Profilometry to BRDF to microstructure control in 3D printing

Jeppe Revall Frisvad

Technical University of Denmark

August 2019

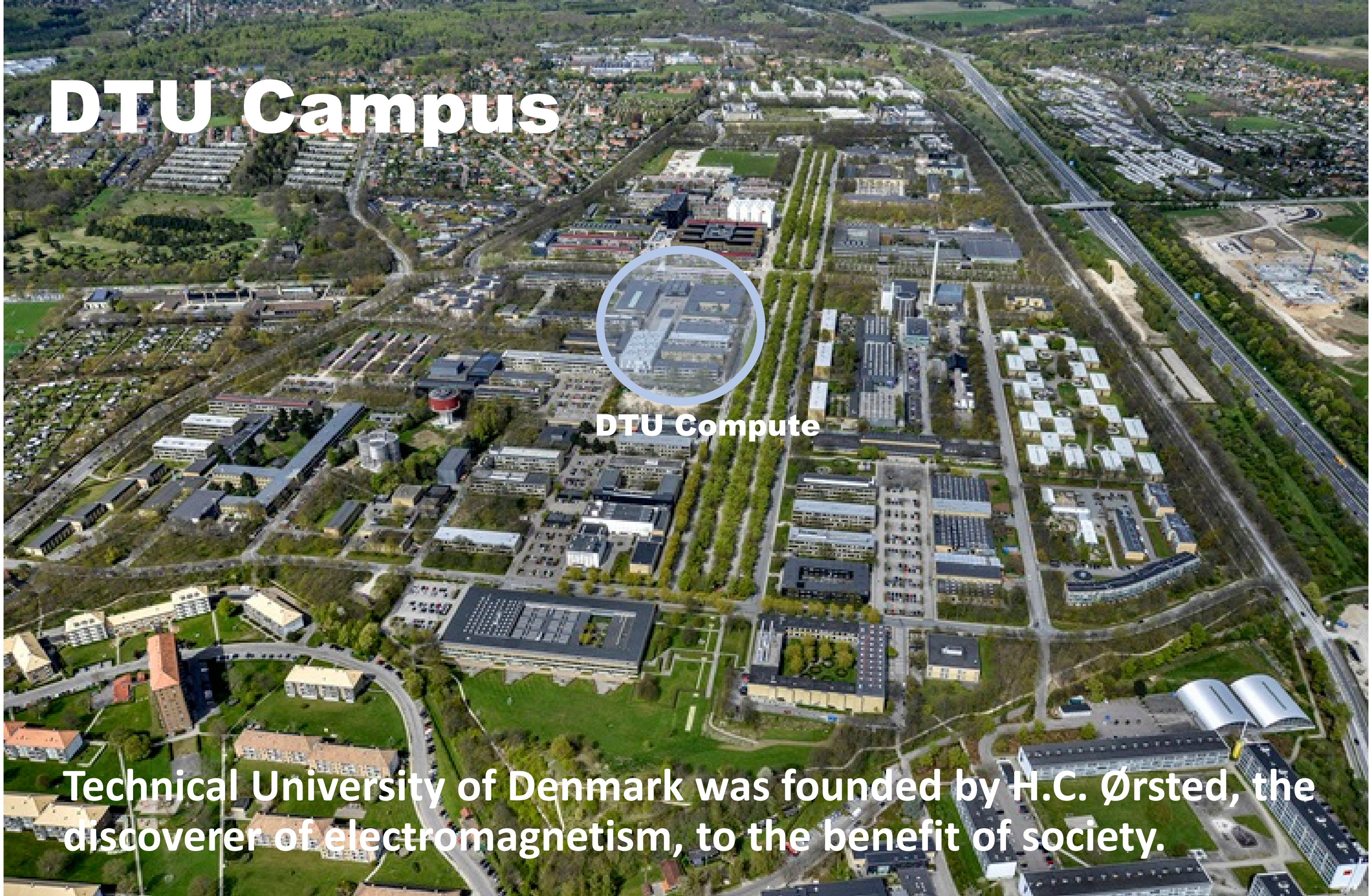
2019 Summer of Contact PhD School

On Advances in Macroscopic Friction Modelling for Fast Simulators used in Robotics, Engineering and Computer Graphics

Agenda

- Introduction
- Tools and measurements
- Multiscale modeling
- Computing the BRDF using Monte Carlo simulation
- Procedural modeling of surface microgeometry
- Applying microgeometry to surfaces in 3D printing

DTU Campus



Technical University of Denmark was founded by H.C. Ørsted, the discoverer of electromagnetism, to the benefit of society.

DTU Compute …

… spans the entire spectrum from fundamental mathematics across mathematical modelling to computer science, which is the basis of the modern digital world.

11 research sections, 400 employees, 100 permanent academic staff members (faculty)

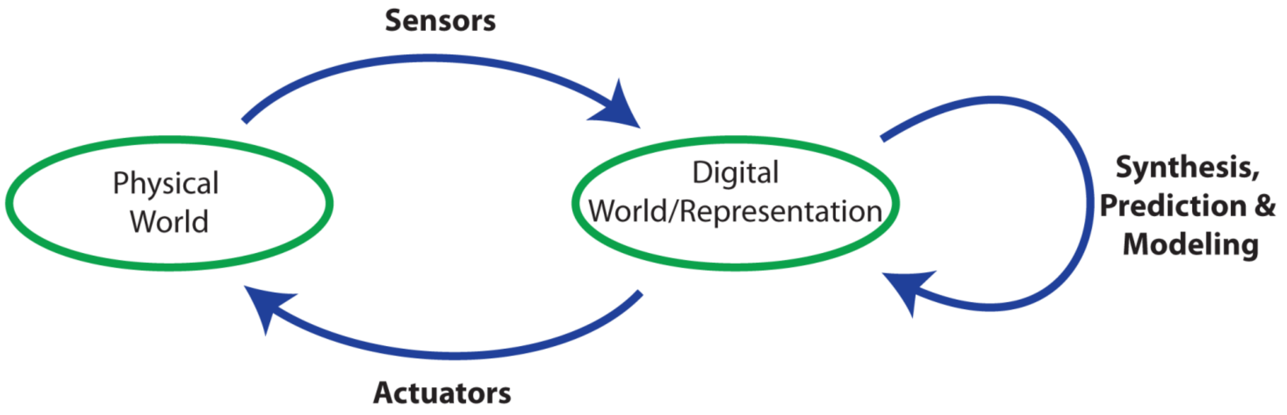
Section for Image Analysis and Computer Graphics

statistical
IMAGE ANALYSIS
medical

statistical
COMPUTER VISION
industrial

3D scan and print
GEOMETRIC DATA
processing

modeling
COMPUTER GRAPHICS
rendering

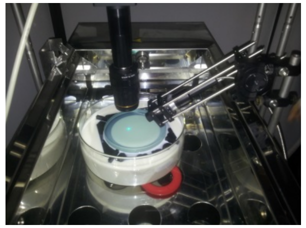

Sensors
Physical World
Digital World/Representation
Synthesis, Prediction & Modeling
Actuators

Jeppe Revall Frisvad jerf@dtu.dk https://people.compute.dtu.dk/jerf/

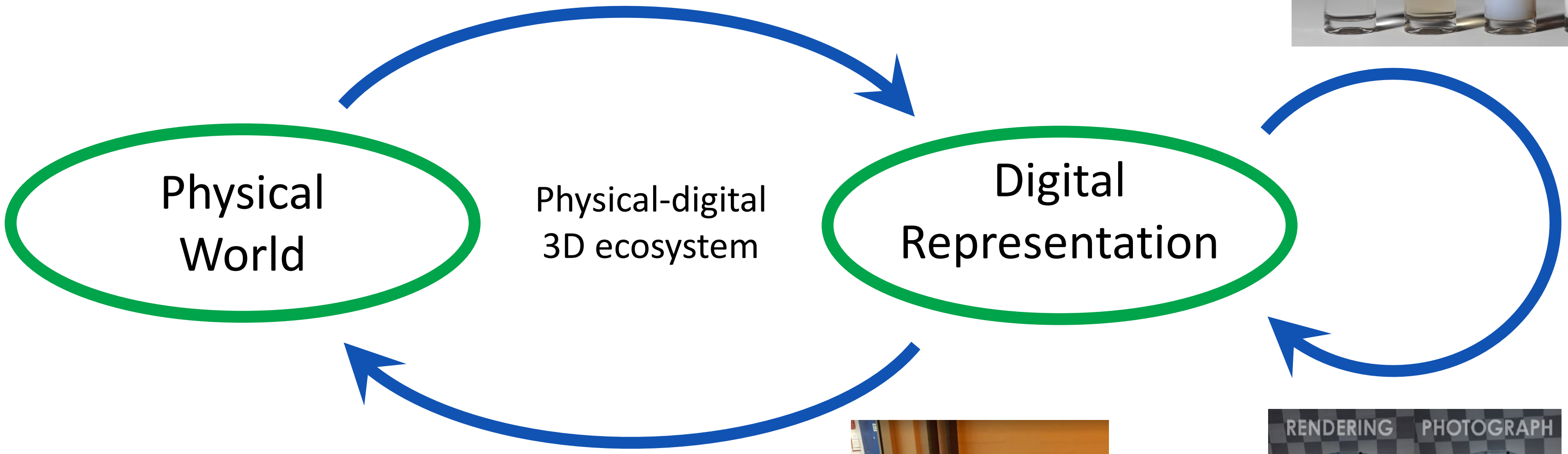
Research examples

[Eiriksson *et al.* 2016]

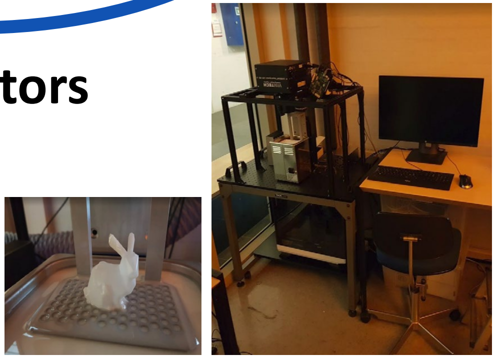
[Abildgaard *et al.* 2015]



[Frisvad *et al.* 2007]


Sensors
Physical World
Physical-digital 3D ecosystem
Digital Representation
Synthesis, Prediction & Modeling
Actuators

[Luongo *et al.* 2018]



[Eiriksson 2018]

RENDERING
PHOTOGRAPH

[Stets *et al.* 2017]

[Dal Corso *et al.* 2016]

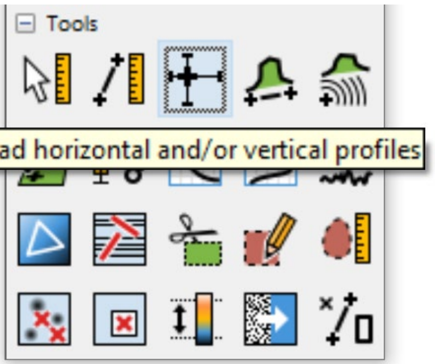
Agenda

- Introduction
- Tools and measurements
- Multiscale modeling
- Computing the BRDF using Monte Carlo simulation
- Procedural modeling of surface microgeometry
- Applying microgeometry to surfaces in 3D printing

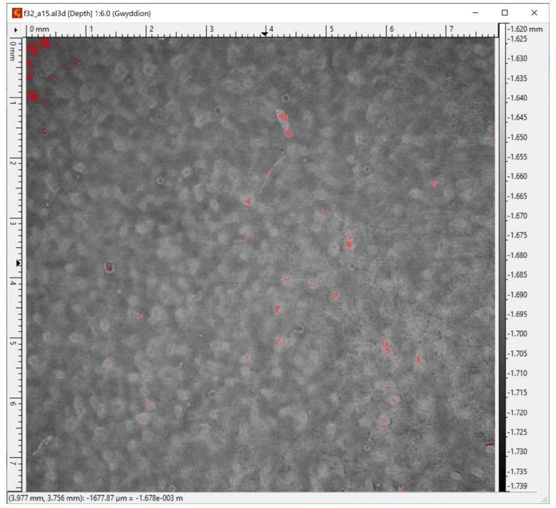
Measuring surface microgeometry

- Alicona Infinite Focus
  - Non-contact
  - Optical
  - Depth by focus-variation
  - Vertical resolution depending on choice of magnification
    - From x2.5: 2300 nm
    - Down to x100: 10 nm
  - Min. measurable roughness:
    - Sa: 3.5 down to 0.015 (arithmetic average height)
  - Output: xxx.al3d file

Inspecting and correcting the data



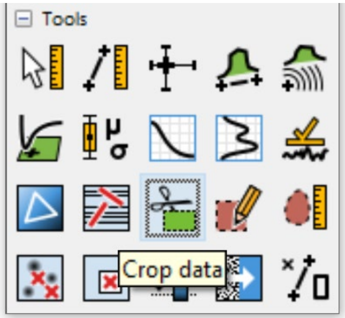
- Free tool for analysis of height fields obtained by microscopy
  Gwyddion: http://gwyddion.net/
- File→Open... (it opens .al3d files)
- Data inspection: Tools – Read horizontal and/or vertical profiles
  Mode: cross / horizontal / vertical (click the image to extract a profile)
- Data correction (outlier removal and adjustment of base):
  - Data Process→Correct Data→Mask of Outliers
    Data Process→Correct Data→Remove Data Under Mask
    Data Process→Mask→Remove Mask
  - Data Process→Level→Flatten Base
- Process: Remove outliers, flatten base, remove outliers
- File→Save as... (store processed data as a .gwy file)

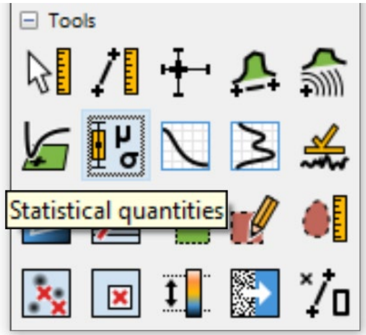


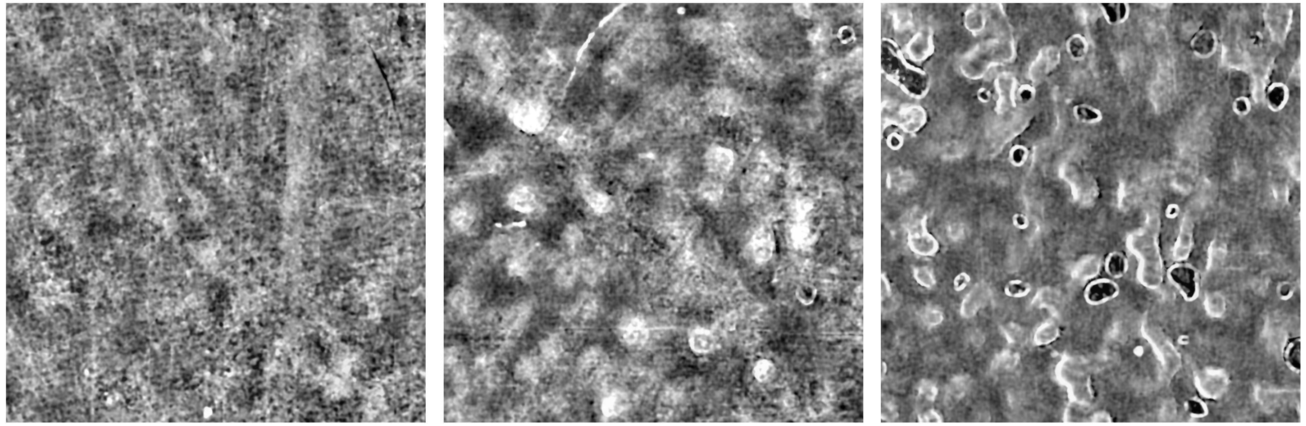
(3.977 mm, 3.756 mm): -1677.87 µm = -1.678e-003 m

Profilometry and data export

- Cutout: Use the Crop data tool with Create new image checked
- Roughness measurement along a line:
  Use the Calculate roughness parameters tool
- Surface roughness: Use the Statistical quantities tool
- Export using File→Save as...
  Choose an image file type (.png, for example)
  Remove decorations (Value scale, Lateral scale, frame, etc.)

