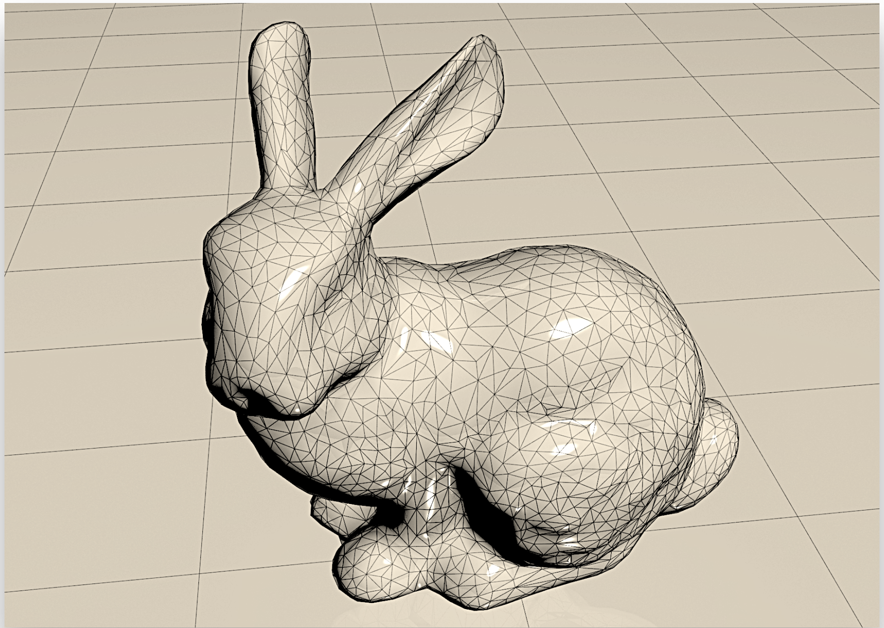
Graphics Systems (2)

Contents

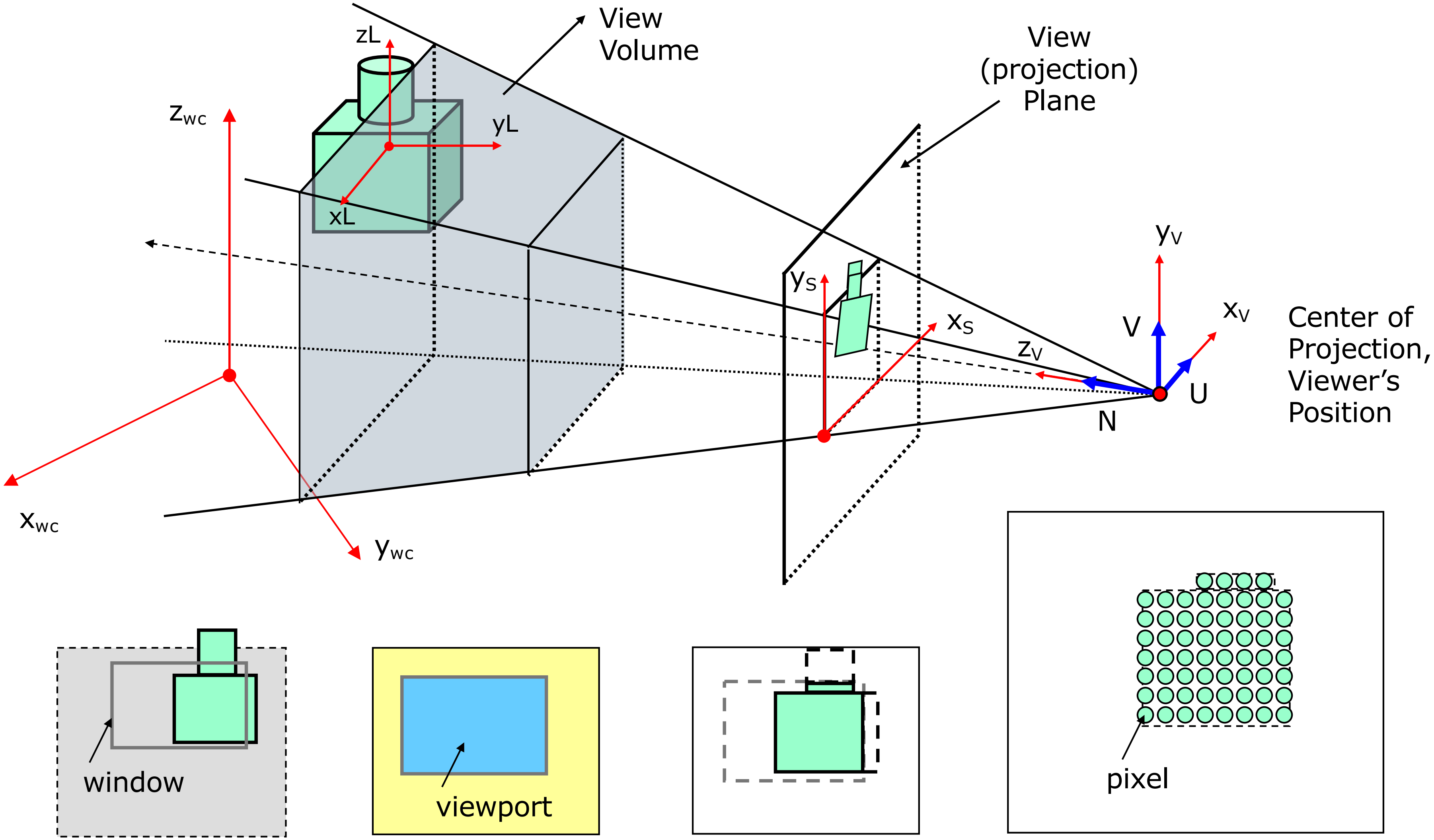
1. Graphics rendering pipeline
2. Evolution of graphics pipeline
3. Rendering strategies
4. Real time rendering pipeline

Graphics rendering pipeline

- Generate (render) a two-dimensional image, provided with a virtual camera, three-dimensional objects, light sources, shading equations, textures, etc.
- Consists of several stages, and the speed is determined by the slowest stage



Graphics rendering pipeline



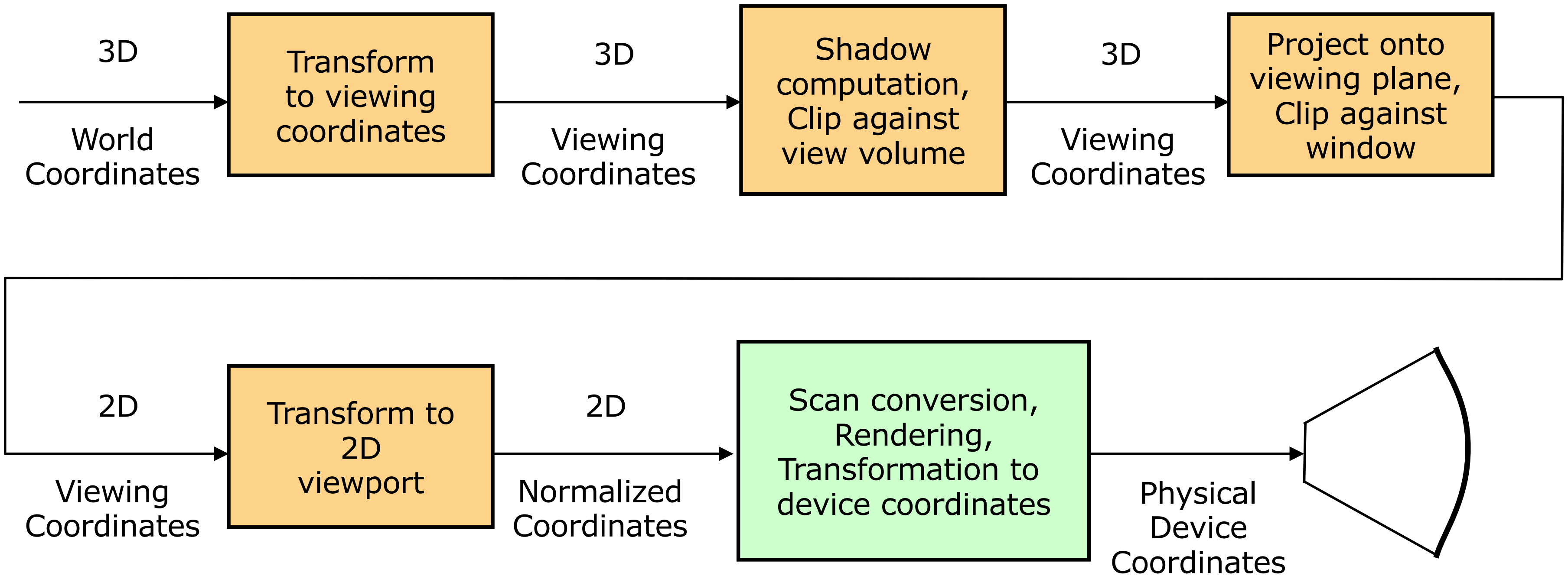
Window definition
in projection plane

Viewport definition
in projection plane

Window's content
inside the viewport

Scan conversion onto the
raster screen

Graphics rendering pipeline



Evolution of graphics pipelines (GPU)

- Fixed-Function pipeline (1992 – 2001)
  - Configurable via parameters
  - Cannot change the algorithms (e.g. Gouraud or Phong shading)
  - OpenGL 1, Direct3D 2
- Hybrid pipeline (2001 – 2009)
  - Shaders (HLSL/Cg – Microsoft/NVIDIA and GLSL - OpenGL)
  - Fixed and programmable features co-exist
  - OpenGL 1.4, Direct3D 8
- Programmable pipeline (2009 – present)
  - No more fixed functions
  - OpenGL 3.2, Direct3D 10, CUDA/OpenCL

