Computer Graphics

Course outline

- Introduction
- Images and display techniques
  - Bases
  - Gamma correction
  - Aliasing and techniques to remedy
  - Storage

Course outline

- 3D Perspective & 2D / 3D transformations
  - Go from a 3D space to a 2D display device
- Two paradigms for image synthesis
- Representation of curves and surfaces
  - Splines & co.
  - Meshes
- Realistic rendering by ray tracing
  - Concepts and theoretical bases

Course outline

- Introduction
- Images and display techniques
  - Bases
  - Gamma correction
  - Aliasing and techniques to remedy
  - Storage

Course outline

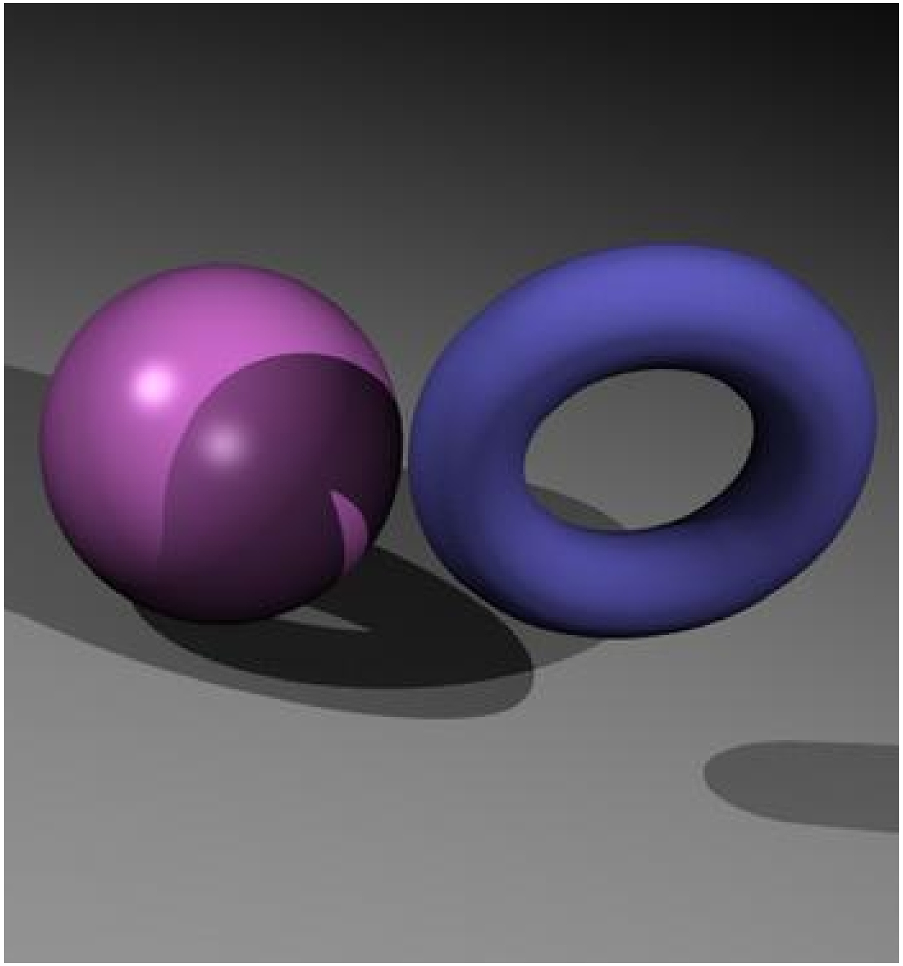
- Lighting
  - Law of reflexion, Textures
- Colorimetry
  - Color space
  - Metamerism
- Graphic pipeline and OpenGL
  - Primitives
  - Discretization (*Rasterization*)
  - Hidden faces
- Animations ?

Ray tracing

- Basic ray tracing
  - One ray by pixel
  - One shadow ray by point source of light
  - One reflected ray, possibly a refracted ray, by intersection

Ray tracing

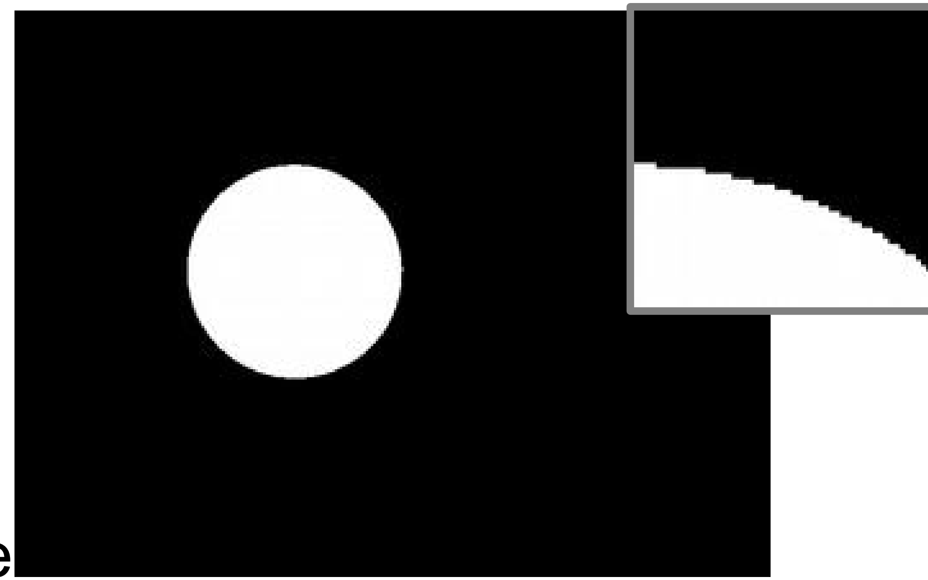
- Discontinuous appearance
  - Perfectly clear silhouette and infinite field depth (clear image from 0 to infinite)
    - Cause : the camera's pupil is like a pinhole
  - Sharp shadows
    - Cause : pointwise sources of light
  - Perfectly clear mirror reflections
    - Cause : infinitely smooth surfaces
  - Presence of aliasing
    - Cause : from a pixel to another, abrupt changes in brightness
    - Aliased objects

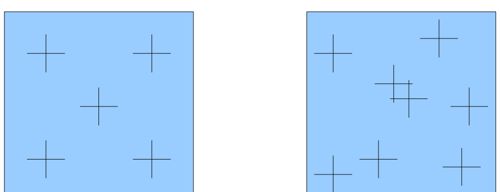


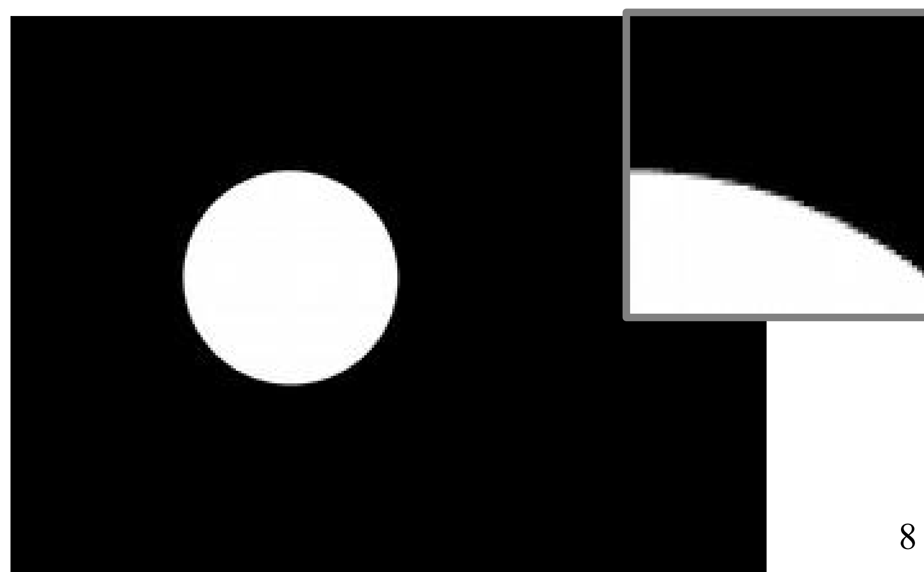
- Modeling imperfections is difficult !
  - Diffuse shadows
  - Depth of field
  - Partially diffuse reflections
    - Imperfect specular surfaces

Ray tracing

- Antialiasing
  - Oversampling : for each pixel, take the average of several rays slightly shifted (5, 8 or 16 ...)
  - Increases rendering time

