
Graphics
"It's Europe! We see the light of Europe!"

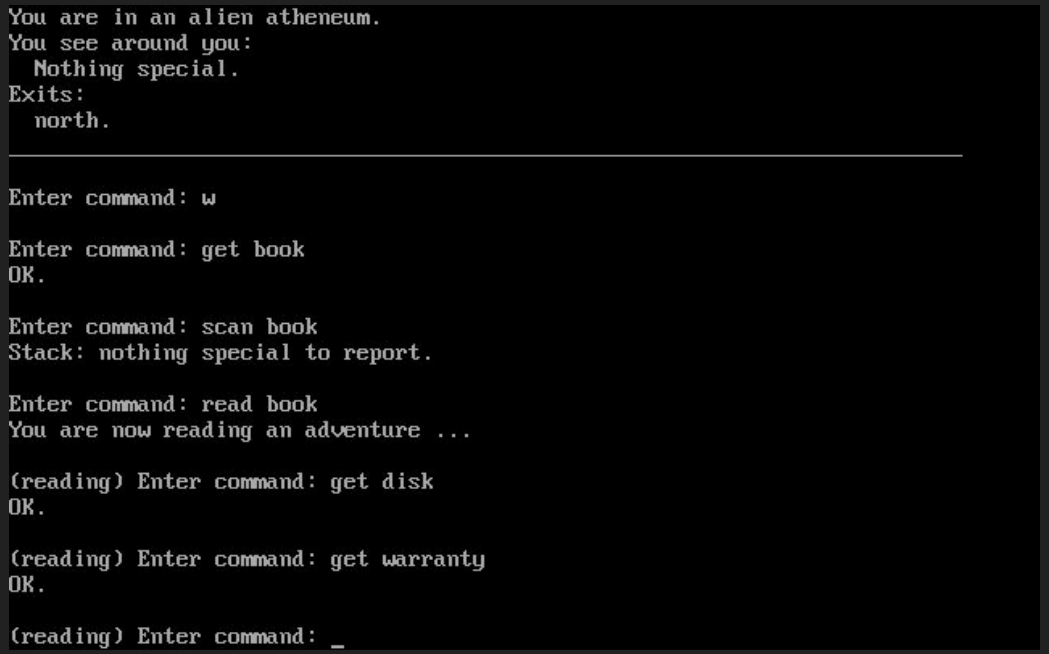

It is a way to share imagination.

Given an idea,
how do you translate it
into something others can see?


Let us start at the beginning.

# Attempt 1

# Text



# One can use ascii art to paint primitive worlds.

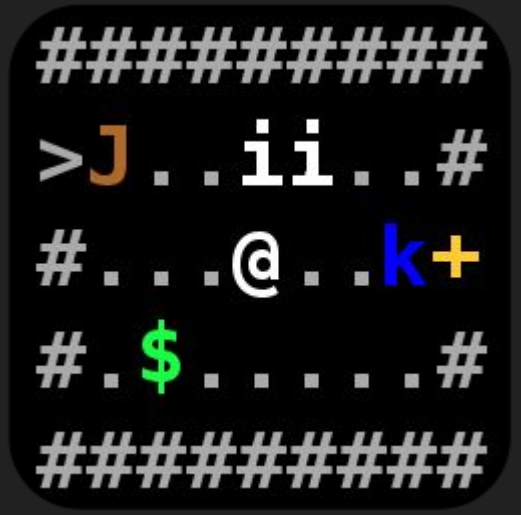
# Text can either be pre-defined, or automatically generated.

System.out.println("mms*mmmmmmkmmmmk^.,_^!skmmmmmmmmmmmmm");
System.out.prinln("MMMMMMmMMtkMsMMoMt*^_,^^oEMMMMmsi(tsZGEM");
System.out.println("M,MMMM^MM_CCmMMMMMi(*!*mMMMMMMmoKMMM");
.....

Colours may be achieved by using special terminal codes.

e.g., System.out.println("\u001B31;1mhello world!");

However, this is unreliable and system-specific.

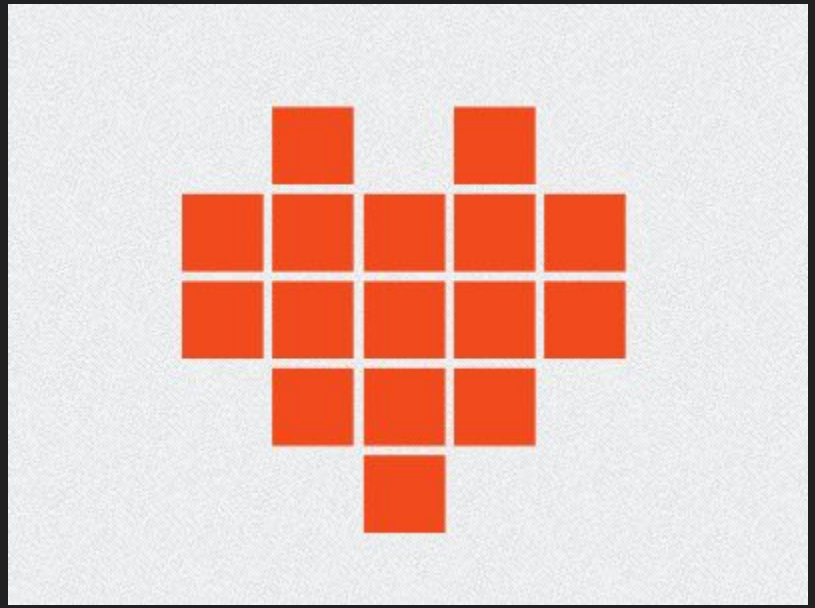


We need something better.

But wait!

Text is ultimately turned into pixels.

So, let's try to go to the very heart of the matter.



