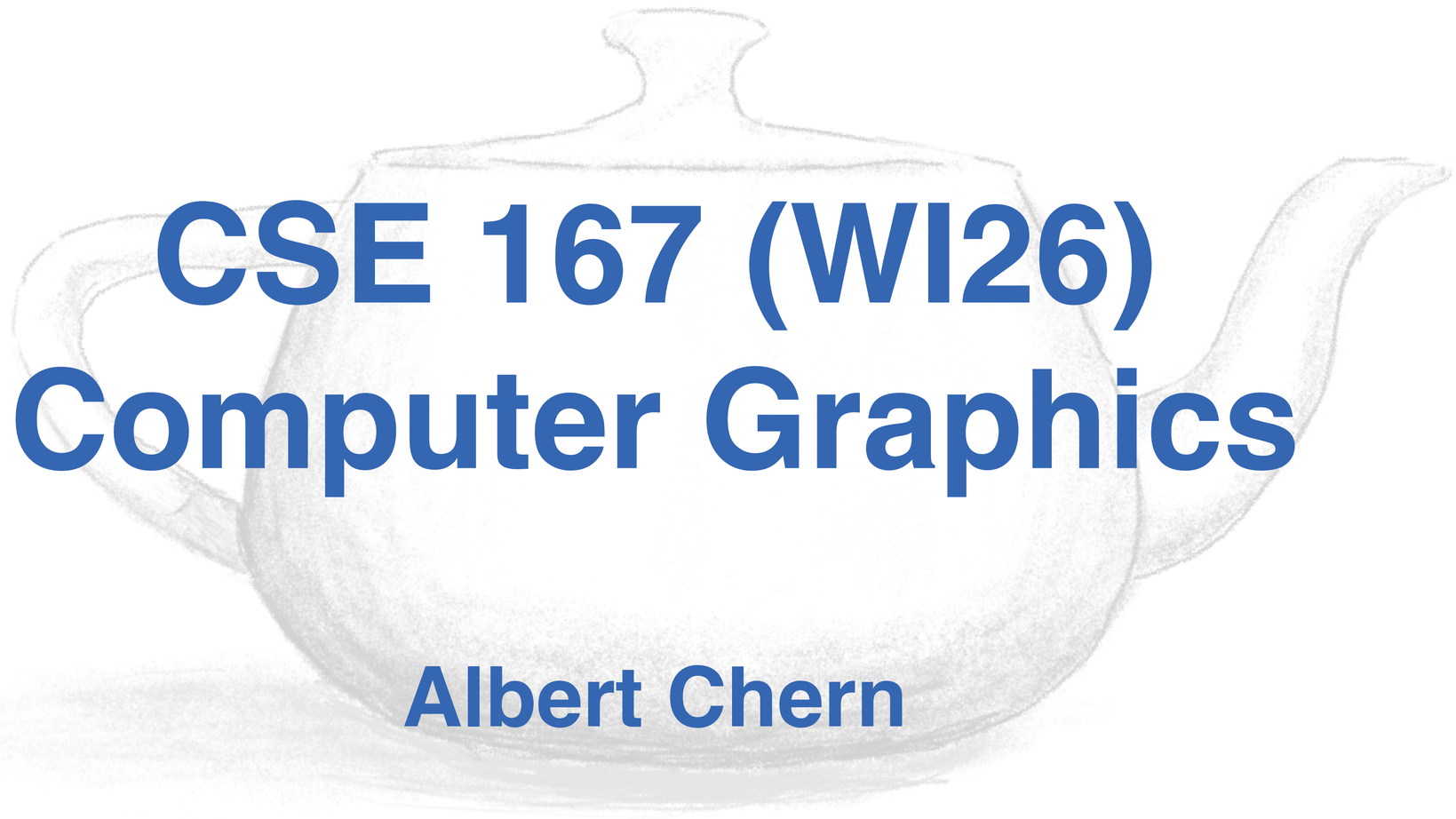


CSE 167 (WI26)
Computer Graphics

Albert Chern

UC San Diego
JACOBS SCHOOL OF ENGINEERING

Course Logistics

- Course logistics
- Getting started
- What is computer graphics
- This course
- How to draw a picture?

Lectures

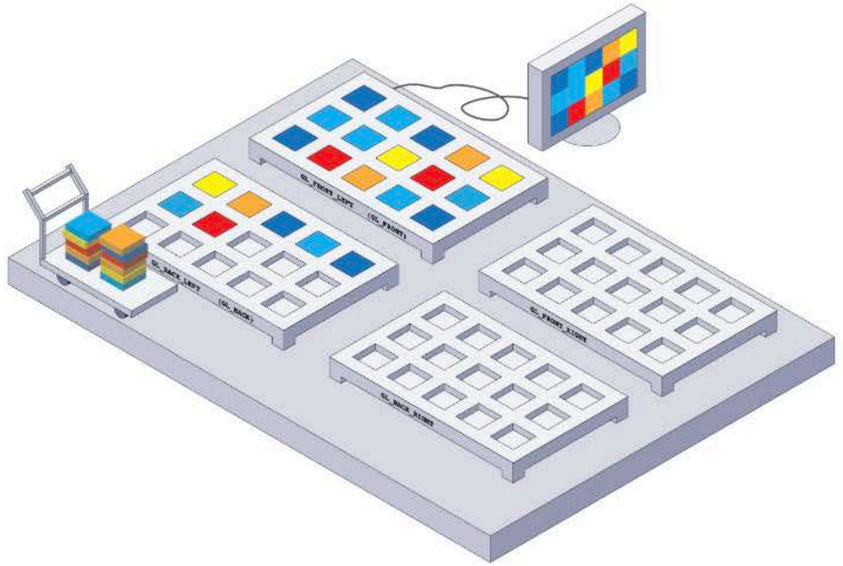
- **Lectures (Albert Chern)**
  - Center Hall 119
- **Discussion (by the TAs)**
  - Tue 19:00–19:50, Pepper Canyon Hall 109 (starting from Week2)
- **TAs**
  - Mustafa Yaldiz
  - Rudresh Veerkhare
- **Tutors**
  - Ritoban Roy-Chowdhury
  - Annapurna Saladi

cseweb.ucsd.edu/~alchern/teaching/cse167_wi26/

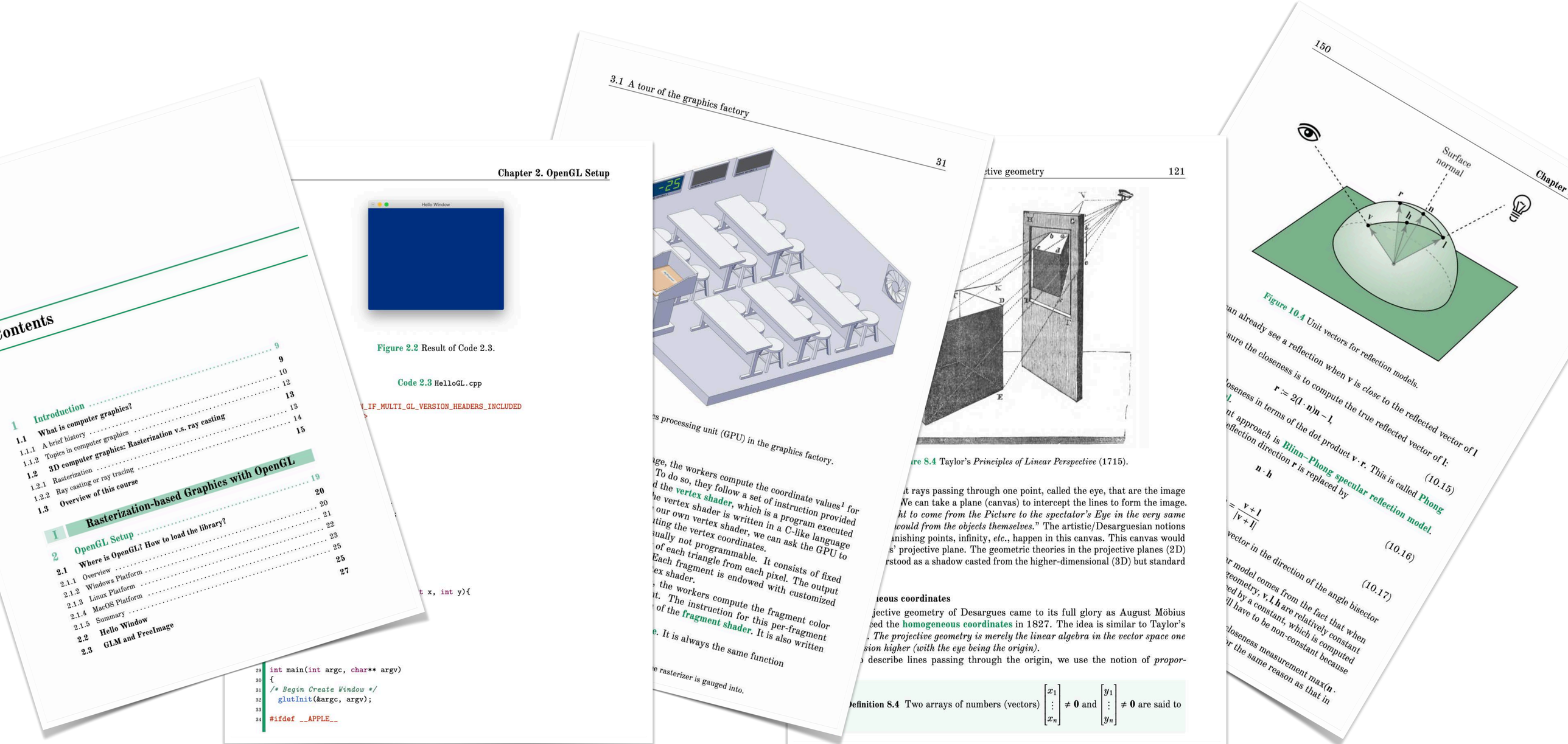
CSE 167 (WI 2026) Computer Graphics

Welcome to CSE 167, Introduction to Computer Graphics.

- **Lecture:** Tue Thu 8:00 - 9:20. Center Hall 119.
- **Discussion Session:** Tue 19:00-19:50. Pepper Canyon Hall 109.
- **Podcast:** Recorded lectures https://podcast.ucsd.edu/watch/wi26/cse167_a00
- **Sites:**
  - **This page:** Slides, lecture notes, HW
  - **Canvas:** Link back to this page, and 1st week FinAid survey quiz.
  - **Piazza:** https://piazza.com/ucsd/winter2026/cse167 Q&A forum for the class.
  - **Gradescope:** https://www.gradescope.com/courses/1212101 (log in with School credential, or use entry code: ZJ324Z) HW submission.
- **Lecture note:** Introduction to Computer Graphics



Lecture Note



Contents

1 Introduction 9
1.1 What is computer graphics? 9
1.1.1 A brief history 10
1.1.2 Topics in computer graphics 12
1.2 3D computer graphics: Rasterization v.s. ray casting 13
1.2.1 Rasterization 13
1.2.2 Ray casting or ray tracing 14
1.3 Overview of this course 15

I Rasterization-based Graphics with OpenGL 19

2 OpenGL Setup 20
2.1 Where is OpenGL? How to load the library? 20
2.1.1 Overview 21
2.1.2 Windows Platform 22
2.1.3 Linux Platform 23
2.1.4 MacOS Platform 25
2.1.5 Summary 25
2.2 Hello Window 27
2.3 GLM and FreeImage

Chapter 2. OpenGL Setup

Hello Window

Figure 2.2 Result of Code 2.3.

