

---

# CM0304 Graphics

---

## I Graphics Hardware

### I.4 Graphics Processors

Xianfang Sun

F. C. Langbein

School of Computer Science  
Cardiff University



Version 2.3

## Overview

- Display processors
  - Pipeline processor architecture
  - Display processor pipeline and performance
- 3D graphics pipeline
- OpenGL architecture

Xianfang Sun F.C. Langbein, CM0304 Graphics – I Graphics Hardware; I.4 Graphics Processors

1

## Display Processor

- Task of the *display processor*:

*Relief the host (CPU) from expensive graphics computations using specialised hardware*

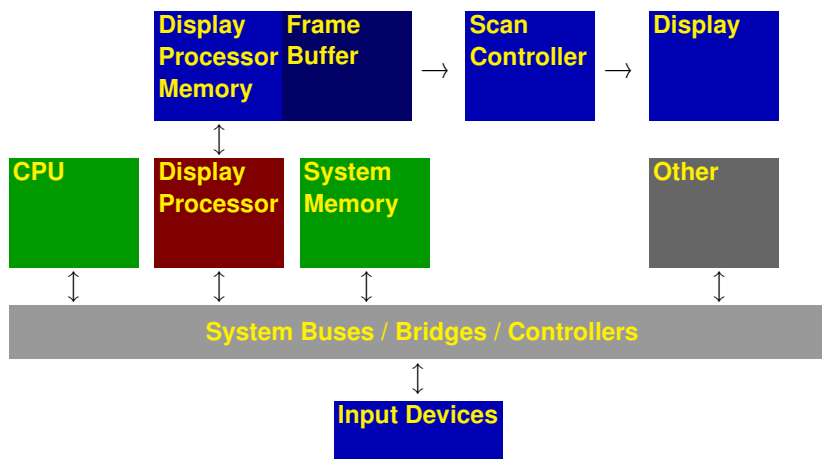
- Initial versions of display processors:
  - Host computes instructions to create image: *display lists*
  - Display processor *executes display lists* in local memory repetitively to refresh image  
(see vector displays)
- Modern display processors: *pipeline architecture*

Xianfang Sun F.C. Langbein, CM0304 Graphics – I Graphics Hardware; I.4 Graphics Processors

2

## Display Processor Architecture

➤ Raster-graphics system with display processor



## Processor Tasks

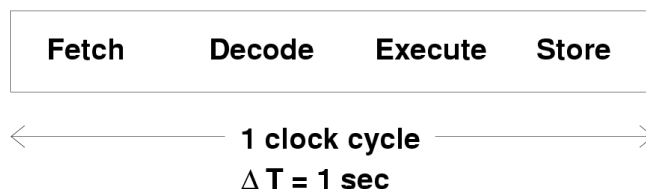
➤ Simplified standard processor has the following tasks per instruction:

- **Fetching**: locate and retrieve the data from memory
- **Decoding**: Interpret or translate the instruction from the software
- **Executing**: Perform the given instruction on the given data
- **Storing**: Place the result of the data back into a memory location

## Unpipelined Processor

➤ A single, unpipelined clock-cycle

– with 1 instruction per clock-cycle at 1Hz



➤ Instructions in 10 seconds:

$$10\text{sec} * \left(\frac{1\text{cycle}}{1\text{sec}}\right) * \left(\frac{1\text{instruction}}{1\text{cycle}}\right) = 10\text{instructions}$$

## Increasing Processor Speed

- To make a processor faster we can
  - **Shrink** the die-size to allow transistors to switch faster
  - And/or give the processor **less to do** per clock-cycle
- 4Hz instead of 1Hz with less work per cycle



← 1 clock cycle →

← Δ T = 1 sec →

- But this does not make the processor faster overall

$$10\text{sec} * \left(\frac{4\text{cycle}}{1\text{sec}}\right) * \left(\frac{1/4\text{instruction}}{1\text{cycle}}\right) = 10\text{instructions}$$

## Pipelining

- Our processor can fetch, decode, execute and store **simultaneously**

Cycle	Fetch Unit	Decode Unit	Execute Unit	Store Unit
1	Instr. 1			
2	Instr. 2	Instr. 1		
3	Instr. 3	Instr. 2	Instr. 1	
4	Instr. 4	Instr. 3	Instr. 2	Instr. 1

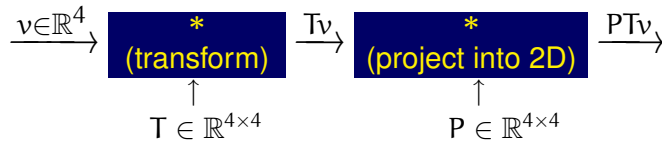
- In 10 seconds at 4Hz
  - 40 clock-cycles
  - 37 instructions  
(4 cycles for instruction 1, then 36 remaining instructions)

