Introduction

Computer Graphics

March 8, 2016

CS380
Introduction to Computer Graphics

3:3:4

Objective

- To provide a broad introduction to the field of Computer Graphics
  - Interactive 2D and 3D graphics display concepts
  - Programming-oriented / top-down approach

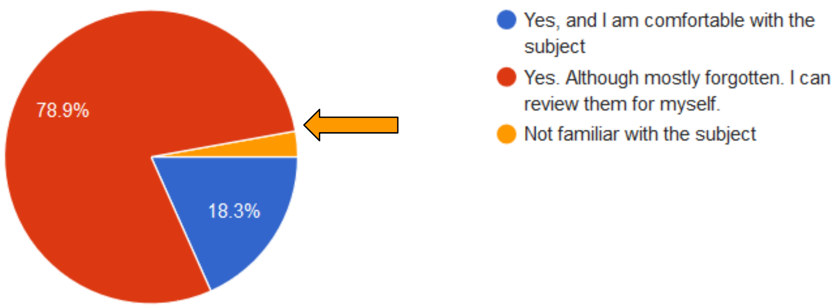
- To gain the fundamental technical backgrounds for Computer Graphics related research fields
- Practice to become a creative thinker and *have fun learning!*

Survey Quiz

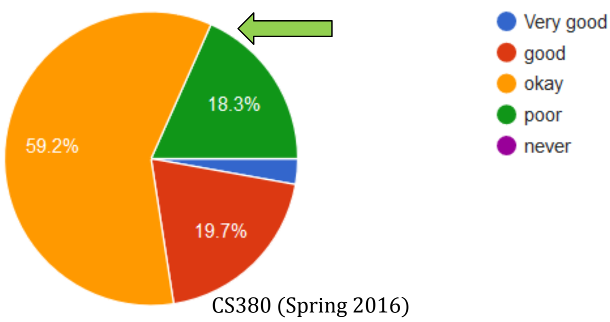
- Linear Algebra
  - Matrix multiplication
  - Vector inner/outer product
  - Eigenvalue and eigenvectors
- Programming in C/C++
  - Allocating memory / pointer / array
  - Simple code understanding

- Notebook computer (MS Window)
  - If you do have one, please bring it to the lab session tomorrow!!

Have you taken 'Introduction to Linear Algebra' or a similar course before?
(71 responses)



How is your experience in C/C++? (71 responses)



Homework #0

- **Due by today midnight!**
- If you are having a difficult time completing, contact a TA for help.
  - Until 8 PM, will be in the office at E3-1.
    - Or send email to set up an appointment.
  - Consider taking this course *next year*, when you become more familiar with programming environment.

LAB session

- Wednesday (tomorrow) 7~10 PM
- E11 (Creative Learning Bldg) #307 (#306)
- Bring your notebook if you have one.

Computer Graphics

- What do you think it refers to?

Computer Graphics

- Concerns with all aspects of producing pictures or images using a computer

- Applications of Computer Graphics
  - Display of information
  - Design
  - Simulation and animation
  - User interfaces

