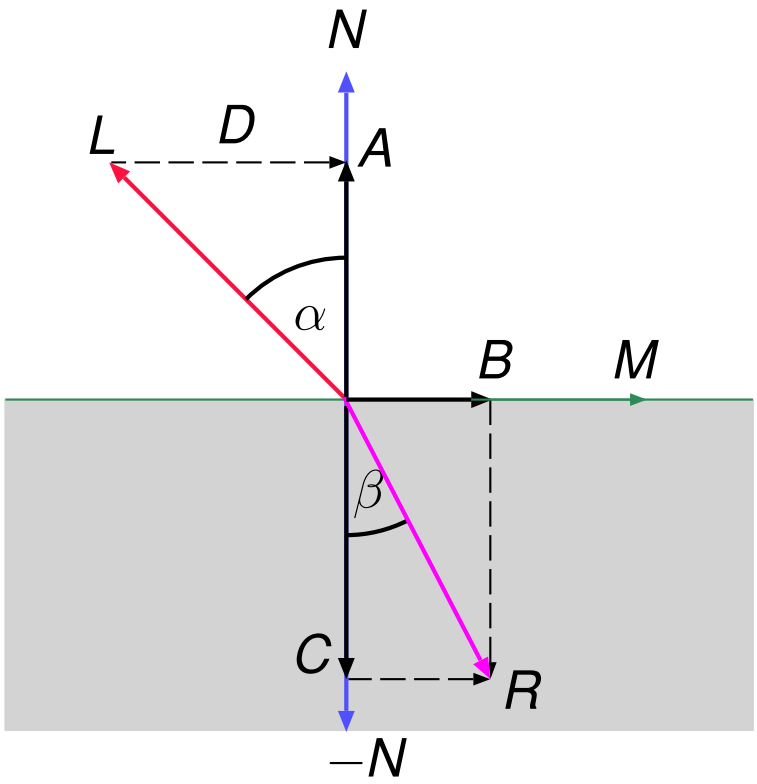


Transparency Rays: Sample Image



[Image credit: SIGGRAPH Educator's Slide Set (Slide #22).]

Transparency Rays: Math



$$||L|| = ||N|| = 1$$

$$A = N \cdot \cos\alpha = N \cdot (L \cdot N)$$

$$D = A - L$$

$$M = D / \sin\alpha \qquad \text{(and we have } ||M|| = 1\text{)}$$

$$B = M \cdot \sin\beta \qquad \text{with } \sin\beta = \frac{\eta_a}{\eta_b} \sin\alpha$$

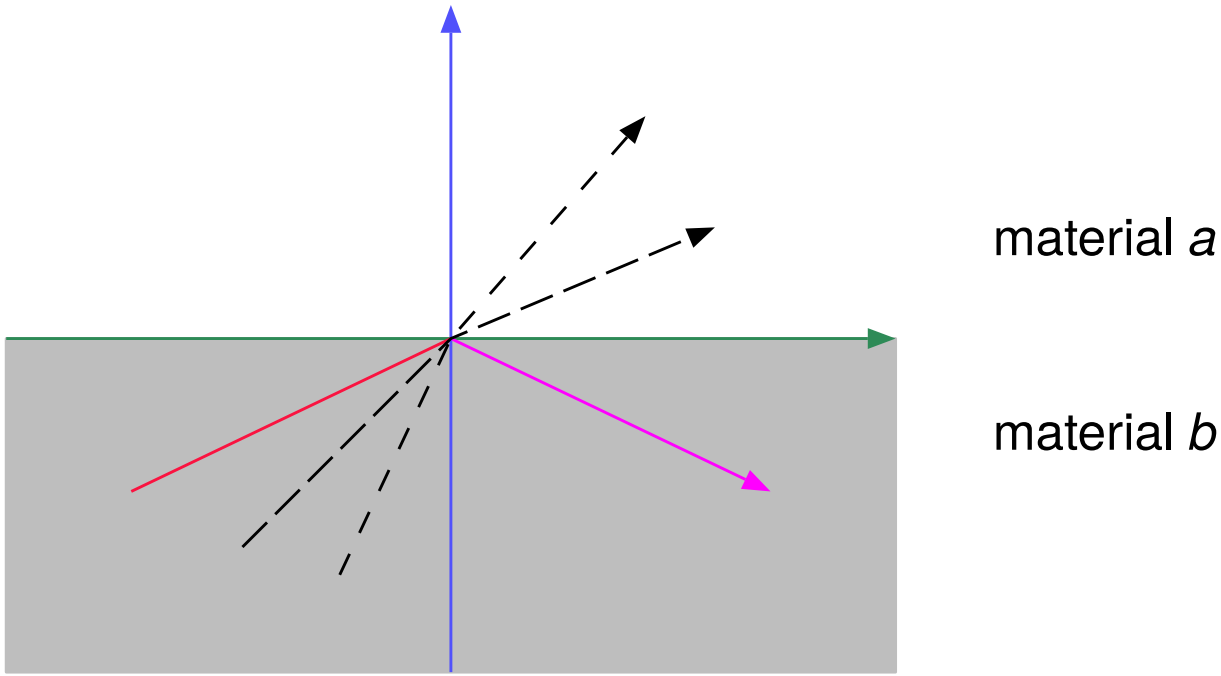
$$C = -N \cdot \cos\beta \qquad \text{with } \cos\beta = \sqrt{1 - \sin^2\beta}$$

$$R = B + C$$

Summarizing, $$R = N\left(\frac{\eta_a}{\eta_b}(L \cdot N) - \sqrt{1 - \left(\frac{\eta_a}{\eta_b}\right)^2 (1 - (L \cdot N)^2)}\right) - \frac{\eta_a}{\eta_b} L.$$

Transparency Rays: Total Internal Reflection

- When a light ray hits a boundary between a dense region to a less dense region, the square root in the formula for R may not exist.
- If this happens, the light ray is *reflected internally*, and we compute reflection instead of refraction.

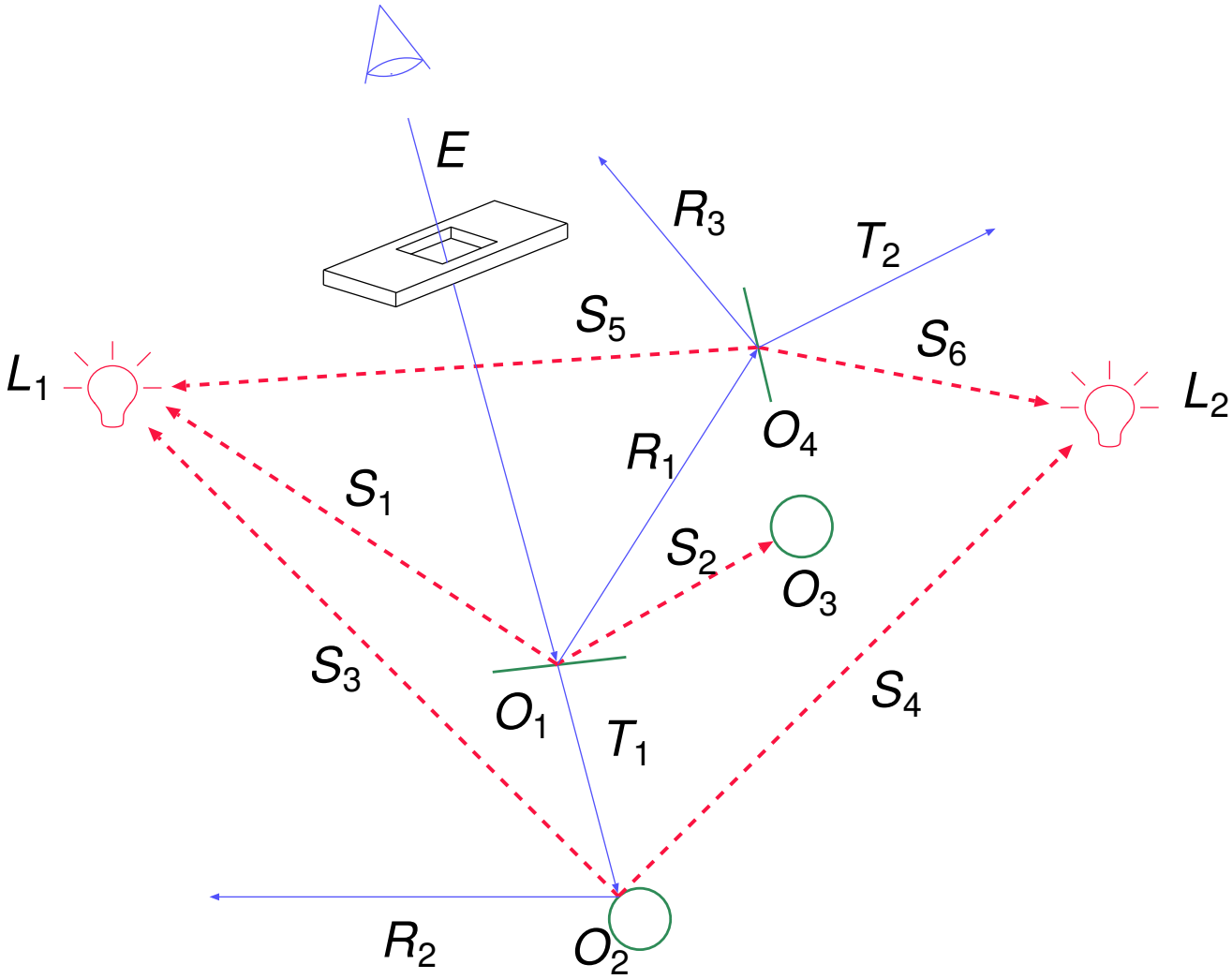


Recursive Ray Tracing

- The colors of the reflected and the transmitted light were found by finding the objects from which they originated.
- What was the color of light leaving those objects? It was a combination of the light rays reaching them, which can be found with the same analysis.
- This suggests a recursive algorithm and we get Whitted-style *(recursive) ray tracing*, as opposed to mere ray casting:
  - First we send an eye ray through every pixel of the screen.
  - This ray is stopped at the first intersection with any object.
  - From this intersection point we send shadow feelers to the light sources of the scene.
  - In addition, we send a reflection ray and a transparency ray.
  - For objects hit by these two rays we apply this scheme recursively.
  - No recursive rays are spawned if a dull/opaque surface is hit. (That is, recursion ends in such a case.)
  - With every reflection, the brightness of light is reduced and after a number of reflections the contribution to our top object's brightness and color is not significant anymore.
  - Thus, we may terminate the recursion after a certain number of recursive calls, e.g., 10 to 15.

