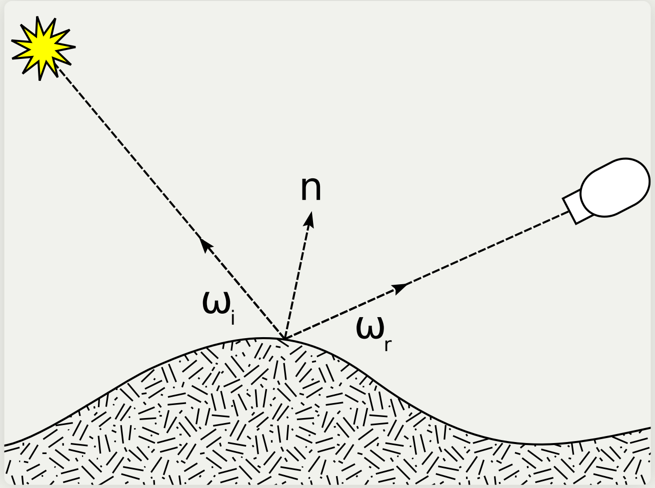
BRDF representations.

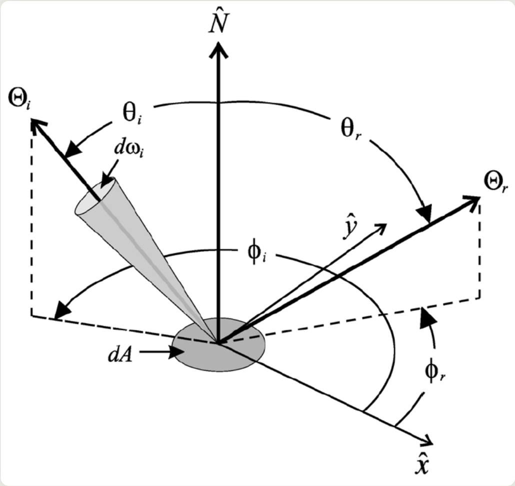
Bidirectional Reflectance Distribution Function

The goal of this lesson is to illustrate different approaches to represent and compute BRDF functions.

Remainder: BRDF

The parameters of the function in a point are the incoming light direction (ω_i), the output light direction (ω_r), computed in the local reference frame of the surface point.





Images shamelessly stolen!

BRDF equation

$$f_r(\vec{\omega}_i, \vec{\omega}_r) = \frac{dL_o(\omega_r)}{dE_i(p, \omega_i)} = \frac{dL_r(\omega_r)}{L_i(\omega_i)\cos(\theta_i)\, d\vec{\omega}_i}$$

- It measures how much the radiance in the directions ω_r depends on the light incoming from the direction ω_i, in a single point.
- $\omega_i = [\theta_i, \varphi_i]$
- θ_i is the altitude, φ_i is the azimuth (horizontal angle)

Spatially Varying BRDF

We add 2 spatial parameters to the BRDF function:

$$f_r(\vec{x}, \vec{\omega}_i, \vec{\omega}_r)$$

s

where $x = [u, v]$ in texture coordinates

It does not capture self-shadowing self-occlusion, inter-reflections.

BTF

Bidirectional Texture Function

Similar to SVBRDF, same parameters but:

- Image based: capture a sample for every direction and light.
- It takes into account the immediate vicinity of a point: self shadowing, self reflection, etc.
- Good for reproducing, not useful for modeling.

6 dimensional functions

- 2 for the incoming angle
- 2 for the outcoming angle, and
- 2 for the surface parametrization

How do we make this data more manageable?

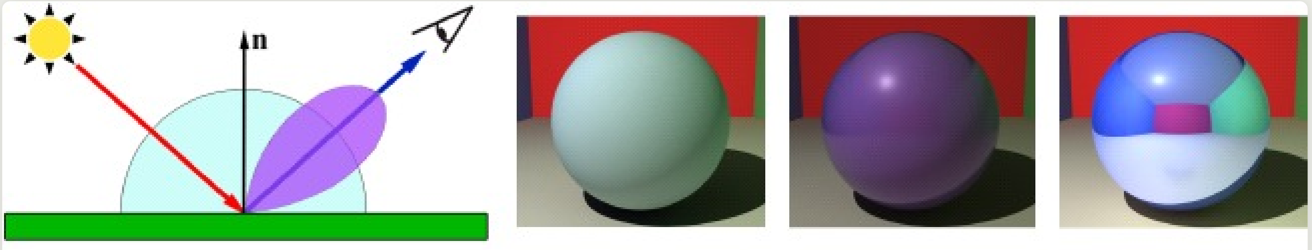
1: Analytical models

2: Data-driven models

Analytical

The formula is computed with an analitic function depending on view direction, light direction and a set of parameters (ambient,diffuse, specular, etc).

Analytical basic example:

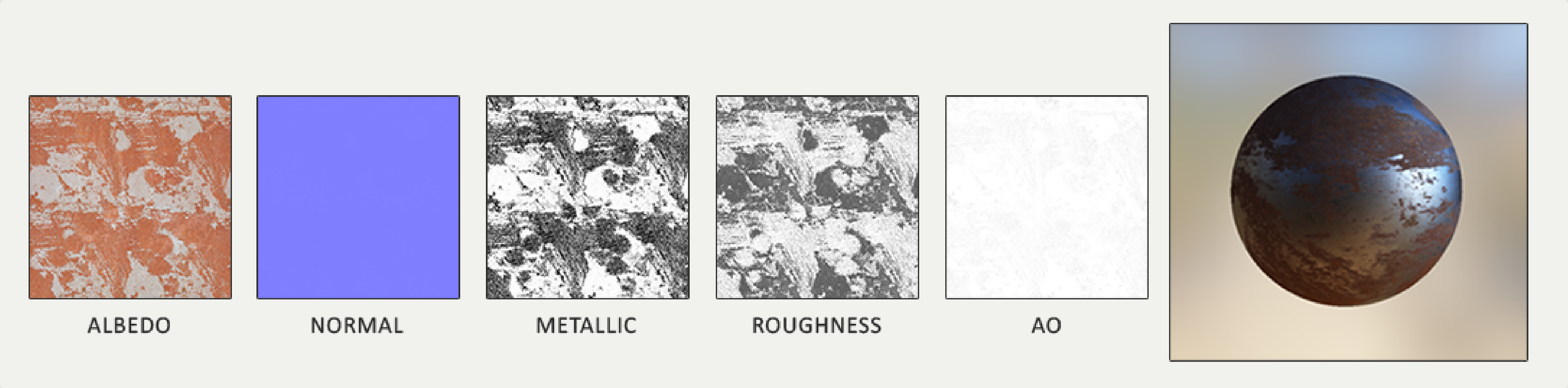


(a) Basic reflection models (b) Diffuse (c) Glossy (d) Specular

Simple approximation, good for plastic, few parameters

$$I = I_a k_a + \sum_p I_p [k_d(\vec{\mathbf{N}} \cdot \vec{\mathbf{L}}) + k_s(\vec{\mathbf{R}} \cdot \vec{\mathbf{V}})^n]$$

Physically based rendering:



- Analitic formulas based on Physics and Optics principles
- Fresnel term, microfacets etc..
- Parameters commonly provided through textures.