# Attempt 2

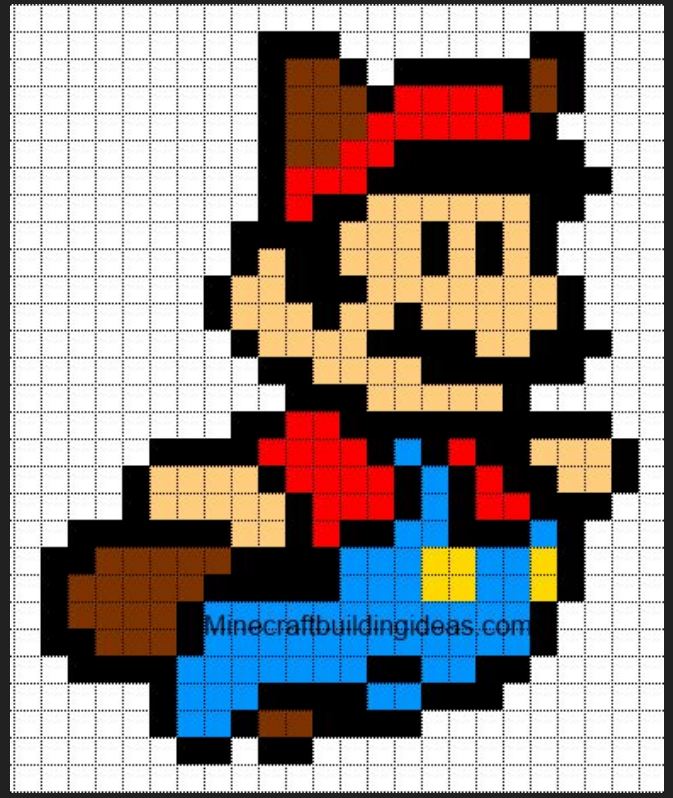
# Pixels

# 1980s

How can I directly control pixels?

```
start:
  mov ah,00          ;subfunction 0
  mov al,mode            ;select mode 18 (or 12h if prefer)
  int 10h            ;call graphics interrupt
;==========================
  mov al,colour          ;colour goes in al
  mov ah,0ch
  mov cx, x_start        ;start drawing lines along x
drawhoriz:
  mov dx, y_end          ;put point at bottom
  int 10h
  mov dx, y_start        ;put point on top
  int 10h
  inc cx             ;move to next point
  cmp cx, x_end          ;but check to see if its end
  jnz drawhoriz
drawvert:            ;(y value is already y_start)
  mov cx, x_start        ;plot on left side
  int 10h
  mov cx, x_end          ;plot on right side
  int 10h
  inc dx             ;move down to next point
  cmp dx, y_end          ;check for end
  jnz drawvert
```

# Create pixel art.

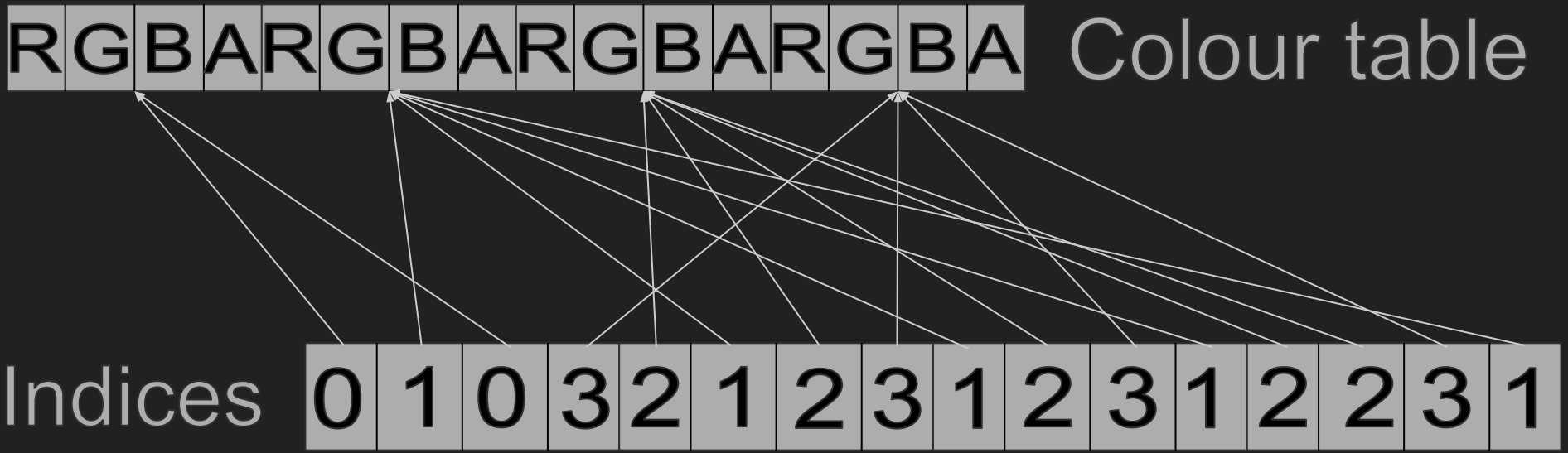


# RGBA Images

These store the colour value of each pixel in an array.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| R | G | B | A | R | G | B | A | R | G | B | A | R | G | B | A |
| R | G | B | A | R | G | B | A | R | G | B | A | R | G | B | A |
| R | G | B | A | R | G | B | A | R | G | B | A | R | G | B | A |
| R | G | B | A | R | G | B | A | R | G | B | A | R | G | B | A |

# Indexed images



Blit it onto the screen!

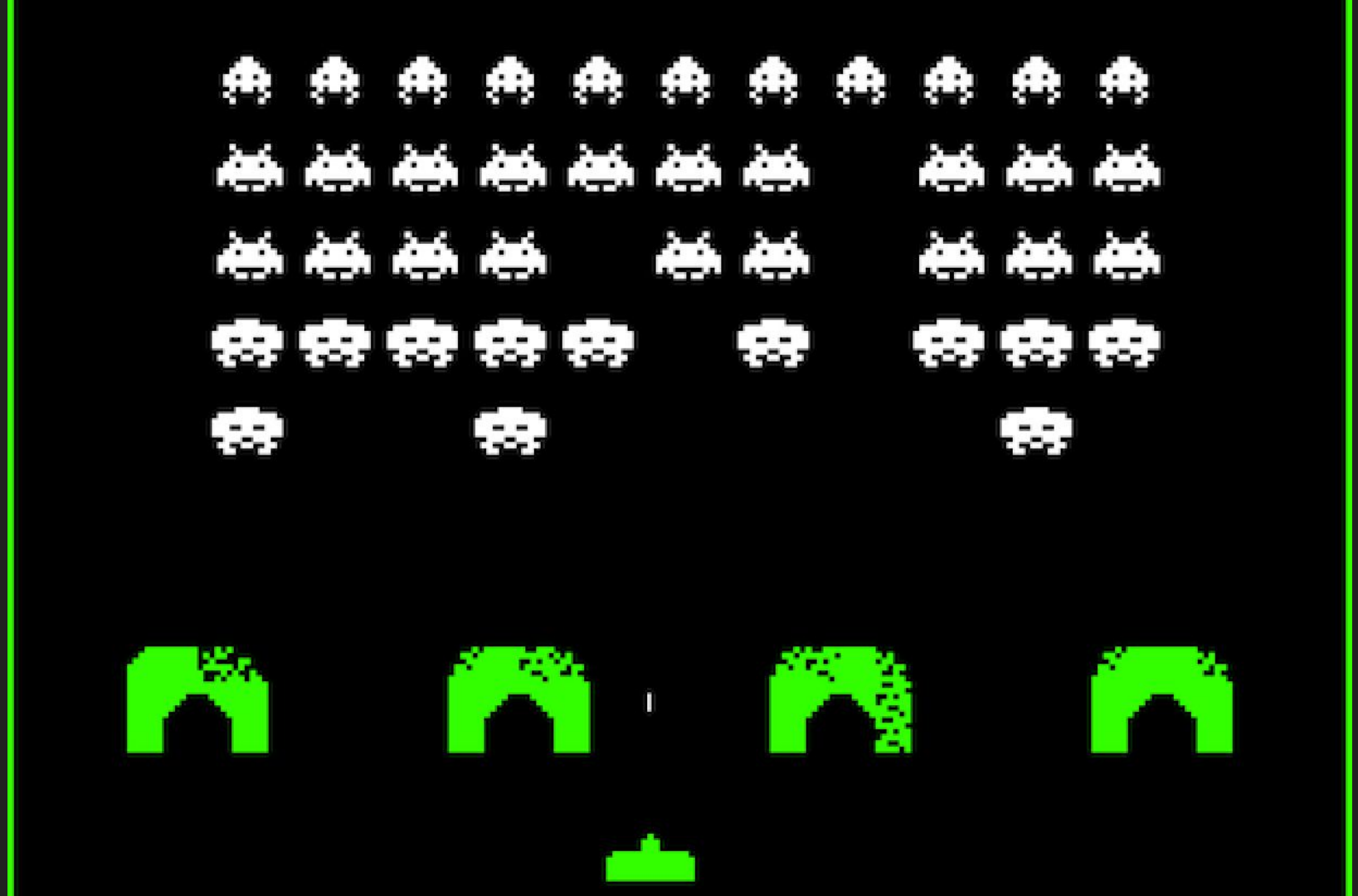
By reusing the same resource,
it is possible to save on memory
and work.

