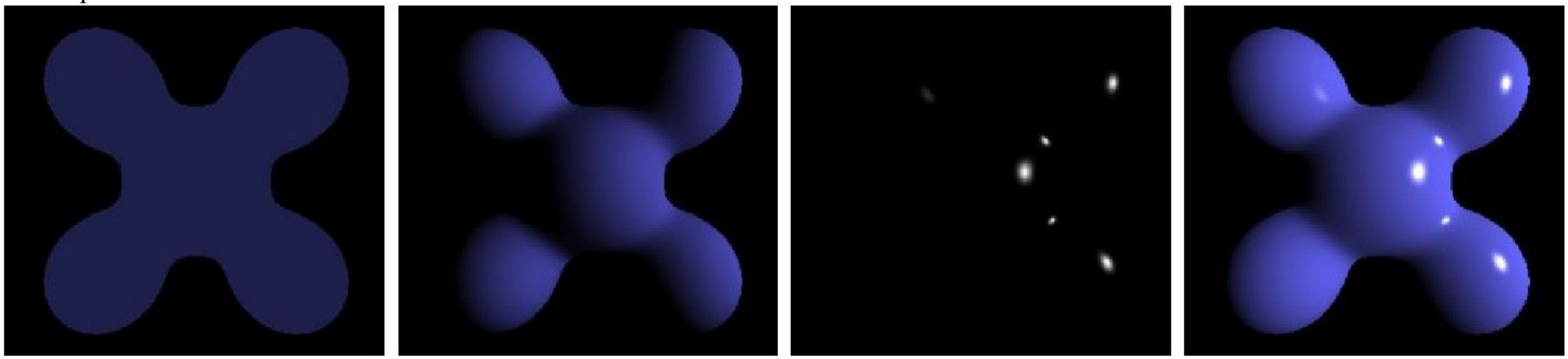
9: Advanced shading techniques

Obtaining realistic renderings in real-time!

Remember the Phong's local model...

[Phong CACM 1975]

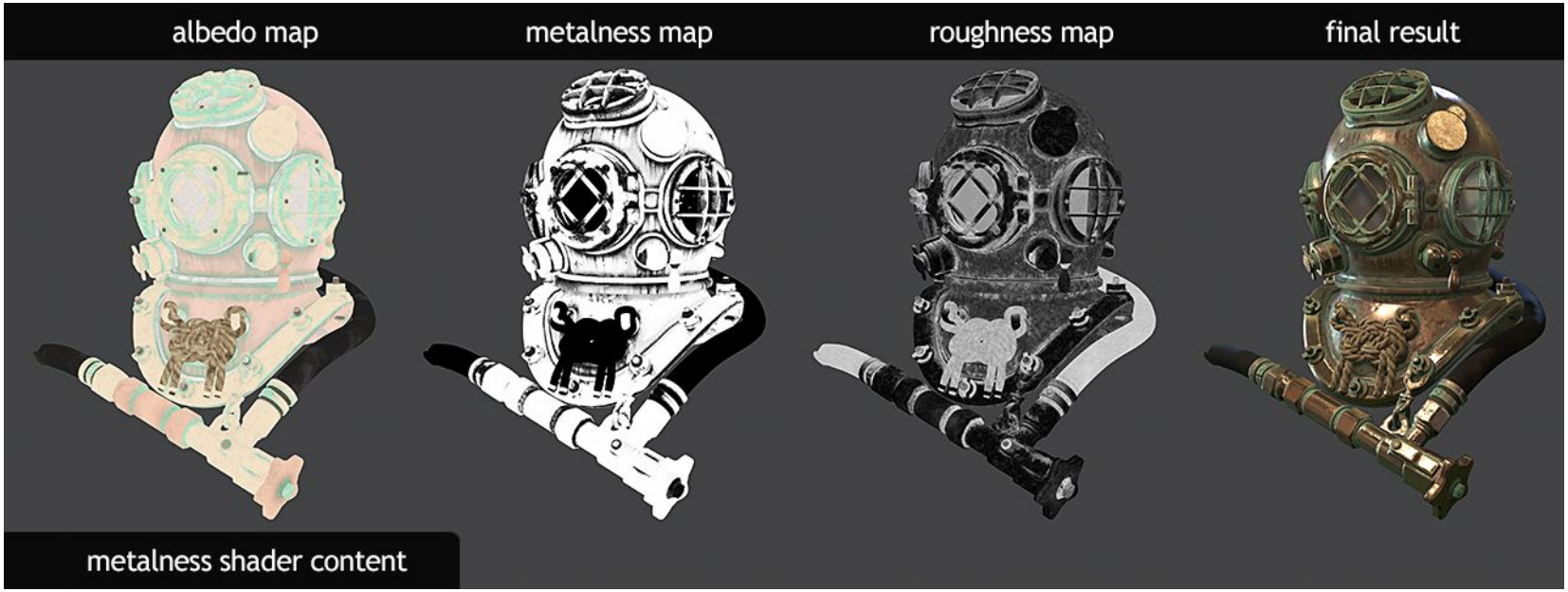
wikipedia



$$L(\mathbf{p} \rightarrow \mathbf{e}) = K_a + \sum_s [K_d(\mathbf{n} \cdot \boldsymbol{\ell}) + K_s(\mathbf{r} \cdot \mathbf{e})^n] I_s$$

ambiant diffuse specular

… varying materials via the use of textures



https://www.marmoset.co/posts/pbr-texture-conversion/

$$L(\mathbf{p} \rightarrow \mathbf{e}) = K_a + \sum_s [K_d(\mathbf{n} \cdot \boldsymbol{\ell}) + K_s(\mathbf{r} \cdot \mathbf{e})^n] I_s$$

ambiant diffuse specular

Use textures for more than material effects

- to modify surface appearance
- to modify lighting properties

$$L(\mathbf{p} \rightarrow \mathbf{e}) = K_a + \sum_s [K_d(\mathbf{n} \cdot \boldsymbol{\ell}) + K_s(\mathbf{r} \cdot \mathbf{e})^n] I_s$$

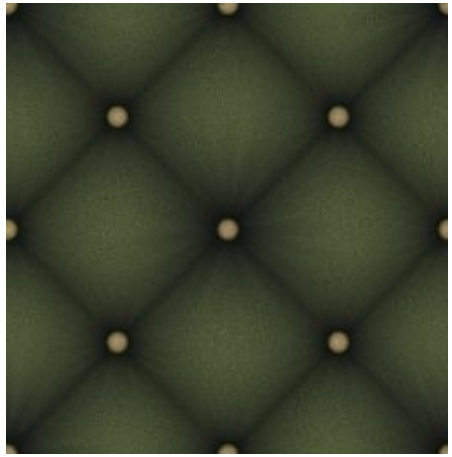
Modify surface appearance

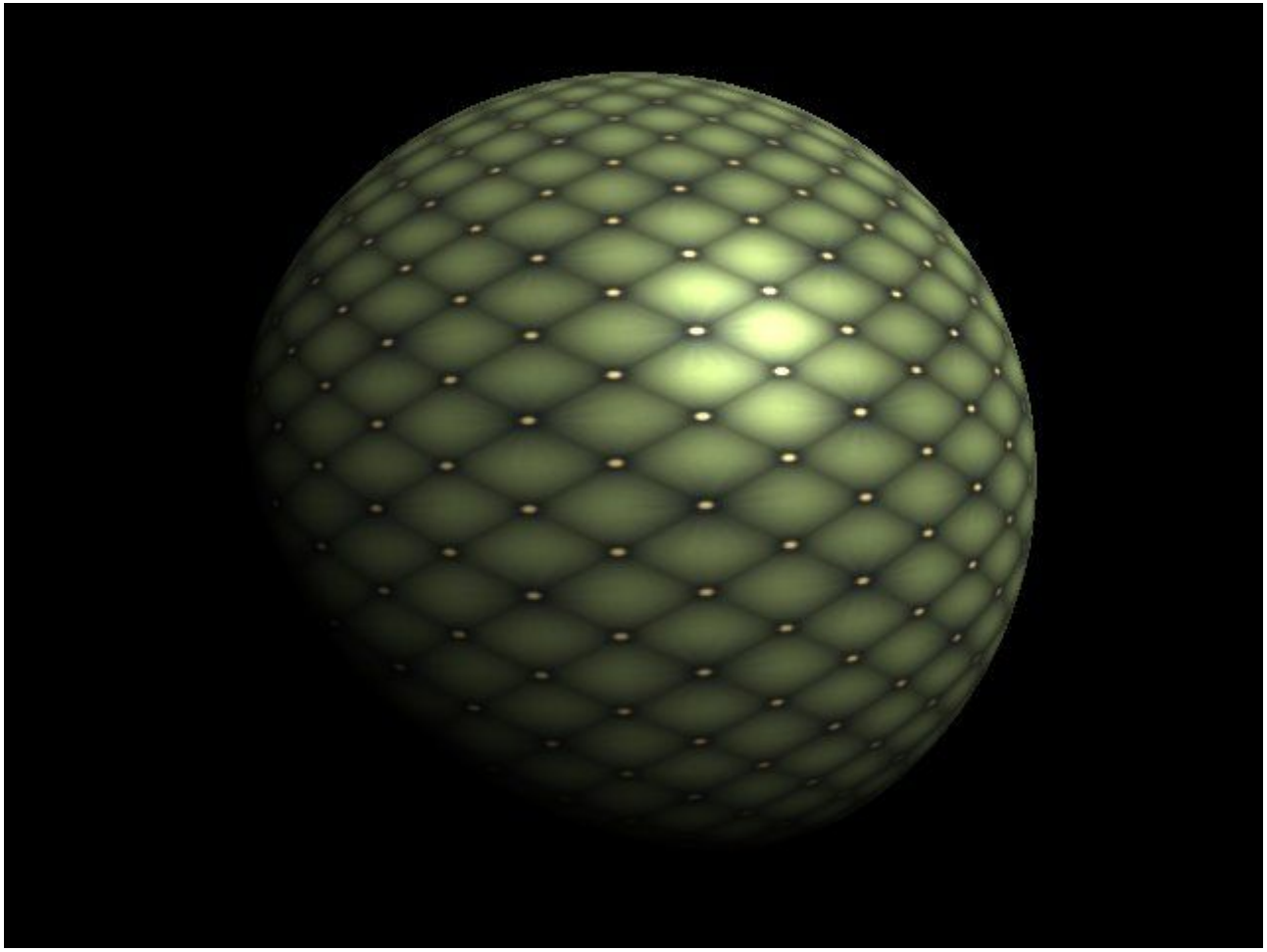
- Normal mapping
- Bump mapping
- Parallax mapping
- Displacement mapping

