Peter Mileff PhD

Programming of Graphics

The graphics pipeline

University of Miskolc
Department of Information Technology

What is the Graphics Pipeline?

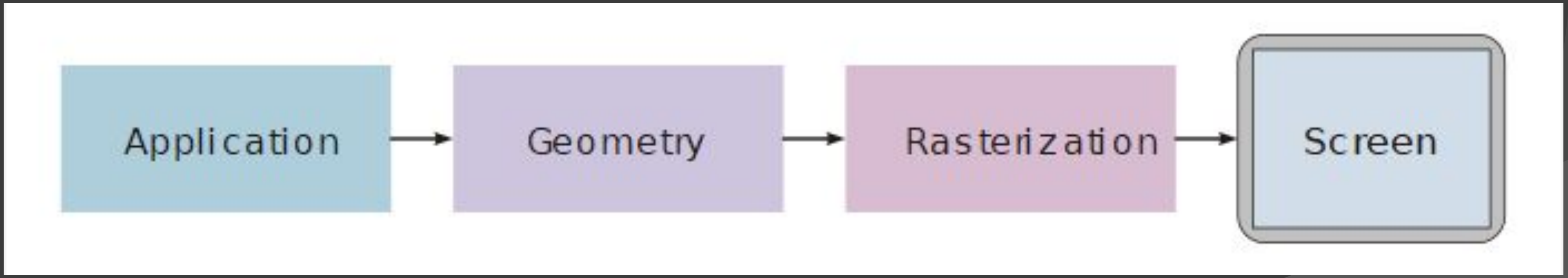
- **A conceptual model that describes what steps a graphics system needs to perform to render a 3D scene to a 2D screen**
- The steps highly depend on the used software and hardware and the desired display characteristics
  - Therefore there is no universal graphics pipeline suitable for all cases
- The model of the graphics pipeline is usually used in real-time rendering.
  - Often, most of the pipeline steps are implemented in hardware

What is the Graphics Pipeline?

- **The implementation of the pipeline can be:**
  - 1. Pure software based: only using the CPU
    - Early era
  - 2. Hardware based using the GPU
    - Today era
  - 3. As a combination of the two
    - Transitional period (today's era)

What is the Graphics Pipeline?

- A graphics pipeline can be divided into three main parts:
  - Application
  - Geometry
  - Rasterization



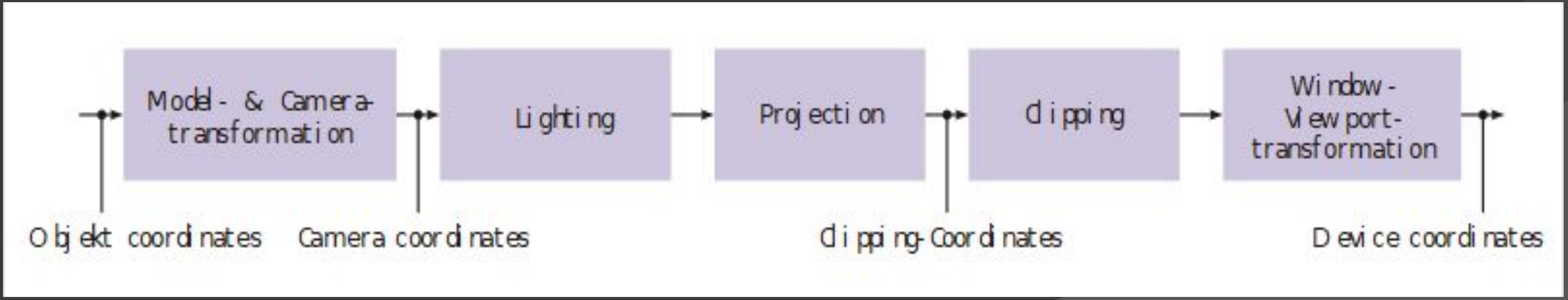
Application

- The application step is executed by the software on the main processor (CPU),
  - it cannot be divided into individual steps, which are executed in a pipelined manner

- However, it is possible to parallelize it on multi-core processors or multi-processor systems.

Geometry

- The geometry step is responsible for the majority of the operations with polygons and their vertices
- It can be divided into the five tasks
  - It depends on the particular implementation of how these tasks are organized as actual parallel pipeline steps.



Model and Camera transformation

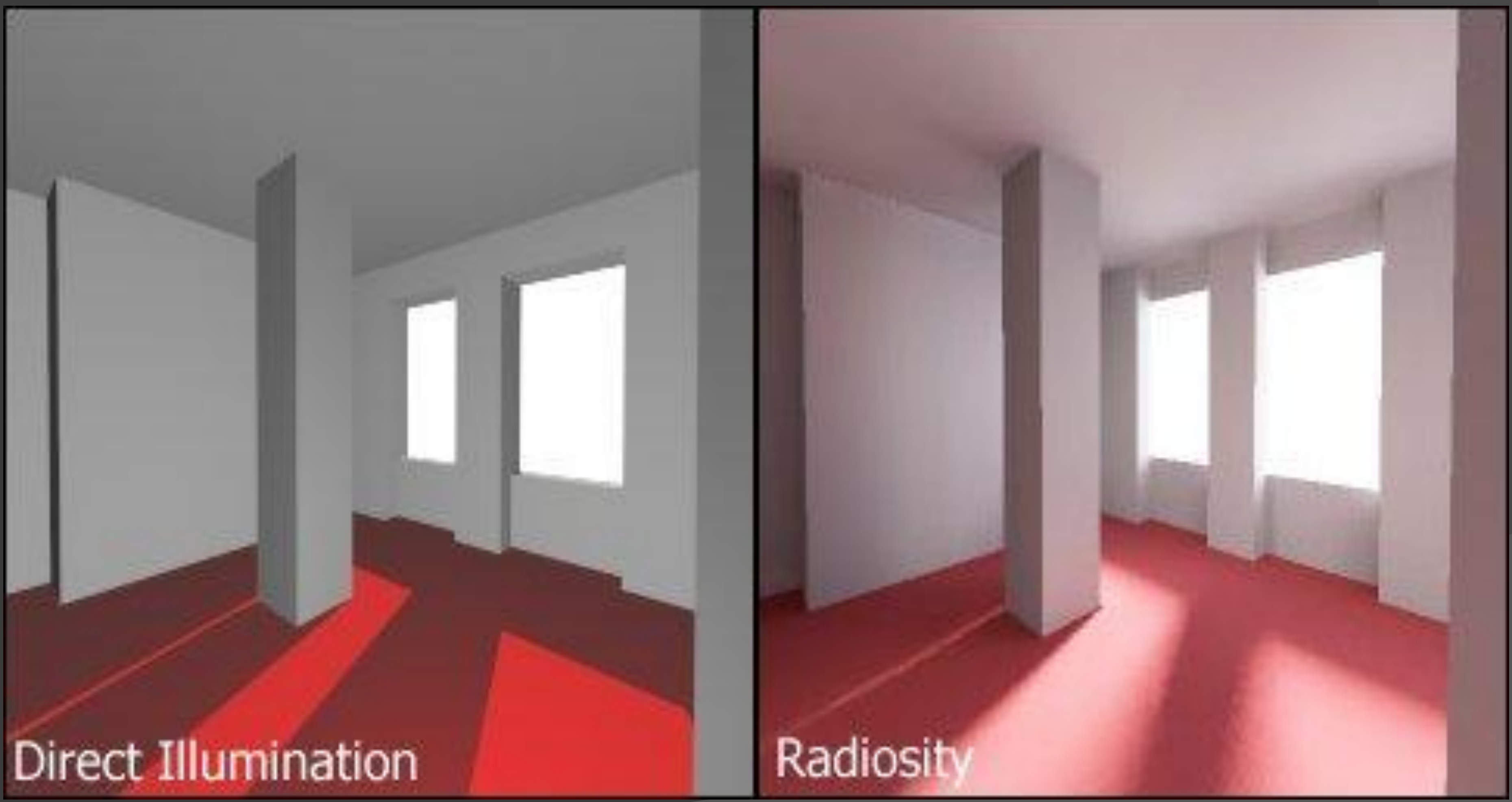
- This stage transforms models and the view into the target place