## Branching

- Add more processing units along the pipeline for further speed increase!?
- **Branching** is a problem
  - Conditional statements create different paths through code
  - Partial solution: **branch prediction**
  - But branch **mispredictions** possible
  - In case of a misprediction the pipeline has to be **flushed** (discard all data and start over)
  - The longer the pipeline the more is lost (the higher the slow down)

## Display Processor Pipeline

- Display processors consist of two sub-systems
  - **Font-end** sub-system to handle geometry (e.g. pipeline on a stream of primitives)
  - **Back-end** sub-system to handle rasterisation (e.g. parallel processing on raster)
  - Pipelining and/or parallel processing used for both
- Special processing unit for individual graphics operations



(vertices given by 4 numbers define geometry and are modified by linear transformations / matrices, this will become clearer later)

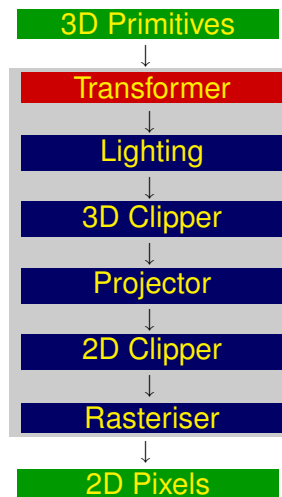
## Performance Issues

- Pipeline performance determined by slowest component
  - **Throughput**: overall rate of data processing (how many primitives pass the pipeline in given time)
  - **Latency**: time required for a single datum to pass the pipeline
  - Must **balance** latency against throughput: the longer the pipeline the higher the throughput, but also the larger the latency
- *Branching is no problem*
  - Flush not required during normal operation
  - Primitives send through the pipeline are either displayed or filtered independently

## Graphics Pipeline Tasks

- Input of graphics pipeline provided by host / user code:
  - 3D models (e.g. triangular meshes)
    - Transformations applied to models (e.g. rotations)
    - Material properties (e.g. colour)
  - Light sources
  - Camera
- Output of graphics pipeline: 2D pixels in a raster
- What operations does the pipeline have to execute?
  - Models (*vertices*) are transformed into pixels by pipeline
  - The rest describes this transformation in terms of *attributes*

## 3D Graphics Pipeline



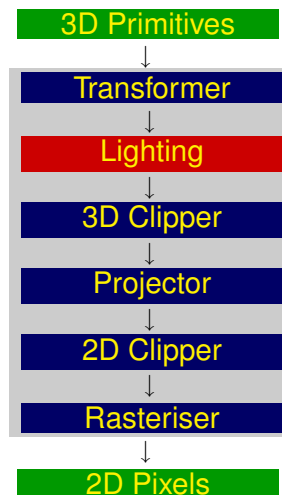
### ➤ Modelling transformations:

- Convert *model coordinates* into *world coordinates*

### ➤ Viewing transformations:

- Convert *world coordinates* into *camera coordinates*
- Multiple transformation matrices can be combined to single matrix
- Parallel realisation allows to multiply vector with matrix in one operation

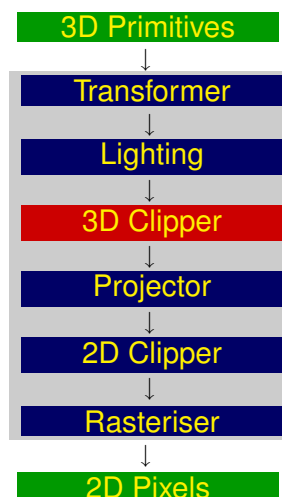
## 3D Graphics Pipeline



### ➤ Evaluate illumination model

- To determine colour/shades of primitives
  - E.g. compute colour value for polygon vertices based on material properties, light sources, etc.
  - *Shading* of whole primitive is done during rasterisation based on these values

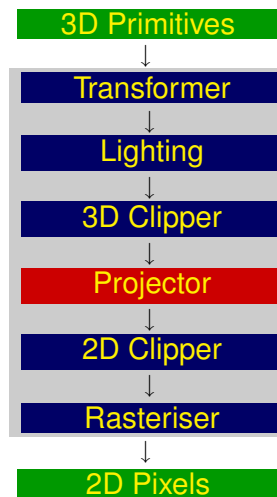
## 3D Graphics Pipeline



### ➤ Clipping selects visible part of the whole scene for displaying

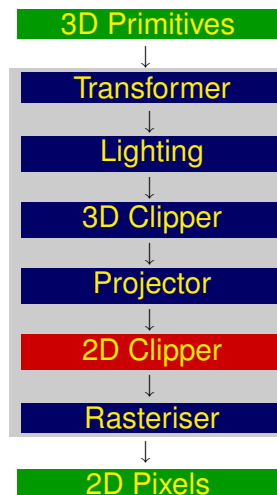
- 3D clipping selects primitives inside viewing volume (cut off objects at planes)
  - For *perspective* projection: frustum (cut-off pyramid)
  - For *parallel* projection: rectangular parallelepiped
- May also remove hidden surfaces

