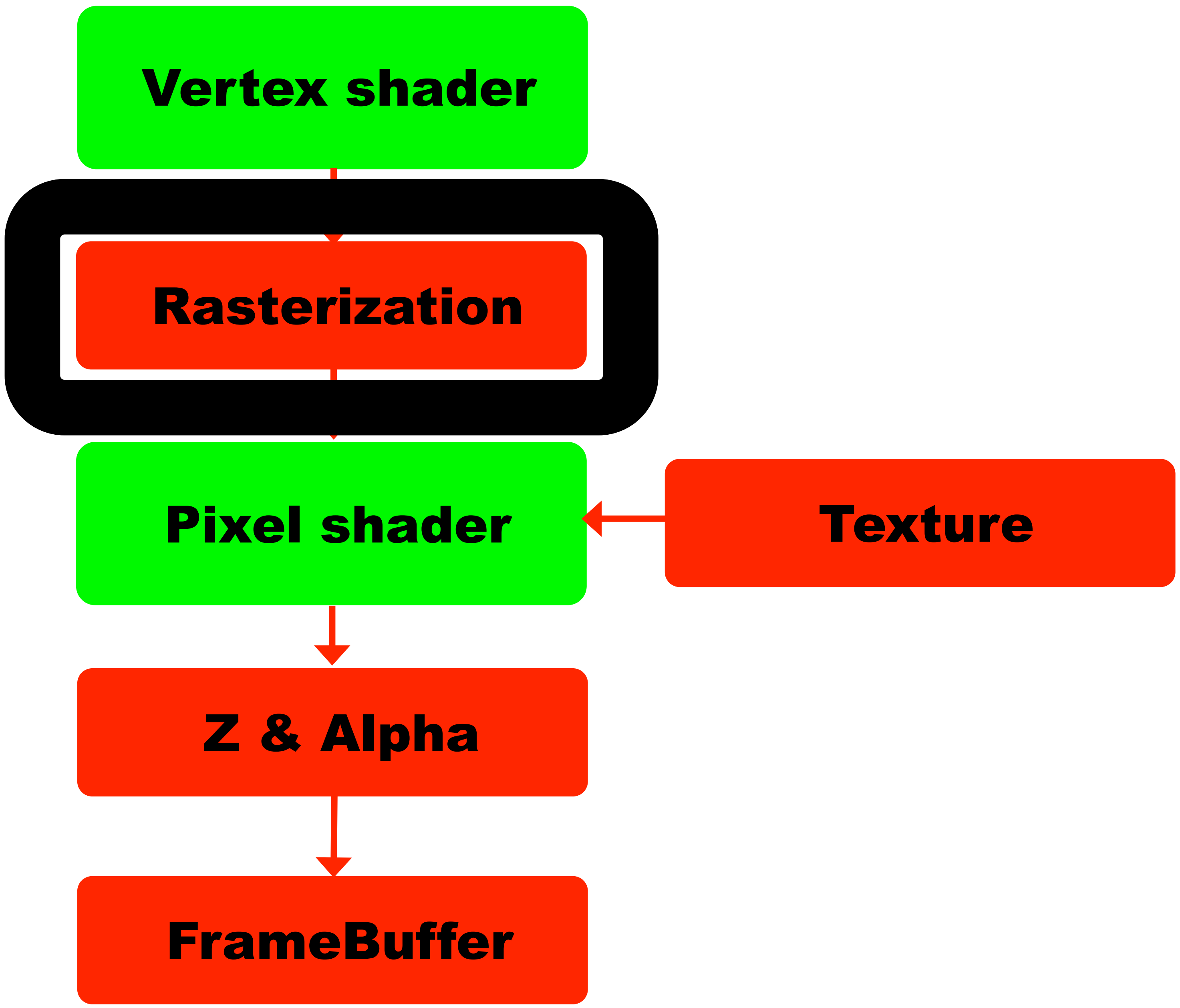
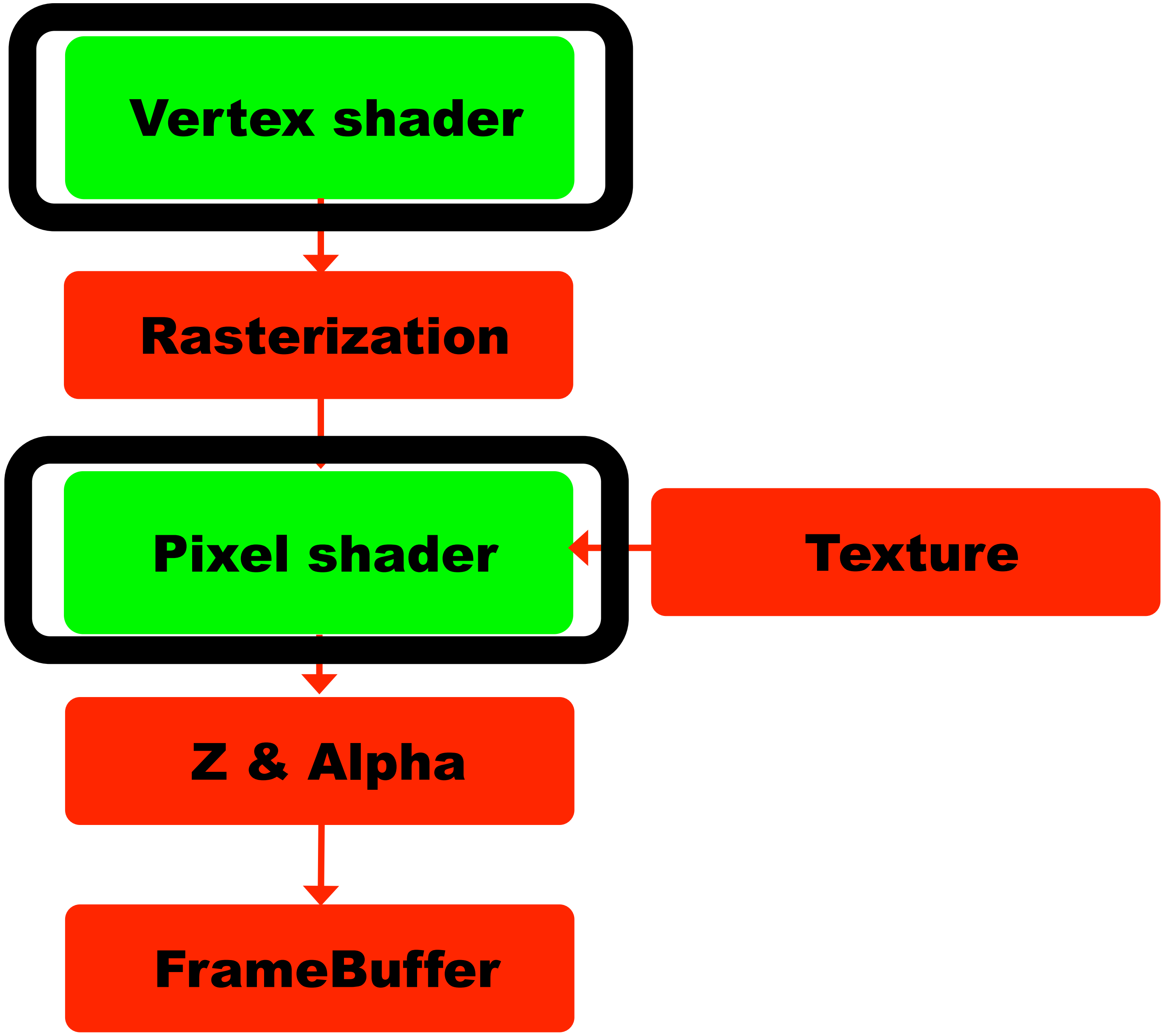
Shading

Michael Doggett
Department of Computer Science
Lund university

Stages we have looked at so far



Today's stages of the Graphics Pipeline



Overview

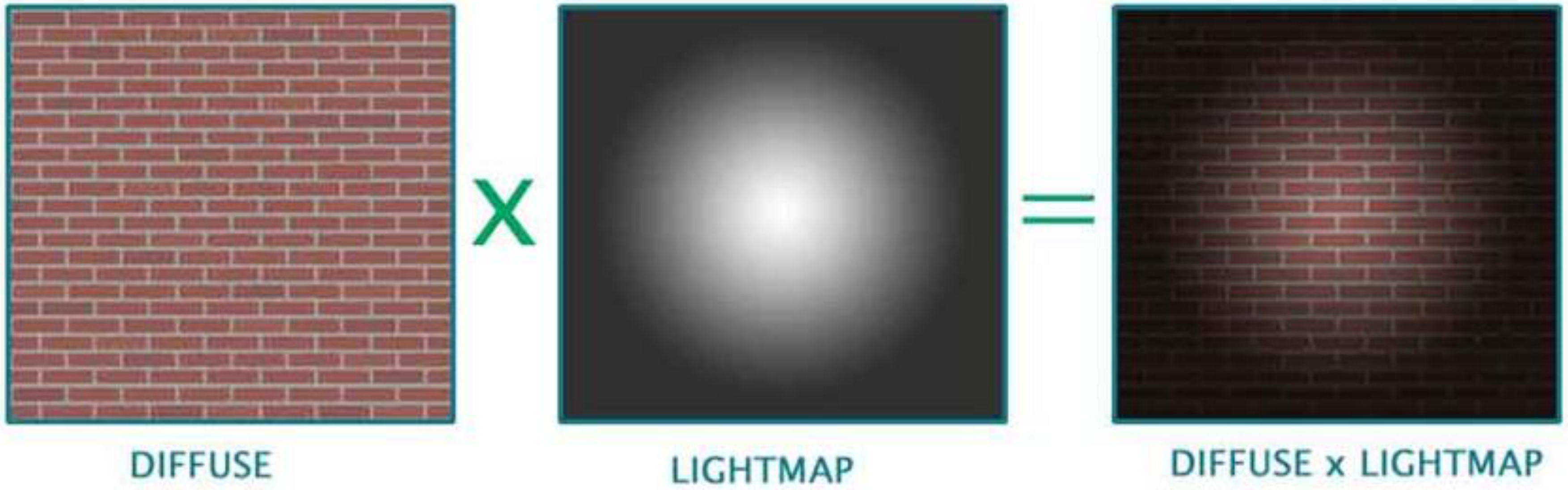
- What effects can we create with programmable shaders?
    - Shader trees
    - Physically Based Shading
    - Glass
    - Skin
    - Ambient Occlusion
    - Surface details
    - Cartoons
    - Non-photorealistic rendering
    - Glow, Fur

Resources

- GPU GEMS 1, 2, 3
  - All freely available on nvidia's web page
  - https://developer.nvidia.com/gpugems/GPUGems/gpugems_pref01.html
- Shader X
  - Book series similar to GPU GEMS
  - Latest version called GPU Pro
- Real-Time Rendering
  - Text book with detailed description of all aspects of real-time effects
- Search the web for lots of example code, blog posts and game development pages
- WebGL based shaders
  - www.shadertoy.com

Light Maps

- Very old technique
  - First used in Quake
- Static



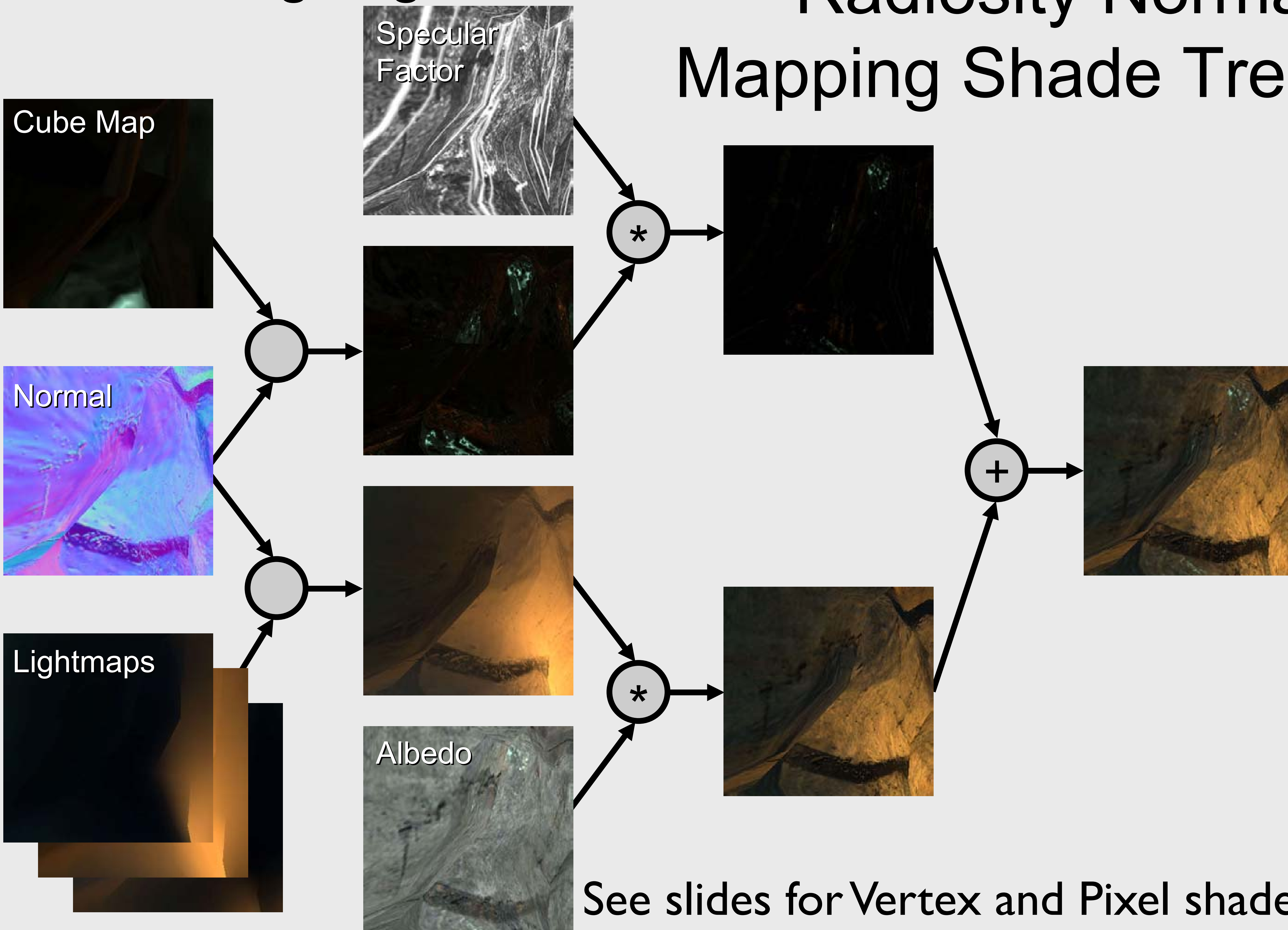
Images from https://www.flipcode.com/archives/Light_Mapping_Theory_and_Implementation.shtml

Half-Life 2 Lighting

Gary McTaggart, Valve, “Half-Life2 Shading/Valve Source Shading” GDC 2004

Half-Life 2 Lighting

Radiosity Normal Mapping Shade Tree



See slides for Vertex and Pixel shaders

Gary McTaggart, Valve, "Half-Life2 Shading/Valve Source Shading" GDC 2004

Physically-**B**ased **S**hading

Image courtesy Michal Iwanicki and Angelo Pesce, "Approximate models for physically based rendering", PBS course 2015

Physically-**B**ased **S**hading

- Material shaders had become very complex
- Better to have consistent materials
  - Something that just works
- PBS uses energy conservation
- Creates a framework to understand and reason about materials
- Used in both Real-Time and Offline

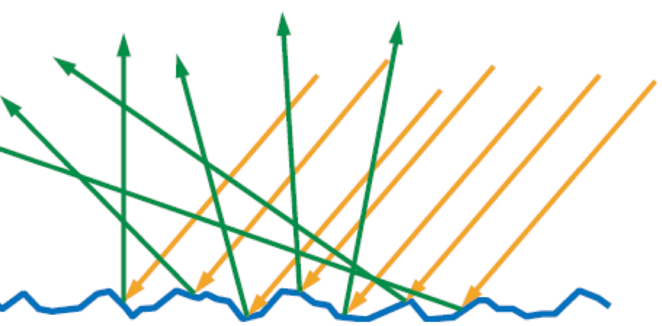
The Rendering Equation

$$L_o = L_e + \int_\Omega L_i \cdot f_r \cdot \cos\theta \cdot d\omega$$

- f_r is the Bidirectional Reflectance Distribution Function (BRDF)

Bidirectional **R**eflectance **D**istribution **F**unction

$$f(\mathbf{l},\mathbf{v}) = \frac{F(\mathbf{l},\mathbf{h})G(\mathbf{l},\mathbf{v},\mathbf{h})D(\mathbf{h})}{4(\mathbf{n}\cdot\mathbf{l})(\mathbf{n}\cdot\mathbf{v})}$$



- Microfacet surface model
  - Microgeometry changes how light is reflected (and refracted)
- Rougher surfaces create blurrier reflections
- **F**(l,h) is the Fresnel term
  - More about this later

Bidirectional **R**eflectance **D**istribution **F**unction

$$f(\mathbf{l},\mathbf{v}) = \frac{F(\mathbf{l},\mathbf{h})G(\mathbf{l},\mathbf{v},\mathbf{h})D(\mathbf{h})}{4(\mathbf{n}\cdot\mathbf{l})(\mathbf{n}\cdot\mathbf{v})}$$

- G() - **G**eometry Function
  - Chance that a micro facet is shadowed and/or masked
    - Several options in the literature
- D() - Normal **D**istribution Function
  - Distribution of normals around a given direction (halfway vector)
  - Size and shape of the spectral highlight
  - Many possible equations

Disney BRDF

Fig21. Wreck-It Ralph 2012.
Image courtesy Brent Burley, "Physically-Based Shading at Disney", 2012

Disney BRDF

- The BRDF is defined by a base color, and 10 scalar parameters:
  - Subsurface, Metallic, Specular, Specular tint, Roughness, Anisotropic, Sheen, Sheen tint, Clearcoat, Clearcoat gloss

