

CS174A Lecture 2

Last Lecture Recap

- ***Intro to Computer Graphics***
 - Applications
 - History
 - First few animated games, interactivity
- ***Examples of 3D animated movies***
 - Realism
 - Special effects
 - Compositing
 - Cartoons

Last Lecture Recap (contd.)

- **Areas**

- Flight simulation
- CAD
- Modeling with clay
- Scientific visualization
- Architectural visualization
- Information visualization
- Art, texture mapping

Last Lecture Recap (contd.)

- *Elements of CG*

- **Modeling**: points, lines, polygons, curves, surfaces, voxels, plant, smoke, cloth
- **Rendering**: 3D scene, lights, point-of-view, shading, visibility, projection
- **Animation**: key-frame, procedural, behavioral, physics-based, motion capture
- **Interaction**: gaming, VR

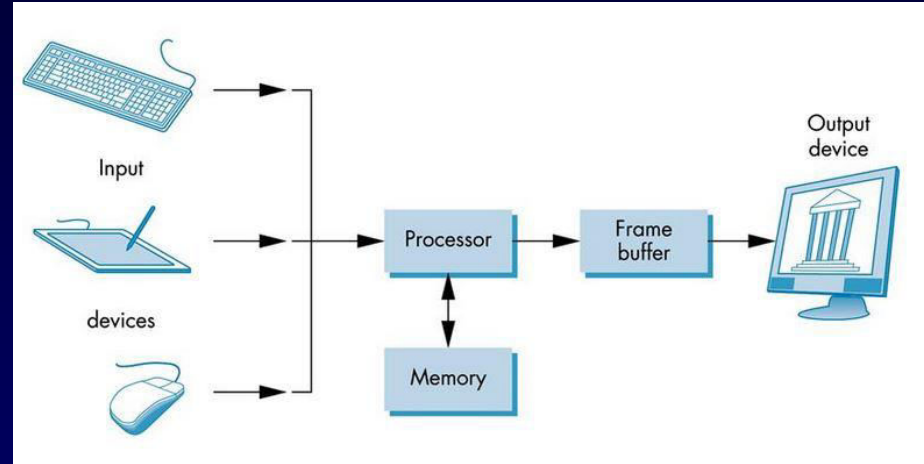
A Basic Graphics System

Input devices

CPU vs. GPU

Computing & rendering system

Output devices



Input Devices

Keyboard

Mouse

Game controller

Tablet & Pen

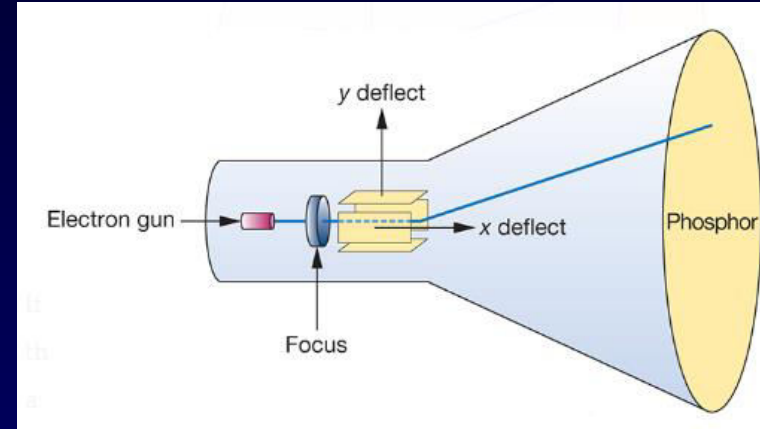
Other sensors

- *Data glove*
- *Sound*
- *Gesture*
- *Etc.*

Output Devices

CRT (Cathode Ray Tube)