Example: Schlick BRDF

$$f = f_{diffuse} + f_{specular}$$

$$f_{diffuse} = (1 - F) * diffuse$$

$$f_{specular} = \frac{F * G * D}{4 * (N \cdot L) * (N \cdot V)}$$

$$f_{specular} = F * Vis * D$$

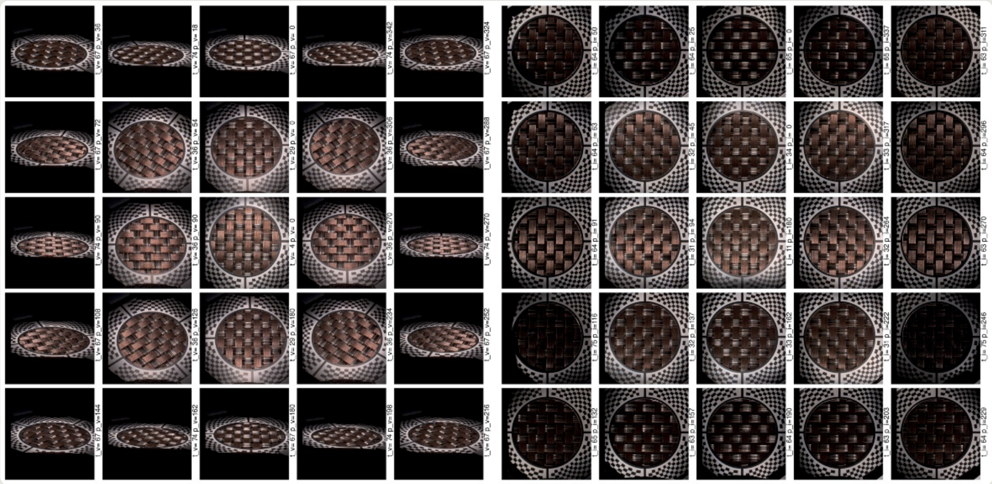
$$Vis = \frac{G}{4 * (N \cdot L) * (N \cdot V)}$$

$$\alpha = roughness^2$$

A more complicated formula and additional parameters

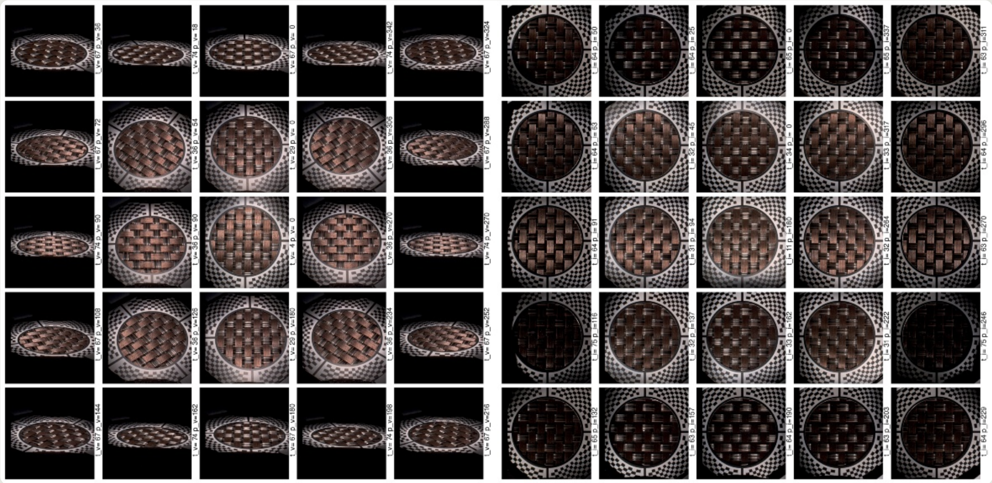
- V: normalized vector to the eye
- L: normalized vector to the light
- N: surface normal
- H: half vector = normalize(L+V)

Data-driven model



Capture a large set of photos from
different view direction and different light direction

Data-driven model



- Can be applied only to measured data, not really useful for modeling.
- The dataset is large, how to represent it in a more compact way?

RTI: a simpler case

Reflectance Trasformation Imaging

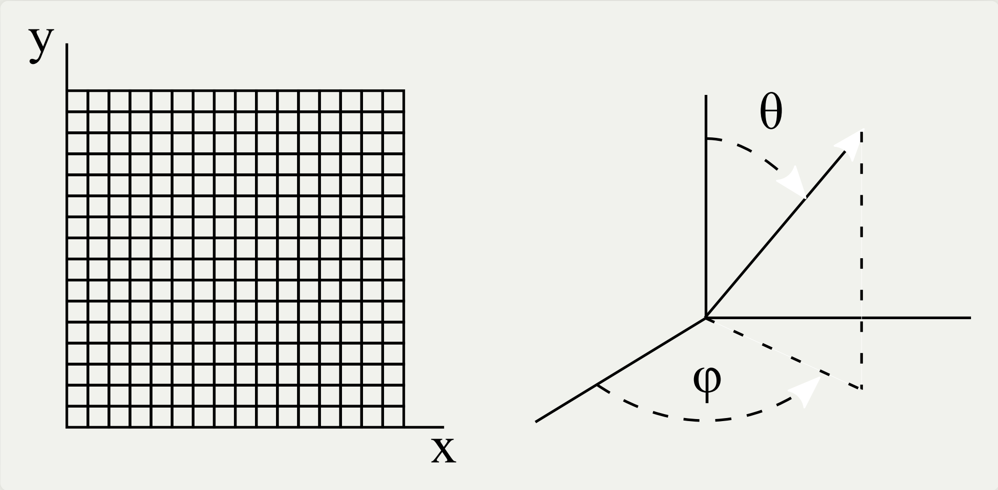
- **Fixed view point**
- Arbitrary lighting
- 4 dimensions.

