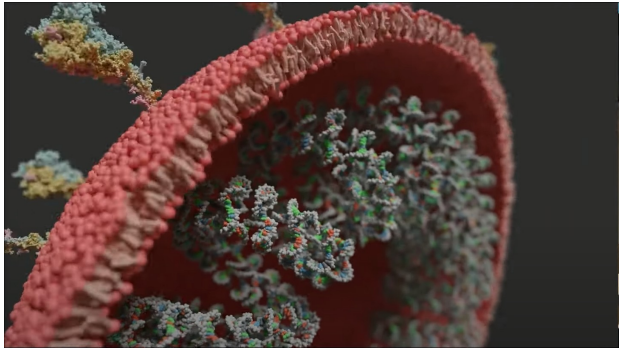
Computer image generation overview

Goal of Computer graphics

- Any interactive virtual environment, animation, application, development and research in computer graphics → **synthesizing images**

Graphics and computer games
https://www.rockstargames.com/reddeadredemption2/

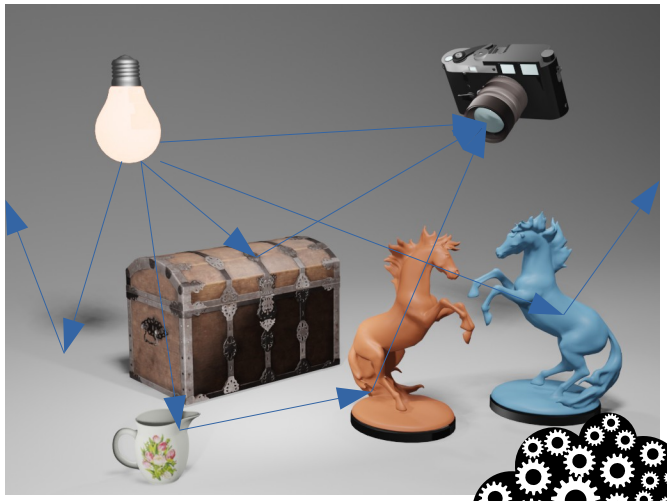
Graphics and animated film
https://www.pixar.com/soul



Graphics and scientific visualization
https://www.youtube.com/watch?v=adhTmwYwOiA&ab_channel=Blender

Image synthesis: intuition

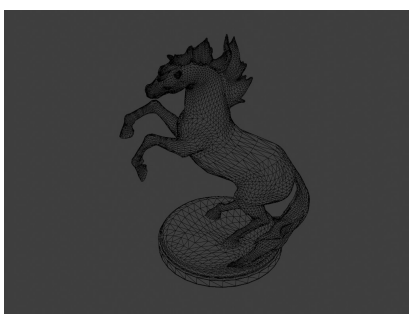
- Analogy with photography:
  - **Light source** emits light
  - **Light travels** through the space, **interacts with objects**
  - Small amount of light reflects into **camera → image**



https://polyhaven.com/models

Fundamental image synthesis questions

- How to represent real world objects, lights and cameras in a computer?
  - 3D scene modeling
- How to generate image based on computer representation of those elements?
  - Rendering and image

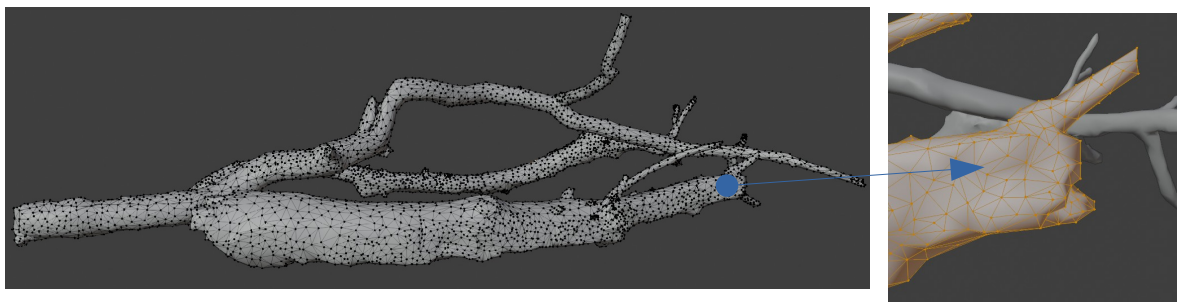


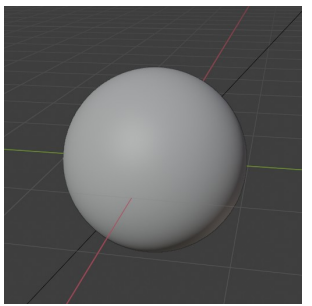
Art of computer graphics

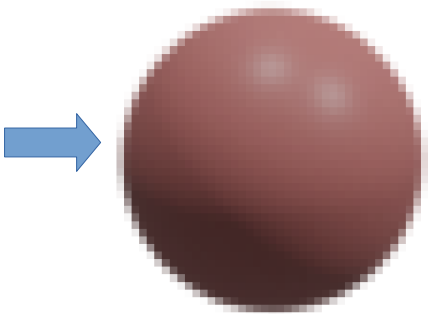
- **Decompose real world objects and phenomena into separate models and processes** which can be modeled **and then again combined into virtual world**.
  - All of those parts are deeply intertwined and one doesn't makes sense without another.
  - Example: without light we can't see objects and without objects we can't see light.

Simplification of a real world

- Computer graphics problems are solved using **simulation and approximation.**
  - The real world objects and phenomena is simulated and approximated to achieve desired outcomes for specific application.
- Description of a real world in a 3D scene must be **discretized** for display
  - Real world and 3D scene are continuous, synthesizing images means creating discrete image: array of pixels





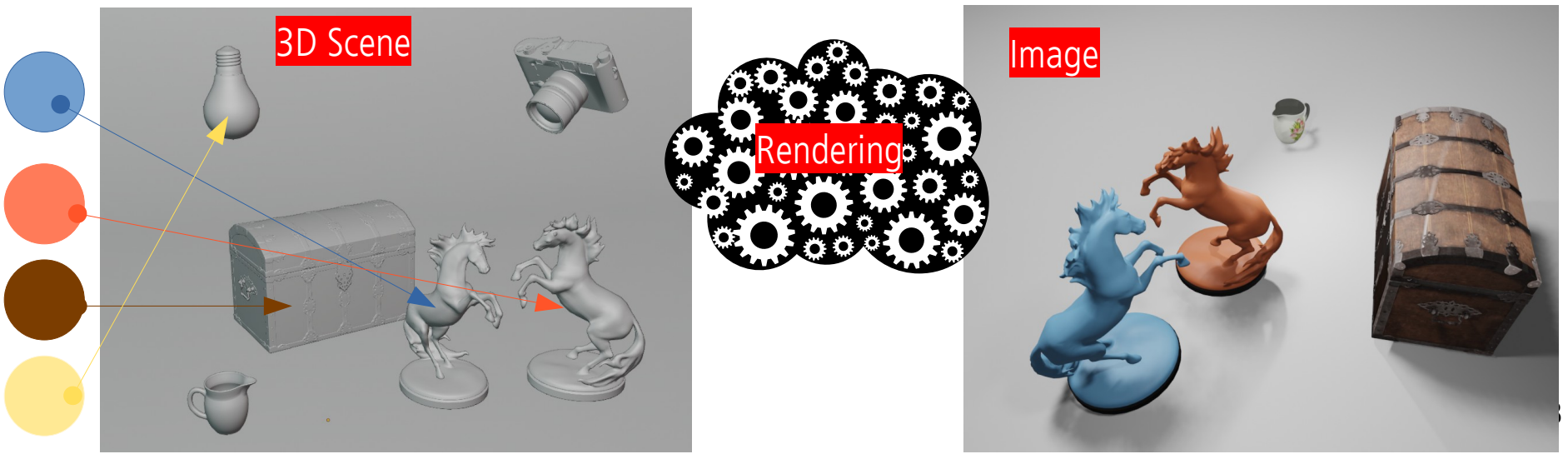


3D scene

Image

Pillars of computer graphics

- To synthesize an image:
  - We need a description of a **3D scene** which contains objects, lights and cameras
  - A way to transform a 3D scene into 2D array of pixels – **rendering**
  - A way of storing 2D array of pixels for raster display – **image**



Modeling and rendering

- Computer graphics is concerned with:
  - Techniques and methods for **modeling** of **3D scene**: what will be rendered
  - Development of **rendering** algorithms for creating image out of a 3D scene
  - Storing and displaying **images**

Modeling and rendering are intertwined

- Rendering process must “understand” how 3D scene is represented
- Modeling of 3D scene relies on representations which are understood by rendering algorithm

3D Scene Modeling

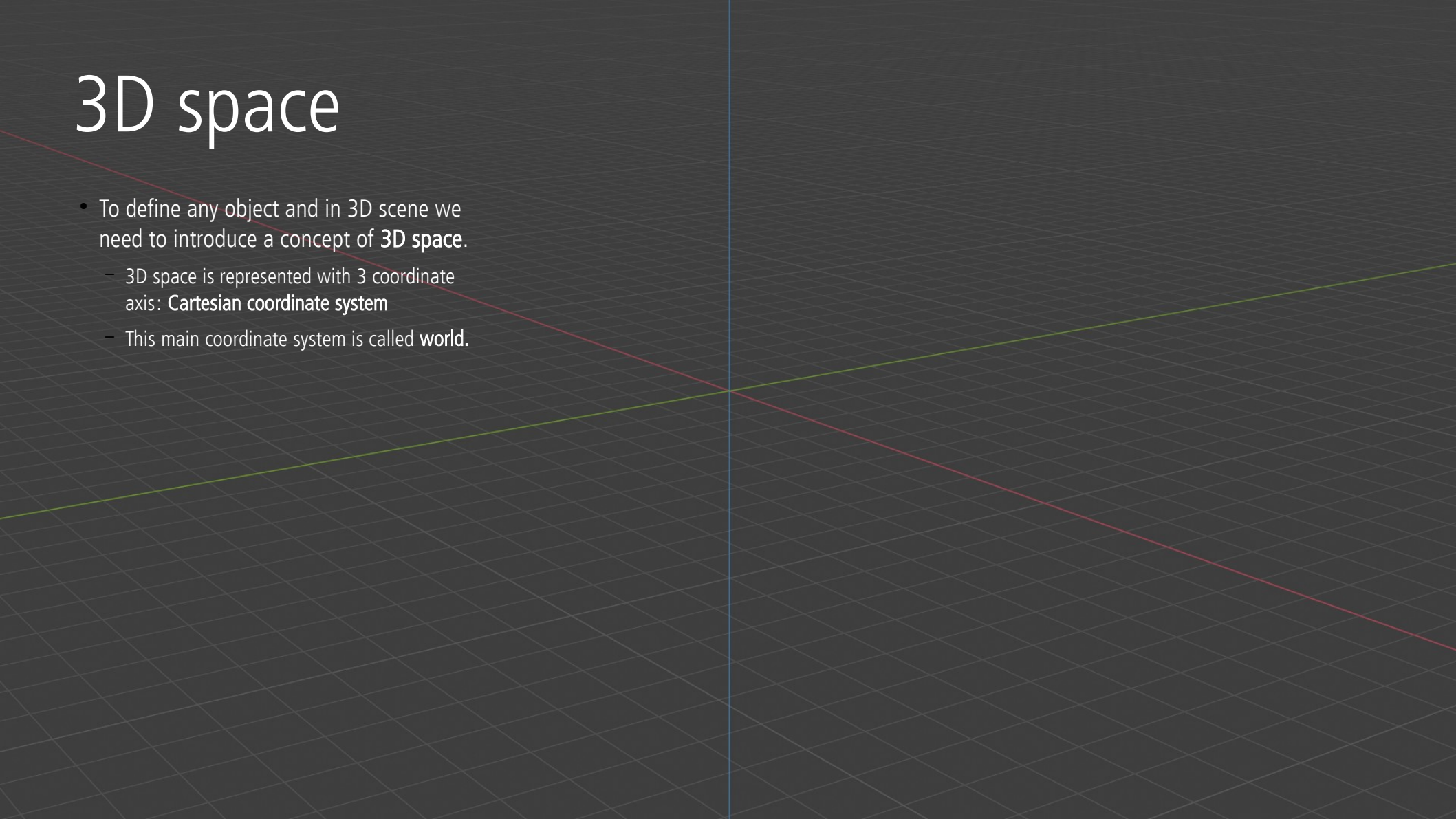
A start of image generation journey: answering what we render

3D scene modeling

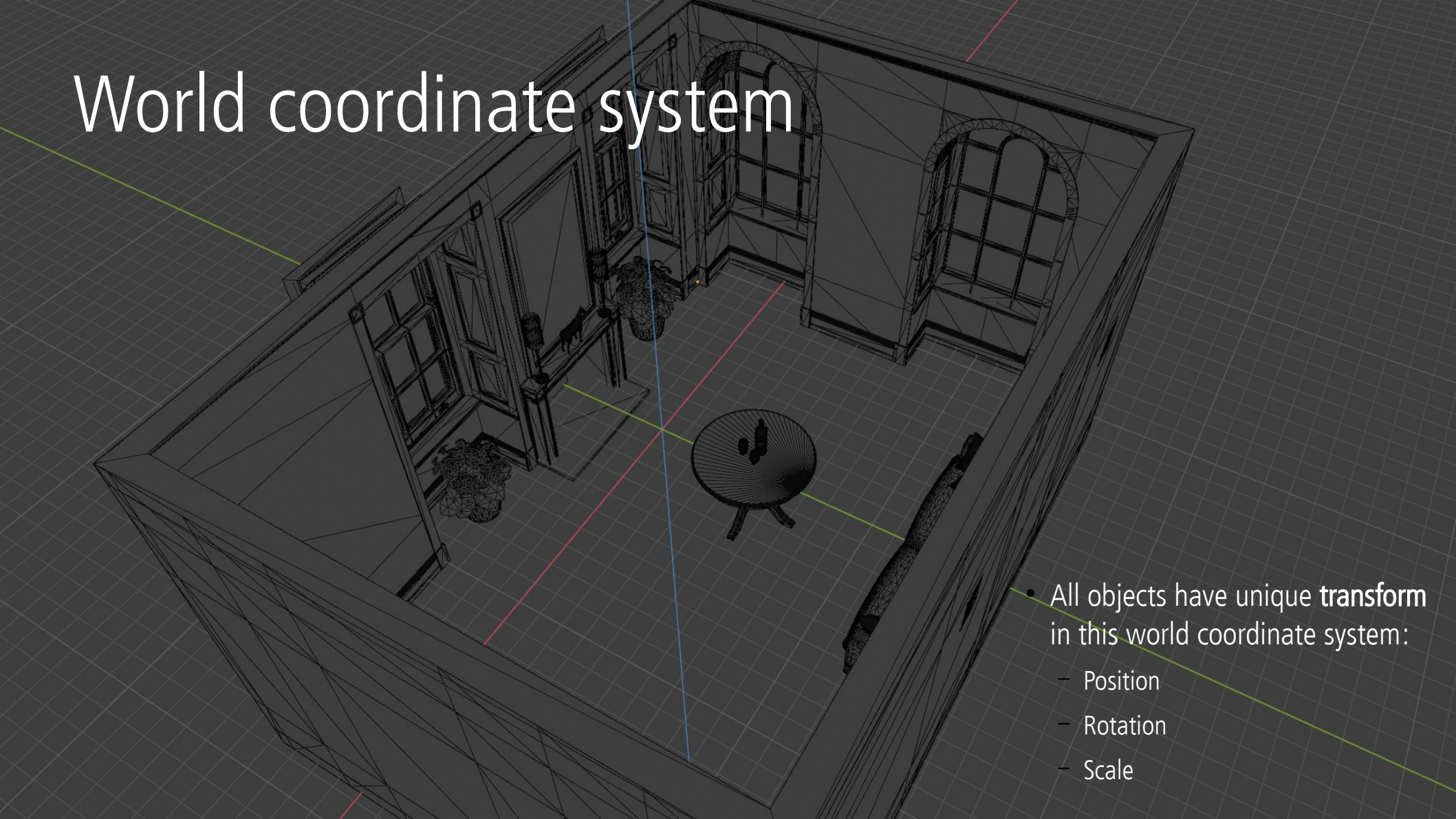
- In order to generate an **image** we need something to **render** → a **3D scene:**
  - **Objects**: what we want to visualize
  - **Cameras**: from where and how are we viewing the objects
  - **Light sources**: enable us to see objects → light reflected to camera