A common technique
is tiling:
reusing squares.

VS

Animate by only drawing sub-regions of your source image.


LET THE GAMES BEGIN!




Protoss Zealot
Hallucination
MENU

?

Rotating and scaling an image requires tremendous computational power.

It requires memory space, and is lossy.

Extra algorithms are needed to minimise quality loss.

Original

Upscaled and back

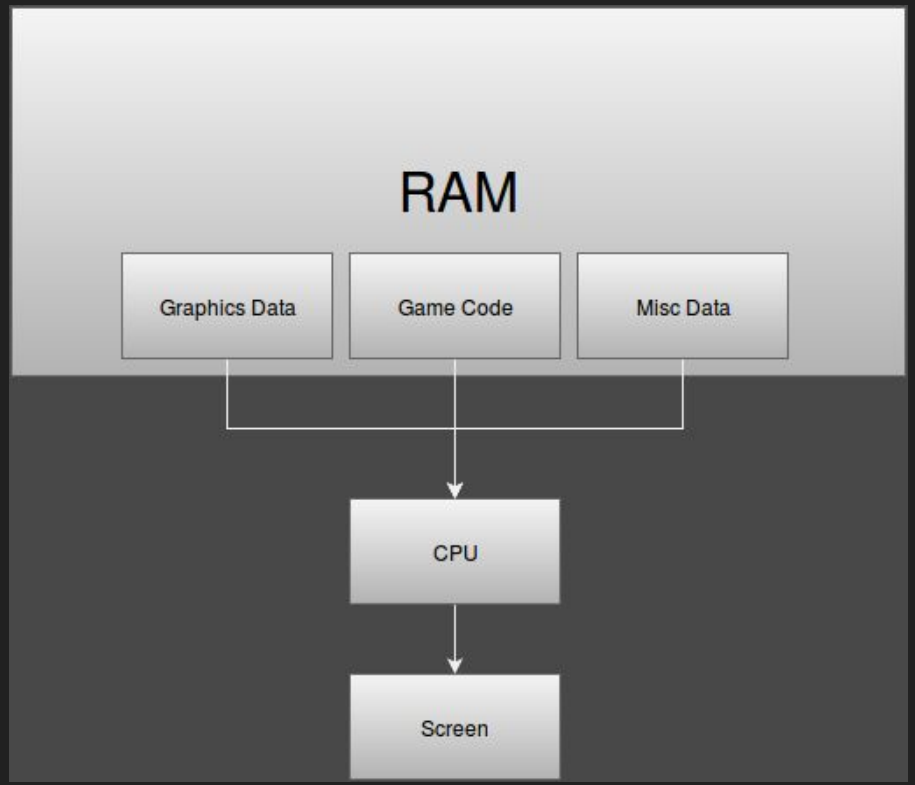
Rotated and back

# 2015

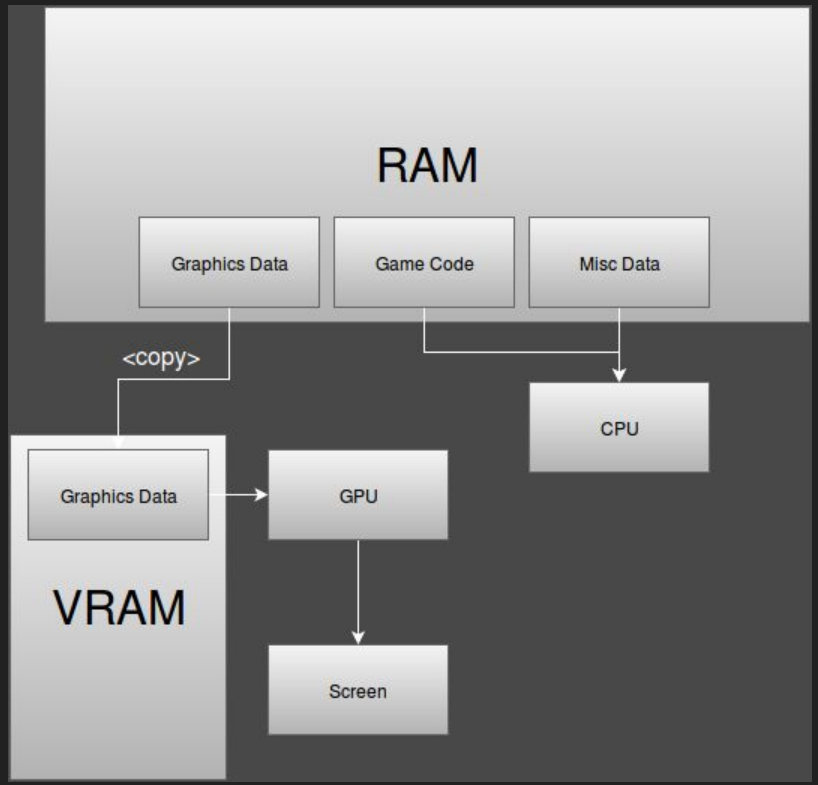
How can I directly control pixels?

you can't.

# 1980s


RAM
Graphics Data
Game Code
Misc Data
CPU
Screen

# 2015


RAM
Graphics Data
Game Code
Misc Data
<copy>
CPU
Graphics Data
GPU
VRAM
Screen

The GPU is a dedicated component
for drawing graphics.


It does things slowly,
but it is very concurrent,
sometimes having 1000+ cores.

Ideal for processing many small things
(pixels, shapes, etc.)

It has its own memory to optimise drawing.

However, this memory is not accessible from the CPU.

To draw something,
you first copy it to GPU memory
and then tell the GPU to draw it.

This is handled for you by LibGDX.

Sometimes, you might want direct control over what the GPU does.