Modify surface appearance

**** Normal mapping ****

Goal: locally perturb normals to create the illusion of modified geometry

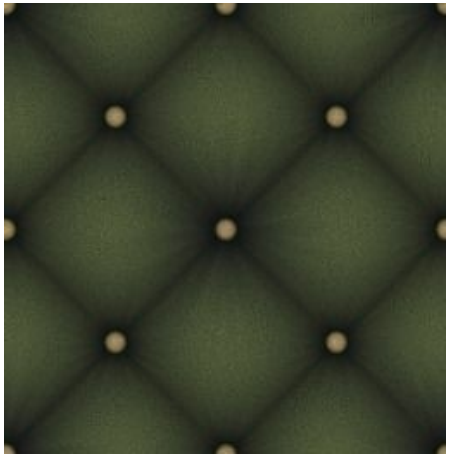


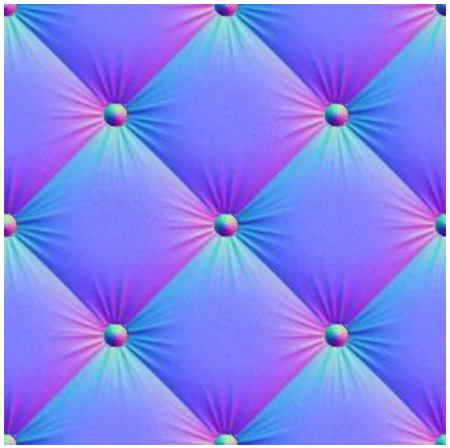


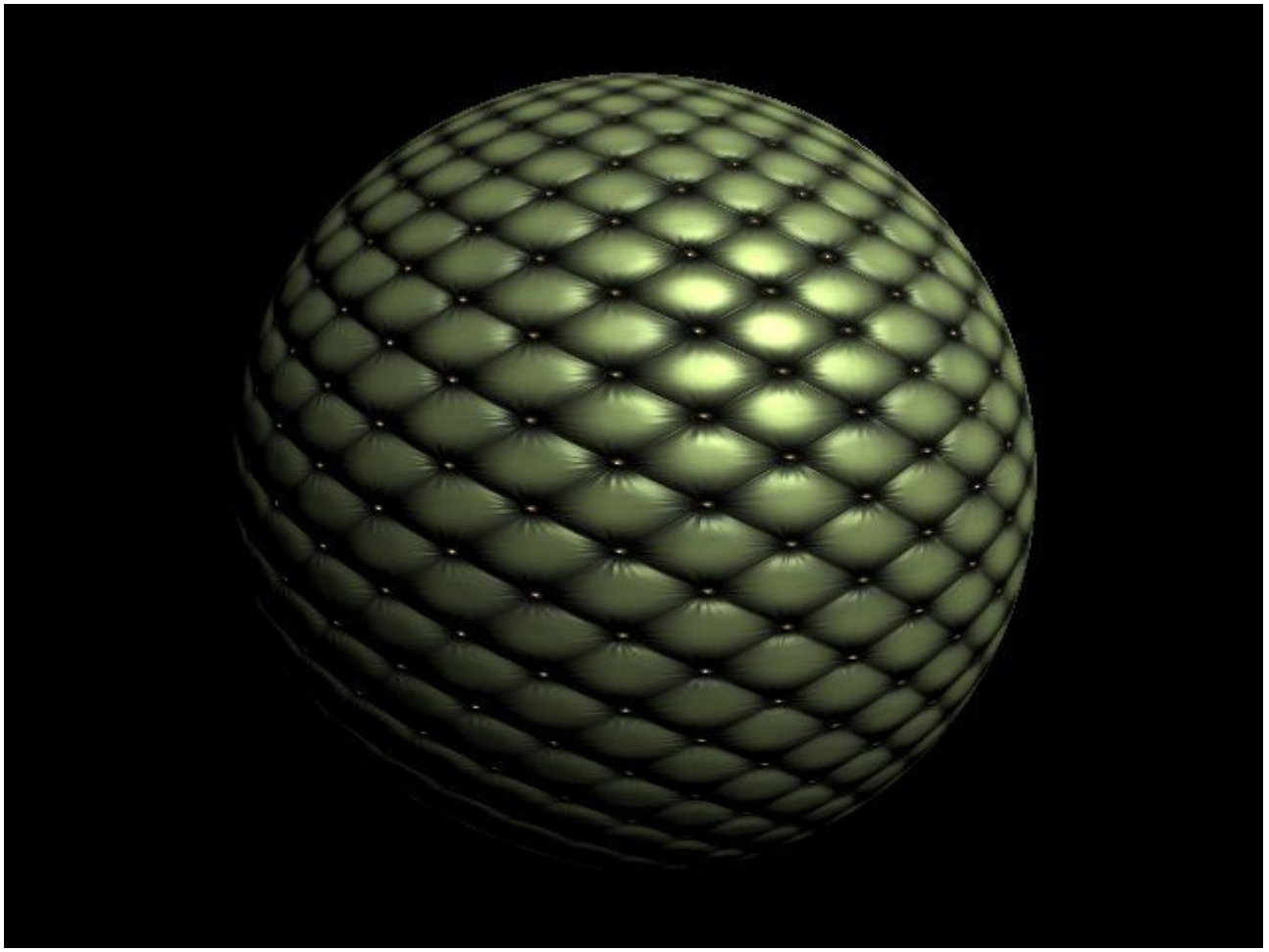
Modify surface appearance

*** *Normal mapping* ***

Goal: locally perturb normals to create the illusion of modified geometry



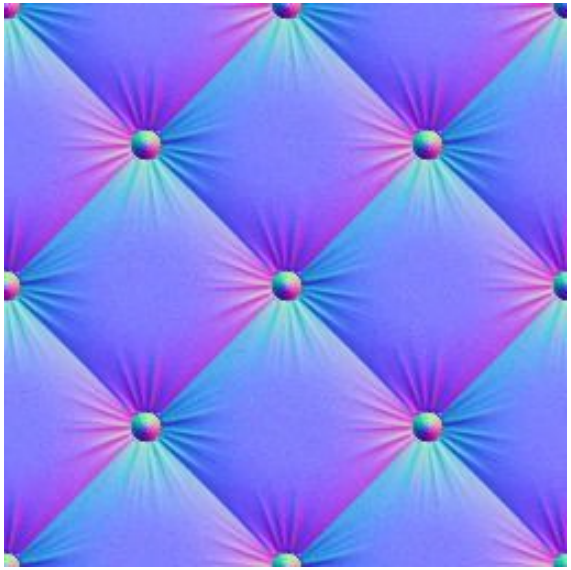




Modify surface appearance

**** Normal mapping ****

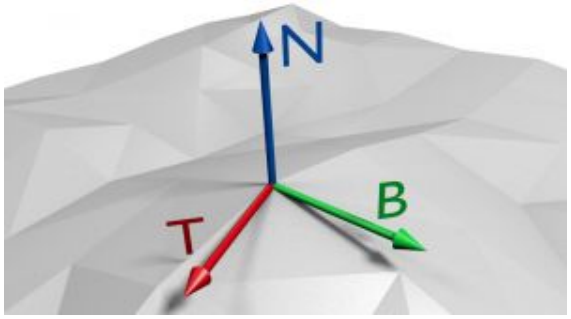
Goal: locally perturb normals to create the illusion of modified geometry



In practice:

- normals stored in the “tangent space”
- requires transformation matrix
  - from world to tangent space
  - (or inversely)

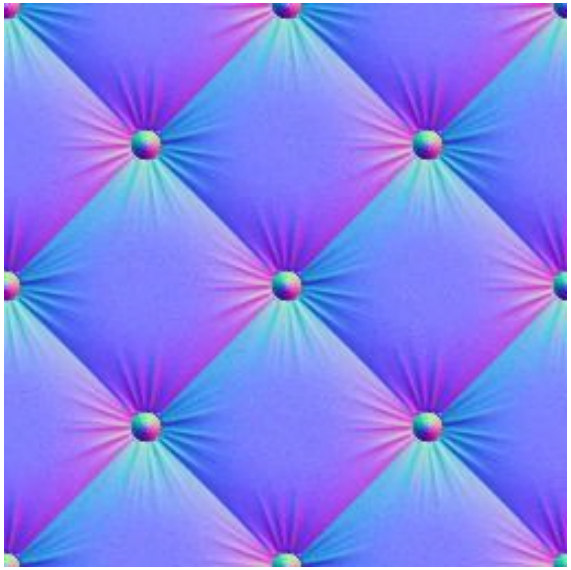
- Tangents computed as a pre-process
  - according to tex coords derivatives
  - see http://www.terathon.com/code/tangent.html
  - stored as a new per-vertex attribute



Modify surface appearance

**** Normal mapping ****

Goal: locally perturb normals to create the illusion of modified geometry

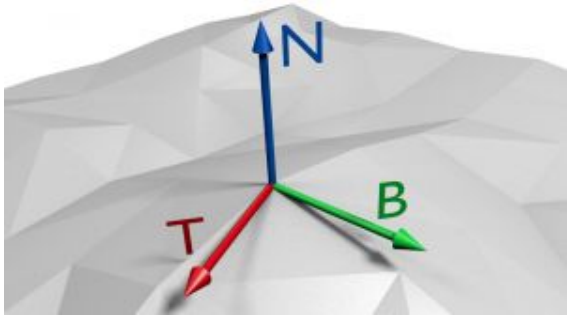


In practice (GPU side):

- Given original normal N and tangent T
- Compute the binormal vector:
  - B = N x T
- Build the TBN matrix:

$$\begin{pmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{pmatrix}$$

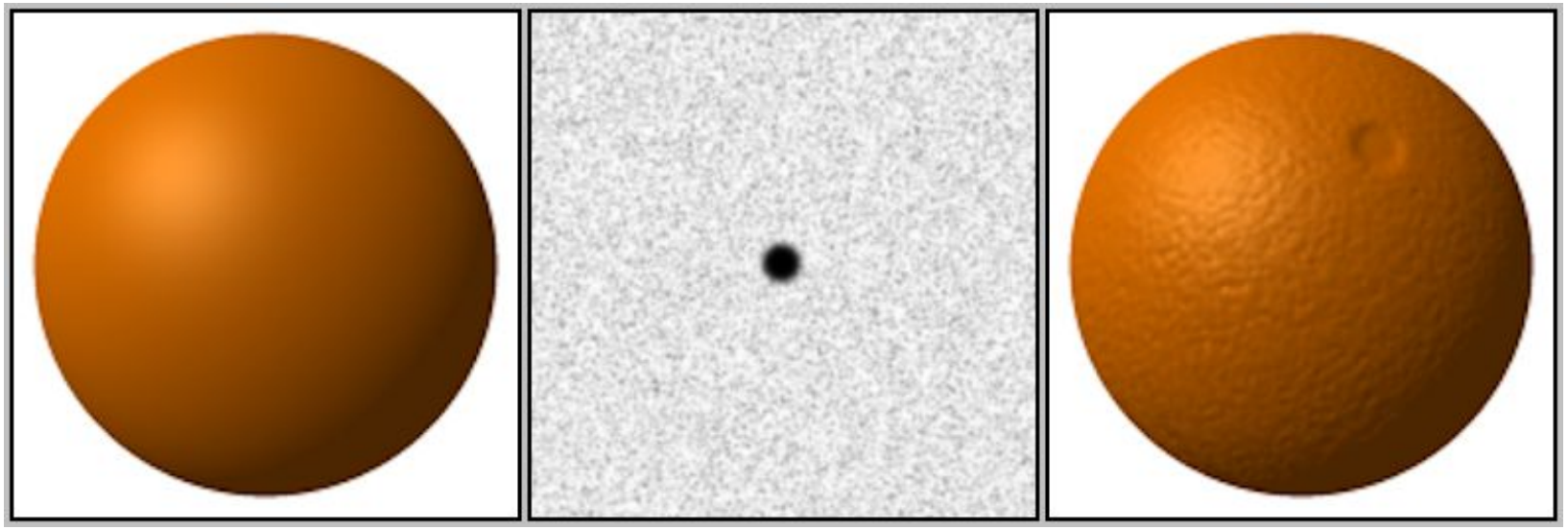
- Transform a vector **v**
  - from tangent to world: $\mathbf{v}_w = \text{TBN}\ \mathbf{v}_t$
  - from world to tangent: $\mathbf{v}_t = \text{TBN}\ \mathbf{v}_w$



Modify surface appearance

**** Bump mapping ****

Same principle: but uses a depth (grey-level) map as input



In practice:

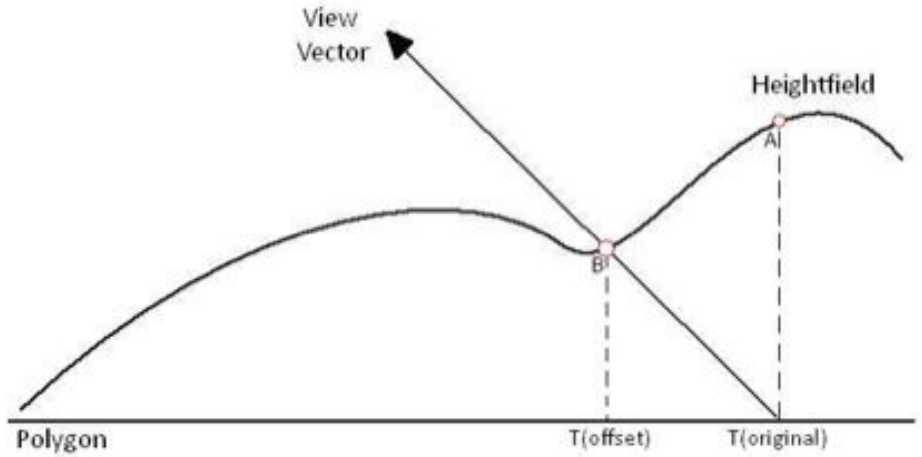
- Compute normals from the depth (using finite differences)
- Apply the normal mapping method

Modify surface appearance

**** Parallax mapping ****

Same goal, but more realistic

- Bump/normal mapping does not fetch the good depth/normal values



In practice:

- walk along the (projected) view vector
- detect the right values at the intersection between the view and perturbed heightfield
- apply the normal mapping approach
- more info: http://sunandblackcat.com/tipFullView.php?l=eng&topicid=28

Modify surface appearance

**** Parallax mapping ****

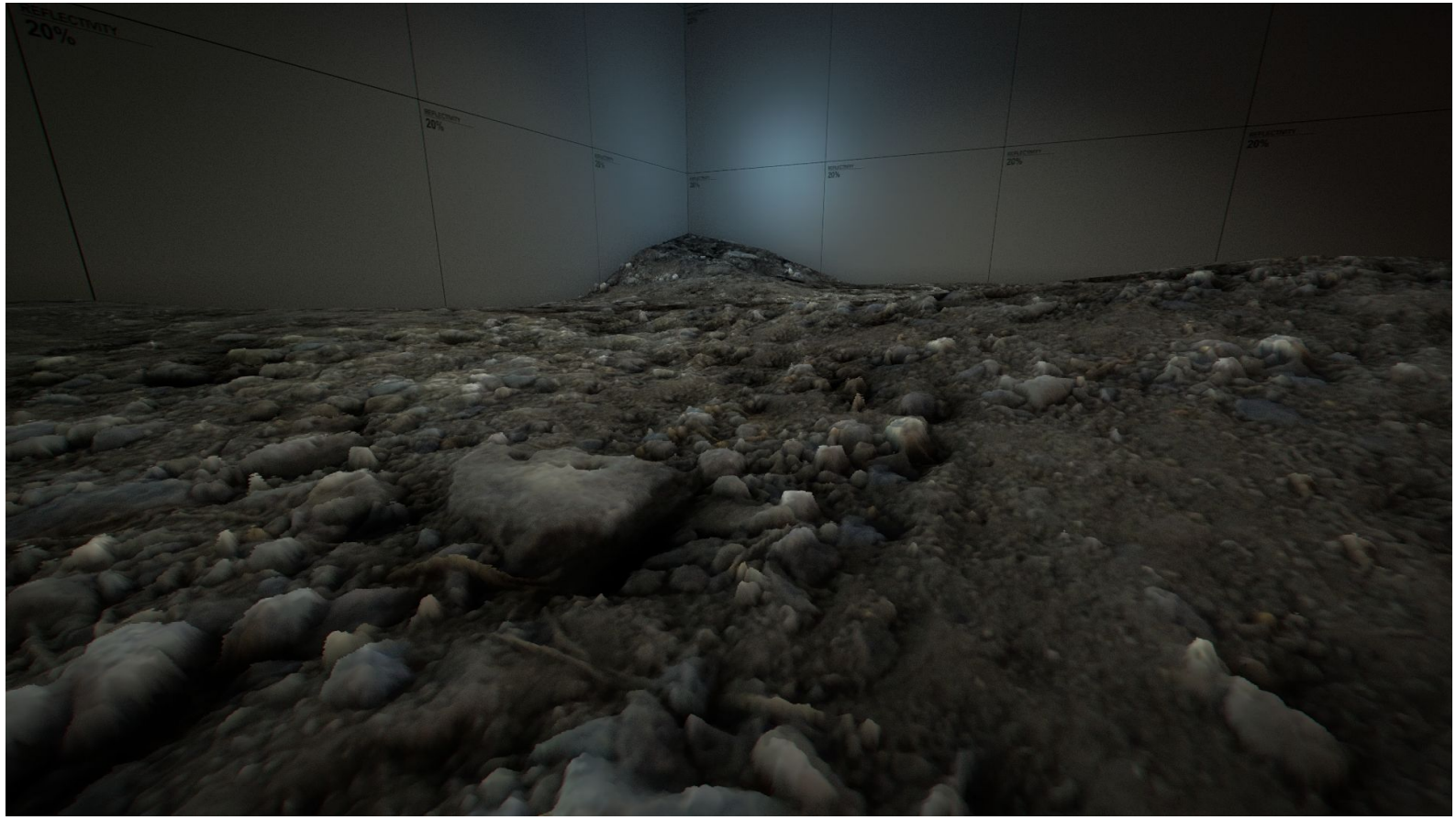
Same goal, but more realistic

without

Modify surface appearance

**** Parallax mapping ****

Same goal, but more realistic

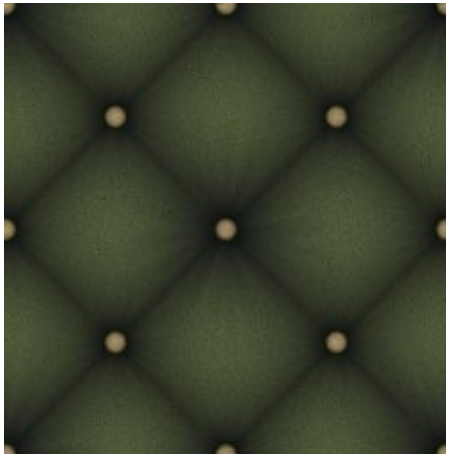


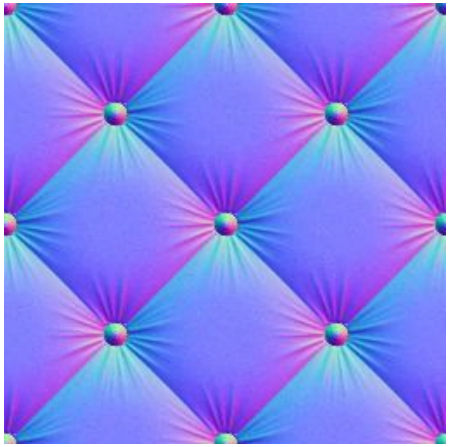
with

Modify surface appearance

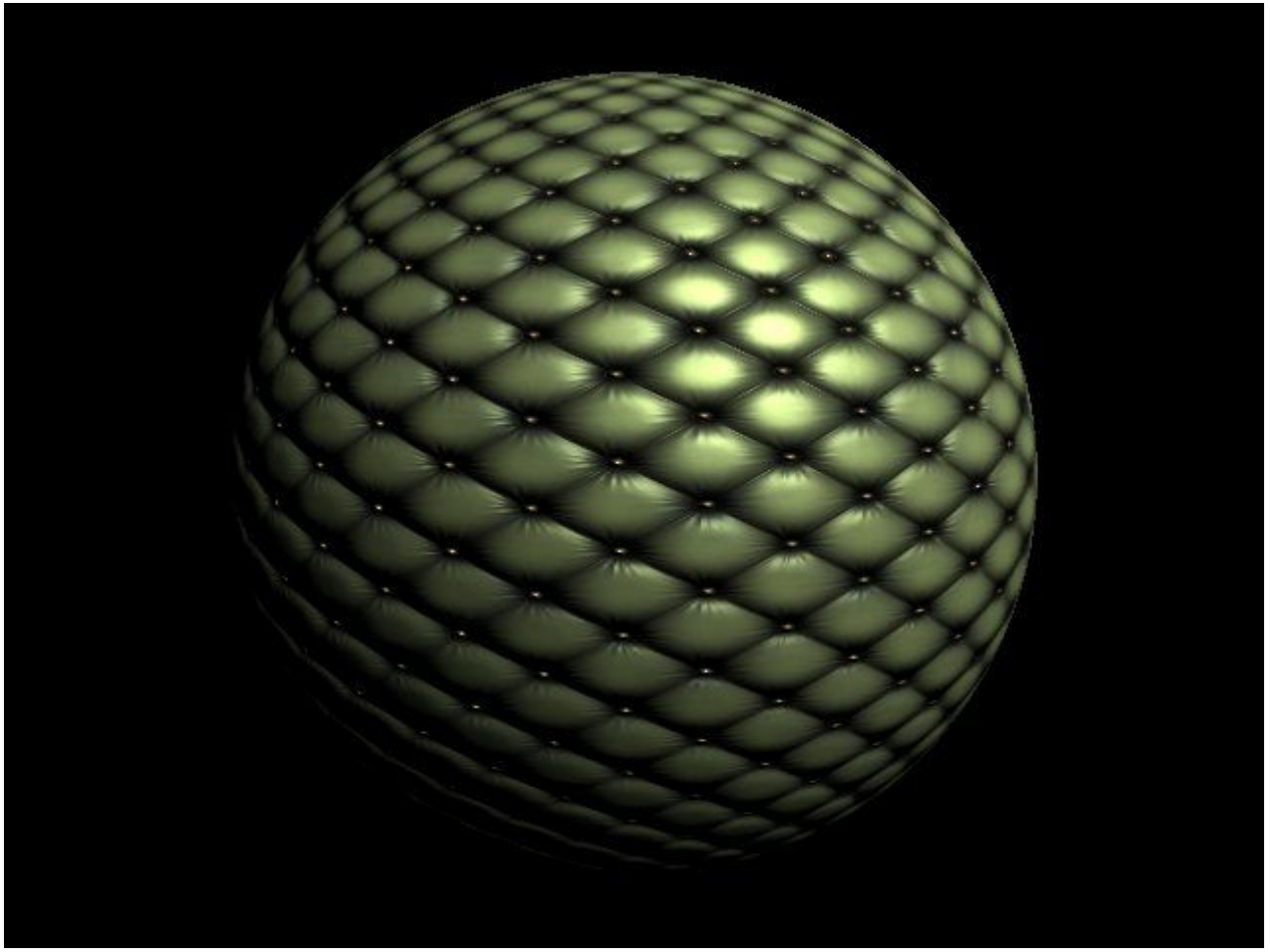
**** Displacement mapping ****

Goal: actually displace vertices!