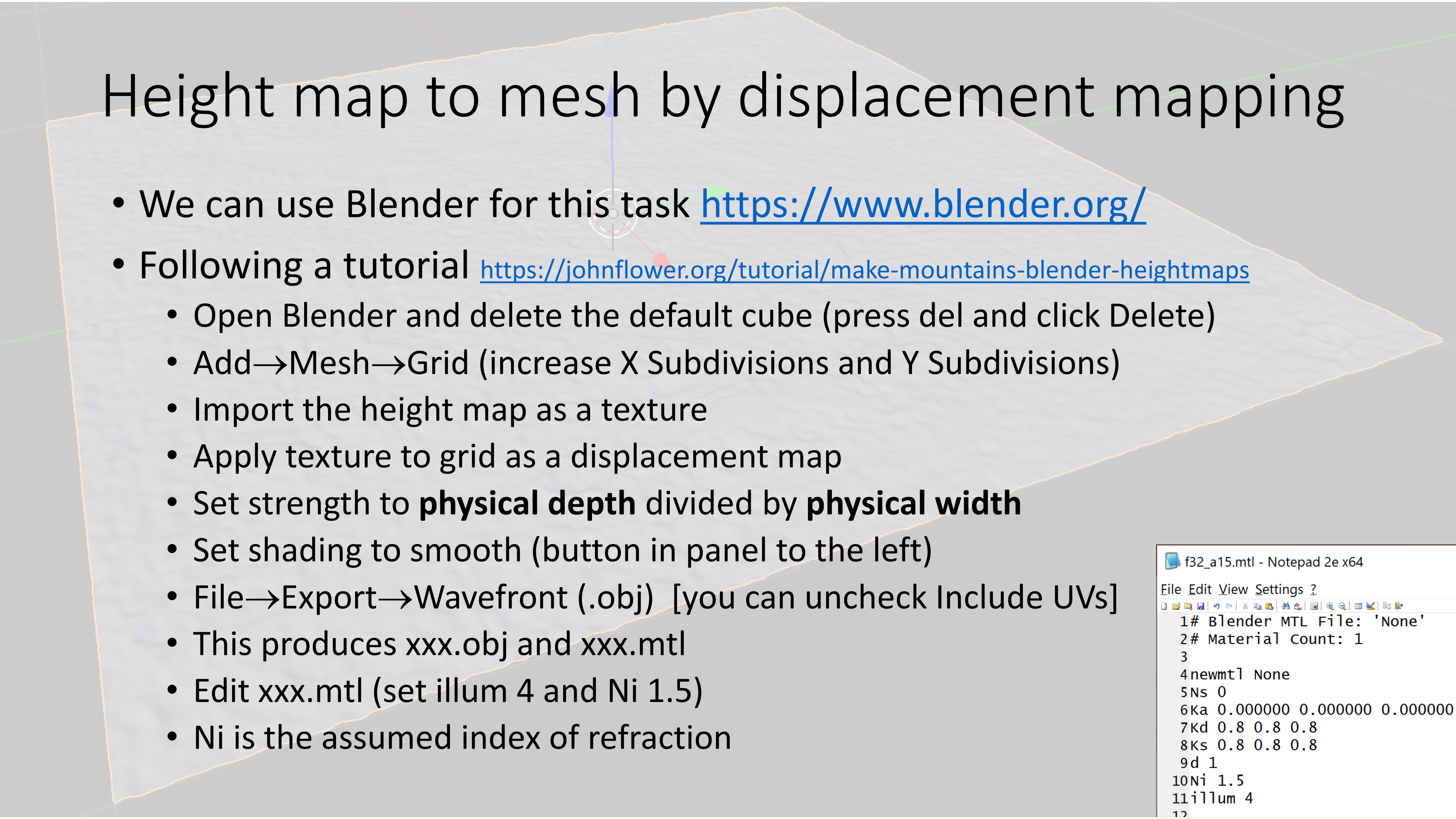




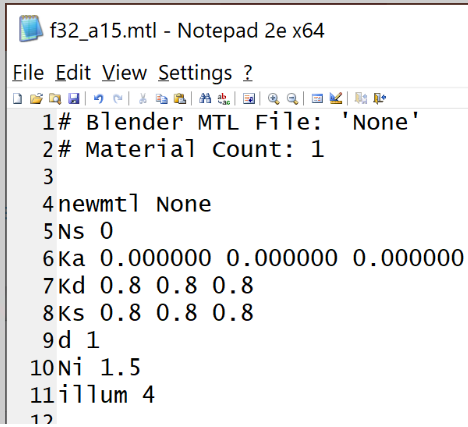


Note the **physical width** of the image and the **physical depth** that the grayscale values correspond to. These are available from the Statistical Quantities tool.

Height map to mesh by displacement mapping



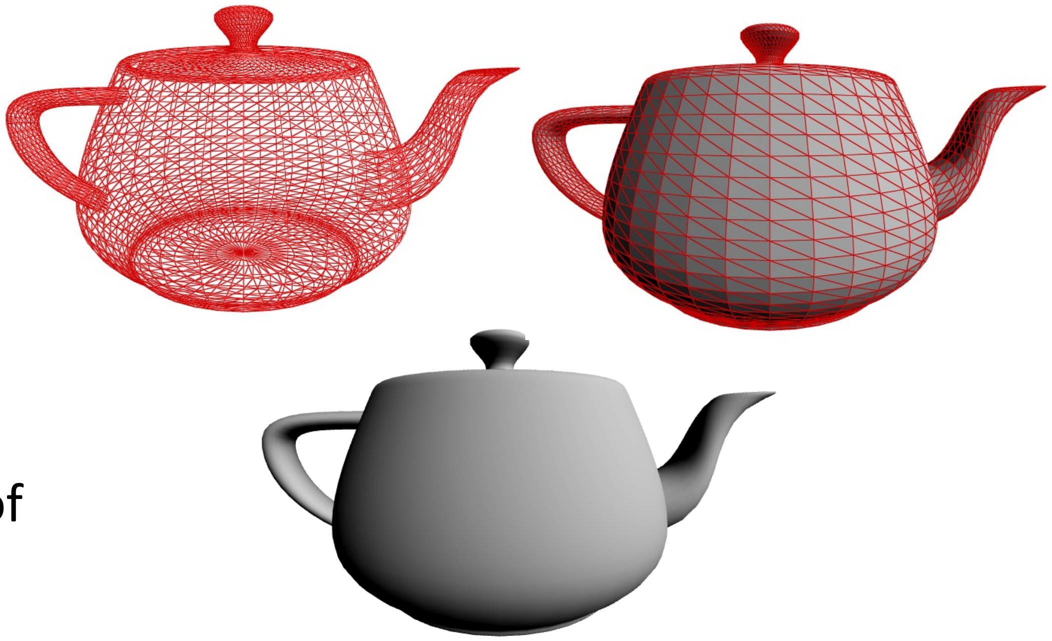
- We can use Blender for this task https://www.blender.org/
- Following a tutorial https://johnflower.org/tutorial/make-mountains-blender-heightmaps
  - Open Blender and delete the default cube (press del and click Delete)
  - Add→Mesh→Grid (increase X Subdivisions and Y Subdivisions)
  - Import the height map as a texture
  - Apply texture to grid as a displacement map
  - Set strength to **physical depth** divided by **physical width**
  - Set shading to smooth (button in panel to the left)
  - File→Export→Wavefront (.obj) [you can uncheck Include UVs]
  - This produces xxx.obj and xxx.mtl
  - Edit xxx.mtl (set illum 4 and Ni 1.5)
  - Ni is the assumed index of refraction



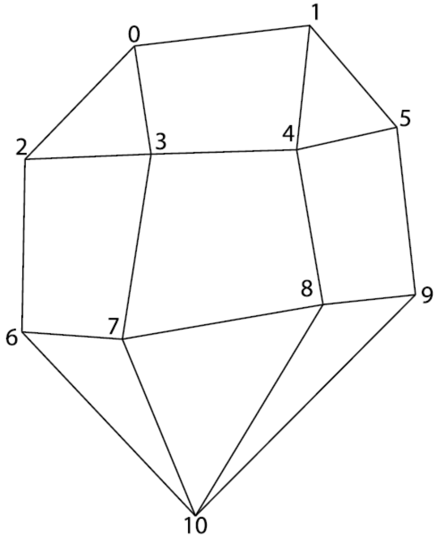
```
# Blender MTL File: 'None'
# Material Count: 1

newmtl None
Ns 0
Ka 0.000000 0.000000 0.000000
Kd 0.8 0.8 0.8
Ks 0.8 0.8 0.8
d 1
Ni 1.5
illum 4
```

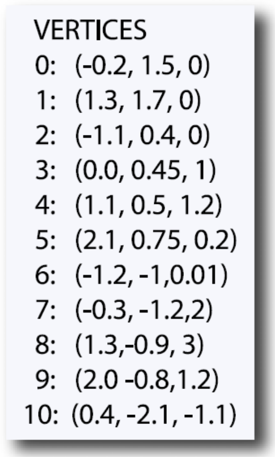
What is a mesh?

- Surface geometry is often modeled by a collection of triangles, where some of them share edges (a triangle mesh).
- Triangles provide a discrete representation of an arbitrary surface. See teapot example.

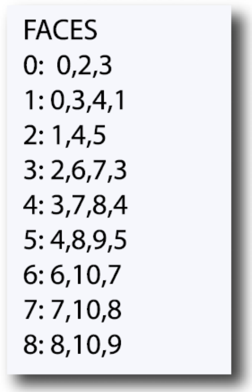


- The **indexed face set** is a popular data representation of polygon meshes.
- Any polygon mesh can be converted to a triangle mesh.
- A .obj file contains indexed face sets.





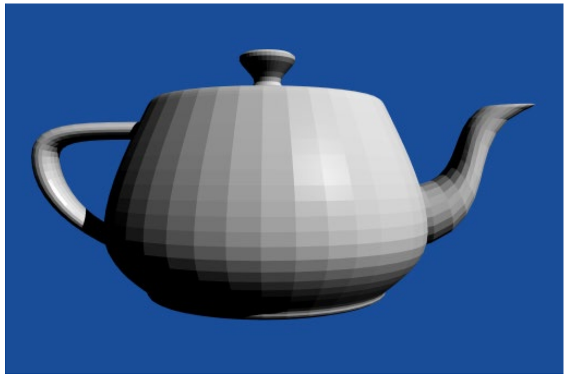
VERTICES	
0:	(-0.2, 1.5, 0)
1:	(1.3, 1.7, 0)
2:	(-1.1, 0.4, 0)
3:	(0.0, 0.45, 1)
4:	(1.1, 0.5, 1.2)
5:	(2.1, 0.75, 0.2)
6:	(-1.2, -1,0.01)
7:	(-0.3, -1.2,2)
8:	(1.3,-0.9, 3)
9:	(2.0 -0.8,1.2)
10:	(0.4, -2.1, -1.1)



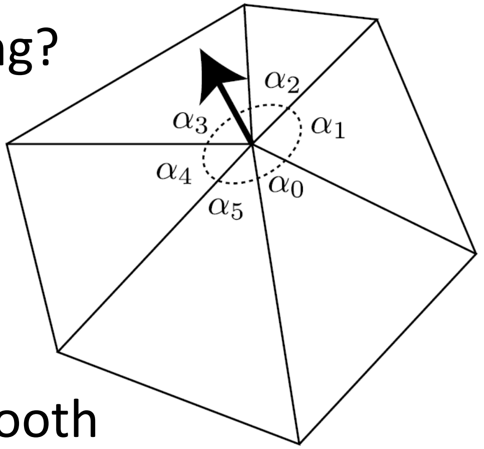
FACES	
0:	0,2,3
1:	0,3,4,1
2:	1,4,5
3:	2,6,7,3
4:	3,7,8,4
5:	4,8,9,5
6:	6,10,7
7:	7,10,8
8:	8,10,9

Figure from Bærentzen et al. Guide to Computational Geometry Processing, Springer, 2012.

What is then a smooth mesh?

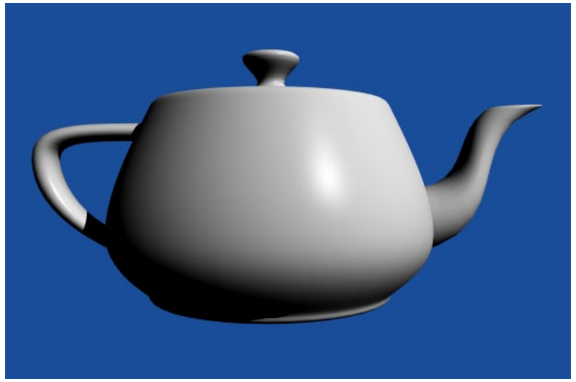


- Triangles are flat. Their **geometric normal**s lead to flat shading.
- How do we make the object smooth? Interpolated per-vertex lighting?
- What is the normal in a vertex?
  The angle-weighted pseudo-normal is a good choice.



$$\vec{n} = \frac{\sum \alpha_i \vec{n}_i}{\|\sum \alpha_i \vec{n}_i\|}$$

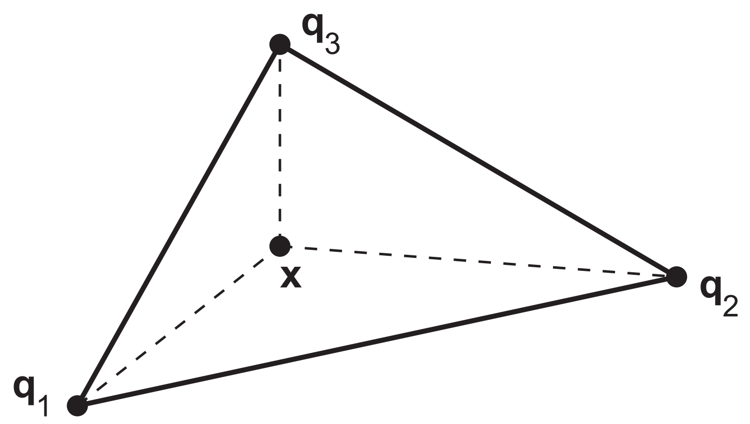
- Another indexed face set is created for the vertex normals.
- Interpolation of the vertex normals across each triangle leads to smooth shading.
- The interpolated normal is called the **shading normal**.



Linear interpolation across triangles

- A point $\boldsymbol{x}$ in a triangle is given by a weighted average of the triangle vertices $(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3)$:

$$\boldsymbol{x} = \alpha\boldsymbol{q}_1 + \beta\boldsymbol{q}_2 + \gamma\boldsymbol{q}_3\,, \qquad \alpha + \beta + \gamma = 1$$



- The weights (α, β, γ) are the barycentric coordinates.
- The point is in the triangle if $\alpha, \beta, \gamma \in [0;\ 1]$. That is,

$$\alpha \geq 0 \ \text{ and } \ \beta \geq 0 \ \text{ and } \ \alpha + \beta \leq 1$$

- Replace the triangle vertices $(\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3)$ by vertex normals and normalize to get the interpolated normal.

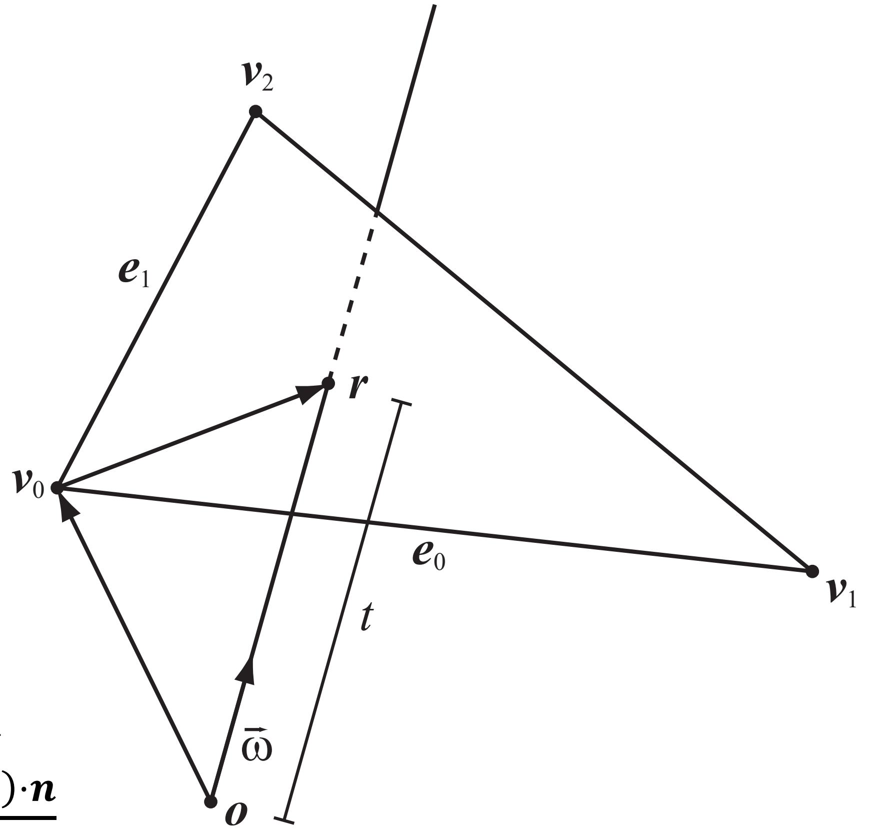
Ray-triangle intersection

- Ray: $\boldsymbol{r}(t) = \boldsymbol{o} + t\vec{\omega}, \quad t \in [t_{\min}, t_{\max}]$
- Triangle: $\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2$
- Edges and geometric normal:

  $\boldsymbol{e}_0 = \boldsymbol{v}_1 - \boldsymbol{v}_0, \boldsymbol{e}_1 = \boldsymbol{v}_0 - \boldsymbol{v}_2, \boldsymbol{n} = \boldsymbol{e}_0 \times \boldsymbol{e}_1$
- Barycentric coordinates:

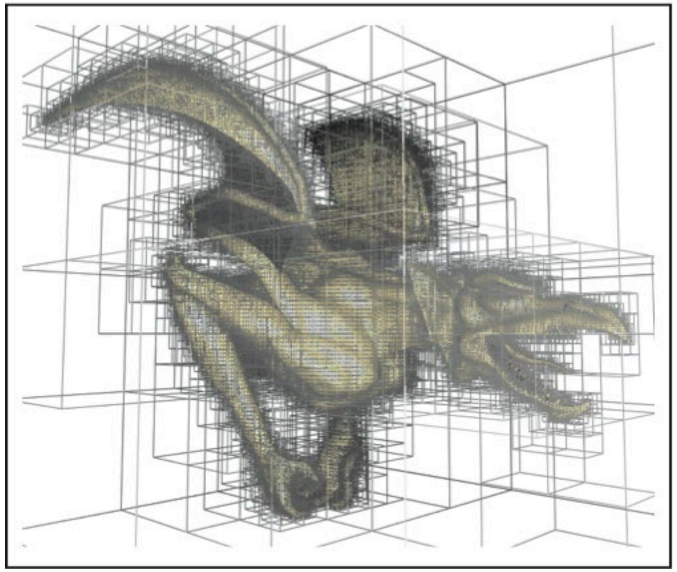
  $\boldsymbol{r}(\alpha, \beta, \gamma) = \alpha \boldsymbol{v}_0 + \beta \boldsymbol{v}_1 + \gamma \boldsymbol{v}_2 = \boldsymbol{v}_0 + \beta \boldsymbol{e}_0 - \gamma \boldsymbol{e}_1$
- The ray intersects the triangle's plane at $t' = \frac{(\boldsymbol{v}_0 - \boldsymbol{o}) \cdot \boldsymbol{n}}{\vec{\omega} \cdot \boldsymbol{n}}$
- Find $\boldsymbol{r}(t') - \boldsymbol{v}_0$ and decompose it into portions along the edges $\boldsymbol{e}_0$ and $\boldsymbol{e}_1$ to get β and γ. Then check

$$\alpha \geq 0 \;\; \text{and} \;\; \beta \geq 0 \;\; \text{and} \;\; \alpha + \beta \leq 1$$

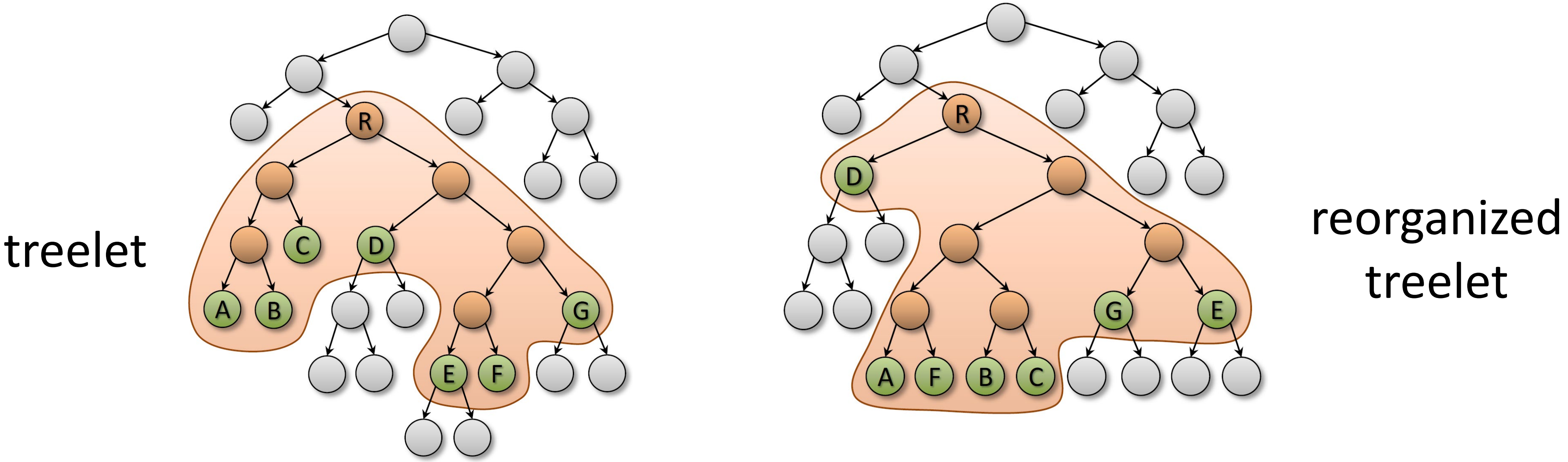


Spatial subdivision

- To model arbitrary geometry with triangles, we need many triangles.
- A million triangles and a million pixels are common numbers.
- Testing all triangles for all pixels requires 10^{12} ray-triangle intersection tests.
- If we do a million tests per millisecond, it still takes more than 15 minutes.
- This is prohibitive. We need to find the relevant triangles.
- Spatial data structures offer logarithmic complexity instead of linear.
- A million tests become twenty operations $\log_2 10^6 \approx 20$
- 15 minutes become 20 milliseconds.


Gargoyle embedded in oct tree [Hughes et al. 2014]

Treelet restructuring bounding volume hierarchy



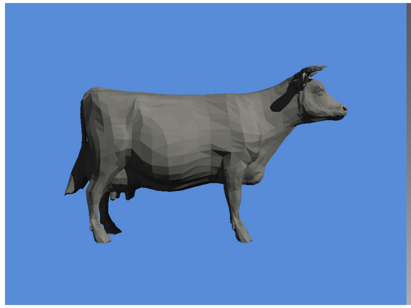
- Practical GPU-based bounding volume hierarchy (BVH) builder.
  1. Build a low-quality BVH (parallel linear BVH).
  2. Optimize node topology by parallel treelet restructuring (keeping leaves and their subtrees intact).
  3. Post-process for fast traversal.

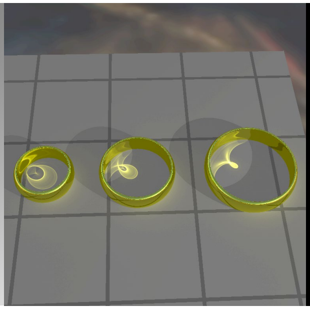
Reference:

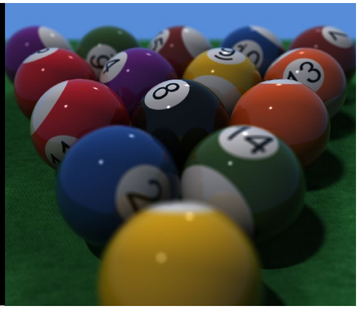
Karras, T., and Aila, T. Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of HPG 2013*, pp. 89-99. ACM, July 2013.

NVIDIA OptiX https://developer.nvidia.com/optix

- Interactive ray tracing demos: cow (sample 6), glass, PPM, PT, Cook.





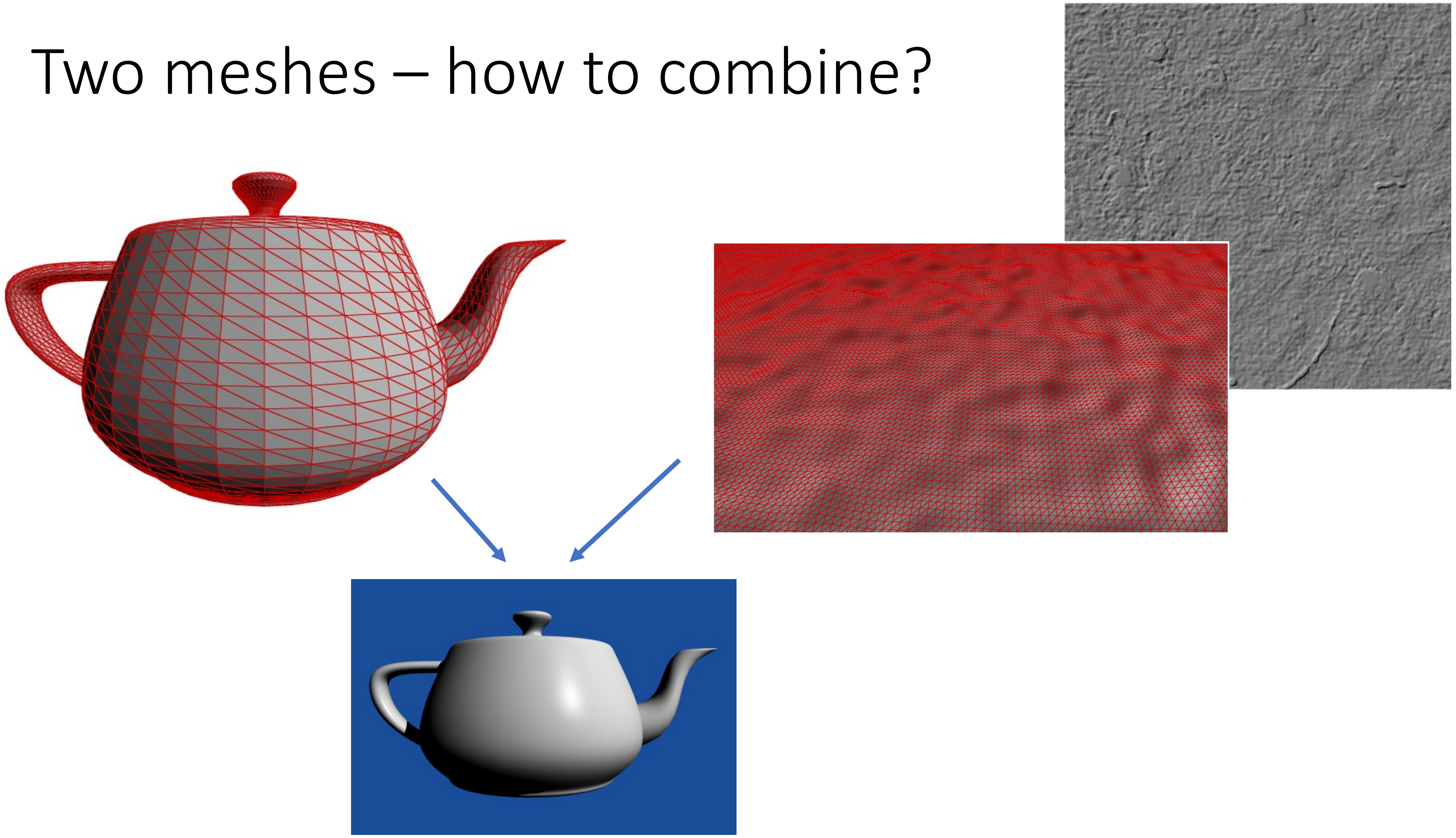


Reference:
Parker, S. G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., and Stich, M. OptiX: a general purpose ray tracing engine. *ACM Transactions on Graphics (SIGGRAPH 2010)* **29**(4):66, July 2010.

Agenda

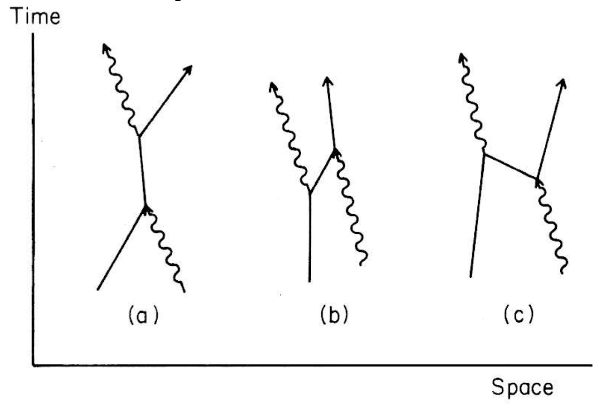
- Introduction
- Tools and measurements
- Multiscale modeling
- Computing the BRDF using Monte Carlo simulation
- Procedural modeling of surface microgeometry
- Applying microgeometry to surfaces in 3D printing

Two meshes – how to combine?

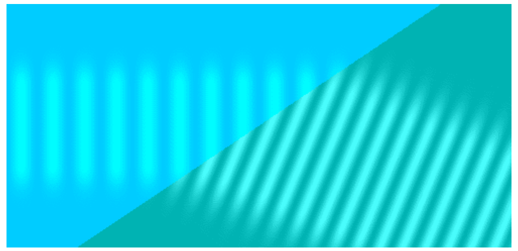


Multiscale modeling of surface geometry

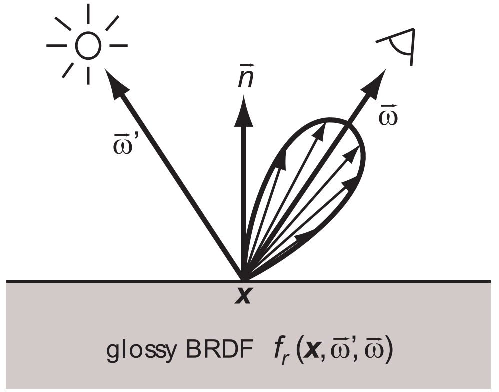
- Smallest scale: everything is quantum particles. Matter is electrons going from place to place (ignore nuclei).
  Light-matter interaction: A photon interacts with an electron.



- Microscopic scale: surfaces but no roughness. All details are defined.
  Light-matter interaction: interaction of electromagnetic waves with surfaces.



- Macroscopic scale: objects with a macrosurface and material specification (roughness/absorption) mapped onto them.
  Light-matter interaction: bidirectional (scattering) distribution functions for rays of light.



$\bar{n}$
$\bar{\omega}'$
$\bar{\omega}$
$\boldsymbol{x}$
glossy BRDF $f_r(\boldsymbol{x}, \bar{\omega}', \bar{\omega})$

Simulation to go from micro to macro

- Take the plane wave solution for Maxwell’s equations.
- The (complex) index of refraction n is a quantity summarizing the microscopic material properties (permittivity, permeability, conductivity).
- Consider an electromagnetic plane wave incident on a surface between two half-space media (of refractive indices n_i and n_t).
- By requiring continuity across the interface, we can derive:
  - The **law of reflection** (direction of reflected light)
  - The **law of refraction** (direction of transmitted light)
  - **Fresnel’s equations** for reflection (amount of reflection vs. transmitted light)
- Neglecting wavelength (assume $\lambda \to 0$), we can trace rays along the direction of energy propagation in the waves (along Poynting’s vector).
- Given surface microgeometry, we can use such **ray tracing** to compute **bidirectional scattering distribution functions** (BSDFs).

Ray tracing specular surfaces

- Fresnel's equations for reflection (R is reflected, $T = 1 - R$ is transmitted)

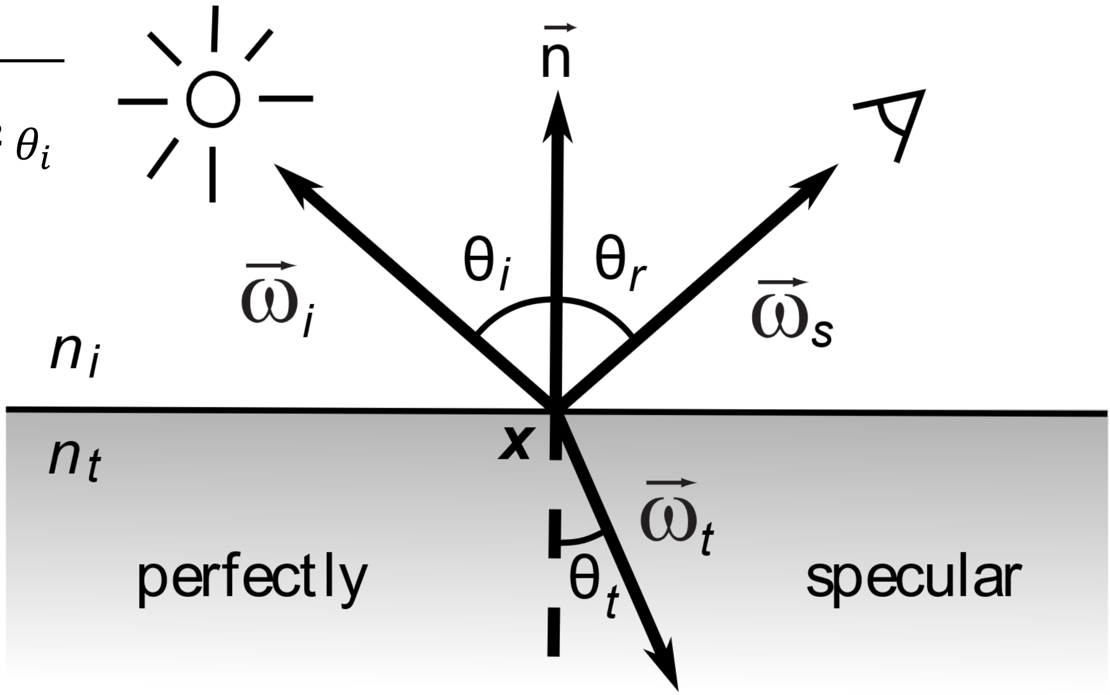
$$\tilde{r}_{\perp} = \frac{n_i \cos\theta_i - n_t \cos\theta_t}{n_i \cos\theta_i + n_t \cos\theta_t}, \qquad \tilde{r}_{\parallel} = \frac{n_t \cos\theta_i - n_i \cos\theta_t}{n_t \cos\theta_i + n_i \cos\theta_t}, \qquad R = \frac{1}{2}\left(|\tilde{r}_{\perp}|^2 + |\tilde{r}_{\parallel}|^2\right)$$