3D space

- To define any object and in 3D scene we need to introduce a concept of **3D space**.
  - 3D space is represented with 3 coordinate axis: **Cartesian coordinate system**
  - This main coordinate system is called **world.**



World coordinate system



- All objects have unique **transform** in this world coordinate system:
  - Position
  - Rotation
  - Scale

Objects in 3D scene

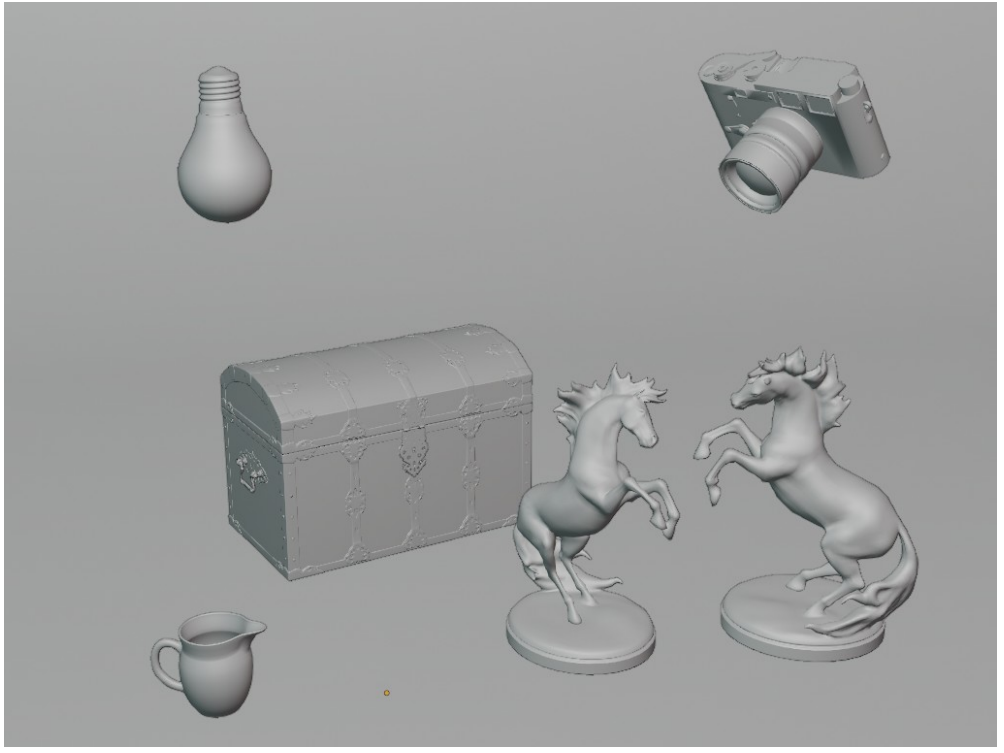
- Modeling 3D scene object is separated into modeling:
  - Object shape
  - Object material

Shape

Shape and material

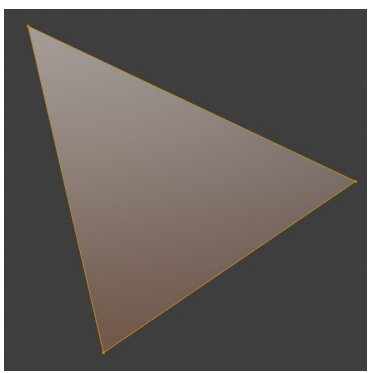
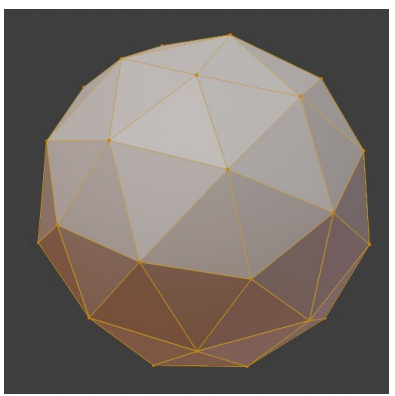
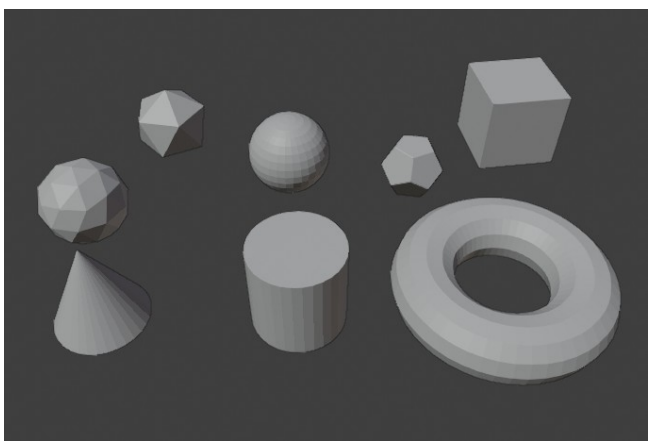
Object shape

- Object shape is needed for:
  - Defining object form, size, etc.
  - Placing object correctly in the scene with respect to other objects
  - Determining which objects are occluded and areas into which shadow is cast by an object
  - Determining visibility

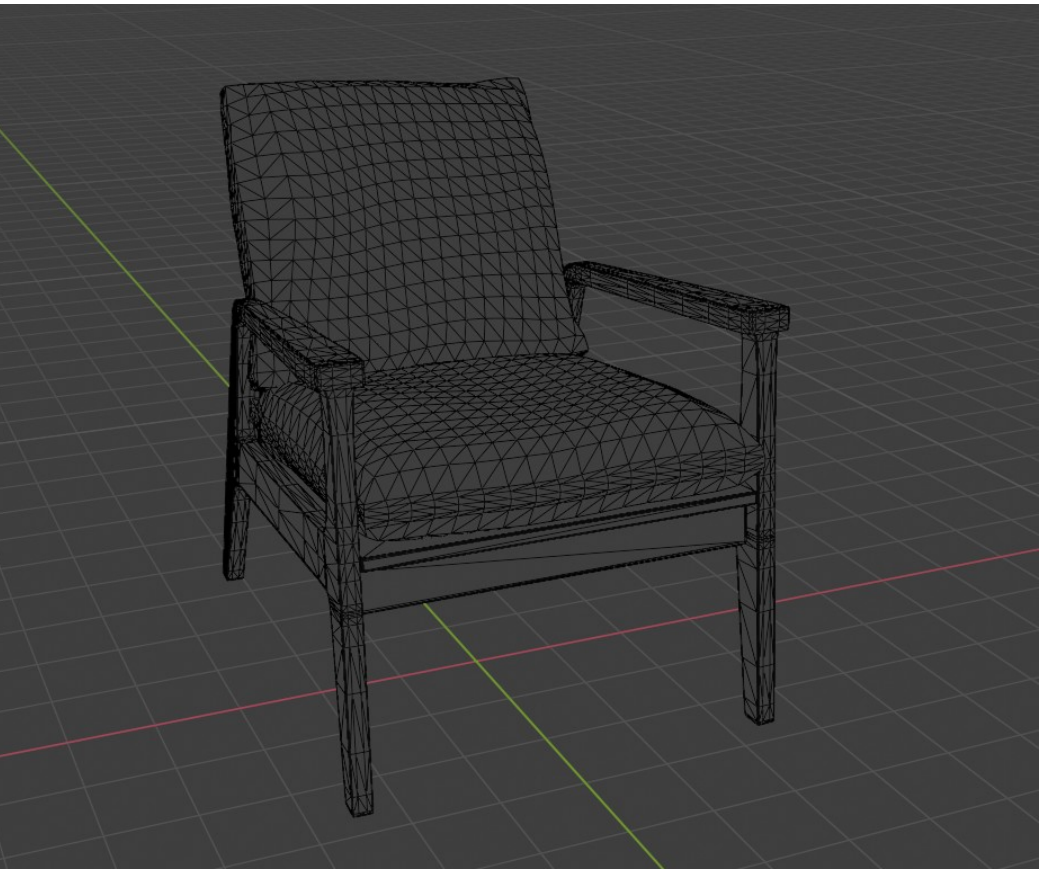


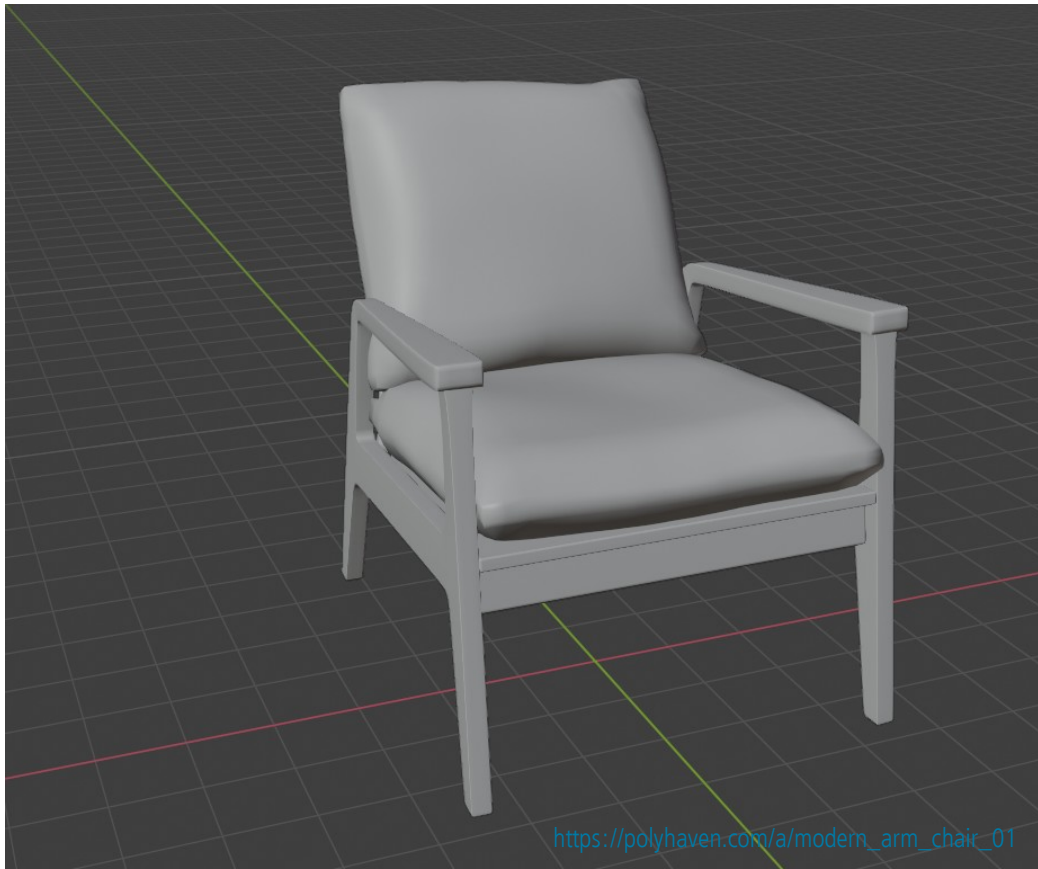
Shape representation

- We start by defining **points** (x,y,z) in 3D space.
- To define a **surface**, simplest way is to connect points to form a **polygon**.
  - Most commonly used polygons are **triangles.**

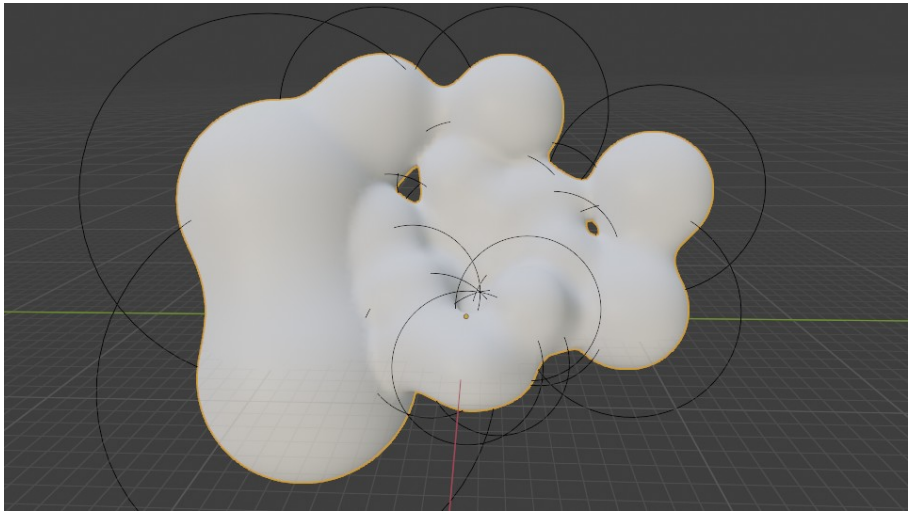

Triangle shape representation



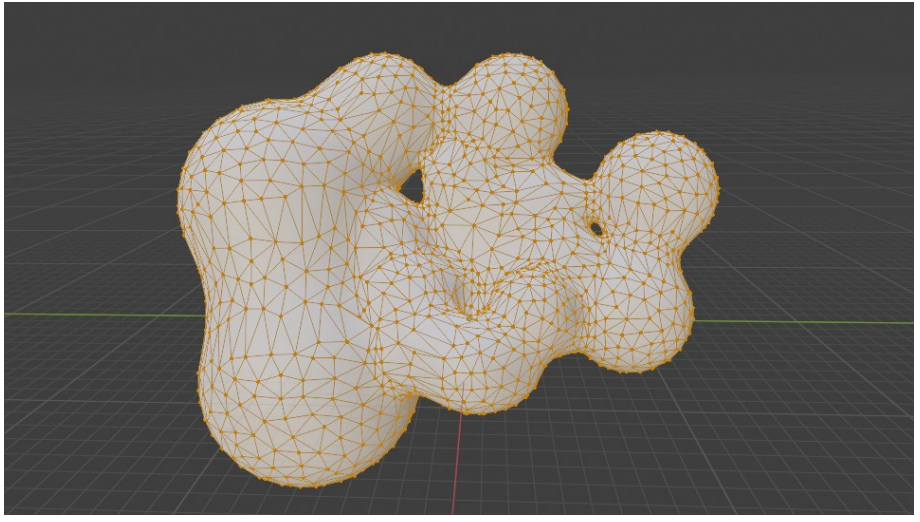


Why triangles?

