Graphics Pipeline

Lecturer: **Rachel McDonnell**

Assistant Professor in Creative Technologies

Rachel.McDonnell@cs.tcd.ie

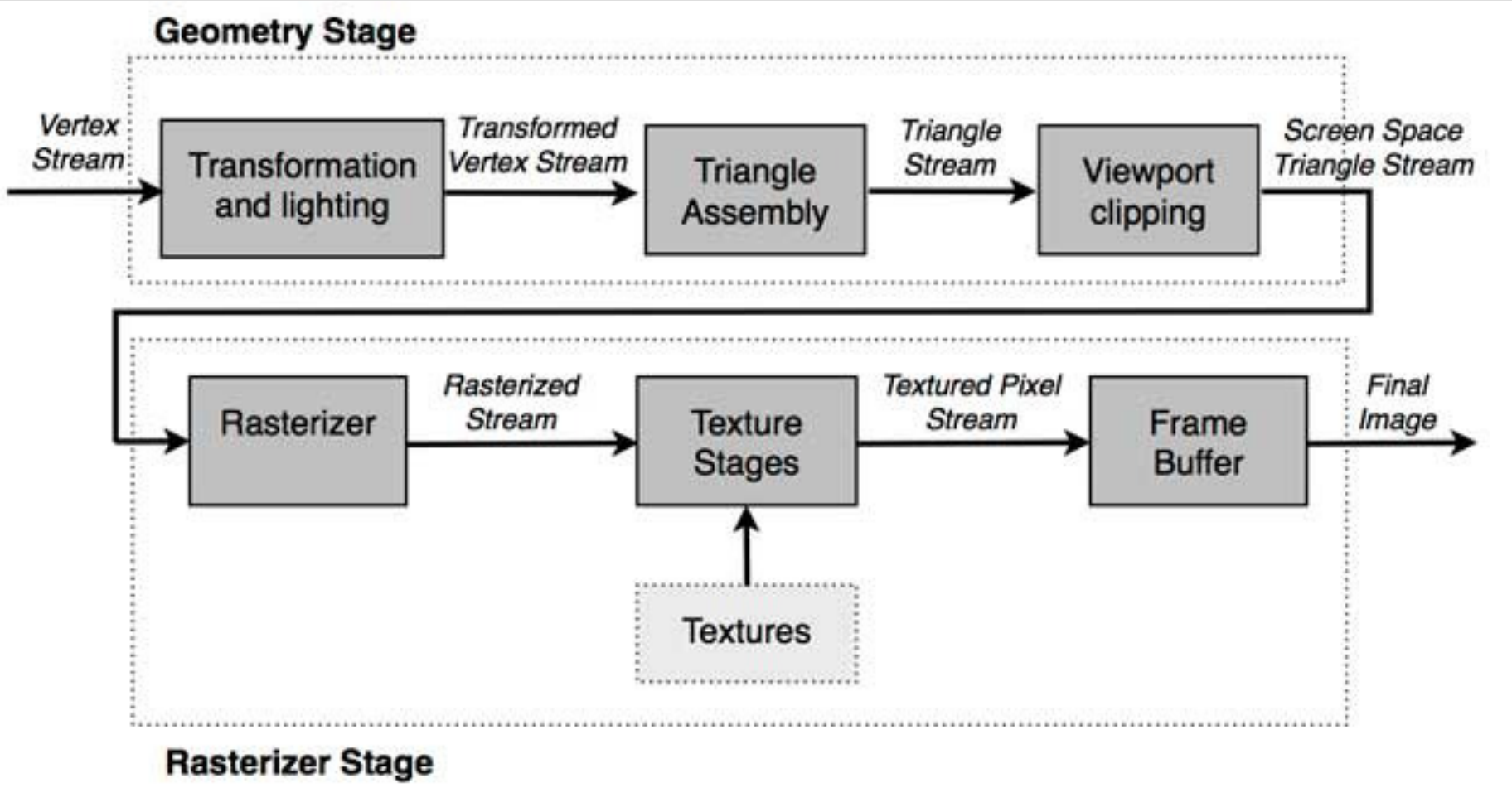
Course www: **https://www.scss.tcd.ie/Rachel.McDonnell/**

