

CSE 167 (FA22)

Computer Graphics

Albert Chern

Course Logistics

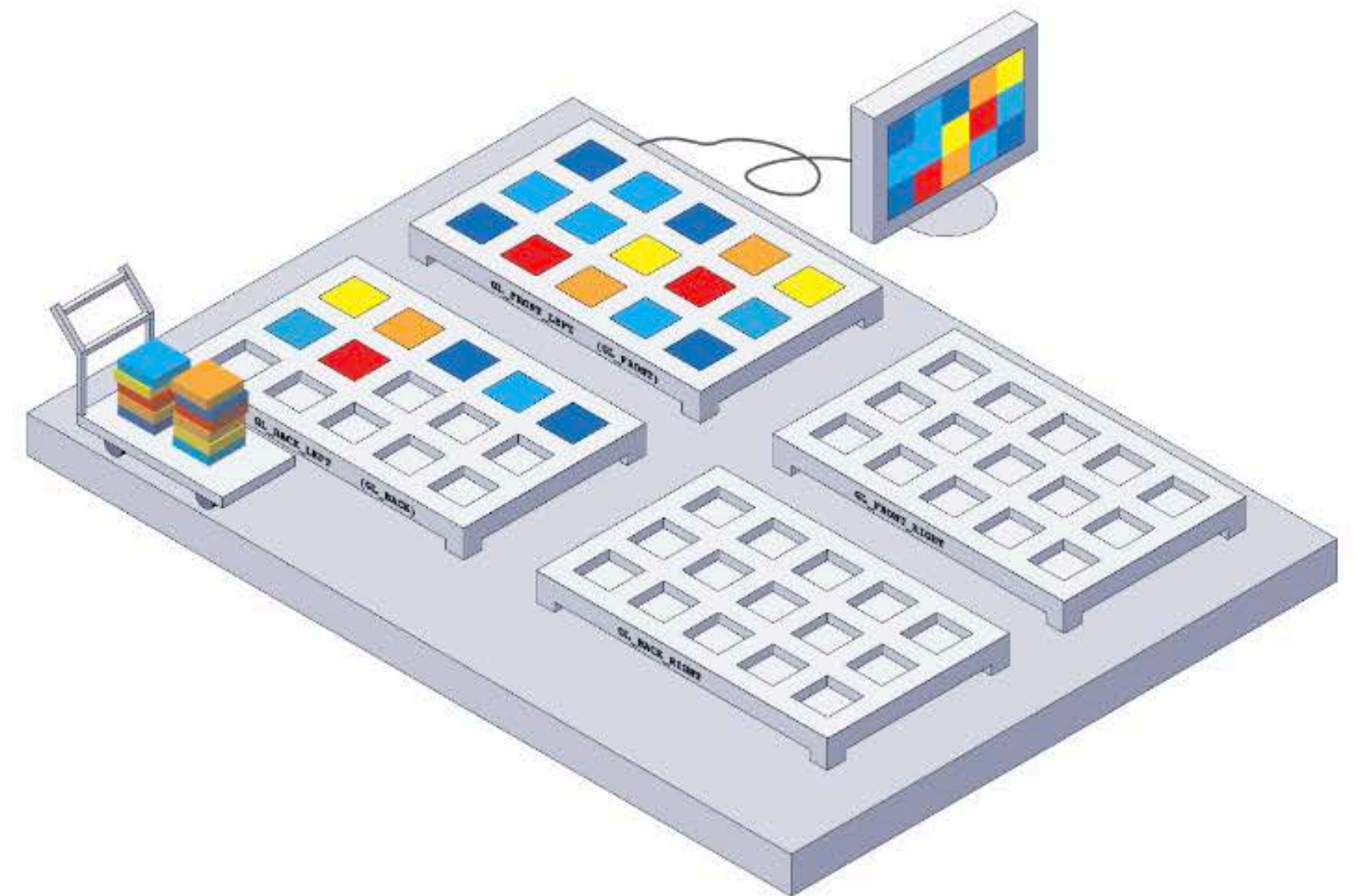
Lectures

- **Lectures (Albert Chern)**
 - ▶ Center Hall 115, In-person or remotely
- **Discussion (by the TAs)**
 - ▶ Fri 18:00–18:50, Center Hall 119
(starting from Week1: 9/30)
- **TAs**
 - ▶ Peter Wu
 - ▶ Sina Nabizadeh
 - ▶ Abhinav Gupta
 - ▶ Dylan Rowe
- **Tutors**
 - ▶ Flynn Sui
 - ▶ Evan Yao

CSE 167 (FA 2022) Computer Graphics

Welcome to CSE 167, Introduction to Computer Graphics.

- **Lecture:** Mon Wed Fri 15:00 - 15:50. [Center Hall 115](#).
- **Discussion Session:** Fri 18:00-18:50. [Center Hall 119](#).
- **Classroom:** You can also participate the class via Zoom.
- **Sites:**
 - **This page:** Slides, lecture notes, HW
 - **Canvas:** Link to Zoom (and Zoom recordings), link to Gradescope, and link back to this page.
 - **Piazza:** <https://piazza.com/ucsd/fall2022/cse167> Q&A forum for the class.
 - **Gradescope:** <https://www.gradescope.com/courses/444876> (log in with School credential, entry code: K3N5V7) HW submission.
- **Lecture note:** [Introduction to Computer Graphics](#)
Visit Canvas (for all info including Zoom links), Gradescope (HW submission) and Piazza (Q&A).



		Office hour	OH location (all in CSE Building (EBU3B))	OS
Instructor	Albert Chern	Tue 3pm–4pm	4112	Mac
TA	Abhinav Gupta	Wed & Fri 8:30am–9:30am	B270A	Mac(Arm), Linux
TA	Mohammad "Sina" Nabizadeh	Tue 1pm–2pm	B215	Windows, Mac, Linux
TA	Dylan Rowe	Mon 10am–12pm	B275	Mac
TA	Baichuan "Peter" Wu	Thu 2pm–3pm & Fri 10am– 11am	B240A	Windows, Mac, Linux
Tutor	Ziyan "Flynn" Sui	Fri 2pm–3pm	B215	Mac(Arm)
Tutor	Yunchao "Evan" Yao	Mon 5pm–6pm	B240A	Windows, Linux

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	

```
29 int main(int argc, char** argv)
30 {
31     /* Begin Create Window */
32     glutInit(&argc, argv);
33
34     #ifdef __APPLE__
```

Chapter 2. OpenGL Setup

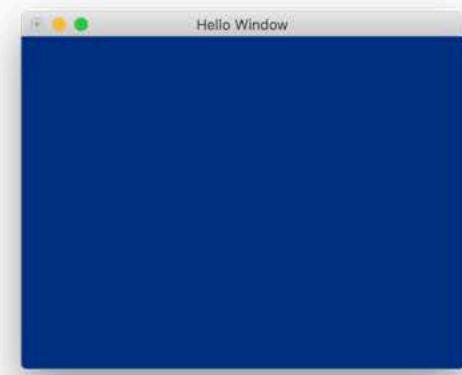


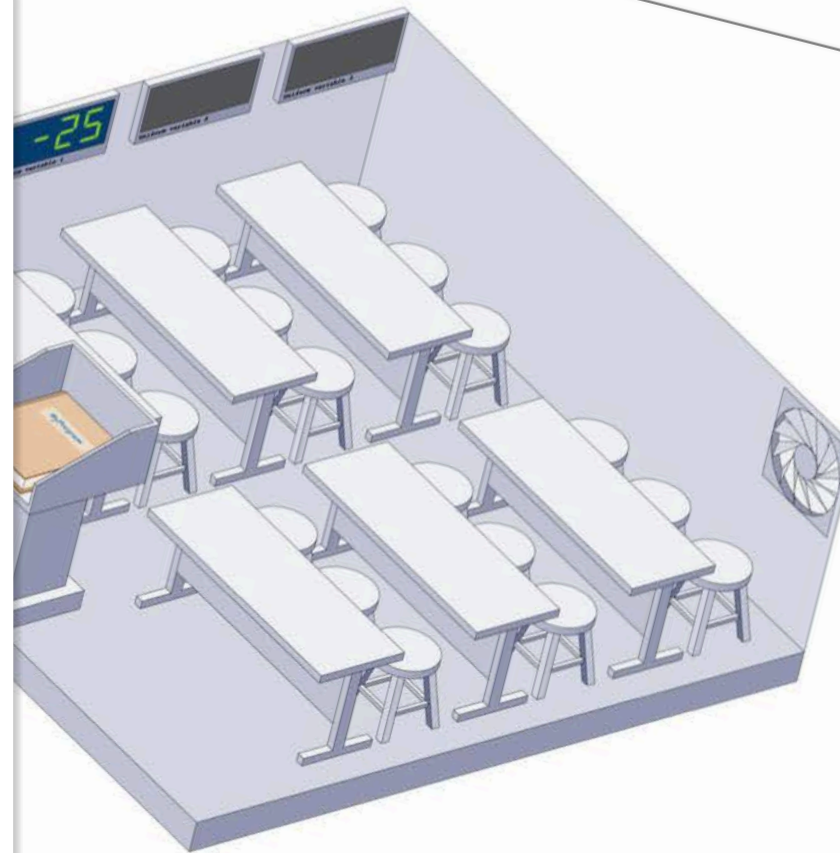
Figure 2.2 Result of Code 2.3.

Code 2.3 HelloGL.cpp

```
#ifndef GL_MULTI_GL_VERSION_HEADERS_INCLUDED
```

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

3.1 A tour of the graphics factory



graphics processing unit (GPU) in the graphics factory.

Each fragment is endowed with customized vertex shader. It consists of fixed coordinate values for each triangle from each pixel. The output of the vertex shader is not programmable. To do so, they follow a set of instructions provided by the vertex shader, which is a program executed by our own vertex shader, we can ask the GPU to compute the fragment color for each fragment. The output of the fragment shader is also written to the fragment color. It is always the same function for each fragment. The rasterizer is gauged into.

Projective geometry

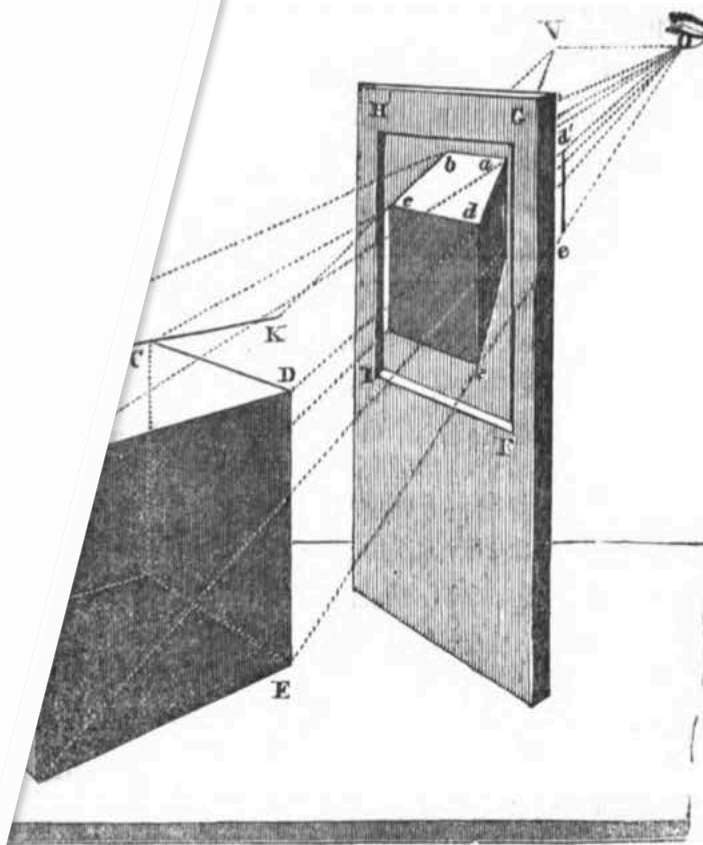


Figure 8.4 Taylor's Principles of Linear Perspective (1715).

rays passing through one point, called the eye, that are the image of the objects. We can take a plane (canvas) to intercept the lines to form the image. The light to come from the Picture to the spectator's Eye in the very same way as if it would come from the objects themselves." The artistic/Desarguesian notions of vanishing points, infinity, etc., happen in this canvas. This canvas would be the eye's projective plane. The geometric theories in the projective planes (2D) are understood as a shadow casted from the higher-dimensional (3D) but standard

Homogeneous coordinates

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

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

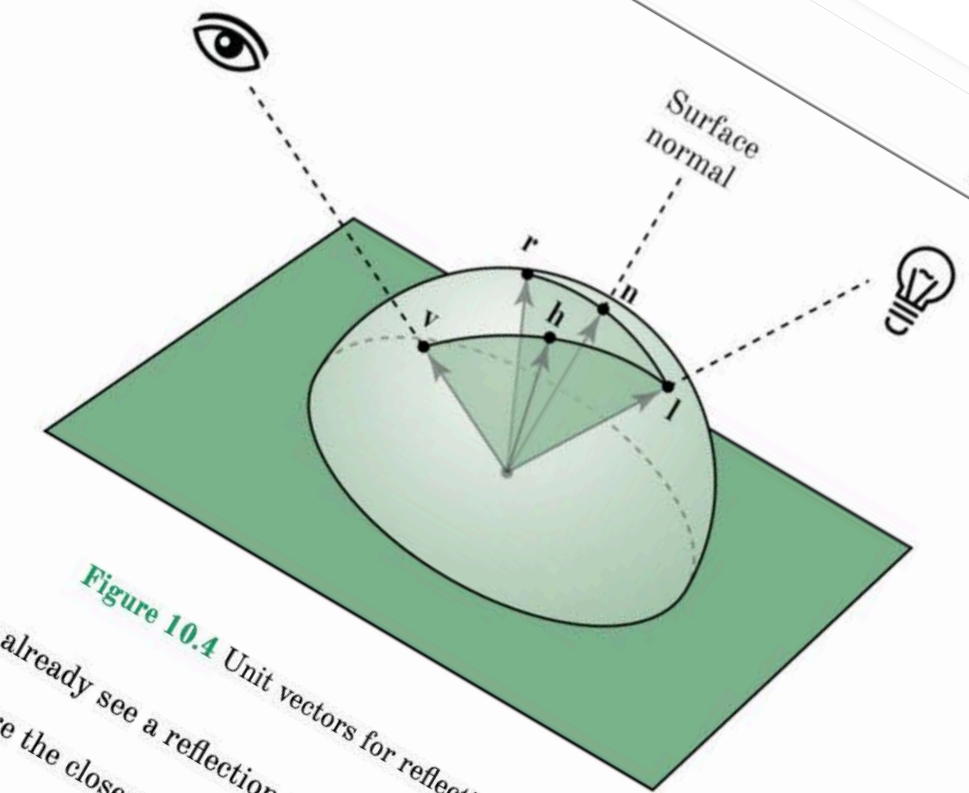


Figure 10.4 Unit vectors for reflection models. An eye already see a reflection when v is close to the reflected vector of I . To ensure the closeness is to compute the true reflected vector of I : $r := 2(I \cdot n)n - I$. This is called Phong reflection model. The closeness in terms of the dot product $v \cdot r$. This is called Blinn-Phong specular reflection model. The reflection direction r is replaced by $n \cdot h$.

