Computer Graphics

Dr./ Ahmed Mohamed Rabie

Chapter 5

Mobile Graphics

In the last few years we have seen a dramatic growth in both the computation and connection capabilities of mobile computing devices. Today, all this power can be packed into a device so compact that it fits in your pocket. In fact you can go out your front door carrying your favorites images, videos, and data files, working wherever you like

Not only has the computation and communication power of mobile devices and handhelds been increased, but the visualization and graphics capabilities have also risen. You can now take a picture or capture a video sequence with your cellular phone or personal digital assistant. Moreover, you can view as many colors as on a graphic workstation and interact with color graphics applications in real time.

Moreover, you can view as many colors as on a graphic workstation and interact with color graphics applications in real time. Even considering these major advancements in mobile devices, there are still unresolved problems. Manufacturers can integrate all the multimedia content they want, but if the battery runs out and users can't, for example, make a brief phone call, the tool will not be effective. The challenge is to integrate all this video and multimedia technology without sacrificing the battery life of the product or weighing it down with expensive batteries.

There have been advancements in microprocessor management of battery power consumption and the graphics capabilities associated with it, but with these enhancements the desktop computer continues to be the leader in providing multimedia experiences to end users. However, this may change very soon.

Even considering the new graphics features, mobile devices are still less capable than desktop computers. In fact, programs run at lower speed, there is less memory for processing and storing programs, the displays are smaller in size and color numbers, and the battery could run out. The most important reason behind these differences is power: one device is plugged in to the power grid, while the other is dependent on battery power.

Another relevant issue to deal with is the usability of mobile devices. In fact, manufacturers have reduced the scale of devices, and are addressing the difficulties in using handhelds. Many of them have problems with inputting data (keyboards are too small or pen-based systems require the user to learn a specific language), or provide miniaturized displays and very small visual interfaces. **The following problems can be distinguished:**

- storage and data size limitations.
- communications and graphics interfaces usability problems.

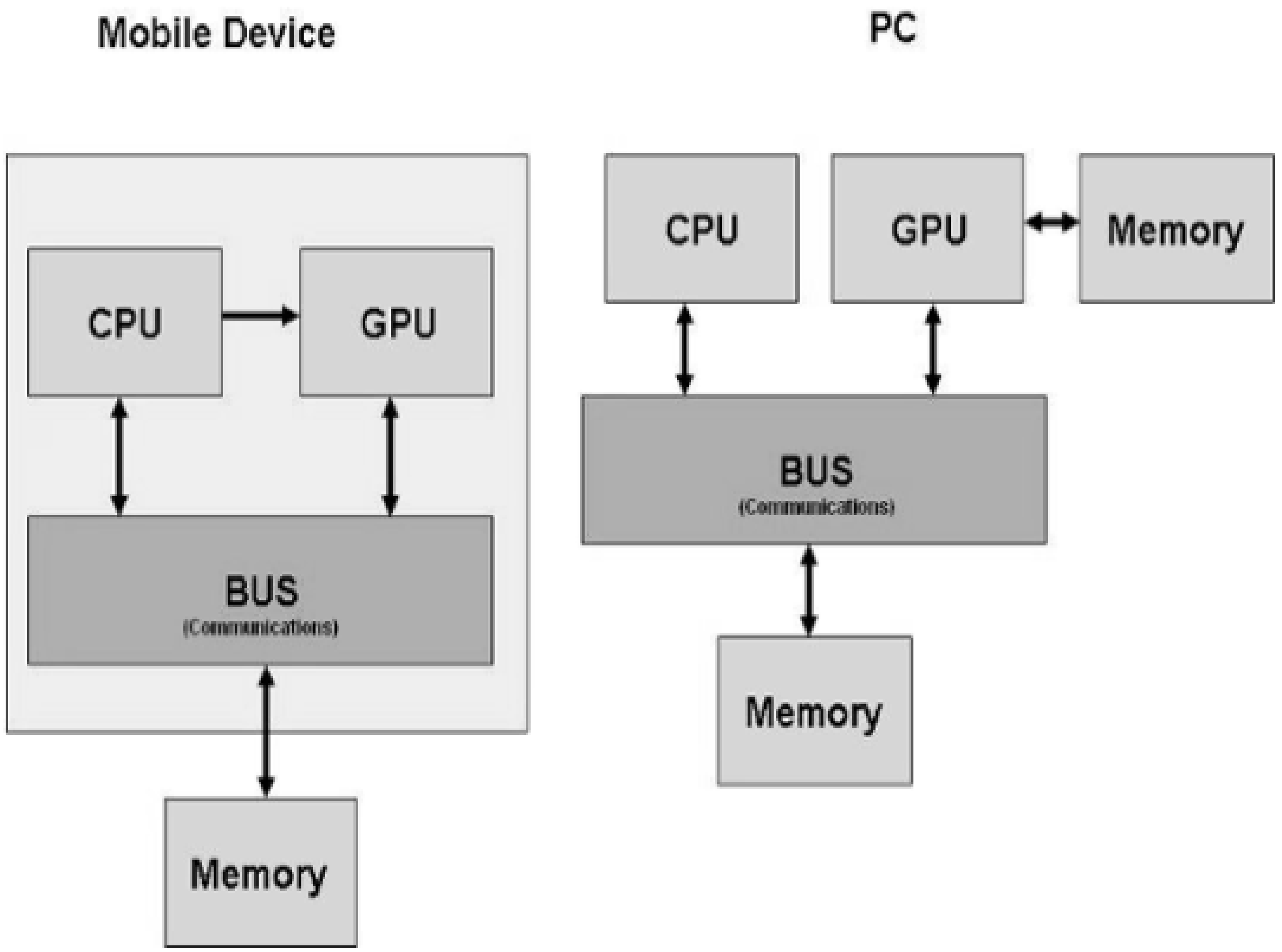
Until now, all the graphics representations in these services were made of 2D maps with text labels. With the introduction of 3D graphics capabilities in mobile devices, many researchers found that using 3D modeling with the mobile guides allows user interfaces to be more intuitive and friendly for these kind of applications. Many usability studies have shown that 3D environments help in understanding real spatial relationships, and thus could help users in navigating both the interfaces and the environment. In fact, by using a 3D representation, the viewer's perception of the environment improves, since users can better understand distances and proportions between objects visualized by the mobile guide and the real environment.