- The World Coordinate System:
    - the coordinate system in which the virtual world is created
    - Usually a rectangular Cartesian coordinate system in which all axes are equally scaled

Lighting

- A scene often contains light sources placed at different positions
  - Lighting makes virtual world more realistic
- Calculating realistic lighting is very performance intensive
- **Two main types of lighting:**
  - **Local illumination:** light bounces once on the way from light source to camera
    - for real time rendering. Fast, bust less realistic
  - **Global illumination:** simulates real world lighting
    - It is a system that models how light is bounced off of surfaces onto other surfaces (indirect light)
    - Mainly used in case of non real time rendering

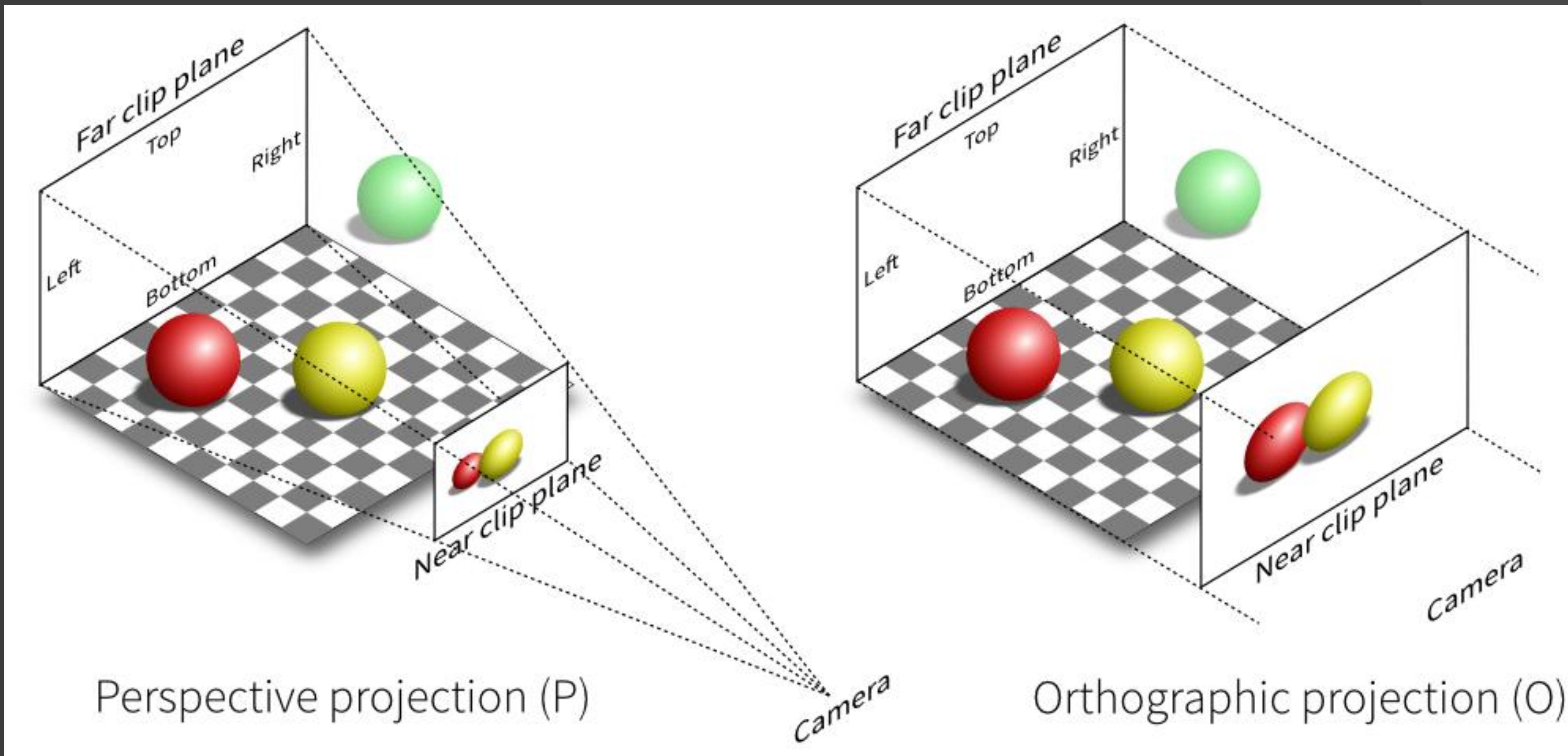
Global vs Local illumination



Projection

- Graphical projection is a protocol by which an image of a three-dimensional object is projected onto a planar surface
- **Two types of projection:**
  - **Parallel:** the lines of sight from the object to the projection plane are parallel to each other.
    - Lines that are parallel in three-dimensional space remain parallel in the two-dimensional projected image.
  - **Perspective:** object positions are transformed to the view plane along lines that converge to a point called projection reference point