Credits: Real-time Rendering, 3rd Edition, Akenine-Moller

Overview

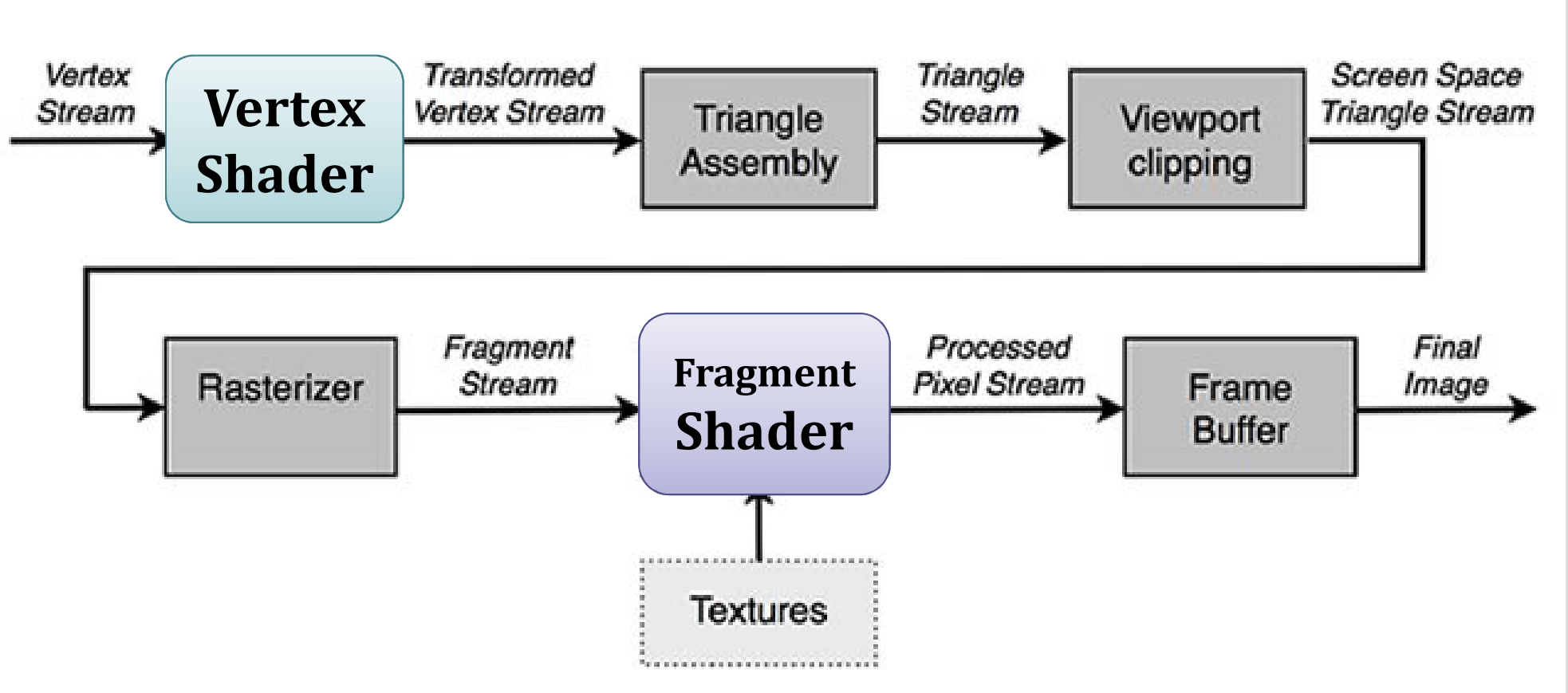
- Graphics Pipeline
- Graphics Programming
  - OpenGL background
  - OpenGL conventions,
  - GLUT Event loop, callback registration
  - OpenGL primitives, OpenGL objects
  - Shaders
  - Vertex Buffer Objects
  - Books, resources, recommended reading

Fixed Function Pipeline



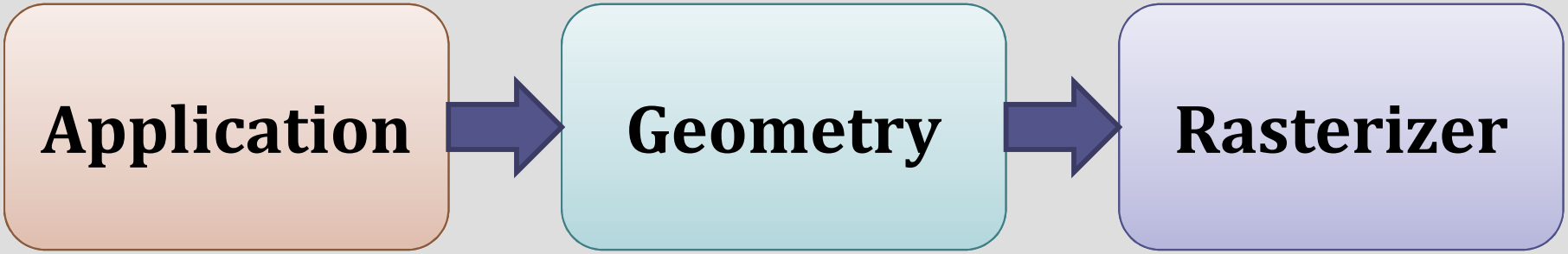
http://www.adobe.com/devnet/flashplayer/articles/how-stage3d-works.html

