A Brief Refreshing Course of Computer Graphics

Tobias Isenberg

The Overall Goal: Photorealism

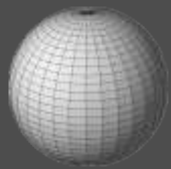
Gilles Tran, using POV-Ray 3.6

The Second Goal: Speed and Efficiency

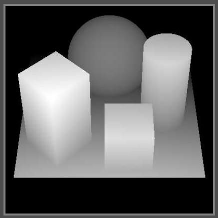
Ryse (2013), by Crytek/Microsoft Studios
Doom (1993), by id Software

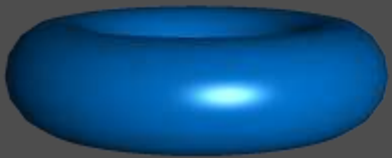
Computer Graphics 101

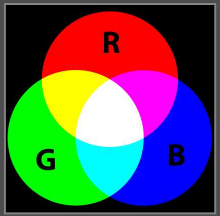
- computer graphics pipeline
- models and object representation
- transformations and projections
- hidden surface removal
- illumination and shading
- texture mapping
- color and color models
- hardware rendering
- local vs. global illumination
- curves and surfaces

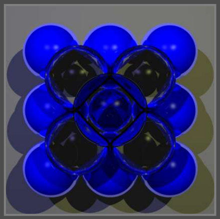


$$\begin{pmatrix} x' \\ y' \\ w' \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ w \end{pmatrix}$$



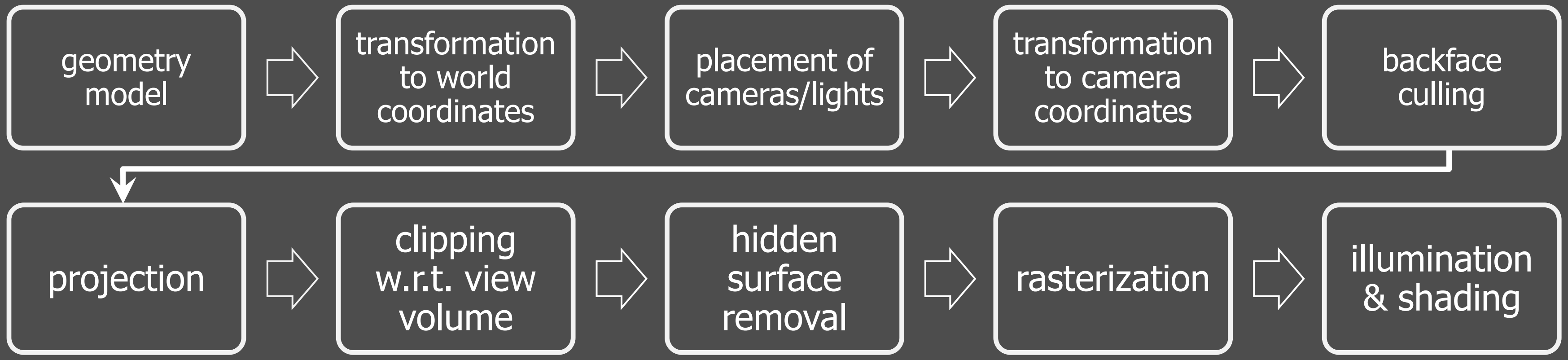






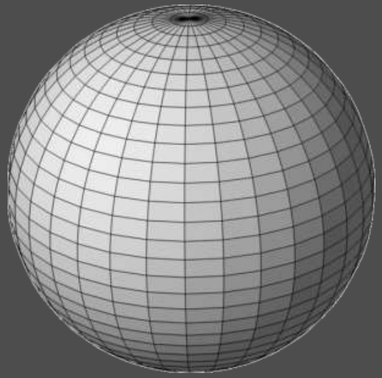
Computer Graphics Pipeline

Computer Graphics/Rendering Pipeline



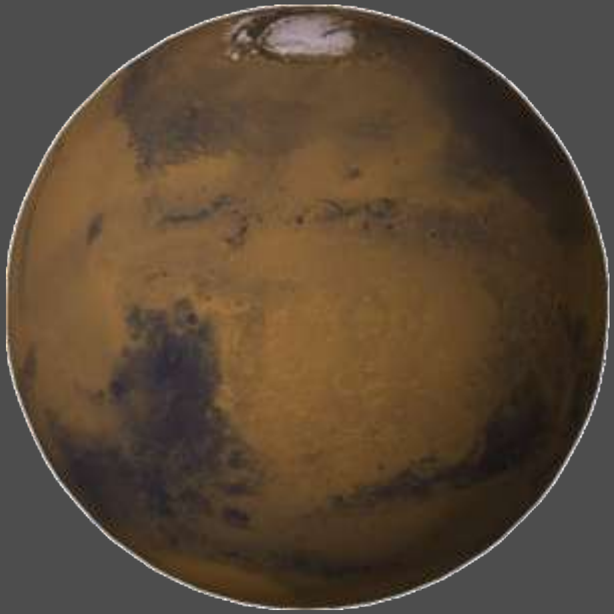
- simple model of physical processes
- independent parallel processing of triangles
- control/parameterization of the individual stages
- implementation of the specific stages in hardware

Models and Object Representation



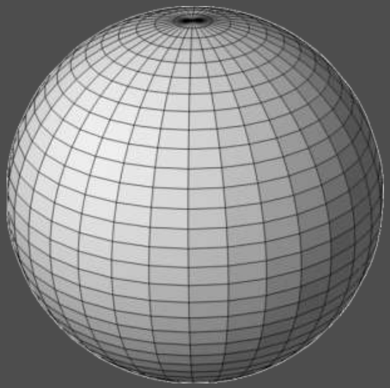
What to Specify for a 3D Shape?

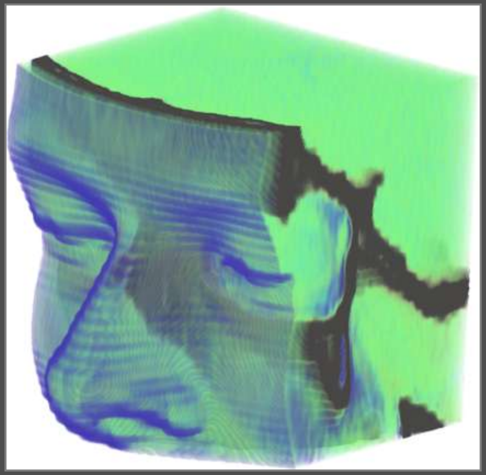
- geometry
  - shapes, positions
  - connectivity, inside/outside
- material properties
  - visuals, textures (plastic, wood, metal, etc.)
  - other material properties (elasticity, mass, etc.)
- behaviour/animation
- more depending on the specific application



How to Specify a 3D Geometry?

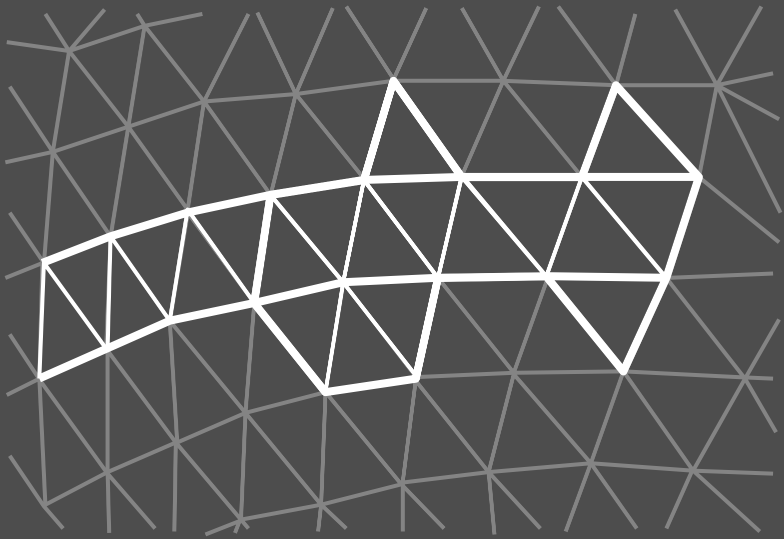
- boundary representations (b-reps)
  - meshes
  - piecewise smooth surfaces
- volume representations
  - voxel models
  - implicit surfaces
  - CSG: constructive solid geometry
  - space partitioning
    - BSP trees: binary space partitioning
    - octrees

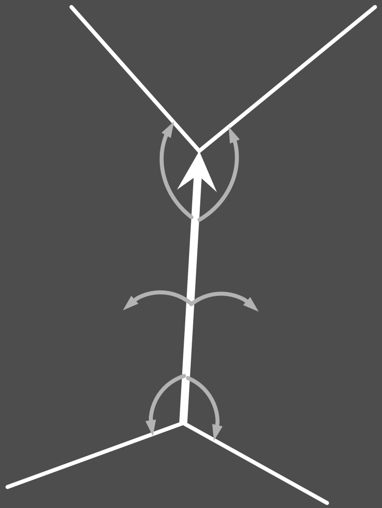




B-Reps: Polygonal Meshes

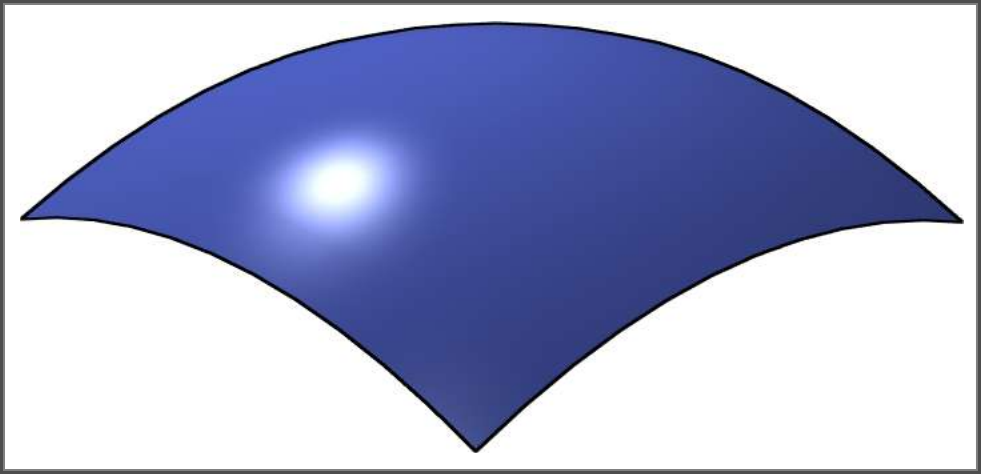
- polygons to define the surface of objects
- triangle meshes
  - polygon with fewest vertices
  - always convex & planar → defines unique surface
- triangle strips: faster rendering
- more complex mesh data structures (e.g., Winged Edge)





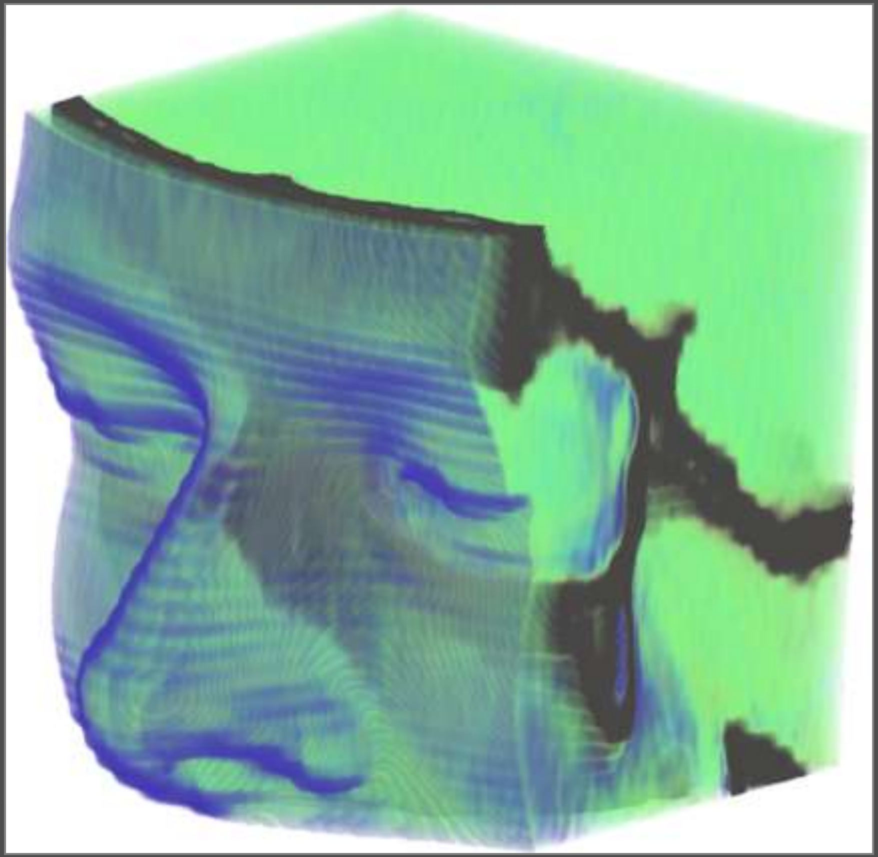
B-Reps: Piecewise Smooth Surfaces

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



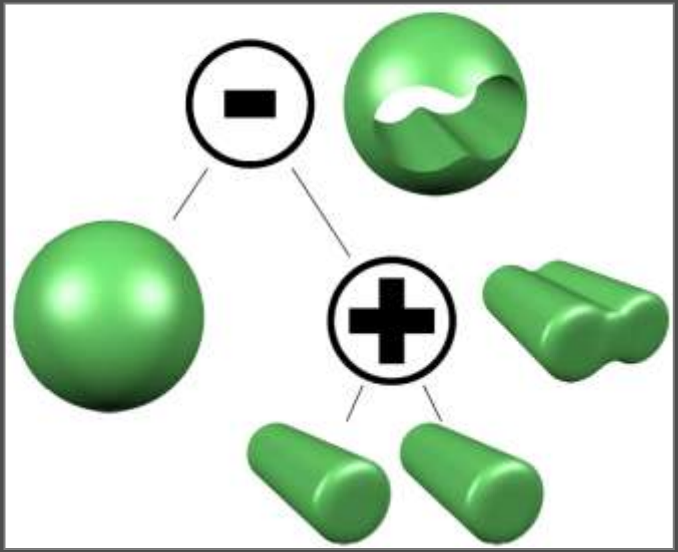
Volumes: Voxel Models

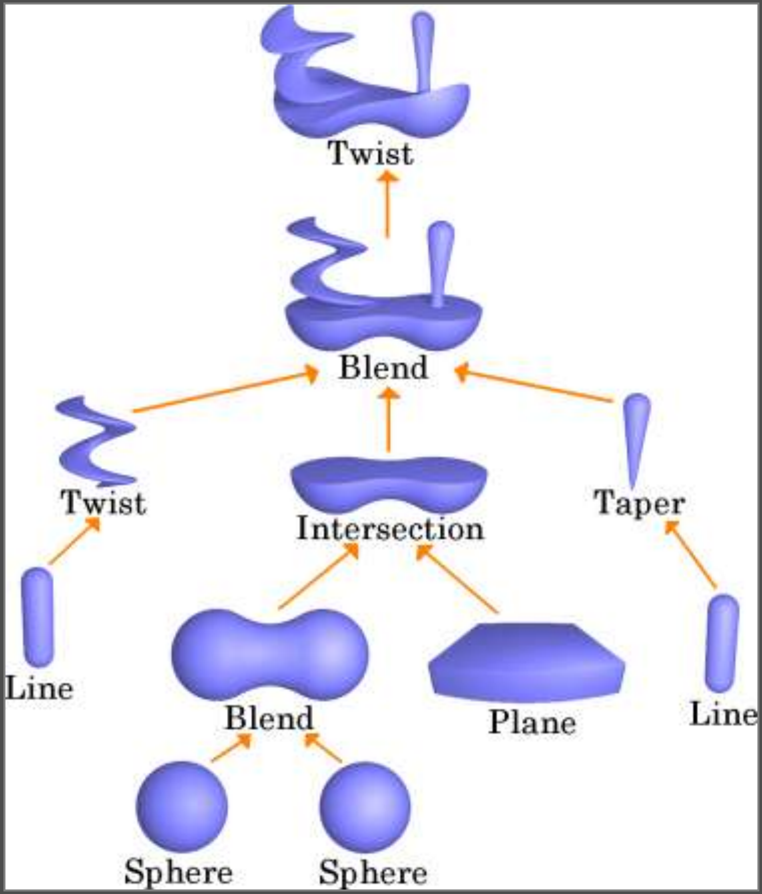
- sampling of a volume in regular intervals
- samples as cubes, or as general boxes
- several properties can be sampled
- shapes: *iso-values* define *iso-surfaces*
- heavily used in medical imaging; based on CT, MRI



Volumes: Construct. Solid Geometry

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

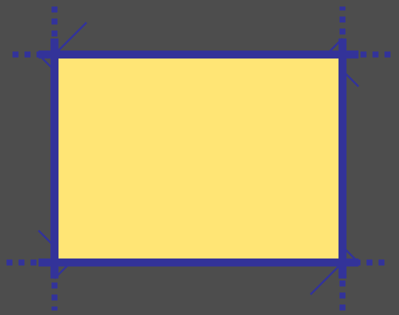




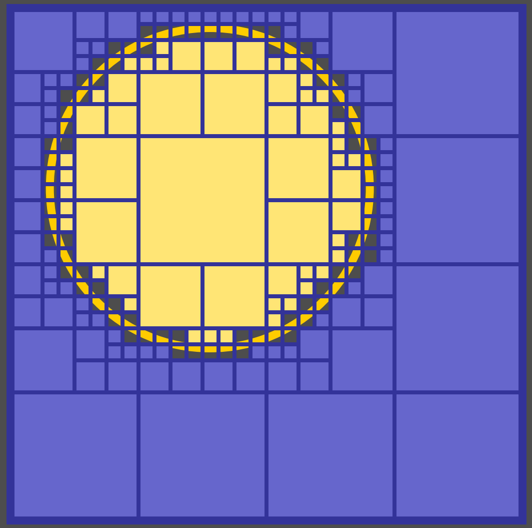
courtesy of
Erwin de Groot
and Brian Wyvill

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

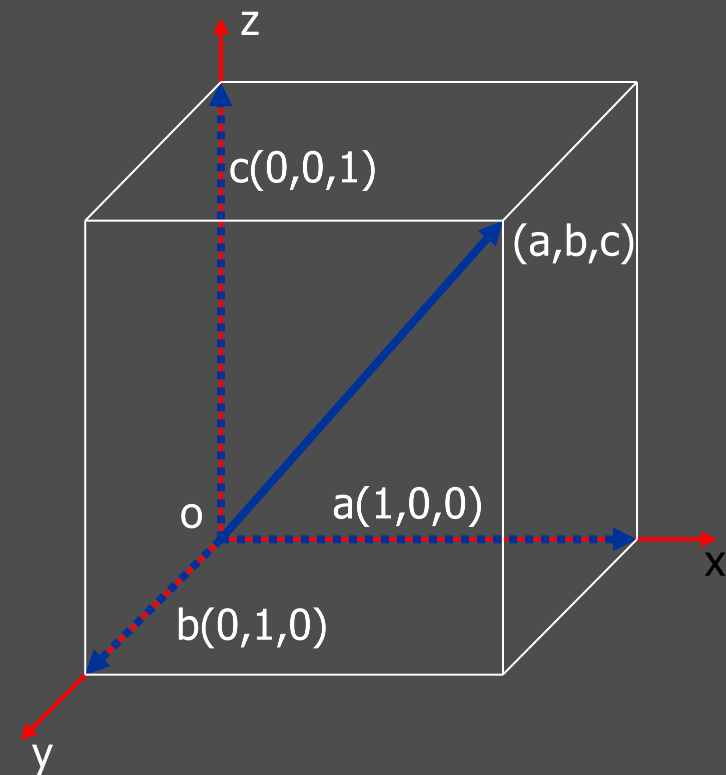
- 3D models represented as points, edges, polygons in 3D coordinate systems
  - object coordinate systems
  - world coordinate system
  - camera coordinate system
  - screen coordinate system
- transformations necessary
- foundation: geometry and linear algebra

Scalars, Points, Vectors, Matrices

- scalar values: real (rational) numbers: 7.39
- points: position in nD space: $(2.53, -1.78)^T$
- vectors: directions in nD space
  - have no position in space
  - is linear combination of basis vectors of the vector space, e.g., $(a, b, c)^T = a(1, 0, 0)^T + b(0, 1, 0)^T + c(0, 0, 1)^T$
- matrices ($n \times m$): transformations between vector spaces

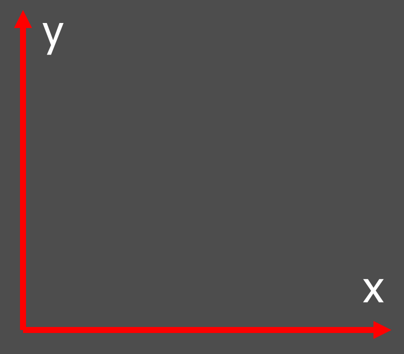
Coordinate System and Coordinates

- coordinate system: set $(o, e_1, e_2, \ldots, e_n)$ consisting of point $o \in A^n$ and basis $(e_1, e_2, \ldots, e_n)$ of vector space A^n
- position vector $v = \overrightarrow{(op)}$ for each $p \in A^n$
- coordinates: scalar components of v with respect to $(e_1, e_2, \ldots, e_n)$ $v = x_1e_1 + x_2e_2 + \ldots + x_ne_n$
- o: point of origin

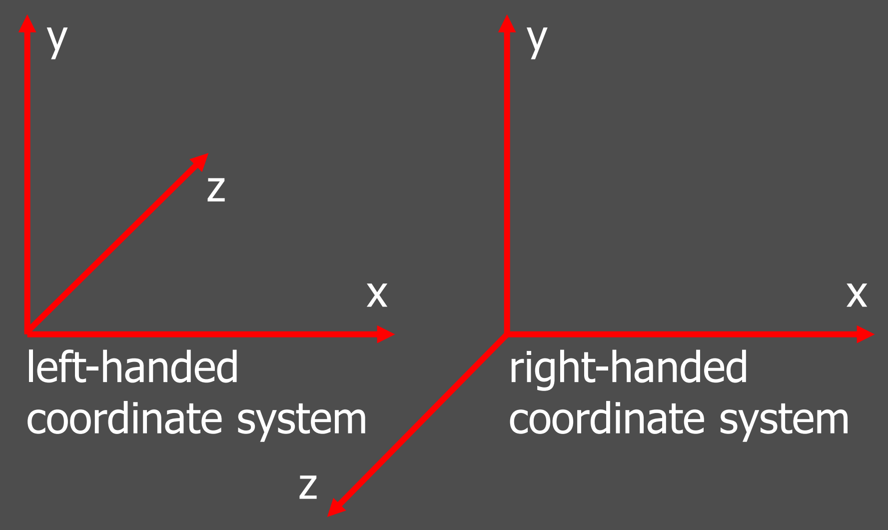


Coordinate Systems in CG: 2D & 3D

- two-dimensional



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

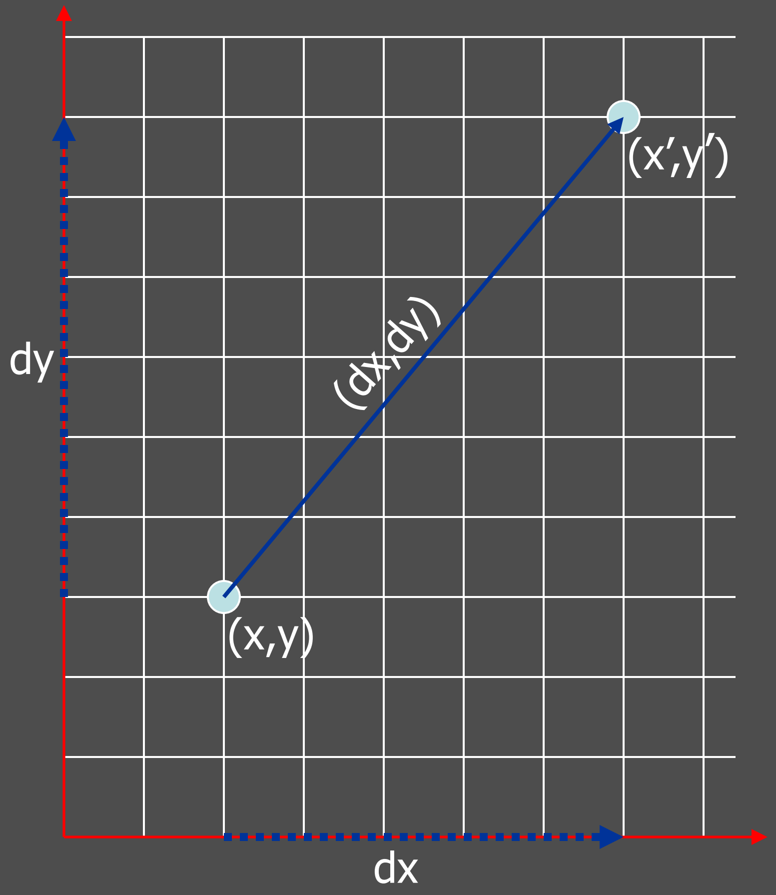


2D Translation

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

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

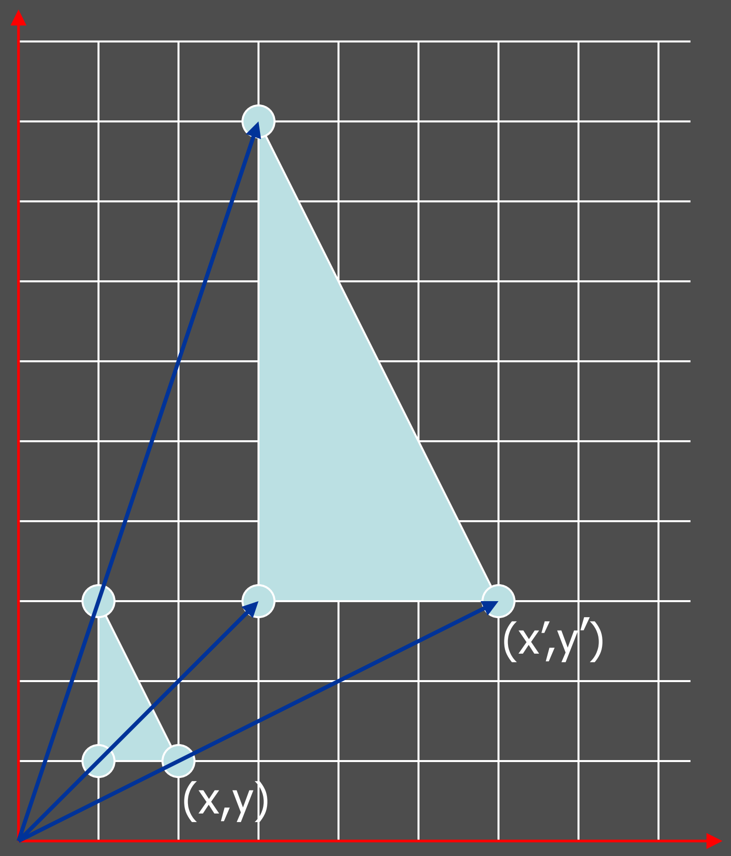
- vector: movement from one point to another