The ideal mobile device will need to support at least a standard OS and standard 3D graphics API. Moreover, it should provide features for playing music, movies, and games. It will need to include effective input devices and support for GPS positioning, not only for localization such as mobile guides applications, but also for multiplayer GPS games where the real user location is fused with game rules, providing augmented reality.

This device should also support standard wireless network connections and a small video camera for interacting or recording. Finally, basic mobile phone capabilities should be required, especially SMS facilities both for communicating and supporting micropayments for applications downloading.

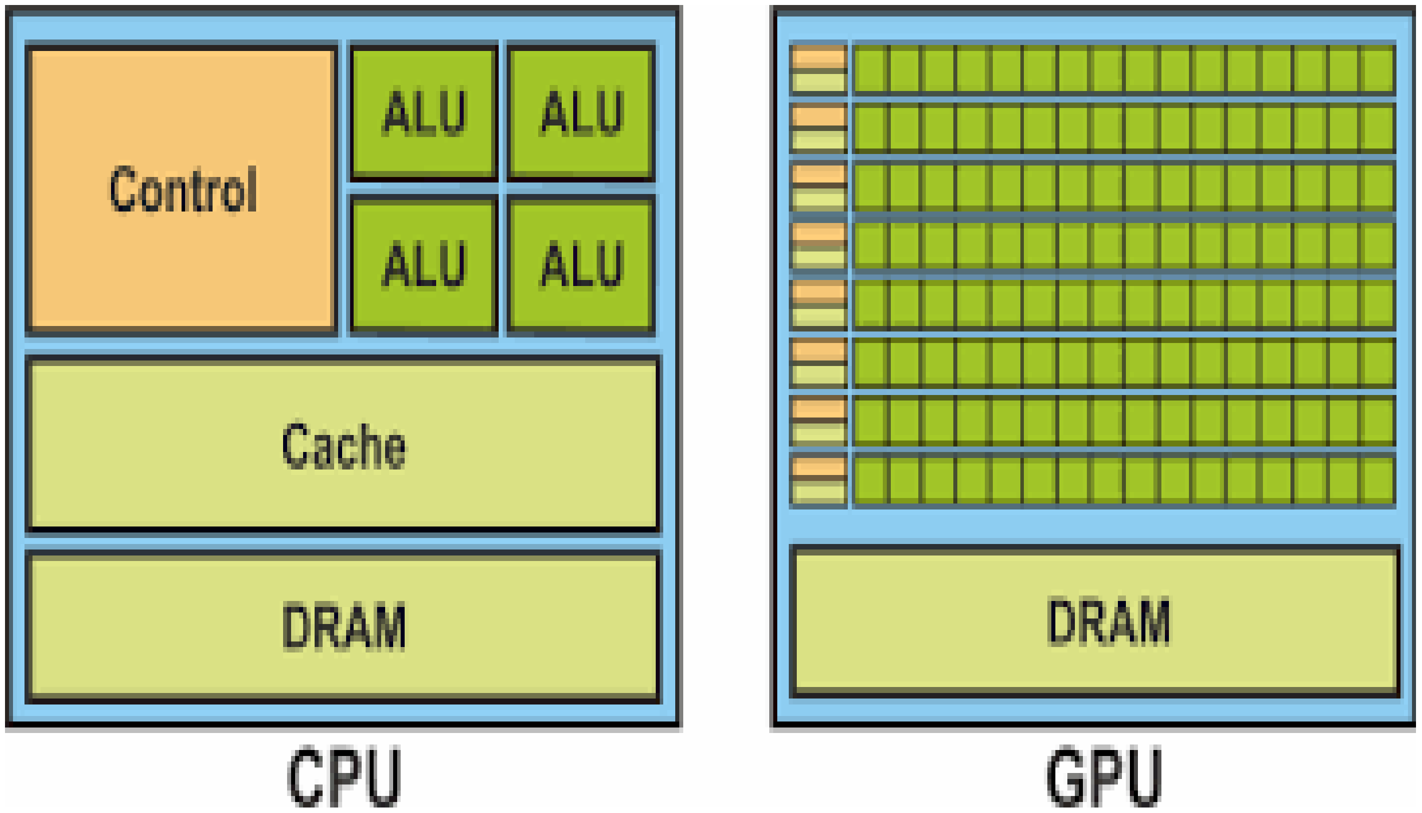
In fact, mobile devices don’t have processors able to process complex mathematical computations; they can’t even store a large amount of data (e.g., required for storing textures). Their screens are too small, with low resolution (screens are rapidly improving), and the batteries still tend to run down very fast when dealing with 3D computer graphics (usually this is related to the large amount of computations required).

The mobile device architecture is very different from desktop computers. It includes only one main memory unit, and the central processing unit (CPU) with the graphic processing unit (GPU) are implemented on the same chip.



Mobile device VS Desktop PC architecture.

GPUs of mobile terminals have evolved considerably. However, despite being claimed the contrary, mobile GPUs are still not on par with consoles and/or desktop GPUs, mostly because of energy considerations. This makes the use of large scenes or datasets quite difficult. As a result, there is a need in the implementation of specialized algorithms that are able to deal with large models


Control
ALU
ALU
ALU
ALU
Cache
DRAM
CPU
DRAM
GPU

The main issues concerning the mobile devices architecture are as follows:

1. Limited memory bandwidth. Exchanging data from and to the memory is very time and power consuming.
2. Mobile device processor units don’t include a floating point unit (FPU).

Floating-point operations could be also emulated by software, but using many floating-point operations, if supported by emulation, heavily reduces the performance. Where available, it's better to choose fixed-point math. For example, to reduce pixel data exchange (thus bandwidth), smaller textures could be used, or to reduce pixel transformation, smaller rendering windows are a good solution.

