# MSc Computer Games and Entertainment

Maths & Graphics Unit 2011/12

Lecturer: Gareth Edwards

# Shading

## An Overview

# Acknowledgments

Artwork and source code has been taken from various academic and commercial sources ,including:

- Cornell University – Bruce Land
- Lund University - Petrik Clarberg
- Dartmouth Computer Science - Fabio Pellacini
- Cardiff University - School of Computer Science
- Wikipedia

## Lecture Overview

- What is shading?
  - flat, facet, Gouraud, Blinn, Phong
- Surface characteristics
  - colour, light response, texture, roughness
- Shading Trees and Shading languages
- Light response
  - BRDF (bidirectional reflectance distribution function), non-physical styles
- Texturing and painting
  - projections, parameterization, procedurals
  - bumps and displacements
- Special Effects
  - transparency, refraction, atmospheric

## What is Colour?

- Colour is....
  - human perception of light, i.e. power spectrum of electromagnetic radiation
  - very complicated subjects!
  - involves a lot of physiology and psychology!
  - not fully understood yet!

- For our purposes we will consider it to be the main colour of an object, surface, polygon, etc.

**Colour** is the visual perceptual property corresponding in humans to the categories called *red*, *yellow*, *blue* and others. Colour derives from the spectrum of light (distribution of light energy versus wavelength) interacting in the eye with the spectral sensitivities of the light receptors. Colour categories and physical specifications of colour are also associated with objects, materials, light sources, etc., based on their physical properties such as light absorption, reflection, or emission spectra. By defining a colour space, colours can be identified numerically by their coordinates.

## Colour

- Visible colours can be describe using 3 numbers
  - which ones?
  - choose the easier representation

- Red, Green, Blue (RGB)
  - colour used by monitors to create images

## Colour

- RGB is hard to use
  - changes to lightness and saturation

- Hue, Saturation, Value (HSV)
  - easier to specify colors
  - based on human perception

## What is shading?

- Defining the material characteristics
  - colour
  - light response
  - textures
  - roughness

- The colour of 3D objects is not the same everywhere
  - an object drawn in a single colour appears flat
  - light-material interactions cause each point to have a different colour or shade in 3D

- Global shading requires to calculate all reflections between all objects
  - in general this is not computable

- Typically a simplified local model is adopted, such as Gourard, Phong or Blinn

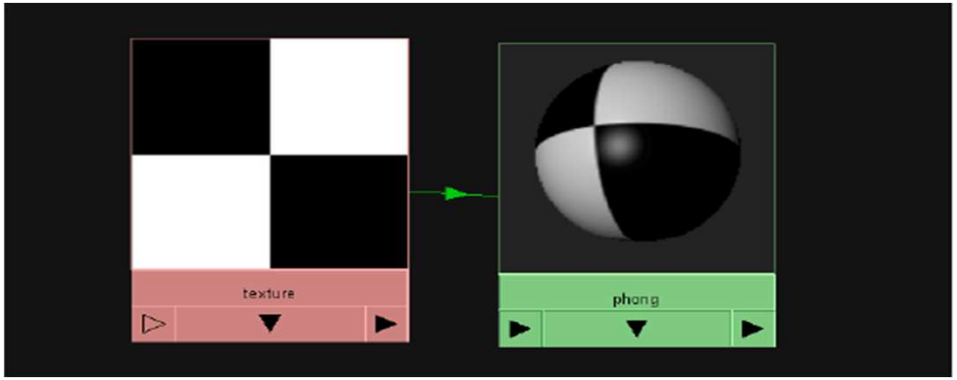
LINK – XoaX: Basic Local Illumination

## Shaders for Shading

- Shading trees
  - shaders are collections of nodes in a graph
  - visual representation
  - used in all the modelling software

- Shading languages
  - shaders are programs
  - programmer representation
    - much more flexible
  - used in productions by the more technically inclined
    - but disappearing since too hard to control

- Shading texture trees
  - Phong shading with a texture



## Polygon Shading

- Typically the constituent components of an object are reduced to triangular or planer quad polygons either:
  - prior to all rendering (real-time)
  - prior to a render (pre-render and real-time)
  - during a render (pre-render)
  - procedurally (pre-render)

- Polygons approximate 3d Shape

- There are many different types of polygon shading, including:
  - Flat
  - Gourard
  - Phong
  - Blinn

LINK - CS160_Fall07 Basic Shading

## Flat Shading

- Typically one illumination calculation per polygon
  - each pixel is assigned the same colour
  - usually computed for centroid of polygon

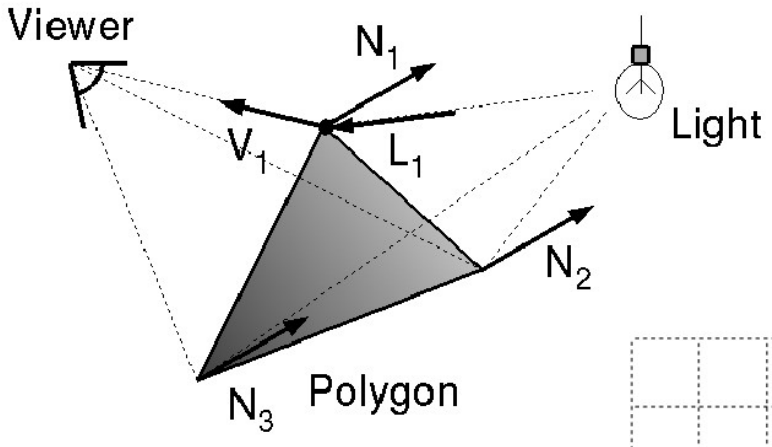
- Good for polyhedral objects, but
  - for point light sources the direction to light varies
  - for specular reflections the direction to eye varies

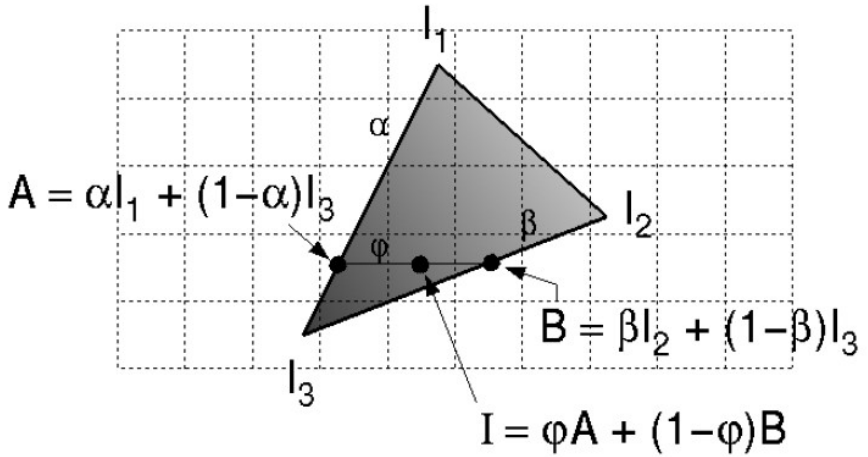
## Facet Shading

- Calculate illumination for each facet – the original polygons, or the reduced triangular or planer quad polygons
  - facets are still visible
  - assumes that polygons are not an approximation of a surface

# Gourard Shading

- Compute illumination for ONLY the vertices of a polygon:
  - use ONLY the vertex normals
  - linearly interpolate between these between vertices
  - bi-linearly interpolate colour between vertices down and across scan lines



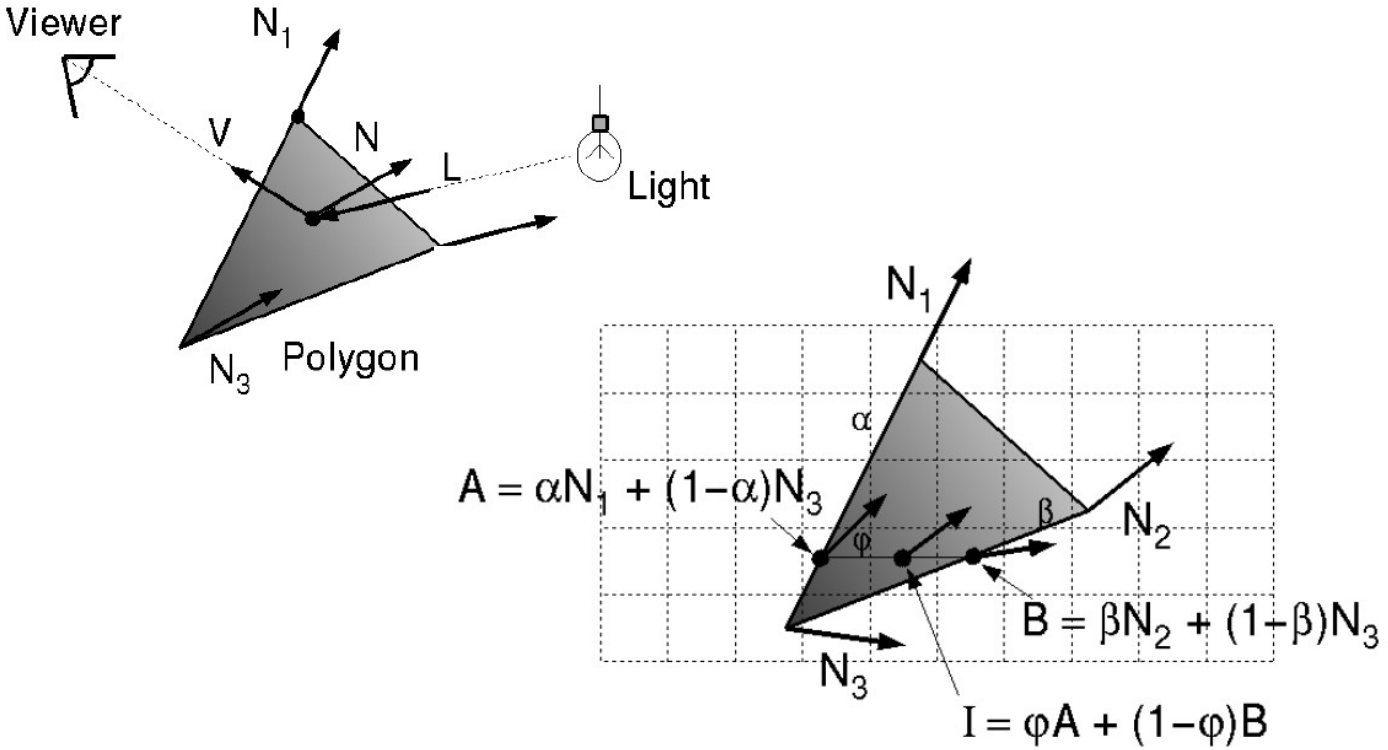




**RESULT**

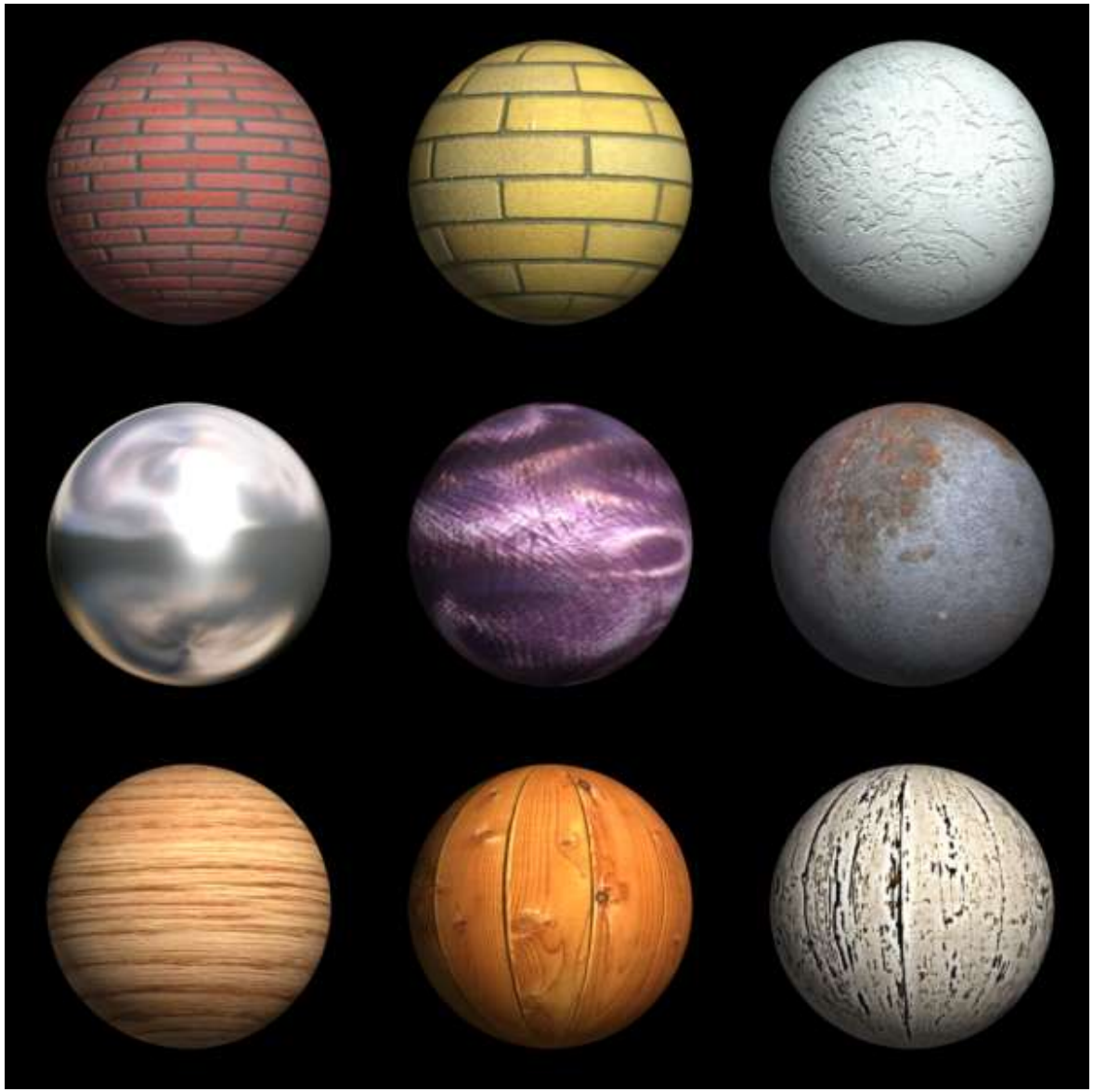
- creates smooth shading
- visible artefacts

