Lecture 8: Texture and mapping

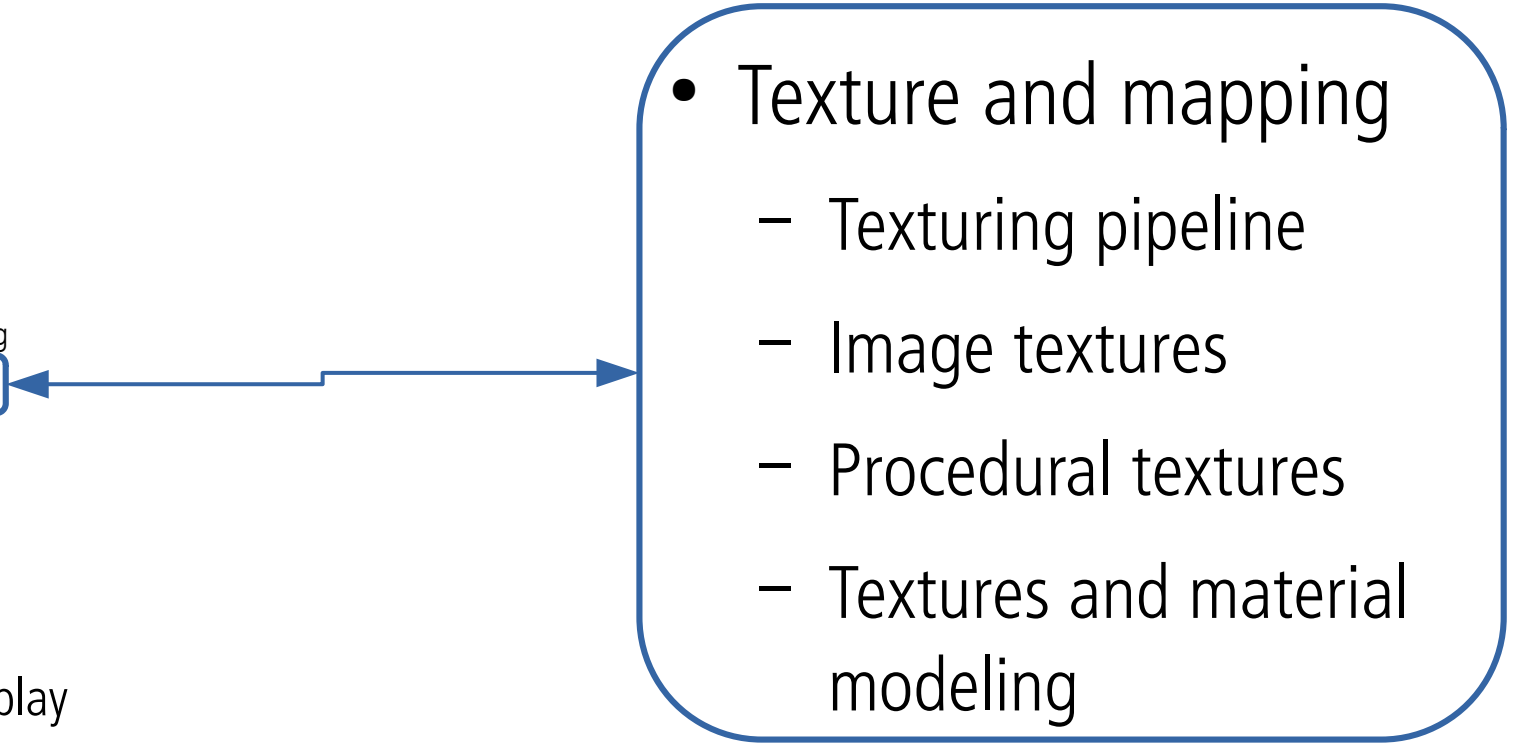
DHBW, Computer Graphics

Lovro Bosnar

15.2.2023.

Syllabus

- 3D scene
  - Object
    - Shape
    - Material
      - Scattering
      - **Texture**
  - Camera
  - Light
- Rendering
- Image and display

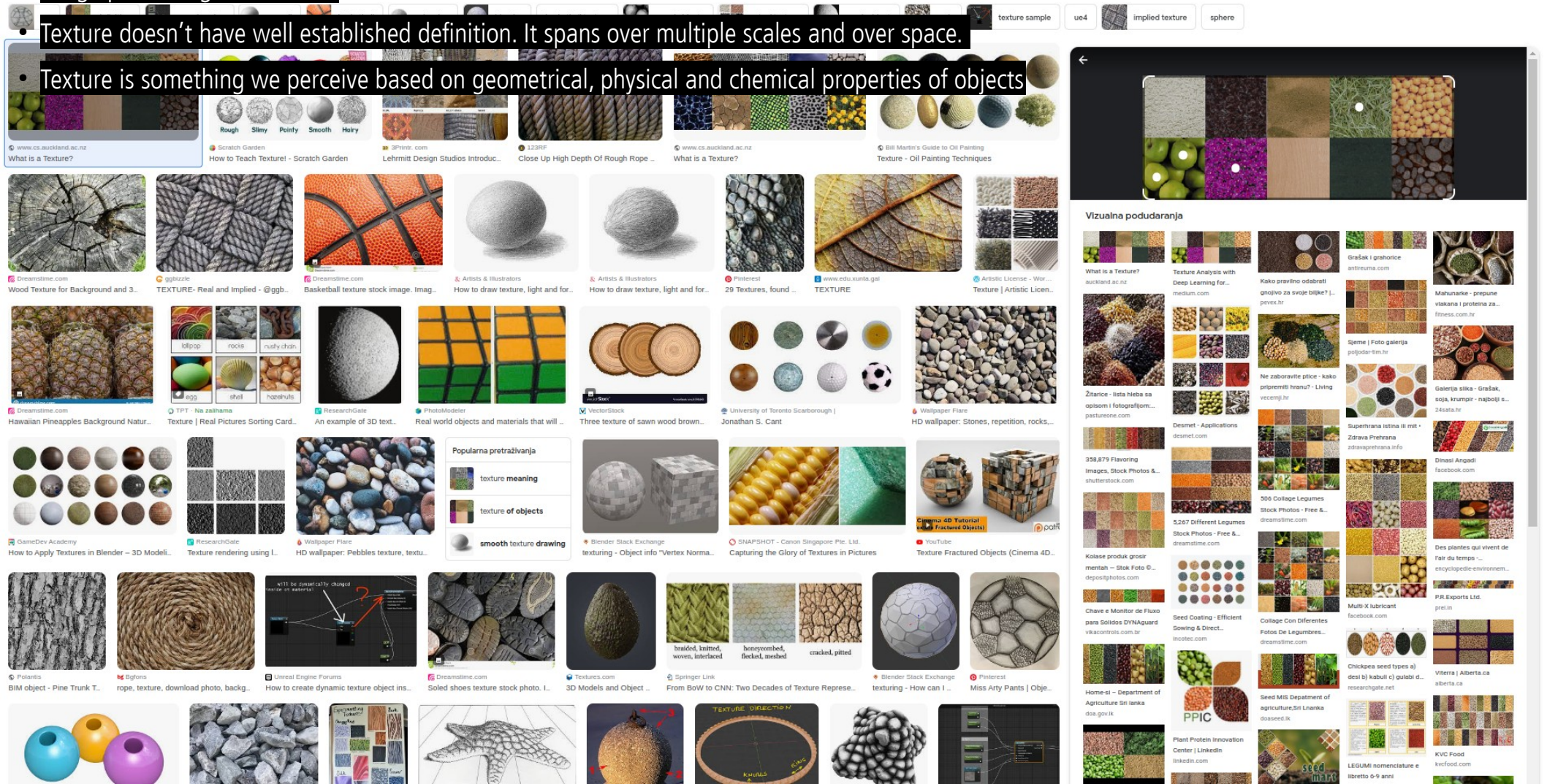



Introduction
SUBSTANCE
DESIGNER

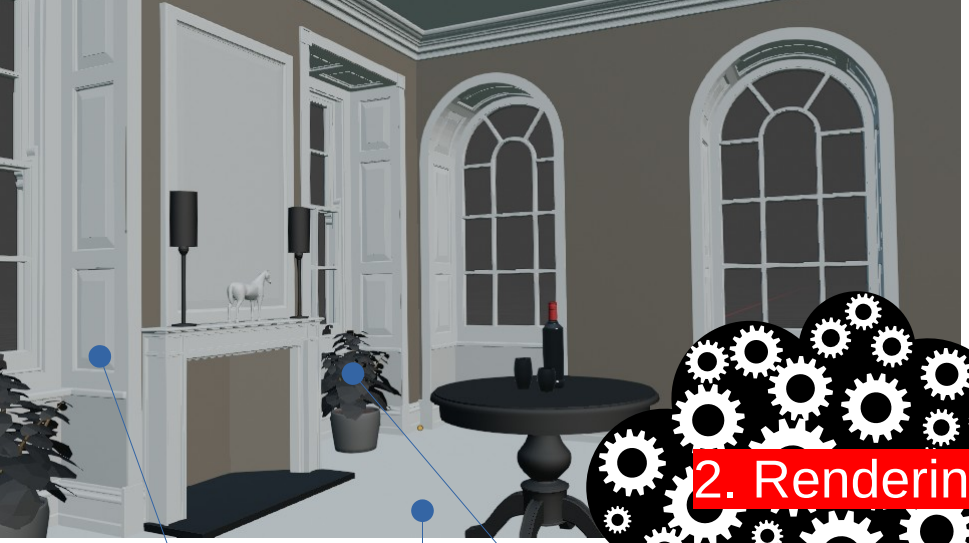
- Metal
- Plastic
- Glass
- Wood
- Fabric
- Stone
- Clouds
- Water
- Tree bark
- Leafs
- Plaster
- Paper
- Leather
- Sky
- Etc.

https://www.xomacenna.com/blog/a-guide-to-organic-modern-design

- Term **texture** is used differently in various disciplines and everyday life: texture of fabric, texture of food, texture in music, texture in image processing, texture in...
- Texture doesn't have well established definition. It spans over multiple scales and over space.
- Texture is something we perceive based on geometrical, physical and chemical properties of objects



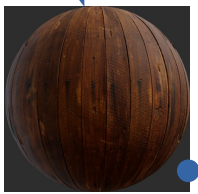
Big picture



1. Scene modeling

3. Image

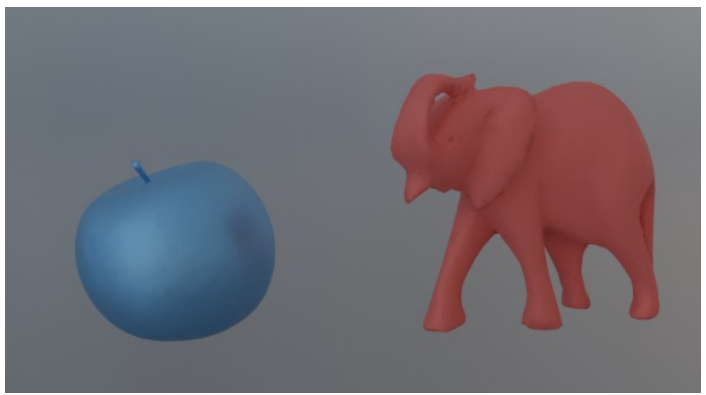
Texture defines look and feel of objects



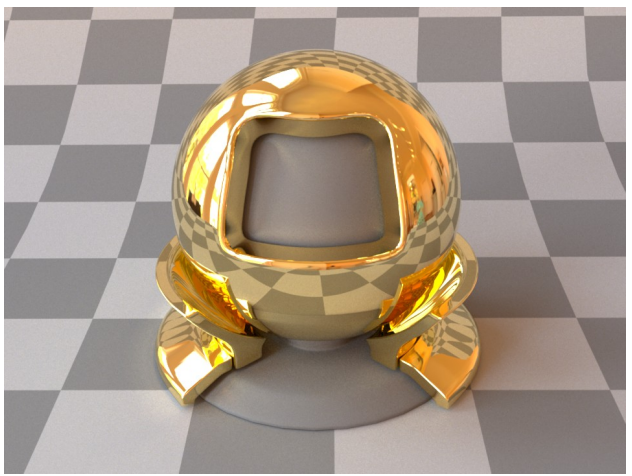
Material characteristics

- **Characteristics that needed to be modeled** in order to obtain required appearance:

Color



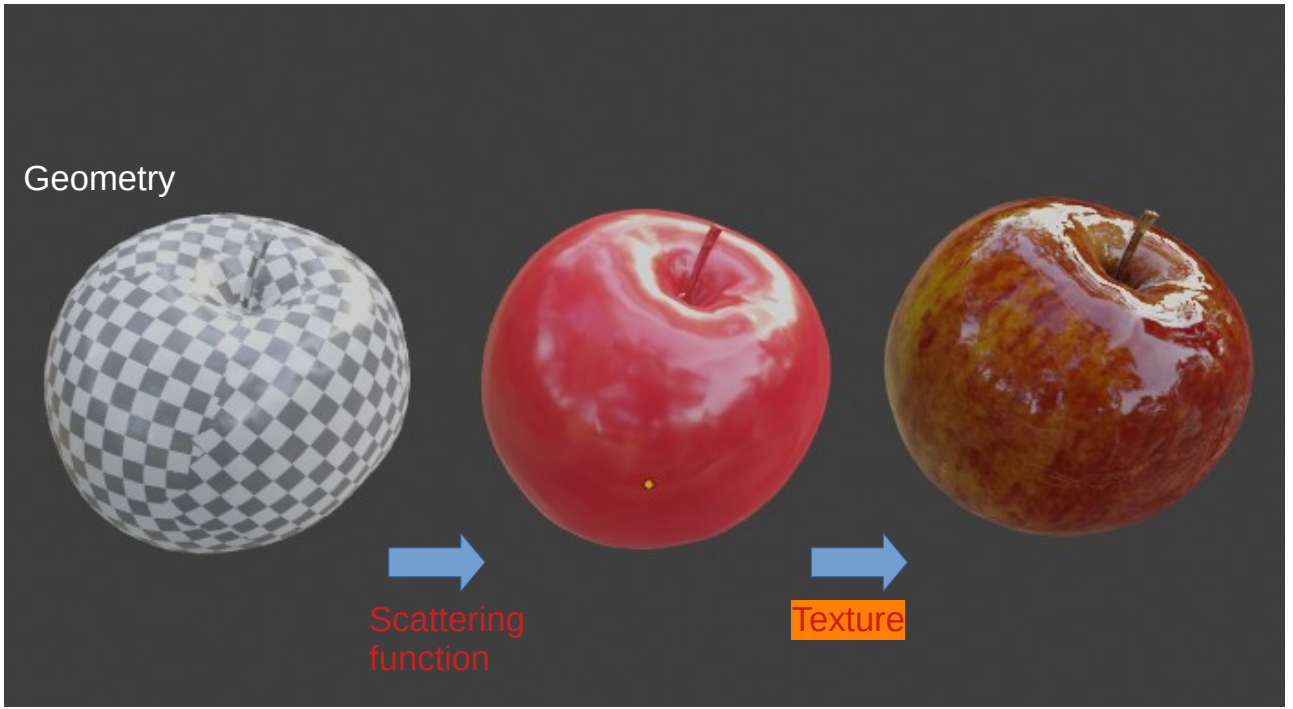
Directional (e.g., reflectivity)



Texture and patterns

Texture in computer graphics

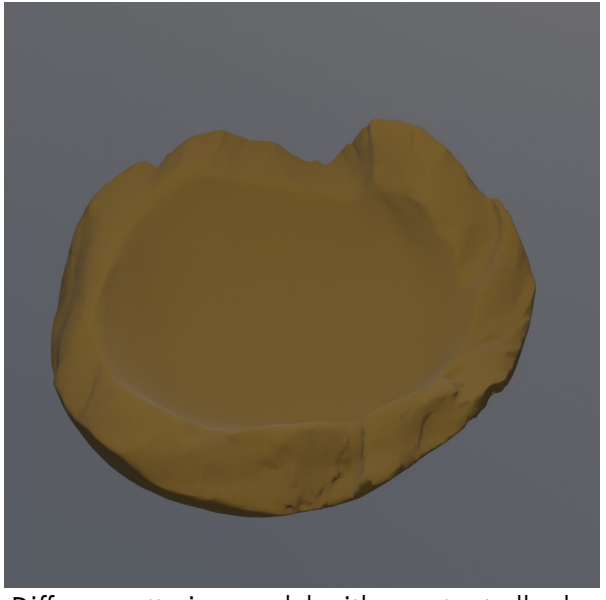
- For 3D modeling, we can try to define texture as function which **varies scattering function properties over surface**.



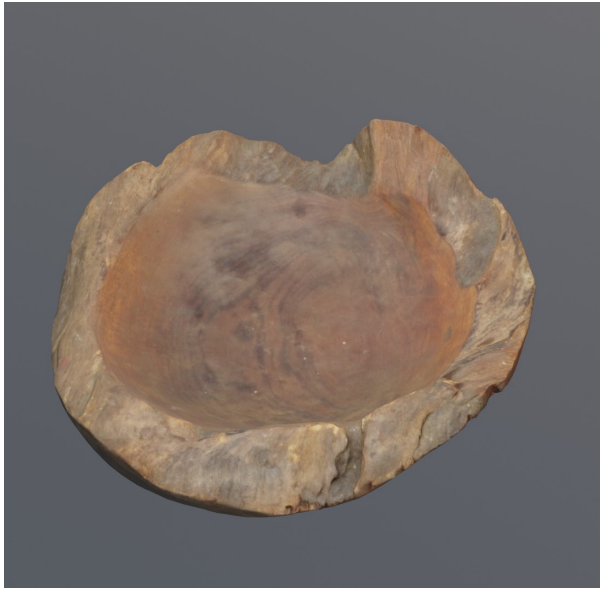
Variation of material over surface

- Real objects rarely have only one material or same material properties over the surface → **homogeneous material**
  - Example: wood has structure at a scale of about 1mm and lower. The material of wood fiber also varies. Therefore, the rich appearance of wood comes from structure and material variation of fibers. This richness is called **texture**.

Homogeneous materials are missing details and variations, thus look too smooth and too perfect.



Diffuse scattering model with constant albedo parameter → homogeneous material.



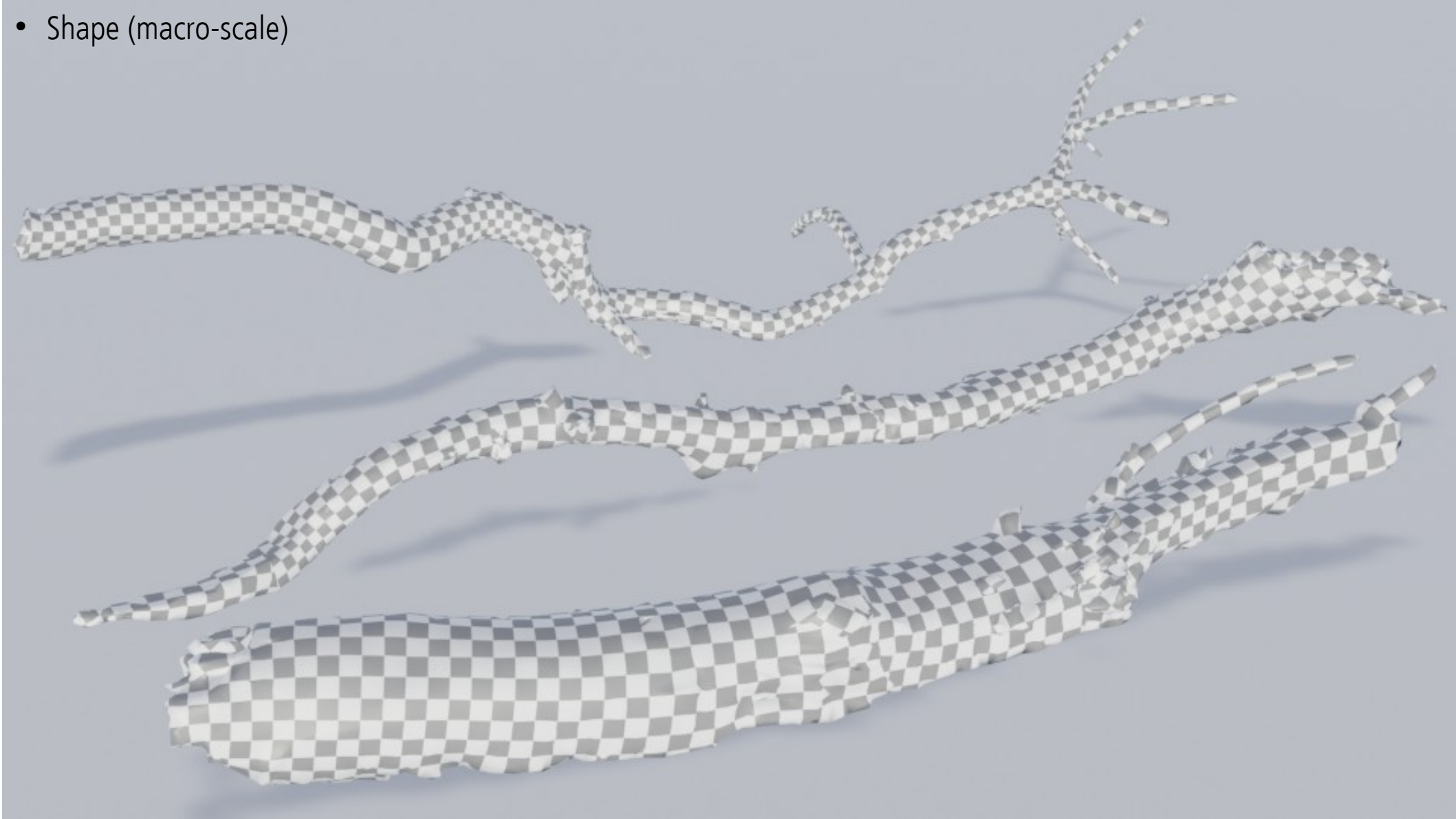
Diffuse scattering model with variation of albedo parameter using texture.

Texture

- Visual variation on the object surface – **spatial variation → details and richness of surface**
  - **Directional or color characteristics → scattering function parameters**
  - **Small-scale geometrical characteristics**: bumps, pores, imperfections, etc. **→ surface normals**
- In computer graphics, **texturing** is process that takes surface and modifies its appearance at each location using image, function or other data source.
  - Its function is to add details to object surface while keeping computational cost low

* Remember that computer graphics is all about simulation and approximation in order to achieve required appearance. It is a trade-off between quality and speed. Depending on viewing distance and size of objects in 3D scene, some details can be represented with lower quality. For example, modeling all details on geometrical (shape) level can be very expensive. Thus, texturing come in handy. Texture represents cheaper way to add details on object surface.

