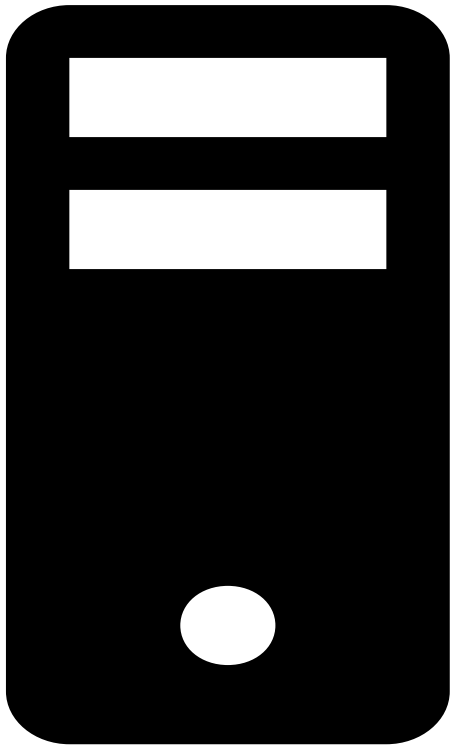
CG and Rendering

Shading

Computer Graphics

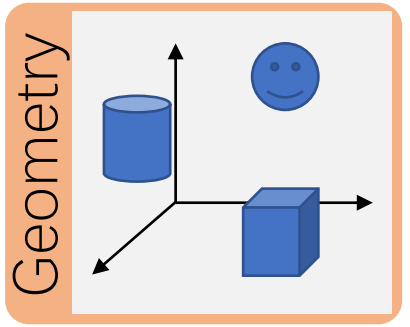


Modeling

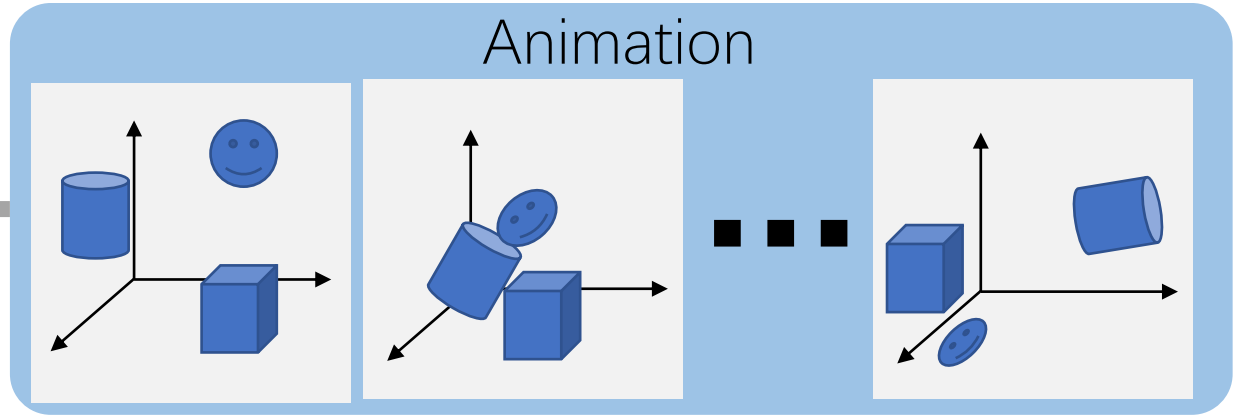
Simulation

Rendering

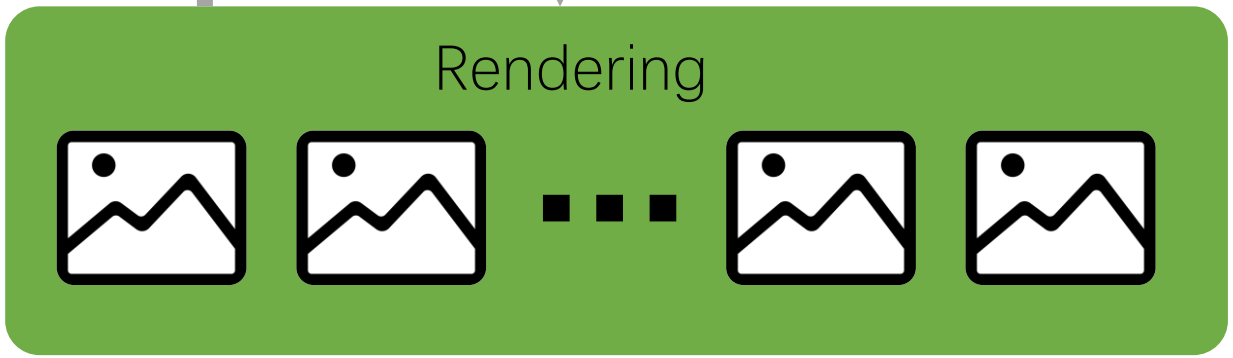
Computer Graphics



Modeling

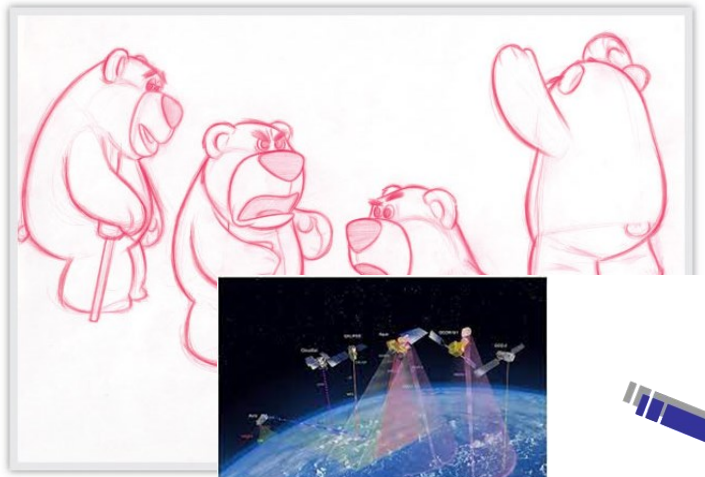


Simulation

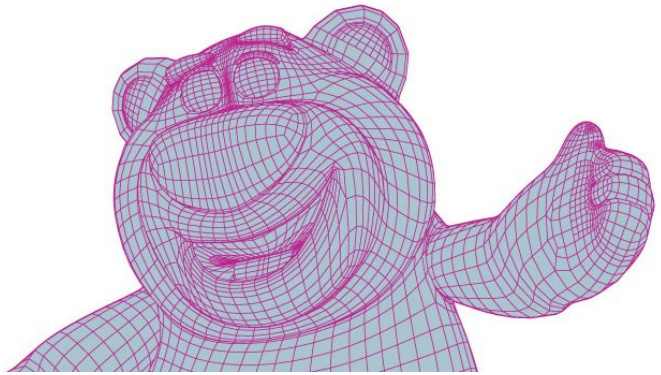


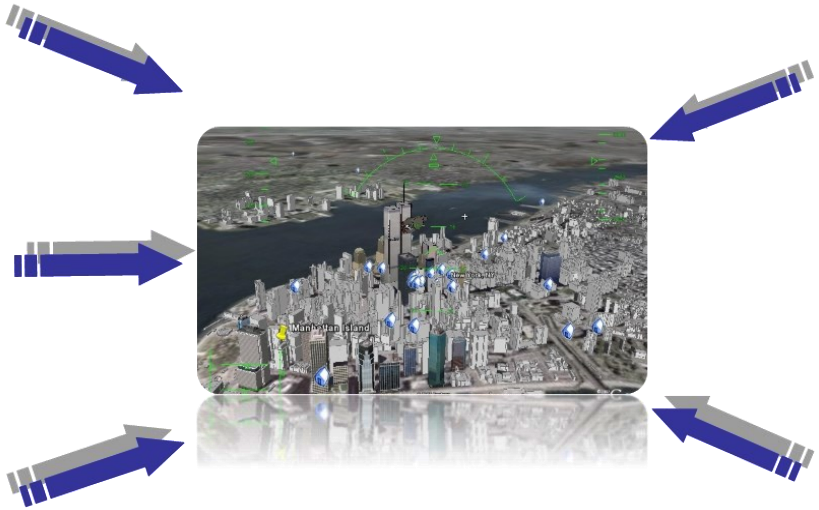
Rendering

Computer Graphics



"Lotso Poses" by Danie
Toy Story 3, 2010
Pencil on paper



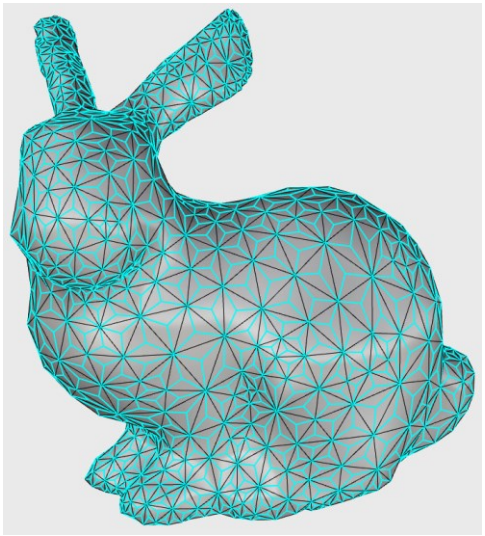


Modeling

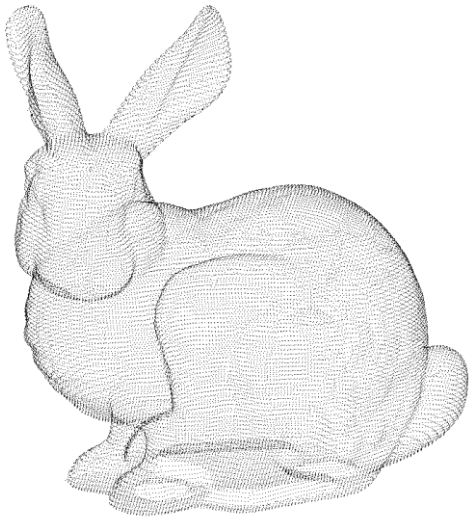
Simulation

Rendering

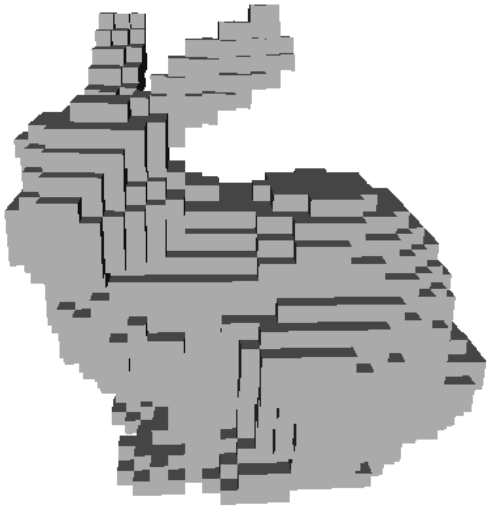
Computer Graphics



Mesh



Point Cloud



Volumetric Grid

Modeling

Simulation

Rendering

Computer Graphics

Modeling

Simulation

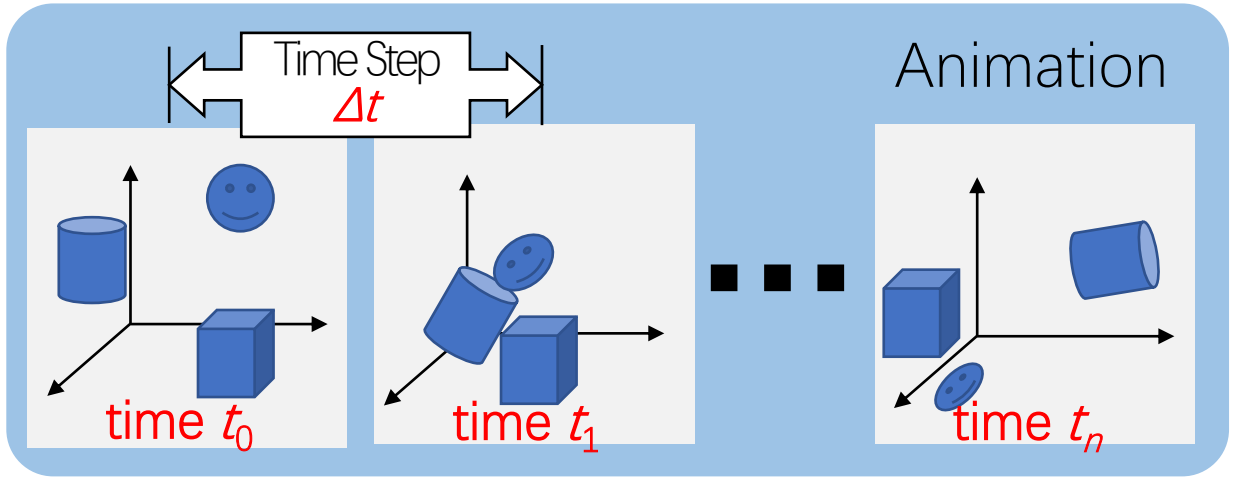
Rendering

Computer Graphics

The state in each time step, including:

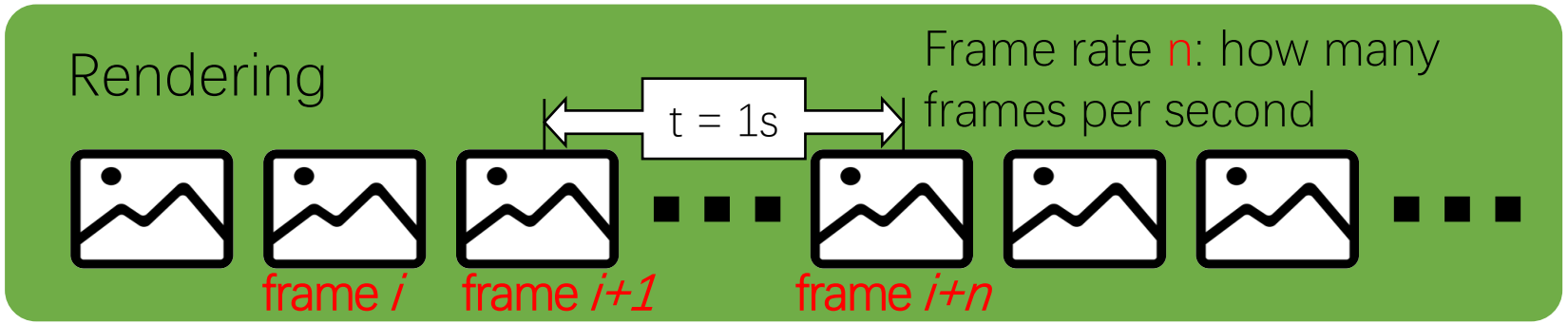
- Position/orientation
- Velocity
- Appearance
- Density
- …

Time step can be $\frac{1s}{N \cdot n}$



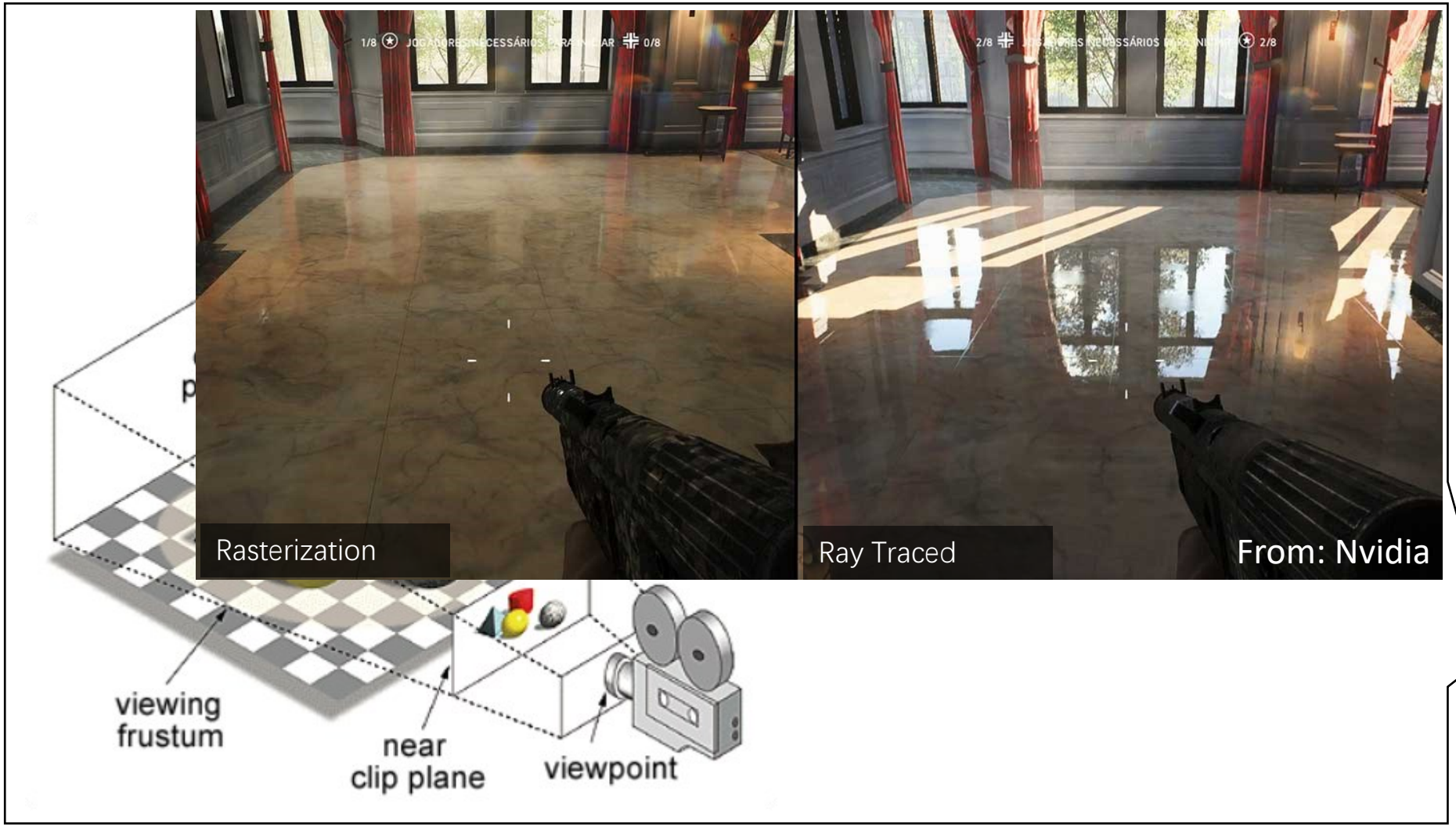
Modeling

Simulation



Rendering

Computer Graphics



Modeling

Simulation

Rendering

Computer Graphics

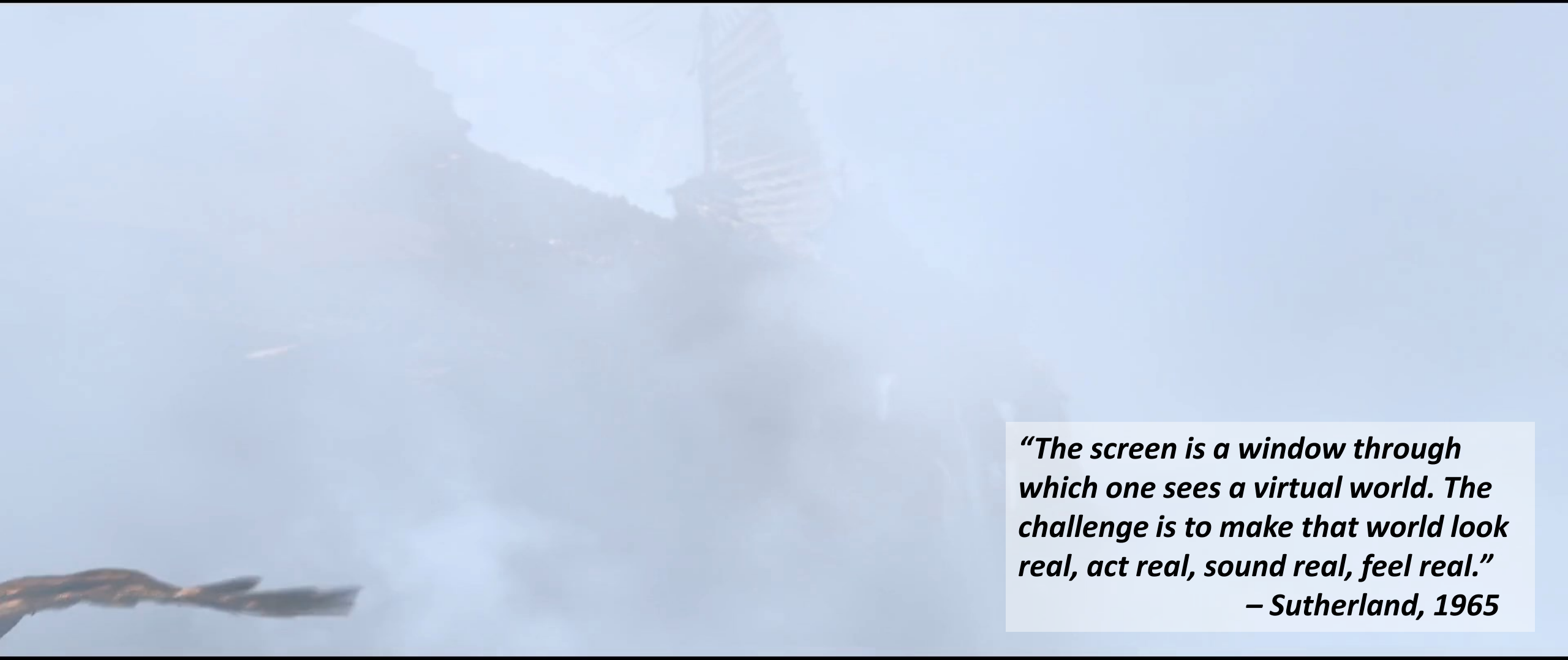
Modeling

Simulation

Rendering

"The screen is a window through which one sees a virtual world. The challenge is to make that world look real, act real, sound real, feel real."