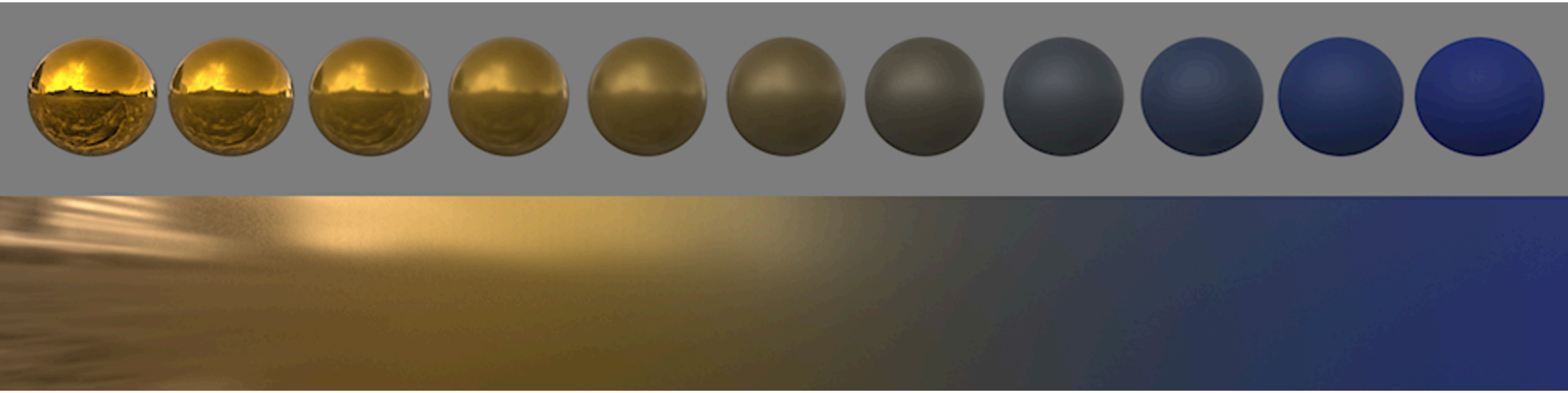


Fig.19 From shiny metallic gold to blue rubber
Image courtesy Brent Burley, "Physically-Based Shading at Disney", 2012

Disney BRDF

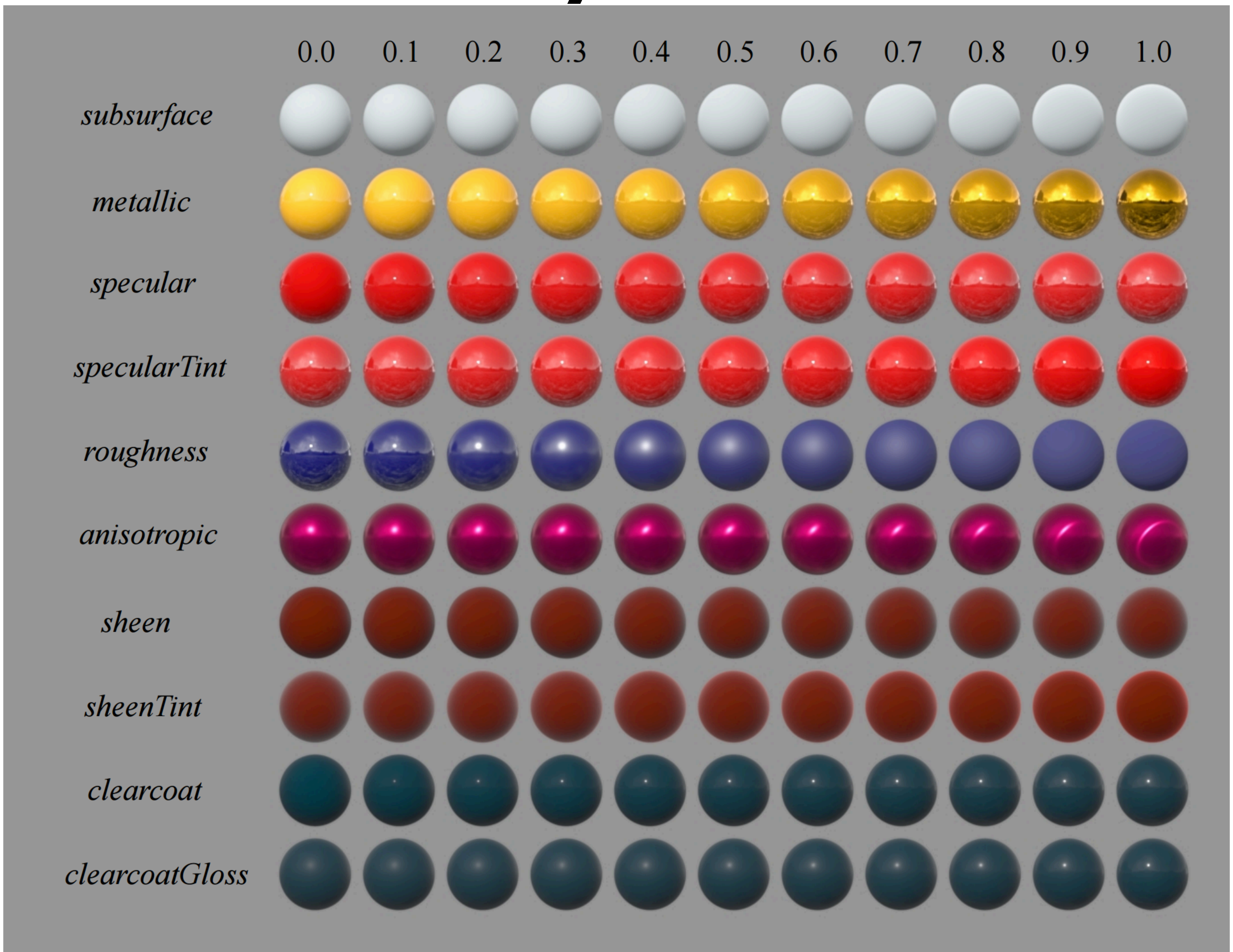
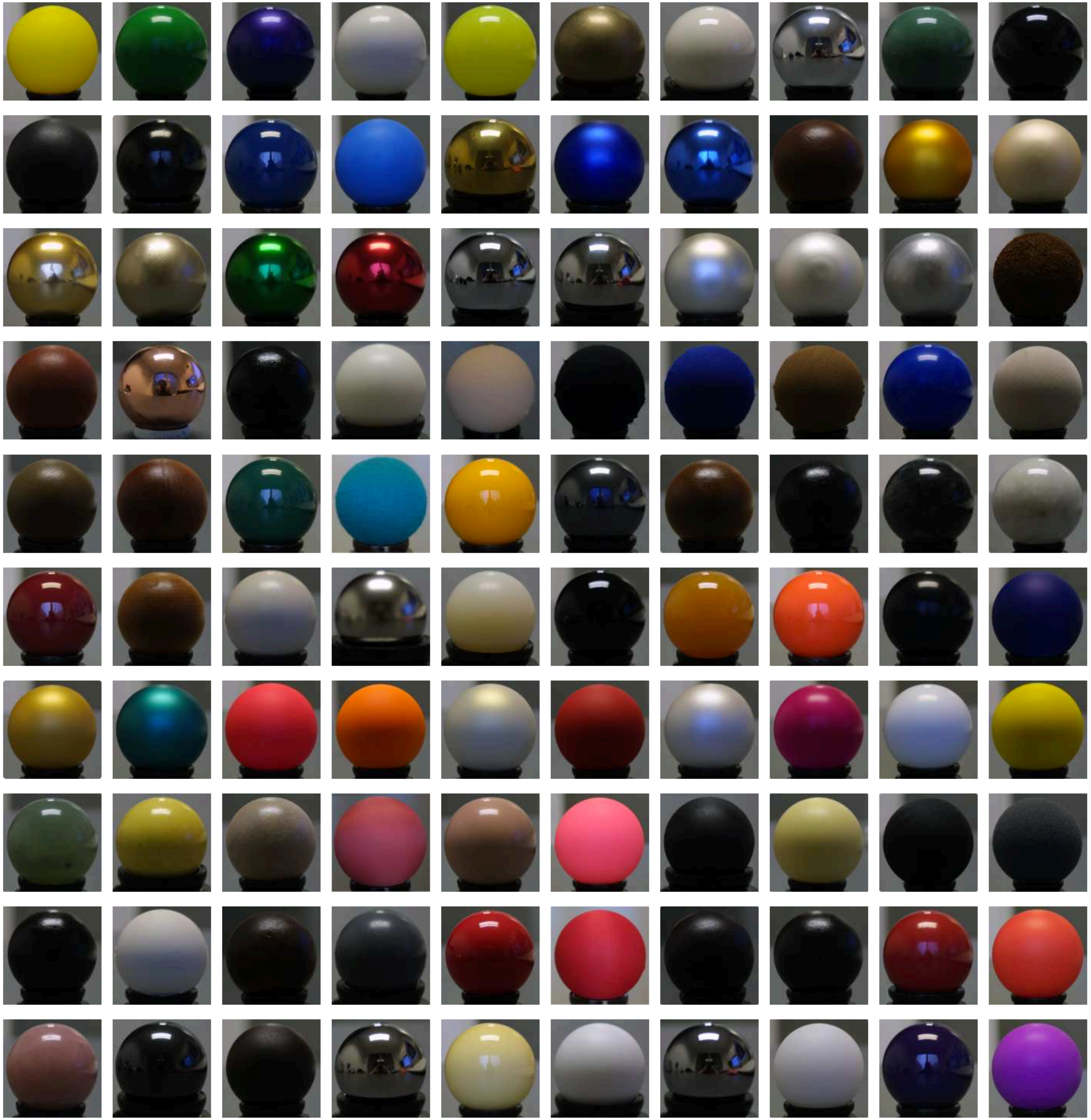


Fig. 16 : Varying each parameter. Image courtesy Brent Burley, “Physically-Based Shading at Disney”, 2012

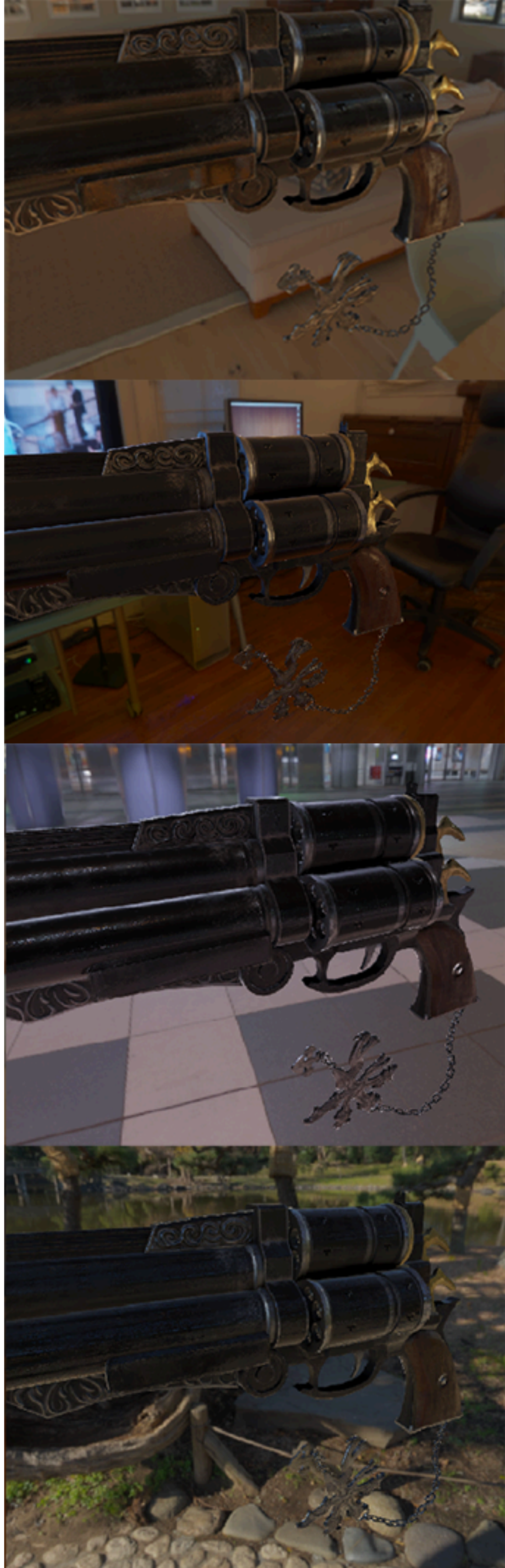
MERL BRDF database of measured materials



SGD implementation of all MERL materials at shadertoy : https://www.shadertoy.com/view/XssGzf

Real-Time PBS

- Cook-Torrance BRDF has a diffuse and specular part
  - Direct lighting (Point lights) are a single direction over hemisphere
  - Image Based Lighting (Environment/Cube maps)
    - Diffuse approximated with pre-filtered Irradiance Cube Map
    - Specular can use Epic’s Split Sum approximation
      - Indirect specular reflections using pre-filtered Cube Map with mip levels for different roughness
      - Pre-computed BRDF in 2D texture LUT using N·V and roughness
- Full details : https://learnopengl.com/PBR/Theory



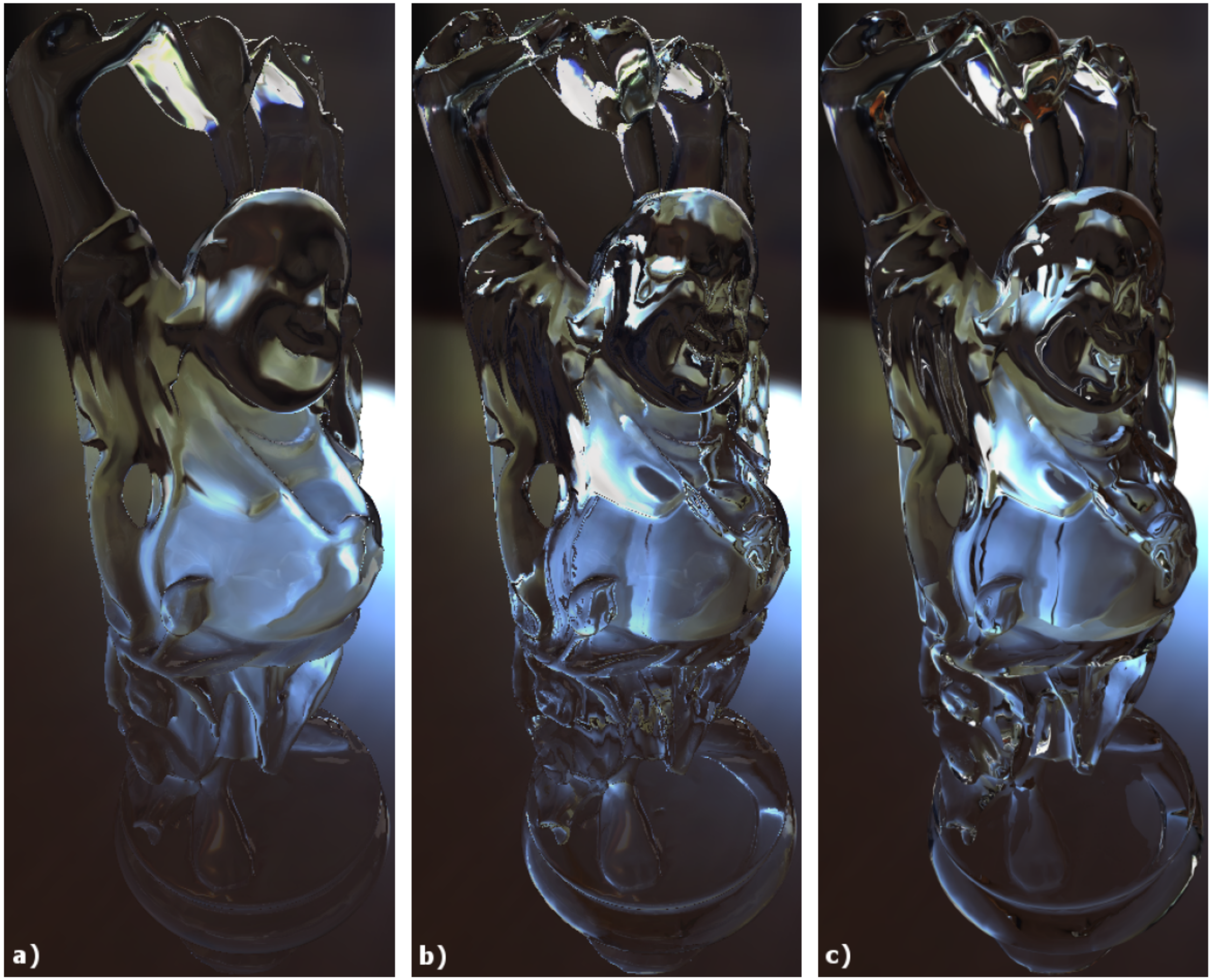
PBS more info

- https://learnopengl.com/PBR/Theory
- Physically Based Shading at ShaderToy https://www.shadertoy.com/view/4sSfzK
- "Physics and Math of Shading", Naty Hoffman
  - youtube : https://youtu.be/j-A0mwsJRmk
  - pdf : http://blog.selfshadow.com/publications/s2015-shading-course/hoffman/s2015_pbs_physics_math_slides.pdf
- PBS course at SIGGRAPH
  - 2020: http://blog.selfshadow.com/publications/s2020-shading-course/ and '17, '16, '15, '14, '13, '12, '10, lots of material

Refraction effects

- Many different techniques
- Increases level of realism a bit (if done well)
- Hacky technique:
  - –Refract only ray at first intersection
  - –This is incorrect...
  - – but simple!

