Graphics pipeline

Graphics rendering pipeline

- Generates (render) a two-dimensional image
  - *Inputs*: 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

Real-Time 3D Graphics API

- Software abstraction of the capabilities of the graphics card
- Application programming interface (**API**)
  - provides hardware independence (each GPU has its own instruction set and interface)
- **Direct3D** - Windows (only)
- **OpenGL** - open source cross-platform

Real-Time 3D Graphics API

- *Functionality*:
  - describe a scene consisting of objects and their properties
  - describe light sources and their properties
  - describe cameras and their properties
  - render the objects (from a particular view and as if they were illuminated by the light sources)

Evolution of graphics pipelines

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

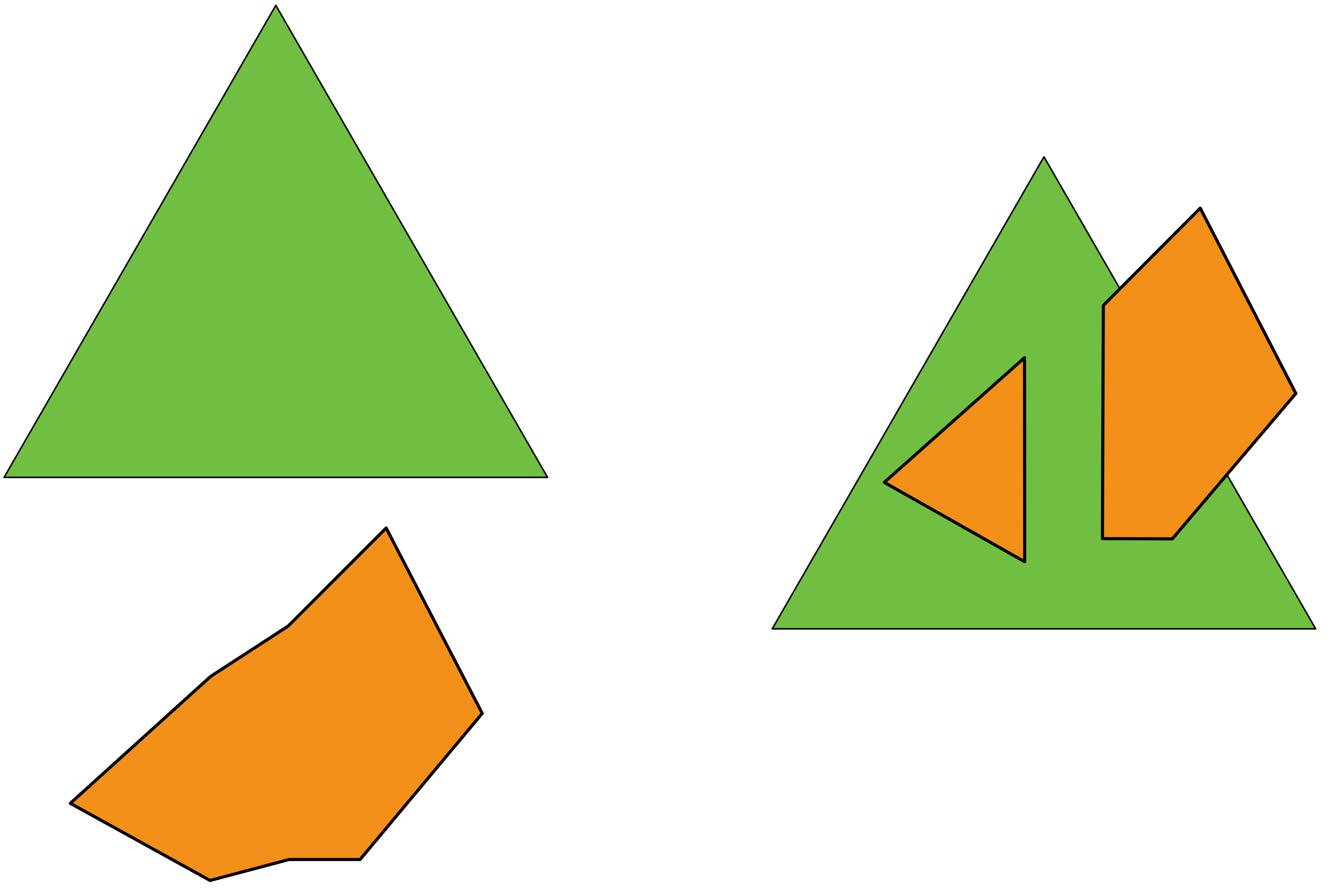
Rasterization

```
foreach triangle T{
 fragments = identify
 pixels   p(x,y)
 covered by T
 foreach fragment F{
  check if F is visible
  if(visible){
   color(x,y) = compute
   shaded value of F
  }
 }
}
```

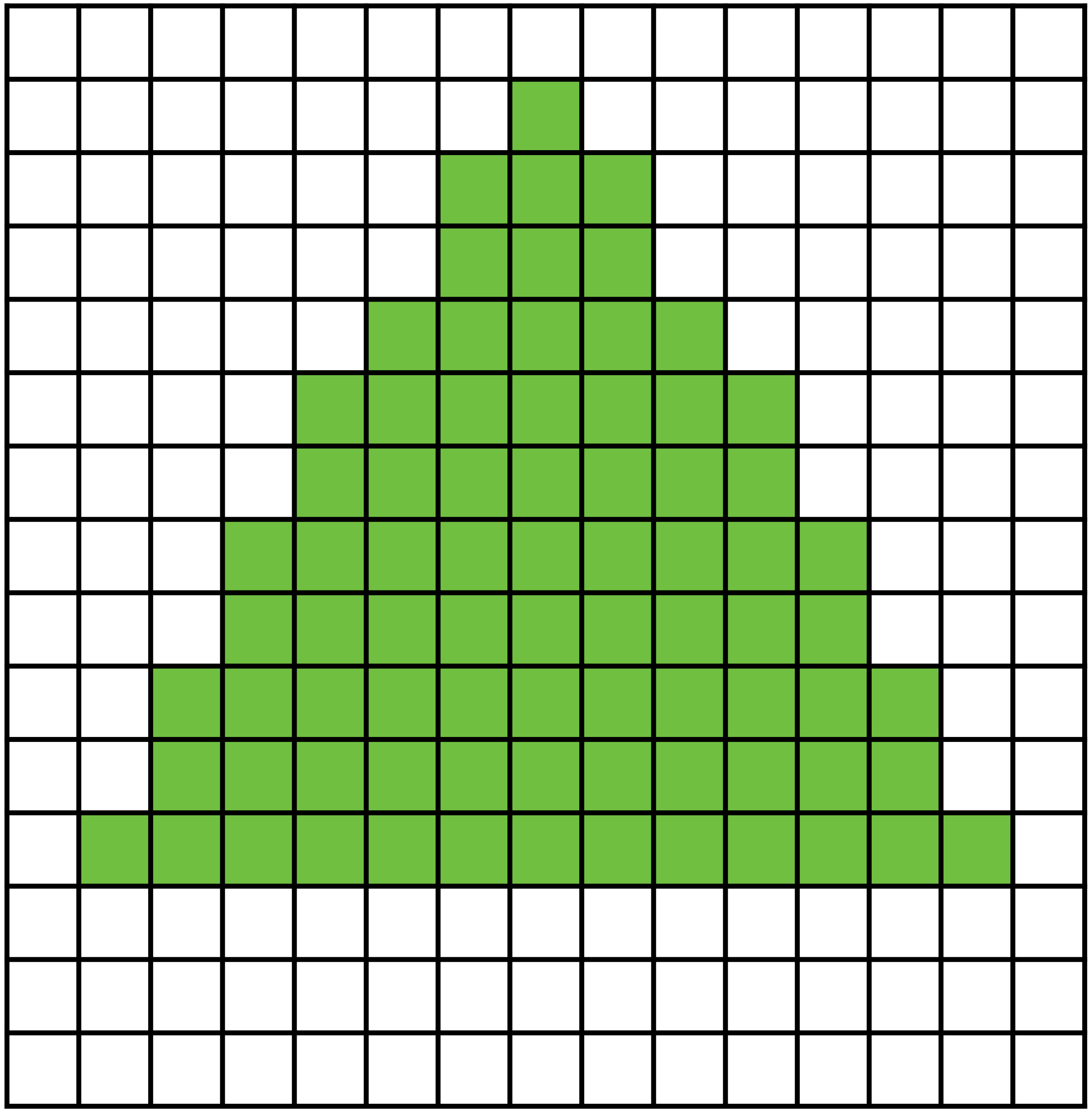
Ray tracing

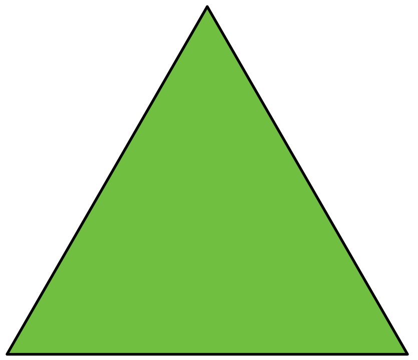
```
foreach pixel p(x,y){
 intersect ray through
 pixel p with objects
 color(x,y) =
 compute_shade(visible
 point)
}
```

