Course on Virtual Reality

Basic Principles of 3D Computer Graphics

Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

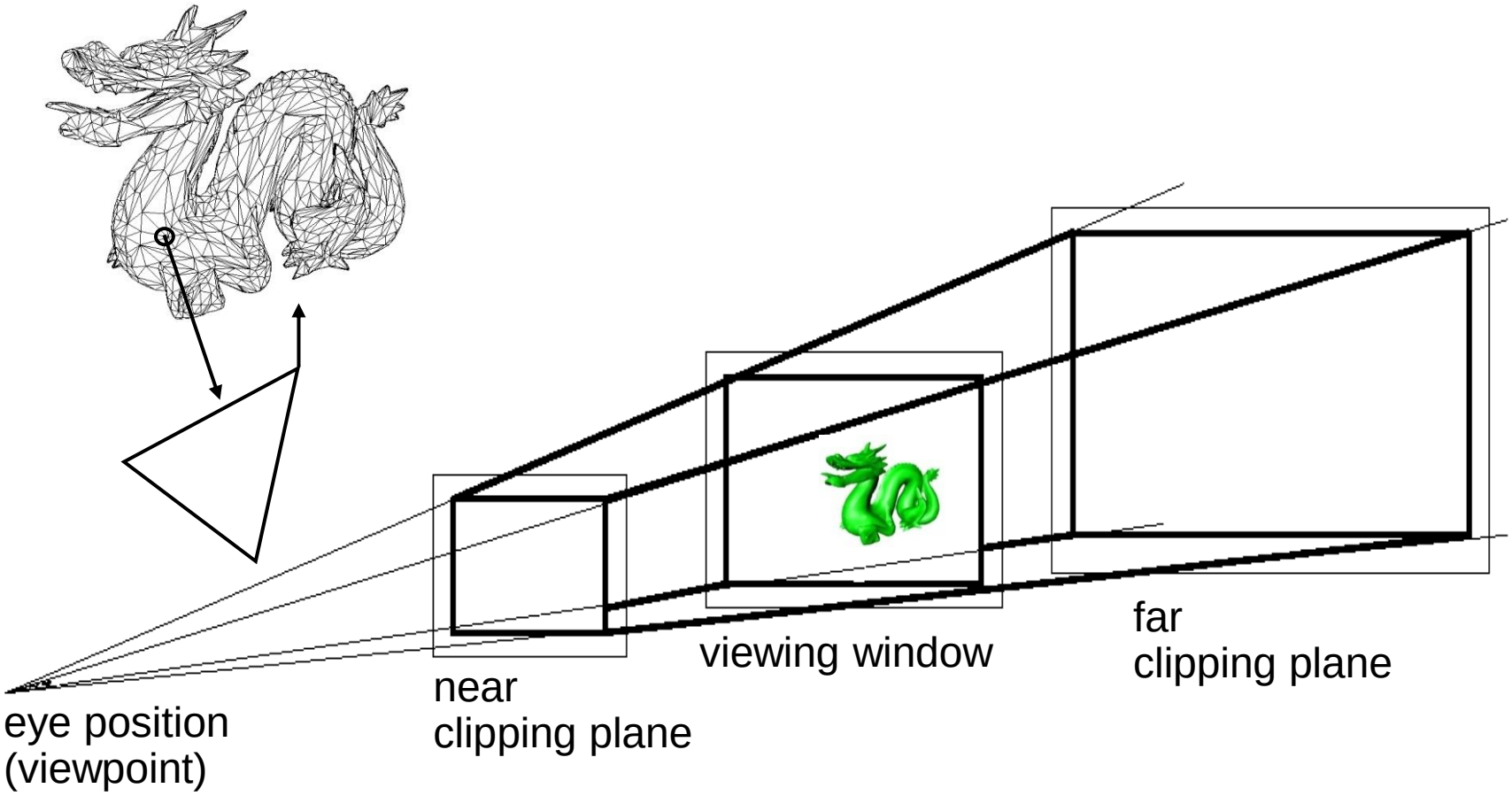
    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

Rendering

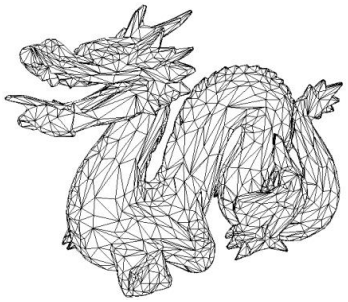
Definition

Rendering is the process of drawing polygonal 3D objects (polyhedrons) onto a two-dimensional plane (screen)

View Volume


eye position
(viewpoint)
near
clipping plane
viewing window
far
clipping plane

Rendering Pipeline Overview



Transformation: position and orientation of objects

Transformation: WKS, SKS

Culling

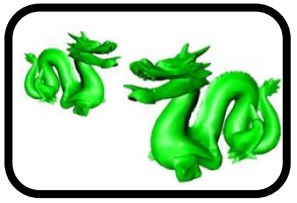
Projection

Clipping

Shading

Hidden Surface Removal

Scan Conversion



Steps (1/2)

1. Transformations:
   - Describe position & orientation of objects within a scene
   - Transform local coordinates of objects into a single world coordinate system (WKS)
   - Optional: Transform into a view Coordinate system (SKS): viewpoint = origin

2. Culling:
   - Remove invisible polygons

3. (Perspective) Projection:
   - Transform Vertices into screen coordinate system

Steps (2/2)

4. Clipping:
   - Truncate geometry outside the view volume

5. Scan Conversion:
   - Transform the polygonal model into a set of pixels (picture elements)

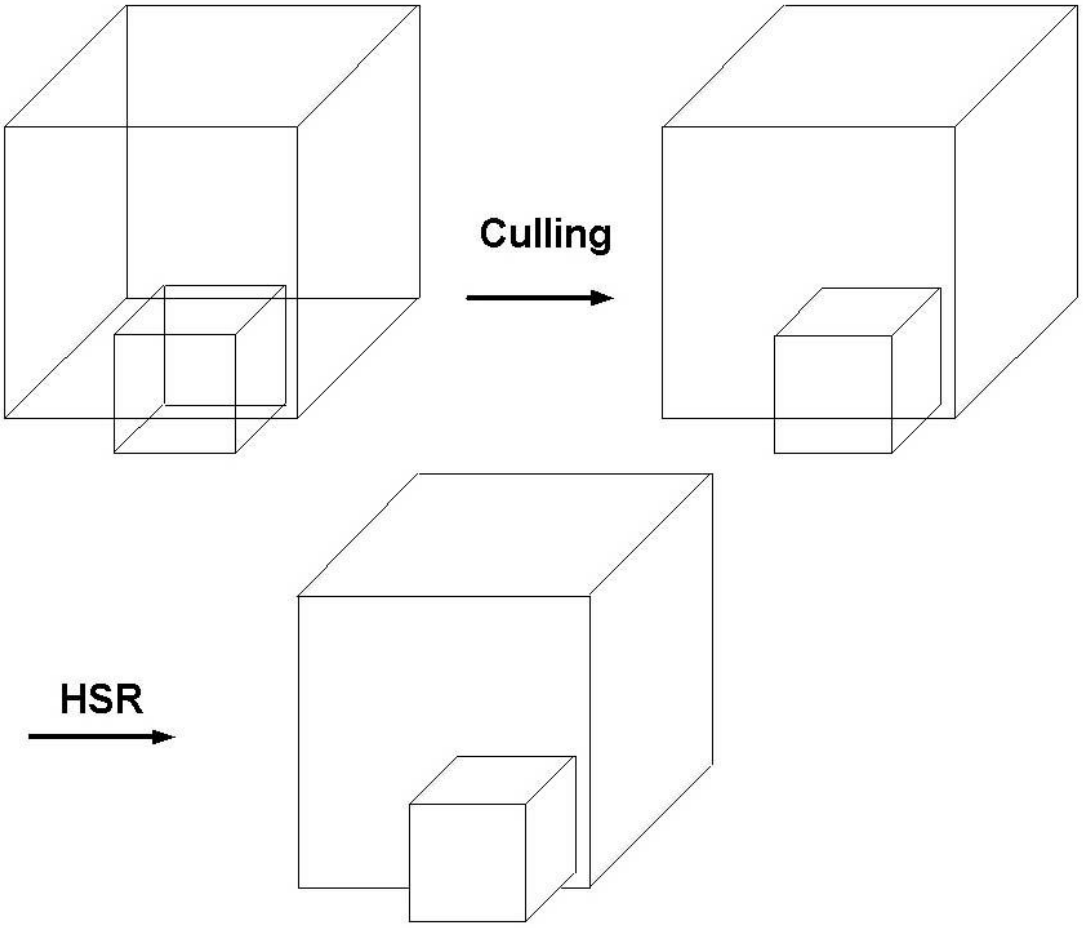
6. Hidden Surface Removal (HSR):
   - Remove invisible pixels, covered by other objects

7. Shading:
   - Find color (intensity) values of single pixels based on an illumination model

Remarks

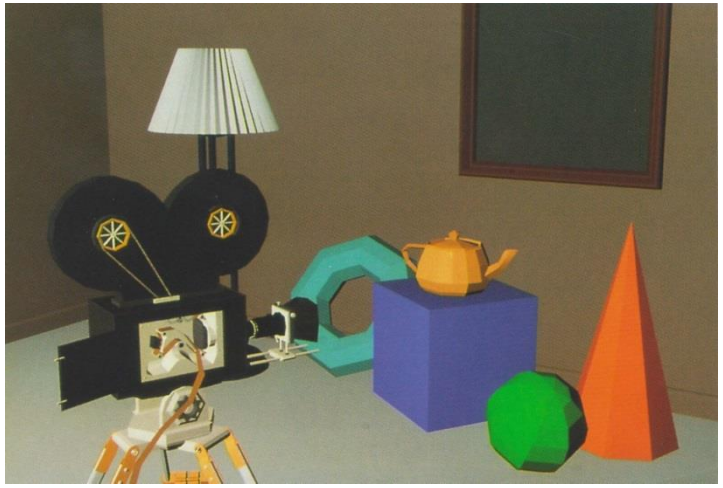
- Step order can vary
- 5-7 are normally combined into a single step (Pixel Loop)
- Pipeline principle: speed up
- 2 is a special case of 6: performance!

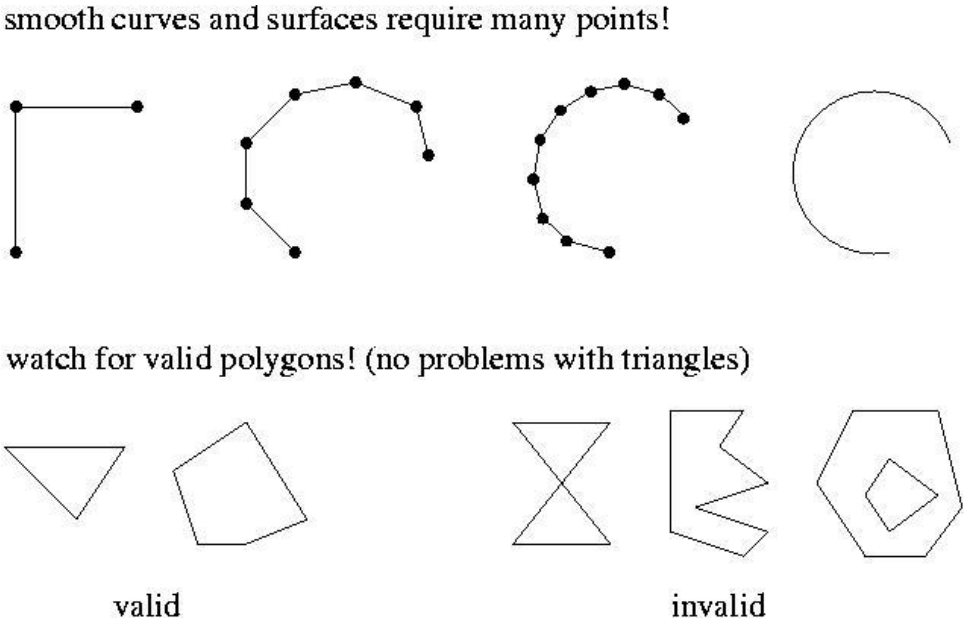
Culling versus Hidden Surface Removal



Polygonal Representation

Approximation of object geometry by a grid of planar, polygonal facets

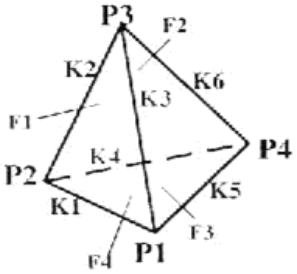


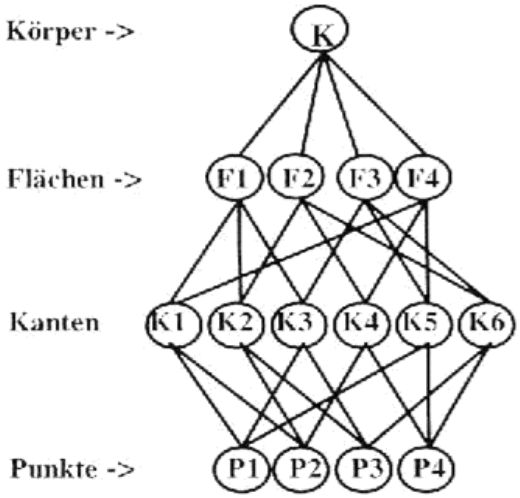


Pictures: Foley et al., Watt

Data Structure for Polygonal Representation

- List of 3D coordinates (x, y, z) (polygon vertices)
- Edges implicitly
- Additional information: normals of polygons or polygon vertices for shading, color, material, …
- Hierarchical structure of single geometries (scene graph)
- Polygons are rendered independently from each other
- Drawback:
  Redundant rendering of common edges
- Alternative:
  explicit storage of edges – each edge is a list of 4 parameters: 2 adjacent vertices and 2 polygons





Pictures: Schuhmann

Polygonal Representation - Benefits

- Hardware Rendering
- Efficient shading algorithms
- Arbitrary geometries
- Trade Off: accuracy (number of polygons)
  versus
  speed (frame rate, polygons per second)

Functional Representation

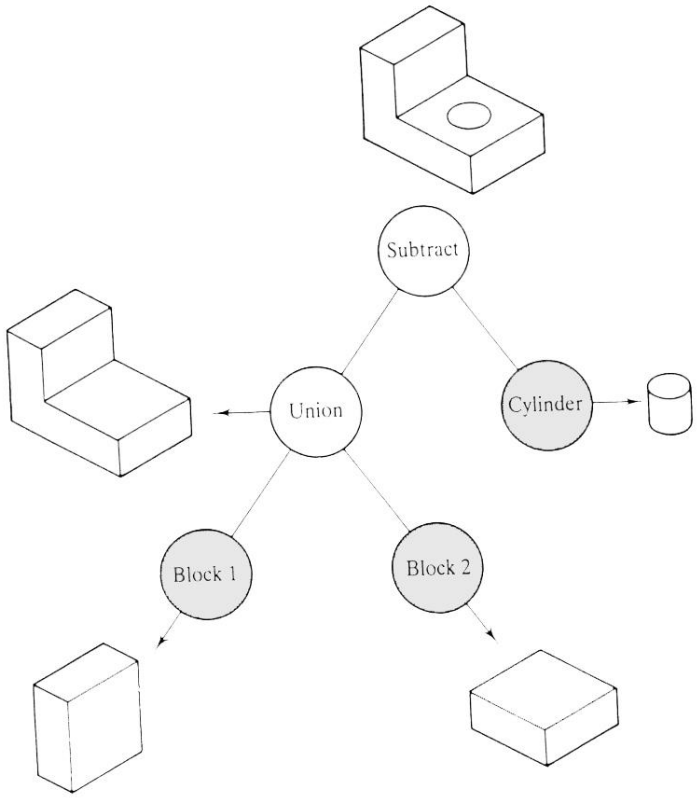
E.g., sphere: $x^2 + y^2 + z^2 = r^2$

- Description of objects by a single function
- Very efficient collision detection
- Problem: Very limited object geometry

Constructive Solid Geometry (CSG)

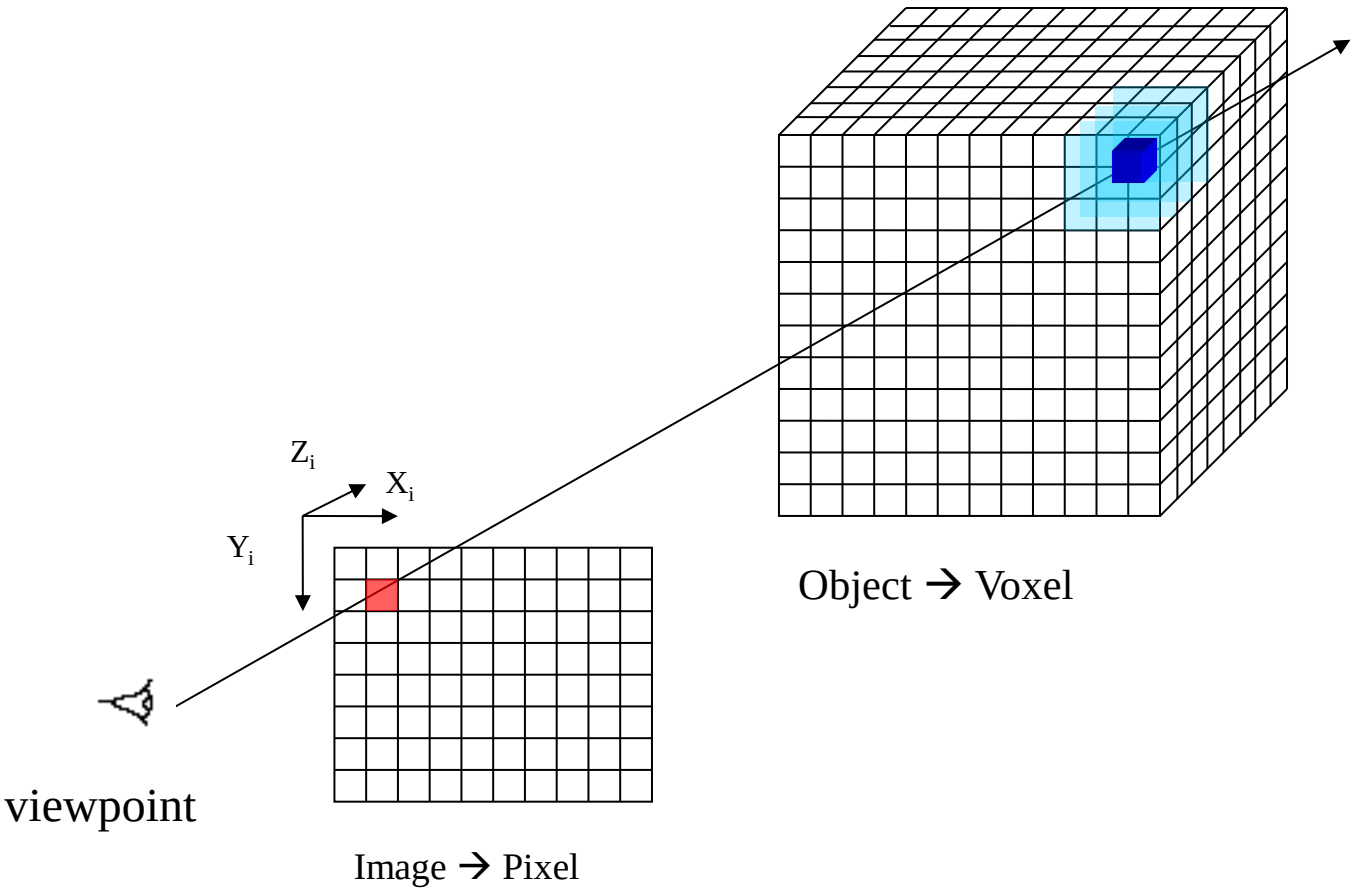
Combining elementary objects (primitives: spheres, cylinders etc.) via boolean set operators (union, intersection, complement): tree structure

- Efficient memory use
- Modeling of complex geometries in mechanical engineering
- Problem: limited object geometry



Picture: Watt

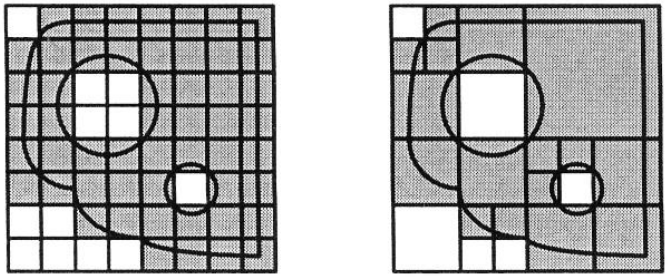
Volume Rendering: Ray Casting



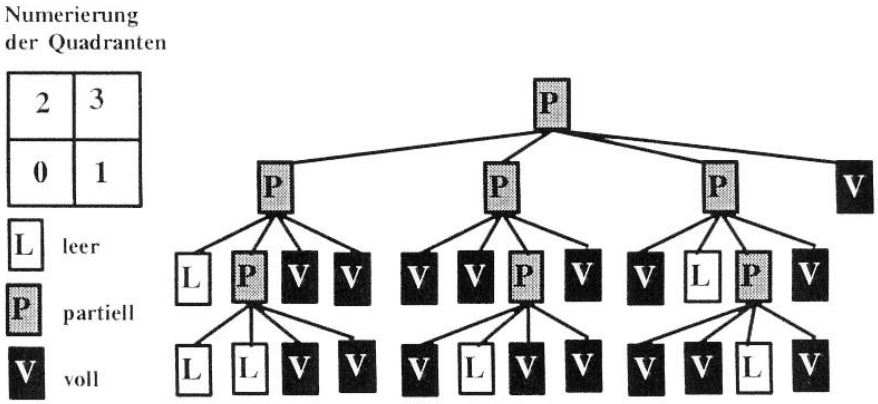
Volume Rendering: Examples

Space Subdivision

space sections covered by geometry are indexed



- Octrees (2D: Quadtrees)
  Recursive quartering of the plane
- Binary Space Partitioning (BSP)
  Adaptive subdivision by variable section size

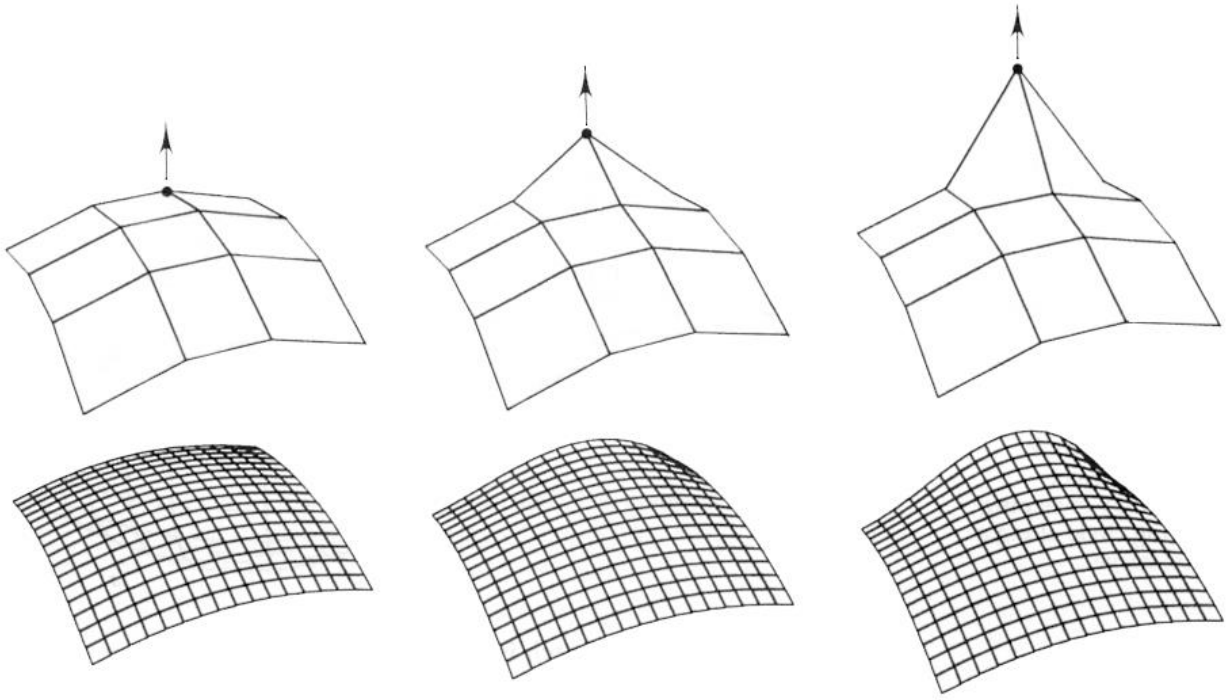


Picture: Schuhmann

Bézier Patches

Parametric representation via cubic polynomials in 2 variables

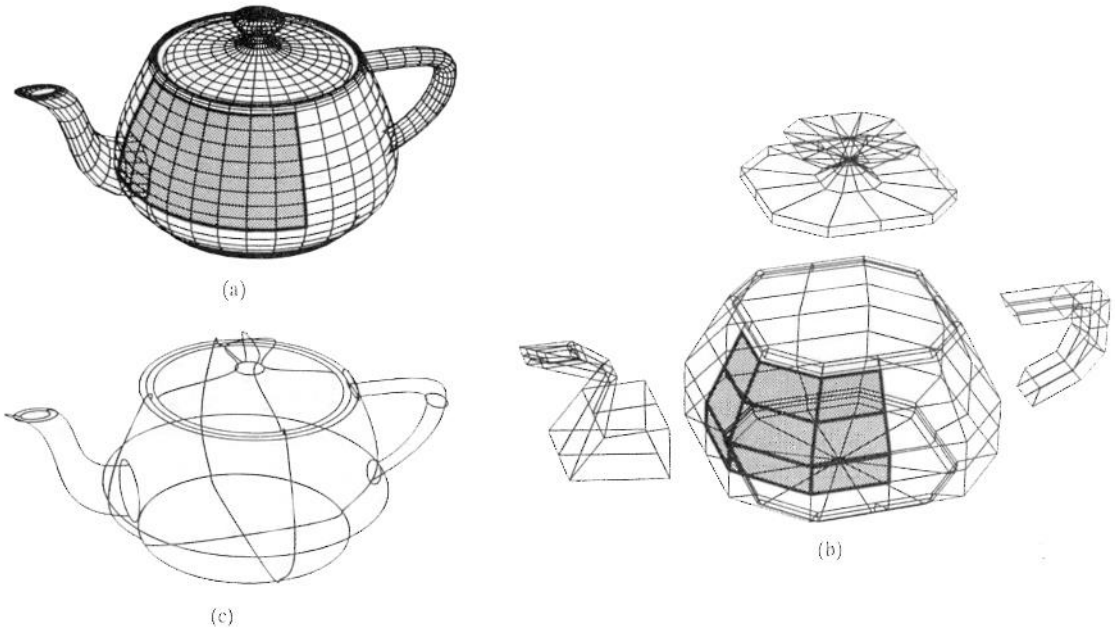
- Exact representation of geometries by the coefficients of the polynomial (16 control points)



Picture: Watt

Bézier Patches: The Utah Tea Pot

- Efficient memory use
- Interactive design of curved surfaces (product design)
- Problem: Integrity of representation across patch boundaries



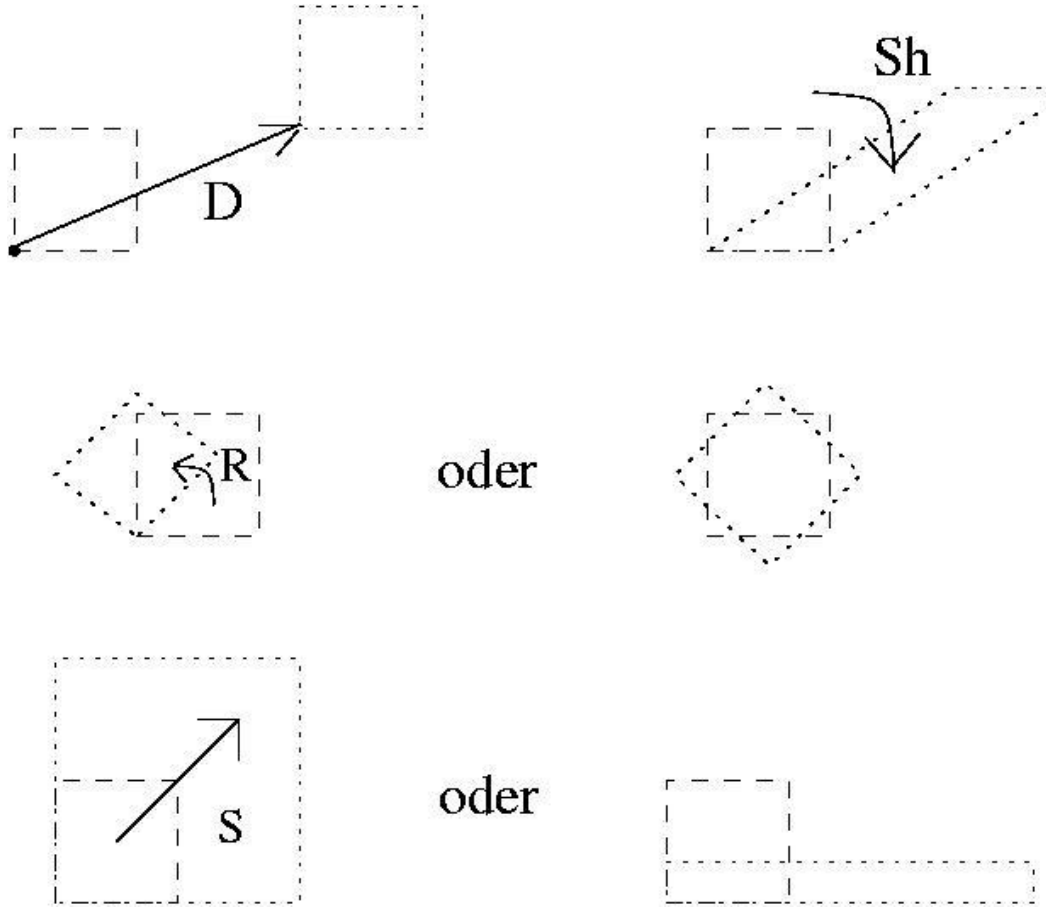
Picture: Watt

Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

Basic Operations (1/2)



Basic Operations (2/2)

Polygonal model: Just transform the vertices!

