University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2016

Tamara Munzner

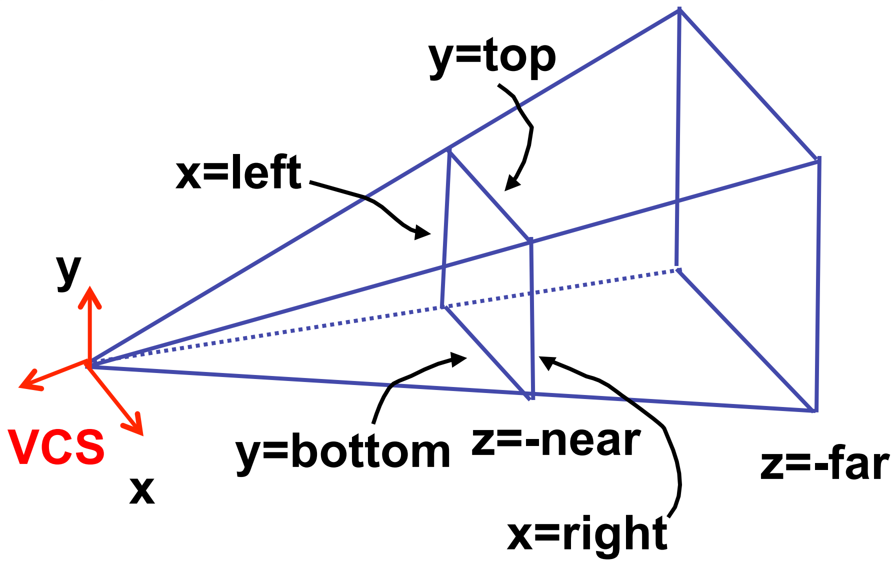
Final Review 2

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2016

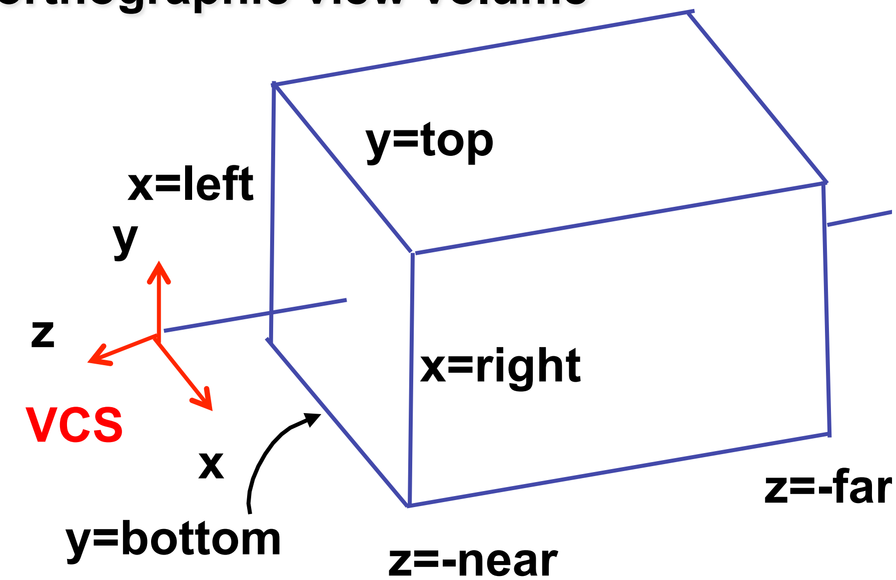
Viewing, Continued

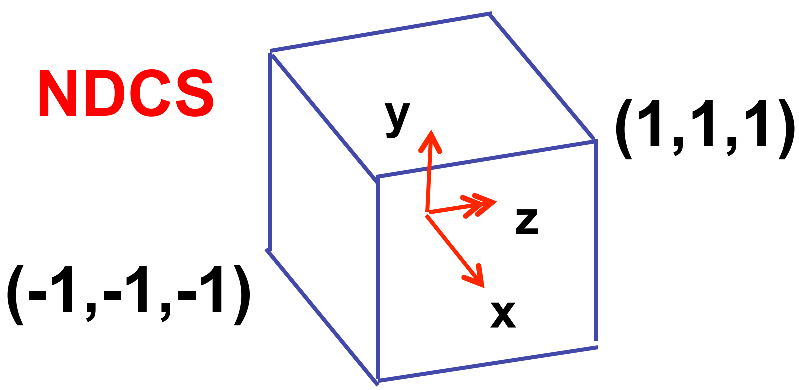
Review: From VCS to NDCS

perspective view volume



orthographic view volume

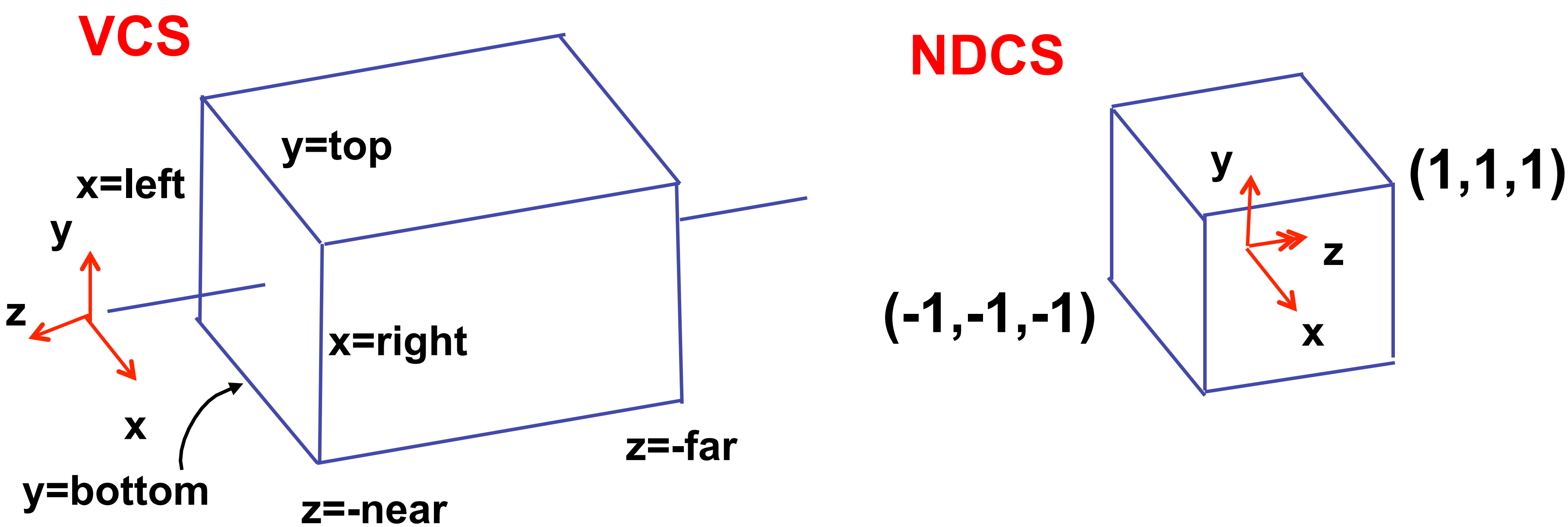




- orthographic camera
  - center of projection at infinity
  - no perspective convergence

Review: Orthographic Derivation

- scale, translate, reflect for new coord sys



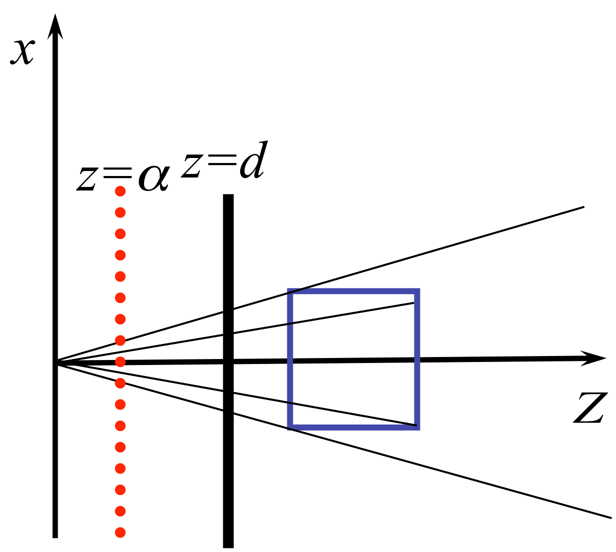
Review: Orthographic Derivation

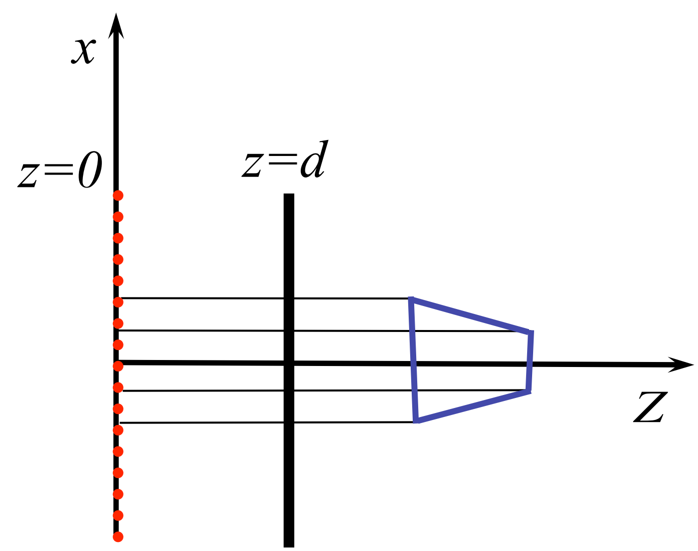
- scale, translate, reflect for new coord sys

$$P' = \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & -\frac{right + left}{right - left} \\ 0 & \frac{2}{top - bot} & 0 & -\frac{top + bot}{top - bot} \\ 0 & 0 & \frac{-2}{far - near} & -\frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix} P$$

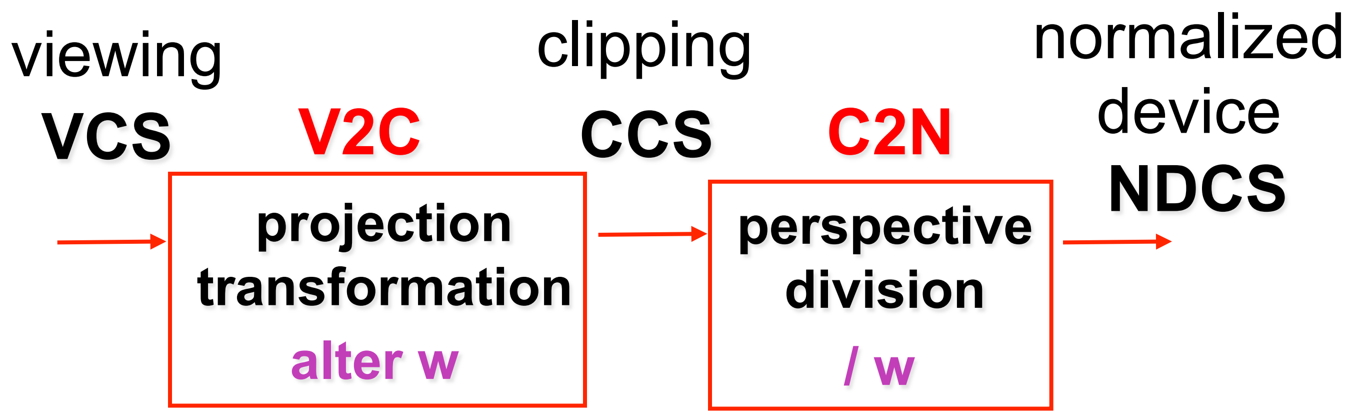
Review: Projection Normalization

- warp perspective view volume to orthogonal view volume
  - render all scenes with orthographic projection!
  - aka perspective warp





Review: Separate Warp From Homogenization

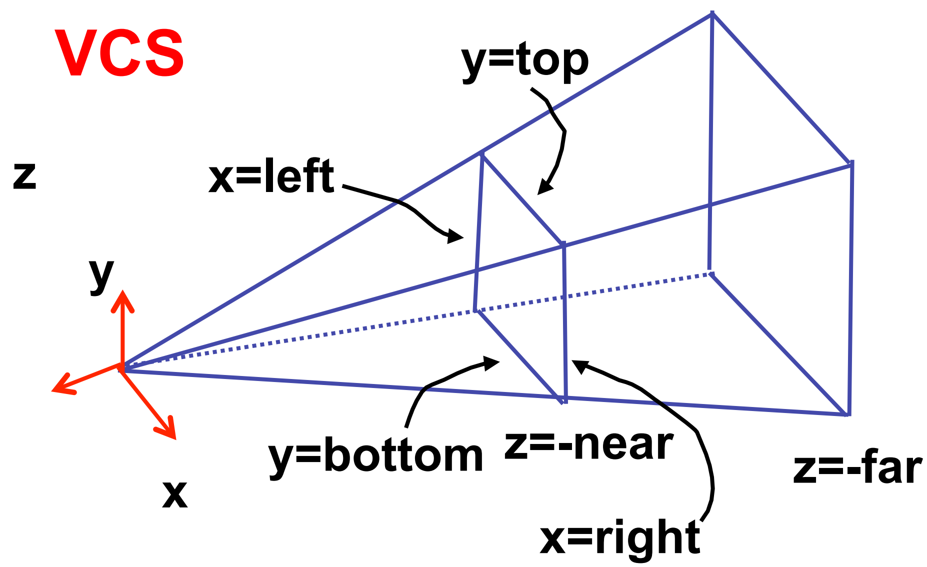


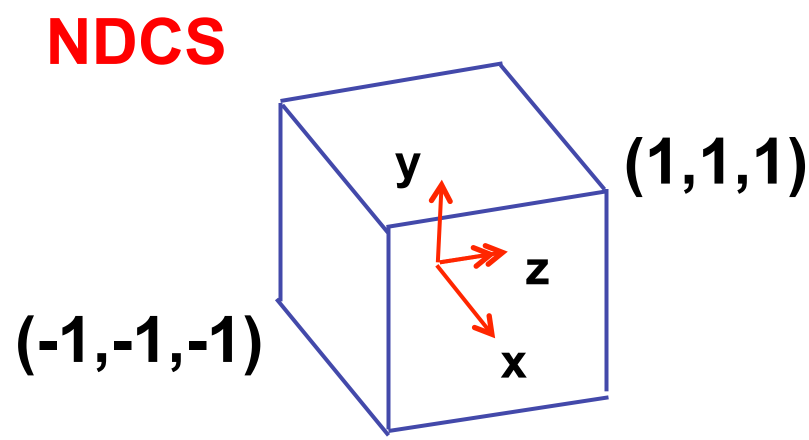
- warp requires only standard matrix multiply
  - distort such that orthographic projection of distorted objects is desired persp projection
    - w is changed
  - clip after warp, before divide
  - division by w: homogenization

Review: Perspective Derivation

- shear
  - change x/y if asymmetric r/l, t/b
- scale
- projection-normalization
  - pre-warp according to z

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



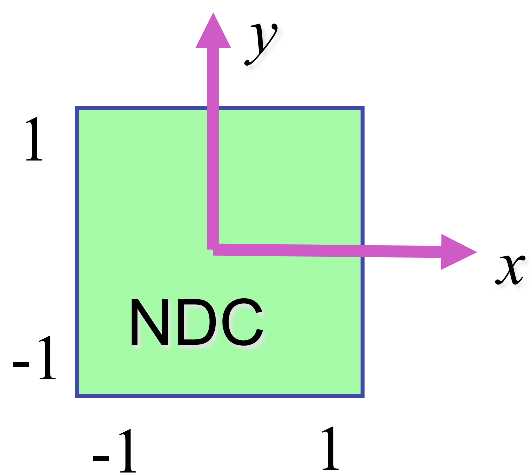


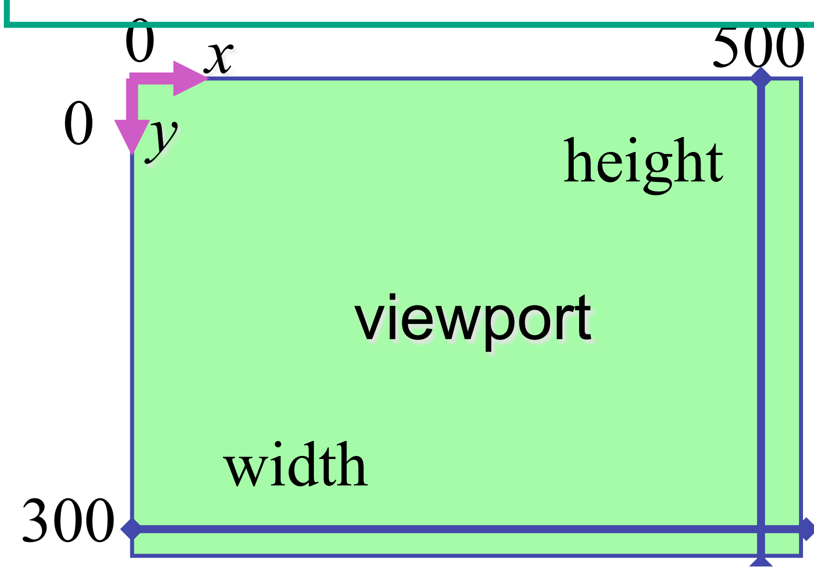
Review: N2D Transformation

$$
\begin{bmatrix} x_D \\ y_D \\ z_D \\ 1 \end{bmatrix} =
\begin{bmatrix} 1 & 0 & 0 & \frac{width}{2} - \frac{1}{2} \\ 0 & 1 & 0 & \frac{height}{2} - \frac{1}{2} \\ 0 & 0 & 1 & \frac{depth}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} \frac{width}{2} & 0 & 0 & 0 \\ 0 & \frac{height}{2} & 0 & 0 \\ 0 & 0 & \frac{depth}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} x_N \\ y_N \\ z_N \\ 1 \end{bmatrix} =
\begin{bmatrix} \frac{width(x_N + 1) - 1}{2} \\ \frac{height(-y_N + 1) - 1}{2} \\ \frac{depth(z_N + 1)}{2} \\ 1 \end{bmatrix}
$$

reminder:
NDC z range is -1 to 1

Display z range is 0 to 1. glDepthRange(n,f) can constrain further, but *depth* = 1 is both max and default





Review: Projective Rendering Pipeline

following pipeline from top/left to bottom/right: moving object POV

scene.add(x)

object **OCS** → **O2W** → **modeling transformation**

m.makeTranlation(x,y,z)
m.makeRotation(a,x,y,z)
x.setMatrix(m)

world **WCS** → **W2V** → **viewing transformation**

camera.lookAt(...)

viewing **VCS** → **V2C** → **projection transformation**

THREE.PerspectiveCamera(...)
alter w

clipping **CCS** → **C2N** (**/ w**) → **perspective division**

normalized device **NDCS** → **N2D** → **viewport transformation**

canvas.{w,h}
gl.viewport(x,y,a,b)

device **DCS**

OCS - object coordinate system

WCS - world coordinate system

VCS - viewing coordinate system

CCS - clipping coordinate system

NDCS - normalized device coordinate system

DCS - device coordinate system

Review: WebGL Example

go back from end of pipeline to beginning: coord frame POV!

CCS

```
gl.viewport(0,0,w,h);
```

VCS

```
THREE.PerspectiveCamera(view angle, aspect, near, far)
```

WCS

```
u_xformMatrix = Identity()
gl.uniformMatrix4fv(u_xformMatrix, false, xformMatrix);
```

OCS1

```
torsoGeometry.applyMatrix(u_xformMatrix );
var torso = new THREE.Mesh(torsoGeometry,normalMaterial);
scene.add(torso);
```

Review: Coord Sys: Frame vs Point

read down: transforming between coordinate frames, from frame A to frame B

read up: transforming points, up from frame B coords to frame A coords

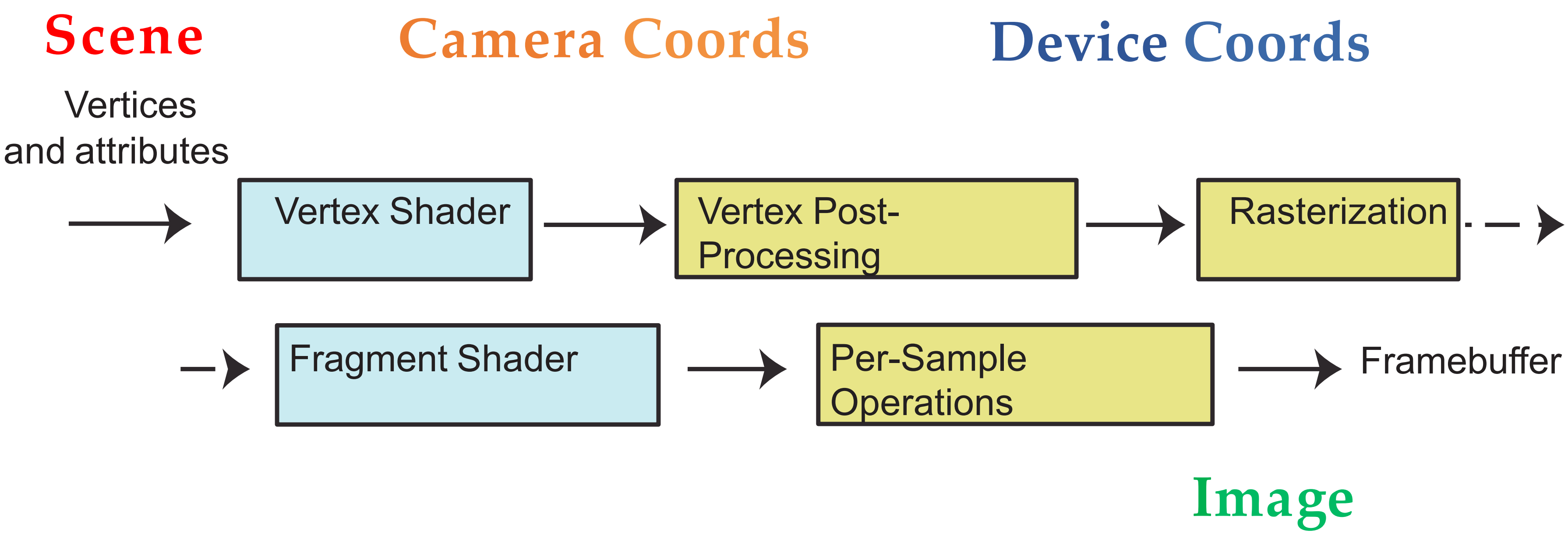
GL command order

GL command order (read down)	Coordinate system		Pipeline interpretation (read up)
	DCS	display	
D2N		gl.viewport(x,y,a,b)	**N2D**
	NDCS	normalized device	
N2V		THREE.PerspectiveCamera(...)	**V2N**
	VCS	viewing	
V2W		camera.lookAt(...)	**W2V**
	WCS	world	
W2O		m.makeRotationX(…)	**O2W**
	OCS	object	
		scene.add(object)	