- Shape (macro-scale)



- Shape (macro-scale)
- Scattering function: diffuse (microscale)
  - Albedo parameter (color) → constant

- Shape (macro-scale)
- Scattering function: diffuse (microscale)
  - Albedo parameter (color) → texture variation

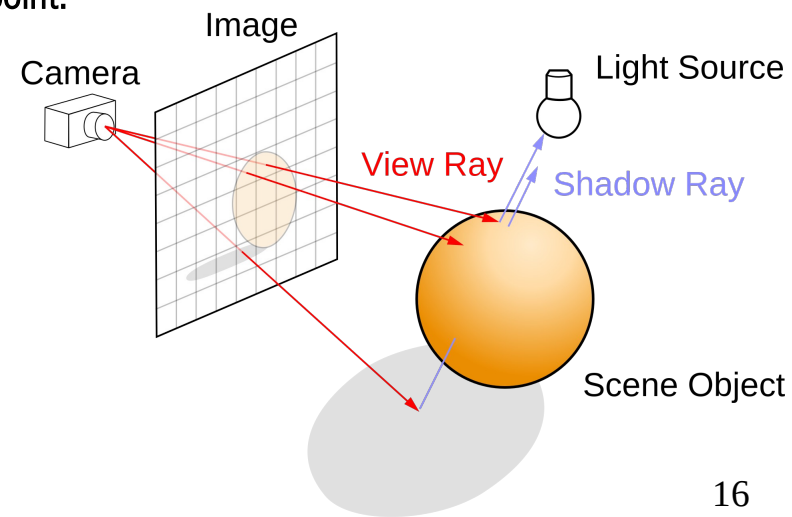
- Shape (macro-scale)
- Scattering function: diffuse (microscale)
  - Albedo parameter (color) → texture variation
- Small-scale geometry: surface normals (mesoscale)
  - Normal perturbation → texture variation

Texturing: variation

- **Texturing: adding surface details which are too expensive to model explicitly with geometry**
- Textures introduce
  - **Scattering function parameters (e.g., color) variation**
  - **Small-scale geometry variation** → Shading normal variation
- Texturing scale
  - **Microscale** variation **➞ scattering function parameters**
  - **Mesoscale** variation **➞ shading normal perturbation**
  - **Macroscale** is given by the object shape

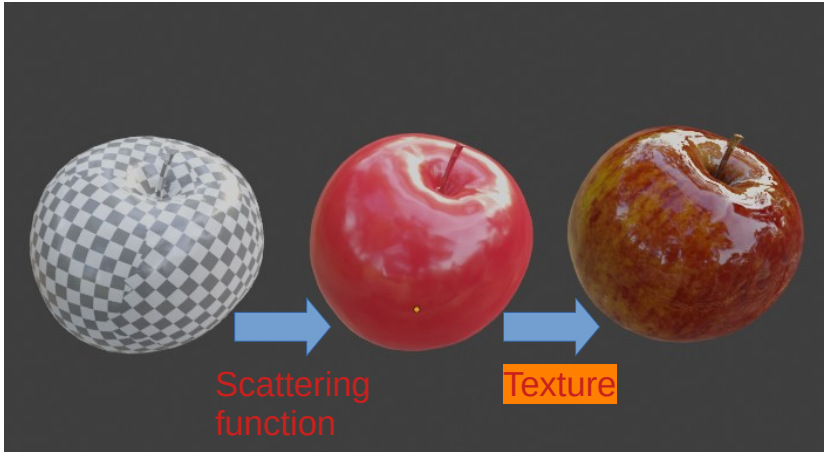
Texture and rendering

- Rendering: computing color of pixels in the virtual image plane
- Color of pixels → color of objects visible from camera
  - For each pixel, viewing ray is generated, traced into scene for closest intersection → **visibility solving**
  - Color is calculated at closest intersection → **shading**
  - The result of shading is color of pixel from which ray was generated
- **Shading**: calculating color at viewing ray intersection with object → **shading point.**



Texture and shading

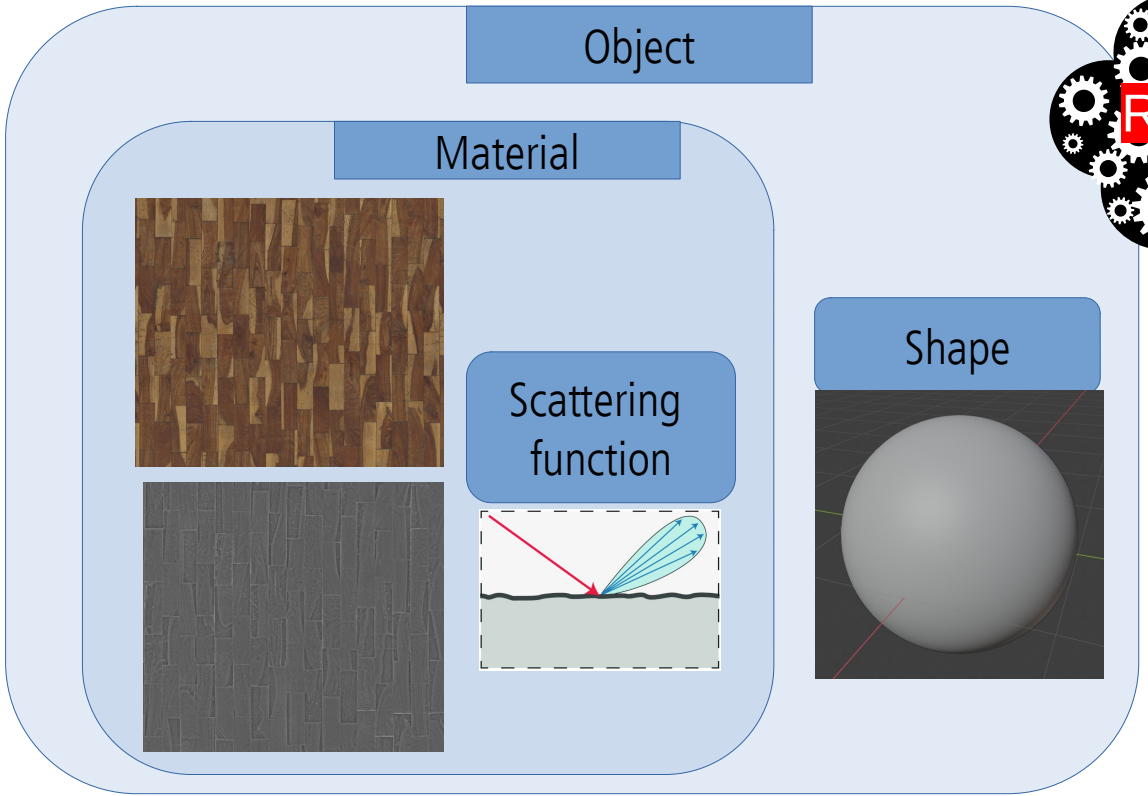
- Shading uses **material**, **shape**, **viewing** and **light** to calculate the color in shading point
- **Material** describes interaction of light with surface:
  - **Scattering model**
  - **Texture model**
- For each shading point, **scattering model is evaluated using texture information** at that point
- Final rendered image, due to variation of colors and intensities over object, the surface will contain pattern
  - Surface with pattern is called textured surface

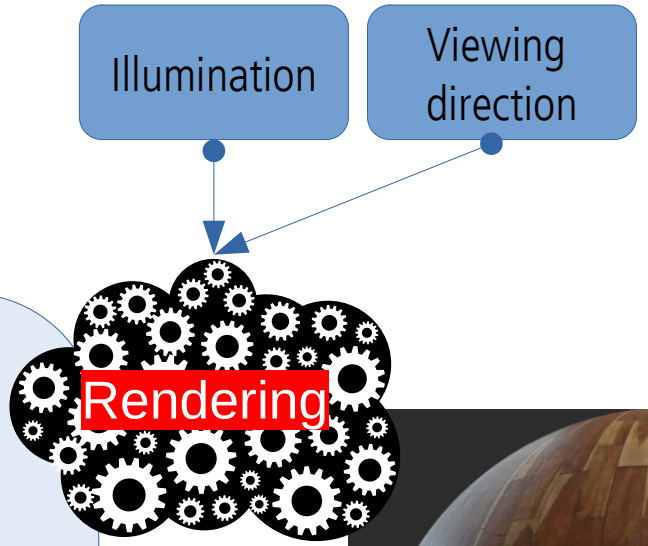


Note on "texture"

Texture in modeling vs texture on rendered surface

- Texture defines shading parameters over object surface → something that we model
- Resulting pattern of colors on rendered surface → generated texture

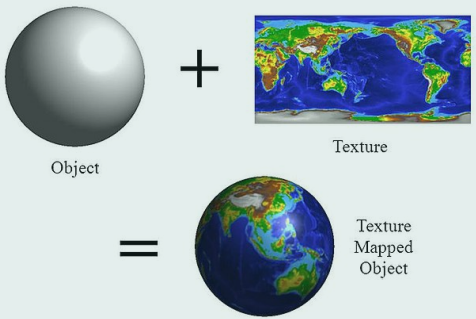




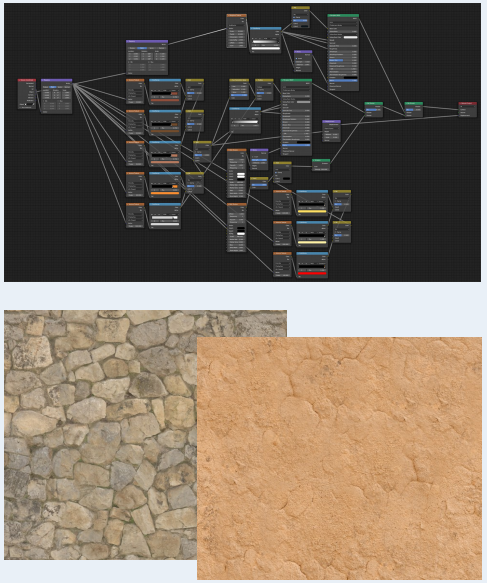
Note on “texture”

- Texture is generally and in computer graphics overloaded term. Meaning depends on the context:
  - Texture can be used for modeling object appearance (material)
  - On GPU rendering, texture can be used as memory buffer to send arbitrary data (e.g., geometry) from CPU to GPU

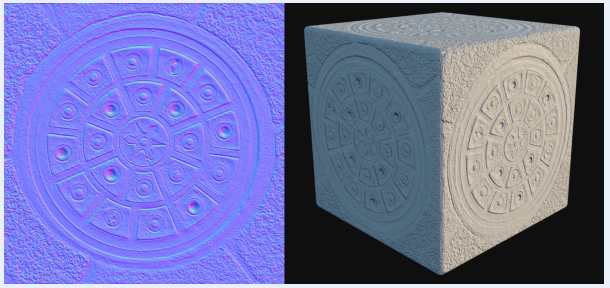
Learning objectives



How to apply textures on objects → **texturing pipeline**



Texturing approaches → **image and procedural**

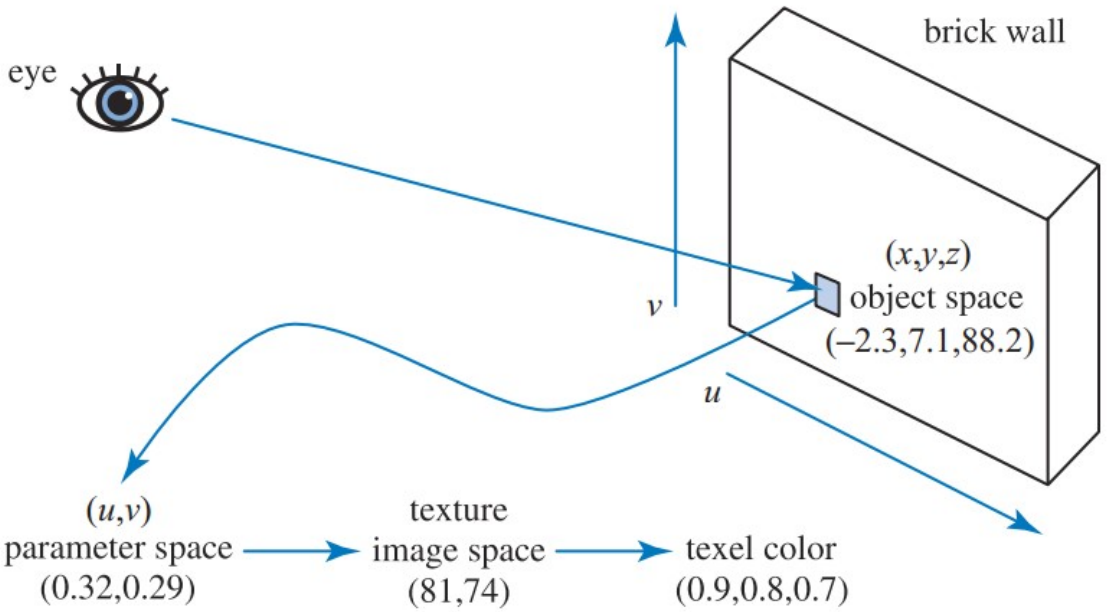


How to use texture for **material modeling** → variation of scattering function parameters and small scale geometry

Texturing pipeline

Texturing pipeline: example

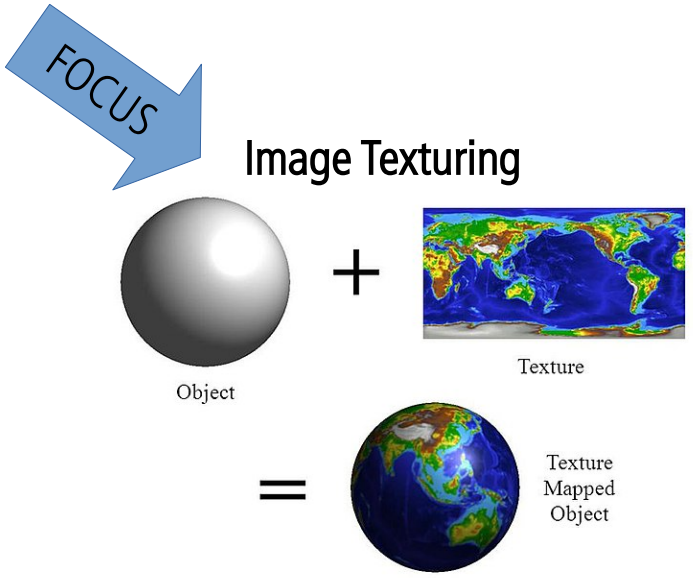
- Brick wall
  - **Camera ray → 3D object position** `(x,y,z)`
  - **Projector function:** mapping from 3D `(x,y,z)` to 2D `(u,v)` → **texture coordinates**
  - **Corresponder function:** continuous texture coordinates `(u,v)` → discrete texel location `(s,t)` in the texture image.
  - **Texture reading:** texel values (e.g., RGB color) are obtained using `(s,t)`
  - Optional **transformation on values**
  - Obtained value is used as input to scattering function which is evaluated for shading



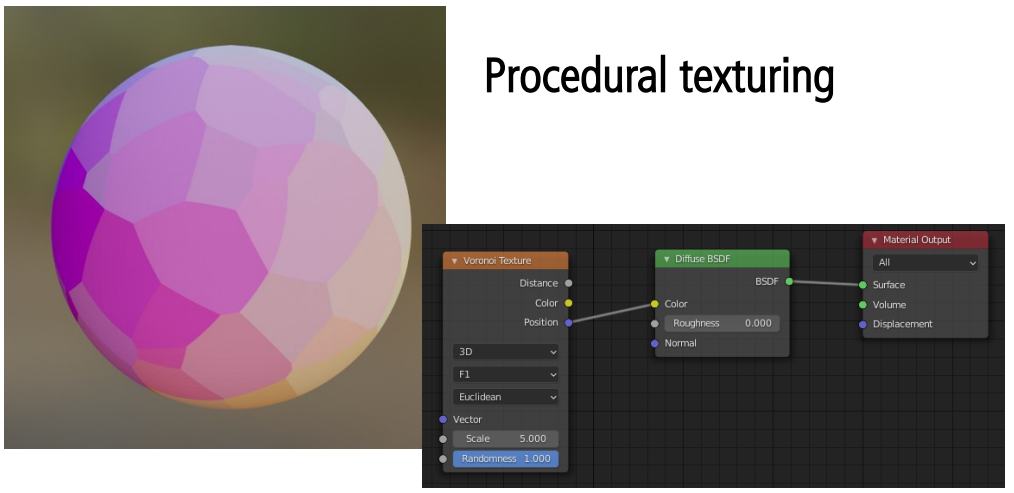
https://www.realtimerendering.com/

Texturing pipeline

- Texturing pipeline is performed by rendering engine and can be applied on **two main types of texturing**:



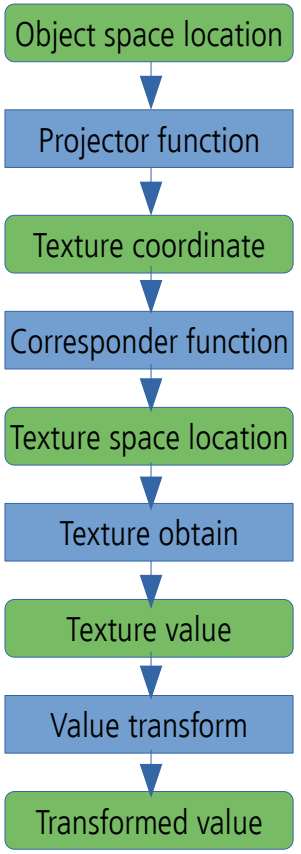
2D image (discrete 2D array of texels) is "pasted" on 3D object.



An algorithm generates pattern on 3D object

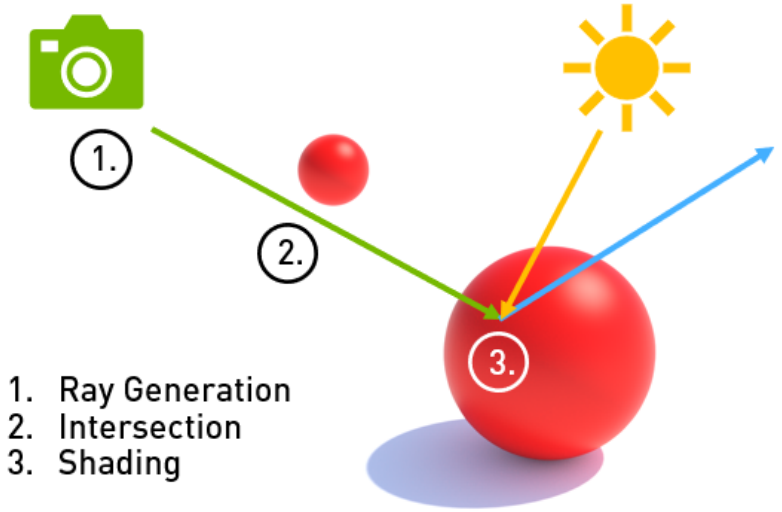
Texturing pipeline

- **Texturing**: variation of shading parameters over surface – **position dependency**
- **Object space location**
  - Shading point → 3D object-space coordinates `(x,y,z)`, e.g., `(1.2, 3.3, -3.4)`
- **Projection function**
  - Mapping process: 3D object space coordinates → 2D texture space coordinates `(u,v)` in `[0,1]`, e.g., `(0.2, 0.5)`
- **Corresponder function**
  - Transform: continuous texture space coordinates (u,v) → discrete texel locations in image texture `(s,t)` e.g., `[image_width, image_height]`, `e.g., [128, 44]`
- **Obtaining texture values (sampling)**
  - Texture space location is used for obtaining texture value
- **Value transform**
  - Optional additional transformations are done on texture value
- **Usage**
  - Final value is used to modify property of surface at that point: scattering function parameter or normal.

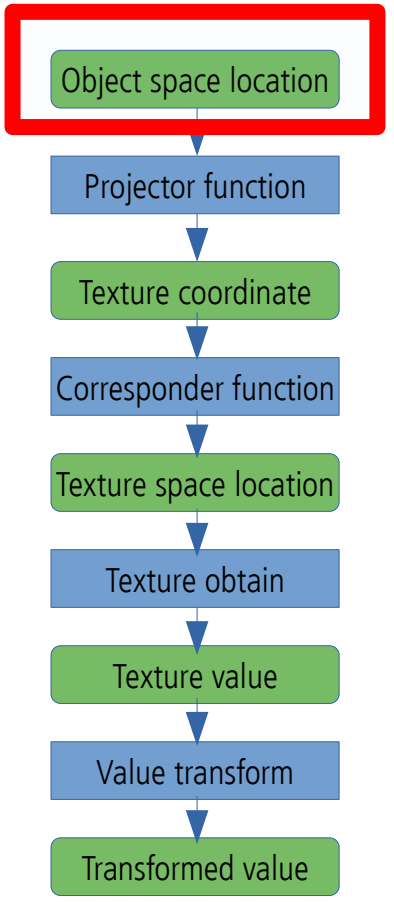


Object space location

- **Intersection** (shading) point
  - **Object space location** `(x,y,z)`
  - World space object location can also be used
    - In this case, as object moves, the coordinates change!

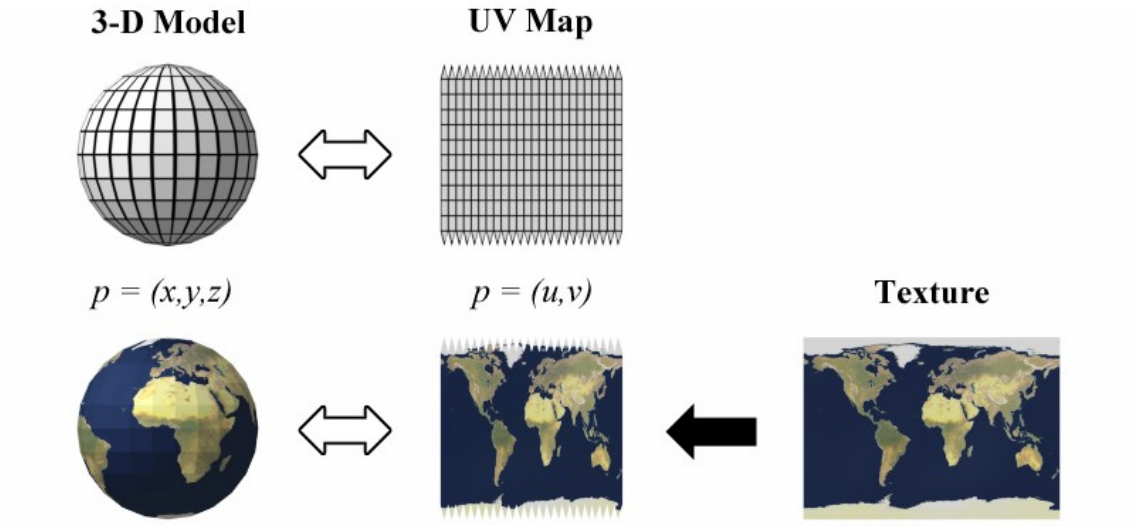


https://developer.nvidia.com/blog/rtx-best-practices/

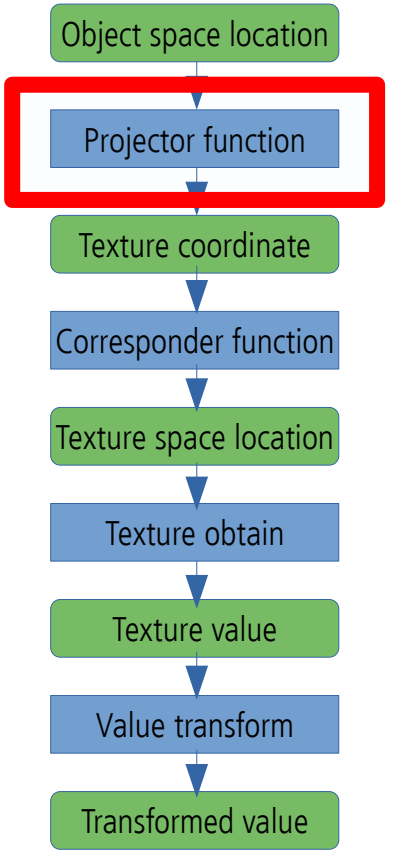


Projector function

- Projector function maps **object space location** into **texture coordinate space**
  - Thus "texture mapping" term
- For image texture: 3D object point `(x,y,z)` to 2D space `(u,v)` in `[0,1]`

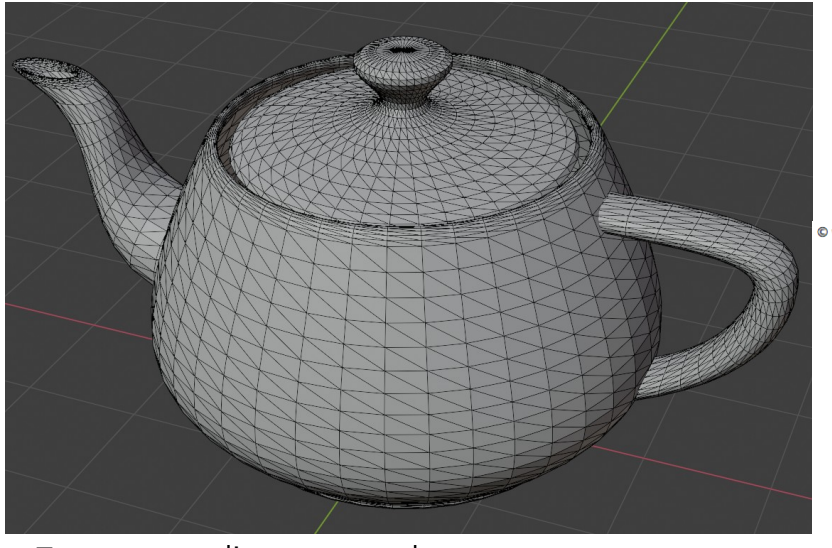


https://users.csc.calpoly.edu/~zwood/teaching/csc572/final10/wdhamilt/

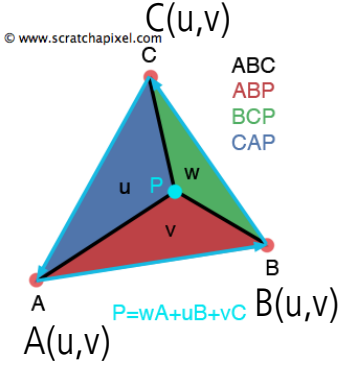


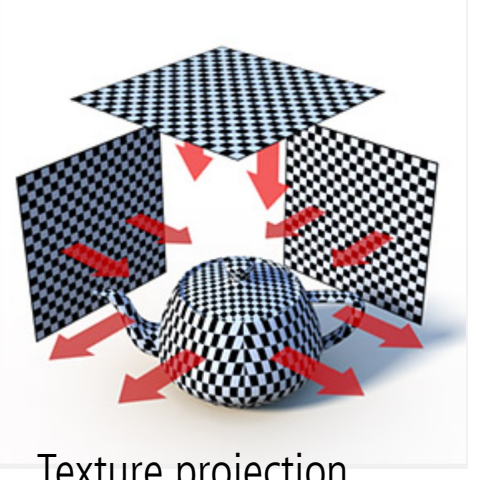
Projector function

- Object space location is used to:
  - **Interpolate precomputed texture coordinates stored per vertex**
  - Calculate texture coordinates using texture projection on the fly



Texture coordinates stored per vertex

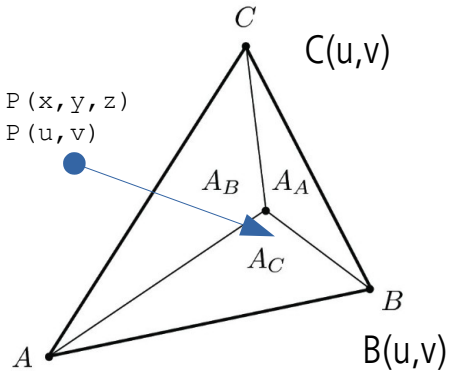




Texture projection

Vertex texture coordinates

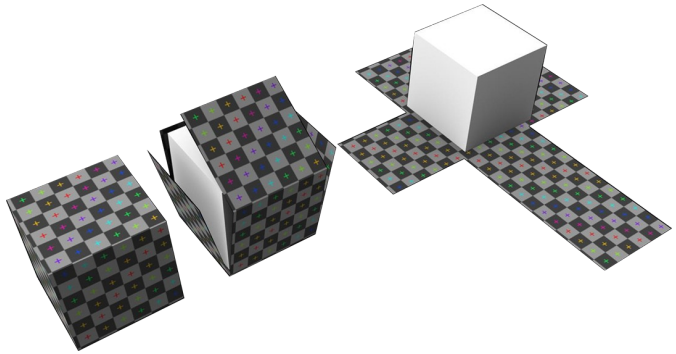
- For intersected/shading point `P(x,y,z)` on triangle of mesh, **barycentric interpolation** is used to calculate texture coordinate `P(u,v)` from ones stored per vertex.



