COMP3271 Computer Graphics

The Graphics Pipeline

2019-20

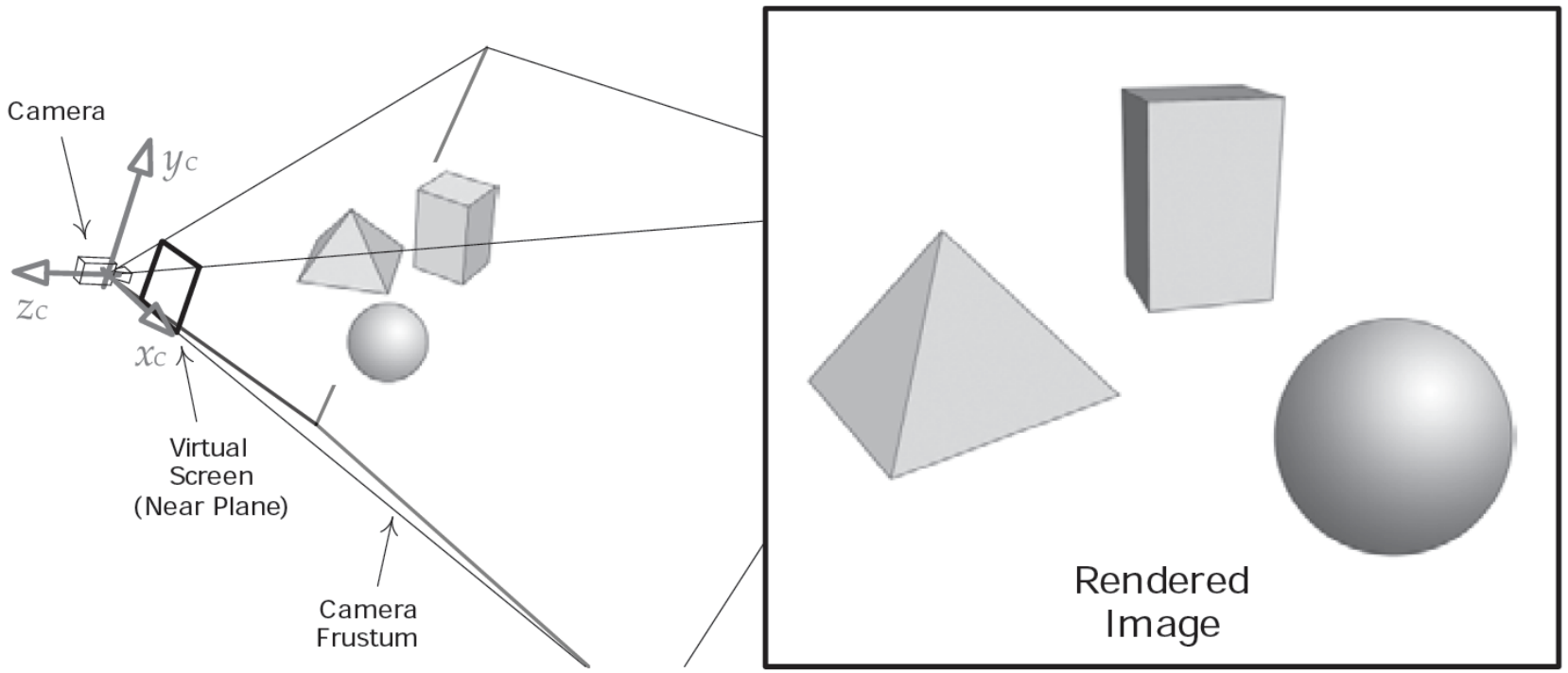
Objectives

Learn the basic design of a graphics system

Introduce pipeline architecture

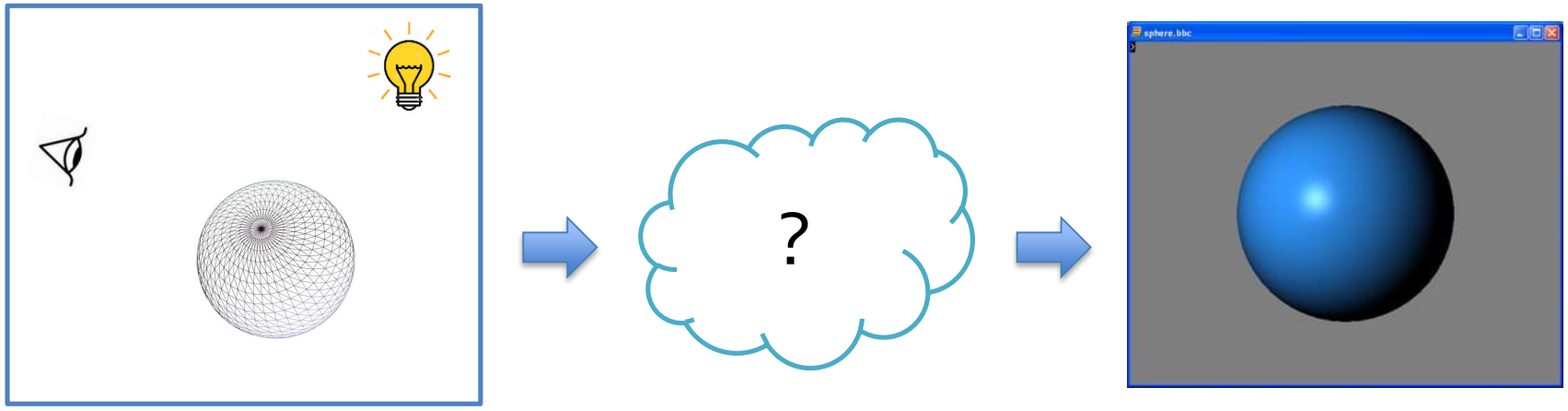
Examine software components for an interactive graphics system

Image Formation Revisited



Can we mimic the synthetic camera model to design graphics hardware software?