GPUs have multiple processing cores that run simultaneously at low speeds. They have two functions-processing of vertices and pixels.

- **Vertices**: The process of vertices revolves around the coordinate systems. The GPU handles the geometric calculation for reproducing dimensional space on your phone's display screen. It results in things like depth, spatial data and the possibility of 3-dimensional space rotation.

- **Pixels**: It's a complex process and requires a lot of processing power. It renders various layers of the images and applies effects. They are required to develop complex textures for creating realistic graphics.

Lastly Once GPU handles both these processes, it transfers the result to the digital readout. (Phone's screen).

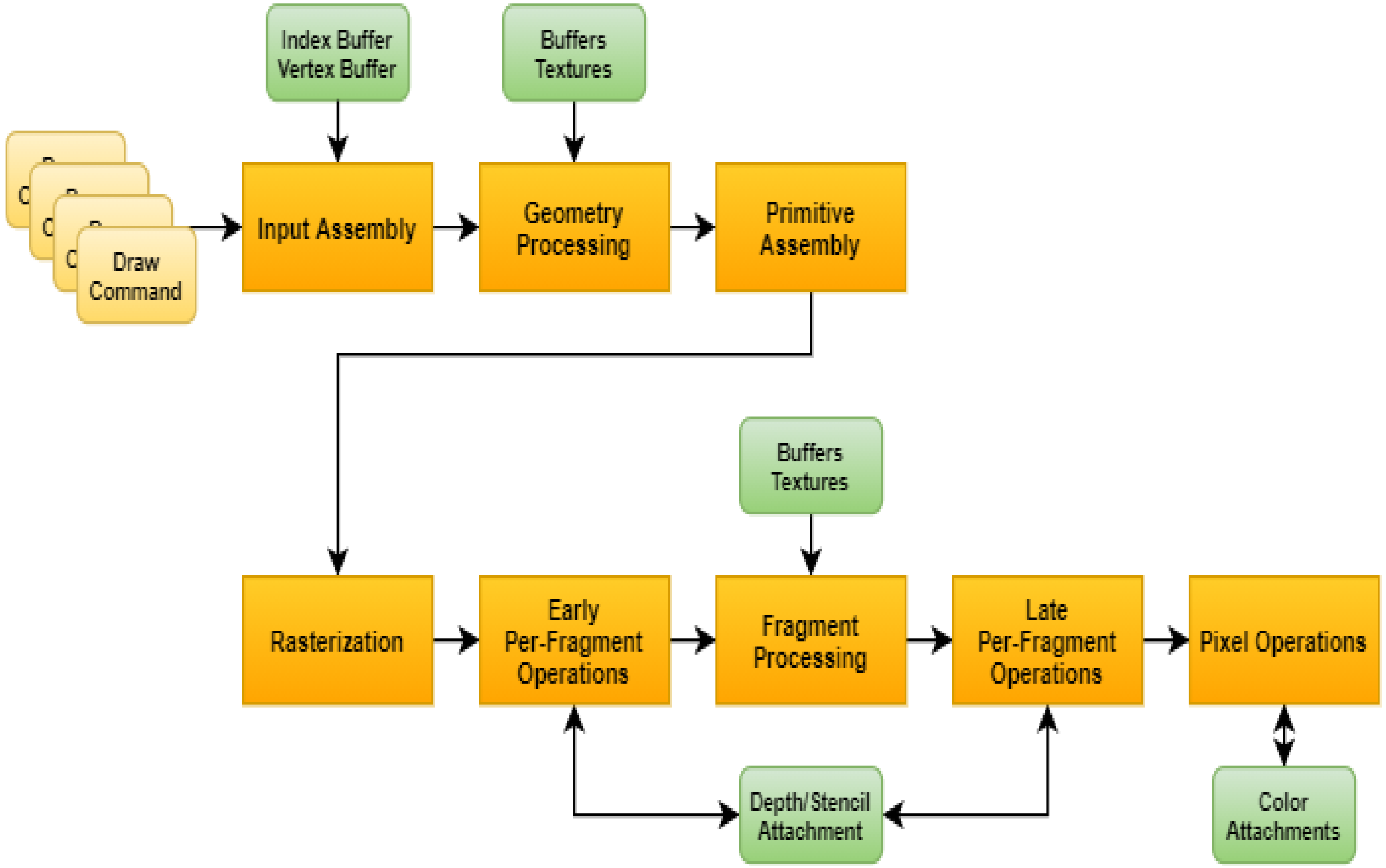
OpenGL® ES is a multiplatform set of API and it's also royalty-free; it includes a complete set of functions for 2D and 3D graphics on embedded systems like mobile devices, vehicles, and appliances. It is a subset of the well-known API set called OpenGL ®, and it operates as an interface between the low-level software layer and graphics devices.

GPU Architecture Types

The behavior of the graphics pipeline is practically standard across platforms and APIs, yet **GPU vendors come up with unique solutions to accelerate it**, the two major architecture types being tile-based and immediate-mode rendering GPUs. Their strengths/weaknesses, and discuss some of the implications the underlying GPU architecture may have on the efficiency of certain rendering algorithms.