Code 2.3 HelloGL.cpp

```
L_IF_MULTI_GL_VERSION_HEADERS_INCLUDED
>
```

```
t x, int y){
```

```
int main(int argc, char** argv)
{
/* Begin Create Window */
  glutInit(&argc, argv);

#ifdef __APPLE__
```

3.1 A tour of the graphics factory 31

cs processing unit (GPU) in the graphics factory.

age, the workers compute the coordinate values[1] for
To do so, they follow a set of instruction provided
d the **vertex shader**, which is a program executed
he vertex shader is written in a C-like language
our own vertex shader, we can ask the GPU to
ating the vertex coordinates.
sually not programmable. It consists of fixed
of each triangle from each pixel. The output
Each fragment is endowed with customized
ex shader.
the workers compute the fragment color
t. The instruction for this per-fragment
of the **fragment shader**. It is also written
e. It is always the same function

e rasterizer is gauged into.

tive geometry 121

re 8.4 Taylor's *Principles of Linear Perspective* (1715).

t rays passing through one point, called the eye, that are the image
We can take a plane (canvas) to intercept the lines to form the image.
ht to come from the Picture to the spectator's Eye in the very same
would from the objects themselves." The artistic/Desarguesian notions
anishing points, infinity, *etc.*, happen in this canvas. This canvas would
s' projective plane. The geometric theories in the projective planes (2D)
rstood as a shadow casted from the higher-dimensional (3D) but standard

eous coordinates

jective geometry of Desargues came to its full glory as August Möbius
ced the **homogeneous coordinates** in 1827. The idea is similar to Taylor's
. *The projective geometry is merely the linear algebra in the vector space one*
sion higher (with the eye being the origin).
describe lines passing through the origin, we use the notion of *propor-*

Definition 8.4 Two arrays of numbers (vectors) $\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \neq \mathbf{0}$ and $\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \neq \mathbf{0}$ are said to

150

Figure 10.4 Unit vectors for reflection models.

an already see a reflection when $\mathbf{v}$ is *close* to the reflected vector of $\mathbf{l}$
sure the closeness is to compute the true reflected vector of $\mathbf{l}$:

$$\mathbf{r} := 2(\mathbf{l} \cdot \mathbf{n})\mathbf{n} - \mathbf{l}. \tag{10.15}$$

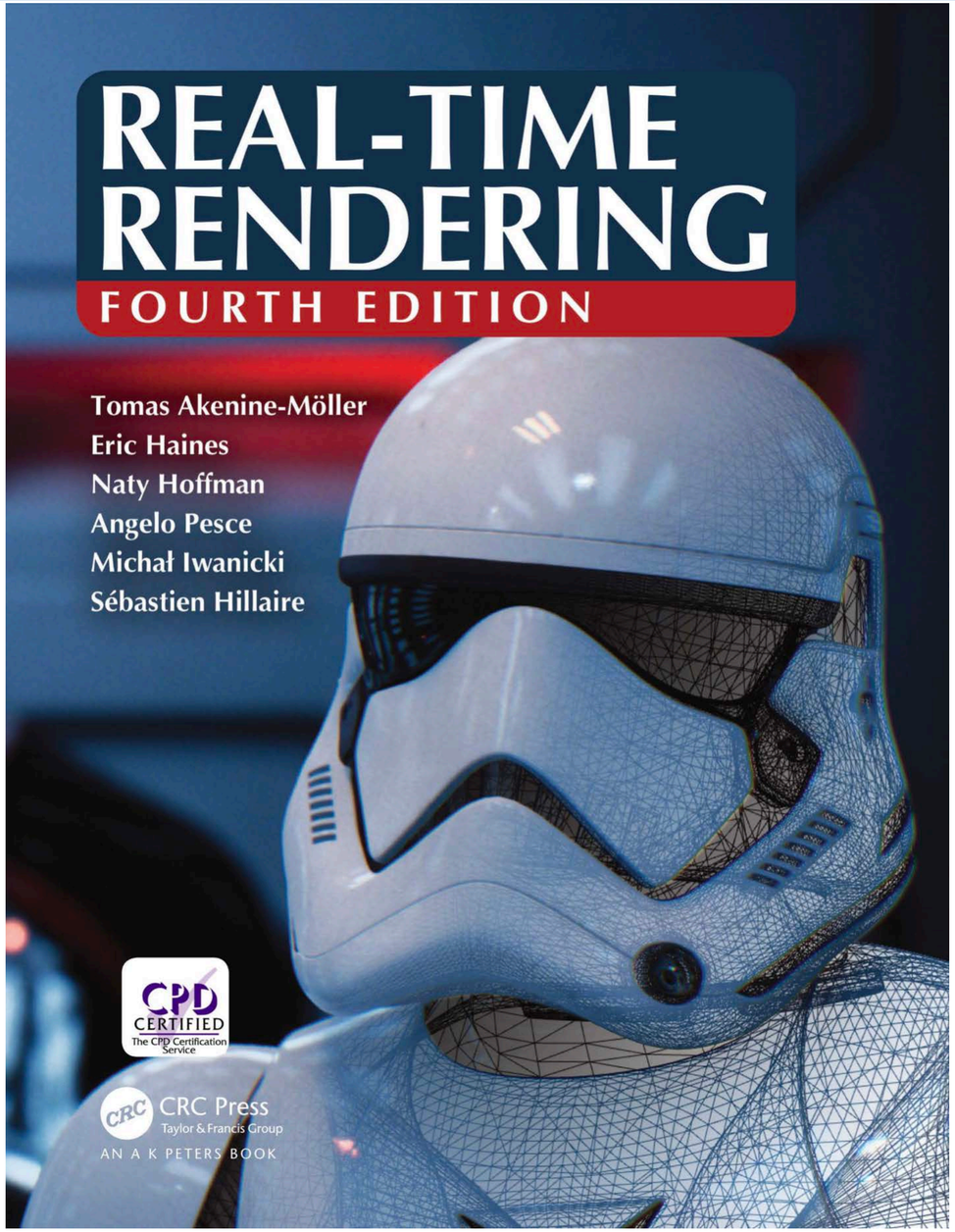
loseness in terms of the dot product $\mathbf{v} \cdot \mathbf{r}$. This is called **Phong**
l.
nt approach is **Blinn–Phong specular reflection model**.
eflection direction $\mathbf{r}$ is replaced by

$$\mathbf{n} \cdot \mathbf{h} \tag{10.16}$$

$$= \frac{\mathbf{v} + \mathbf{l}}{|\mathbf{v} + \mathbf{l}|} \tag{10.17}$$

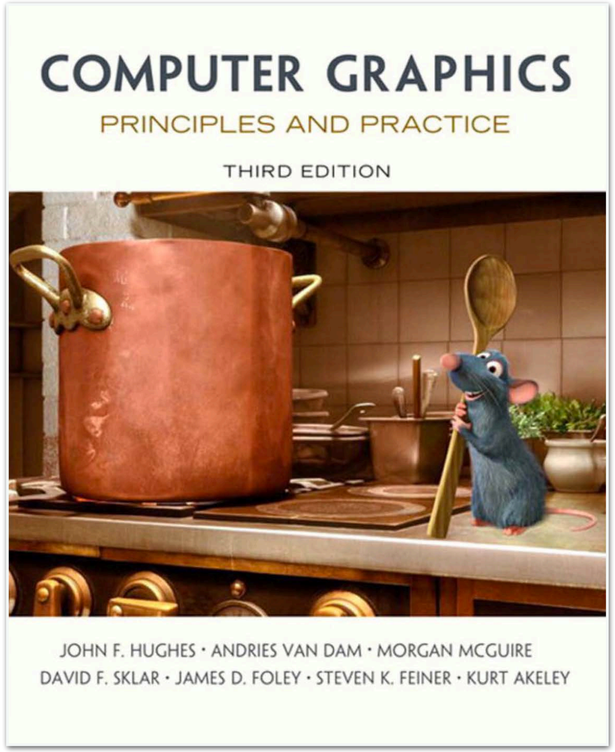
vector in the direction of the angle bisector
r model comes from the fact that when
geometry, $\mathbf{v}, \mathbf{l}, \mathbf{h}$ are relatively constant
ed by a constant, which is computed
ll have to be non-constant because
closeness measurement $\max(\mathbf{n} \cdot$
r the same reason as that in

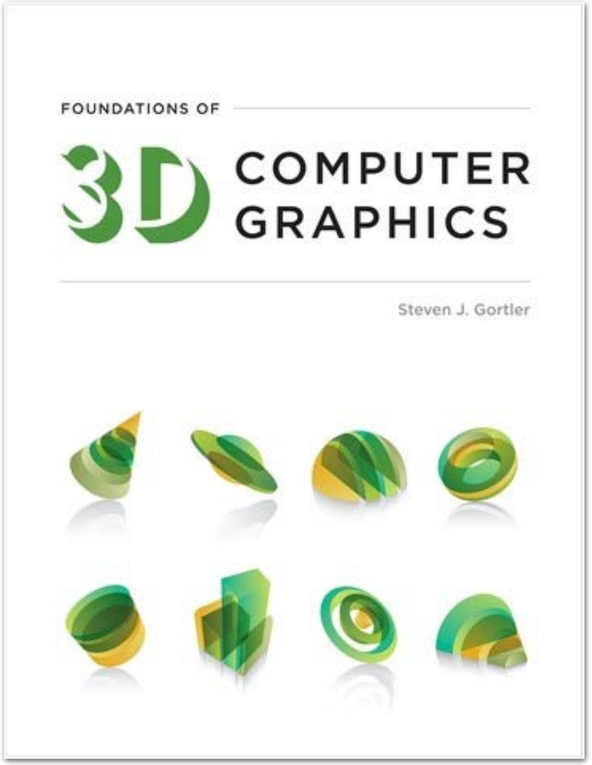
References

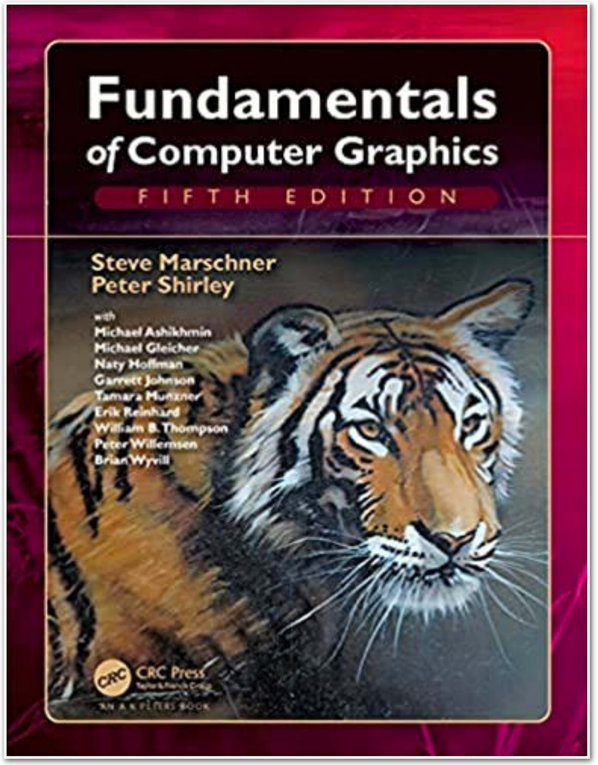


(Available online from UC Library)

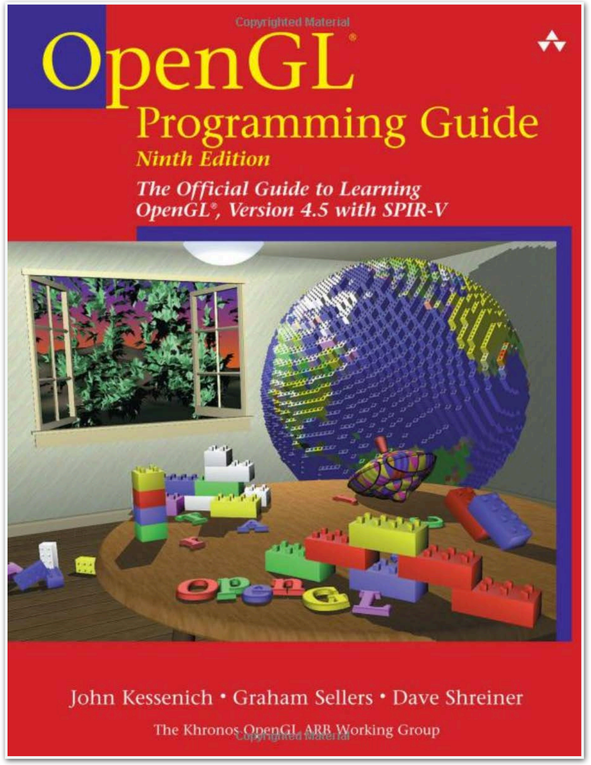
References

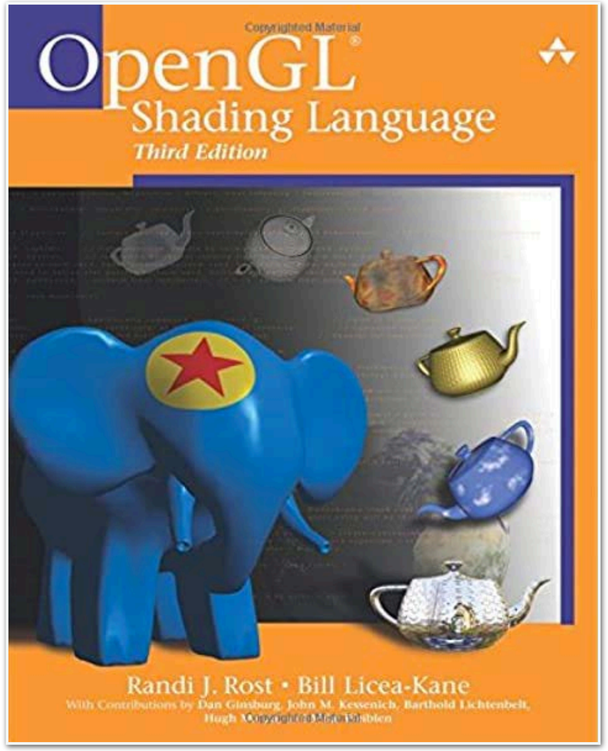

COMPUTER GRAPHICS
PRINCIPLES AND PRACTICE
THIRD EDITION
JOHN F. HUGHES · ANDRIES VAN DAM · MORGAN MCGUIRE
DAVID F. SKLAR · JAMES D. FOLEY · STEVEN K. FEINER · KURT AKELEY