Graphics System Overview



Graphics Application

Rendering on Screen

Need a Graphics Application Programmer Interface (API)

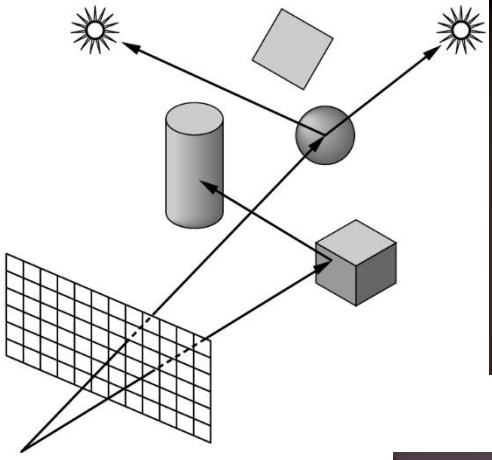
- Graphics applications then only need to deal with scene specification / modeling by specifying
  - Objects
  - Materials
  - Viewer
  - Lights

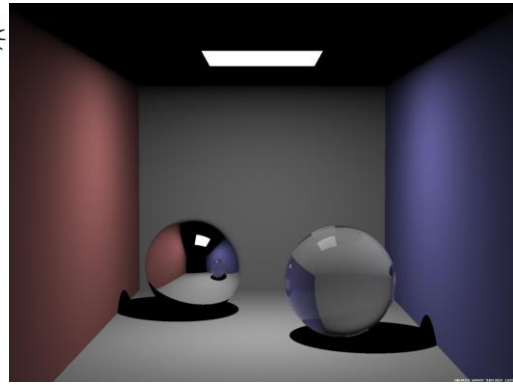
But how is the API implemented?

Physical Approaches

Ray tracing: follow rays of light from center of projection until they either are absorbed by objects or go off to infinity

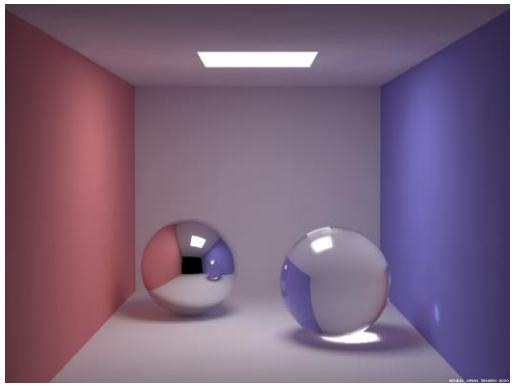
- Can handle global effects
  - Multiple reflections
  - Translucent objects
- Slow
- Must have whole database available at all times





Radiosity: Energy based approach

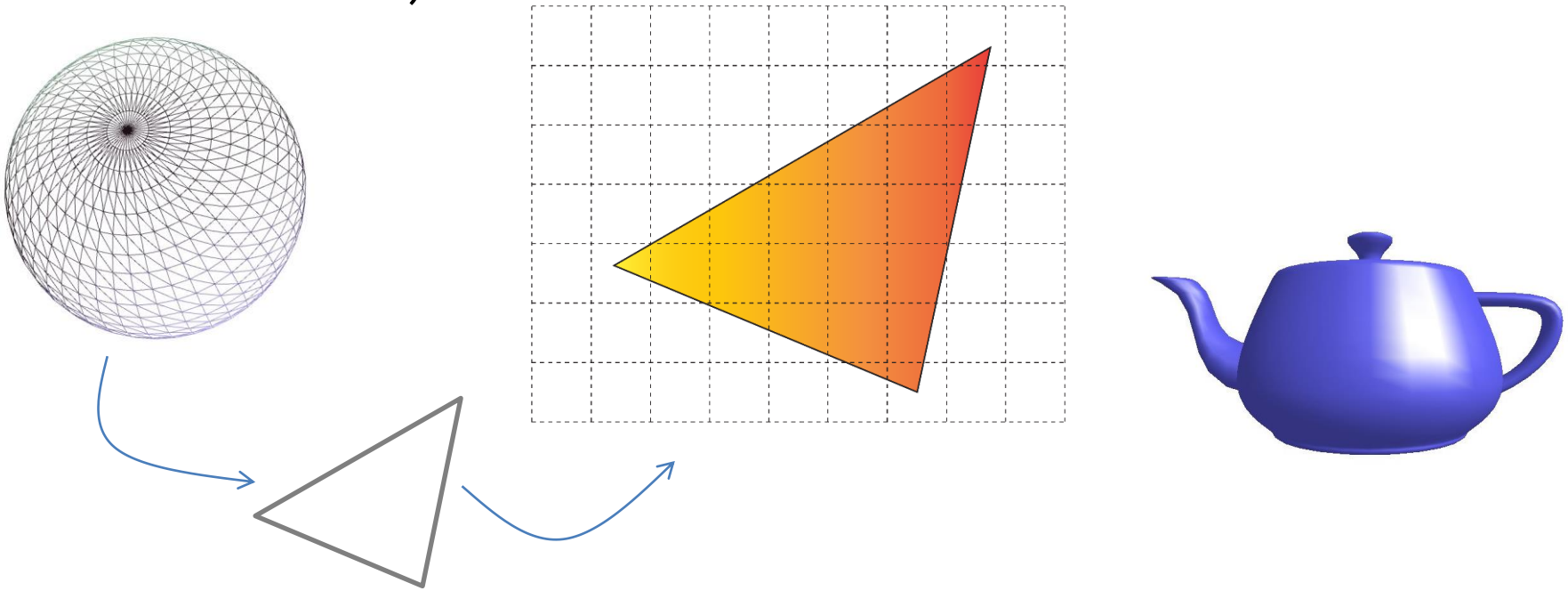
- Can handle diffuse inter-reflection among objects
- Very slow