Rendering strategies

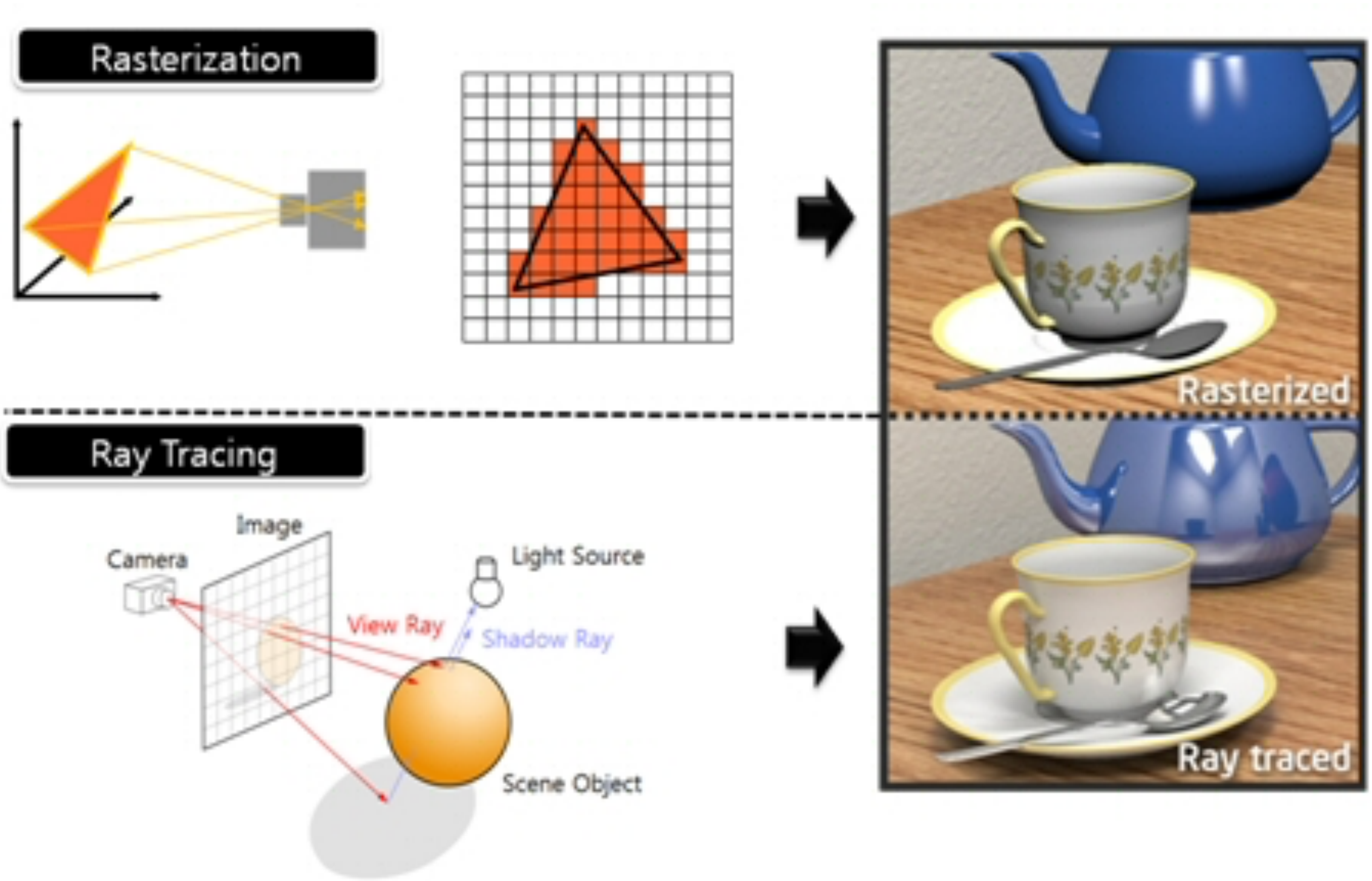
Rasterization

```
foreach triangle T{
    identify pixels p(x,y)
    covered by T;
    color(x,y) = shaded value
}
```

Ray tracing

```
foreach pixel p(x,y){
    intersect ray through pixel p
    with objects;
    color(x,y) = compute_shade(
    visible point)
}
```

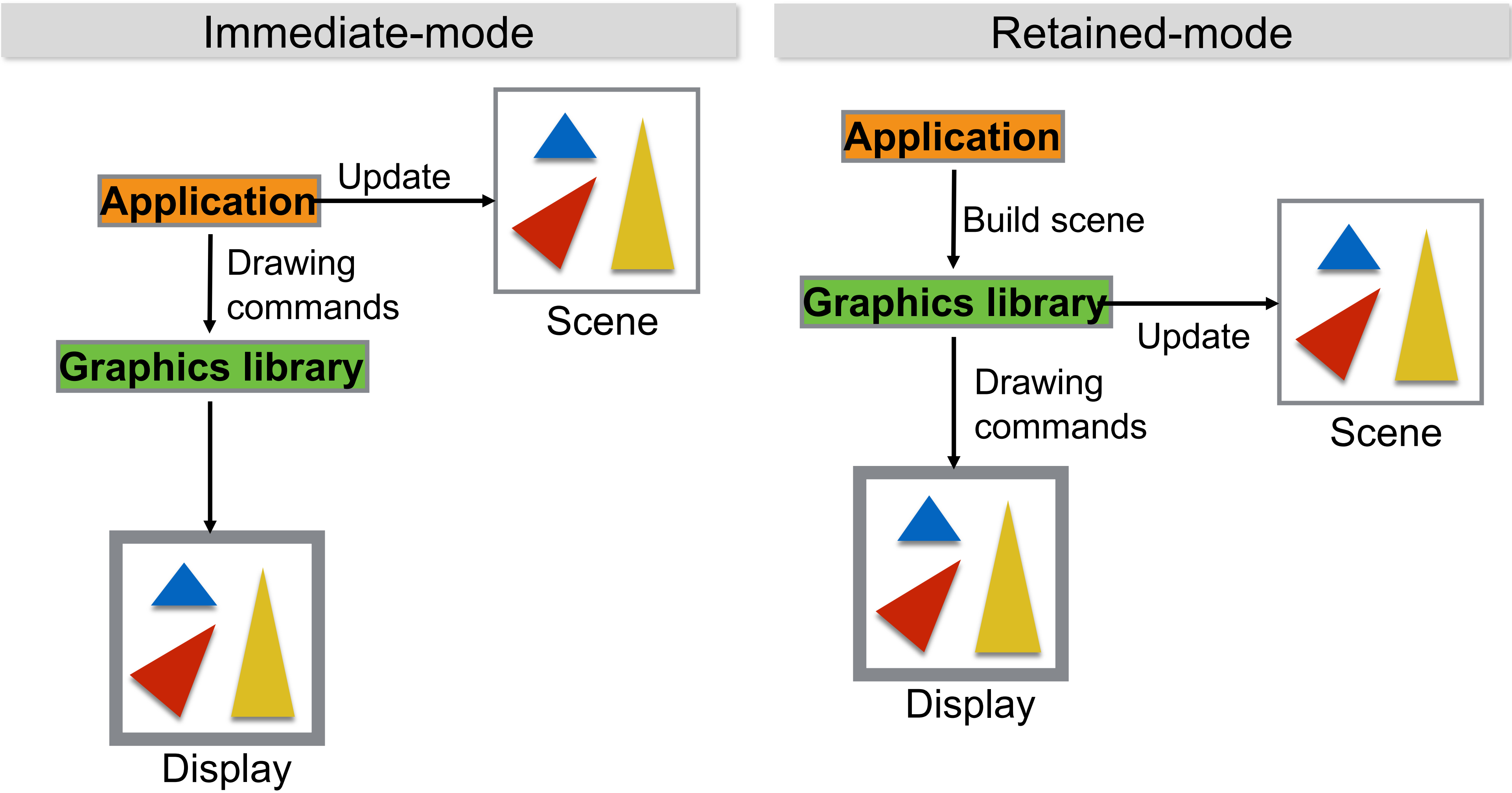
Rasterization vs Ray Tracing



Immediate-mode vs Retained-mode

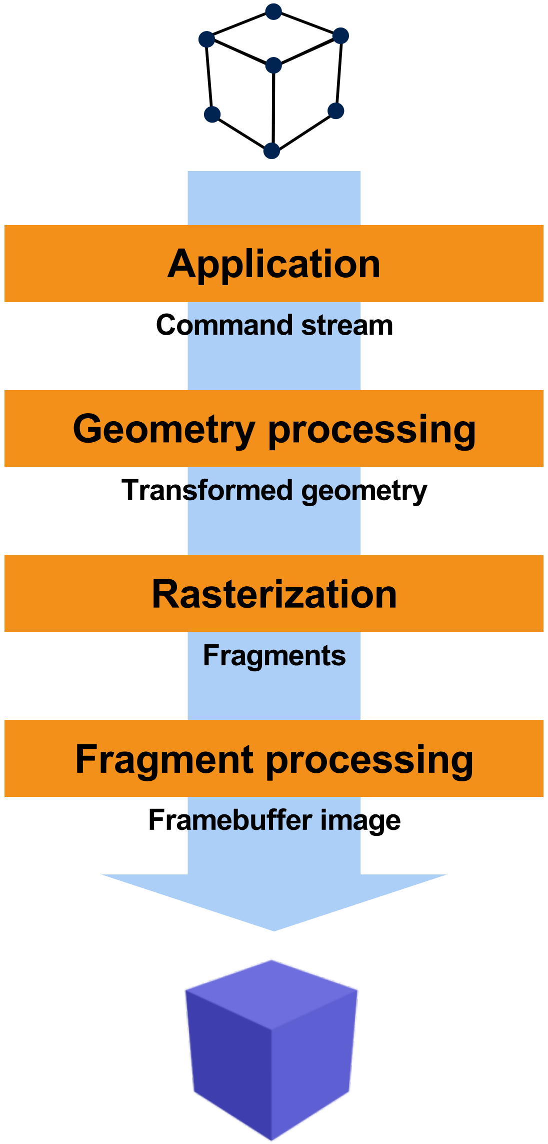
- Immediate-mode (IM)
  - The framework doesn’t store anything
  - The application directly draws to the container element
  - The management of the drawn objects is the task of the application
  - Offers control and flexibility
- Retained-mode (RM)
  - The scene (lines, shapes, images,…) is kept in memory
  - The actual drawing is performed once, using the stored model
  - The application simply adds and removes drawing primitives
  - Offers abstraction