This is achieved using OpenGL.
(an alternative is DirectX)

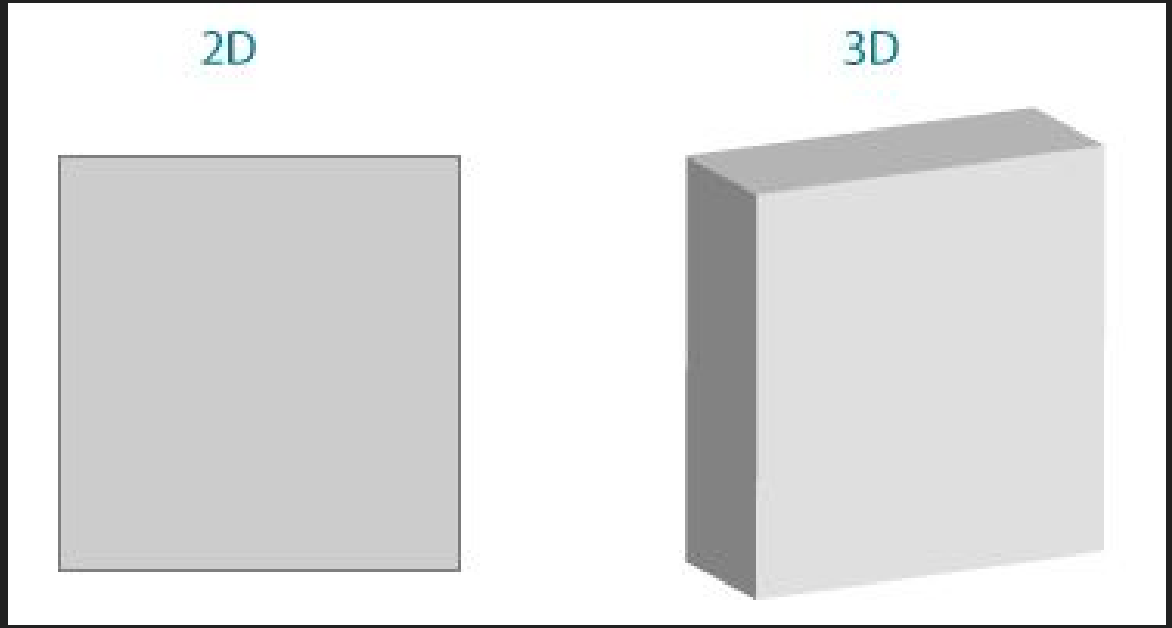
# OpenGL Example for sending an image to the GPU (in C++)

```cpp
//The image details in RAM
int width = /* image width */;
int height = /* image height */;
char data[] = /* image data */;

//An ID representing a copy of the image inside the GPU.
GLuint textureID = ~0u;

//Tell the GPU to generate one empty image, and tell us its ID for future reference.
glGenTextures(1, &textureID);

//Tell us that this is the image we currently want to work with.
glBindTexture(GL_TEXTURE_2D, textureID);

//Copy our image to the GPU for future use.
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
```

Let us leave 2D for now,
and move towards 3D.

# 2D vs 3D?

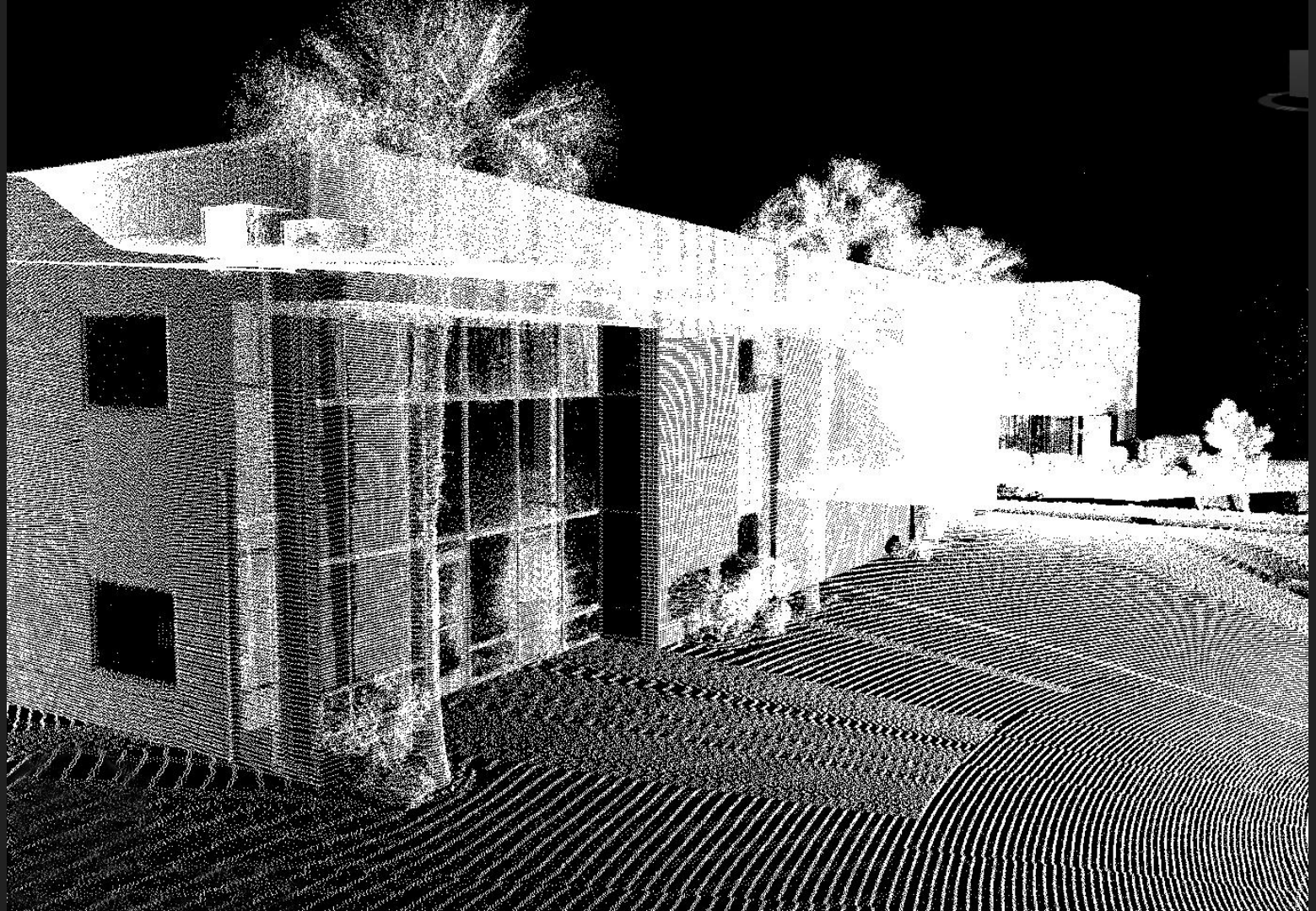
(x, y)
vs
(x, y, z)



# How do we represent a 3D shape?

# Attempt 1

# 3D Points



# Advantages

Easy to handle
(they're just points!)

Easy to generate using
a laser scanner.

