A Brief Refreshing Course of Computer Graphics

Tobias Isenberg

The Overall Goal: Photorealism

Gilles Tran, using POV-Ray 3.6

The Second Goal: Speed and Efficiency

Doom (1993), by id Software

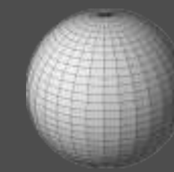
The Second Goal: Speed and Efficiency

Ryse (2013); by Crytek/Microsoft Studios

Computer Graphics 101

- computer graphics pipeline

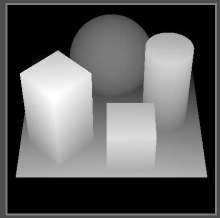
- models and object representation



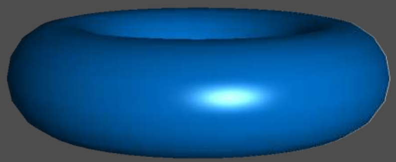
- transformations and projections

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

- hidden surface removal

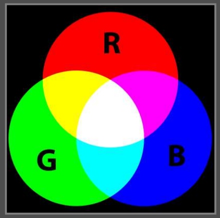


- illumination and shading



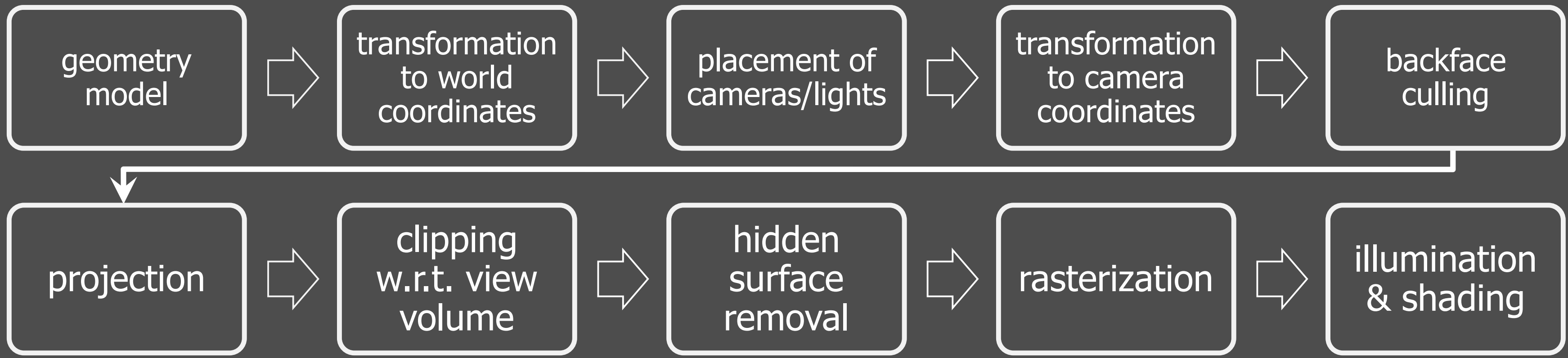
- texture mapping

- color and color models



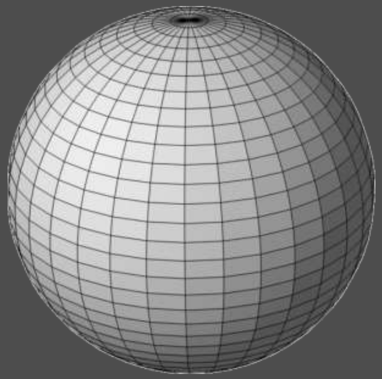
Computer Graphics Pipeline

Computer Graphics/Rendering Pipeline



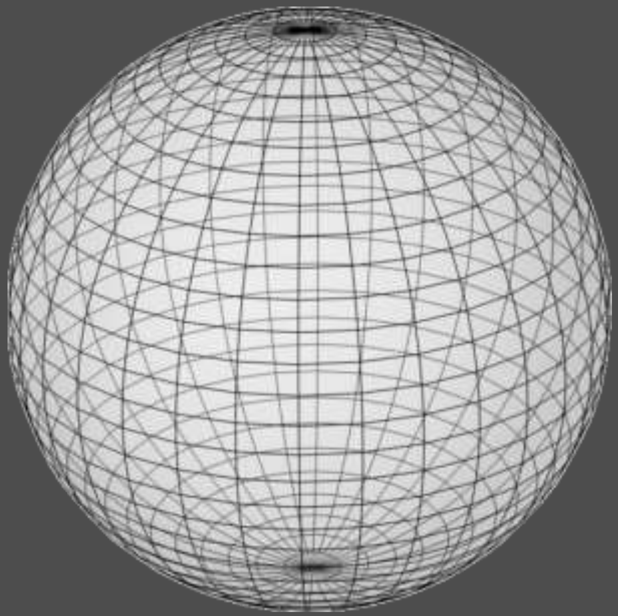
- simple model of physical processes
- independent parallel processing of triangles
- control/parameterization of the individual stages
- implementation of the specific stages in hardware

Models and Object Representation



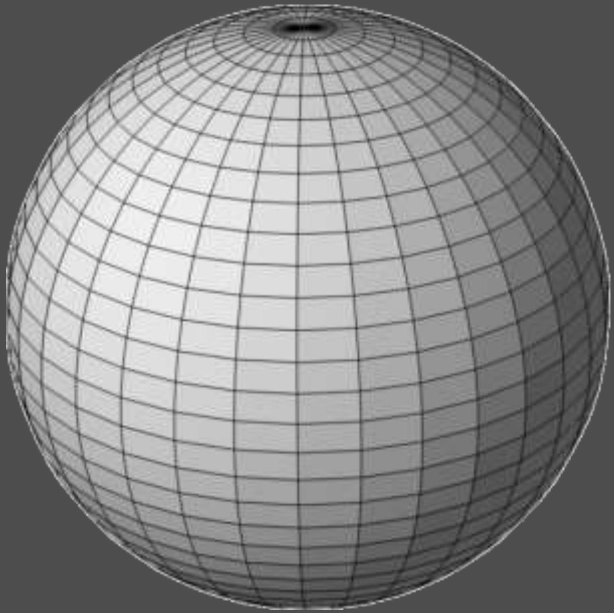
What to Specify for a 3D Shape?

- geometry
  - shapes, positions
  - connectivity, inside/outside
- material properties
  - visuals, textures (plastic, wood, metal, etc.)
  - other material properties (elasticity, mass, etc.)
- behaviour/animation
- more depending on the specific application



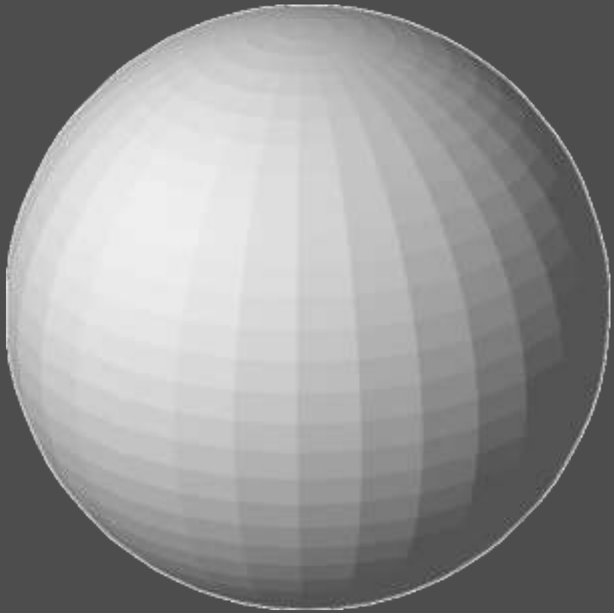
What to Specify for a 3D Shape?

- geometry
  - shapes, positions
  - connectivity, inside/outside
- material properties
  - visuals, textures (plastic, wood, metal, etc.)
  - other material properties (elasticity, mass, etc.)
- behaviour/animation
- more depending on the specific application



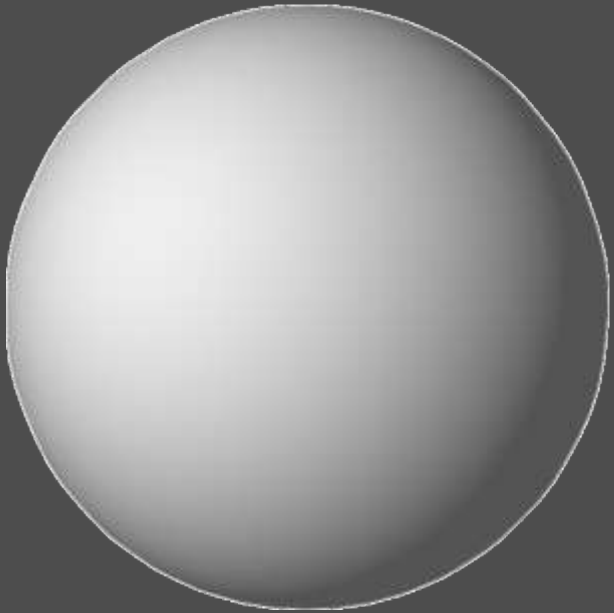
What to Specify for a 3D Shape?

- geometry
  - shapes, positions
  - connectivity, inside/outside
- material properties
  - visuals, textures (plastic, wood, metal, etc.)
  - other material properties (elasticity, mass, etc.)
- behaviour/animation
- more depending on the specific application



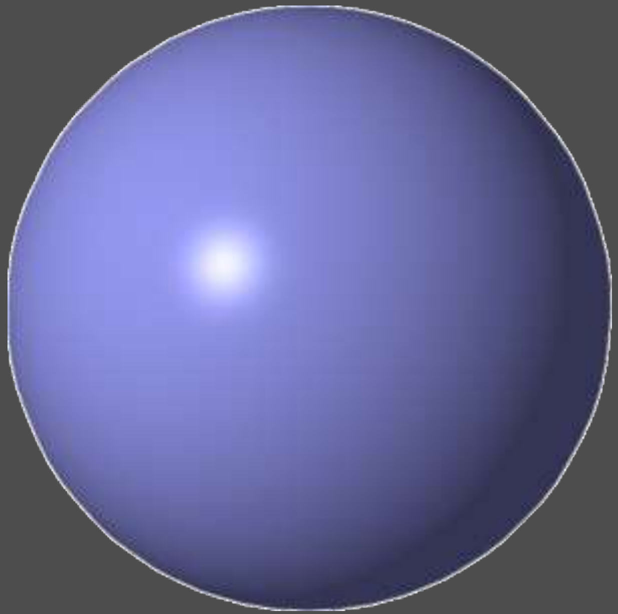
What to Specify for a 3D Shape?

- geometry
  - shapes, positions
  - connectivity, inside/outside
- material properties
  - visuals, textures (plastic, wood, metal, etc.)
  - other material properties (elasticity, mass, etc.)
- behaviour/animation
- more depending on the specific application



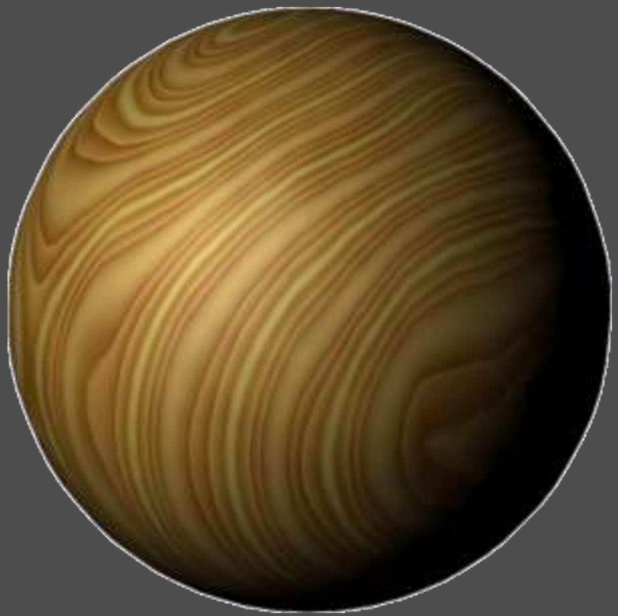
What to Specify for a 3D Shape?

- geometry
  - shapes, positions
  - connectivity, inside/outside
- material properties
  - visuals, textures (plastic, wood, metal, etc.)
  - other material properties (elasticity, mass, etc.)
- behaviour/animation
- more depending on the specific application



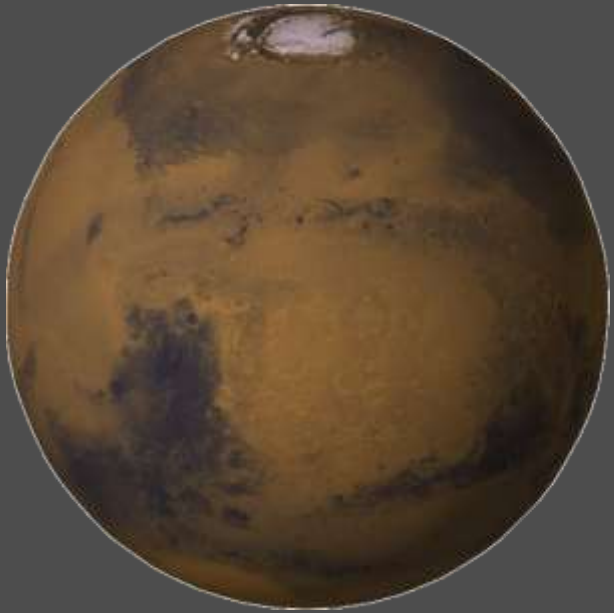
What to Specify for a 3D Shape?

- geometry
  - shapes, positions
  - connectivity, inside/outside
- material properties
  - visuals, textures (plastic, wood, metal, etc.)
  - other material properties (elasticity, mass, etc.)
- behaviour/animation
- more depending on the specific application



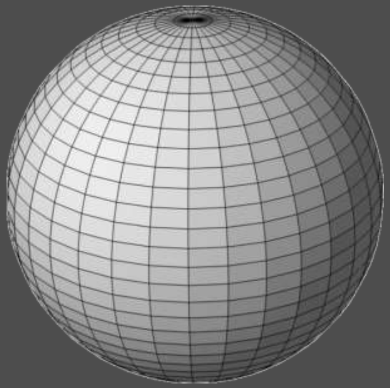
What to Specify for a 3D Shape?

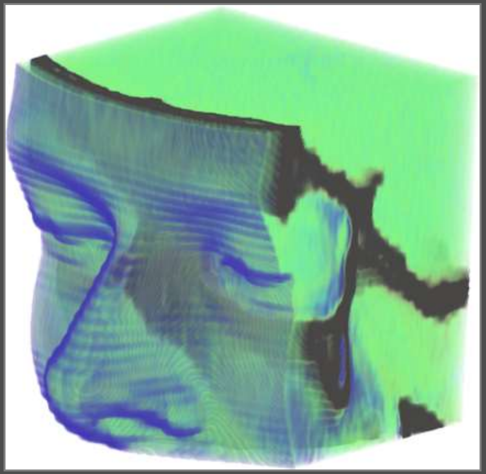
- geometry
  - shapes, positions
  - connectivity, inside/outside
- material properties
  - visuals, textures (plastic, wood, metal, etc.)
  - other material properties (elasticity, mass, etc.)
- behaviour/animation
- more depending on the specific application



How to Specify a 3D Geometry?

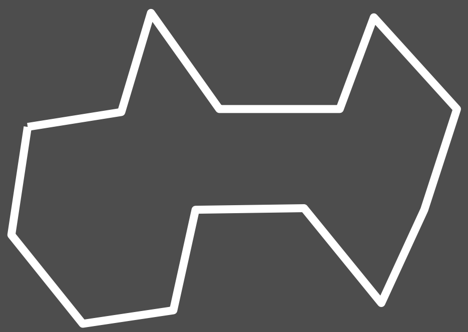
- boundary representations (b-reps)
  - meshes
  - piecewise smooth surfaces
- volume representations
  - voxel models
  - implicit surfaces
  - CSG: constructive solid geometry
  - space partitioning
    - BSP trees: binary space partitioning
    - octrees





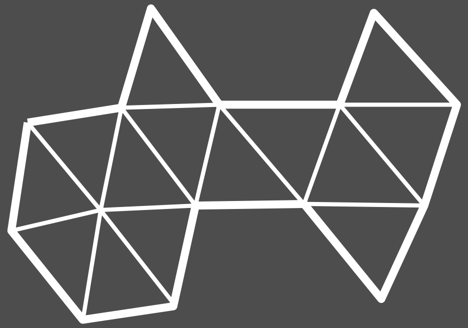
B-Reps: Polygonal Meshes

- polygons to define the surface of objects



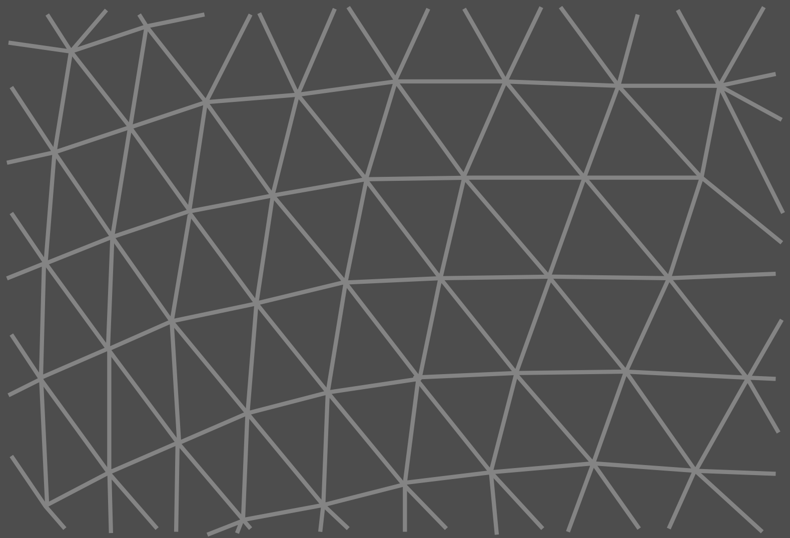
B-Reps: Polygonal Meshes

- polygons to define the surface of objects
- triangle meshes
  - polygon with fewest vertices
  - always convex & planar → defines unique surface



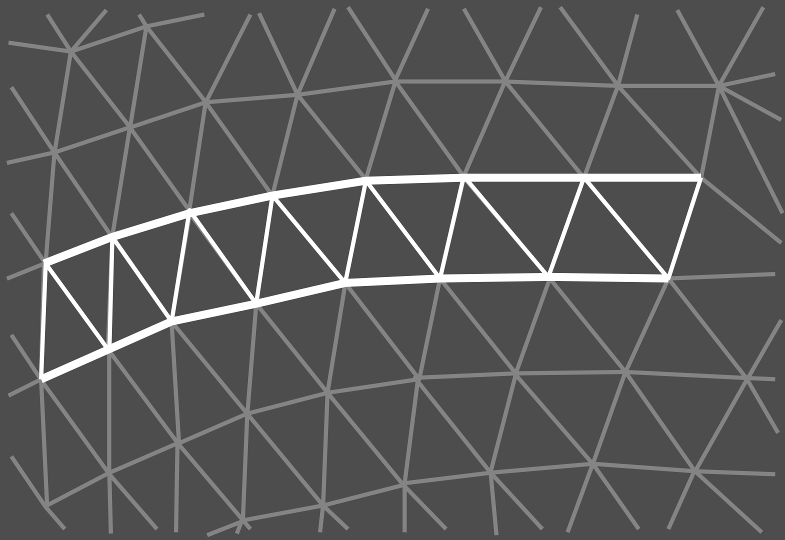
B-Reps: Polygonal Meshes

- polygons to define the surface of objects
- triangle meshes
  - polygon with fewest vertices
  - always convex & planar → defines unique surface



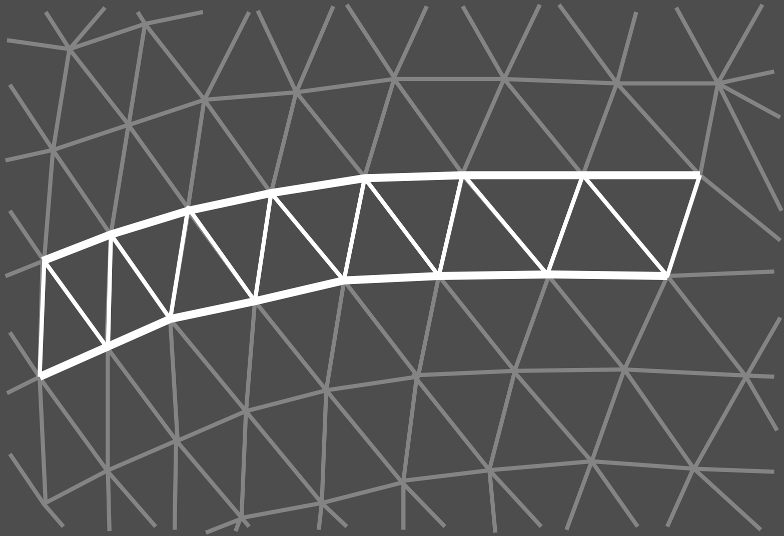
B-Reps: Polygonal Meshes

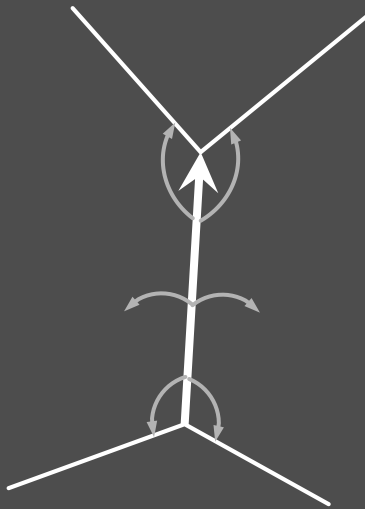
- polygons to define the surface of objects
- triangle meshes
  - polygon with fewest vertices
  - always convex & planar → defines unique surface
- triangle strips: faster rendering



B-Reps: Polygonal Meshes

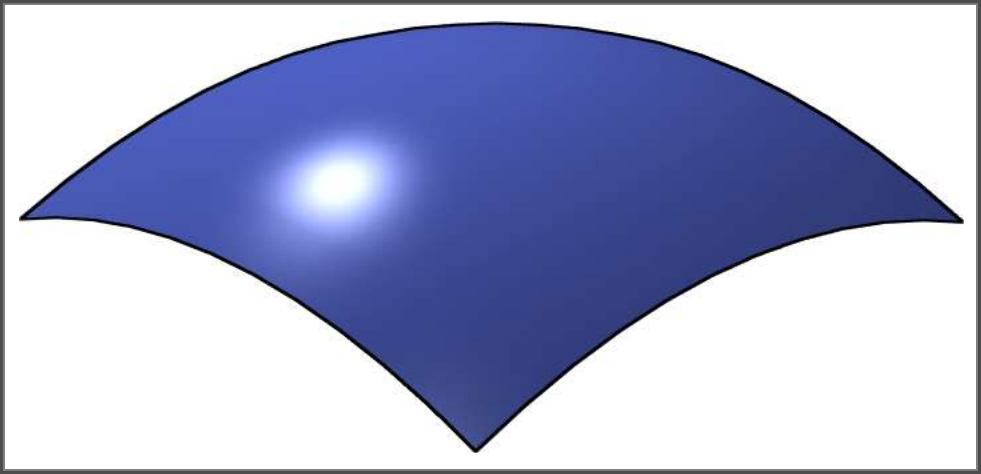
- polygons to define the surface of objects
- triangle meshes
  - polygon with fewest vertices
  - always convex & planar → defines unique surface
- triangle strips: faster rendering
- more complex mesh data structures (e.g., Winged Edge)





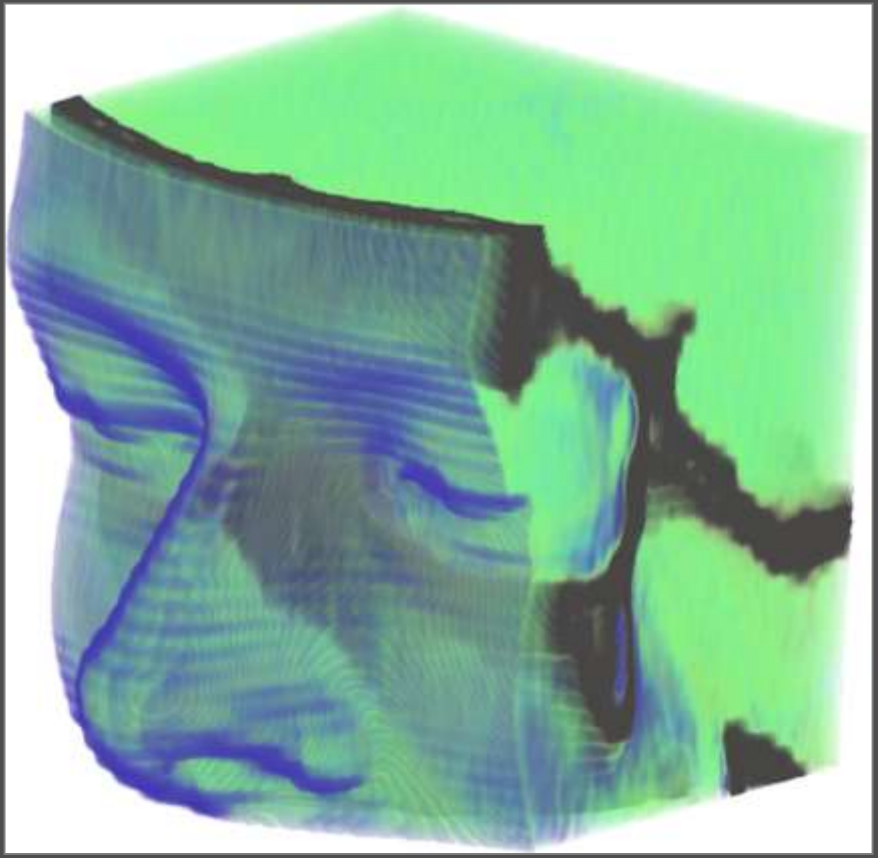
B-Reps: Piecewise Smooth Surfaces

- surface constructed from patches
- patches can be curved and are smooth
- patches satisfy a continuity constraint
- e.g., Bézier, Spline, NURBS surfaces



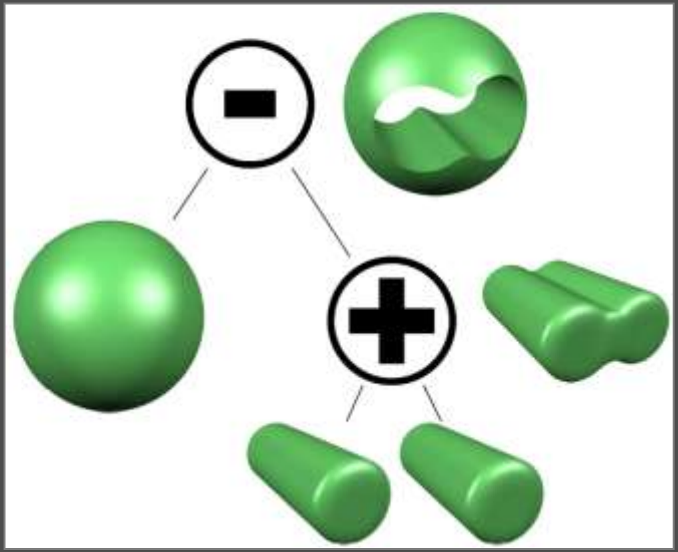
Volumes: Voxel Models

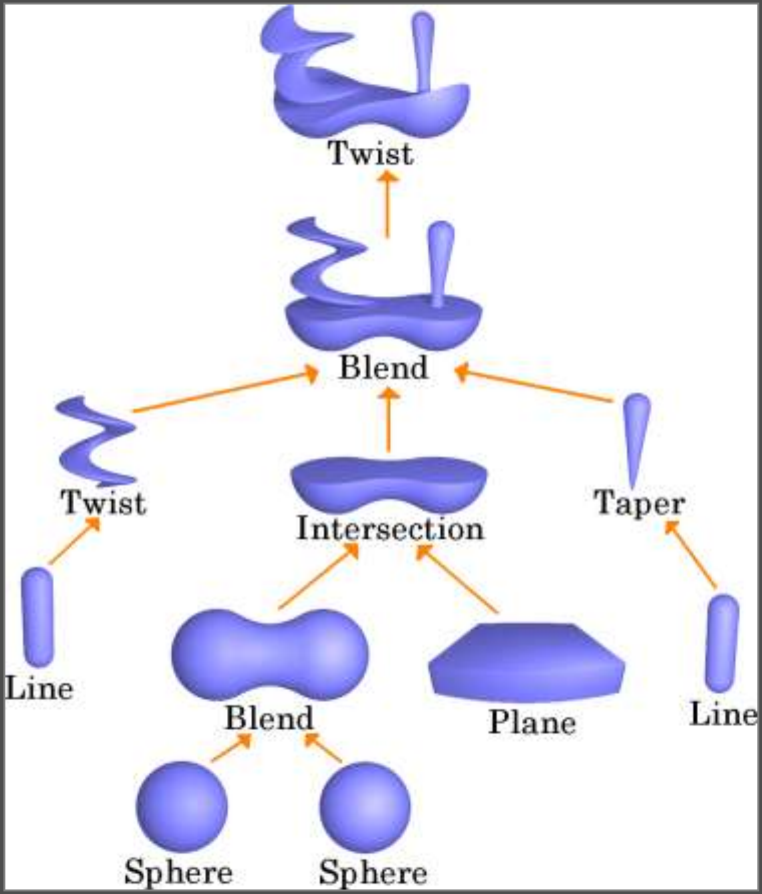
- sampling of a volume in regular intervals
- samples as cubes, or as general boxes
- several properties can be sampled
- shapes: *iso-values* define *iso-surfaces*
- heavily used in medical imaging; based on CT, MRI