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

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

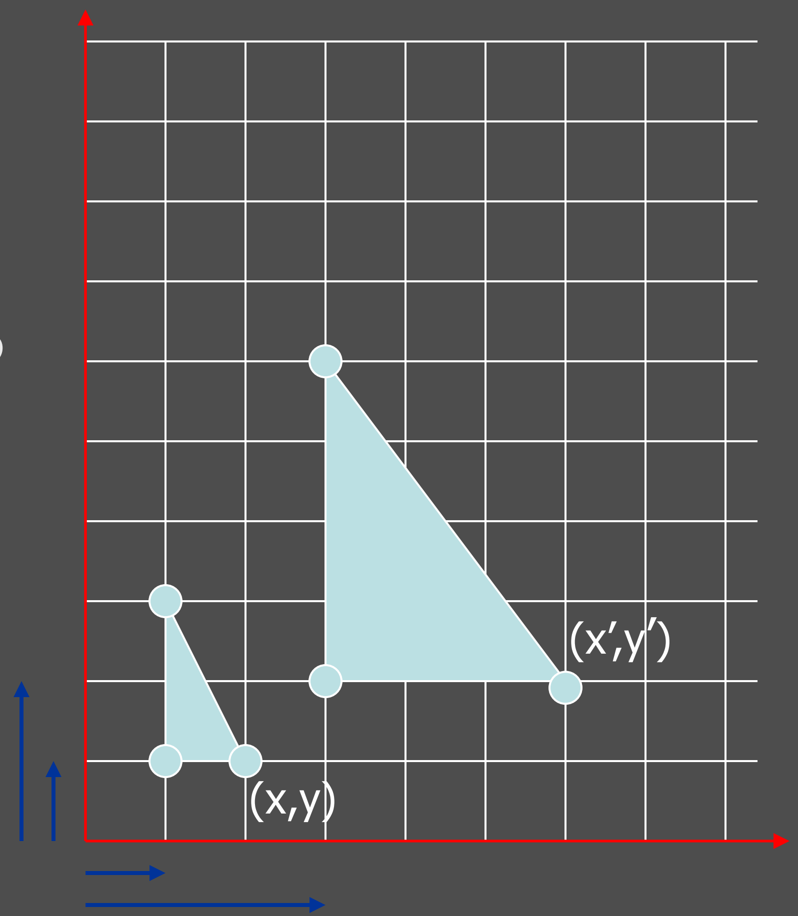


2D Non-Uniform Scaling

- 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

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

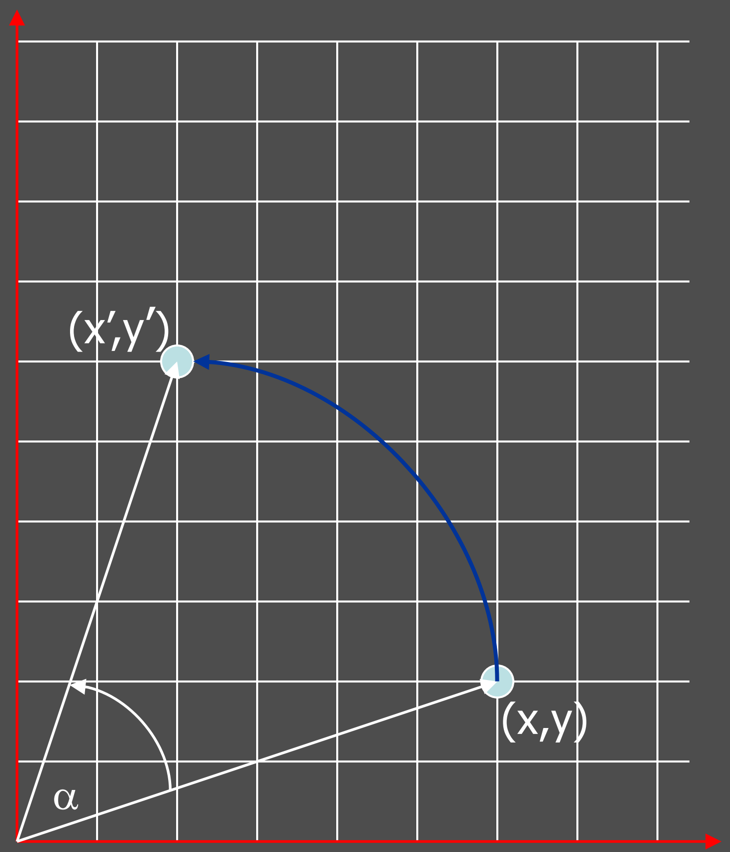
- application: mirroring



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!

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

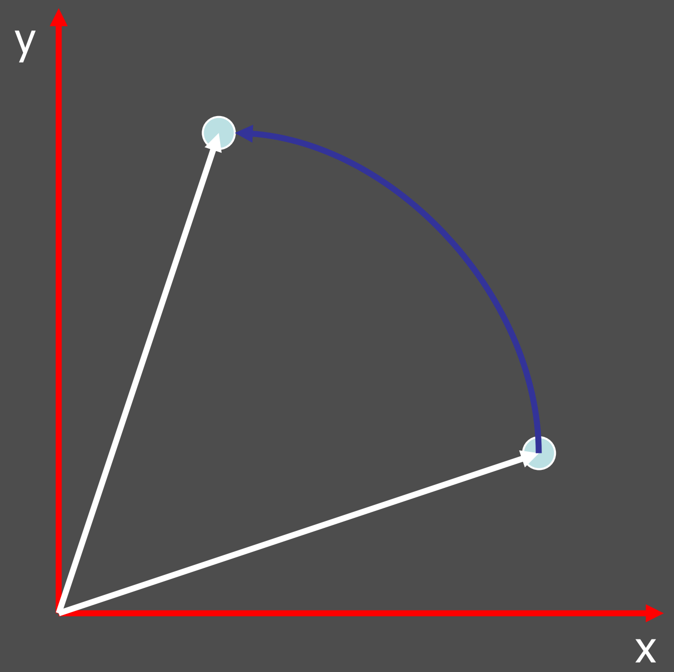
- rotation as matrix multiplication

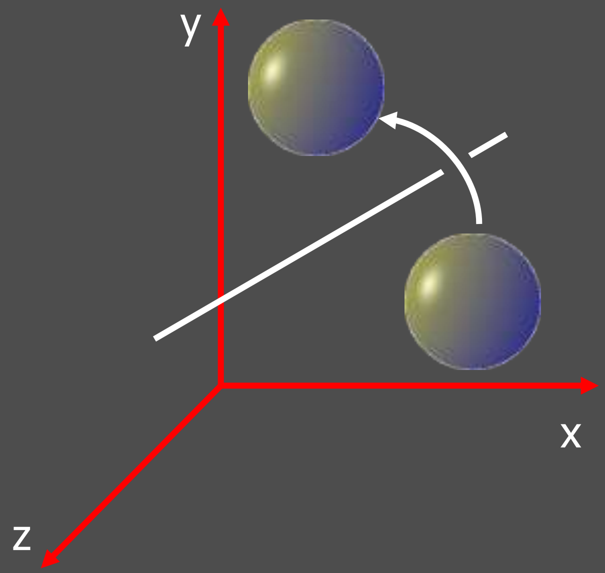
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

- 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

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

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

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

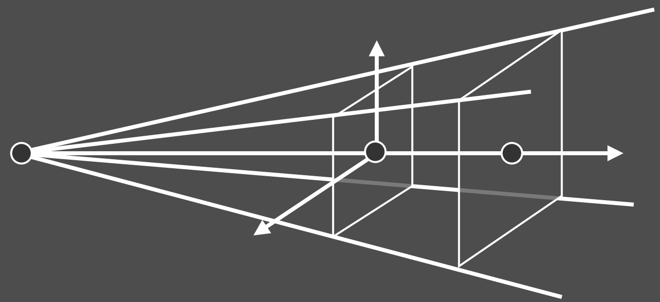
Transformations: Summary

- geometric transformations:
  linear mapping from $\Re^n$ to $\Re^n$
- we are interested in
  $\Re^2 \rightarrow \Re^2$ and $\Re^3 \rightarrow \Re^3$
- transformations most relevant for CG:
  - translation
  - rotation
  - scaling
  - mirroring
  - shearing

Transformations: Summary

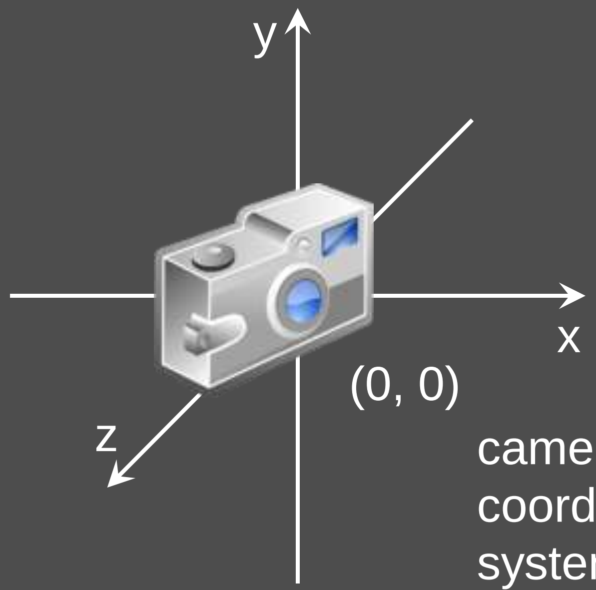
- unified representation of geometric transformations as matrices in homogeneous coordinates
- concatenation of transformation by multiplying the respective matrices
- order matters: for **column** vectors the **first** transformation comes **last** in the sequence
- concatenated transformations can be pre-computed (saving run-time)

Viewing and Projections

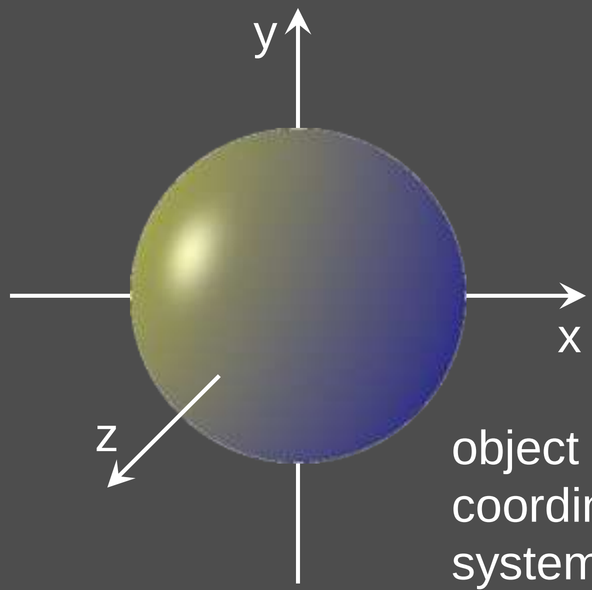


Model-View Transformation

- transformation **directly** into camera coordinates:
  → object-dependent transformations



camera
coordinate
system



object
coordinate
system

- each object has its own model-view matrix
- hierarchies possible, objects reusable

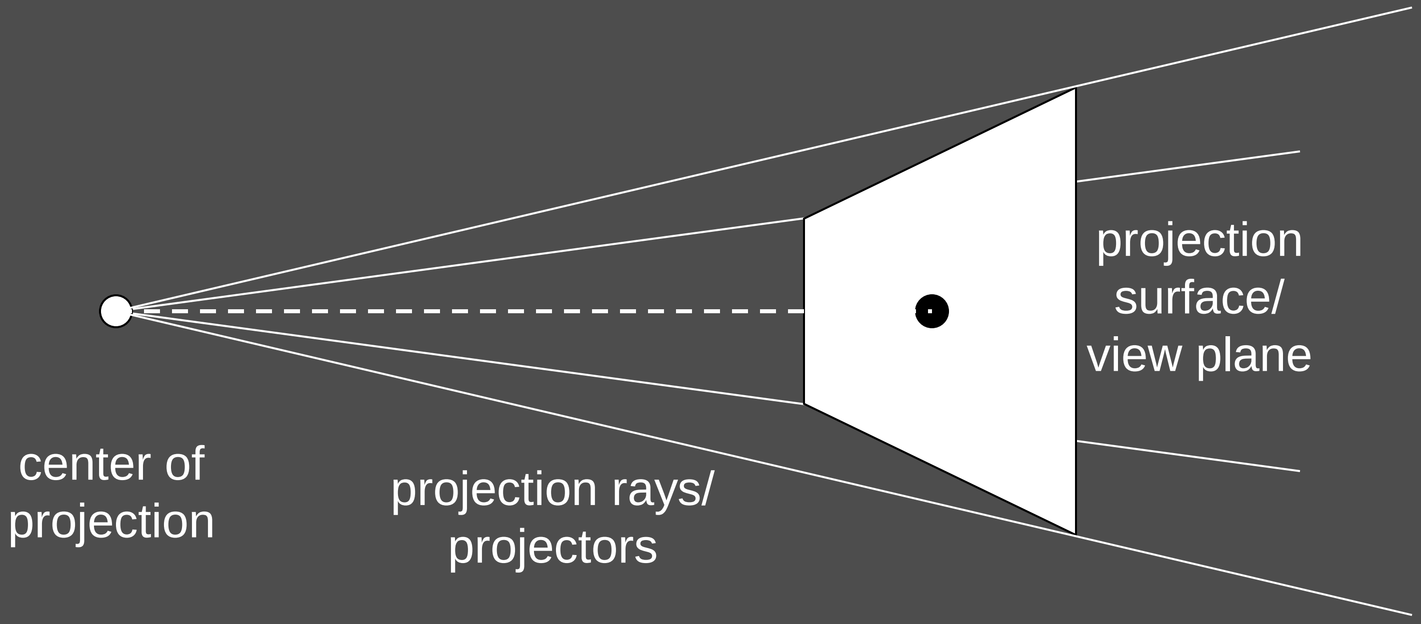
Model-View Transformation

- 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)

Projections

planar projection:

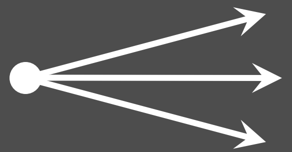
- projection rays are straight lines
- projection surface/view plane is planar
- projections of straight lines are also straight



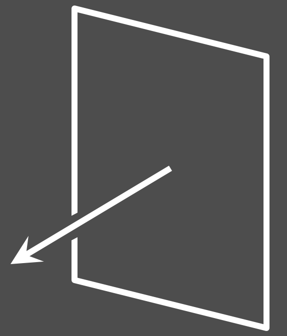
Introduction – Terms

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

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

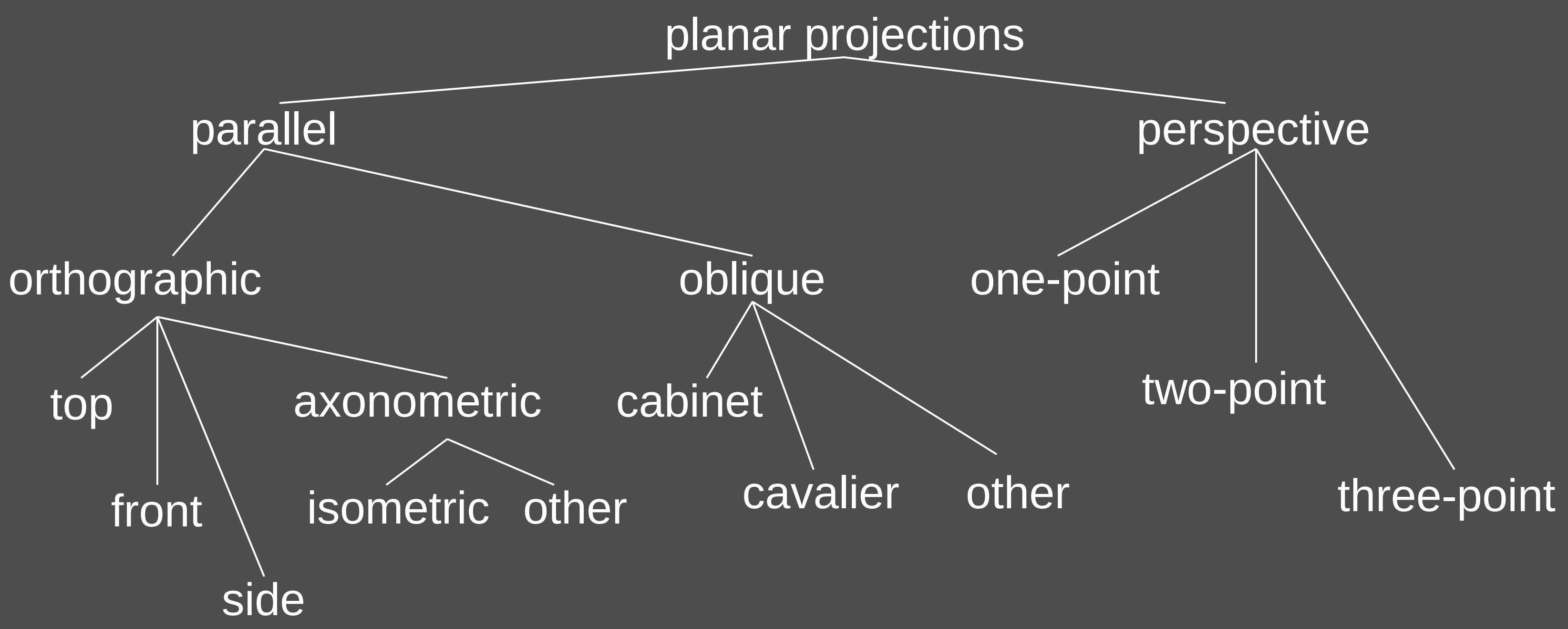


- projection on **view plane**
- vector perpendicular to view plane: **view plane normal** (vpn)



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

Classification of Planar Projections



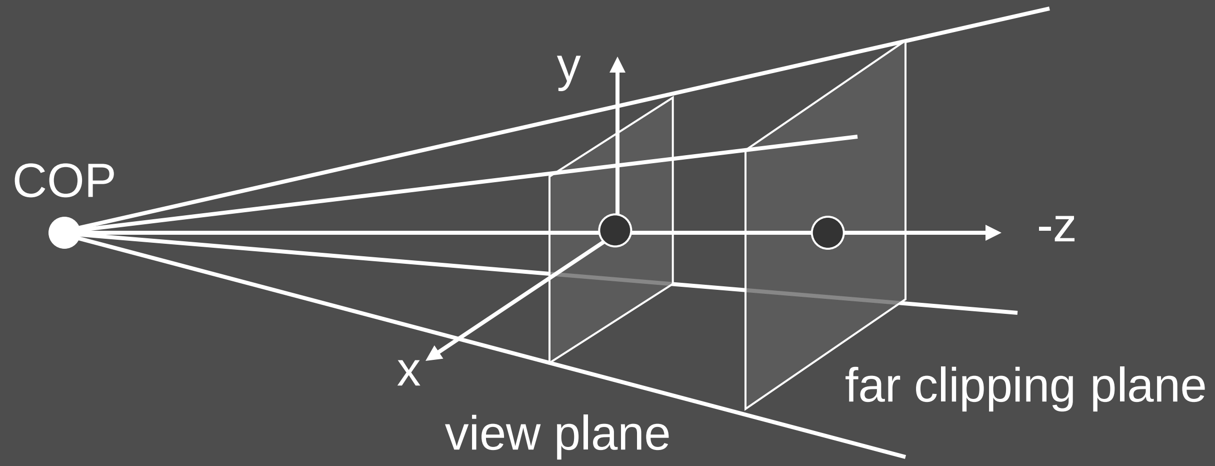
- parallel projections:
  all projectors parallel to each other
- perspective projections:
  projectors diverge from cop

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)

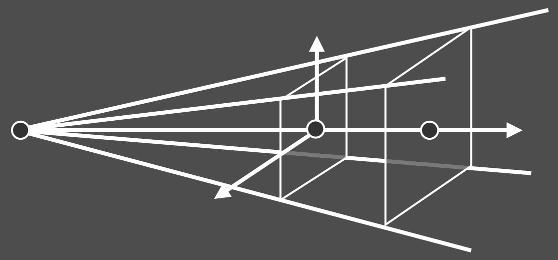
Perspective Projections: Camera Model

- 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)

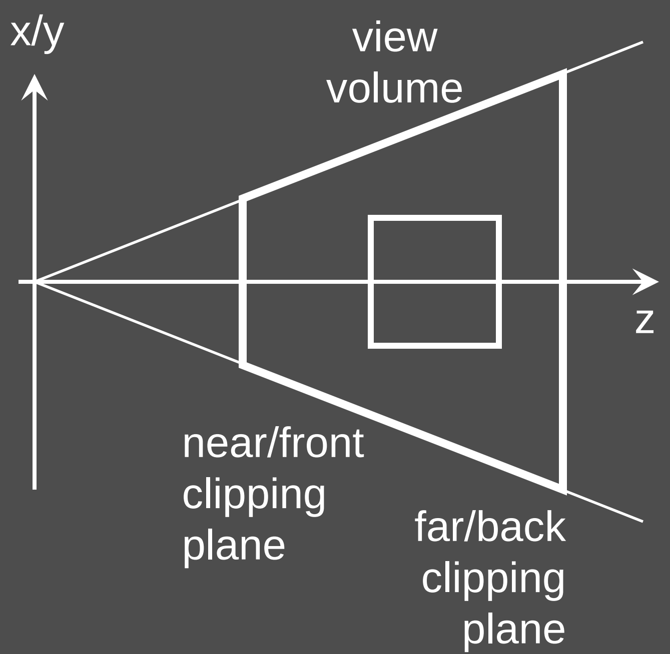


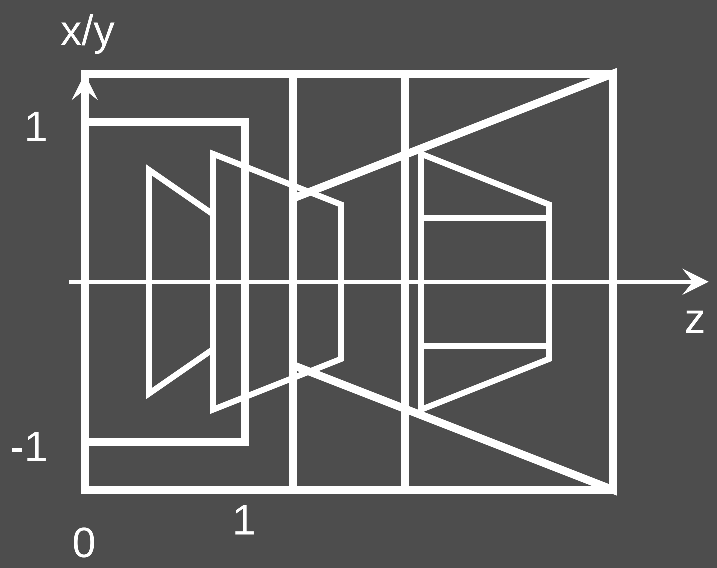
Perspective Projections: Camera Model

- 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



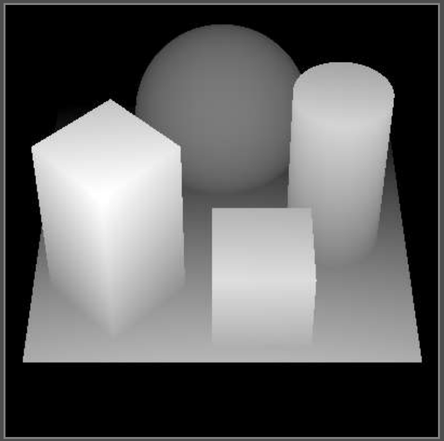
Projections: Canonical View Volumes





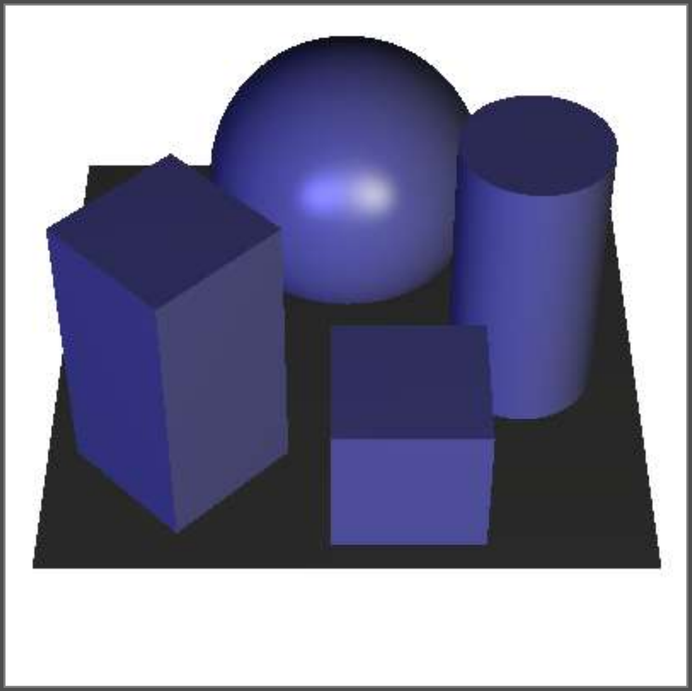
- shearing, translation, scaling
- view volume is only implicitly transformed: included objects are actually transformed!