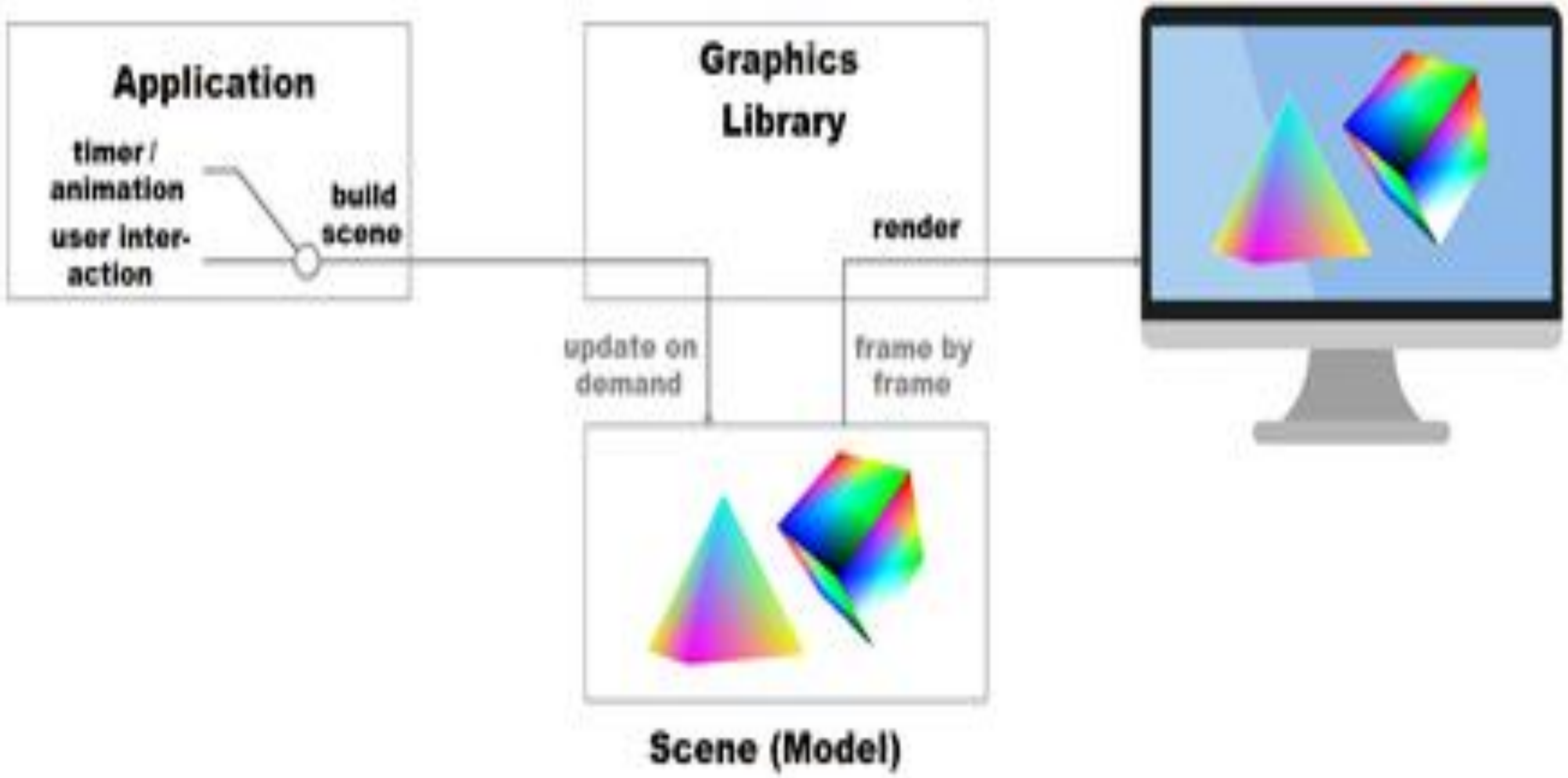
1- Immediate-Mode Rendering: We can really call this GPU architecture the "traditional" one, as immediate-mode rendering **(IMR)** GPUs implement the logical graphics pipeline, as described by the various graphics APIs, quite literally:

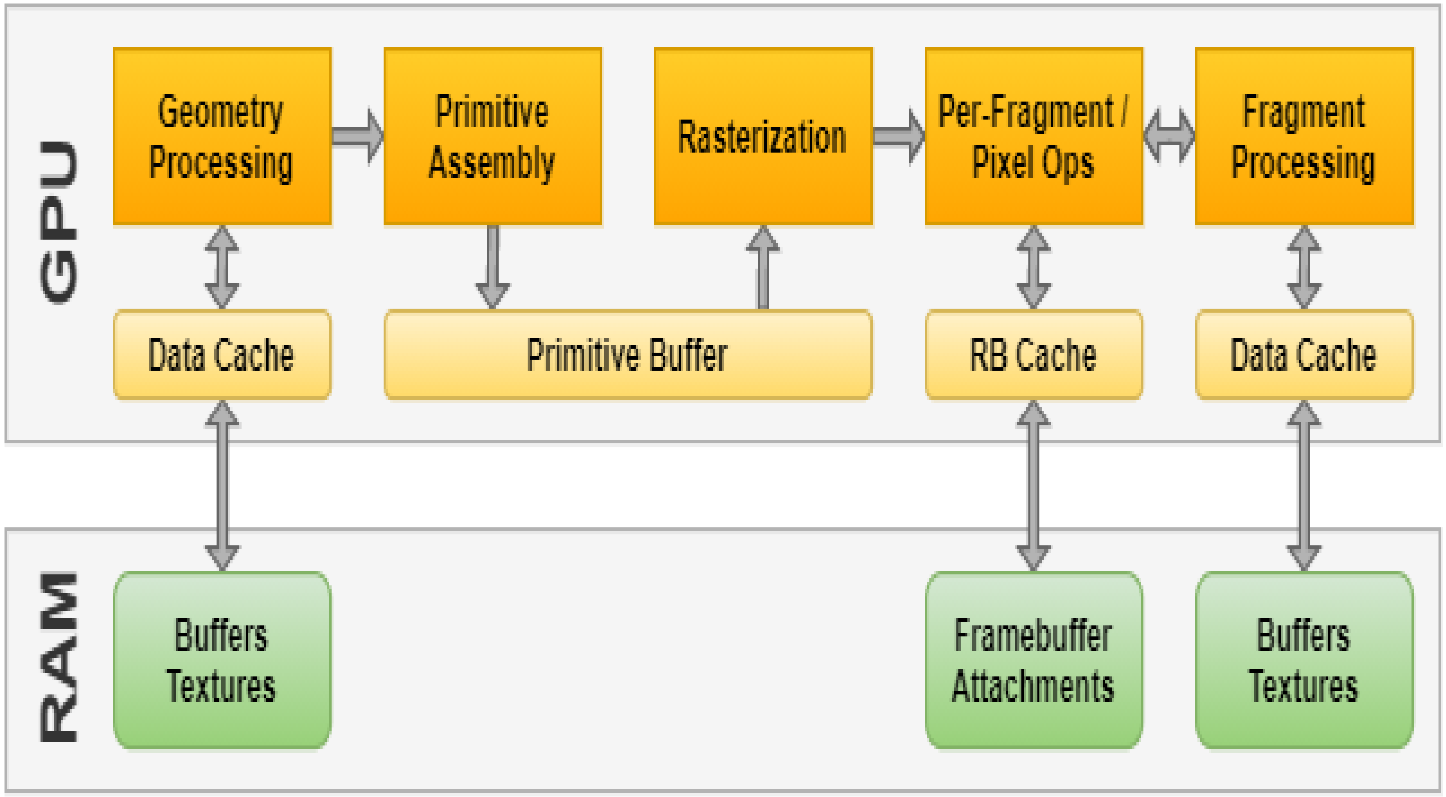
- Incoming draws trigger the generation of geometry workload with a corresponding set of vertices to be processed (with appropriate primitive connectivity information according to the primitive type).
- Vertices/primitives are then fed to the various geometry processing stages (vertex shading and any other additional processing stages like tessellation or geometry shading, if enabled, or mesh shading in latest architectures).