Computer Graphics History

1950s …

… 2007 …

Computer Graphics History

- 2007
- 2008

Feature Animation Films

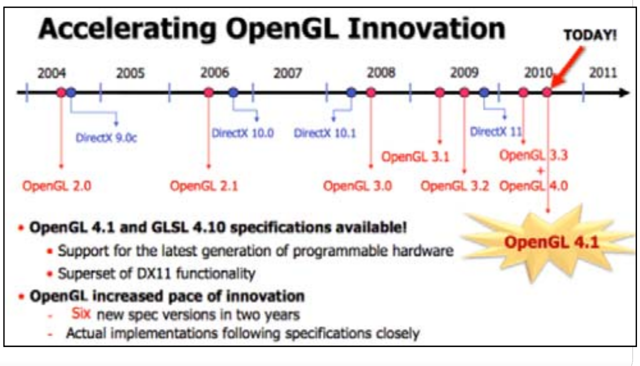
Computer Graphics History

- 2007
- 2008
- 2009
- 2010
- 2011
- 2012
- 2013

Feature Animation Films

Display Hardware

3D Interaction Devices



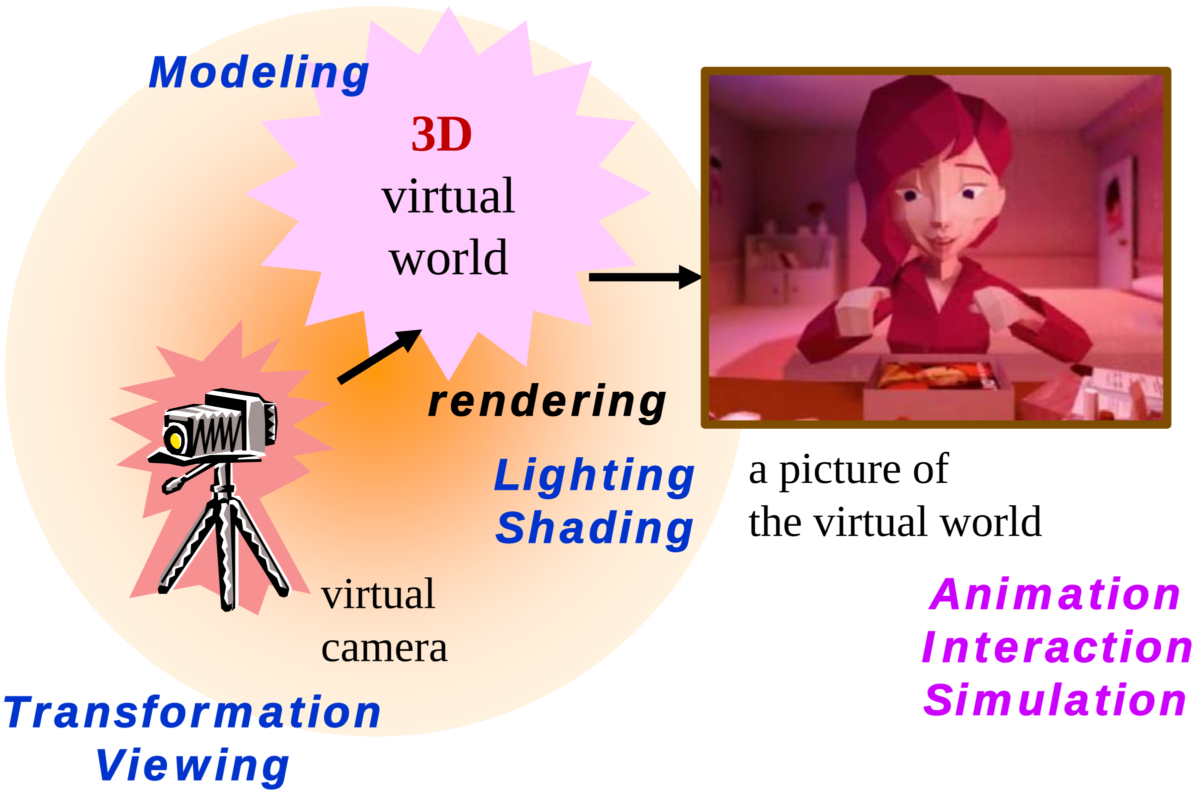
http://news.cnet.com/8301-30685_3-20011645-264.html

Computer Graphics History

- 1992 .... OpenGL 1.0
- :
- 2004 .... OpenGL 2.0
- :
- 2007 Feature
- 2008 Animation .... OpenGL 3.0
- 2009 Films
- 2010 .... Display Hardware .... OpenGL 4.0
- 2011 OpenGL ES 2.0 → WebGL
- 2012 .... 3D Interaction Devices .... OpenGL 4.3
- 2013 OpenGL ES 3.0
- :
- 2016 .... OpenGL 4.5

High performance GPU

3D Computer Graphics



3D Computer Graphics

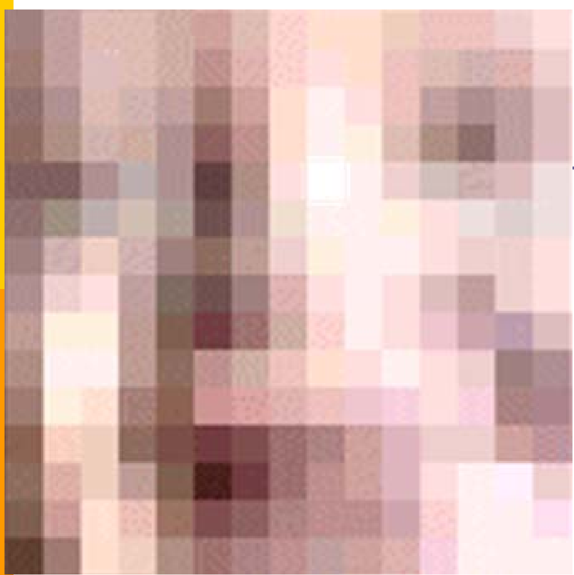
- Algorithm for visual simulation

- Main Theme
  - Imaging
    - **Representing 2D images**
  - Modeling
    - **Representing 3D objects**
  - Rendering
    - **Constructing 2D images from 3D models**
  - Animation
    - **Simulating changes over time**

Amazing thing about human brain …

Originally from http://electronics.howstuffworks.com

- A home video showing a happy baby playing with a toy encoded an MPEG file so that you can view it on your computer



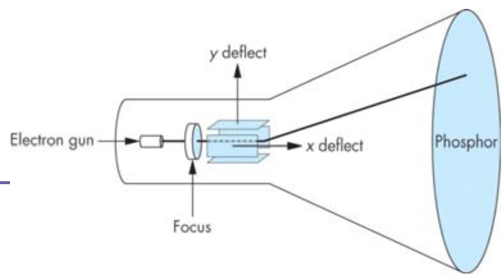
If you divide a still image into a collection of small colored dots, your brain will reassemble the dots into a meaningful image

If you divide a moving scene into a sequence of still pictures and show the still images in rapid succession, the brain will reassemble the still images into a single moving scene.

