Modern OpenGL with Python

Roberto De Ioris

@20tab
@boiagames
@unbit

http://www.aiv01.it/

Europython 2016
Bilbao 2016

Why OpenGL ?

… or DirectX …

… or Metal …

… or Vulkan …

APIs for GFX cards

what does it mean ?

OpenGL®

an industry standard drafted in 1991/1992
by Silicon Graphics (now SGI)

now controlled by Khronos Group

currently at version 4.5

lot of documentation

portability

DISCLAIMER

heavily c-oriented api with performance and compatibility as the only objectives

Drawing with your computer (the optimal way)

ask your OS for a drawable context (a window, or the whole screen)

agree on a pixel format (RGB, RGBA, B&W…)

allocate a memory representation of your canvas based on pixel format (like 640*480*3 array of bytes)

write pixel data (respecting the format) into the allocated memory

transfer the whole allocated memory content to the gfx card

The optimal way sucks

slow as hell

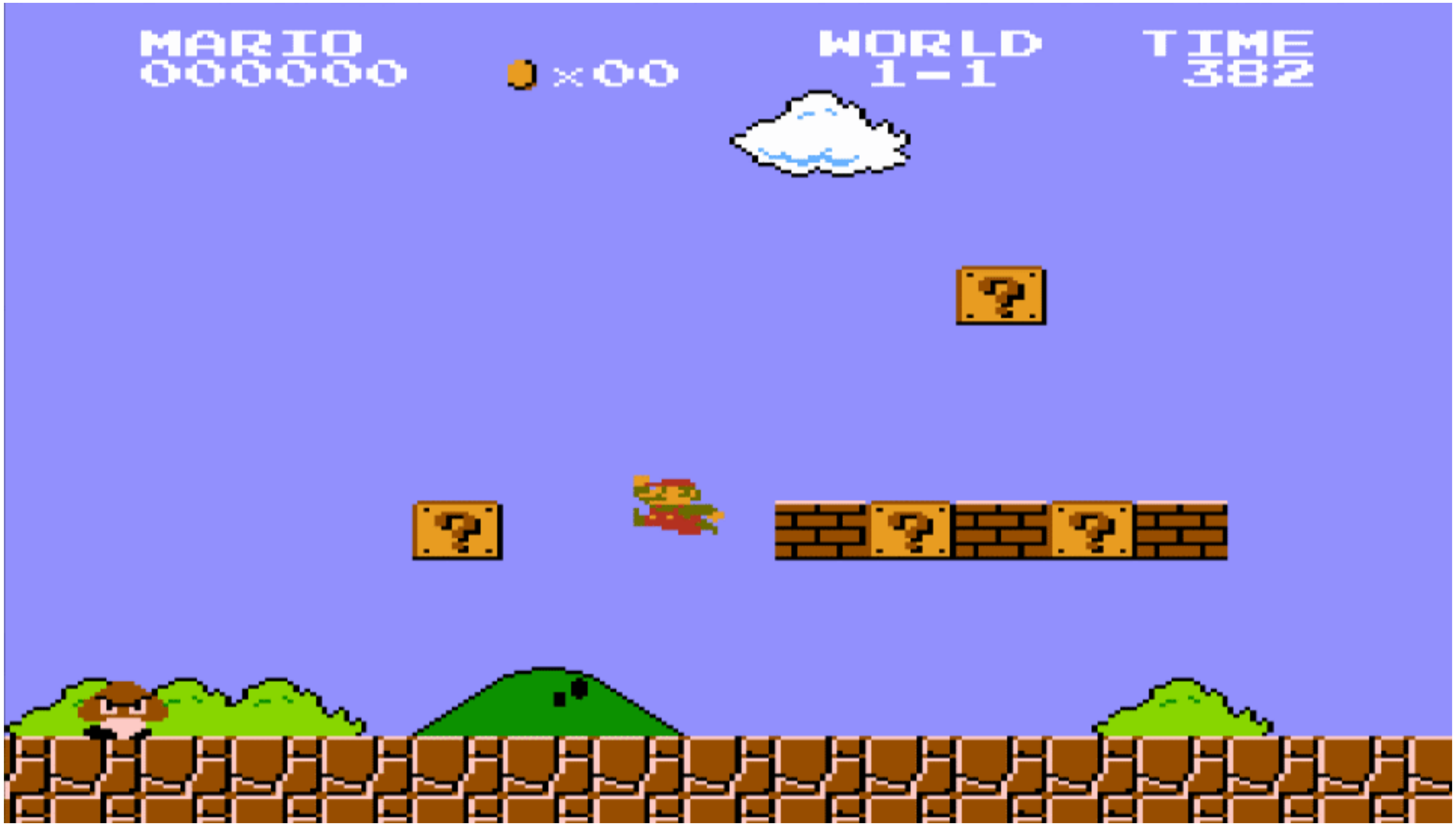
lot of memory
(think about 1920*1080*RGBA[4])

high load on the hardware bus
(multiply it for 60 fps !)

back to 30 years ago

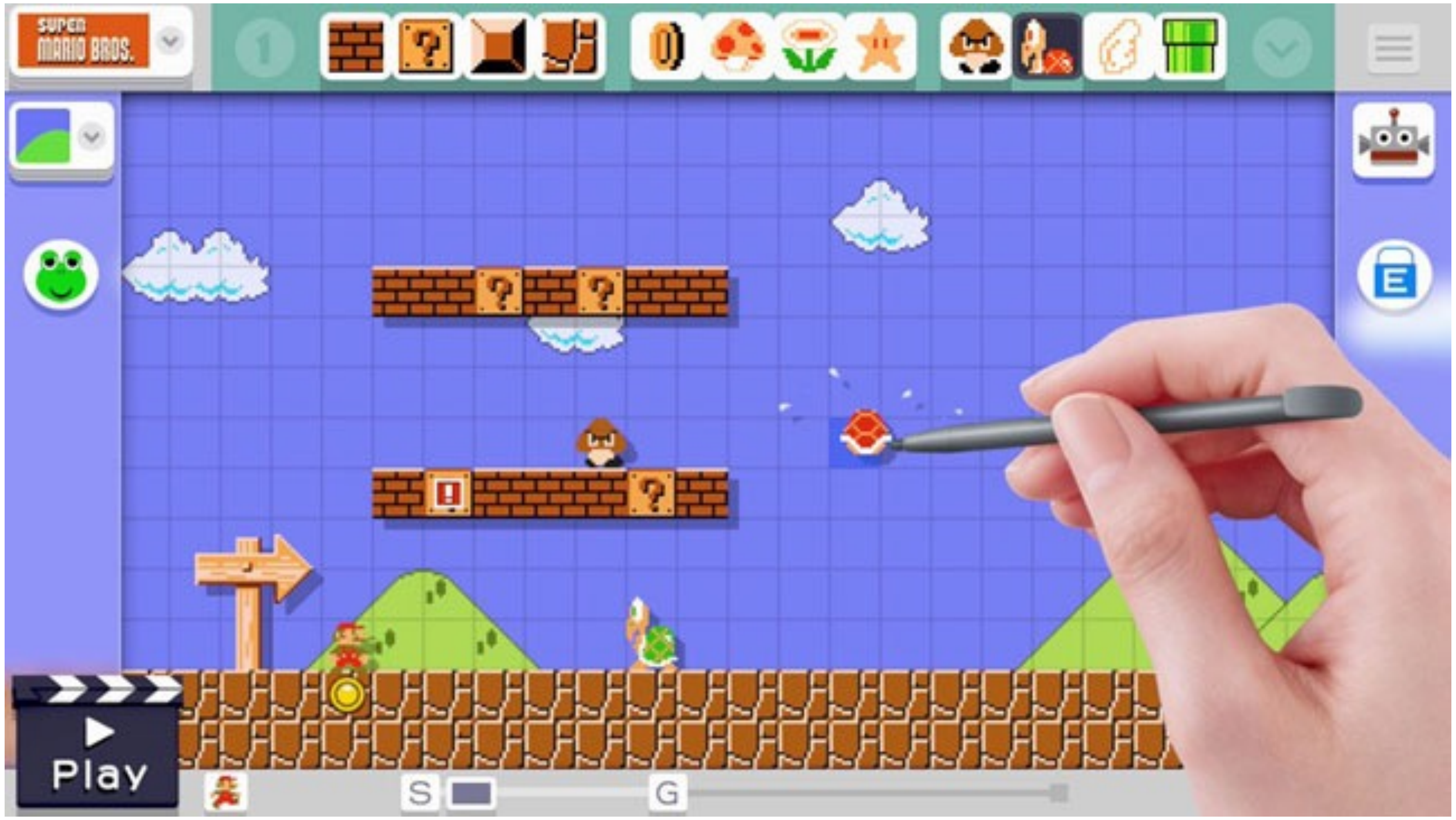
tilemaps
dedicated coprocessors
(hardware sprites)
limit colors and resolution
hack all over the place
forget about realtime 3d