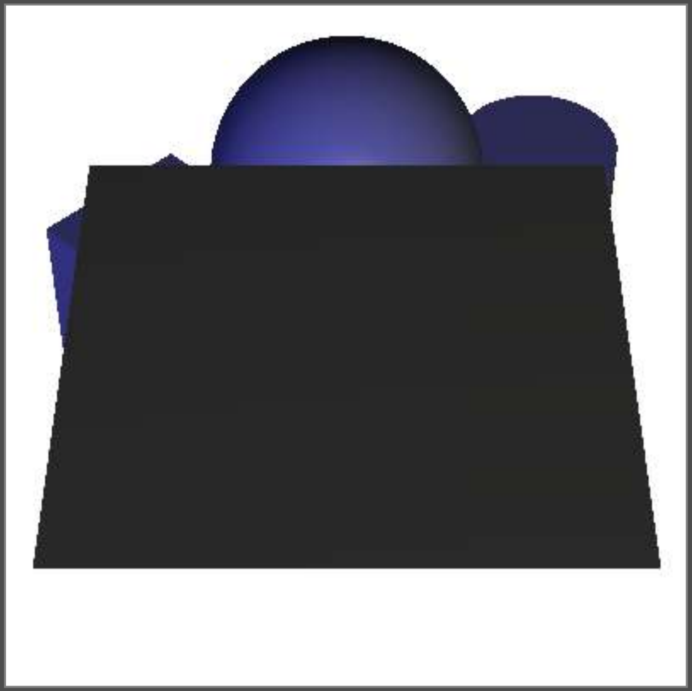
Hidden Surface Removal



Hidden Surface Removal: Motivation

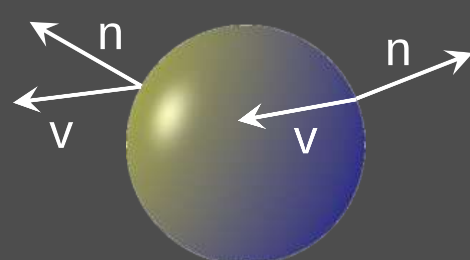
- goals:
  - model parts independently processed
  - at the same time: show front parts only
  - avoid unnecessary processing





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



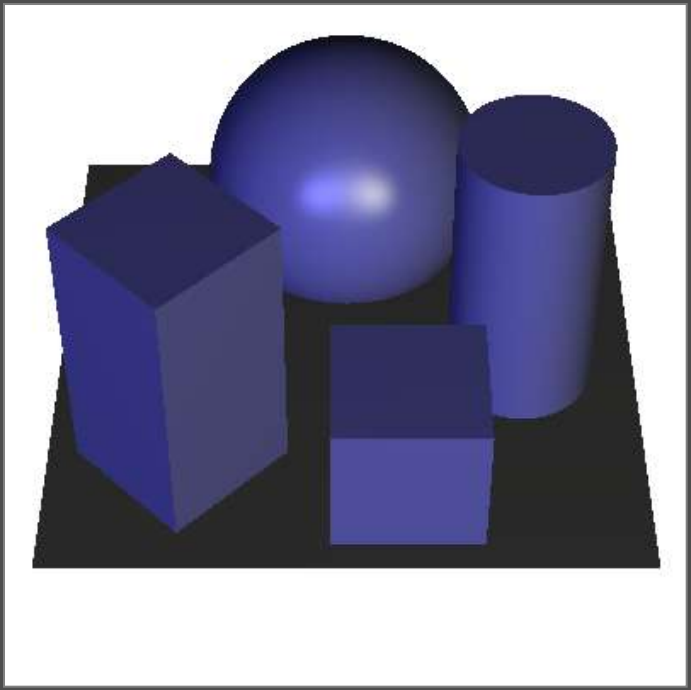
$$n \cdot v \gtrless 0$$

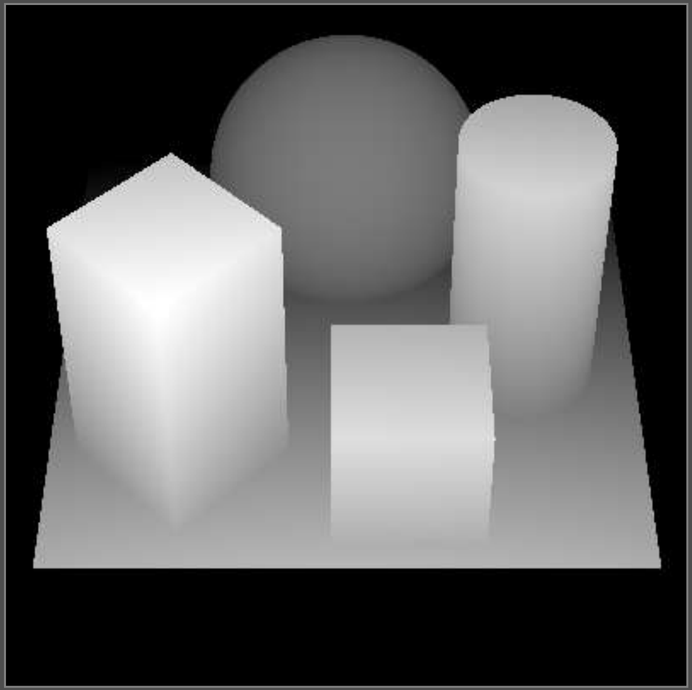
z-Buffering

- 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





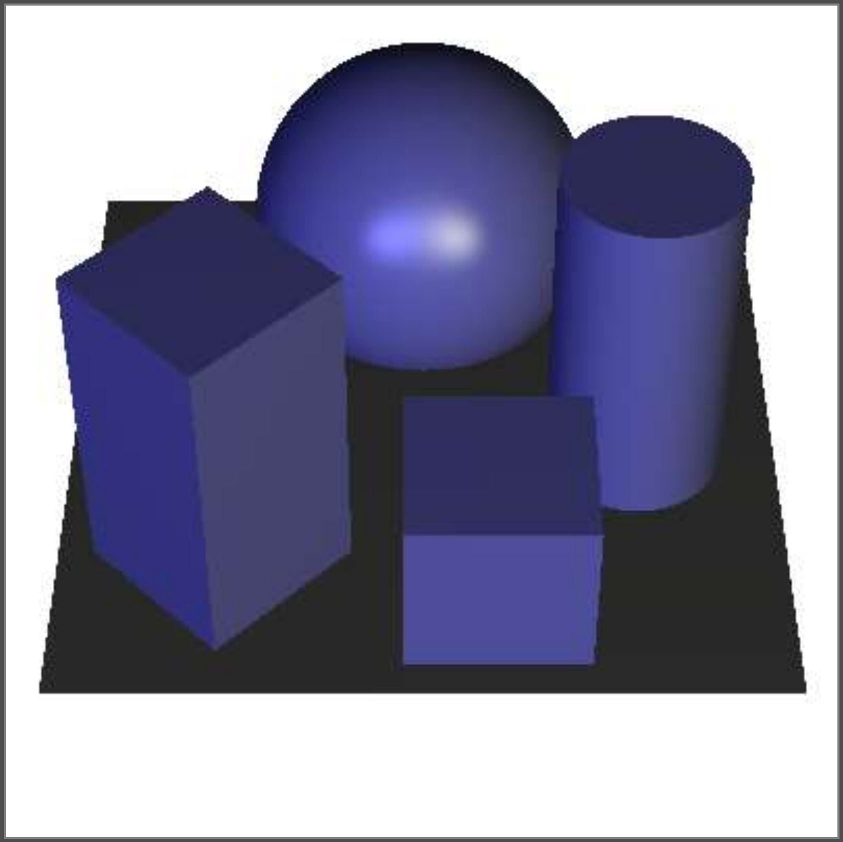
z-Buffering

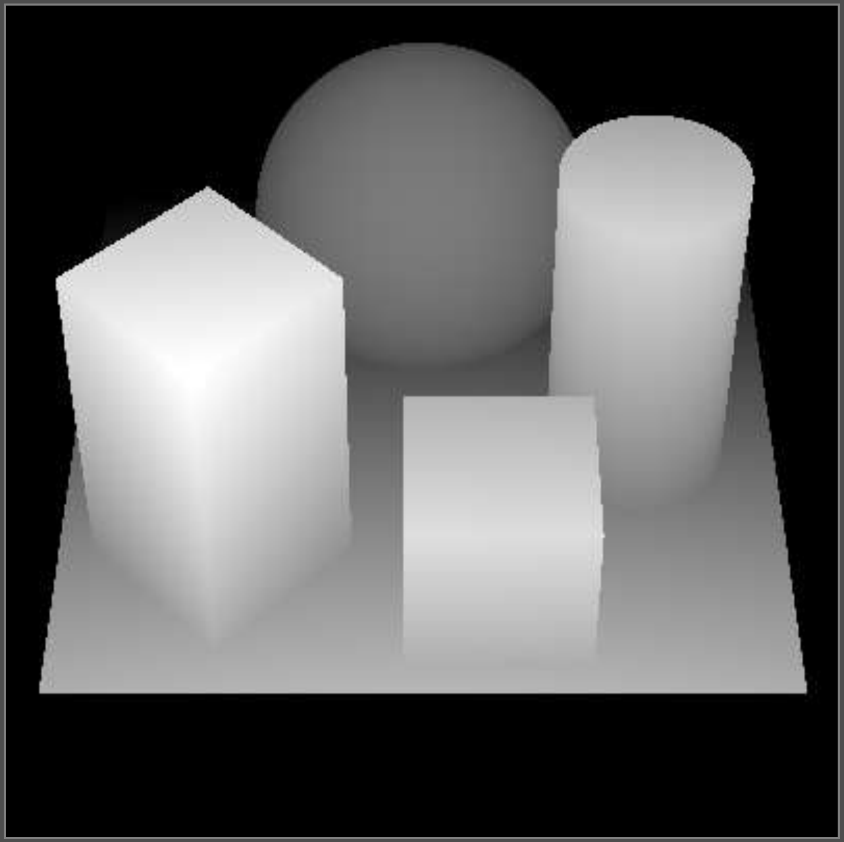
- treat each primitive (triangle) individually: scene $\rightarrow$ objects $\rightarrow$ triangles $\rightarrow$ pixels
- use *z*-buffer data to determine if a new triangle is (partially) hidden or not
- at each time, the part of scene that has been processed thus far is correctly displayed

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)

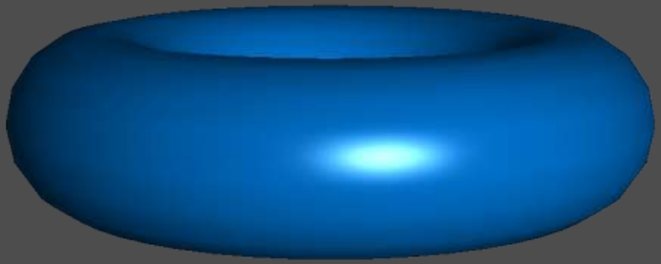




z-Buffering

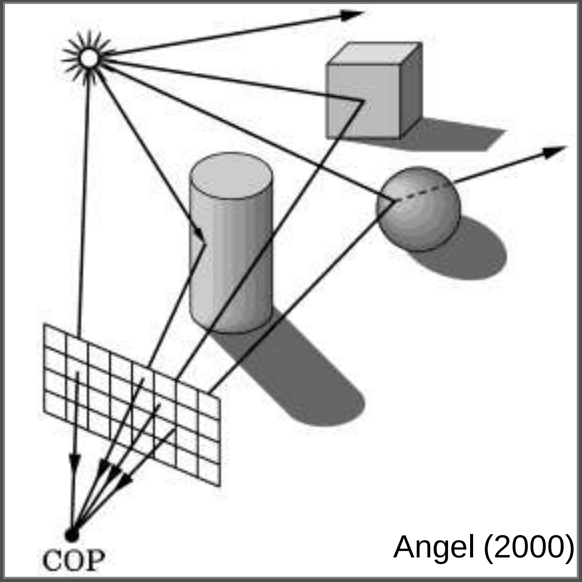
- advantages
  - can be implemented in hardware
  - can process infinitely many primitives
  - does not need sorting of primitives (only need to know distance to camera)
  - can handle cyclic and penetrating triangles
- disadvantages
  - needs memory to keep all *z*-values (image size @ 8bit or 16bit)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

Illumination and Shading



Illumination Models

- real world:
  - surfaces emit, absorb, reflect, and scatter light
  - light intensity and color dependent on surface position and orientation w.r.t. the light source
  - light is usually reflected or refracted several times
  - usually several sources of light
  - final intensity/color at a point is sum of several light paths ending at that point



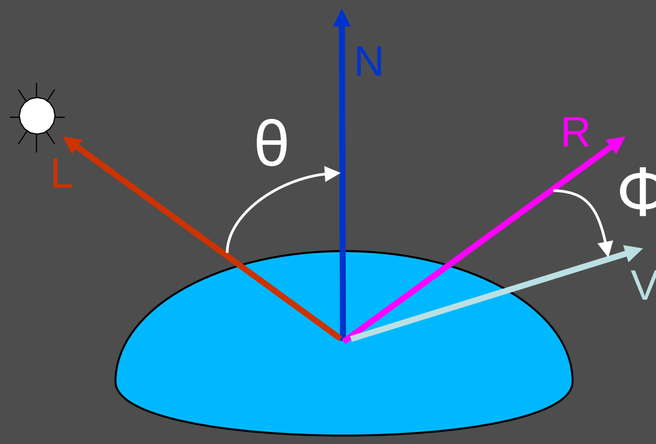
Angel (2000)

Illumination Models

- computer graphics world:
  - mathematical description necessary
  - leads to equation using integrals: the *rendering equation*
  - it’s usually not solvable
  - we need approximation!
    - simplifying light sources
    - simplifying materials
    - simplifying computation
    - speed-up

Light Interaction with Surfaces

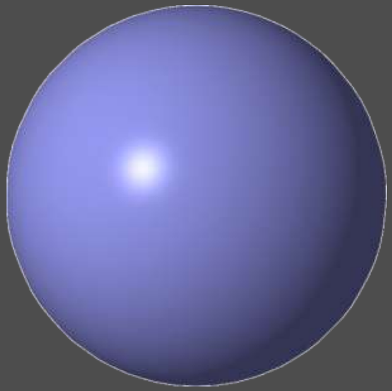
- angle θ between L & N determines diffuse reflection
- reflection angle equals θ
- angle Φ between R & V determines perceived brightness
- maximal reflection if R = V (Φ = 0)

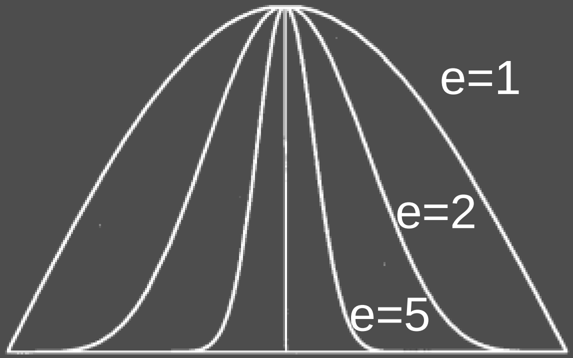


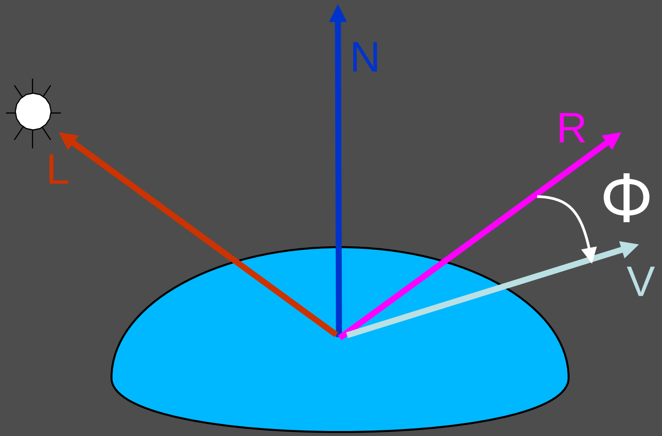
L – vector to light source
N – surface normal vector
R – reflected light ray
V – vector to viewer/observer

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

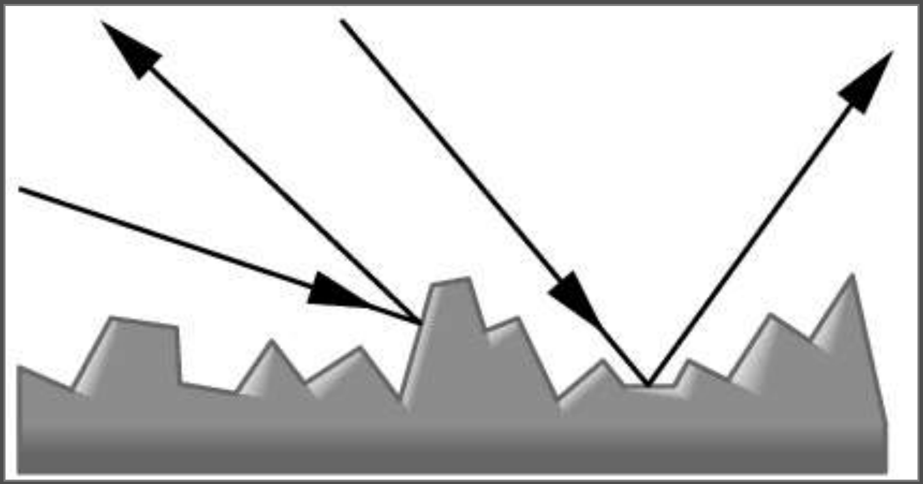






Light Interaction with Surfaces

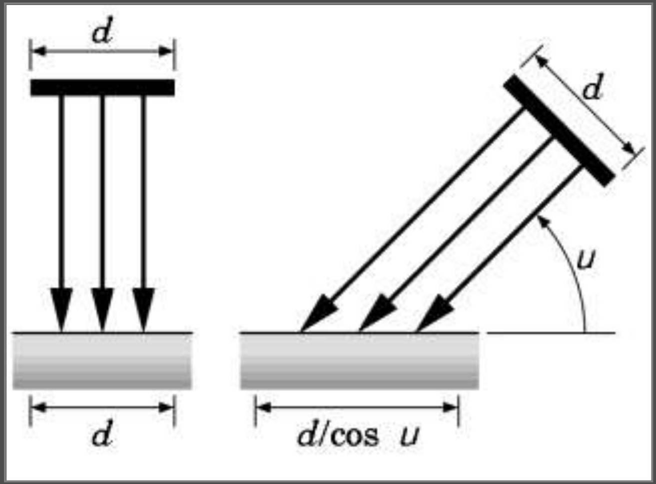
- **diffuse reflection**: equal reflection in all directions on rough surfaces, depends only on θ and not observer – examples?
- due to light scattering on rough surfaces on randomly oriented microscopic facets



Angel (2000)

Light Interaction with Surfaces

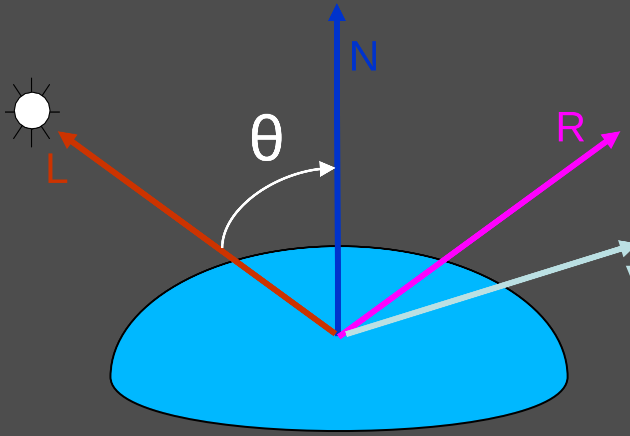
- diffuse reflection modeled according to Lambert's law by cosine function:
  $I \sim \cos\theta = L \bullet N$ for normalized L, N



Angel (2000)

because light distributes over larger area

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

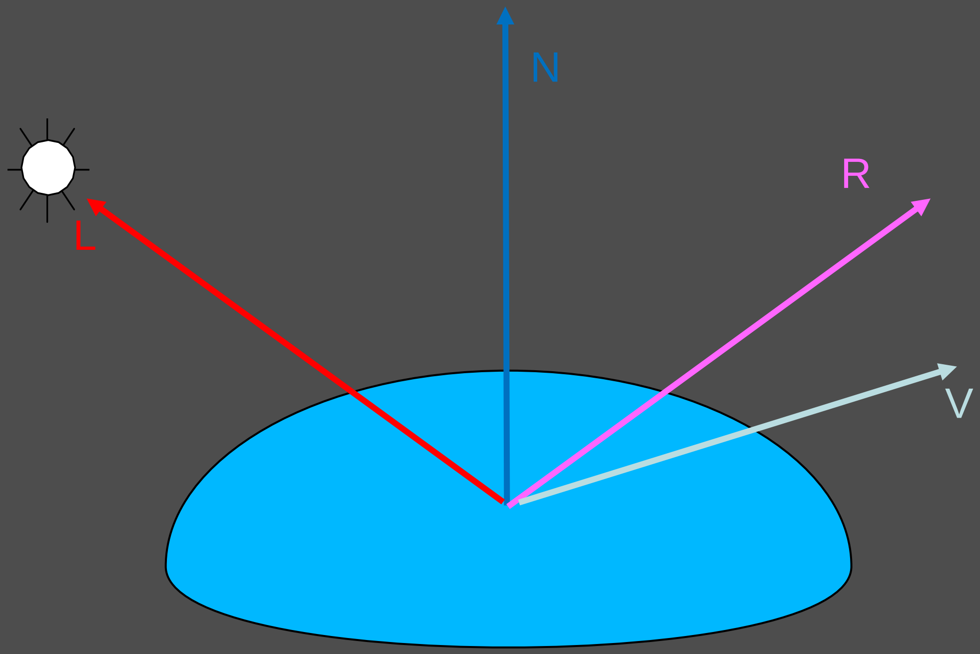


Phong Illumination Model (1973)

- 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

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



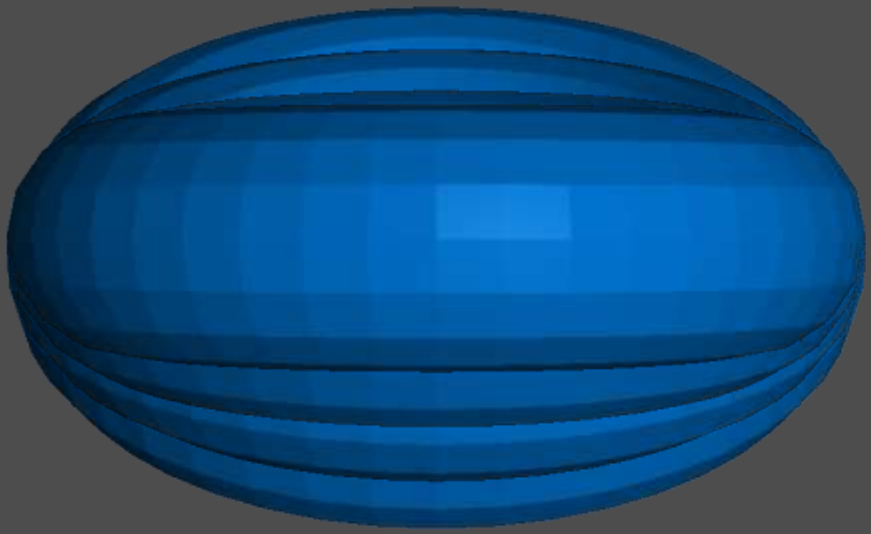
L – vector to light source
N – surface normal vector
R – reflected light ray
V – vector to viewer/observer

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

Shading of Polygonal Models

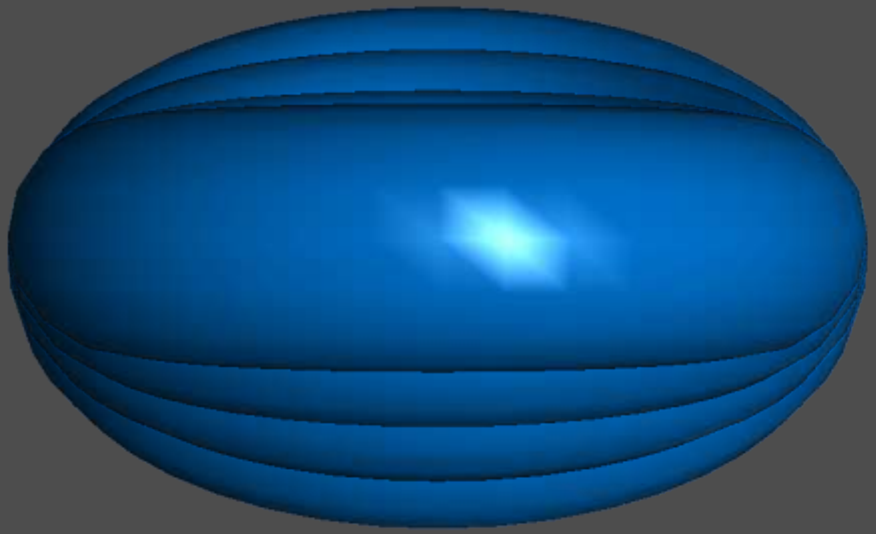
- *status:* we can approximate color at a point
- *goal:* we want to render the whole model
- *constraint:* efficiency and quality
- *approach:* **shade** all pixels of a triangle based on color computation at a few points
- *three techniques:*
  - flat shading
  - Gouraud shading
  - Phong shading (≠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

