Game programming

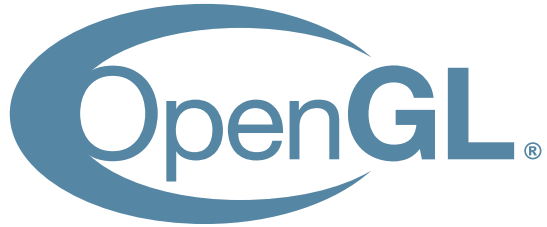
Lecture 2: Instancing and font rendering..

By Dr. Rokia Abdein

What is vertex buffer?

- An Open Graphics Library® (OpenGL®) **vertex buffer** is an area of computer memory often located directly on a graphics card that allows very fast access to an array of vertices and their properties.
- At the most basic level, an OpenGL® vertex buffer **is just a simple buffer**, an allocated area of memory in which data can be stored. It becomes a vertex buffer when a vertex array is stored within it. An OpenGL® vertex array is an array of data structures that defines all the properties of individual vertices.
- **This information can include** the X, Y and Z locations of the vertex in the 3D scene, the color of the vertex, the normal and other properties.

What is API (OpenGL, DirectX)



- An application programming interface (API) is a way for two or more computer programs or components to communicate with each other. It is a type of software interface, offering a service to other pieces of software.
- OpenGL (Open Graphics Library) is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. **The API is typically used to interact with a graphics processing unit**, to achieve hardware-accelerated rendering.

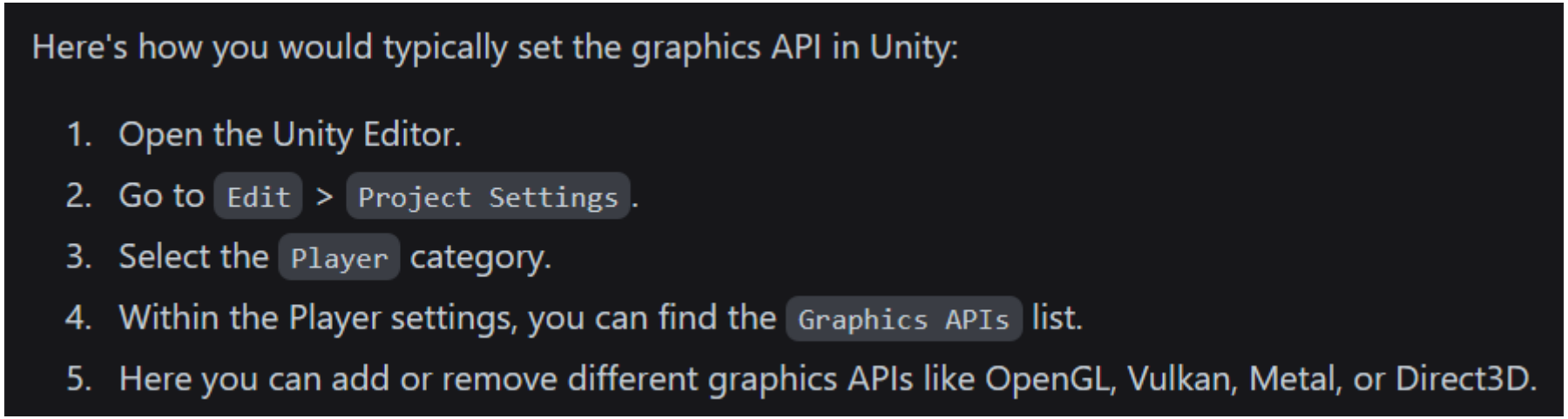
Question: Does Unity Use OpenGL?

- Unity is a versatile game engine that supports multiple graphics APIs on various platforms.
- Whether Unity uses OpenGL depends on the platform and the settings you choose for your project.
- On desktop platforms like Windows, macOS, and Linux, Unity has traditionally supported both DirectX (on Windows) and OpenGL. However, OpenGL support could be phased out or replaced with newer APIs such as Vulkan or Metal (for macOS) as they offer improved performance and more modern features.

Question: Does Unity Use OpenGL?

- In earlier versions of Unity, OpenGL was often the default for Linux and macOS, while Direct3D was the default for Windows. Unity provided the choice to switch between graphics APIs if needed.

Question: Does Unity Use OpenGL?