- The law of refraction

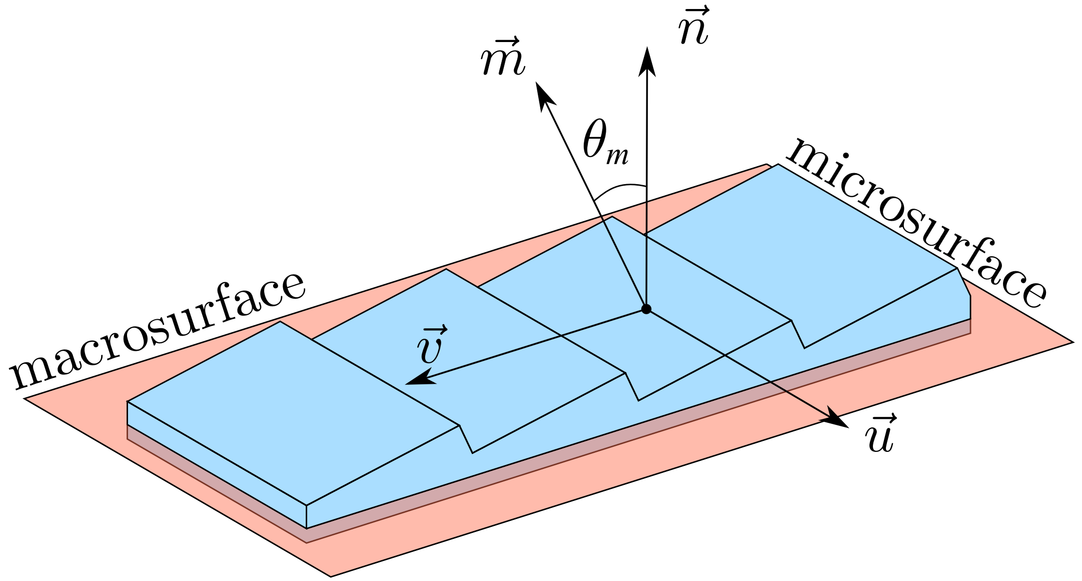
$$n_t \sin\theta_t = n_i \sin\theta_i \;\Rightarrow\; \cos\theta_t = \sqrt{1 - \left(\frac{n_i}{n_t}\right)^2 \sin^2\theta_i}$$

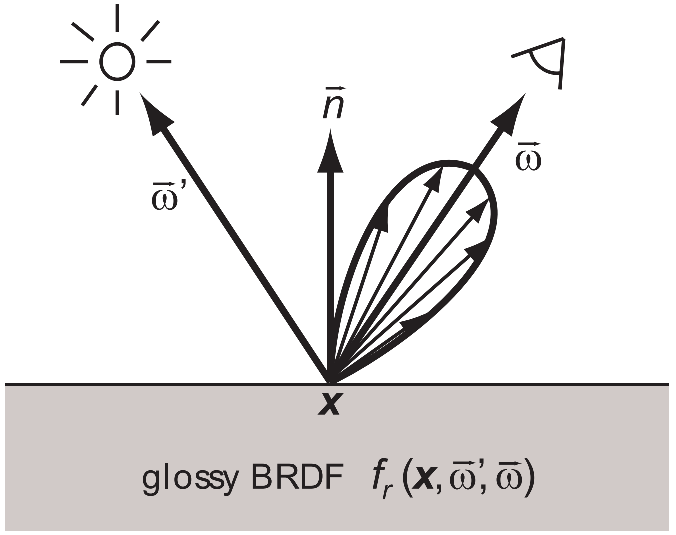
- Directions of reflected and refracted light

$$\vec{\omega}_s = 2(\vec{\omega}_i \cdot \vec{n})\vec{n} - \vec{\omega}_i$$
$$\vec{\omega}_t = \frac{n_i}{n_t}\left((\vec{\omega}_i \cdot \vec{n})\vec{n} - \vec{\omega}_i\right) - \vec{n}\cos\theta_t$$



Macro and micro surface




$\vec{n}$
$\vec{\omega}'$
$\vec{\omega}$
$\boldsymbol{x}$
glossy BRDF $f_r(\boldsymbol{x}, \vec{\omega}', \vec{\omega})$

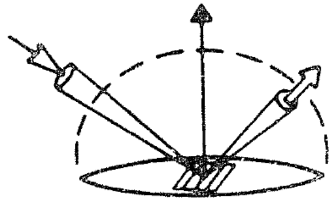
The microsurface defines the geometry of the differential area $\mathrm{d}A$ at the position $\boldsymbol{x}$ on the macrosurface.

- The BSDF defines the ratio of light incident in a surface point $\boldsymbol{x}$ from a direction $\vec{\omega}'$ that scatters into another direction $\vec{\omega}$.
- The BRDF f_r is the reflectance part of the BSDF.

Bidirectional Reflectance Distribution Function

- The definition of the BRDF: $f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_r) = \frac{\mathrm{d}L_r(\boldsymbol{x}, \vec{\omega}_r)}{\mathrm{d}E(\boldsymbol{x}, \vec{\omega}_i)}$

  The ratio of an element of reflected radiance $\mathrm{d}L_r$ to an element of irradiance $\mathrm{d}E$.

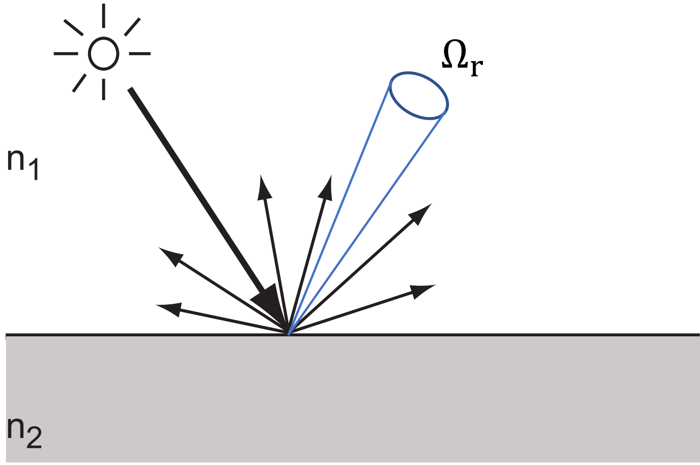


- An element of irradiance due to incident radiance within a differential element of solid angle $\mathrm{d}\omega_i$:

  $$\mathrm{d}E(\boldsymbol{x}, \vec{\omega}_i) = L_i(\boldsymbol{x}, \vec{\omega}_i) \cos\theta_i \, \mathrm{d}\omega_i$$

- Radiance is radiant flux per projected area per solid angle:

  $$L = \frac{\mathrm{d}^2\Phi}{\cos\theta \, \mathrm{d}A \, d\omega}, \qquad \Phi_r = \int_A \int_{\Omega_\mathrm{r}} L_r \cos\theta_r \, \mathrm{d}\omega_\mathrm{r} \, \mathrm{d}A$$



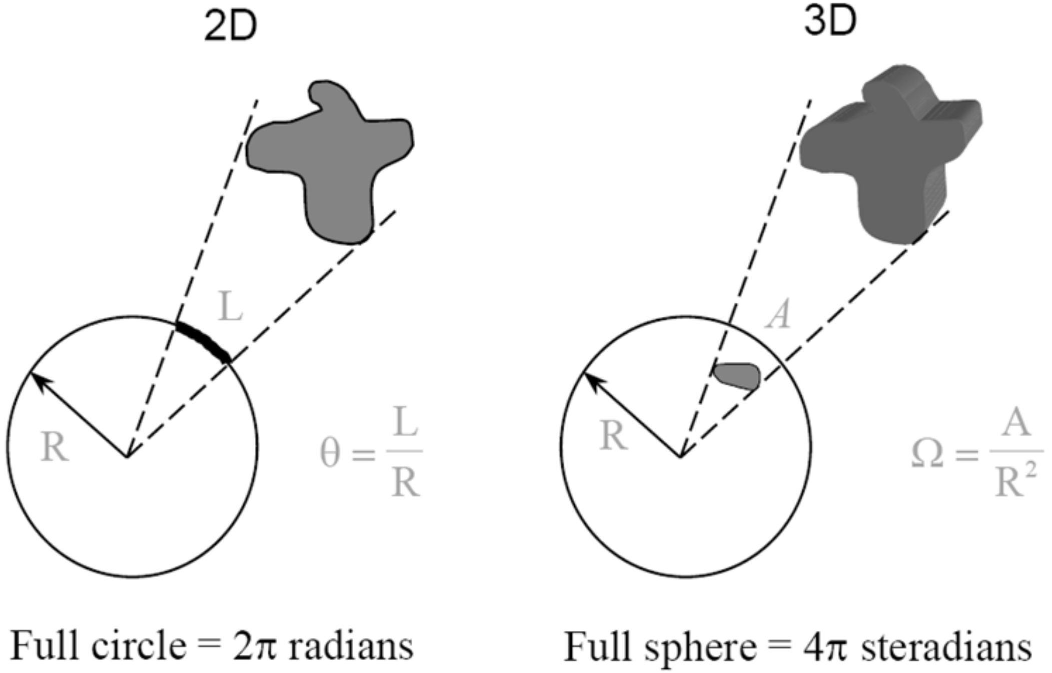
- Radiant flux Φ is a measurable quantity. So, we can set up a measurement equation for reflected radiant flux Φ_r, where A is the microsurface.
- We can then **evaluate a BRDF by solving the measurement equation**.

Agenda

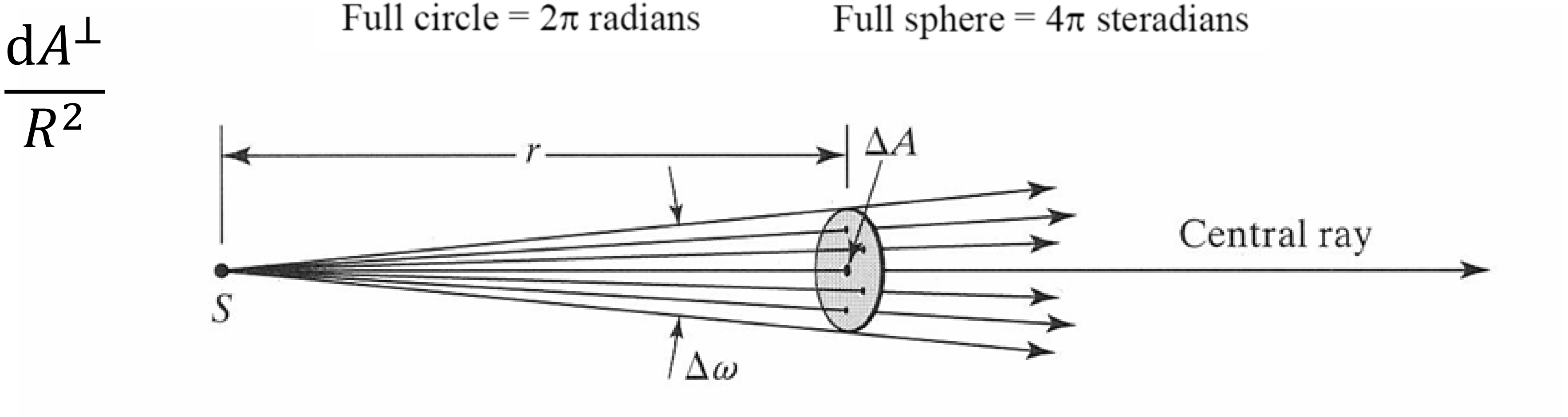
- Introduction
- Tools and measurements
- Multiscale modeling
- Computing the BRDF using Monte Carlo simulation
- Procedural modeling of surface microgeometry
- Applying microgeometry to surfaces in 3D printing

Solid angles

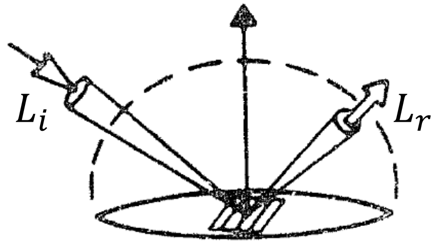
Solid angle is area on the unit sphere.



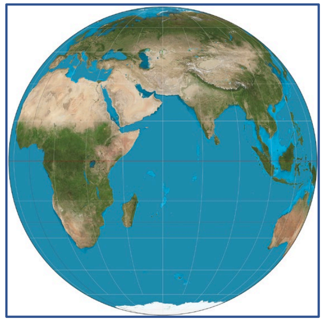
$$\mathrm{d}\omega = \frac{\mathrm{d}A^{\perp}}{R^2}$$

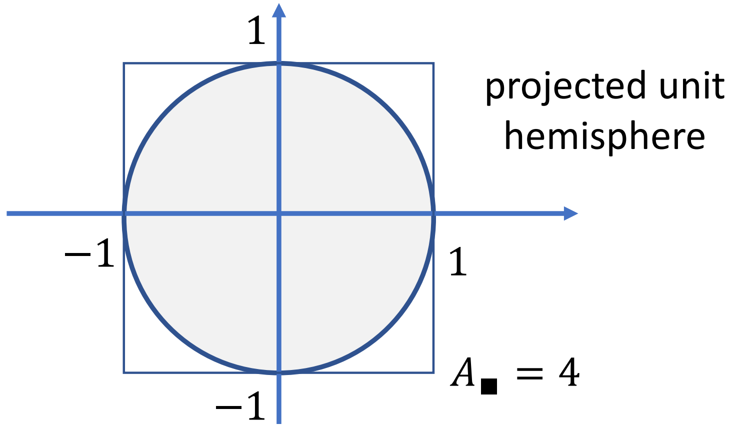


BRDF discretization



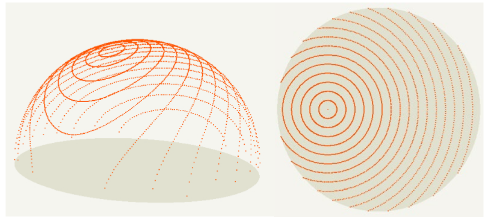
- For small enough bins $L_r \approx \dfrac{\Phi_r(A,\Omega_r)}{\cos\Theta_r\, A\, \Omega_r}$, $L_i \approx \dfrac{\Phi_i(A,\Omega_i)}{\cos\Theta_i\, A\, \Omega_i}$
  - Θ_x is the angle between the macrosurface normal $\vec{n}$ and the direction $\vec{\Omega}_x$ in the center of the solid angle Ω_x of the bin.





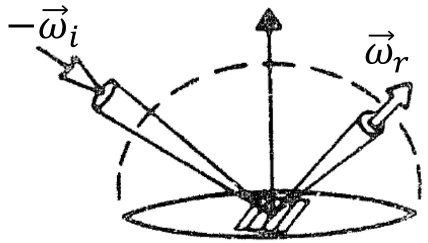
- Orthographic projection map.
- Map resolution $W \times H$

- Solid angle is area on a unit sphere.
- The area of a pixel is then $A_p = \dfrac{4}{WH} \approx \Omega_r \cos\Theta_r$



T. Tongbuasirilai, J. Unger, J. Kronander, and M. Kurt. Compact and intuitive data-driven BRDF models. *The Visual Computer*, 2019. To appear.

Reflected radiance equation



- The definition of the BRDF: $f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_r) = \dfrac{\mathrm{d}L_r(\boldsymbol{x}, \vec{\omega}_r)}{\mathrm{d}E(\boldsymbol{x}, \vec{\omega}_i)}$
- Here $\boldsymbol{x}$ is a location on the macrosurface.
- The reflected radiance is then

$$L_r(\boldsymbol{x}, \vec{\omega}_r) = \int_{2\pi} f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_r)\, \mathrm{d}E(\boldsymbol{x}, \vec{\omega}_i) = \int_{2\pi} f_r(\boldsymbol{x}, \vec{\omega}_i, \vec{\omega}_r)\, L_i(\boldsymbol{x}, \vec{\omega}_i) \cos\theta_i \, \mathrm{d}\omega_i$$

- Suppose we consider small solid angle bins, then

$$L_r = f_r(\boldsymbol{x}, \Omega_i, \Omega_r) \int_{\Omega_i} L_i(\boldsymbol{x}, \vec{\omega}_i) \cos\theta_i \, \mathrm{d}\omega_i = f_r(\boldsymbol{x}, \Omega_i, \Omega_r) E(\boldsymbol{x}, \Omega_i)$$

- We can use this to set up an expression for evaluating the BRDF

$$f_r(\boldsymbol{x}, \Omega_i, \Omega_r) = \frac{L_r}{E} = \frac{\Phi_r(A, \Omega_r)}{\cos\Theta_r \; A \; \Omega_r \; E(\boldsymbol{x}, \Omega_i)}$$

BRDF measurement

- BRDF: $f_r(\boldsymbol{x}, \Omega_i, \Omega_r) = \dfrac{\Phi_r(A, \Omega_r)}{\cos \Theta_r \, A \, \Omega_r \, E(\boldsymbol{x}, \Omega_i)}$
- Reflected radiant flux: $\Phi_r(A, \Omega_r) = \int_A \int_{\Omega_\mathrm{r}} L_r(\boldsymbol{x}_m, \vec{\omega}_r) \cos \theta_r \, \mathrm{d}\omega_\mathrm{r} \, \mathrm{d}A$
- Irradiance: $E(\boldsymbol{x}, \Omega_i) = \int_{\Omega_i} L_i(\boldsymbol{x}, \vec{\omega}_i) \cos \theta_i \, \mathrm{d}\omega_i$

- The Fresnel BRDF of the microsurface:

$$f_m(\boldsymbol{x}_m, \vec{\omega}_i, \vec{\omega}_r) = R(\vec{m}, \vec{\omega}_i) \frac{\delta(\vec{\omega}_r - \vec{\omega}_s)}{\cos \theta_i}$$

- The reflected radiance from a point $\boldsymbol{x}_m$ on the microsurface:

$$L_r(\boldsymbol{x}_m, \vec{\omega}_r) = \int_{2\pi} f_m(\boldsymbol{x}_m, \vec{\omega}_i, \vec{\omega}_r) \, L_i(\boldsymbol{x}_m, \vec{\omega}_i) \cos \theta_i \, \mathrm{d}\omega_i$$

- Then

$$\Phi_r(A, \Omega_r) = \int_A \int_{\Omega_\mathrm{r}} \int_{2\pi} R(\vec{m}, \vec{\omega}_i) \delta(\vec{\omega}_r - \vec{\omega}_s) \, L_i(\boldsymbol{x}_m, \vec{\omega}_i) \, \mathrm{d}\omega_i \cos \theta_r \, \mathrm{d}\omega_\mathrm{r} \, \mathrm{d}A$$

Evaluating difficult integrals

$$\Phi_r(A,\Omega_r) = \int_A \int_{\Omega_\mathrm{r}} \int_{2\pi} R(\vec{m},\vec{\omega}_i)\delta(\vec{\omega}_r - \vec{\omega}_s)\, L_i(\boldsymbol{x}_m,\vec{\omega}_i)\,\mathrm{d}\omega_i \cos\theta_r \,\mathrm{d}\omega_\mathrm{r}\,\mathrm{d}A$$

- This measurement equation is difficult (if not impossible) to solve analytically.
- Trapezoidal integration and Gaussian quadrature only works well for smooth low-dimensional integrals.
- This integral is 6-dimensional, recursive, and could involve discontinuities.
- Three known mathematical methods for solving this type of problem:
  - Truncated series expansion
  - Finite basis (discretization)
  - Sampling (random selection)
- Monte Carlo integration is probably the simplest way to use sampling.