Simplified illustration of the immediate-mode rendering pipeline.

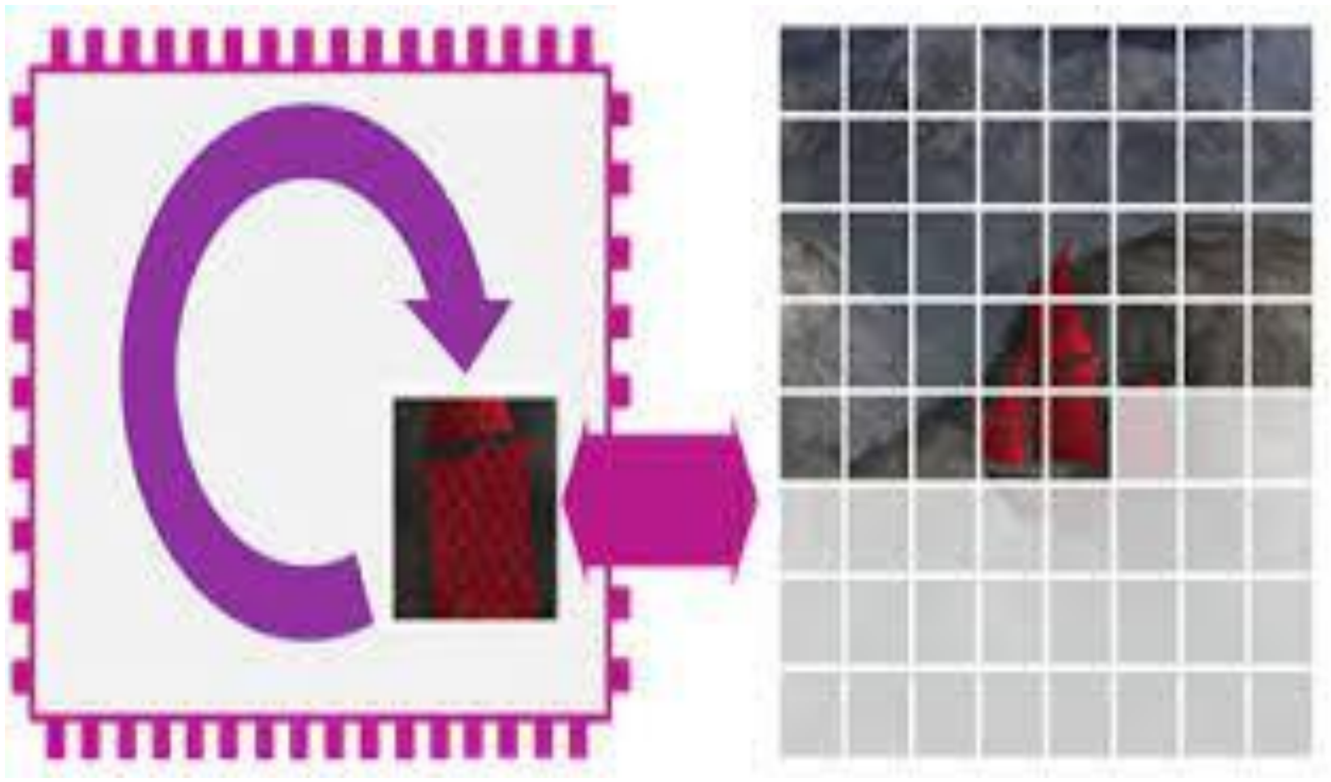
- The resulting primitives are then culled (and potentially clipped), transformed to framebuffer space, and sent to the rasterizer
- Fragments generated by the rasterizer then go through the various per-fragment operations and fragment processing (potentially discarded by the fragment shader)
- Finally the remaining fragment's color values get written to the corresponding color attachments (potentially in non-trivial ways in case of multisampling, as an example)