Volumes: Construct. Solid Geometry

- Boolean operators to combine shapes
- unions, intersections, set differences
- build up CSG trees

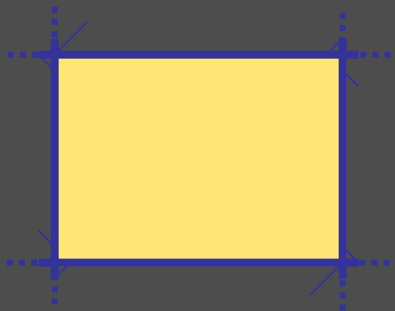


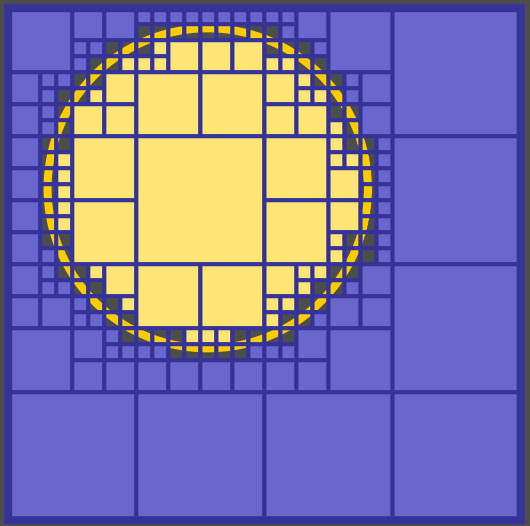


courtesy of Erwin de Groot and Brian Wyvill

Volumes: BSP and Octrees

- space partitioning: define sub-spaces
- **b**inary **s**pace **p**artitioning: half-spaces
  - planes define borders between inside and outside; hierarchy
  - only current subspace affected
- octrees: partitioning on regular 2×2×2 grid (= 8)
  - mark cells inside, outside, or subdivide
  - 2D case: quadtrees (2×2)





Transformations

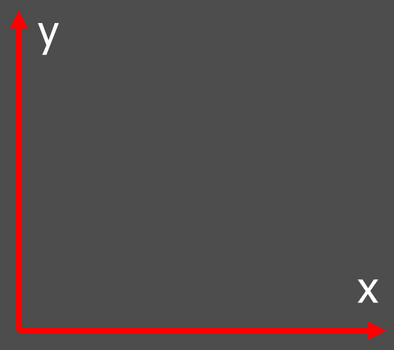
$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

Transformations

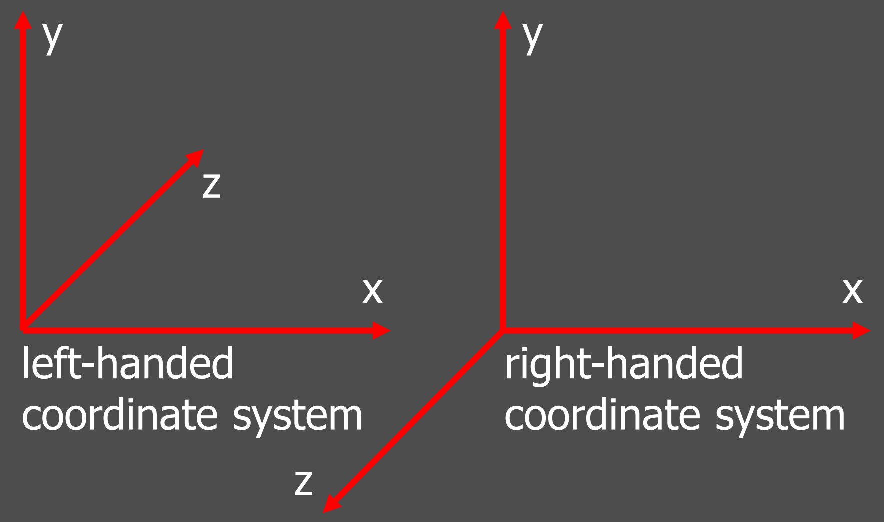
- 3D models represented as points, edges, polygons in 3D coordinate systems
  - object coordinate systems
  - world coordinate system
  - camera coordinate system
  - screen coordinate system
- transformations necessary
- foundation: geometry and linear algebra

Coordinate Systems in CG: 2D & 3D

- two-dimensional



- three-dimensional
  - 2 mirrored systems
  - cannot be matched by rotation
  - we use right-handed

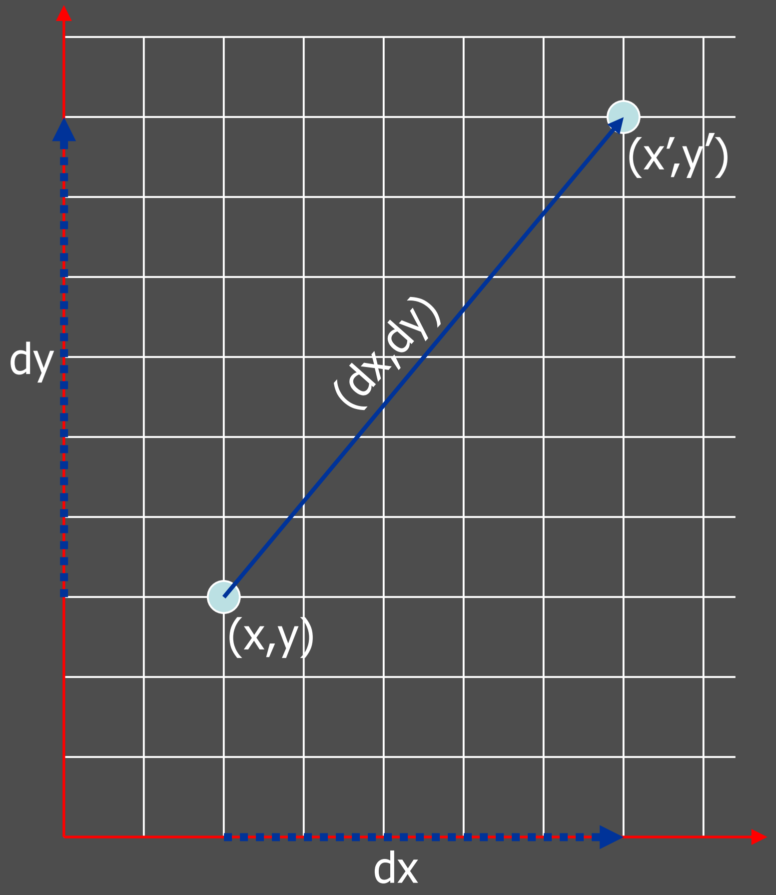


2D Translation

- move point $(x, y)^T$ on a straight line to $(x', y')^T$
- represent translation by a translation vector that is added

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} dx \\ dy \end{pmatrix}$$

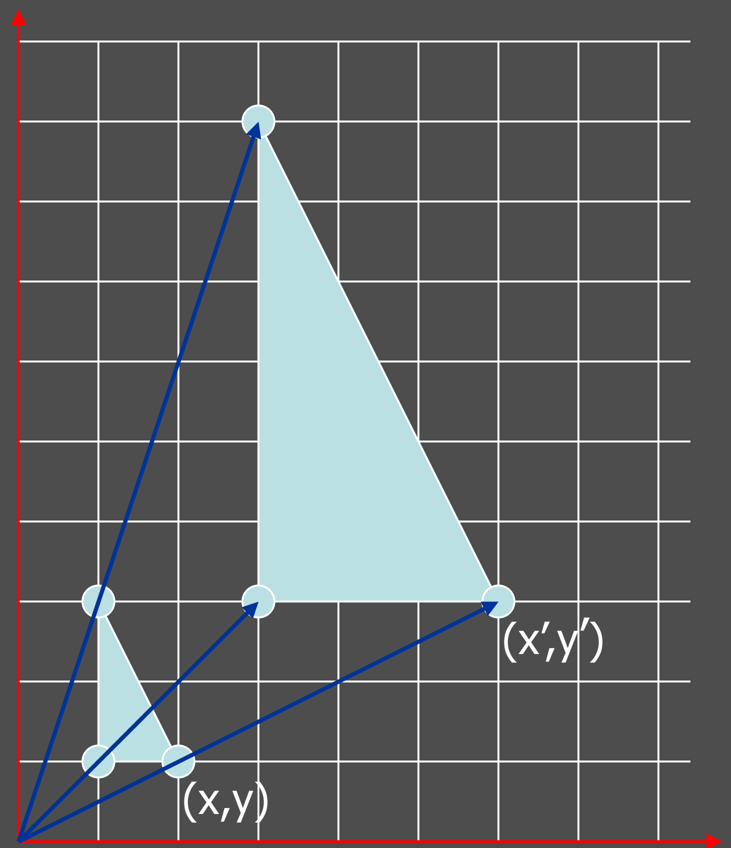
- vector: movement from one point to another



2D Uniform Scaling

- center of scaling is o
- scaling uniformly in all directions
- stretching of $(x, y)^T$'s position vector by scalar factor α to get $(x', y')^T$
- mathematically: multiplication with α

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \alpha \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \alpha x \\ \alpha y \end{pmatrix}$$

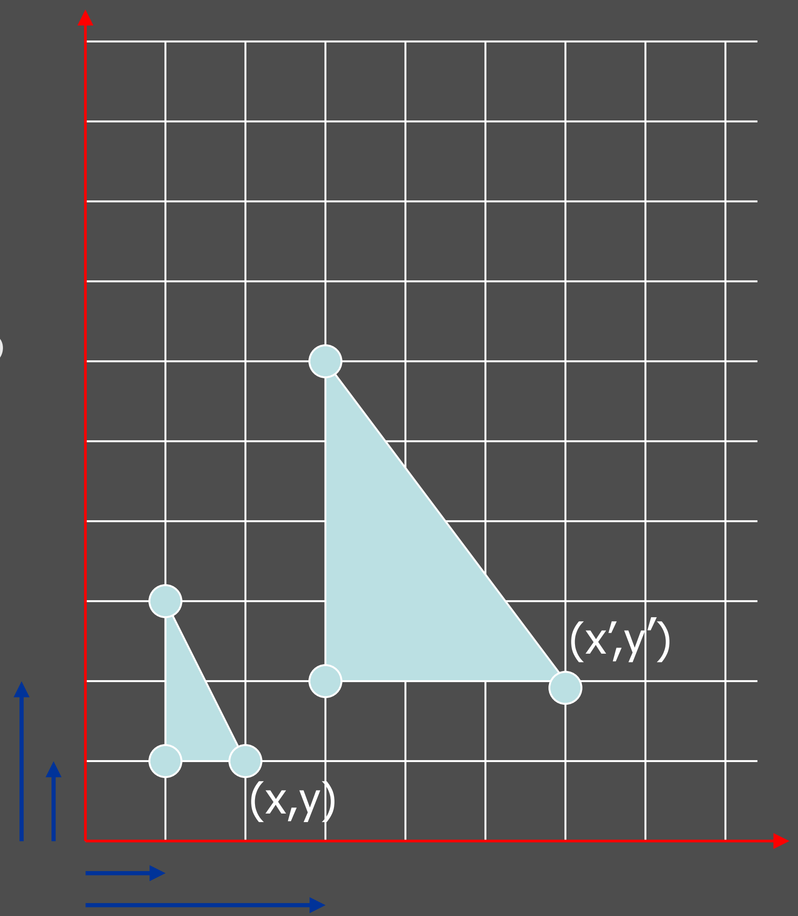


2D Non-Uniform Scaling

- center of scaling is o
- scaling in *x*-direction by α and in y-direction by β (scaling vector $(\alpha, \beta)^T$)
- mathematically: multiplication with α and β according to axis

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha x \\ \beta y \end{pmatrix}$$

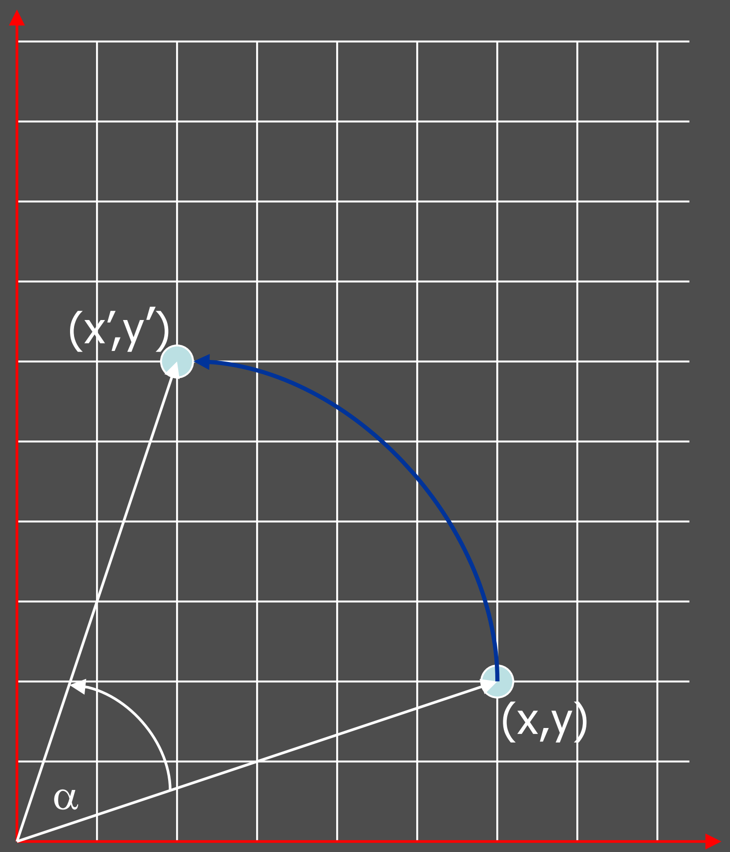
- application: mirroring



2D Rotation

- center of rotation is o
- point $(x, y)^T$ is rotated by an angle α around o to obtain $(x', y')^T$
- positive angles α mean counter-clockwise rotation
- $x' = x \cos\alpha - y \sin\alpha$
  $y' = x \sin\alpha + y \cos\alpha$
- matrix multiplication:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



Transformations in 2D: Results

- translation: addition of translation vector
  scaling: multiplication of factor(s)
  rotation: matrix multiplication
- problems:
  - non-uniform treatment of transformations
  - no way to combine N transformation into one
- idea: all transformations as matrix multiplications!
- only scaling and translation to do

Scaling using 2D Matrix

- general 2D matrix multiplication format

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- scaling formula

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha x \\ \beta y \end{pmatrix} \text{ (possibly with } \alpha = \beta \text{)}$$

- scaling as matrix multiplication

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Translation using 2D Matrix

- general matrix multiplication format

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- tanslation formula

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} dx \\ dy \end{pmatrix}$$

- translation as matrix multiplication

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

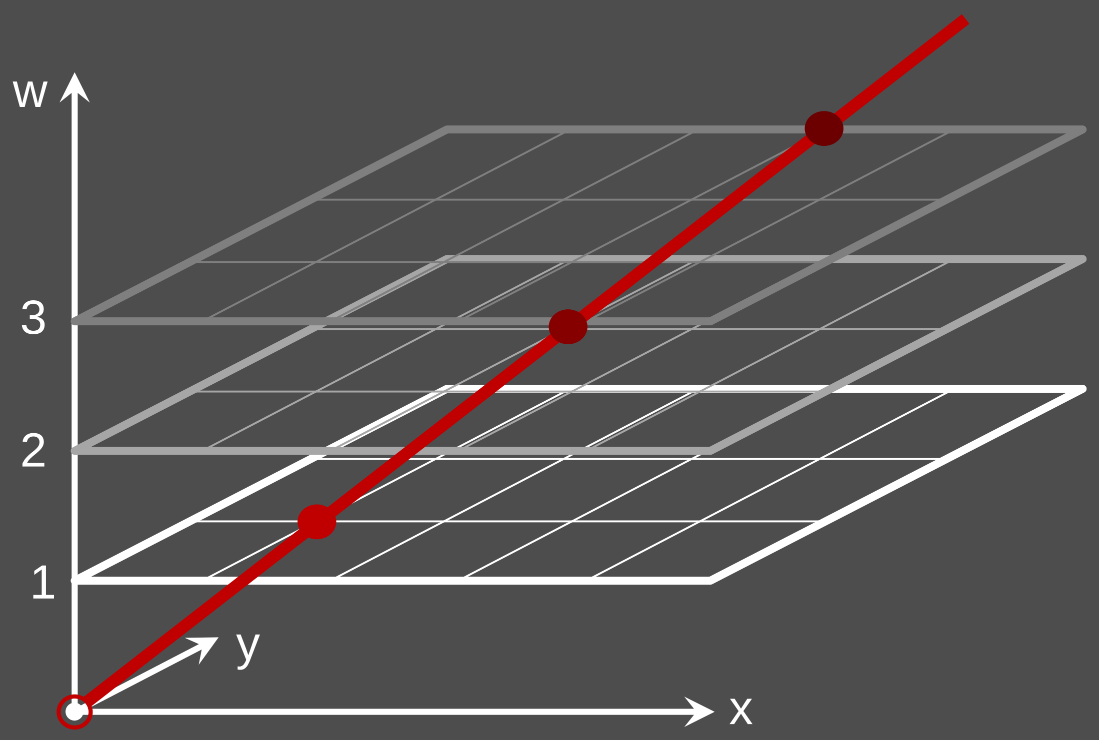
not possible in 2×2 matrix! ☹

Solution: Homogeneous Coordinates

- add an additional dimension to our vector space A^n: $n \rightarrow n+1$
- $(x, y)^T$ represented as $(wx, wy, w)^T$, $w \neq 0$
- normalized using $w = 1 \rightarrow (x, y, 1)^T$
- each point in A^n is equivalent to a line in homogeneous space A^{n+1} that originates in the origin o but without including o
- homogeneous coordinates are not to be confused with "regular" 3D coordinates!

Homogeneous Coordinates in 2D

- each point in A^n is equivalent to a line in homogeneous space A^{n+1} that originates in the origin o but without including o



Solution: Homogeneous Coordinates

- advantages of homogeneous coordinates
  - uniform treatment of transformations
  - all transformations can be represented
  - combined transformations as one matrix
- procedure: matrix-vector multiplication

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

- goal: derive transformation matrices

Translation in Homogeneous Coords

- general matrix multiplication format

$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$

- translation formula

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} dx \\ dy \end{pmatrix}$$

- translation as matrix multiplication

$$\begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Scaling in Homogeneous Coords

- scaling as matrix multiplication

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha & 0 \\ 0 & \beta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- scaling as homogeneous matrix multiplication

$$\begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Rotation in Homogeneous Coords

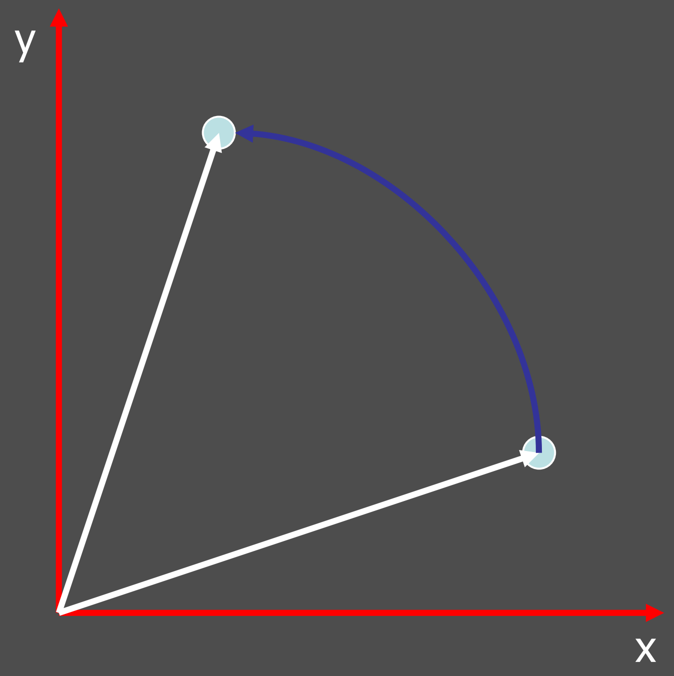
- rotation as matrix multiplication

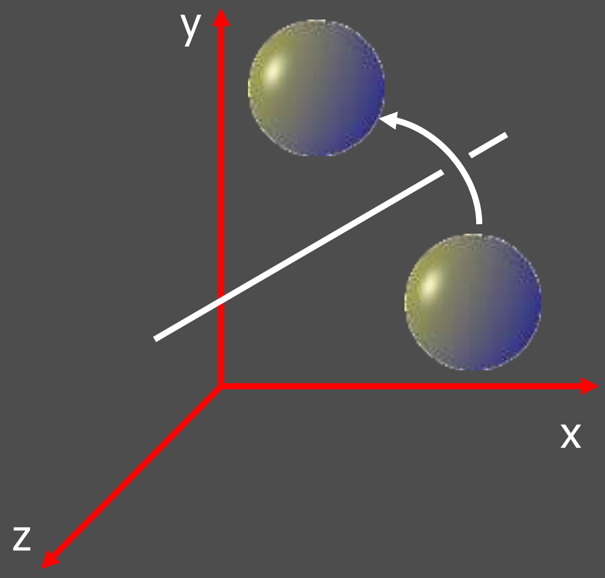
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- rotation as homogeneous matrix multiplication

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Transformations in 3D Space





Geometric Transformations in 3D

- same approach as in 2D
- also use homogeneous coordinates (for the same reasons)
- vectors/points from 3D to 4D

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- transformation matrices are now 4×4 (instead of 3×3)

Transformation Matrices in 3D

- translation
  - translation vector $(dx, dy, dz)^T$
- scaling
  - for uniform scaling $s_x = s_y = s_z$
  - otherwise individual factors may differ
  - mirroring using factors of -1 and 1 depending on the mirror plane

$$T = \begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformation Matrices in 3D

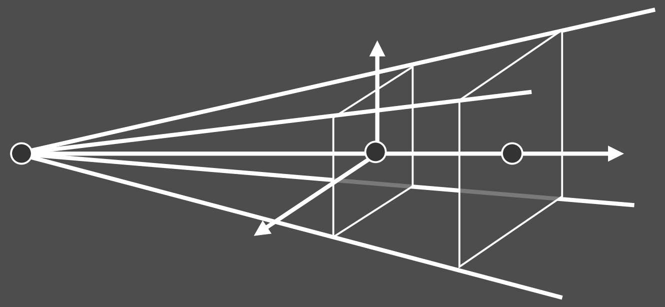
- rotation
  - rotation around 3 axes possible now
  - each has individual rotation matrix
  - rotation around positive angles in right-handed coordinate system
  - rotation axis stays unit vector in matrix

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

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

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

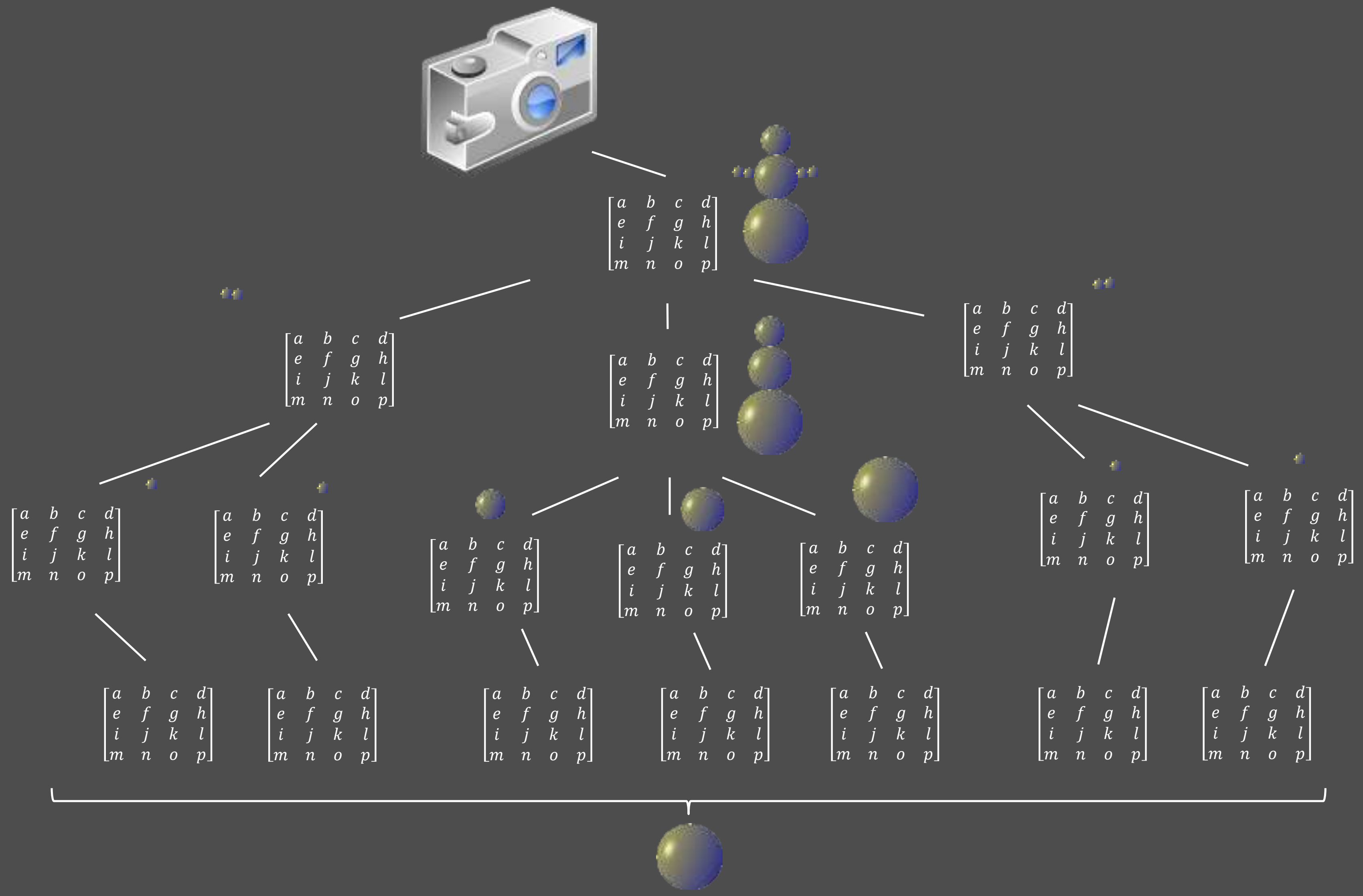
Viewing and Projections



Model-View Transformation

- model-view transformation steps:
  1. translate object origin to camera location
  2. rotate to align coordinate axes
  3. possibly also scaling
- this process is used in OpenGL: no explicit world coordinates!
- object & camera locations and orientations may be specified in a world coordinate system (e.g., in modeling systems)
- allows object hierarchies with animation
- separate from projection (next)