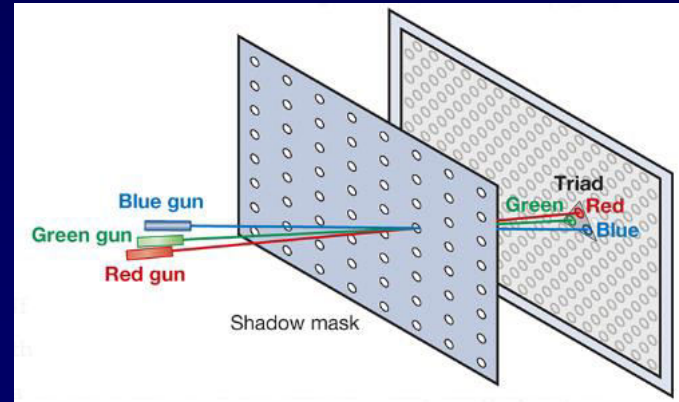
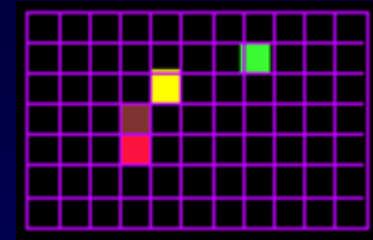
- Electrons strike Phosphor coating and emits light
- Direction of beam controlled by deflection plates
- Random-scan, calligraphic or vector CRT
- Moving beam to new location
- Refresh rate: 60 Hz – 85 Hz



Output Devices (Contd.)

Raster CRTs ($n \times m$ phosphor)

- Framebuffer
 - 1 bit: 2 levels only, b/w
 - 8 bits: gray scale, 256 gray levels or colors
 - 8 bits per color (RGB) = 24 bits = 16K colors
 - 12 bits per color: HDR
- 3 different colored Phosphors: triads
- Shadow mask
- Interlaced vs. Non-Interlaced displays
- Interlaced: used in commercial TV
- Single vs. double buffering



Output Devices (Contd.)

Screen Resolutions of Raster CRTs (n x m phosphor)

- TV: 640x480 pixels
- HD: 1920x1080
- 35mm: 3000x2000

Output Devices (Contd.)

Memory Time & Space Requirements

- Screen resolution = $n \times m$
- Refresh rate = r Hz
- Interlaced vs. non-interlaced
- Color depth = b bits/pixel

If non-interlaced, memory read time = $1 / (n * m * r)$ secs

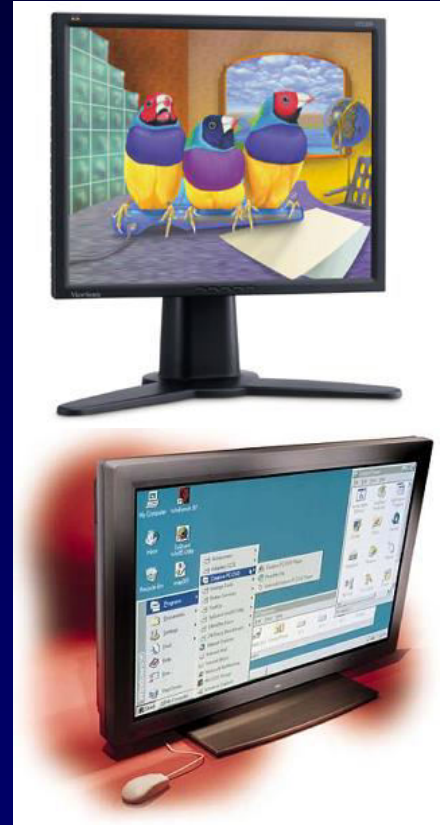
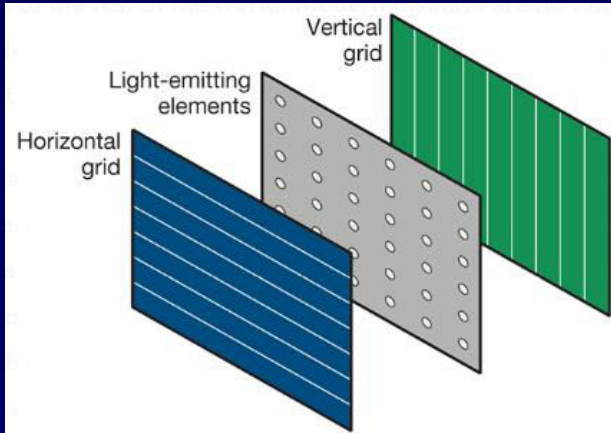
If interlaced, memory read time = $2 / (n * m * r)$ secs

Memory space per second = $(n * m * b * r) / 8$ bytes

Output Devices (Contd.)

Flat Screen Displays

- Raster based: active matrix with transistors at grid points
- LEDs: light emitting diodes
- LCDs: polarization of liquid crystals
- Plasma: energize gases to glow plasma



Output Devices (Contd.)