Graphics Programmable Pipeline



Graphics Pipeline Overview

- Coarse Division
- Each stage is a pipeline in itself



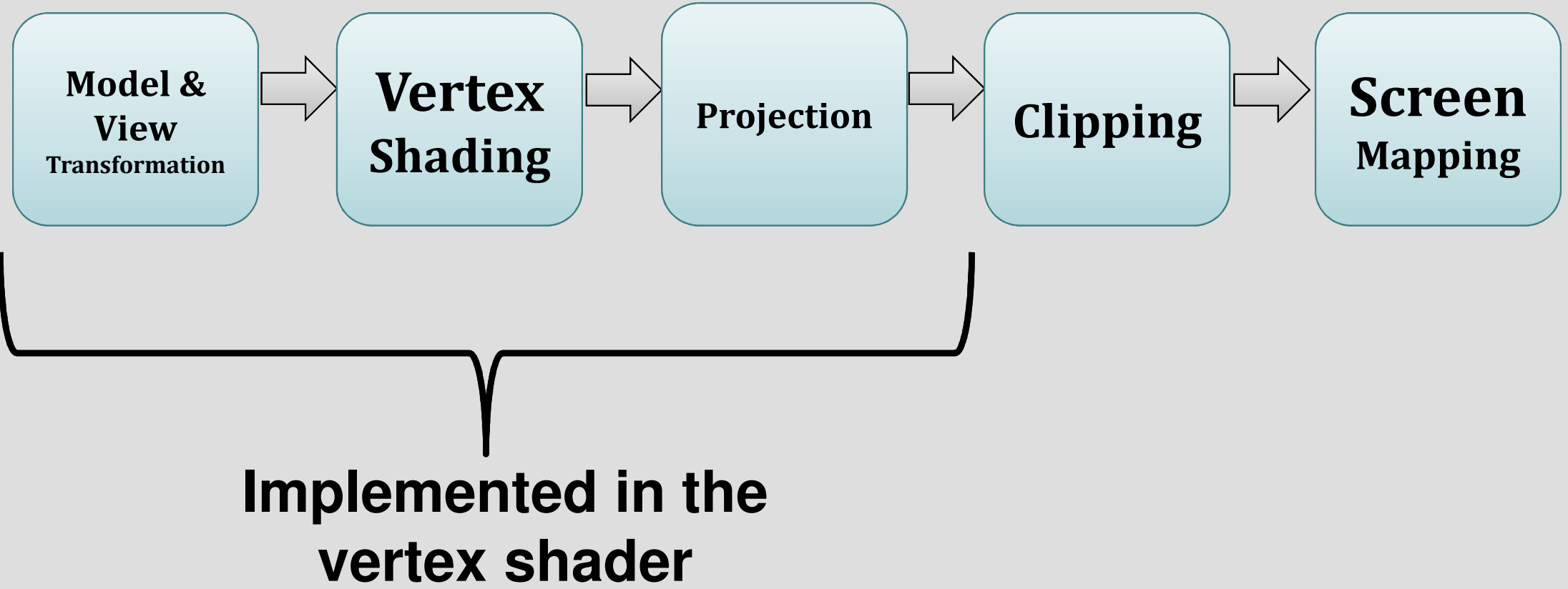
- The slowest pipeline stage determines the *rendering speed (fps)*

The Application Stage

- Developer has full control
- Executes on the CPU
- At the end of the application stage, the rendering primitives are fed to the geometry stage