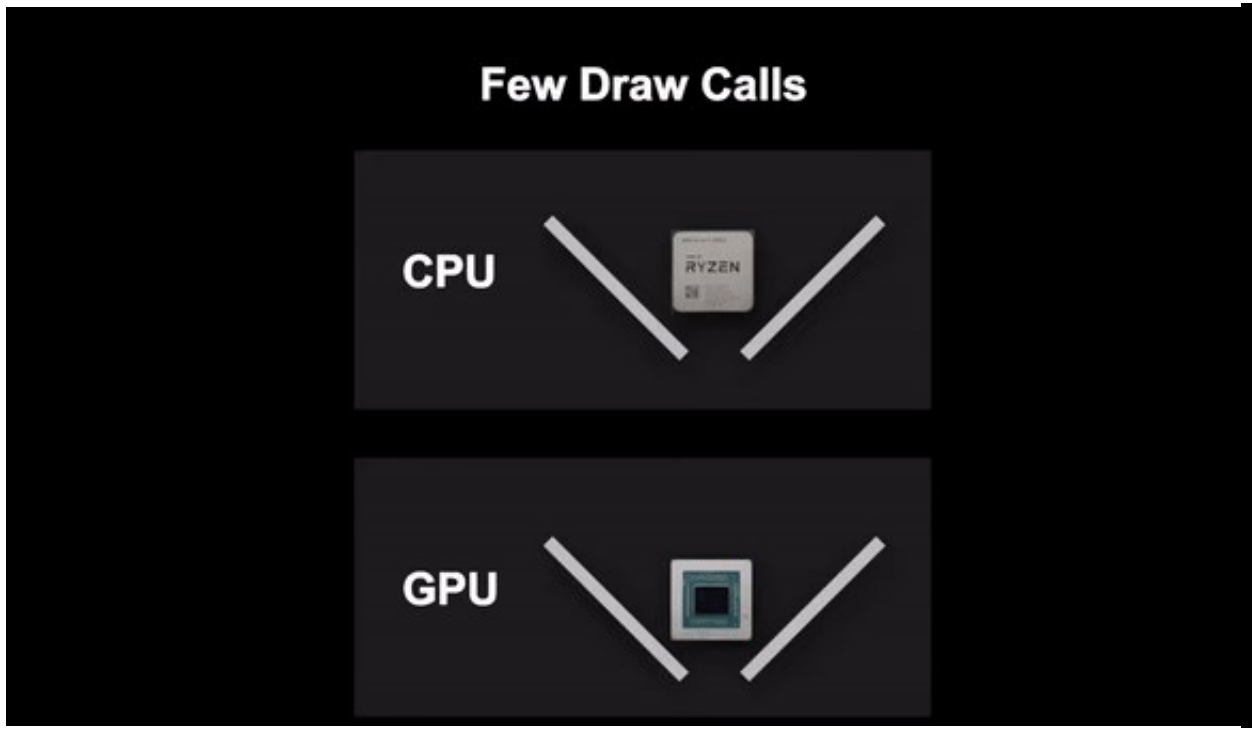


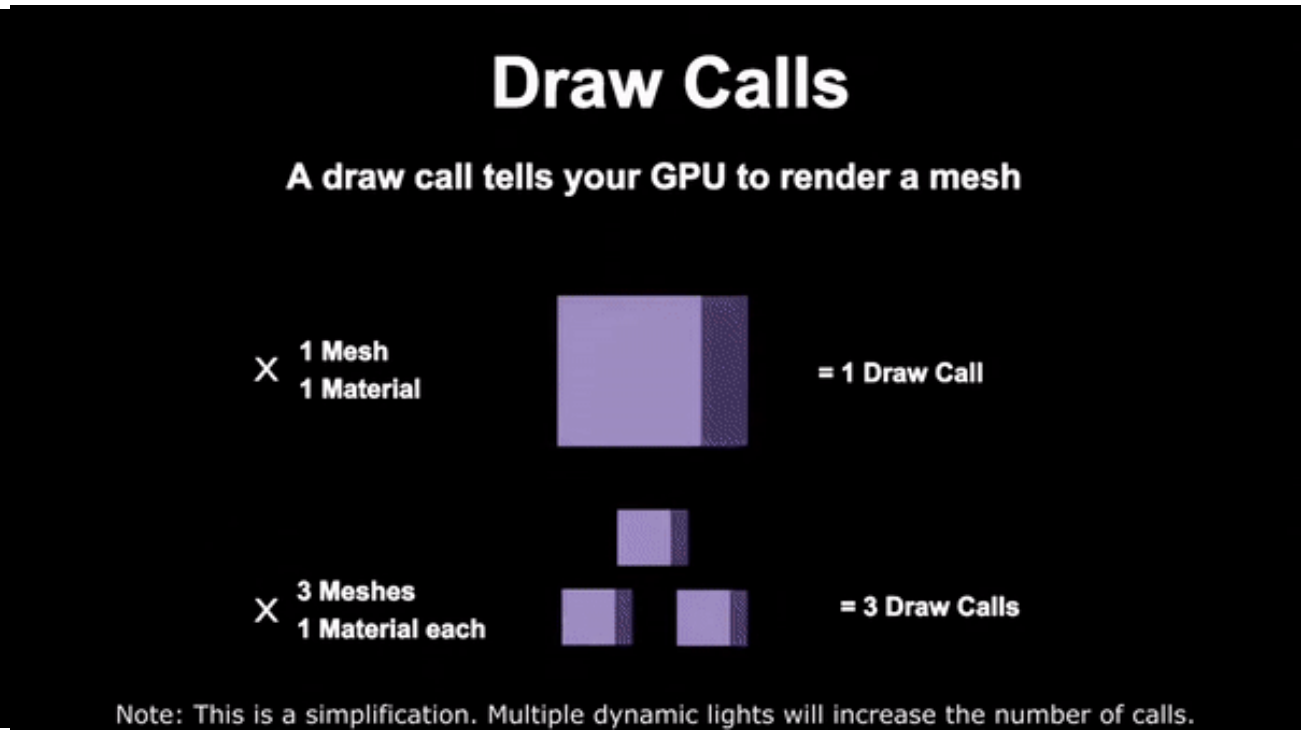
- Keep in mind that Unity automatically picks the most suitable graphics API for each platform unless you explicitly change the order or remove certain APIs from the list.

Shaders..

- In computer graphics, a **shader** is a computer program that calculates the appropriate levels of light, darkness, and color during the rendering of a 3D scene—a process known as shading.
- **Mesh**: A collection of vertices, edges, and faces that act as the foundation of a model in a video game.

What is draw call?





What is draw call?

- Draw calls is a request from CPU to the GPU through the graphic API (e.g openGL) to render a game object on the screen.
- Draw call is done in 2 phases:
  - Prepare the data to be sent (such as mesh, texture, etc...)
  - Send the request with required data to the GPU.
- But why the data need preparation?
- Because the data on the CPU is stored in a way that the GPU can't understand, so it need processing from the CPU before it can be sent, and this phase is the most expensive part in the draw call.

What is draw call?

- It's worth mentioning that the data that are sent to the GPU called "Render state", and when you send new data you "changed the render state".
- For simplicity, let's say a complete draw call cost 100ms, preparing the data require 80ms and sending the request cost 20ms.

What is draw call?

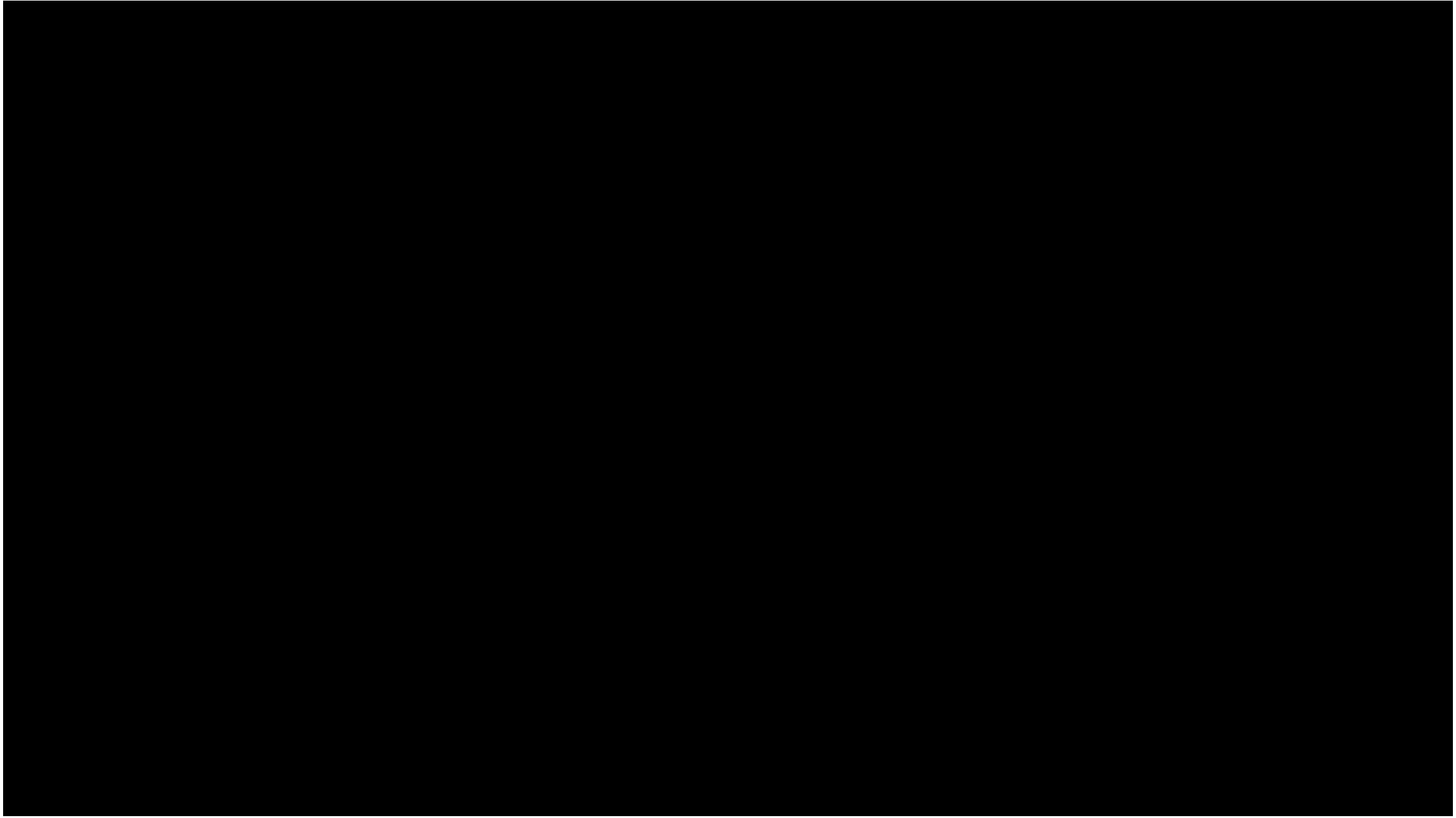
- Draw calls = how many objects are being drawn to the screen. You want to keep this number down to maintain good performance, so watch out for pixel lights as they make objects get drawn as many times as there are lights affecting them.
- to clarify... actually each material in an object drawn to screen. ie an object with 1 material = 1 draw call, an object with 4 mats = 4 draw calls. lights, shadows etc all add on top of that.
- So the Draw calls depend on number of materials being used in a scene the no of objects being rendered with materials applied on them.