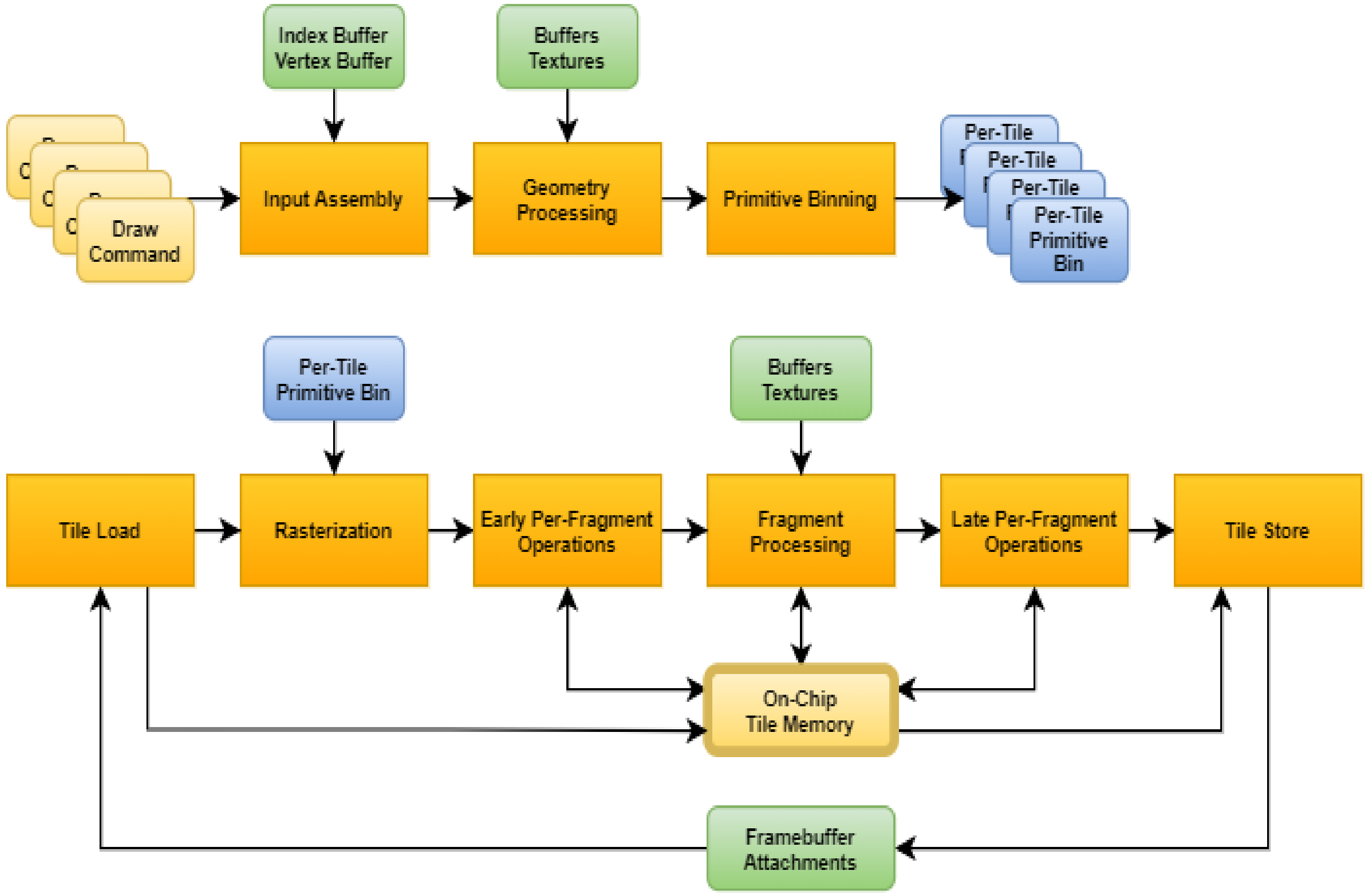

Application
timer / animation
user inter-action
build scene
Graphics Library
render
update on demand
frame by frame
Scene (Model)



Key data paths of immediate-mode GPUs.

2- Tile-Based Rendering: As the name suggests, tile-based rendering (TBR) GPUs execute the graphics pipeline on a per-tile basis. What this means is that the framebuffer space (also referred to as the render area) is split into equisized rectangular areas called tiles, and rasterization, as well as all other back-end stages, are executed separately for each individual tile after the front-end stages completed.