Flat Shading



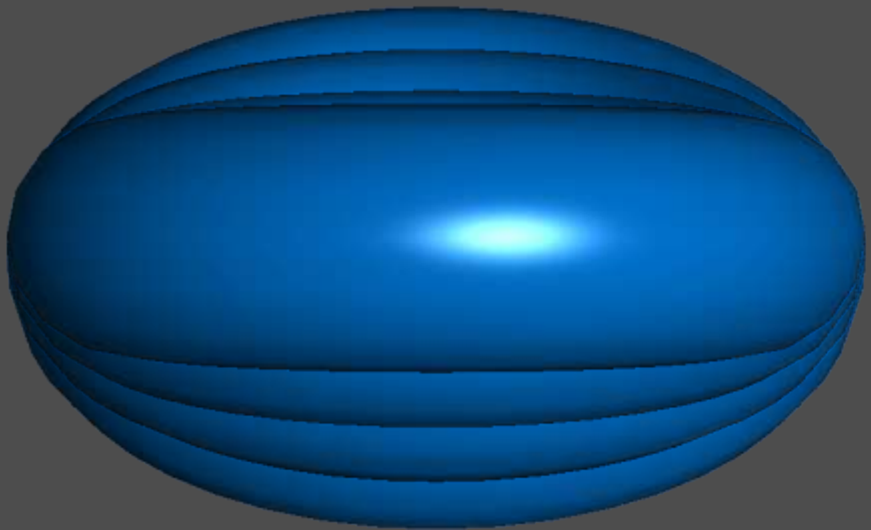
- no interpolation
- all pixels same color
- two methods:
  - one point per triangle/quad
  - average of triangle's/quad's vertices
- low quality: single primitives easily visible
- fast computation & easy implementation

Gouraud Shading (1971)



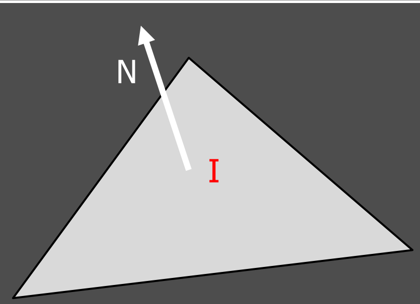
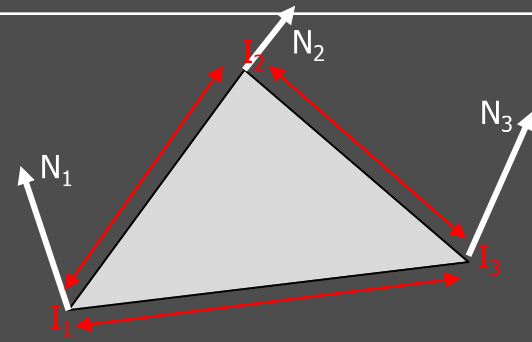
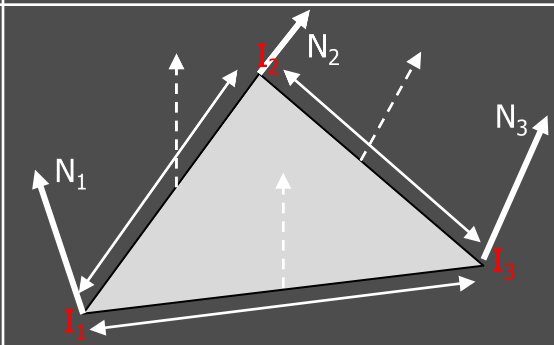
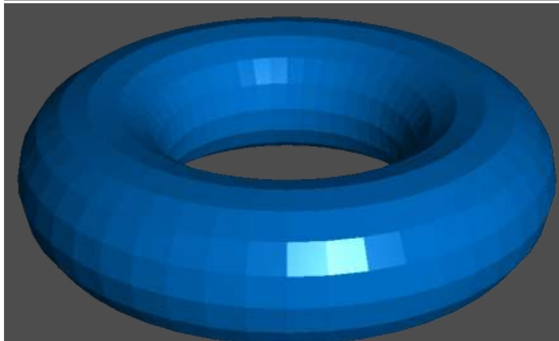
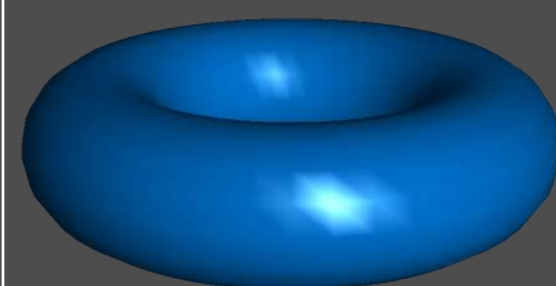
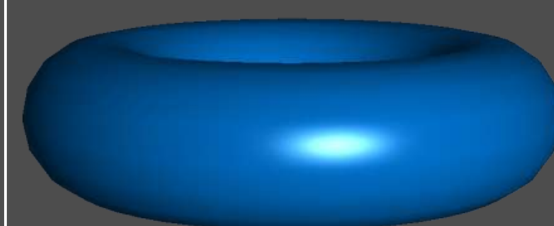
- computation of colors at all vertices
- linear interpolation of colors over primitive
- more computation but better quality than flat shading
- usually implemented in graphics hardware
- highlights problematic: highlight shapes and highlights in the middle of triangles

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

Flat	Gouraud	Phong
		
one color value for entire polygon	one color value per vertex & interpolation inside the triangle	vertex normals interpolated and one color value per pixel
		

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)

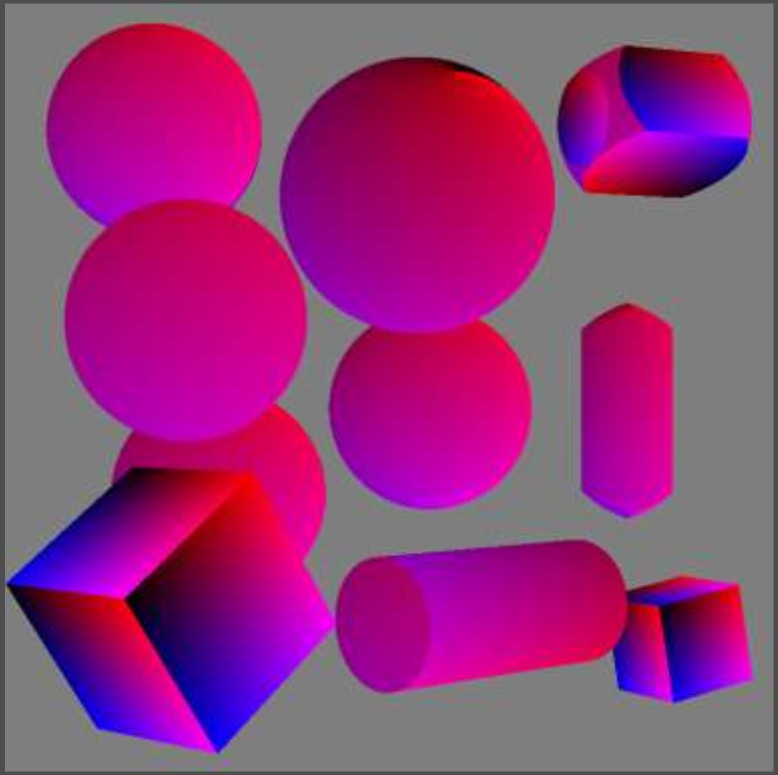
Texture Mapping Motivation

- 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)

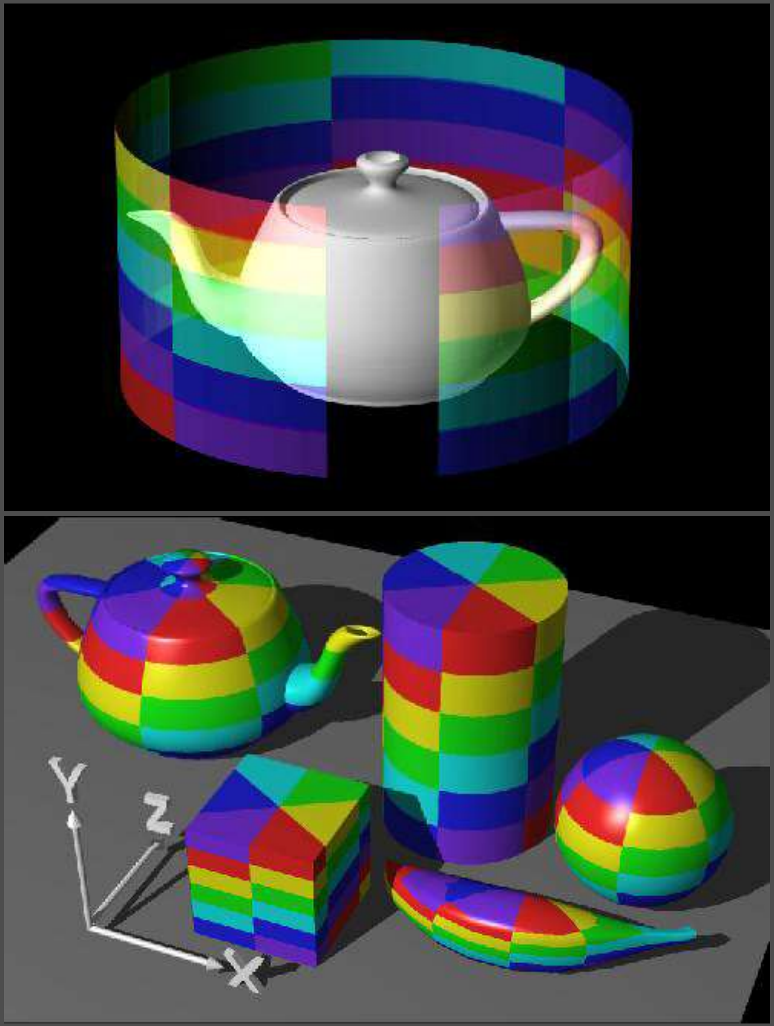
- obvious (*u*, *v*) coordinates derived for primitive shapes (e.g., boxes, spheres, cones, cylinders, etc.)



2-Step Cylindrical Mapping

- mapping onto cylinder surface given by height h_0 and angle θ_0

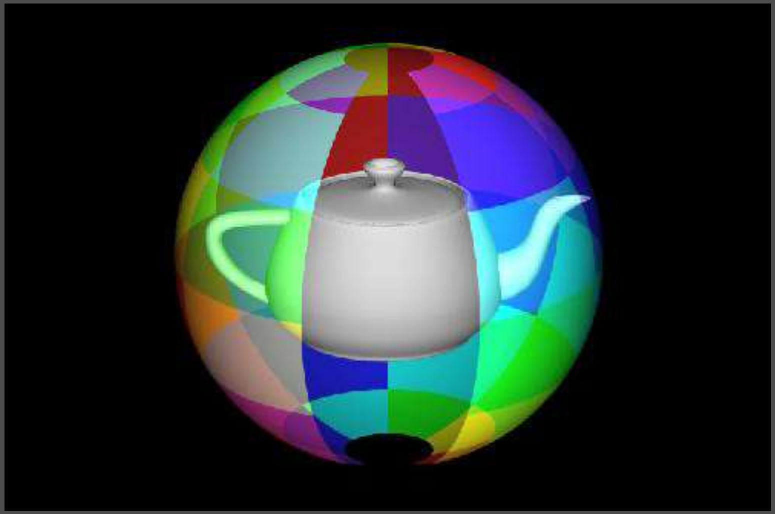
- discontinuity along one line parallel to center axis

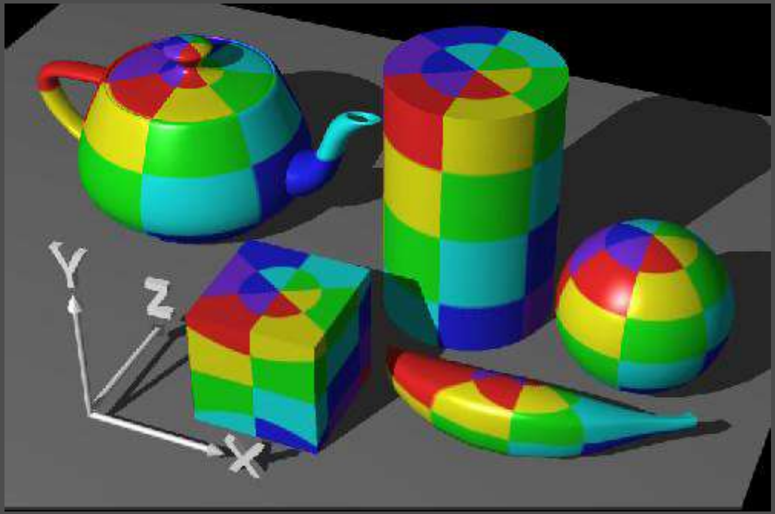


from R. Wolfe: *Teaching Texture Mapping*

2-Step Spherical Mapping

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

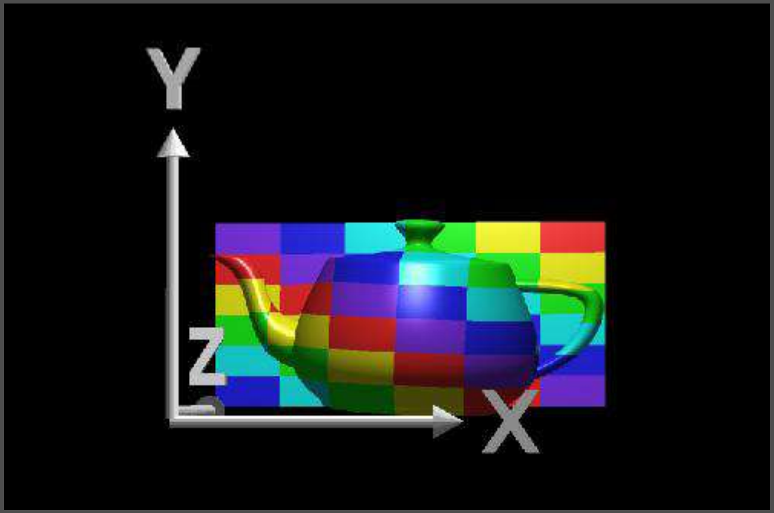


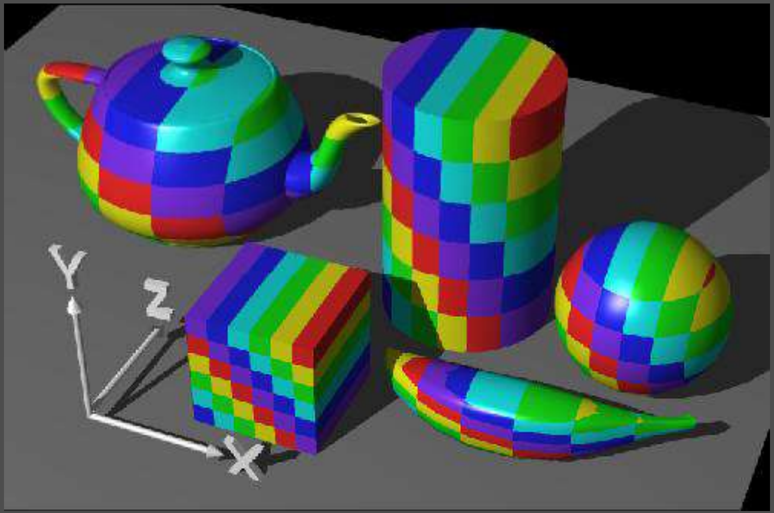


from R. Wolfe: *Teaching Texture Mapping*

2-Step Planar Mapping

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

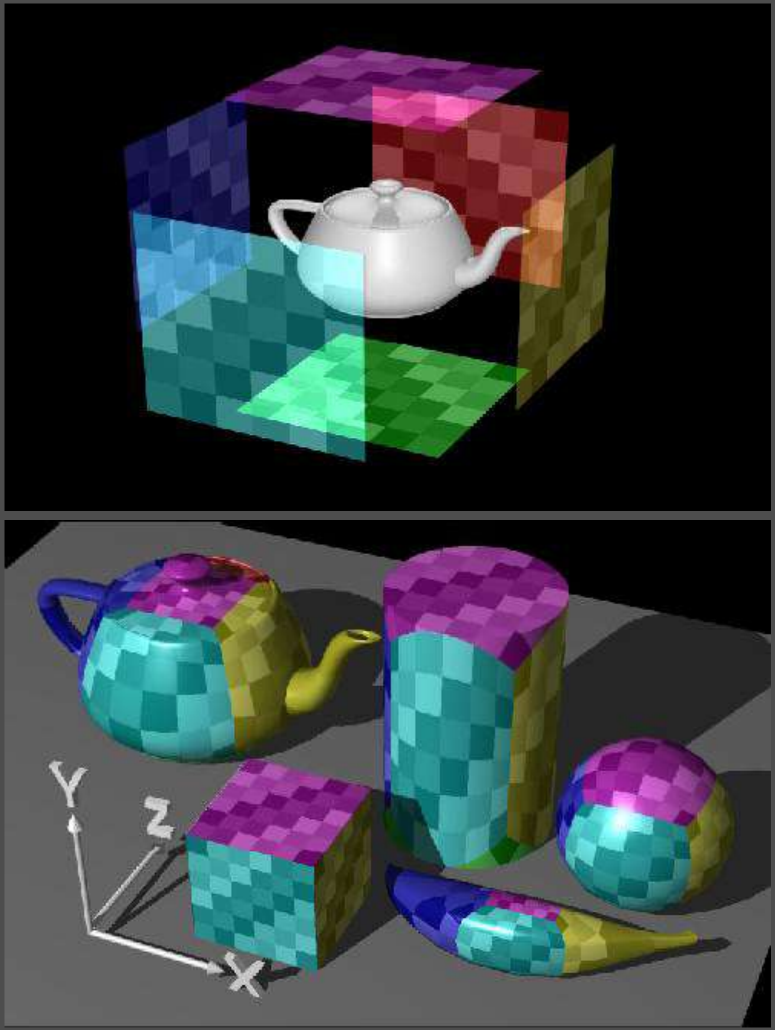




from R. Wolfe: *Teaching Texture Mapping*

2-Step Box Mapping

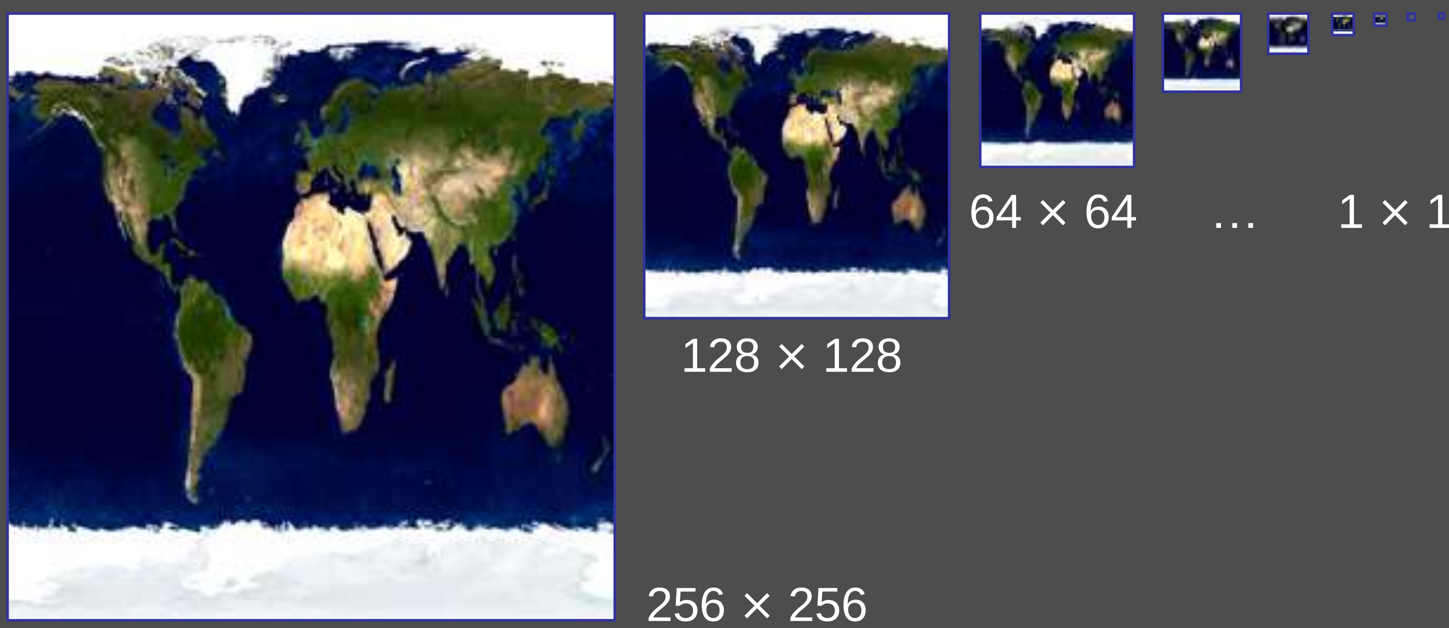
- enclosing box is usually axis-parallel bounding box of object
- six rectangles onto which the texture is mapped
- similar to planar mapping



from R. Wolfe: *Teaching Texture Mapping*

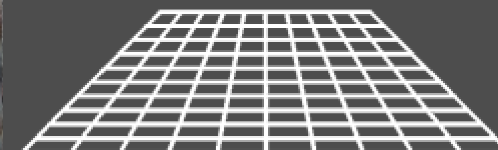
Texture Mapping: Mip Mapping

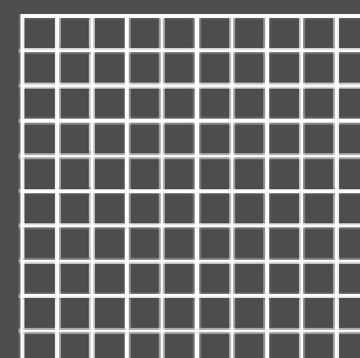
- optimal texture mapping (speed & quality): texel size ≈ pixel size
- idea: use stack of textures and select the most appropriate one w.r.t. situation



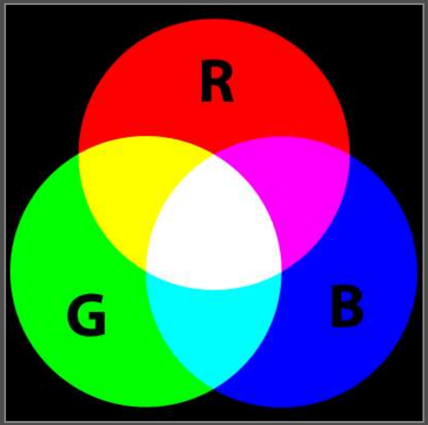
Texture Mapping: Anisotropic Filtering

- 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?

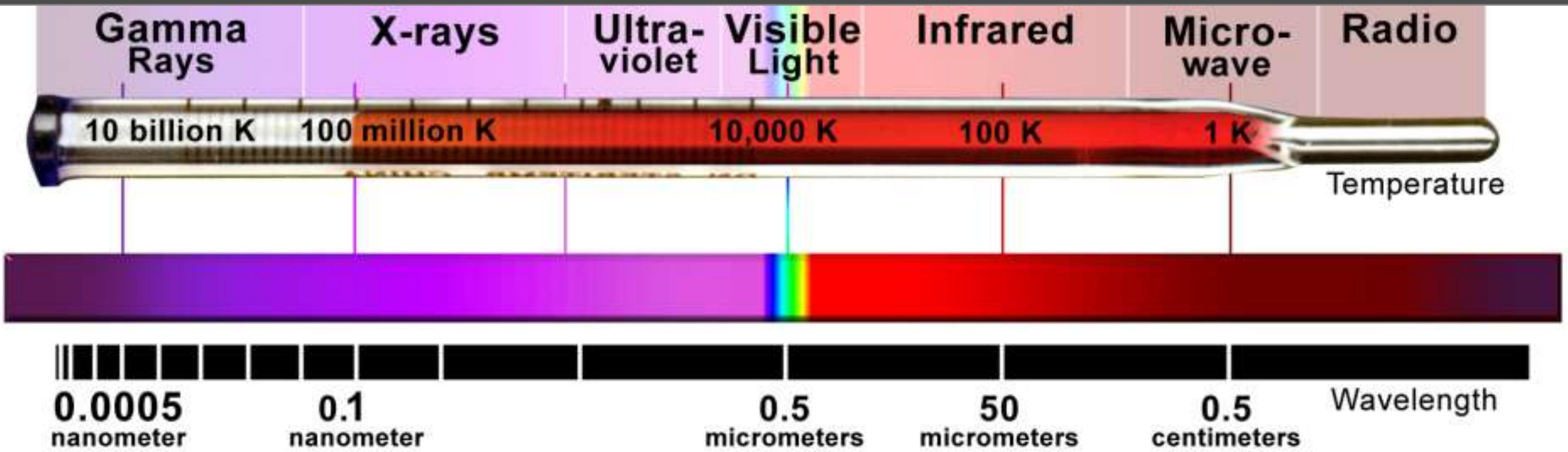
- let's find out …

What is color?

What is Color?