What is draw call?

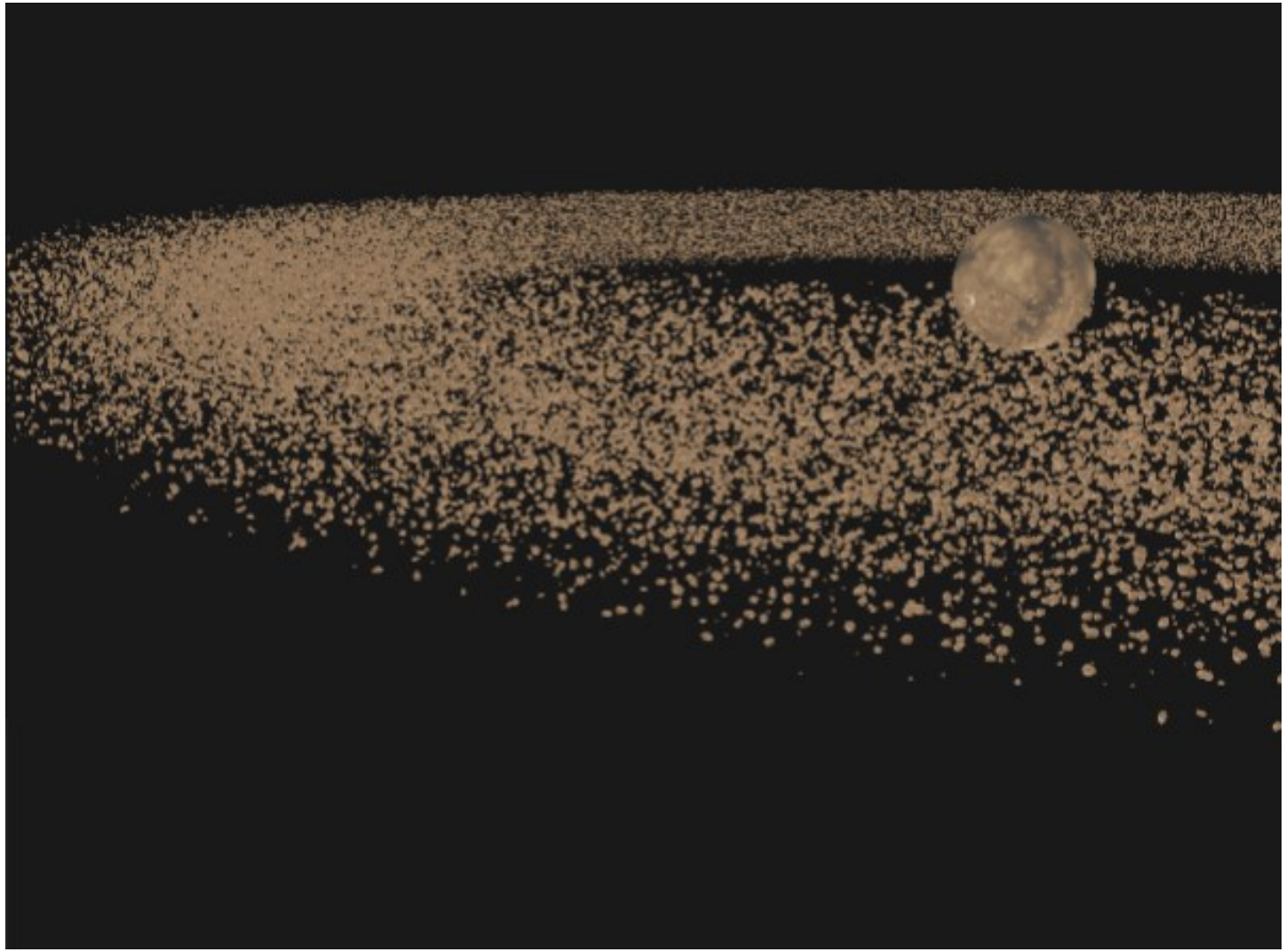
- Suppose there are a lot of objects that do not share the same material, texture, shader, etc. In that case, the Renderer has to change its state to draw the next object, and the draw calls become expensive and take more time to execute.

What is draw call?

- Say you have a scene where you're drawing a lot of models where most of these models contain the **same set of vertex data**, but with different world **transformations**.
- Think of a scene filled with **grass leaves**: each grass leave is a small model that consists of only a few triangles. You'll probably want to draw quite a few of them and your scene may end up with thousands or maybe tens of thousands of grass leaves that you need to render each frame.
- Because each leaf is only a few triangles, the leaf is rendered almost instantly. However, the thousands of render calls you'll have to make will drastically reduce performance.



Asteroid field



Instancing..

```
for(unsigned int i = 0; i < amount_of_models_to_draw; i++)
{
    DoSomePreparations(); // bind VAO, bind textures, set uniforms etc.
    glDrawArrays(GL_TRIANGLES, 0, amount_of_vertices);
}
```

When drawing many instances of your model like this you'll quickly reach a performance bottleneck because of the many draw calls. Compared to rendering the actual vertices, telling the GPU to render your vertex data with functions like glDrawArrays or glDrawElements eats up quite some performance since OpenGL must make necessary preparations before it can draw your vertex data (**like telling the GPU which buffer to read data from, where to find vertex attributes and all this over the relatively slow CPU to GPU bus)**. So even though rendering your vertices is super fast, giving your GPU the commands to render them isn't.

Instancing..

- It would be much more convenient if we could send data over to the GPU once, and then tell OpenGL to draw multiple objects using this data with a single drawing call. Enter instancing.
- Instancing is a technique where we draw many (equal mesh data) objects at once with a single render call, saving us all the CPU -> GPU communications each time we need to render an object.

Instancing..

