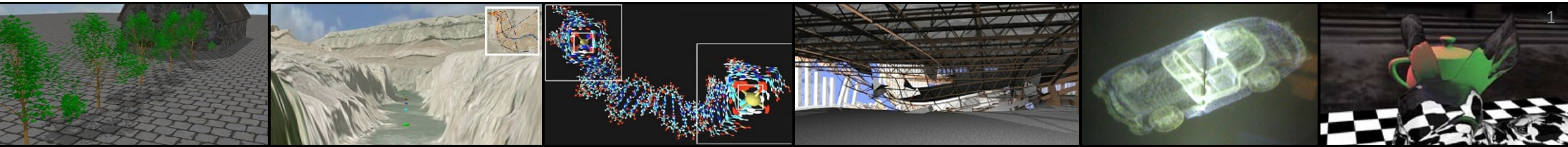
CIS 4930-001: Introduction to Augmented and Virtual Reality

Lighting, Shading, and Other Effects

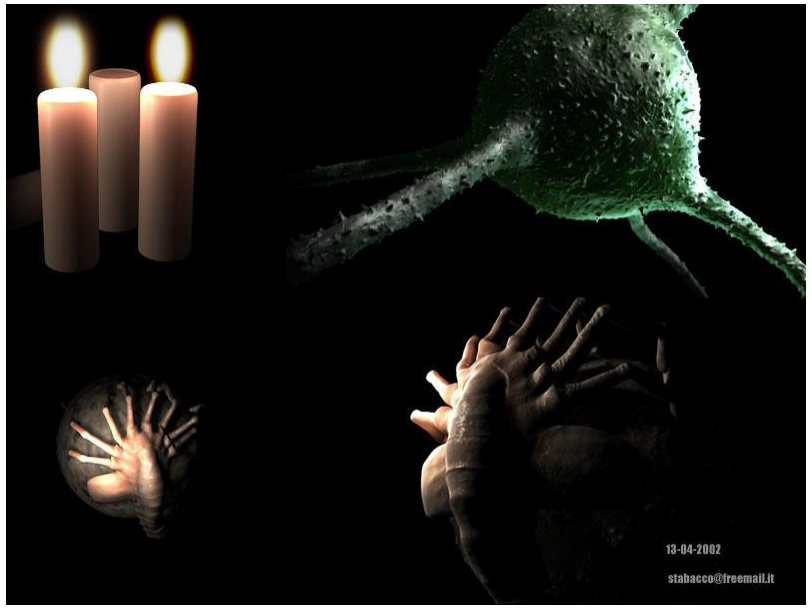
Paul Rosen
Assistant Professor
University of South Florida

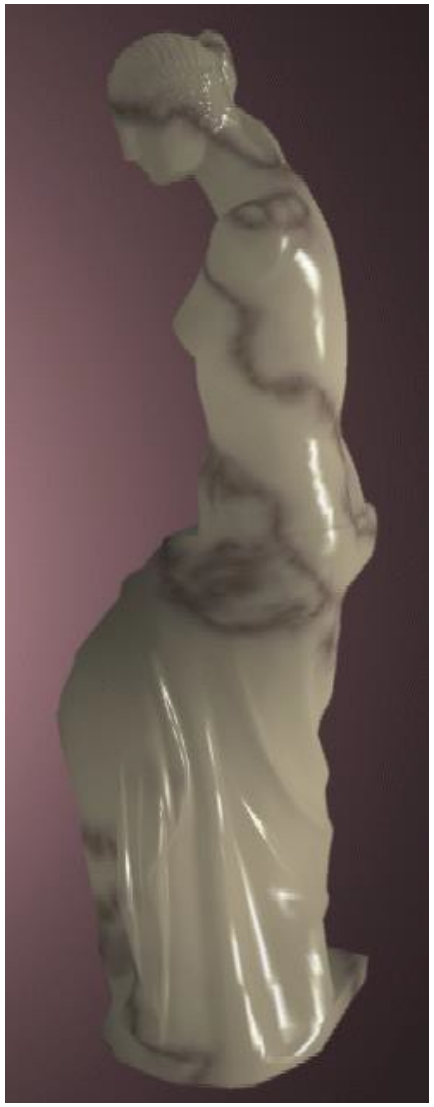
Some slides from: Anders Backman, Mark Billinghurst, Doug Bowman, David Johnson, Gun Lee, Ivan Poupyrev, Bruce Thomas, Geb Thomas, Anna Yershova, Stefanie Zollman



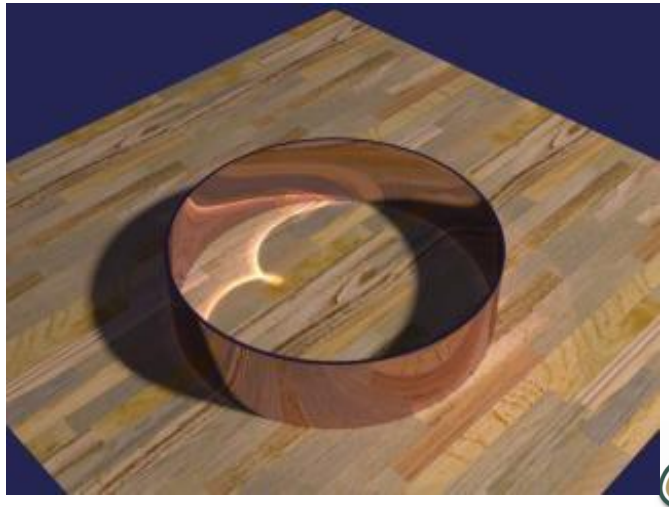
GOAL OF RENDERING – REALISM

Subsurface scattering





Caustics, Jensen et al.

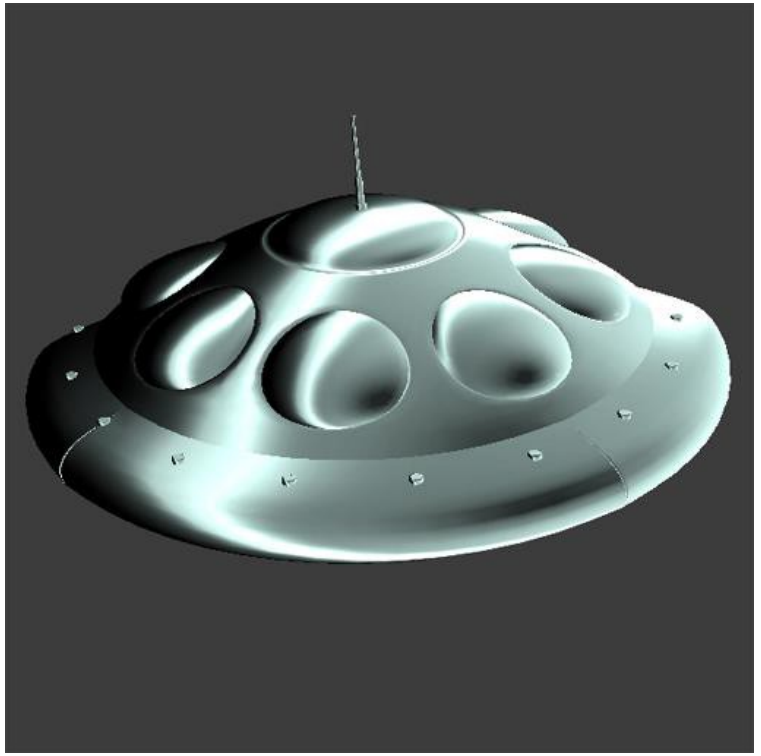


More Examples

High Dynamic Range Imaging

Refraction

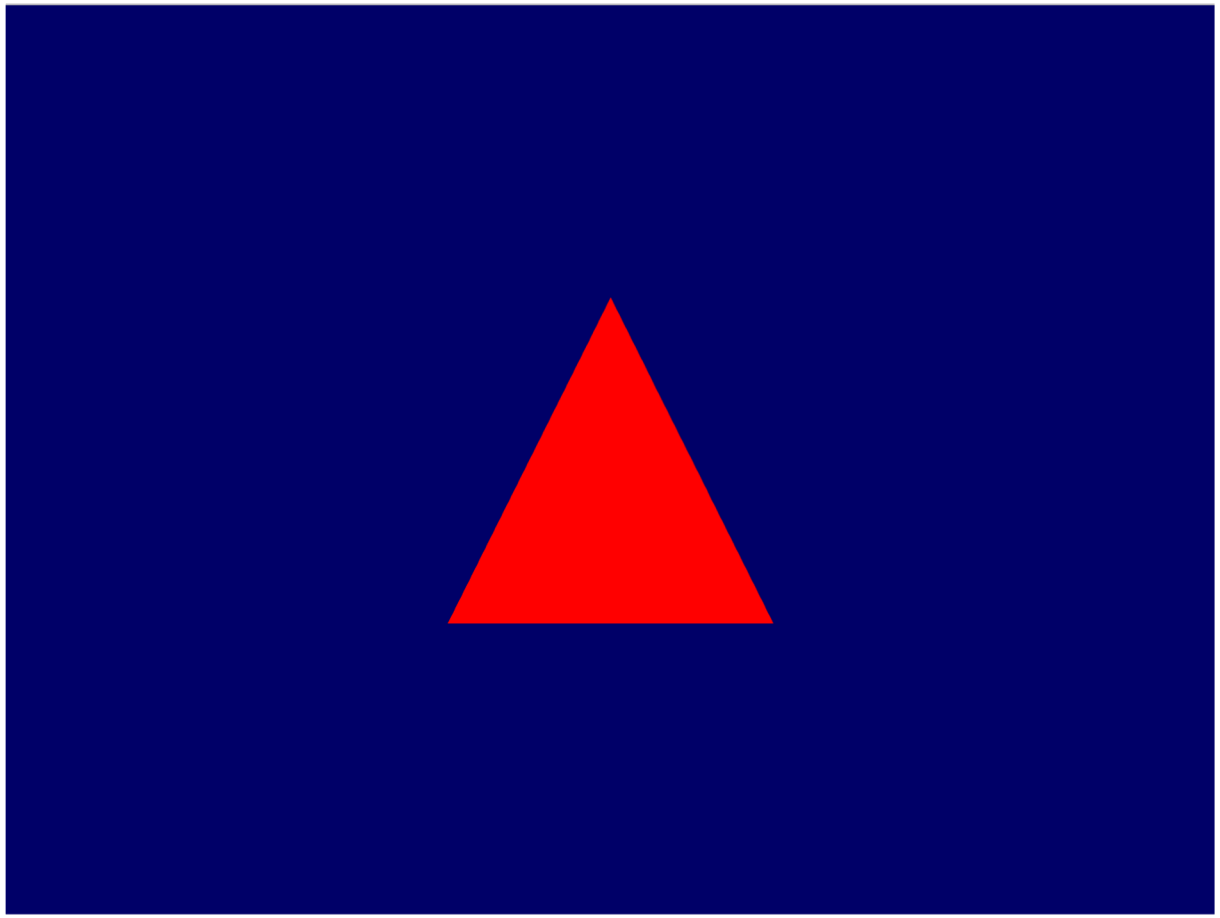
Anisotrophic lighting



Remember, Real time bugs life – Unreal Engine

https://youtu.be/qlyzo9ll9Vw

WHAT IS MISSING?



WHY ILLUMINATION?

Illumination is important for perception and understanding of 3D scenes

Has visual cues for humans

Provides information about:

- Positioning of light sources
- Characteristics of light sources
- Materials
- Viewpoint

ILLUMINATION MODEL

Can be complex (most of which is out of scope for this course)

Equation for computing illumination includes:

- Light attributes (intensity, color, position, direction, shape)
- Surface attributes (color, reflectivity, transparency)
- Interaction between lights and objects

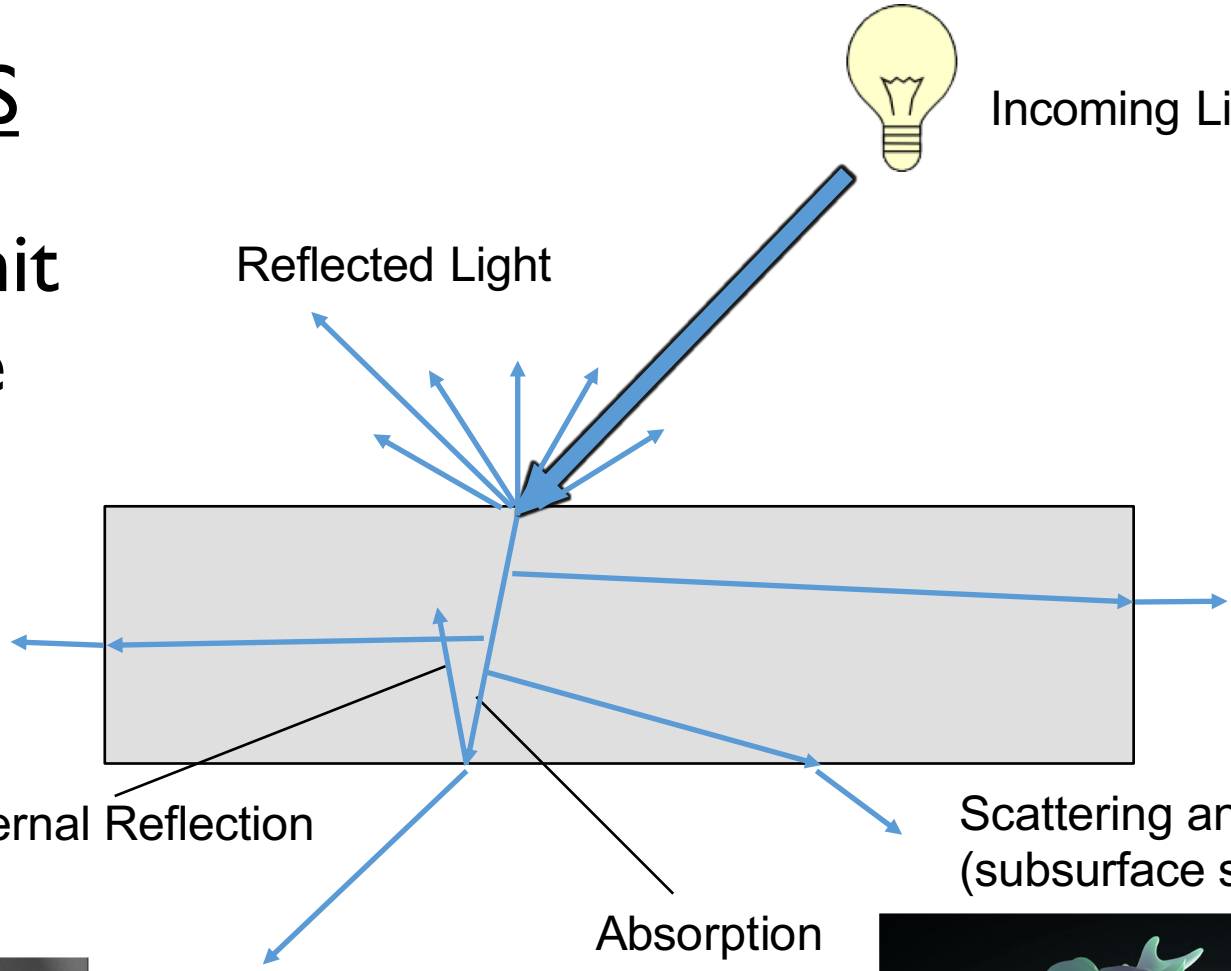
General rendering equation

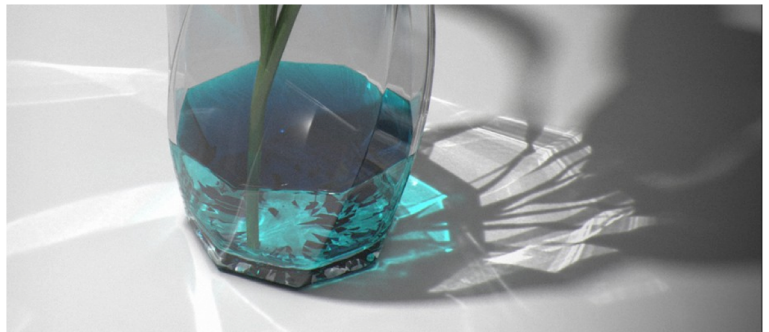
- Introduced 1986 by Kajiya
- Global illumination model

ALL ABOUT LIGHTS

When light (photons) hit a surface we have three different interactions

- Reflection
- Absorption
- Transmittance



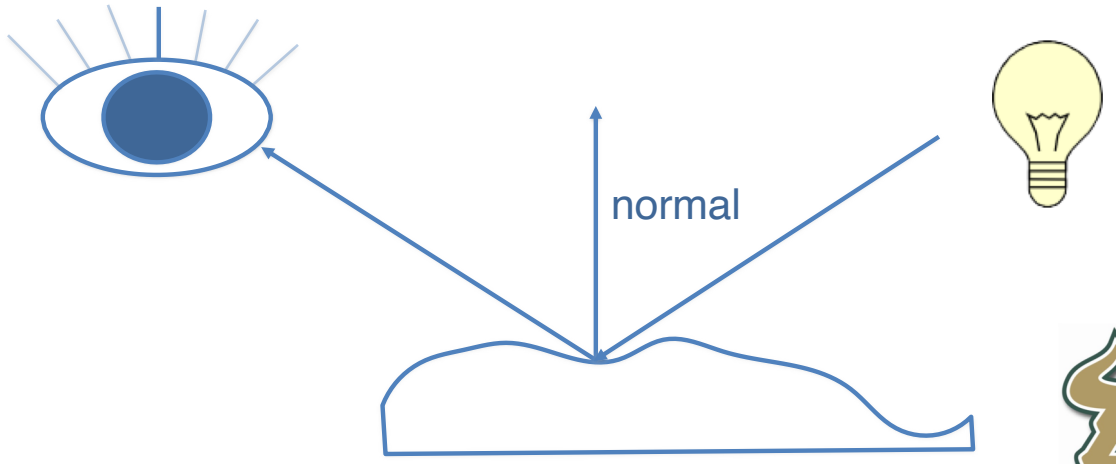


LOCAL ILLUMINATION MODEL

OpenGL cannot render full global illumination

We need an simplified approximation—local illumination model

- Use point light sources for simplification
  - Idealized mathematical point of view
  - In real life there are no point lights
- Does not consider light path after bouncing off other objects
- Function of:
  - Viewer position
  - Light source
  - Surface material properties
  - Geometry

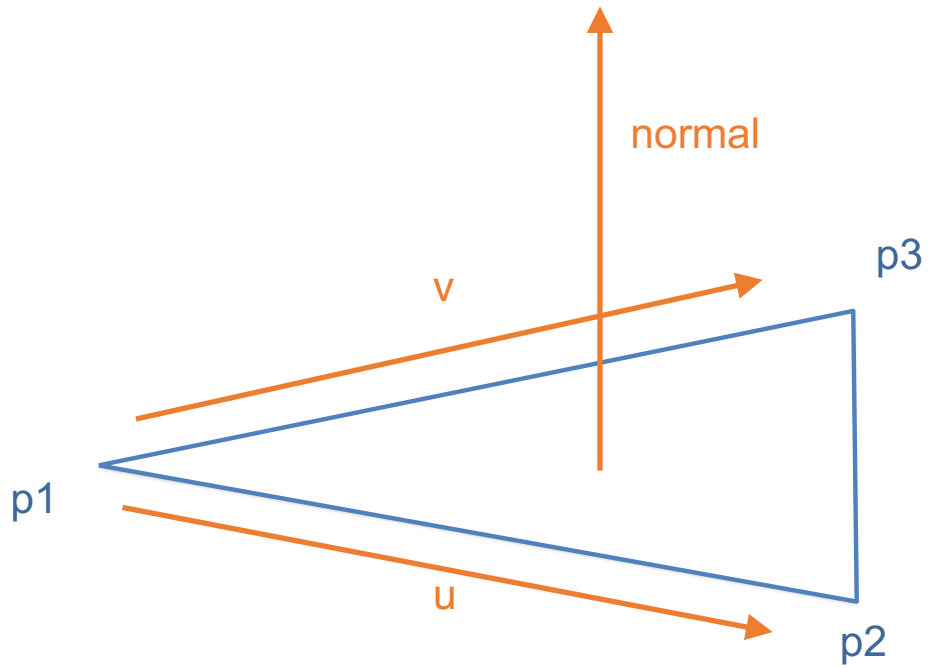


NORMAL

Perpendicular to tangent plane of surface

For triangles:

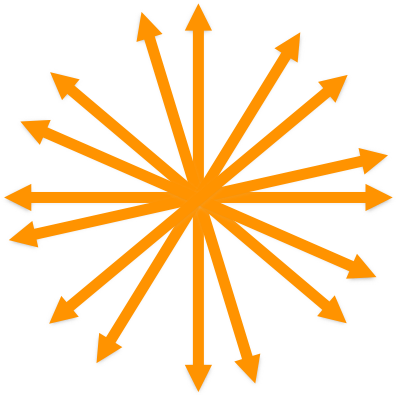
- Cross product of two edges of that triangle
- $n = u \times v$
- $u = p2 - p1$
- $v = p3 - p1$
- $n_x = u_y v_z - u_z v_y$
- $n_y = u_z v_x - u_x v_z$
- $n_z = u_x v_y - u_y v_x$



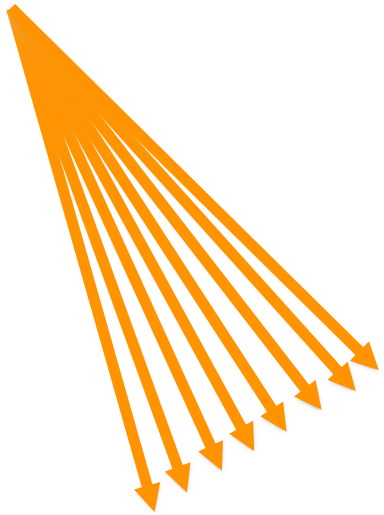
For vertices:

- Average the normal of triangles it contributes too

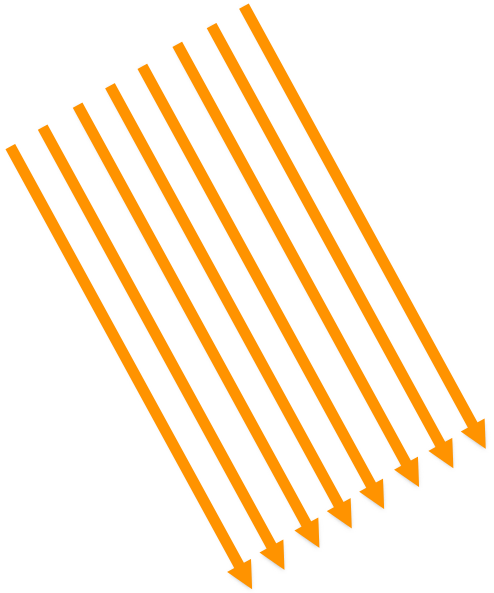
Light Source Types



Point Light

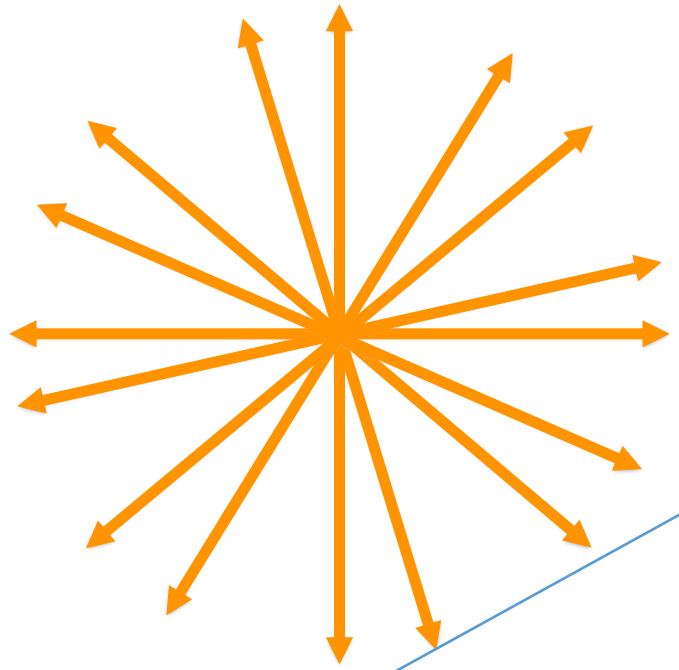


Spot Light



Directional Light

POINT LIGHT



Starts at one point and spreads out in all directions

- Defined by position
- OpenGL light position: (x,y,z,1)
- Can be translated and rotated

Direction is different at each vertex

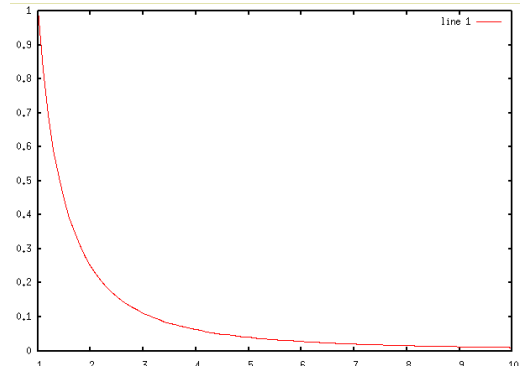
- light direction = light position – vertex position

Example: light bulb

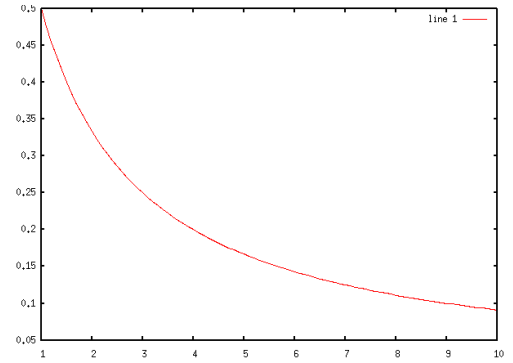
ATTENUATION

A certain amount of flux is transmitted from a light source, the larger distance, the larger area for the photons to hit, leads to less radiance.

- I.e., the further away, the less light hits a spot.