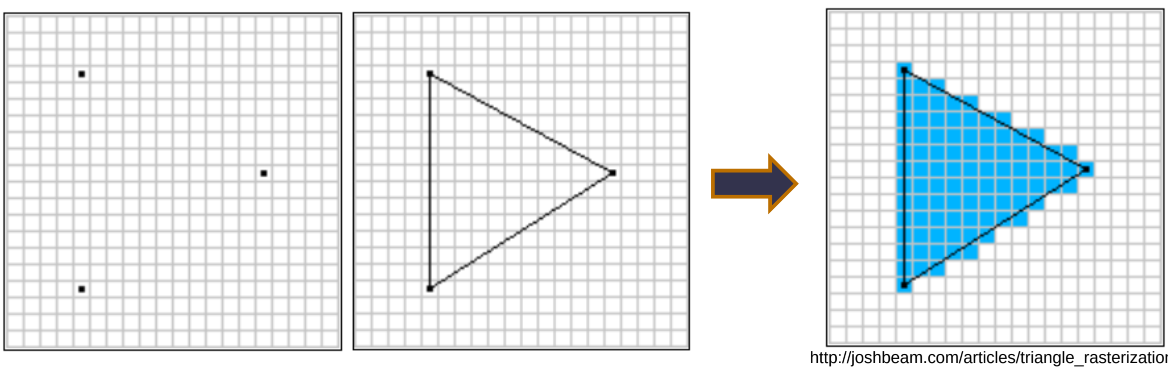
Pixels and the Frame Buffer

- Raster-based graphics systems
  - A picture is produced as an array – the raster – of picture elements within the graphics system.
- Collectively, the pixels are stored in a part of memory – **frame buffer**.
  - Depth of the frame buffer = number of bits that are used for each pixel
    - 1 bit : 2 colors,
    - 8 bits : 256 colors,
    - 24 bits: true color
  - Resolution – the number of pixels in the frame buffer
- **Rasterization**, scan conversion: converting of geometric entities to pixel assignments in the frame buffer

Raterization

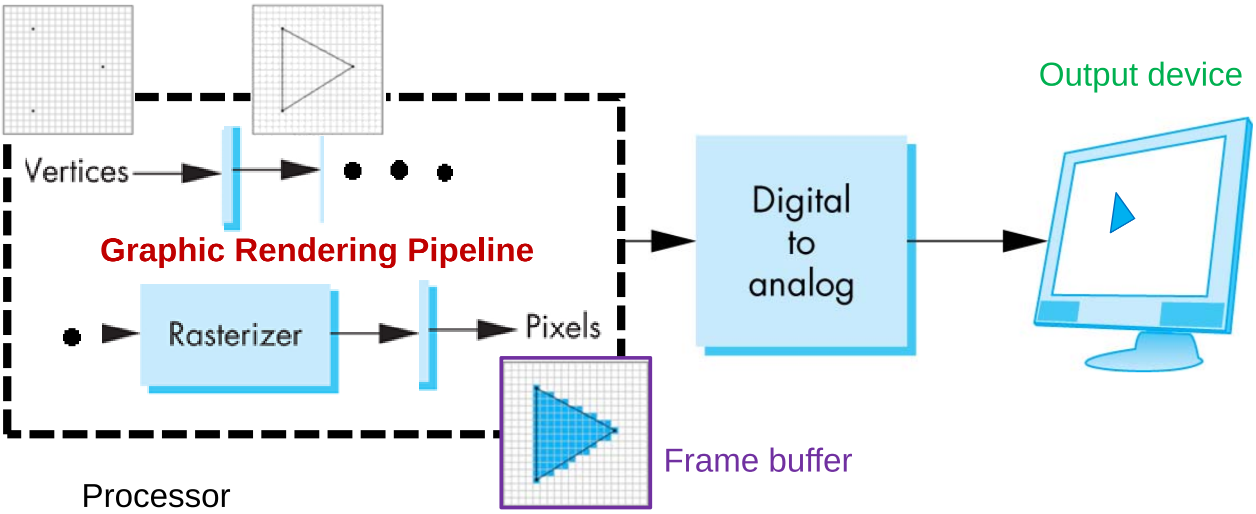


- Raster: (n) the rectangular formation of parallel scanning lines that guide the electron beam on a television screen or a computer monitor
- Rasterization:
  the task of taking an image described in shapes and converting it into a raster image (pixels or dots) for output on a video display or printer