- These *instanced* versions of the classic rendering functions take an extra parameter called the instance count that sets the number of instances we want to render. **We sent all the required data to the GPU once, and then tell the GPU how it should draw all these instances with a single call.** The GPU then renders all these instances without having to continually communicate with the CPU.
- Instancing is an optimization on the GPU that uses a sort of shared buffer to draw identical geometry without having to copy the entire mesh for each call to draw that mesh. For each unique object type (couch, chair, table, etc.) the mesh and material/textures are stored just once and a small amount of instance data (a translation/rotation/scale matrix for example) is stored for each instance.

Instancing..

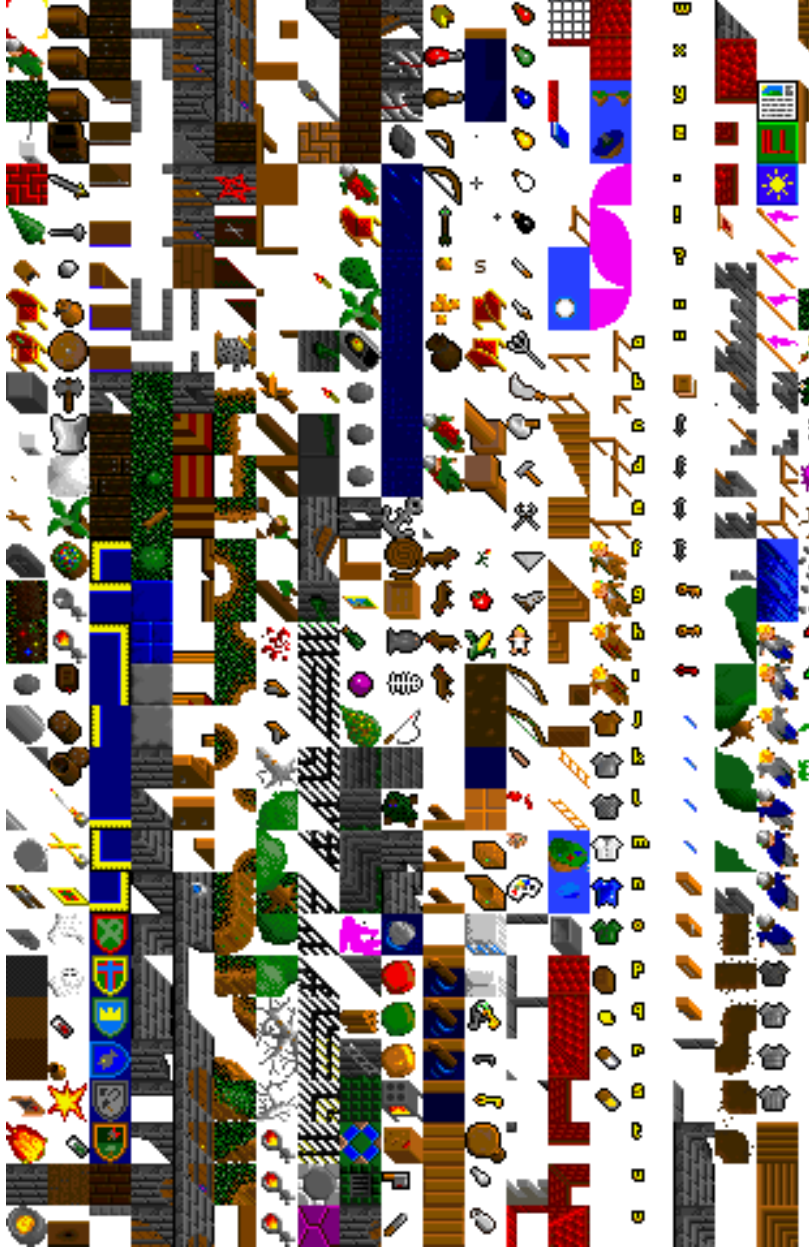
- instancing, which can be utilized by creating **a vertex buffer containing the instance geometry** and an additional **vertex buffer** with the **per-instance data**. By using instancing, we're able to completely eliminate our redundant quad vertices (and index buffer) **at the cost of an additional but smaller buffer that holds only the per-instance data.** This buffer is directly hooked up to the **vertex shader.**
- whenever we are rendering a lot more than 100 instances (which is quite common) we **will eventually hit a limit on the amount of uniform data we can send to the shaders.** One alternative option is known as **instanced arrays**. Instanced arrays are defined as a **vertex attribute** (allowing us to store much more data) that are **updated per instance instead of per vertex**.

Texture atlas..

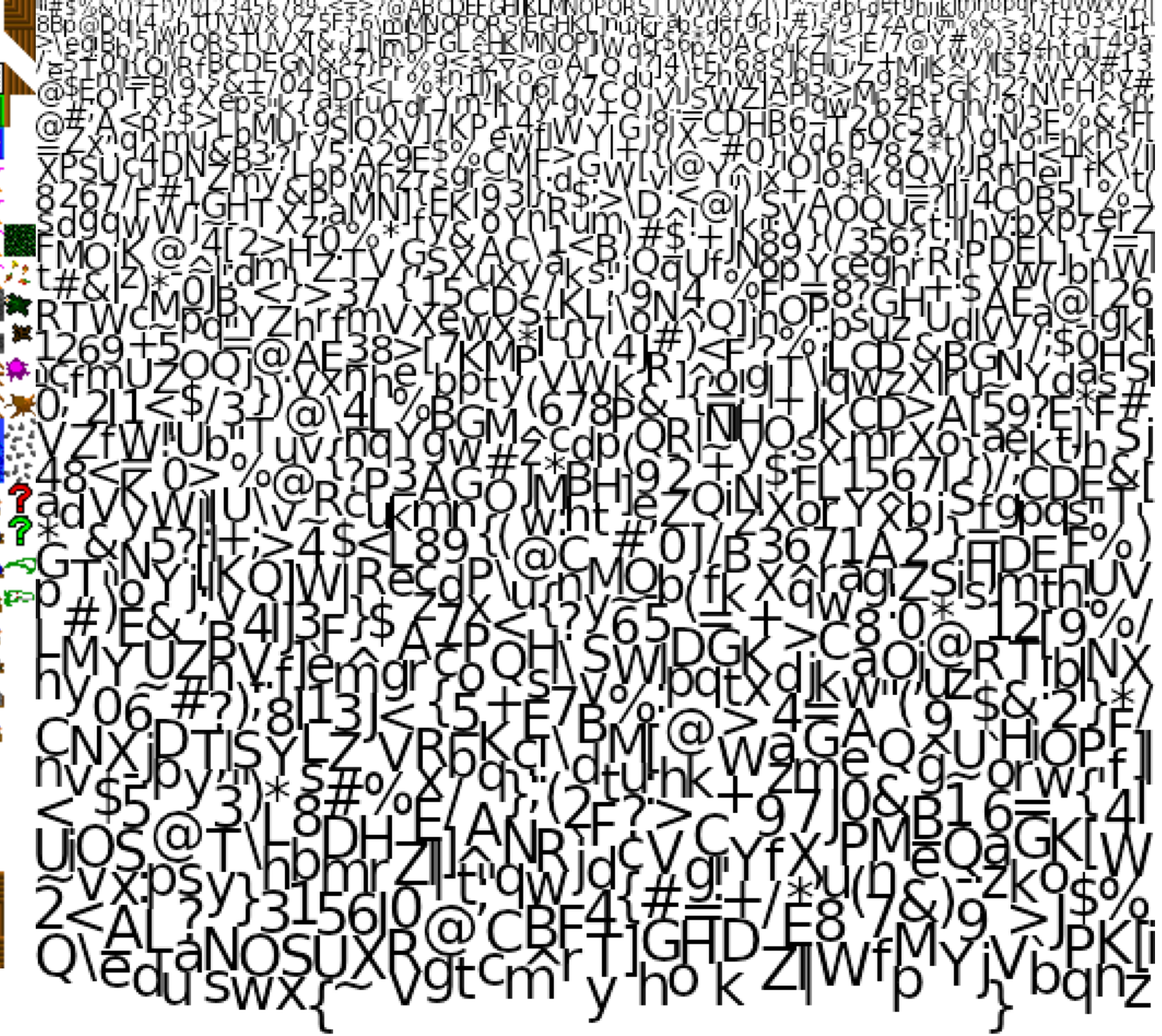
- In an application where many small textures are used frequently, it is often more efficient to store the textures in a **texture atlas which is treated as a single unit by the graphics hardware**. This reduces both the disk I/O overhead and the overhead of a context switch by increasing memory locality.
- An atlas is a texture that contains multiple smaller textures, known as sprites. **By using an atlas, game developers can reduce draw calls and memory usage, leading to better performance and faster loading times.** In Unity, an atlas can be created using the Sprite Atlas tool.