Projection

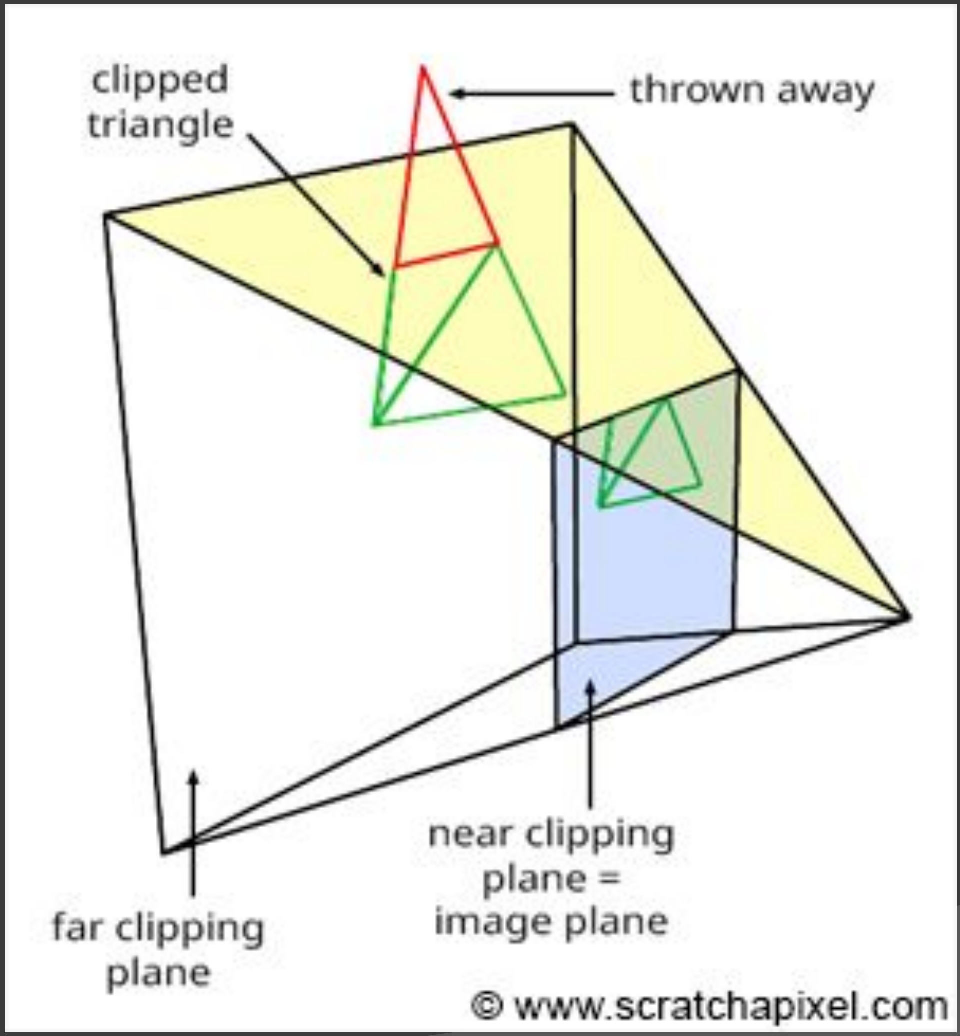


Far clip plane
Top
Right
Left
Bottom
Near clip plane
Camera
Perspective projection (P)
Far clip plane
Top
Right
Left
Bottom
Near clip plane
Camera
Orthographic projection (O)

Clipping

- Only the primitives which are within the visual volume need to actually be rastered
- This visual volume is defined as the inside of a frustum, a shape in the form of a pyramid with a cut off top
- Primitives which are completely outside the visual volume are discarded;
  - This is called **frustum culling**

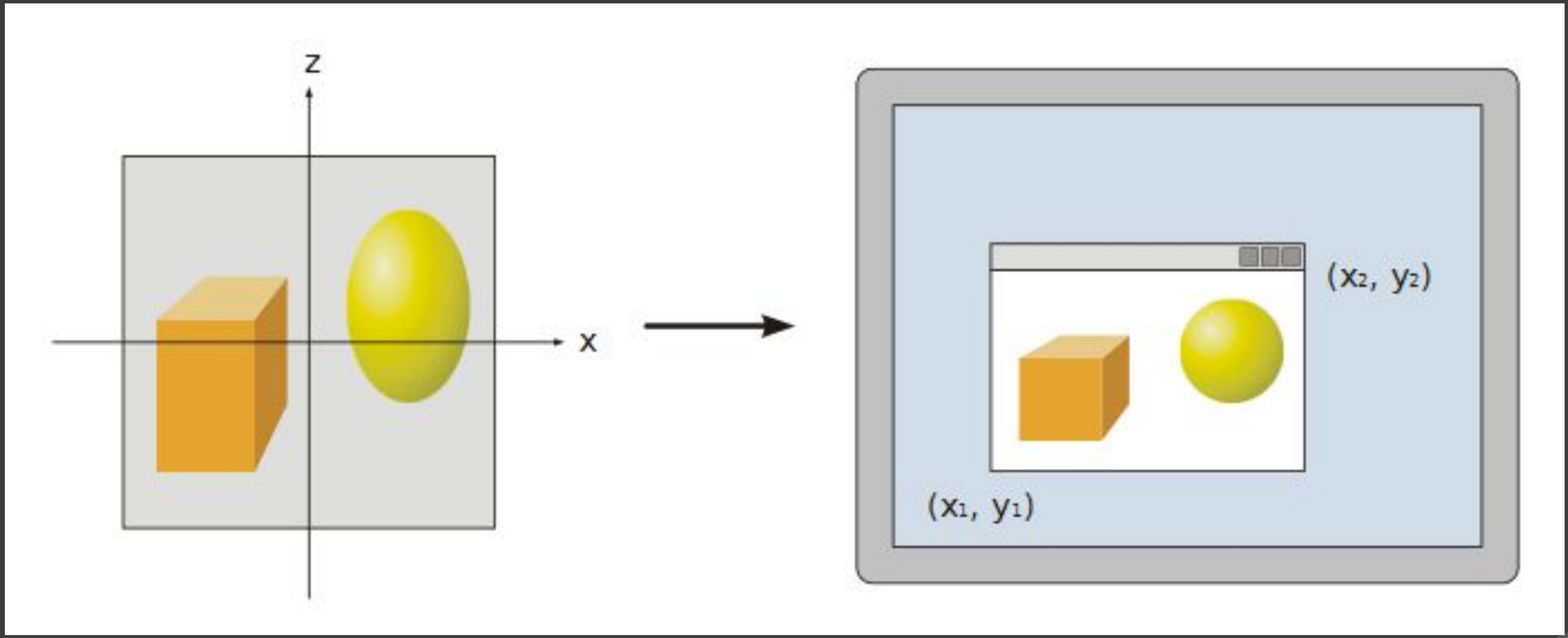
Clipping


clipped
triangle
thrown away
far clipping
plane
near clipping
plane =
image plane
© www.scratchapixel.com

Window-Viewport transformation

- In order to output the image to any target area (viewport) of the screen, Window-Viewport transformation must be applied
- **This is a shift, followed by scaling**
- The resulting coordinates are the device coordinates of the output device
  - The viewport contains 6 values:
    - height and width of the window in pixels,
    - the upper left corner of the window in window coordinates (usually 0, 0)
    - and the minimum and maximum values for Z (usually 0 and 1)