FOUNDATIONS OF
3D COMPUTER GRAPHICS
Steven J. Gortler


Fundamentals
of Computer Graphics
FIFTH EDITION
Steve Marschner
Peter Shirley
with
Michael Ashikhmin
Michael Gleicher
Naty Hoffman
Garrett Johnson
Tamara Munzner
Erik Reinhard
William B. Thompson
Peter Willemsen
Brian Wyvill
CRC Press

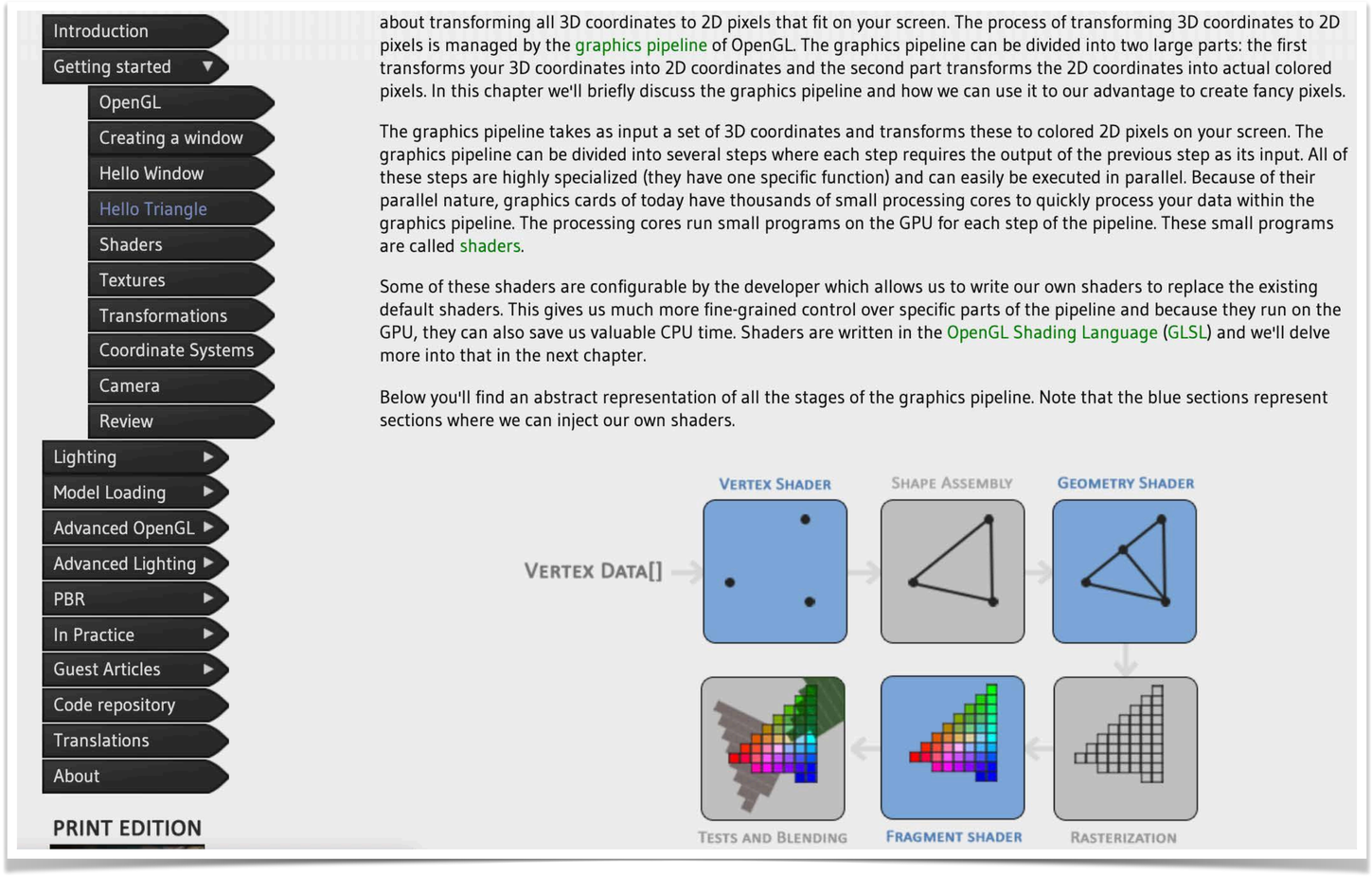
References


Copyrighted Material
OpenGL
Programming Guide
Ninth Edition
The Official Guide to Learning
OpenGL®, Version 4.5 with SPIR-V
John Kessenich • Graham Sellers • Dave Shreiner
The Khronos OpenGL ARB Working Group


Copyrighted Material
OpenGL
Shading Language
Third Edition
Randi J. Rost • Bill Licea-Kane

References

https://learnopengl.com



Prerequisite and some expectation

- **Experience with**
  - basic linear algebra (matrix & vector)
  - C++
- **You can expect that you will**
  - Command and run programs on GPUs (using OpenGL)
  - Deal with lots of floating point numbers (continuous math)
  - Think of problems geometrically
  - View math operators “structurally”
  - See some physics (optics, mechanics)

Grades

- **No quiz or exam**
- **Weekly written exercise 40%** *individual work*
- **Programming HW 60%**
  - HW0–4: 40% *individual work*
  - Final Project: 20% *group of 1 or 2*

- **Passing grade: 70%**
- **Curve letter grade condition:** 3/4 of the class complete the CAPE course evaluation in Week 9,10.

Individual Work

- **Can I discuss problem set with classmates?** *Yes*
- **Can I consult references from book or online?** *Yes*
- **Can I use AI?** *Yes, but view the AI as one of your classmates or colleagues.*

  Don’t copy answers from the work of your classmate.

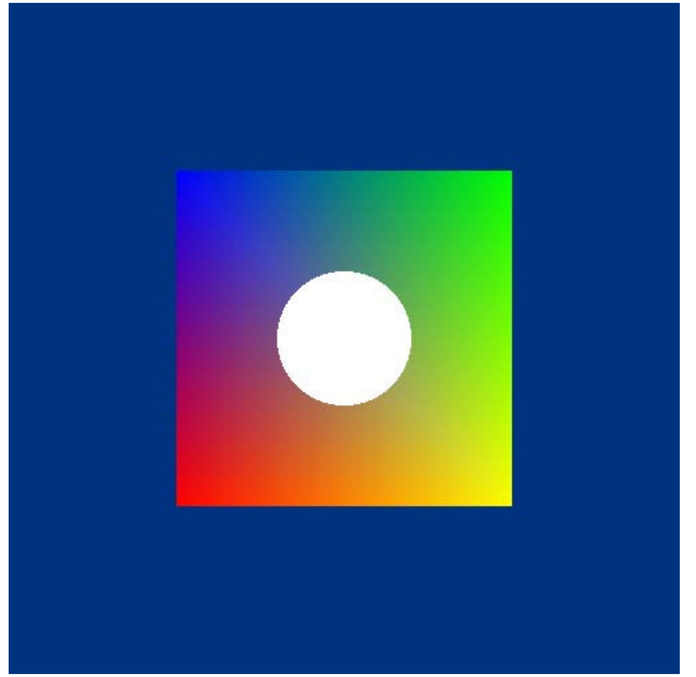
- The final answer should be out of your own work.

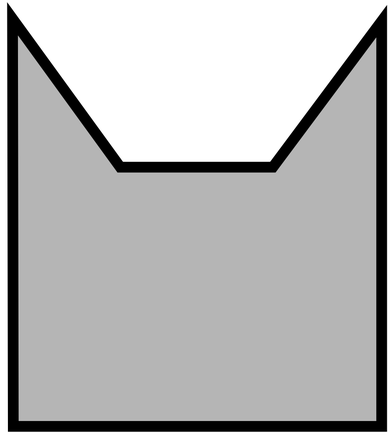
Getting Started

- Course logistics
- Getting started
- What is computer graphics
- This course
- How to draw a picture?

Getting started

- In week1&2, we will start working with **OpenGL**
- HW0 & Exer1 are due 1/16
  - Compile, run, and upload the result (should look like this Fig).
  - Modify the code to get the cat/fox shape
  - Compilation problem: office hour or piazza.
  - Later this week, we explain what happens in the code.





Today

Computer Graphics

Today

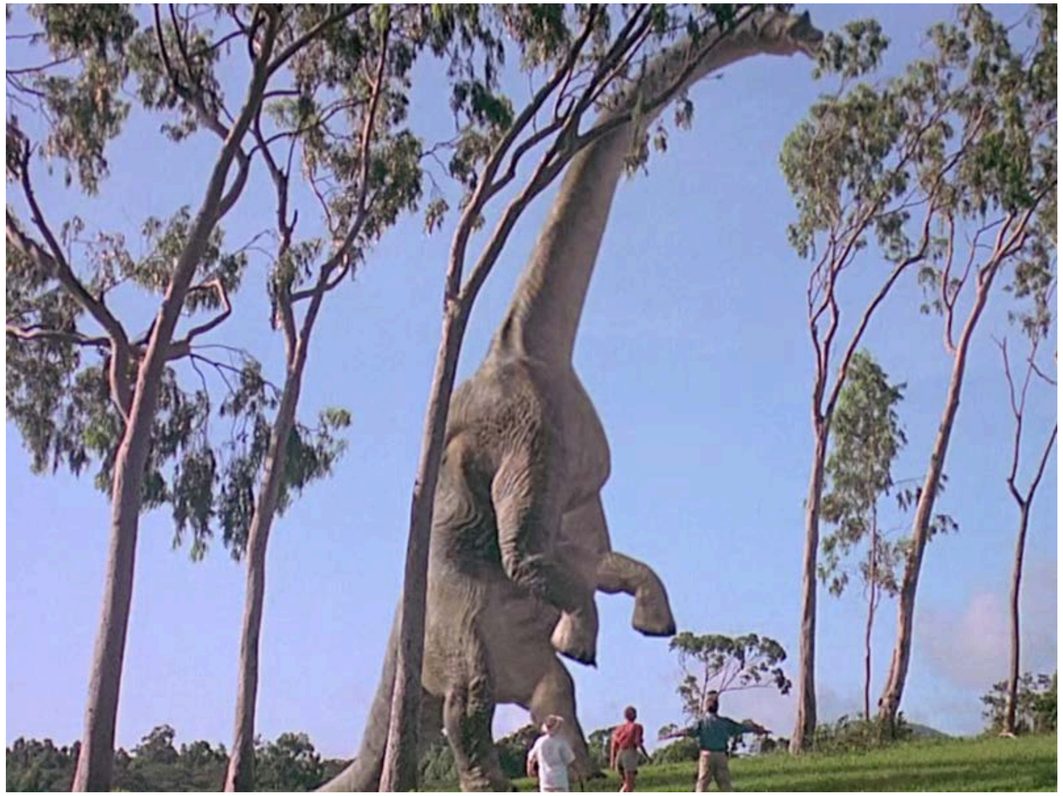
Computer Graphics

- What is computer graphics? (brief history)
- Topics of this course
- How to draw programmatically? (computer graphics pipeline)

What is Computer Graphics?

- Course logistics
- Getting started
- What is computer graphics
- This course
- How to draw a picture?

What is Computer Graphics?