Other Output Devices

- Printers & Plotters: raster based, no refresh
- Stereo Displays: 3D TVs/movies, fast switching of left and right eye polarized images

Output Devices (Contd.)

VR (Virtual Reality)

- Flat panel technology
- Stereoscopic
- **Foveated Rendering**: hi-res where viewer is focusing, lo-res elsewhere
- Track body, finger, and head locations
- Other input devices: force sensing gloves, sound



Exotic Display Devices



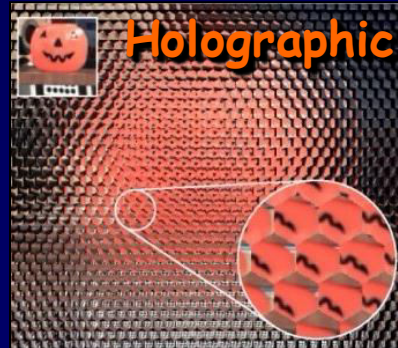
Immersive



Head-Mounted



Autostereoscopic



Holographic



Volumetric

SIGGRAPH 2018 Research Trailers

May 2018:

<https://www.youtube.com/watch?v=t952yS8tcg8>

November 2018:

https://www.youtube.com/watch?v=wdKpXvF_3AU

SIGGRAPH 2017 Research Trailers

May 2017:

https://www.youtube.com/watch?v=3OGKh_9Rj_8

November 2017:

<https://www.youtube.com/watch?v=5YvIHREdVX4>

One Example of today's industry “hacks”

<https://www.youtube.com/watch?v=ct3vWWI86f8>

- Tools packaged with production software suites
- Video does not claim to use any math that converges to real-life physics formulas
 - Approximation is not convergence!



The Difficulties of Learning Computer Graphics

(And teaching it!)

Background

- Most of today's approaches to creating a low-level, math-based graphics program come with a substantial learning curve
 - Numerous complex steps
- In terms of preparation and learning, it is costly to build prototypes using low-level 3D graphics programs today



Background

- A common alternative is to:
 - Forgo low-level control
 - Employ overpowered industrial tools to wrap basic graphics functionality
 - Simple mathematics of projecting 3D triangles onto a 2D plane of pixels



Background

- Graphics beginners are faced with long lists of setup steps
 - Especially in the case of newer "shader-based" approaches
 - These approaches are both more complex and harder to learn

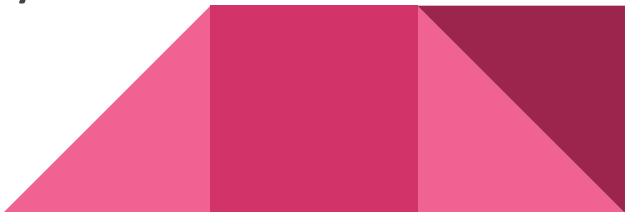


Background

- The graphics learning curve is extreme
- Drawing just a single triangle requires:
 - Secondary "shader" programs
 - Multiple types of GPU memory management
 - At least three computer languages (in the case of WebGL)



WARNING

- Graphics tutorials online abound that still use outdated “pre-shader” coding paradigms
 - They're alluring due to their simplicity
 - However, support for their older commands has been:
 - Removed from new graphics cards
 - Never existed in web browser implementations
 - Eventually it dawns on newcomers that they must commit to learning the new way
- 

Background

- Mandatory setup steps initialize the graphics card (GPU):
 - Give it the shader program code
 - Give it raw data buffers pertinent to the 3D scene
 - Obtain pointers to these in GPU memory



Background

- Even after setup:
 - All the subsequent shape-drawing actions of the programmer are still cluttered with boilerplate code
 - Loading and switching between pointers to the GPU
- Nothing is drawn without these steps; there is no built-in way to organize them



Background

- Common graphics card interfaces exposed for doing the above steps are called:
 - DirectX
 - OpenGL
 - More widely available and more prevalent in education



Background

- Regardless, graphics methods all follow a similar pattern
- The phrase ``OpenGL program'' is widely taken to imply C++ due to the ubiquity of the language in early graphics education, but it need not be:
 - Python, Java, and JavaScript can make the same OpenGL calls



Background

- C++
- Python
- JavaScript
- Java
- 3D Graphics programming in one language feels familiar in all the others




Background

- JavaScript is currently the only means of running code on browsers inside of web pages
- When JavaScript is used, the OpenGL commands are called WebGL
 - They are still the same API function calls as would appear in C++