## Phong Shading

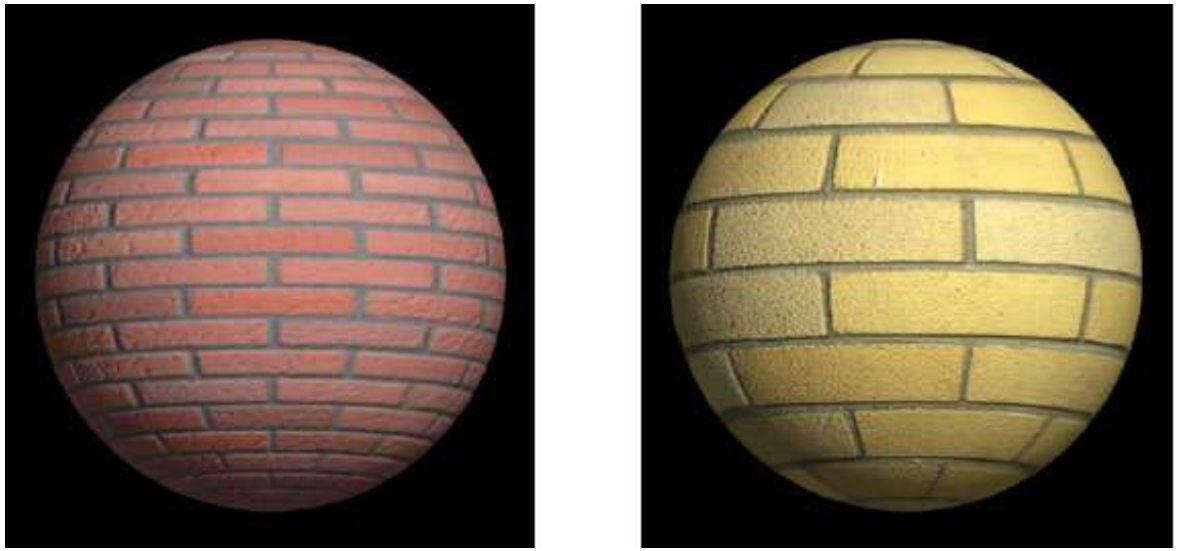
- One or more lighting calculations per pixel
  - linearly interpolate vertex normals across polygon
  - very smooth result
  - bi-linear interpolation of normals from vertices
  - not to be confused with Phong Illumination model



## Phong Shading – Lit and Textured

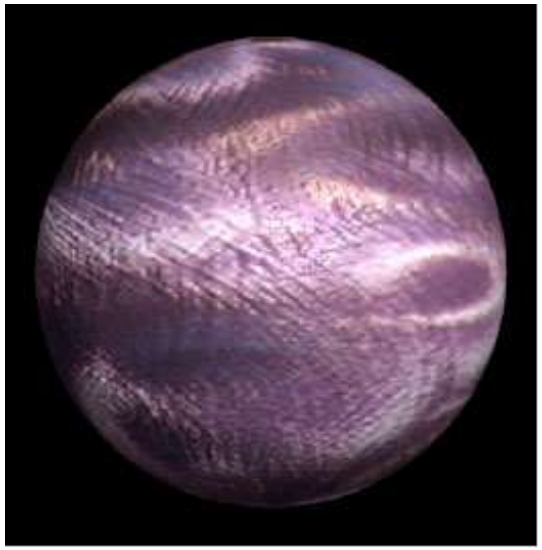


# Phong Shading – Colour



**Color** or **colour** (see spelling differences) is the visual perceptual property corresponding in humans to the categories called *red*, *yellow*, *blue* and others. Colour derives from the spectrum of light (distribution of light energy versus wavelength) interacting in the eye with the spectral sensitivities of the light receptors. Colour categories and physical specifications of colour are also associated with objects, materials, light sources, etc., based on their physical properties such as light absorption, reflection, or emission spectra. By defining a colour space, colours can be identified numerically by their coordinates.

# Phong Shading – Light Response



**Light** is electromagnetic radiation, particularly radiation of a wavelength that is visible to the human eye (about 400–700 nm, or perhaps 380–750 nm). In physics, the term *light* sometimes refers to electromagnetic radiation of any wavelength, whether visible or not.

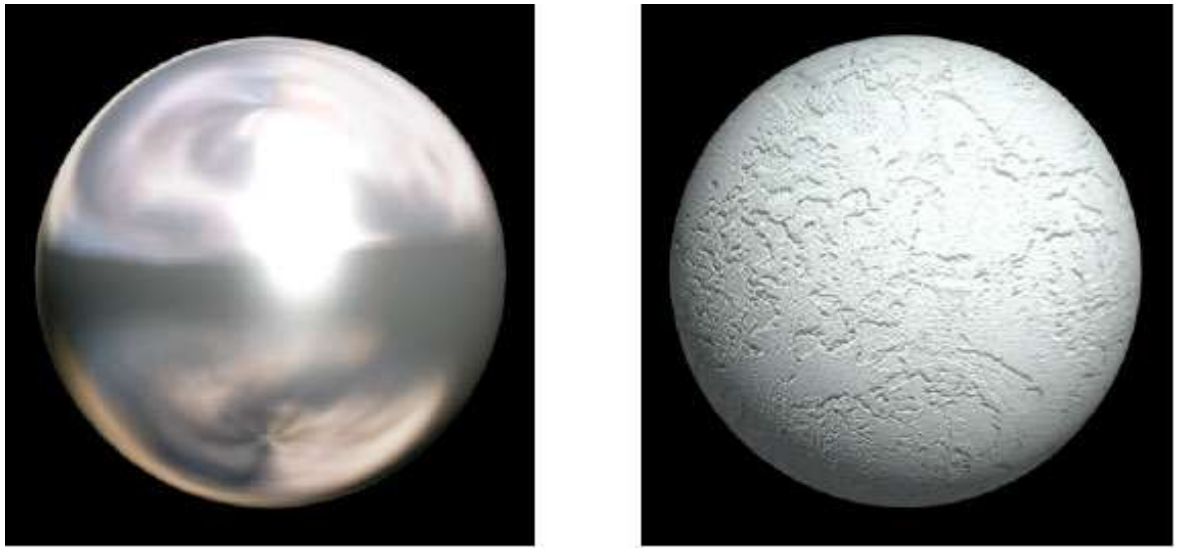
**Light**, which exists in tiny "packets" called photons, exhibits properties of both waves and particles. This property is referred to as the wave–particle duality. The study of light, known as optics, is an important research area in modern physics.

**Light Response** is the reflection/refraction/etc of light from an uneven or granular surface (assuming that there is no such thing as a perfectly smooth surface!).

## Phong Shading – Texture

**Texture mapping** is a method for adding detail, surface texture (a bitmap or raster image), or colour to a computer-generated graphic or 3D model. Its application to 3D graphics was pioneered by Dr Edwin Catmull in his Ph.D. thesis of 1974.

## Phong Shading – Roughness



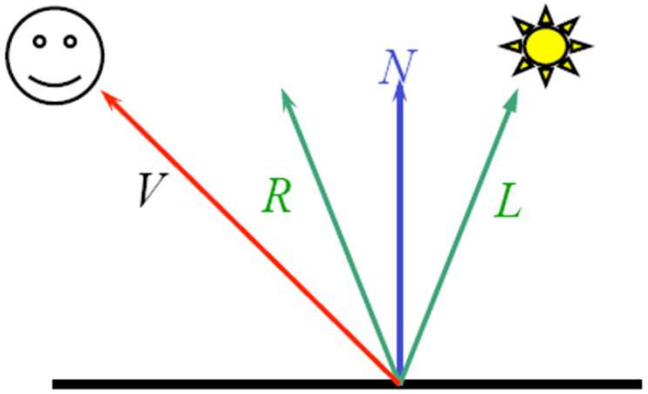
**Roughness** is a measure of the texture of a surface. It is quantified by the vertical deviations of a real surface from its ideal form. If these deviations are large, the surface is rough; if they are small the surface is smooth. Roughness is typically considered to be the high frequency, short wavelength component of a measured surface.

## Light Response

- Specifies how a surface reflects light
  - will be covered in more detail in this Lecture
  - often called Light reflection model

## Light Response – Phong Model

$$I = k_a + k_d \sum_l (N \cdot L) + k_s \sum_l (R \cdot V)^n$$




Typical formula for calculating light response (reflection) from a # of lights

Where Ka is the ambient material colour, Kd is the diffuse material colour, Ks is the specular material colour, *N* is the normal, *L* is the normalized direction to the light, R is the normalised reflected direction of the light, V is the eye/surface vector (must also be normalised), n is the specular power and l is the number of the lights.

What's the problem with this? Well, it's not energy conserving. In itself, this isn't really a problem since we don't calculate multiple bounces of light in games, so we're not adding energy to the scene as light bounces around like would happen in a ray tracer.

## Light Response – Phong Model

What's the problem with this?

Well, it's not energy conserving!

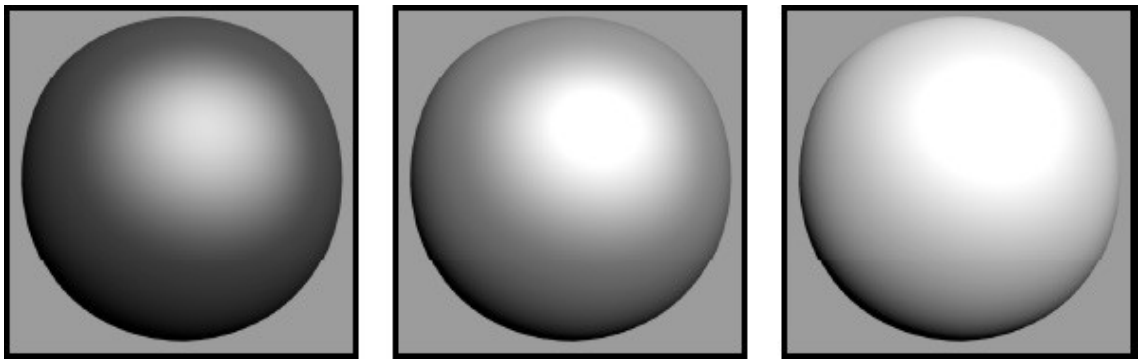
In itself, this isn't really a problem since we don't calculate multiple bounces of light in games, so we're not adding energy to the scene as light bounces around like would happen in a ray tracer.

.... but it is wrong

We will discuss BRDF later in this lecture

## Light Response – Diffuse

$$I = k_a + k_d \sum_l (N \cdot L) + k_s \sum_l (R \cdot V)^n$$



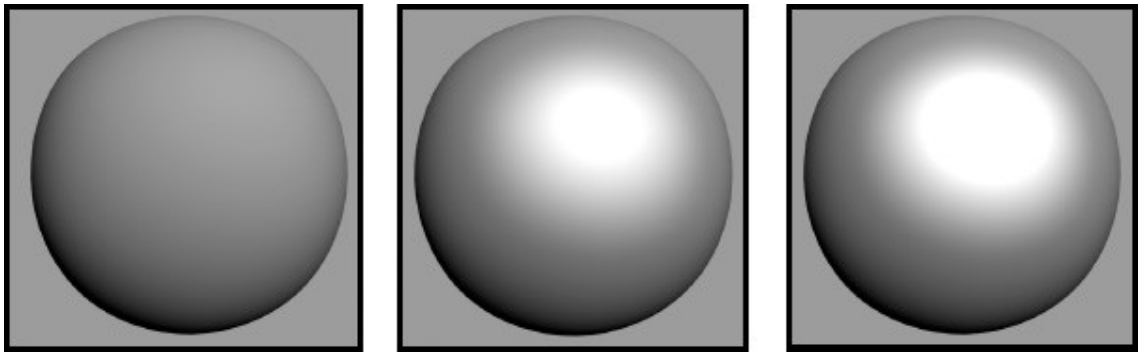
Typical formula for calculating light response (reflection) from a # of lights

Where Ka is the ambient material colour, Kd is the diffuse material colour, Ks is the specular material colour, *N* is the normal, *L* is the normalized direction to the light, R is the normalised reflected direction of the light, V is the eye/surface vector (must also be normalised), n is the specular power and l is the number of the lights.

What's the problem with this? Well, it's not energy conserving. In itself, this isn't really a problem since we don't calculate multiple bounces of light in games, so we're not adding energy to the scene as light bounces around like would happen in a ray tracer.

# Light Response – Specular

$$I = k_a + k_d \sum_l (N \cdot L) + k_s \sum_l (R \cdot V)^n$$



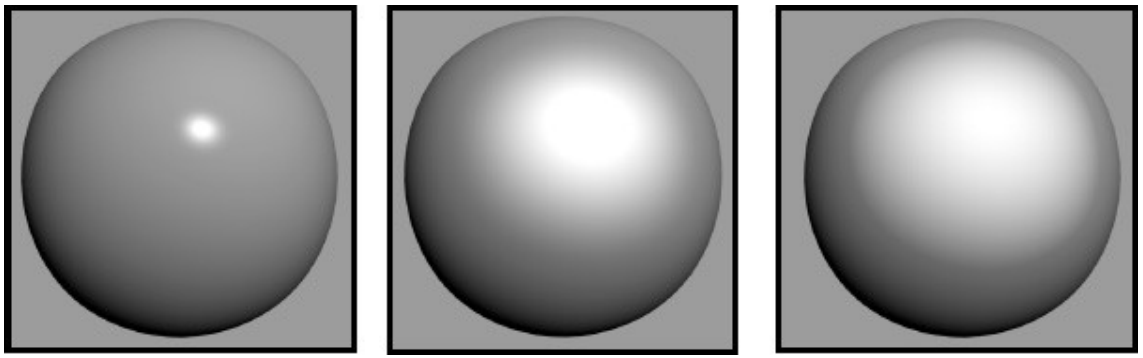
Typical formula for calculating light response (reflection) from a # of lights

Where Ka is the ambient material colour, Kd is the diffuse material colour, Ks is the specular material colour, *N* is the normal, *L* is the normalized direction to the light, R is the normalised reflected direction of the light, V is the eye/surface vector (must also be normalised), n is the specular power and l is the number of the lights.

What's the problem with this? Well, it's not energy conserving. In itself, this isn't really a problem since we don't calculate multiple bounces of light in games, so we're not adding energy to the scene as light bounces around like would happen in a ray tracer.