– Sutherland, 1965



From: Computer Animation Festival Trailer, SIGGRAPH 2014

Rendering

Rasterization vs. Ray Tracing

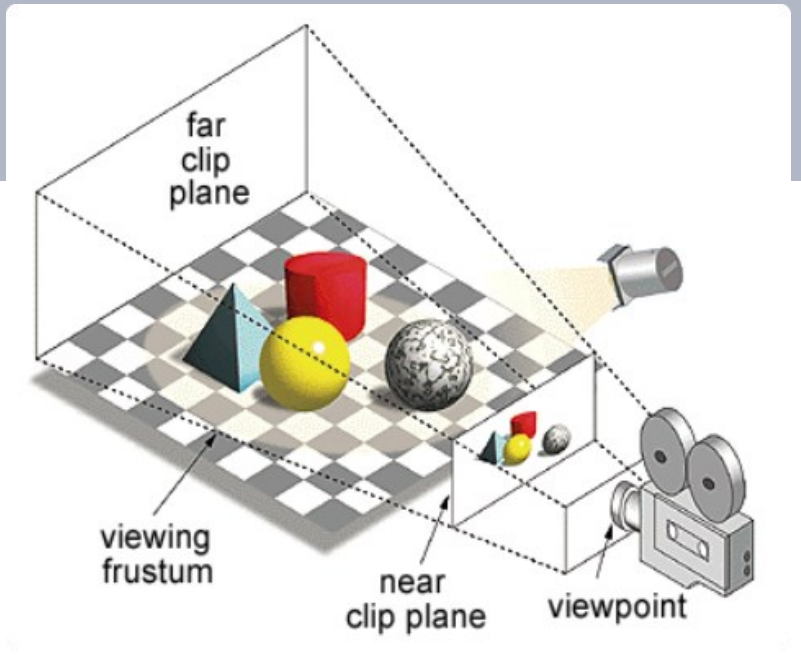
- Rasterization:
  - Analogy: Like painting—coloring objects
  - Method: Uses empirical lighting models (Shading, Illustration, Pipeline)

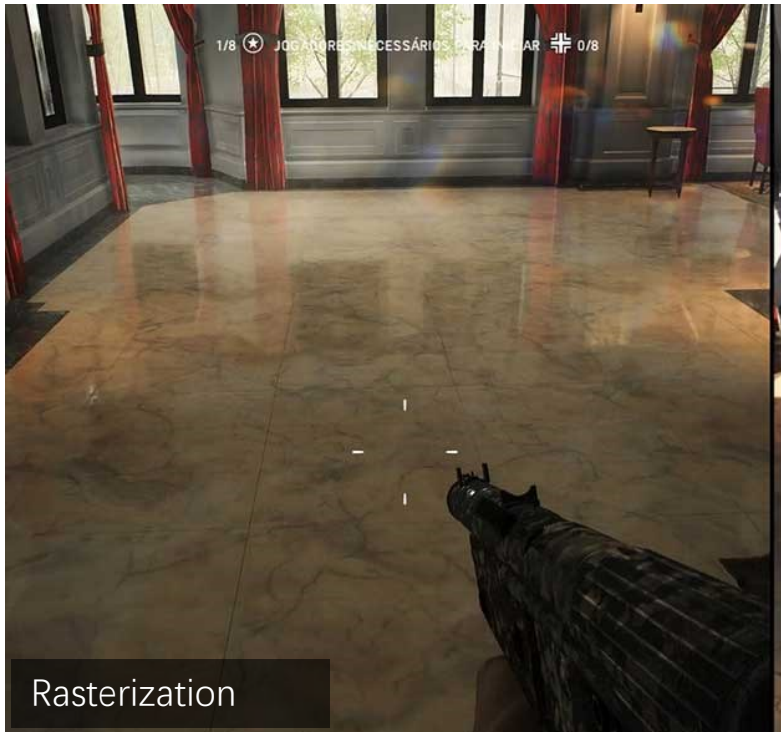
    Efficient & Lack realism

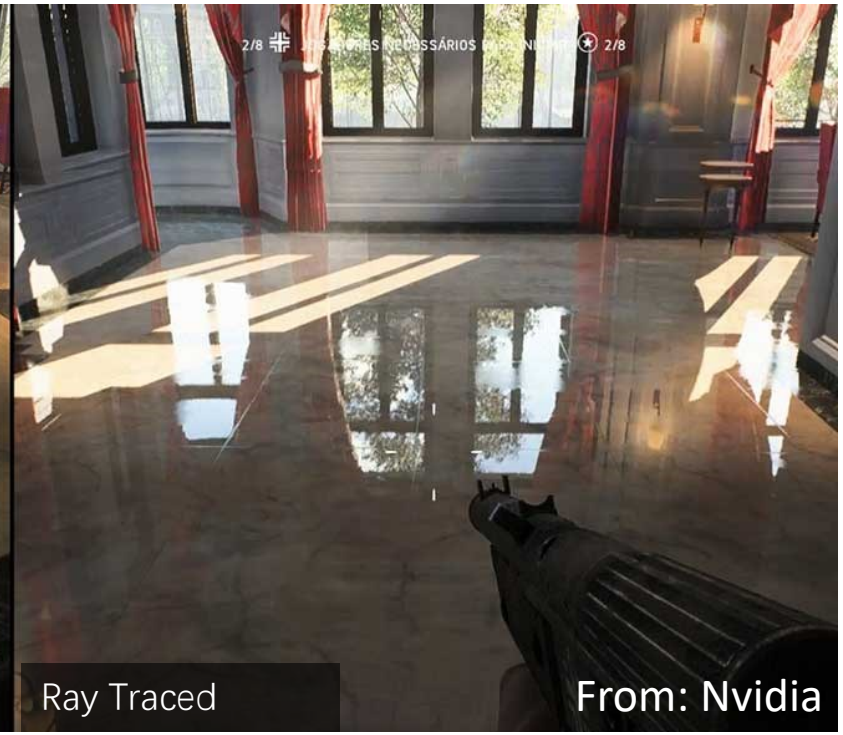
    Usually darker
- Ray Tracing:
  - Analogy: Like photography
  - Method: Traces light interactions (reflection, refraction):

    Realism & Less efficient

    Usually brighter, but full of details







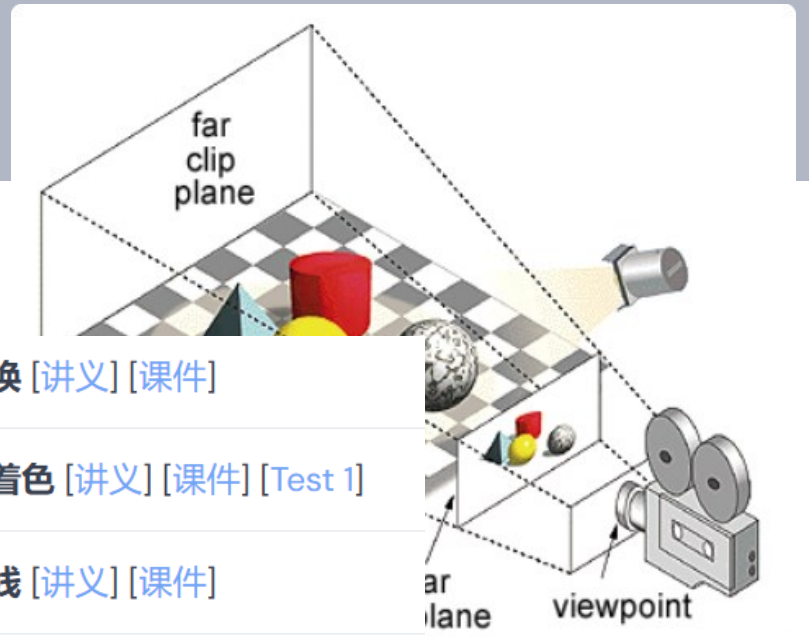
Rendering

Rasterization vs. Ray Tracing

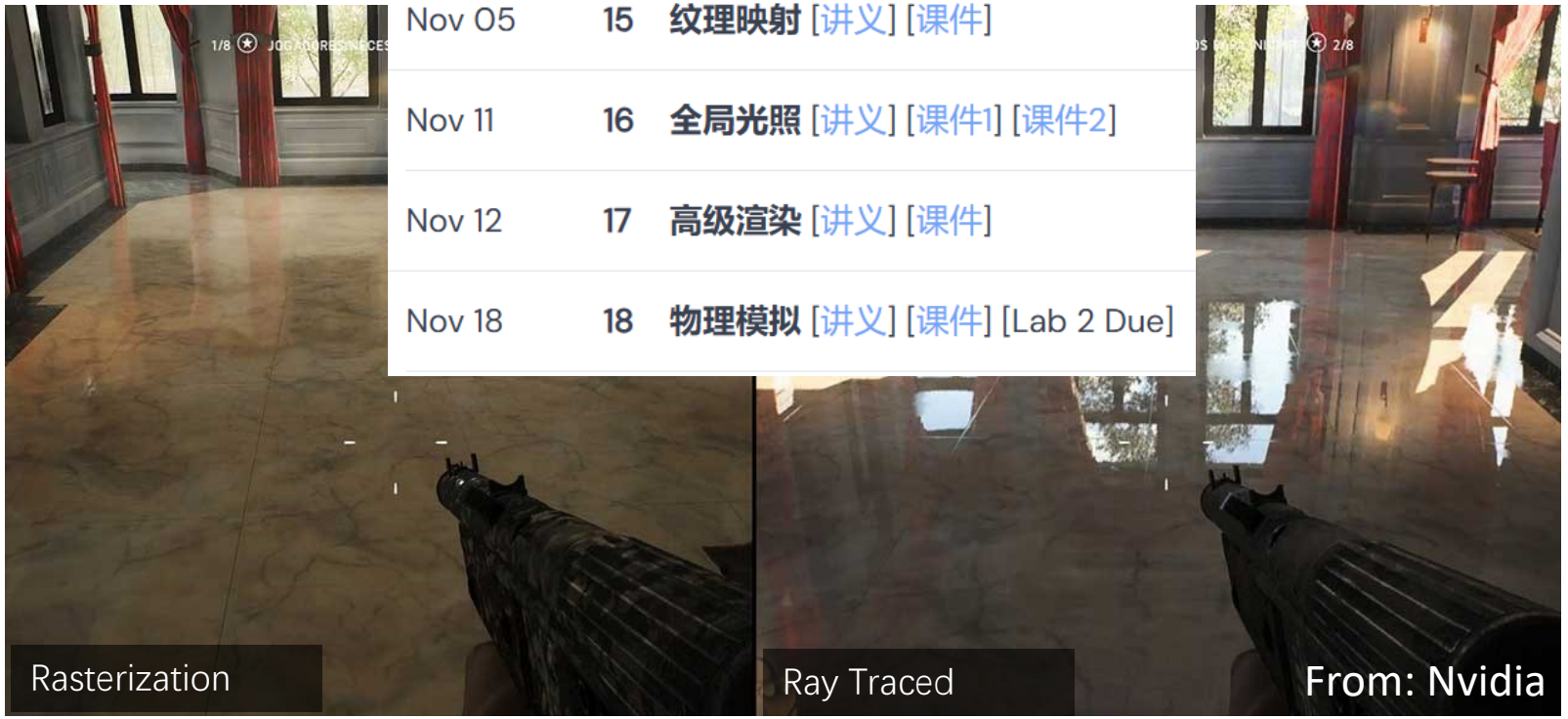
- Rasterization:
  - Analogy: Like painting—coloring objects
  - Method: Uses empirical lighting models (Shading, Illustration, Pipeline)

    Efficient & Lack realism
- Ray Tracing:
  - Analogy: Like photography
  - Method: Traces light interactions (reflection, refraction):

    Realism & Less efficient



Oct 28	12	**几何变换** [讲义] [课件]
Oct 29	13	**光照和着色** [讲义] [课件] [Test 1]
Nov 04	14	**渲染管线** [讲义] [课件]
Nov 05	15	**纹理映射** [讲义] [课件]
Nov 11	16	**全局光照** [讲义] [课件1] [课件2]
Nov 12	17	**高级渲染** [讲义] [课件]
Nov 18	18	**物理模拟** [讲义] [课件] [Lab 2 Due]

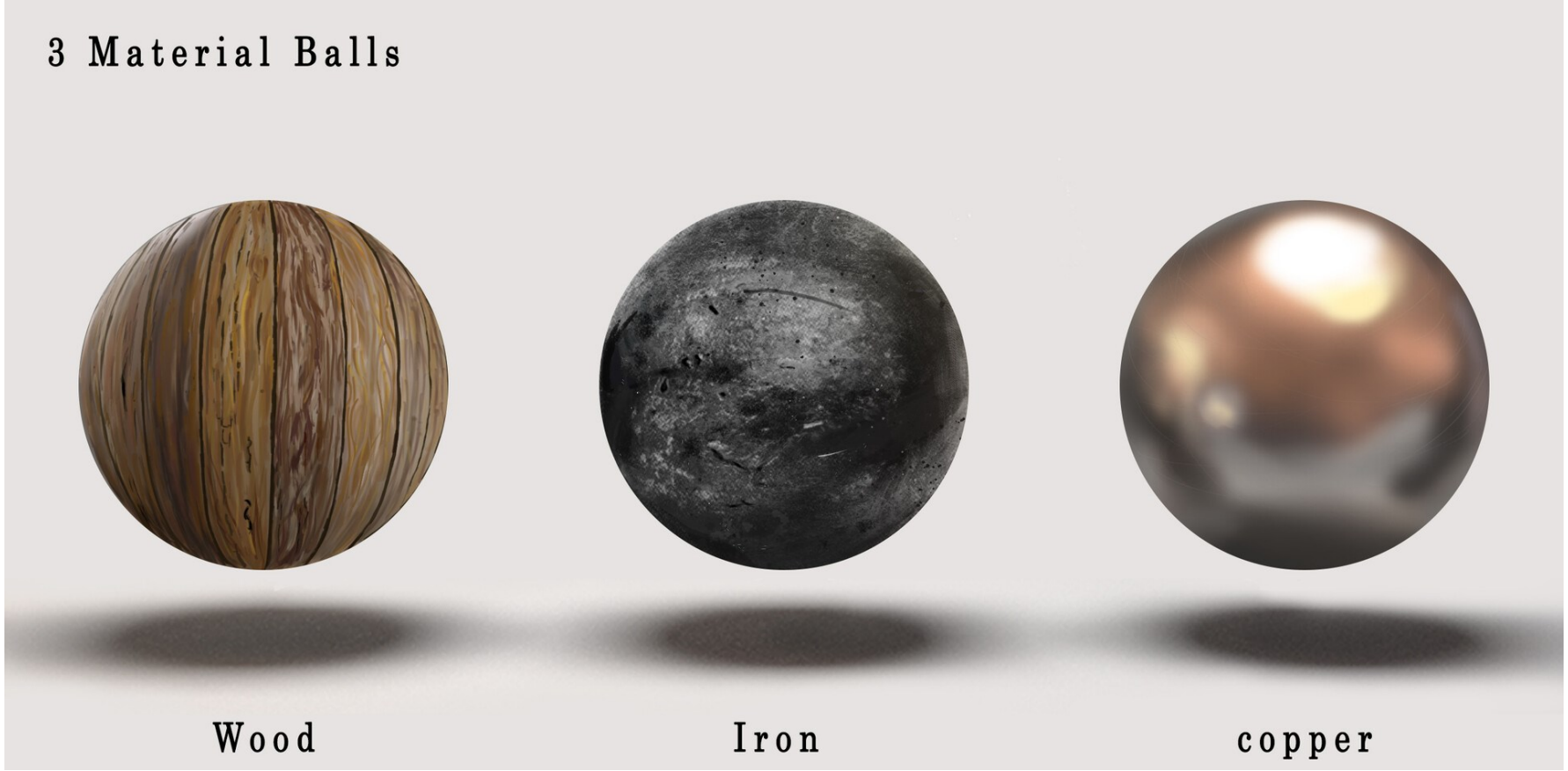


CG and Rendering

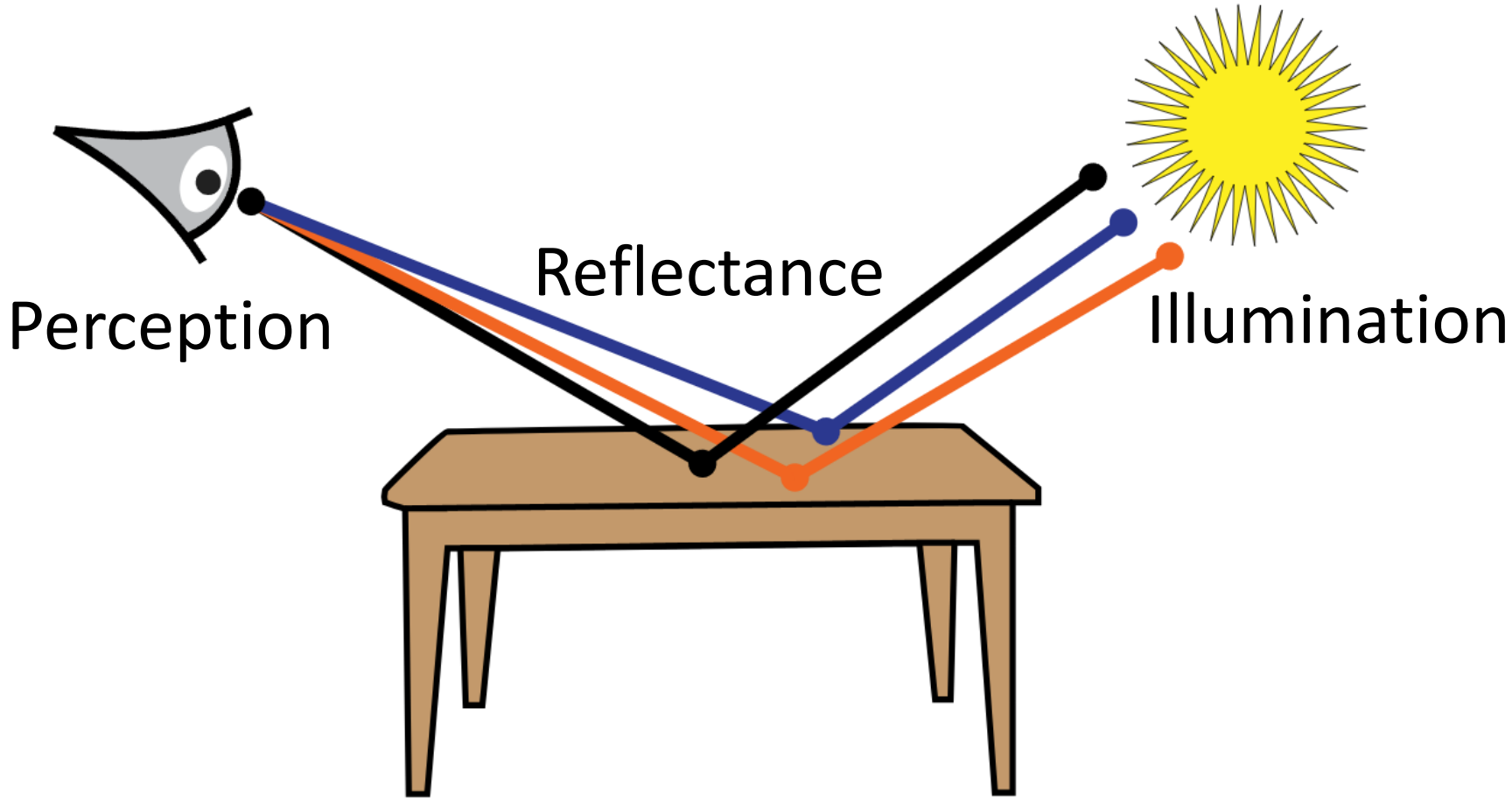
Shading

Shading

Shading in Graphics: Coloring a geometry to make it appear like a specified material with light response.