Practical Approach

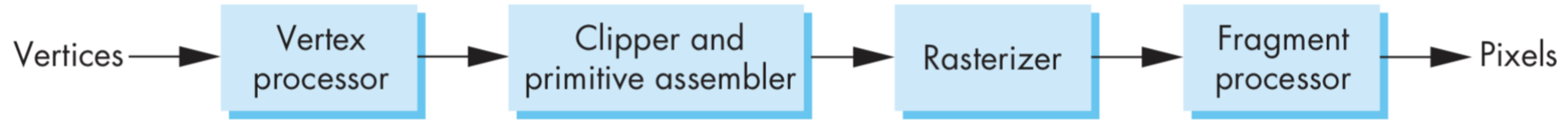
Process objects one at a time in the order they are generated by the application

- Consider only local lighting (e.g., no inter-object light interaction)



The Graphics Pipeline

Pipeline architecture



application program

All steps can be implemented in hardware on the graphics card

display

Graphics Pipeline

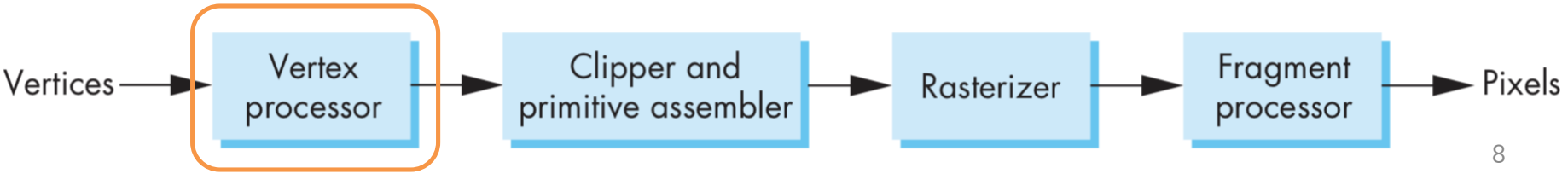
Vertex Processing

Much of the work in the pipeline is in converting object representations from one coordinate system to another

- Object coordinates
- World coordinates
- Camera (eye) coordinates
- Screen coordinates

Every change of coordinates is equivalent to a matrix transformation

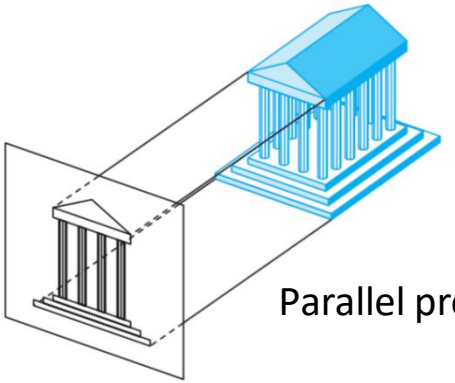
Vertex processor also computes vertex colors



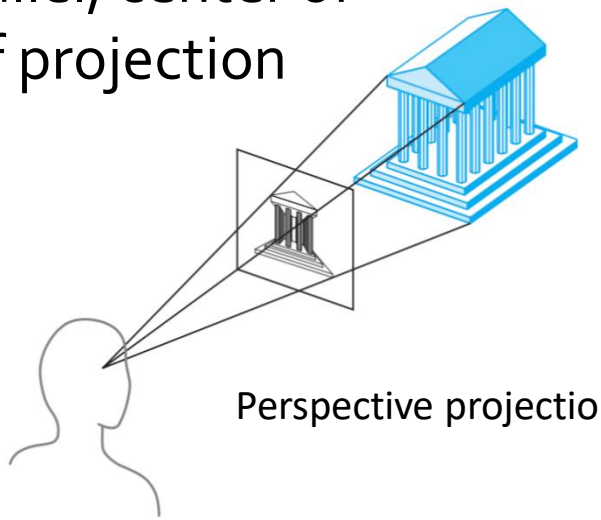
Projection

Projection is the process that combines the 3D viewer with the 3D objects to produce the 2D image