# Light Response – Power

$$I = k_a + k_d \sum_l (N \cdot L) + k_s \sum_l (R \cdot V)^n$$



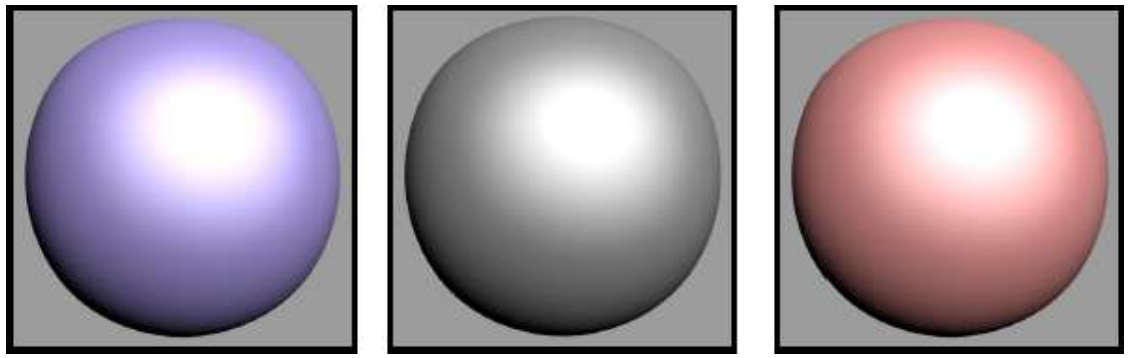
Typical formula for calculating light response (reflection) from a # of lights

Where Ka is the ambient material colour, Kd is the diffuse material colour, Ks is the specular material colour, *N* is the normal, *L* is the normalized direction to the light, R is the normalised reflected direction of the light, V is the eye/surface vector (must also be normalised), n is the specular power and l is the number of the lights.

What's the problem with this? Well, it's not energy conserving. In itself, this isn't really a problem since we don't calculate multiple bounces of light in games, so we're not adding energy to the scene as light bounces around like would happen in a ray tracer.

# Light Response – Diffuse Colour

$$I = k_a + k_d \sum_l (N \cdot L) + k_s \sum_l (R \cdot V)^n$$



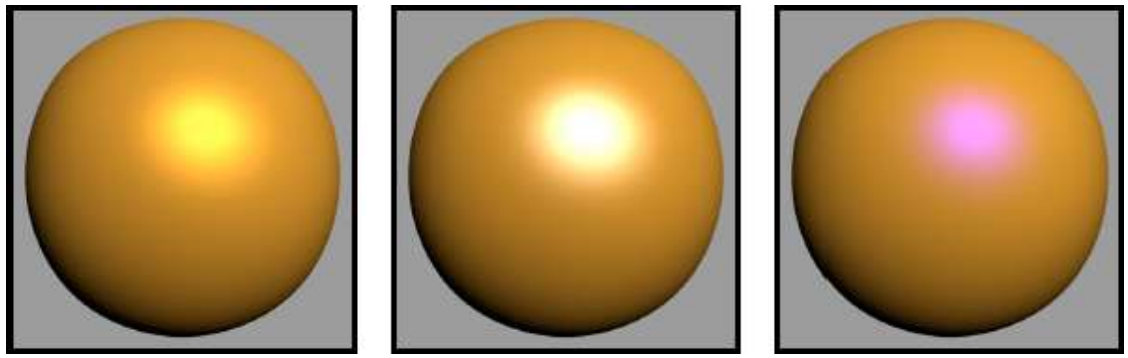
Typical formula for calculating light response (reflection) from a # of lights

Where Ka is the ambient material colour, Kd is the diffuse material colour, Ks is the specular material colour, *N* is the normal, *L* is the normalized direction to the light, R is the normalised reflected direction of the light, V is the eye/surface vector (must also be normalised), n is the specular power and l is the number of the lights.

What's the problem with this? Well, it's not energy conserving. In itself, this isn't really a problem since we don't calculate multiple bounces of light in games, so we're not adding energy to the scene as light bounces around like would happen in a ray tracer.

# Light Response – Specular Colour

$$I = k_a + k_d \sum_l (N \cdot L) + k_s \sum_l (R \cdot V)^n$$



Typical formula for calculating light response (reflection) from a # of lights

Where Ka is the ambient material colour, Kd is the diffuse material colour, Ks is the specular material colour, *N* is the normal, *L* is the normalized direction to the light, R is the normalised reflected direction of the light, V is the eye/surface vector (must also be normalised), n is the specular power and l is the number of the lights.

What's the problem with this? Well, it's not energy conserving. In itself, this isn't really a problem since we don't calculate multiple bounces of light in games, so we're not adding energy to the scene as light bounces around like would happen in a ray tracer.

# Phong Normalization Factor derivation

I'll do pure-specular only (i.e. $C_d = 0$, $C_s = 1$), the mixed case is easy from there. Also, we're only interested in the maximum of reflected energy, which in the Phong model occurs when $L$ and $N$ are parallel to each other, which makes $R = N$ too (in all other cases, $R$ is "on the other side" of $N$ relative to $V$, hence the angle between $R$ and $V$ can never be smaller than the angle between $R$ and $N$). Anyway, this means that $R \cdot V = N \cdot V$, which is a value we already know, namely $\cos\theta$.

Moving on, the integral we now need to calculate is

$$\int_\Omega (\cos\theta)^n \mathrm{d}\omega \tag{1}$$

with $\Omega$ being the upper hemisphere; integrating in spherical coordinates, this is

$$\int_0^{2\pi}\int_0^{\pi/2} (\cos\theta)^n \sin\theta \,\mathrm{d}\theta\mathrm{d}\phi = 2\pi\int_0^{\pi/2} (\cos\theta)^n \sin\theta \,\mathrm{d}\theta =: 2\pi I_n \tag{2}$$

and using integration by parts with $f = (\cos\theta)^n$, $g' = \sin\theta$ on $I_n$ we get

$$\begin{aligned} I_n &= [(\cos\theta)^n(-\cos\theta)]_0^{\pi/2} - \int_0^{\pi/2} n(\cos\theta)^{n-1}(-\sin\theta)(-\cos\theta)\,\mathrm{d}\theta \\ &= \left[-(\cos\theta)^{n+1}\right]_0^{\pi/2} - n\int_0^{\pi/2} (\cos\theta)^n \sin\theta\,\mathrm{d}\theta \\ &= (-0+1) - nI_n \end{aligned}$$

so $(n+1)I_n = 1$ which means that $I_n = \frac{1}{n+1}$. Plugging this into (2) tells us that (1) equals $\frac{2\pi}{n+1}$, so the normalization factor if we want it to integrate to 1 is the reciprocal, which is $\frac{n+1}{2\pi}$.

Why $\frac{n+1}{2}$ and not $n+2$? Because this is the derivation for the original Phong formulation, where the $R\cdot V$ term is not multiplied by $\cos\theta$. If you write that version of the Phong model as a BRDF, you end up with a $\cos\theta$ in the numerator to cancel out the $\cos\theta$ factor in the reflection equation. This numerator is complete nonsense physically, so the modern formulation of the Phong model removes it. Then the integral becomes

$$\int_\Omega (R\cdot V)\,\cos\theta\,\mathrm{d}\omega \overset{L=N}{=} \int_\Omega (\cos\theta)^{n+1}\,\mathrm{d}\omega$$

and our normalization factor computation cranks out $\frac{n+2}{2}$, as expected.

## Blinn-Phong normalization factor

I'll again limit myself to the specular term and again assume that the maximum reflected energy occurs with $L = N$ (I have no proof for the latter though, but I do have some experimental evidence. If I find a nice proof later, I'll update this document accordingly. Anyway, with $L = N$, things get a lot simpler than the general case because $L$, $N$, $V$, and $H$ all lie in the same plane and we can work exclusively with angles. Particularly, the angle $\theta_h$ between $H$ and $N$ is exactly half of the angle $\theta$ between $V$ and $N$, and the integral we need to evaluate boils down to

$$\int_\Omega (\cos\theta_h)^n \cos\theta \,\mathrm{d}\omega = \int_\Omega (\cos\theta/2)^n \cos\theta \,\mathrm{d}\omega$$

(I'll only do the BRDF version with the extra factor of $\cos\theta$ here). Again integrating in spherical coordinates, we get

$$\int_0^{2\pi}\int_0^{\pi/2} (\cos\theta/2)^n \cos\theta\sin\theta \,\mathrm{d}\theta\mathrm{d}\phi = 2\pi\int_0^{\pi/2} (\cos\theta/2)^n \cos\theta\sin\theta \,\mathrm{d}\theta \tag{3}$$

and using the half-angle formula $\cos(\theta/2) = \sqrt{\frac{1+\cos\theta}{2}}$ and the substitution $t = \cos\theta$ (which gives $\mathrm{d}t = -\sin\theta\,\mathrm{d}\theta$) we get

$$(3) = -2\pi\int_1^0 \left(\sqrt{\frac{1+t}{2}}\right)^n t\,\mathrm{d}t = 2\pi\int_0^1 \left(\frac{1+t}{2}\right)^{n/2} t\,\mathrm{d}t$$

which can be evaluated using integration by parts, this time using $f = t$ and $g' = ((1+t)/2)^{n/2}$. This yields:

$$\begin{aligned}
&2\pi\left(\left[\frac{4}{n+2}t\left(\frac{1+t}{2}\right)^{(n+2)/2}\right]_{t=0}^1 - \frac{4}{n+2}\int_0^1\left(\frac{1+t}{2}\right)^{(n+2)/2}\mathrm{d}t\right)\\
=&\frac{8\pi}{n+2}\left(\left[t\left(\frac{1+t}{2}\right)^{(n+2)/2}\right]_{t=0}^1 - \frac{4}{n+4}\left[\left(\frac{1+t}{2}\right)^{(n+4)/2}\right]_{t=0}^1\right)\\
=&\frac{8\pi}{n+2}\left[\cos(\theta)\cos(\theta/2)^{n+2} - \frac{4}{n+4}\cos(\theta/2)^{n+4}\right]_{\theta=\pi/2}^0\\
=&\frac{8\pi\left[\cos(\theta/2)^{n+2}\left(4\cos(\theta/2)^2 - (n+4)\cos(\theta)\right)\right]_{\theta=0}^{\pi/2}}{(n+2)(n+4)}\\
=&\frac{8\pi\left[\cos(\theta/2)^{n+2}\left(2(1+\cos(\theta)) - (n+4)\cos(\theta)\right)\right]_{\theta=0}^{\pi/2}}{(n+2)(n+4)}\\
=&\frac{8\pi\left[\cos(\theta/2)^{n+2}\left(2 - (n+2)\cos(\theta)\right)\right]_{\theta=0}^{\pi/2}}{(n+2)(n+4)}\\
=&\frac{8\pi\left(2^{1-(n+2)/2} - (2-(n+2))\right)}{(n+2)(n+4)}\\
=&\frac{8\pi(2^{-n/2}+n)}{(n+2)(n+4)}
\end{aligned}$$

which makes the Blinn-Phong normalization factor $\frac{(n+2)(n+4)}{8\pi(2^{-n/2}+n)}$, not $\frac{n+8}{8\pi}$.

# Implementation of Phong Shading in C++

```
/*
 *  phong.cpp
 *  asrTracer
 *
 *  Created by Petrik Clarberg on 2006-03-06.
 *  Copyright 2006 Lund University. All rights reserved.
 *
 */
#include "defines.h"
#include "intersection.h"
#include "phong.h"

using namespace std;
namespace asr {


/** Constructs a Phong material with diffuse color d,
*                    specular color s, and exponent (shininess) e,
*                    and (optionally) reflectivity r and transparency t,
*                    and index of refraction n.
*/
Phong::Phong(const Color& d, const Color& s, float e, float r, float t, float n)
: Material(r,t,n)
, mDiffColor(d)
, mSpecColor(s)
, mExponent(e)
{
}
```

# Implementation of Phong Shading in C++

```
/** Destructor. Does nothing. */
Phong::~Phong() { }

/** Returns the BRDF at the intersection is for the light direction L. */
Color Phong::getBRDF(const Intersection& is, const Vector& L)
{
                const Vector& N = is.mNormal;
                const Vector& V = is.mView;

                // Compute diffuse weight (kd)
                float kd = N * L;                                   // kd = dot(N,L)
                if(kd<0.0f) kd = 0.0f;                               // if kd<0 the light is on the back-side -> return black

                // Compute reflected view direction and specular weight (ks).
                Vector R = (2.0f*(N*V))*N - V;

                float ks = R * L;                                   // ks = dot(R,L)
                if(ks<0.0f) ks = 0.0f;                               // clip to 0 if negative
                ks = std::pow(ks, mExponent);                       // ks = ks^exponent

                return kd*mDiffColor + ks*mSpecColor;               // return weighted color (diffuse+specular)
}


} // namespace asr
```

# Implementation of Phong Shading in C++

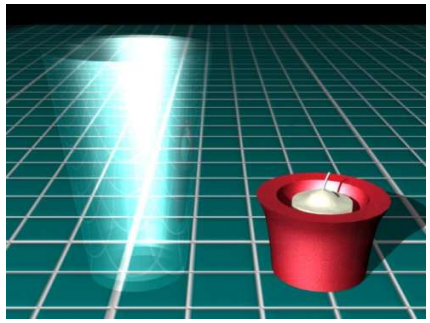
```
/*
 *  phong.h
 *  asrTracer
 *
 *  Created by Petrik Clarberg on 2006-03-06.
 *  Copyright 2006 Lund University. All rights reserved.
 *
 */
#ifndef __PHONG_H__
#define __PHONG_H__

#include "material.h"
namespace asr {

/** Class representing a simple phong material.
*                       The BRDF is computed at an intersection point by evaluating
*                       Phong's shading model. The material has a diffuse and a specular
*                       component, which can have different colors. The Phong exponent
*                       specifies the amount of shininess, ranging from perfectly diffuse
*                       to highly specular.
*/
class Phong : public Material
{
public:
Phong(const Color& d, const Color& s, float e, float r=0.0f, float t=0.0f, float n=1.0f);
virtual ~Phong();

Color getBRDF(const Intersection& is, const Vector& L);

protected:
Color mDiffColor;                       ///< The diffuse color.
Color mSpecColor;                                       ///< The specular color.
float mExponent;                        ///< The Phong exponent (shininess).
};

} // namespace asr
#endif // __PHONG_H__
```