Immediate-mode vs Retained-mode



Real-time rendering pipeline

1. Application
   - runs on general-purpose CPUs.
2. Geometry processing
   - operations such as transformations, projections, etc.
   - computes what is to be drawn, how it should be drawn, and where it should be drawn
   - typically performed on a graphics processing unit (GPU)
3. Rasterization
   - rasterize all the primitives
   - processed completely on the GPU
4. Fragment processing
   - draws (renders) an image
   - processed completely on the GPU

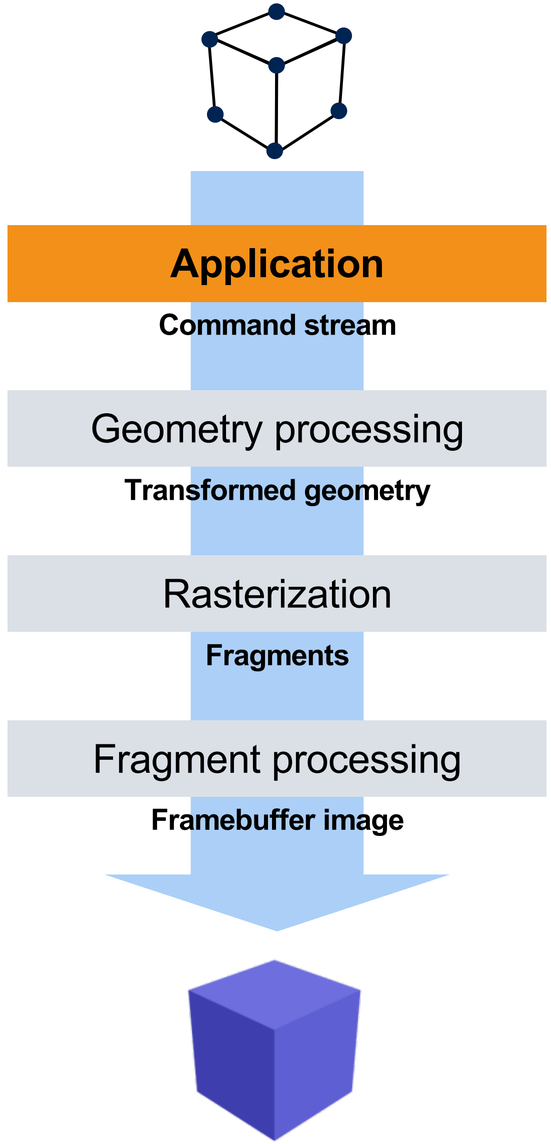


Rendering speed

- Update rate of images
- Expressed in:
  - frames per second (fps) - the number of images rendered per second
  - Hertz(Hz) - the frequency of update
- Depends on the complexity of the computations
- Determined by the slowest of the pipeline stages

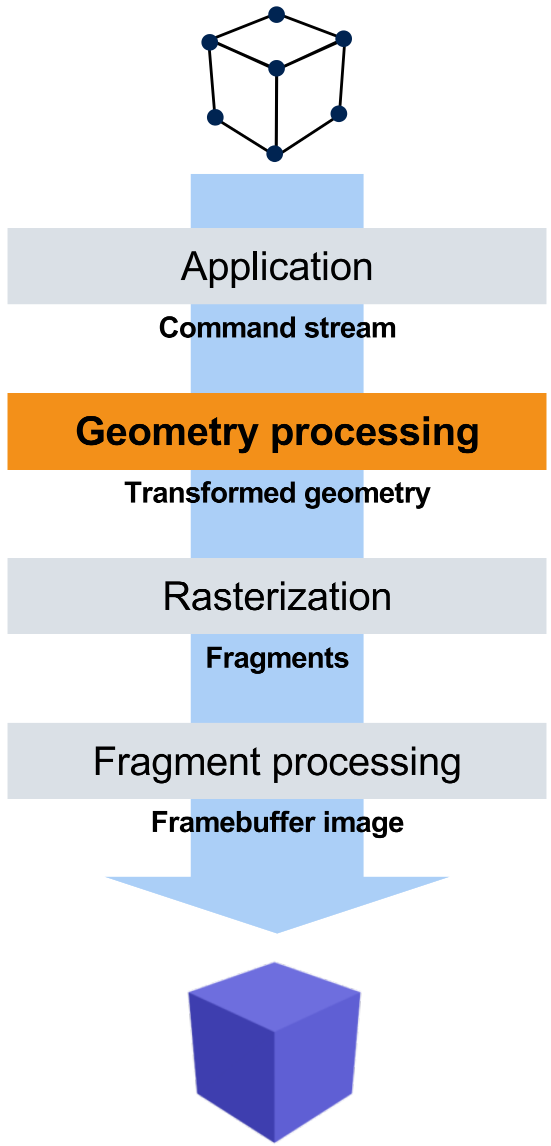
Application stage

- Send rendering primitives (points, lines, triangles) to the geometry stage
- Deals with collision detection, AI, etc.
- Takes care of input
  - keyboard
  - mouse
  - head-mounted helmet, etc..



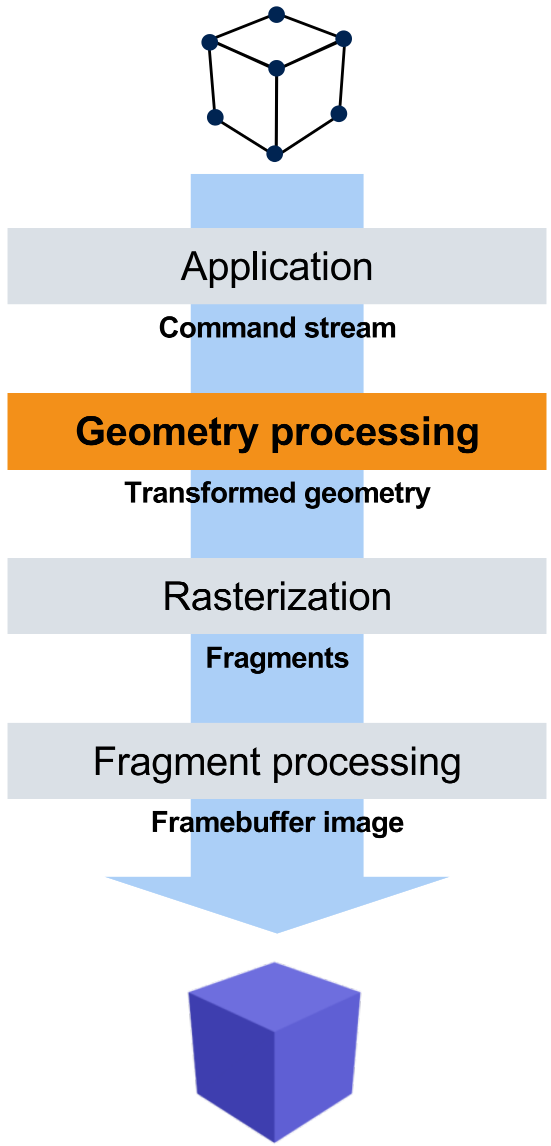
Geometry stage

- Per-polygon and Per-vertex operations
- Functional stages
  - model and view transform
  - vertex shading
  - projection
  - clipping
  - screen mapping



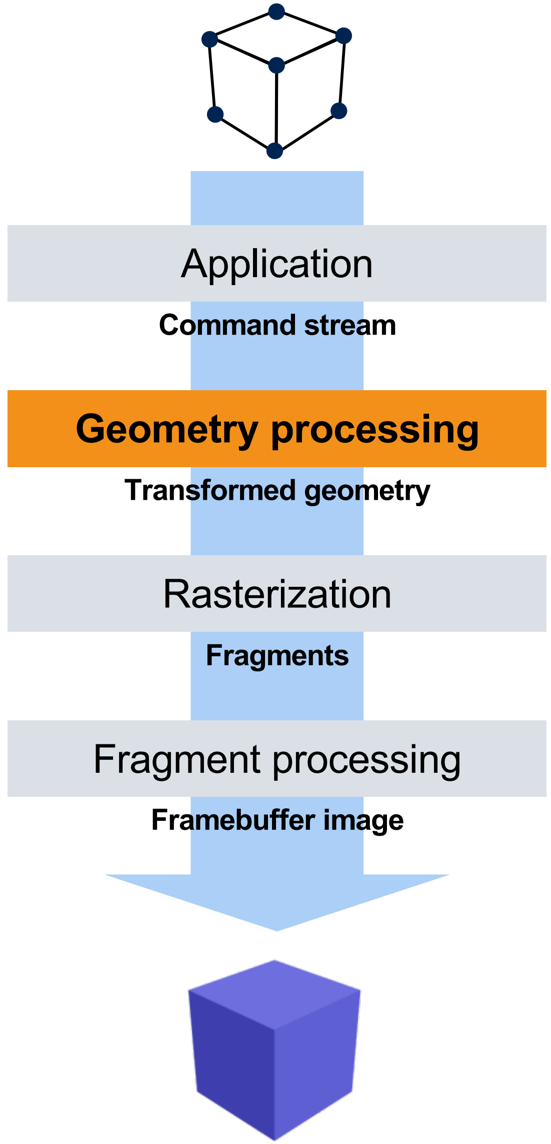
Model and view transform

- Each 3D model is transformed to different coordinate systems
  - Local (model) coordinates / model space
  - World coordinates / world space
- Model transformation - vertices and normal vectors
- World space - unique, all models exist in the same space