A=1, B=1, C=1



A=0, B=1, C=1

Variables

- d – Distance from light to surface point
- A – Quadric component
- B – Linear component
- C – Constant component

$$Attenuation = \frac{1}{Ad^2 + Bd + C}$$

SPOT LIGHT

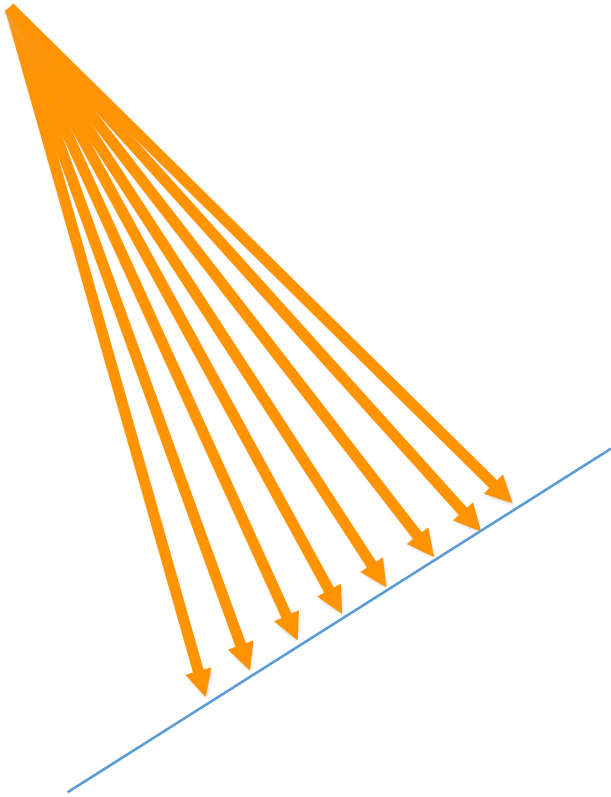
Light starts at one point and spreads out as cone with defined angle

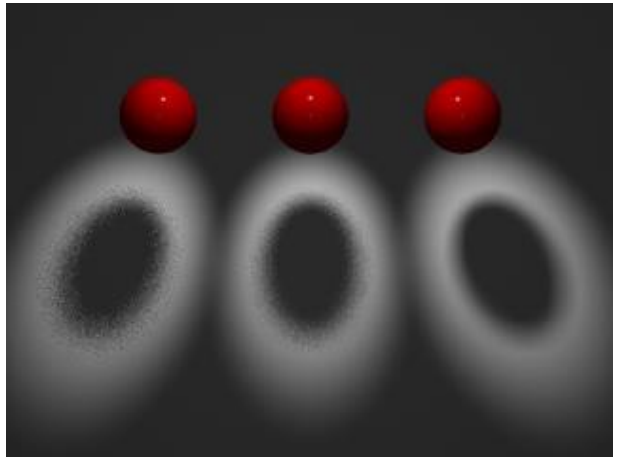
- Similar in principal to point light

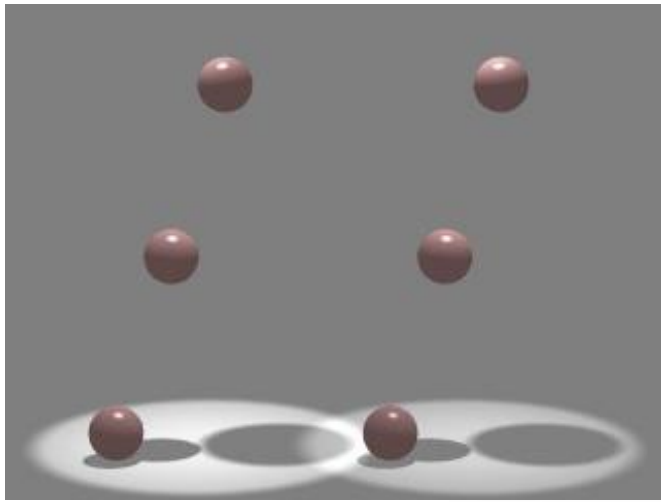
Described by position, direction and width of beam

Useful for dramatic light effects

- e.g. theatre spot light







DIRECTIONAL LIGHT

Described by direction only

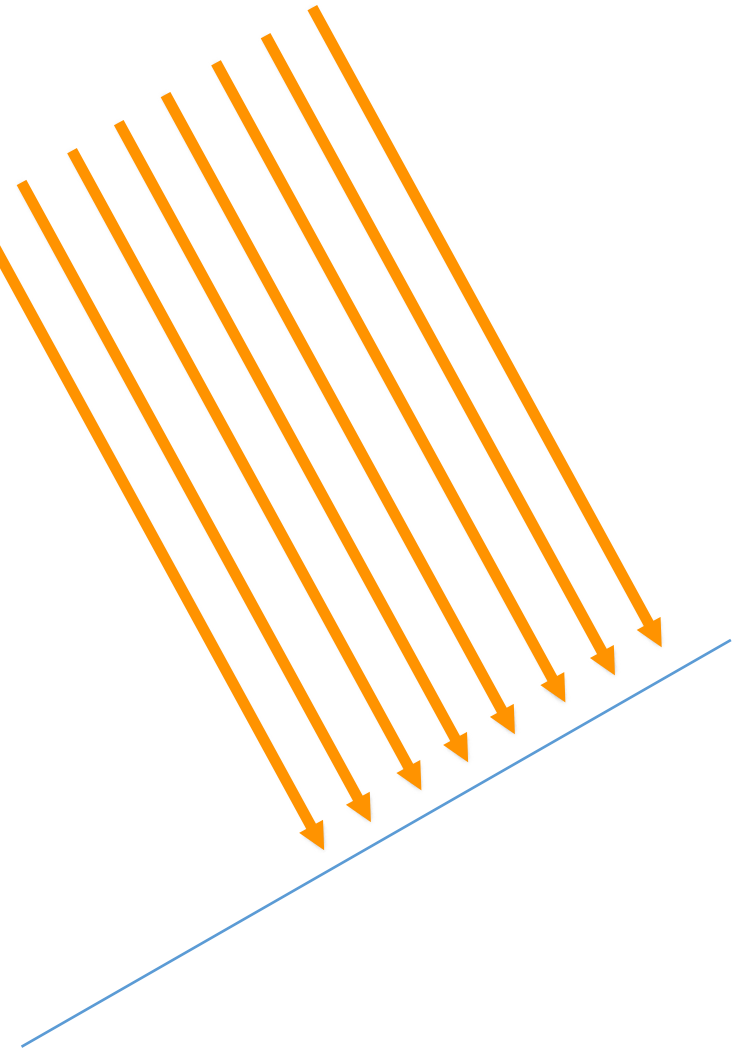
- No position
- Light vector has to be normalized
- OpenGL light position: (x,y,z,0)
- Can only be rotated

Used for light sources that are infinitely far away

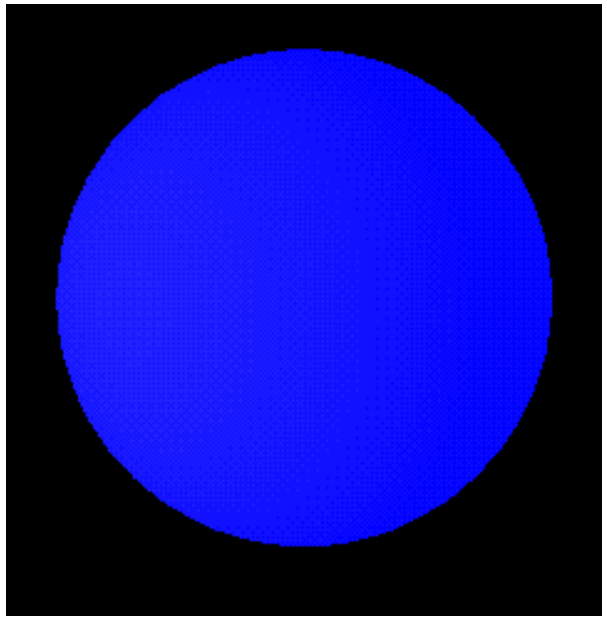
- Direction is same for all points

Intensity does not change depending on distance (no attenuation)

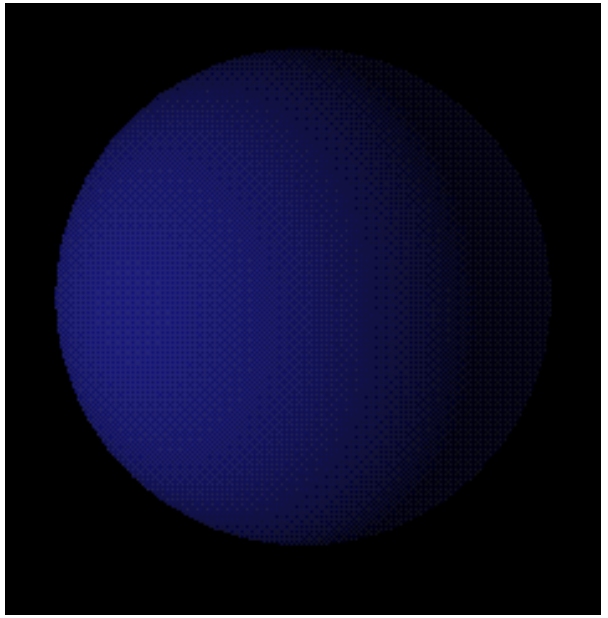
Used for modelling sun light



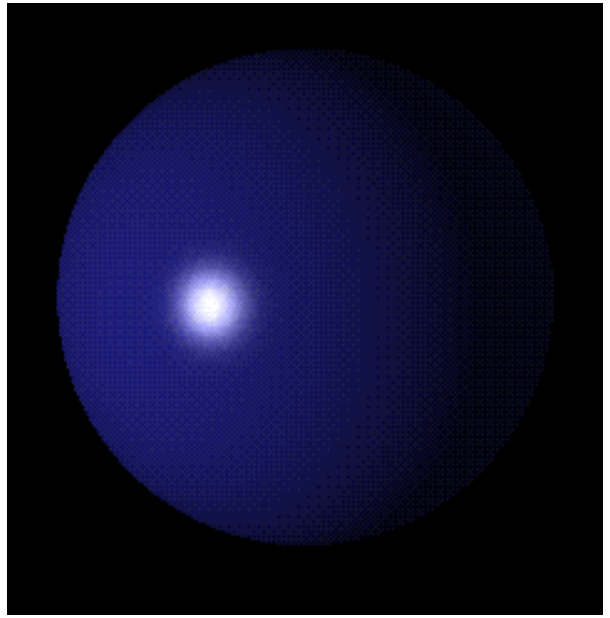
REFLECTION MODEL



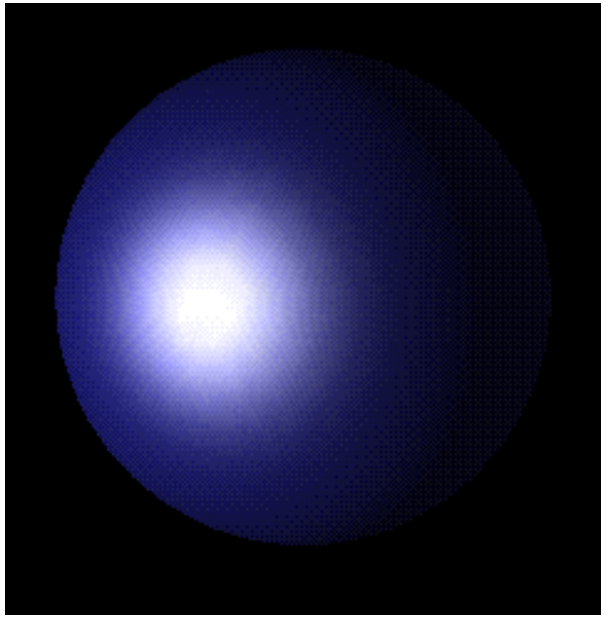
Ambient



Diffuse



Specular

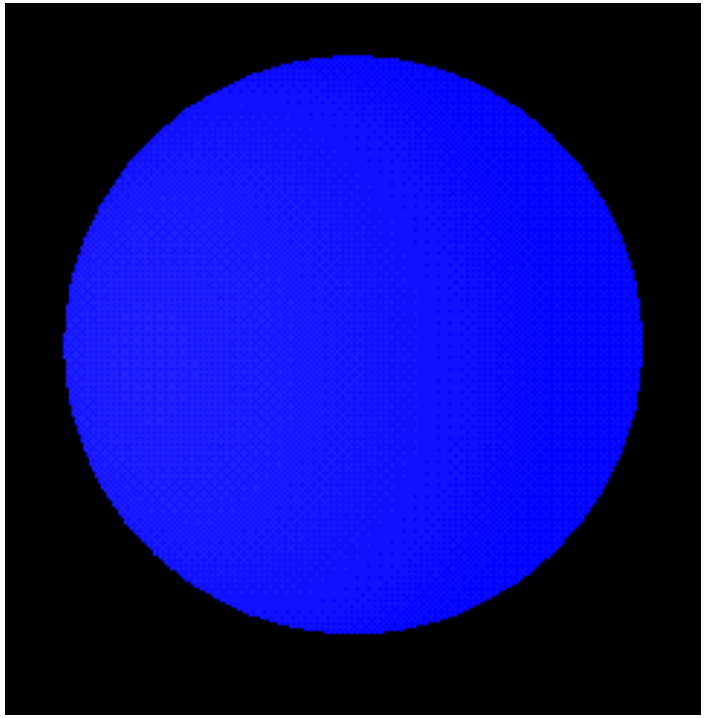


Combined

AMBIENT COMPONENT

Indirect illumination from light that has been reflected multiple times

- Does not come from a specific direction



$$I_{ambient} = I_a \, k_a$$

Variables:

- Ambient light component I_a
- Ambient material factor k_a

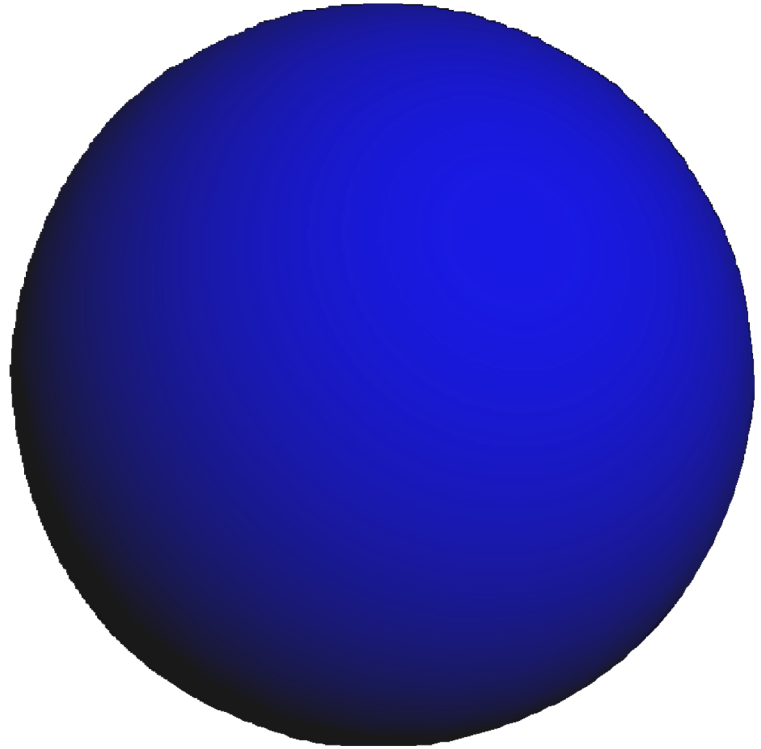
DIFFUSE COMPONENT

Also called Lambertian reflection

Ideal diffuse surface reflects light equally in all directions

Incident ray is reflected in many directions

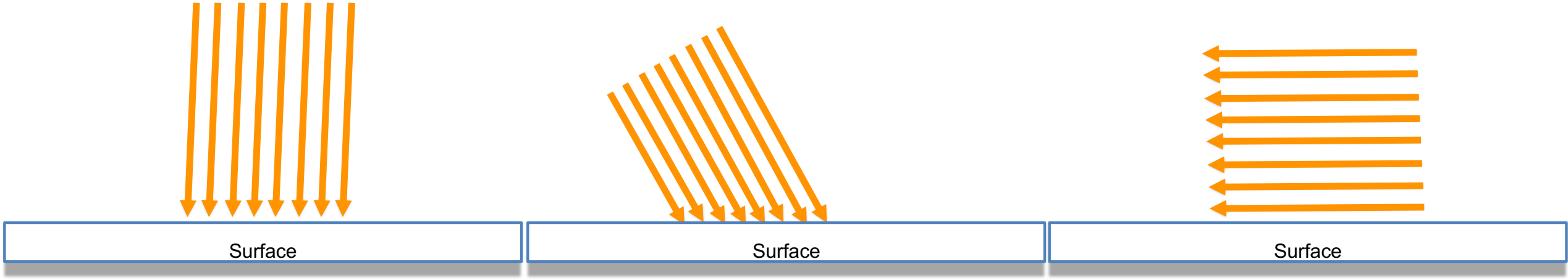
- Independent of view angle (reflects equally in all directions)
- But dependent on direction of incoming light (angle between normal N and incident light L : angle of incidence θ)



DIFFUSE COMPONENT

Incoming light rays with perpendicular angle to the surface reflect more light

The larger the angle θ between normal and incoming light rays, the less light is reflected



θ = 0°: Maximum Brightness

θ = 45°:

θ = 90°: Dark

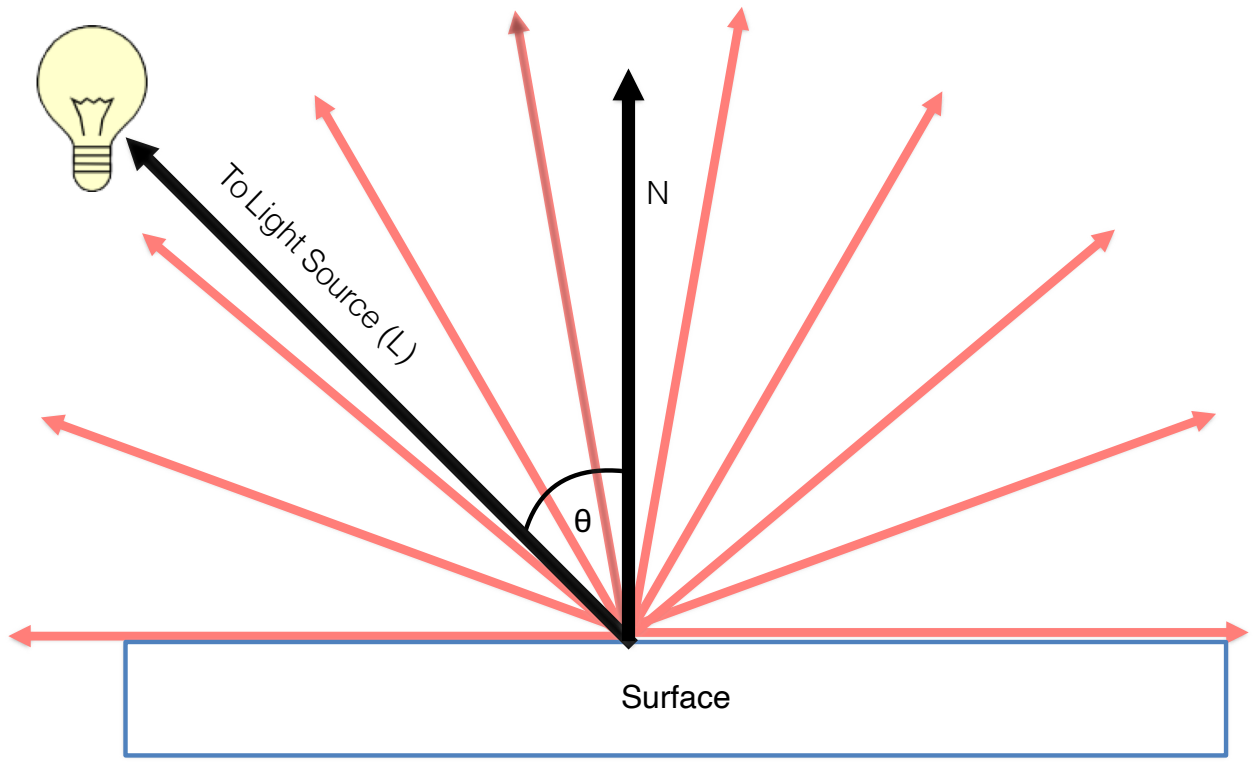
DIFFUSE COMPONENT

Variables:

- Diffuse light component I_d
- Diffuse material factor k_d
- Light direction l
- Surface normal n

Careful about:

- Vector directions
- Negative dot products

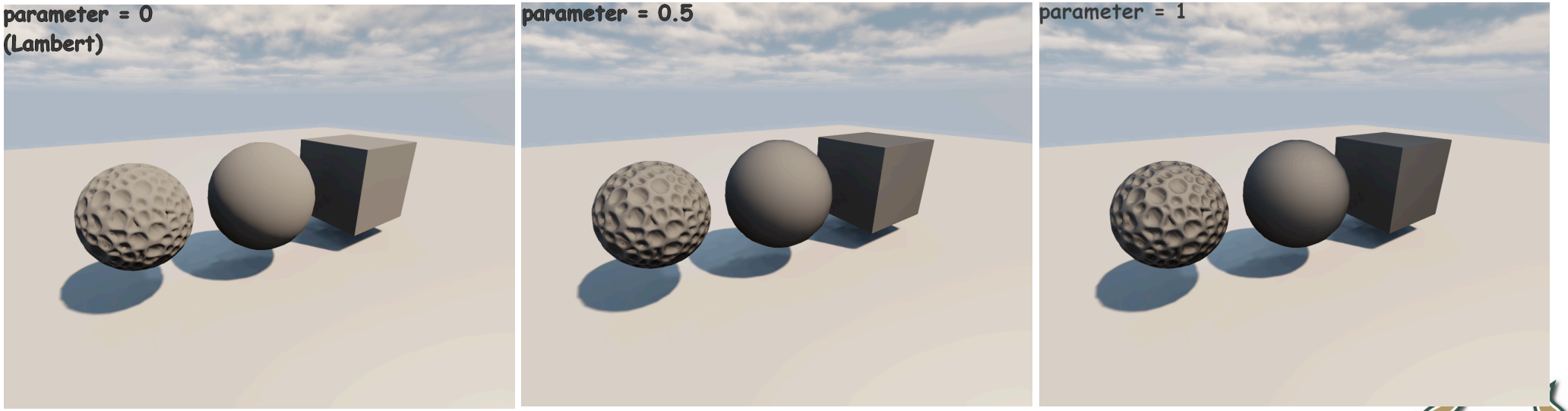


$$I_{diffuse} = I_d k_d \cos\theta = I_d k_d \frac{N \cdot L}{|N||L|}$$

Other Diffuse Reflection Models

Oren-Nayar (rough surfaces)

Minnaert (Velvet-like shading)



https://en.wikipedia.org/wiki/Oren%E2%80%93Nayar_reflectance_model

http://oliverm-h.blogspot.com/2012/02/

SPECULAR COMPONENT

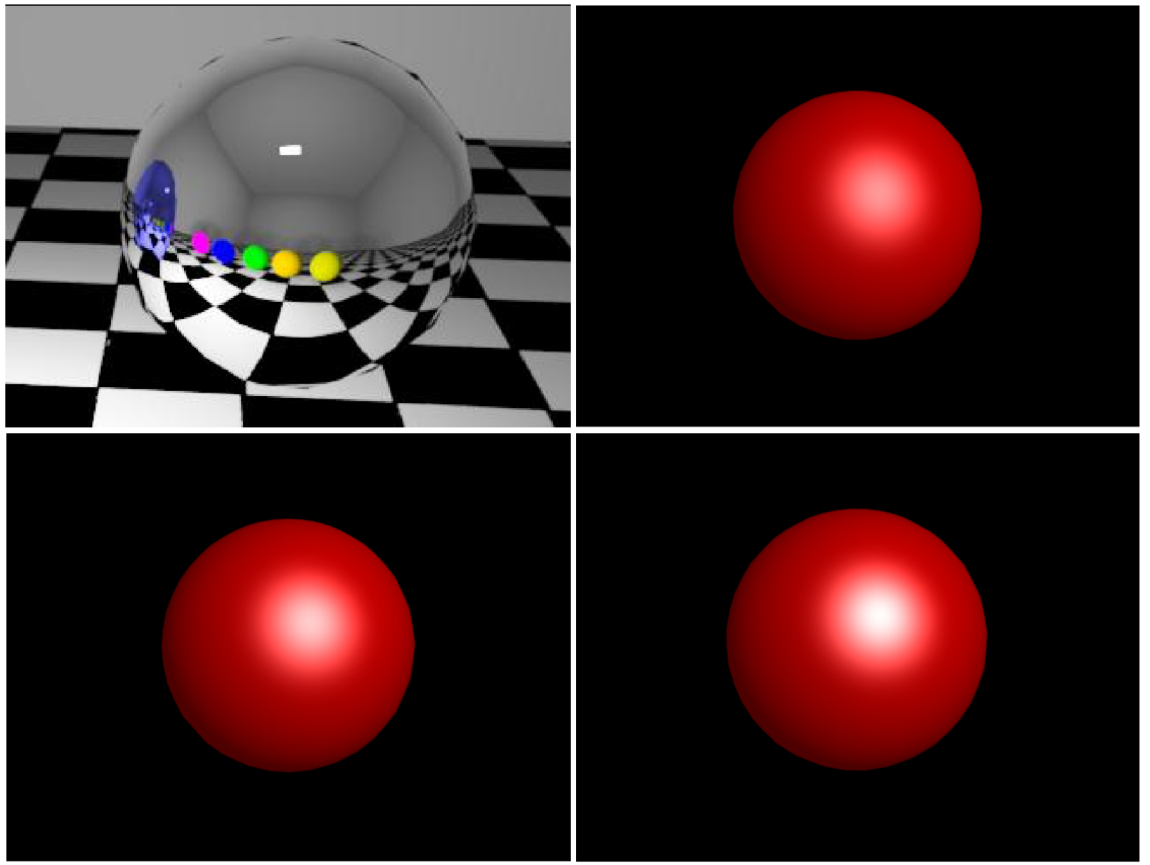
Simulates highlights from shiny objects—called *specular highlight*

For ideal reflectors

- angle of incidence equals angle of reflection (only visible of R equals V)

For non-perfect reflectors

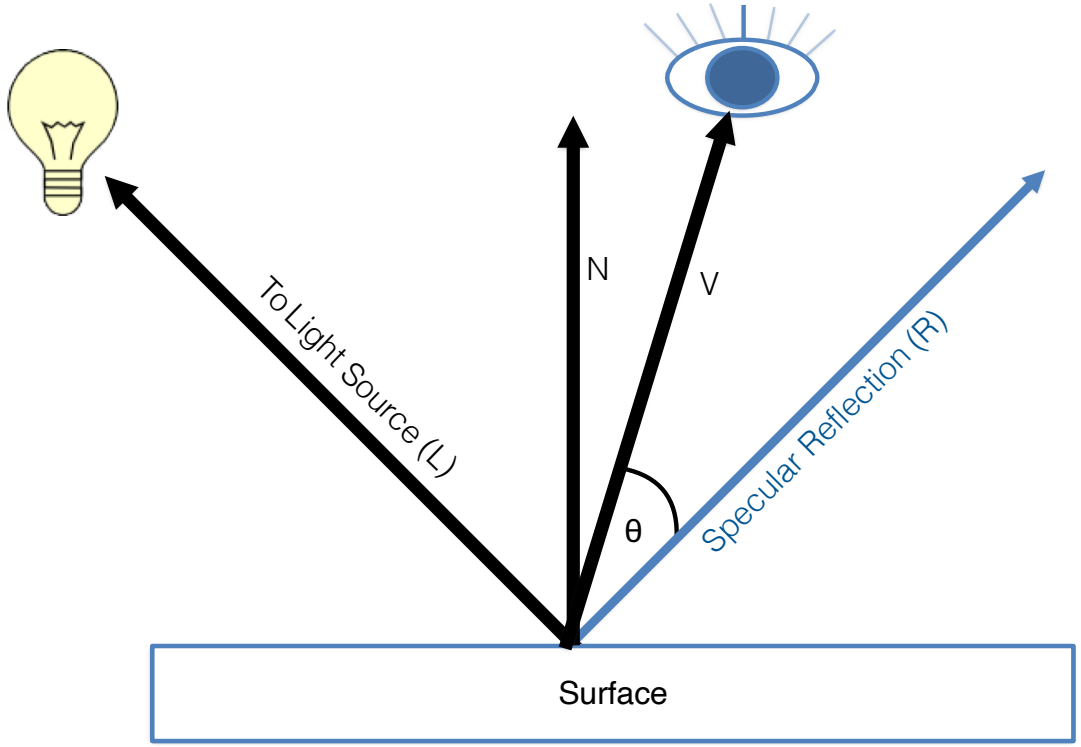
- highlight is visible for a range of angles



Phong Specular Component

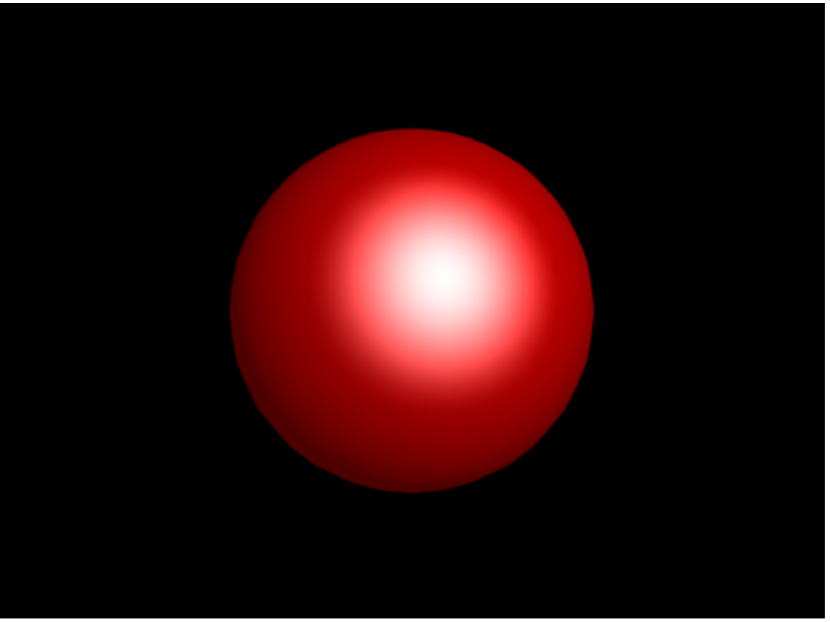
Consists of:

- Direction to light source
- Reflected ray
- $R = 2(N \cdot L)\, N - L$
  - (in GLSL use reflect method)
- Specular material factor k_s
- Specular exponent n_s (the larger, the smaller the highlight)
- k_s and n_s have no physical meaning
  - a lot of tweaking required to achieve desired result

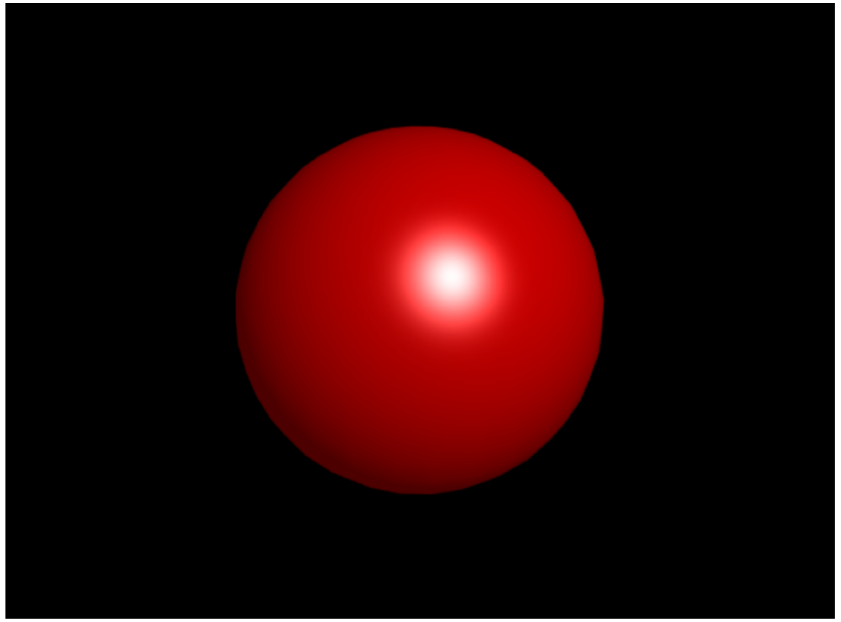


$$I_{specular} = I_s k_s (R \cdot V)^{n_s}$$

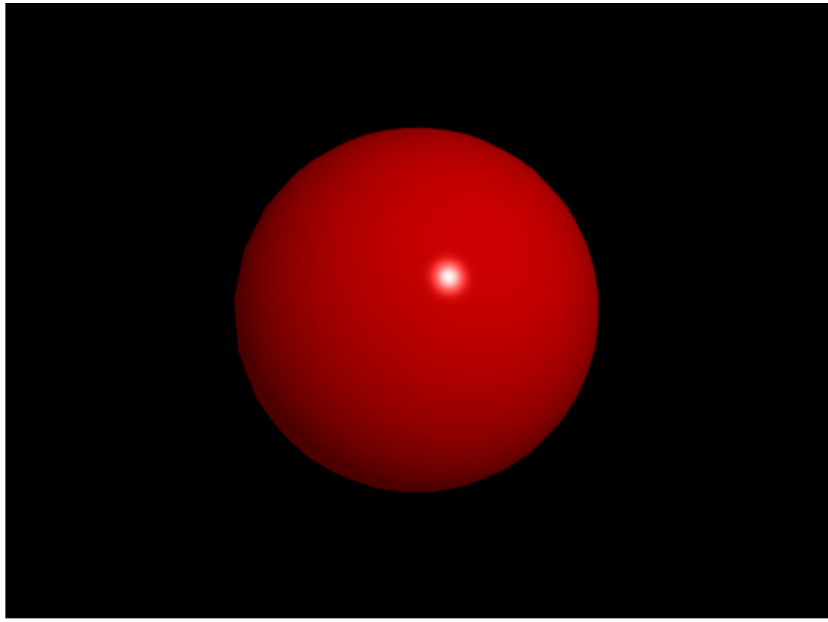
SPECULAR COMPONENT



$n_s = 2.0$

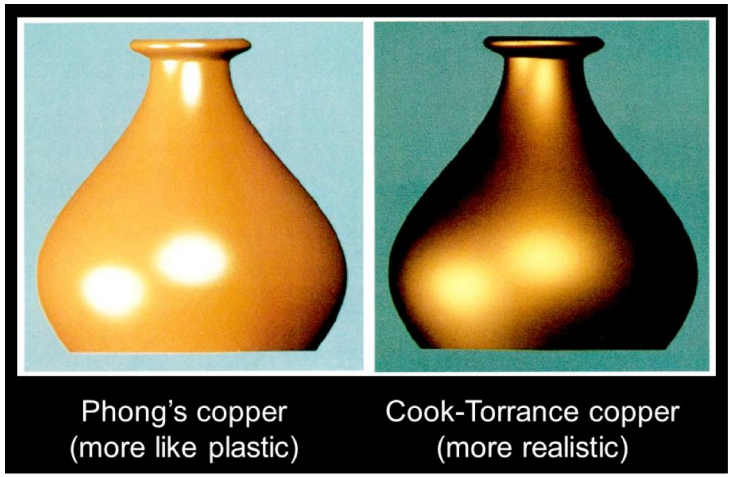


$n_s = 10.0$

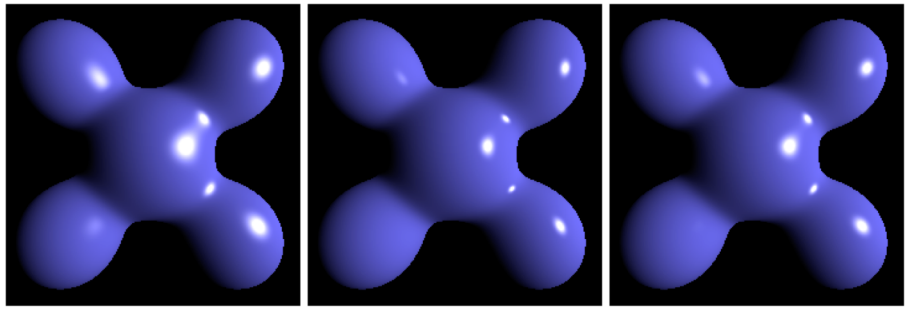


$n_s = 100.0$

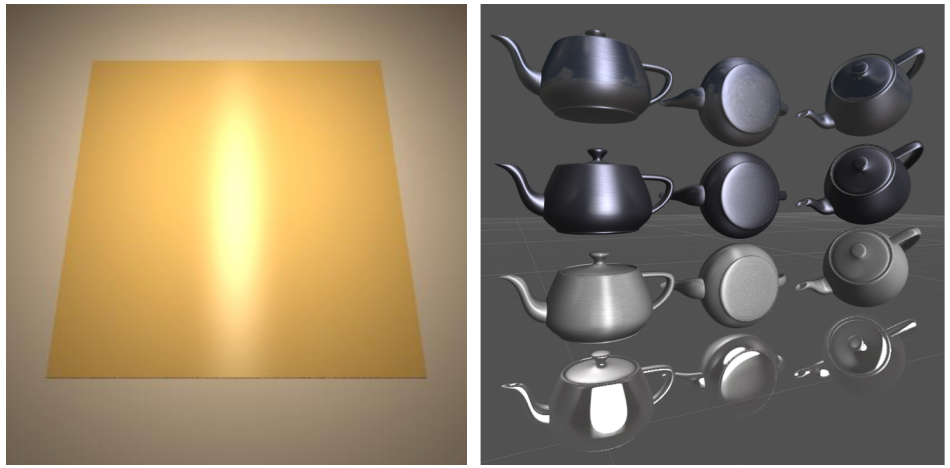
Other Specular Models



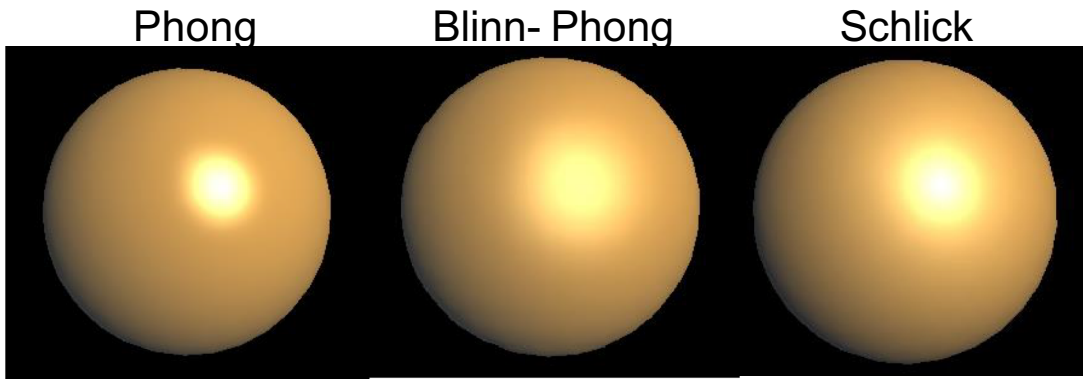
Blinn-Phong



Cook-Torrance



Ward anisotropic



Schlick's approximation

https://en.wikipedia.org/wiki/Blinn%E2%80%93Phong_reflection_model
https://slideplayer.com/slide/9989581/
https://gamedev.stackexchange.com/questions/67679/how-can-i-create-a-shader-that-will-reproduce-this-lighting-effect-on-terrain

Special Note on Mirror Reflections

To calculate real reflection you need global illumination (e.g., ray tracing)

Planar mirrors can be calculated efficiently

- At the cost of an additional rendering pass under a linear transformation

Non-planar mirror, really depends

- Hacks available—see https://ieeexplore.ieee.org/abstract/document/5733331

Similar issues with other effects like caustics

Phong Reflectance Model


=

+

+


$$Illumination = \quad I_{ambient} = I_a\, k_a \qquad I_{diffuse} = I_d k_d \frac{N \cdot L}{|N||L|} \qquad I_{specular} = I_s k_s (R \cdot V)^{n_s}$$

$$Illumination = I_a\, k_a + I_d k_d \frac{N \cdot L}{|N||L|} + I_s k_s (R \cdot V)^{n_s}$$

MATERIAL DEFINITIONS

Material Template Library (MTL) can be used to define material settings

Defines ambient (Ka), diffuse (Kd), specular (Ks) colors and the specular exponent (Ns)

Also allows to define opacity (d)

- 1.0 means fully opaque

Pick illumination model

Set texture maps (map_Kd)

```
newmtl EarthMaterial
Ka     0.640000  0.640000  0.640000
Kd     0.640000  0.640000  0.640000
Ks     0.050000  0.050000  0.050000
Ns     30.0000
d      0.5
illum  2
map_Kd ColorMap.bmp
```

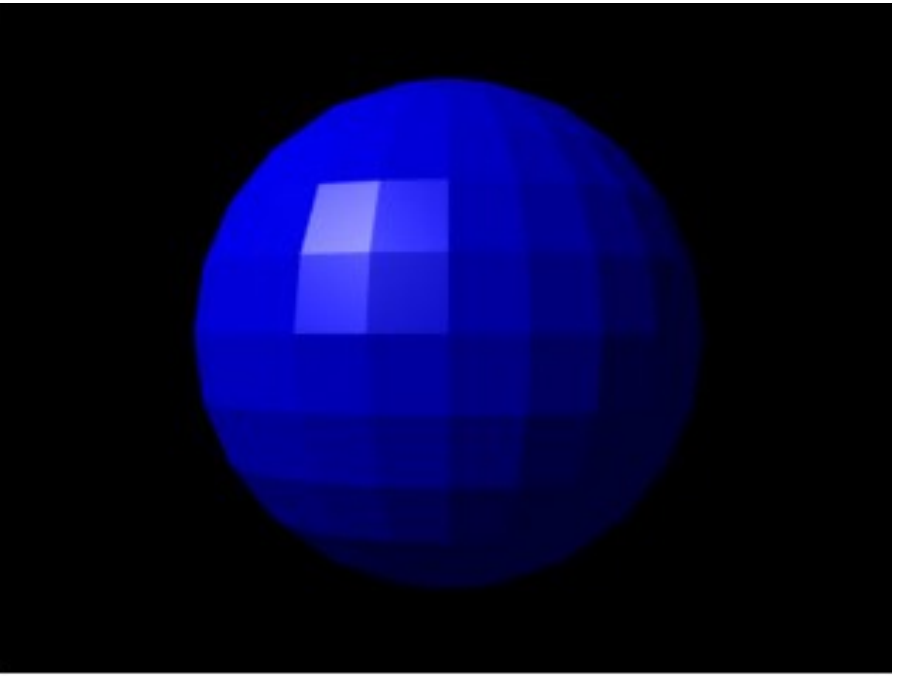
Example

Bidirectional Reflectance Distribution Function (BRDF)

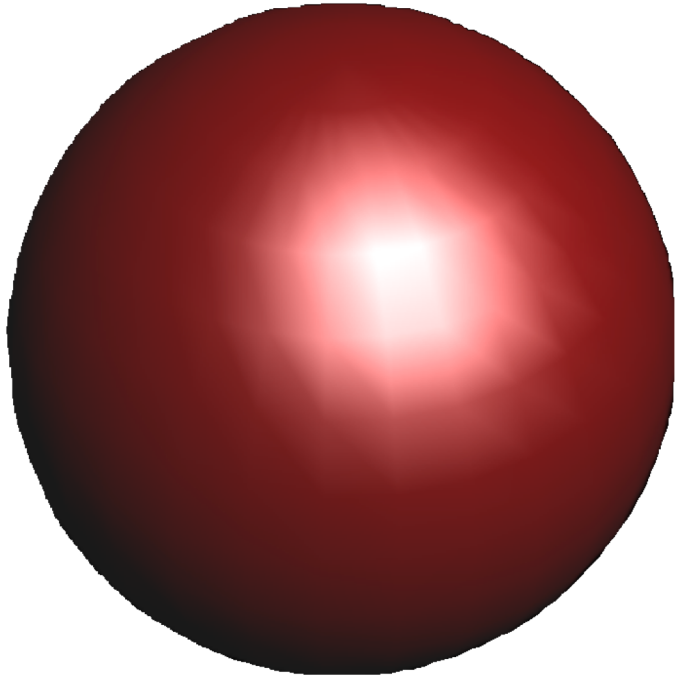
BRDF models how much energy is reflected by light hitting a point on a surface with

- Incoming angles (θi Φi)
- Outgoing angles (θo Φo)
- Also models how light of different wavelength are absorbed and reflected

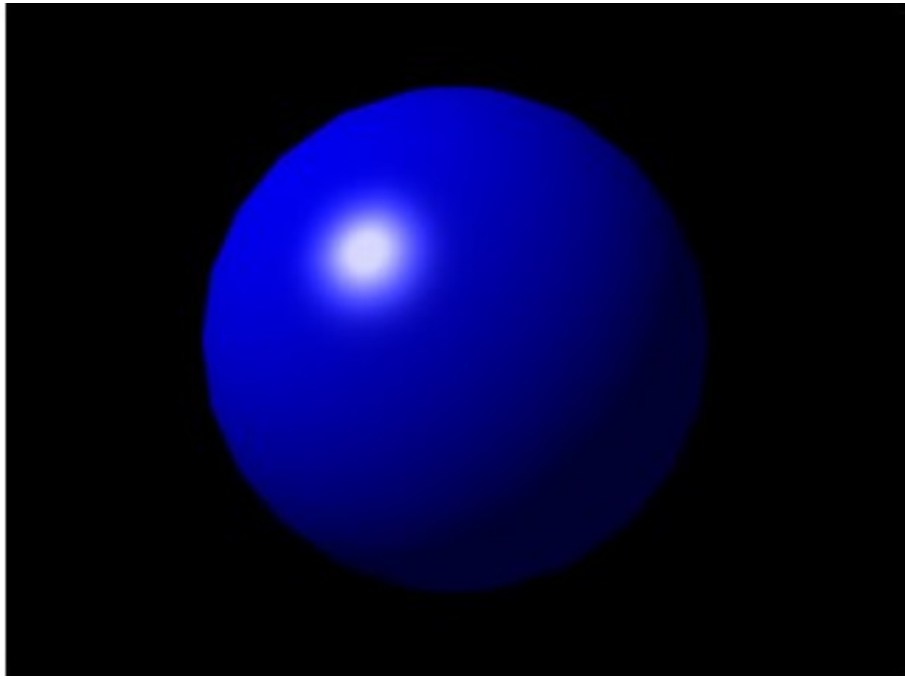
Interpolation/Shading Models



Flat Shading



Gouraud Shading



Phong Shading

FLAT SHADING

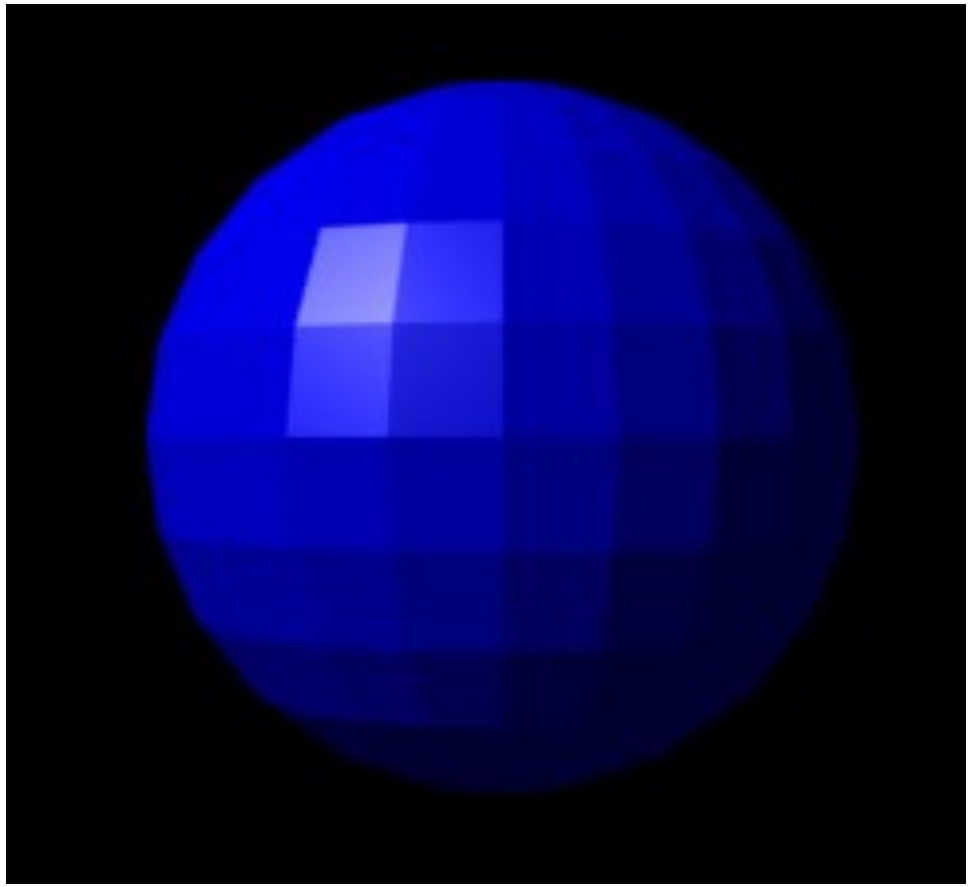
Per polygon lighting

Was used for high speed rendering

All pixels of one polygons have the same color

Difference between polygons

No smooth transitions



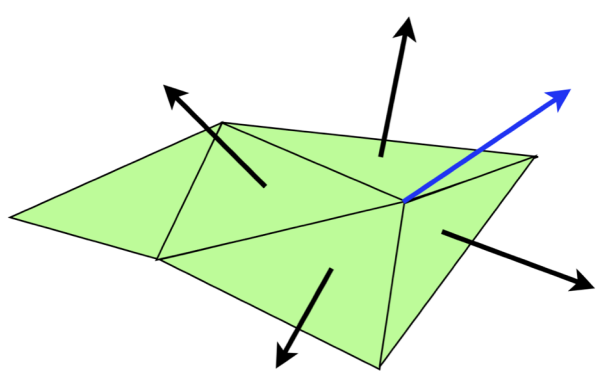
GOURAUD SHADING

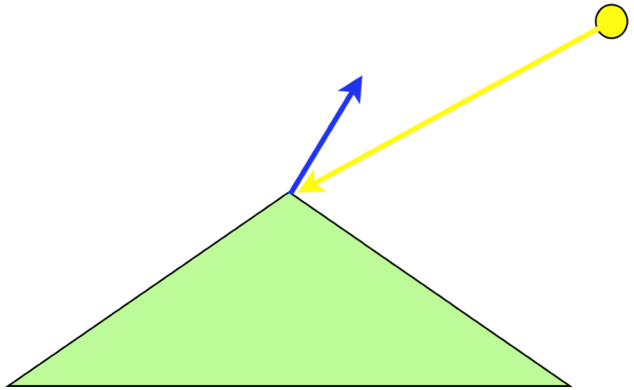
Calculate normal per vertex

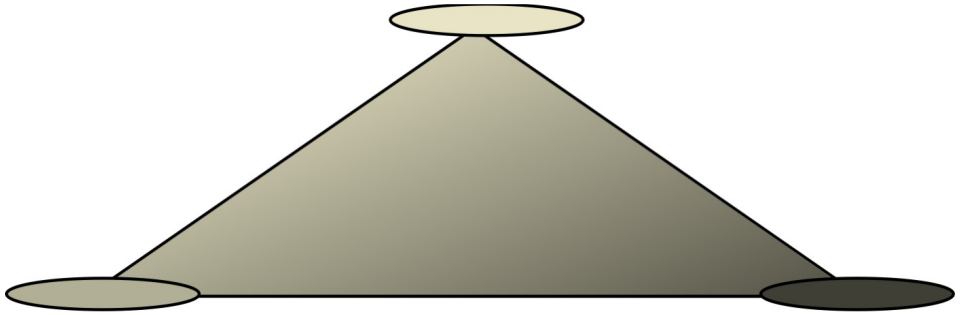
- Average of face normals

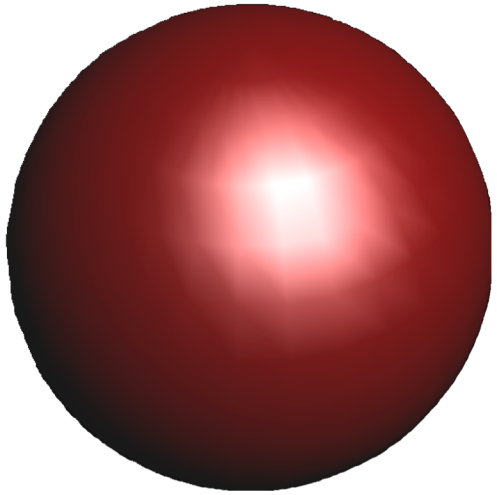
Calculate lighting per vertex

Interpolate per interior pixel









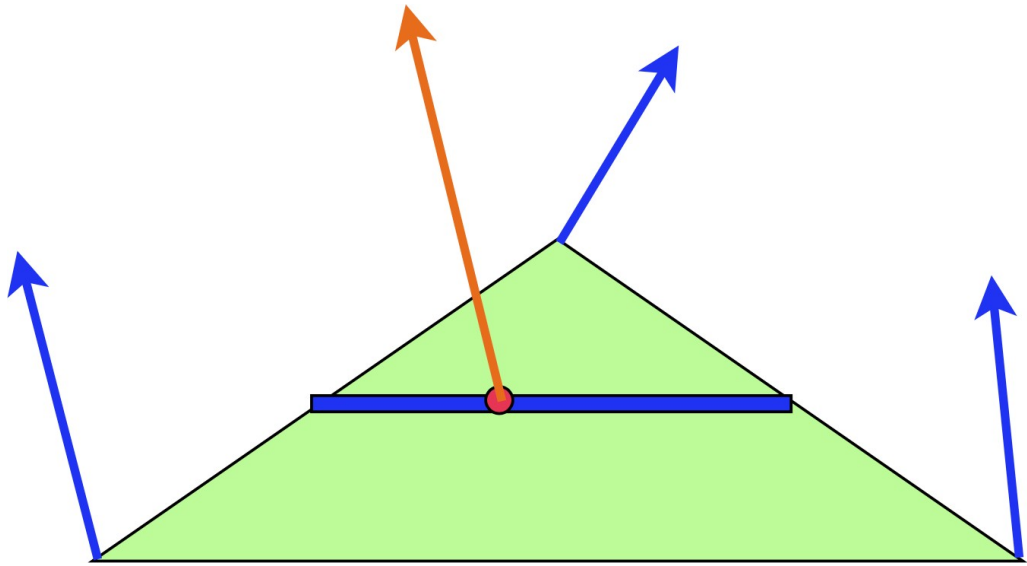
PHONG SHADING

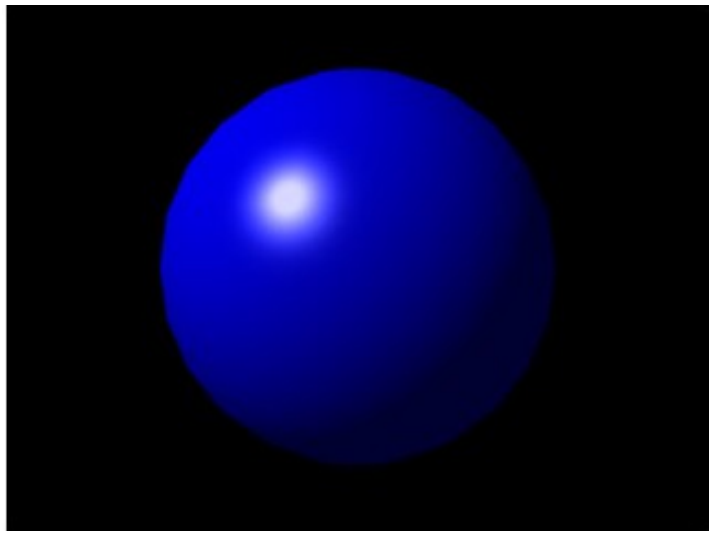
This is **NOT** Phong illumination

Per fragment/pixel

- Interpolate the surface normals
- Then do calculation lighting calculation using the interpolated normal

Gives better results, especially for highlights





AMBIENT OCCLUSION

Not all objects receive direct light, but ambient light lack direction

Without shadows objects appear flat

- Problem for self-shadowing, scenes without direct lighting (e.g., forest scenes)

Shadows from ambient light is called Ambient Occlusion

- It does not depend on light direction so it can be precomputed for static objects
- But if objects are animated/deformed we need **Dynamic Ambient Occlusion**

Dynamic Ambient Occlusion

Object-space methods (expensive & pre-computed)

Screen spaced methods – SSAO (Screen-Space-Ambient-Occlusion)

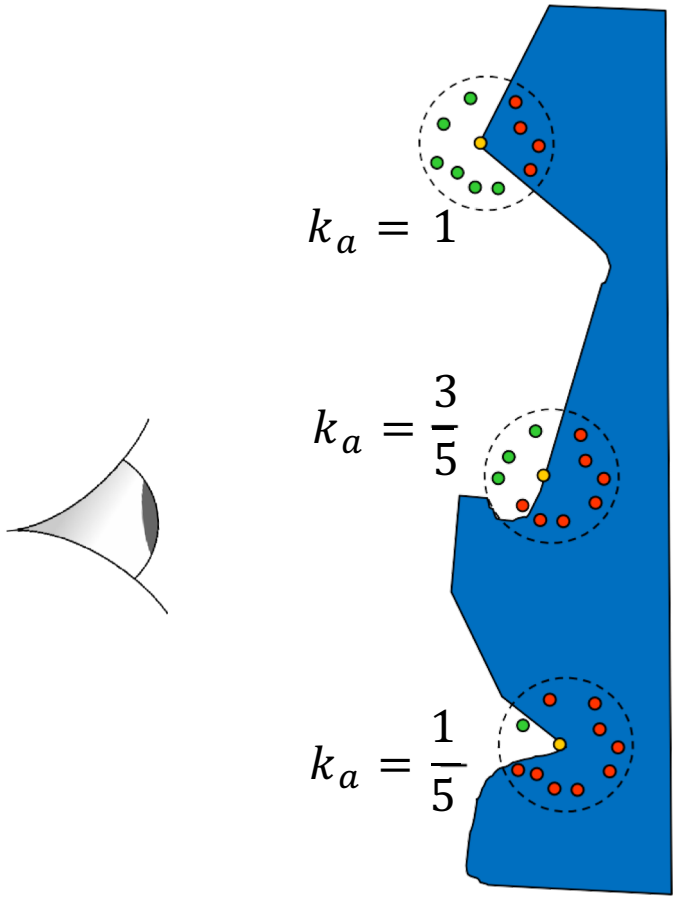
SSAO

Use the z-buffer (render scene to depth map)

- Sample depth map in a sphere around current rendering point
- For each rendered dot, we generate (on a sphere) a number of sample points (e.g. 10)
- We are interested in the points on the surrounding hemisphere of the object.
  - Approximately ½ of the sampled points are invalid, because they are not in the hemisphere above the surface (but we do not know which).

