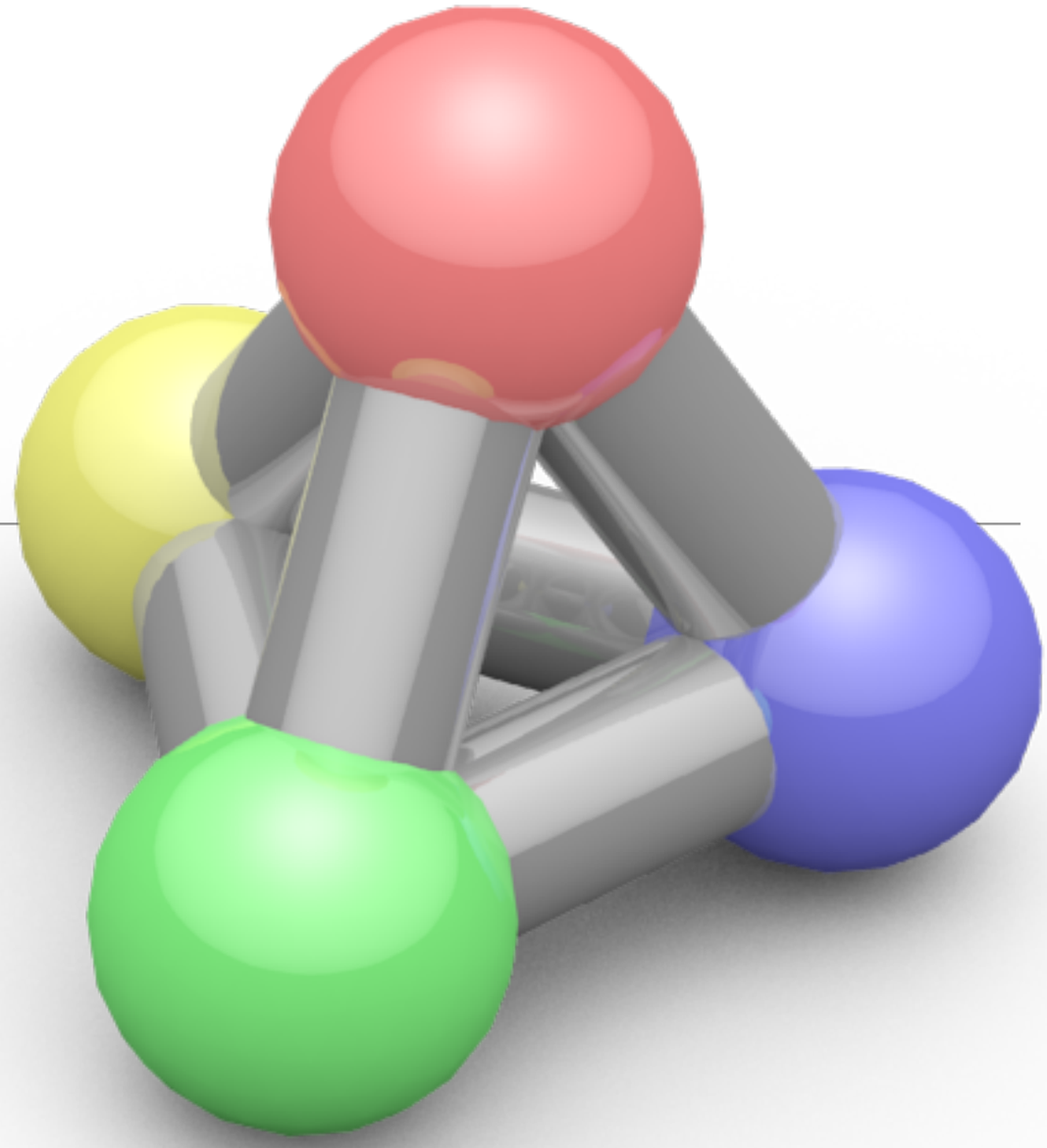
Graphics Systems



CPSC 453 – Fall 2016
Sonny Chan

Outline for Today

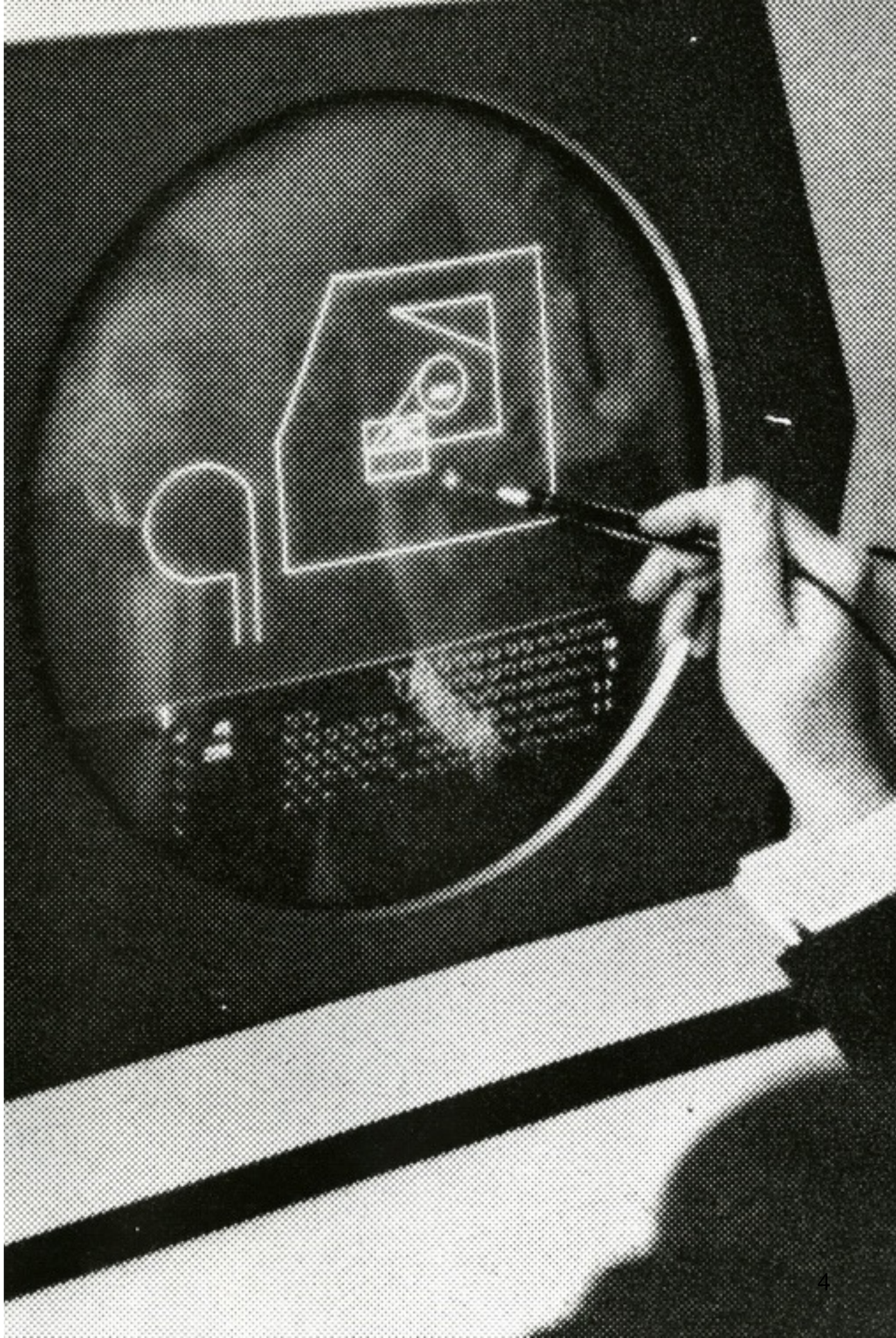
- A brief history of computer graphics systems
- The computer graphics pipeline
  - some notes on rasterization
- The OpenGL graphics system
  - what it is and what it isn’t