Demo RTI:

http://pc-ponchio.isti.cnr.it/brdf/relight/relight.html?rti=bottle_bln18

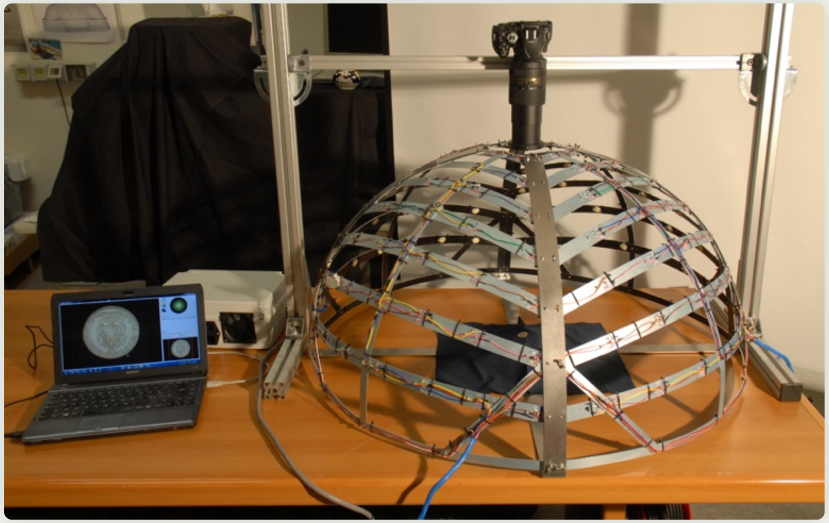
RTI formula

$$[R, G, B] = f(x, y, \theta, \varphi)$$



Sampling the function

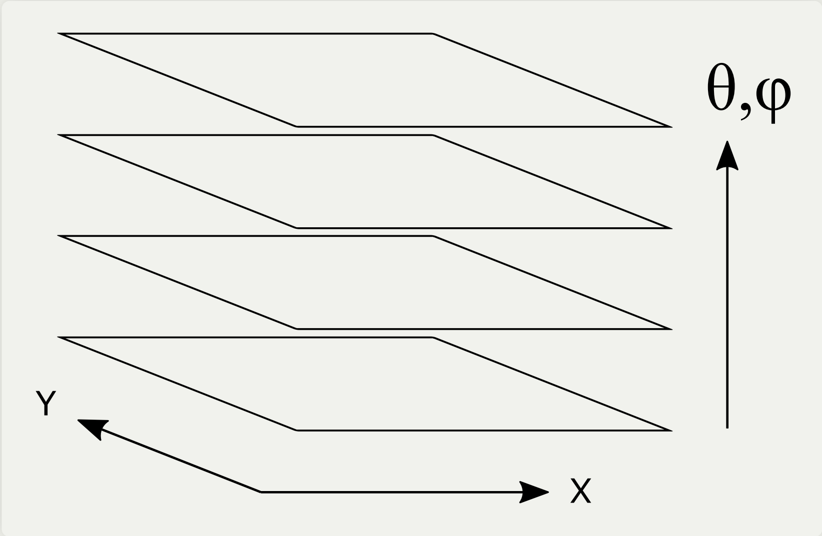
Light dome



Manual light positioning

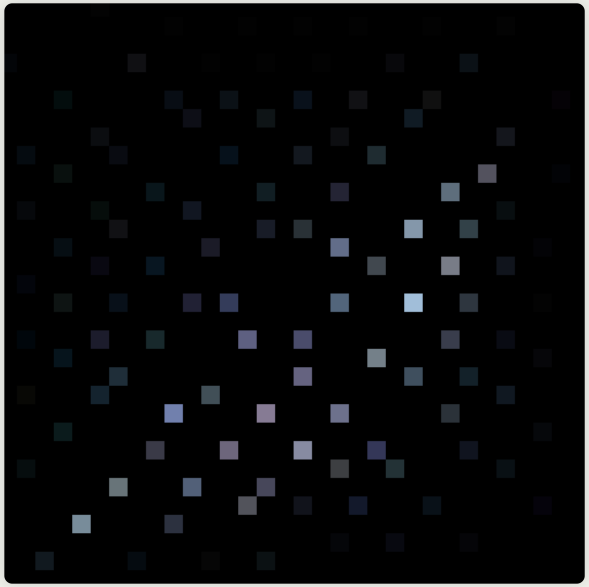
A stack of images

~50-100 images ~500MB!



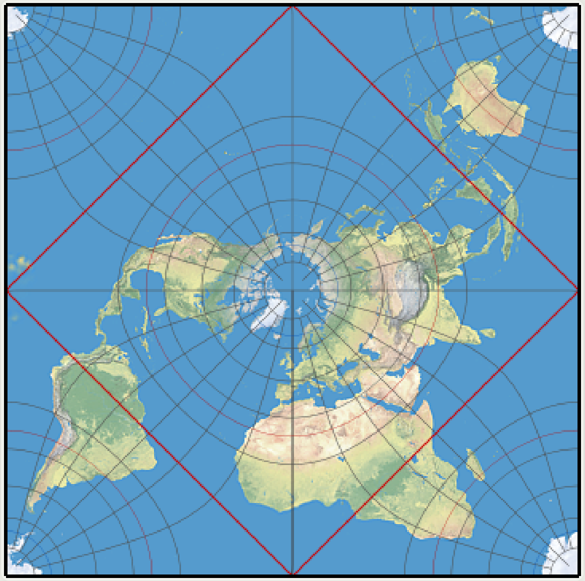
All the pixels for one light

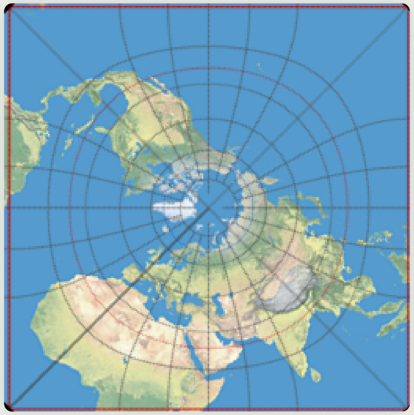
All the lights for one pixel

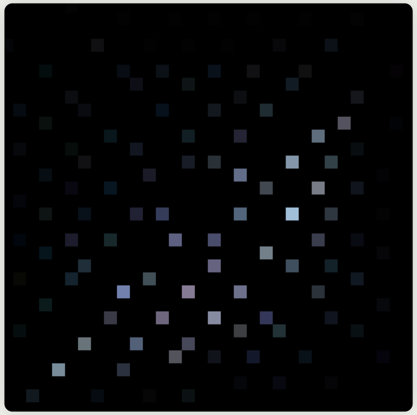


(a column in the stack)

How to map a half sphere to square?