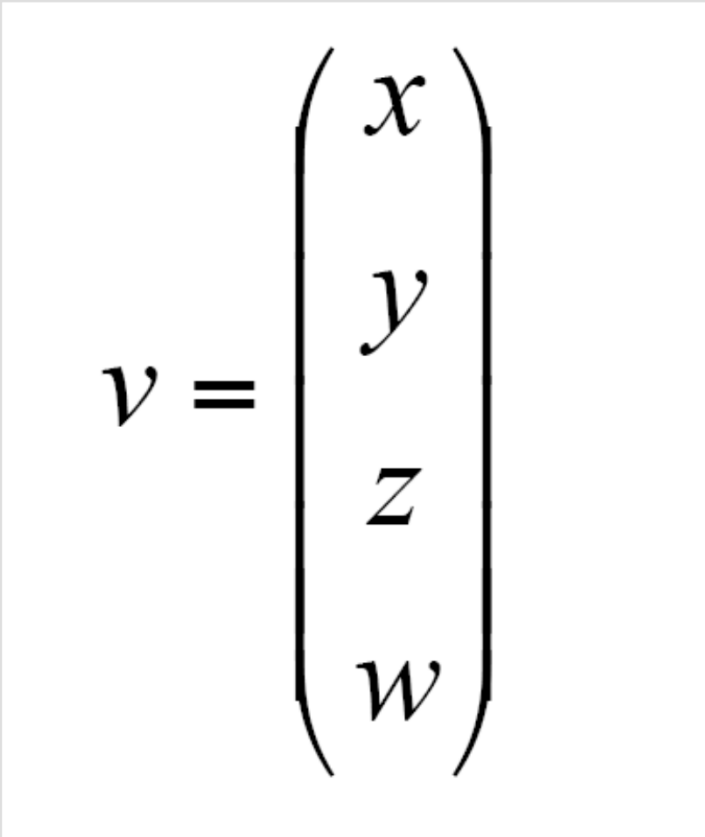
The Geometry Stage

- Responsible for the per-polygon and per-vertex operations



OpenGL Vertices

- OpenGL uses a 4 component vector to represent a vertex.
- Known as a homogenous coordinate system
- z = 0 in 2D space
- w = 1 usually



For further information on homogenous coordinate system, see Appendix G of the Red Book: http://fly.cc.fer.hr/~unreal/theredbook/appendixg.html

Model & View Transformation

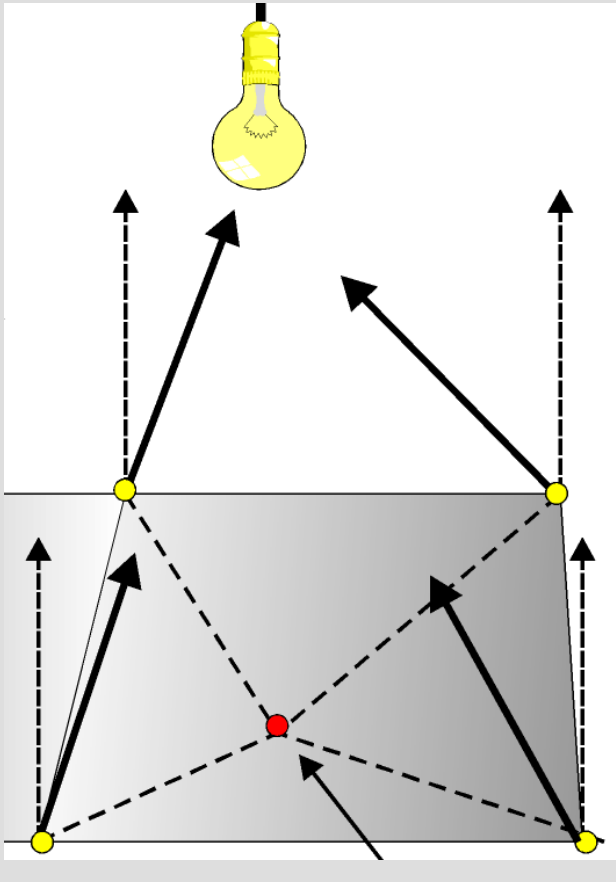
Model & View Transform

- Models are transformed into several *spaces* or *coordinate systems*
- Models initially reside in *model space*
  - i.e. no transformation
- "*Model transform*" positions the object in *world coordinates* or *world space*
- The *view transform*