# Disdvantages

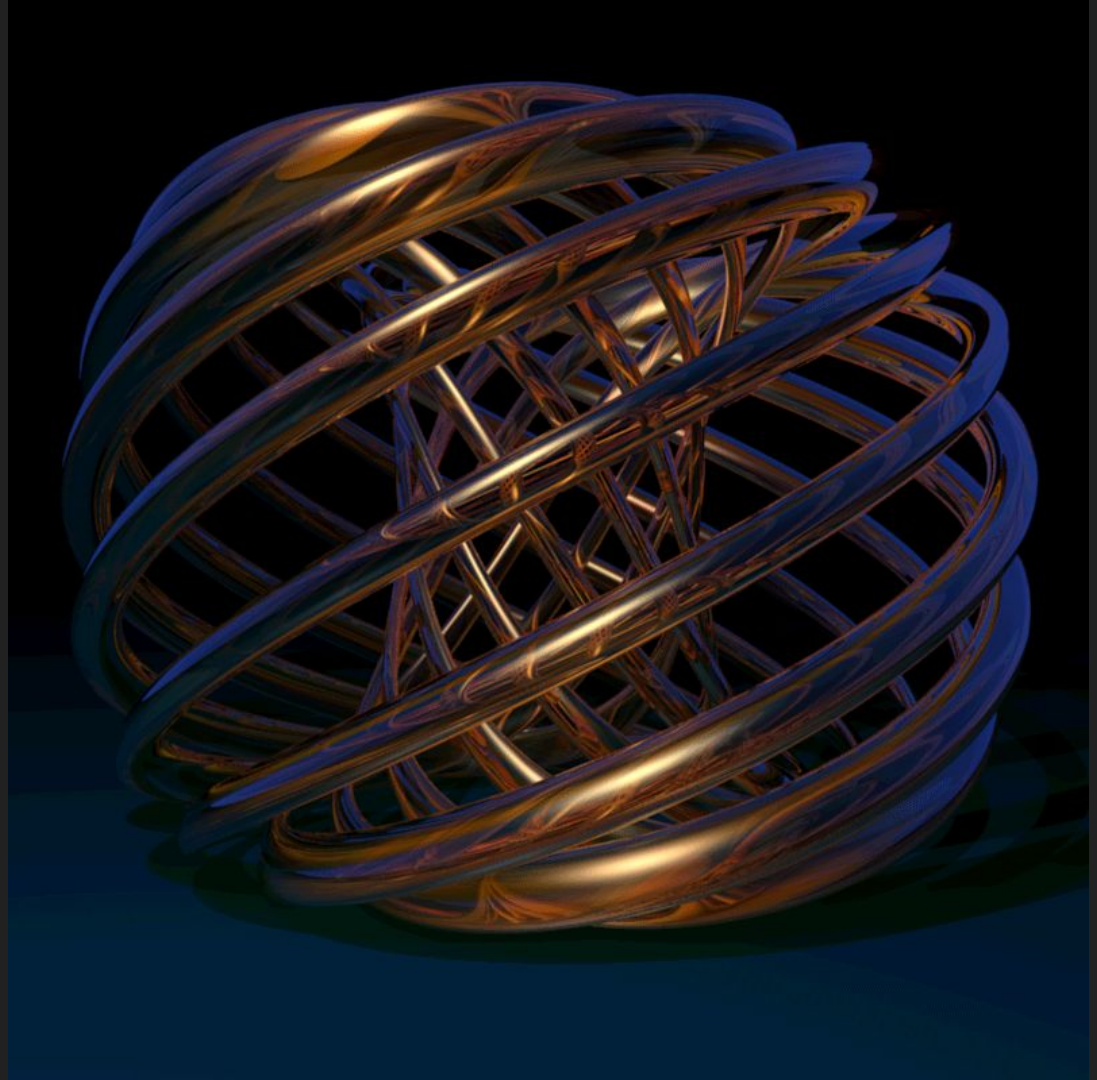
Difficult to compute
(there are billions of them!)

Gaps

# Attempt 2

# 3D functions

$$x(t) = \cos(qt) \cdot (3 + \cos(pt))$$
$$y(t) = \sin(qt) \cdot (3 + \cos(pt))$$
$$z(t) = \sin(pt)$$



# Advantages

Infinite precision

Allows intricate patterns

# Disdvantages

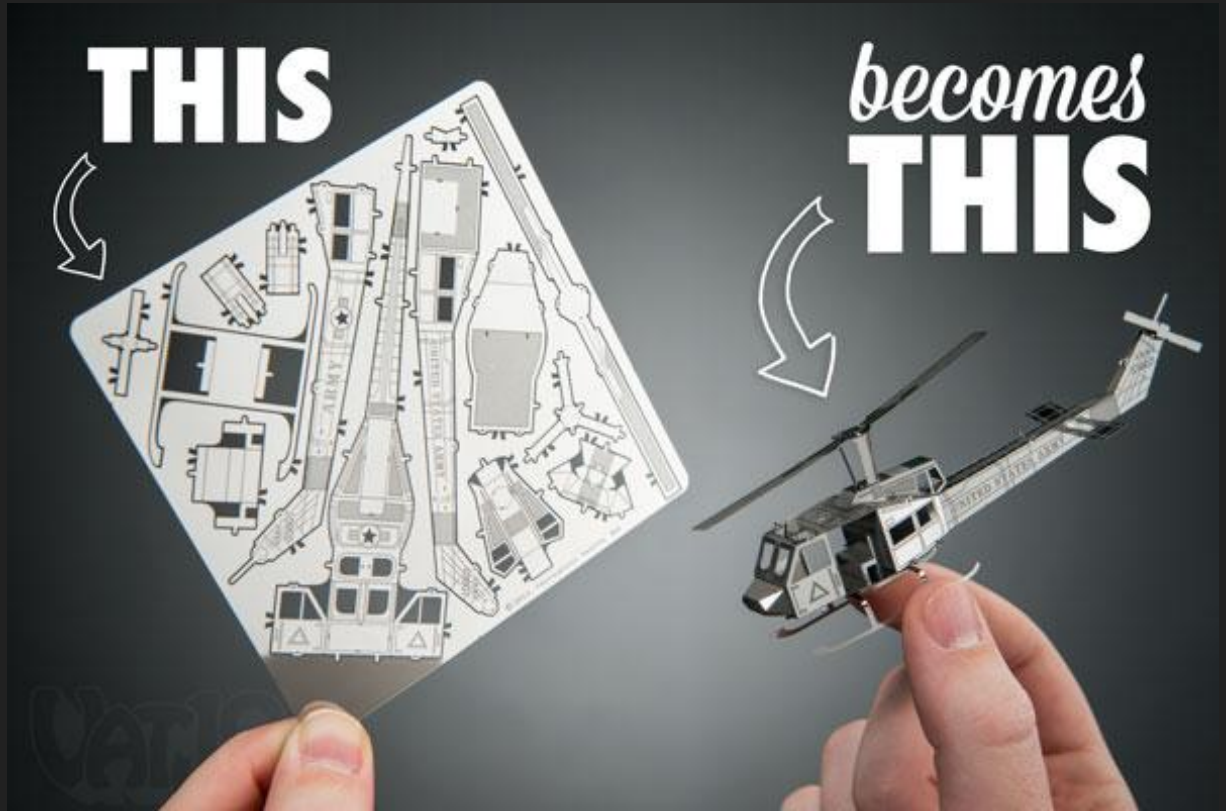
Most things are difficult to describe using maths (e.g., person)