$$\alpha = \frac{A_A}{A_A + A_B + A_C}$$
$$\beta = \frac{A_B}{A_A + A_B + A_C}$$
$$\gamma = \frac{A_C}{A_A + A_B + A_C}$$

`P(u,v) = αA(u,v) + βB(u,v) + γC(u,v)`

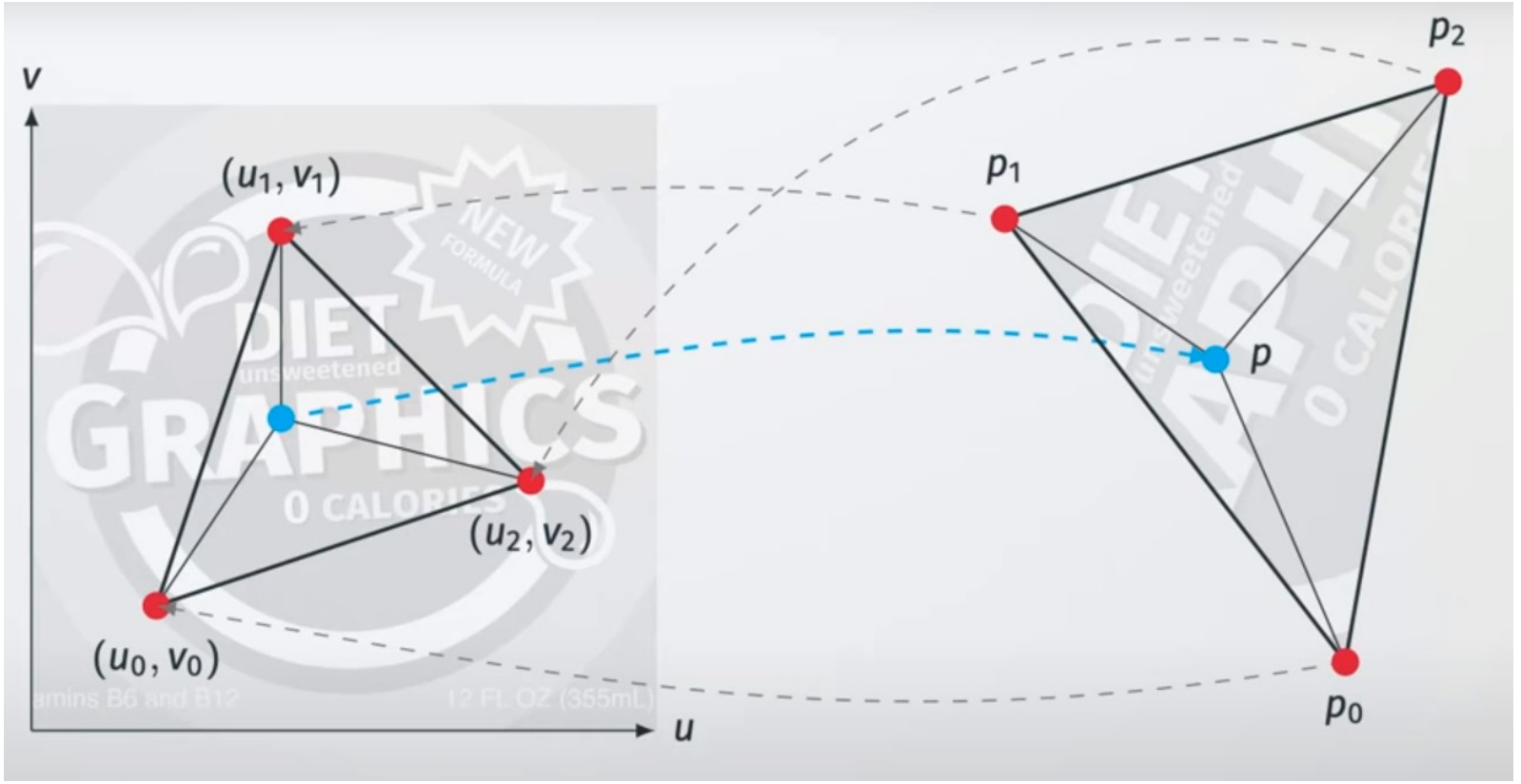
$\alpha + \beta + \gamma = 1$



https://help.disguise.one/en/Content/3D-Workflow/UV-Mapping/What-is-UV-mapping.html

```
# Blender v2.92.0 OBJ File: ''
# www.blender.org
o Cube_Cube.002
v -1.000000 -1.000000 1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
vt 0.625000 0.000000
vt 0.375000 0.250000
vt 0.375000 0.000000
vt 0.625000 0.250000
vt 0.375000 0.500000
vt 0.625000 0.500000
vt 0.375000 0.750000
vt 0.625000 0.750000
vt 0.375000 1.000000
vt 0.125000 0.750000
vt 0.125000 0.500000
vt 0.875000 0.500000
vt 0.625000 1.000000
vt 0.875000 0.750000
vn -1.0000 0.0000 0.0000
vn 0.0000 0.0000 -1.0000
vn 1.0000 0.0000 0.0000
vn 0.0000 0.0000 1.0000
vn 0.0000 -1.0000 0.0000
vn 0.0000 1.0000 0.0000
s off
f 2/1/1 3/2/1 1/3/1
f 4/4/2 7/5/2 3/2/2
f 8/6/3 5/7/3 7/5/3
f 6/8/4 1/9/4 5/7/4
f 7/5/5 1/10/5 3/11/5
f 4/12/6 6/8/6 8/6/6
f 2/1/1 4/4/1 3/2/1
f 4/4/2 8/6/2 7/5/2
f 8/6/3 6/8/3 5/7/3
f 6/8/4 2/13/4 1/9/4
f 7/5/5 5/7/5 1/10/5
f 4/12/6 2/14/6 6/8/6
```

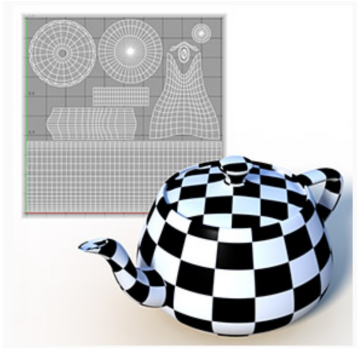
Vertex texture coordinates



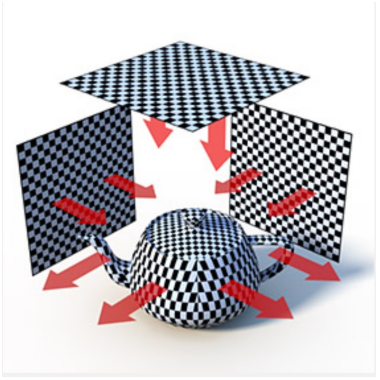
https://www.youtube.com/watch?v=78kP5VaKPdQ&list=PL4TptkuzgxxUVZ-_DiO33kp4_rkoAy1BC&index=3&ab_channel=ChristophGarth

Generating vertex texture coordinates

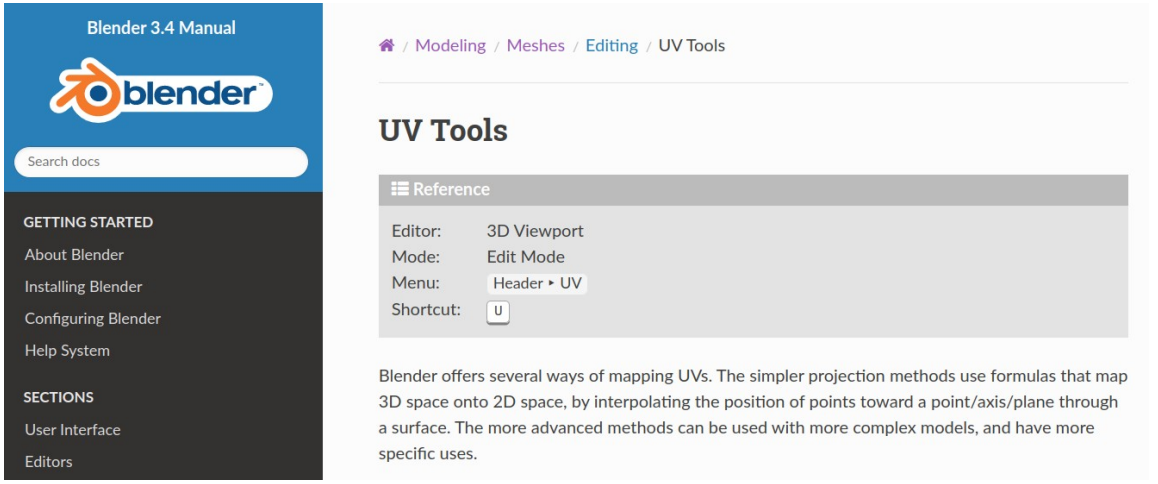
- **Generating texture coordinates per vertex:**
  - **Mesh unwrapping**
  - **Texture projection**
- Modeling packages implement different methods for generating texture coordinates per vertex and enable artist to edit texture coordinates



Unwrapping



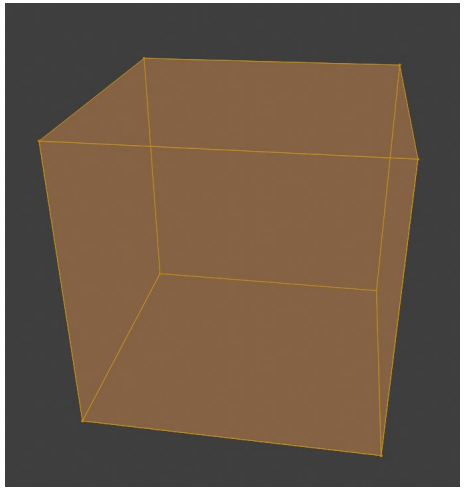
Texture projection

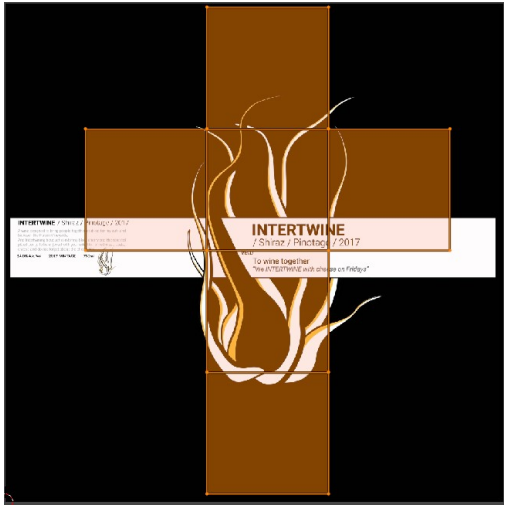


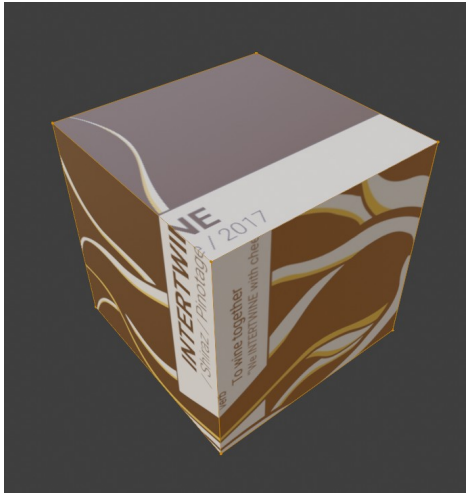
https://docs.blender.org/manual/en/latest/modeling/meshes/editing/uv.html

Mesh unwrapping

- **Mesh (UV) unwrapping**: each vertex is assigned with 2D texture coordinate `(u,v)` in `[0,1]`
  - Unwrapping methods belongs to larger field called **mesh parametrization**
- Goal is that each polygon gets fair share of texture area (e.g., evade stretching) but maintaining connectivity – which determines where separate parts of texture meet and visible seams.

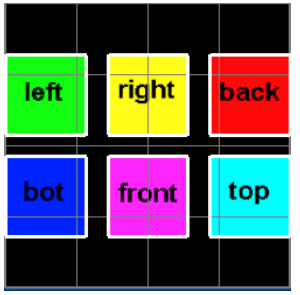


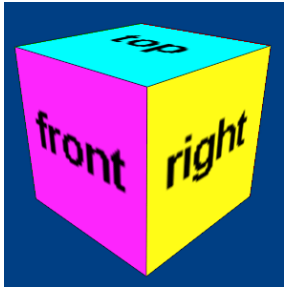




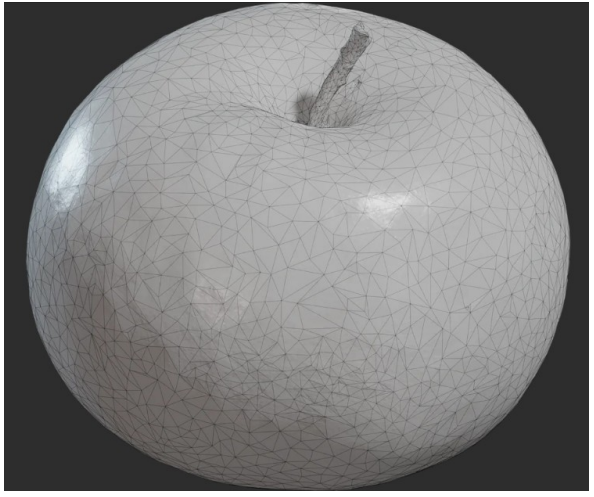
Mesh unwrapping and texture atlas

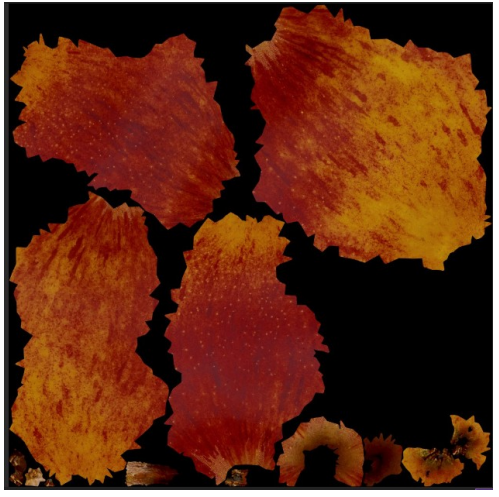
- For complex meshes, **unwrapping separates mesh into multiple parts** which are more convenient to represent on flat plane.
- **Each mesh part is assigned with image texture** → piecewise parametrization
- **All image textures for each part can be stored into singe texture → texture atlas**
  - Highly memory efficient





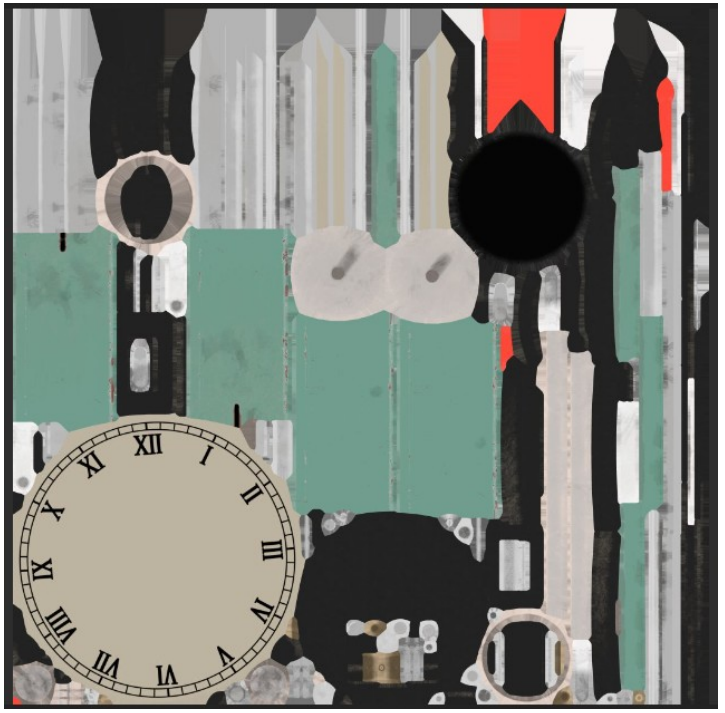
https://www.unwrap3d.com/u3d/tutorial_uv_mapping_texture_atlas.aspx





Blender example: https://www.youtube.com/watch?v=gZRDwbHEB34&ab_channel=GrantAbbitt

Texture atlas

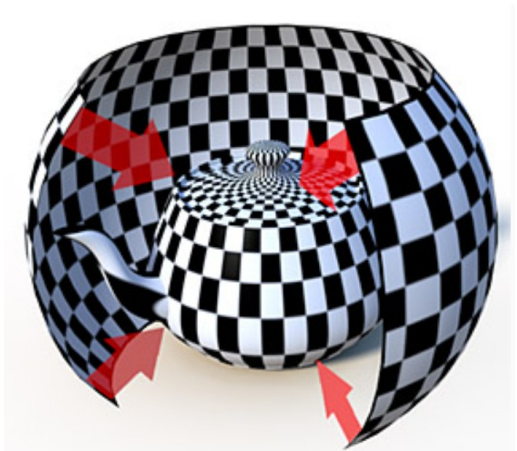


Texture projection

- Transform 3D object point `(x,y,z)` in space into 2D texture coordinates `(u,v)` using projections:
  - Spherical
  - Cylindrical
  - Planar
  - Cubic
  - Box
  - Front

Spherical texture projection

- Texture is wrapped into a spherical shape and projected onto a shape.
- Good for texturing round objects.
- Surfaces perpendicular to projection cause stretching.
- Blinn and Newell developed environment mapping based on sphere projection that is called latitude-longitude mapping



https://learn.foundry.com/modo/901/content/help/pages/shading_lighting/shader_items/projection_type_samples.html

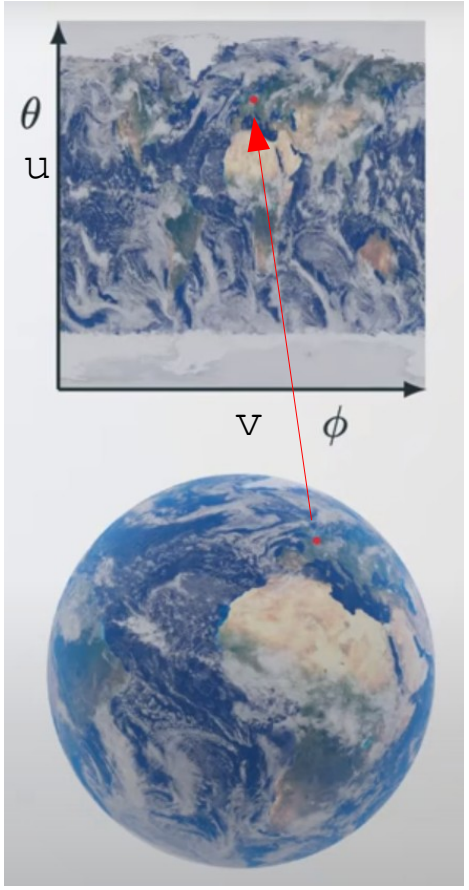
Spherical texture projection

- Example: mapping 2D image texture on 3D sphere
- Any point `(x, y, z)` on 3D sphere with radius `r` can be transformed to polar coordinates:
  - Azimuth Φ in `[0, 2π]`
  - Elevation θ in `[0, π]`
- Azimuth and elevation coordinates are normalized to obtain texture coordinates `u` and `v`

$$\theta = \cos^{-1}\left(\frac{z}{r}\right)$$

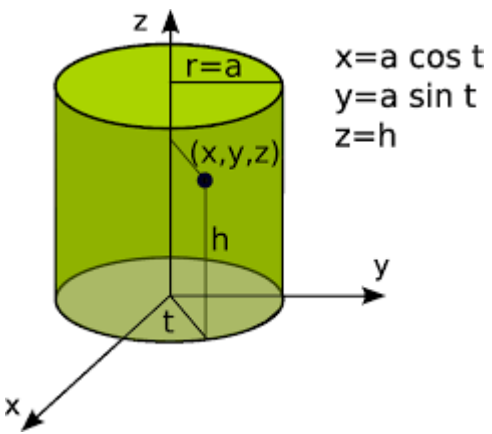
$$\phi = \operatorname{atan2}(x, y) = \begin{cases} \cos^{-1}\left(\frac{x}{r}\right) & y \geq 0 \\ -\cos^{-1}\left(\frac{x}{r}\right) & y < 0 \end{cases}$$

$$u = \theta/\pi,\ v = \phi/(2\pi)$$



Cylindrical texture projection

- Texture is wrapped in cylindrical shape and projected onto surface.
  - Cylinder coordinates `(t,h)` → texture coordinates
- Good for cylindrical surfaces (cans, bottles, etc.).
- Surfaces perpendicular to projection might cause stretching.

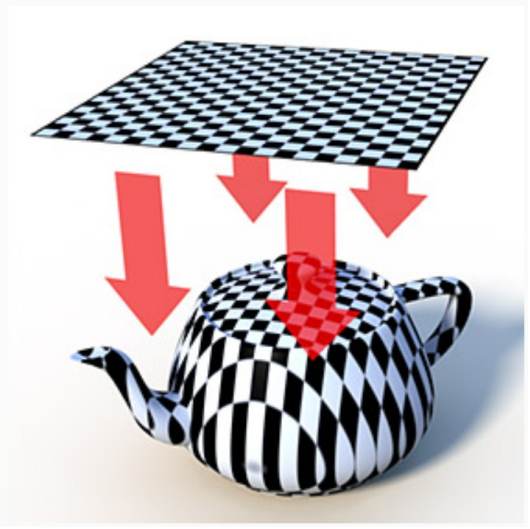


x=a cos t
y=a sin t
z=h

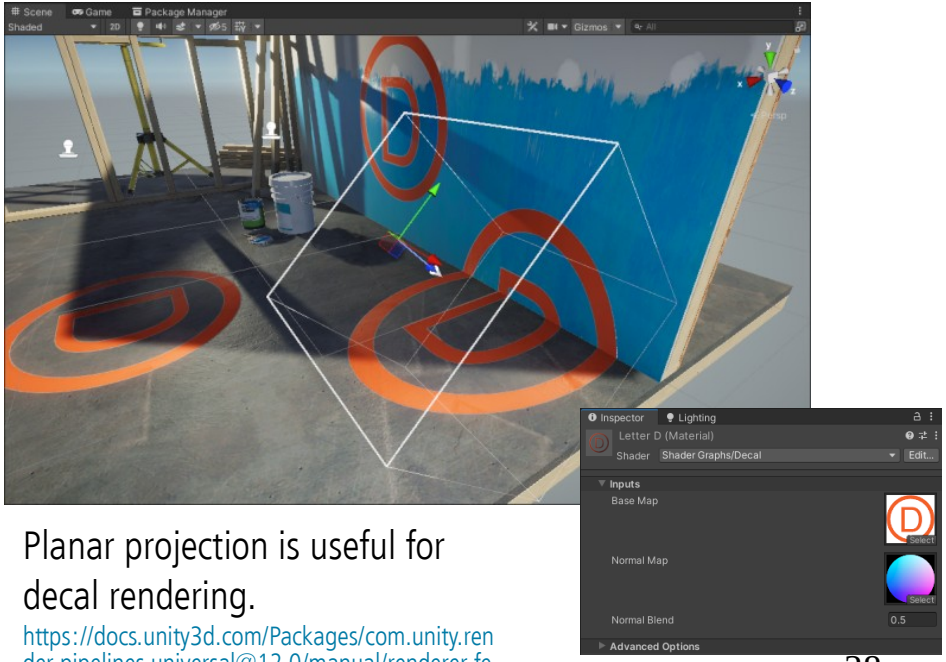
https://learn.foundry.com/modo/901/content/help/pages/shading_lighting/shader_items/projection_type_samples.html

Planar texture projection

- Similar to concept of movie projector but image is projected orhographically.
  - Any vector defining projection plane can be used.
- Great for flat or nearly flat surfaces.
- Other surfaces might cause stretching of texture.



https://learn.foundry.com/modo/901/content/help/pages/shading_lighting/shader_items/projection_type_samples.html

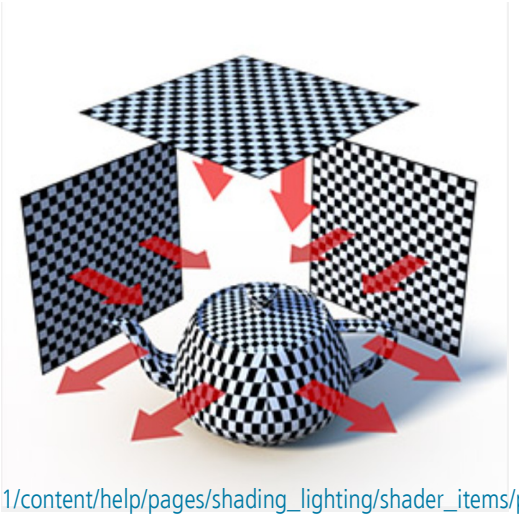


Planar projection is useful for decal rendering.

https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@12.0/manual/renderer-feature-decal.html

Cubic texture projection

- Texture is planar-projected on a surface from all three axis directions: X, Y, and Z
- Polygon receives a certain projection, based on its normal direction.
- Best for cube-shaped objects, and occasionally on detailed surfaces where texture seams are not of great concern

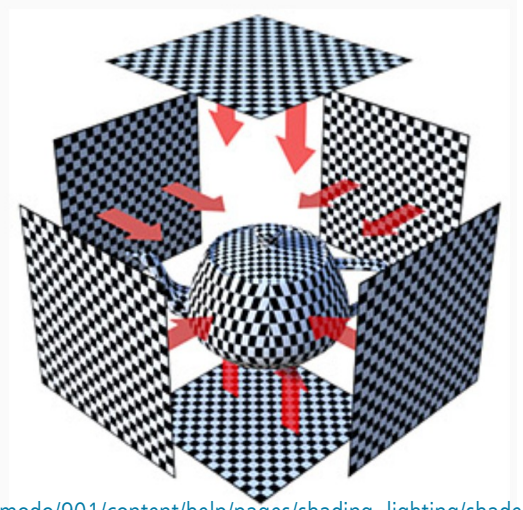


https://learn.foundry.com/modo/901/content/help/pages/shading_lighting/shader_items/projection_type_samples.html

Box texture projection

- Projects a texture in a planar fashion from all six directions, eliminating reverse projections on rear facing polygons inherent to the Cubic method (aka **triplanar mapping**)
- Best on cube-shaped objects, and occasionally on detailed surfaces where texture seams are not of great concern
  - For curved and tilted surfaces efficient **blending** of multiple projection planes must be used.

https://ryandowlingsoka.com/unreal/triplanar-dither-biplanar/



https://learn.foundry.com/modo/901/content/help/pages/shading_lighting/shader_items/projection_type_samples.html

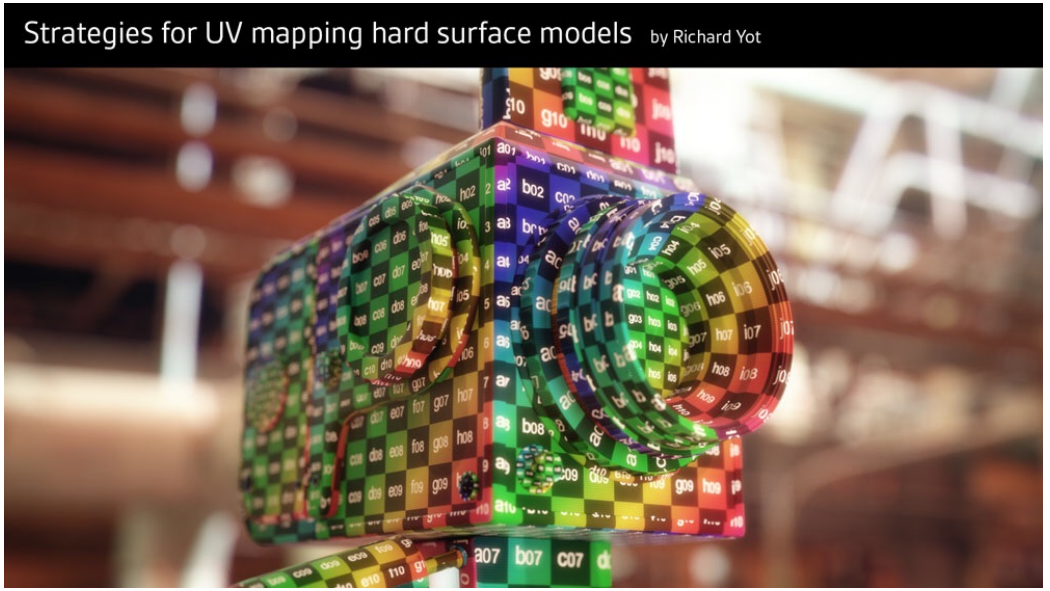
Front texture projection

- Texture is projected out from a camera or light's position

https://learn.foundry.com/modo/901/content/help/pages/shading_lighting/shader_items/projection_type_samples.html

Projector function: complex shapes

- For highly complex shapes, separating into simpler shapes can be done and then texture projection can be done separately.