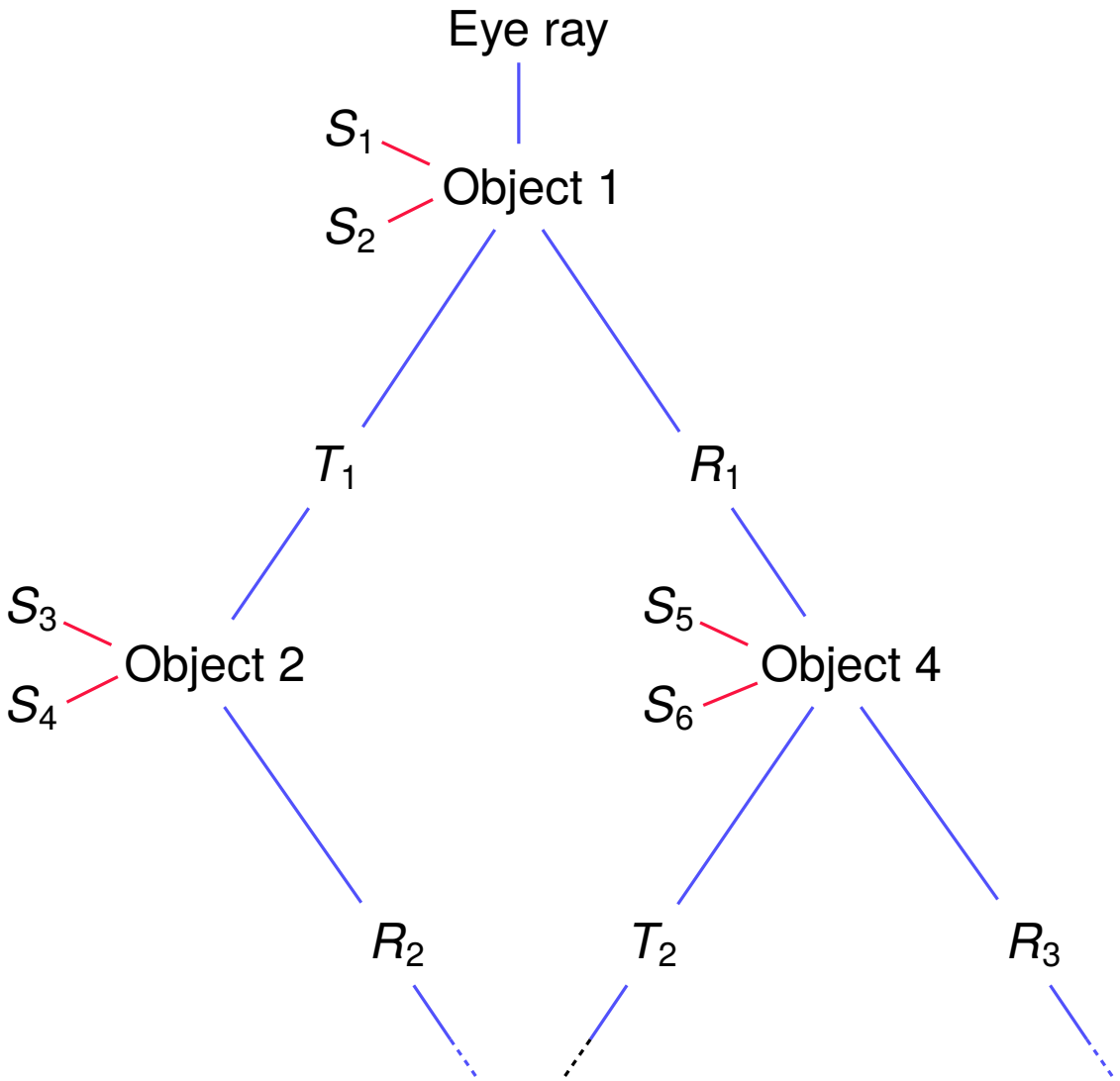
Recursive Ray Tracing

- An eye ray E propagated through a scene. Many of the intersections spawn reflected, transmitted and shadow rays.

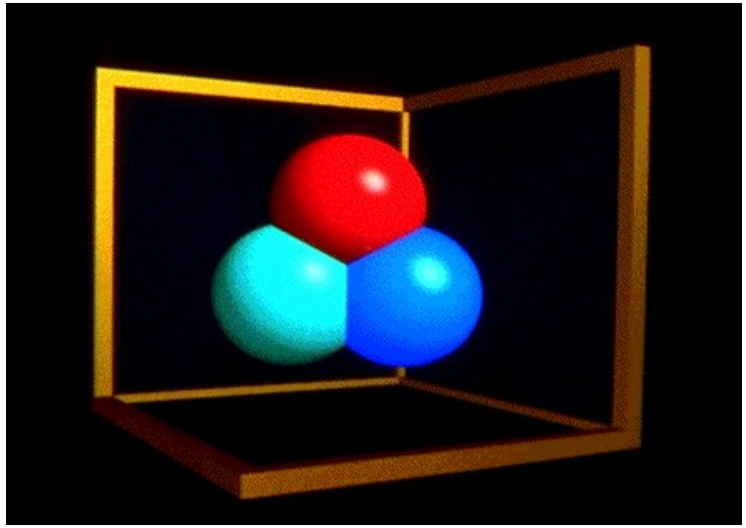


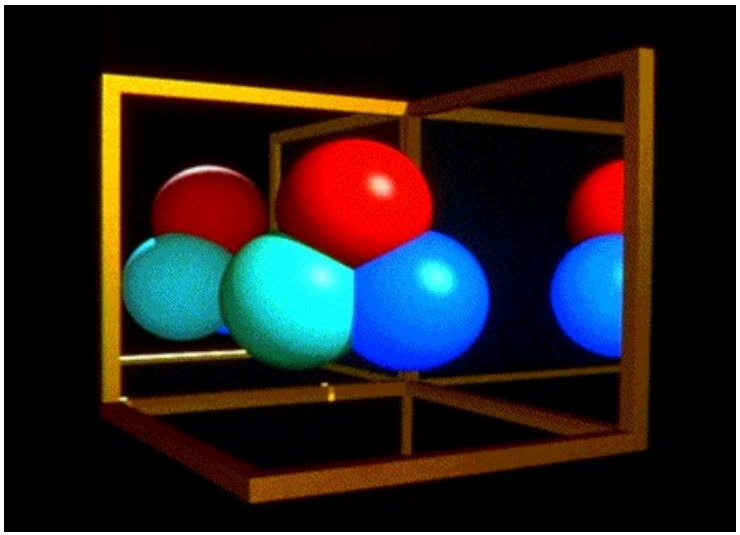
Recursive Ray Tracing: Ray Tree

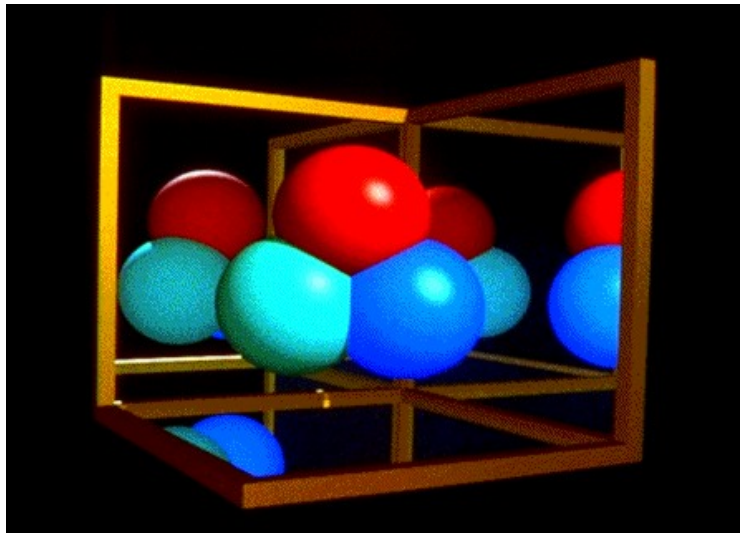
- Recursive ray tracing generates a ray tree.