- Are objects that we have seen created using triangles?
  - NO!
- Modeling is often done using different shape representations
  - It is very often that different shape representations are turned into triangles before/during rendering stage - using **triangulation**.
- Triangle is most widely used **primitive** for surface shape representation
  - **Simple**
  - **Efficient for storage and transfer**
  - **Efficient for rendering**



Triangulation



Other shape representations

- Various representations for 3D models exists.
  - Some are better for **modeling**
  - Other are better for **rendering** (triangulated mesh).

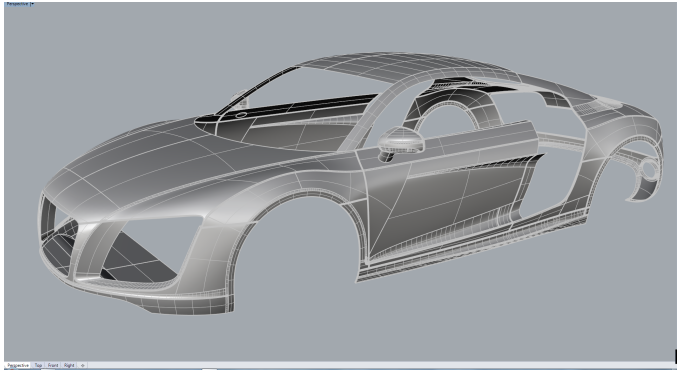
Shape representations for modeling

- In professional software, various shape representations for efficient modeling exist.

Hair modeling using parametric curves:
https://developer.nvidia.com/gpugems/gpugems2/part-iii-high-quality-rendering/chapter-23-hair-animation-and-rendering-nalu-demo

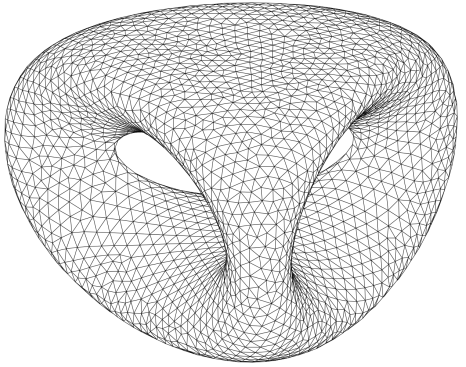
Implicit surface representation:
https://www.shadertoy.com/view/ld3Gz2

Parametric surfaces:
https://forums.cgsociety.org/t/audi-r8-v10-2010-nurbs-model-with-rhino-3d/1761843/3



Shape representations for rendering

- Almost always prior rendering, different shape representations are converted to triangulated mesh for efficient rendering.
  - One representation for rendering makes rendering process highly efficient
  - Graphics cards (graphics processing units, GPU) is highly efficient for processing triangles
  - Lot of research and hardware development was focused onto efficient rendering of triangles*.



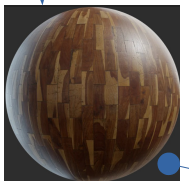
* It is easy to imagine that different rendering primitive is used. But due to historical events this turned out to be a triangle. What is important to note is that mapping of various representations to triangle mesh is possible and thus it is not important which is the rendering primitive as long as it supports various representations.

Object material

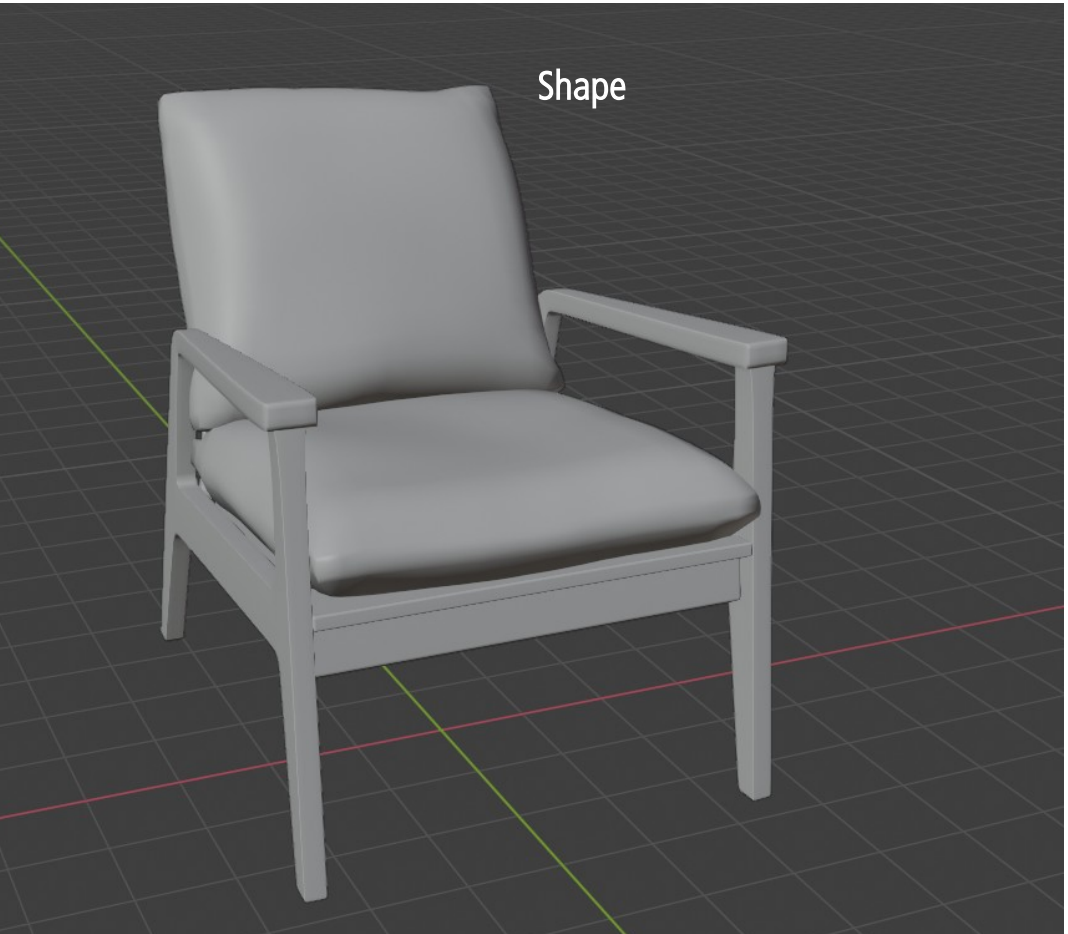
- Shape defines what is visible, material defines how it looks.

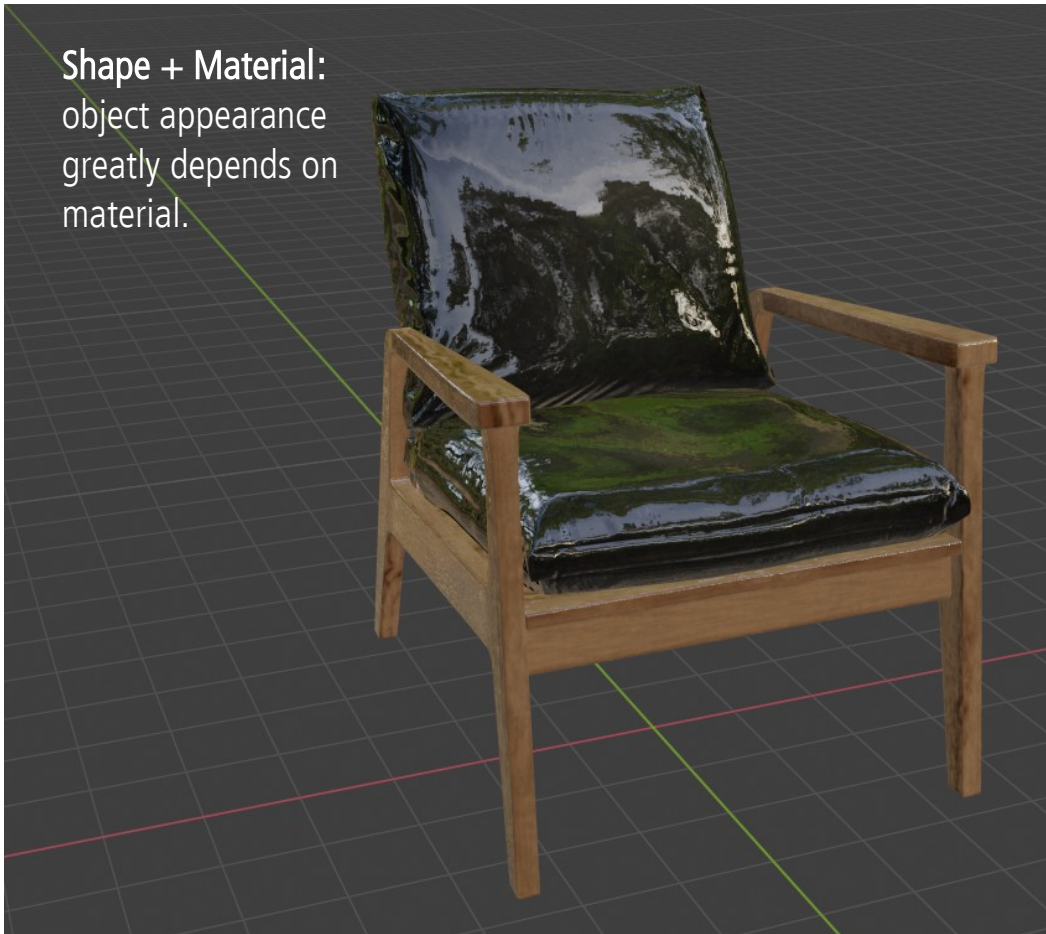
Material modeling

Rendering



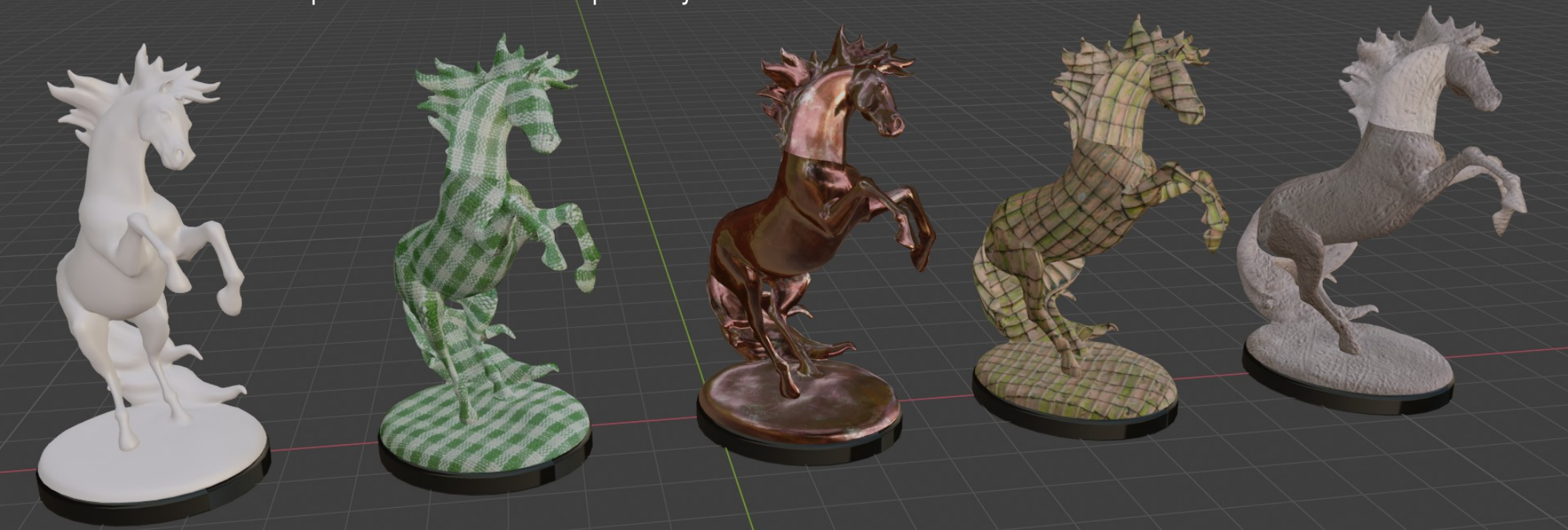
3D objects: appearance





Material and shape

Material and shape can be modeled separately.