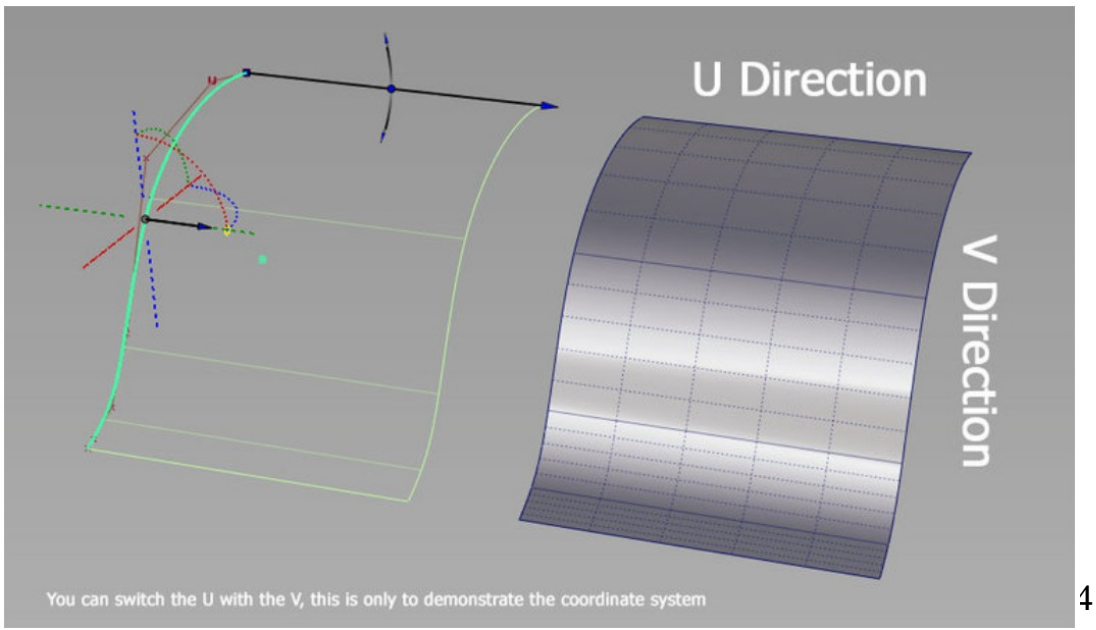
[983] Pagán, Tito, "Efficient UV Mapping of Complex Models," Game Developer, vol. 8, no. 8, pp. 28-34, August 2001

Projector function: different approaches

- **Texture coordinates must be generated in this step**
  - Using position and normal for deriving those is only one way of doing it
- If object is supplied with any kind of additional information (e.g., curvature or temperature) it can be also used for texture coordinate generation

Projector function: inherent texture coordinates

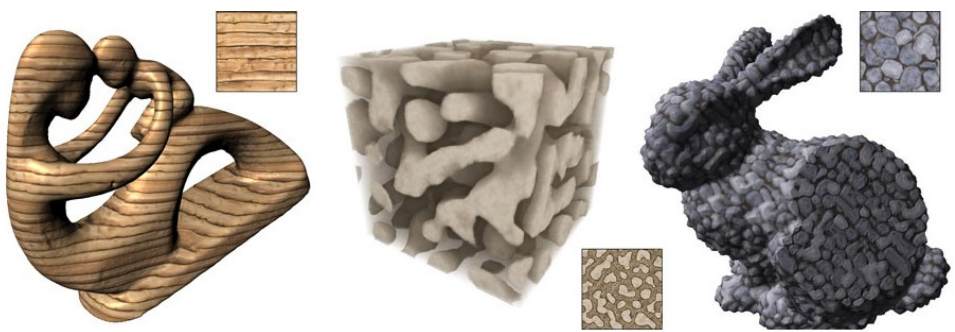
- Parametric surfaces (e.g., NURBS) have natural set of (u,v) texture coordinates which defines any (x,y,z) point on surface
- In this case, we can say that surface representation already has inherent texture coordinates and projector function is not needed.

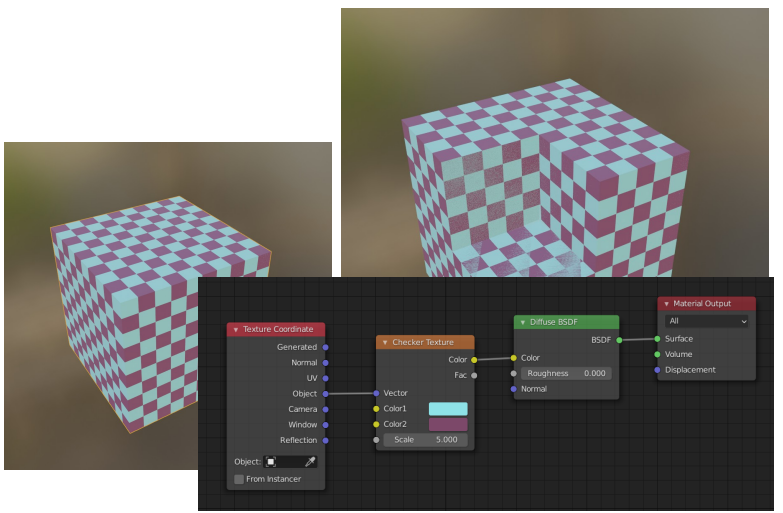


https://www.ebalstudios.com/blog/introduction-nurbs

Projector function: different texture coordinates

- Some steps of texturing pipeline can be omitted depending on texturing method
- Texture coordinate space doesn't have to be always 2D → for volumetric objects, texture coordinates are (u,v,w)
- Procedural texturing can directly use object space locations (x,y,z) – **solid texturing**
- For animation, texture coordinates can be (u,v,w,t) where t is time and determines change in texture





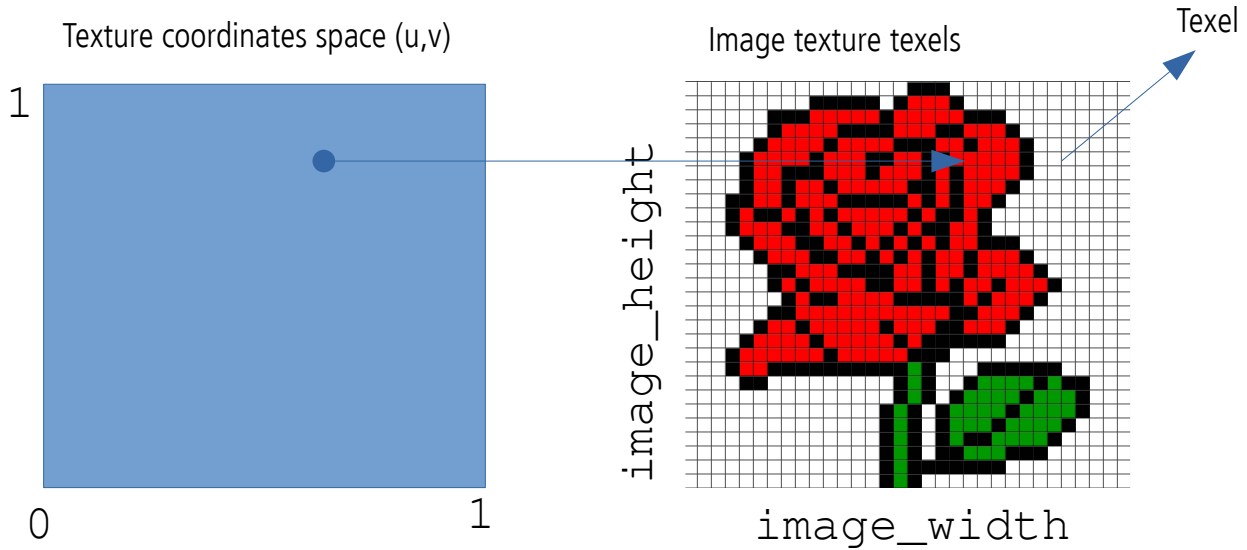
Examples of solid texturing: https://johanneskopf.de/publications/solid/

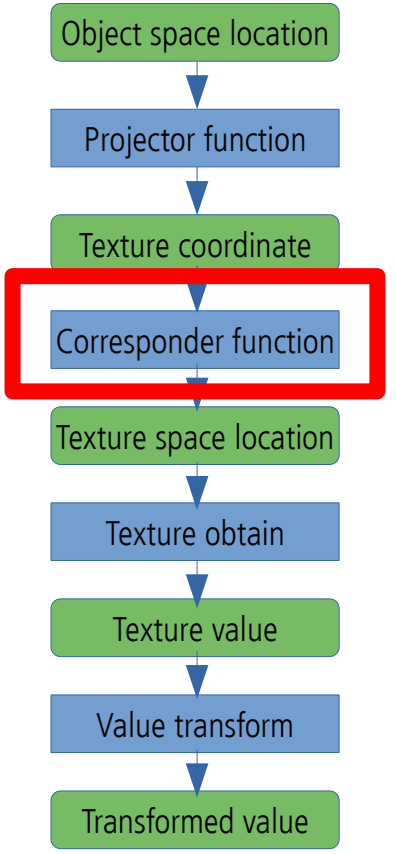
Projector function: multiple texture coordinates*

- Multiple textures can be applied on objects and thus multiple sets of texture coordinates can be applied
- Idea is the same: those texture coordinates are interpolated across surface and used for obtaining texture value

Corresponder function

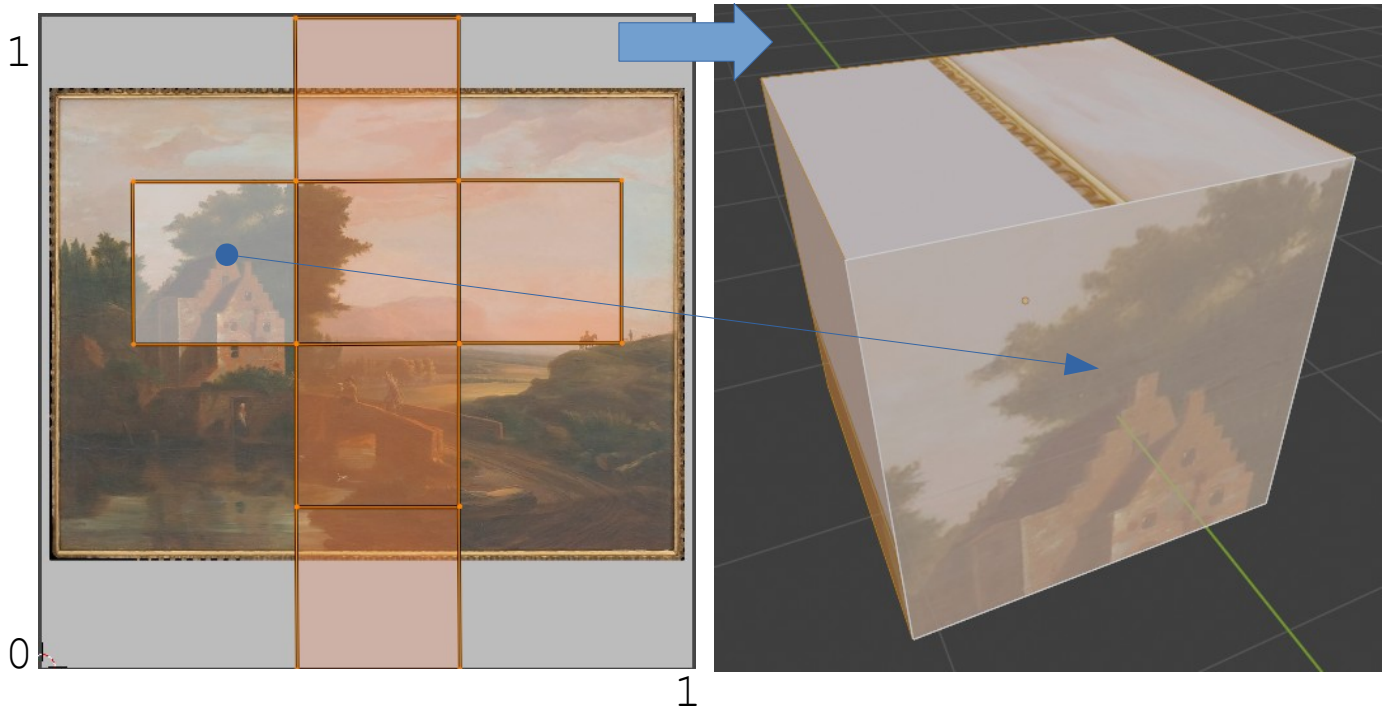
- Transform: **texture coordinates (u,v)** in `[0,1]` → image texture **texel locations (s,t) in `[image_width, image_height]`**
  - This transformation is independent of texture resolution! That is, image of any resolution can be used for sampling given texture coordinates.





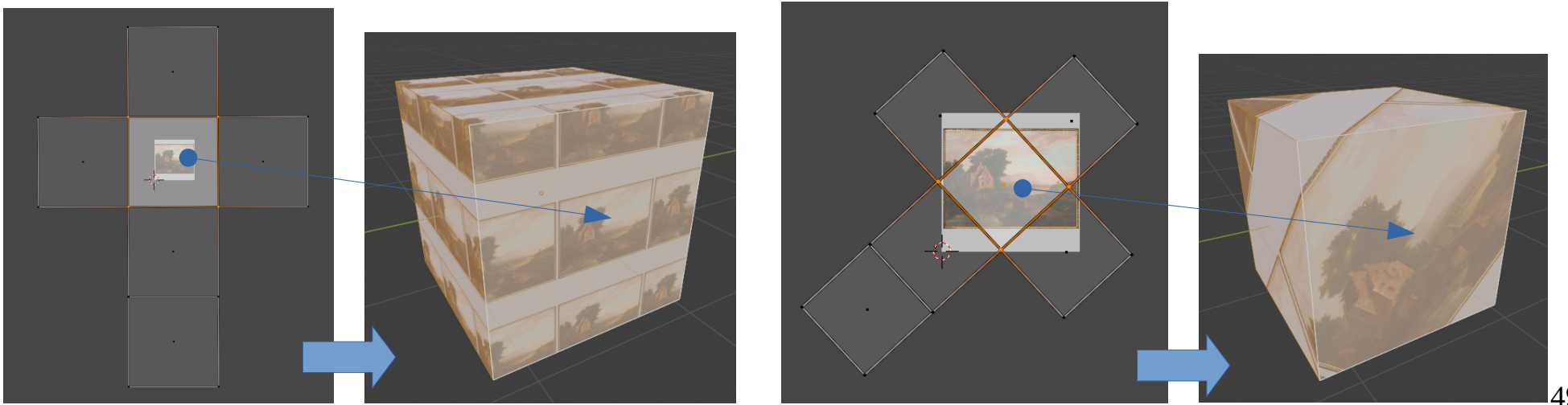
Corresponder function

- Corresponder function determines which part of image texture will be used



Corresponder function

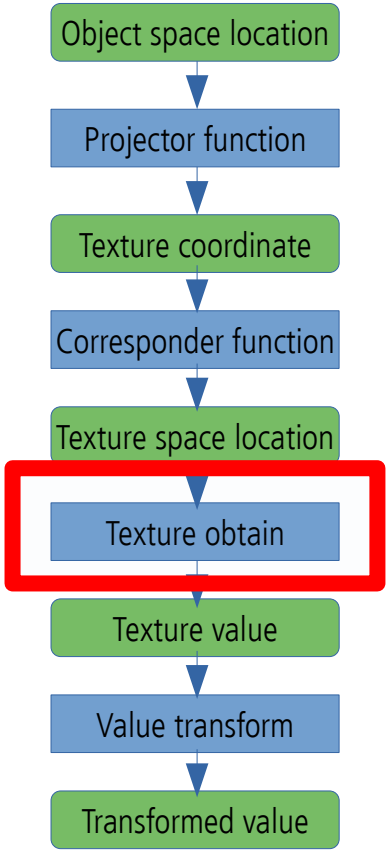
- **Transformations on texture coordinates: rotate, scale, translate, shear or project**
- **Originally (u,v) is in [0,1]. Scaling can move (u,v) to be larger than one**
  - Image wrapping: **repeat** (other: wrap, mirror, clamp, border, wang tiles, etc.)



Demo: https://math.hws.edu/graphicsbook/demos/c4/texture-transform.html

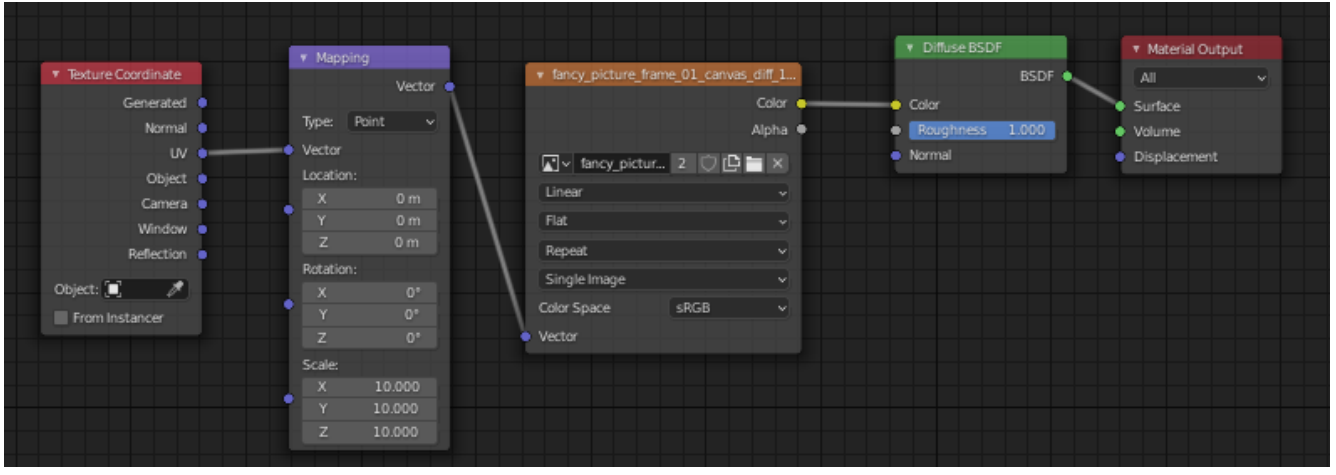
Obtaining texture values

- Texel locations (s,t) in `[image_width, image_height]` are used for obtaining texture value:
  - Reading image texture
  - Evaluating procedural texture



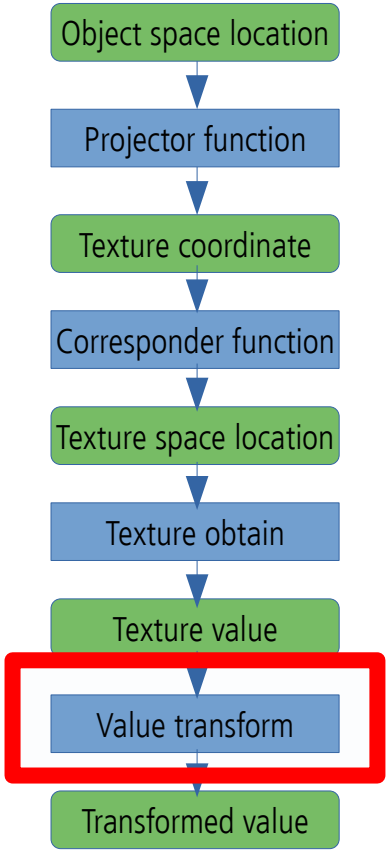
Obtaining texture values

- Corresponder function gave image texture texel coordinates
  - Those coordinates are used for reading texel values
- Note: procedural textures values can be obtained without corresponder function, e.g., directly from object or texture space coordinates



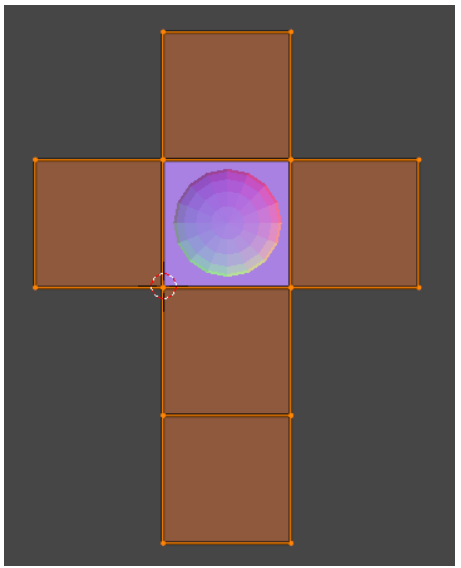
Transforming texture values

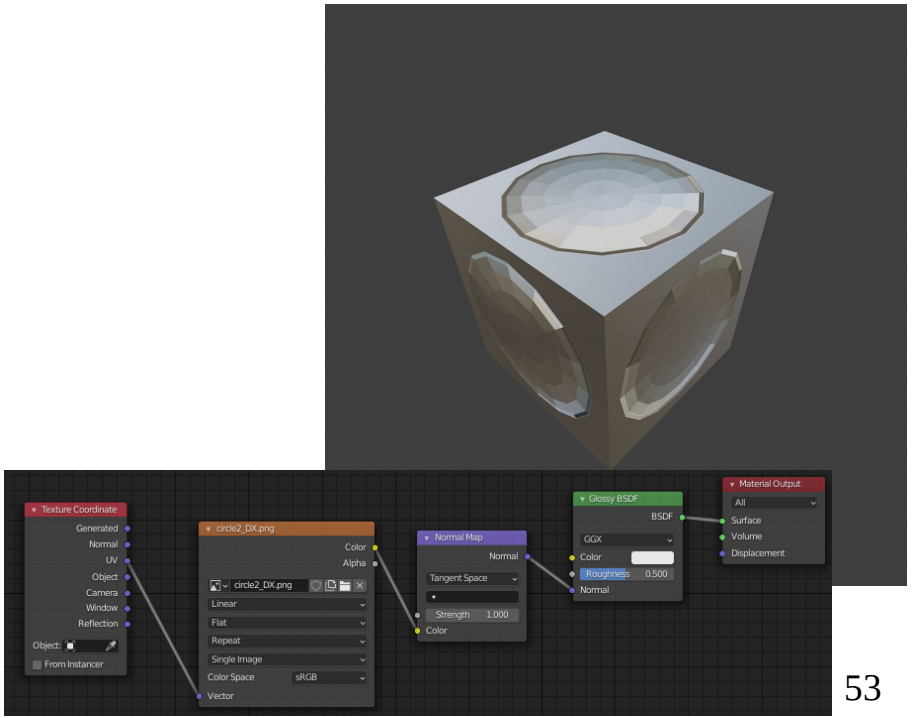
- Optional additional transformations on texture value
  - e.g., normalization of image texture values
- Image textures contain RGB(A) values
  - Many data types can be stored/encoded using those values.
  - Although image textures always contain color values, those values can be interpreted differently then just color