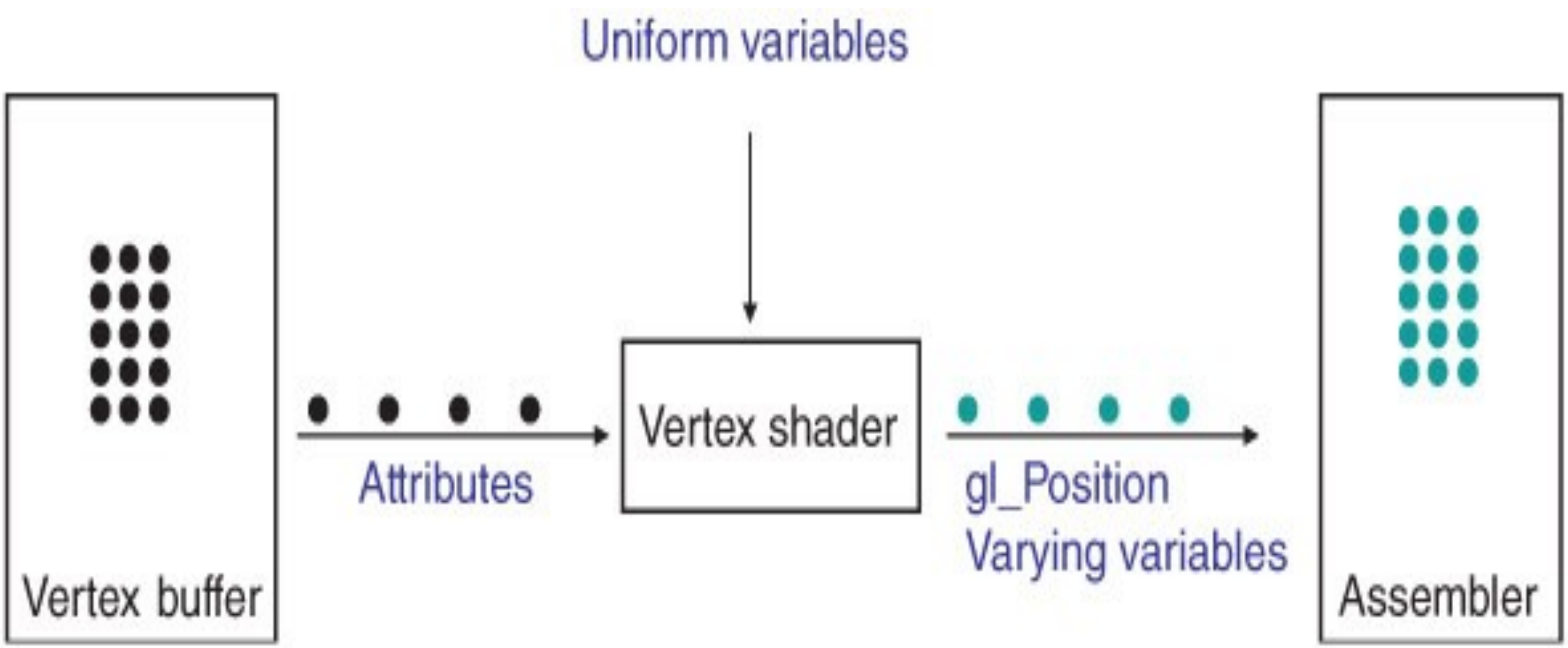
pipeline interpretation

Post-Midterm Material

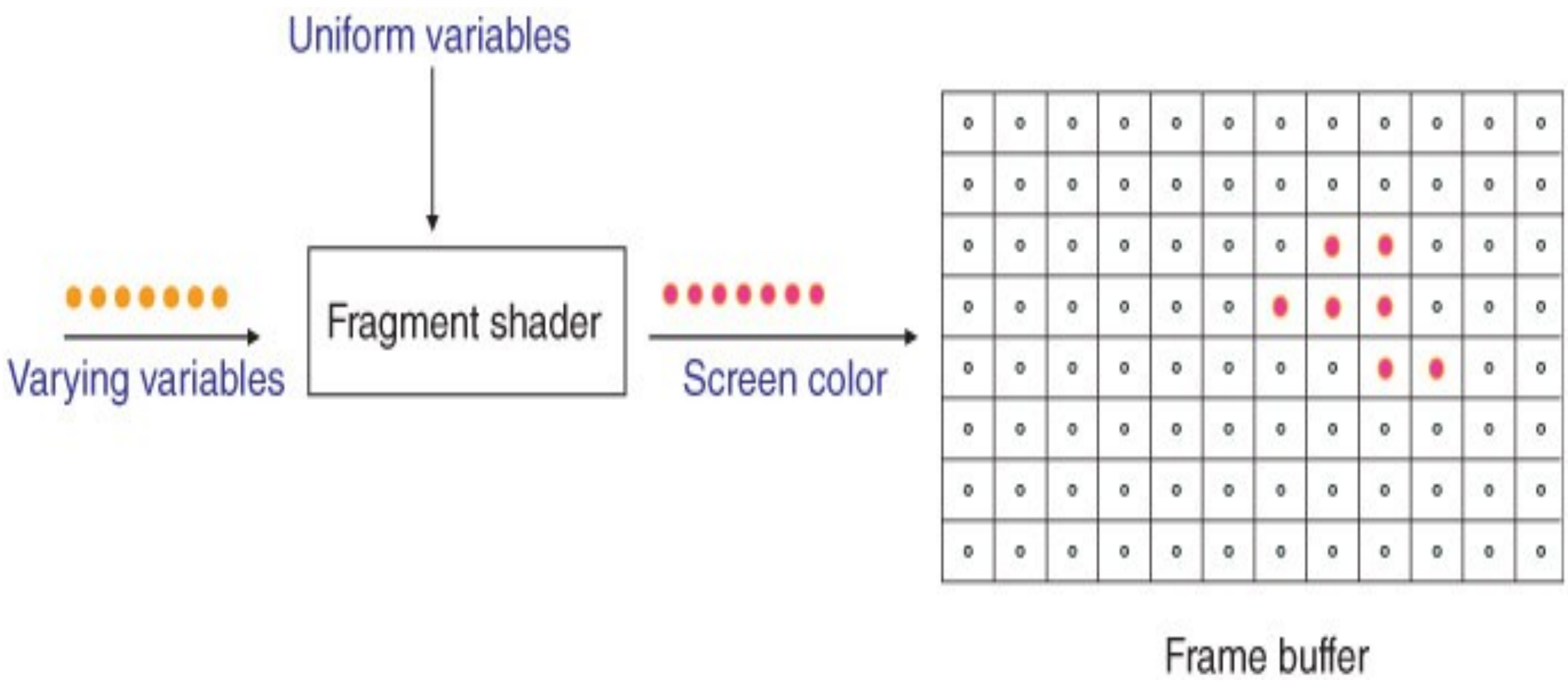
OPENGL RENDERING PIPELINE



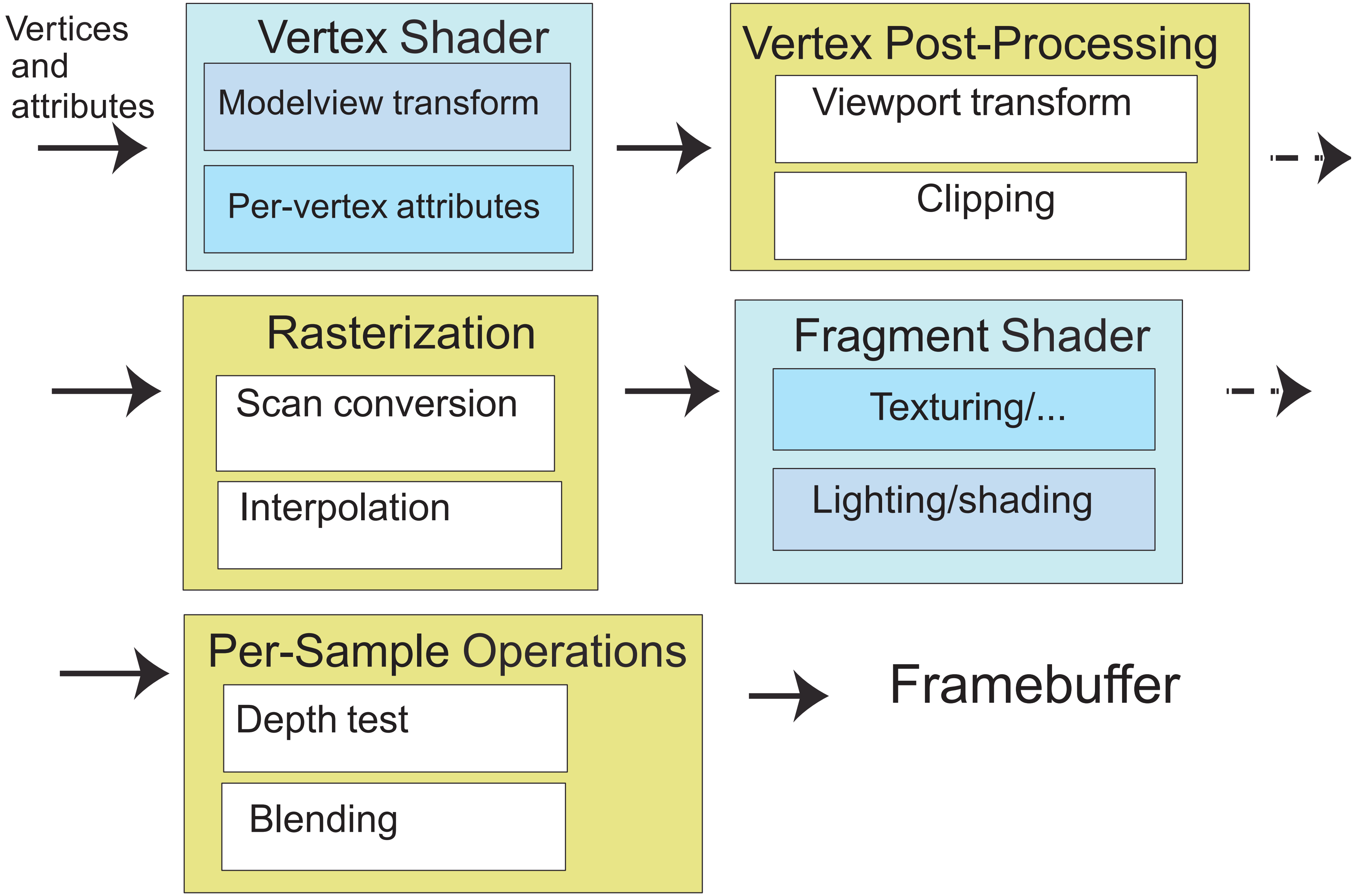
VERTEX SHADER



FRAGMENT SHADER



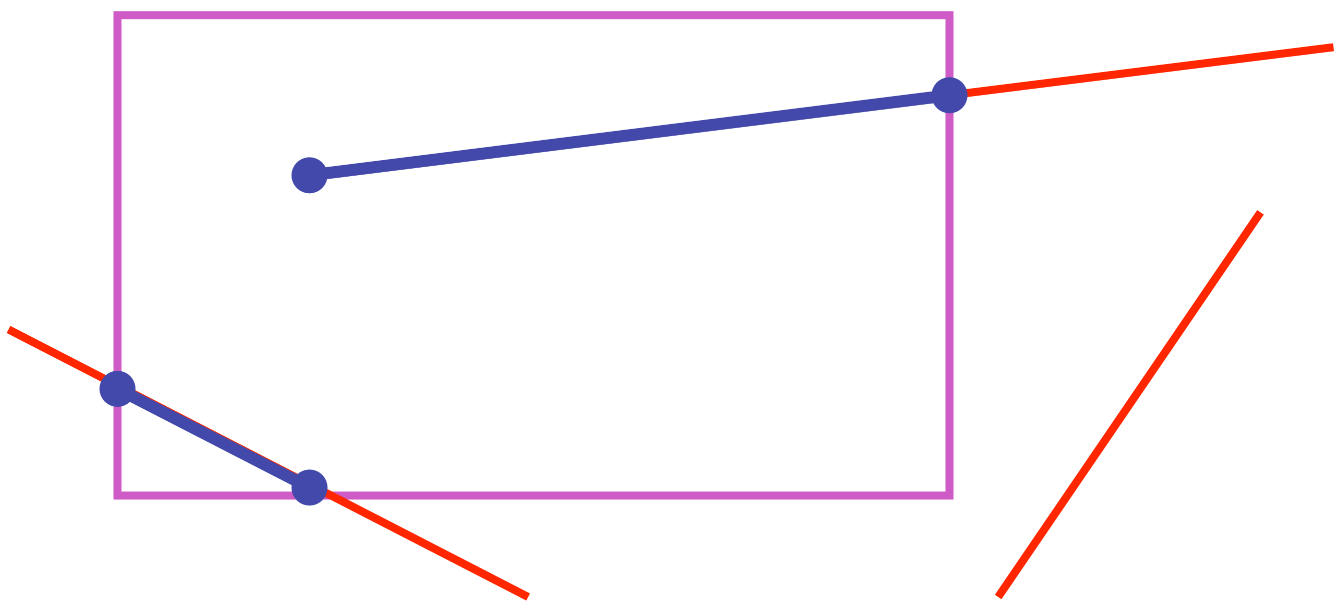
OPENGL RENDERING PIPELINE



Clipping/Rasterization/Interpolation

Review: Clipping

- analytically calculating the portions of primitives within the viewport



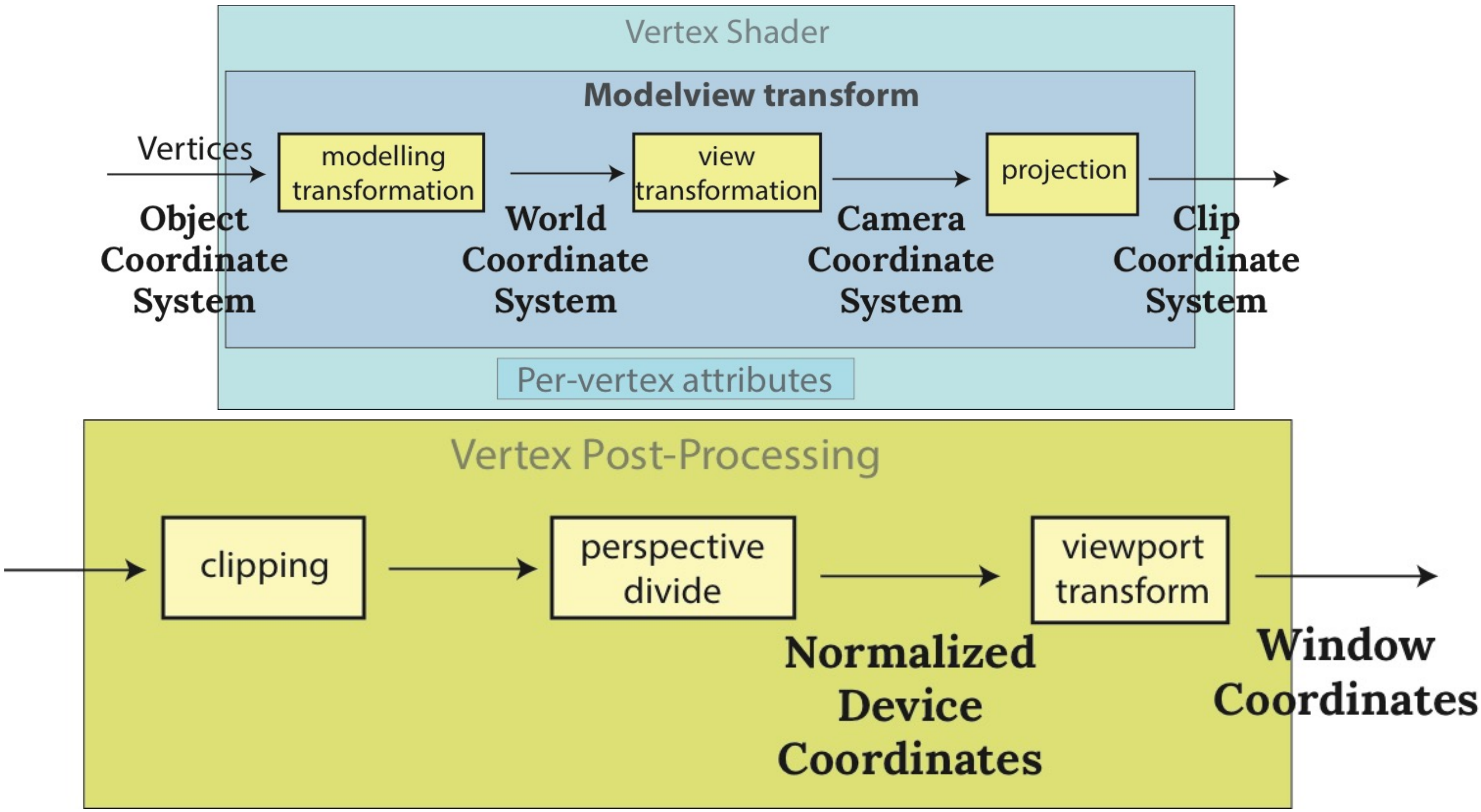
Review: Clipping

$$-w_c < x_c < w_c$$

$$-w_c < y_c < w_c$$

$$-w_c < z_c < w_c$$

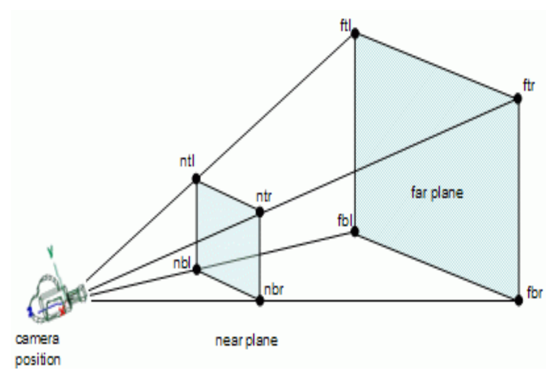
- Perform clipping in clip-coordinates!
  - After projection and before dividing by w



Review: Clipping coordinates

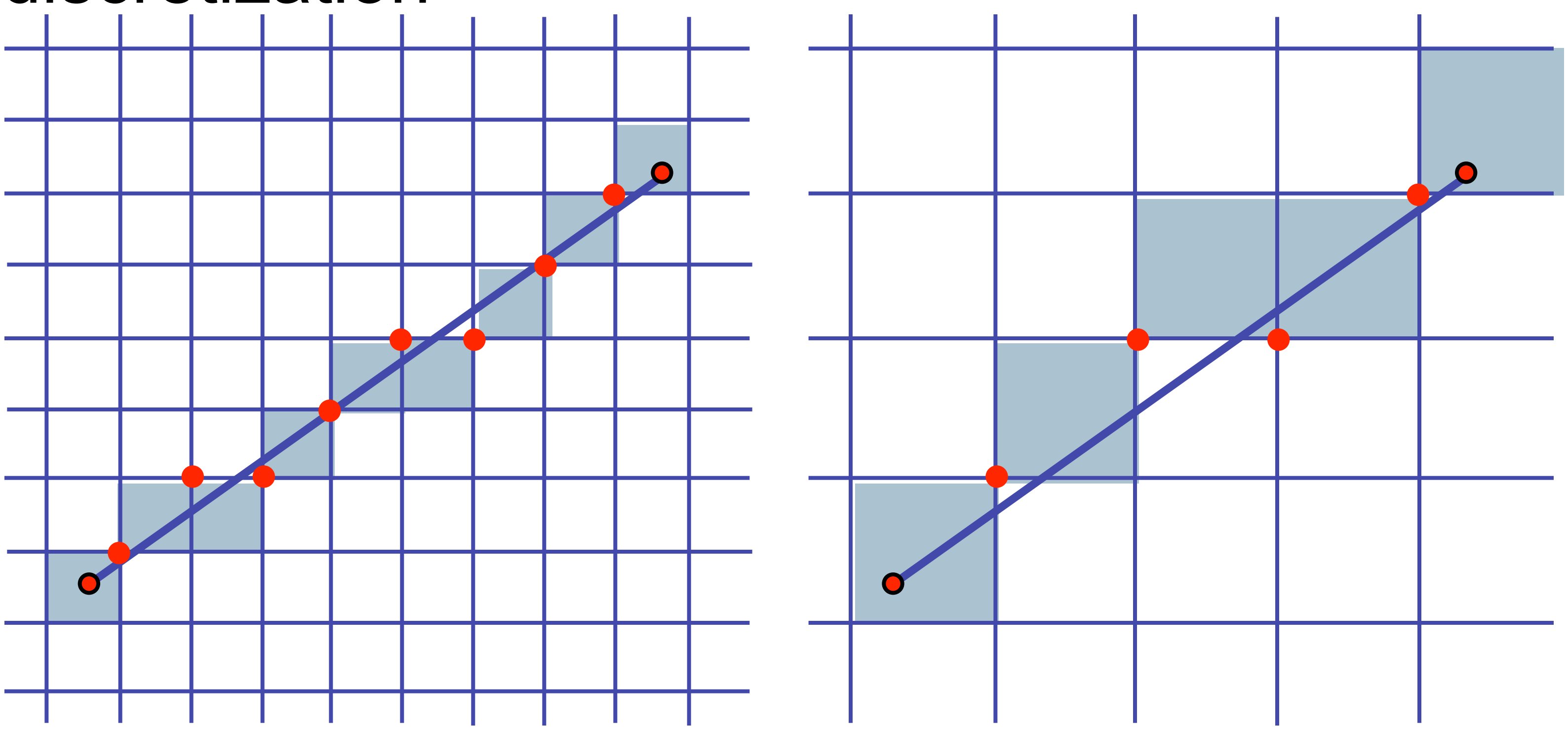
- Eye coordinates (projected) → clip coordinates → normalized device coordinates (NDCs)
- Dividing clip coordinates (x_c, y_c, z_c, w_c) by the $w_c (w_c = w_n)$ component (the fourth component in the homogeneous coordinates) yields normalized device coordinates (NDCs).