- Conventions:
  - Points are described as column or row vectors P (x, y, z)
  - Transformations are described as matrices:
    - Translation P´ = P + D (D = translation vector)
    - Rotation P´ = P * R (R = rotation matrix)
    - Scaling P´ = P * S (S = scale matrix)
    - Shearing P´ = P * Sh (Sh = shear matrix)

Homogeneous Coordinates

Inconsistency:
Translation is vector-vector-addition, all other operations are matrix vector-multiplication

Solution:
Homogeneous coordinates – Increase dimension by 1

P(x,y,z) $\longrightarrow$ P(X,Y,Z,w), with $x = \frac{X}{w}, y = \frac{Y}{w}, z = \frac{Z}{w}$

Convention: w =1

Translation

$$P' = P \cdot T$$

$$T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

$$x' = x + T_x, \quad y' = y + T_y, \quad z' = z + T_z$$

Scaling

$$P' = P \cdot S$$

$$S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$x' = x \cdot S_x, \quad y' = y \cdot S_y, \quad z' = z \cdot S_z$$

Rotation

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation about the z-axis:

$$x' = x\cos\theta - y\sin\theta \quad , \quad y' = y\sin\theta + y\cos\theta \quad , \quad z' = z$$

Application

Concatenation of transformations by matrix-matrix-multiplication:

Representation of arbitrary linear transformations by a single matrix

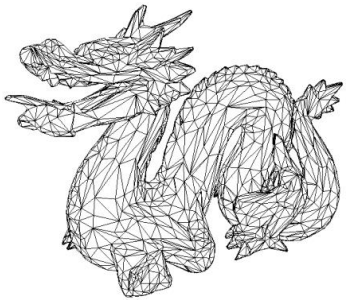
- Operations on points of a single coordinate system
- Convert one coordinate system into another:
  - Local Object Coordinate Systems
  - Coordinate Systems for groups of objects
  - World Coordinate System
  - View Coordinate System
  - Screen Coordinate System

Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

Position in Rendering Pipeline



Transformation: position and orientation of objects

Transformation: WKS, SKS

Culling

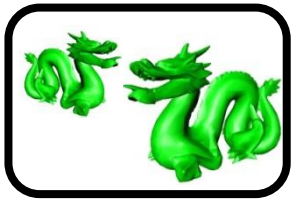
Projection

Clipping

Shading

Hidden Surface Removal

Scan Conversion

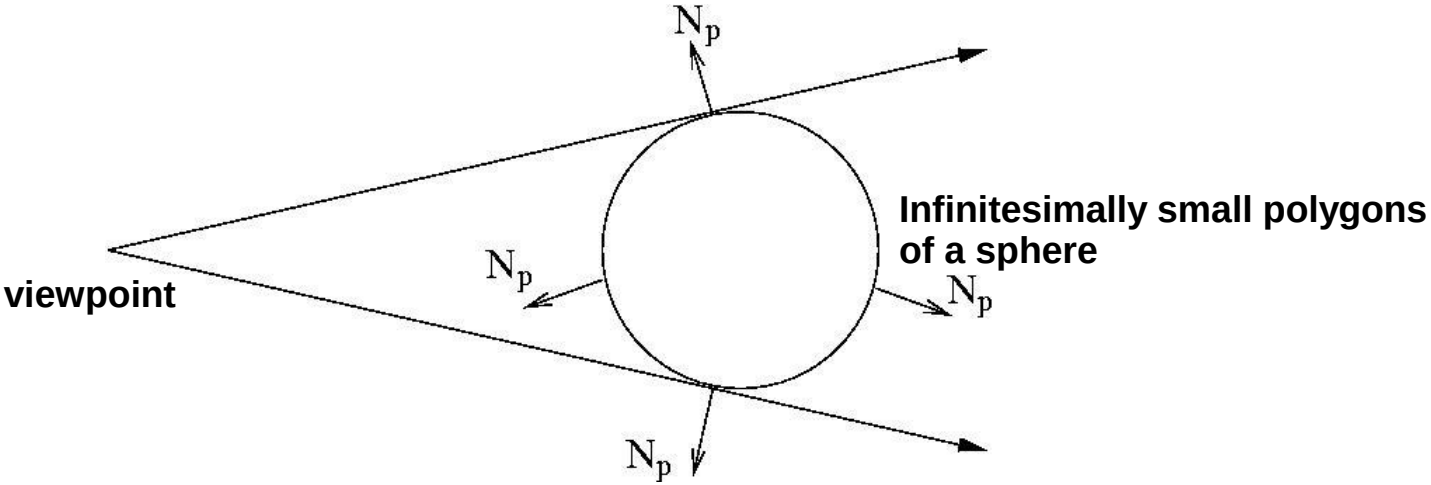


Algorithm

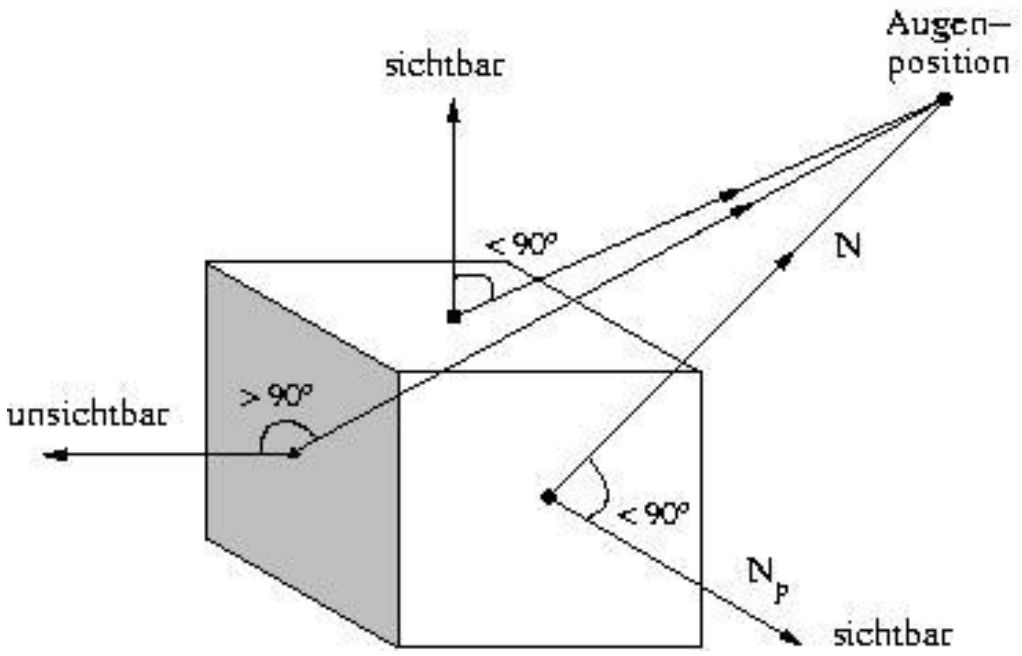
- Remove invisible polygons to speed up rendering (culling, backface elimination)
- Compare polygon orientation with “line of sight” vector

Simple, fast „algorithm“

– Polygon is invisible if and only if the angle between the polygon normal vector N_P and the line of sight vector N is larger than 90 degrees



Backface Elimination of a Cube

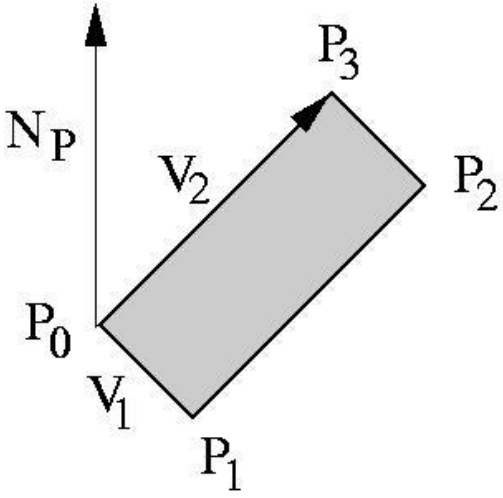


Picture based on Watt

Computation

$$\theta\left(N_P, N\right) < 90 \Leftrightarrow N_p \cdot N > 0 \quad \left(\cos\theta := \frac{N_p\, N}{\left|N_P\right|\left|N\right|}\right)$$

- Determination of N and N_P:
  - N is the position vector in the viewing coordinate system
  - Calculate N_P from 3 (non-collinear) polygon vertices



$$V_1 = P_1 - P_0$$
$$V_2 = \mathbf{P_3} - P_0$$
$$N_P = V_1 \times V_2$$

Remarks

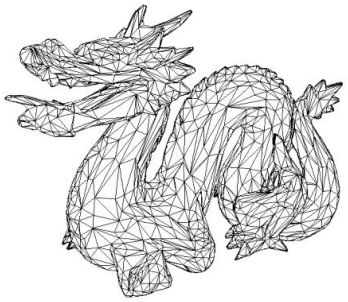
- Compute N_P's offline before the simulation starts, or store them with the model
- Reuse the normal vectors for shading
- At an average, half of the polygons of a polyhedron are invisible: Culling eliminates them at the very beginning of the rendering process

Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

Position in Rendering Pipeline



Transformation: position and orientation of objects

Transformation: WKS, SKS

Culling

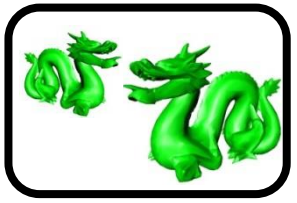
Projection

Clipping

Shading

Hidden Surface Removal

Scan Conversion



Definition

- General:
  A projection is a function from n-dimensional space to (n-1)-dimensional space

- Computer Graphics:
  - 3D ➡ 2D (screen, projection plane, viewing window)
  - Projection plane is flat: straight lines are mapped to straight lines

Perspective versus Parallel Projection

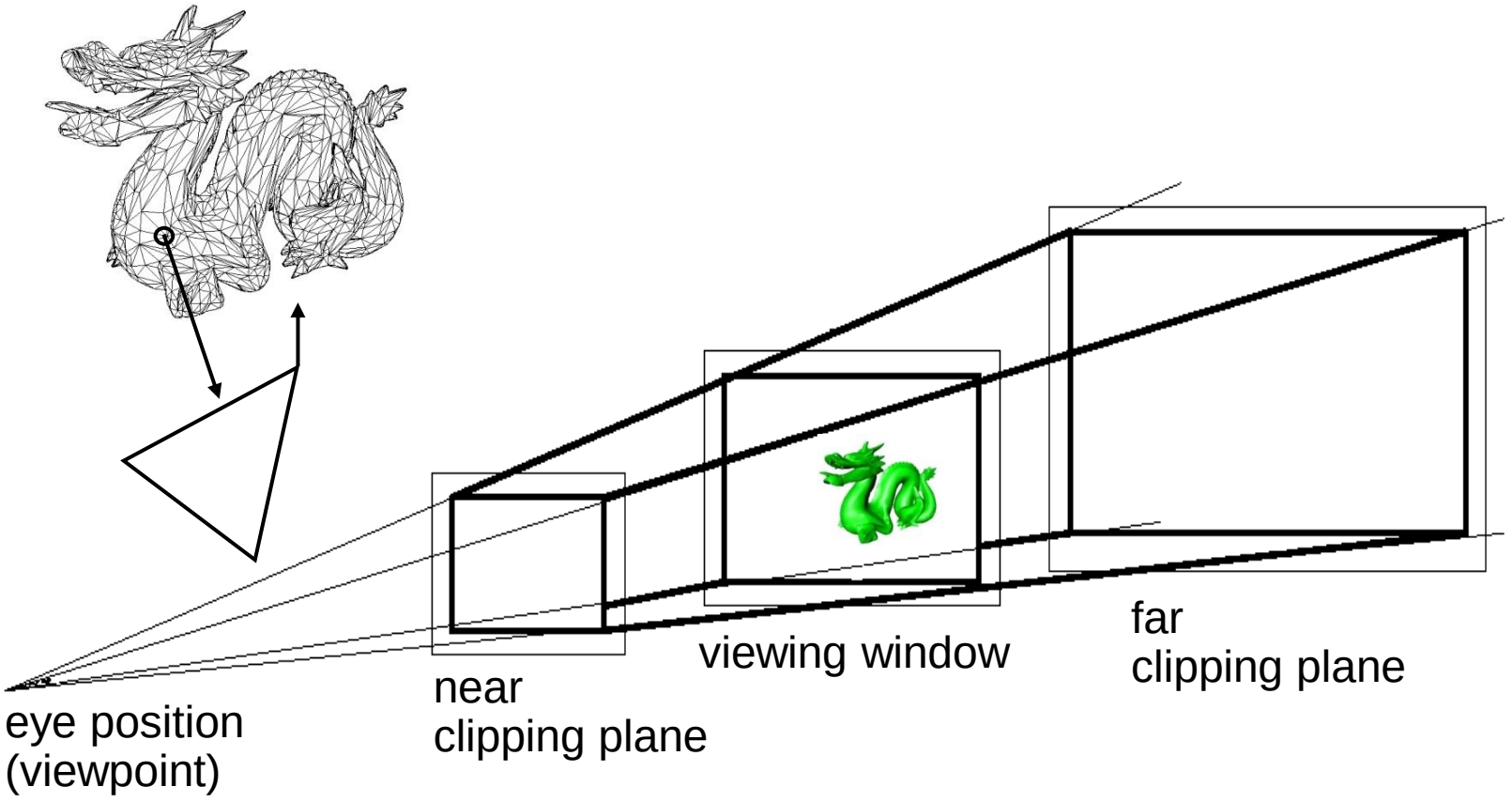
Perspective Projection

- Projectors (projection beams) meet in the center of projection (eye position, viewpoint)
- Objects get „smaller“ with rising distance from the center of projection
- view volume is a pyramid

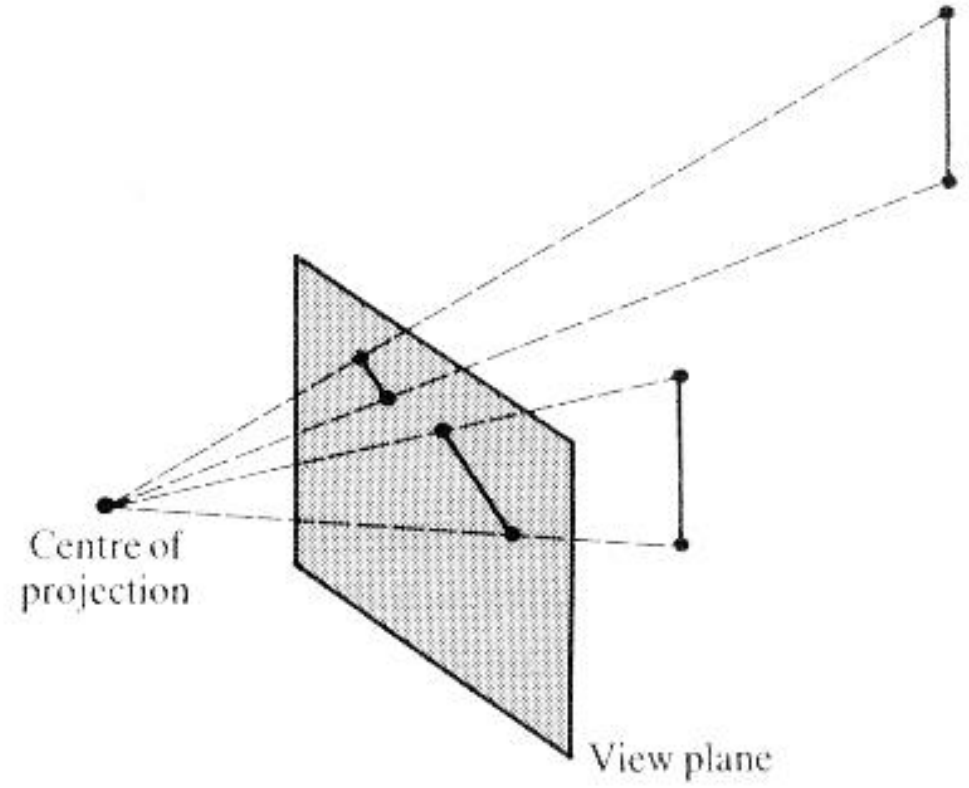
Parallel Projection

- Projectors (projection beams) are parallel, center of projection is in infinitum
- Does not correspond to natural viewing experience
- view volume is a cube

View Volume


eye position
(viewpoint)
near
clipping plane
viewing window
far
clipping plane

Perspective Shortening



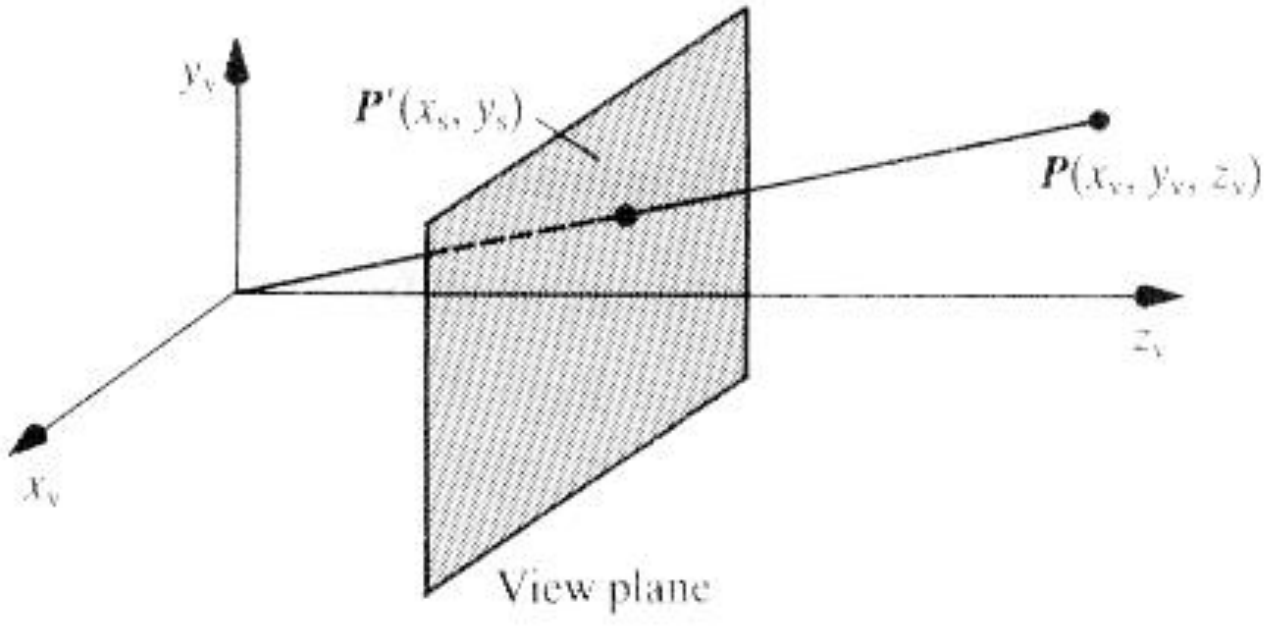
Picture: Watt

A very Simple Projection - Assumptions

- Perspective projection
- No „diagonal“ projection: Straight line from the center of projection to the center of the screen is parallel to the normal vector of the viewing window
- Constant distance *d* from the center of projection to the screen
- Computation:
  - View coordinate system
  - Normal vector of the viewing window is parallel to the z-axis

A very Simple Projection – Intersection Point

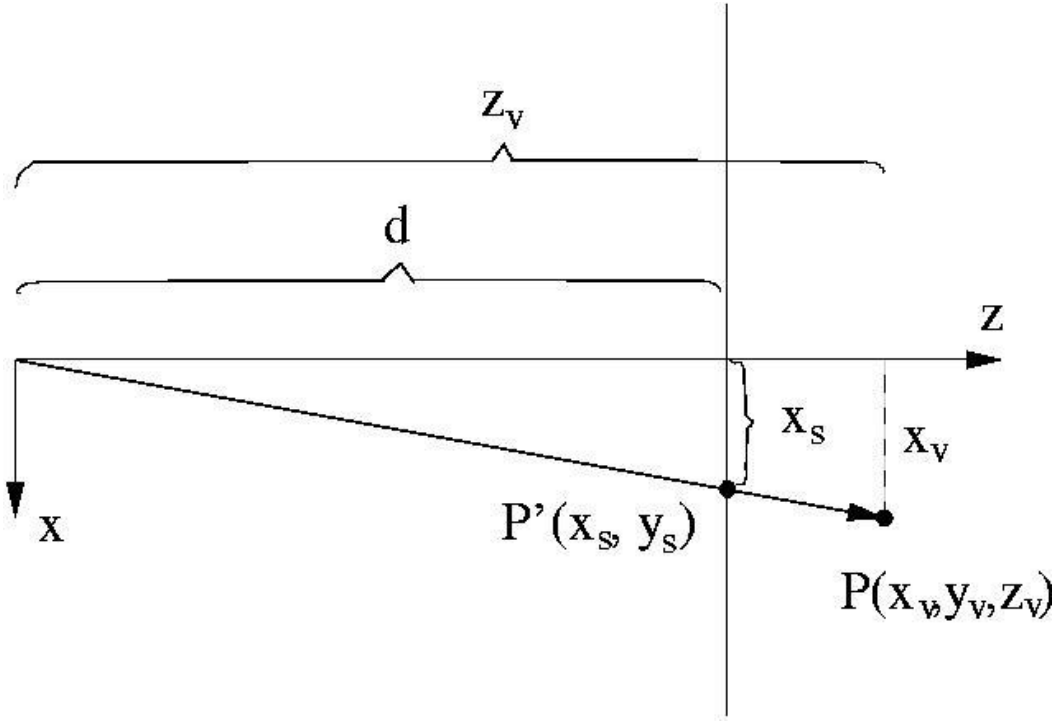
- For each point $P(x_v, y_v, z_v)$ of the object geometry in 3D space, we are looking for the intersection point $P'(x_s, y_s)$ of the projector (straight line from viewpoint to P) with the view plane.



Picture: Watt

A very Simple Projection – Computation

- Computation via similar triangles



$$\frac{x_s}{d} = \frac{x_v}{z_v} \quad (y_s \text{ analog}) \quad \Rightarrow x_s = \frac{x_v d}{z_v} = \frac{x_v}{z_v / d} \quad (y_s = \frac{y_v d}{z_v} = \frac{y_v}{z_v / d})$$

A very Simple Projection – Projection Matrix

Homogeneous coordinates:

$$X = x_V\,,\; Y = y_V \quad , \quad Z = z_V \quad , \quad w = \frac{z_V}{d}$$

$$(X\ Y\ Z\ w) = \begin{pmatrix} x_V & y_V & z_V & 1 \end{pmatrix} T_{Proj} \quad mit$$

$$T_{Proj} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{d} \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$x_s = \frac{X}{w}, y_s = \frac{Y}{w}, z_s = \frac{Z}{w} = d$$

Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

Points, Lines, and Polygons

OpenGL: Canonic View Volume: Test against rectangle

- Points: trivial (Test on $-1 < x, y, z < 1$)
- Lines: Cohen-Sutherland Algorithm
- Polygons: Sutherland-Hodgeman Algorithm

Clipping of Lines – Brute Force Method

Problem:
For a given rectangle (x_{min}, y_{min}, x_{max},y_{max}) – OpenGL: (-1,-1,1,1) – and a line with start point P_1 (x_1,y_1) and end point P_2 (x_2,y_2)

$$\overline{P_1P_2} = P_1 + t(P_2 - P_1) \quad 0 \leq t \leq 1$$

3 cases:

- P_1 and P_2 are within the rectangle: Draw $\overline{P_1P_2}$
- P_1 XOR P_2 are within the rectangle: Calculate the intersection point S of the line with one of the rectangle edges and draw $\overline{SP_2}$ or $\overline{P_1S}$,respectively
- P_1 and P_2 are outside the rectangle: Calculate all intersection points of the line with the rectangle edges
  - No intersection point: Nothing to draw
  - 2 intersection points: Draw $\overline{S_1S_2}$

Clipping of Lines – Cohen Sutherland

Idea:
Subdivision into 9 partitions, each being the extension of the rectangle edges Bottom (B), Top (T), Left (L), and Right (R)

Define outcode (P): = $B_p T_p L_p R_p$ (4 Bit-Number)

$$R_p = 1 \Leftrightarrow x > x_{max} \Leftrightarrow x_{max} - x < 0 \Leftrightarrow$$ P outside R

$$L_p = 1 \Leftrightarrow x < x_{min} \Leftrightarrow x - x_{min} < 0 \Leftrightarrow$$ P outside L

$$T_p = 1 \Leftrightarrow y > y_{max} \Leftrightarrow y_{max} - y < 0 \Leftrightarrow$$ P outside T

$$B_p = 1 \Leftrightarrow y < y_{min} \Leftrightarrow y - y_{min} < 0 \Leftrightarrow$$ P outside B

Clipping of Lines – Cohen Sutherland (cont.)

2 simple tests:

- trivial_accept: both points within rectangle
- trivial_reject: both point outside with respect to a single edge

Otherwise:

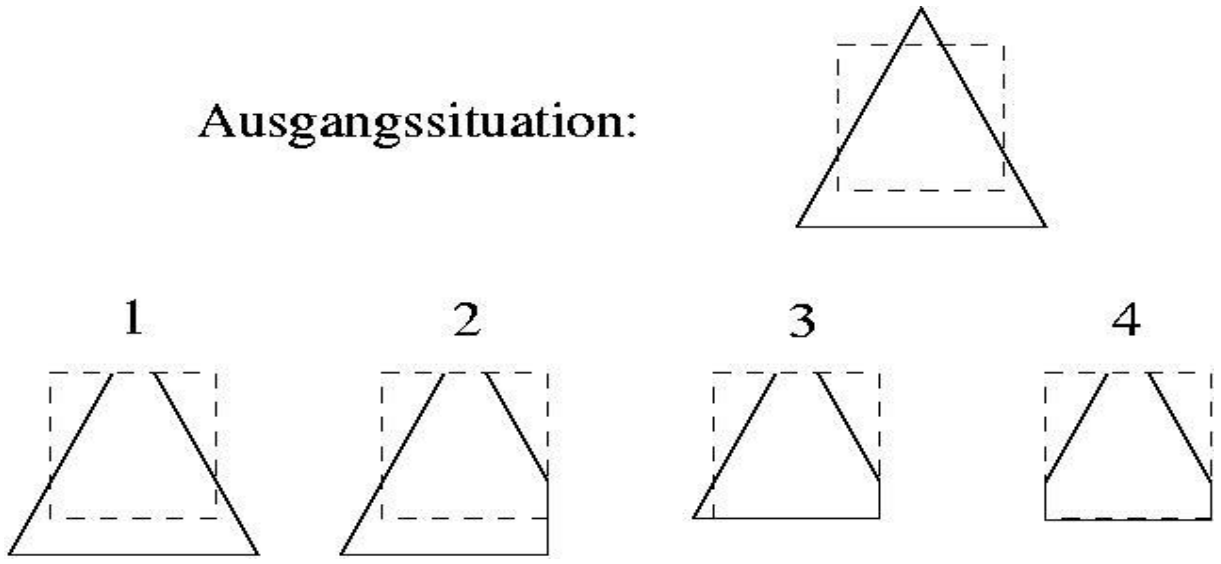
- Find intersection point S with an edge that lies between the two points
- Recursion: S replaces P_1 or P_2, respectively

See Blackboard for pseudo code of algorithm CS_Clip

Clipping of Polygons – Sutherland Hodgeman

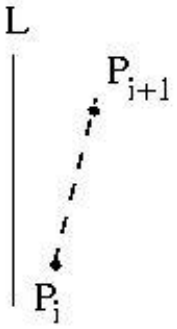
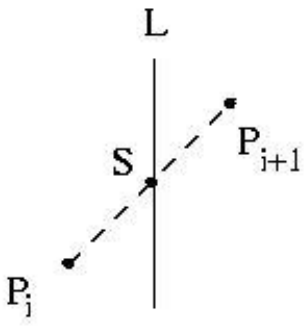
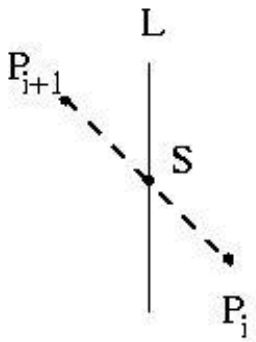
Just clipping of single edges does not work! Insert additional edges on the border of the rectangle!

Idea: Clip whole polygon against the first edge, then clip the resulting polygon against the second edge, ...



Clipping of Polygons – Sutherland Hodgeman (cont.)

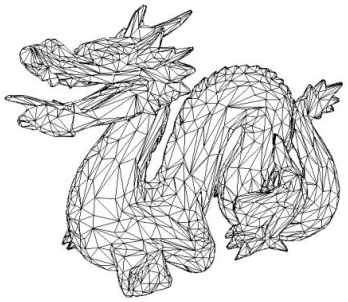
4 cases:

in –> out	out –> in	in –> out	out –> out
			
output(P_{i+1})	ermittle S output(S) output(P_{i+1})	ermittle S output(S)	kein Output

See Blackboard for pseudo code of algorithm

Pipelining:

Position in Rendering Pipeline



Transformation: position and orientation of objects

Transformation: WKS, SKS

Culling

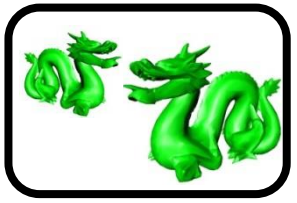
Projection

Clipping

Shading

Hidden Surface Removal

Scan Conversion



Topics

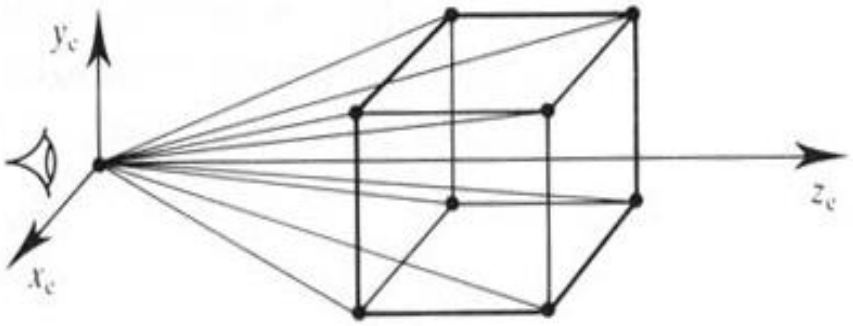
- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

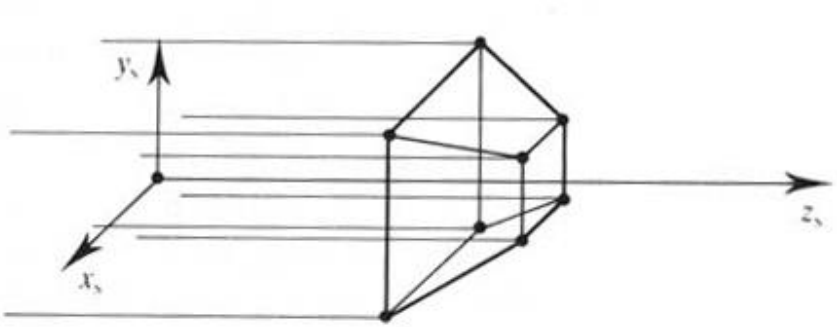
    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

Z-Buffering

Z-Buffer

- Special memory on graphics hardware, 1 entry for every pixel
- Updated with the highest z value of 3D object points that cover its pixel
- Accuracy depends on
  - Length of memory words („depth“) (usually 24 bits/pixel)
  - Z-value of front and back clipping plane
- Compare points lying on the same projector (OpenGL: parallel projection!)





Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

Light and Reflection

Local reflection (in real time CG):

- Interaction between object surfaces and light sources

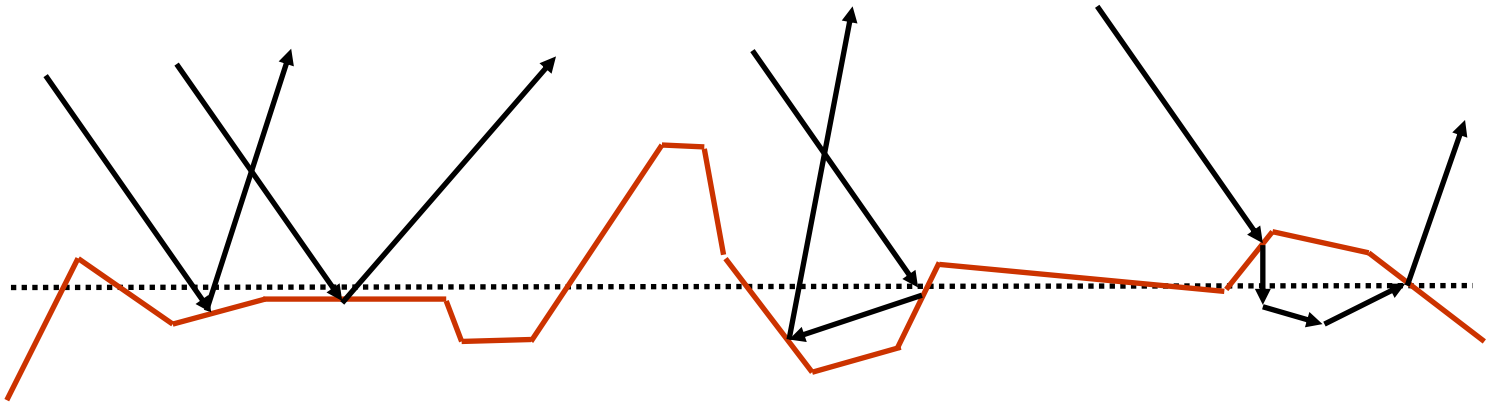
Global reflection (in photorealistic CG):

- Interaction between object surfaces and light sources
- Interaction between object surfaces and surfaces of other objects
- ➢ Ray Tracing (later in this lecture)
- ➢ Radiosity (see computer graphics lectures, e.g., Kobbelt – Informatik VIII)

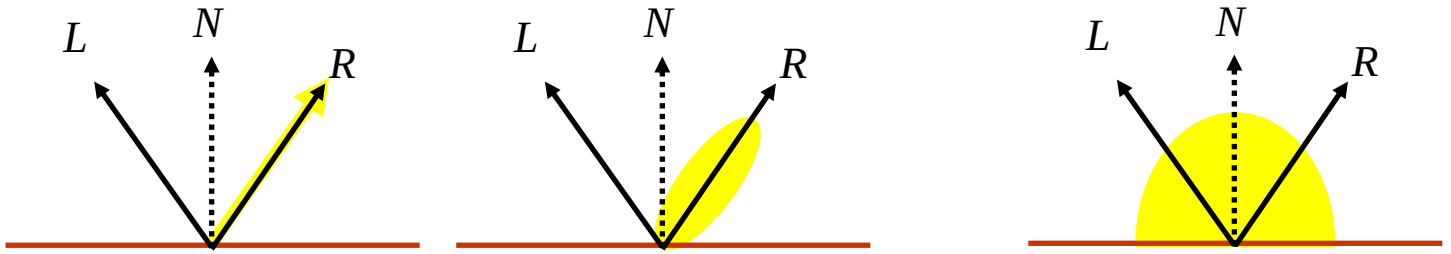
Variables:

- Angle of incoming light
- Surface material
- Position of viewer
- Characteristics of light source (wavelength, …)

Reflection on Surfaces



a) Direct reflection b) Multi-surface reflection c) Subsurface reflection



1) Mirror reflection
(perfect specular reflection)

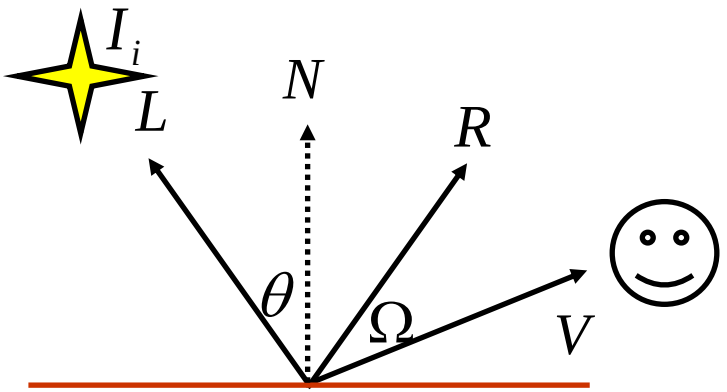
2) Directed diffuse reflection

3) Diffuse Reflection

Phong Reflection Model

Linear combination from 3 components:

- Diffuse
- Specular
- Ambient



Reflection coefficients: k_d, k_s, k_a

Diffuse component: $I_d = I_i k_d \cos\theta = I_i k_d (L \cdot N)$ $\quad I_d = k_d \sum_n I_{i,n} (L_n \cdot N)$

Specular component: $I_s = I_i k_s \cos^n \Omega = I_i k_s (R \cdot V)^n$

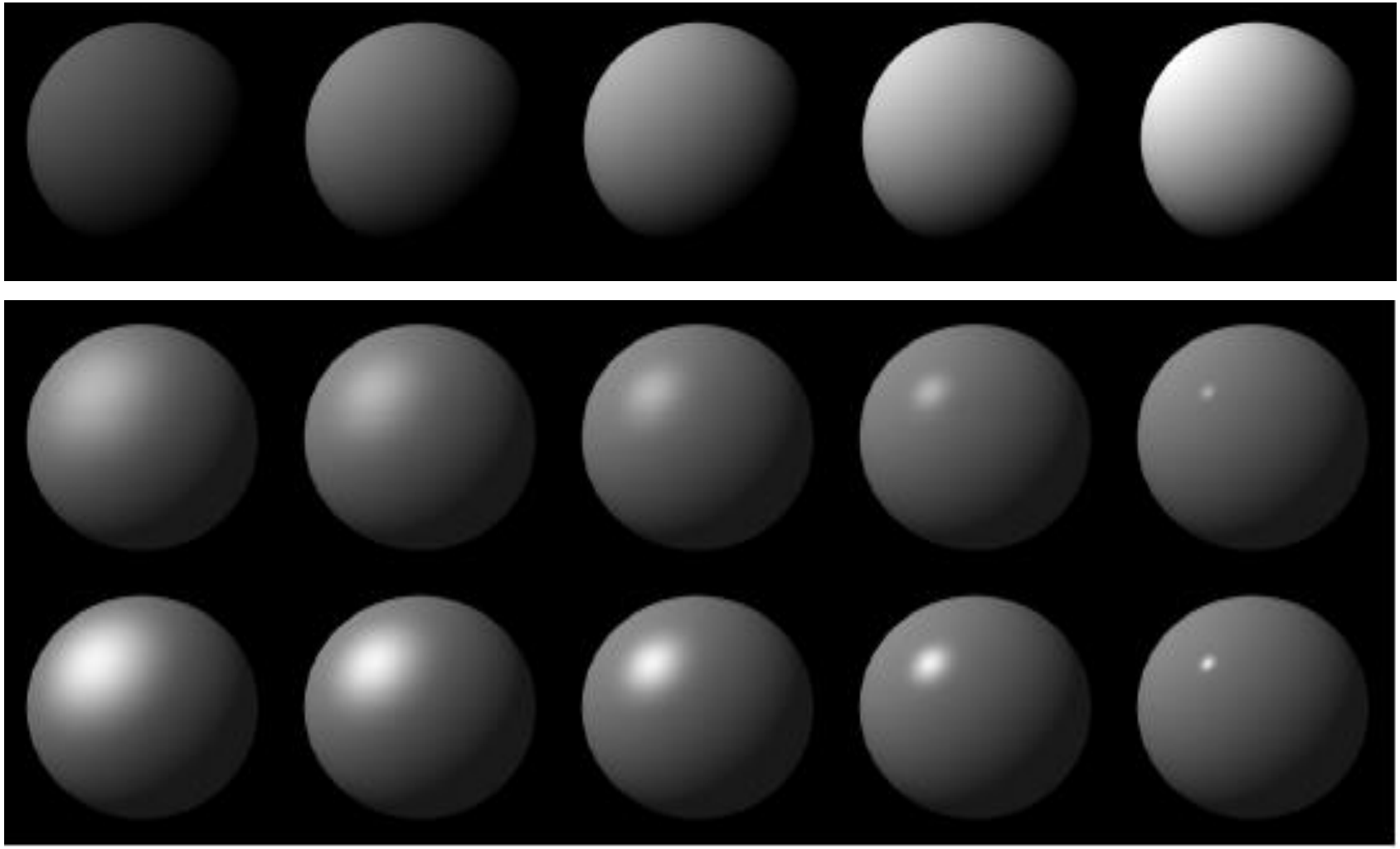
n: Index for surface roughness

Perfect mirror: $n \rightarrow \infty$ (Ray Tracing: Recursion)

Ambient component: $I_g = I_a k_a$

Overall intensity: $I = I_a k_a + I_i (k_d (L \cdot N) + k_s (R \cdot V)^n)$

Diffuse & Specular



Pictures: Westermann

Phong Reflection Model - Optimization

Problem:

- Calculation of R
- Simple and efficient, has to be done many times, however

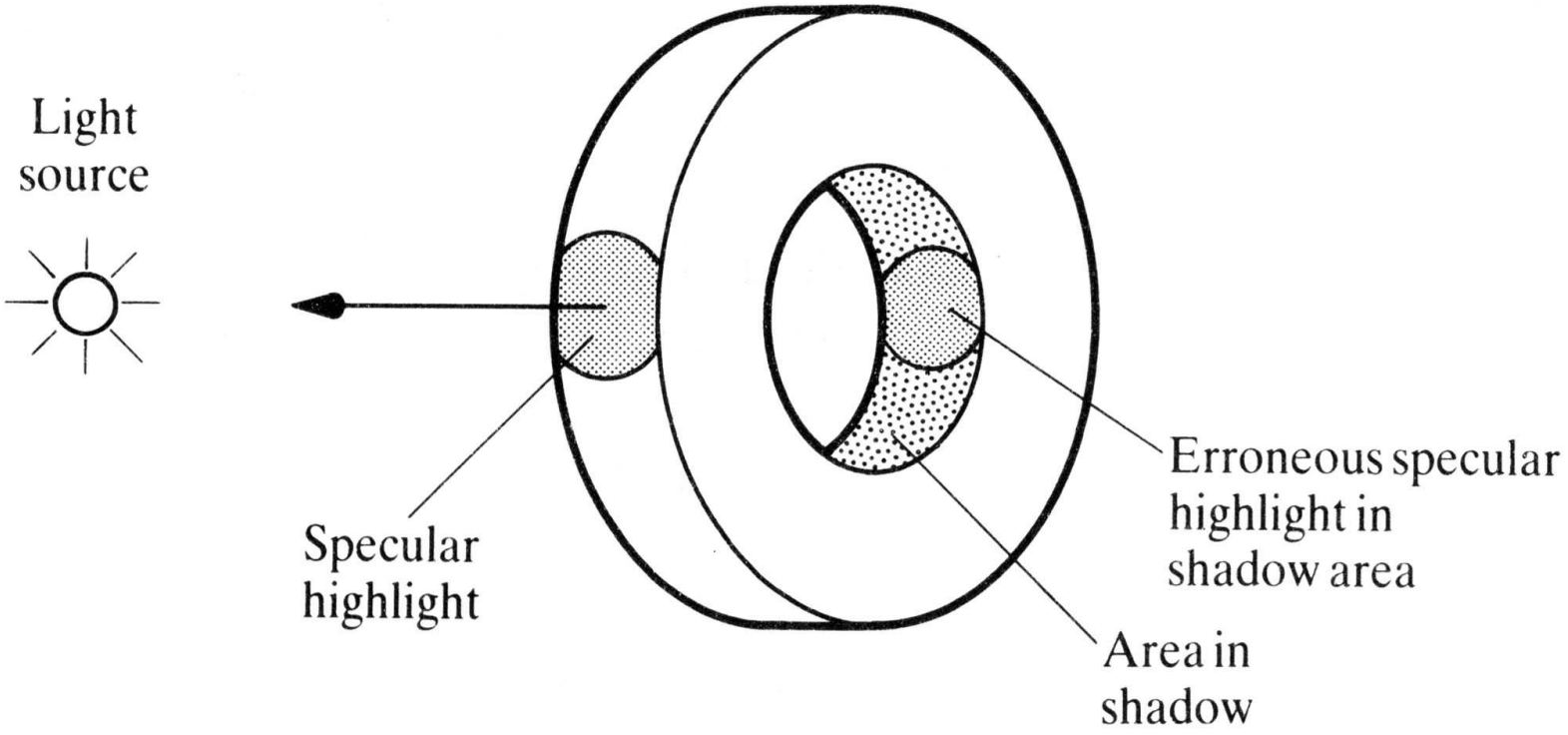
Approximation:

- Suppose light source and viewpoint to be at infinity
  - *L* and *V* are constant throughout the scene (not over time!)
- Replace RV by NH with $H = \frac{L+V}{2}$ (see literature for details)
  - *I* only depends on *N*

Phong Reflection Model - Simplifications

- Light sources are points, no intensity distribution
- Light sources and viewer are located at infinity
- Diffuse and specular components are local
  - Ambient light
  - Objects are somewhat plastic
  - No shadows
- Empirical (no physical) model, especially for the specular component
- Color of specular component = color of light source

Phong Reflection Model - Errors



Picture: Watt

What is Shading?

