§1 Basics

1.1 Basics

1.1 Basics

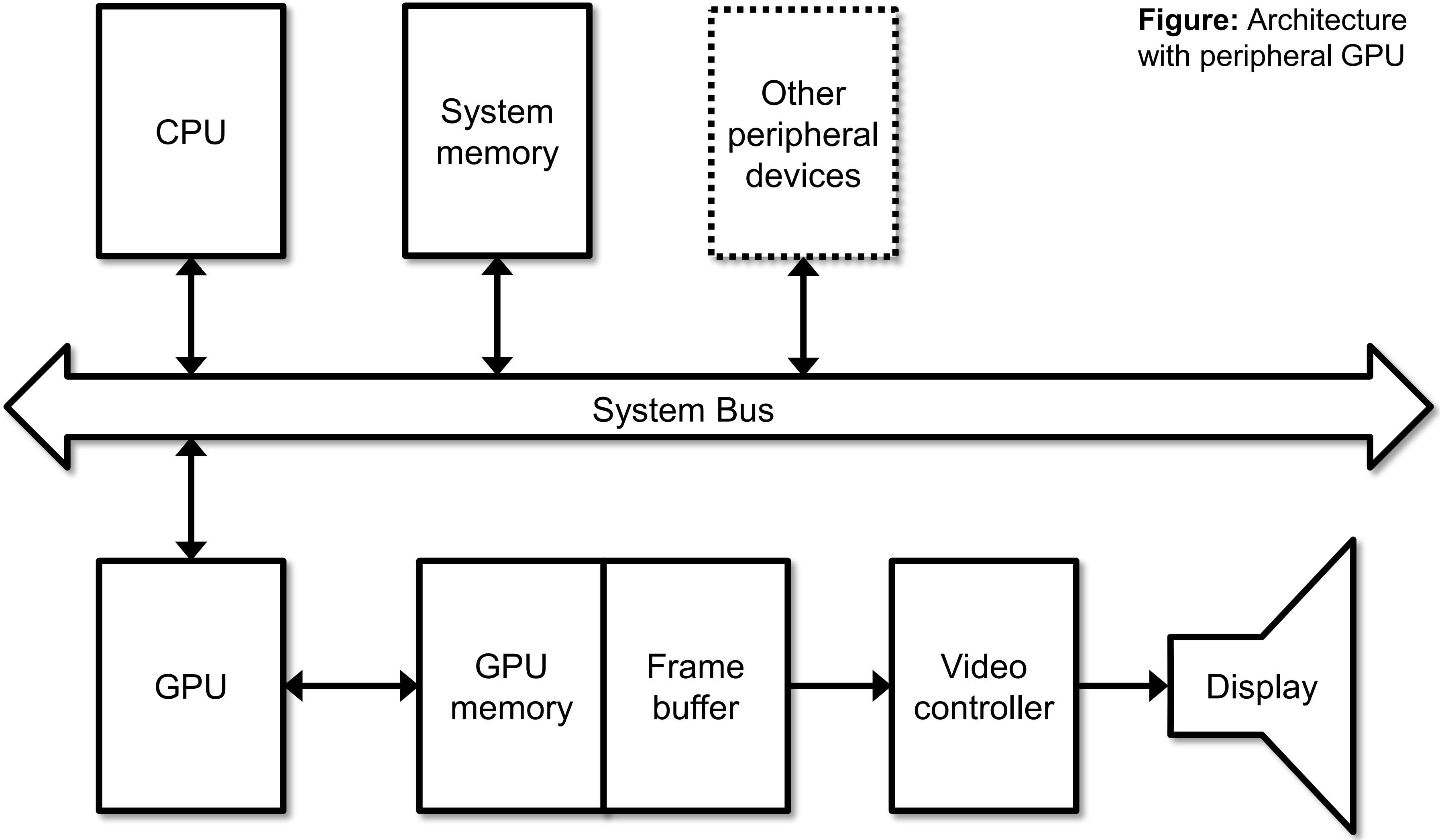
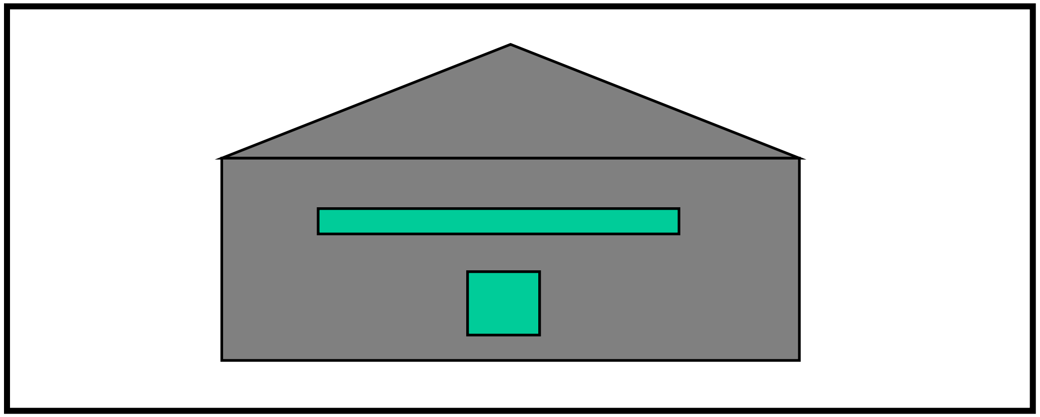