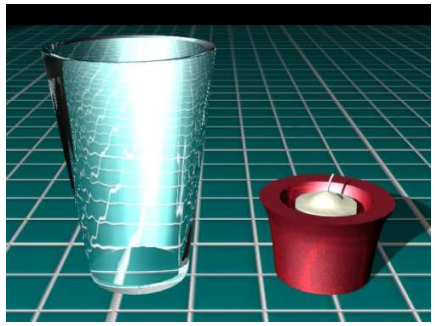
# Implementation of Phong Shading in Java

```
/**
 * Phong material.
 * @author fabio
 */
public class Phong extends Material {

/**
 * Diffuse color.
*/
public Color diffuse;

/**
 * Specular color.
 */
public Color specular;

/**
 * Specular exponent.
 */
public double exponent;
```

```
/**
 * Evaluate material for direct lighting.
 * @param N Surface normal.
 * @param L Light direction (point towards the light).
 * @param I View direction  (point towards the surface).
 */

public Color computeDirectLighting(Vec3 N, Vec3 L, Vec3 I) {
                double NdL = Math.max(N.dot(L),0);
                if(NdL > 0) {
                                Vec3 R = L.negate().reflect(N);
                                double RdV = Math.max(0,-R.dot(I));
                                return diffuse.scale(NdL).add(specular.scale(Math.pow(RdV,exponent)));
                } else {
                                return new Color(0,0,0);
                }
}

/**
 * Evaluate material mirror reflection.
 * @param N Surface normal.
 * @param I View direction  (point towards the surface).
 */
public Color computeReflection(Vec3 N, Vec3 I) {
                return new Color(0,0,0);
}

/**
 * True if this material has mirror reflections.
 * @param N Surface normal.
 * @param I View direction  (point towards the surface).
 */
public boolean hasReflection(Vec3 N, Vec3 I) {
                return false;
}
```

# Transparency





- Opacity coefficient k tells how much light is blocked

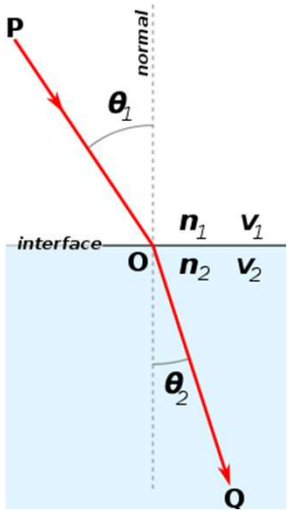
$$I = k\, I_{reflected} + (1 - k)\, I_{transmitted}$$

  - where k is 0 for translucent surface and 1 for opaque surface
  - where $I_{reflected}$ is intensity of reflected light
  - $I_{transmitted}$ is intensity of transmitted light from behind the surface

- Requires expansion of visible surface detection to access polygons further behind
  - see A Buffer, etc.

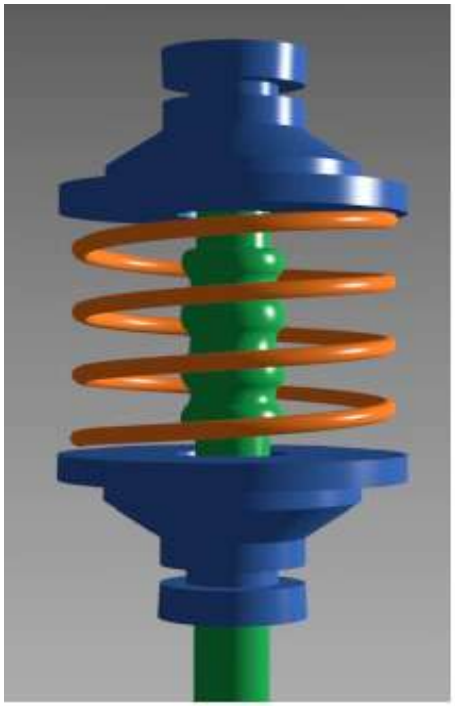
- **Snell's Law**
  - Refraction direction required for physically correct transparency computation

  – Refraction of light at the interface between two media of different refractive indices, with $n_2 > n_1$. Since the velocity is lower in the second medium ($v_2 < v_1$), the angle of refraction $\theta_2$ is less than the angle of incidence $\theta_1$; that is, the ray in the higher-index medium is closer to the normal.

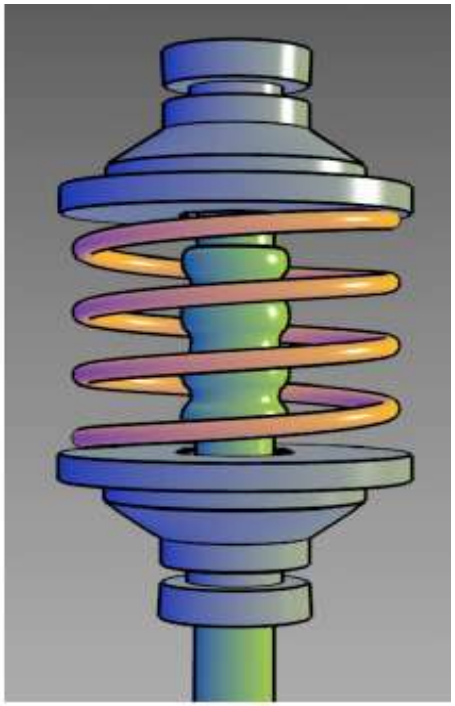


## Non-PhotoRealistic styles (NPR)

- Pixar/ILM look: a mixture of the real and NPR
  - realistic, physically-based building boxes
  - combined in non-realistic ways



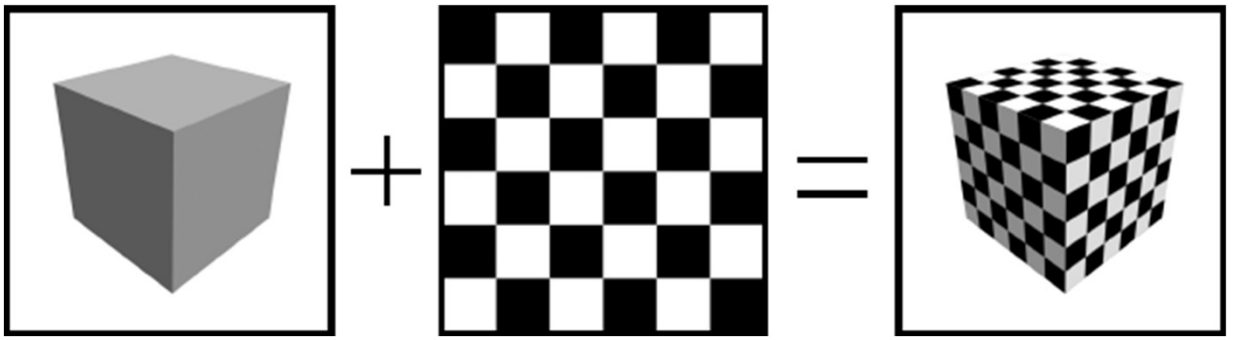
Phong

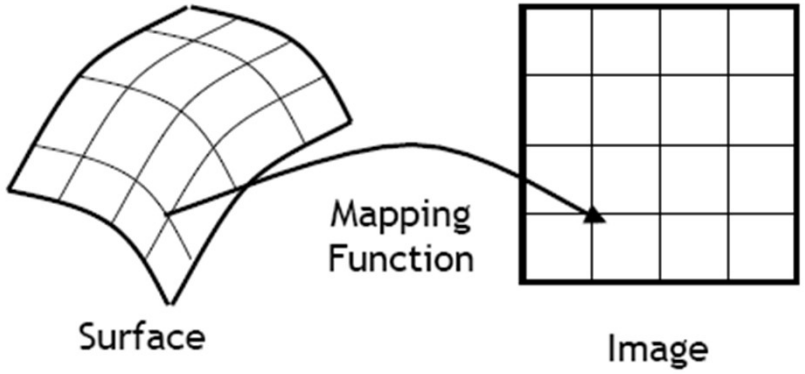


NPR

# Texture Mapping

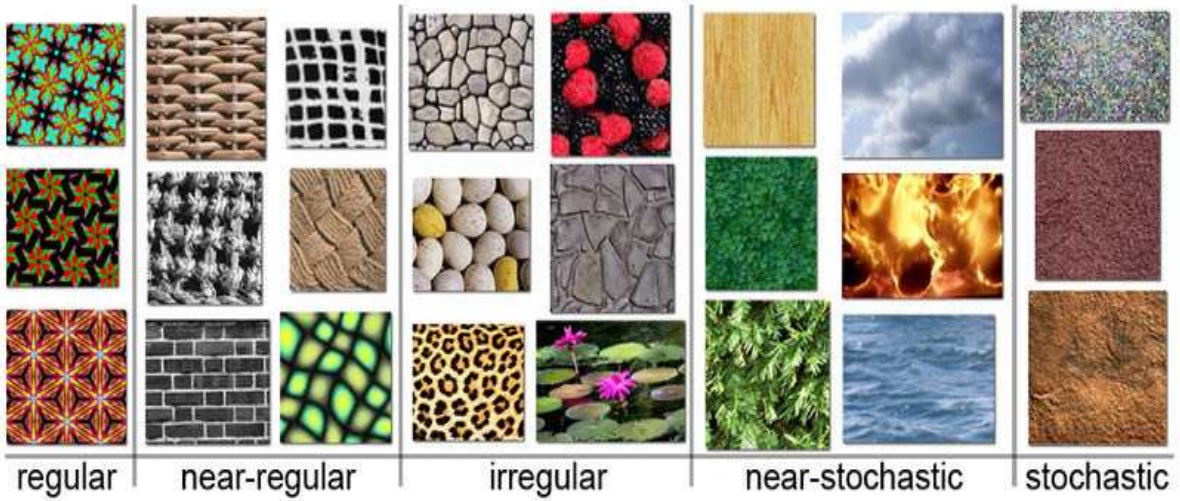
- In the real world properties vary across surface
- In CG we change shading parameters for each point on the surface
- Takes an image and apply it on to a surface
- Mapping function
  - for each point on the surface specifies a point in the image





# Texture Categorisation

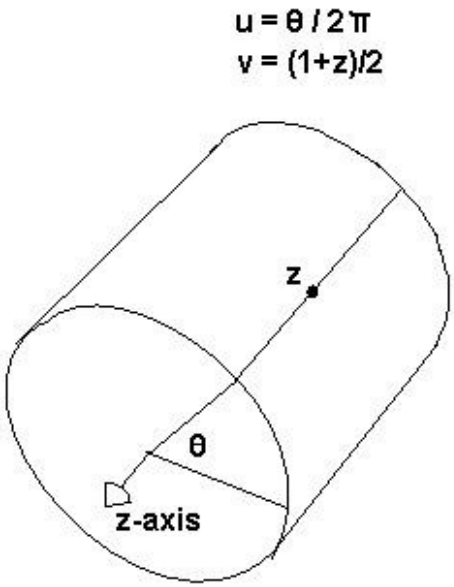
- Texture can be arranged along a spectrum going from stochastic to regular:
  - **Stochastic textures.** Texture images of stochastic textures look like noise: colour dots that are randomly scattered over the image, barely specified by the attributes minimum and maximum brightness and average colour. Many textures look like stochastic textures when viewed from a distance. An example of a stochastic texture is roughcast.
  - **Structured textures.** These textures look like somewhat regular patterns. An example of a structured texture is a stonewall or a floor tiled with paving stones.

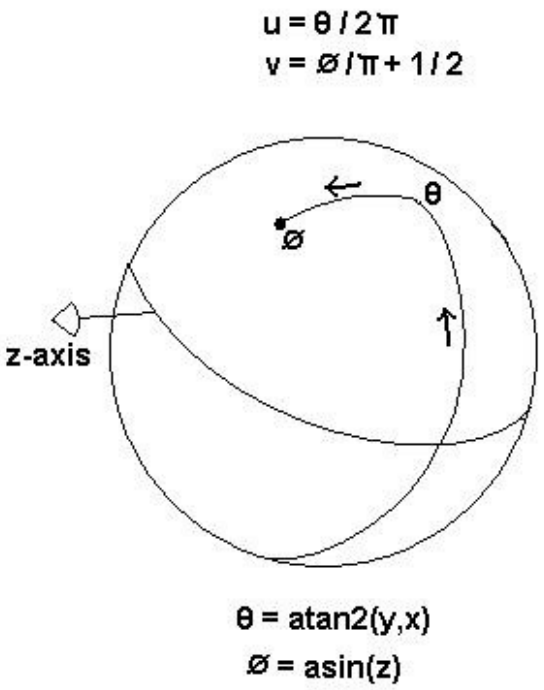


## Texture Projections

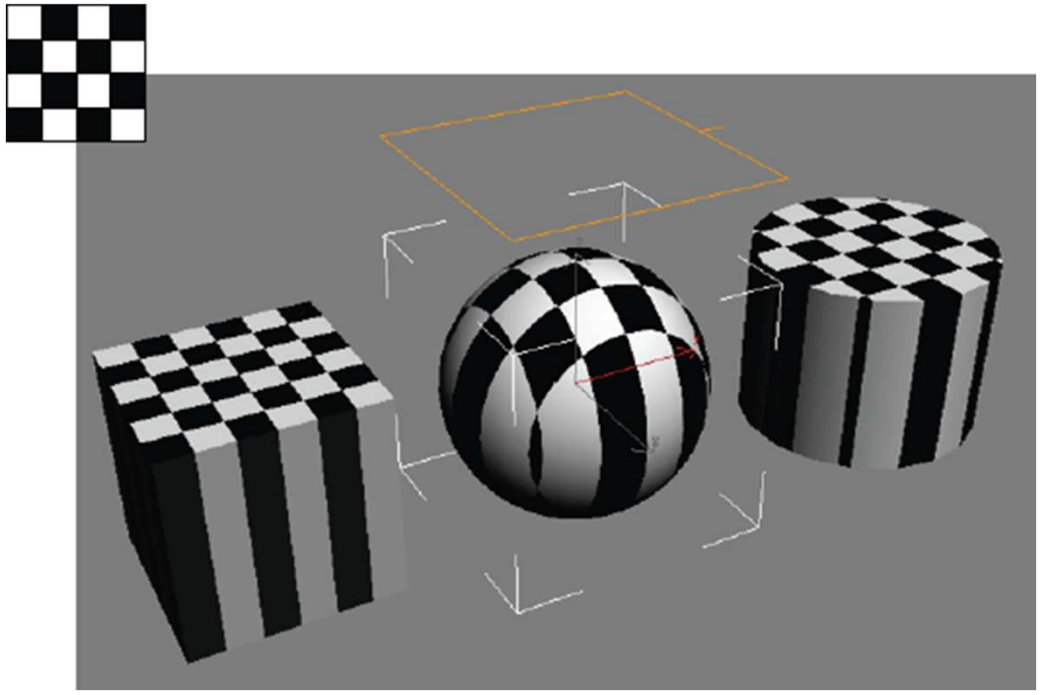
- Defines mapping by projection
  - projection follows simple shapes such as plane, box, cylinder, sphere