Window-Viewport transformation



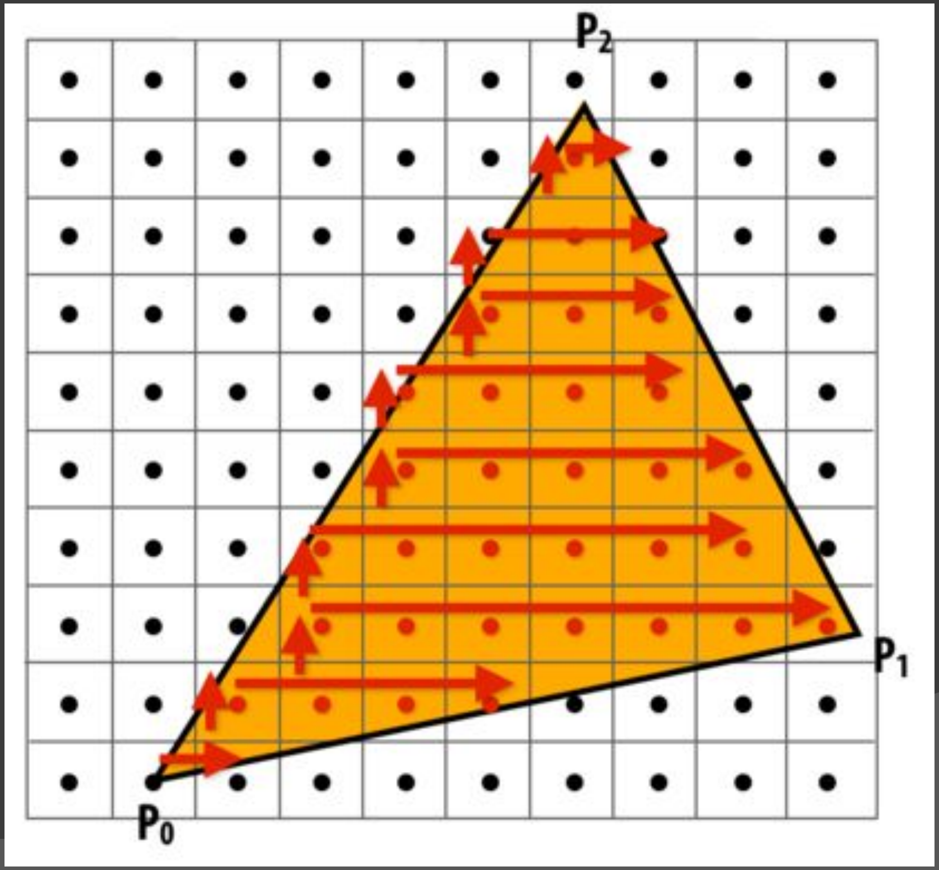
Rasterization

- This is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (pixels or dots)
  - the grid points are also called fragments
  - discrete fragments are created from continuous surfaces
  - Each fragment corresponds to one pixel in the frame buffer and this corresponds to one pixel of the screen

- Fragments can be colored and illuminated.
- Furthermore, it is necessary to determine the visible, closer to the observer fragment, in the case of overlapping polygons
  - Visibility algorithms

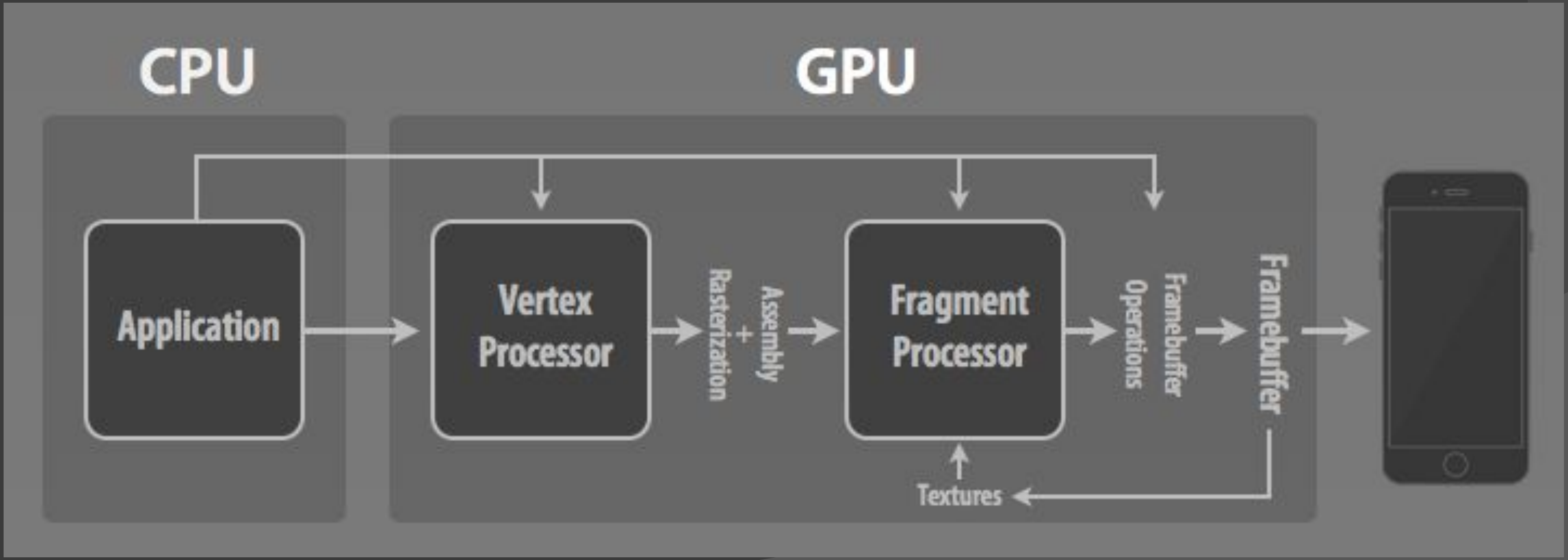
Rasterization

- Rasterization is usually performed by polygon (usually triangle) filling:



Modern GPU pipeline

- Modern GPUs use programmable pipeline
- **<u>Why programming is needed?</u>**
  - We can customize the pipeline stages via *vertex* and **fragment shaders**



Modern GPU pipeline...

Modern GPU pipeline

- Programmable pipeline allows developers to create customized programs
  - Using **Shading languages**
- With shaders we can extend the graphics card's pipeline functionalities
  - We are able to achieve several "non API supported" features
    - E.g.: Own lighting and shadowing algorithms
- The shading language code that is intended for execution is called **shader**
- Because two programmable processor is defined we have:
  - Vertex shader (vertex program)
  - Fragment shader (fragment program)

The vertex processor

- The vertex processor is a configurable hw unit
  - It operates on incoming vertex values and their associated data
- The vertex processor usually performs traditional graphics operations such as:
  - Vertex transformation
  - Normal transformation and normalization
  - Texture coordinate generation
  - Texture coordinate transformation
  - Lighting
  - Color material

The vertex processor

- Because of its general-purpose programmability, this processor can also be used to perform a variety of other computations
- The vertex processor is responsible for running the vertex shaders
- The input for a vertex shader is the vertex data:
  - its position, color, normals, etc, depending on what the application sends
  - There are called: **vertex attributes**
- Vertex programs are applied to each vertex of a model
  - But vertex program cannot create new vertices!