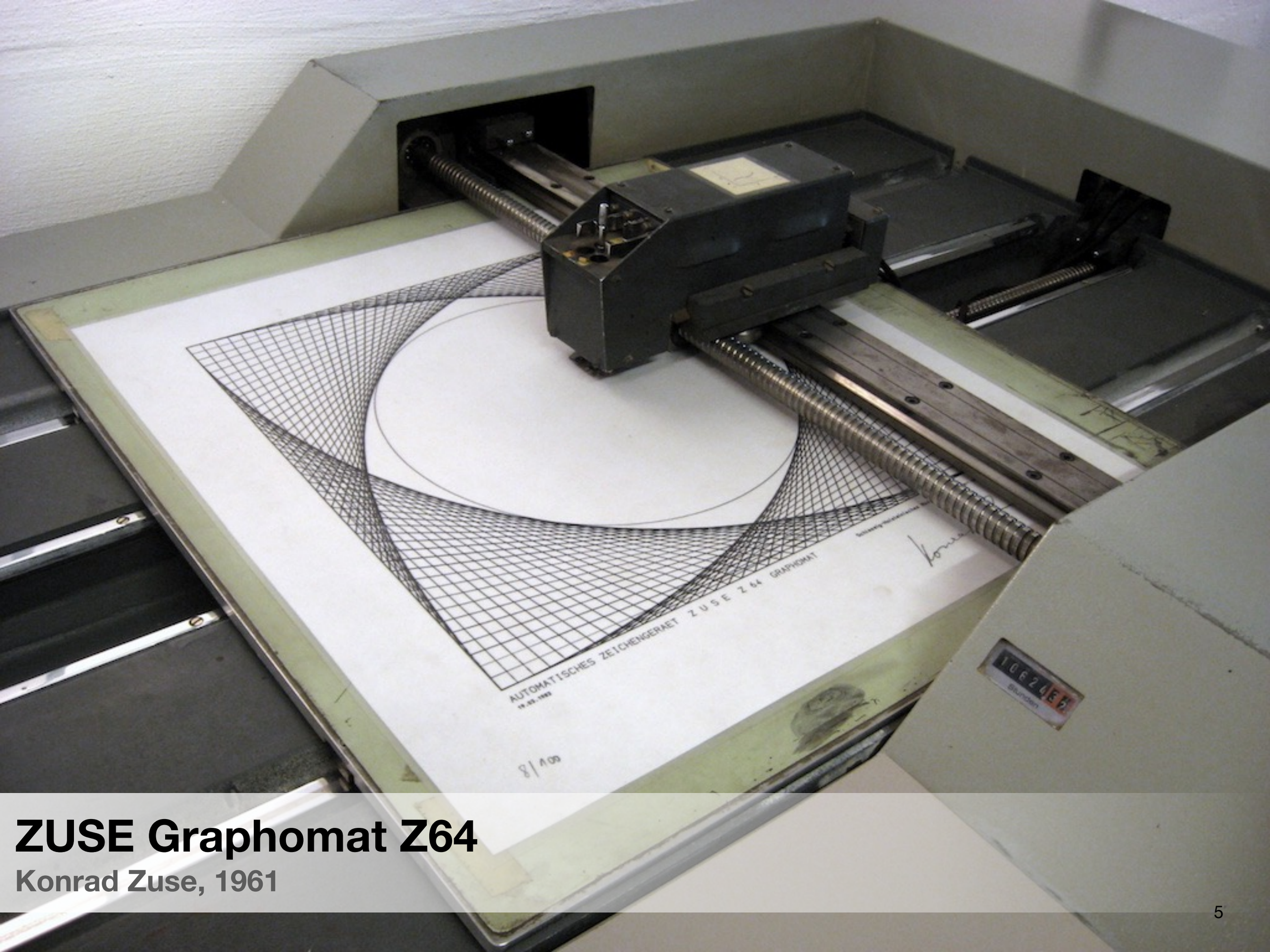
A Brief History

of computer graphics systems

Sketchpad

Ivan Sutherland, 1965

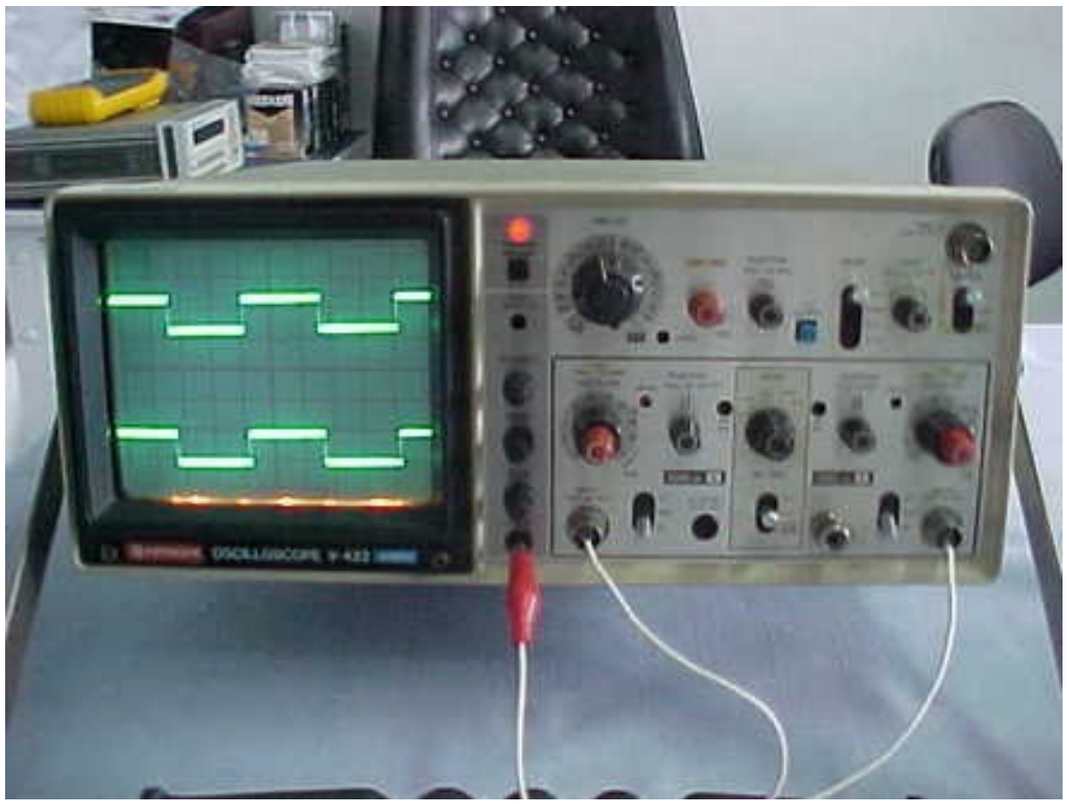


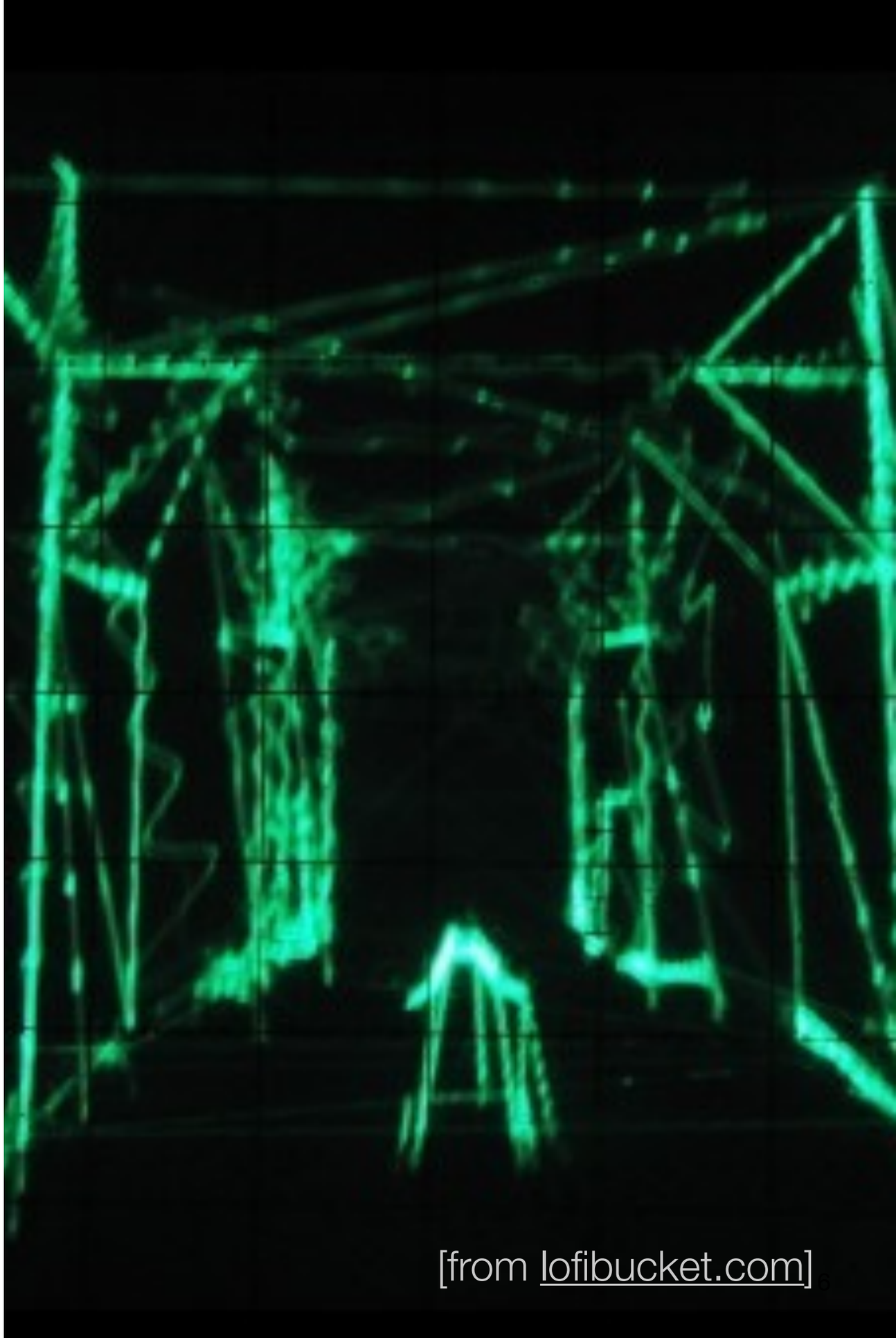


ZUSE Graphomat Z64

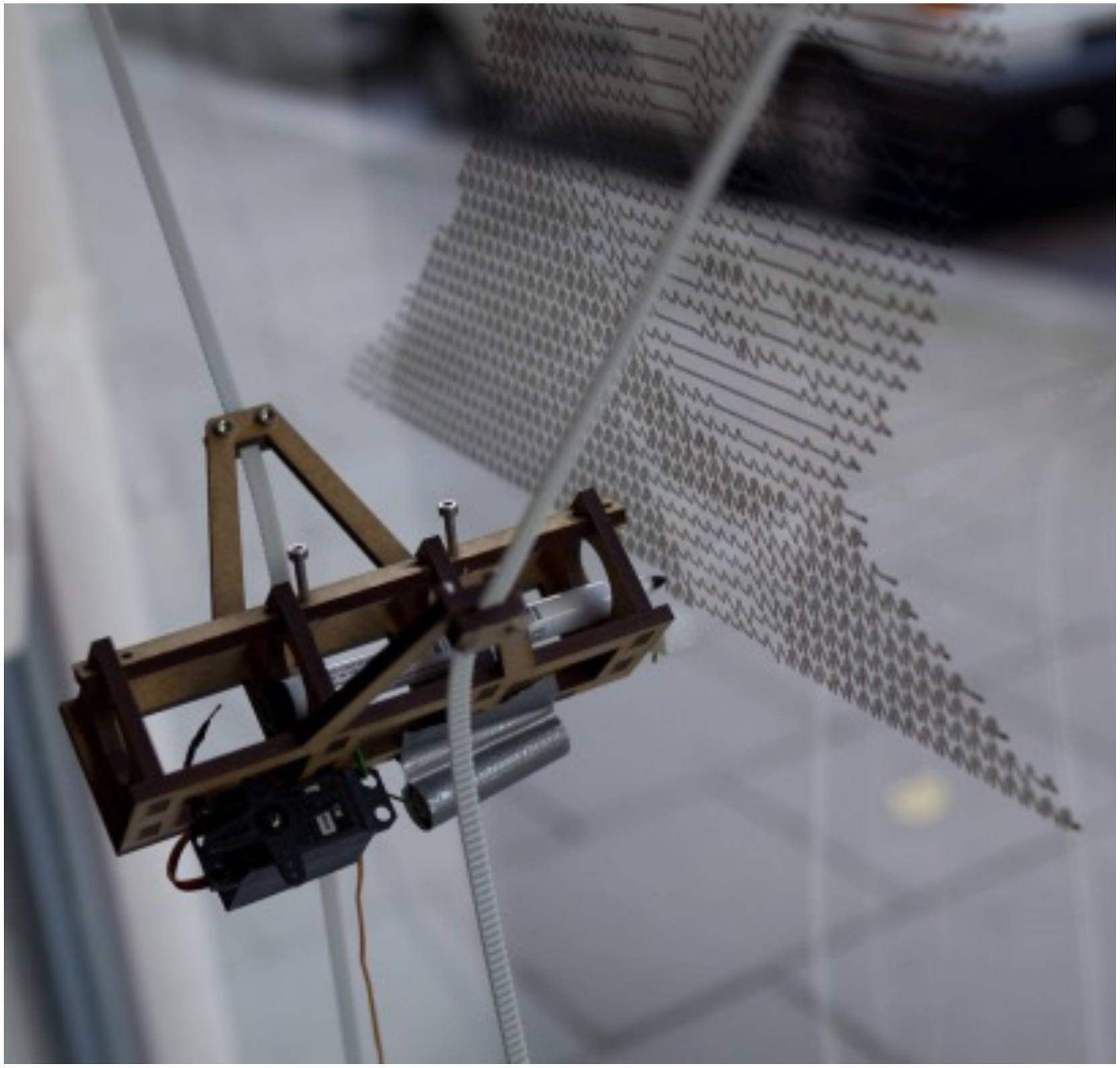
Konrad Zuse, 1961

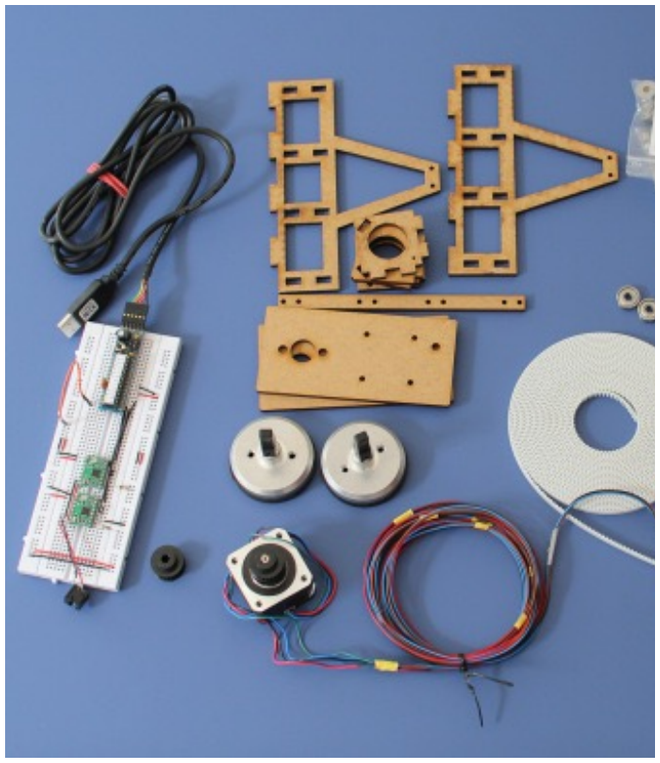
Cathode Ray Tube

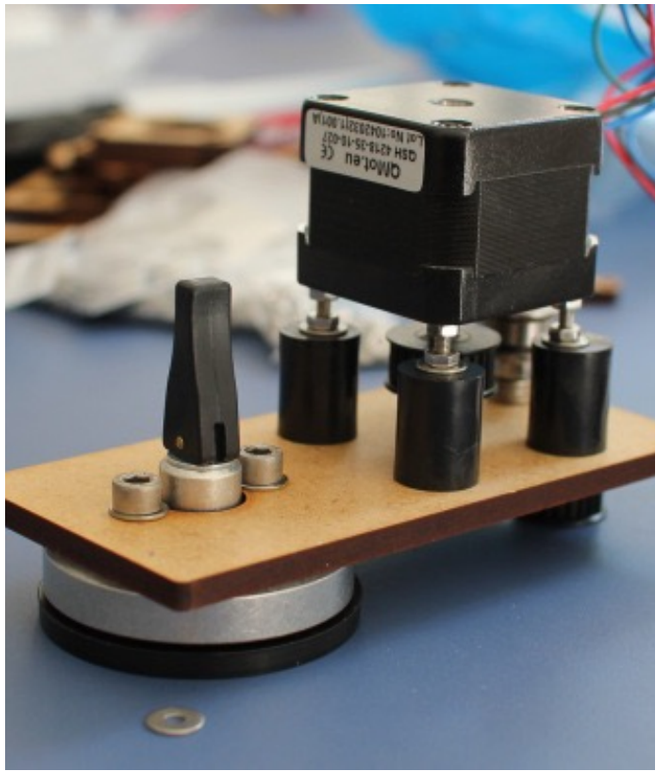




[from lofibucket.com]







Der Kritzler, Alex Weber

Controlling a Plotter

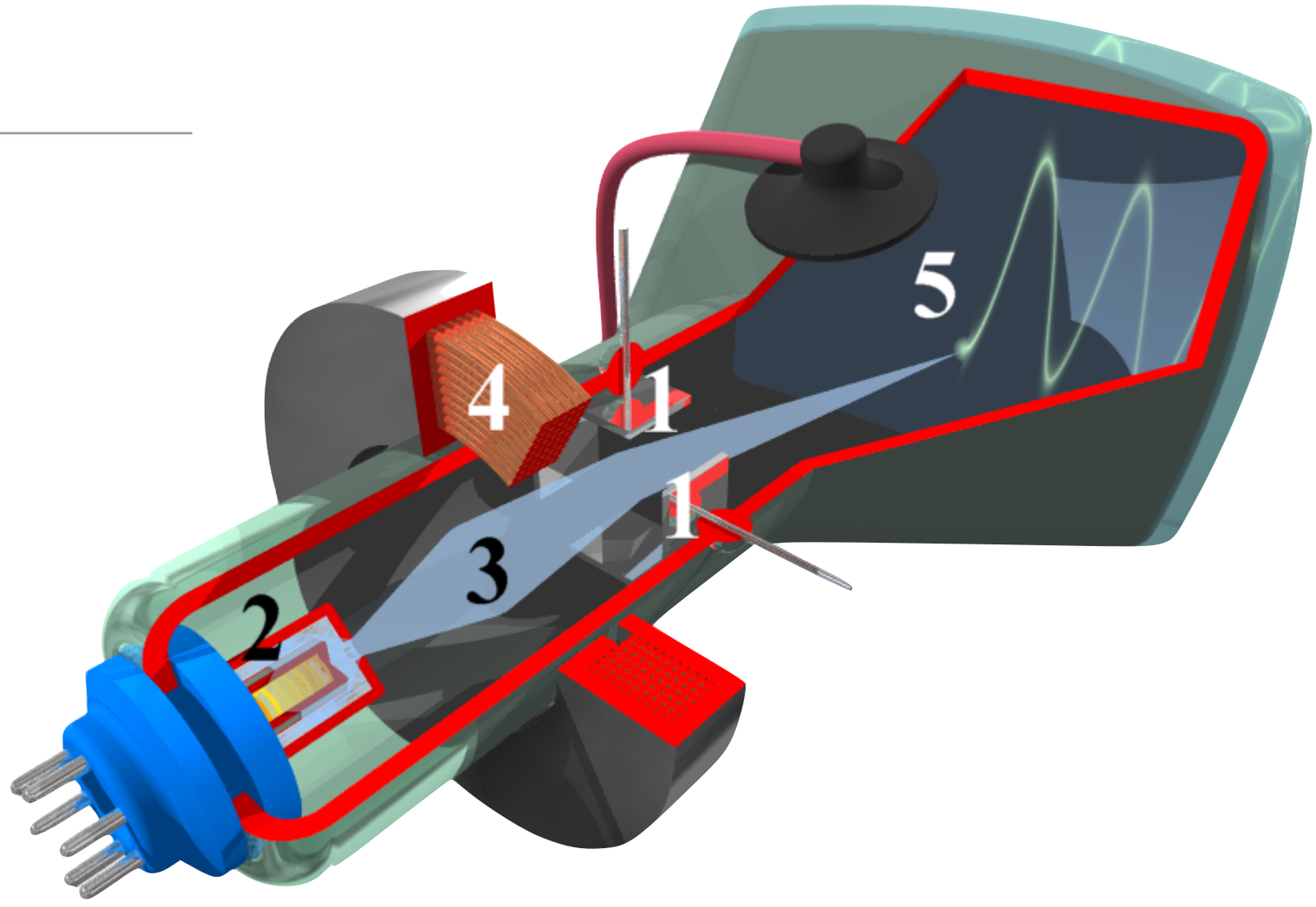
Hewlett Packard Graphics Language

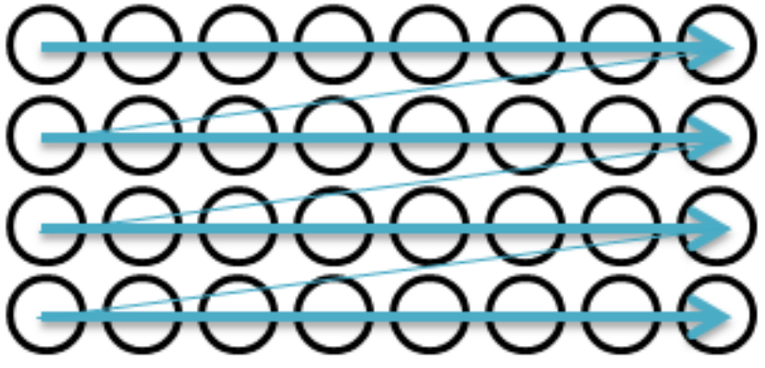
```
IN;
PU;
SP3;
PU940,5104;
PD856,5104,772,5100,688,5092,…,940,5104;
PU;
SP2;
PU940,3928;
PD1024,3928,1104,3920,1184,3912,…,940,3928;
PU;
SP1;
PU-6784,5164;
PD-6864,4876,-6916,4876,-6972,5084,…,-6784,5164;
PU-6704,5116;
PD-6752,5116,-6752,5164,-6704,5164,-6704,5116;
PU-6704,4876;
PD-6752,4876,-6752,5088,-6704,5088,-6704,4876;
PU-6468,4876;
PD-6516,4876,-6516,5008,-6520,5024,…,-6468,4876;
PU-6232,4876;
PD-6408,4876,-6408,5164,-6356,5164,…,-6232,4876;
PU-6140,4876;
PD-6192,4876,-6192,5164,-6140,5164,-6140,4876;
PU-5860,4876;
PD-5912,4876,-6020,5060,-6028,5072,…,-5860,4876;
PU-5620,4876;
PD-5804,4876,-5804,5164,-5620,5164,…,-5620,4876;
PU-6900,4036;
…
PU;
SP;
IN;
```

[excerpted from winline.com]

Raster Scanning

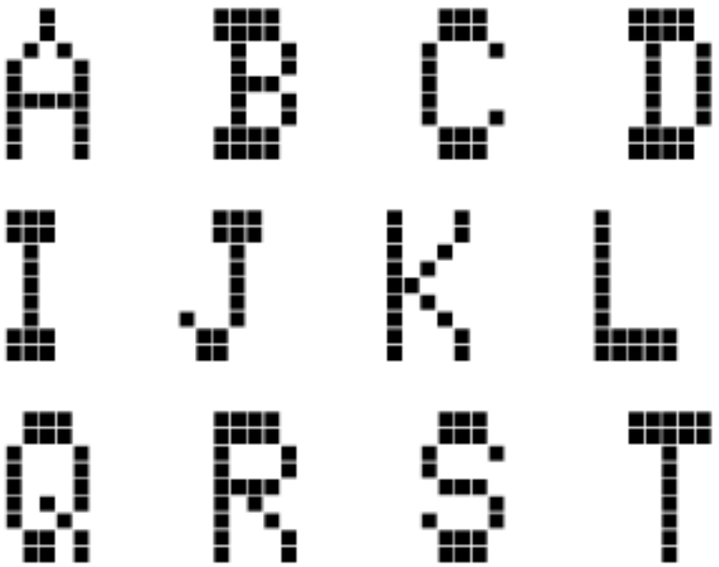
1. Deflection voltage electrode;
2. Electron gun;
3. Electron beam;
4. Focusing coil;
5. Phosphor-coated inner side of the screen





Bit-Mapped Displays

- The CRT electron beam scans out the contents of a memory buffer
  - 0 = off, 1 = on
- Why did raster displays take so long to appear?



Super Mario Bros.

Nintendo, 1985

Clock

Settings

Program Manager

File Options Window Help

Main

File Manager

Control Panel

Print Manager

ClipBook Viewer

MS-DOS Prompt

Windows Setup

PIF Editor

Read Me

File Manager - [C:\WINDOWS*.

File Disk Tree View

Options Tools Window Help

C:

a c

dos

old_do

windov

sys

system

_default.pif

accessor.grp

applicat.grp

arcade.bmp

argyle.bmp

awcas.dll

C: 477MB free, 498MB total

Total

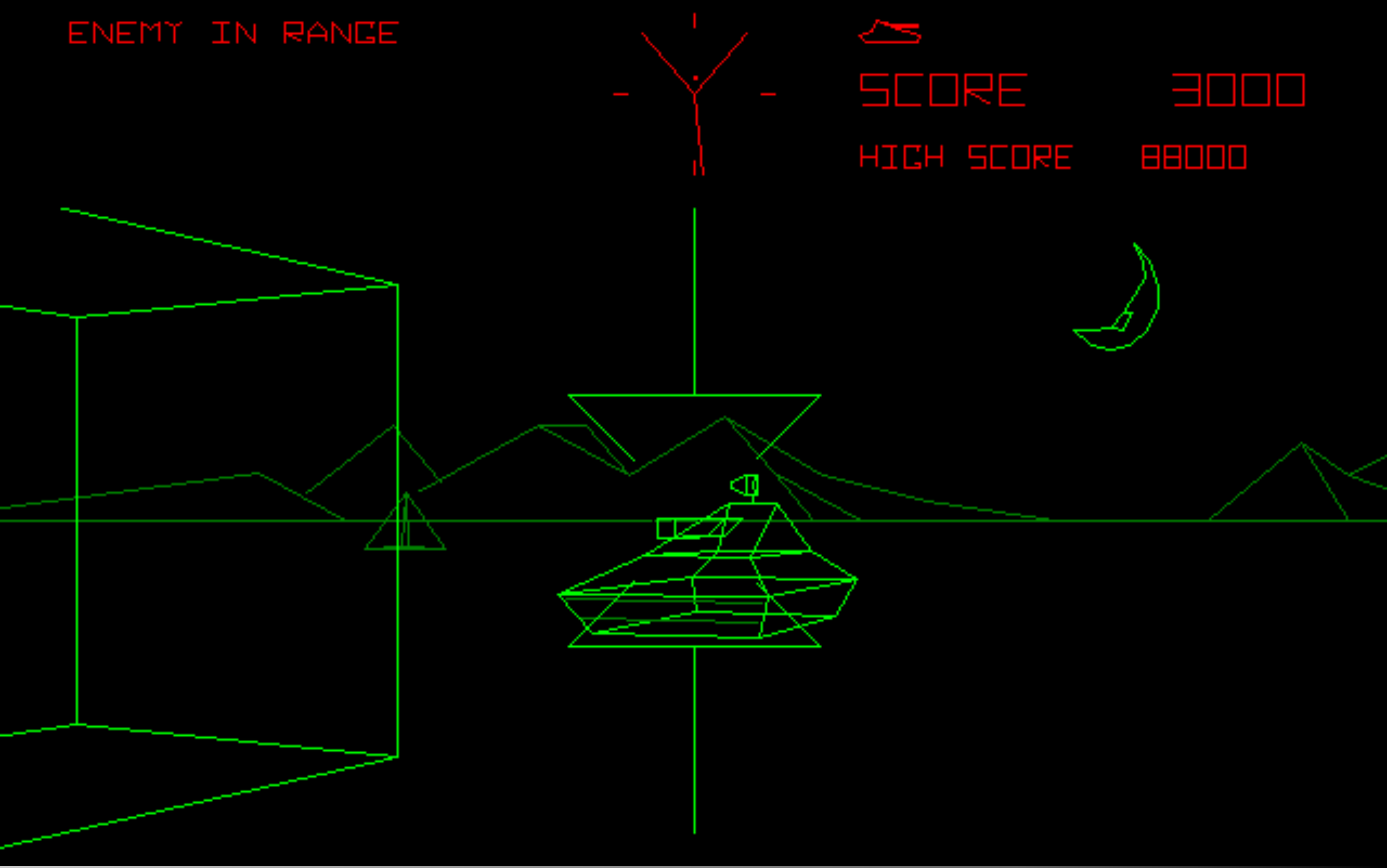
Games

StartUp

Applica

Minesweeper

Game Help



Battlezone

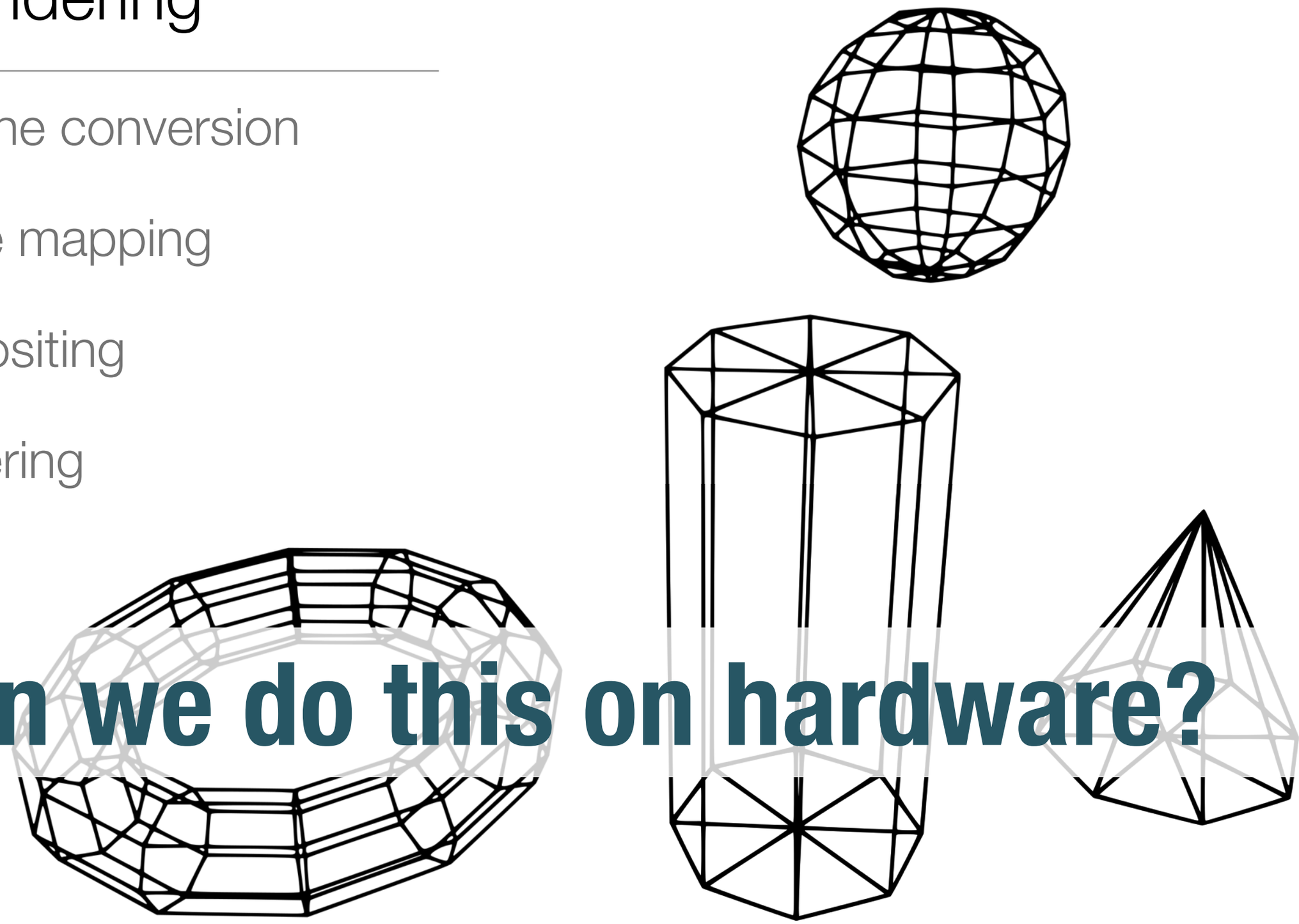
Atari, 1980

Wolfenstein 3D

id Software, 1992

3D Rendering

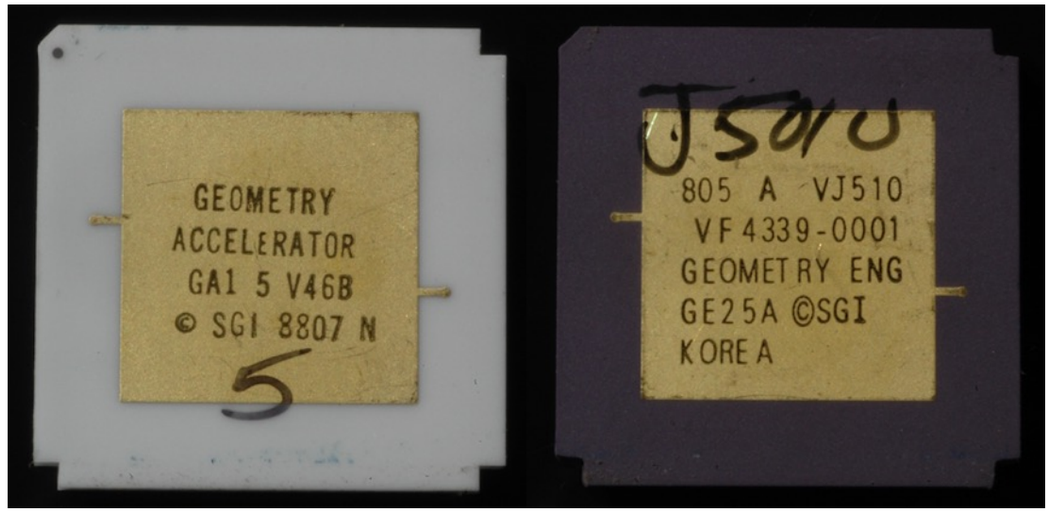
- Scan line conversion
- Texture mapping
- Compositing
- Z-buffering



Can we do this on hardware?