Super Mario Bros (1985, NES)



… like this

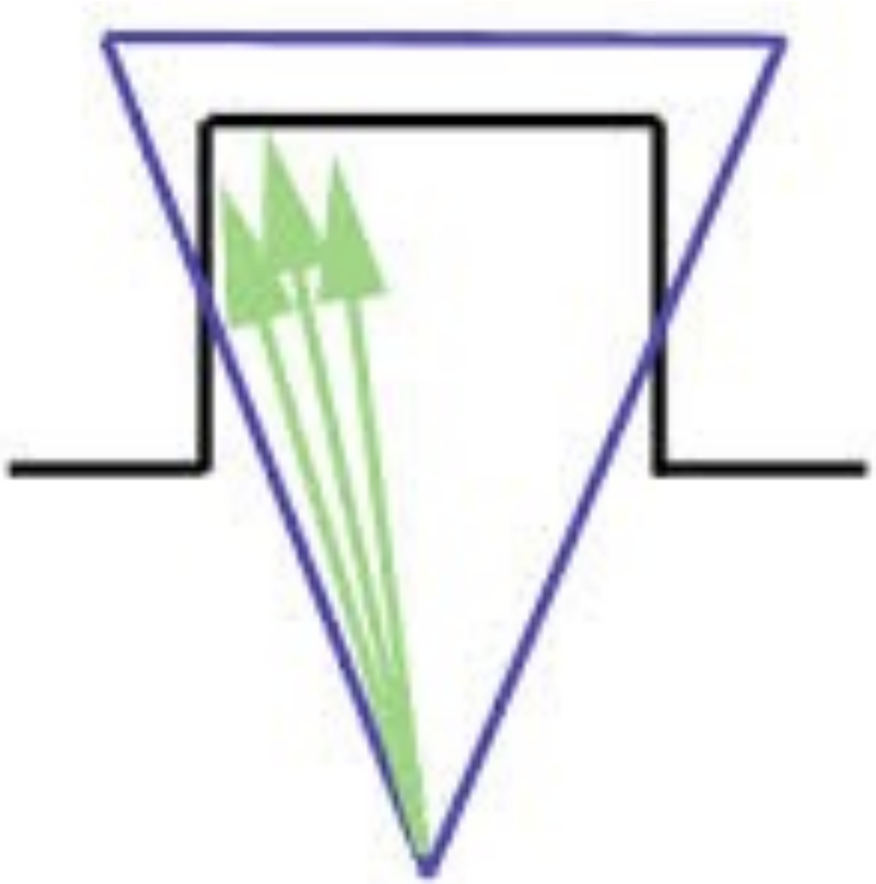
Super Mario Maker (Nintendo WiiU)