Shading

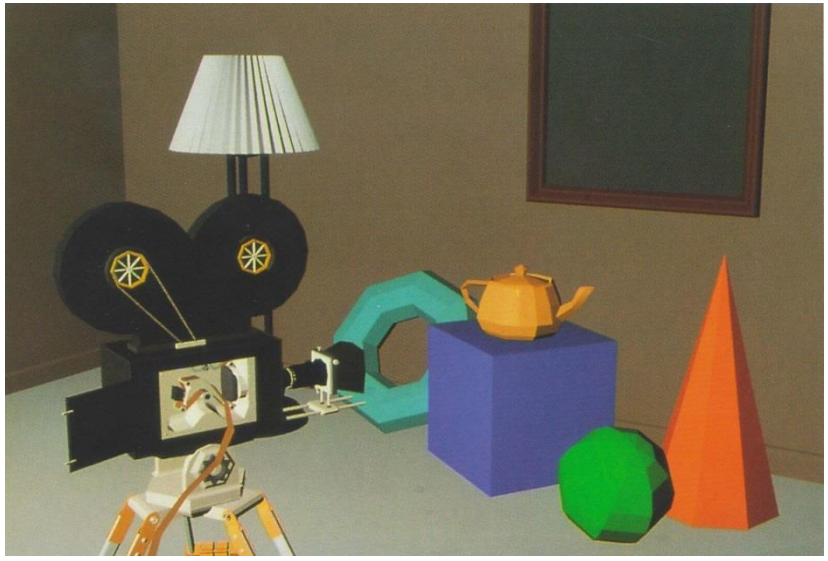
Definition: (Incremental, interpolative)
Application of a reflection model on polygons by calculation of intensities at polygon vertices and interpolation of these values for the inner points

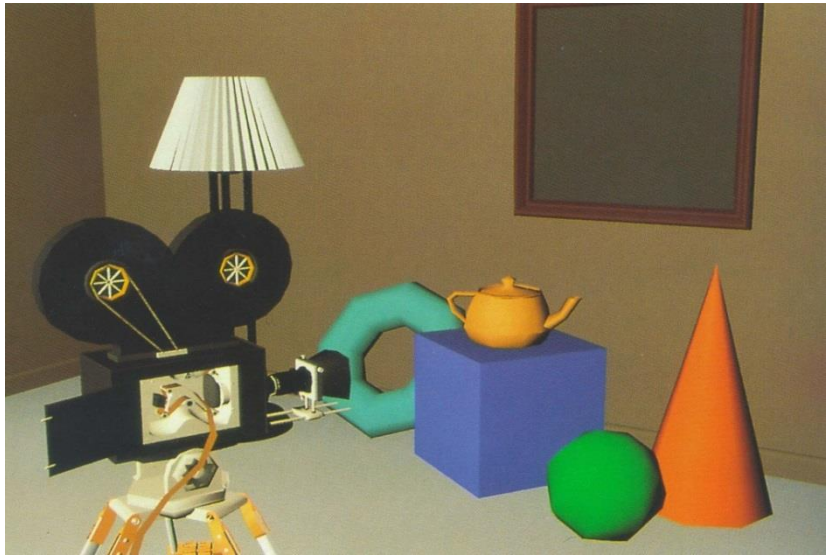
CG: Phong reflection model

1. Flat Shading
2. Gouraud Shading
3. Phong Shading

Flat Shading and Gouraud Shading

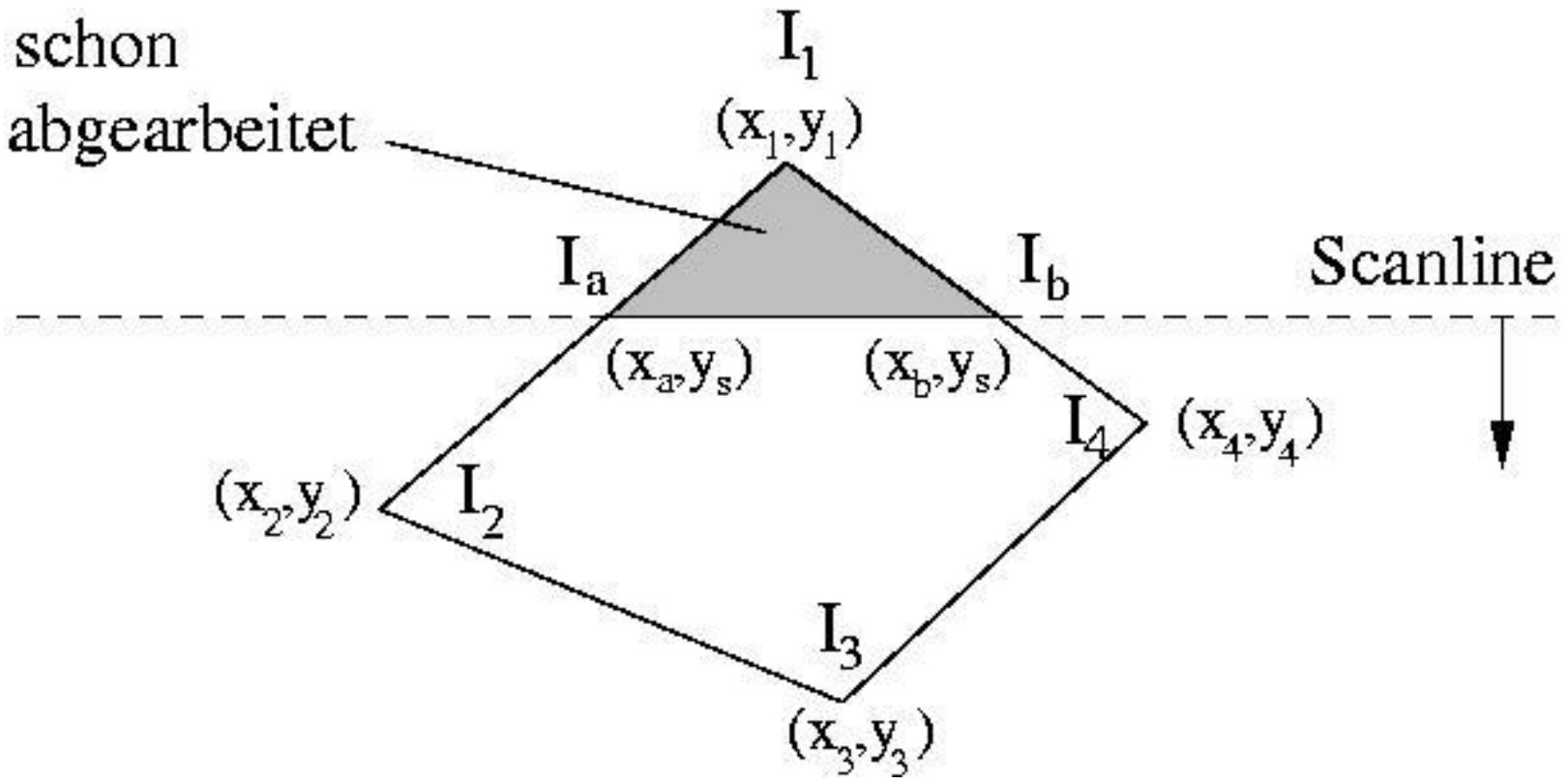
- **Flat Shading**
  No interpolation, all inner points of a polygon get the same intensity
- **Gouraud Shading**
  (Bilinear) interpolation of the inner points from the vertex intensities





Pictures: Foley et al.

Gouraud Shading – Bilinear Interpolation



Gouraud Shading – Diffuse Component

- Calculate the vertex normal vector as average from the adjacent polygon normal vectors (offline!)
- Calculate the vertex intensities according to Phong model
- Interpolation process (integrated in scan conversion) in scan line order
  - Interpolate intensities at the intersection points of scan line and polygon edges from the vertex intensities

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

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

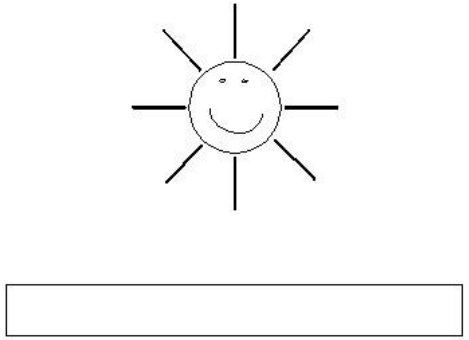
  - Interpolate intensities for the inner points along the scan line from the intensities at the intersection points

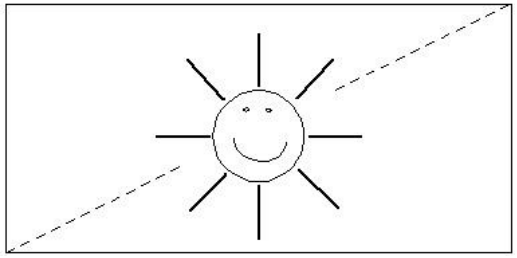
$$I_s = \frac{1}{x_b - x_a}\left(I_a\left(x_b - x_s\right) + I_b\left(x_s - x_a\right)\right)$$

Has to be calculated for every pixel: incremental approach, see literature

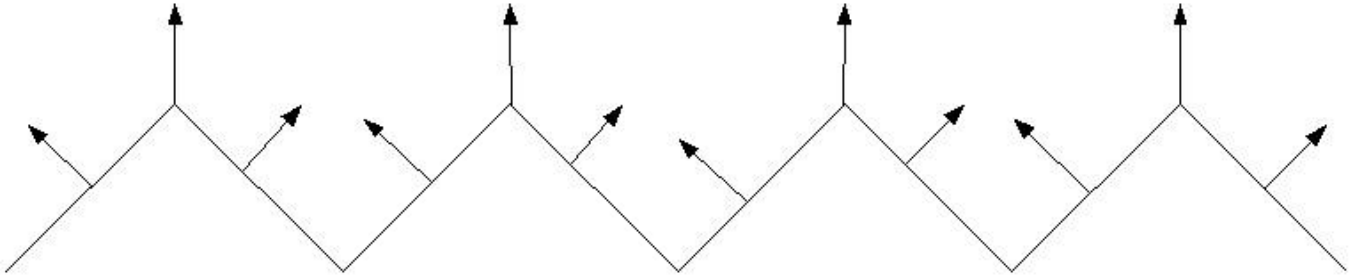
Gouraud Shading – Some Problems

- No highlights in the middle of polygons (example: virtual table)



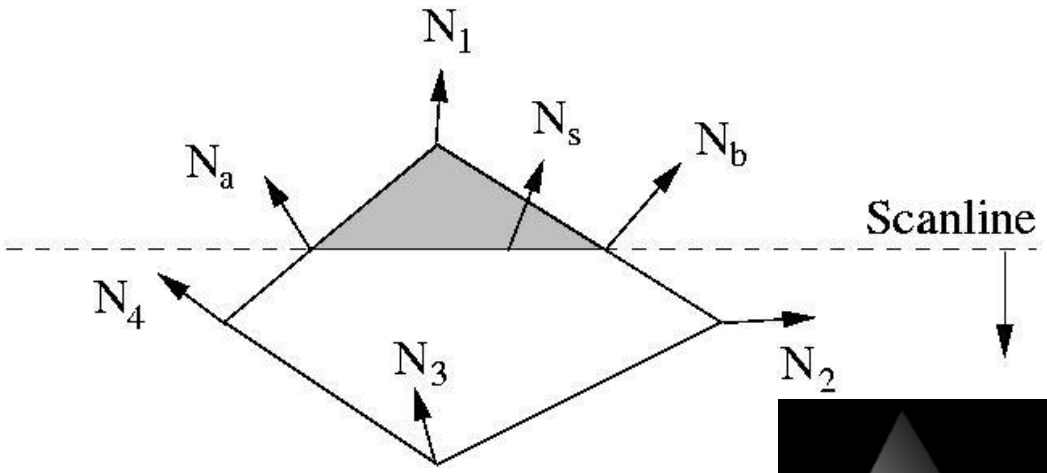


- Corrugations are smoothed out

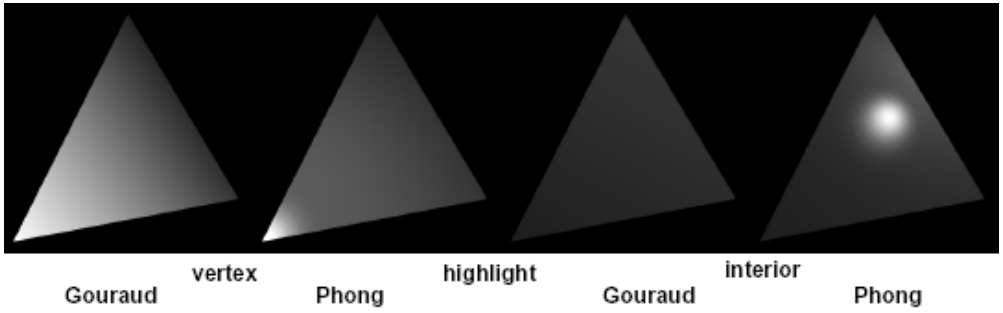


Phong Shading

- Interpolation of vertex normal vectors instead of intensities
- Get an individual normal vector for each pixel as an approximation of the “real” normal vector on a curved surface



Picture: Westermann



Phong Shading – Interpolation of Normal Vectors

$$N_a = \frac{1}{y_1 - y_2}\left(N_1(y_s - y_2) + N_2(y_1 - y_s)\right)$$

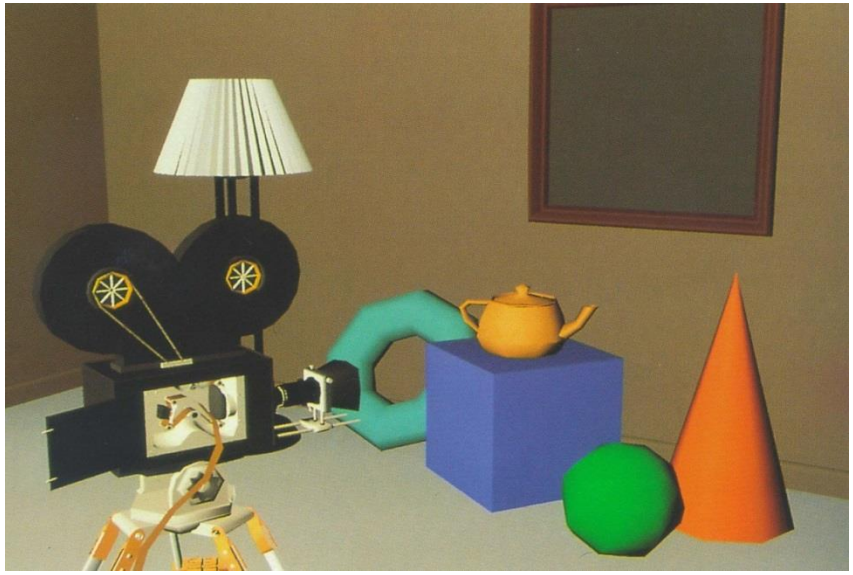
$$N_b = \frac{1}{y_1 - y_4}\left(N_1(y_s - y_4) + N_4(y_1 - y_s)\right)$$

$$N_s = \frac{1}{x_b - x_a}\left(N_a(x_b - x_s) + N_b(x_s - x_a)\right)$$

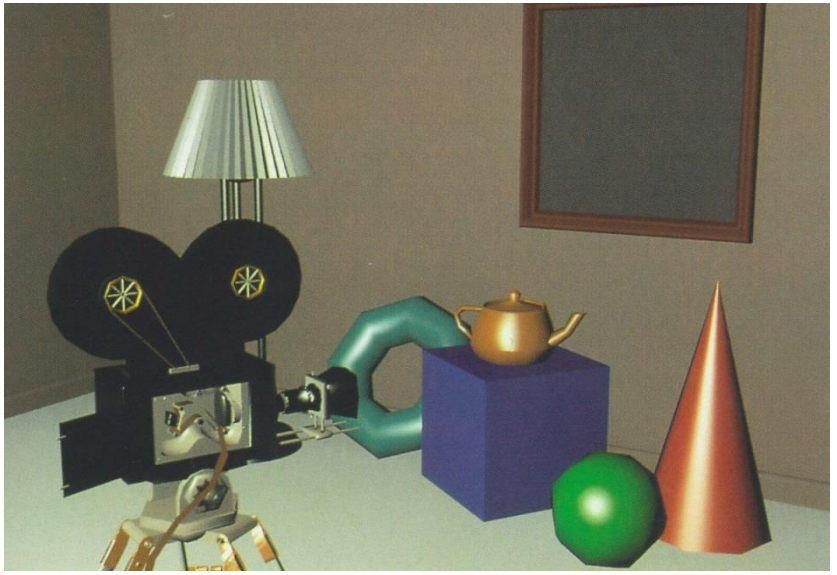
- Phong: three times more complex than Gouraud
- Also: Calculate intensity at each pixel
- Speed up: Combine Gouraud and Phong Shading (Interpolate normal vector for every second pixel and interpolate intensity for the other pixels)
- Phong shading never realized in graphics hardware (?) – can be simulated with textures

Gouraud Shading versus Phong Shading

Simple Gouraud Shading



Phong Shading
with Specular Highlights



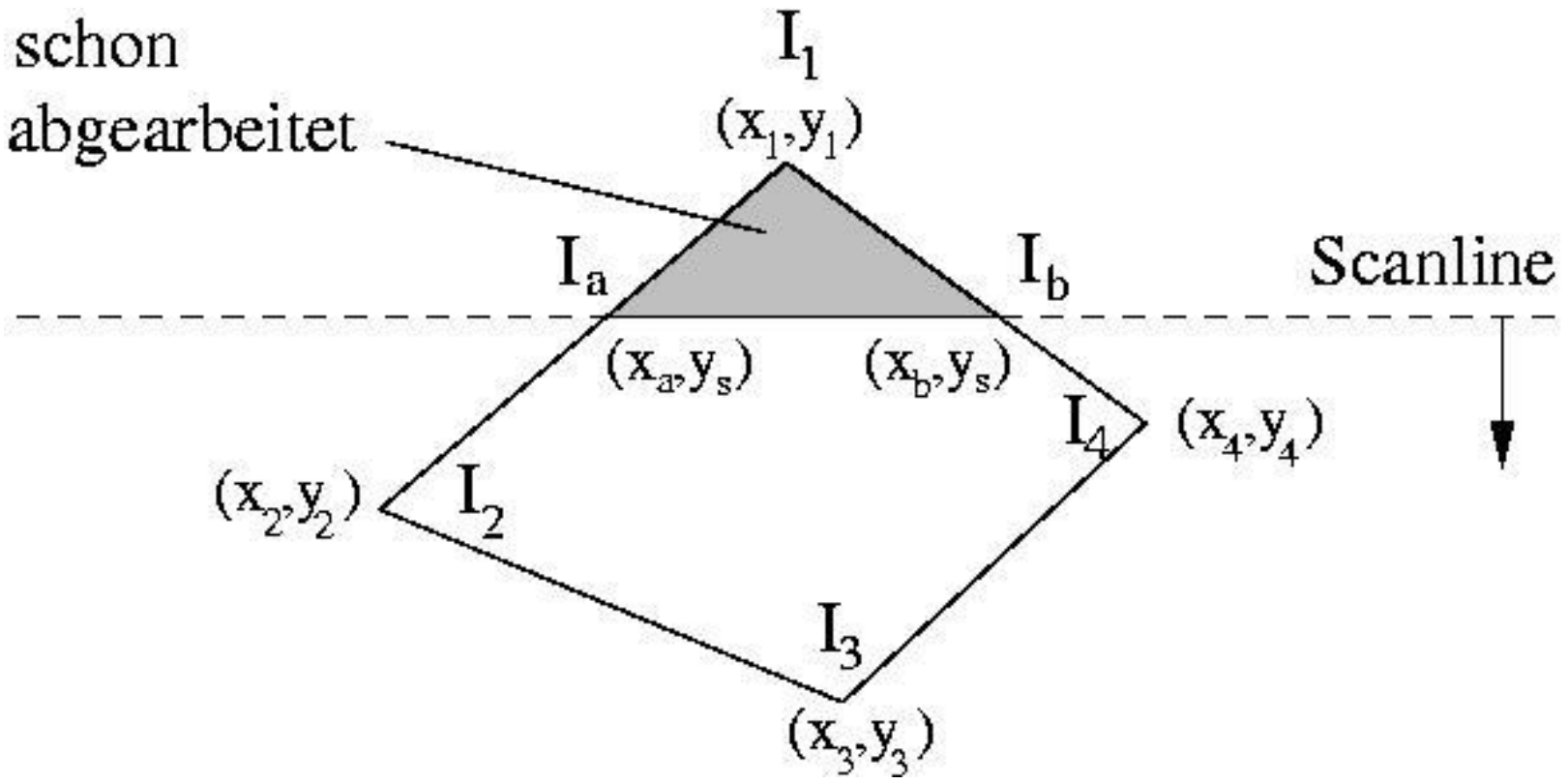
Pictures: Foley et al.

Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

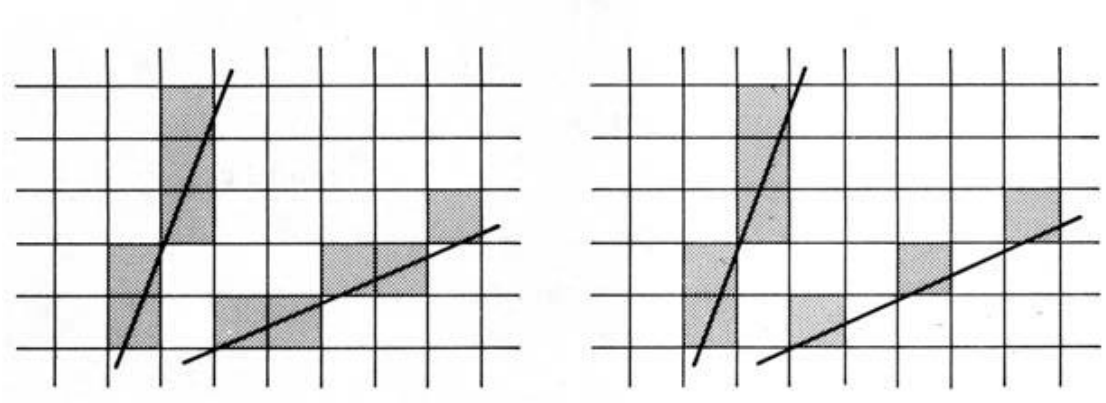
    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

Gouraud Shading – Bilinear Interpolation



DDA Algorithm

- Scan conversion or rasterization determines the actual pixels of the view window for which intensity values are required.
- Rasterization of solid objects:
  Polygon fill via horizontal line segments (“scanline order”)
  Required: Right/Left border of a segment, i.e. the intersection point of scanline and polygon edges (“Digital Differential Analyzer”)



Picture: Watt

/* Let (x_s, y_s), (x_e, y_e) be start and end vertex of a polygon edge, $y_e > y_s$ */

$x := x_s$
$m := (x_e - x_s)/(y_e - y_s)$
For $y := y_s$ to y_e do
    output $(round(x), y)$
    $x := x + m$

Rasterization of Lines – Brute Force Approach

Rendering of lines which are non-parallel to the screen axes:

- Which pixels should be set?
- “Wireframe”: Avoid gaps!

Simple strategy: Use DDA algorithm

- Follow the line horizontally or vertically, according to slope
- Set the pixel closest to the line

$$P_1 = (x_1, y_1),\ P_2 = (x_2, y_2)$$

$$x_1 \le x_2,\ y_1 \le y_2,\ x_i, y_i \qquad \Delta y := y_2 - y_1 \le x_2 - x_1 =: \Delta x$$

No gaps

$$0 \le m = \frac{\Delta y}{\Delta x} \le 1$$

The Bresenham Algorithm - Idea

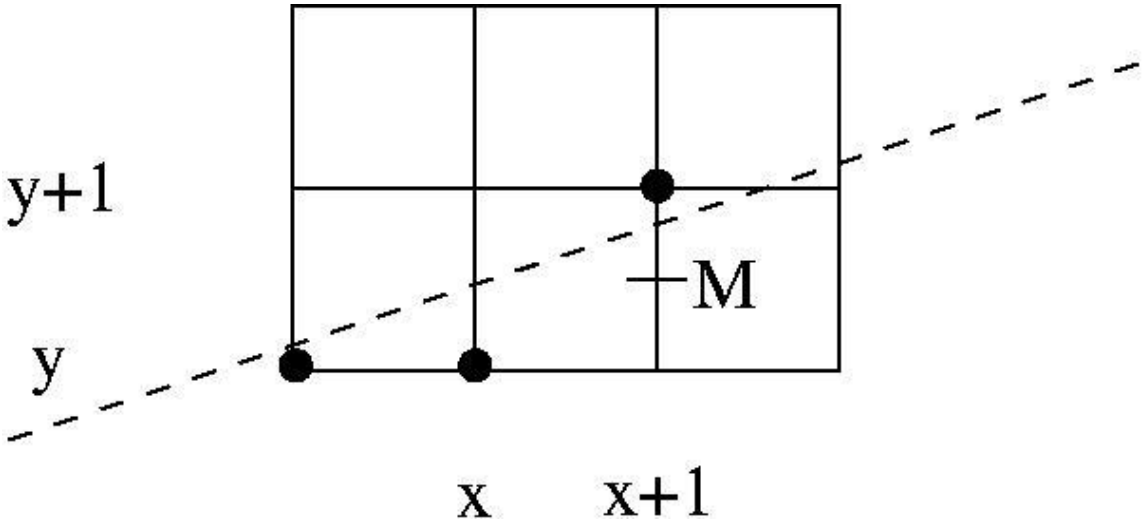
- Goal:
  Replace Float by Integer operations (faster)
- Idea:
  After pixel (x, y) has been set, there are 2 possibilities for the next step:
  - E (East) case: Set pixel (x+1, y)
  - NE (North East) case : Set pixel (x+1, y+1)
- Let the sign of a decision variable d decide about E or NE
- Incrementally modify d from step to step

The Bresenham Algorithm – Decision Variable

Line equation: $$y = \frac{\Delta y}{\Delta x}(x - x_1) + y_1$$

Implicit: $$F(x, y) = 0 \text{ mit } F(x, y) = \frac{\Delta y}{\Delta x}(x - x_1) - (y - y_1)$$

Decision variable d: $$F\left(x + 1, y + \frac{1}{2}\right) =: F(M)$$



The Bresenham Algorithm – Initialization

Initialize *d*:

$$d = F\left(x_1 + 1, y_1 + \frac{1}{2}\right) = \frac{\Delta y}{\Delta x} - \frac{1}{2} \quad |\cdot 2\Delta x \text{ (does not change the sign of } d)$$

$$\Rightarrow d = 2\Delta y - \Delta x \quad \text{INTEGER!}$$

The Bresenham Algorithm – East and North East

Two cases for M:

- M is below the line $\Rightarrow d > 0\ (NE)$

$$x := x+1, y := y+1$$

$$d_{NEW} := F\left(x+2, y+\frac{3}{2}\right) = \frac{\Delta y}{\Delta x}(x+2-x_1) - \left(y+\frac{3}{2}-y_1\right) =$$

$$\frac{\Delta y}{\Delta x}(x+1-x_1) + \frac{\Delta y}{\Delta x} - \left(y+\frac{1}{2}-y_1\right) - 1 = d_{OLD} + \frac{\Delta y}{\Delta x} - 1$$

- M is above the line $\Rightarrow d \leq 0\ (E)$

$$x := x+1, y := y$$

$$d_{NEW} := F\left(x+2, y+\frac{1}{2}\right) = d_{OLD} + \frac{\Delta y}{\Delta x}$$

The Bresenham Algorithm – Pseude Code

/* Let (x_s, y_s) ,(x_e, y_e) be start and end vertex of a polygon edge, $y_e > y_s$ /*

$x := x_1$,y $:= y$

$\Delta x := x_2 - x_1, \Delta y := y_2 - y_1$

$d := 2\Delta y - \Delta x$

$incrE := 2\Delta y$ /* Δy / Δx multiplied with 2 Δx */

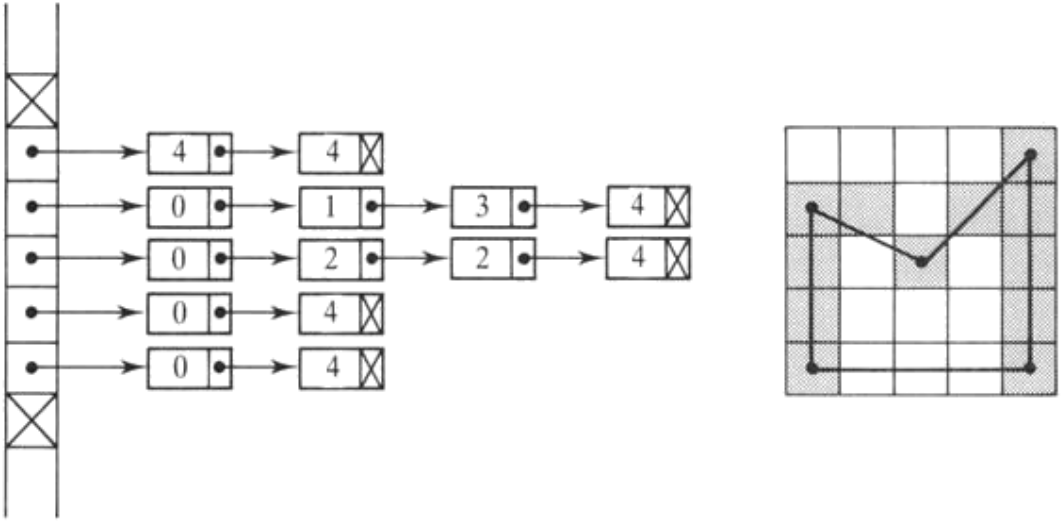
$incrNE := 2(\Delta y - \Delta x)$ /* Δy / Δx -1 multiplied with 2 Δx */

The Bresenham Algorithm – Pseude Code (cont.)