Figure: Architecture with peripheral GPU

1.1 Basics

Frame buffer

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	2	2	2	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



- **Double buffering:**
  - Write to one buffer…
  - … read from the other buffer for display.

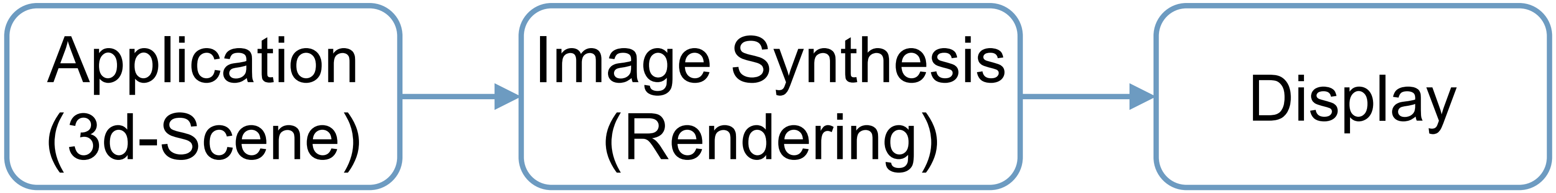
1.1 Basics

The computer graphics-pipeline / rendering-pipeline

- The process of image synthesis,
  - i.e., mapping of the geometric model, the object(s), or the scene to an image on the display (output device),

  is called ***rendering***.
- A concrete implementation of this process in soft- and/or hardware is called the ***rendering-pipeline***.
  - The individual stages of this pipeline are realized by the basic algorithms of computer graphics.
  - The individual stages can be implemented in soft- and/or hardware!
  - The structure of the rendering-pipeline can vary drastically depending on the type and realization of the rendering.

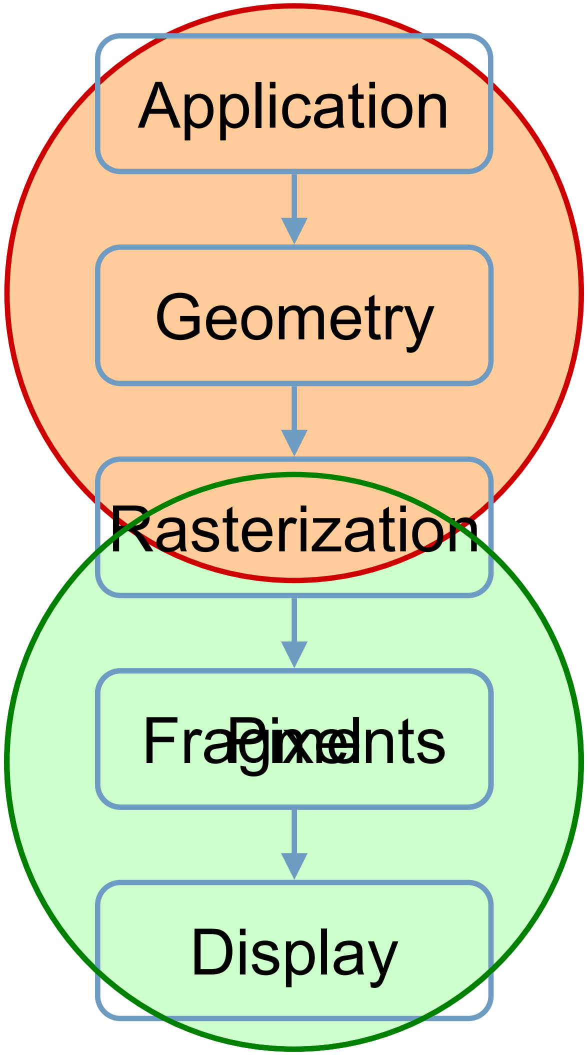
1.1 Basics



Rendering Pipeline

- Subdivide the rendering into simple standard stages.
- Dependent on
  - Hardware of output device (screen, graphic card, etc.),
  - Algorithm for image synthesis (illumination, shading, etc.), etc.
- Some stages can be missing in a concrete realization or occur in a different order.
- De-facto standard: OpenGL Rendering Pipeline, etc.