Rasterisation

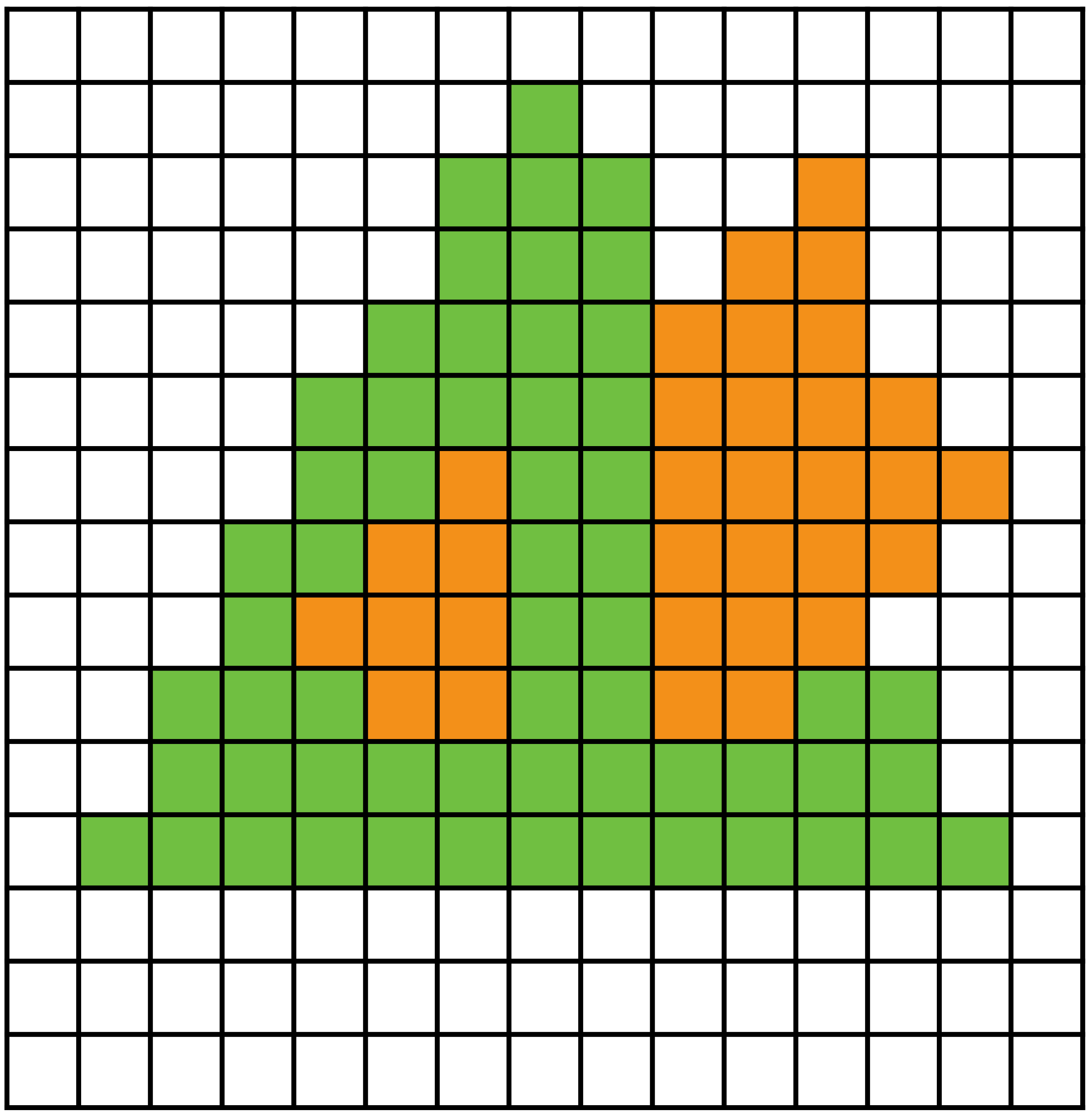


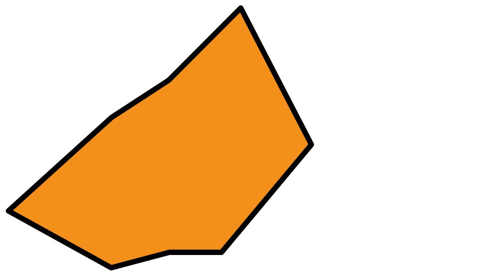
Rasterisation



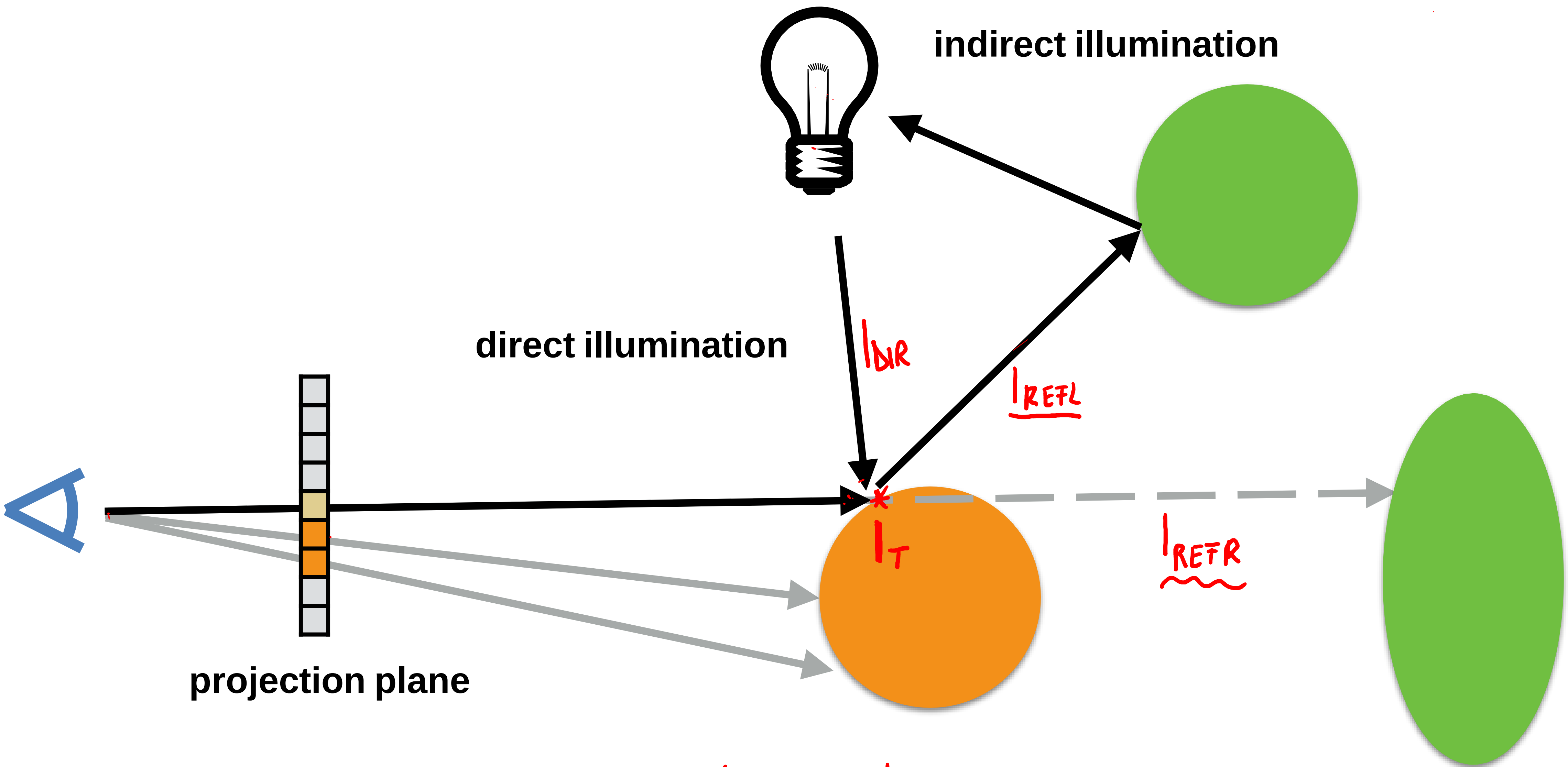


Rasterisation





Ray tracing



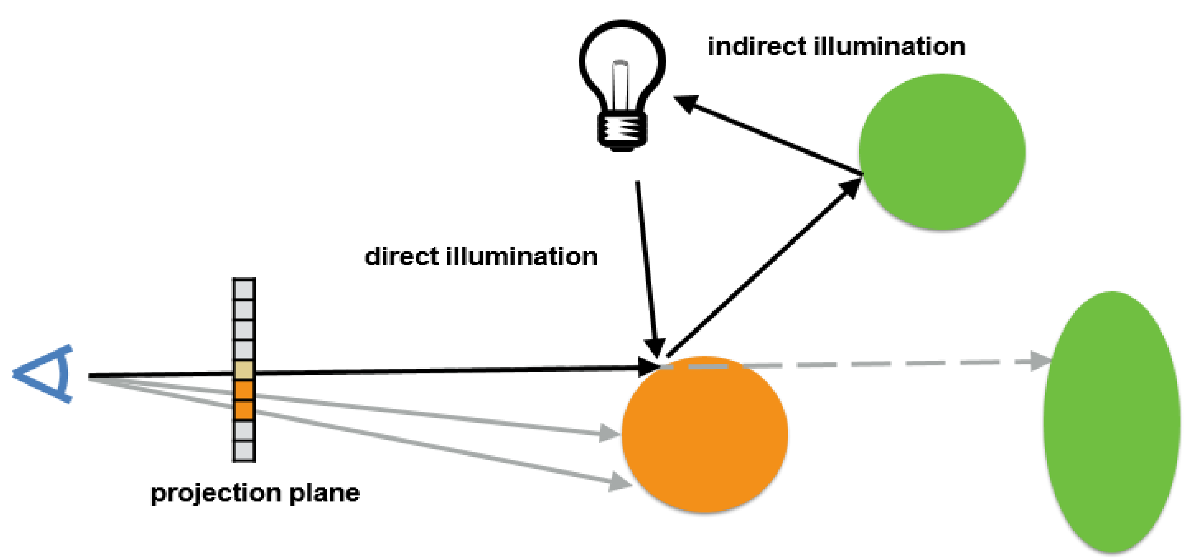
$I_T = I_{DIR} + \underline{I_{REFL}} + I_{REFR}$

Ray tracing

$$I_T = I_{DIR} + I_{REFL} + I_{REFR}$$

$$\boxed{I'_T = I'_{DIR} + I'_{REFL} + I'_{REFR}} \qquad \boxed{I''_T = I''_{DIR} + I''_{REFL} + I''_{REFR}}$$

... ...



Rasterization vs Ray Tracing

Raster -> Blender v2.79 (Render -> OpenGL)

Rasterization vs Ray Tracing

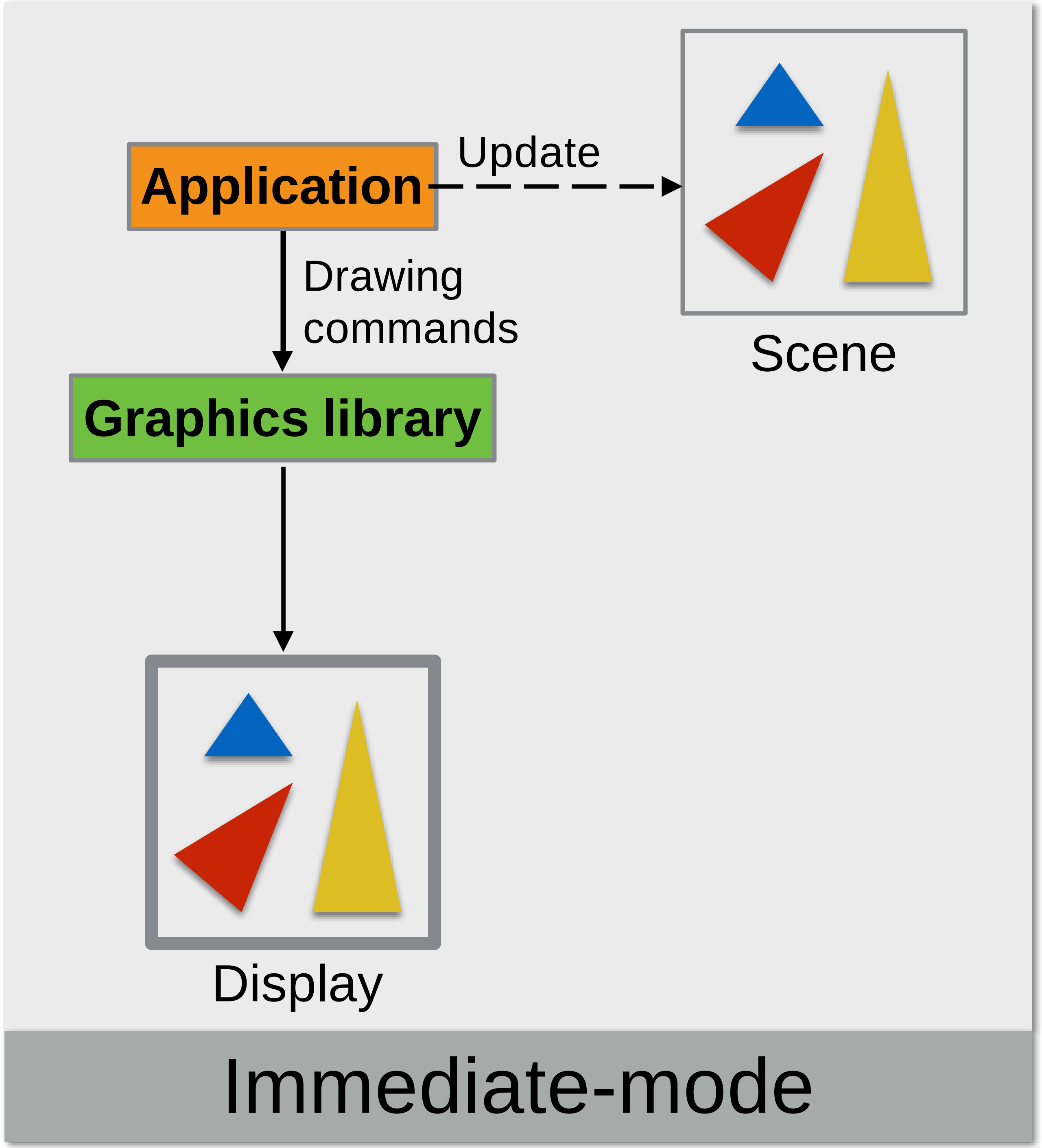
Ray Tracing → Blender 2.79 (Cycles Renderer)

Ray Tracing - Reflection/Refraction

Immediate-mode vs Retained-mode

Immediate-mode (IM)

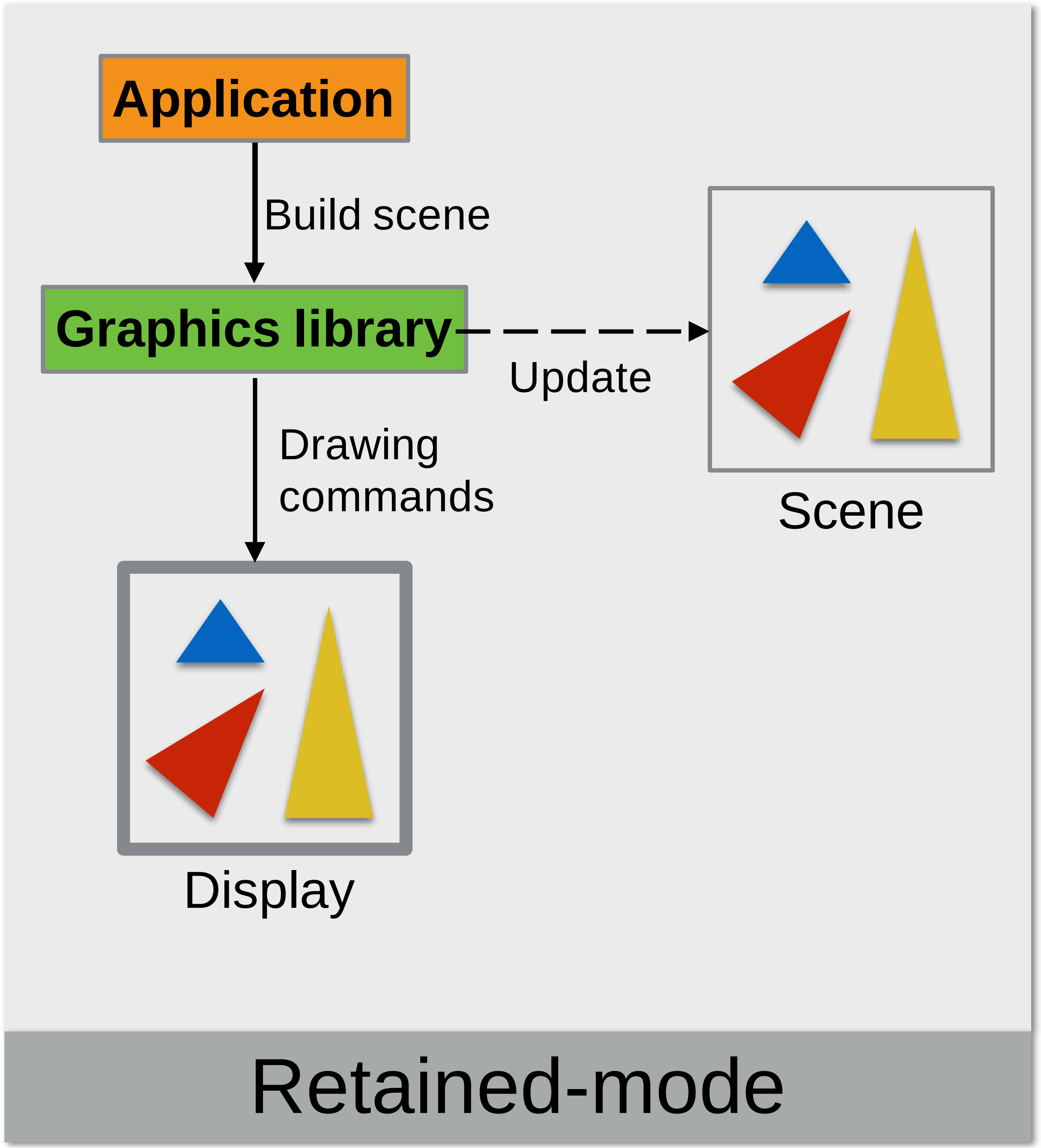
- The scene of objects is not saved at graphics library or GPU level
- The application resends all drawing commands for the new frame
- Offers control and flexibility



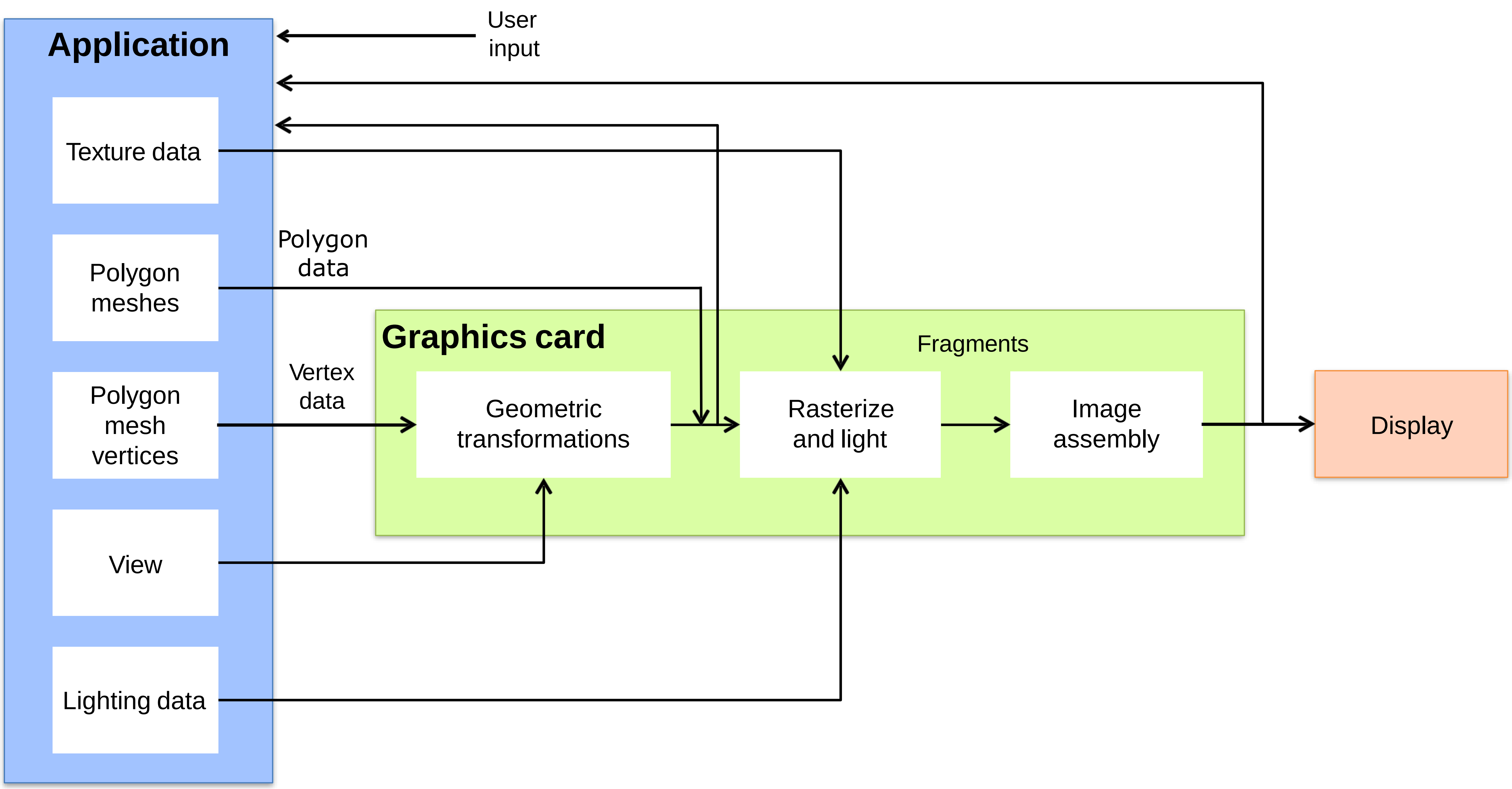
Immediate-mode vs Retained-mode

Retained-mode (RM)