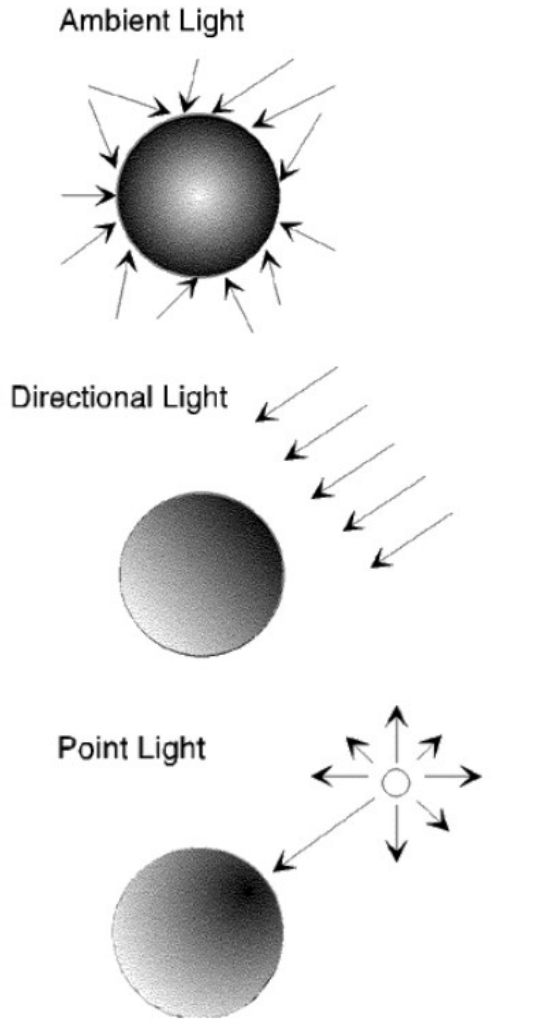
Physics: Light Transport



Light Sources

Types of Light Sources

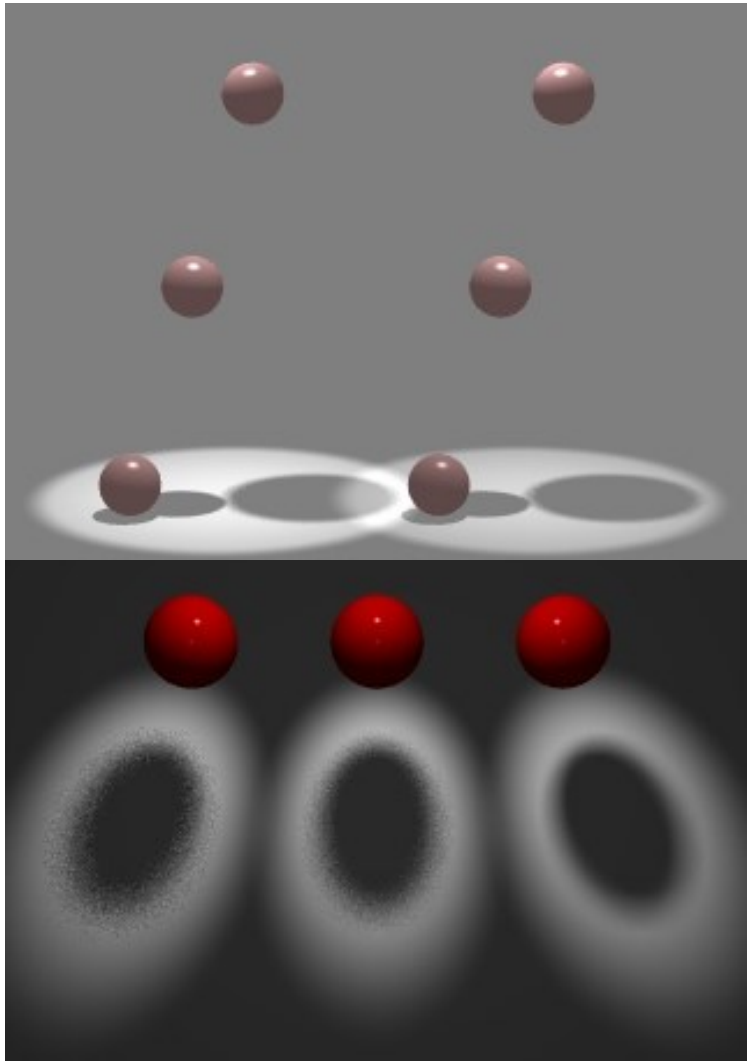
- Ambient: equal light in all directions
  - a hack to model inter-reflections
- Directional: light rays oriented in same direction
  - good for distance light sources (sunlight)
- Point: light rays diverge from a single point
  - approximation to a light bulb



From: https://www.okino.com/

More Light Sources

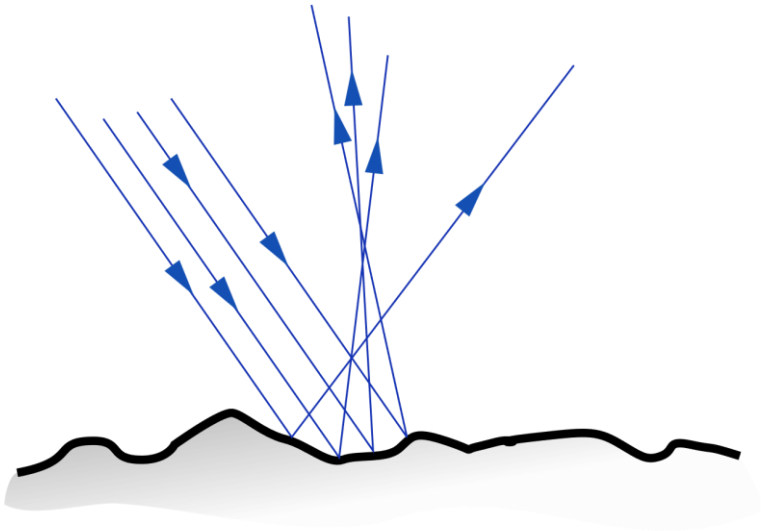
- Spotlight: point source with directional fall-off
  - intensity is maximal along some direction D, falls off away from D
  - specified by color, point, direction, fall-off parameters
- Area Source: Luminous 2D surface
  - radiates light from all points on its surface
  - generates soft shadows



Reflection

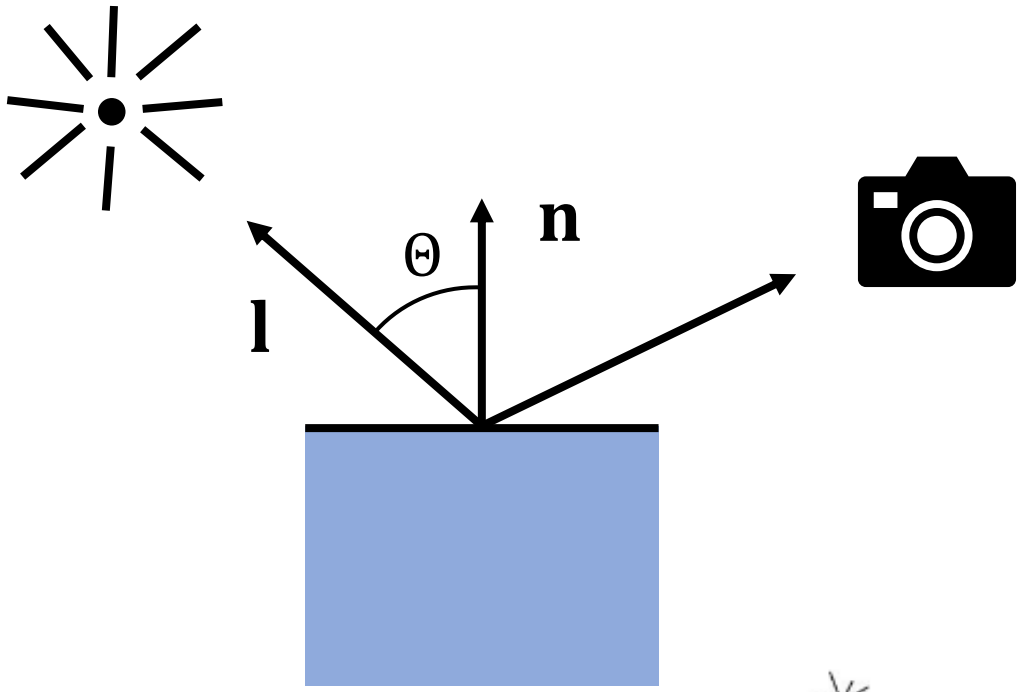
Diffuse Reflection

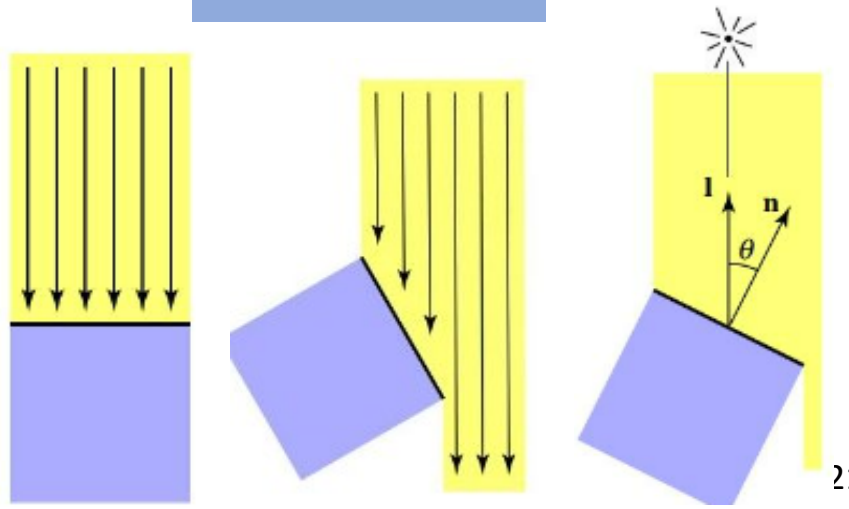
- Simplest kind of reflector (also known as Lambertian Reflection)
- Models a matte surface -- rough at the microscopic level
- Ideal diffuse reflector
  - incoming light is scattered equally in all directions
  - viewed brightness does not depend on viewing direction
  - brightness does depend on direction of illumination



Lambert's Law

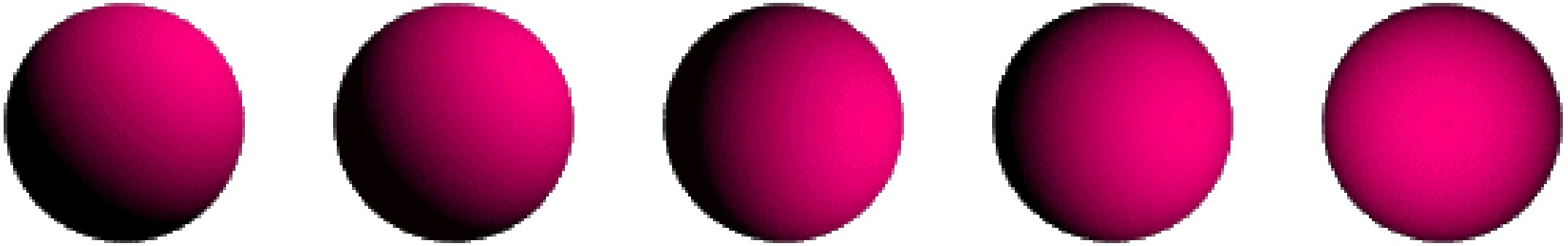
- $I_d = k_d I_l \cos\theta = k_d I_l (\boldsymbol{n} \cdot \boldsymbol{l})$
- I_d: diffuse reflection intensity
- k_d: reflectance coefficient in [0,1]
- I_l: light intensity
- θ: light/normal angle





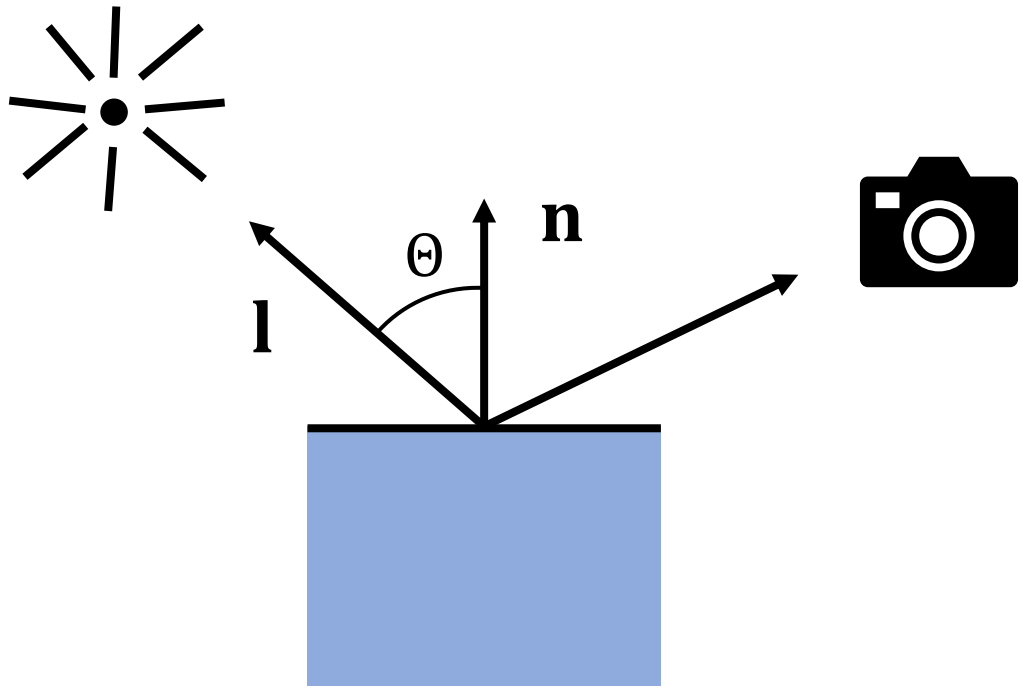
Examples of Diffuse Illumination

The same sphere lit diffusely from different lighting angles



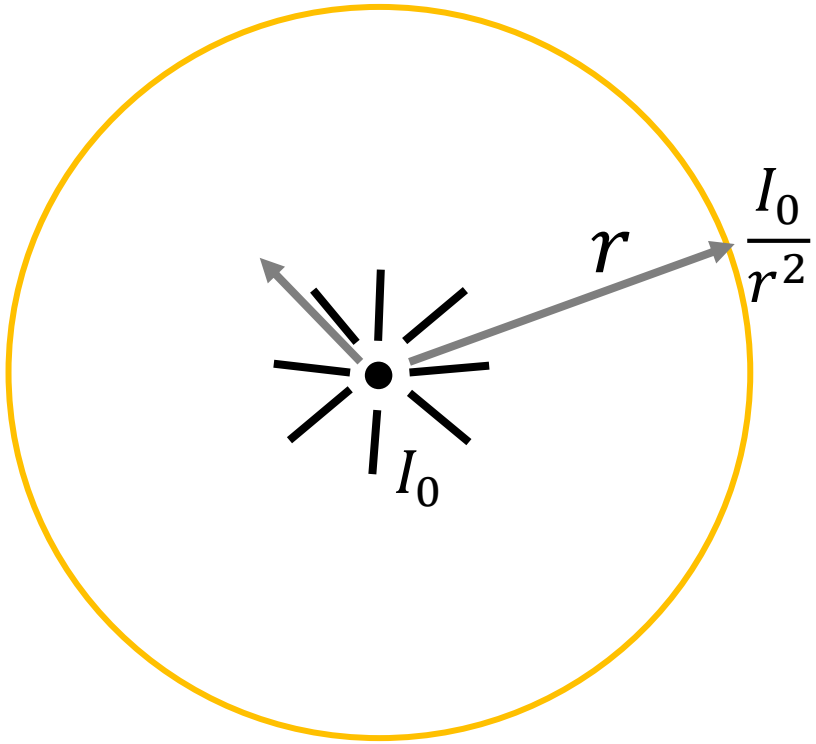
Ambient + Diffuse Reflection

- CG started using the Lambertian model and then added more terms as extra effects were required
- $I_{d+a} = k_a I_a + k_d I_l \cos\theta$
  - I_a: ambient light intensity
  - k_a: ambient reflectance
- This is diffuse illumination plus a simple ambient light term
  - a trick to account for a background light level caused by multiple reflections from all objects in the scene



Further Simple Illumination Effects

- Light attenuation:
  - light intensity falls off with the square of the distance from the source - so we add an extra term for this
  - $I_{d+a} = k_a I_a + f_{att} k_d I_l \cos\theta$, where $f_{att} = 1/d^2$, d is the light source to surface distance

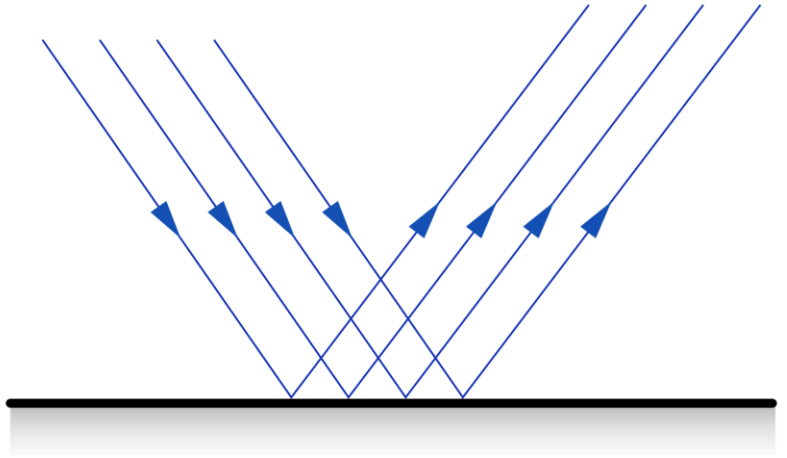


Further Simple Illumination Effects

- Light attenuation:
  - light intensity falls off with the square of the distance from the source - so we add an extra term for this
  - $I_{d+a} = k_a I_a + f_{att} k_d I_l \cos\theta$, where $f_{att} = 1/d^2$, d is the light source to surface distance
- Colored lights and surfaces:
  - just have three separate equations for RGB (or CIE, YIQ, etc.)
- Atmospheric attenuation:
  - use viewer-to-surface distance to give extra effects
  - the distance is used to blend the object's radiant color with a "far" color (e.g., a nice hazy gray)

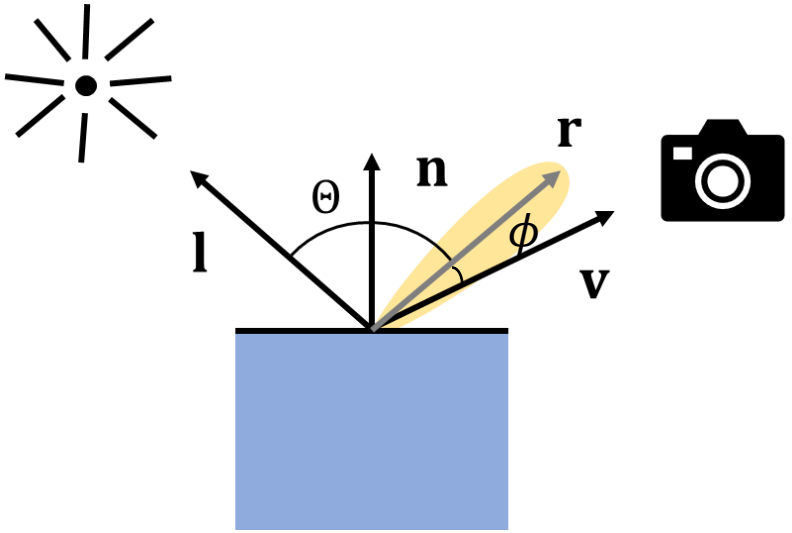
Specular Reflection

- Shiny surfaces change appearance when viewpoint is varied
  - specularities (highlights) are view-dependent
  - caused by surfaces that are microscopically smooth
- For shiny surfaces part of the incident light reflects coherently
  - an incoming ray is reflected in a single direction (or narrow beam)
  - direction is defined by the incoming direction and the surface normal
- A mirror is a perfect specular reflector
  - approximate specular reflectors give fuzzy highlights

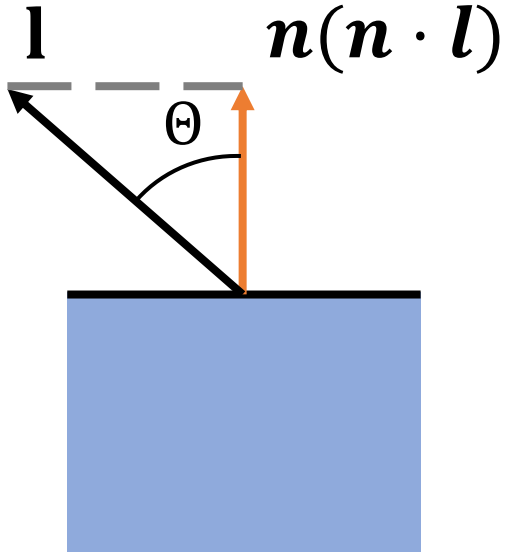


Phong Illumination

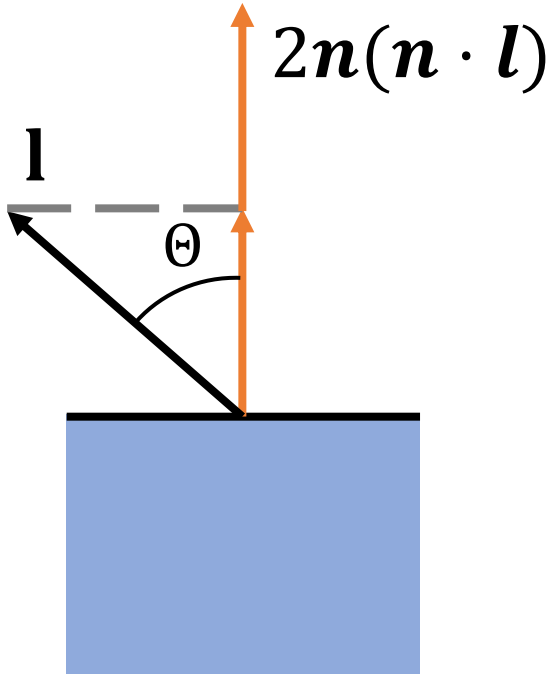
- One function that approximates specular falloff is called the Phong Illumination model
- $I_s = k_s I_l \, (\cos\phi)^{n_s}$
  - ϕ: angle between reflected light ray and viewer
  - k_s: specular reflectance
  - n_s: rate of specular falloff
- no physical basis, yet widespread use in computer graphics