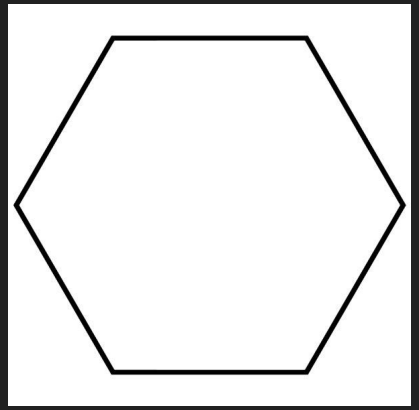
Difficult to draw efficiently

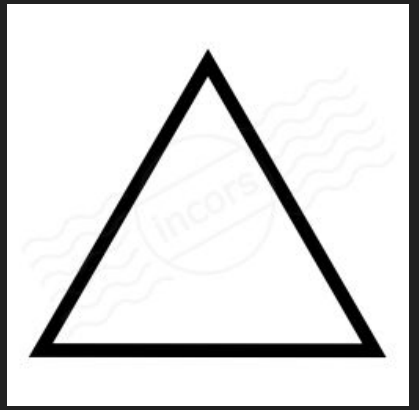
Too "perfect"

# Attempt 3

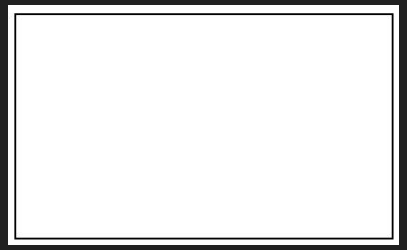
# Using 2D shapes!

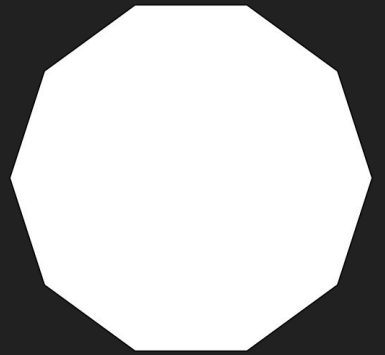




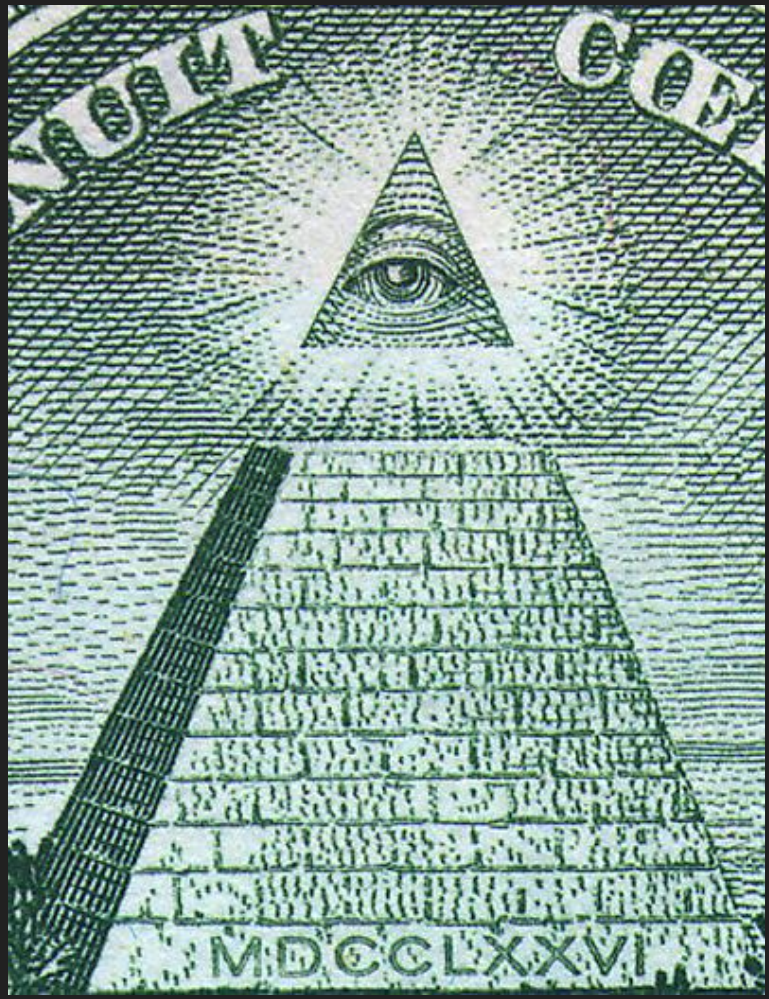


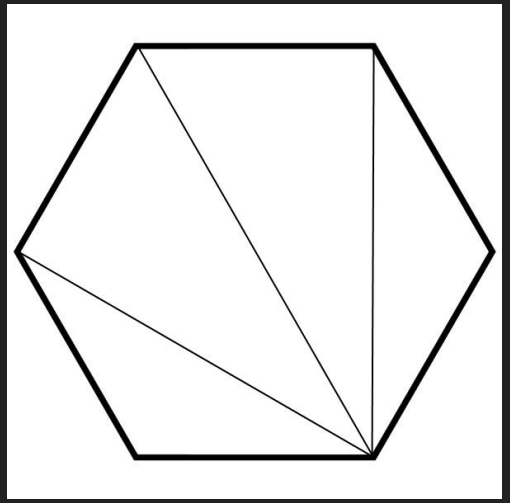
Okay,
but which shape should we use?

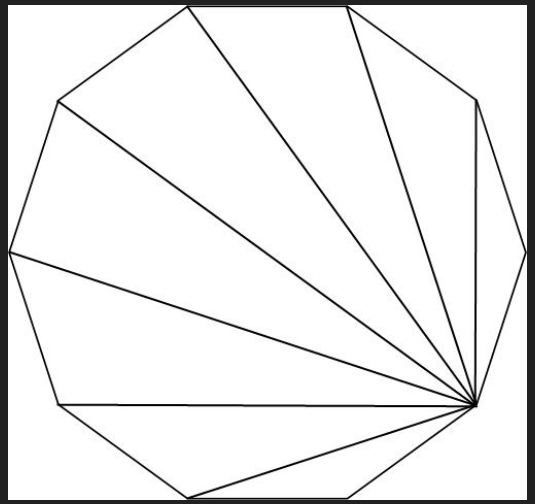


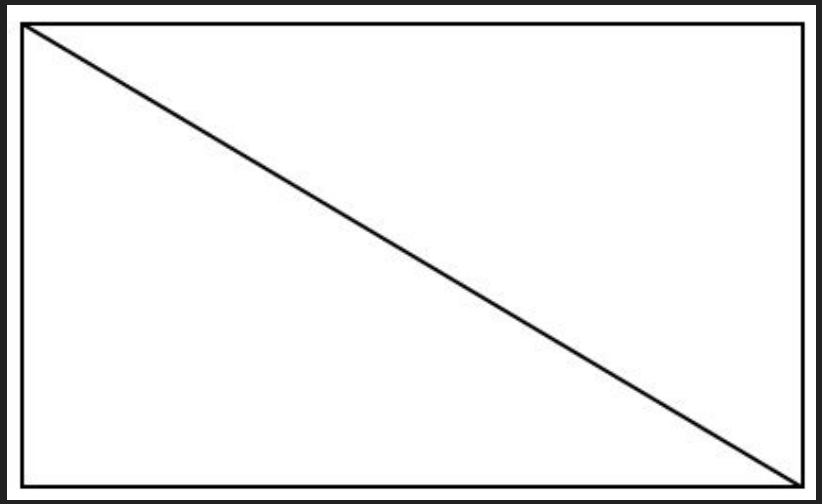


Everything is a triangle!