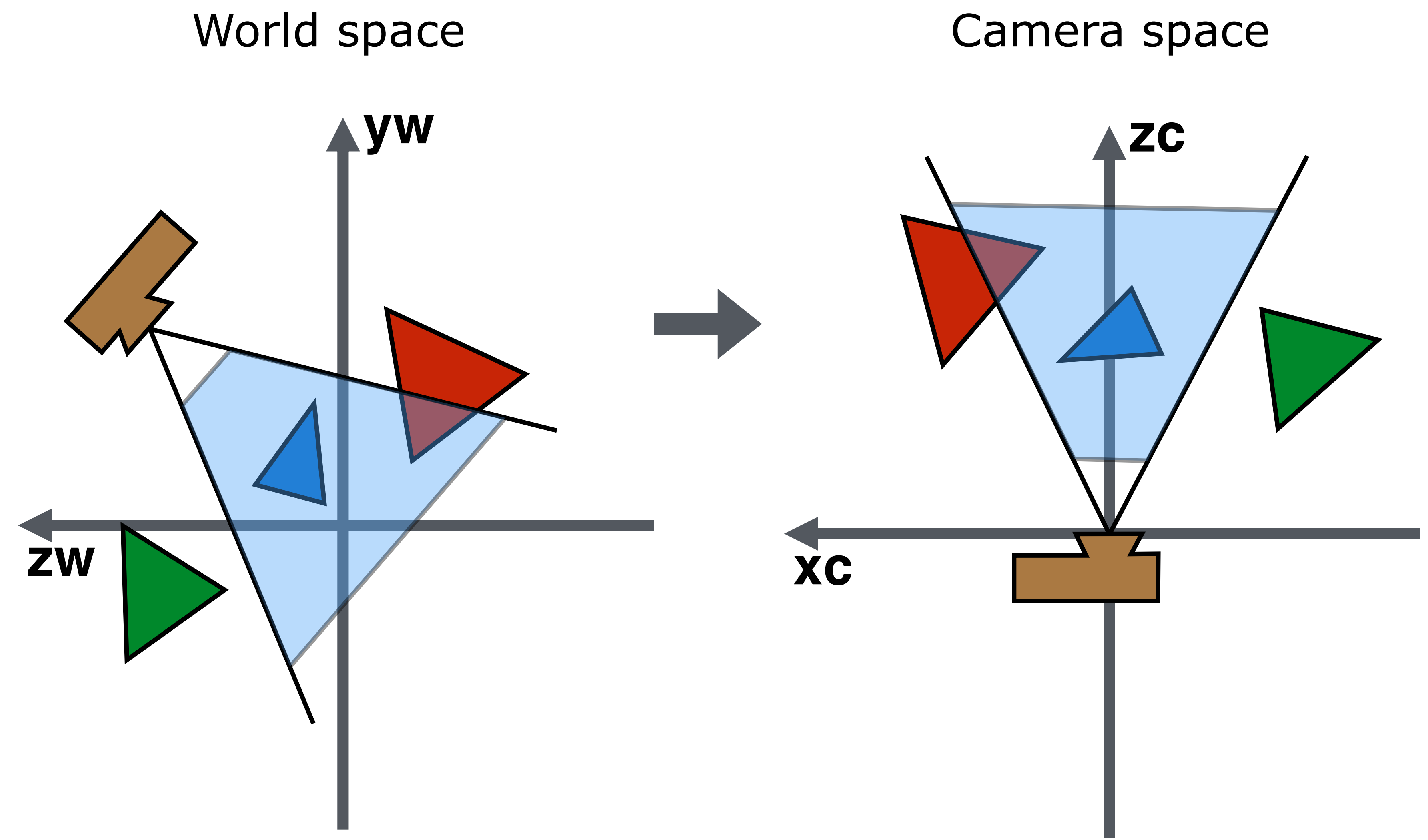


View transformation

- The position of the visualization camera is specified in world coordinates
- Has a direction, which aims the camera
- To make the projection and clipping easier, the camera and all the 3D models are transformed with the view transformation
- View transformation - place the camera at the origin and make it look toward the direction of the negative z-axis

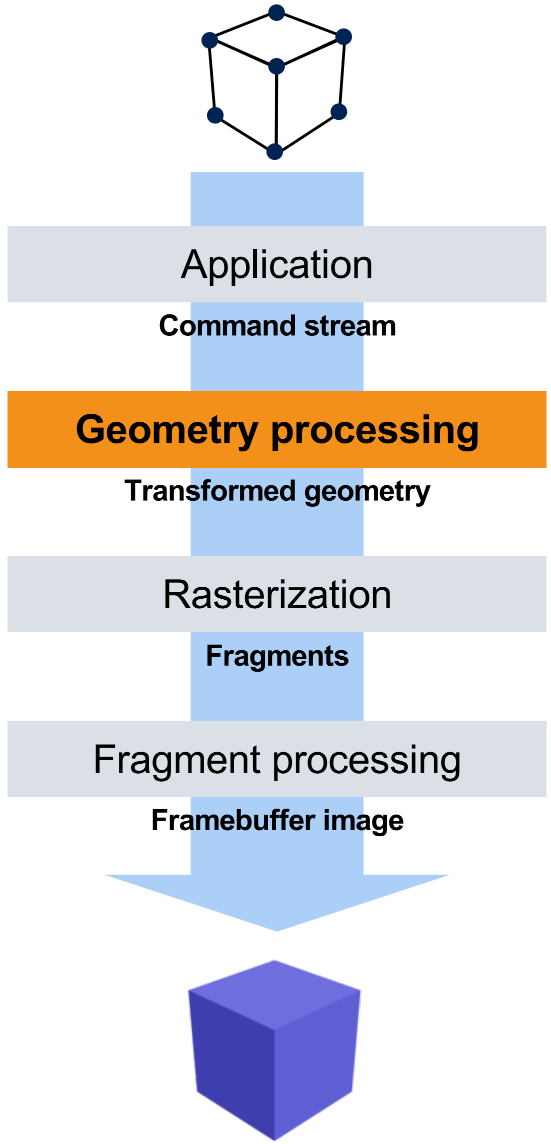


View transformation



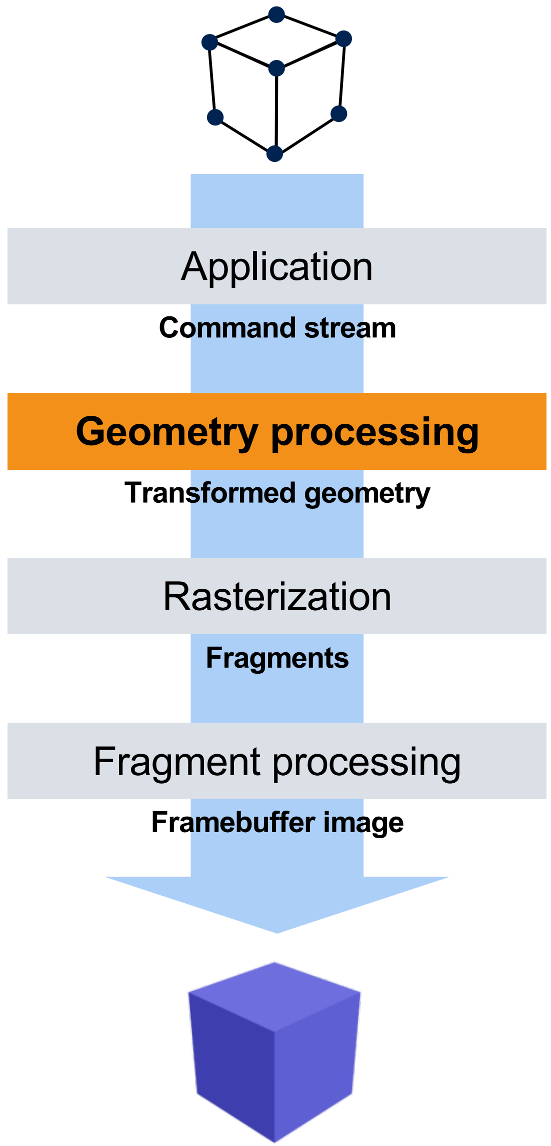
Vertex shading

- To produce a realistic image define materials and lights
- Shading - the operation of determining the effect of a light on a material
- Vertex shading results (colors, vectors, texture coordinates, etc.) are sent to the rasterization stage (to be interpolated)
- Shading computations can happen in world space or camera space



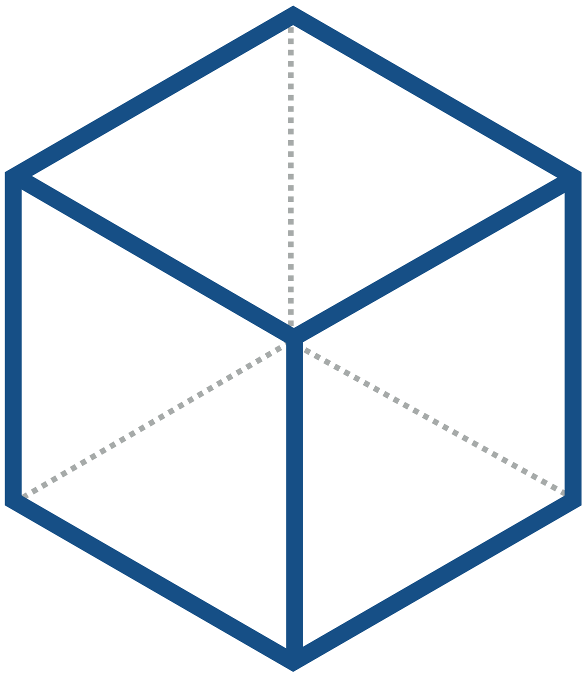
Projections

- Transforms the view volume into a unit cube - canonical view volume
- Can be defined between $(-1, -1, -1)$ and $(1,1,1)$ or $(0, 0, 0)$ and $(1,1,1)$
- Most commonly used projections
  - orthographic (parallel) projection
  - perspective projection
- After projection the 3D models are in normalized device coordinates.