Silicon Graphics, Inc.

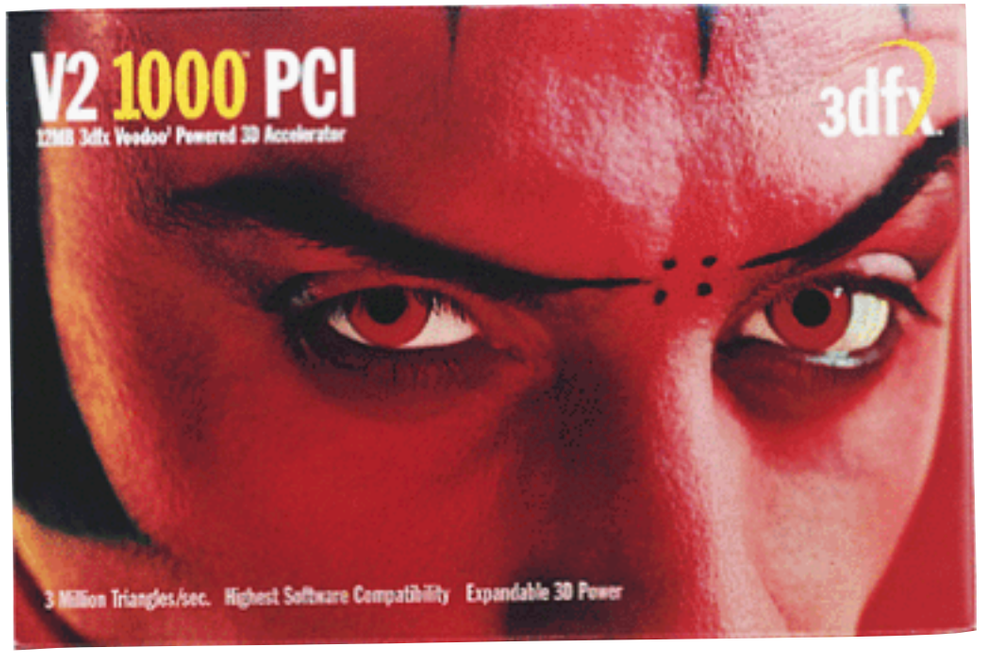
James H. Clark, 1982

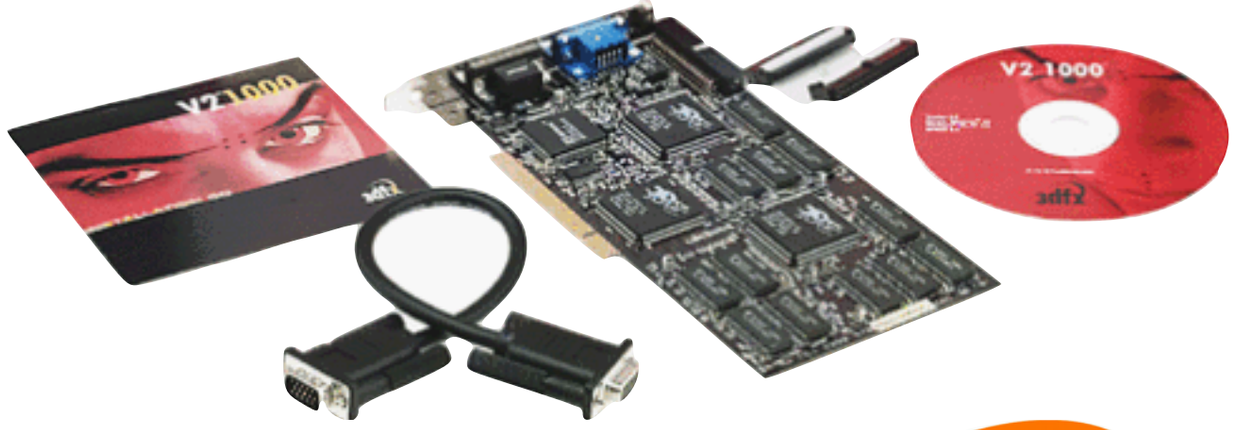


[from rcsri.org]

Hardware Acceleration

for the consumer market, 1994

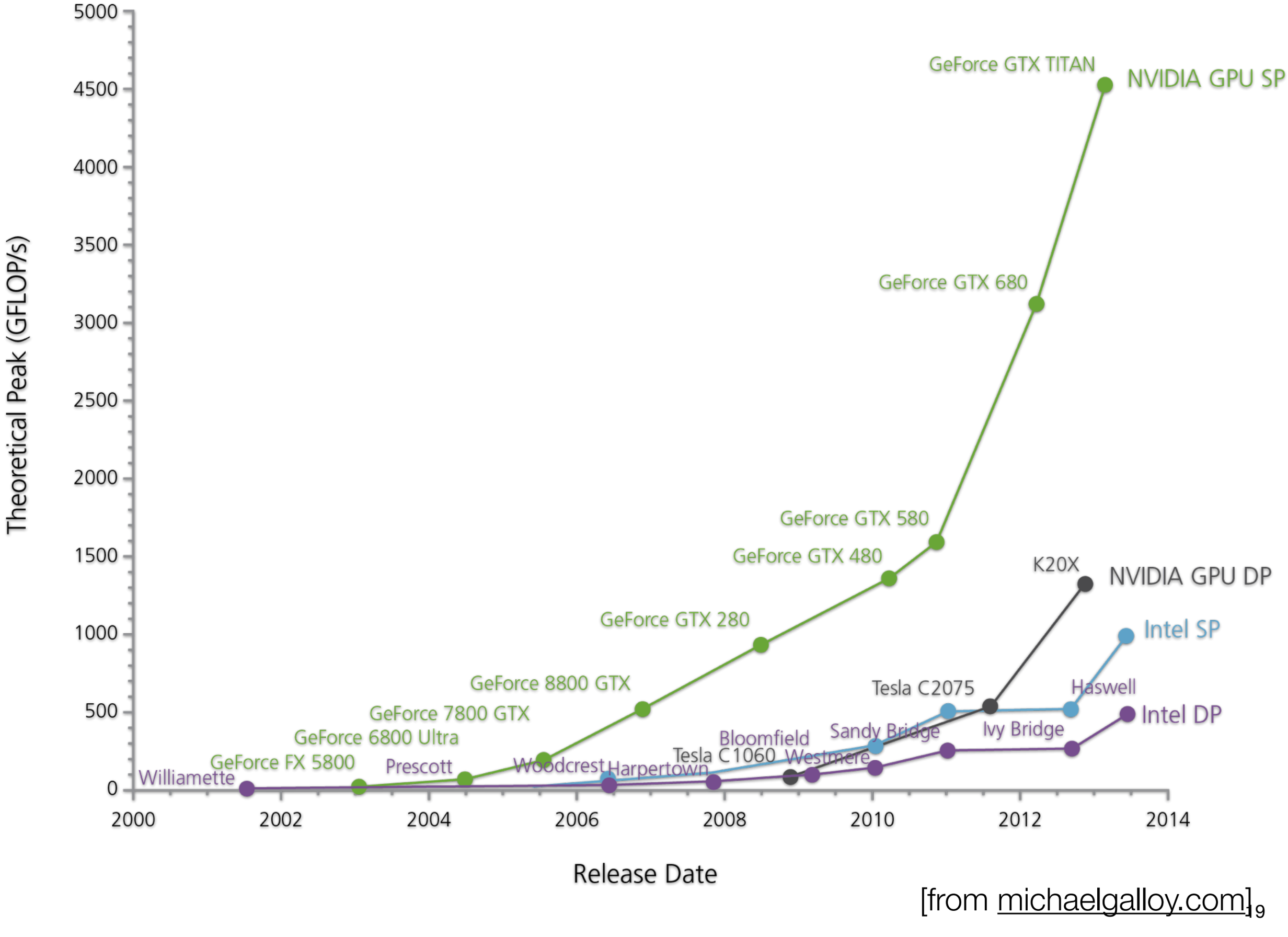




3dfx™

CPU vs GPU

Quake, id Software, 1996



[from michaelgalloy.com]

A Modern Graphics System

NVIDIA GeForce GTX GPUs

Intel Core i7 Extreme

64 GB memory

8 cores

350 GigaFLOPS

NVIDIA GeForce GTX TITAN Z

12 GB memory

5760 cores

8 TeraFLOPS

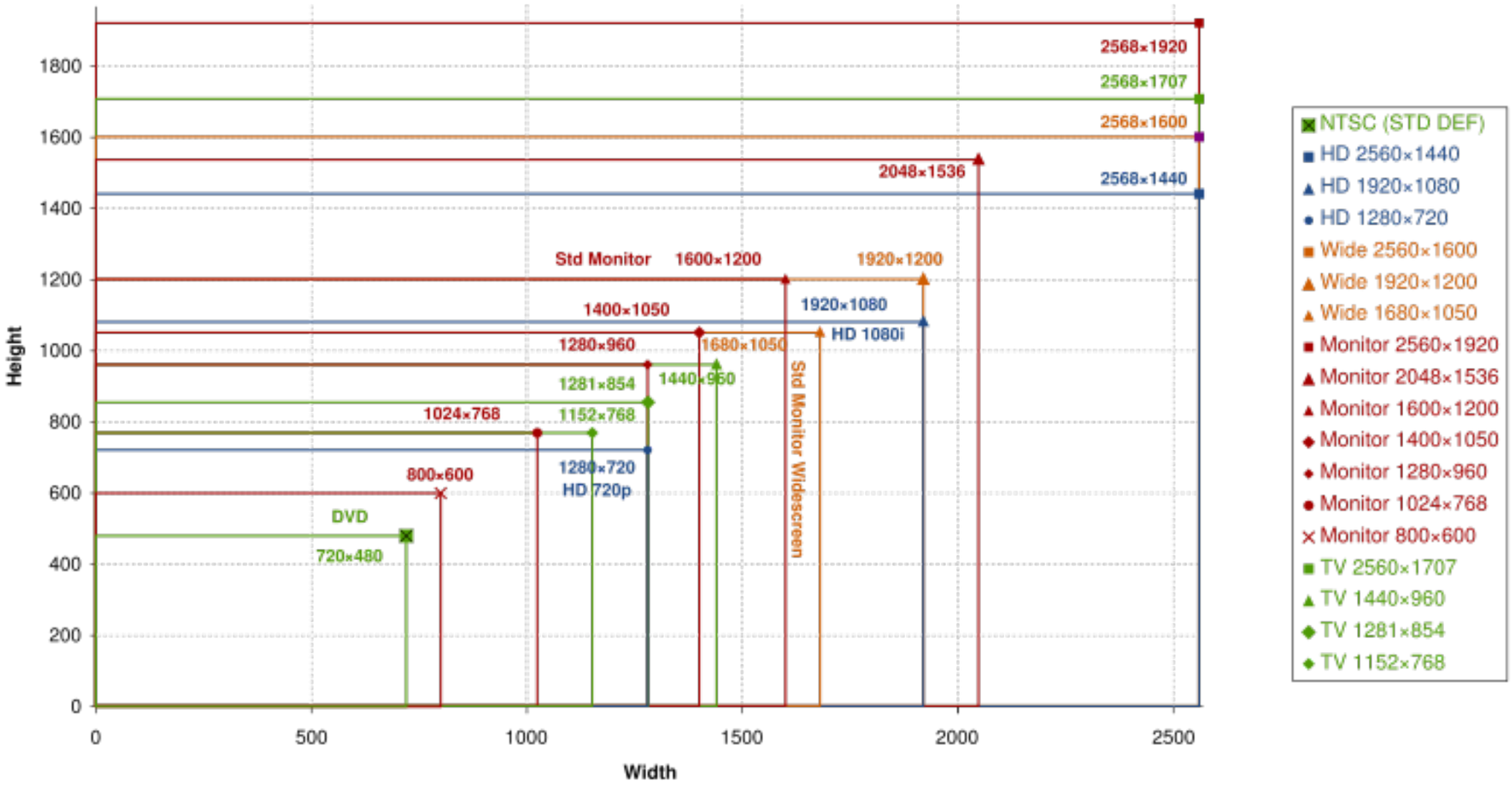
Do we really need all those cores?

Standard monitor 4:3

TV 3:2

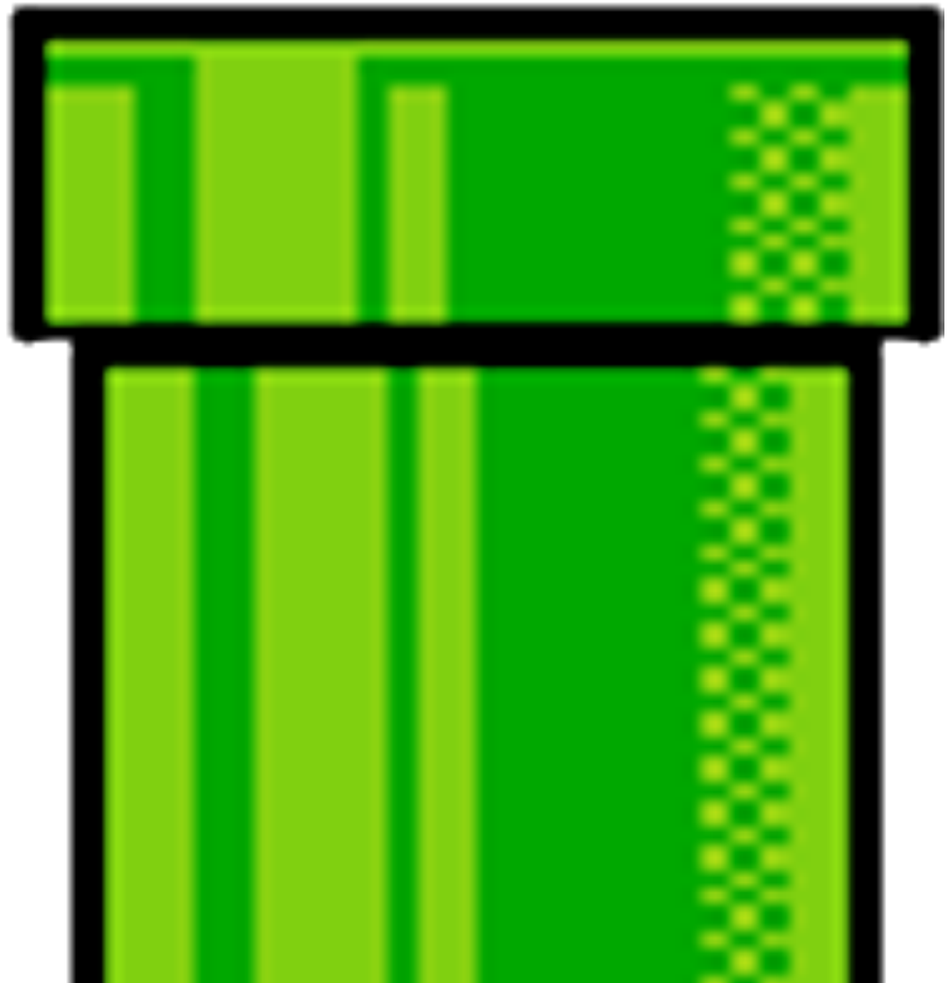
Widescreen monitor 16:10

HD 16:9



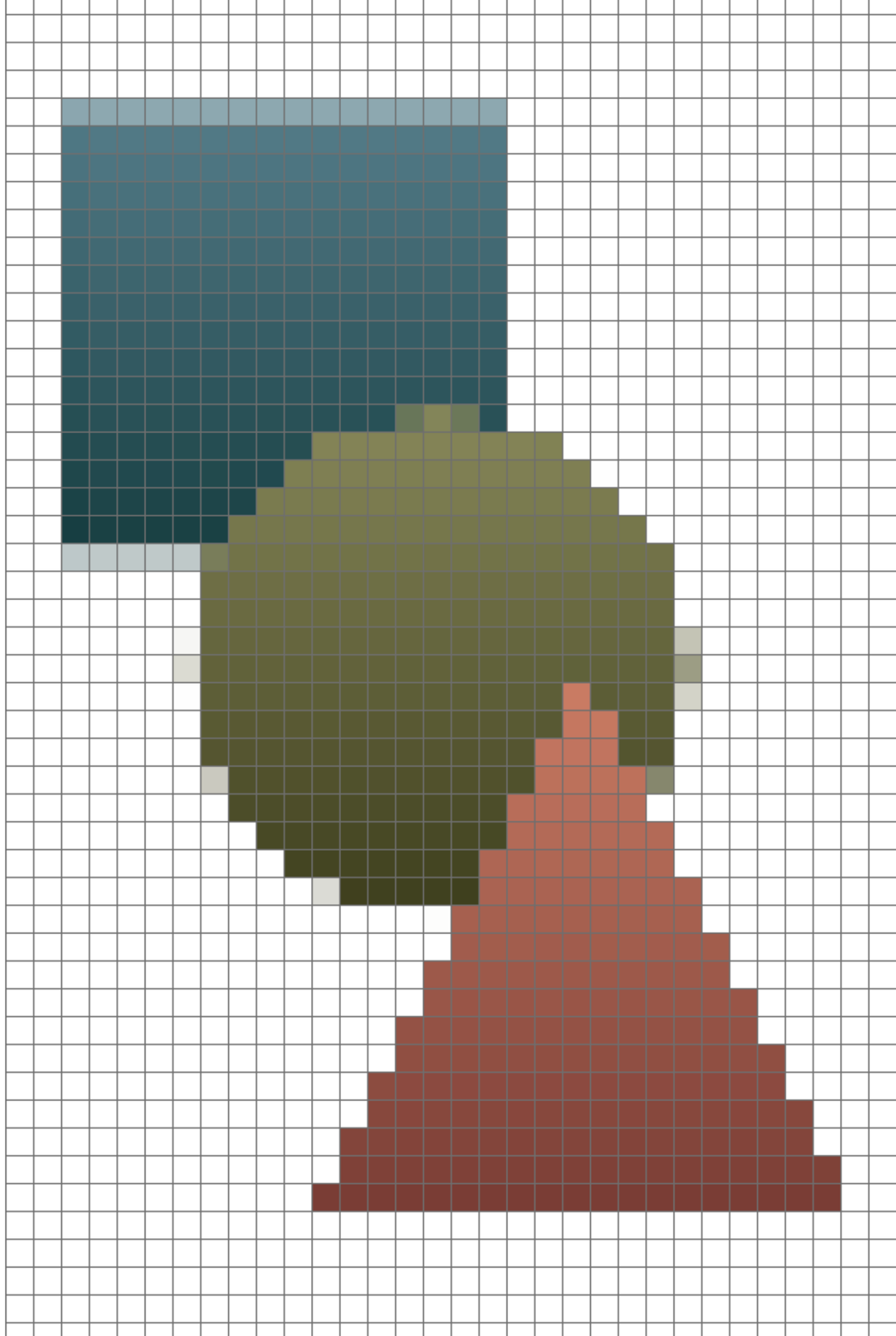
CGA	320 × 200	64k pixels
VGA	640 × 480	307k pixels
XGA	1024 × 768	0.8M pixels
HD 1080	1920 × 1080	2M pixels
WQXGA	2560 × 1600	4M pixels
UHD-1	3840 × 2160	8M pixels

The Graphics Pipeline

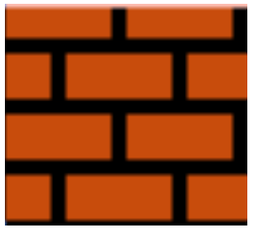


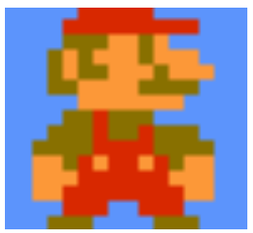
What exactly is a pipeline?