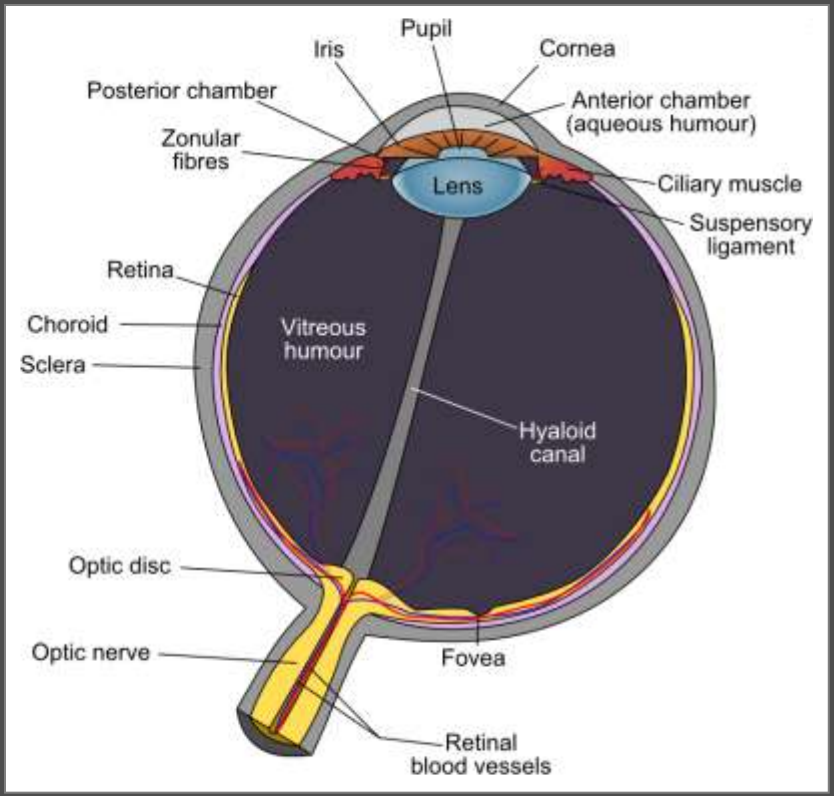
- color is a **human reaction** to light (change)
  (can also be influenced by cultural background)
- what is light?
- light is the visible part (370–730nm) of the electromagnetic spectrum

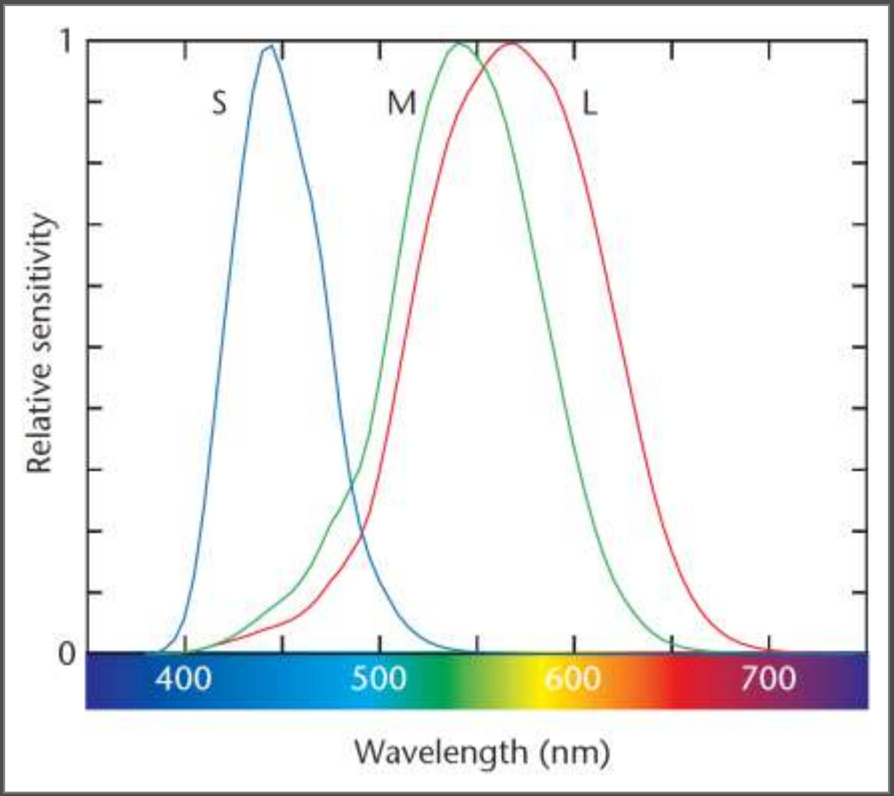
from NASA



Human Color Perception

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

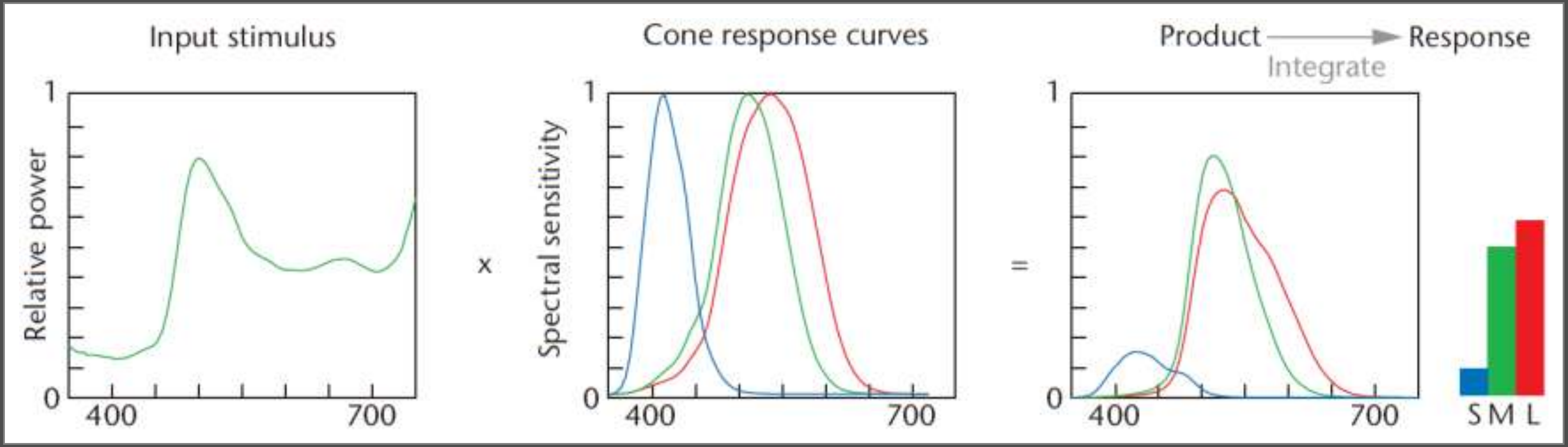




Stone 2005

Human Color Perception

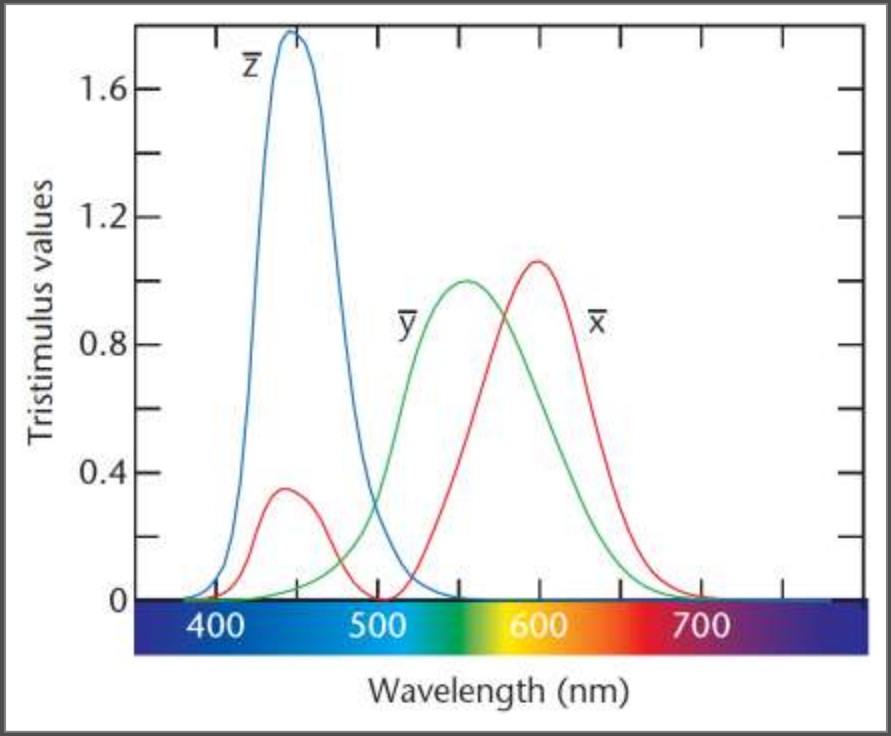
- colored light = spectral distribution function: light intensity as function of wavelength
- converted into 3 response values by cones (short, medium, and long wavelengths)



Stone 2005

Describing Color Vision: XYZ Color Model

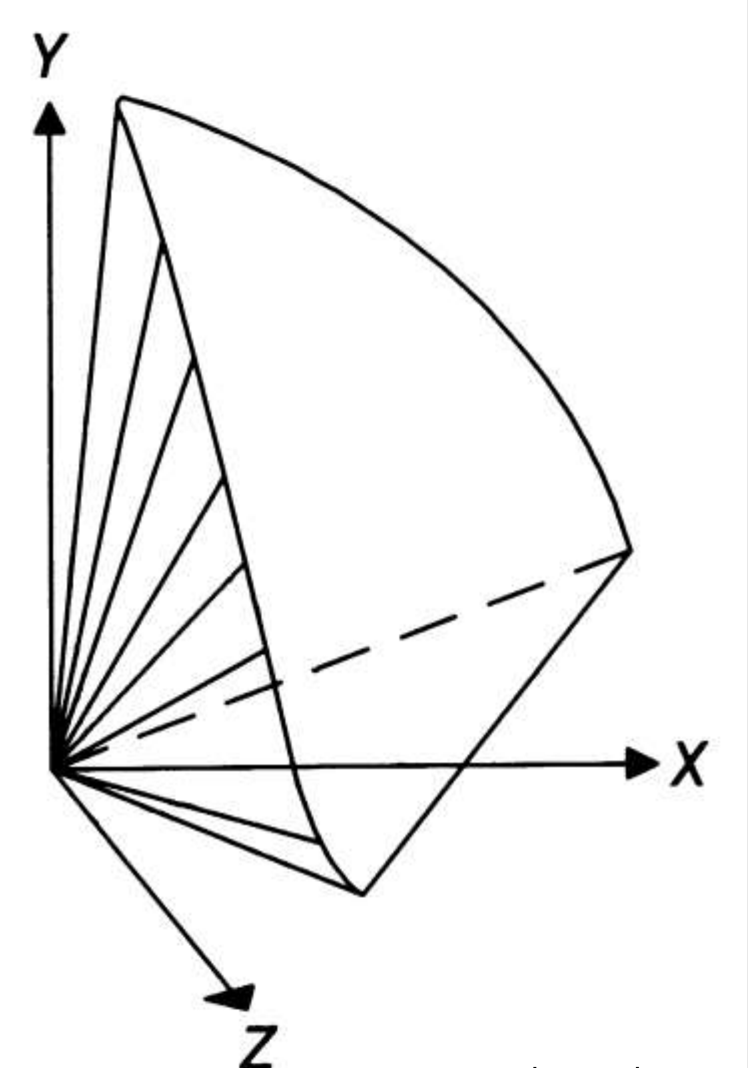
- definition of three primary colors: X, Y, Z
  - color-matching functions are non-negative
  - Y follows the standard human response to luminance, i.e., the Y value represents perceived brightness
  - can represent all perceivable colors
- mathematically derived from experiments



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

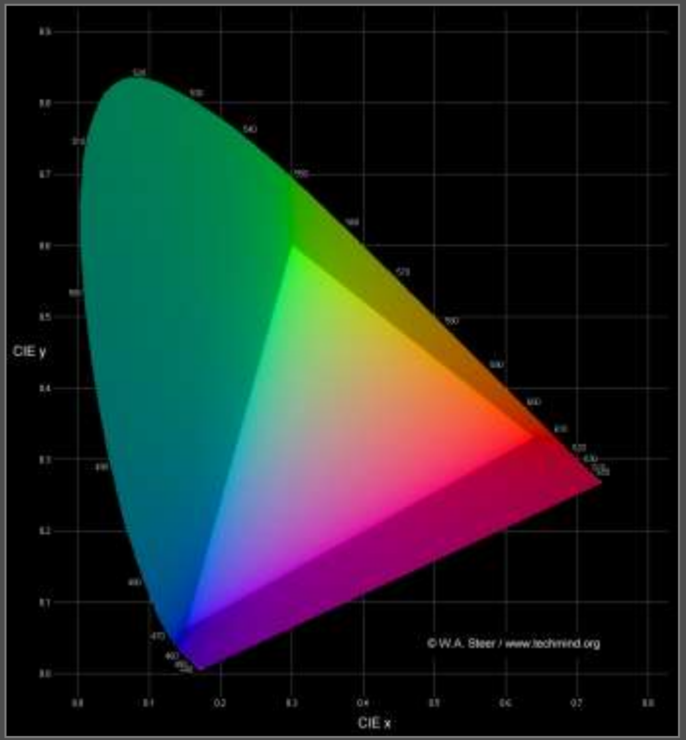


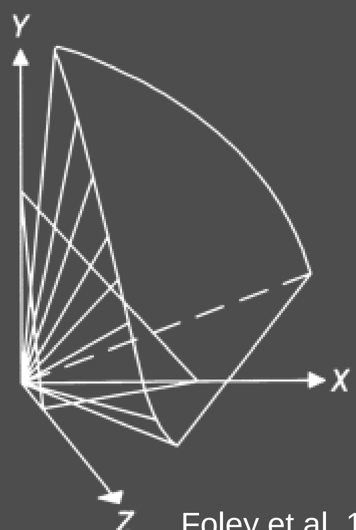
Foley et al. 1990

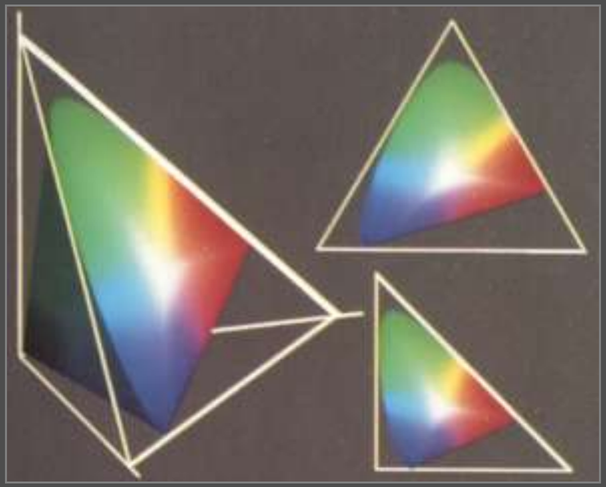
CIE Chromaticity Diagram

- 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

http://www.techmind.org/



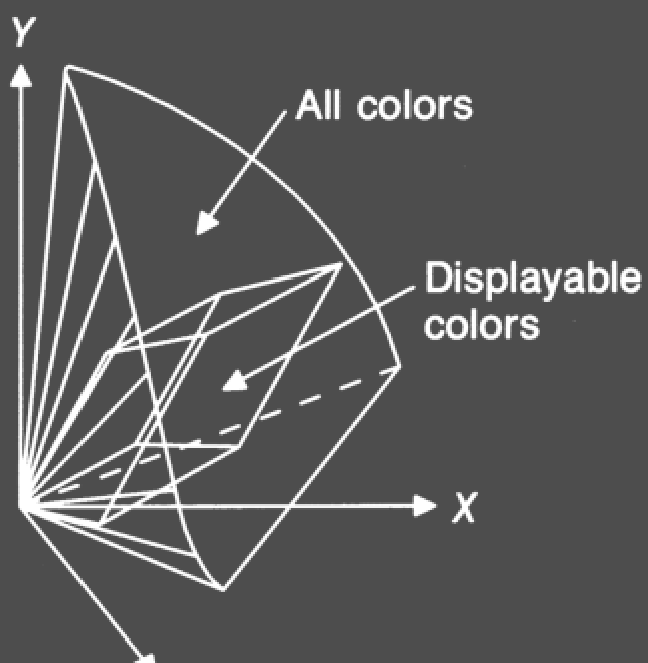




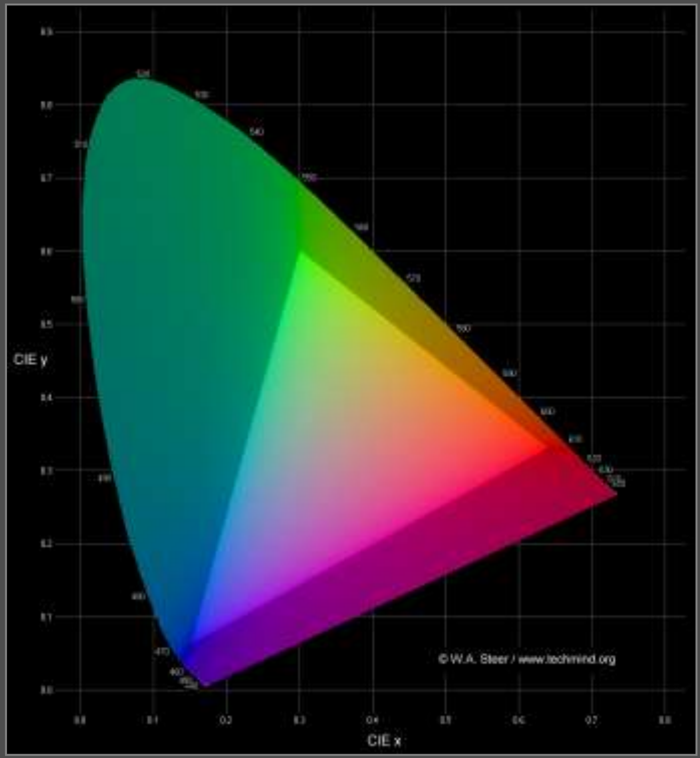
Foley et al. 1990

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

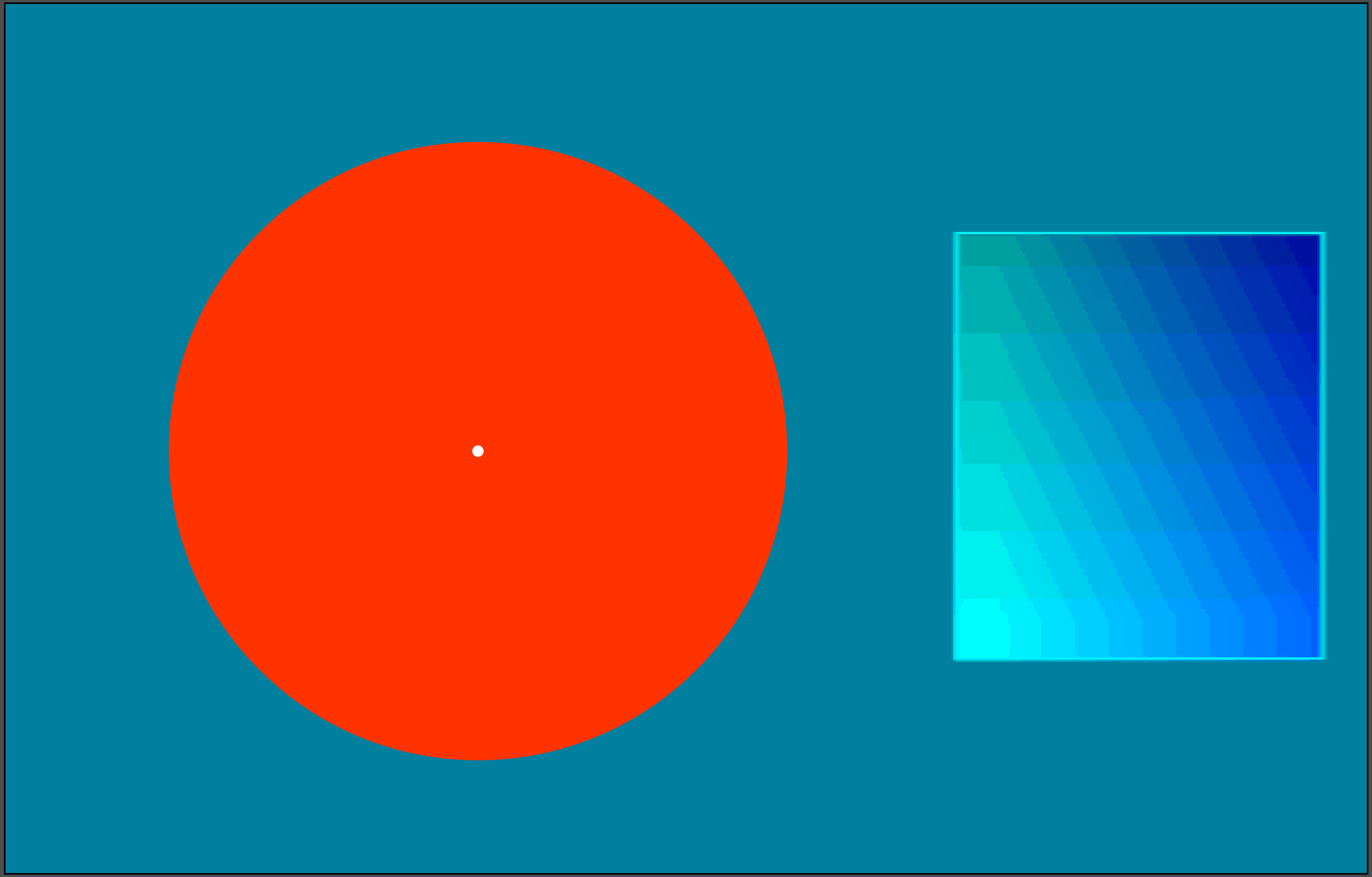


http://www.techmind.org/

Can RGB Represent Any Color?

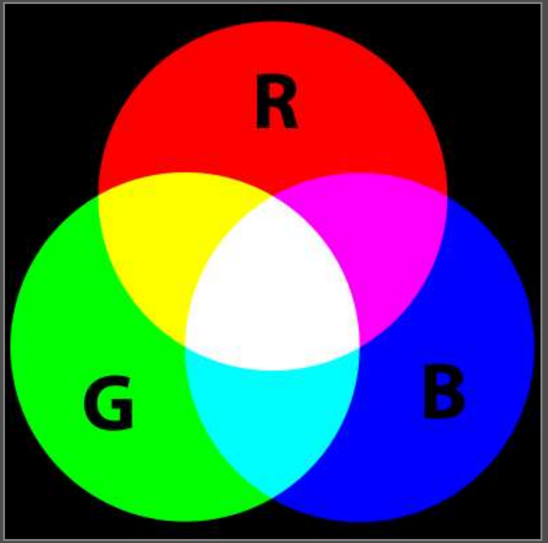
- "But my shiny new LCD monitor is state-of-the-art, it can surely show all colors!"
- → Let's see a color that it cannot show …
- another experiment, same proceedings as before
- look at the dot and continue staring at it

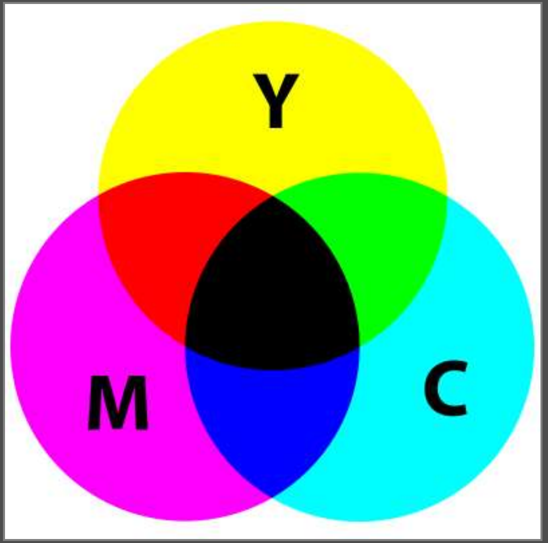
Let's see REAL cyan …



Additive vs. Subtractive Color

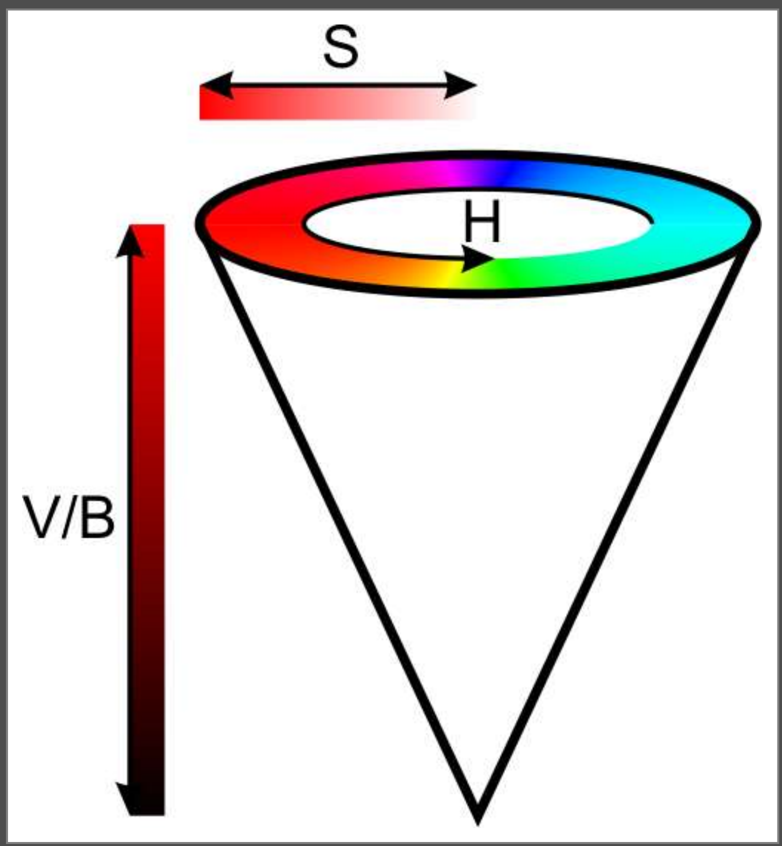
- (physical) color mixing depends on color production process
- device-dependent color models
  - light emission: additive mixing (CRTs etc.): **RGB model**
  - light absorption: subtractive mixing (printing process): **CMY(K) model**