- Useful for man made object and simple shapes

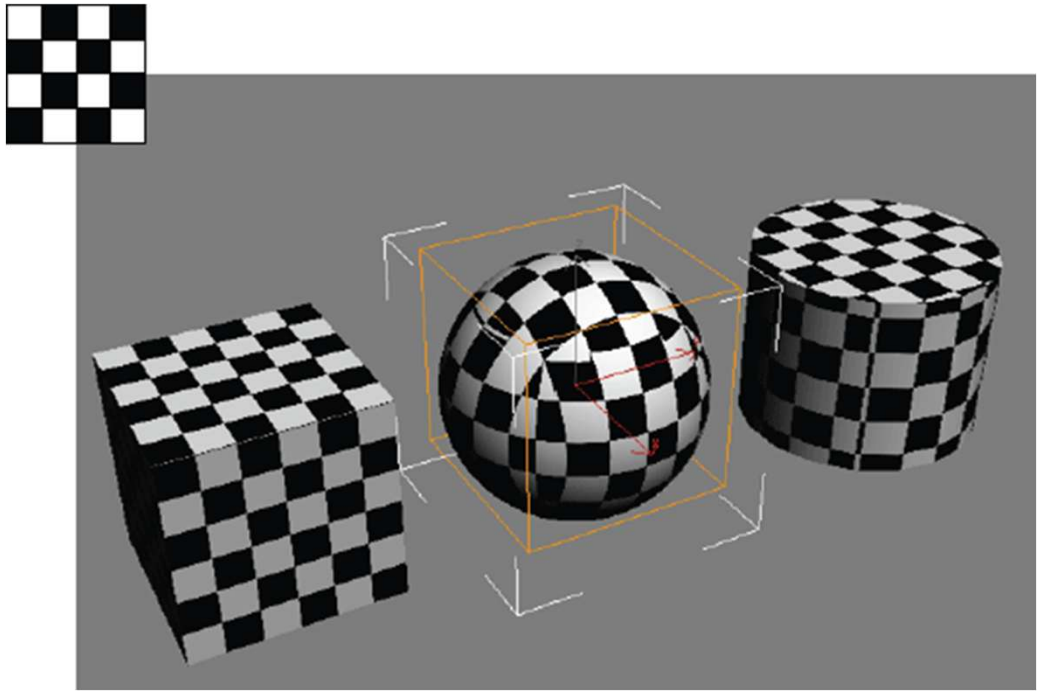


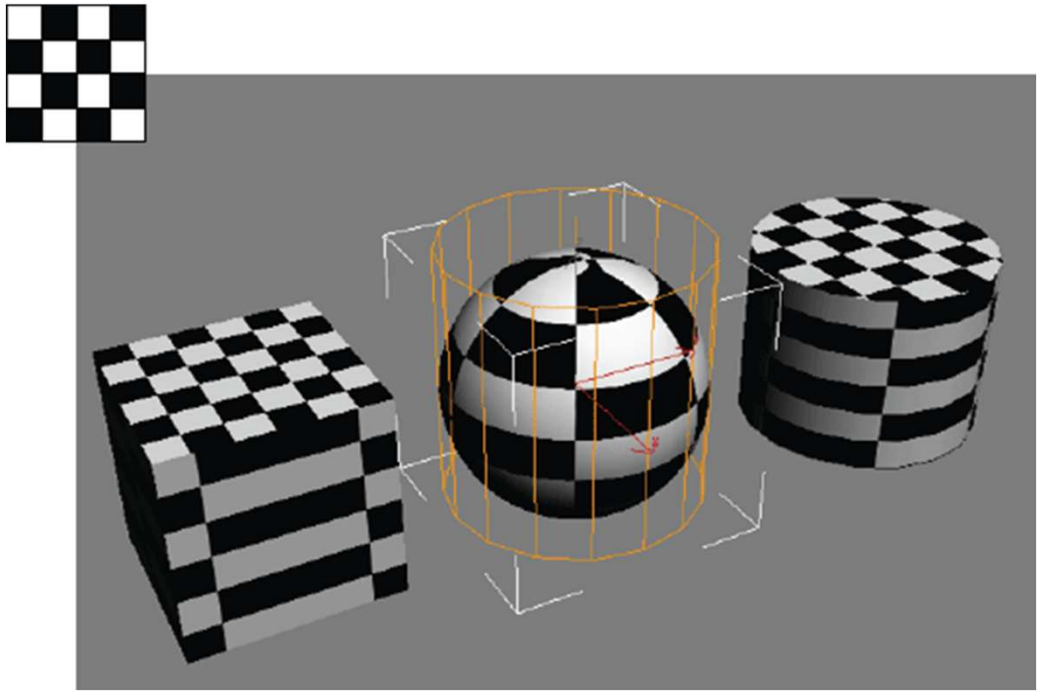
## Projections – Planer



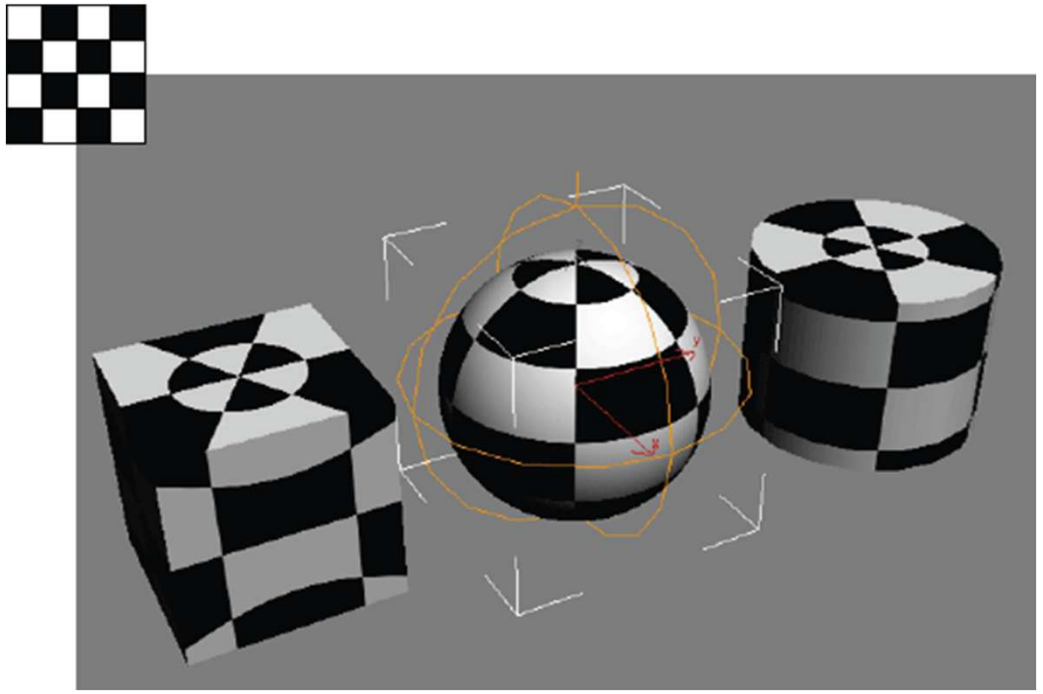
## Projections – Cubical



## Projections – Cylindrical



## Projections – Spherical



# Projection – Texture Reflection Mapping

- Any method of mapping three-dimensional points to a two-dimensional plane.

# BRDF

## (Physically Correct) Bidirectional Reflectance Distribution Functions

- The **bidirectional reflectance distribution function** (**BRDF**; $f_r(\omega_i,\omega_o)$) is a four-dimensional function that defines how light is reflected at an opaque surface.

- The function takes an incoming light direction, $\omega_i$, and outgoing direction, $\omega_o$, both defined with respect to the surface normal $n$, and returns the ratio of reflected radiance exiting along $\omega_o$ to the irradiance incident on the surface from direction $\omega_i$.

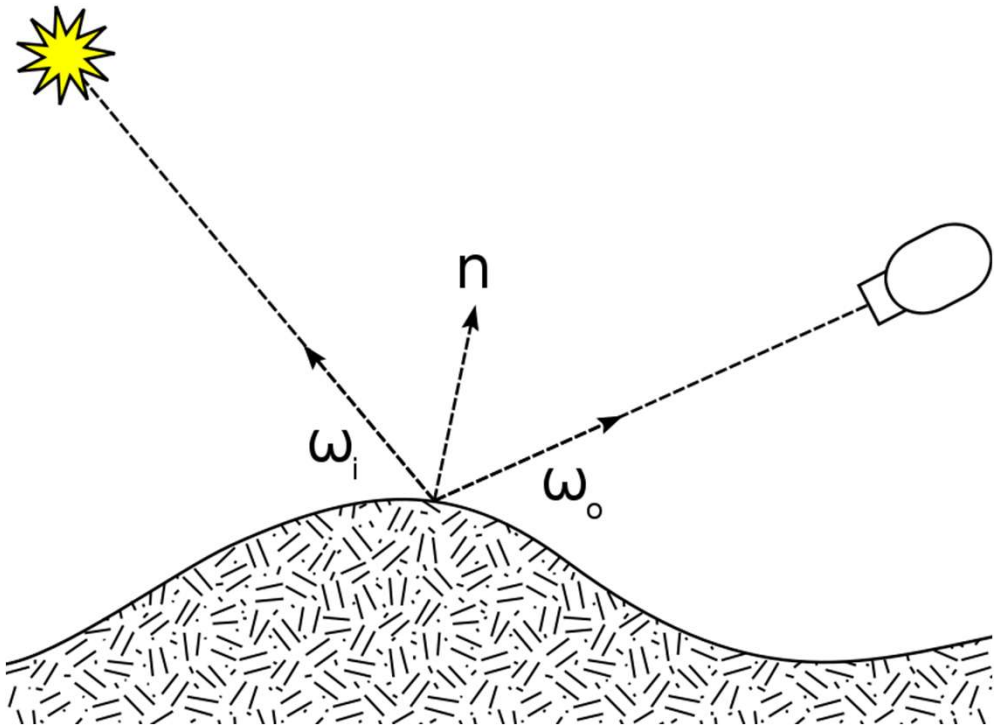
- Note that each direction $\omega$ is itself parameterized by azimuth angle $\phi$ and zenith angle $\theta$, therefore the BRDF as a whole is 4-dimensional.

LINK - BRDF based illumination model for Lacquerware

# BRDF

## (Physically Correct) Bidirectional Reflectance Distribution Functions

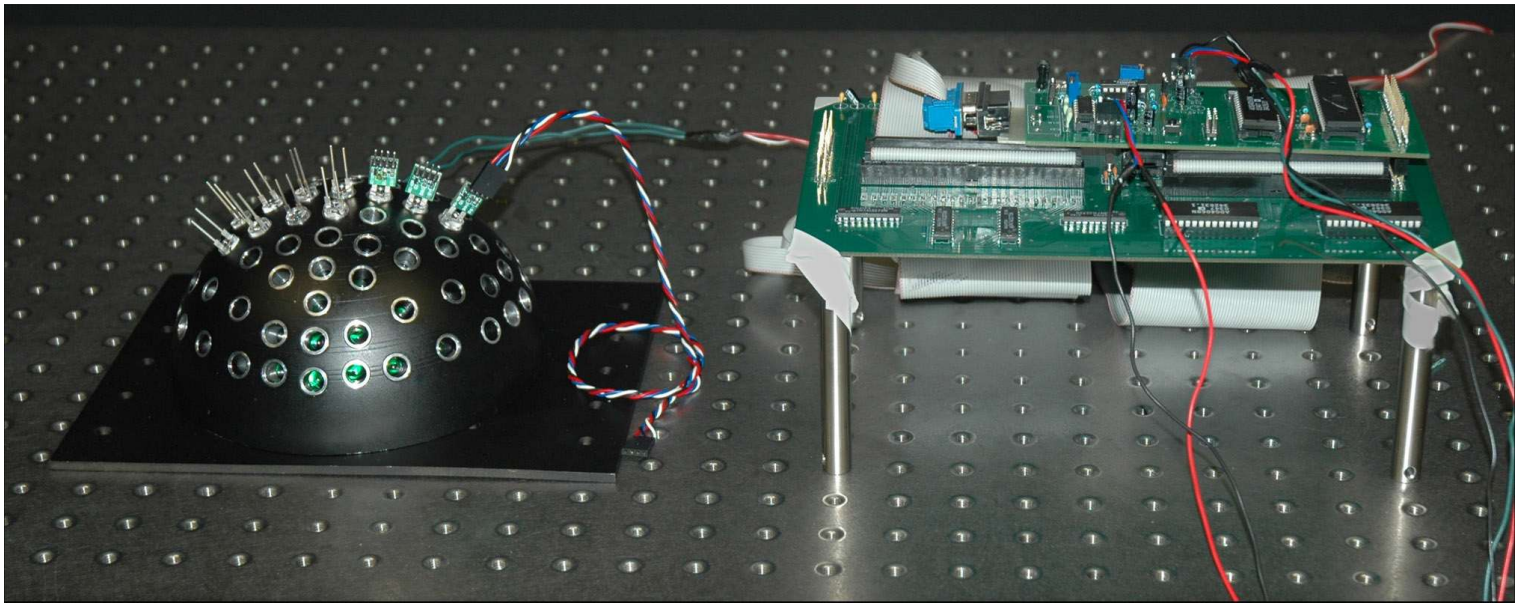
- The function takes an incoming light direction, $\omega_i$, and outgoing direction, $\omega_o$, both defined with respect to the surface normal *n*, and returns the ratio of reflected radiance exiting along $\omega_o$ to the irradiance incident on the surface from direction $\omega_i$.



# BRDF

## (Physically Correct) Bidirectional Reflectance Distribution Functions

- It can even be measured!!!



LINK - **Jiaping Wang** (http://www.jiapingwang.com/)

# BRDF

## Creating Physically Correct Bidirectional Reflectance Distribution Functions

- BRDF Shop has been created as an interface for the quick and intuitive development of arbitrary, but physically correct, bidirectional reflectance distribution functions, or BRDFs.

- The interface gives artists the ability to create a BRDF through positioning and manipulating highlights on a spherical canvas.

- The inventors have developed a novel mapping between painted highlights and specular lobes of an extended Ward BRDF model.