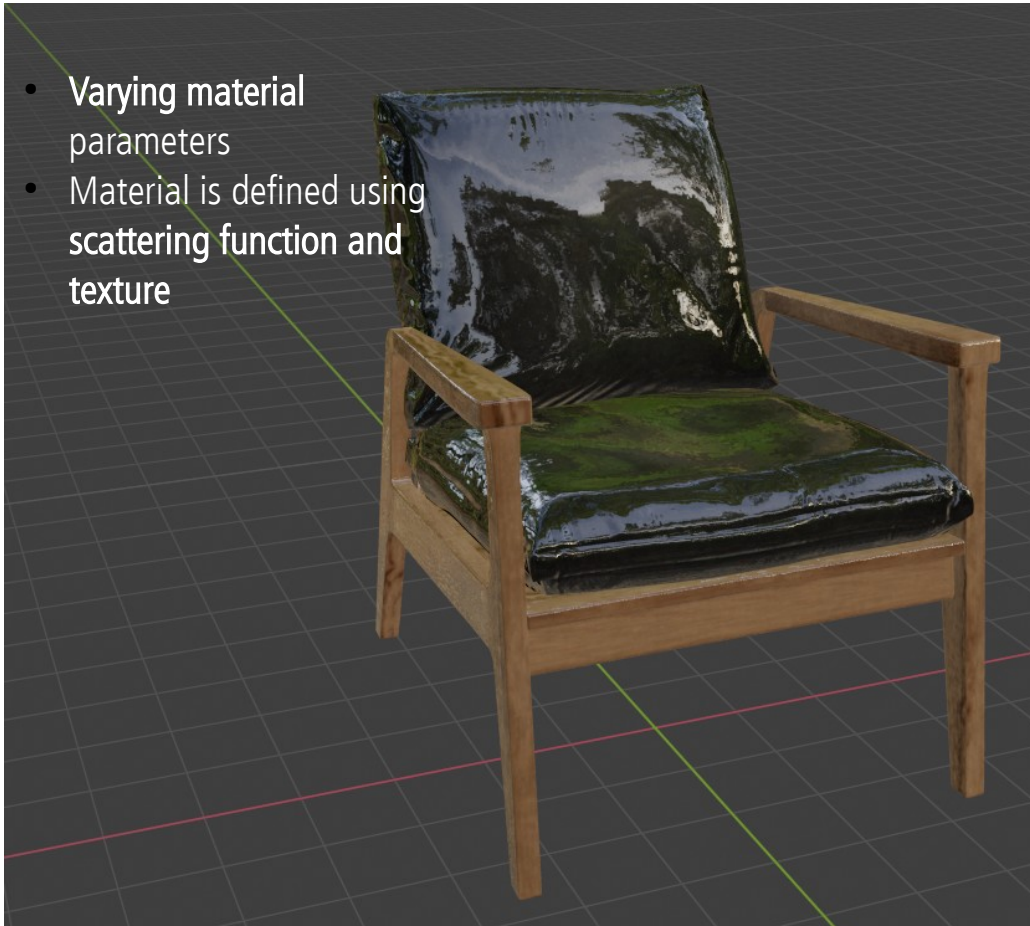


Material

- Defines light-matter interaction → object appearance
- Material modeling is decomposed into:
  - How light scatters on surface → **scattering model**
  - How small scale geometry and properties vary over surface → **texture**

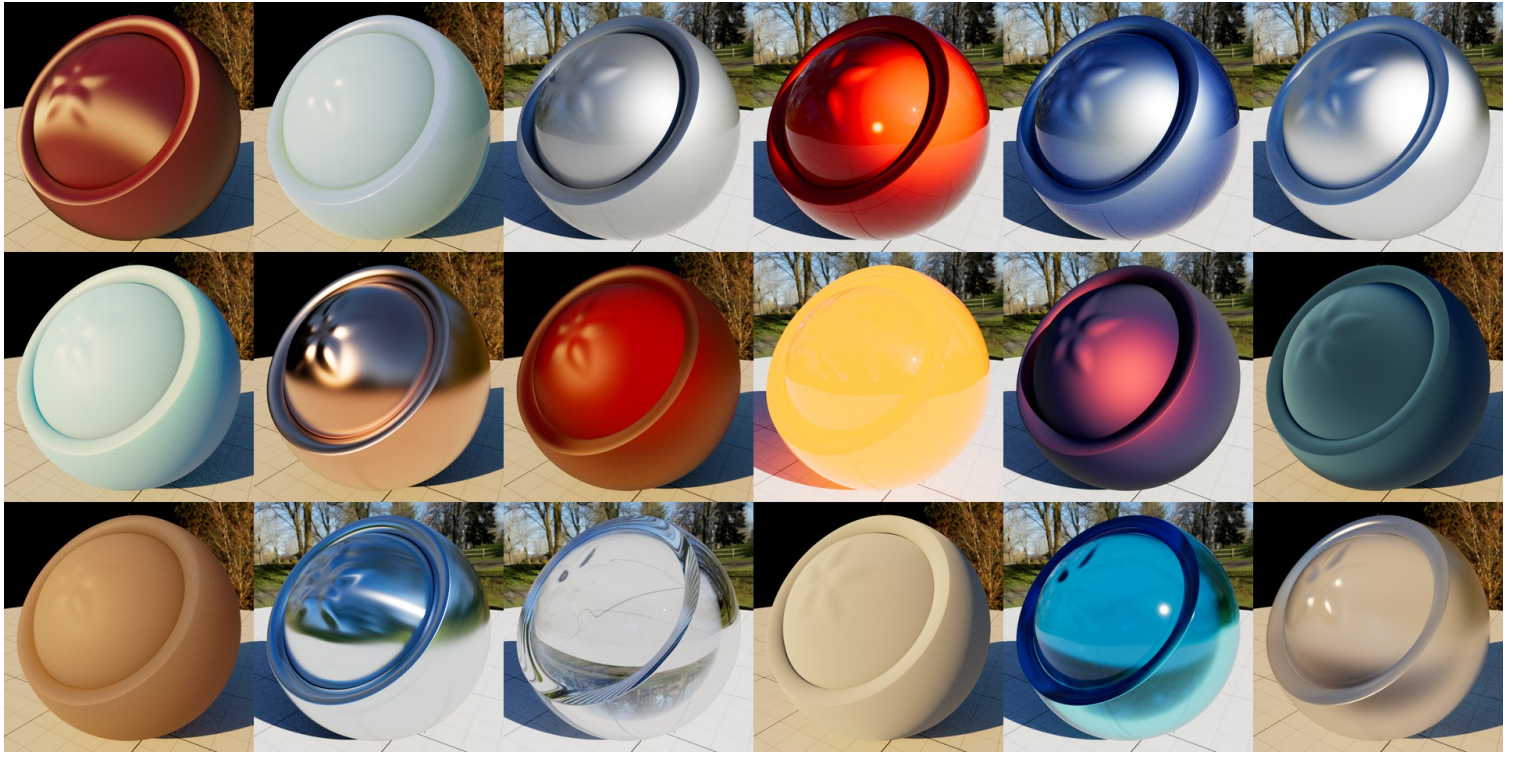
Material


• **Uniform material** parameters
• Material is defined using only **scattering model***


• **Varying material** parameters
• Material is defined using **scattering function and texture**

Material: light scattering

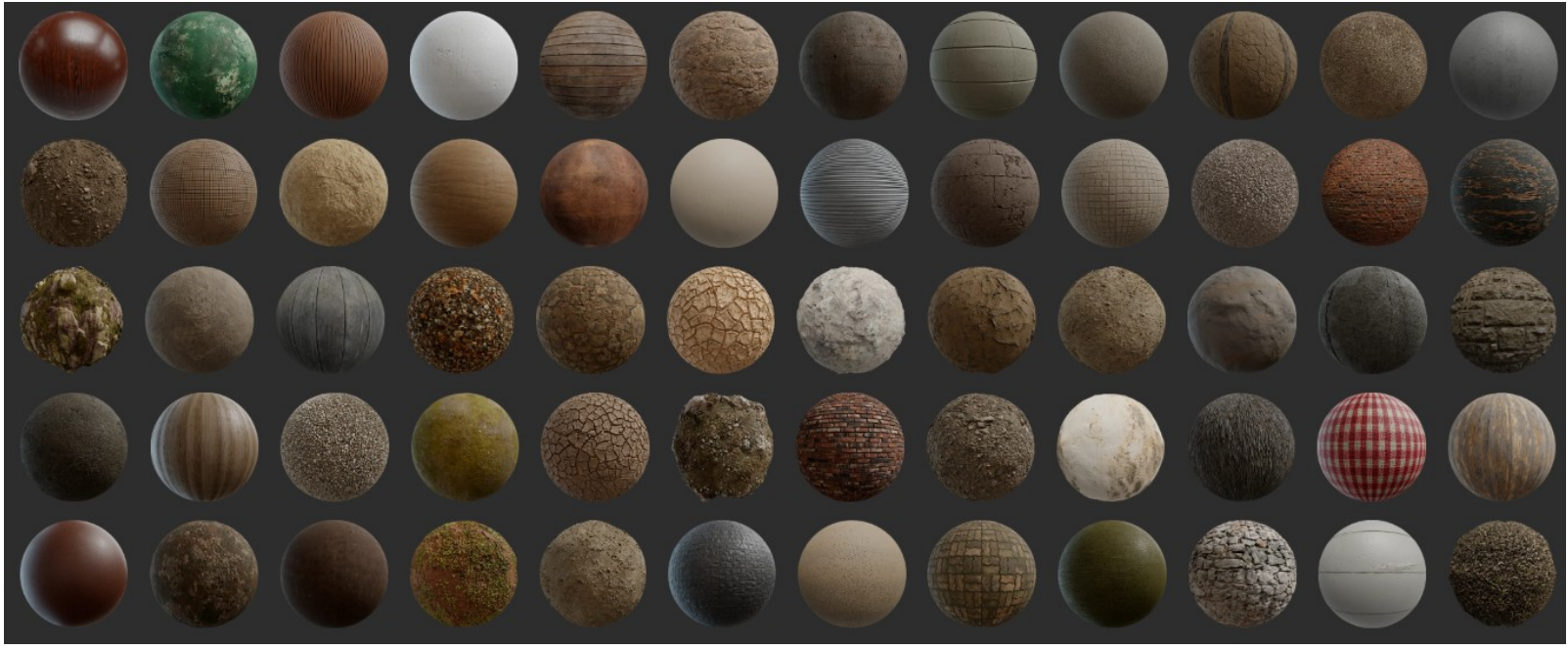
- Defines if object appears like:
  - Metal
  - Plastic
  - Glass
  - Glossy
  - Matte
  - Hazy
  - Reflective
  - Transmissive
  - Etc.



https://appleseedhq.net/gallery.html

Material: texture

- Gives look and feel to the object:
  - Wood
  - Dirt
  - Cloth
  - Fabric
  - Rock
  - Brick
  - Plaster
  - Finishing
  - Imperfections
  - Etc.



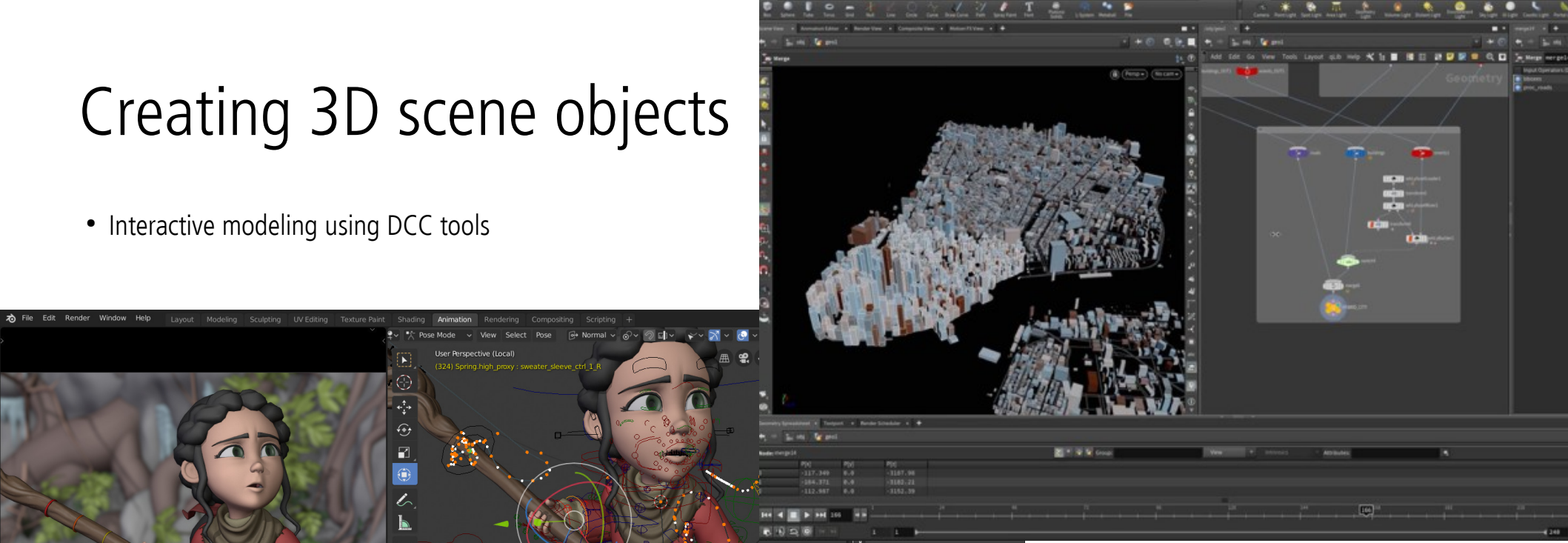
https://polyhaven.com/textures

Material

- Generally, material defines:
  - How light travels between objects: participating media, e.g., fog
  - How light interacts with surfaces: surface scattering, e.g., metals
  - How light interacts sub-surface, e.g., wax, leafs

Creating 3D scene objects

- Interactive modeling using DCC tools



https://www.sidefx.com/products/houdini/

https://www.blender.org/

Creating 3D scene objects

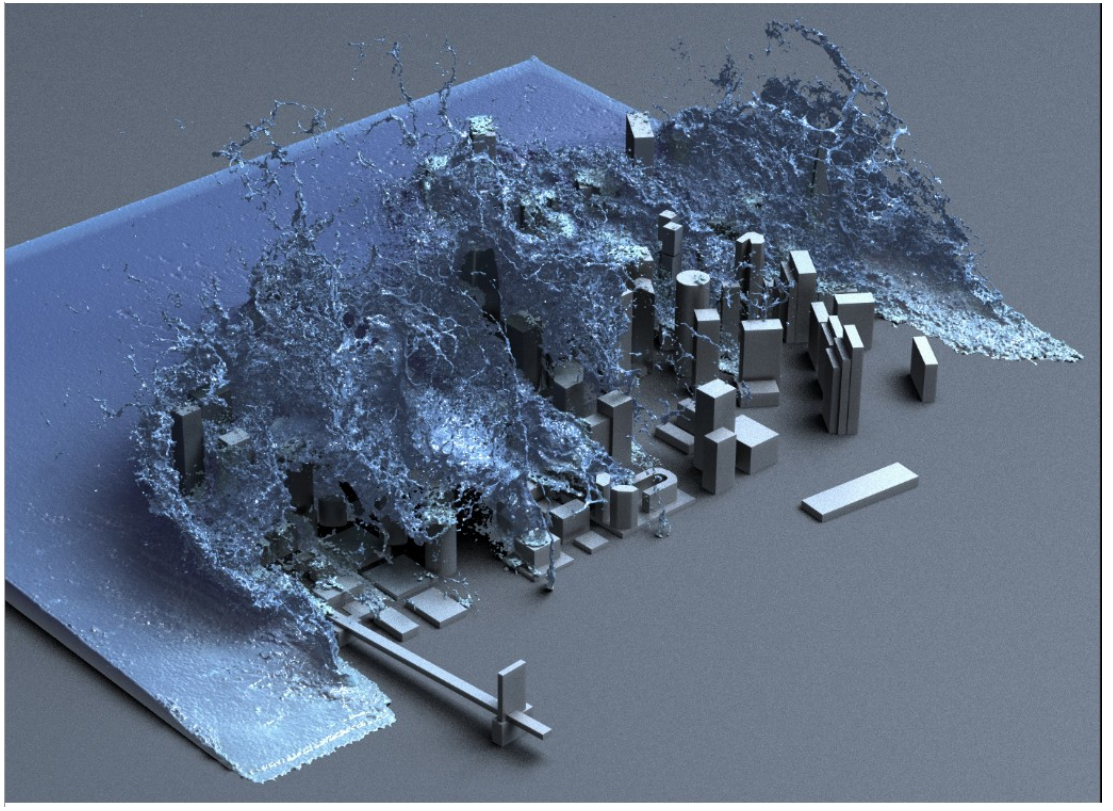
- 3D scanning can be used to "import" real object into digital world.