$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \begin{bmatrix} s_x & 0 & -c_x & 0 \\ 0 & s_y & -c_y & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_e \\ y_e \\ z_e \\ 1 \end{bmatrix}$$



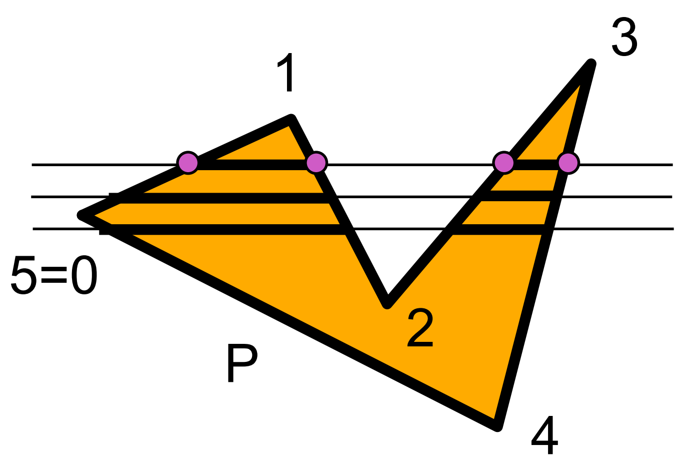
Review: Scan Conversion

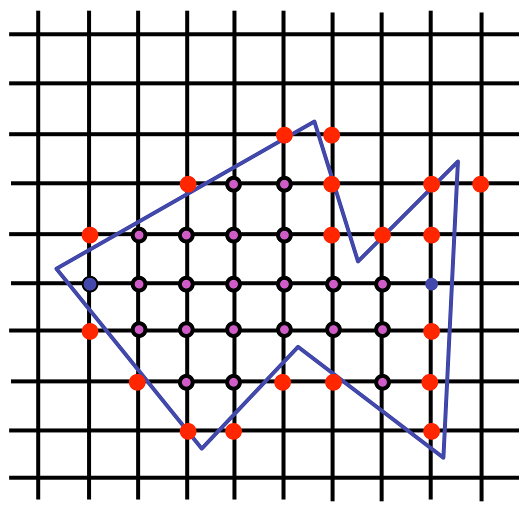
- convert continuous rendering primitives into discrete fragments/pixels
  - given vertices in DCS, fill in the pixels
- display coordinates required to provide scale for discretization

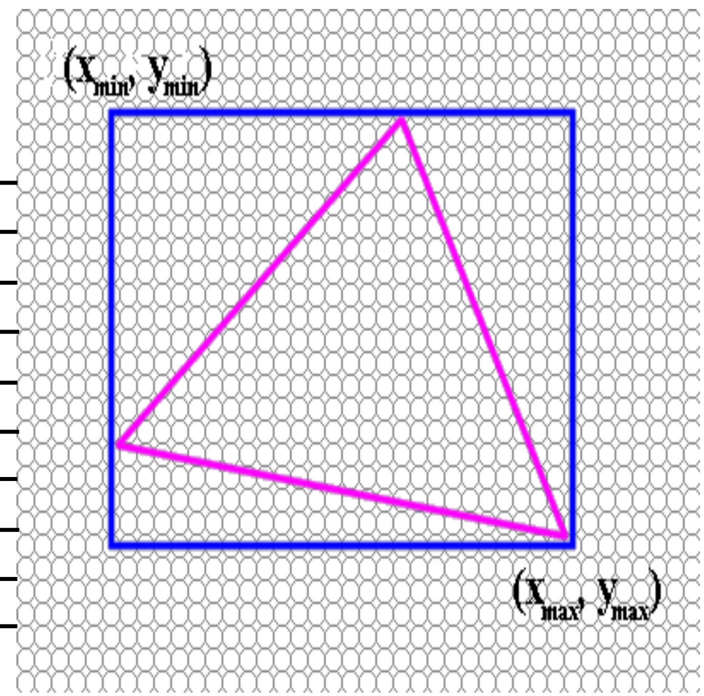


Review: Scanline Idea

- scanline: a line of pixels in an image
- basic structure of code:
  - Setup: compute edge equations, bounding box
  - (Outer loop) For each scanline in bounding box...
  - (Inner loop) …check each pixel on scanline, evaluating edge equations and drawing the pixel if all three are positive

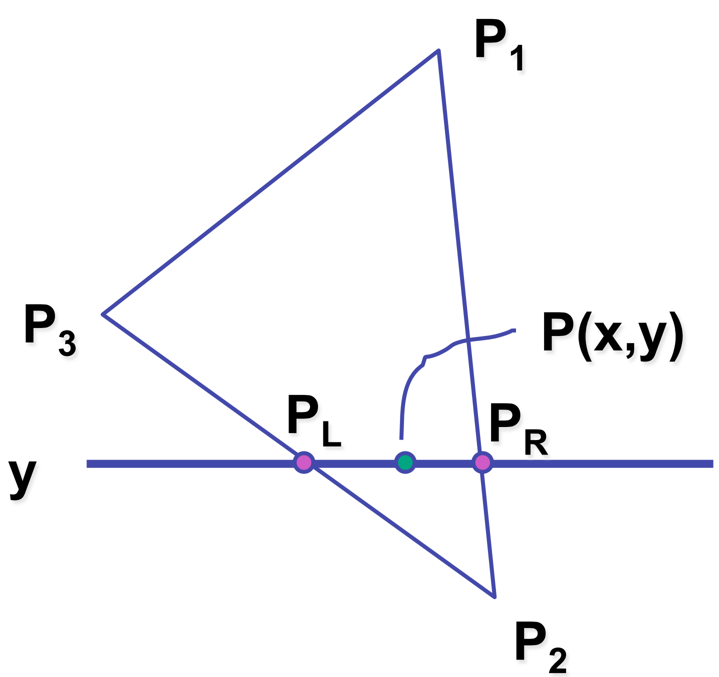




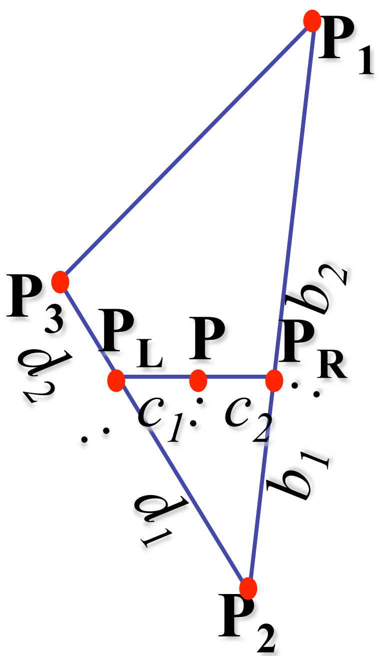


Review: Bilinear Interpolation

- interpolate quantity along L and R edges, as a function of y
    - then interpolate quantity as a function of x



Review: Bilinear interpolation



$$P = \frac{c_2}{c_1 + c_2} \cdot P_L + \frac{c_1}{c_1 + c_2} \cdot P_R$$

$$P_L = \frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3$$

$$P_R = \frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1$$

$$P = \frac{c_2}{c_1 + c_2}\left(\frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3\right) + \frac{c_1}{c_1 + c_2}\left(\frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1\right)$$

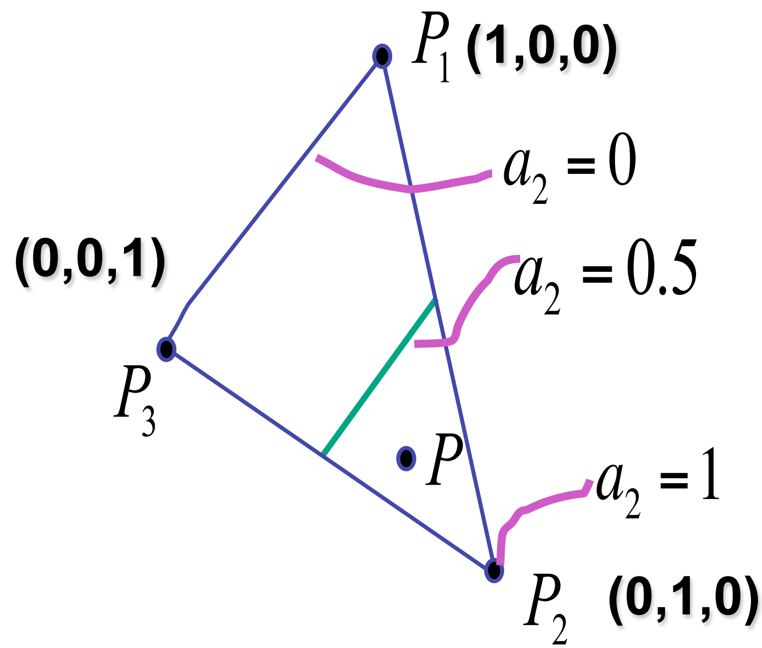
Review: Barycentric Coordinates

- weighted (affine) combination of vertices

$$P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3$$

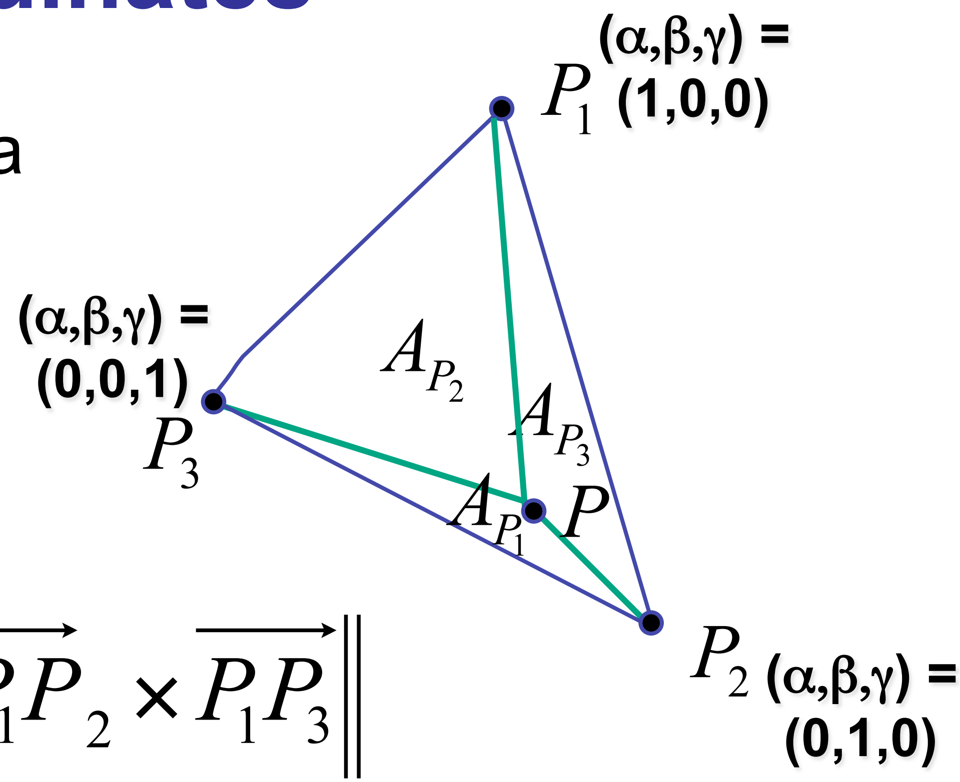
$$\alpha + \beta + \gamma = 1$$

$$0 \leq \alpha, \beta, \gamma \leq 1$$



Review: Computing Barycentric Coordinates

- 2D triangle area
  - half of parallelogram area
    - from cross product



$A = A_{P1} + A_{P2} + A_{P3}$

$\alpha = A_{P1} / A$

$$A = \frac{1}{2} \left\| \overrightarrow{P_1 P_2} \times \overrightarrow{P_1 P_3} \right\|$$

$\beta = A_{P2} / A$

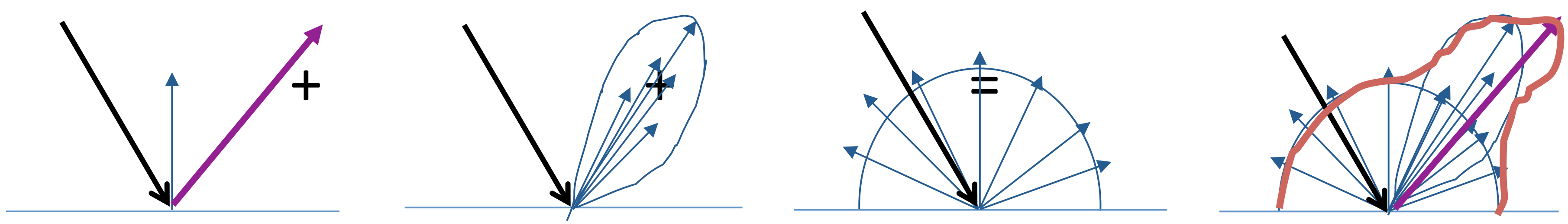
weighted combination of three points

$\gamma = A_{P3} / A$

Lighting/Shading

Review: Reflectance

- *specular*: perfect mirror with no scattering
- *gloss*: mixed, partial specularity
- *diffuse*: all directions with equal energy

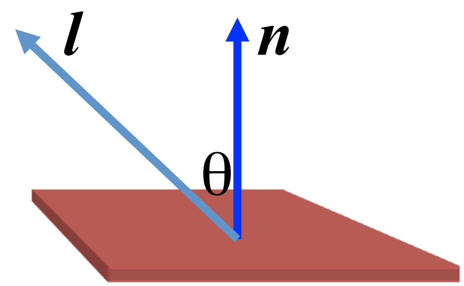


specular + glossy + diffuse =

reflectance distribution

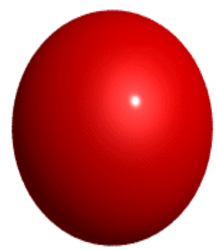
Review: Reflection Equations

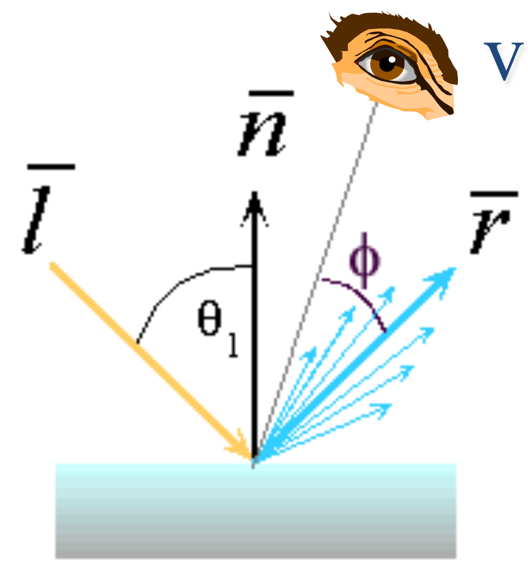
$$I_{diffuse} = k_d\, I_{light}\, (n \bullet l)$$



$$I_{specular} = k_s I_{light} (v \bullet r)^{n_{shiny}}$$

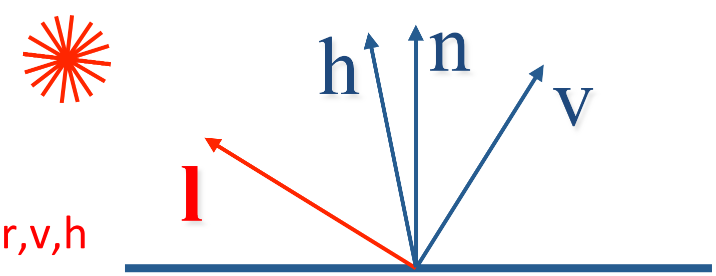
$$R = 2\,(N\,(N \cdot L)) - L$$





$$I_{specular} = k_s I_{light} (h \bullet n)^{n_{shiny}}$$

$$h = (l + v)/2$$



reminder: normalize all vectors: n,l,r,v,h

Review: Reflection Equations

full Phong lighting model

– combine ambient, diffuse, specular components

$$\mathbf{I}_{\mathbf{total}} = \mathbf{k}_{\mathbf{a}}\mathbf{I}_{\mathbf{ambient}} + \sum_{i=1}^{\#lights} \mathbf{I}_{\mathbf{i}}(\mathbf{k}_{\mathbf{d}}(\mathbf{n} \bullet \mathbf{l}_{\mathbf{i}}) + \mathbf{k}_{\mathbf{s}}(\mathbf{v} \bullet \mathbf{r}_{\mathbf{i}})^{n_{shiny}})$$

- Blinn-Phong lighting

$$\mathbf{I}_{\mathbf{total}} = \mathbf{k}_{\mathbf{a}}\mathbf{I}_{\mathbf{ambient}} + \sum_{i=1}^{\#lights} \mathbf{I}_{\mathbf{i}}(\mathbf{k}_{\mathbf{d}}(\mathbf{n} \bullet \mathbf{l}_{\mathbf{i}}) + \mathbf{k}_{\mathbf{s}}(\mathbf{h} \bullet \mathbf{n}_{\mathbf{i}})^{n_{shiny}})$$

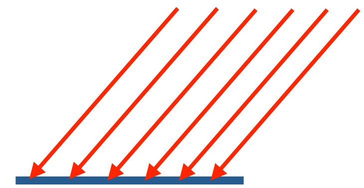
– **don't forget to normalize all lighting vectors!! n,l,r,v,h**

Review: Lighting

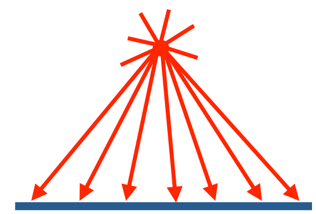
- lighting models
  - ambient
    - normals don't matter
  - Lambert/diffuse
    - angle between surface normal and light
  - Phong/specular
    - surface normal, light, and viewpoint
- light and material interaction
  - component-wise multiply
    - $(l_r,l_g,l_b) \times (m_r,m_g,m_b) = (l_r{*}m_r,\ l_g{*}m_g,\ l_b{*}m_b)$

Review: Light Sources

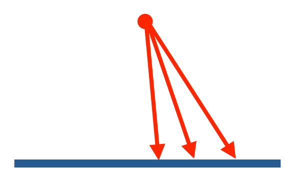
- directional/parallel lights
  - point at infinity: $(x,y,z,0)^T$



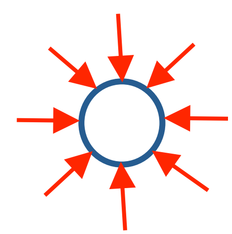
- point lights
  - finite position: $(x,y,z,1)^T$



- spotlights
  - position, direction, angle



- ambient lights

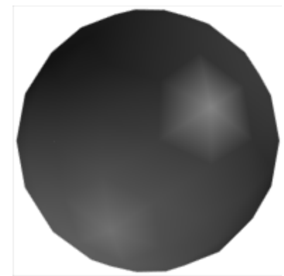


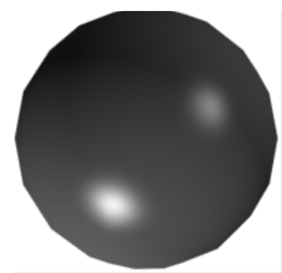
Review: Light Source Placement

- geometry: positions and directions
  - standard: world coordinate system
    - effect: lights fixed wrt world geometry
  - alternative: camera coordinate system
    - effect: lights attached to camera (car headlights)

Review: Shading Models

- flat shading
  - for each polygon
    - compute Phong lighting just once
- Gouraud shading
  - compute Phong lighting at the vertices
  - for each pixel in polygon, interpolate colors
- Phong shading
  - for each pixel in polygon
    - interpolate normal
    - compute Phong lighting

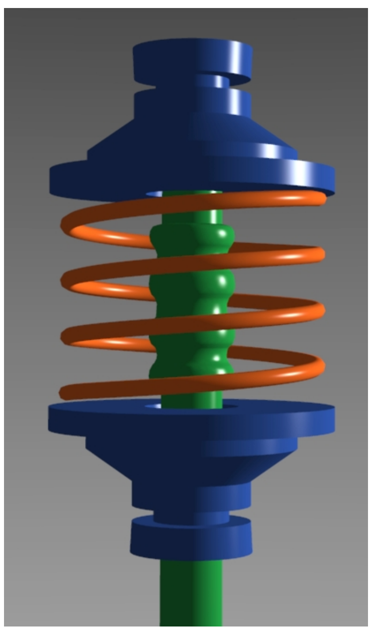




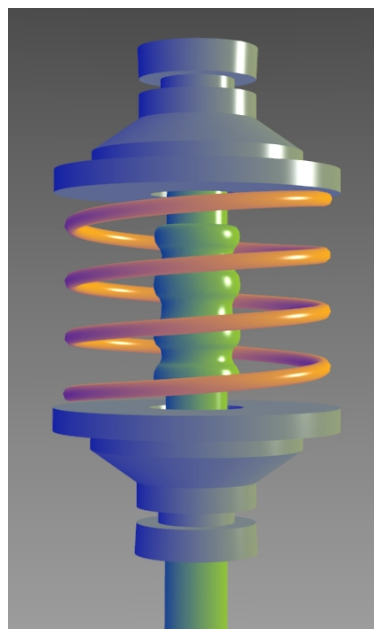
Review: Non-Photorealistic Shading

- cool-to-warm shading: $k_w = \frac{1 + \mathbf{n} \cdot \mathbf{l}}{2}, c = k_w c_w + (1 - k_w) c_c$
- draw silhouettes: if $(\mathbf{e} \cdot \mathbf{n_0})(\mathbf{e} \cdot \mathbf{n_1}) \leq 0$,**e**=edge-eye vector
- draw creases: if $(\mathbf{n_0} \cdot \mathbf{n_1}) \leq threshold$