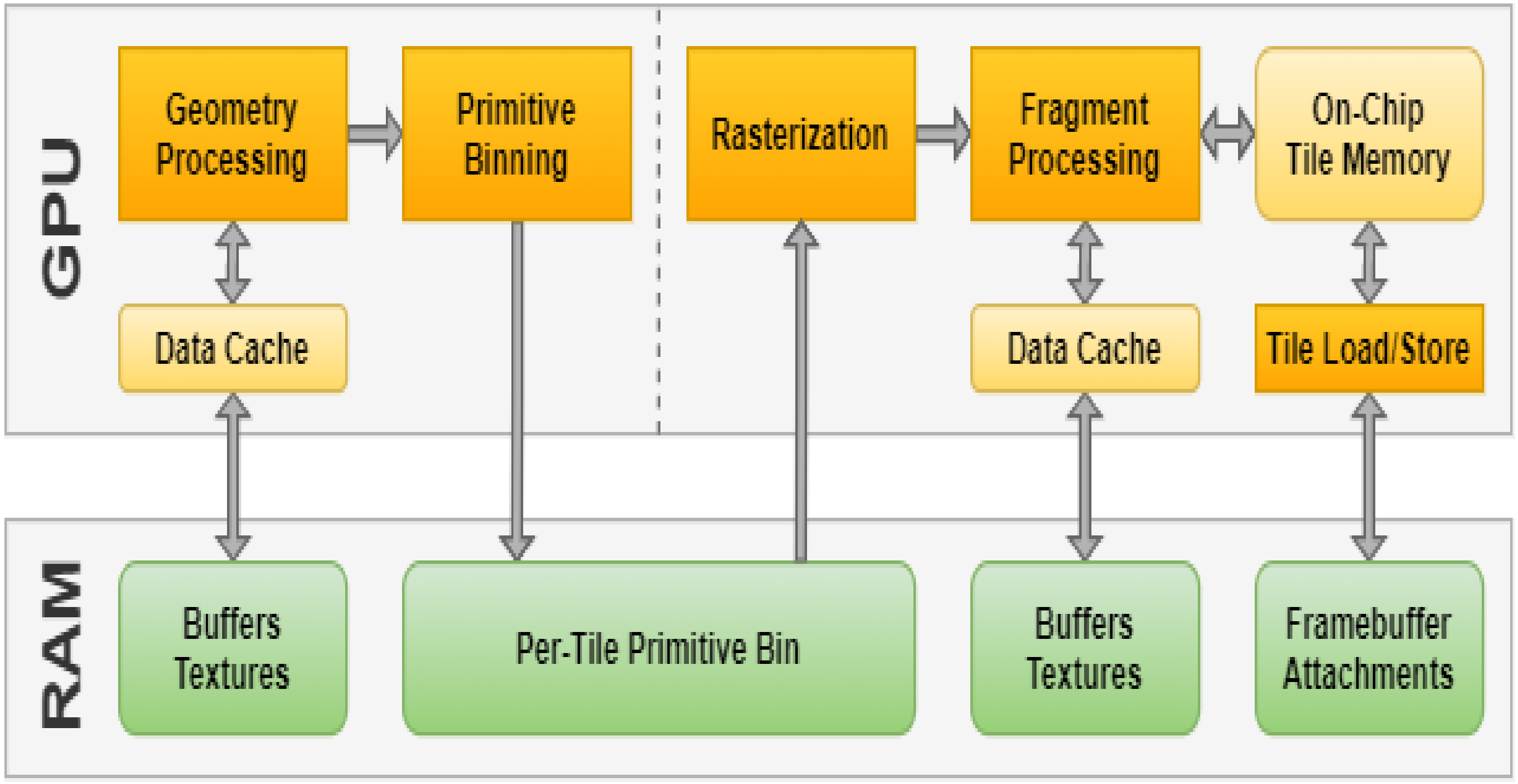




Simplified illustration of the tile-based rendering pipeline.

The traditional primitive assembly stage is replaced with a primitive binning stage whereas culled primitives are accumulated into one or more bins, depending on which tile within the render area do they overlap with. After the front-end stages complete, a separate per-tile pipeline is launched that:

- Starts with the tile load operation responsible to load framebuffer attachment data corresponding to the tile into dedicated on-chip storage.



Key data paths of tile-based GPUs.

- Then primitives in the bin corresponding to the tile are rasterized and go through the usual per-fragment operation and fragment processing stages, but when those need to read/write framebuffer data they access the on-chip storage instead of the in-memory attachment resources.
- Finally the tile store operation is responsible to store back any modified framebuffer attachment data from the on-chip storage to memory.

It can be seen that the key **difference** between a **TBR and an IMR GPU** is the way they communicate primitive data between the front-end and back-end stages and the way they access framebuffer data. These dissimilarities have profound implications though.

Mobile Visualization

Data visualization is the graphic representation of data. It involves creating images that contextualize information and help see the relationships among the represented data. By using graphical entities like charts, graphs, and maps, data visualization tools provide a convenient way to see and understand trends in data, outliers, and even tell a story.

Data visualization for mobile is not a new concept but it's far from losing its relevance, especially in the world of marketing. After all, more than relying on quality information, it's necessary to ensure the right conditions for its analysis. That is the reason we use graphics, animations, and other visual resources to facilitate the process.

Mobile visualization system designs have to account for lots of limitations, most notably, screen space and user interaction methods. Moreover, mobile devices come in a variety of screen sizes and resolutions and hence the visualization should be scalable across various devices. While all visualization should display relevant data, the limited display resolution of mobile devices makes the appropriate visual representation very crucial.

We have to avoid cluttering the screen, but at the same time display all the essential information in an aesthetically pleasing manner. Hence we need to first extract relevant data at an appropriate level of abstraction, and then match the data to an appropriate visual representation. In the case of network data during a football game, we must also take into account the time variation, and provide synchronized visual display.