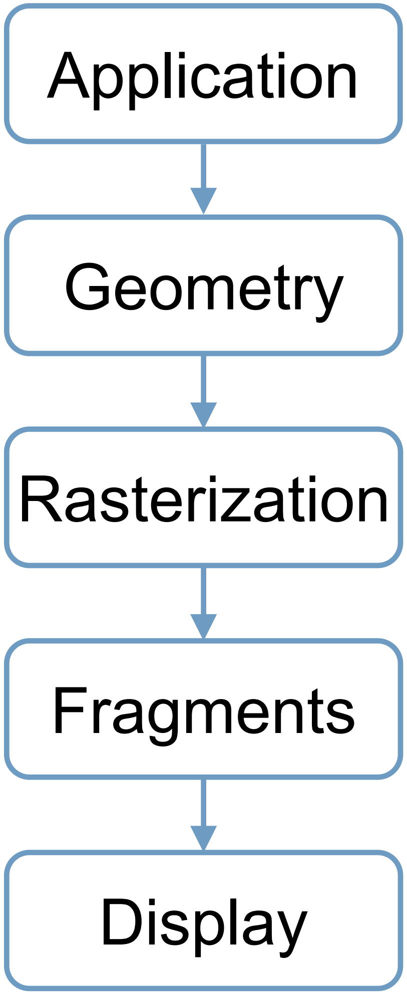
1.1 Basics



Manipulation of geometric objects and primitives (vertex).

Manipulation of images and image points (fragment/pixel).

1.1 Basics



Classical graphics-pipeline:

- The middle components are static.
- They are realized in soft- and/or hardware.

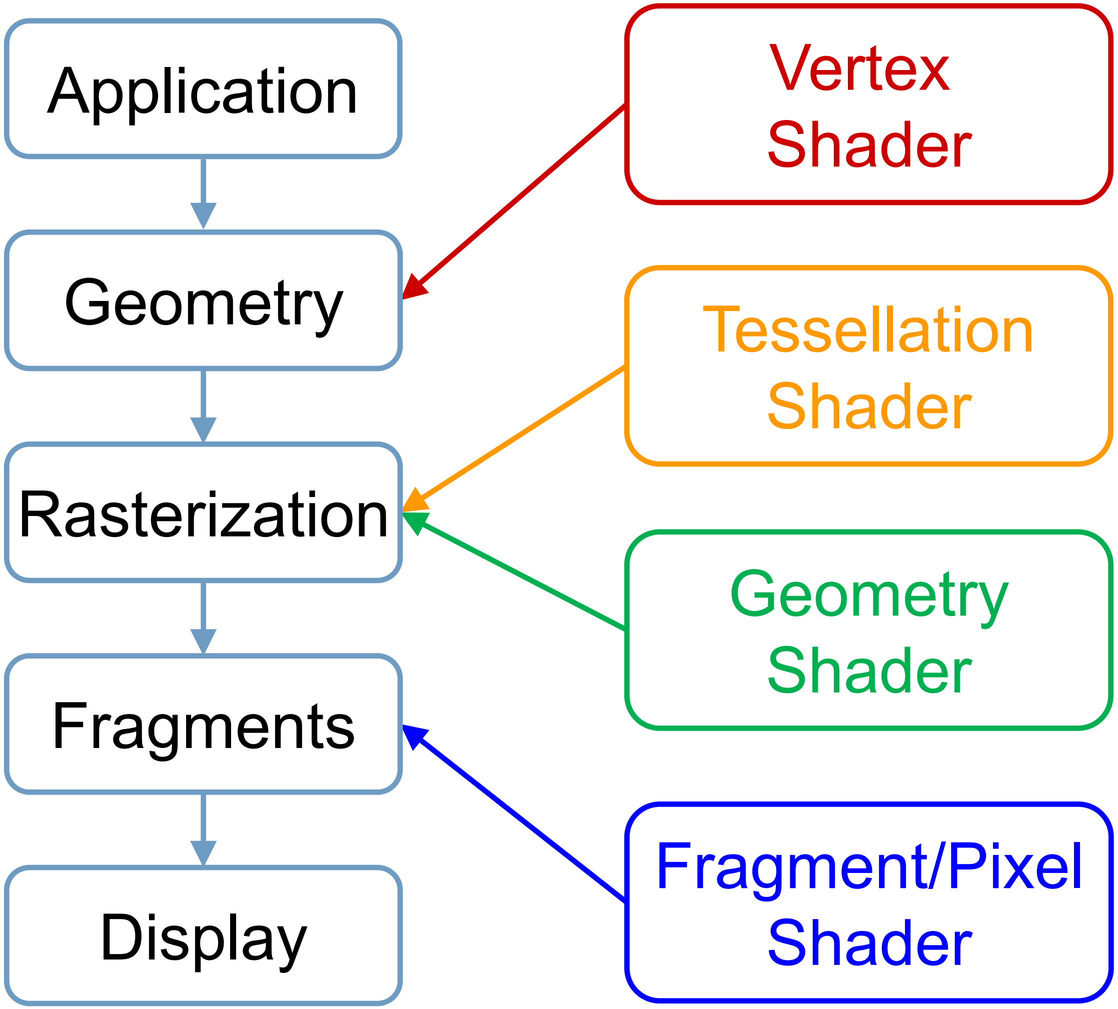
Modern graphics-pipeline:

- The middle components are dynamic.
- They are realized in **shaders**.

Shader:

- Programs, that run directly on the graphics hardware.

1.1 Basics



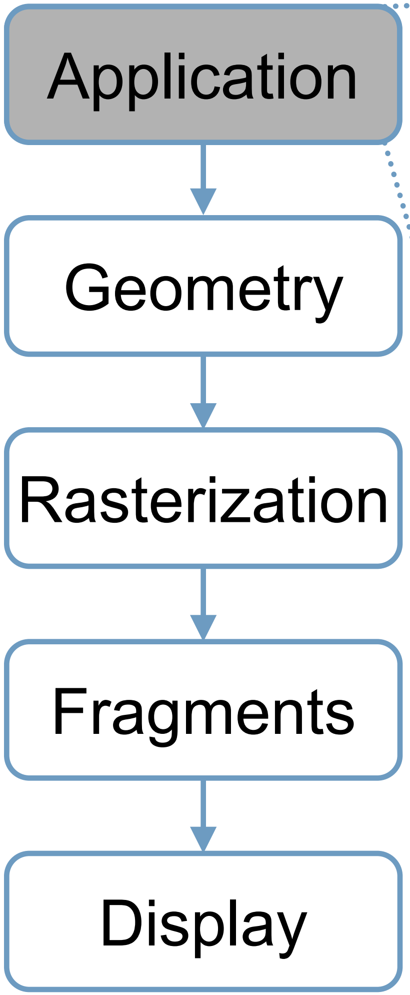
Manipulation of geometric objects and primitives (vertex).

Subdivide the geometry into primitives (triangles).

Compute new geometry from given geometry.

Manipulation of images and image points (fragment/pixel).

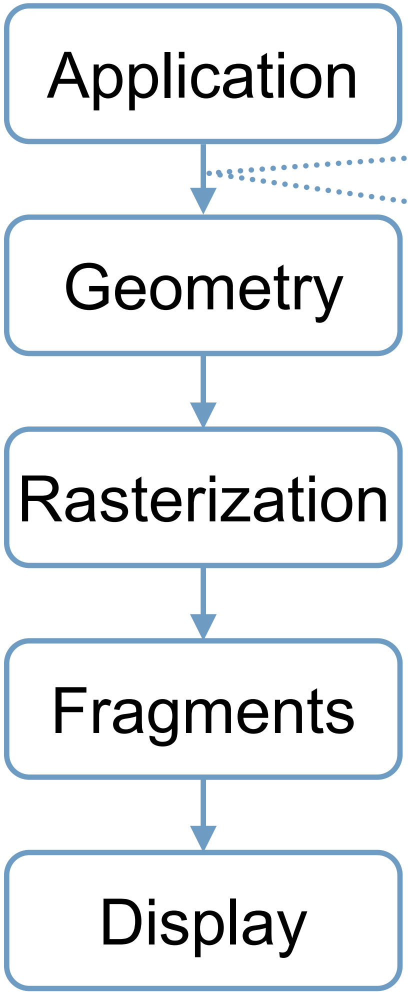
1.1 Basics



Tasks:

- Data base query
- Input processing
- Generation of scene (§6.7):
  - Construction, collision detection, simulation, animation, etc.
- Acceleration techniques:
  - Space partitioning (§6.4), etc.
- Subdivision into primitives:
  - Triangulation (§6.2), etc.
- Auxiliary functions for spline evaluation, etc.