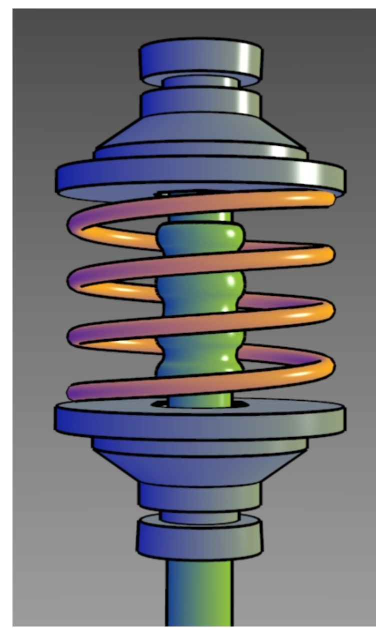
standard



cool-to-warm



with edges/creases

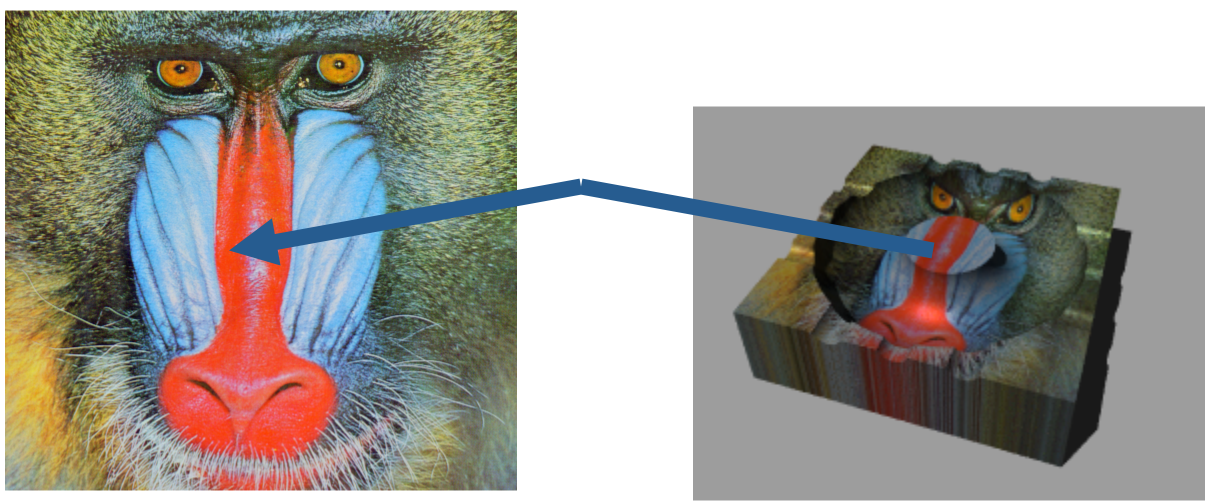


http://www.cs.utah.edu/~gooch/SIG98/paper/drawing.html

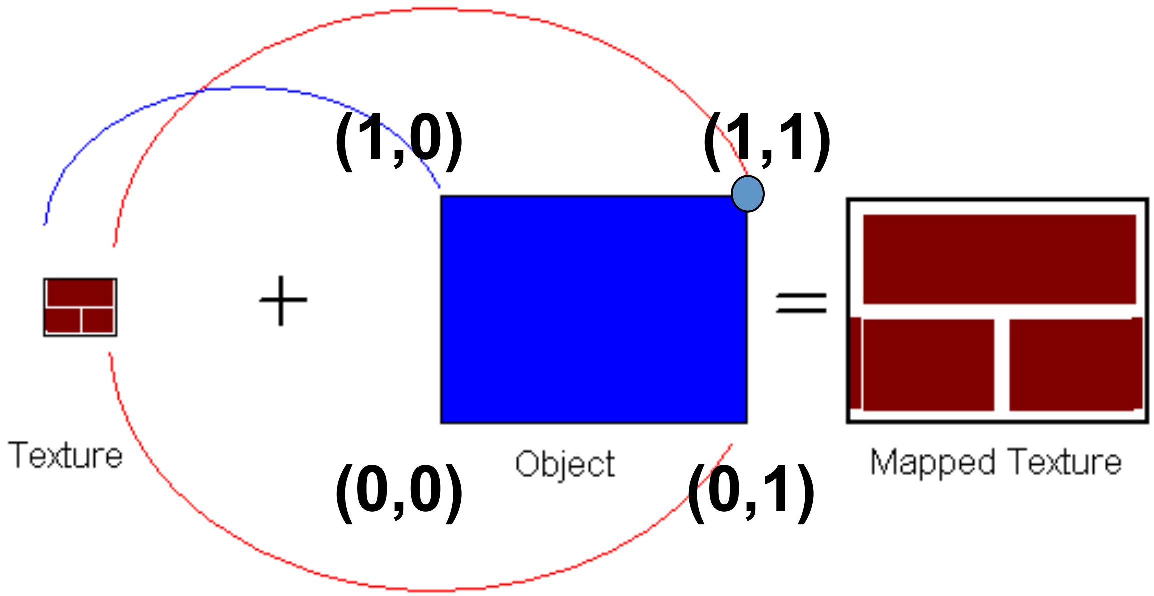
Texturing

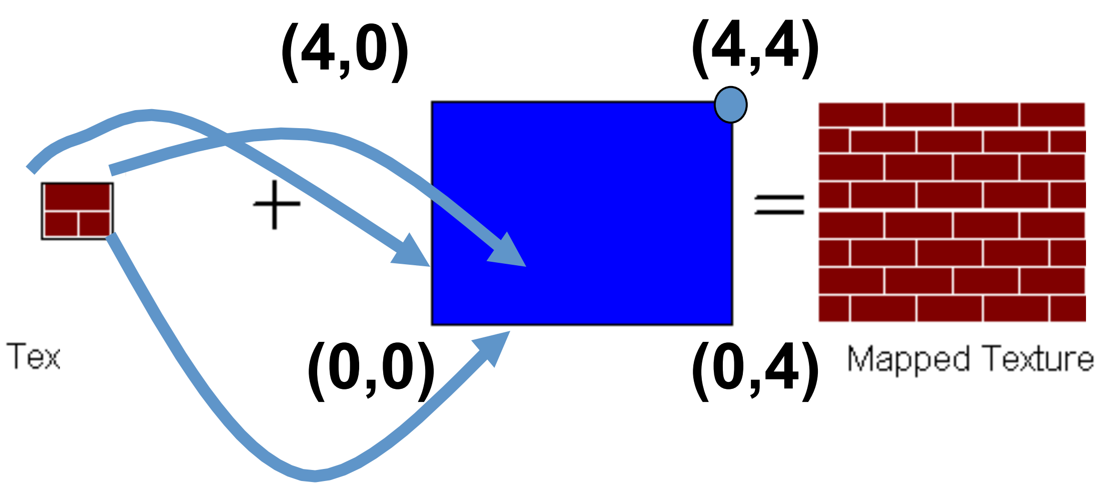
Review: Texture Coordinates

- texture image: 2D array of color values (texels)
- assigning texture coordinates (u,v) at vertex with object coordinates (x,y,z,w)
  - sometimes called (s,t) instead of (u,v)
  - use interpolated (u,v) for texel lookup at each pixel
  - use value to modify a polygon color or other property
  - specified by programmer or artist

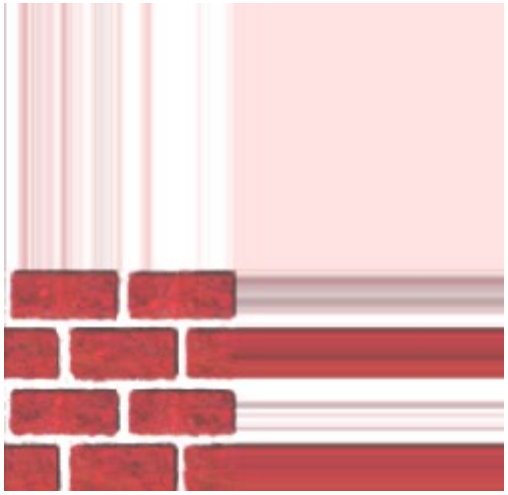


Review: Tiled Texture Map



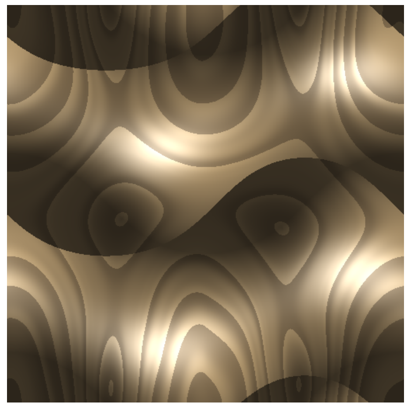


clamp vs repeat

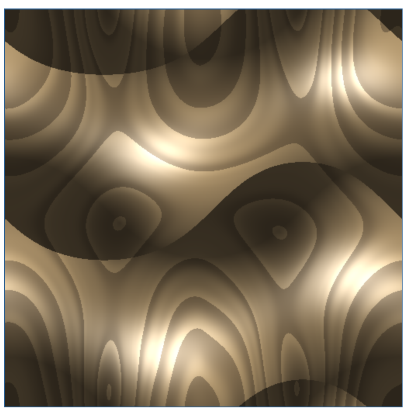


Review: Fractional Texture Coordinates

texture image

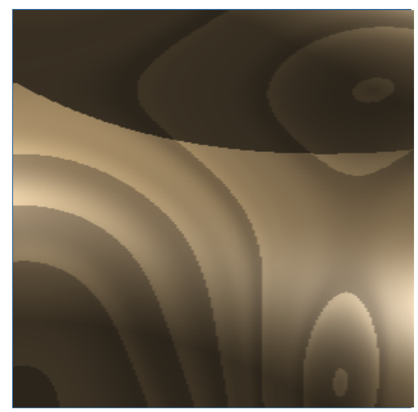


(0,1) **(1,1)**



(0,0) **(1,0)**

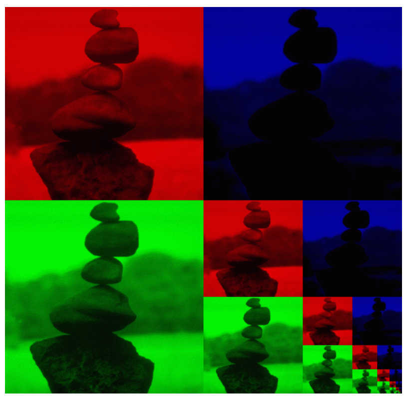
(0,.5) **(.25,.5)**

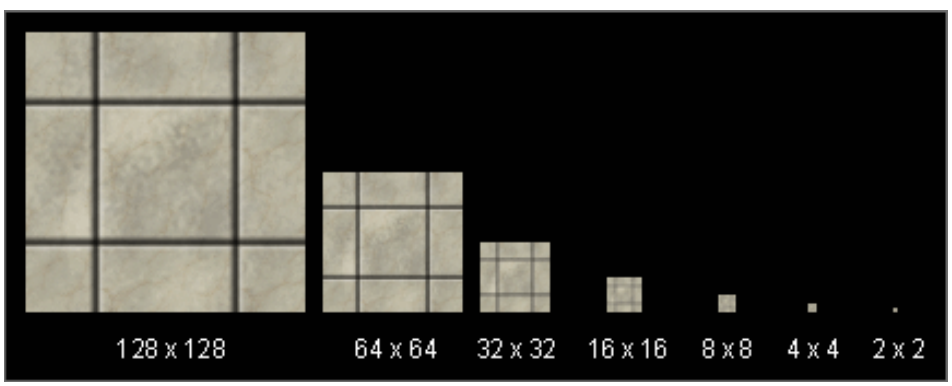


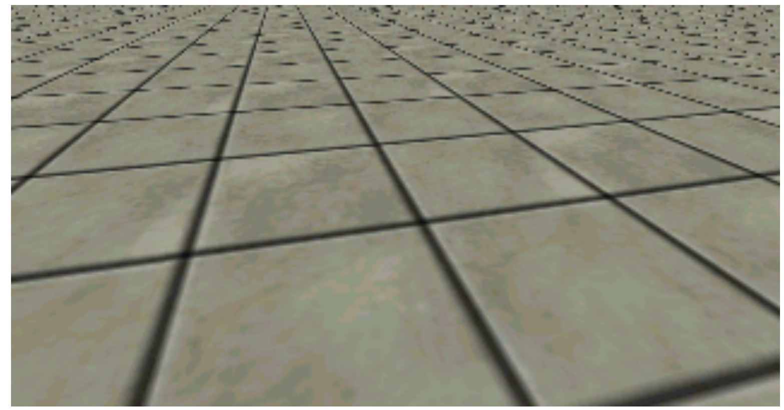
(0,0) **(.25,0)**

Review: MIPmapping

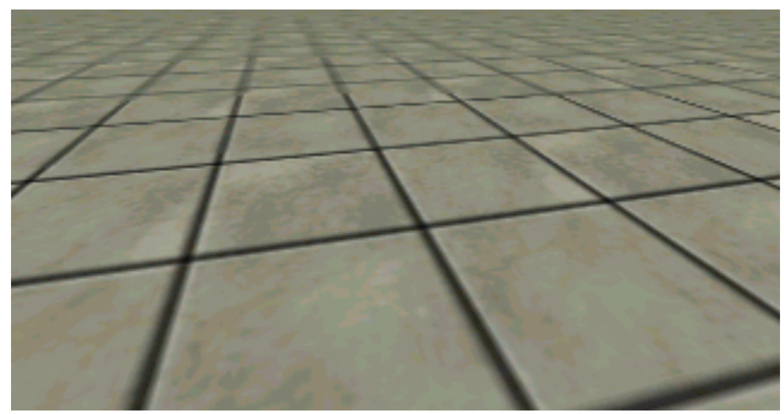
- image pyramid, precompute averaged versions
  - avoid aliasing artifacts
  - only requires 1/3 more storage







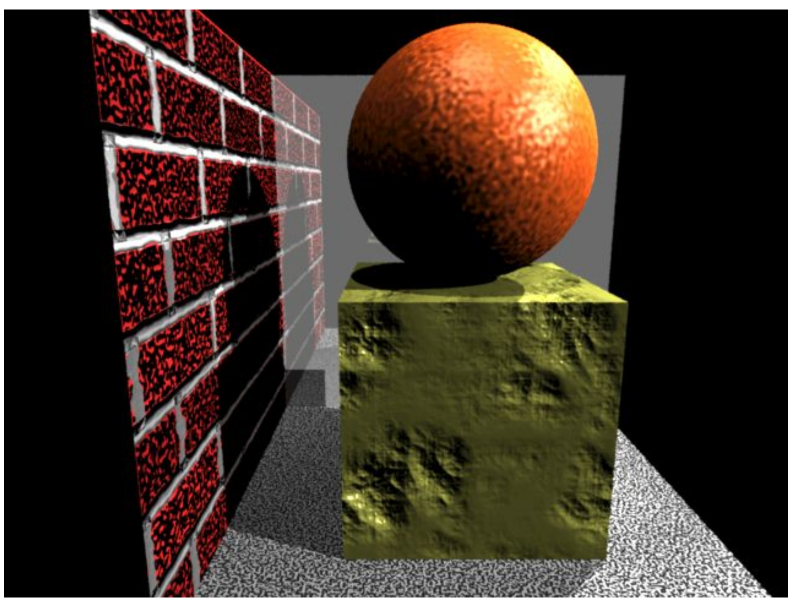
Without MIP-mapping

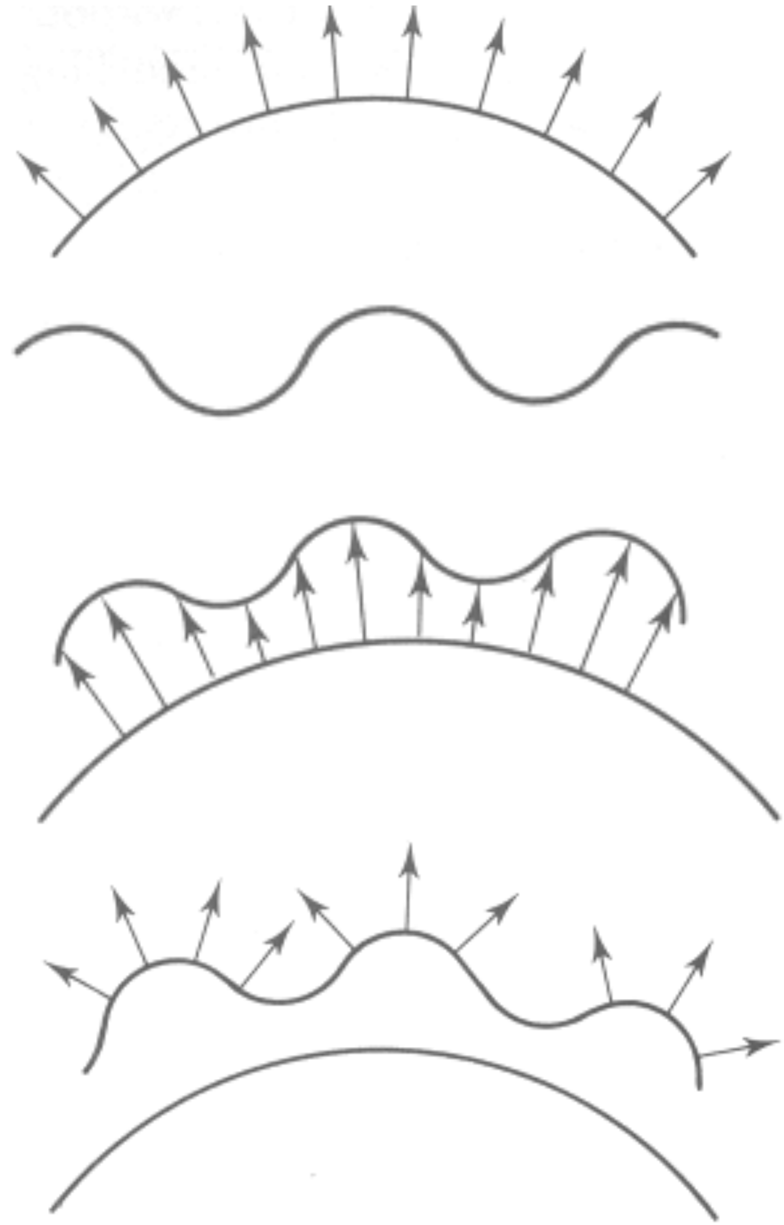


With MIP-mapping

Review: Bump Mapping: Normals As Texture

- create illusion of complex geometry model
- control shape effect by locally perturbing surface normal

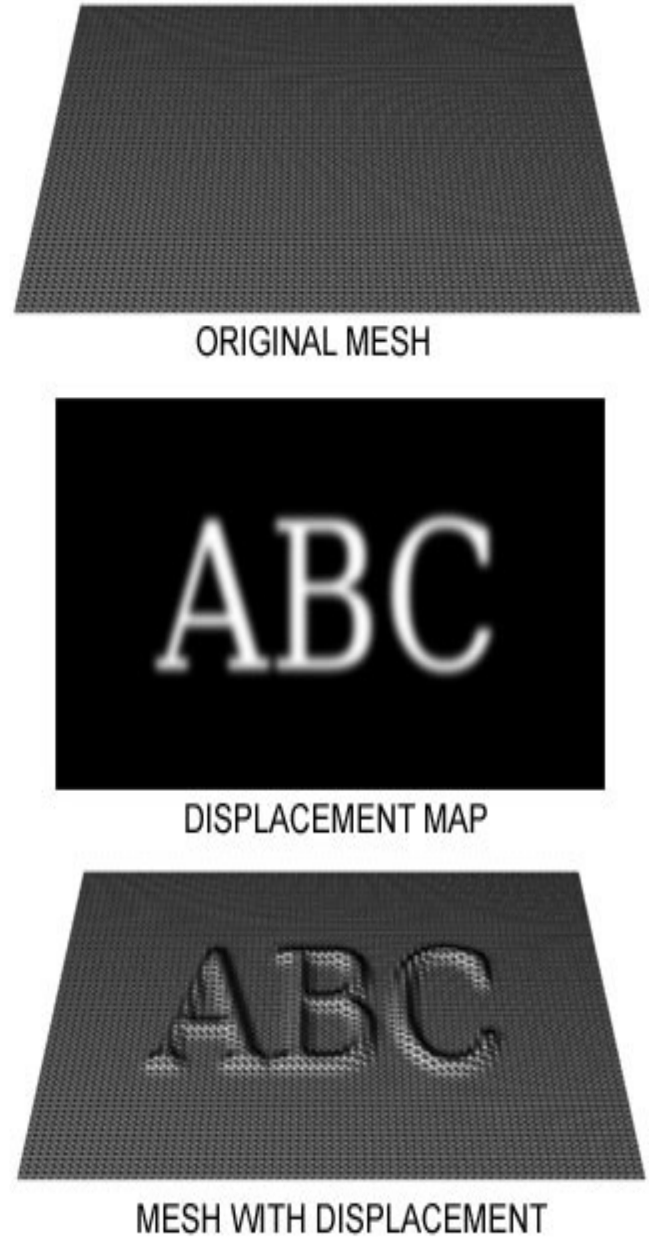




Review: Displacement Mapping

- bump mapping gets silhouettes wrong
  - shadows wrong too

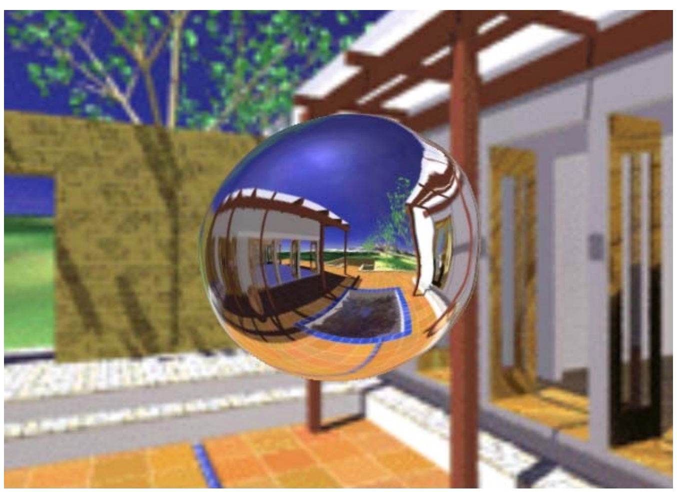
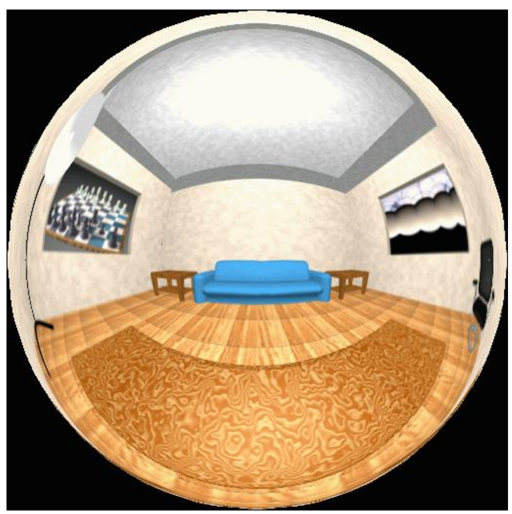
- change surface geometry instead
  - only recently available with realtime graphics
  - need to subdivide surface



https://en.wikipedia.org/wiki/Displacement_map ping#/media/File:Displacement.jpg

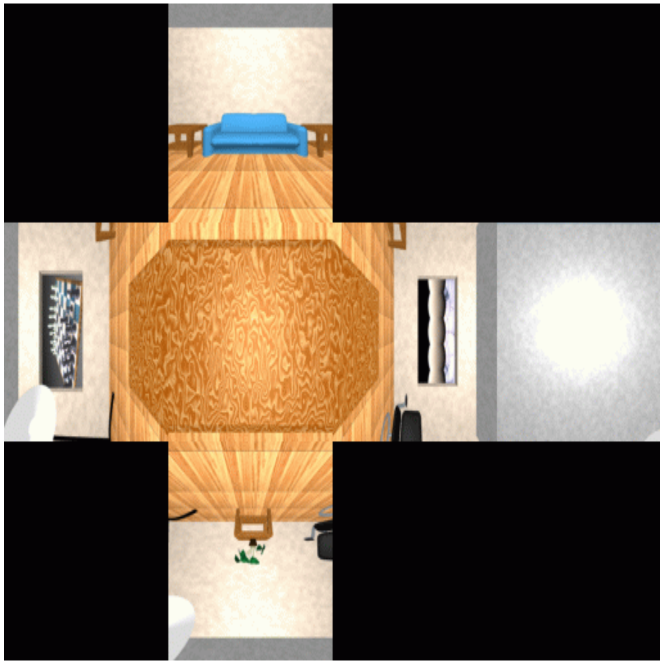
Review: Environment Mapping

- cheap way to achieve reflective effect
  - generate image of surrounding
  - map to object as texture
- sphere mapping: texture is distorted fisheye view
  - point camera at mirrored sphere
  - use spherical texture coordinates

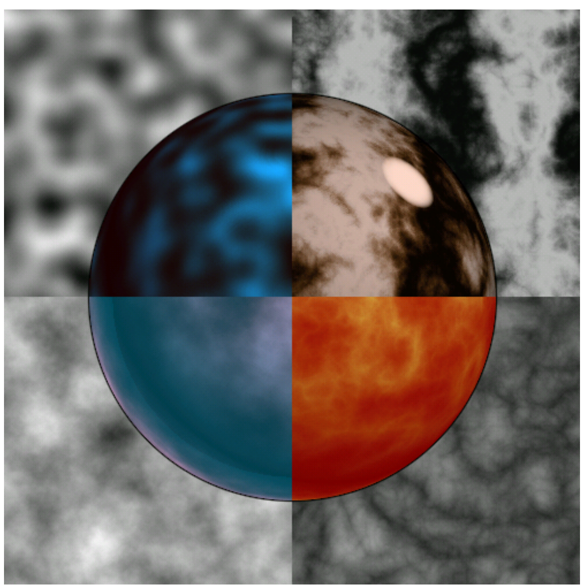
Review: Environment Cube Mapping

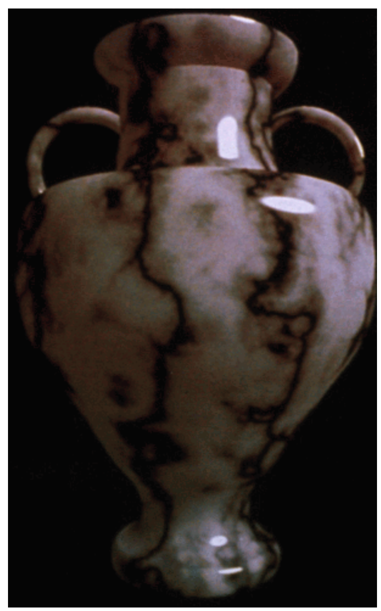
- 6 planar textures, sides of cube
  - point camera in 6 different directions, facing out from origin