- The application updates the scene of objects which is stored at graphics library or GPU level
- The actual rendering is not called by the application
- Offers abstraction



Graphics Pipeline



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

Real-time rendering pipeline

Application

- runs on general-purpose CPUs

Vertex processing

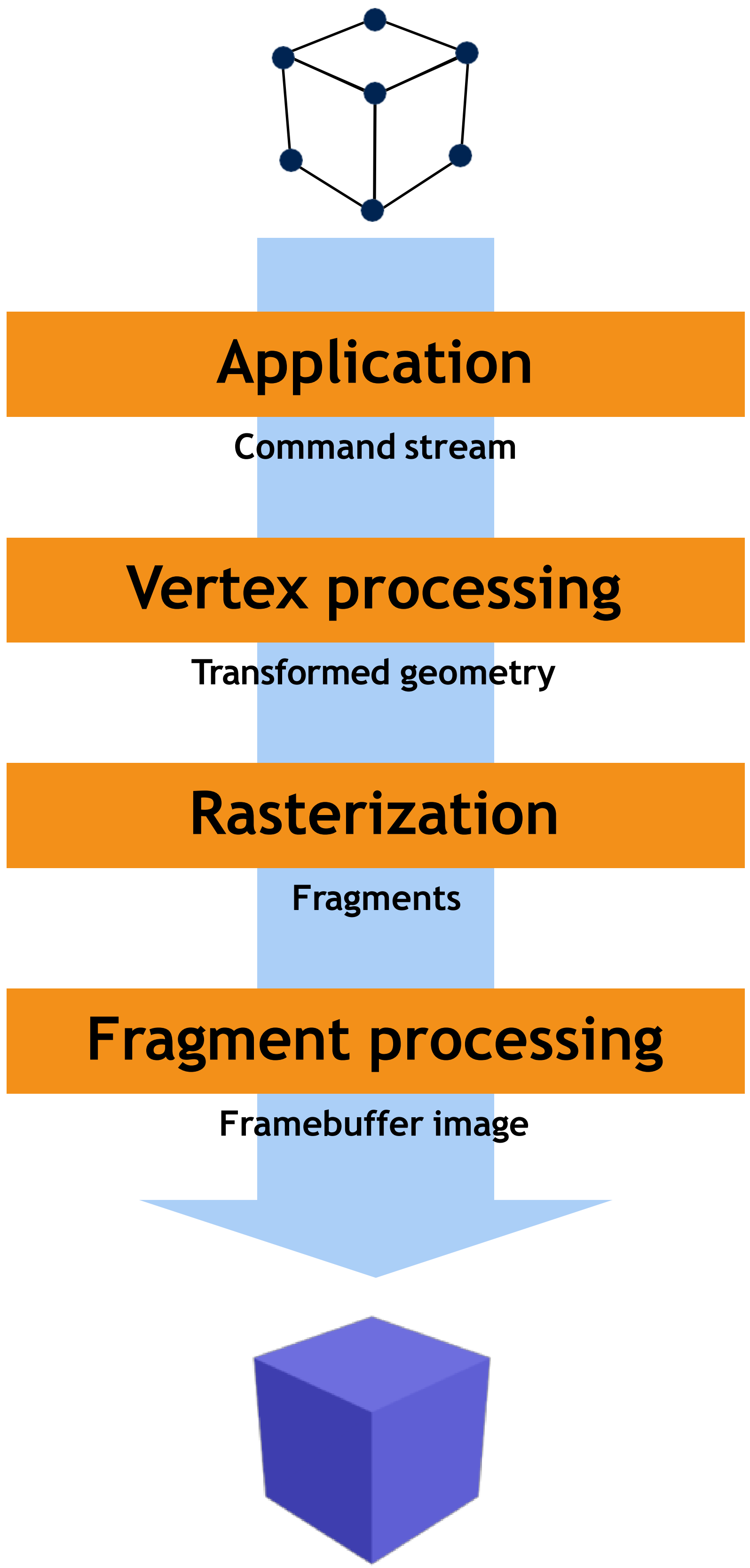
- operations such as transformations, projections, etc.
- computes what is to be drawn, how it should be drawn, and where it should be drawn
- typically performed on a graphics processing unit (GPU)

Rasterization

- rasterize all the primitives
- processed completely on the GPU

Fragment processing

- draws (renders) an image
- processed completely on the GPU



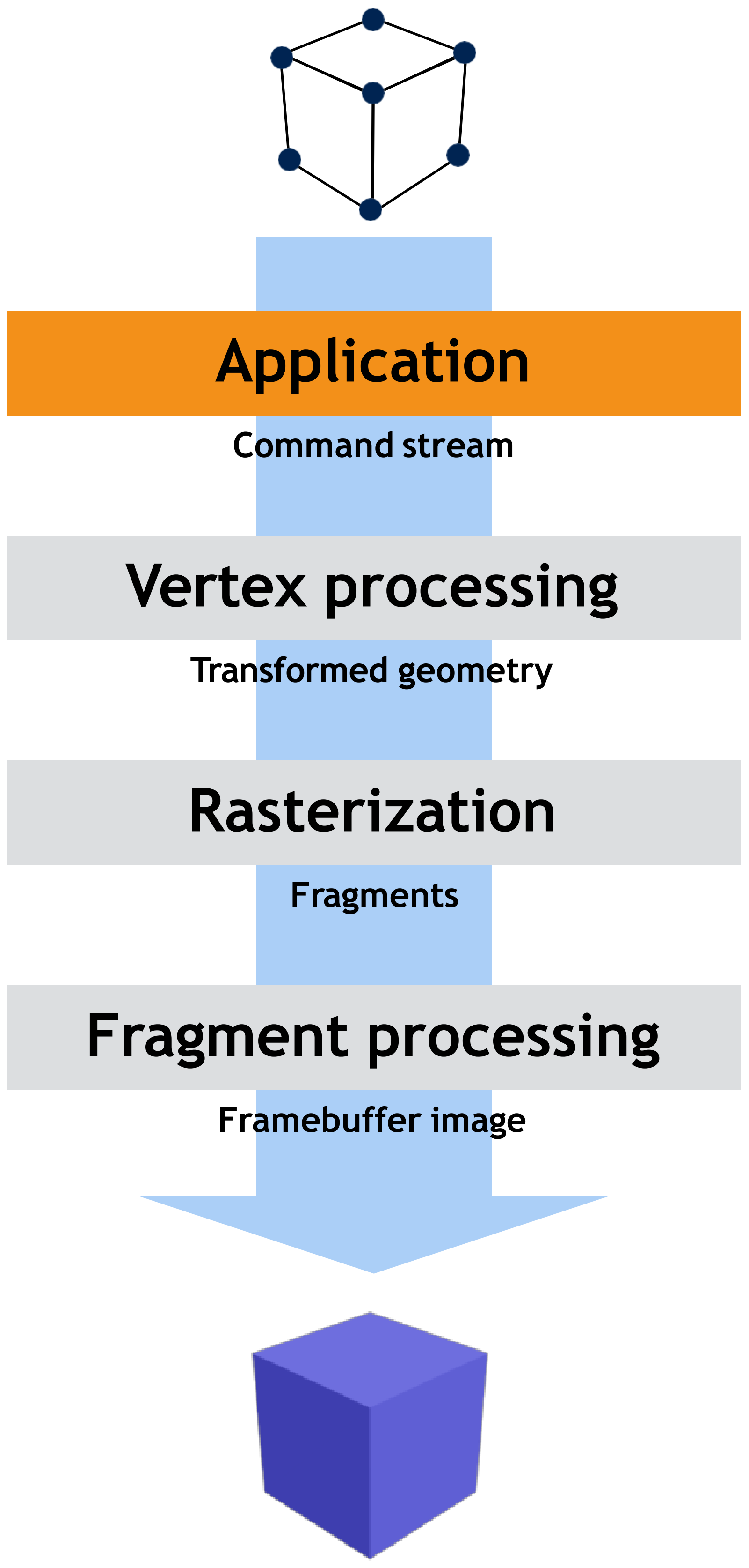
Rendering speed

- Update rate of images
- Expressed in:
  - **frames per second (fps)** - the number of images rendered per second

- Depends on the complexity of the computations
- Determined by the slowest of the pipeline stages