While $(x < x_2)$
    $x := x + 1$
    if $d > 0$ then
        $y := y + 1$
        $d := d + incrNE$
    else
        $d := d + incrE$
    endif
    output(x, y)
endwhile

Polygon Edge List

- Data structure:
  Edge list for each polygon by an array of linked lists, one element for each scan line



- Closed polygons: Even number of list elements
- Also works for concave polygons with holes
- Insertion sort: Sort while inserting elements into the list
- Convex polygons: 2 entries per array element only

Picture: Watt

Rendering (Pixel) Loop

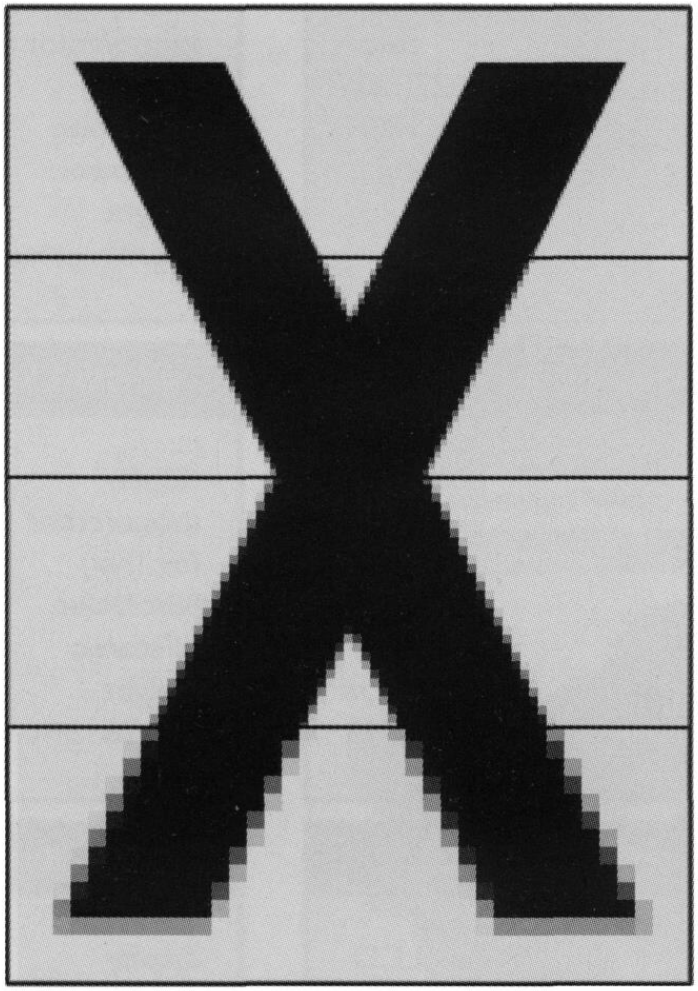
For every polygon
  For each edge (x_s, y_s), (x_e, y_e) of the polygon /* $y_e > y_s$ */
    $x := x_s$ /*DDA */
    $m := \frac{x_e - x_s}{y_e - y_s}$
    For $y := y_s$ to y_e do
      insert $round(x)$ into linear list $A[y]$ /* insertion sort */
      $x := x + m$
    endfor
  endfor
  For $y := y_{min}$ to y_{max} do
    For every pair (x_i, x_{i+1}) in $A[y]$ do
      if z-value(x, y) > z-buffer(x, y) then
        update (z-buffer)
        $I_d = k_d \sum_n I_{i,n} (L_n \cdot N)$ /* diffuse illumination, explicit */
        set_framebuffer (x, y, I_d)
      endif
    endfor
  endfor
endfor

Rendering Loop

Remarks:

- Rendering polygon per polygon
- Arbitrary order of polygons
- z-buffering recommended
- Edges shared by two polygons are rendered twice
- Alternative: Scanline order rendering (see literature for details)

Display Resolution



Today:

- SXGA, 1280x1024
- UXGA, 1600x1200
- WUXGA, 1920x1200

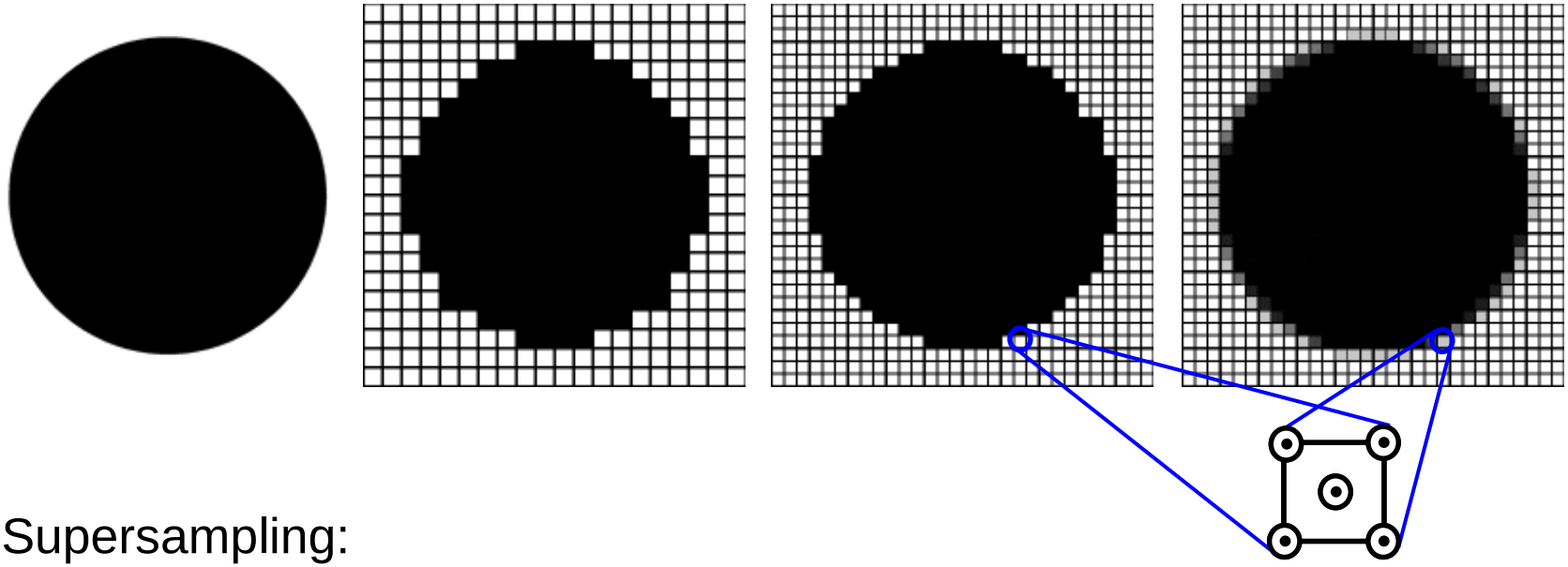
See also:

- Color depth
- Anti-Aliasing

Picture: Burdea et al.

Optimization - Quality

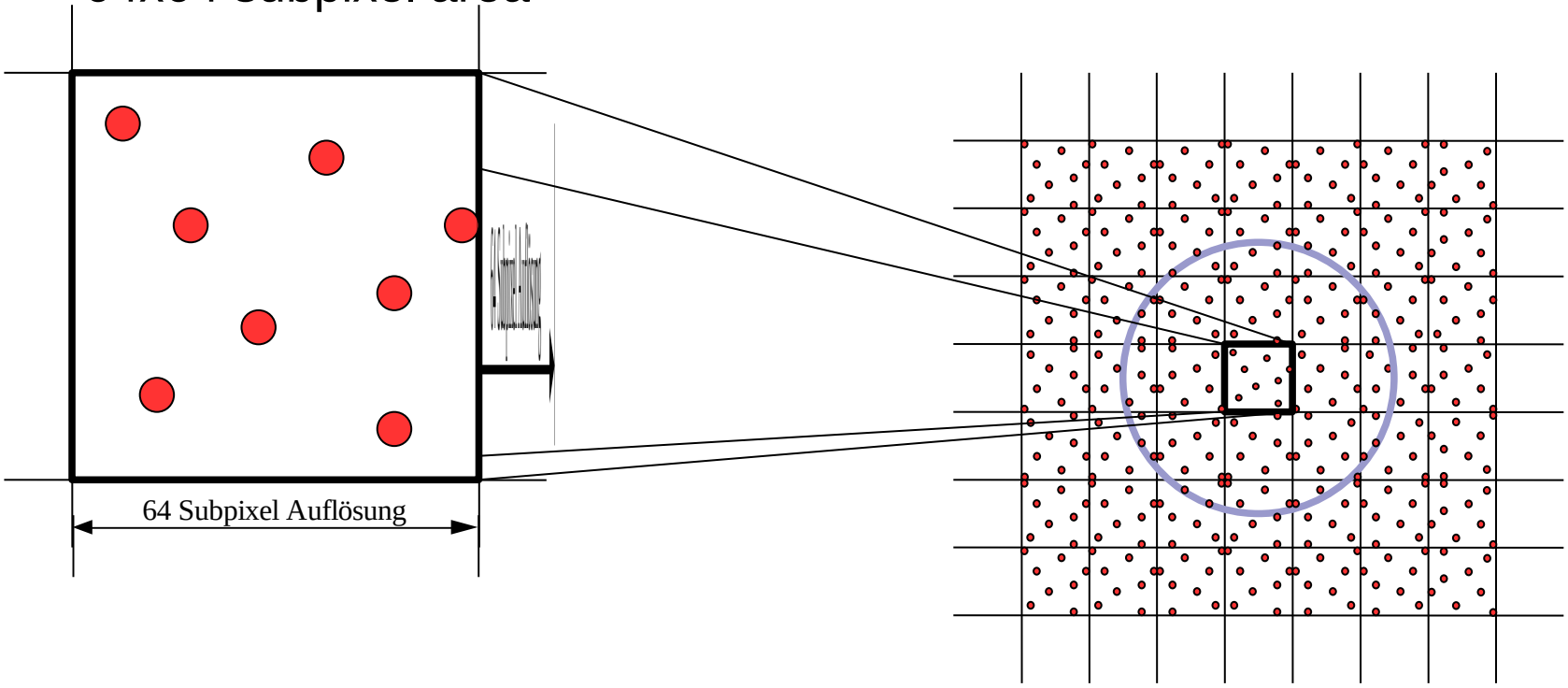
- Anti-Aliasing



- Supersampling:
  - Treat pixels as a plane
  - Send more rays through each pixel
  - Averaging the intensities of single rays leads to pixel color

Multisampling on XVR 4000

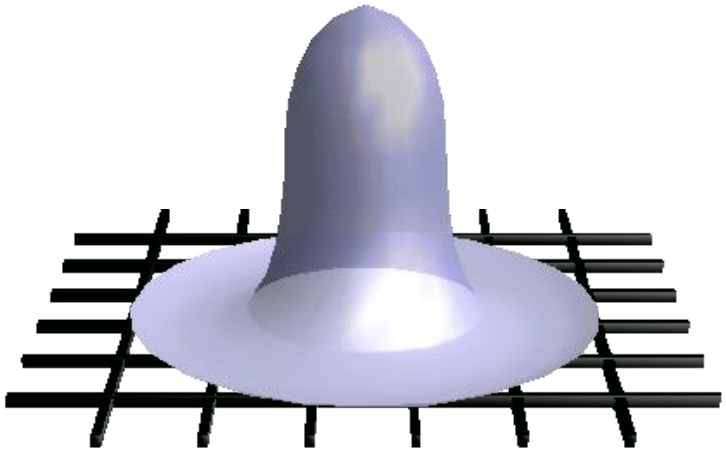
1. 1-16 Samples, random distribution that changes with each pixel, calculated in 64x64 subpixel area
2. Samples are collected and filtered within an area shaped as circle and 5x5 pixels large



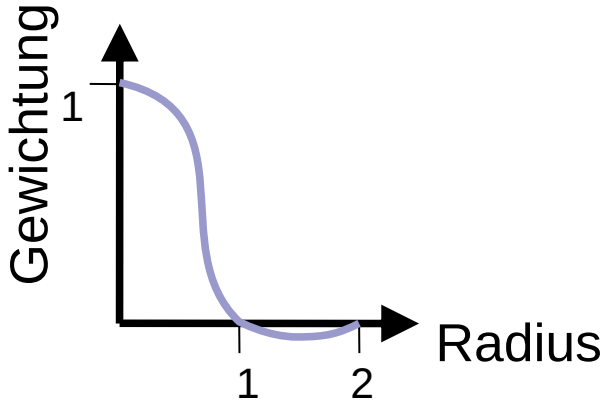
Courtesy of C. Gonzalez, Sun Microsystems

Multisampling on XVR 4000 (cont.)

3. All samples within the circle are weighted, normalized and accumulated

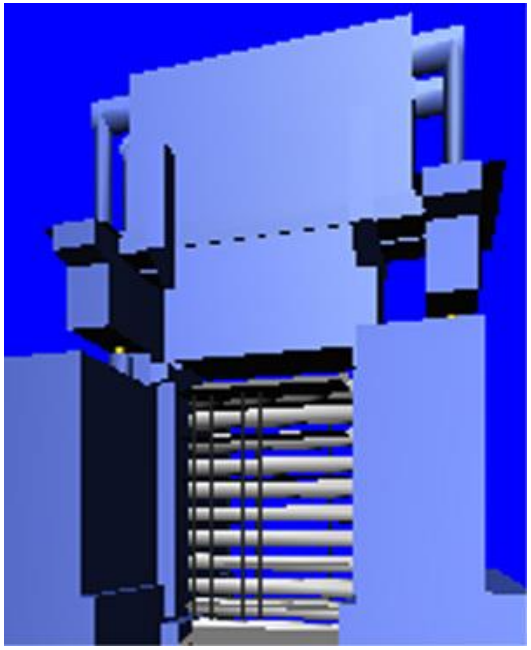


4. The filter curve is programmable. User can adapt filter characteristics to the application.

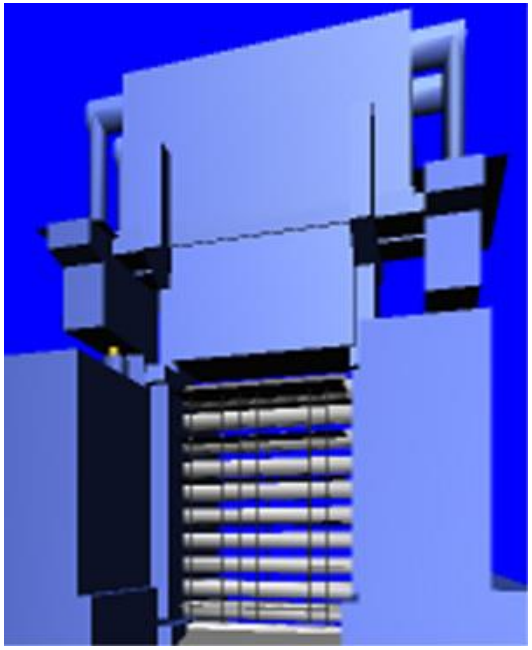


Courtesy of C. Gonzalez, Sun Microsystems

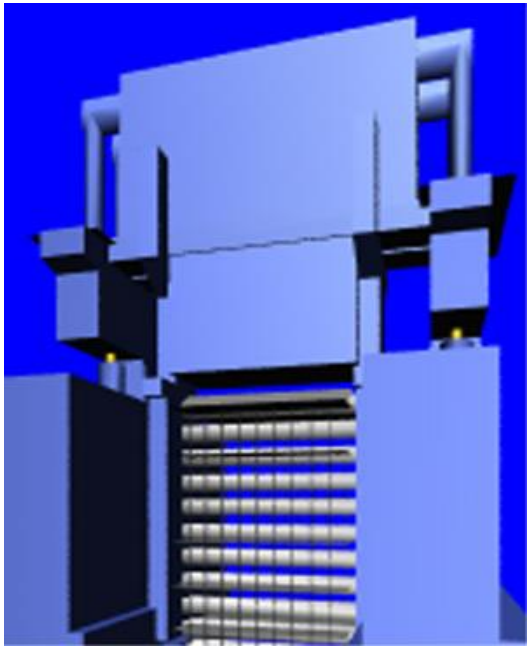
Anti-Aliasing



1 sample



4 samples



16 samples

Pictures: IT Center RWTH

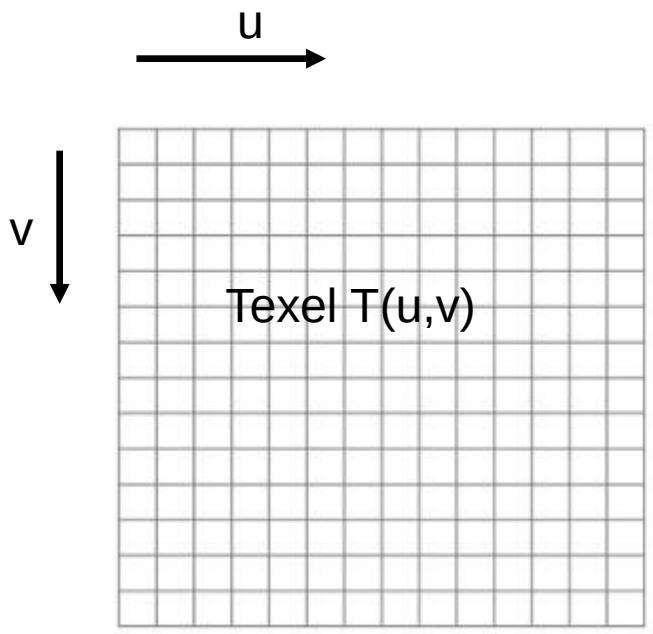
Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

What is Texture Mapping?

Texture Mappping: Usage of textures in CG (Catmull 1974)



Texture Map

Parameter Modulation:

- Color
  - Photo
  - Functional or procedural
  - Randomized
- Specular and diffuse component (Environment Mapping)
- Transparency (α- Blending)
- Perturbation of the normal vector (Bump Mapping)

Motivation

Increase realism of virtual scenes:

- Integrate real photos into the virtual scene
- Render the structure of surfaces without increasing geometry complexity
- Simulate mirror reflections without ray tracing
- Render semi-transparent surfaces (e.g., glass)

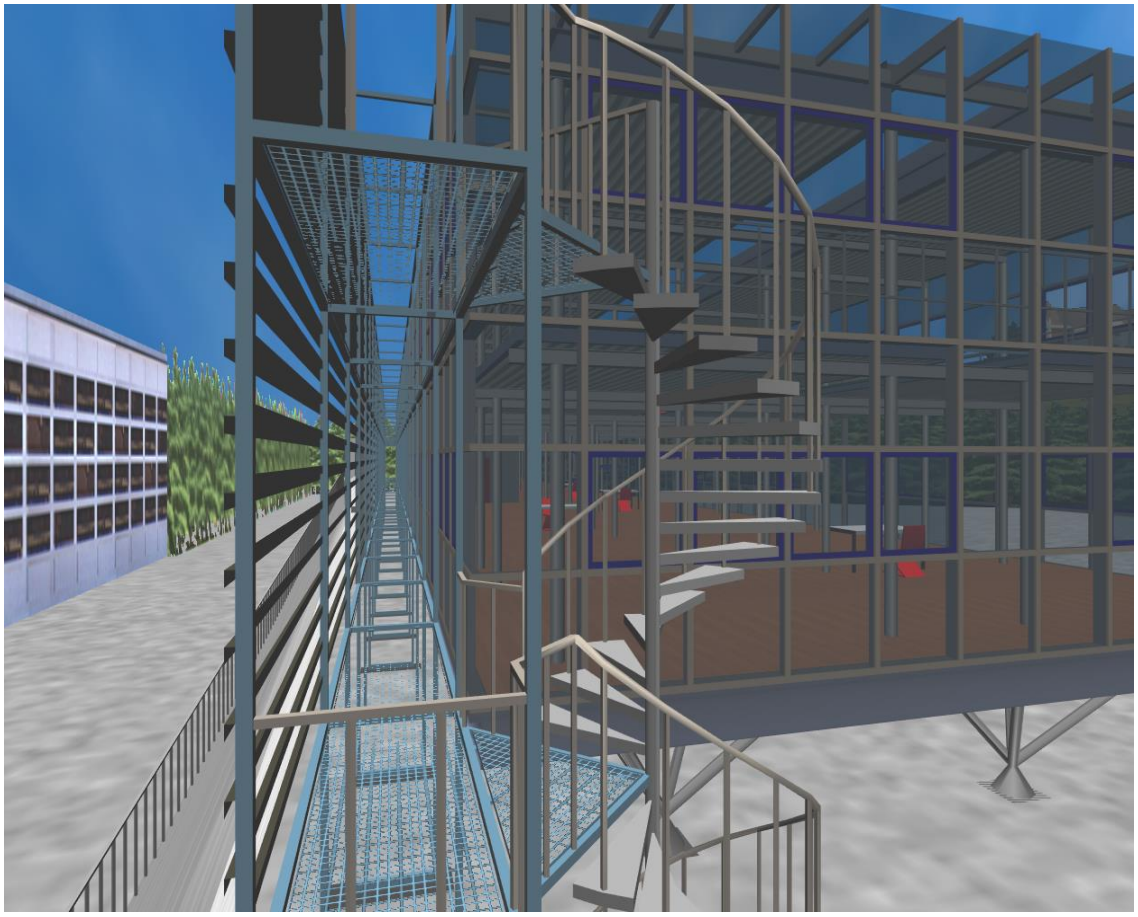
Scientific Visualization

- LIC (linear integral convolution)
- Imaging in medicine: Render volume by textures

Modulation of Color

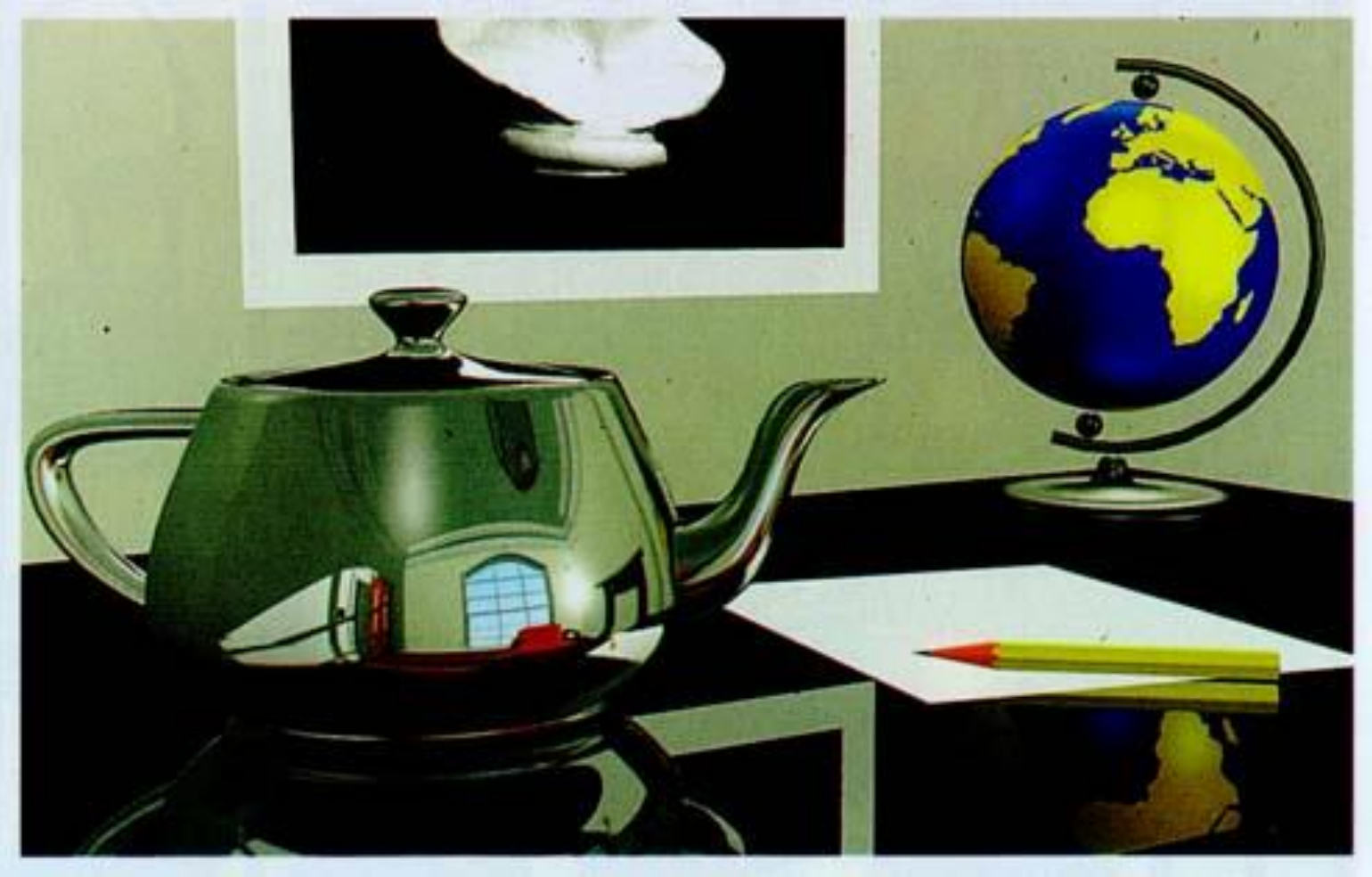
Picture: Foley et al.

Alpha Blending



Virtual building at Schinkel Straße, RWTH Aachen University
(Generated by IT Center (former RZ) in 1998)

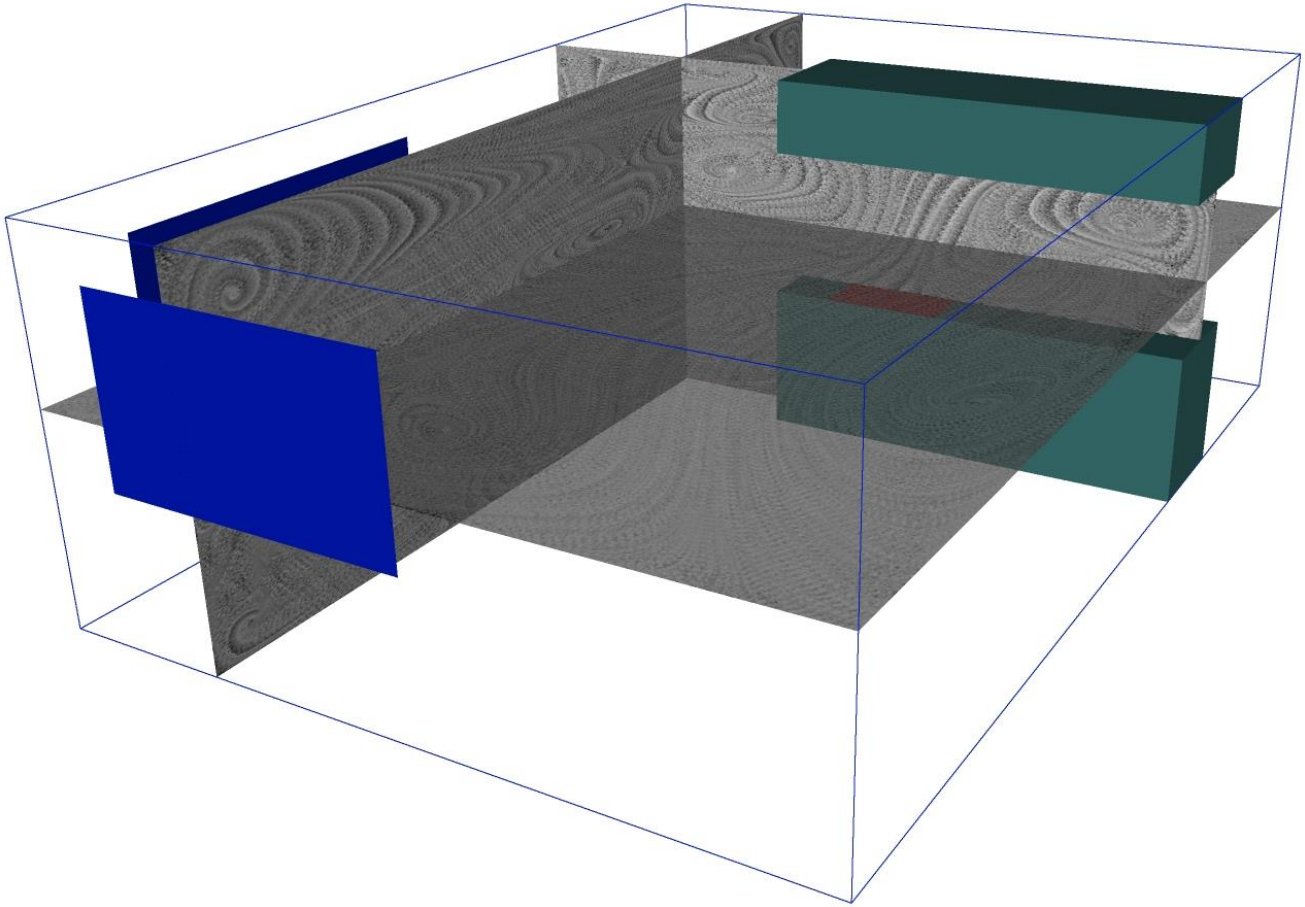
Environment Mapping



Picture: Watt

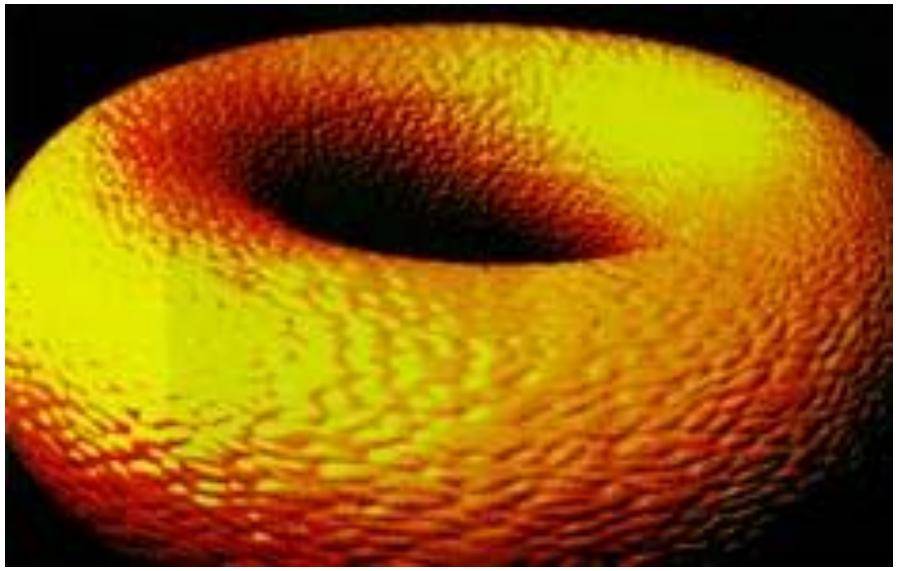
Linear Integral Convolution

Picture: IT Center RWTH



Simulation of air flow within a kitchen
(here: faster method without convolution – “integrate-and-draw”)

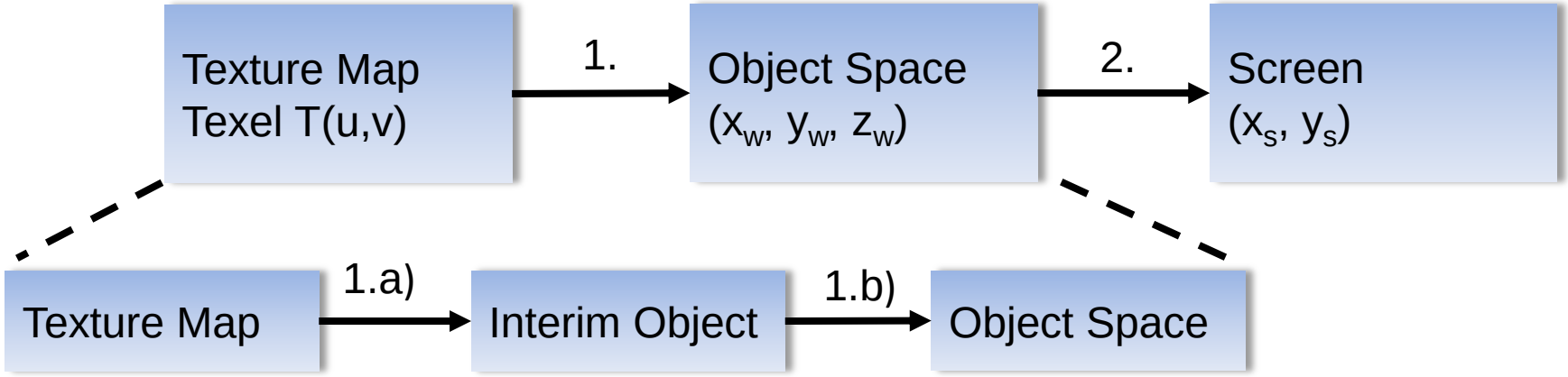
Bump Mapping



Pictures: Watt

From Texture Map to Pixels

1. Surface parametrization
2. Projection (interpolation, see Shading)



Use 2-phase parametrization to avoid distortions for polygon models:

a) S-Mapping

b) O-Mapping

S-Mapping

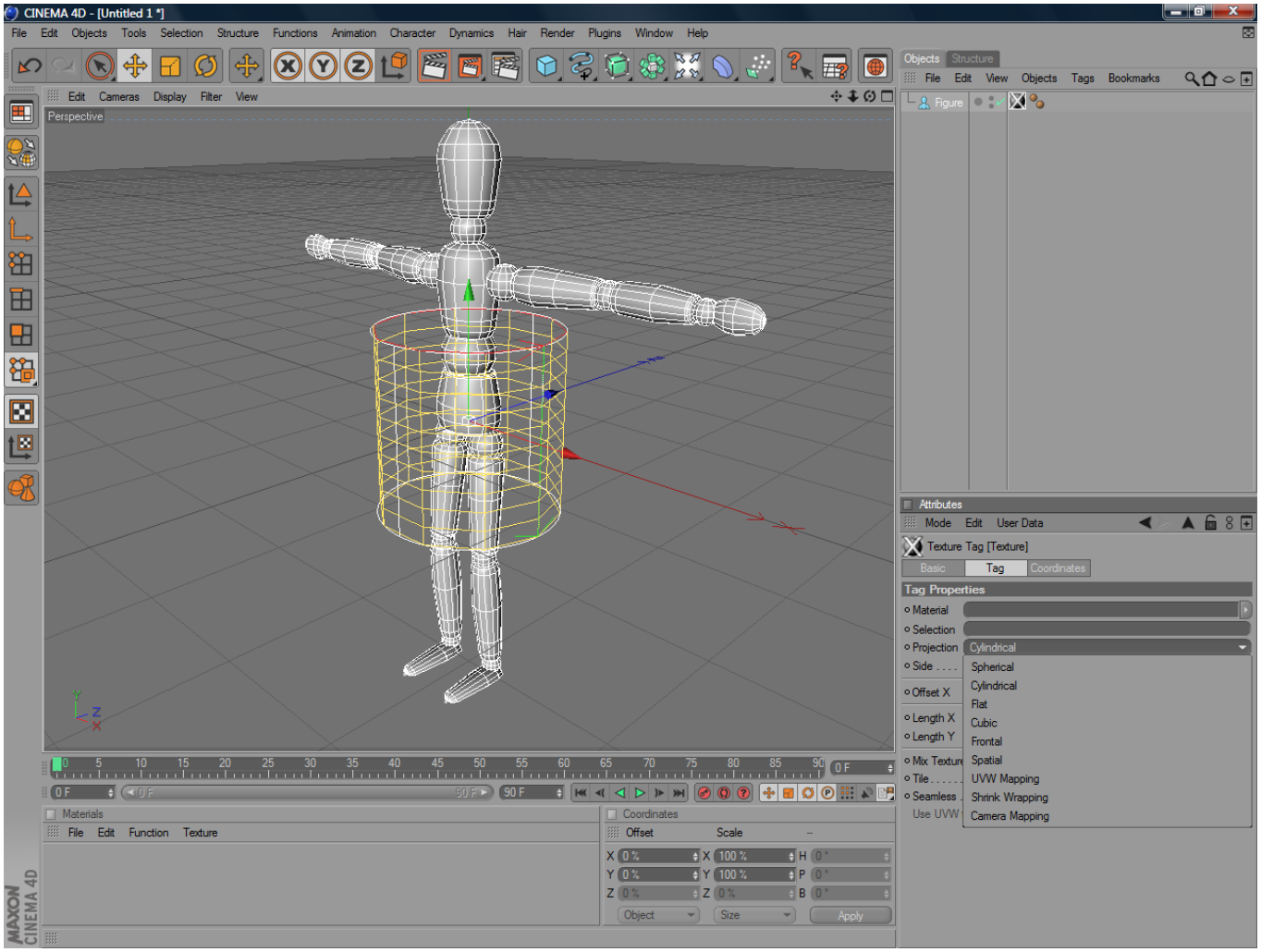
$T(u,v) \longrightarrow T'(x_i, y_i, z_i)$: „S-Mapping“