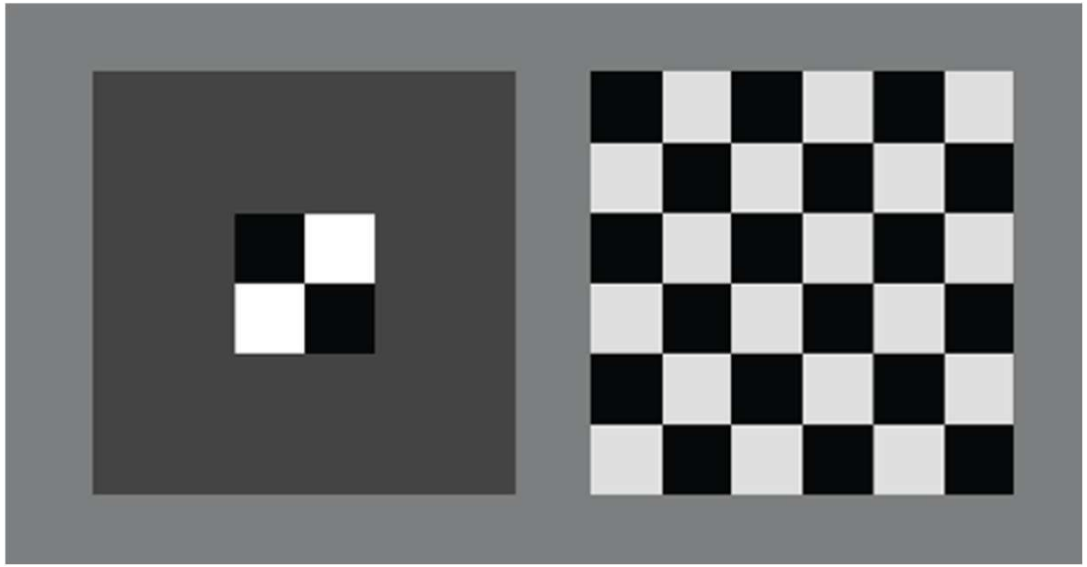
- The implementation of BRDF Shop uses programmable graphics hardware to provide a real-time visualization of the material on a complex object in environment lighting.

LINK – Demonstrations of BRDF-Shop - Part 1

LINK – Demonstrations of BRDF-Shop - Part 2

# Texture Tiling

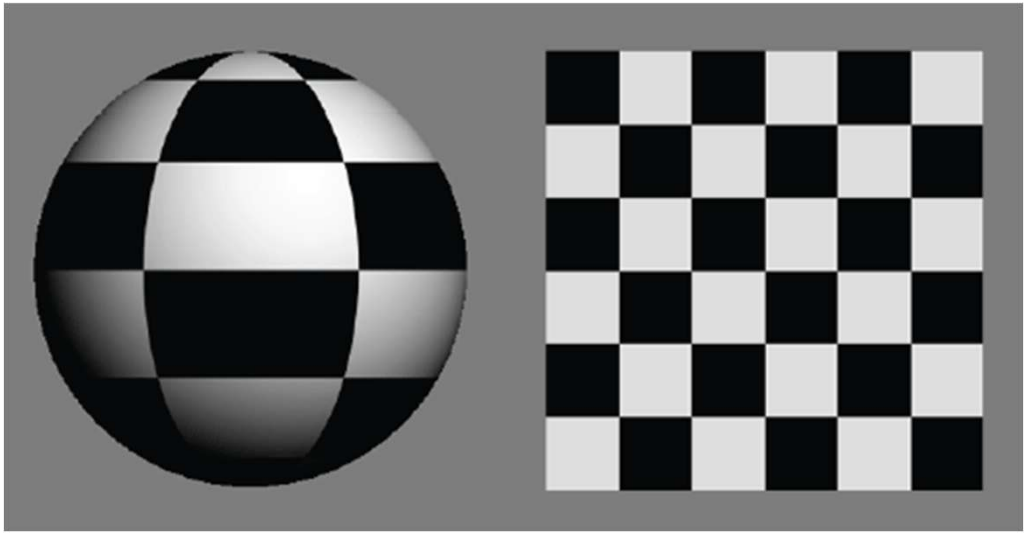
- Repeat texture when mapping is out of it



The simplest way to generate a large image from a sample image is to tile it. This means multiple copies of the sample are simply copied and pasted side by side. The result is rarely satisfactory. Except in rare cases, there will be the seams in between the tiles and the image will be highly repetitive.

## Texture Distortions

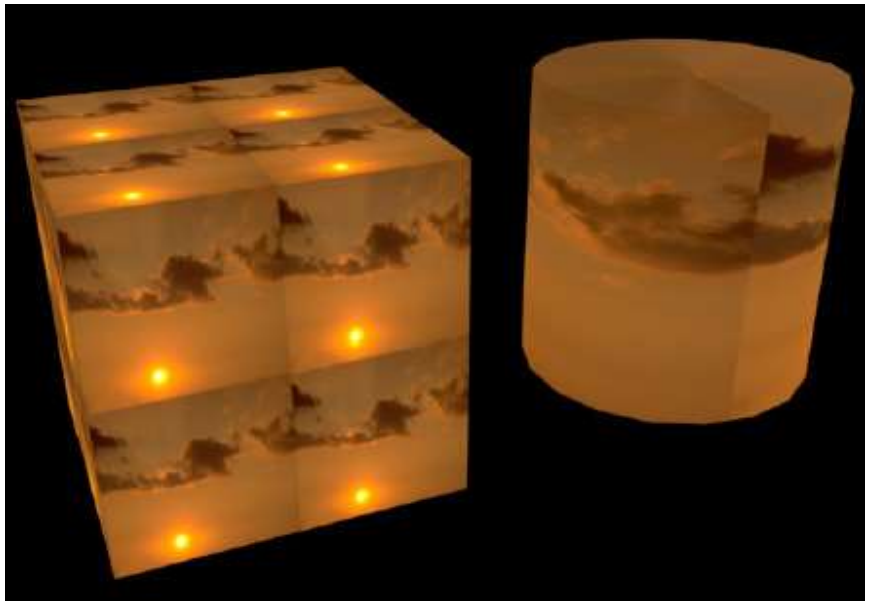
- Textures get minified and magnified on surfaces
  - always try to minimize this



The mapping of regular textures onto regular objects or shapes is a fundamentally flawed conceptual process.

# Texture Seams

- Seams are created by tiling or joining texture edges



The tiling of regular textures onto regular objects or shapes is a fundamentally flawed conceptual process.

This is best demonstrated by texture seams – where a visual discontinuity from non blended textures or texture processes generates unwanted aesthetic results.

## So:

- Texture projection and tiling create fundamental visual problems

- Seams and distortions are unavoidable
  - inherent in any form of mapping

- Mathematically speaking
  - seams originates from discontinuities in the mapping function
  - distortions originates from the behaviour of the second derivative of the mapping function

## Texture Quilting

- A new image is synthesized by stitching together small patches of existing images

LINK - Paper on "Image Quilting" by Efros and Freeman

## Texture Parameterization

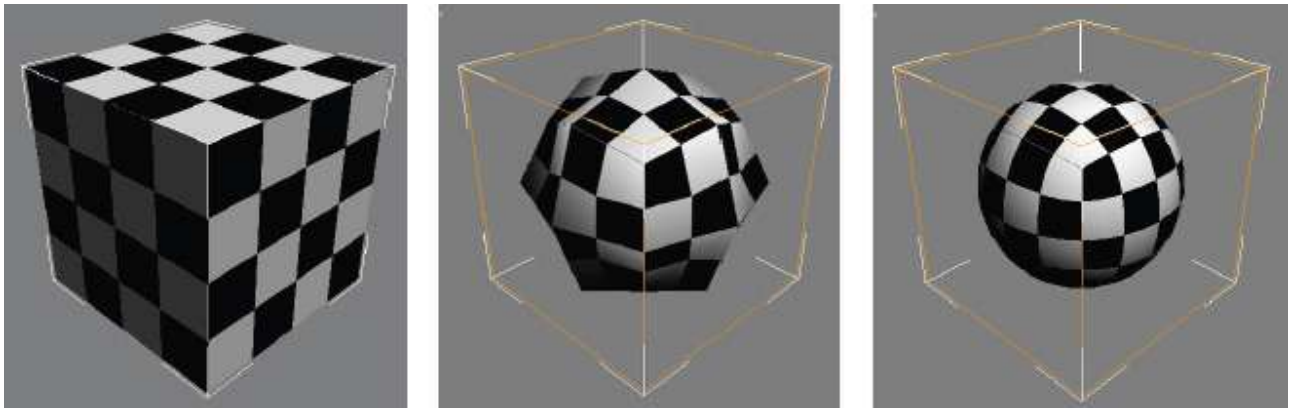
- Projections do not work if geometry if different

- Store the mapping function onto the geometry
  - store texture coordinates for each control point
  - interpolate along the surface
  - often called UV mapping

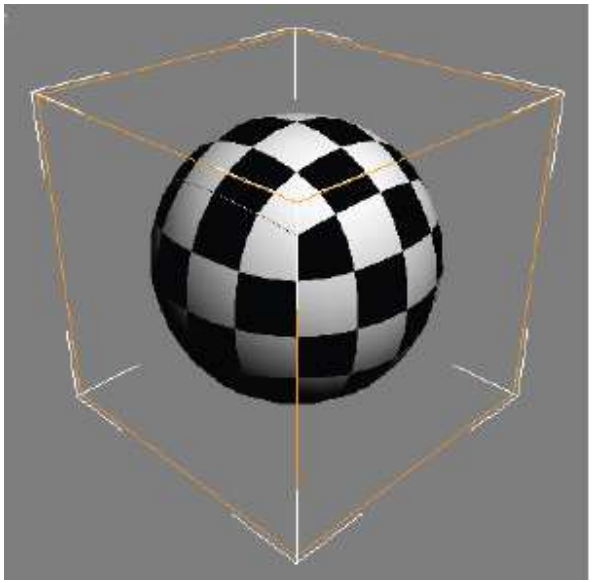
- Parameterization: process of assigning intrinsic texture coordinates onto objects

## Subdivision Surfaces and Texture Mapping

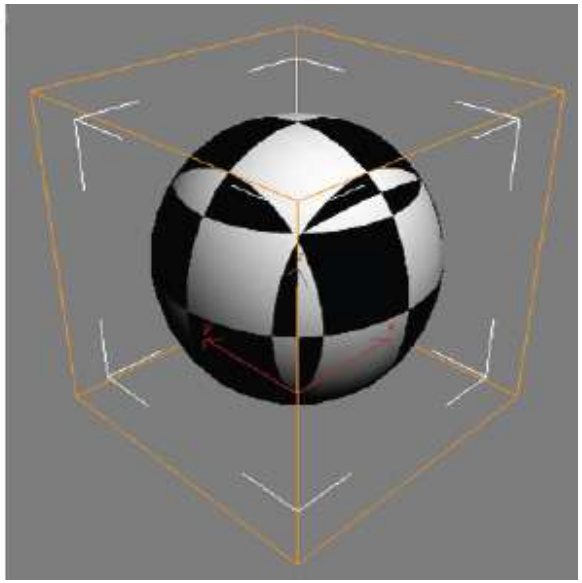
- Seams are created by tiling or joining texture edges



# Subdivision Surfaces and Texture Mapping

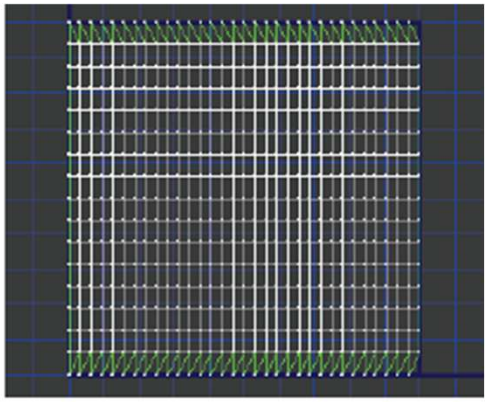


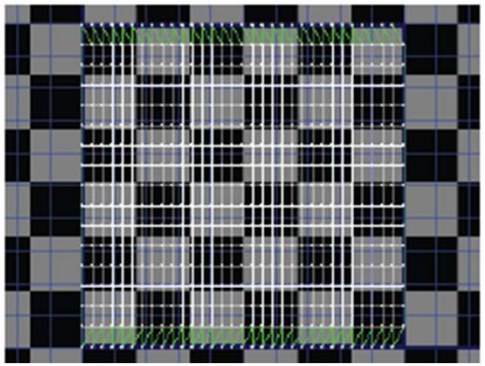
Parameterization



Projection

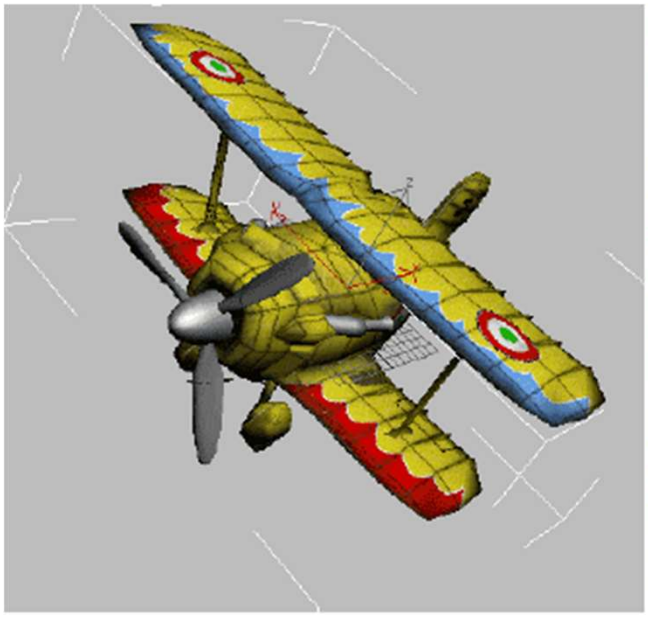
# Visualizing and editing UV Texture Maps