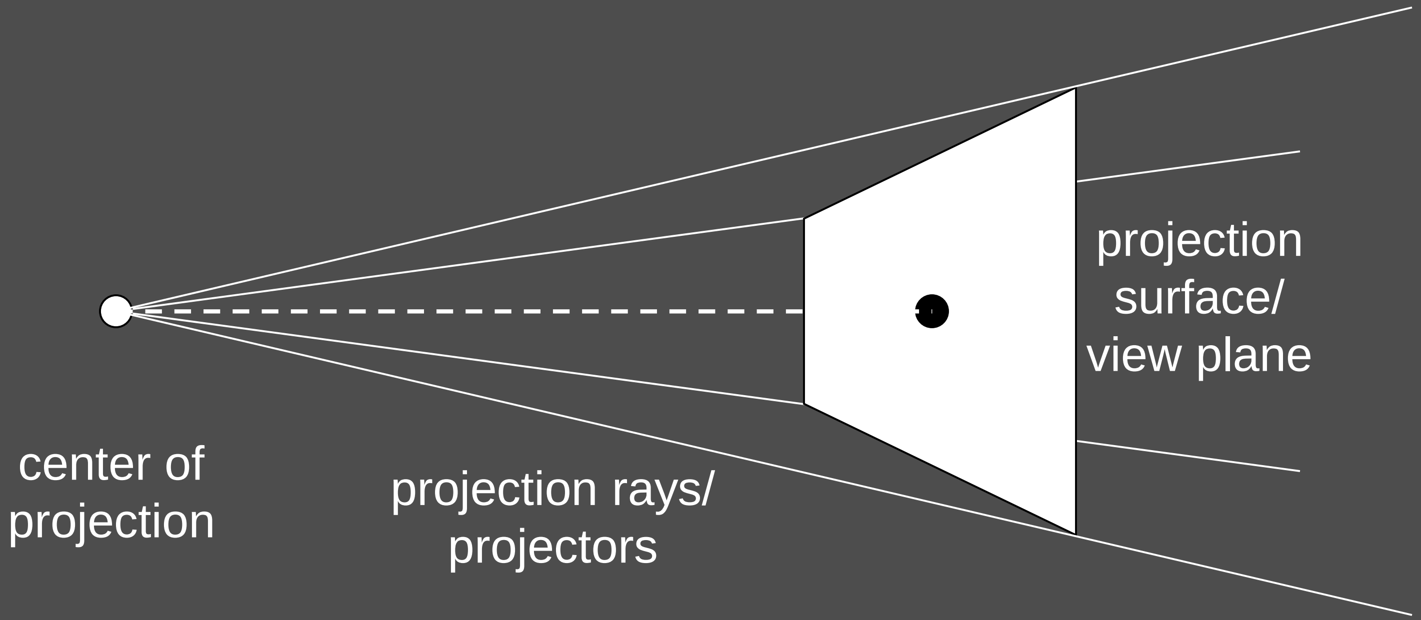
Model-View Transformation



Projections

planar projection:

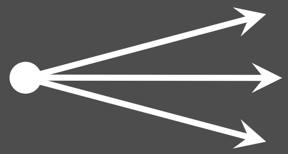
- projection rays are straight lines
- projection surface/view plane is planar
- projections of straight lines are also straight



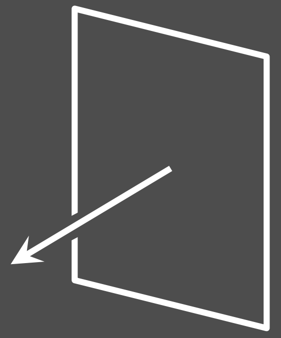
Introduction – Terms

- parallel projection: characterized by **direction of projection** (dop)

- perspective projection: **center of projection** (cop)

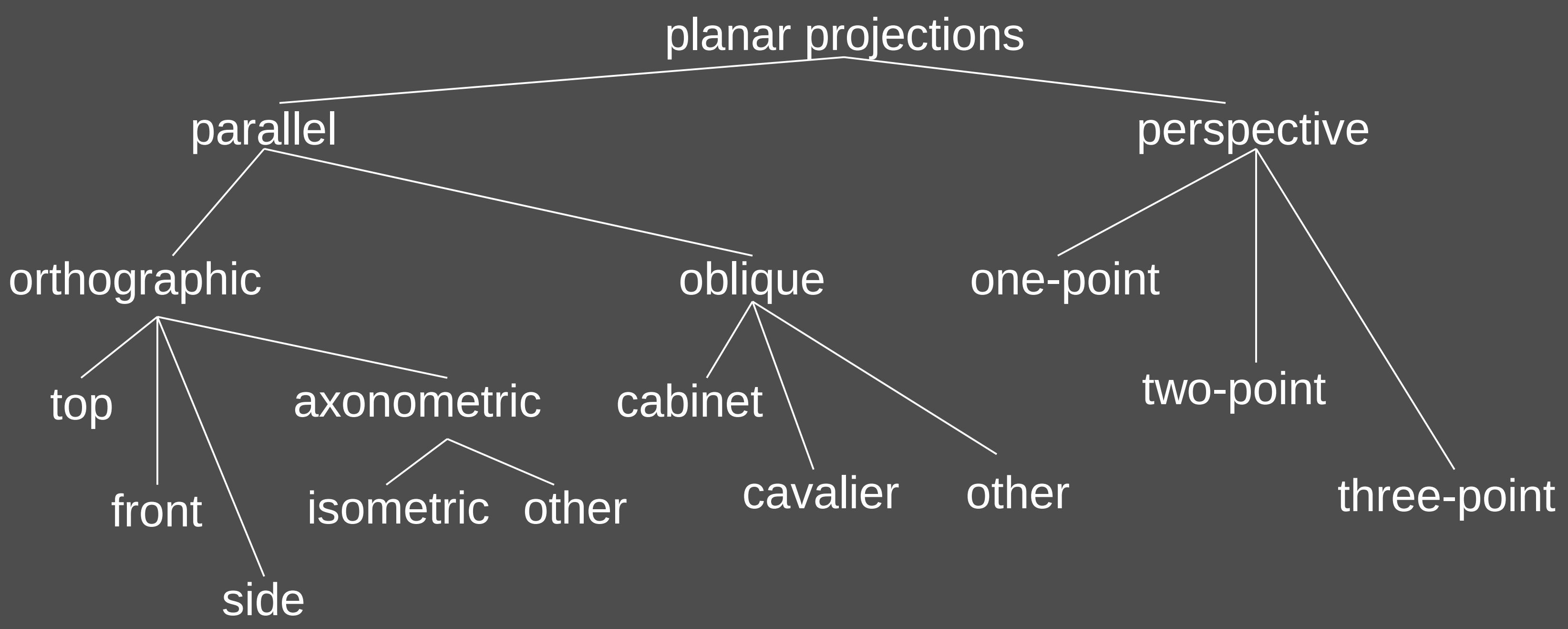


- projection on **view plane**
- vector perpendicular to view plane: **view plane normal** (vpn)



- rays characterizing projection: **projectors** (parallel or diverging from cop)

Classification of Planar Projections



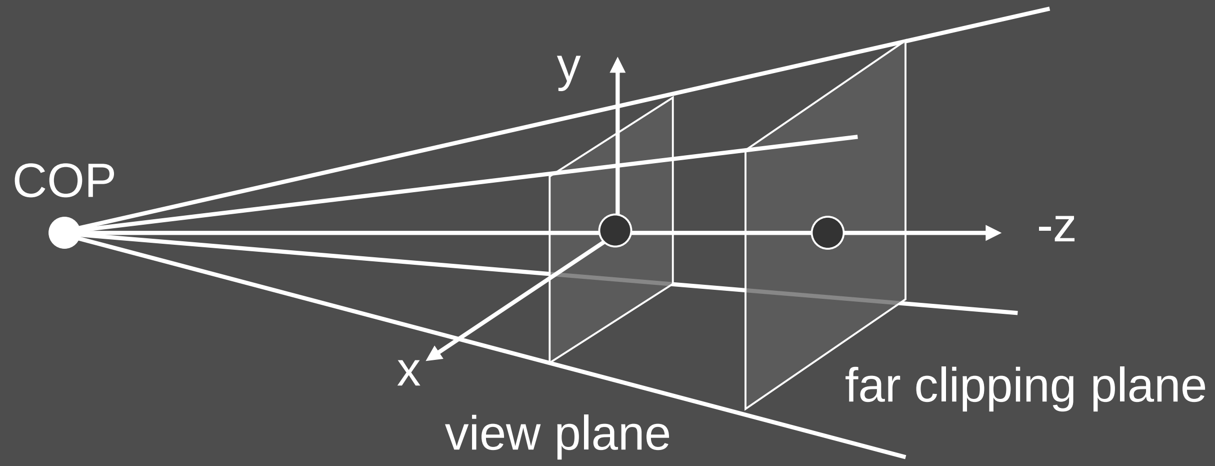
- parallel projections:
  all projectors parallel to each other
- perspective projections:
  projectors diverge from cop

Perspective Projections: Camera Model

- inspired by real cameras
- parameters:
  - position, orientation
  - aperture
  - shutter time
  - focal length, type of lens (zoom vs. wide)
  - depth of field
  - size of resulting image
  - aspect ratio (4:3, 16:9, 1.85:1, 2.35:1, 2.39:1)
  - resolution (digital cameras)

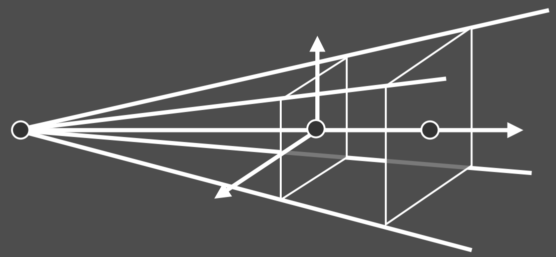
Perspective Projections: Camera Model

- simplified model in CG
  - position: point in 3D (= cop)
  - view direction: vector in 3D (vpn)
  - image specification: viewport
  - clipping planes for cutting off near and far objects (near and far clipping plane)

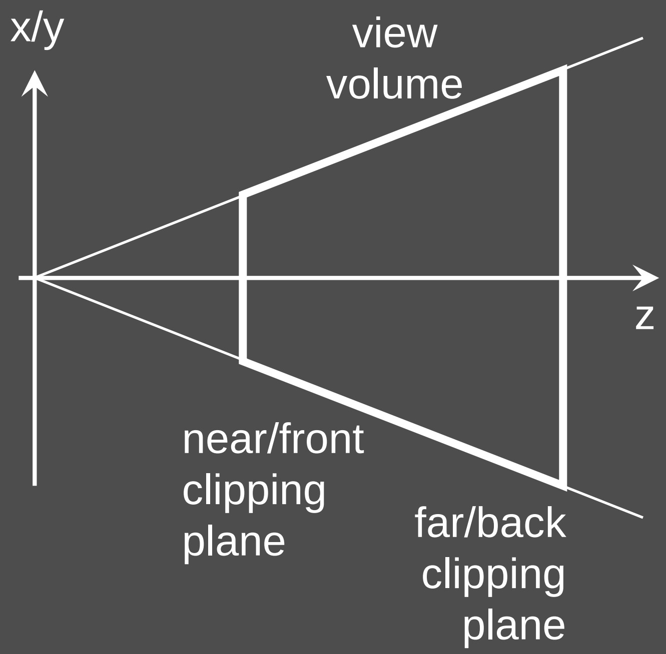


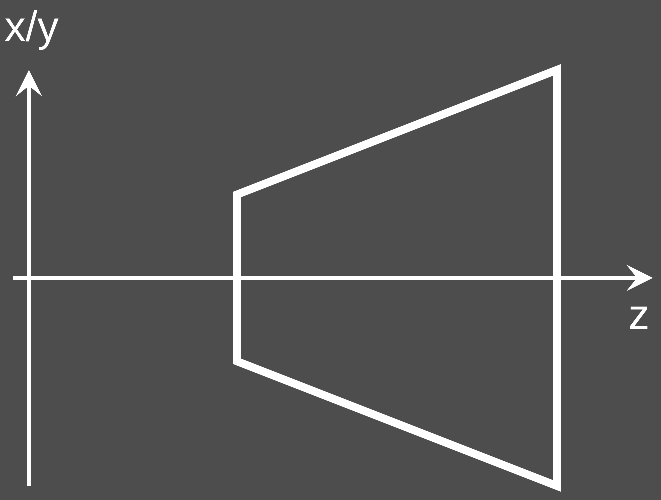
Perspective Projections: Camera Model

- differences between real and CG camera?
  - position of view plane w.r.t. COP
  - orientation of the image
  - type of camera (pinhole vs. lens)
    - type of "refraction"
    - lens effects (lens flare)
    - depth of field: none vs. existing
  - types of possible projections
  - picture taking times: 0 vs. >0
  - existence of far clipping plane
  - shape of view volume

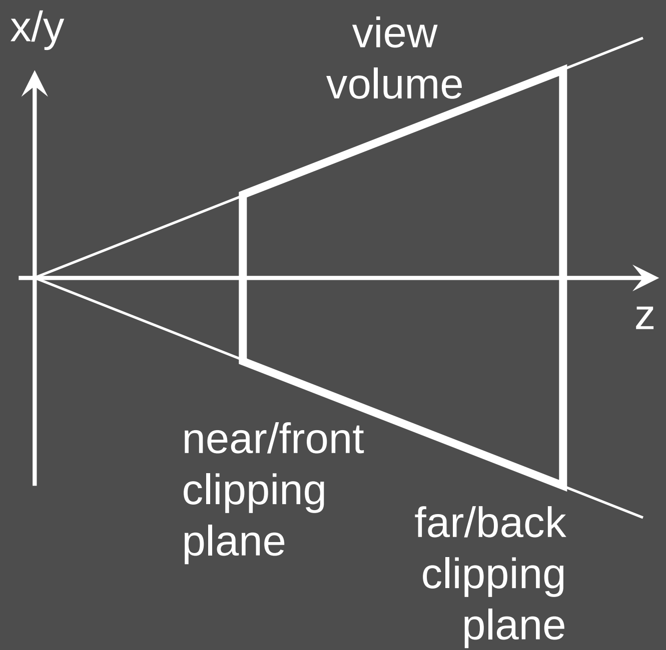


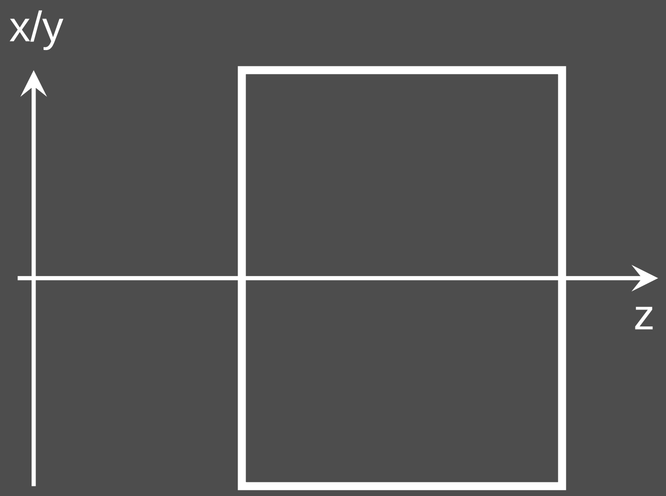
Projections: Canonical View Volumes



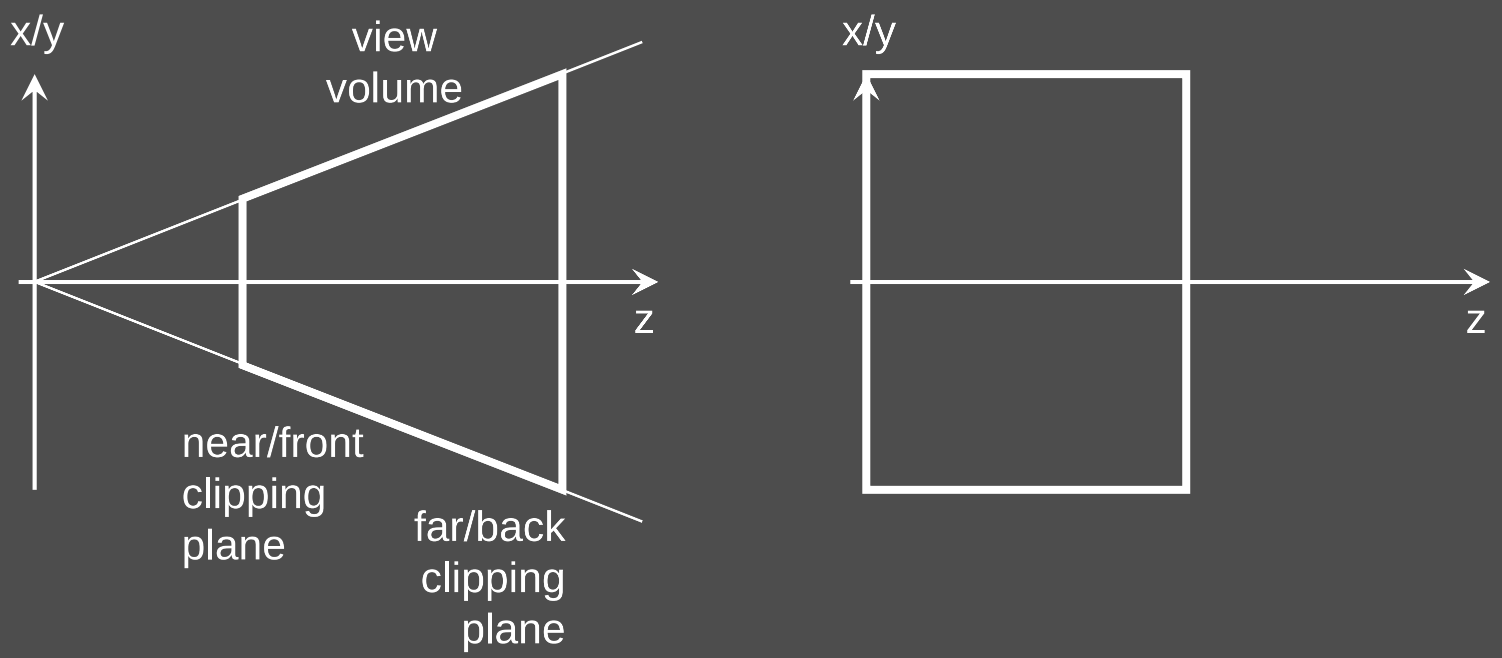


Projections: Canonical View Volumes

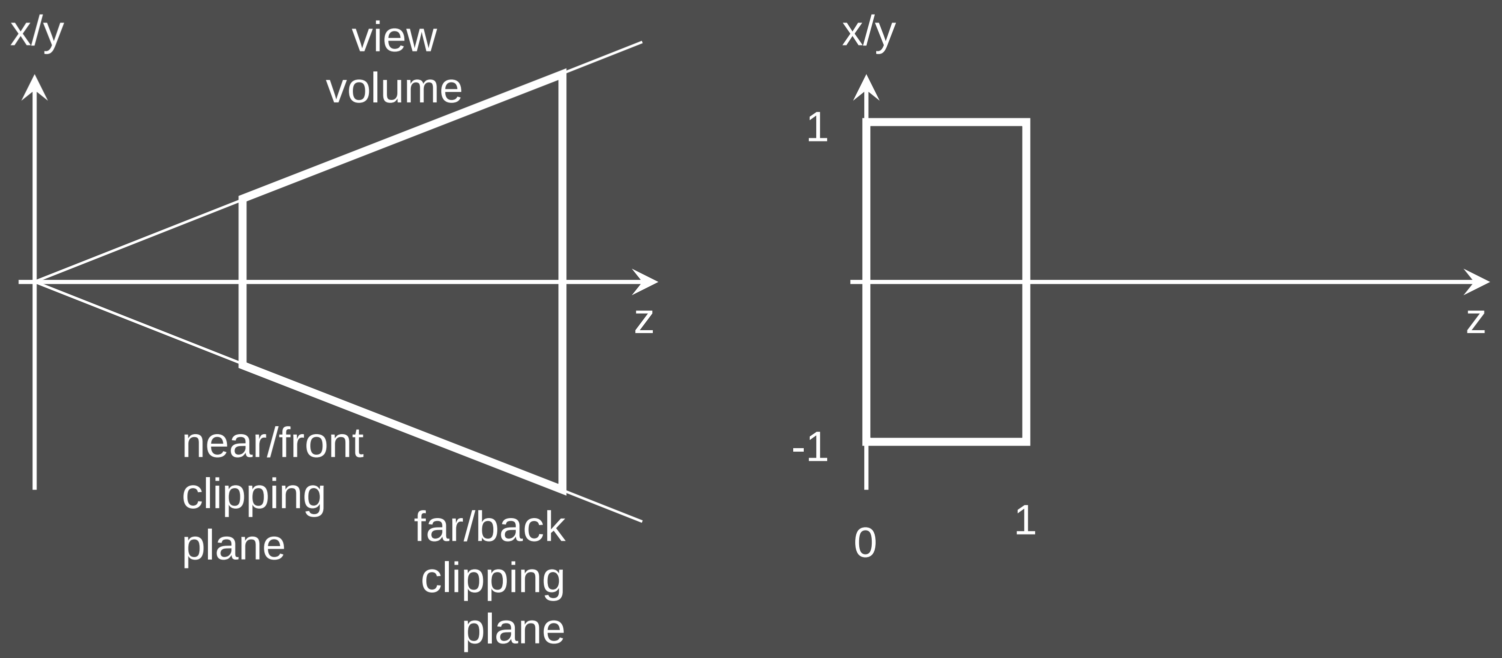




Projections: Canonical View Volumes

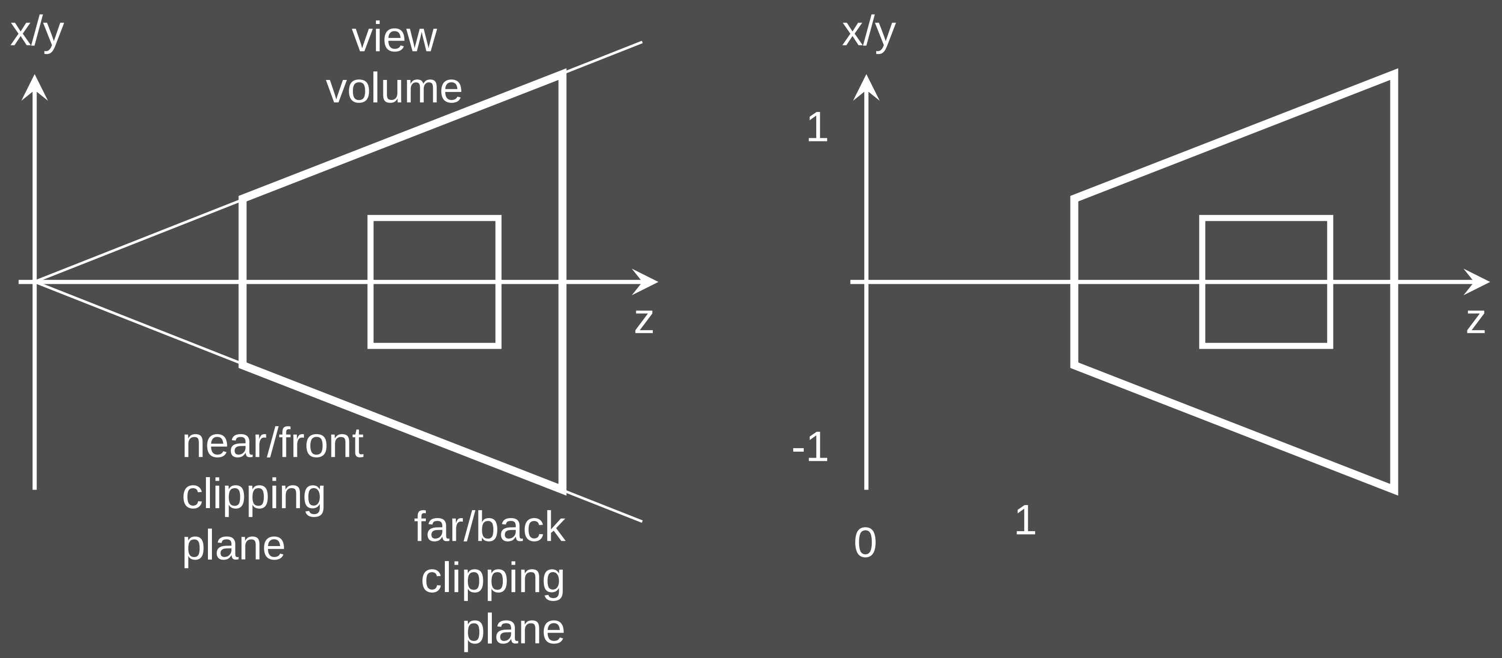


Projections: Canonical View Volumes



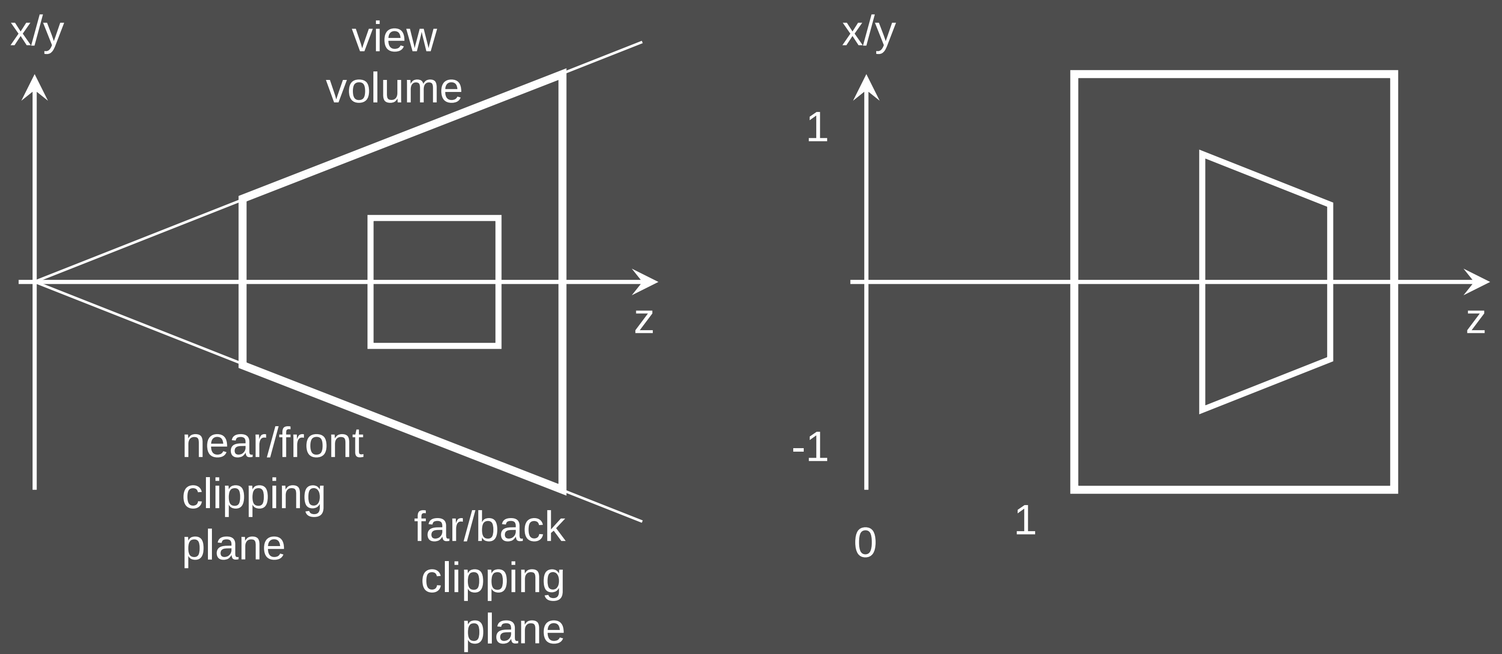
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