Refraction comparison with better techniques



Refraction using one interface

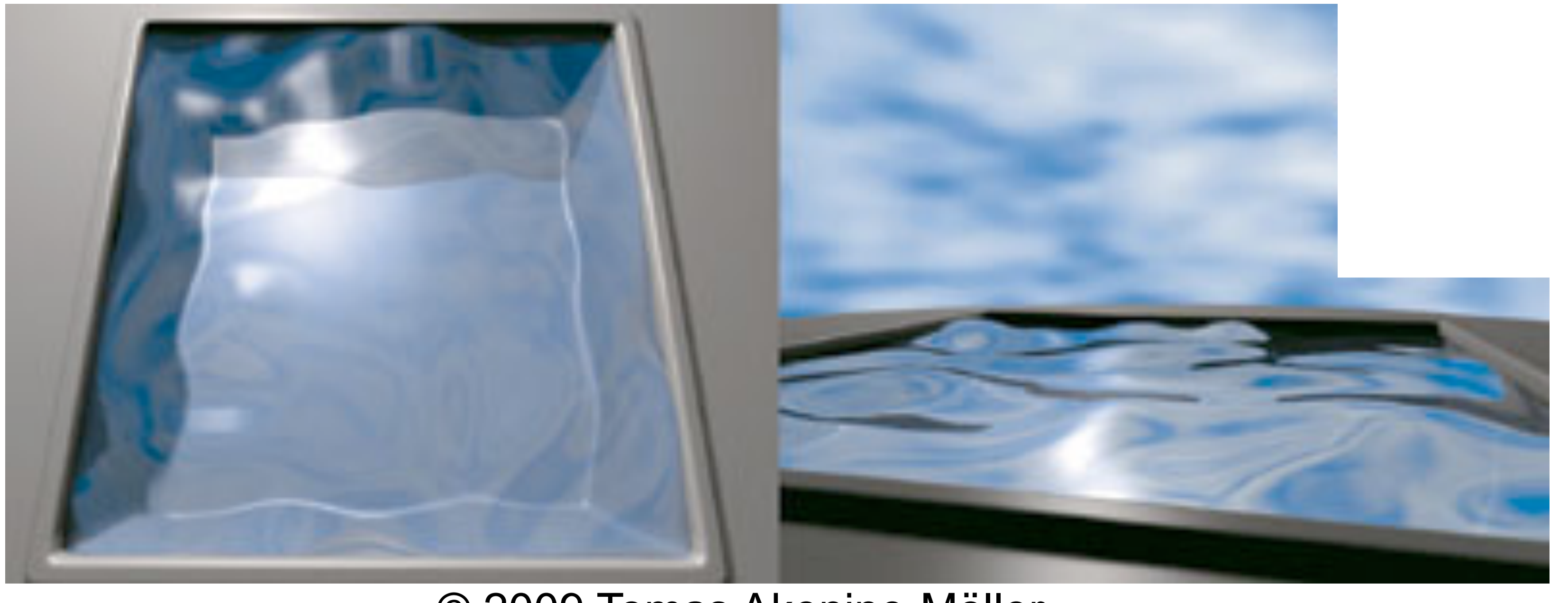
Refraction using Wyman's technique (not part of the course)

Ray tracing

Images courtesy of Chris Wyman

Refraction (cont'd)

- Back to simplest technique:
  - Refract at first intersection
- Add **Fresnel** reflection for improved realism
  - Reflection term is bigger at grazing angles
  - Perpendicular to the surface you see through
  - Parallel to the surface you see reflection
  - Especially so for dielectrics (transparent): glass, plastics, water



Fresnel (cont'd)

– Depends on: Coefficient of extinction, Incident angle, Index of refraction

$$F = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left(1 + \frac{[c(g+c)-1]^2}{[c(g-c)+1]^2}\right)$$

$$g = \sqrt{n^2 + c^2 - 1}$$

$$c = \mathbf{v} \cdot \mathbf{h}$$

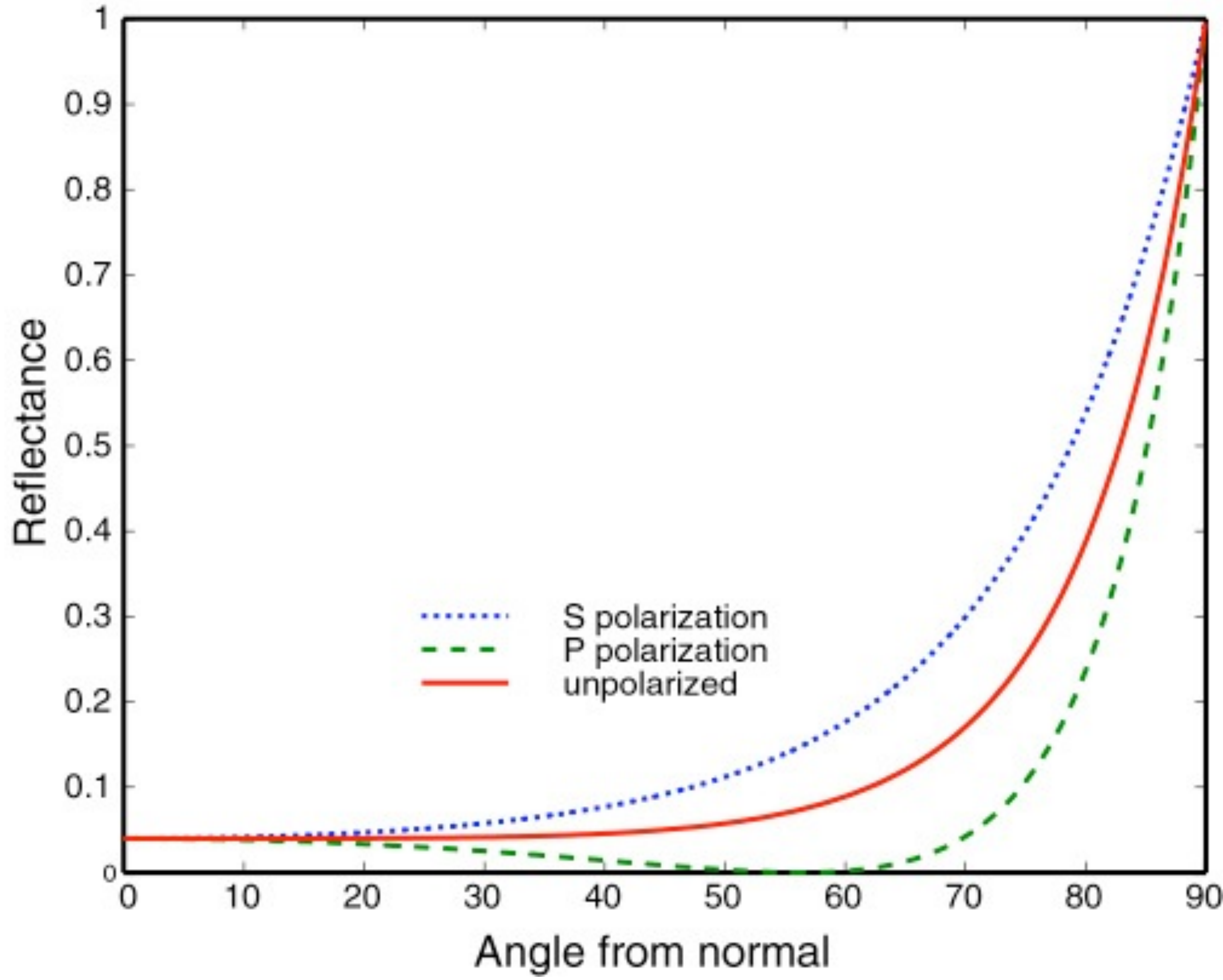
v is the view vector
h is the halfway vector
n is the index of refraction

Fresnel (cont'd)

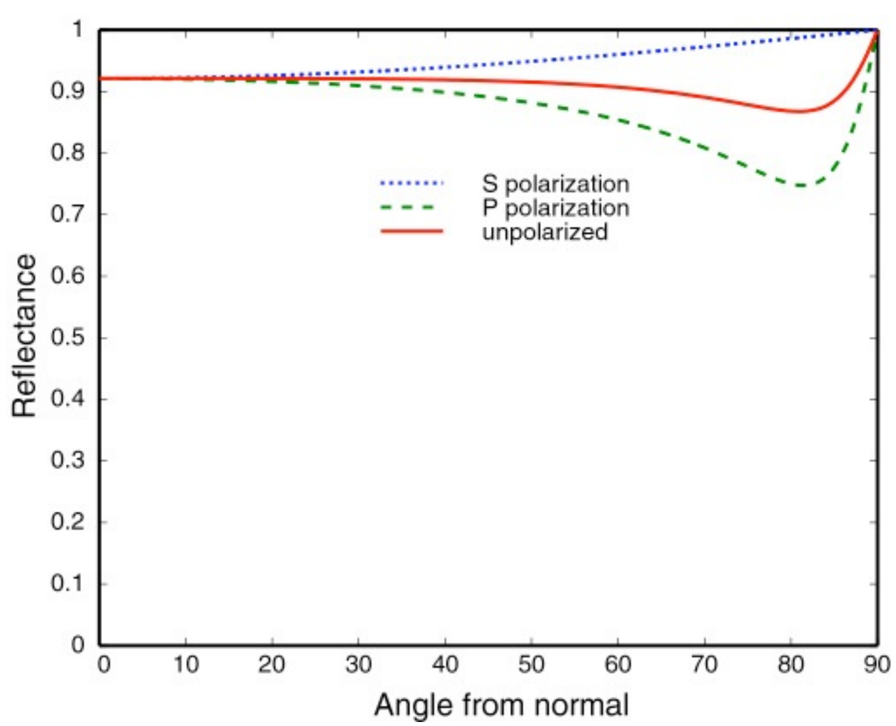
- F describes the reflectance at a surface at various angles

Images courtesy of Steve Westin

Dielectrics



Metals

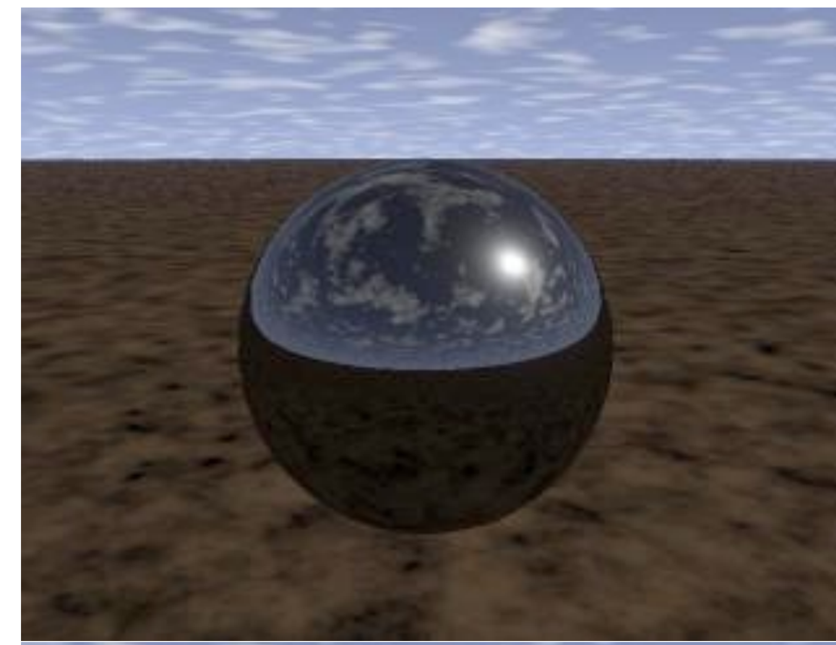


Fast Fresnel Approximation

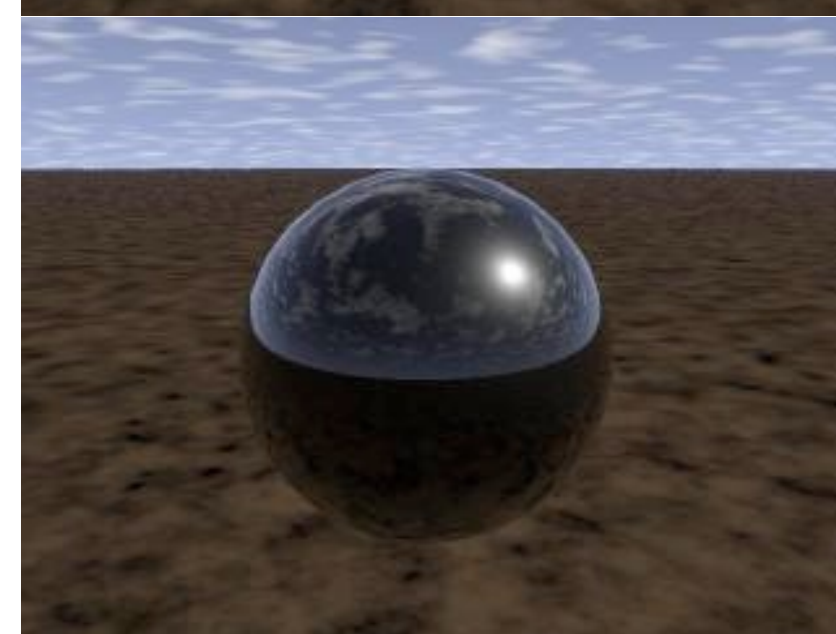
$$F = R_0 + (1 - R_0)(1 - \mathbf{v} \cdot \mathbf{n})^5$$

- **Sometimes called "Schlick" approximation**
- **v is the view vector**
- **n is the normal**
- **R_0 is reflectance when v.n=0**
- **Multiple reflected value by F**
    - **add to refracted value**

Metal



Glass



Images courtesy Steve Westin

Skin rendering:
subsurface scattering hacks

- We cheat to get real-time performance
  - Though, more sophisticated real-time algorithms exist
- Ideally: subsurface scattering
  - Photons enter material, bounces around ***inside*** material, and then exits at another point
- Hacks:
  - **Wrapping + color shifting**
  - **Depth maps**
  - Texture space diffusion

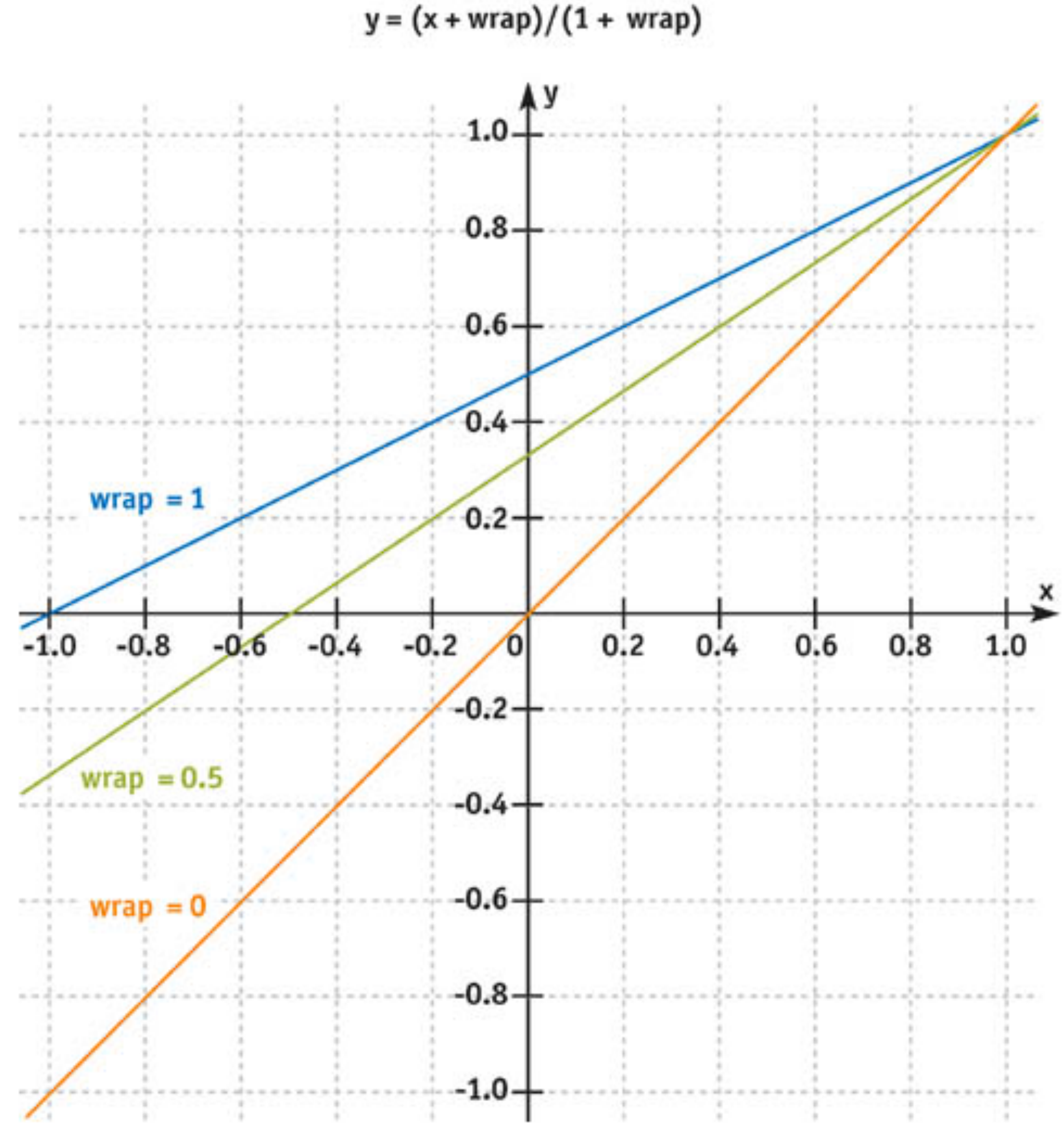
Rendered using NVIDIA's Gelato

Image courtesy of Henrik Wann Jensen

Approximating Skin

- Wrap lighting
  - Lighting wraps around the object beyond where it would normally go dark