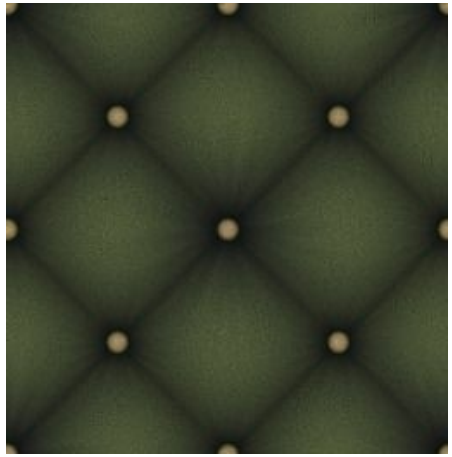
normal mapping

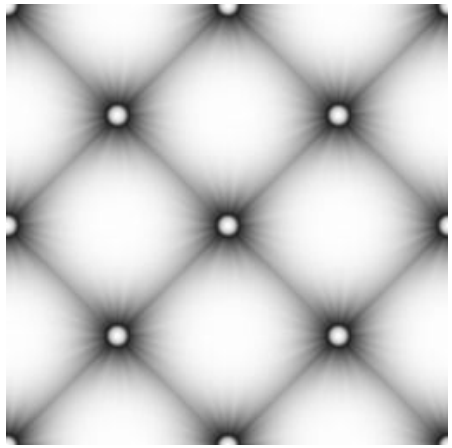


Modify surface appearance

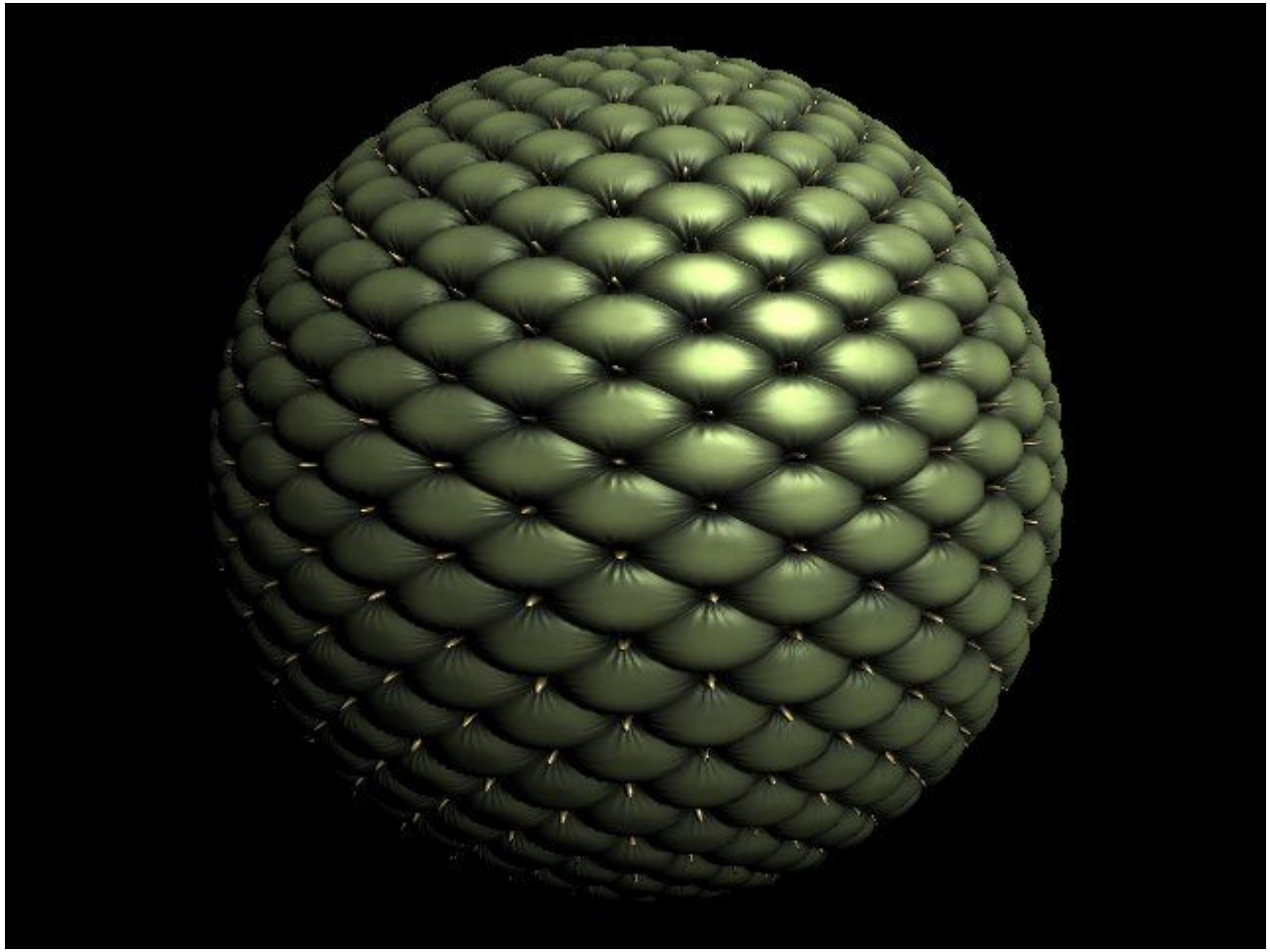
**** Displacement mapping ****

Goal: actually displace vertices!





displacement mapping



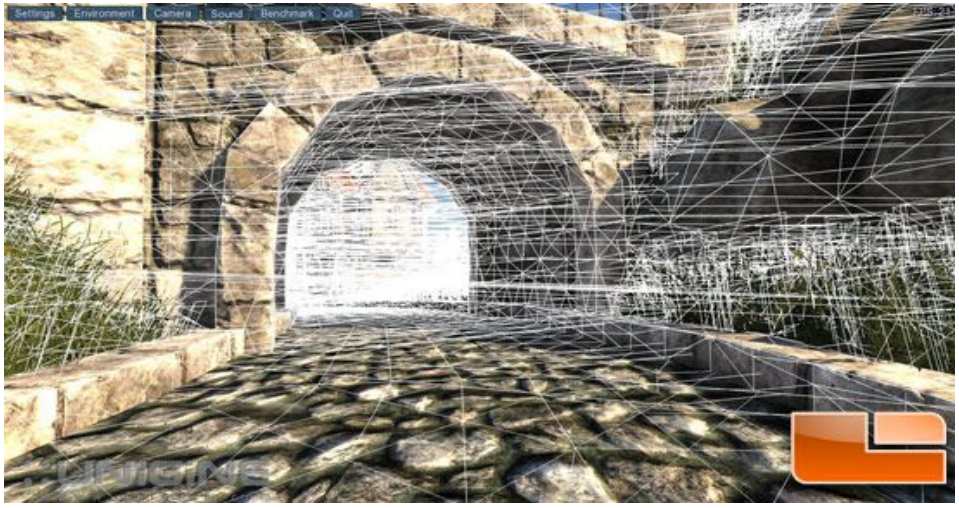
Modify surface appearance

**** Displacement mapping ****

Goal: actually displace vertices!

In practice:

- move vertices along their normals
- requires highly tessellated meshes
  - rely on adaptive tesselation
  - using the tesselation shader



Modify lighting effects

- Environment mapping
- Prefiltered environment maps
- Ambient occlusion
- Shadow mapping

Modify lighting effects

*** *Environment mapping*

***

Goal: realistic reflections/refractions

latitude-longitude map

cube map

light probe

Modify lighting effects

*** *Environment mapping*

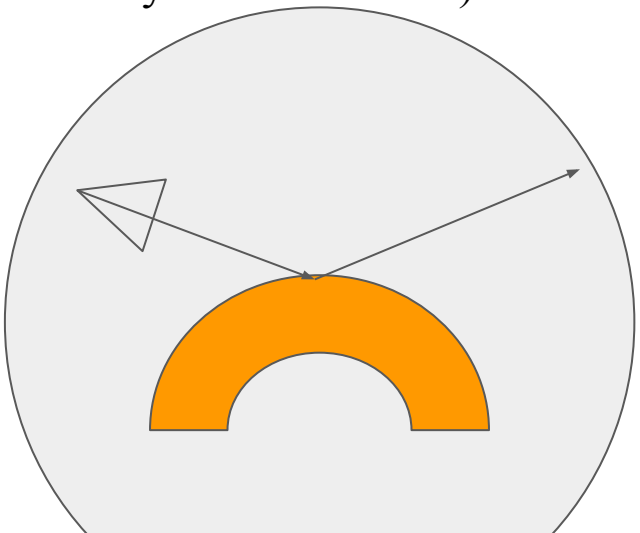
***

Goal: realistic reflections/refractions

In practice:

- Create the texture
  - acquired (using HDR photos of a mirrored sphere for instance)
  - or synthesized in real-time

- Using the reflected/refracted vector
  - compute the corresponding coordinates (spherical/cube/probe)
  - fetch texture color (basically 2 lines of code)