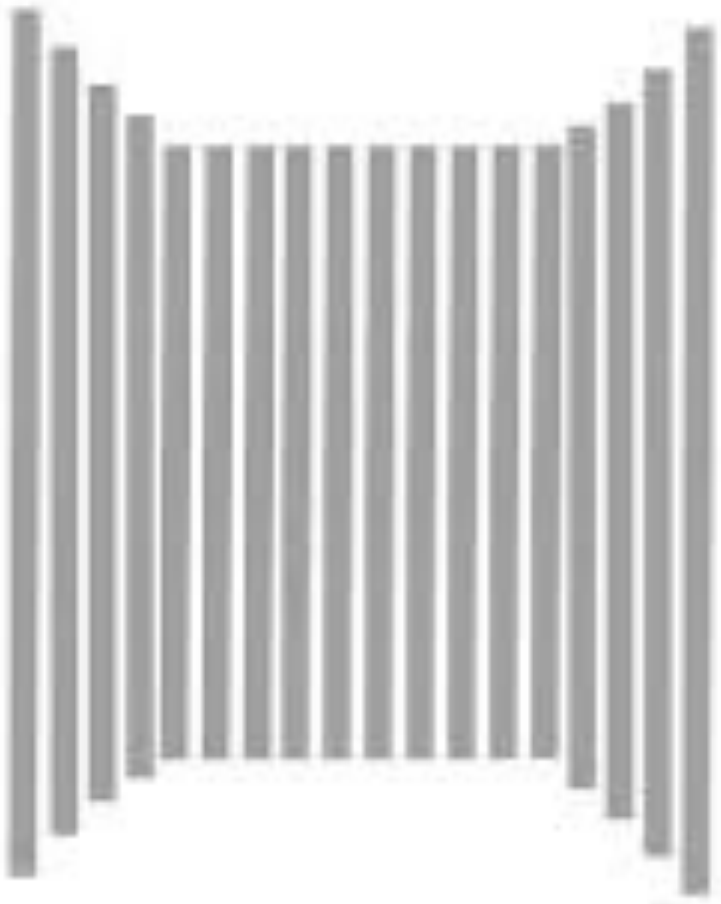


start of the 90’s

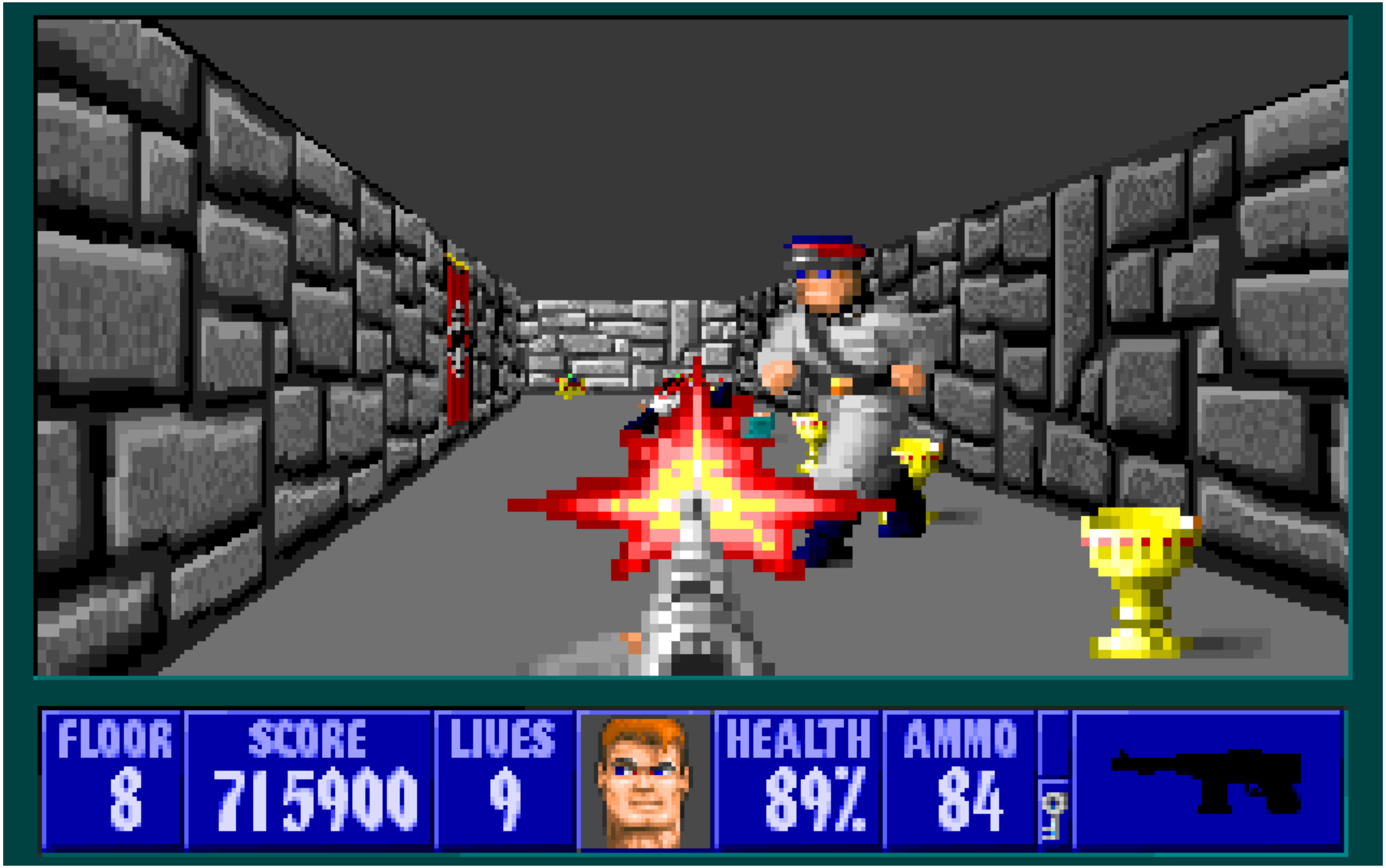
the raycasting revolution

Raycasting for fake 3d





Wolfenstein 3D (Id software,1991/1992)

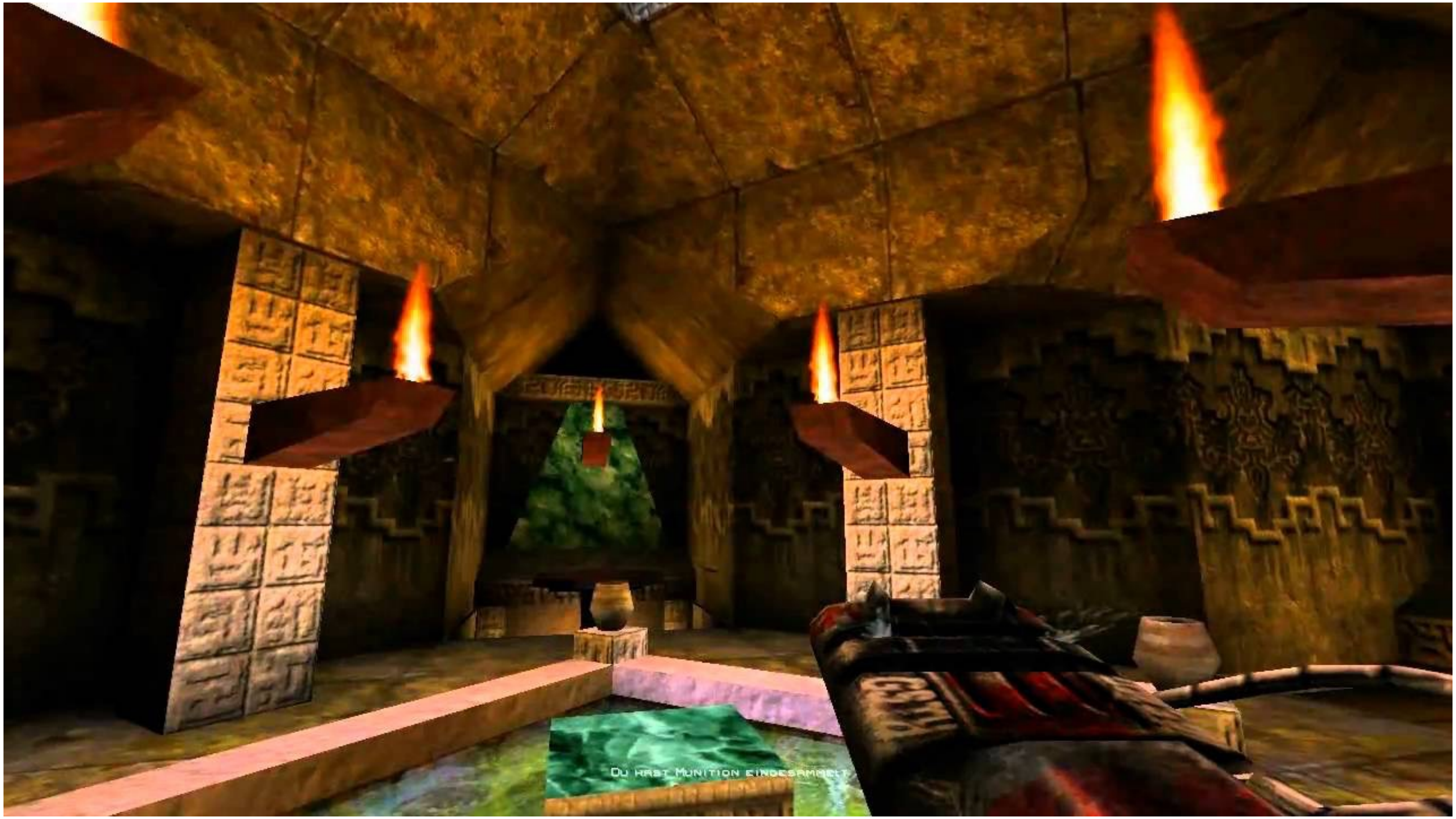


back to 20 years ago

3DFX and Voodoo
Glide
MiniGL

yes realtime 3D !!!