Interim objects

- enclose the virtual object
- simple mapping functions

- Cylinder (without top and bottom sides)
- Sphere
- Cube (No distortion, but clipping necessary)

Texture Mapping User Interface in Cinema4D



S-Mapping Example: Cylinder

$$S_{Cylinder} : (\Theta, h) \rightarrow (u, v) = \left(\frac{r}{c}(\Theta - \Theta_0), \frac{1}{d}(h - h_0) \right), \text{ with}$$

Θ_0, h_0 : Texture position

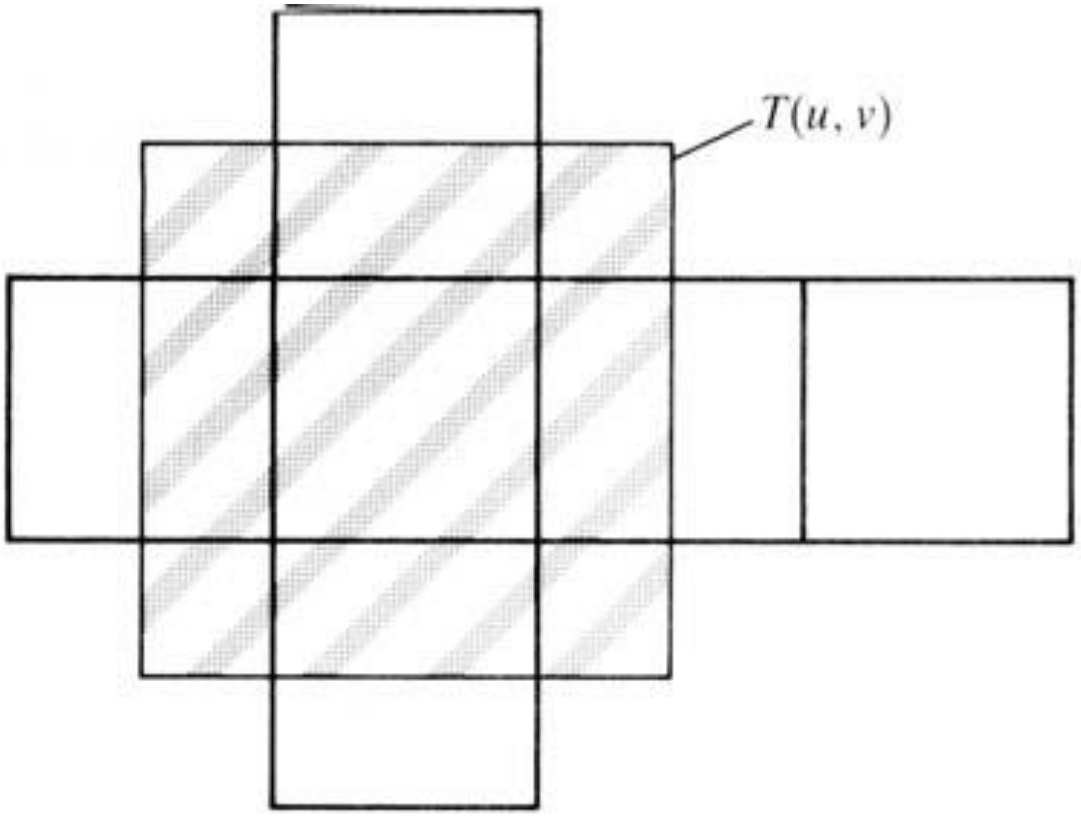
c, d : scaling factors

r : Cylinder radius

$\Theta : [0, 2\Pi]$

$h : [0, \text{cylinder height}]$

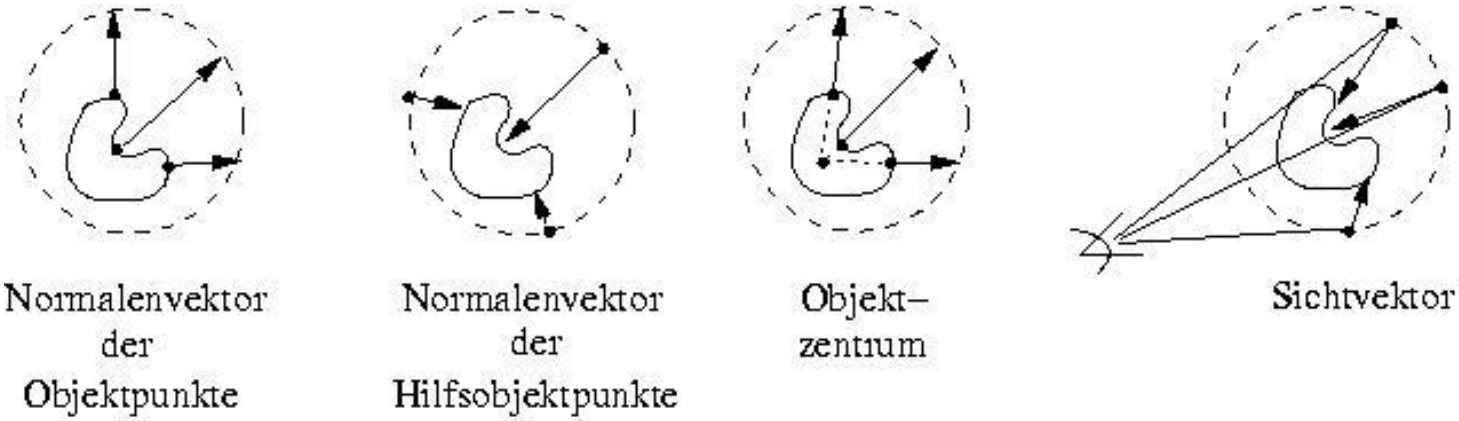
S-Mapping on a Cube



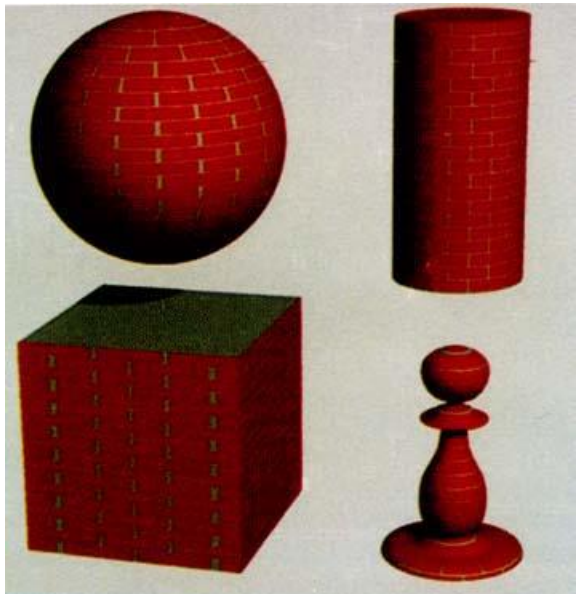
O-Mapping

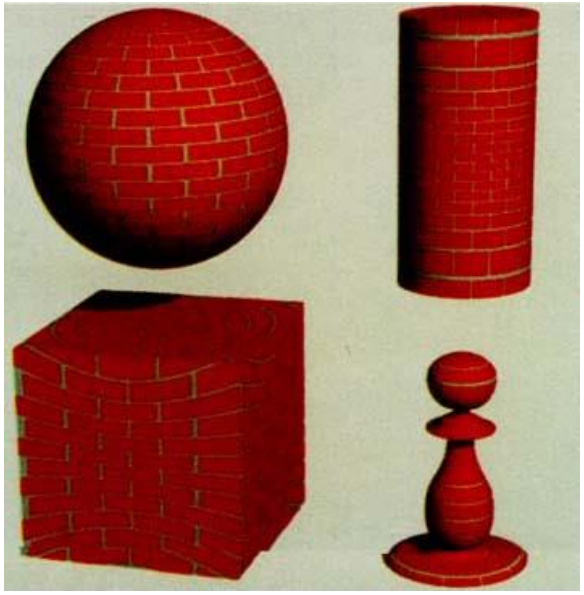
$$T'(x_i, y_i, z_i) \rightarrow O(x_w, y_w, z_w)$$

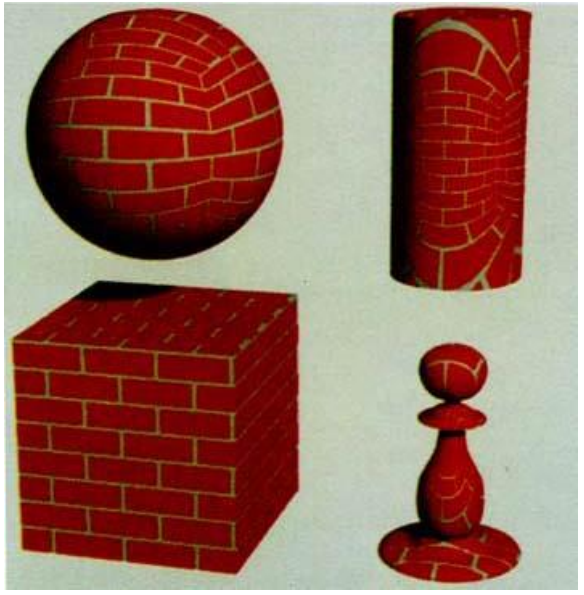
- 4 strategies:
  1. Intersection point of the surface normal vector at every position (x_w, y_w, z_w) and T´
  2. Intersection point of the interim object at every position (x_i, y_i, z_i) with the object surface
  3. Projection of the object midpoint onto the interim surface
  4. Mirror reflection of a sight vector at the interim object. Texture seems to walk over the object surface when the viewer moves (Environment Mapping).



S-Mapping – Effects of Different Interim objects

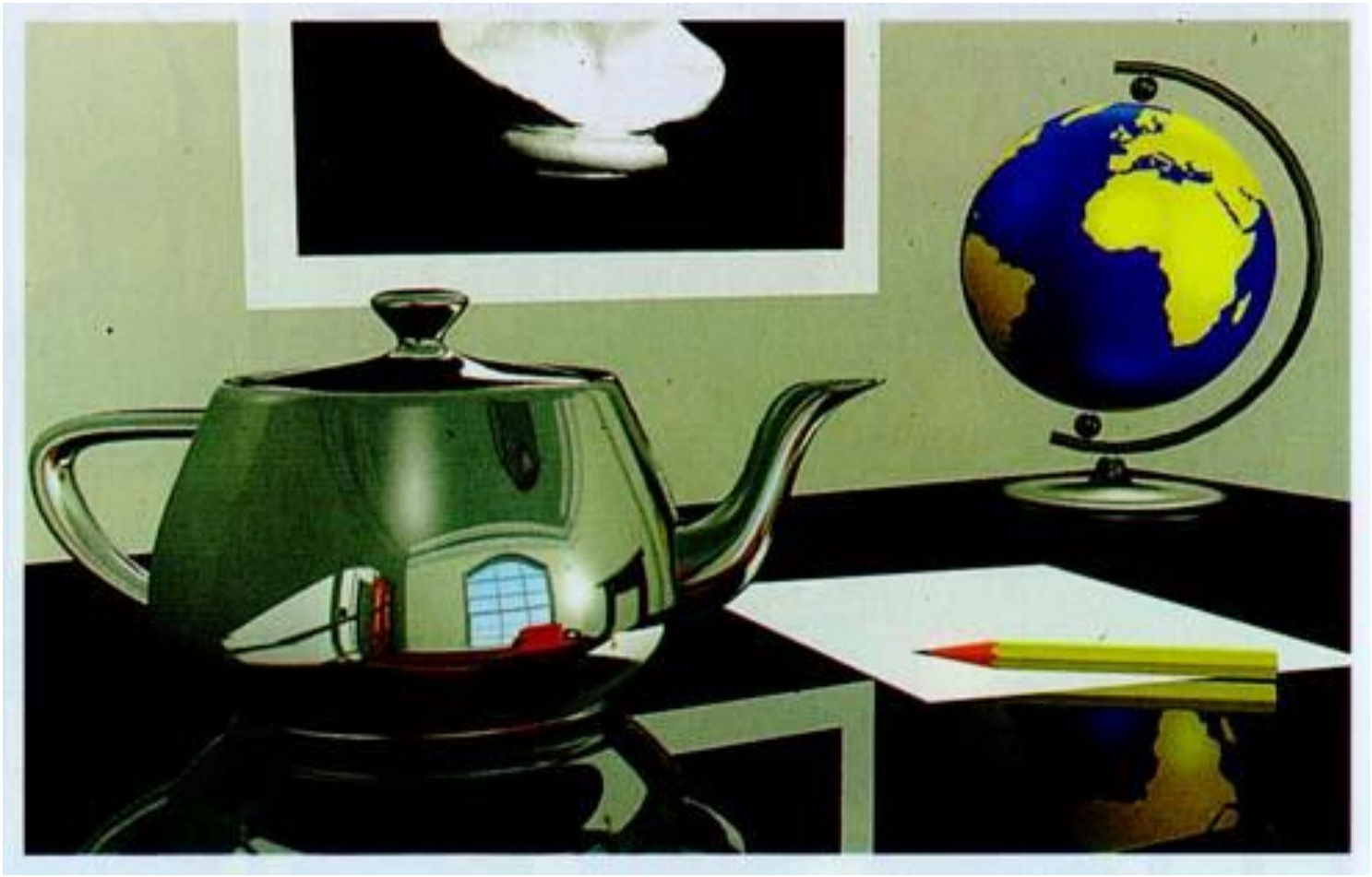






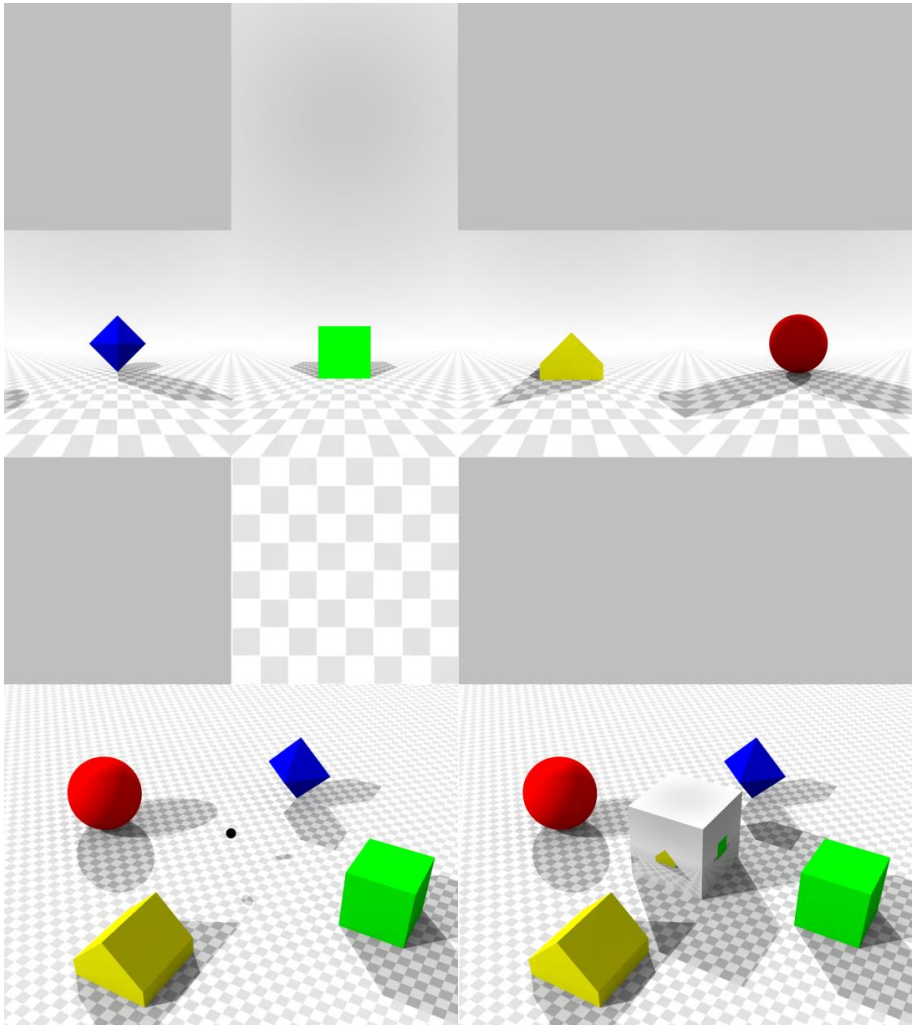
Picture: Watt

Environment Mapping



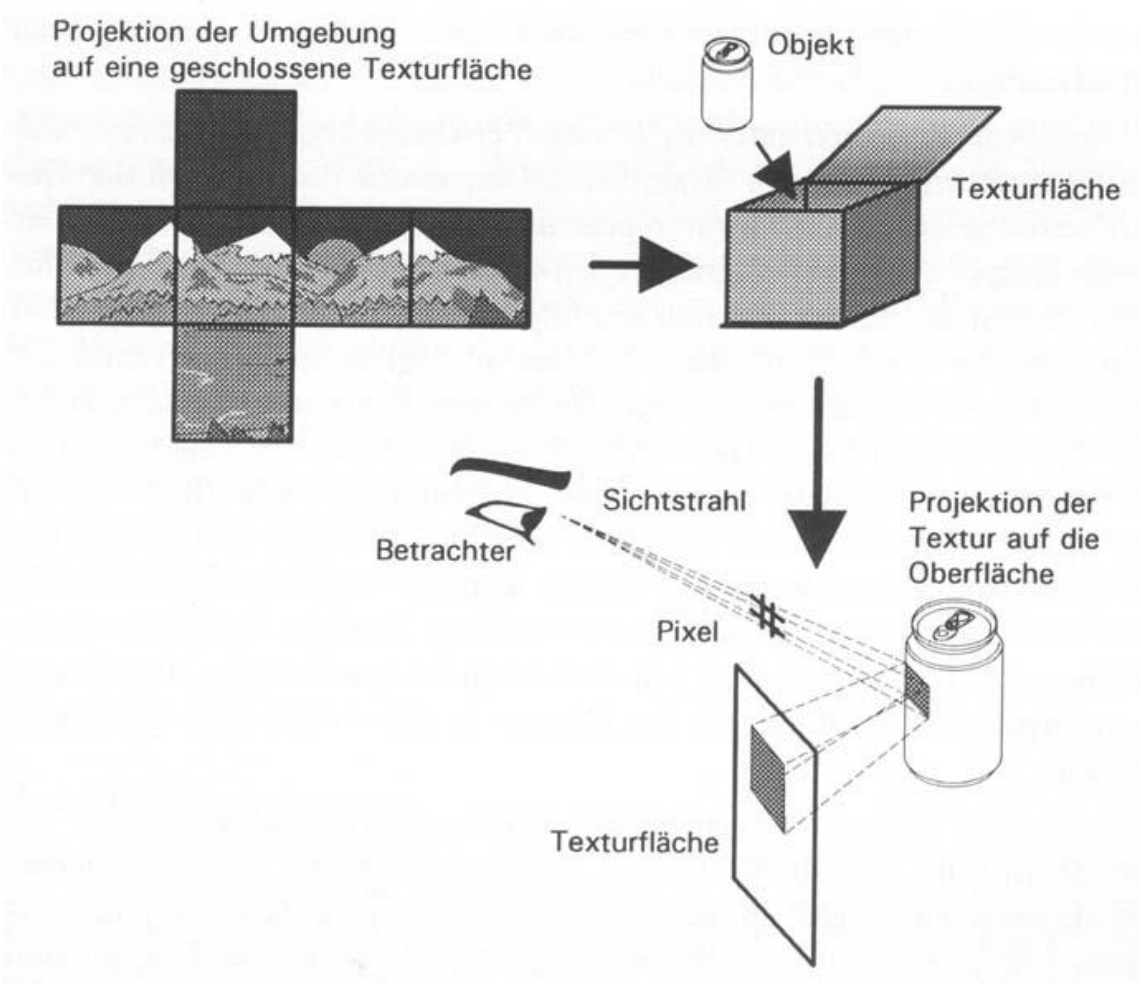
Picture: Watt

Cube Mapping



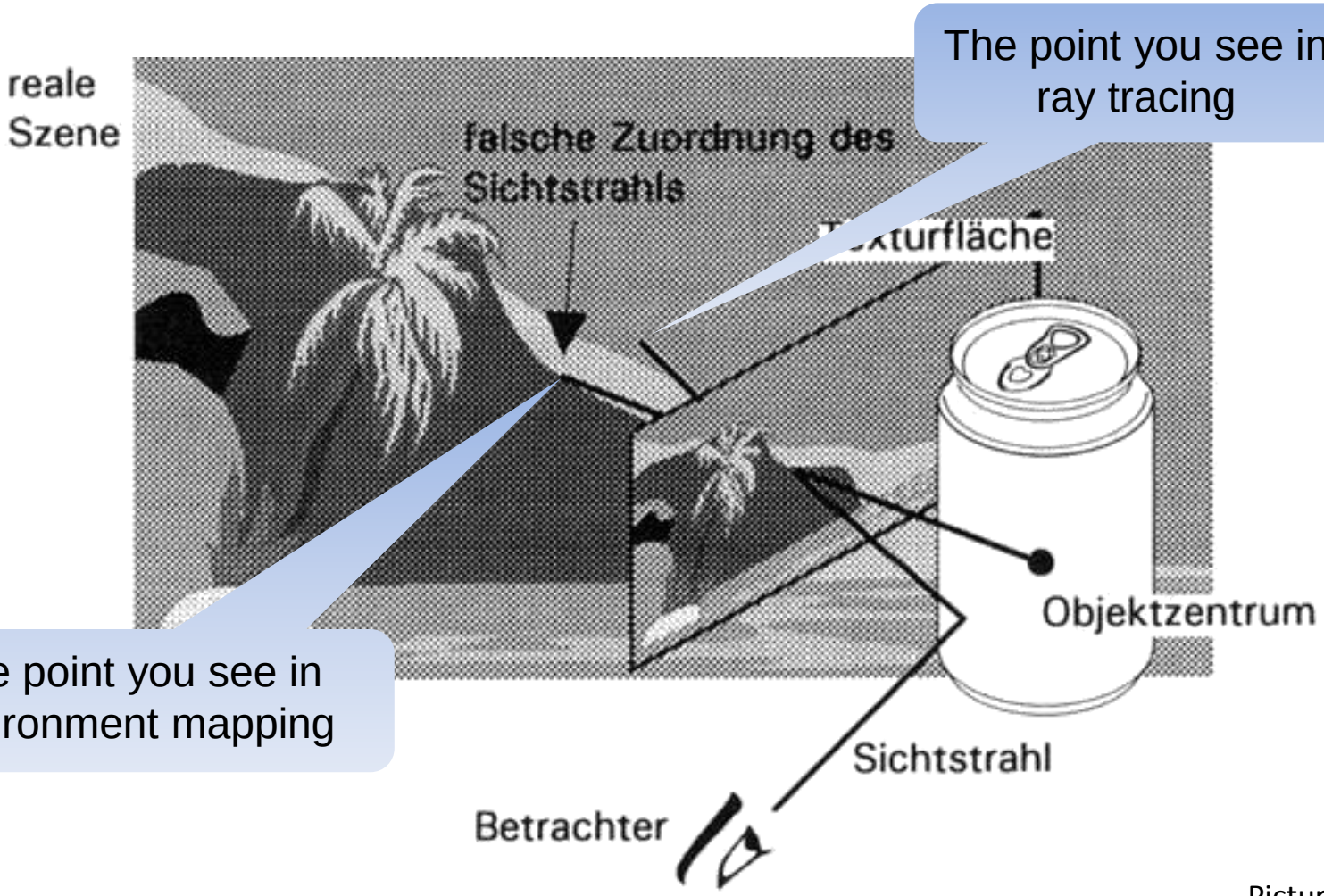
Picture: www

The Principle of Environment Mapping



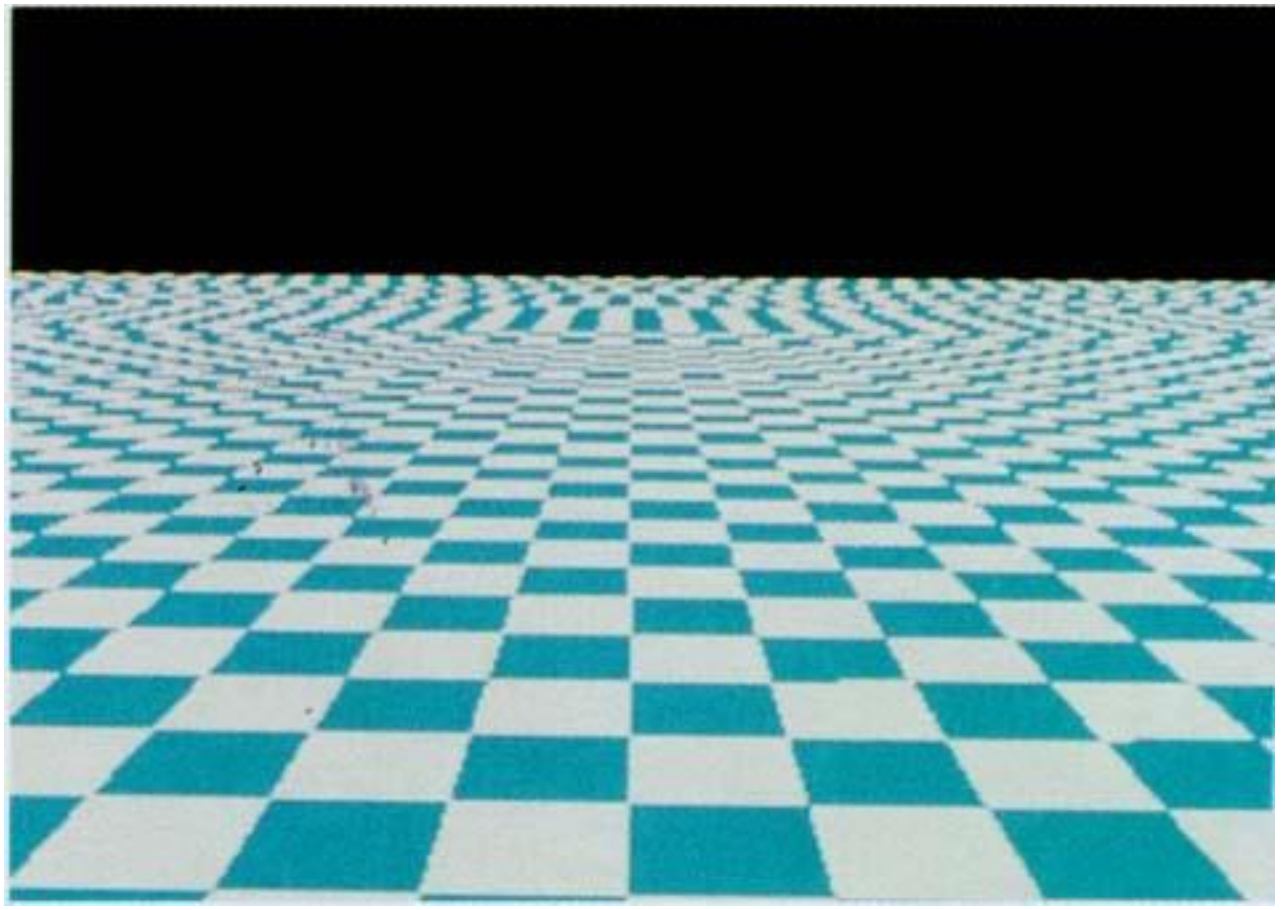
Picture: Tönnies

Environment Mapping versus Ray Tracing



Picture: Tönnies

Aliasing Example



Aliasing Effects

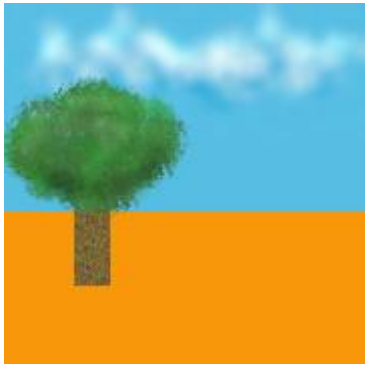
2 extreme situations:

1. Viewer is very close to the textured object: Only one texel for many pixels (tiling effect)
2. Textured object is very far from the viewer: Many texels for only one pixel (see example on last slide)

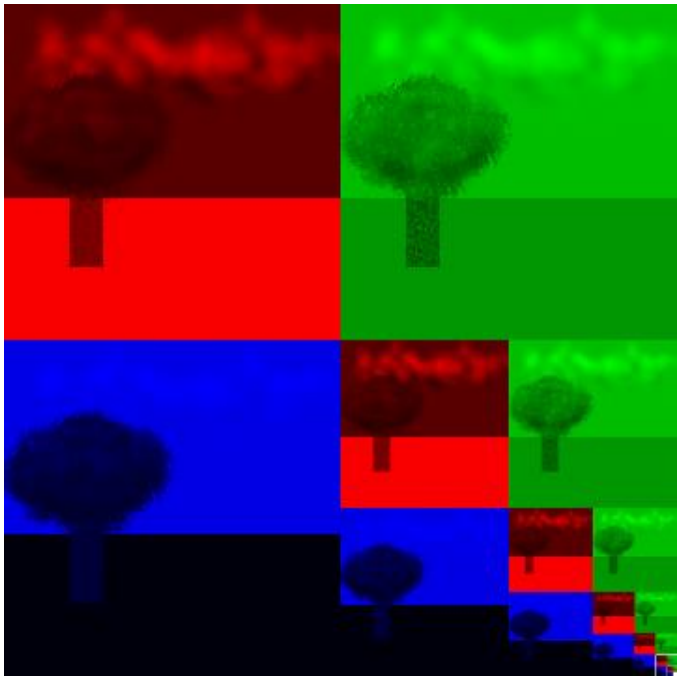
ad 1/2. Texture hierarchies and Mip Mapping

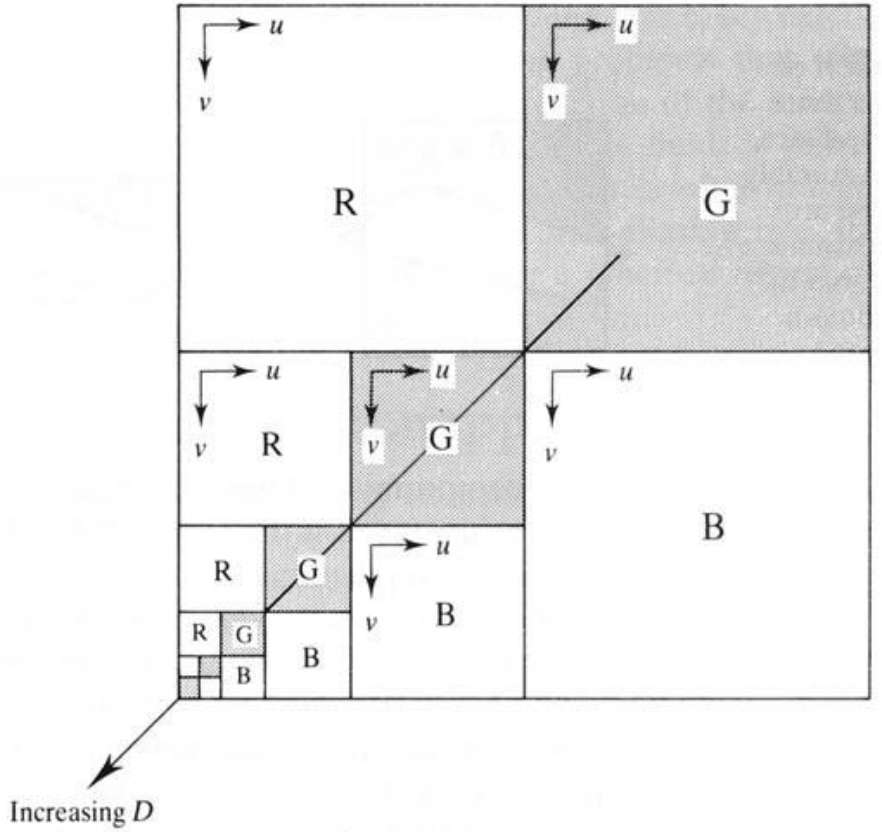
ad 2. Shrink Wrap Method and Inverse Mapping

Texture Hierarchies

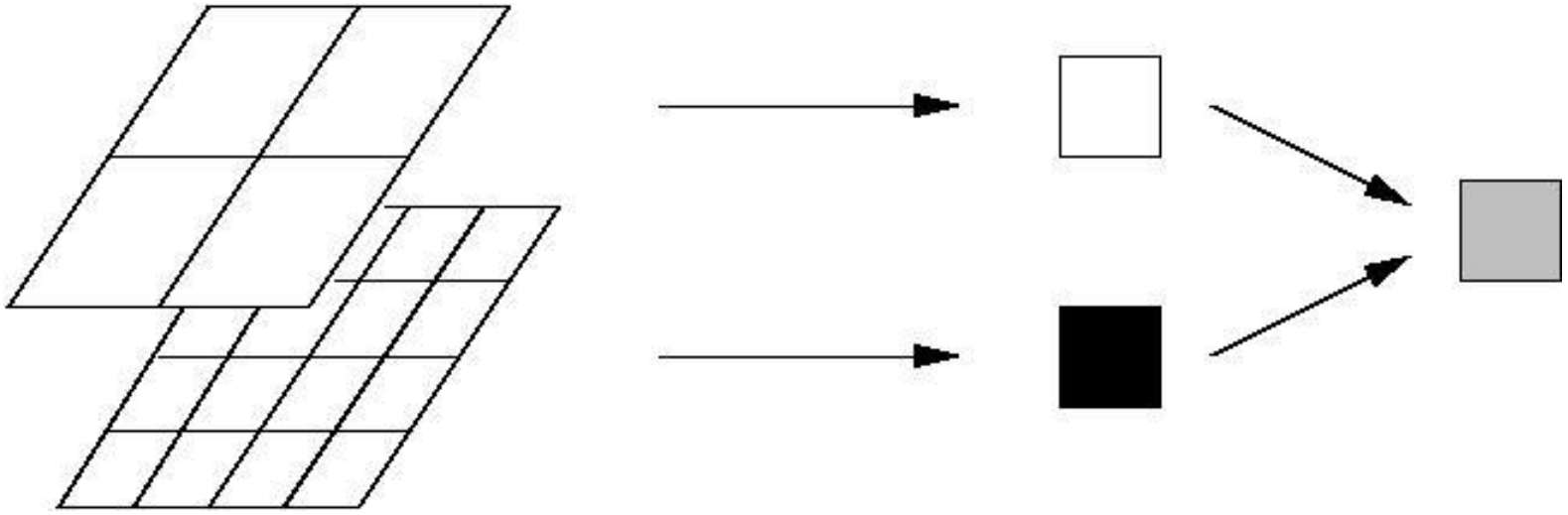


Picture: WWW



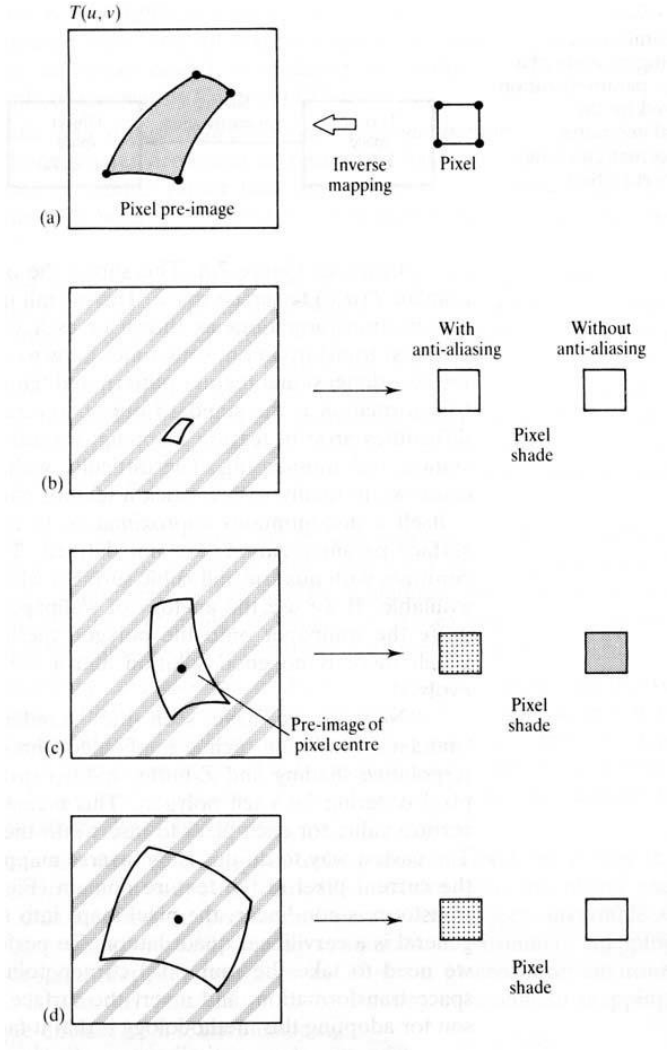


Mip Mapping – Trilinear Interpolation



Shrink Wrap Method & Inverse Mapping

- Start from the 4 pixel corners (Inverse Mapping)
- Transform all 4 corners into texture coordinates
- Integrate over the surface in the texture map that is covered by the pixel



Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

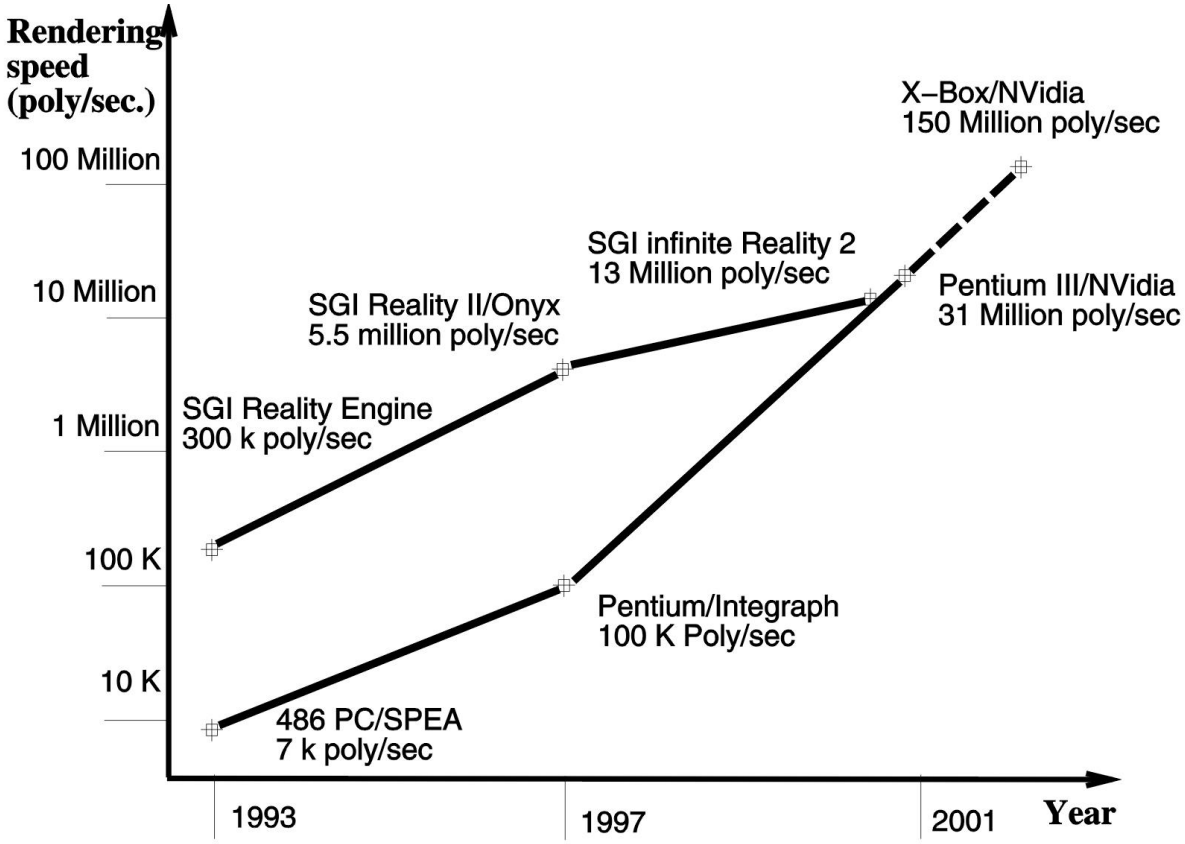
How to Measure Graphics Performance

- **Speed:**
  - Number of polygons per second (Attention: Size and shape of polygons, shading method, ...)
  - Fill Rate
  - Data transport: Graphics hardware – CPU – main memory
  - Benchmarks: ViewPerf, ... (see WWW)
- **Quality:**
  - Memory – Frame Buffer (size, color depth), Z-Buffer (depth), Texture Memory, ...
  - Texture Mapping Features
  - Hardware Anti-Aliasing
  - Special Features: Stereo, Vertex & Pixel Shaders, ...
- **Flexibility:**
  - Configuration Interface
  - Number of resolutions and screens supported, ...

History of Graphics Performance

Picture:Burdea et al

2014: 1.3 Billion poly/sec
(NVIDIA Quadro FX 6000)



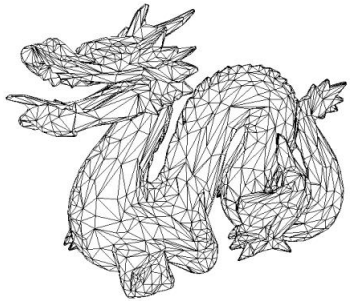
How to make Graphics faster

- Chip development, processor clock speed
- Fast and larger busses (PC: AGP, PCI Express)
- Add more memory: Resolution, color depth, z-buffer depth, amount of textures, anti-aliasing
- Pipelining
  - „macroscopical“ (Rendering Pipe)
  - onChip (float-multiplication)
- „Real“ parallel (SIMD,...)
  - „macroscopical“ (more Chips)
  - onChip (more ALUs, …)

2 strategies:

- Geometry on CPU, scan conversion on graphics hardware
- Both on graphics hardware (common today)

Rendering Pipeline



Transformation:
Position und orientation
of objects

Transformation:
WKS, SKS

Culling

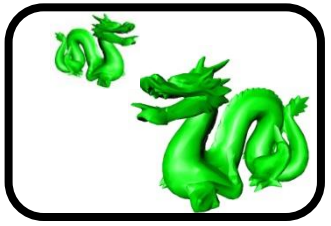
Projection

Clipping

Shading

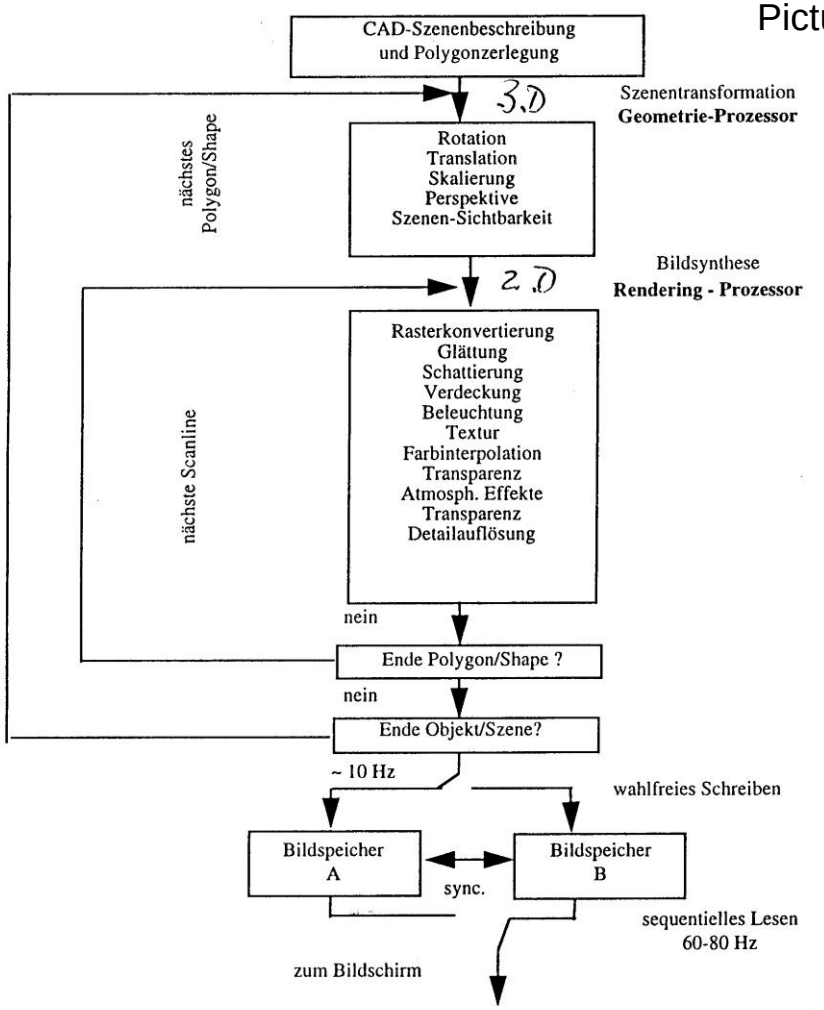
Hidden Surface
Removal

Scan
Conversion



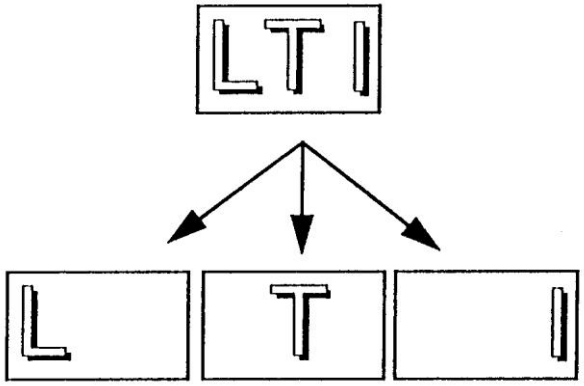
Rendering Pipeline – 3-D & 2-D

Picture: Kraiss, Technische Informatik, RWTH Aachen



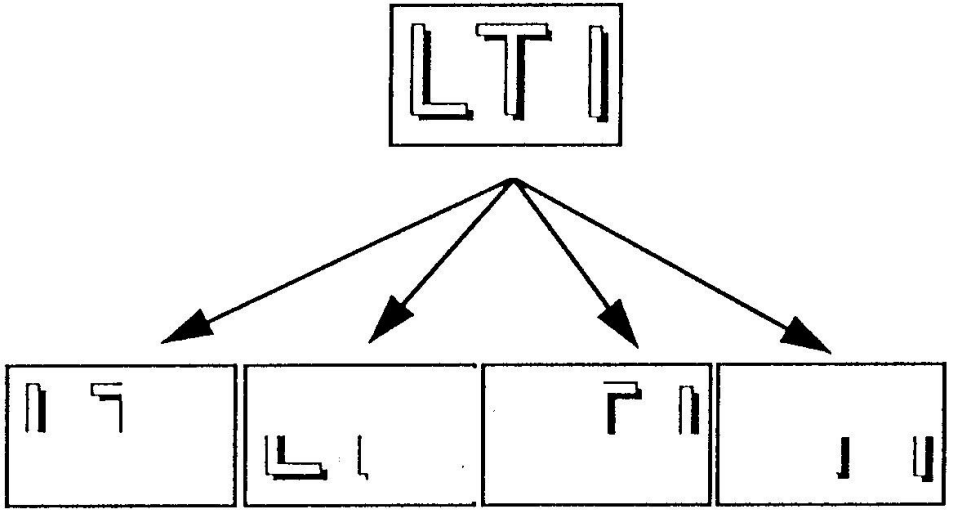
Composition versus Partition

Picture: Kraiss, Technische Informatik, RWTH Aachen



Composition

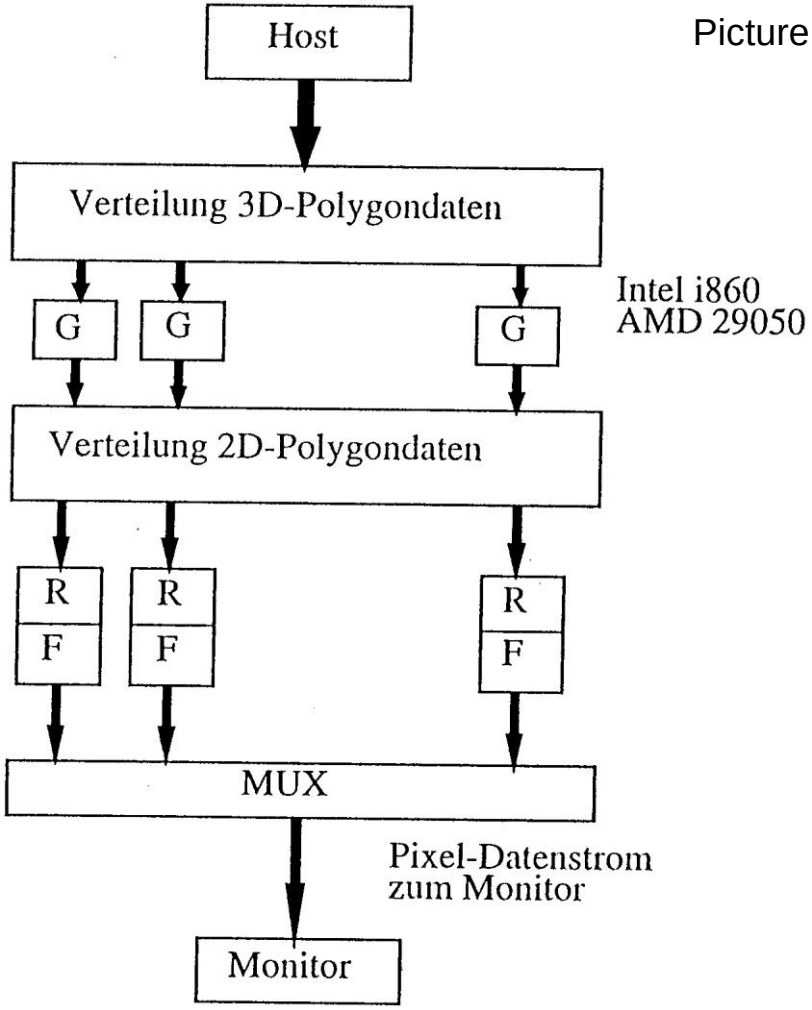
„Sort last“



Partition

„Sort First“

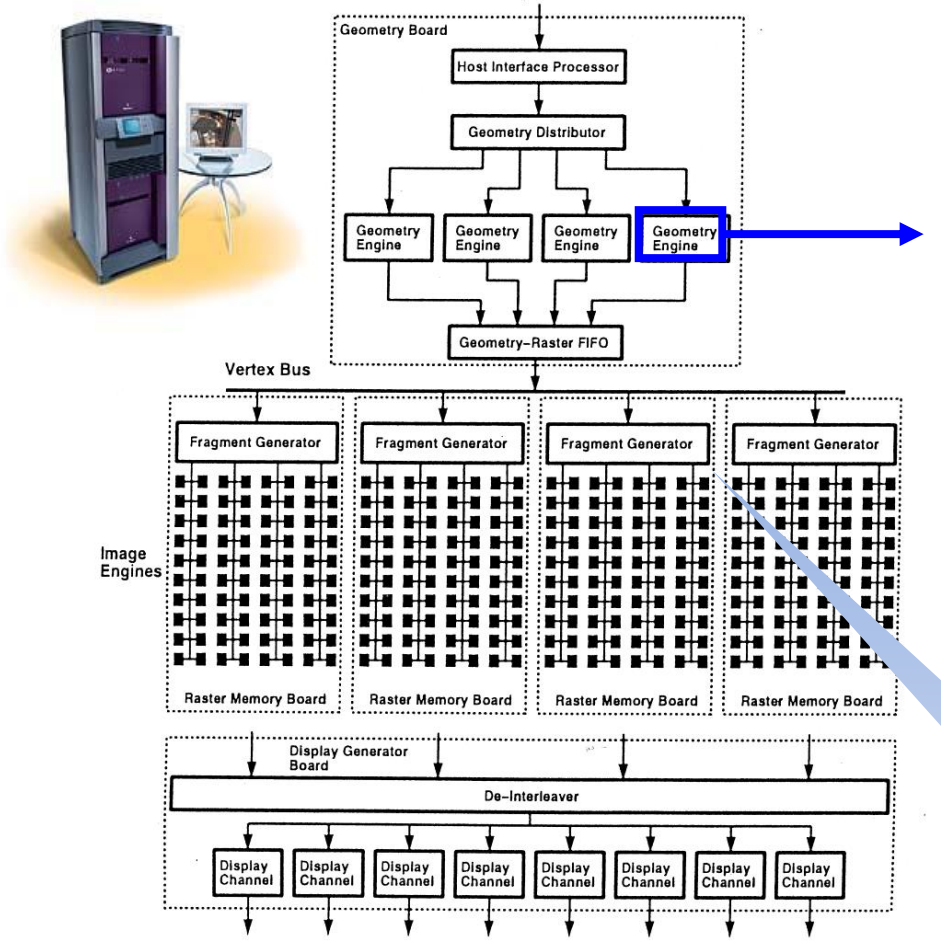
Architecture for Partitioning

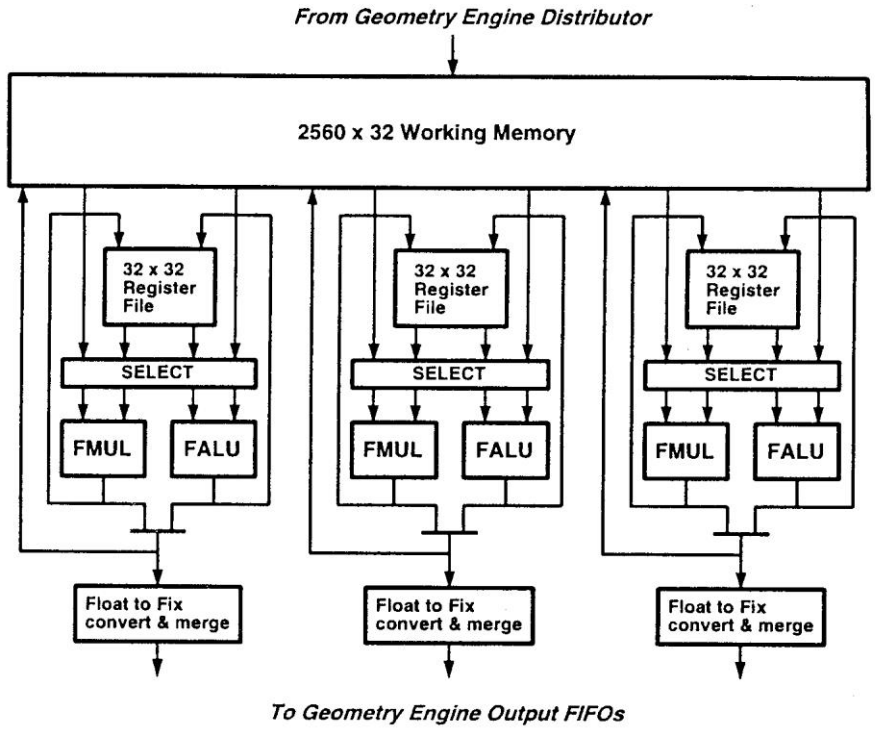


Picture: Kraiss, Technische Informatik, RWTH Aachen

SGI Infinite Reality – Geometry Engine

Picture: Silicon Graphics Inc.

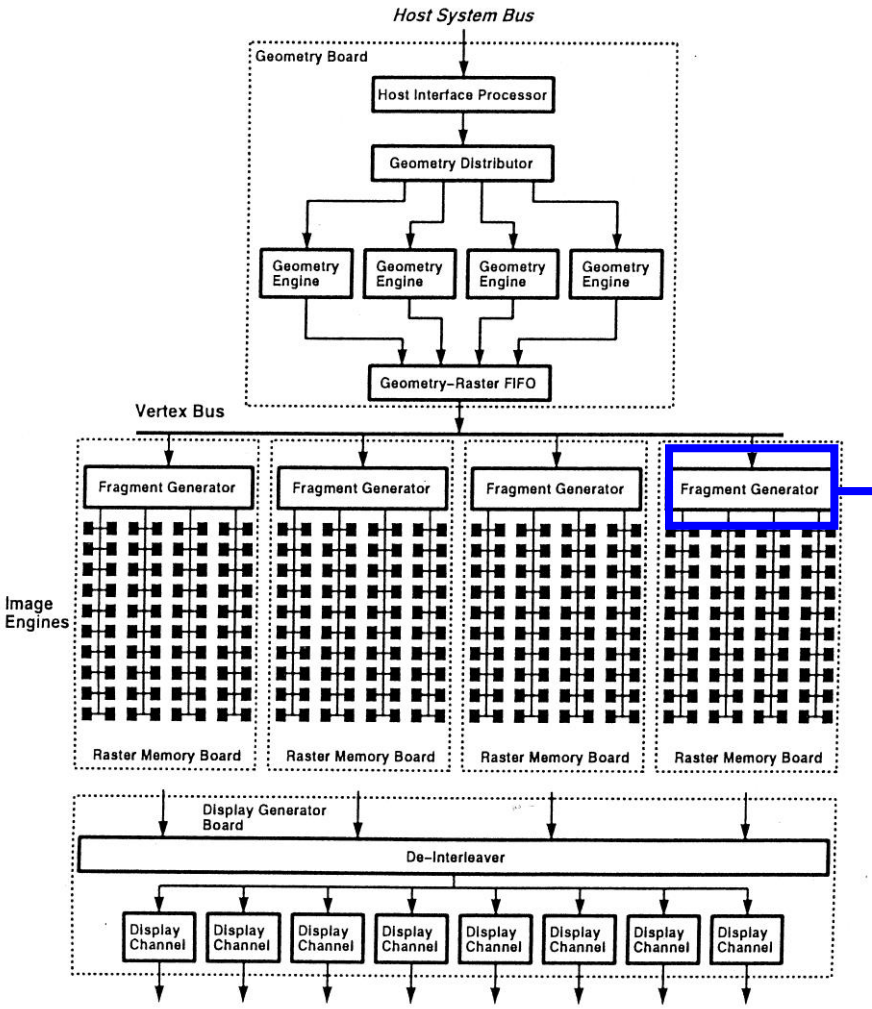


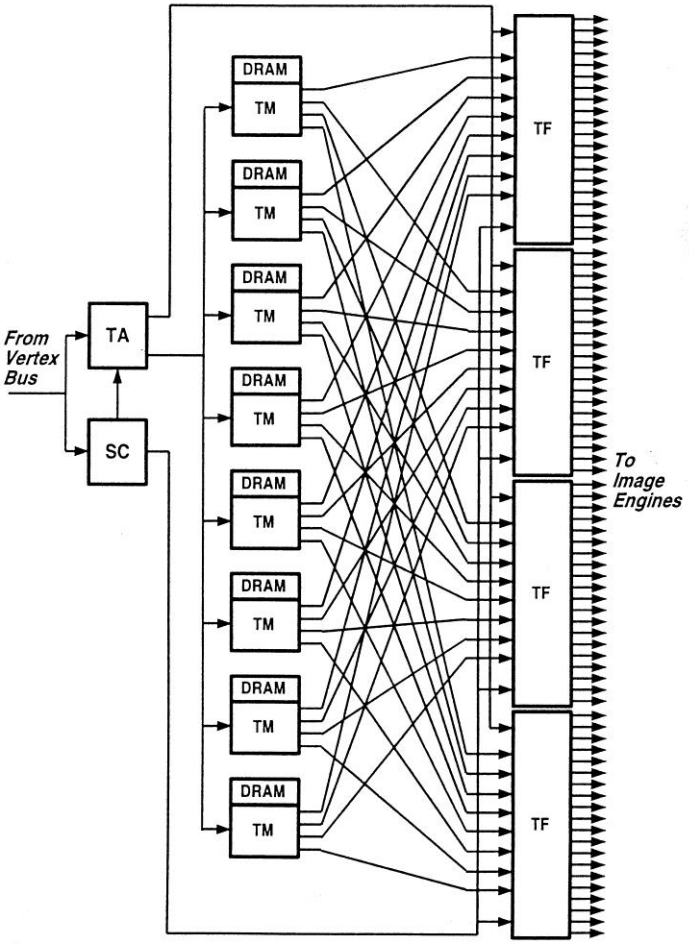


80 x 16 = 1280
(see SXGA)

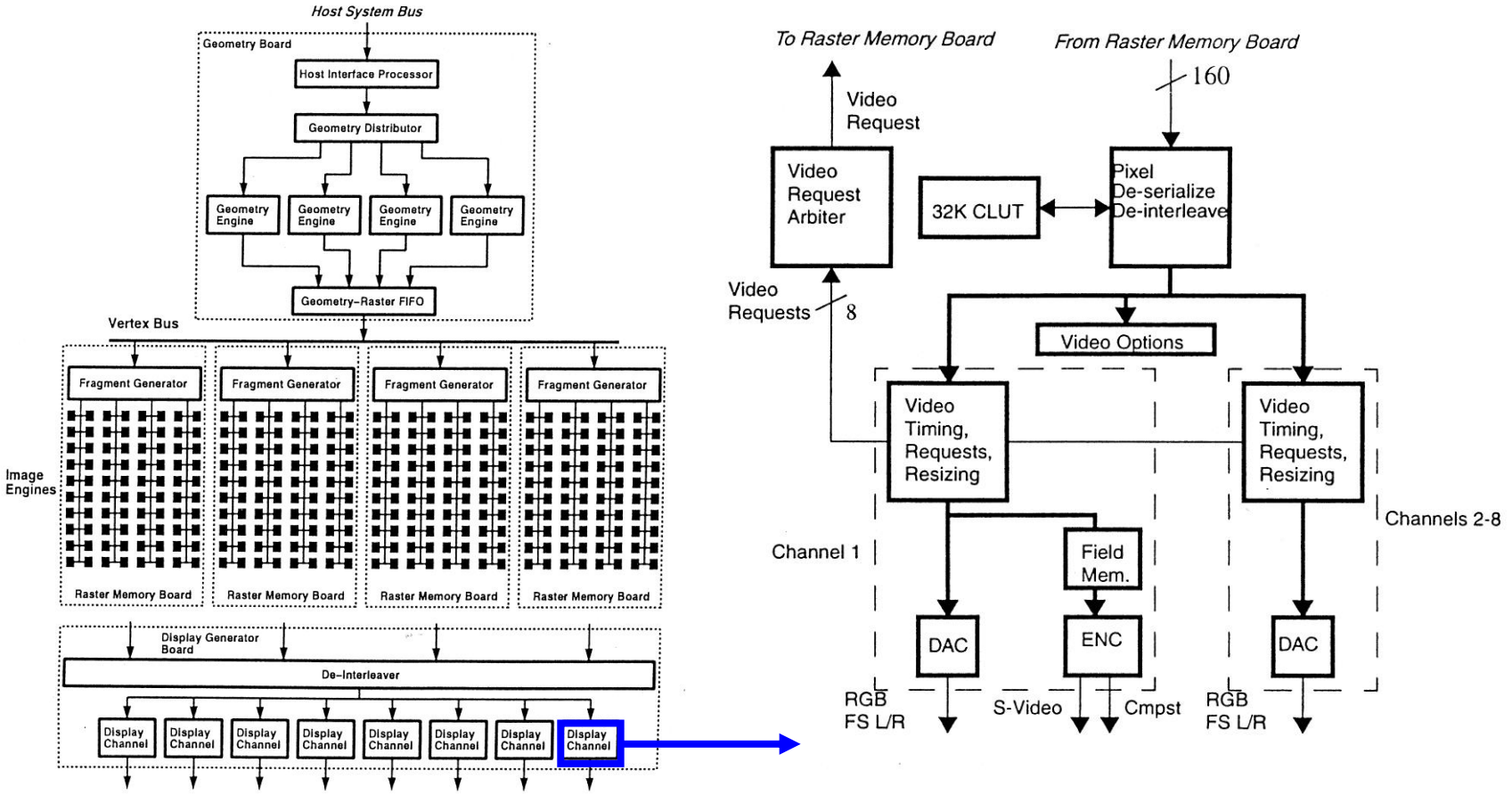
SGI Infinite Reality – Raster Manager

Picture: Silicon Graphics Inc.



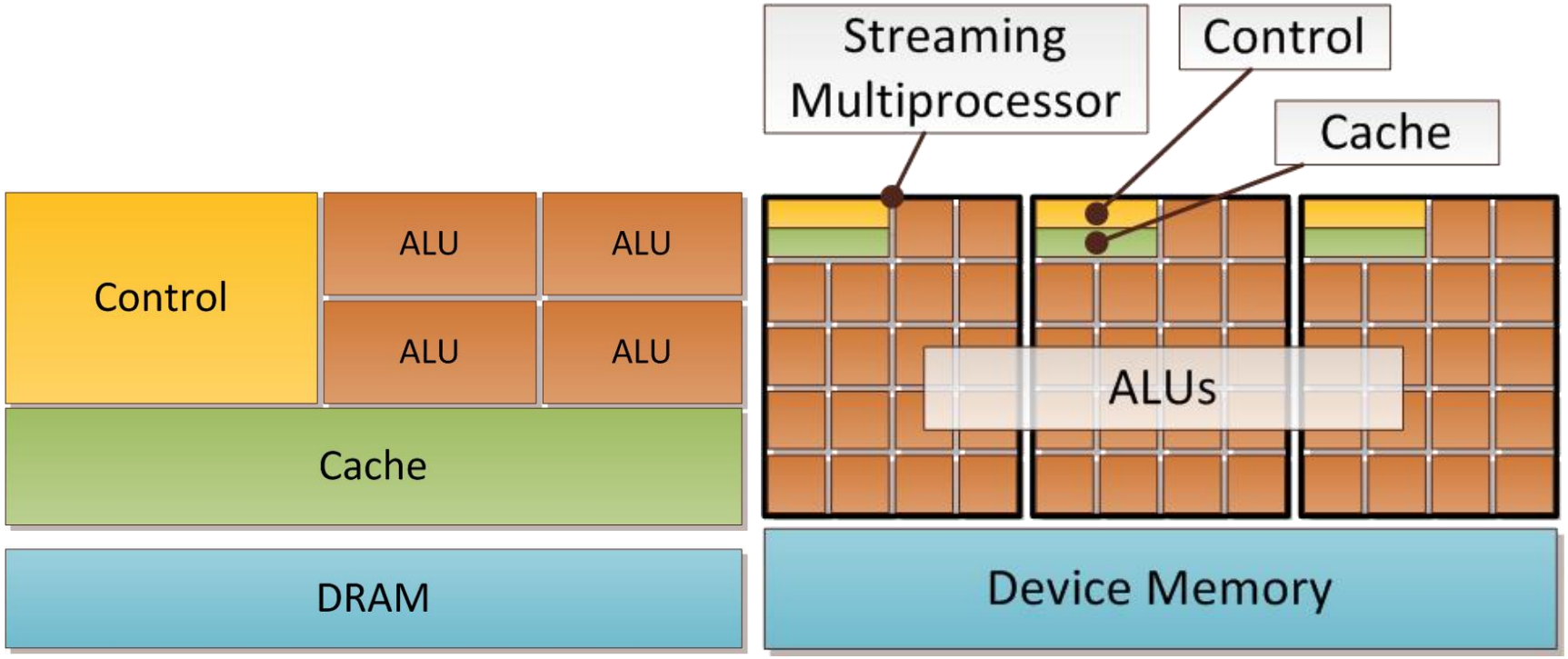


SGI Infinite Reality – Display Generator



Picture: Silicon Graphics Inc.