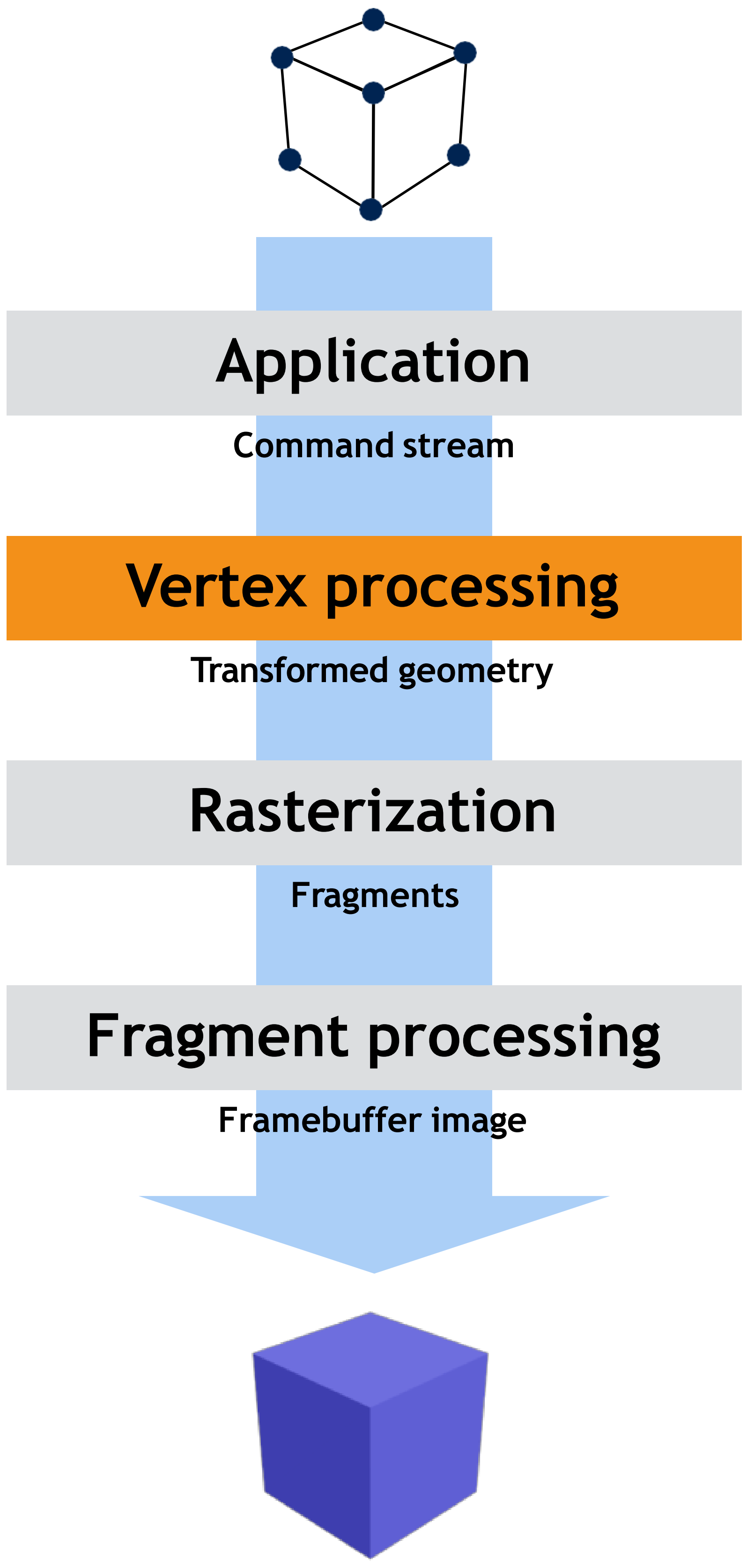
Application stage

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



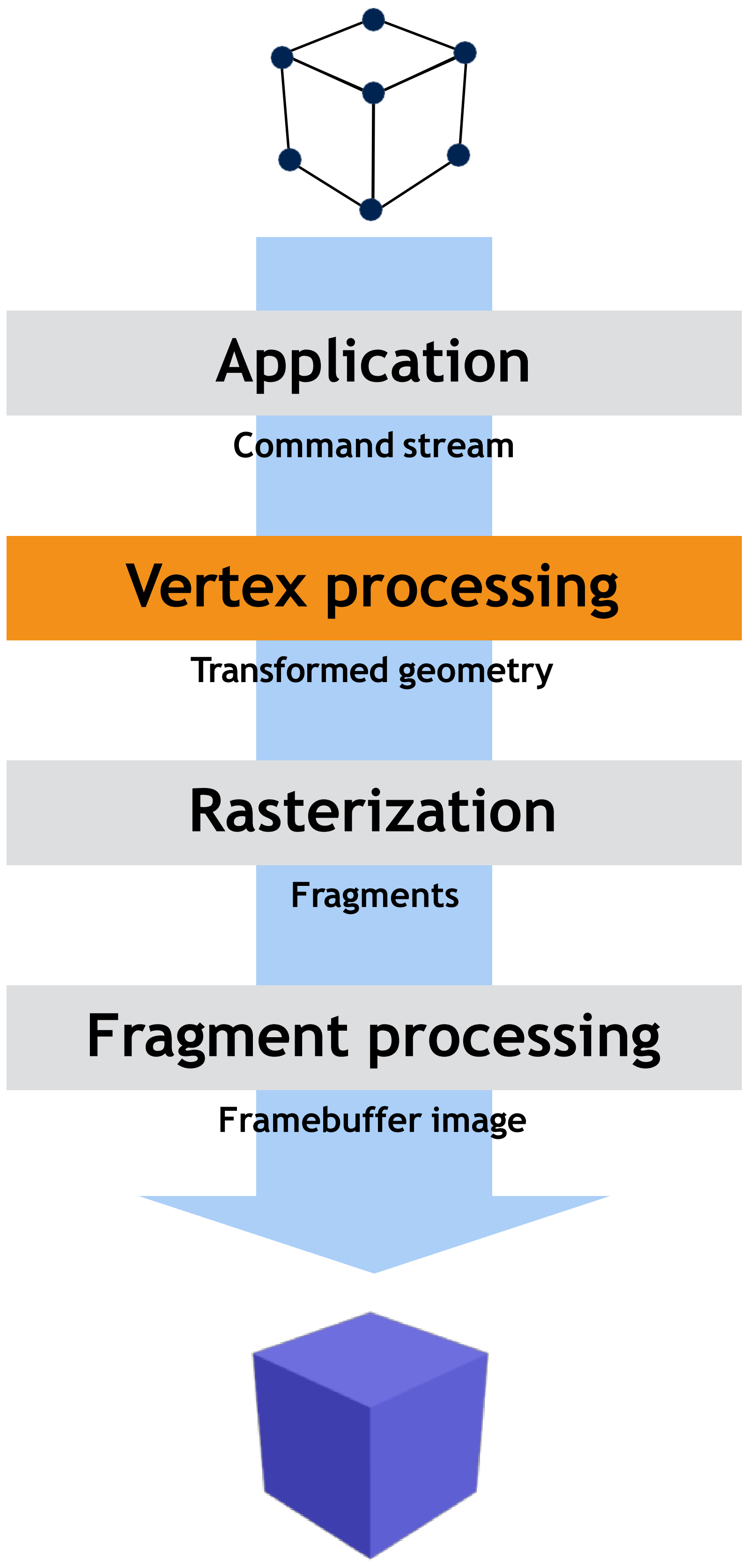
Vertex stage

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



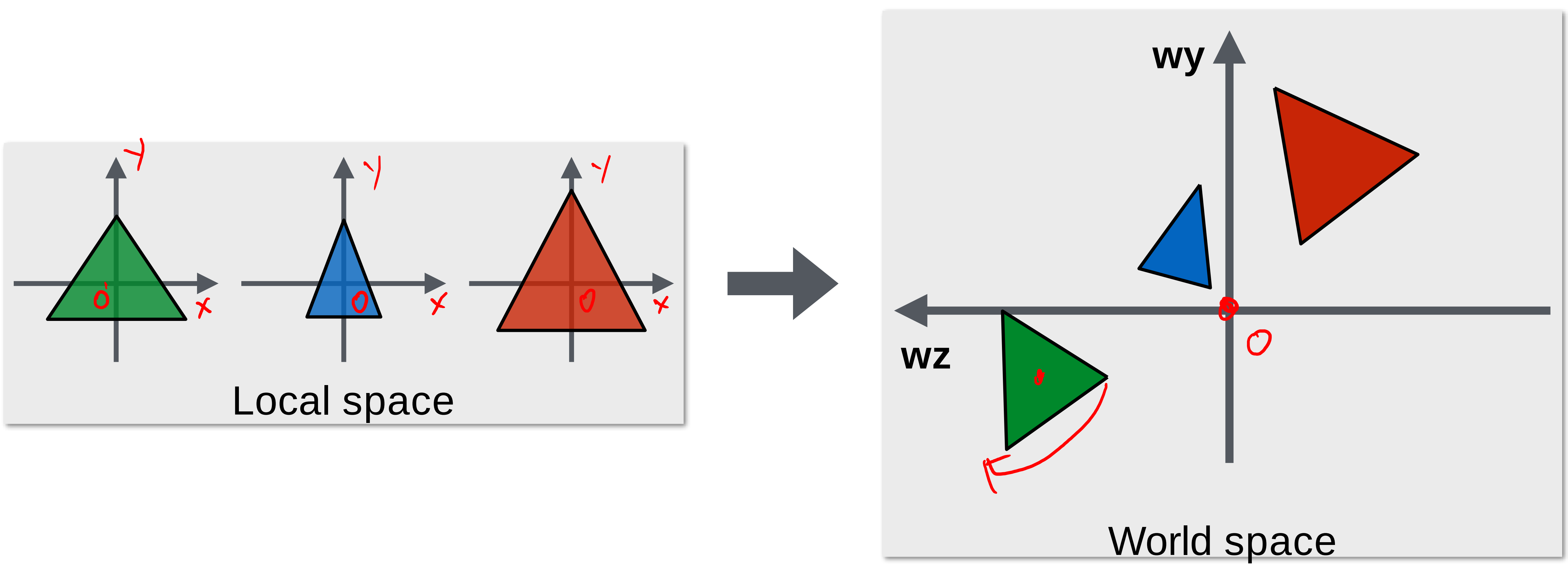
Model and view transform

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



Model transform

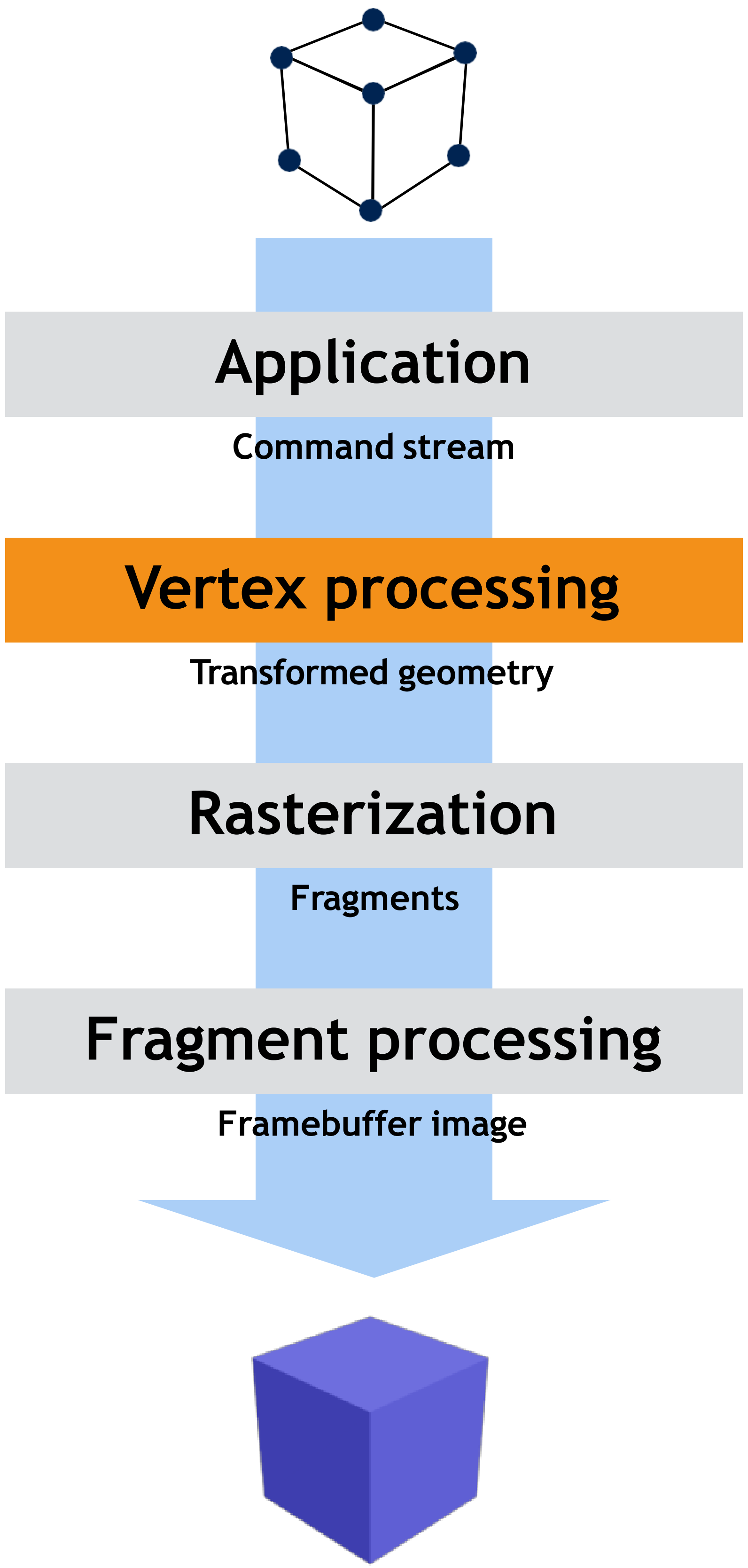
- Change the position and orientation of objects
- Possible deformation of objects