Monte Carlo integration

- The law of large numbers

$$\Pr\left\{\frac{1}{N}\sum_{i=1}^{N} f(X_i) \to E\{f(X)\}\right\} = 1 \quad \text{for} \quad N \to \infty$$

”it is certain that the estimator goes to the expected value as the number of samples goes to infinity”

- Approximating an arbitrary integral by stochastic sampling:

$$F = \int_A f(x)\,\mathrm{d}x = \int_A \frac{f(x)}{\mathrm{pdf}(x)}\,\mathrm{pdf}(x)\,\mathrm{d}x = E\left\{\frac{f(x)}{\mathrm{pdf}(x)}\right\}$$

- Using the law of large numbers, we have the Nth estimator

$$F_N = \frac{1}{N}\sum_{i=1}^{N} \frac{f(X_i)}{\mathrm{pdf}(X_i)}$$

where X_i sampled on A and $\mathrm{pdf}(x) > 0$ for all $x \in A$.

Monte Carlo estimator

- Let us consider one (sampled) direction of incidence $\vec{\omega}_\ell \in \Omega_i$:

$$L_i(\boldsymbol{x}, \vec{\omega}_i) = \delta(\vec{\omega}_i - \vec{\omega}_\ell) L_e$$

$$E(\boldsymbol{x}, \Omega_i) = \int_{\Omega_i} L_i(\boldsymbol{x}, \vec{\omega}_i) \cos\theta_i \, \mathrm{d}\omega_i = (\vec{\omega}_\ell \cdot \vec{n}) L_e = L_e \cos\theta_\ell$$

- Emitted radiance L_e is considered constant across the microsurface.
- Because of the directional δ-functions (deterministic light paths), we need now only sample the microsurface area A.
- The Monte Carlo estimator then becomes

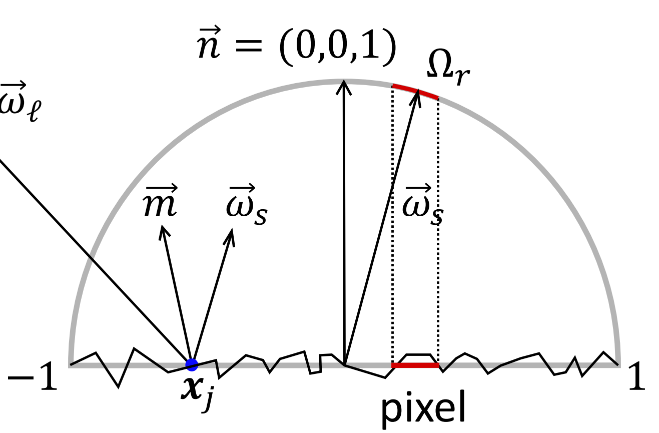
$$f_r(\boldsymbol{x}, \vec{\omega}_\ell, \Omega_r) = \frac{1}{N} \sum_{j=1}^{N} \frac{R(\vec{m}, \vec{\omega}_i) \, L_i\big(\boldsymbol{x}_j, \vec{\omega}_i\big) \cos\theta_r}{\cos\Theta_r \, A \, \Omega_r \, L_e \cos\theta_\ell \, \mathrm{pdf}(\boldsymbol{x}_\mathrm{j})} [\vec{\omega}_s \in \Omega_r]$$

- $[*]$ is an Iverson bracket, which is 1 if $*$ is true, 0 otherwise.

Computing a BRDF from microgeometry

- One randomly placed sample per pixel (jitter sampling):

$$N = WH, \qquad \mathrm{pdf}(\boldsymbol{x}_\mathrm{j}) = \frac{1}{A}, \qquad \Omega_r \cos\Theta_r \approx \frac{4}{WH}$$



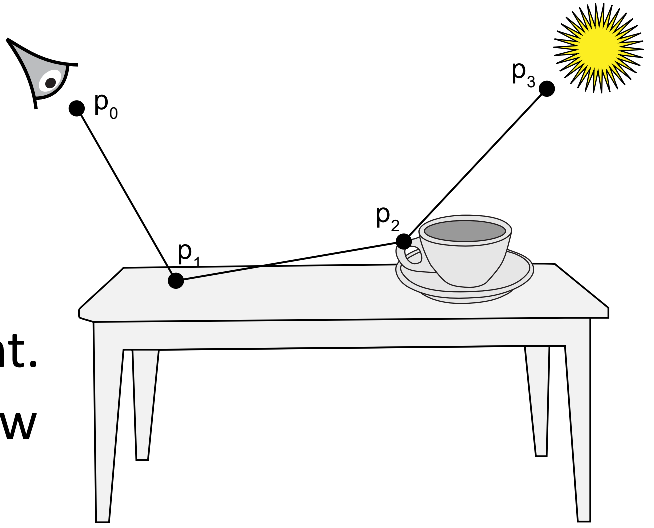
- Since the constant emitted radiance cancels out, we set $L_e = 1$.
- Suppose we always choose either reflection or transmission:
  - Russian roulette with $R(\vec{m}, \vec{\omega}_i)$ as the probability of reflection.
- The Monte Carlo estimator is then

$$f_r(\boldsymbol{x}, \vec{\omega}_\ell, \Omega_r) = \frac{1}{4(\vec{\omega}_\ell \cdot \vec{n})} \sum_{j=1}^{N} L_i(\boldsymbol{x}_j, \vec{\omega}_i)(\vec{\omega}_s \cdot \vec{m}) \ [\xi < R(\vec{m}, \vec{\omega}_i) \wedge \vec{\omega}_s \in \Omega_r]$$

- $\xi \in [0,1]$ is a continuous, uniform, random variable.
- We compute $L_i(\boldsymbol{x}_j, \vec{\omega}_i)$ by **path tracing the microsurface**.

Progressive unidirectional path tracing

1. **Generate rays** from the camera through pixel positions.
2. **Trace the rays** and evaluate the rendering equation for each ray.
3. **Randomize the position** within the pixel area to Monte Carlo integrate (measure) the radiance arriving in a pixel.



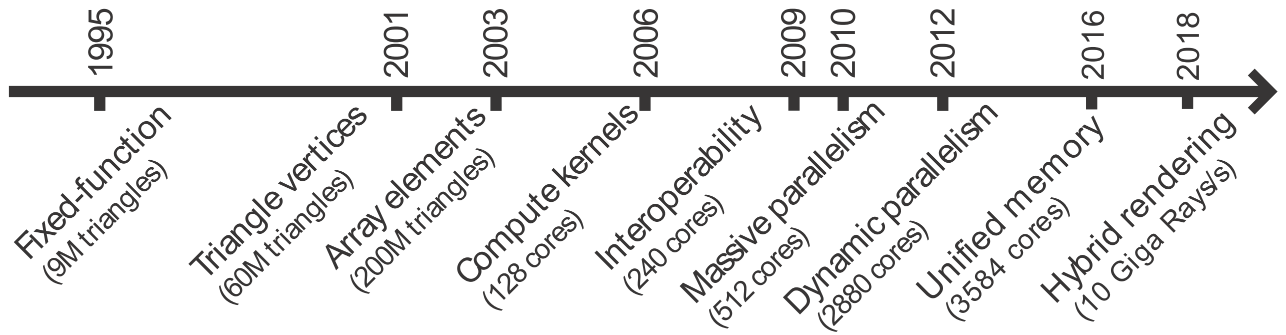
http://www.pbr-book.org/

- Noise is reduced by **progressive updates** of the measurement.
- Update the rendering result in a pixel L_j after rendering a new frame with result L_{new} using

$$L_{j+1} = \frac{L_{\text{new}} + jL_j}{j+1}$$

- Progressive (stop and go) rendering is convenient for several reasons:
  - No need to start over.
  - Result can be stored and renfined later if need be.
  - Convergence can be inspected during progressive updates.

Timeline on the programmability of the GPU


1995
2001
2003
2006
2009
2010
2012
2016
2018
Fixed-function (9M triangles)
Triangle vertices (60M triangles)
Array elements (200M triangles)
Compute kernels (128 cores)
Interoperability (240 cores)
Massive parallelism (512 cores)
Dynamic parallelism (2880 cores)
Unified memory (3584 cores)
Hybrid rendering (10 Giga Rays/s)

1995 Fixed-function rasterization pipeline in hardware.

2001 Vertex shaders (rst programmable part of the pipeline).

2003 Fragment/pixel shaders (GPGPU).

2006 Unied shaders (CUDA) and geometry shaders.

2008 Double precision arithmetics.

2009 Compute shaders (interoperability) and tesselation shaders.

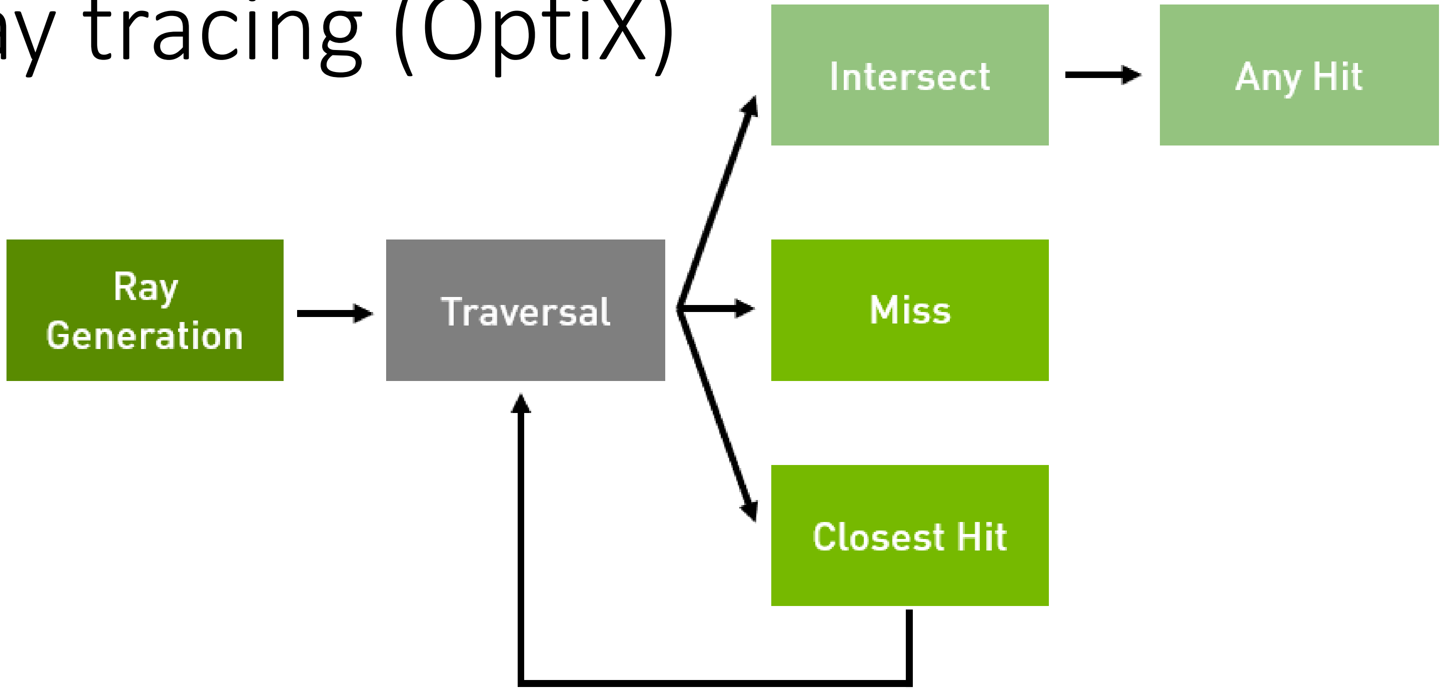
2010 Streaming multiprocessor architecture. **Programmable ray tracing pipeline on the GPU**.

2012 Dynamic parallelism (threads spawn threads).

2016 Unied memory (on demand data migration and dynamic memory allocation).

2018 **Hybrid rendering** (CUDA cores, **RT cores**, DNN tensor cores).

GPU ray tracing (OptiX)



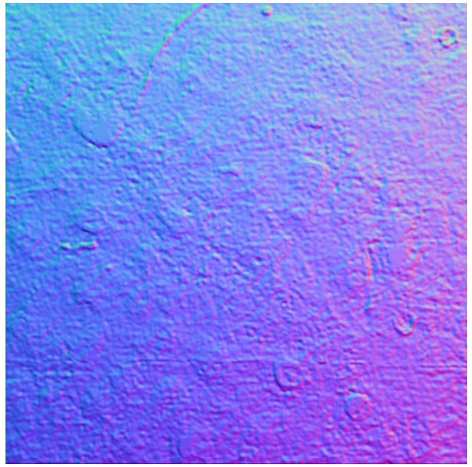
- Sample surface positions in the “Ray Generation” program and do progressive updates.
- Return ray direction from the “Miss” program
- Implement shader for specular surfaces in the “Closest Hit” program.

Ray direction as the result (miss program)

```
// Standard ray variables
rtDeclareVariable(Ray, ray, rtCurrentRay, );
rtDeclareVariable(PerRayData_radiance, prd, rtPayload, );

RT_PROGRAM void miss()
{
   prd.result = ray.direction;
}
```

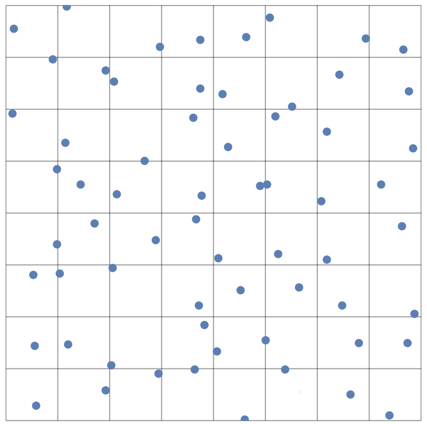
```
// Payload for radiance ray type
struct PerRayData_radiance
{
  optix::float3 result;
  int depth;
  unsigned int seed;
  optix::float3 hit_normal;
};
```

- Set up the scene so that the microsurface mesh covers the square observed by the orthographic camera $[-1,1] \times [-1,1]$.
- Trace light rays toward observed points.
- Return the final ray direction of each path.
- Use the directions to estimate the BRDF.

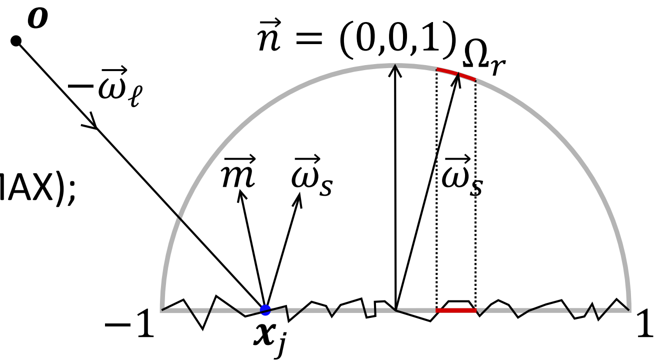

Direction of reflected ray in sampled positions.