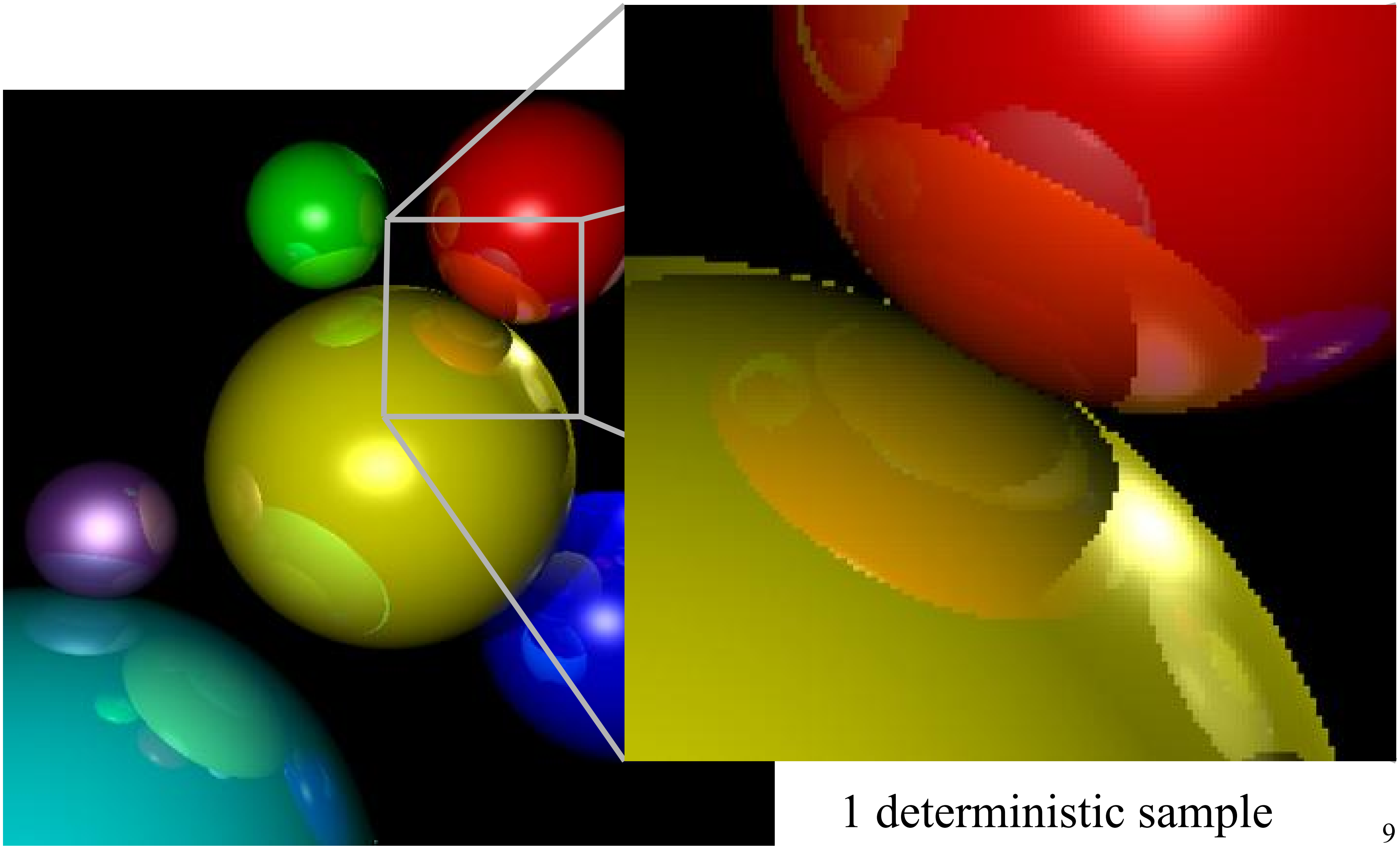






  - Randomize position to avoid any Moiré effect

Ray tracing

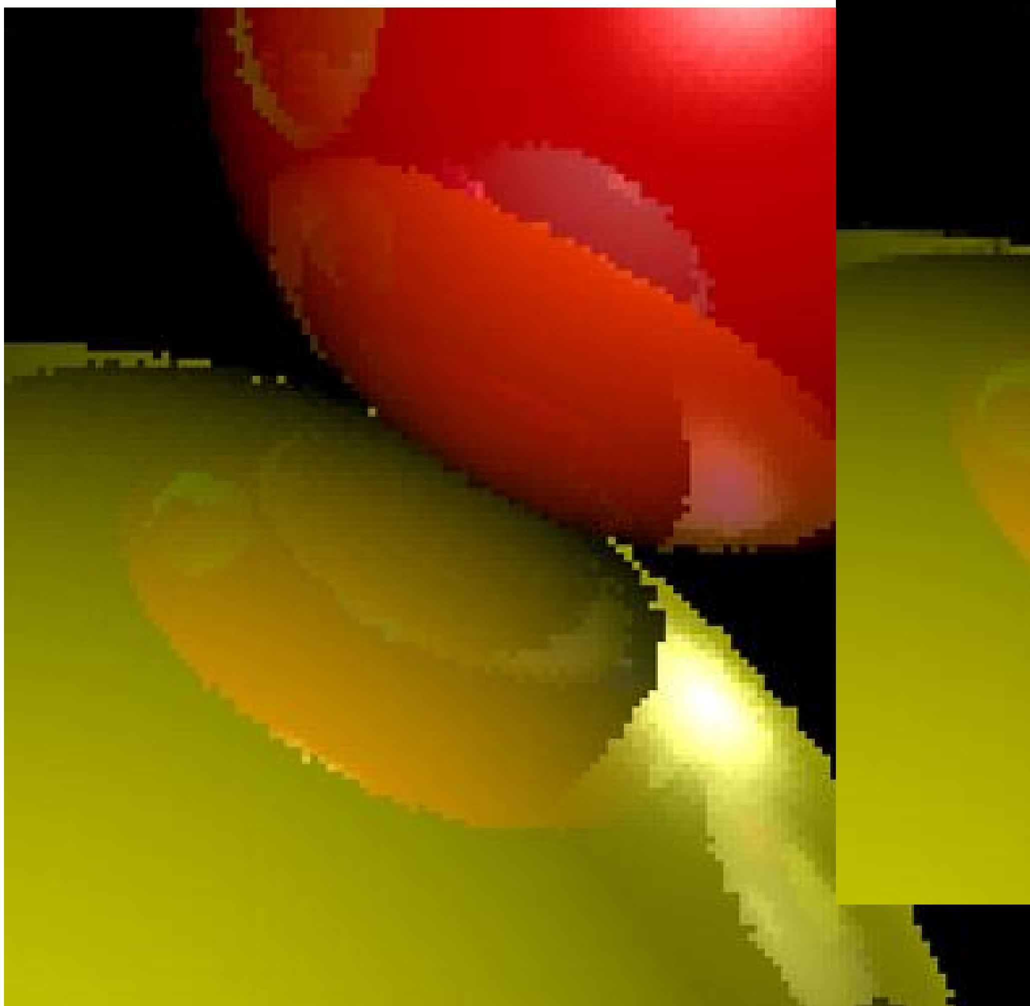


1 deterministic sample

Computer Graphics

Ray tracing

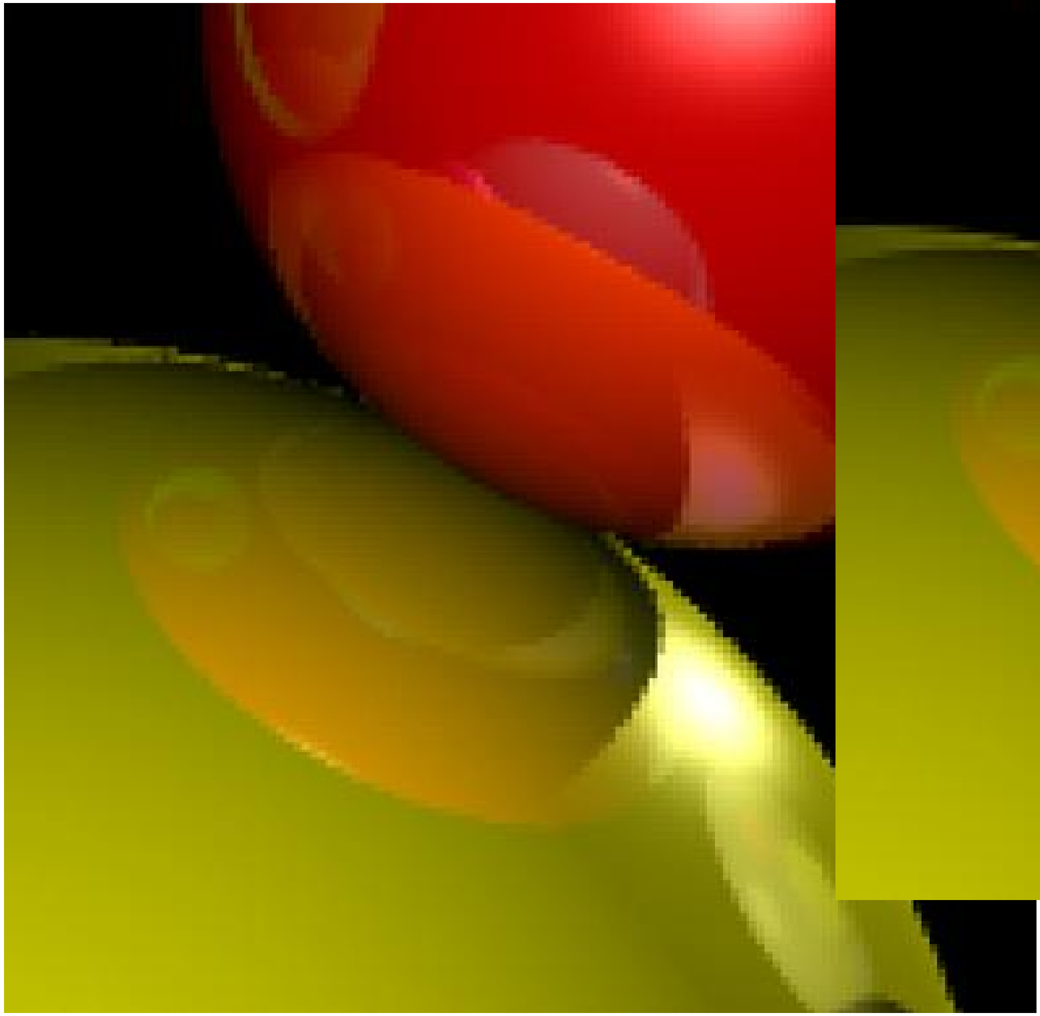
1 random sample

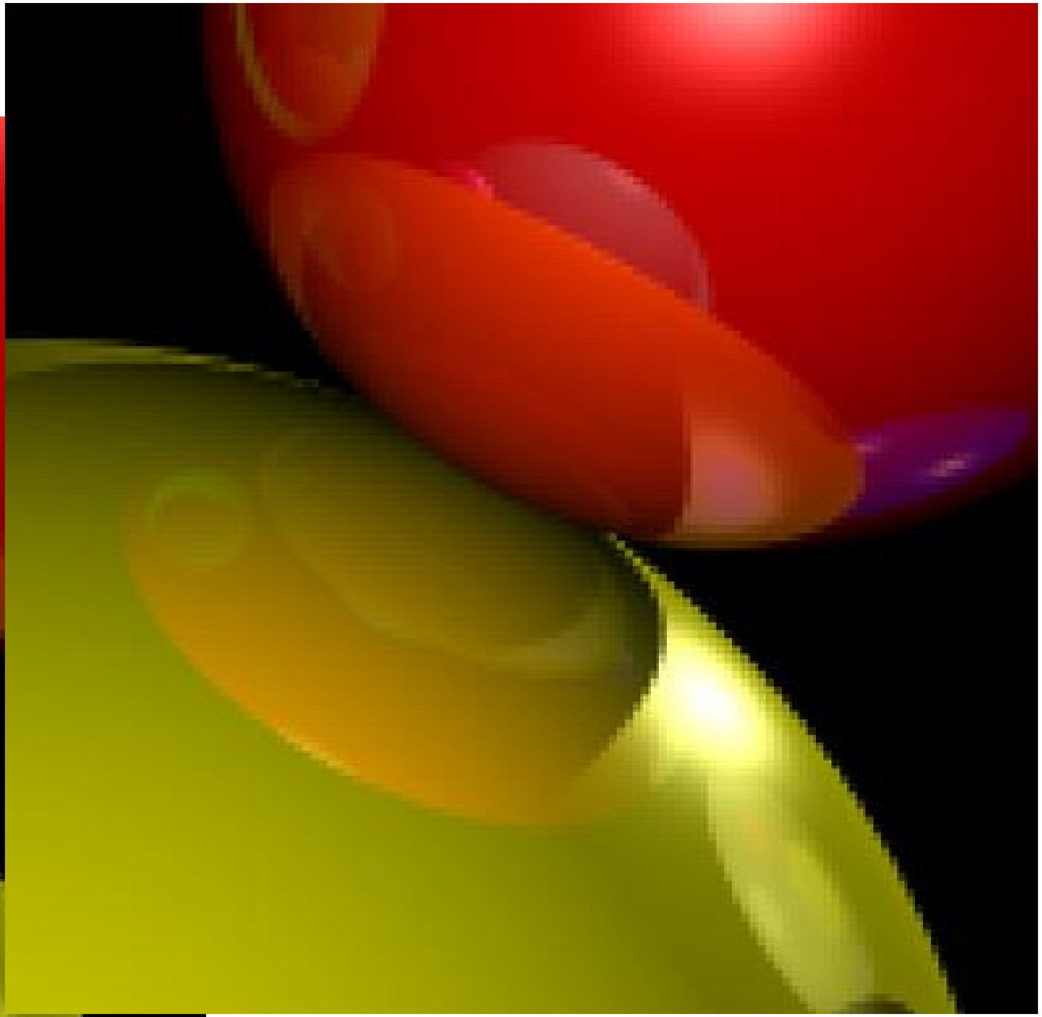


Oversampling on 10 samples

Ray tracing

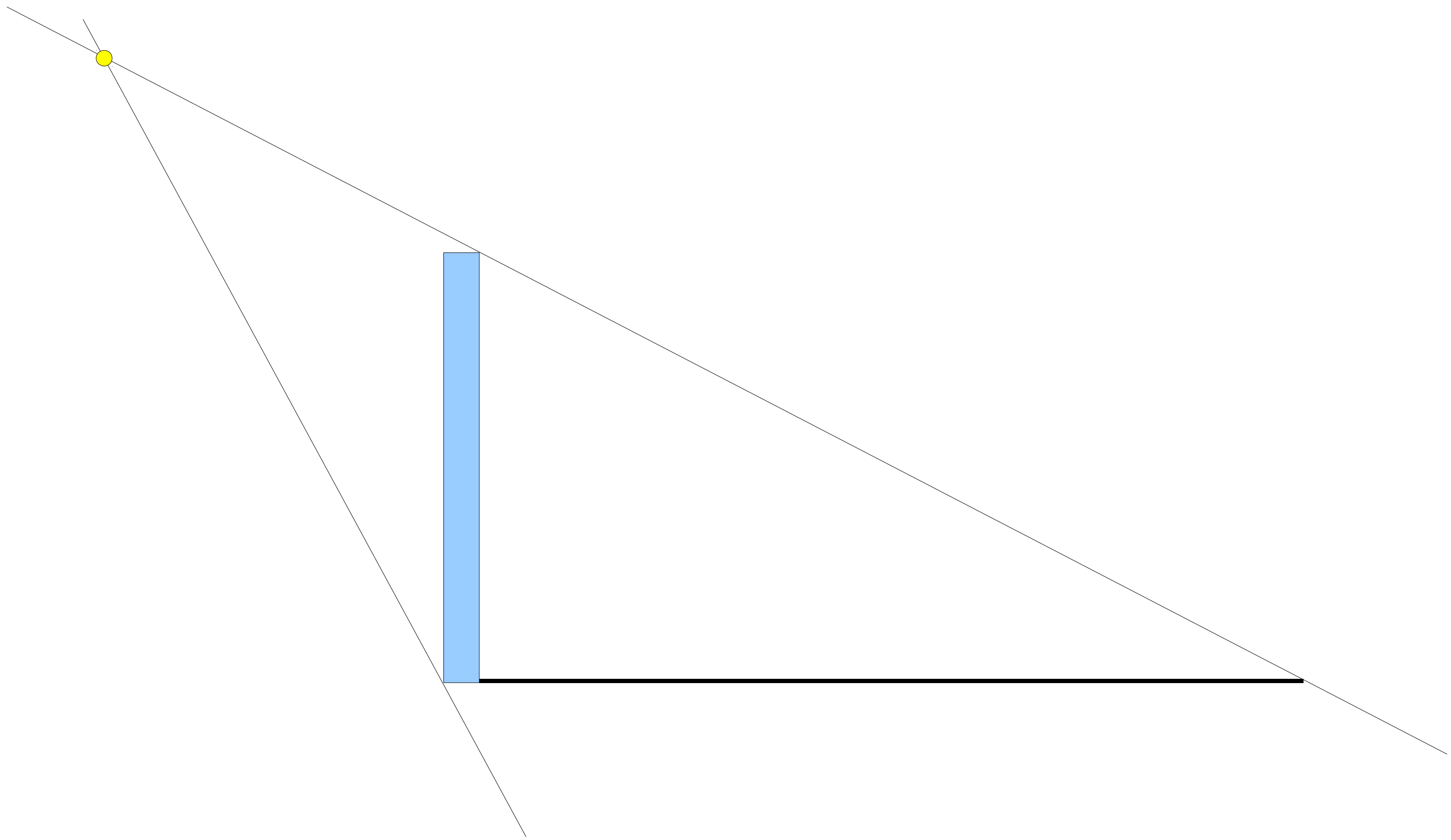
Oversampling on 10 samples



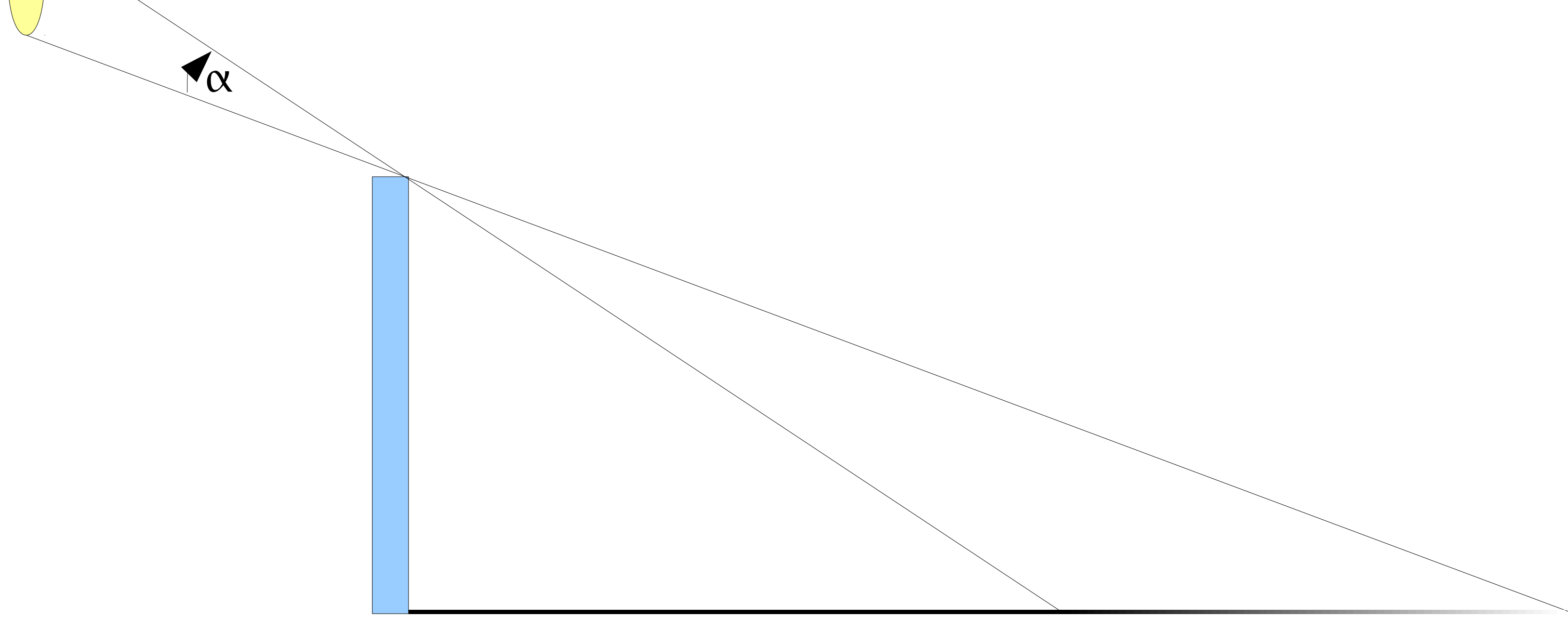


Oversampling on 100 samples

- Diffuse samples vs sharp shadows



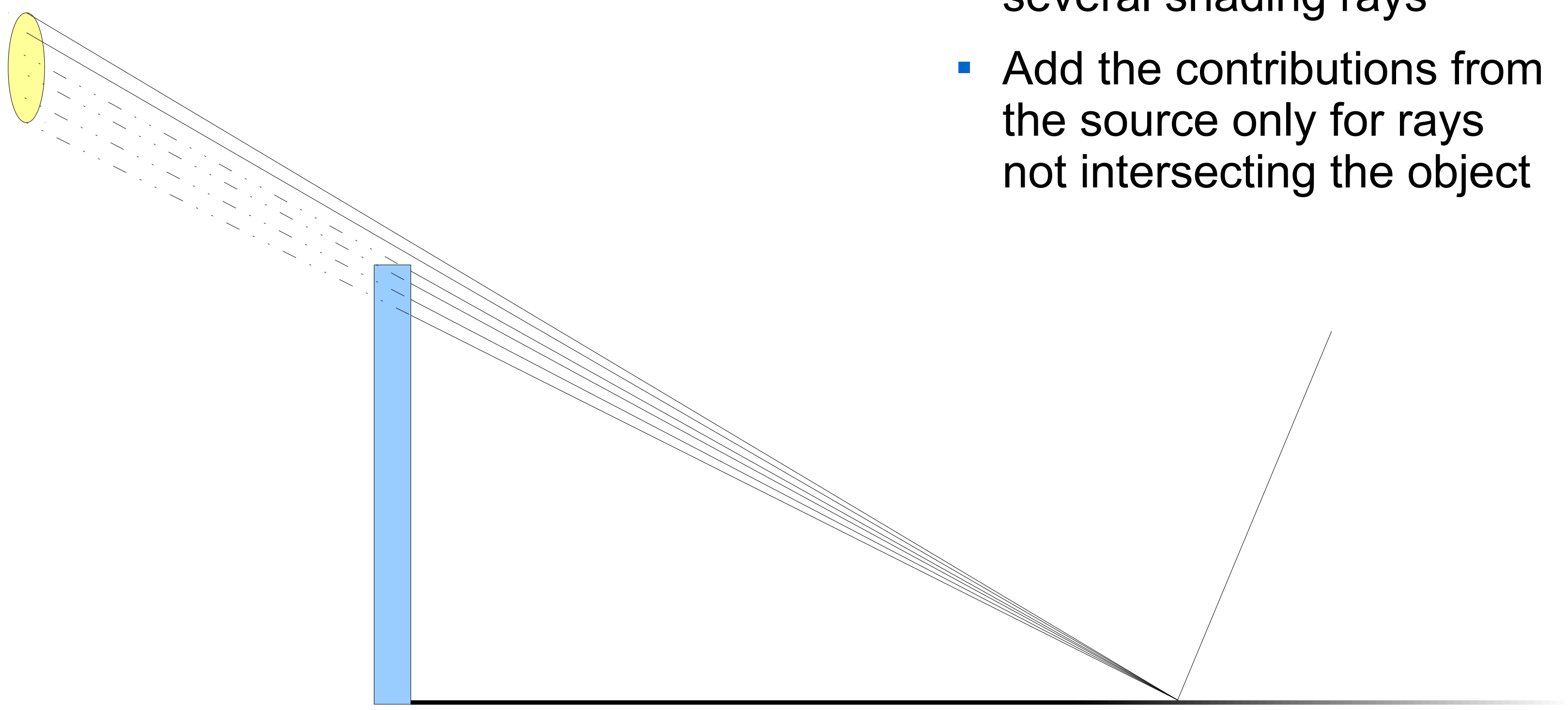
- Diffuse shadows vs sharp shadows
  - Apparent diameter of the sun (and of the moon) : α=0.53°



Computer Graphics

Lancer de rayon

- Diffuse shadows



For each point, calculate several shading rays

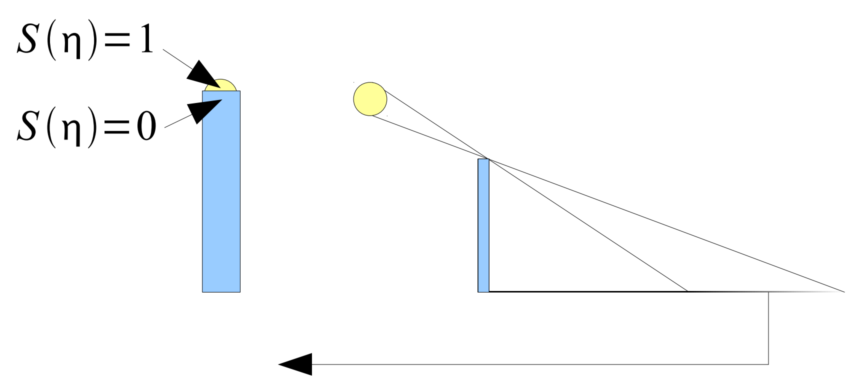
- Add the contributions from the source only for rays not intersecting the object

- Amounts to decompose the light source into several, slightly offsetted in space, pointwise light sources of lower intensity

Ray tracing

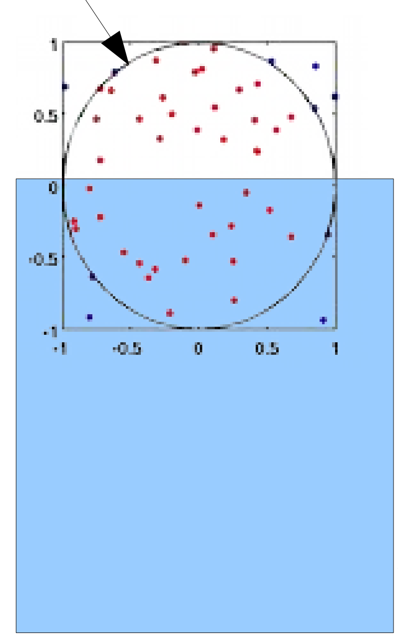
- Two approaches
  - Place additional light sources and perform rendering
    - Problem : it takes a lot of such sources to achieve a realistic result
    - The sources are positioned once and for all ...
  - Use a sampling technique ...
    - … such that for each point, the fraction of light source not hidden by objects is approximately calculated by evaluating the following integral :

$$I = L \iint_{\Omega} S(\eta)\, d\eta$$



- Monte Carlo integration
  - One wishes to compute approximately $\iint_\Omega f(\eta)\,d\eta$
  - We set

$$\iint_\Omega f(\eta)\,d\eta \approx \frac{1}{N\,\text{meas}(\Omega)} \sum_{i=1}^{N} f(\eta_i) \qquad \text{meas}(\Omega) = \frac{4}{\pi}$$



  - We choose the points η_i in a pseudo-random way (in a canonical setting)
    - For each η_i we verify that we are in Ω, and if this is the cas, we verify that the source is visible, If these conditions are met it returns 1, otherwise 0.
    - The factor $\text{meas}(\Omega)$ is calculated so as to obtain 1 if the light source is totally visible.

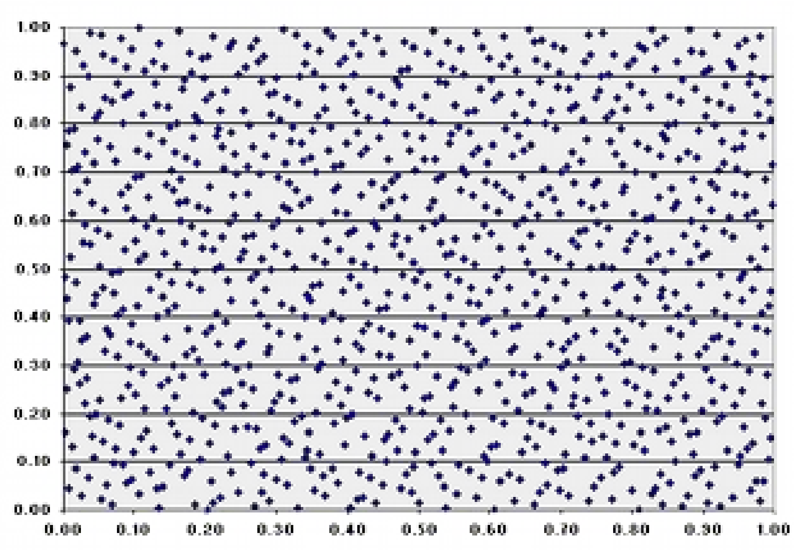
- Characteristics of the Monte Carlo integration
  - No regular grid
  - If, for N points, the result is not accurate enough, it is easy to use 2N points without losing the calculations already made
  - Convergence rate
    - Random sequence: $\frac{1}{\sqrt{(N)}}$
    - "Low variance" sequence (Quasi-Monte-Carlo): $\frac{1}{N}$ (in practice)

$\frac{(\ln N)^d}{N}$ (worste cas)

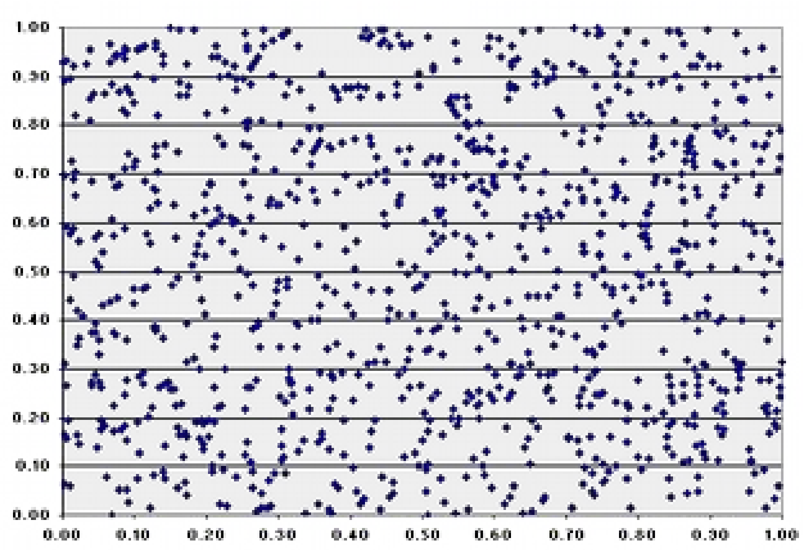
		Monte Carlo	Quasi-Monte Carlo	
			Best	Worse
d	N	$1/\sqrt{N}$	1/N	$\ln(N)^d/N$
1	1,000	0.03162	0.00100	0.00691
1	100,000	0.00316	0.00001	0.00012
2	10,000	0.01000	0.00010	0.00848
5	10,000	0.01000	0.00010	6.628
10	10,000	0.01000	0.00010	479945.5
50	100,000	0.00316	0.00001	1.14626E+48