$$f(n \cdot l) = max\left(\frac{n \cdot l + w}{1 + w}, 0\right)$$



"Real-time Approximations To Subsurface Scattering", Simon Green, GPU Gems, 2004

Approximating Skin

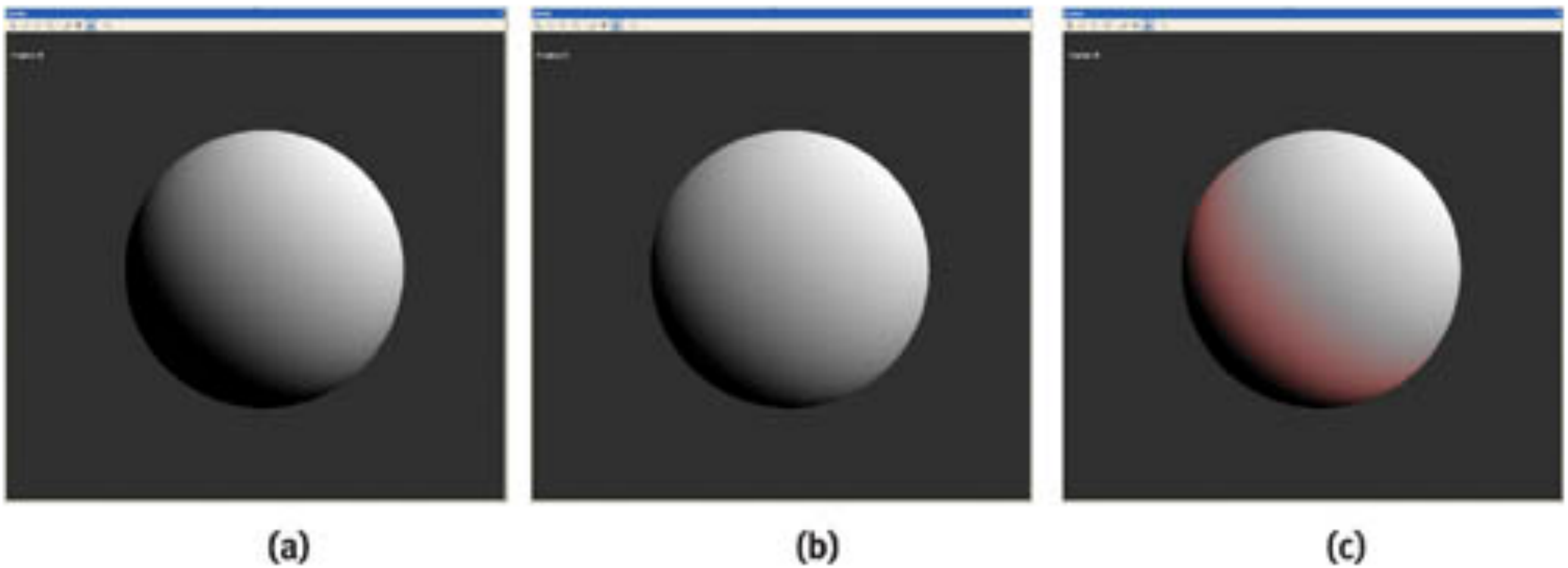
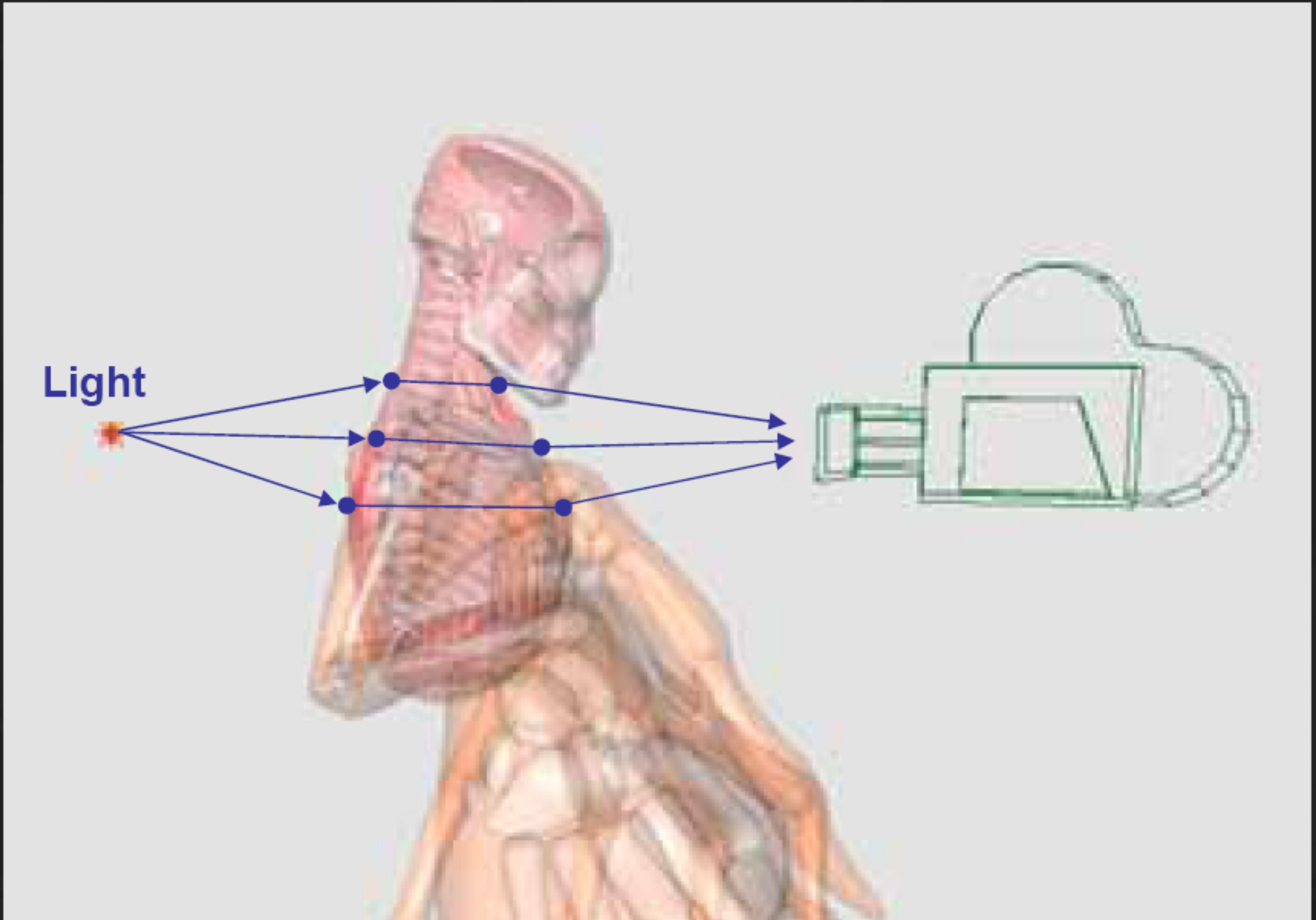


Figure 16-2 Applying Wrap Lighting to Spheres

- Wrap lighting
  - Blend in red as the lighting approaches zero

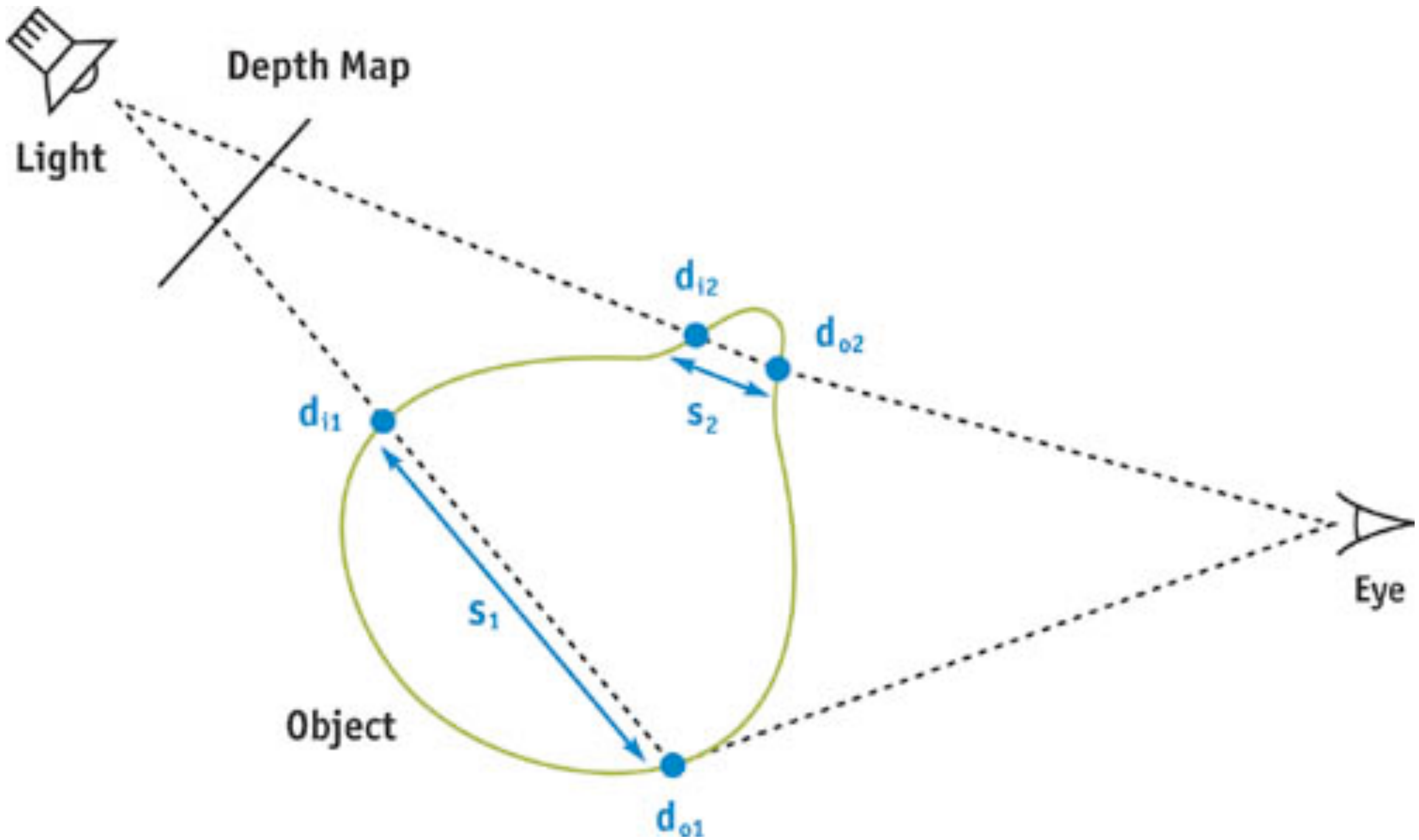
”Real-time Approximations To Subsurface Scattering”,
Simon Green, GPU Gems, 2004

Try to approximate amount of light going through the surface



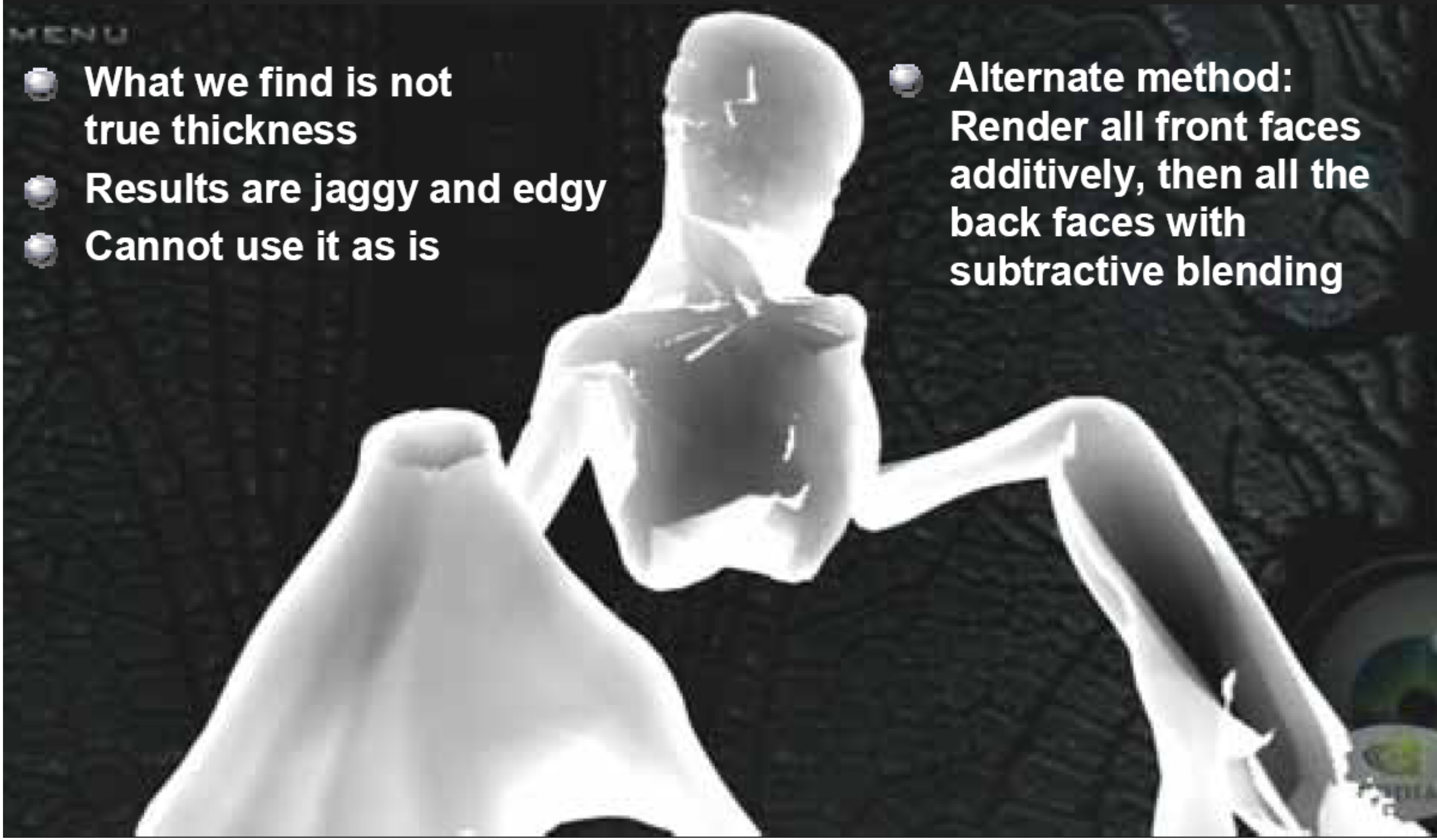
Subsurface scattering using Depth Maps

- Compute object thickness from camera's POV
- Use thickness to look up a color for the skin



- Render back faces into FP16 buffer
- Render front face, fetch back face depths from buffer
- Compute distance, scale to [0,1]
- Look up color

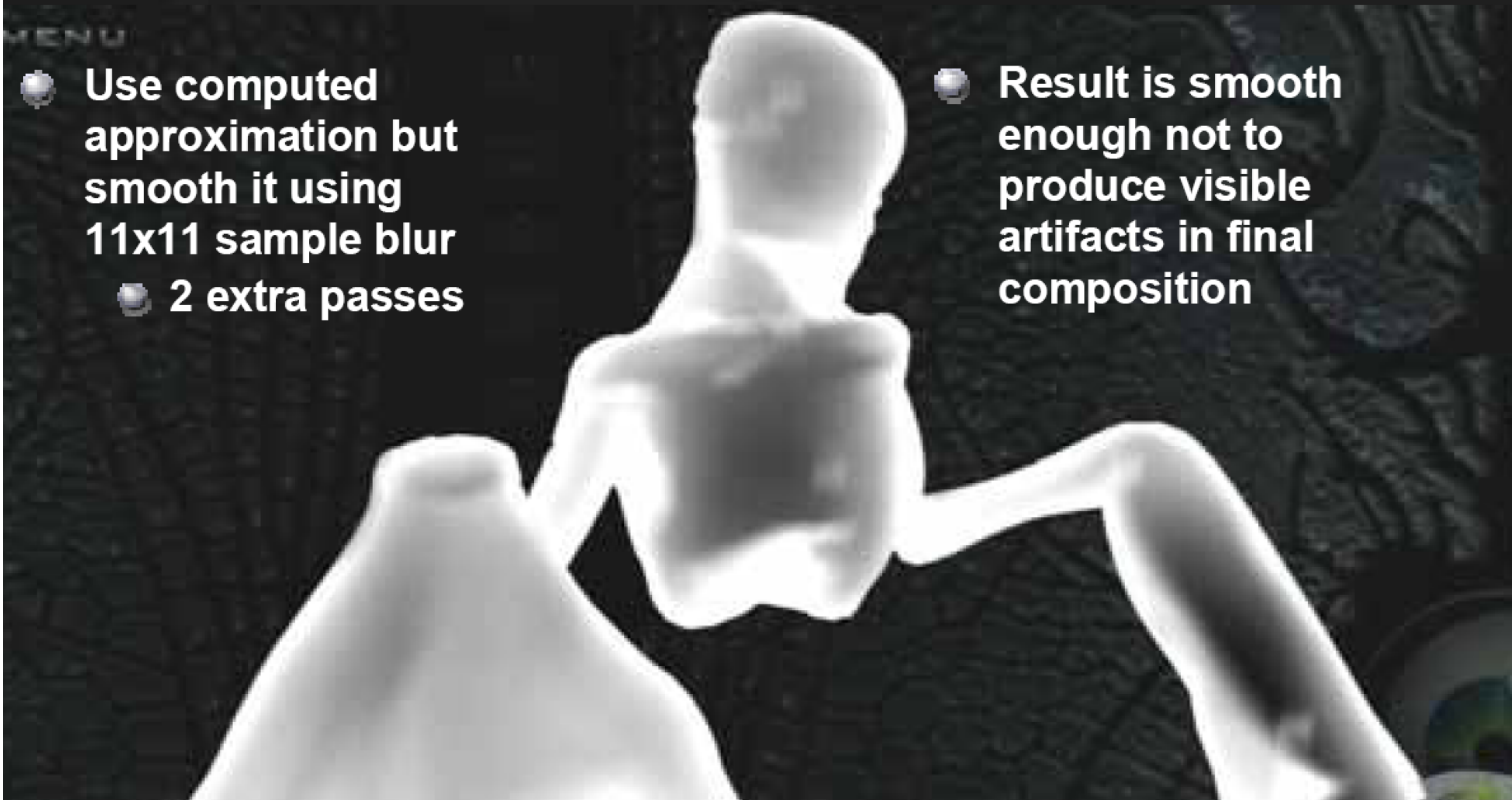
Computing Thickness



- What we find is not true thickness
- Results are jaggy and edgy
- Cannot use it as is

- Alternate method: Render all front faces additively, then all the back faces with subtractive blending

Computing Thickness

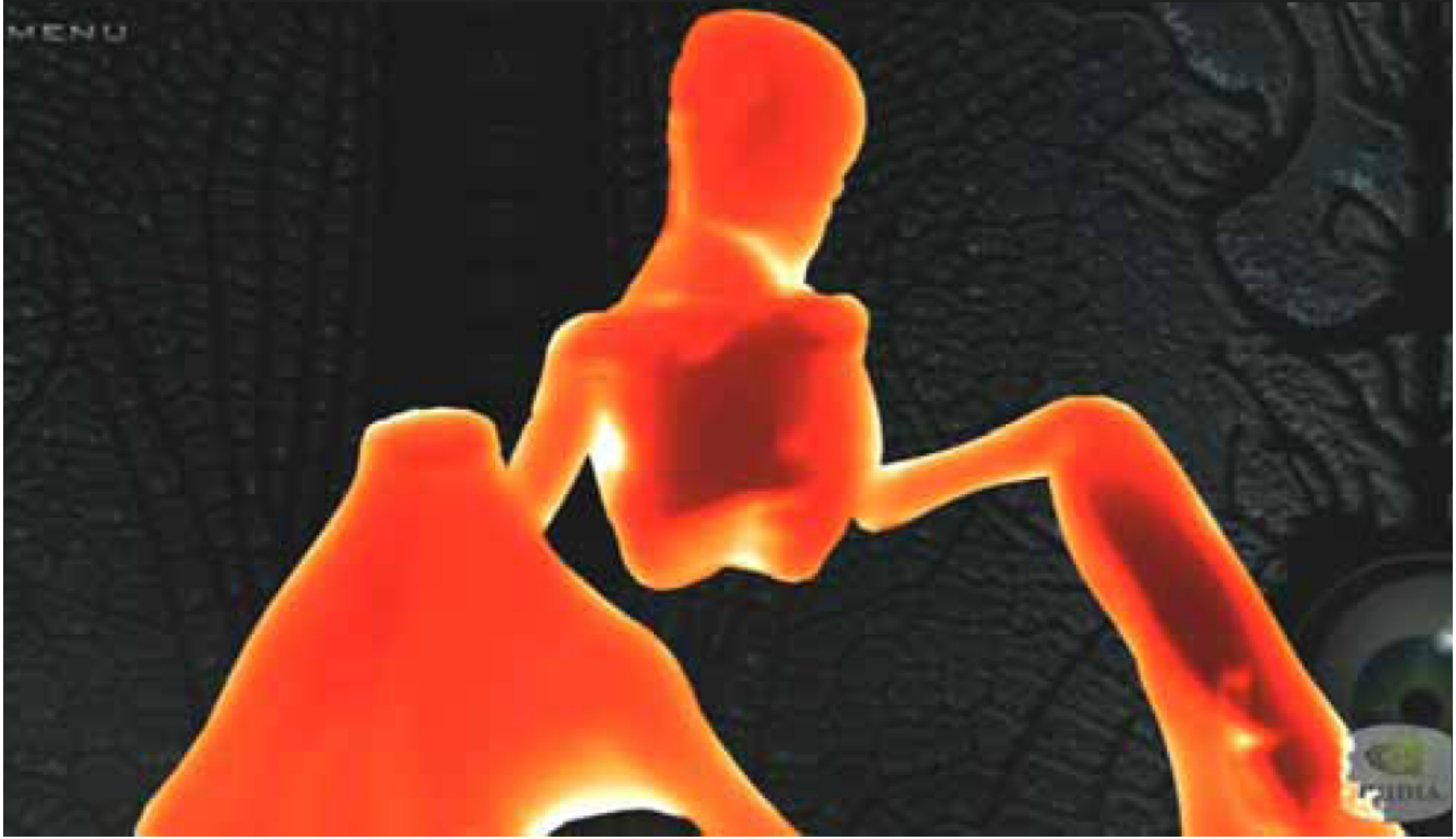


- Use computed approximation but smooth it using 11x11 sample blur
  - 2 extra passes
- Result is smooth enough not to produce visible artifacts in final composition