Orthographic projection (camera program)

image plane (ip)
jitter sampling

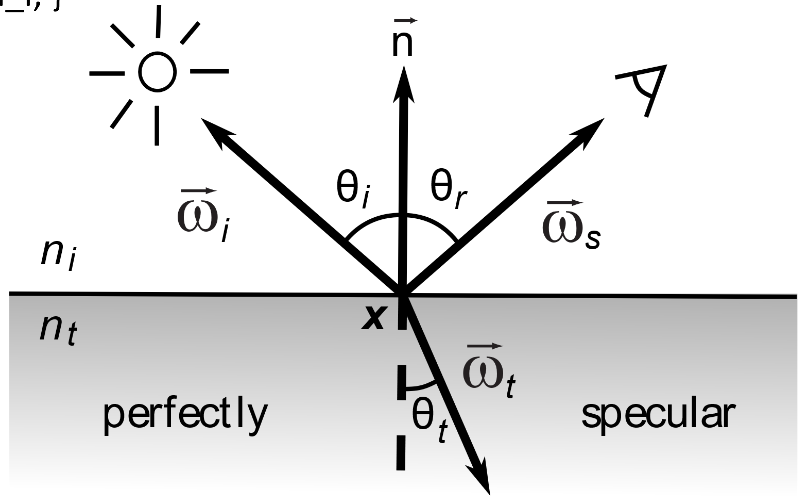


```
RT_PROGRAM void pinhole_camera()
{
  PerRayData_radiance prd;
  prd.depth = 0;
  prd.seed = tea<16>(launch_dim.x*launch_index.y + launch_index.x, frame);
  prd.result = prd.hit_normal = make_float3(0.0f);

  float2 jitter = make_float2(rnd(prd.seed), rnd(prd.seed));
  float2 ip_coords = 2.0f*jitter - 1.0f;
  Directional& light = lights[0];
  float3 origin = make_float3(ip_coords.x, ip_coords.y, 0.0f) - 10.0f*light.direction;
  Ray ray(origin, light.direction, radiance_ray_type, scene_epsilon, RT_DEFAULT_MAX);
  rtTrace(top_object, ray, prd);

  if(prd.result.z > 0.0f) {
   uint2 new_idx = make_uint2(make_float2(launch_dim)*(0.5f + make_float2(prd.result.x, prd.result.y)*0.5f));
   output_buffer[new_idx] += make_float4(0.25f/(-light.direction.z)*dot(prd.result, prd.hit_normal));
  }
}
```



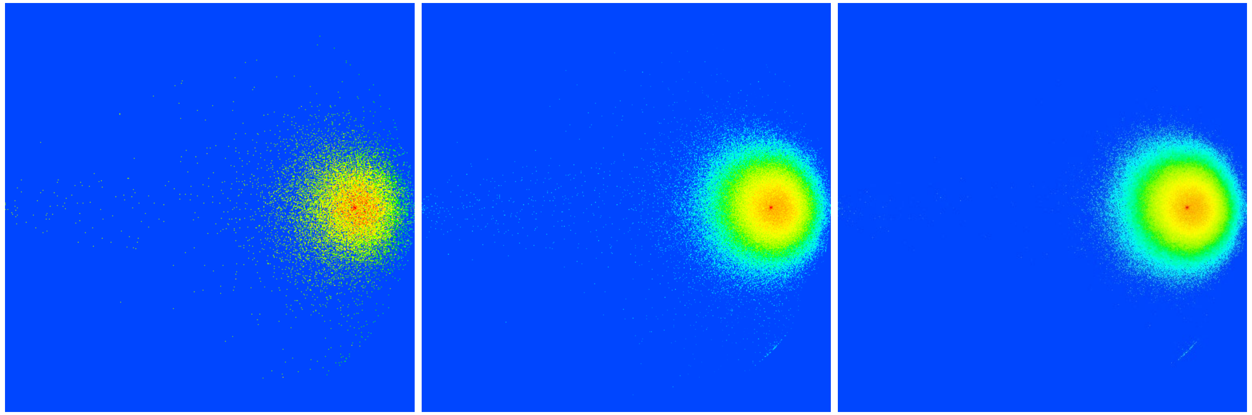
Specular material shader (closest hit program)

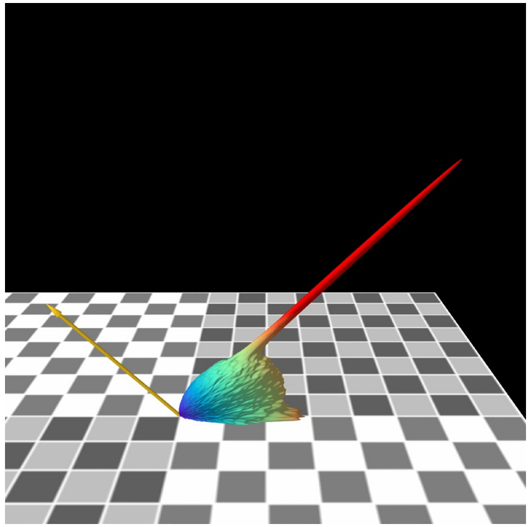
```
RT_PROGRAM void transparent_shader()
{
   if(prd.depth++ > max_depth) return;
   float3 hit_pos = ray.origin + t_hit*ray.direction;
   float3 normal = normalize(rtTransformNormal(RT_OBJECT_TO_WORLD, shading_normal));
   float n1_over_n2 = 1.0f/ior;
   float cos_theta_i = dot(-ray.direction, normal);
   if(cos_theta_i < 0.0f) { n1_over_n2 = ior; normal = -normal; cos_theta_i = -cos_theta_i; }
   prd.hit_normal = normal;

   float R = 1.0f;  // Compute Fresnel reflectance
   float sin_theta_t_sqr = n1_over_n2*n1_over_n2*(1.0f - cos_theta_i *cos_theta_i);
   float cos_theta_t = 0;
   if(sin_theta_t_sqr < 1.0f) {
      cos_theta_t = sqrtf(1.0f - sin_theta_t_sqr);
      R = fresnel_R(cos_theta_i, cos_theta_t, n1_over_n2);
   }
   float3 dir;       // Russian roulette to choose reflection or refraction
   if(rnd(prd.seed) < R) dir = reflect(ray.direction, normal);
   else dir = n1_over_n2*ray.direction + normal*(n1_over_n2*cos_theta_i - cos_theta_t);
   Ray new_ray(hit_pos, dir, radiance_ray_type, scene_epsilon, RT_DEFAULT_MAX);
   rtTrace(top_object, new_ray, prd);
}
```



Progressive updates (second camera program)

```
rtBuffer<float4, 2> output_buffer;
rtBuffer<float4, 2> p_output_buffer;
rtDeclareVariable(uint2, launch_index, rtLaunchIndex, );
rtDeclareVariable(uint, frame, , );

RT_PROGRAM void progression_camera()
{
   float4 c = output_buffer[launch_index];
   float4 curr_sum = (frame != 0) ? p_output_buffer[launch_index] * ((float)frame) : make_float4(0.0f);
   p_output_buffer[launch_index] = (c + curr_sum) / ((float)(frame + 1));
   output_buffer[launch_index] = make_float4(0.0f);
}
```





Agenda

- Introduction
- Tools and measurements
- Multiscale modeling
- Computing the BRDF using Monte Carlo simulation
- Procedural modeling of surface microgeometry
- Applying microgeometry to surfaces in 3D printing

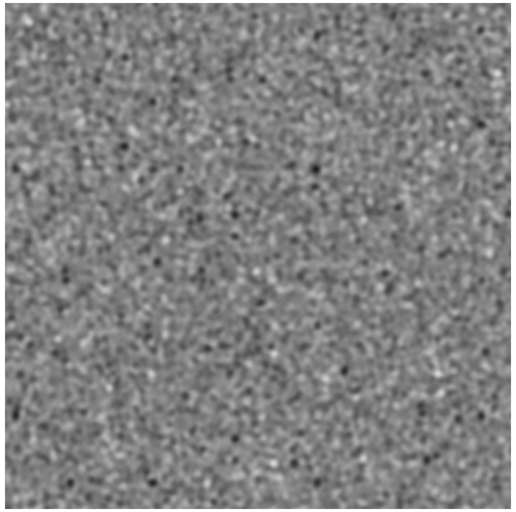
Noise

- Noise explorer:
  https://people.compute.dtu.dk/jerf/code/noise/

- Sparse convolution noise
  - Convolution of randomly placed ($\boldsymbol{x}_{i,j}$) impulses of random value ($\alpha_{i,j}$).
  - Use a filter kernel with compact support and insert a regular grid (cell vertices $\boldsymbol{q}_i$).

$$\text{cubic}(\boldsymbol{v}) = \begin{cases} (1 - 4\,\boldsymbol{v}\cdot\boldsymbol{v})^3 & \text{for } v \cdot v < \frac{1}{4} \\ 0 & \text{otherwise} \end{cases}$$

  - Use a seeded RNG: $\text{rnd}(t)$
  - Choose a number of impulses per cell (N).



$$0.5\,(A\,\text{noise}(B\,\boldsymbol{p}) + 1)$$
$$A = 1, B = 1$$

$$\text{noise}(\boldsymbol{p}) = \frac{4}{5\sqrt[3]{N}} \sum_{i=0}^{7} \sum_{j=1}^{N} \alpha_{i,j}\,\text{cubic}(\boldsymbol{x}_{i,j} - \boldsymbol{p})$$

$$\boldsymbol{x}_{i,j} = \boldsymbol{q}_i + \boldsymbol{\xi}_{i,j}$$

$$\alpha_{i,j} = \text{rnd}\left(t_{n_{i,j}}\right)(1 - 2(j \bmod 2))$$

$$\boldsymbol{\xi}_{i,j} = \left(\text{rnd}\left(t_{n_{i,j}+1}\right), \text{rnd}\left(t_{n_{i,j}+2}\right), \text{rnd}\left(t_{n_{i,j}+3}\right)\right)$$

$$n_{i,j} = 4(N\boldsymbol{q}_i \cdot \boldsymbol{a} + j)$$

$$\boldsymbol{q}_i = \left\lfloor \boldsymbol{p} - \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right) \right\rfloor + \left(i \bmod 2, \left\lfloor \frac{i}{2} \right\rfloor \bmod 2, \left\lfloor \frac{i}{4} \right\rfloor \bmod 2\right)$$

My choice: $N = 30$ and $\boldsymbol{a} = (1, 1000, 576)$

Noise-based modeling

- Noise with octaves
  - Number of octaves $\Omega \geq 1$
  - Lacunarity $\ell > 1$
  - Fractional increment (roughness) $H \in (0,1]$

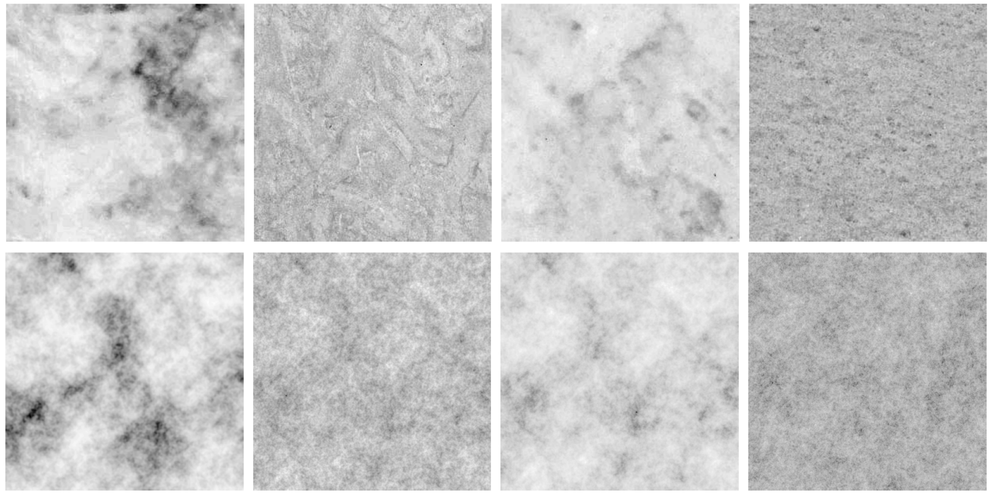
$$\mathrm{fBm}(\boldsymbol{p}) = \sum_{i=0}^{\Omega-1} \ell^{-Hi} \mathrm{noise}(\boldsymbol{p}\,\ell^{i})$$

  - $H = 1$ is a monofractal
    (same fractal dimension everywhere)

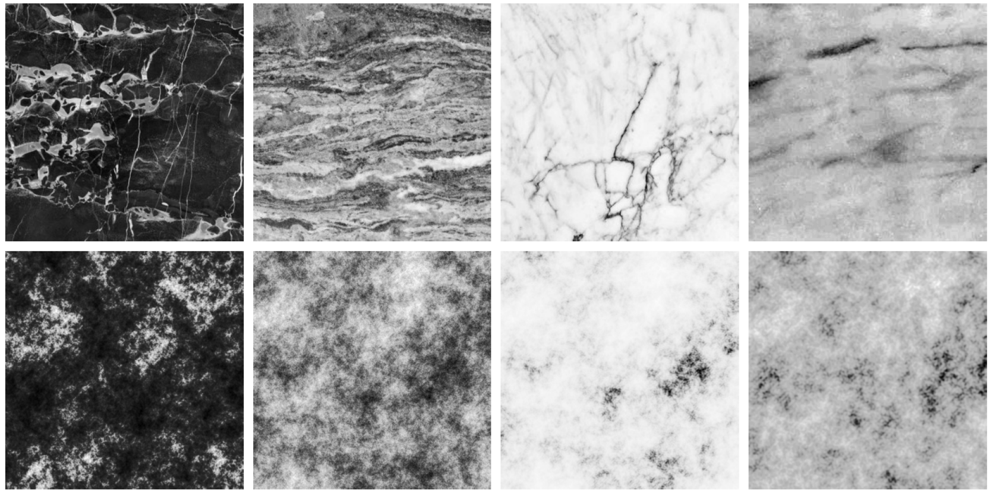
- Absolute value for sharp edges

$$\mathrm{turbulence}(\boldsymbol{p}) = \sum_{i=\Omega_{\mathrm{low}}}^{\Omega_{\mathrm{high}}} \frac{1}{2^{i}} \left|\mathrm{noise}(2^{i}\,\boldsymbol{p})\right|$$

Top rows: input from marble images



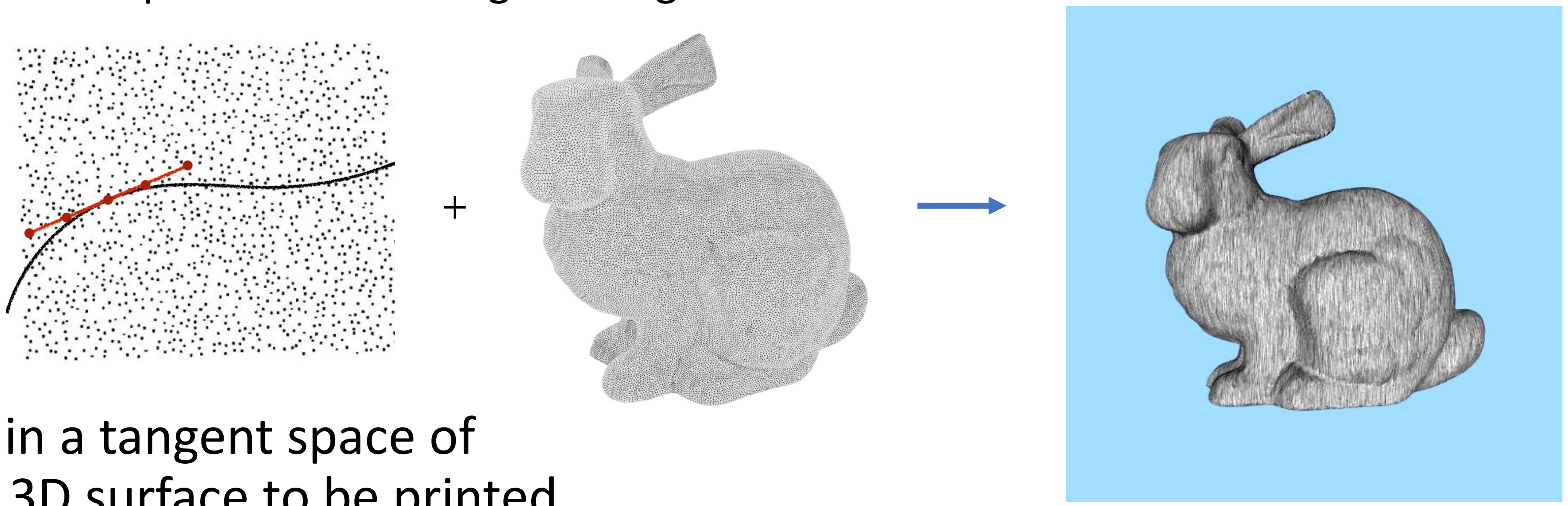
Kim Harder Fog. Noise-based texture synthesis by analysis of image examples. MSc thesis, Technical University of Denmark, 2017.



Bottom rows: fBm fit

Line integral convolution

- Given a 3D noise function like sparse convolution noise:
  - For each pixel take the integral along a line.



- Obtain a tangent space of your 3D surface to be printed. https://people.compute.dtu.dk/jerf/code/hairy/

References:
- Battke, H., Stalling, D., Hege H.-C. Fast line integral convolution for arbitrary surfaces in 3D. In *Visualization and Mathematics*, pp. 181-195. Springer, 1997.
- Frisvad, J. R. Building an orthonormal basis from a 3d unit vector without normalization. *Journal of Graphics Tools* **16**(3):151-159, August 2012.