The fragment processor

- The fragment processor is a programmable unit
- It is responsible for running the **fragment shaders**
- **How it works?**
  - Fragment programs operate on each generated fragment (pixel)
  - The task of a shader is to take the fragment attributes and uniform parameters as input
  - and compute a final color for that fragment which will be written to the render target (e.g. screen)

The fragment processor

- The fragment attributes are the interpolated attributes of the associated vertices
- It usually performs traditional graphics operations such as the following:
  - Calculation of accurate lighting models
  - Post-processing effects like glow and depth-of-field
  - Texture access and application
  - Fog
  - Color sum
  - Alpha blending
- A wide variety of other computations can be performed on this processor

Shading language...

Shader language

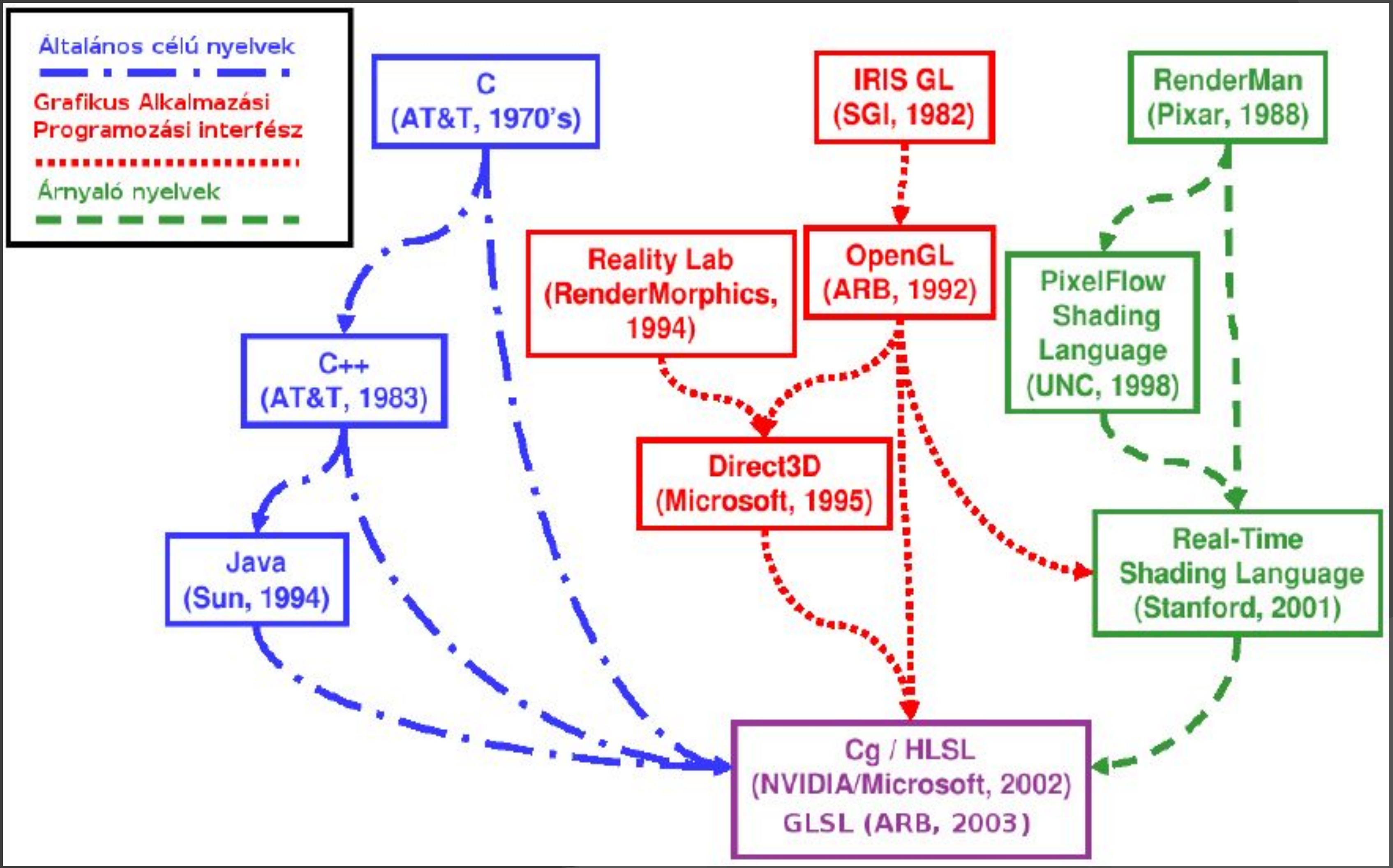
- Shading languages are usually used to program the programmable GPU rendering pipeline
- Makes possible to the programmer to replace the fixed function pipeline
- **Properties of these languages:**
  - Initially there were no high-level languages,
  - only in the Assembly language it was possible to write a shader program
  - Today's languages are the result of many years of development
  - Three main direction of the evolution:
    - 1. General programming languages
    - 2. Graphical interface languages
    - 3. The shader languages

Shader language

- <u>The common language of the programs is C language</u>
  - In terms of the syntax and semantics
  - today's major shading languages are based on this

- **<u>Known languages:</u>**
  - CG (C for Graphics) – NVIDIA
    - One of the first GPU shading languages
  - **GLSL** (GLslang) – OpenGL
  - HLSL (High Level Shading Language) – MICROSOFT
    - XBOX – XNA – DirectX

Evolution of shading languages



OpenGL Assembly Language
(Sample (old) fragment shader)

```
!!ARBfp1.0
TEMP color;
MUL color, fragment.texcoord[0].y, 2.0;
ADD color, 1.0, -color;
ABS color, color;
ADD result.color, 1.0, -color;
MOV result.color.a, 1.0;
END
```

GLSL Language
(Simple color fragment shader)

```
#version 330
out vec4 outputFragment;

void main() {
    outputFragment = vec4(0.4,1,1,1);
}
```

GLSL overview (short)…

GLSL language

- OpenGL Shading Language (often **glslang**)
- Developed by the OpenGL ARB group as part of the OpenGL 1.4 extension
- From the OpenGL 2.0 it was integrated into the standard

- **Main Characteristics:**
  - High level language,
  - Based on C language syntax
  - Makes possible to program the pipeline directly

GLSL language

- **Main Characteristics:**
  - Platform independent. Supported by GNU/Linux, Unix, BSD, Windows and Mac OS X
  - Shaders written in GLSL can be used on any graphics card that supports GLSL
  - Each graphics card driver includes the GLSL compiler,
    - Card manufacturers can optimize the code generated by the compiler according to the card architecture
- The shader programs are primarily based on the data-parallelism,
  - the way the parallels depend on the implementation of the driver
    - Different optimizations
  - communication between parallel running programs is not supported

GLSL language

- The programs are represented as a text form
- We write vertex and pixel shaders during coding
- They can be served into different files or into the main application code as a string