CPUs versus Modern GPUs

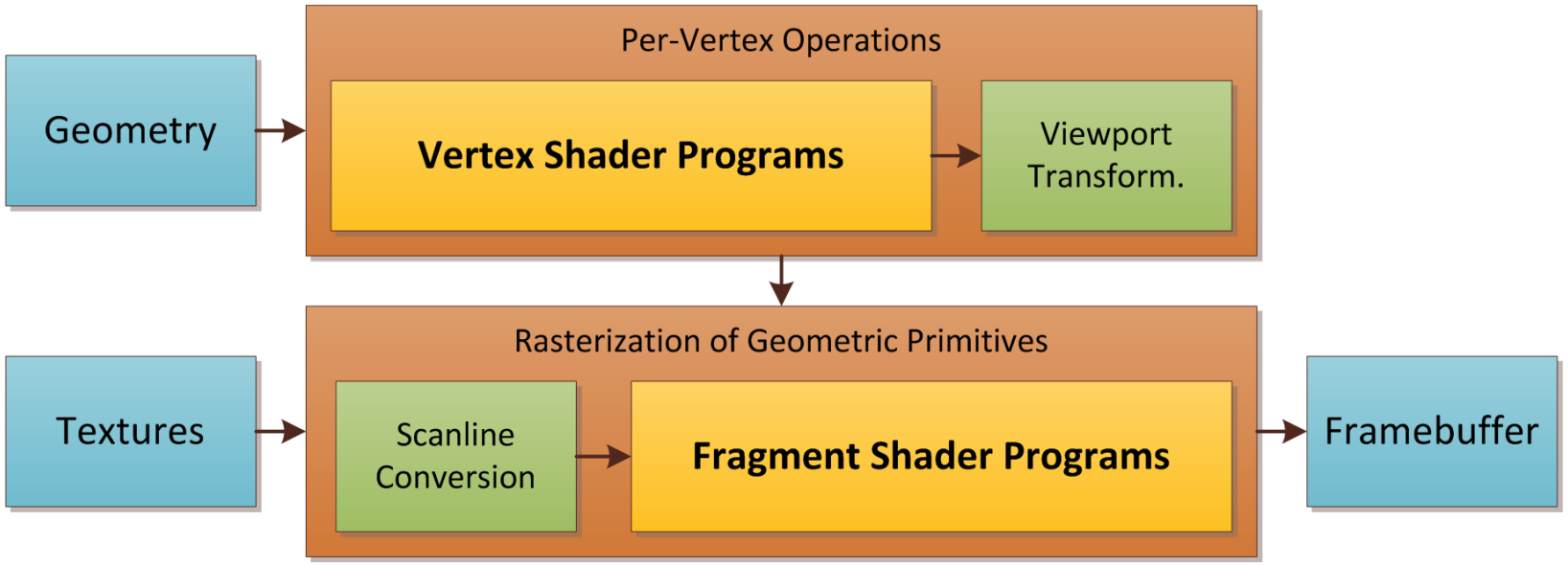


2011: About 100 GFLOPS

2011: About 1500 GFLOPS

Pictures: Tobias Rick

Modern GPU Architecture



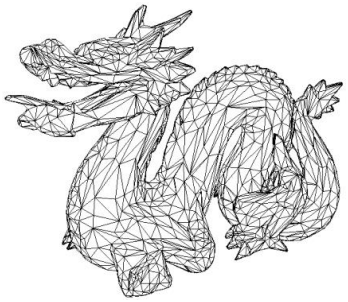
Pictures: Tobias Rick

Topics

- Rendering Pipeline in a Nutshell
- Representation of rigid objects
- Rendering Pipeline
  - Transformations
  - Culling
  - Projection

    » **Stereoscopic, Viewer-Centered Projections**
  - Clipping
  - Hidden Surface Removal
  - Shading
  - Scan Conversion
- Texture Mapping
- Graphics Hardware
- Ray Tracing

Position in Rendering Pipeline



Transformation: position and orientation of objects

Transformation: WKS, SKS

Culling

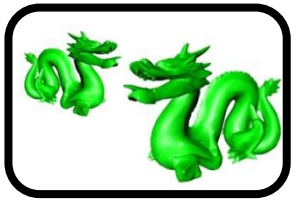
Projection

Clipping

Shading

Hidden Surface Removal

Scan Conversion



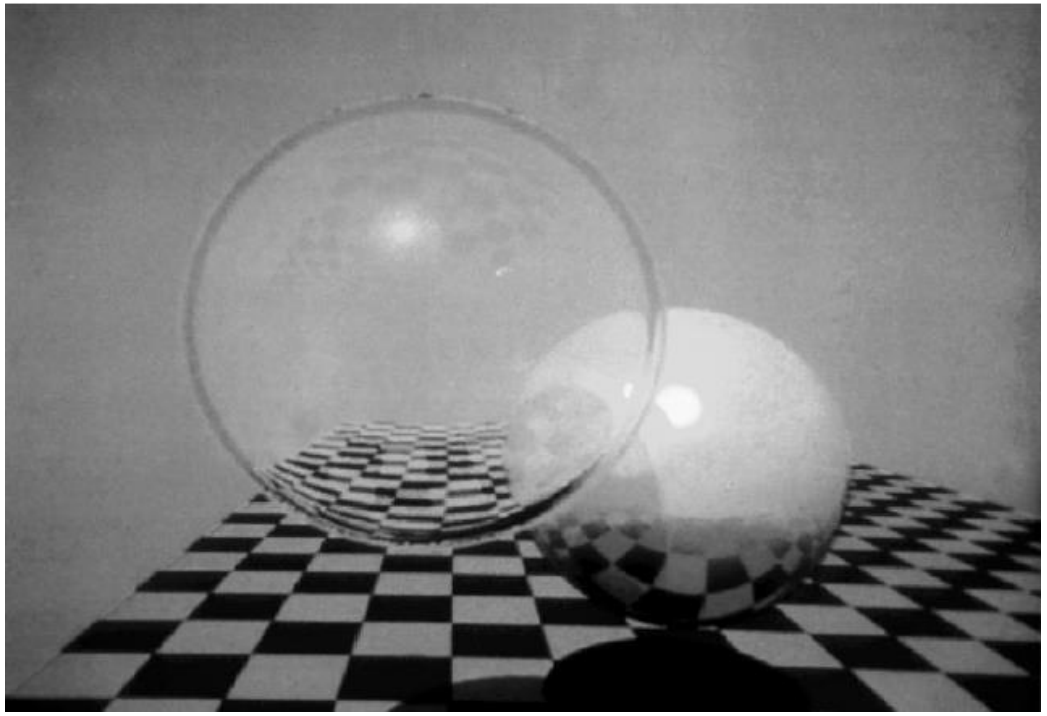
Ray Tracing -The Idea

- Whitted 1980:
  - Create photorealistic images by following single light beams
  - Simulate the process of light distribution by the laws of ideal mirror reflection and refraction

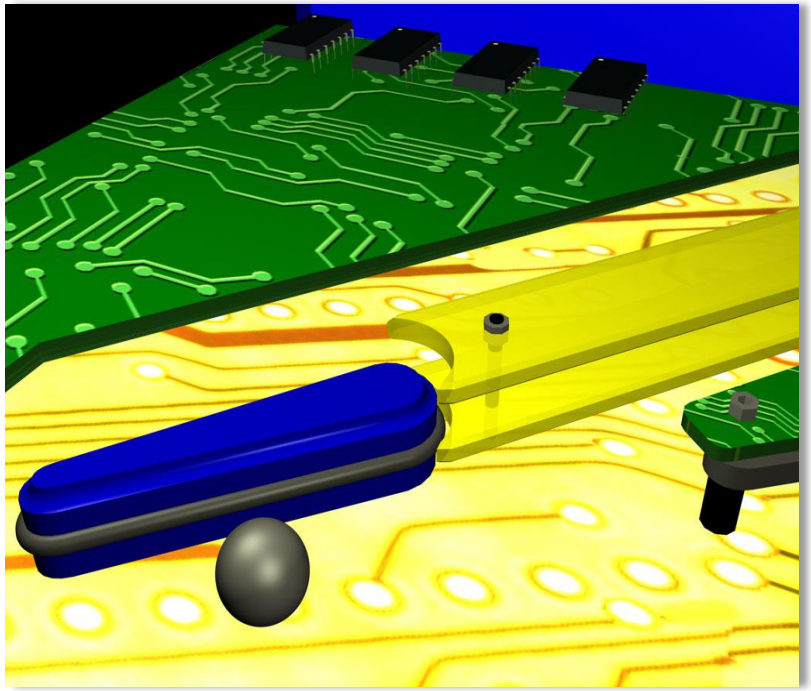
Root:

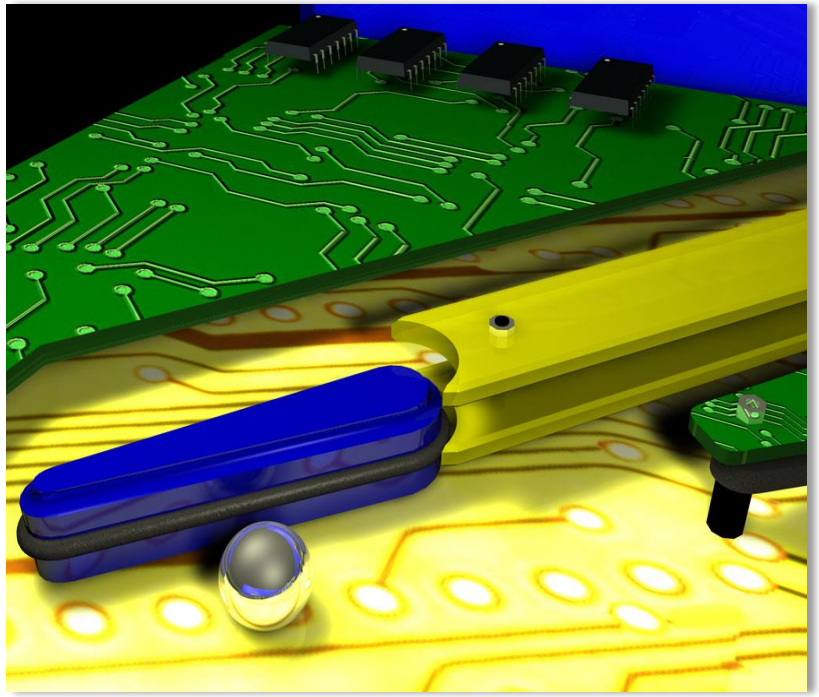
Ray optics in physics (e.g., Descartes)

- Occlusion (HSR)
- Shadows
- Reflection
- Refraction



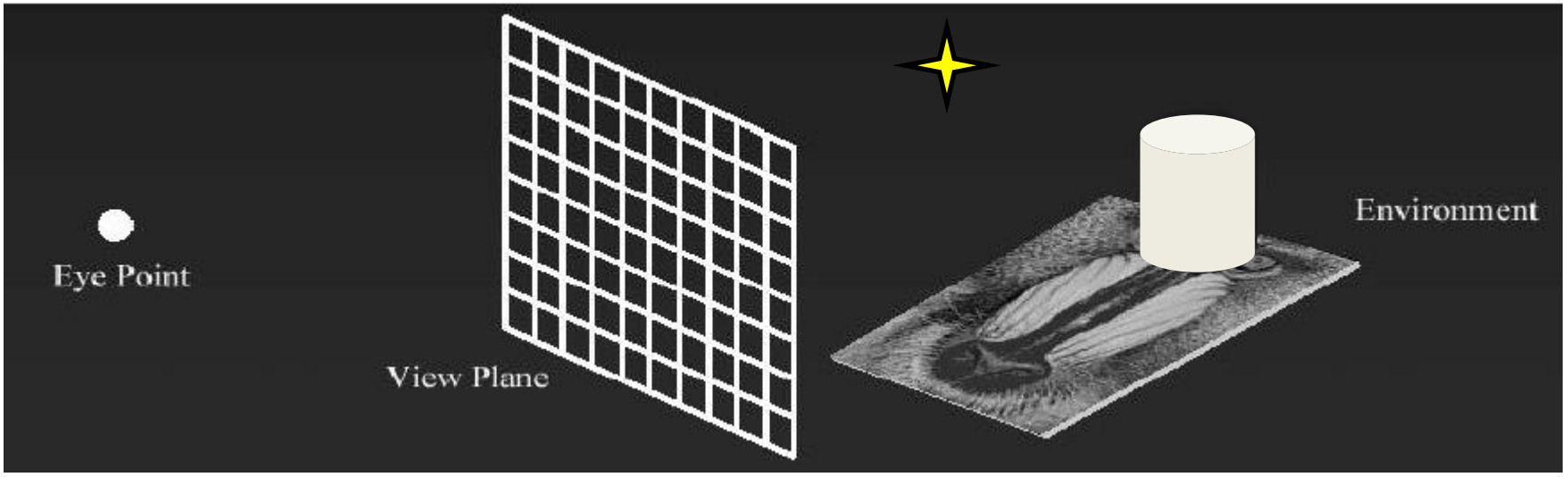
Gouraud Shading versus Ray Tracing





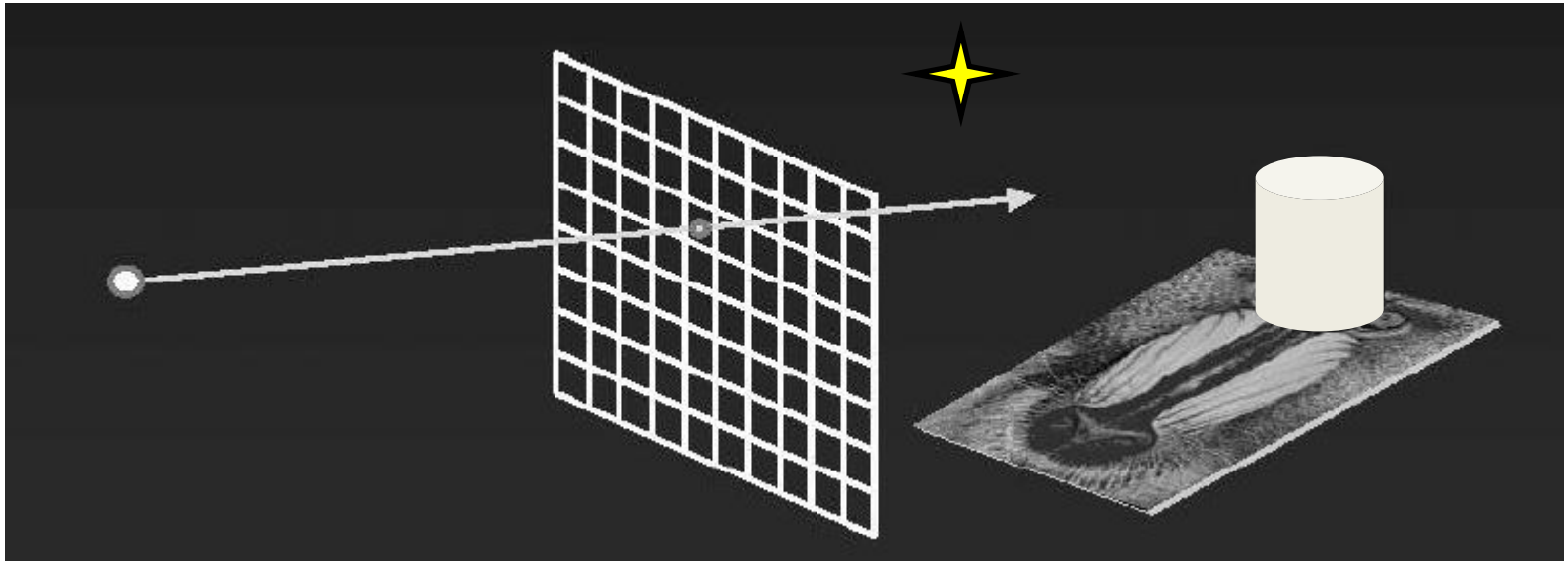
Virtual Flipper at VR Group, RWTH Aachen

The starting situation

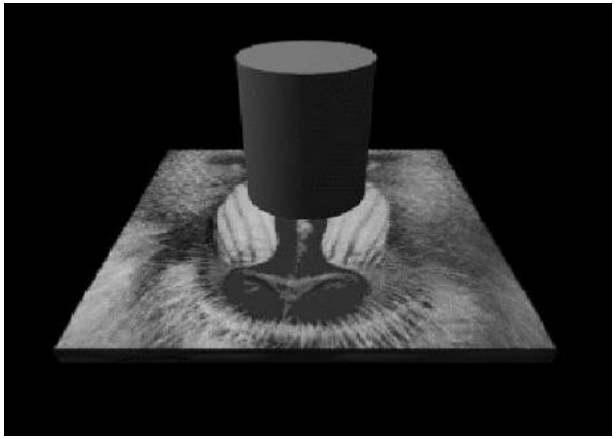


- 3-D scene
  - Objects
  - (Point) light sources
- Eye position
- Screen (viewing plane), Pixel

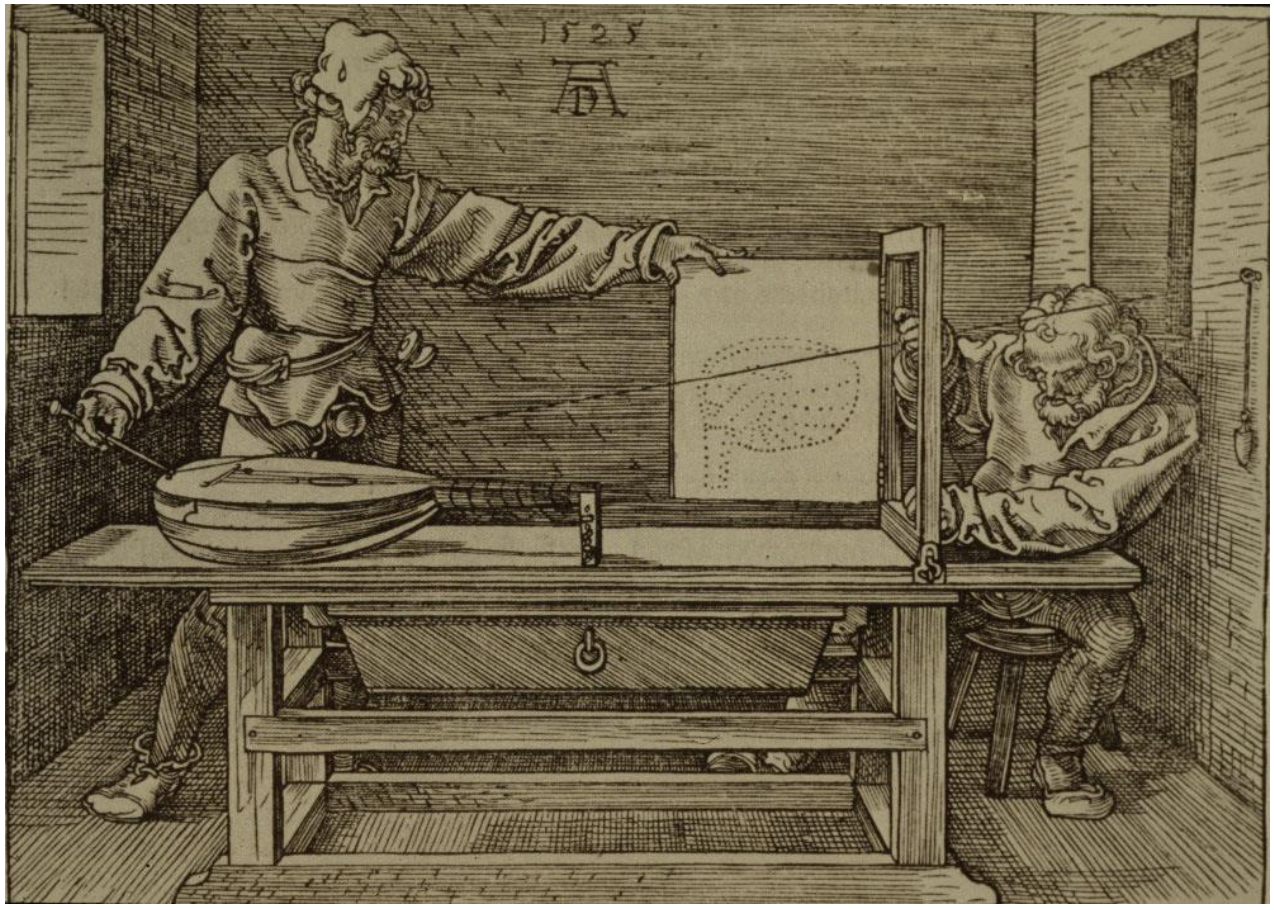
First Step – Ray Casting



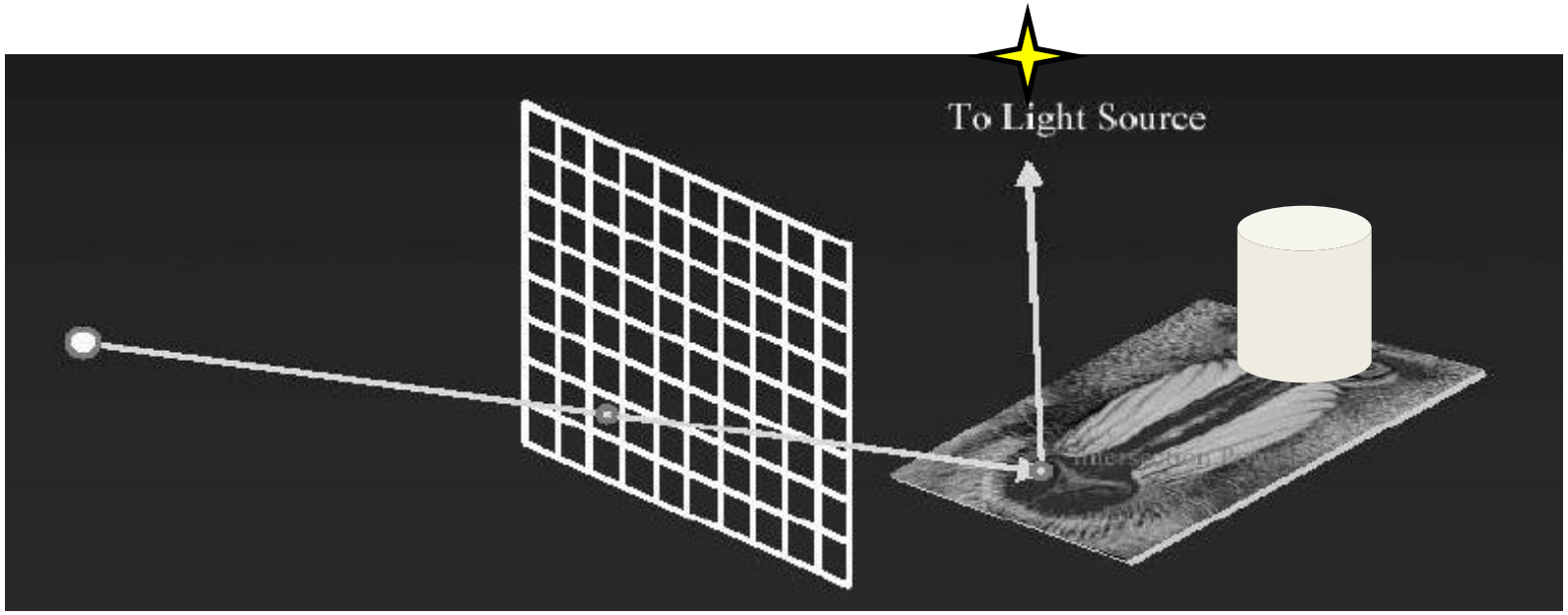
- View rays through all pixels of the viewing plane
- Intersection point of the view ray with an object
- Appel (1968)
  Hidden Surface Removal



Albrecht Dürer, 1525

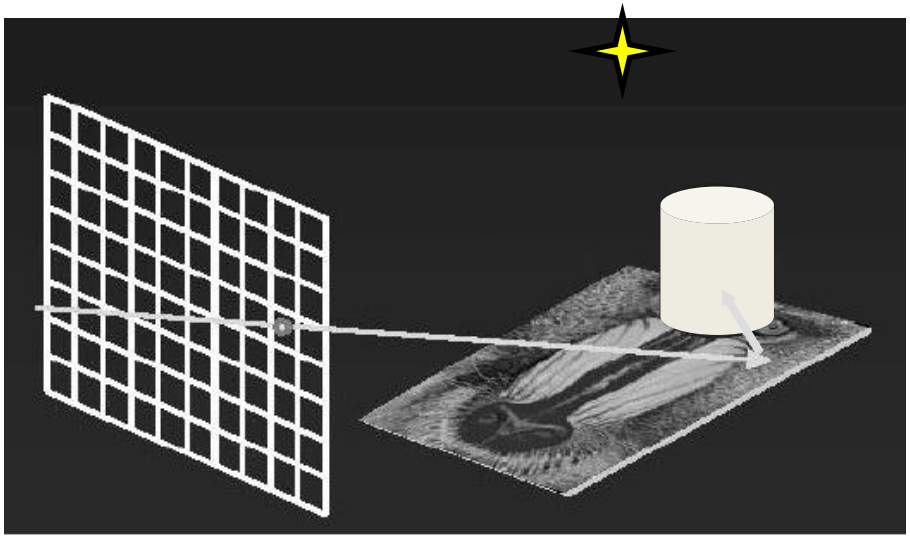


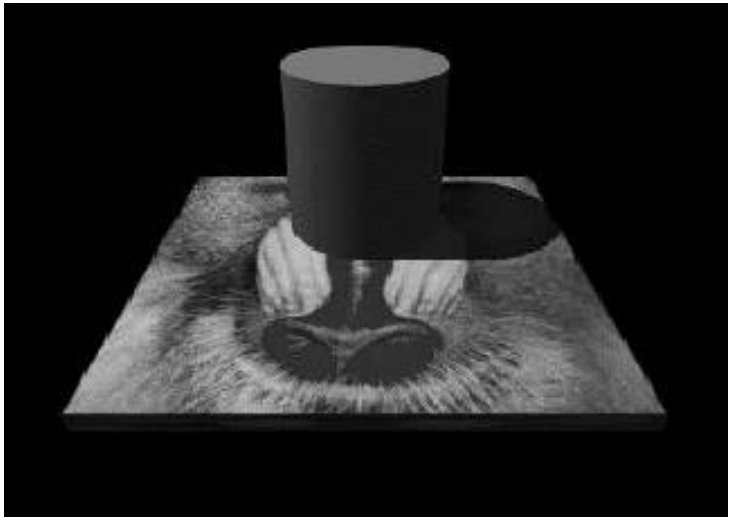
Shadow Feelers I



- „Shadow Feelers“: Rays from the intersection point to the light sources
- Calculate intensity at the intersection point (e.g., Phong Reflection Model)

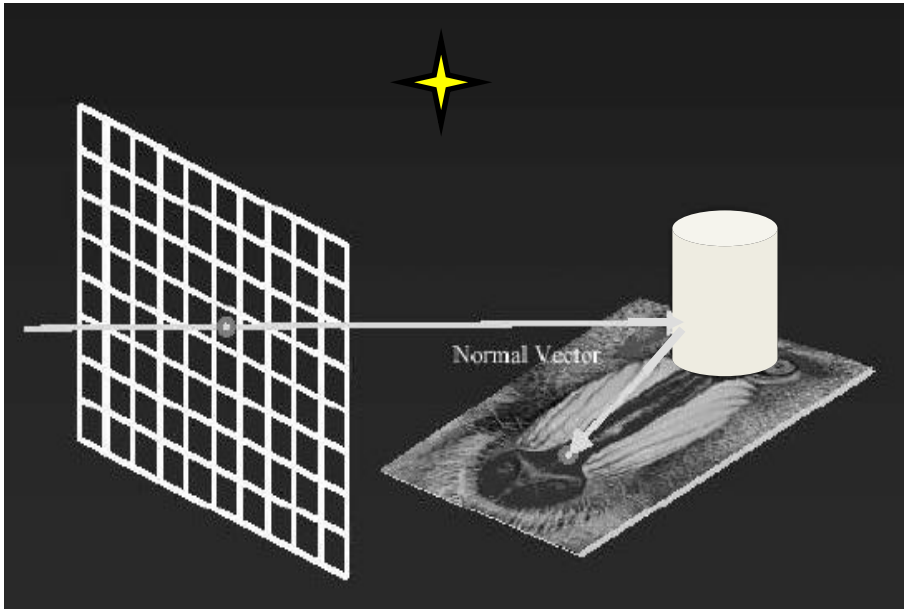
Shadow Feelers II

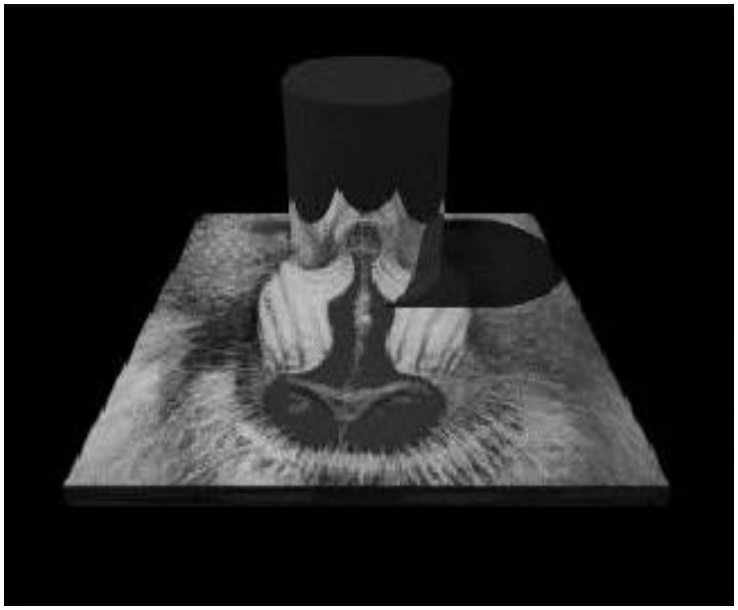




- Object between intersection point and light source: Intersection point is in the shadow

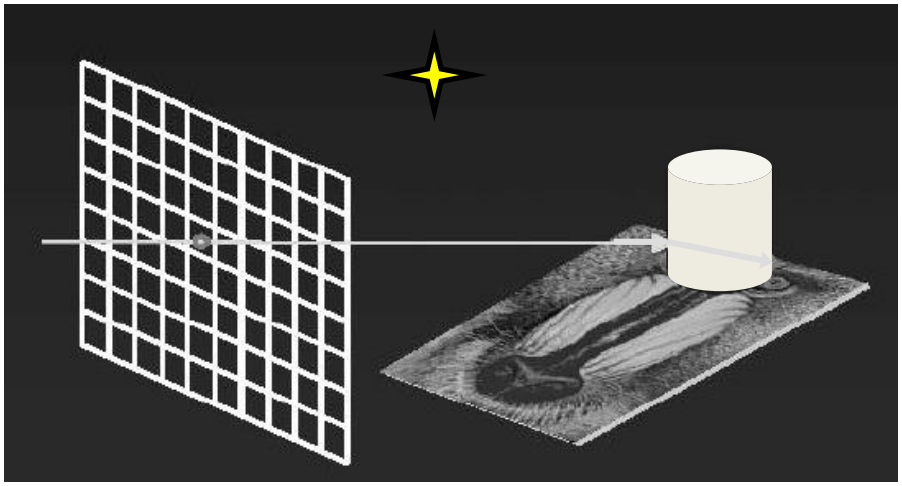
Reflection

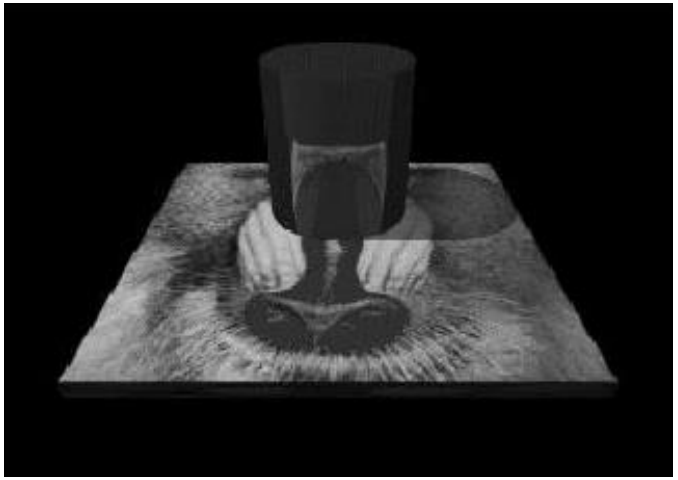




- Intersection point on mirroring object (specular reflection): Calculate and follow the reflected ray

Deflection / Refraction

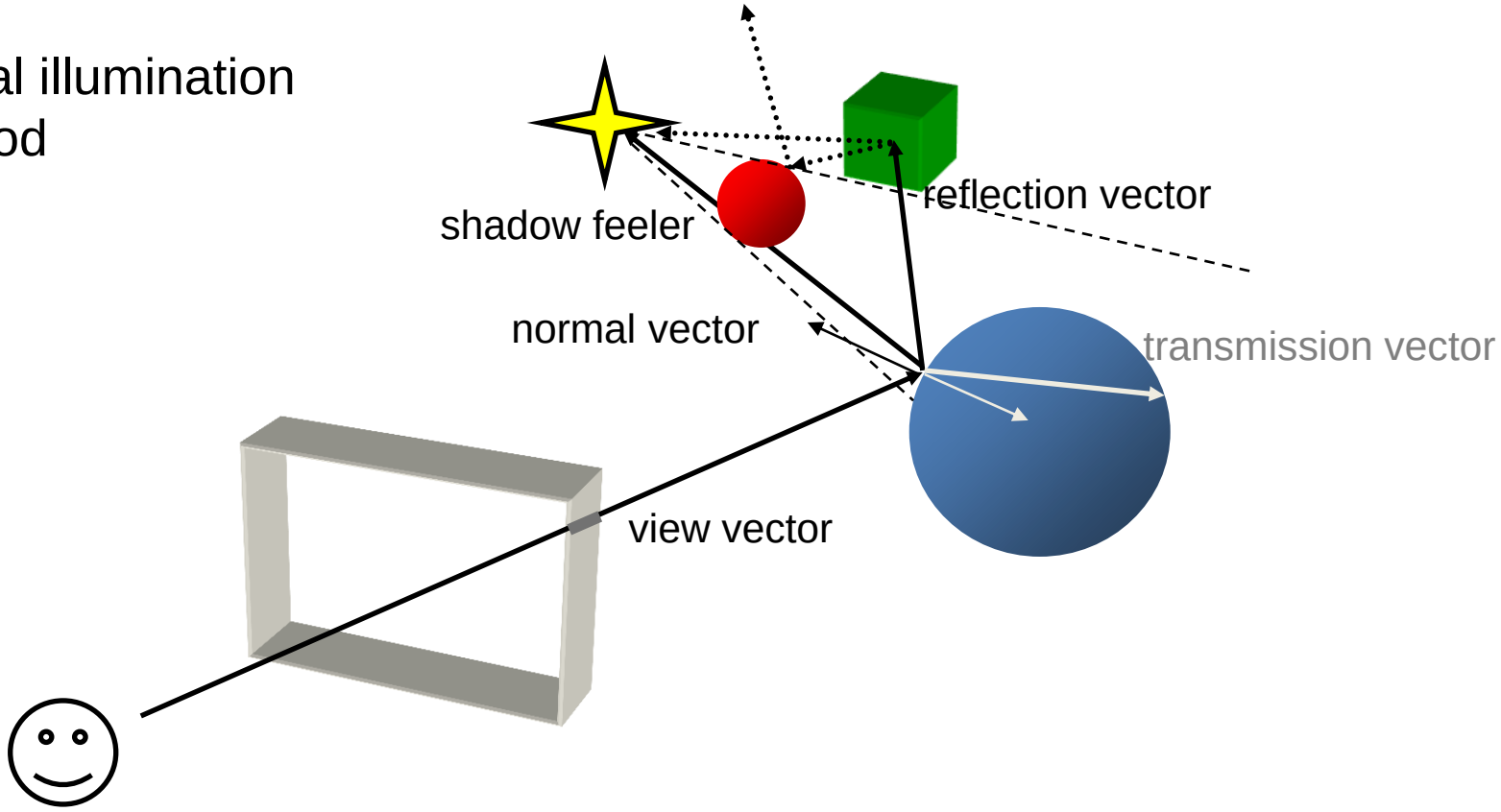




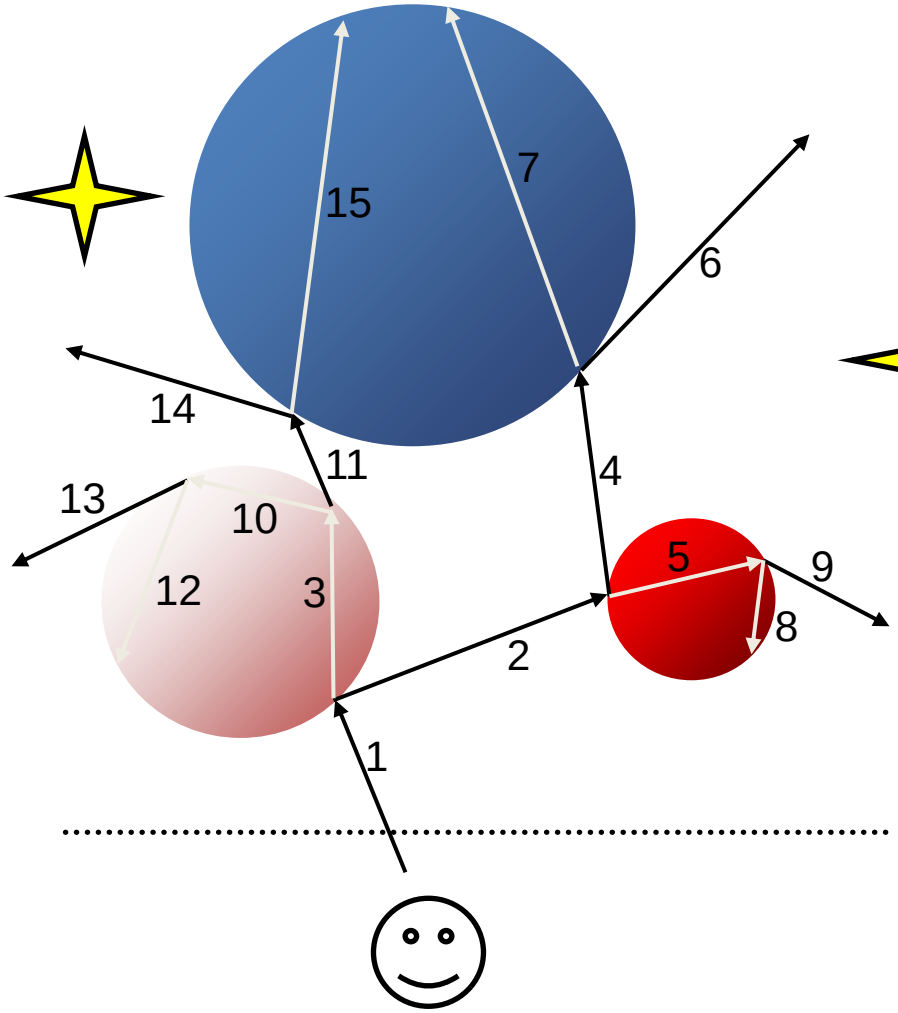
- Intersection point on a transparent object: Calculate and follow the refracted ray