http://joshbeam.com/articles/triangle_rasterization/

Graphics System



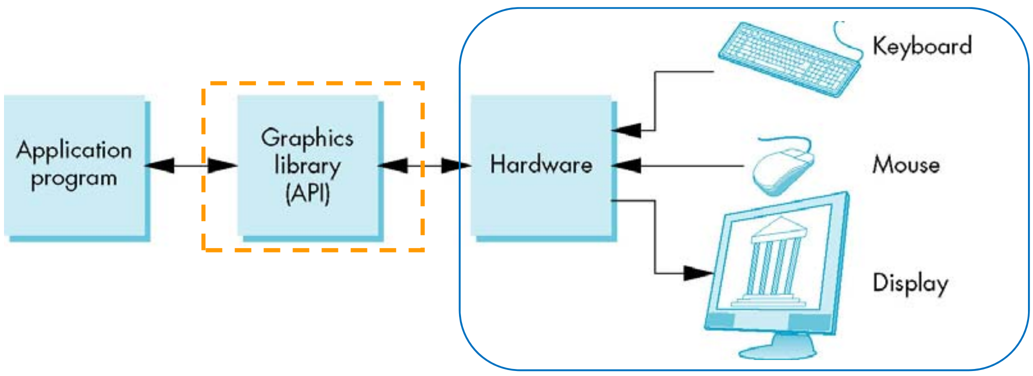
Processor

Memory

Input devices

Graphics Architecture

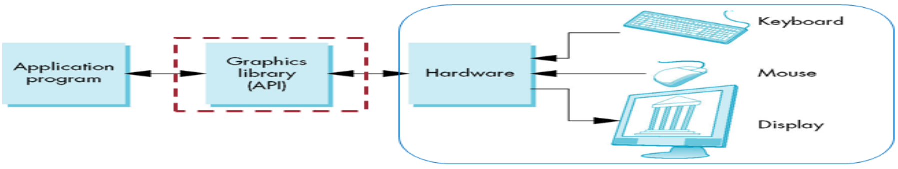
- How to interact with a graphics system?
  - Application programmer's model of graphics system



  - The interface between an application program and a graphics system can be specified through a set of functions that resides in a graphics library.
    - API: Application Programmer's Interface

Graphics Architecture

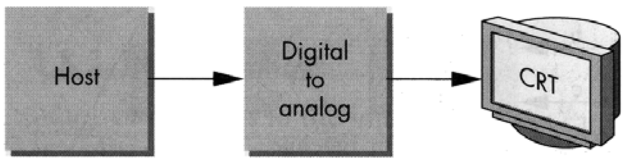
- On one side of the API is the application program. On the other is some combination of hardware and software that implements the functionality of the API.



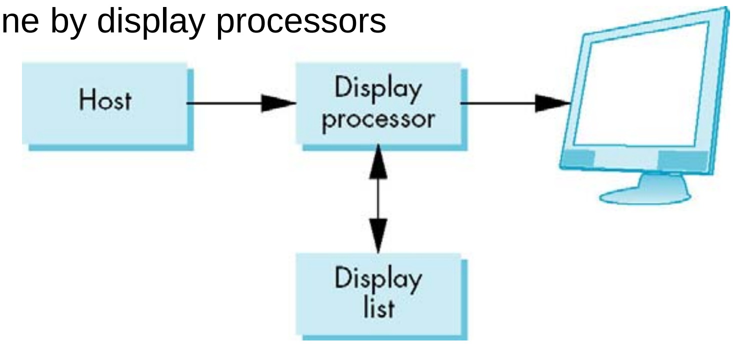
- Researchers have taken various approaches to developing architectures to support graphics APIs.
  - Early graphics system
    - Based on a CRT display
  - Display Processors
  - Pipeline Architectures
  - Programmable Pipeline

Graphics Architectures

- Early graphics system
  - All operations of the pipeline is done by the host



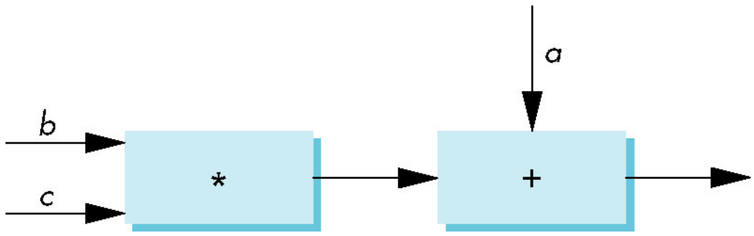
- Display Processors
  - Scan conversion is done by display processors