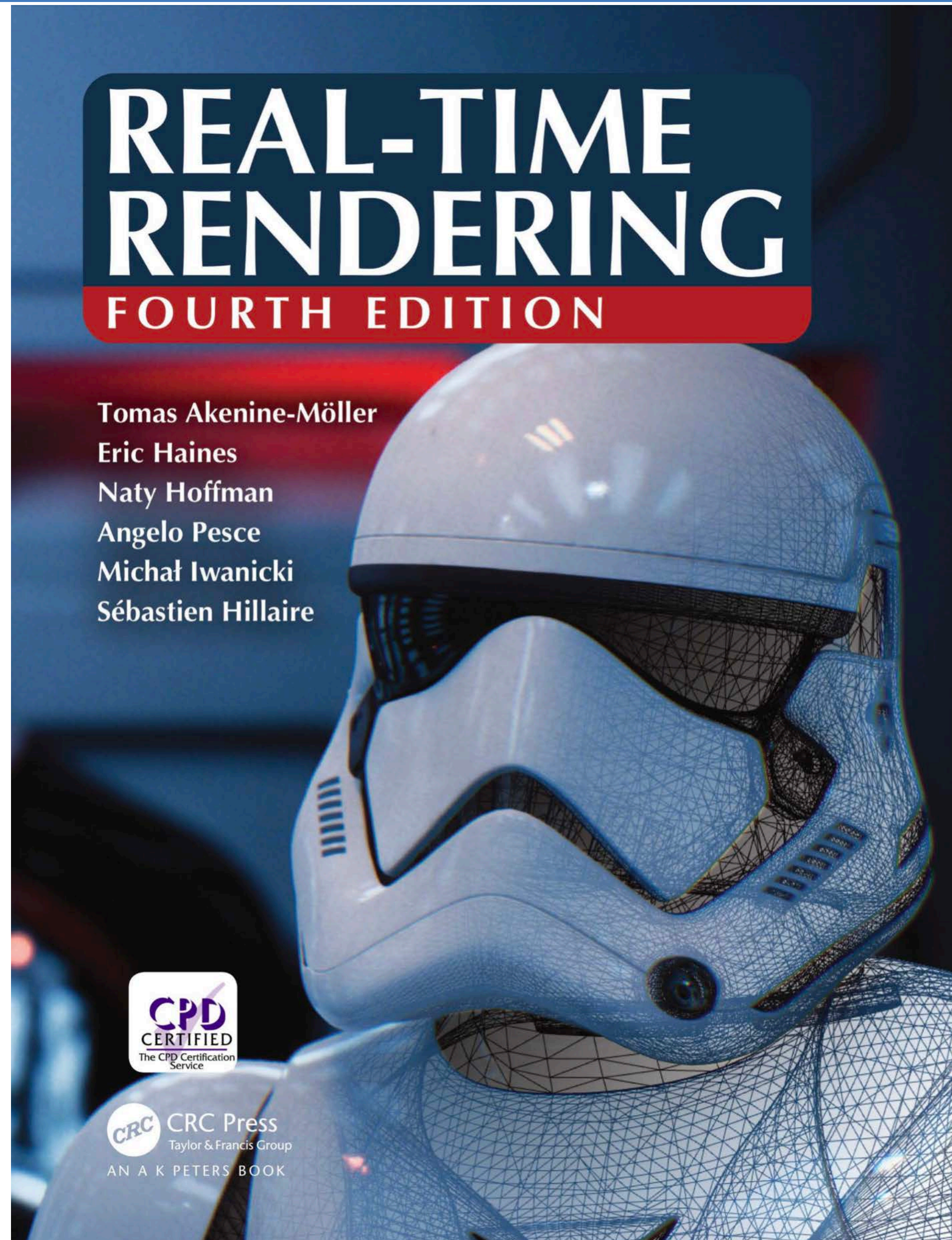
$$r = \frac{v + I}{|v + I|}$$

$$(10.16)$$

$$(10.17)$$

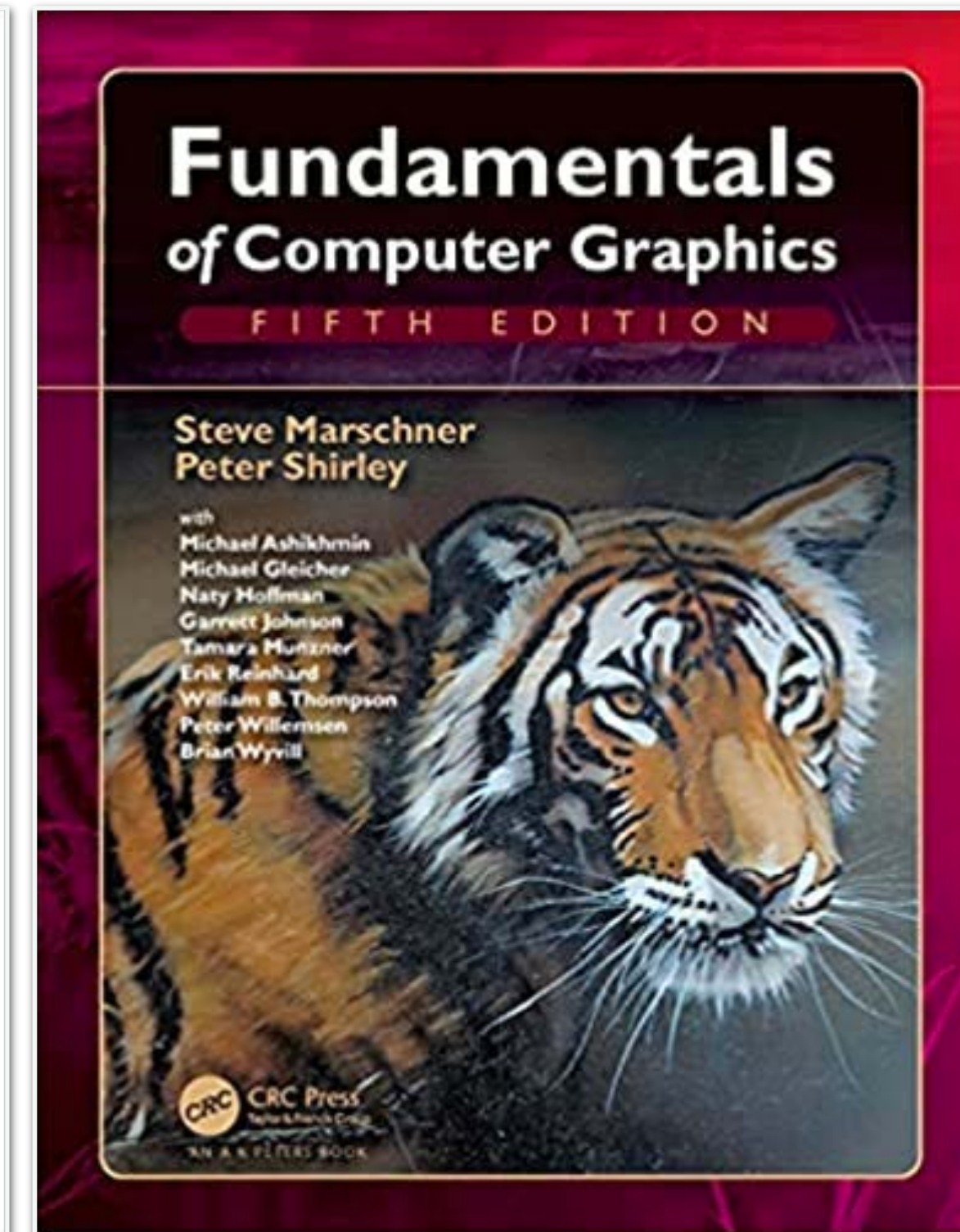
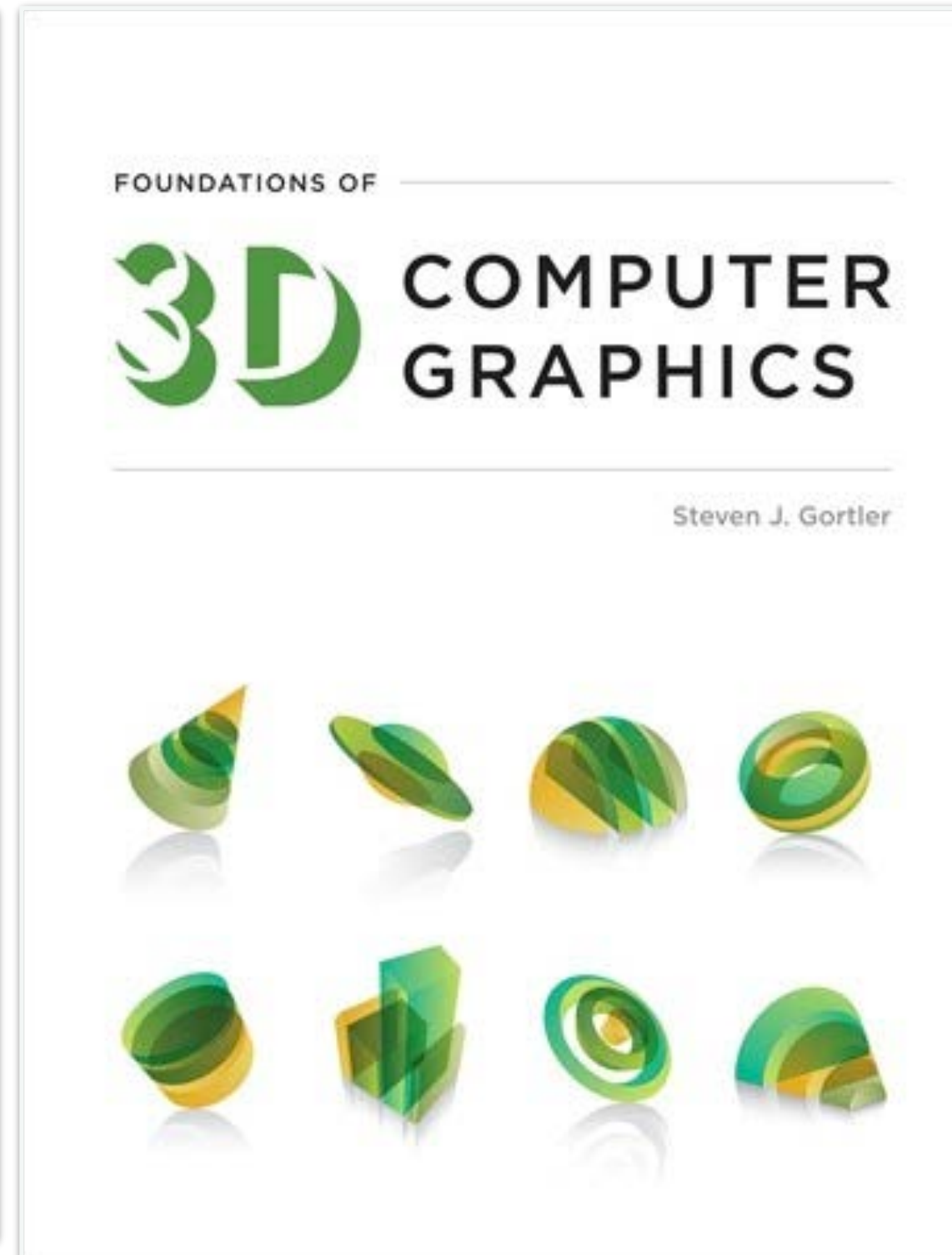
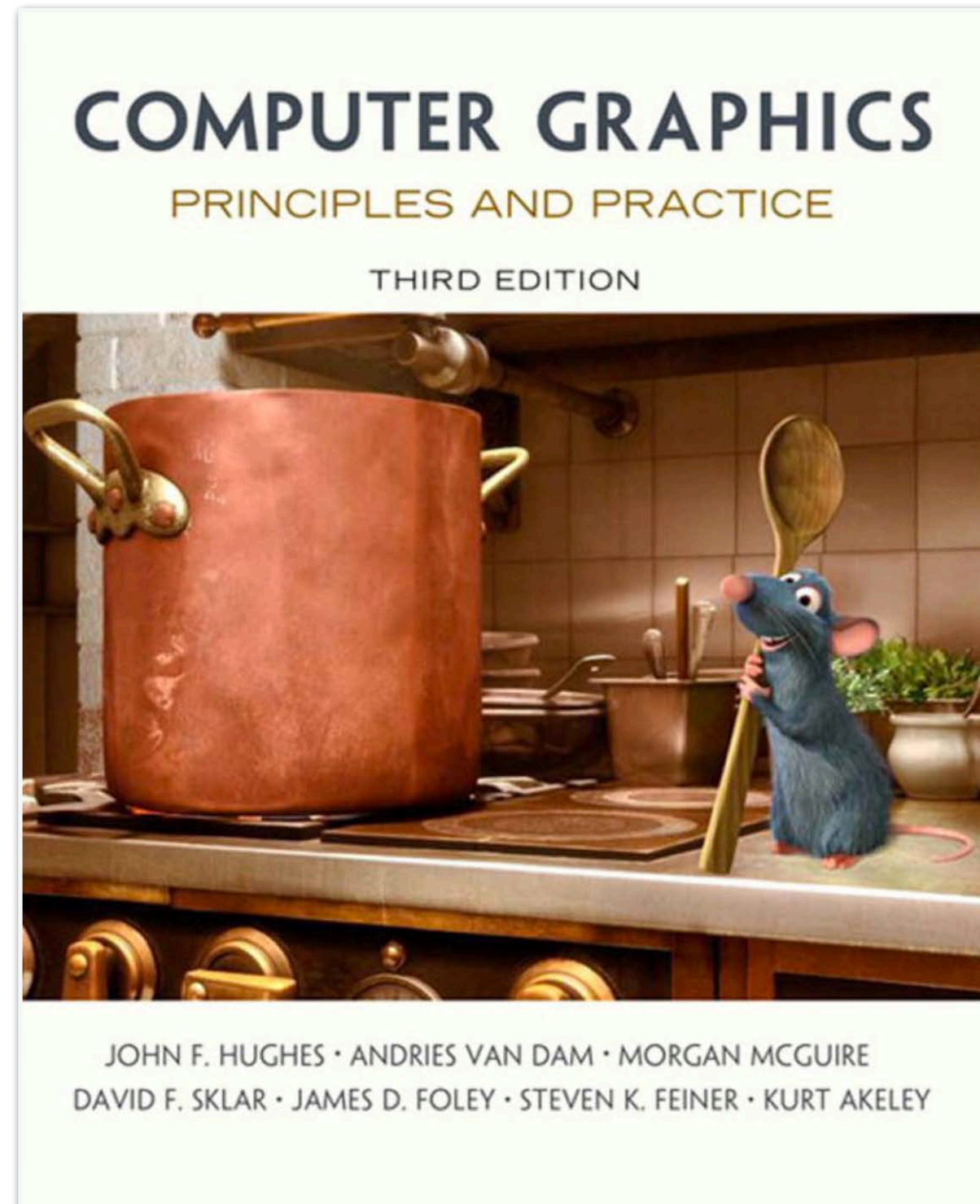
vector in the direction of the angle bisector. The Phong model comes from the fact that when projective geometry, v, I, h are relatively constant and n is fixed by a constant, which is computed by the eye. It will have to be non-constant because the closeness measurement $\max(n \cdot h)$ for the same reason as that in

References

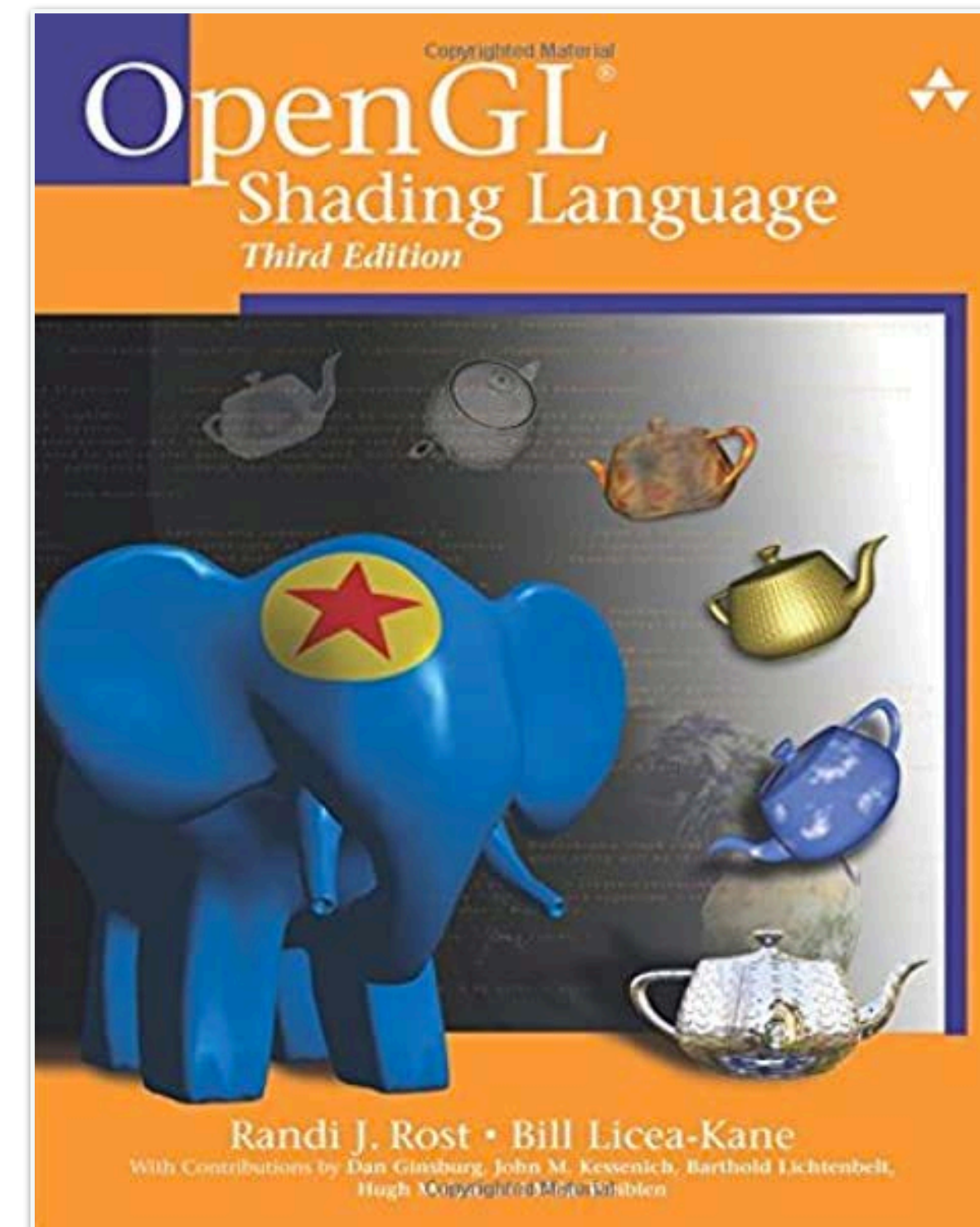
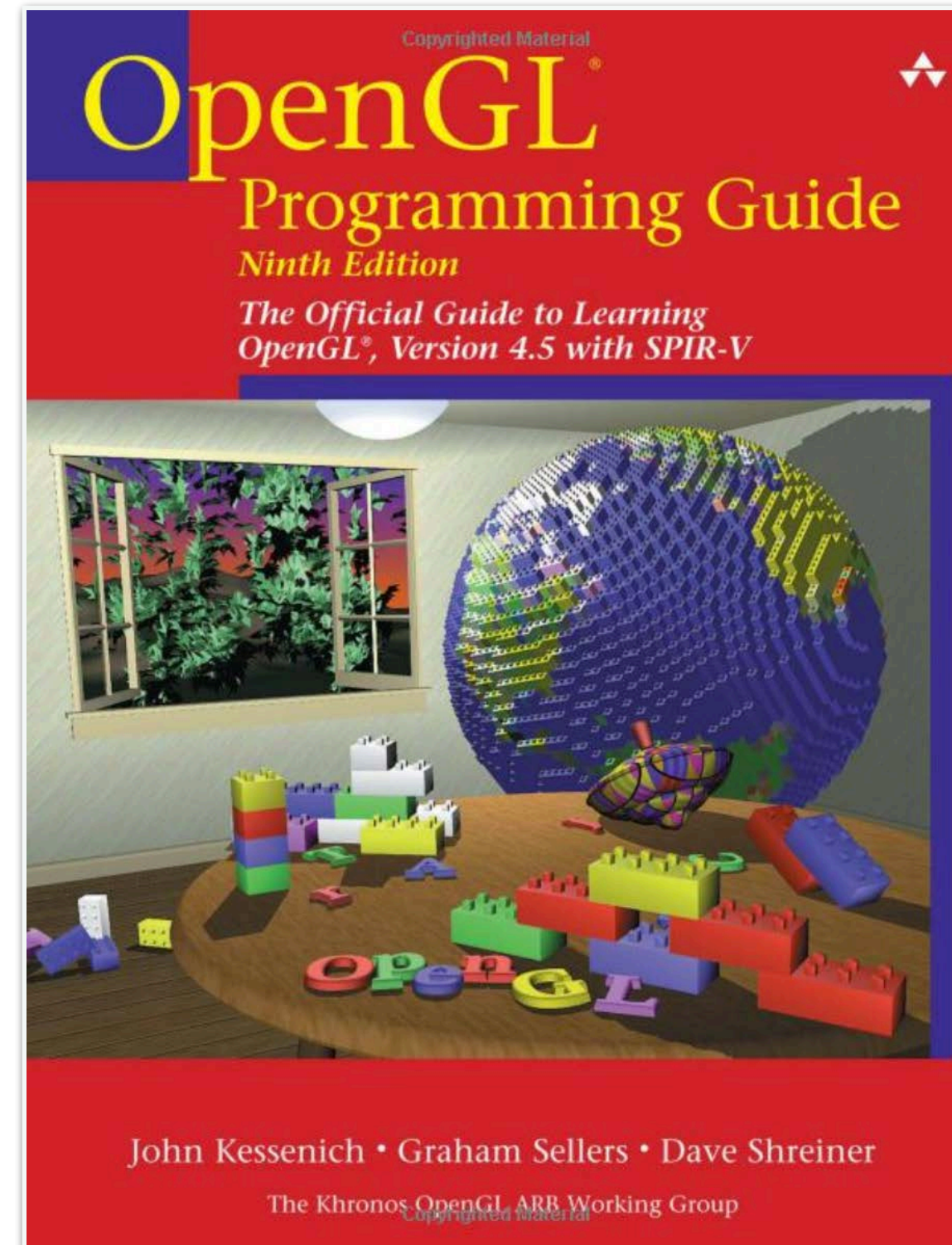


(Available online
from UC Library)

References



References



References

<https://learnopengl.com>

- Introduction
- Getting started
 - OpenGL
 - Creating a window
 - Hello Window
 - Hello Triangle
 - Shaders
 - Textures
 - Transformations
 - Coordinate Systems
 - Camera
 - Review

- Lighting
- Model Loading
- Advanced OpenGL
- Advanced Lighting
- PBR
- In Practice
- Guest Articles
- Code repository
- Translations
- About

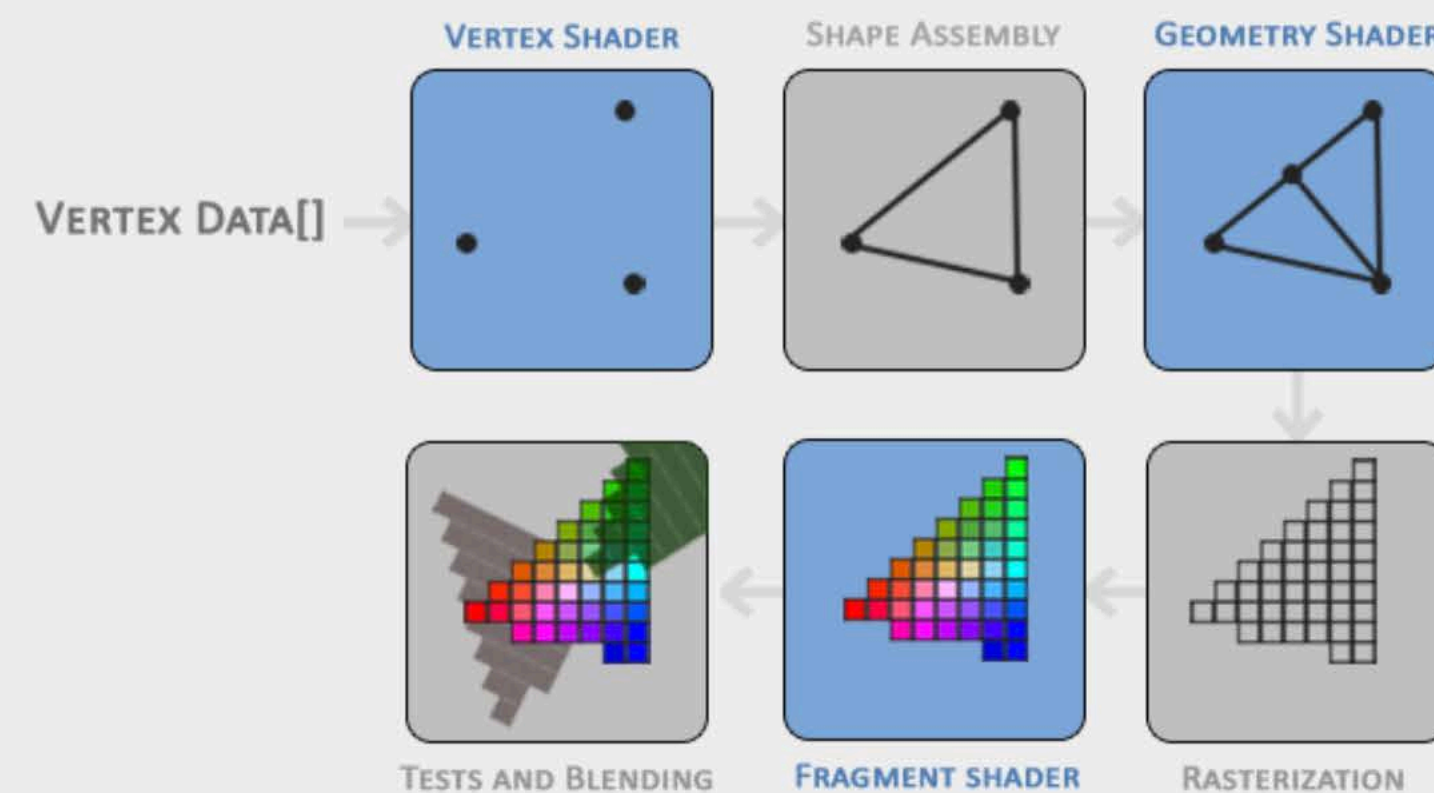
PRINT EDITION

about transforming all 3D coordinates to 2D pixels that fit on your screen. The process of transforming 3D coordinates to 2D pixels is managed by the **graphics pipeline** of OpenGL. The graphics pipeline can be divided into two large parts: the first transforms your 3D coordinates into 2D coordinates and the second part transforms the 2D coordinates into actual colored pixels. In this chapter we'll briefly discuss the graphics pipeline and how we can use it to our advantage to create fancy pixels.

The graphics pipeline takes as input a set of 3D coordinates and transforms these to colored 2D pixels on your screen. The graphics pipeline can be divided into several steps where each step requires the output of the previous step as its input. All of these steps are highly specialized (they have one specific function) and can easily be executed in parallel. Because of their parallel nature, graphics cards of today have thousands of small processing cores to quickly process your data within the graphics pipeline. The processing cores run small programs on the GPU for each step of the pipeline. These small programs are called **shaders**.

Some of these shaders are configurable by the developer which allows us to write our own shaders to replace the existing default shaders. This gives us much more fine-grained control over specific parts of the pipeline and because they run on the GPU, they can also save us valuable CPU time. Shaders are written in the **OpenGL Shading Language (GLSL)** and we'll delve more into that in the next chapter.

Below you'll find an abstract representation of all the stages of the graphics pipeline. Note that the blue sections represent sections where we can inject our own shaders.



Grades

- **No quiz or exam**
- **Weekly written exercise 40%** *individual work*
- **Programming HW 60%**
 - ▶ HW0–4: 40% *individual work*
 - ▶ Final HW: 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.