- **<u>But they are served into files:</u>**
  - E.g.: example.vert, example.frag
  - <u>File extension is not relevant</u>
    - The compiler decides based on the content

GLSL language

Using the shaders:

- Loading the shader files must be done manually by the programmer
  - OpenGL does not provide a direct support for this
- **What does it really mean?**
  - We only need to load the shaders into memory manually
  - From here, OpenGL gives the option of creating a real shader object

The geometry world…

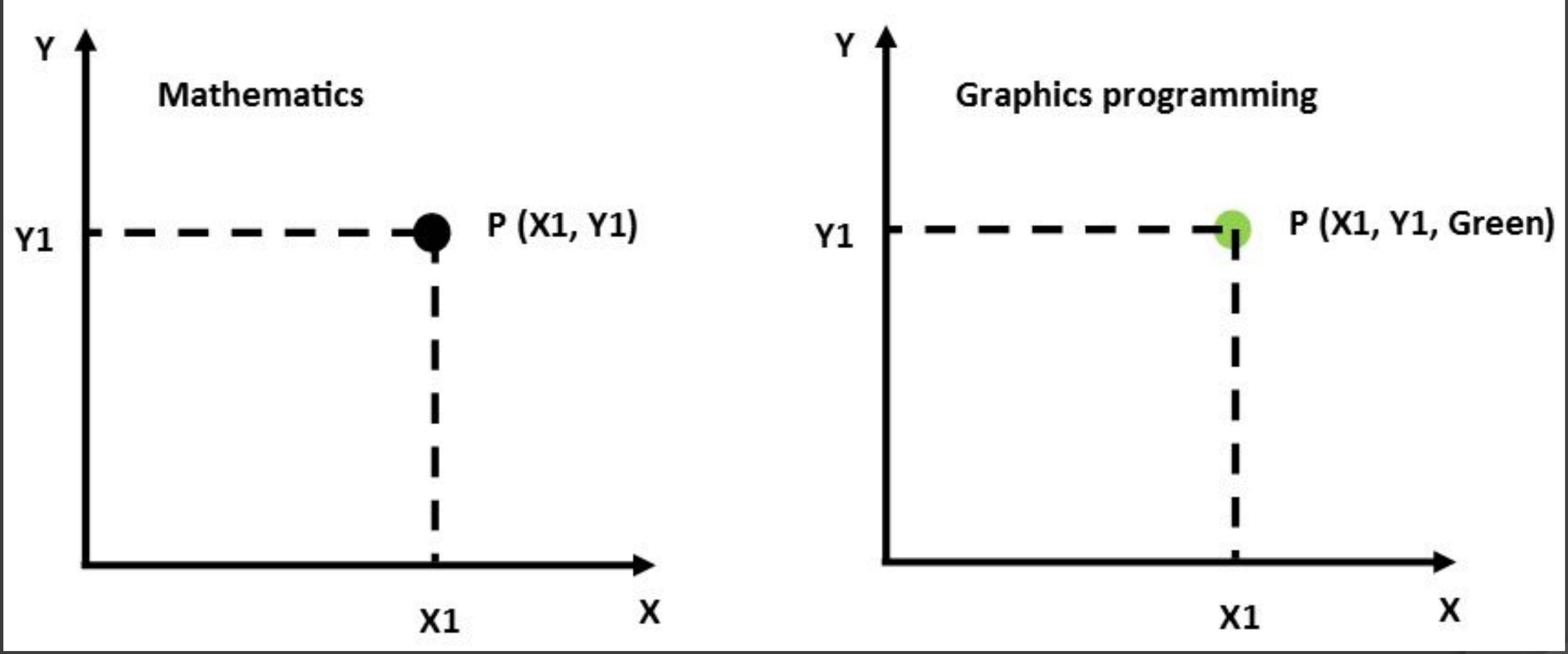
The geometry world

- Computer graphics is built from **geometry**
  - What we want to render to the screen
  - These are so-called **geometric primitives**
    - They can be used to generate the desired geometry
- Today's most common approach to represent a model is polygonal modeling
  - It is an approach for modeling objects by representing or approximating their surfaces using polygons
    - A geometry can be defined by points, lines, triangles, quads, triangle strips, etc.
  - Example:
    - A square can be composed out of 2 triangles
    - A triangle can be composed from 3 points

The geometry world

- A 3D point is represented as a **vertex**
  - Two vertices defines a line and become an edge
  - Three vertices with three edges define a triangle
- Rendering the geometry primitives:
  - We need to define the vertices
  - These points will then reside in system memory
  - The GPU will need access to these points
    - The application will use the 3D API to transfer the defined vertices from system memory into the GPU memory.
    - Also note that the order of the points can not be random