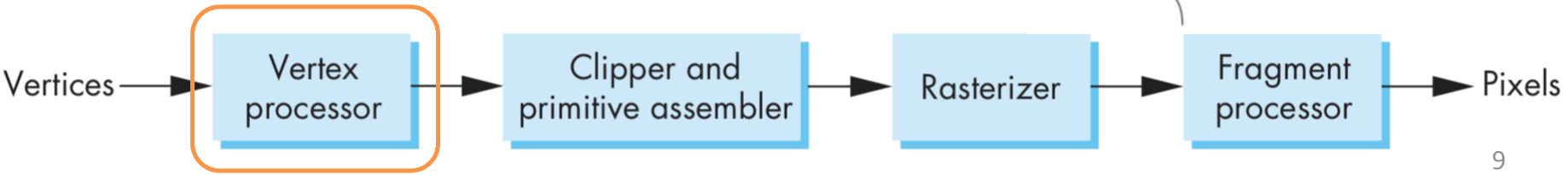
- Perspective projections: all projectors meet at the center of projection
- Parallel projection: projectors are parallel, center of projection is replaced by a direction of projection



Parallel projections



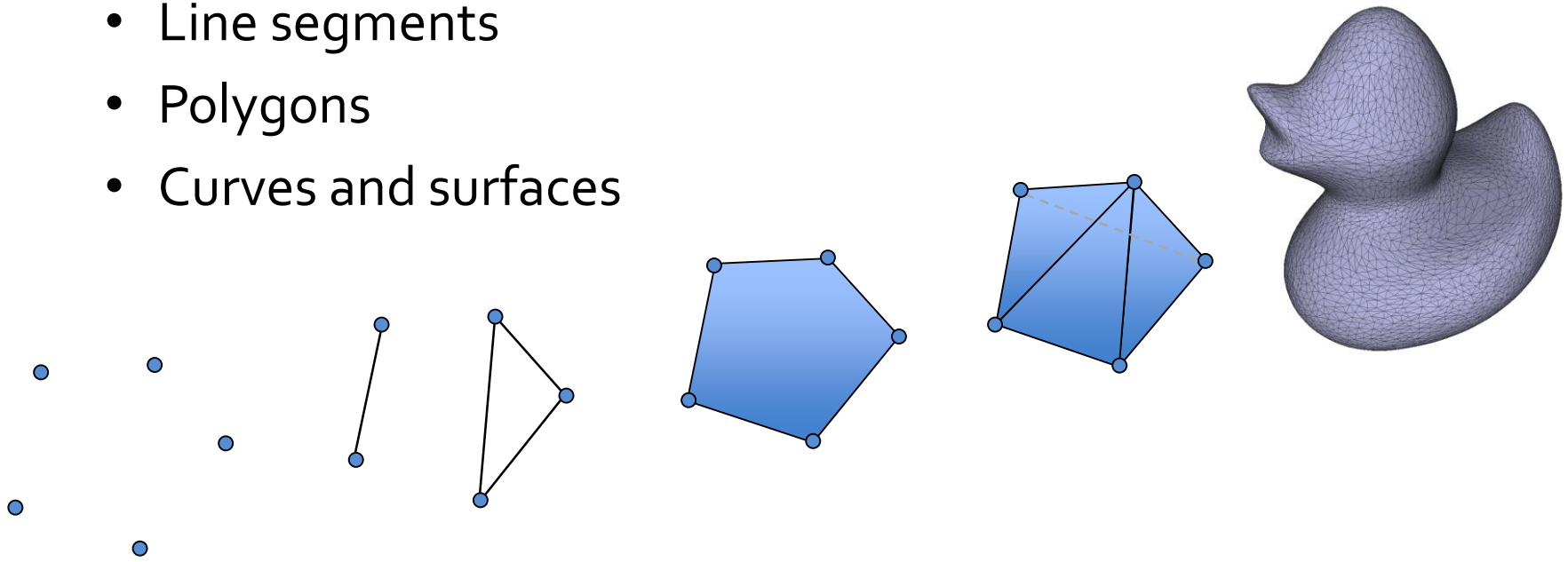
Perspective projections

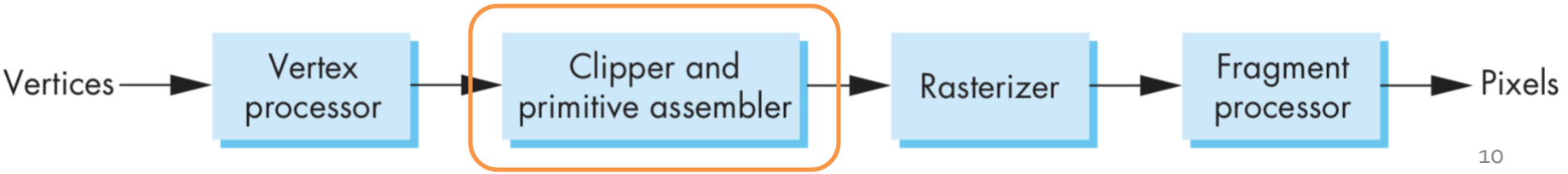


Primitive Assembly

Vertices must be collected into geometric objects before clipping and rasterization can take place