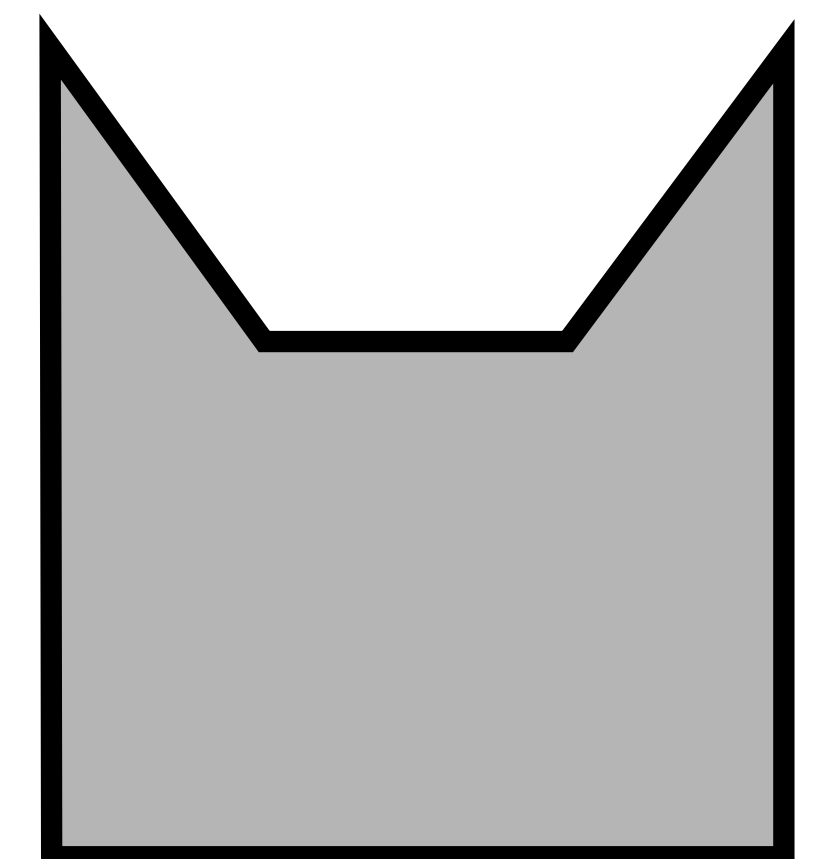
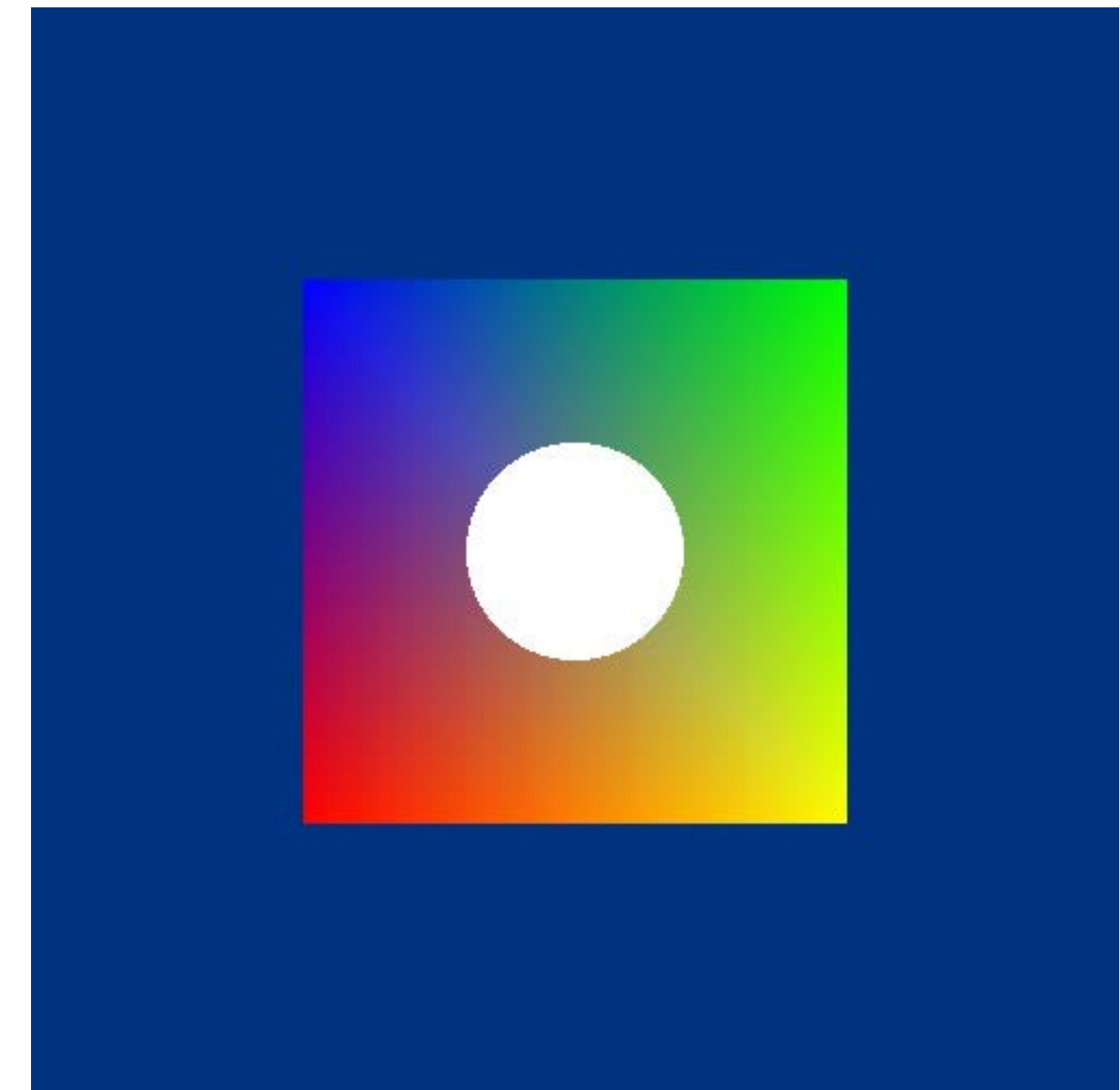
Prerequisite and some expectation

- **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)

Getting Started

Getting started

- Next week, we will start working with **OpenGL**
- HW0 & Exer1 are due 9/30
 - ▶ Compile, run, and upload the result (should look like this Fig).
 - ▶ Modify the code to get the cat/fox shape
 - ▶ Compilation problem: office hour (see TA/tutor's platform) or piazza.
 - ▶ Next week, we explain what happens in the code.



Today

Computer Graphics

Computer Graphics

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

What is Computer Graphics?

What is Computer Graphics?

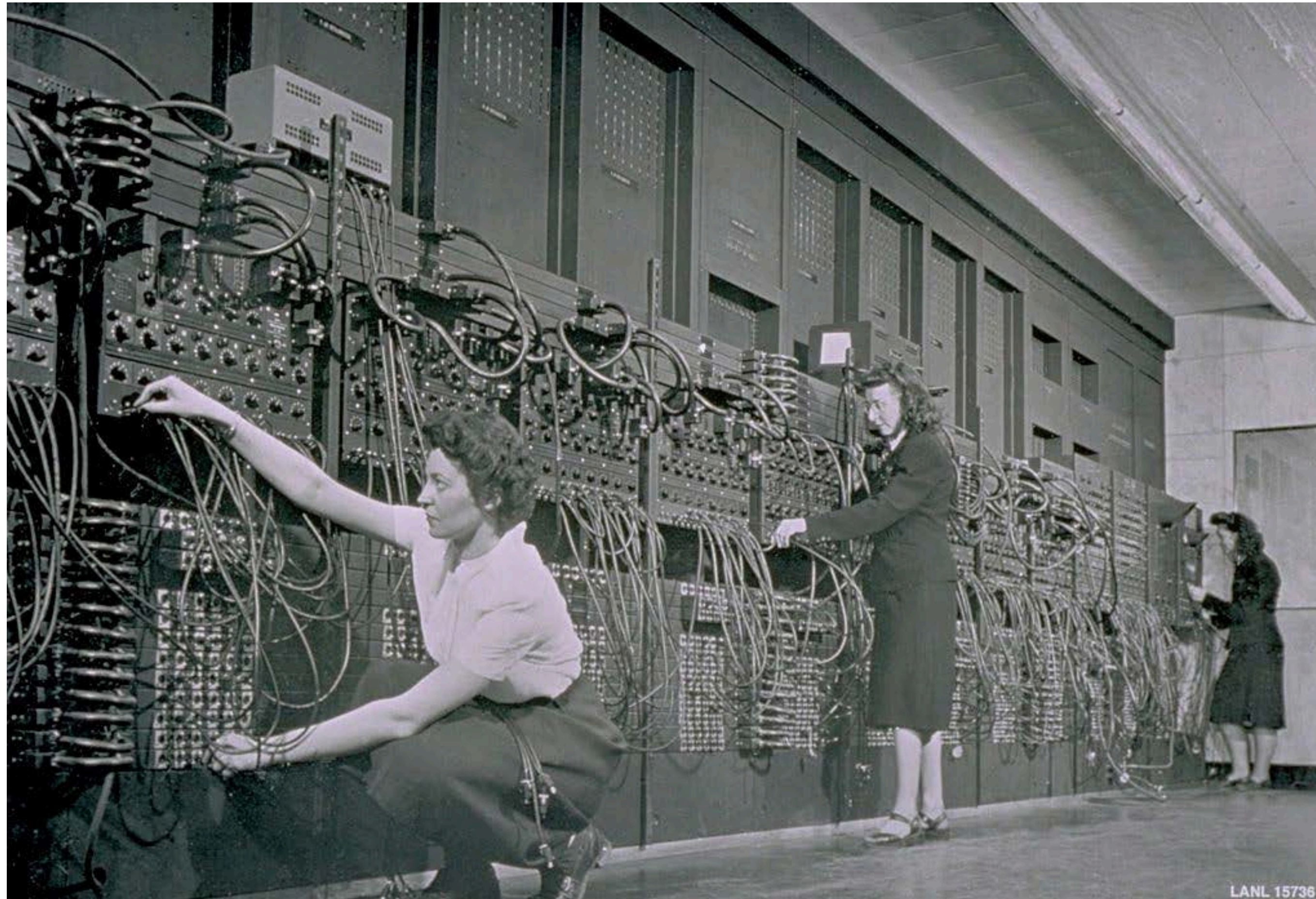


Jurassic Park 1993



Ratatouille 2007

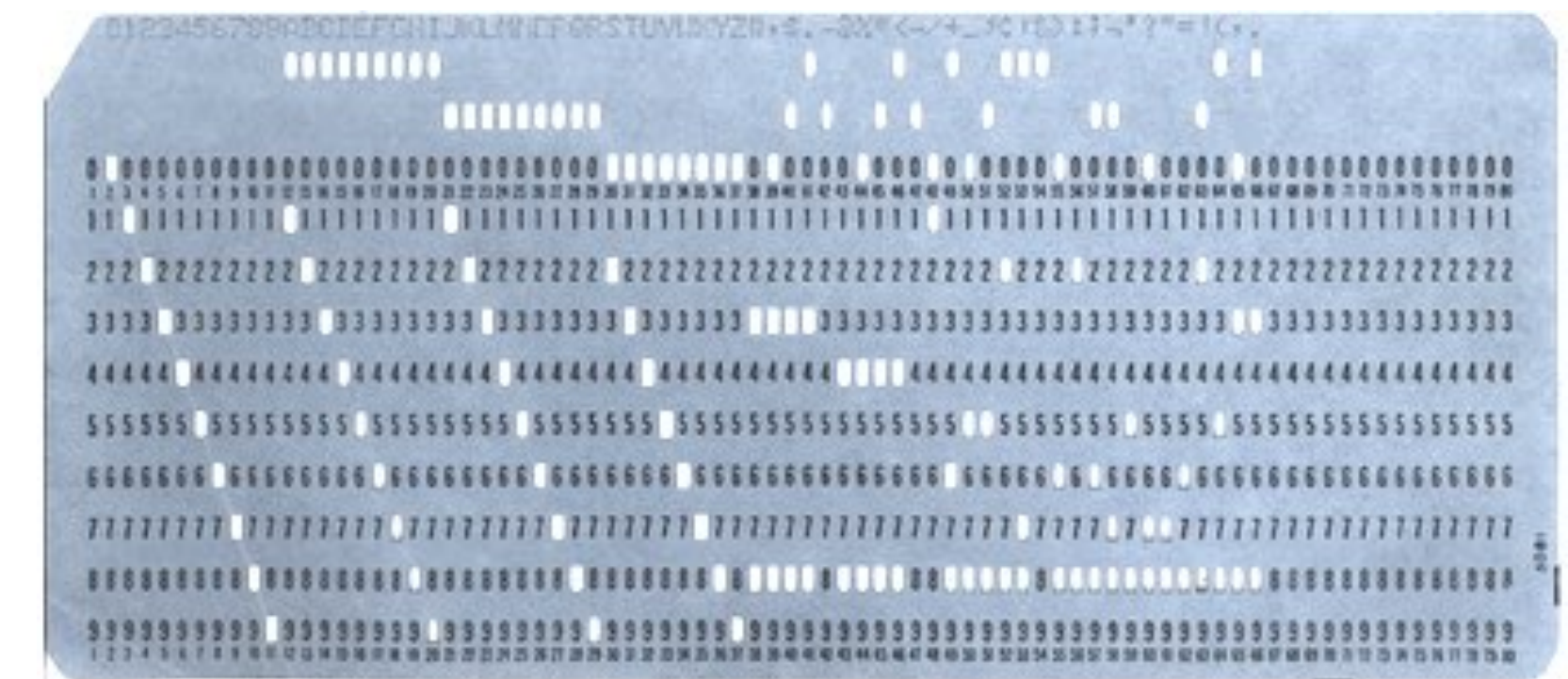
Why Computer Graphics?



early computer (ENIAC) 1945

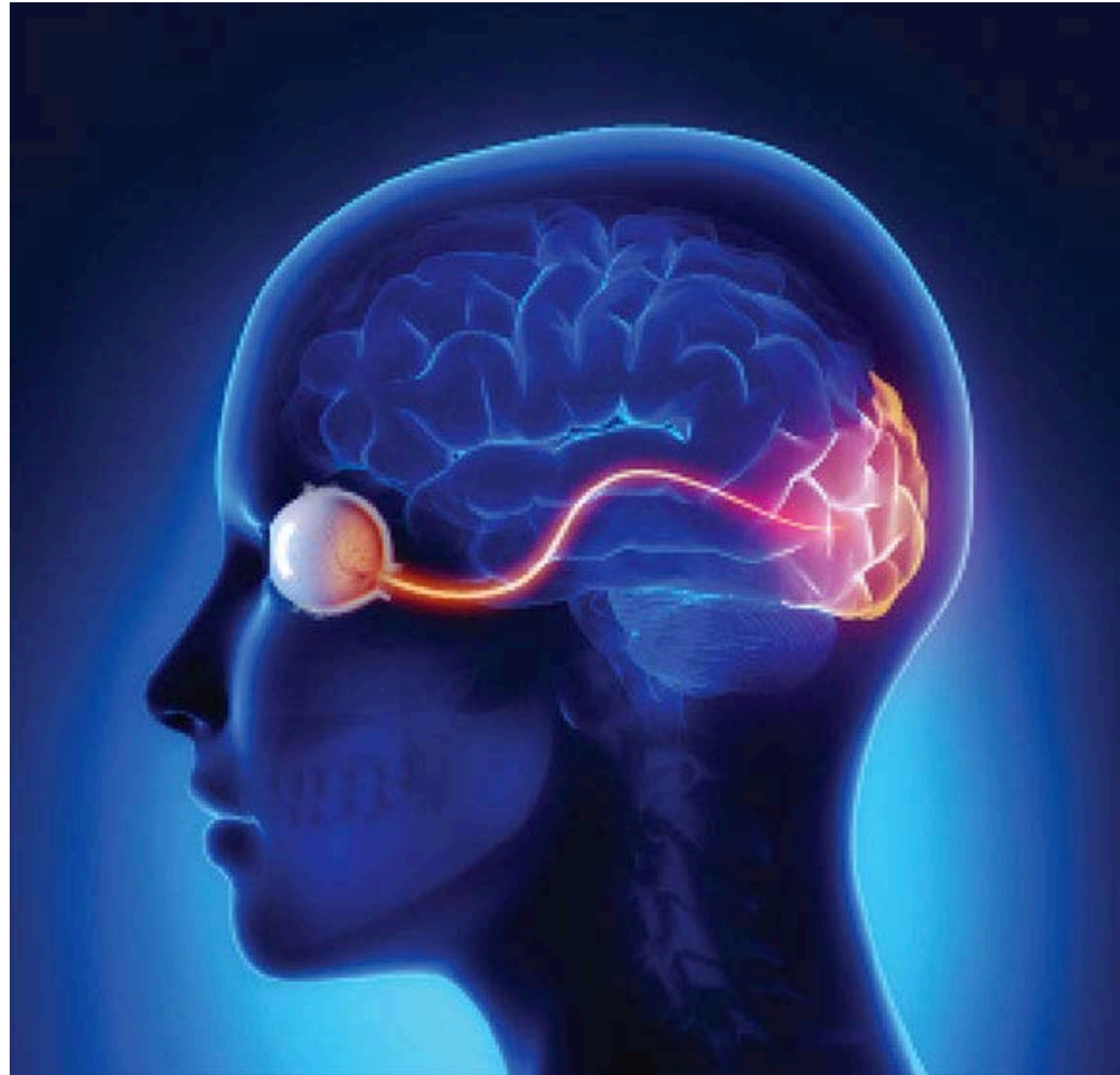
LA	A	B	C	A	B	C	La	Ch	N	Gn	Ac	Ci	Ct	SM	Ir	HM	Wl	A	C	E	F	a	d
Ch	D	E	F	D	E	F	Lo	Cin	S	Sk	Ma	Lo	FV	Ot	Ca	X	Tb	B	D	A	a	b	e
Lo	G	H	I	G	H	I	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cin	K	L	M	K	L	M	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
CS	N	O	P	N	O	P	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
LS	Q	R	S	Q	R	S	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Ka	b	c	d	b	c	d	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
RN	e	f	g	e	f	g	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
QC	h	i	j	h	i	j	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
AV	k	l	m	k	l	m	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
Eu	n	o	p	n	o	p	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
Er	q	r	s	q	r	s	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9

punched card 1890's



ENIAC I/O 1945

Why Computer Graphics?



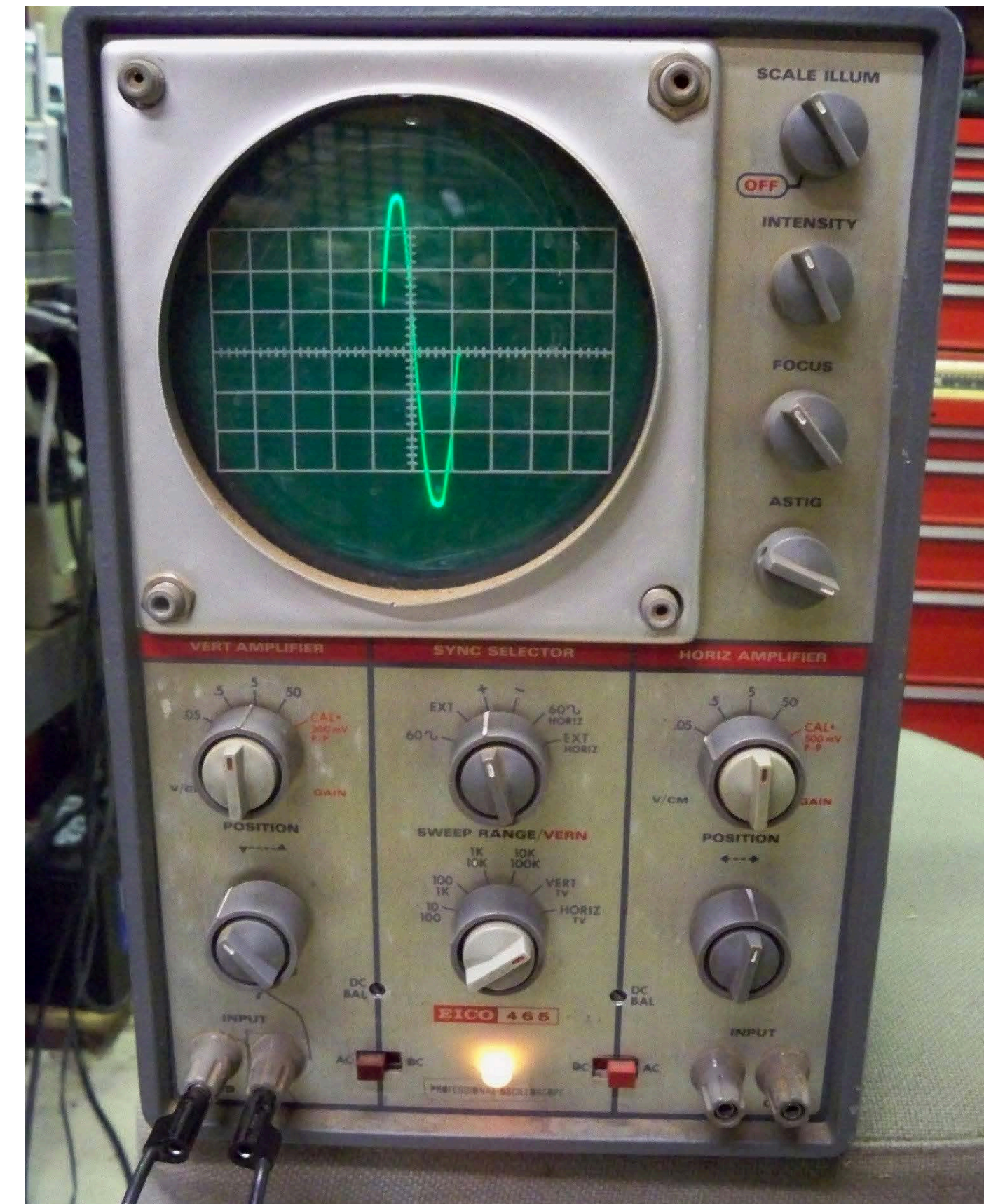
30% of the brain is devoted to visual processing
Most efficient way to receive information is in the form of visual data

Computer Graphics

Computer graphics

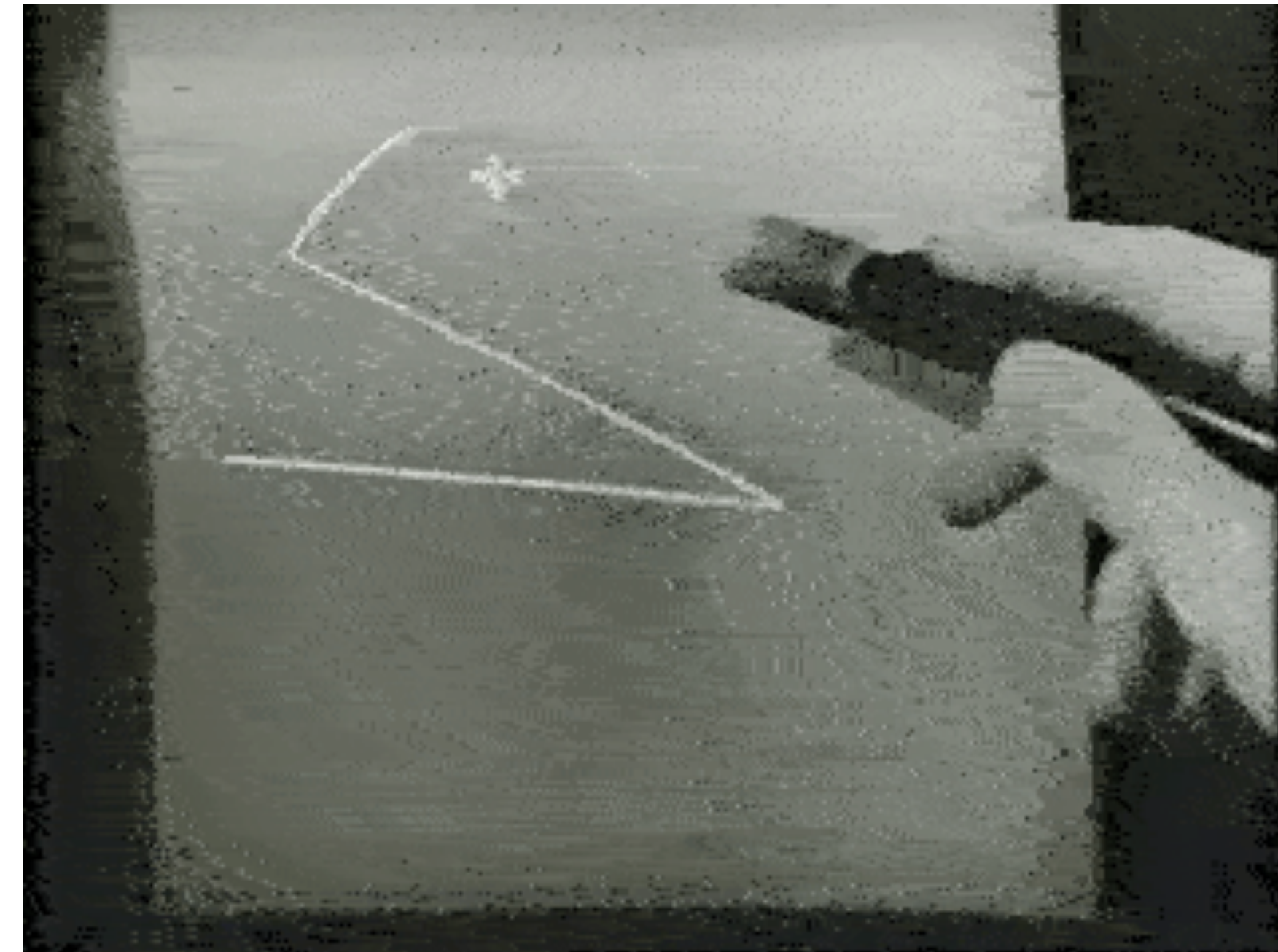
The use of computer to synthesize visual informations.