Transforming texture values

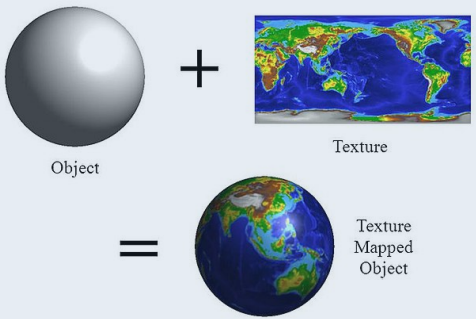
- Example: RGB values can be interpreted as vectors.
  - Normal map image texture contains vectors encoded in RGB



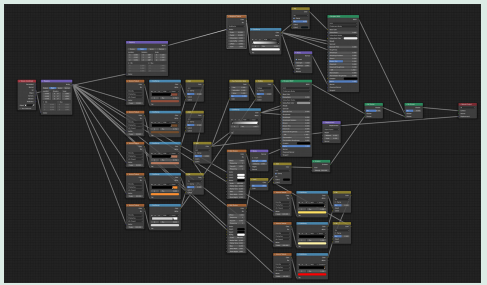


Example: encoding normal vectors into image: **normal maps**

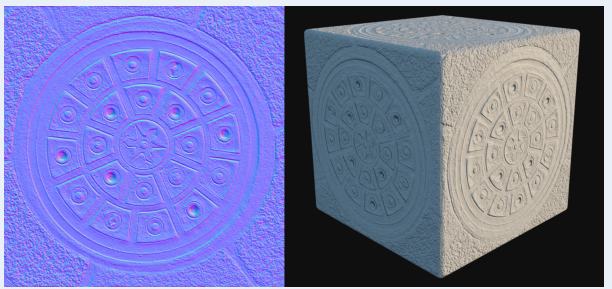
Learning objectives



How to apply textures on objects → **texturing pipeline**



Texturing approaches → **image and procedural**

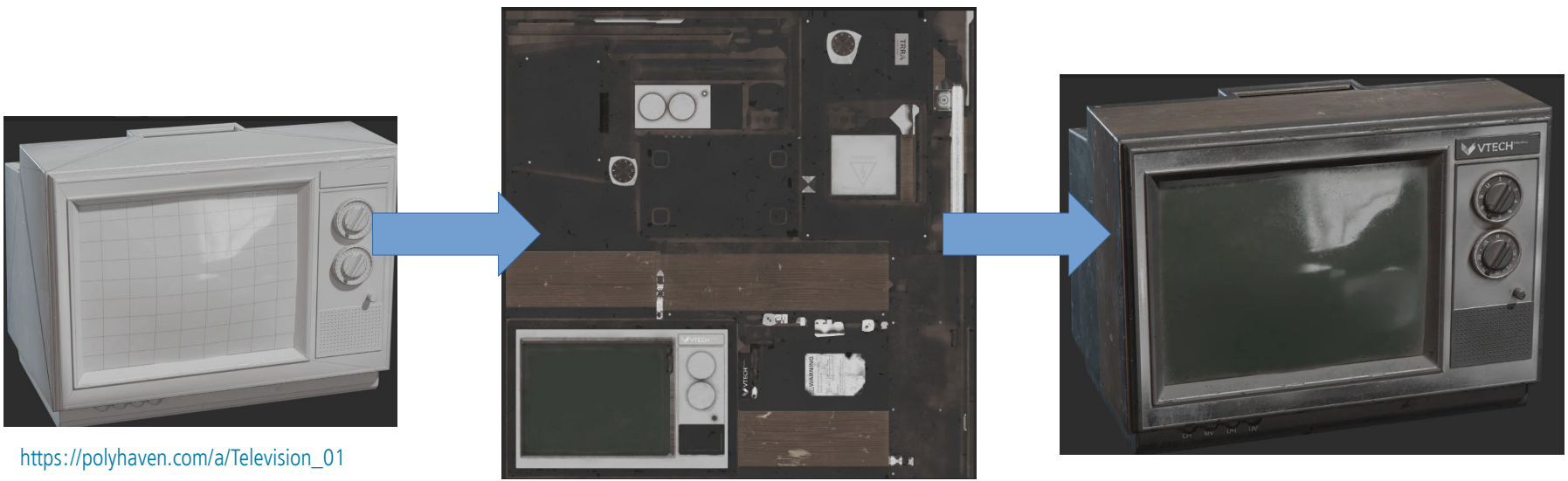


How to use texture for **material modeling** → variation of scattering function parameters and small scale geometry

Image textures

Image textures

- Image texturing process can be seen as "gluing" image on 3D surface
- For particular position on surface `(x,y,z)` we explained how to obtain texture coordinates `(u,v)`
- Image textures are usually 2^m x 2^n texels. Thus called power-of-two textures (POT)*



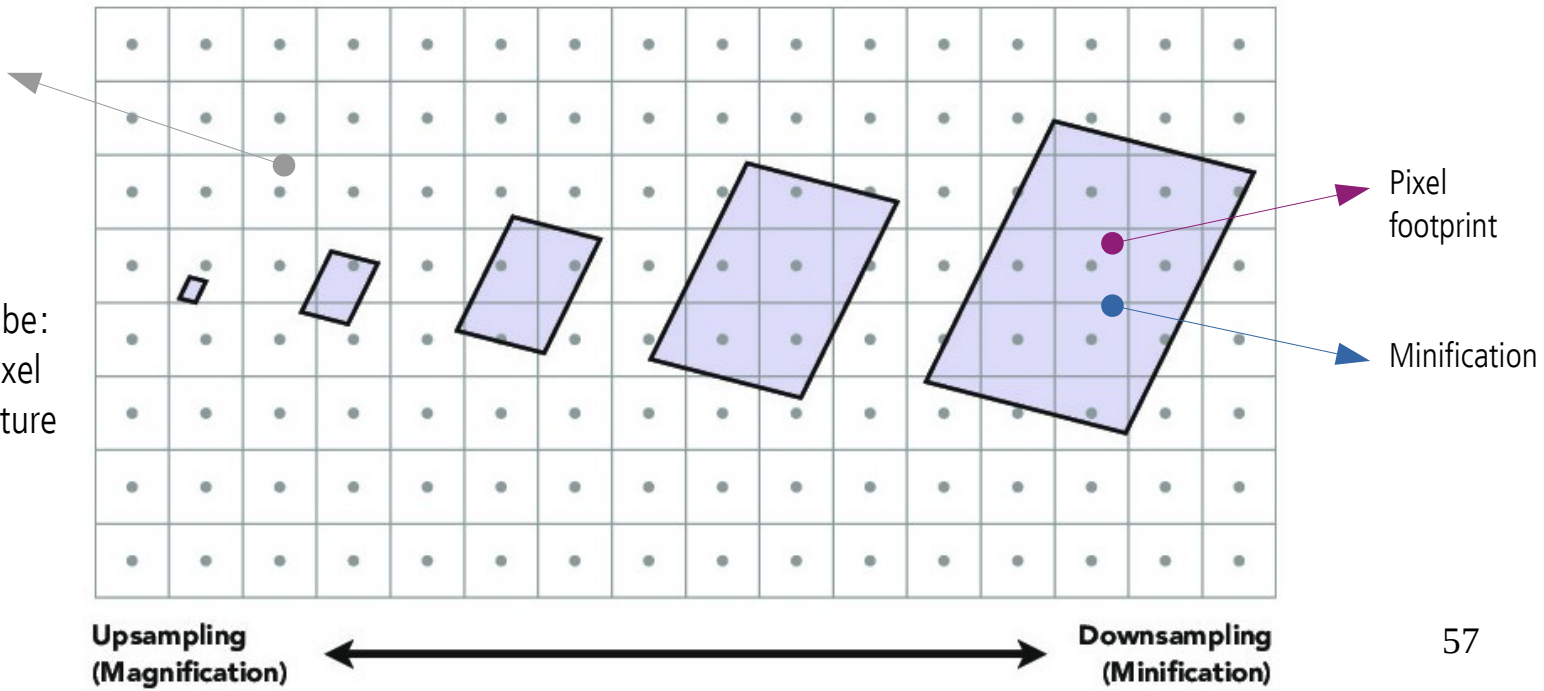
https://polyhaven.com/a/Television_01

Texture atlas

* Older GPUs couldn't support mipmapping for non-POT textures. Modern GPUs handle all textures correctly.

Magnification and minification

- If one or more pixels of virtual image plane falls under one image texture texel: **magnification**.
- If one pixel of virtual image plane covers more then one image texture texels: **minification.**

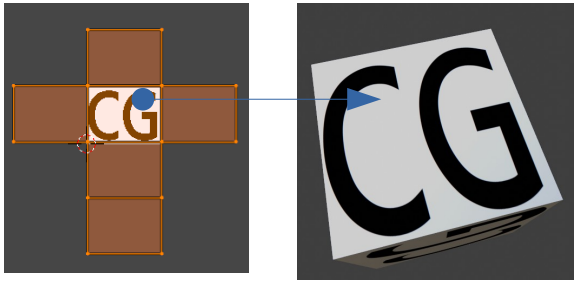


Virtual image pixel footprint can be:

- Smaller than image texture texel
- Larger than several image texture texels

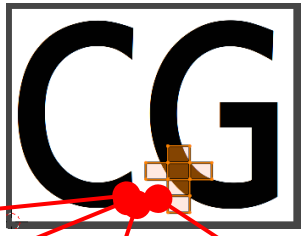
https://cs184.eecs.berkeley.edu/sp21/lecture/5-40/texture-mapping

Texture magnification

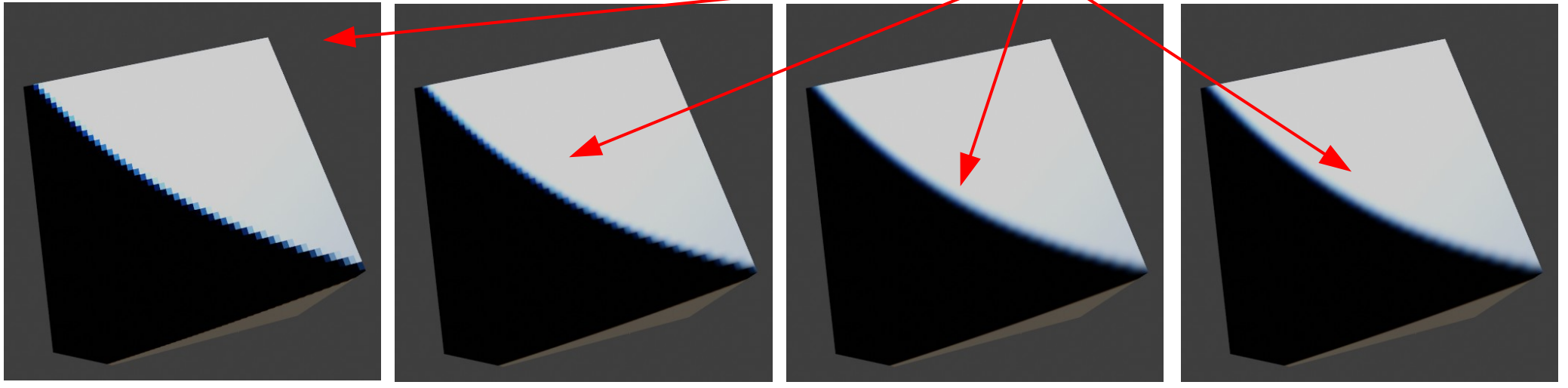


No texture magnification needed.

- System for reading image texture must magnify the image texture.
  - https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/image.html
  - https://www.khronos.org/opengl/wiki/Sampler_Object
- Magnification types:



Texture magnification needed!



Nearest neighbor (closest)
Problem: pixelization

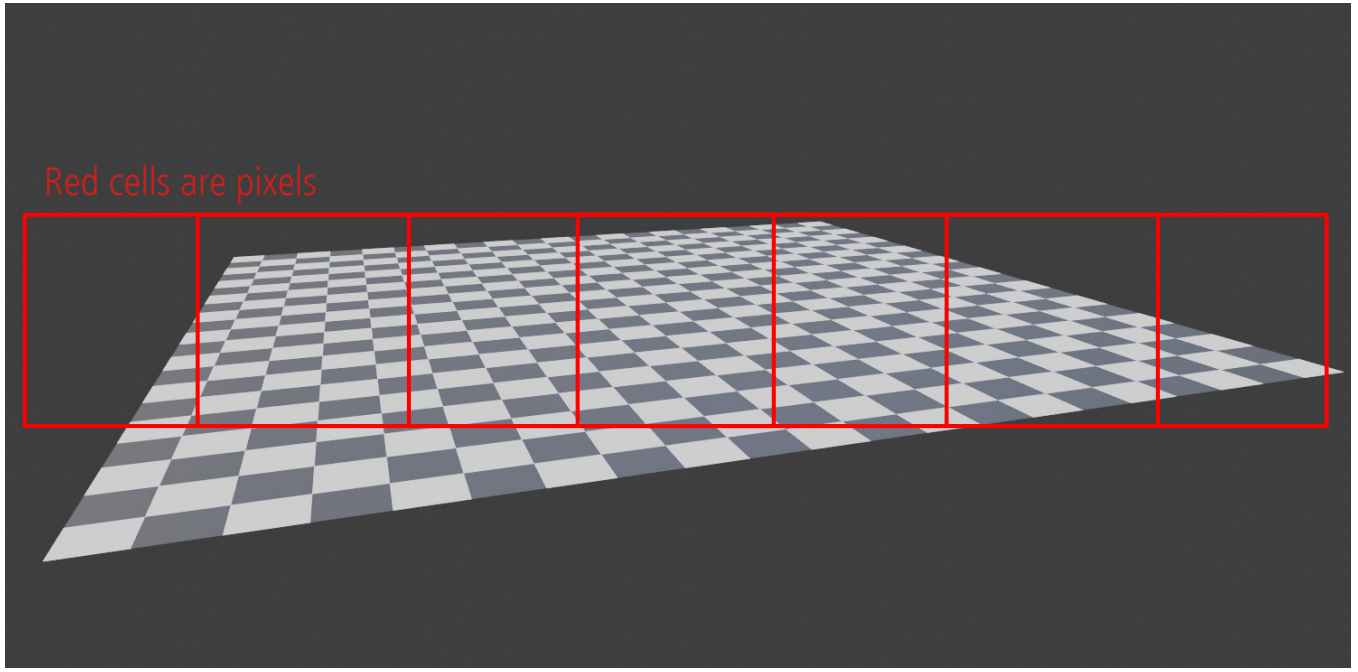
Linear

Cubic filtering

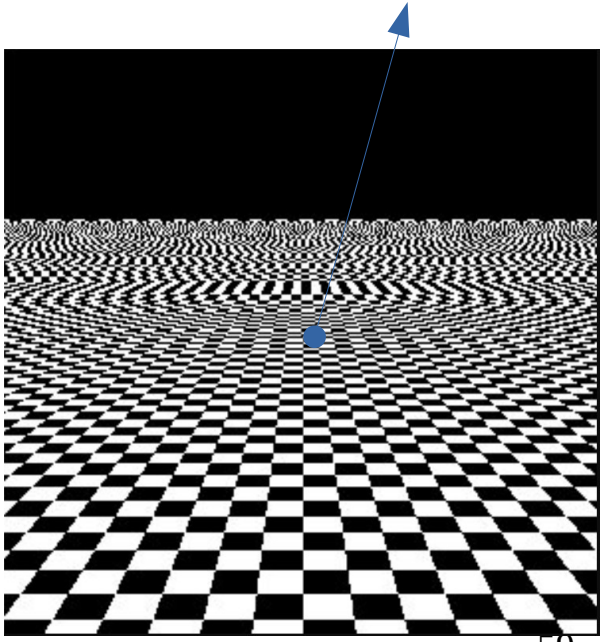
Bilinear interpolation
Problem: blur

Texture minification

- If one pixel of virtual image plane covers more then one image texture texels → **minification**
  - Cumulative effect of texels influencing the pixel must be taken in account
  - How to most accurately represent information under the pixel?



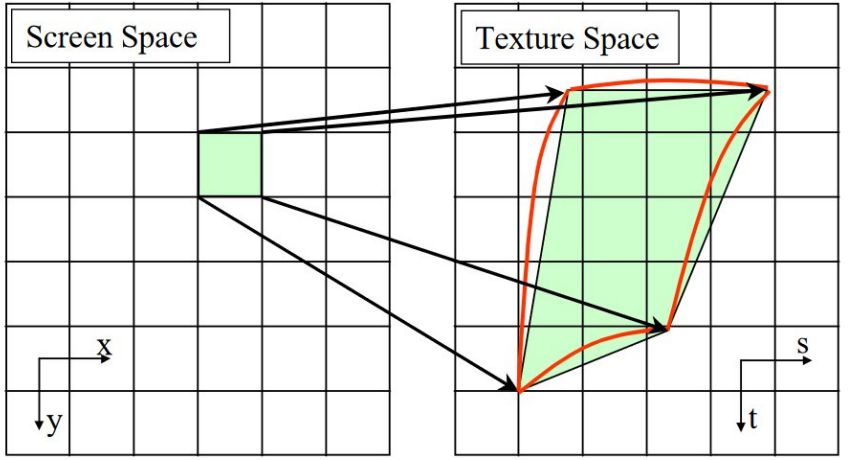
Moire pattern



https://textureingraphics.wordpress.com/what-is-texture-mapping/anti-aliasing-problem-and-mipmapping/

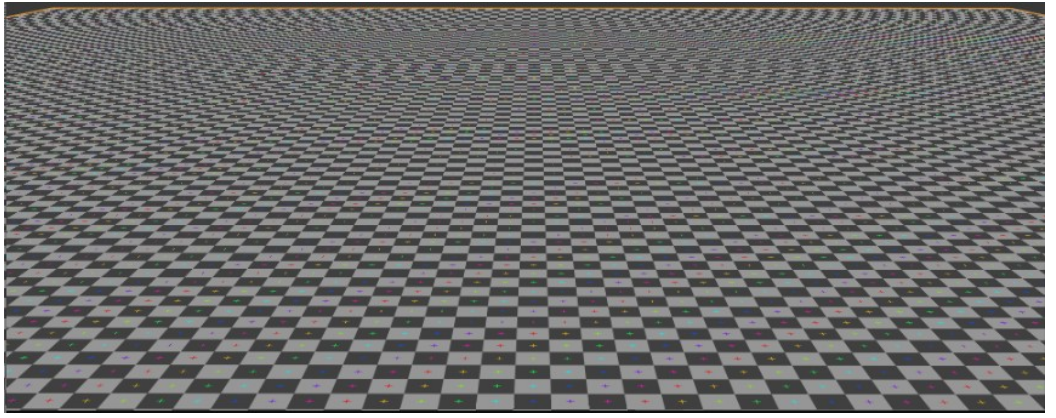
Aliasing

- Frequency of a texture depends how closely are spaced texels on the screen.
- **Nyquist limit** – texture frequency must be not larger than half the sample frequency.
  - **To properly display a texture on a screenm, without aliasing, we need at least one texel per pixel**
- **Anti-aliasing techniques:**
  - Texture frequency (information) must decrease: **mip-mapping**
  - Pixel sampling frequency must increase: **multiple samples per pixel**



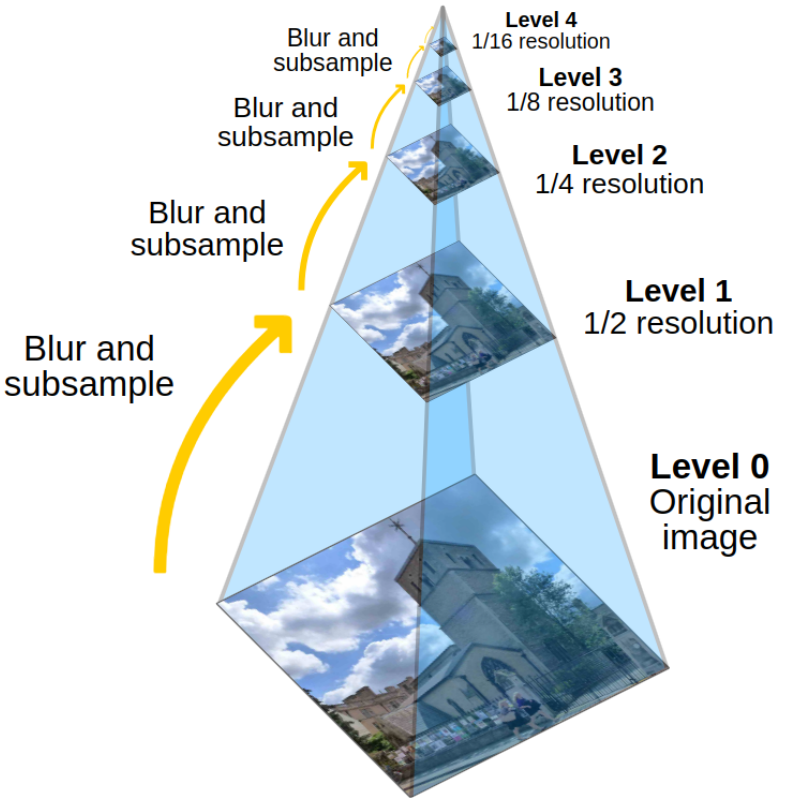
Minification

https://www.researchgate.net/publication/3410856_MIP-map_Level_Selection_for_Texture_Mapping



Anti-aliasing: mip mapping

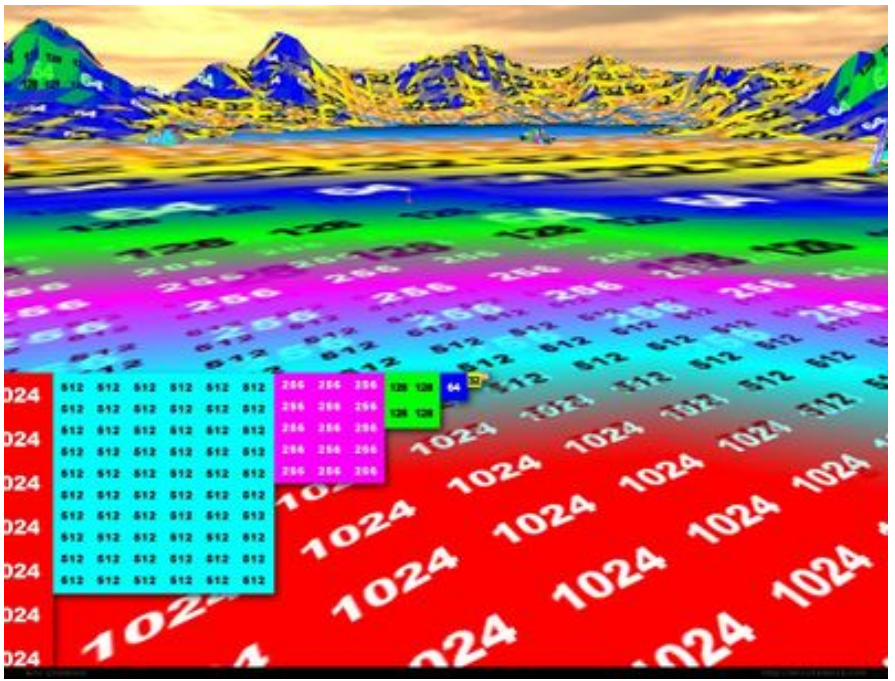
- **MIP – minimization filter**
  - Original image is filtered down repeatedly into smaller images.
  - Before rendering, original image is augmented with sets of smaller versions of this image
- **Image pyramid, Mip map chain:**
  - Original texture is downsampled by quarter of original area
  - New texel is calculated as average of four neighbor texels in original image.
  - This process is repeated until one or both of image texture dimensions equal to one
- **High quality mip mapping requires:**
  - Good filtering: for averaging new texel value from 2x2 of texels it is recommended to use Gaussian, Lanczos or Kaiser filter
  - Images must be in linear colorspace



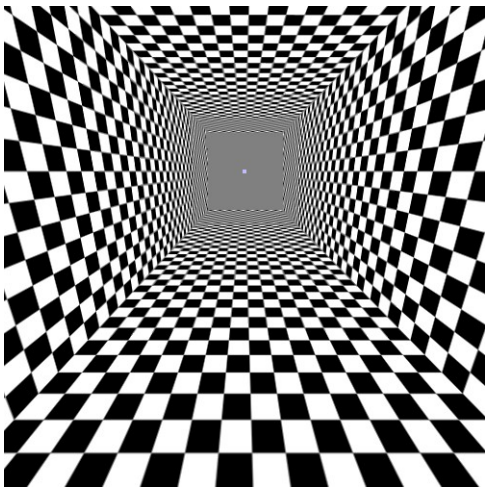
MIP - (multum in parvo (latin)) – many things in small place