- Line segments
- Polygons
- Curves and surfaces

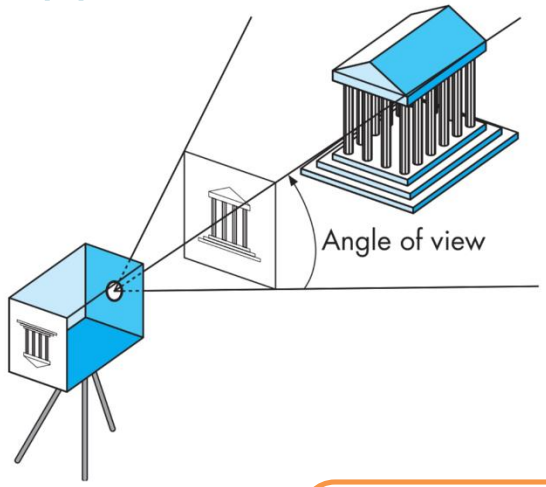


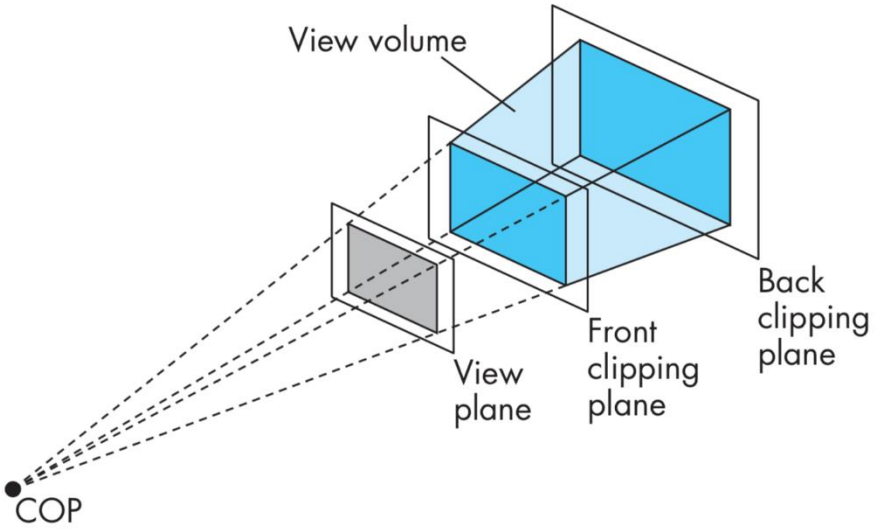


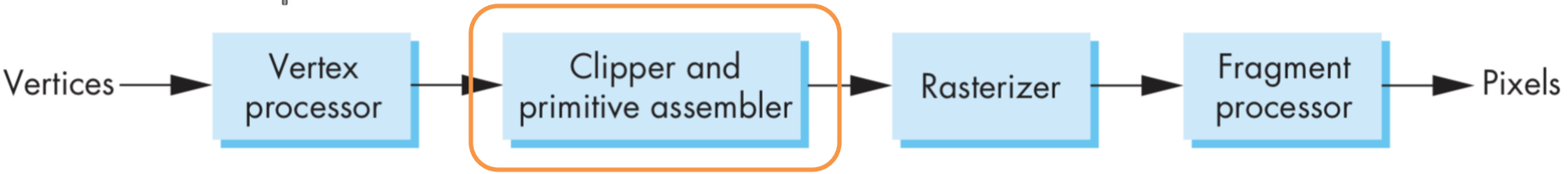
Clipping

Just as a real camera cannot "see" the whole world, the virtual camera can only see part of the world or object space

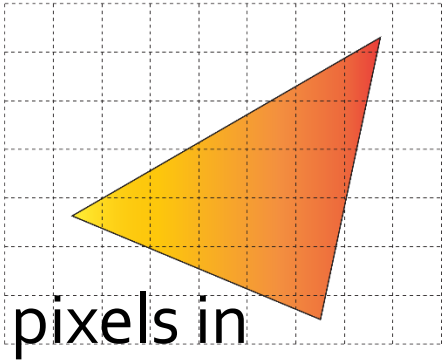
- Objects that are not within this volume are said to be clipped out of the scene







Rasterization



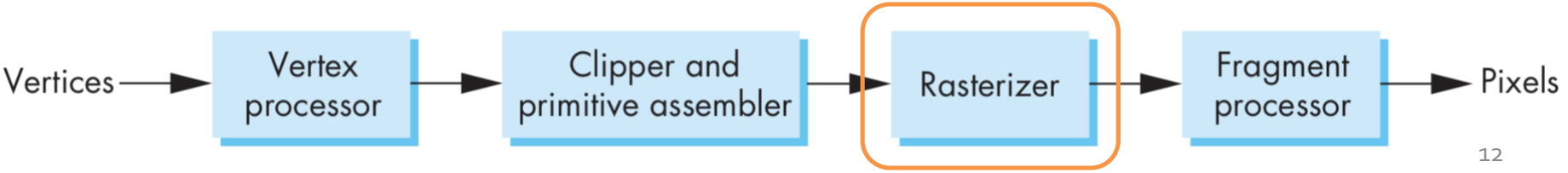
If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors

Vertex attributes are interpolated over objects by the rasterizer

Rasterizer produces a set of fragments for each object

Fragments are "potential pixels"

- Have a location in **frame buffer**
- Associated with color and depth attributes



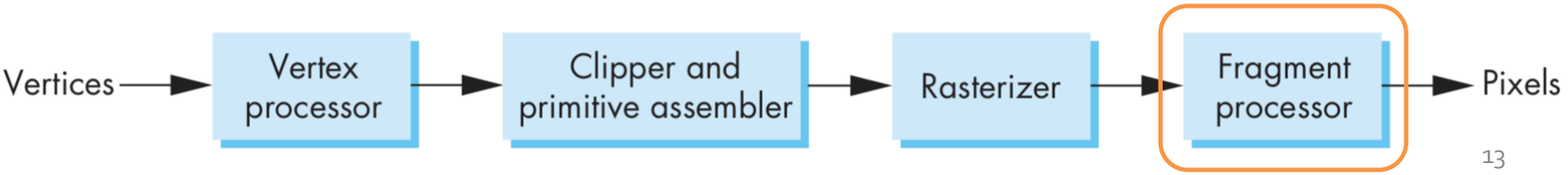
Fragment Processing

Fragments are processed to determine the final color of the corresponding pixel in the frame buffer

Colors can be determined by texture mapping or interpolation of vertex colors

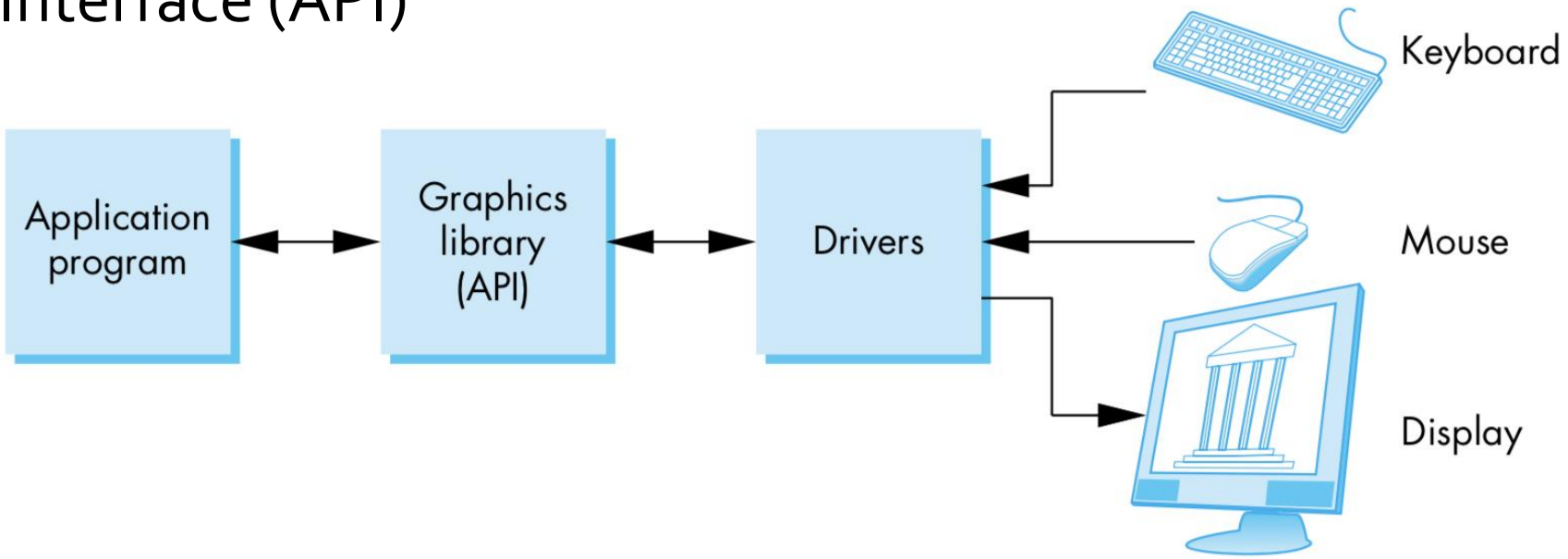
Fragments may be blocked by other fragments closer to the camera

- Hidden-surface removal
- Need another memory buffer (depth buffer)



The Programmer's Interface

Programmer sees the graphics system through a software interface: the Application Programmer Interface (API)