History of Graphics: Visual output



CRT monitors 1950's–1960's

History of Graphics: Visual input



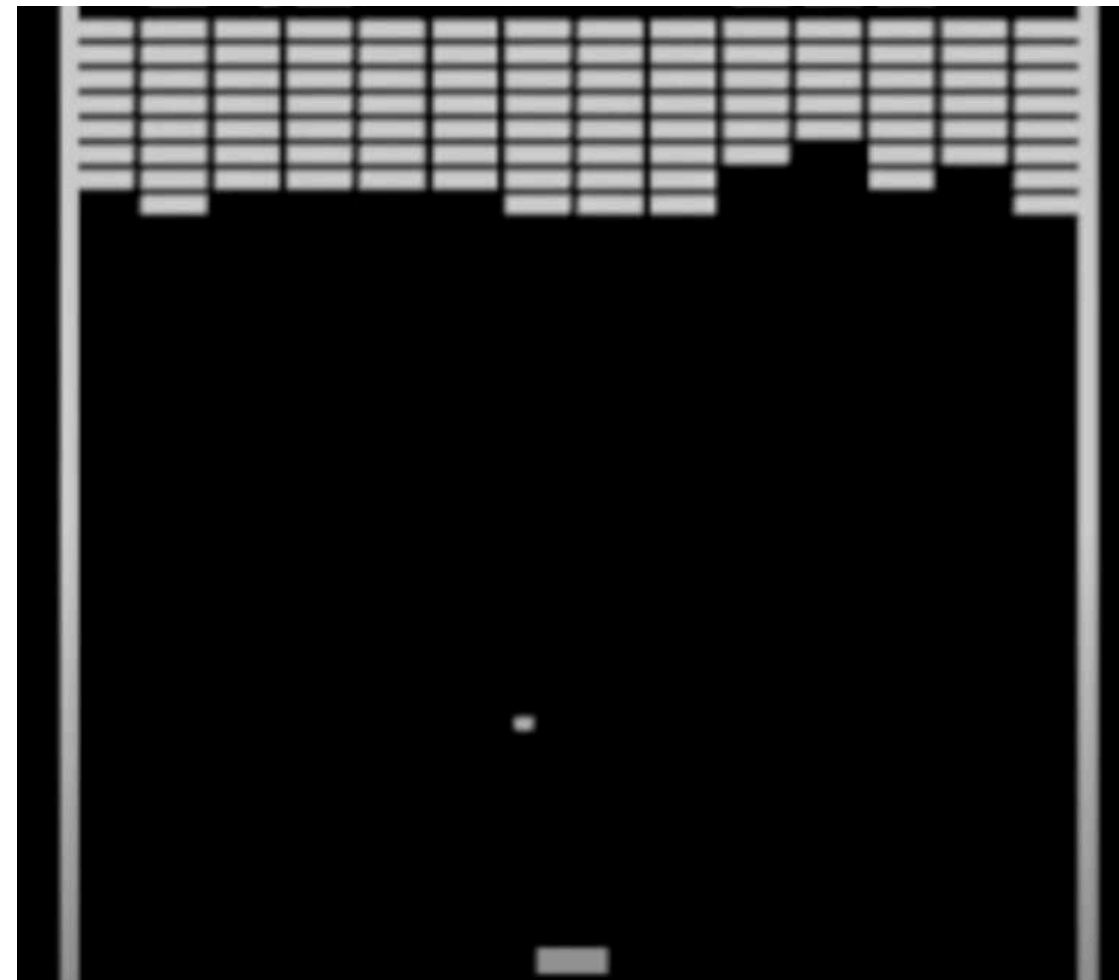
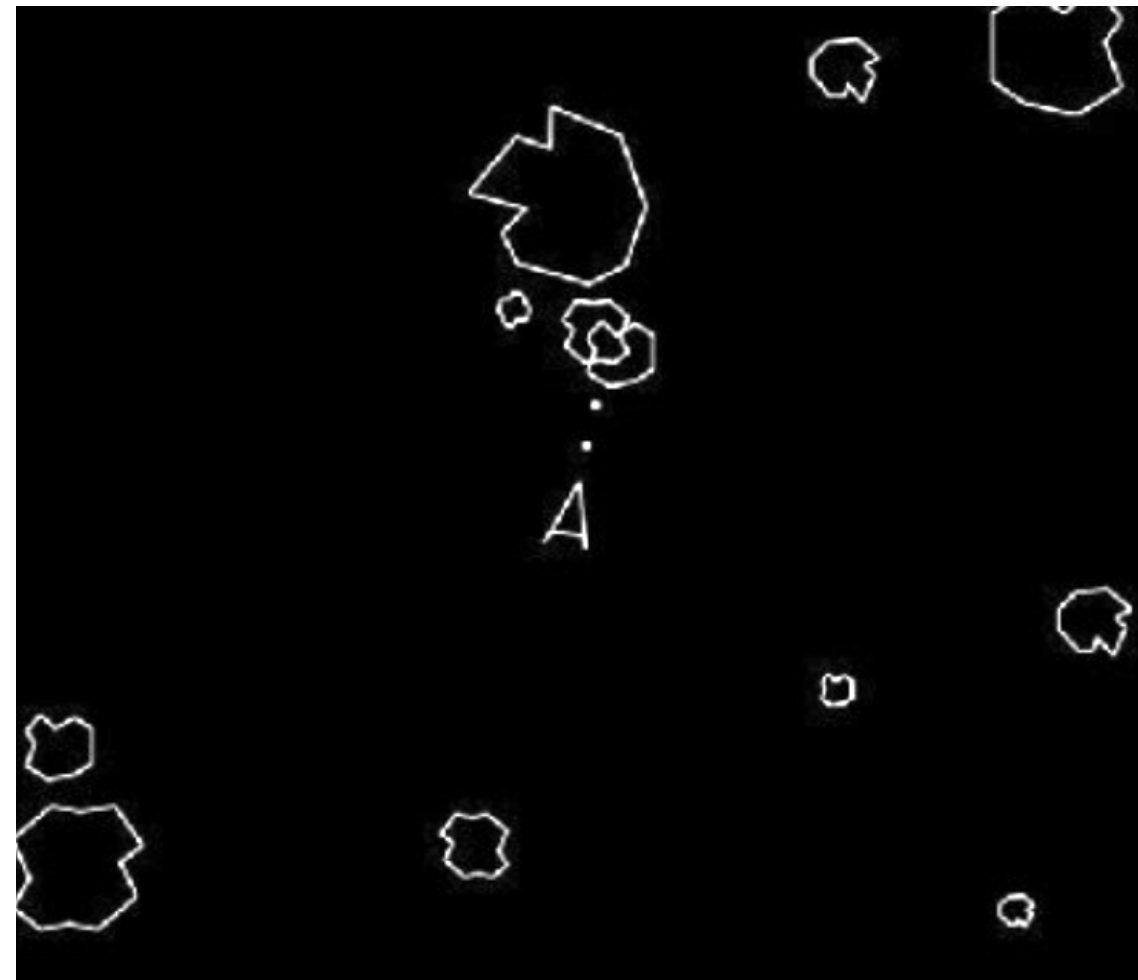
“Sketchpad” – Ivan Sutherland 1963

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

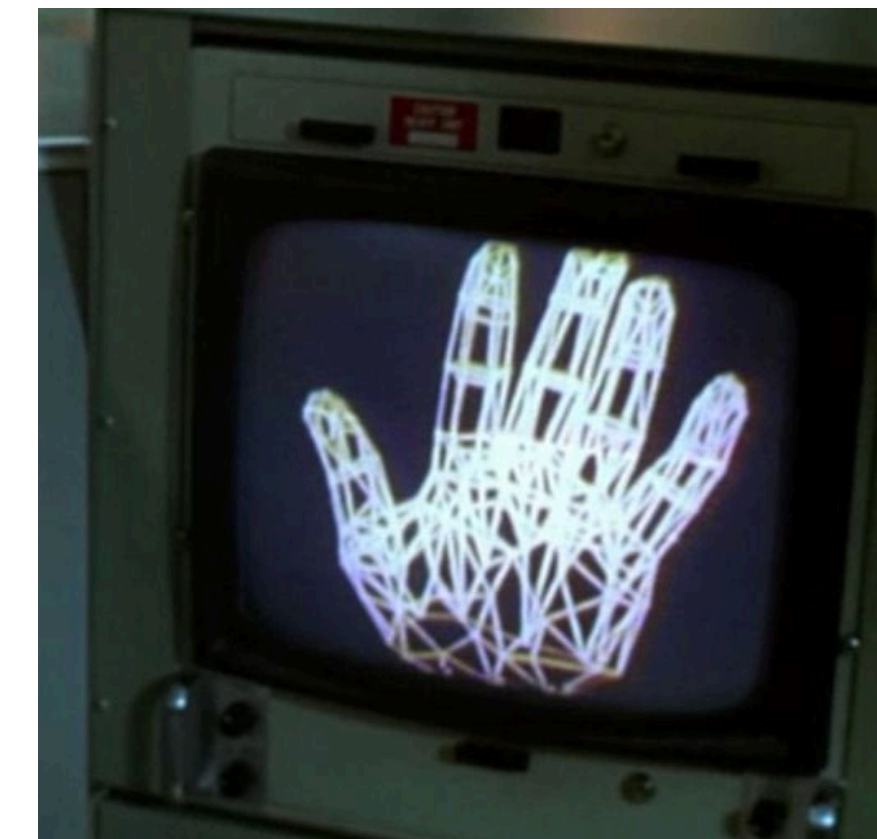


Arcade games 1970's

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

3DCG: Ed Catmull & others in Utah

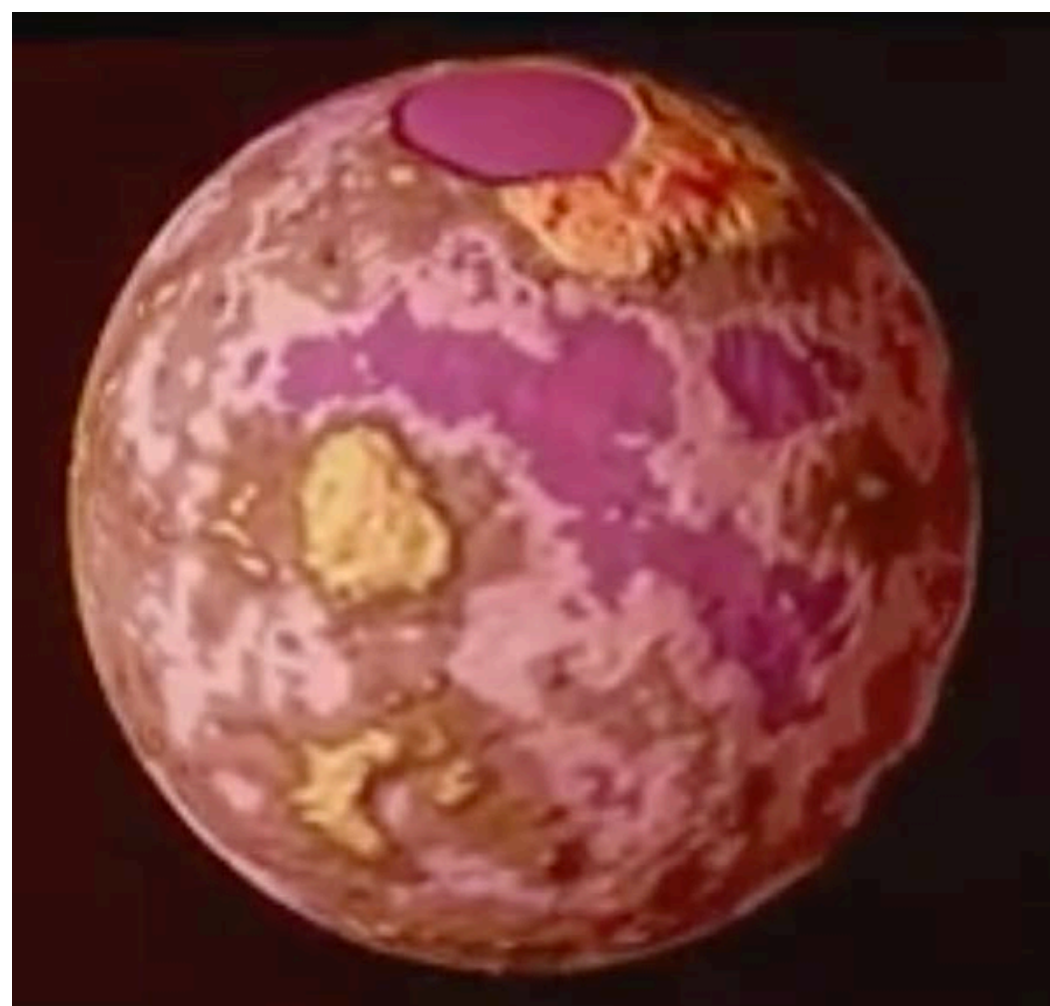
- ▶ Z-buffering
- ▶ Texture mapping
- ▶ Subdivision



Lucasfilm *Star Wars IV* 1977

- ▶ Computer graphics in blockbusters

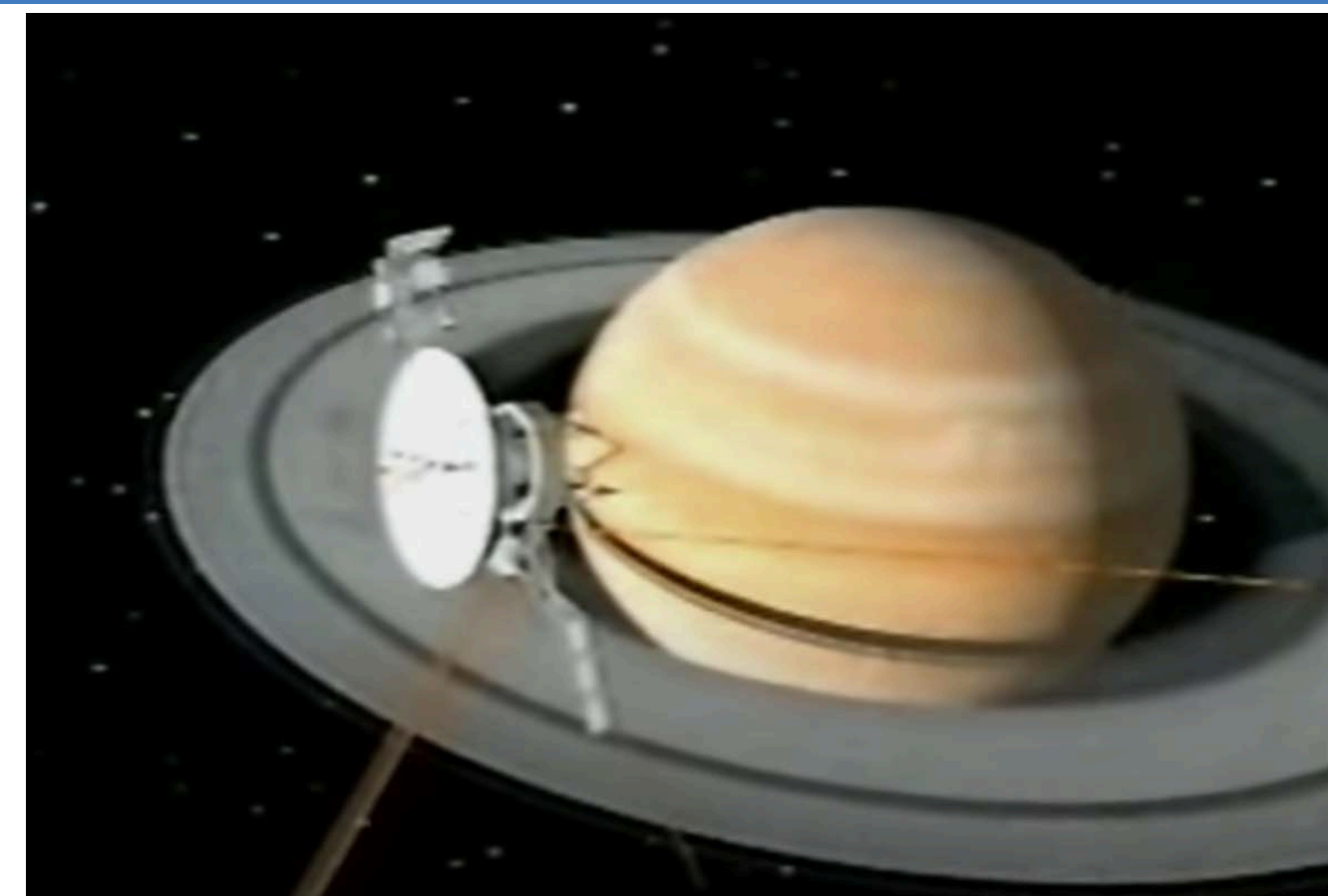
History of Graphics: Voyagers & Cosmos



Bump map for Venus 1978



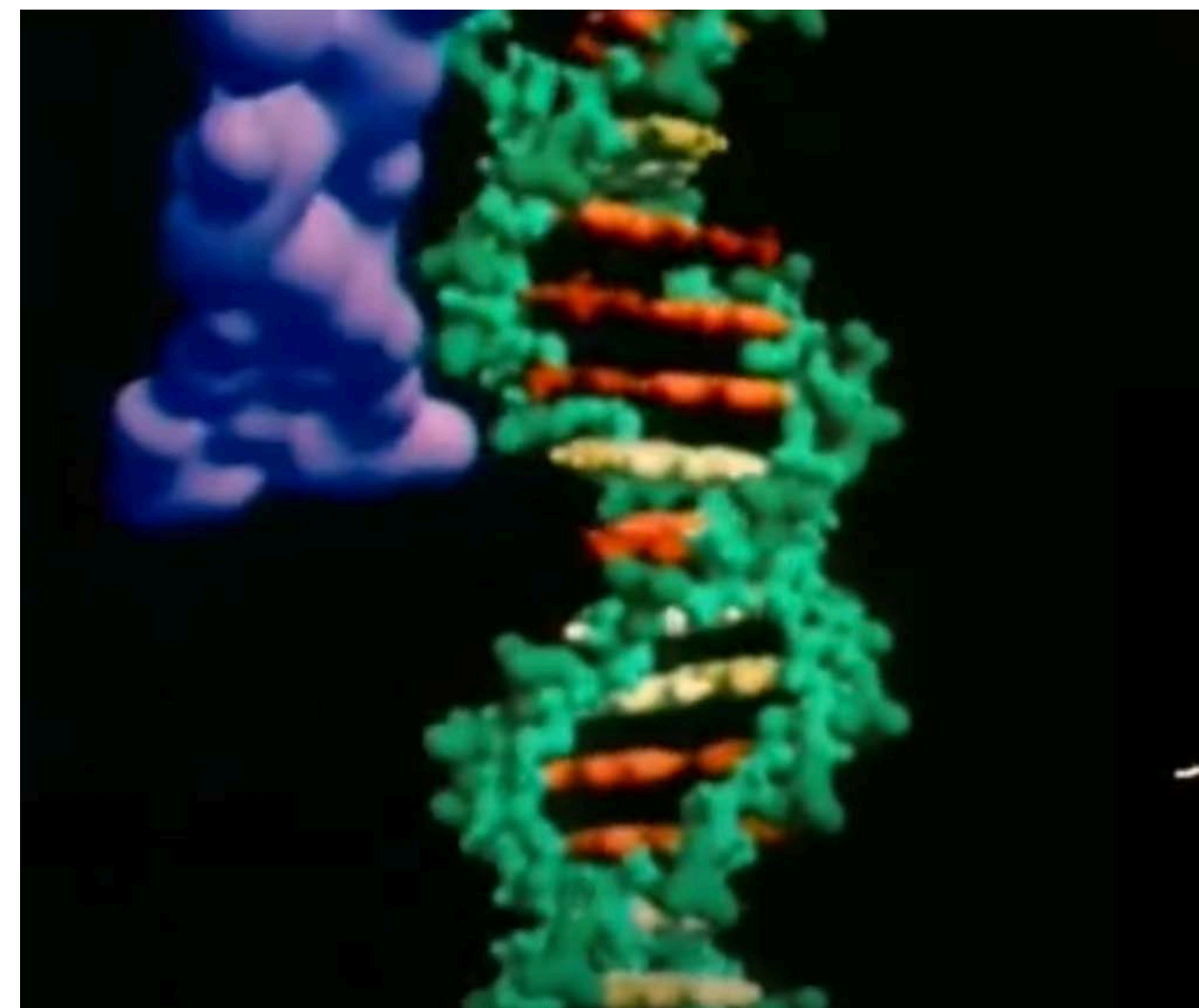
Texture reconstruction for moons of Jupiter