Computer Graphics



In the early days…

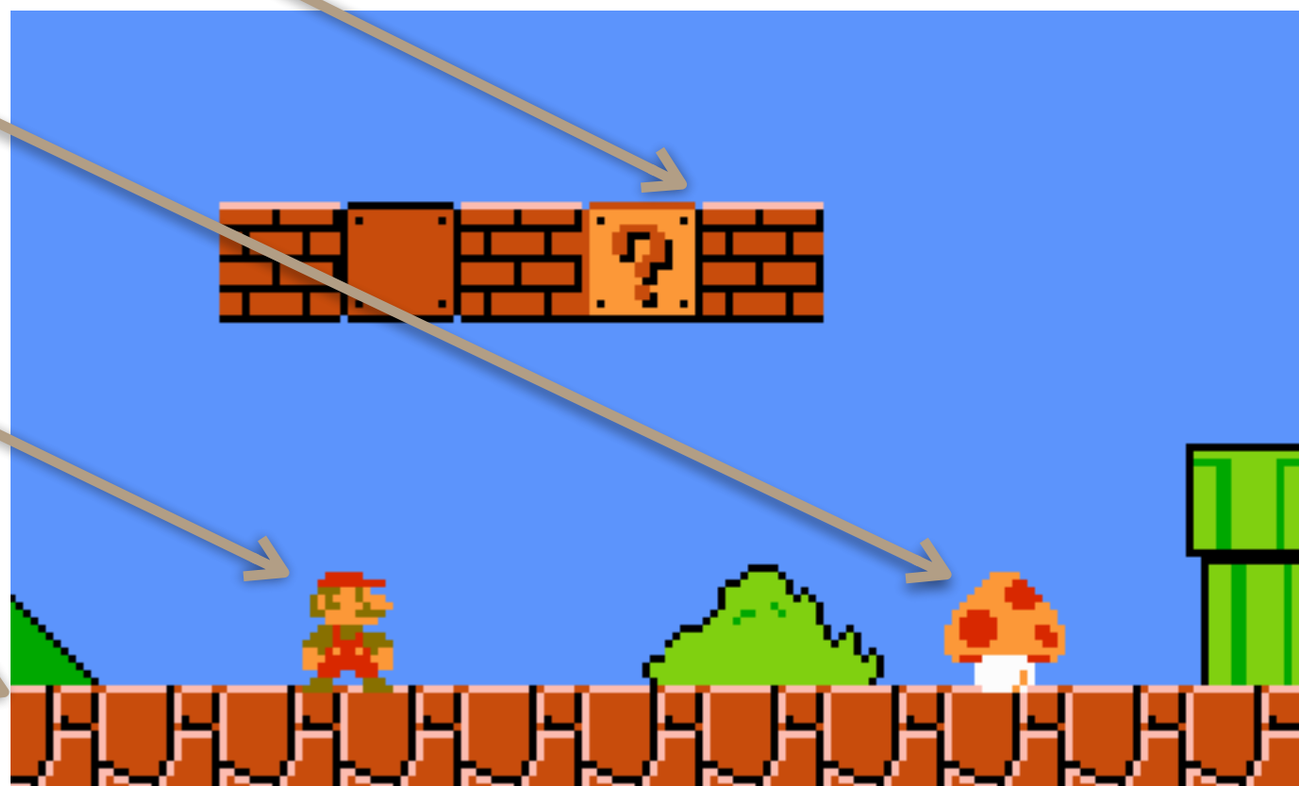




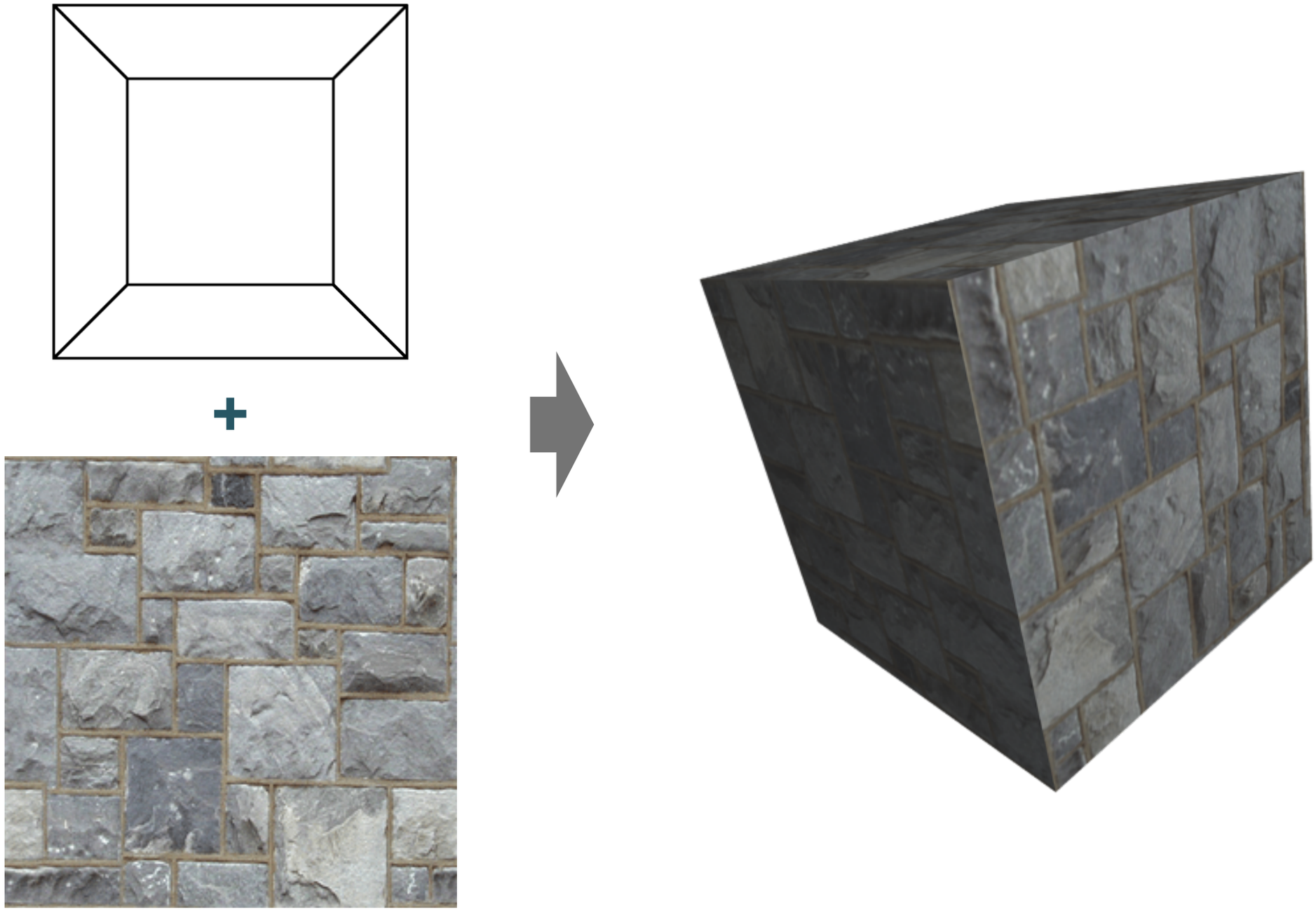
algorithms

data

```
0000000 0000 0001 0001 1010
0000010 0000 0016 0000 0028
0000020 0000 0001 0004 0000
0000030 0000 0000 0000 0010
0000040 0004 8384 0084 c7c8
0000050 00e9 6a69 0069 a8a9
0000060 00fc 1819 0019 9898
0000070 0057 7b7a 007a bab9
0000080 8888 8888 8888 8888
0000090 3b83 5788 8888 8888
00000a0 d61f 7abd 8818 8888
00000b0 8b06 e8f7 88aa 8388
```

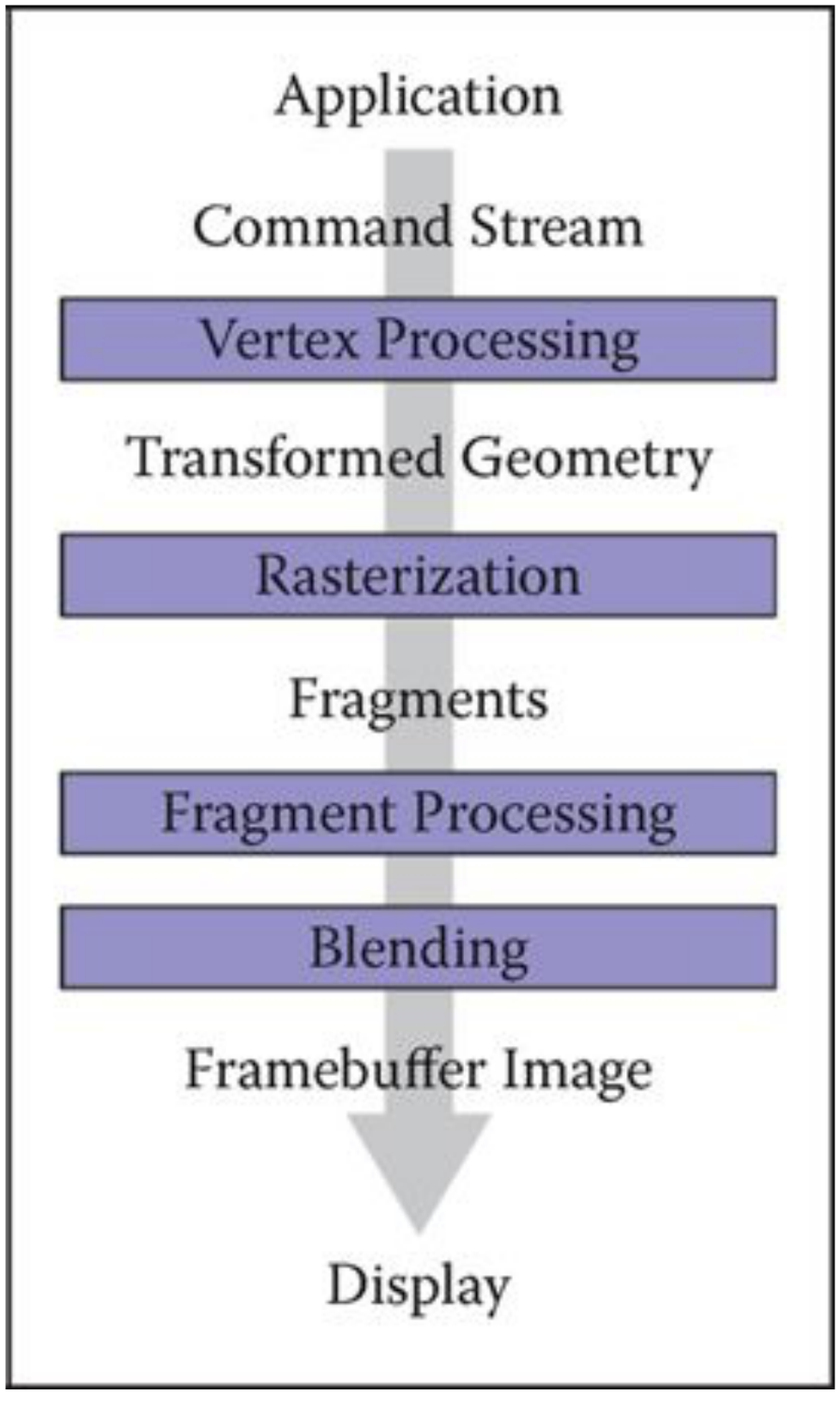


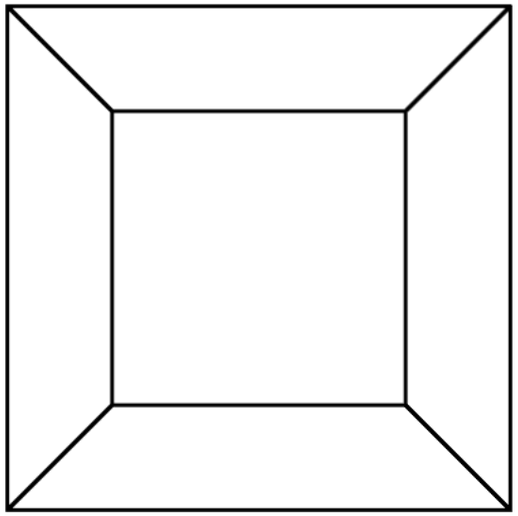
Three-Dimensional Graphics

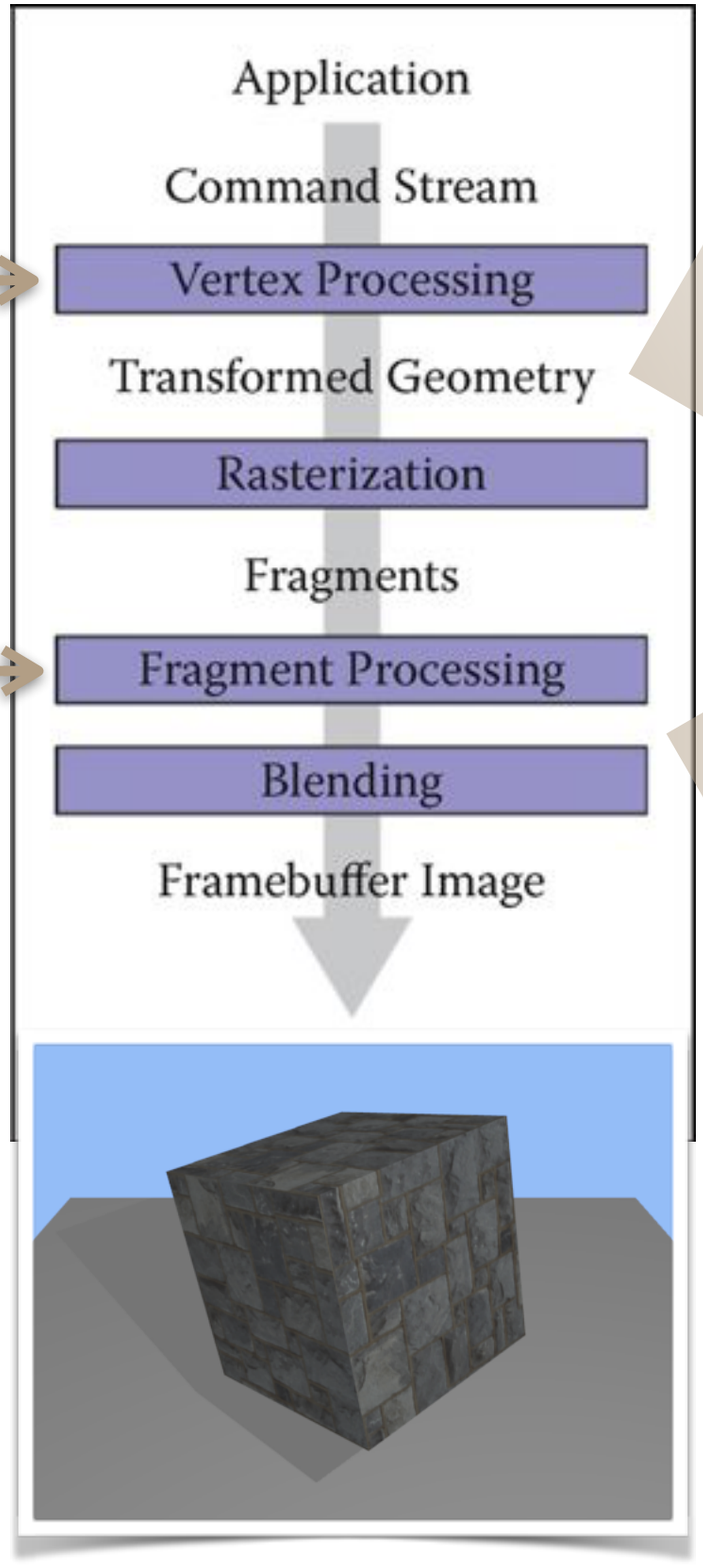


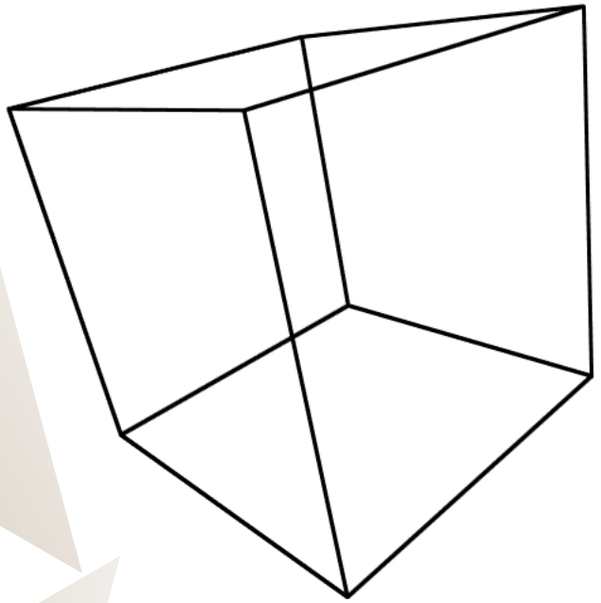
The Graphics Pipeline

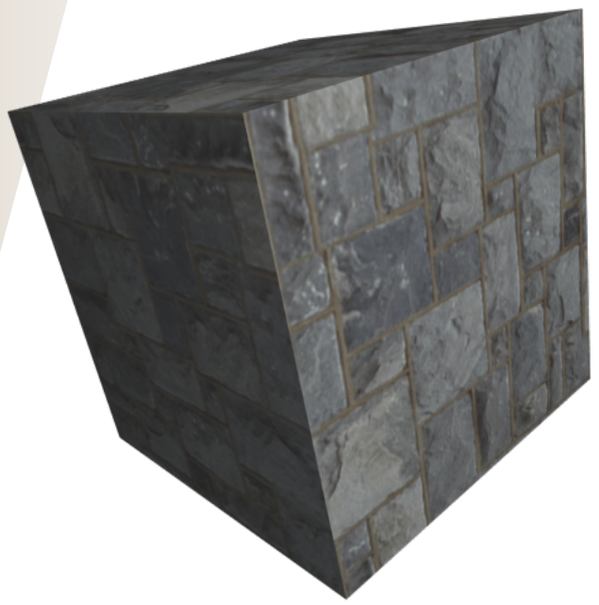
… according to your textbook



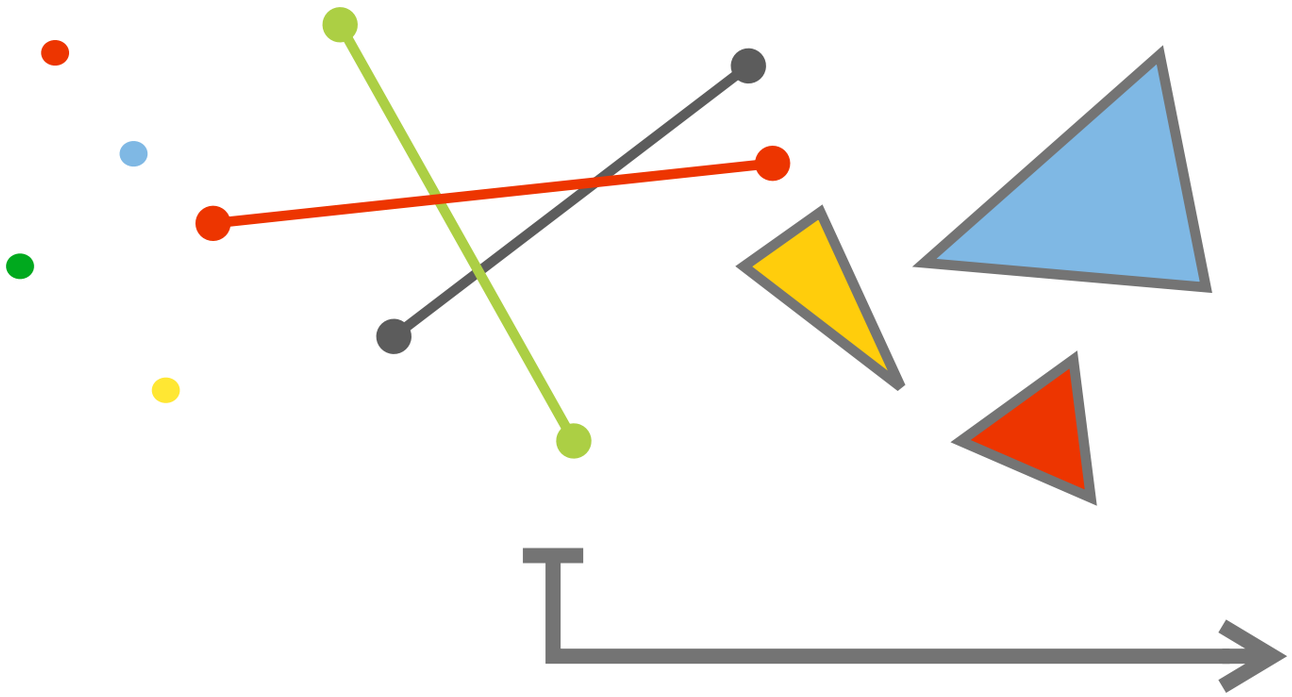



Application
Command Stream
Vertex Processing
Transformed Geometry
Rasterization
Fragments
Fragment Processing
Blending
Framebuffer Image