- Yellow dots are points that are to be rendered
- Green dots are visible from the camera
- Red dots are obscured by the geometry/scene

- 5 or more dots are visible (green) we have $k_a = 1$
- 4 or less we reduce k_a with the corresponding factor



SSAO Example

https://youtu.be/IfdAILHTcZk

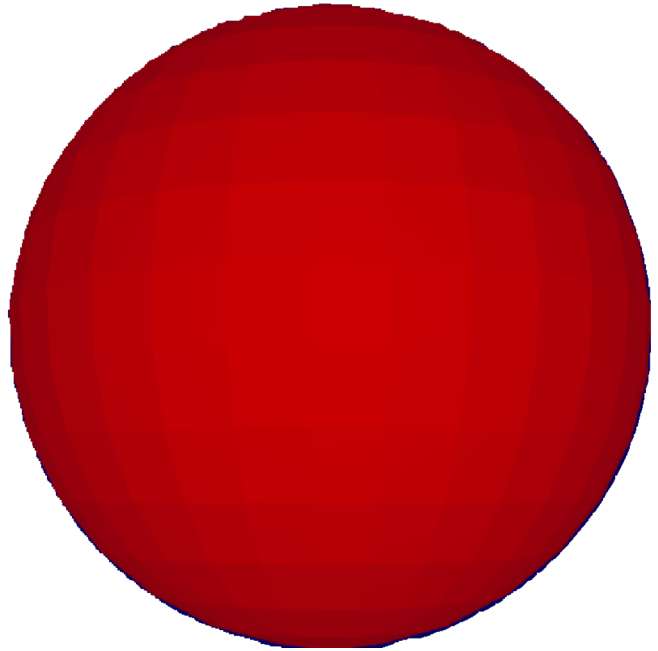
WHY TEXTURING?

Detail is expensive to model

- Often the surface of an object is viewed only from a distance

Enhance visual appearance of plain surfaces and objects by applying texture details

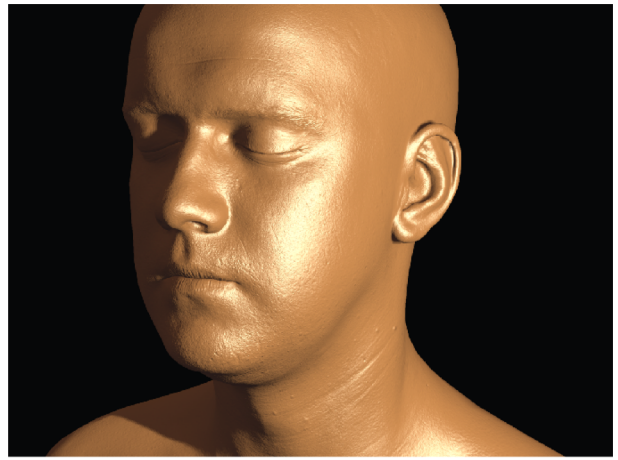
- We can ‘paint’ the detail on the object instead of increasing the complexity of the object

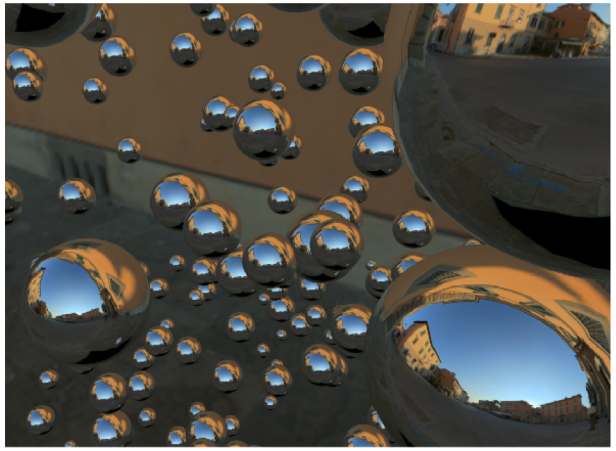


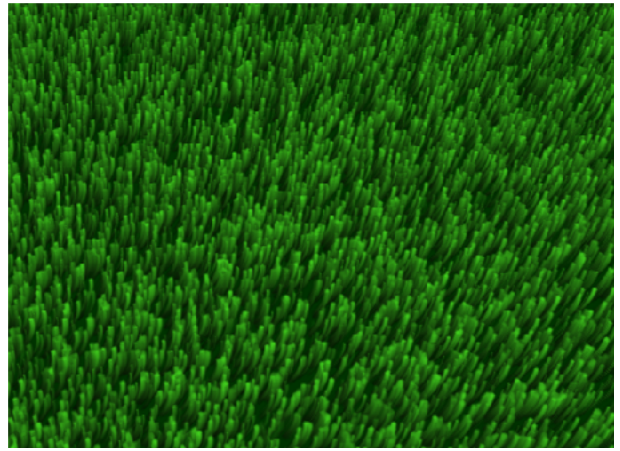
TEXTURING

Simulation of different material properties:

- Color
- Reflection
- Gloss
- Transparency
- Bumpiness







Texture Map Basics

Usually a 2D image (can be 1D or 3D also)

- Texels (texture elements) hold data

Can hold arbitrary information (e.g., color, depth, alpha, normals)

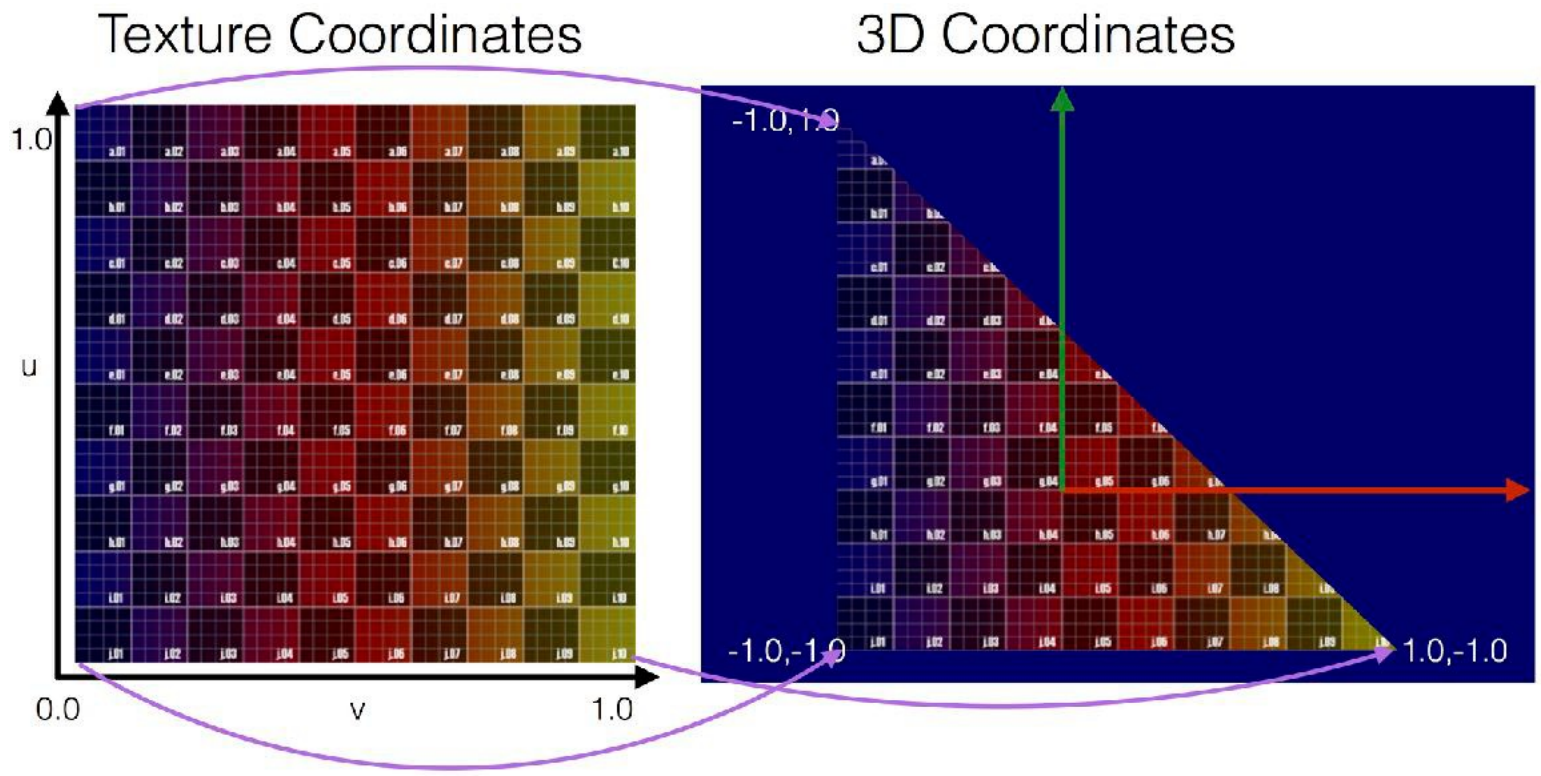
- Meaning interpreted by shader
- Basis for most advanced rendering effects

We a coordinate system on the surface to find the right part of the texture

- (u, v) coordinate, where u and v ranging from 0.0 to 1.0

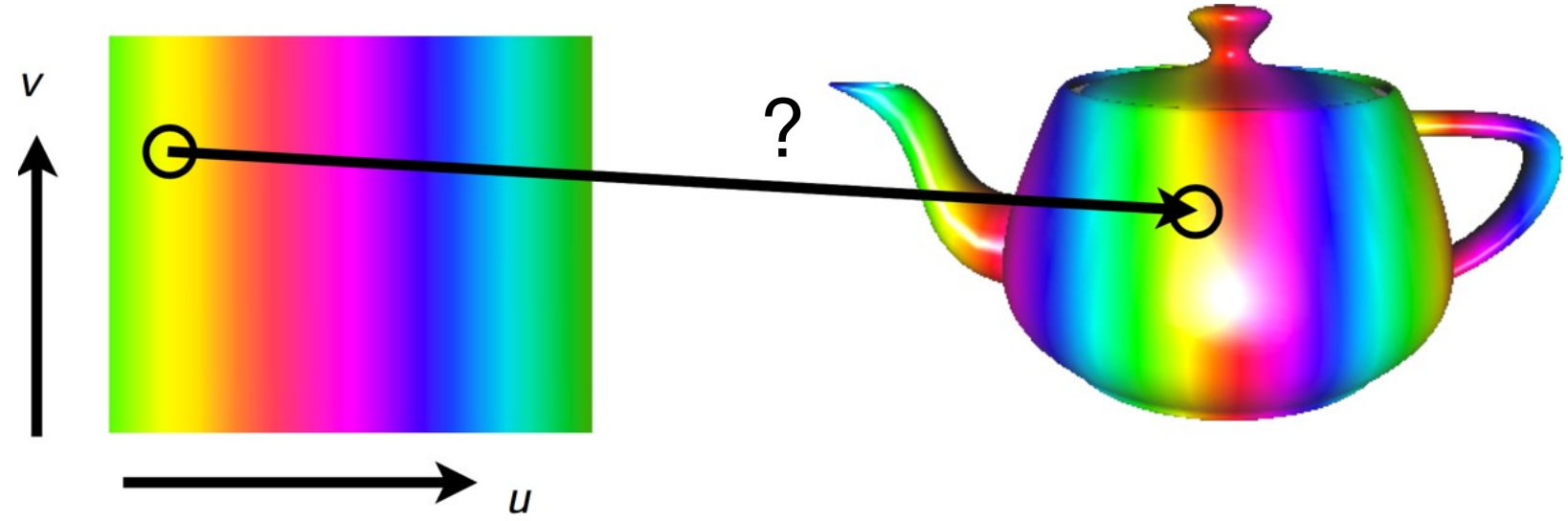
TEXTURE MAPPING

Process of finding u,v coordinates for each vertex

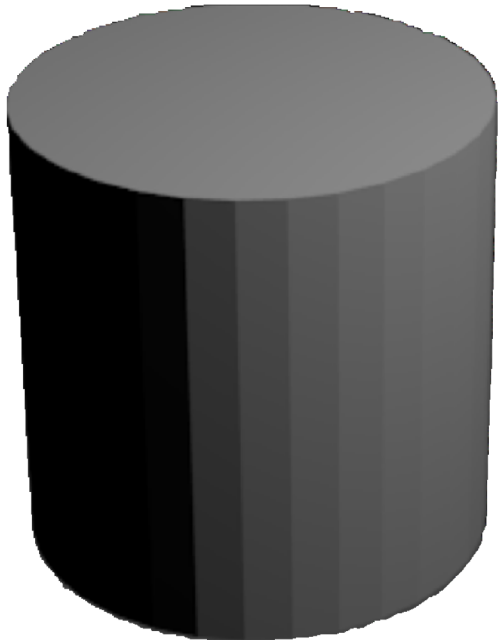


TEXTURE MAPPING

Often objects are not that simple



PARAMETERIZATION



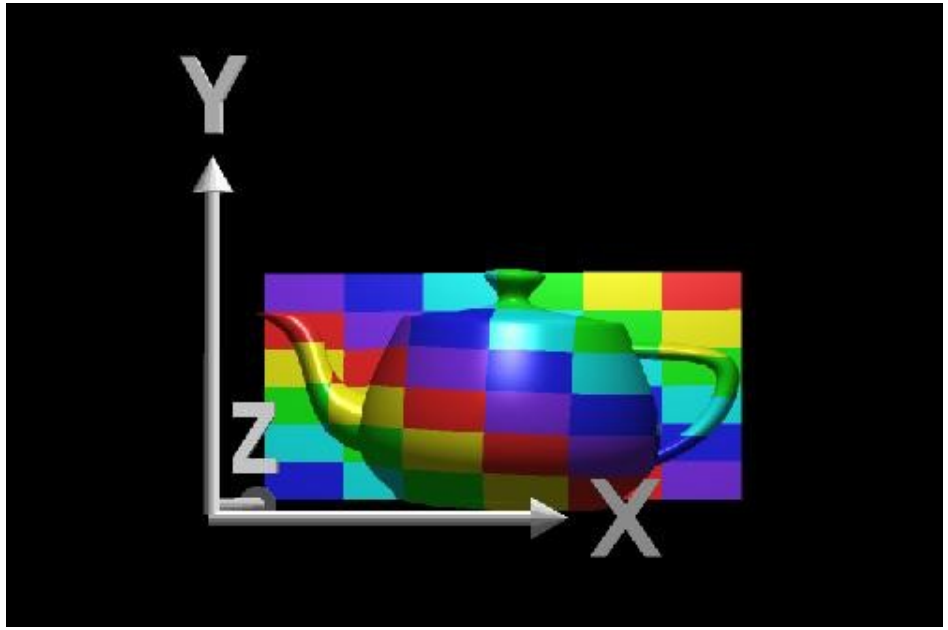
+

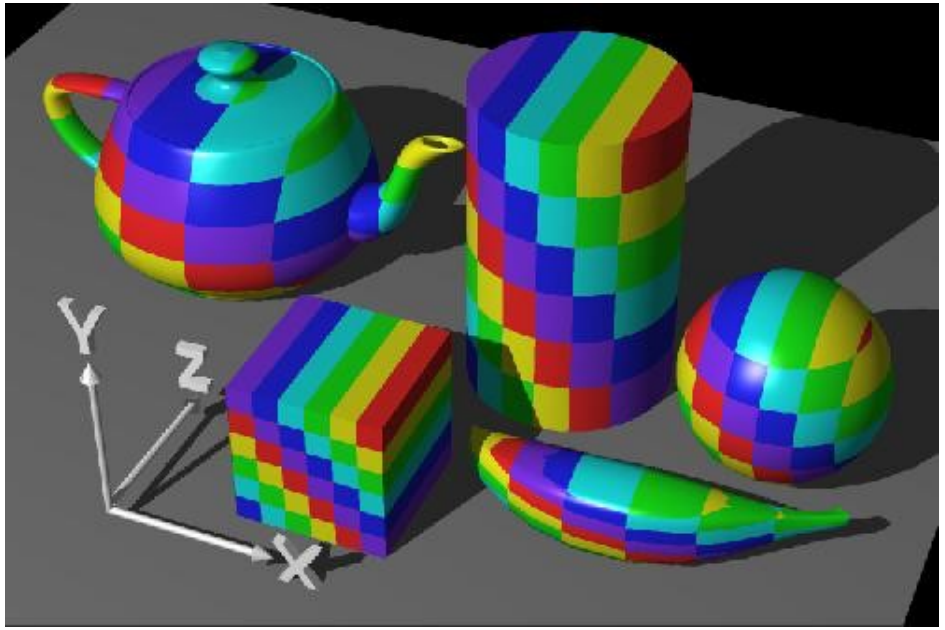
=

PARAMETRIZATION—PLANAR MAPPING

Eliminates one axis (z) and use x and y directly: (u,v)=(x,y)

Looks only good from front



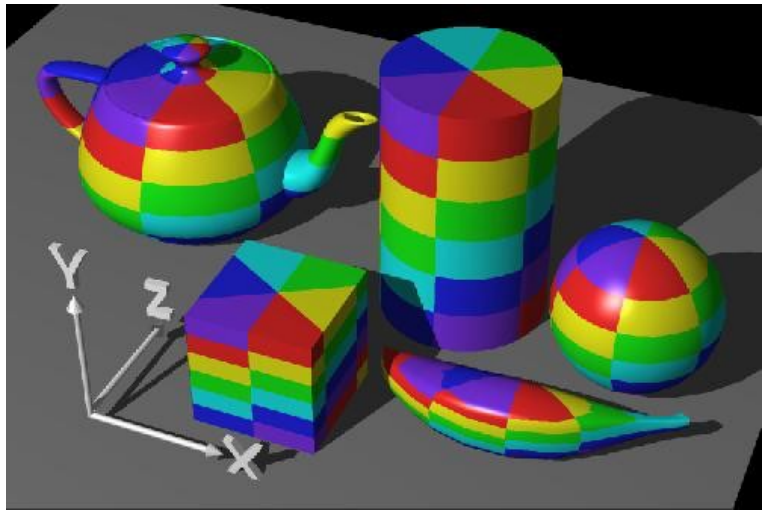


PARAMETRIZATION—CYLINDRICAL MAPPING

Compute angles between vertex and object center

Converts point P(x,y,z) to cylindrical coordinates P(r, θ,z)

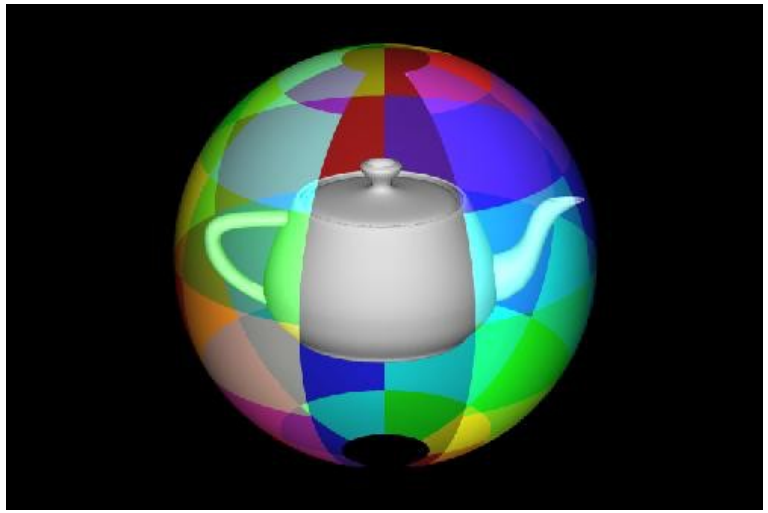
Wraps texture around the object

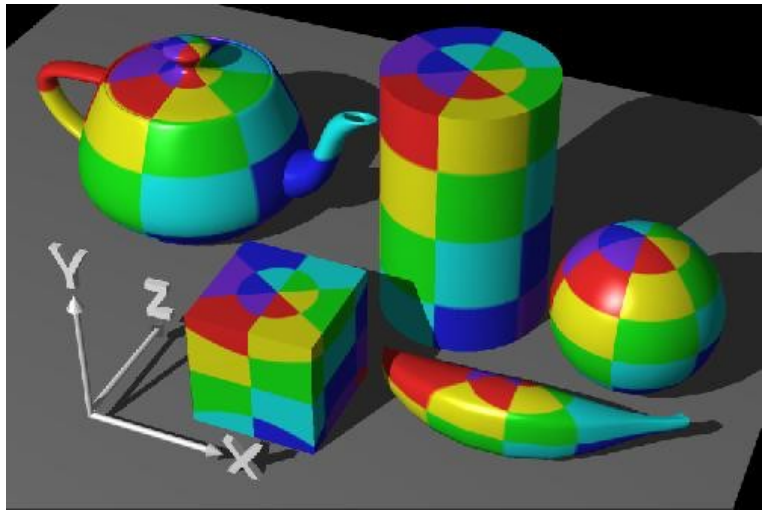


PARAMETRIZATION—SPHERICAL MAPPING

p(x,y,z) value of a point is converted into spherical coordinates (theta, phi)