Projections: Canonical View Volumes



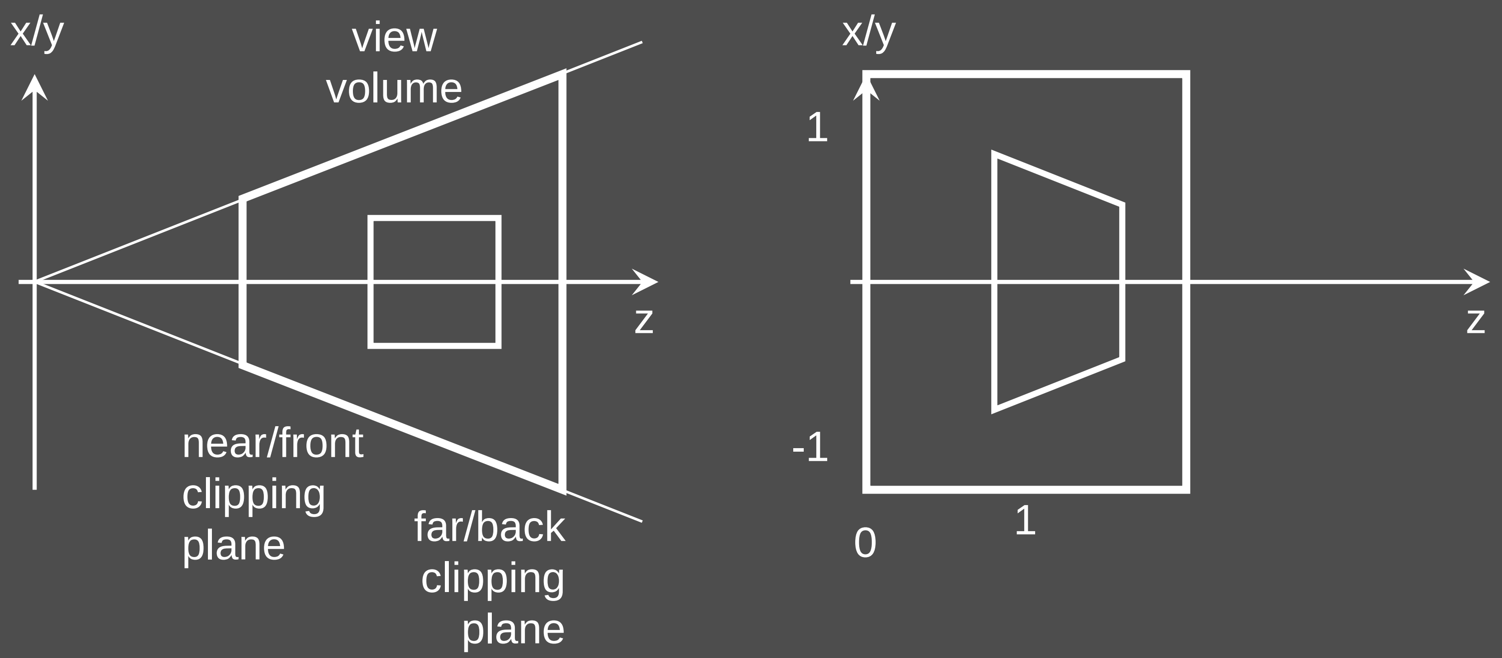
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

Projections: Canonical View Volumes



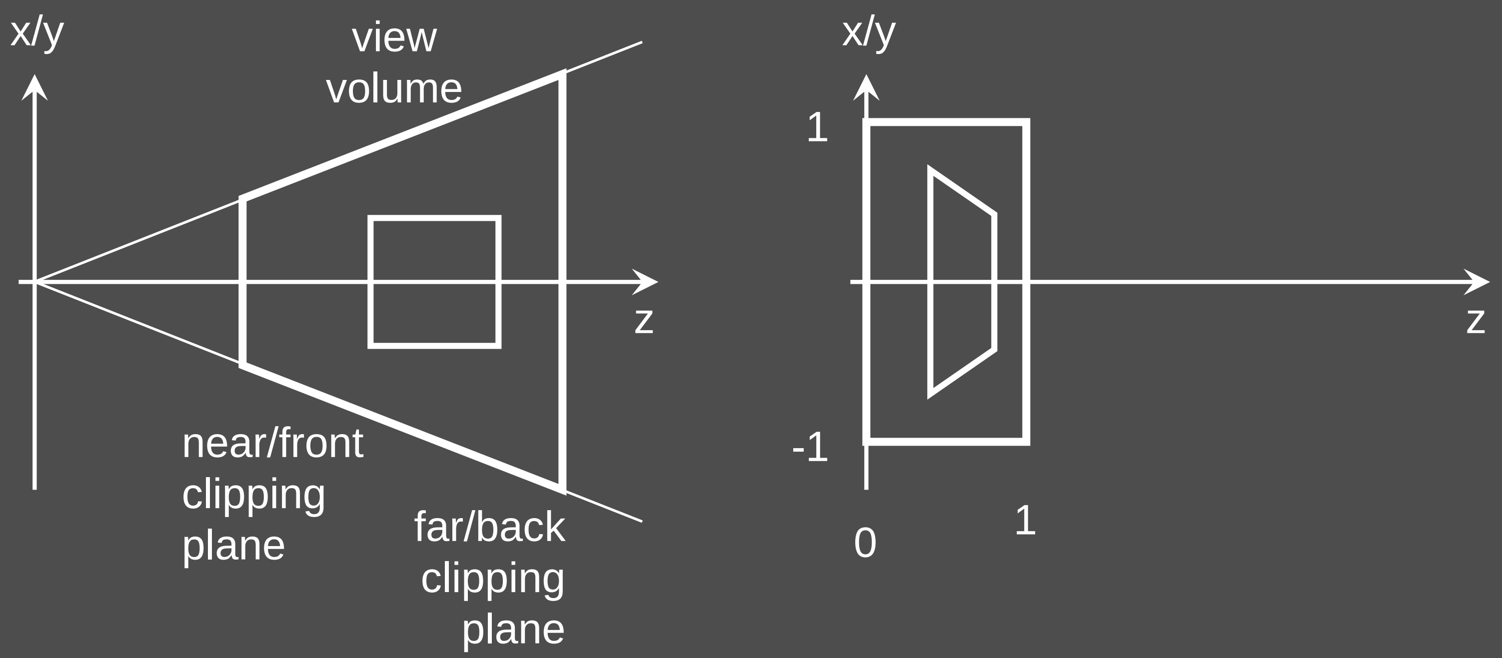
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

Projections: Canonical View Volumes



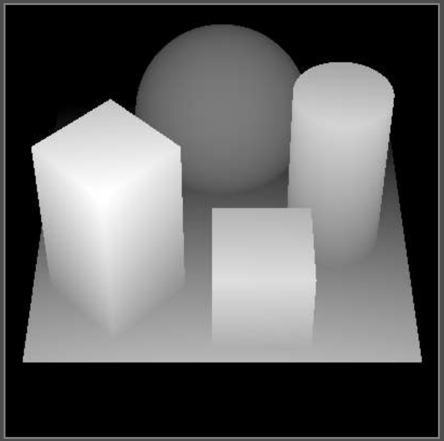
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

Projections: Canonical View Volumes



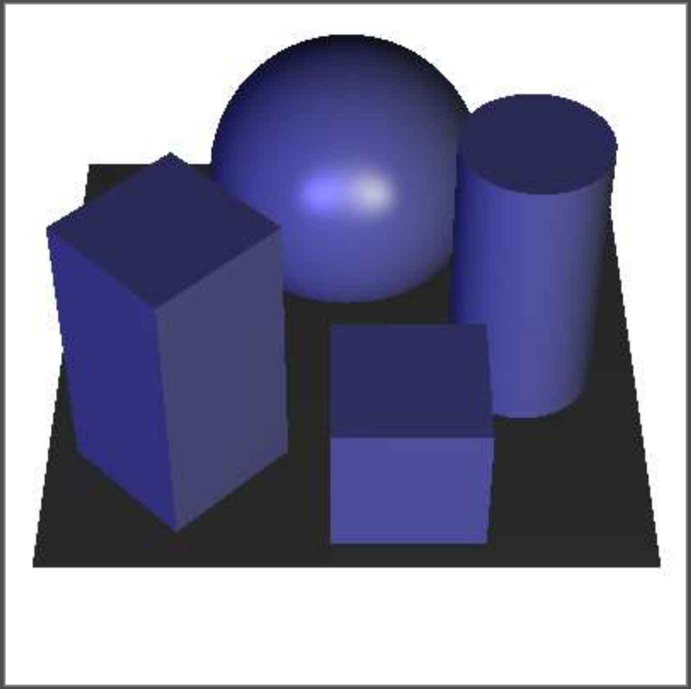
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

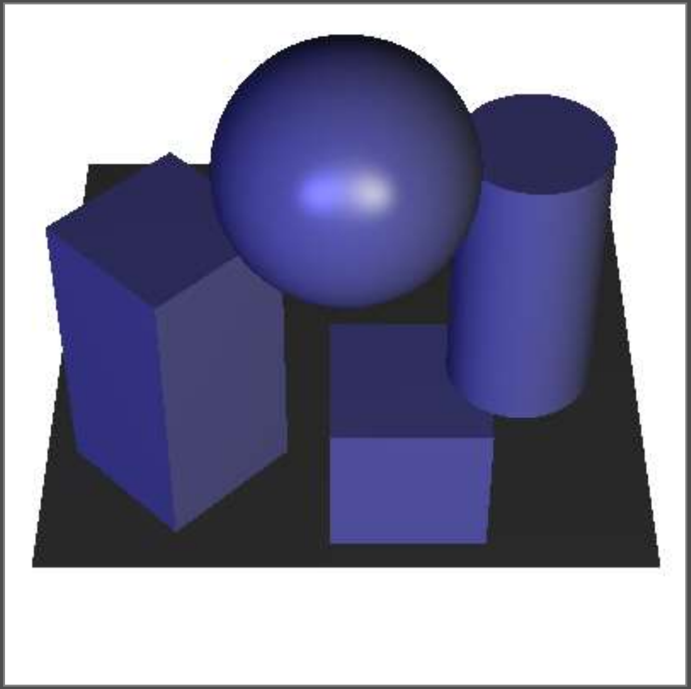
Hidden Surface Removal



Hidden Surface Removal: Motivation

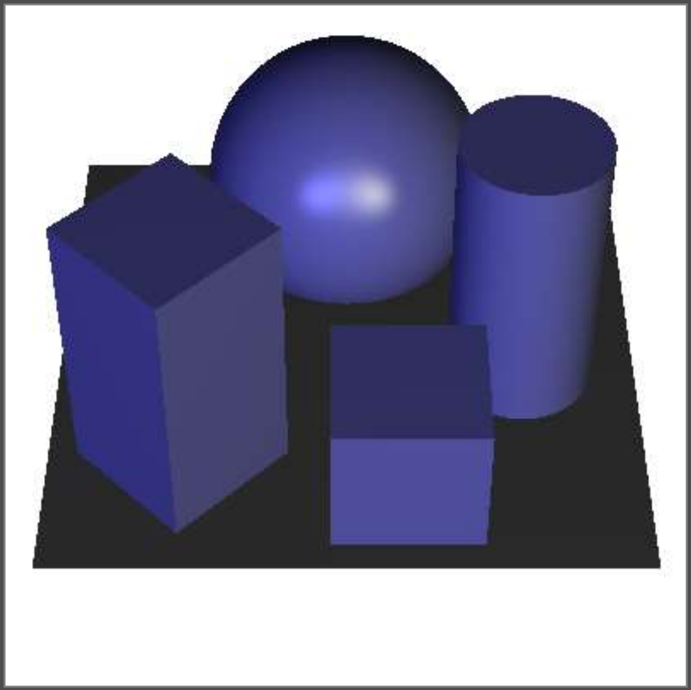
- goals:
  - model parts independently processed
  - at the same time: show front parts only
  - avoid unnecessary processing

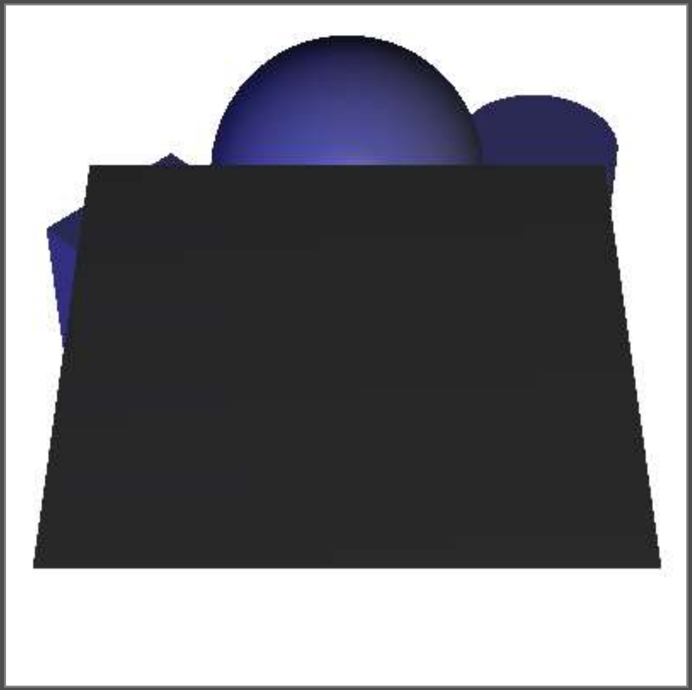




Hidden Surface Removal: Motivation

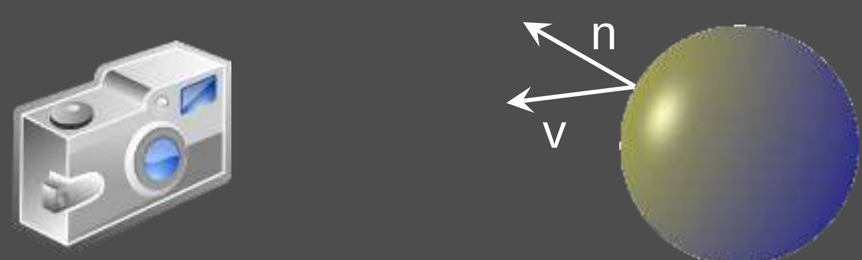
- goals:
  - model parts independently processed
  - at the same time: show front parts only
  - avoid unnecessary processing





Back Face Culling

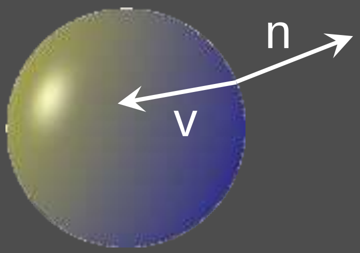
- back faces (usually) not visible
- remove these: reduction of computation
- removal early in the pipeline
- reduction of polygon count by approx. ½ of the total polygon number
- computation: compare dot product of surface normal with view direction with 0



$$n \cdot v > 0$$

Back Face Culling

- back faces (usually) not visible
- remove these: reduction of computation
- removal early in the pipeline
- reduction of polygon count by approx. ½ of the total polygon number
- computation: compare dot product of surface normal with view direction with 0



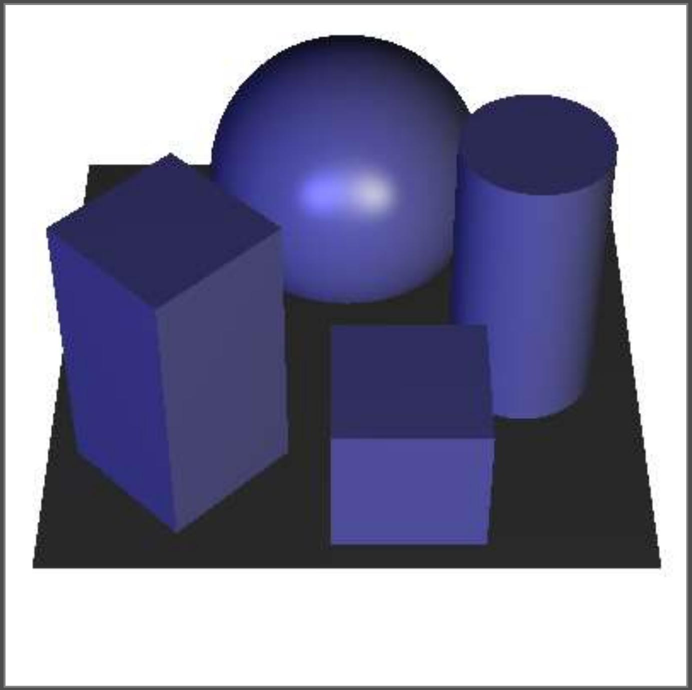
$$n \cdot v < 0$$

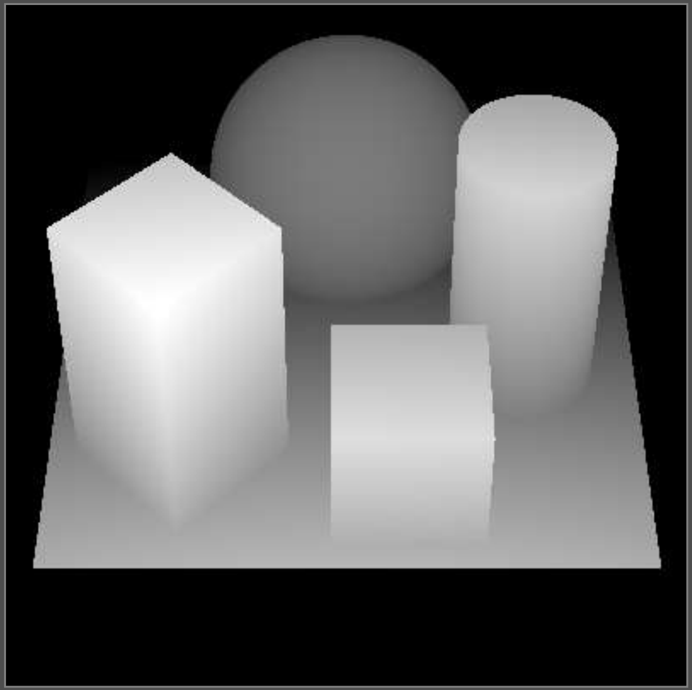
z-Buffering

- idea:
  - rendering from back to front (Painter's algorithm)
  - avoid need to sort & problems with cyclic triangles
- realization by
  - trading speed for memory usage
    (memory is cheap [now anyway], time is not)
  - trading speed for accuracy
    (only compute what we really need/want)

z-Buffering

- introduce new pixel buffer: *z*-buffer (in addition to the frame buffer for image)
- *z*-buffer stores *z*-values of pixels





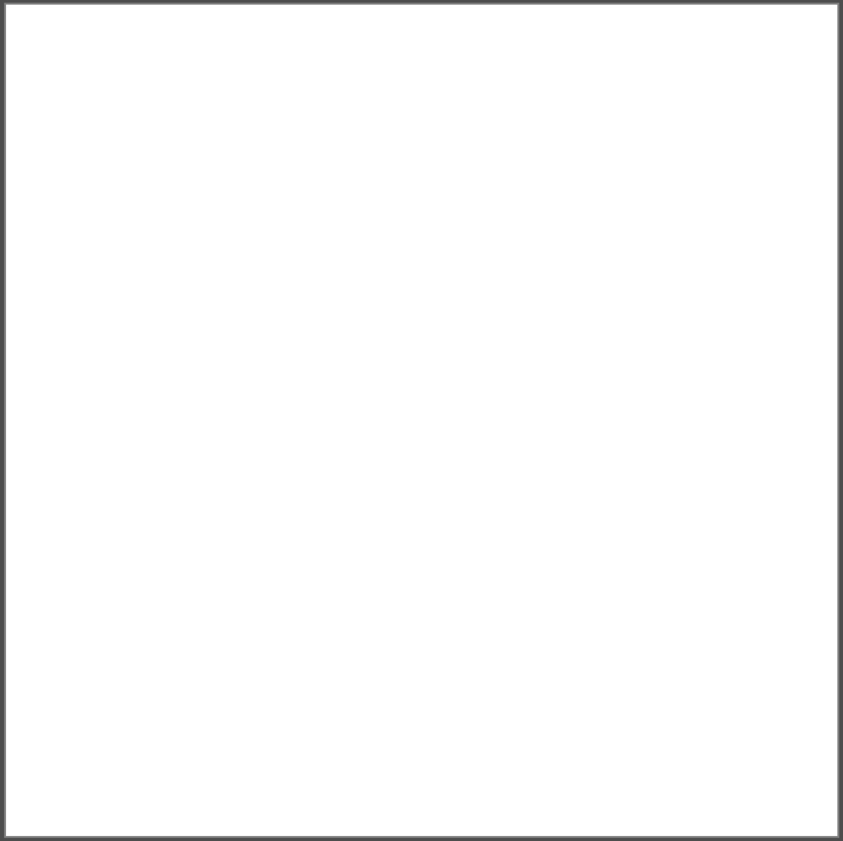
z-Buffering

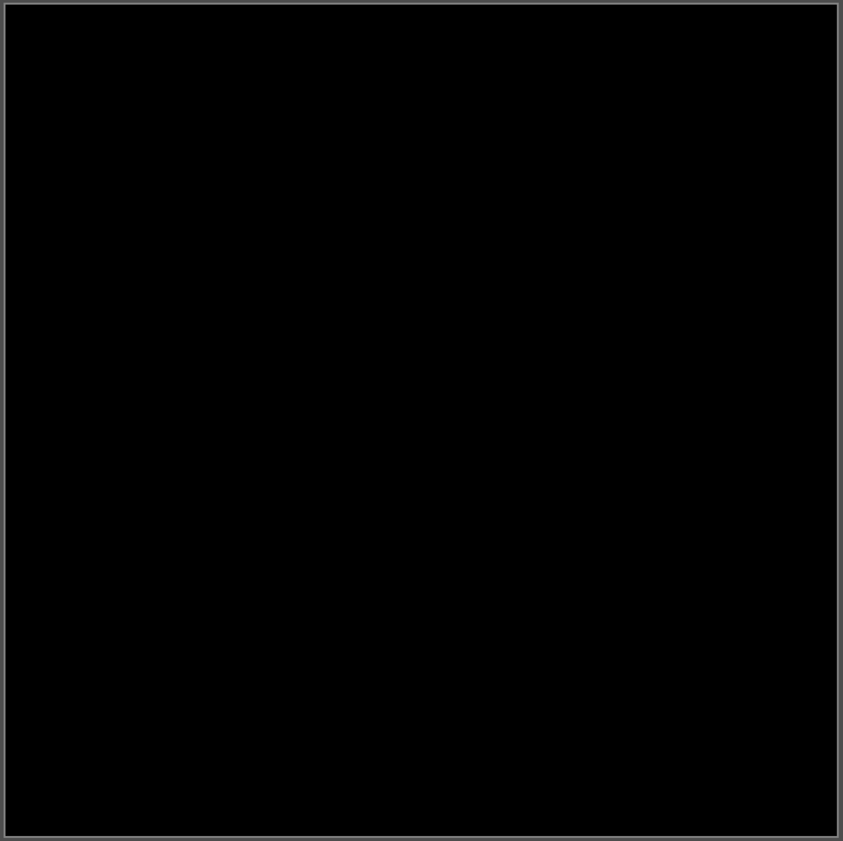
- treat each primitive (triangle) individually:
  scene → objects → triangles → pixels
- use *z*-buffer data to determine if a new triangle is (partially) hidden or not
- at each time, the part of scene that has been processed thus far is correctly displayed

z-Buffering: Algorithm

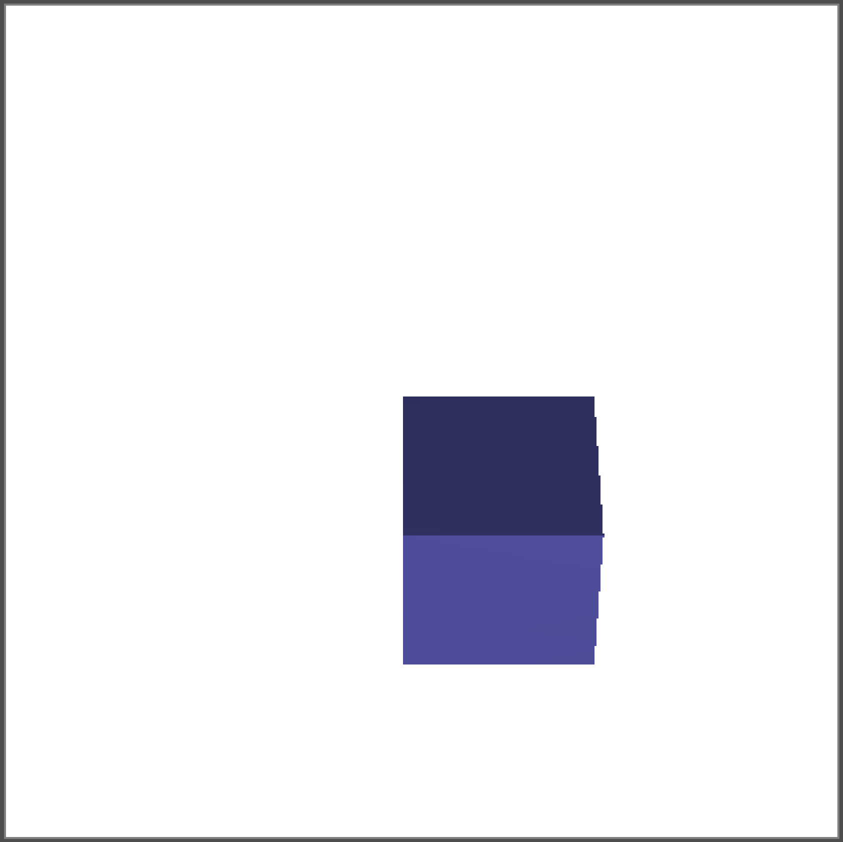
- initialize *z*-buffer and frame buffer
- for each triangle
  - project all vertices of the triangle
  - interpolate *z*-values for each pixel (scan line)
  - before shading a pixel, test
    if its *z*-value is closer to camera (i.e. higher)
    than the current *z*-buffer value
  - if so: update *z*-buffer value and shade pixel
  - otherwise: discard pixel and continue
- after scene processing *z*-buffer contains depth map of scene

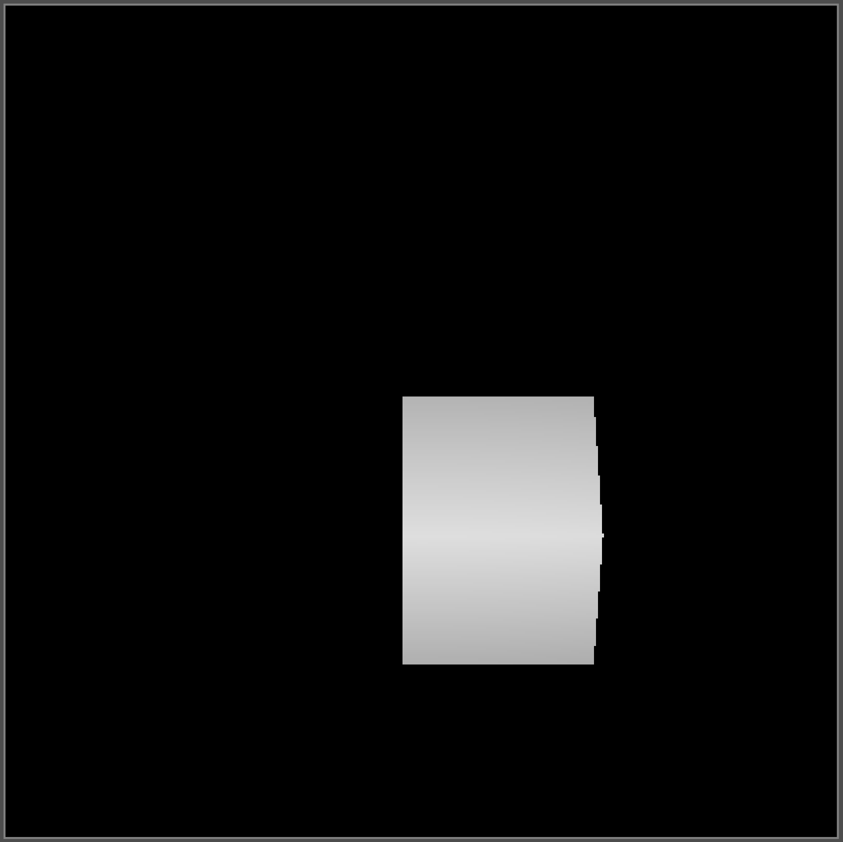
z-Buffering: Example (by object)