Recursive Ray Tracing: Samples



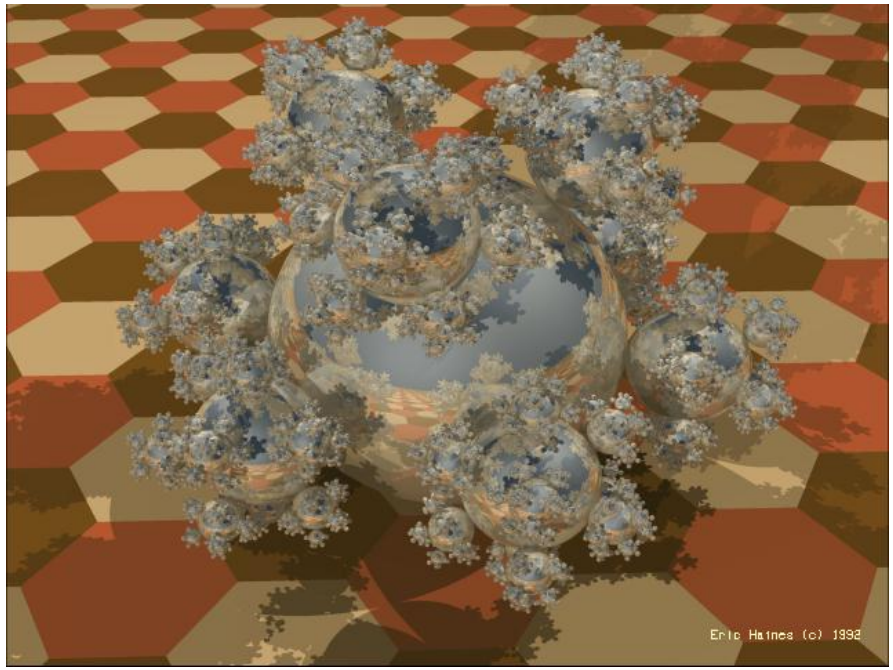




[Image credit: SIGGRAPH Educator's Slide Set (Slides #24–26).]

Recursive Ray Tracing: Samples

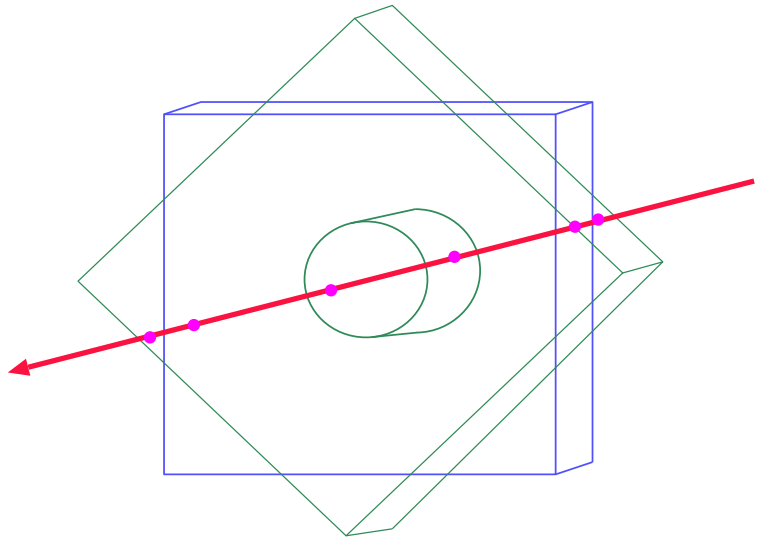
- The Sphereflake consists of 7381 spheres; the floor's texture was modeled by a procedural function.



[Image credit: Eric Haines]

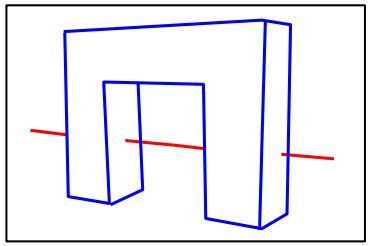
Ray Tracing CSG Objects

- Basic idea:
  - Shoot a ray toward each of the primitives and compute *hit lists* of linked lists where the ray enters and exits each primitive.
  - Use the hit lists to compute where the ray enters and exits the combined solid and adjust the surface normals properly.
  - At this point a shading calculation is performed, and, if necessary, secondary rays have to be generated.
  - The hit lists for the left and right children of a node are combined by using the so-called *Roth Diagram*.
- CSG trees can be pruned during ray tracing:
  - If the left or right subtree of an intersection operation returns an empty list, then the other subtree need not be processed.
  - If the left subtree of a difference operation returns an empty list, then the right subtree need not be processed.

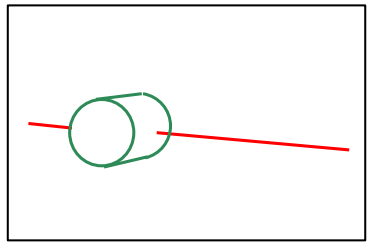


UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

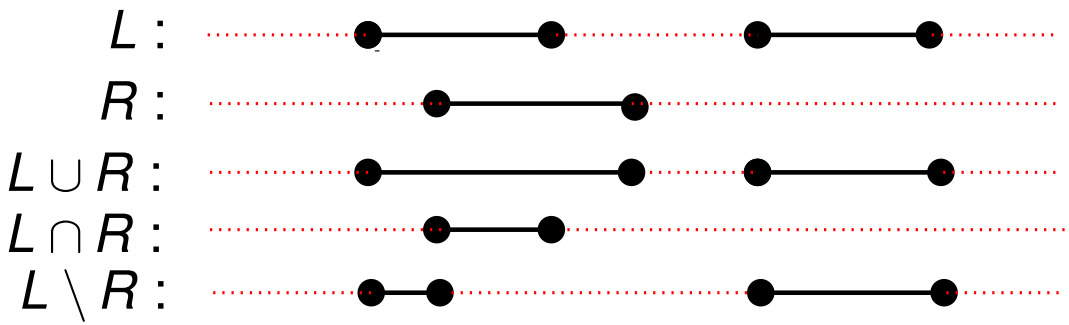
Ray Tracing CSG Objects: Hit Lists and Roth Diagram

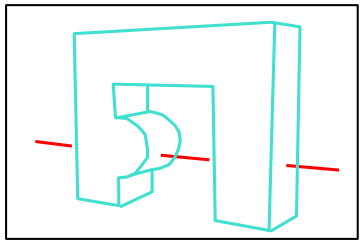


Left

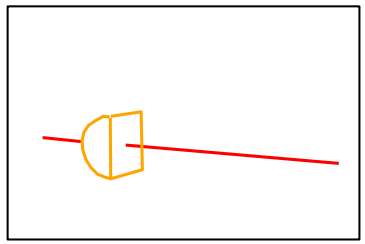


Right

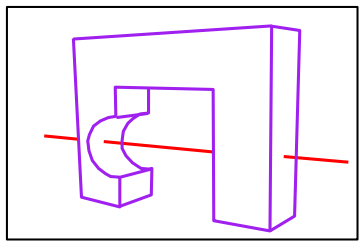




$L \cup R$



$L \cap R$



$L \setminus R$

Efficiency Considerations

- One of the greatest challenges of ray tracing is efficient execution. Efficiency has therefore been the focus of much research from the early days on.
- For accelerating the process of ray tracing, there are three very distinct strategies to consider:
  1. Reducing the *total number* of rays intersected with the environment: Fewer rays.
  2. Reducing the *average cost* of intersecting a ray with the environment: Faster intersection tests and fewer intersection tests.
  3. Replacing individual rays with a more *general entity*: *Generalized Rays*. This includes approaches like pencil tracing, cone tracing (with both circular and polygonal cross-sections), and beam tracing. The main idea of all these approaches is to trace many rays simultaneously.

Fewer Rays Due to Intensity Attenuation

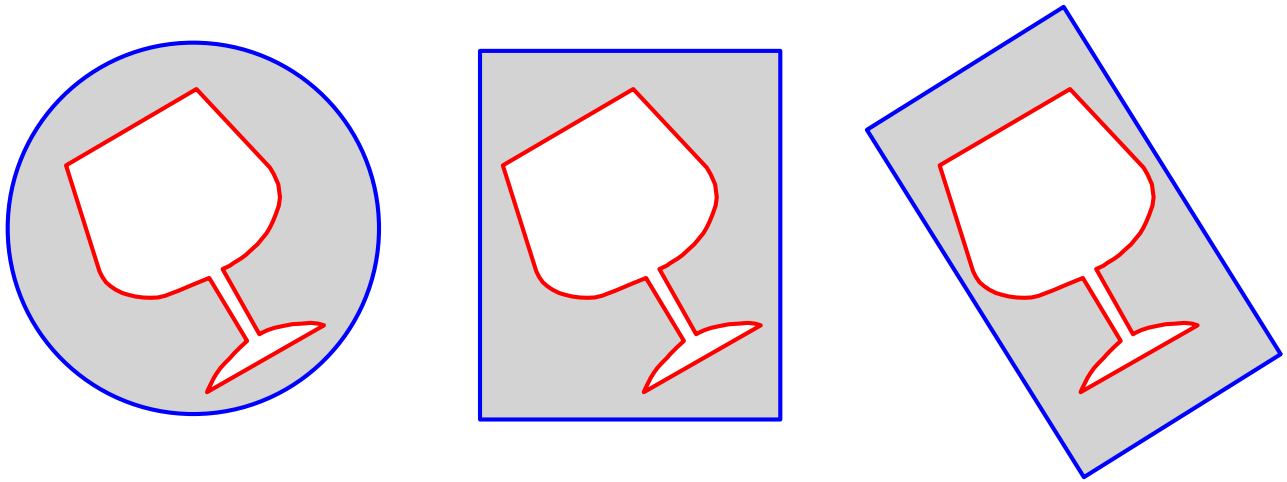
- In order to ensure that recursion ends, in a naive ray tracer it is necessary to define a maximum depth — e.g., 10 to 15 — to which rays are traced recursively.
- For a particular scene this maximum depth is preset to a value which will depend on the nature of the scene. (Highly reflective surfaces and transparent objects need greater maximum depth than scenes with lots of dull/opaque objects.)
- [Hall&Greenberg (1983)]: The percentage of highly transparent or reflective surfaces of a scene is, in general, small and it is thus inefficient to trace every ray to the maximum depth.
- In particular, light is attenuated in various ways as it passes through a scene. E.g., a ray that is reflected at a surface is attenuated by the global specular reflection coefficient of this surface.
- A ray that is examined as a result of ray tracing will make a contribution to the eye ray that is attenuated by several of these coefficients.
- If the product of these coefficients falls below some threshold then there might be little to gain by tracing back further than the actual ray: the recursion is stopped.
- Limiting the recursion based on the attenuation factors accumulated is called *adaptive-tree depth control*.
- However, there are theoretical arguments (and practical examples) that show that adaptive tree-depth control can be arbitrarily wrong.