https://quixel.com/megascans/home/

https://www.myminifactory.com/object/3d-print-bust-of-antinous-as-dionysus-50345

Creating 3D scene objects

- Simulations
  - Some objects are very hard to model by hand and capture from a real world.
  - Physical simulations are employed to generate shape.



https://people.csail.mit.edu/kuiwu/gvdb_sim.html

Light in 3D scene

https://discover.therookies.co/2018/06/24/learn-how-to-light-and-render-like-a-pixar-artist/

Lights

- Light sources define:
  - Position and direction
  - Color and intensity

Lights

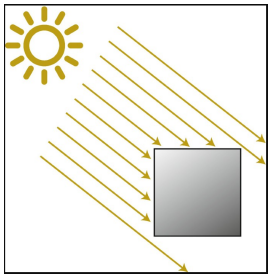
- Light sources can further have:
  - Size
  - Shape

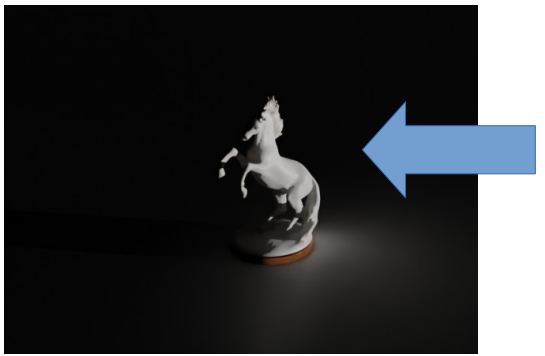
Physical lights have size and shape

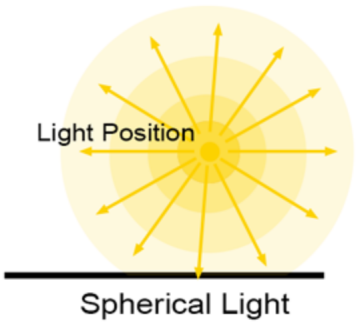
Non-physical lights only determine position/direction and intensity

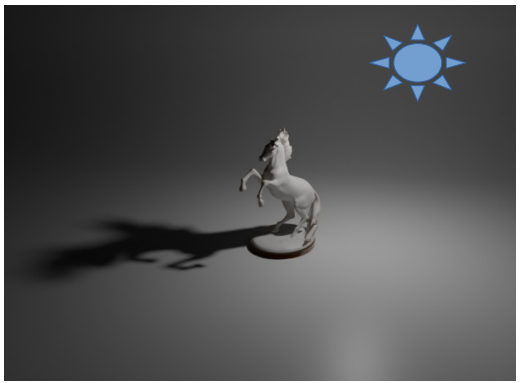
Lights

- Foundamental light sources are:
  - Point lights
  - Directional lights









Non-physical lights: no shape or size!

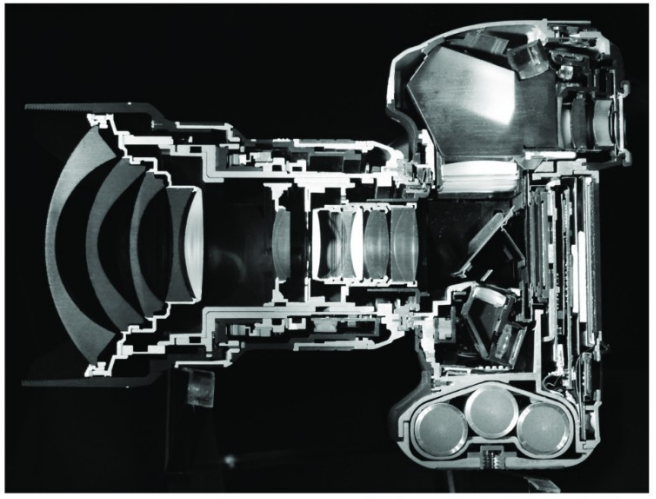
https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/shading-lights

Camera

- Camera defines point of view and it is used to simulate effects of real-world cameras.

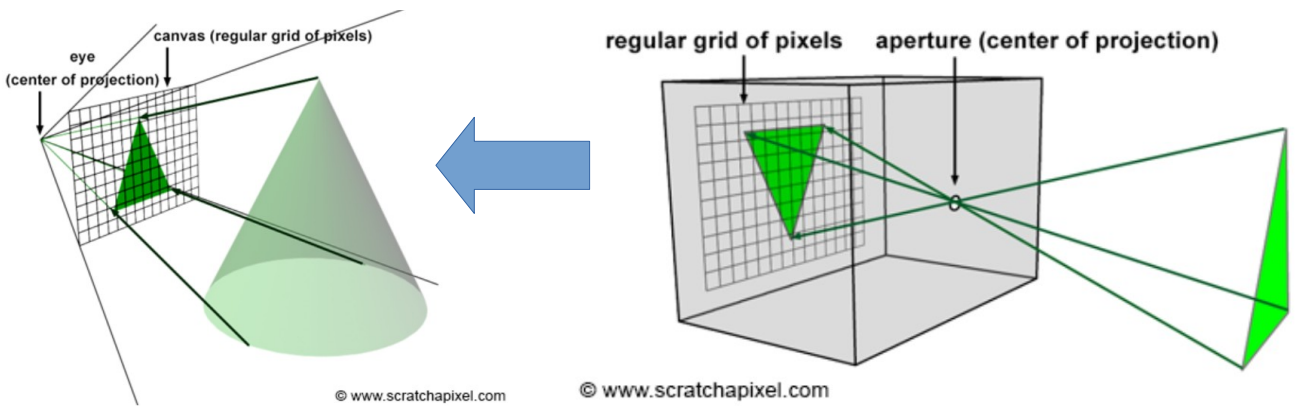
Camera

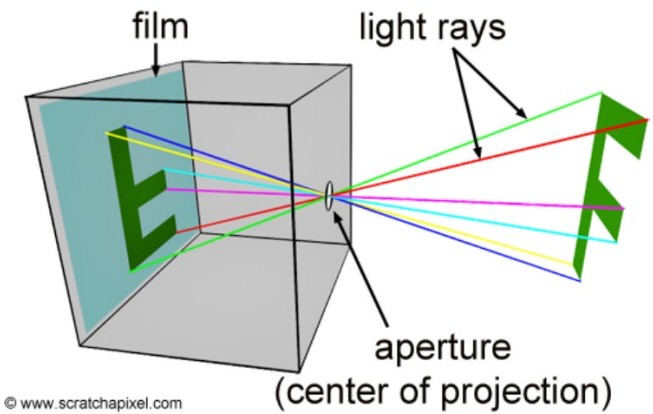
- Fundamental camera model in computer graphics: pinhole camera:
  - Eye (aperture)
  - Focal length
  - Film size
  - (Lens)



Cross-section of Nikon D3, 14-24mm F2.8 lens

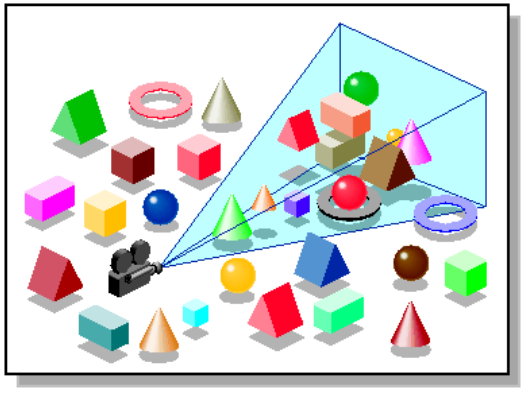
Pinhole camera simplification

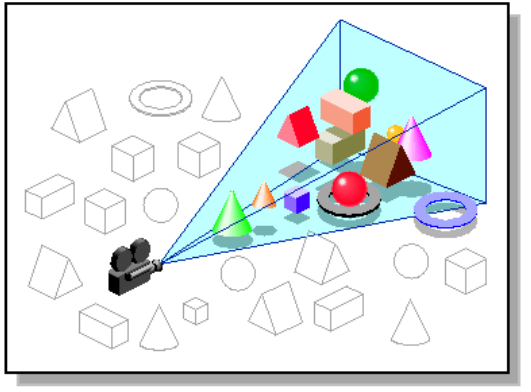


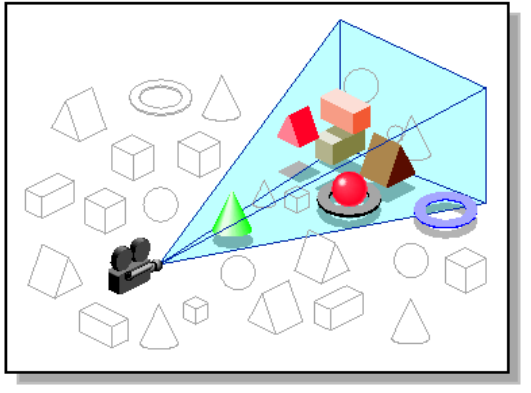


Camera

- Camera defines portion of visible scene → **viewing frustum**
- Amount of visible scene depends on:
  - Film size
  - Focal length







Animated objects

- Single image can show static objects in 3D scene
- Introducing **time component** in 3D scene modeling:
  - Objects or their parts can move: **scaling, translating, rotating → animation**
- Rendering image after each movement in 3D scene → sequence of images

Animation

- Animation can be created:
  - Interactive modeling: rigging, skinning and keyframing
  - Procedurally: e.g., simulation
- Animation can be:
  - Pregenerated
  - Generated at render-time

https://www.fudgeanimation.com/experiments/top-5-rigging-tips/

https://dreamfarmstudios.com/blog/what-you-might-not-know-about-the-vfx-component-of-the-3d-animation-pipeline/

Interaction with 3D scene

- 3D application can listen to user input and modify 3D scene based on that input
  - Game engines: Godot, Unity, Unreal

https://www.orithegame.com/

https://unity.com/solutions/digital-twins

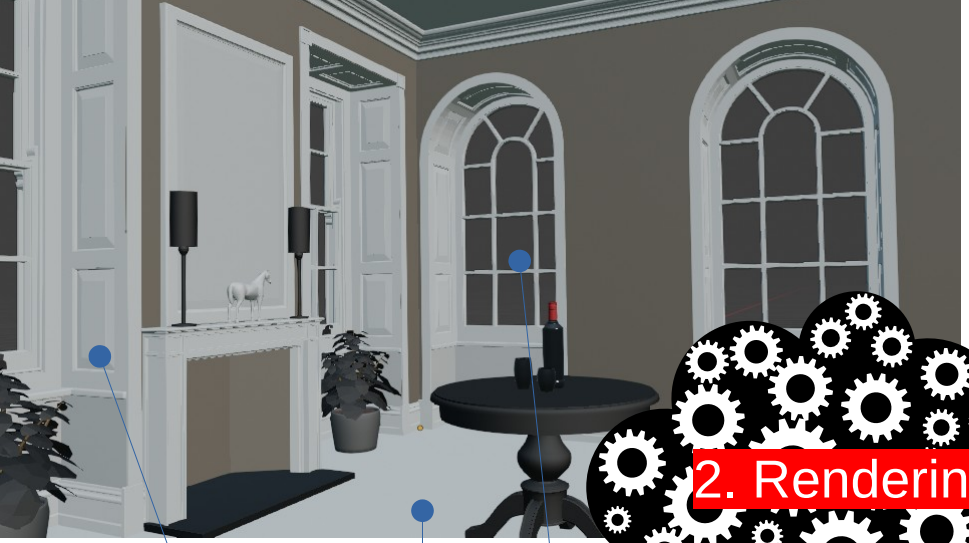
3D scene: recap

- **3D scene**
  - 3D objects
  - Lights
  - Cameras
- **Rendering** takes 3D scene information for generating image.

Rendering

A process of generating image from a 3D scene

Rendering:
generating images
from a 3D scene.