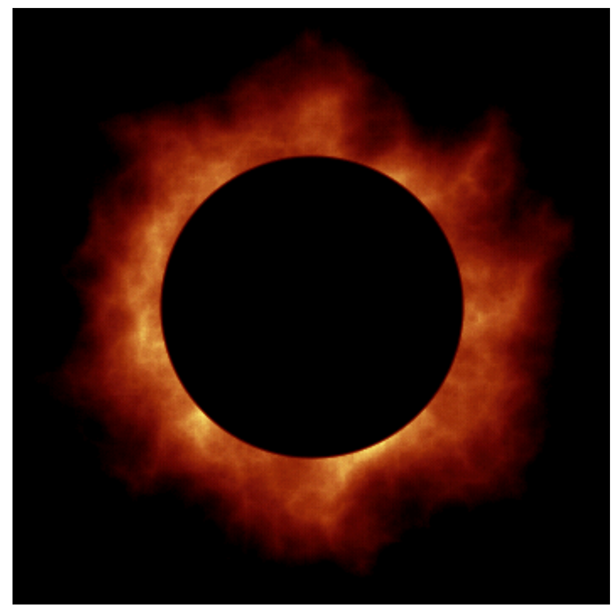


Review: Perlin Noise as Procedural Texture

- several good explanations
  - http://www.noisemachine.com/talk1
  - http://freespace.virgin.net/hugo.elias/models/m_perlin.htm
  - http://www.robo-murito.net/code/perlin-noise-math-faq.html



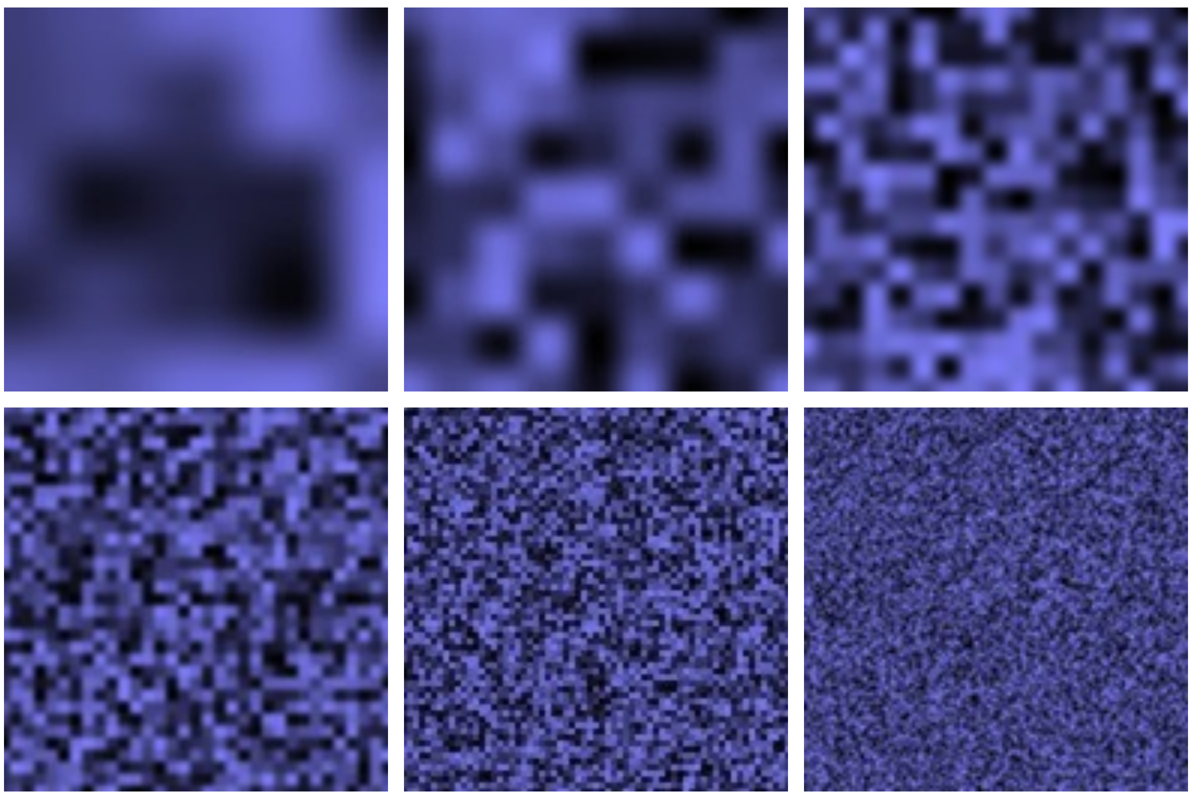


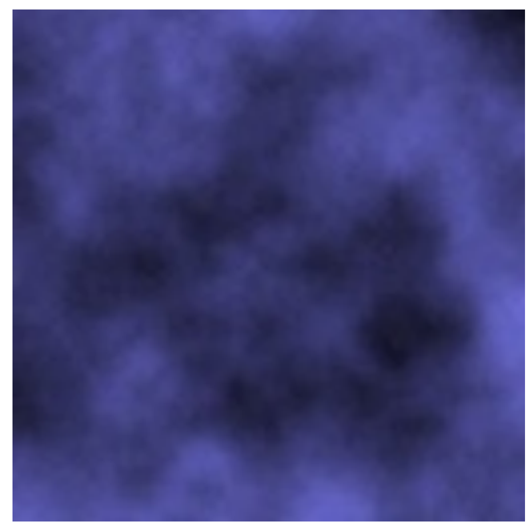


http://mrl.nyu.edu/~perlin/planet/

Review: Perlin Noise

- coherency: smooth not abrupt changes
- turbulence: multiple feature sizes

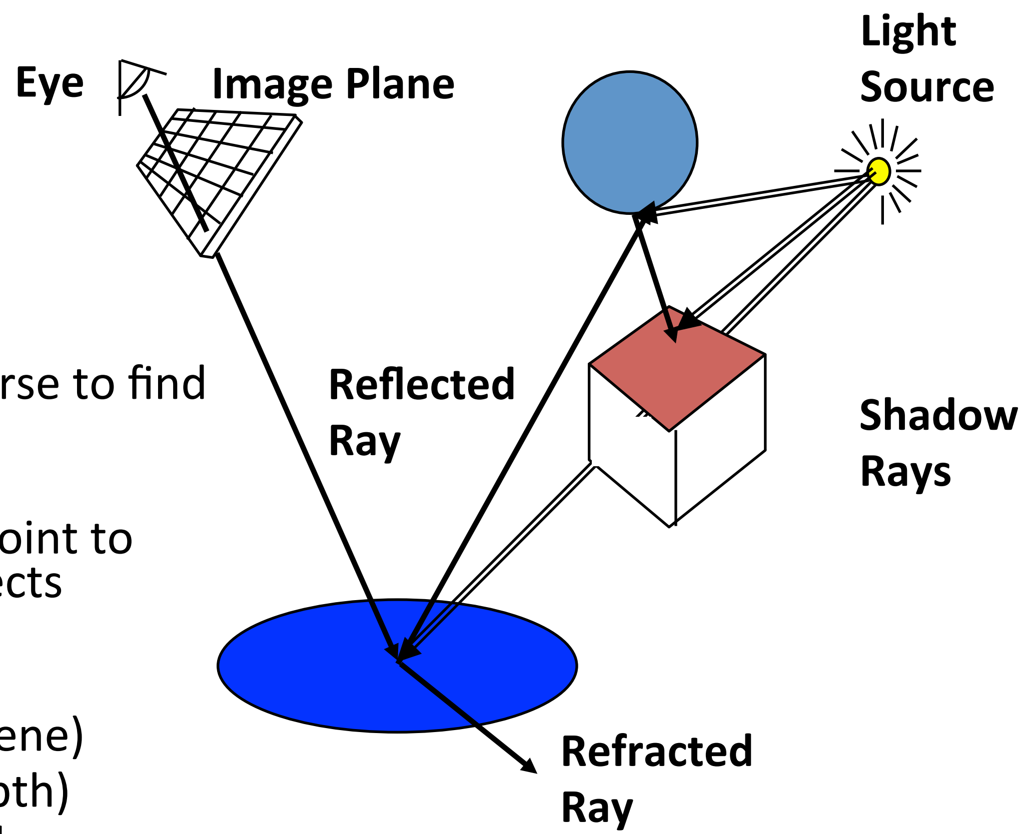




Ray Tracing

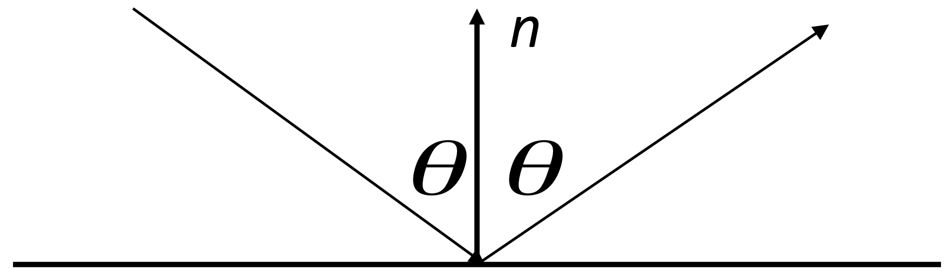
Review: Recursive Ray Tracing

- ray tracing can handle
  - reflection (chrome/mirror)
  - refraction (glass)
  - shadows
- one primary ray per pixel
- spawn secondary rays
  - reflection, refraction
    - if another object is hit, recurse to find its color
  - shadow
    - cast ray from intersection point to light source, check if intersects another object
  - termination criteria
    - no intersection (ray exits scene)
    - max bounces (recursion depth)
    - attenuated below threshold



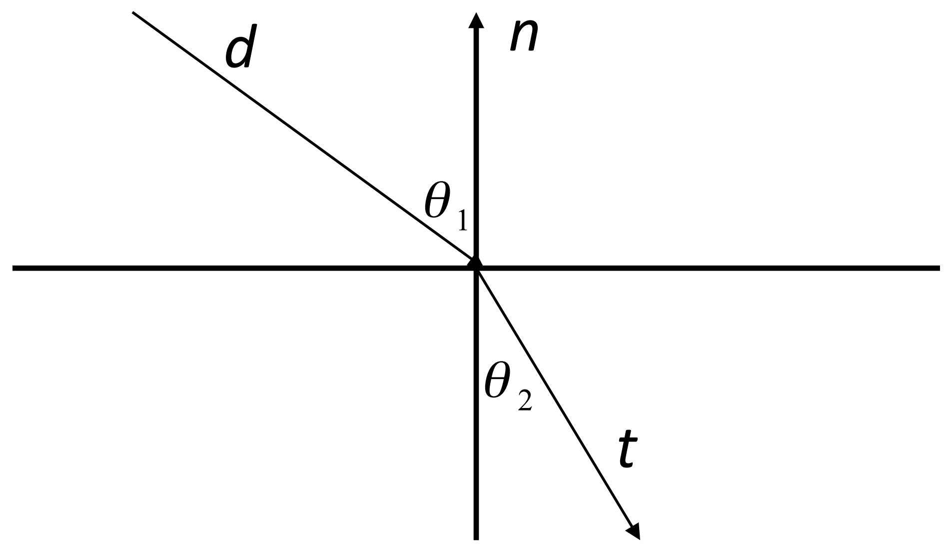
Review: Reflection and Refraction

- reflection: mirror effects
  - perfect specular reflection



- refraction: at boundary
- Snell's Law
  - light ray bends based on refractive indices c_1, c_2

$$c_1 \sin\theta_1 = c_2 \sin\theta_2$$



Review: Ray Tracing

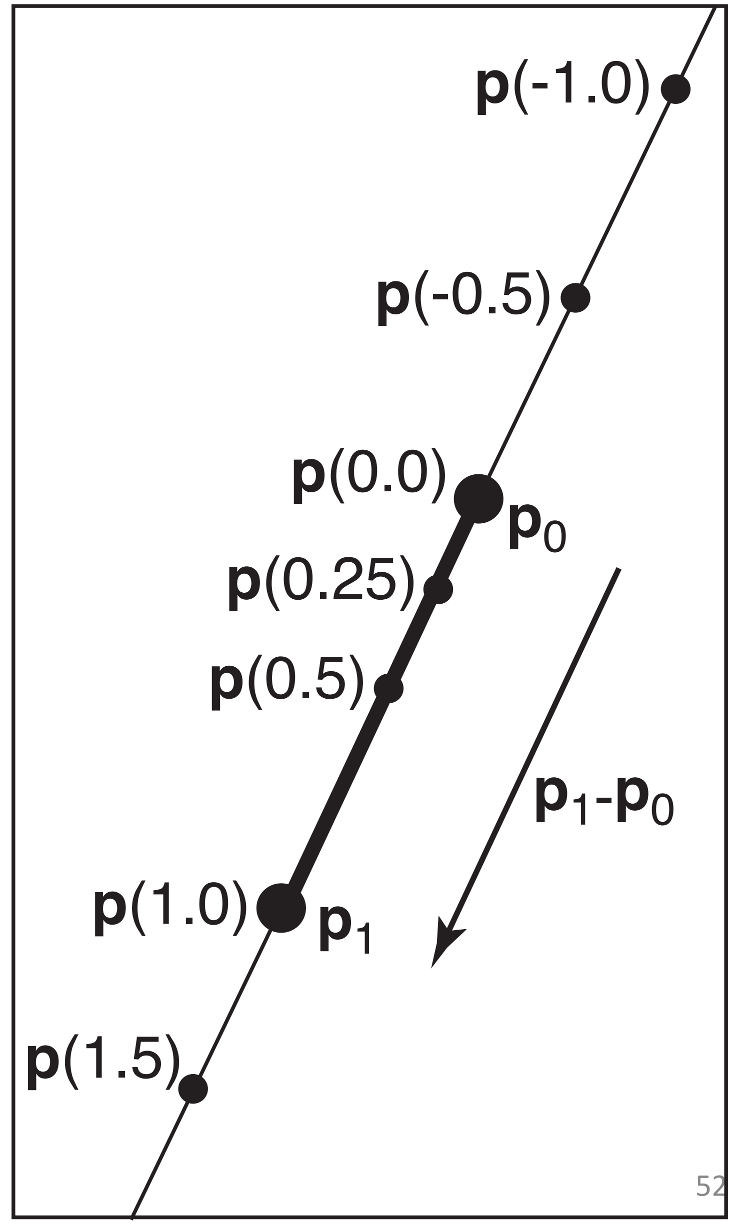
- issues:
  - generation of rays
  - intersection of rays with geometric primitives
  - geometric transformations
  - lighting and shading
  - efficient data structures so we don't have to test intersection with *every* object

Backstory: 2D Parametric Lines

- $$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + t(x_1 - x_0) \\ y_0 + t(y_1 - y_0) \end{bmatrix}$$

- $\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0)$
- $\mathbf{p}(t) = \mathbf{o} + t(\mathbf{d})$

- start at point $\mathbf{p}_0$,
  go towards $\mathbf{p}_1$,
  according to parameter t
  - $\mathbf{p}(0) = \mathbf{p}_0$, $\mathbf{p}(1) = \mathbf{p}_1$



Review: Ray-Sphere Intersections, Lighting

- Intersections: solving a set of equations
  - Using implicit formulas for primitives
- Direct illumination: gradient of implicit surface

Example: Ray-Sphere intersection

ray: $x(t) = p_x + v_x t,\ \ y(t) = p_y + v_y t,\ \ z(t) = p_z + v_z t$

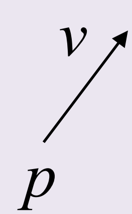
(unit) sphere: $x^2 + y^2 + z^2 = 1$

quadratic equation in t :

$$0 = (p_x + v_x t)^2 + (p_y + v_y t)^2 + (p_z + v_z t)^2 - 1$$

$$= t^2 (v_x^2 + v_y^2 + v_z^2) + 2t(p_x v_x + p_y v_y + p_z v_z)$$

$$+ (p_x^2 + p_y^2 + p_z^2) - 1$$



Example: Sphere normals

$$\mathbf{n}(x, y, z) = \begin{pmatrix} 2x \\ 2y \\ 2z \end{pmatrix}$$

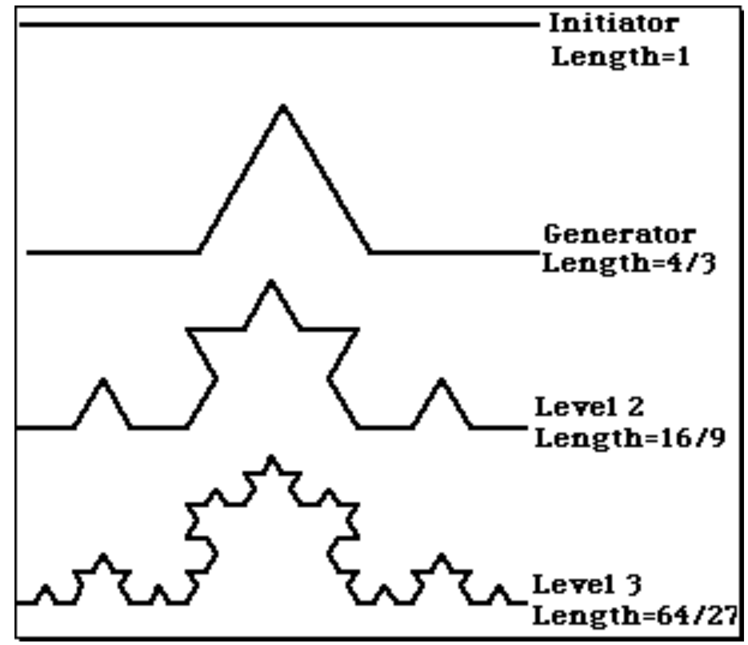
Procedural/Collision

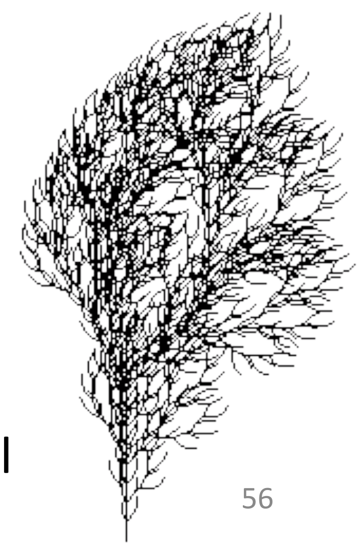
Review: Procedural Modeling

- textures, geometry
  - nonprocedural: explicitly stored in memory
- procedural approach
  - compute something on the fly
    - not load from disk
  - often less memory cost
  - visual richness
    - adaptable precision
- noise, fractals, particle systems

Review: Language-Based Generation

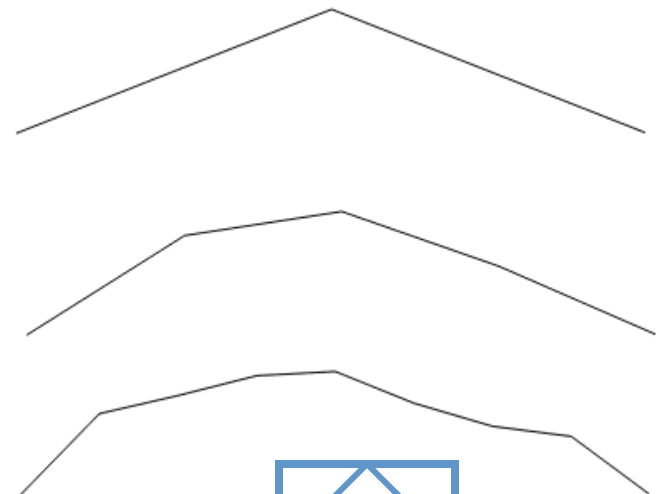
- L-Systems
  - F: forward, R: right, L: left
  - Koch snowflake:
    F = FLFRRFLF
  - Mariano's Bush:
    F=FF-[-F+F+F]+[+F-F-F]
    - angle 16



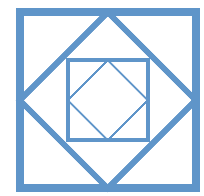


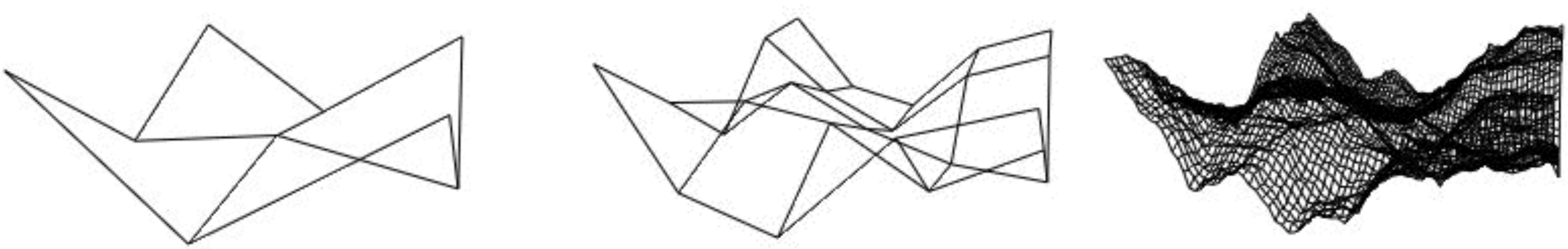
http://spanky.triumf.ca/www/fractint/lsys/plants.html

Review: Fractal Terrain



- 1D: midpoint displacement
  - – divide in half, randomly displace
  - – scale variance by half
- 2D: diamond-square
  - – generate new value at midpoint
  - – average corner values + random displacement
    - scale variance by half each time

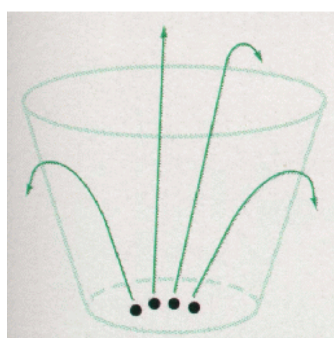


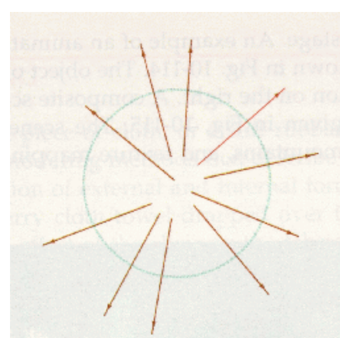


http://www.gameprogrammer.com/fractal.html

Review: Particle Systems

- changeable/fluid stuff
  - fire, steam, smoke, water, grass, hair, dust, waterfalls, fireworks, explosions, flocks
- life cycle
  - generation, dynamics, death
- rendering tricks
  - avoid hidden surface computations