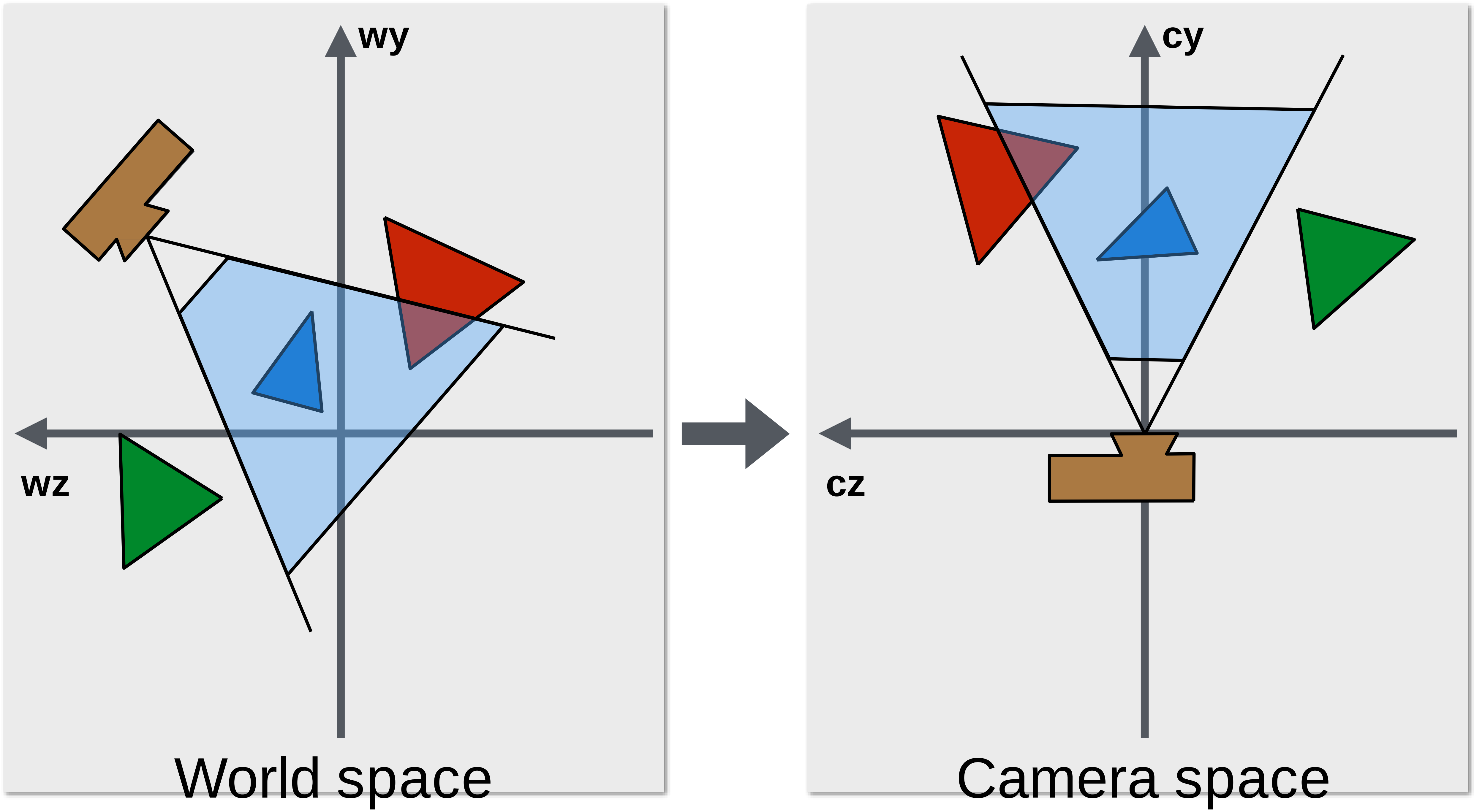
View transform

Camera transform

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

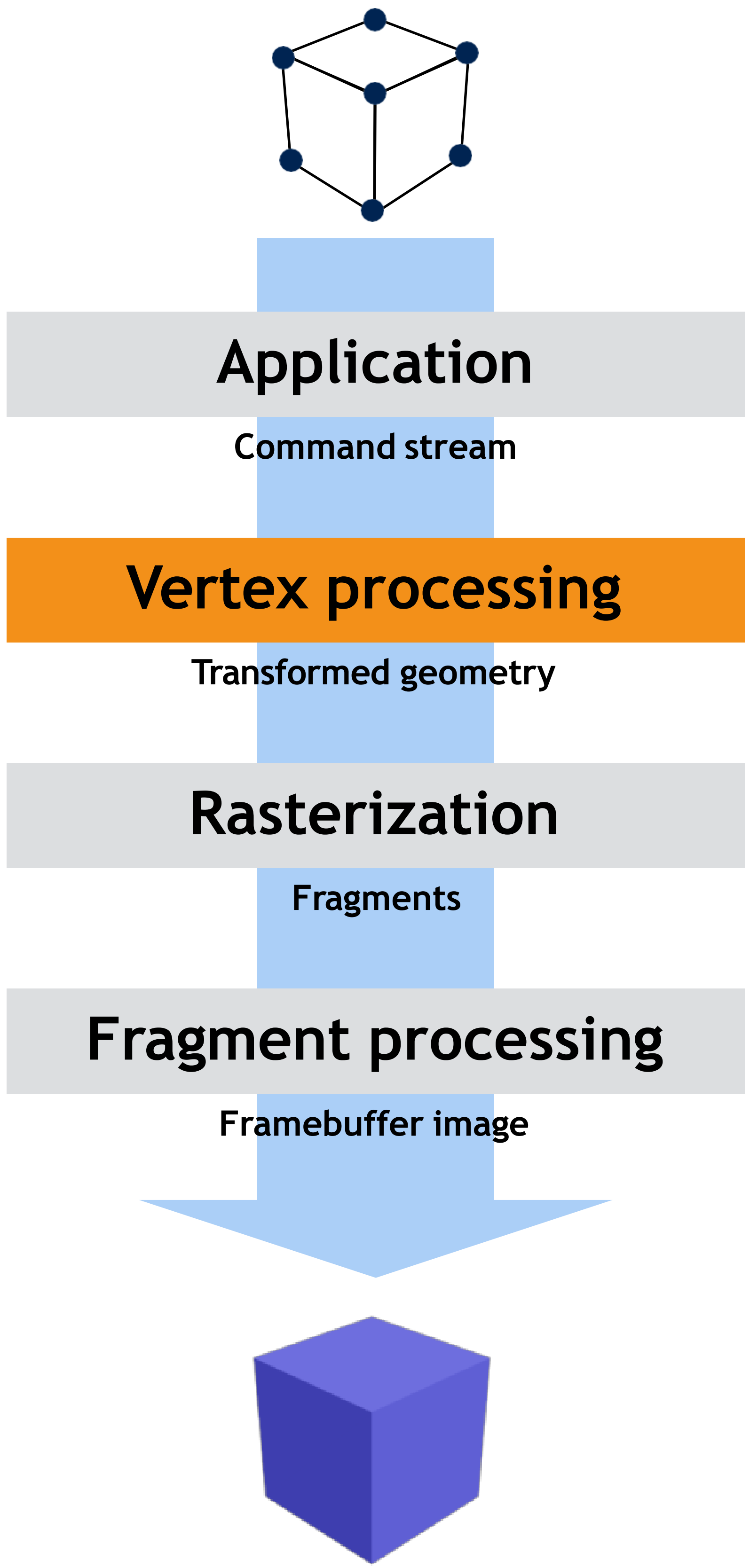


View transform



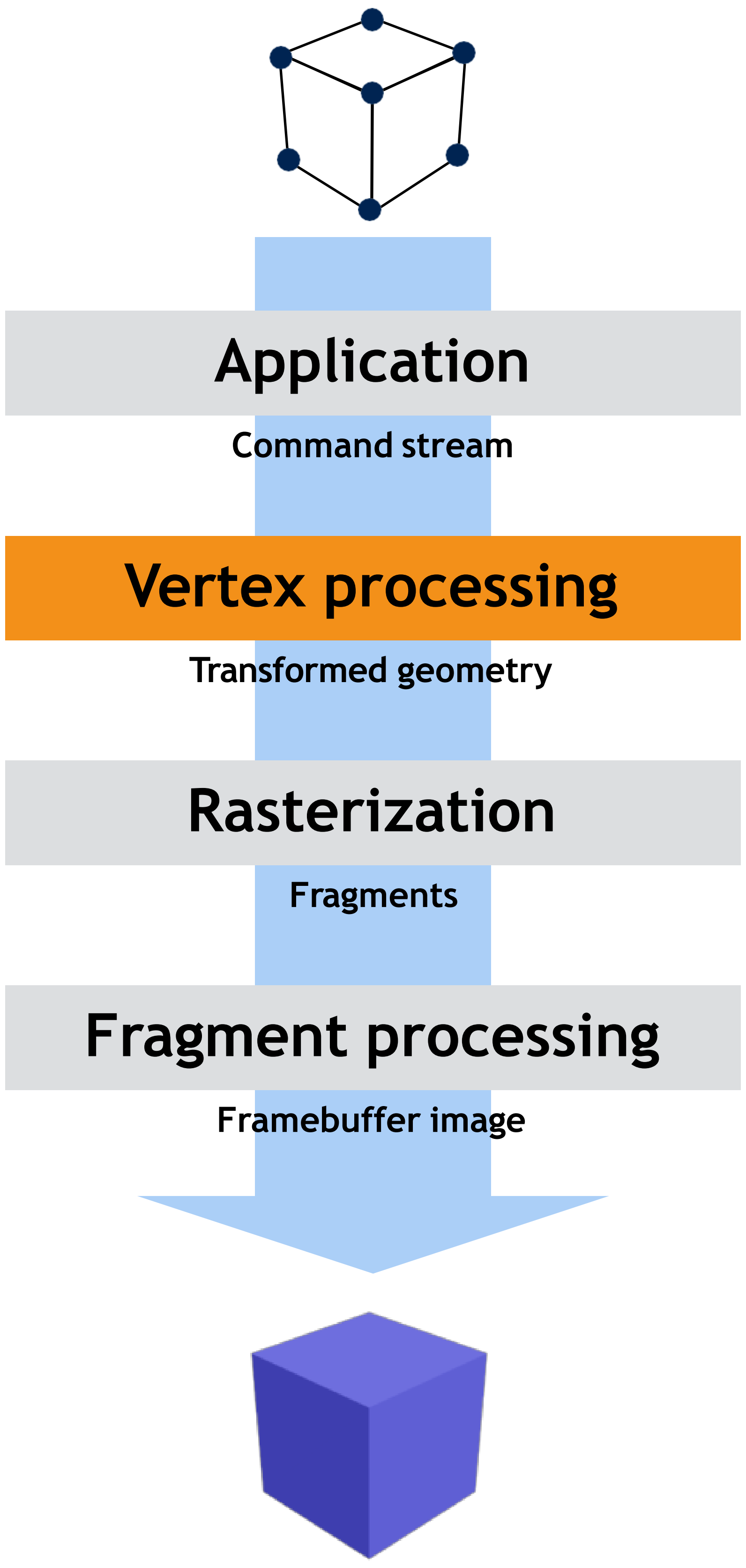
Vertex shading

- 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

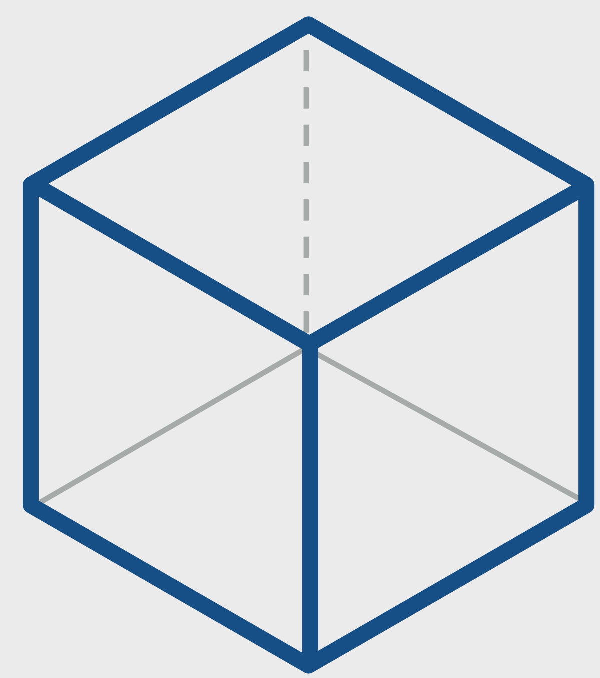


Projection

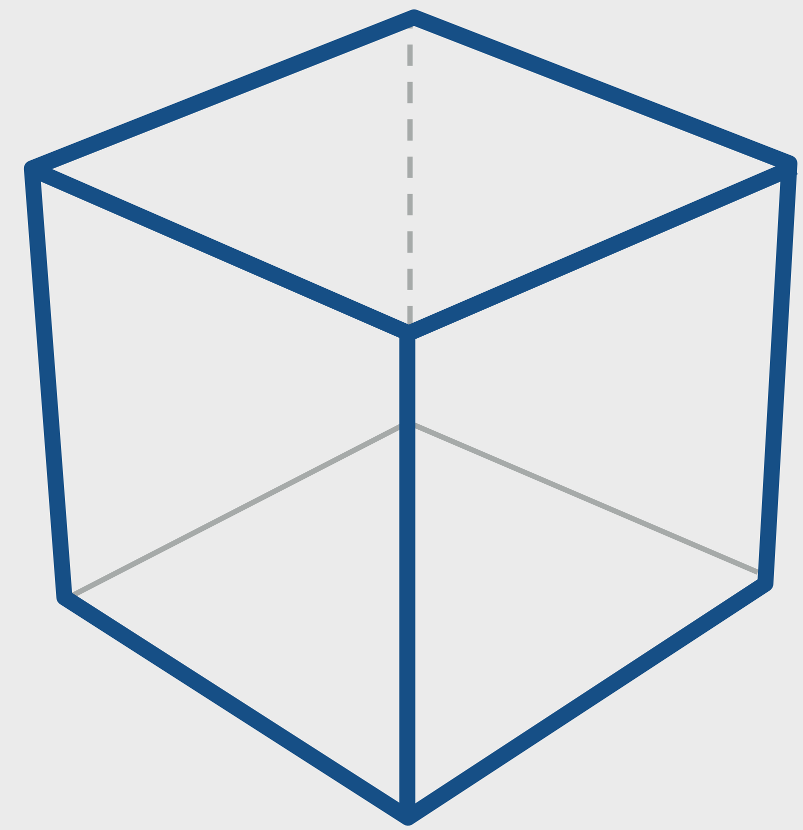
- Transforms the view volume into a unit cube - **canonical view volume**
- Can be defined between (−1, −1, −1) and (1,1,1) or (0, 0, 0) and (1,1,1)
- Most commonly used projections
  - **orthographic** (parallel) projection
  - **perspective** projection
- After projection the 3D models are in **normalized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



Projections



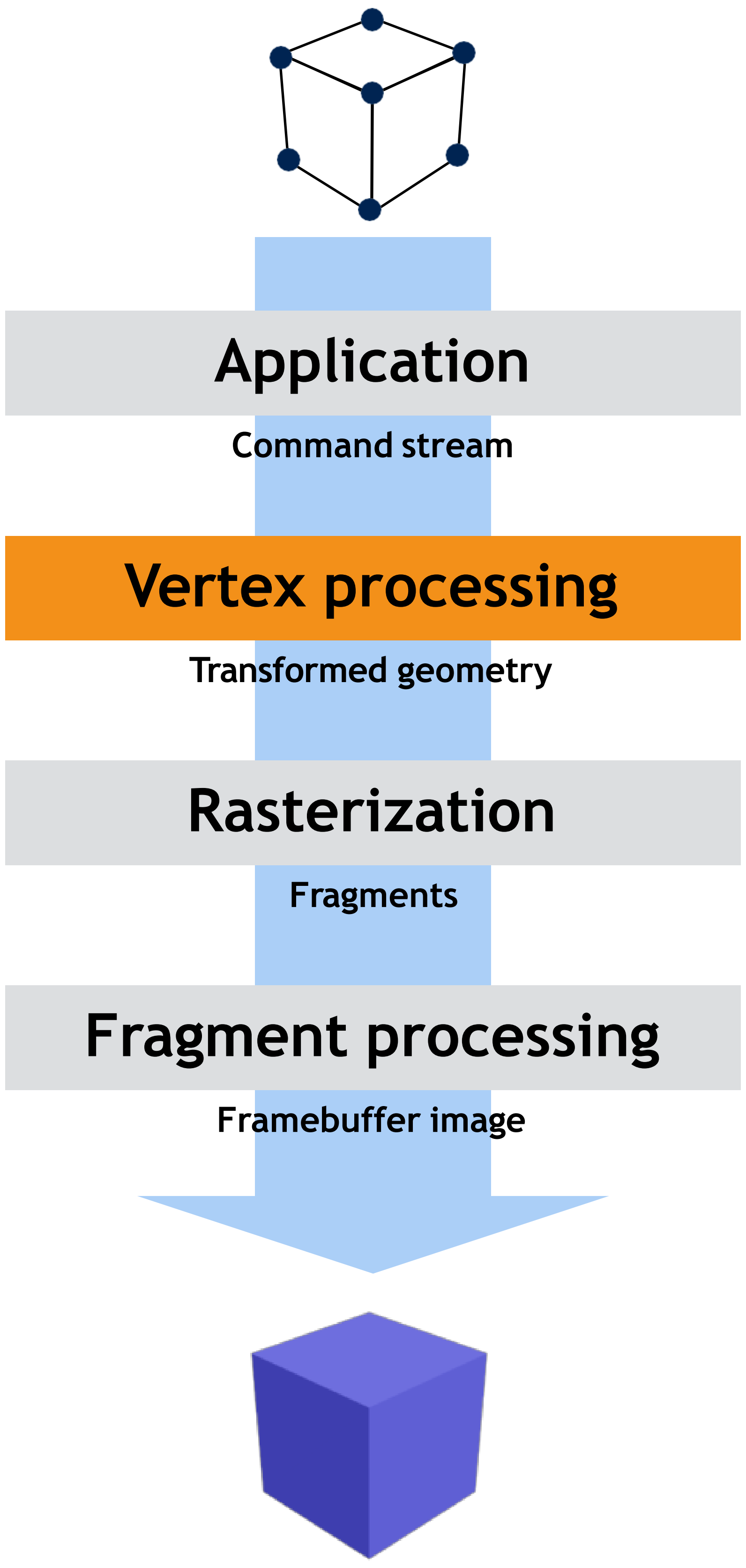
Orthographic projection



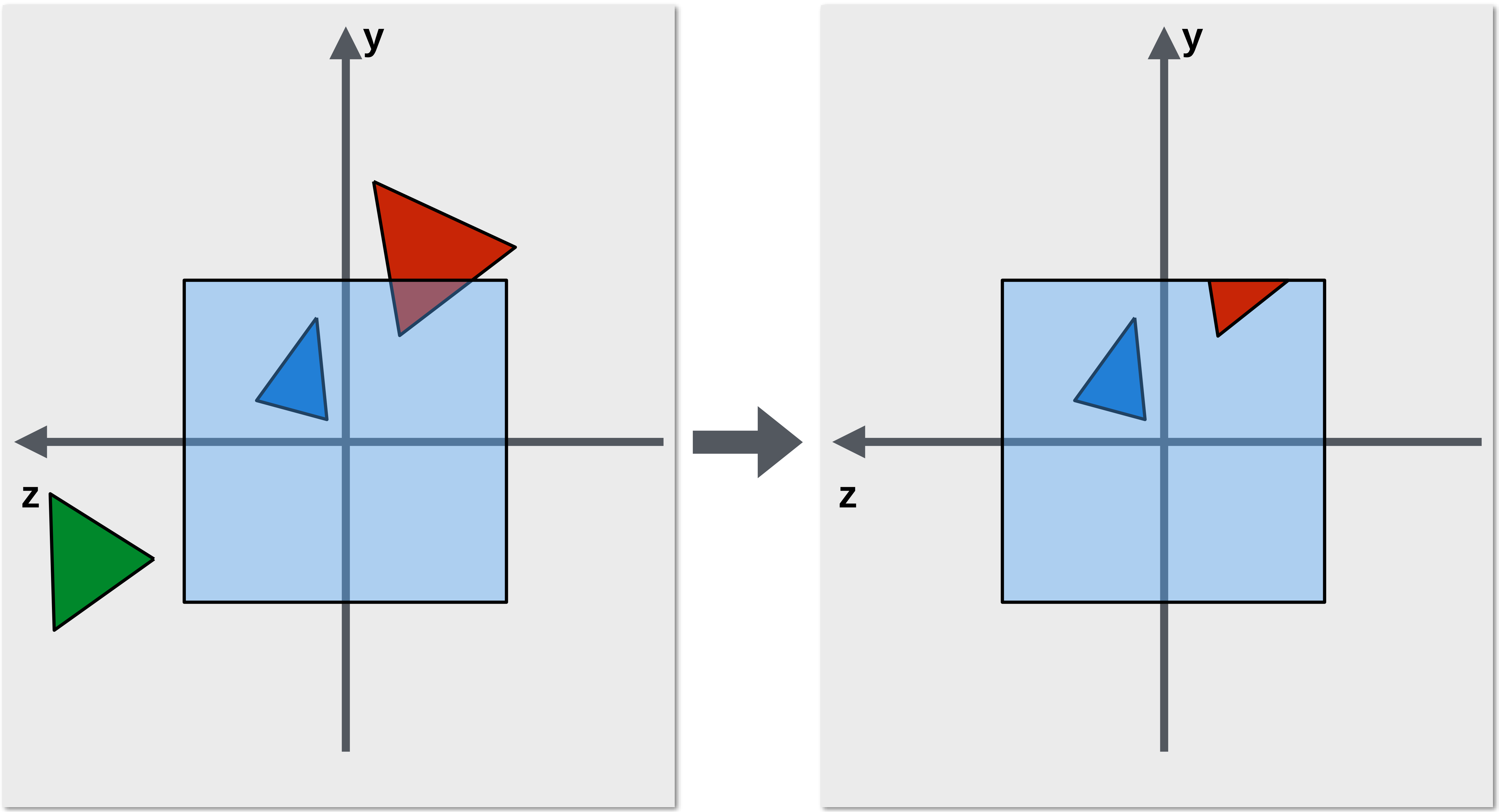
Perspective projection

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 is
- Primitives that are entirely outside - do not pass them further
- Primitives that are partially inside - clip them and pass to the next stage