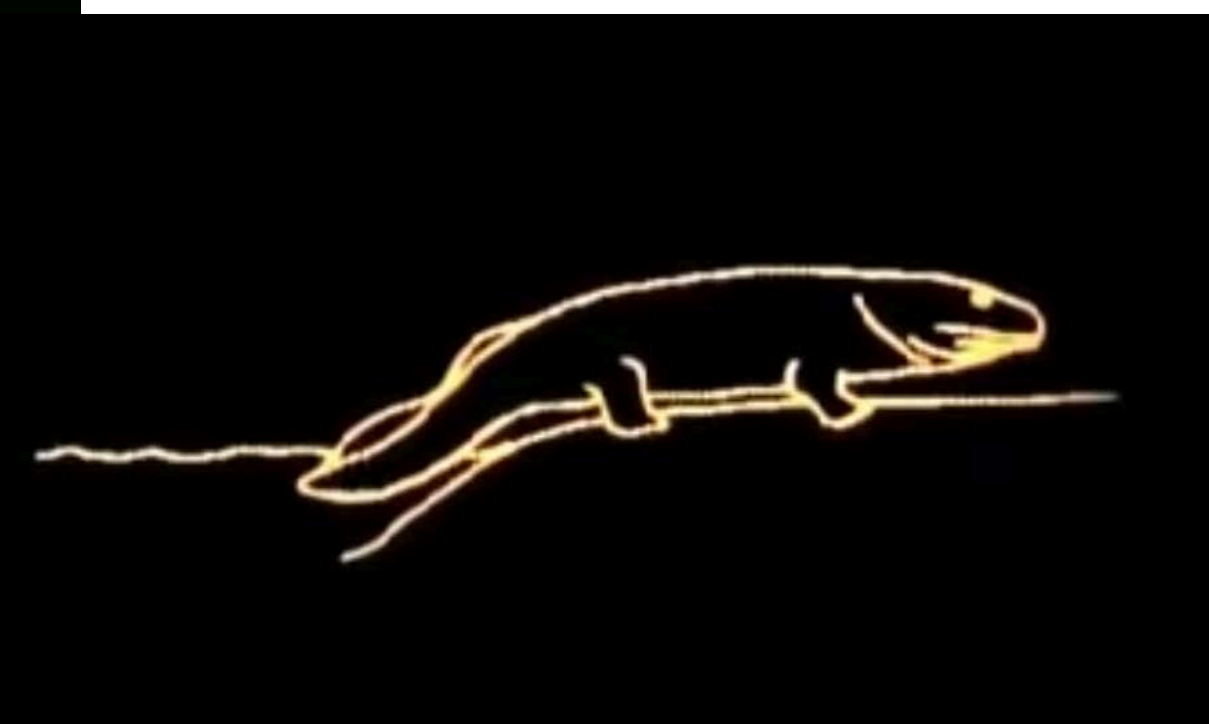
V2 Saturn flyby 1981



V1 Jupiter flyby 1979



Cosmos "DNA" 1980



Cosmos "evolution" 1980

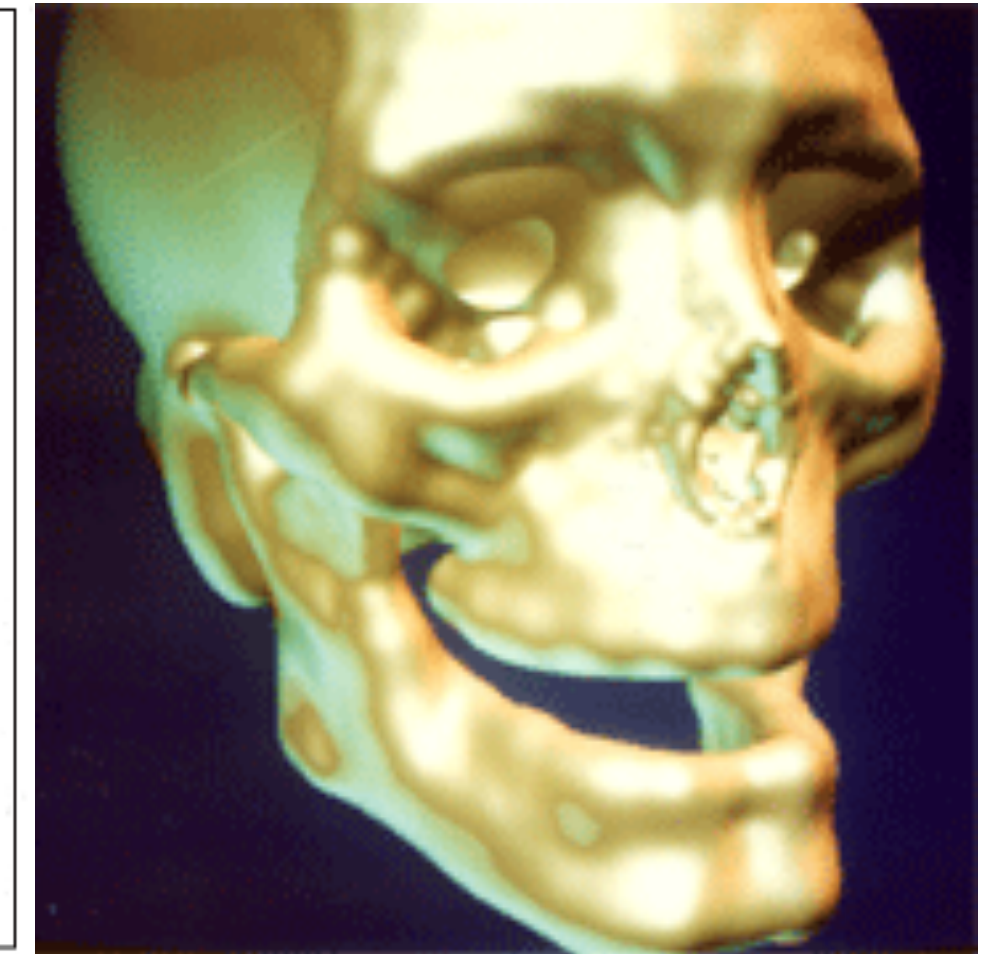
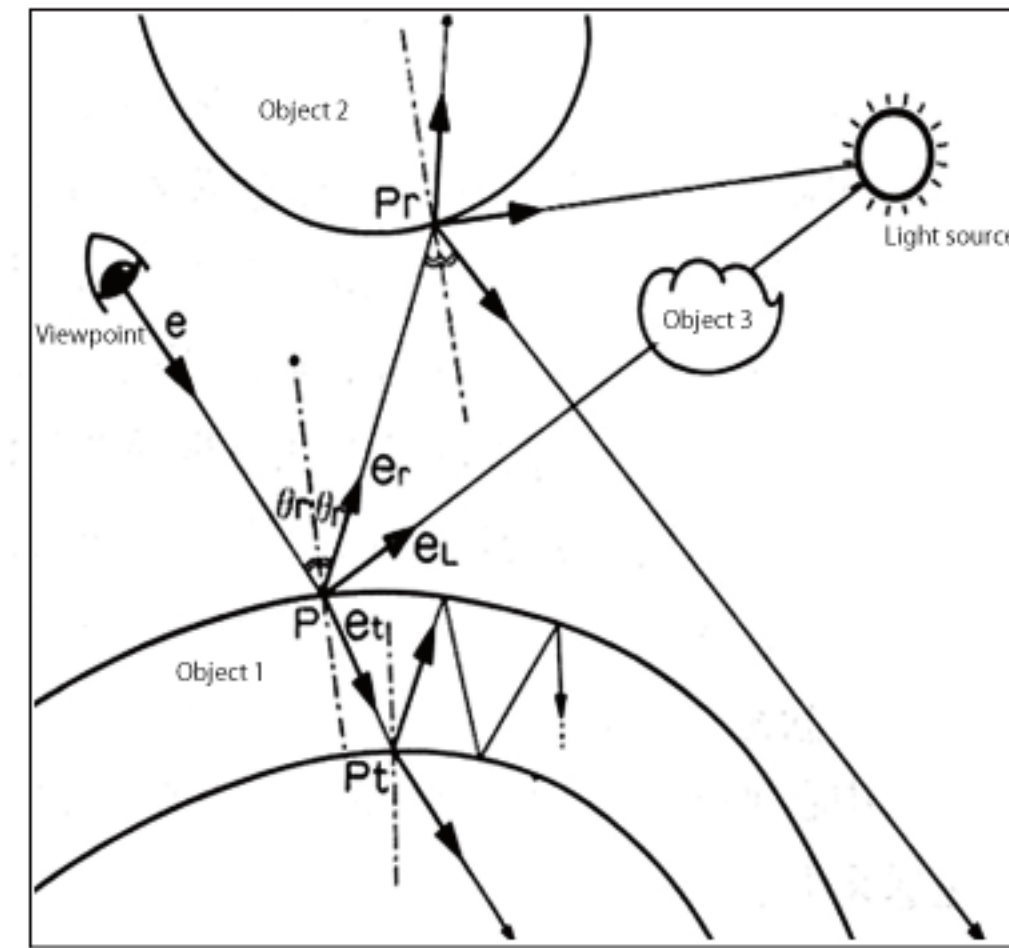


James Blinn

History of Graphics: 1980's

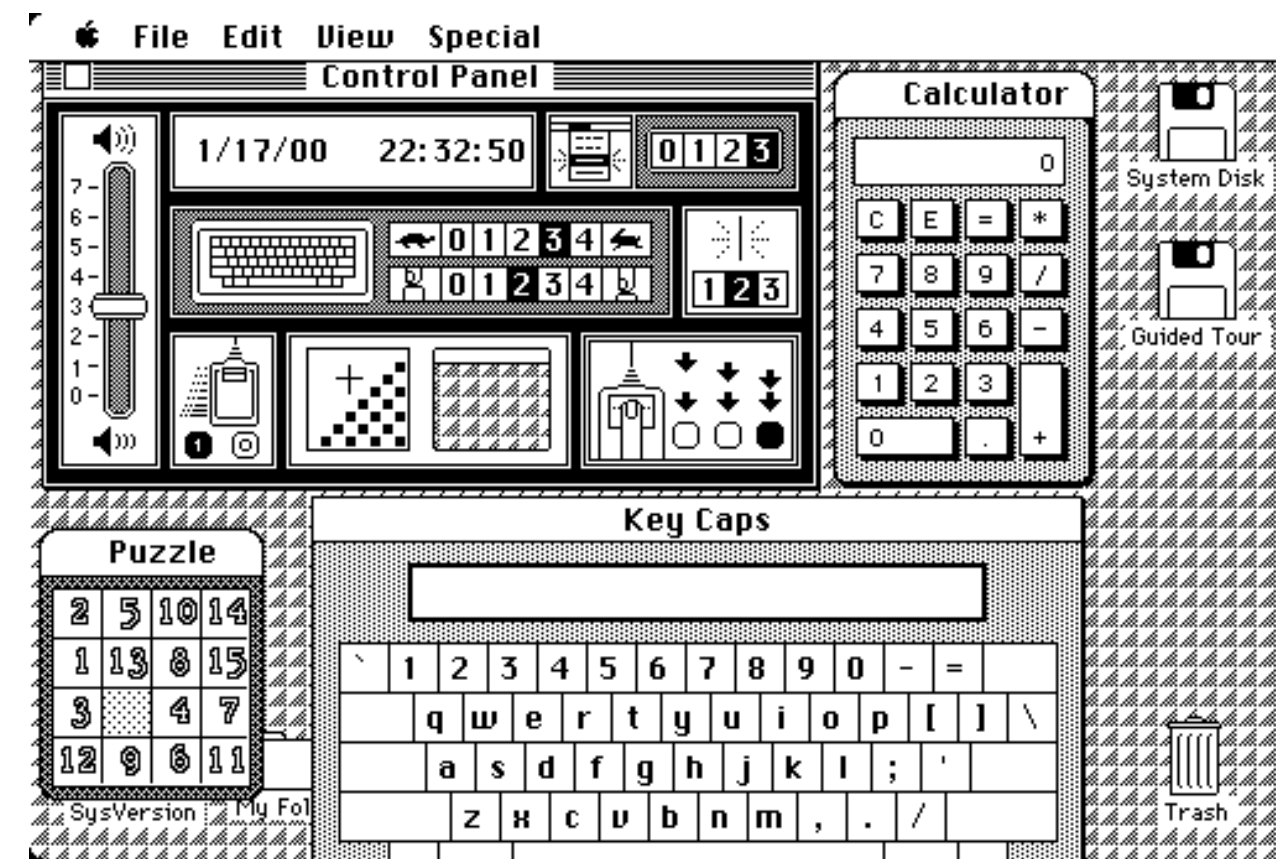


Nintendo 1981



$$I(P) = I_o(P) + \text{src}(\theta_r) \cdot I(\text{Pr}) + t \cdot I(\text{Pt})$$

LINKS-1 CG System 1983



Macintosh 1984
PC graphics



Pixar *Luxo Jr.* 1986

History of Graphics: interface with GPU

Graphics API's 1990s

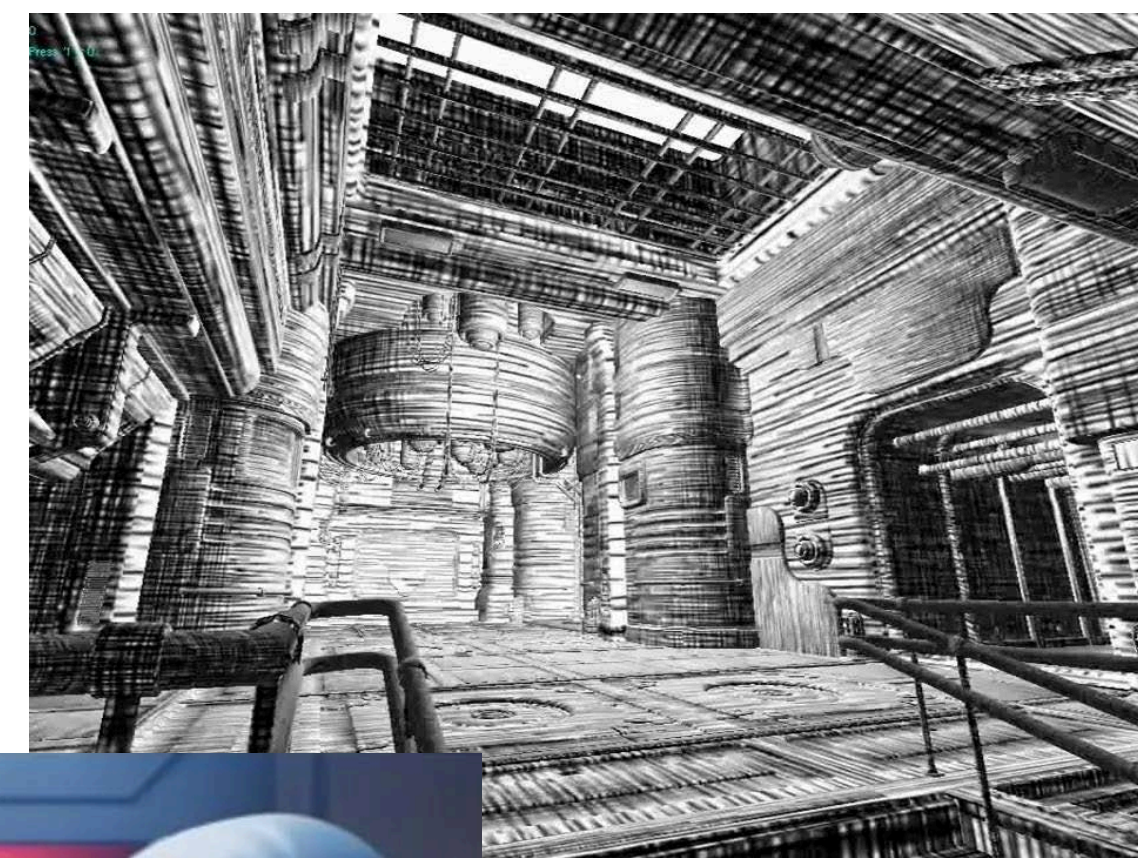
- ▶ Simple to tell GPU to draw
- ▶ Hardcoded fixed function for drawing (fast but not flexible).