API Contents

Functions that specify what we need to form an image

- Objects
- Viewer
- Light Source(s)
- Materials

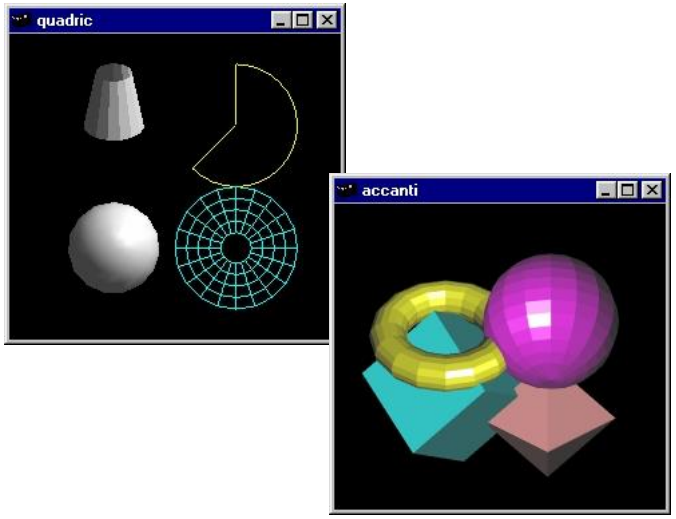
Other information

- Input from devices such as mouse and keyboard
- Capabilities of system

Object Specification

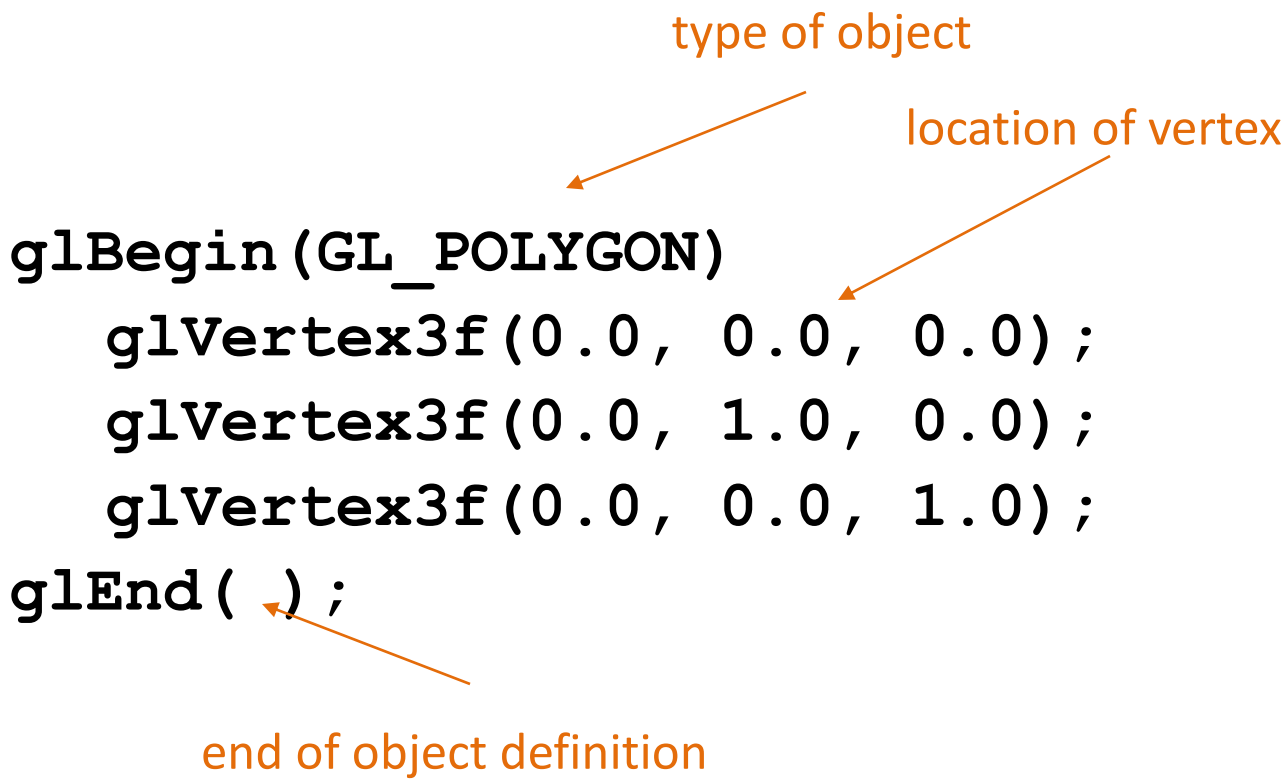
Most APIs support a limited set of primitives including

- Points (0D object)
- Line segments (1D objects)
- Polygons (2D objects)
- Some curves and surfaces
  - Quadrics
  - Parametric polynomials



All are defined through locations in space or vertices

Example (old style)



Example (GPU based)

Put geometric data in an array

```
vec3 points[3];
points[0] = vec3(0.0, 0.0, 0.0);
points[1] = vec3(0.0, 1.0, 0.0);
points[2] = vec3(0.0, 0.0, 1.0);
```