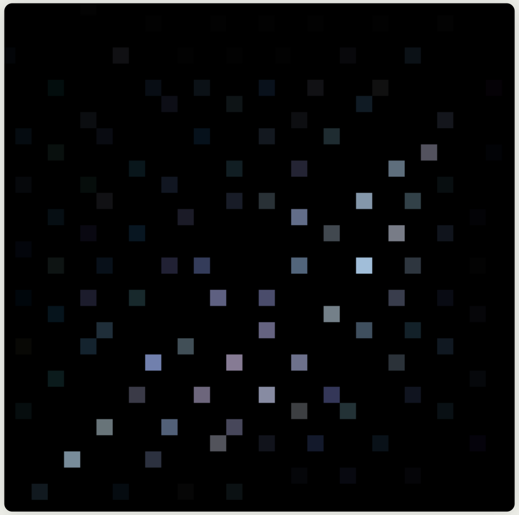


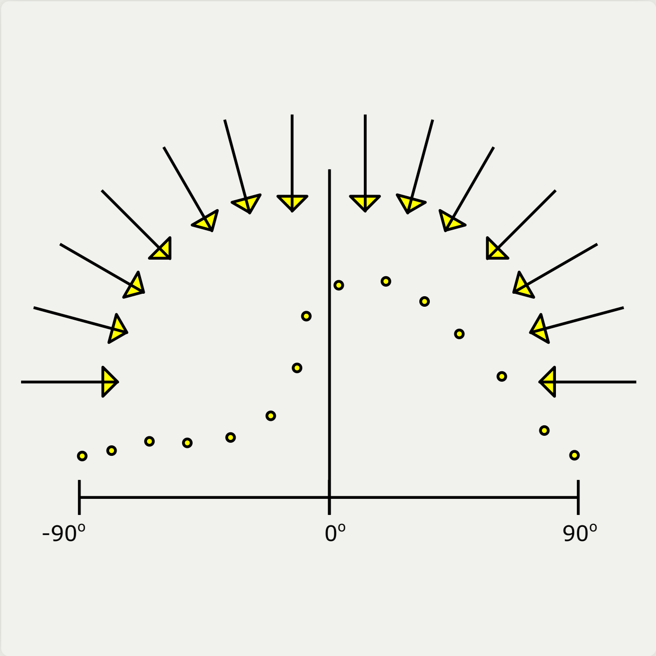
Flattened octahedron projection:

Thomas Engelhardt, Carsten Dachsbacher: Octahedron Environment Maps

Sampled function*

*Actually one function per channel RGB

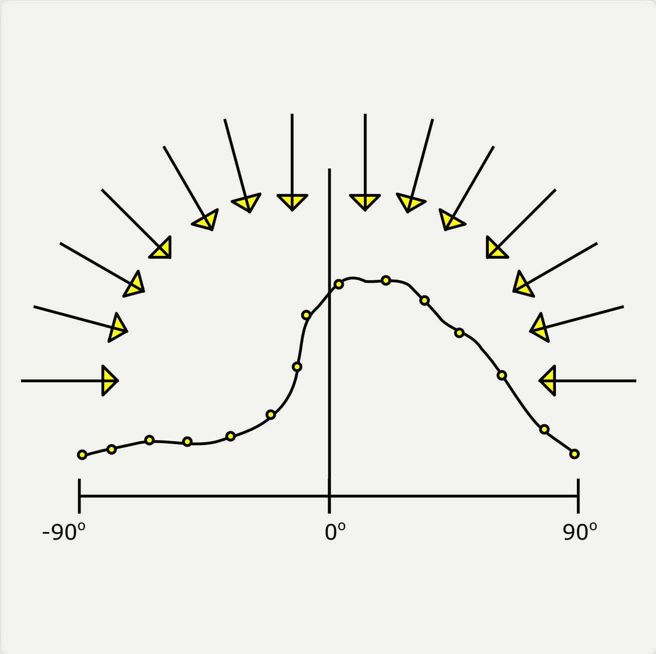




Intensity function for a single pixel depending on the 2 angles.

It is measured for a set of discrete angles

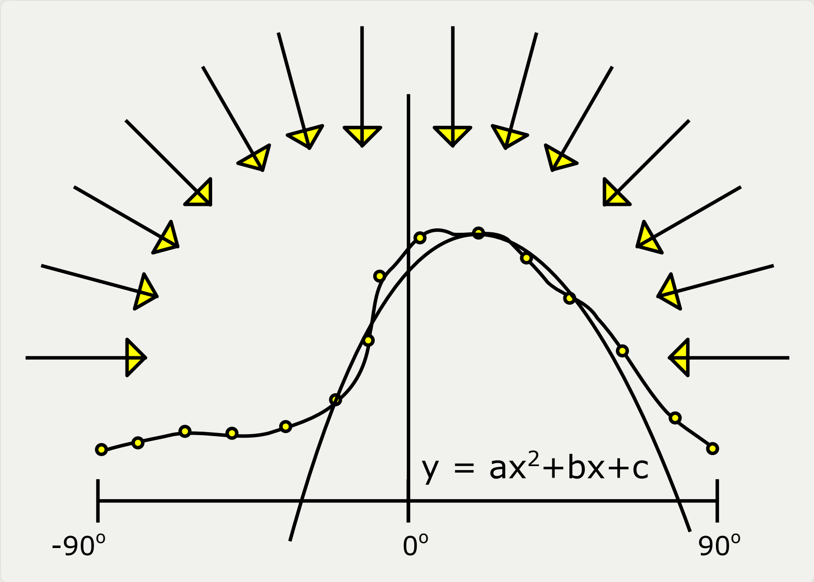
Continuous function



We need a continuos function to be able to render for any light angle

Which interpolation/approximation method?