Vertex

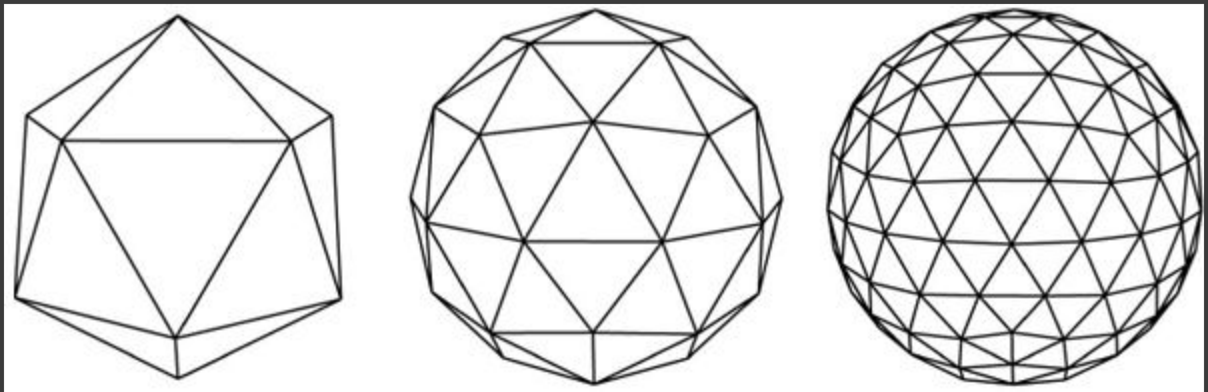


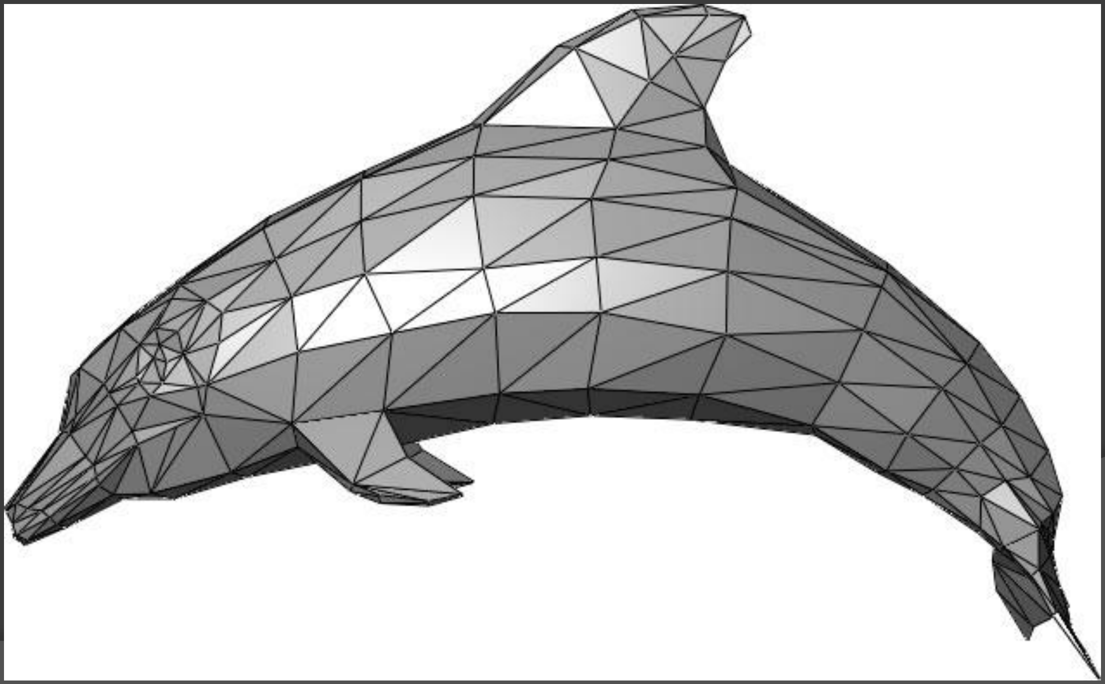
- A vertex can have several attributes, like:
  - Color, texture coordinate, normal, etc

The geometry world

- In games, complex polygon models are used
  - They are built from (a lot of) triangles
    - A modern game can contain 1M+ triangles
- Why triangle?
  - is the simplest polygon in Euclidean space
  - Graphics card can handle them effectively

The geometry world





The geometry world in practice

- We can call the highest level geometry structure as a **model**
  - Contains everything
- In practice, models should be logically subdivided into **objects**
  - It is usually called: **mesh**
- <u>Main characteristics of an object:</u>
  - They are individually renderable
  - Material and effect property varies usually by object

The geometry world in practice

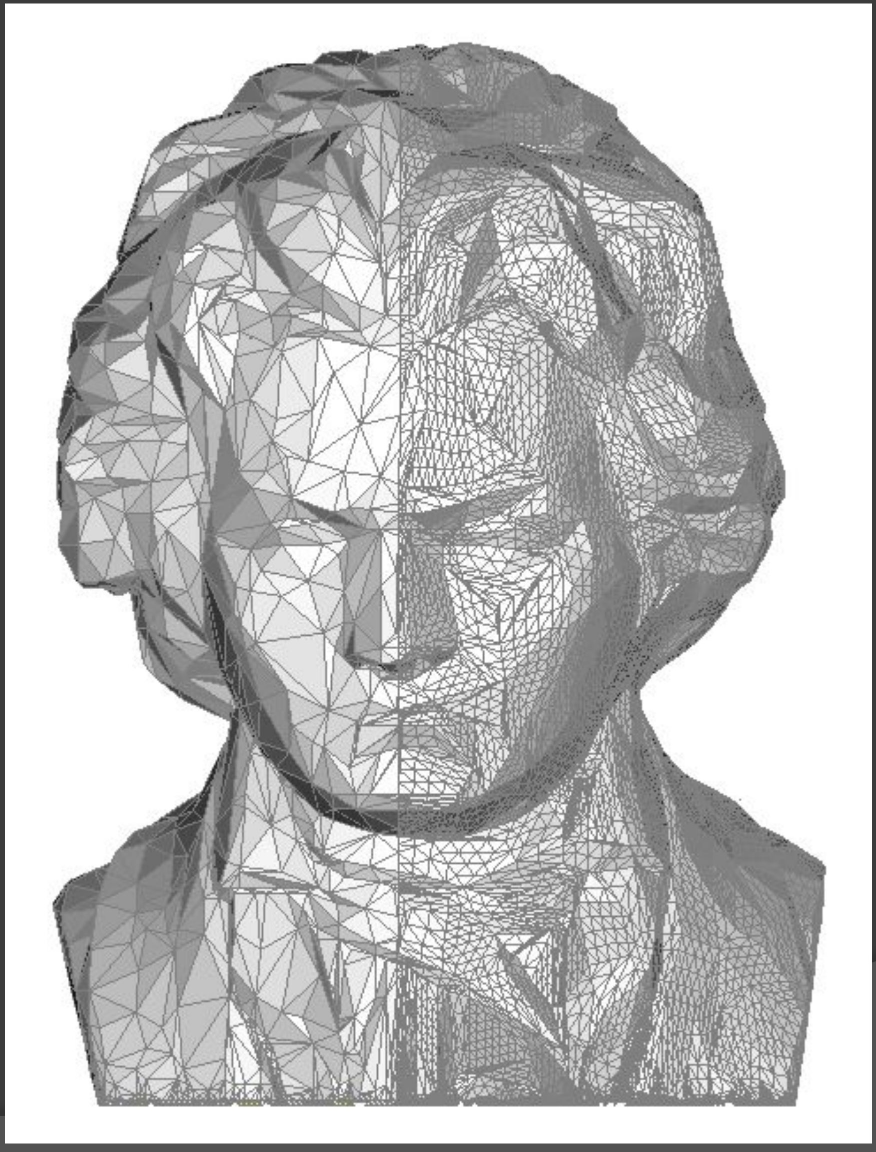
- <u>**The reason of the logical separation:**</u>
  - It is not appropriate to more a complex model as a large set of vertices
    - Better is to divide the model into several logical objects
  - Applying logical separation is also preferred in 3D Modelling softwares
  - <u>**Why?**</u>
  - This way it is easier to handle parts that belong together, but still represented as separate units
    - E.g.: modifying, replacing the parts