OBS: without reduction of variance techniques for both MC and QMC

- Sequences used for the MC integration

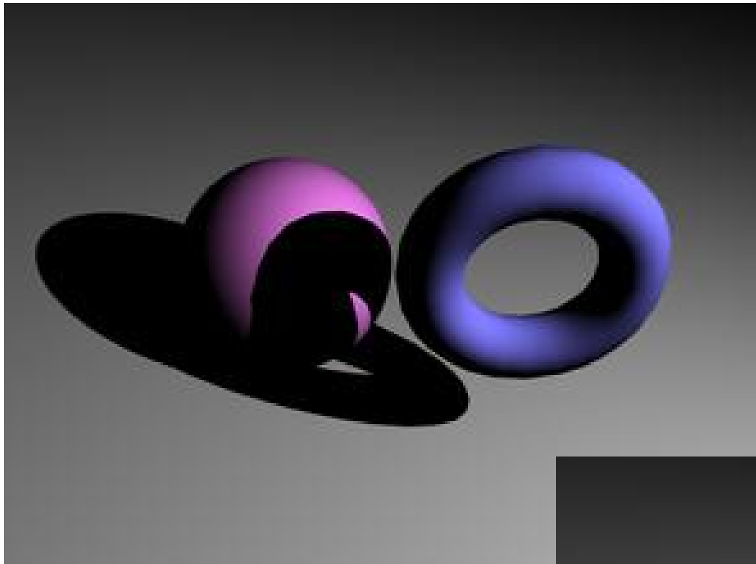


Quasi-random low variance sequence obtained by clustering

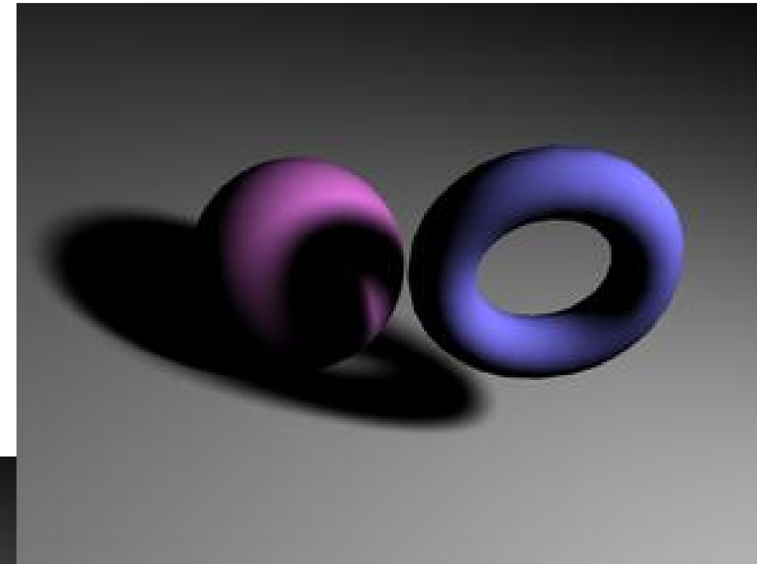


Random sequence

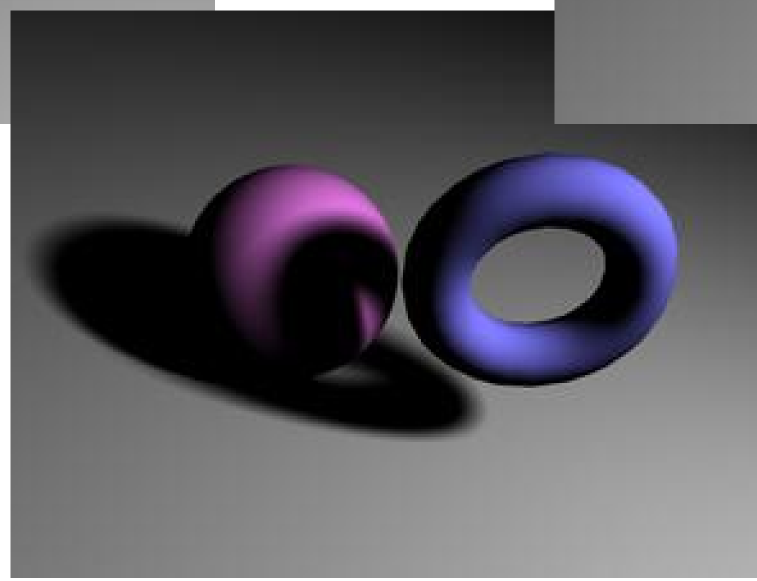
Ray tracing



1 source, intensity I



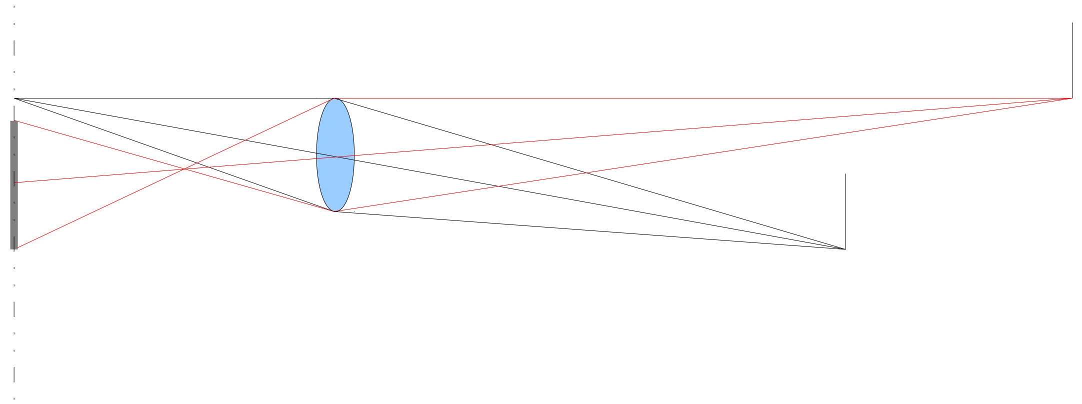
8 samples with Quasi-Monte-Carlo integration



40 fixed sources I_i=I/40

Ray tracing

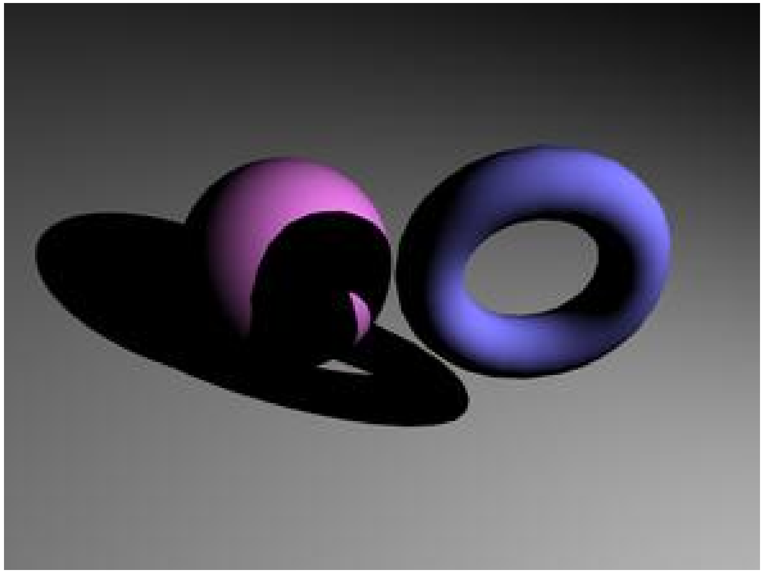
- Depth of field
  - The camera lense, eye, etc. .. cannot produce a sharp image from 0 to infinity.
    - Geometrically, only objects on a given plane are sharp.
  - Important concept !
    - Qualitative distance information
    - Bring what is important out (eg. portrait on blurred background)

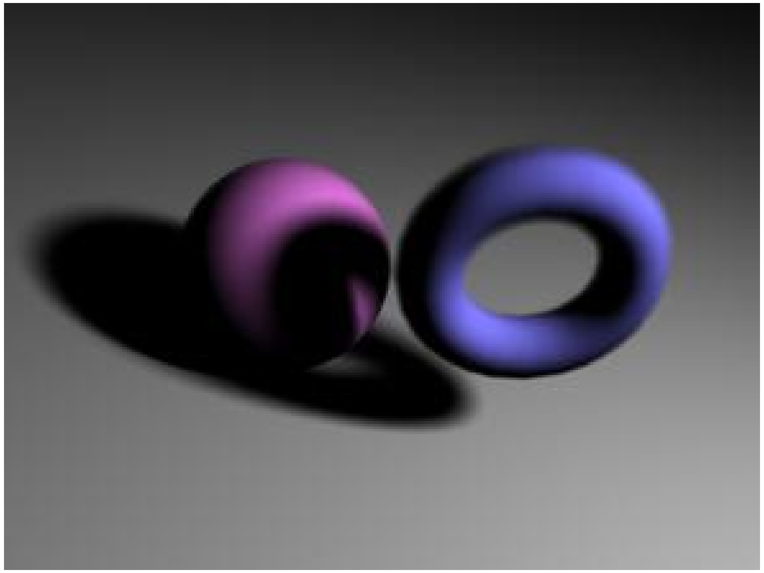


Ray tracing

- Three techniques to account for the depth of field:
  - By Post-processing
    - Calculate the image without changing...
    - At each point of the image, one has an idea of the depth (Z-buffer)
    - With this z variable, blur the image starting with points far away and proceed toward the observer
    - This is done is in Blender
    - Not very accurate and frequently, artifacts are visible
  - Successive rendering
    - Several complete renderings with camera positions slightly shifted, which are then merged
    - Similar to the calculation of shadows with n stationary sources – slow !
  - Monte Carlo
    - Each pixel is made with n separate calculations with slightly modified (quasi-random) positions of the camera
    - Similar to the calculation of shadows (2nd method)

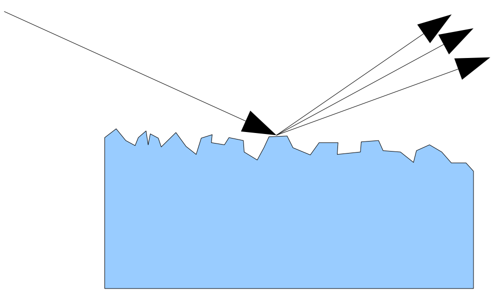
Ray tracing





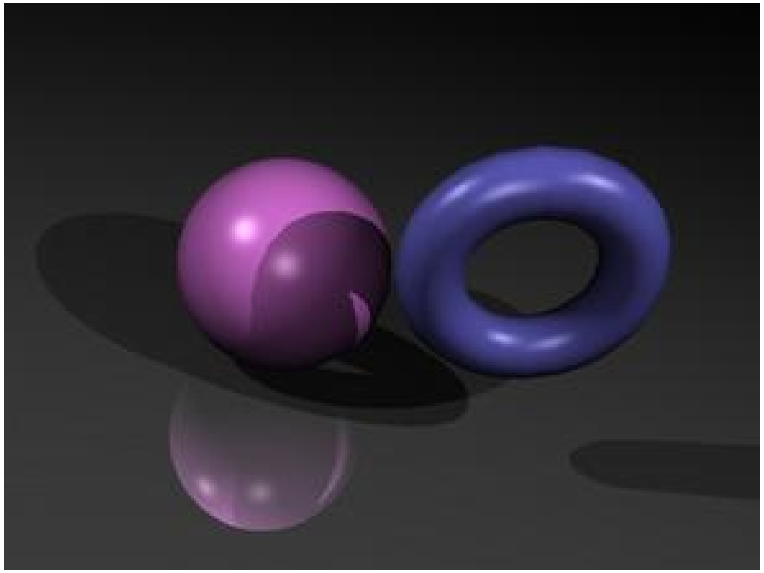
Ray tracing

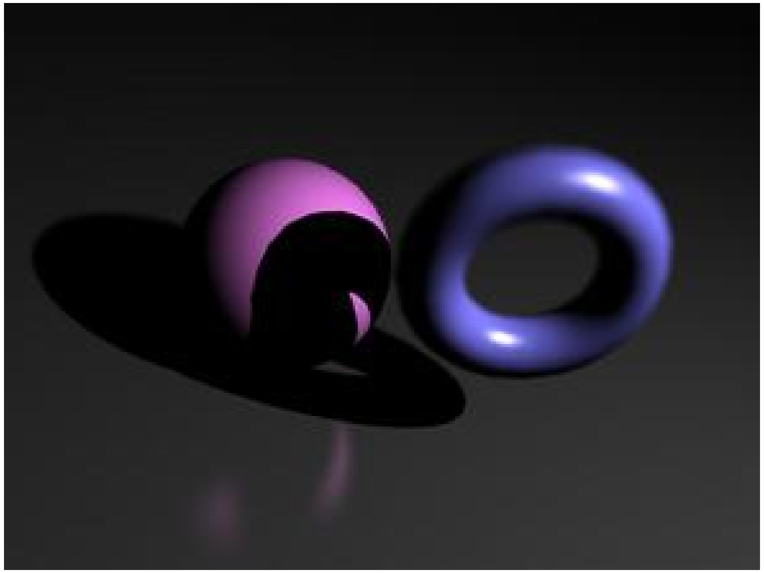
- Imperfect mirror reflections
  - Due to surface irregularities
    - The incident beam is reflected in a direction having a certain variance around the "mirror" configuration
  - Sample randomly



    - It is indeed a specular reflection
    - Here, we want to see the image of other objects (Phong shading does not allows this)

Ray tracing

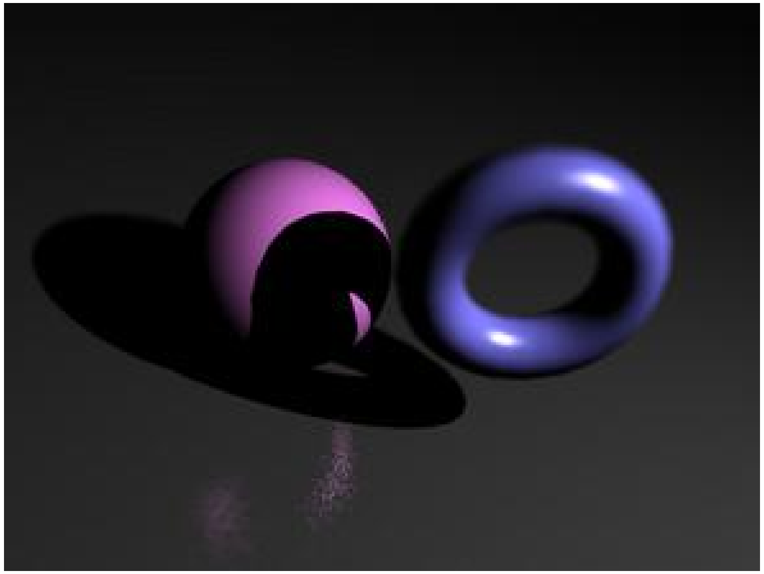




100 samples per point on the flat surface

Ray tracing

- Ray tracing with sampling
  - Provides more realistic images
  - Significant CPU cost ... but simple implementation
  - Also called "distribution ray tracing"
    - Each pixel is not the result of a single ray but instead the result of a distribution of rays
    - Non-deterministic approach are used (quasi-random distribution) to avoid the Moiré pattern
    - Statistical treatment is possible



2 samples

Rendering equation

Rendering equation

- Until now, the procedure only takes into account the illumination of diffuse surfaces by direct light sources
- It solves very approximately what is called the "rendering equation"
  - Light energy balance written for any point on the surface

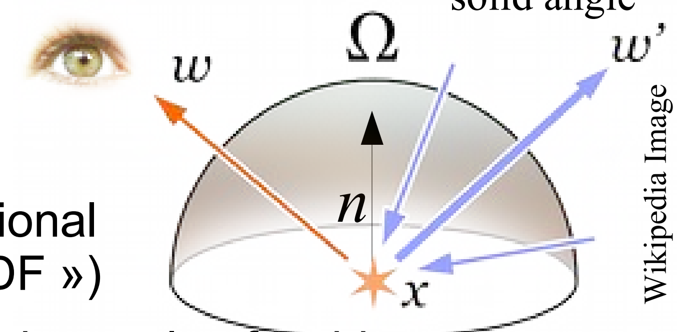
$$L_o(x,\omega,\lambda)=L_e(x,\omega,\lambda)+\int_{\Omega} L_i(x,\omega',\lambda)(\omega'\cdot n)\,f_r(x,\omega',\omega,\lambda)\,d\omega'$$

Infinitesimal solid angle

$\mathrm{lm}\cdot\mathrm{m}^{-2}\cdot\mathrm{sr}^{-1}$

sr^{-1}

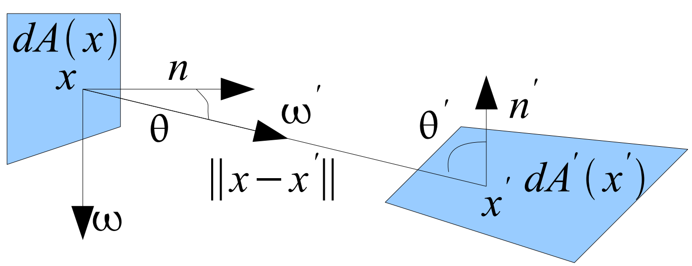
- $L_o(x,\omega,\lambda)$ = light leaving the point
- $L_e(x,\omega,\lambda)$ = emitted light
- $L_i(x,\omega,\lambda)$ = incident light
- $f_r(x,\omega',\omega,\lambda)$ = function of bidirectional spectral reflectance (« BRDF »)
- $\omega'\cdot n$ Attenuation factor related to the angle of incidence



Rendering equation

- Expression of the rendering equation as a function of infinitesimal surface elements

$$L(x,\omega,\lambda)=L_e(x,\omega,\lambda)+\int_{\Omega} f_r(x,\omega',\omega,\lambda)\,L(x,\omega',\lambda)(\omega'\cdot n)\,d\omega'$$



$$(\omega'\cdot n)\,d\omega'=\cos\theta\,d\omega'$$

$$d\omega'=\frac{\cos\theta'}{\|x-x'\|^2}\,dA'(x')$$

$$L(x,\omega,\lambda)=L_e(x,\omega,\lambda)+\int_{S} f_r(x,\omega',\omega,\lambda)\,L(x',\omega',\lambda)\,G(x,x')\,dA'(x')$$

$$G(x,x')=\frac{\cos\theta\cos\theta'}{\|x-x'\|^2}\,V(x,x')$$

$$V(x,x')=\begin{cases}1 & \text{if visible}\\ 0 & \text{if not visible}\end{cases}$$

Rendering equation

- How to solve ?
  - Finite element method (radiosity)
    - Mesh surfaces
    - Approximation of the radiosity on each element
    - Solving a linear system with N unknowns
    - Restricted to a simple form of diffuse reflection models
  - Stochastic methods
    - Based on ray tracing
    - Takes the conservation of energy into account
    - Based on the calculations of probabilistic paths from the light source to the observer
    - More varied physical models

Rendering equation

- Stochastic methods
  - Metropolis Light Transport : an efficient method
    - Solves the rendering equation in an *unbiased* way
    - Simple implementation, good convergence properties
    - Cf article of 1997:

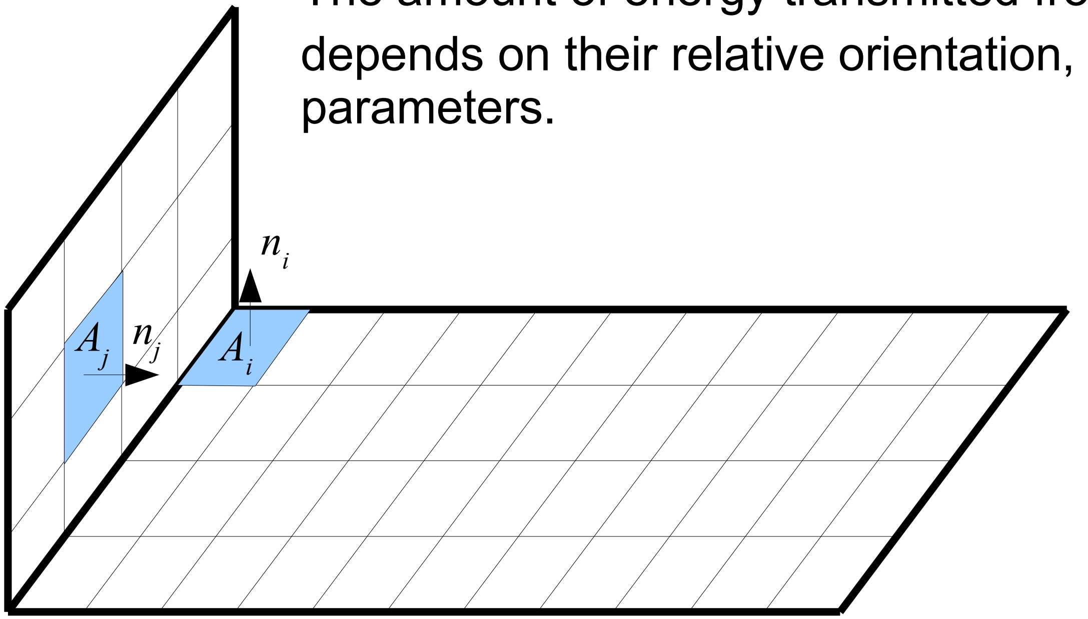
      E. Veach and L.J. Guibas, Metropolis Light Transport. In SIGGRAPH' 97: Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques, 1997, pp. 65-76.