Vertex Shading

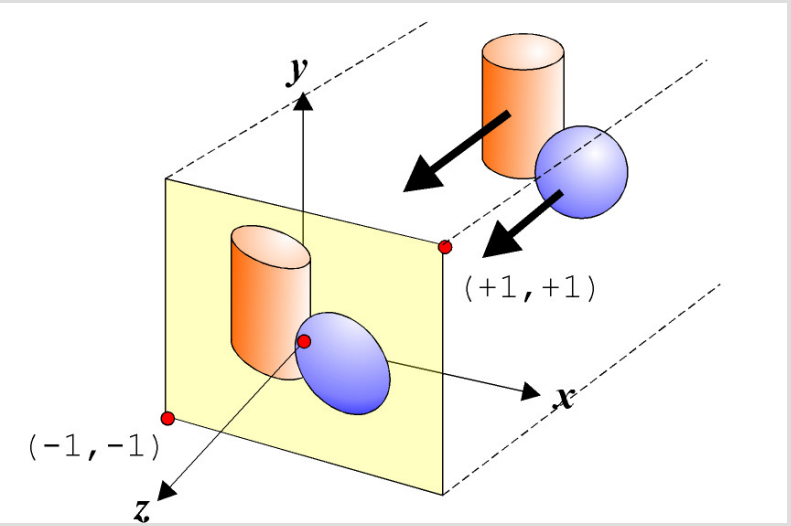
Vertex Shading

- Shading means determining the effect of a light on a material
- A variety of material data can be stored at each vertex
  - Points location
  - Normal
  - Color
- Vertex shading results (colors, vectors, texture coordinates, or any other kind of shading data) are then send to the rasterization stage to be interpolated



Projection

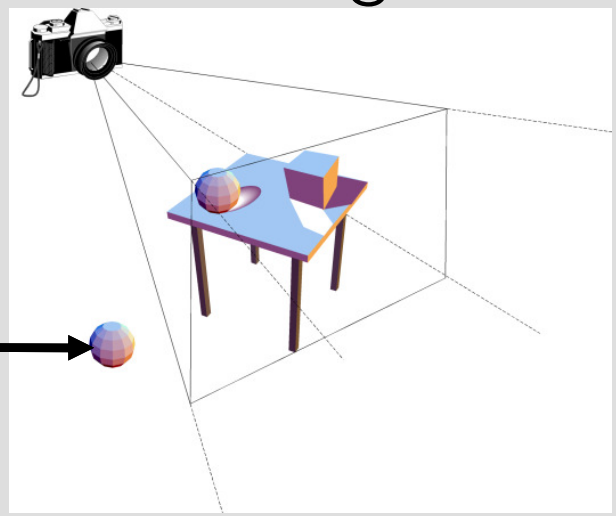
- After shading, rendering systems perform *projection*
- Models are projected from three to two dimensions
- *Perspective* or *orthographic* viewing



Clipping

Clipping

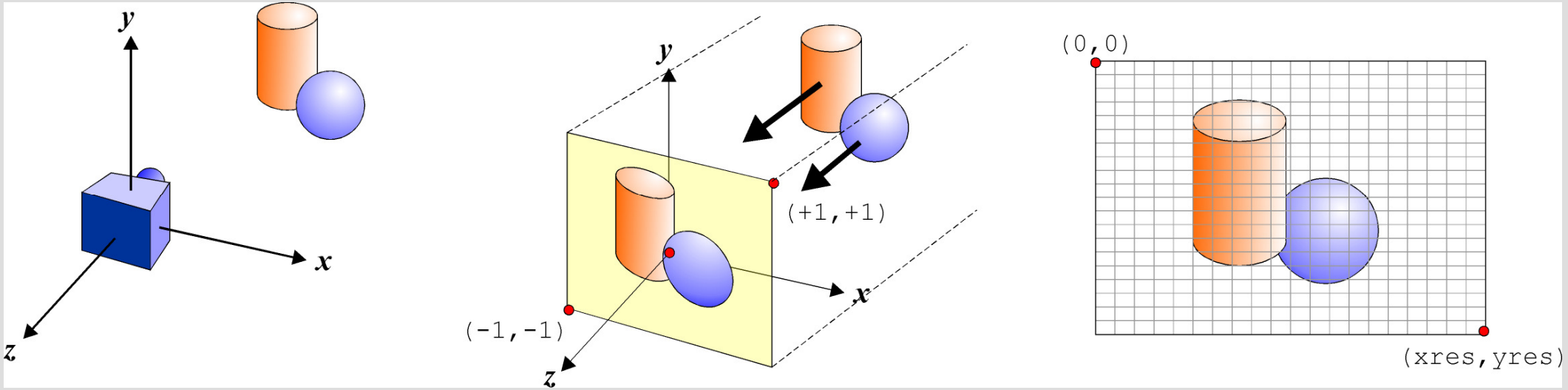
- The computer may have model, texture, and shader data for all objects in the scene in memory
- The virtual camera viewing the scene only "sees" the objects within the field of view
- The computer does not need to transform, texture, and shade the objects that are behind or on the sides of the camera
- A clipping algorithm skips these objects making rendering more efficient