Polynomial fitting



Approximatin g the function with a polynomial in 2 variables is the most simple approach.

Polynomial Texture Map: PTM

$$R(l) = r_0 + r_1 l_x + r_2 l_y + r_3 l_x^2 + r_4 l_y^2 + r_5 l_x l_y$$

$$G(l) = g_0 + g_1 l_x + g_2 l_y + g_3 l_x^2 + g_4 l_y^2 + g_5 l_x l_y$$

$$B(l) = b_0 + b_1 l_x + b_2 l_y + b_3 l_x^2 + b_4 l_y^2 + b_5 l_x l_y$$

l_x and l_y are the x and y components of the light direction vector

Requires 18 coefficients per pixel:

$r_0 \ldots r_5,\ g_0 \ldots g_5,\ b_0 \ldots b_5$

Malzbender et al. Polynomial Texture Maps (2001)

LPTM: hue is constant.

$$Y(l) = y_0 + y_1 l_x + y_2 l_y + y_3 l_x^2 + y_4 l_y^2 + y_5 l_x l_y$$

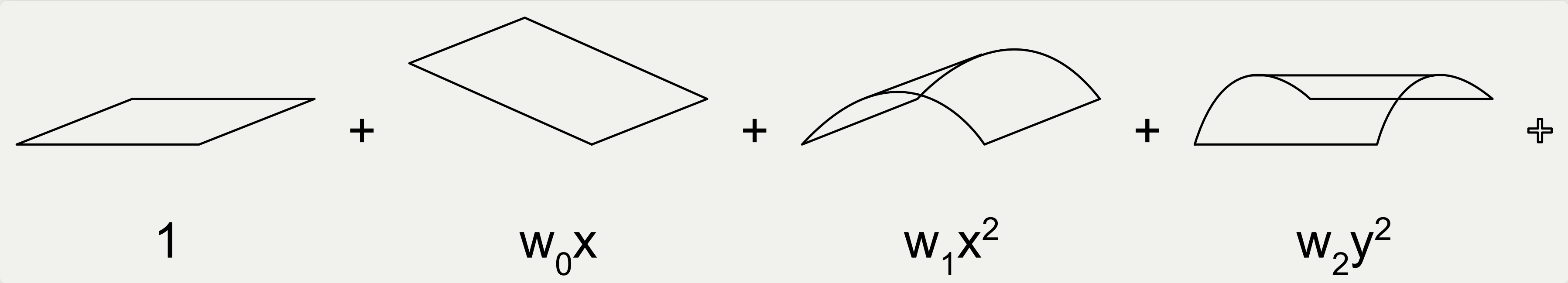
$$R(l) = r_0 Y(l)$$

$$G(l) = g_0 Y(l)$$

$$B(l) = b_0 Y(l)$$

Requires only 9 coefficients, 6 for the luminosity, 3 for the RGB hue.

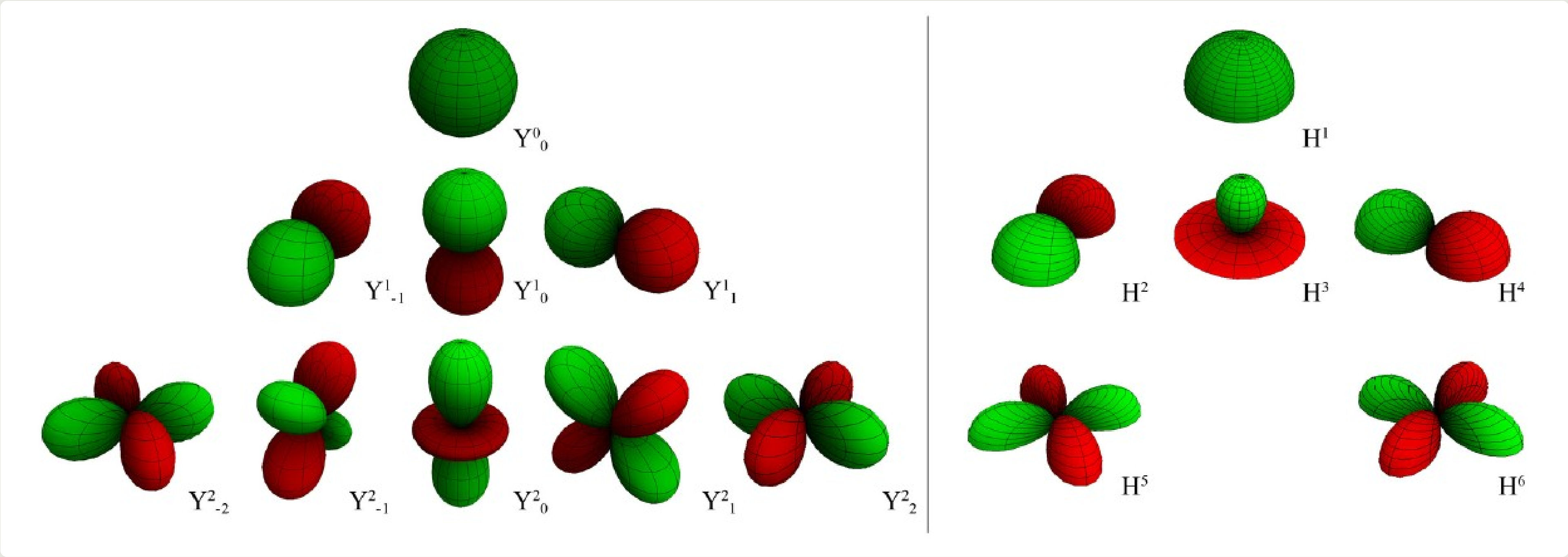
It's a weighted sum of a polynomial basis



Spherical Harmonics

vs.