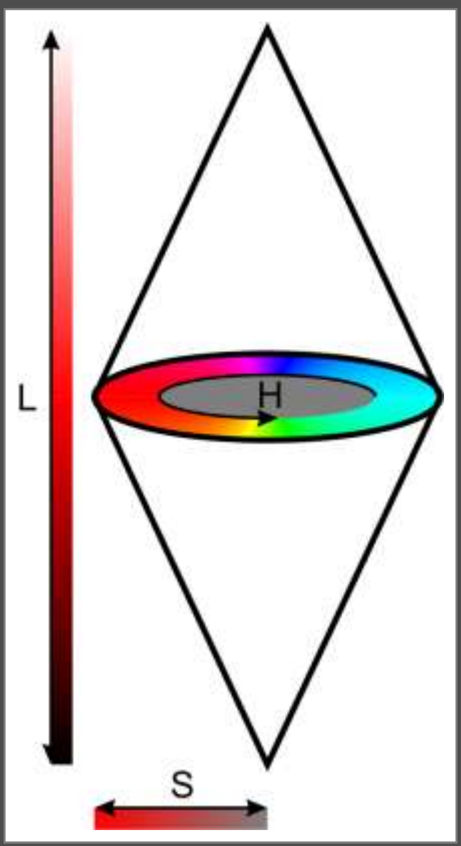




HSV/HSB and HSL/HLS Color Models

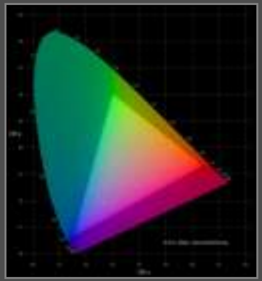
- human's inefficient with device models
- perceptual models to ease color specification

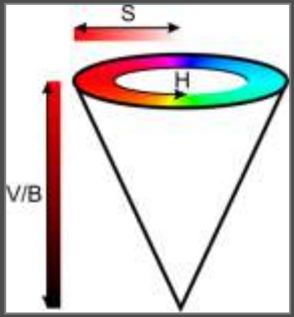


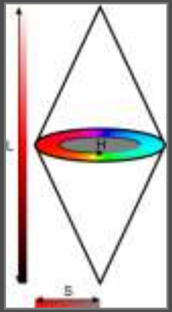


Color Models Summary

- 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)

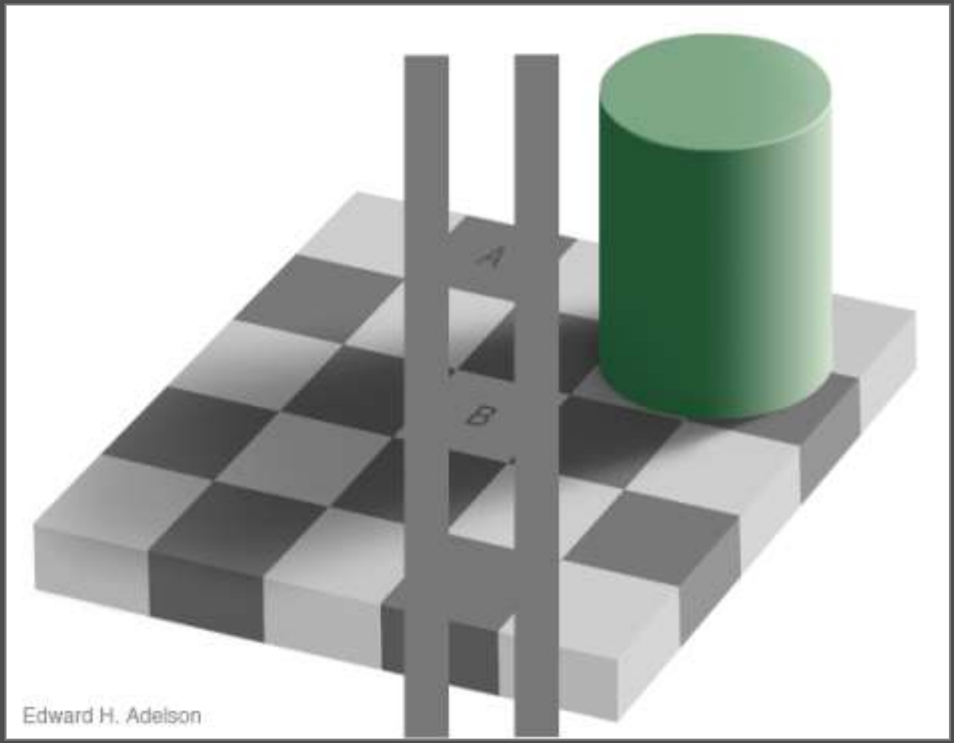


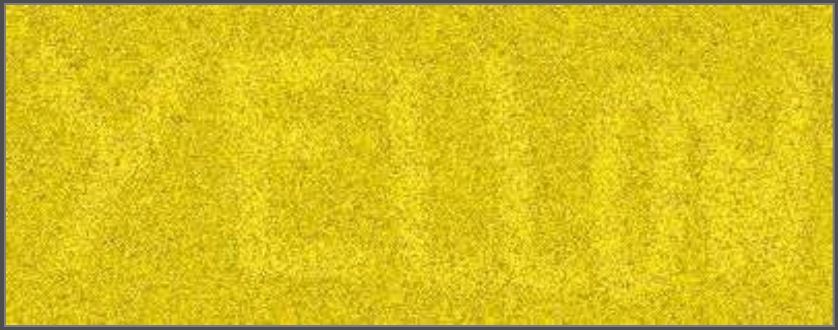




Other Color Topics

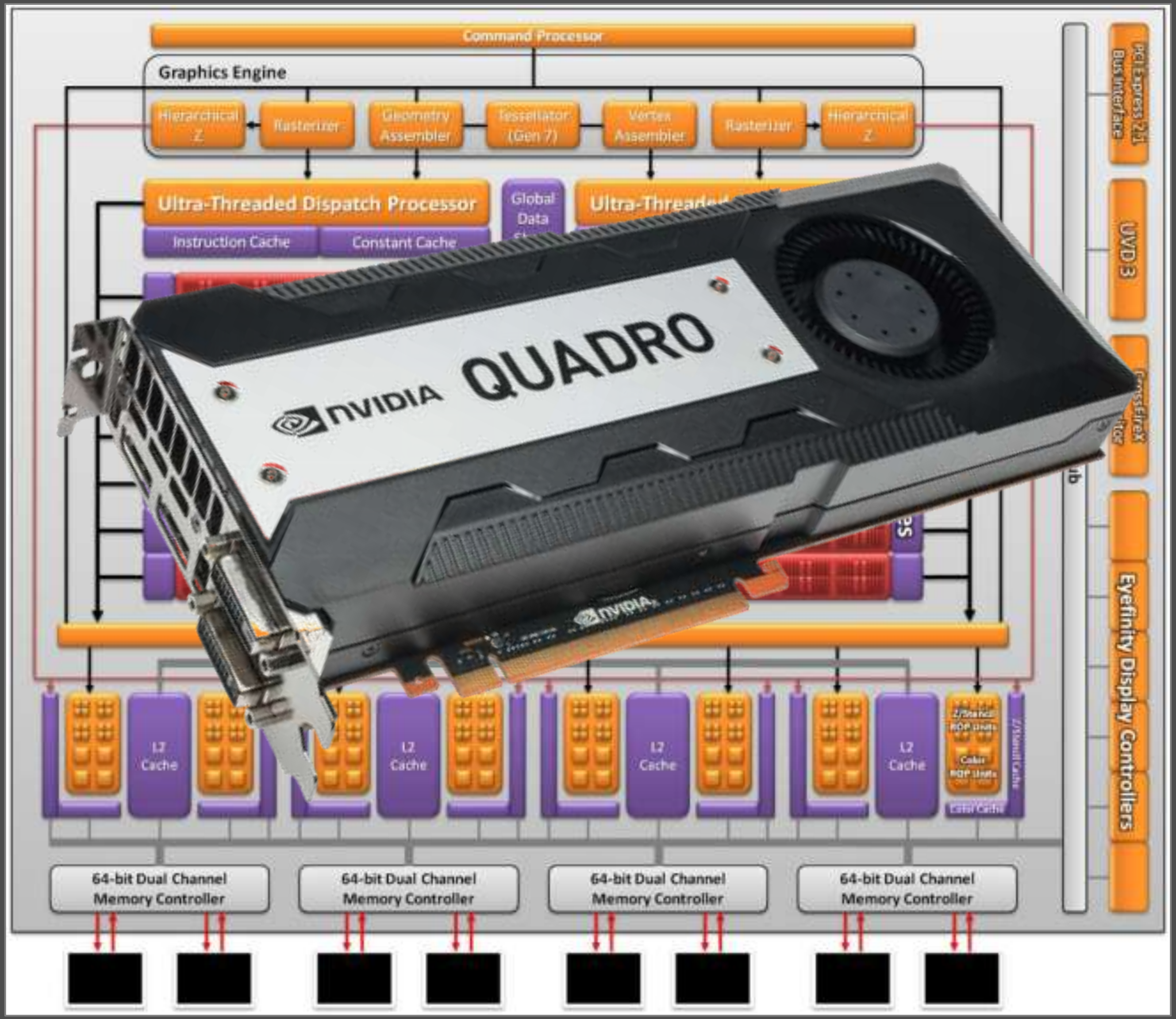
- color perception depending on context
- color deficiency





Hardware Rendering

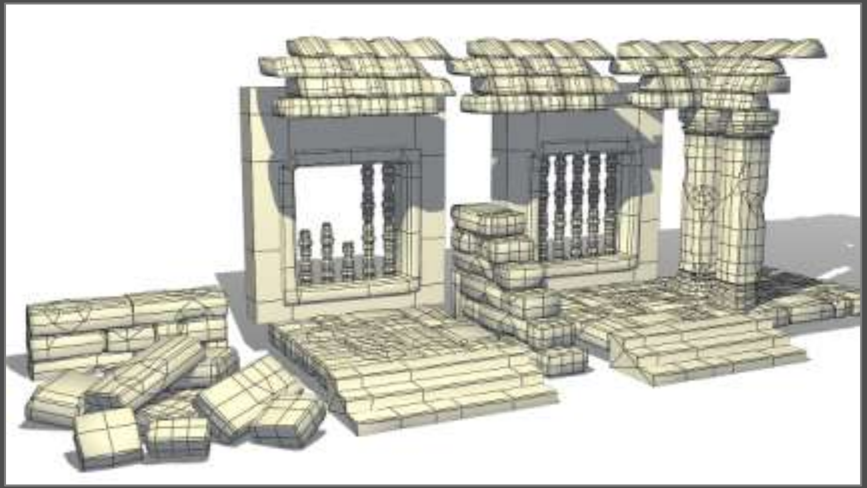
Why is graphics hardware effective?



from nVidia

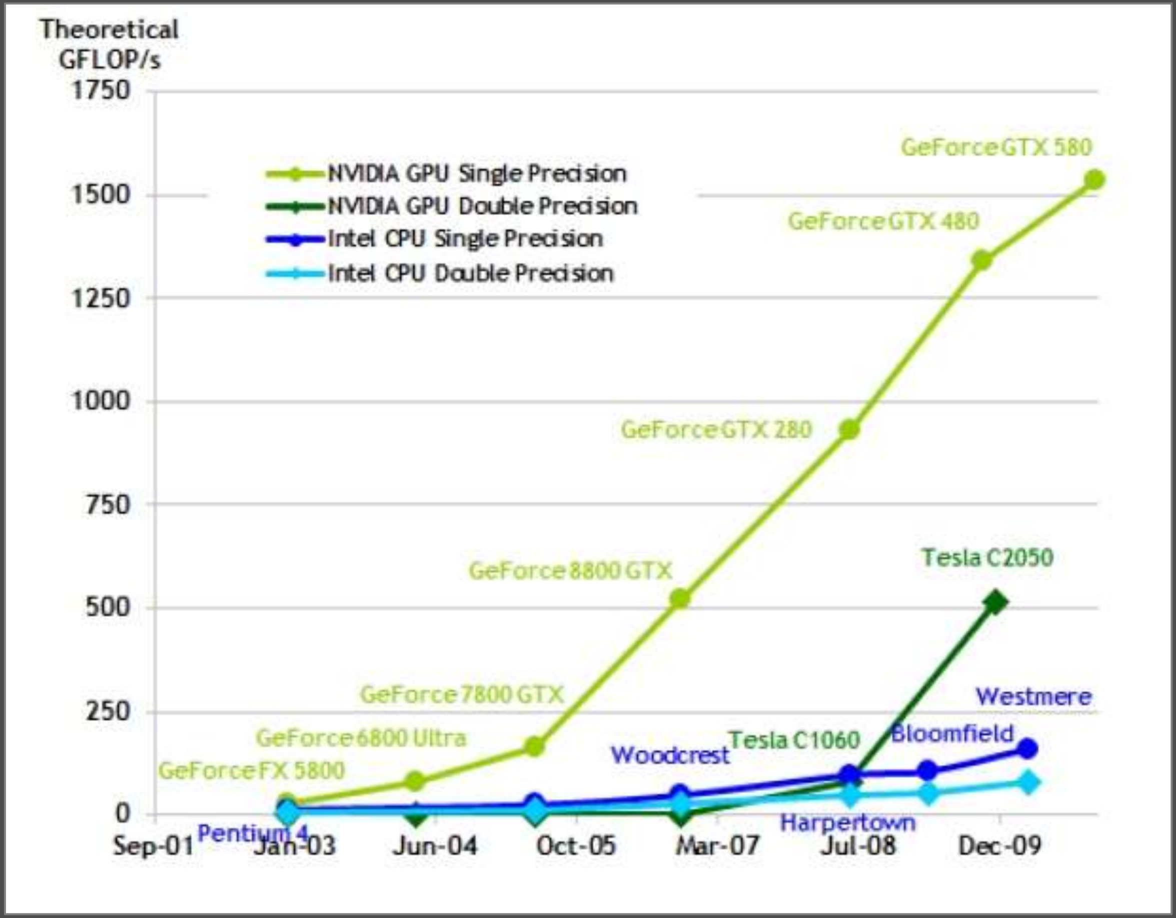
Why is graphics hardware effective?

- 3D rendering can be easily parallelized:
  - mesh: thousands of vertices; for each vertex we:
    - transform: object space → eye space → screen space
    - illuminate (compute vectors, attenuation, etc.)
    - shade (Gouraud per-vertex shading)
    - various other tasks …
  - mega-pixel images; for each pixel we:
    - shade (Phong per-pixel shading)
    - perform texture mapping
    - perform blending
    - various other tasks …



Why is graphics hardware effective?

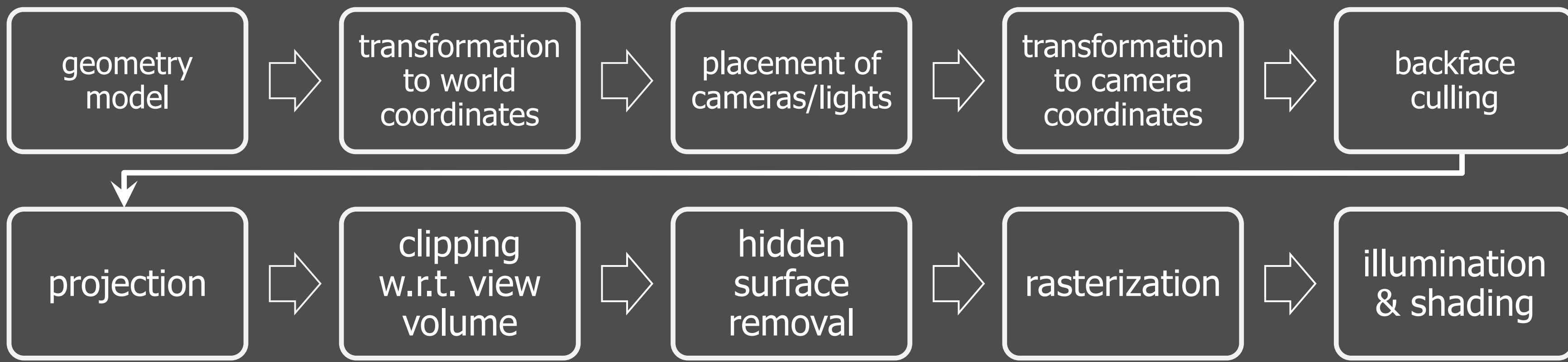
- GPU throughput is increasing at a much faster rate than CPU throughput.



from nVidia

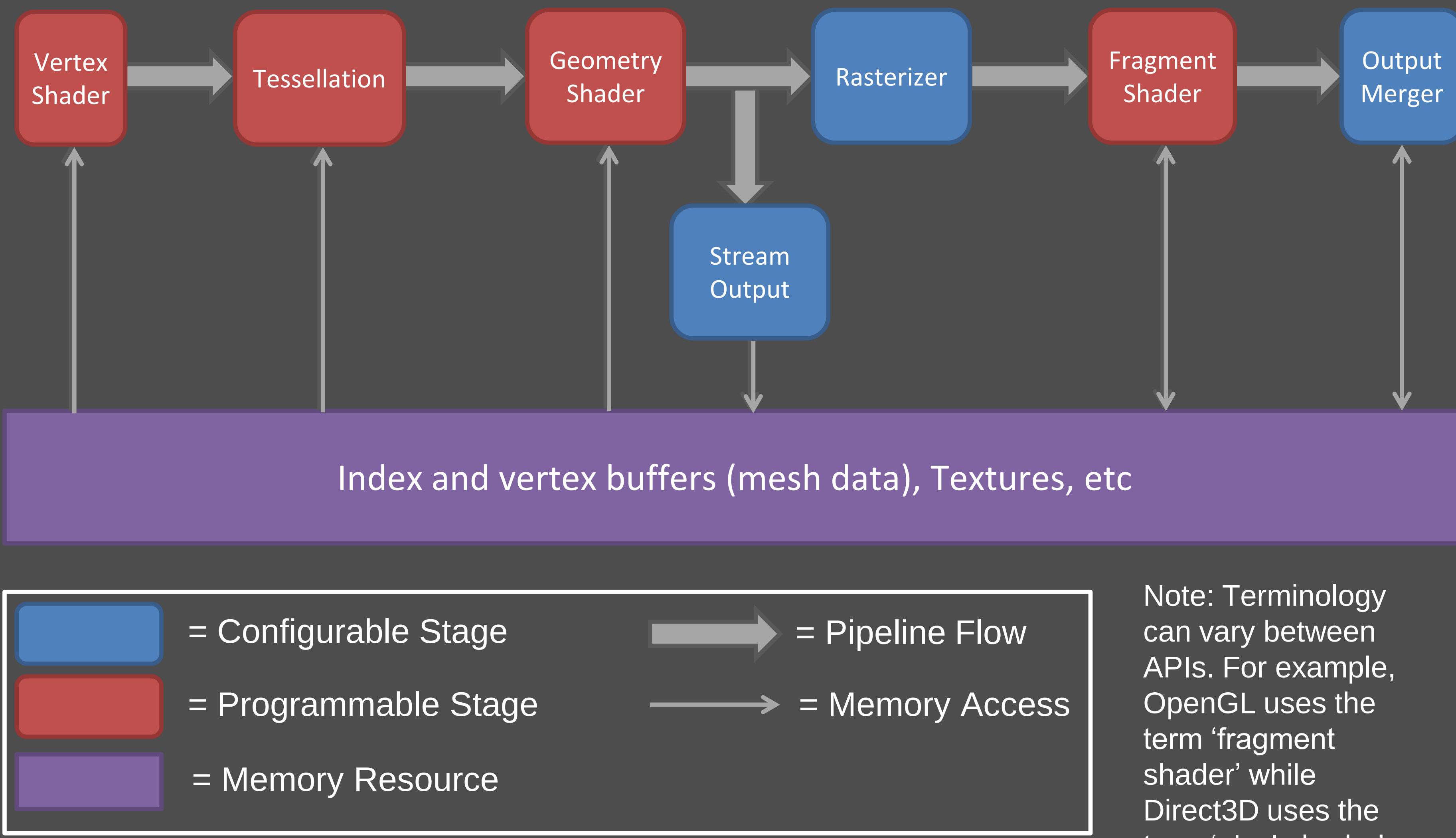
The graphics hardware pipeline

- traditional graphics pipeline:



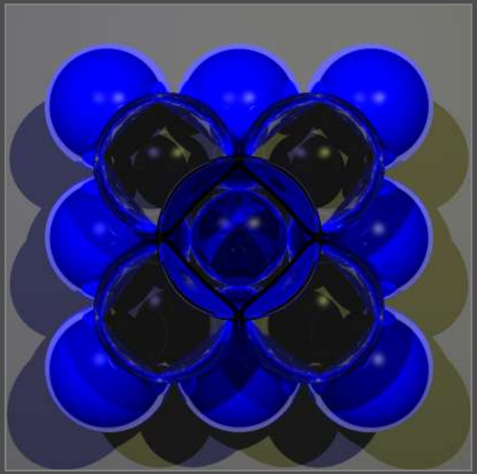
- transition to the programmable modern graphics hardware pipeline ...

The Modern Graphics Hardware Pipeline



Note: Terminology can vary between APIs. For example, OpenGL uses the term ‘fragment shader’ while Direct3D uses the term ‘pixel shader’.

Local vs. Global Illumination



Global Illumination: Introduction

- so far: only one interaction between light (from light source) and surface considered
  → **local illumination models**
- now: model several steps of reflection
  → **global illumination models**
- two techniques: raytracing & radiosity

Coombe et al., 2004

Raytracing

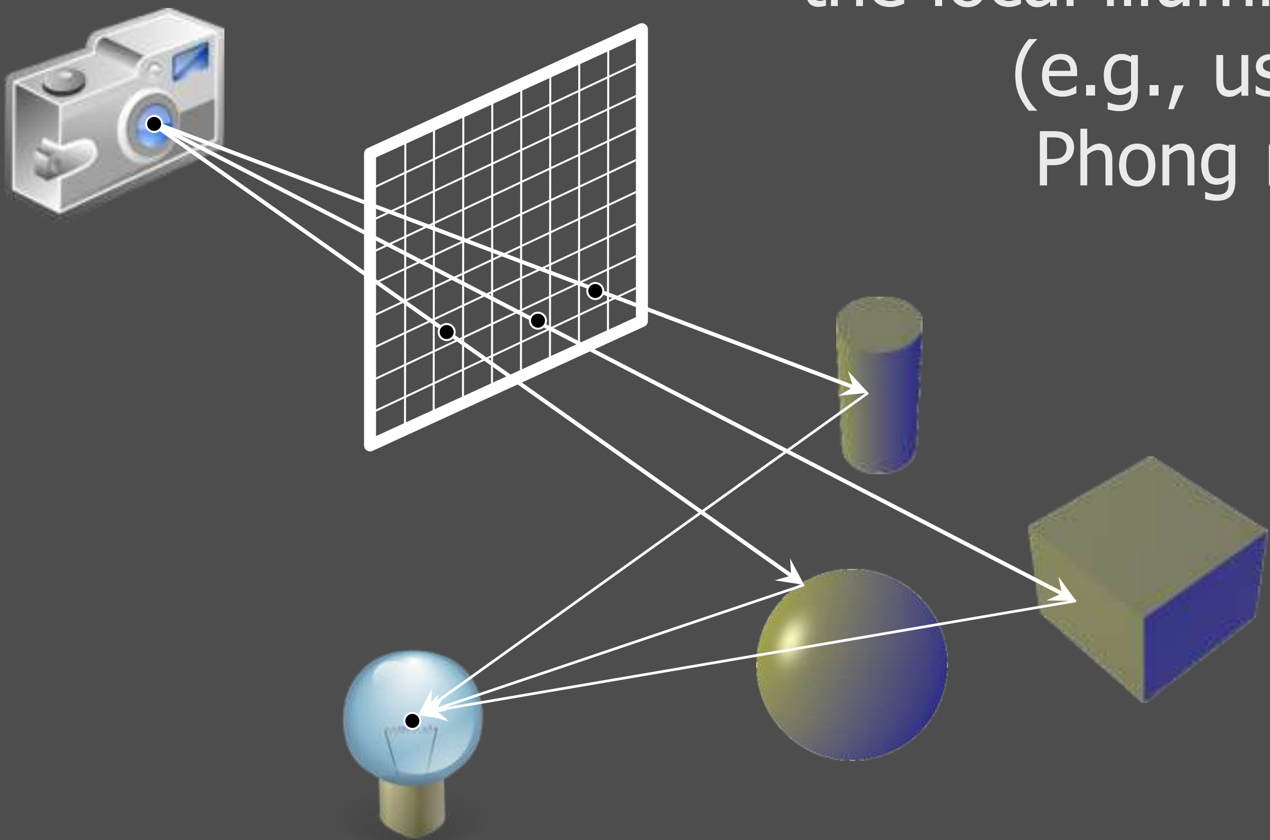
- solving of the rendering equation through Monte Carlo integration, i.e., through **repeated** (random) **sampling**
- sampling for each pixel of output image:

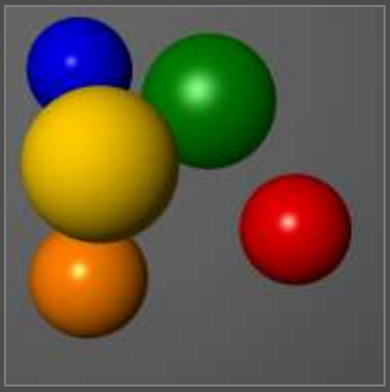
$$\underbrace{L_o(x,\vec{\omega}')}_{\text{pixel color}} = \underbrace{L_e(x,\vec{\omega}')}_{\text{emitted light of (first) intersection point}} + \int_\Omega \underbrace{f_r(x,\vec{\omega},\vec{\omega}')(\vec{\omega}\cdot\vec{n})}_{\text{illumination model (e.g., Phong model)}} \underbrace{L_i(x,\vec{\omega})}_{\text{light from light sources plus reflection plus refraction rays}} d\vec{\omega}$$

- further approximations needed: $\rightarrow$ **recursion** for specular light rays only