Faster Intersection Tests

- The main goal is to reduce the average cost of computing an intersection.
- We distinguish between the following two sub-goals:
  - *Faster Tests for Ray/Object Intersections*: The number of intersection tests is not reduced, but conservative pre-tests (e.g., by means of bounding volumes) are employed in order to reduce the average cost of an intersection test.
    Note that such an approach will not be of much help if the sheer magnitude of the number of intersection tests constitutes a problem: In terms of the O-notation, we would only change the multiplicative constants but could not decrease the order!
  - *Fewer Tests for Ray/Object Intersections*: Bounding volume trees and similar concepts are employed in order to reduce the average number of intersection tests.

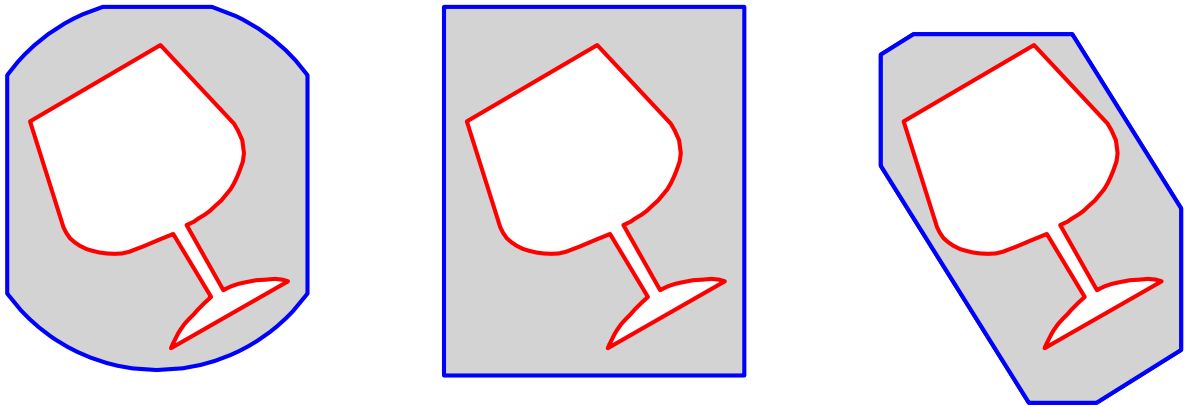
Faster Intersection Tests: Bounding Volumes

- The most fundamental and ubiquitous tool for ray tracing acceleration is the *bounding volume*: A bounding volume is a volume which contains a given object and permits a simpler ray-intersection check than the object.
- Common bounding volumes are spheres, axis-aligned bounding boxes (AABBs), oriented bounding boxes (OBBs), discrete-orientation polytopes (k-dops, plane-sets), convex hulls, ...
- Only if a ray intersects the bounding volume then the object itself needs to be checked for intersection.
- Normally, the use of bounding volumes results in a significant net gain in speed.



Faster Intersection Tests: Bounding Volumes

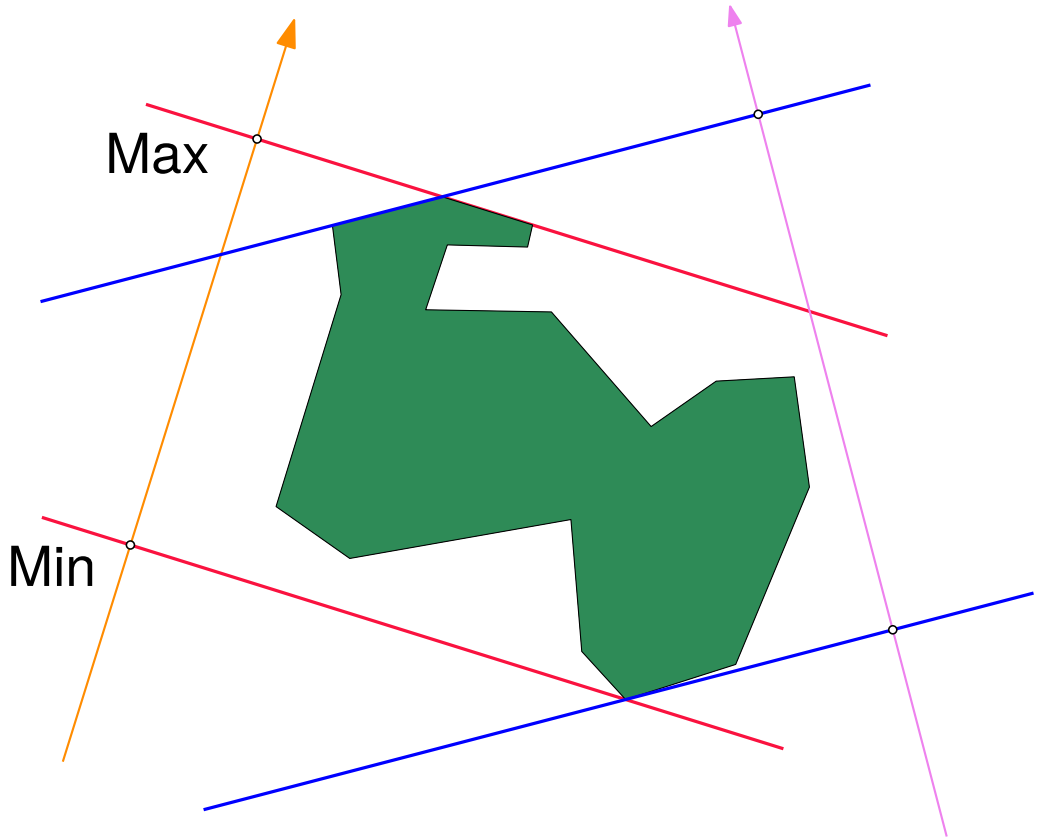
- Virtually all bounding volumes form convex objects: This is a desirable fact because it guarantees that any ray will intersect the boundary of the object in at most two points.
- Intersections of multiple bounding volumes can be used to obtain a better fit.
- Each approach requires a different ray-intersection algorithm for best performance.
- Little agreement on what is best . . .



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Faster Intersection Tests: Bounding Volumes

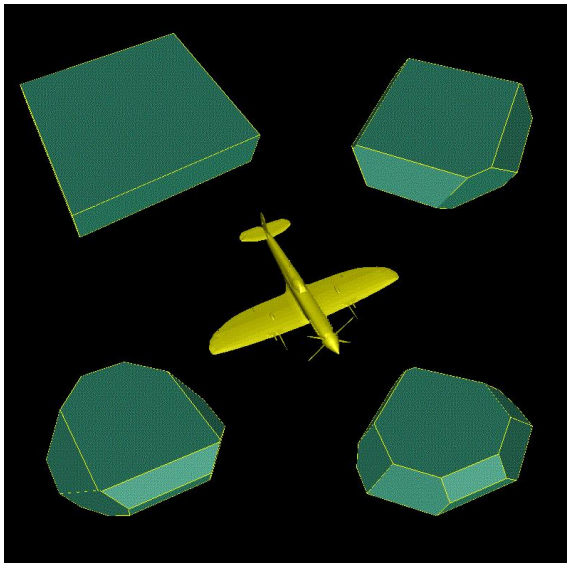
- A *plane-set normal* defines a family of parallel planes orthogonal to it. Two values associated with a plane-set normal select two of these planes and define a slab.
- The intersection of several such slabs form a parallelepiped bounding volume.
- For normals chosen among a given set of $\frac{k}{2}$ normals, the resulting bounding volume is also known as a *discrete-orientation polytope*, or *k-dop* for short.