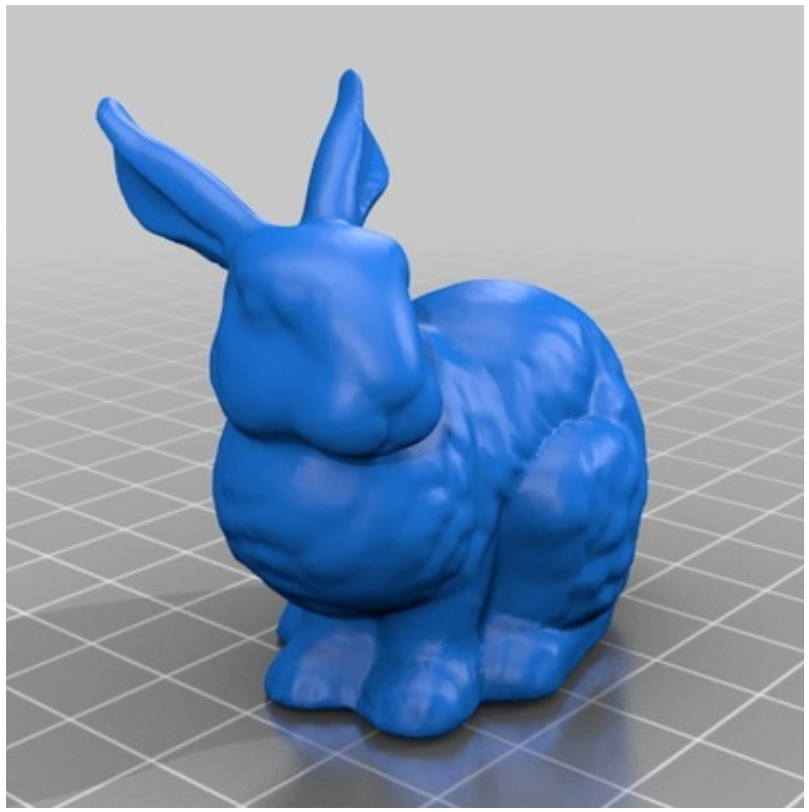




Rasterization / Scan Conversion



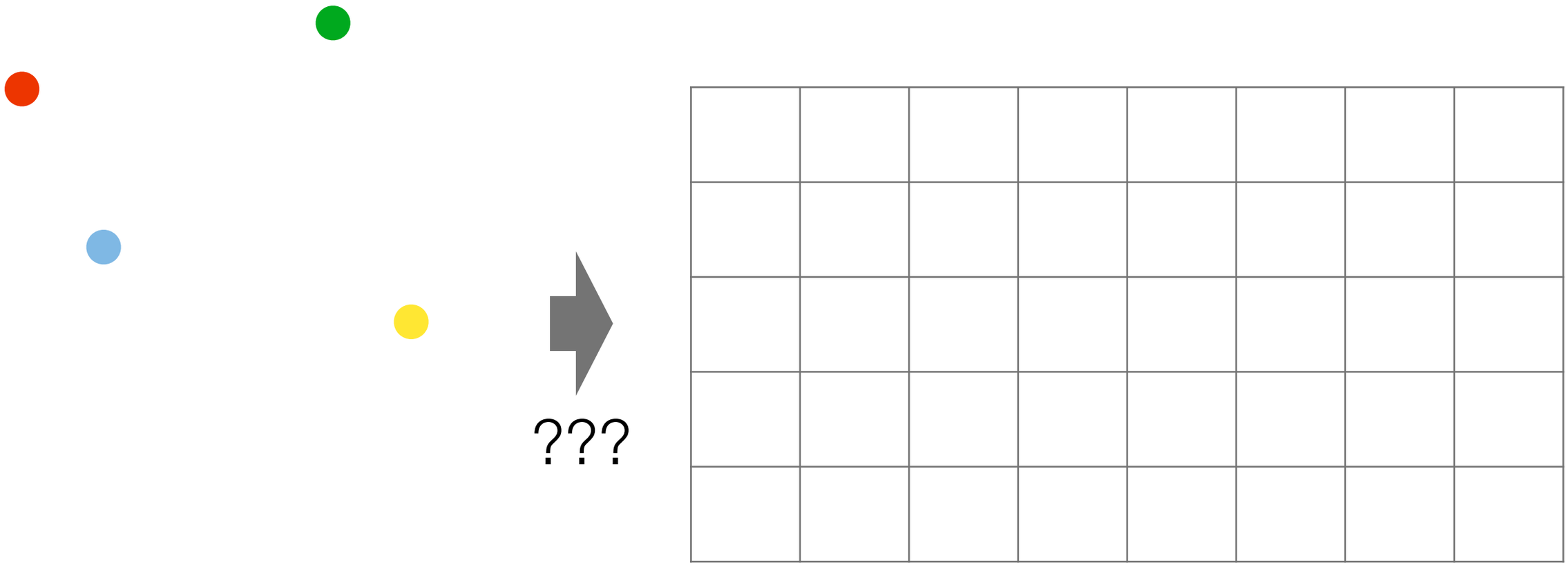
have this



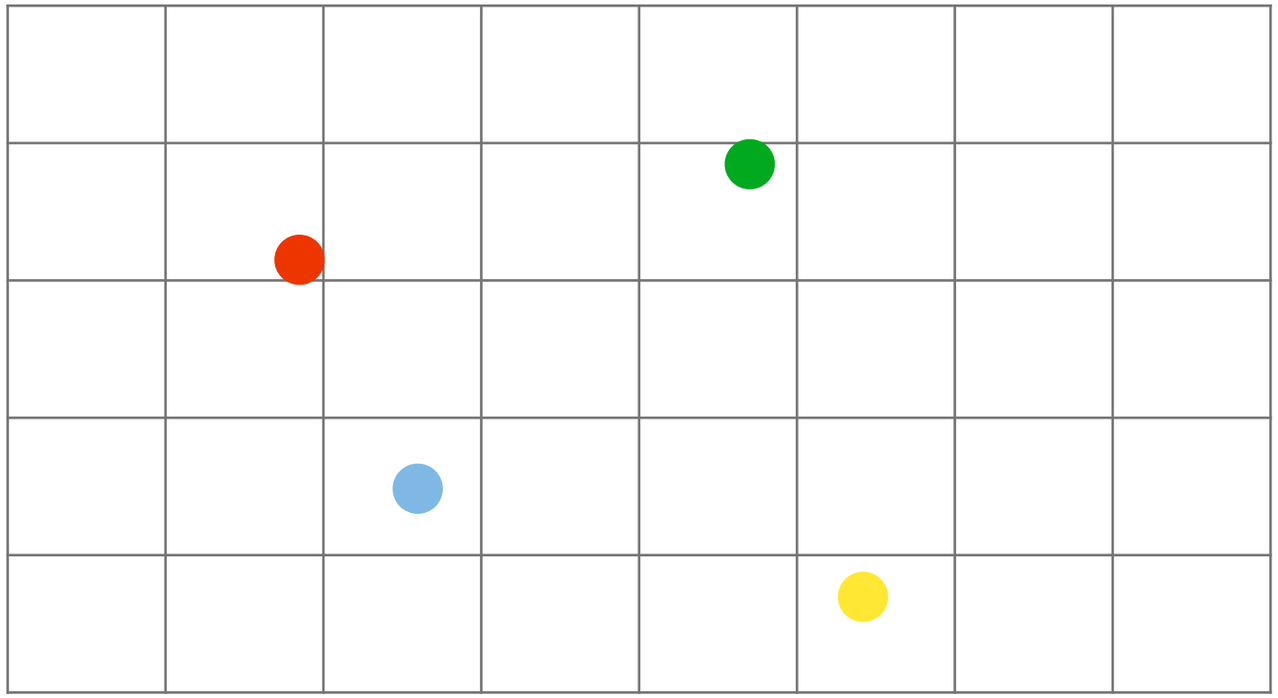
want this

[courtesy of K. Breeden, Stanford University]

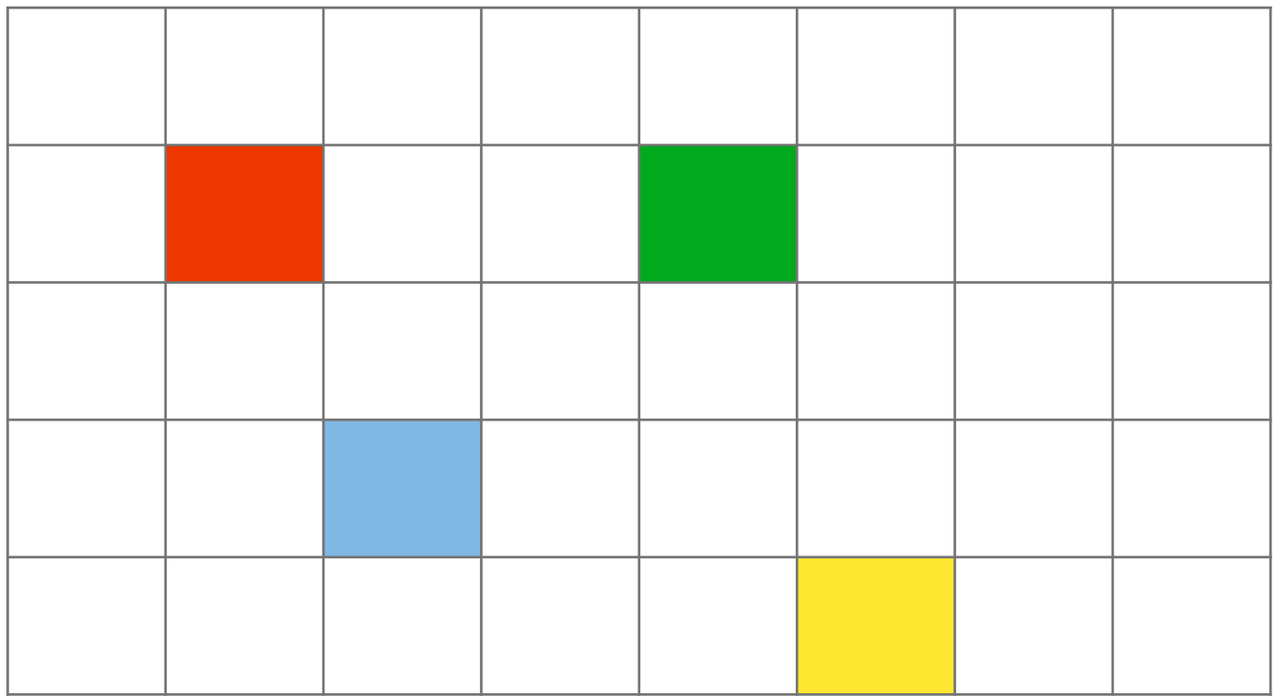
Points



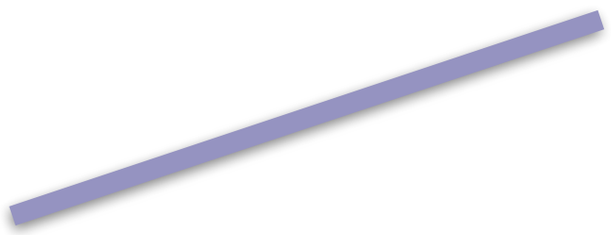
Points

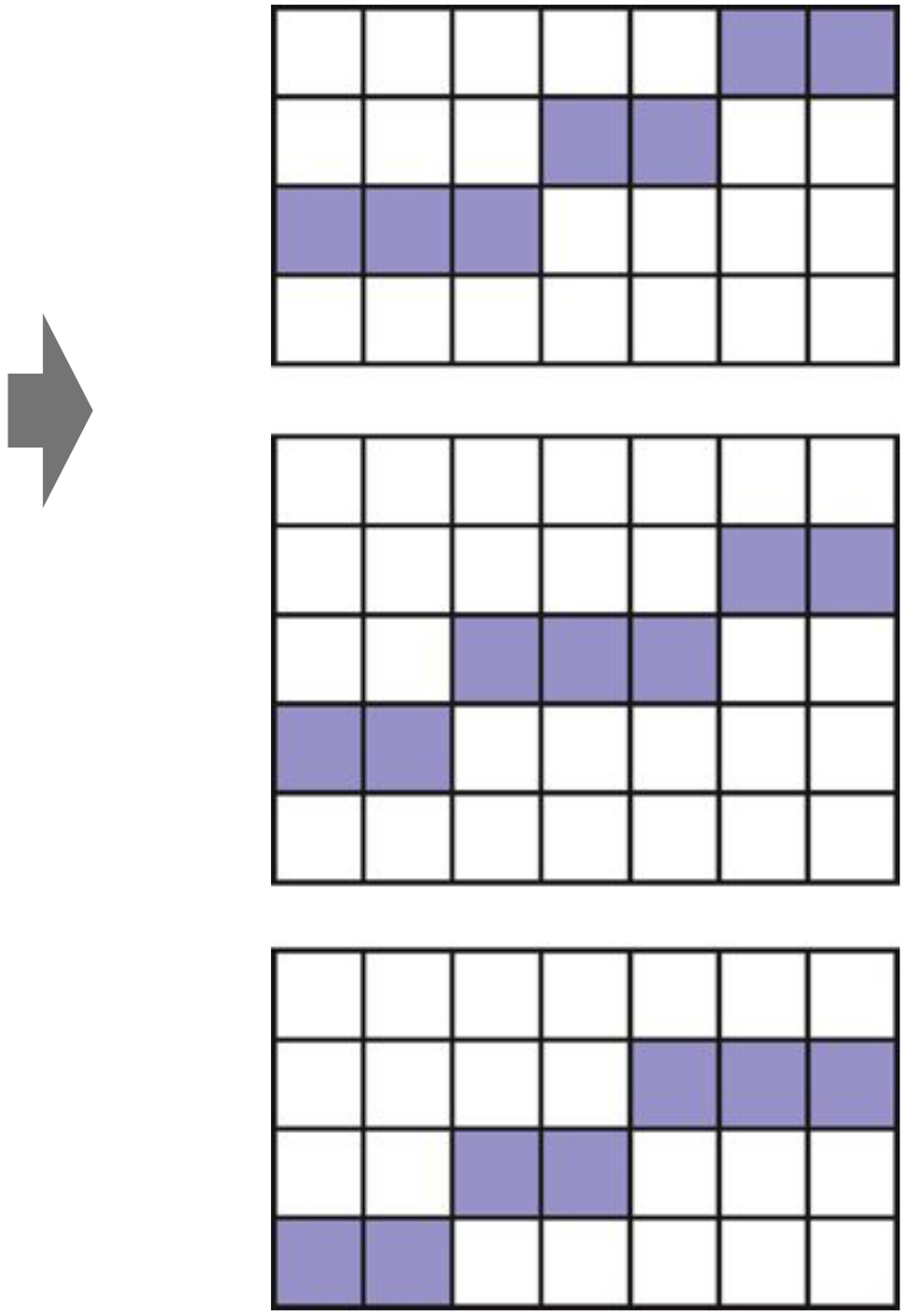


Points



Lines





- Which pixels do we fill in?
- Every square it touches?