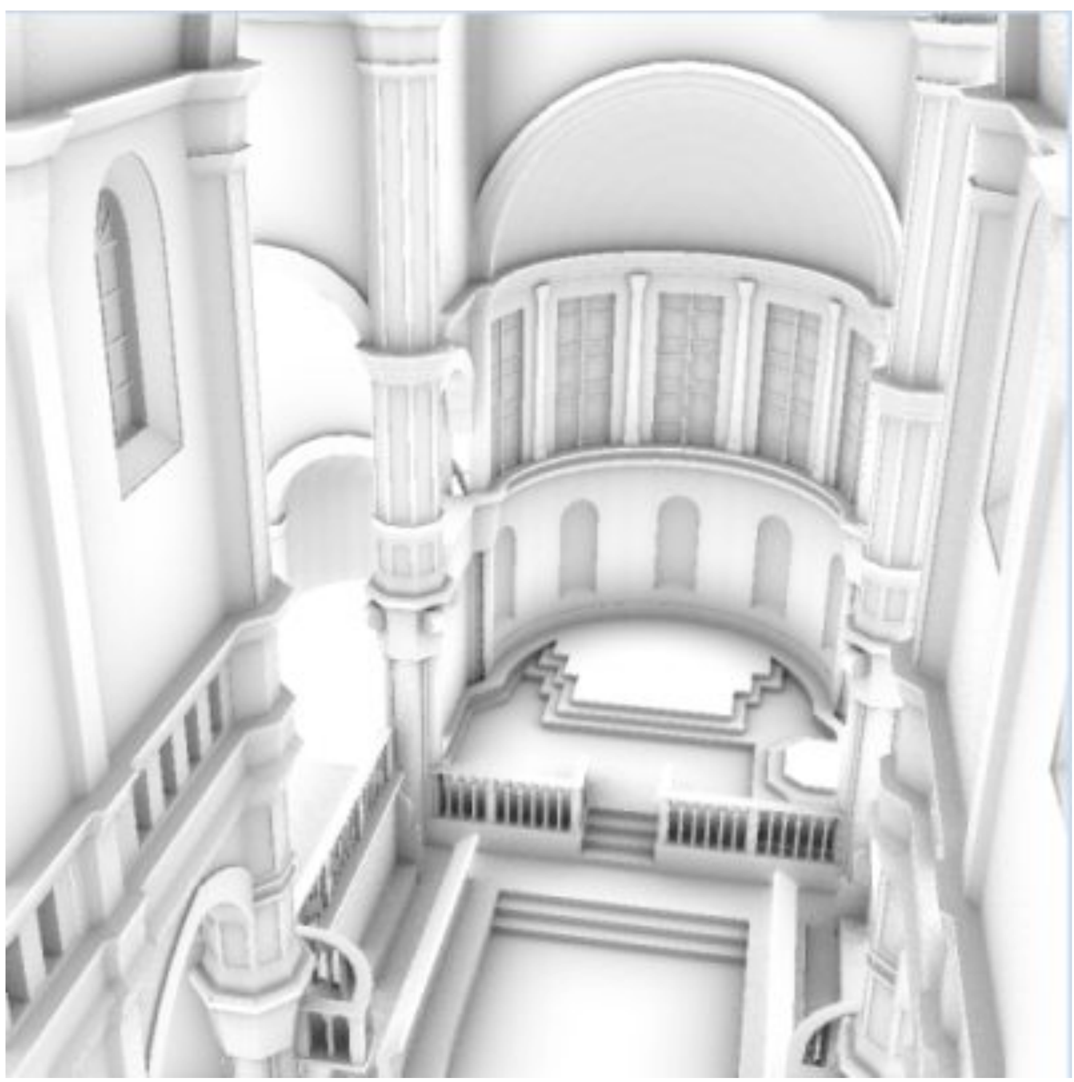
Translucent Skin Color

- Given thickness of the surface, find the color of the skin when the light travels through it
- Use the normalized thickness as a texture coordinate for the following texture
  - Allows for good control over how each of the densities of the surface looks and how quickly they change
  - Thicker parts end up dark red, membranes end up faint orange

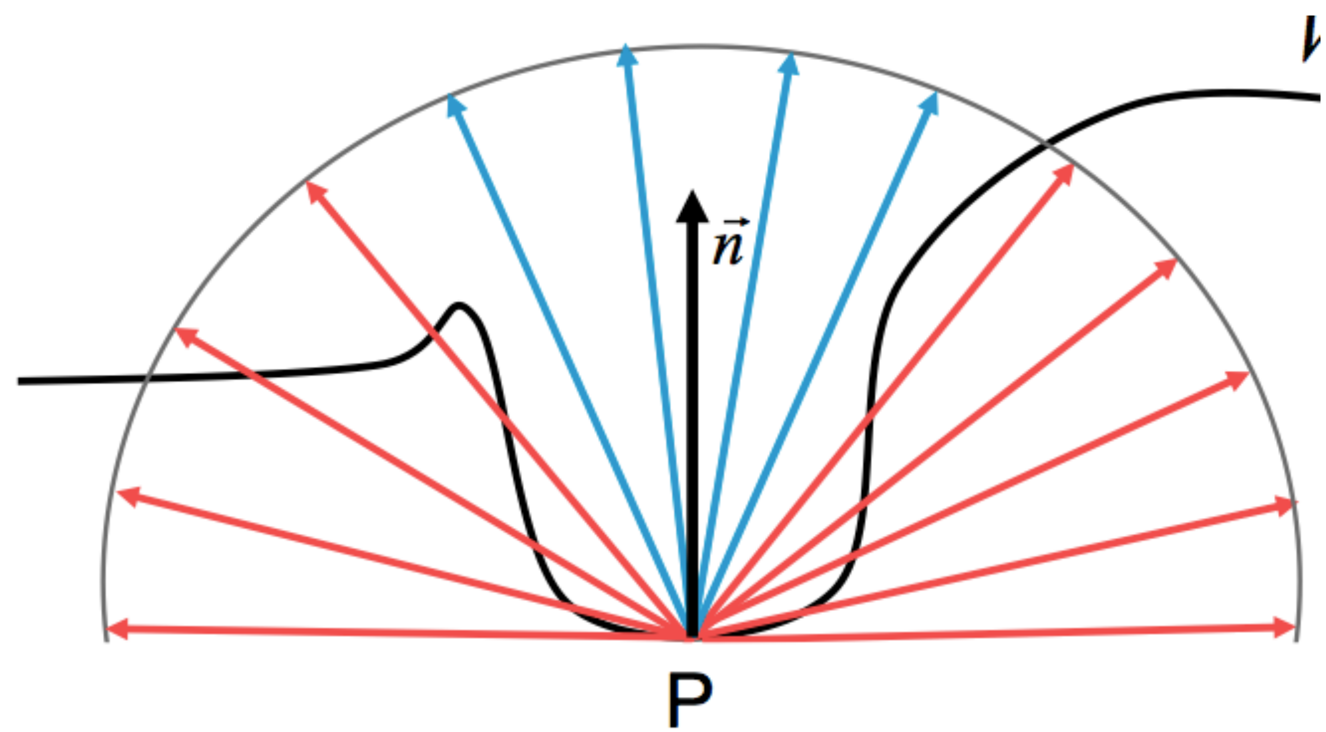
Layer in final composition



Ambient Occlusion



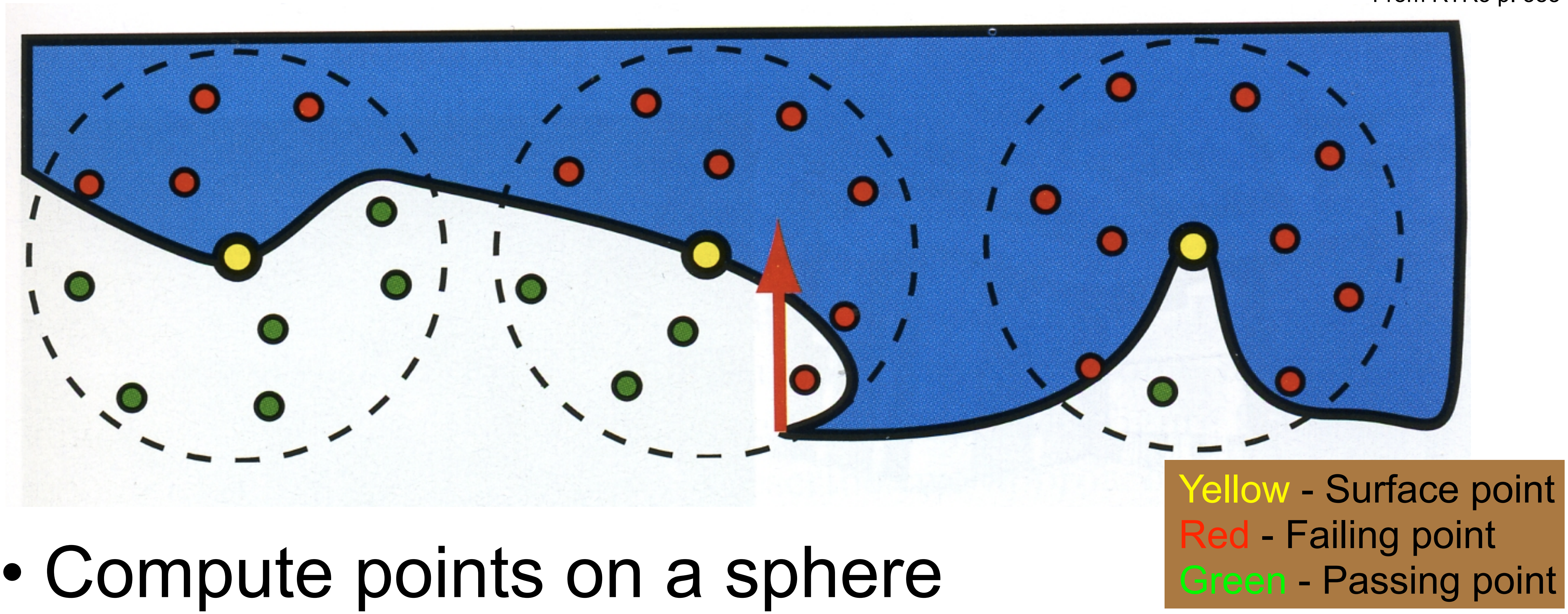
- How much ambient light hits a point?
- Calculate the local occlusion around a point in the scene



2D cross section of 3D **hemisphere**

Screen Space Ambient Occlusion (SSAO)

From RTR3 p. 383



- Compute points on a sphere
- Compare z and count passing points
- Divide by number of points in hemisphere

Mittring07, “Finding Next Gen-CryEngine2”, ARTRi3DGG, SIGGRAPH course

Screen Space Ambient Occlusion (SSAO)

From RTR p. 384

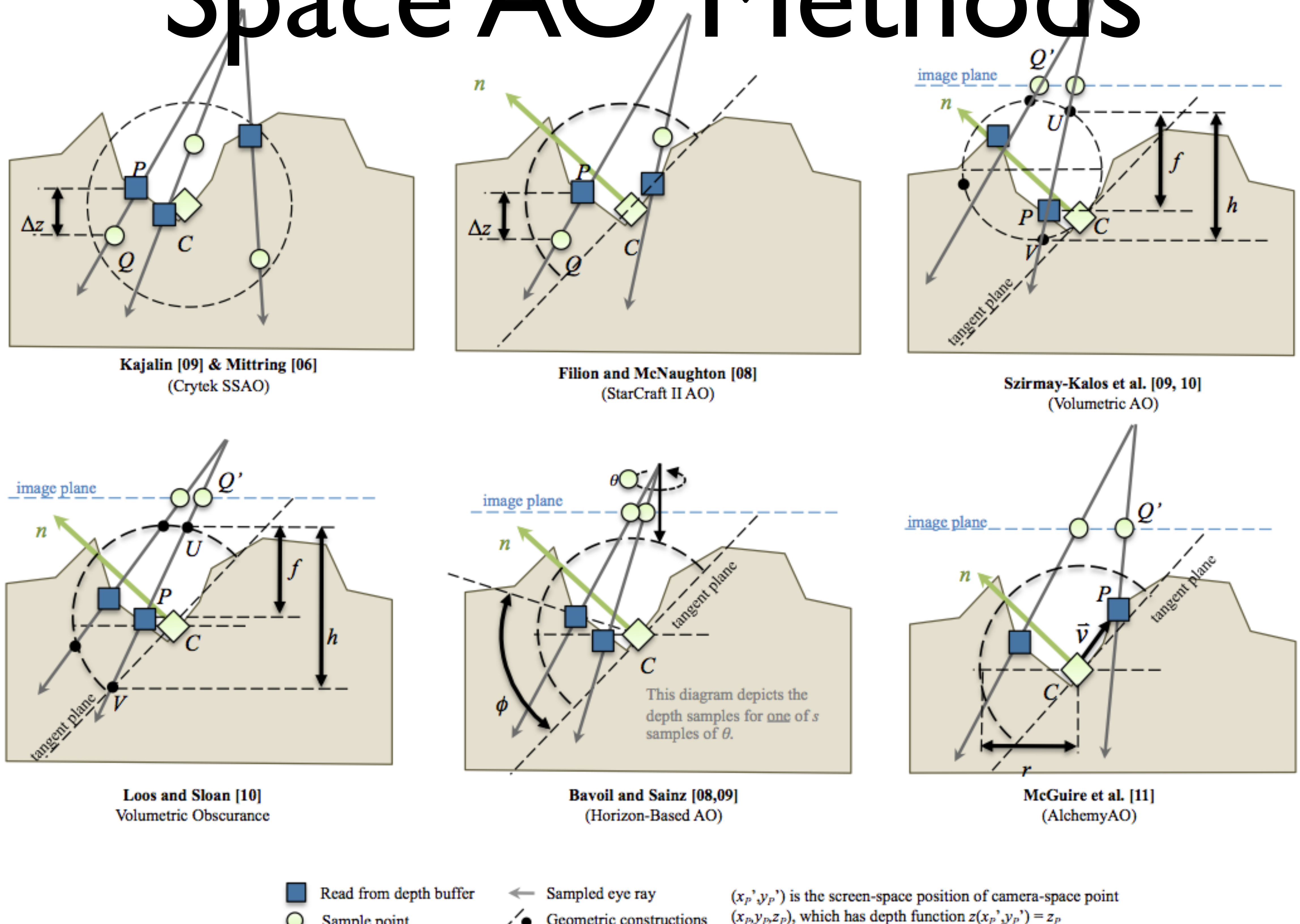
AO

albedo

AO + albedo

Left + specular + shadows

Overview of Screen-Space AO Methods



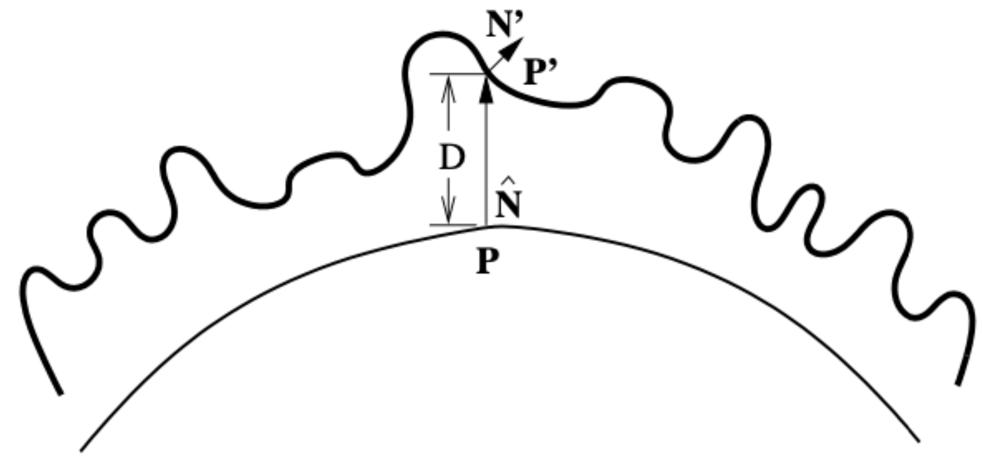
McGuire et al., "Scalable Ambient Obscurance", High Performance Graphics 2012