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



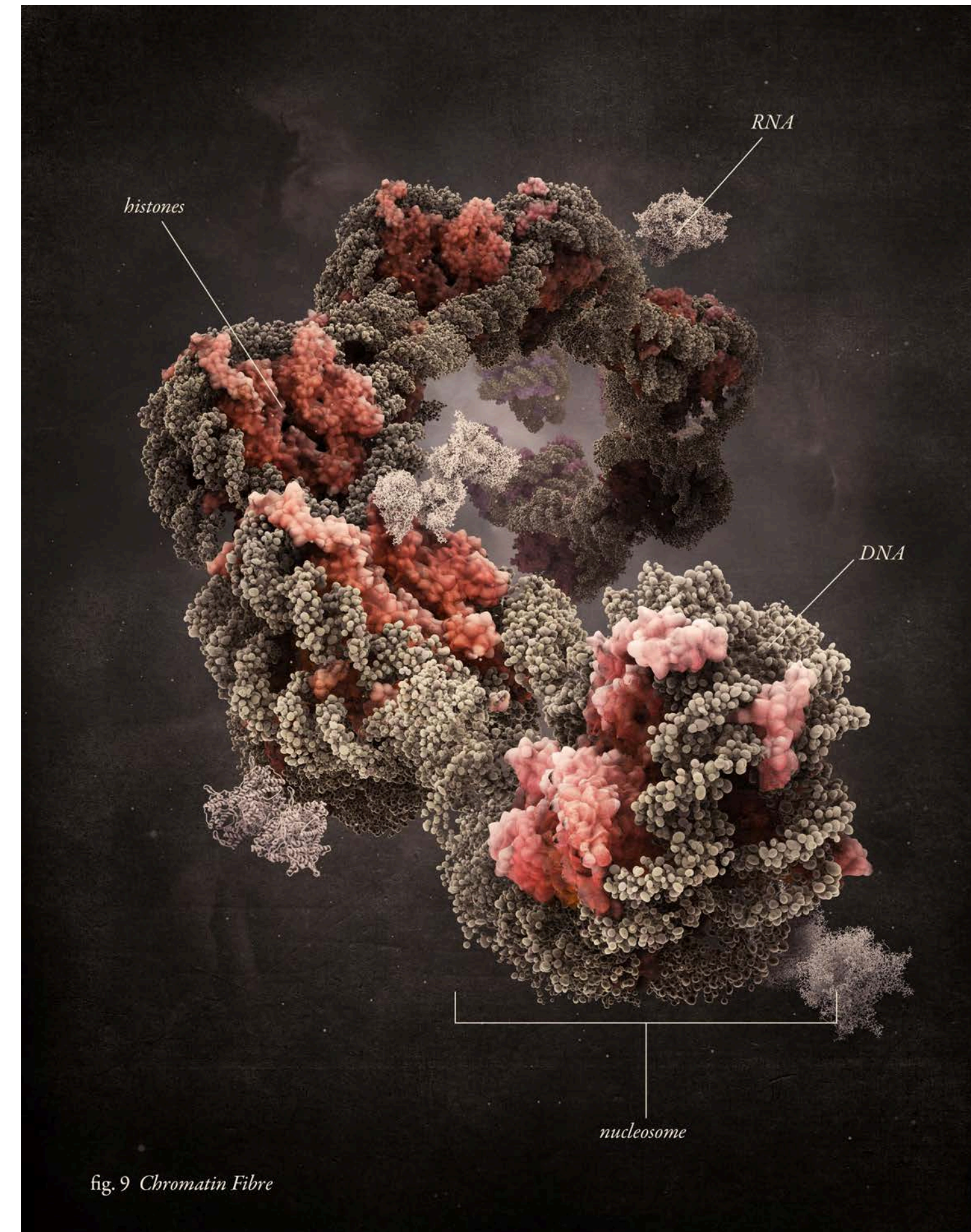
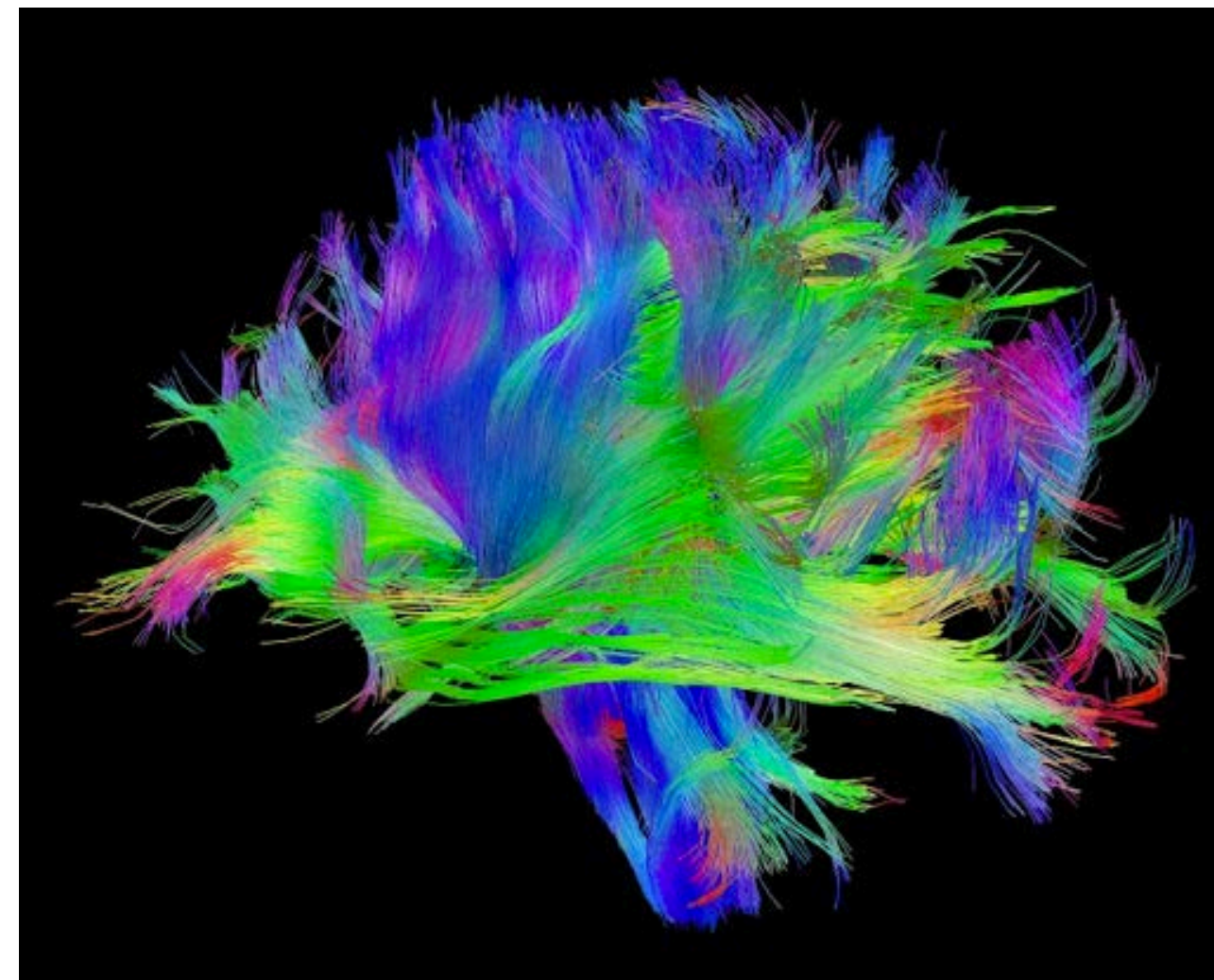
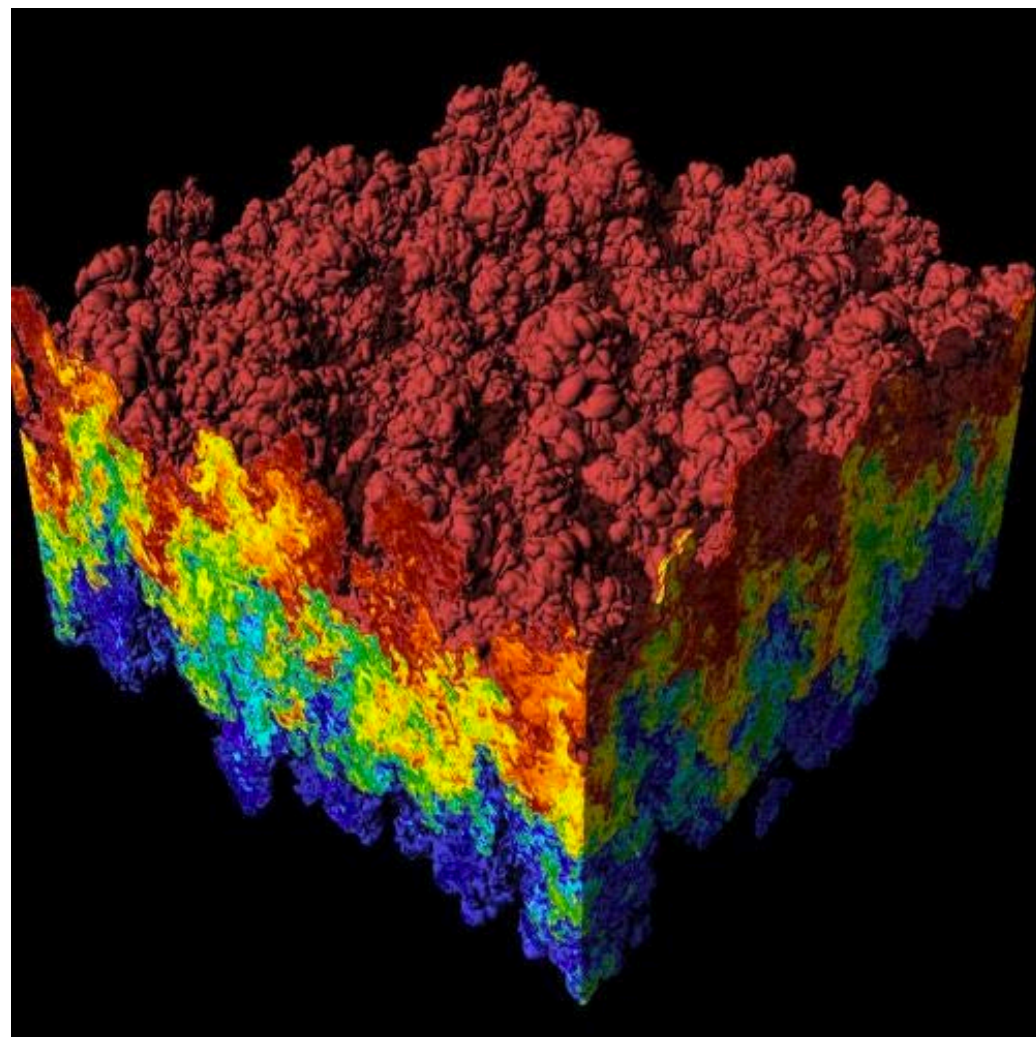
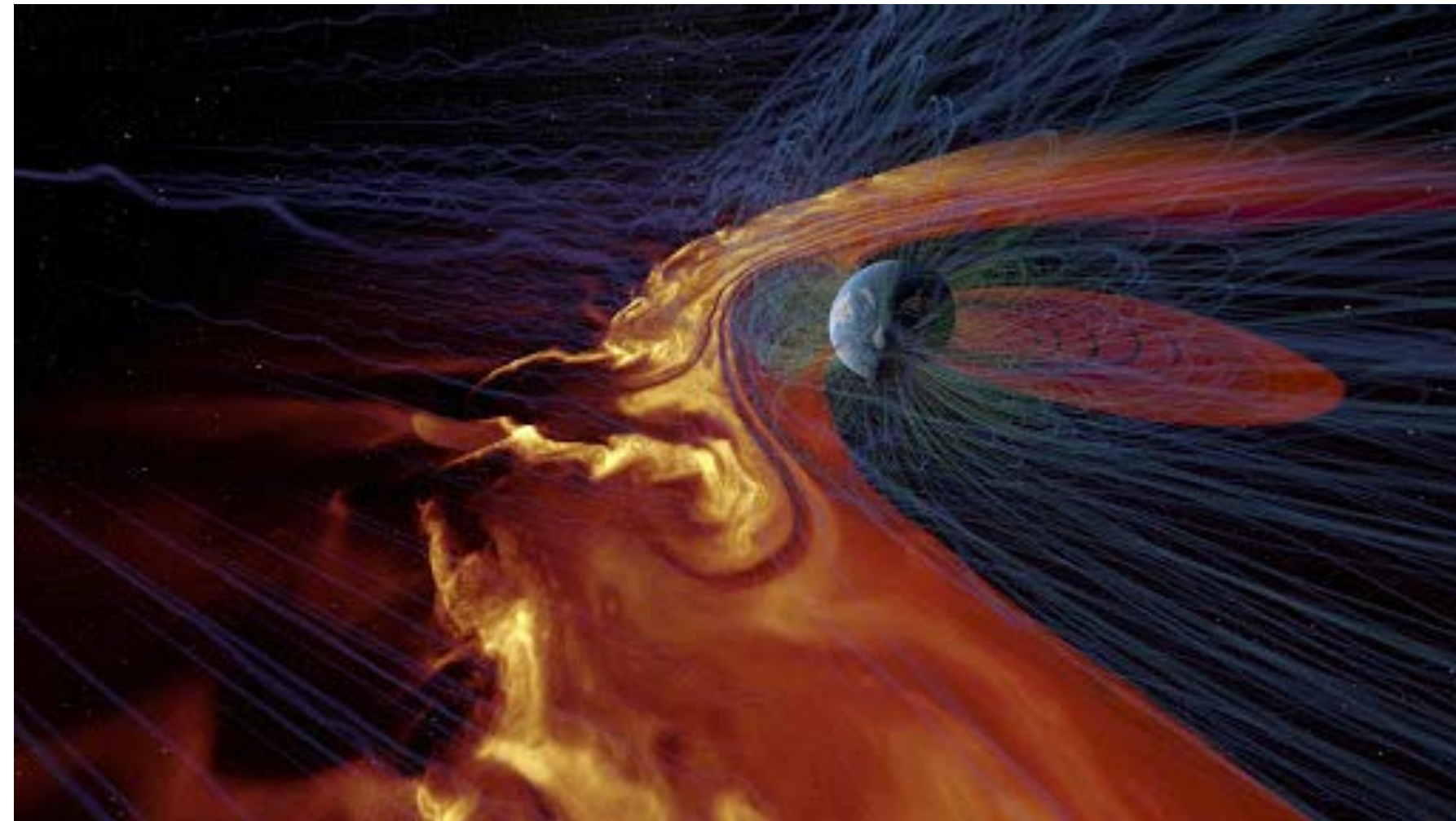
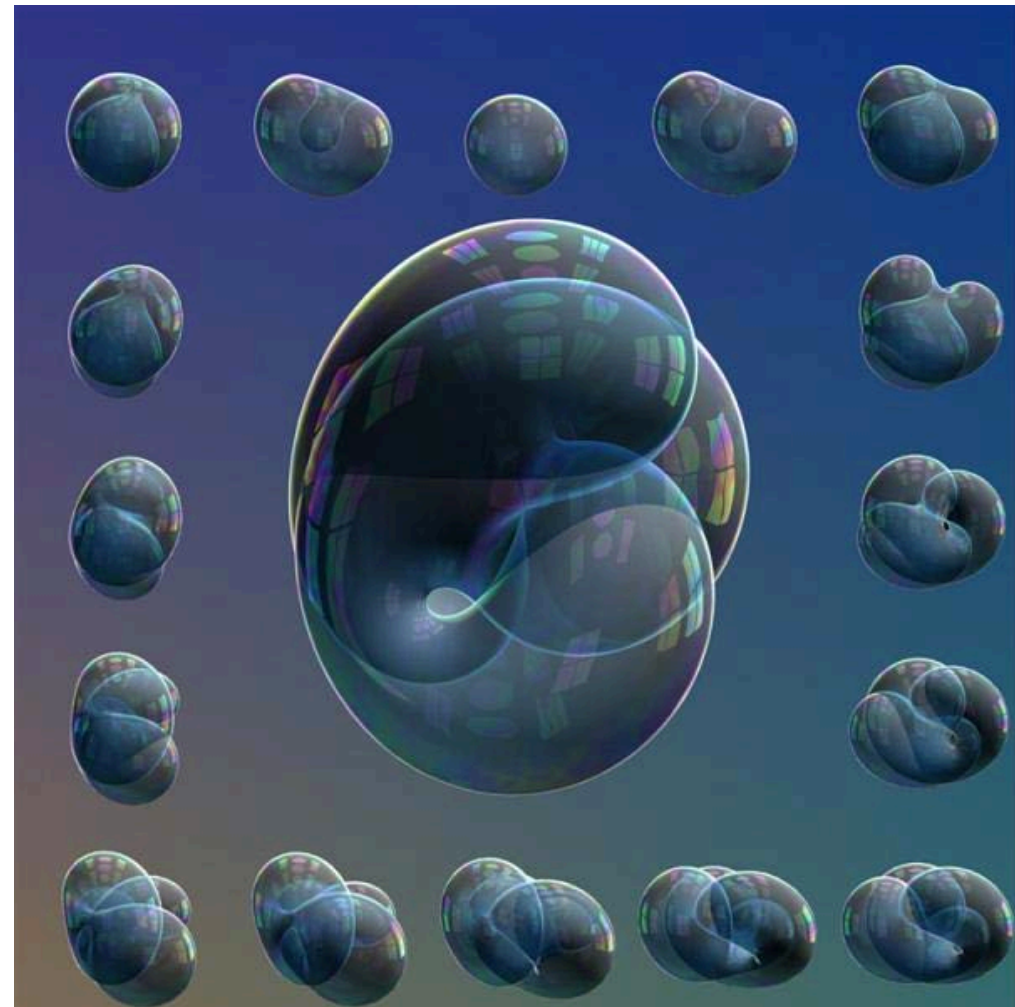
Beyond controlling pixels

Physical simulations in entertainment



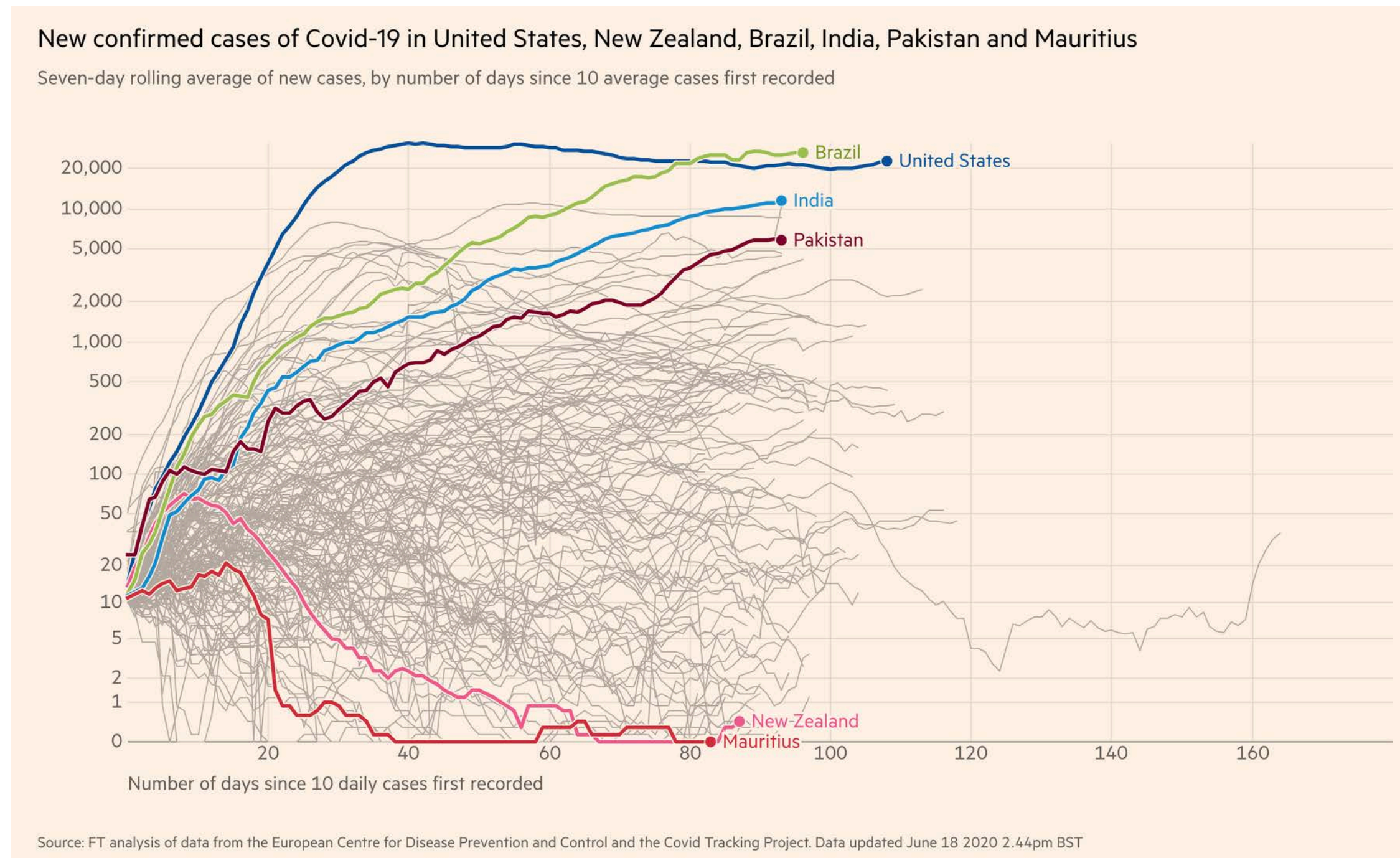
Beyond controlling pixels

Mathematical and Scientific Visualizations

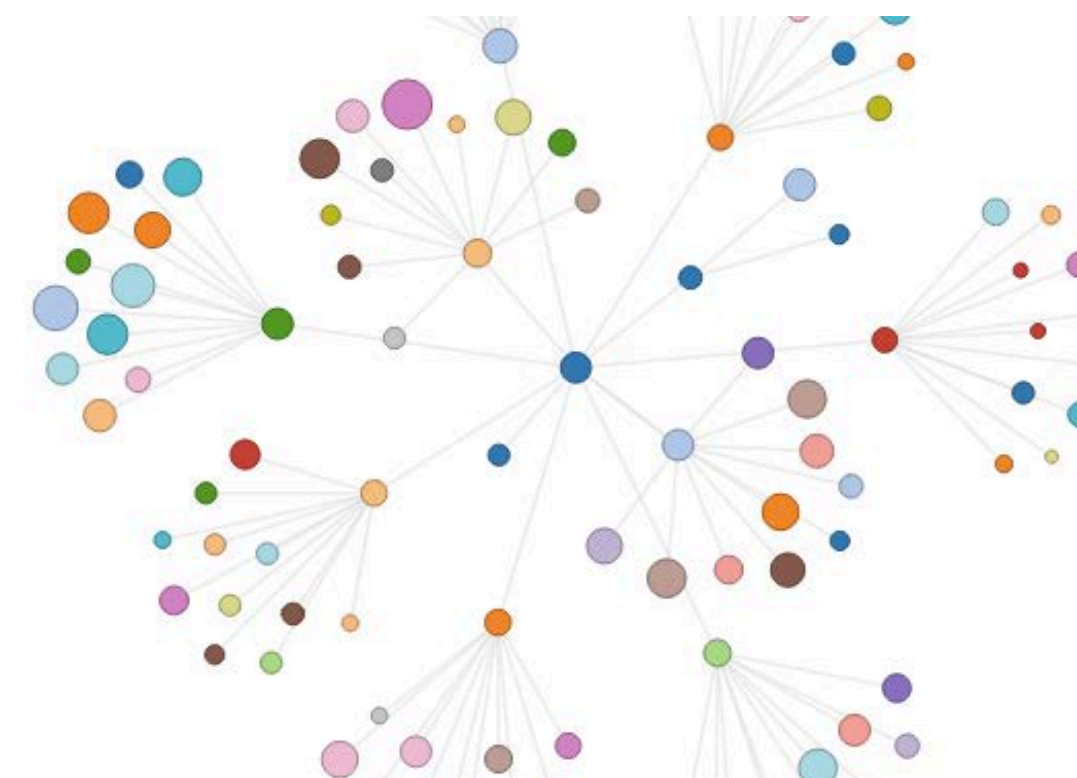
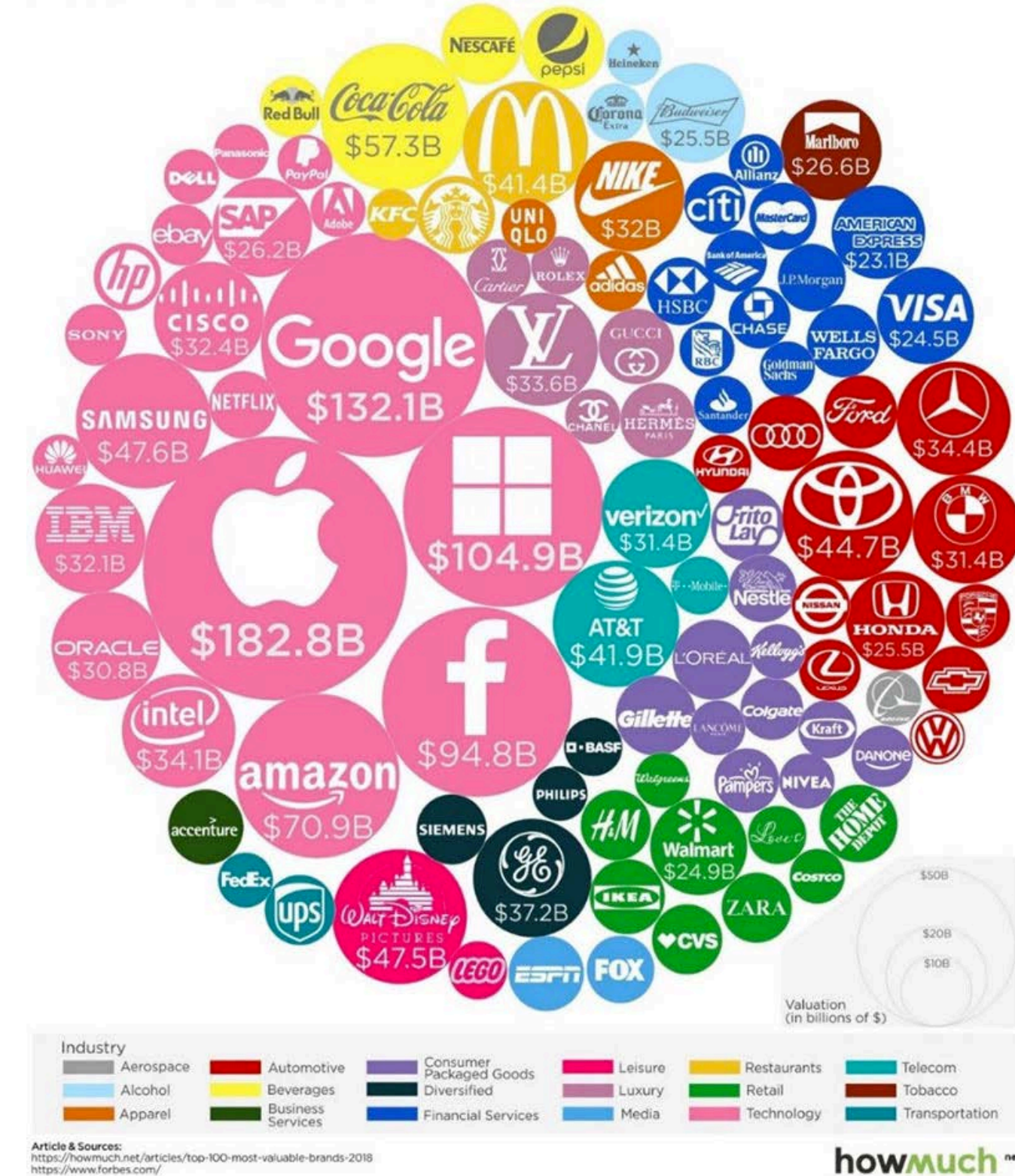


Beyond controlling pixels

Data visualizations

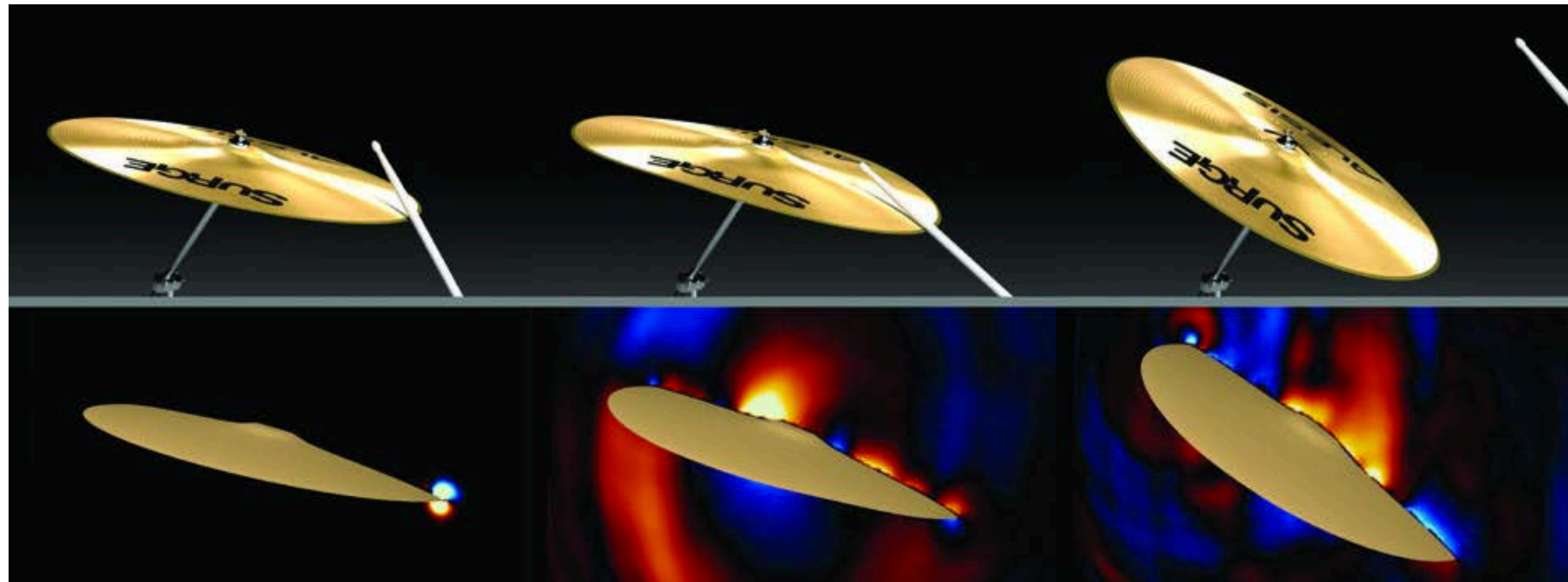


Brand Value (in billions of \$)