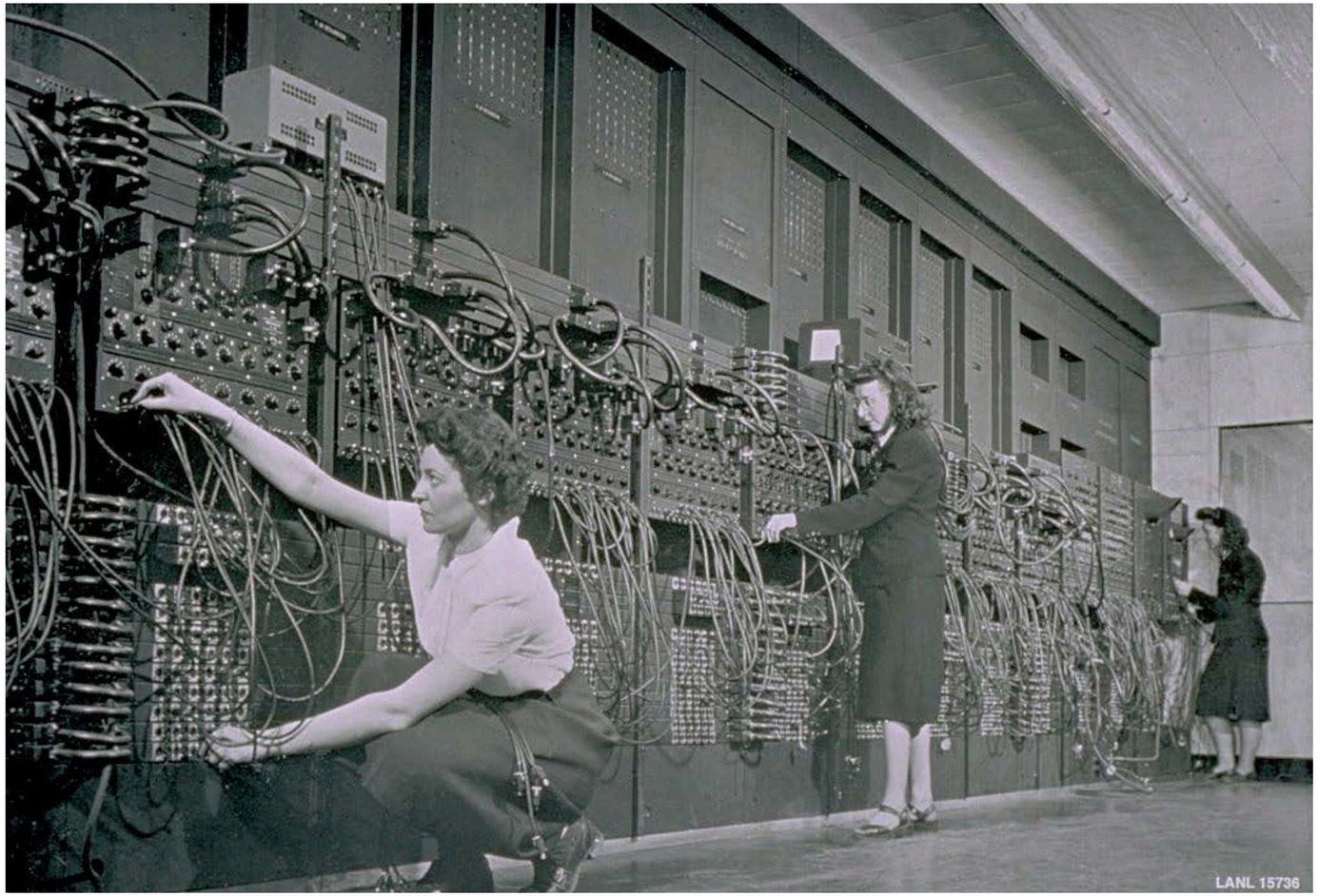


Jurassic Park 1993

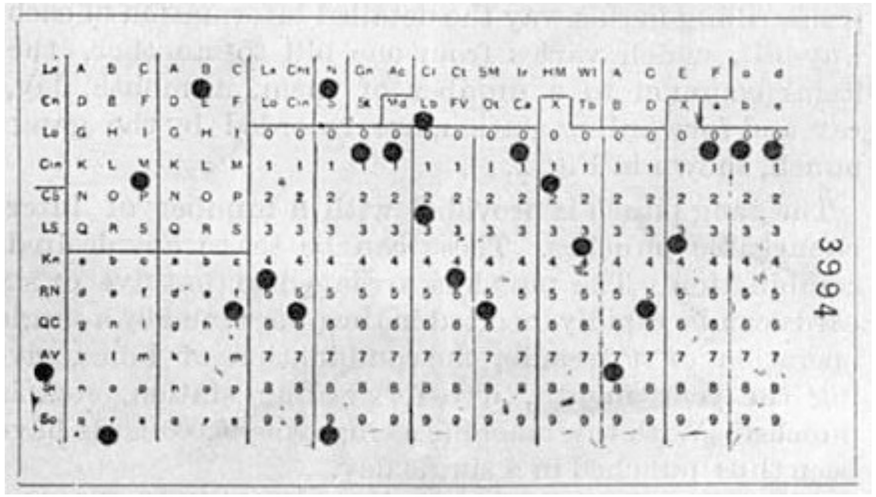


Ratatouille 2007

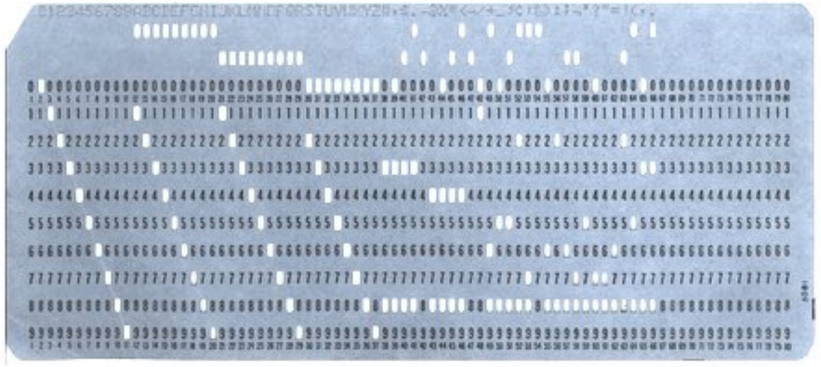
Why Computer Graphics?



early computer (ENIAC) 1945



punched card 1890's



ENIAC I/O 1945

Why Computer Graphics?



30% of the brain is devoted to visual processing

Most efficient way to receive information is in the form of visual data

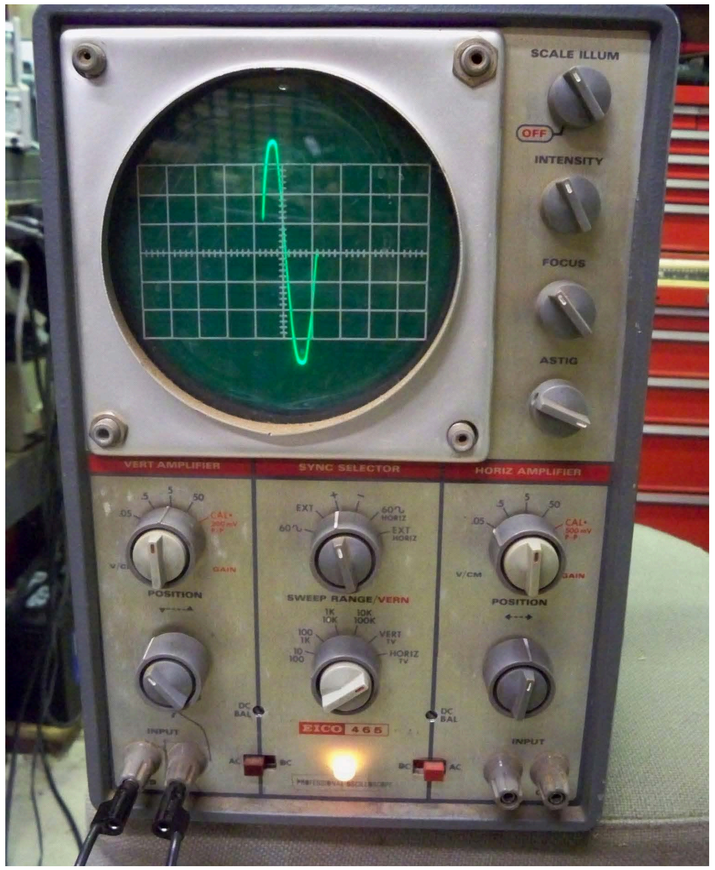
Computer Graphics

Computer graphics

The use of computer to synthesize visual informations.

History of Graphics: Visual output

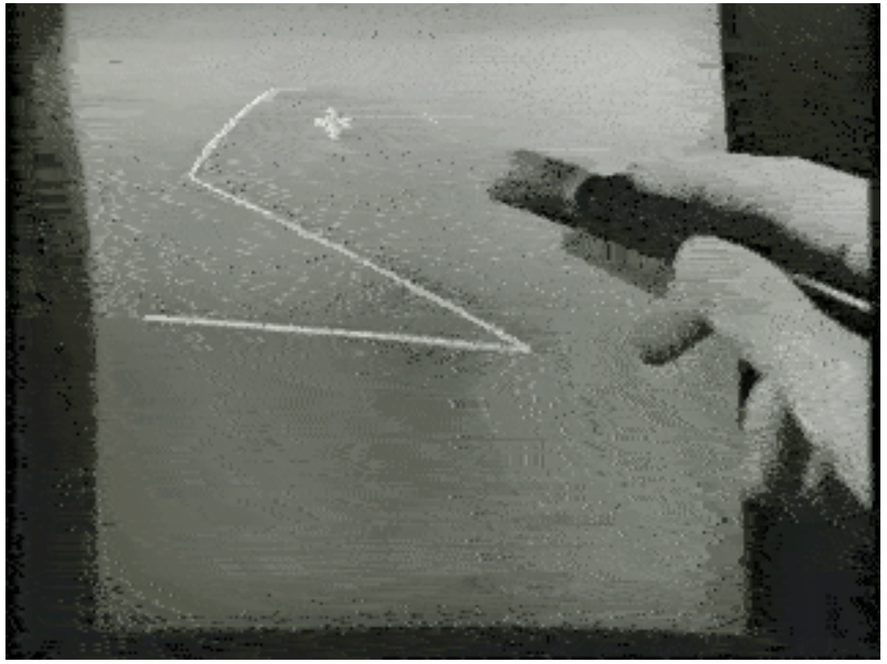




CRT monitors 1950’s–1960’s

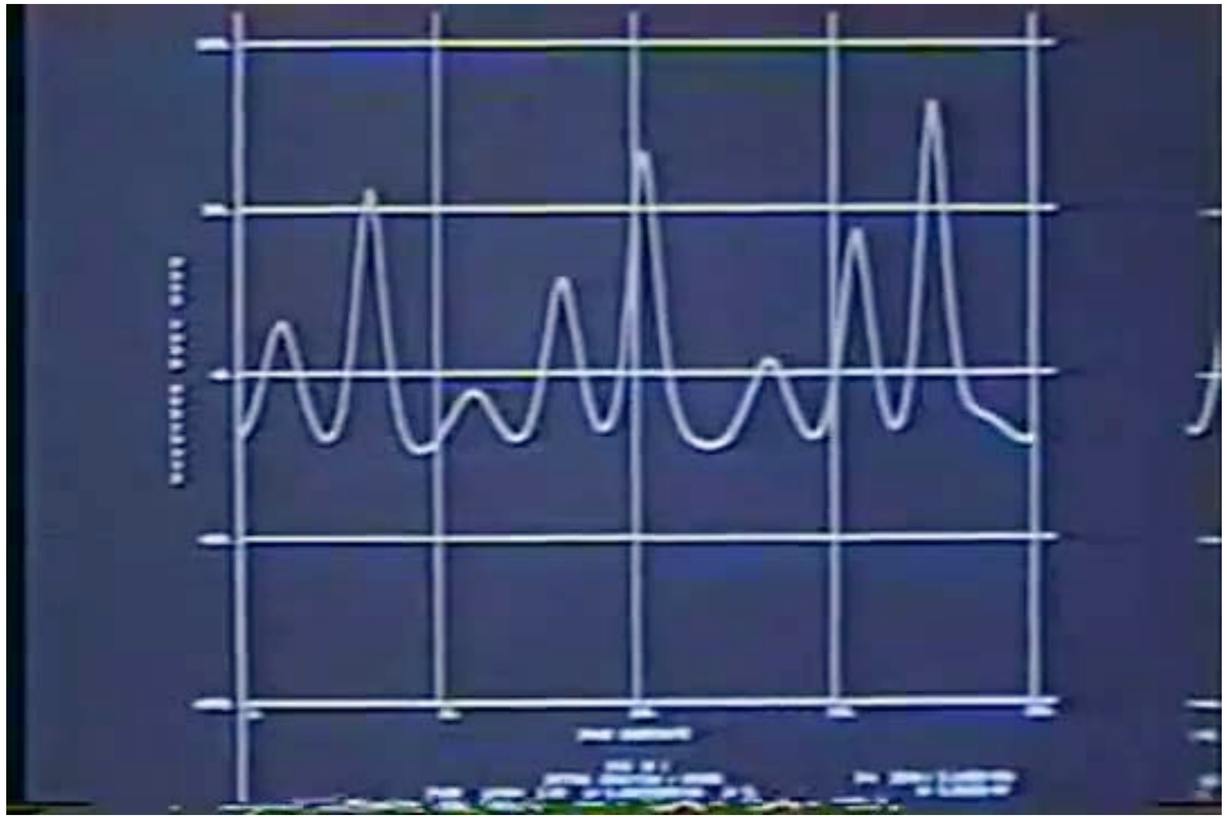
History of Graphics: Visual input





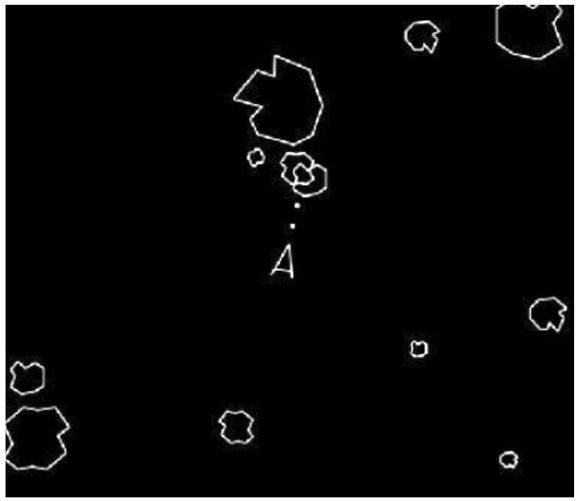
“Sketchpad” – Ivan Sutherland 1963

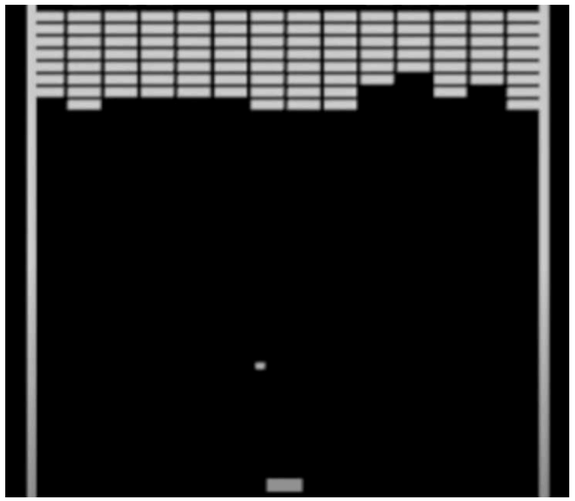
History of Graphics: Visualizing math



Discovery of “solitons” in the Korteweg–de Vries (KdV) equation by Zabusky & Kruskal in 1965 while making the above film