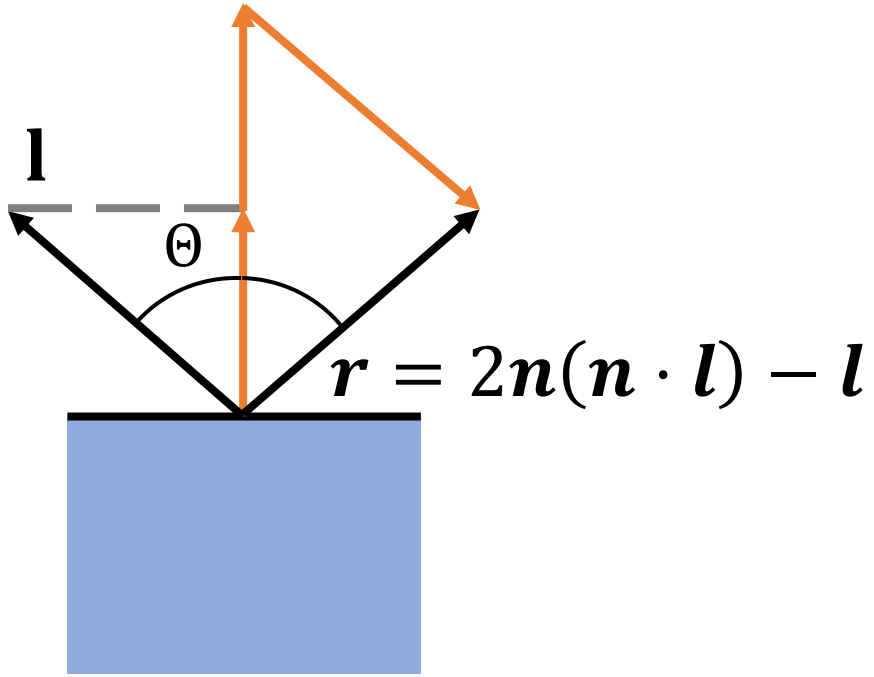
Computing the Reflected Ray



Project l onto n



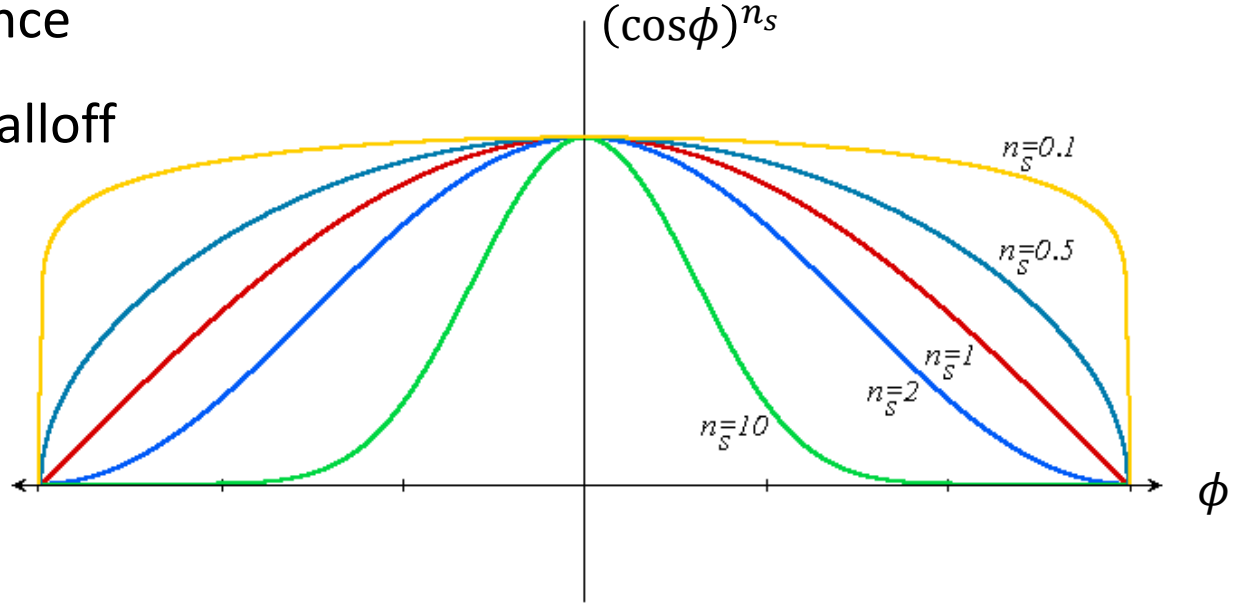
Double length



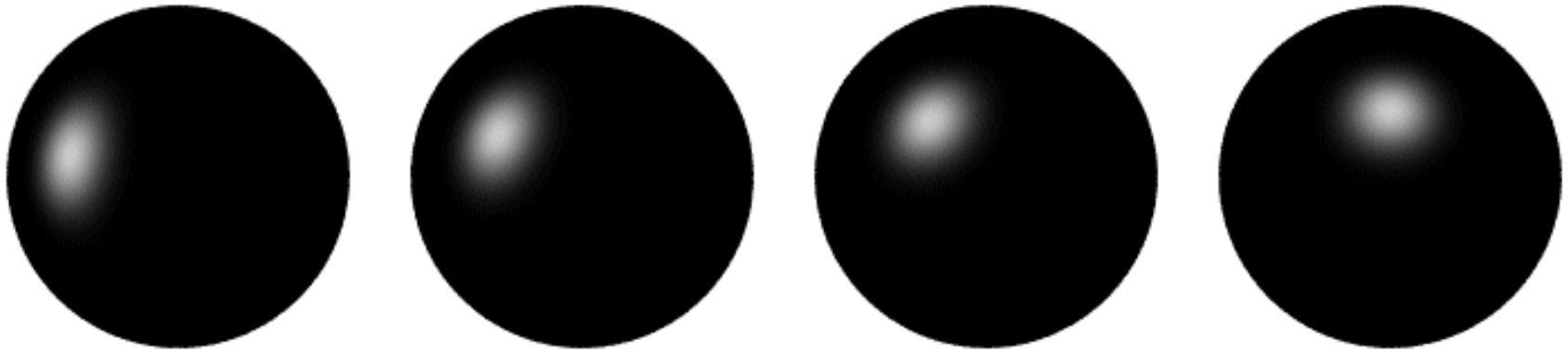
Subtract l

Phong Illumination Curves

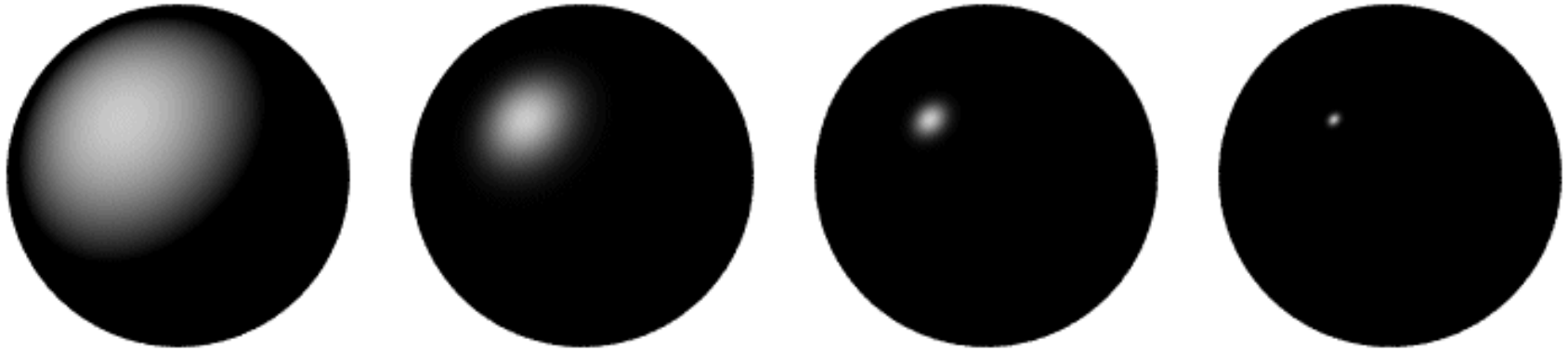
- The specular exponents are often much larger than 1; values of 100 are not uncommon.
- $I_s = k_s I_l \, (\cos\phi)^{n_s}$
  - ϕ: angle between reflected light ray and viewer
  - k_s: specular reflectance
  - n_s: rate of specular falloff



Phong Illumination



Moving the light source



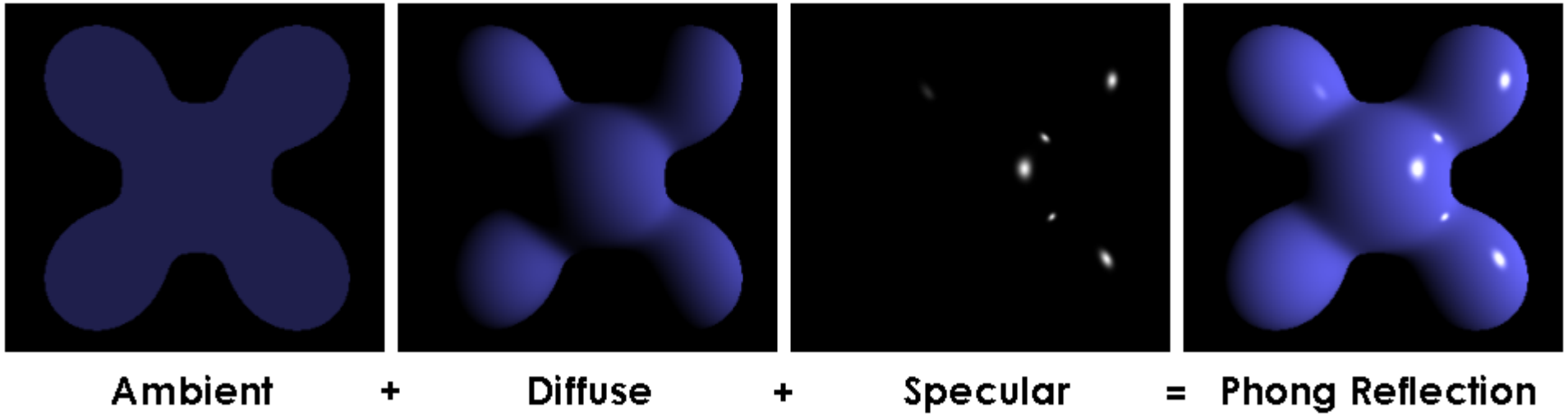
Changing n_S

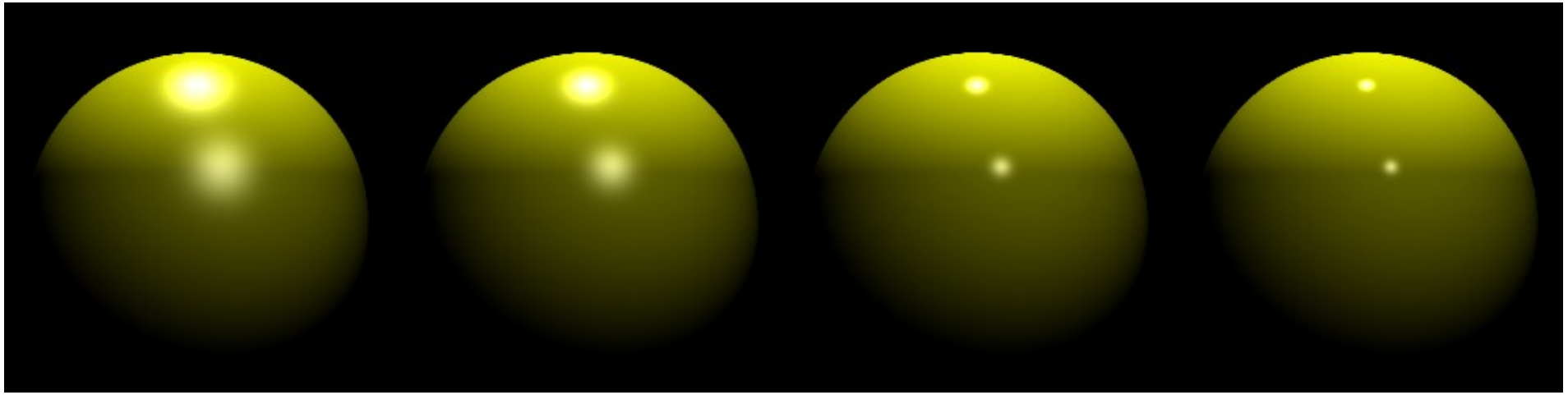
Putting It All Together

$$I = k_a I_a + f_{att} I_l (k_d \max(0, \cos\theta) + k_s (\cos\phi)^{n_s})$$

- Combining ambient, diffuse, and specular illumination
- For multiple light sources
  - Repeat the diffuse and specular calculations for each light source
  - Add the components from all light sources
  - The ambient term contributes only once
- The different reflectance coefficients can differ.
  - Simple “metal”: ka and kd share material color, ks is white
  - Simple plastic: ks also includes material color
- Remember, when cosine is negative lighting term is zero!

Some Examples





Shading

Where do we Illuminate ?

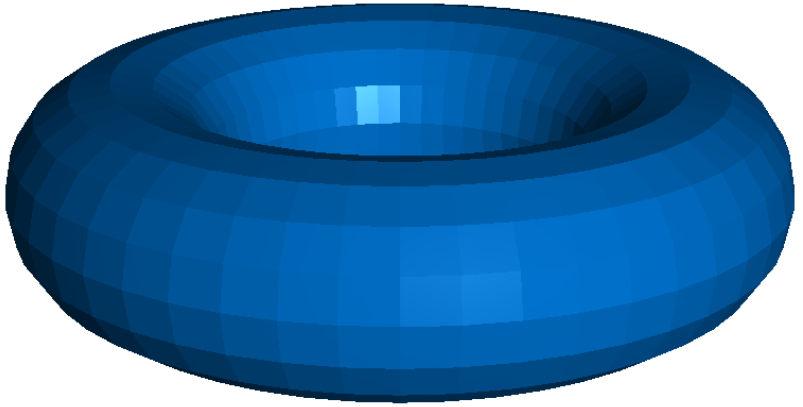
To this point, we have discussed how to compute an illumination model at a single point on a surface. But, at which points on the surface is the illumination model applied? Where and how often it is applied has a noticeable effect on the result.

Illuminating can be a costly process involving the computation of and normalizing of vectors to multiple light sources and the viewer.

Flat Shading

- The simplest shading method applies only one illumination calculation at one point for each primitive. Which one? Usually the centroid. For a convex facet the centroid is given as follows:

- $centroid = \frac{1}{vertices}\sum_{i=1}^{vertices} \boldsymbol{p}_i$



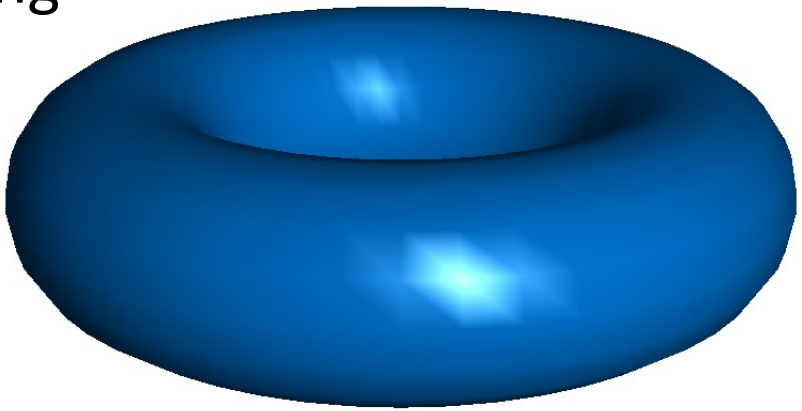
- This technique is called *constant or flat shading*. It is often used on polygonal primitives.

- Facts:
  - For point light sources the direction to the light source varies over the facet
  - For specular reflections the direction to the eye varies over the facet
  - Facet facts are clearly visible

Gouraud Shading

The *Gouraud Shading* method applies the illumination model on a subset of surface points and interpolates the intensity of the remaining points on the surface. In the case of a polygonal mesh the illumination model is usually applied at each vertex and the colors in the triangles interior are linearly interpolated from these vertex values during polygon rasterization.

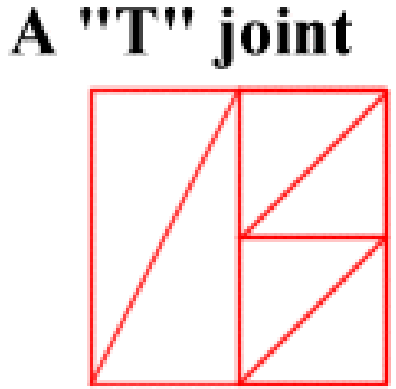
Henri Gouraud



Facet artifacts are still visible

Gouraud Shading

- Gouraud shading interpolates illumination values at each vertex using screen-space interpolation.
- Screen-space interpolation is wrong!
- However, you usually will not notice because the transition in colors is very smooth.
- There are some cases where the errors in Gouraud shading become obvious.
  - When switching between different levels-of-detail representations
  - At "T" joints.

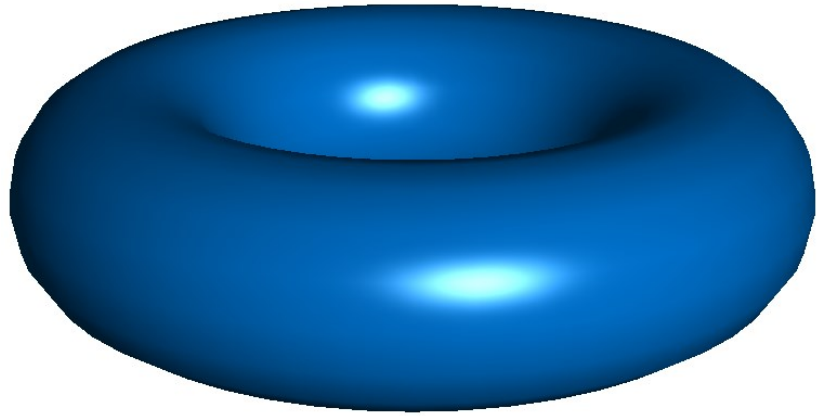


Phong Shading

In *Phong shading* (not to be confused with Phong's illumination model), the surface normal is linearly interpolated across polygonal facets, and the Illumination model is applied at every point.

Bui Tuong Phong (裴祥風)

A Phong shader assumes the same input as a Gouraud shader, which means that it expects a normal for every vertex. The illumination model is applied at every point on the surface being rendered, where the normal at each point is the result of linearly interpolating the vertex normals defined at each vertex of the triangle (in the world/model space).