Beyond controlling pixels

Synthesize sound (not only visual information)



and touch senses



Beyond controlling pixels

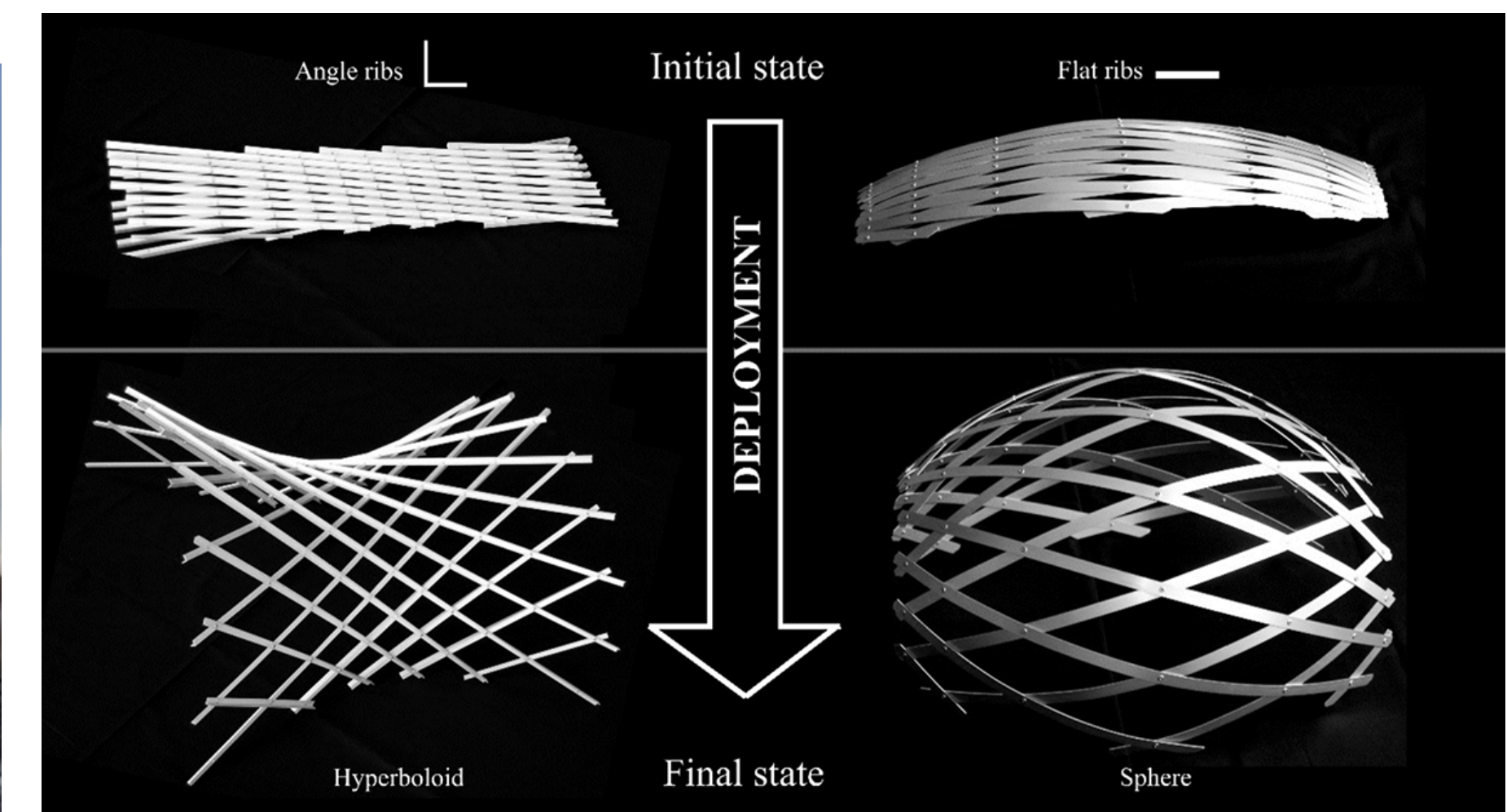
3D printing



Constrained deformable geometry

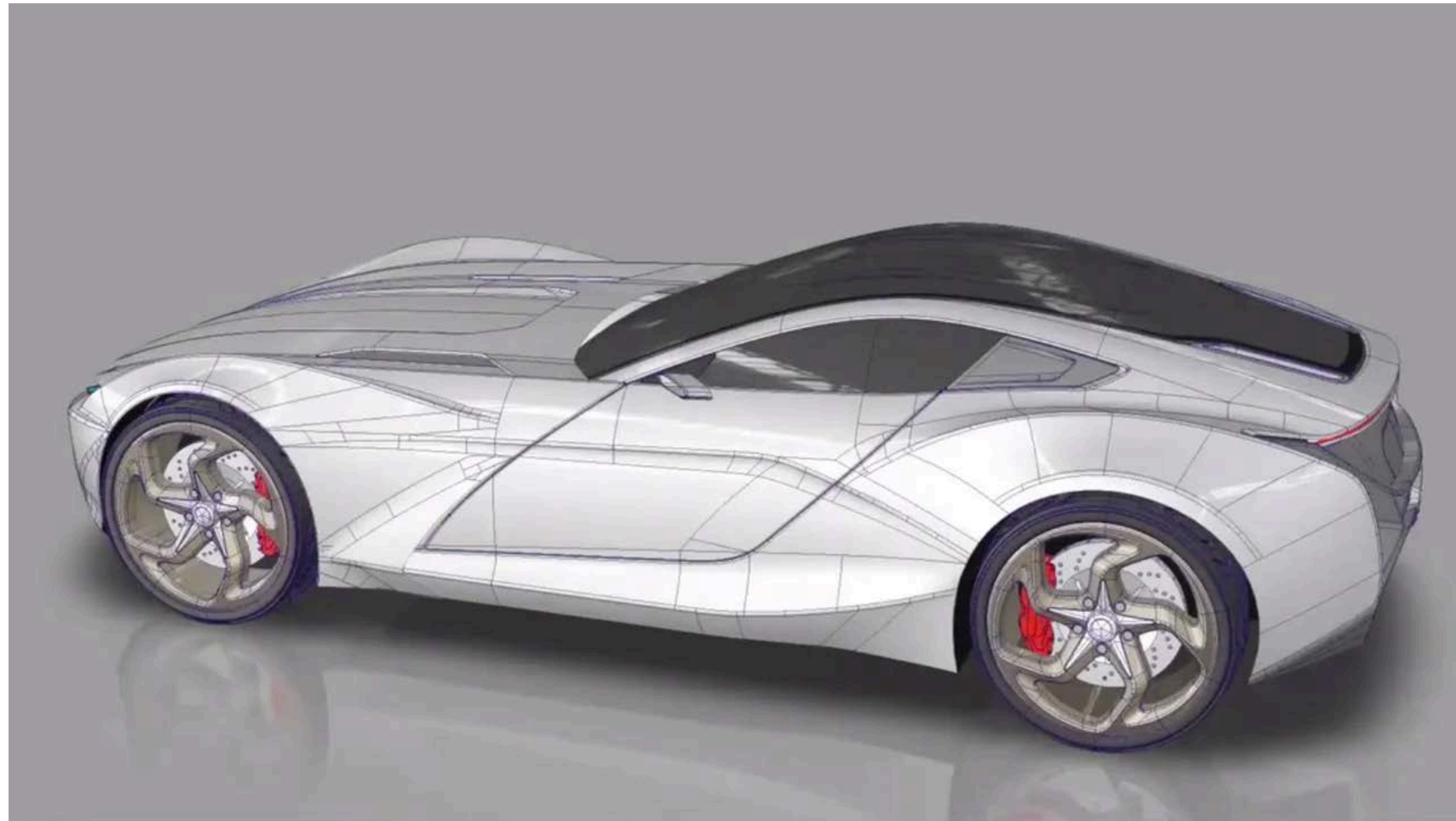


architecture



Beyond controlling pixels

Industrial computer aided design



Fonts on your screen and on newspapers



OUR STORY

Formed in New Zealand in 2004, Image Mechanics has gone from strength to strength since opening our Sydney studio, continuing to deliver premium branding, design and digital services.

With a ton of successful new media projects under our belt, we've turned our experience designing and developing websites and applications to the art of crafting unique and exciting user experiences within the mobile space. This means we're able to take an integrated approach to creating innovative applications to enhance your business and brand, looking at the wider picture beyond just apps.

If you're after a mobile strategy to revolutionize your business, optimise your website for mobile devices, or something else bold and brave, give us a call. We're not just designers - we're innovators, too.

Frontier of Computer Graphics

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

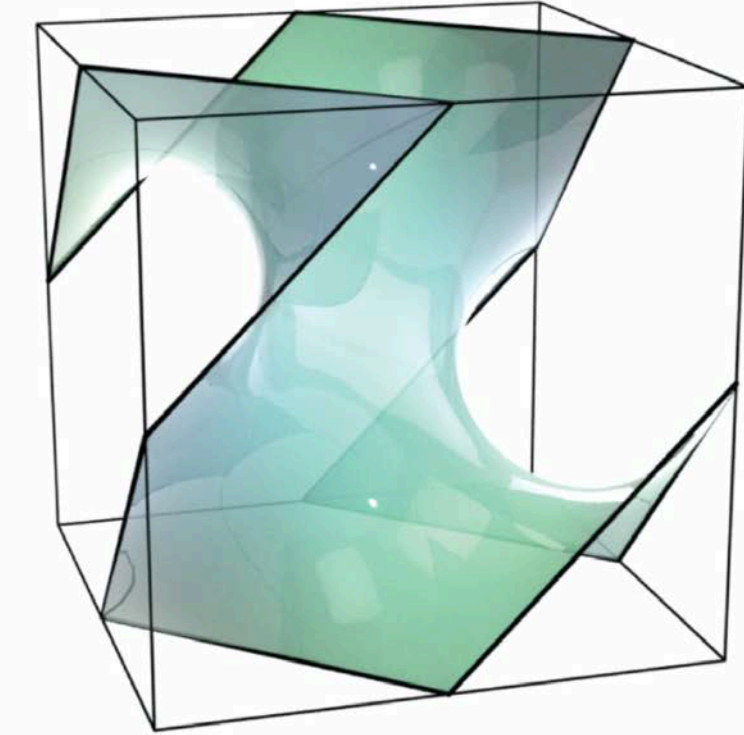
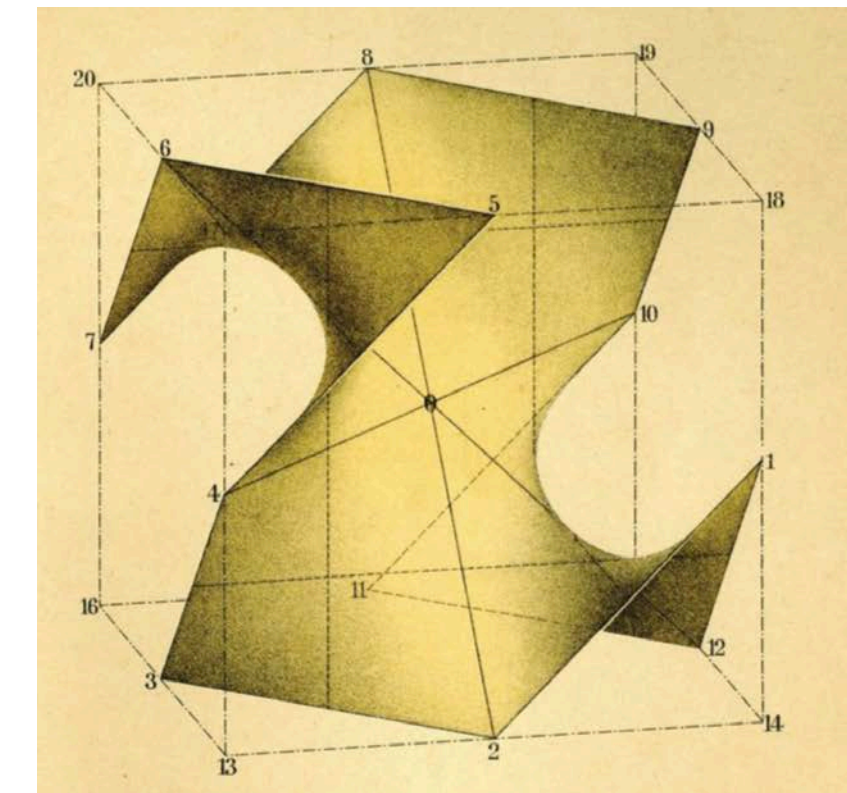
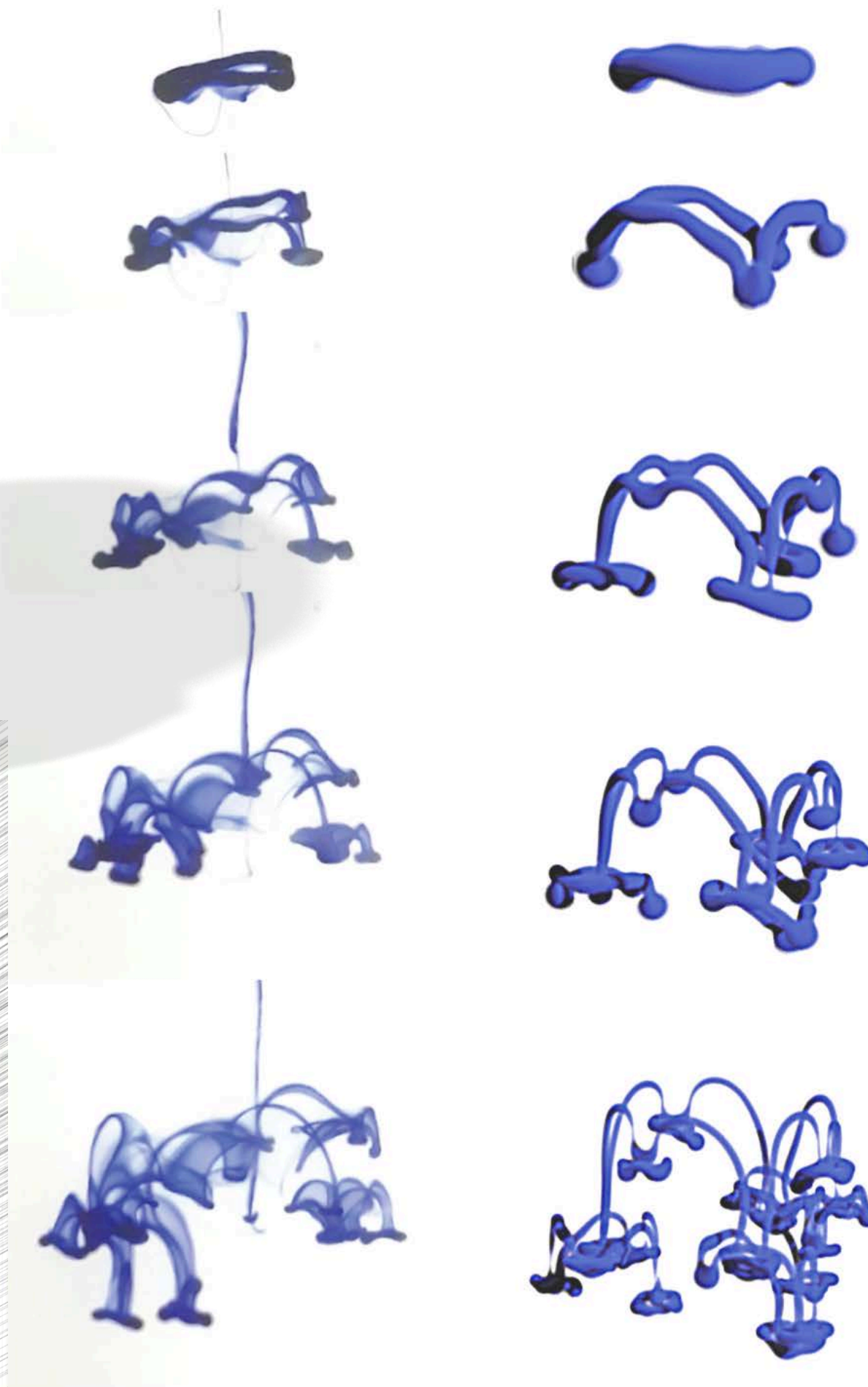
Youtube search “**SIGGRAPH technical paper trailer**”



Our research

Geometry processing & physical simulation

- Find underlying geometric structures to make simulations easier





Nabizadeh, Wang, Ramamoorthi, C.
Covector Fluids
2022

Topics of this course

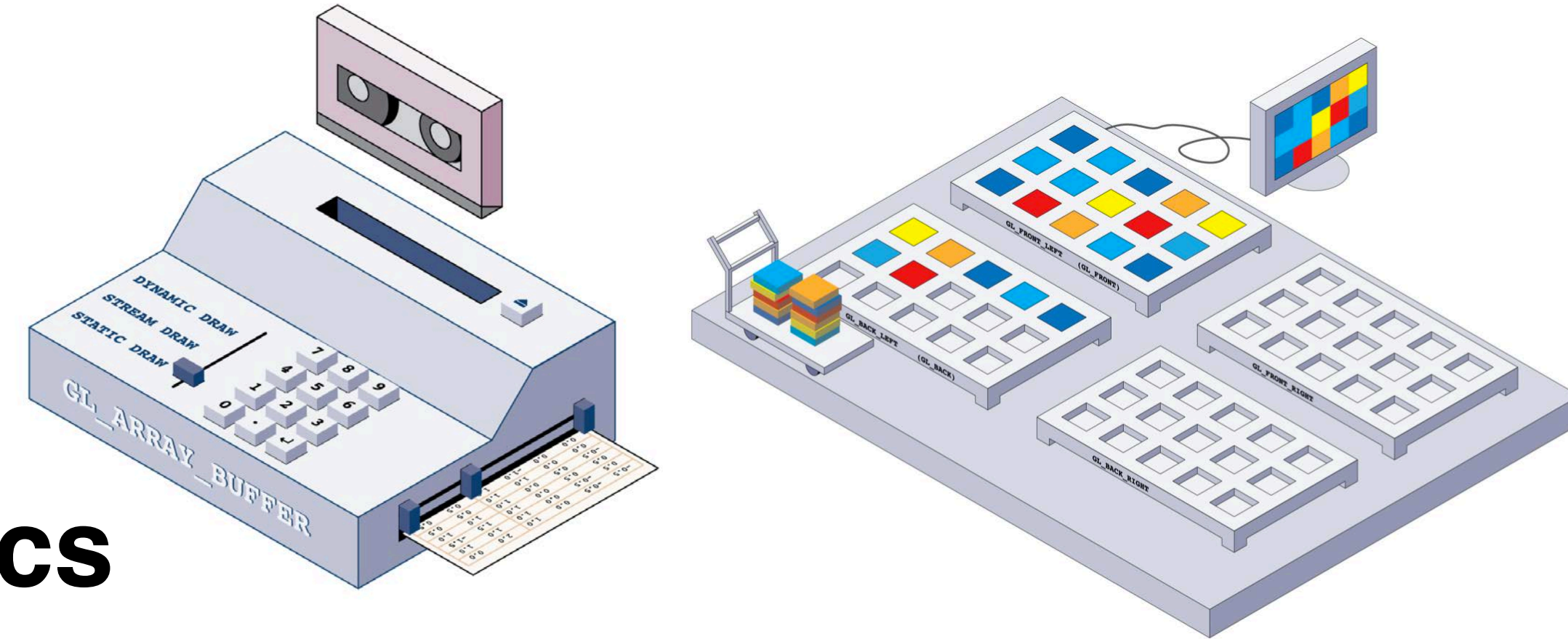
This course

Modern OpenGL

- Command the graphics card

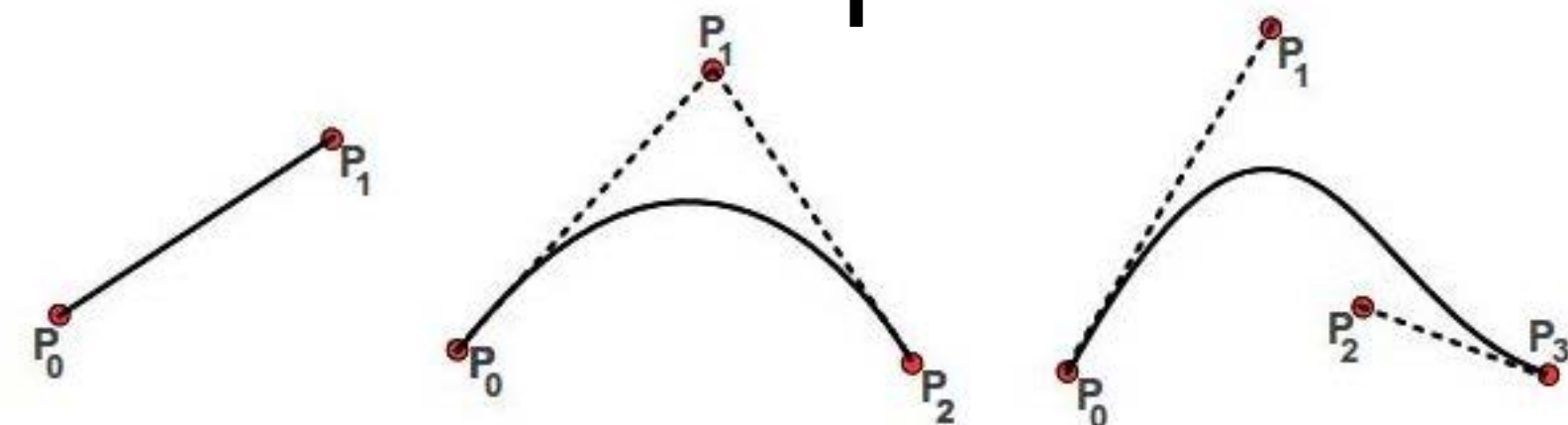
Foundation of 3D Computer Graphics

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



Foundation of Vector Graphics

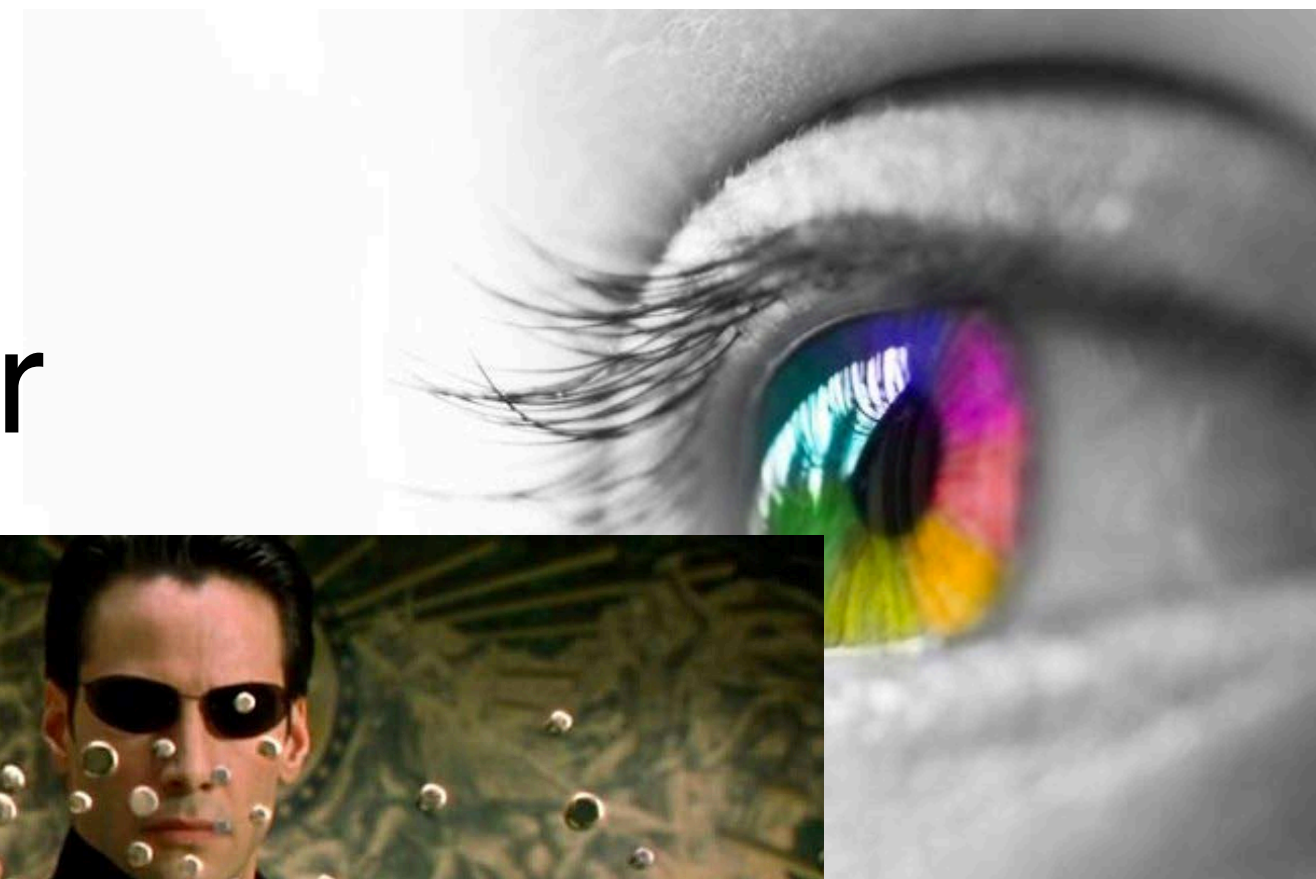
- Build smooth geometries from only a few control points