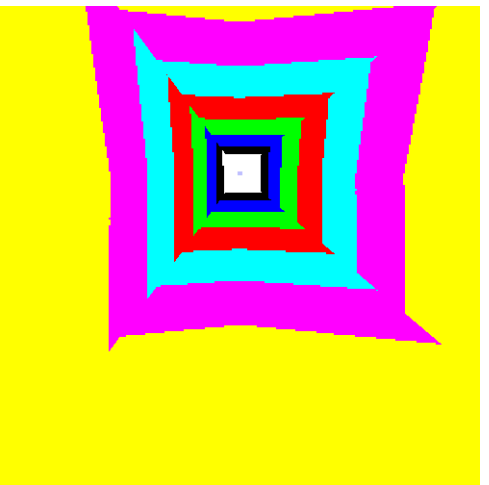
Anti-aliasing: mipmapping

- Depending on camera distance to texture, different mipmaps levels are used to achieve 1:1 pixel to texel ratio to achieve Nyquist rate.
  - OpenGL: https://www.khronos.org/opengl/wiki/Sampler_Object



http://wiki.polycount.com/wiki/Mip_Mapping

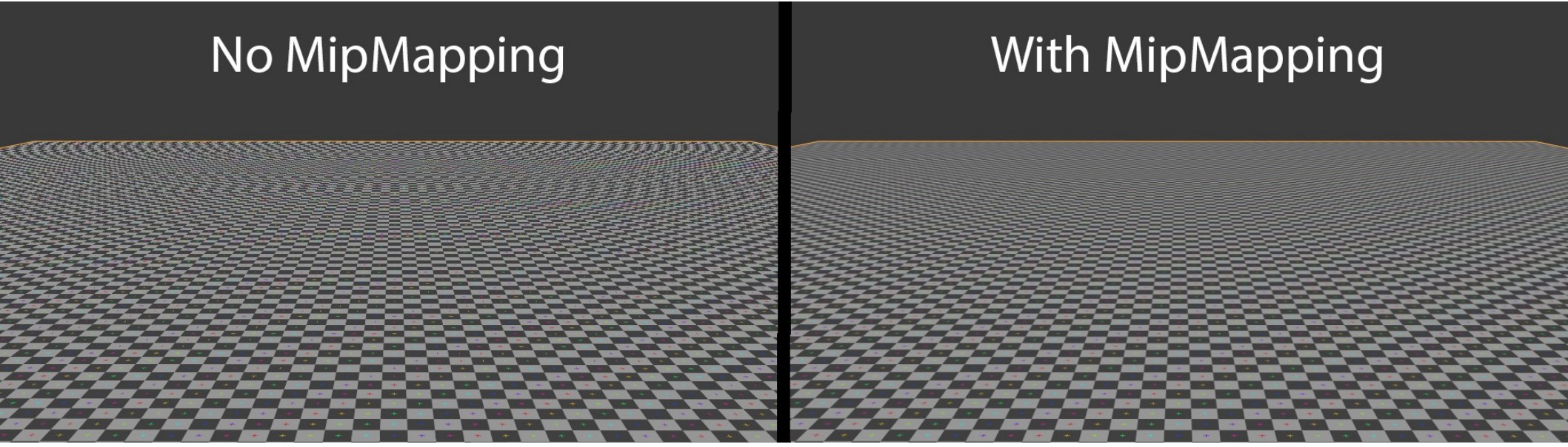




https://paroj.github.io/gltut/Texturing/Tut15%20Needs%20More%20Pictures.html

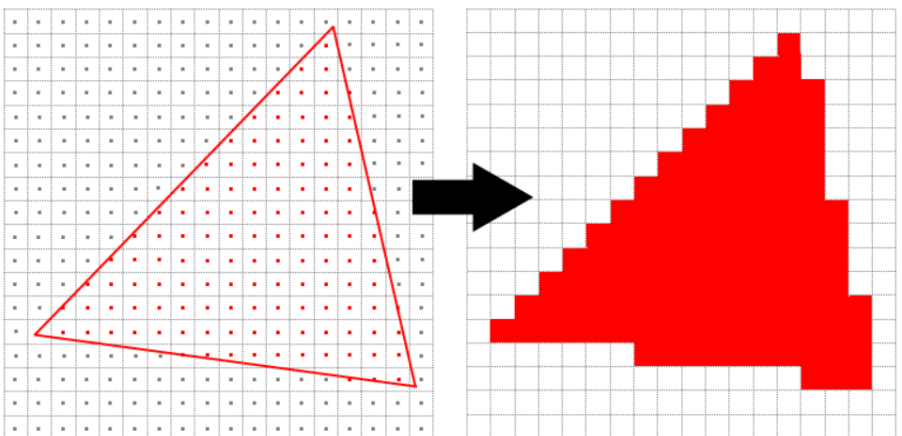
Anti-aliasing: mipmapping

- Achieving 1:1 pixel to texel ratio removes aliasing
  - Moire pattern is removed

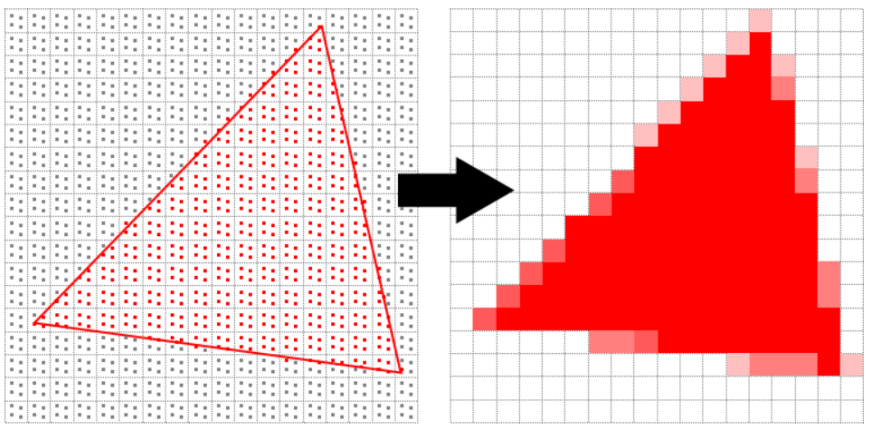


Antialiasing: sampling

- If one pixel of virtual image plane covers more then one image texture texels → **minification**
- Aliasing appears always when there are a lot of details under a pixel footprint in the scene
- To solve this, multiple sample per pixels can be used to "catch" high frequency details.
  - This method is called Multisample anti-aliasing (MSAA)



https://vulkan-tutorial.com/Multisampling



MSAAx4

Alpha texture values

- Image textures have (R,G,B) and **alpha channels**.
- Alpha values are used for blending or alpha testing for transparency.

Decals/cutouts: rendering only parts of the image on object surface.

https://xoio-air.de/2017/10-free-cutout-trees-from-tony-textures/

Billboarding method is used for standalone flat polygons with image texture, always facing the camera.

https://80.lv/articles/how-to-fake-a-large-scale-forest-in-blender/

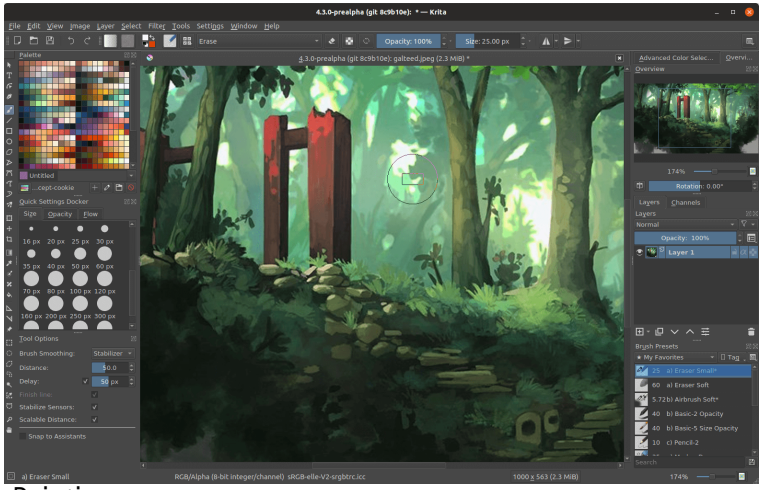
Alpha texture values

- Alpha value is extensively used when adding VFX to rendered or real images - **compositing**.

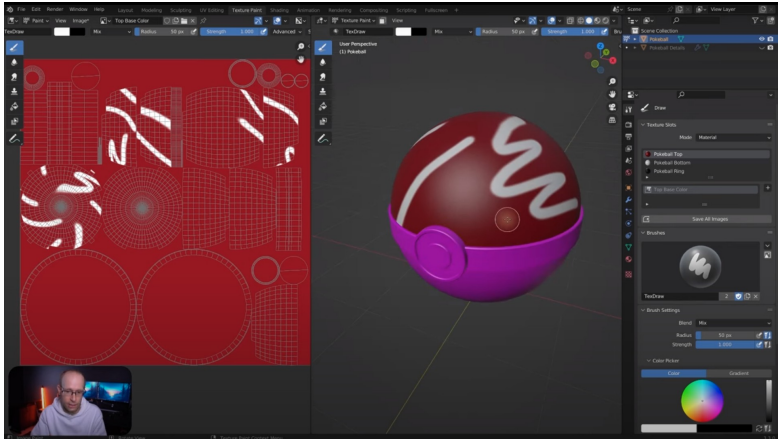
https://www.soundstripe.com/blogs/the-best-techniques-tools-for-compositing

Sources of image textures

- Texture image can be created by:
  - Drawing on canvas: https://krita.org/en/
  - Drawing on 3D object directly: https://docs.blender.org/manual/en/latest/sculpt_paint/texture_paint/index.html
  - Taking photographs
  - Performing measurements of real surfaces
  - Creating from existing geometry or surfaces (process called baking: https://docs.blender.org/manual/en/latest/render/cycles/baking.html)
- Library of textures:
  - PolyHaven: https://polyhaven.com/textures
  - Substance library: https://substance3d.adobe.com/assets/allassets?assetType=substanceMaterial&assetType=substanceAtlas&assetType=substanceDecal



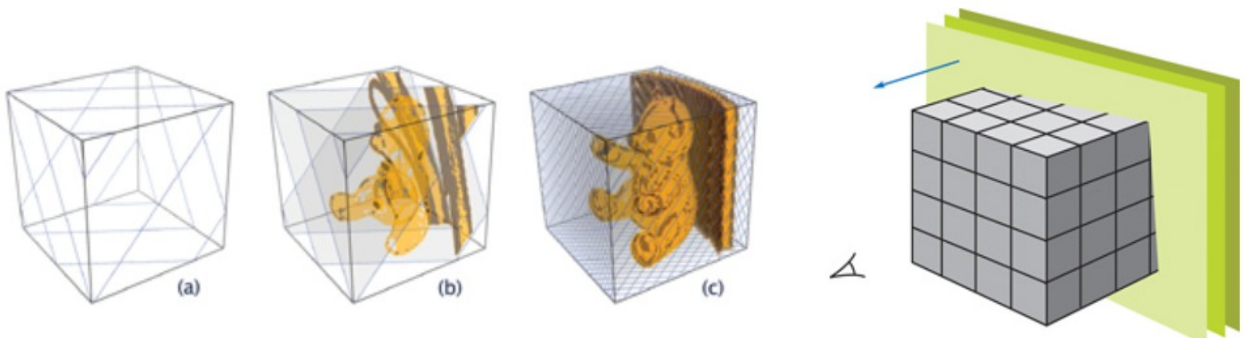
Painting on canvas

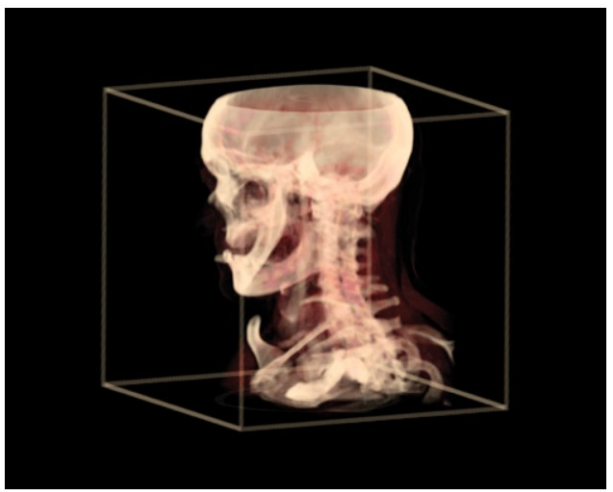


Texture painting on 3D object

Volume textures

- **Extension of 2D image is 3D image, accessed with (u,v,w) coordinates**
- Object locations can be used as texture coordinates.
  - Complex problem of finding good 2D parameterize for 3D mesh is not needed
- Volume texture can represent volumetric structure of material such as wood or marble.
- Example: medical imaging data
  - Three dimensional grid is visualized as slices by moving polygon through it or using ray-casting.
- Problem: volumetric image textures are very space demanding



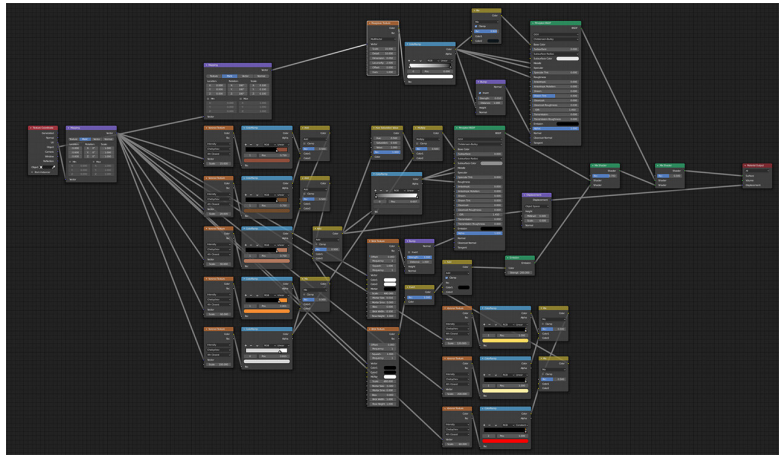


https://developer.nvidia.com/sites/all/modules/custom/gpugems/books/GPUGems/gpugems_ch40.html
https://developer.nvidia.com/gpugems/gpugems/part-vi-beyond-triangles/chapter-39-volume-rendering-techniques

Procedural textures

Procedural textures

Umbrella term for a number of techniques in computer graphics to create 3D models and textures from sets of rules - code segments or algorithms that specify some characteristic of a computer-generated model or effect[1]



1. Musgrave, F. K., Peachey, D., Perlin, K., & Worley, S. (1994). Texturing and modeling: a procedural approach. Academic Press Professional, Inc.

https://www.reddit.com/r/blender/comments/khkz98/finished_my_first_nodevember_link_in_comments/?utm_source=share&utm_medium=web2x&context=3

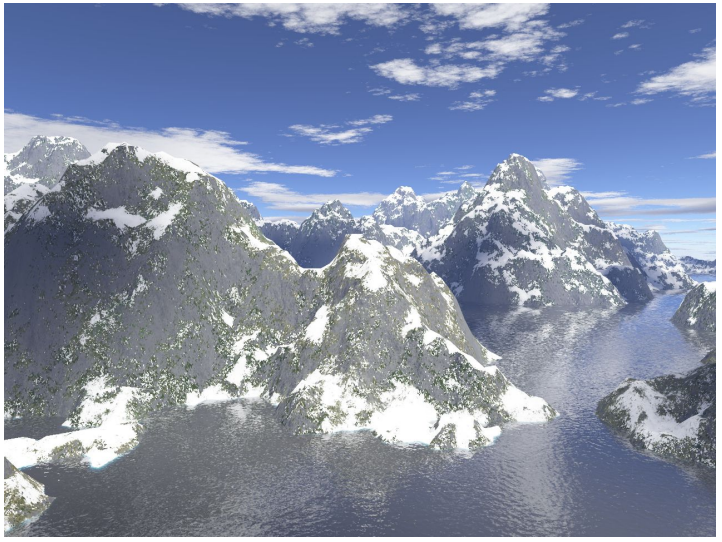
Procedural textures

- **Texture values**, instead of image lookup, are **evaluated from a function**
- Procedural texture defines values for every point in 3D space or 3D surface point → 3D solid textures
  - Parametrization of surface is therefore not needed

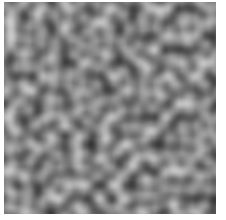
Often based on **noise function**.

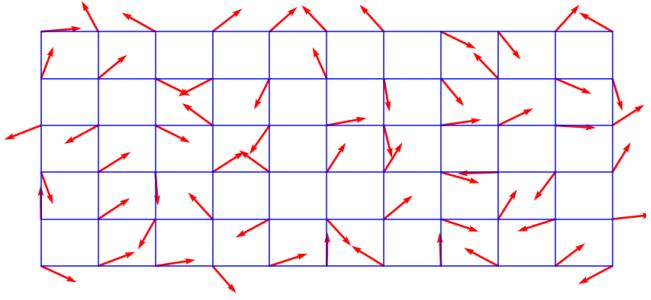
Often used for natural phenomena such as seas, clouds and mountains, etc.

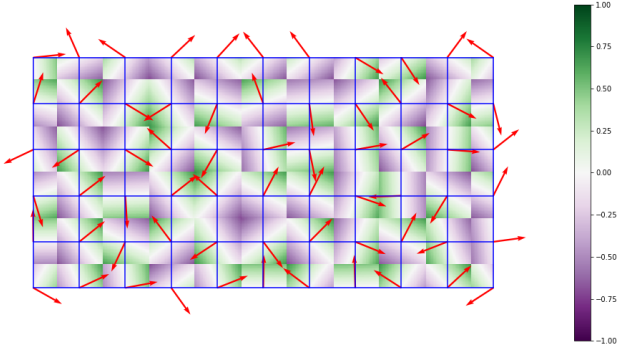
Procedural textures: Perlin noise

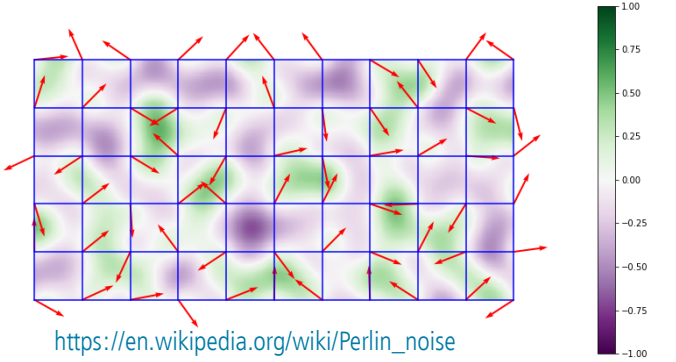


- Property: **local changes are gradual, while global changes can be large**
- Most popular noise function upon which wide range of methods is built
- Type of **gradient noise**.
  - N dimensional grid where each grid intersection has associated n dimensional unit length gradient vector
  - To calculate point inside grid:
    - (1) find cell inside which point lies,
    - (2) calculate dot product of vector from this point to cell corner and gradient vector,
    - (3) interpolate dot products





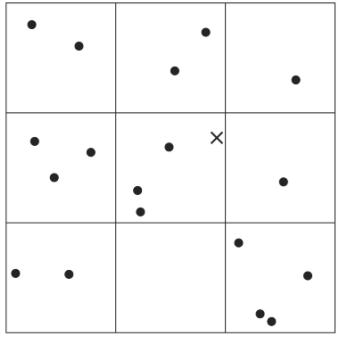


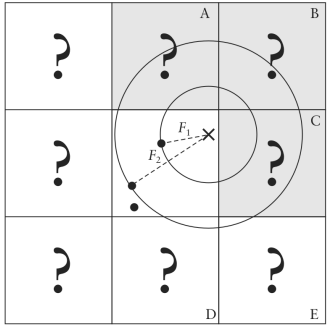


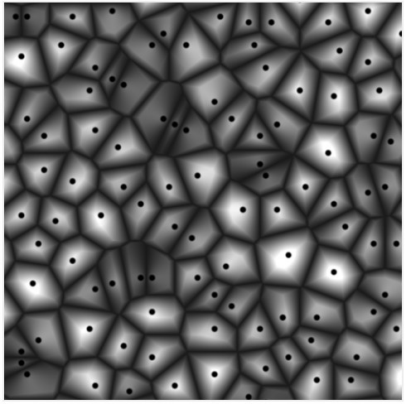
https://en.wikipedia.org/wiki/Perlin_noise

Procedural textures: Cellular texture

- Steven Worley introduced **Voroni-like textures** that can be used for many natural textures: **cells, flagstones, lizard skin, etc.**
  - Another fundamental procedural texture upon which more complex procedural textures are built
- This method is based on **measuring distance from feature points**
  - Space is diced into cubes. Each cube contain random feature points.
  - For current point, find cube inside which lies and check distance to feature points in this and neighboring cubes.
  - First distance is used as value. More combinations with distances are possible.







https://thebookofshaders.com/12/

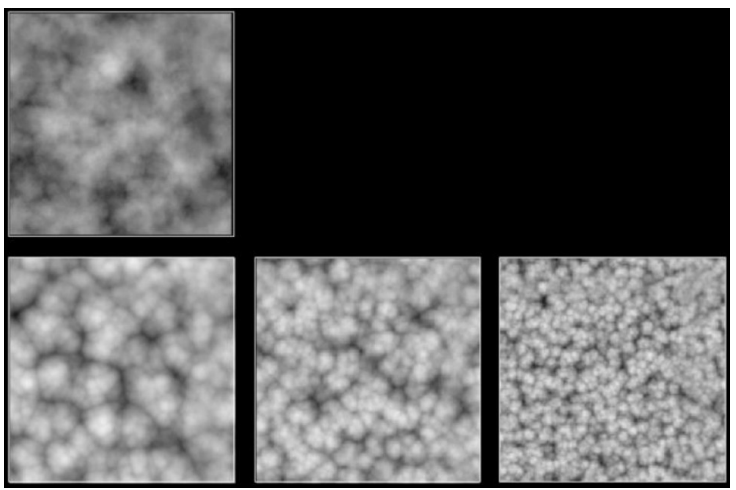
1. Musgrave, F. K., Peachey, D., Perlin, K., & Worley, S. (1994). Texturing and modeling: a procedural approach. Academic Press Professional, Inc.