History of Graphics: 1970's





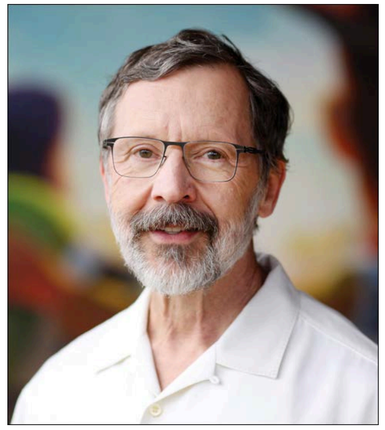


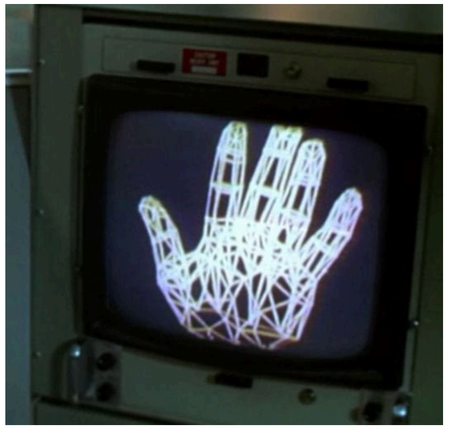
Arcade games 1970's

- Raster (pixel) graphics
- Graphic processing unit (GPU)
- Realtime

3DCG: Ed Catmull & others in Utah

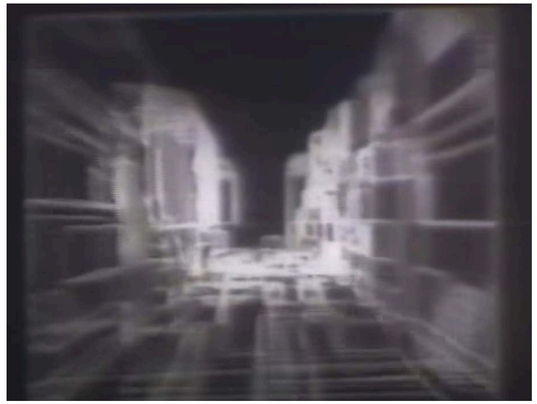
- Z-buffering
- Texture mapping
- Subdivision







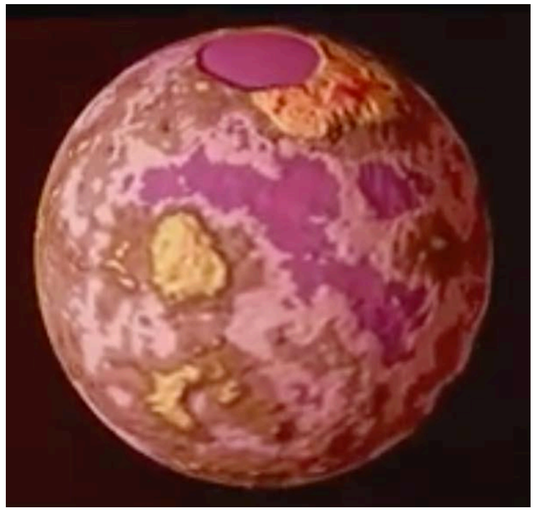




Lucasfilm *Star Wars IV* 1977

- Computer graphics in blockbusters

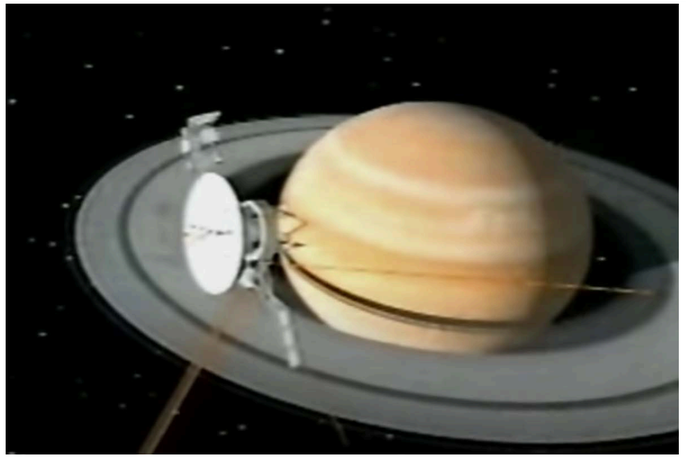
History of Graphics: Voyagers & Cosmos



Bump map for Venus 1978



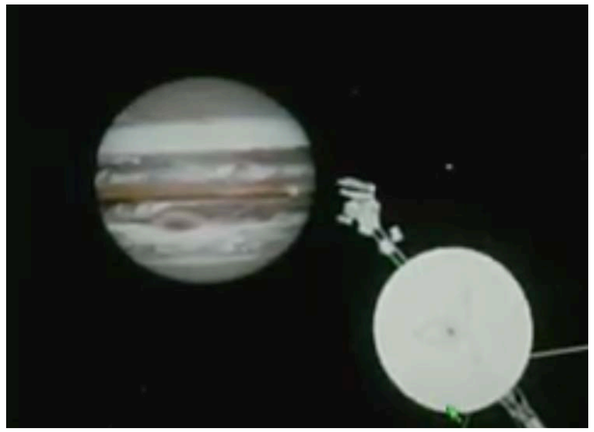
Texture reconstruction for moons of Jupiter



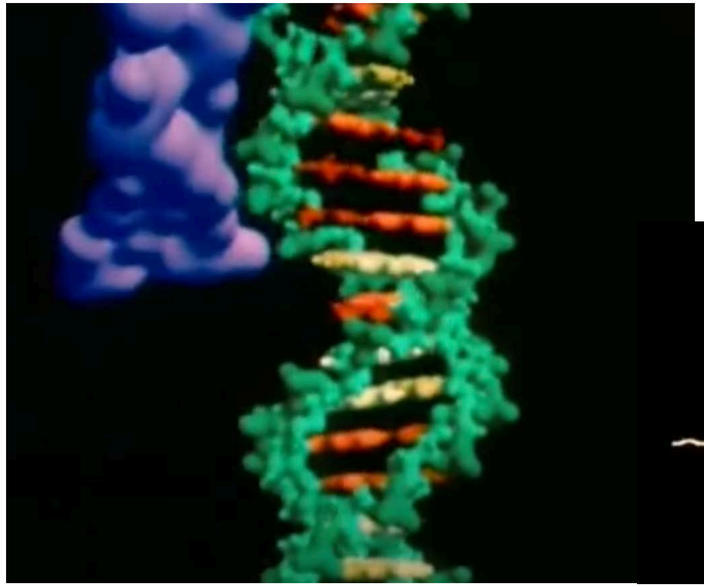
V2 Saturn flyby 1981



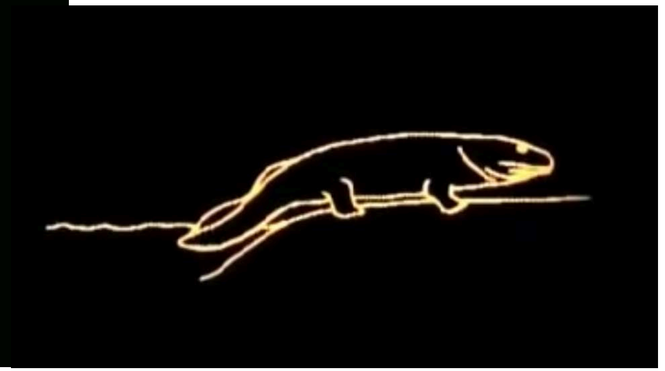
James Blinn



V1 Jupiter flyby 1979



Cosmos “DNA” 1980



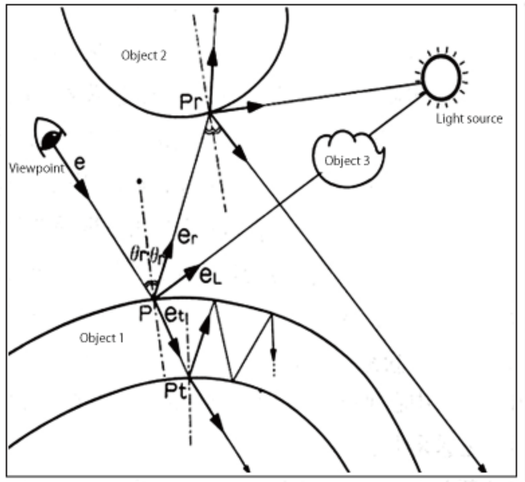
Cosmos “evolution” 1980

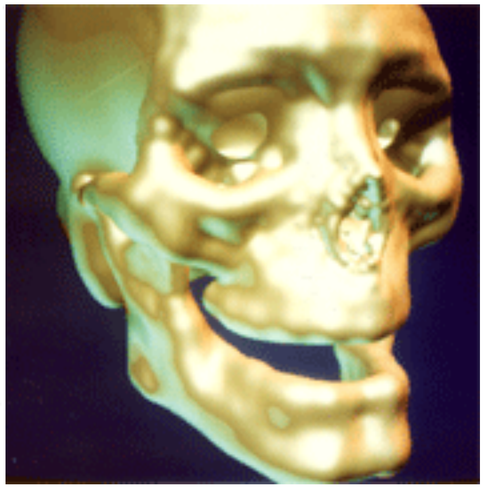
History of Graphics: 1980's





Nintendo 1981





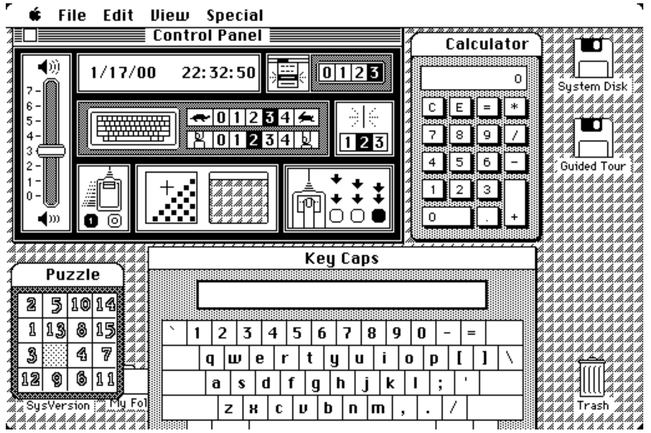
$I(P) = Io(P) + src(\theta r) \cdot I(Pr) + t \cdot I(Pt)$

LINKS-1 CG System 1983



Xerox Alto 1973





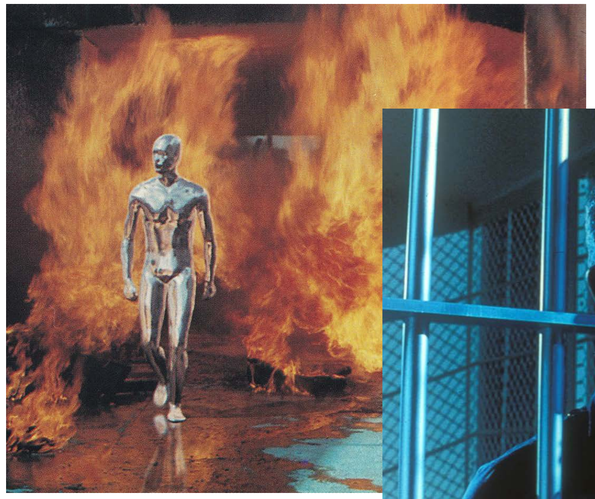
Macintosh 1984

GUI for Personal Computers



Pixar *Luxo Jr.* 1986

History of Graphics: 1990's

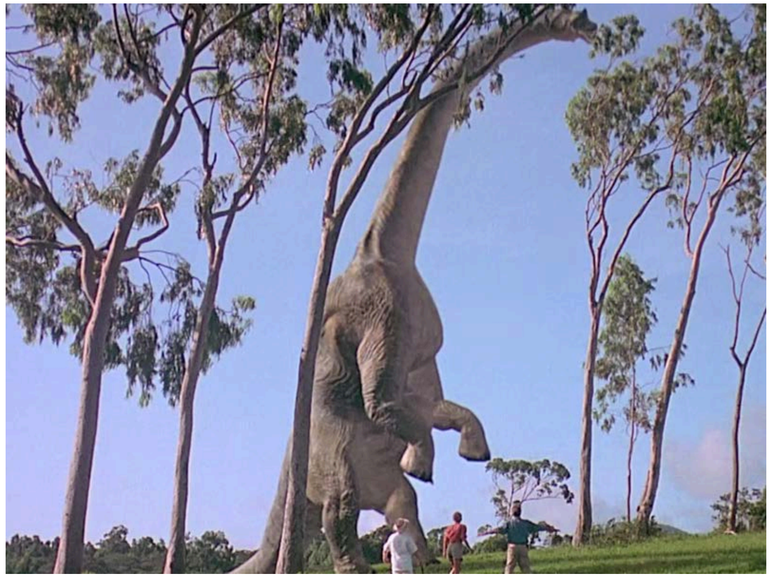




Terminator-2 1991



Beauty and the Beast 1991



Jurassic Park 1993



Toy Story 1995

History of Graphics: interface with GPU

Graphics API's 1990s

- Simple to tell GPU to draw
- Hardcoded fixed function for drawing (fast but not flexible).