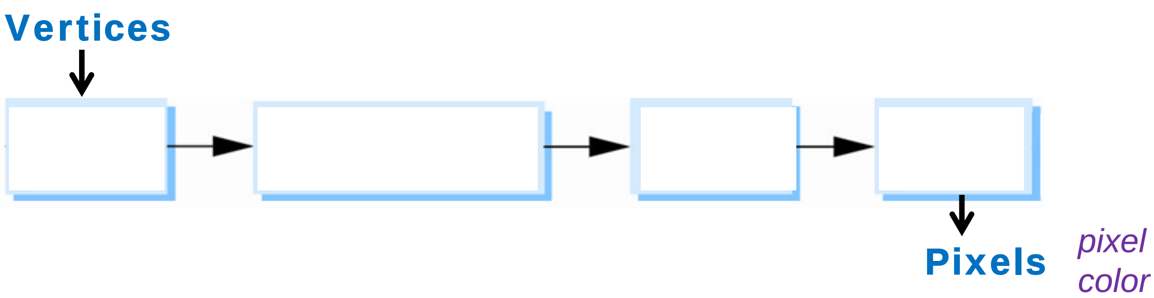
- Pipeline Architectures

Pipeline

- “throughput”

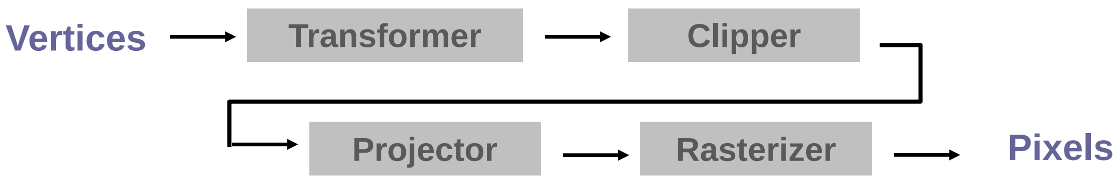


- Geometric pipeline

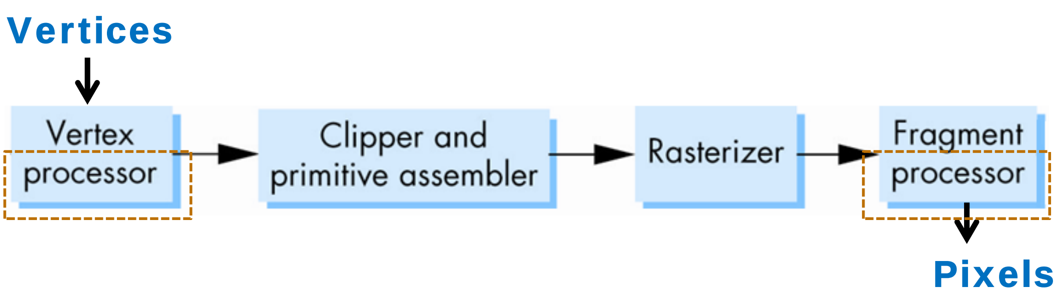


Graphics Pipeline

- Fixed



- Programmable



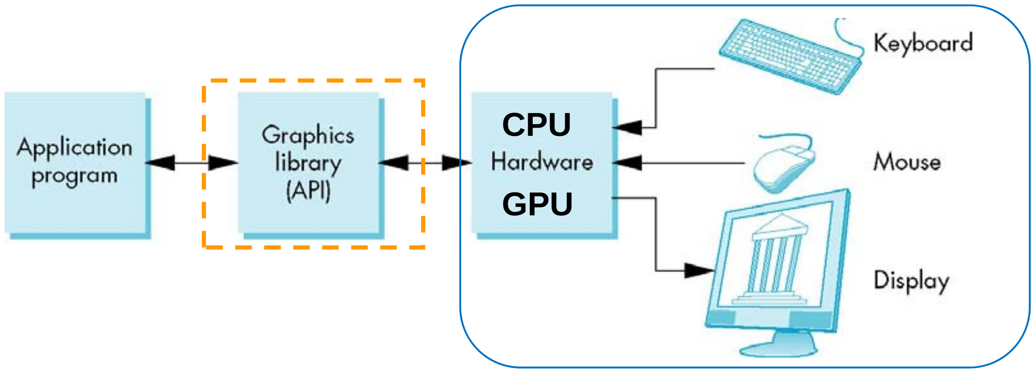
Graphic Processing Unit (GPU)

- Highly tuned for graphics
  - Main architecture based on parallelism and pipelining
    - Many ALU (Arithmetic Logic Units) and Single Instruction Multiple Data (SIMD)
    - Fewer components for the cache and flow control
- GPU strategy: make the workload (as many threads as possible) run as fast as possible

Figure 14: Comparison of an array-increment operation on GPUs (left) and CPUs (right).