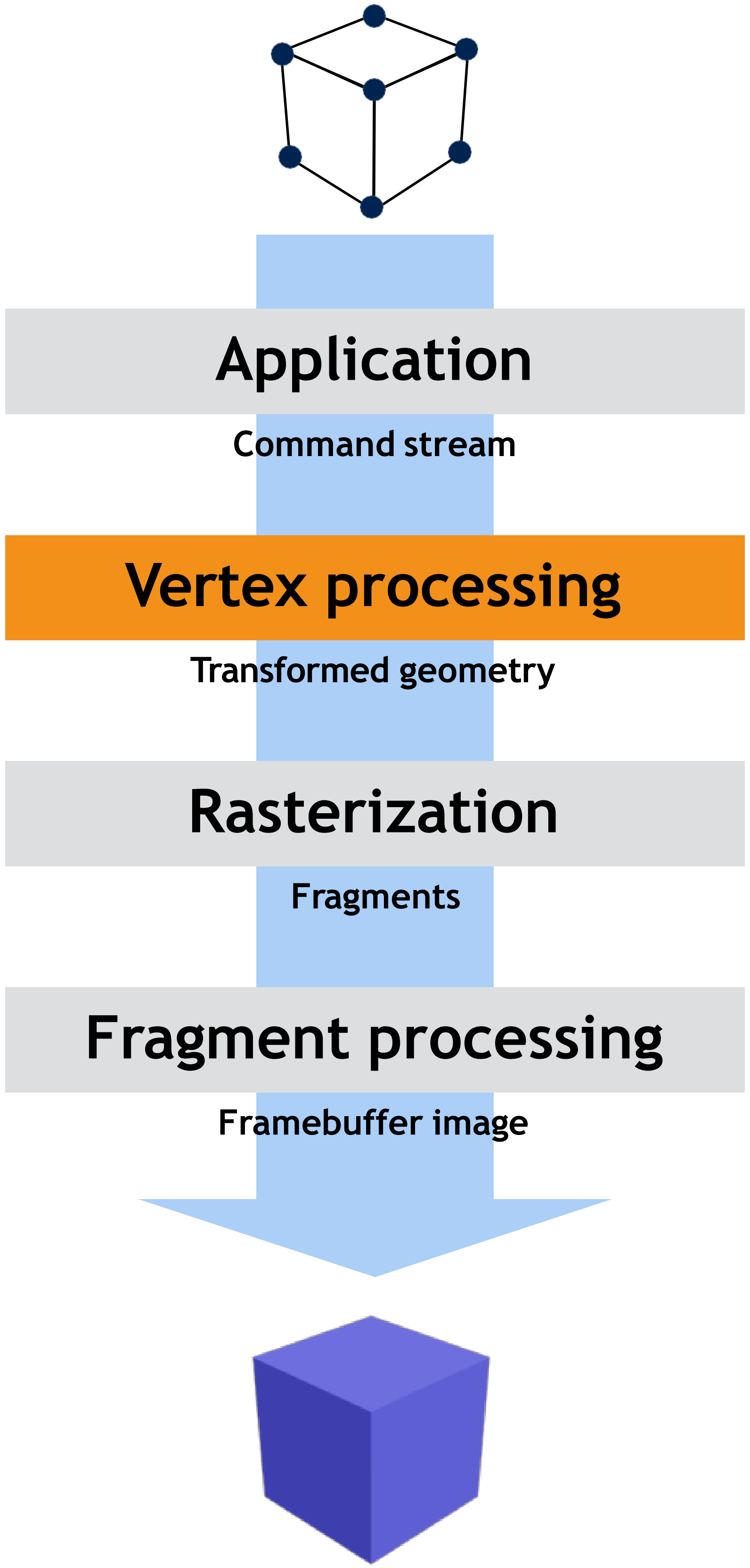


Clipping

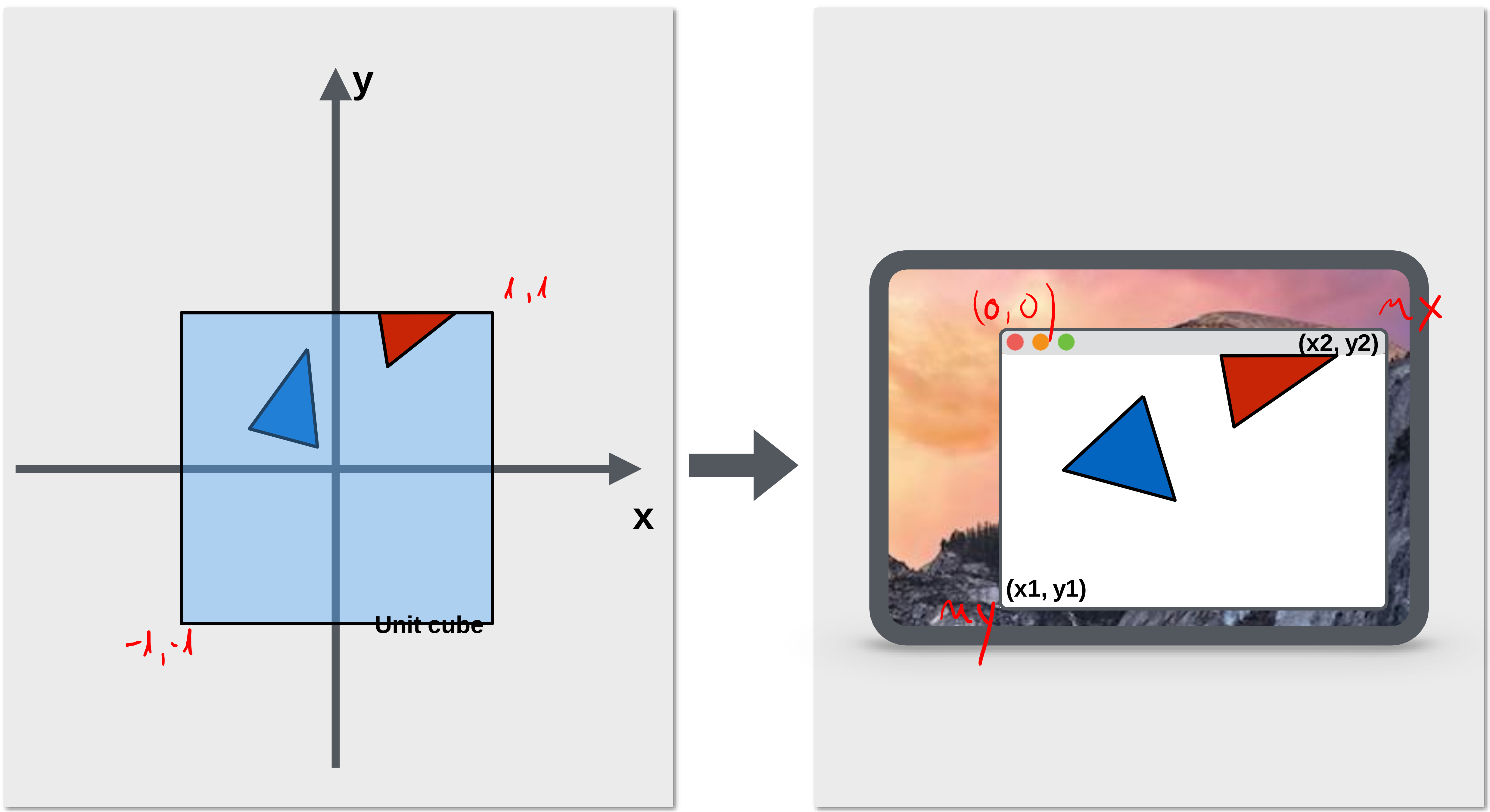


Screen mapping

- 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

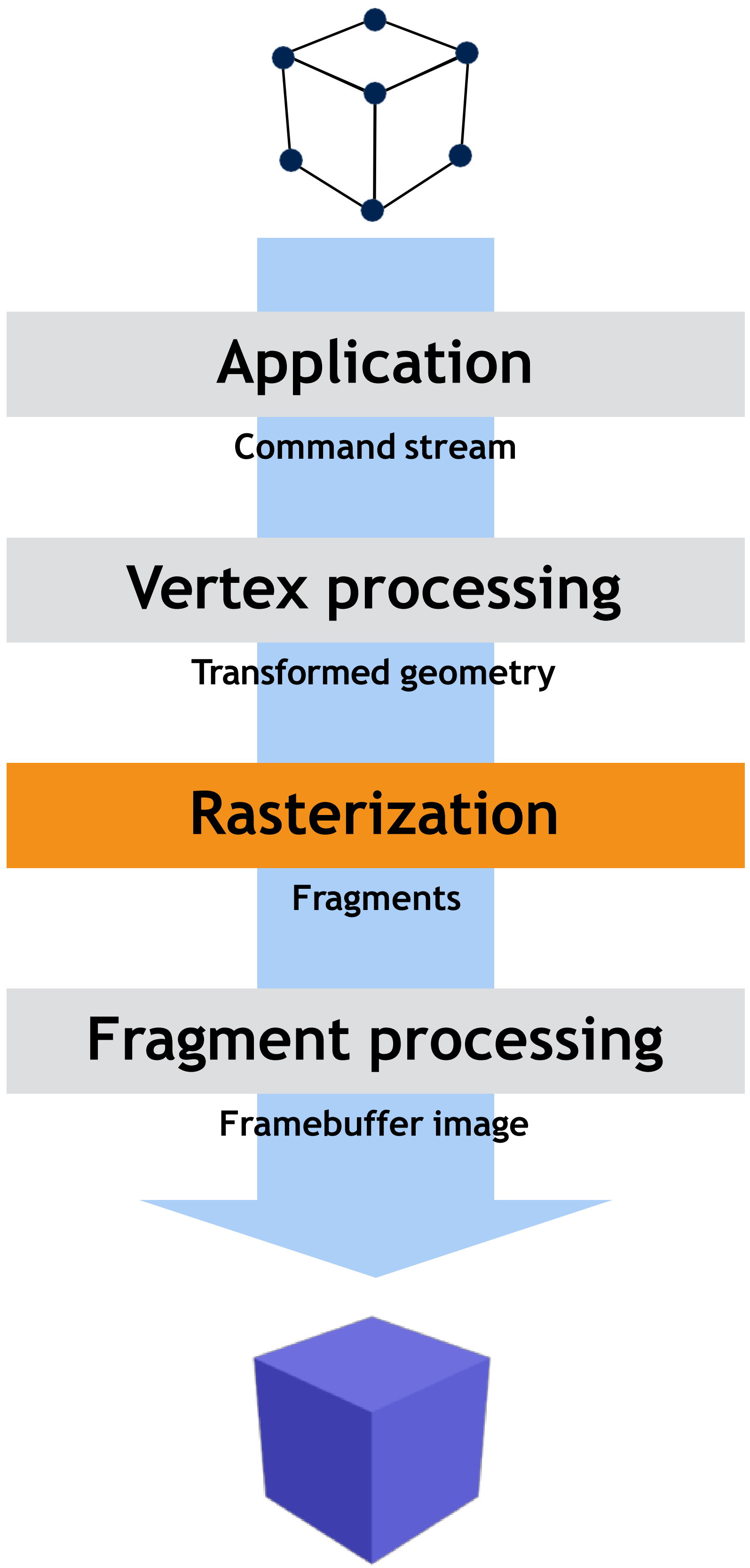


Screen mapping

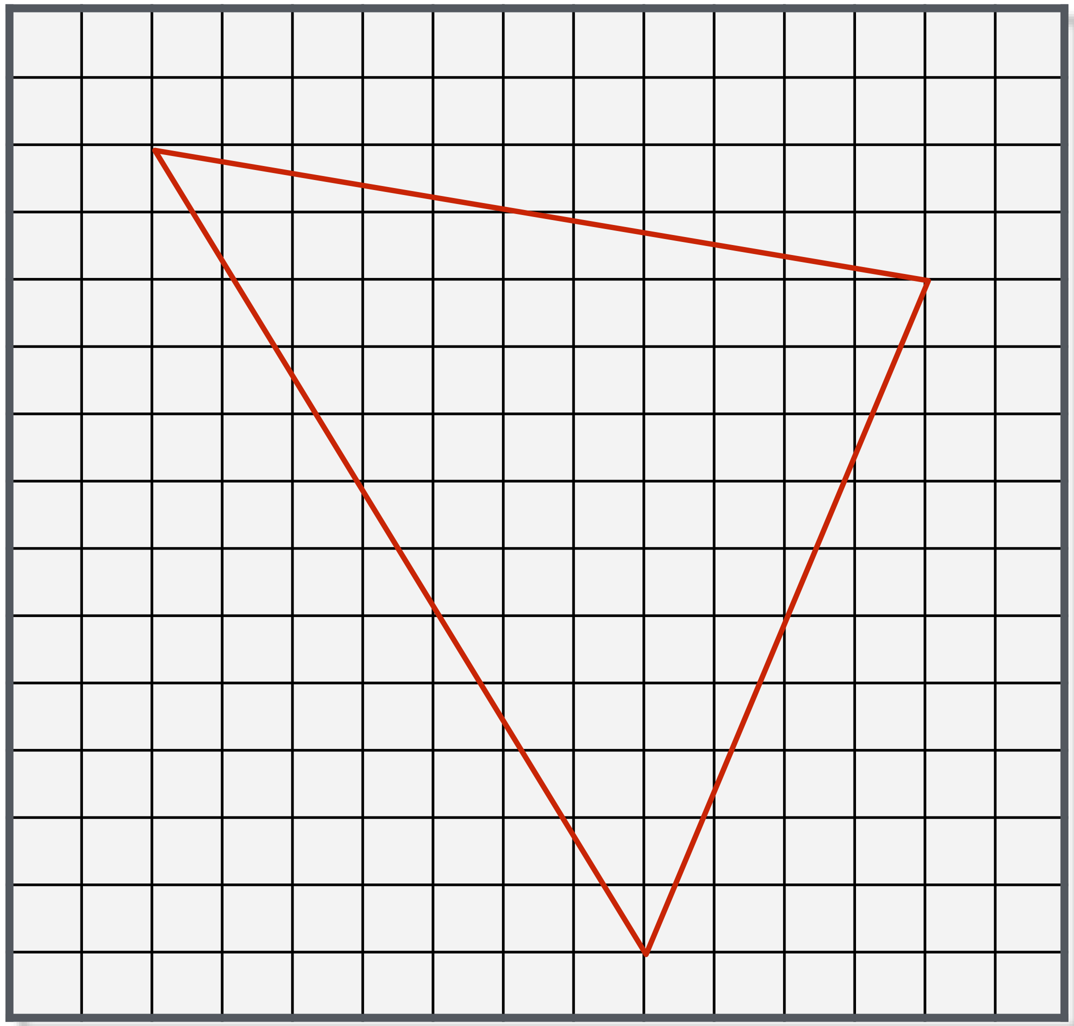


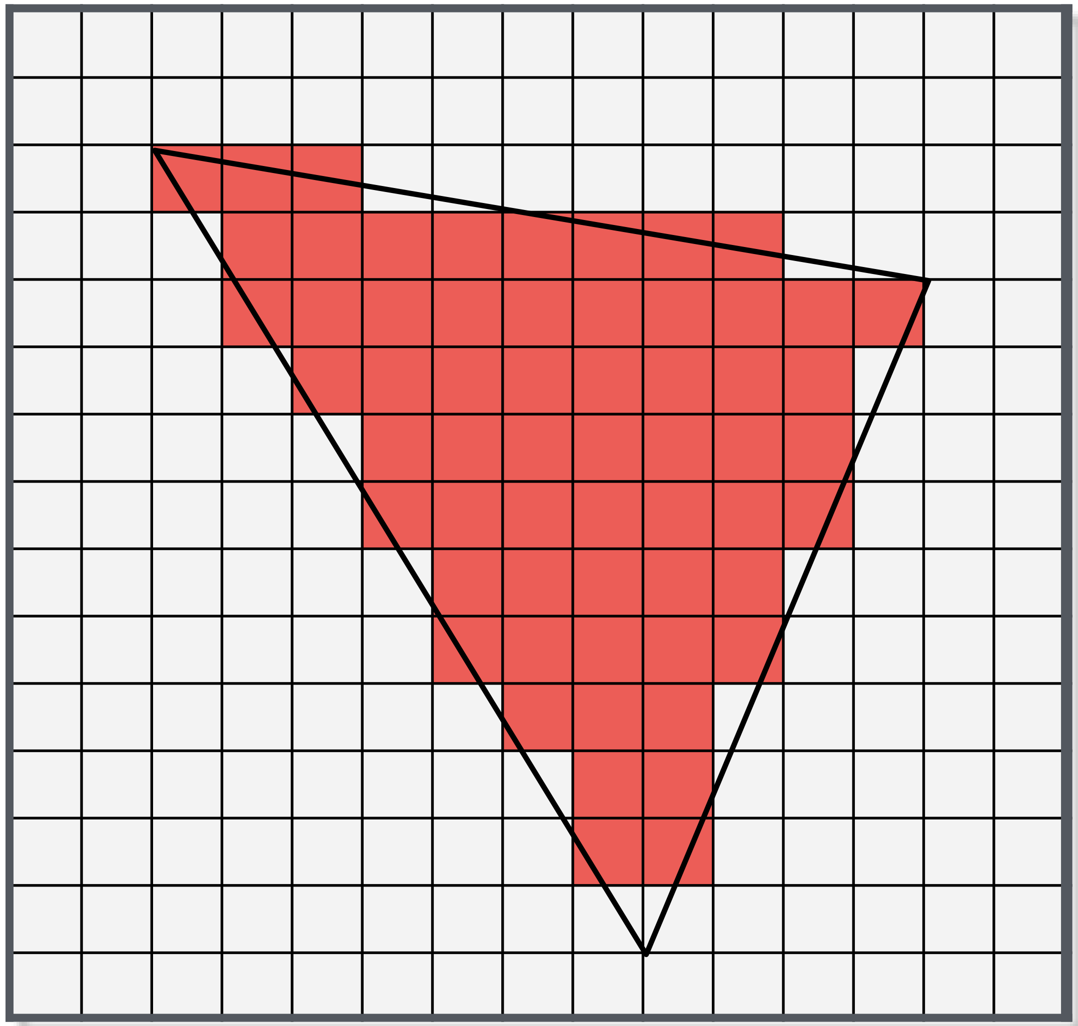
Rasterizer Stage

- 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



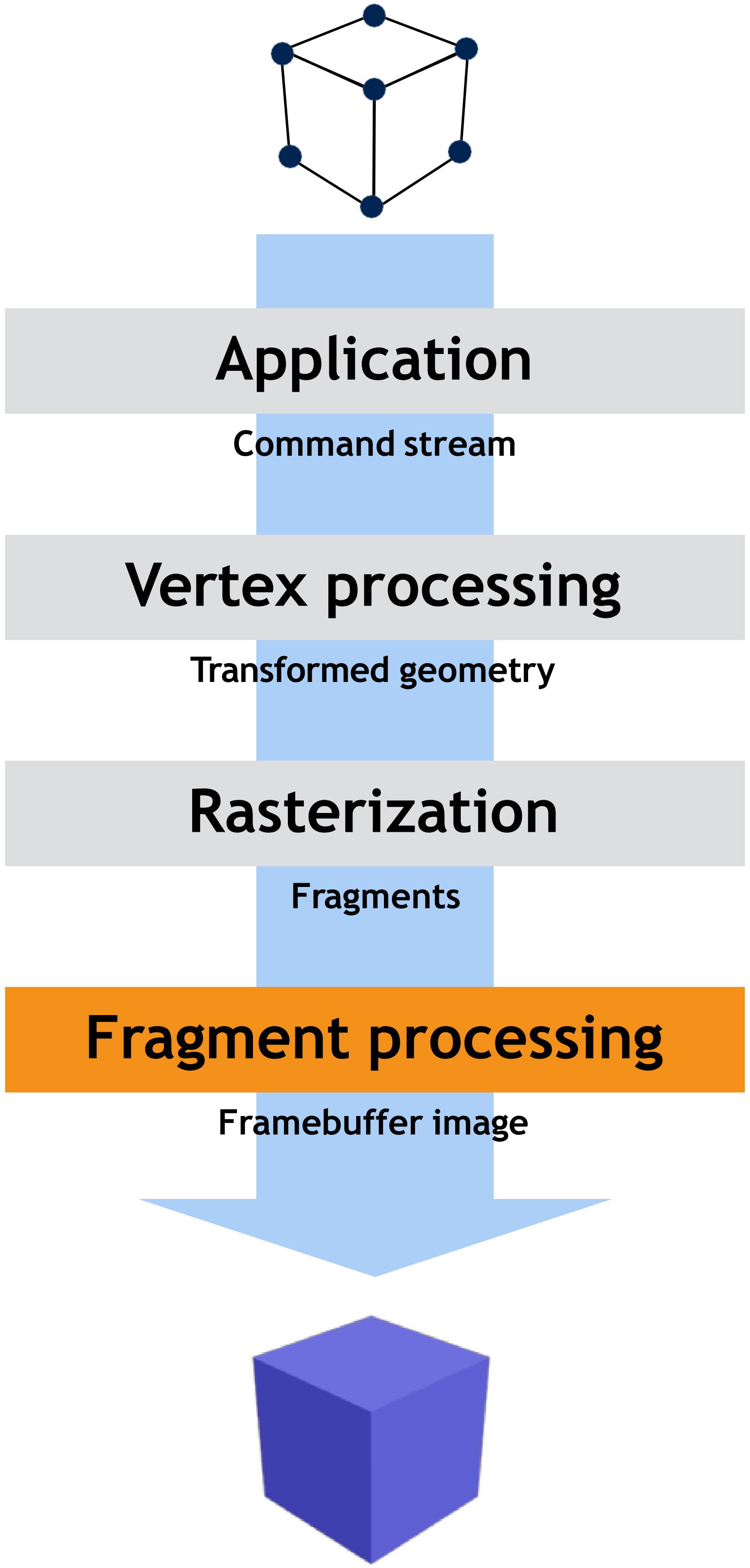
Example of rasterization