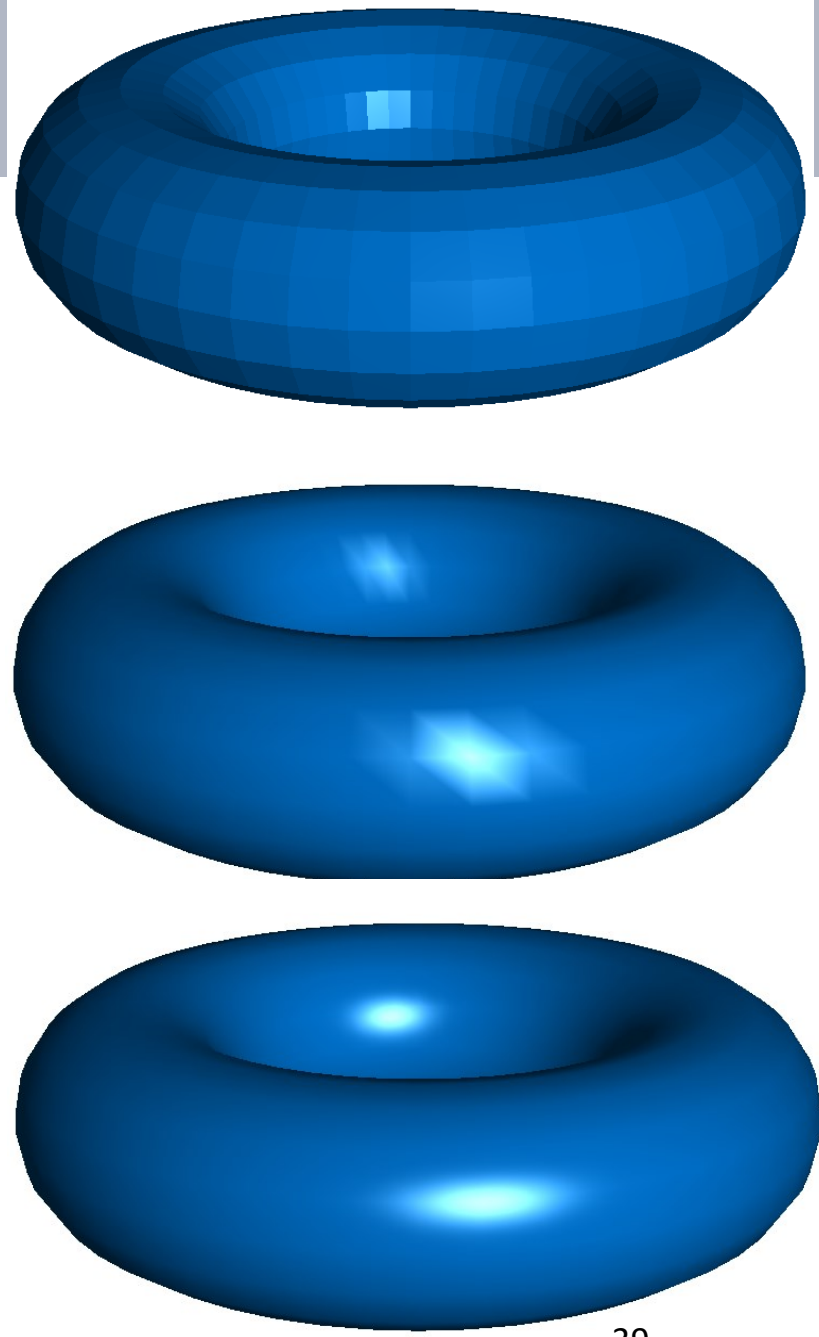


Phong shading will usually result in a very smooth appearance,

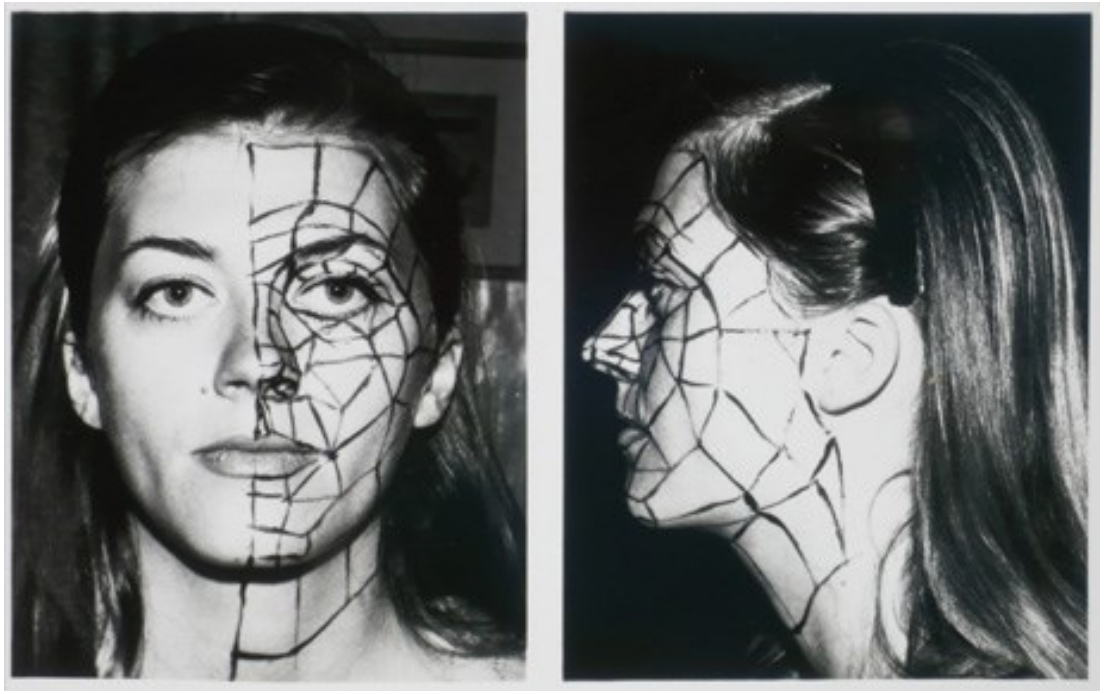
facet artifacts can only be seen along silhouettes.

Comparison

faceted shading Gouraud shading Phong shading



"Images de synthèse : palme de la longévité pour l'ombrage de Gouraud" 1971



Visage de Sylvie Gouraud pendant le relevé de points.

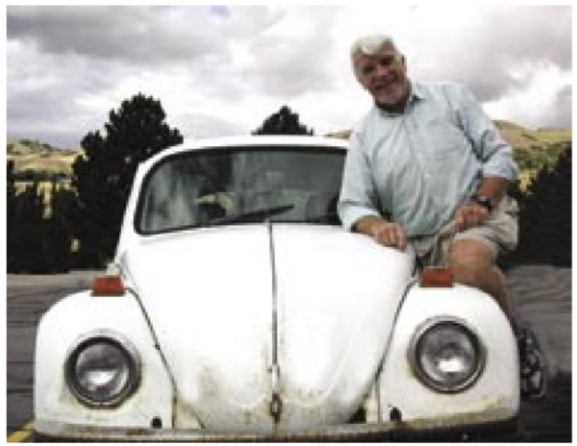
Reconstruction des facettes à l'ordinateur, à gauche, puis lissage avec ombrage de Gouraud, à droite.

Robert Remembers: the VW Bug

By Robert McDermott

The year-1972, the teacher-Ivan Sutherland, the objective-create a rendered model that would achieve instant recognition of a physical icon.

In the infancy of computer graphics, and in my first year of study, a University of Utah graduate course began a



project that resulted in a computational benchmark and a small piece of computer graphics history.

Sutherland challenged his students to choose something iconic to realistically render. We selected the Volkswagen Beetle—as a symbol of global culture, because it was large enough to measure as a group, and because Ivan's wife, Marsha, owned one.

the x, y, and z coordinates of the painted points on the car surface. The Beetle was assumed to have left to right symmetry so we measured only half of the car. The process was slow and tedious, taking many class sessions to complete. Marsha was a wonderful sport as she drove the car around town festooned with our markings.

After each measurement session, we entered by hand the lists of point coordinates into text data files. The lists of polygons were also entered as lists of integer indices referencing the sets of points. The system used to render the files included hardware developed by Gary Watkins (BS '67, PhD '70) to imprint shaded images onto a direct film recorder, taking several minutes per photographic image. Polaroid film was used to get pictures as quickly as possible for the assignment and later 4x5 sheet negatives were produced to get higher quality images. Mike Milochek developed the sheet film and printed photographic images over stretches of days at a time.

At this time, the software to render

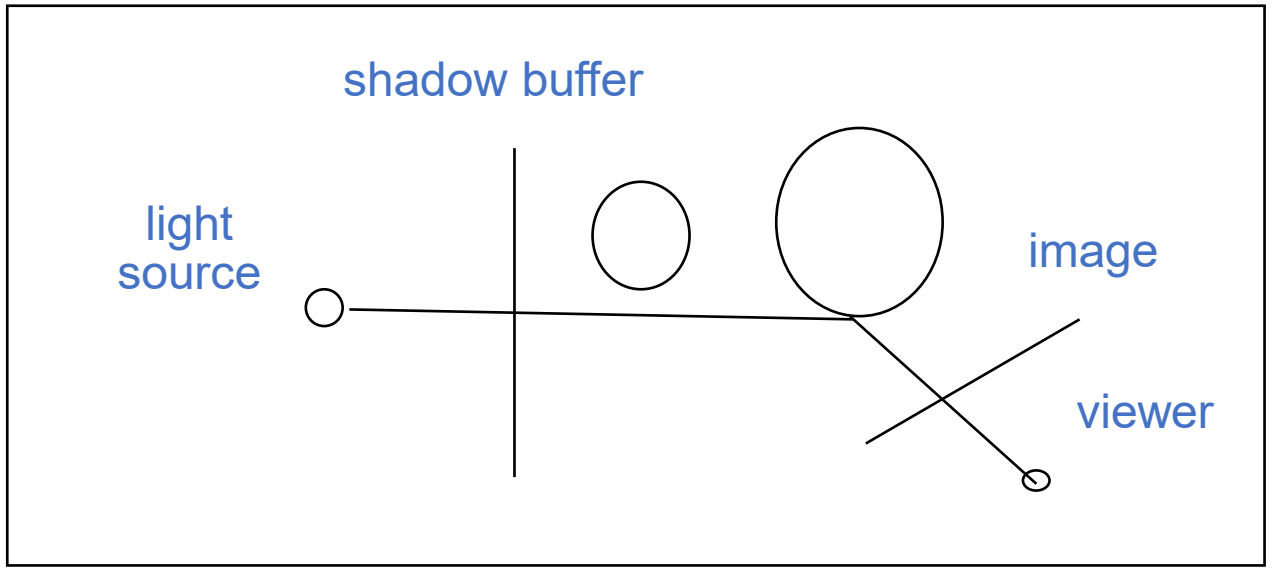
More About Shading

- Approximating Phong Shading using Gouraud Shading?

- Texture mapping gets you further
  - Assign radiance based on an image

Shadows

- Shadows occur where objects are hidden from a light source
  - omit any intensity contribution from hidden light sources
- Working out what is hidden is simply a visibility problem
  - can the light source see the object?
  - use the z-buffer shadow algorithm:
    - run the algorithm from the light source's viewpoint
    - save the z-buffer as the shadow buffer
    - run the real z-buffer algorithm, transforming each point into the light source's coordinates and comparing the z value against the shadow buffer
    - (Yeah, I know we haven't done z-buffers yet)

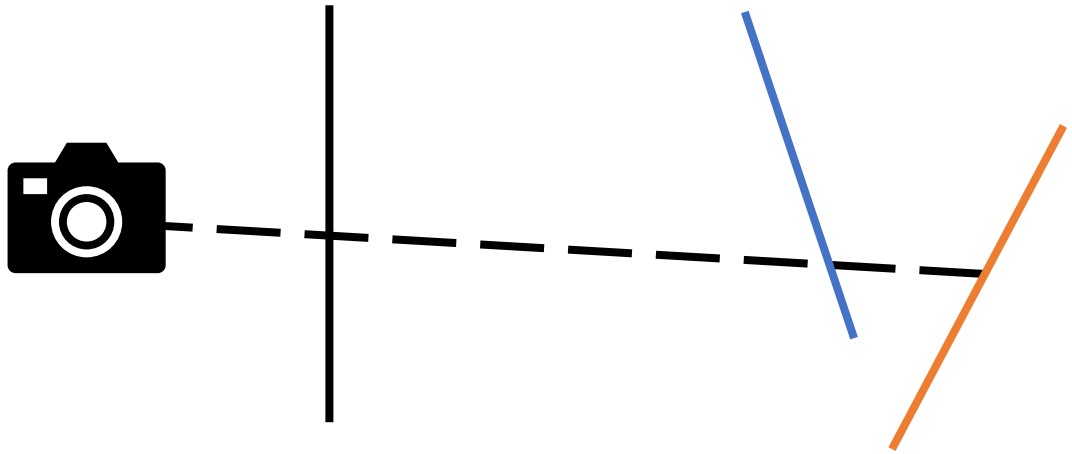


Visibility

The Visibility Problem

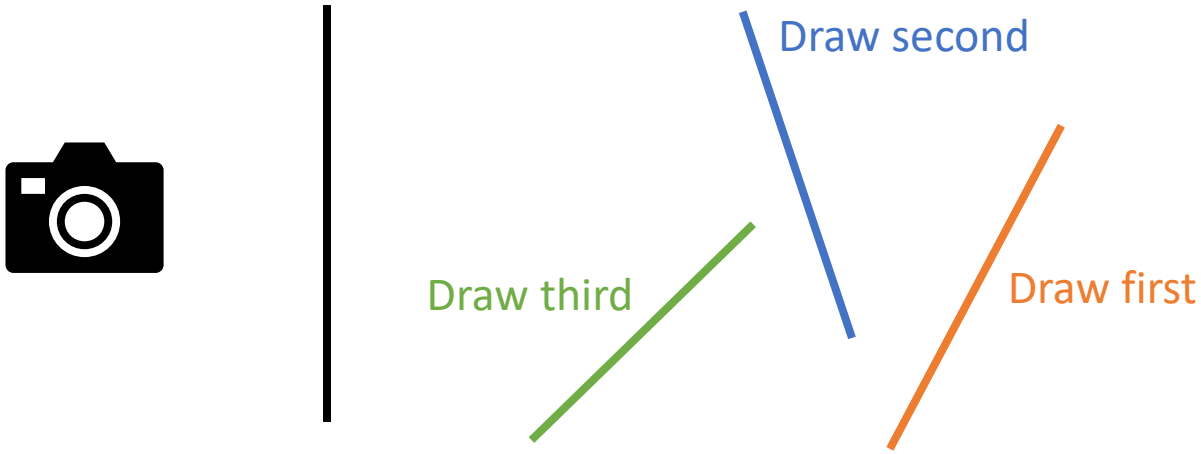
What is the nearest surface seen at any point in the image?

How would YOU solve this problem?



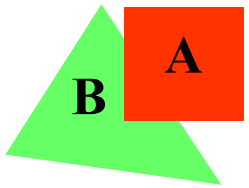
Painter's Algorithm

- Sort objects by depth (z)
- Loop over objects in back-to-front order
  - Project to image
  - scan convert: image[x,y] = shade(x,y)

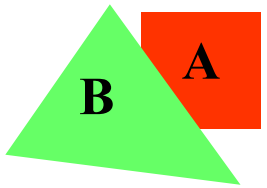


Sorting Objects by Depth

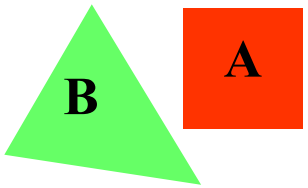
Depth ordering is a *partial ordering.* Outcomes are



A occludes B
draw B before A

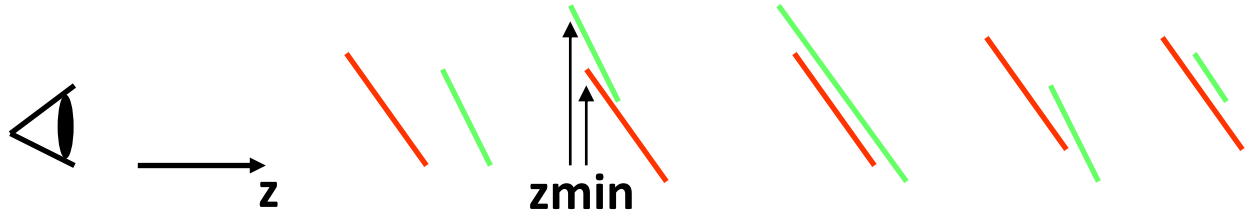


B occludes A
draw A before B

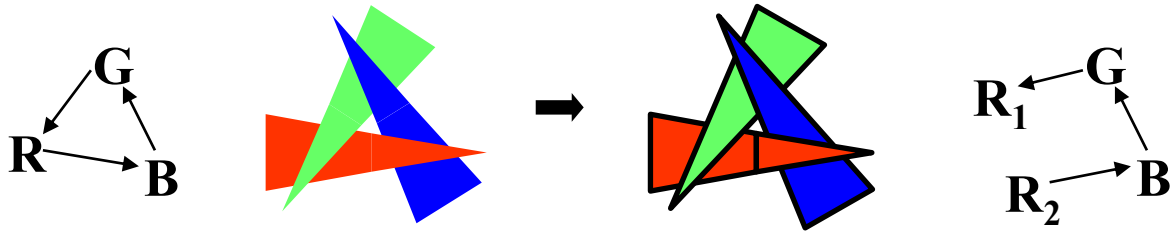


don't care
draw in either order

Sorting objects by their zmin doesn't always work! (same for zmax)



Sometimes ordering is cyclic! What to do? Split objects!



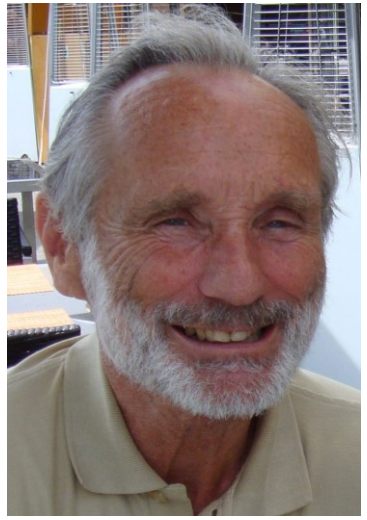
Painter's Algorithm

- Strengths
  - Simplicity: draw objects one-at-a-time, scan convert each
  - Handles transparency well
- Drawbacks
  - Sorting can be expensive (slower than linear in the number of objects)
  - Clumsy when ordering is cyclic, because of need to split
  - Interpenetrating polygons need to be split, too
  - Hard to sort non-polygonal objects
- Sometimes no need to sort, or trivial
  - If objects are arranged in a grid, e.g. triangles in a height field z(x,y), such as a triangulated terrain

Z-Buffer Algorithm (1974)

- Initialization

```
loop over all x,y
    zbuf[x,y] = infinity
```



Wolfgang Straßer

Edwin "Ed" Catmull

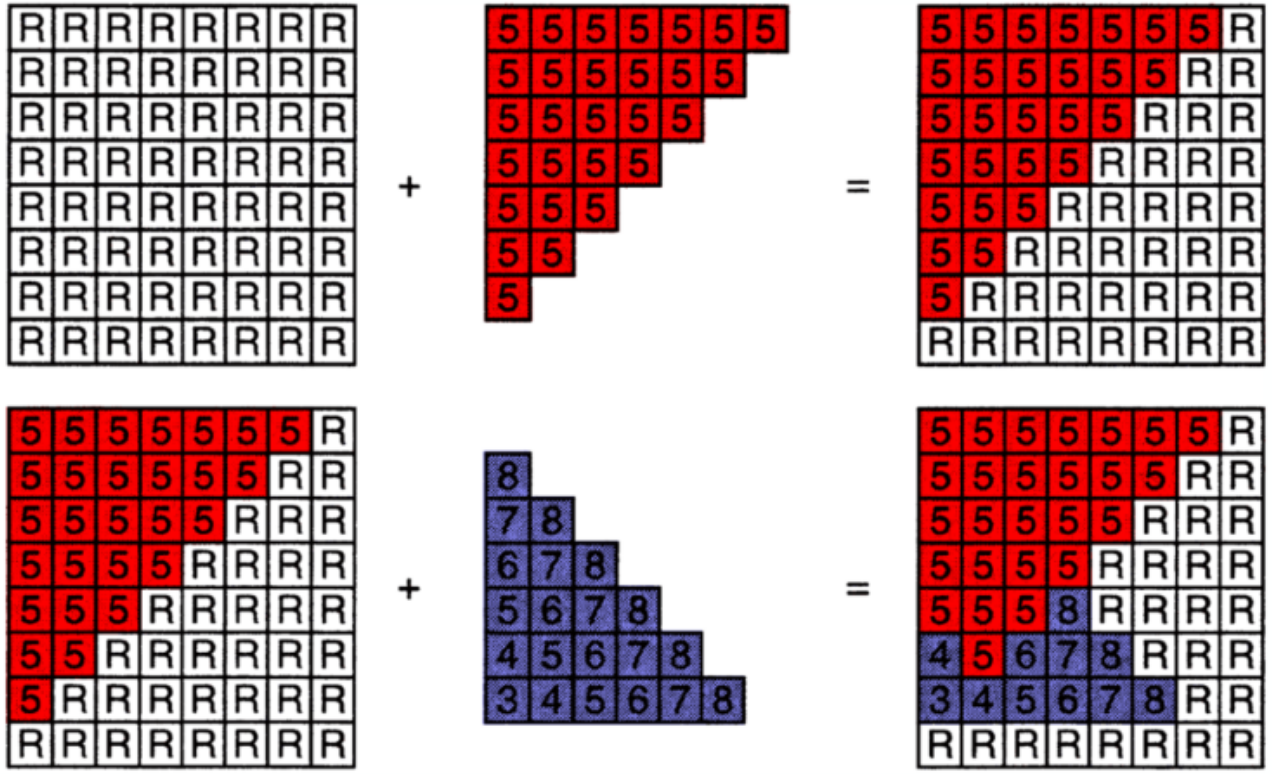
- Drawing steps

```
loop over all objects
    scan convert object (loop over x,y)
        if z(x,y) < zbuf[x,y]        /* compute z of this object at this pixel & test */
            zbuf[x,y] = z(x,y)       /* update z-buffer */
            image[x,y] = shade(x,y)  /* update image (typically RGB) */
```

Z-Buffer Algorithm (1974)

- Initialization

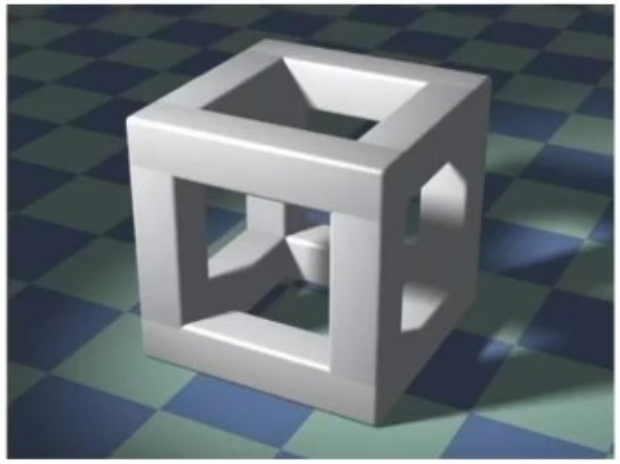
```
loop over all x,y
    zbuf[x,y] = infinity
```



- Drawing steps

```
loop over all objects
    scan convert object (loop over x,y)
        if z(x,y) < zbuf[x,y]        /* compute z of this object at this pixel & test */
            zbuf[x,y] = z(x,y)       /* update z-buffer */
            image[x,y] = shade(x,y)  /* update image (typically RGB) */
```

Z-Buffer Algorithm



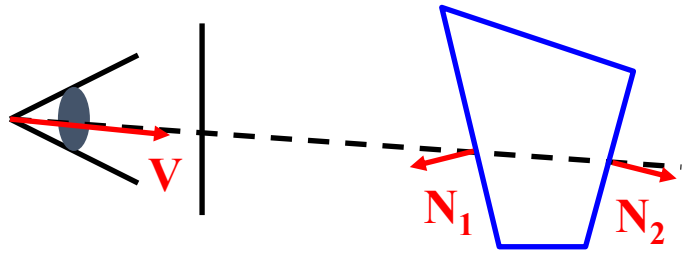
- Strengths
  - Simple, no sorting or splitting
  - Easy to mix polygons, spheres, other geometric primitives
- Drawbacks
  - Can’t handle transparency well
  - Need good Z-buffer resolution or you get depth ordering artifacts
    (16 bits/pixel of “z”)

Really Hard Visibility Problems

- Extremely high scene complexity
  - a building walkthrough
  - A fly-by of any outdoor scene
- Z-buffering requires drawing EVERY triangle for each image
  - Not feasible in real time
- Usually Z-buffering is combined with spatial data structures
  - BSP trees are common (similar concept to octrees, later!)
- For really complex scenes, visibility isn't always enough
  - Objects WAY in the distance are too small to matter
  - Might as well approximate far-off objects with simpler primitives
  - This is called geometry simplification – another big subject!