    - An addition to Blender allows you to use this method cf website

      www.luxrender.net
    - Allows the calculation of global illumination for 'difficult' scenes
      - ex. dark room next to a bright room separated by a door slightly ajar

Rendering equation

- Finite element method (radiosity method)
  - Assume all surfaces are discretized in n patchs P_i . Each patch P_i has an aera A_i, an orientation n_i, a radiosity B_i, ...
    - The amount of energy transmitted from the patch P_i to the patch P_j depends on their relative orientation, area, and other geometrical parameters.



Radiosity

- Radiosity equation

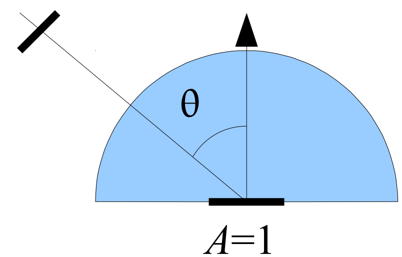
$$L(x,\omega,\lambda)=L_e(x,\omega,\lambda)+\int_S f_r(x,\omega',\omega,\lambda)\,L(x',\omega',\lambda)\,G(x,x')\,dA'(x')$$

  - We assume a Lambertian reflection
    There is therefore independence from the orientation ω

$$L(x,\omega,\lambda)=L(x,\lambda) \qquad L_e(x,\omega,\lambda)=L_e(x,\lambda)$$

    - We can therefore integrate the radiance (luminance) in any direction (hemisphere) : *radiosity* is obtained

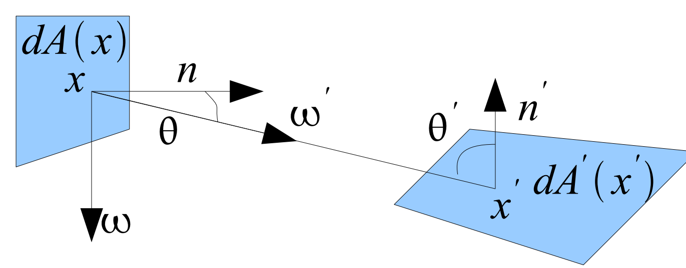
$$B(x,\lambda)=\int_\Omega L(x,\lambda)\cos\theta\,d\theta=\pi\,L(x,\lambda)$$



    - Similarly with the reflectivity:

$$f_r(x,\omega',\omega,\lambda)=f_r(x,\lambda)$$

$$R(x,\lambda)=\int_\Omega f_r(x,\lambda)\cos\theta\,d\theta=\pi\,f_r(x,\lambda)$$



Radiosity

$$L(x,\omega,\lambda)=L_e(x,\omega,\lambda)+\int_S f_r(x,\omega',\omega,\lambda)\,L(x',\omega',\lambda)\,G(x,x')\,dA'(x')$$

- We finally obtain (at each point x):

$$B(x)=B_e(x)+R(x)\int_S B(x')\,F(x,x')\,dA'(x')$$

$$F(x,x')=\frac{G(x,x')}{\pi}$$

- F is a shape factor : percentage of light leaving dA' coming on dA .

Radiosity

- Radiosity equation

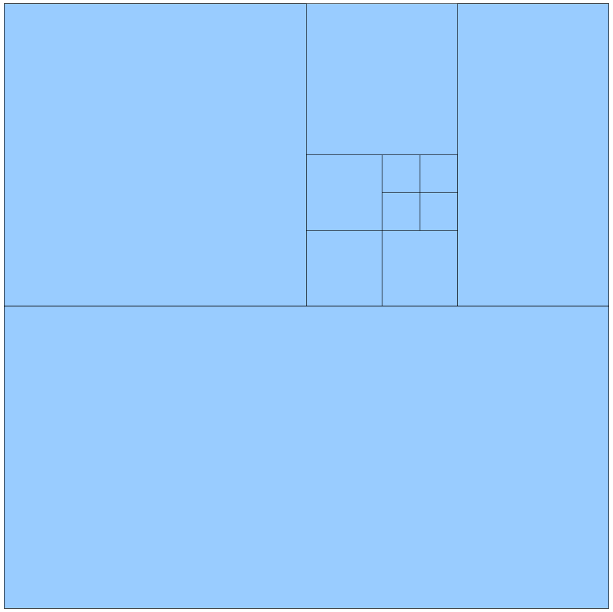
$$B(x) = B_e(x) + R(x) \int_S B(x') F(x, x') \, dA'(x')$$

  - Integral equation of the second kind

$$f(x) = g(x) + \int k(x, x') f(x') \, dx'$$

Radiosity

- Solving of radiosity equation: space discretization
- No need for a conforming mesh
- Variable size



Radiosity

- Solving of radiosity equation: space discretization

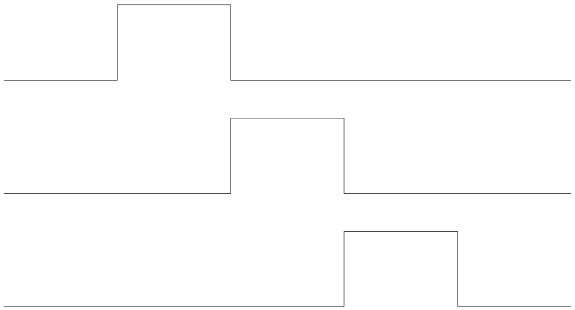
$$B(x) = B_e(x) + R(x) \int_S B(x') F(x, x') dA'(x')$$

- It will be assumed constant variable on patch P_i

$$B(x) = \sum_i B_i N_i(x)$$

$$B_e(x) = \sum_i E_i N_i(x)$$

$$R(x) = \sum_i R_i N_i(x)$$



  - The shape functions N_i are equal to 1 on the patch P_i and 0 everywhere else...

- Conversion towards a discrete linear system...

$$B(x)=B_e(x)+R(x)\int_S \sum_j B_j N_j(x')F(x,x')dA'$$

$$\sum_i B_i N_i(x)=\sum_i E_i N_i(x)+\sum_i R_i N_i(x) B_j\left[\sum_j \int_S N_j(x')F(x,x')dA'\right]$$

- This equation is satisfied for each point x . So we can integrate on each patch...

$$\int_S \left(\sum_i B_i N_i(x)\right) dA=$$

$$\int_S \left(\sum_i E_i N_i(x)+\sum_i R_i N_i(x) B_j\left[\sum_j \int_S N_j(x')F(x,x')dA'\right]\right) dA$$

$$\longrightarrow \quad B_i A_i=E_i A_i+R_i \sum_j B_j \iint_{S^2} F(x,x')N_i(x)N_j(x')dA\,dA'$$

Radiosity

- We have

$$F(x,x')=\frac{\cos\theta\cos\theta'}{\pi\|x-x'\|^2}V(x,x')$$

- We set :

$$T_{ij}=T_{ji}=\int_{A_i}\int_{A_j}\frac{\cos\theta\cos\theta'}{\pi\|x-x'\|^2}V(x,x')\,dA\,dA'$$

$$T_{ij}=A_iF_{ij}$$

- Reciprocity :

$$T_{ij}=A_iF_{ij}\qquad T_{ji}=A_jF_{ji}\longrightarrow A_jF_{ji}=A_iF_{ij}$$

- Unity sum : $\sum_j F_{ij}=\sum_i F_{ji}=1$

Radiosity

- Finally,

$$B_i A_i = E_i A_i + R_i \sum_j B_j A_j F_{ji}$$

- Use reciprocity … $A_j F_{ji} = A_i F_{ij}$

$$B_i = E_i + R_i \sum_j B_j F_{ij}$$

- Linear system of radiosity

$$\begin{pmatrix} 1 - R_1 F_{11} & -R_1 F_{12} & \cdots & -R_1 F_{1n} \\ -R_2 F_{21} & 1 - R_2 F_{22} & \cdots & -R_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -R_n F_{n1} & -R_n F_{n2} & \cdots & 1 - R_n F_{nn} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix} \qquad (I - K) B = E$$

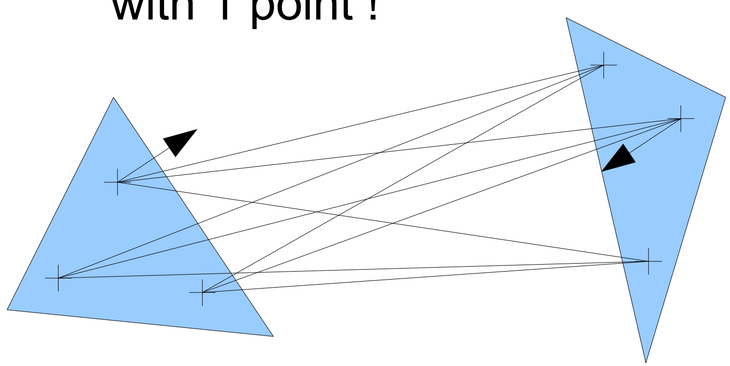
Radiosity

- It remains to calculate F_{ij} and solve the system.
  - For a scene, it is done once, whatever the viewpoint
  - Calculation of F_{ij} (This is the most expensive operation!)
    - Purely geometrical
    - There are n^2
    - Many vanish :
      - Mutually hidden patches
      - Incompatible orientation
      - F_{ii} terms if the patches are plane

Computer Graphics

Radiosity

- Calculation of Fij
  - $T_{ij} = A_i F_{ij}$ $\qquad T_{ij} = \int_{A_i} \int_{A_j} \frac{\cos\theta \cos\theta'}{\pi \|x - x'\|^2} V(x, x') \, dA \, dA'$
  - Brute force: numerical integration for each pair of patches Pi - Pj
  - One can use a Gaussian quadrature. $\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$
  - If the patches are distant, the term is calculated with 1 point !

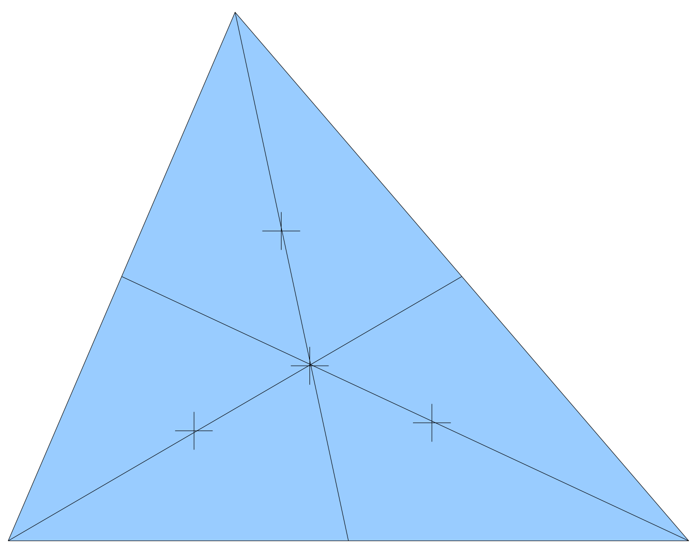


n	$\pm\xi_i$	W_i
1	0.00000 00000 00000	2.00000 00000 00000
2	0.57735 02691 89626	1.00000 00000 00000
3	0.00000 00000 00000	0.88888 88888 88889
	0.77459 66692 41483	0.55555 55555 55556
4	0.33998 10435 84856	0.65214 51548 62546
	0.86113 63115 94053	0.34785 48451 37454
5	0.00000 00000 00000	0.56888 88888 88889
	0.53846 93101 05683	0.47862 86704 99366
	0.90617 98459 38664	0.23692 68850 56189
6	0.23861 91860 83197	0.46791 39345 72691
	0.66120 93864 66265	0.36076 15730 48139
	0.93246 95142 03152	0.17132 44923 79170

Radiosity

- Gauss points for the triangle

$$\int_T f(x,y)\,dxdy \approx A \sum_{i=1}^{n} w_i f(x_i, y_i)$$



n	W_i	ζ_1^i	ζ_2^i, ζ_3^i	M	p
1	1.000000000000000	0.333333333333333	0.333333333333333		1
			0.333333333333333	1	
3	0.333333333333333	0.666666666666667	0.166666666666667		2
			0.166666666666667	3	
4	-0.562500000000000	0.333333333333333	0.333333333333333		3
			0.333333333333333	1	
	0.520833333333333	0.600000000000000	0.200000000000000		
			0.200000000000000	3	
6	0.109951743655322	0.816847572980459	0.091576213509771		4
			0.091576213509771	3	
	0.223381589678011	0.108103018168070	0.445948490915965		
			0.445948490915965	3	
7	0.225000000000000	0.333333333333333	0.333333333333333		5
			0.333333333333333	1	
	0.125939180544827	0.797426985353087	0.101286507323456		
			0.101286507323456	3	
	0.132394152788506	0.059715871789770	0.470142064105115		
			0.470142064105115	3	
12	0.050844906370207	0.873821971016996	0.063089014491502		6
			0.063089014491502	3	
	0.116786275726379	0.501426509658179	0.249286745170910		
			0.249286745170910	3	
	0.082851075618374	0.636502499121399	0.310352451033785		
			0.053145049844816	6	
13	-0.149570044467670	0.333333333333333	0.333333333333333		7
			0.333333333333333	1	
	0.175615257433204	0.479308067841923	0.260345966079038		
			0.260345966079038	3	
	0.053347235608839	0.869739794195568	0.065130102902216		
			0.065130102902216	3	
	0.077113760890257	0.638444188569809	0.312865496004875		
			0.048690315425316	6	

Radiosity

- There are other methods:
  - Projection of patches Pj on a "hemicube" centered on the patch Pi
  - The calculation of Fij is reduced to the calculation of the form factor between patch Pi and the projection of Pj on the hemicube.
  - The hemicube is discretized: the projection is a collection of squares whose contribution to Fij is simple

  cf. « Computer Graphics : Theory into practice, Jeffrey McConnell, Jones & Bartlett ed. » for more details.

- Solving of the linear system
  - Iterative by Gauss-Seidel
  - Direct methods (LU or gaussian pivoting) usually too slow.
  - We do not seek high accuracy...

$$\begin{pmatrix} 1-R_1F_{11} & -R_1F_{12} & \cdots & -R_1F_{1n} \\ -R_2F_{21} & 1-R_2F_{22} & \cdots & -R_2F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -R_nF_{n1} & -R_nF_{n2} & \cdots & 1-R_nF_{nn} \end{pmatrix} \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix} \qquad (I-K)B=E$$

Computer Graphics

Radiosity

- Jacobi iterations

$$(I-K)B=E$$

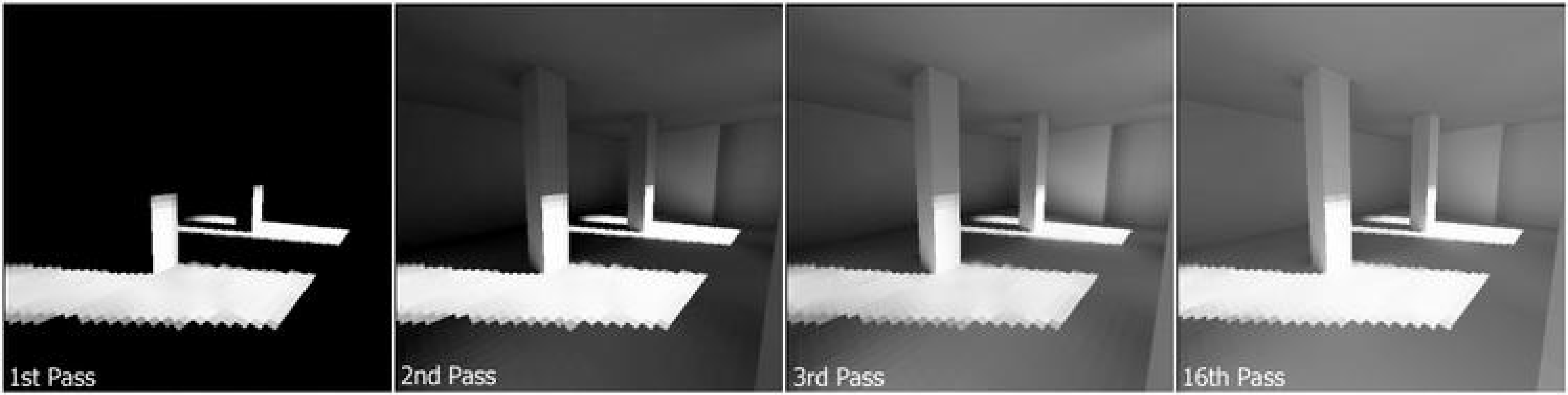
$$B^{(k)}=K\,B^{(k-1)}+E$$

  - For i = 1 to n $\quad B_i^{(k)}=E_i+R_i\sum_{j=1}^{n}F_{ij}B_j^{(k-1)}$
  - We initialize with $B^{(0)}=E$
  - Slow convergence (but easily scalable)
- Gauss-Seidel iterations
  - For i = 1 to n $\quad B_i^{(k)}=E_i+R_i\left(\sum_{j=1}^{i-1}F_{ij}B_j^{(k)}+\sum_{j=i}^{n}F_{ij}B_j^{(k-1)}\right)$
  - Same initialisation.
  - Faster convergence than the Jacobi method (but parallelization is more difficult)

Radiosity

- After the radiosity calculation are done, display the resulting radiosity
- The value of radiosity of each patch is simply used as a texture, changing emissivity of the corresponding surface.
- The construction of the image is done by conventional raytracing
  - Note: Initialization of emitting surface can be done by applying the algorithm of ray-tracing with "classical" (point) sources
  - The radiosity calculation does not depend on the position of the viewer
    - The result of the calculation can be used, also for the real-time rendering (OpenGL or others)

- Radiosity calculation: evolution of convergence

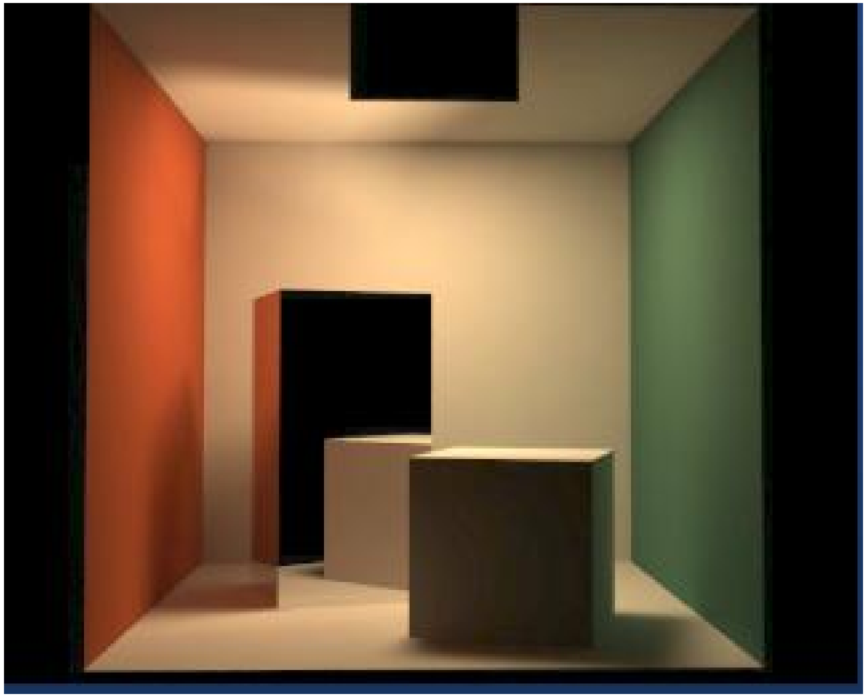


- Issues with radiosity simulations
  - Difficulty with sharp shadows
  - Result depends on the discretization !
  - High memory use / slow calculations
  - Limitations to diffuse surfaces

Radiosity

- Cornell box (1984-1985)

Measured radiosity (CCD image)



Calculated radiosity

Textures

Textures

- The materials we see in everyday life have variable surface properties
- Example : wood
  - Uniform at a very large scale, but strongly variable at small scale
  - Color varies
  - Specularity
  - Direction of anisotropy
  - Etc...