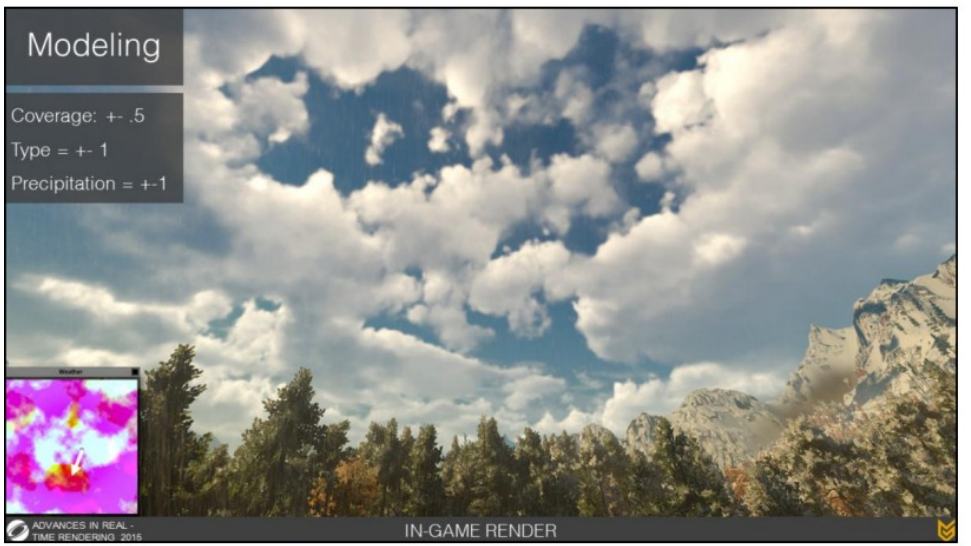
Noise in production

- More complex types of noise are built using Perlin and Worley noise:
  - **Fractal noise**
  - **Simplex noise**
- **Combining Perlin and Worley noise** is often used for more richer details.
- Volume textures are attractive application for procedural texturing since they can be synthesized on the fly

Perlin-Worley

3 layers of Worley

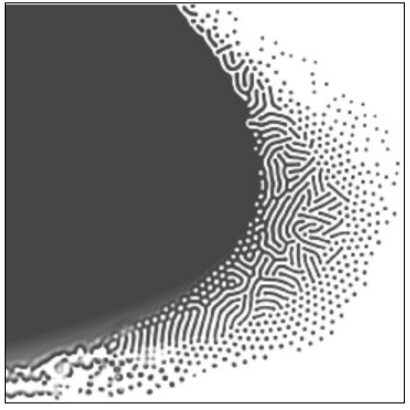


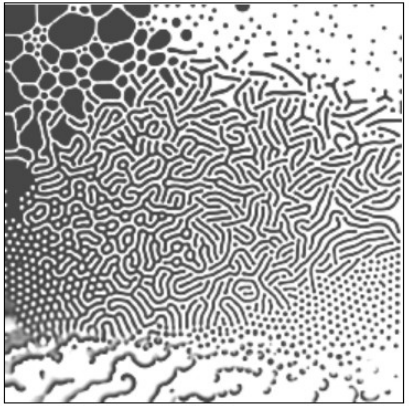


https://www.guerrilla-games.com/read/the-real-time-volumetric-cloudscapes-of-horizon-zero-dawn

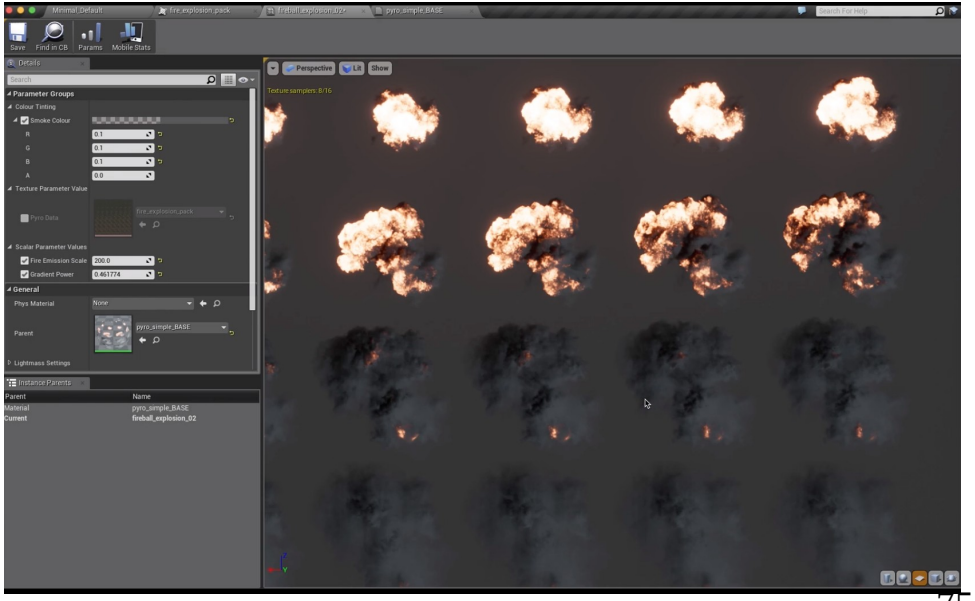
Procedural texture: physical simulation

- Perlin and Cellular textures are mathematical models and phenomenological models for representing natural phenomena
- Physically based simulation can also be used to create textures
  - Reaction-diffusion model
  - Fluid simulation to texture





Raction-diffusion model: https://www.karlsims.com/rd.html

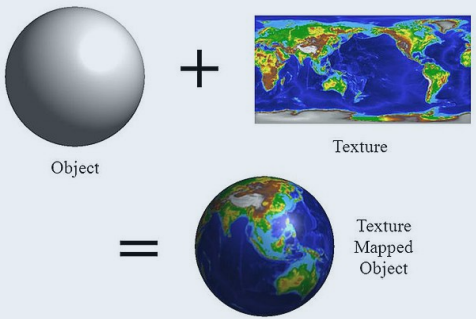


Fluid simulations:
https://www.sidefx.com/ja/tutorials/channel-packing-pyro-data-using-the-texture-sheets-rop/

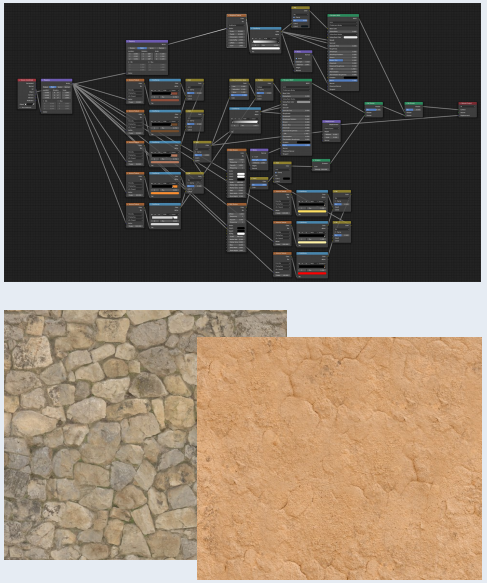
Procedural textures: Antialiasing*

- Procedural texturing is much more popular in offline applications while in real-time rendering image textures are far more common
  - GPUs are extremely efficient with image textures for which antialiasing is also easier
- Antialiasing can be hard for procedural textures
- Helping factor is that "inside information" about the texture is available:
  - For example, if procedural texture is built on summing noise functions, then frequencies which would cause aliasing can be omitted

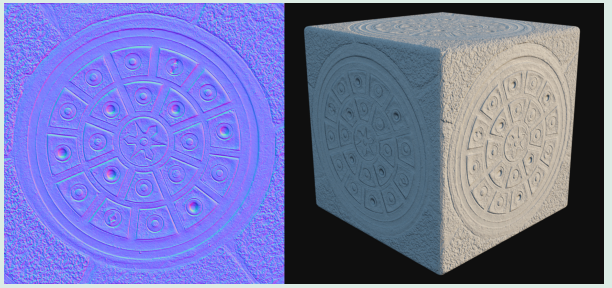
Learning objectives



How to apply textures on objects → **texturing pipeline**



Texturing approaches → **image and procedural**

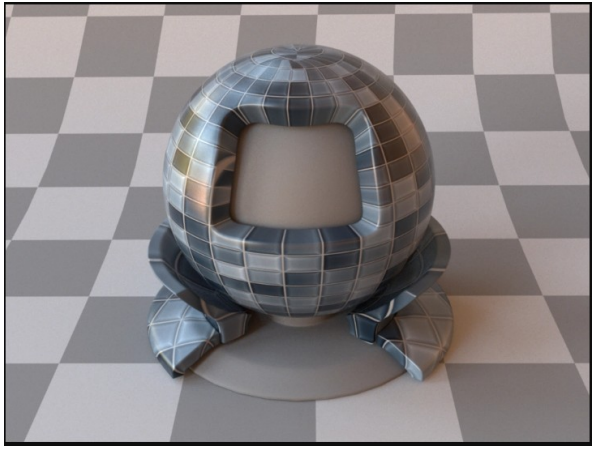


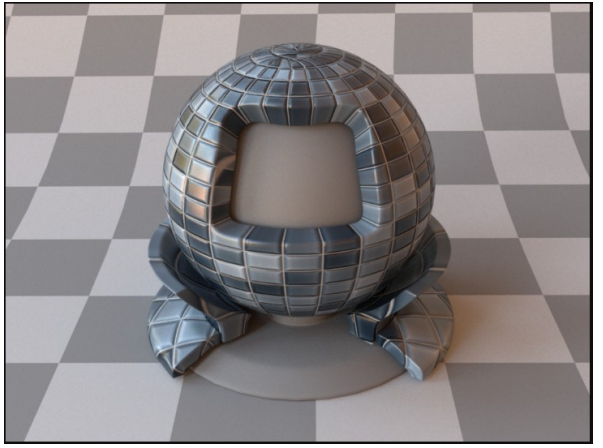
How to use texture for **material modeling** → variation of scattering function parameters and small scale geometry

Textures and material modeling

Usage of texture values

- Texture is used for **varying material parameters** over surface:
  - **Scattering function parameters**: color, reflectance, roughness, etc.
  - **Small scale surface geometry:** bumps, irregularities, etc.

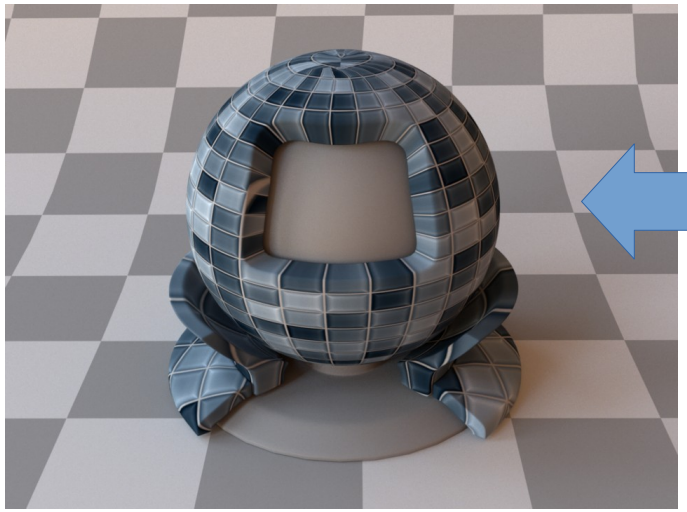




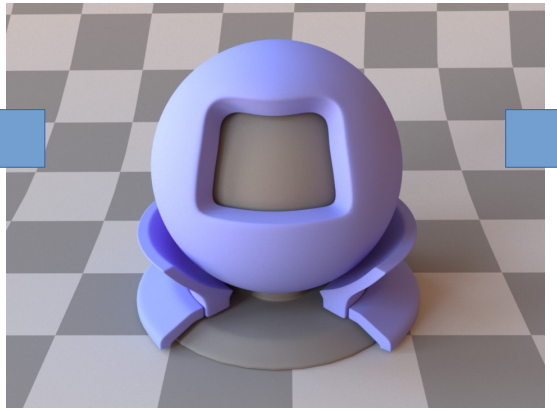
https://mitsuba2.readthedocs.io/en/latest/generated/plugins.html#textures

Varying scattering function parameters

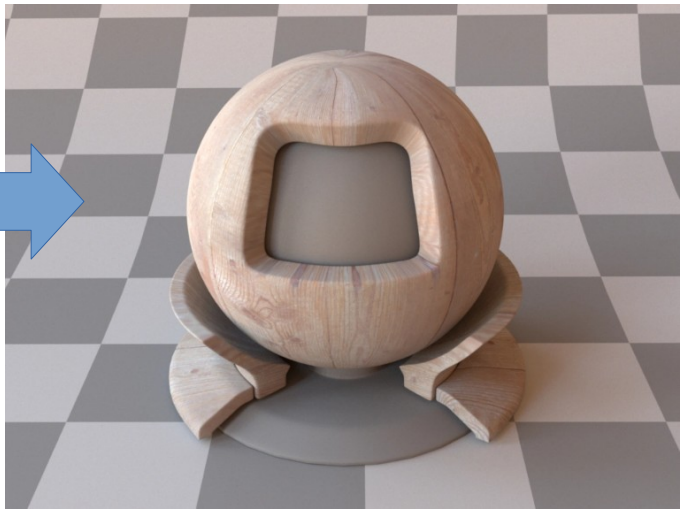
- **Texture can be used to vary scattering function parameters over surface**
- Texture variation of color, reflectance, roughness, etc.



Diffuse scattering model (Lambertian) with varying albedo (color) parameter using texture



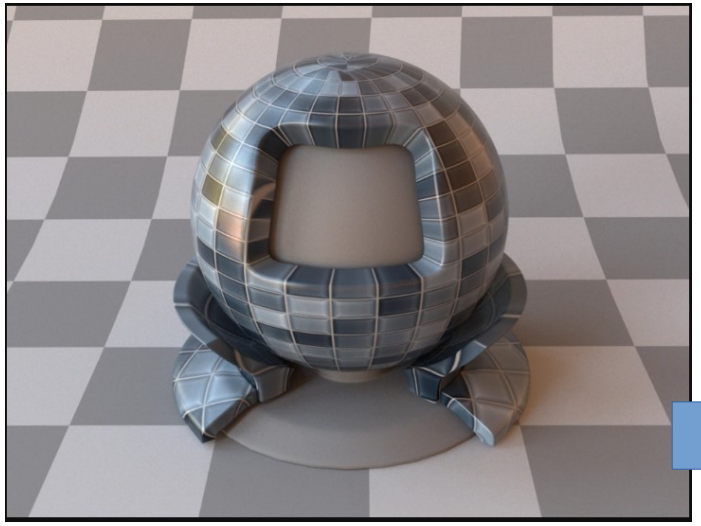
Diffuse scattering model (Lambertian) with constant albedo (color) parameter



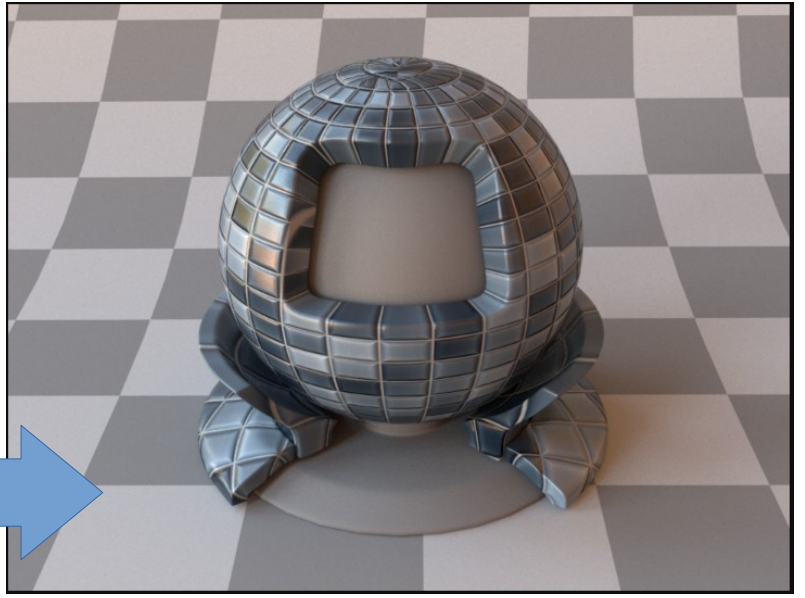
Diffuse scattering model (Lambertian) with varying albedo (color) parameter using texture

Varying small scale surface geometry

- Texture can be used to represent small-scale surface details/geometry.
- Texture is employed in shading step during rendering to give more richer 3D appearance than just color
  - Surface doesn't appear smooth



Glossy scattering model with **varying color parameter using texture**

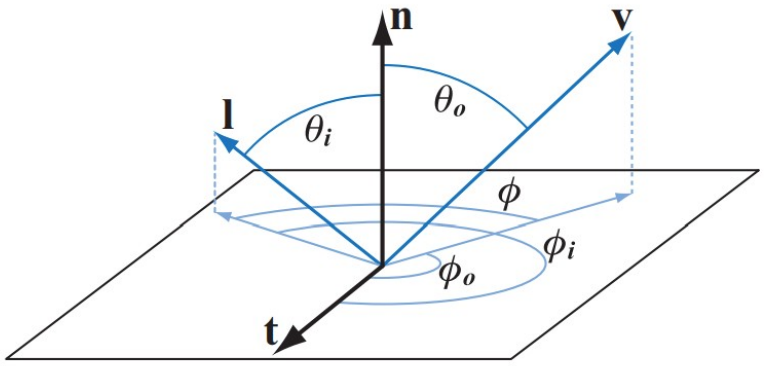


Glossy scattering model with **varying color and surface details using texture**

https://mitsuba2.readthedocs.io/en/latest/generated/plugins.html#textures

Small scale surface geometry

- Texture values are used for perturbing surface or surface normal
  - Perturbing surface or surface normal causes perturbing of BSDF basis and results in variation in light reflection



Besides incoming and outgoing direction, scattering function (BRDF) depends on surface normal.

https://learnopengl.com/Advanced-Lighting/Normal-Mapping

Small-scale geometry: scale of modeling

Macro-features: cover many pixels.
- Represented by **model shape** (e.g., polygonal mesh).
- Example: pillars

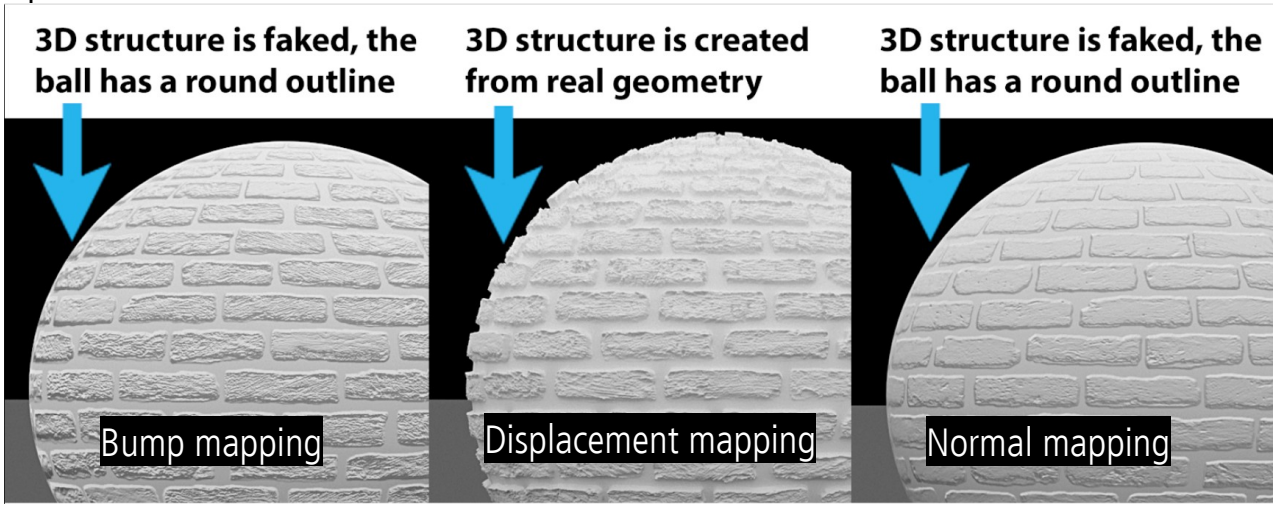
Micro-features
- Described with **scattering function and texture**
- Simulates the microscopic response of surface geometry and substance
- Example: glossy and diffuse effects are modeled on micro-scale

https://www.pbr-book.org/3ed-2018/Materials/Bump_Mapping

Meso-features: cover few pixels
- Everything between two scales.
- Represents detail that is too expensive for shape representation (e.g., polygonal mesh) but large enough to be observable.
- Example: brick bumps and dents
- For this scale family of **bump mapping methods** is used

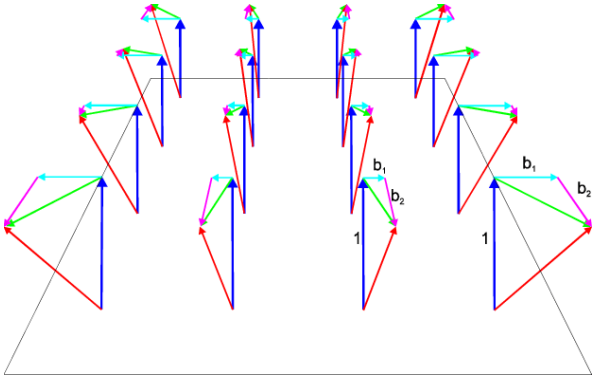
Small-scale geometry variation

- Methods which do not introduce additional geometrical information. Only approximation to achieve desired surface details:
  - **Bump, normal mapping**
  - **Parallax mapping**
- Methods which introduce/require additional geometrical information. Give more realistic surface appearance but are computationally expensive:
  - **Displacement mapping**



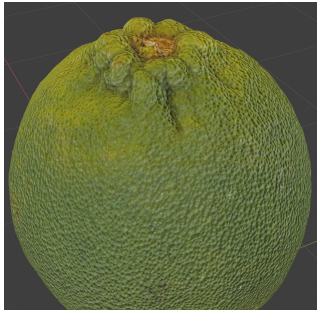
Bump mapping

- Idea (**Blinn method**):
  - **Surface will appear to have small scale details if normal is slightly perturbed**
  - Geometry is not changed!
- Information for perturbing surface normal can be encoded in image texture as color or procedural texture.
- Original bump mapping – **offset vector bump map**
  - Two signed values are store per texel in image texture – the amount of normal to be varied in `u` an `v` image axis
  - Those values are used to scale vectors perpendicular to normal (tangent and bitangent) and are added to normal to change direction



https://www.researchgate.net/publication/228890410_Antialiasing_of_Bump_Maps

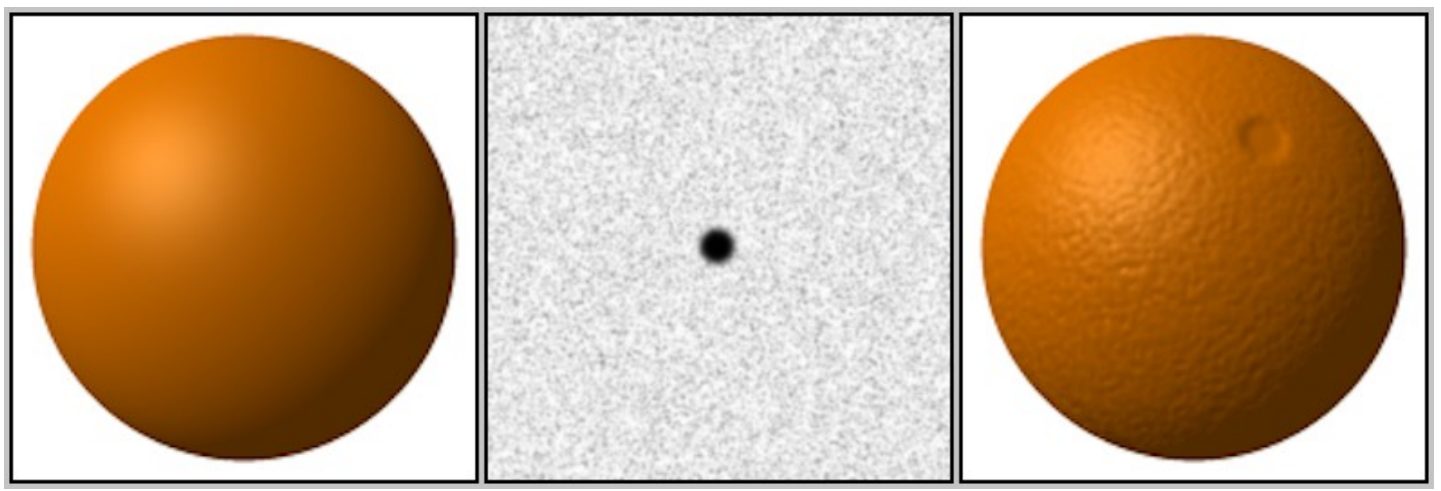
Original surface normal



Perturbed surface normal

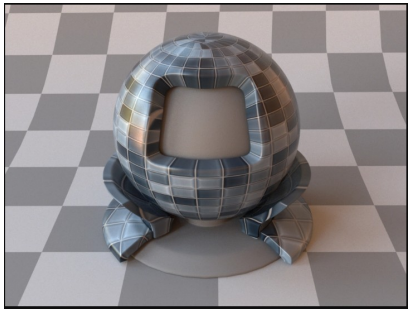
Bump mapping

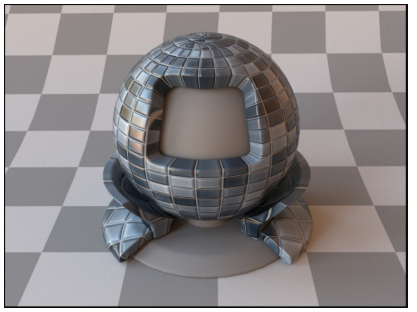
- Modern approach is to use **heightfield** to modify surface normal direction
- **Image texture contains one floating point number per texel representing height**
  - Brigher color: higher values
- Height values are used to derive u and v signed values similar to those in original method – this is done by taking differences between neighboring columns are rows to obtain u and v (e.g., Sobel filter)



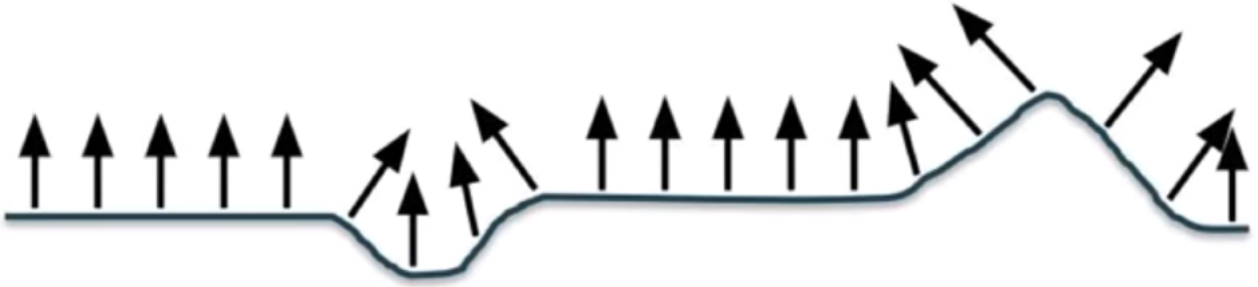
https://en.wikipedia.org/wiki/Bump_mapping

Bump mapping





- Actual geometrical details are simulated by just perturbing surface normals
- No additional geometry is added!

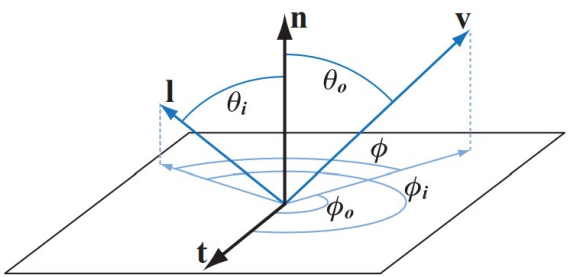


Actual surface with actual geometrical details and corresponding normals.