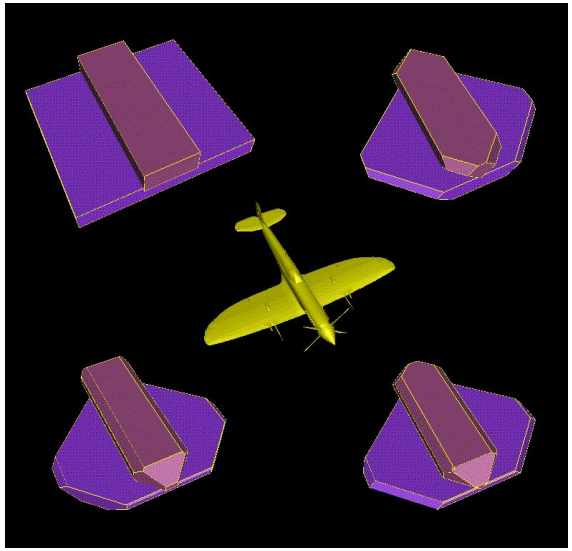


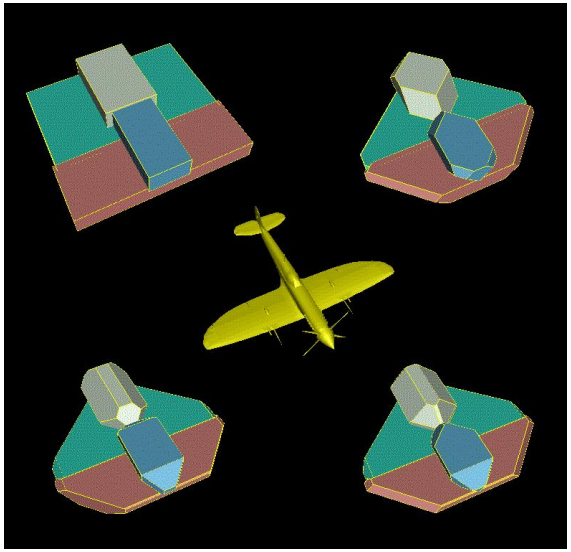
Faster Intersection Tests: Bounding Volume Trees

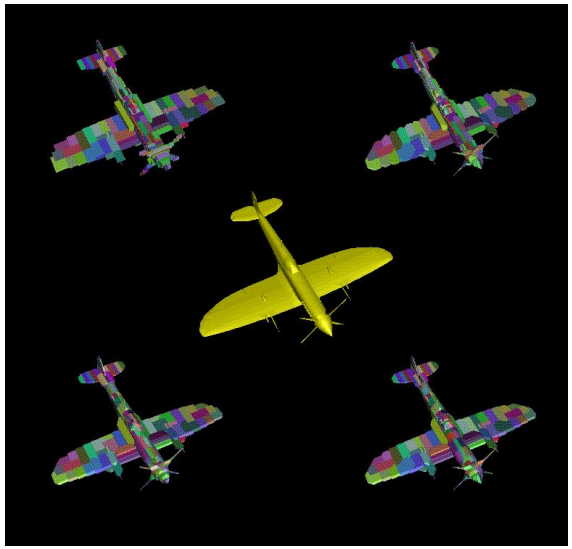
- By enclosing several bounding volumes within one larger bounding volume it may be possible to eliminate many objects from consideration with a single intersection check: If a ray does not intersect the parent volume then there is no need to test it against the bounding volumes or objects contained within.
- A hierarchy is formed by repeated application of this principle.
- Since a hierarchy of bounding volumes forms a tree, the resulting structures are commonly called *bounding-volume trees* (BVTs).
- BVTs are also widely employed in other applications, e.g., for collision detection.

Faster Intersection Tests: Bounding Volume Trees







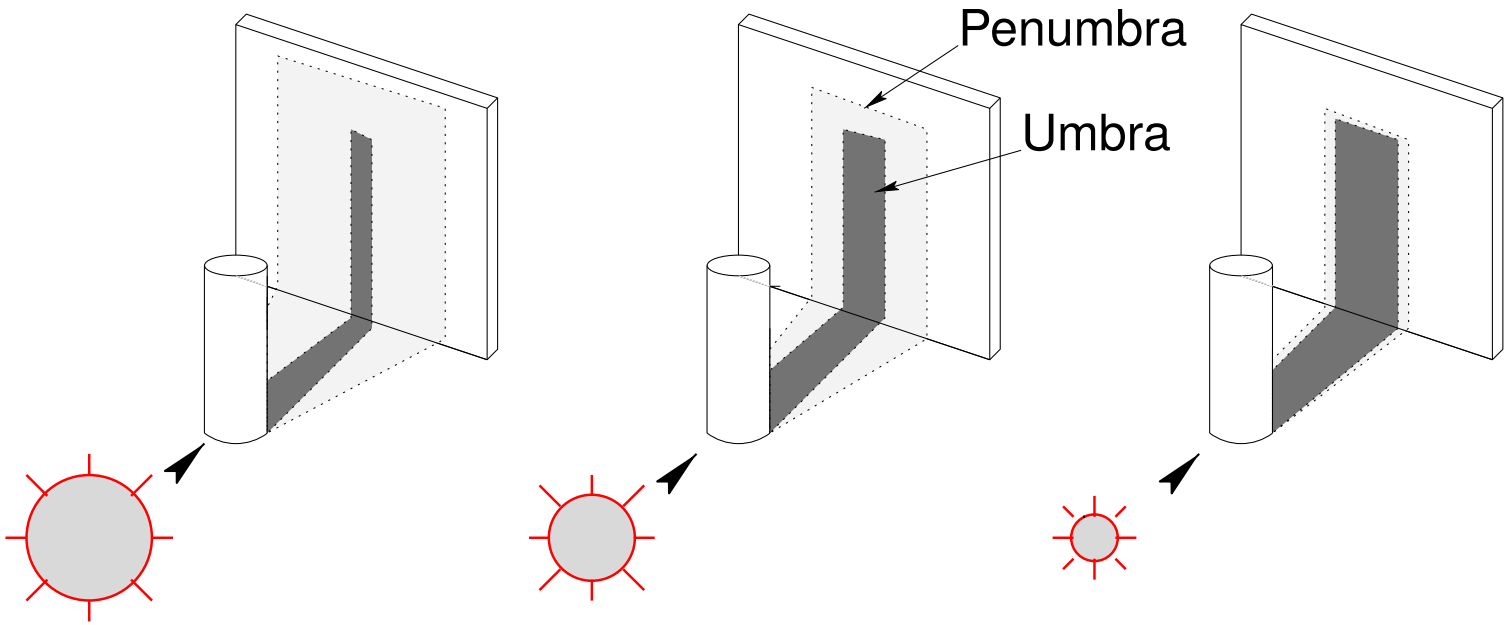


Blurring Effects: Motivation

- In the real world many things are not so discrete or sharp as we assumed so far:
  - light sources are no points,
  - mirrors are not perfect,
  - mirrors are not even,
  - times of exposure are not zero, and
  - aperture and focal length of an optical system are badly modelled by a camera “hole” that is infinitely small.
- These issues cause blurring effects, i.e., some amount of fuzziness that makes photographs look natural in detail.
- Since our visual system is accustomed to look for these visual cues, we tend to perceive pictures as unreal if most or even all of these blurring effects are missing.

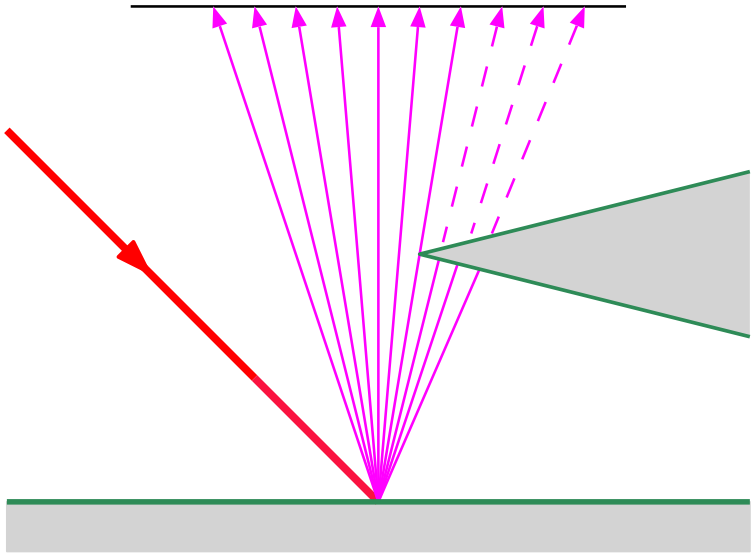
Blurring Effects: Penumbra and Soft Shadows

- Real light sources are never pure point lights. They do not produce sharp shadows, but instead a penumbra region occurs:
  - That part of a light source's shadow that is totally blocked from the light source is the shadow's *umbra*.
  - That part of the shadow that is only partially shielded from the source is the shadow's *penumbra*.



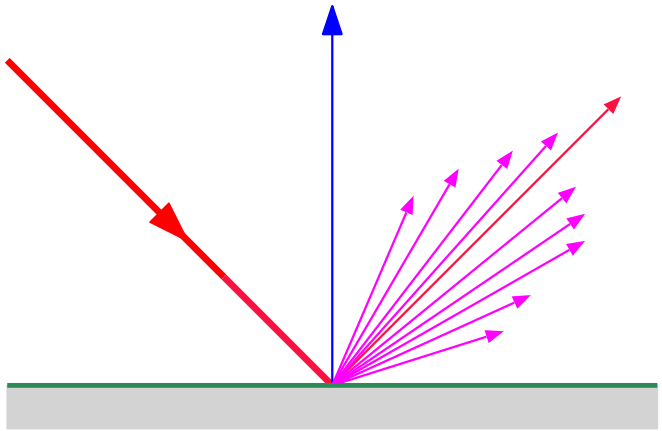
Blurring Effects: Penumbra and Soft Shadows

- Real light sources can be handled by tracing several rays to points on the light source, and by averaging over all these rays.
- In order to avoid regular anomalies in the darkness of the shadow, the points on the light source usually are randomly distributed.



Blurring Effects: Diffuse Reflection and Transparency

- The mirrored view of an object will always exhibit some diffusion.
- The diffusion caused by an uneven mirror can be simulated by tracing rays from the surface in the mirror direction, where each ray is slightly perturbed.
- The distribution can be weighted according to the same function that determines highlights.



- The problem of translucency is similar to the problem of diffuse reflection.
- Translucency is calculated by distributing the secondary rays about the main direction of the transmitted light.
- The distribution of the transmitted rays is defined by a specular transmittance function.