Agenda

- Introduction
- Tools and measurements
- Multiscale modeling
- Computing the BRDF using Monte Carlo simulation
- Procedural modeling of surface microgeometry
- Applying microgeometry to surfaces in 3D printing

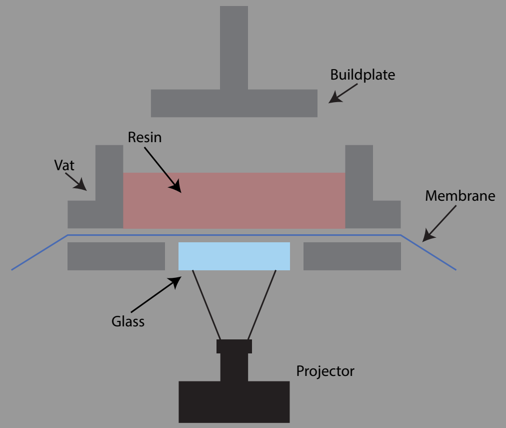
Types of 3D Printers

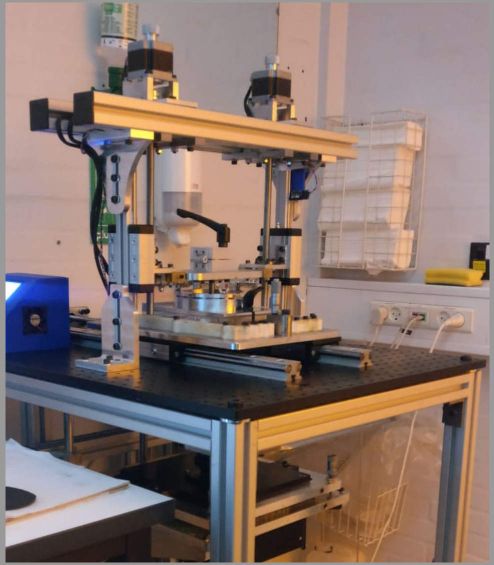
- Material Extrusion: fused deposition modelling (FDM), direct writing assembly (DWA), . . . ;
- Powder: binder jetting, selective laser sintering, . . . ;
- Lamination: laminated object manufacturing (LOM), selective deposition lamination (SDL), . . . ;
- Photopolymerization: stereolitography (SLA), digital light processing (DLP), . . . ;

Intro on 3D printing
What is a DLP?

Elements of a DLP printer:

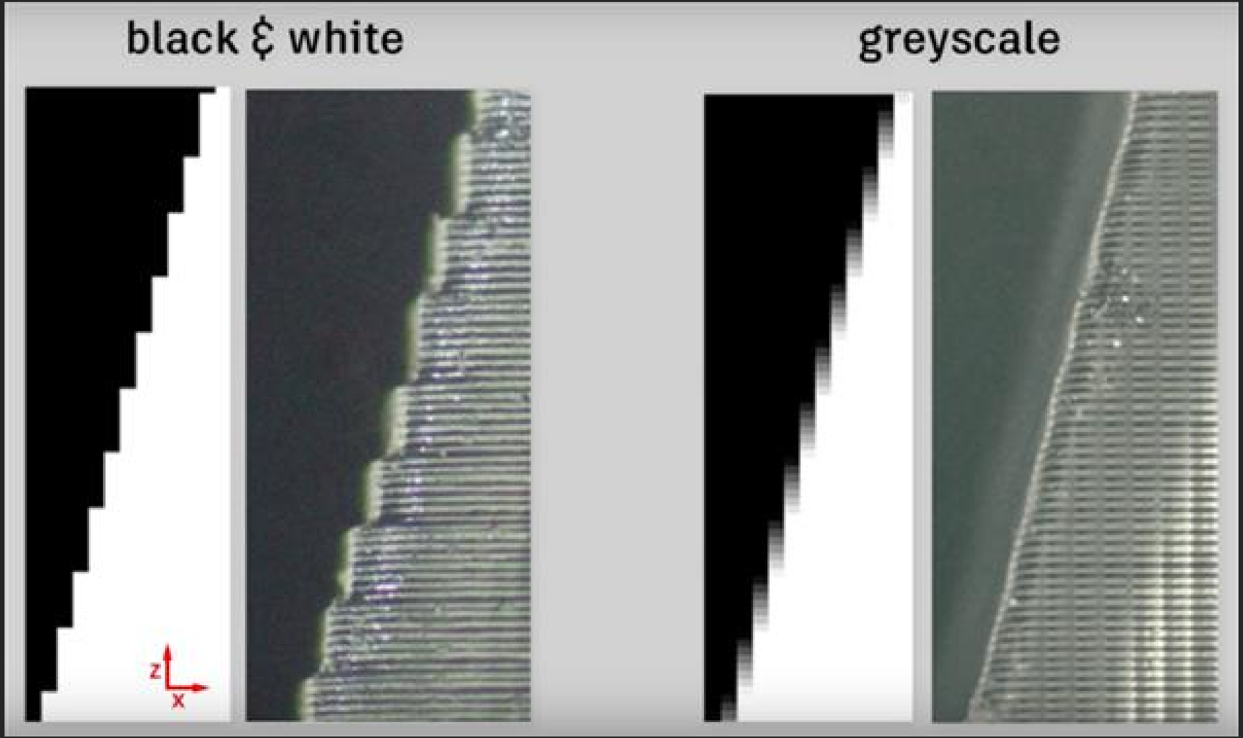
- Photopolymer;
- Vat;
- Building Platform (Step precision: 1μm);
- DLP Projector (2560×1600 micro-mirrors, pixel pitch of 7.54μm)
- Membrane (optional)





Effect of Grayscale (Autodesk [Greene 2016])

- subvoxel resolution;
- subvoxel offset;
- antialiasing;



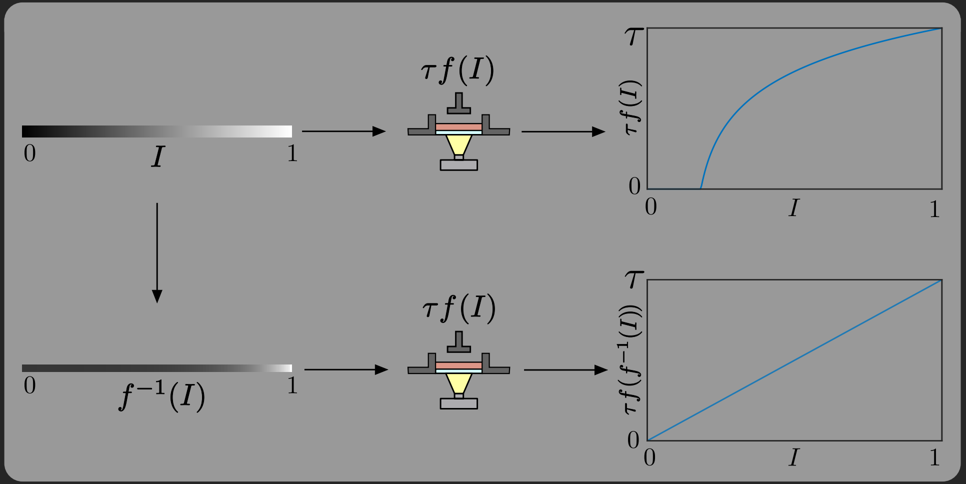
How to use grayscale
Control Voxel Growth

Relationship between curing process and uv intensity is not linear:

$$\tau f(I) = \begin{cases} \alpha + \beta \log(I - \gamma)\,, & \text{for } I > e^{-\alpha/\beta} + \gamma\,, \\ 0\,, & \text{for } I \leq e^{-\alpha/\beta} + \gamma\,, \end{cases}$$

If we know α, β, and γ we can correct before projection:

$$f^{-1}(I) = \begin{cases} e^{\frac{\tau\, I - \alpha}{\beta}} + \gamma\,, & \text{for } I > 0\,, \\ 0\,, & \text{for } I = 0\,, \end{cases}$$

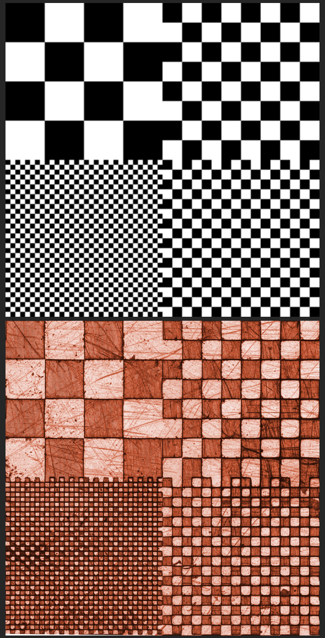


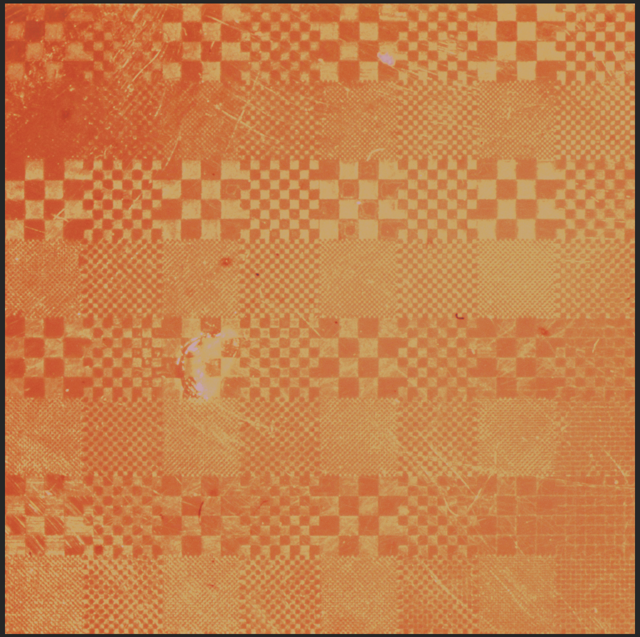
Experiments
Parameters Calibration

We need to calibrate the projector intensity and the exposure time:

- fix exposure time;
- print calibration sample with increasing intensity;
- fix intensity;
- print calibration sample with increasing exposure tim;

Layer thickness: 18μm;
UV LED amplitude: 230;
Exposure time: $3s$;





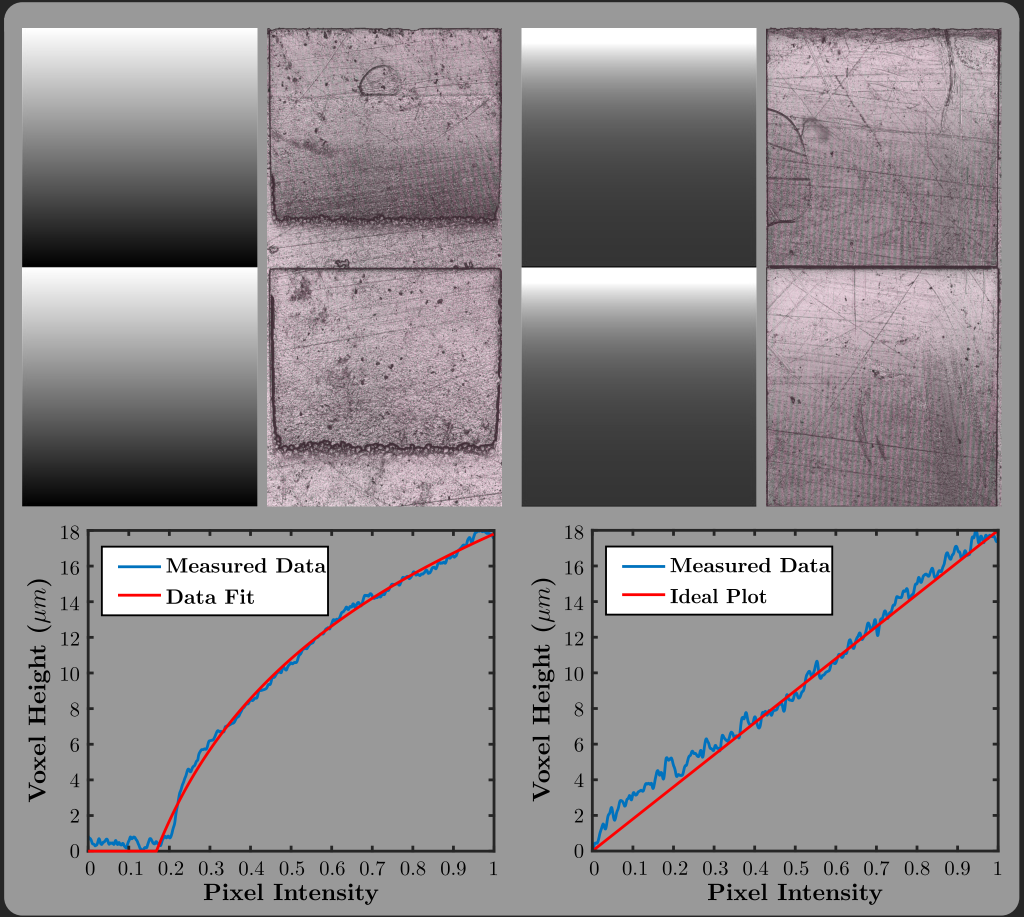
Experiments

Voxel Height Measurements

We need to estimate c, β, and γ to obtain linearity:

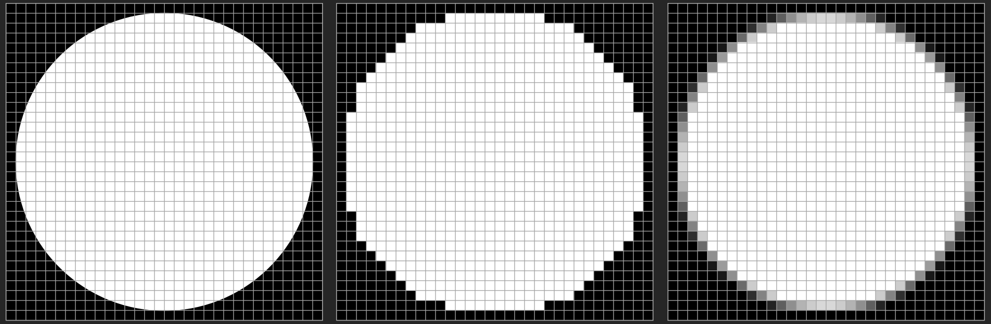
- print linear pattern with all gray values;
- measure the surface height with a resolution of 0.4μm;
- find a fit and estimate the parameters;

$\alpha = 17.71\ \mu$m
$\beta = 10.24\ \mu$m
$\gamma = -0.01$



How to use grayscale
Applying Grayscale

- Supersampling during slicing to apply AA (rasterization/ray tracing);



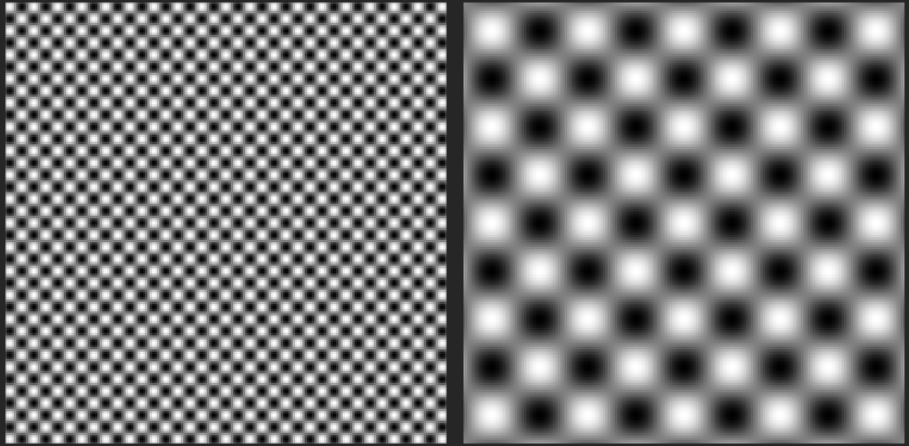
Applying Grayscale

- Supersampling during slicing to apply AA (rasterization/ray tracing);

Patterns to control the microstructure and roughness of the surface:

- 2D sinusoid

$$I(u,v) = \frac{1}{2}\sin\left(\frac{2\pi}{\lambda_u}u\right)\sin\left(\frac{2\pi}{\lambda_v}v\right) + \frac{1}{2};$$



How to use grayscale
Applying Grayscale

- Supersampling during slicing to apply AA (rasterization/ray tracing);

Patterns to control the microstructure and roughness of the surface:

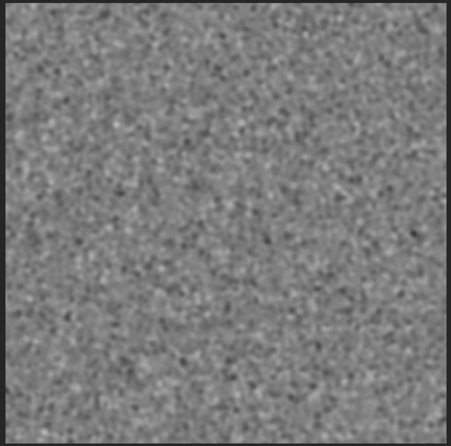
- 2D sinusoid

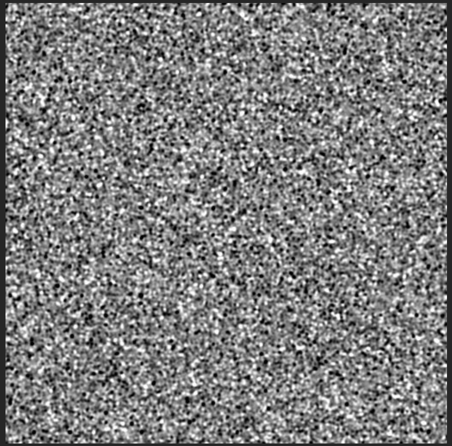
$$I(u,v) = \frac{1}{2}\sin\left(\frac{2\pi}{\lambda_u}u\right)\sin\left(\frac{2\pi}{\lambda_v}v\right) + \frac{1}{2};$$