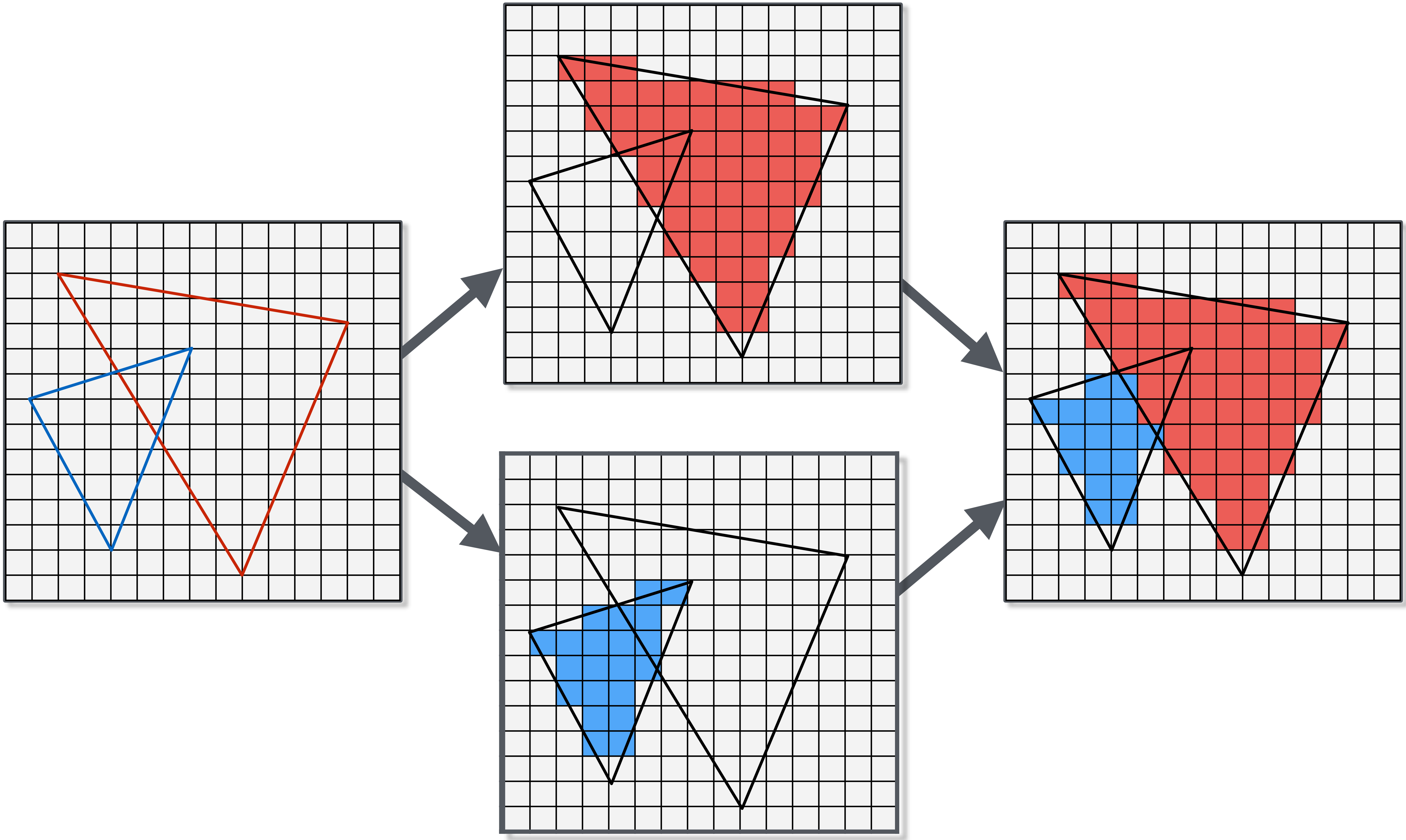


Fragment processing

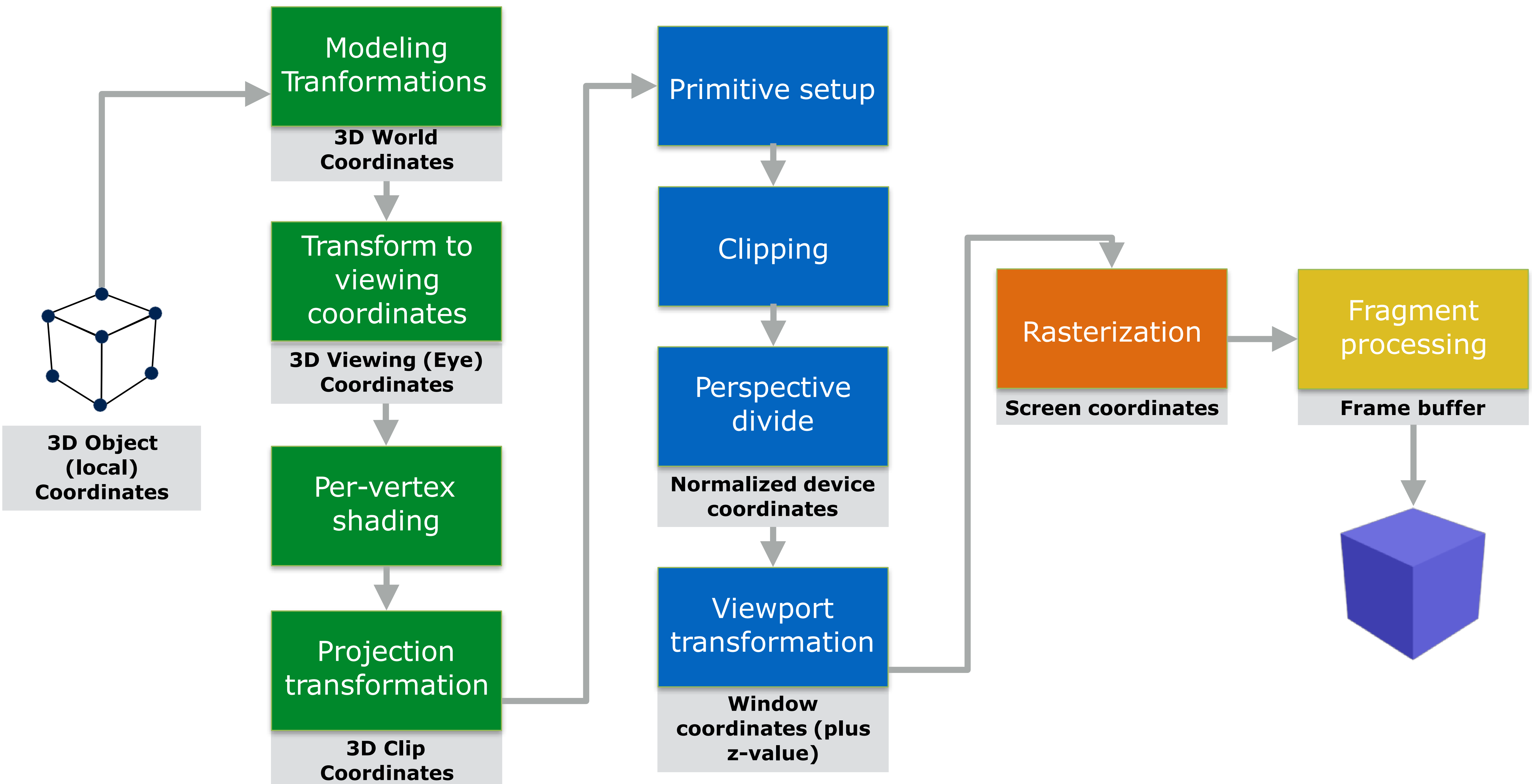
- Process each fragment from the rasterization process into a set of colors and a single depth value.
- **Scissor test** - discard fragments outside of a certain rectangular portion of the screen
- **Stencil test** - test the fragment's stencil value against the value in the current stencil buffer; if 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