Blurring Effects: Depth of Field

- A camera never produces a sharp image of all objects in a scene.
- Rather, only those objects that are located within a range determined by the focal length and the aperture of the camera's optical system will appear sharp: *depth of field*.
- Depth of field can be generated by distributing several rays along the main ray direction, using a weighted distribution.
- That is, more rays are cast with a small variation and fewer rays are cast with a larger variation.
- This corresponds to several rays entering a real world camera since the aperture is always greater than a single point.
- The wider the rays are distributed, the fewer objects will appear sharp and crisp.

Blurring Effects: Motion Blur

- The image of any moving object will always be blurred to some extent as soon as the time of exposure is a finite interval.
- This effect is easy to simulate, too: Instead of calculating one picture only for one distinct moment in time, several pictures are calculated for different moments within the exposure time.
- The mean of these calculations will yield a picture with motion blur.
- Motion blur is particularly important when generating animations. Without motion blur, figures in animated still frames move cartoon-like, i.e., they do not move smoothly.

Distributed Ray Tracing

- To render pictures with all the effects described on the previous slides is very costly, if not impossible, if they are calculated in a straightforward way.
- For example, if one uses
  - 10 points of time per pixel for motion blur,
  - 10 points on the lens for the depth-of-field effect,
  - 10 shadow feelers per intersection point, and
  - 10 reflectance directions per ray for gloss,

  then one will have to cope with 10 000 rays per pixel, and with some 10 000 000 000 rays for only the very top part of the ray tree when ray tracing a scene on a 1000×1000 pixel display.
- The number of rays needed sky-rockets once truly recursive ray tracing is applied.
- Obviously, such a simple-minded approach is hardly feasible even on state-of-the-art rendering platforms . . .

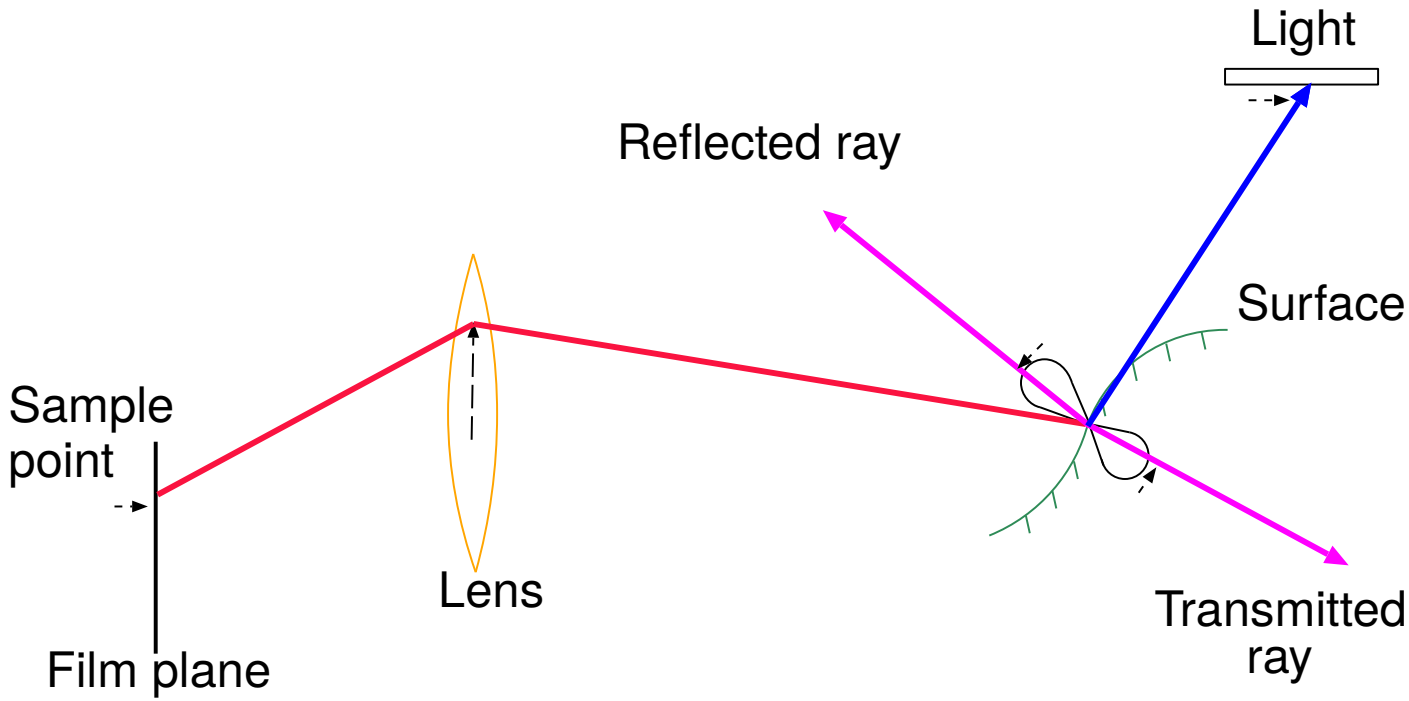
Distributed Ray Tracing

- [Cook et al. 1984]: Distributed ray tracing makes use of the fact that the true value of a pixel is given by a multi-dimensional integral (via Kajiya's rendering equation)
  - over time,
  - over the lens aperture,
  - over the light sources,
  - over the pixel area (for anti-aliasing), and
  - over all reflectance (and transparency) directions.
- If r reflectance/transparency directions are considered, we get a $(7 + 2r)$-dimensional integral.
- This integral can be estimated as a whole with the Monte Carlo method:
  - For every sample during this integration only one discrete value in each dimension is needed.
  - This one value can be a random value from the relevant interval in the respective dimension.
- That is, we get a sample value by considering one random path inside of the ray tree rather than the entire tree.
- The final pixel value is obtained by averaging several samples.
- This approach reduces the efforts drastically without sacrificing image quality.

Distributed Ray Tracing

1. Determine the spatial location of the starting point of the ray randomly within the pixel.
2. Determine the time for the ray randomly within the relevant time interval, and set up the scene and camera at that time.
3. Randomly determine a point of the lens for the ray to go through and calculate the refraction of the ray.
4. Intersect this ray with the scene and evaluate the intersection point closest to the lens, using standard ray tracing.
5. For each light source determine a random point on it and trace a ray to this point. If no object is intersected, include the influence of the light source to the color of the ray.
6. Determine a reflection ray randomly, evaluate the reflection ray with distributed ray tracing and include the influence of this ray to the color.
7. Determine a transparency ray randomly, evaluate it with distributed ray tracing and include the influence of this ray to the color.
8. Repeat Steps 1–7 until the mean of the obtained colors fulfills some quality criterion.

Distributed Ray Tracing

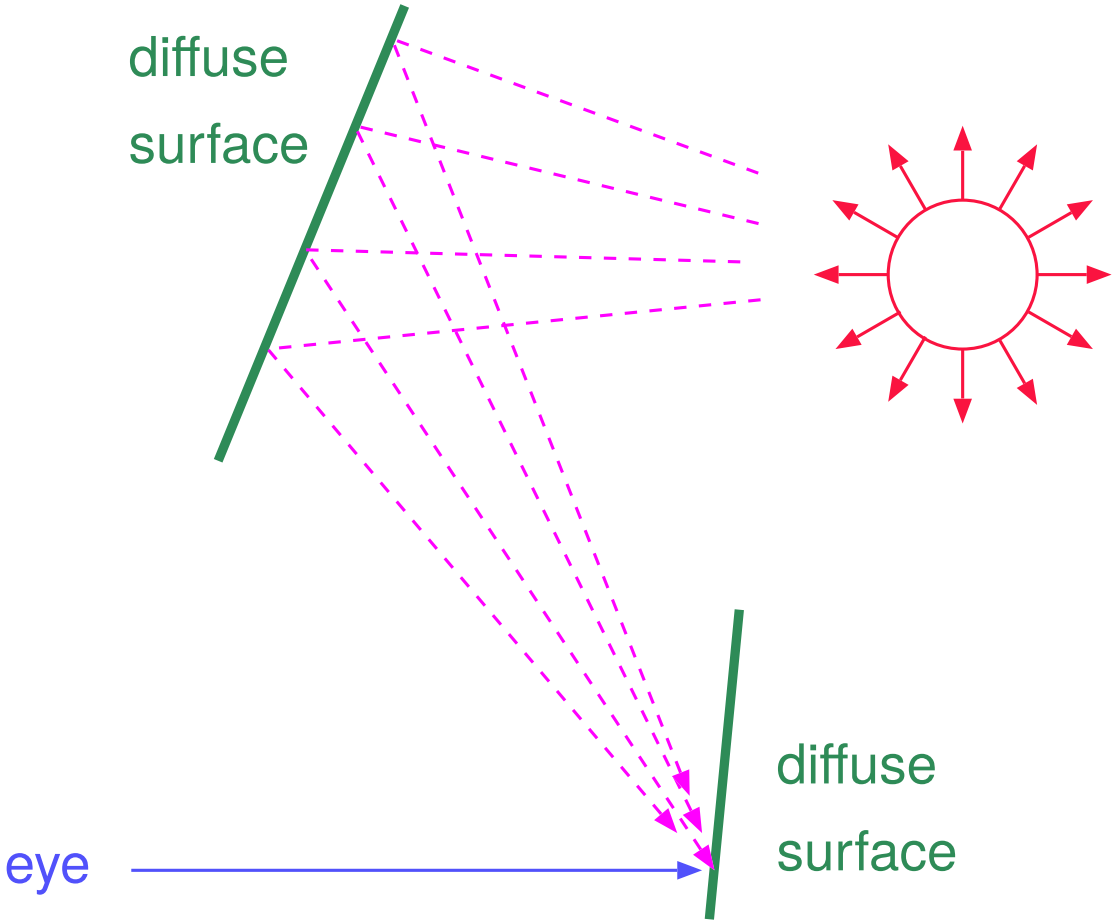


- The random selections in Steps 1–3 and 5–7 depend on different distribution functions for the different random variables, of course:
  - for Steps 1 and 2, the Gaussian turns out to be well suited,
  - for Steps 3 and 5, a uniform distribution is obvious, while
  - for Steps 6 and 7, the distributions correspond to the specular reflection term of the underlying shading model.