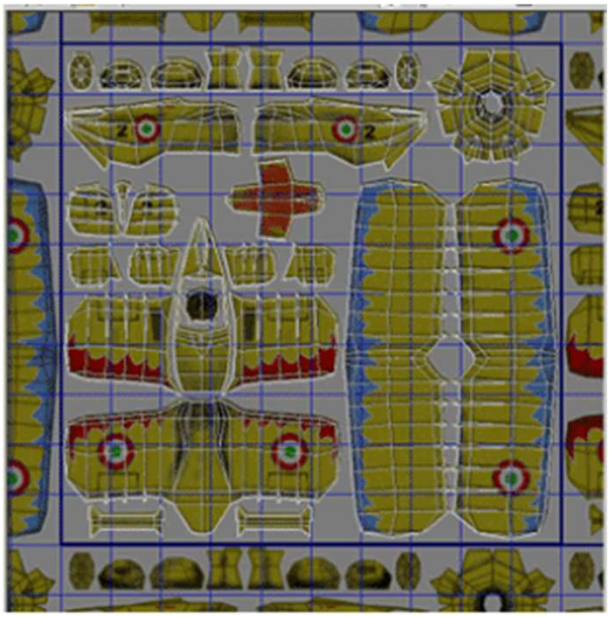





## Texture Atlas or Dictionary

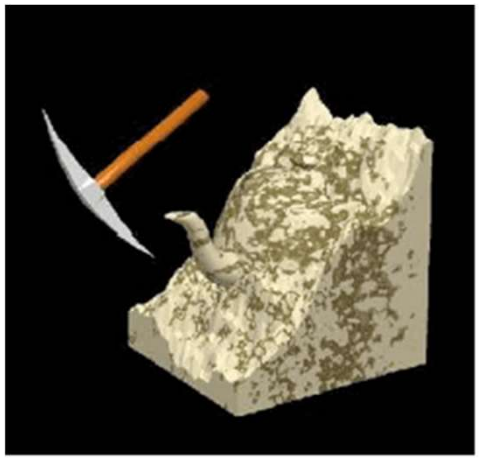
- breaks up model into a single texture





## Procedural Texturing

- Texture variations are generated algorithmically
  - user can tweak parameters that control texture look
  - often specified in 3d for solid materials



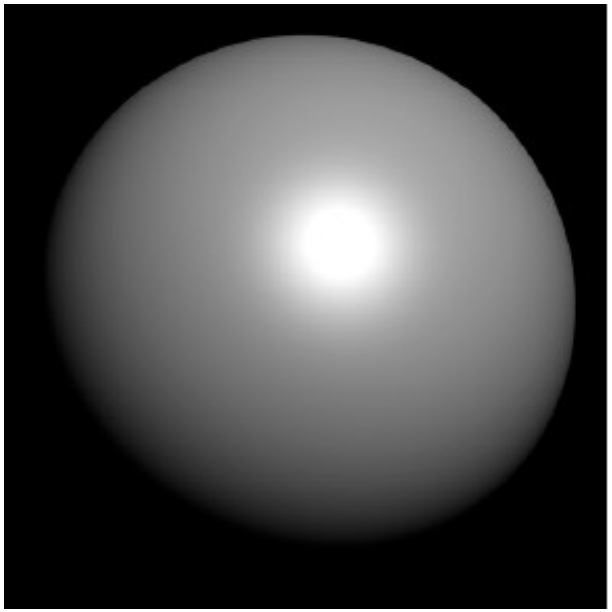
[Wolfe / SG97 Slide set]

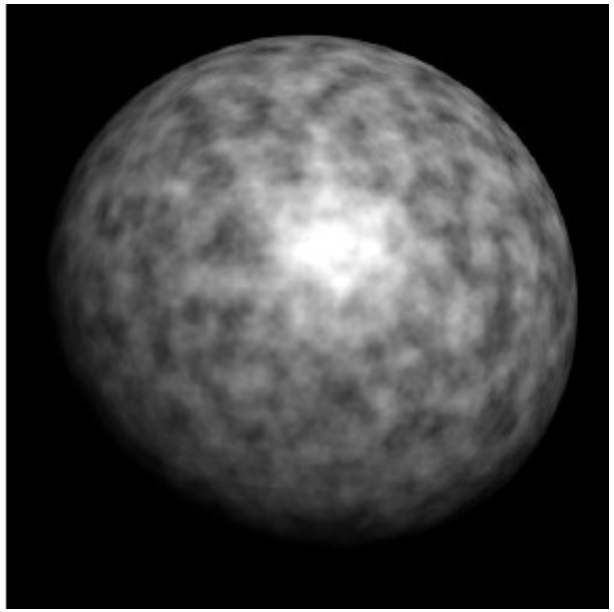
# Procedural Texturing

```
/* Copyrighted Pixar 1988 */
/* From the RenderMan Companion p.355 */
/* Listing 16.19  Blue marble surface shader*/

/*
 * blue_marble(): a marble stone texture in shades of blue
 * surface
 */

blue_marble(
      float   Ks    = .4,
              Kd    = .6,
              Ka    = .1,
              roughness = .1,
              txtscale = 1;
      color   specularcolor = 1)
{
  point PP;          /* scaled point in shader space */
  float csp;         /* color spline parameter */
  point Nf;          /* forward-facing normal */
  point V;           /* for specular() */
  float pixelsize, twice, scale, weight, turbulence;

  /* Obtain a forward-facing normal for lighting calculations. */
  Nf = faceforward( normalize(N), I);
  V = normalize(-I);

  /*
   * Compute "turbulence" a la [PERLIN85]. Turbulence is a sum of
   * "noise" components with a "fractal" 1/f power spectrum. It gives the
   * visual impression of turbulent fluid flow (for example, as in the
   * formation of blue_marble from molten color splines!). Use the
   * surface element area in texture space to control the number of
   * noise components so that the frequency content is appropriate
   * to the scale. This prevents aliasing of the texture.
   */
  PP = transform("shader", P) * txtscale;
  pixelsize = sqrt(area(PP));
  twice = 2 * pixelsize;
  turbulence = 0;
  for (scale = 1; scale > twice; scale /= 2)
     turbulence += scale * noise(PP/scale);

/* Gradual fade out of highest-frequency component near limit */
  if (scale > pixelsize) {
     weight = (scale / pixelsize) - 1;
     weight = clamp(weight, 0, 1);
     turbulence += weight * scale * noise(PP/scale);
  }

  /*
   * Magnify the upper part of the turbulence range 0.75:1
   * to fill the range 0:1 and use it as the parameter of
   * a color spline through various shades of blue.
   */
  csp = clamp(4 * turbulence - 3, 0, 1);
  Ci = color spline(csp,
  color (0.25, 0.25, 0.35),     /* pale blue       */
     color (0.25, 0.25, 0.35),  /* pale blue       */
     color (0.20, 0.20, 0.30),  /* medium blue     */
     color (0.20, 0.20, 0.30),  /* medium blue     */
     color (0.20, 0.20, 0.30),  /* medium blue     */
     color (0.25, 0.25, 0.35),  /* pale blue       */
     color (0.25, 0.25, 0.35),  /* pale blue       */
     color (0.15, 0.15, 0.26),  /* medium dark blue */
     color (0.15, 0.15, 0.26),  /* medium dark blue */
     color (0.10, 0.10, 0.20),  /* dark blue       */
     color (0.10, 0.10, 0.20),  /* dark blue       */
     color (0.25, 0.25, 0.35),  /* pale blue       */
     color (0.10, 0.10, 0.20)   /* dark blue       */
     );

  /* Multiply this color by the diffusely reflected light. */
  Ci *= Ka*ambient() + Kd*diffuse(Nf);

  /* Adjust for opacity. */
  Oi = Os;
  Ci = Ci * Oi;

  /* Add in specular highlights. */
  Ci += specularcolor * Ks * specular(Nf,V,roughness);
}
```