Unreal (Epic games, 1998)



Du hast Munition eingesammelt.

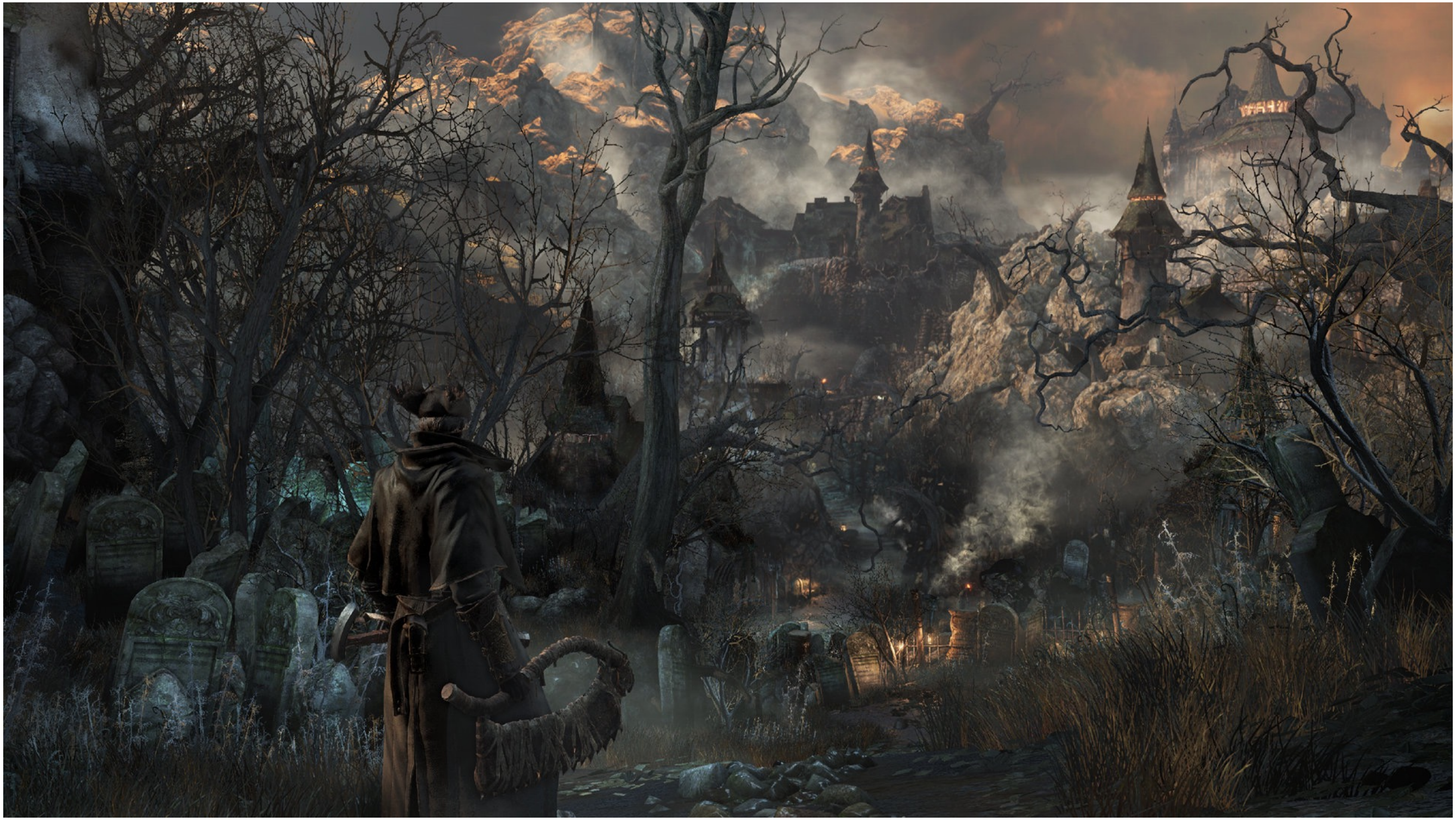
Today

more hardware power

NVidia and AMD

programmable gfx cards

Bloodborne (From Software, 2015 PS4)



What is 3D graphics ?

define shapes by polygons (mainly triangles)