Wraps texture around the object

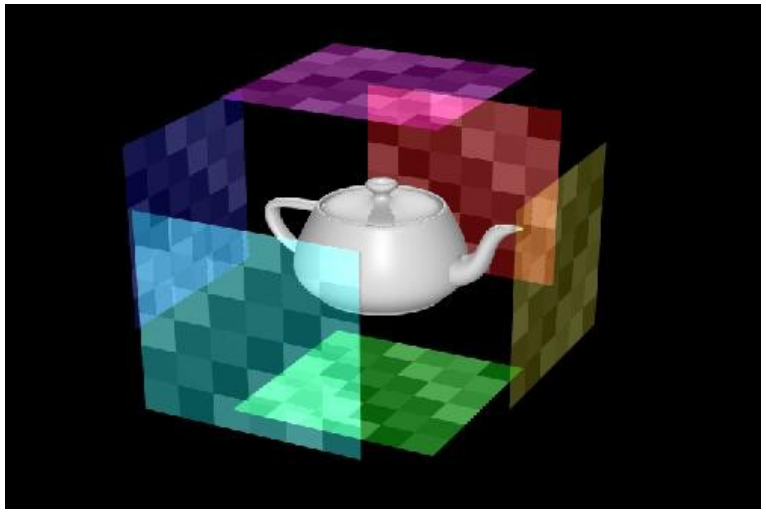


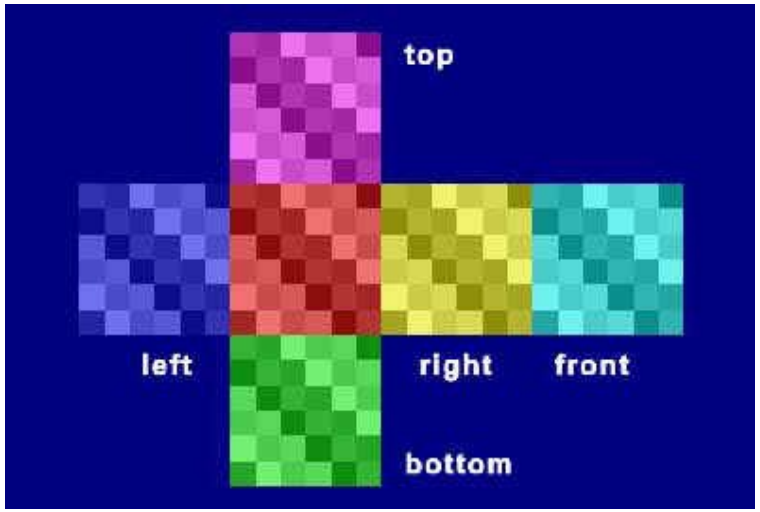


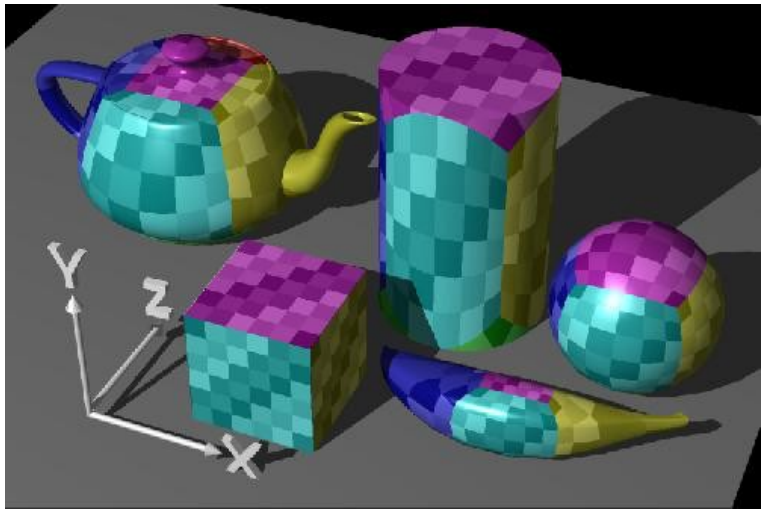
PARAMETRIZATION—BOX MAPPING

Similar to planar mapping

But uses 6 textures instead of 1

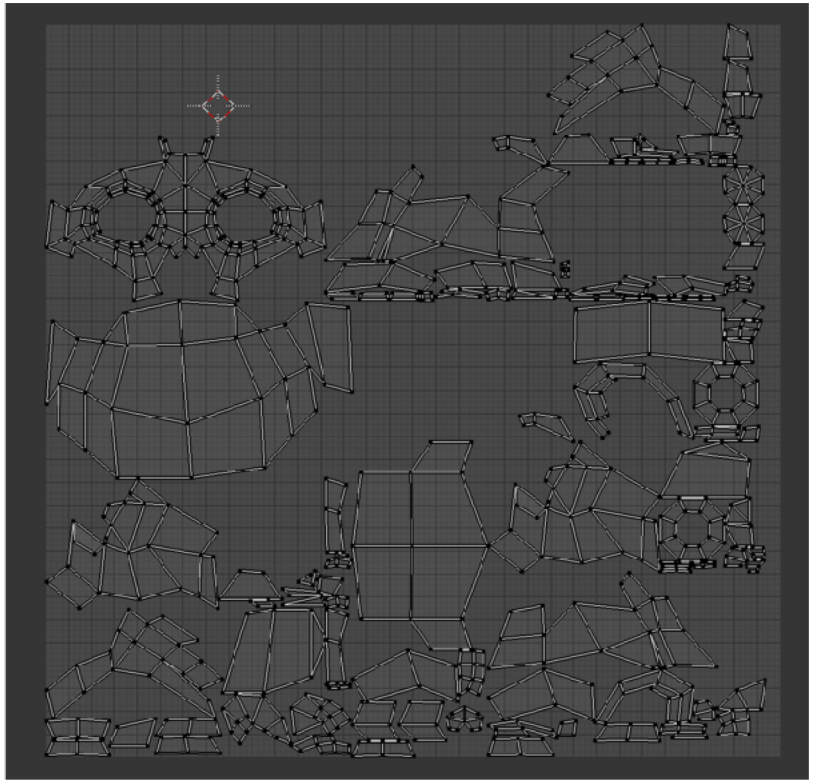


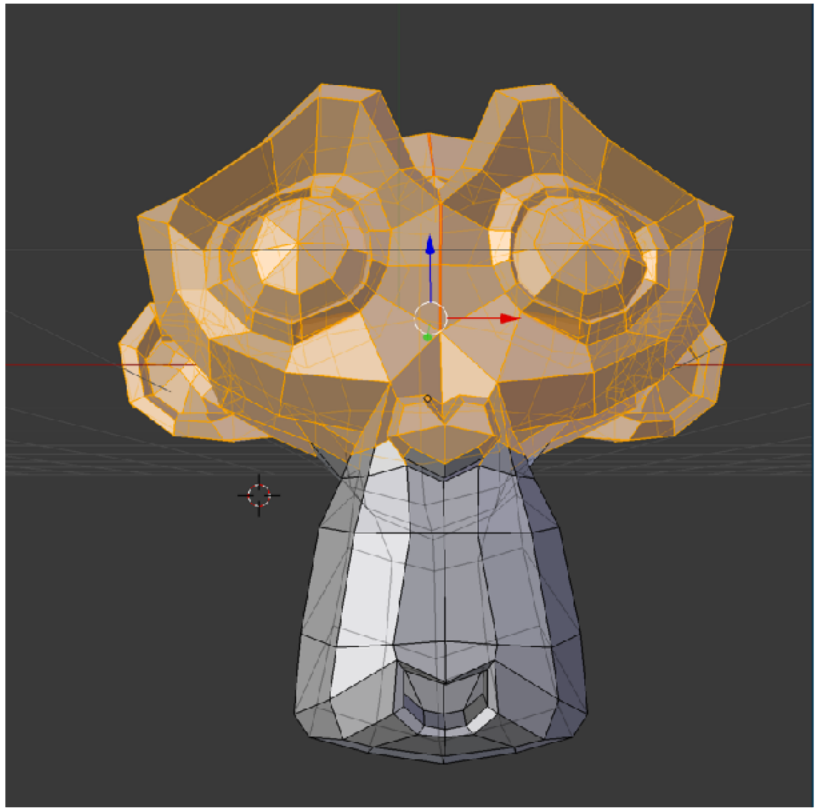




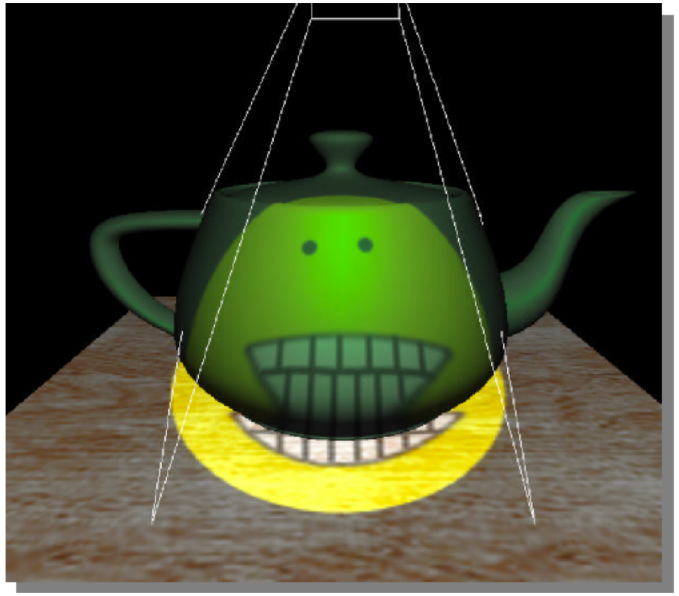
PARAMETRISATION—MANUAL MAPPING

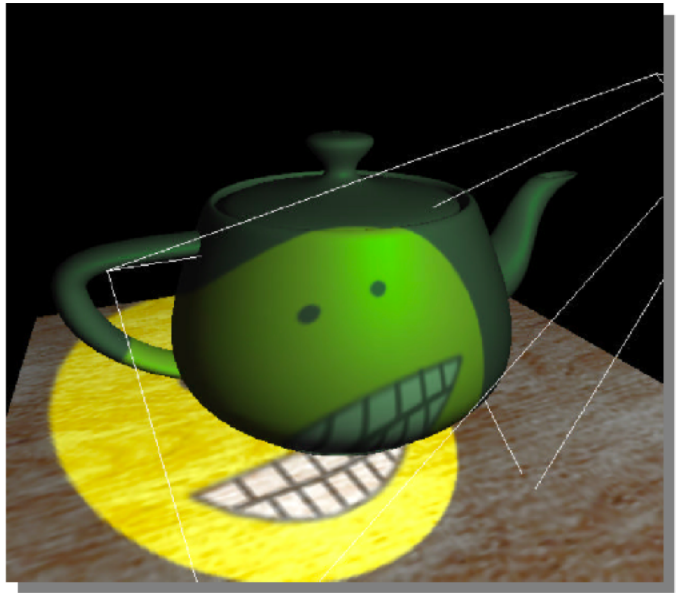
Unwrap object (e.g. in Blender)





PROJECTIVE TEXTURE MAPPING





Texture is used as light source

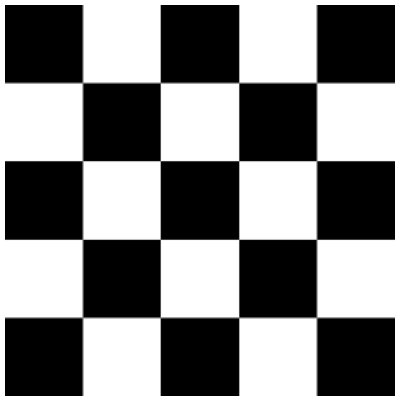
Like a slide projector

Uses input photos as textures with projective texture mapping

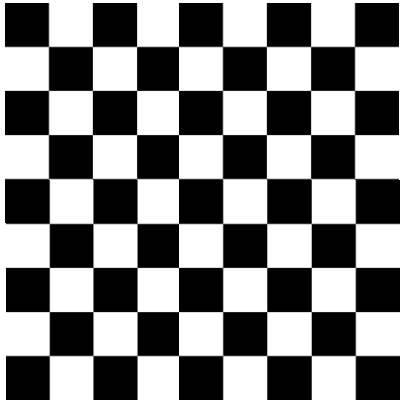
TEXTURE ADDRESSING

Texture coordinates have to be mapped to texels

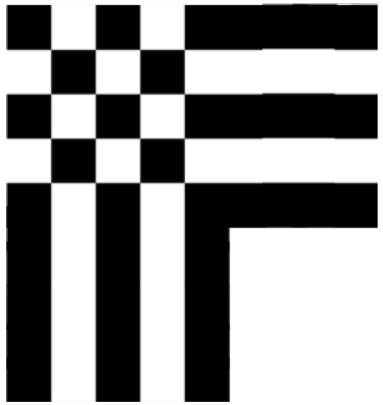
What happens if coordinates are outside of [0,1)?



Image



GL_WRAP

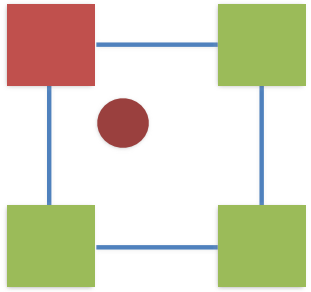


GL_CLAMP

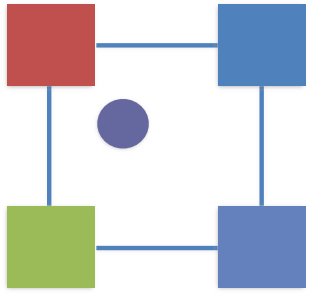
TEXTURE FILTERING

Interpolate texel value from neighbors

Nearest Neighbor
(lower quality)

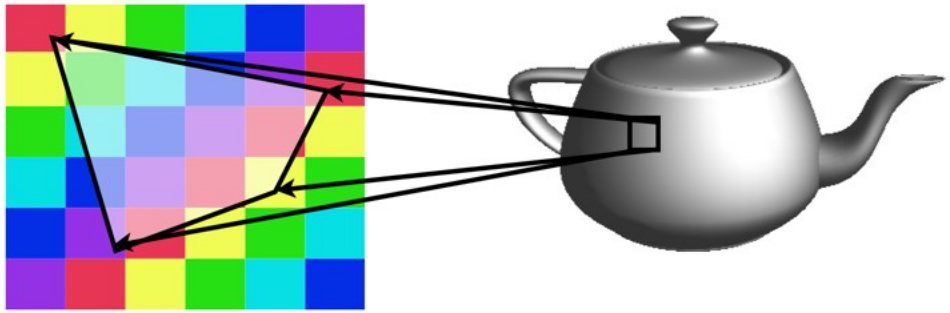


Linear interpolate neighbors
aka Bilinear Interpolation
(better quality, slower)

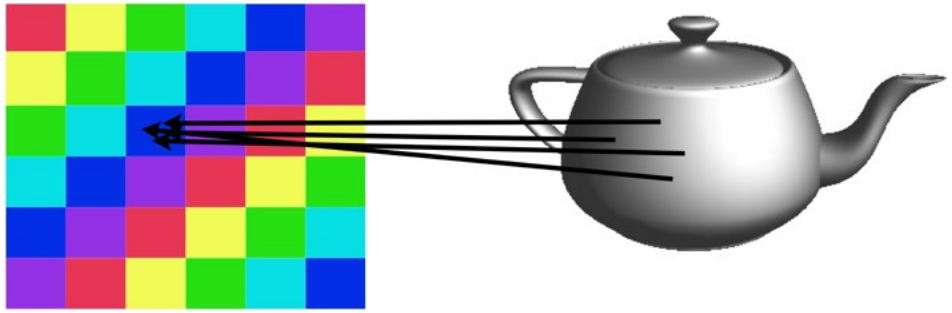


CHALLENGES

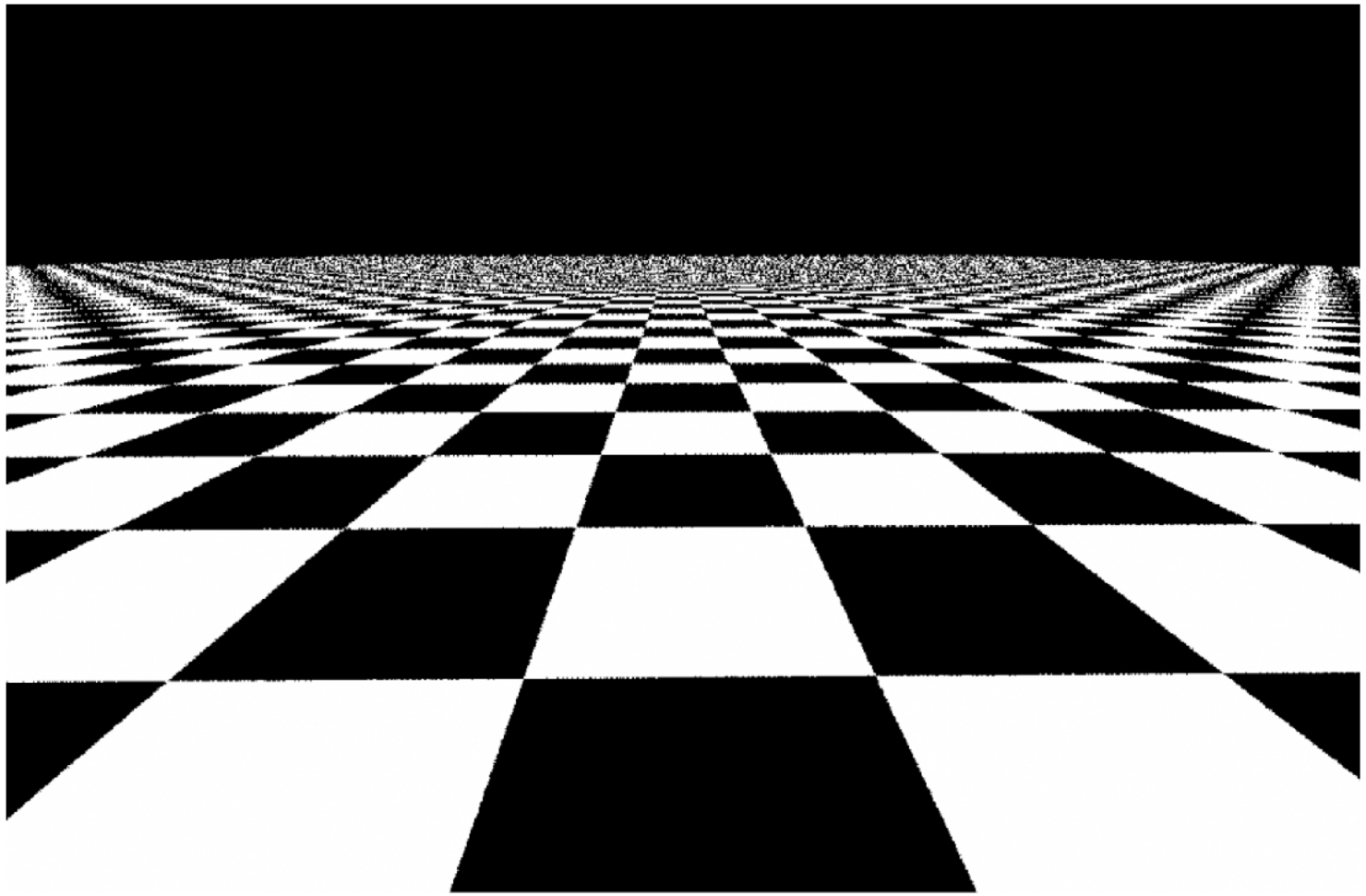
Undersampling: one pixel maps to an area covering many texel (Minification)



Oversampling: many pixels map to an area contained by only one texel (Magnification)



MIPMAPS



Problem of undersampling

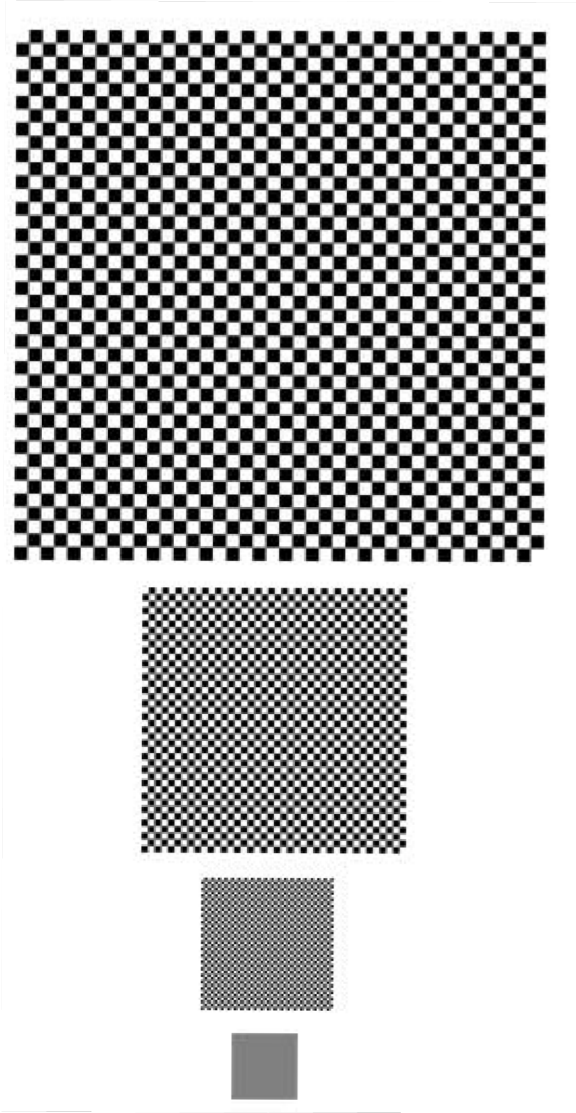
MIPMAPS

Start with most detailed texture

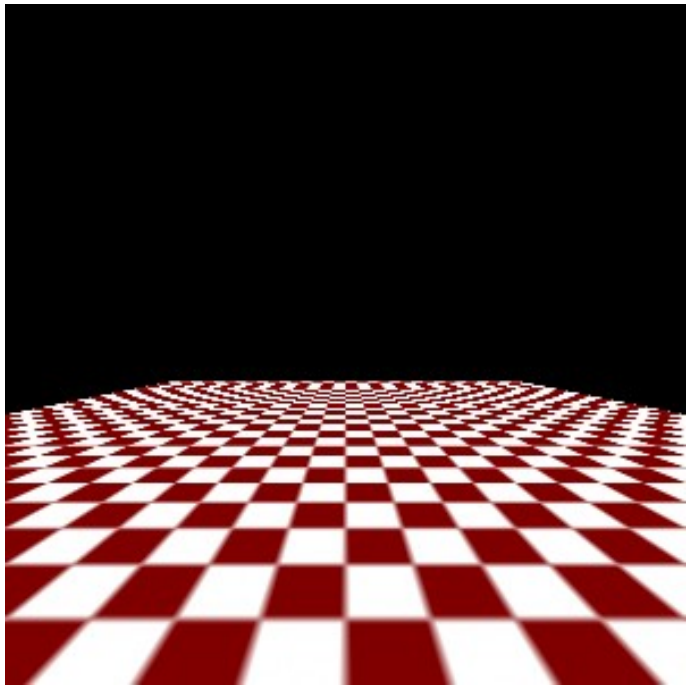
Reduce to half size by combining (averaging) adjacent four texels

Continue to minimum.

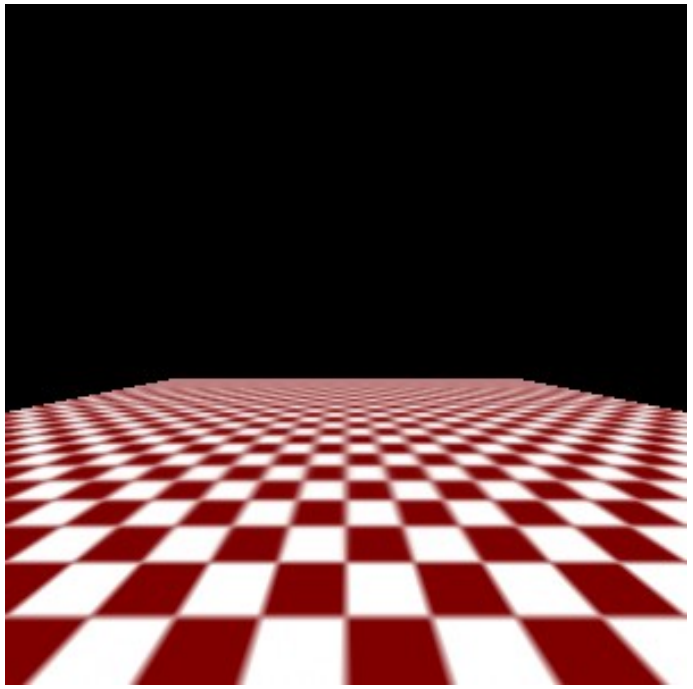
During rendering choose texture level according to distance



MIPMAPS



No mipmaps



Mipmaps

MIPMAPS


Floor (128x128)
mag: Linear
min: LinearMipMapLinear
Painting (748x600)
mag: Linear
min: Linear
Floor
mag: Nearest
min: NearestMipMapNearestFilter
Painting
mag: Nearest
min: Nearest

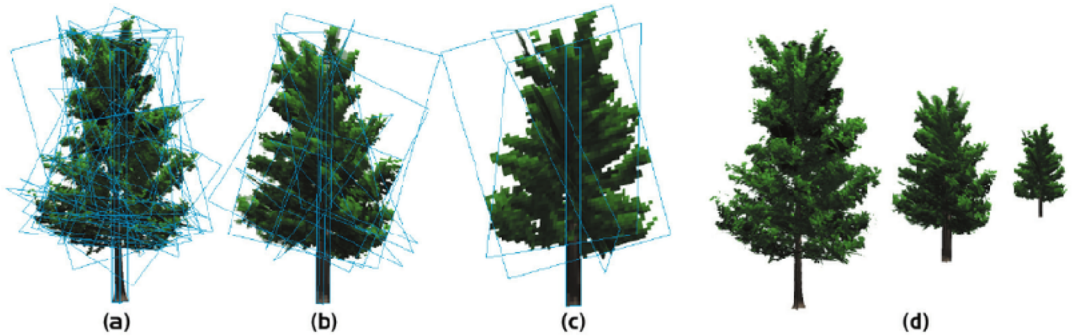
ALPHA MAPPING

Alpha map contains opacity values

Use for blending between multiple textures

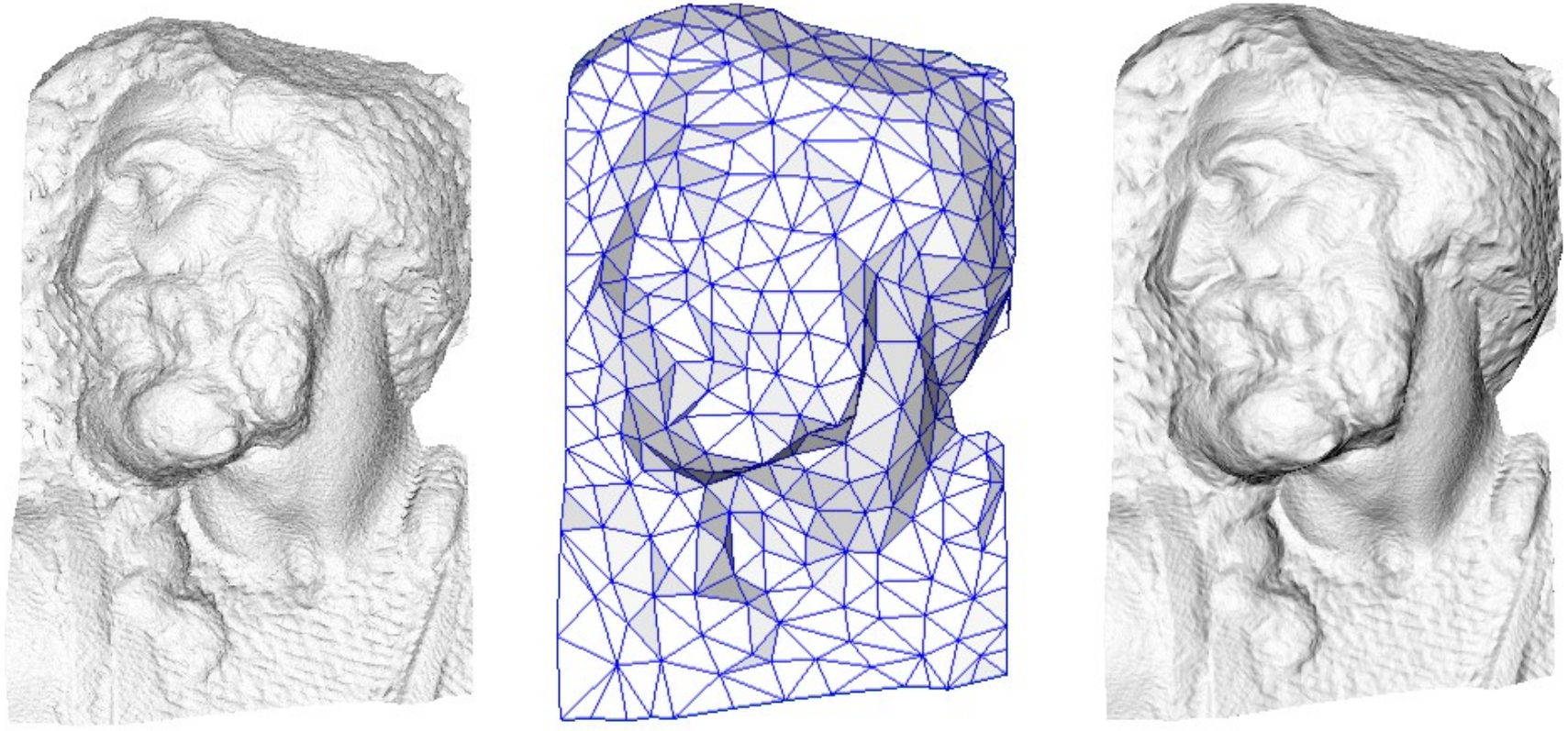
Creating transparency effects

Billboard rendering



Bump mapping/Normal mapping

Basic idea: instead of highly tessellated meshes. Store details in a texture and modify the *normal* during lighting

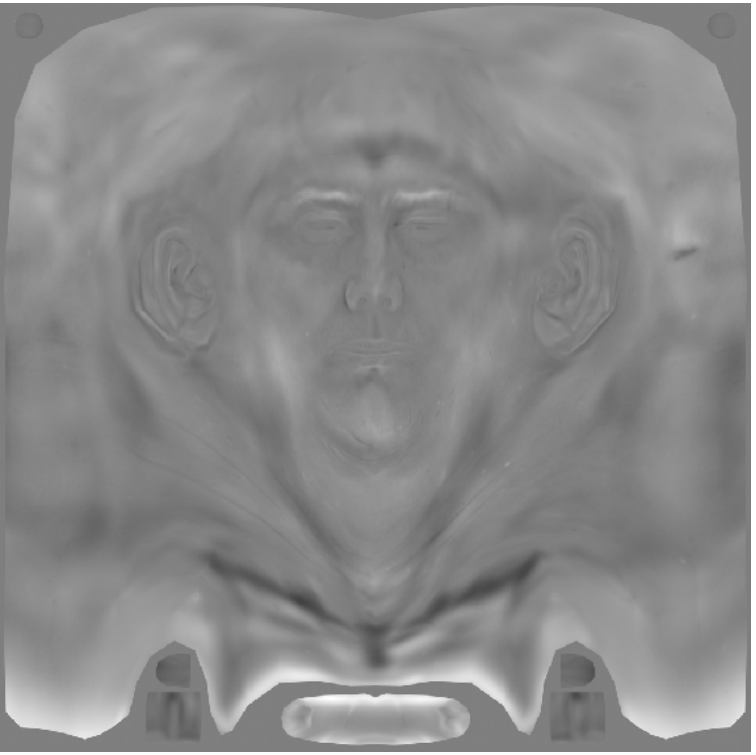


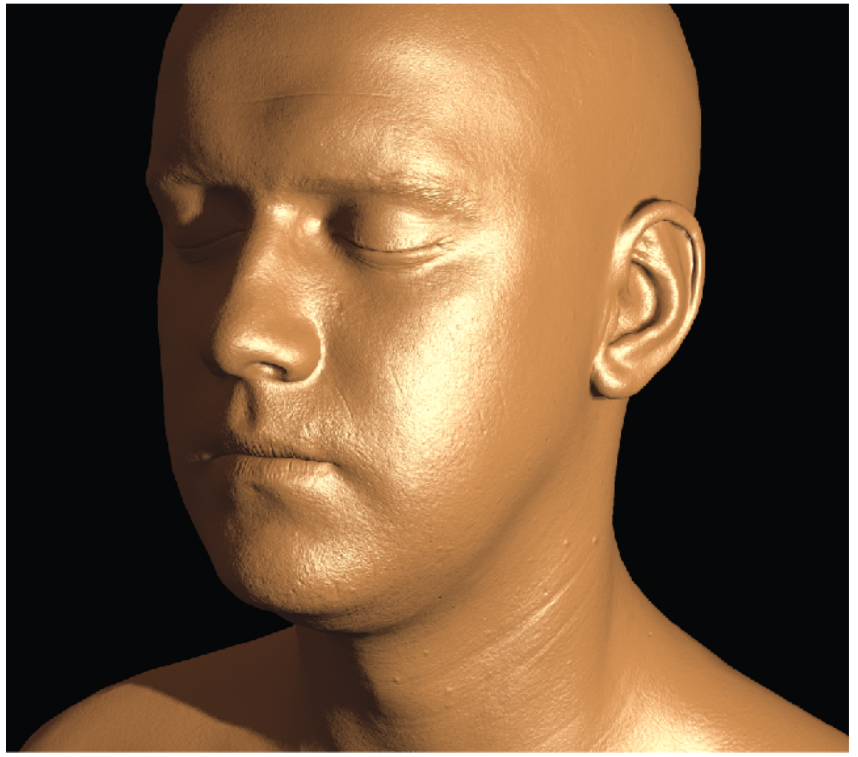
original mesh
4M triangles

simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles

BUMP MAPPING





HTTPS://THREEJS.ORG/EXAMPLES/#WEBGL_MATERIALS_BUMPMAP_SKIN

BUMP MAPPING

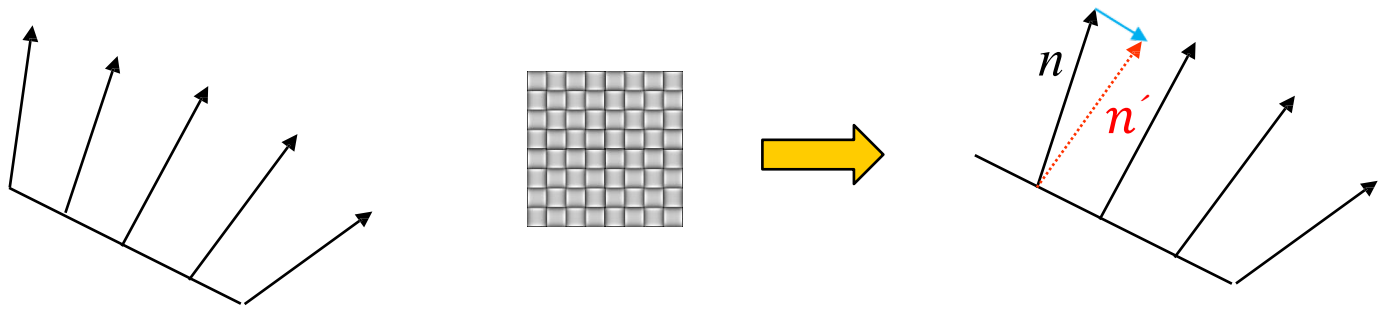
A neat trick to get flat surfaces appear uneven

Requires Phong shading

During lighting, we lookup the perturbation vector b (from the bump map texture) to calculate the perturbated normal n':

$$n' = (n_x + b_u, n_y + b_v, n_z)$$

This will bump the normal in the specified directions along the surface.