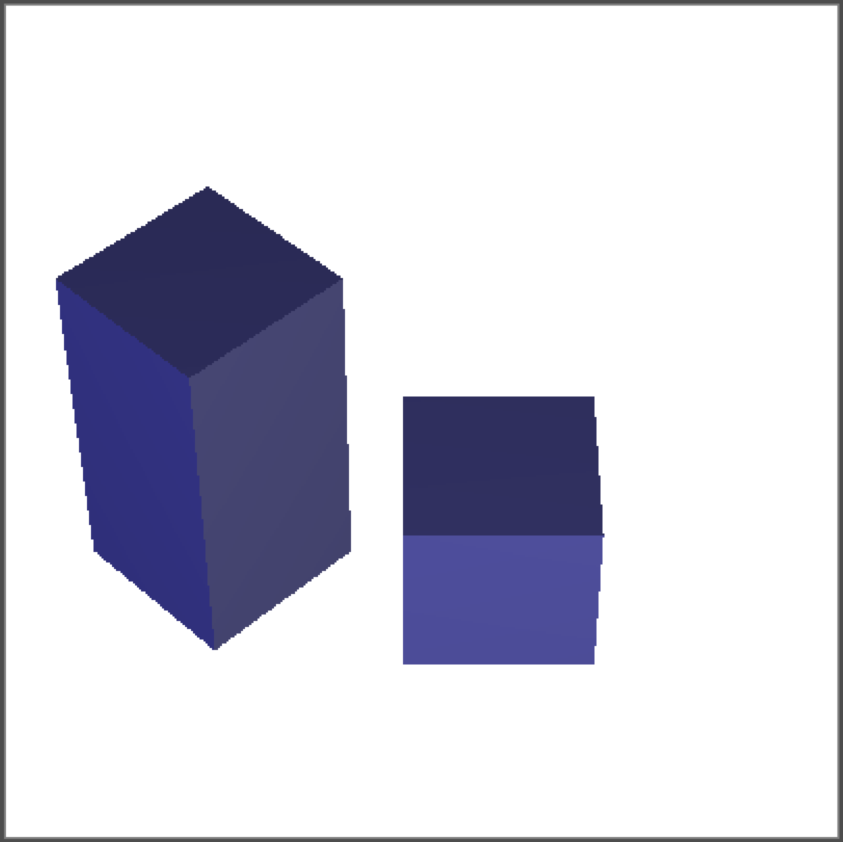


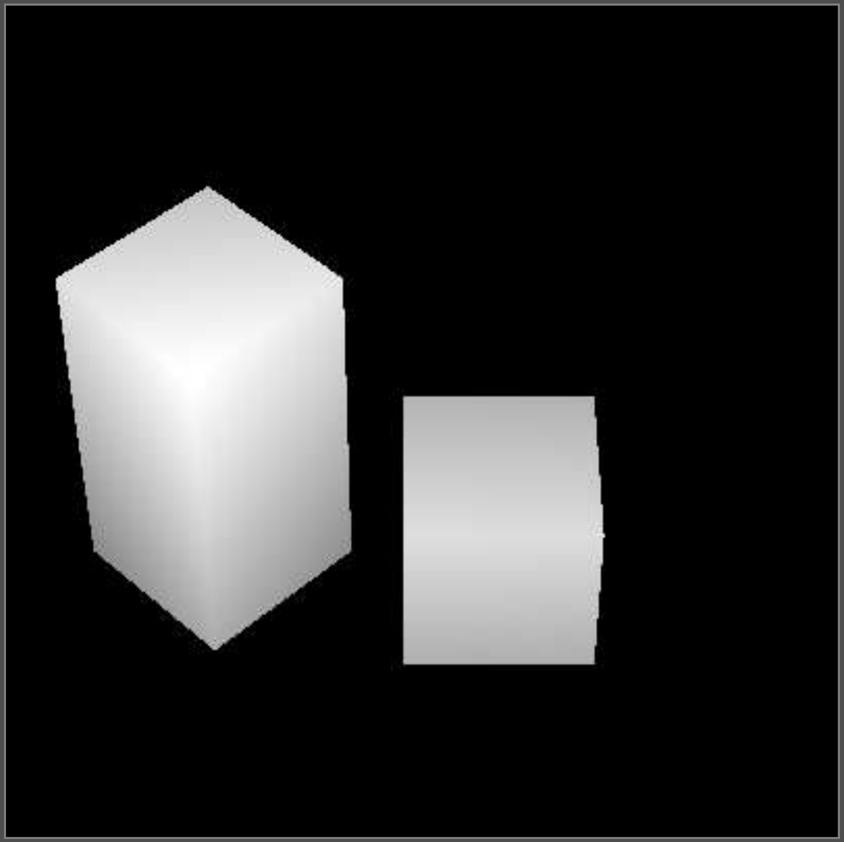
z-Buffering: Example (by object)



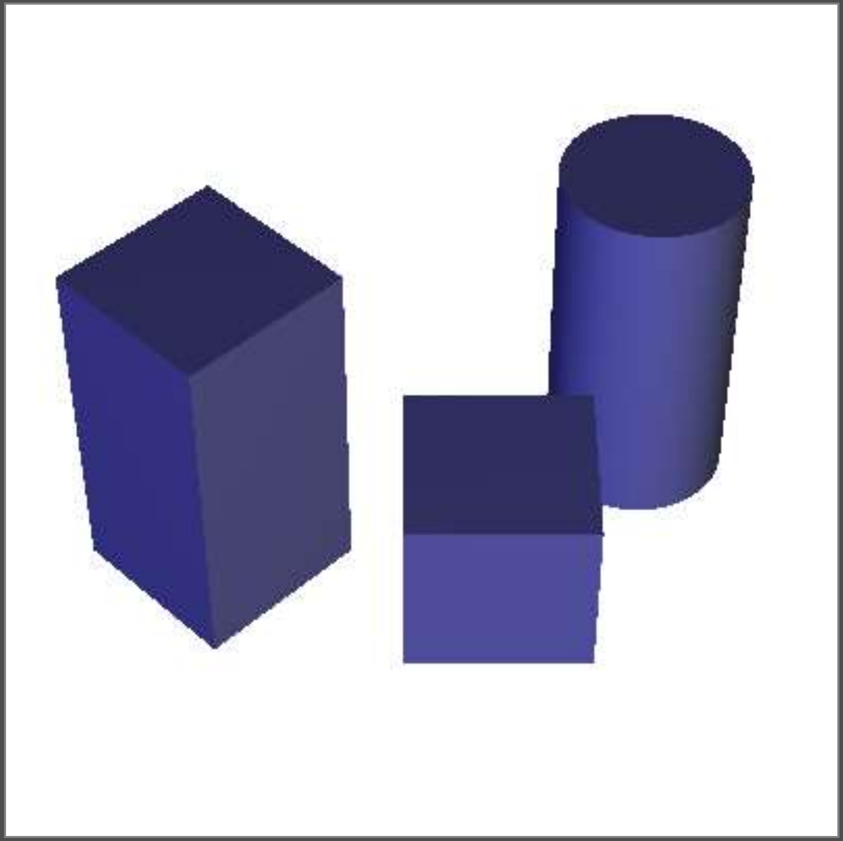


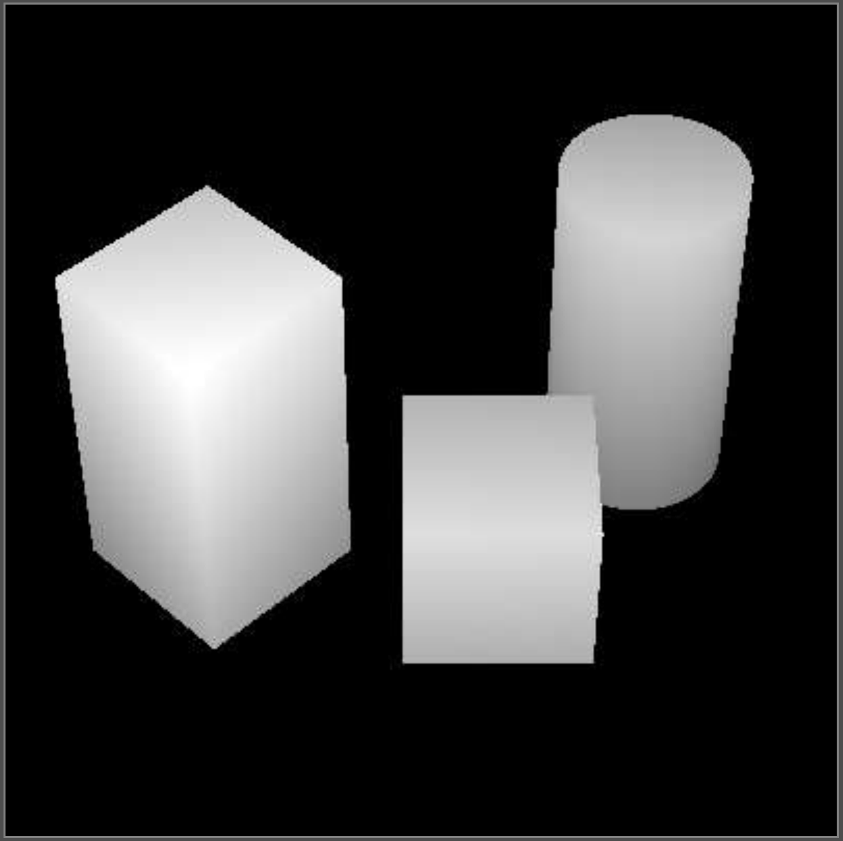
z-Buffering: Example (by object)



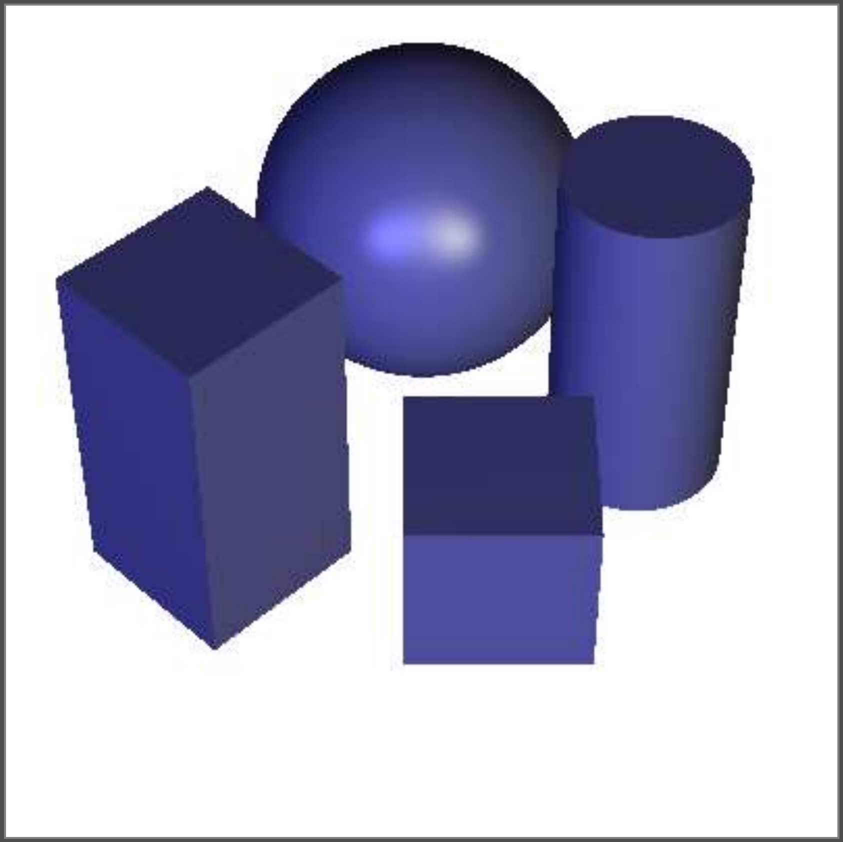


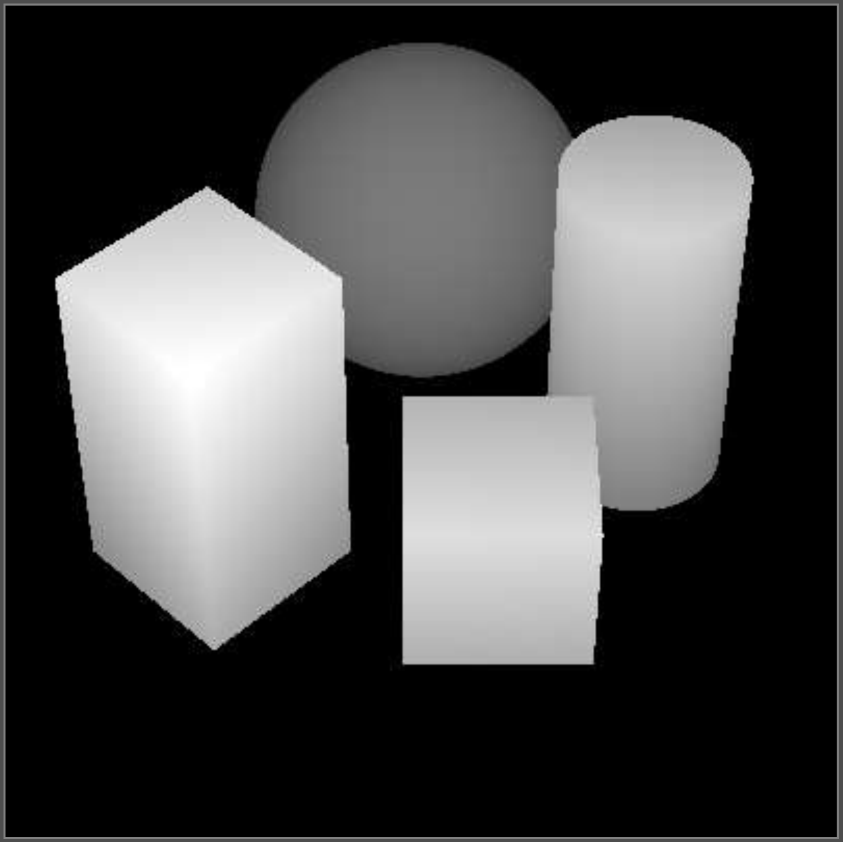
z-Buffering: Example (by object)



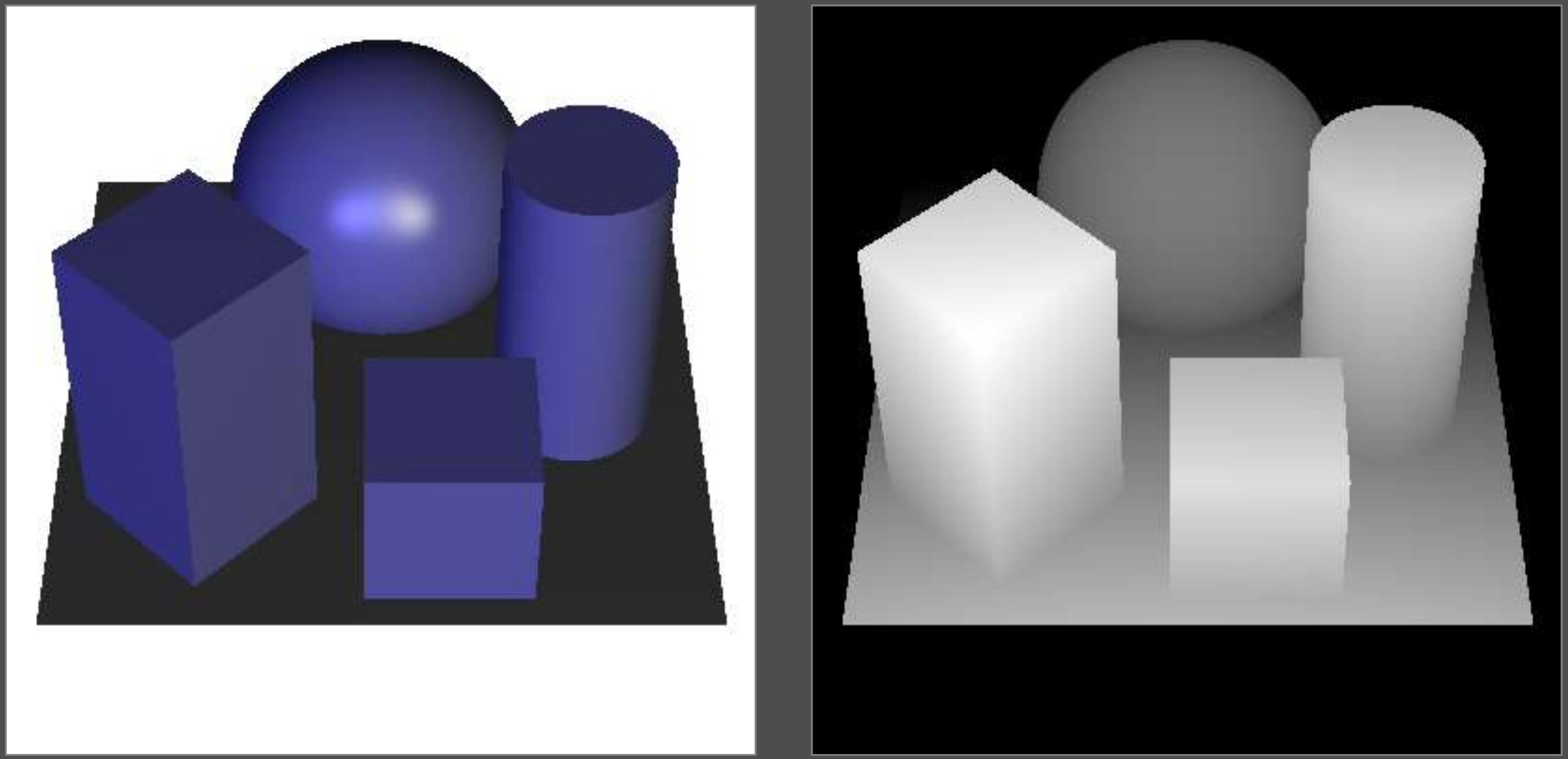


z-Buffering: Example (by object)





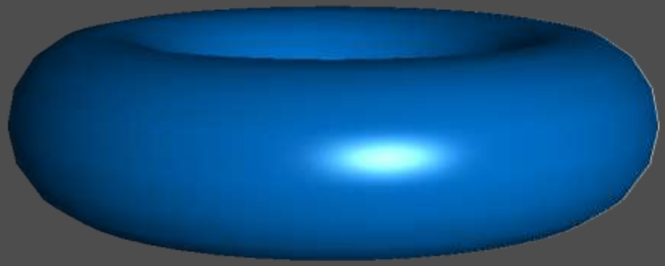
z-Buffering: Example (by object)



z-Buffering

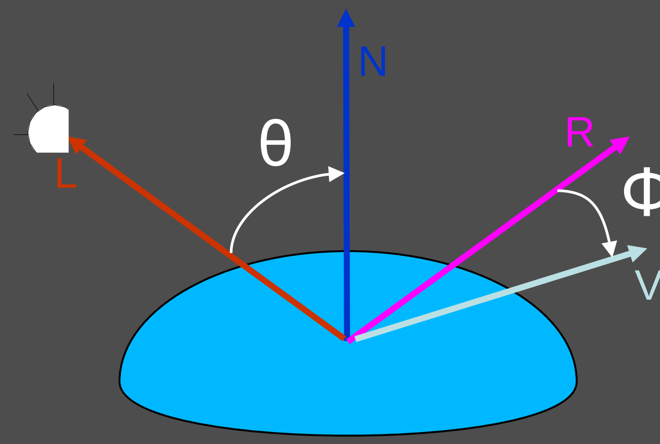
- advantages
  - can be implemented in hardware
  - can process infinitely many primitives
  - does not need sorting of primitives
    (only need to know distance to camera)
  - can handle cyclic and penetrating triangles
- disadvantages
  - needs memory to keep all *z*-values
    (image size @ 8bit or 16bit)
  - cannot handle transparency properly

Illumination and Shading



Light Interaction with Surfaces

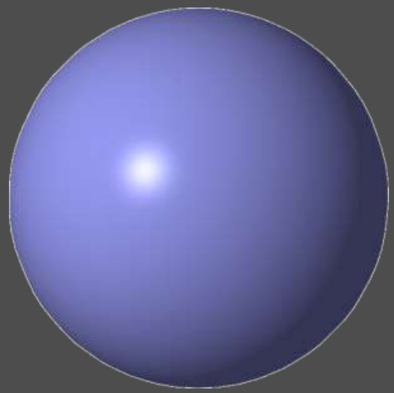
- angle θ between L & N determines diffuse reflection
- reflection angle equals θ
- angle Φ between R & V determines perceived brightness
- maximal reflection if R = V (Φ = 0)

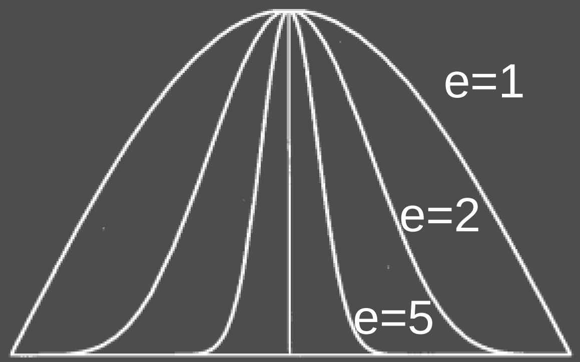


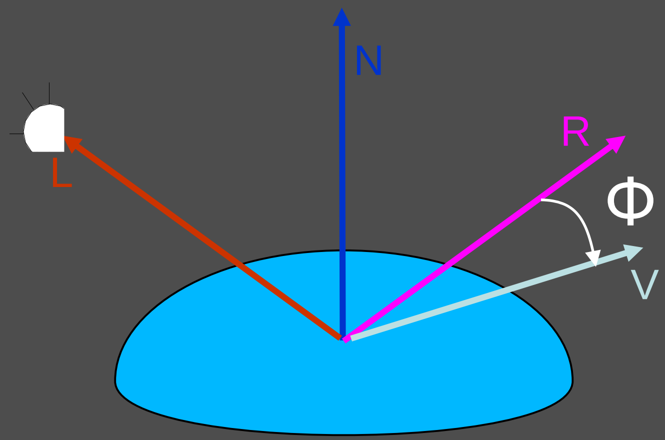
L – vector to light source
N – surface normal vector
R – reflected light ray
V – vector to viewer/observer

Light Interaction with Surfaces

- **directed reflection**: reflection only for small Φ: smooth surfaces
- light attenuation depending on angle Φ: shininess
- modeled using cosine function and exponent: $I \sim \cos(\Phi)^e$ (e.g., metals: $e \approx 100$)
- physical reality: anisotropic materials

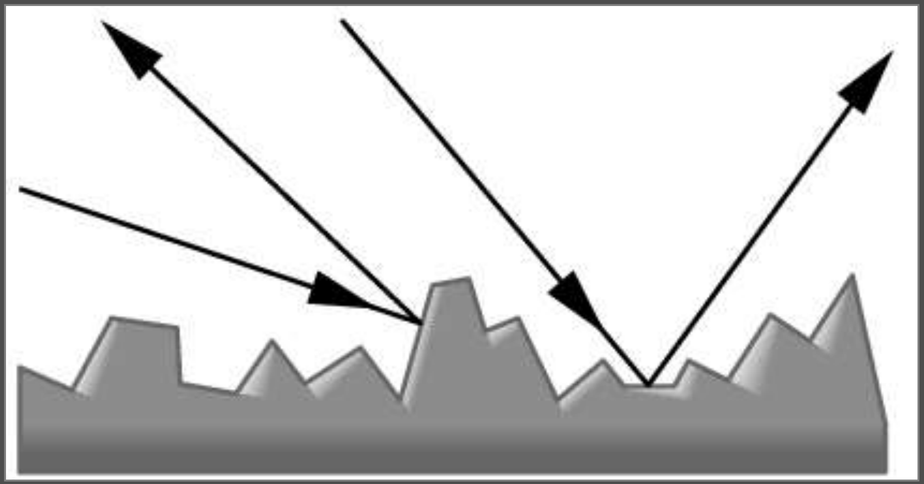






Light Interaction with Surfaces

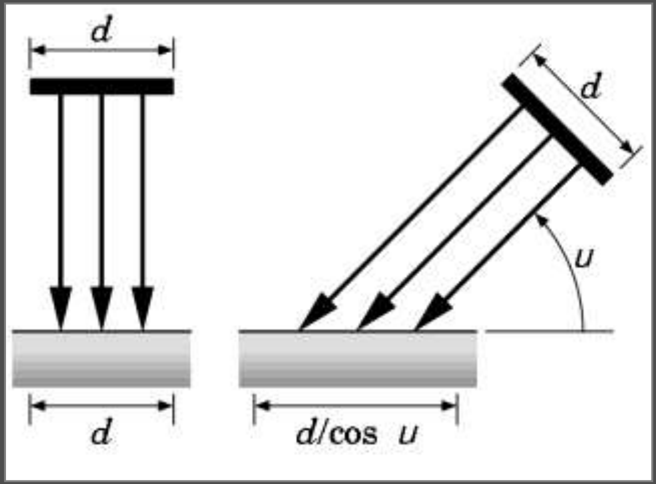
- **diffuse reflection**: equal reflection in all directions on rough surfaces, depends only on θ and not observer – examples?
- due to light scattering on rough surfaces on randomly oriented microscopic facets



Angel (2000)

Light Interaction with Surfaces

- diffuse reflection modeled according to Lambert's law by cosine function:
  $I \sim \cos\theta = L \bullet N$ for normalized L, N

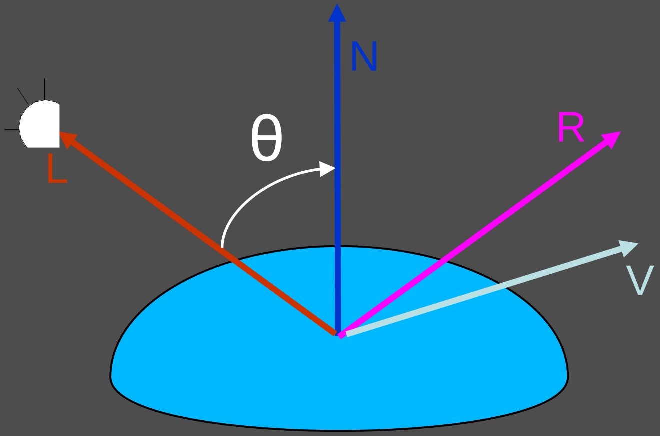


Angel (2000)

because light distributes over larger area

$\theta = 0^o \rightarrow$ max. intensity

$\theta = 90^o \rightarrow 0$ intensity

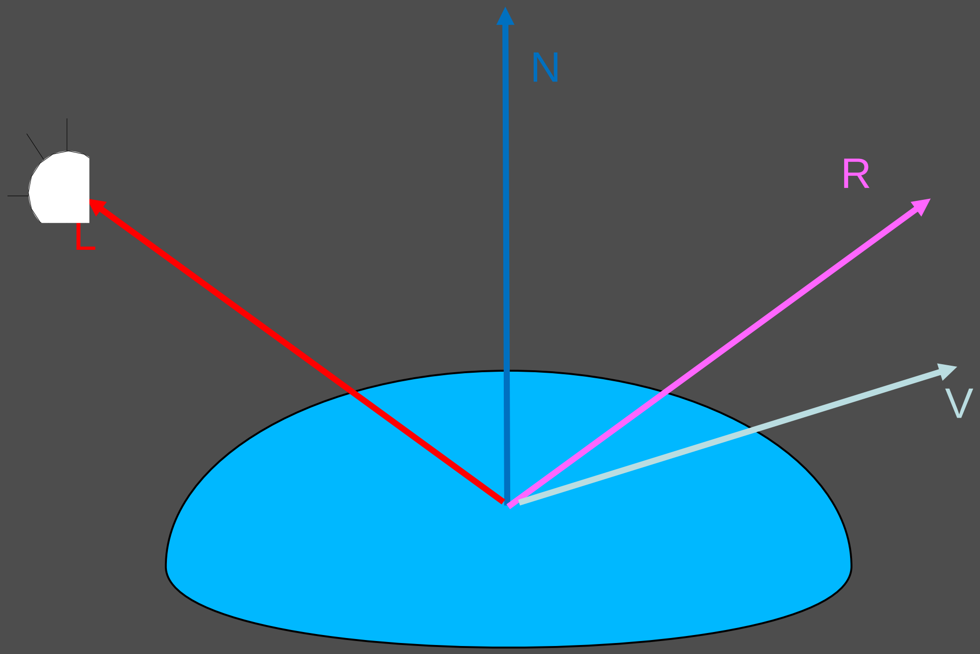


Phong Illumination Model (1973)