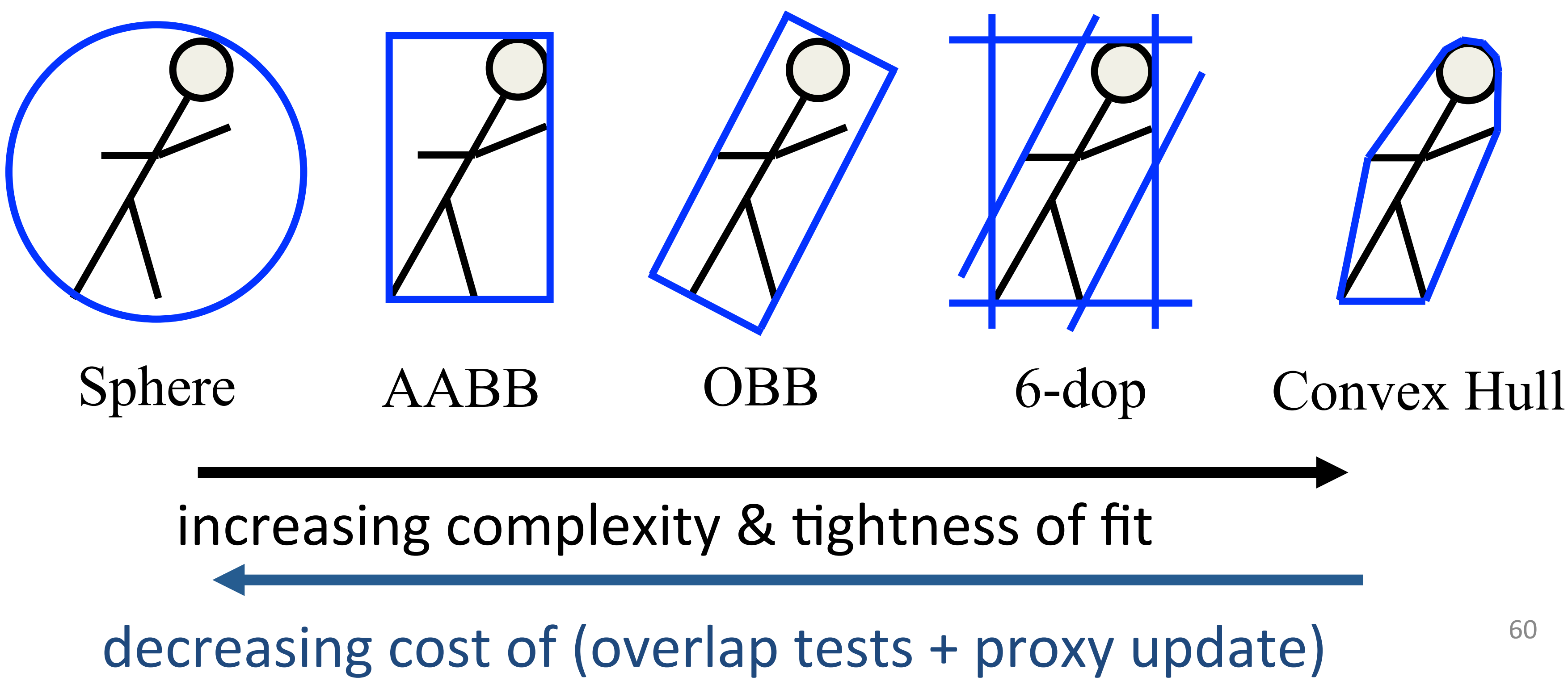


Review: Collision Detection

- boundary check
  - perimeter of world vs. viewpoint or objects
    - 2D/3D absolute coordinates for bounds
    - simple point in space for viewpoint/objects
- set of fixed barriers
  - walls in maze game
    - 2D/3D absolute coordinate system
- set of moveable objects
  - one object against set of items
    - missile vs. several tanks
  - multiple objects against each other
    - punching game: arms and legs of players
    - room of bouncing balls

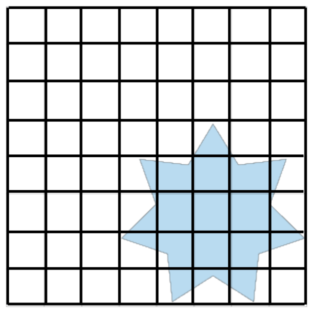
Review: Collision Proxy Tradeoffs

- collision proxy (bounding volume) is piece of geometry used to represent complex object for purposes of finding collision
- proxies exploit facts about human perception
  - we are bad at determining collision correctness
  - especially many things happening quickly

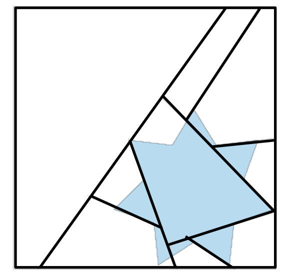


Review: Spatial Data Structures

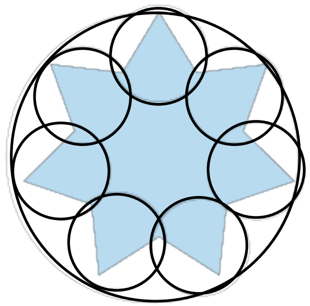
uniform grids



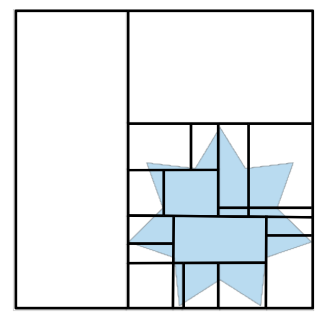
BSP trees



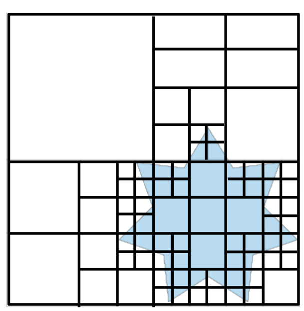
bounding volume hierarchies



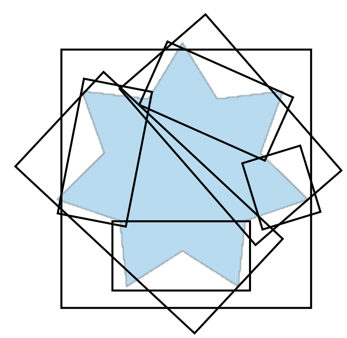
kd-trees



octrees



OBB trees



61

Hidden Surfaces / Picking / Blending

Review: Z-Buffer Algorithm

- augment color framebuffer with Z-buffer or depth buffer which stores Z value at each pixel
  - at frame beginning, initialize all pixel depths to ∞
  - when rasterizing, interpolate depth (Z) across polygon
  - check Z-buffer before storing pixel color in framebuffer and storing depth in Z-buffer
  - don't write pixel if its Z value is more distant than the Z value already stored there

Review: Depth Test Precision

– reminder: perspective transformation maps eye-space (VCS) *z* to NDC *z*

$$\begin{bmatrix} E & 0 & A & 0 \\ 0 & F & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} Ex + Az \\ Fy + Bz \\ Cz + D \\ -z \end{bmatrix} = \begin{bmatrix} -\left(\frac{Ex}{z} + A\right) \\ -\left(\frac{Fy}{z} + B\right) \\ -\left(C + \frac{D}{z}\right) \\ 1 \end{bmatrix}$$

$$z_{NDC} = -\left(C + \frac{D}{z_{VCS}}\right) \qquad C = \frac{-(f+n)}{f-n} \qquad D = \frac{-2fn}{f-n}$$

– thus: depth buffer essentially stores 1/z (for VCS z)
  – high precision for near, low precision for distant

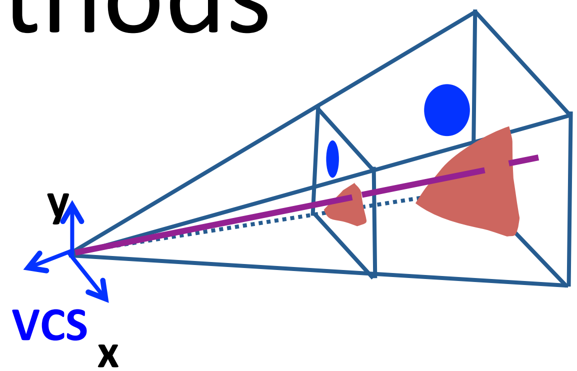
Review: Integer Depth Buffer

- reminder from viewing discussion: depth ranges
  - VCS range [zNear, zFar], NDCS range [-1,1], DCS z range [0,1]
- convert fractional real number to integer format
  - multiply by 2^n then round to nearest int
  - where n = number of bits in depth buffer
- 24 bit depth buffer = 2^24 = 16,777,216 possible values
  - small numbers near, large numbers far
- consider VCS depth: $z_{DCS} = (1<<N)*(a + b / z_{VCS})$
  - N = number of bits of Z precision, $1<<N$ bitshift $= 2^n$
  - a = zFar / (zFar - zNear)
  - b = zFar * zNear / (zNear - zFar)
  - z_{VCS} = distance from the eye to the object

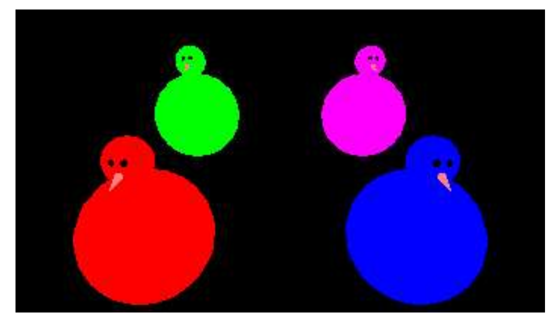
Full derivation at https://www.opengl.org/archives/resources/faq/technical/depthbuffer.htm

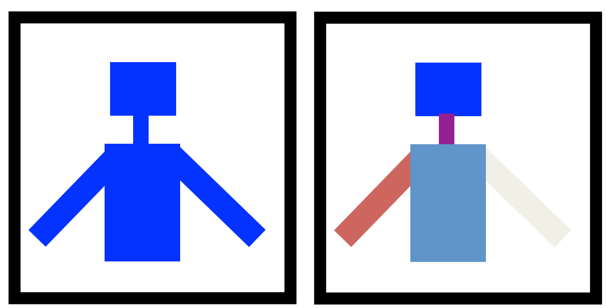
Review: Picking Methods

- raycaster intersection support

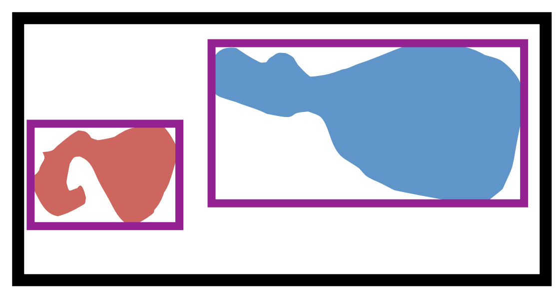


- offscreen buffer color coding



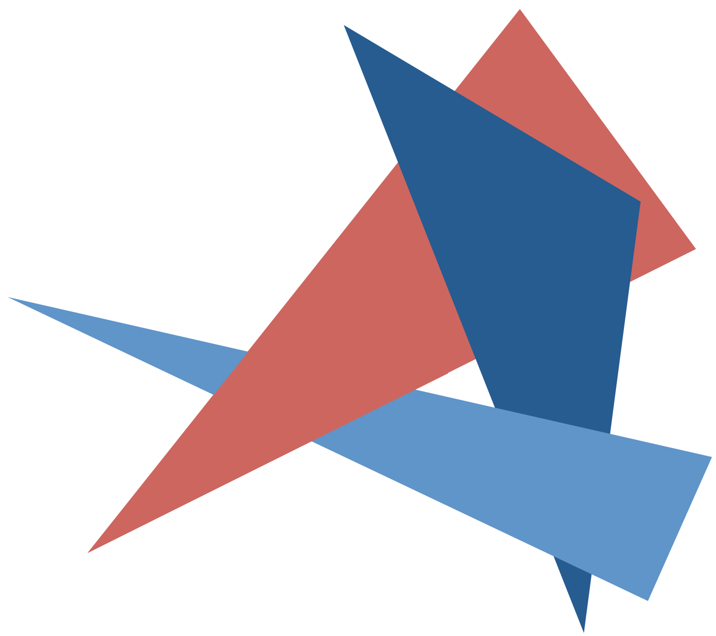


- bounding extents



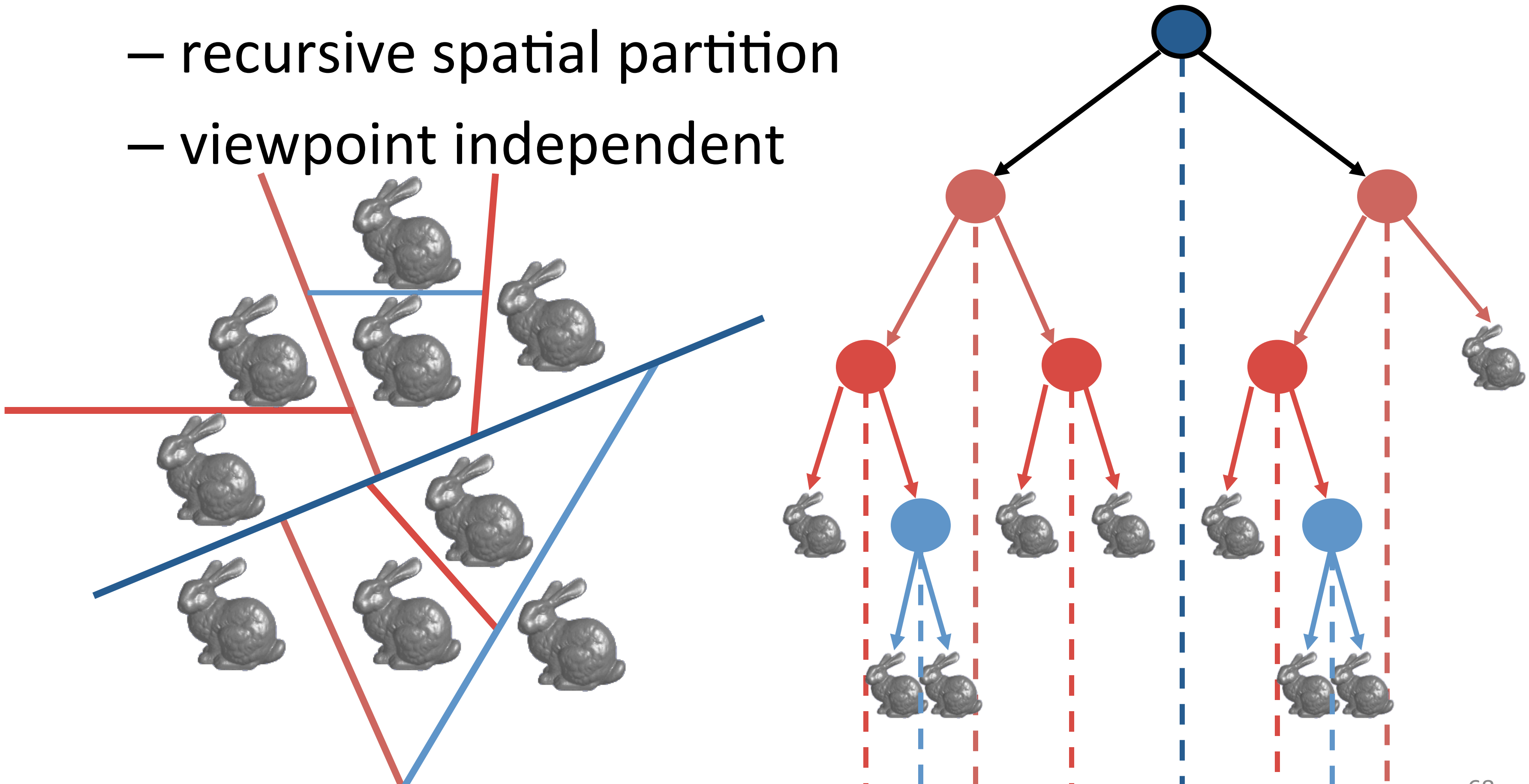
Review: Painter's Algorithm

- draw objects from back to front
- problems: no valid visibility order for
  - intersecting polygons
  - cycles of non-intersecting polygons possible



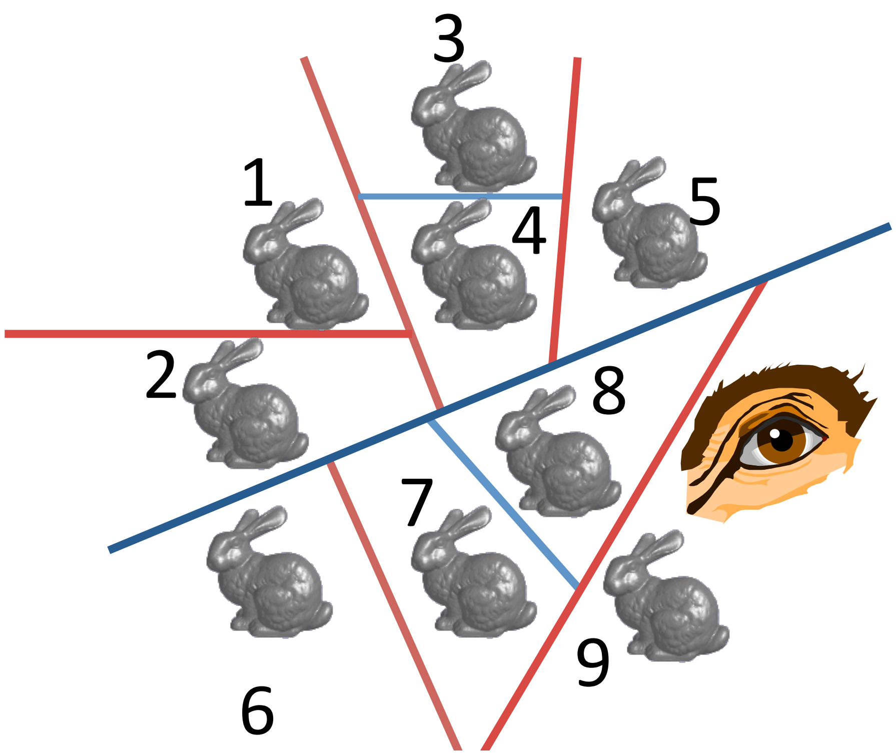
Review: BSP Trees

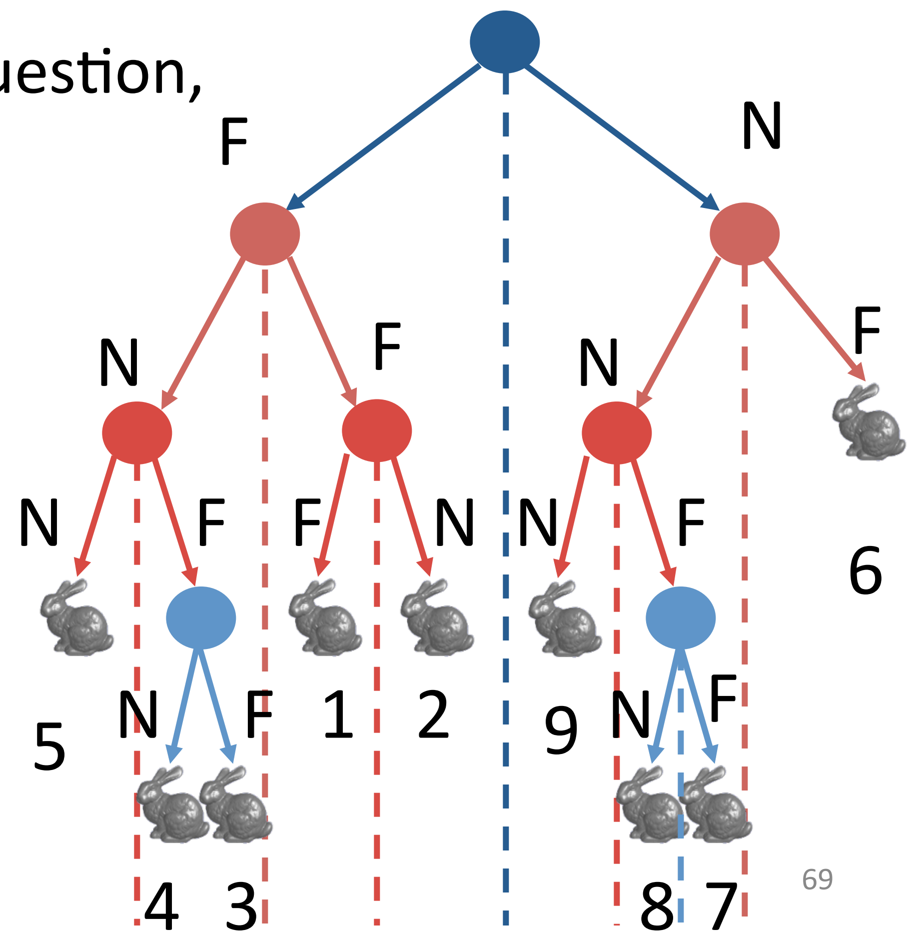
- preprocess: create binary tree
  - recursive spatial partition
  - viewpoint independent



Review: BSP Trees

- runtime: correctly traversing this tree enumerates objects from back to front
  - viewpoint dependent: check which side of plane viewpoint is on **at each node**
  - draw far, draw object in question, draw near





Review: Object Space Algorithms

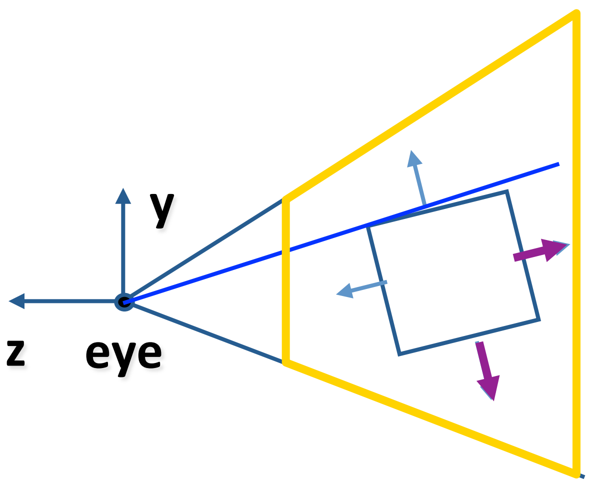
- determine visibility on object or polygon level
  - using camera coordinates
- resolution independent
  - explicitly compute visible portions of polygons
- early in pipeline
  - after clipping
- requires depth-sorting
  - painter's algorithm
  - BSP trees

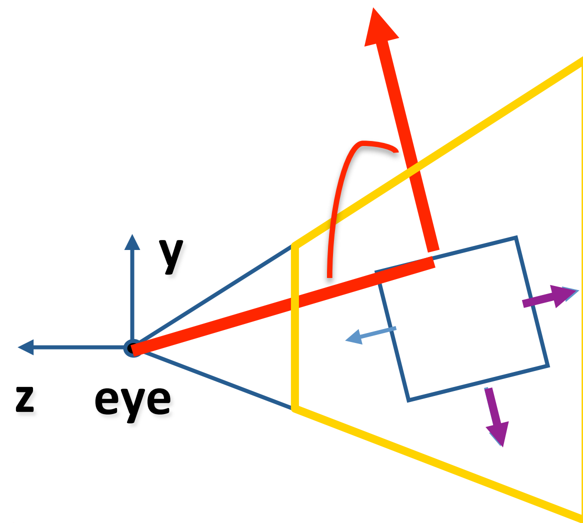
Review: Image Space Algorithms

- perform visibility test for in screen coordinates
  - limited to resolution of display
  - Z-buffer: check every pixel independently
- performed late in rendering pipeline