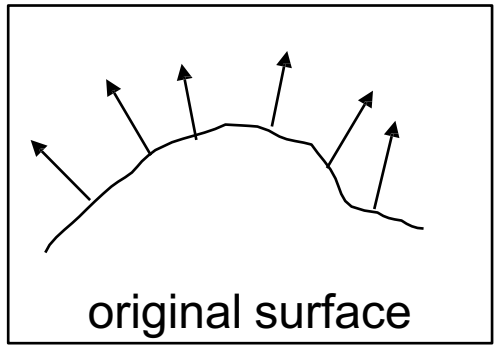


METHOD CREATED BY JAMES BLINN

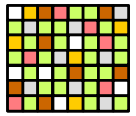
NORMAL MAPPING

Store the actual normal for each point on a surface in a texture

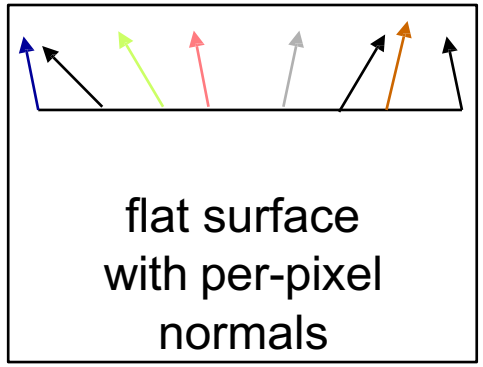
- Each texel contains rgb
- We encode the normals $n = (nx, ny, ny)$ as [-1,1] in rgb colors (0...255)
- Beware of artifacts
  - Interpolation requires renormalization and can produce odd results
  - 8-bit format will result in quantization errors



encoded normals in (rgb)



apply to flat surface each texel is the normal at that position in (u,v)

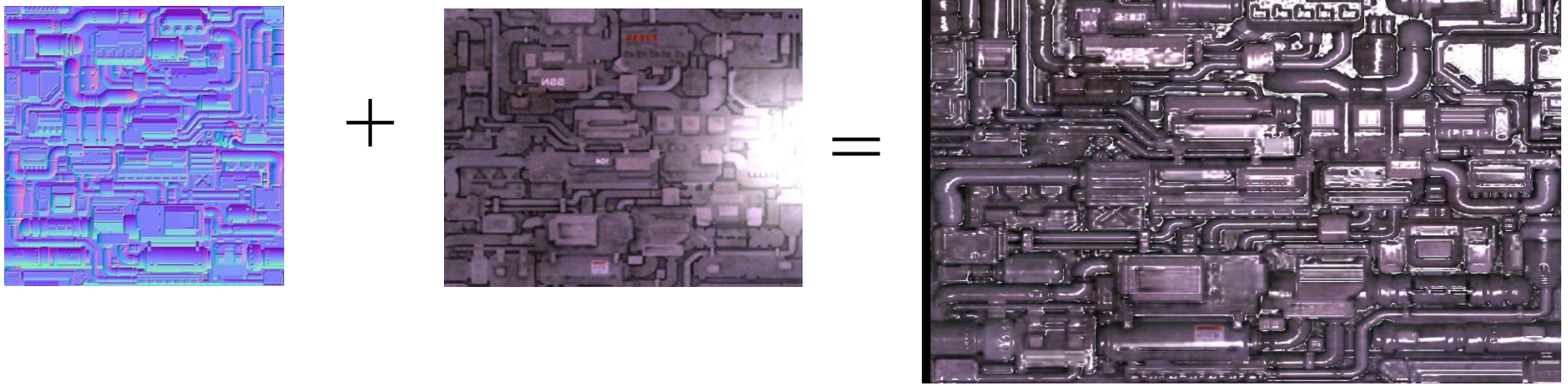


Normal mapping

Normals are stored in object space

For illumination:

- Transform into world space
- Or, (preferably) use tangent-space (object space)
  - Why? performance.



PARALLAX MAPPING

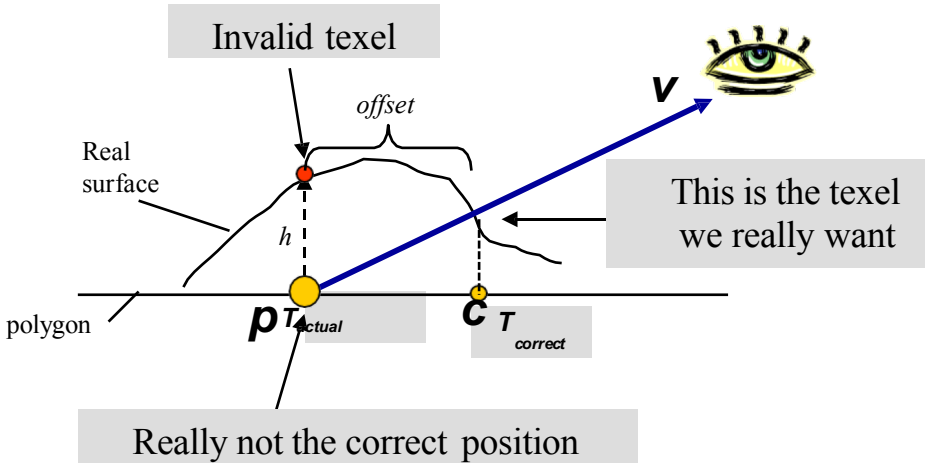
Normal mapping is approximating an uneven surface

- Loses the actual shape of the object, particularly bad at grazing angles

Parallax mapping is a better approximation

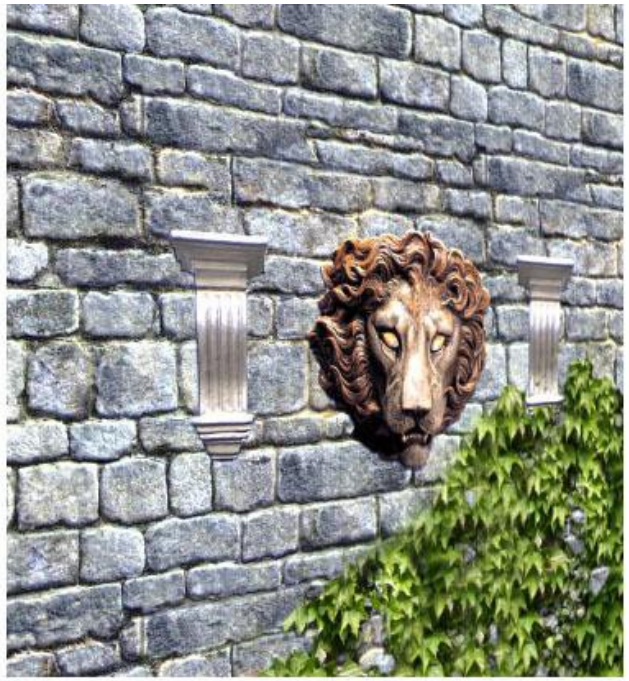
Uses normal map + height map

Requires a (somewhat) expensive search for the intersecting location in the height map



Parallax Mapping

Texture Mapped



Normal Mapped

Parallax Mapped

Steep Parallax Mapped

Morgan McGuire and Max McGuire (Iron Lore Entertainment)

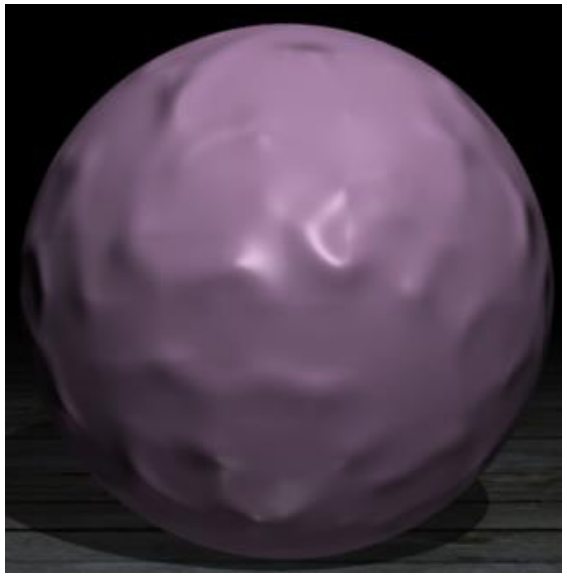
DISPLACEMENT MAPPING

Deform the actual geometry

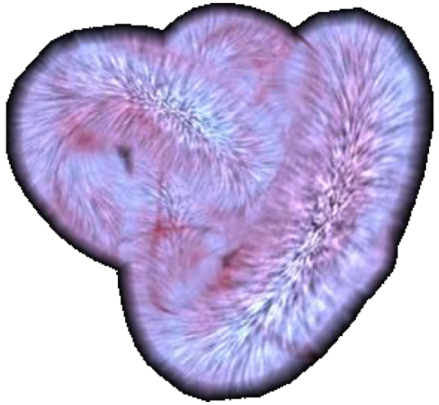
- Works for high resolution meshes or using tessellation shaders

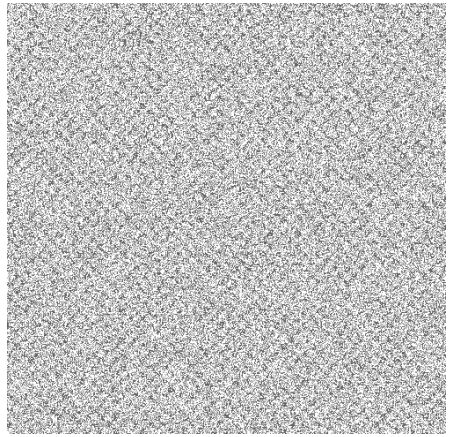
Perturb the location of the surface (the vertices), usually along the normal direction, by scalar values given in the displacement map

Gives a correct silhouette (whereas Normal mapping won't)



FUR SHADING

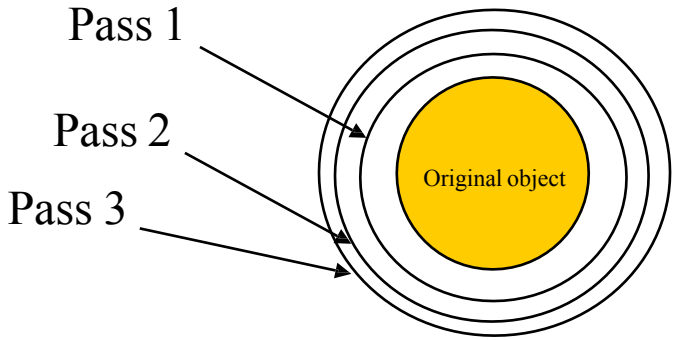




Alpha texture

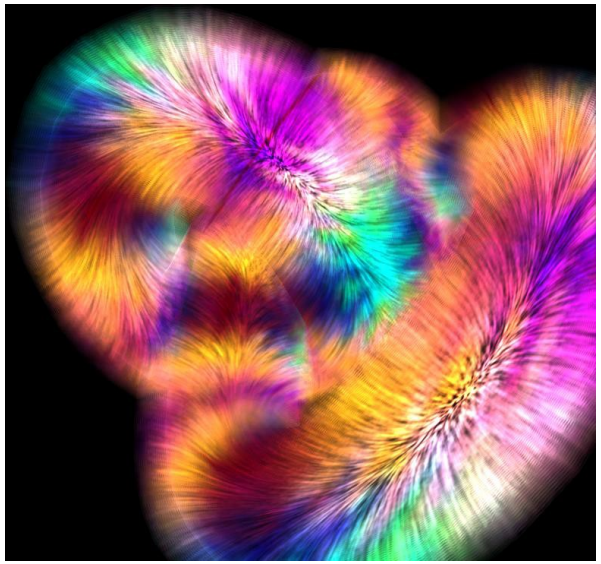
Vertex shader

- Render base object as usual, but move vertices (expand in direction of normal) outwards
- Transparency enabled
  - BlendEquation: ADD
  - BlendDestination: ONE
  - BlendSource: SRC_ALPHA

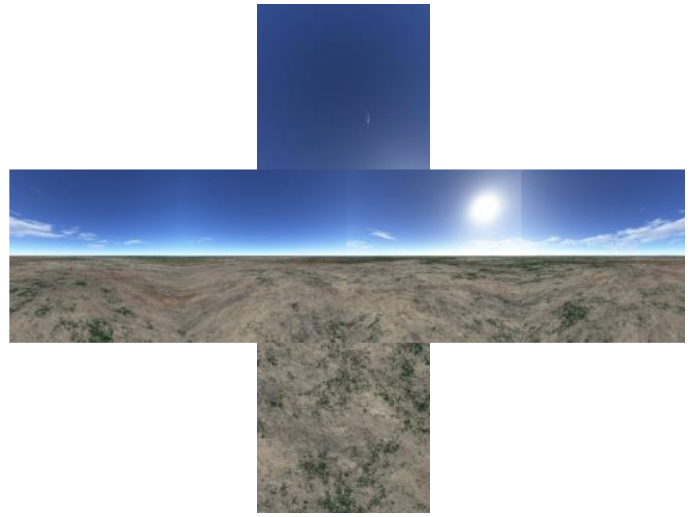


Render n-times (n ~ 14)

- Apply a fur (e.g., noise) alpha texture.



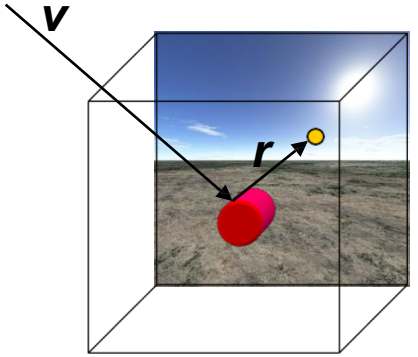
ENVIRONMENT MAPPING



In outdoor environments, we can't model everything

Skybox contains an image of the environment at infinity

- Imagine texturing a box with a surrounding distant environment.



For reflections—When rendering point p on the surface of an object inside the box, take the reflection vector r and shoot that towards the environment map

In OpenGL we have CUBE_MAPS that supports this.

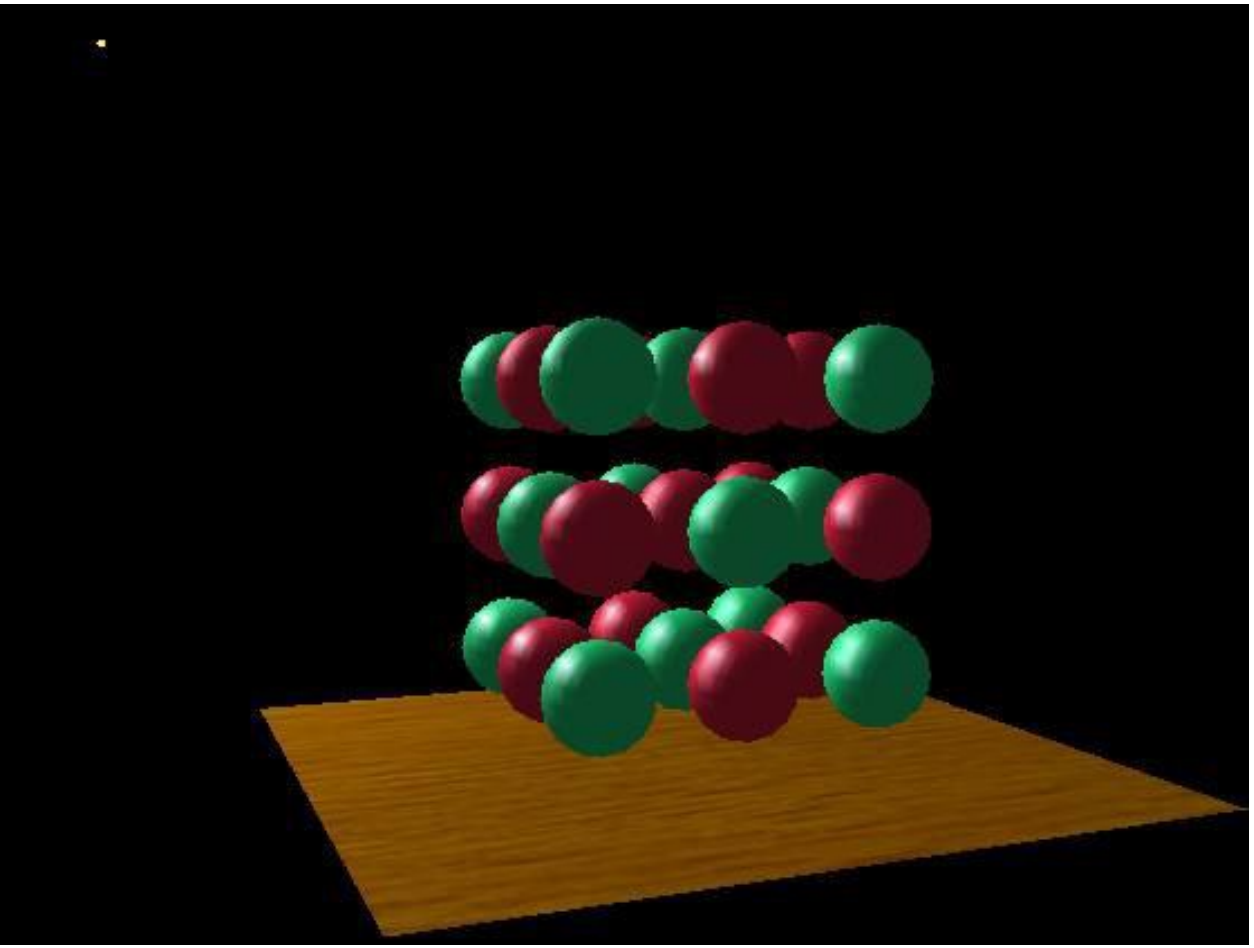
ENVIRONMENT MAPPING

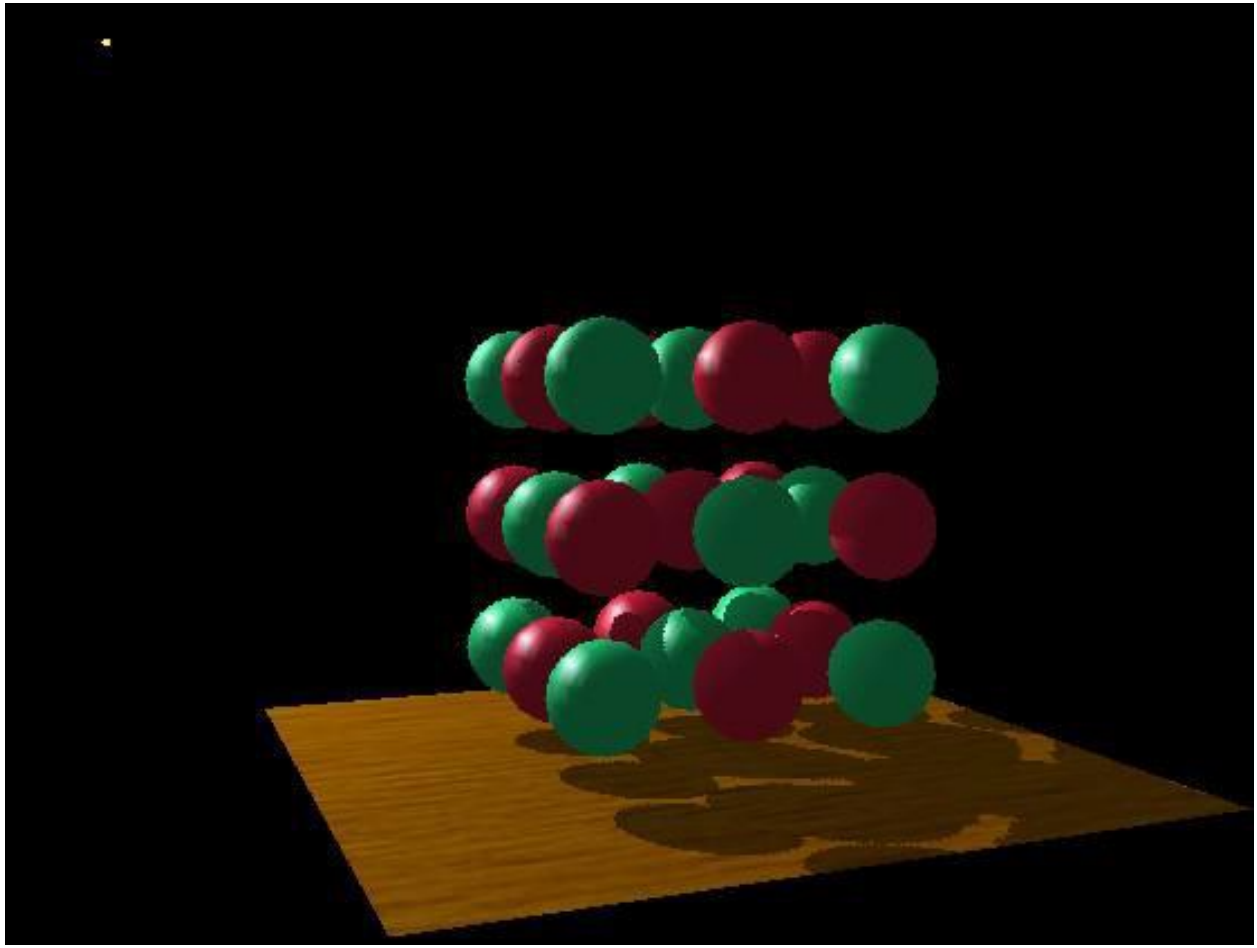
RENDER TO A TEXTURE

Some effects are only (efficiently) possible with multiple passes on the geometry or as a postprocess.

- Recreating nature:
  - Shadows
  - Creating mirrors, believable water surfaces with refraction
- Effects:
  - Deferred shading
  - Depth of field
  - Dynamic textures
  - Multi-pass techniques, anti-aliasing, motion blur
- Simulation
  - To use the GPU for other than just generating pretty pictures

Which image looks more natural?