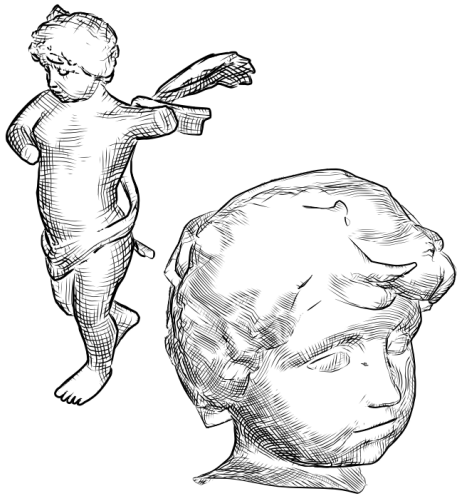
Backface Culling

- Each polygon is either front-facing or back-facing
  - A polygon is backfacing if its normal points away from the viewer,
  - i.e. V•N >= 0
- When it works
  - If object is closed, back faces are never visible so no need to render them
  - Easy way to eliminate half your polygons
  - Can be used with both z-buffer and painter's algorithms
  - If object is convex, backface culling is a complete visibility algorithm!
- When it doesn't work
  - If objects are not closed, back faces might be visible



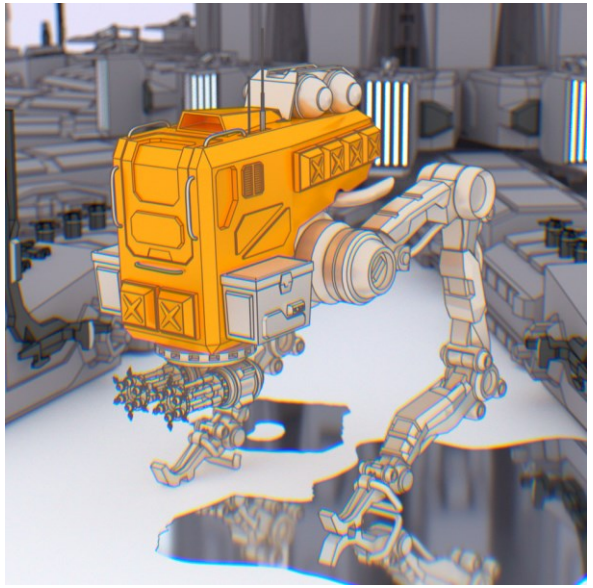
Non-Photorealistic Rendering

NPR



[Hertzmann et.al 2000]

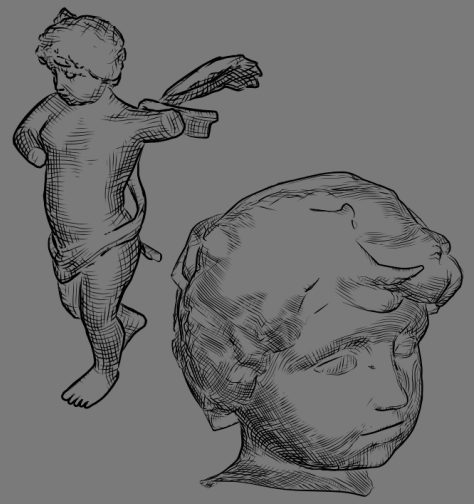
© miHoYo Genshin Impact



[Rex West 2021]

© Riot Arcane

NPR



[Hertzmann et.al 2000]

Line

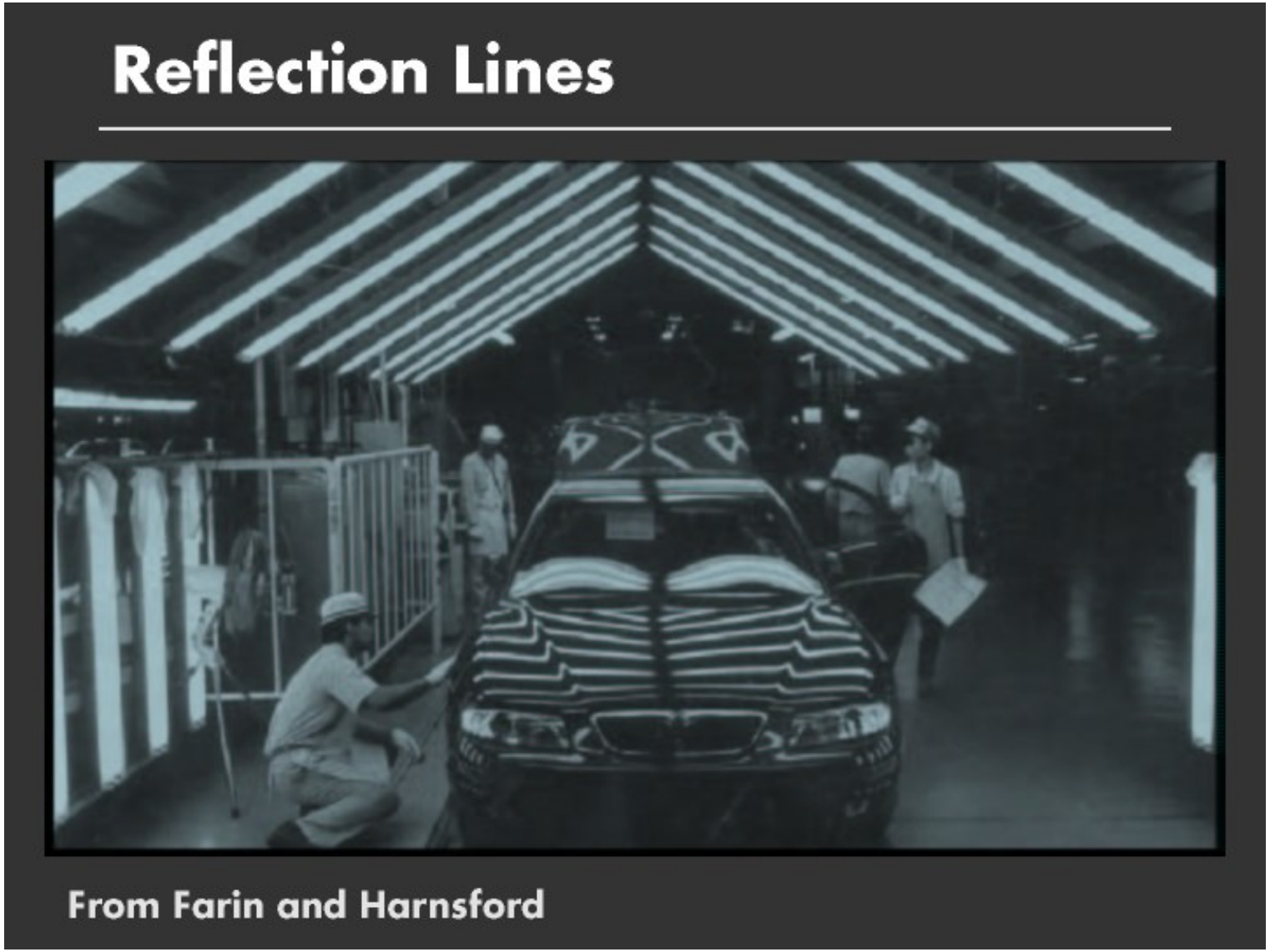
[Rex West 2021]

© miHoYo Genshin Impact

+ Shading

© Riot Arcane

Line



From Pat Hanrahan

Line

Feature Detectors!

Picasso, Portrait of Igor Stravinsky, 1920.

Graphite and charcoal, Musée Picasso, Paris, France

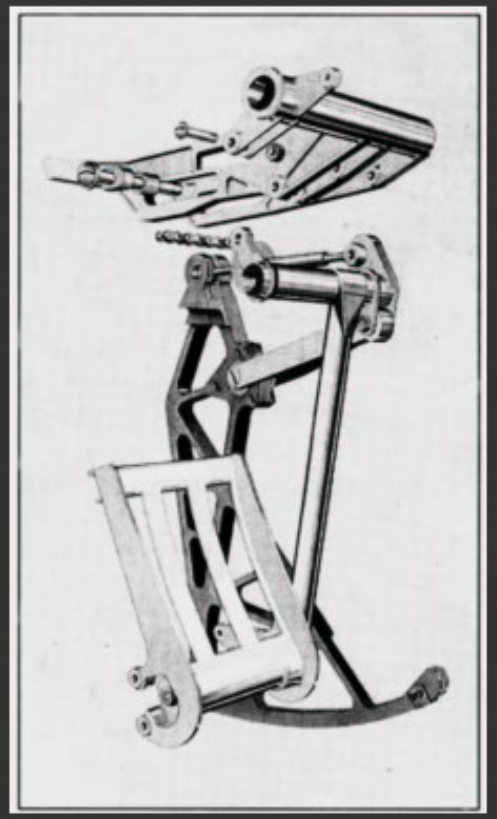
From Pat Hanrahan

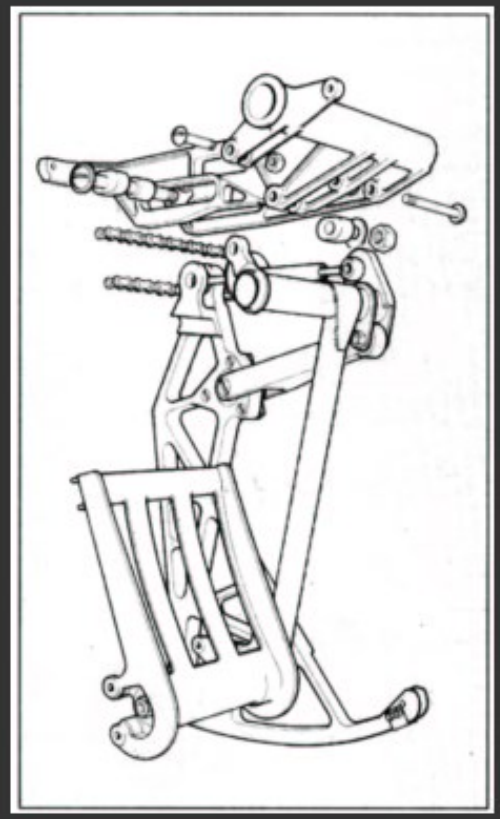
Line

Conveying Shape

Shading

Lines



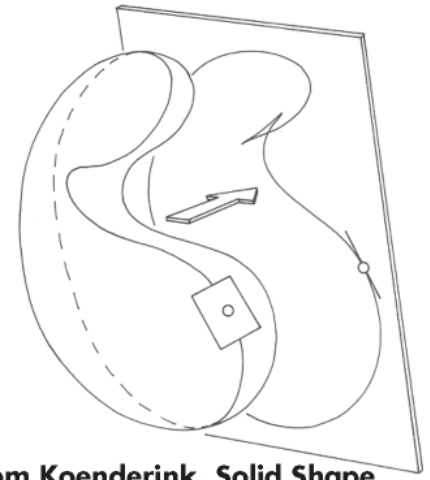


From Gooch[2]

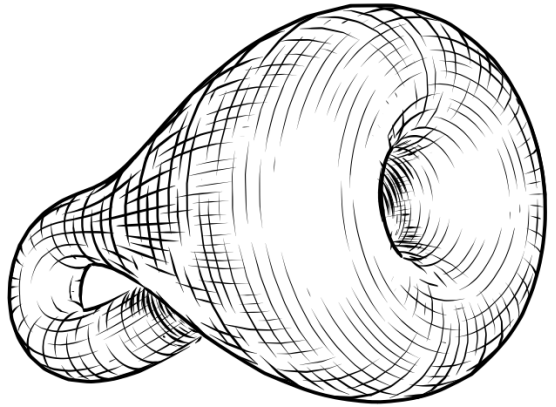
From Pat Hanrahan

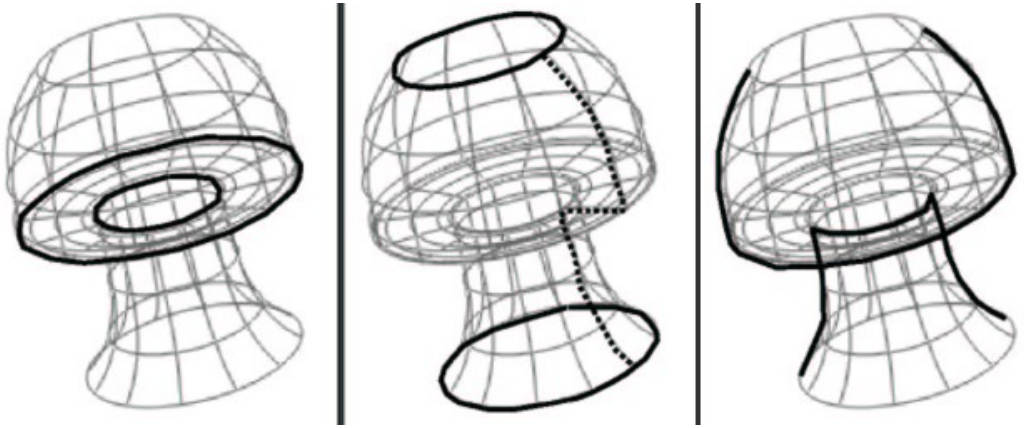
Types of Lines

- Discontinuities: creases and self-intersections
- Boundaries between surface patches
- Silhouettes and contours and cusps
- Parabolic lines
- Isoparametric lines
- Lines of curvature
- Attached and unattached shadows
- Isoluminance and luminance extrema
- Highlights
- …



From Koenderink, Solid Shape

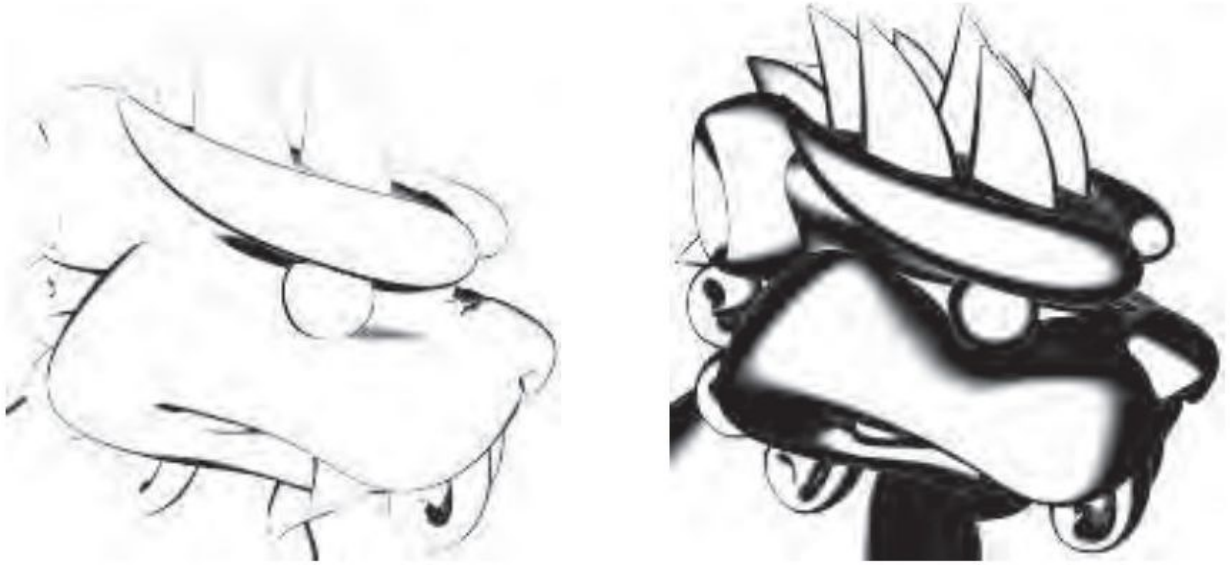




From Pat Hanrahan

Example: Contour Extraction

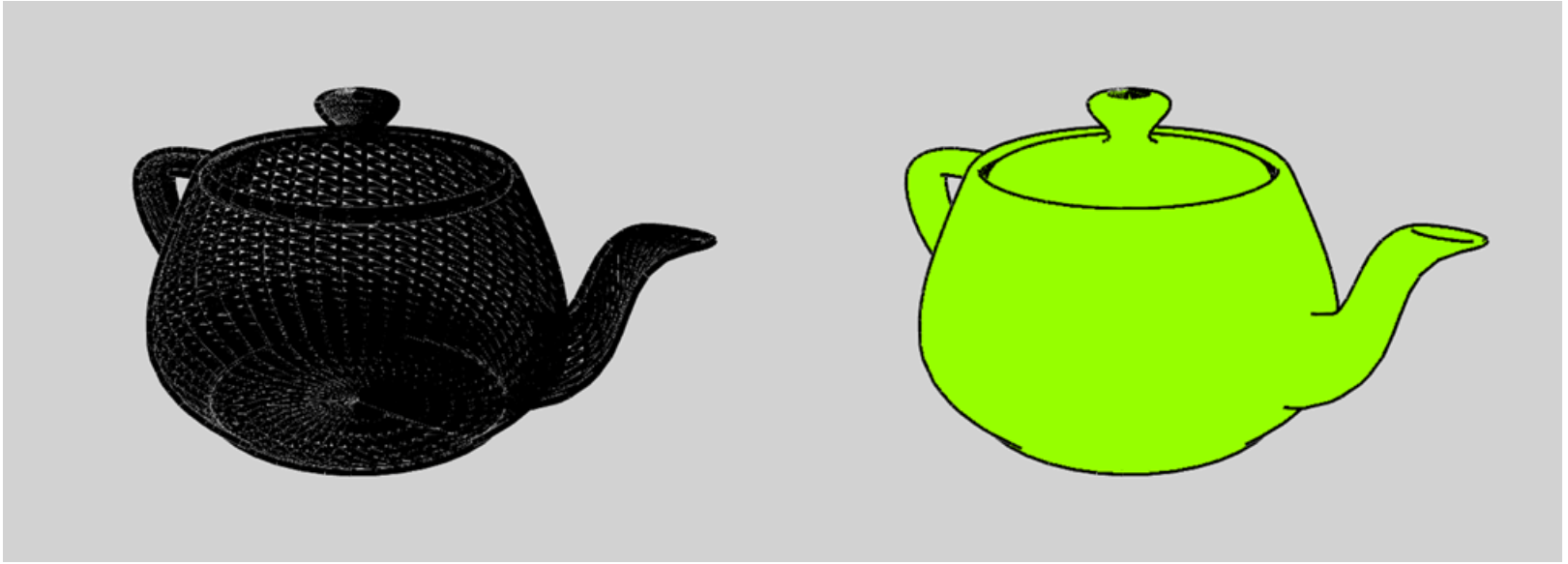
- Method 1: Shading Normal Contour Edges
  - View direction is near tangent to normal at edges
  - Line width is hard to control