Textures

- Example 2 : checkerboard
  - Repeated geometry
  - Color changes
  - Handmade artifact...

- To design a texture:
  - Solution 1
    - Model each area as separate objects, and assign a specific material
      - Works for simple textures (checkerboards)
      - Difficult to realize continuous variations such as for wood
  - Solution 2
    - Define a function that assigns to each point on the surface a different characteristic
      - An artifact's surface is bi-dimensional (u,v)
      - One can thus apply an image onto the geometry of the artifcat
      - Often, simple bitmaps are used

Textures

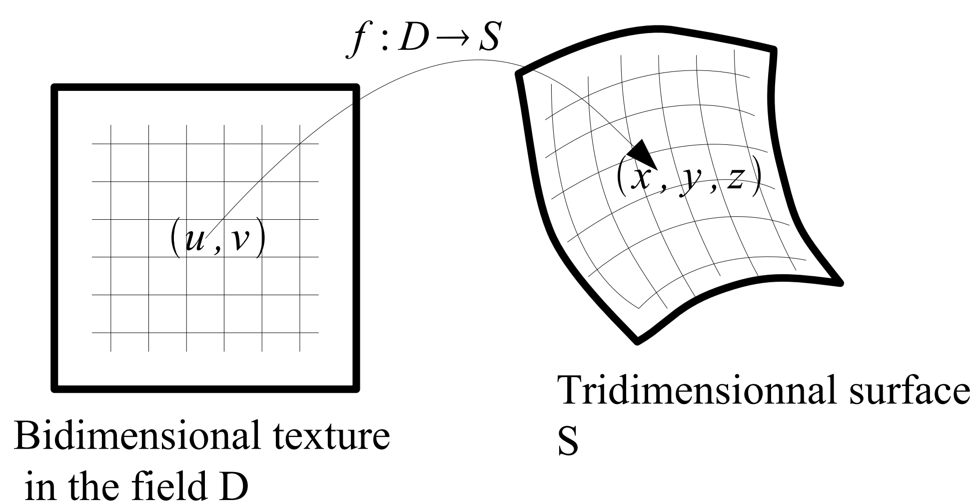
Textures can account for variations of the surface's properties

- This is a function (often a scalar) of space coordinates on a surface
  - Affects the color, reflection model parameters (phong p. ex.)
  - May also affect the geometry: the surface itself is disturbed or the calculation of normal is disturbed
- Allows very fine modeling even with a coarse geometric model
  - Instead of having a very fine geometric model, a kind of image is applied on the surface at each point defining the precise characteristics.

Textures

- Texture = function of (u,v)
  - Problems is the application of textures: where is the image projected onto the surface defined in (u, v)?
  - Only easy for rectangles (direct mapping)
  - Otherwise, transformations are needed (more interesting!)
- We're talking about flat textures, but there are also 3D textures
  - Function of (u,v,w)
  - This texture is assessed only on the surface of the volume (special case: transparent volumes)
  - Interesting for solid materials
  - Often this is defined analytically (or via a procedural definition)
  - Example : rendering of a block of carved wood.

- Texture coordinates
  - How to apply the texture?
    - What is needed is a function f that maps (u,v) to (x,y,z)
    - This looks like a parametric surface definition
    - In fact, if the surface is parametrically defined, we have this function f naturally
    - When calculating the intersection of a ray with the surface, the couple (u,v) is immediately obtained.



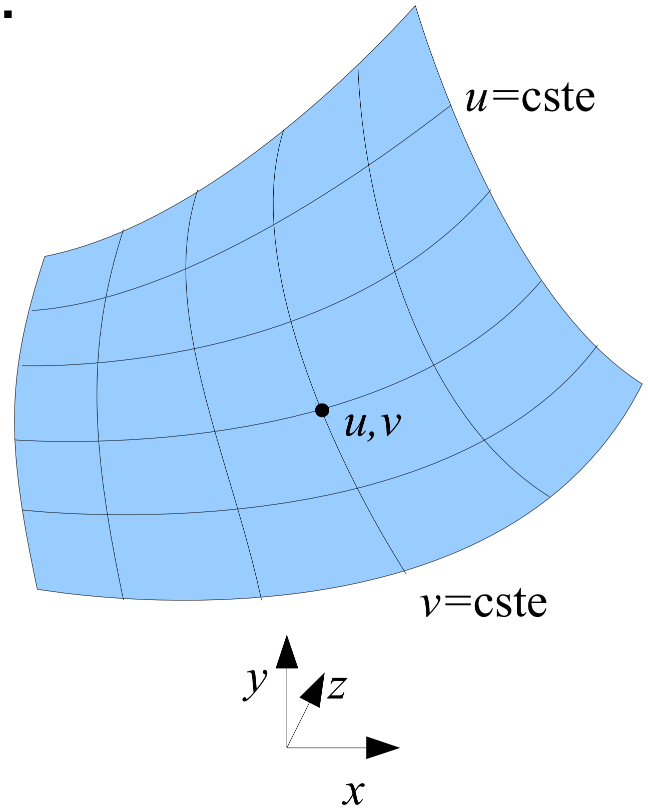
- Texture coordinates
  - Parameterization (u,v) do not generally preseve measures of angles, lengths, or areas.
  - We would like the placement of the texture to be controlled so that its appearance (in (x,y,z) coordinates) is suitable.
  - In the following, let's consider an application f : $(u,v) \longrightarrow (x,y,z)$.
    - This function defines a surface
    - In particular, it allows to apply an image on a surface

A surface is expressed in this form:

$$\vec{P}(u,v)=\begin{cases} x=f(u,v) \\ y=g(u,v) \\ z=h(u,v) \end{cases}$$

u,v are two real parameters

- All points on the surface are obtained by varying u and v.



Computer Graphics

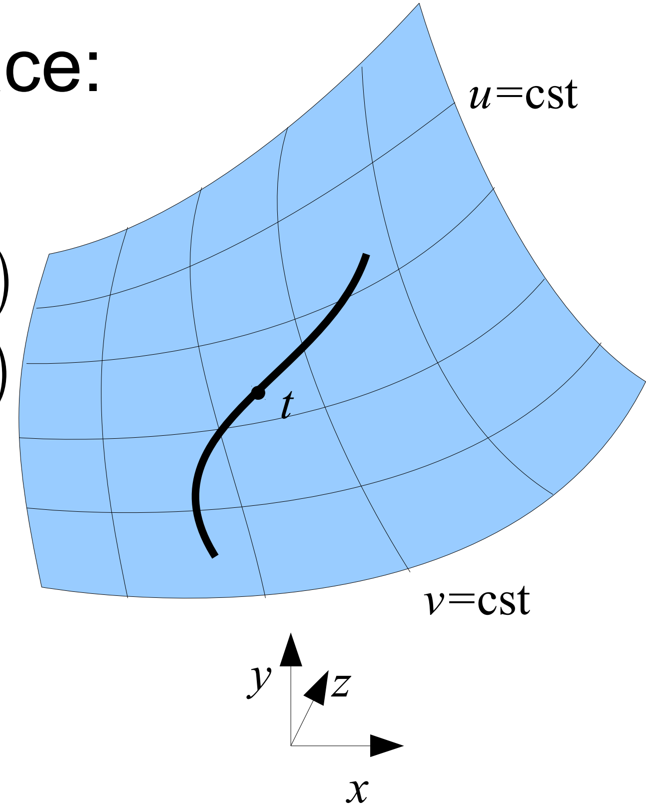
Textures

We can define a curve on the surface:

Parametric space | Ambient space

$$\vec{\Gamma}^{uv}(t):\begin{cases} u=u(t) \\ v=v(t) \end{cases} \longrightarrow \vec{P}(u,v):\begin{cases} x=f(u,v) \\ y=g(u,v) \\ z=h(u,v) \end{cases}$$

$$\vec{\Gamma}(t):\begin{cases} x=f(u(t),v(t)) \\ y=g(u(t),v(t)) \\ z=h(u(t),v(t)) \end{cases}$$



Regularity and continuity of the parameterization

- A parametric surface is of class C_k if the application $P(u,v)$ is of class C_k. (i.e. k-times differentiable)
- A parameterization is *regular* if and only if

$$\frac{\partial \vec{P}}{\partial u}(u_0, v_0) \times \frac{\partial \vec{P}}{\partial v}(u_0, v_0) \neq \vec{0} \quad \forall (u_0, v_0) \in D \subset \mathbb{R}^2$$

- The points that are not satisfying this are *singular points*.
- Equivalent-C_k parametrizations are regular ont the same domain...

Differential geometry for parametric surfaces

- Position P : $\vec{P}(u,v)=\begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}$ $\vec{P}^{\mathrm{u}}=\frac{\partial \vec{P}}{\partial u}$ $\vec{P}^{\mathrm{uv}}=\frac{\partial^2 \vec{P}}{\partial u \partial v}$ $\ldots$
- Unit tangent vectors T^{u} and T^{v} :

  $$\vec{T}^{\mathrm{u}}(u,v)=\frac{\partial P}{\partial u}\cdot\left|\frac{\partial P}{\partial u}\right|^{-1}=\frac{P^{\mathrm{u}}}{\left|P^{\mathrm{u}}\right|} \qquad \vec{T}^{\mathrm{v}}(u,v)=\frac{\partial P}{\partial v}\cdot\left|\frac{\partial P}{\partial v}\right|^{-1}=\frac{P^{\mathrm{v}}}{\left|P^{\mathrm{v}}\right|}$$

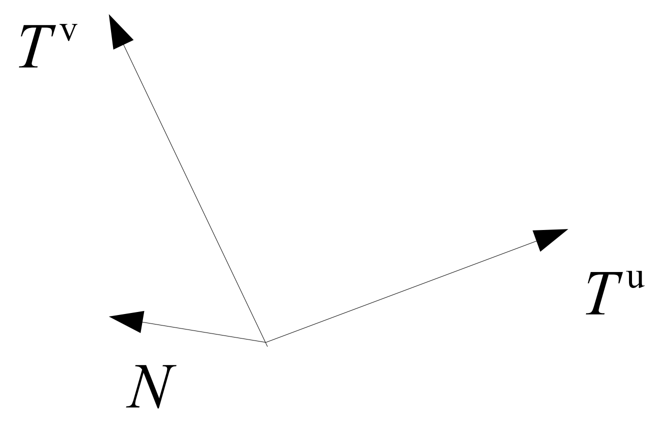
  - These vectors are not always perpendicular
- Tangent plane (parametric form)

$$\vec{Pt}_{(u_0,v_0)}(a,b)=\vec{P}(u_0,v_0)+a\cdot\vec{T}^{\mathrm{u}}(u_0,v_0)+b\cdot\vec{T}^{\mathrm{v}}(u_0,v_0)$$

$$(a,b)\in\mathbb{R}^2$$

- Normal vector N :

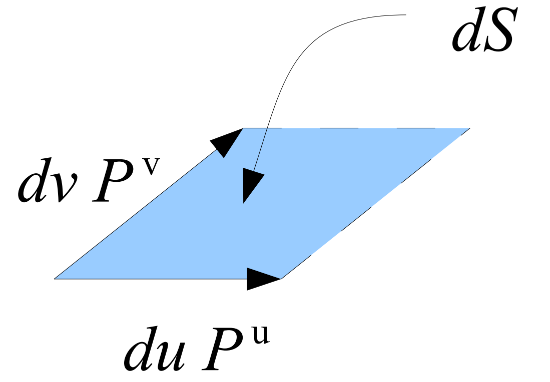
$$N(u,v)=\frac{Norm(u,v)}{|Norm(u,v)|} \text{ with } Norm(u,v)=T^{\mathrm{u}}\times T^{\mathrm{v}} \text{ or } P^{\mathrm{u}}\times P^{\mathrm{v}}$$



- Area:

$$A=\iint_S dS$$

$$dS=|du\cdot P^{u}\times dv\cdot P^{v}|=|P^{u}\times P^{v}|\,dudv$$



1st fundamental form

- Other notation of a the area

$$|\vec{a}\times\vec{b}|^2=(\vec{a}\cdot\vec{a})\cdot(\vec{b}\cdot\vec{b})-(\vec{a}\cdot\vec{b})^2$$

Lagrange's identity

$$dS=\sqrt{(e\,g-f^2)}\,dudv \quad \text{with } e=P^u\cdot P^u \ ,\ f=P^u\cdot P^v \ ,\ g=P^v\cdot P^v$$

$$A=\iint_D \sqrt{(eg-f^2)}\,dudv$$

- Calculate the length of a curve on a surface

$$\vec{P}(u,v):\begin{cases} x=f(u,v) \\ y=g(u,v) \\ z=h(u,v) \end{cases} \qquad P^{\mathrm{u}}=\frac{\partial P(u,v)}{\partial u} \qquad P^{\mathrm{v}}=\frac{\partial P(u,v)}{\partial v}$$

$$\vec{\Gamma}^{uv}(t):\begin{cases} u=u(t) \\ v=v(t) \end{cases} \qquad \vec{\Gamma}(t):\begin{cases} x=f(u(t),v(t)) \\ y=g(u(t),v(t)) \\ z=h(u(t),v(t)) \end{cases} \qquad \Gamma'=\frac{dP(u(t),v(t))}{dt}$$

$$u'=\frac{du(t)}{dt} \quad \ldots$$

$$d\,\Gamma^{uv}(t)=\begin{pmatrix} du \\ dv \end{pmatrix}=\begin{pmatrix} u'\,dt \\ v'\,dt \end{pmatrix}$$

$$L=\int_a^b \left|\vec{\Gamma}'(t)\right| dt=\int_a^b \sqrt{\left|\vec{\Gamma}'(t)\right|^2}\, dt$$

- Derivation in series:

$$\Gamma'(t)=u'(t)\,P^{\mathrm{u}}(u(t),v(t))+v'(t)\,P^{\mathrm{v}}(u(t),v(t))$$

$$|\Gamma'(t)|^2=e\,u'(t)^2+2\,f\,u'(t)\,v'(t)+g\,v'(t)^2$$

$$\text{with } e=P_u\cdot P_u\ ,\ f=P_u\cdot P_v\ ,\ g=P_v\cdot P_v$$

- If we set

$$ds = \sqrt{e\,u'(t)^2 + 2 f\,u'(t)\,v'(t) + g\,v'(t)^2}\,dt$$

which amounts to $ds = \sqrt{e\,du^2 + 2 f\,du dv + g\,dv^2}$

(we have $L = \int_{s(a)}^{s(b)} ds = s(b) - s(a)$)

, it is in fact a quadratic form:

$$e\,u'(t)^2 + 2 f\,u'(t)\,v'(t) + g\,v'(t)^2 = \begin{pmatrix} u'(t) & v'(t) \end{pmatrix} \begin{pmatrix} e & f \\ f & g \end{pmatrix} \begin{pmatrix} u'(t) \\ v'(t) \end{pmatrix}$$

$$L = \int_a^b \sqrt{\begin{pmatrix} u'(t) & v'(t) \end{pmatrix} \begin{pmatrix} e & f \\ f & g \end{pmatrix} \begin{pmatrix} u'(t) \\ v'(t) \end{pmatrix}}\,dt$$

- Angle between two curves ...

$$\Gamma'_1(t)\cdot\Gamma'_2(t)=\left|\Gamma'_1(t)\right|\left|\Gamma'_2(t)\right|\cos\alpha=\begin{pmatrix}u'_1(t) & v'_1(t)\end{pmatrix}\begin{pmatrix}e & f\\ f & g\end{pmatrix}\begin{pmatrix}u'_2(t)\\ v'_2(t)\end{pmatrix}$$

$$\cos\alpha=\frac{\begin{pmatrix}u'_1(t) & v'_1(t)\end{pmatrix}\begin{pmatrix}e & f\\ f & g\end{pmatrix}\begin{pmatrix}u'_2(t)\\ v'_2(t)\end{pmatrix}}{\sqrt{\begin{pmatrix}u'_1(t) & v'_1(t)\end{pmatrix}\begin{pmatrix}e & f\\ f & g\end{pmatrix}\begin{pmatrix}u'_1(t)\\ v'_1(t)\end{pmatrix}\begin{pmatrix}u'_2(t) & v'_2(t)\end{pmatrix}\begin{pmatrix}e & f\\ f & g\end{pmatrix}\begin{pmatrix}u'_2(t)\\ v'_2(t)\end{pmatrix}}}$$

- The 1rst fundamental form is the application

$$\phi_1(d\,\Gamma_1^{uv}, d\,\Gamma_2^{uv}) = \begin{pmatrix} du_1 & dv_1 \end{pmatrix} \begin{pmatrix} e & f \\ f & g \end{pmatrix} \begin{pmatrix} du_2 \\ dv_2 \end{pmatrix} = \begin{pmatrix} du_1 & dv_1 \end{pmatrix} \mathrm{M}_1 \begin{pmatrix} du_2 \\ dv_2 \end{pmatrix}$$

with $e = P_u \cdot P_u$, $f = P_u \cdot P_v$, $g = P_v \cdot P_v$

  - It is a symmetric bilinear form that can "measure" actual distances from variations in the parametric space ...
  - The M_1 matrix is a representation of the metric tensor.
  - M_1 is also related to the Jacobian matrix of the transformation $(u,v) \rightarrow (x,y,z)$ (It is $J^T J$). $J = \begin{pmatrix} \partial x/\partial u & \partial x/\partial v \\ \partial y/\partial u & \partial y/\partial v \\ \partial z/\partial u & \partial z/\partial v \end{pmatrix}$

$$L = \int_a^b \sqrt{\phi_1(d\,\Gamma^{uv}, d\,\Gamma^{uv})}\, dt \qquad \cos\alpha = \frac{\phi_1(d\,\Gamma_1^{uv}, d\,\Gamma_2^{uv})}{\sqrt{\phi_1(d\,\Gamma_1^{uv}, d\,\Gamma_1^{uv})\,\phi_1(d\,\Gamma_2^{uv}, d\,\Gamma_2^{uv})}}$$

$$A = \iint_D \sqrt{\det \mathrm{M}_1}\, du dv \quad \left(= \iint_D \det \mathrm{J}\, du dv \text{ under some conditions} \right)$$