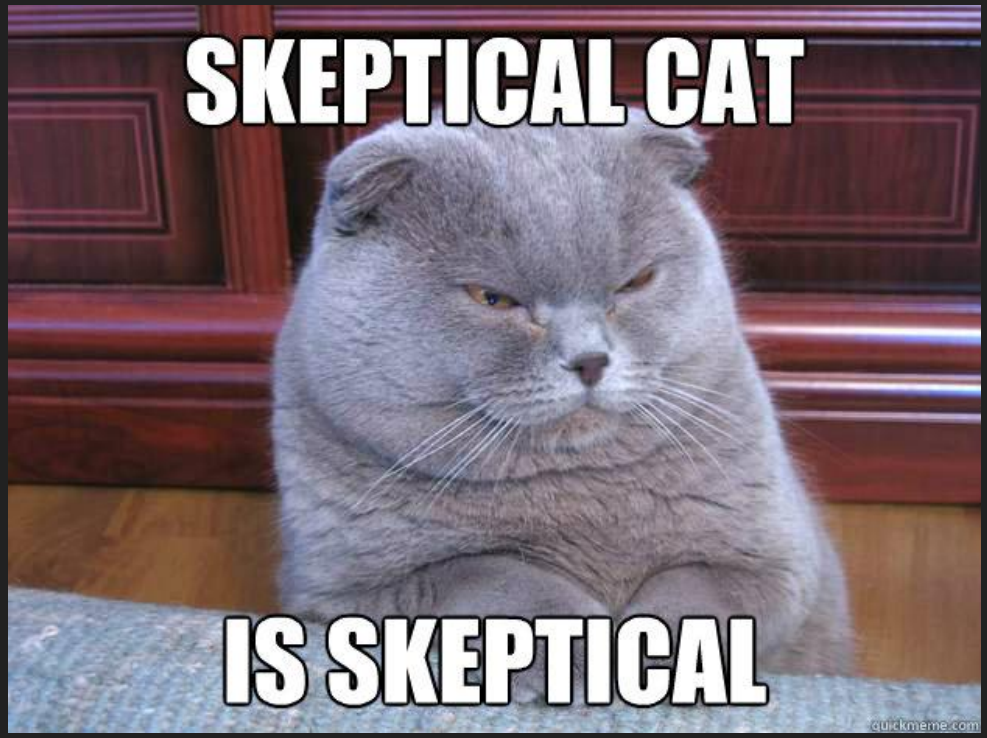




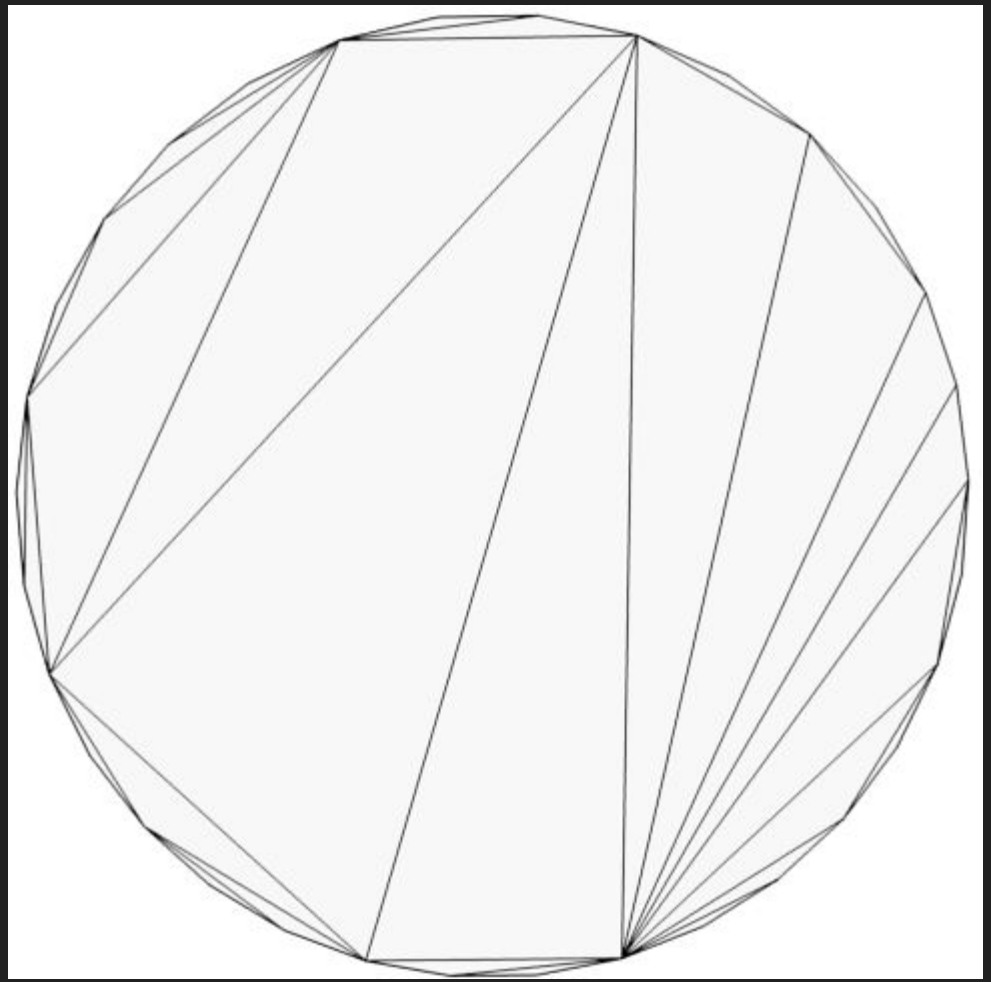


"But what about circles!"

- I hear you say.



Still just triangles.



# That's how games work

# Advantages

Simple to handle, as each shape is just a few connected points!