HemiSpherical Harmonics (HSH)



HSH light coefficients

H_0 = 1 / sqrt(2π)

H_1 = sqrt(6/π)(cos(φ) sqrt(cos(θ)-cos(θ)^2))

H_2 = sqrt(3/(2π))(2cos(θ)-1)

H_3 = sqrt(6/π)(sqrt(cos(θ) - cos(θ)^2)sin(φ))

H_4 = sqrt(30/π)(cos(2φ)(-cos(θ) + cos(θ)^2))

H_5 = sqrt(30/π)(cos(φ)(2cos(θ)-1)sqrt(cos(θ) - cos(θ)^2))

H_6 = sqrt(5/(2π))(1 - 6cos(θ) + 6cos(θ)^2)

H_7 = sqrt(30/π)((2cos(θ)-1)sqrt(cos(θ) - cos(θ)^2)sin(φ))

H_8 = sqrt(30/π)((-cos(θ) + cos(θ)^2)sin(2*φ))

HSH

$$R(\psi,\varphi) = r_0H_0(\psi,\varphi) + \ldots + r_8H_8(\psi,\varphi)$$

$$G(\psi,\varphi) = g_0H_0(\psi,\varphi) + \ldots + g_8H_8(\psi,\varphi)$$

$$B(\psi,\varphi) = b_0H_0(\psi,\varphi) + \ldots + b_8H_8(\psi,\varphi)$$

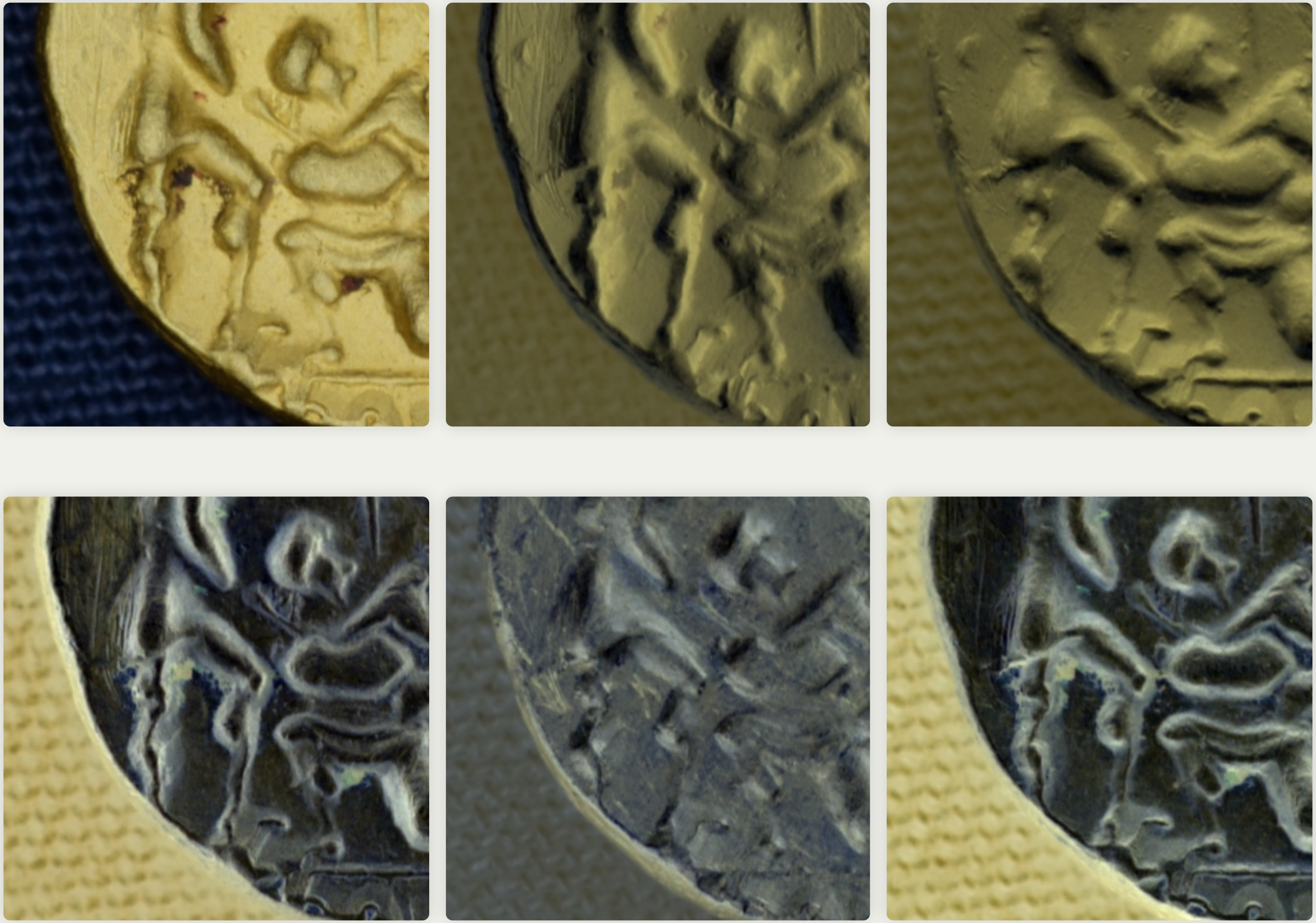
Requires 27 coefficients for the weighted sum of a basis of 27 trigonometrical functions.

As PTM might increase the polinomial degree in HSH higher order harmonics could be used.