- most common CG model for illumination (by Bùi Tường Phong): $I_{Phong} = I_a + I_d + I_s$
- ambient light: base illumination of scene
  - simulates light scattering on objects
  - necessary because repeated diffuse reflection is not considered in local illumination model
  - depends on color of all objects in scene
  - should always be kept very small
- diffuse light: light from diffuse reflection
- specular light: light from directed reflection

Phong Illumination Model

$$I_{Phong} = I\,k_a + \frac{1}{a + bd + cd^2}\left(I\,k_d(L \cdot N) + I\,k_s(R \cdot V)^e \right)$$



L – vector to light source
N – surface normal vector
R – reflected light ray
V – vector to viewer/observer

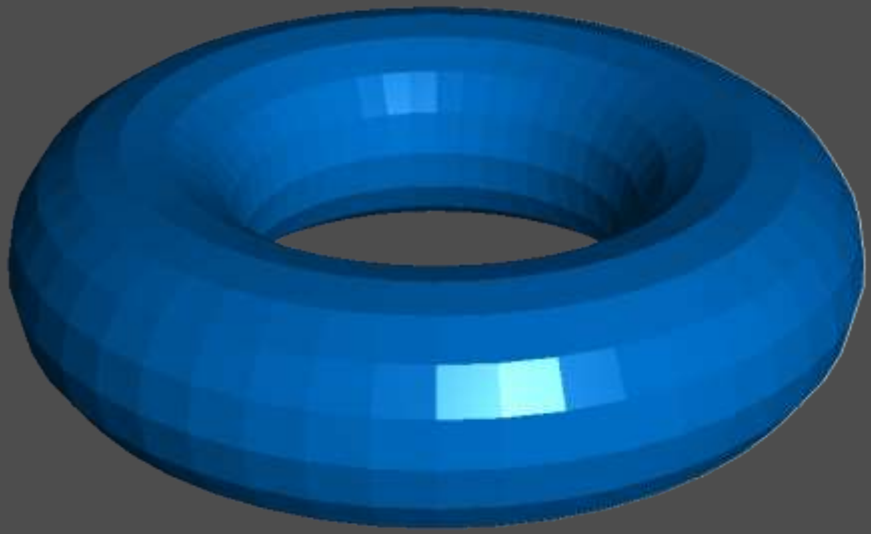
has to be evaluated for all light sources and for each of the base colors

Shading of Polygonal Models

- *status:* we can approximate color at a point
- *goal:* we want to render the whole model
- *constraint:* efficiency and quality
- *approach:* **shade** all pixels of a triangle based on color computation at a few points
- *three techniques:*
  - flat shading
  - Gouraud shading
  - Phong shading ($\neq$ Phong illumination)
- *shading $\neq$ shadow computation*

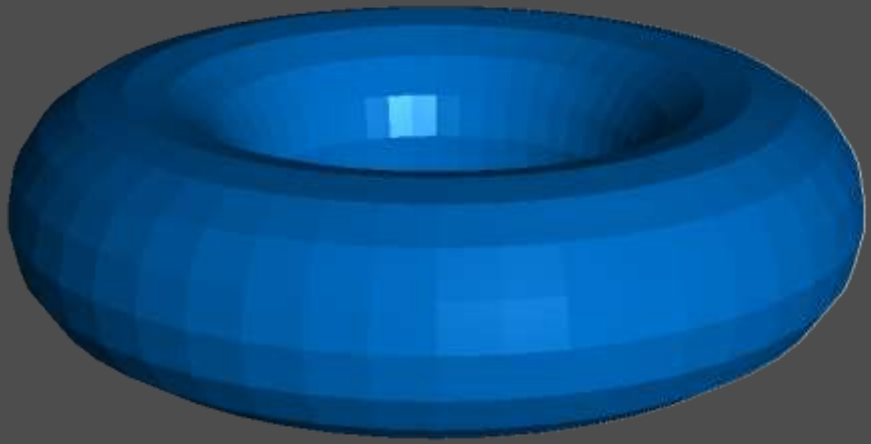
Flat Shading

- no interpolation
- all pixels same color
- two methods:
  - one point per triangle/quad
  - average of triangle's/quad's vertices
- low quality: single primitives easily visible
- fast computation & easy implementation



Flat Shading

- no interpolation
- all pixels same color
- two methods:
  - one point per triangle/quad
  - average of triangle’s/quad’s vertices
- low quality: single primitives easily visible
- fast computation & easy implementation



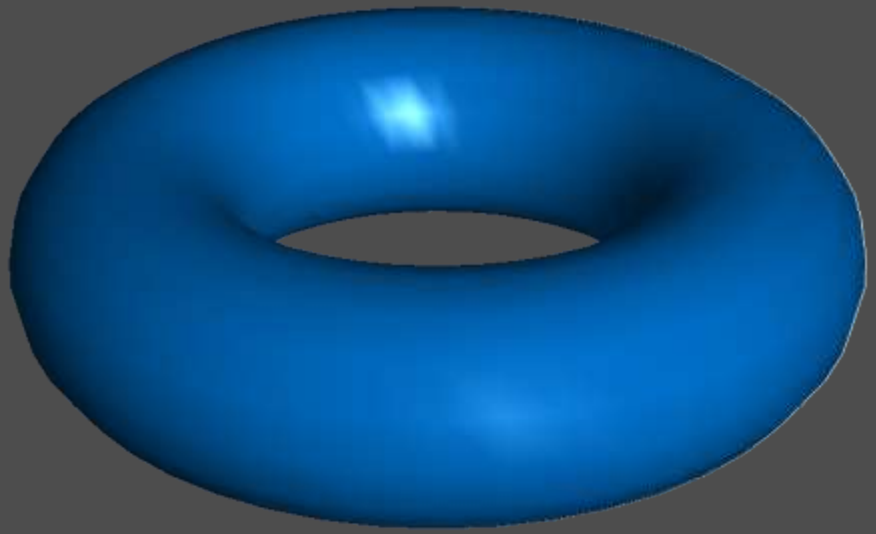
Flat Shading

- no interpolation
- all pixels same color
- two methods:
  - one point per triangle/quad
  - average of triangle’s/quad’s vertices
- low quality: single primitives easily visible
- fast computation & easy implementation

Flat Shading

- no interpolation
- all pixels same color
- two methods:
  - one point per triangle/quad
  - average of triangle's/quad's vertices
- low quality: single primitives easily visible
- fast computation & easy implementation

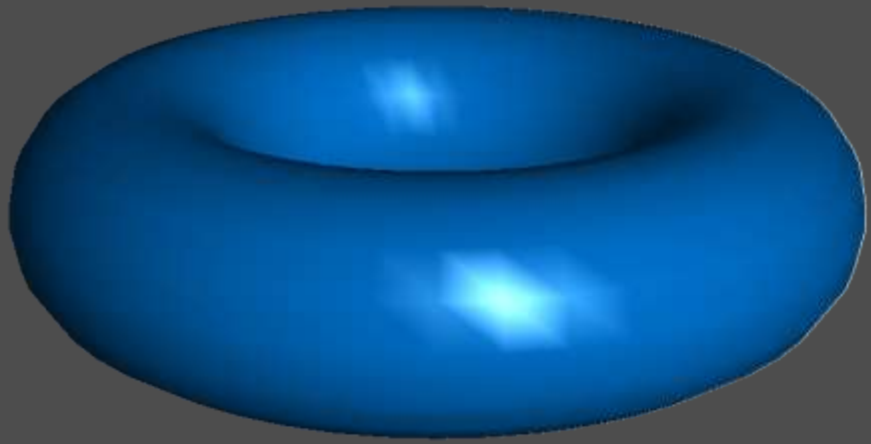
Gouraud Shading (1971)



- computation of colors at all vertices
- linear interpolation of colors over primitive
- more computation but better quality than flat shading
- usually implemented in graphics hardware
- highlights problematic: highlight shapes and highlights in the middle of triangles

Gouraud Shading (1971)

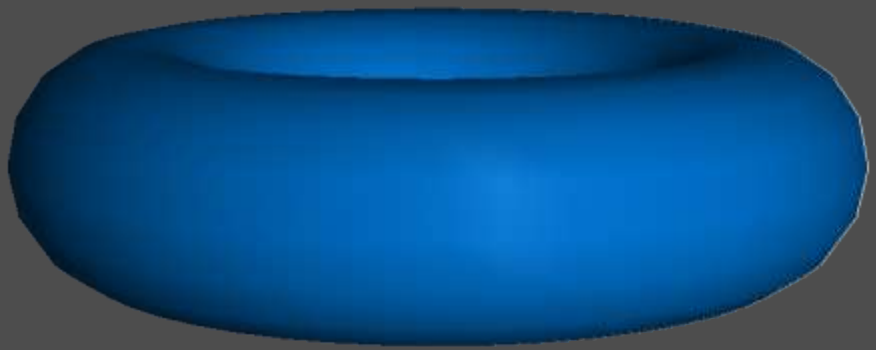
- computation of colors at all vertices
- linear interpolation of colors over primitive
- more computation but better quality than flat shading



- usually implemented in graphics hardware
- highlights problematic: highlight shapes and highlights in the middle of triangles

Gouraud Shading (1971)

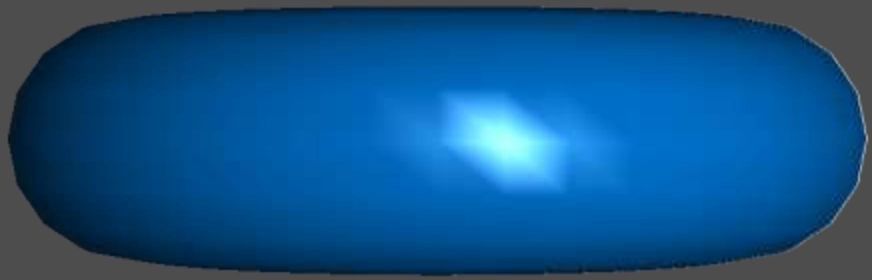
- computation of colors at all vertices
- linear interpolation of colors over primitive
- more computation but better quality than flat shading



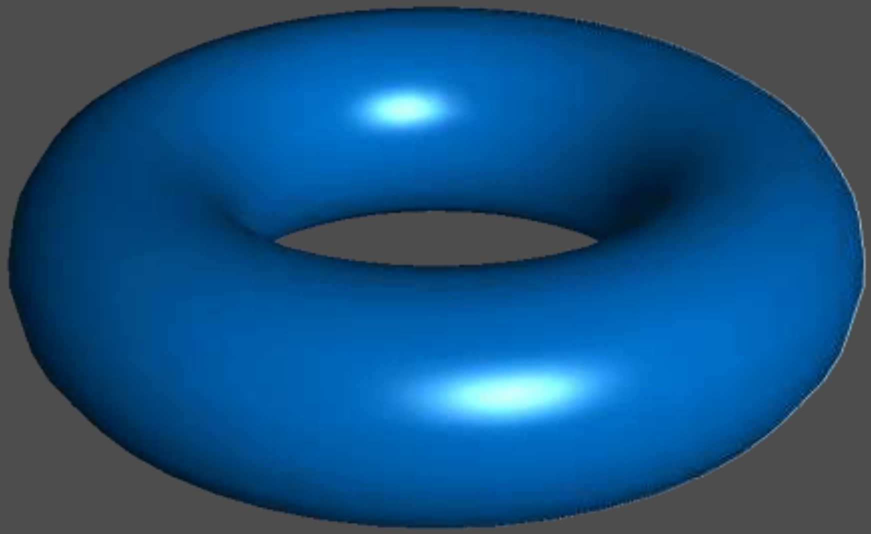
- usually implemented in graphics hardware
- highlights problematic: highlight shapes and highlights in the middle of triangles

Gouraud Shading (1971)

- computation of colors at all vertices
- linear interpolation of colors over primitive
- more computation but better quality than flat shading
- usually implemented in graphics hardware
- highlights problematic: highlight shapes and highlights in the middle of triangles



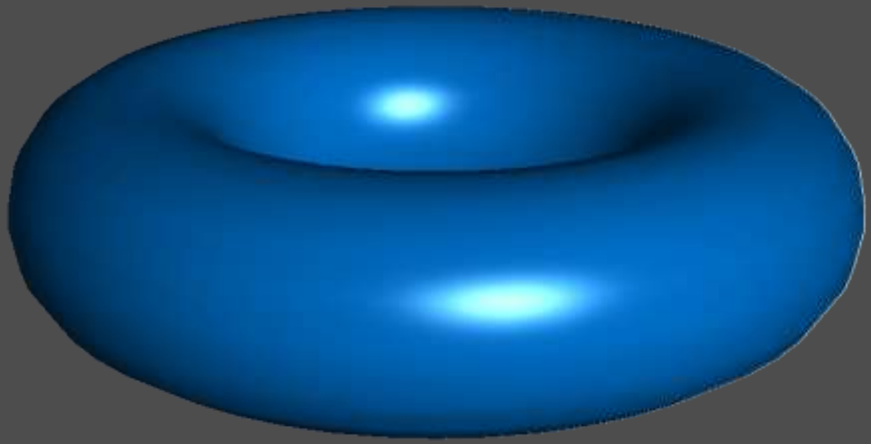
Phong Shading (1973)



- linear interpolation of normals for each pixel
- color computation for each pixel separately
- best quality, highlights are shown correctly
- but **computationally more expensive**
- problems:
  - polygons still visible at silhouettes
  - traditionally not implemented in hardware (nowadays not a problem with shaders)

Phong Shading (1973)

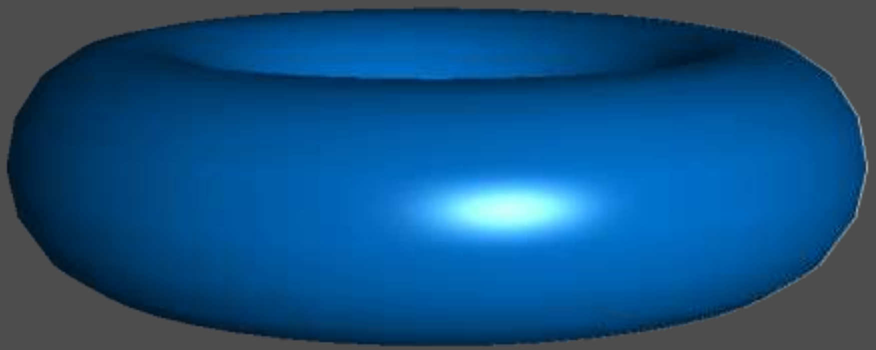
- linear interpolation of normals for each pixel
- color computation for each pixel separately
- best quality, highlights are shown correctly



- but **computationally more expensive**
- problems:
  - polygons still visible at silhouettes
  - traditionally not implemented in hardware (nowadays not a problem with shaders)

Phong Shading (1973)

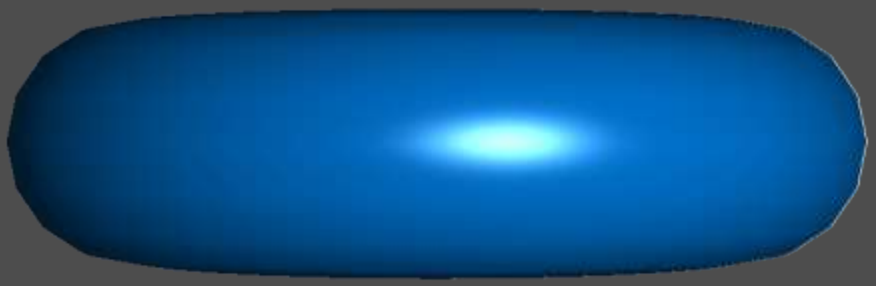
- linear interpolation of normals for each pixel
- color computation for each pixel separately
- best quality, highlights are shown correctly



- but **computationally more expensive**
- problems:
  - polygons still visible at silhouettes
  - traditionally not implemented in hardware (nowadays not a problem with shaders)

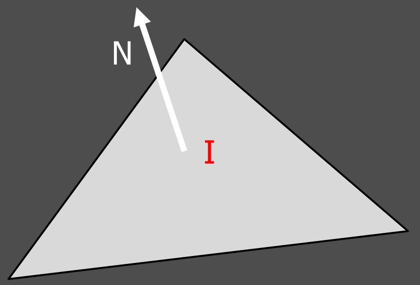
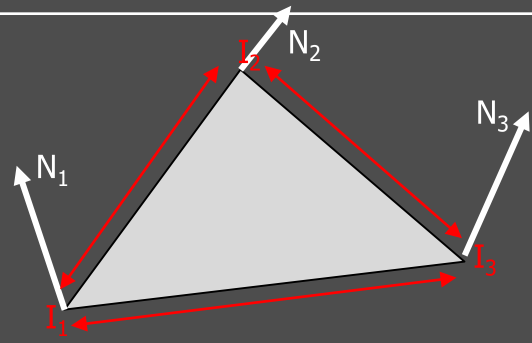
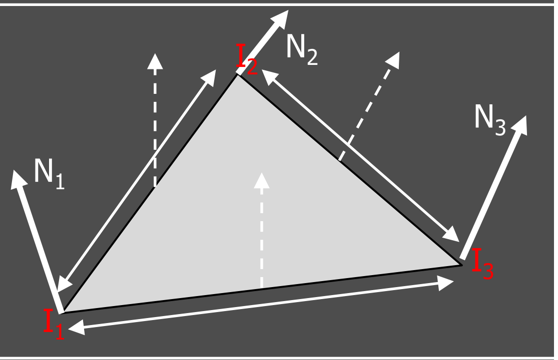
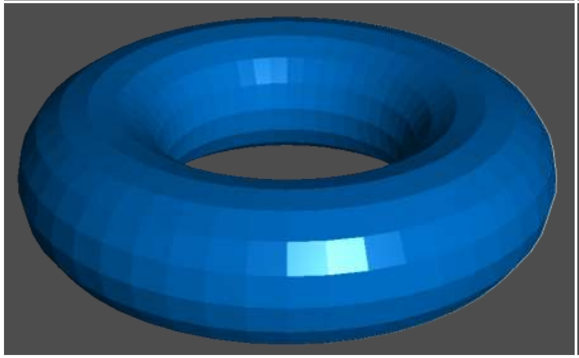
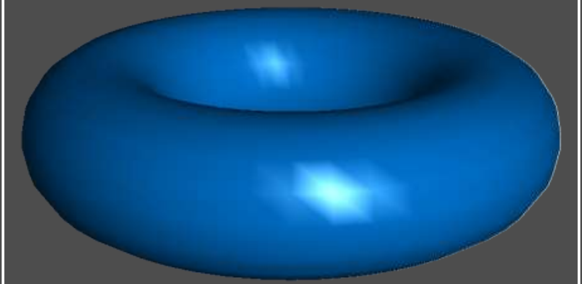
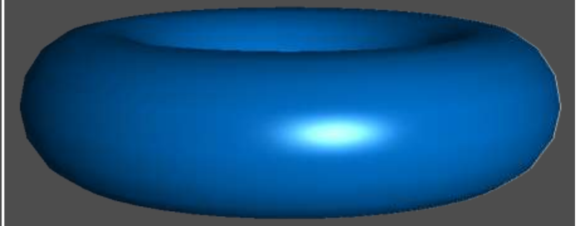
Phong Shading (1973)

- linear interpolation of normals for each pixel
- color computation for each pixel separately
- best quality, highlights are shown correctly



- but **computationally more expensive**
- problems:
  - polygons still visible at silhouettes
  - traditionally not implemented in hardware (nowadays not a problem with shaders)

Polygonal Shading: Comparison

<table>
<tr><th>Flat</th><th>Gouraud</th><th>Phong</th></tr>
<tr>
<td>
</td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>one color value for entire polygon</td>
<td>one color value per vertex & interpolation inside the triangle</td>
<td>vertex normals interpolated and one color value per pixel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</table>

Texture Mapping

Texture Mapping Motivation

- so far: detail through polygons & materials
- example: brick wall
- problem: many polygons & materials needed for detailed structures
  $\rightarrow$ inefficient for memory and processing
- new approach necessary: texture mapping
- introduced by Ed Catmull (1974), extended by Jim Blinn (1976)

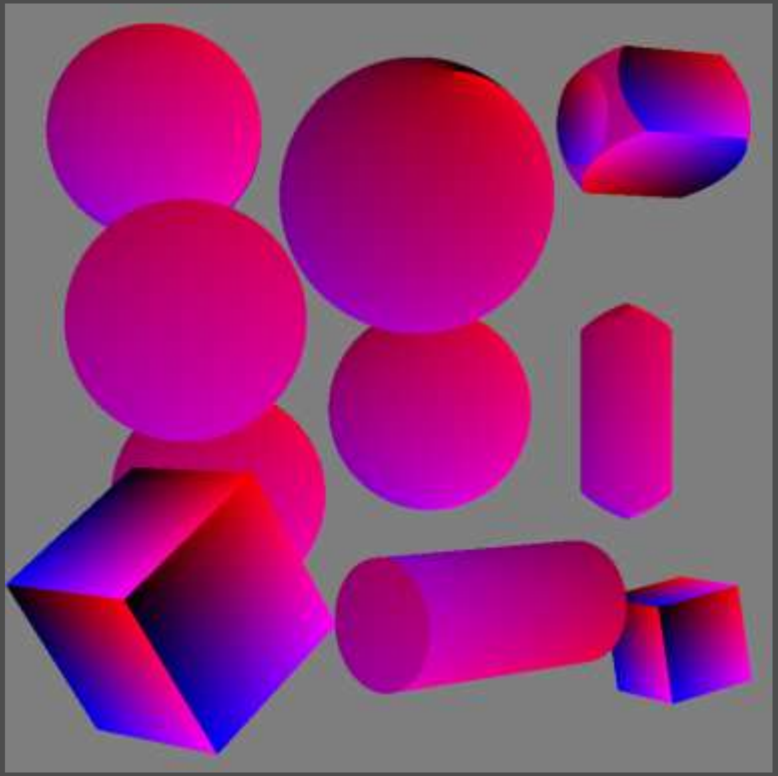
Texture Mapping Motivation

- several properties can be modified
  - color: diffuse component of surface
  - reflection: specular component of surface to simulate reflection (environment mapping)
  - normal vector: simulate 3D surface structure (bump mapping)
  - actual surface: raise/lower points to actually modify surface (displacement mapping)
  - transparency: make parts of a surface entirely or to a certain degree transparent

Inherent Texture Coordinates

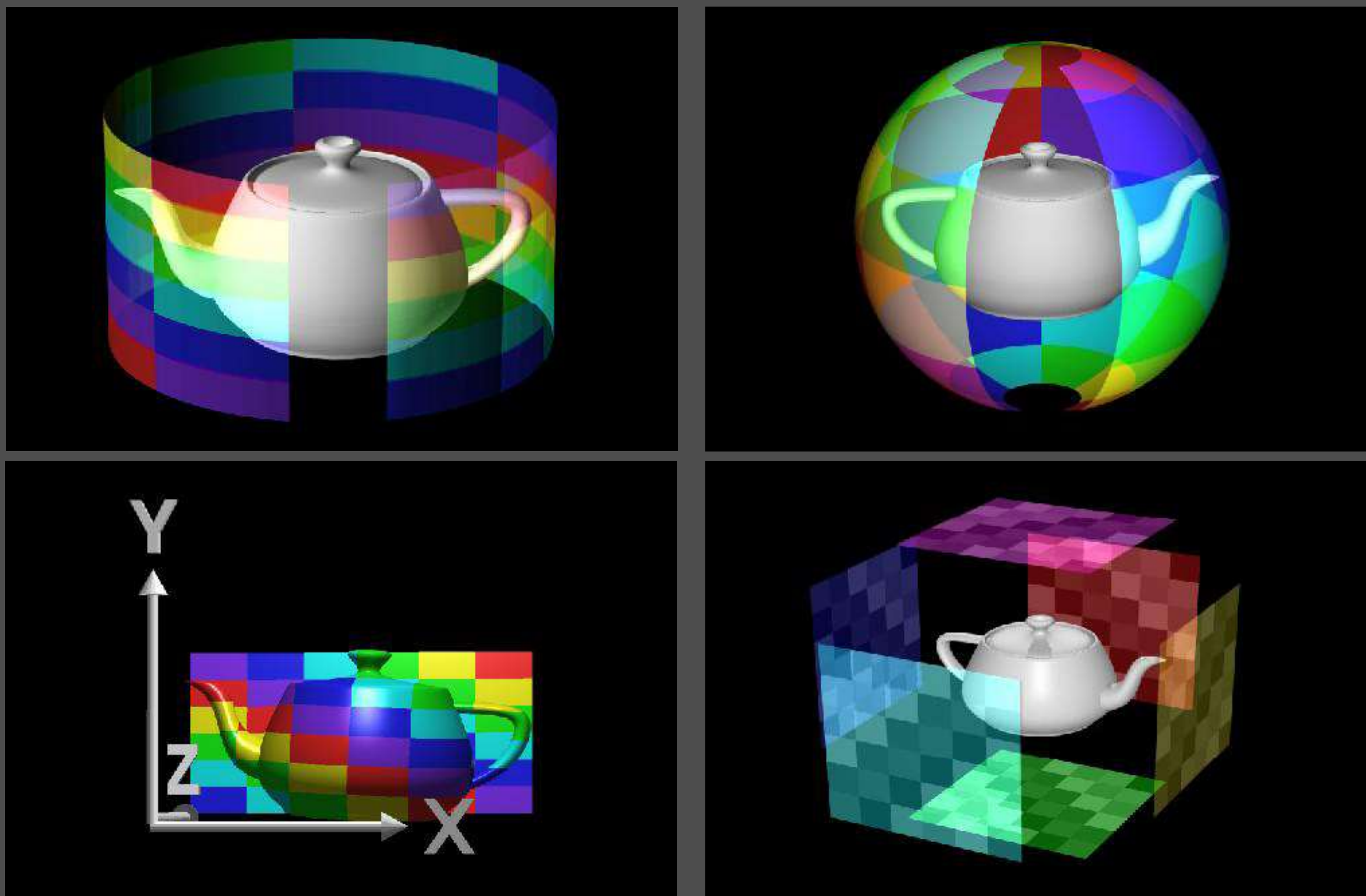
- (*u*, *v*) coordinates derived from parameter directions of surface patches (e.g., Bézier and spline patches)

- obvious (*u*, *v*) coordinates derived for primitive shapes (e.g., boxes, spheres, cones, cylinders, etc.)