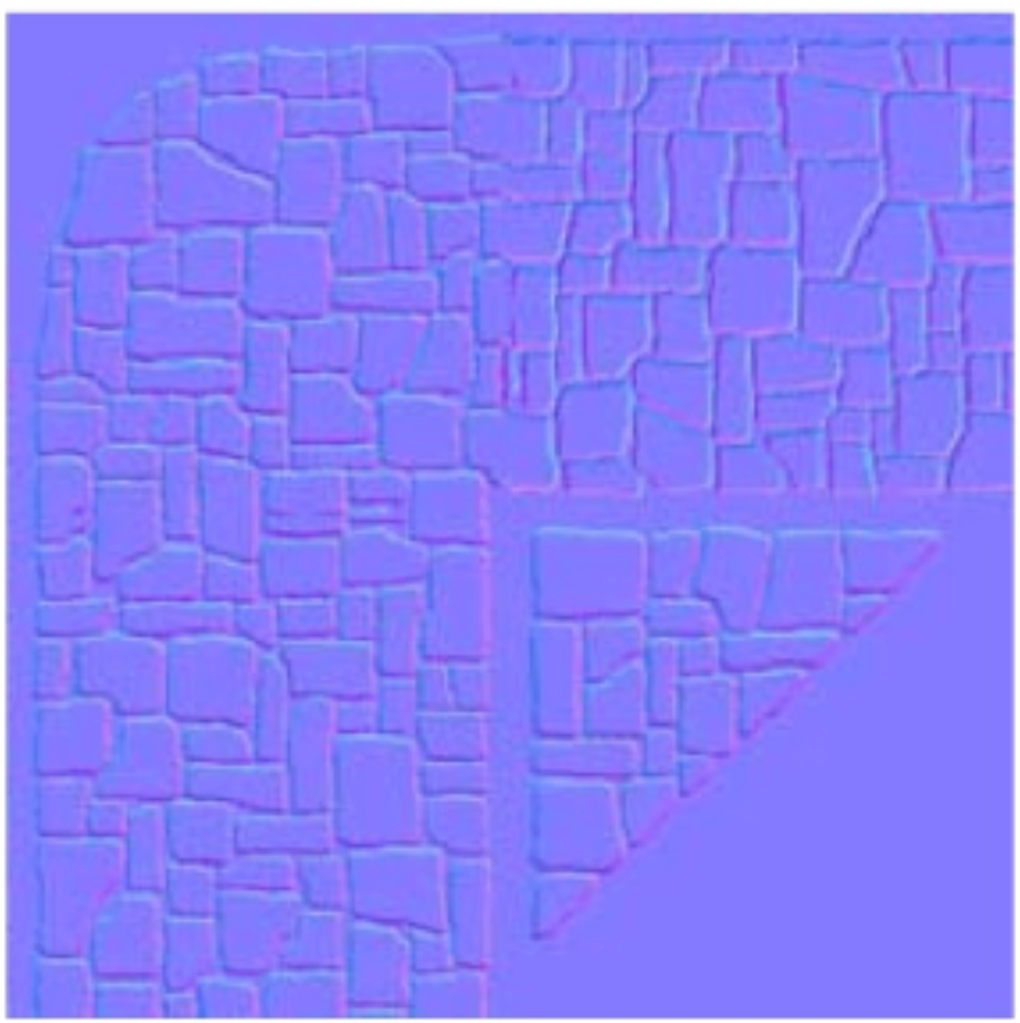
Scalable Ambient Obscurance

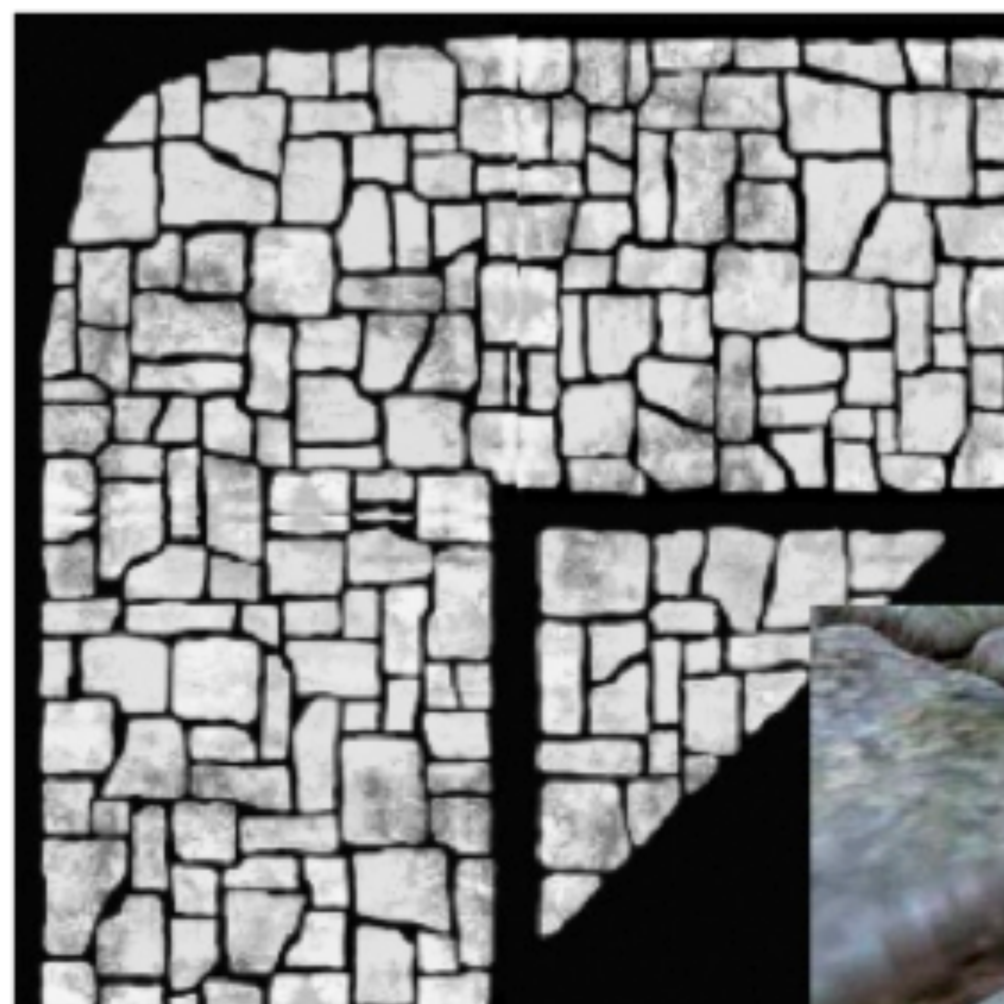
McGuire et al., “Scalable Ambient Obscurance”, High Performance Graphics 2012

Adding surface details

- Normal/Bump mapping
- Displacement mapping
- Parallax mapping





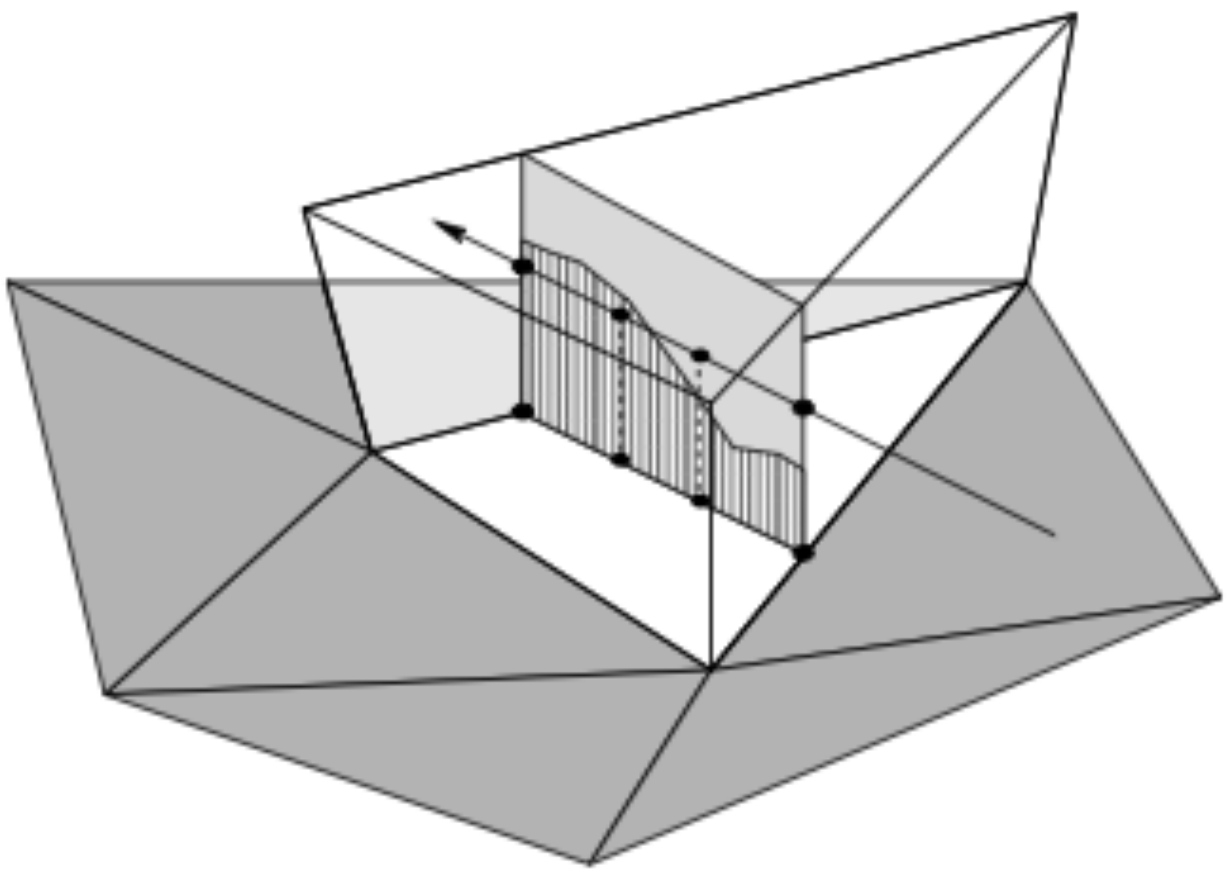


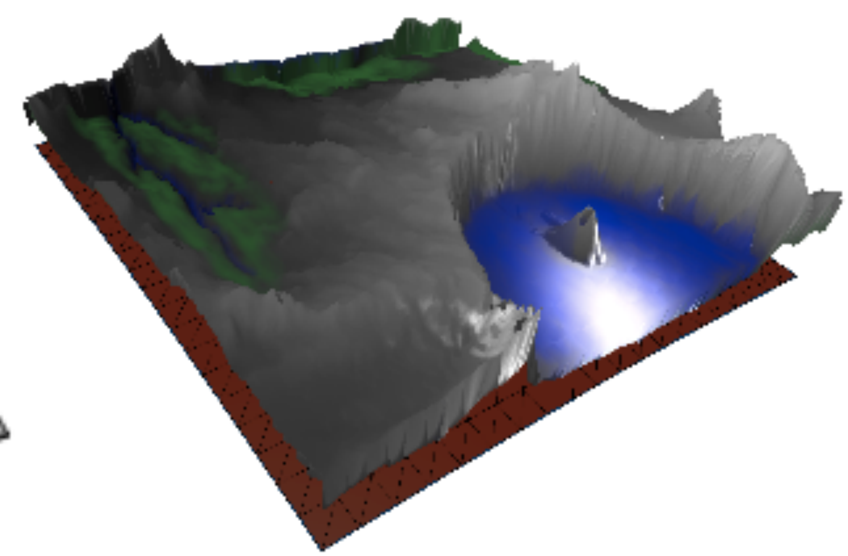
Parallax Occlusion Mapping

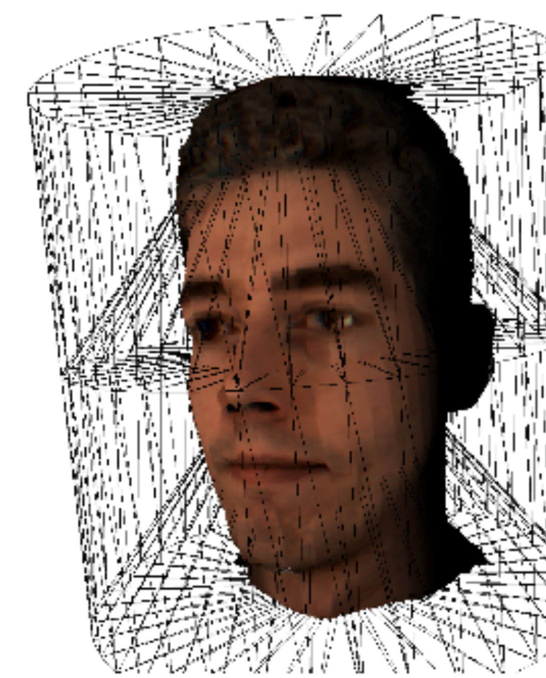
courtesy http://ati.amd.com/developer/gdc/2006/GDC06-Tatarchuk-Parallax_Occlusion_Mapping.pdf

Per-Pixel Displacement Mapping

- Extrude triangles up from the surface
- Ray trace through the displacement map



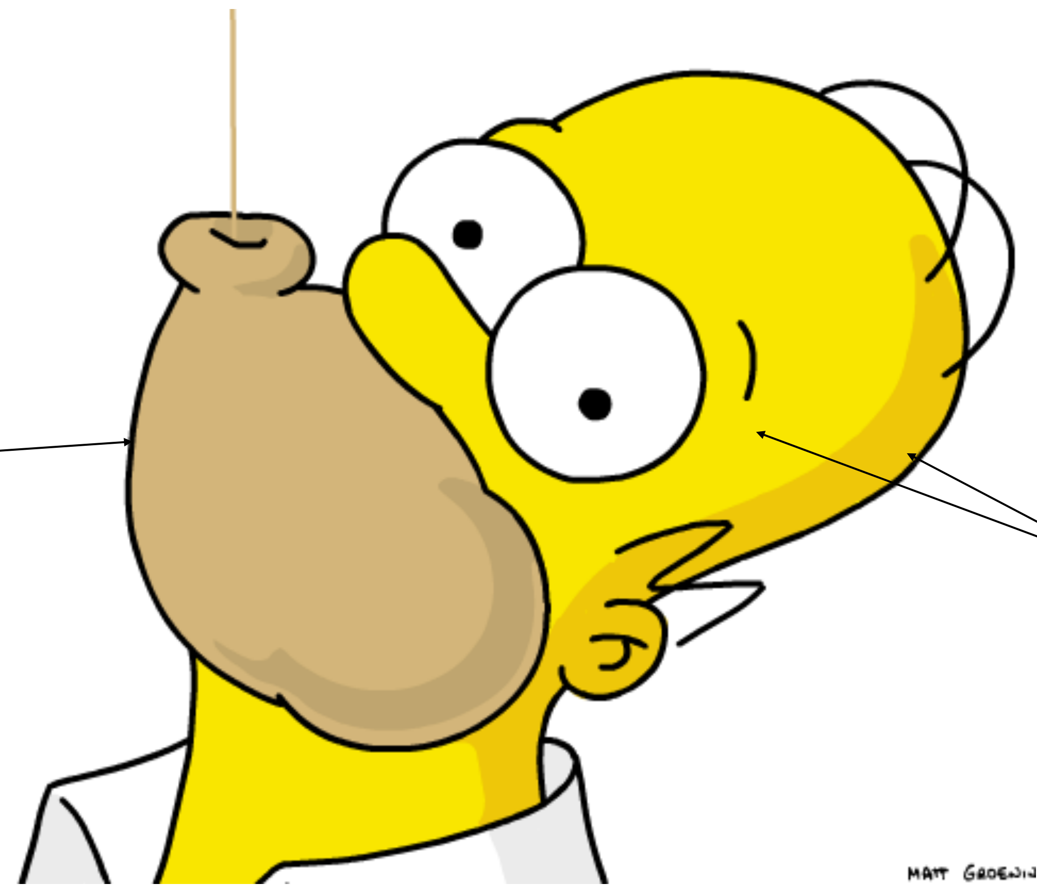




“Hardware Accelerated Per-Pixel Displacement Mapping”, Hirche et. al., Graphics Interface 2004

Toon Shading

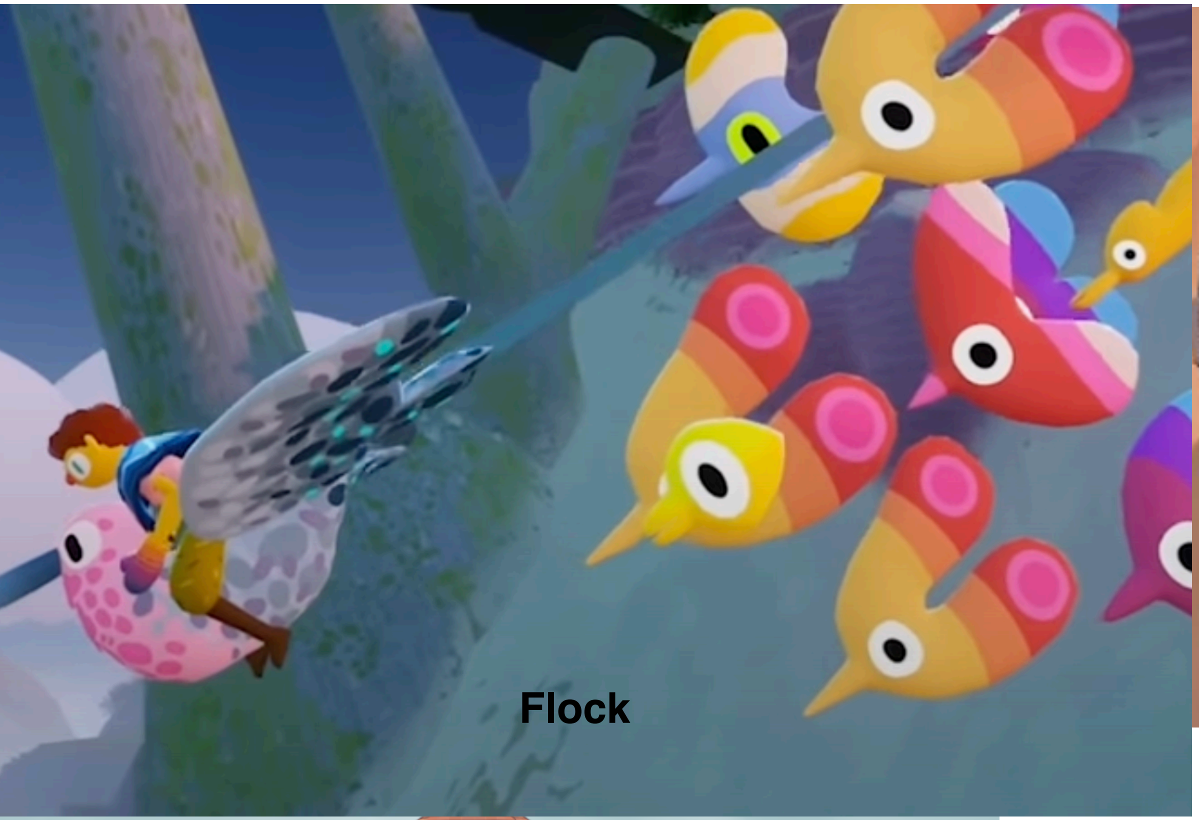
Dark outlines of important silhouettes

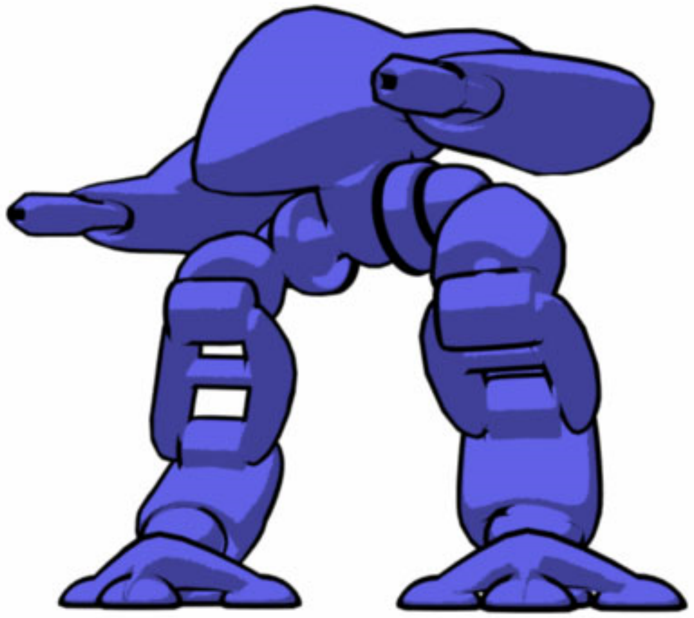


Few shades in shading

- Characteristics?
- Why not do it using shaders?