From Real-Time Rendering 4th Edition

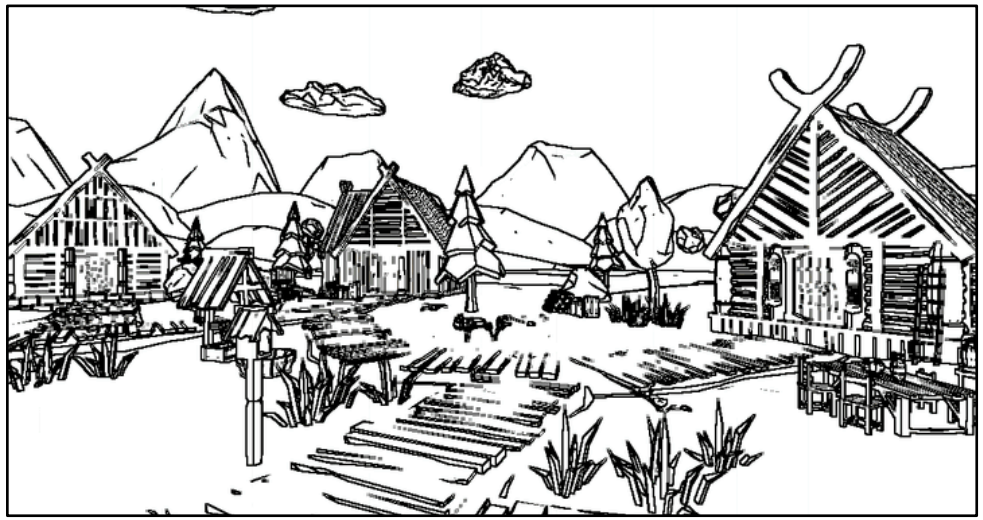
Example: Contour Extraction

- Method 1: Shading Normal Contour Edges
- Method 2: Procedural Geometry Silhouetting
  - Render a slightly bigger model by shifting the geometry
  - Compose to the original rendering result with z-buffer



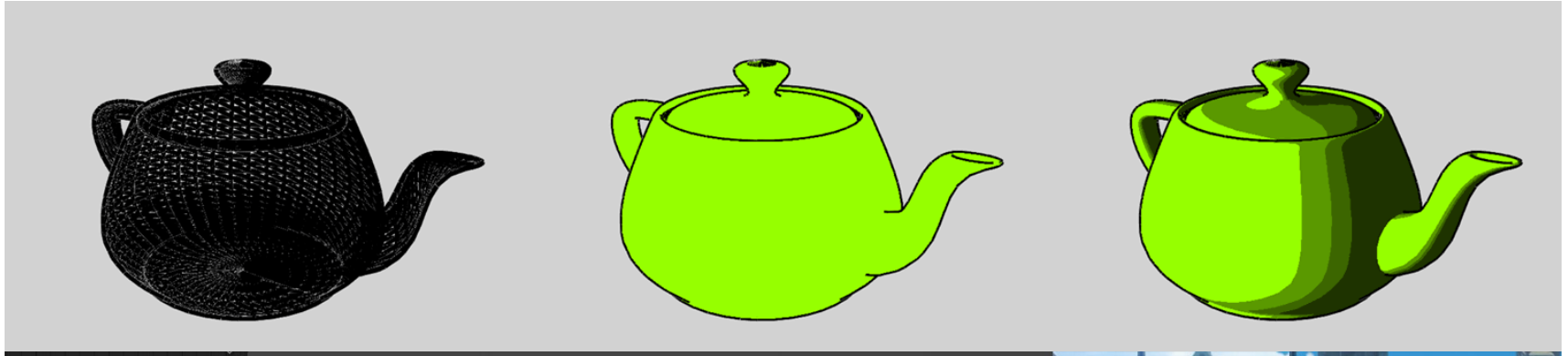
Example: Contour Extraction

- Method 1: Shading Normal Contour Edges
- Method 2: Procedural Geometry Silhouetting
- Method 3: Edge Detection by Image Processing



Toon(Cel) Shading

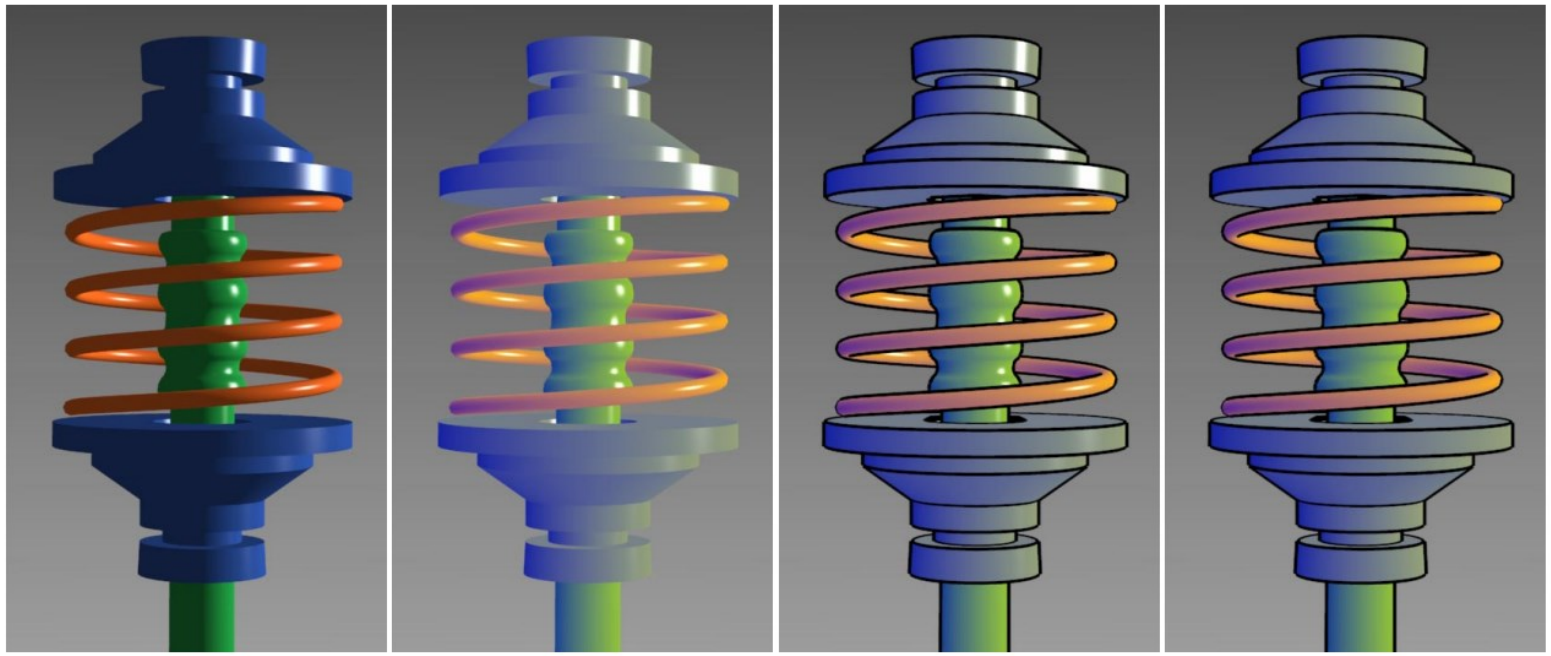
Limited number of colors: mimic comic book/cartoon



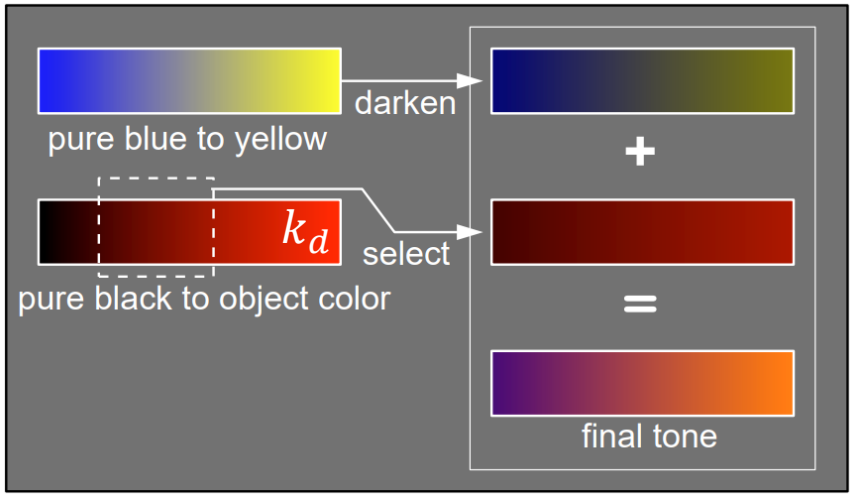
Gooch shading – tone-based shading of matte objects

- Cold-to-warm replaces dark-to-bright

  gooch98.pdf



$$I = \left(\frac{1 + \hat{\boldsymbol{l}} \cdot \hat{\boldsymbol{n}}}{2}\right) k_{cool} + \left(1 - \frac{1 + \hat{\boldsymbol{l}} \cdot \hat{\boldsymbol{n}}}{2}\right) k_{warm}$$



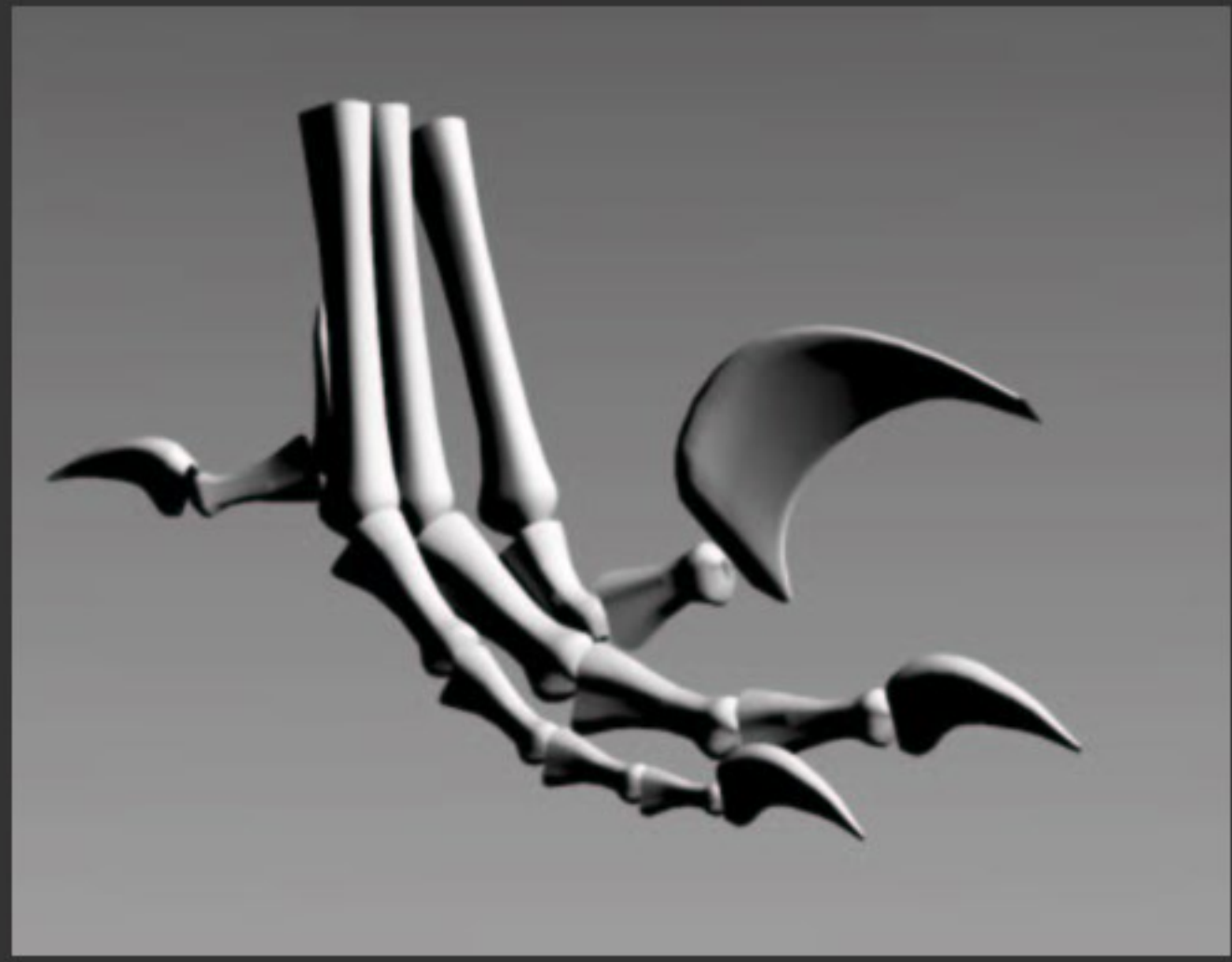
a) Phong model for colored object.

b) New shading model with highlights, cool-to-warm hue shift, and without edge lines.

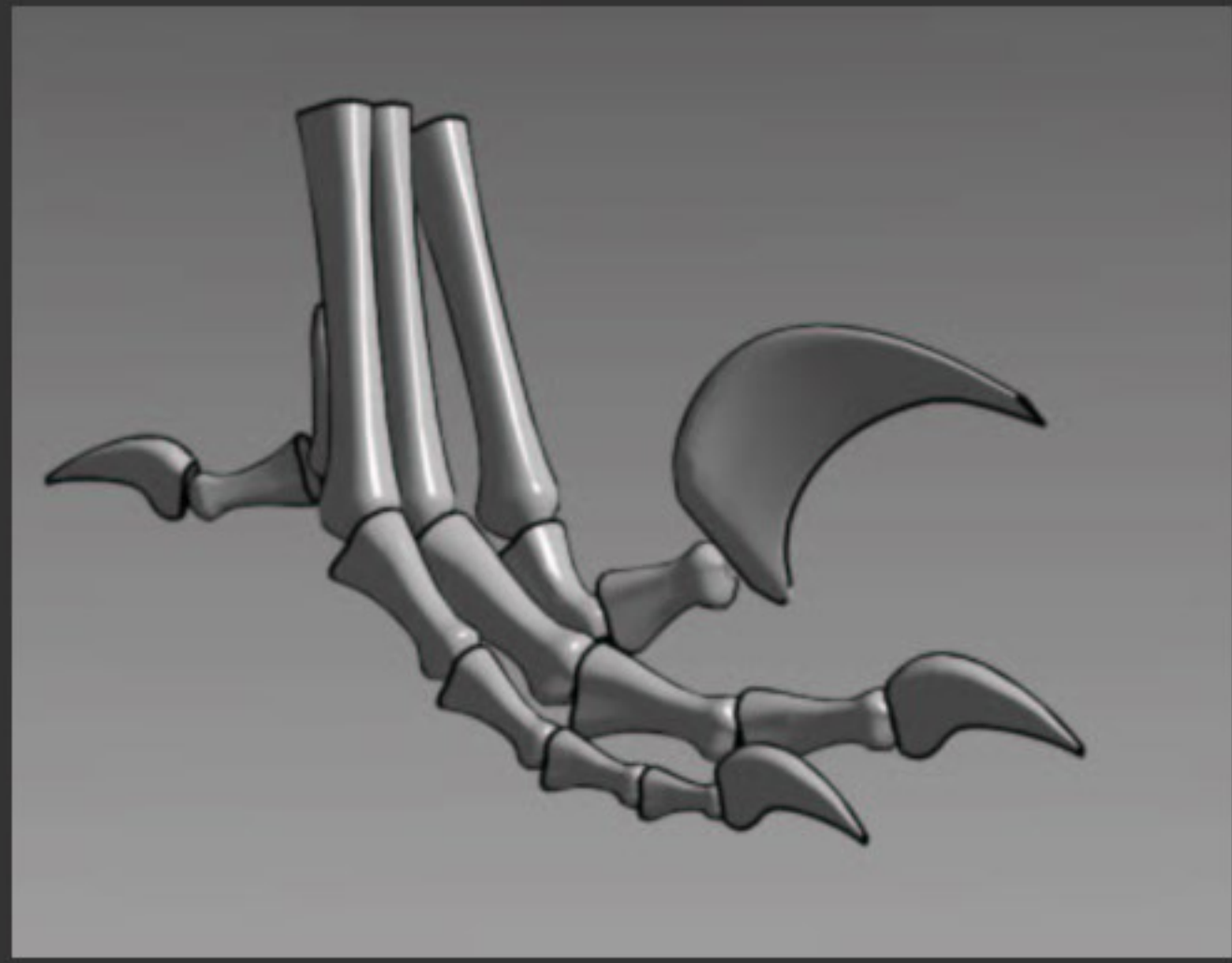
c) New model using edge lines, highlights, and cool-to-warm hue shift.