Screen Mapping

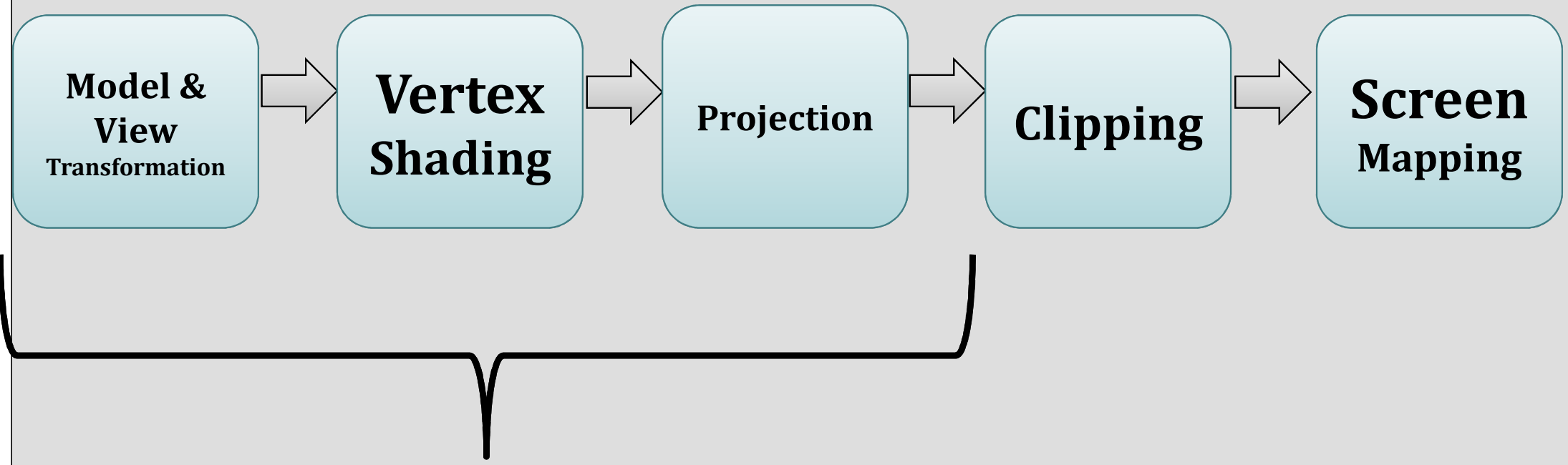
Screen Mapping

- Only the clipped primitives inside the view volume are passed to this stage
- Coordinates are in 3D
- The x- and y-coordinates of each primitive are transformed to for *screen coordinates*



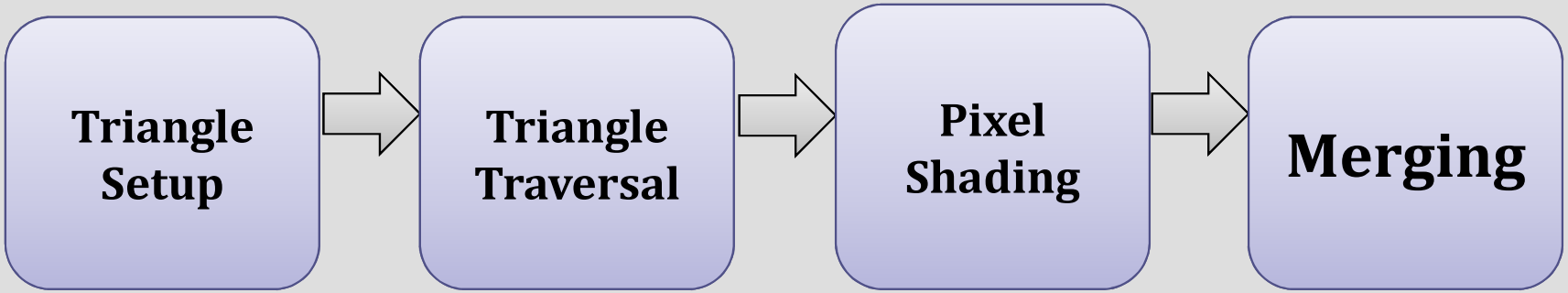
The Geometry Stage

- Responsible for the per-polygon and per-vertex operations



The Rasterizer Stage

- Given the transformed and projected vertices with their associated shading data (from geometry stage)



- The goal of the rasterizer stage is to compute and set colors for the pixels covered by the object
- *Rasterization:* conversion from 3D vertices in screen-space to pixels on the screen