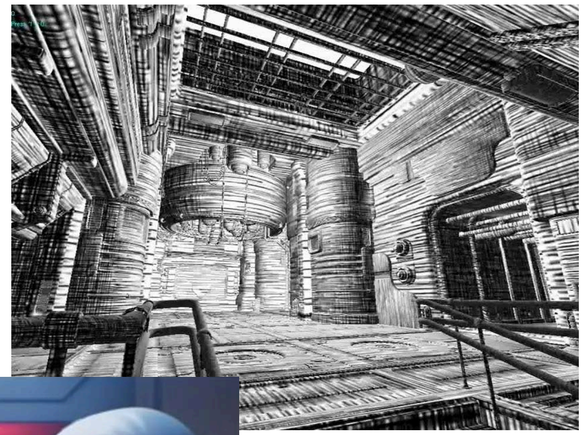
OpenGL™

Microsoft® DirectX®

Graphics API's after mid 2000–10s

- Drawing stages are programmable.
- Flexible shading.
- General purpose parallel computing.







New era of graphics 2020

- Realtime photorealism

Beyond controlling pixels

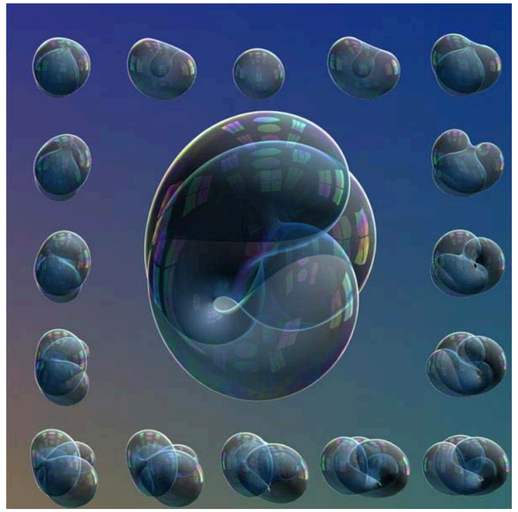
Physical simulations in entertainment

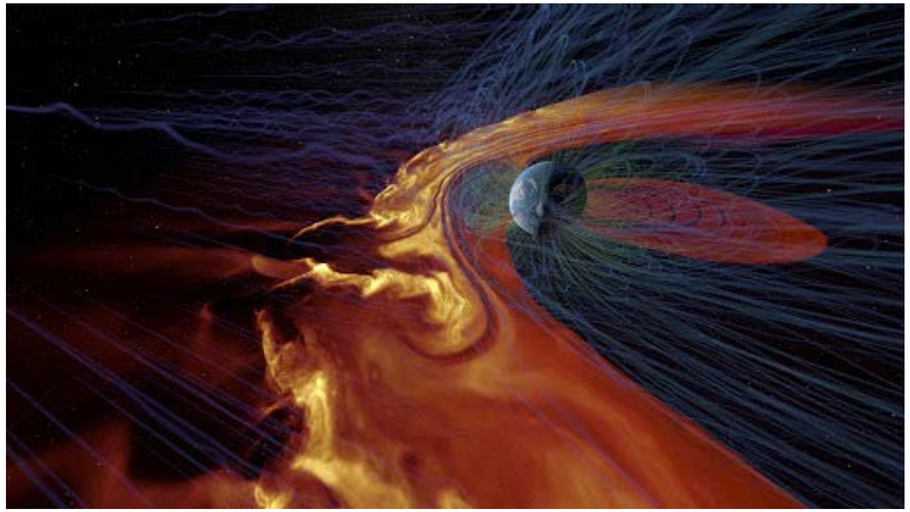


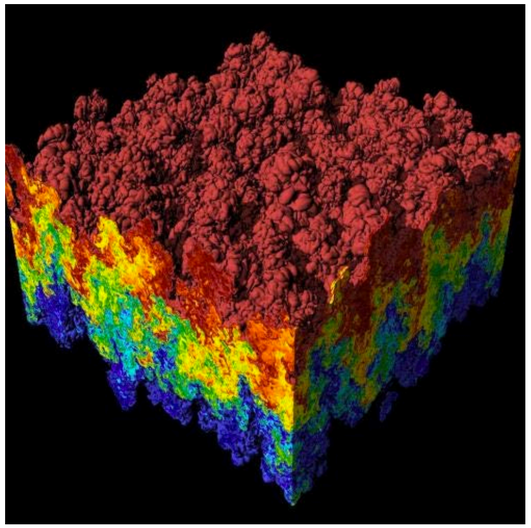


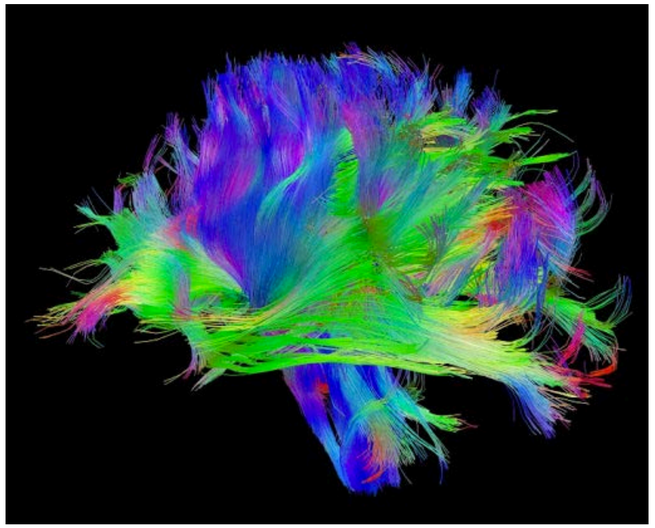
Beyond controlling pixels

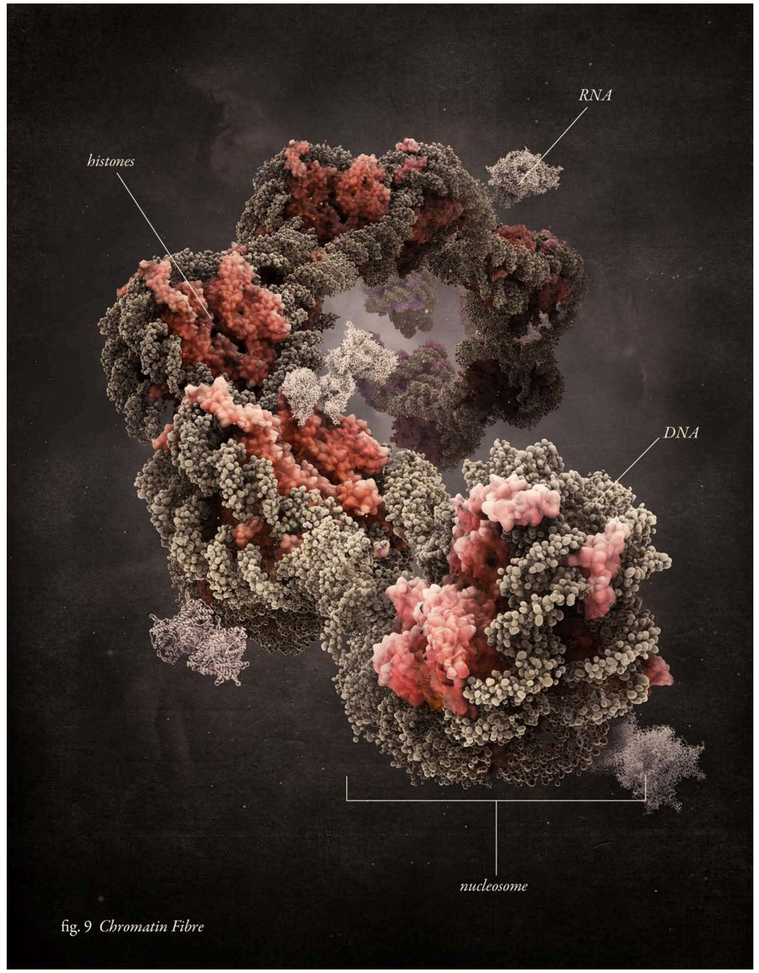
Mathematical and Scientific Visualizations