## 3D Graphics Pipeline



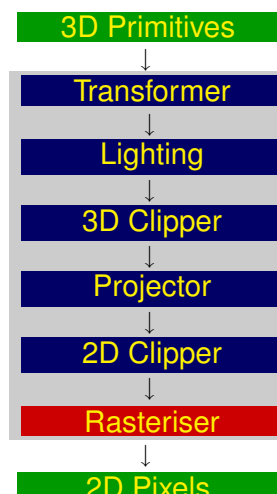
- **Project** 3D primitives onto plane to give 2D shapes
  - Projection matrix specifies type of projection
  - Camera properties specify projection in detail

## 3D Graphics Pipeline



- **2D clipping:**
  - Cut off (partially or completely) 2D shapes outside a rectangular area
  - Apply viewport transformation (project rectangular area onto raster)
  - May also remove hidden surfaces

## 3D Graphics Pipeline



- Convert 2D shapes into pixels
  - **Scan conversion:** draw 2D polygons, lines, points in frame buffer
  - May include shading of lines and polygons

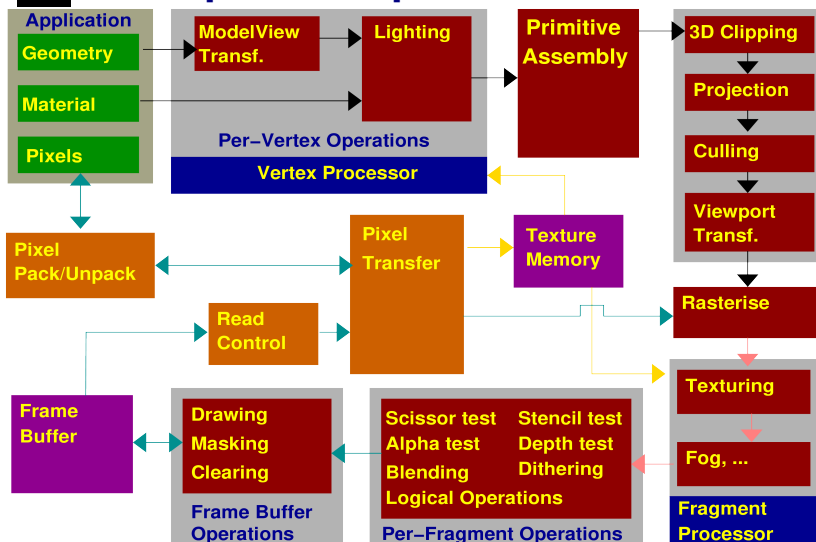
# OpenGL

- **OpenGL**: Open Graphics Library
  - Originally IRIS GL from Silicon Graphics
- OpenGL is
  - A software interface to graphics hardware
  - A graphics programming library
  - A standard for 3D graphics
- At the lowest possible level that still allows device independence
  - OpenGL is partly implemented in software and partly in hardware depending on the device
  - No high-level modelling operations, etc.

# OpenGL State Machine

- OpenGL provides an **imperative** interface to a **state machine**
  - Graphics primitives pass the state machine in a fixed graphics pipeline (but OpenGL Shading Language allows to program processors in the pipeline)
  - Most functions change the state of the machine (e.g. turn processing units on or off, define material, shape types, light sources, ...)
  - States are **persistent**, e.g. if we set the material it remains the same until we change it again
  - Attributes of primitives (vertices) are states of the machine

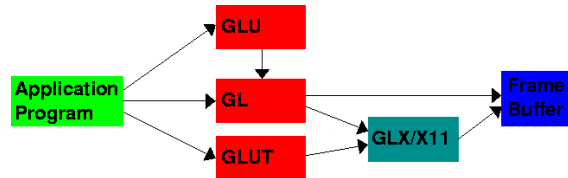
# The OpenGL Pipeline



## OpenGL Components

### ► Components of the OpenGL interface:

- **GL**: core OpenGL functions
- **GLU**: graphics utility library  
(helpers for creating common objects, e.g. spheres, the teapot)
- **GLUT**: GL Utility Toolkit  
(interface to windowing system via xlib; alternatives: glib+GTK, QT)
- **GLX**: low-level interface to X11  
(different interfaces for other platforms: glw for windows)



## Summary

- ### ► What is a pipeline processor architecture?
- How does it increase processing speed?
  - What are the core performance issues?
- ### ► What is the task of a display processor?
- What determines the performance of a display processor?
- ### ► What are the major components of a graphics pipeline and how do they interact?
- ### ► What is the underlying model for the OpenGL library?
- What are the components of OpenGL?