GPU Architecture

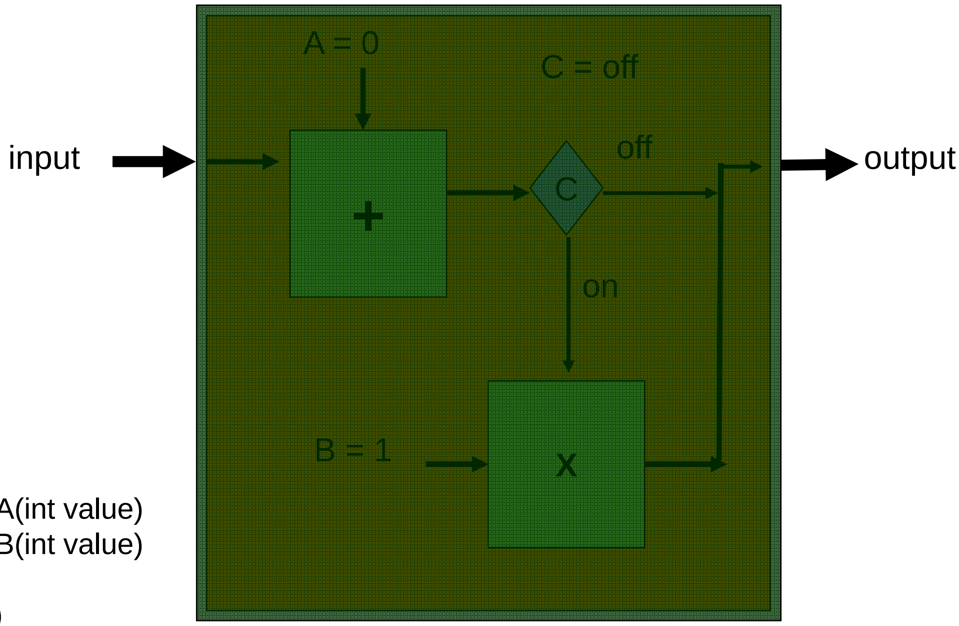
- From fixed function
- To **configurable**
- To programmable



OpenGL (old model)

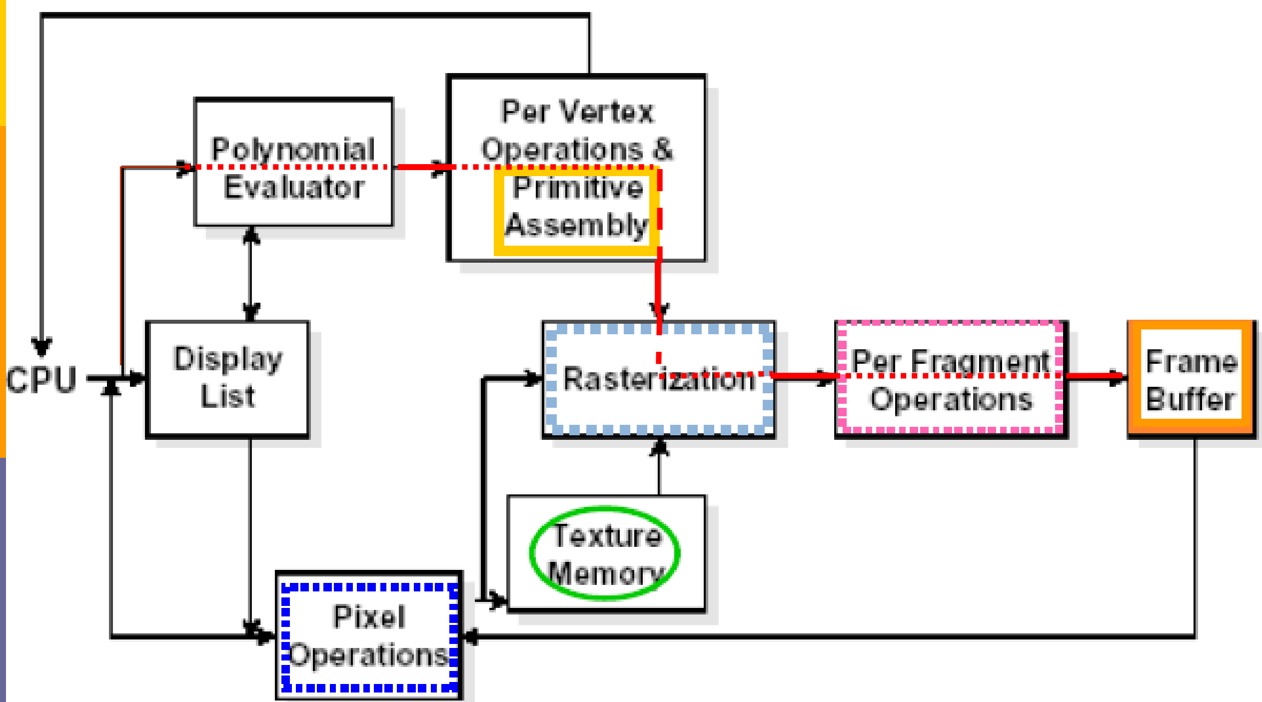
- The prime goal is <u>to study computer graphics</u>.
- We are using an API to help us attain that goal.