Additional topics

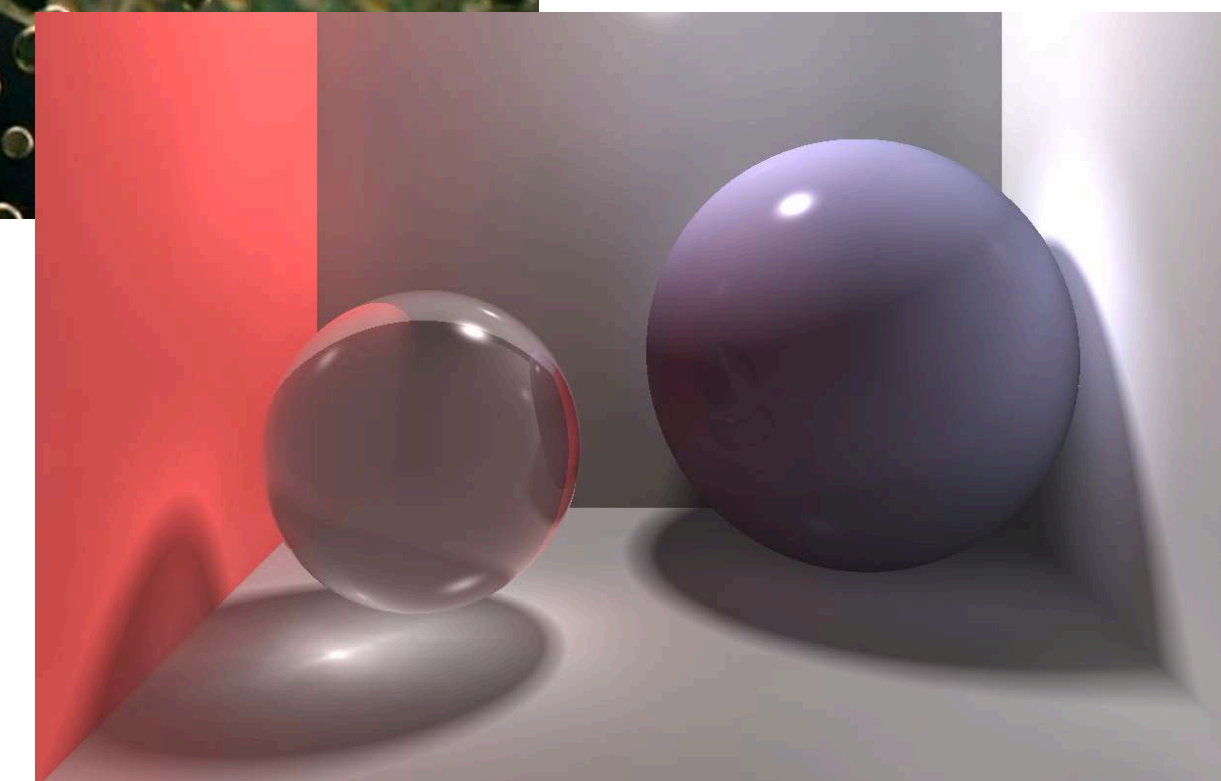
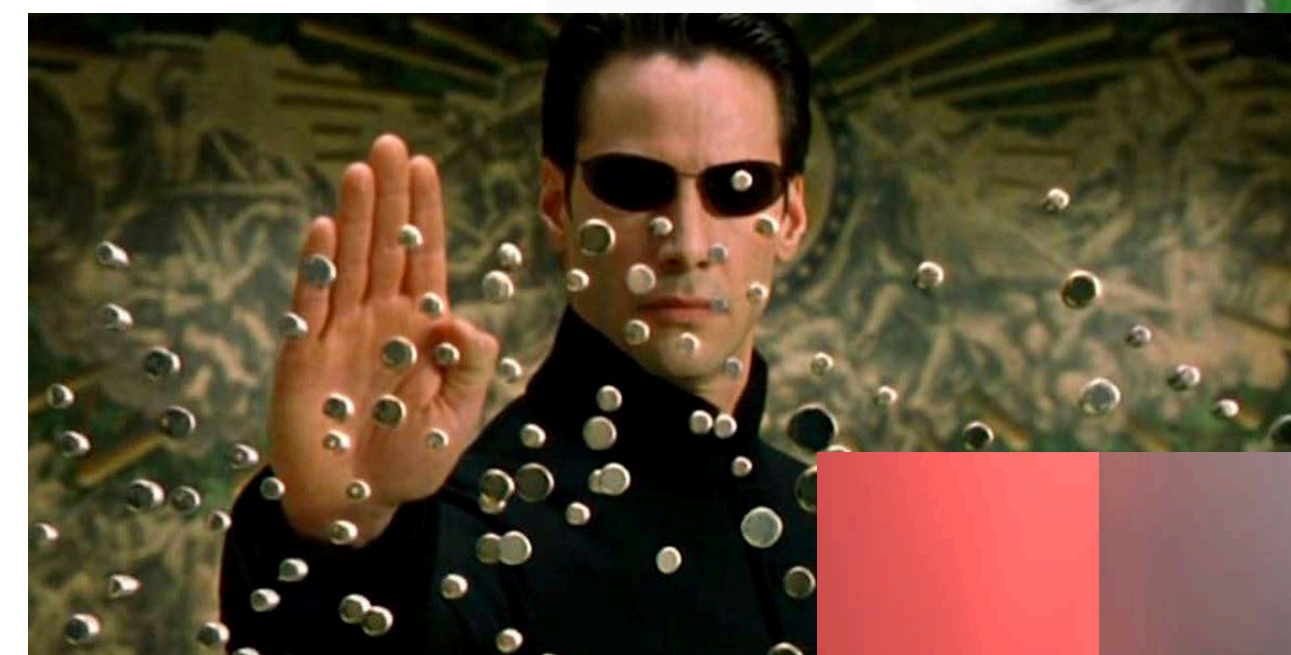
Perception of color

- Physical color, displayed color, perceived color



Physics-based animation

- Behind the scenes of special effects.

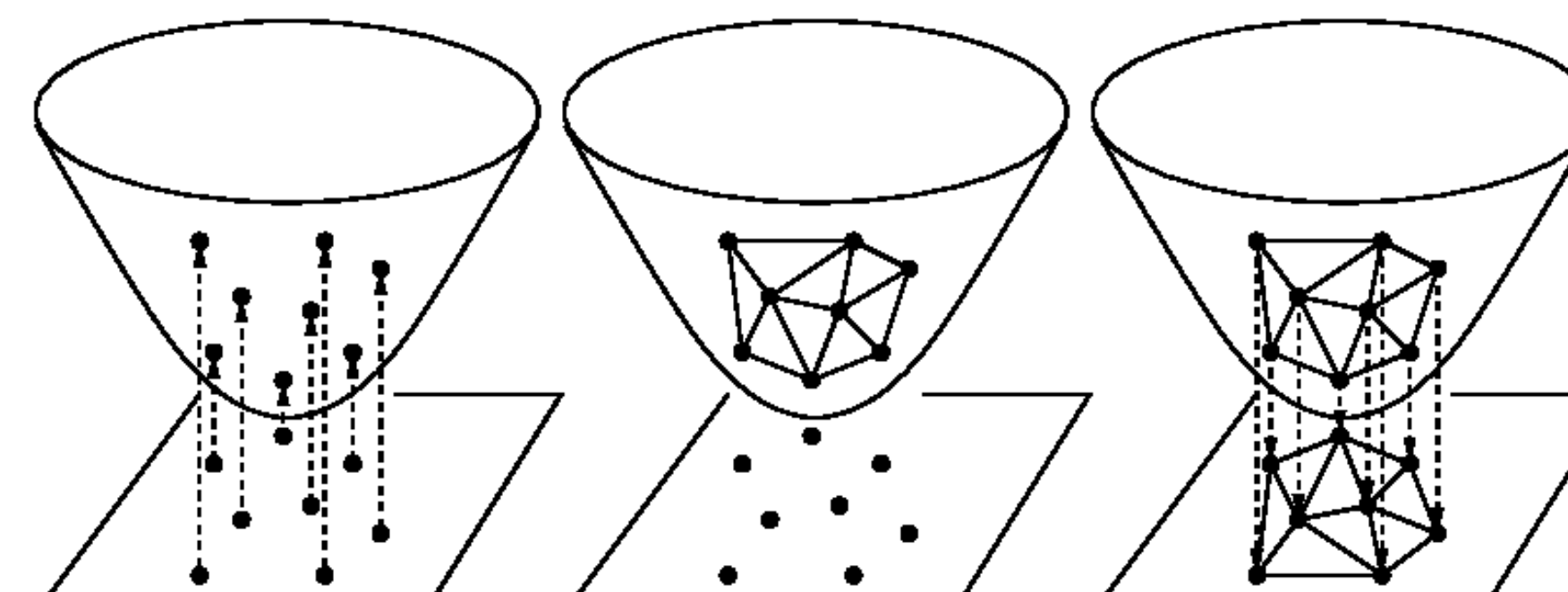


Optics

- Light transport equation

Geometry processing

- Differential geometry of discrete meshes



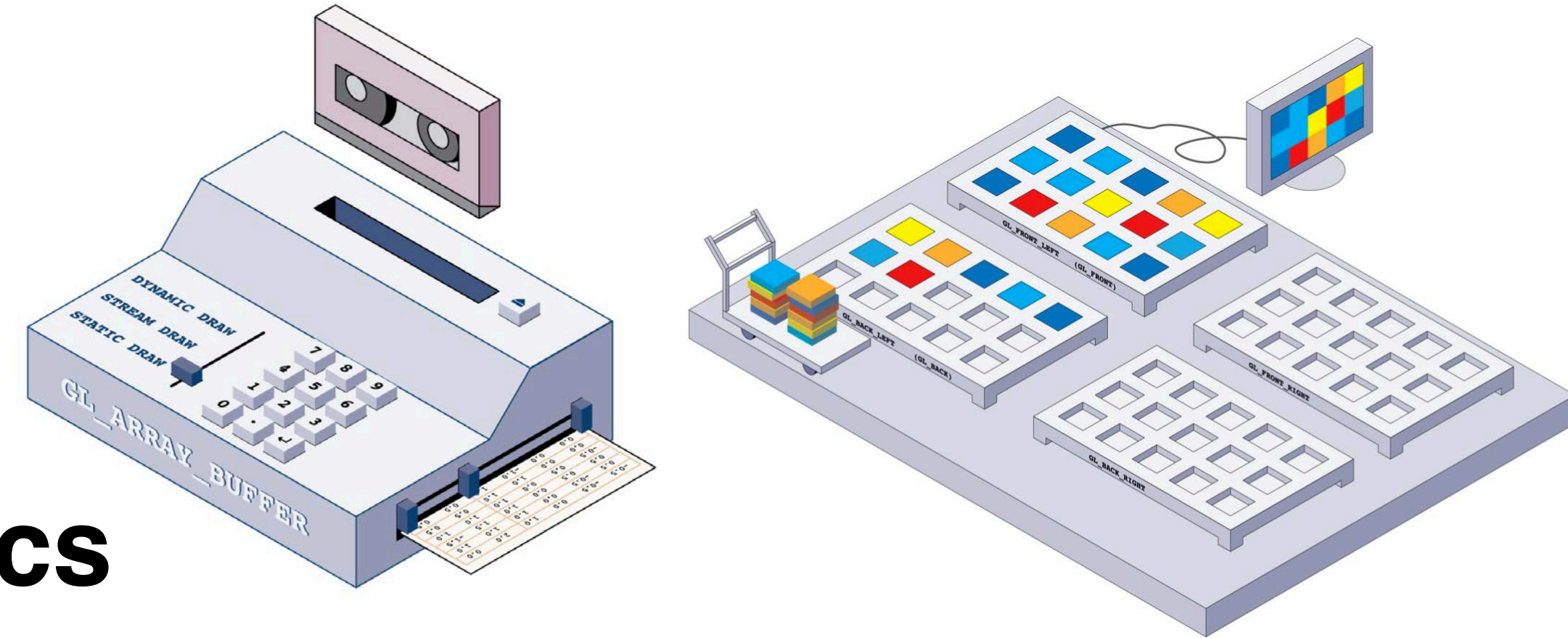
This course

Modern OpenGL

- Command the graphics card

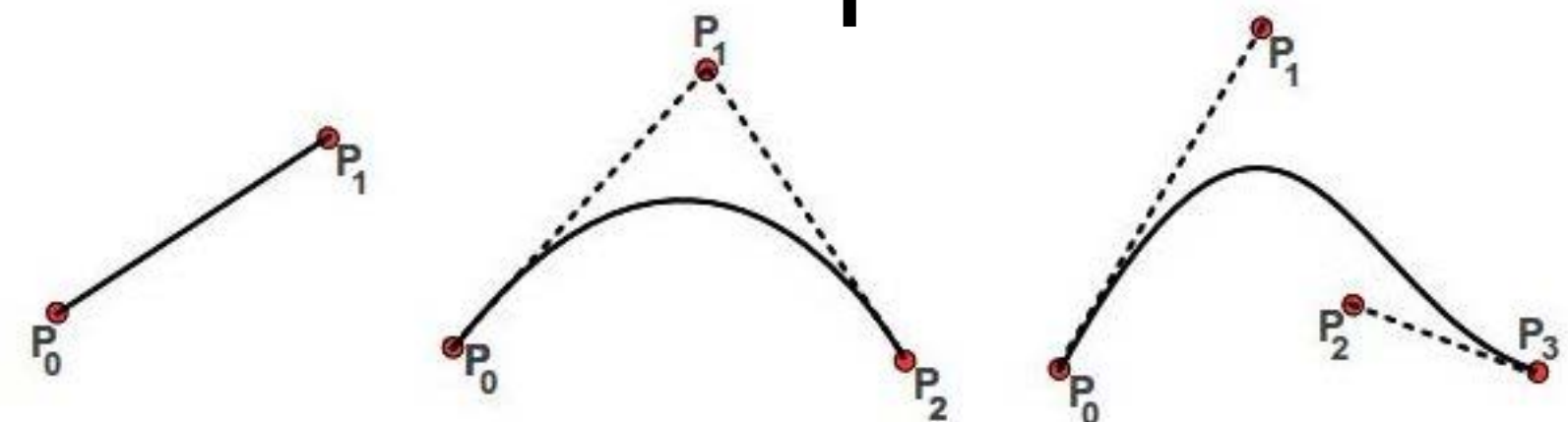
Foundation of 3D Computer Graphics

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



Foundation of Vector Graphics

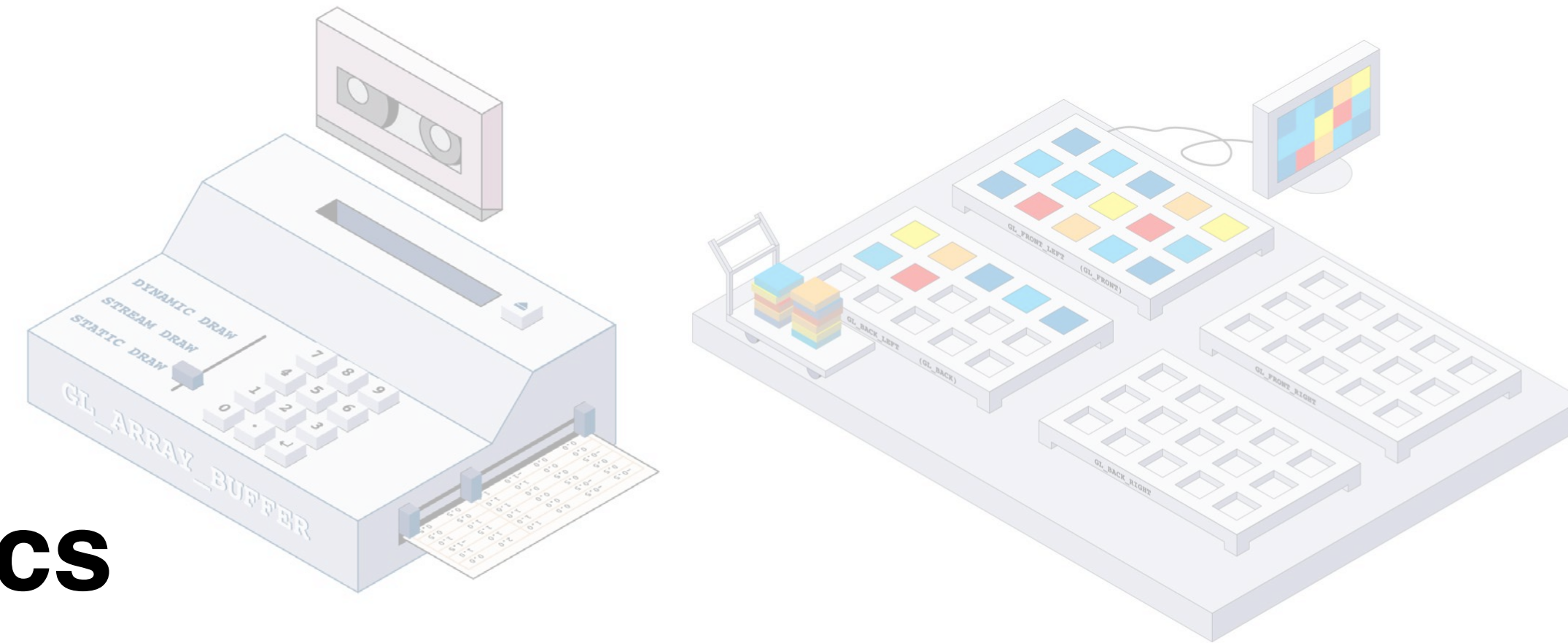
- Build smooth geometries from only a few control points



This course

Modern OpenGL

- Command the graphics card



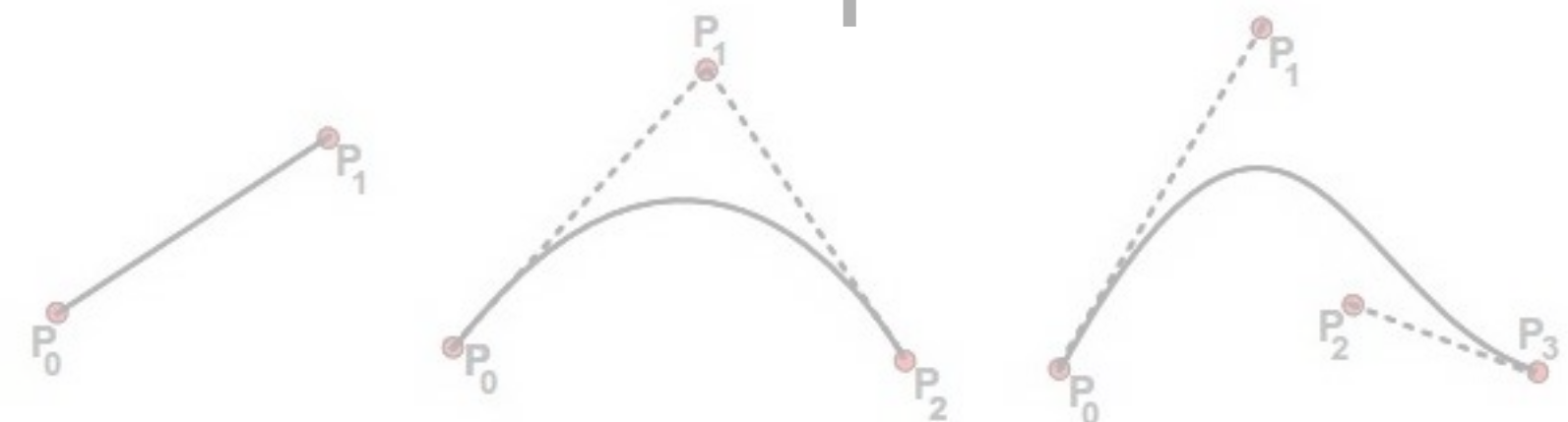
Foundation of 3D Computer Graphics

- 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