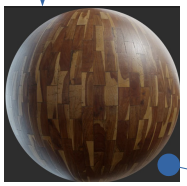


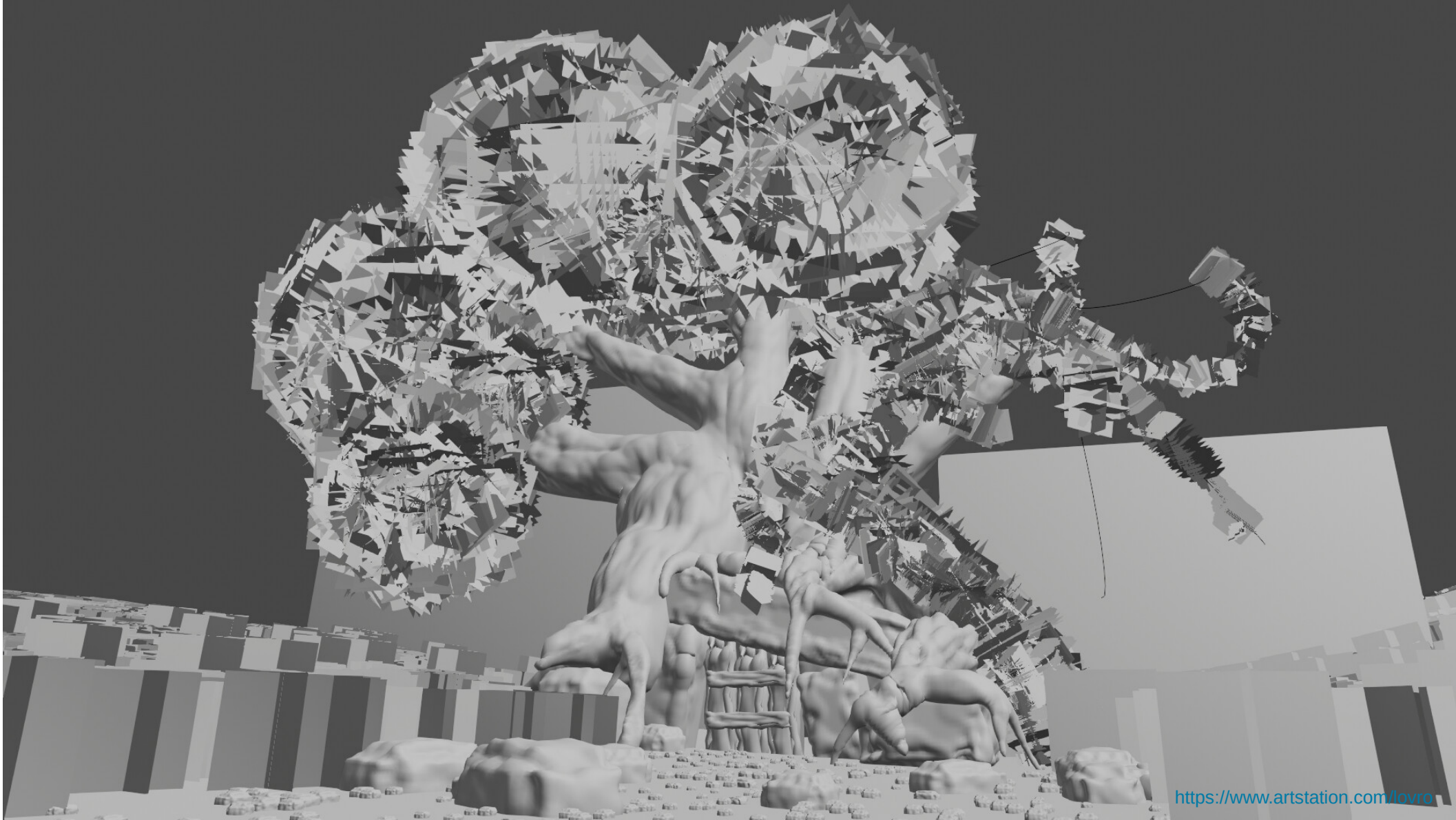
1. Scene modeling

2. Rendering

3. Image

Material
modeling




https://www.artstation.com/lovro


https://www.artstation.com/lovro

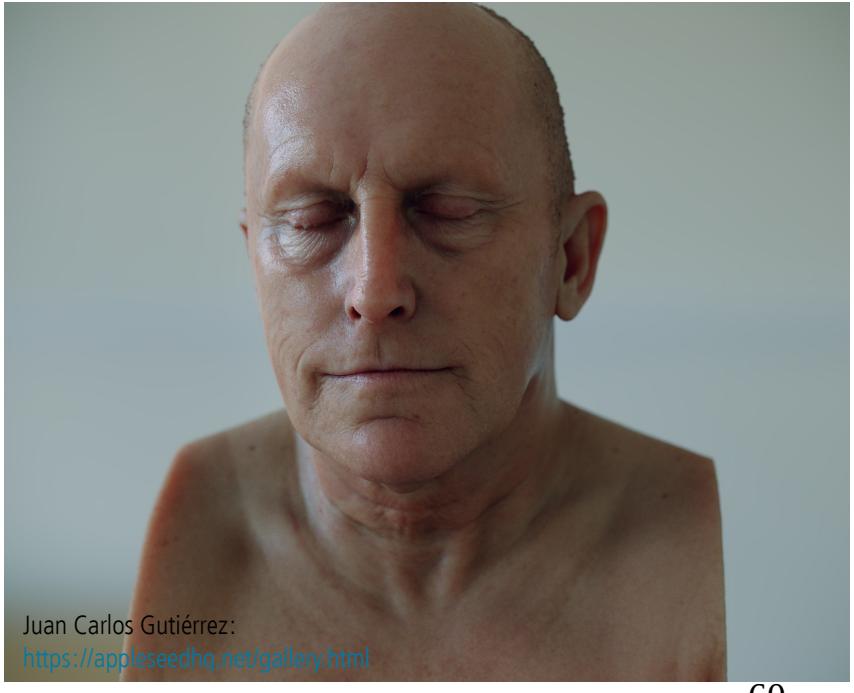

https://www.artstation.com/lovro


https://www.artstation.com/lovro

Rendering: photo-realistic images

- Often, goal is to create **photo-realistic** images – images that look like a photograph* of a real world.
  - To make objects appear as they do in real world, we need to simulate laws of physics that are related to appearance – optics, that is, simulating light transport and interaction with objects in 3D scene and camera.

Juan Carlos Gutiérrez:
https://appleseedhq.net/gallery.html



Juan Carlos Gutiérrez:
https://appleseedhq.net/gallery.html

* Note that there is also distinction between images that look like a real photo and that are "correct" like a real photo. In graphics, almost always it enough to create images that look like a real photo but the way they are produced doesn't necessary be correct in terms of how real world works. Again, this is trade-off between quality and speed and decision depends on application.

Rendering: non-photorealistic images

- On the other hand, creating **non-photo-realistic** (NPR) images* – images that contain wide range of expressive styles, is also often desired.

* Specific look of NPR images often comes from exaggerating some characteristics of 3D scene or rendering algorithm. Therefore, it is better to start with photo-realistic rendering and then altering this method to obtain required style in NPR.

Rendering

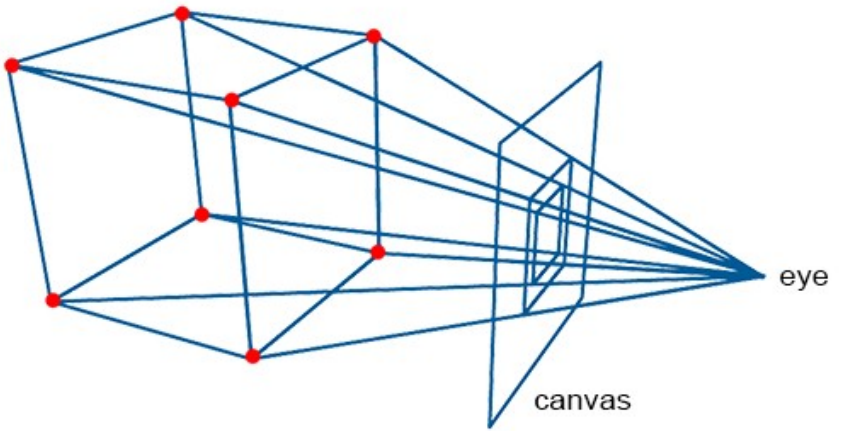
- In both cases, image is generated using certain **rendering** method.
- Rendering process takes 3D scene information and creates 2D image.

Rendering: simulating visual system

- **Foreshortening effect:** objects that are further appear smaller than those which are closer
- Important for photo-realistic rendering since it simulates how our visual system works
  - It defines shape and size with respect to the distance to the eye.

Foreshortening effect: trace rays from corners of objects to eye and intersecting them with imaginary canvas that lies in between → **perspective projection.**

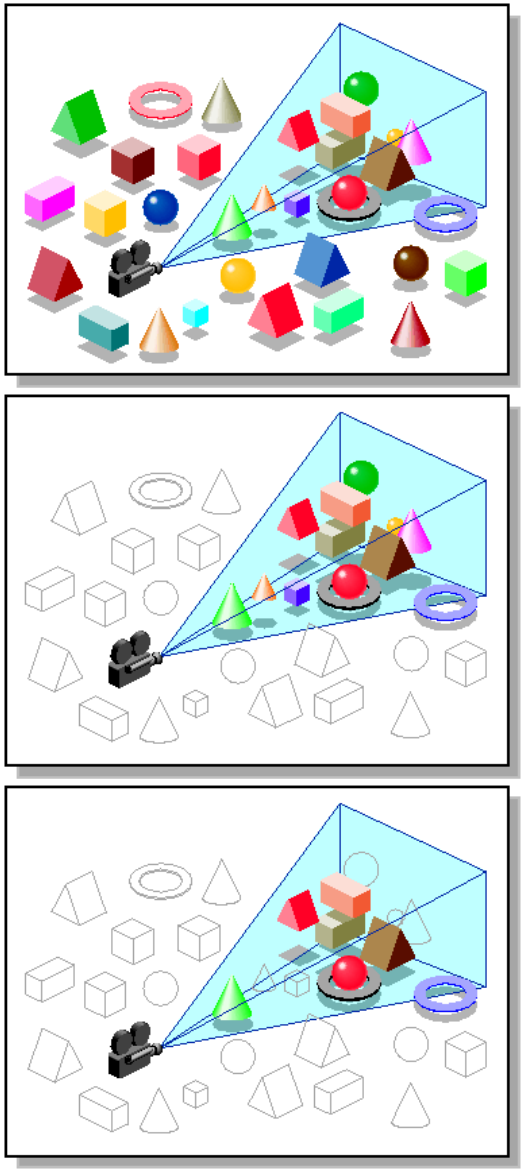
Direct result of such method is called **wireframe rendering**.



© www.scratchapixel.com

Rendering: visibility problem

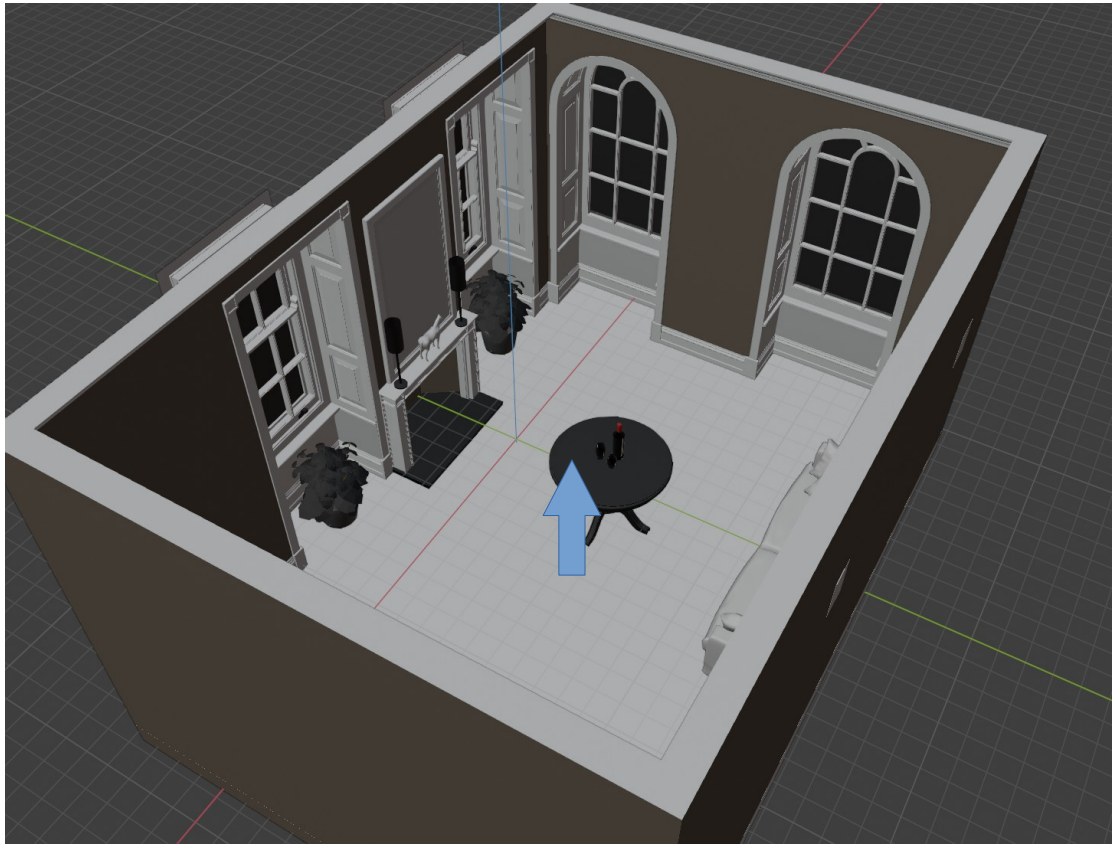
- Camera placed at certain point of view covers only small portion of 3D scene
- **Visibility problem*:** determine if two points in 3D scene are visible (not obstructed)
  - Determine which objects are visible from camera
  - Determine which surfaces are visible to each other in 3D scene
- Solving the visibility relies in object **shape information**
- Solution to visibility problem in computer graphics can be solved using two main methods:
  - **Rasterization****
  - **Ray-tracing**



https://techpubs.jurassic.nl/manuals/nt/developer/Optimizer_PG/sgi_html/ch05.html

* Also known as: hidden surface elimination, hidden surface determination, hidden surface removal, occlusion culling and visible surface determination
** This is umbrella term and some popular methods are painter's algorithm and z-buffer. Almost all GPUs use an algorithm from this category.

Rendering: visibility problem



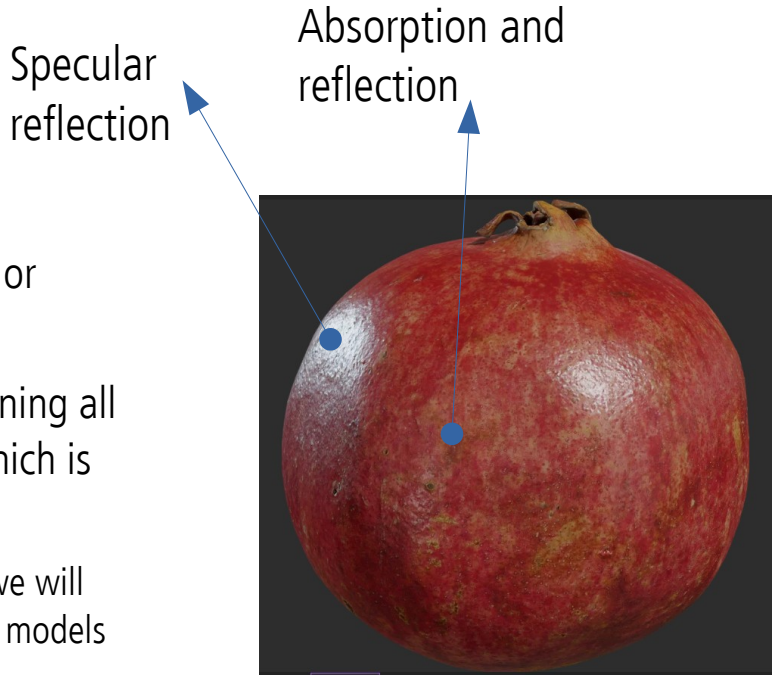
Rendering: appearance

- Once visibility problem is solve; which geometry is visible from certain viewpoint, **appearance of object** is calculated
  - Appearance: The look of objects in terms of color, texture and brightness.
- Object appearance depends on:
  - Object shape and material
  - Illumination
  - Camera viewpoint
- Appearance of object is simulated using **shading**

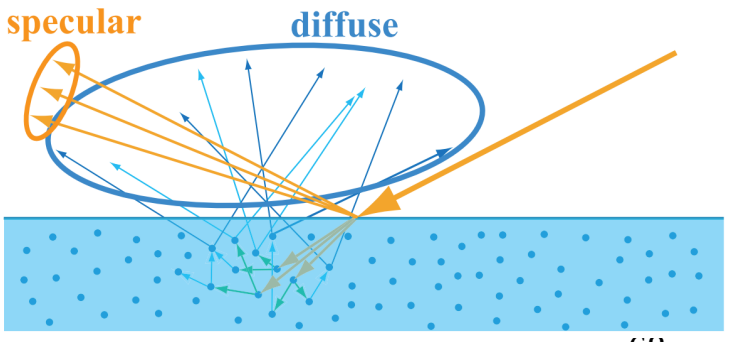
Rendering: shading

- **Shading** defines interaction of light with object material
  - Gathering light falling on surface → **light transport**
- Implementation of mathematical model which simulates light interaction with surface is called a **shader**.
- Real-world light-matter interaction is complex topic.
  - In CG there is always trade-off between quality of image and rendering speed.