## Procedural Diffuse Texture Maps

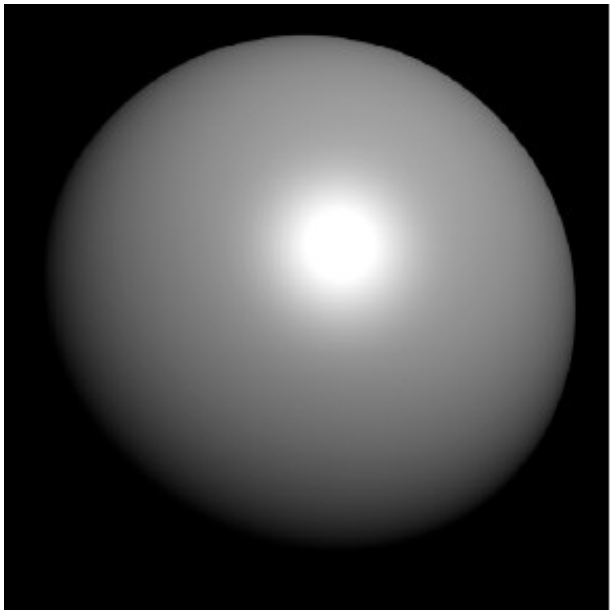
- Variations to diffuse colour

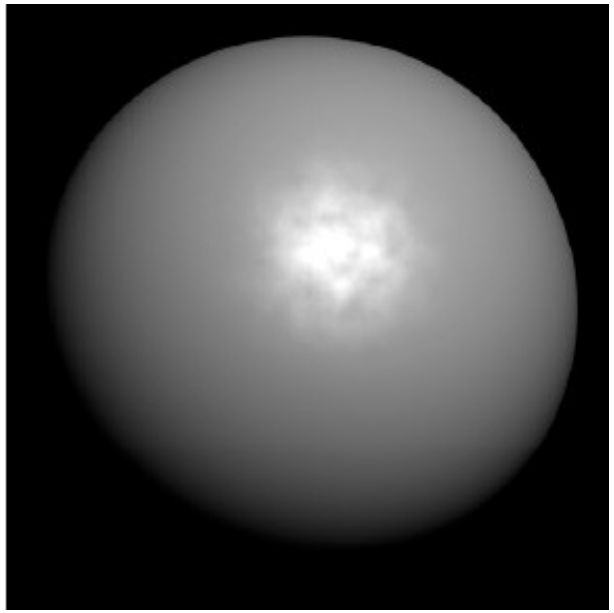




## Procedural Specular Texture Maps

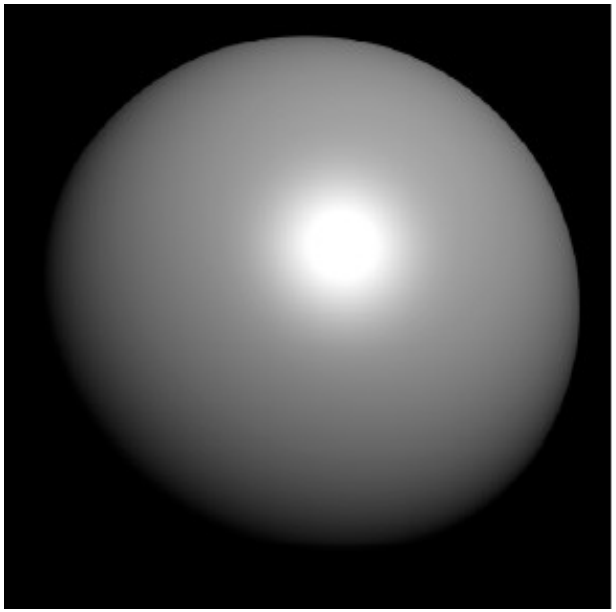
- Variation of the highlights

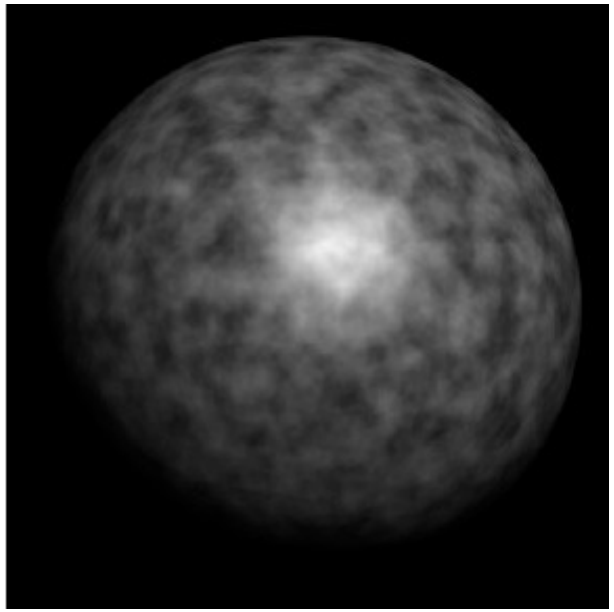




## Procedural Transparency Texture Maps

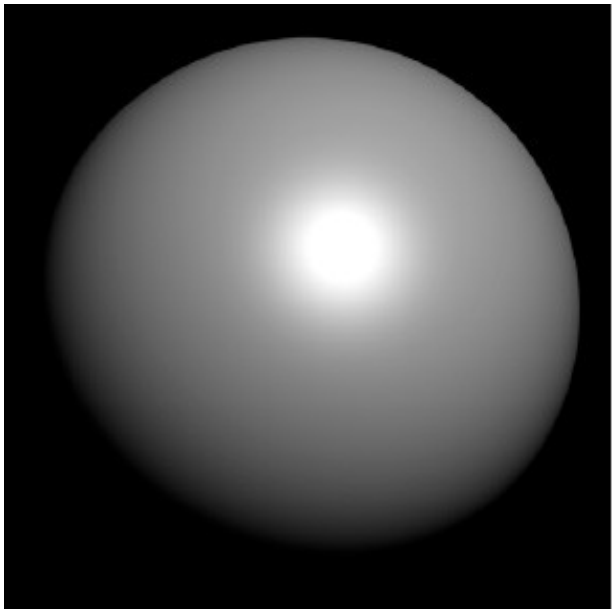
- Variation of object transparency

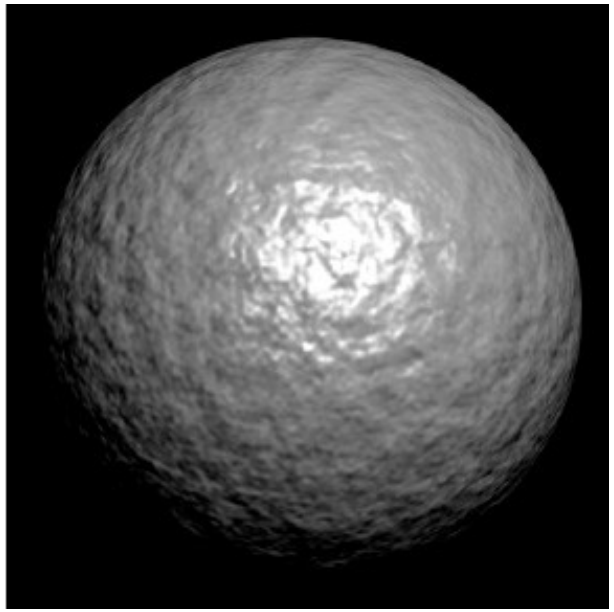




## Bump maps

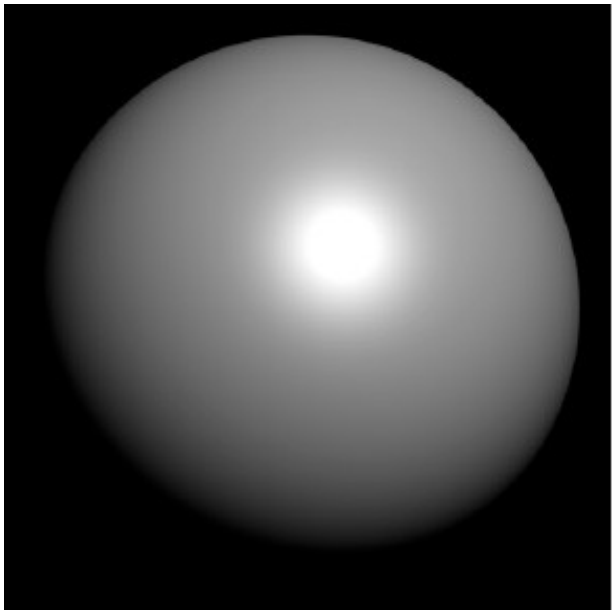
- Variations of surface normals

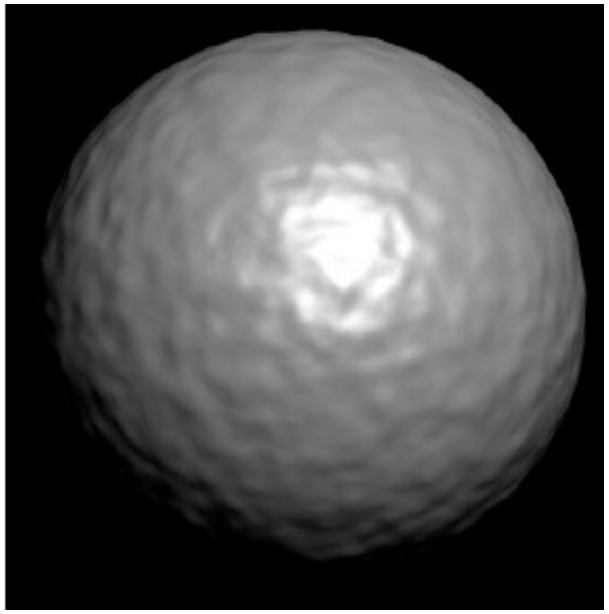




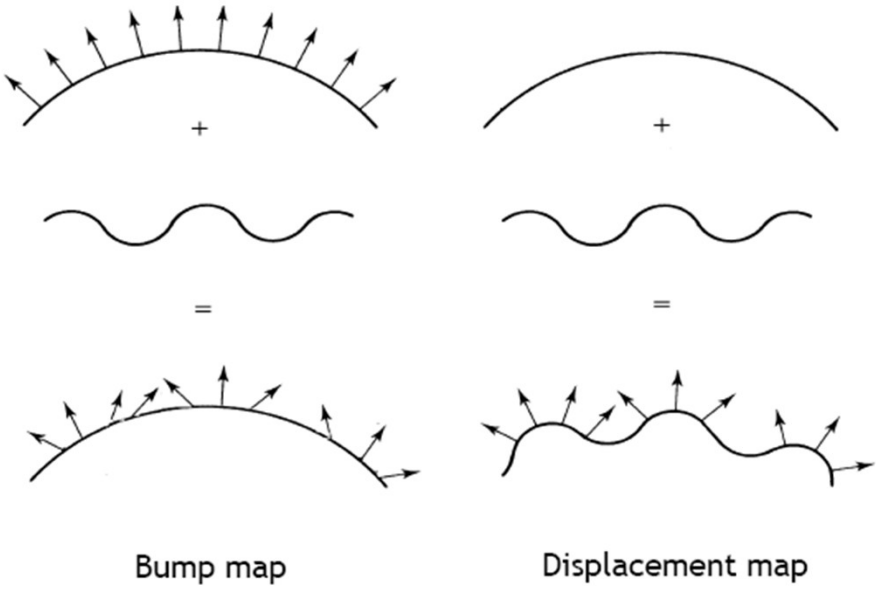
## Procedural Displacement Texture Maps

- Variations of surface position and normals

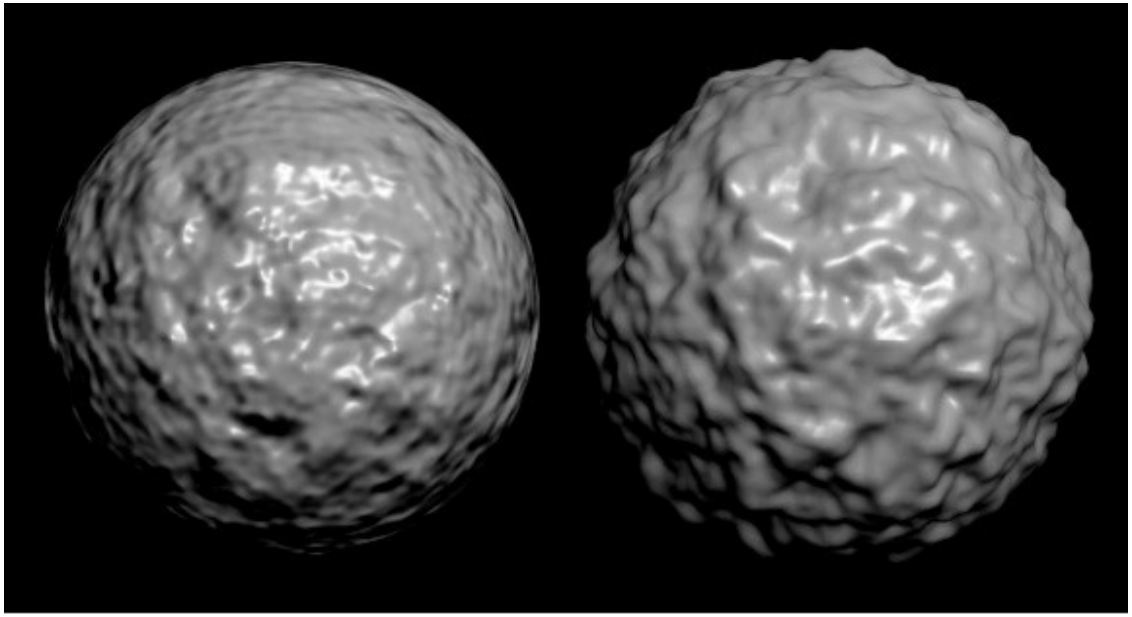




# Bump vs. displacement maps



# Procedural Bump v Displacement Texture Maps



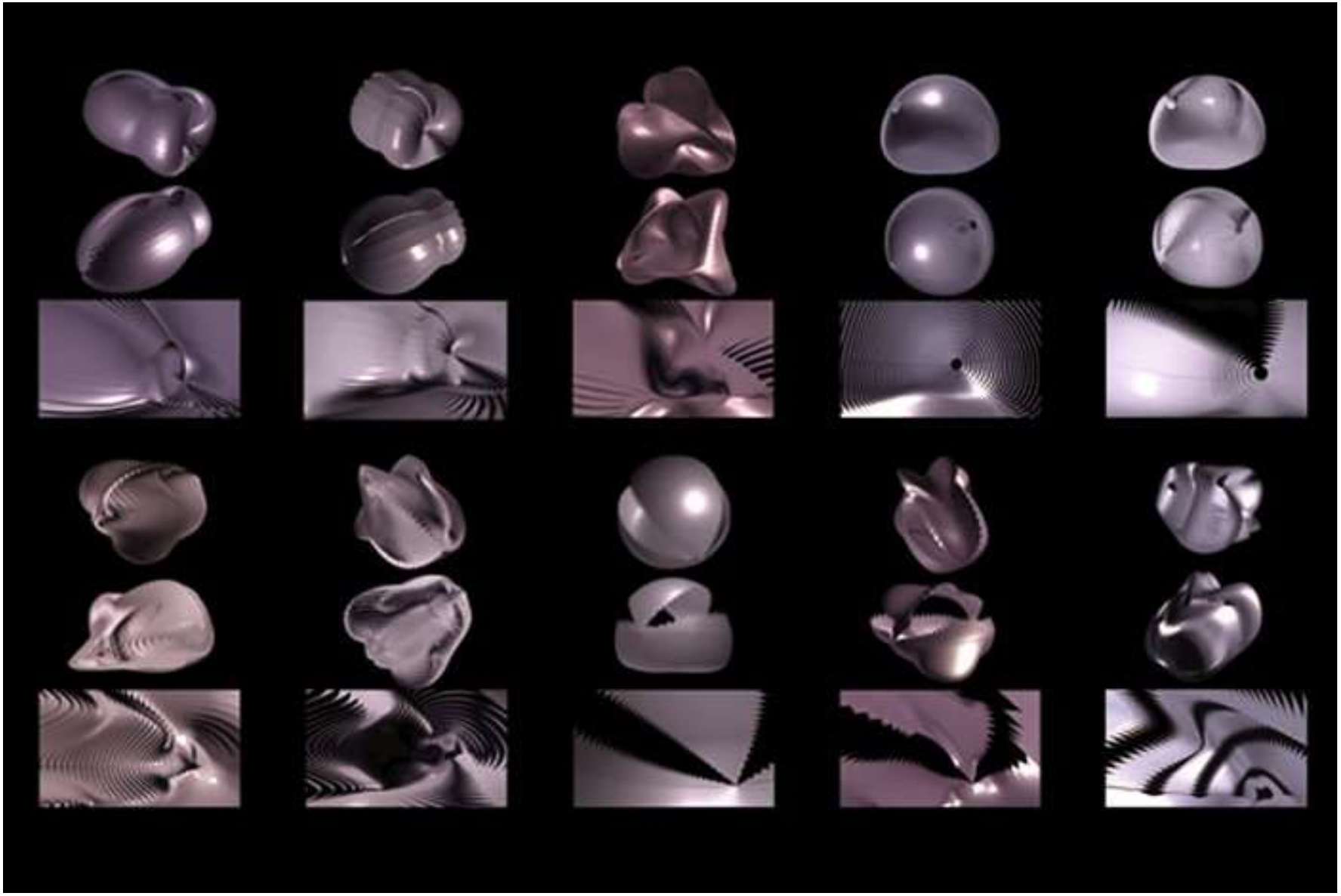
## Type of maps

- almost always combine various types

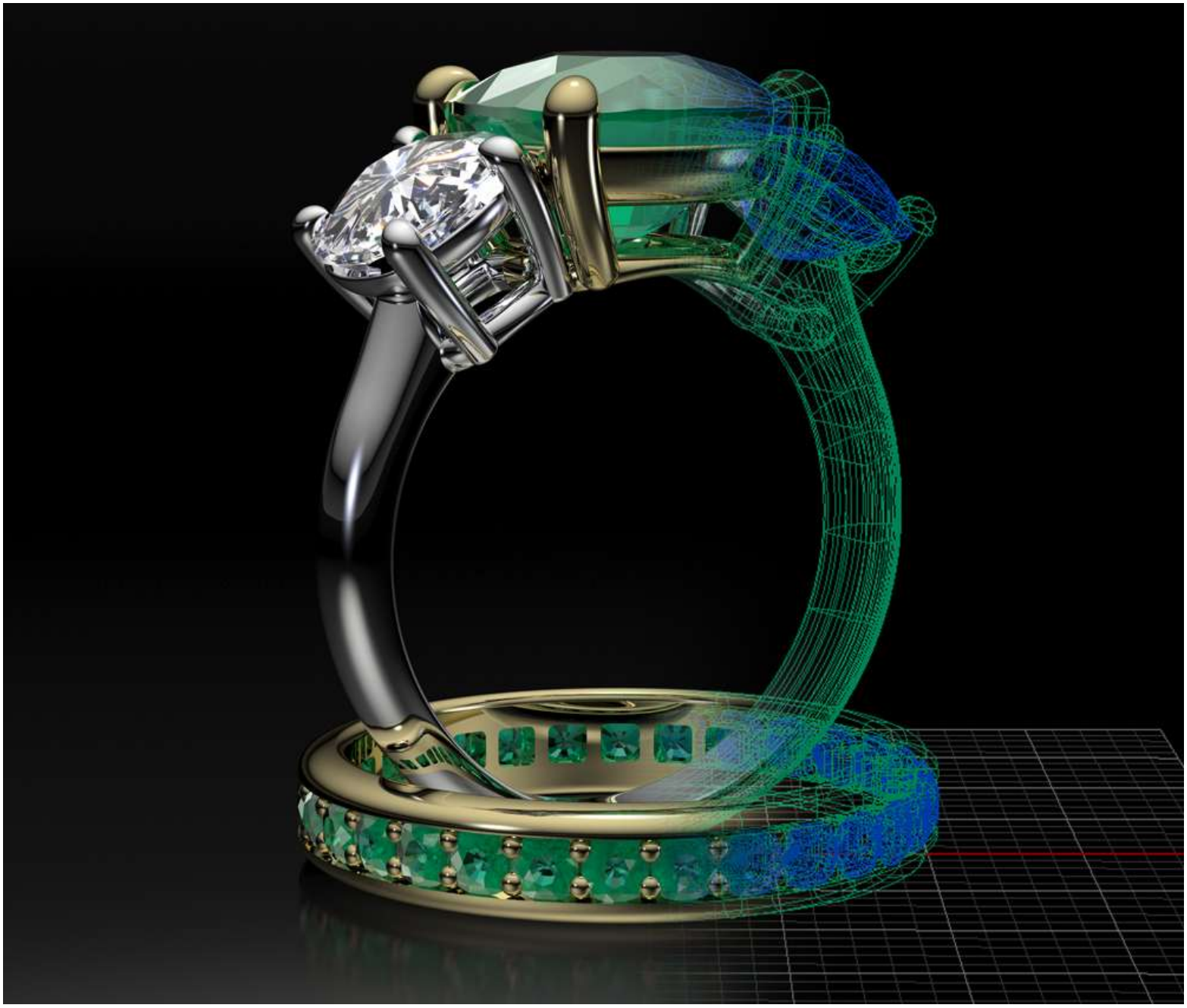
LINK - Cube mapping, Bump mapping, Phong shading

# Gallery

## Gallery



# Gallery



# Gallery

# Gallery

END

## For next week .....

- Rendering Engines:  Group research project
- Divide into four groups
- Research 6 render engines (liaise with other groups to provide for the maximum coverage of all available Render Engines)
- Document :
  - Introduction, including:
    - What is a Render Engine
    - Why it is required
    - By whom
  - Detail: (500 words min per Render Engine) and provide images and video (where available)
- Present research and document  (15 minutes  maximum for each group)

## Future Reading (or as soon as you can) .....

- For future reading – a very good book:
  - Physically Based Rendering: From Theory To Implementation“

    Matt Pharr;