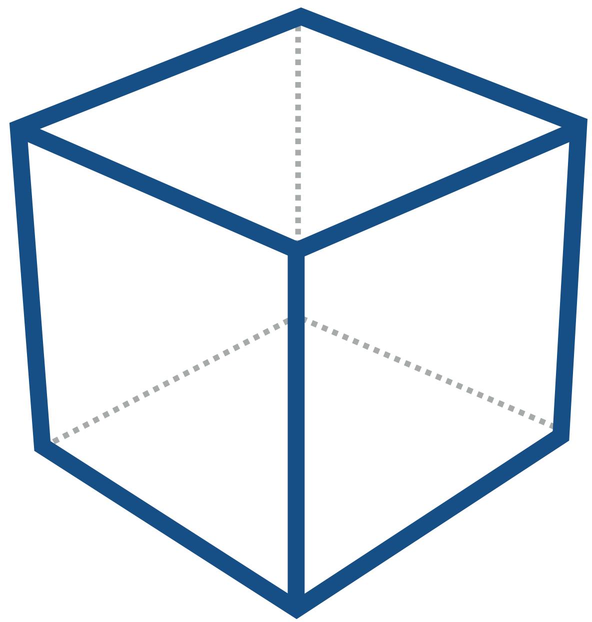


Projections

Orthographic projection



Perspective projection



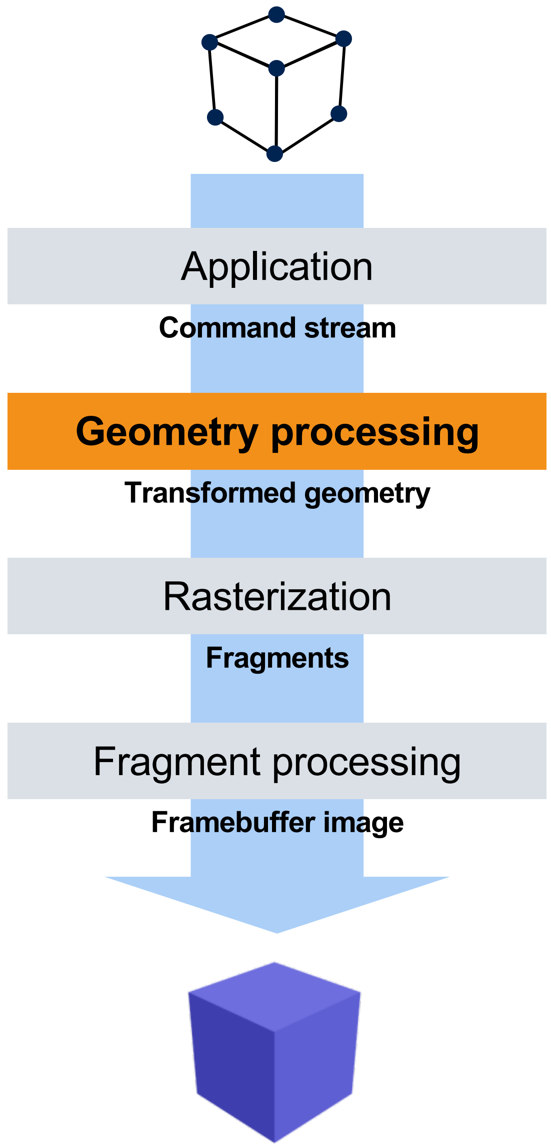
Clipping

- Pass to the rasterization step only the primitives which are wholly or partially inside the view volume
- Primitives that are completely inside

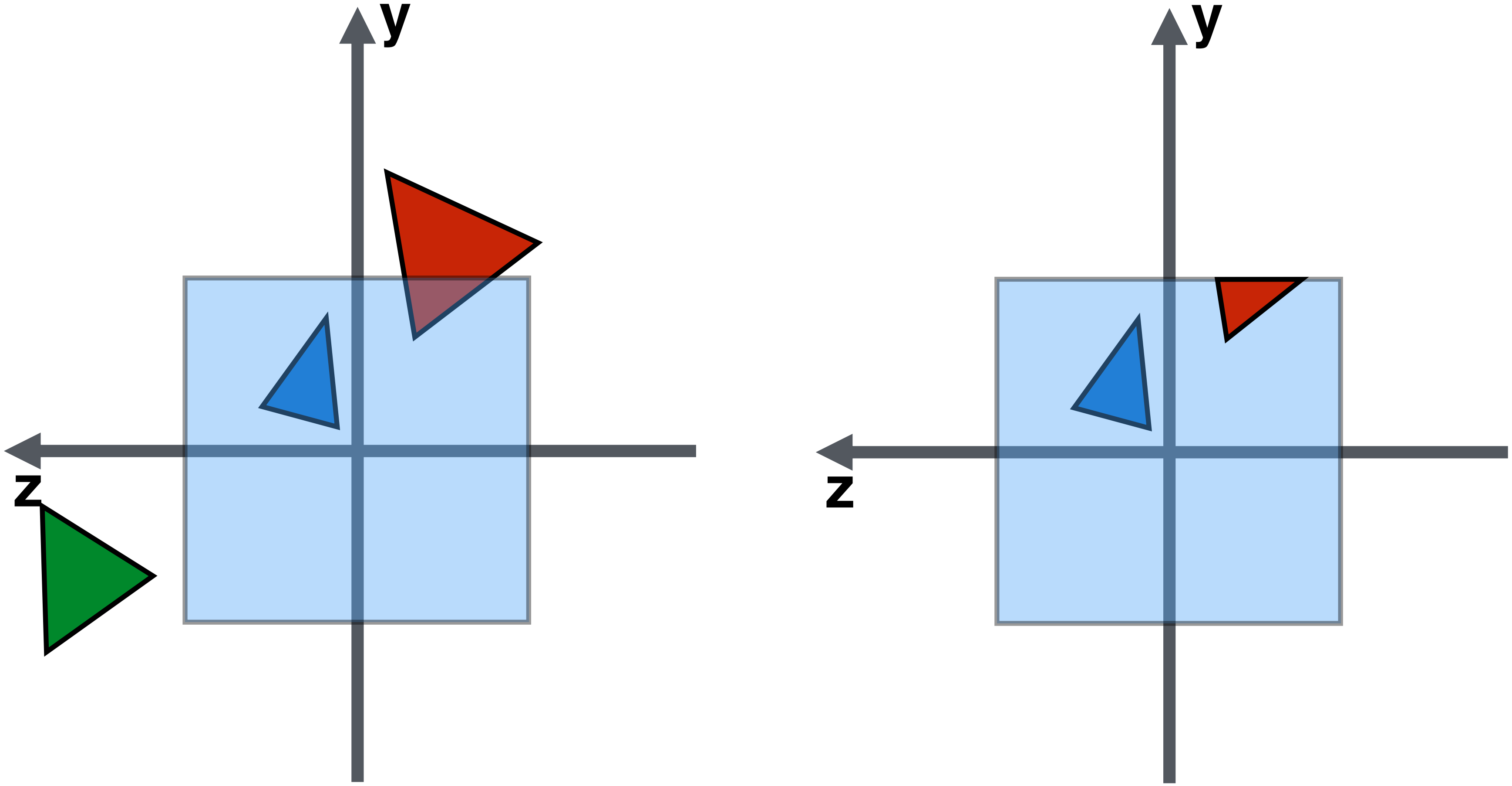
  - pass to the next stage as they are
- Primitives that are entirely outside