Rendering: shading

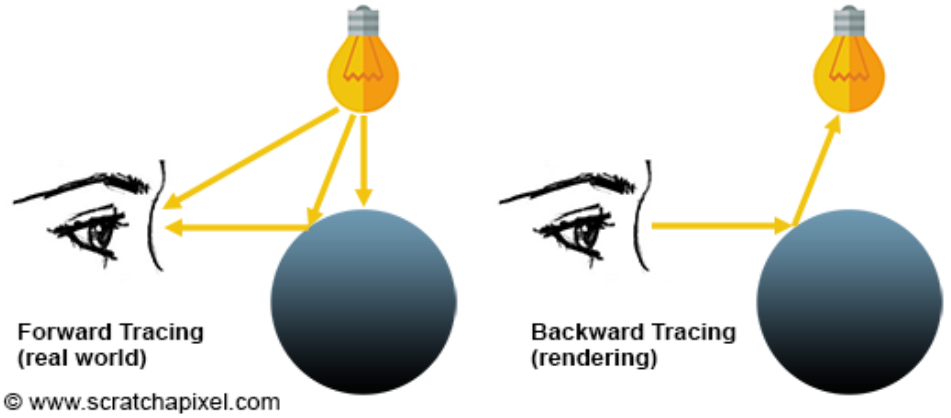


- When light comes in contact with an object it can be: absorbed or reflected
- **Absorption** gives objects their unique color. If white light (containing all colors) falls onto object which absorbs all colors except red – which is reflected, we perceive this object as red.
  - Color of object is one foundational material information of object as we will see later. It can be defined directly or indirectly using physically-based models which calculate color.
- **Reflection**. Light which is not absorbed is reflected. Direction of reflection depends on:
  - Surface orientation → **surface normal** vector.
  - **Scattering** – a model which describes what happens with light as small scale. This model can have uniform parameters or varying parameters – giving textured look



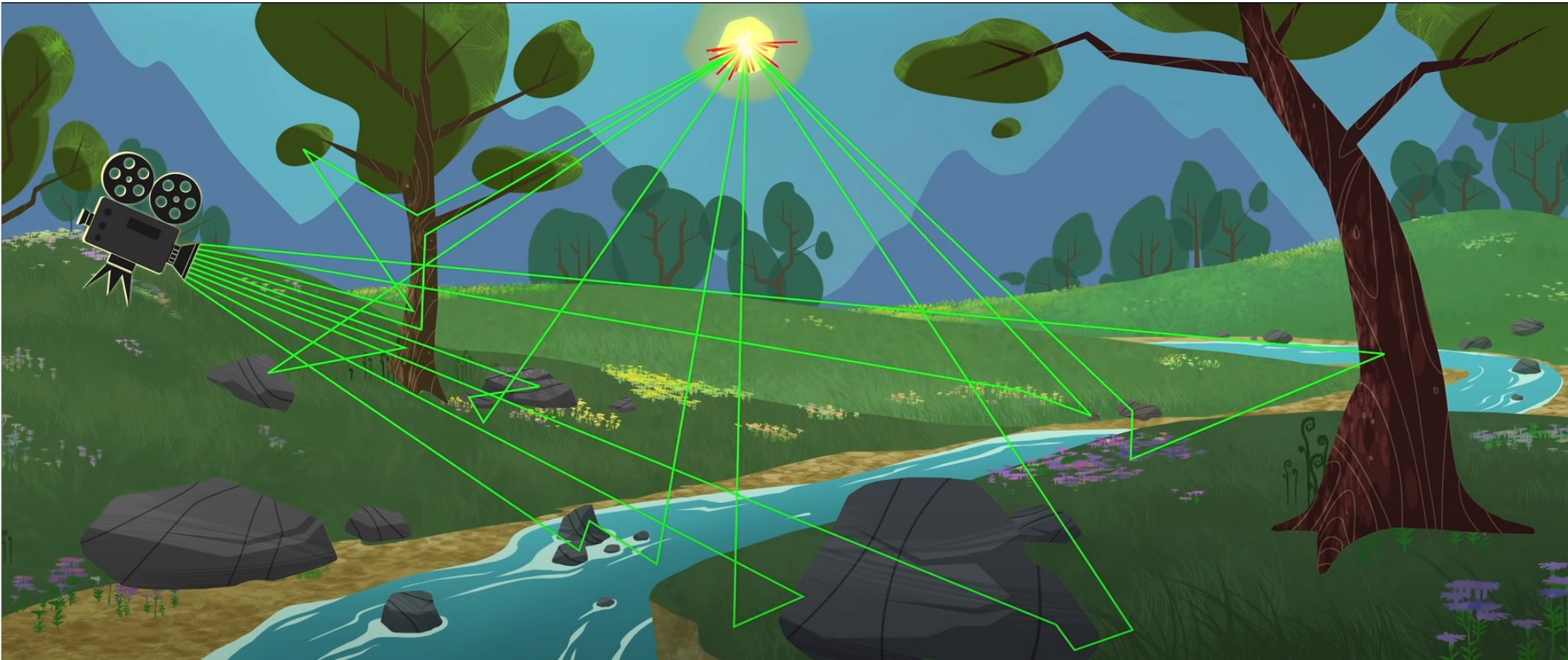
Rendering: light transport

- Crucial information for shading is amount of light falling on surface → **light transport**
- Light is emitted from light sources, reflects from objects and eventually might fall into camera → **forward tracing**
  - Extremely expensive to calculate complete light behavior in whole scene.
- **Backward tracing**: starting from camera, find visible objects and incoming light from there

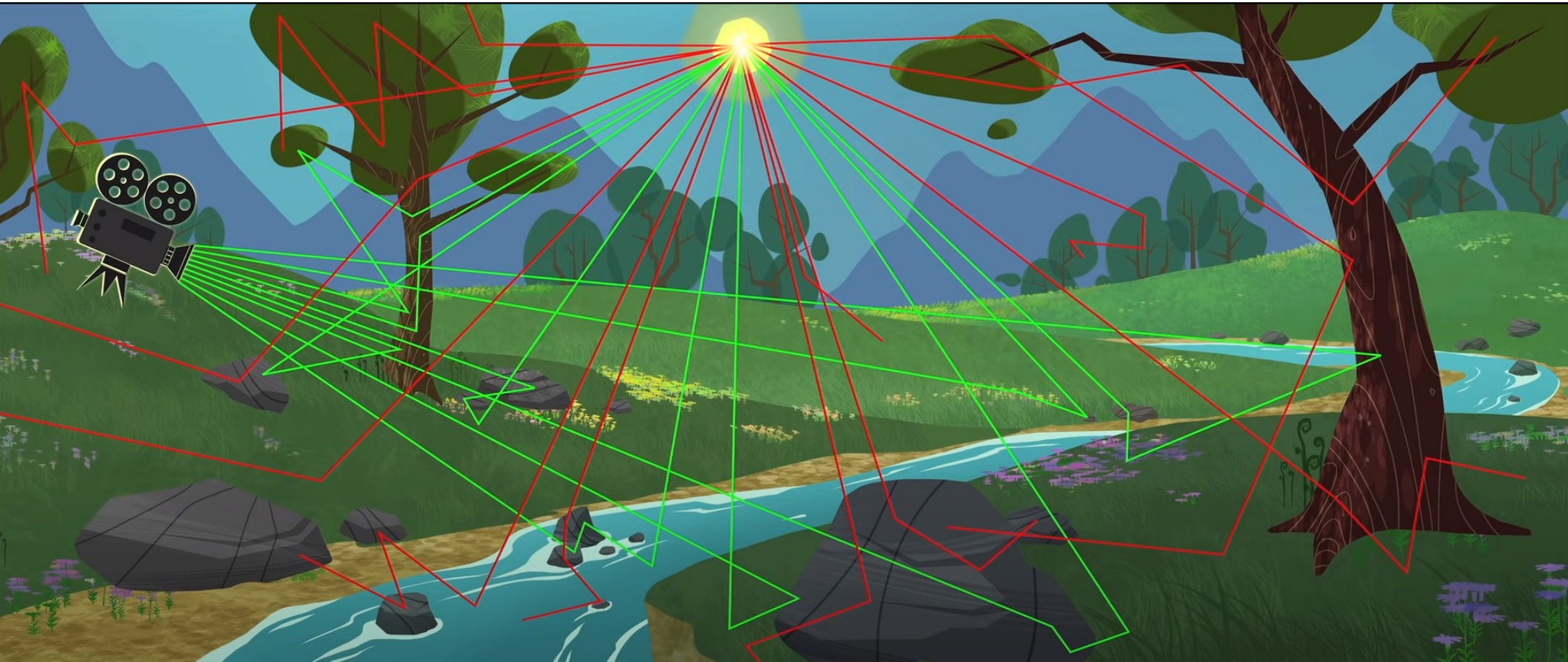


Forward tracing

Backward tracing

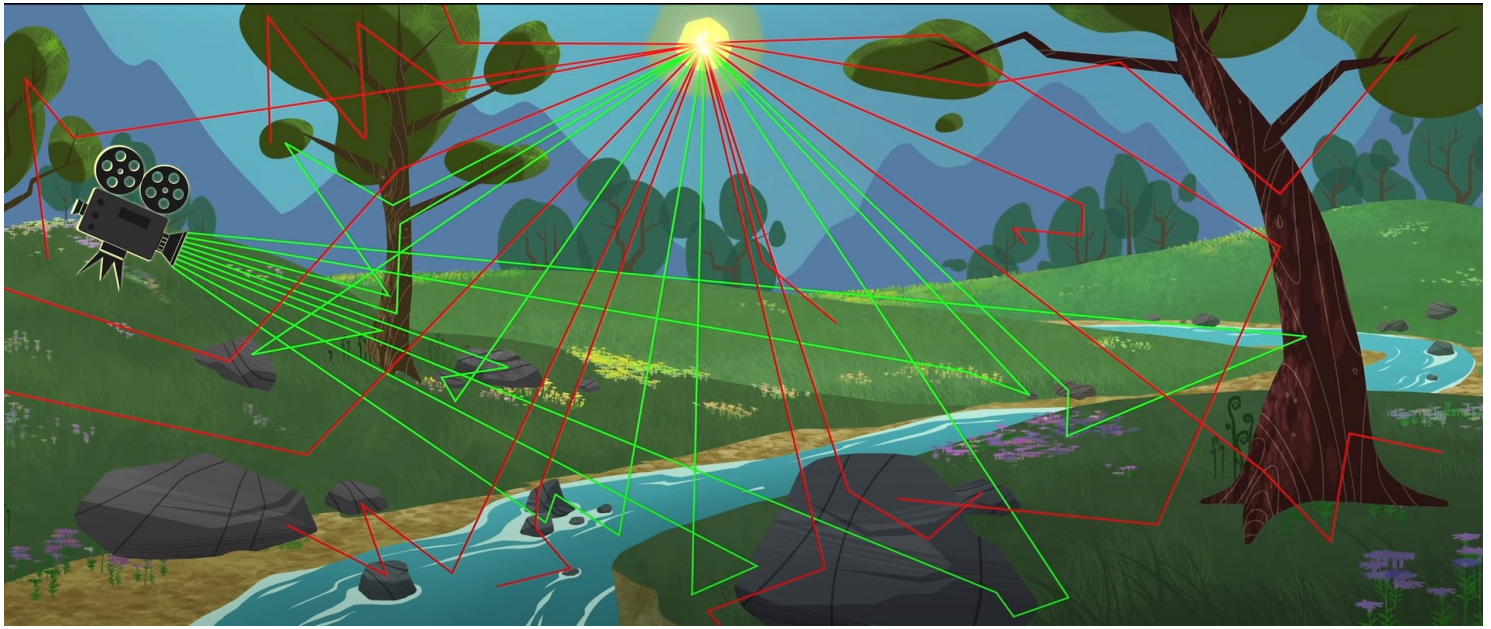


Forward tracing is expensive!



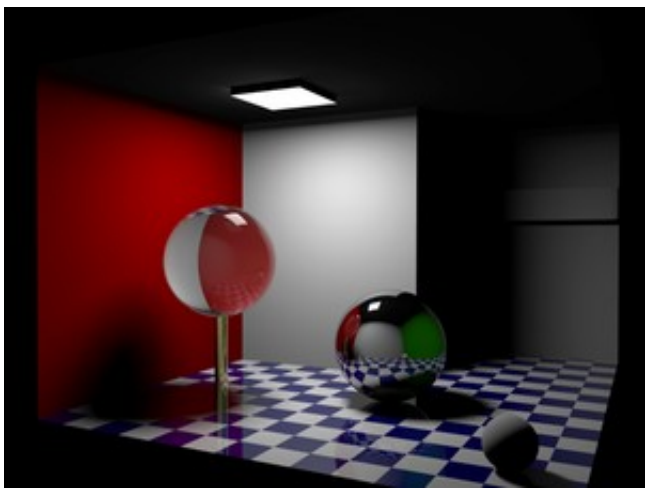
Forward tracing

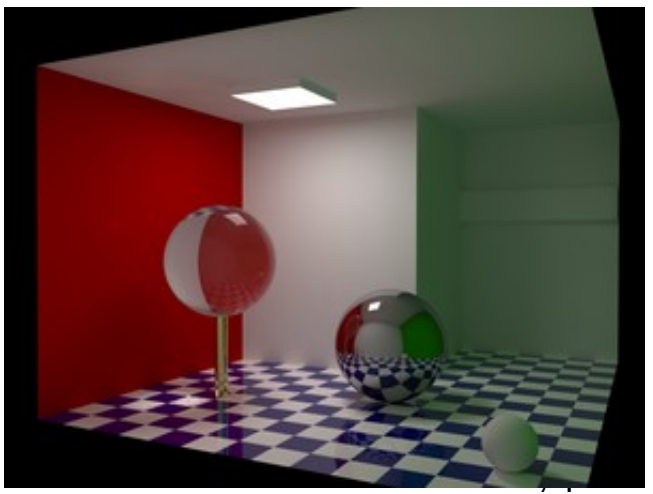
- Object surface which is visible from camera might receive light from any light source or surface which is facing.
- Amount of light which falls into camera is extremely small. Thus rendering employs light transport only for visible surfaces starting from the camera and only for relevant light paths.