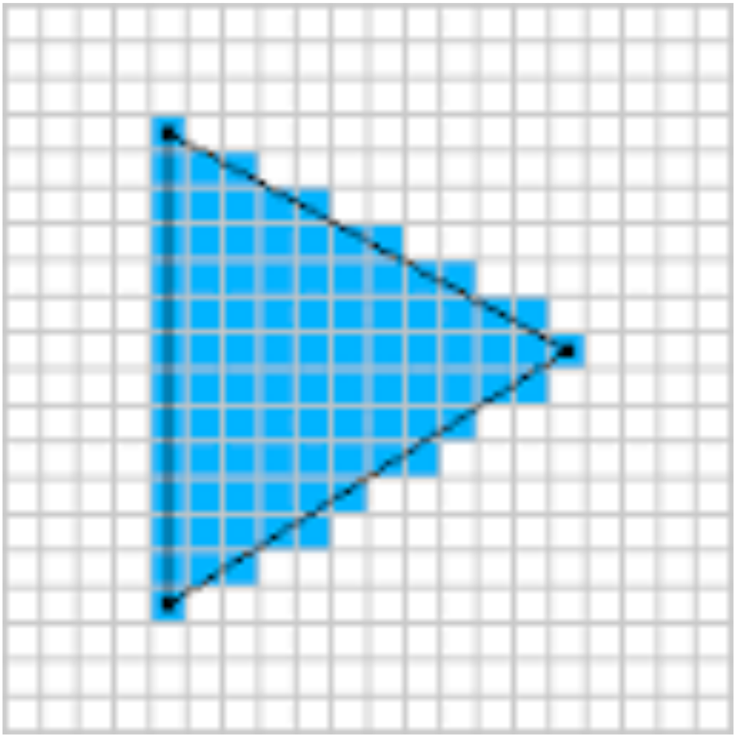
define polygons by vertices

fake your eyes (well, your brain ...) projecting polygons

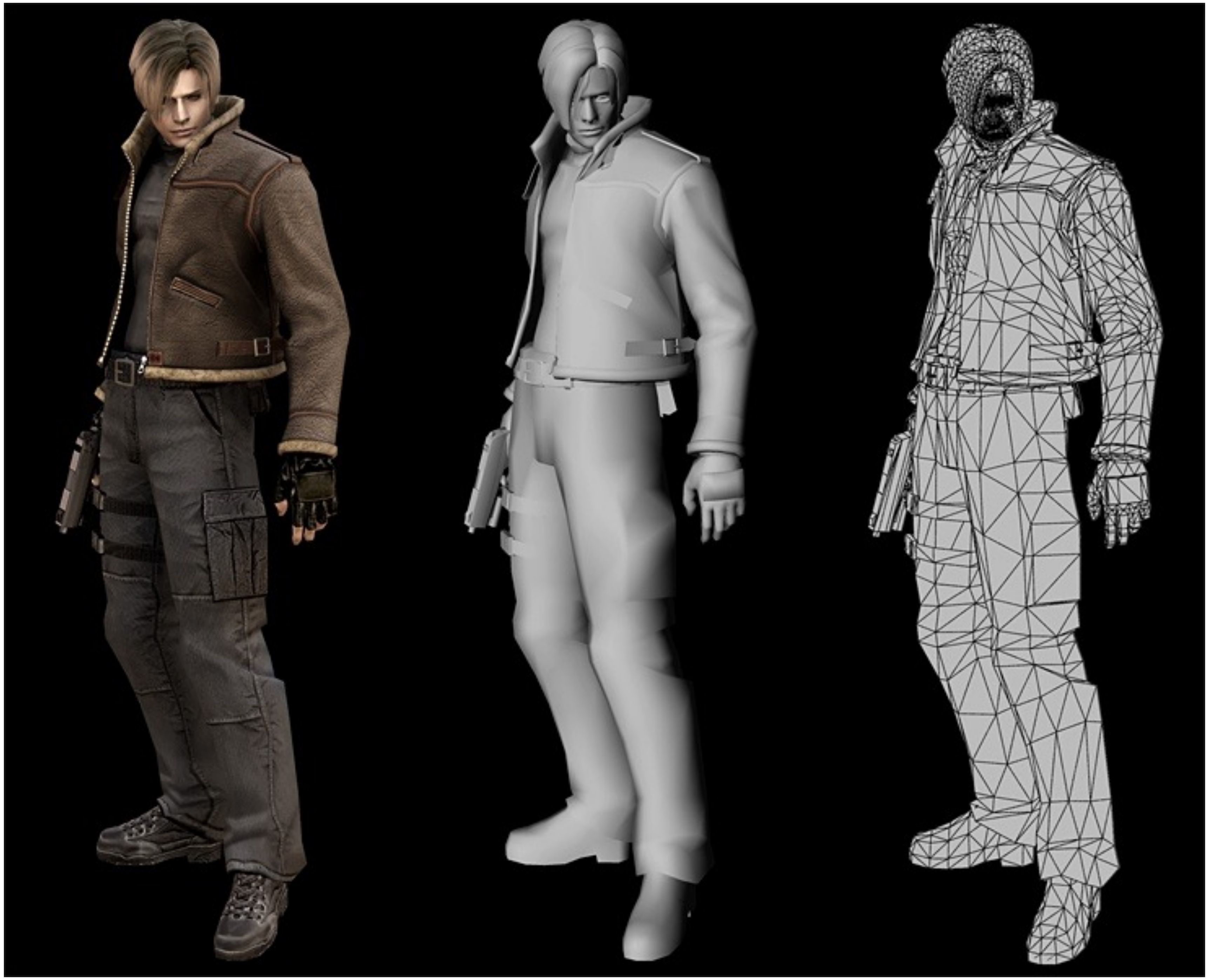
rasterize polygons accounting lighting and textures

Rasterization ?

fill 2d shapes line by line starting from top to the bottom

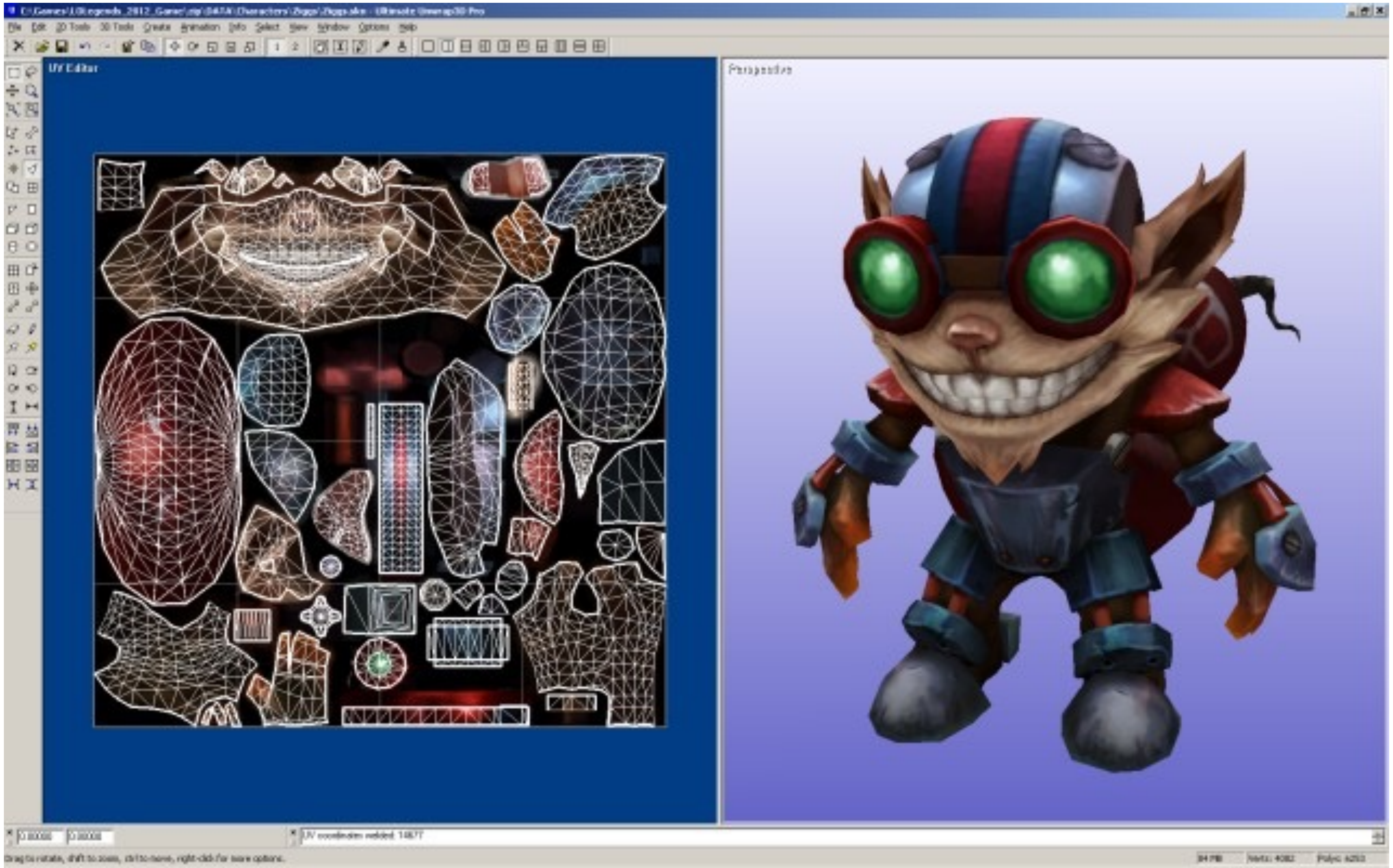


Resident Evil 4 (Capcom)