Raycasting

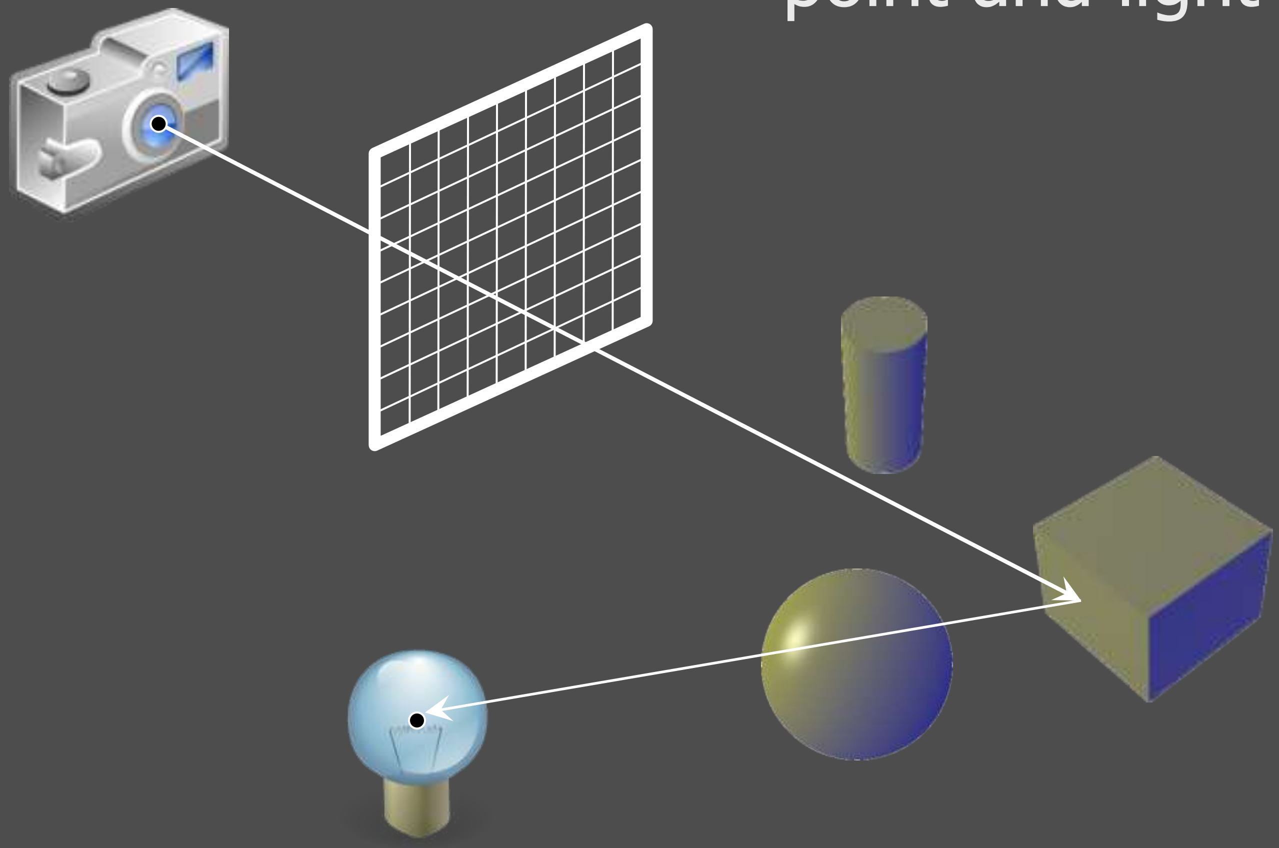
- shoot rays; for each intersection compute the local illumination (e.g., using the Phong model)

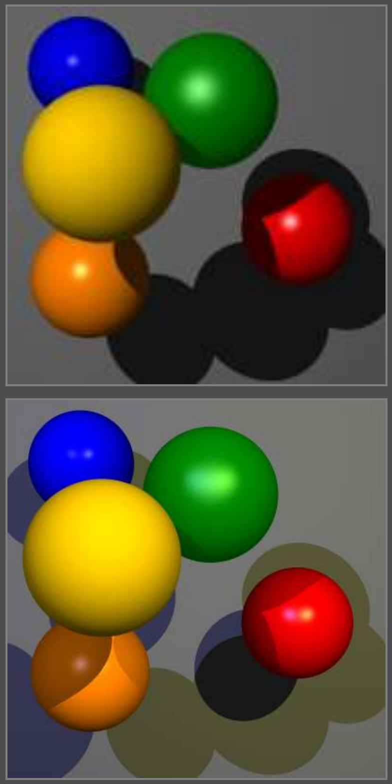




Generating Shadows

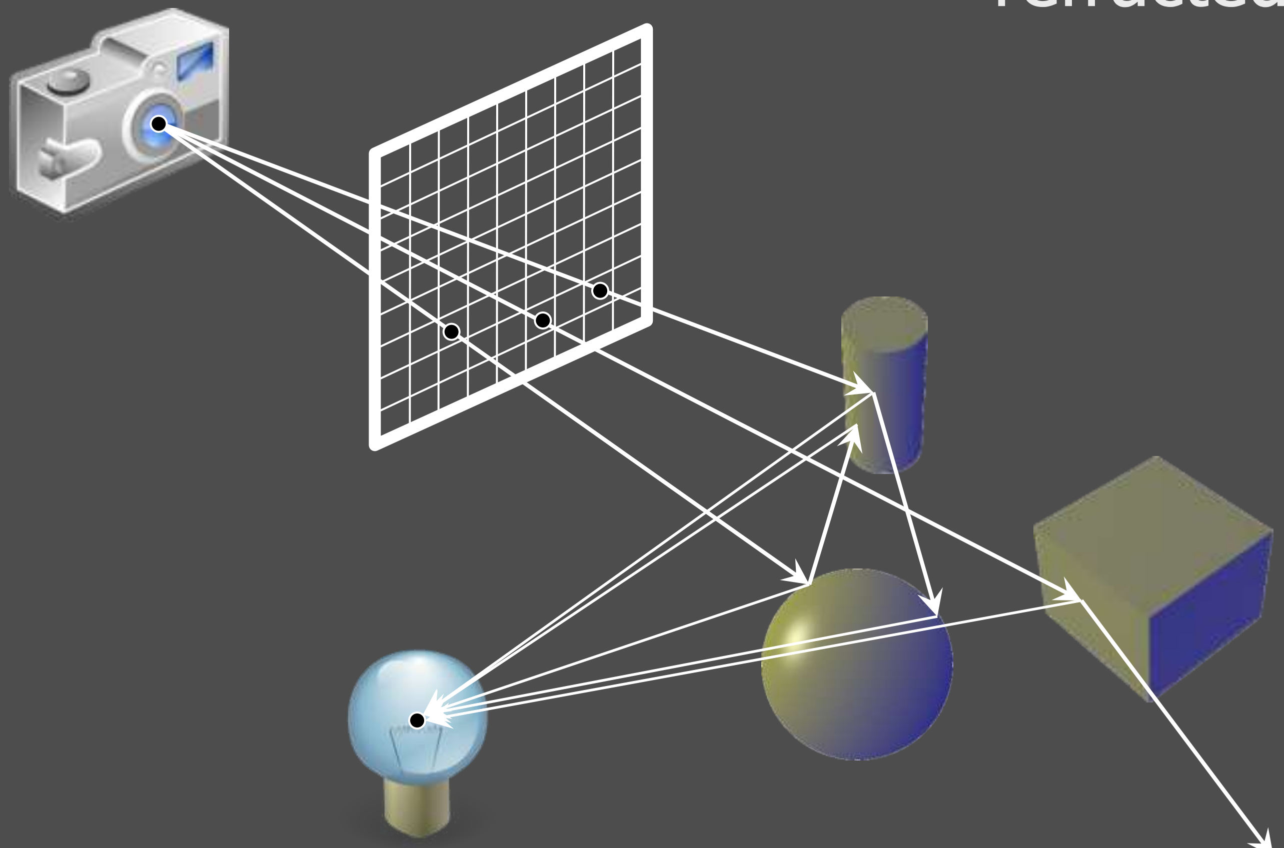
- shadow rays to test intersections between point and light source

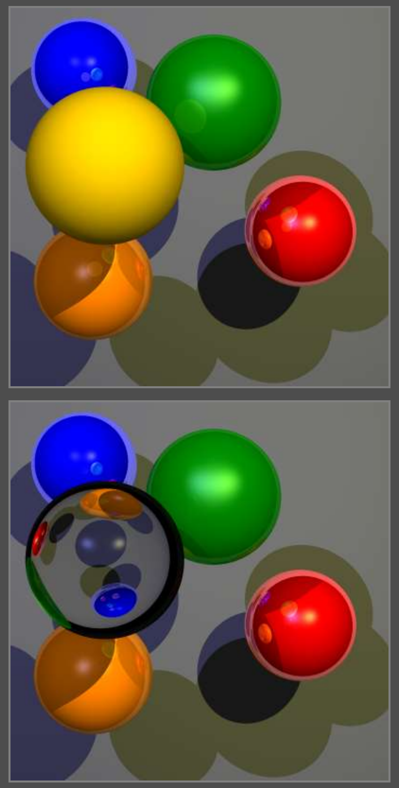




True Raytracing (Recursion)

- recursion: also follow reflected rays and refracted rays

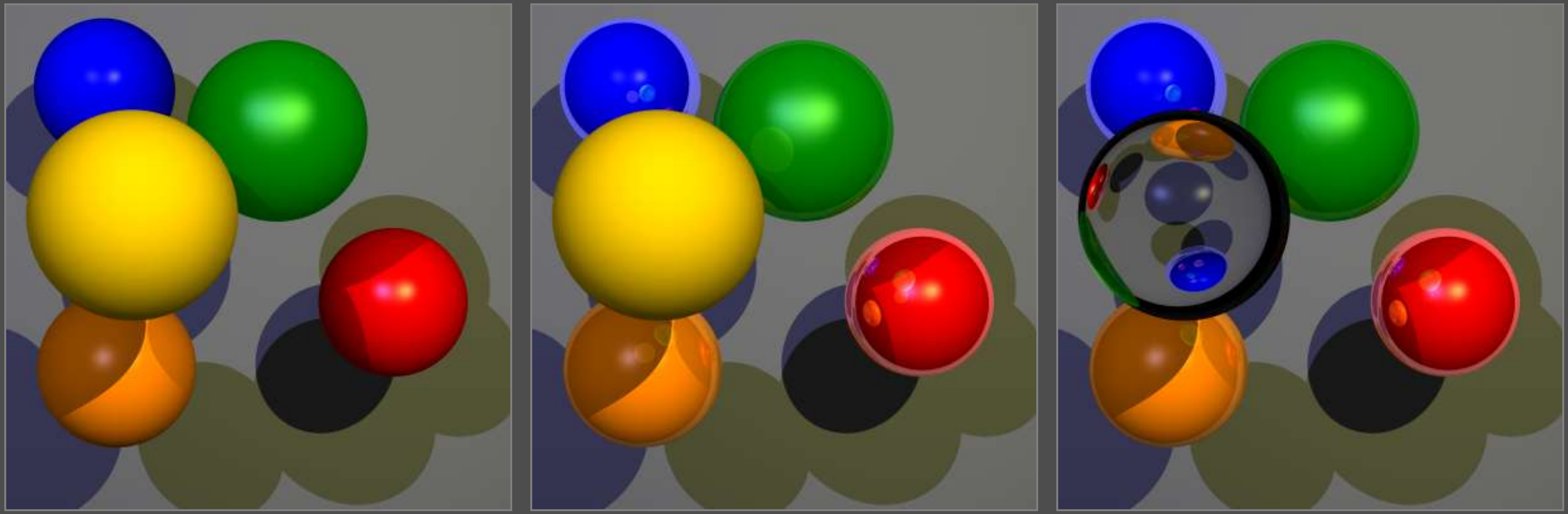




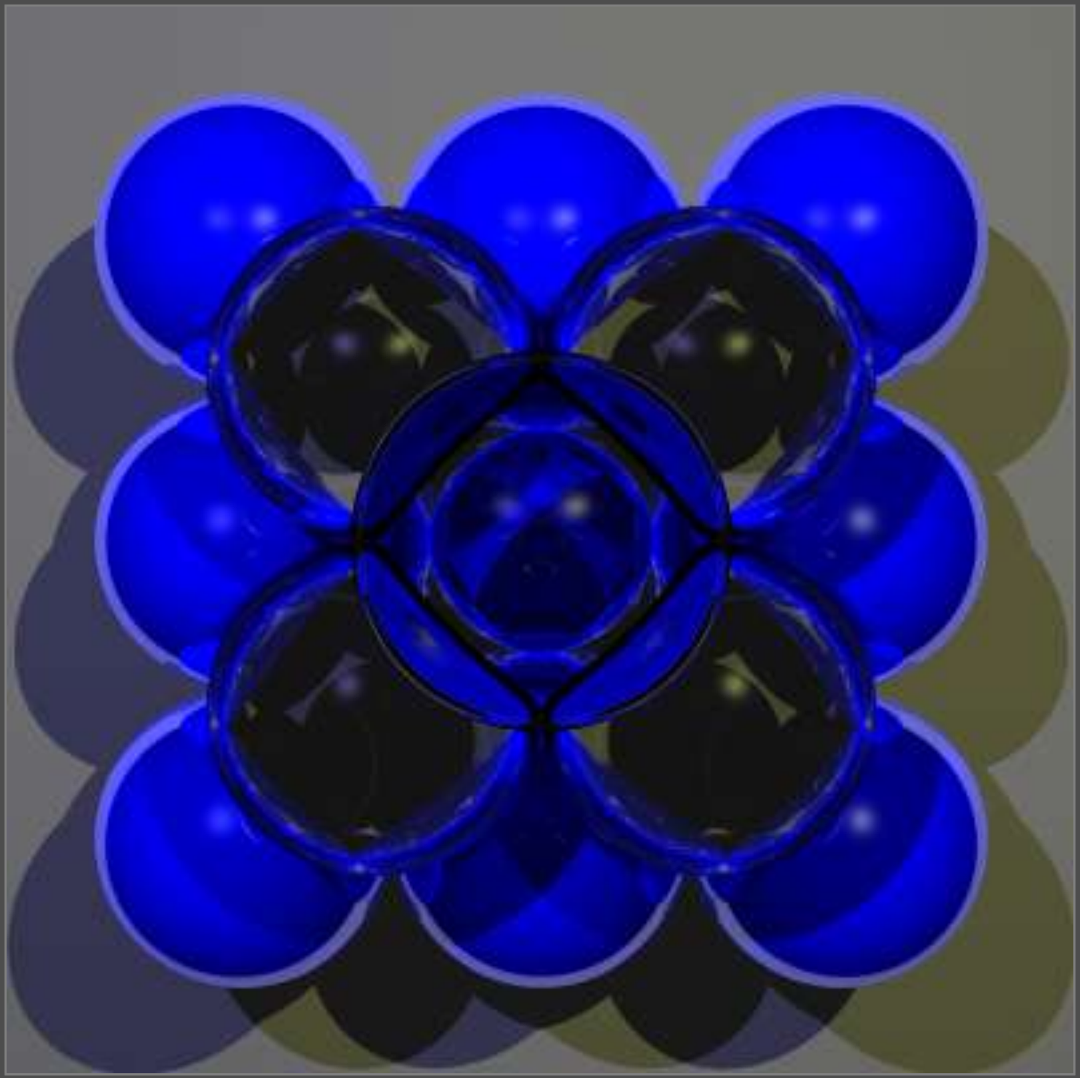
Adapted Illumination Model

- enhance the Phong model with terms for reflected (I_r) and refracted portion (I_t)
  (model by Whitted):

$$I = k_a I_a + \sum_i \left(k_d I_{d_i} (L \cdot N) + k_s I_{s_i} (R \cdot V)^e \right) + I_r + I_t$$



Raytracing Recursion Steps



Recursion [illegible]

Properties of Raytracing

- models directed global reflections: mirroring effects, shadows (shadow rays)
- hidden surface removal included
- does not account for certain light effects
  - caustics (photon tracing)
  - aerial light sources
  - (repeated) indirect diffuse reflection
- computationally expensive, but highly parallelizable (CPU and GPU)

Radiosity

- up to now: repeated diffuse reflection difficult → but this happens in reality!
- **radiosity**: technique to model this **repeated diffuse reflection**

from Cornell University

Radiosity

- **radiosity**: rate at which energy leaves a surface
- let's consider just diffuse reflection for now
- general idea:
  1. sub-divide scene into many small parts
  2. determine light interactions between parts of the scene independent of viewing
  3. simulate the illumination situation
  4. render a specific view

Radiosity: Examples

Cornell University, 1988

Cohen et al., 1988

Radiosity: Quality

photograph

simulation

difference image

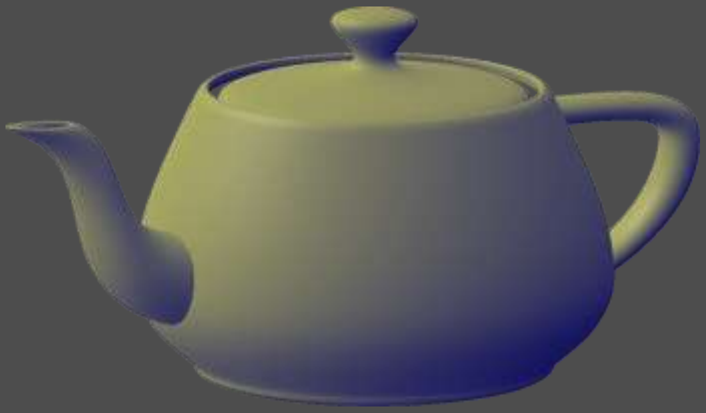
from Cornell University

Radiosity: Summary

from Cornell University

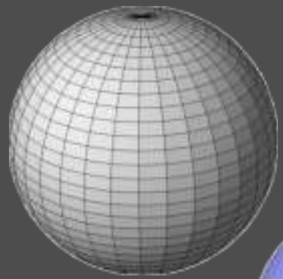
- good representation of repeated diffuse reflection in a scene
- approximation of the rendering equation
- discretization and patches
- very expensive due to huge matrix sizes
- different strategies to solve linear system
- only needs to be re-computed when scene changes, not for moving through the scene

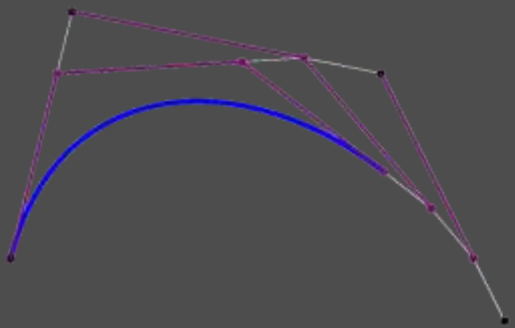
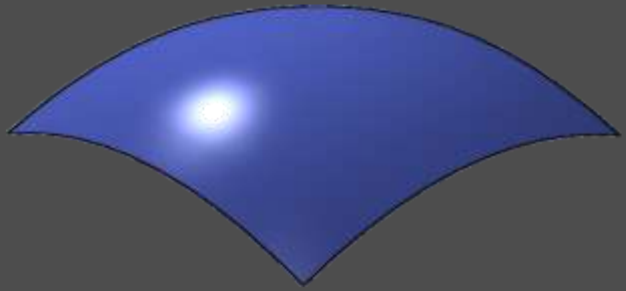
Curves & Surfaces



Curves and Smooth Surfaces

- object representations so far
  - polygonal meshes (shading)
  - analytical descriptions (raytracing)
- flexibility vs. accuracy
- now: flexible yet accurate representations
  - piecewise smooth curves: *Bézier curves, splines*
  - smooth (freeform) surfaces
  - subdivision surfaces



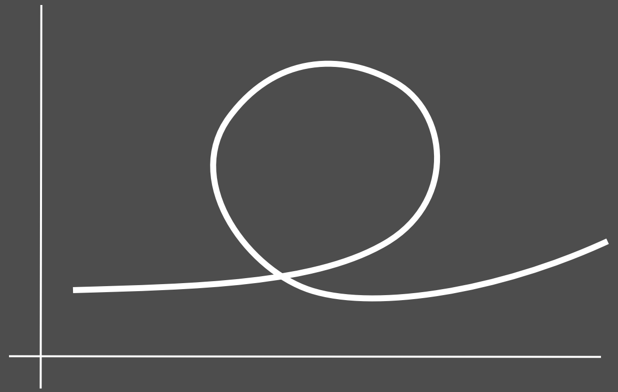



Again, why do we need all this?

- not only representation, but also **modeling**
- and it's all about cars! shiny cars! ☺

Specifying Curves

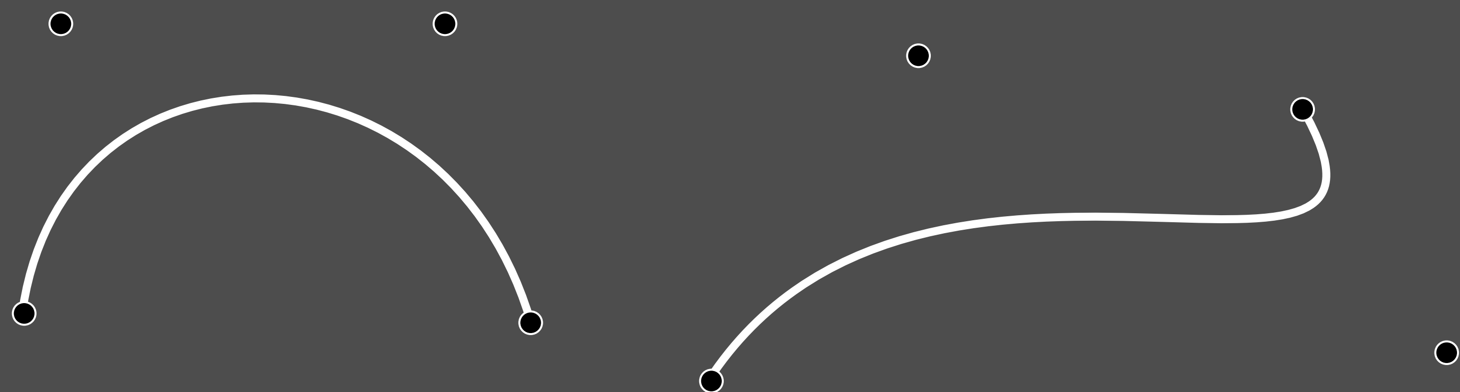
- functional descriptions
  - $y = f(x)$ in 2D;
    for 3D also $z = f(x)$
  - cannot have loops
  - functions only return one scalar, bad for 3D
  - difficult handling if it needs to be adapted



- parametric descriptions
  - independent scalar parameter $t \in \Re$
  - typically $t \in [0, 1]$, mapping into $\Re^2$ / $\Re^3$
  - point on the curve: $P(t) = (x(t), y(t), z(t))$

Polynomial Parametric Curves

- use **control points** to specify curves

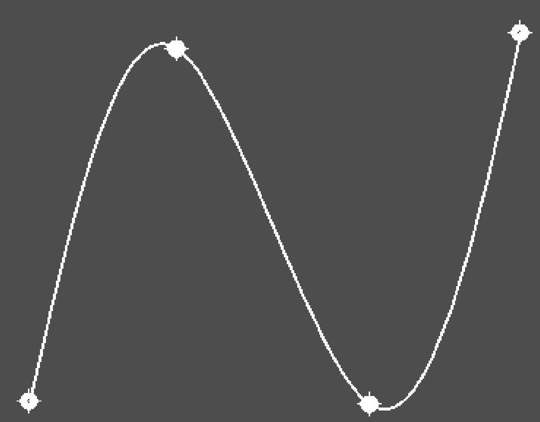


- n control points for a curve segment
- set of **basis** or **blending functions**:

$$P(t) = \sum_{i=0}^{n} P_i B_{i,n}(t)$$

Interpolating vs. Approximating

- two different curves schemes: curves do not always go through all control points
  - **approximating curves** not all control points are on the resulting curve
  - **interpolating curves** all control points are on the resulting curve



Bézier Curves: Blending Functions

- formulation of curve: $$P(t) = \sum_{i=0}^{n} P_i B_{i,n}(t)$$
- $B_{i,n}$ – **Bernstein polynomials** (control point weights, depend on t):

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} = \frac{n!}{i!\,(n-i)!} t^i (1-t)^{n-i}$$