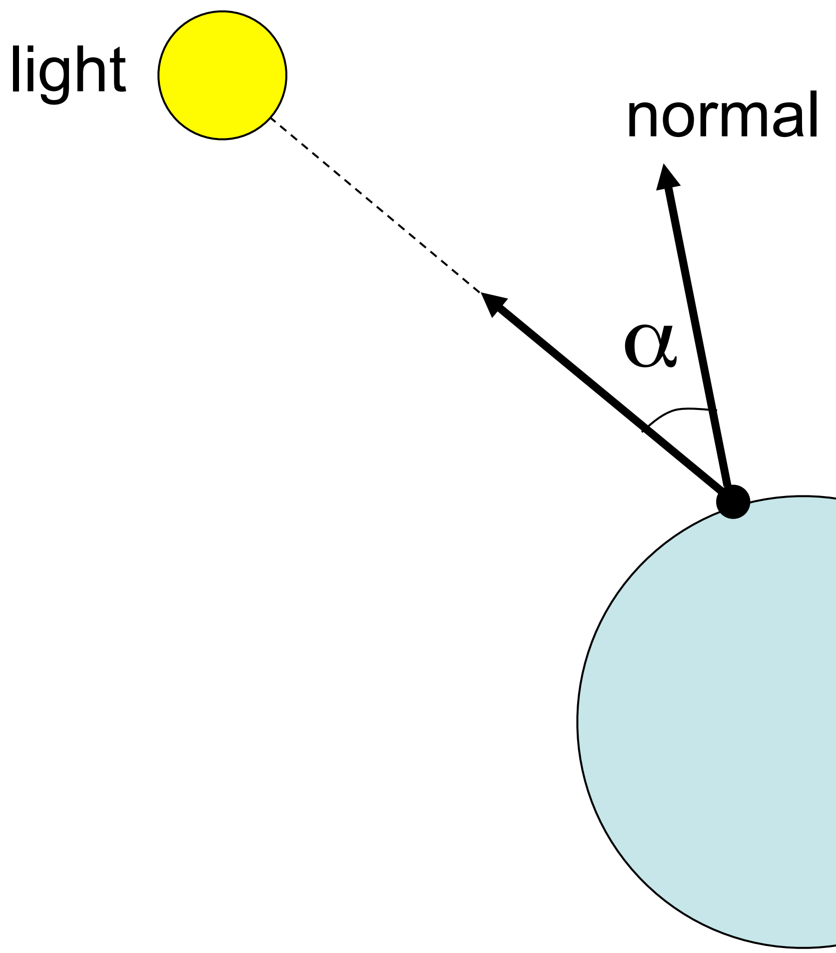
Some Cartoon renderings





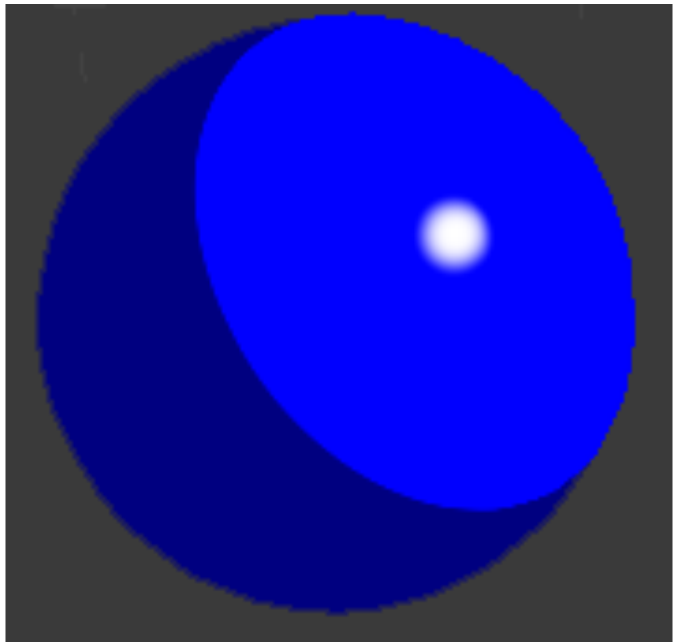
Few shades...

- Simple
  - Two slightly different ways of doing it



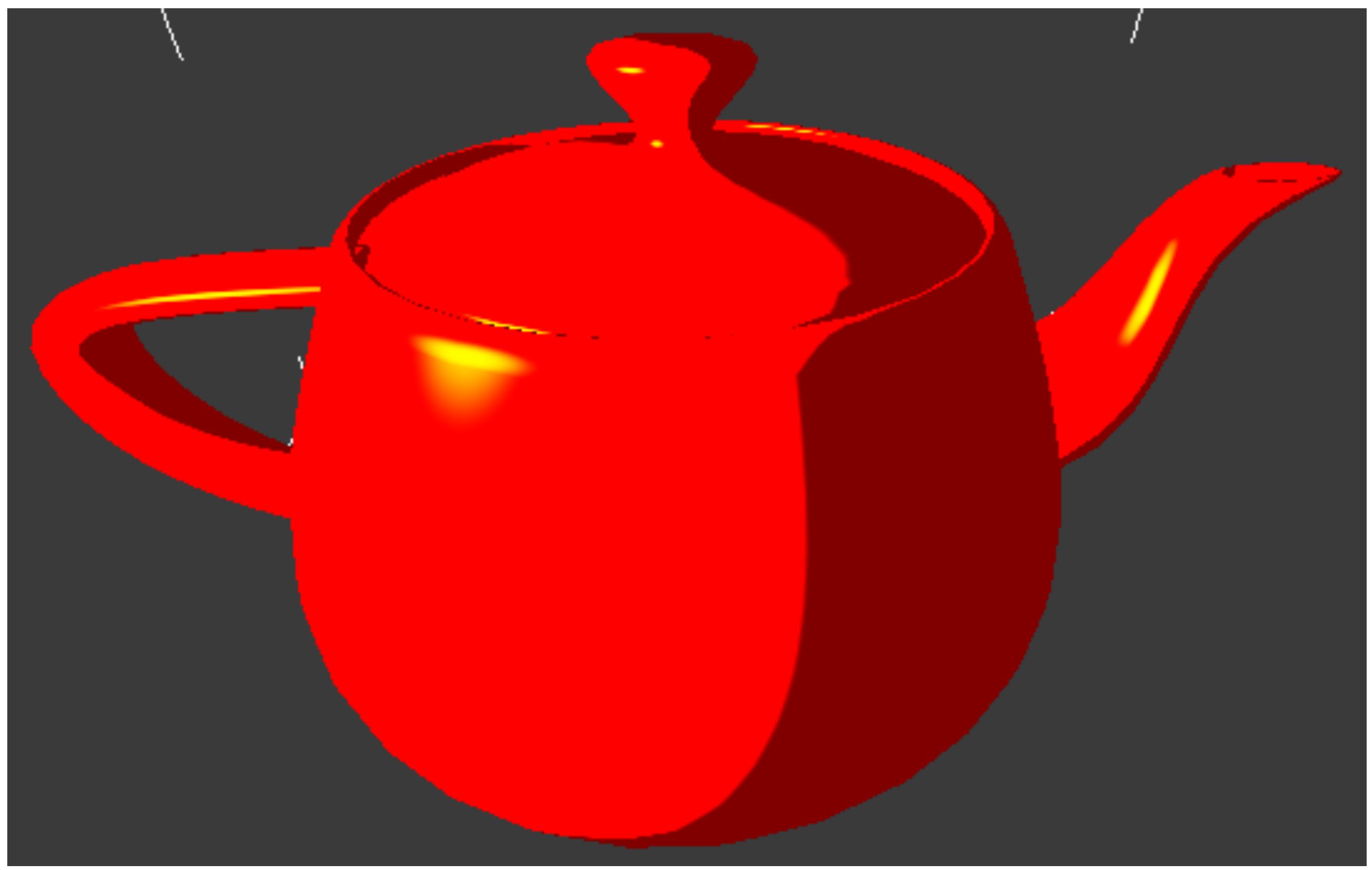
$$\cos(\alpha)=\mathrm{dot}(\mathbf{n},\mathbf{l})$$

- Let different intervals on $\cos(\alpha)$ correspond to different colors
  - Do it using computations or a 1D texture

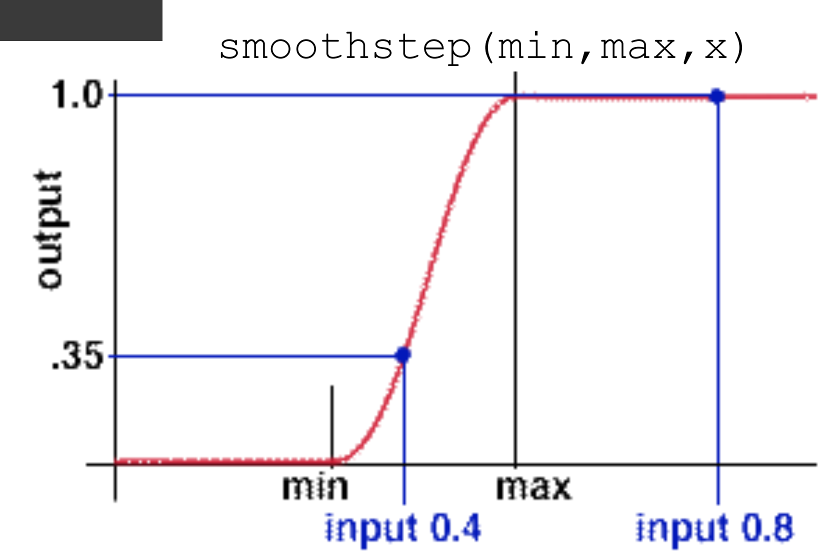


Few shades (cont'd)

- A bit more complex example
  - Uses entire Phong shading equation, with step functions to do thresholding

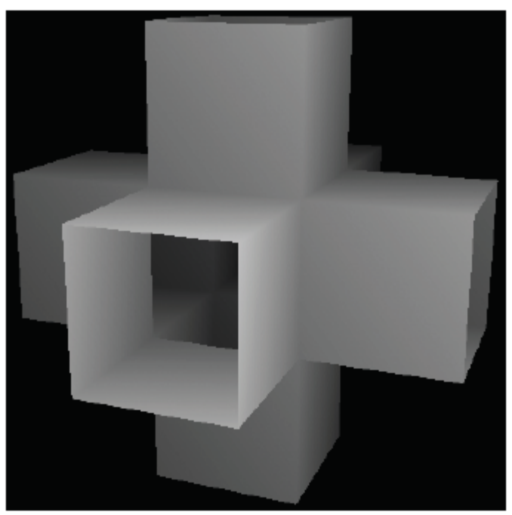


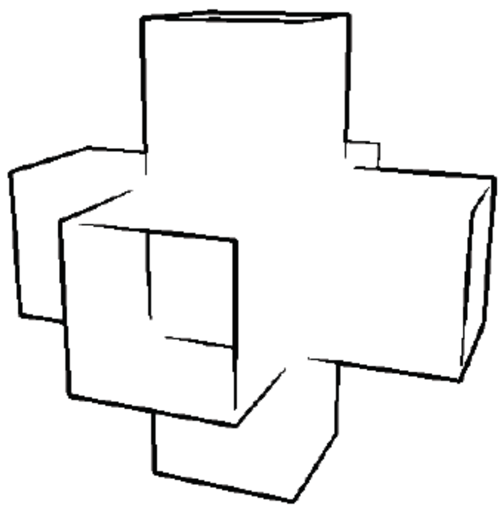
- Filtering:
  - if you use 1D textures, you can just turn on mipmapping
  - If you compute: use smoothstep()

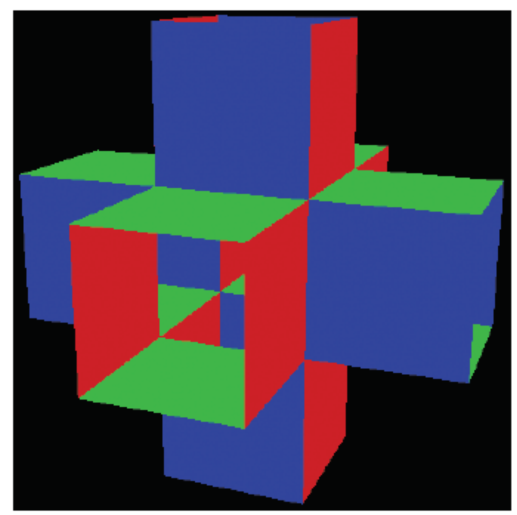


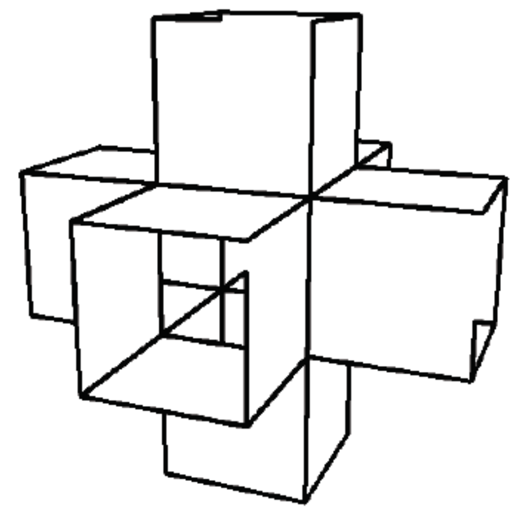
Silhouette rendering

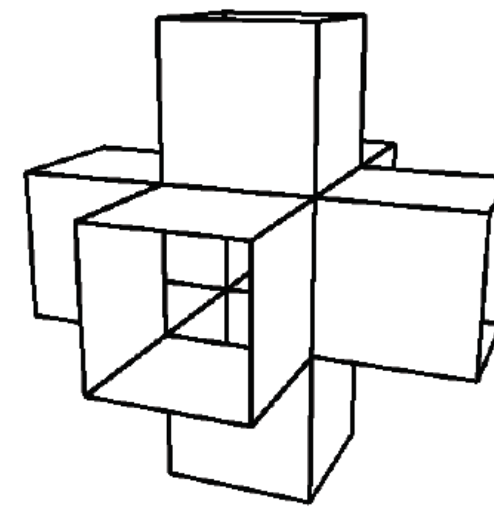
- In screen space:
  - Use edge detection
  - On both depth buffer and normal buffer
  - Finds both silhouettes and important ”internal silhouettes” (creases)


Depth buffer


Edges derived from depth buffer


Normal buffer

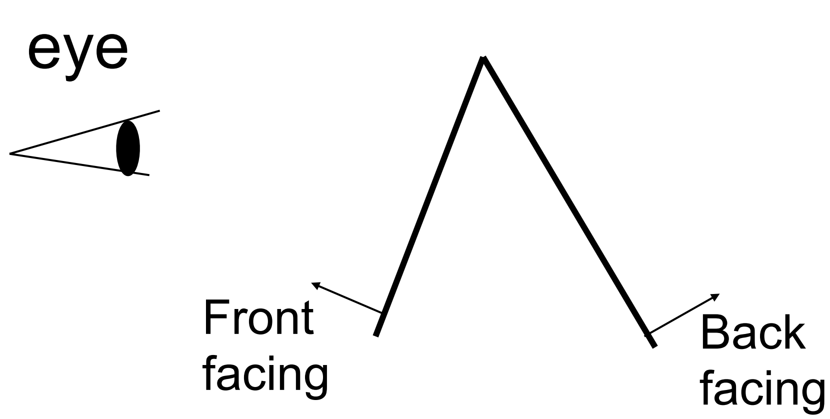

Edges derived from normal buffer

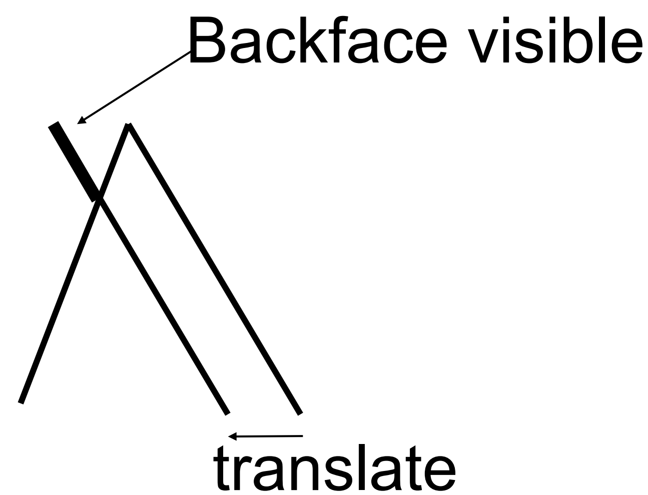

Combination: normal + depth

Images courtesy of Aaron Hertzmann

Silhouette rendering (cont'd)

- **Procedural geometry silhouetting**
- Basic idea:
  - Render frontfaces as usual, and then
  - Render backfaces so the silhouettes become visible