Send array to GPU

Tell GPU to render as triangle

Camera Specification

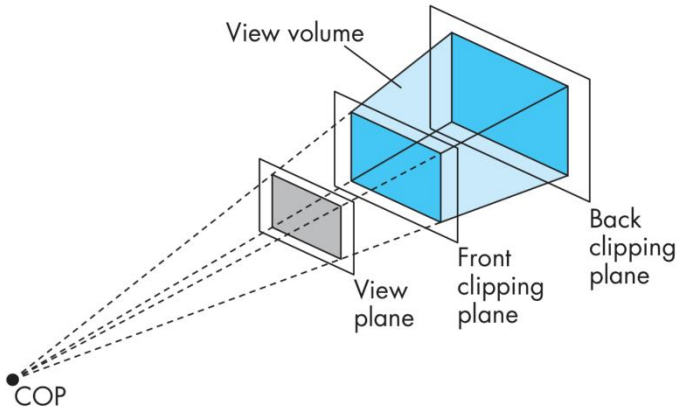
Six degrees of freedom

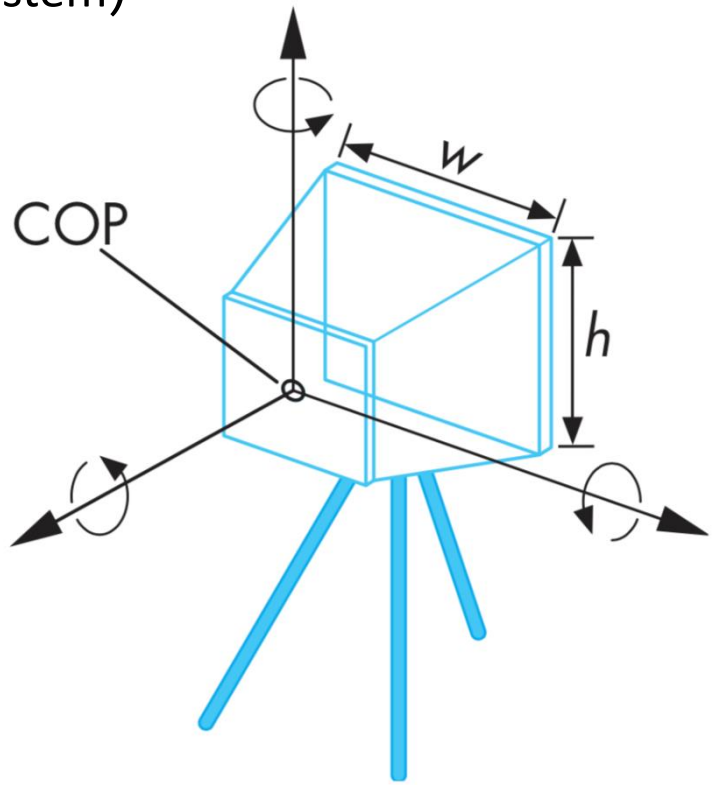
- Position of center of lens (COP: center of projection)
- Orientation (camera coordinates system)

Film size (viewport size)

Lens (where the view plane is)

Orientation of film plane





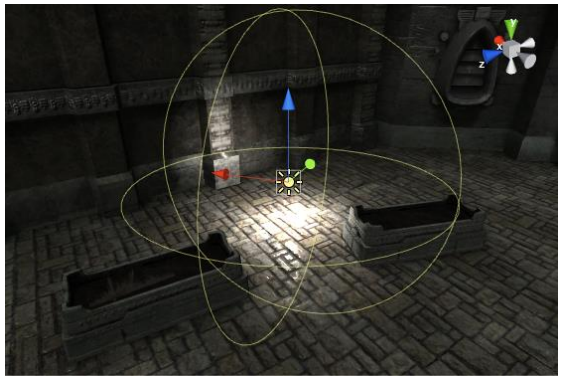
Lights and Materials

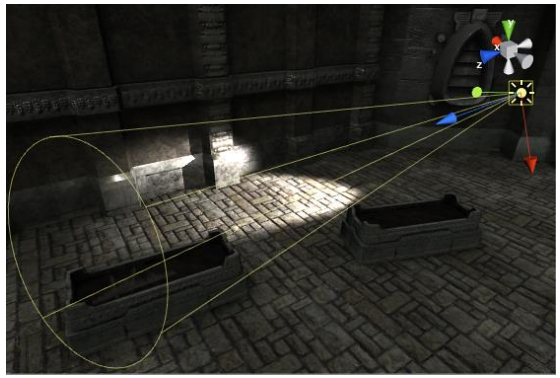
Types of lights

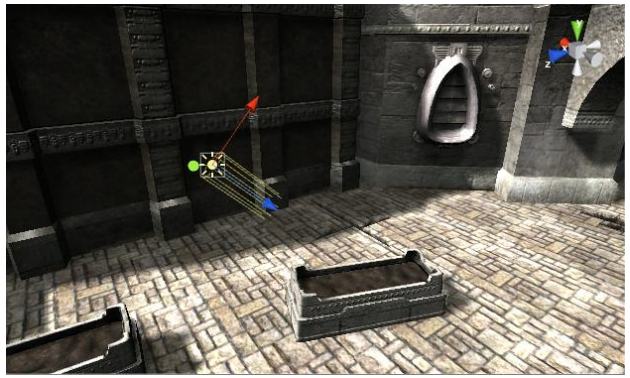
- Point sources vs. area sources
- Spot lights
- Near and far sources
- Color properties

Material properties

- Absorption: color properties
- Scattering
  - Diffuse
  - Specular







Non-programmable Pipeline

The graphics pipeline was implemented in CPU/GPU and the basic operations or functionalities were fixed (hardwired).

Advanced geometric computation or rendering effects (e.g., ray tracing, radiosity) had to be implemented in the application (algorithmic) level and thus it is difficult to achieve real-time graphics.

Programmable Pipeline

Pipeline architecture has now advanced so that the vertex processor and the fragment processor are now programmable by the application.

- Vertex shaders - alter location and color of vertices (e.g., bump mapping, new light-material models)
- Fragment shaders – can apply texture and lighting on a per-fragment basis