Texture atlas..

- Another benefit of using atlases is that they can help to reduce memory usage. **By packing multiple sprites into a single texture, you can avoid wasting memory on unused space.** This can be particularly useful for **mobile devices**, where memory is often limited.



A texture atlas for a video game



A texture atlas of glyphs

What is glyph?

- A glyph is any kind of purposeful mark. In typography, a glyph is "the specific shape, design, or representation of a character". It is a particular graphical representation, in a particular typeface, of an element of written language. A grapheme, or part of a grapheme (such as a diacritic), or sometimes several graphemes in combination (a composed glyph) can be represented by a glyph.

Various glyphs representing the lower case letter “a”

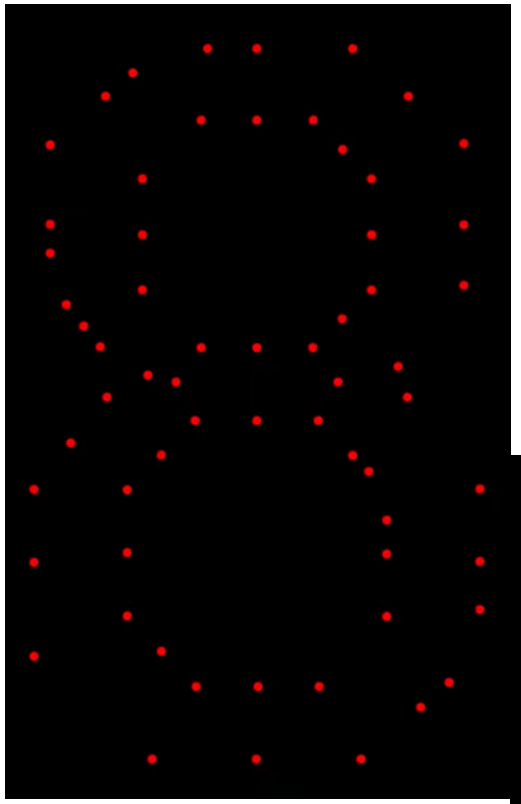
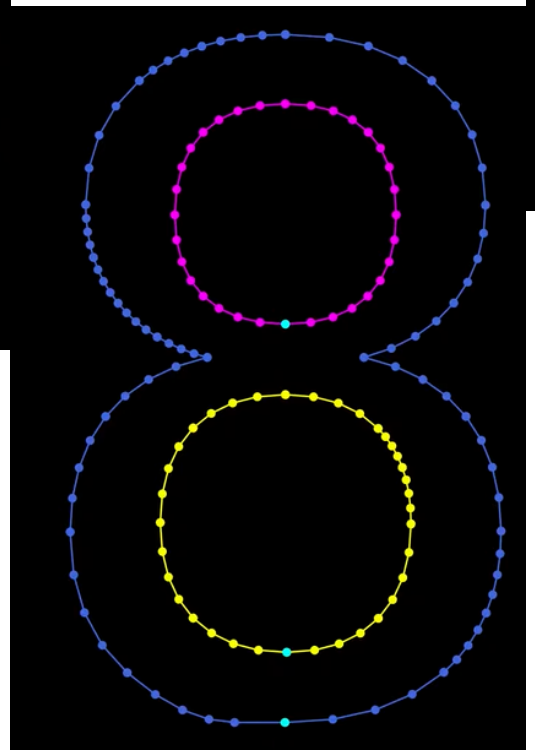
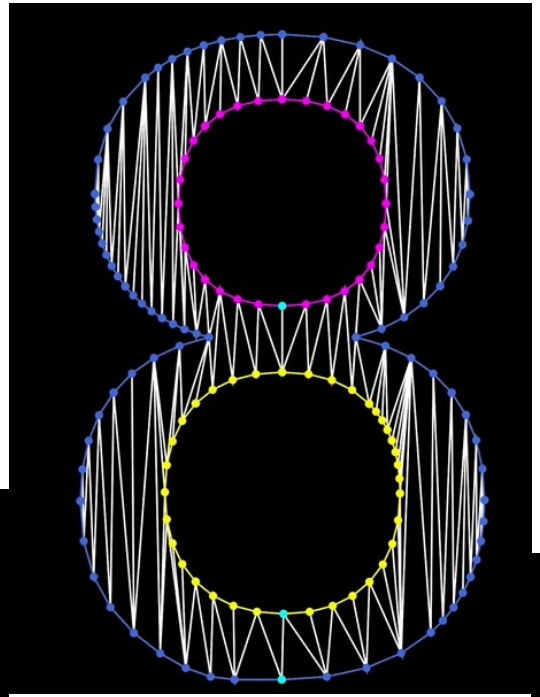
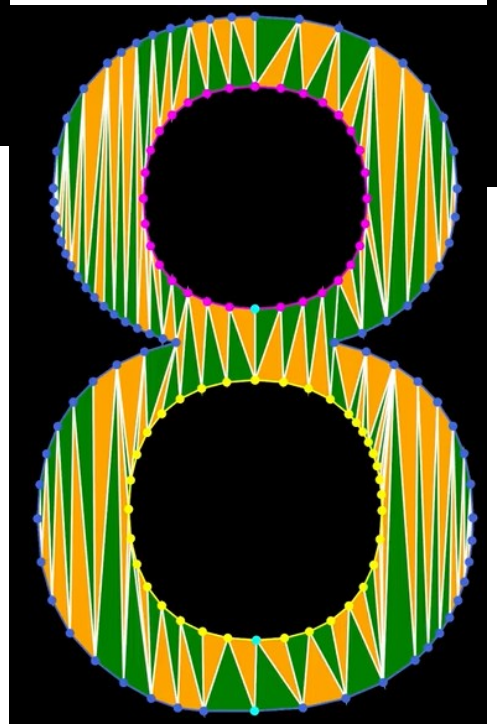
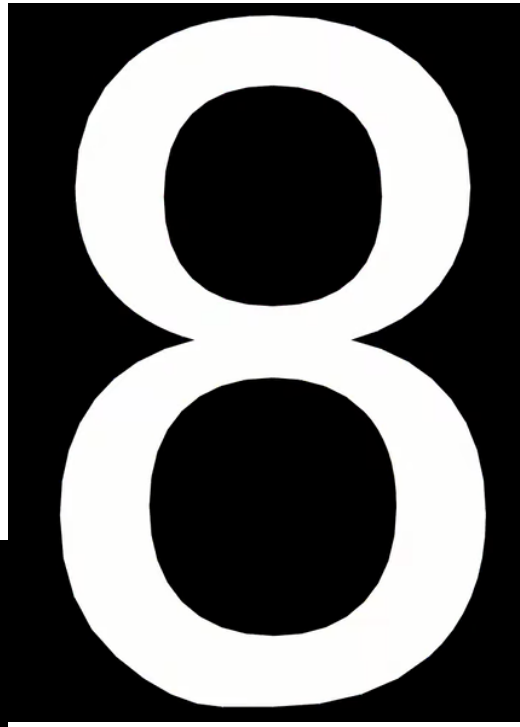
Font rendering..