  - do not pass them further
- Primitives that are partially inside

  - clip them and pass to the next stage

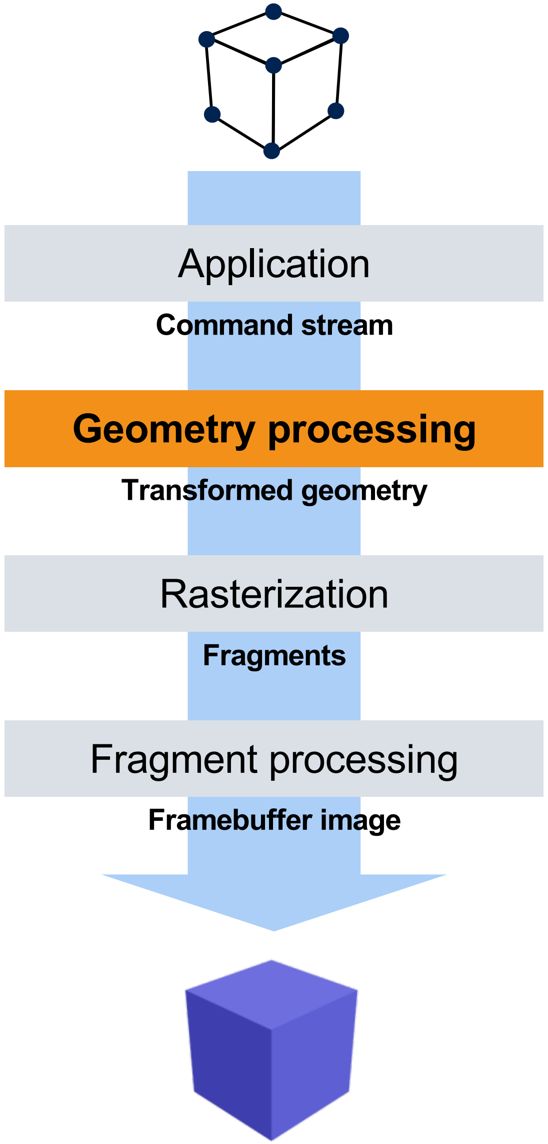


Clipping

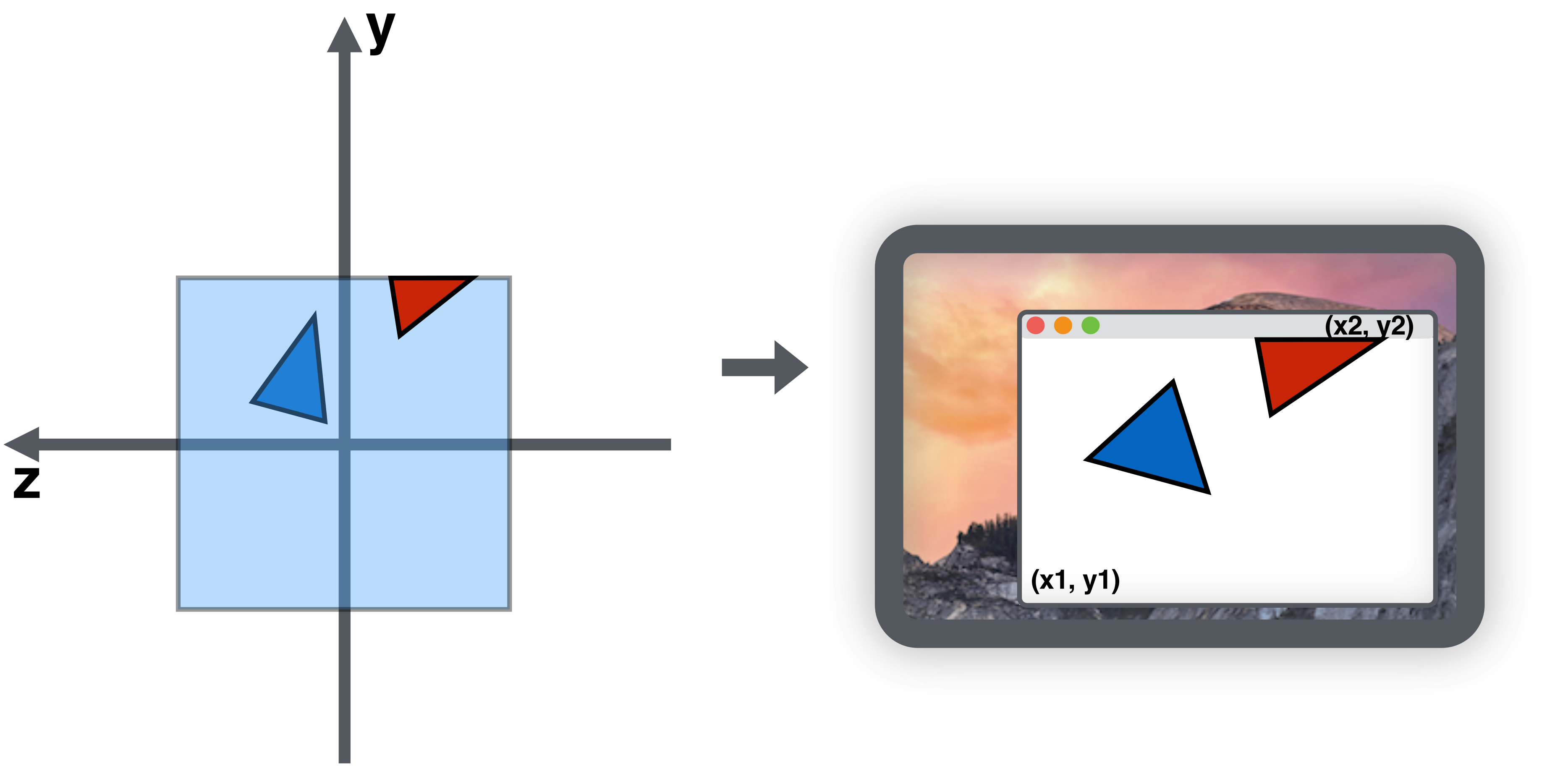


Screen mapping

- The x- and y-coordinates of each primitive are transformed into screen coordinates
- Screen coordinates together with the z-coordinates are also called window coordinates
- The z-coordinate is not affected by this mapping

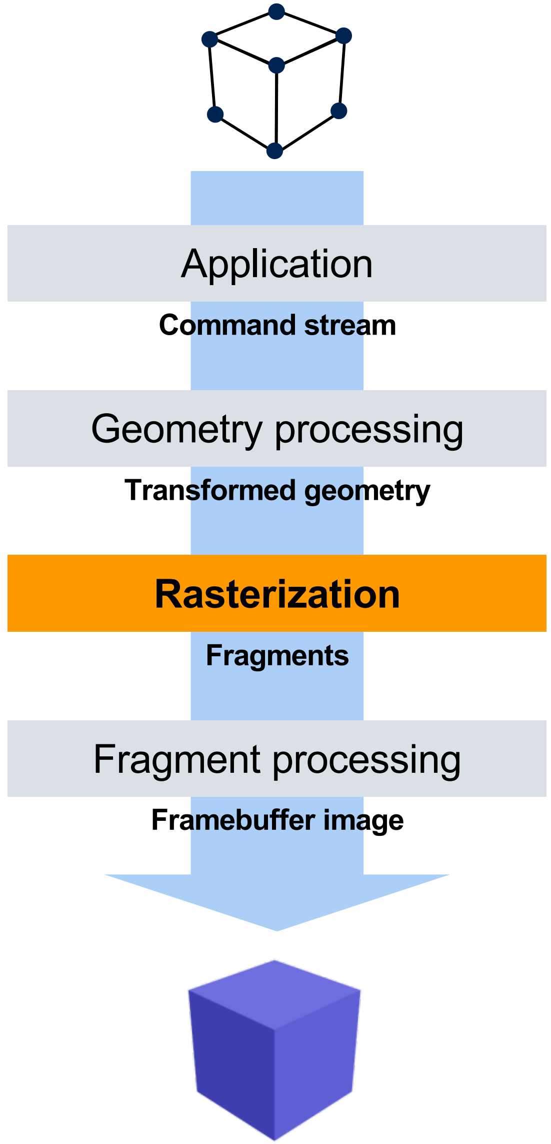


Screen mapping

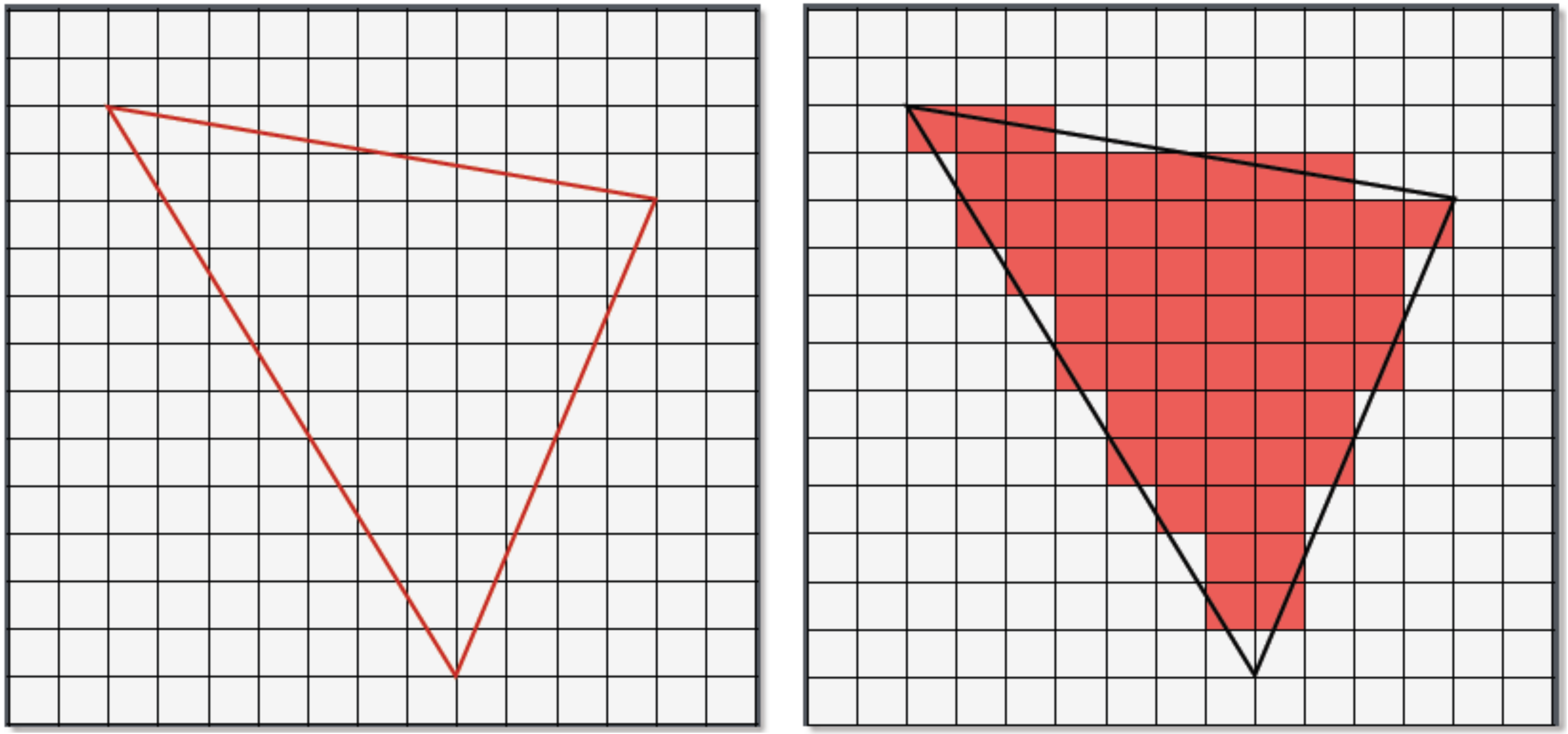


Rasterization stage

- Given a primitive (described by the vertex coordinates, color and texture information) convert it into a set of fragments
- Fragment data
  - raster position
  - depth (z-value)
  - interpolated attributes (color, texture coordinates, etc.)
  - stencil
  - alpha