Textures

- Back to our textures
  - An image is to be applied on a curved surface
  - If the coordinates u,v are used as texture coordinates, the areas are not generally preserved.
  - Using the first fundamental form, we can compensate this by defining an alternative parametrization
  - Example: a sphere

$$P(u,v)=\begin{cases} x(u,v)=r\cdot\cos u\cos v \\ y(u,v)=r\cdot\sin u\cdot\cos v \\ z(u,v)=r\cdot\sin v \end{cases}$$

$$\begin{pmatrix} e & f \\ f & g \end{pmatrix} \qquad \begin{aligned} e&=r^2\cdot(\sin^2 u\cos^2 v+\cos^2 u\cos^2 v) \\ f&=r^2\cdot(\sin u\cos v\sin v\cos u-\sin u\cos v\sin v\cos u)=0 \\ g&=r^2\cdot(\cos^2 u\sin^2 v+\sin^2 u\sin^2 v+\cos^2 v) \end{aligned}$$

- Back to our textures
  - First fundamental form and the metric tensor

$$\mathrm{M}_1 = \begin{pmatrix} e & f \\ f & g \end{pmatrix}$$

$$e = r^2 \cdot (\sin^2 u \cos^2 v + \cos^2 u \cos^2 v) = r^2 \cos^2 v$$
$$f = r^2 \cdot (\sin u \cos v \sin v \cos u - \sin u \cos v \sin v \cos u) = 0$$
$$g = r^2 \cdot (\cos^2 u \sin^2 v + \sin^2 u \sin^2 v + \cos^2 v) = r^2$$

  - Area ratio as a function of the position

$$dA = \sqrt{det\,\mathrm{M}_1}\, dudv$$

$$\sqrt{det\,\mathrm{M}_1} = \sqrt{e \cdot g} = r^2 |\cos v|$$

  - Ratio of the distances in function of the position (iso-u and iso-v)

$$dL = \sqrt{\phi_1 (d\,\Gamma^{uv}, d\,\Gamma^{uv})}$$

Textures

- We set $d\Gamma^{uv} = dt \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ (we are interested in the direction u)

$$dL = \sqrt{\phi_1(d\Gamma^{uv}, d\Gamma^{uv})} = dt\sqrt{\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} e & f \\ f & g \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix}}$$

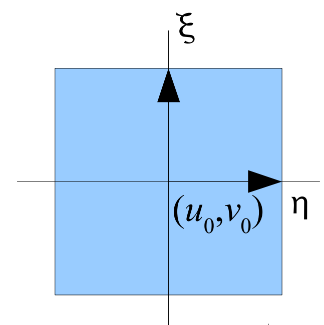
$$\sqrt{\begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} e & f \\ f & g \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix}} = \sqrt{e} = r|\cos v|$$

- With $d\Gamma^{uv} = dt \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ (direction v)

$$\sqrt{\begin{pmatrix} 0 & 1 \end{pmatrix}\begin{pmatrix} e & f \\ f & g \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix}} = \sqrt{g} = r$$

(independent of the position)

- An image to display is centered at (u_0,v_0) on the sphere : how to prevent it from deforming too much?
  - What is the connection that must be made between the texture coordinates (η,ξ) and the parameters (u,v) ?
  - Suppose that (η,ξ) is aligned with (u, v)
  - We want to keep the distances according to η and ξ .
  - We set



$$\Gamma^1 : \begin{pmatrix} u = f_1(\eta^*) \\ v \end{pmatrix} \qquad d\Gamma^1 : \begin{pmatrix} \dfrac{\partial f_1(\eta^*)}{\partial \eta^*} d\eta^* \\ 0 \end{pmatrix}$$

$$\Gamma^2 : \begin{pmatrix} u \\ v = f_2(\xi^*) \end{pmatrix} \qquad d\Gamma^2 : \begin{pmatrix} 0 \\ \dfrac{\partial f_2(\xi^*)}{\partial \xi^*} d\xi^* \end{pmatrix}$$

- Calculate the actual distance along Γ^1: it must be equal to η

$$L(\eta)=\int_0^{\eta}\sqrt{\phi_1(d\Gamma^1,d\Gamma^1)}=\eta$$

$$L(\eta)=\int_0^{\eta}\sqrt{\begin{pmatrix}\dfrac{\partial f_1(\eta^*)}{\partial \eta^*}d\eta^* & 0\end{pmatrix}\cdot\begin{pmatrix}e & f\\ f & g\end{pmatrix}\begin{pmatrix}\dfrac{\partial f_1(\eta^*)}{\partial \eta^*}d\eta^*\\ 0\end{pmatrix}}$$

$$L(\eta)=\int_0^{\eta}\sqrt{e\left(\frac{\partial f_1(\eta^*)}{\partial \eta^*}\right)^2}\,d\eta^*=r\cos v\left[f_1(\eta)-f_1(0)\right]$$

$$r\cos v\left[f_1(\eta)-f_1(0)\right]=\eta$$

$$f_1(\eta)=\frac{\eta}{r\cos v}+f_1(0)\text{ with }f_1(0)=u_0$$

- Same along Γ^2 : actual distance = ξ

$$L(\xi)=\int_0^{\xi}\sqrt{\phi_1(d\Gamma^2,d\Gamma^2)}=\xi$$

$$L(\xi)=\int_0^{\xi}\sqrt{g\left(\frac{\partial f_2(\xi^*)}{\partial \xi^*}\right)^2}\,d\xi^*=r\left[f_2(\xi)-f_2(0)\right]$$

$$r\left[f_2(\xi)-f_2(0)\right]=\xi$$

$$f_2(\xi)=\frac{\xi}{r}+f_2(0)\ \text{with}\ f_2(0)=v_0$$

- Change of coordinates

$$u=\frac{\eta}{r\cos v}+u_0=\frac{\eta}{r\cos\left(\frac{\xi}{r}+v_0\right)}+u_0 \quad\rightarrow\quad \eta=(u-u_0)\,r\cos v$$

$$v=\frac{\xi}{r}+v_0 \quad\rightarrow\quad \xi=(v-v_0)\,r$$

- Is the area preserved ?

$$dA=\sqrt{det\,\mathrm{M}_1^*}\,d\eta\,d\xi$$

  - Yes if $\sqrt{det\,\mathrm{M}_1^*}=1$

$$M_1^*=\begin{pmatrix} e^* & f^* \\ f^* & g^* \end{pmatrix}$$

with $e^*=P_\eta\cdot P_\eta$, $f^*=P_\eta\cdot P_\xi$, $g^*=P_\xi\cdot P_\xi$

- Computation of the terms of the metric tensor

$$P(u,v)=\begin{cases} x(u,v)=r\cdot\cos u\cos v \\ y(u,v)=r\cdot\sin u\cdot\cos v \\ z(u,v)=r\cdot\sin v \end{cases} \rightarrow P(\eta,\xi)=\begin{cases} x=r\cdot\cos u(\eta,\xi)\cos v(\eta,\xi) \\ y=r\cdot\sin u(\eta,\xi)\cdot\cos v(\eta,\xi) \\ z=r\cdot\sin v(\eta,\xi) \end{cases}$$

$$u(\eta,\xi)=\frac{\eta}{r\cos\left(\frac{\xi}{r}+v_0\right)}+u_0 \quad v(\eta,\xi)=\frac{\xi}{r}+v_0$$

$$P_\eta=r\cdot\begin{cases} -\frac{\partial u}{\partial\eta}\sin u\cos v \\ \frac{\partial u}{\partial\eta}\cos u\cos v \\ 0 \end{cases} \qquad P_\xi=r\cdot\begin{cases} -\frac{\partial u}{\partial\xi}\sin u\cos v-\frac{\partial v}{\partial\eta}\cos u\sin v \\ \frac{\partial u}{\partial\xi}\cos u\cos v-\frac{\partial v}{\partial\eta}\sin u\sin v \\ -\frac{\partial v}{\partial\xi}\cos v \end{cases}$$

Computer Graphics

Textures

$$e^* = P_\eta \cdot P_\eta = r^2 \left(\frac{\partial u}{\partial \eta}\right)^2 \cos^2 v$$

$$f^* = P_\eta \cdot P_\xi = r^2 \frac{\partial u}{\partial \eta} \frac{\partial u}{\partial \xi} \cos^2 v$$

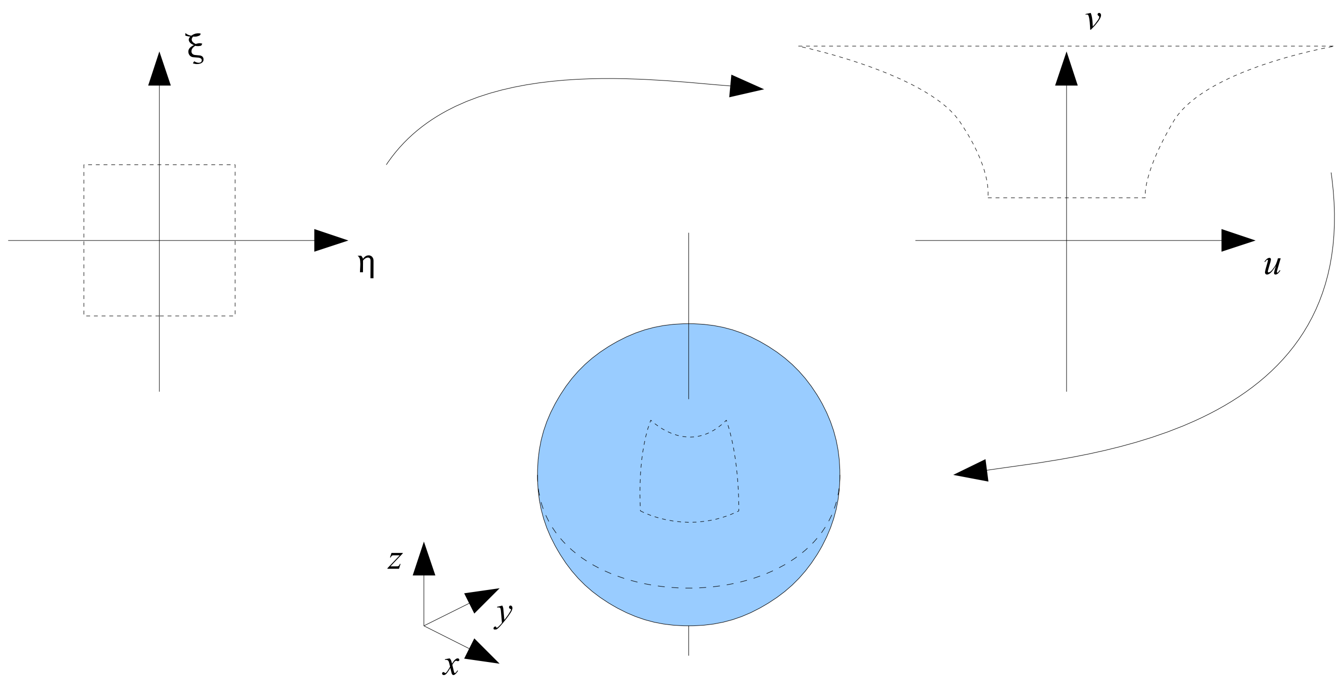
$$g^* = P_\xi \cdot P_\xi = r^2 \left(\left(\frac{\partial u}{\partial \xi}\right)^2 \cos^2 v + 1\right)$$

$$\sqrt{\det \mathrm{M}_1^*} = \sqrt{e^* g^* - f^{*2}} = r \frac{\partial u}{\partial \eta} \cos v$$

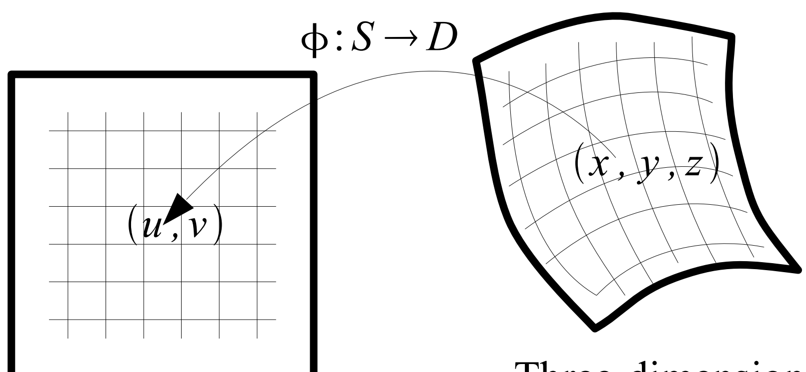
$$= \frac{r}{r \cos(\frac{\xi}{r} + v_0)} \cos(\frac{\xi}{r} + v_0) = 1\,!$$

→ Preservation of the area (this is by chance)

Textures



- Texture coordinates
  - For non parametric surfaces
    - Example : discretized surfaces (triangles)
    - (u, v) is not obtained in the calculation of the intersection of a ray with the surface
    - We must define the inverse operation $f : \phi : S \to D$
      - For a point *P*, the texture is obtained at the position $\phi(P)$
    - We must construct a plausible parameterization of the triangulated surface
    - u,v must be stored at each vertex of the mesh

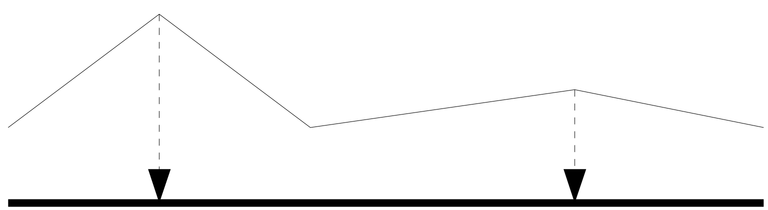


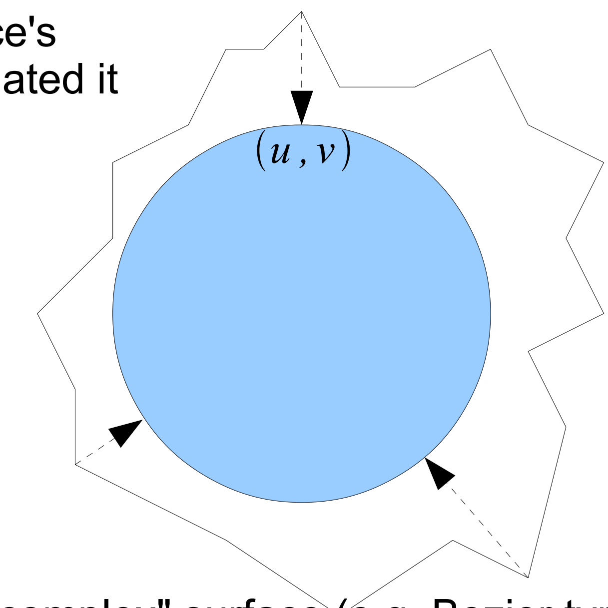
Dimensional texture in the domain D

Three-dimensional surface S

Textures

- Construction of a parameterization of a triangulated surface (mesh)
  - Possibility 1) : Projection of the mesh vertices on a topologically equivalent surface (ex. Plane, sphere ...)
    - Knowing the projection, the surface's parameterization is directly associated it





    - If the triangulation comes from a "complex" surface (e.g. Bezier type), this technique is used. The projection step is not necessary (the vertices are on the surface), it suffices to assign to each vertex the values of the parameter on the original surface.

Textures

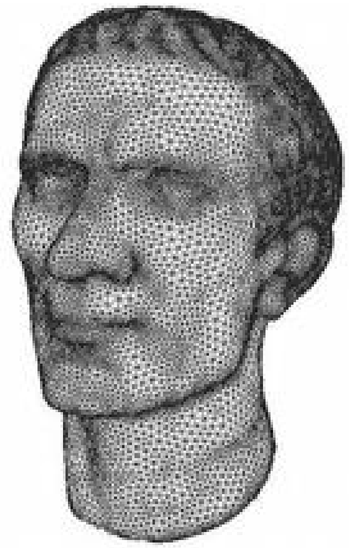
- Construction of a parameterization of a triangulated surface (mesh)
  - Possibility 2) : "Flatten" the mesh and build parameterization

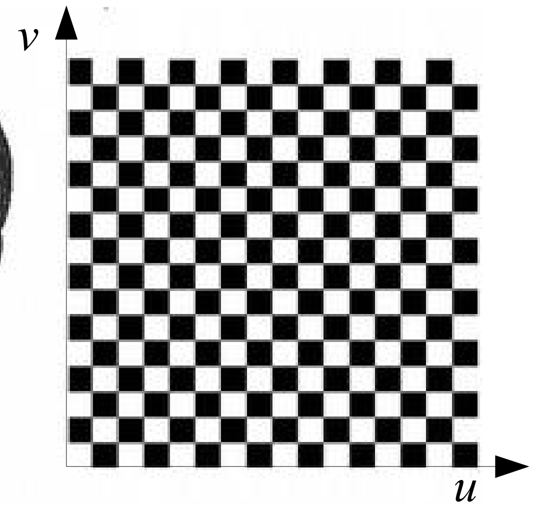
    Goal here: build a parameterization that respect some of the following qualities :
    - Low distortion (angle conservations)
    - Area ratios are roughly preserved
    - Natural coordinates (in geodesic distances)
  - Rather complex algorithms, solutions to the problem are quite recent
    - Amounts to solve partial differential equations

Computer Graphics

Textures

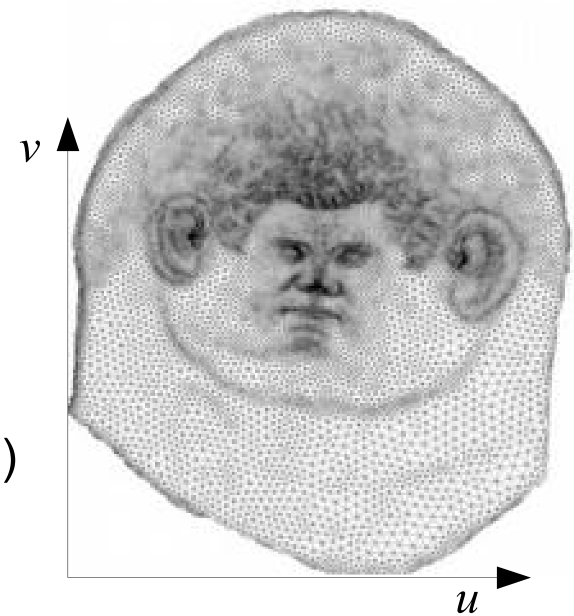
Triangles in
real space





Texture in the
parameteric
space

Triangles
in the parameteric
space (here,
conforming mapping)

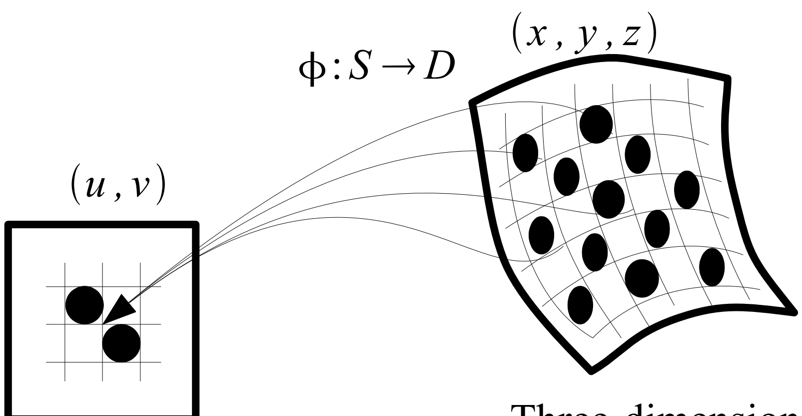


Texture
in the real space

CGAL library (L. Saboret, P. Alliez, B. Lévy)

http://www.cgal.org/...