- The most common font format is the vector-based TrueType format. This format represents font glyphs (in other words, alphabetic characters and other symbols) as vector data, specifically, **quadratic Bezier curves** and line segments.
- As a result, TrueType fonts are compact, easy to author, and scale well with different display resolutions. The downside of a vector font, however, is that **it is not straightforward to directly render this type of data on graphics hardware**. There are, however, a few different ways to map the vector representation to a form that graphics hardware can render.

Font rendering..

- One way is to generate geometry directly from the vector curves. However, **while modern GPUs are quite efficient at rendering large numbers of triangles,** the number of polygons generated from converting a large number of complex vector curves to a triangle mesh could number in the tens of thousands.
- This increase in triangle throughput can greatly decrease application performance.

Vector vs. raster graphics..

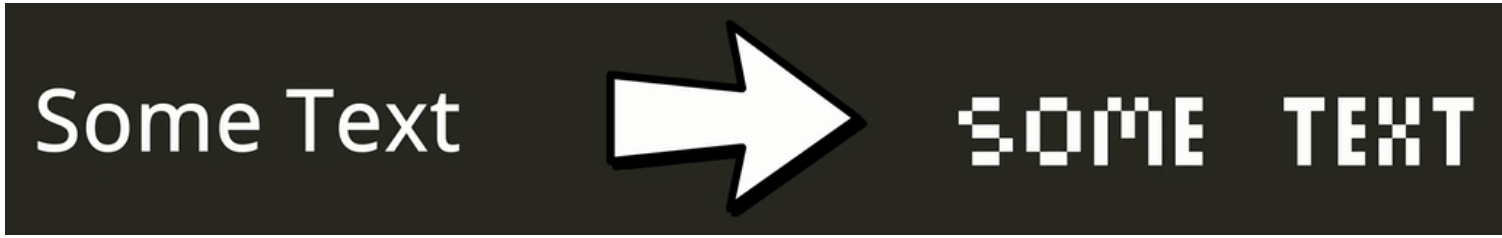
Font rendering..

Because of these limitations, the most common approach relies on **rasterizing vector graphics into a bitmap and displaying each glyph as a rectangle composed of two triangles** (from here on referred to as a quad).

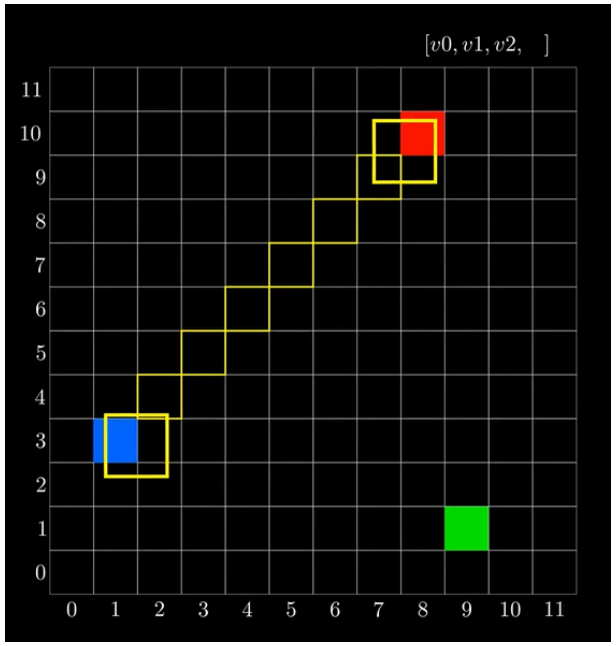
The most obvious **drawback is the resolution dependence** caused by the font page being rasterized at a predefined resolution, which leads to distortion when rendering a font at a non-native resolution.

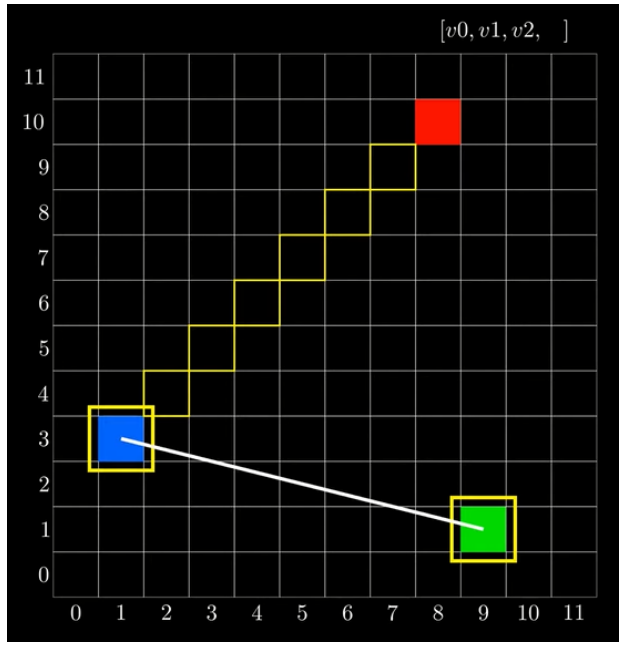
Overall, the benefits of the raster approach outweigh the drawbacks, because rendering **bitmap fonts is incredibly easy and efficient**.

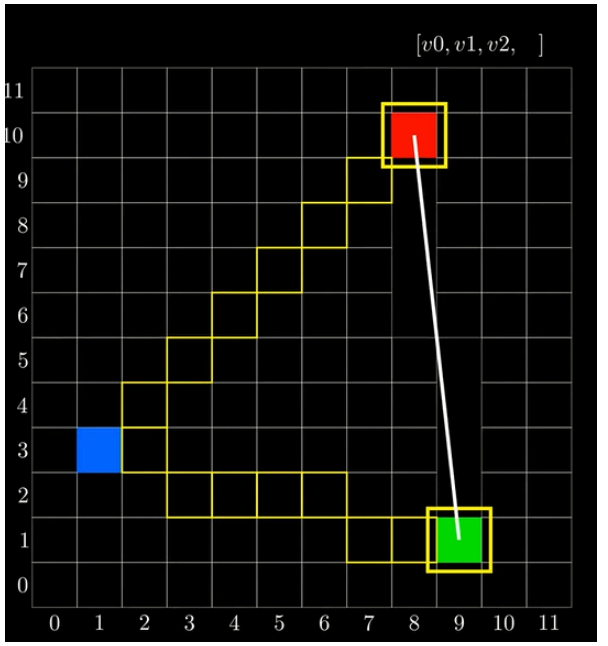
Font rasterization.