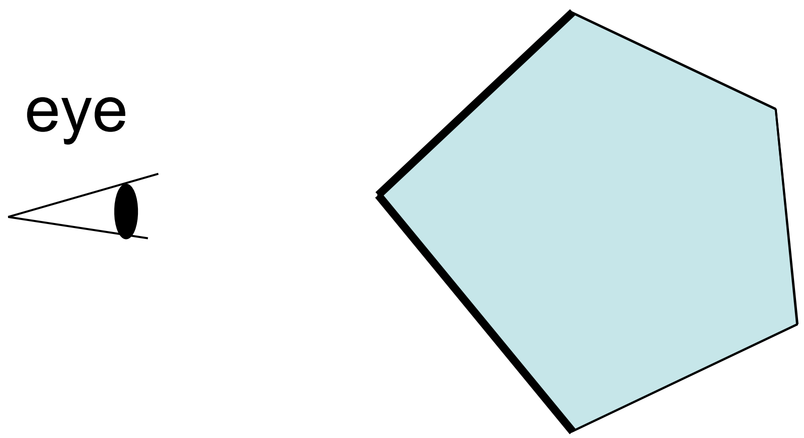




Problem: thickness depends on orientation of backfacing triangle

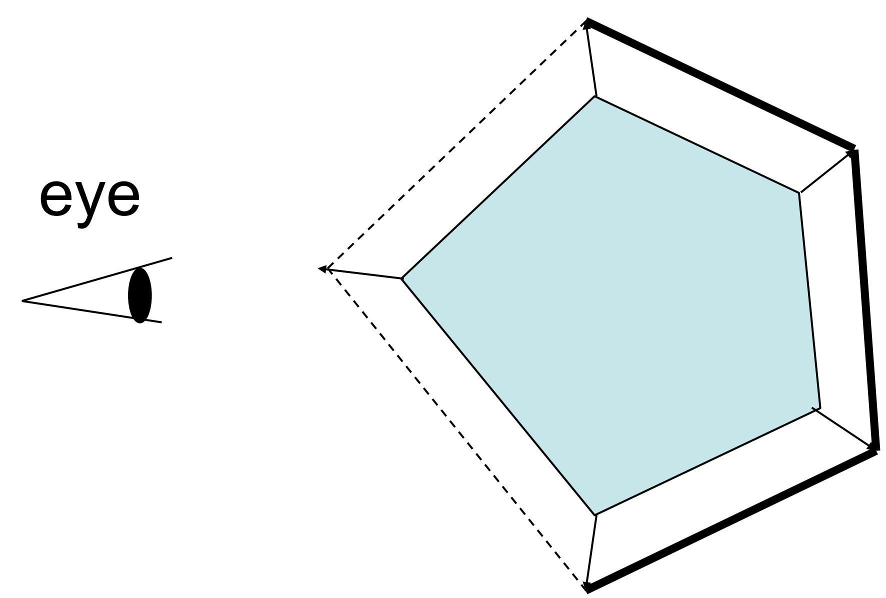
Silhouettes

- Using enlarged objects technique
  - First pass render front faces

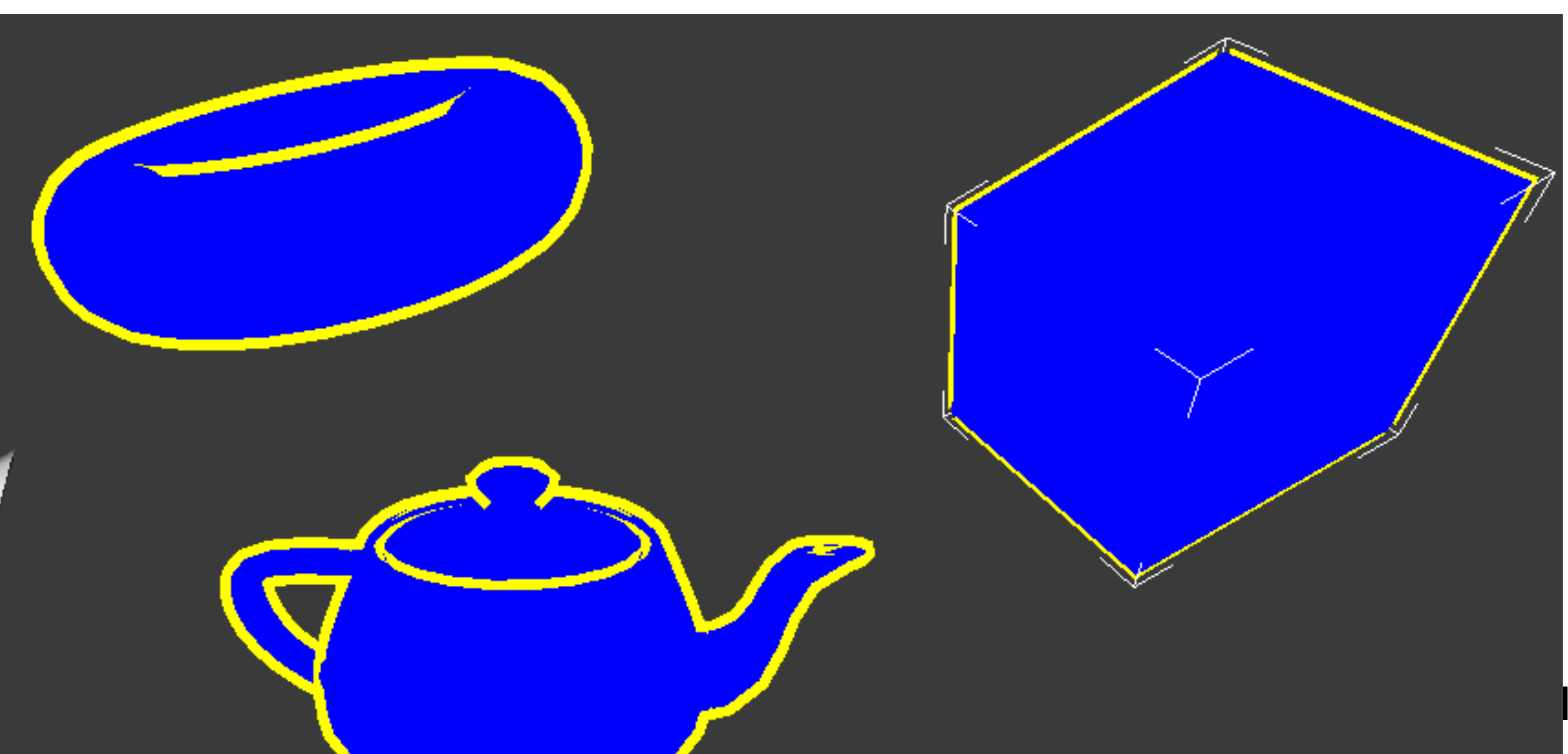


Frontfaces are rendered

Second pass:

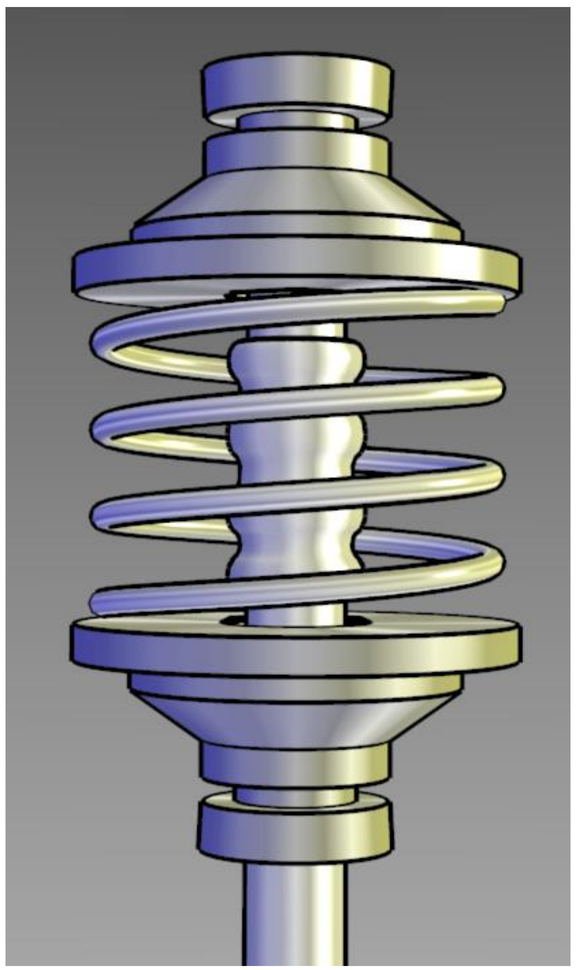


Extruded backfaces are rendered in silhouette color



Non-photorealistic rendering

- Cartoon rendering is one example of this



Technical illustration

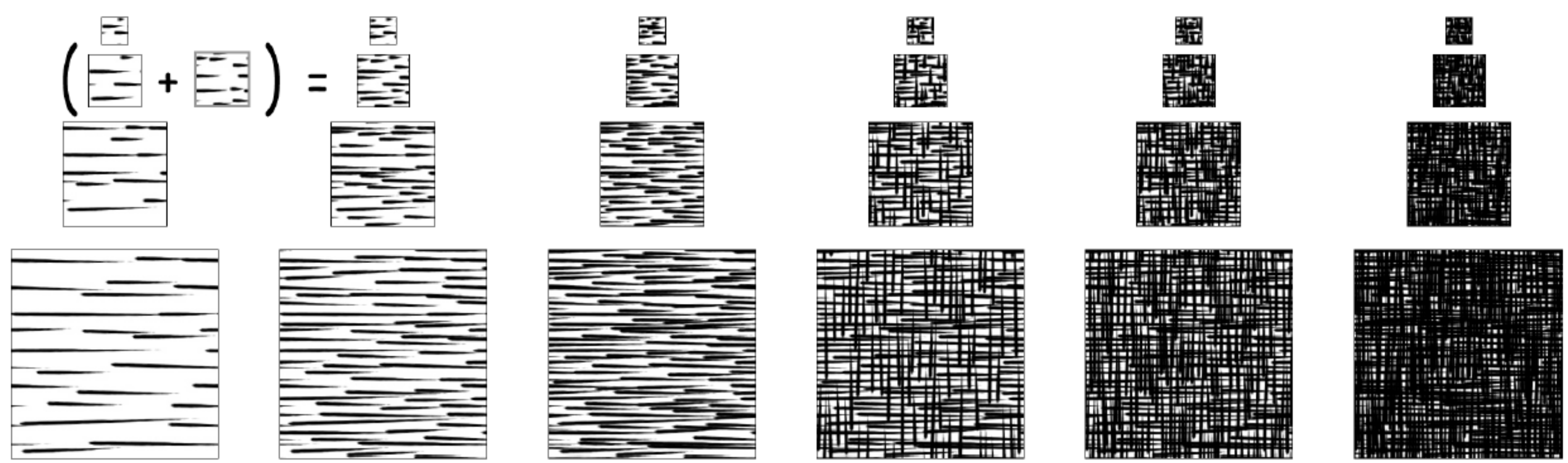
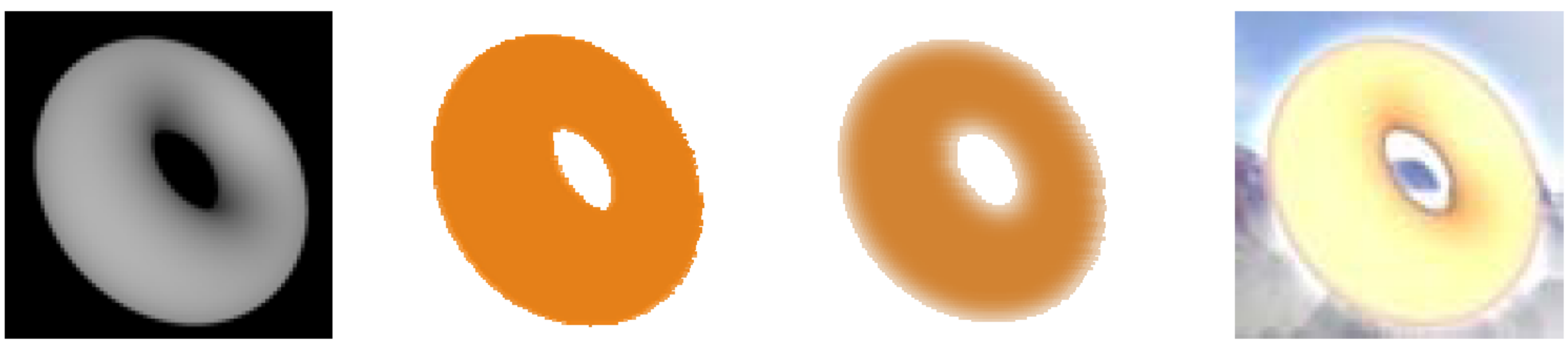


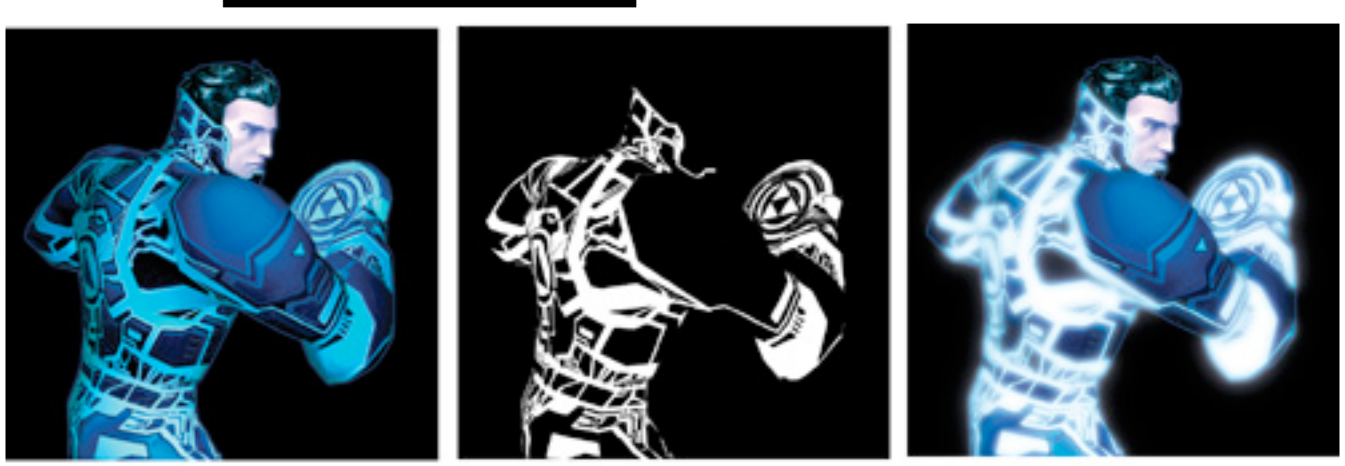
Figure 2: A Tonal Art Map. Strokes in one image appear in all the images to the right and down from it.

"Real-Time Hatching", E. Praun et. al., SIGGRAPH 2001

Glow

- Render object in ”glow color” to separate texture
- Apply low-pass blur filter
- Blend with rendering of object





(a) (b) (c)

“Real-Time Glow”, Greg James, John O’Rorke, Chapter 21, GPU Gems

”Fur”

Closeup

Images courtesy of Pixar Inc.

Real-time fur rendering

(a) surface (inner, opaque shell)

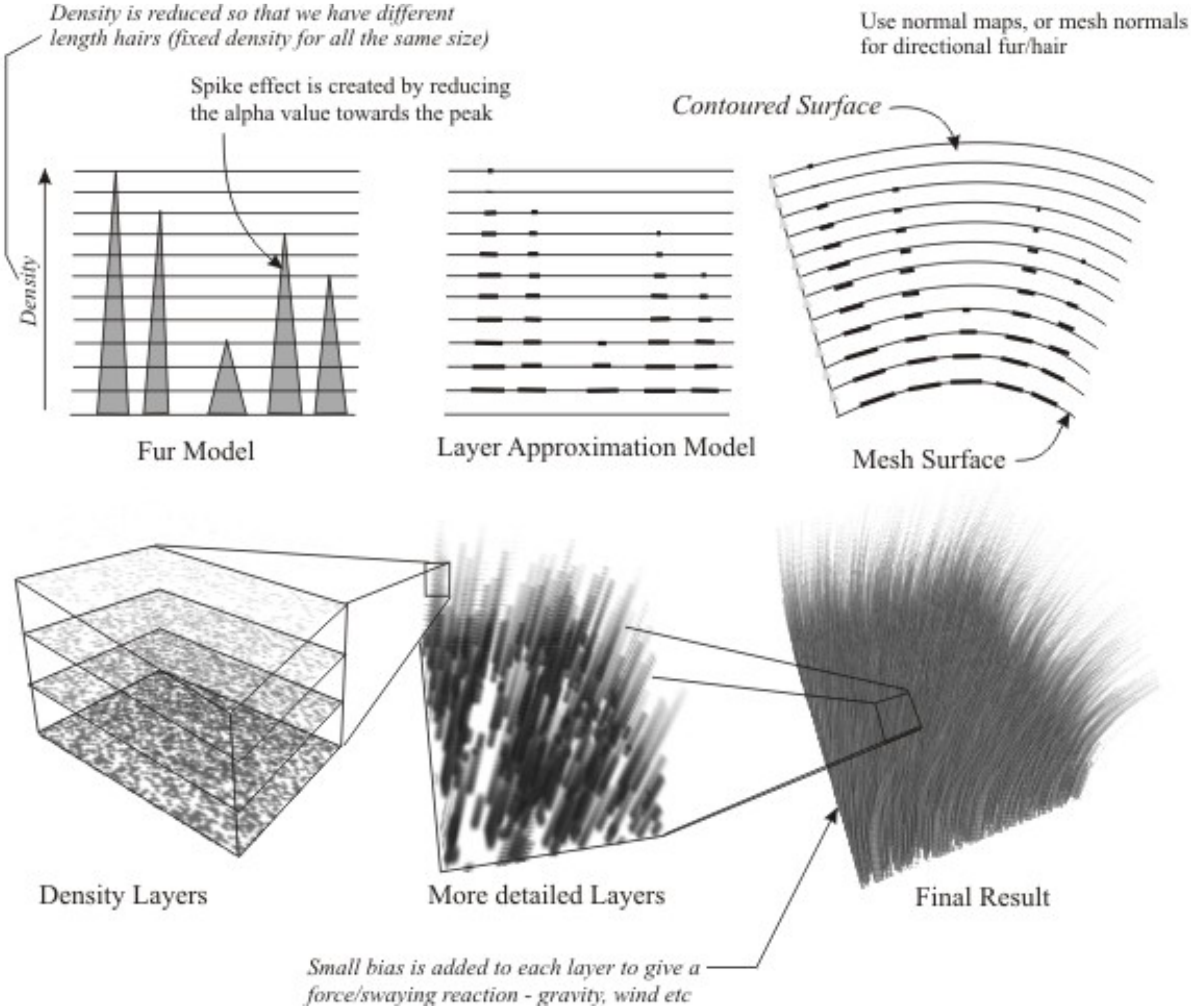
(b) fins (alpha-blended)

(c) shells (non-overlapped patches)

(d) final image (a+b+c)

- Render ”inner surface”
- Render ”fins”, around silhouttes
- Render many concentric shells, semi-transaprent

“Real-Time Fur over Arbitrary Surfaces”, Jerome Lengyel, et. al. I3D 2001



Density is reduced so that we have different length hairs (fixed density for all the same size)
Spike effect is created by reducing the alpha value towards the peak
Use normal maps, or mesh normals for directional fur/hair
Contoured Surface
Density
Fur Model
Layer Approximation Model
Mesh Surface
Density Layers
More detailed Layers
Final Result
Small bias is added to each layer to give a force/swaying reaction - gravity, wind etc

MD22

Next ...

- Work on assignment 1 - Ray Tracing
  - Read the assignment
  - Get the code running
  - Post questions on Discord
- Next week
  - Monday - Seminar
    - Shadows and Deferred shading - Lab 2
  - Tuesday/Wednesday Lab
    - Assignment 1 - Ray Tracing
  - Thursday - Lecture
    - Performance