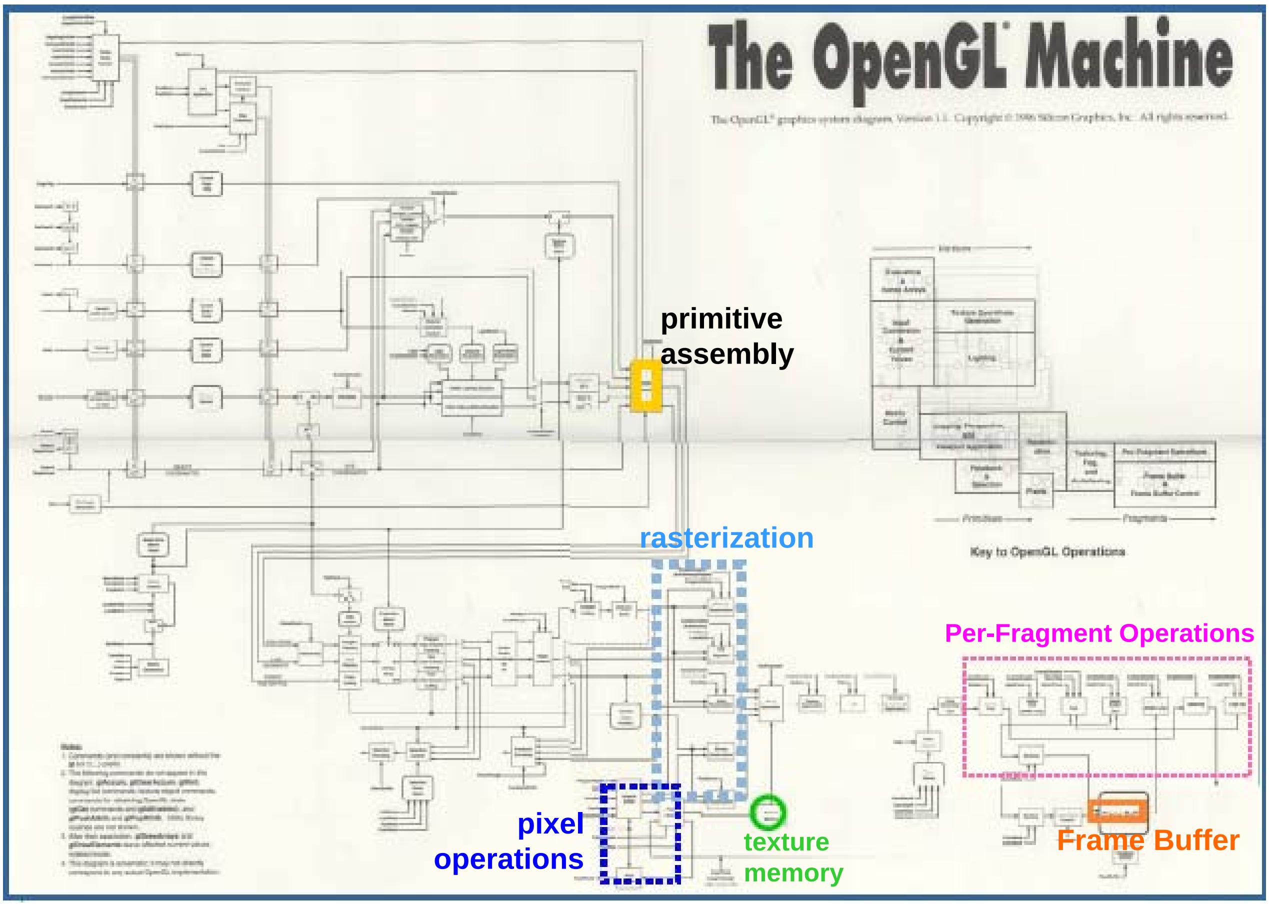
- OpenGL
  - Graphics rendering API
  - Looking at how it is organized and implemented
    → <u>a state machine</u>
    - A black box that contains a finite-state machine



func_set_value_A(int value)
func_set_value_B(int value)
func_enable_C()
func_disable_C()

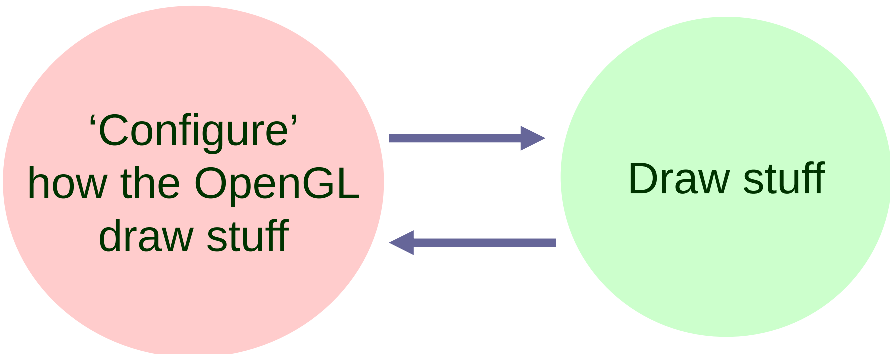
OpenGL Architecture





How (original)OpenGL works

- Conceptual Model



General Structure of an Interactive Graphics Program

- Configure and open a window
- Initialize graphics state
- Process user **events**
- Draw an image

```
display() {…}

myreshape() {…}

init() {…}

main() {
    glutInit…()
    glutCreateWindow()
    glutDispalyFunc(display)

    glutReshapeFunc(myreshape)

    init()

    glutMainLoop()
}
```