How texturing works

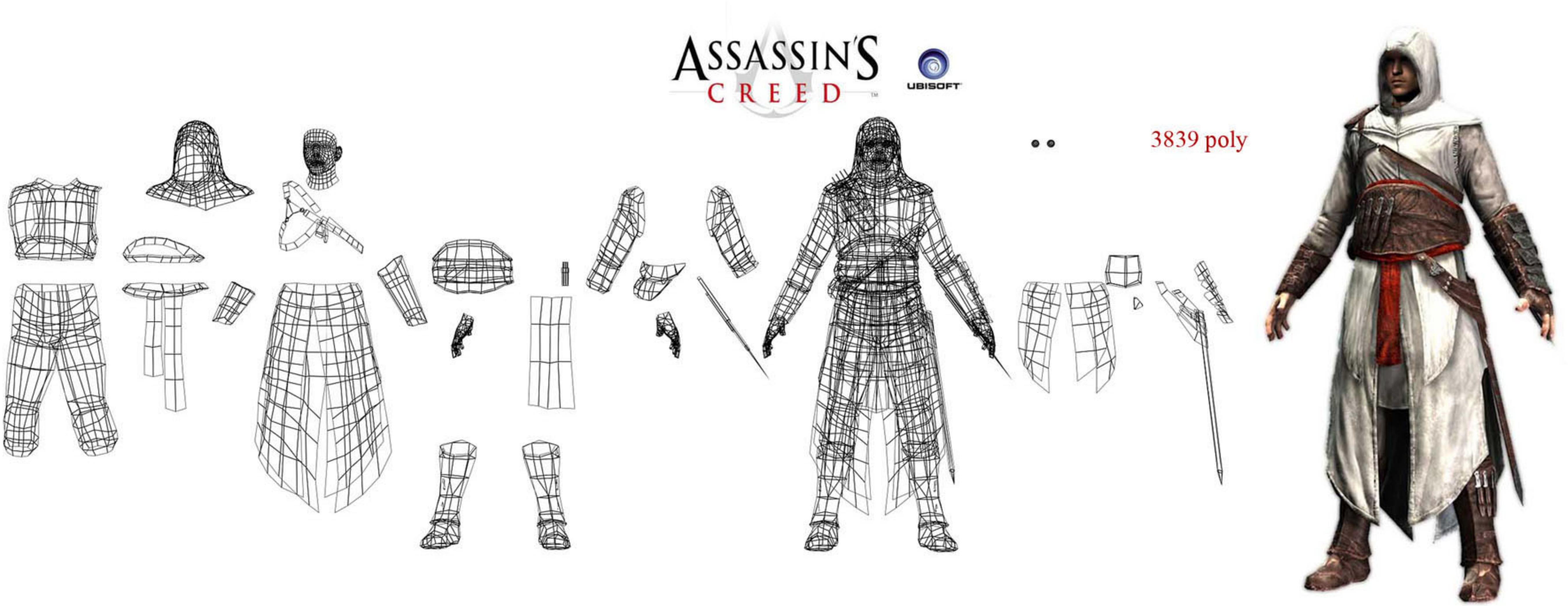
(unwrap3d)



UV coordinates welded: 14677

Drag to rotate, shift to zoom, ctrl to move, right-click for more options.

Assassin’s Creed (Ubisoft)



SHOW ME THE CODE !

The Game Loop

1. clear the screen
2. manage input
3. update game logic
4. redraw everything
5. back to 1

Getting a window

GLUT
GLFW
pyglet

Python and OpenGL

pyglet
pygame
PyOpenGL

PyOpenGL

ctypes and numpy

Requirements

an OpenGL context

a bit of CG theory

GLSL

Conventions

Right-handed

Z on forward (increasing over the viewer)

0, 0, 0 on the center of the world

column-major matrices

Old OpenGL

The static pipeline

lot of algorithms included in the gfx card itself

basically no way to introduce new algorithms

easy to start with

Modern OpenGL

lot of dedicated memory storage (for vertices, textures ...)

an api for triangles rasterization

welcome to the GPU concept

high learning curve

GLSL

OpenGL Shading Language

pseudo-C

not hard by itself (if you know what you want to do)

The modern pipeline

Define Vertex array objects (VAO)

Upload Vertex buffer objects (VBO)

Upload textures (optional)

Upload Shaders

Draw call

Vertex shader (for each vertex)

Rasterization (via interpolation)

Fragment shader (for each pixel !)

Creating the VAO

bind it

create a VBO for vertices data

upload vertices data to the GPU

map the VBO to the first VAO attribute

Defining a triangle

3 vertices

|

3 vectors

|

3 vector3 (or vector2 for 2D)

|

3 vector3f (or vector2f for 2D)

|

3 * 3 (or 2 for 2D) * float

Shaders

Create a Vertex Shader
Create a Fragment Shader
Compile them
Link them to a program