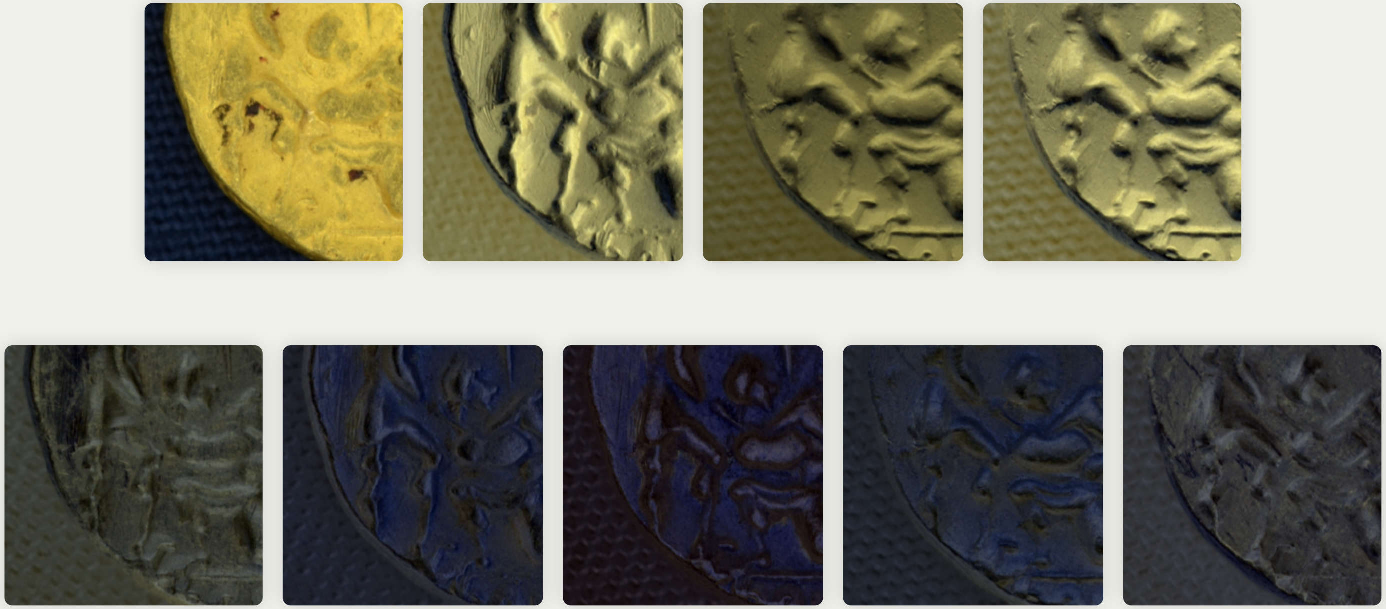
Comparison of PTM and HSH

http://pc-ponchio.isti.cnr.it/brdf/coin10/oldschool.html

Coefficient planes for PTM



Coefficient planes for HSH



RAW: 80MB

HSH: 3.0MB

PTM: 1.9MB

LPTM: 1.4MB

RTI Rendering

Texture lookup, dequantization, light coefficients.

```
uniform sampler2D planes[6];
uniform float light[6], bias[6], scale[6];
varying vec2 v_texcoord;

void main(void) {
        vec3 color = vec3(0);
        for(int j = 0; j < 6; j++) {
                vec4 c = texture2D(planes[j], v_texcoord);
                color.x += light[j]*(c.x - bias[j*3+0])*scale[j*3+0];
                color.y += light[j]*(c.y - bias[j*3+1])*scale[j*3+1];
                color.z += light[j]*(c.z - bias[j*3+2])*scale[j*3+2];
        }
        gl_FragColor = vec4(color, 1.0);
};
```

Data driven approach

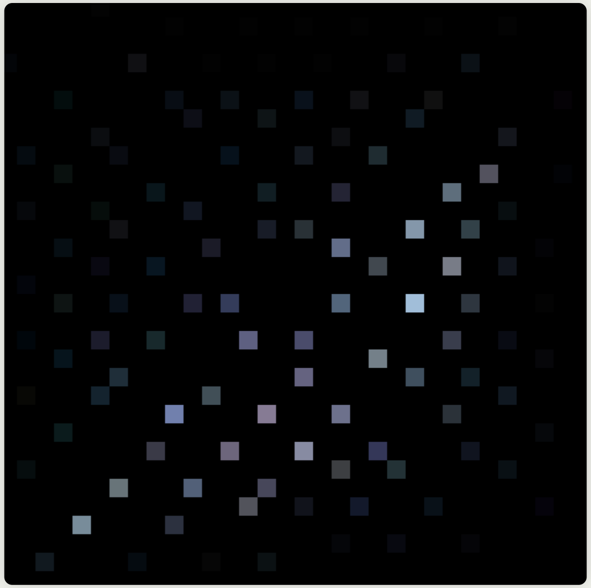
Can we craft customized bases for our dataset?

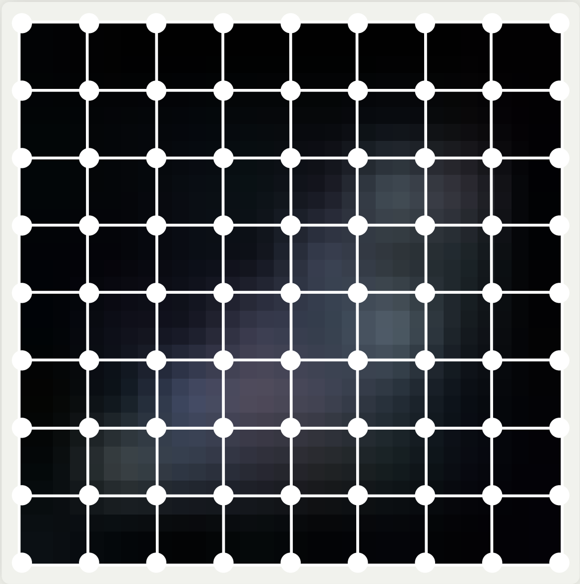
We want to create a set of functions that can be combined to approximate accurately the measured data, using as few coefficients as possible.

Bilinear interpolation

A different way to create a continuous function approximating the measured data

The RTI function, for each pixel, is resampled over a 9x9 grid, using the octahedron projection.





Bilinear interpolation rendering

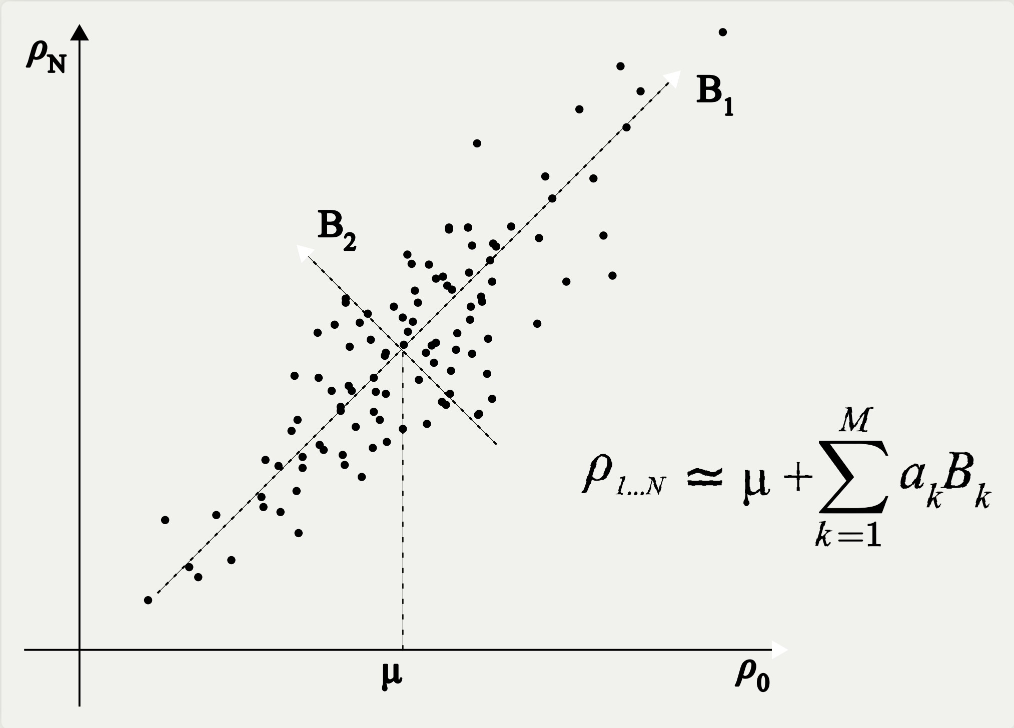
$$\rho = (\rho_0 , \cdots , \rho_{81})$$