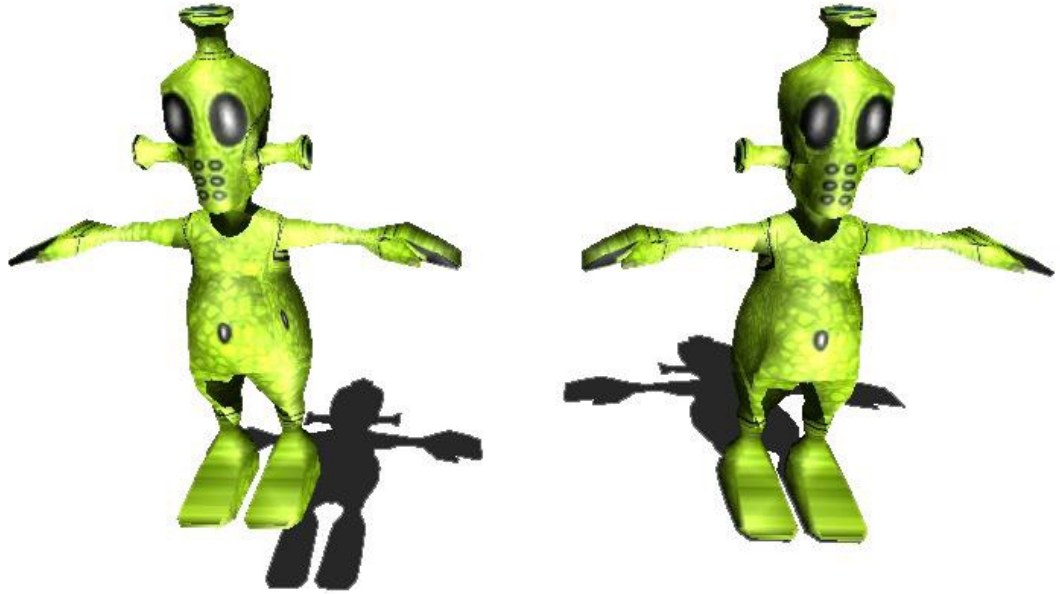


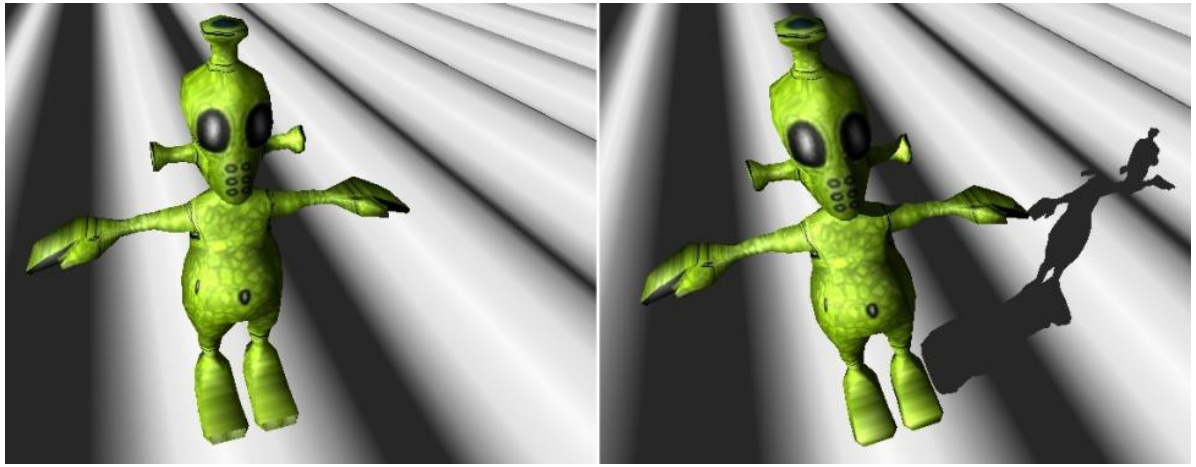


SHADOWS

Reveals the relative position between objects

Reveals the underlying geometry





Types of Light Sources in RT CG

Use point light sources for simplification

- Idealized mathematical point of view
- In real life there are no point lights

Area light sources

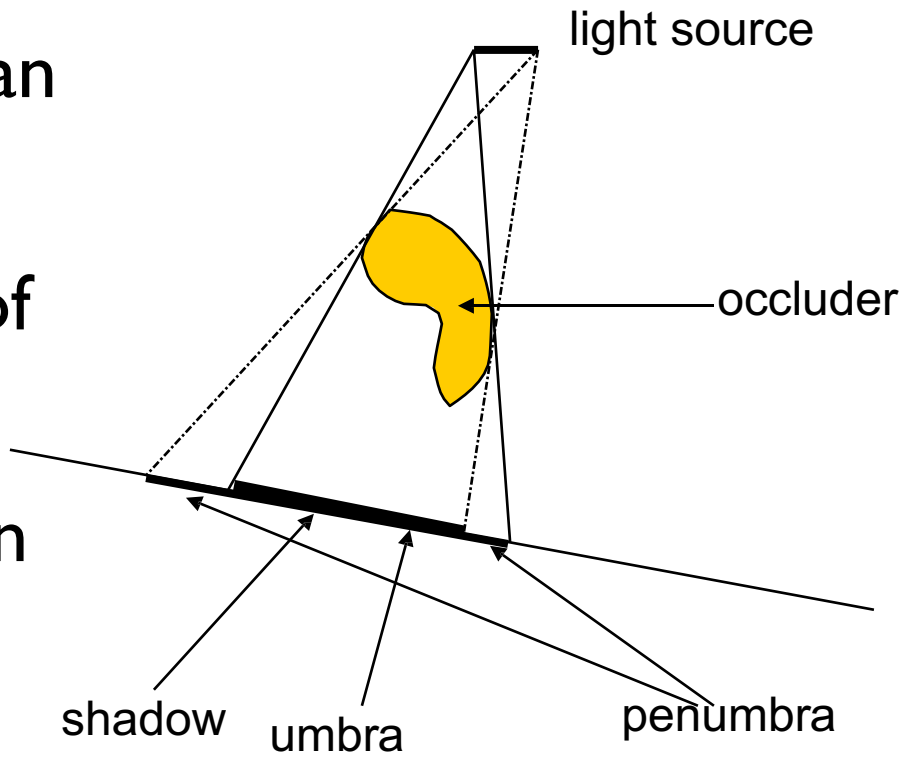
- Result in soft shadows (expensive to compute accurately)

SHADOWS

It is better to have an inaccurate shadow, than none at all (Wanger).

They eye is fairly forgiving about the shape of the shadow.

A blurred black circle applied as a texture on the floor can anchor a person to the ground



Shadow parts

- Umbra: Totally shadowed (hard shadow)
- Penumbra: Partially shadowed (soft shadow)

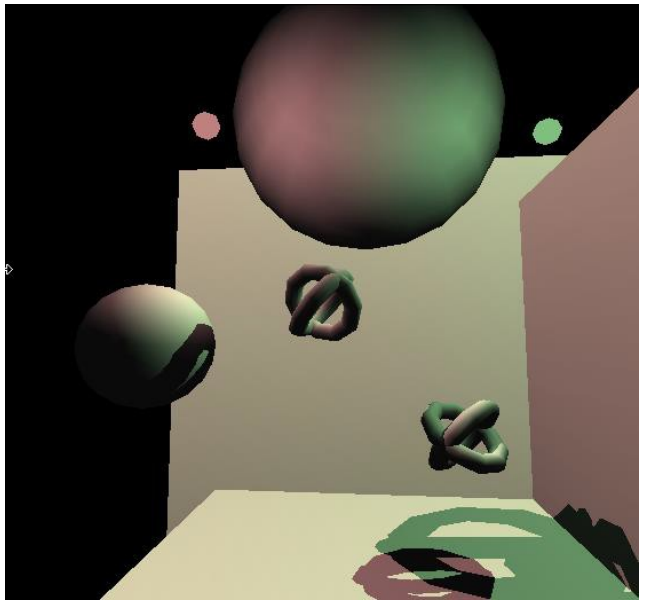
Hard Shadows

We want:

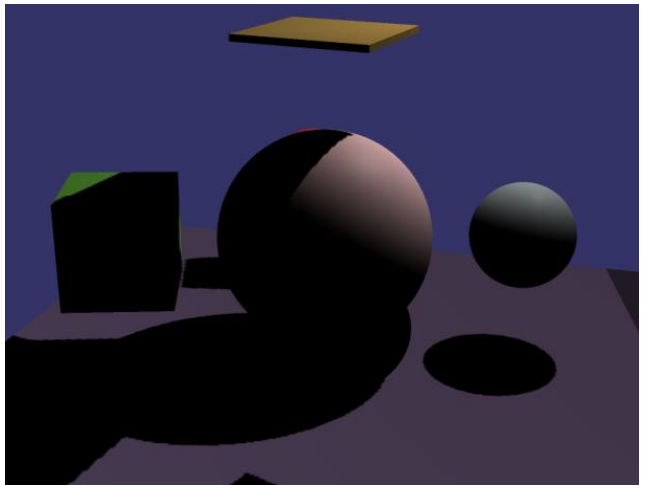
- Speed
- Shadows on curved surfaces
- Self shadowing (occluders can shadow themselves)

Two basic algorithms

- Shadow volumes (Crow, 1977).
- Shadow maps (Williams, 1978).



Shadow volumes



Shadow Maps

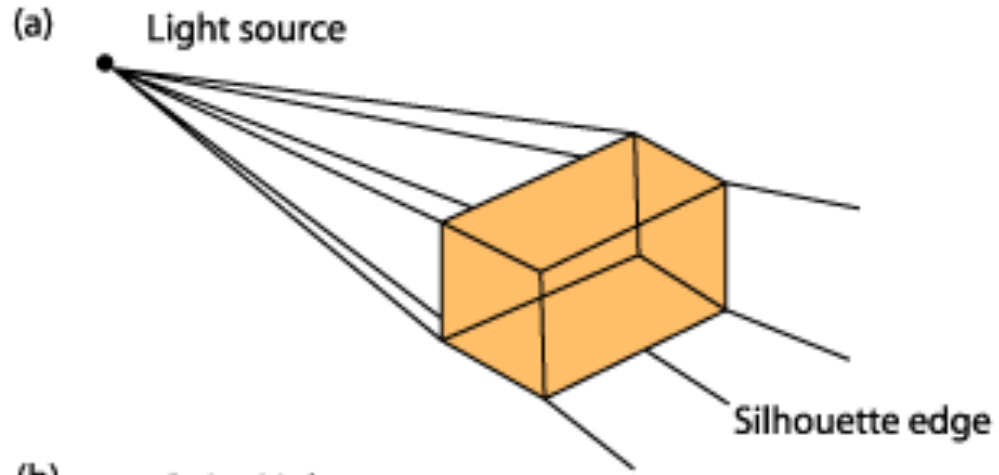
SHADOW VOLUMES

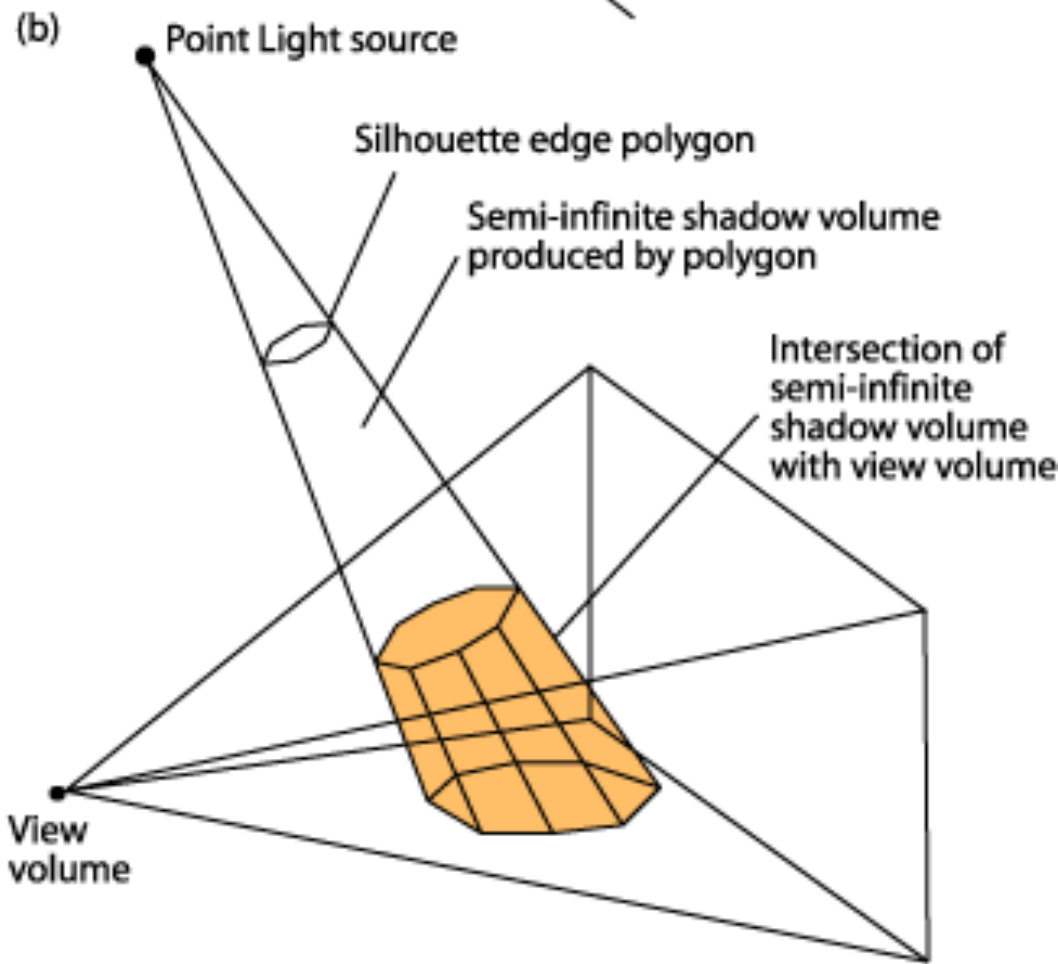
Shadows can be seen as the volume of the view frustum it contains.

- i.e., all points inside a shadow volume are in shadow.

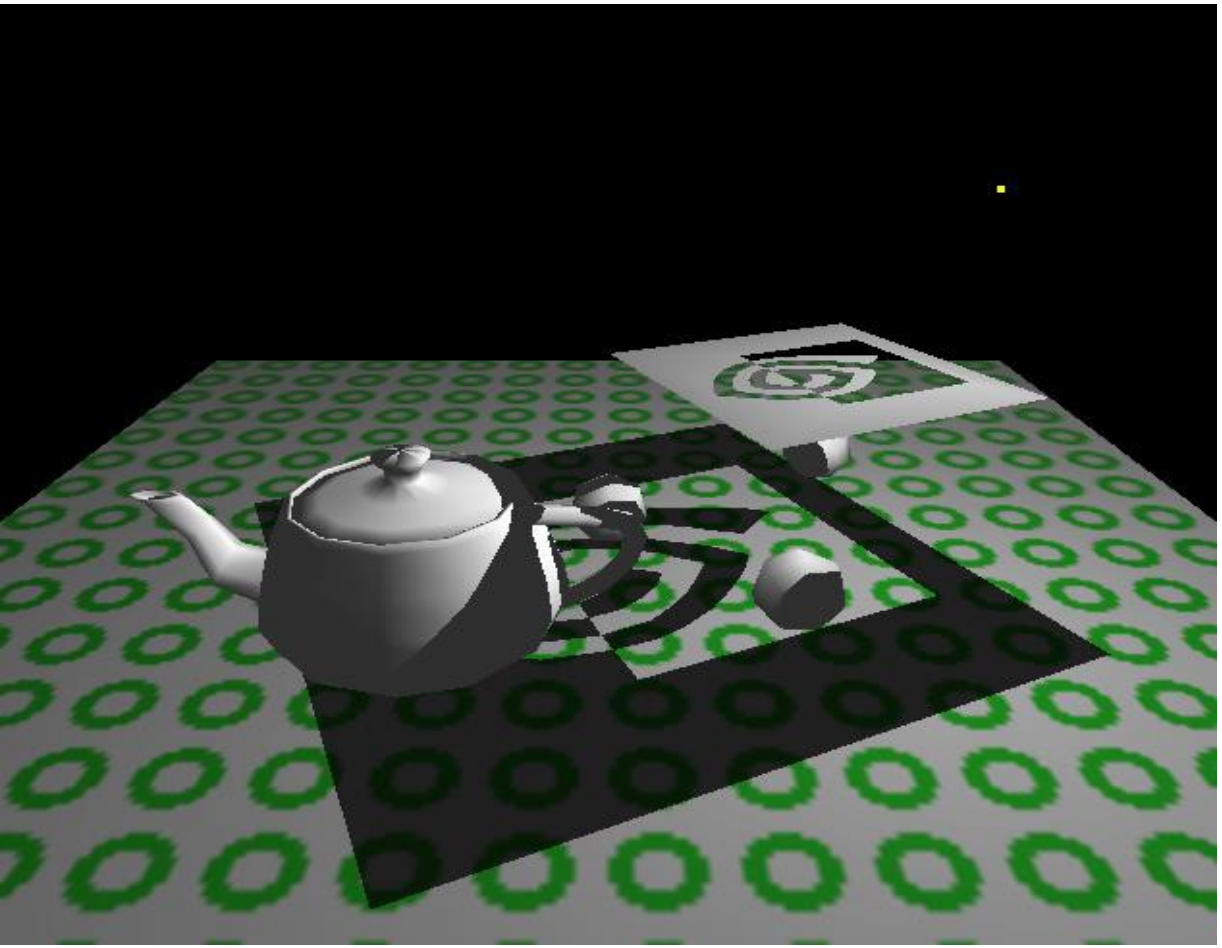
This volume is defined by the polygons enclosing it, shadow polygons.

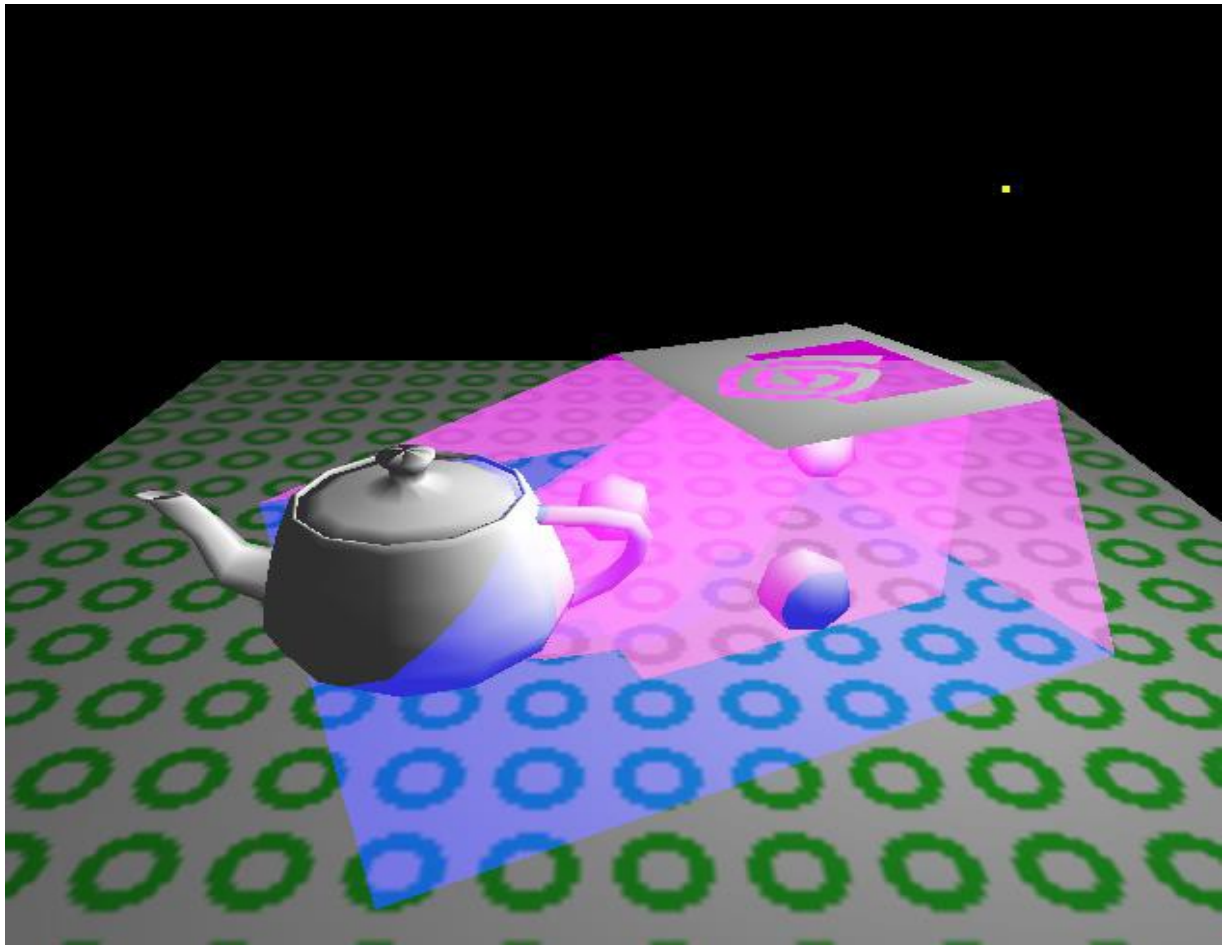
A shadow polygon is created by creating a polygon emerging from the occluders silhouette (seen from the light source) away from the light source.





SHADOW VOLUMES





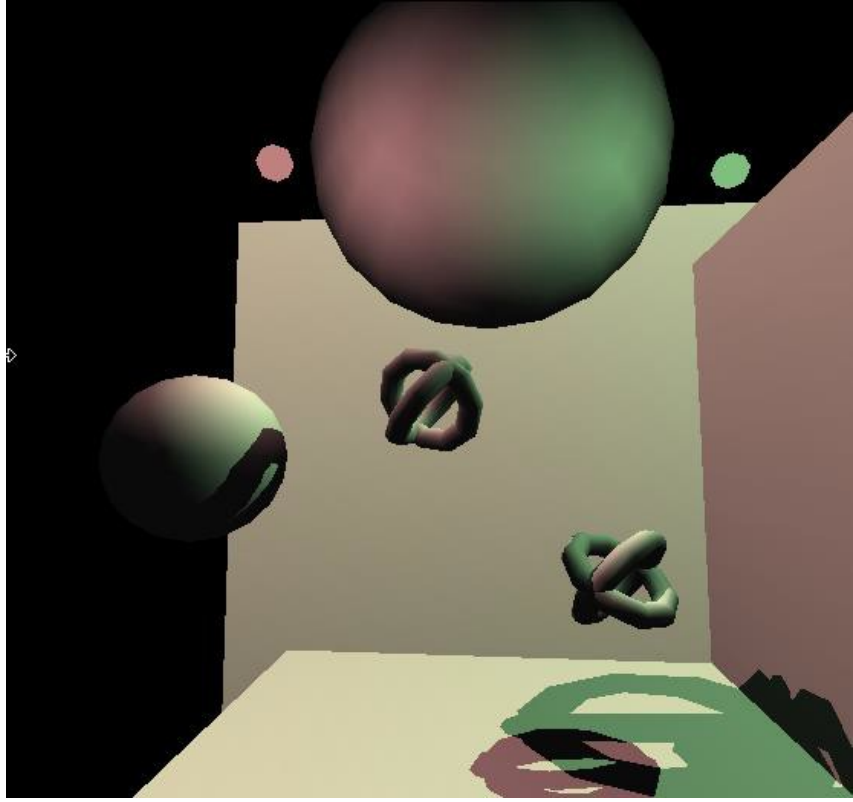
SHADOW VOLUMES

Advantages

- Support omni-directional lights
- Exact shadow boundaries

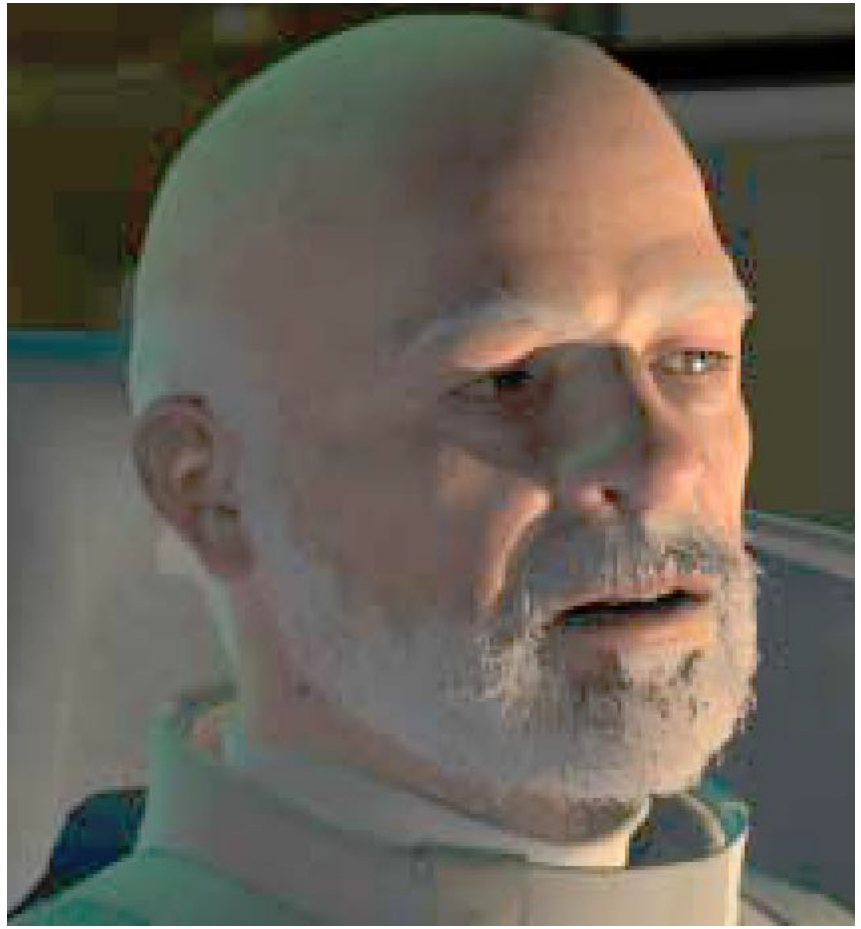
Disadvantages

- Fill-rate intensive
- Expensive to compute shadow volumes / silhouettes
- Hard shadow boundaries, not soft shadows
- Difficult to implement robustly (silhouettes)



Doom3

Shadow Maps



Shadow Maps

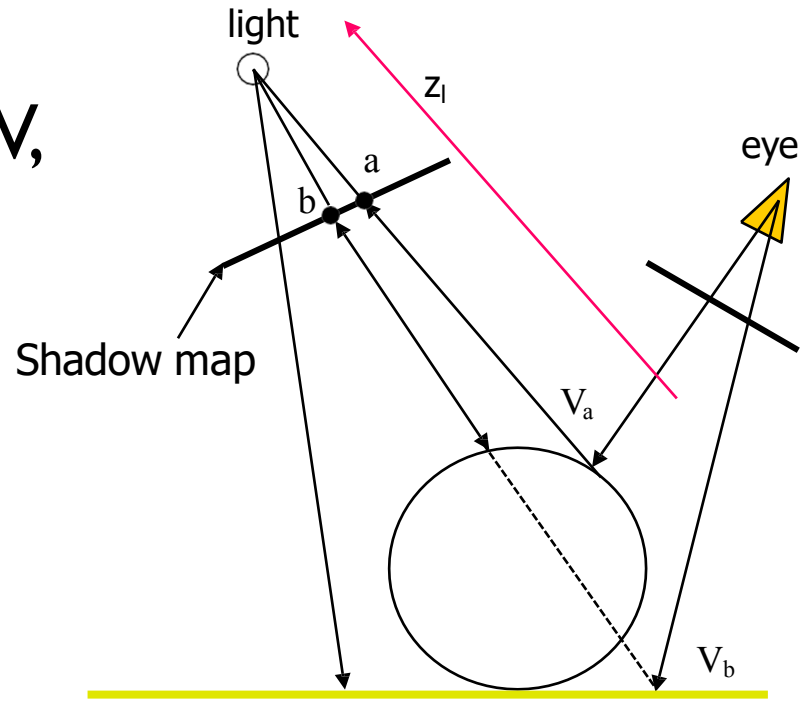
Render the scene twice

First from the light source

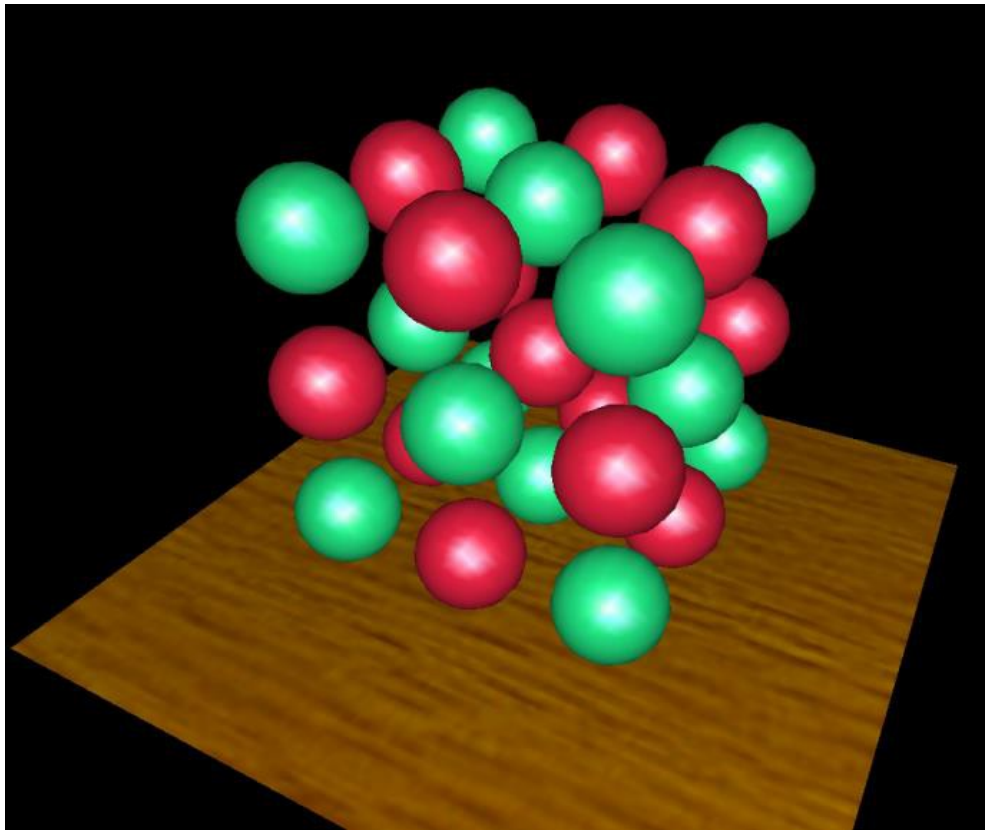
- Establish projection (position, direction, FOV, etc.)
- Render into depth texture (color & other effects disabled)

Second from the camera

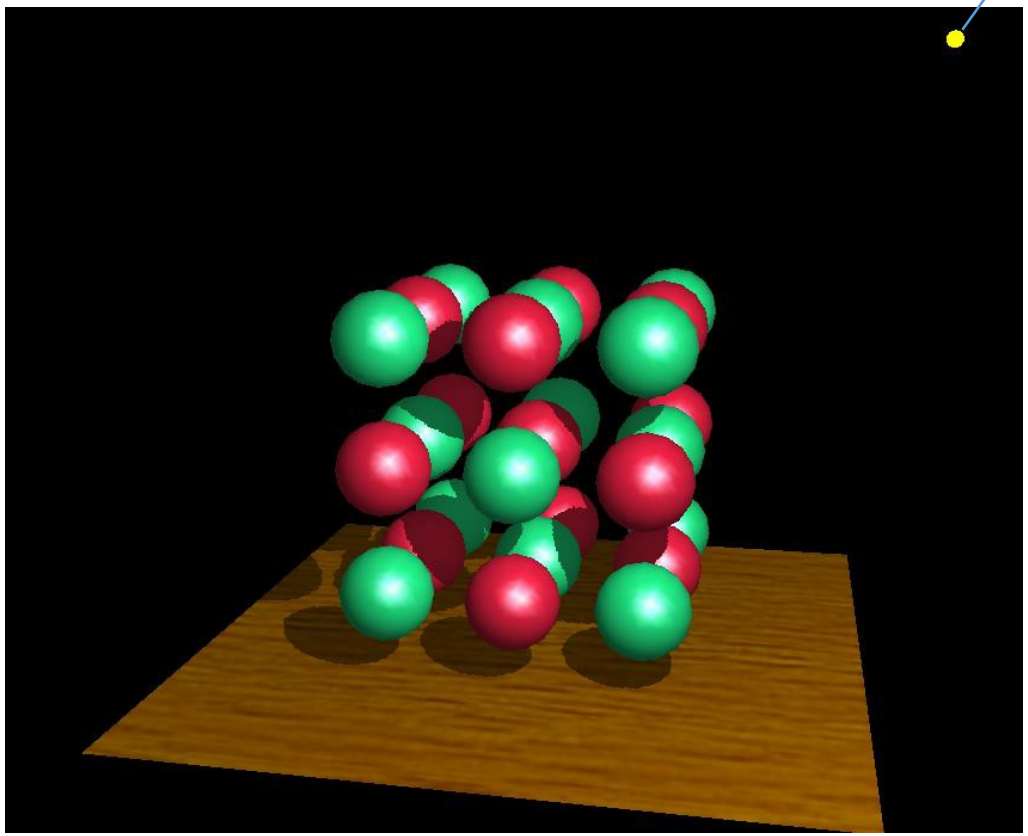
- At each pixel, project into depth texture (using light's projection)
- Compare projected z to depth texture z
  - If less than or equal to this value it is lit (point closer to camera than corresponding point in Shadow Map).