Back to code again

drawing a simple triangle

OpenGL default state

width -1 -> 1

height -1 -> 1

forward 1 -> -1

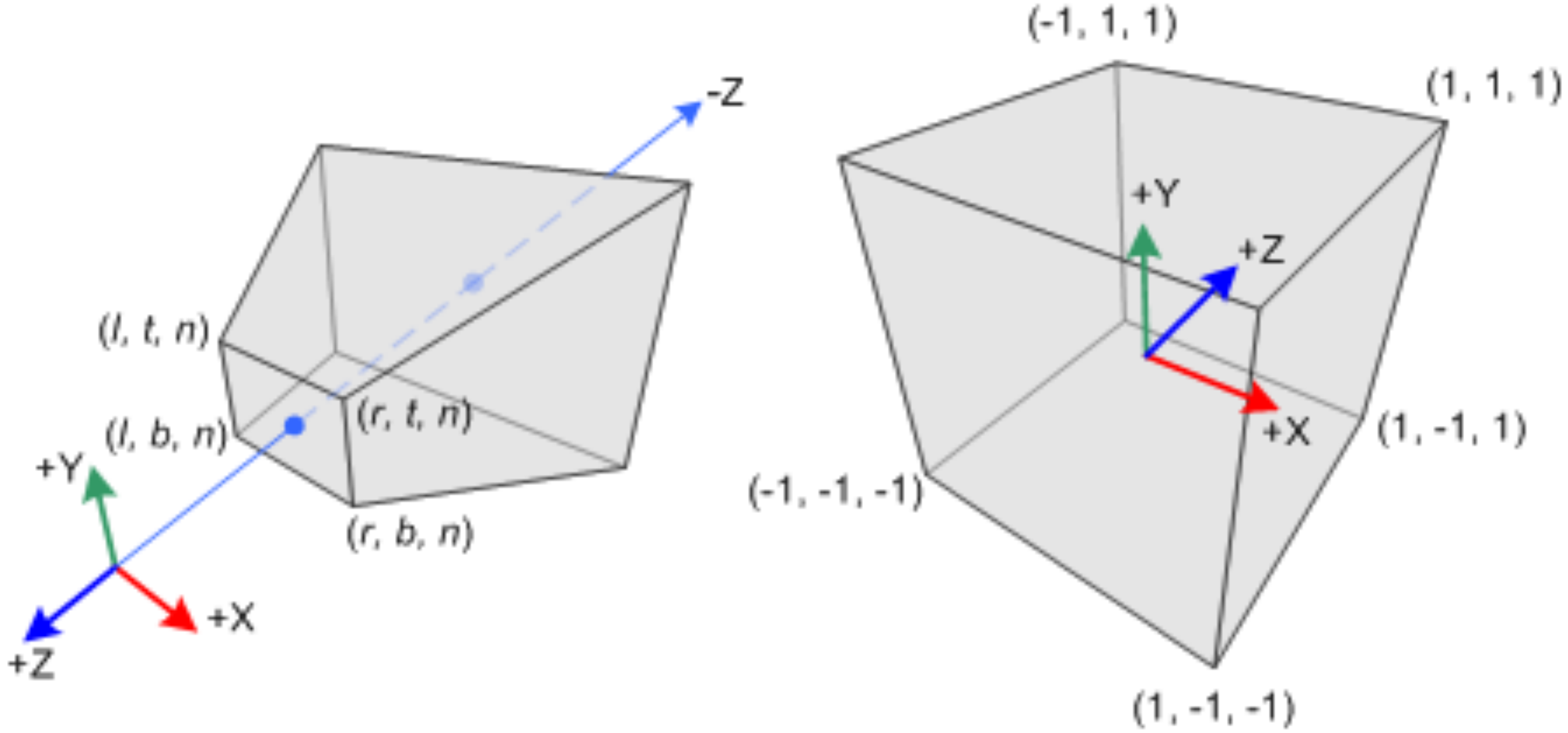
Adding the third dimension

giving meaning to the Z axis

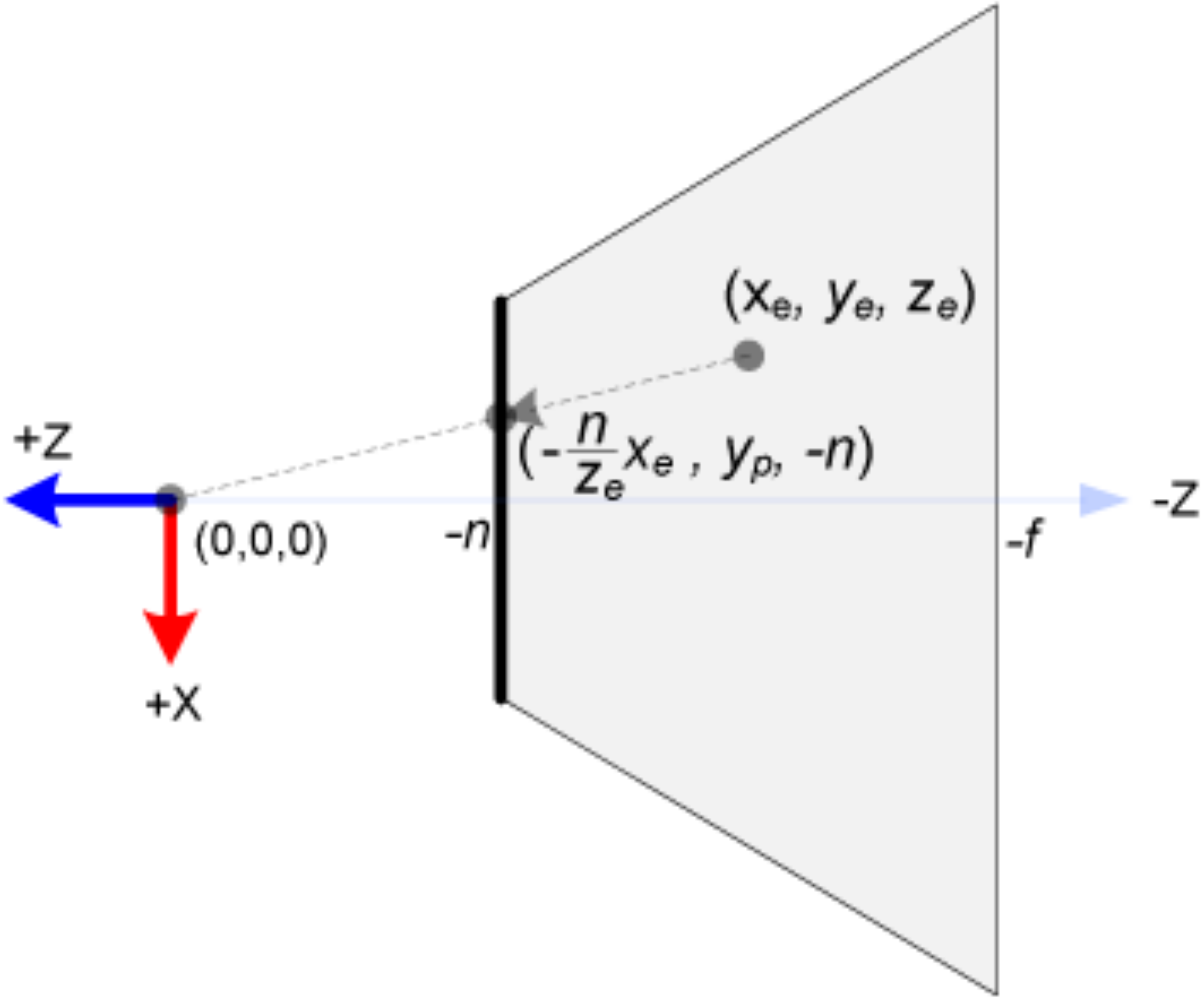
The camera paradox

Move the world not the camera
(as the camera does not exist)

Perspective



Solving perspective



The 3d transformations pipeline

local -> world
world -> camera
camera -> projection
next vertex please !

Translation matrix

$$\begin{bmatrix} 1 & 0 & 0 & X \\ 0 & 1 & 0 & Y \\ 0 & 0 & 1 & Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + X \cdot 1 \\ y + Y \cdot 1 \\ z + Z \cdot 1 \\ 1 \end{pmatrix}$$

Scale Matrix

$$\begin{bmatrix} SX & 0 & 0 & 0 \\ 0 & SY & 0 & 0 \\ 0 & 0 & SZ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} SX \cdot x \\ SY \cdot y \\ SZ \cdot z \\ 1 \end{pmatrix}$$

Rotation matrices

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ \cos\theta \cdot y - \sin\theta \cdot z \\ \sin\theta \cdot y + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x + \sin\theta \cdot z \\ y \\ -\sin\theta \cdot x + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x - \sin\theta \cdot y \\ \sin\theta \cdot x + \cos\theta \cdot y \\ z \\ 1 \end{pmatrix}$$

Combining matrices

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \cdot \begin{bmatrix} A & B & C & D \\ E & F & G & H \\ I & J & K & L \\ M & N & O & P \end{bmatrix} =$$

$$\begin{bmatrix} aA + bE + cI + dM & aB + bF + cJ + dN & aC + bG + cK + dO & aD + bH + cL + dP \\ eA + fE + gI + hM & eB + fF + gJ + hN & eC + fG + gK + hO & eD + fH + gL + hP \\ iA + jE + kI + lM & iB + jF + kJ + lN & iC + jG + kK + lO & iD + jH + kL + lP \\ mA + nE + oI + pM & mB + nF + oJ + pN & mC + nG + oK + pO & mD + nH + oL + pP \end{bmatrix}$$