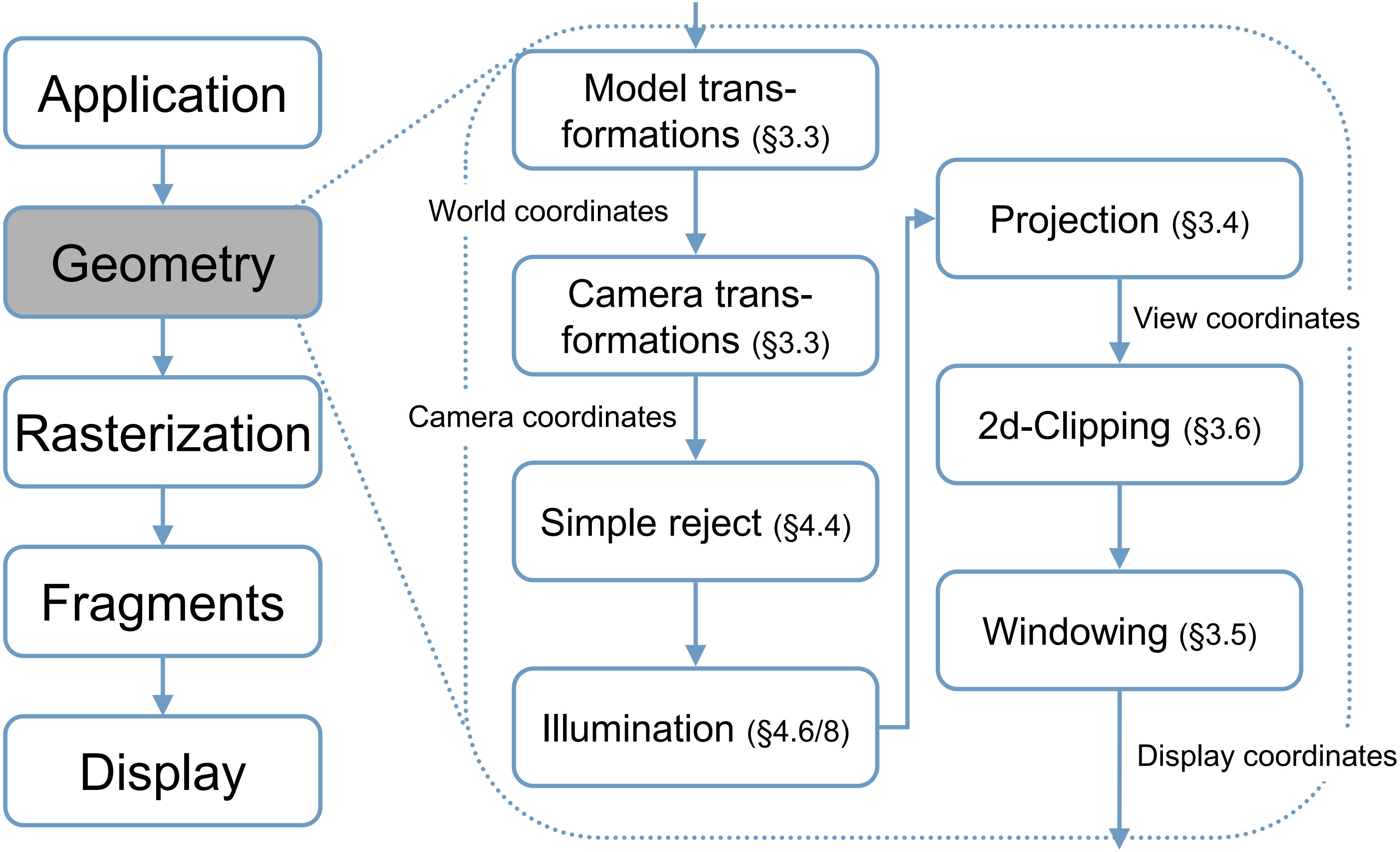
1.1 Basics



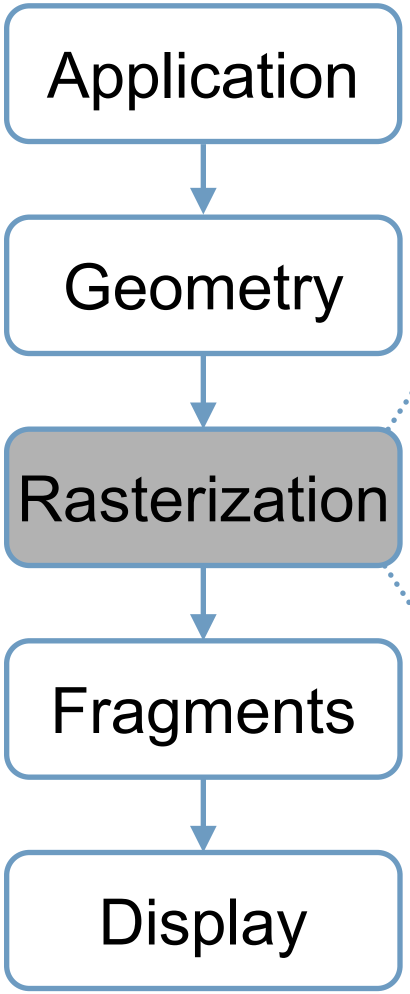
Bus interface:

- Receiving and sending of commands and data
  - Triangles, normals, material properties, etc.
- Command buffering
- Command interpretation

1.1 Basics


Application
Geometry
Rasterization
Fragments
Display
Model trans-formations (§3.3)
World coordinates
Camera trans-formations (§3.3)
Camera coordinates
Simple reject (§4.4)
Illumination (§4.6/8)