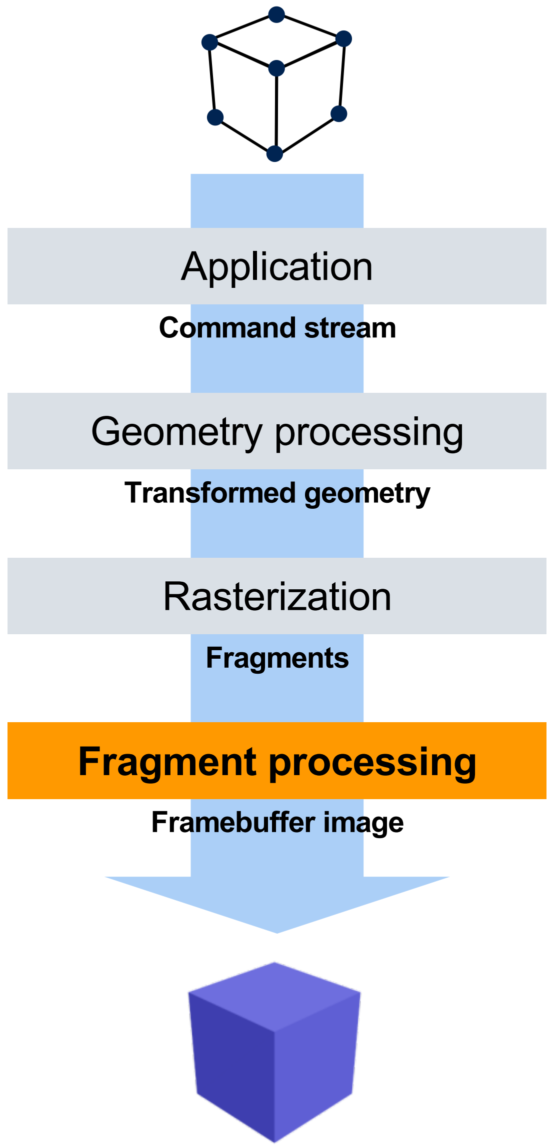


Example of rasterization

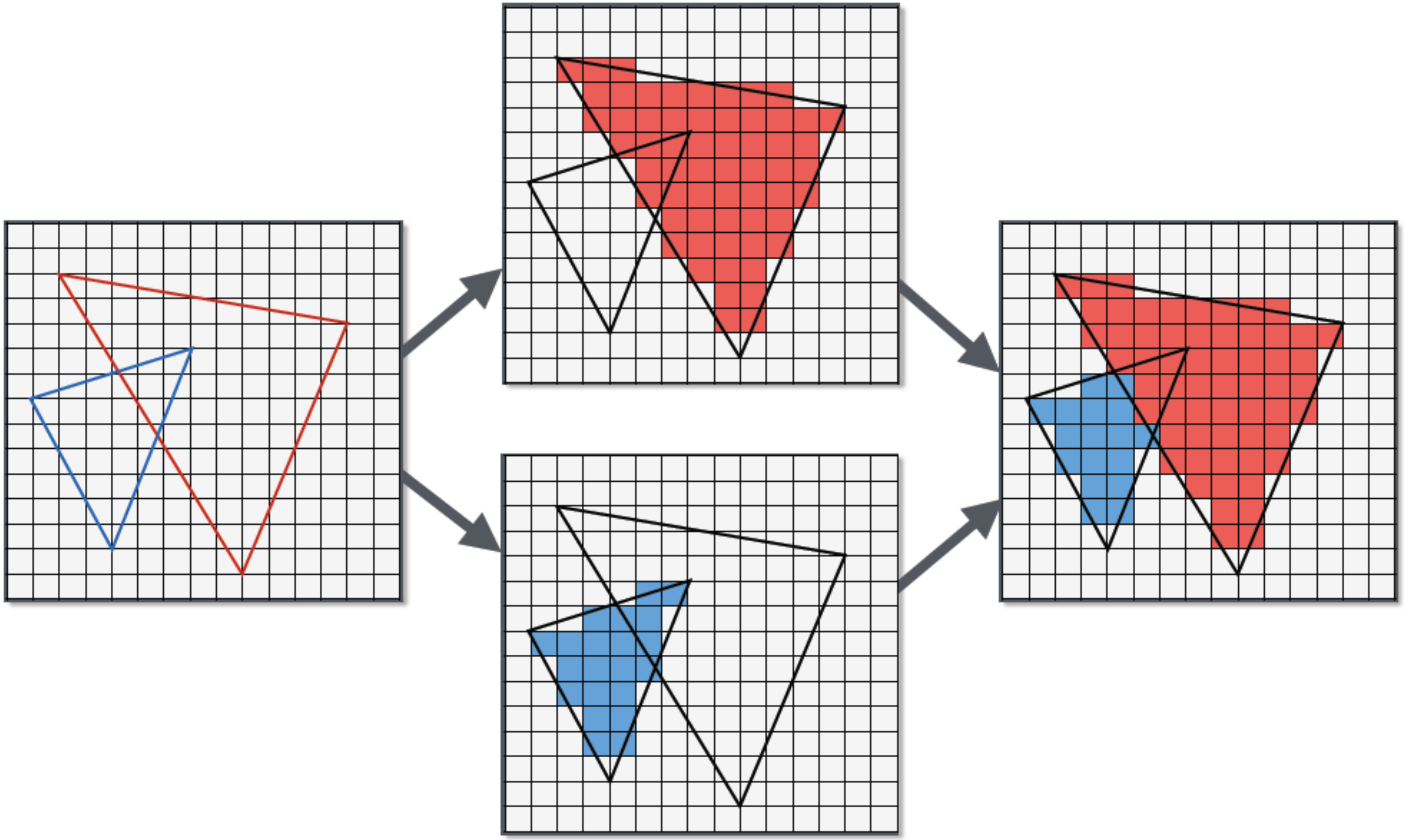


Fragment processing

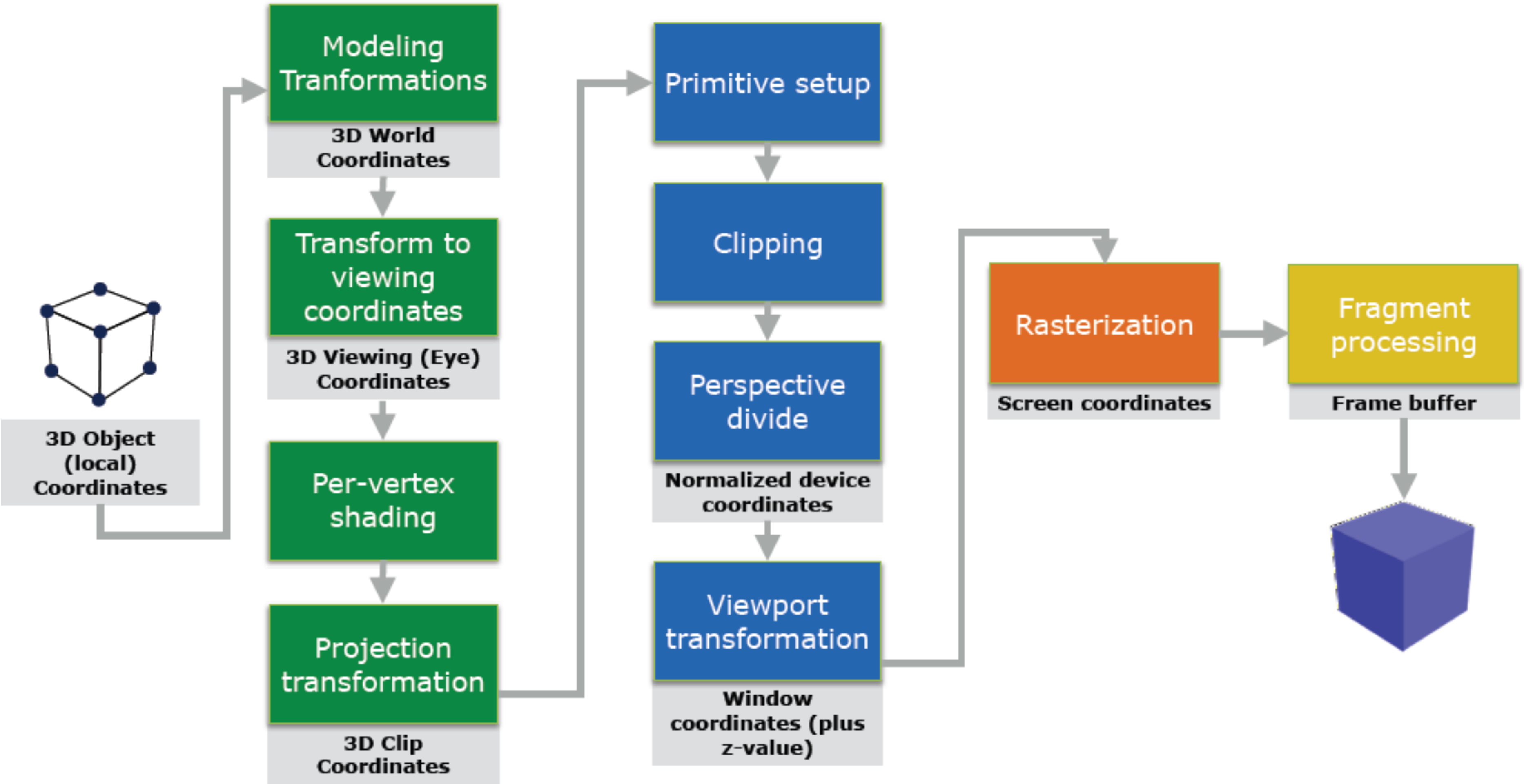
- Process each fragment from the rasterization process into a set of colors and a single depth value
- *Scissor test* - discard fragments outside of a certain rectangular portion of the screen
- *Stencil test* - test the fragment's stencil value against the value in the current stencil buffer; if the test fails, the fragment is culled
- *Depth test* - test the depth of the current fragment with the existing depth; if the test passes update the depth buffer
- *Blending* - combine the fragment's color with the existing color in the frame buffer