2-Step Mapping

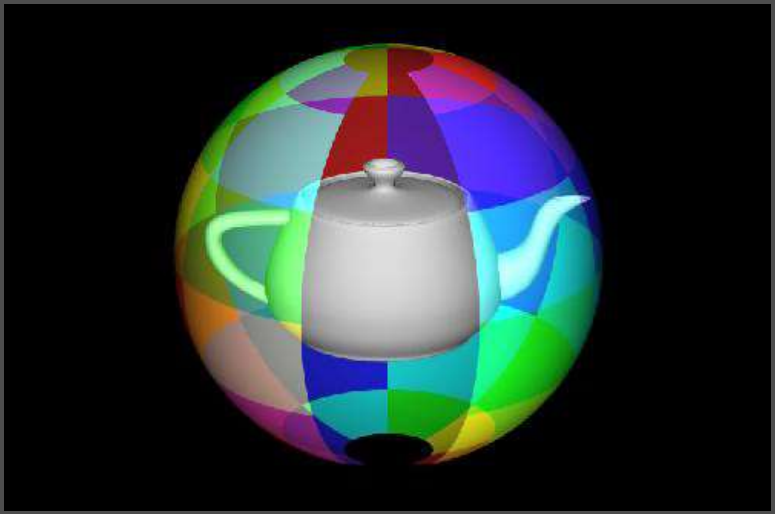
- four models: cylinder, sphere, plane, cube

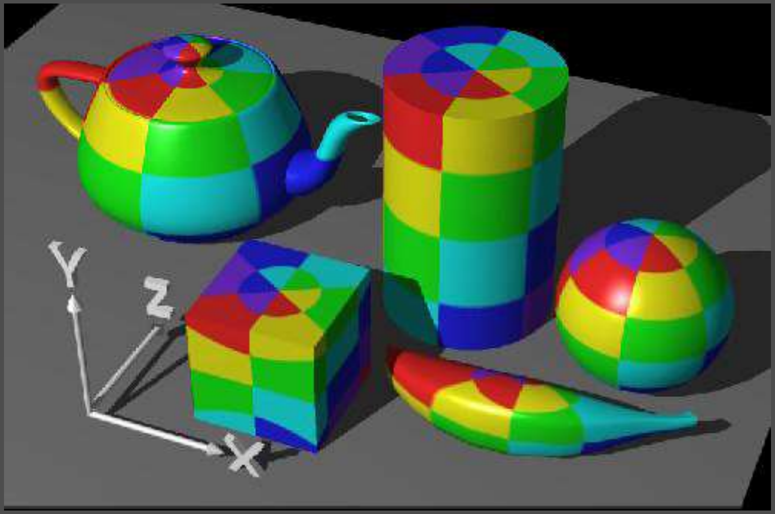


from R. Wolfe: *Teaching Texture Mapping*

2-Step Spherical Mapping

- mapping onto surface of a sphere given by spherical coordinates
- no non-distorting mapping possible between plane and sphere surface

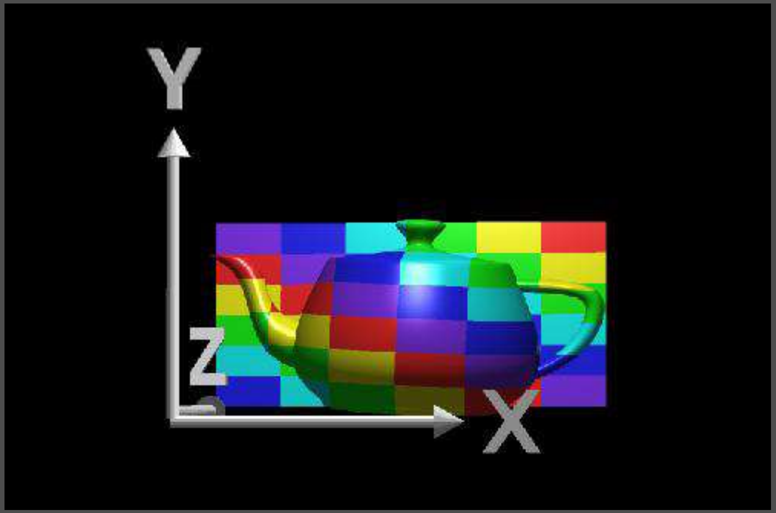


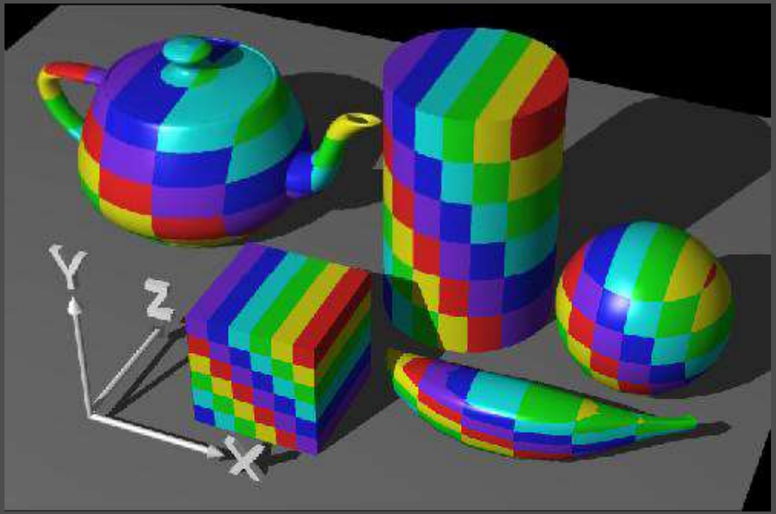


from R. Wolfe: *Teaching Texture Mapping*

2-Step Planar Mapping

- mapping onto planar surface given by position vector and two additional vector

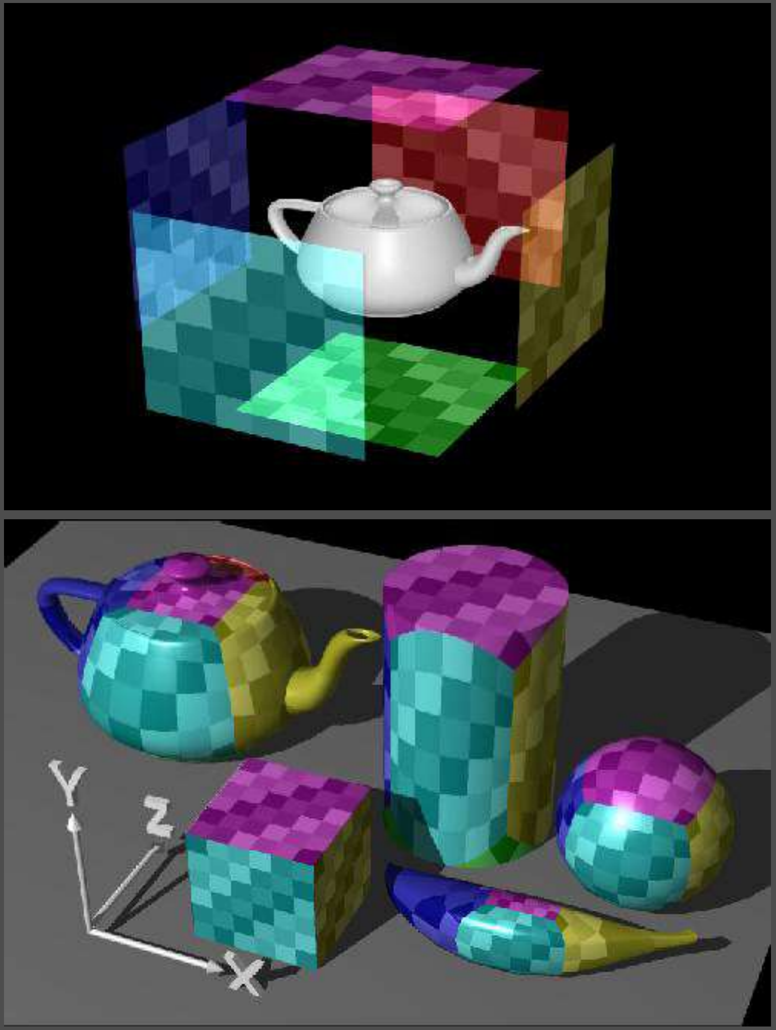




from R. Wolfe: *Teaching Texture Mapping*

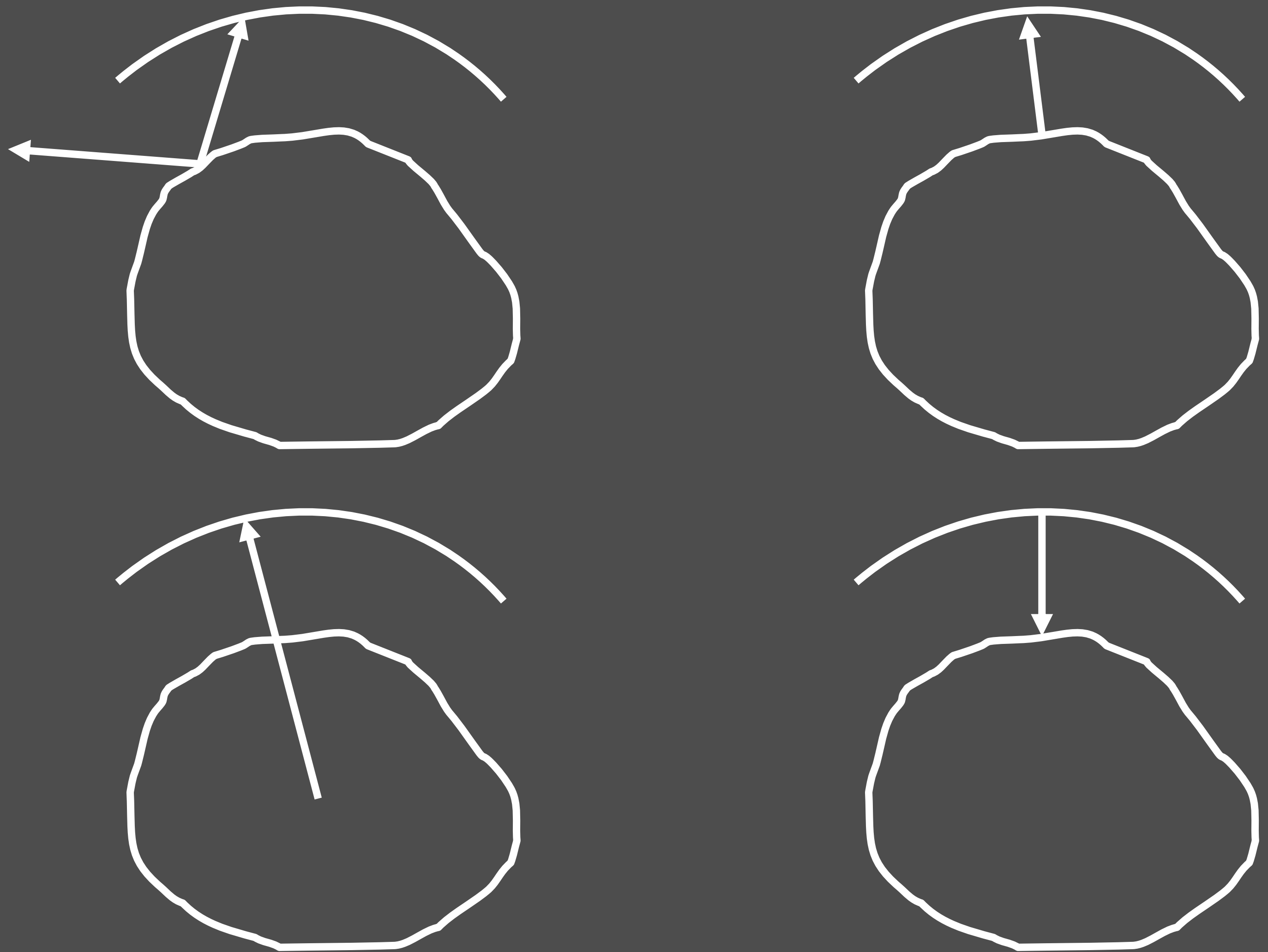
2-Step Box Mapping

- enclosing box is usually axis-parallel bounding box of object
- six rectangles onto which the texture is mapped
- similar to planar mapping



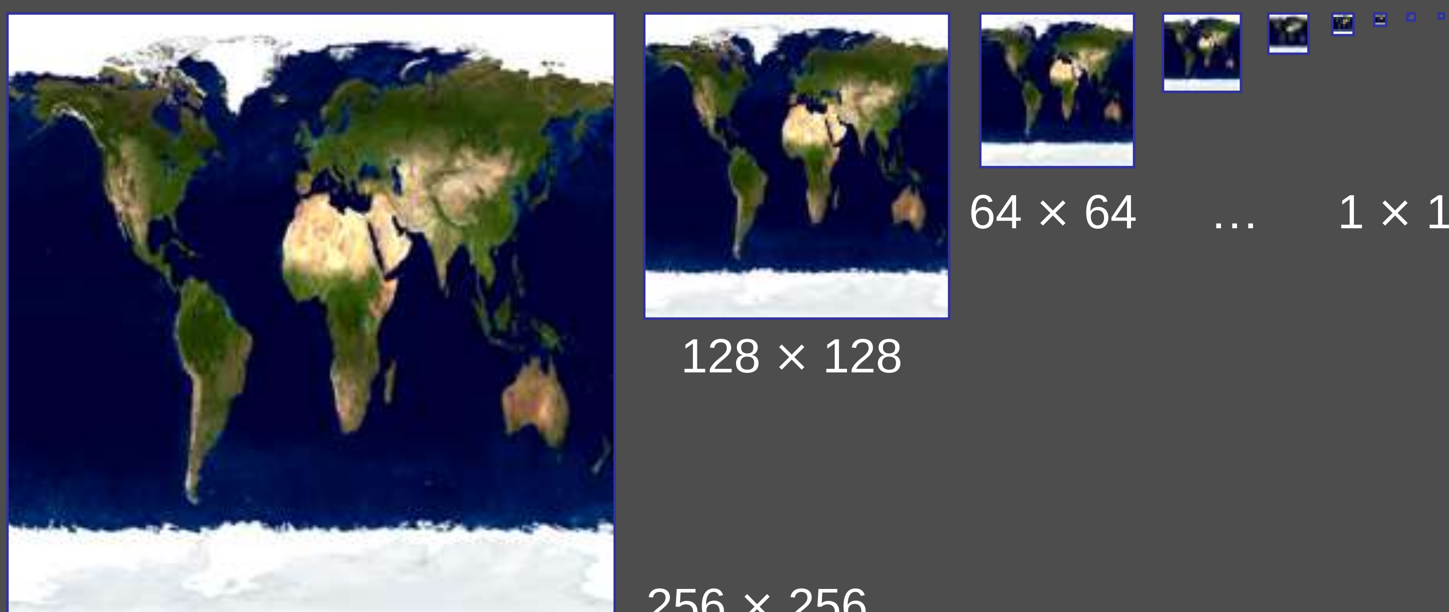
from R. Wolfe: *Teaching Texture Mapping*

O Mapping: Object to Surface



Texture Mapping: Mip Mapping

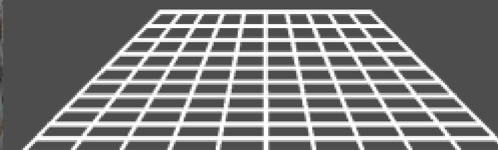
- optimal texture mapping (speed & quality): texel size ≈ pixel size
- idea: use stack of textures and select the most appropriate one w.r.t. situation

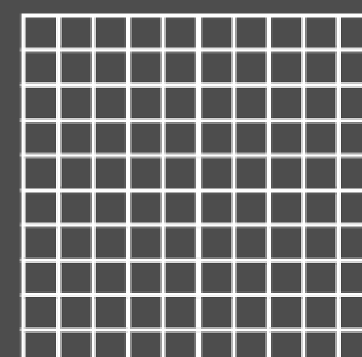


256 × 256 128 × 128 64 × 64 … 1 × 1

Texture Mapping: Anisotropic Filtering

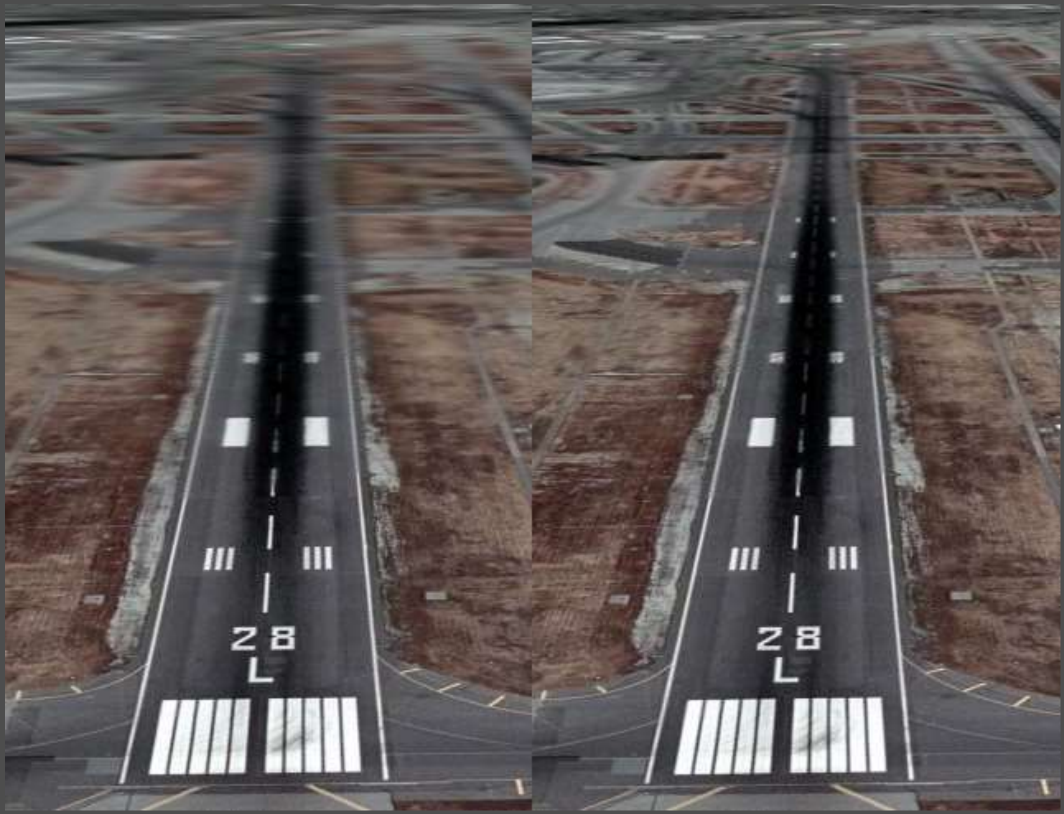
- large textures not perpendicular to viewing direction: blurring problems w.r.t. angle
- appropriate mip map selection not possible



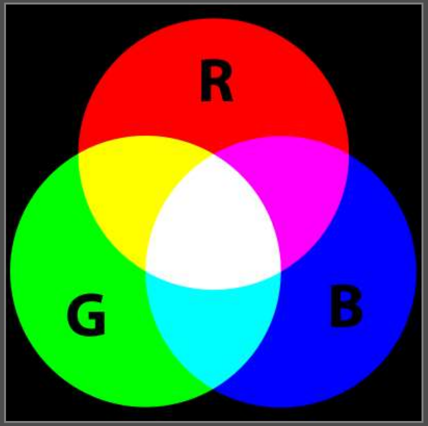


Texture Mapping: Anisotropic Filtering

- large textures not perpendicular to viewing direction: blurring problems w.r.t. angle
- appropriate mip map selection not possible
- generate mip maps favoring one direction: 256×128, 256×64, 128×64, 128×32, ...



Color and Color Models



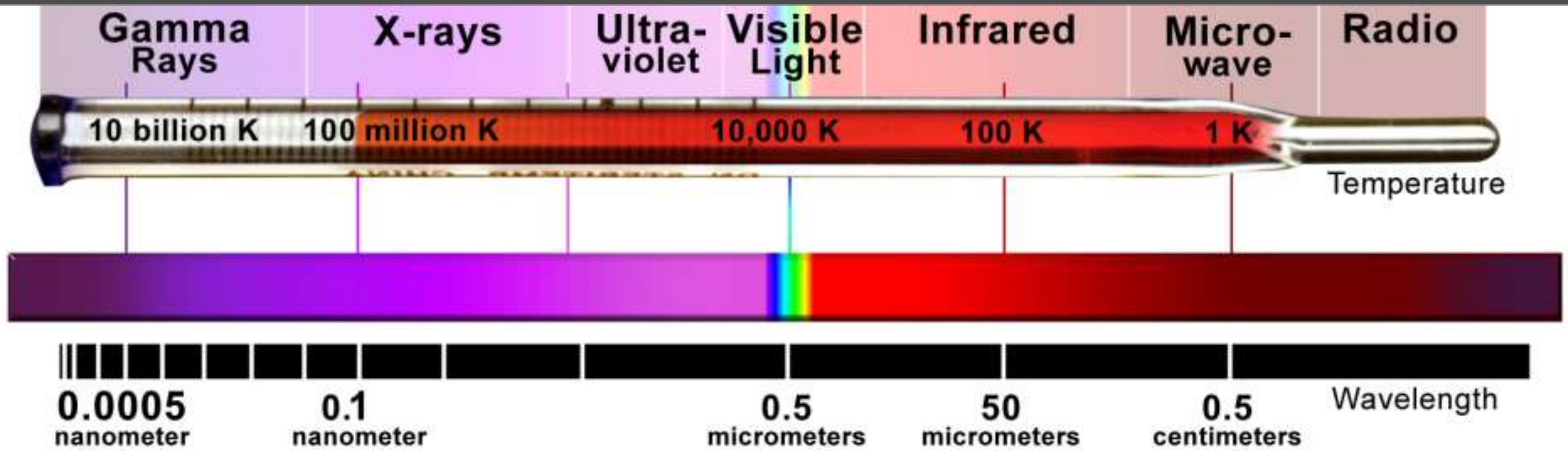
What is color?

What is color?

What is Color?

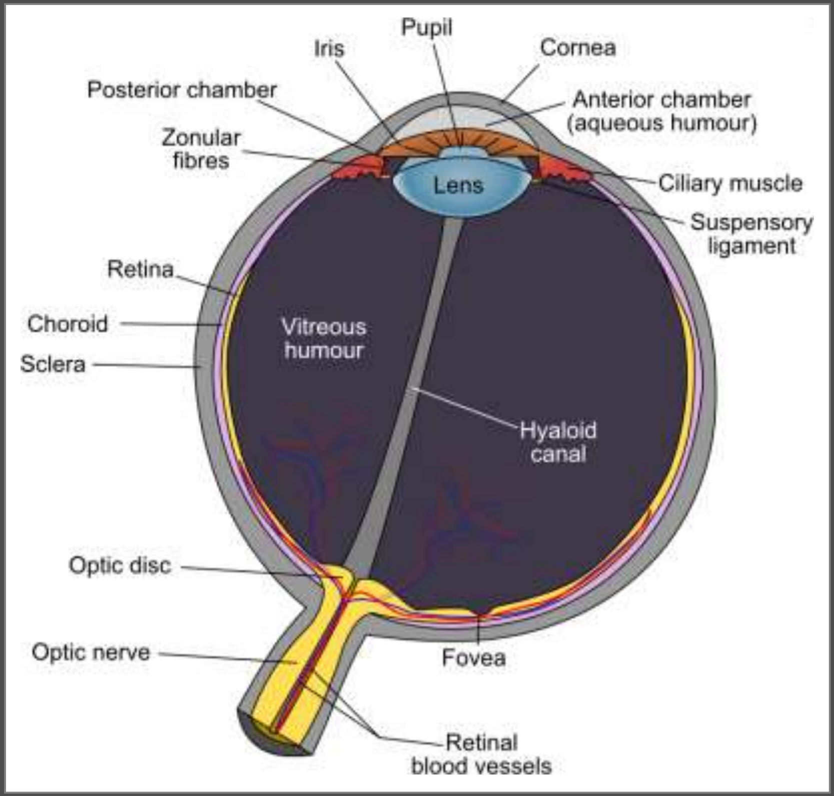
- color is a **human reaction** to light (change) (can also be influenced by cultural background)
- what is light?
- light is the visible part (370–730nm) of the electromagnetic spectrum

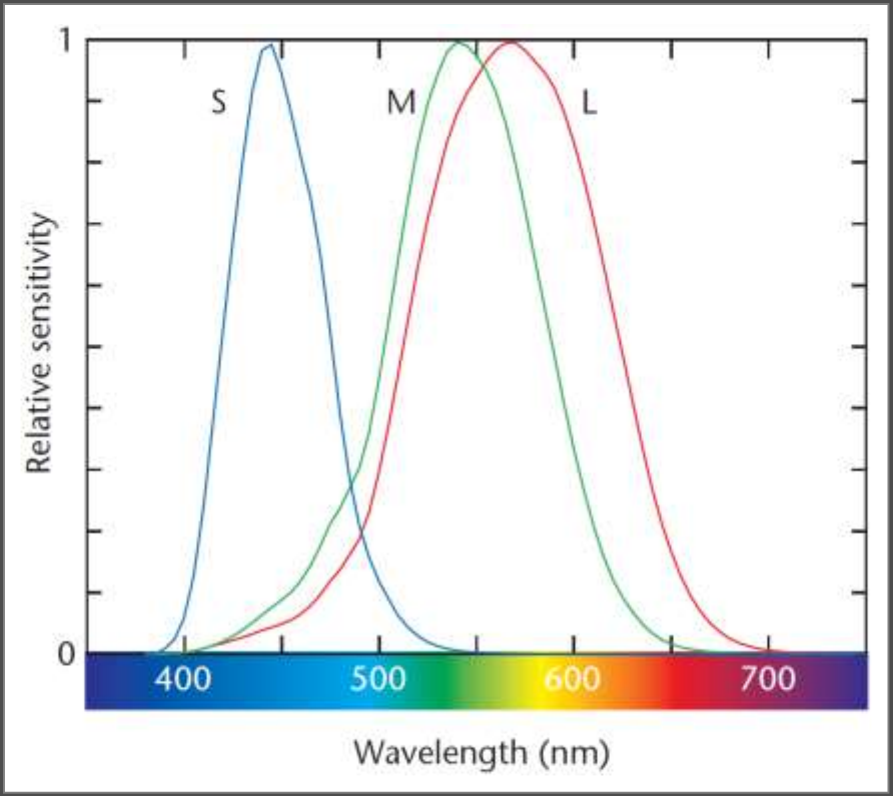
from NASA



Human Color Perception

- light is converted into signals by cone cells
- three cone types with different sensitivities

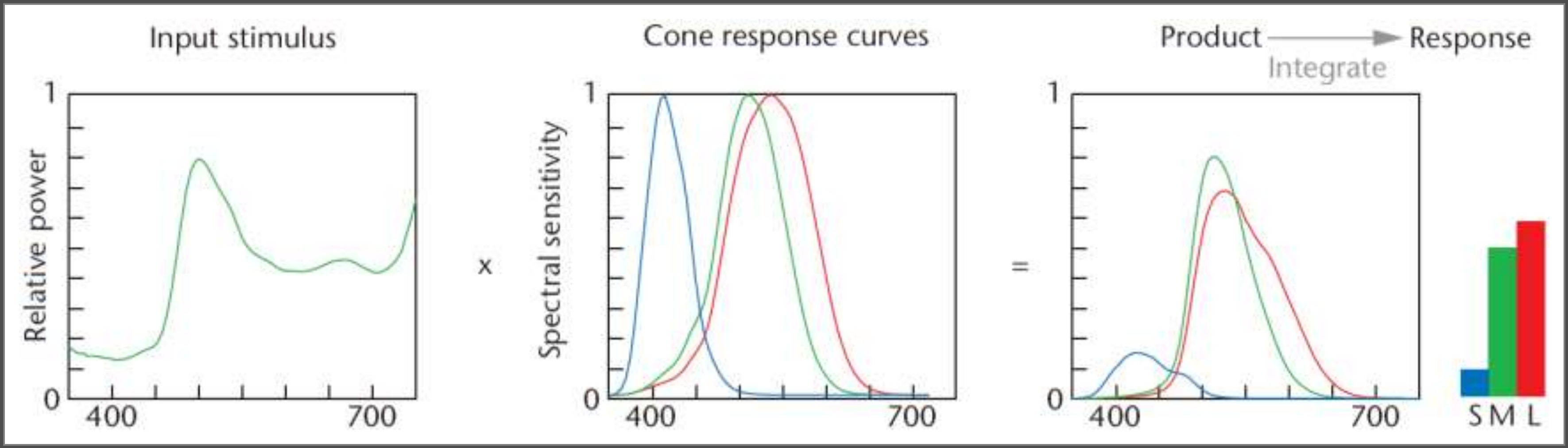




Stone 2005

Human Color Perception

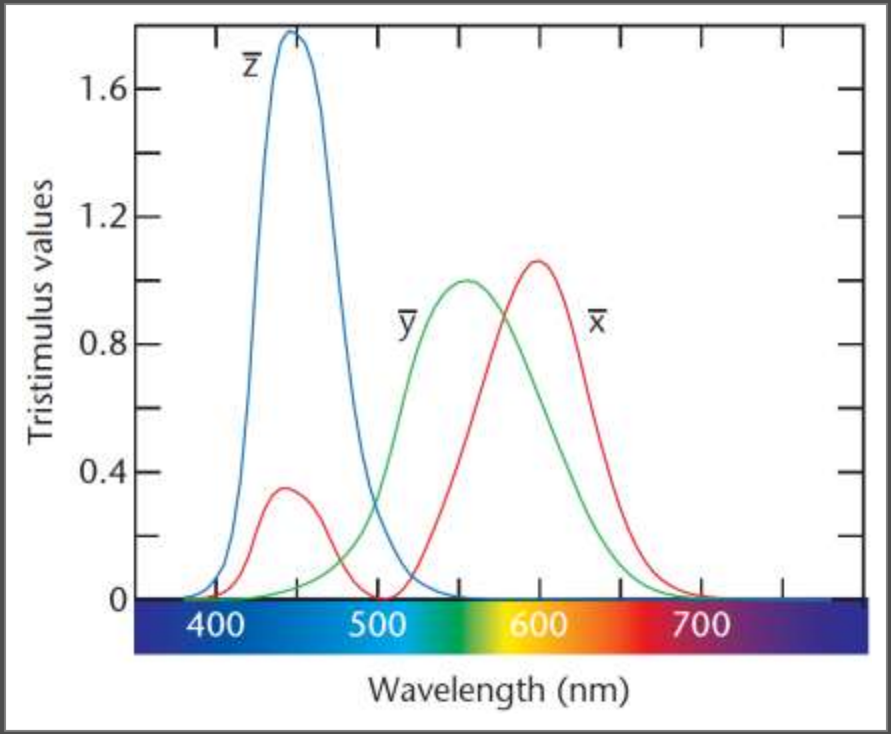
- colored light = spectral distribution function: light intensity as function of wavelength
- converted into 3 response values by cones (short, medium, and long wavelengths)