Triangle Setup

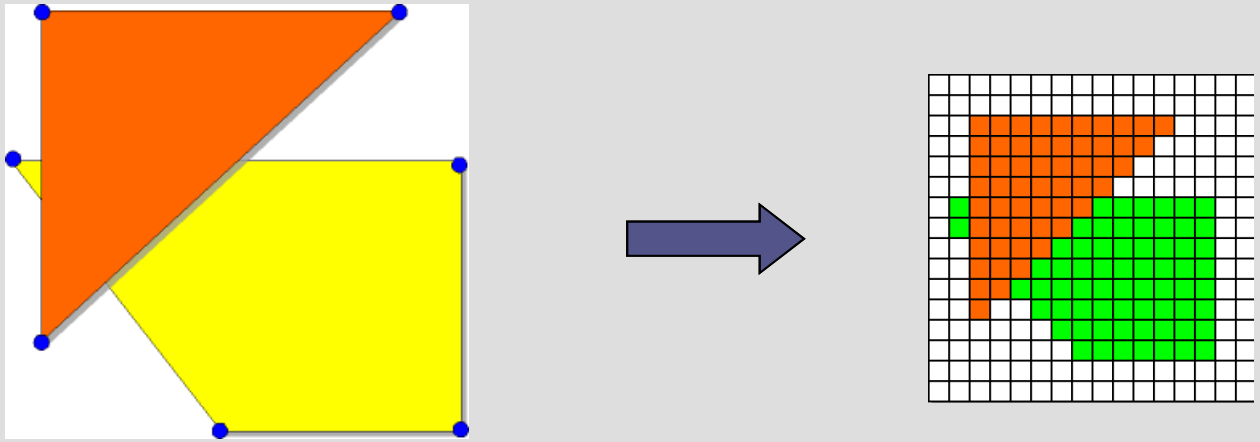
Triangle Setup

- Vertices are collected and converted into triangles.
- Information is generated that will allow later stages to accurately generate the attributes of every pixel associated with the triangle.

Triangle Traversal

Triangle Traversal

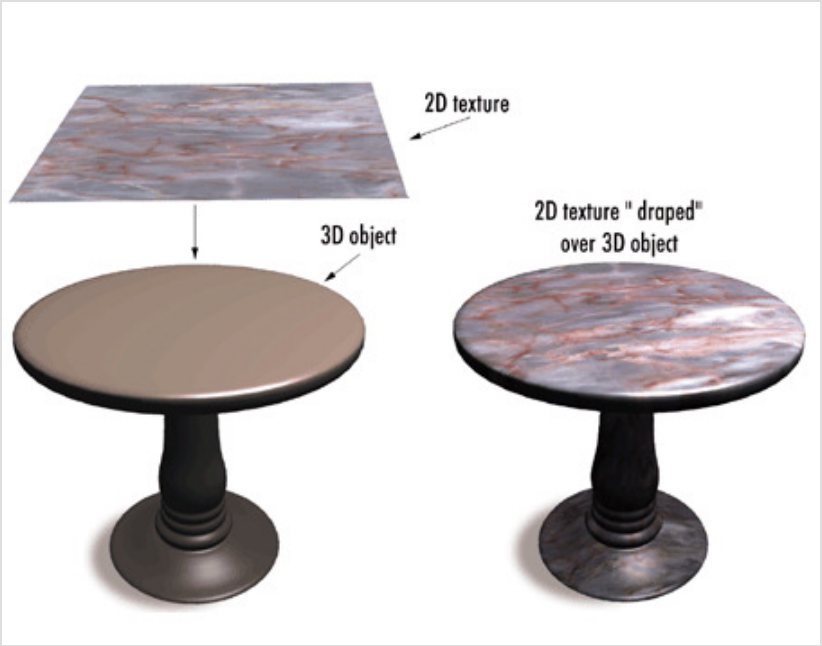
- Which pixels are inside a triangle?
- Each pixel that has its centre covered by the triangle is checked
- A *fragment* is generated for the part of the pixel that overlaps the triangle
- Triangle vertices interpolation



Pixel Shading

Pixel Shading

- Per-pixel shading computations are performed here
- End result is one or more colours to be passed to the next stage
- Executed by programmable GPU cores
- NB: Texturing is employed here