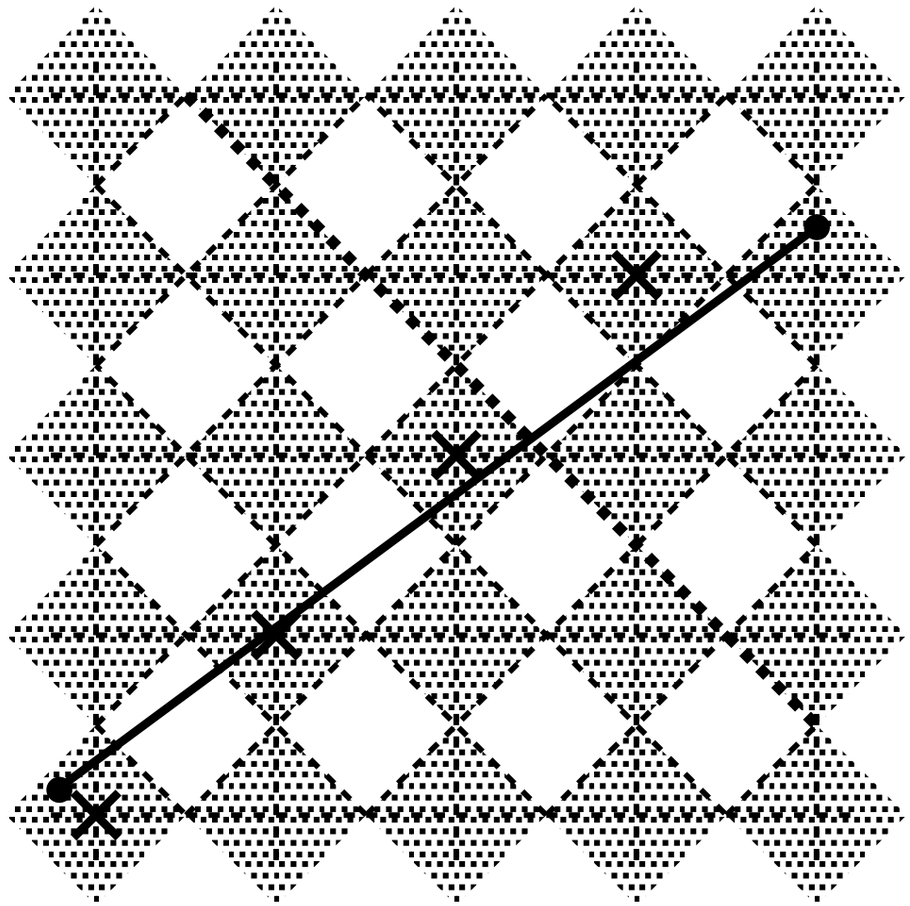
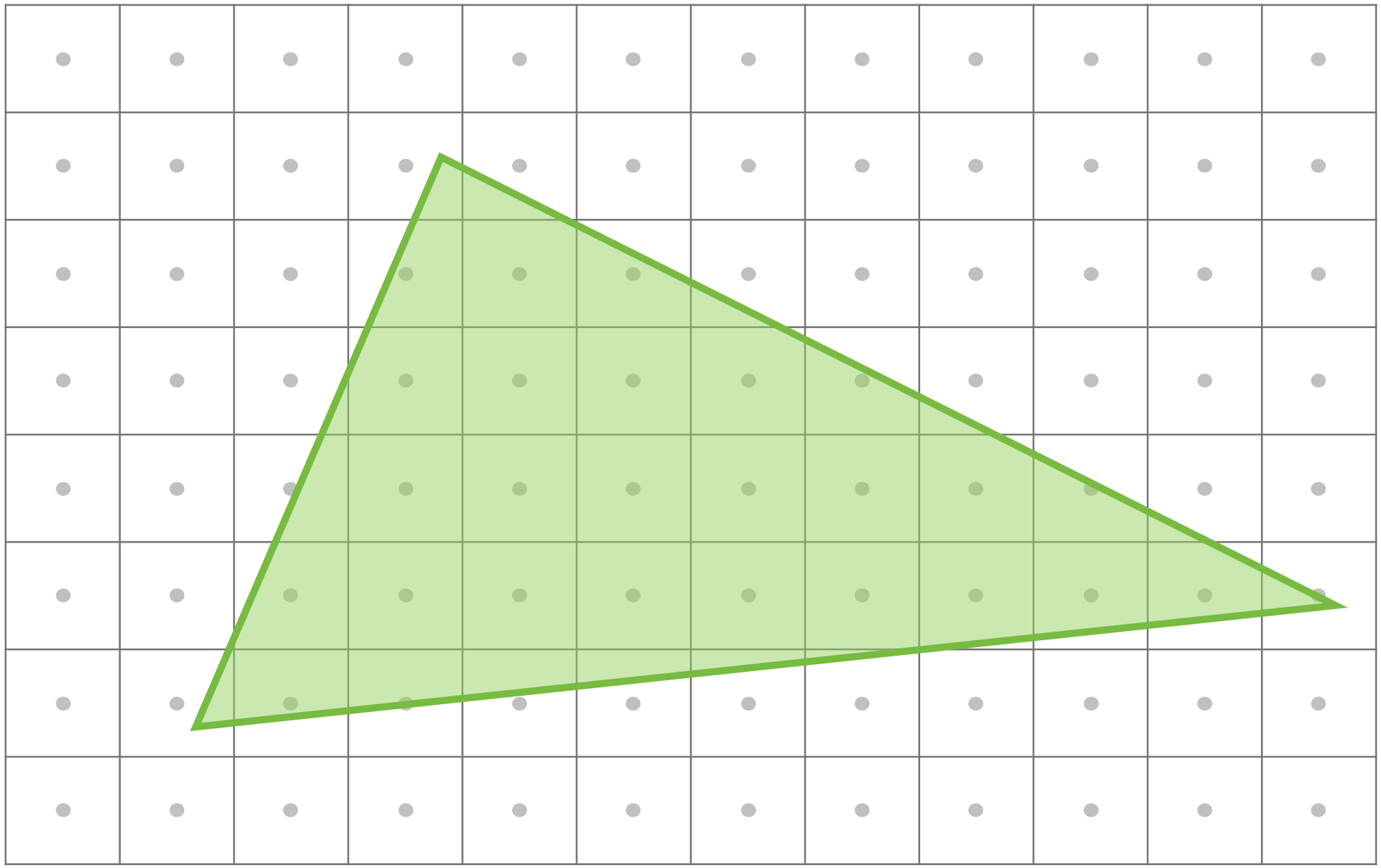


Figure 14.2. Visualization of Bresenham's algorithm. A portion of a line segment is shown. A diamond shaped region of height 1 is placed around each fragment center; those regions that the line segment exits cause rasterization to produce corresponding fragments.

[from OpenGL 4.5 Core Profile specification, p447]

Triangles

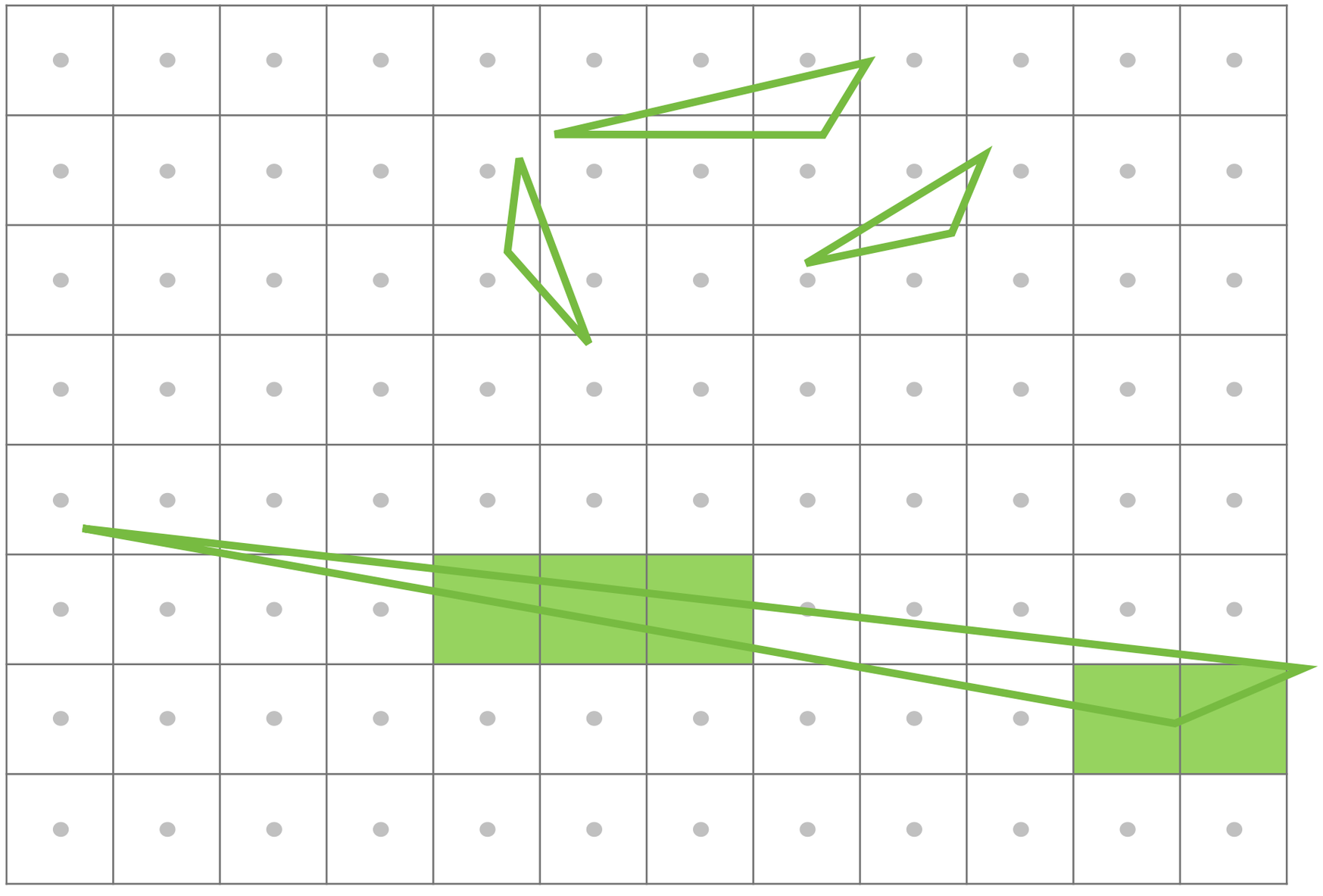


[courtesy of K. Breeden, Stanford University]

Triangles

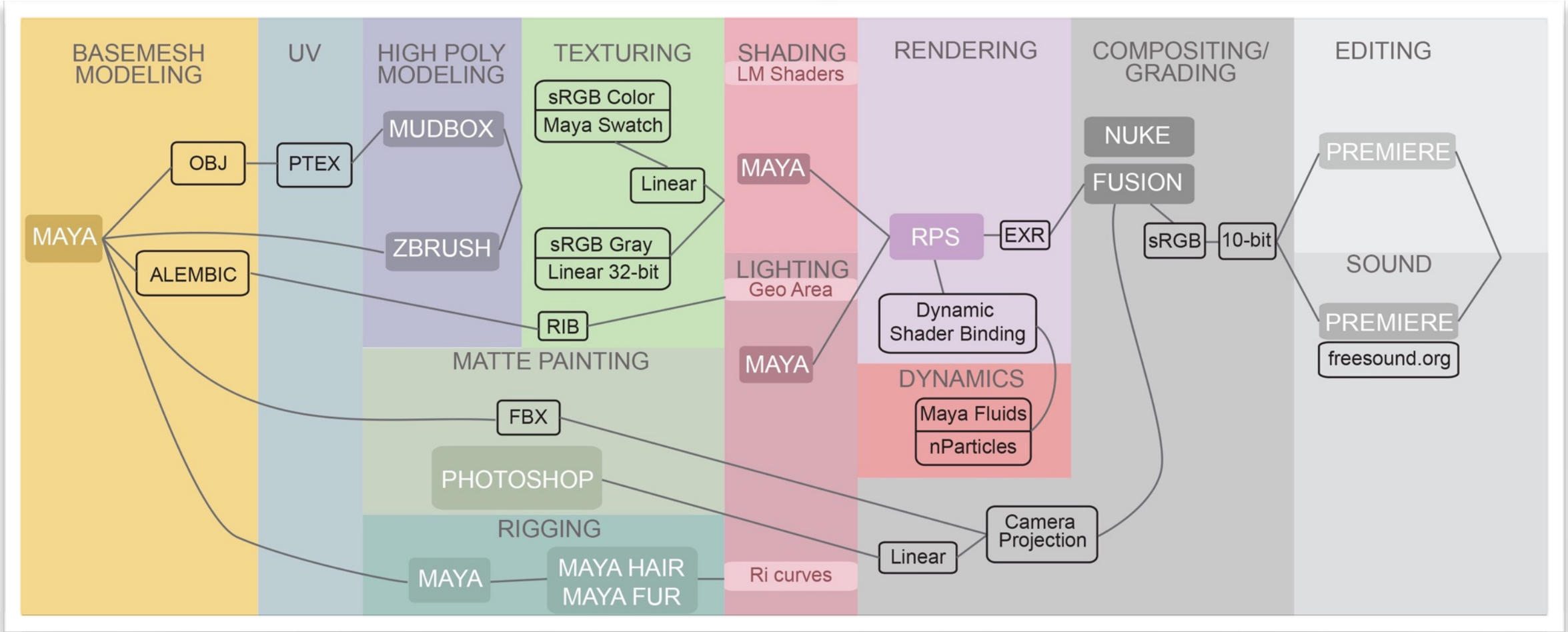
[courtesy of K. Breeden, Stanford University]

Other considerations?



[courtesy of K. Breeden, Stanford University]

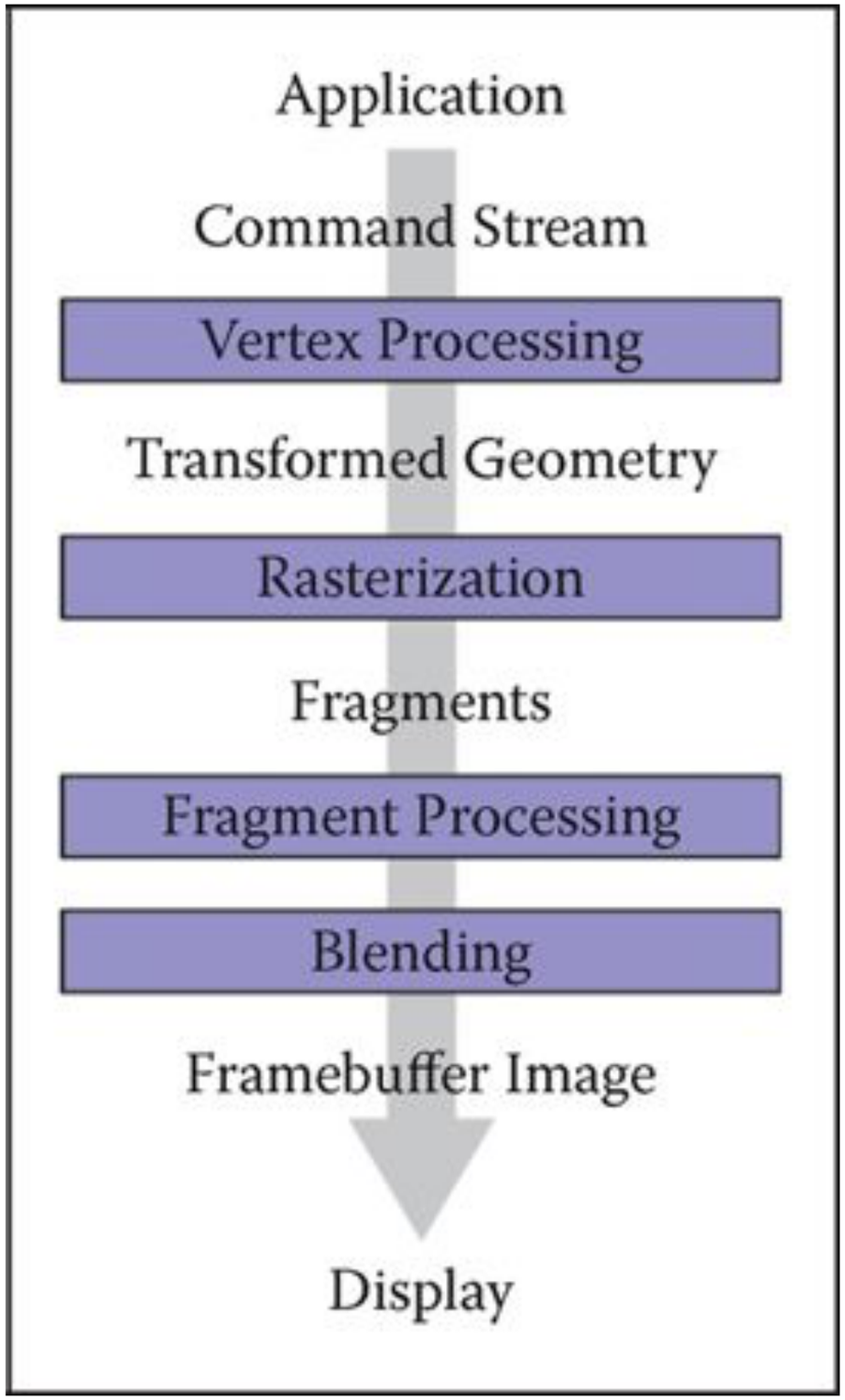
Other graphics pipelines?

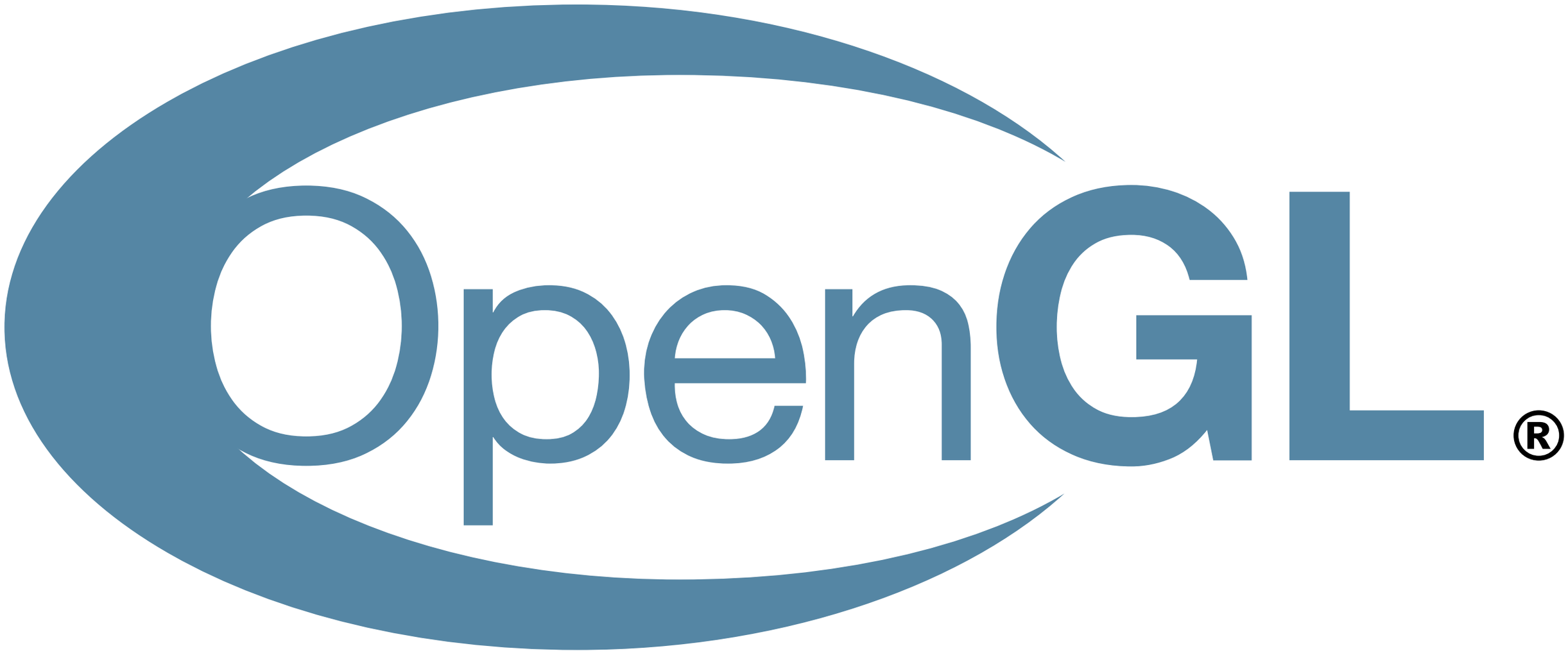


RENDERMAN

[from community.renderman.pixar.com]

What exactly is a pipeline?





What is it…

and what does it do for me?

“OpenGL is an application programming interface to graphics hardware.”

–OpenGL 4.5 Core Profile Specification, §1.2

OpenGL

just gives you a way to control your drawing machine

GeForce GTX 645
576 cores

Other Alternatives?