- Texture coordinates
  - For periodic textures
    - The application $\phi: S \to D$ may be non bijective



Three-dimensional surface S

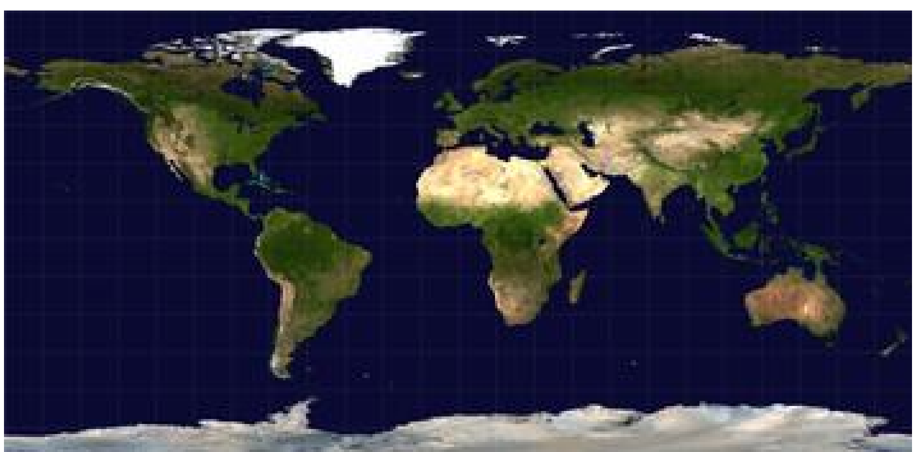
Dimensional texture
in the domain D

    - Example : brick wall, checkerboard, etc...
      - But also pseudo-random textures - one should not see the periodicity but the base texture is set on a small area only.

- Examples of applying textures
  - Parametric surface: a sphere

$$\begin{pmatrix} u \\ v \end{pmatrix} \rightarrow \begin{pmatrix} x = r \sin u \cos v \\ y = r \sin u \sin v \\ z = r \cos u \end{pmatrix}$$

JPG image (projection called « Plate Carrée »)



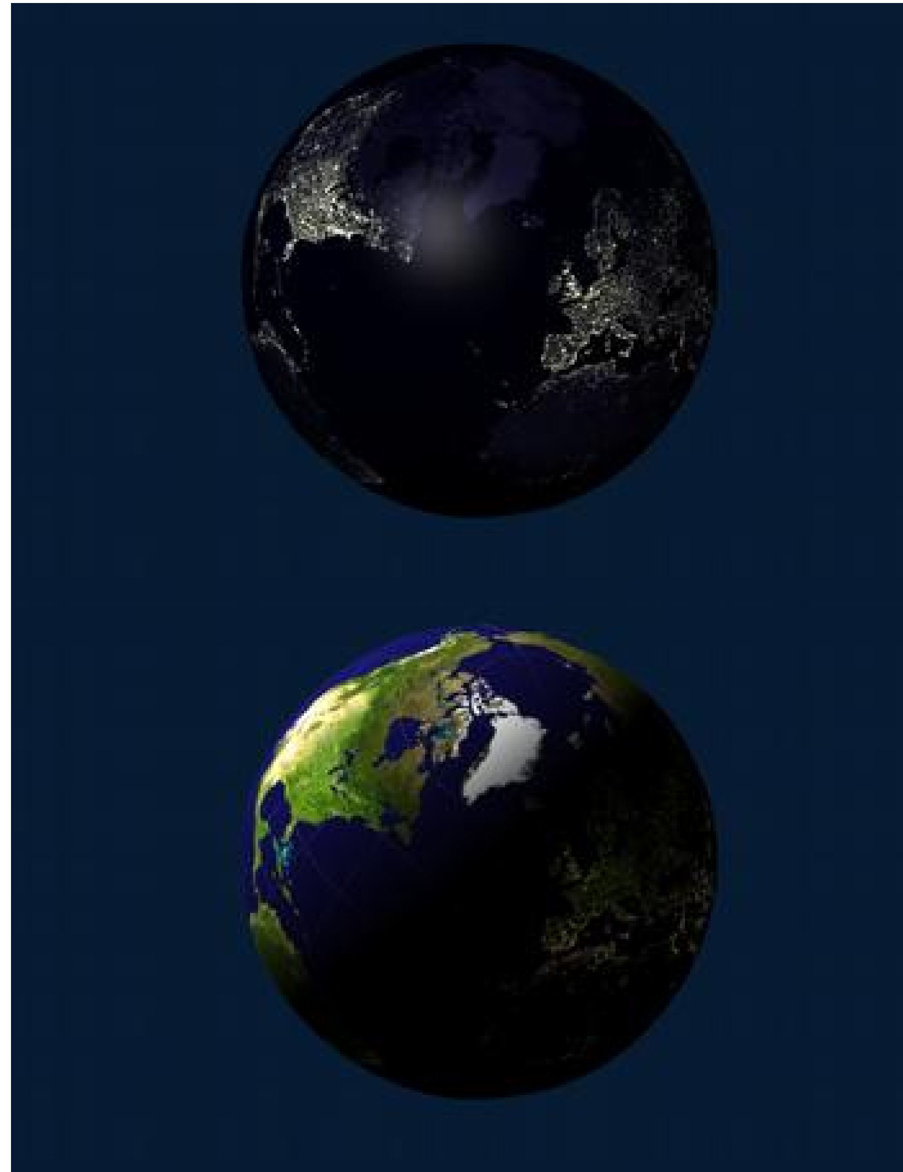
Textures

- The application of texture here concerns the color (color defined by the used image)
- One can change the other parameters of the models of reflection
  - Emissivity (light source)
  - Specular
  - etc...
- It can also affect the geometry
  - Local modification of the surface shape
  - Modification of the normal vectors (cf. Phong interpolation)

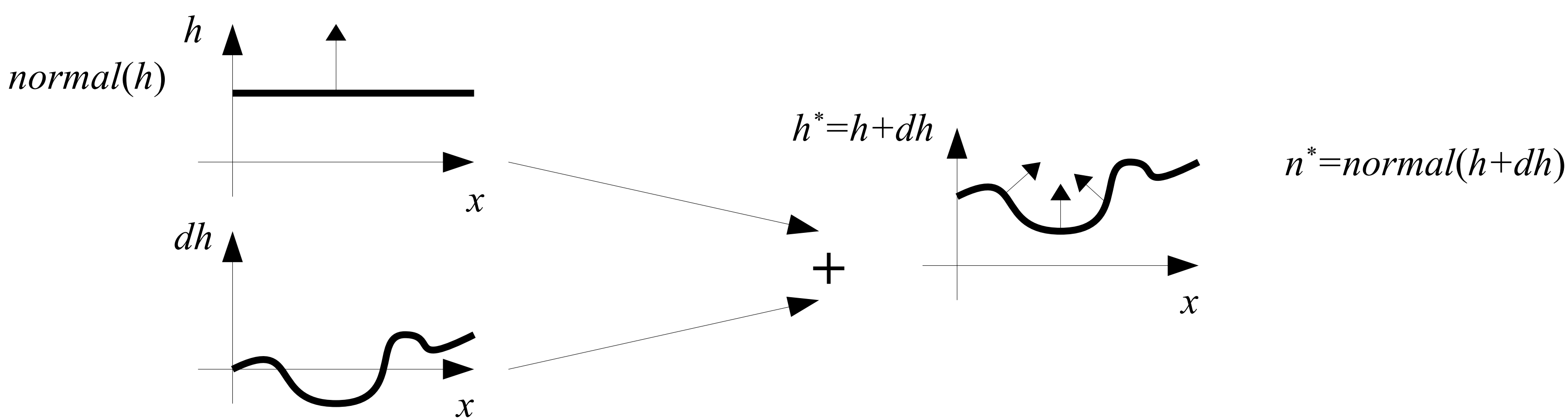
Computer Graphics

Textures

- Using a picture at night, changing the emissivity setting in addition to the color

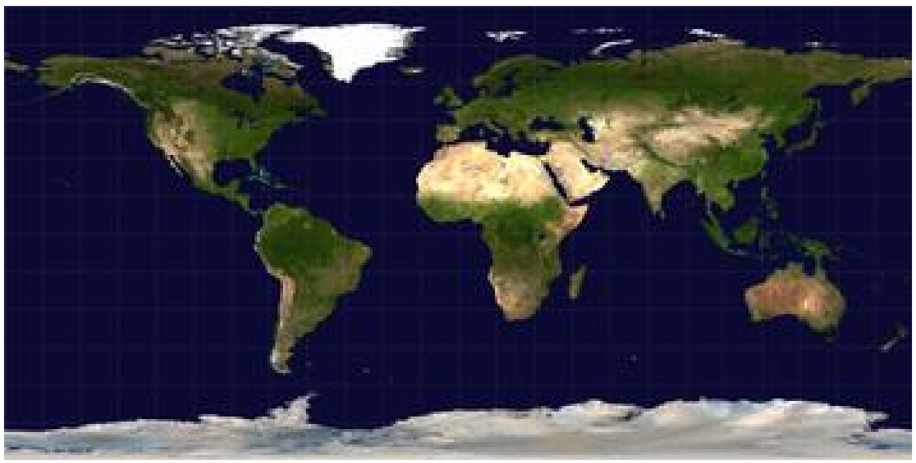


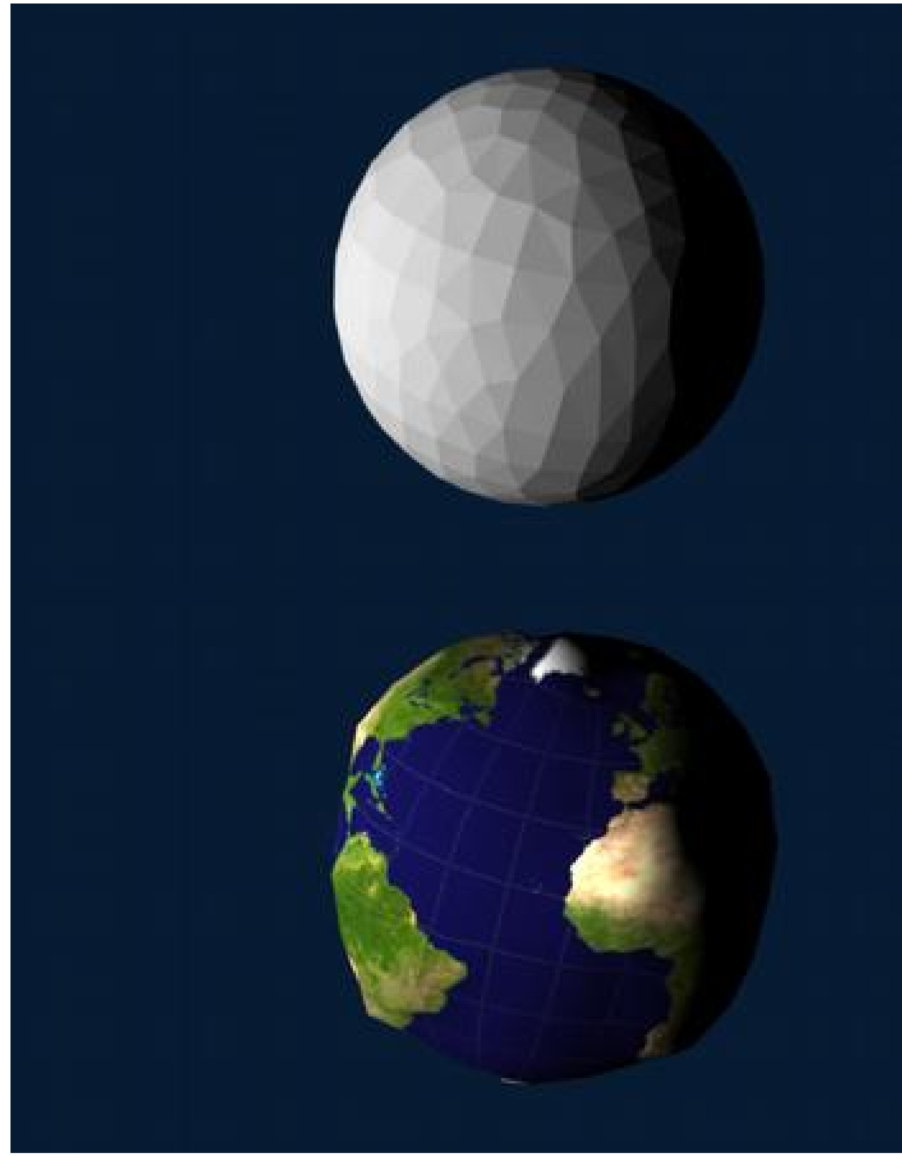
- Geometry modification
- Possibility 1 - Simply moving the geometry
  - Texture (scalar = grayscale) is interpreted as the normal component of the displacement vector.
  - The ray intersects the new geometry
  - The normal is computed according to the new geometry



Textures

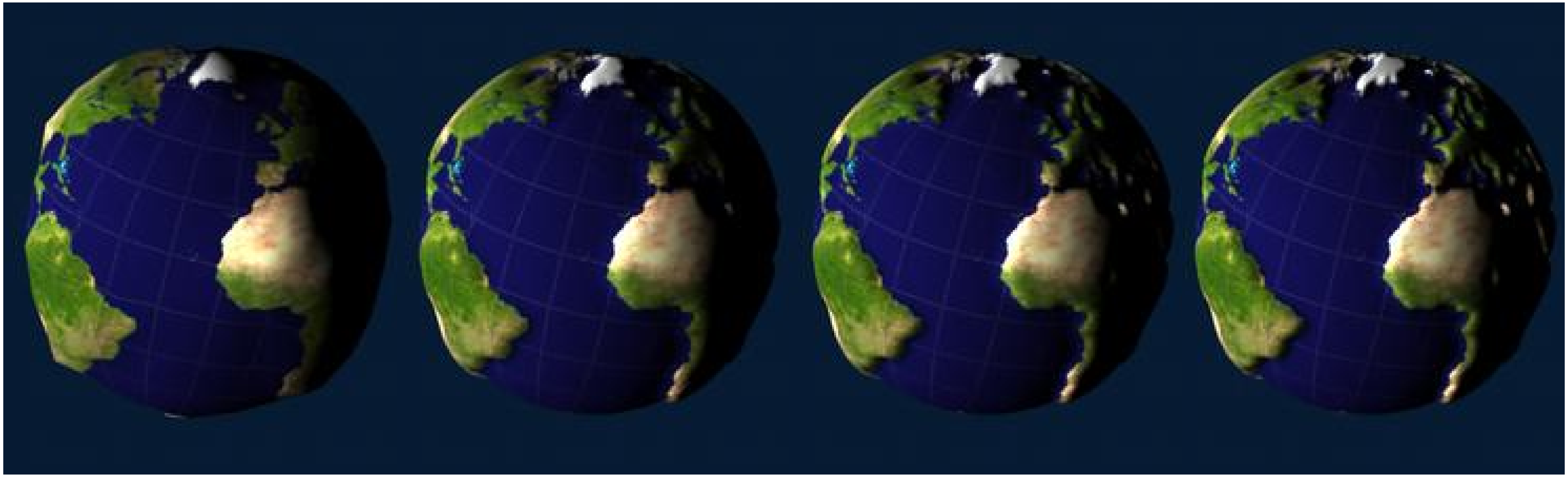
- Result





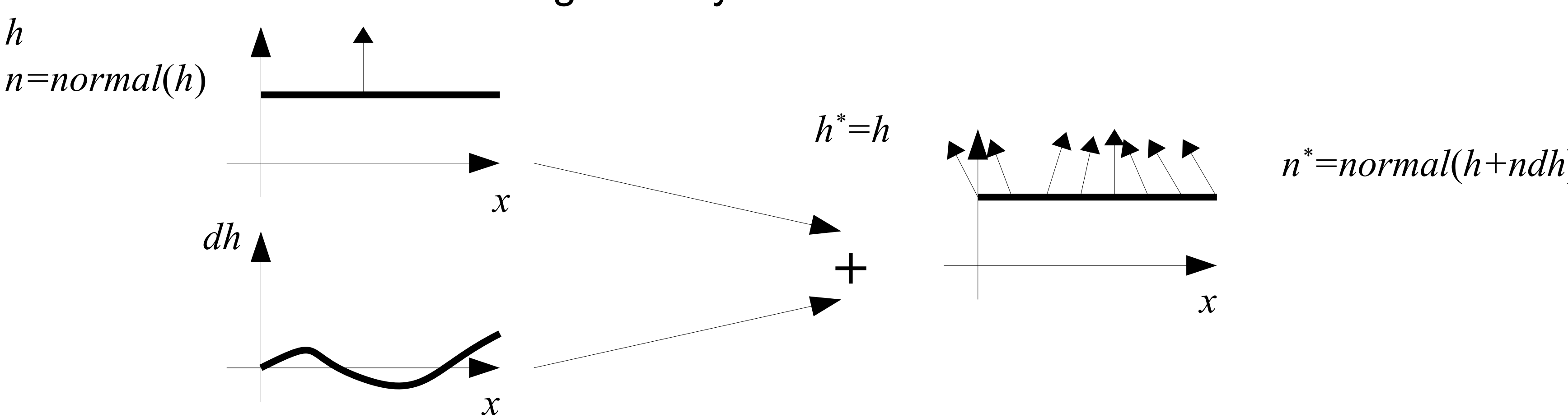
Textures

- Problems
  - The resolution of the texture is very often much greater than that of the geometry (often discretized)
  - Significant cost if we want to respect the texture's resolution - the discretization of the geometry should be very fine



750 19000 76000 1265000

- Geometry modification
- Possibility 2 - We only change the calculation of normals (called "bump mapping")
  - Texture (scalar = grayscale) is interpreted as the normal component of the displacement vector.
  - The ray intersects the original geometry.
  - The normal is calculated at each point as if we were dealing with the new modified geometry



Textures

- Calculate new normals

$$t_u = \frac{\partial t}{\partial u} \qquad t_v = \frac{\partial t}{\partial v}$$

$$n = t_u \times t_v$$

$$n^* = \frac{\partial\left(h + \frac{n}{\|n\|} dh\right)}{\partial u} \times \frac{\partial\left(h + \frac{n}{\|n\|} dh\right)}{\partial v}$$

$$n^* = n + \frac{h_v (t_u \times n) - h_u (t_v \times n)}{\|n\|} \quad (\cdot k)$$

$$h_u = \frac{\partial\, dh}{\partial u} \quad h_v = \frac{\partial\, dh}{\partial v}$$