Your Textbook

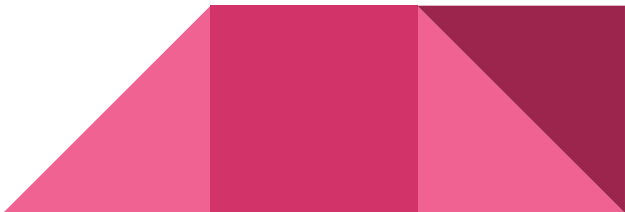
- Edward Angel's "Interactive Computer Graphics...with WebGL" is the leading gentle introduction for those wanting to do graphics programming
 - Used at:
 - UCLA and other schools
 - SIGGRAPH's official intro course
 - Has supplemental code and links to demos that run on the web
 - Chapter 2 is a suggested organization of WebGL programs
 - hello world
 - Other web tutorials share program structure
- 

Edward Angel's The Case for Teaching Computer Graphics with WebGL: A 25-Year Perspective

- Described his rationale for eventually moving his helpful C++ based libraries over to WebGL
 - Worsening learning curve of C++ graphics setup
 - Difficulty in setting up uniform C++ compiling environments for all students
 - Your all's inconsistent hardware



Edward Angel's The Case for Teaching Computer Graphics with WebGL: A 25-Year Perspective

- He found that WebGL has:
 - Comparable performance to C++
 - The advantages of a standardized environment on all platforms (including phones)
 - An interpreted code engine that aids development
 - Advanced coding tools built right into modern web browsers
- 

Sounds good

- WebGL is the current best platform for graphics training
 - Code examples on the web are more easily:
 - Run as demos
 - Analysed
 - Packaged with inline tutorials
 - Compiled to any machine
 - Debugged
 - Hosted
 - Shared
 - Remixed



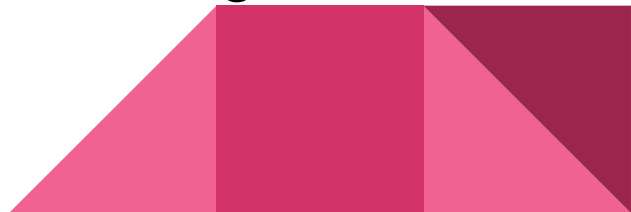
JavaScript

- JavaScript's functional programming styles
 - Specific graphics applications
 - Also tend toward smaller total source code



JavaScript

- We make full use of the 2015 “es6” version of JavaScript
 - Adds further brevity and power to the language
 - Be sure to include “es6” in all Google searches!!
- Performance workarounds such as WebAssembly and Emscripten can run at near native speeds
- JavaScript's benefits for prototyping and sharing code on the web are clear



Example sites that allow graphics programming right inside the browser

- JSFiddle
- Dwitter
- WebGL Playground
- Shadertoy
- The 174a web server



The Textbook's web demos

- <https://www.cs.unm.edu/~angel/WebGL/7E/Common/>



Linear Algebra Review

Linear Algebra: The Algebra of Vectors and Matrices (and Scalars)

Vector spaces

Matrix algebra

Coordinate systems

Affine transformations

Vectors

N-tuple of scalar elements

$$\mathbf{v} = (x_1, x_2, \dots, x_n), \quad x_i \in \mathbb{R}$$

Vector:

Bold lower-case

Scalar:

Italic lower-case

Vectors

N-tuple:

$$\mathbf{v} = (x_1, x_2, \dots, x_n), \quad x_i \in \mathbb{R}$$

Magnitude:

$$|\mathbf{v}| = \sqrt{x_1^2 + \dots + x_n^2}$$

Unit vectors

$$\mathbf{v} : |\mathbf{v}| = 1$$

Normalizing a vector

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{|\mathbf{v}|}$$

Operations with Vectors

Addition

$$\mathbf{x} + \mathbf{y} = (x_1 + y_1, \dots, x_n + y_n)$$

Multiplication with scalar (scaling)