Local and global illumination

- Taking in account only light from light sources → **direct (local) illumination**
- Taking in account light reflected from other objects → **indirect illumination**
- Simulating direct and indirect illumination → **global illumination**
  - Often solved with ray-tracing based methods: path tracing
  - Another approach: radiosity

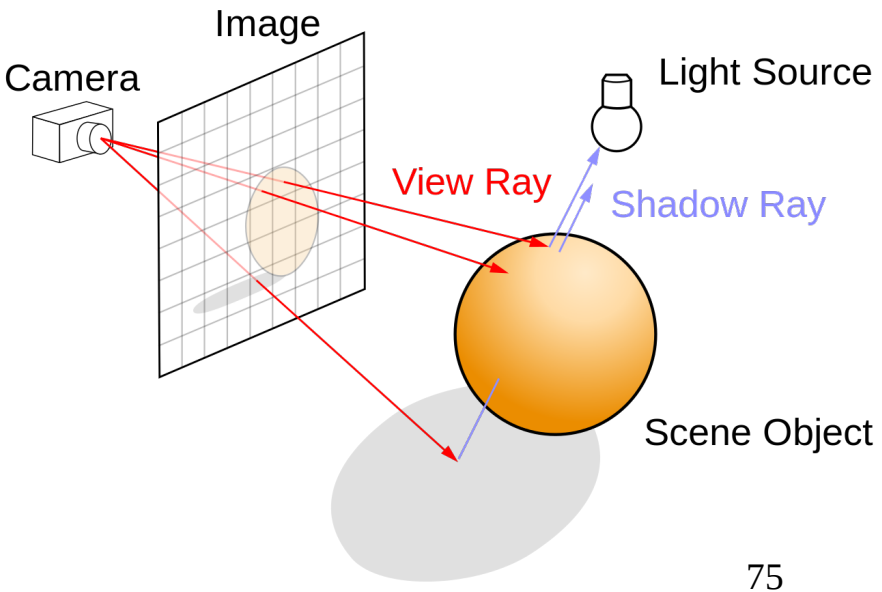




https://en.wikipedia.org/wiki/Global_illumination

Rendering: recap

- Rendering:
  - Solving **visibility problem**: what is visible from camera point of view
    - **Perspective projection**
    - **Object shape**
  - **Shading**: calculating appearance of visible objects
    - Material and shape of object
    - Viewing direction
    - Light falling on the object: **light-transport** (again, visibility problem)

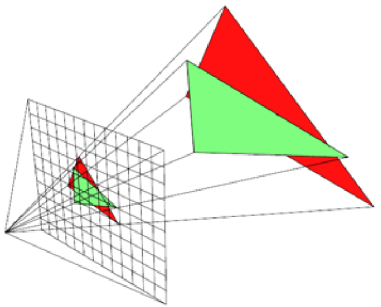


Practical rendering

- In practice, rendering is implemented using
  - **Rasterization**
  - **Ray-tracing**
- These two methods are essentially solving visibility problem
- Shading is influenced by this choice since it depends on visibility problem

Rasterization

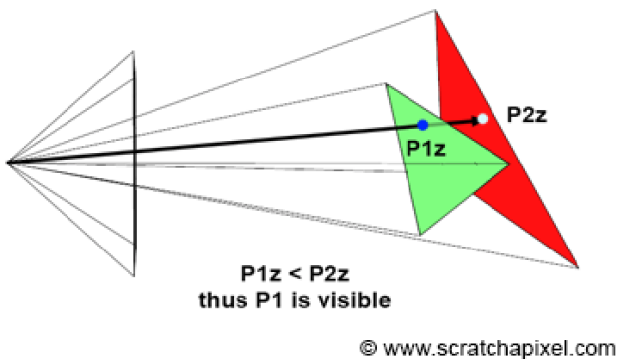
- Visibility problem is solved by projecting object on surface of canvas from given point of view
  - Project point `P` onto screen to compute `P'`
  - Perspective projection matrix
- Object-centric approach




```
for T do in triangles
    for P do in pixels
        determine if P inside T
    end for
end for
```

Ray-tracing

- Trace rays from camera for each pixel on canvas and intersect objects in the scene → closest objects are visible ones
  - Perspective camera
- Image-centric approach




```
for P do in pixels
    for T do in triangles
        determine if ray through P hits T
    end for
end for
```

Scene

Blender, WorkBench
Scene: Crytek Spinoza (Sponza Palace in Dubrovnik)
https://casual-effects.com/data/index.html

Rasterization-based render

Ray-tracing-based render

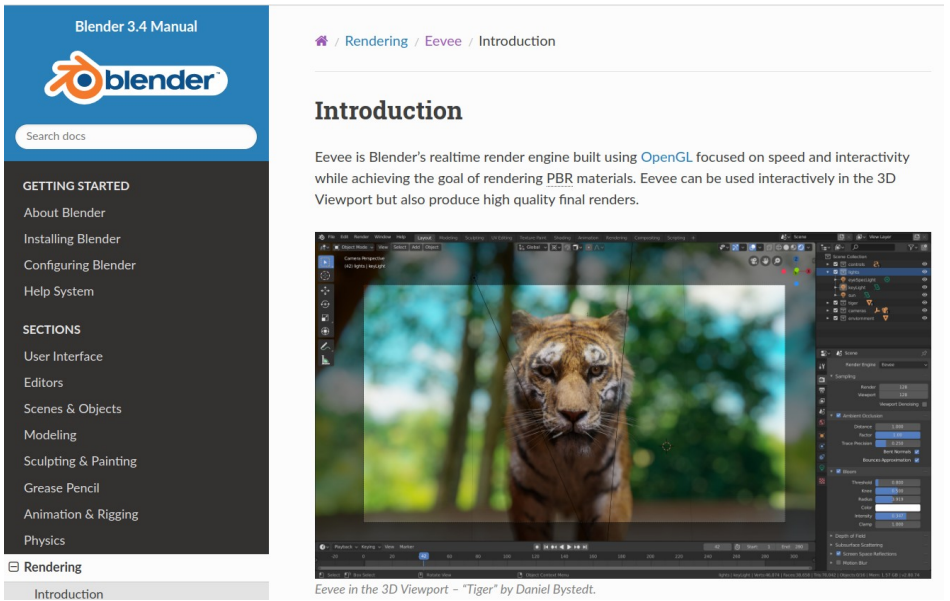
Blender, Cycles
Scene: Crytek Spinoza (Sponza Palace in Dubrovnik)
https://casual-effects.com/data/index.html

Rasterization vs ray-tracing

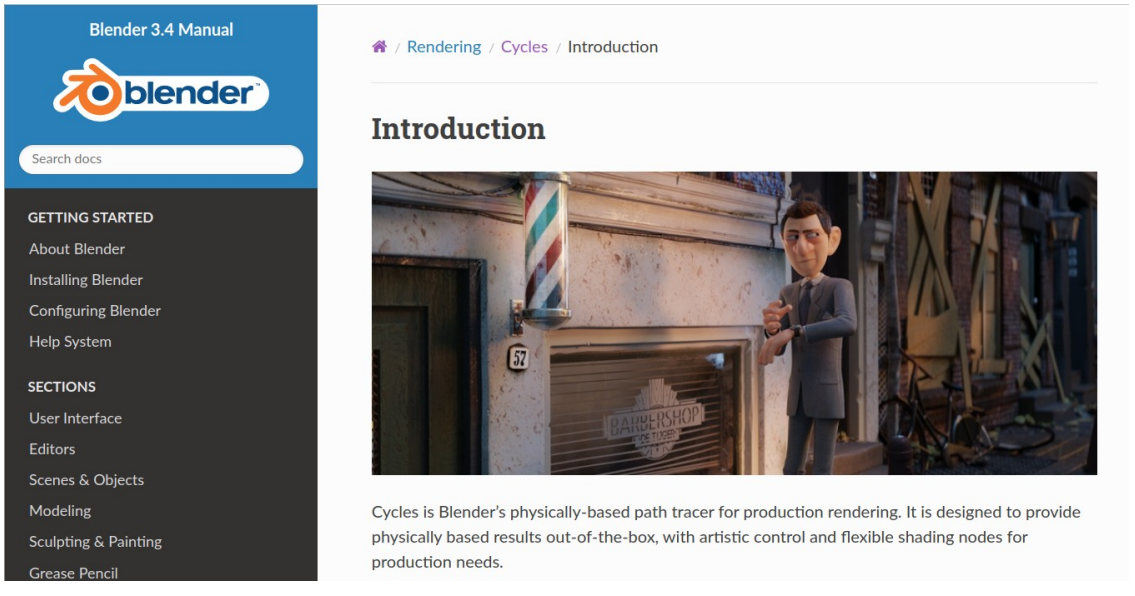
- **Rasterization** can quickly determine visibility but has limited shading capabilities when it comes to light transport
  - Geometry (object shape) is projected onto canvas and large amount of information is lost
  - Mainstream method for used by graphic cards and in real-time rendering
- **Ray-tracing** has advanced shading possibilities when it comes to light-transport but it is slow
  - Shadows, soft shadows, reflections, refractions, volumetric scattering, etc.
  - Easier to create photo-realistic images
  - Often requires acceleration-structure for solving visibility problem

Practical rendering

- Blender EEVEE: rasterization-based rendering engine
- Blender Cycles: ray-tracing based rendering engine

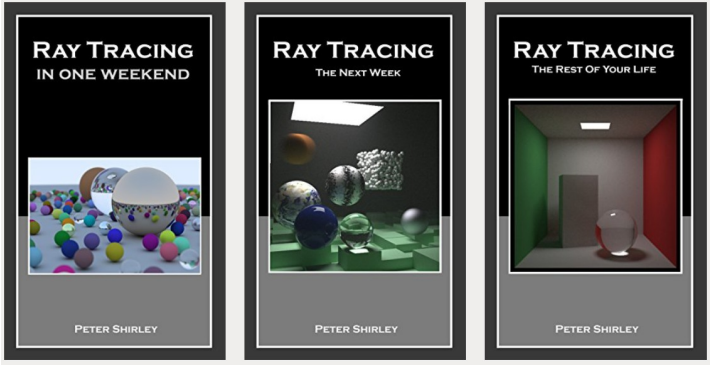


Eevee in the 3D Viewport – "Tiger" by Daniel Bystedt.



Practical rendering

- Projects!
- Programming your own rasterization-based rendered:
  - https://learnopengl.com/
- Programming your own ray-tracing based rendered:
  - https://raytracing.github.io/

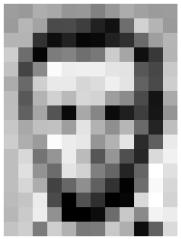


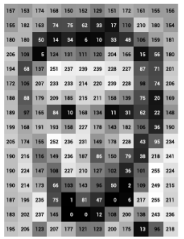
Image

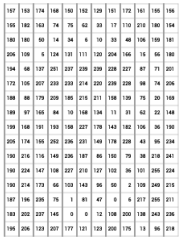
A result of 3D scene rendering

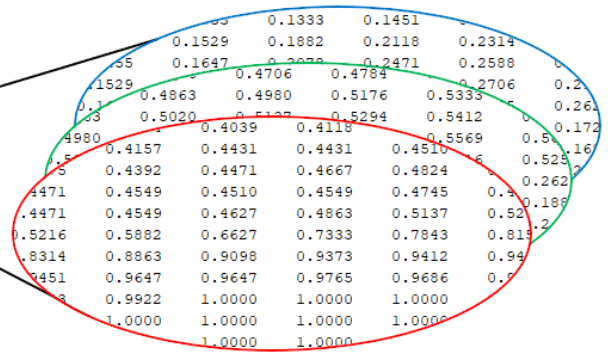
Image and display

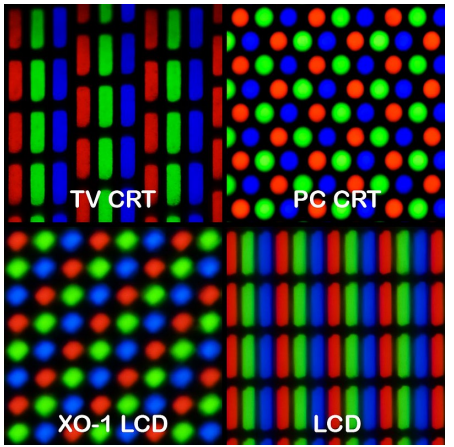
- Rendered image is displayed on **raster devices → raster images**
  - Raster: array of pixels → **discretization**
- Raster images: collection of pixels with **color values** (R,G,B)







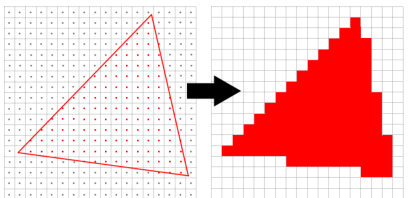


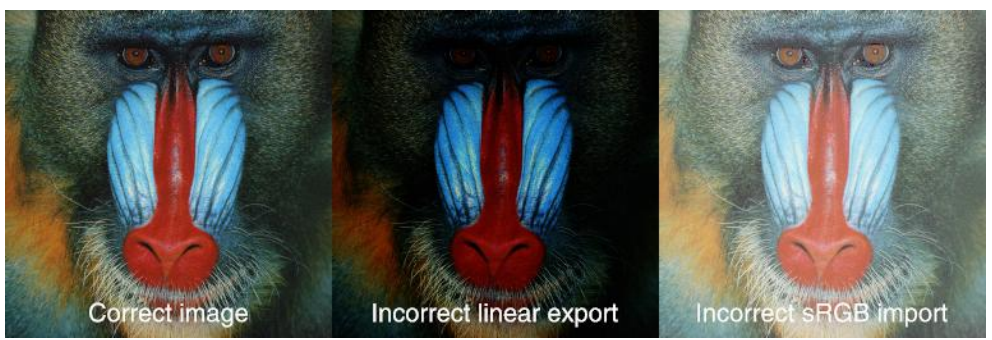


https://www.mathworks.com/help/images/image-types-in-the-toolbox.html

Image and display

- Limitations of display device and raster discretization cause problems:
  - Aliasing
  - Display gamut





https://www.gamedesigning.org/gaming/anti-aliasing/

Image post-processing

- Example: color-grading for achieving desired look on rendered image

https://advances.realtimerendering.com/other/2016/naughty_dog/NaughtyDog_TechArt_Final.pdf

Merging all together

3D scene, rendering and Image

Image synthesis: big picture

- **3D scene**, **rendering** and resulting **image** are highly intertwined elements of image synthesis process