Microsoft DirectX

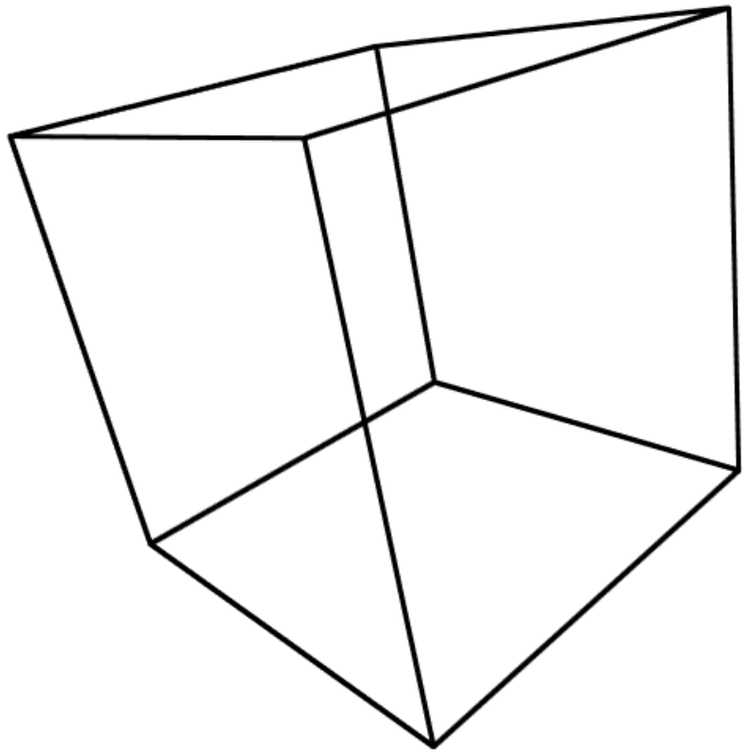
Apple Metal

AMD Mantle

NVIDIA CUDA

Khronos OpenCL

Do we even need OpenGL to do graphics?



$$I(x,x') = g(x,x')\left[\epsilon(x,x') + \int_S \rho(x,x',x'')I(x',x'')dx''\right]$$

$$r'' = \|x' - x''\|$$
$$dx''_p = dx'' \cos\psi''$$
$$\cos\psi' = \frac{1}{r''}\langle \mathbf{n}', x' - x'' \rangle$$
$$\cos\psi'' = \frac{1}{r''}\langle \mathbf{n}'', x' - x'' \rangle$$
$$\cos\sigma' = \frac{1}{r''}\langle \mathbf{t}', x' - x'' \rangle$$
$$d\omega'' = \frac{dx''_p}{r''^2} = \frac{1}{r''^2}\cos\psi'' dx''$$

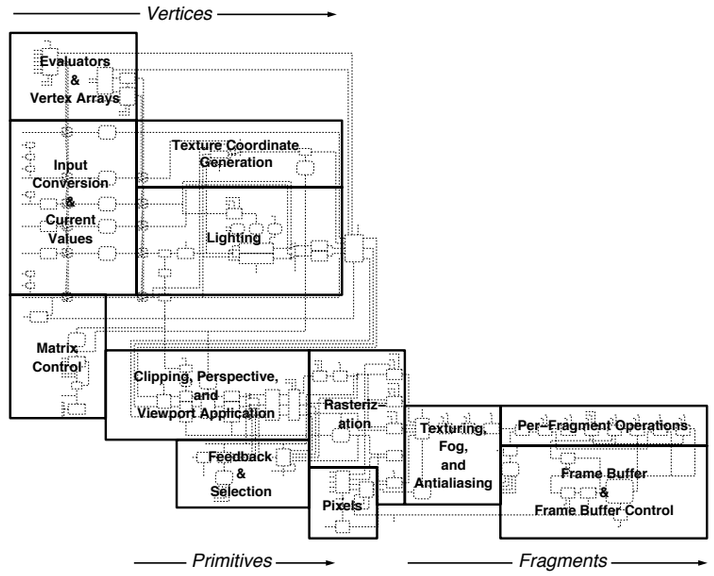
	Data	Algorithms
CPU	arrays/structures in memory	C++ code
OpenGL	buffers, textures	shaders

Open Graphics Library

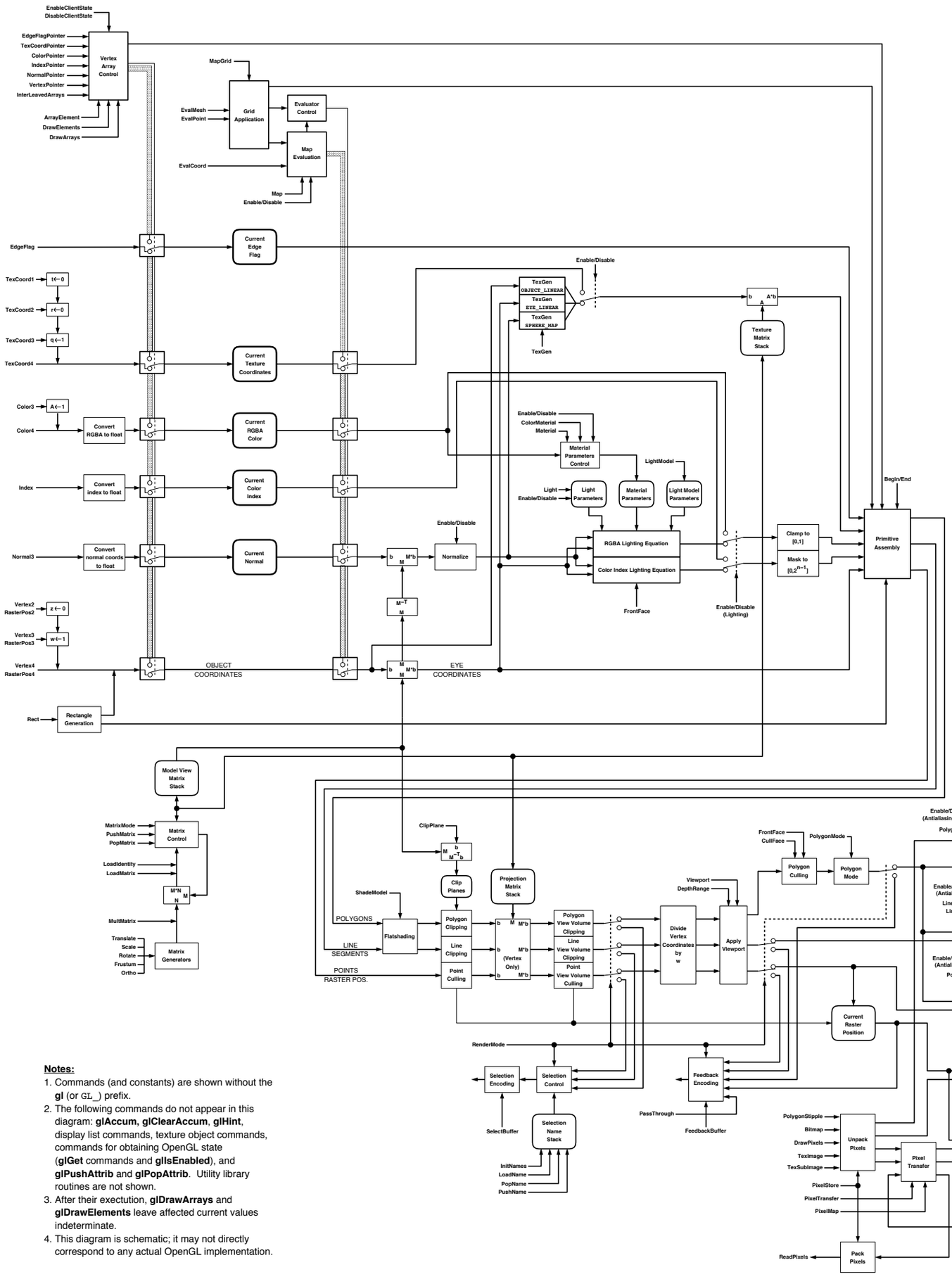
… a misnomer?!

The OpenGL® Machine

[version 1.1]

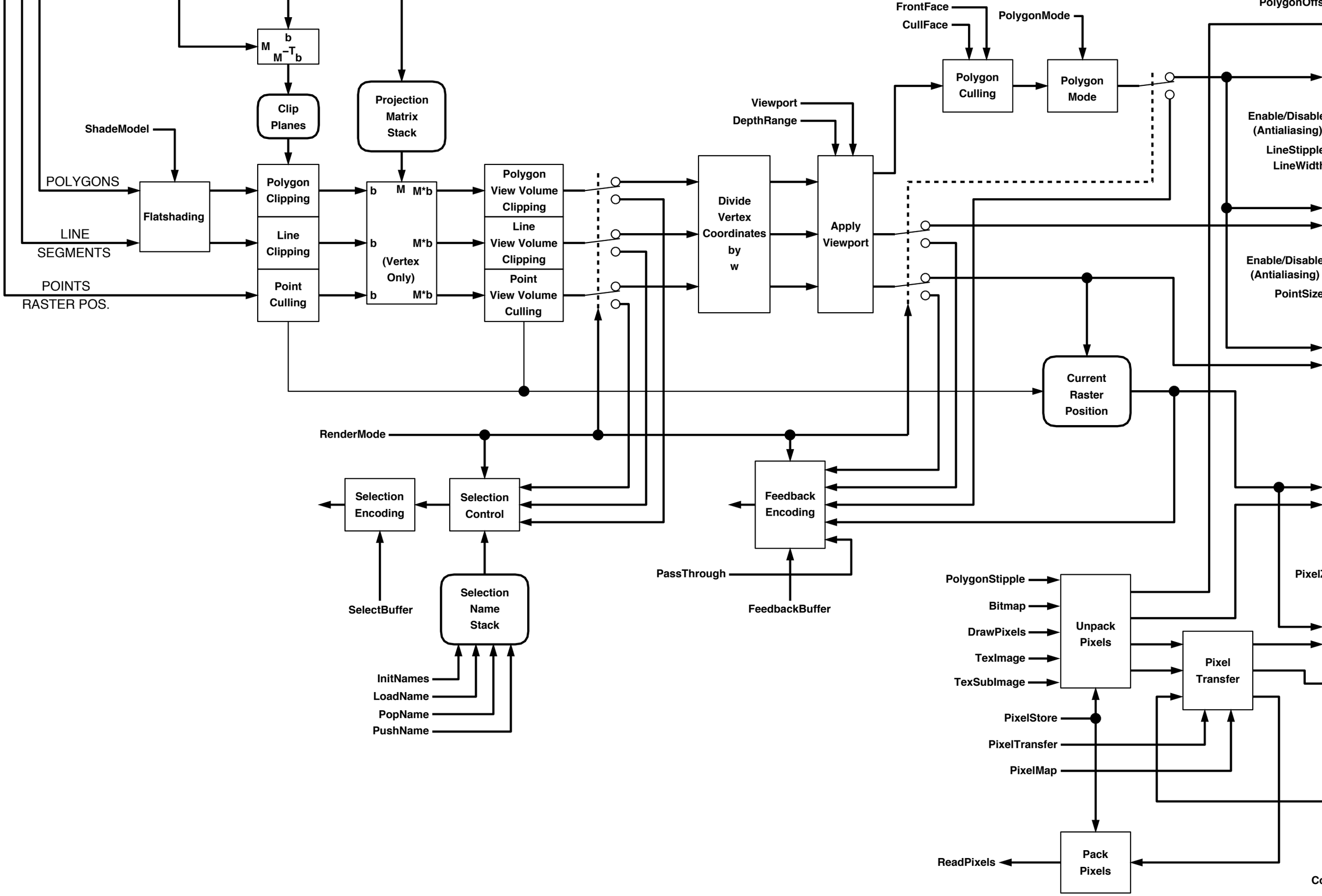


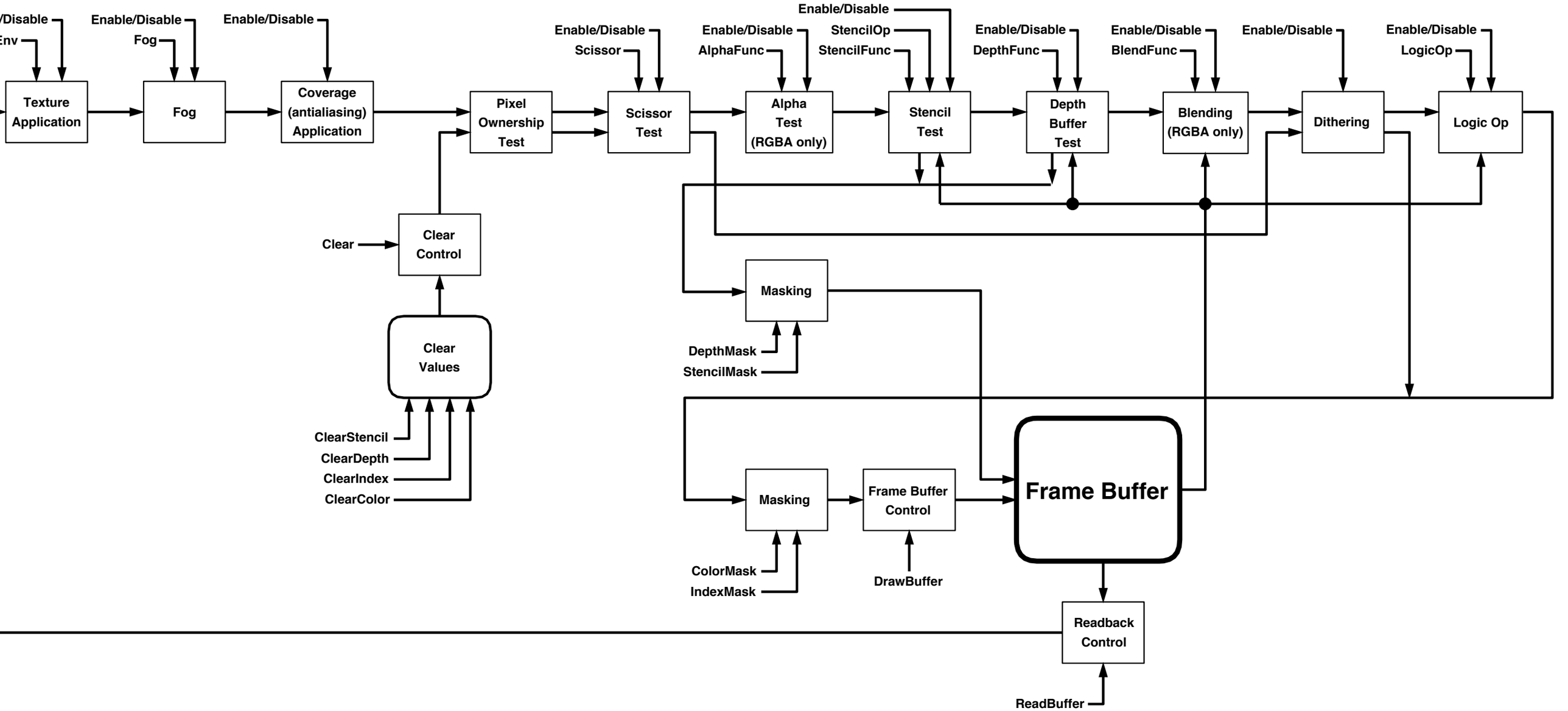
Key to OpenGL Operations



Notes:

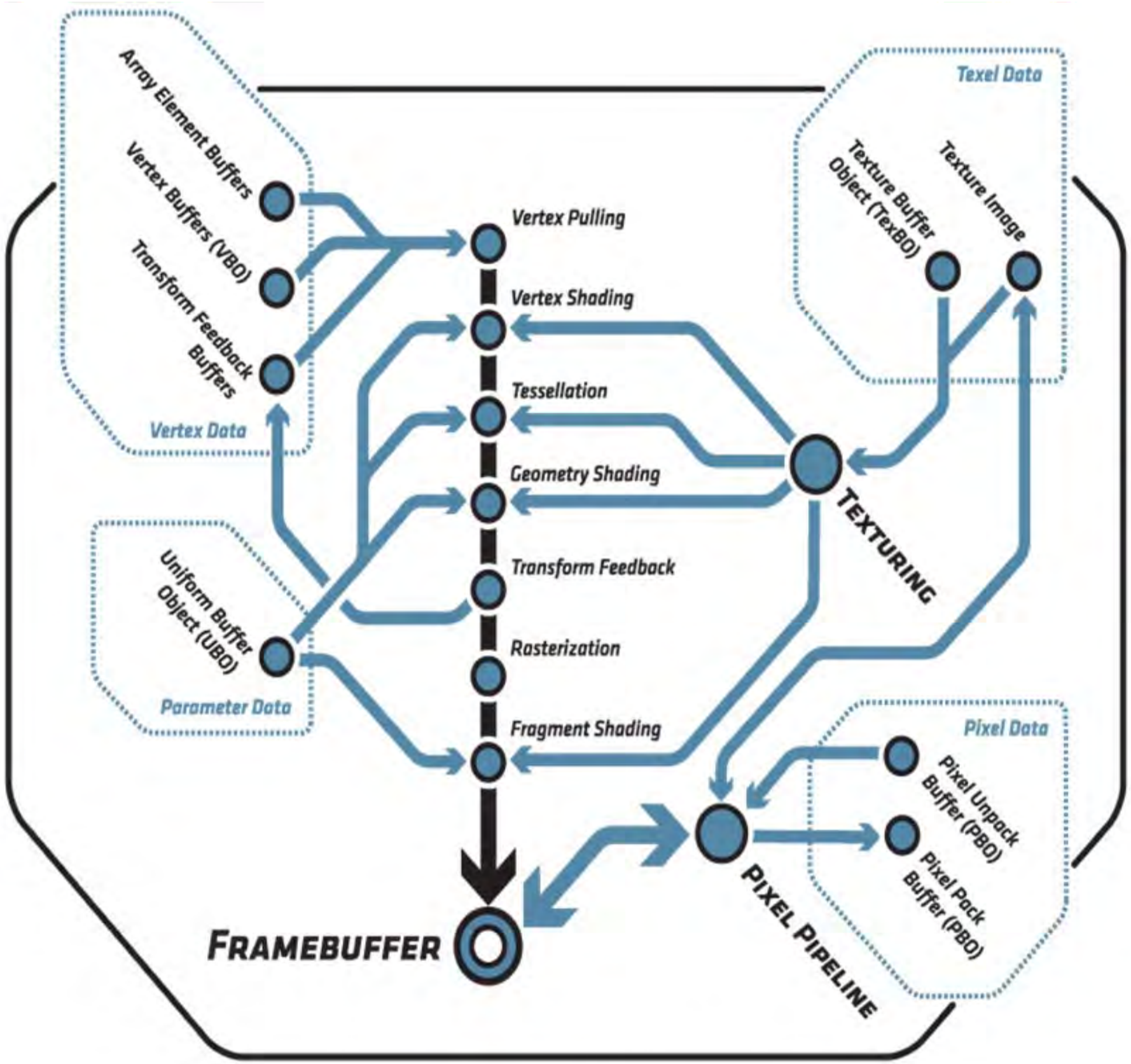
1. Commands (and constants) are shown without the **gl** (or GL_) prefix.
2. The following commands do not appear in this diagram: **glAccum, glClearAccum**, **glHint**, display list commands, texture object commands, commands for obtaining OpenGL state (**glGet** commands and **glIsEnabled**), and **glPushAttrib** and **glPopAttrib**. Utility library routines are not shown.
3. After their exectution, **glDrawArrays** and **glDrawElements** leave affected current values indeterminate.
4. This diagram is schematic; it may not directly correspond to any actual OpenGL implementation.