Merging

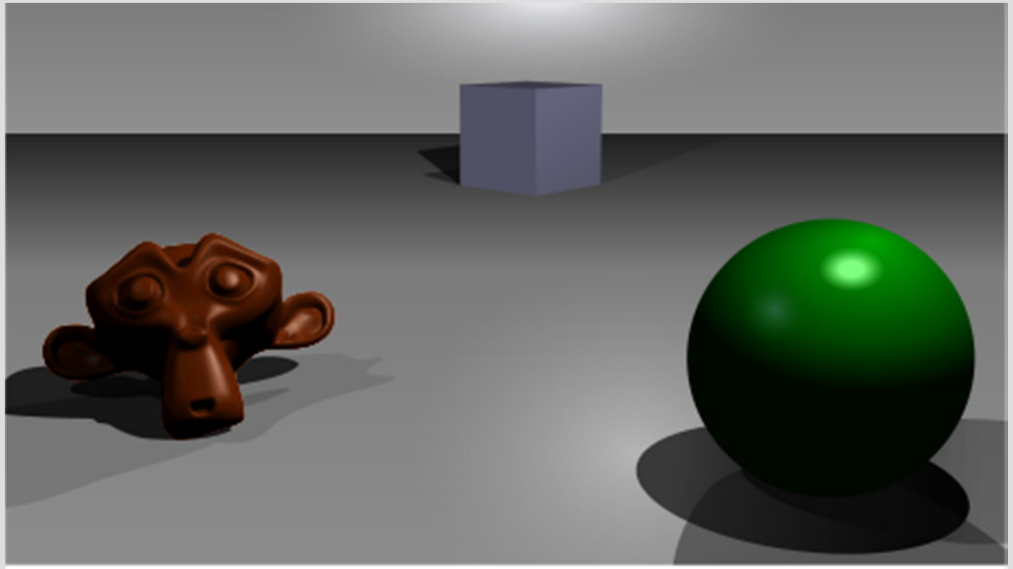
Merging

- Information for each pixel is stored in the *colour buffer* (a rectangular array of colours)
- Combine the fragment colour produced by the shading stage with the colour currently stored in the buffer

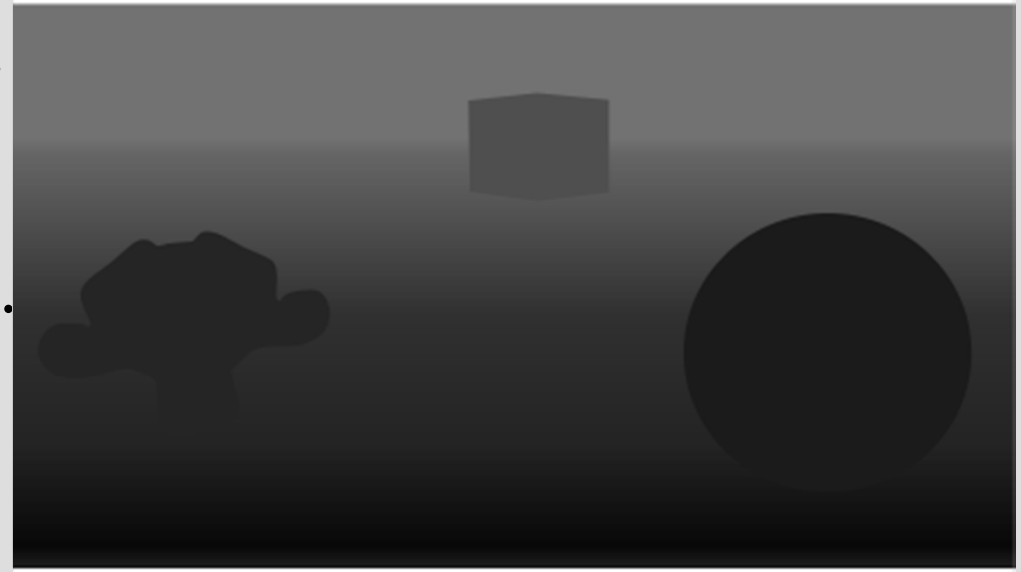
- This stage is also responsible for resolving visibility
  - Using the z-buffer

Z-Buffer

- Arranged as a 2D array with one element for each screen pixel.
- Stores the z-value from the camera to the currently closest primitive
- If another object of the scene must be rendered in the same pixel, the method compares the two depths and chooses the one closer to the observer.
- The chosen depth is then saved to the z-buffer, replacing the old one.



A simple three-dimensional scene



Z-buffer representation

Double Buffering

- Passed the rasterizer stage, those primitives that are visible from the point of view of the camera are displayed on screen
- The screen displays the contents of the color buffer
- To avoid perception of primitives being rasterized, *double buffering* is used
- Rendering takes place off screen in a *back buffer*
- Once complete, contents are swapped with the *front buffer*