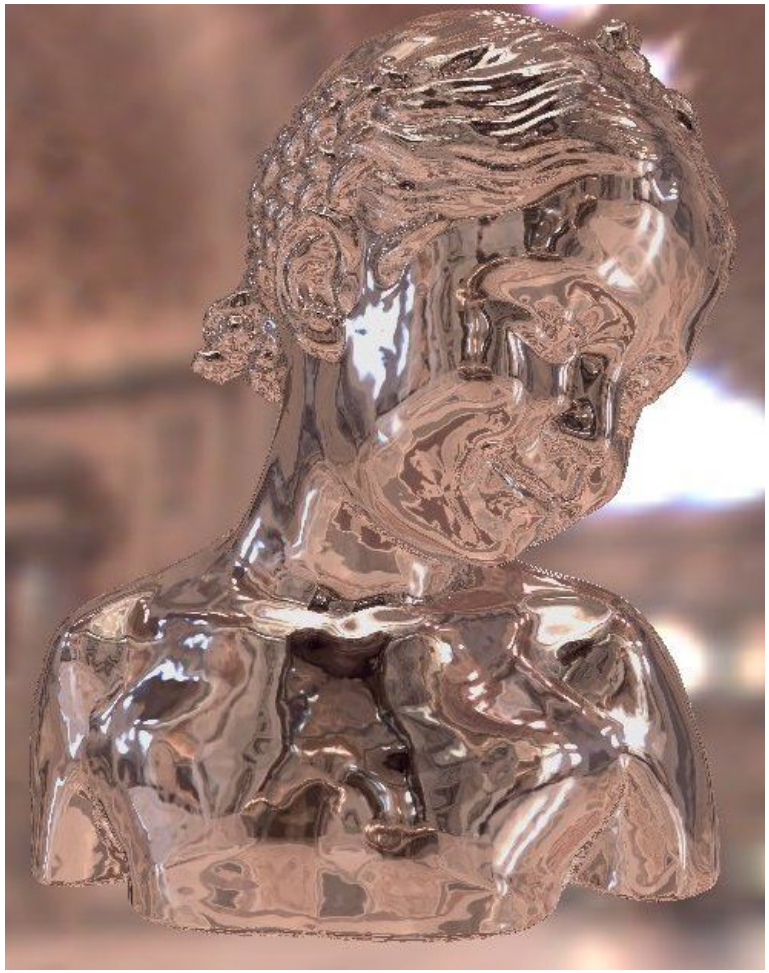


Modify lighting effects

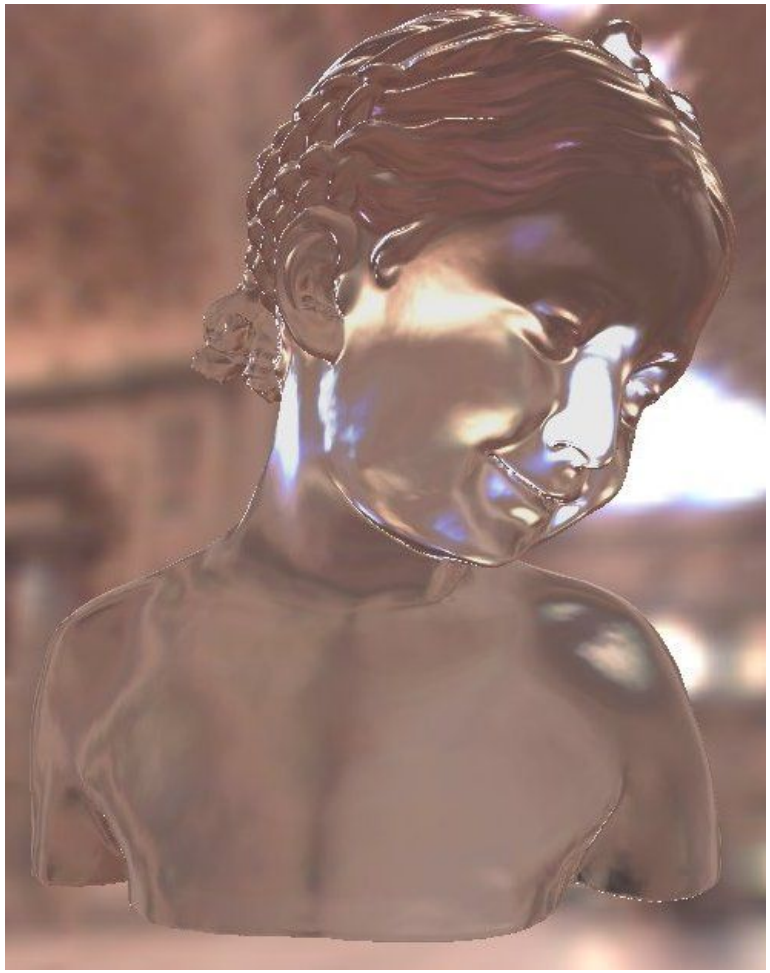
*** *Environment mapping*

***

Goal: realistic reflections/refractions



Reflection



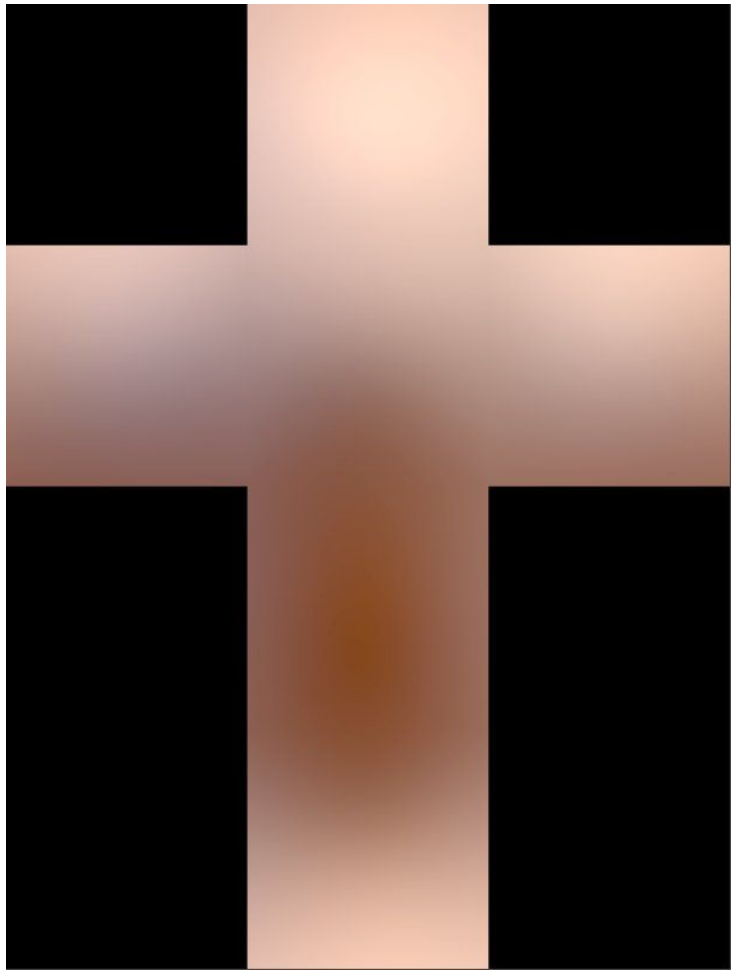
Refraction

Modify lighting effects

**** Prefiltered environment mapping ****

Goal: realistic glossy reflections/refractions, diffuse surfaces

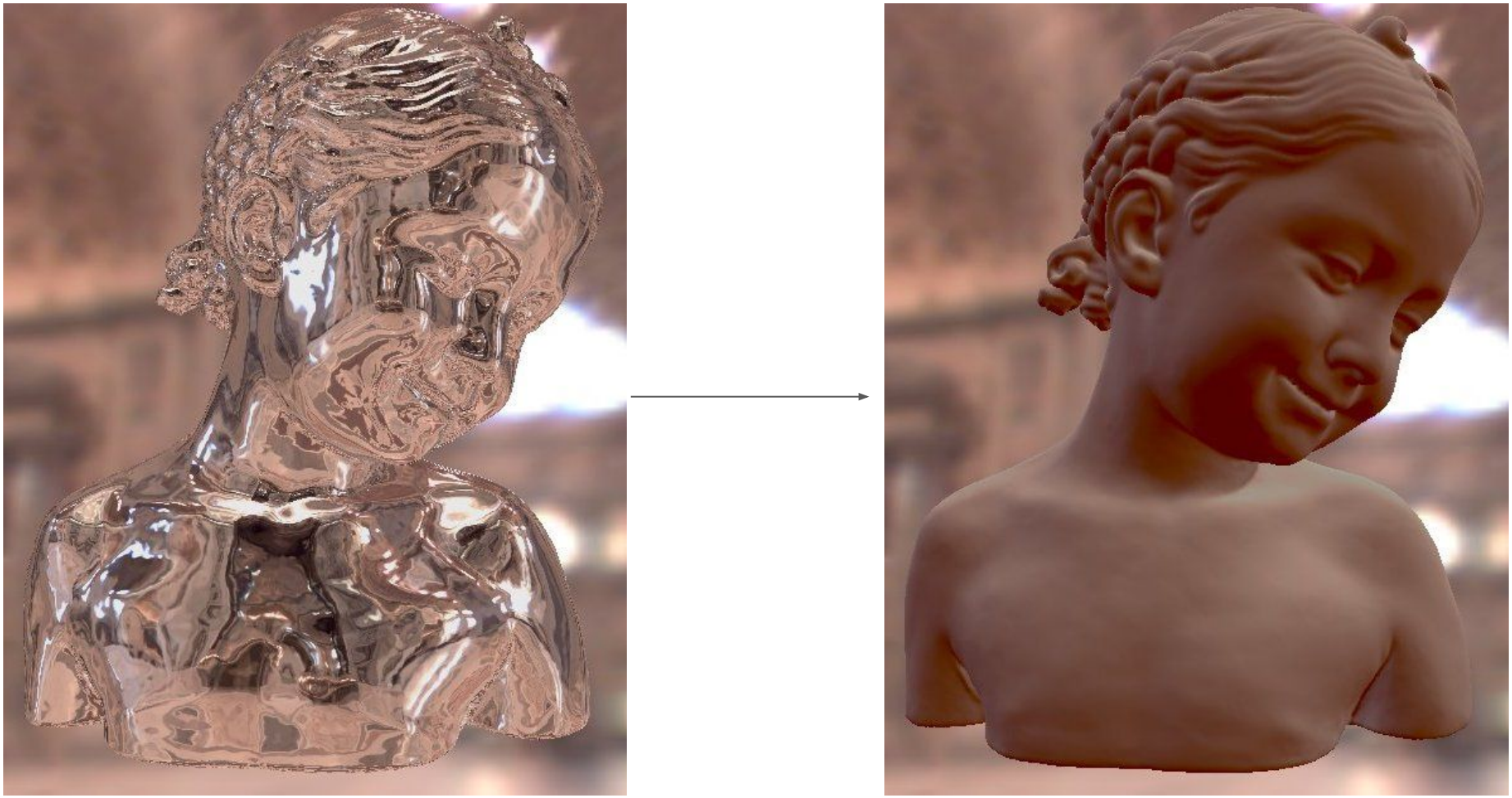
Convolution
(cosine lobe)



Modify lighting effects

**** Prefiltered environment mapping ****

Goal: realistic glossy reflections/refractions, diffuse surfaces



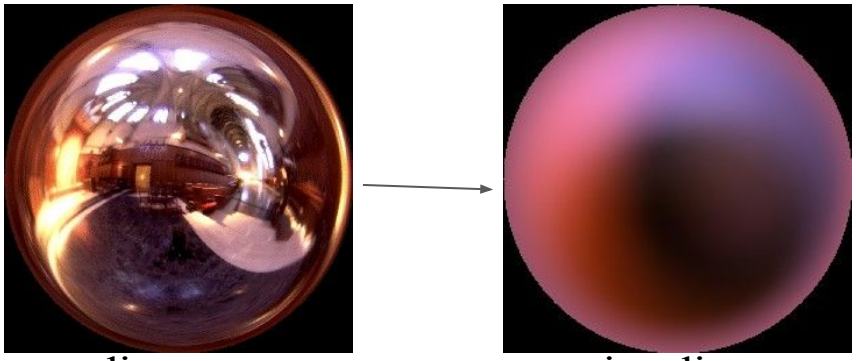
Modify lighting effects

**** Prefiltered environment mapping ****

Goal: realistic glossy reflections/refractions, diffuse surfaces

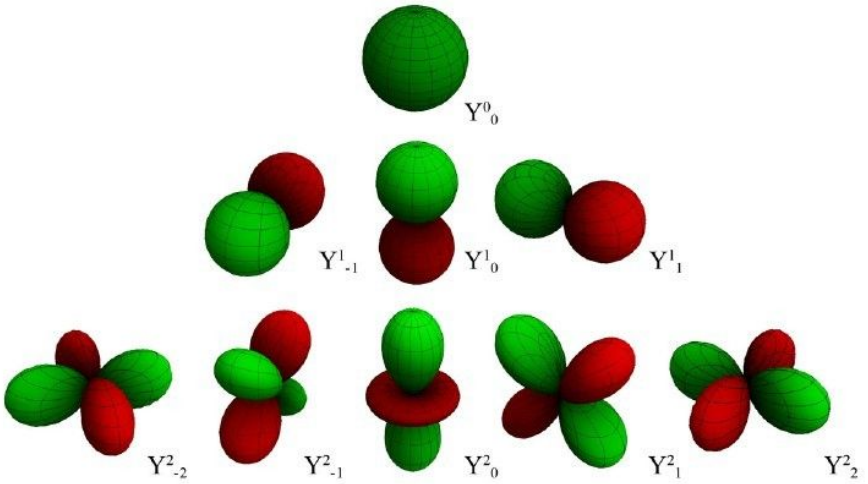
For pure diffuse surfaces:

- Low-frequency image



radiance irradiance

- can be represented with a few coefficients of refinable basis functions
  - e.g. spherical harmonics

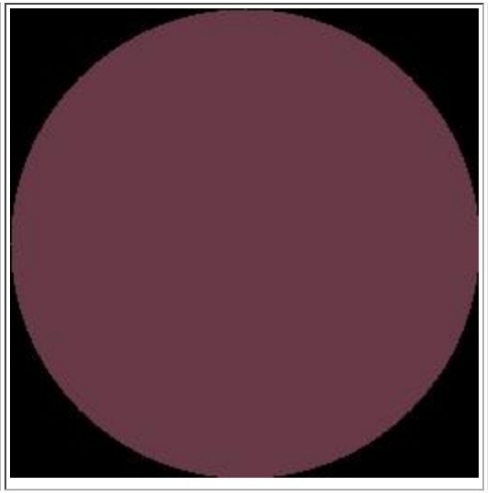


Modify lighting effects

**** Prefiltered environment mapping ****

Goal: realistic glossy reflections/refractions, diffuse surfaces

For pure diffuse surfaces:

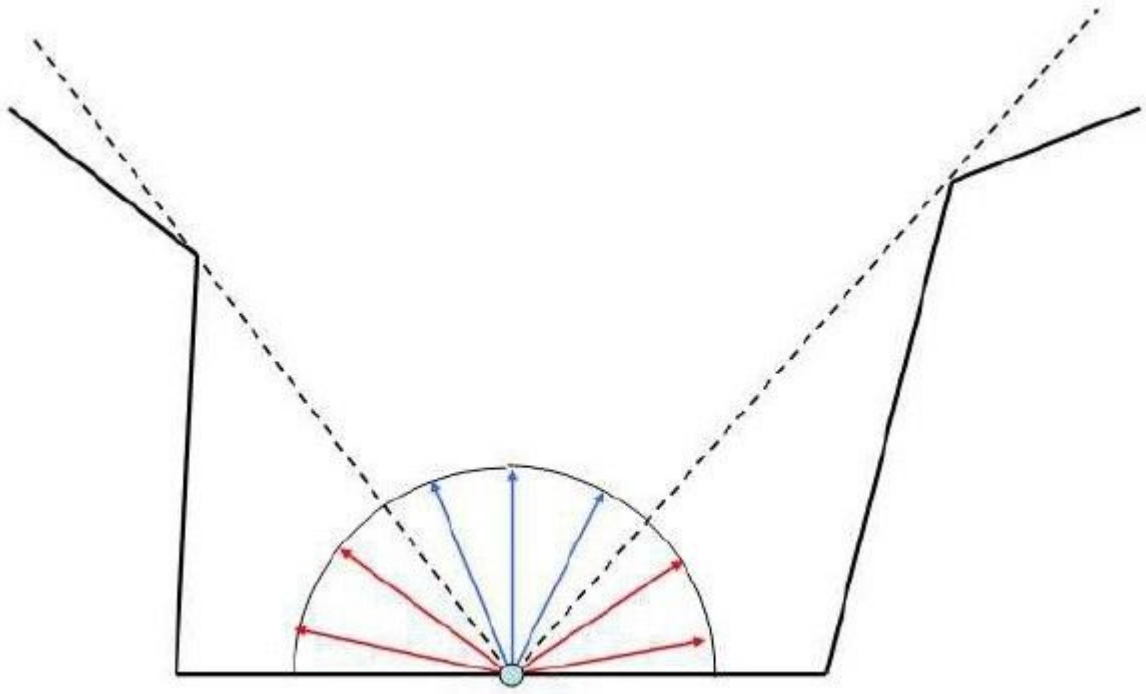
			
Original	Approx order 0 (1 coefs) RMS = 25%	Approx order 1 (4 coefs) RMS=8%	Approx order 2 (9 coefs) RMS=1%

- 9 coefficients sufficient
- can be pre-computed **for each vertex** (as attributes)! and evaluated in real-time

Modify lighting effects

**** ambient occlusion ****

Goal: compute (averaged) visibility at each surface point [Miller 94]



$$A = \frac{1}{\pi} \int V(\omega)(\mathbf{n} \cdot \omega) d\omega$$

Modify lighting effects

**** ambient occlusion ****

Goal: compute (averaged) visibility at each surface point [Miller 94]

autodesk.com

Original model | With ambient occlusion | Extracted ambient occlusion map

Modify lighting effects

**** ambient occlusion ****

Goal: compute (averaged) visibility at each surface point [Miller 94]

More and more screen-space solutions/approximations:

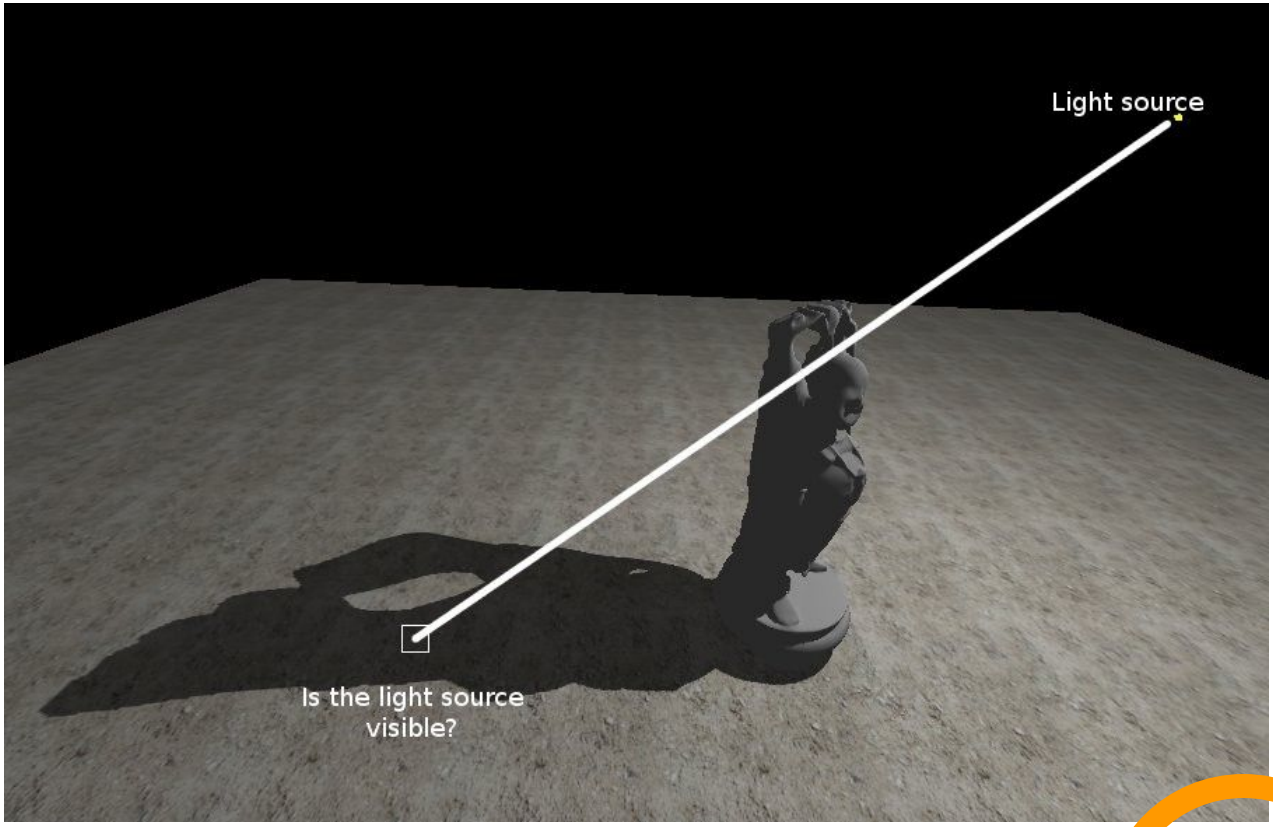
- allows dynamic scenes/anim/deformations
- based on depth + normal maps

https://www.youtube.com/watch?time_continue=27&v=-IFxjKT7MXA
https://en.wikipedia.org/wiki/Screen_space_ambient_occlusion
http://john-chapman-graphics.blogspot.fr/2013/01/ssao-tutorial.html

Modify lighting effects

**** Shadow mapping ****

Goal: create cast shadows



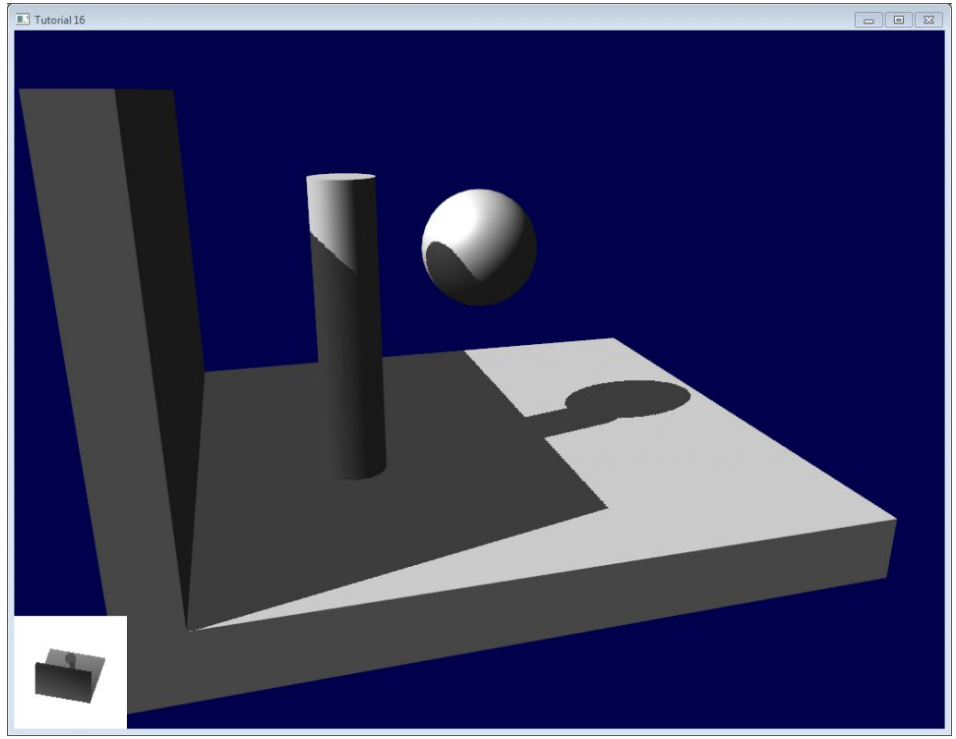
$$L(\mathbf{p} \rightarrow \mathbf{e}) = \rho_a L_a + \sum_k \rho(\mathbf{p}, \mathbf{e}, \boldsymbol{\ell}_k)\ (\mathbf{n} \cdot \boldsymbol{\ell}_k)\ V(\mathbf{p}, \boldsymbol{\ell})\ L(\mathbf{p} \leftarrow \boldsymbol{\ell}_k)$$