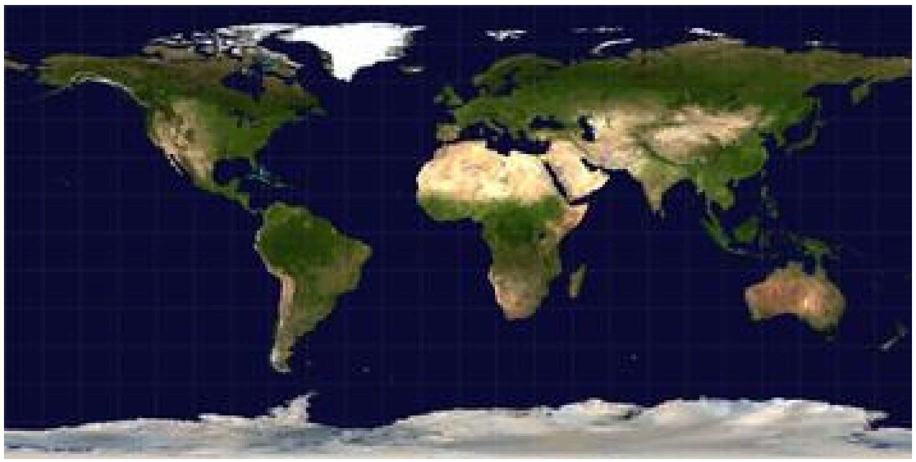
Blinn formula

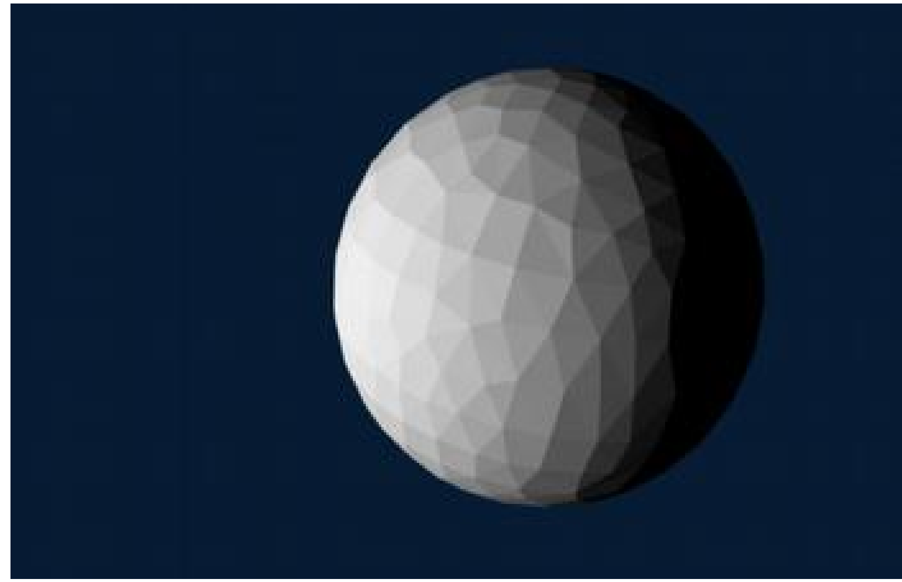
$-1 < k < 1$ can control the depth effect

Computer Graphics

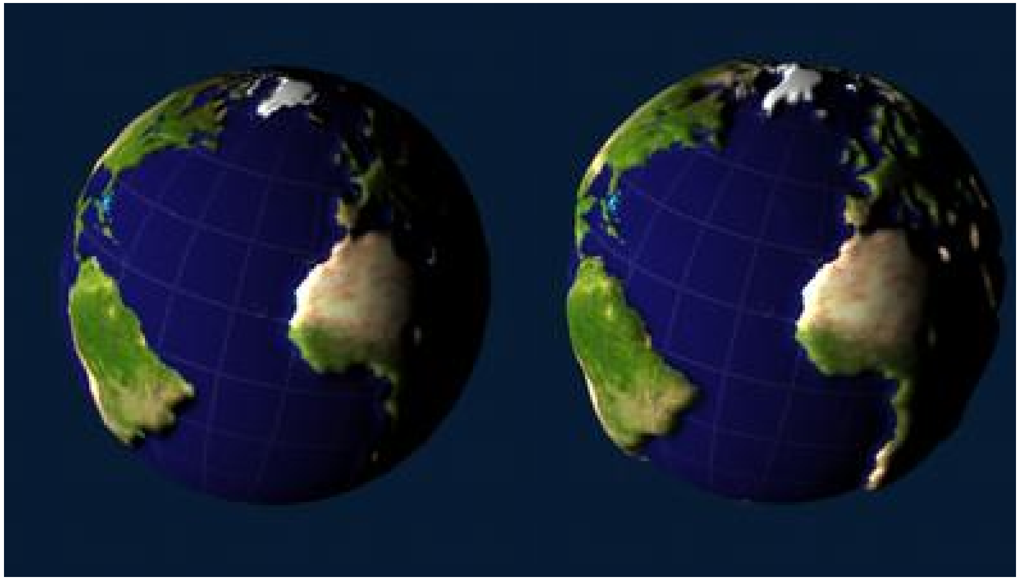
Textures

- Result

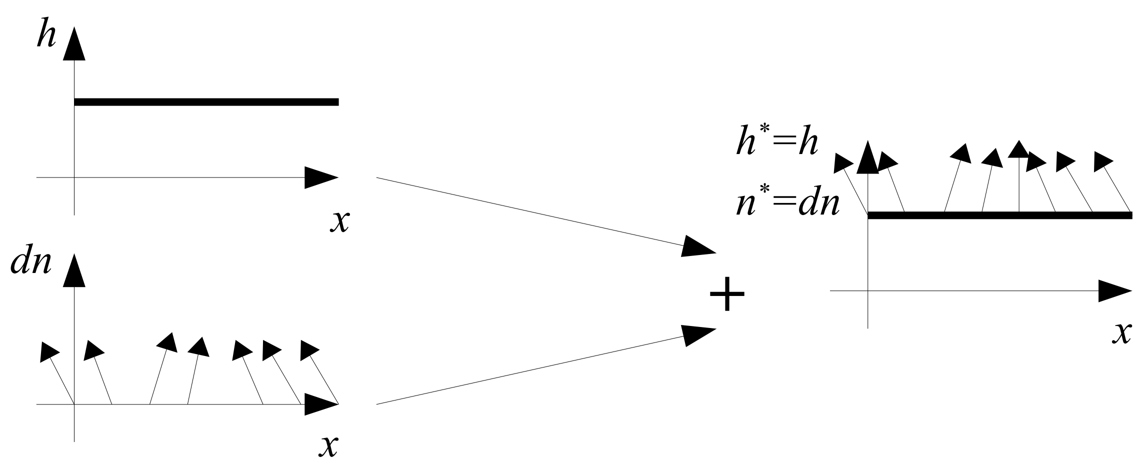




- Comparison

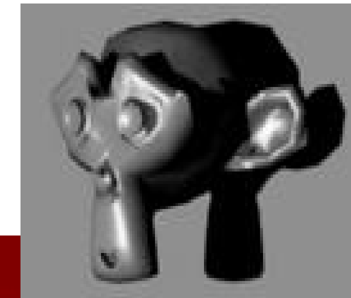


- Geometry modification
- Possibility 3 - Using a normal map
  - The texture is directly interpreted as the normal to the surface (texture with two component = colors)
  - The ray intersects the original geometry.
  - The normal for each point is obtained by the texture

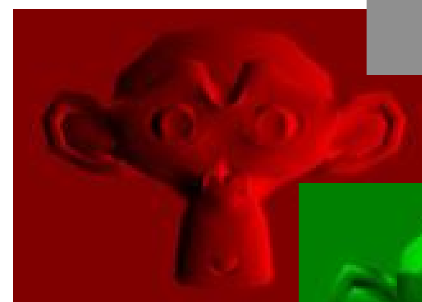


Textures

- How are normal maps created ?
  - 1 –accurate model of the geometry

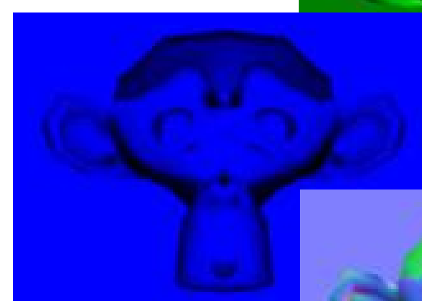


  - 2 – Calculation of rendering such that :



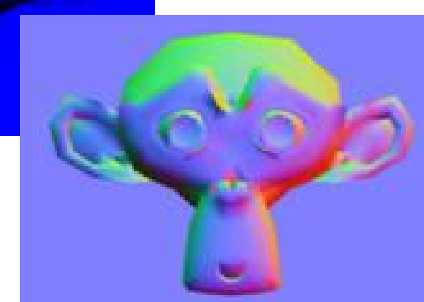
    - The red channel is the x value of the normal (between -1 and 1)

    - The green channel is the y value of the normal (between -1 and 1)



    - The blue channel is the z value of the normal (between 0 and 1 !)

    - Additional constraint :
    We always have $x^2+y^2+z^2=1$

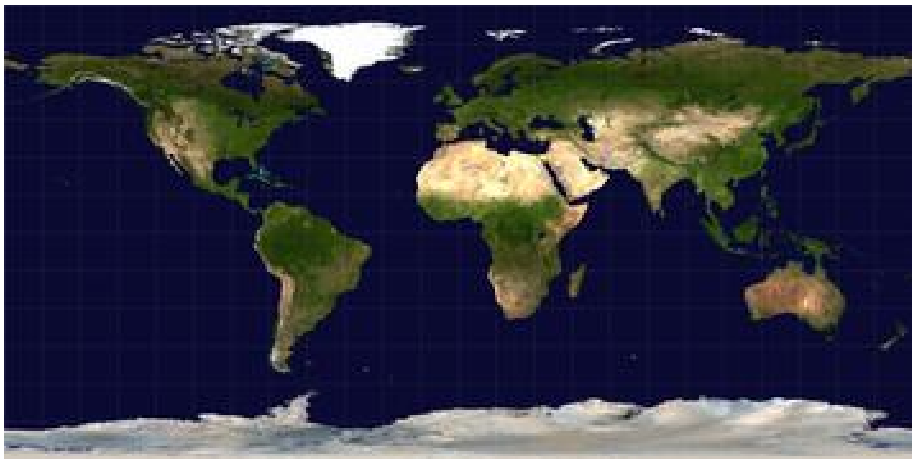


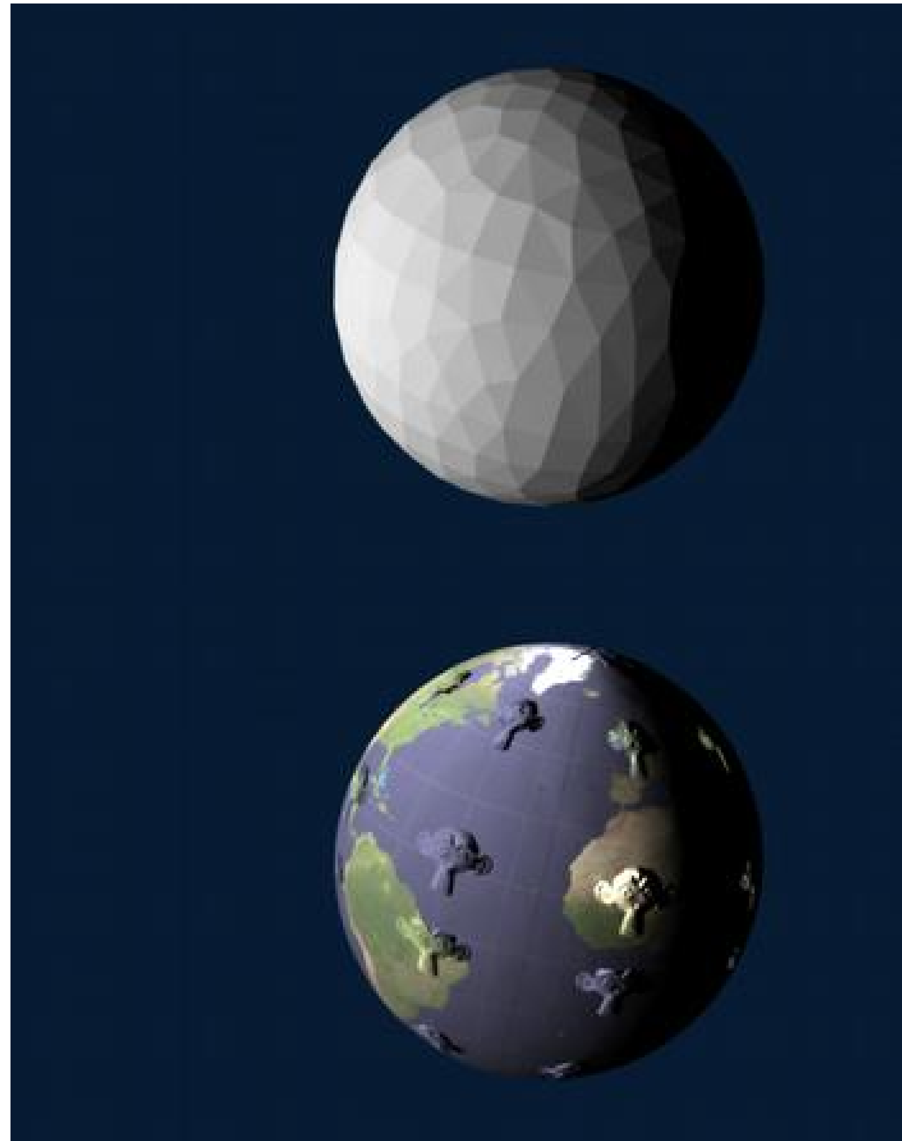
  - The implementation of this scheme depends on the sofware used ... There are tutorials for Blender.
    - Once the normal map is computed, one can perform a rendering on a simplified geometry.

Computer Graphics

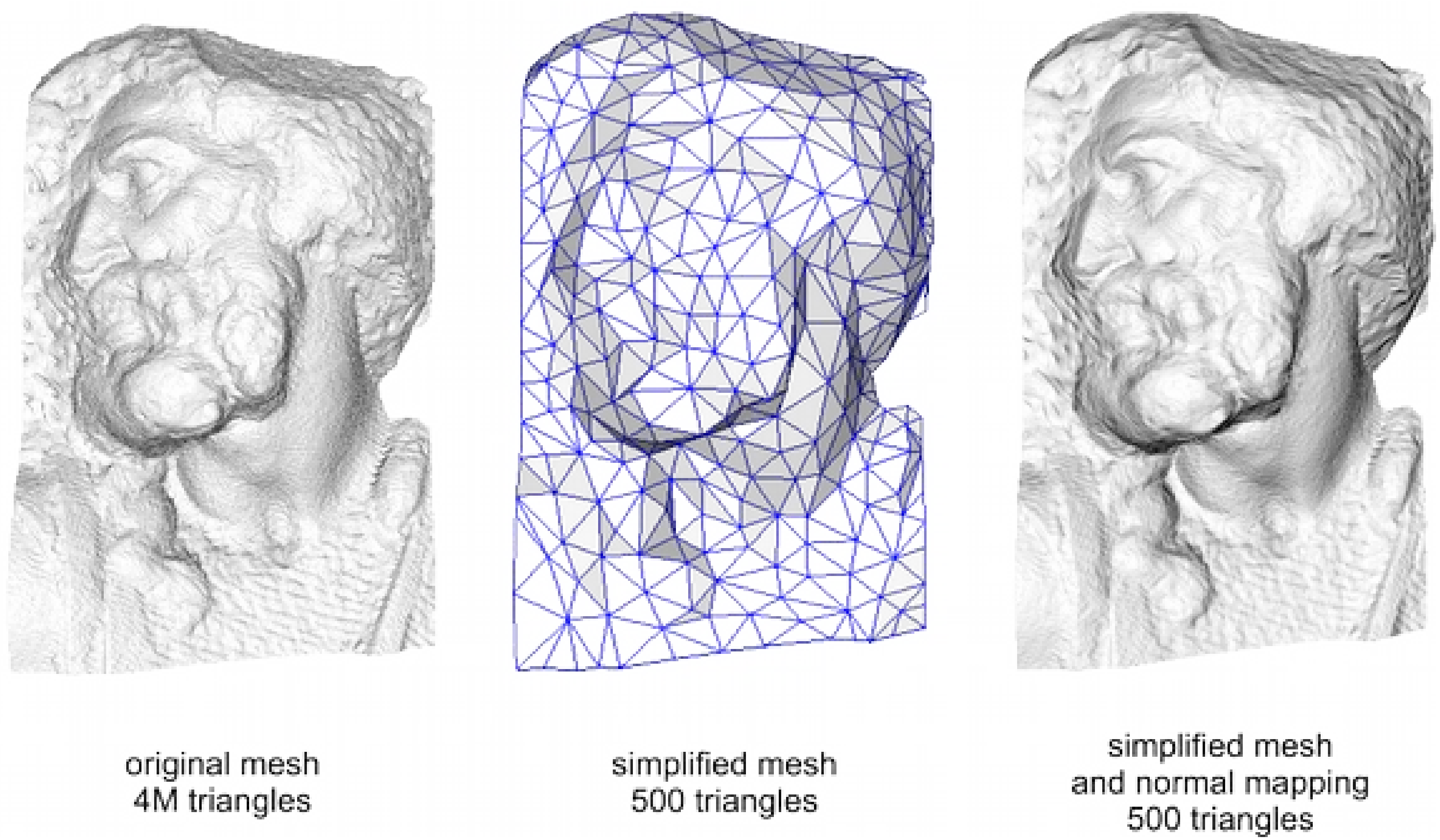
Textures

- Result





- Shades on a sculpture



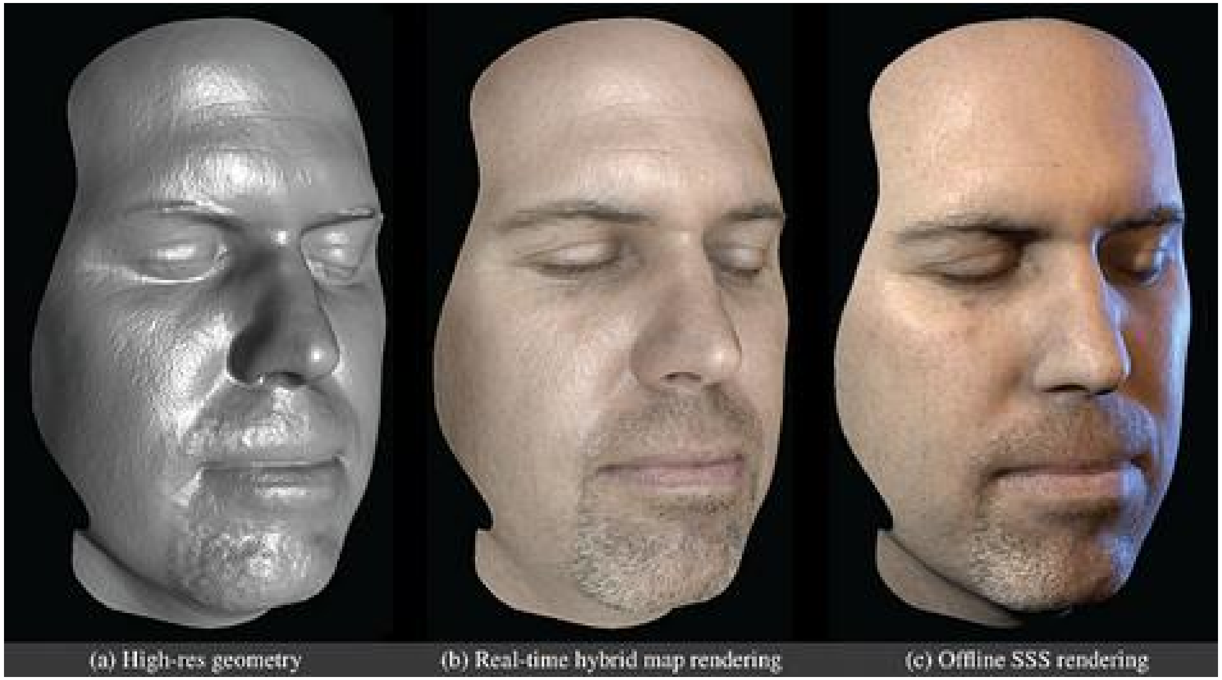
wikipedia

Textures

- Normal maps: a tool for generating realistic images using a simplified geometry
  - The overall shape of the object is approximated by a coarse mesh
  - Details are approached at each point by the knowledge on the one hand of the real normal, and on the other hand, of the color (two textures)
- Differences with respect to technique 2?
  - Accuracy: normals are accurate (close to the discretization)

    In the other case (slide 89), they are evaluated from the displacement map by differentiation. However, the relative error in the derivative is MUCH greater than that of the original variable.
    It is therefore much better to store it directly !

- (Very) realistic rendering



Wan-Chun Ma Tim Hawkins Pieter Peers Charles-Felix Chabert Malte Weiss Paul Debevec
University of Southern California Institute for Creative Technologies