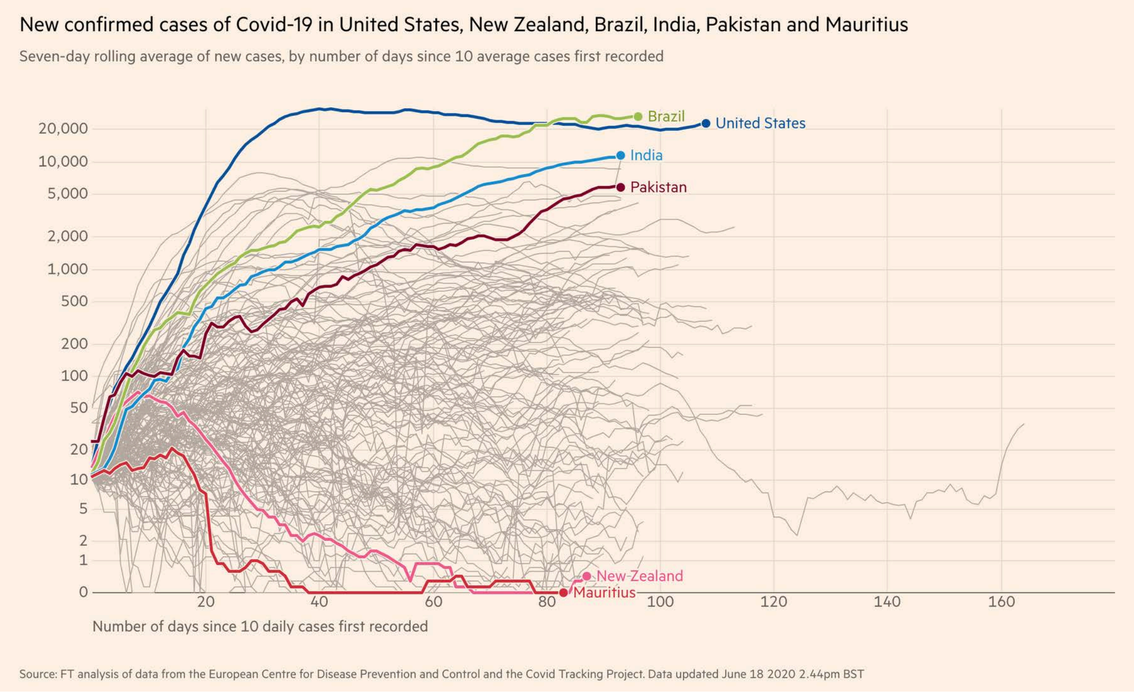


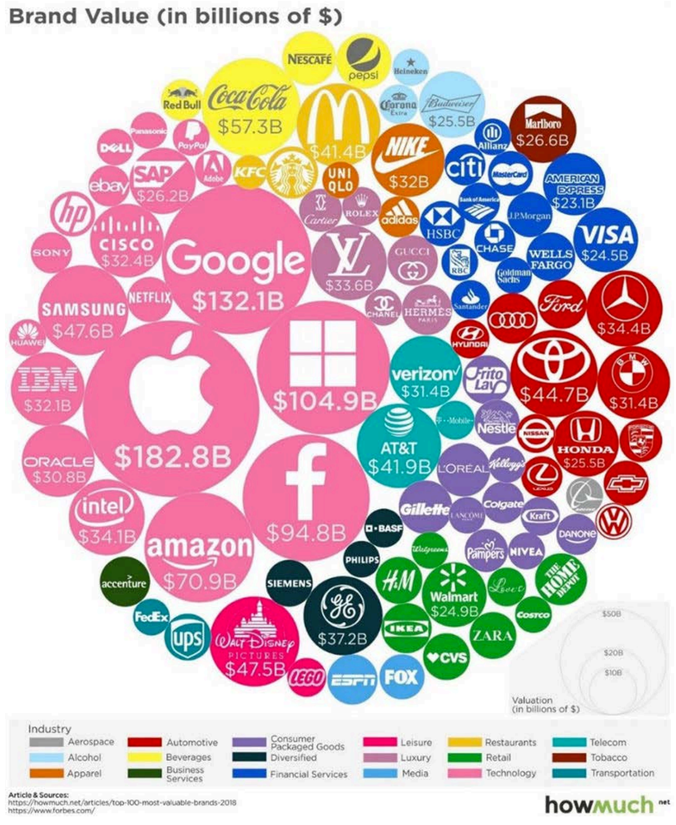


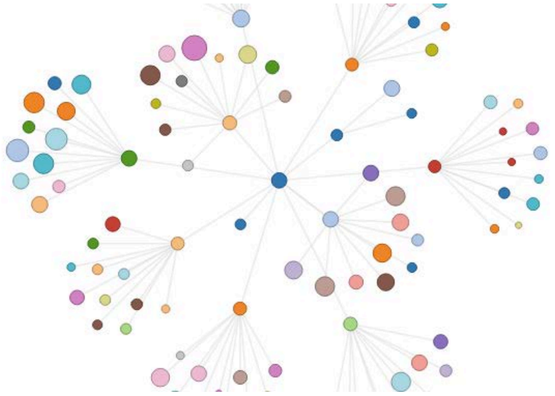


Beyond controlling pixels

Data visualizations

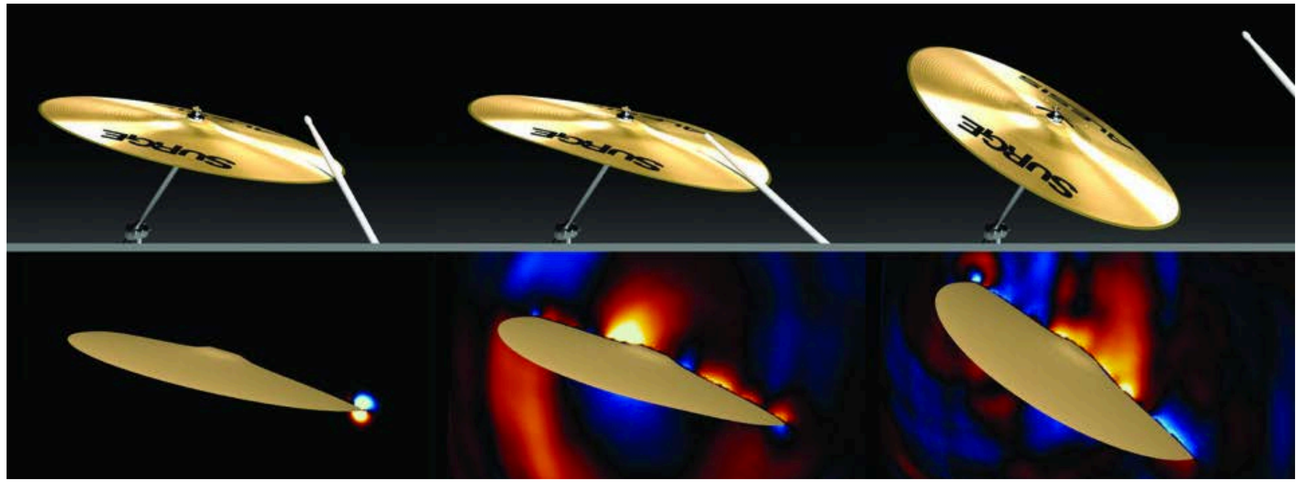






Beyond controlling pixels

Synthesize sound (not only visual information)

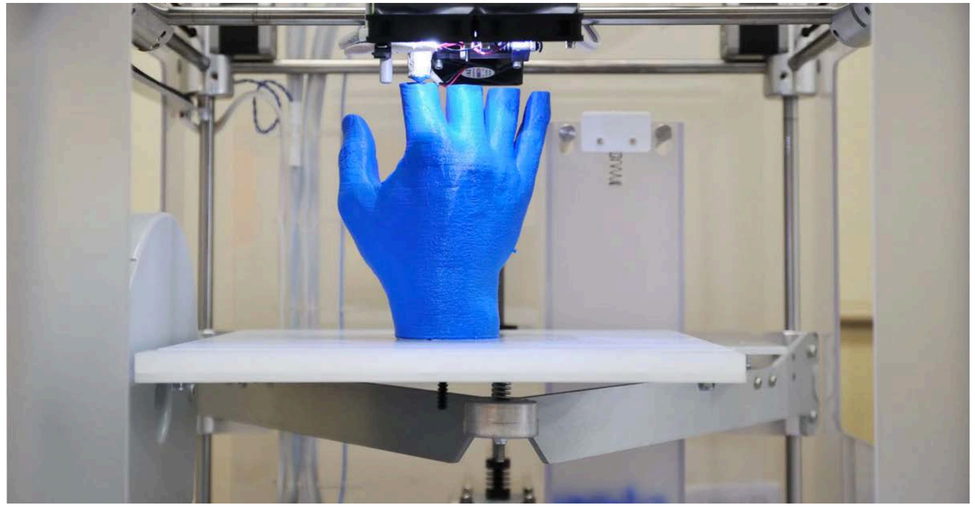


and touch senses



Beyond controlling pixels

3D printing



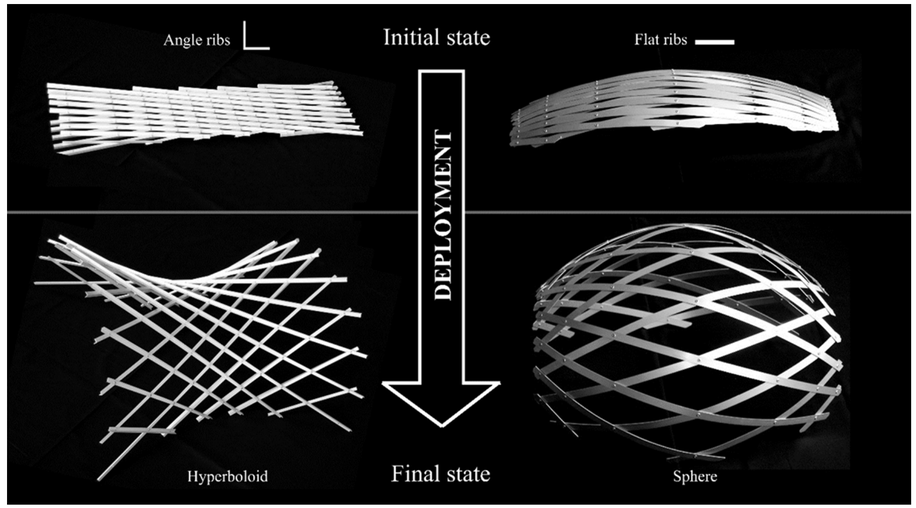
Constrained deformable geometry



architecture

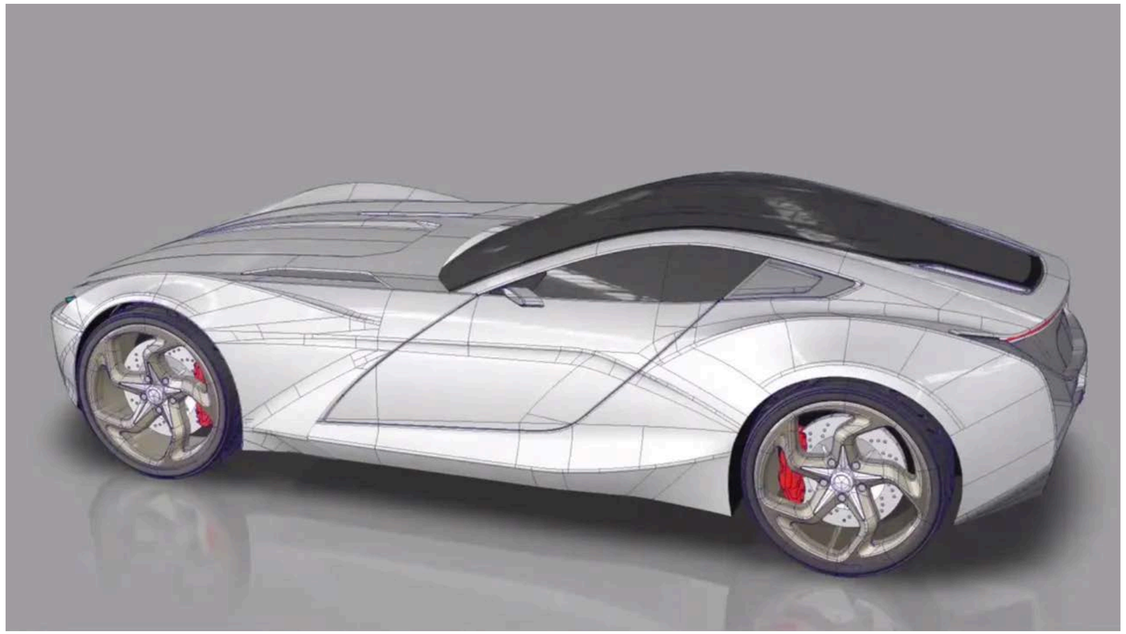






Beyond controlling pixels

Industrial computer aided design



Fonts on your screen and on newspapers



Frontier of Computer Graphics

SIGGRAPH: Annual conference since 1974 with ~20k attendees per year.

Youtube search “**SIGGRAPH technical paper trailer**”









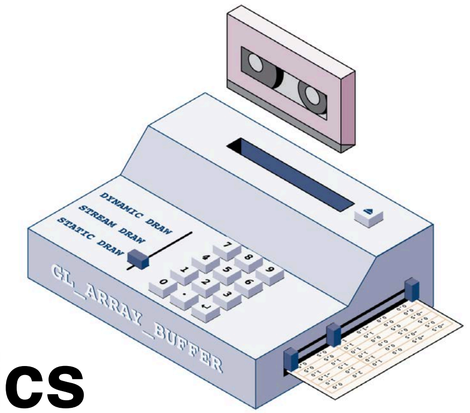
Topics of this course

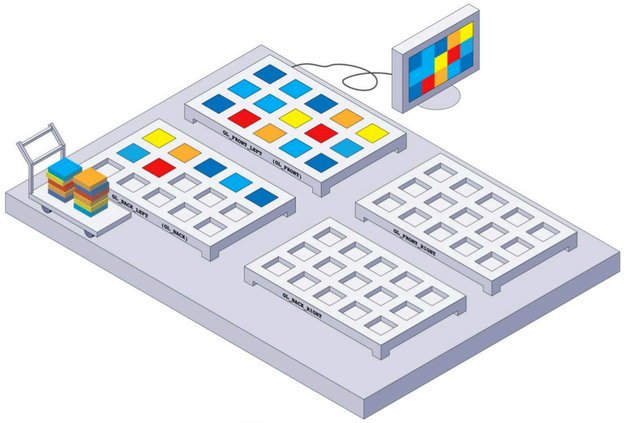
- Course logistics
- Getting started
- What is computer graphics
- This course
- How to draw a picture?

This course

Modern OpenGL

- Command the graphics card





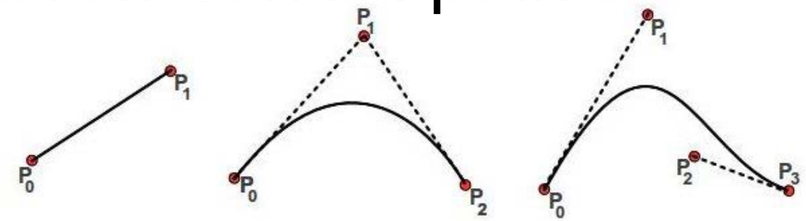
Foundation of 3D Computer Graphics

- Convert geometries in a 3D scene into pixel colors in a 2D screen.