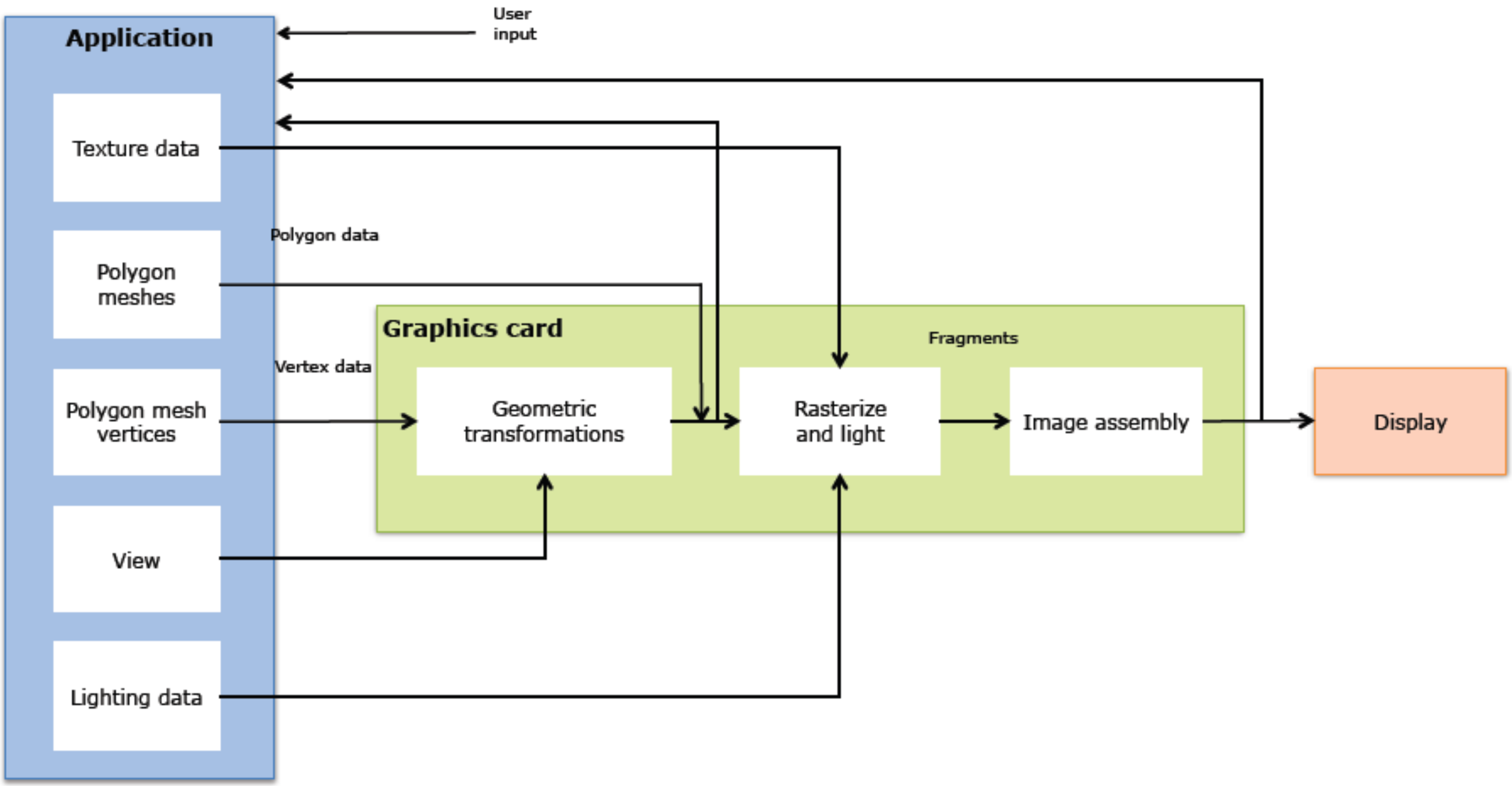
Fragment processing



Coordinates systems

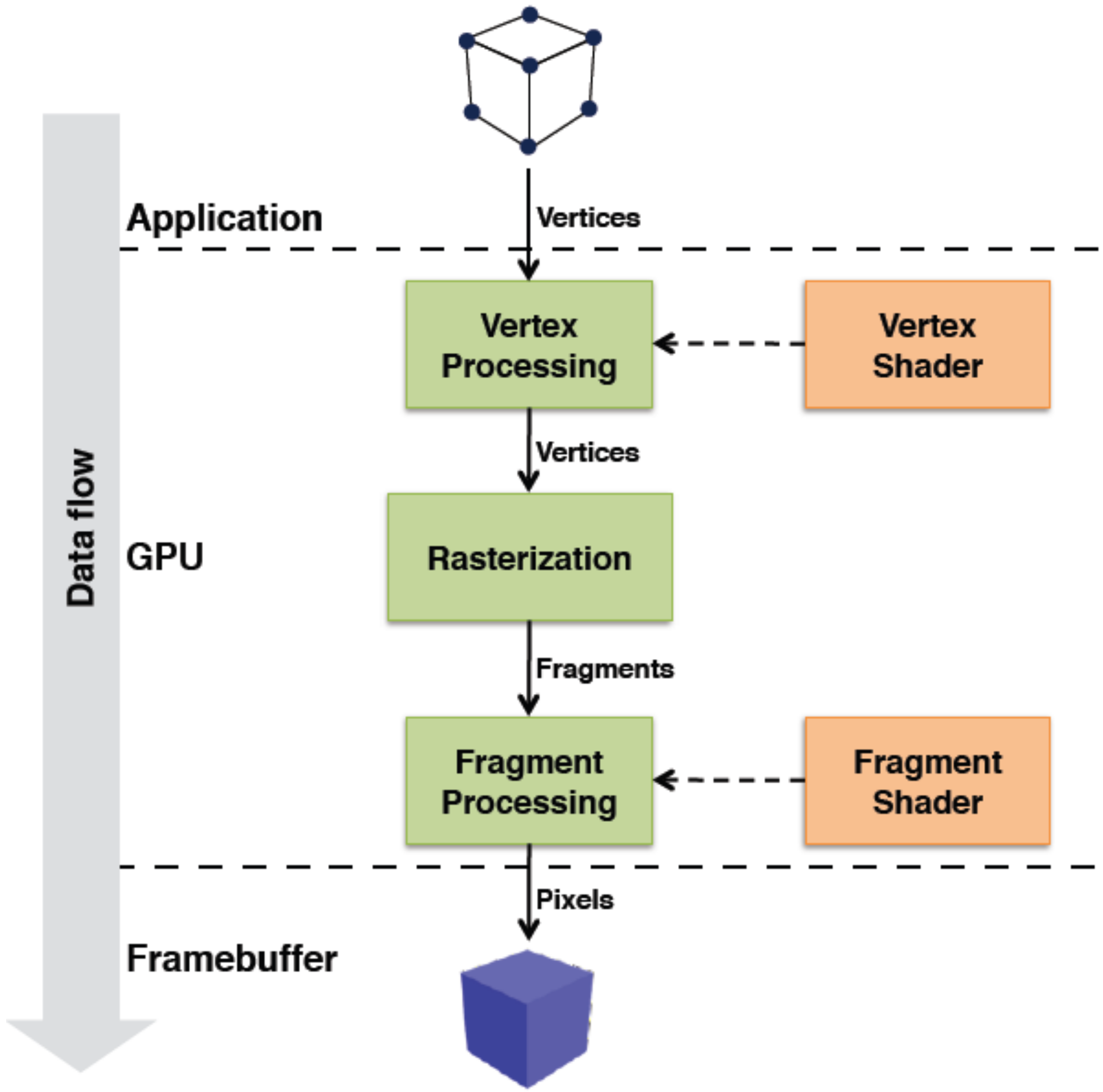


Graphics pipeline



[Source: Computer Graphics: Principles and Practice (3rd Edition), John F. Hughes et al.]

Basic graphics pipeline



Questions and proposed problems

1. Exemplify the transformation of a cube along the graphics rendering pipeline. Where and why the system may compute the shadows?
2. Where and why the system may remove the hidden parts of the objects, along the graphics pipeline?
3. What space and coordinate system are used to rasterize the graphics model?
4. Explain the motivation for the design and development of the graphics processing units (i.e. GPU).
5. Describe the trend in the evolution of GPUs. What are the main steps?
6. Explain the concept of the rendering strategy. Exemplify the rasterization and the ray tracing strategies.
7. Explain the concept of the rendering mode. Exemplify the immediate and the retained modes.
8. Describe the concept of real-time rendering pipeline. Exemplify for a cube the passing throughout the phases of Application, Geometry processing, Rasterization, and Fragment processing.

Questions and proposed problems

9. Explain what means “per-polygon and per-vertex operations” in the Geometry stage of the graphics pipeline.
10. Explain what means “per-polygon and per-vertex operations” in the Geometry stage of the graphics pipeline.
11. Explain why the initial transformation of the 3D model is performed within the Local coordinates system and then, in the World coordinate system.
12. Explain what means the view transformation.
13. What means shading operations? Why the shading is computed regarding the vertex?
14. Explain why the vertex shading computation can be achieved both in the world space and in the camera space.
15. Explain the rasterization for the following vector primitives given by vertices: point A(x,y,z), line (A(x,y,z), B(x,y,z)), polygon (V1(x,y,z), V2(x,y,z), …, Vn(x,y,z)).