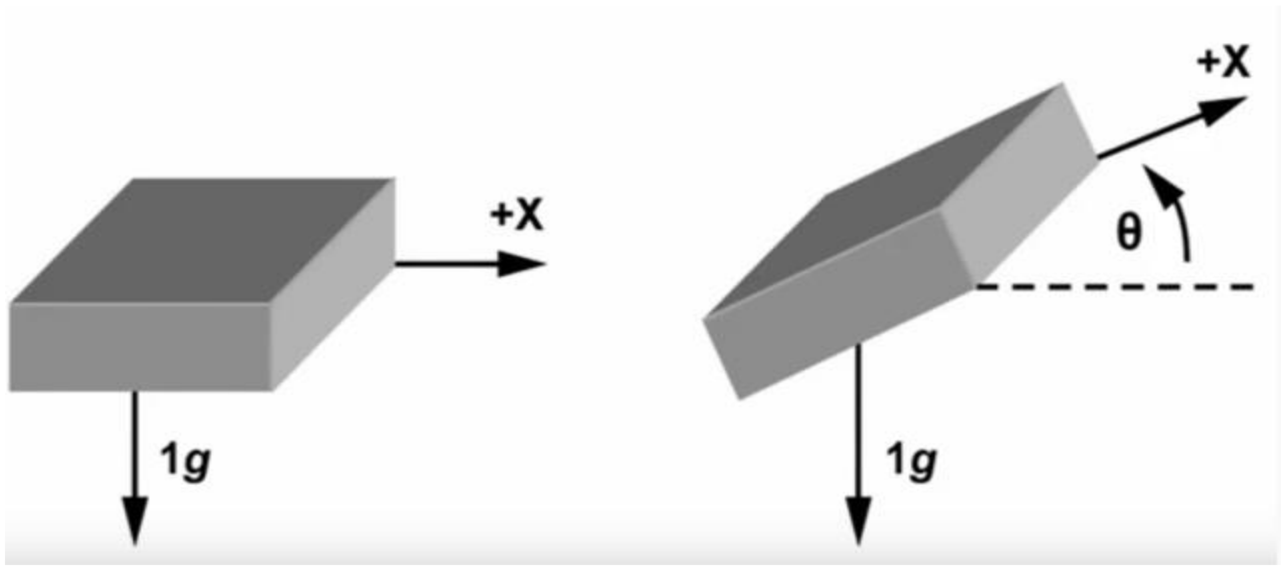
Designing Graphics for Mobile Devices

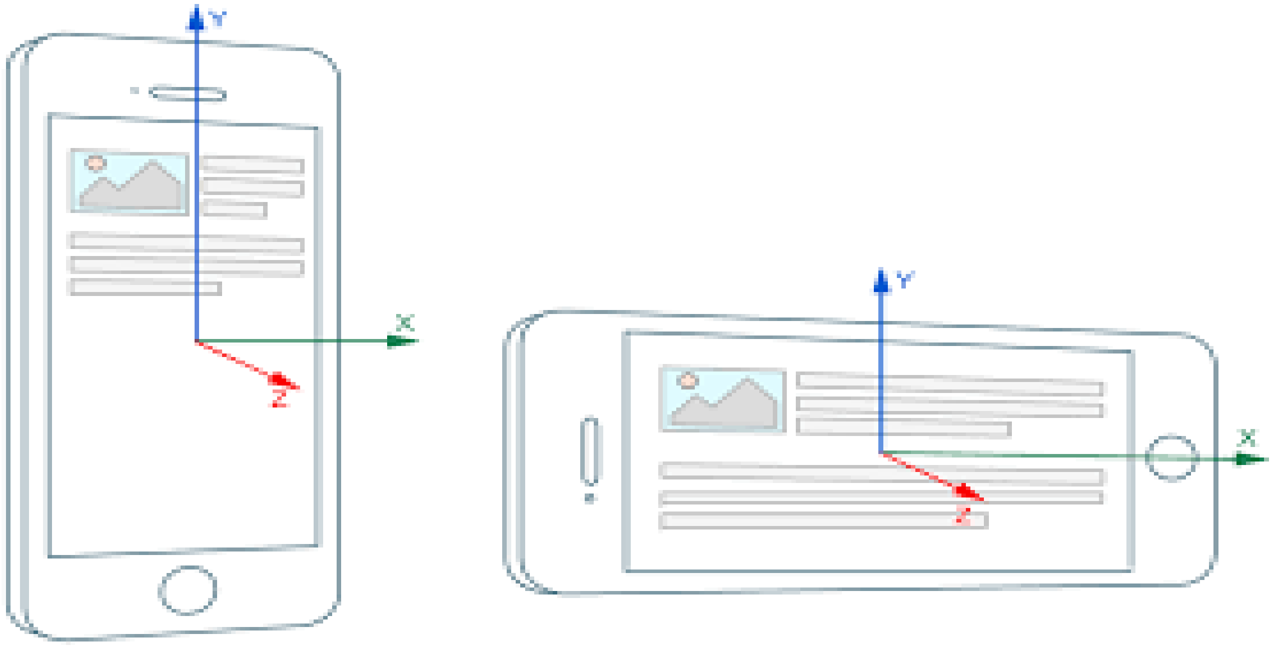
Mobile devices are becoming the most preferred platforms people are using to access the internet. Whether it is for chatting, browsing, connecting with friends and other social interactions, research shows that the number of users with mobile devices outnumbers those using desktop devices. This has presented businesses and websites owners who provide content on the internet or those who wish to use the internet to gain visibility to come up with mobile friendly designs. Consequently, designers working on designs to show on the internet are now compelled to adapt their designs for proper rendition on the mobile devices. If you are a designer working on graphics for display on mobile devices

here are a few points to get you going on the right path.

- **Resolutions and screen size:** The various mobile devices for which you are adapting your designs have different screen sizes and resolutions. It implies that you can’t create graphics with the same resolution and expect it to render well across all the devices. This will make the design not to appear in the right manner if it is displayed on devices that are not compatible with the resolutions you had designed it for. Therefore, check the display screens as well as the pixel densities and make designs that will render without any resolution problems on the mobile devices.

- **flexible designs:** Most of the mobile devices, except for those manufactured by Blackberry will have a G-Sensor. A G-Sensor is a technology fitted in the mobile devices to understand the orientation of the devices and display information in the right format. It is the technology for determining whether the device is held vertically or horizontally so that it adjusts the devices' display to reflect the same, either landscape display or portrait display. As much as the mobile devices have the G-Sensors, designers working on graphics meant to mobile display should not rely on it but use appropriate designs skills that will lead to this flexibility. In graphics design, as well as web design, this calls for a fluid layout so which will allow the content to be tilted or stretched to offer a good viewing experience.


+X
1g
+X
θ
1g


Y
X
Z
Y
X
Z

- **Color :** The color is another important consideration to have in mind if you are designing graphics for display on mobile devices. Depending on the manufacturer, mobile devices will have different levels of color support. For example, Blackberry and HTC support up to 65K colors while iPhone, Nokia, and Samsung support up to 16 million colors. With color, however, the devices will adjust accordingly, with those having higher displaying better quality.

- **Image Formats:** PNG is the most preferred image format when designing graphics for display on mobile devices. It supports up to 254 partial transparency pixel levels, and this gives you the versatility you need to take advantage of alpha transparency. PNG image format has greater color depths, and such images will still perform well in instances of variable transparencies. Other image formats like GIF are pretty limiting and will always result in loss of transparency leading to poor image rendition on the mobile devices.

- **Animations**: Since most smart phones don't support Flash (at this moment) animations can be done by calling images in a sequence through programming. You can also create animations in Flash and convert that to video formats that the smart phones support.

Mobile Graphics

• OS	– Android – iOS – Windows Phone – Firefox OS, Ubuntu Phone, Tizen...
• Programming Languages	– C++ – Obj-C / Swift – Java – C# / Silverlight – Html5/JS/CSS
• Architectures	– X86 (x86_64): Intel / AMD – ARM (32/64bit): ARM + (Qualcomm, Samsung, Apple, NVIDIA,...) – MIPS (32/64 bit): Ingenics, Imagination.
• 3D APIs	– OpenGL / GL ES – D3D / ANGLE – Metal / Mantle / Vulkan (GL Next)
• Cross-development	– Qt – Marmalade / Xamarin / – Muio – Monogame / Shiva3D / Unity / UDK4 / Cocos2d-x