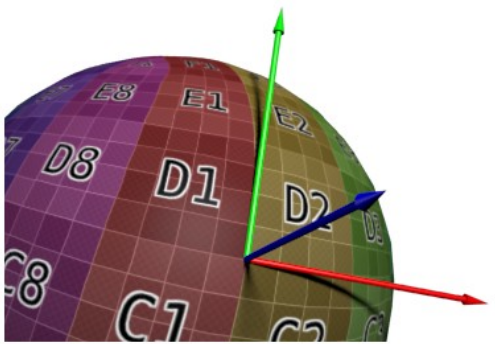
Simulated geometrical details perturbing normals

Bump mapping: basis modification

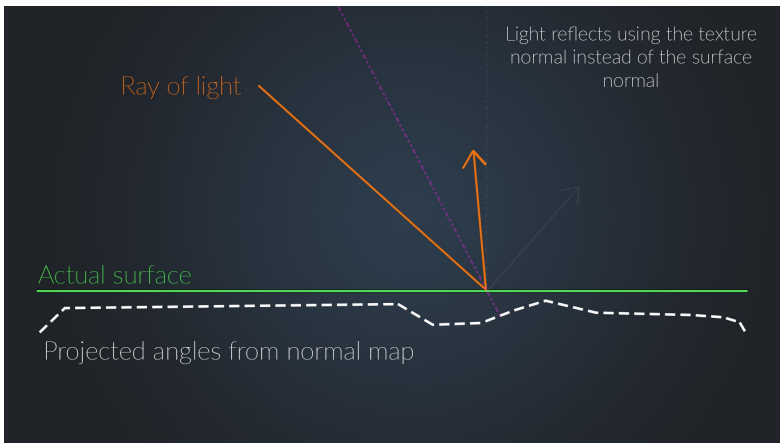
- **Perturbing surface normal must be done with respect to some frame of reference: tangent frame basis**
- **Scattering model is evaluated on basis created from surface normal**
  - Perturbing the basis by perturbing the normal we achieve different incoming light and viewing directions – just like the surface would contain small bumps and dents
  - Also, directions of light and viewing are often transformed in this space – so they are in the same space as surface normal which is needed for correct for evaluation
  - Tangent frame is used to determine orientation of **anisotropic** surfaces



BRDF basis



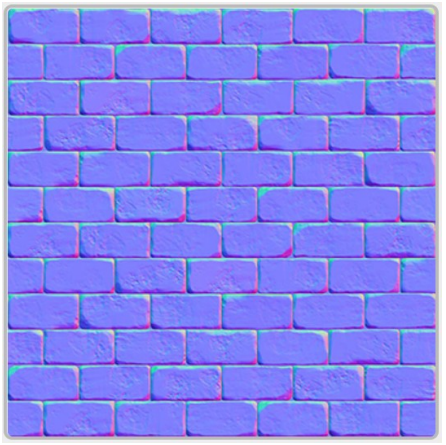
Tangent frame:
http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/

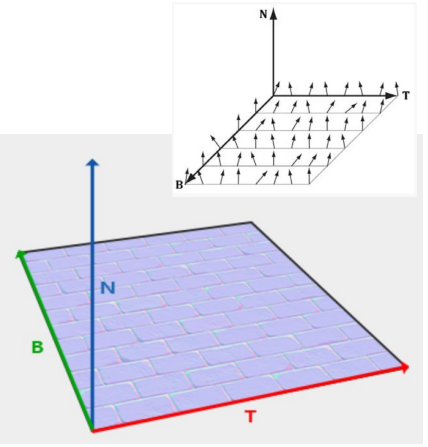


https://cgcookie.com/posts/normal-vs-displacement-mapping-why-games-use-normals

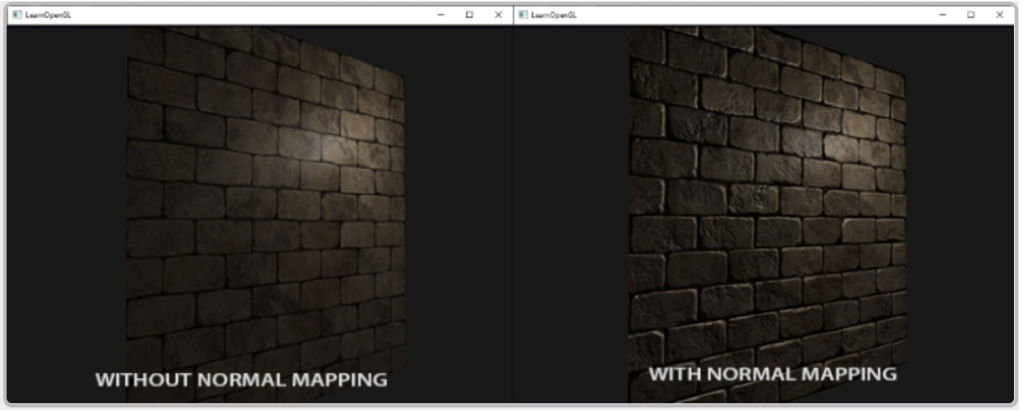
Normal mapping

- Normal vectors are directly stored into image texture or evaluated from procedural texture.
- Normal map image texture encodes `(x,y,z)` vector with values in `[−1,1]` to (R,G,B) image texture
  - For PNG images, 8bit unsigned int is used. Therefore, conversion [-1,1] → [0,255] is performed
- Normal map are often defined in **tangent space**
  - Color of normal map is determined with `(x,y,z)` directions of encoded vectors





http://ycpcs.github.io/cs470-fall2014/labs/lab12-2.html



https://learnopengl.com/Advanced-Lighting/Normal-Mapping

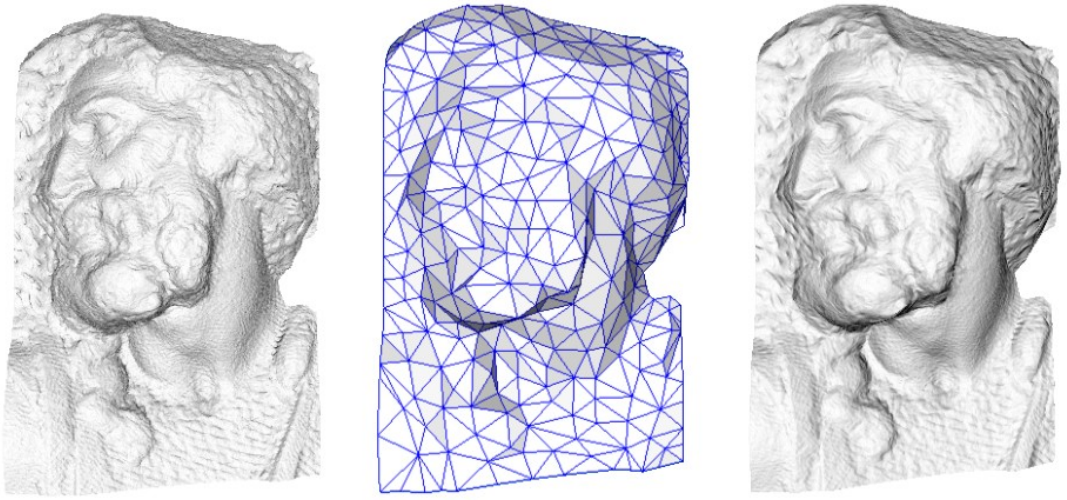
Normal map types*

- **Tangent space:** based on tangent direction of face. Normal map can be then maximally reused for different objects, transformations and deformations.
  - Blue: normal direction
  - Red: left/right tangent direction
  - Green: up/down tangent direction
- **Object space**: based on entire object rather than each face individually
  - Faster to compute, tied to to particular model which can not be deformed
- **World space**: based on global coordinates
  - Object must not be rotated
  - Useful for large, static environment objects

https://cgcookie.com/posts/normal-vs-displacement-mapping-why-games-use-normals

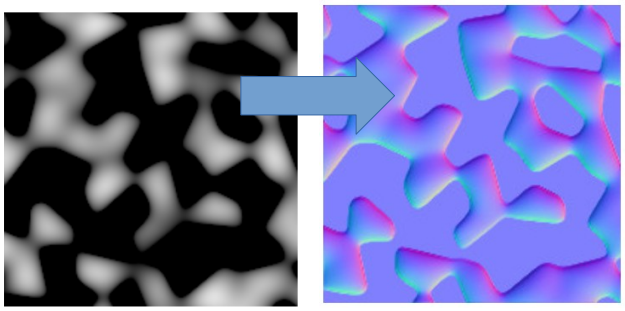
Normal mapping

- Advantage of normal (and bump) mapping is that complex geometry details can be represented in cheap way.

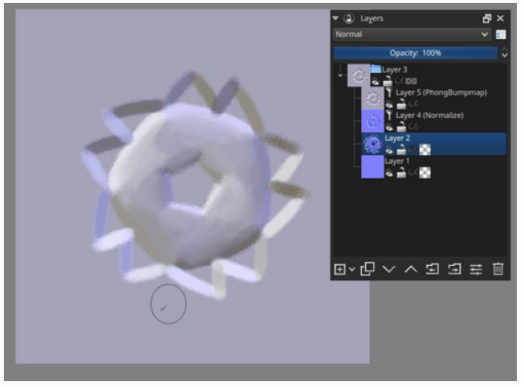


https://helpx.adobe.com/substance-3d-painter/using/baking.html

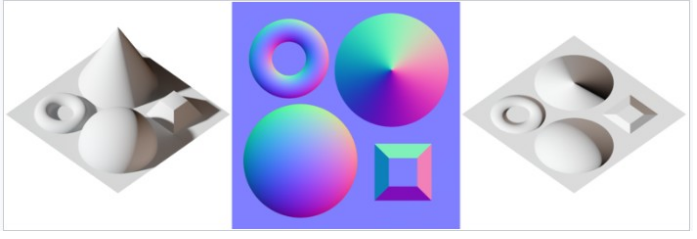
Creating bump/normal maps



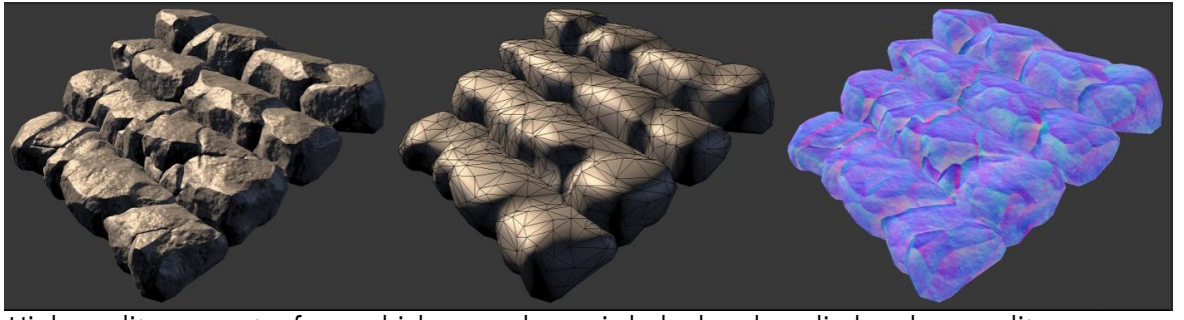
Height maps can be created by scanning, hand-painting, from photographs, etc.
Normal maps can be created from height maps:
https://docs.gimp.org/2.10/en/gimp-filter-normal-map.html



Normal maps can be created by hand-painting
https://docs.krita.org/en/reference_manual/brushes/brush_engines/tangen_normal_brush_engine.html



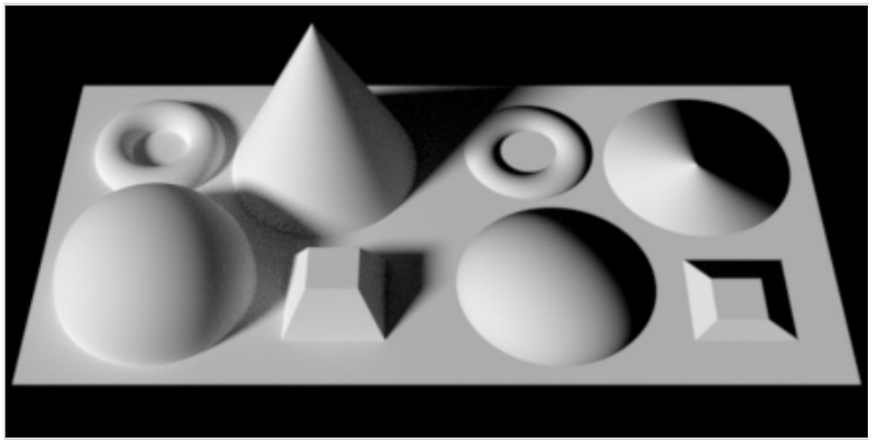
Normal map can be baked from geometry:
https://en.wikipedia.org/wiki/Normal_mapping
http://wiki.polycount.com/wiki/Normal_map



High quality geometry from which normal map is baked and applied on low quality geometry to achieve computationally cheap high quality appearance.

Bump and Normal mapping: practical notes

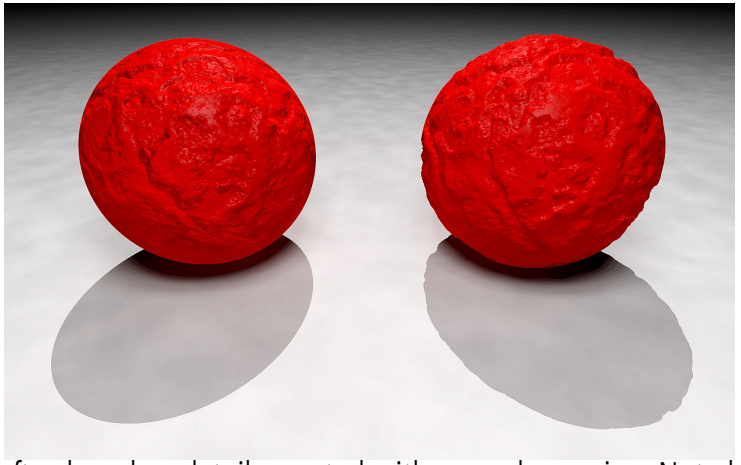
- **Normal/bump mapping, is not creating additional geometry**
  - **Surface details created that way can not cast shadow on the surface**
  - To compute shadows for normal/bump mapping **horizon mapping** is used.



Real geometry (left) and geometry created with normal mapping (right). Note how cone and sphere are not casting shadow in case of normal mapping.

Bump and Normal mapping: practical notes

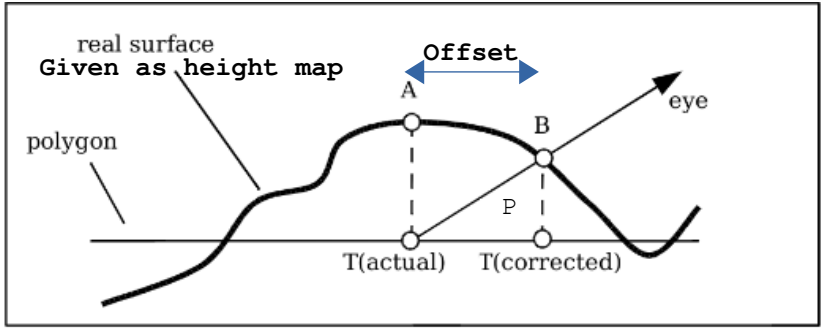
- **Normal/bump mapping, is not creating additional geometry**
  - **Shilouettes of bump/normal mapped surface are smooth.**
- Relationship between normal and shaded color is not linear (e.g., normal map applied to specular scattering model). Therefore, filtering of normal maps for **anti-aliasing** requires additional work.



Left sphere has details created with normal mapping. Note how silhouette is smooth compared to the right sphere with actual geometry.

Parallax mapping

- **Parallax: positions of geometrical details on object move relative to one another as the observer moves and occlude (block) each other from view**
- Parallax mapping: approximation technique which is **offsetting texture coordinates based on height map**
  - What should have be seen in pixel by examining the height of what was found to be visible

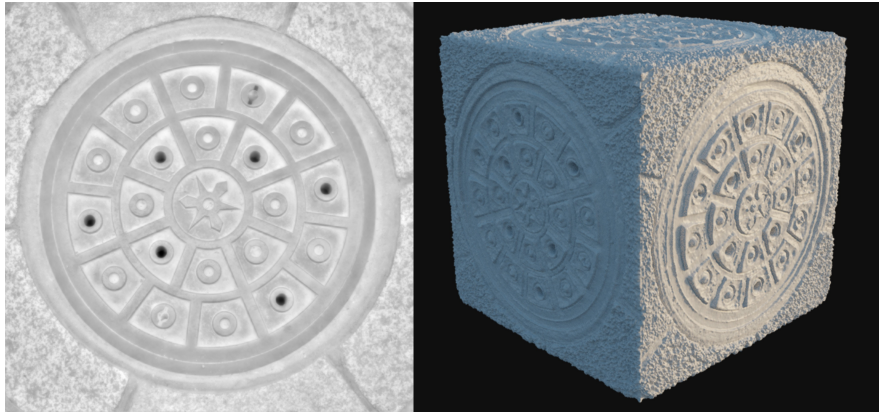


When viewing surface at a given point, height value is retrieved at that location and used to shift the texture coordinate to retrieve a different part of the surface. Amount of shift is based on retrieved height and angle of the eye to the surface

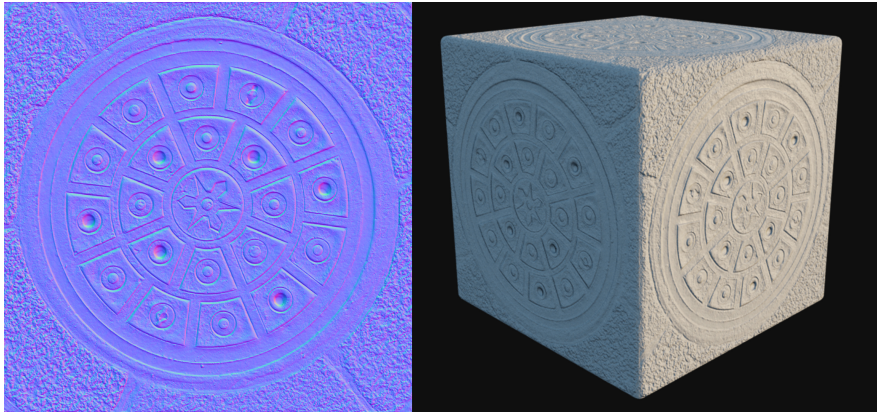
https://github.com/marcusstenbeck/tncg14-parallax-mapping/tree/master/documents

Parallax mapping

- Also known as virtual displacement mapping
- Enhancement: **parallax occlusion mapping or relief mapping**: ray-marching is used along view vector until approximate intersection point on height-field texture



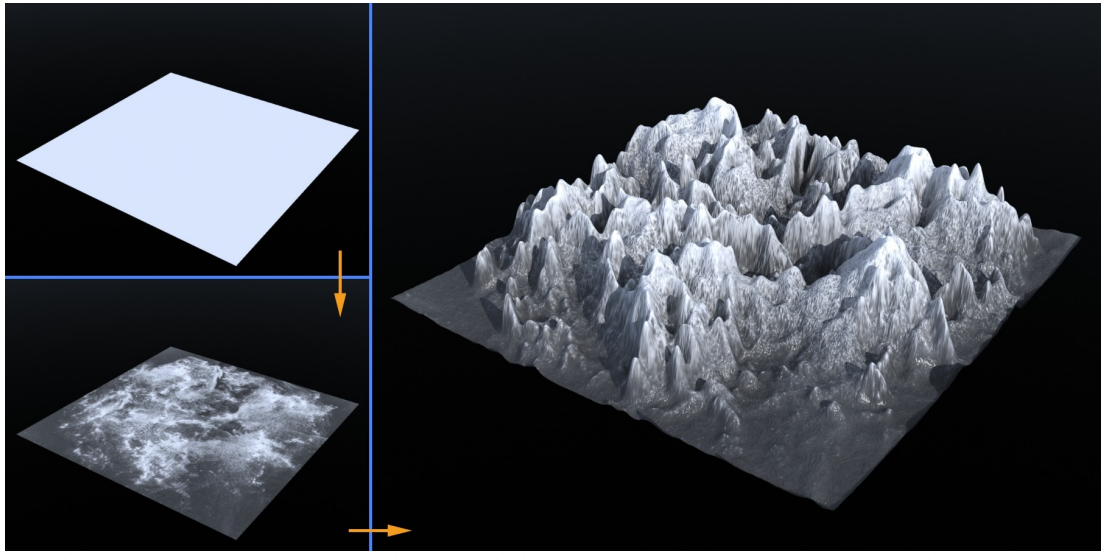
Parallax mapping. Note how details of surface are occluding each other and creating silhouettes



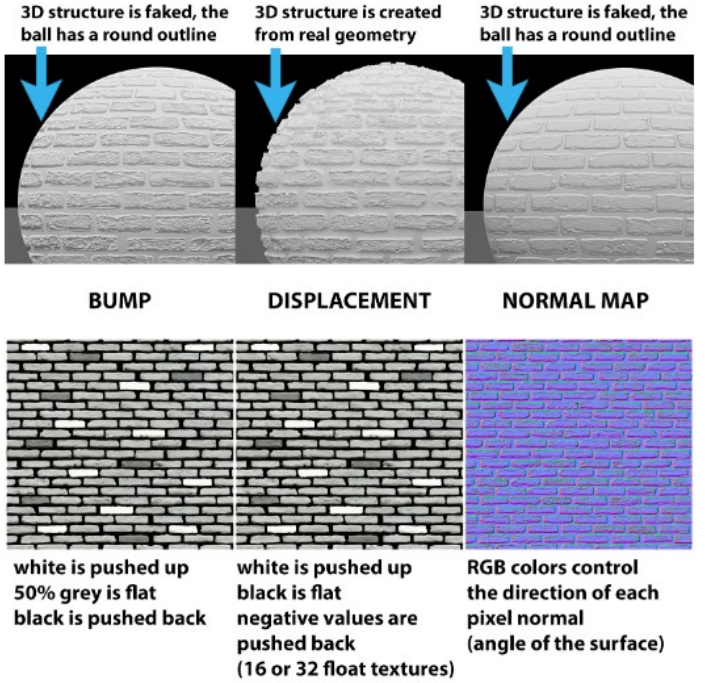
Normal mapping. Note how details of surface look flat compared to parallax mapping.

Displacement mapping

- This method changes the locations of actual geometry vertices in a mesh using height texture information.
- Most computationally expensive technique since requires fine geometry
  - Geometry must be subdivided prior or during rendering before displacement mapping
  - Modern GPUs support on-the-fly tessellation (tessellation shader) for subdivision of base geometry



Displacement mapping is good for modeling larger geometry features such as terrains.https://ca.wikipedia.org/wiki/Displacement_mapping



https://garagefarm.net/blog/how-displacement-maps-work-and-how-to-optimize-them-in-v-ray-part-1

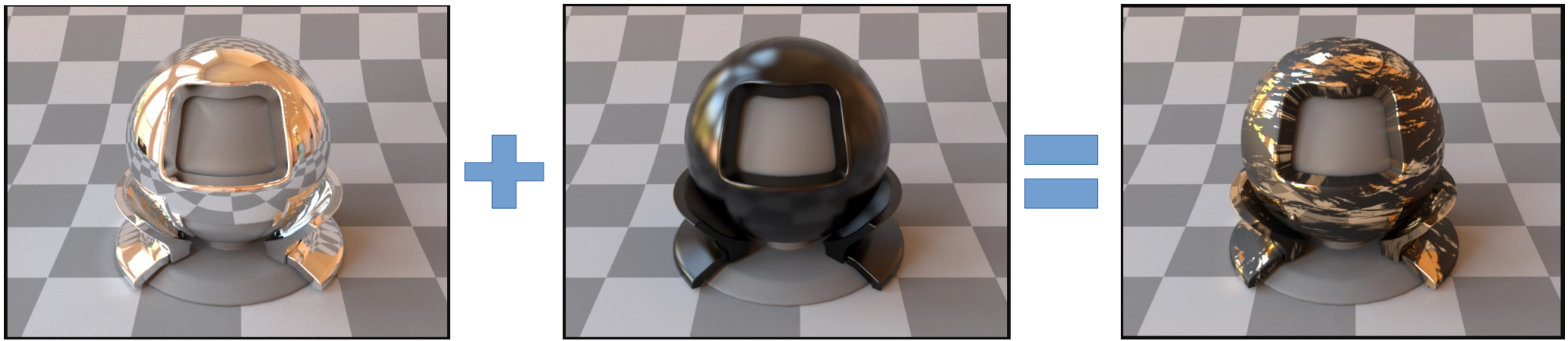
Note on small scale geometry

- Note that texture is often used to represent small scale geometry which is expensive to represent and render directly.
- However, high quality rendering techniques exist which enable modeling and rendering of high quality geometry without need for normal, bump, parallax or displacement mapping.

Unreal Nanite technology for displaying large and highly detailed meshes.
https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/

Distributing materials using texture

- Texture can be used to **distribute different materials** over surface
  - Texture can be used as a "mask" which determines which scattering function and shading calculations will be performed where



Texture can be used to mix different scattering functions over surface.
https://mitsuba2.readthedocs.io/en/latest/generated/plugins.html#textures

Storing and transferring materials

- Similarly to standardized mesh storage and transfer, material standards are defined.
- Material standards for storage and transfer:
  - MaterialX: https://materialx.org/
- Standards for storing the whole scene including the material:
  - GLTF: https://github.com/KhronosGroup/glTF
  - USD: https://graphics.pixar.com/usd/release/intro.html

Exploring textures

- Industry standard texturing tools:
  - https://www.adobe.com/products/substance3d-painter.html
  - https://www.adobe.com/products/substance3d-designer.html
  - https://substance3d.adobe.com/assets
- Procedural texturing
  - https://thebookofshaders.com/
  - https://www.shadertoy.com/
  - Texturing and Modeling: A Procedural Approach: https://dl.acm.org/doi/10.5555/572337
- More on texture: https://www.realtimerendering.com/#texture

Summary

- Questions: https://github.com/lorentzo/IntroductionToComputerGraphics/tree/main/lectures/8_texture

Literature

- https://github.com/lorentzo/IntroductionToComputerGraphics