- Solid sparse convolution noise

$$\mathrm{noise}(\mathbf{p}) = \frac{4}{5\sqrt[3]{N}}\sum_{i=0}^{7}\sum_{j=1}^{N}\alpha_{i,j}\,\mathrm{cubic}(\mathbf{x}_{i,j}-\mathbf{p})$$

$$I(\mathbf{p}) = \frac{A}{2}\,\mathrm{noise}(B\,\mathbf{p}) + 1/2$$





How to use grayscale
Applying Grayscale

- Supersampling during slicing to apply AA (rasterization/ray tracing);

Patterns to control the microstructure and roughness of the surface:

- 2D sinusoid

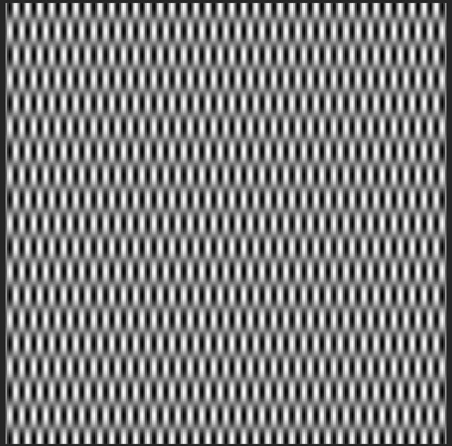
$$I(u,v) = \frac{1}{2}\sin\left(\frac{2\pi}{\lambda_u}u\right)\sin\left(\frac{2\pi}{\lambda_v}v\right) + \frac{1}{2};$$

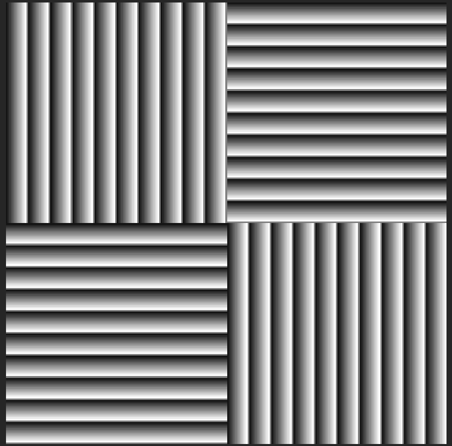
- Solid sparse convolution noise

$$\mathrm{noise}(\mathbf{p}) = \frac{4}{5\sqrt[3]{N}}\sum_{i=0}^{7}\sum_{j=1}^{N}\alpha_{i,j}\,\mathrm{cubic}(\mathbf{x}_{i,j}-\mathbf{p})$$

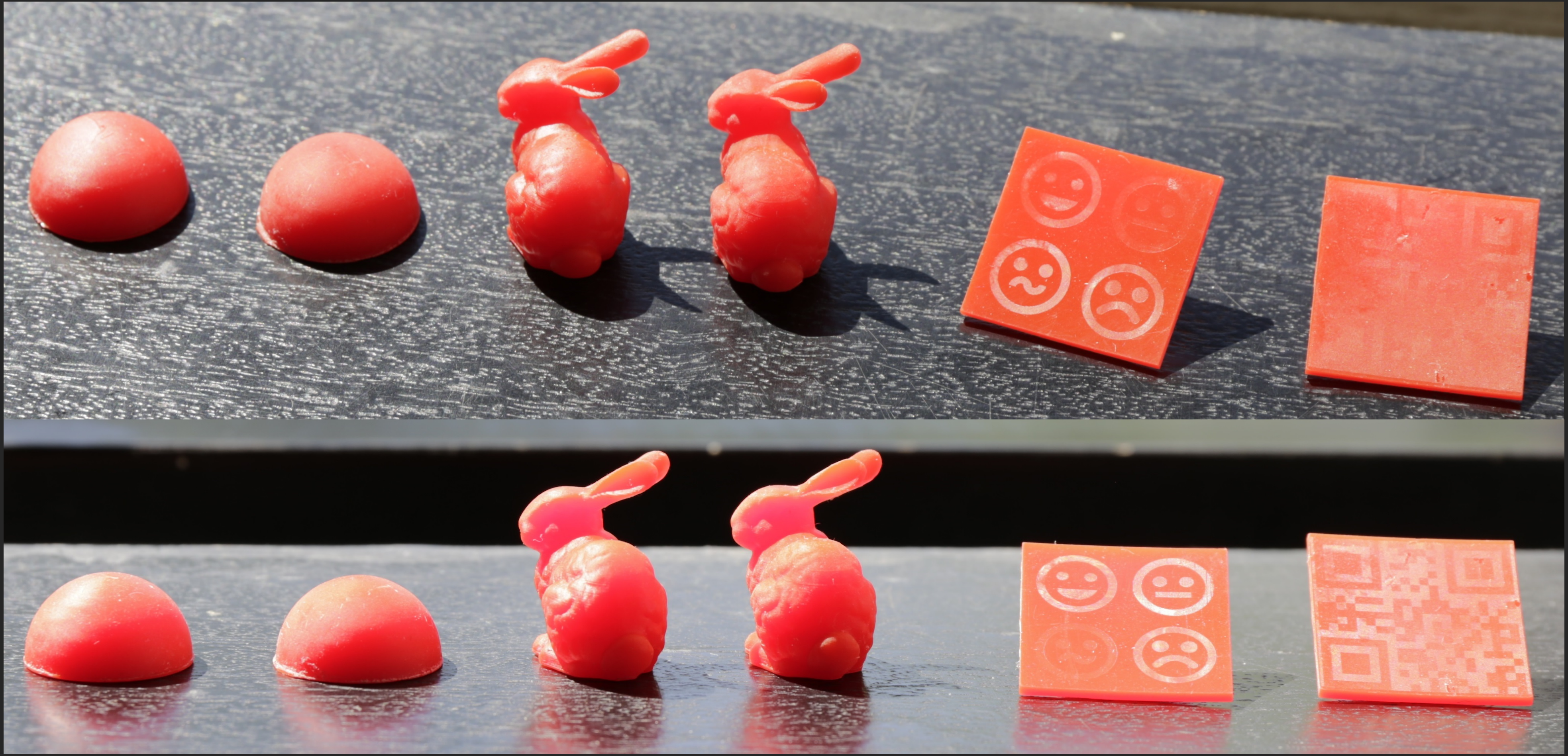
$$I(\mathbf{p}) = \frac{A}{2}\,\mathrm{noise}(B\,\mathbf{p}) + 1/2$$

- Ridged structure and 2D sinusoid to control anisotropy;

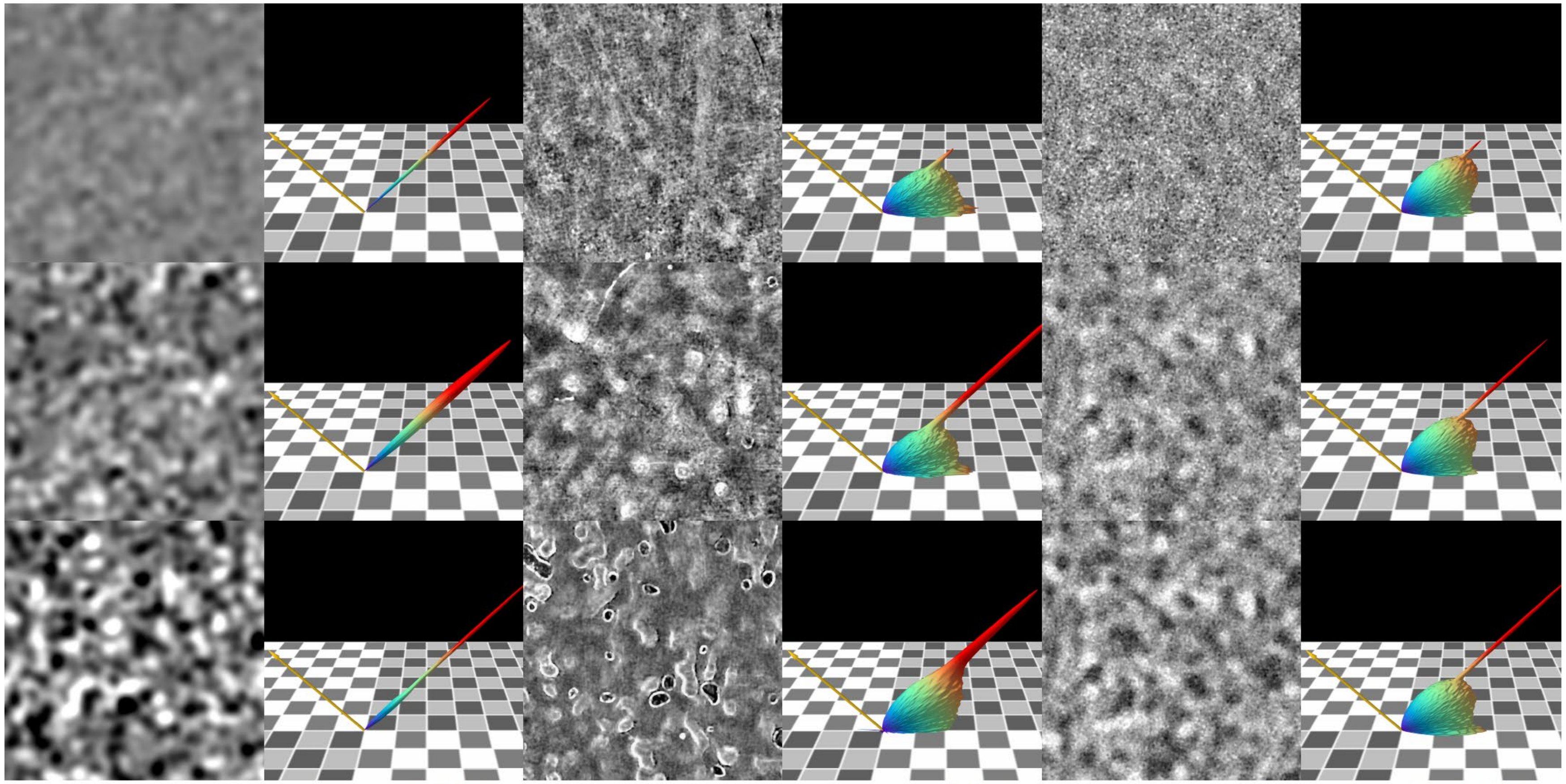




Results V

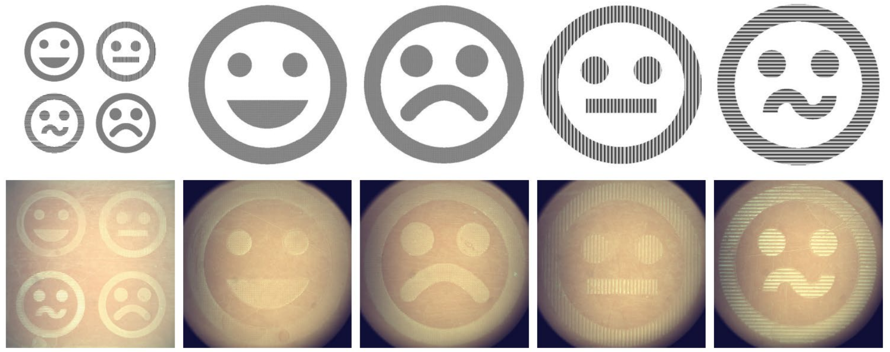


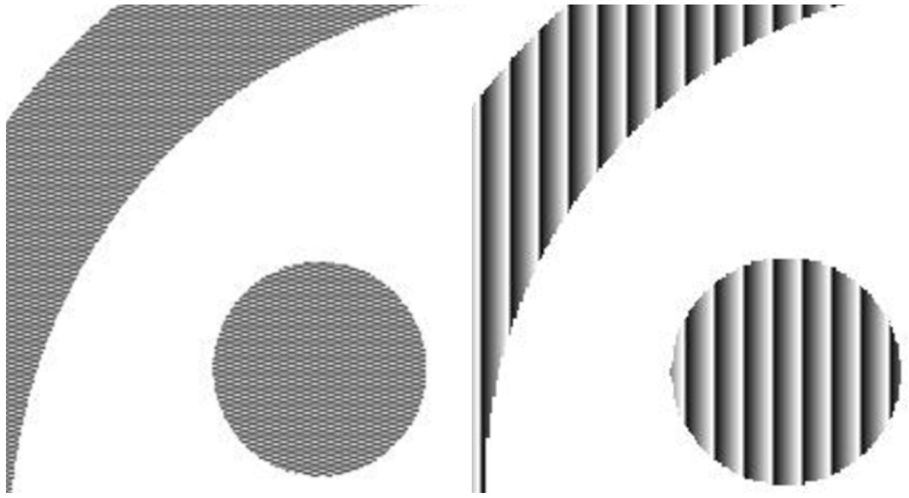
BRDF printing and 3D printer ground noise


(a) input (b) input BRDF (c) measured (d) printed BRDF (e) input + ground (f) model BRDF

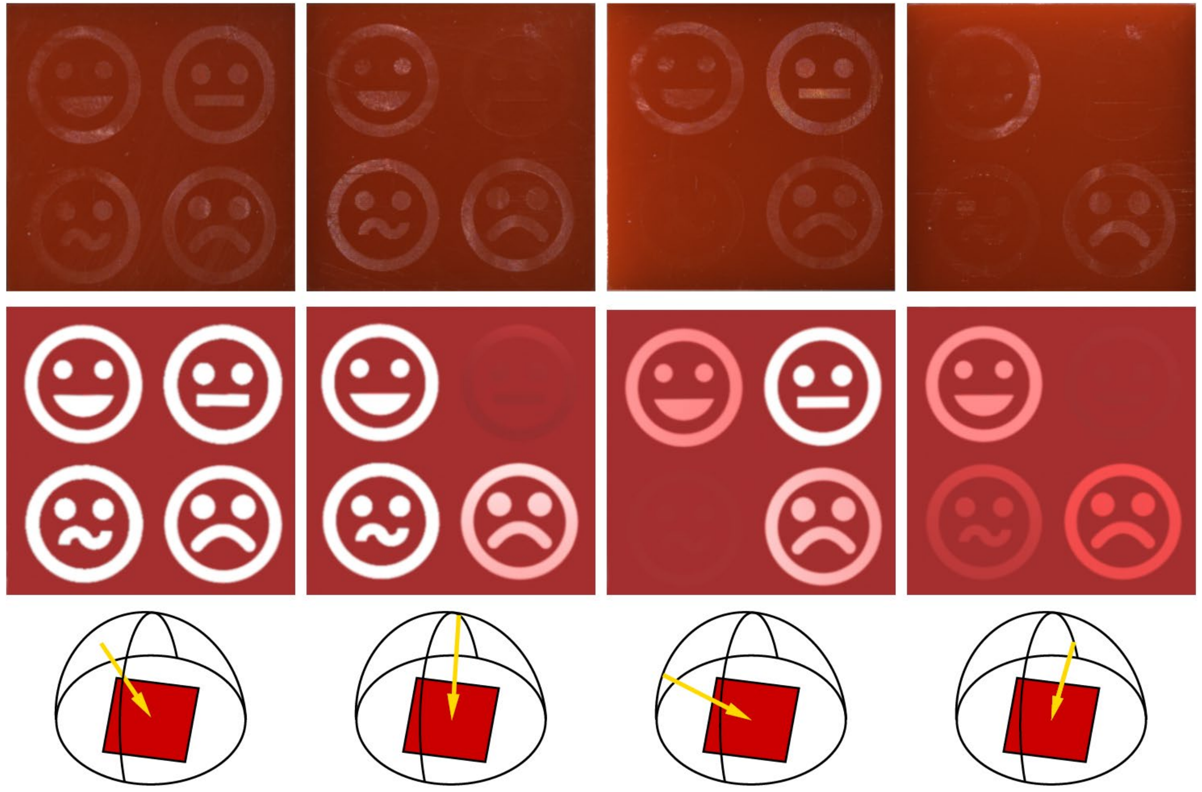
$$\mathrm{ground}(\boldsymbol{p}) = \frac{2}{3}\,\mathrm{noise}\left(\frac{\boldsymbol{p}}{50\ \mu m}\right) + \frac{1}{9}\,\mathrm{noise}\left(\frac{\boldsymbol{p}}{25\ \mu m}\right) + \frac{1}{12}\,\mathrm{noise}\left(\frac{\boldsymbol{p}}{2\ \mu m}\right)$$

Anisotropic smileys (ridges and sinusoids)





Input texture and microscope images.



Top row:	3D printed
Middle row:	computed
Bottom row:	direction of incidence

Thank you for your attention

The slides on dark gray background are courtesy of Andrea Luongo.
Thanks to all co-authors of the work mentioned below.

A. Luongo, V. Falster, M. B. Doest, M. M. Ribo, E. R. Eiriksson, D. B. Pedersen, and J. R. Frisvad. Microstructure control in 3D printing with digital light processing. *Computer Graphics Forum*, 2019. To appear.