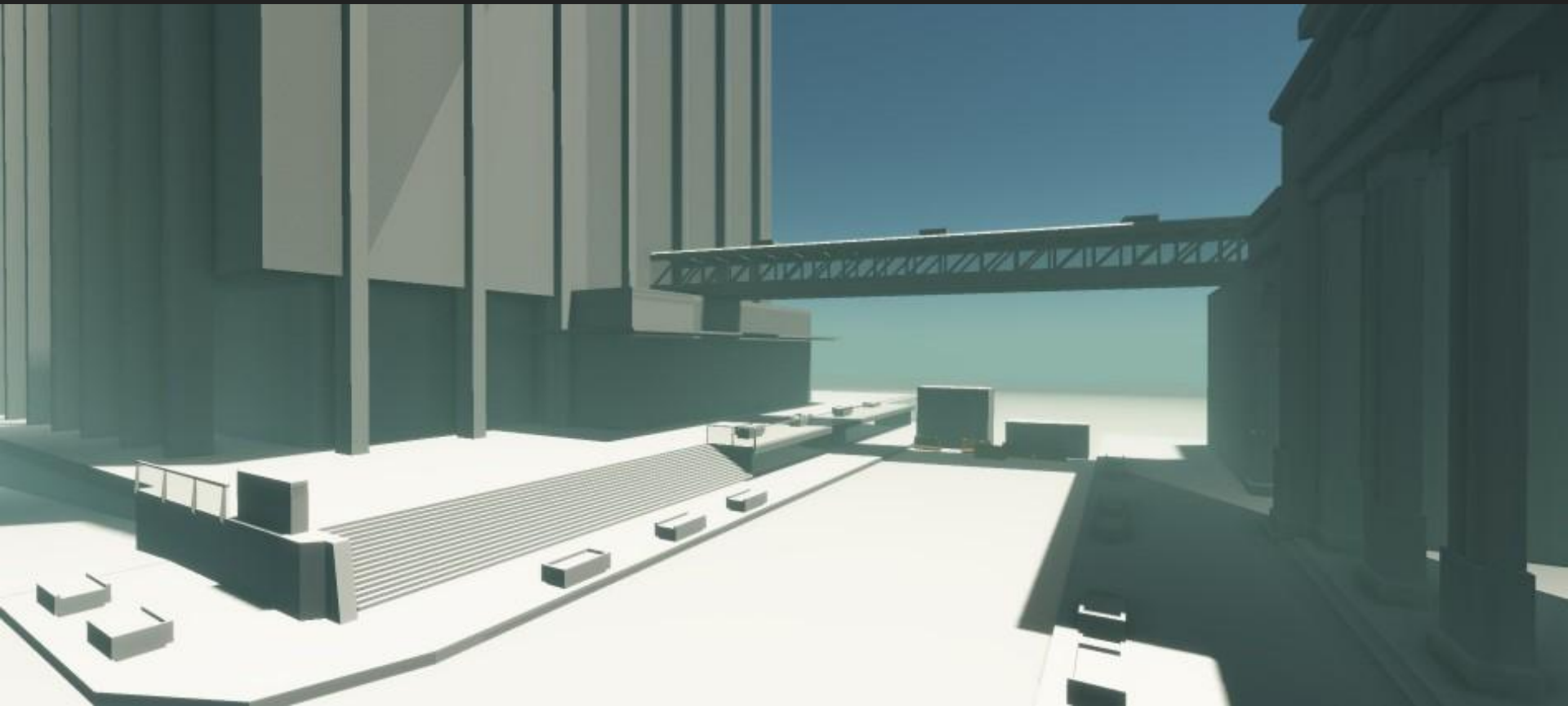
Easy to define by artists using special tools

# Disdvantages

Can appear blocky with not enough shapes ("polygons")

Representing minute details requires many triangles

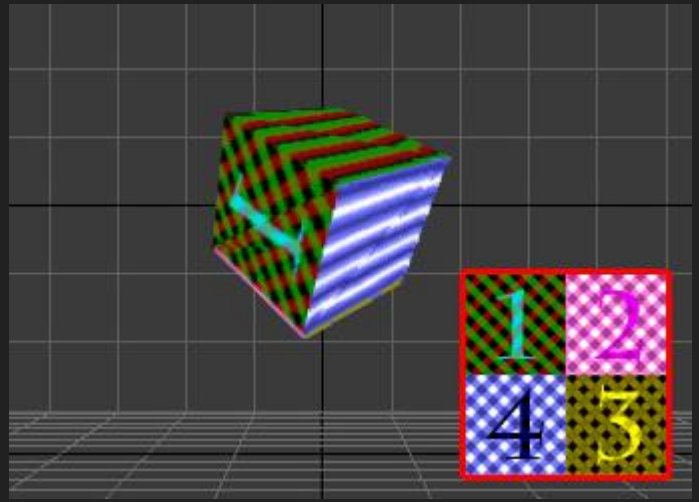
# But empty triangles are boring



How do we add "texture" to a triangle anywhere on the screen?

# Each vertex will have 5 coordinates:

(x, y, z, u, v)

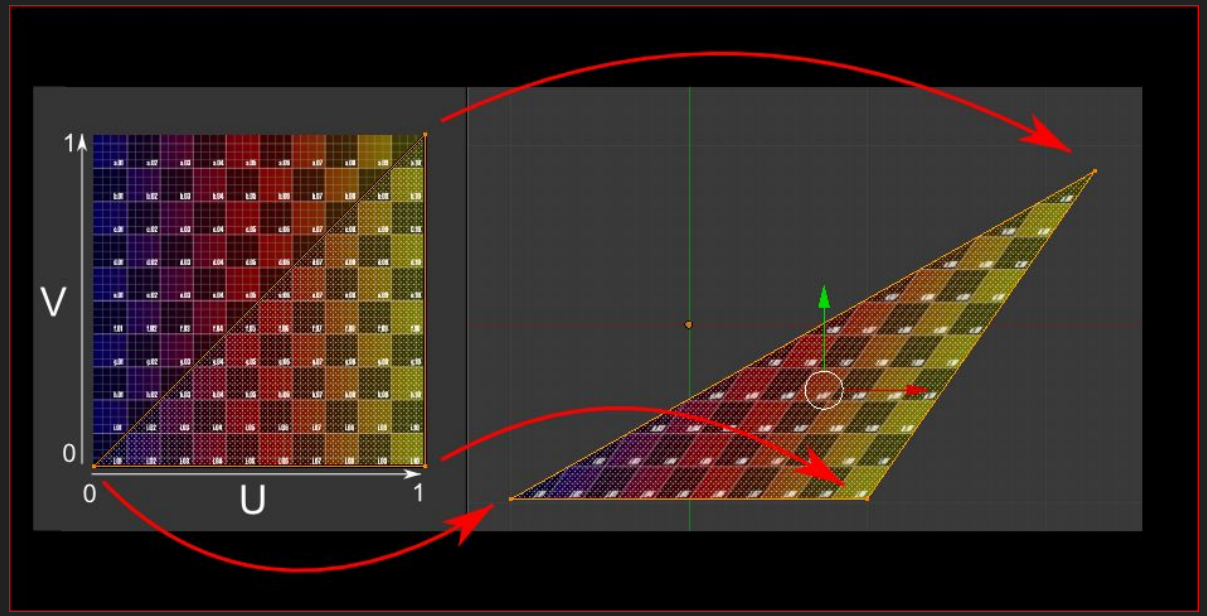


u, v

x, y, z

Texture Position

3D Position



position = transform(vertexPosition.xyz)
colour = sourceTexture(u,v)