Distributed Ray Tracing

- Recall that Monte Carlo integration does not suffer from the curse of dimensionality.
- That is, the number n of samples required to achieve a specific error rate does not grow exponentially with the number of dimensions.
- Its error rate is proportional to $1/\sqrt{n}$.
- That is, if one wants to cut the error rate into half then one needs to quadruple the number of samples.

Shortcomings of Ray Tracing



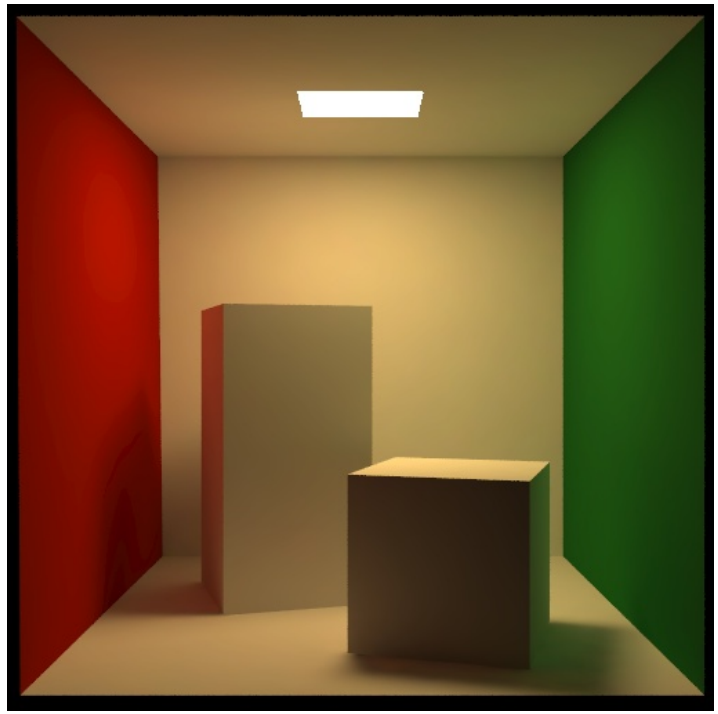
Diffuse-to-diffuse light transport

Even distributed ray tracing will not handle LD^*E light transport correctly!

Cornell Box

- [Goral et al. (1984)]: One area light source in the center of a white ceiling, green right wall, red left wall, white back wall, and white floor.
- Physical model created and exact settings recorded.

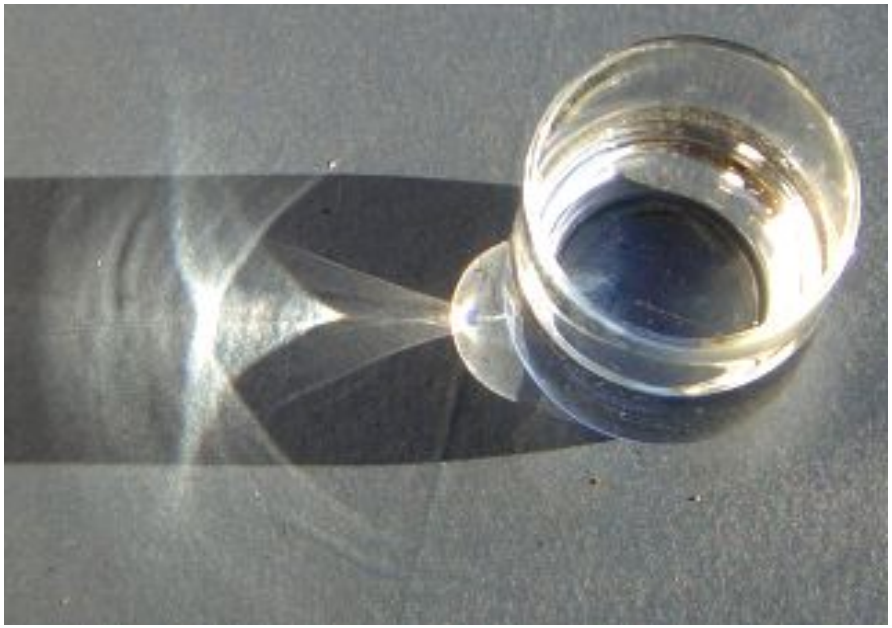
- Diffuse reflection! Hence, we should see global illumination with ambient occlusion, soft shadows, and some amount of color bleeding.
- First realistic renderings of the Cornell box achieved by means of the radiosity approach [Goral et al. (1984)].
- The Cornell box has become a standard test piece of computer graphics for multiple diffuse reflection.

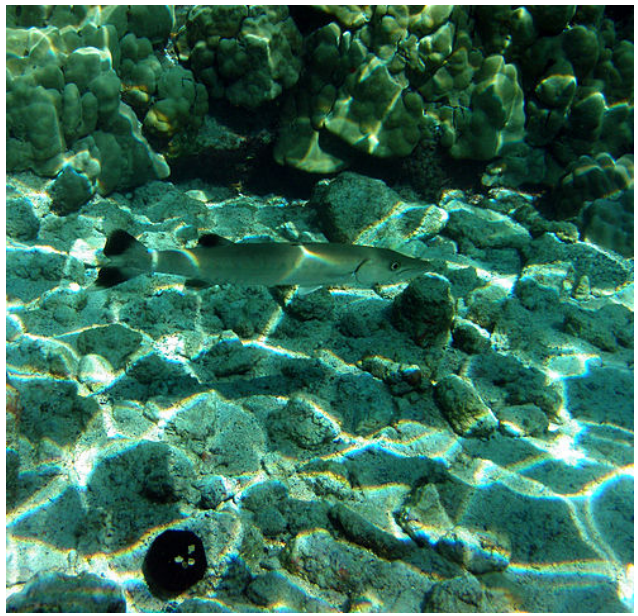


[Image credit: Cornell University Program of Computer Graphics]

Caustics

- A caustic is (the projection of) an envelope of light rays reflected or refracted by a curved surface or object: Light focuses through a specular surface onto another surface.
- Caustics are visible as high-intensity curves.
- In video games it is common to head for visually pleasing results without quest for physical correctness, e.g., by resorting to pre-computed textures.



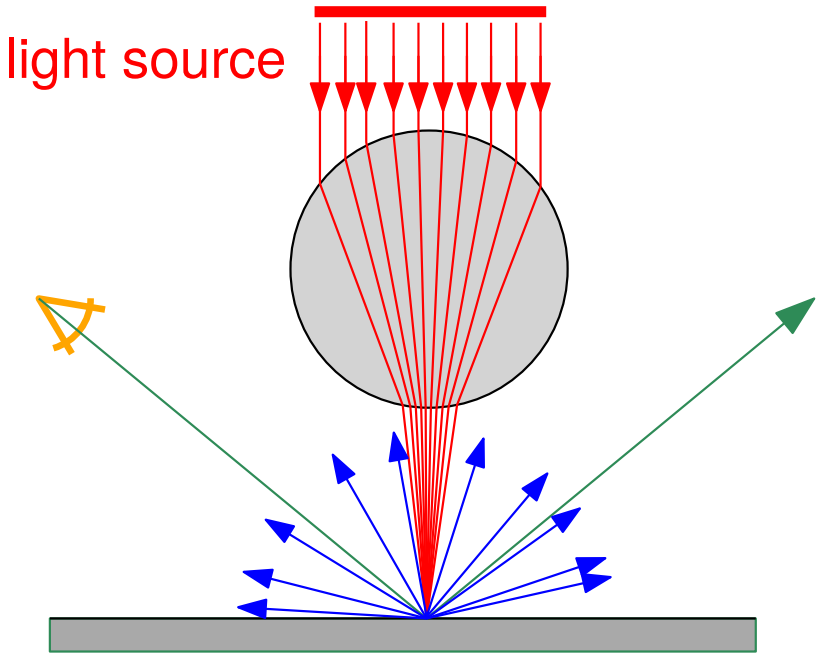


[Image credits: Wikipedia.]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Caustics

- A caustic is difficult to capture by ray tracing if it occurs on a diffuse surface.



Path Tracing

- In a nutshell, the simple reflectance model used by standard ray tracing is replaced by a *BSDF*, and Monte Carlo sampling of the paths is applied.
- A BSDF can be interpreted as a probability distribution function that tells us the probability that light which arrives from one direction is reflected (or transmitted) in some other direction.
- Of course, it needs to be defined for every point on the surface of an object.
- In the ray tracing, the actual reflectance/transmission directions are picked at random according to the BSDF distributions.
- The recursion is stopped when a maximum number of bounces is reached or when an emissive surface is hit.
- The results of the samples are averaged to obtain a pixel's final color.
- Path tracing tends to require hundreds or even thousands of samples per pixel until an image of acceptable quality is obtained.