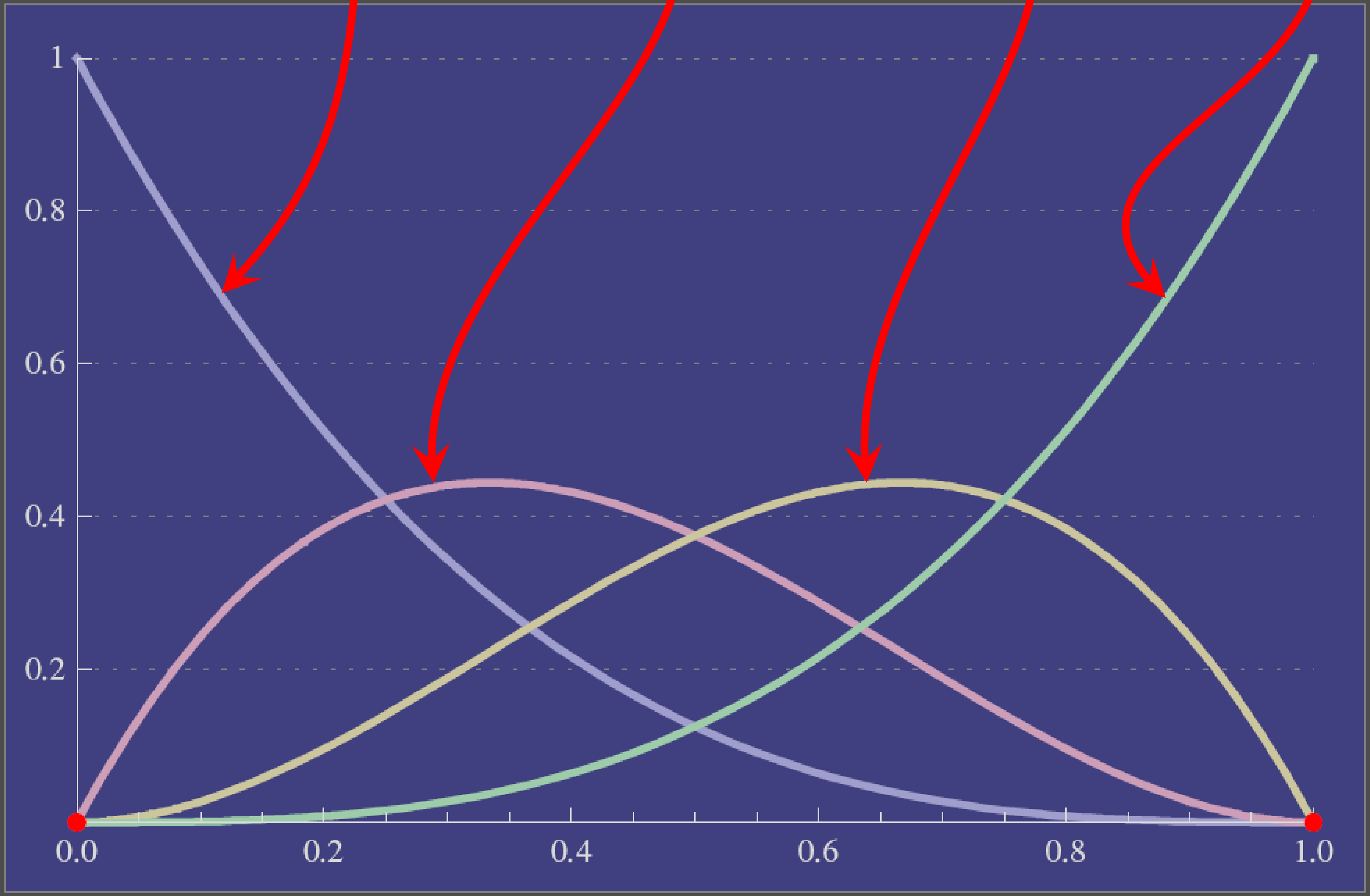
- Bézier curve example for n = 3:

$$P(t) = P_0 B_{0,3}(t) + P_1 B_{1,3}(t) + P_2 B_{2,3}(t) + P_3 B_{3,3}(t)$$

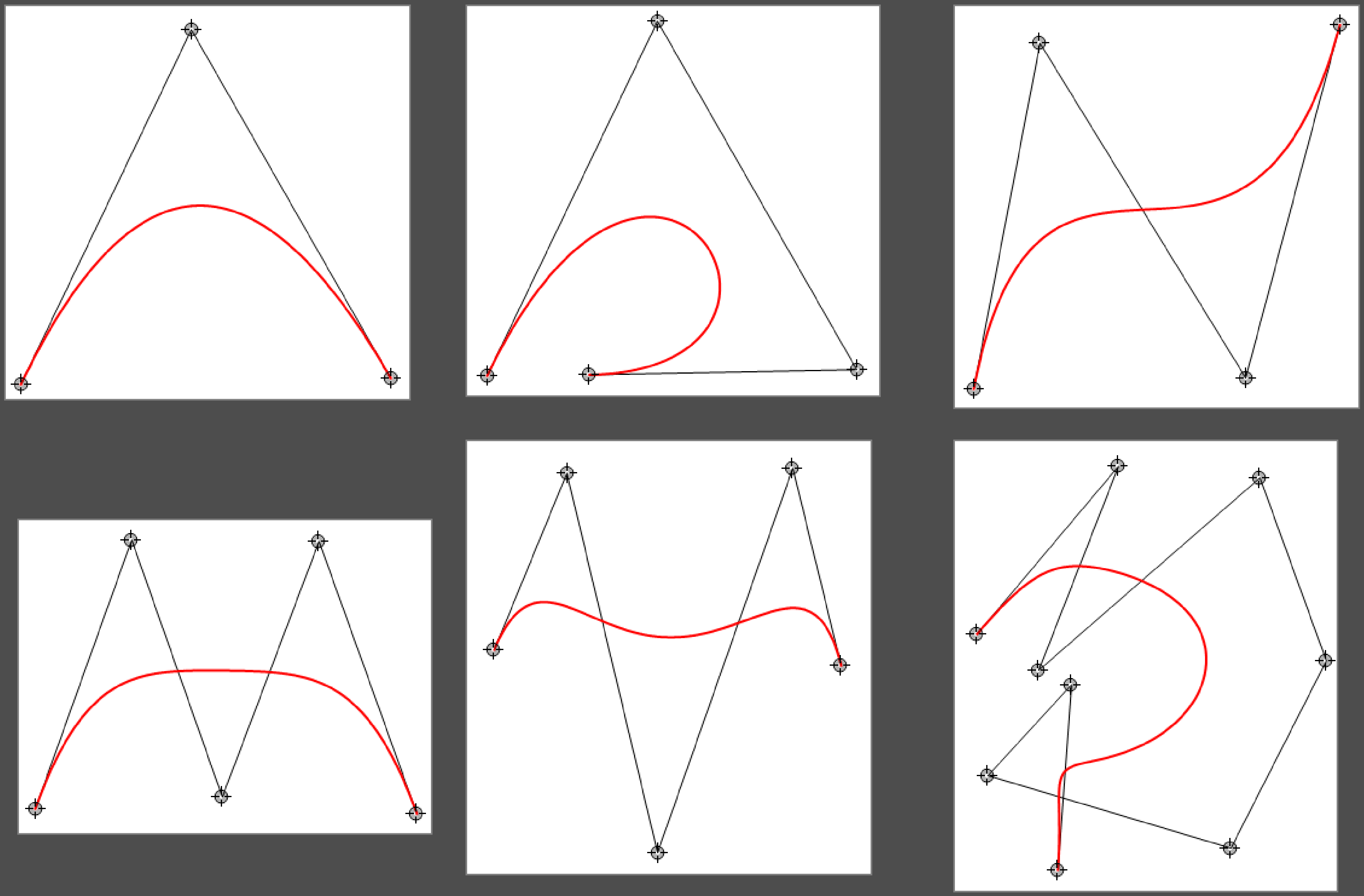
$$= P_0\,(1-t)^3 + P_1\,3t(1-t)^2 + P_2\,3t^2(1-t) + P_3\,t^3$$

Bernstein Polynomials Visualized

$$P(t) = P_0 \boxed{(1-t)^3} + P_1 \boxed{3t(1-t)^2} + P_2 \boxed{3t^2(1-t)} + P_3 \boxed{t^3}$$

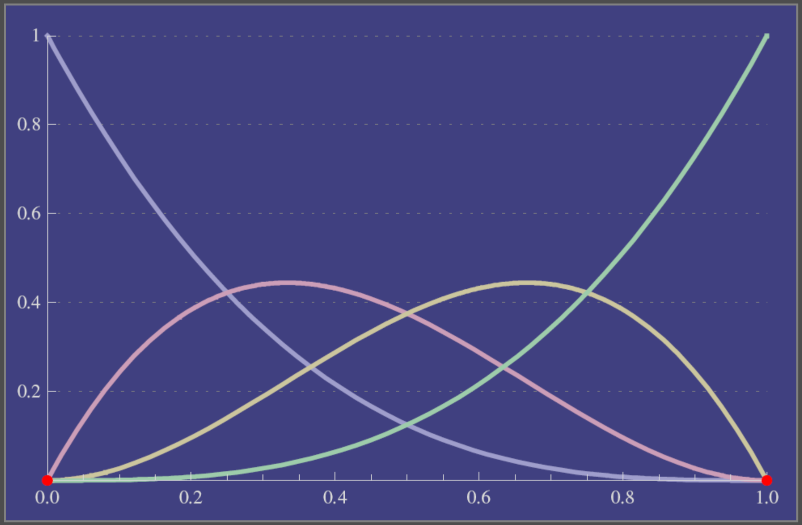


Bézier Curves: Examples



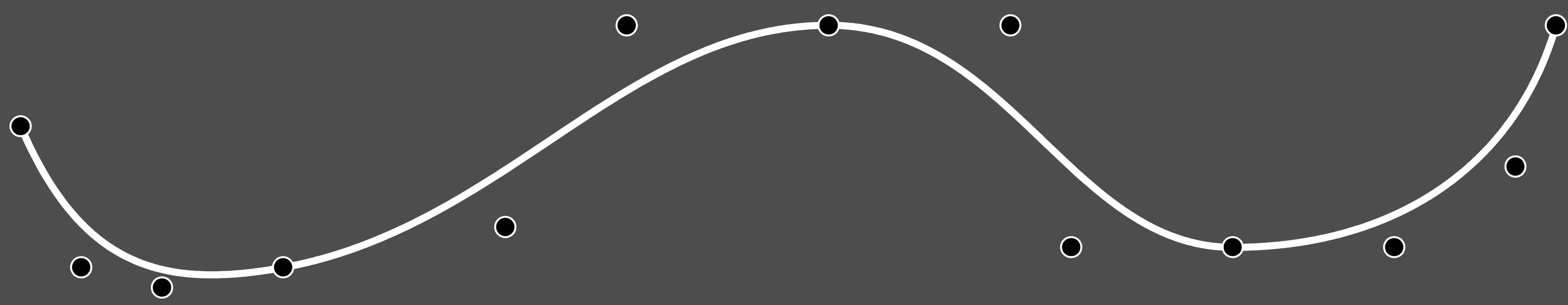
Bézier Curves: Properties

- curve always inside the **convex hull** of the control polygon
- **approximating** curve: only first & last control points are interpolated – why?
- each control point affects the entire curve, **limited local control** → problem for modeling



Piecewise Smooth Curves

- low order curves give sufficient control
- *idea*: connect segments together
  - each segment only affected by its own control points → local control
  - make sure that segments connect smoothly

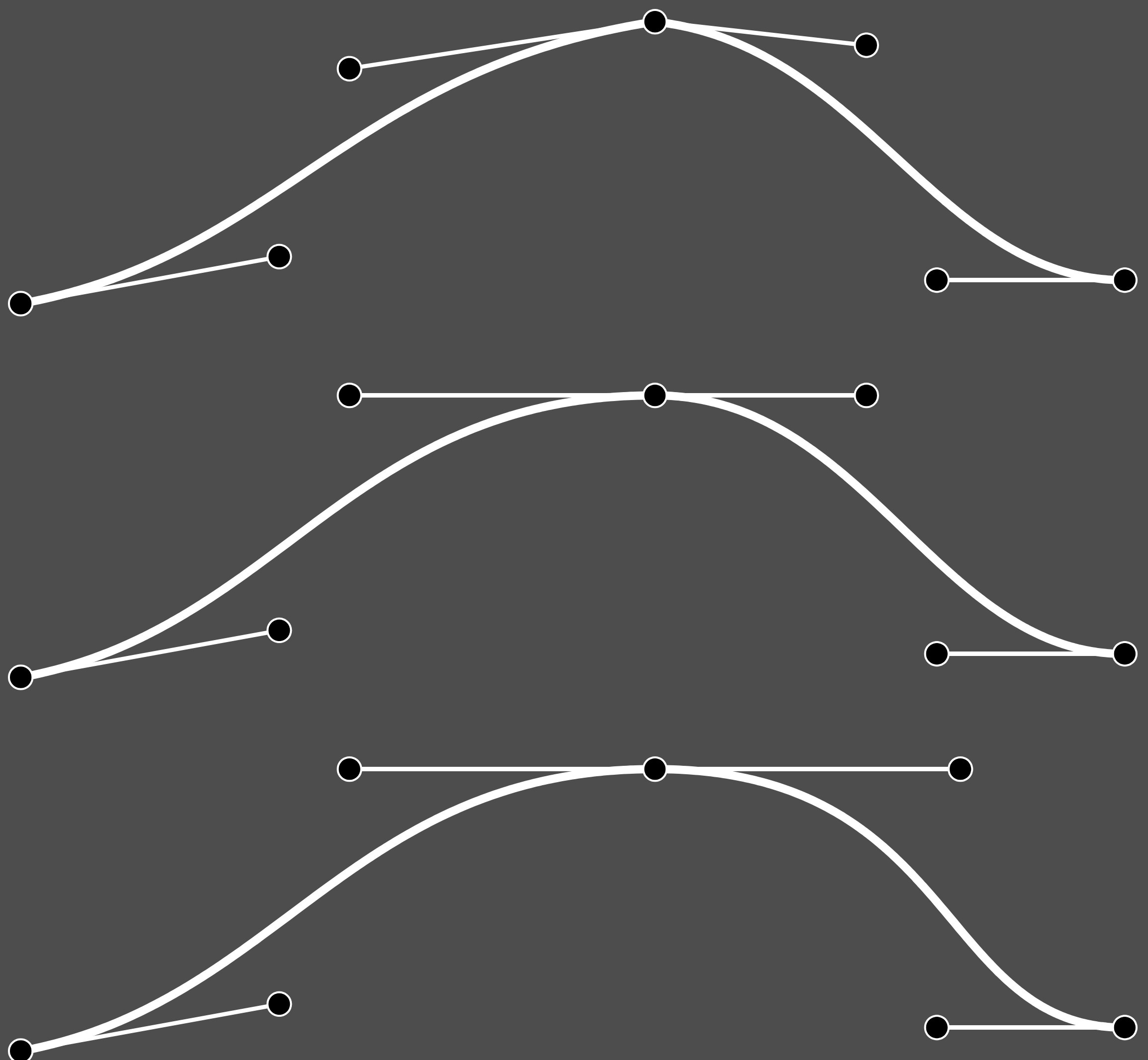


- *problem*: what are smooth connections?

Continuity Criteria

- a curve *s* is said to be C^n-continuous if its n^{th} derivative $d^n s/dt^n$ is continuous of value
  $\rightarrow$ **parametric continuity**: shape & speed
- not only for individual curves, but also *and in particular* for where segments connect
- **geometric continuity**: two curves are G^n-continuous if they have proportional n^{th} de-rivatives (same direction, speed can differ)
- G^n follows from C^n, but not the other way
- car bodies need at least G^2-continuity

Continuity Criteria: Examples



$G^0 = C^0$

G^1

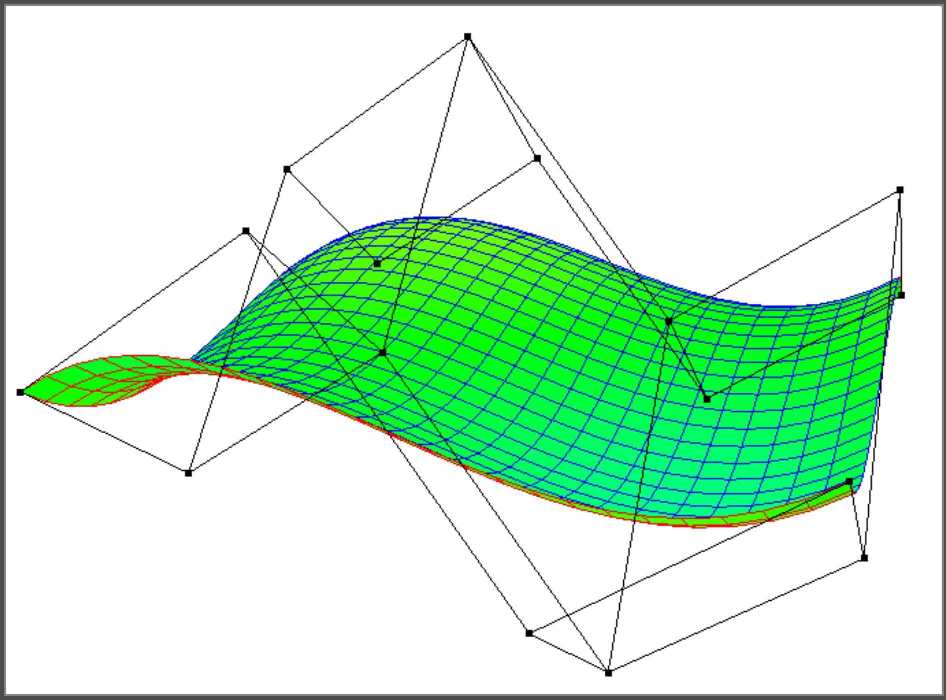
C^1

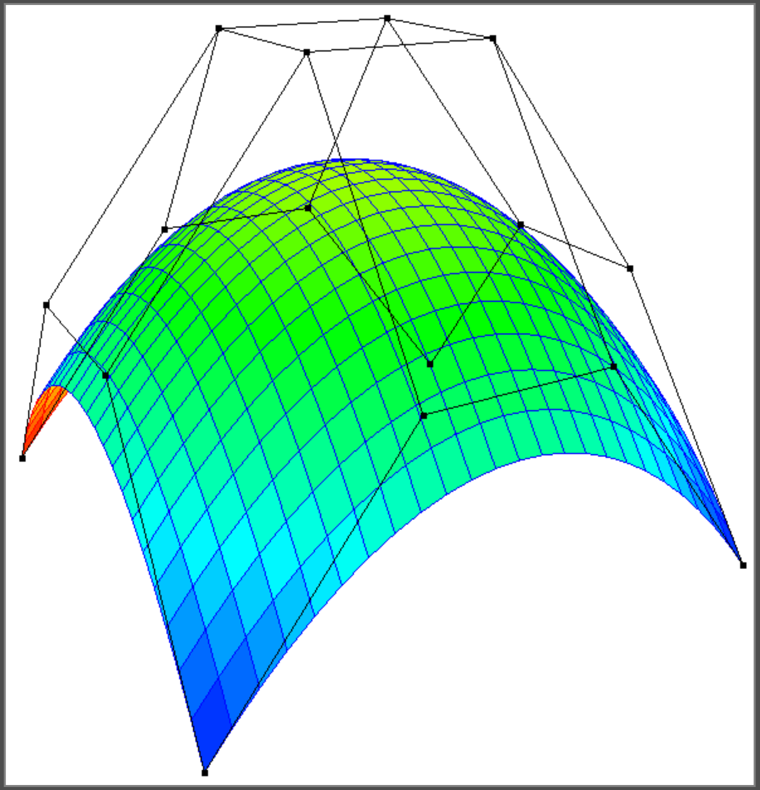
Freeform Surfaces

- base surfaces on parametric curves
- Bézier curves $\rightarrow$ Bézier surfaces/patches
- spline curves $\rightarrow$ spline surfaces/patches
- mathematically:
  application of curve formulations
  along two parametric directions

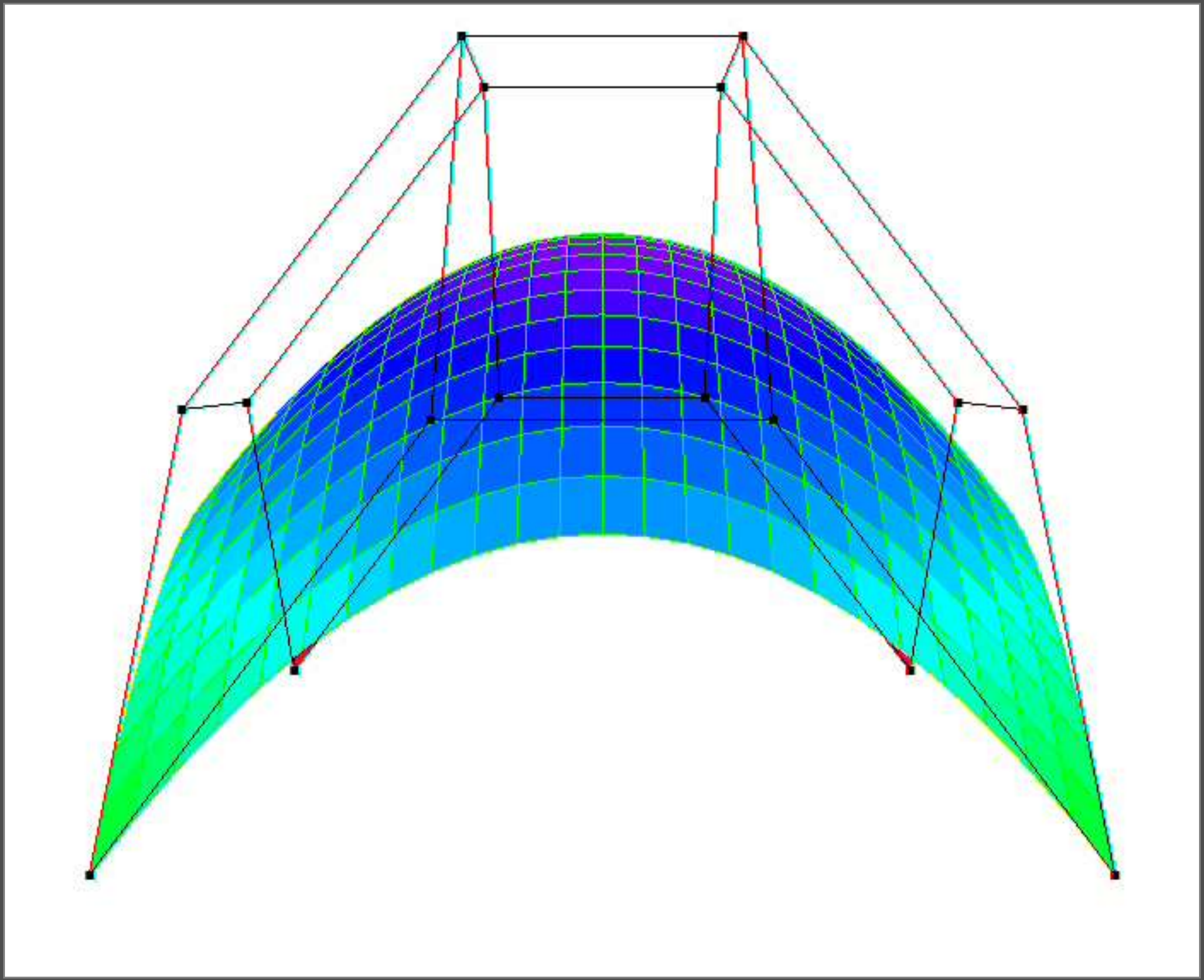
Freeform Surfaces: Principle

- Bézier surface: control mesh with m × n control points now specifies the surface:

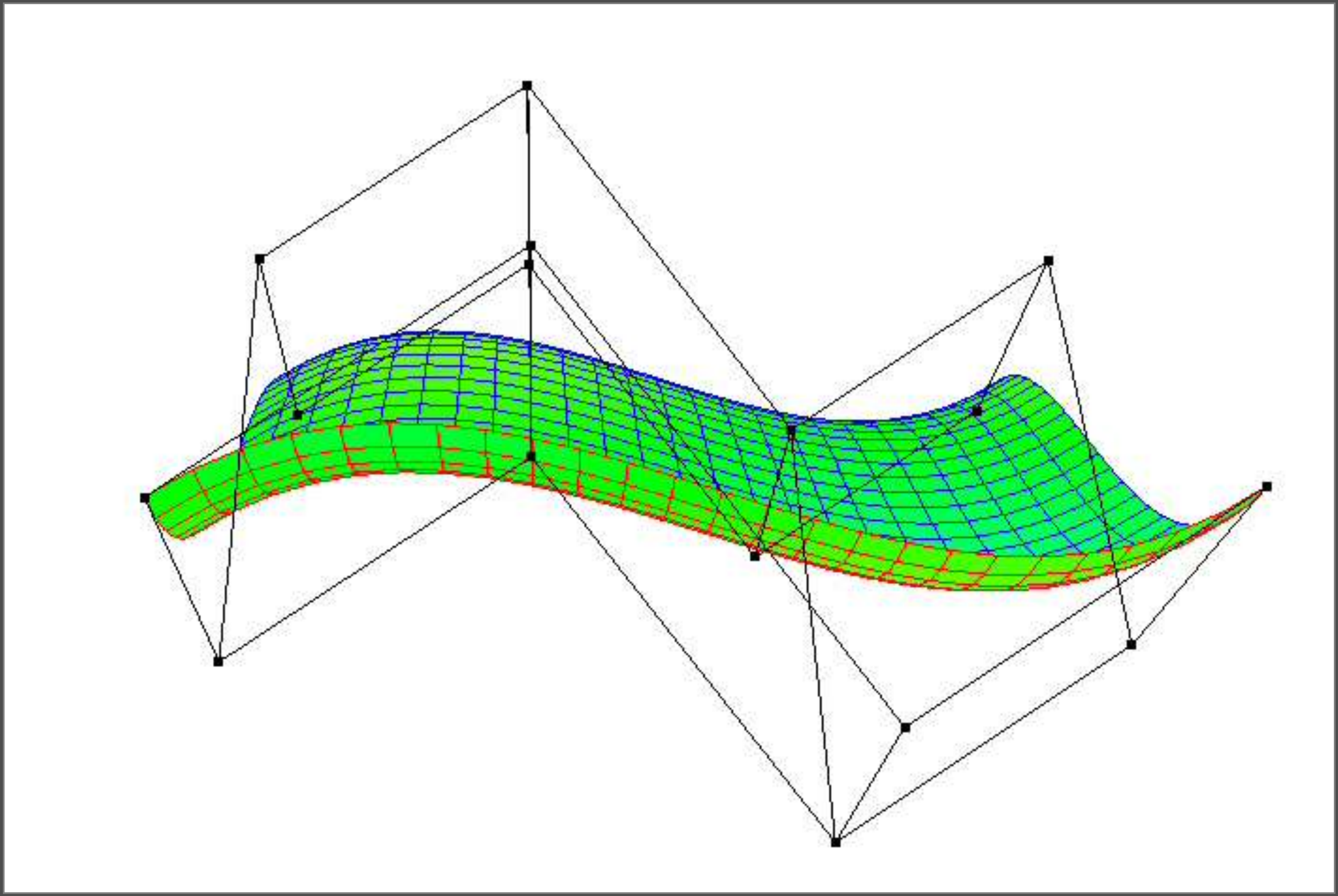




Freeform Surfaces: Examples



Freeform Surfaces: Examples



Subdivision Surfaces

- but we already have so many polygon models, is there anything we can do?
- sure there is: **subdivision surfaces**!
- basic idea:
  - model coarse, low-resolution mesh of object
  - recursively refine the mesh using rules
  - use high-resolution mesh for rendering
  - limit surface should have continuity properties and is typically one of the freeform surfaces

Subdivision Surfaces: Example