Input Process States

- Input process: *what the input device is doing by the graphics library's point of view*
- Measure process
  - What the device returns to the user program
    - (keyboard: a string)
    - (locator: position)
  - Input device returns a set of logical measures
  - Current value of the measure may be echoed on the display
- Trigger
  - Is a physical input on the device with which the user can signal the computer
    - (keyboard: "return" key)
    - (locator: "click" button on the pointing device)
  - Input device returns the trigger signal

Input Modes

- Initialization of input device
  = start of the measure process
  - explicit function call in API
  - automatic

- Measure of devices in 3 distinct modes:
  - (defined by the relationship between the measure process and the trigger)
    - Request mode
    - Sample mode
    - Event mode

Input Modes

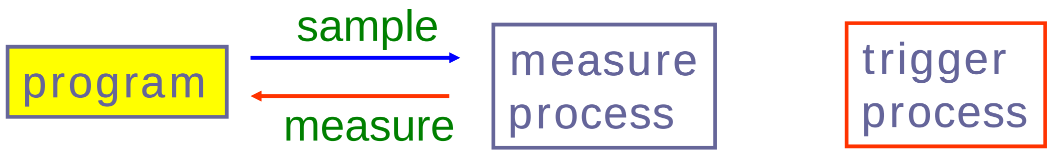
- **Request mode**
- Sample mode
- Event mode

  - Ex) Scanf
  - The input is taken from the device when the measure process receives the trigger signal

Input Modes

- Request mode
- **Sample mode**
- Event mode

```
Request_locator (device_id, &measure)
Sample_locator (devide_id, &measure)
```

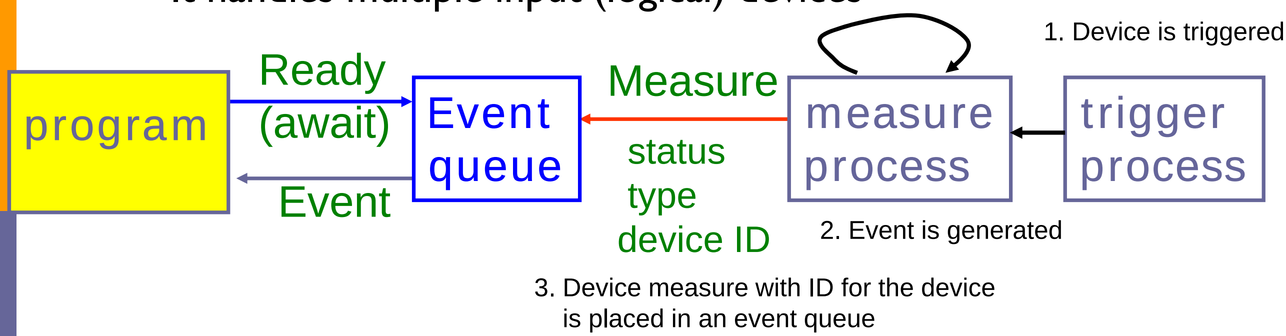


- The input is immediate. The input is taken from the device as soon as the input procedure is executed.
- Ex) getc

Input Modes

- Request mode
- Sample mode
- **Event mode**
  - It handles multiple input (logical) devices

Placing events in the event queue is completely independent of what the application program does with these events



- **Callback** function is associated with a specific type of event.

Programming Event-Driven Input

- Use of the callback mechanism
- Handling the events that are recognized by the window system
  - We write callback function that govern how the application program responds to these events

General Structure of an Interactive Graphics Program

- Configure and open a window
- Initialize graphics state
- Process user **events**
- Draw an image

GLUT → GLFW

```
display() {…}

myreshape() {…}

init() {…}

main() {
    glutInit…()
    glutCreateWindow()
    glutDispalyFunc(display)

    glutReshapeFunc(myreshape)

    init()

    glutMainLoop()
}
```

Lab Session Tomorrow

- OpenGL Introduction
  - OpenGL 4.0
  - GLSL (OpenGL Shader Language)
  - GLEW, GLM
  - GLFW
- Should have completed HW#0.
  - Programming environments should be set already.
- Bring your notebook computer if you have one.

End Note

- Will be used for your attendance check
  - Write your name &
  - What you learned in the lecture.
    - 1~2 keywords (and possibly with a brief explanation)
      - Ex)
        * raterization (픽셀화 하는 과정)
        * configurable architecture (그래픽스 파이프라인에서 속성들은 사용자가 정의해줄 수 있음)
    - Do not need to write up everything.
      Just a short summary would do it.
    - But no summary, only a half attendance will be counted.