$$c = \sum_{i=1}^{81} W_i(\theta,\varphi)\, \rho_i$$

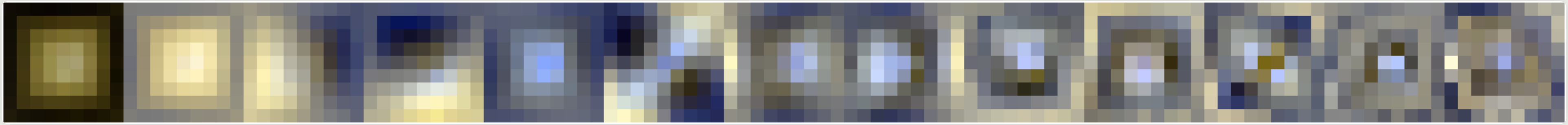
- The 9x9 grid can be represented by ρ, a 81 dimensional array
- $W_i(\theta, \varphi)$ is the function that for a light direction weights the contribute of each pixel of the grid.
  W_i would be zero except for the 4 nearby pixels.
- We are using 81 coefficients, though!

Principal component analysis (PCA)

We can approximate a vector ρ using a suitable basis



A custom base using PCA



This are the first 13 elements of the PCA basis for the coin dataset

Rendering algorithm:

$$c = \sum_{i=1}^{81} W_i(\theta,\varphi)\,\rho_i$$

$$\rho = (\rho_0, \cdots, \rho_{81})$$

PCA:

$$\rho \cong \mu + \sum_{k=1}^{M} a_k B_k$$

$$c \cong \sum_{i=1}^{N} W_i(\theta,\varphi)\left(\mu_i + \sum_{k=1}^{M} a_k B_{k,i}\right)$$

$$c = \sum_{i=1}^{N} W_i(\theta,\varphi)\,\mu_i + \sum_{k=1}^{M} a_k \sum_{i=1}^{N} W_i(\theta,\varphi)\,B_{k,i}$$

Constant for all pixels.

$$c = w_0(\theta,\varphi) + \sum_{k=1}^{M} a_k w_k(\theta,\varphi)$$

- Replace ρ with the PCA approximation
- We can precompute the weighted sum of the basis B of the PCA: they are the same for all the pixels.
- The final per pixel computation is a weighted sum, exacy as in the PTM and HSH case.

Quality comparison:

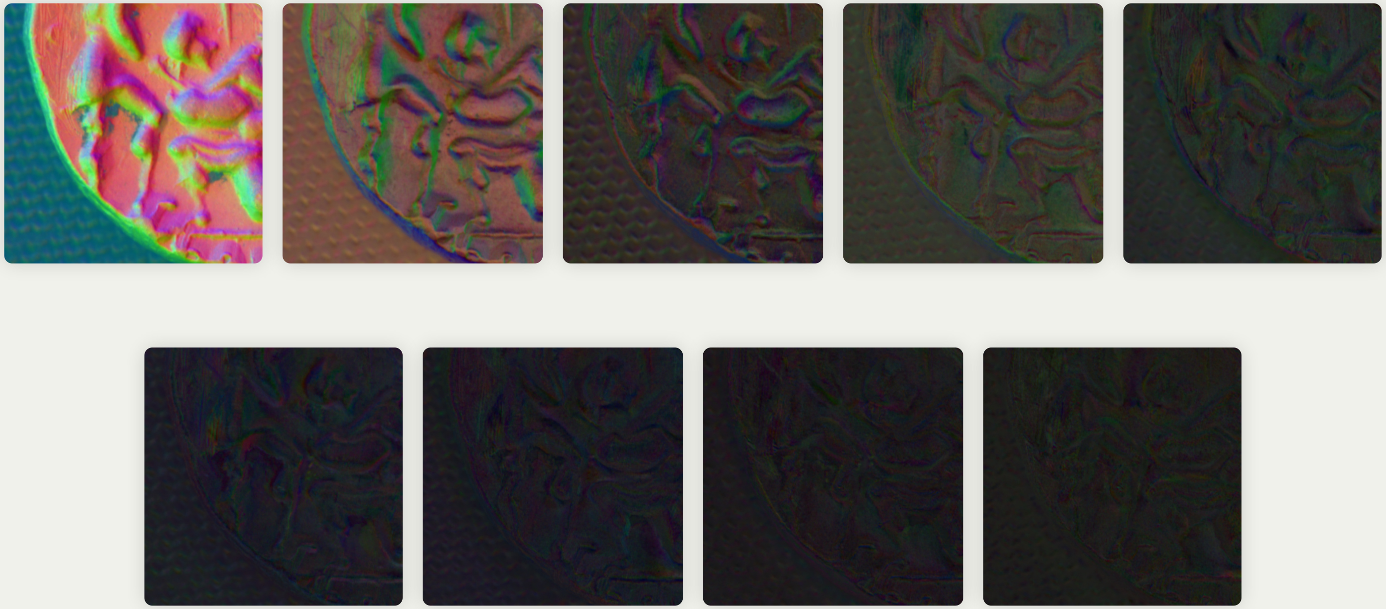
Reflections are the hardest part to replicate

HSH: 27 coeff, 610Kb

PCA: 18 coeff, 473Kb

Original image: 210Kb

PCA coefficient planes



References

- PBR: https://learnopengl.com/PBR/Theory
- PTM: Malzbender et al. Polynomial Texture Maps
- HSH: Gautron et al. A Novel Hemispherical Basis for Accurate and Efficient Rendering.
- BILINEAR: Ponchio et. al. A Compact Representation of Relightable Images for the Web