Modify lighting effects

**** Shadow mapping ****

Goal: create cast shadows

http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-16-shadow-mapping/

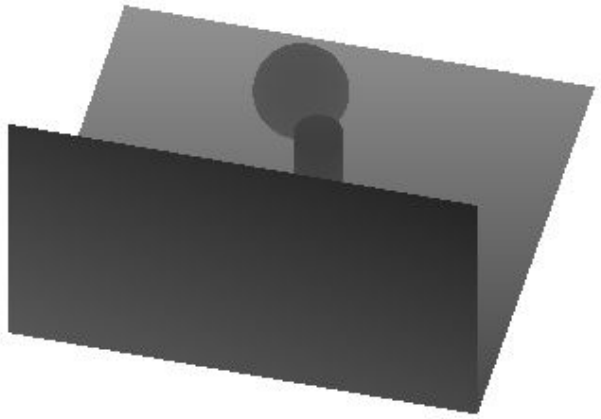


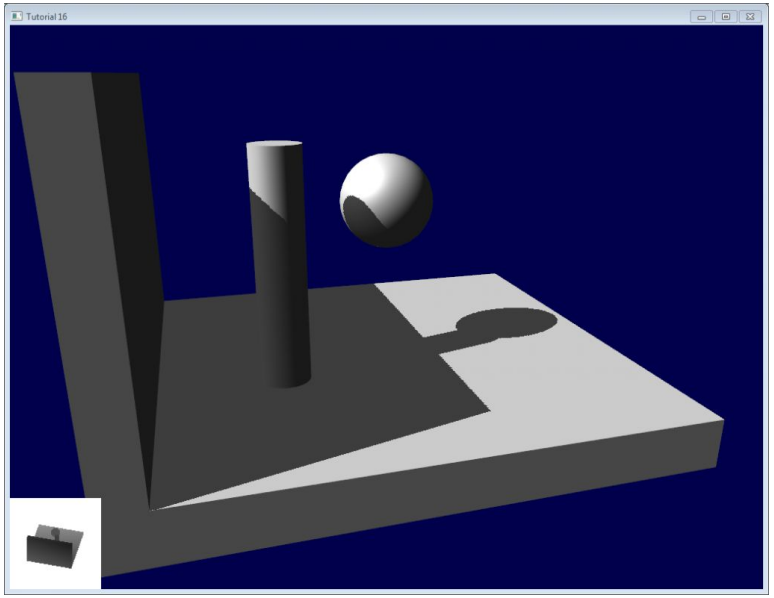
Modify lighting effects

**** Shadow mapping ****

Goal: create cast shadows

- draw scene from light
  - store depth in a texture
  - → called shadow map

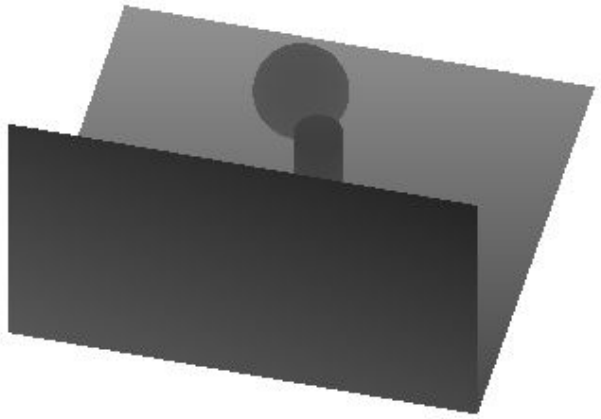




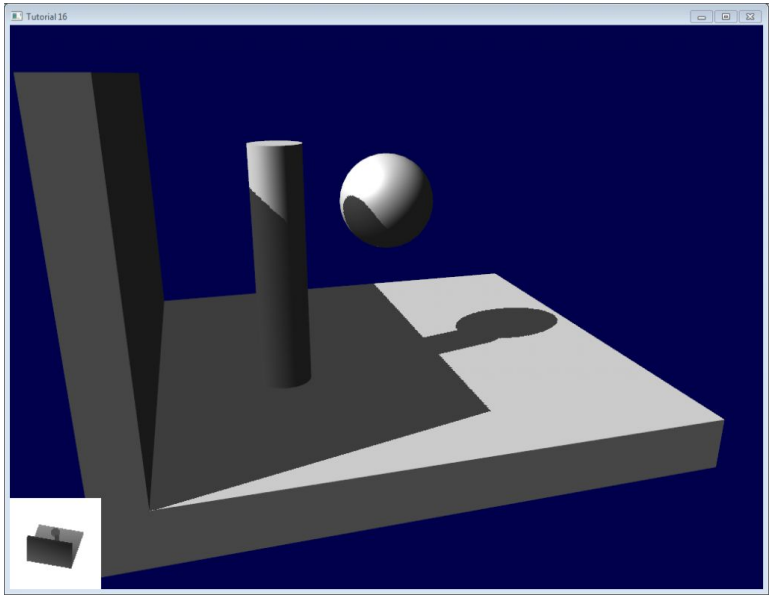
Modify lighting effects

**** Shadow mapping ****

Goal: create cast shadows



- 1: draw scene from light
  - store depth in a texture
  - → called shadow map



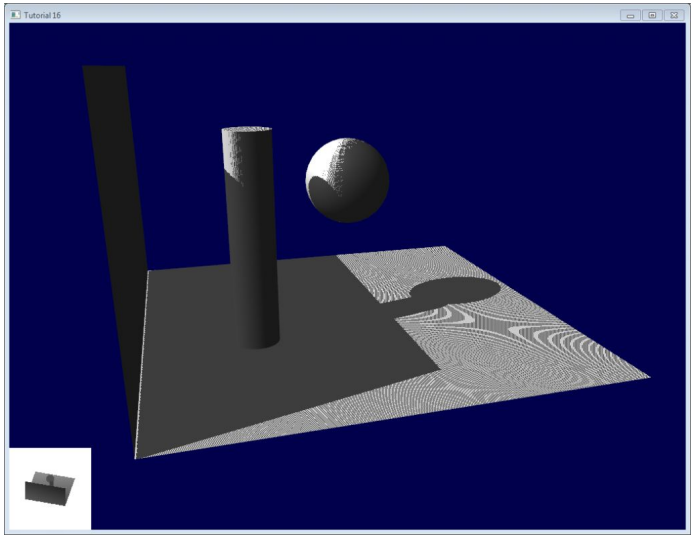
- 2: draw scene from camera
  - compute lighting (as usual)
  - project each point in the **light space**
  - compare depth with the one fetched in the shadow map
  - if farther: in the shadow!

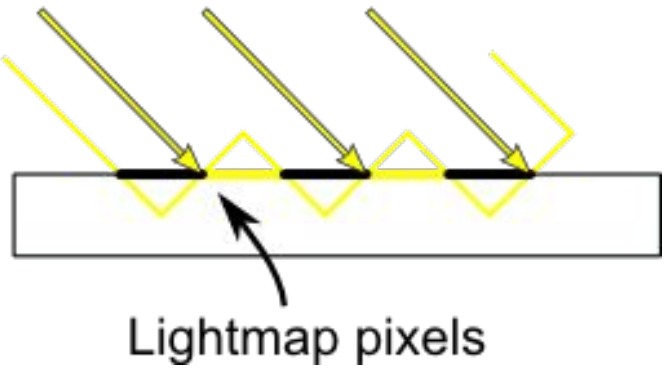
Modify lighting effects

**** Shadow mapping ****

Goal: create cast shadows

- Some issues:
  - acne effect → require small bias



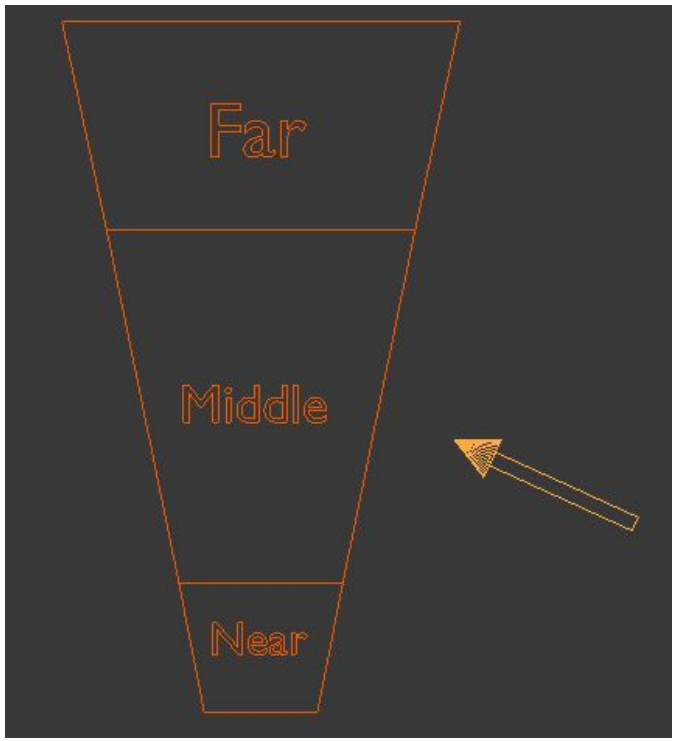


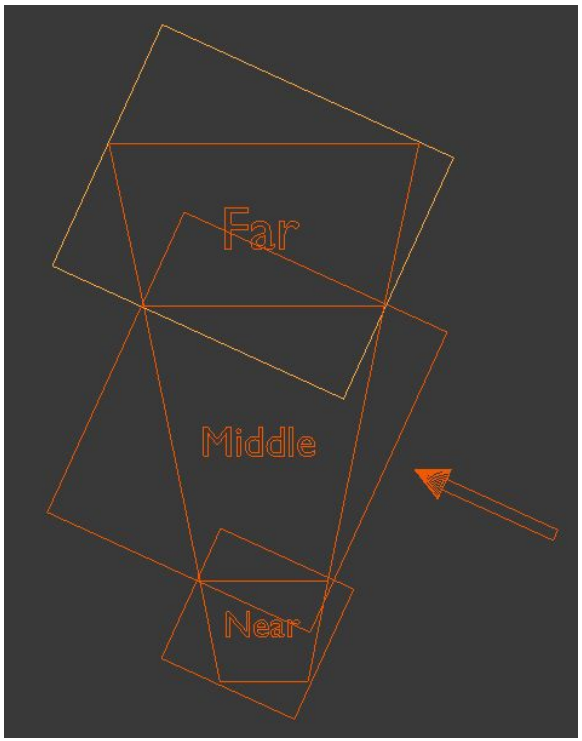
Modify lighting effects

**** Shadow mapping ****

Goal: create cast shadows

- Some issues:
  - acne effect → require small bias
  - shadow map resolution → cascaded shadow maps





http://ogldev.atspace.co.uk/www/tutorial49/tutorial49.html

Playing with images

Plenoptic function [Adelson and Bergen 91]

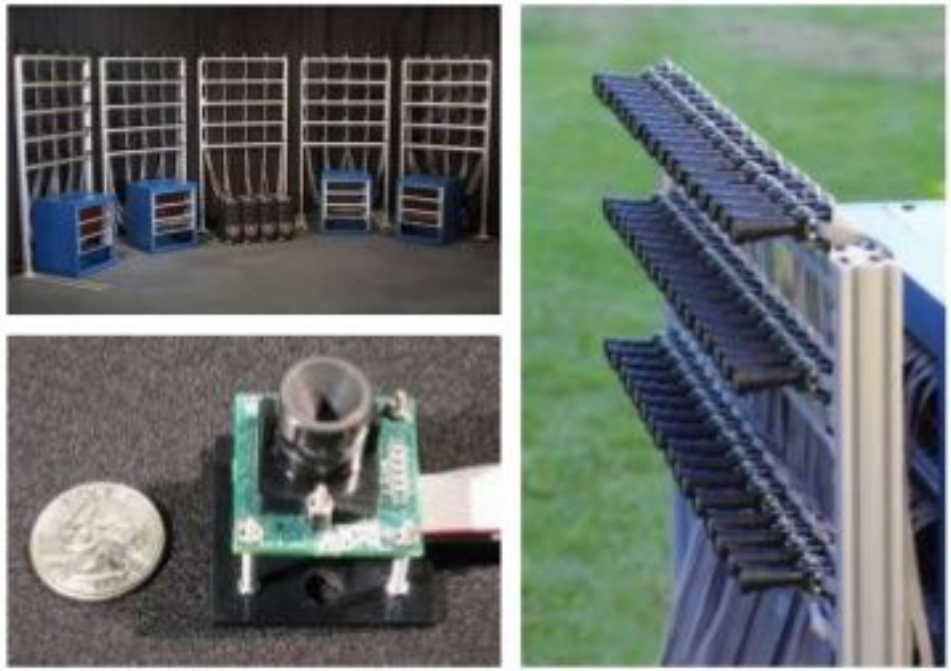
$$P(\theta, \phi, \lambda, t, V_x, V_y, V_z)$$

- P defines the intensity as a function of viewpoint, time, wavelength
- capture all possible images around p
- → Image Based Rendering (IBR) = reconstruct P from samples

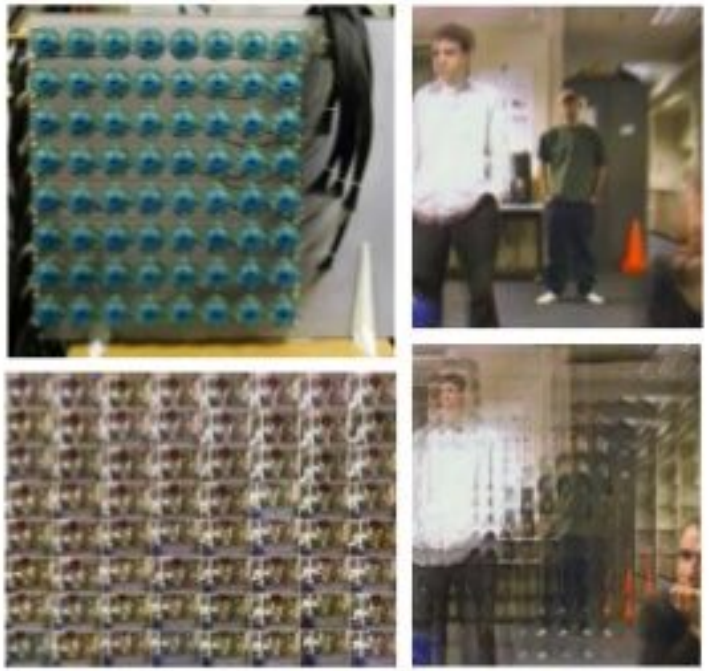
Playing with images

**** light fields ****

Example: light field acquisition



Stanford Multi-Camera Array
125 cameras using custom hardware
[Wilburn et al. 2002, Wilburn et al. 2005]