Projection (§3.4)
View coordinates
2d-Clipping (§3.6)
Windowing (§3.5)
Display coordinates

1.1 Basics


Application

Geometry

Rasterization

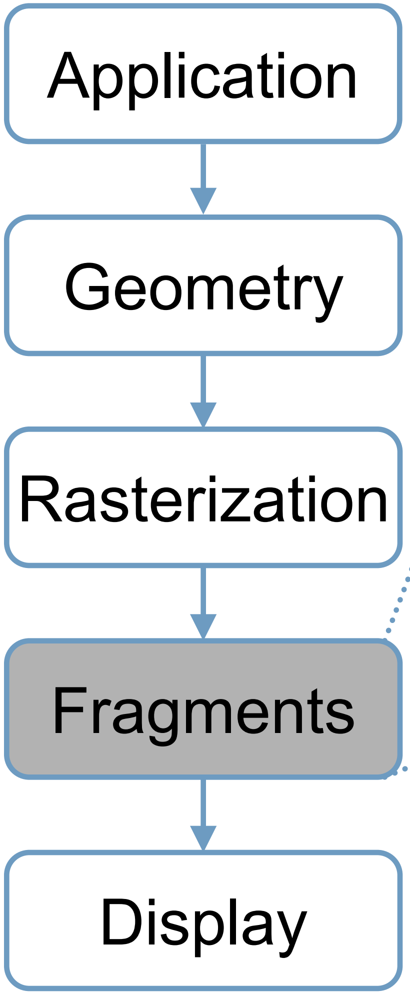
Fragments

Display

Tasks:

- **Primitive assembly**
  - Triangles, lines, etc.
- **Triangle setup**
  - Interpolation of
    - depth values (§4.4),
    - color values (§4.7),
    - texture coordinates (§5), etc.
- Rasterization (§2.2-5)
- Anti-Aliasing (§2.8)
- Z-Buffer (§4.4)