The geometry world in practice

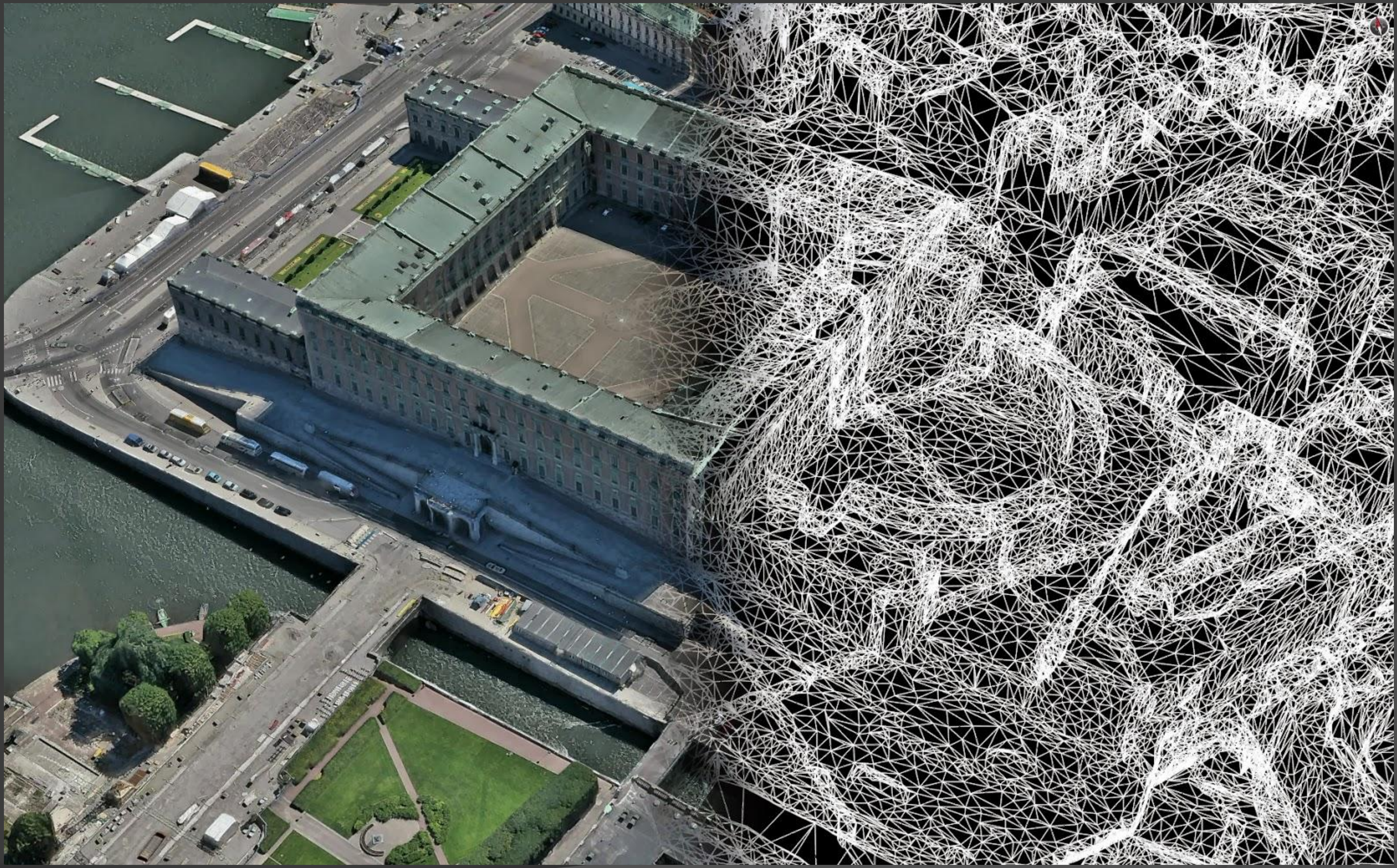
Example: A model of a car

- It is advisable to design the wheels of a car as a separate object
  - Because it can move, rotate, etc
- The logical units can have names and other properties
  - During the implementation it is better to refer to an object by its name

The geometry world



The geometry world

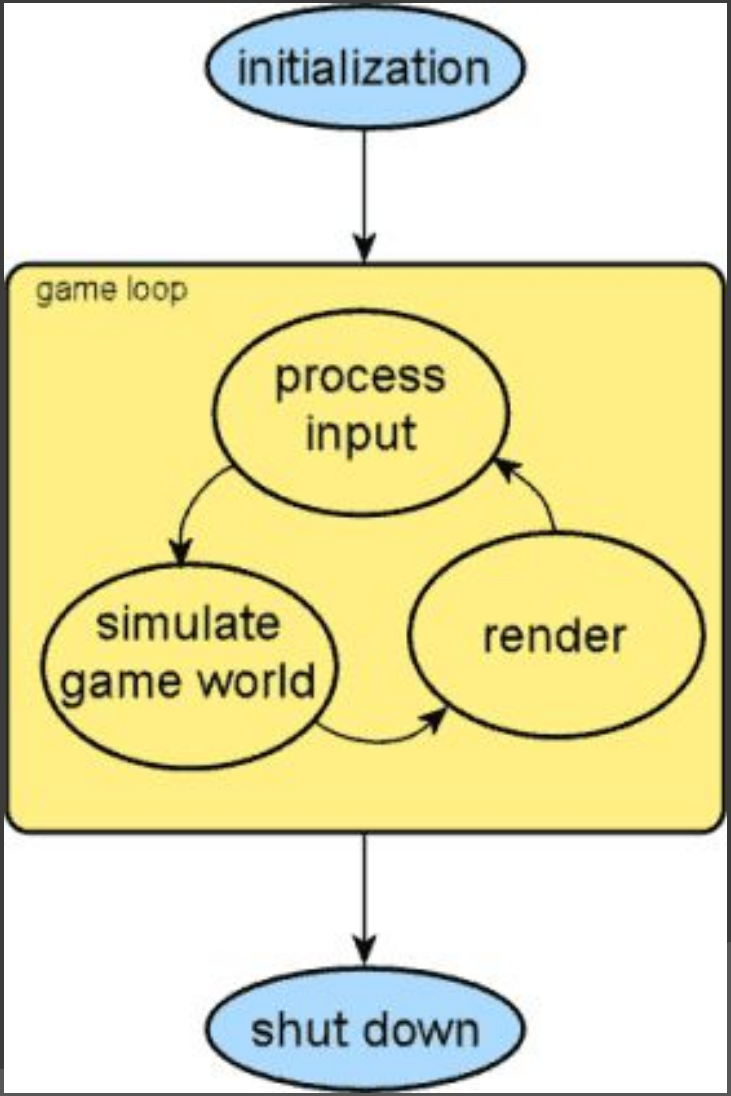


The rendering / game loop...

Rendering Loop / Game loop

- **Loop:** Graphics are repeatedly drawn on screen and interactive (frames)
- This is real-time rendering
- This is the type of rendering used in games
- This style of rendering contrasts offline rendering
  - where single images or frames are calculated over a long period of time
- Rendering loop should reach **50-60** Frames per Secundum (FPS)

Rendering Loop / Game loop



Rendering Loop / Game loop

Initialization:

- Choosing an OpenGL profile and configuring capabilities for a rendering context
- Creating a window and an OpenGL Context
- Loading resources needed by program

Process Input:

- Listen for mouse and keyboard events
- Update user's view (often called a camera)

Rendering Loop / Game loop

Update (Simulate Game World):

- Calculate geometry
- Rearrange data
- Perform computations

Render:

- Draw scene geometry from a particular view

Shut Down:

- Save persistent data
- Clean up resources on graphics card

GAME OVER