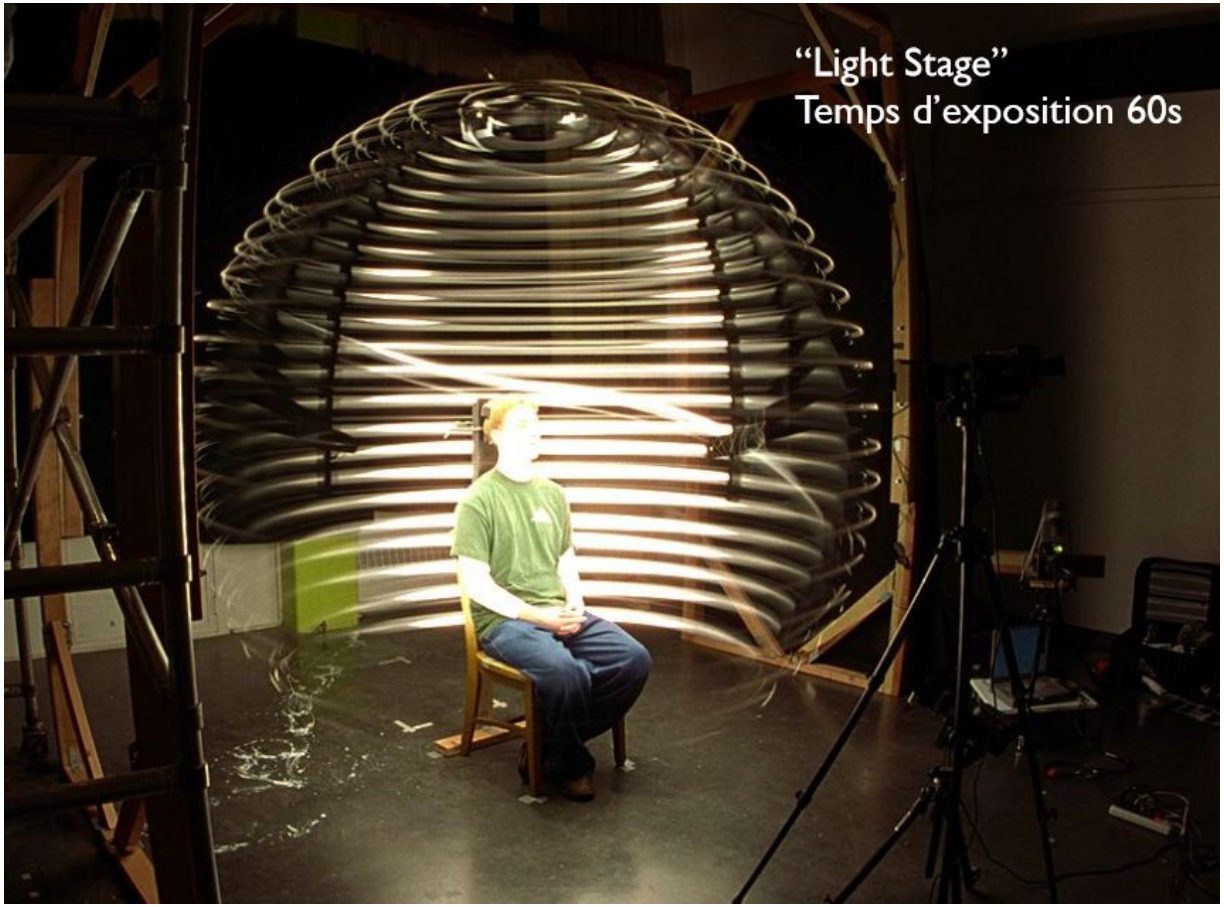
Distributed Light Field Camera
64 cameras with distributed rendering
[Yang et al. 2002]

refocusig example: http://lightfield.stanford.edu/lfs.html

Playing with images

*** *relighting* ***

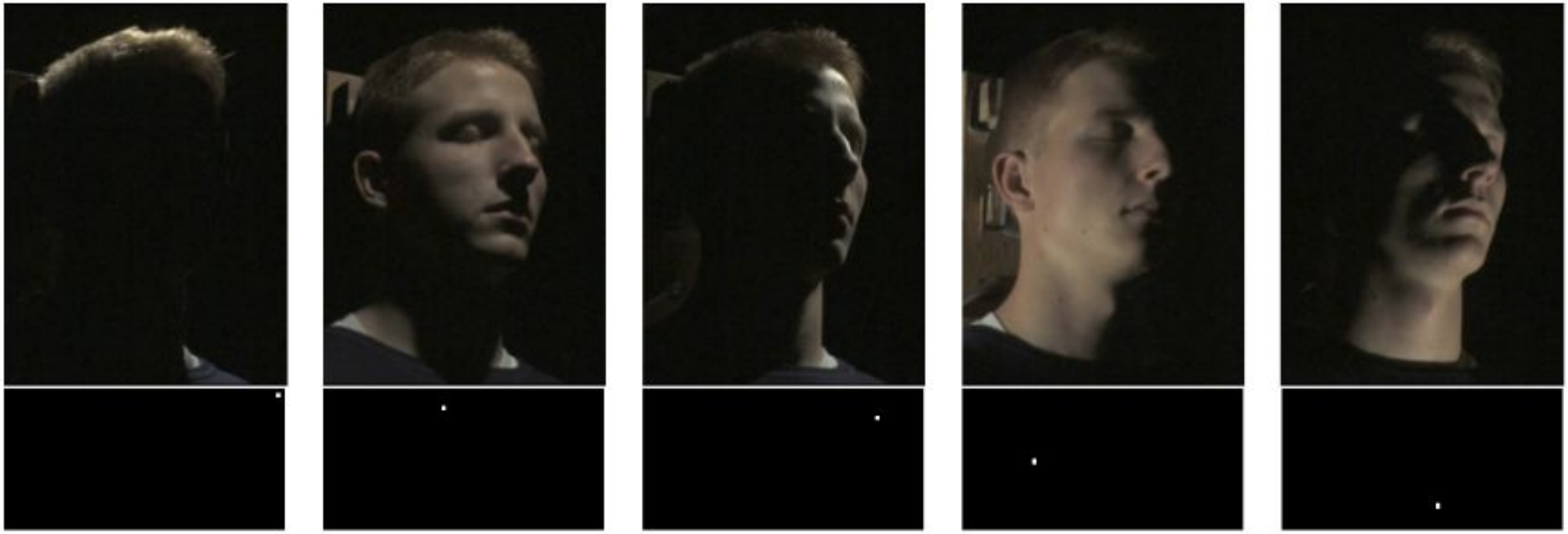
Example: relighting



Playing with images

**** relighting ****

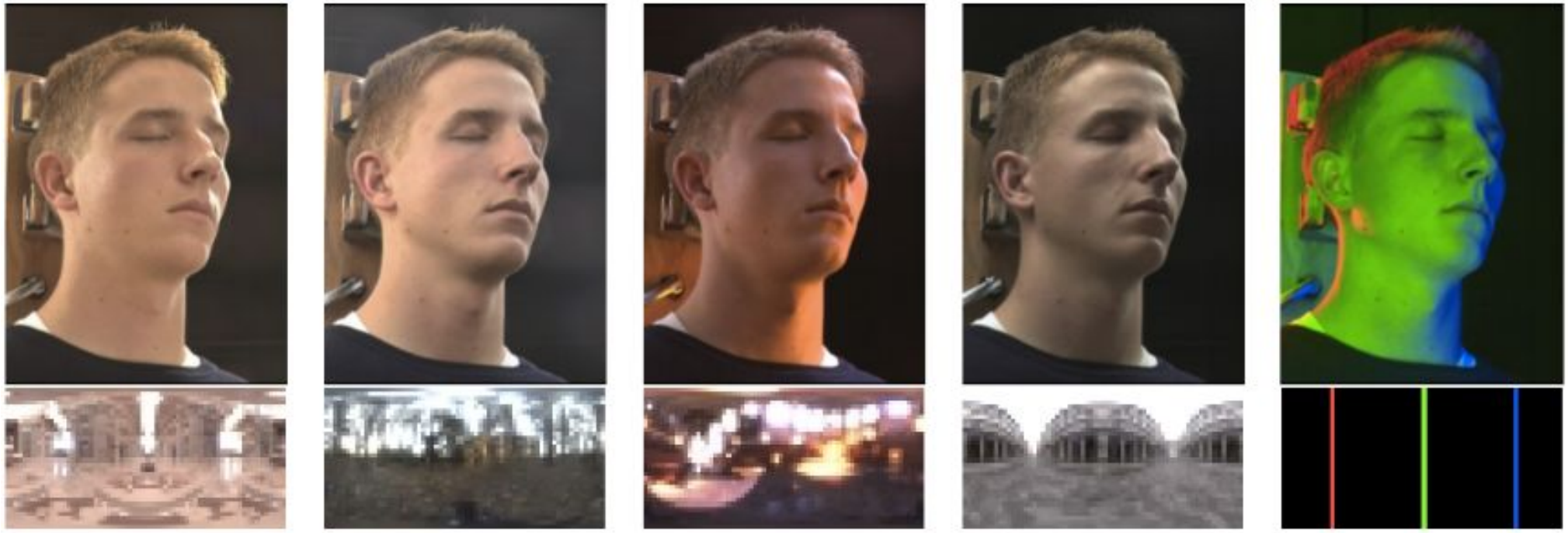
Example: relighting



Playing with images

**** relighting ****

Example: relighting



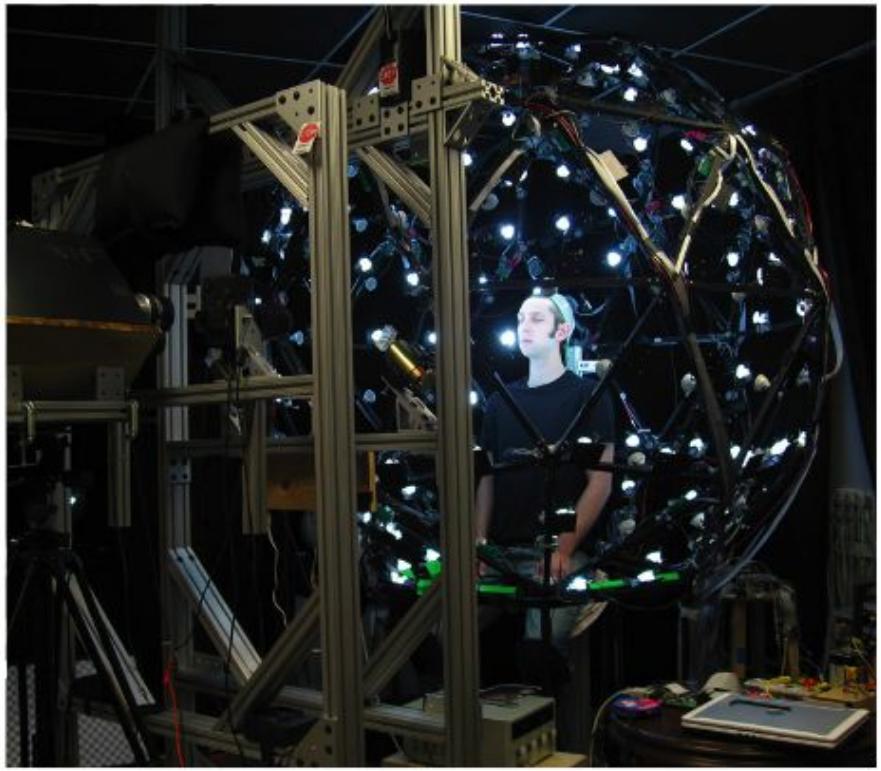
Playing with images

**** relighting ****

Example: relighting

The Curious Case of Benjamin Button

Avatar



more: https://www.youtube.com/watch?v=piJ4Zke7EUw