Shadow Map Examples



Lights point of view



Eye's point of view

Shadow Map: Advantages

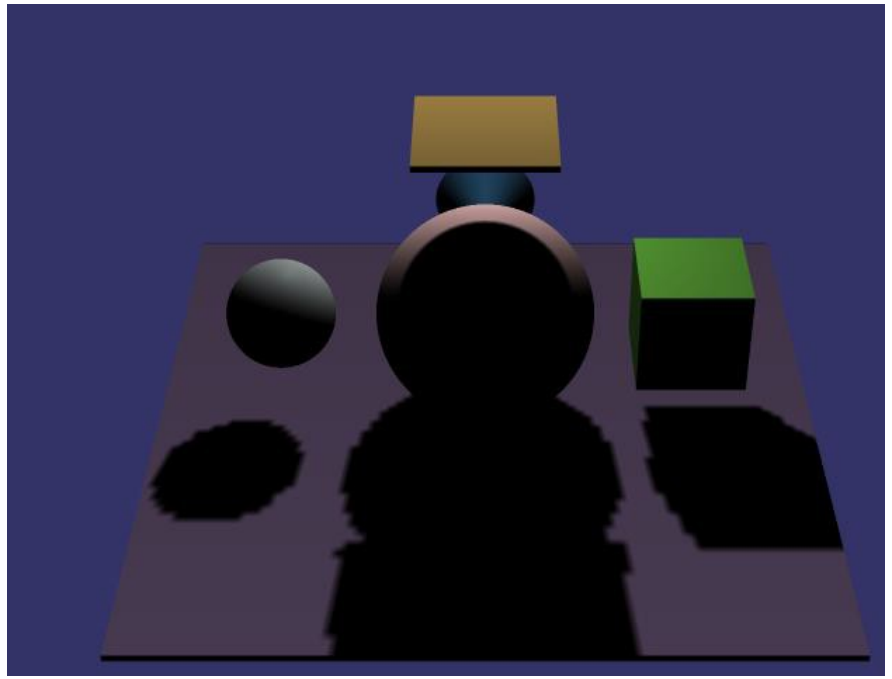
Fast, supported by hardware

Extra pass on the geometry
(without any shading)

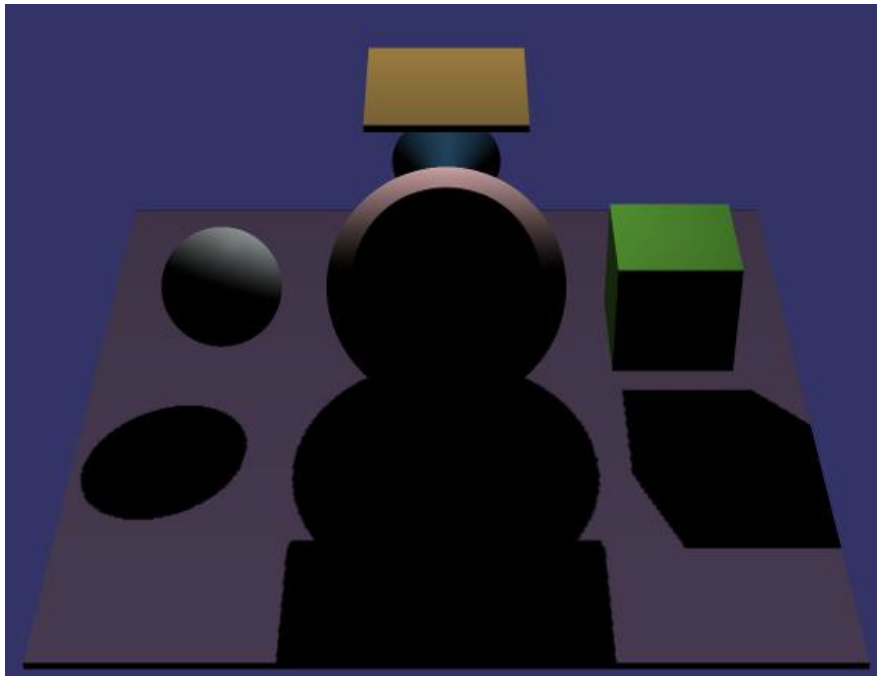
Shadow Map problems

Aliasing problems: resolution of the shadow buffer is limited

128x128



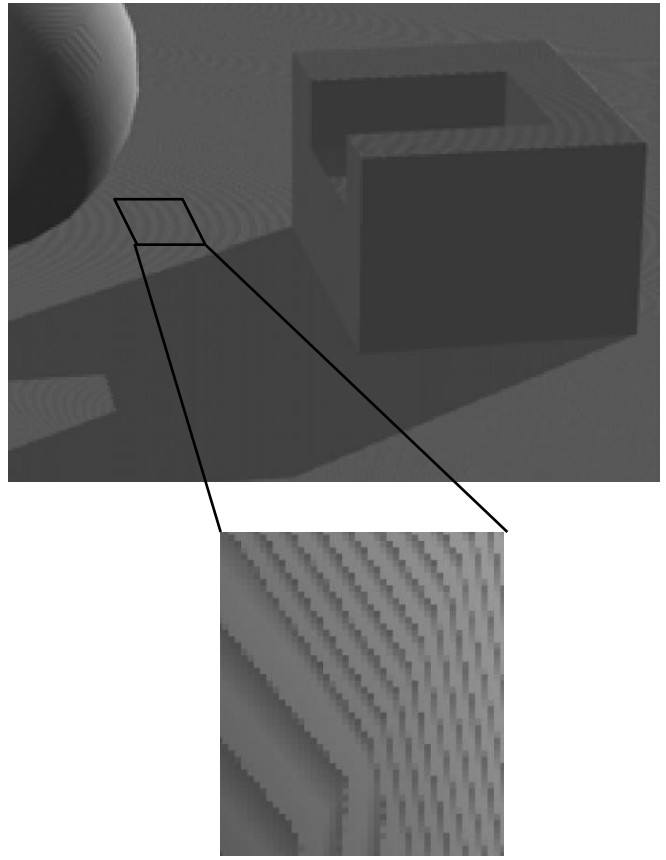
1024x1024



Shadow Map problems

Precision of the Shadow map (Acne)

- Floating point precision: stored value in the shadow buffer have limited precision (8/16/24/32 bits)
- Texture sampling location
- The equality test suffers from poor imprecision



Shadow Map Problems

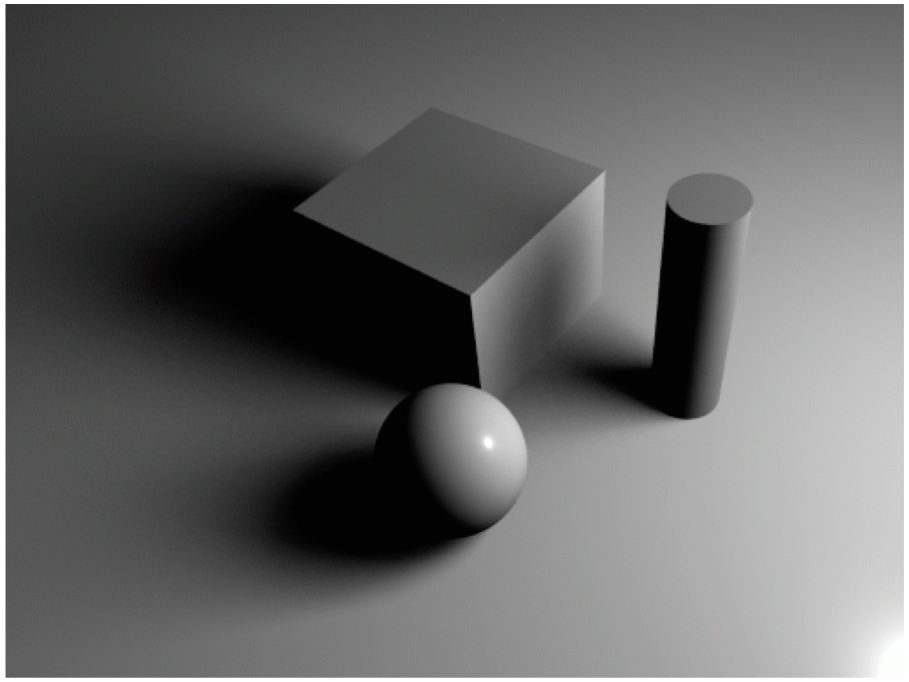
Lots of advanced techniques to reduce the impact of these issues (they come at a cost)

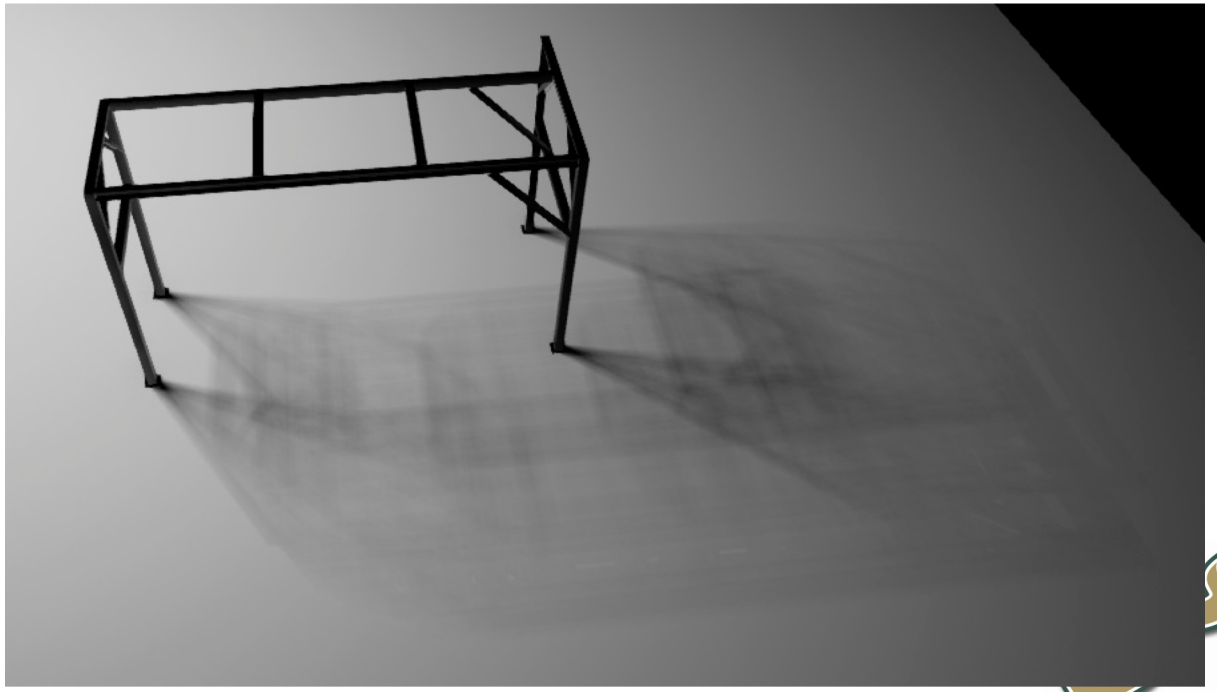
SOFT SHADOWS

If light occupies an area, we should get soft shadows

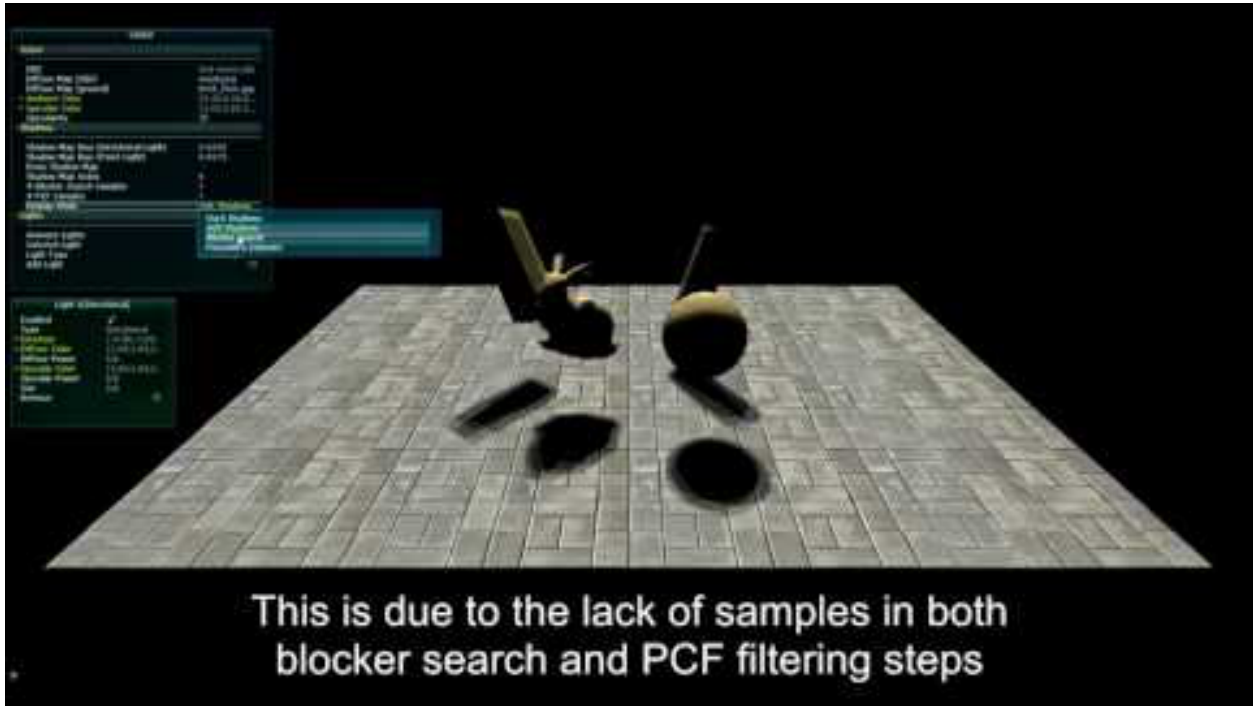
Common approaches:

- Raytracing, Radiosity
- Sample an area light, by rendering shadows from several positions at the surface of the light and average the result (accumulation)
- Jitter (sample) shadow test location





Soft Shadow Maps



Percentage Closer Filtering

Screen Space Soft Shadows

https://youtu.be/C70V0UGtoAw
https://youtu.be/g-AFYDhyN3w

FORWARD SHADING VS DEFERRED SHADING

Forward shading

- Lighting calculations performed for all fragments on all geometries

Deferred shading

- Lighting calculations only on visible fragments as a postprocess

Deferred Shading

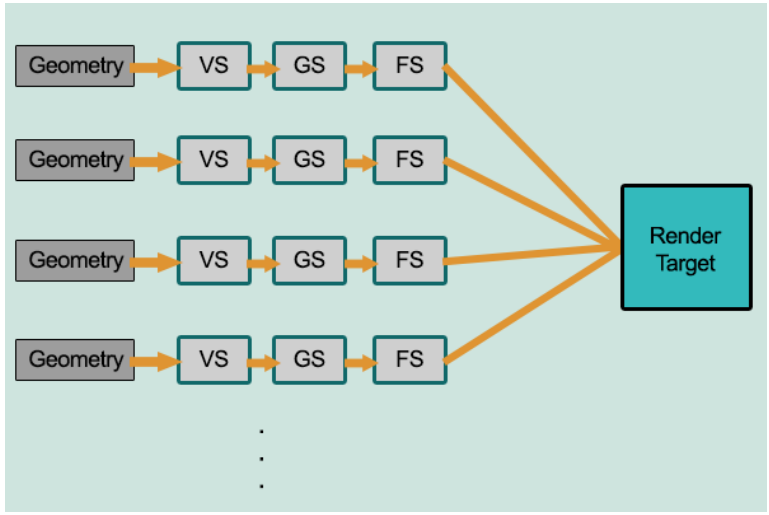
Do expensive lighting equation on visible geometry only

Lights can have "active volume" – avoid shading for invisible lights

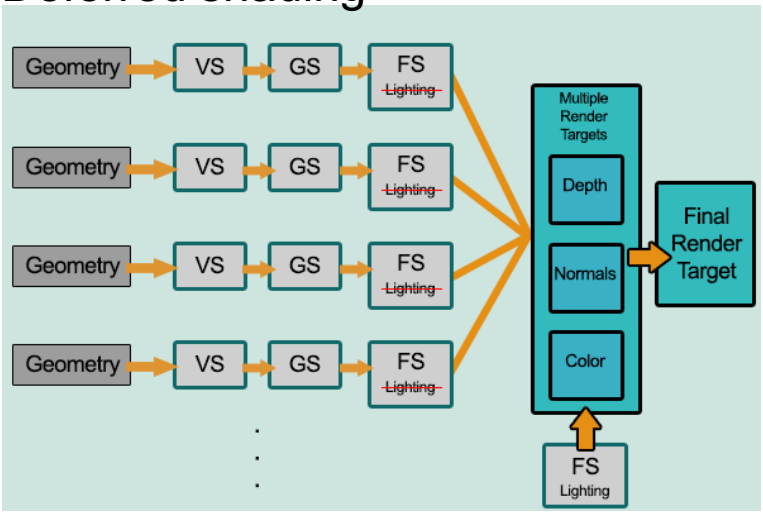
Render to G-Buffer

- Perform shading on G-Buffer

Forward shading



Deferred shading



DEFERRED SHADING

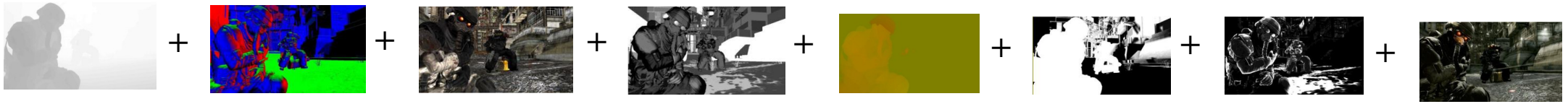
G-Buffer (geometry-buffer) contains:

- Depth buffer (geometry) – 32 bit FLOAT
- Normal buffer – 32 bit (RGB)
- Diffuse (albedo) color – 32 bit (RGB)

Other possible buffers (example from Killzone 2)

- Specular strength
- Roughness
- Light buffer
- Render light source volumes accumulated to a buffer
- Determine where lights are active
- Motion vector buffer
- Sunlight occlusion

FINAL RESULT



=

R8	G8	B8	A8
Depth 24bpp			Stencil
Lighting Accumulation RGB			Intensity
Normal X (FP16)		Normal Y (FP16)	
Motion Vectors XY		Spec-Power	Spec-Intensity
Diffuse Albedo RGB			Sun-Occlusion

Deferred shading

Pros

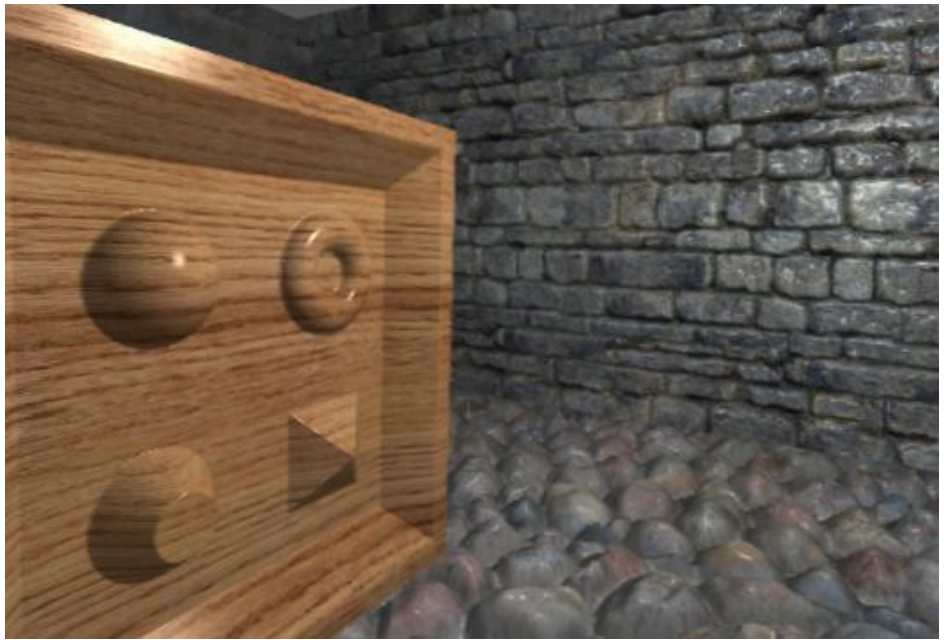
- Efficient light calculation
- Allows for many light sources (thousands)
- "Simpler" rendering pipeline (avoids "Mega Shaders")

Cons

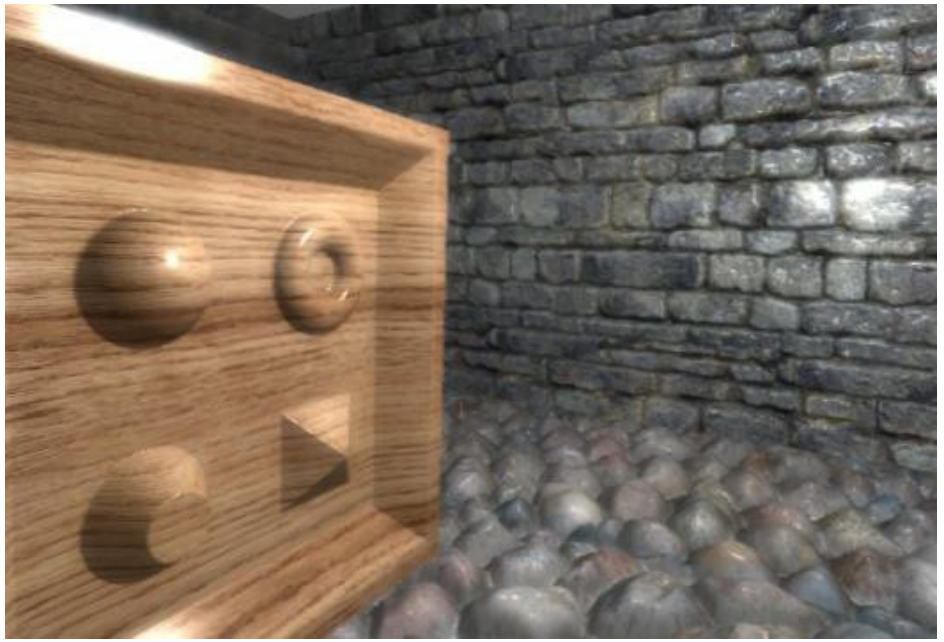
- Transparency can be a problem
  - Can be handled by sorting, depth-peeling, order-independent-transparency
- More materials requires more buffers
- Antialiasing in HW not used
  - Can be solved with edge detection + smoothing

POST-PROCESSING EFFECTS

Depth of field, blur, bloom, antialiasing, motion blur,...



Without bloom



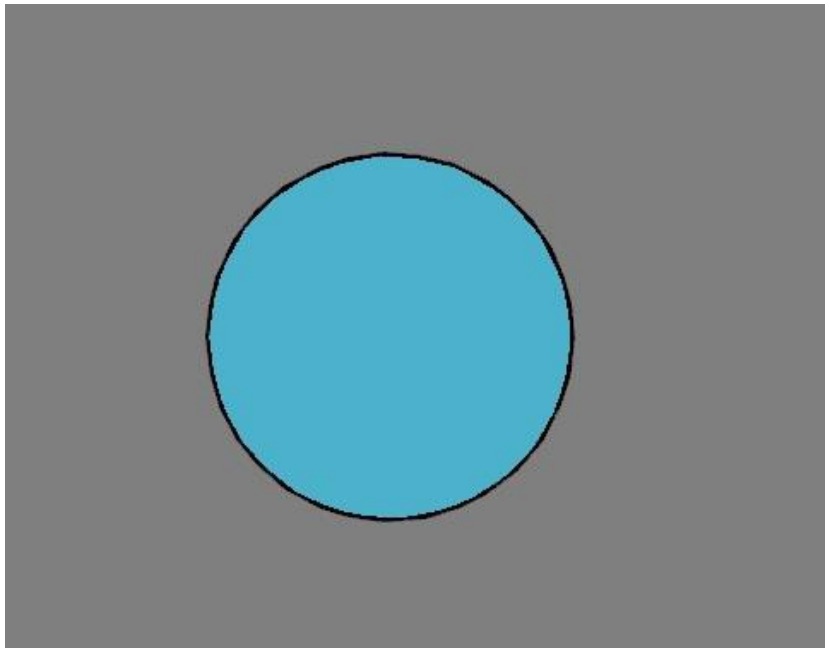
With bloom

Toon Shader

Determine dot product between view vector and normal

- $V \cdot N \in (-\infty, 0] \rightarrow$ black
- $V \cdot N \in (0, 0.5) \rightarrow$ base color
- $V \cdot N \in [0.5, \infty) \rightarrow$ highlight color

Can utilize 1D texture for flexibility



```
float color = texture1D(toonTexture, dot(V_eye, N));
```