Review: Back-face Culling

VCS

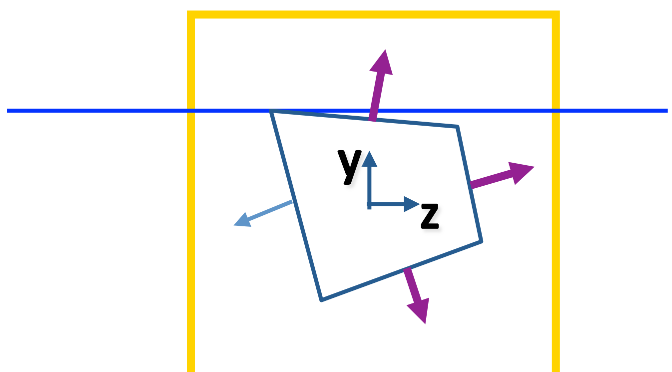




VCS: cull if angle between eye-face vector and normal > 90

NDCS

eye



NDCS: cull if $N_Z > 0$

Review: Invisible Primitives

- *why might a polygon be invisible?*
  - polygon outside the *field of view / frustum*
    - solved by clipping
  - polygon is *backfacing*
    - solved by backface culling
  - polygon is *occluded* by object(s) nearer the viewpoint
    - solved by hidden surface removal

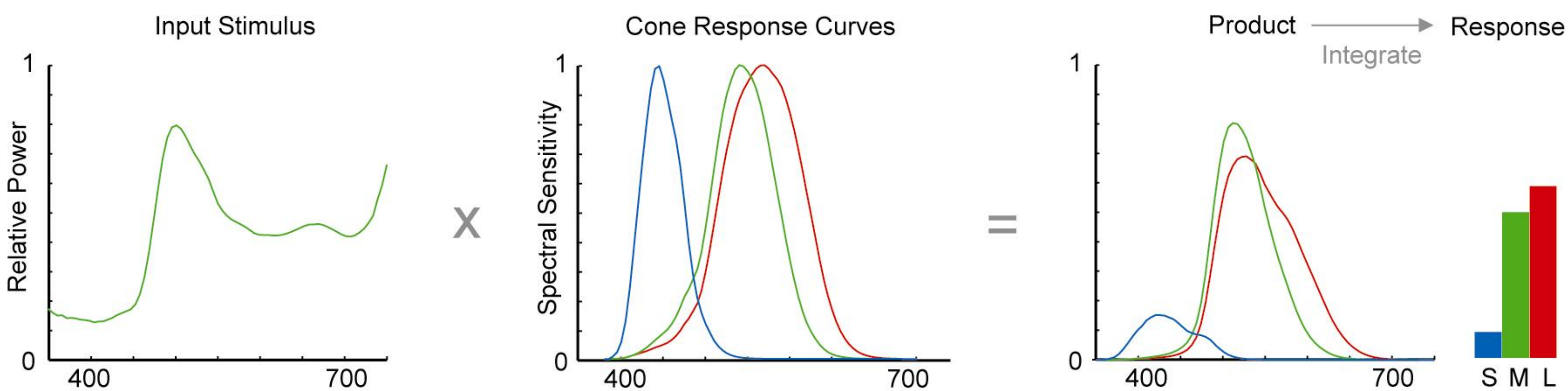
Review: Blending with Premultiplied Alpha

- specify opacity with alpha channel α
  - α=1: opaque, α=.5: translucent, α=0: transparent
- how to express a pixel is half covered by a red object?
  - obvious way: store color independent from transparency (r,g,b,α)
    - intuition: alpha as transparent colored glass
      - 100% transparency can be represented with many different RGB values
    - pixel value is (1,0,0,.5)
    - upside: easy to change opacity of image, very intuitive
    - downside: compositing calculations are more difficult - not associative
  - elegant way: premultiply by α so store (αr, αg, αb,α)
    - intuition: alpha as screen/mesh
      - RGB specifies how much color object contributes to scene
      - alpha specifies how much object obscures whatever is behind it (coverage)
      - alpha of .5 means half the pixel is covered by the color, half completely transparent
      - only one 4-tuple represents 100% transparency: (0,0,0,0)
    - pixel value is (.5, 0, 0, .5)
    - upside: compositing calculations easy (& additive blending for glowing!)
    - downside: less intuitive

Color

Backstory & Review: Trichromacy

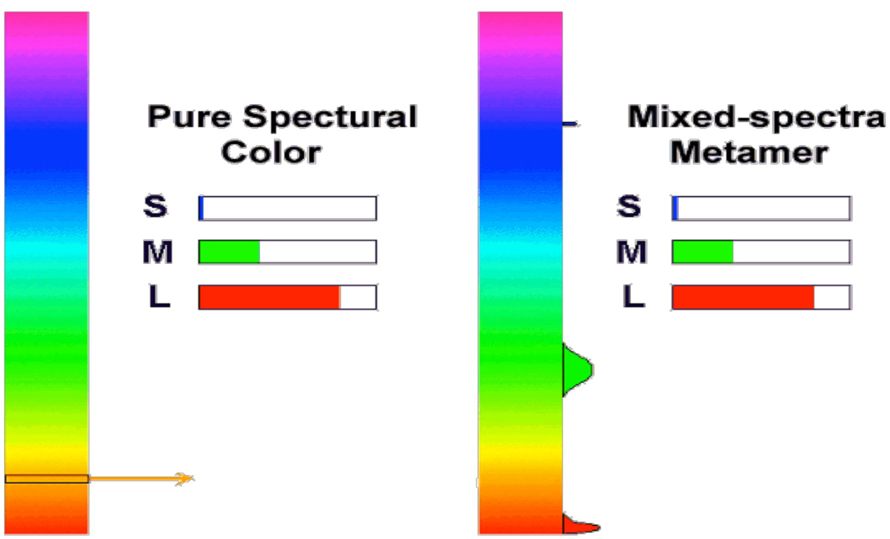
- trichromacy
  - three types of cones: S, M, L
  - color is combination of cone stimuli
    - different cone responses: area function of wavelength
- for a given spectrum
  - multiply by responses curve
  - integrate to get response

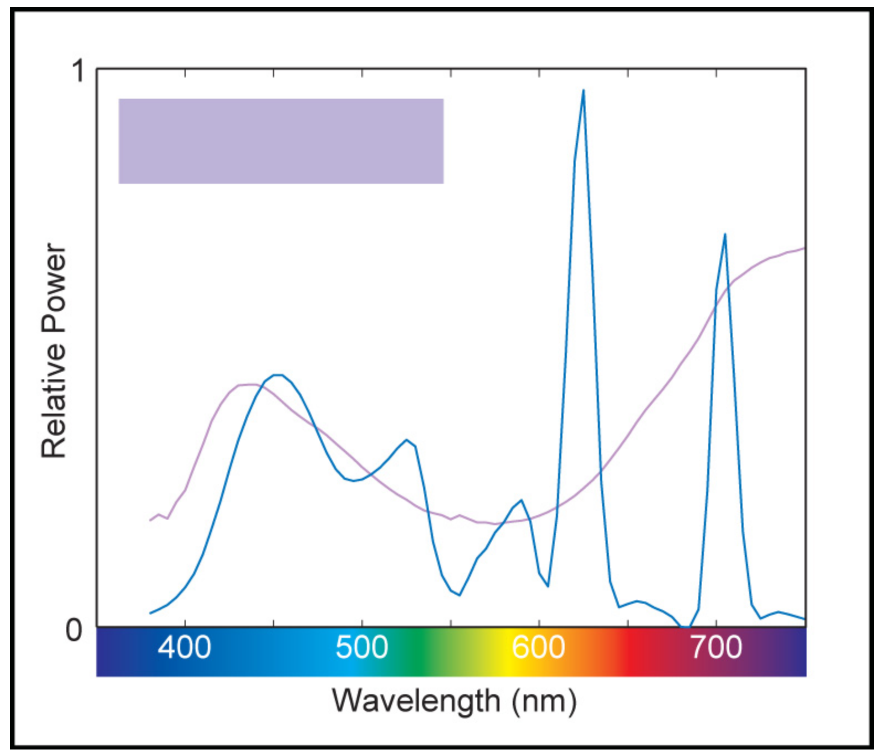


From *A Field Guide to Digital Color*, © A.K. Peters, 2003

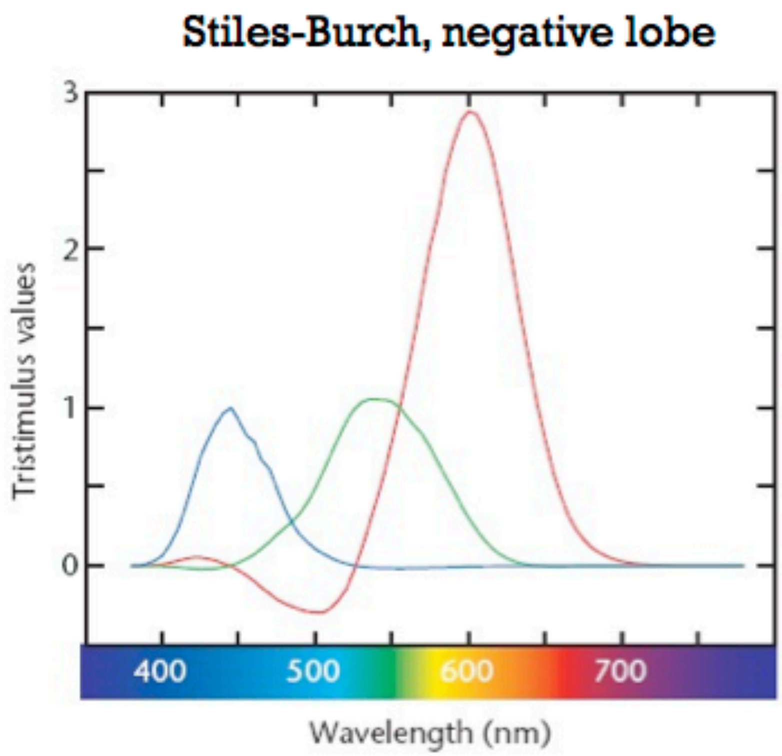
Review: Metamers

- brain sees only cone response
  - different spectra appear the same: metamers

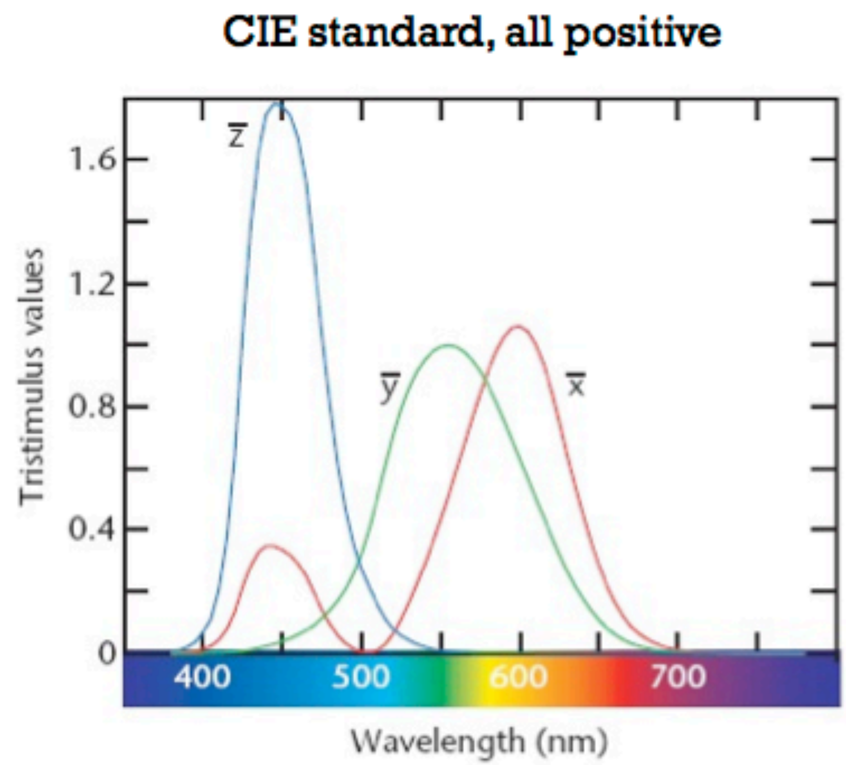




Review: Measured vs. CIE XYZ Color Spaces

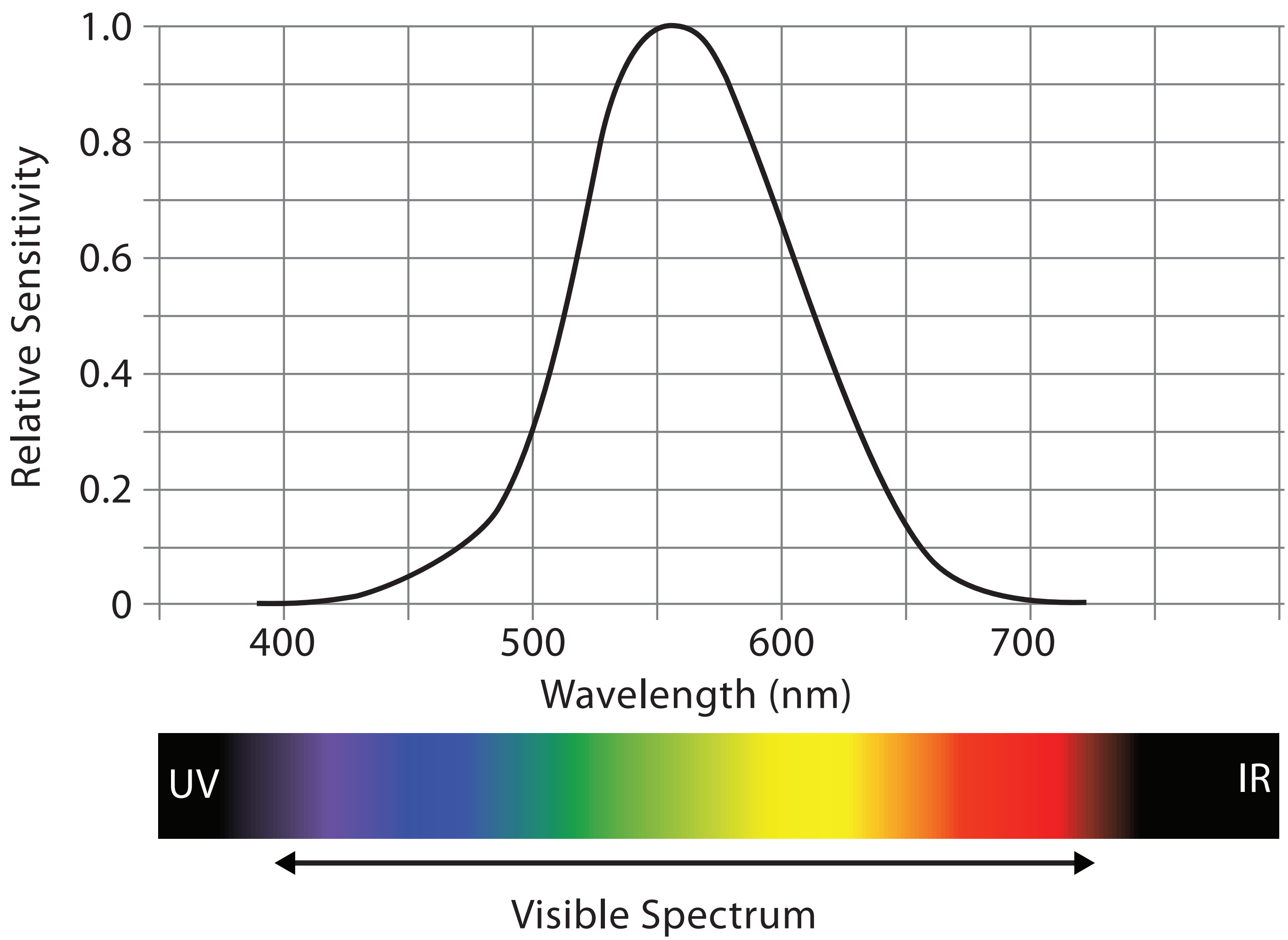


- measured basis
  - monochromatic lights
  - physical observations
  - negative lobes



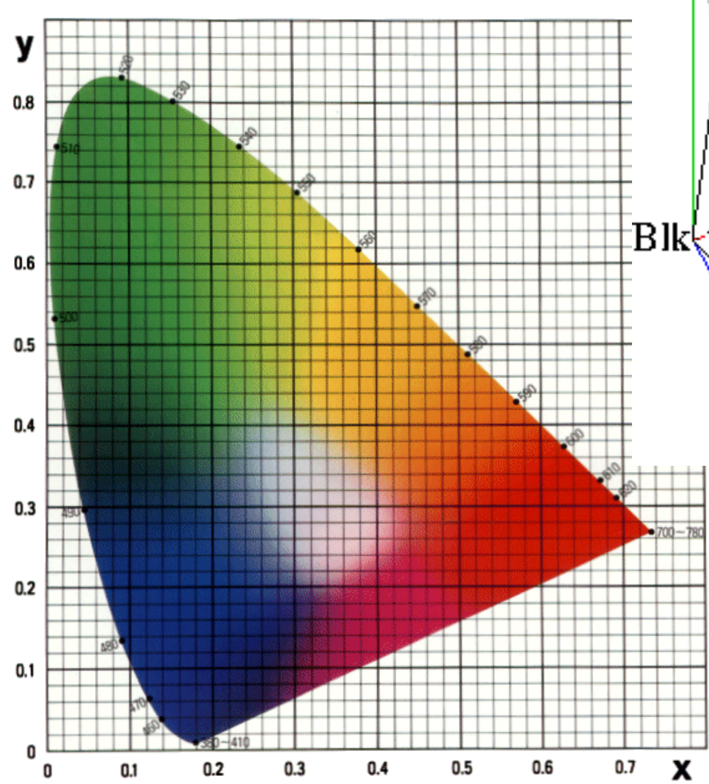
- transformed basis
  - "imaginary" lights
  - all positive, unit area
  - Y is luminance

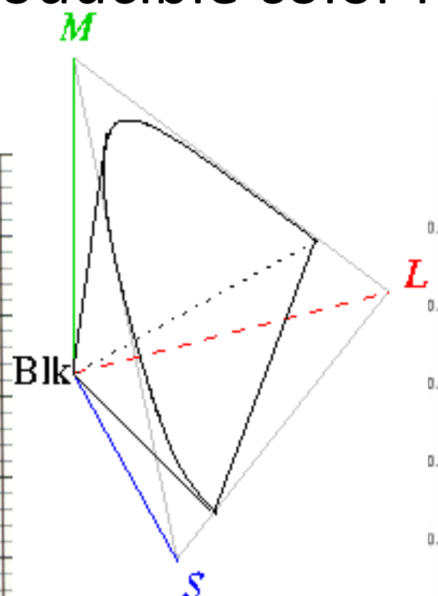
Backstory: Spectral Sensitivity Curve

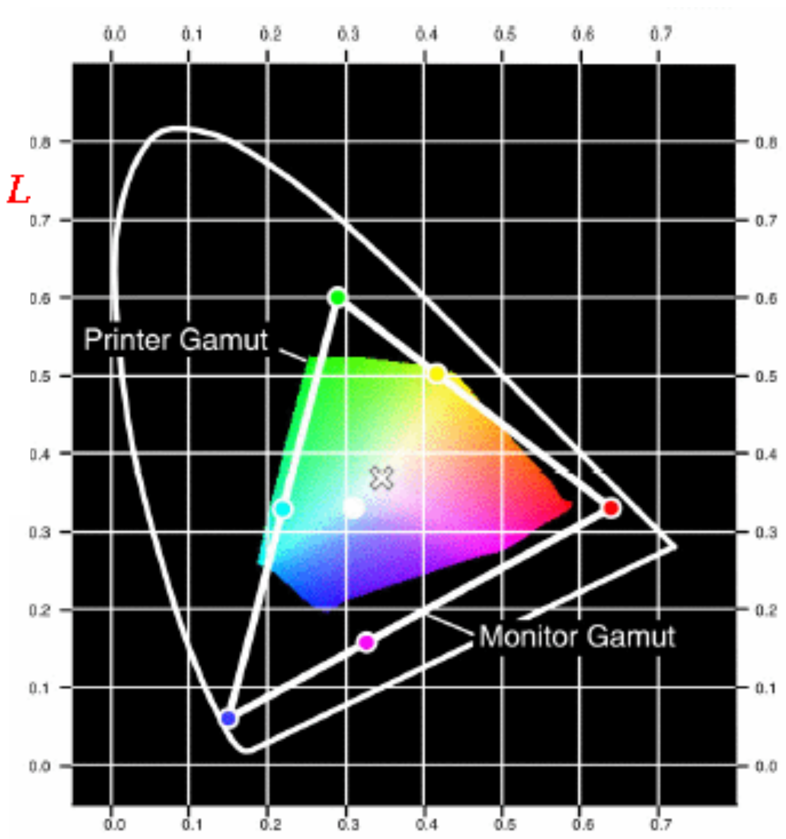


Review: CIE Chromaticity Diagram and Gamuts

- plane of equal brightness showing chromaticity
- gamut is polygon, device primaries at corners
  - defines reproducible color range

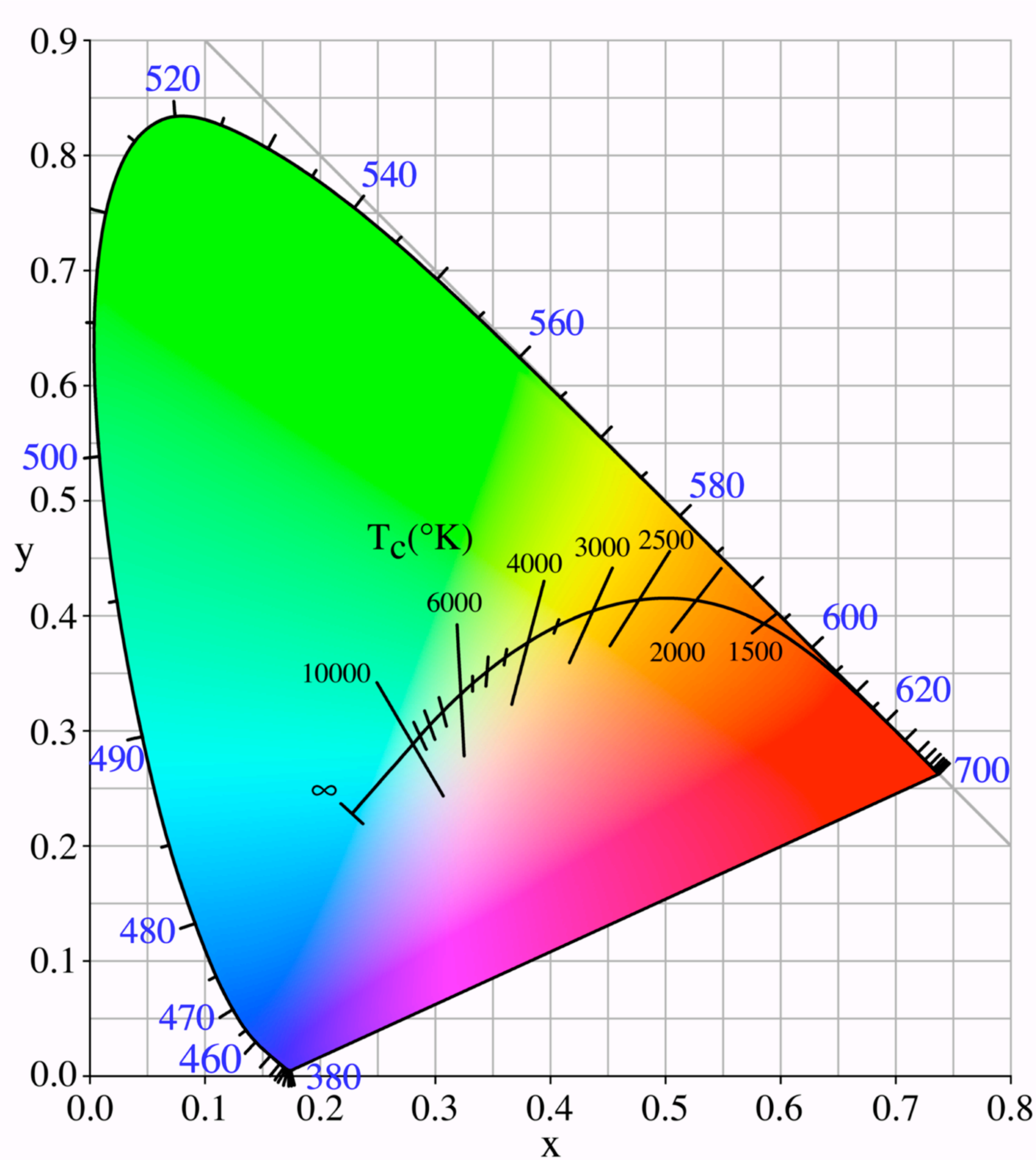






Review: Blackbody Curve

- illumination:
  - candle 2000K
  - A: Light bulb 3000K
  - sunset/ sunrise 3200K
  - D: daylight 6500K
  - overcast day 7000K
  - lightning >20,000K