Meshes (finally drawing in 3D)

welcome perspective

the OBJ format

Z-fighting

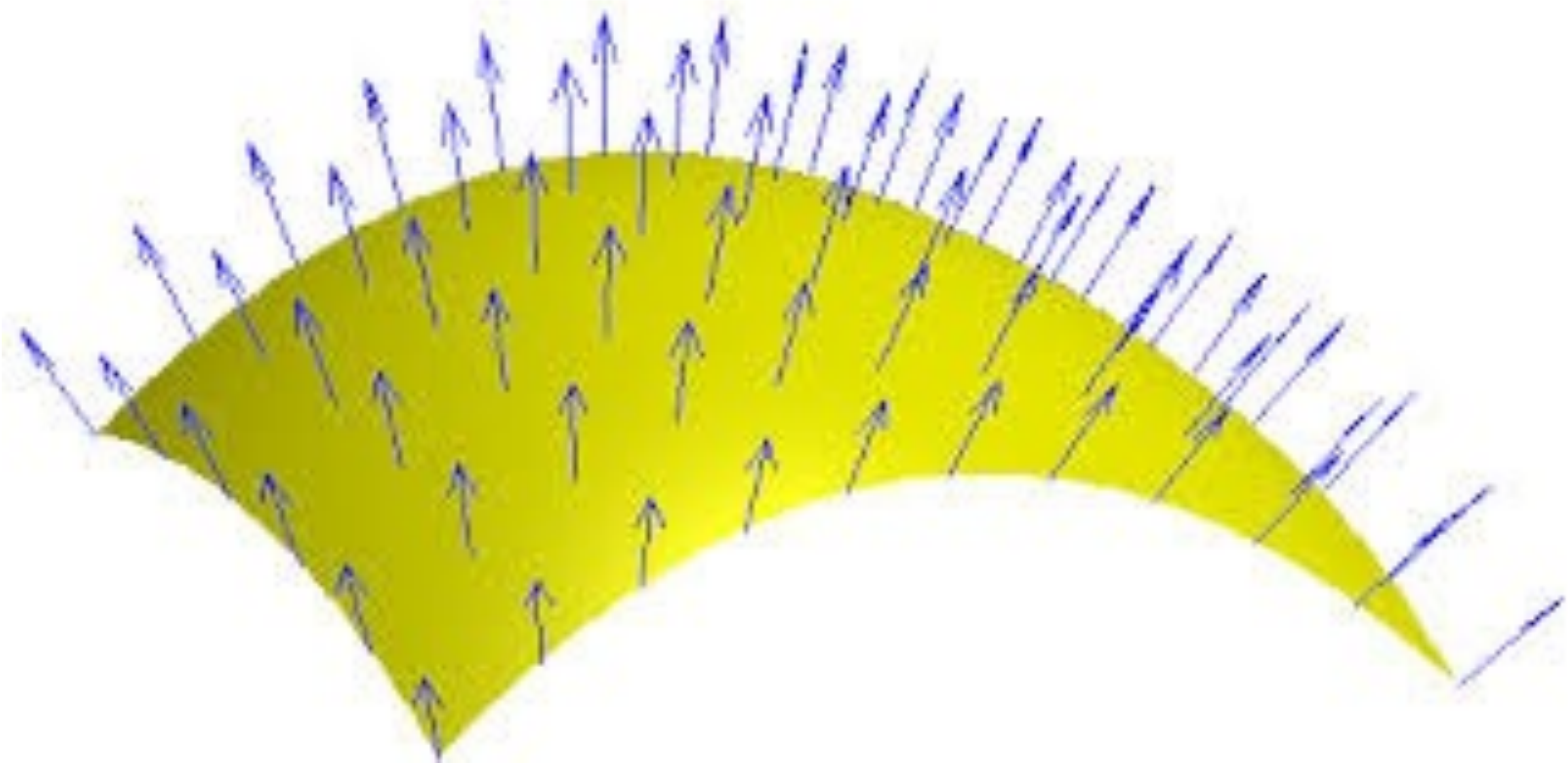
Lighting

forget about accuracy

raytracing and pathtracing are a no-go for realtime

sacrifice indirect lighting ?

Normals



Shading approaches

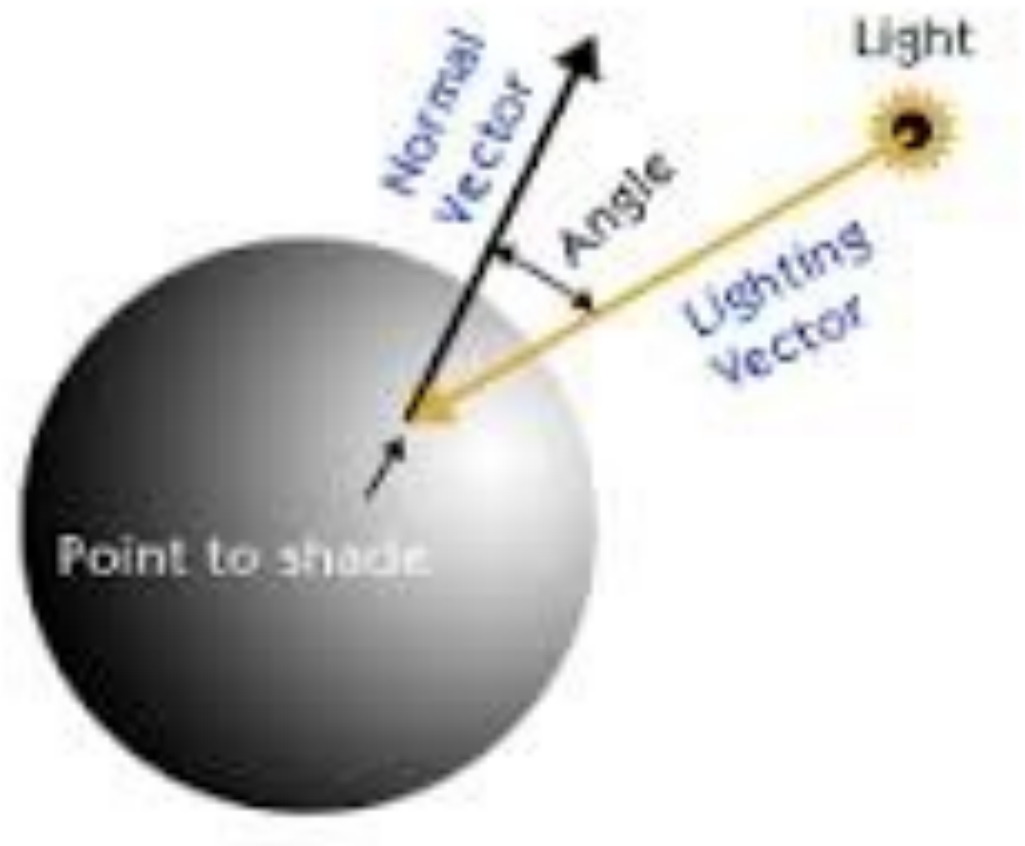
Flat shading

Gouraud shading

Phong shading

+PBR

Lambert diffuse lighting



Texturing

texture units

creation and upload

Is it enough for a game ?

Going AAA

(… but think about it)

Baking lights

pre-compute static objects shading as textures

pre-compute shadows

pre-compute indirect lighting
(to obtain effects like color bleeding)

Advanced topics

skeletal meshes

stencil buffer

instancing

a lote more thing but ...

Videogames are games !

they must be fun, being beatiful is “optional”
(think about old classics)

Hey, what about Vulkan ?

https://www.khronos.org/vulkan/

Thanks (and some useful link)

http://www.scratchapixel.com/

https://open.gl/

http://lodev.org/cgtutor/raycasting.html

https://www.facebook.com/aiv01

https://github.com/aiv01