$$a\mathbf{x} = (ax_1, \dots, ax_n), \quad a \in \mathbb{R}$$

Properties

$$\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$$

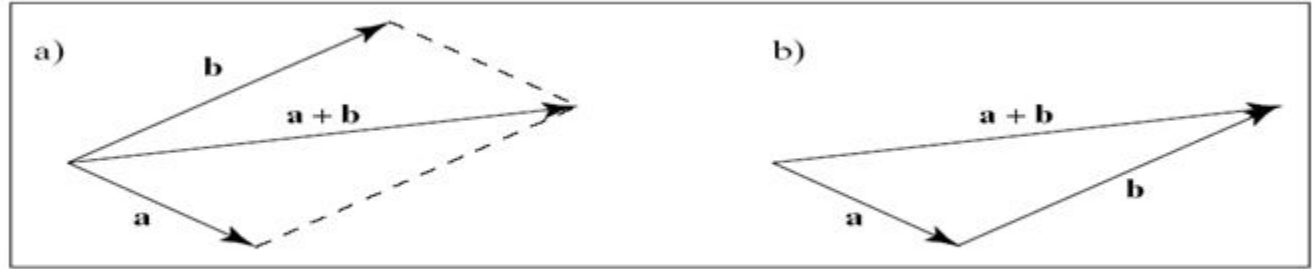
$$(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$$

$$a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}, \quad a \in \mathbb{R}$$

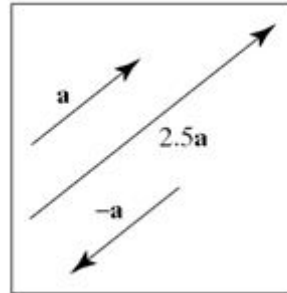
$$\mathbf{u} - \mathbf{u} = \mathbf{0}$$

Visualization of 2D and 3D Vectors

Addition

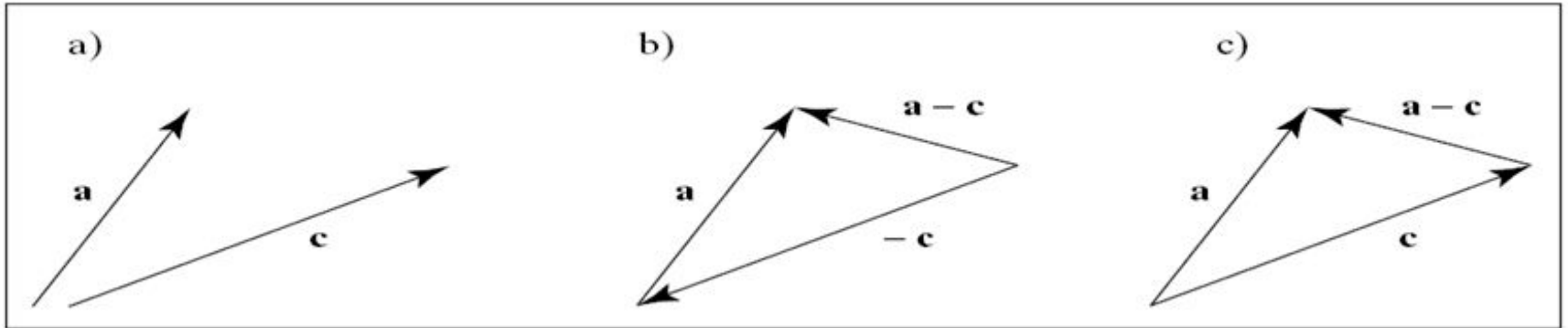


Scaling



Subtraction

Adding the negatively scaled vector



Linear Combination of Vectors

Definition

A linear combination of the m vectors $\mathbf{v}_1, \dots, \mathbf{v}_m$ is a vector of the form:

$$\mathbf{w} = a_1\mathbf{v}_1 + \dots + a_m\mathbf{v}_m, \quad a_1, \dots, a_m \text{ in } \mathbb{R}$$

Special Cases

Linear combination

$$\mathbf{w} = a_1 \mathbf{v}_1 + \dots + a_m \mathbf{v}_m, \quad a_1, \dots, a_m \text{ in } \mathbb{R}$$

Affine combination:

A linear combination for which $a_1 + \dots + a_m = 1$

Convex combination

An affine combination for which $a_i \geq 0$ for $i=1, \dots, m$

Linear Independence

For vectors $\mathbf{v}_1, \dots, \mathbf{v}_m$

If $a_1\mathbf{v}_1 + \dots + a_m\mathbf{v}_m = \mathbf{0}$ iff $a_1 = a_2 = \dots = a_m = 0$

then the vectors are linearly independent