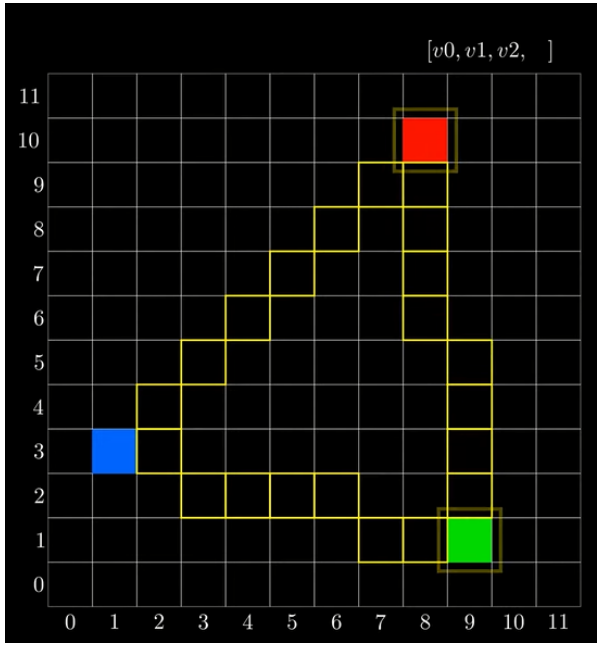


- Font rasterization is the process of converting text from a vector description (as found in scalable fonts such as TrueType fonts) to a raster or bitmap description.

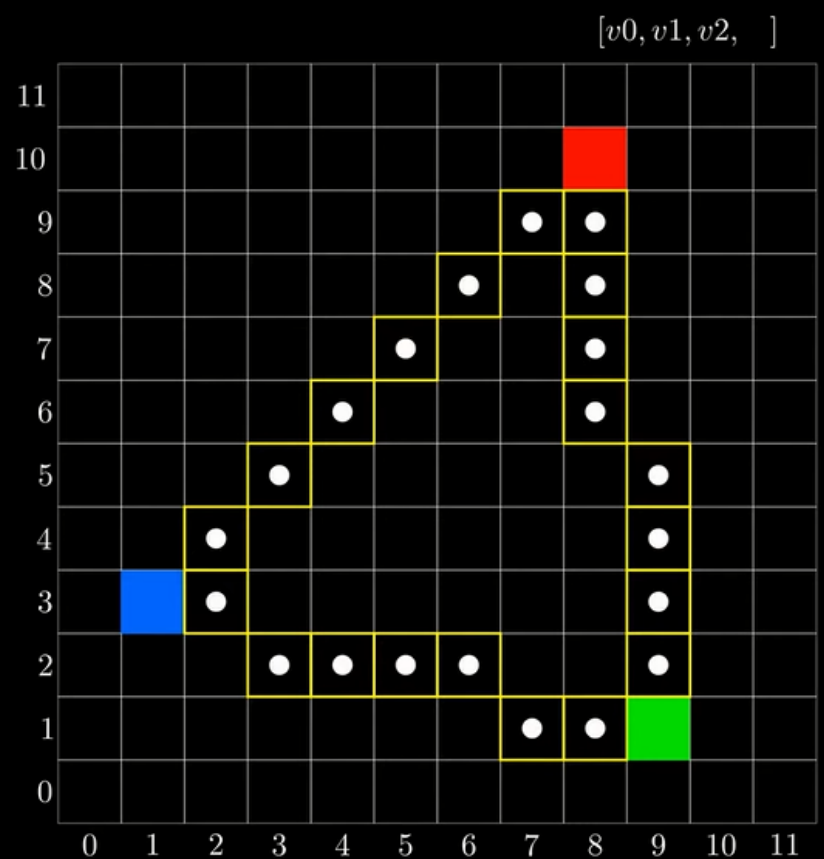








Save each position
as its new vertex



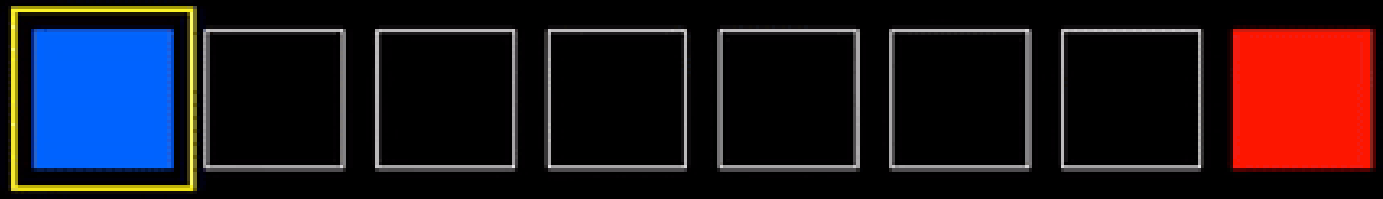
Interpolation

$[v0, v1, v2, ...]$

Linear Interpolation

$$\frac{7}{7} * \begin{bmatrix} 0 \\ 102 \\ 255 \end{bmatrix} + \frac{0}{7} * \begin{bmatrix} 255 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 102 \\ 255 \end{bmatrix}$$

0%
100%



Font rasterization.

- Rasterizing vector graphics into a bitmap and displaying each glyph as a rectangle composed of two triangles (from here on referred to as a quad), as shown in Figure 1.1.2. A font texture page is generated with an additional UV offset table that maps glyphs to a location in that texture very similar to how a texture atlas is used.