Stone 2005

Describing Color Vision: XYZ Color Model

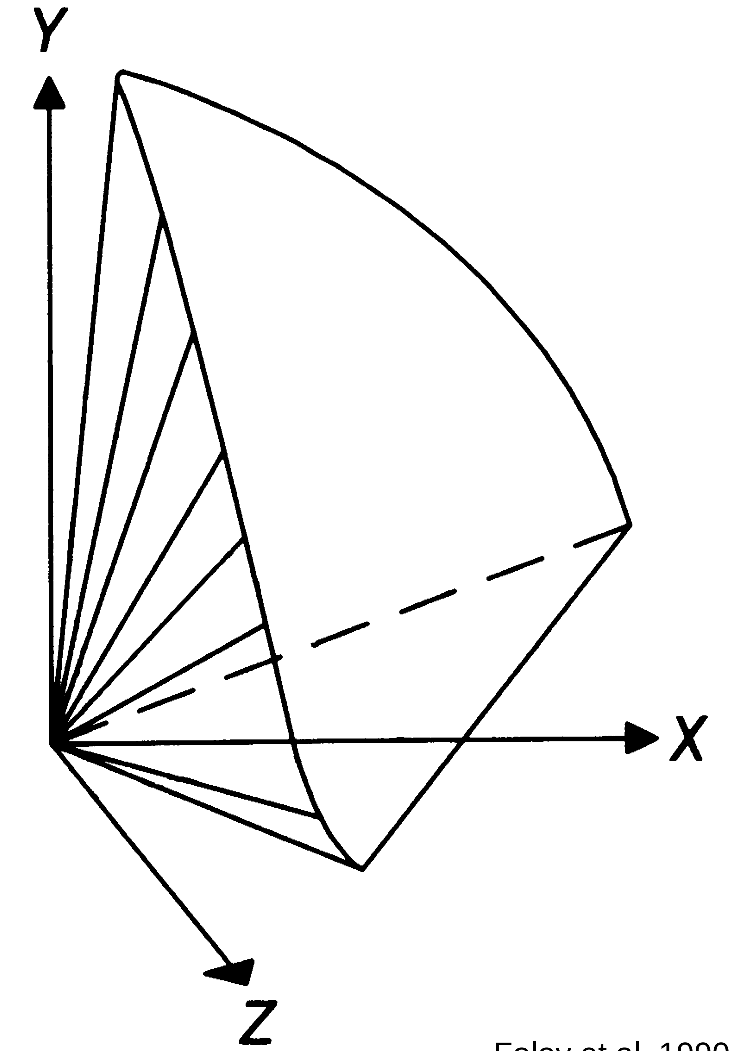
- definition of three primary colors: X, Y, Z
  - color-matching functions are non-negative
  - Y follows the standard human response to luminance, i.e., the Y value represents perceived brightness
  - can represent all perceivable colors
- mathematically derived from experiments



Stone 2005

XYZ CIE Color Space

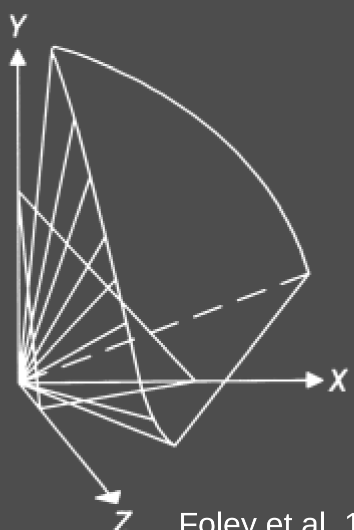
- plotting XYZ space in 3D
- all colors that are perceivable by humans form a deformed cone
- *X*, *Y*, and *Z*-axes are outside this cone

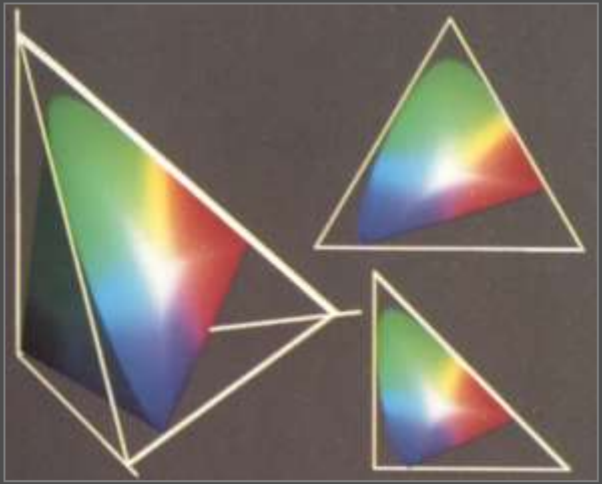


Foley et al. 1990

CIE Chromaticity Diagram

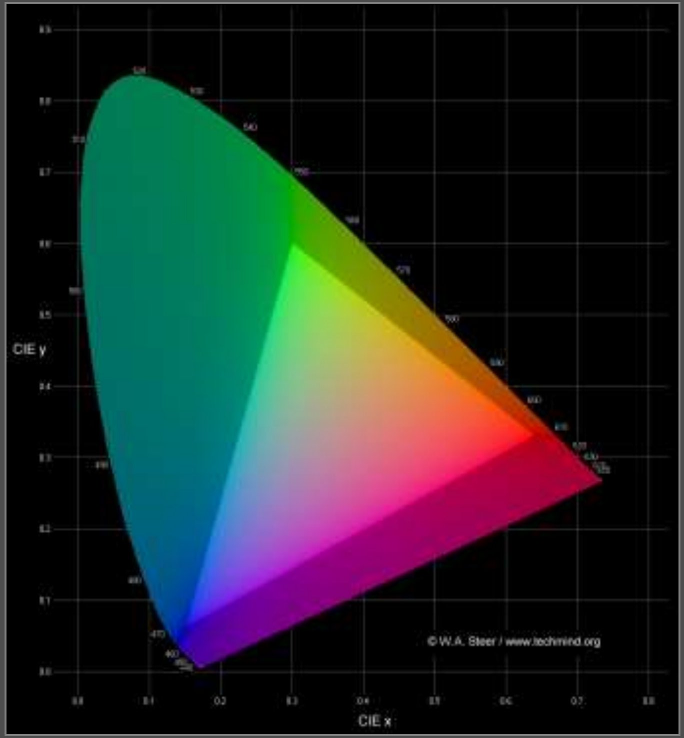
- projection of XYZ space onto X+Y+Z = 1 (to factor out a color's brightness):
  $x = X/(X+Y+Z)$ $\qquad$ $y = Y/(X+Y+Z)$
- monochromatic colors on upper edge
- color gamut: colors visible on a device through color adding





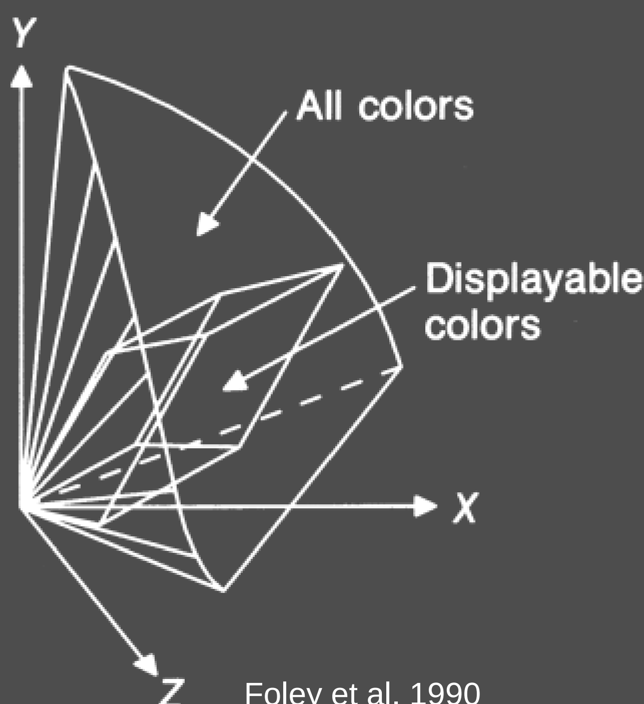
Foley et al. 1990

http://www.techmind.org/

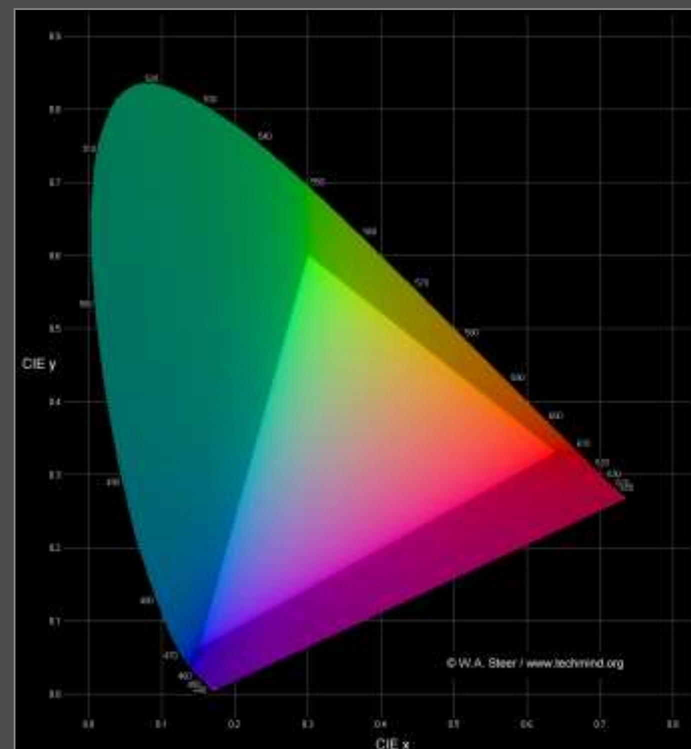


Can RGB Represent Any Color?

- no, because all colors form horseshoe shape in CIE chromaticity diagram and RGB gamut is triangular

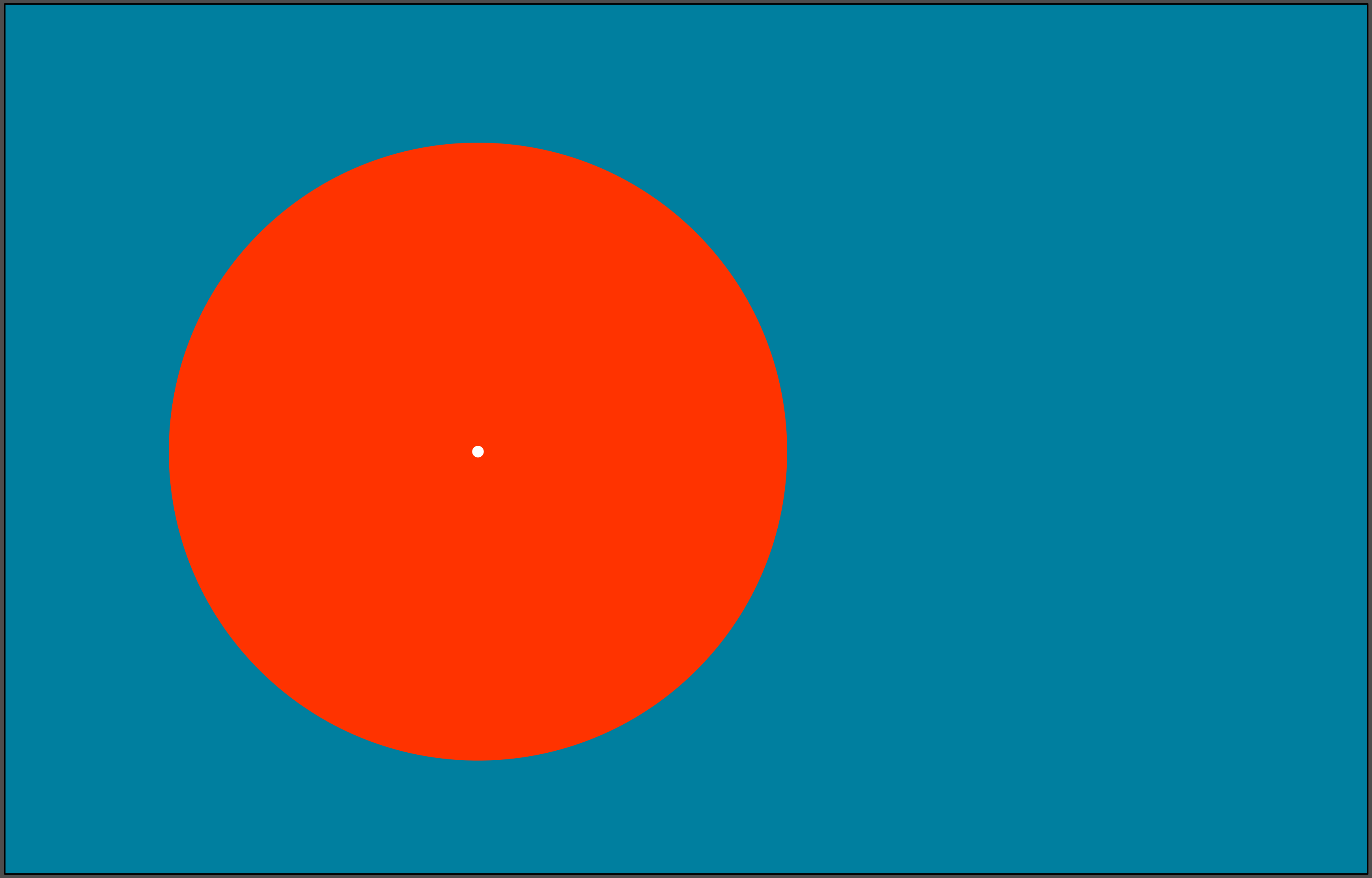


Foley et al. 1990

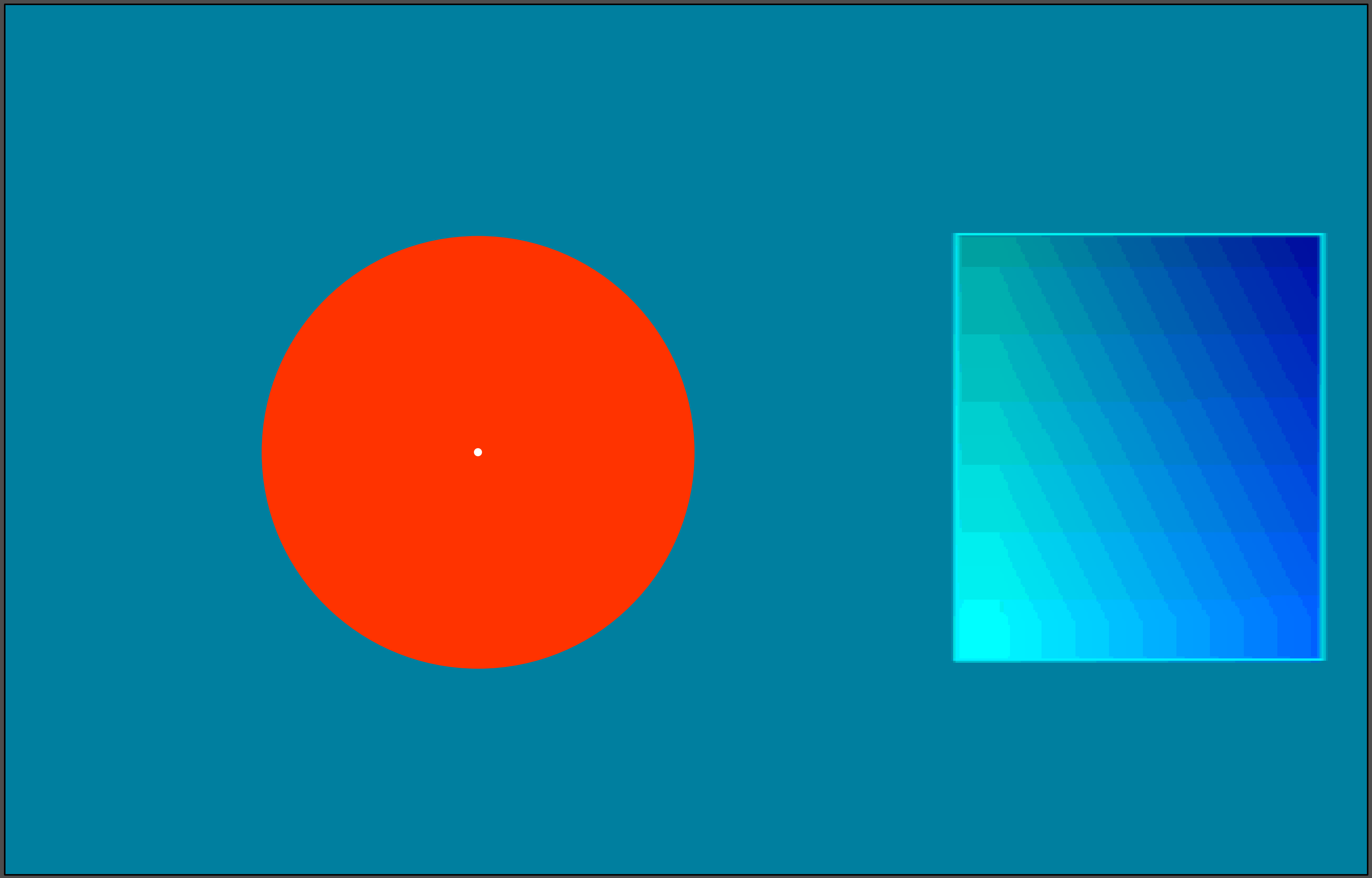


http://www.techmind.org/

Let’s see REAL cyan …

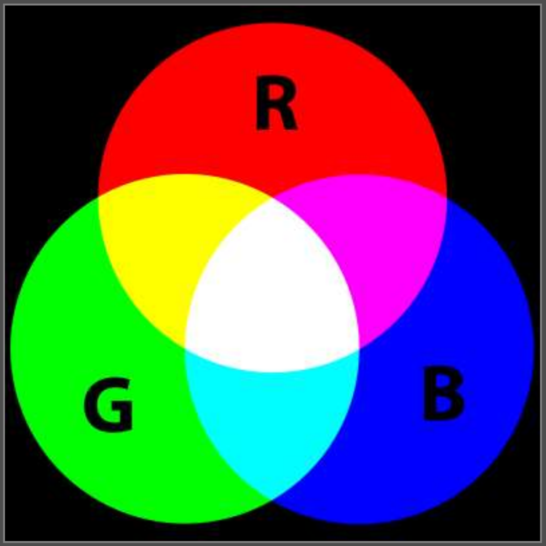


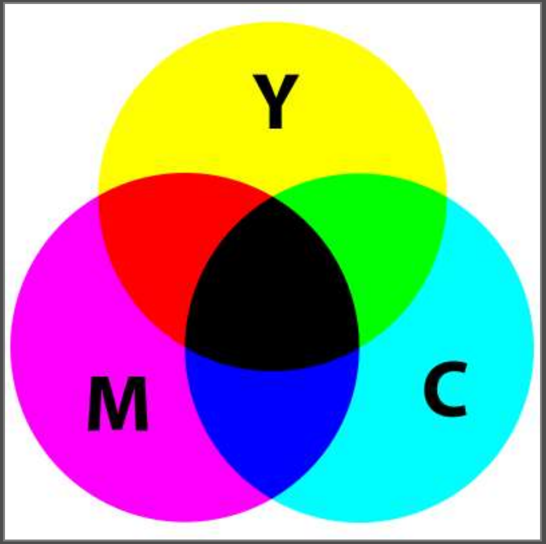
Let’s see REAL cyan …



Additive vs. Subtractive Color

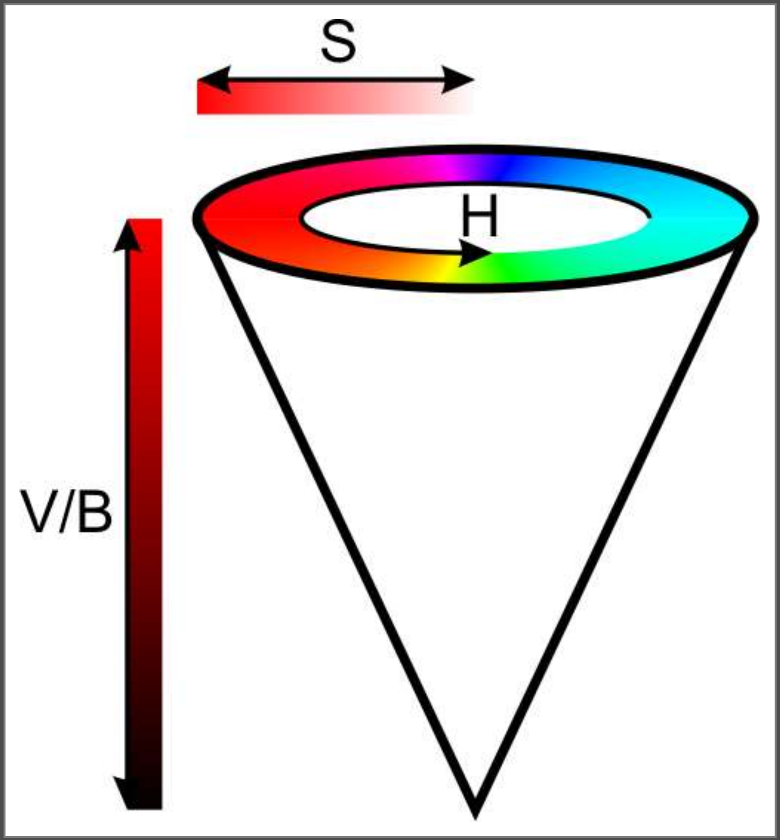
- (physical) color mixing depends on color production process
- device-dependent color models
  - light emission: additive mixing (CRTs etc.): **RGB model**
  - light absorption: subtractive mixing (printing process): **CMY(K) model**

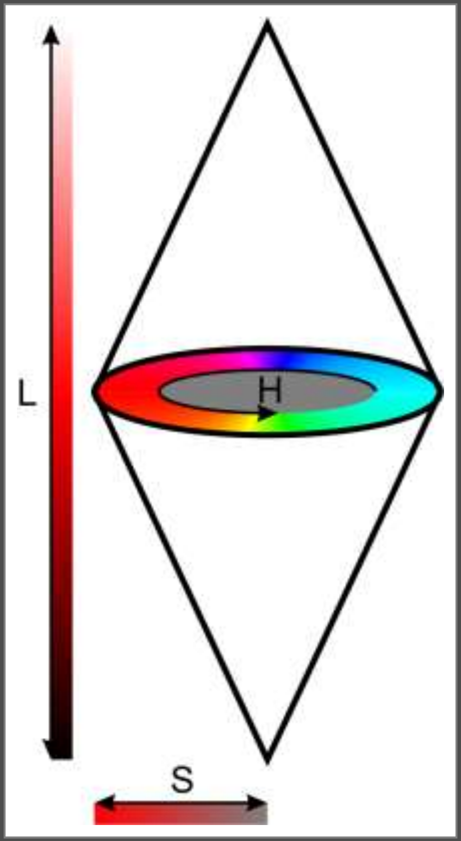




HSV/HSB and HSL/HLS Color Models

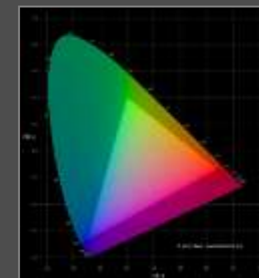
- human's inefficient with device models
- perceptual models to ease color specification

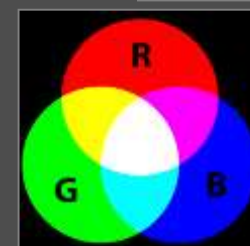


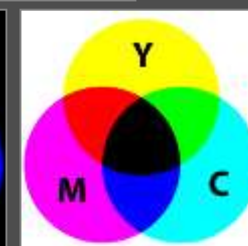


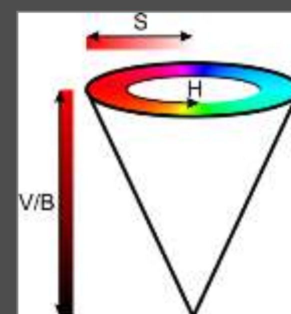
Color Models Summary

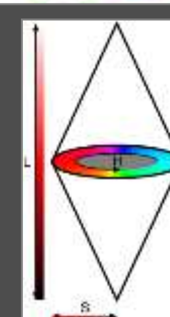
- human vision-oriented
  - CIE XYZ & LMS
- hardware-oriented
  - RGB & CMY(K)
- perceptual models
  - HSV/HSB & HSL/HLS
- other models (e.g., for TV)
  - YIQ (NTSC) & YUV (PAL)





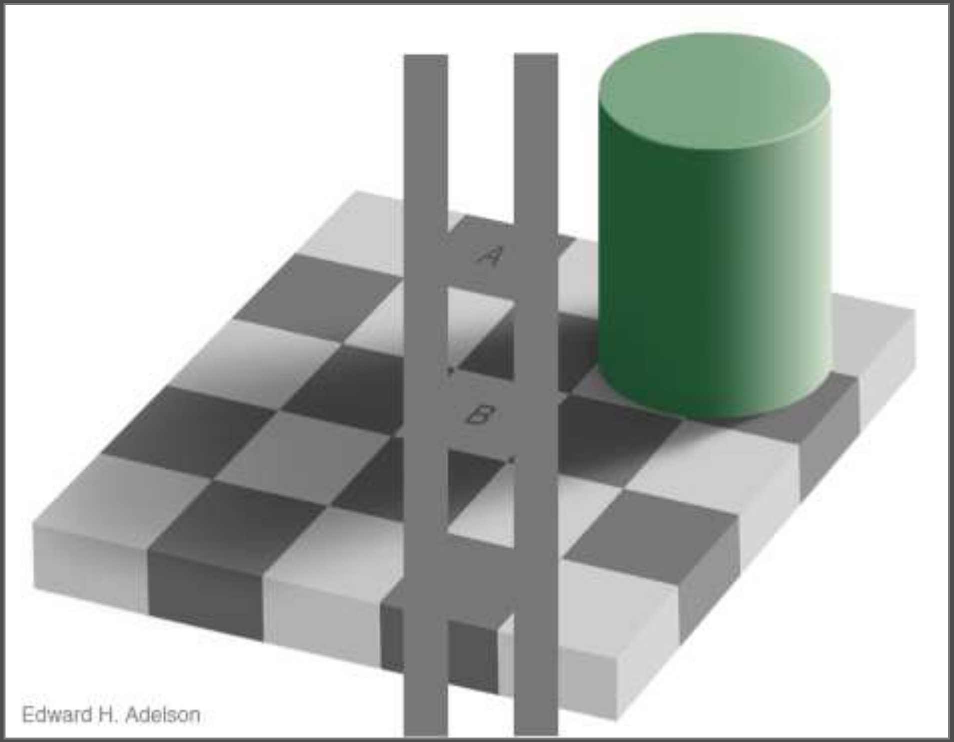






Other Color Topics

- color perception depending on context
- color deficiency



Other Color Topics

- color perception depending on context
- color deficiency