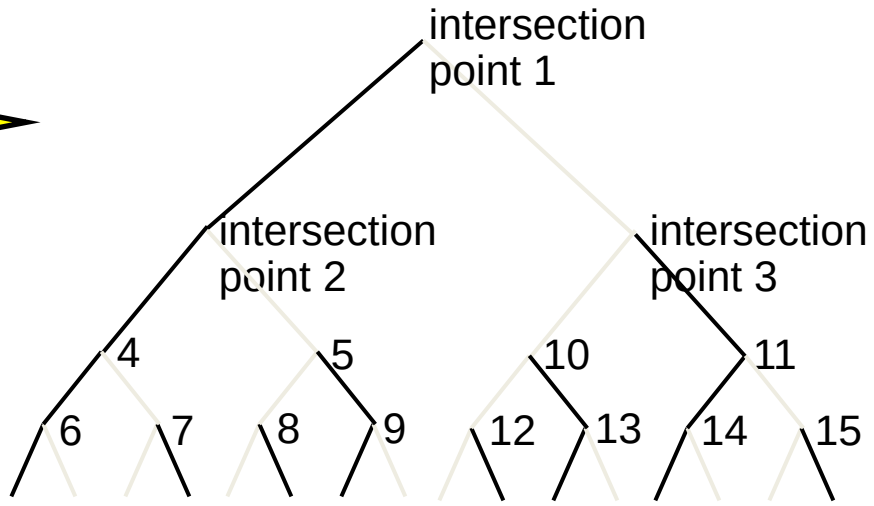
Ray Types

- Reciprocity of reflection: Backwards Ray Tracing
- Global illumination method

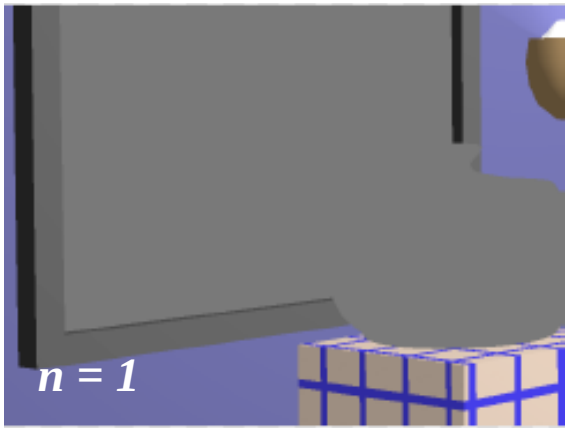


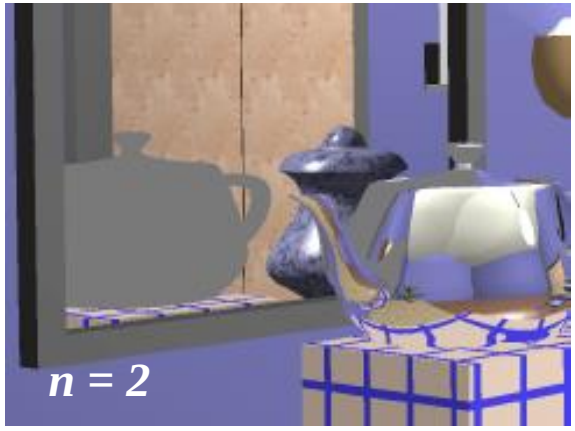
Recursion

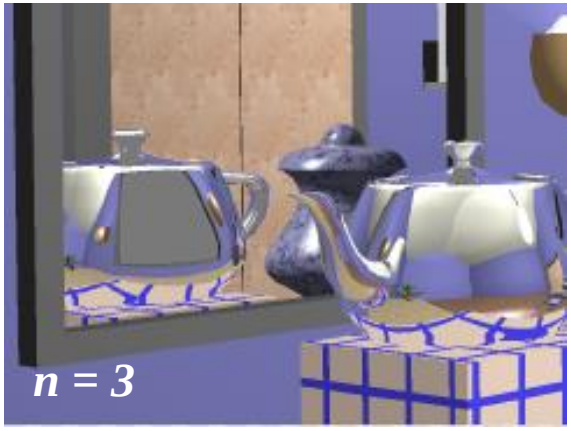

15
7
6
14
11
13
10
12
3
4
5
9
8
2
1

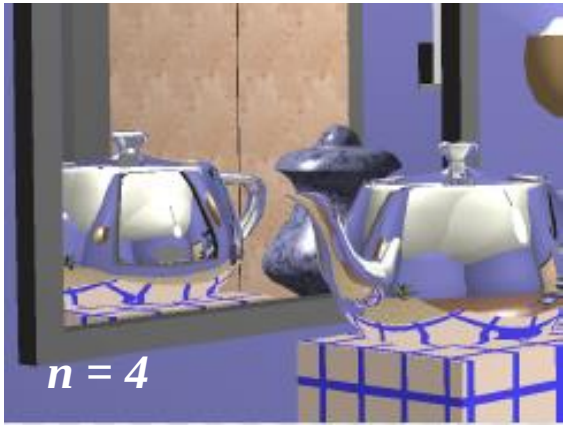

intersection point 1
intersection point 2
intersection point 3
4
5
10
11
6
7
8
9
12
13
14
15

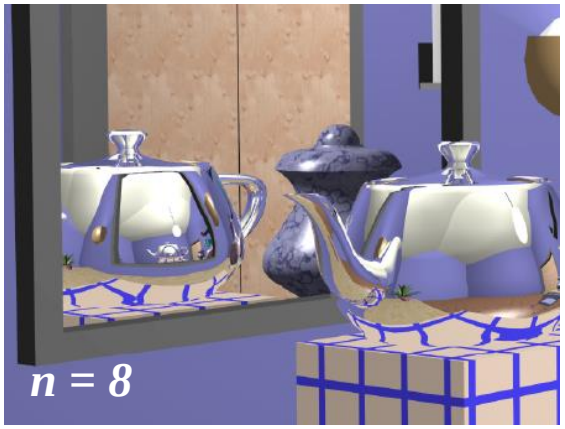
The Effect of Recursion Depth











The Basic Algorithm – Pseudo Code for View Rays

```
CalcImage
{
        For (y=0; y<YRES; y++)
        {
                For (x=0; x<XRES; x++)
                {
                        ViewRay.Start = ViewPoint
                        ViewRay.Dir = CalcViewDir(x,y,ViewPoint);
                        Colour = TraceRay(ViewRay,0);
                        Plot(x,y,Colour);
                }
        }
}
```

The Basic Algorithm – Pseudo Code for Ray Tracer

```
Colours TraceRay (ray: Ray, depth: int)
{
        if (depth>MAXDEPTH) return black
        else
        {
                Object, IntersectionPoint =     Schneide Strahl mit allen Objekten und
                                                ermittle nächstgelegenen Schnittpunkt;
                if (NoIntersection) return background_color
                else
                {
                        LocalColour = Anteile der sichtbaren Lichtquellen;
                        ReflectedRay = CalcReflectedRay(ray,IntersectionPoint,Object);
                        RefractedRay = CalcRefractedRay(ray,IntersectionPoint,Object);
                        ReflectedColour = TraceRay (ReflectedRay, depth+1);
                        RefractedColour = TraceRay (RefractedRay, depth+1);
                        return combine(LocalColour, ReflectedColour, RefractedColour);
                }
        }
}
```

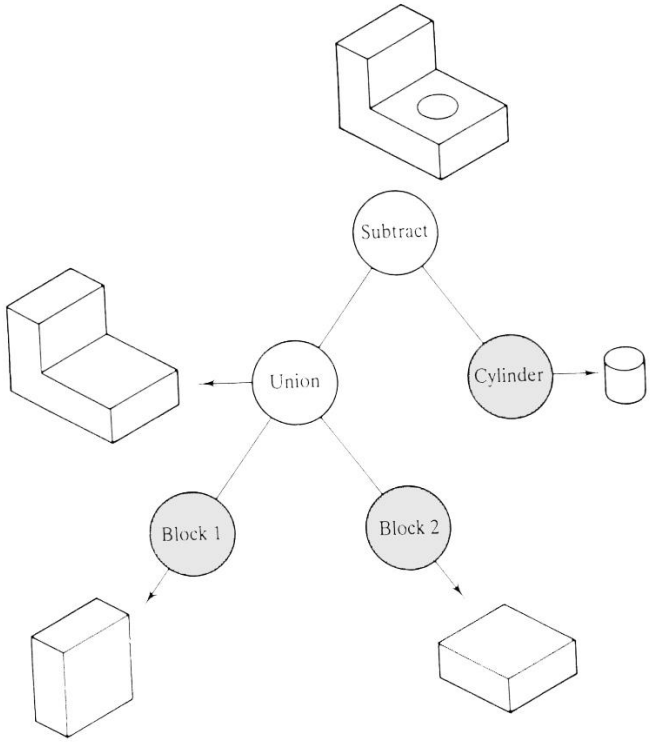
?

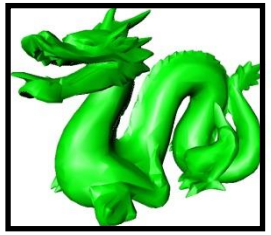
?

?

Intersections – Object Geometries

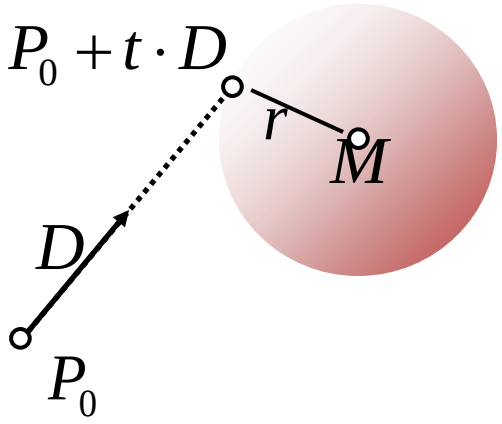
- Spheres
- Planes
- Polygons
- Polyhedrons
- Cubes
- Quadrics (cylinders, cones, ellipsoids, ...)
- Compound objects
  (e.g., Constructive Solid Geometry)
- Strategy: Approximate complex geometries by bounding volumes
  „Good“ volumes: spheres, cubes





Intersections – Sphere

$$P_S = P_0 + t \cdot D$$



$$(1)\ x_S = x_0 + t \cdot x_D, y_S = y_0 + t \cdot y_D, z_S = z_0 + t \cdot z_D$$

$$(2)\ (x_S - x_M)^2 + (y_S - y_M)^2 + (z_S - z_M)^2 = r^2$$

(1) in (2):

$$(x_0 + t \cdot x_D - x_M)^2 + (y_0 + t \cdot y_D - y_M)^2 + (z_0 + t \cdot z_D - z_M)^2 = r^2$$

$$A \cdot t^2 + B \cdot t + C = 0 \text{ with}$$

$$A = x_D^2 + y_D^2 + z_D^2 = 1 (D \text{ normalized})$$

$$B = 2\left(x_d (x_0 - x_M) + y_d (y_0 - y_M) + z_d (z_0 - z_M)\right)$$

$$C = (x_0 - x_M)^2 + (y_0 - y_M)^2 + (z_0 - z_M)^2 - r^2$$

Intersections – Sphere (cont.)

$$t_{0/1} = \frac{-B \pm \sqrt{B^2 - 4C}}{2}$$

- Select smaller t
- If $B^2 - 4C < 0$, the ray misses the sphere
- Intersection point $P_S = (x_S, y_S, z_S) = (x_0 + t \cdot x_D, y_0 + t \cdot y_D, z_0 + t \cdot z_D)$
- Normal vector $N_S = \left(\frac{x_S - x_M}{r}, \frac{y_S - y_M}{r}, \frac{z_S - z_M}{r} \right)$

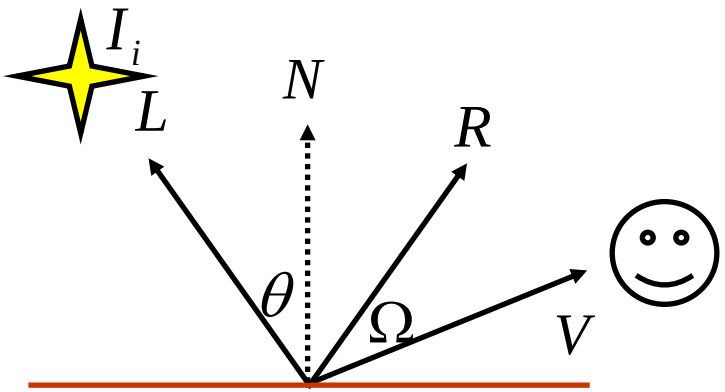
- Overhead:17 additions, 17 multiplications, 1 square root, 3 compare
- Optimization: Check first, whether ray shows away from the sphere

The Basic Algorithm – Pseudo Code for Ray Tracer

```
Colours TraceRay (ray: Ray, depth: int)
{
        if (depth>MAXDEPTH) return black
        else
        {
                Object, IntersectionPoint =     Schneide Strahl mit allen Objekten und
                                                ermittle nächstgelegenen Schnittpunkt;
                if (NoIntersection) return background_color
                else
                {
                        LocalColour = Anteile der sichtbaren Lichtquellen;   ?
                        ReflectedRay = CalcReflectedRay(ray,IntersectionPoint,Object);
                        RefractedRay = CalcRefractedRay(ray,IntersectionPoint,Object);
                        ReflectedColour = TraceRay (ReflectedRay, depth+1);
                        RefractedColour = TraceRay (RefractedRay, depth+1);
                        return combine(LocalColour, ReflectedColour, RefractedColour);
                }
        }
}
```

Phong Reflection Model

- Linear combination from 3 components:
  - Diffuse
  - Specular
  - Ambient



- Reflection coefficients: k_d, k_s, k_a
- Diffuse component: $I_d = I_i k_d \cos\theta = I_i k_d (L \cdot N)$

$$I_d = k_d \sum_n I_{i,n} (L_n \cdot N)$$

- Specular component: $I_s = I_i k_s \cos^n \Omega = I_i k_s (R \cdot V)^n$

  n: Index for surface roughness

  Perfect mirror: $n \to \infty$ (Ray Tracing: Recursion)

- Ambient component: $I_g = I_a k_a$
- Overall intensity: $I = I_a k_a + I_i (k_d (L \cdot N) + k_s (R \cdot V)^n)$

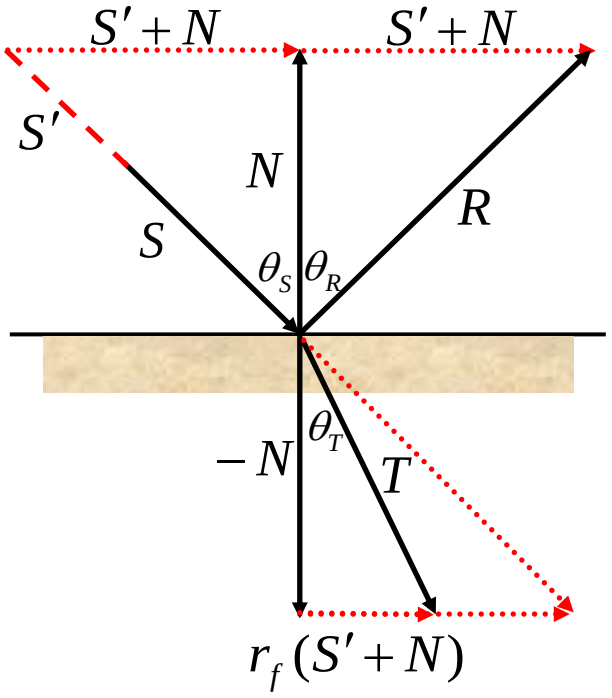
The Basic Algorithm – Pseudo Code for Ray Tracer

```
Colours TraceRay (ray: Ray, depth: int)
{
        if (depth>MAXDEPTH) return black
        else
        {
                Object, IntersectionPoint =     Schneide Strahl mit allen Objekten und
                                                ermittle nächstgelegenen Schnittpunkt;
                if (NoIntersection) return background_color
                else
                {
                        LocalColour = Anteile der sichtbaren Lichtquellen;
                        ReflectedRay = CalcReflectedRay(ray,IntersectionPoint,Object);
                        RefractedRay = CalcRefractedRay(ray,IntersectionPoint,Object);
                        ReflectedColour = TraceRay (ReflectedRay, depth+1);
                        RefractedColour = TraceRay (RefractedRay, depth+1);
                        return combine(LocalColour, ReflectedColour, RefractedColour);
                }
        }
}
```

?

Calculation of Reflection & Transmission Rays

- Sight (view) vector S
- Reflection vector R
- Transmission vector T
- Normal vector N



- $R: \theta_S = \theta_R$ bzw. $\cos\theta_S = \cos\theta_R \quad (-S \cdot N = N \cdot R)$
- $T: \dfrac{\sin\theta_S}{\sin\theta_T} = \dfrac{\eta_1}{\eta_2}$ Snell's Law
- Algebraic solution $R/T = \alpha S + \beta N$
- Geometrical solution

$$S' = S / S \cdot N$$

$$R = S' + 2N$$

$$T = r_f(S'+N) - N \quad \text{mit } r_f = \sqrt{\left(\frac{\eta_1}{\eta_2}\right)^2 |S'|^2 - |S'+N|^2}^{-1}$$

Algebraic Solution for Transmission Rays

Equation 1:

$$
\begin{aligned}
\frac{\sin\theta_S}{\sin\theta_T} &= \frac{\eta_1}{\eta_2} = \eta \\
\sin\theta_S \eta &= \sin\theta_T \\
\sin^2\theta_S \eta^2 &= \sin^2\theta_T \\
\left(1-\cos^2\theta_S\right)\eta^2 &= \left(1-\cos^2\theta_T\right) \\
\left(1-\cos^2\theta_S\right)\eta^2 - 1 &= \cos^2\theta_T \\
&= \left[-N\cdot T\right]^2 \\
&= \left[-N\cdot\left(\alpha S+\beta N\right)\right]^2 \\
&= \left[\alpha\left(-N\cdot S\right)+\beta\left(-N\cdot N\right)\right]^2 \\
&= \left[\alpha\cos\theta_S-\beta\right]^2
\end{aligned}
$$

Equation 2:

$$
\begin{aligned}
1 &= T\cdot T \\
&= \left(\alpha S+\beta N\right)\cdot\left(\alpha S+\beta N\right) \\
&= \alpha^2\left(I\cdot I\right)+2\alpha\beta\left(I\cdot N\right)+\beta^2\left(N\cdot N\right) \\
&= \alpha^2-2\alpha\beta\cos\theta_S+\beta^2
\end{aligned}
$$

Find solution for α and β, and insert into $T=\alpha S+\beta N$

The Basic Algorithm – Pseudo Code for Ray Tracer

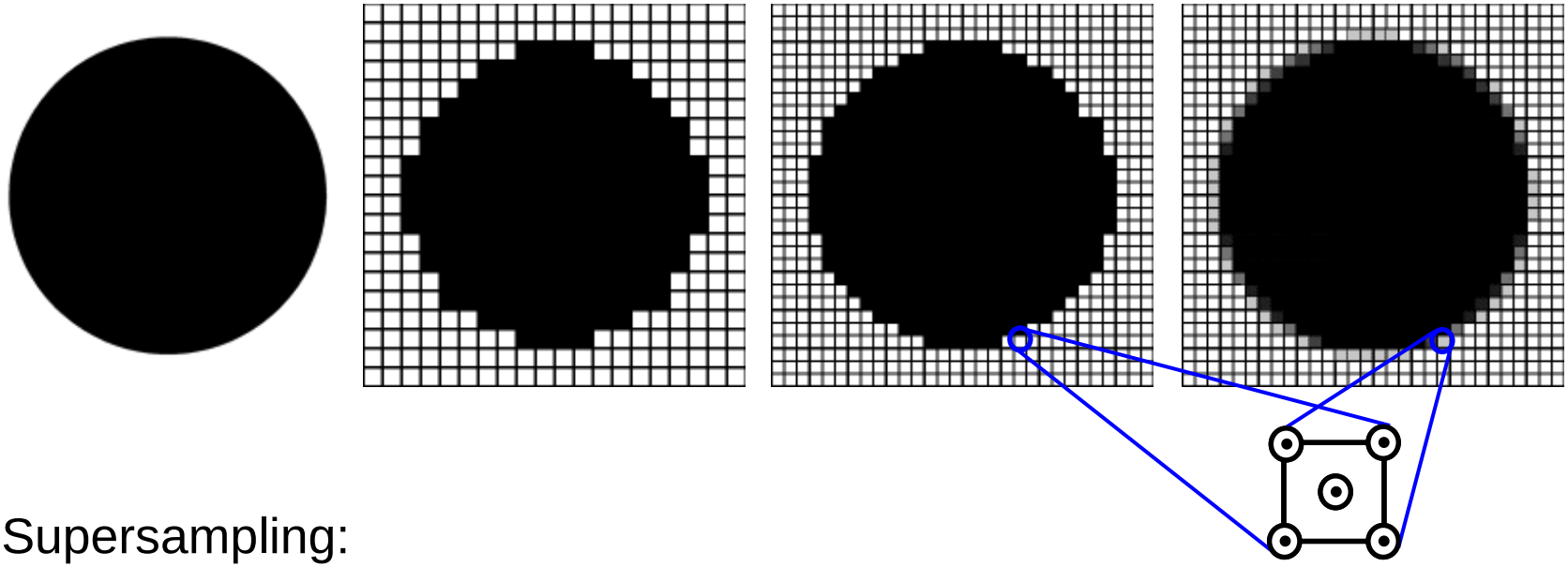
```
Colours TraceRay (ray: Ray, depth: int)
{
        if (depth>MAXDEPTH) return black
        else
        {
                Object, IntersectionPoint =     Schneide Strahl mit allen Objekten und
                                                ermittle nächstgelegenen Schnittpunkt;
                if (NoIntersection) return background_color
                else
                {
                        LocalColour = Anteile der sichtbaren Lichtquellen;
                        ReflectedRay = CalcReflectedRay(ray,IntersectionPoint,Object);
                        RefractedRay = CalcRefractedRay(ray,IntersectionPoint,Object);
                        ReflectedColour = TraceRay (ReflectedRay, depth+1);
                        RefractedColour = TraceRay (RefractedRay, depth+1);
                        return combine(LocalColour, ReflectedColour, RefractedColour);
                }
        }
}
```

Topics

- What is Ray Tracing?
- The Basic Algorithm
- Intersections
- Light and Reflection
- Reflection- and Transmission Rays
- Optimization
  - Quality
  - Speed

Optimization - Quality

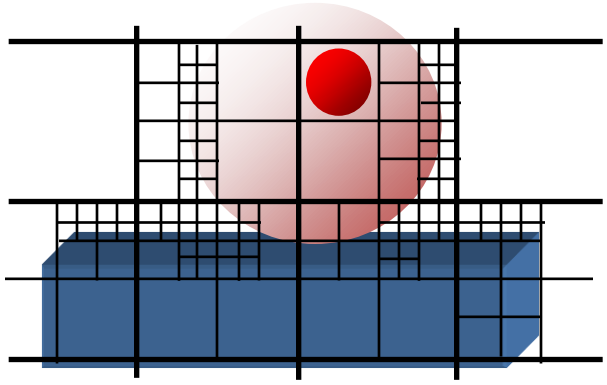
- Anti-Aliasing

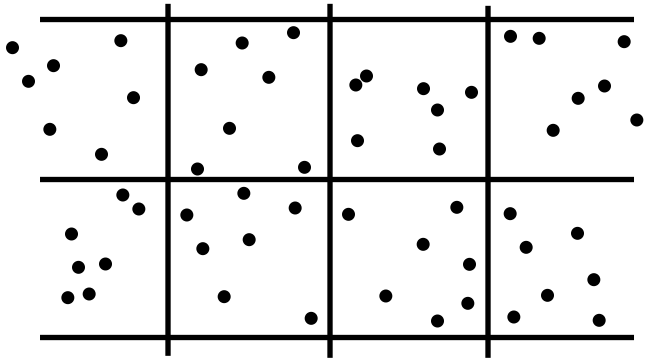


- Supersampling:
  - Treat pixels as a plane
  - Send more rays through each pixel
  - Averaging the intensities of single rays leads to pixel color

Optimization – Quality (cont.)

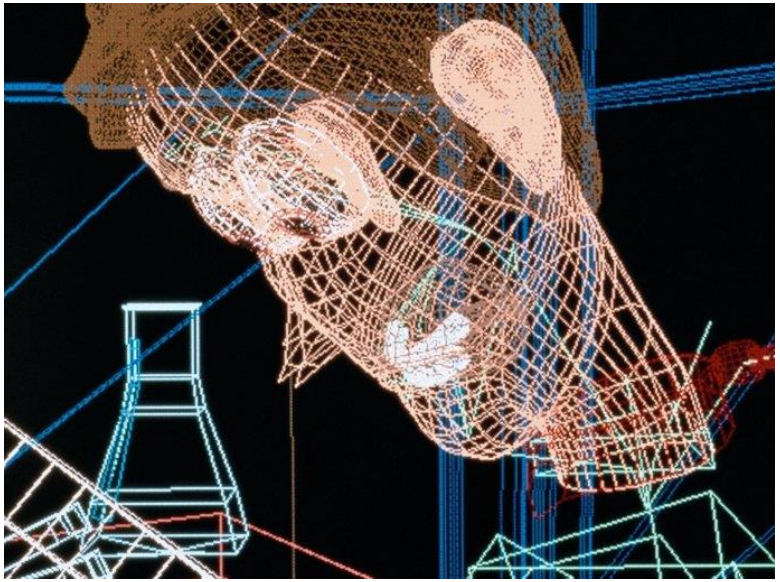
- Anti-Aliasing
  - Adaptive Supersampling:
    - First look at the rays at the 4 pixel corners
    - Does the intensity difference of adjacent rays exceed threshold?
    - Recursive grid refinement (Quad-Trees)
  - Stochastic Ray Tracing:
    - Irregular but uniformly dense distribution of rays





Toy Story

- First completely computer-animated movie in 1995
- „Rendering Farm“: 117 computers
- Computation of a single image: from 45 minutes up to 30 hours



Optimization - Speed

Ray Tracing is slow!

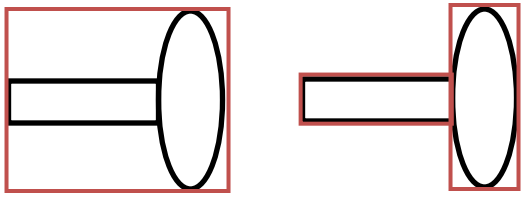
- Standard resolution SXGA (1280x1024): 1.3 Mio pixels
- Small scene with 50 objects: 65 Mio intersection points must be determined in the first step
- Algorithm complexity increases exponentially with recursion depth (in case of „specular“ objects)
- Shadow feelers cause additional intersection points (number of light sources?)
- Situation is even harder for Distributed Ray Tracing and Anti-Aliasing

- Make use of parallelism!
- Optimize the algorithm for the calculation of single intersection points
- Reduce the number of intersection points

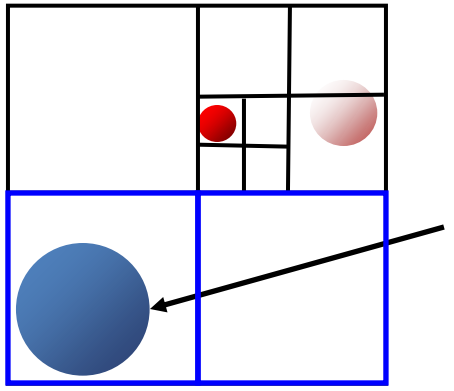
Optimization – Speed (II)

- Optimize the intersection point algorithm

- Bounding Volumes
  - TradeOff: Accuracy of approximation versus complexity of intersection point calculation
  - Option: Stepwise approximation



- Space subdivision
  - Octrees: Nearly the same complexity in the leaves

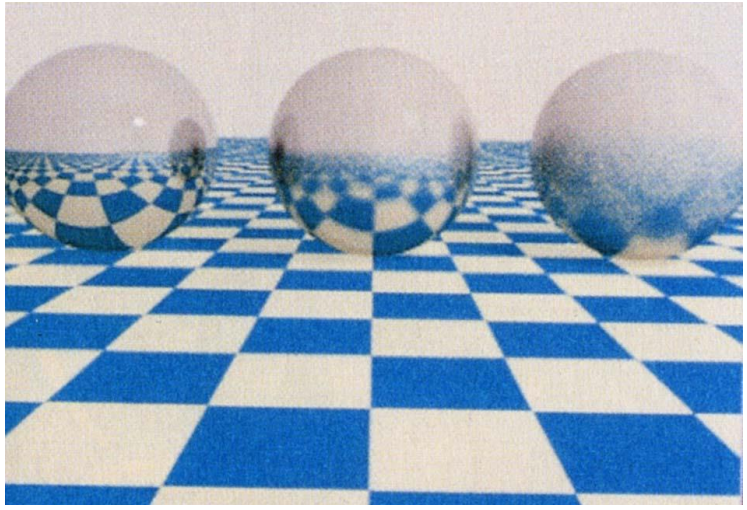


- Make use of the coherency for adjacent rays
  - View rays
  - Rays ending at the same plane (see polygons!)

See chapter on collision detection!

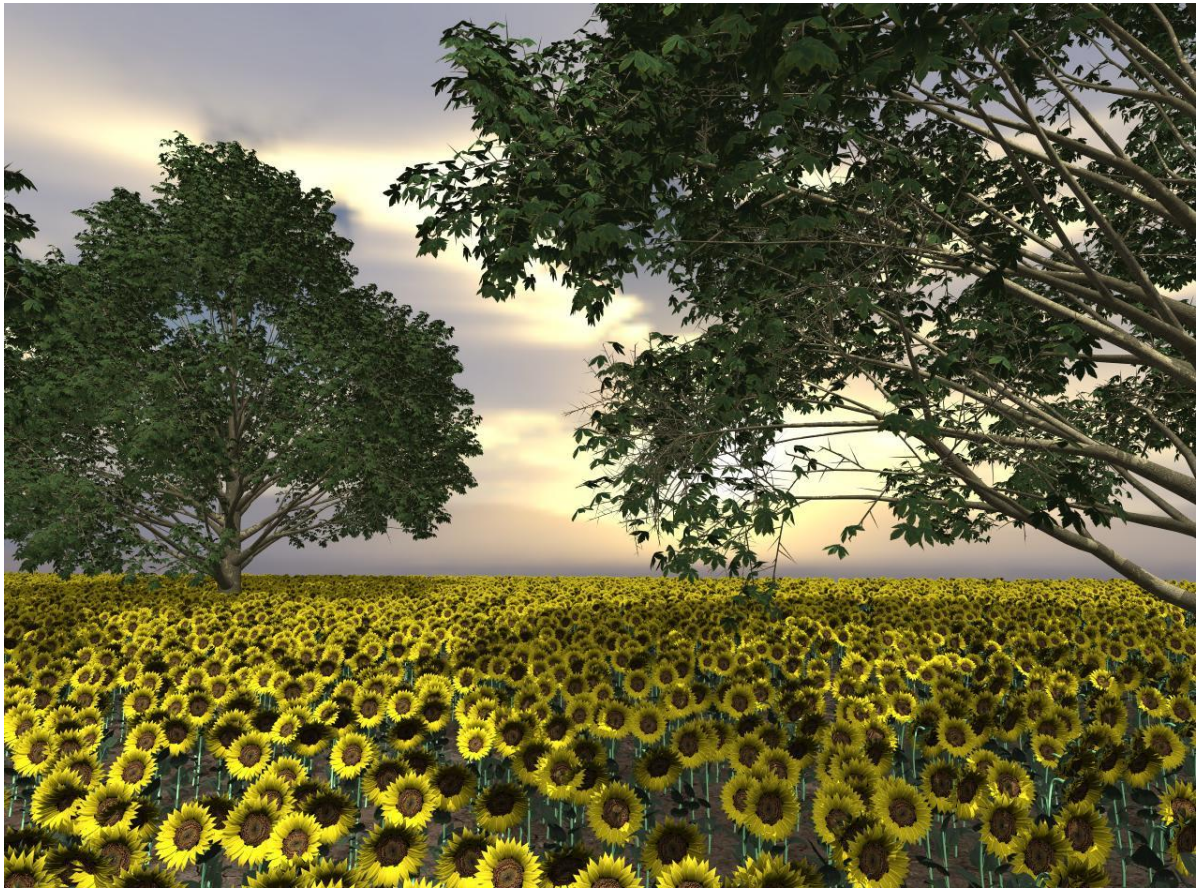
Summary

- ✓ Elegant, recursive algorithm for global illumination
- ✓ Reflection, transmission, occlusion culling, hidden surface removal and shadows
- ✓ Very realistic images if there are many specular and transparent objects in the scene



- Local modeling of diffuse reflection
  - ➢ Distributed Ray Tracing
  - ➢ Radiosity
  - ➢ Plausibility versus authenticity: see Radiance!
- Ray Tracing is slow
  - ➢ Special hardware for Ray Tracing (e.g., Advanced Rendering Technologies, Cambridge)
  - ➢ Anyway: Ray Tracing may be even faster than traditional shading for scenes with millions of polygons and a lot of occlusion!

Real Time Ray Tracing



A. Dietrich, I. Wald, P. Slusallek (scene data from O. Deussen): OpenRT

28.000 sun flowers, 1 Billion polygons, 6 frames/sec (640x480 pixels), PC-Cluster (24 nodes)

Literature

- Where to find the images used in the slides:
  - Foley, van Dam, Feiner, Hughes: Computer Graphics: Principles and Practice. Addison Wesley, 1992
  - Glassner: An Introduction to Ray Tracing, Morgan Kaufmann Publishers, 2000
  - Tönnies, Lemke: 3D-Computergrafische Darstellungen, Oldenbourg Verlag, 1994
  - Watt: 3D Computer Graphics, Addison Wesley, 1992
  - Whitted: An improved illumination model for shaded display, Comm. of the ACM 23(6), 1980
  - http://www.gris.informatik.tu-darmstadt.de/lehre/vorl_ueb/gdvII/slides/rr-bw.pdf
  - http://www2.inf.fh-bonn-rhein-sieg.de/~ahinke2m/Vorlesung/CGVI03/