d) Approximation using conventional Phong shading, two colored lights, and edges

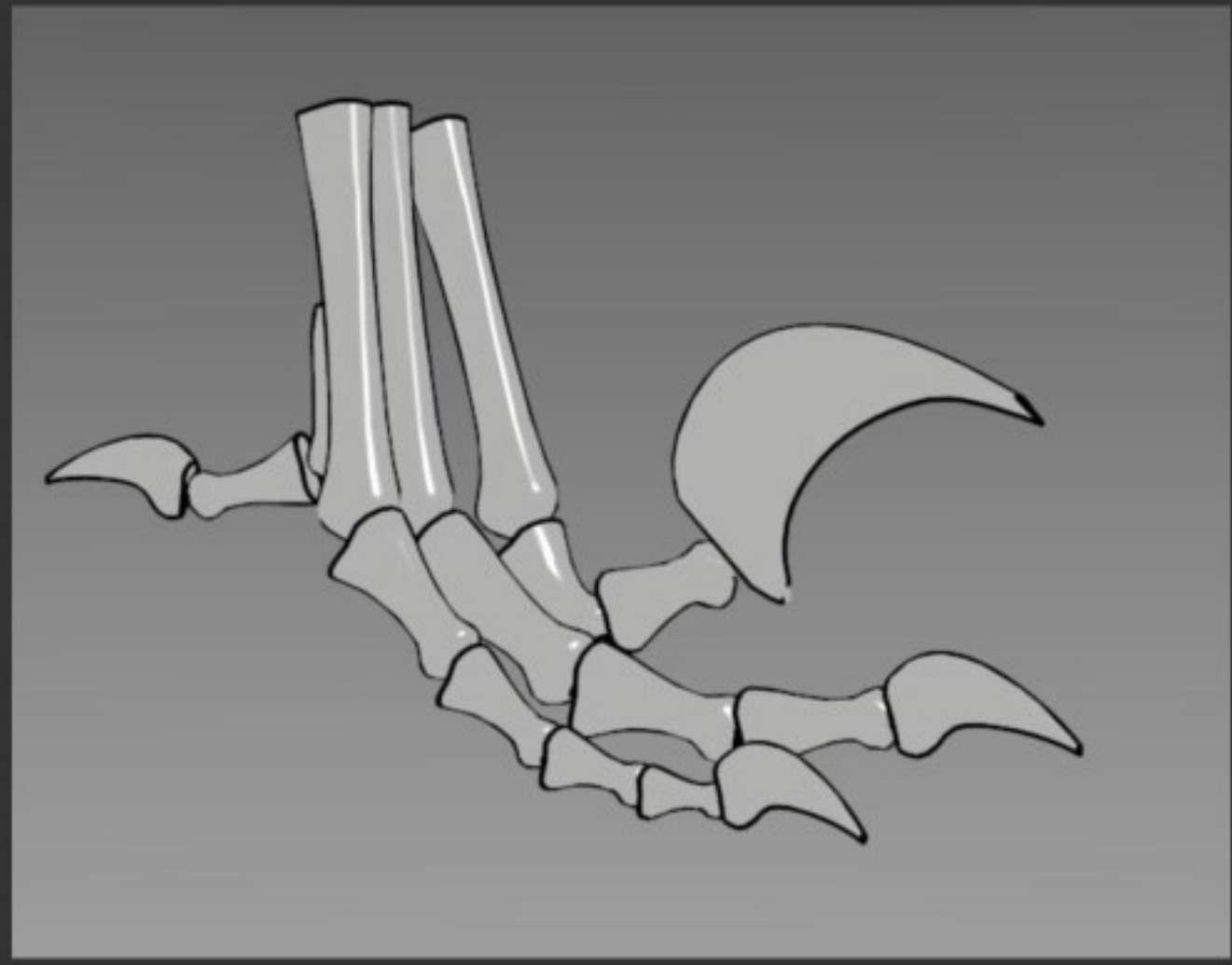
Diffuse + No Ambient



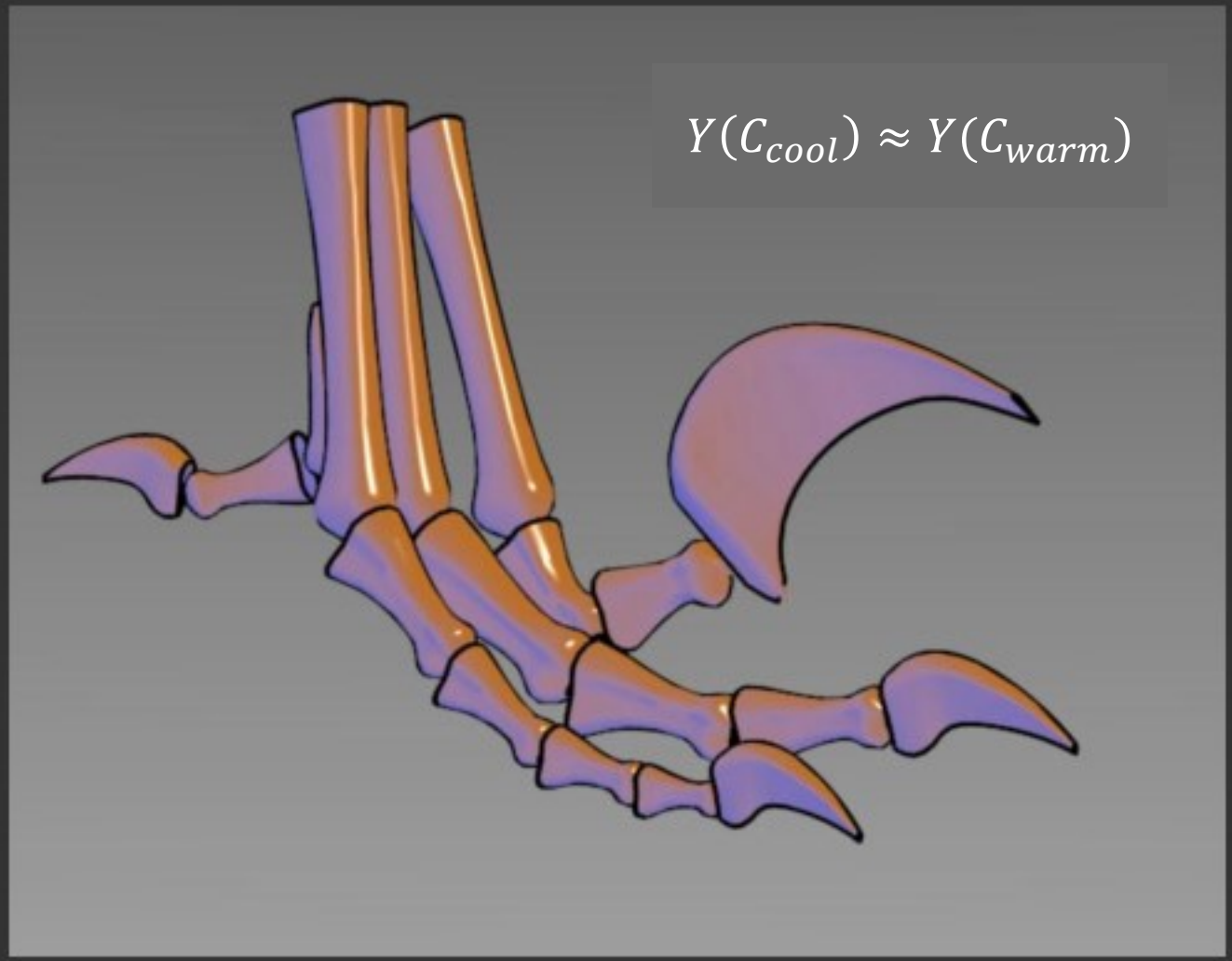
Lower Diffuse; Add Ambient



Edges + Highlights

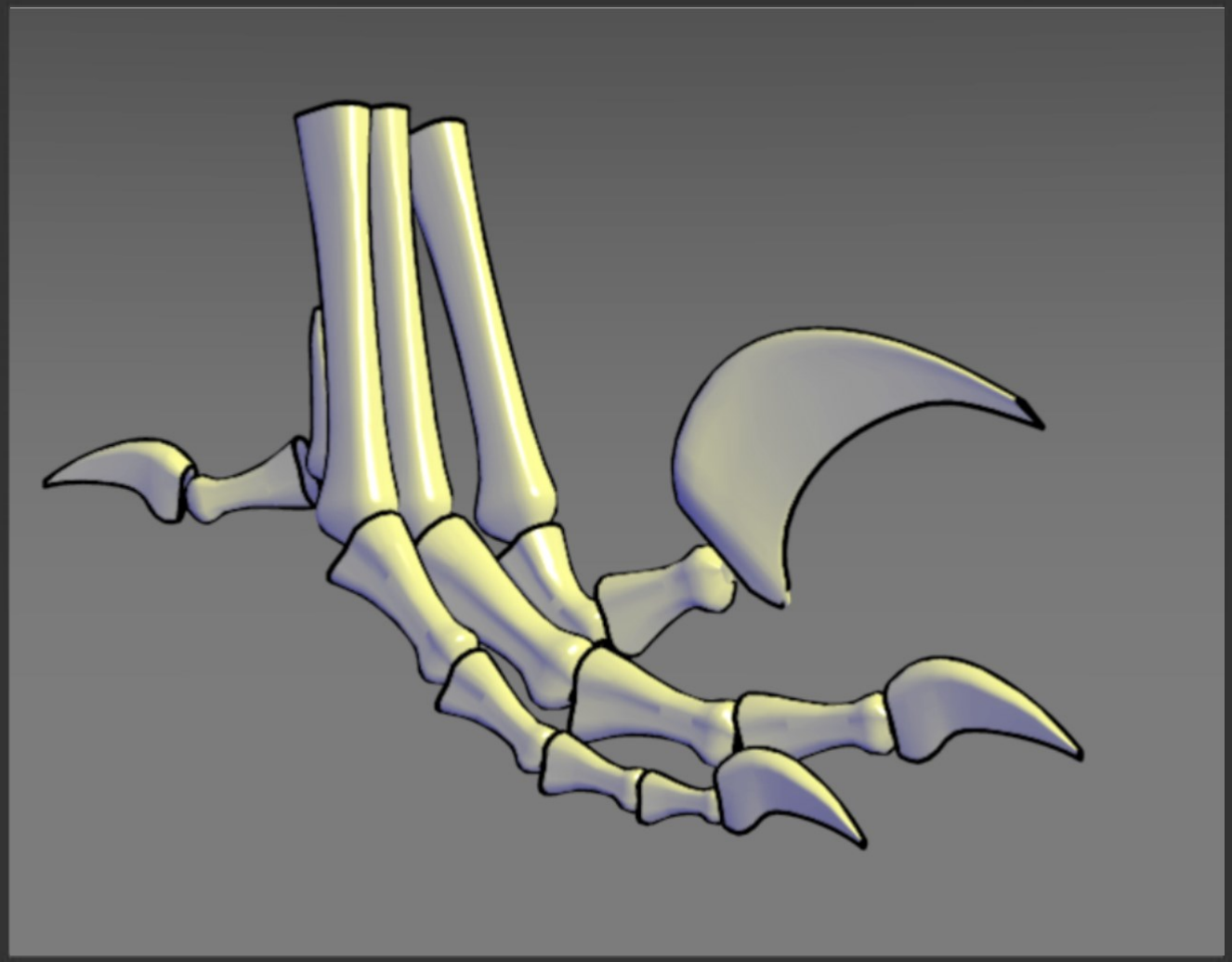


Tone-based Shading



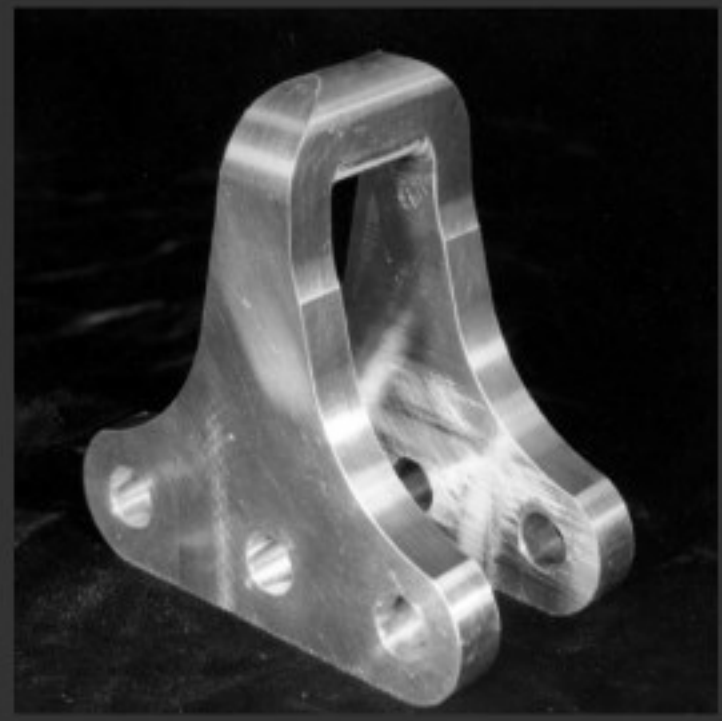
$$I = \left(\frac{1 + \hat{\boldsymbol{l}} \cdot \hat{\boldsymbol{n}}}{2}\right) k_{cool} + \left(1 - \frac{1 + \hat{\boldsymbol{l}} \cdot \hat{\boldsymbol{n}}}{2}\right) k_{warm}$$

Tone-based Shading



Blue to yellow + object color

Anisotropic Metallic Objects

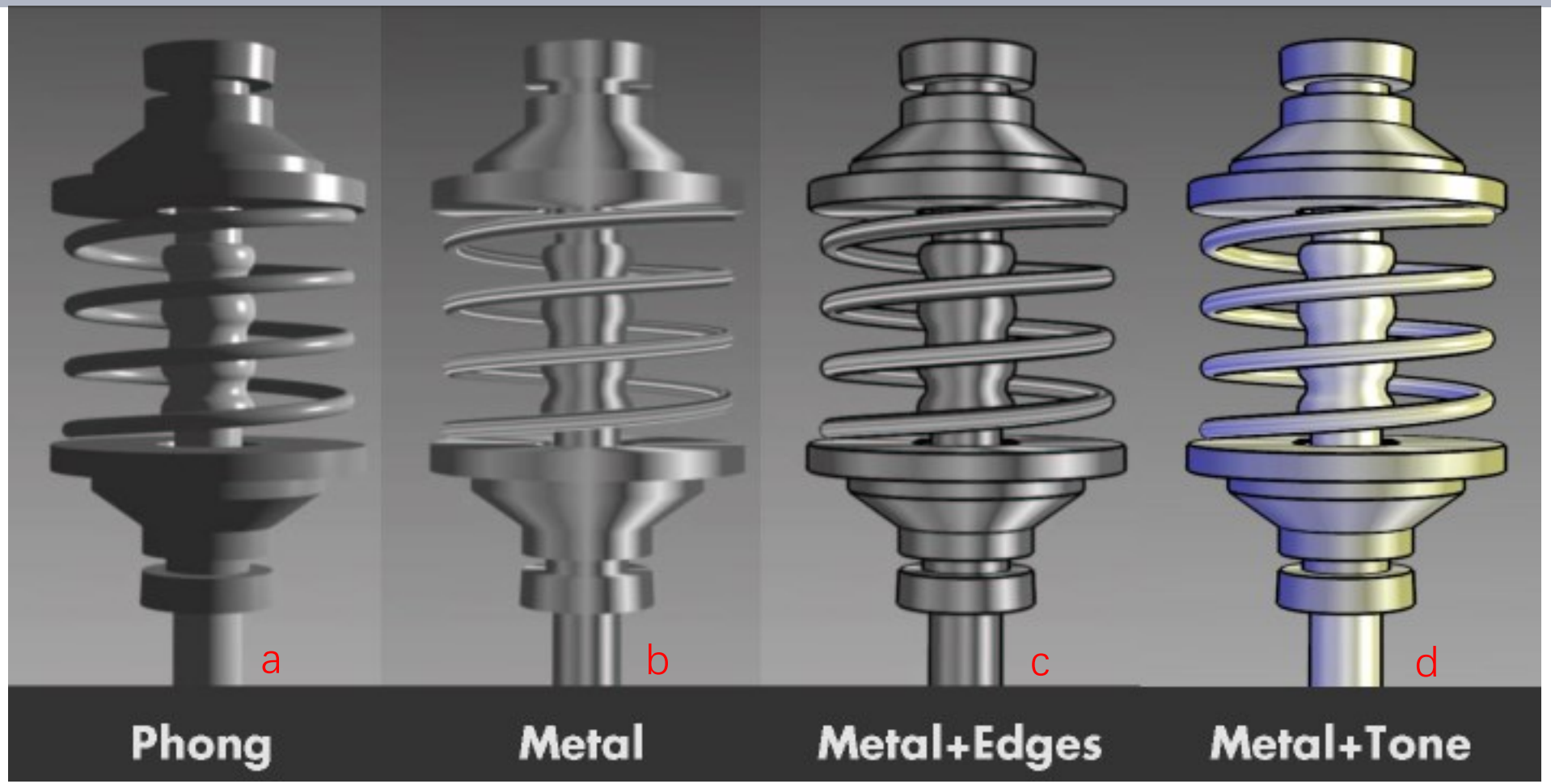


Shading of Metal Objects

In practice illustrators represent a metallic surface by alternating dark and light bands, what is known as "*anisotropic reflection*" on milled metal parts. Lines are streaked in the direction of the axis of minimum curvature,

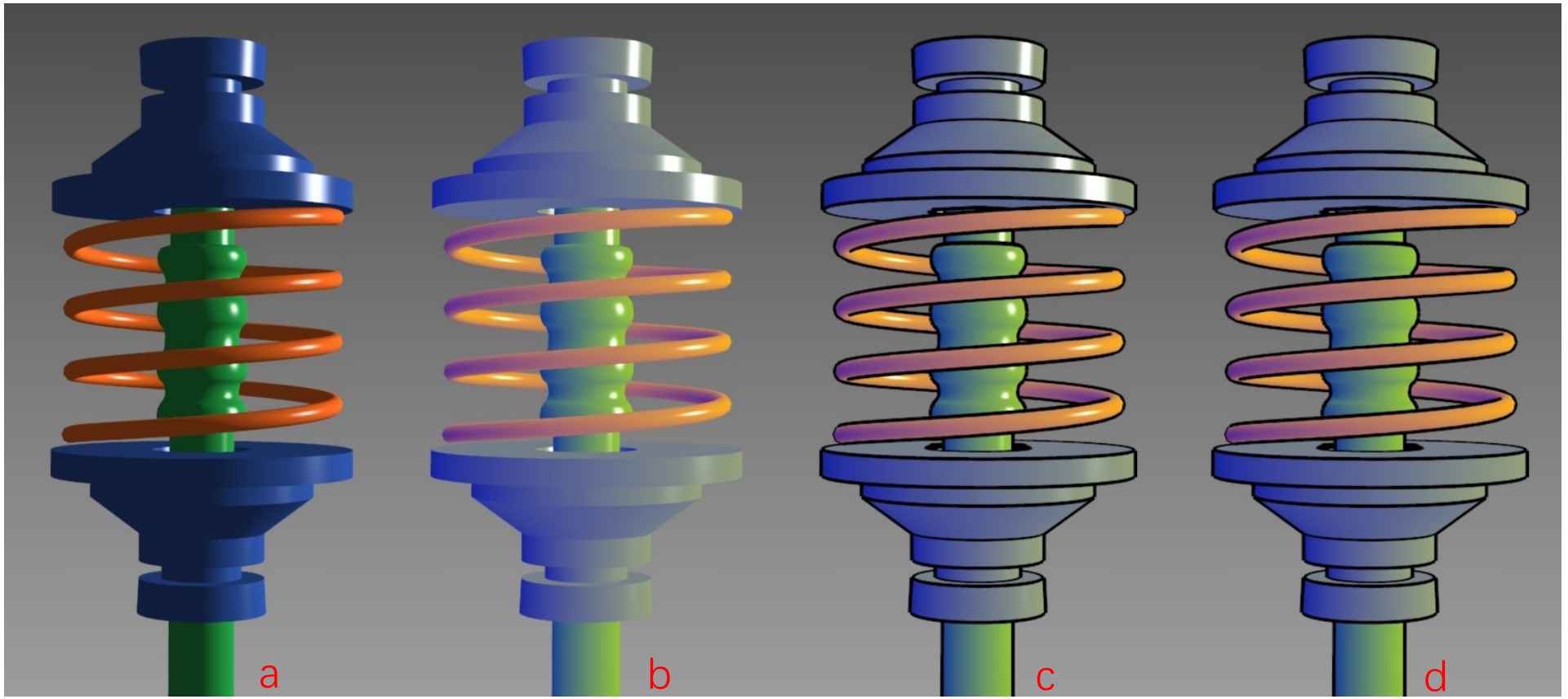
To simulate a milled object, we map a set of 20 stripes of varying intensity along the parametric axis of maximum curvature. The stripes are random intensities between 0.0 and 0.5 with the stripe closest to the light source direction overwritten with white. Between the stripe centers the colors are linearly interpolated.

Metallic Shading



a) Phong shaded object. **b**) New metal-shaded object without edge lines. **c**) New metal-shaded object with edge lines. **d**) New metal-shaded object with a cool-to-warm shift.

Metallic Shading



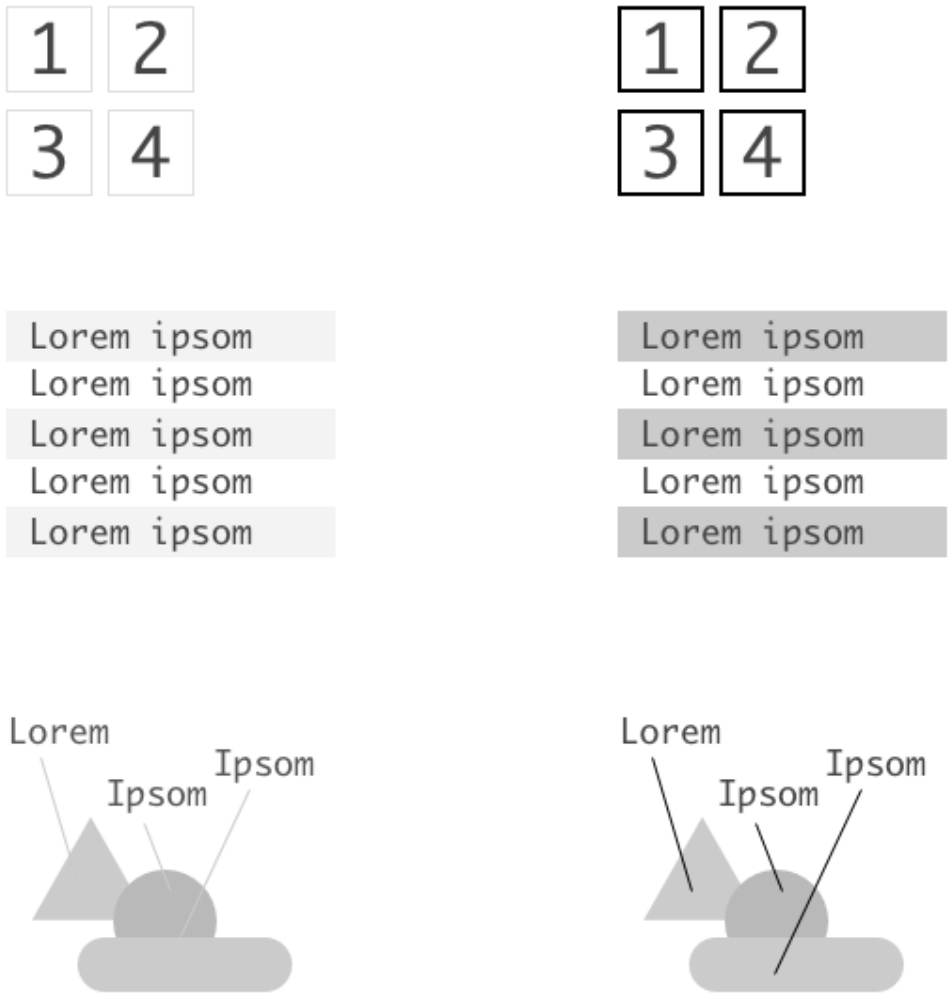
a) Phong model for colored object. **b**) New shading model with highlights, cool-to-warm hue shift, and without edge lines. **c**) New model using edge lines, highlights, and cool-to-warm hue shift. **d**) Approximation using conventional Phong shading, two colored lights, and edge lines.

Learning from Technical Illustration

Characteristics in technical illustrations:

- surface boundaries, silhouettes, and discontinuities: lines drawn with black curves
- matte objects: shaded with intensities far from black or white with warmth or coolness of color indicative of surface normal; a single light source provides white highlights
- metal objects: shaded as if very anisotropic

Learning from Technical Illustration



Left: Do this — Right: Don't do this

Edward Tufte. *Visual Explanations*. Graphics Press, 1997:

Tufte advocates the strategy of *the smallest effective difference* :

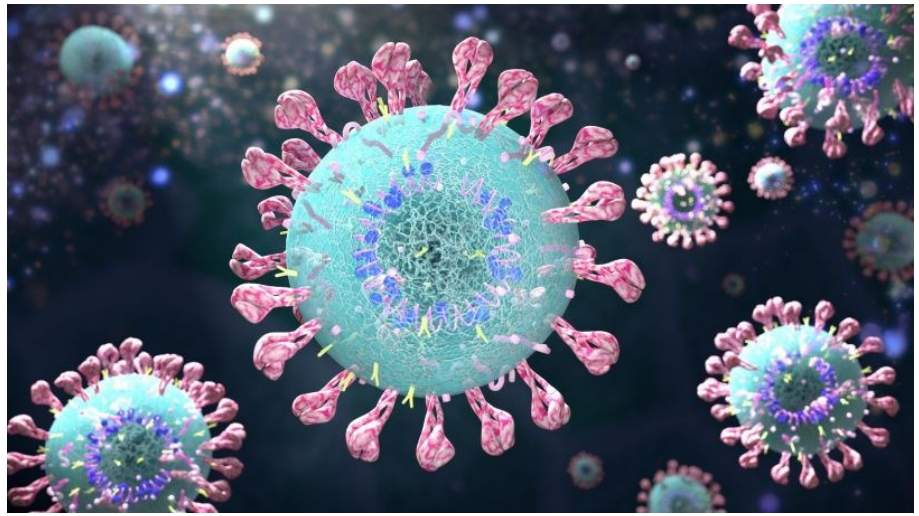
Make all visual distinctions as subtle as possible, but still clear and effective.

The principle provides a possible explanation of why cross-hatching is common in black and white drawings and rare in colored drawings: a more subtle, but adequately effective difference to communicate surface orientation.

Biological and Medical Illustration

© Ella Marushchenko

Asthma © Todd Buck



Anatomic Groove