Foundation of Vector Graphics

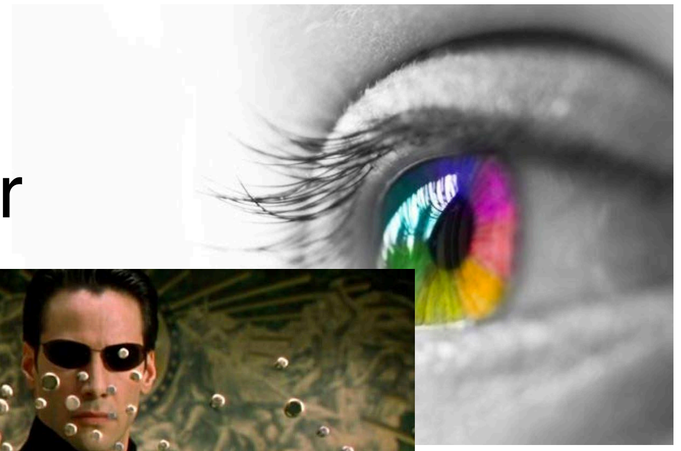
- Build smooth geometries from only a few control points



Additional topics

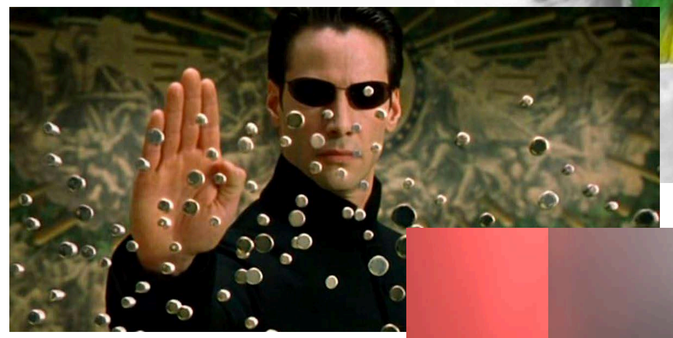
Perception of color

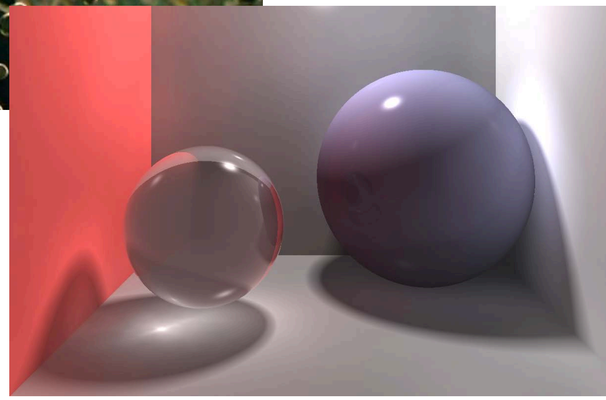
- Physical color, displayed color, perceived color



Physics-based animation

- Behind the scenes of special effects.



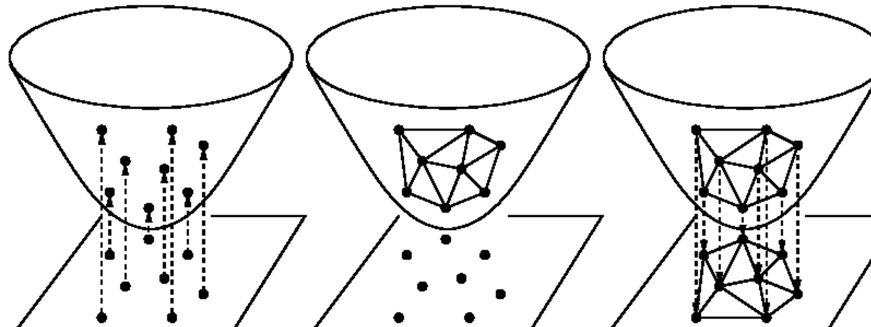


Optics

- Light transport equation

Geometry processing

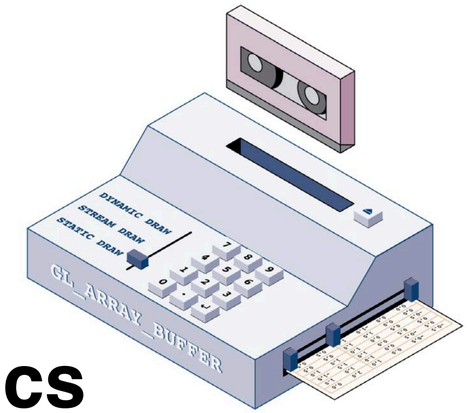
- Differential geometry of discrete meshes

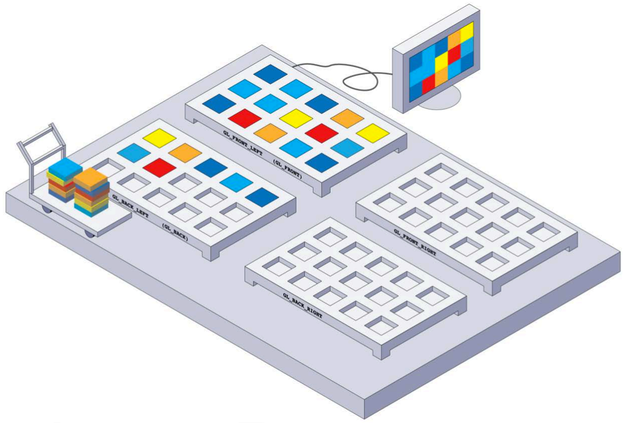


This course

Modern OpenGL

- Command the graphics card





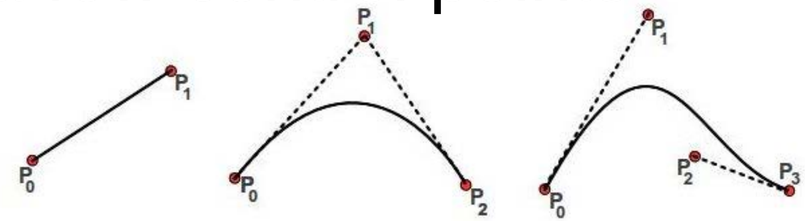
Foundation of 3D Computer Graphics

- Convert geometries in a 3D scene into pixel colors in a 2D screen.



Foundation of Vector Graphics

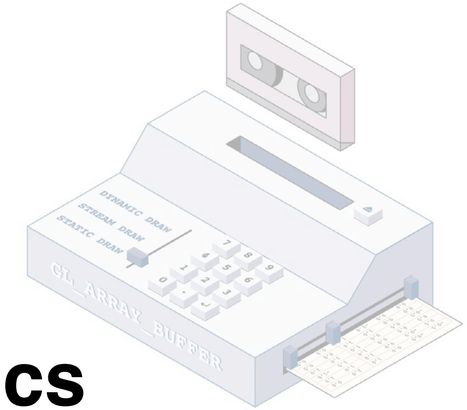
- Build smooth geometries from only a few control points



This course

Modern OpenGL

- Command the graphics card





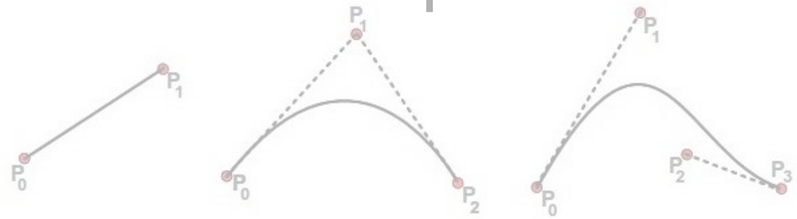
Foundation of 3D Computer Graphics

- Convert geometries in a 3D scene into pixel colors in a 2D screen.



Foundation of Vector Graphics

- Build smooth geometries from only a few control points



3D Computer Graphics

Foundation of 3D Computer Graphics

- Convert geometries in a 3D scene into pixel colors in a 2D screen.



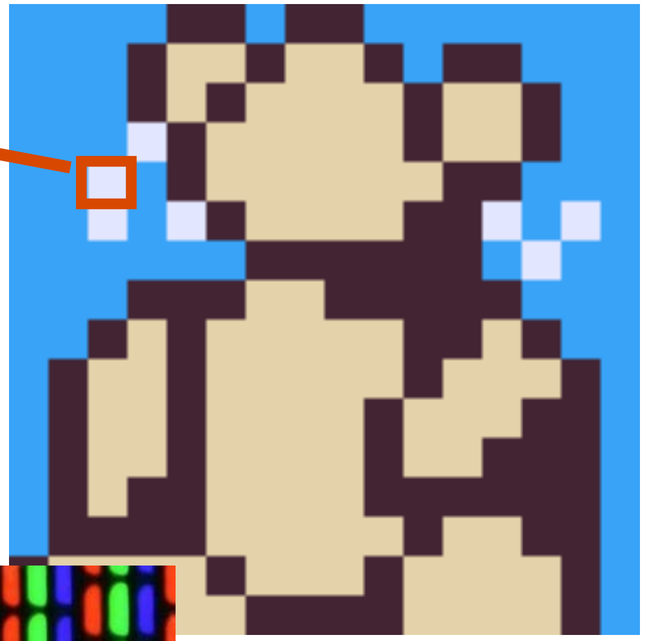
- **How to draw pictures algorithmically?**
  - Rasterization v.s. Ray tracing
  - Graphics pipeline
  - Hardware: GPU

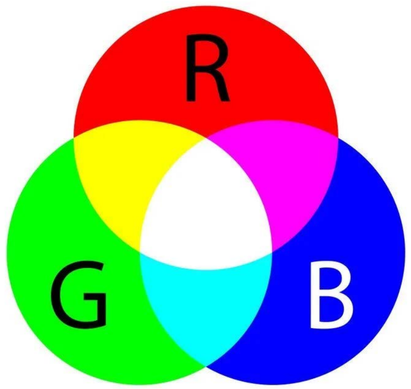
How to draw a picture?

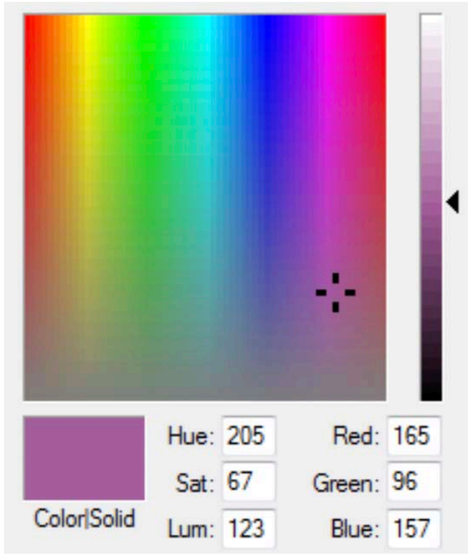
- Course logistics
- Getting started
- What is computer graphics
- This course
- How to draw a picture?

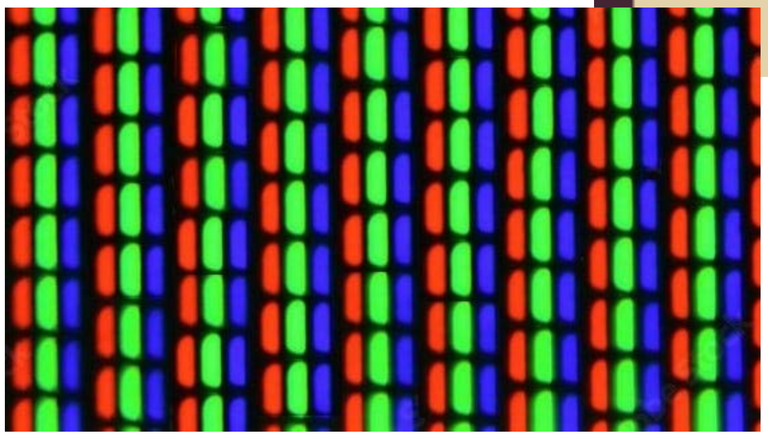
Picture

- A **picture** or an **image** is (usually) an 2D array of colored blocks
- Each colored block is called a ***picture element***, or ***pixel*** for short.
- Every color is described by 3 real numbers between 0 and 1 (or 0–255 in 8-bit integer)



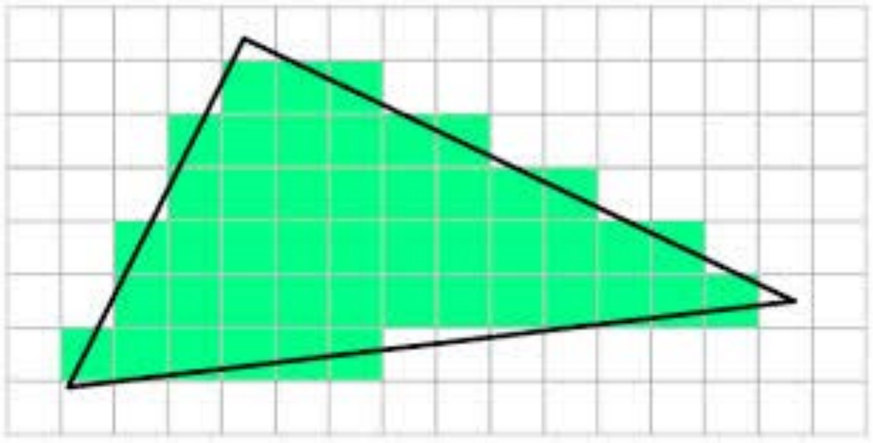






Turn a shape into a picture

- A 2D shape like line, circle, triangle can be efficiently described by a few mathematical parameters
  - Triangle: (x1,y1),(x2,y2),(x3,y3) coordinates of 3 vertices
  - Circle: (x,y; r) center and radius
- How do you turn the mathematical shape into a picture (assign color to pixels)?
  - This process is called ***rasterization***



Overlay multiple shapes

- What if we have multiple triangles to rasterize?
  - Need to rasterize each triangle in the correct order, so that the overwriting color is the top one.
  - Then how can we possibly draw the picture on the right?