Path Tracing

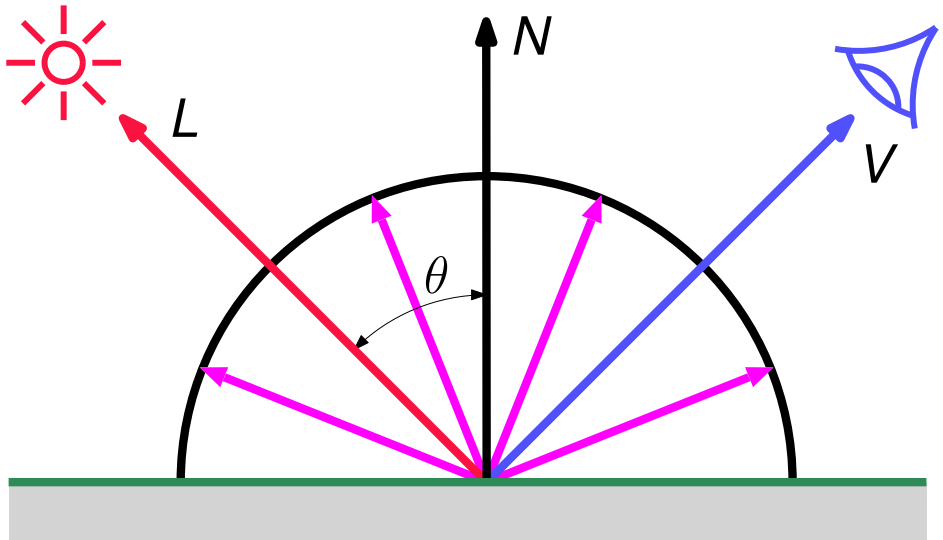
- Noise can be reduced by progressive updating of the result obtained so far: If L_j is the pixel value after the j-th sampling then a new estimator can be obtained by combining it with a new result L_{new} as

$$L_{j+1} := \frac{j \cdot L_j + L_{new}}{j+1}.$$

- This supports stop-and-go rendering, with no need to start from scratch when refining a result.
- Furthermore, the convergence can be inspected between progressive updates.
- The BSDF sampling blends in neatly with the sampling done by distributed ray tracing. Thus, all the effects achieved by distributed ray tracing can also be achieved by path tracing.
- Path tracing handles (most) $L(D|S)^*E$ paths.
- Noise caused by the sampling is a problem when still images are shown as part of an animation: “De-noising” techniques are used.
- Path tracing is parallelizable at both the sample and at the pixel level.
- Recent GPUs provide impressive support for path tracing!

Path Tracing: Importance Sampling

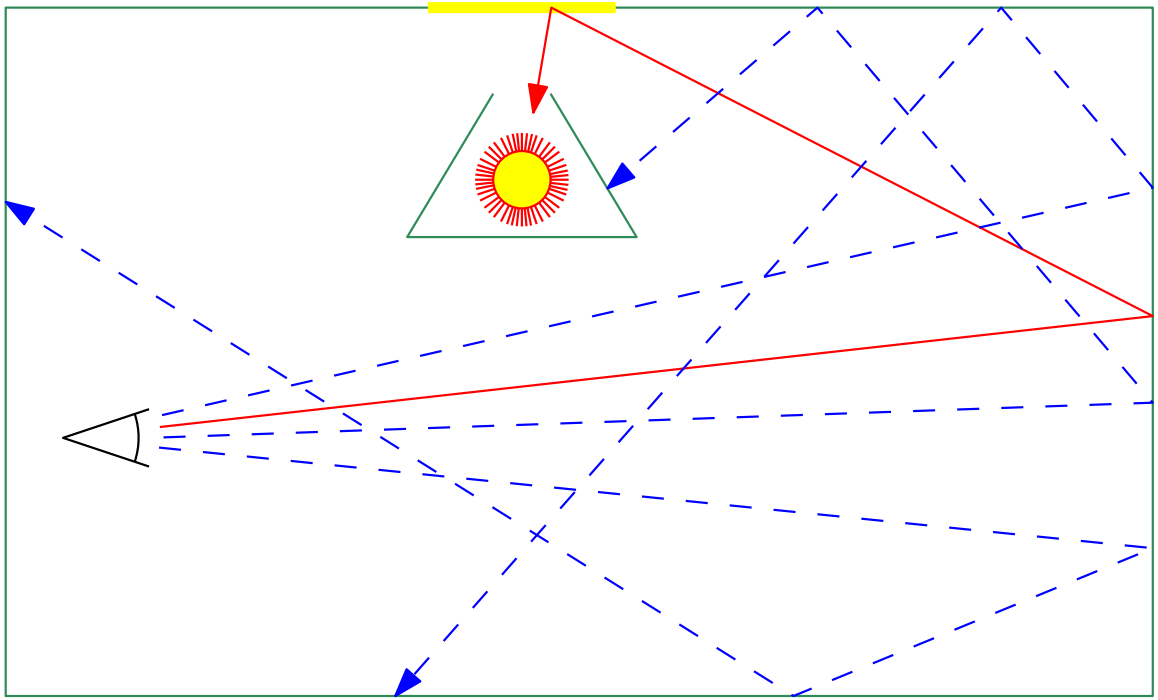
- Importance sampling is an active area of research with lots of recent results!
- E.g., recall that Lambert's Cosine Law tells us that the intensity of diffusely reflected light is dampened by the factor $\cos\theta$.
- Hence, for a surface with a (mostly) diffuse reflection one can improve the convergence of the algorithm by modifying the random sampling of the BSDF such that bounced rays with small θ are more likely to occur.
- See `https://intro-to-restir.cwyman.org/` for a 2023 ACM SIGGRAPH Course on "reservoir-based spatiotemporal importance resampling".



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

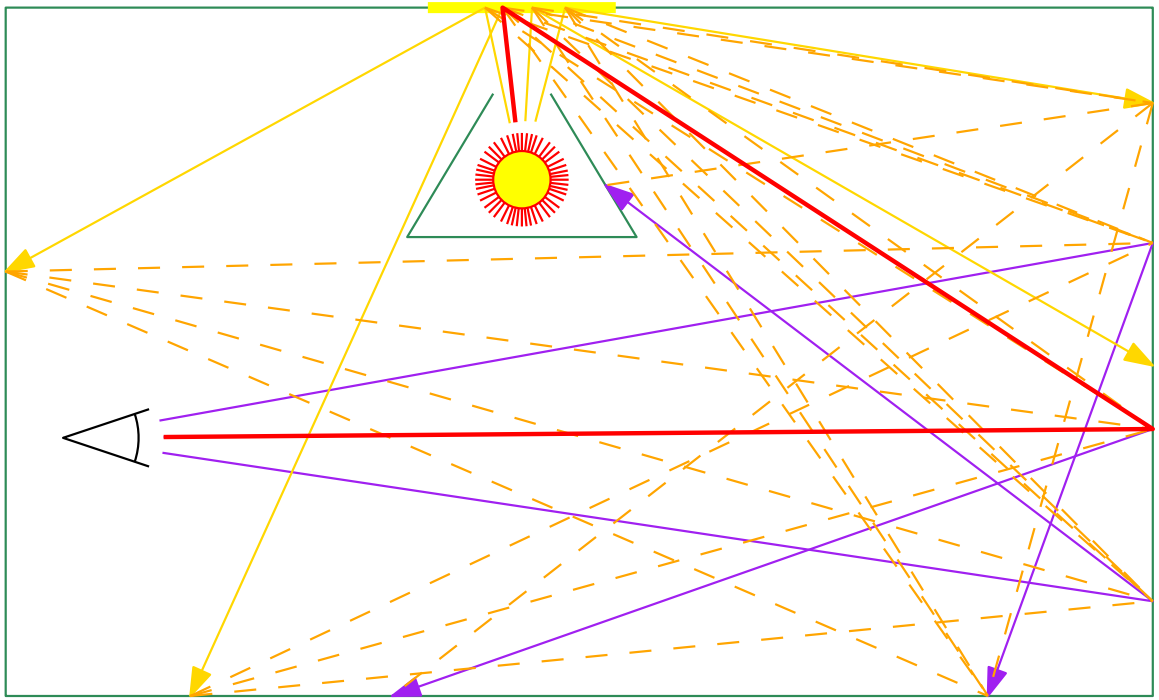
Bidirectional Path Tracing

- Standard path tracing can exhibit high variance if most of the scene is lit only via indirect illumination.
- In this case most shadow feelers will not reach the light source(s) and, thus most paths will have no contribution.
- But a few paths will have a large contribution.
- Hence, we get a large variance and slow convergence.



Bidirectional Path Tracing

- [Lafortune&Willems (1993), Veach&Guibas (1994)]: Send out (short) eye paths.
- Send out (short) light paths, following the same BSDF sampling rules.
- Connect every vertex of an eye path with every vertex in a light path.
- This gives tons of paths from the light to the eye.
- Again, Monte Carlo sampling is applied to these paths.
- Good for (1) caustics, (2) highly reflective or transparent materials where light can bounce in unpredictable ways, and (3) in case of complex indirect lighting.



UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

Photon Mapping

- Concept similar to bidirectional path tracing.
- Again, rays (“photons”) are traced from the light sources into the scene.
- Whenever a surface is hit, the amount of light brought to it is recorded and a new ray is picked based on local BSDF sampling.
- The information stored on the light-surface impacts is called a *photon map*.
- This information is used during the subsequent gathering phase, based on path tracing.
- Photon mapping allows to capture caustics and color bleeding.

[Image credits: Wikipedia.]

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab

The End!

I hope that you enjoyed this course, and I wish you all the best for your future studies.

UNIVERSITÄT SALZBURG
Computational Geometry and Applications Lab