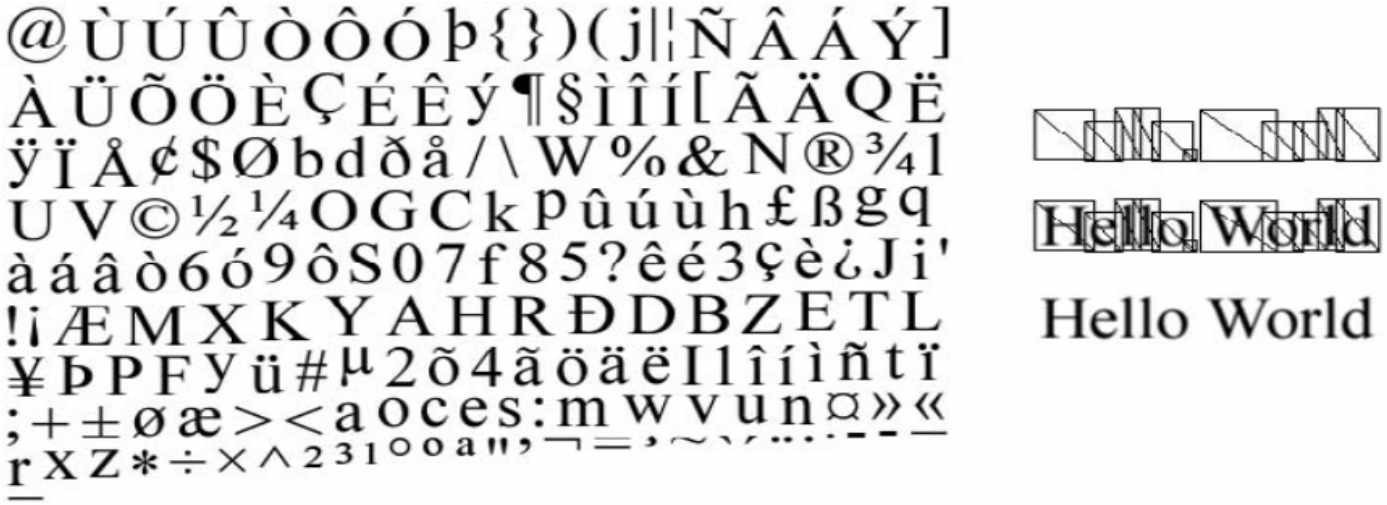
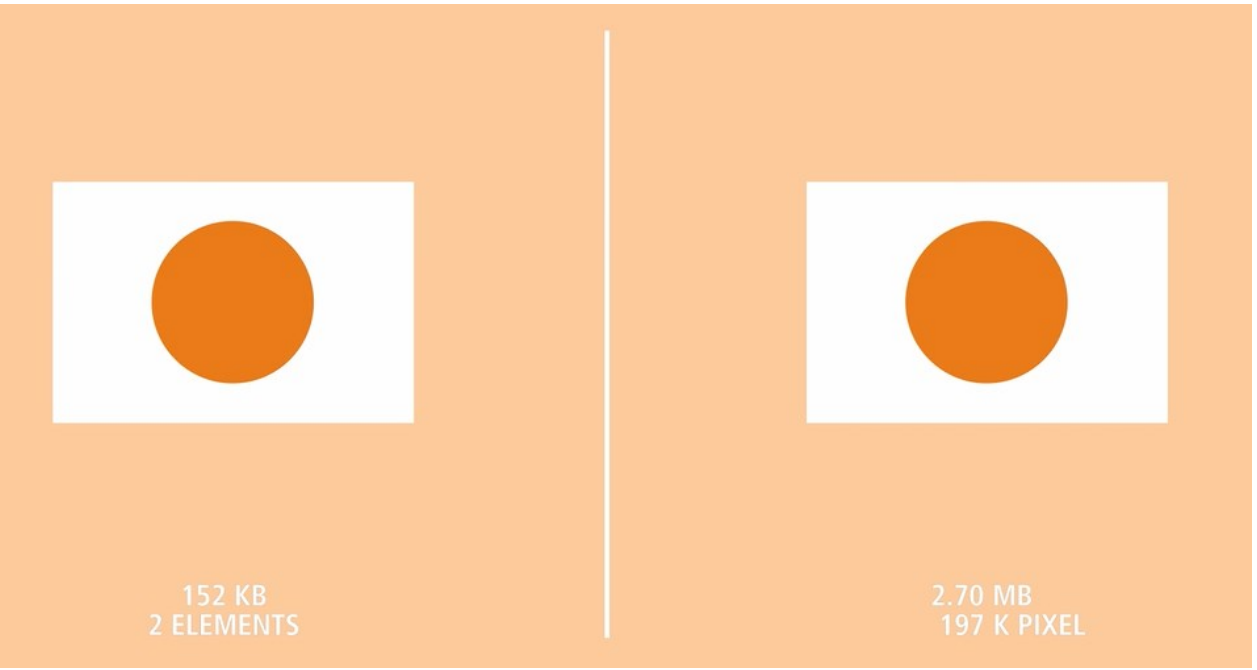


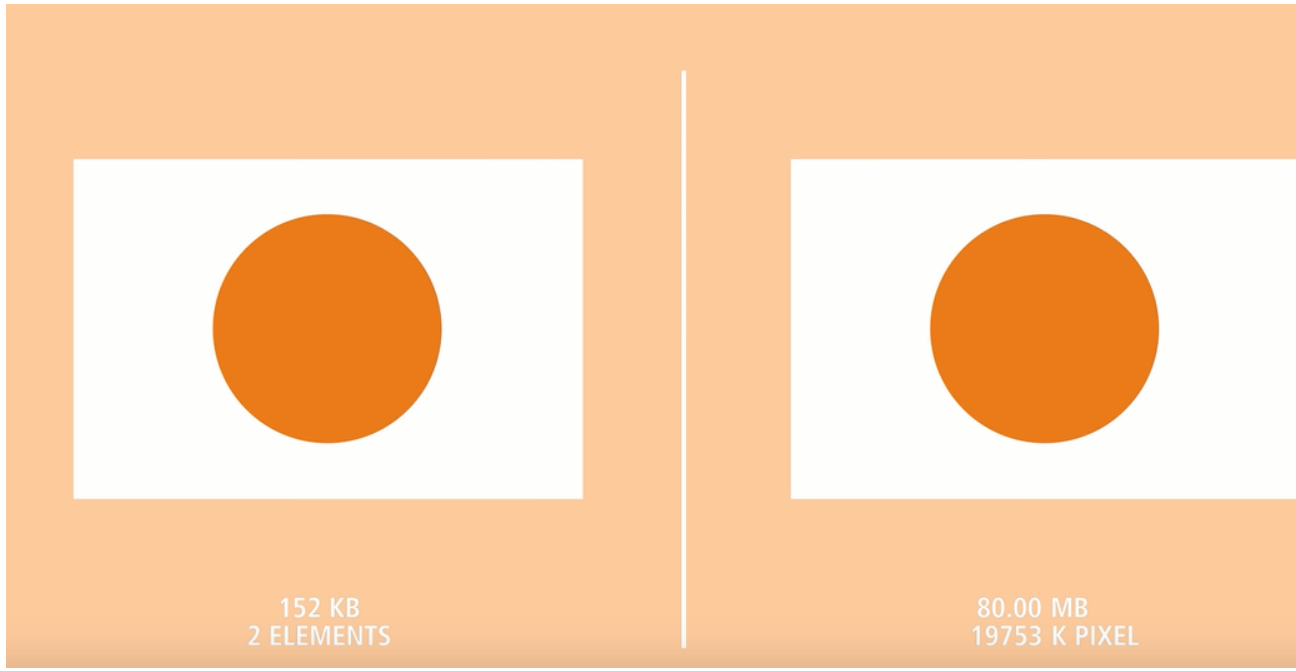
Figure 1.1.2 A font page and a glyph rendered on a quad.

Vector vs. raster graphics..

- **Raster graphics** made out of millions of colored picture points known as pixels.
- **Vector graphics** are made up of points which are defined by coordinate and mathematical formula surpass these incur points can be connected, rounded and colored in.
- Because raster graphics have limited amount of pixels they become blurry when scaled up in size , vector graphics are mathematically calculated and scaling won’t affect its quality.
- However when we scale both down , they both look good, there is also difference in file size

Vector vs. raster graphics..





vector

raster