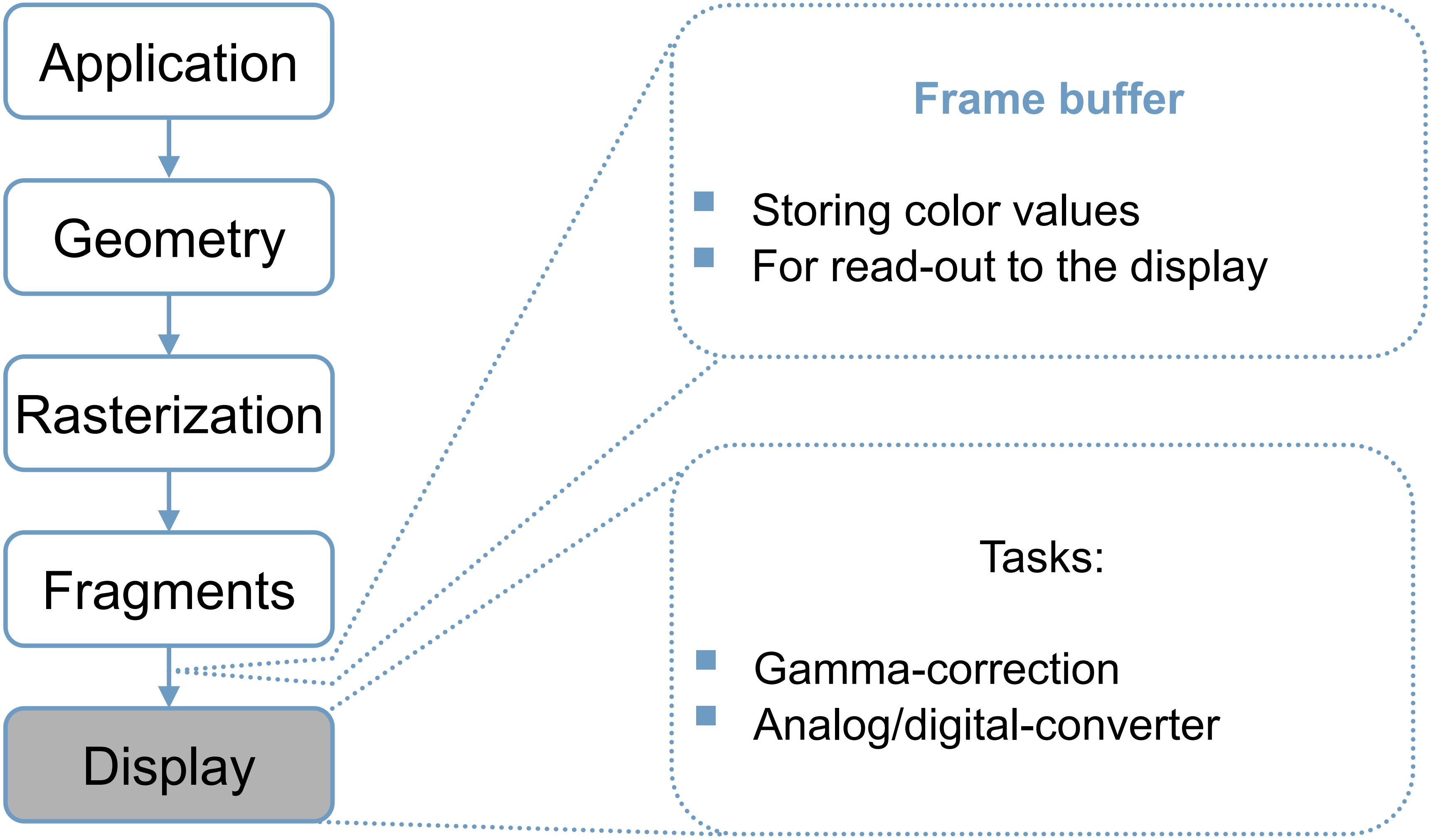
1.1 Basics



Tasks:

- Transparency (§4.4)
- Texturing (§5)
- Stencil tests
- Fog, etc.
- Dithering, color reduction

1.1 Basics



Application

Geometry

Rasterization

Fragments

Display

Frame buffer

- Storing color values
- For read-out to the display

Tasks:

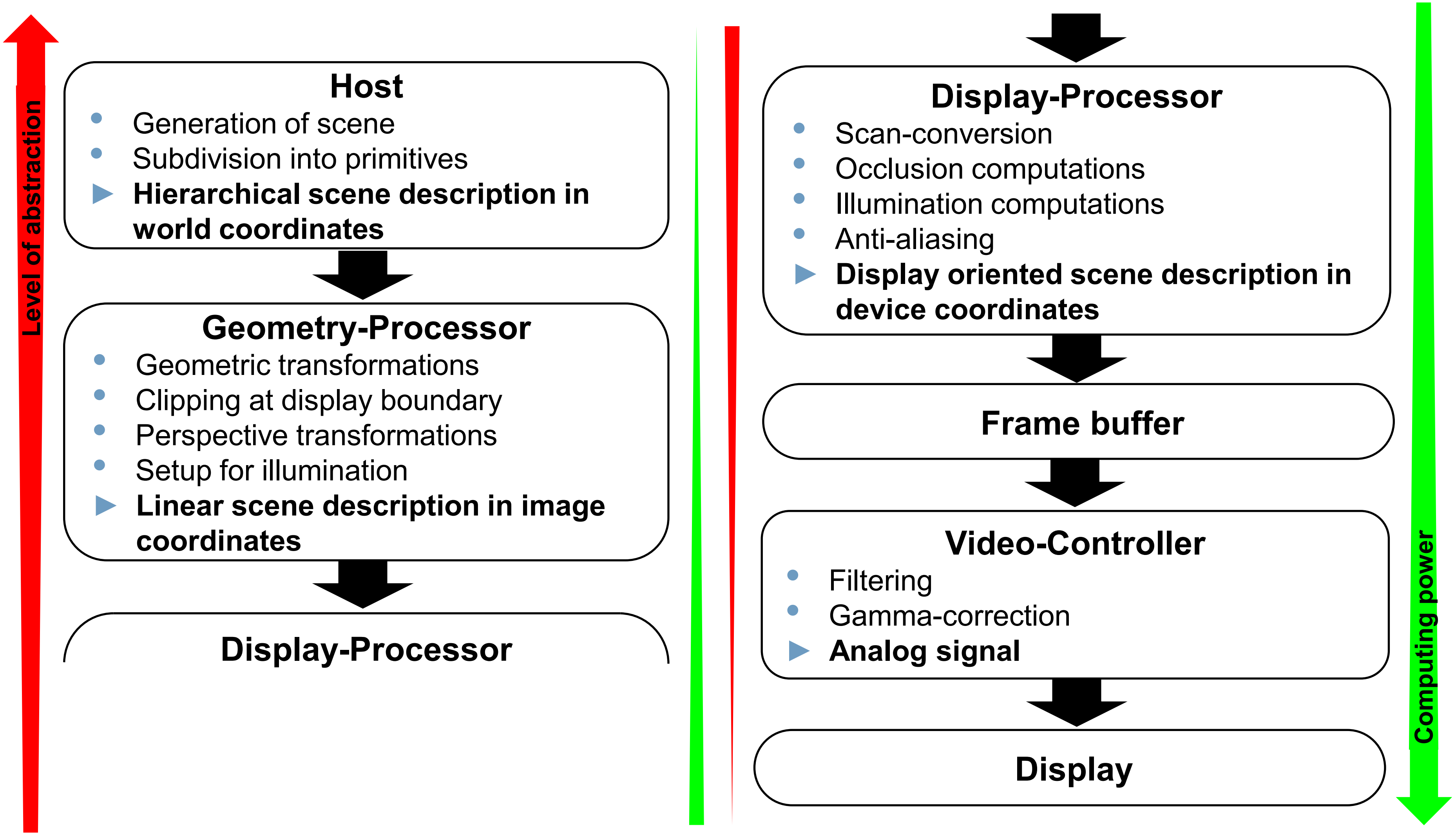
- Gamma-correction
- Analog/digital-converter

1.1 Basics

In Chapter §4.9 we will see different realizations of rendering-pipelines, depending on

- the **visibility algorithms** (§4.4), ← **Which objects do we see?**
- the **illumination model** (§4.6 & §4.8), ← **What is the color of the objects, that we see?**
- the **shading model** (§4.7). ← **What it the color of the pixels covered by the objects, that we see?**

1.1 Basics



Host

- Generation of scene
- Subdivision into primitives
- ▶ **Hierarchical scene description in world coordinates**

Geometry-Processor

- Geometric transformations
- Clipping at display boundary
- Perspective transformations
- Setup for illumination
- ▶ **Linear scene description in image coordinates**

Display-Processor

Display-Processor

- Scan-conversion
- Occlusion computations
- Illumination computations
- Anti-aliasing
- ▶ **Display oriented scene description in device coordinates**

Frame buffer

Video-Controller

- Filtering
- Gamma-correction
- ▶ **Analog signal**

Display

1.1 Basics

Alternatives to OpenGL

- **DirectX:** only for Windows-Platforms (e.g., xbox)
  - Large API collection for multimedia-applications:
  - 2d-graphics, 3d-grafik, audio, various input devices, etc.
  - Compute shaders, etc.
- **Vulkan:** (aka Next Generation OpenGL)
  - Derived from Mantle (AMD)
  - Low-level rendering API, open, cross-platform
  - Improved integration of CPU-GPU-communication
  - High-performance, multi-threading
  - Debugger (GLAVE)
- **Metal:** only for Apple-Hardware
  - Low-level, low-overhead rendering API

1.1 Basics

Shader languages

- OpenCL
- GLSL
- CUDA
- Cg
- etc.

Goals

- What are the typical stages of a rendering pipeline?