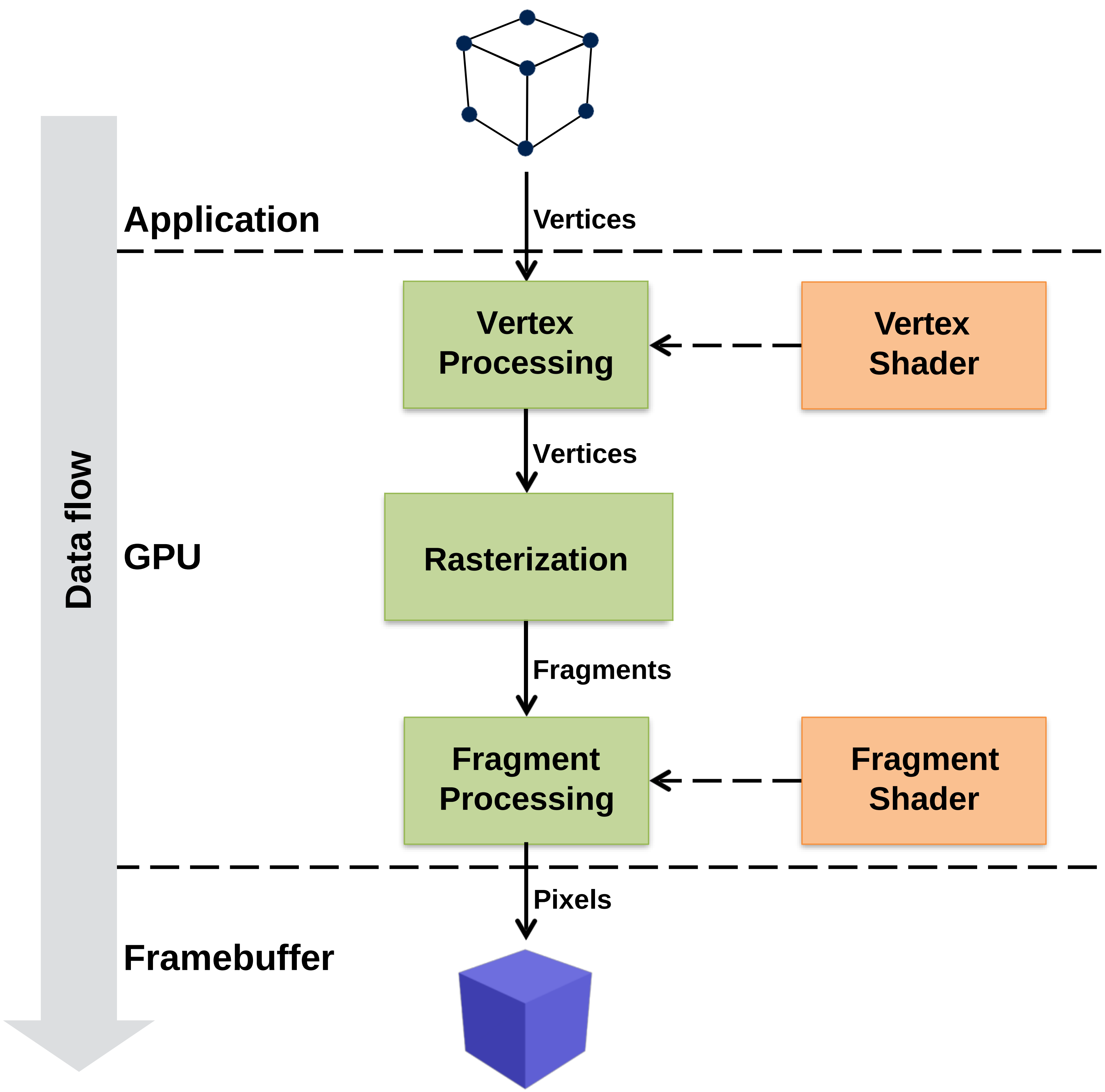
Fragment processing



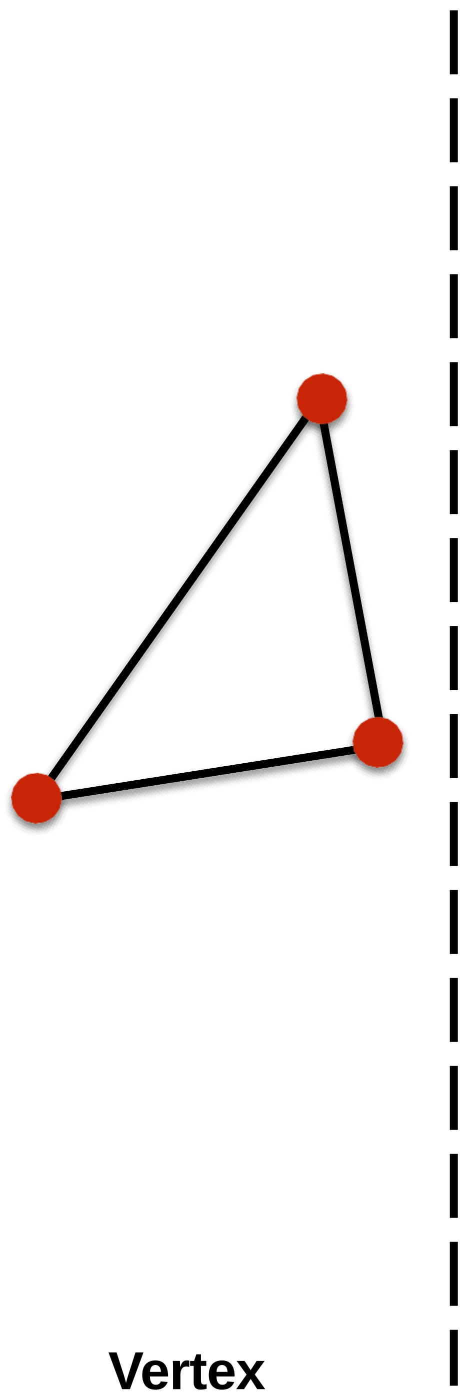
Coordinates systems



Programmable graphics pipeline

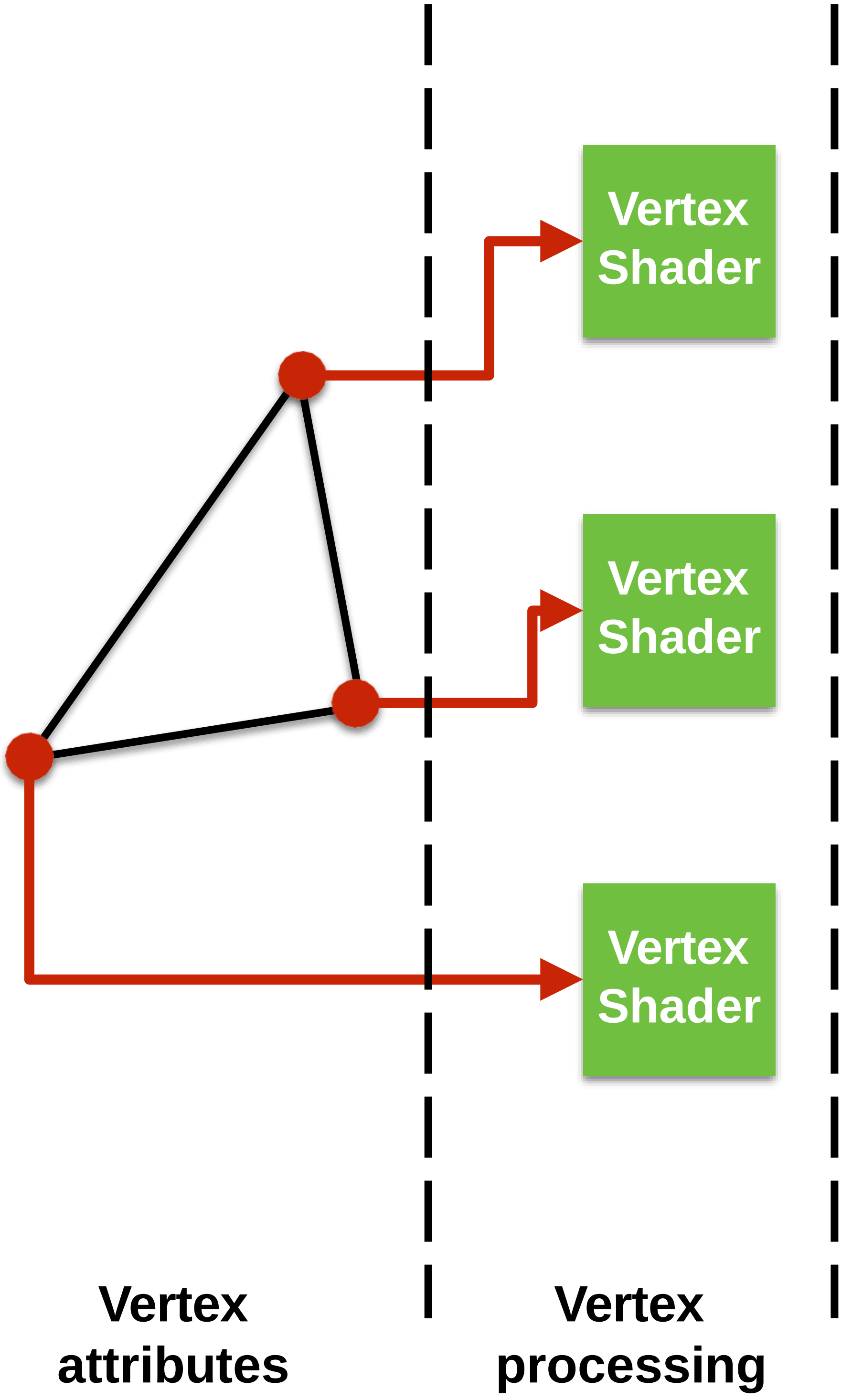


Programable graphics pipeline

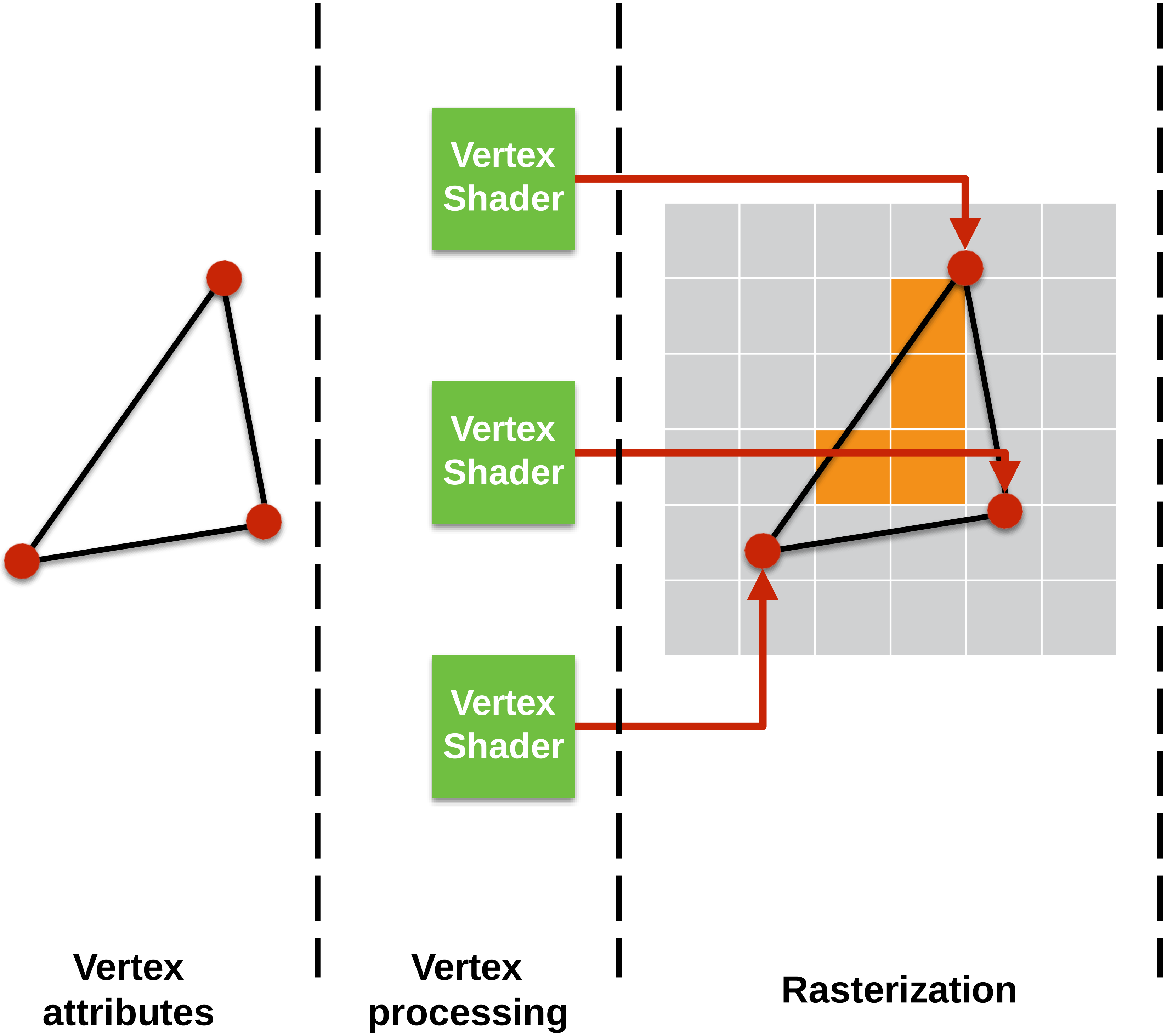


Vertex attributes

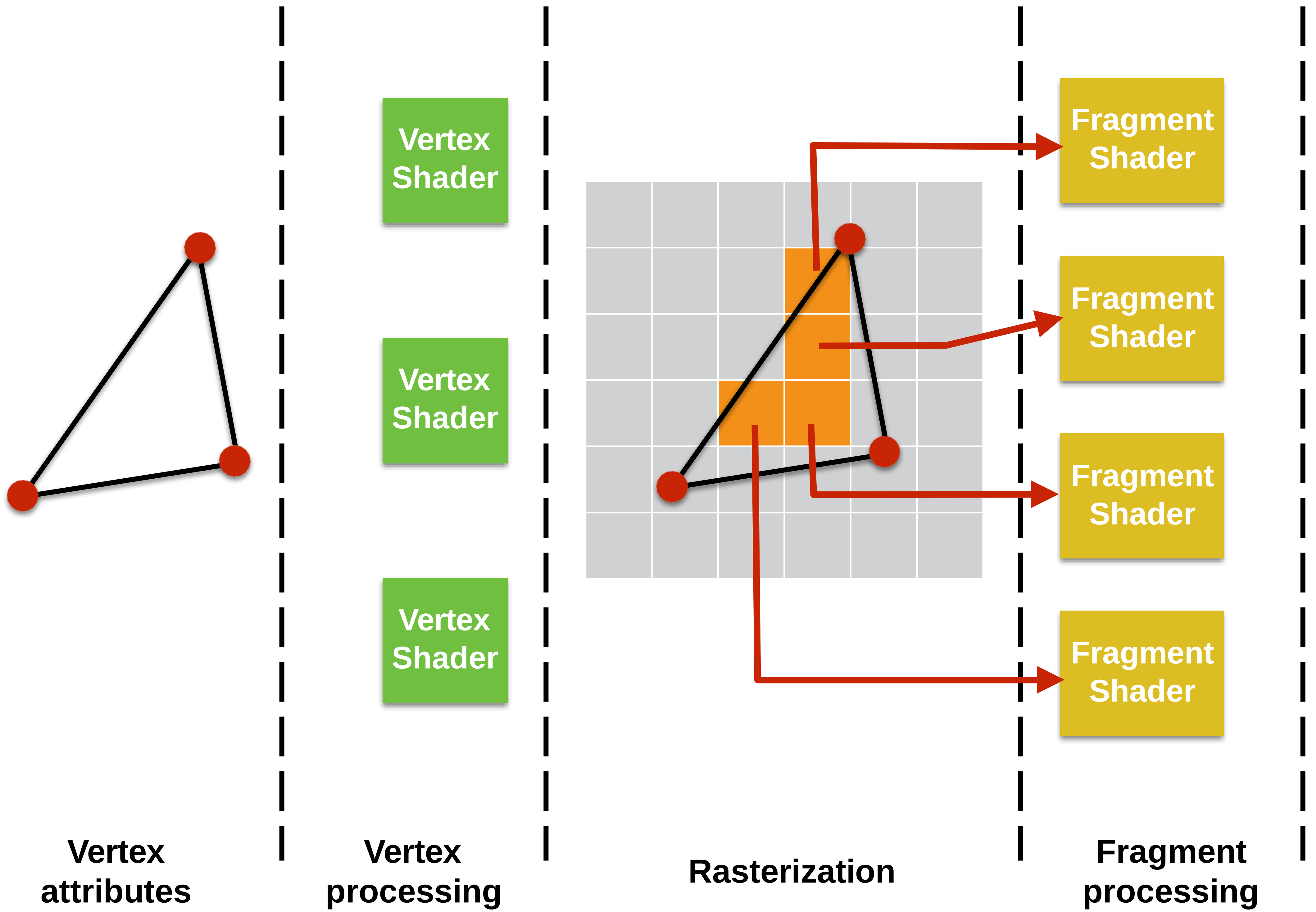
Programable graphics pipeline



Programable graphics pipeline



Programable graphics pipeline



Programable graphics pipeline