Review: Color Constancy

- automatic "white balance" from change in illumination
- vast amount of processing behind the scenes!
- colorimetry vs. perception

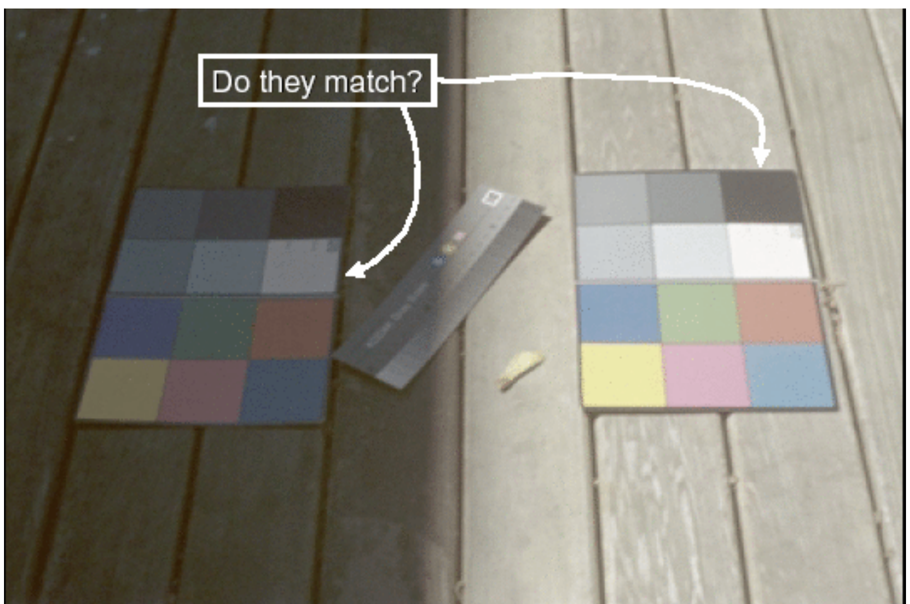
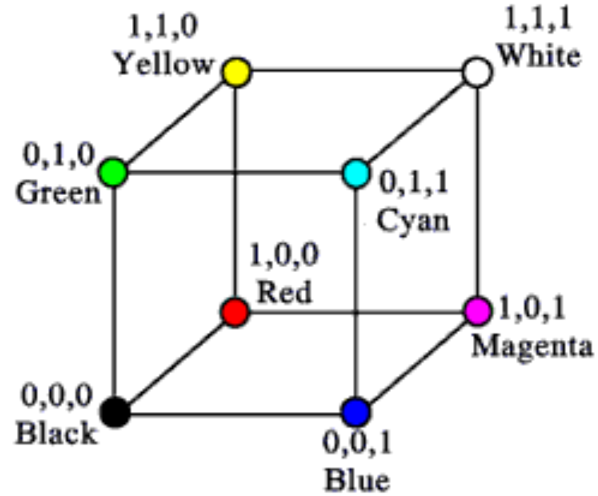


Image courtesy of John MCann

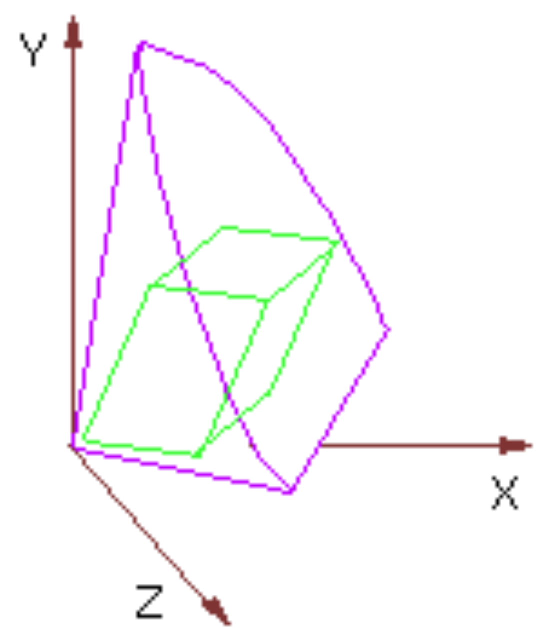
From Color Appearance Models, fig 8-1

Review: RGB Color Space (Color Cube)

- define colors with (r, g, b) amounts of red, green, and blue
  - used by OpenGL
  - hardware-centric

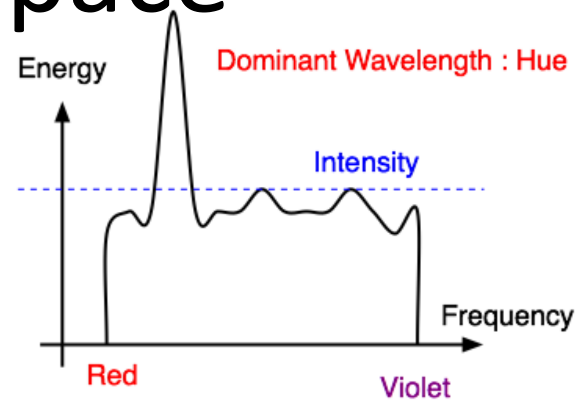


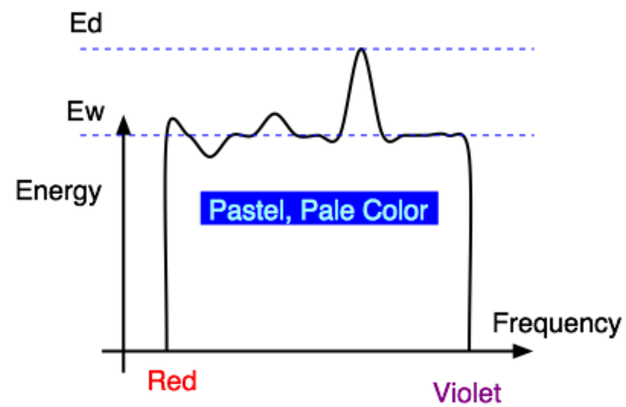
- RGB color cube sits within CIE color space
  - subset of perceivable colors
  - scale, rotate, shear cube

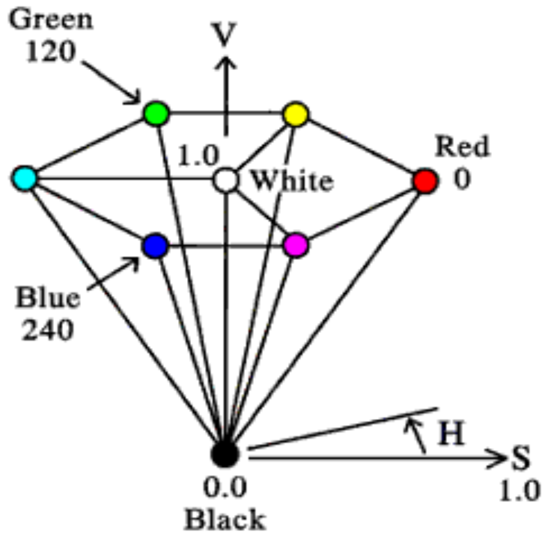


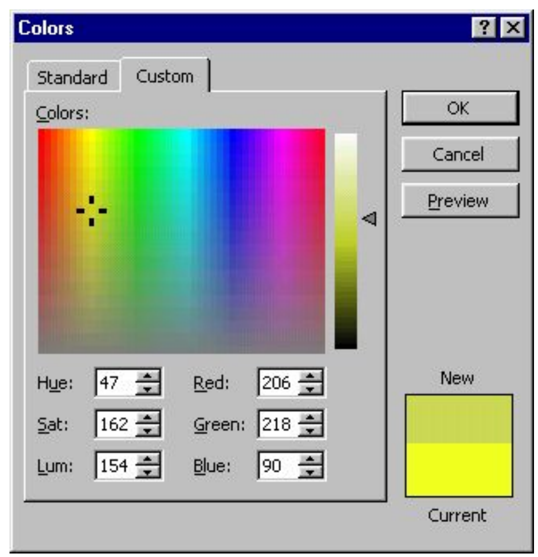
Review: HSV Color Space

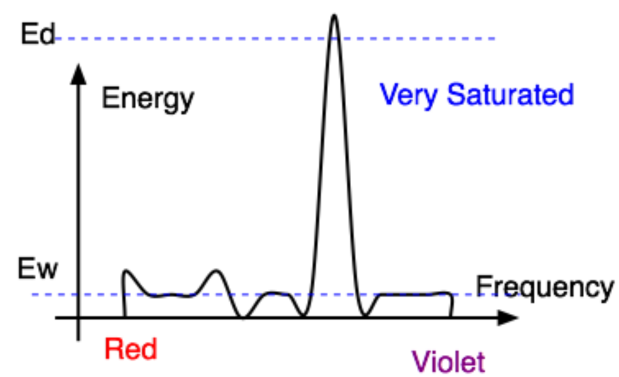
- hue: dominant wavelength, “color”
- saturation: how far from grey
- value: how far from black/white
  - aka brightness, intensity: HSB / HSV / HSI similar
- cannot convert to RGB with matrix alone











Review: HSI/HSV and RGB

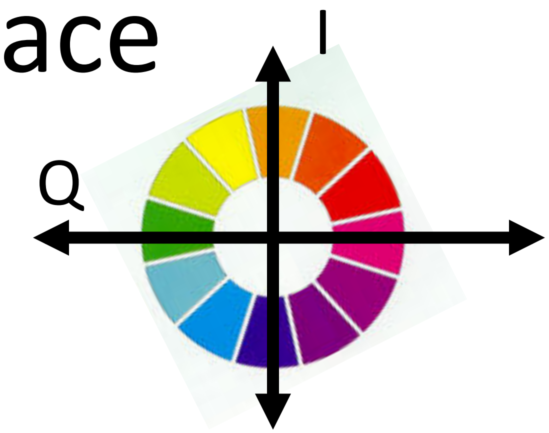
- HSV/HSI conversion from RGB
  - hue same in both
  - value is max, intensity is average

$$H = \cos^{-1}\left[\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right] \quad \text{if (B > G)}, \quad H = 360 - H$$

- HSI: $$S = 1 - \frac{\min(R,G,B)}{I} \qquad I = \frac{R+G+B}{3}$$

- HSV: $$S = 1 - \frac{\min(R,G,B)}{V} \qquad V = \max(R,G,B)$$

Review: YIQ Color Space



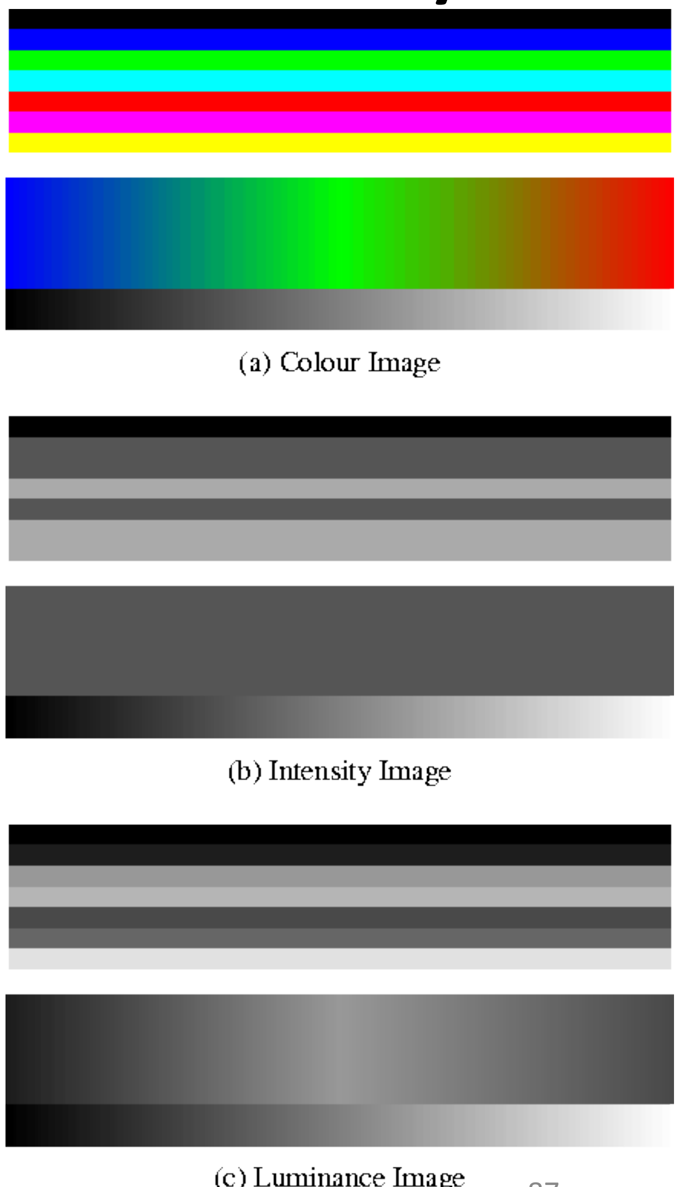
- color model used for color TV
  - Y is luminance (same as CIE)
  - I & Q are color (not same I as HSI!)
  - using Y backwards compatible for B/W TVs
  - conversion from RGB is linear

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

  - green is much lighter than red, and red lighter than blue

Review: Luminance vs. Intensity

- luminance
  - Y of YIQ
  - 0.299R + 0.587G + 0.114B
  - captures important factor
- intensity/value/brightness
  - I/V/B of HSI/HSV/HSB
  - 0.333R + 0.333G + 0.333B
  - not perceptually based



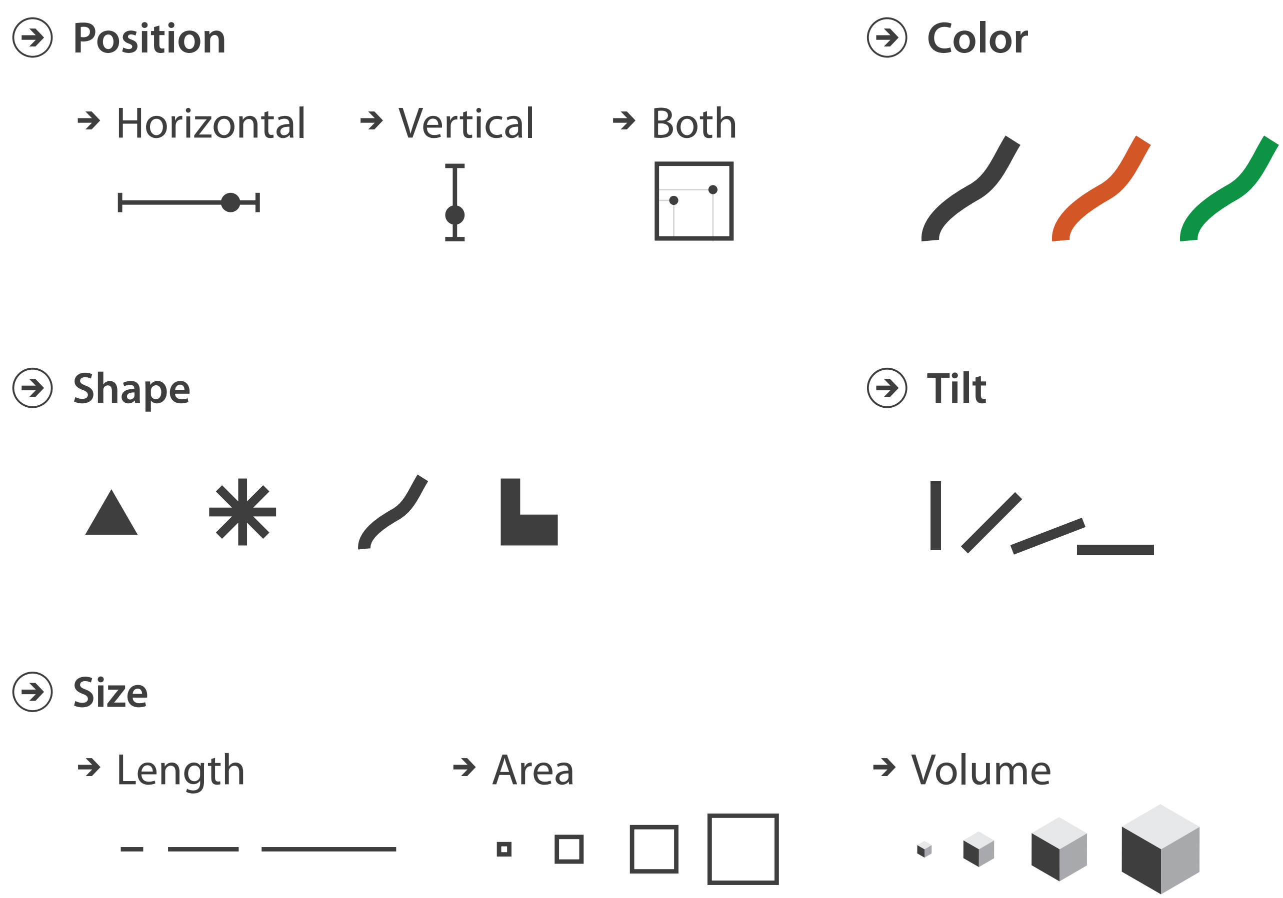
www.csse.uwa.edu.au/~robyn/Visioncourse/colour/lecture/node5.html

Visualization

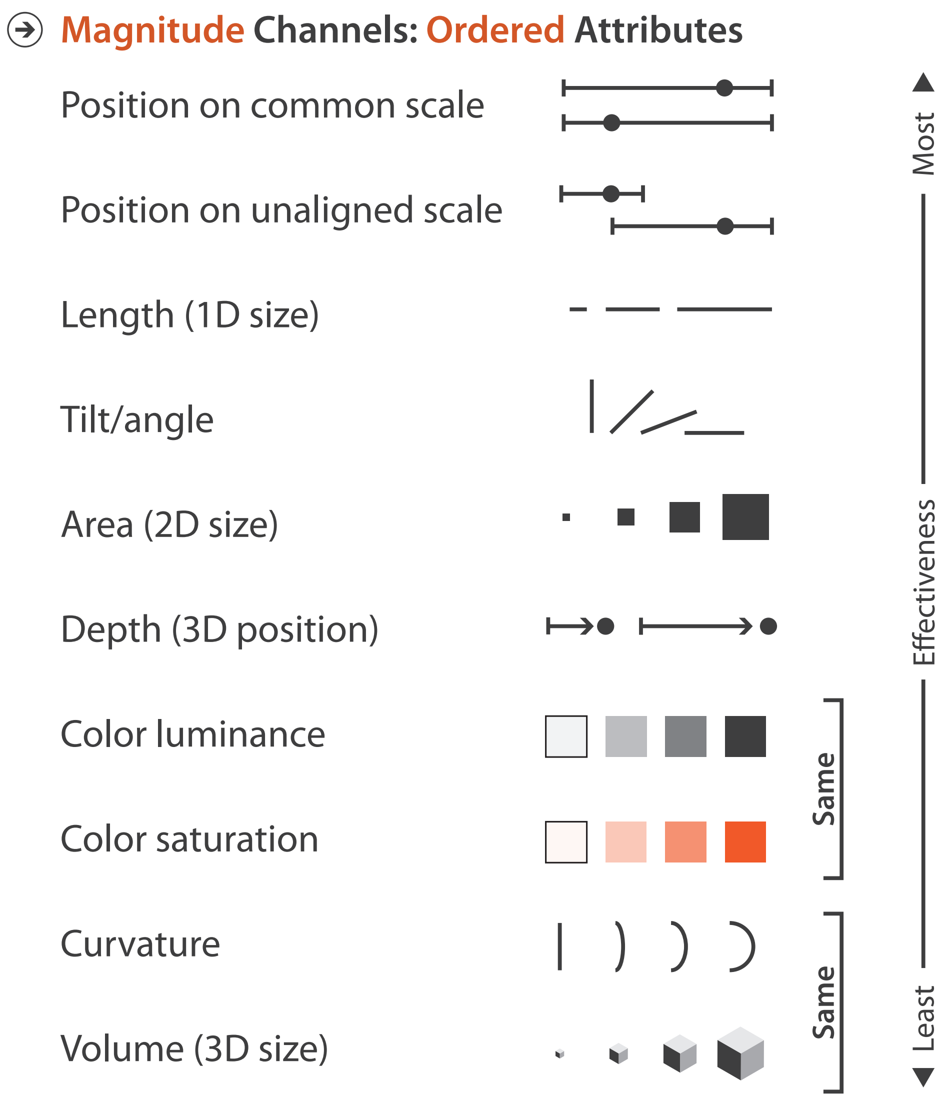
Review: Marks and Channels

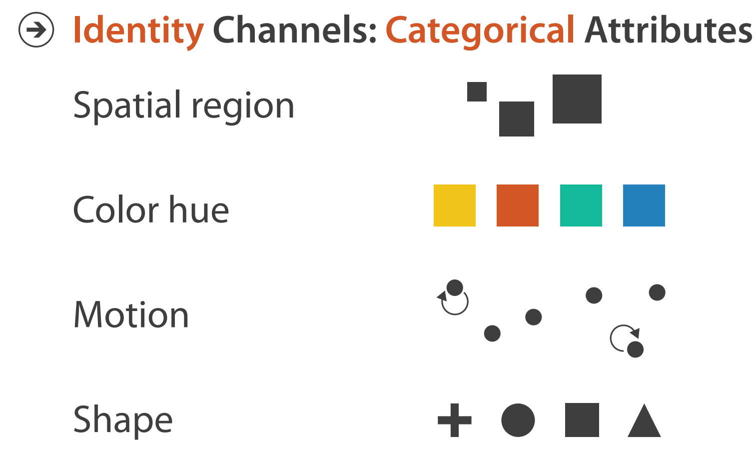
- marks
  - geometric primitives

- channels
  - control appearance of marks



Review: Channel Rankings





- expressiveness principle
  – match channel and data characteristics
- effectiveness principle
  – encode most important attributes with highest ranked channels