M
b
M
M
–T
b
Clip
Planes
Projection
Matrix
Stack
ShadeModel
POLYGONS
LINE
SEGMENTS
POINTS
RASTER POS.
Flatshading
Polygon
Clipping
Line
Clipping
Point
Culling
b
M
M*b
b
M*b
(Vertex
Only)
b
M*b
Polygon
View Volume
Clipping
Line
View Volume
Clipping
Point
View Volume
Culling
Divide
Vertex
Coordinates
by
w
Viewport
DepthRange
Apply
Viewport
FrontFace
CullFace
PolygonMode
Polygon
Culling
Polygon
Mode
Enable/Disable
(Antialiasing)
LineStipple
LineWidth
Enable/Disable
(Antialiasing)
PointSize
Current
Raster
Position
RenderMode
Selection
Encoding
Selection
Control
SelectBuffer
Selection
Name
Stack
InitNames
LoadName
PopName
PushName
Feedback
Encoding
PassThrough
FeedbackBuffer
PolygonStipple
Bitmap
DrawPixels
TexImage
TexSubImage
Unpack
Pixels
Pixel
Transfer
PixelStore
PixelTransfer
PixelMap
Pack
Pixels
ReadPixels


/Disable
Env
Enable/Disable
Fog
Enable/Disable
Texture Application
Fog
Coverage (antialiasing) Application
Pixel Ownership Test
Enable/Disable
Scissor
Scissor Test
Enable/Disable
AlphaFunc
Alpha Test (RGBA only)
Enable/Disable
StencilOp
StencilFunc
Stencil Test
Enable/Disable
DepthFunc
Depth Buffer Test
Enable/Disable
BlendFunc
Blending (RGBA only)
Enable/Disable
Dithering
Enable/Disable
LogicOp
Logic Op
Clear
Clear Control
Clear Values
ClearStencil
ClearDepth
ClearIndex
ClearColor
Masking
DepthMask
StencilMask
Masking
Frame Buffer Control
Frame Buffer
ColorMask
IndexMask
DrawBuffer
Readback Control
ReadBuffer

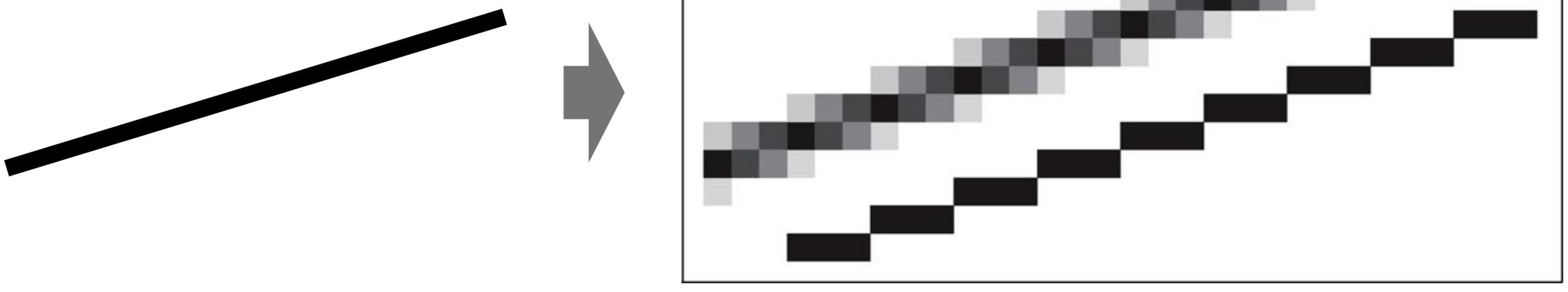
The OpenGL 2.1 Machine

OpenGL 3+


Array Element Buffers
Vertex Buffers (VBO)
Transform Feedback Buffers
Vertex Data
Vertex Pulling
Vertex Shading
Tessellation
Geometry Shading
Transform Feedback
Rasterization
Fragment Shading
Texel Data
Texture Buffer Object (TexBO)
Texture Image
TEXTURING
Uniform Buffer Object (UBO)
Parameter Data
Pixel Data
Pixel Unpack Buffer (PBO)
Pixel Pack Buffer (PBO)
PIXEL PIPELINE
FRAMEBUFFER

OpenGL isn’t completely unhelpful…

- Scan line conversion
- Image sampling and interpolation
- And, of course, it’s pretty darned fast!!!

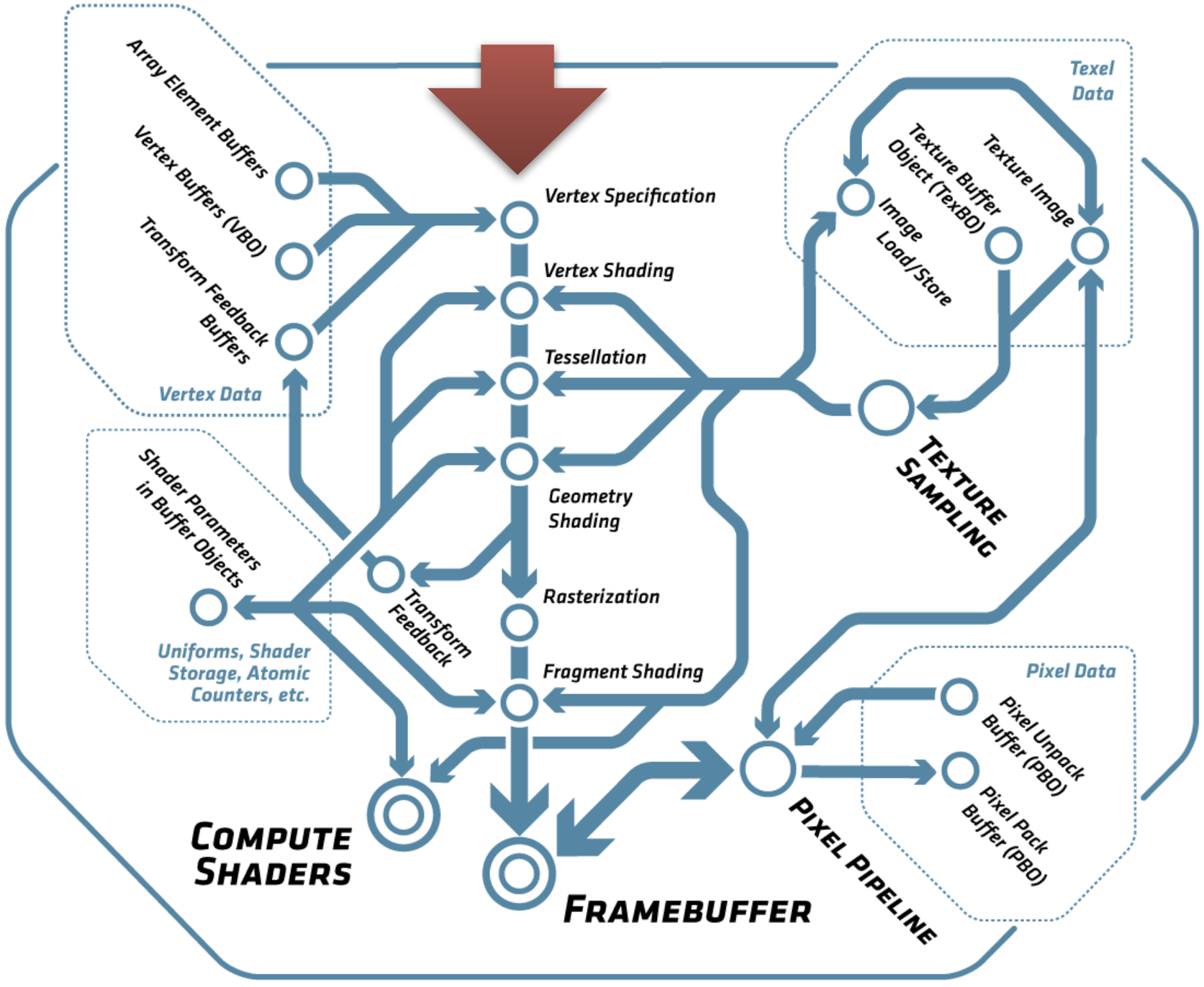


OpenGL in this class

- We will be using OpenGL 3.2+ Core Profile
  - which means you'll have have to work harder
  - but it's perfect for learning the theory of computer graphics!

- Be careful which references you use
  - OpenGL 4.5 pages: https://www.opengl.org/sdk/docs/man/
  - https://www.opengl.org/registry/doc/glspec45.core.pdf
  - https://www.opengl.org/registry/doc/GLSLangSpec.4.50.pdf

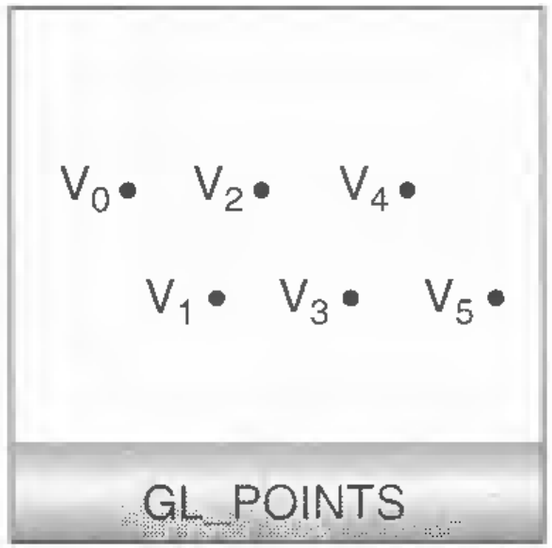
The OpenGL 4 Rendering Pipeline

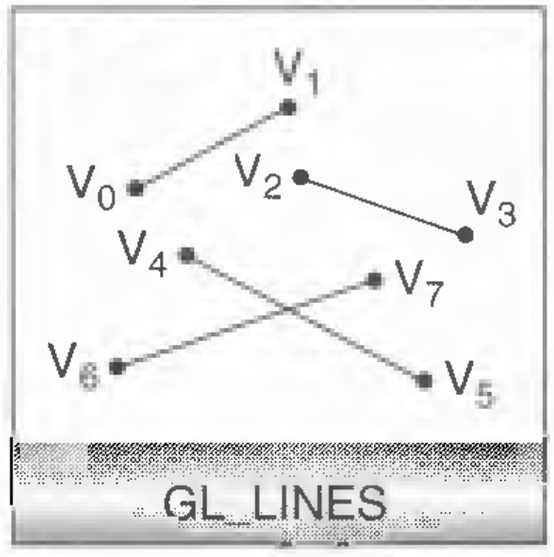


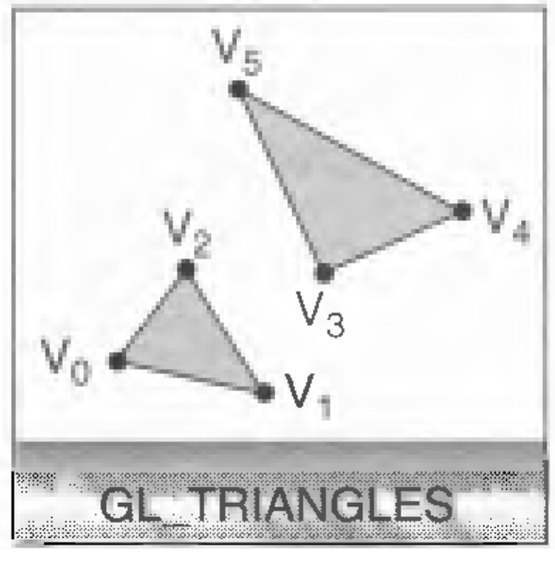
[from OpenGL 4.5 Core Profile specification]

Primitives

- Points
- Lines
- Triangles

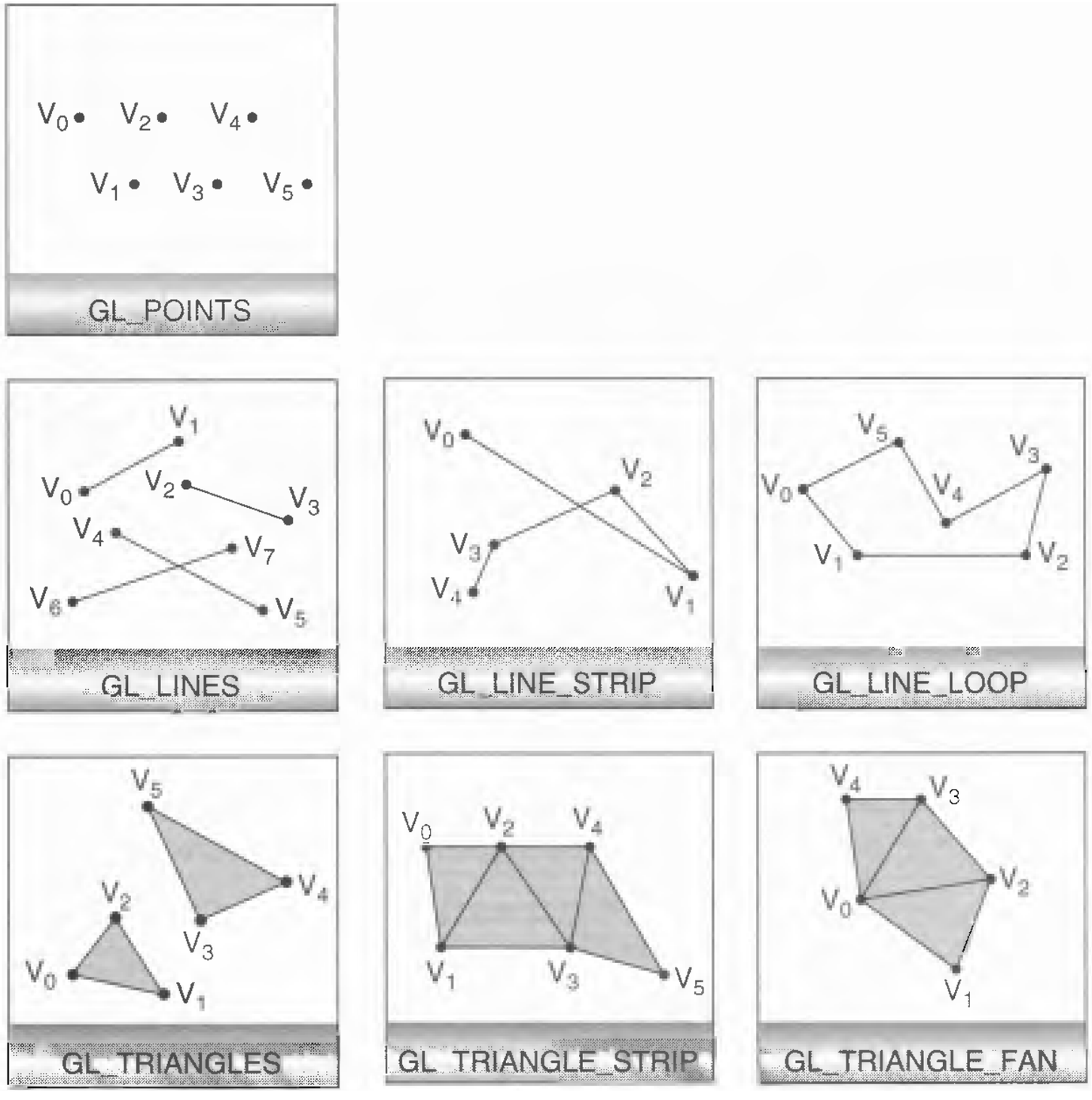






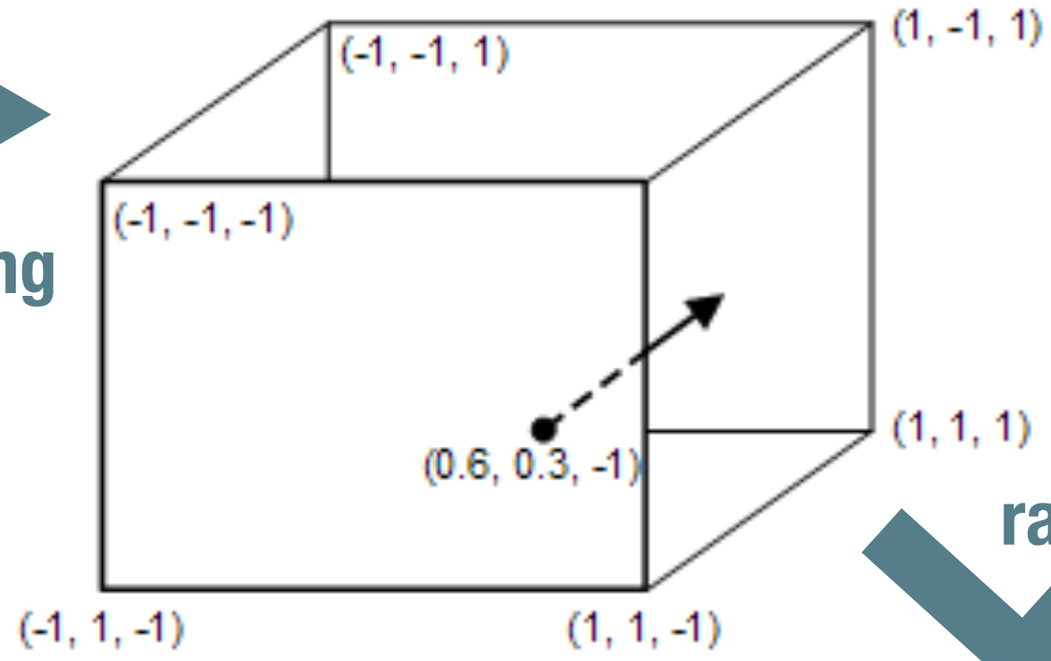
Primitives

- Points
- Lines
- Triangles
- … and some derivatives



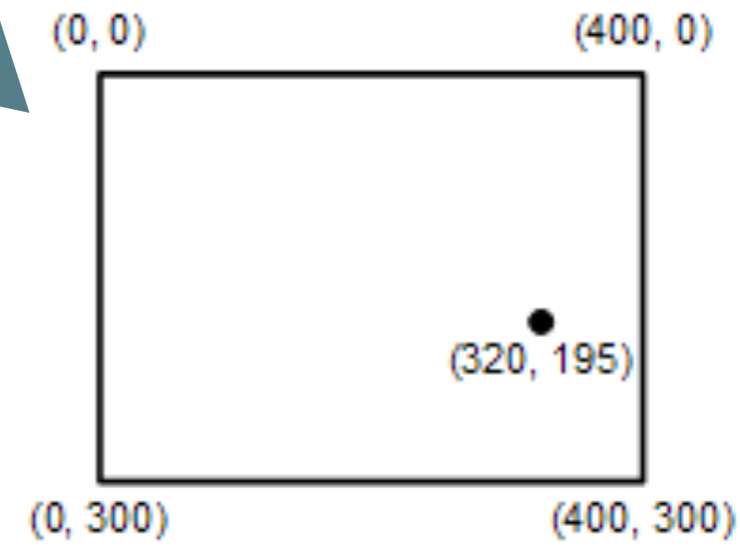
Coordinate Frames

vertex processing



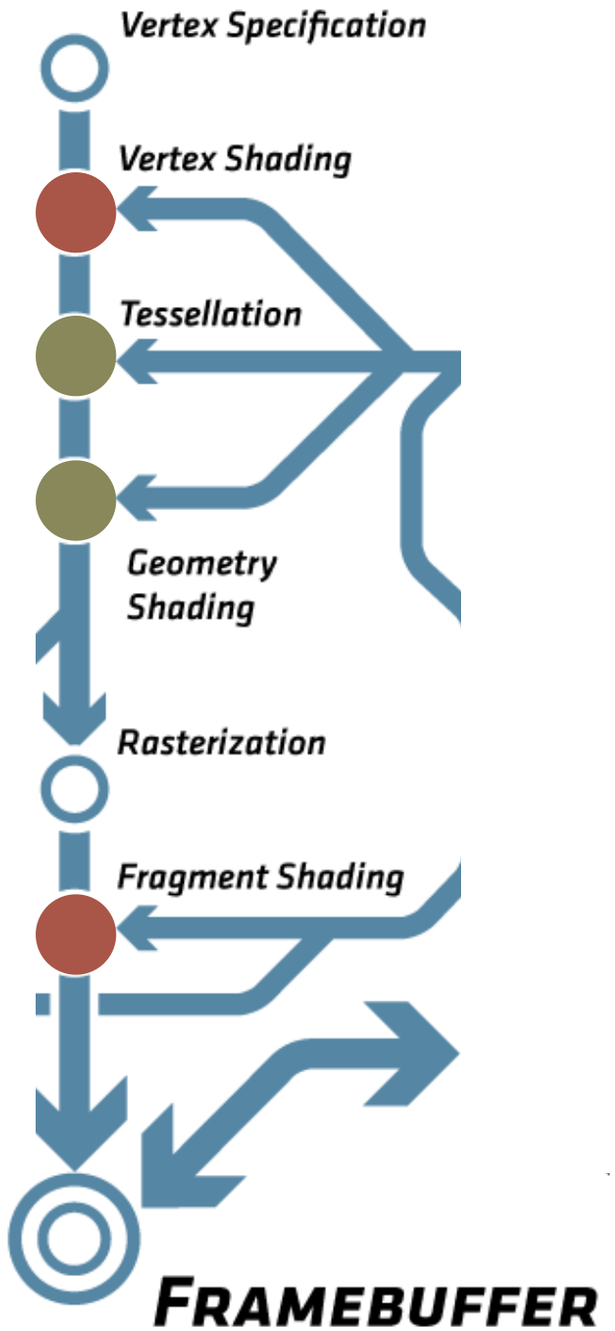
normalized device coordinates

rasterization



window coordinates

Working with Shaders



- Almost every stage is programmable
  - **Vertex**
  - **Tessellation**
  - **Geometry**
  - **Fragment**
- Specify inputs and outputs at each stage, then write whatever code you want!
  - OpenGL takes care of compiling and running the code on your GPU

Things to Remember

- A **graphics system** is just a **drawing machine**
  - though they've evolved a lot over the years
- The **graphics pipeline** transforms **data** into **images**
  - many stages where different algorithms are executed
- **OpenGL** is an API that lets you **run code on your GPU**
  - and can help with rasterization and interpolation